



# インテル® マス・カーネル・ライブラリ

---

## リファレンス・マニュアル

© 2005, Intel Corporation.  
無断での引用、転載を禁じます。

資料番号 : 630813-019J

Web サイト : <http://www.intel.com/jp/developer/software/products/> (日本語)  
<http://www.intel.com/software/products/> (英語)

改定番号	改定履歴	日付
-001	初版。	1994 年 11 月
-002	crotg と zrotg の両関数を追加。従来の版では説明されていなかった関数 ?her2k、 ?symm、?syrk、?syr2k のドキュメント・バージョン。また、ページの割り当てを変更。	1995 年 5 月
-003	タイトルを『Intel BLAS Library for the Pentium® Processor Reference Manual』から現在のものに変更。?rotm と ?rotmg の両関数を追加するとともに付録 C を更新。	1996 年 1 月
-004	並列化機能を持つインテル® マス・カーネル・ライブラリのリリース 2.0 について説明。並列化に関する情報を第 1 章と、第 2 章の「BLAS レベル 3 ルーチン」のセクションに追加。	1996 年 11 月
-005	2 次元の FFT を追加。1 次元 FFT と 2 次元 FFT の両方に C インターフェイスを追加。	1997 年 8 月
-006	インテル・マス・カーネル・ライブラリのリリース 2.1 について説明。第 2 章にスパース BLAS のセクションを追加。	1998 年 1 月
-007	インテル・マス・カーネル・ライブラリのリリース 3.0 について説明。LAPACK ルーチン (第 4 章と第 5 章) ならびに CBLAS インターフェイス (付録 C) の説明を追加。マニュアルから「クイック・リファレンス」を削除。現時点では、マス・カーネル・ライブラリ 3.0 の「クイック・リファレンス」が HTML 形式で利用可能。	1999 年 1 月
-008	マス・カーネル・ライブラリのリリース 3.2 について説明。FFT ルーチンの説明を変更。第 4 章と第 5 章で LAPACK ルーチンの NAG 名を削除。	1999 年 6 月
-009	固有値問題用の新しい LAPACK ルーチンを第 5 章に追加。	1999 年 11 月
-010	マス・カーネル・ライブラリのリリース 4.0 について説明。VML 関数について説明する第 6 章を追加。	2000 年 6 月
-011	マス・カーネル・ライブラリのリリース 5.1 について説明。LAPACK のセクションを拡張し、すべての計算ルーチンとドライバルーチンの一覧を追加。	2001 年 4 月
-6001	インテル・マス・カーネル・ライブラリのリリース 6.0 beta について説明。DFT インターフェイスとベクトル統計ライブラリ関数を新規に追加。	2002 年 7 月
-6002	インテル・マス・カーネル・ライブラリのリリース 6.0 beta update について説明。DFT 関数の説明を更新。CBLAS インターフェイスの説明を拡張。	2002 年 12 月
-6003	インテル・マス・カーネル・ライブラリのリリース 6.0 gold について説明。DFT 関数をアップデート。補助 LAPACK ルーチンの説明を本マニュアルに追加。	2003 年 3 月
-6004	インテル・マス・カーネル・ライブラリのリリース 6.1 について説明。	2003 年 7 月
-6005	インテル・マス・カーネル・ライブラリのリリース 7.0 beta について説明。ScaLAPACK およびスパースソルバの説明を含む。	2003 年 11 月
-017	インテル・マス・カーネル・ライブラリおよびインテル® クラスタ・マス・カーネル・ライブラリのリリース 7.0 gold について説明。補助 ScaLAPACK および代替スパース・ソルバ・インターフェイスの追加。	2004 年 4 月
-018	インテル・マス・カーネル・ライブラリおよびインテル・クラスタ・マス・カーネル・ライブラリのリリース 8.0 beta について説明。スパース BLAS および DFTI のセクションを拡張。新機能の追加: スパース BLAS、クラスタ DFTI、反復法スパースソルバ、多倍長演算、区間連立ソルバ、および畳み込み / 相関。LAPACK 関数に Fortran95 インターフェイスを追加。	2005 年 3 月
-019	インテル・マス・カーネル・ライブラリおよびインテル・クラスタ・マス・カーネル・ライブラリのリリース 7.0 gold について説明。BLAS およびスパース BLAS 関数に Fortran95 インターフェイスを追加。	2005 年 08 月

**【輸出規制に関する告知と注意事項】**

本資料に掲載されている製品のうち、外国為替および外国為替管理法に定める戦略物資等または役務に該当するものについては、輸出または再輸出する場合、同法に基づく日本政府の輸出許可が必要です。また、米国産品である当社製品は日本からの輸出または再輸出に際し、原則として米国政府の事前許可が必要です。

**【資料内容に関する注意事項】**

- ・本ドキュメントの内容を予告なしに変更することがあります。
- ・インテルでは、この資料に掲載された内容について、市販製品に使用した場合の保証あるいは特別な目的に合うことの保証等は、いかなる場合についてもいたしかねます。また、このドキュメント内の誤りについても責任を負いかねる場合があります。
- ・インテルは、インテル製品の内部回路以外の使用では責任を負いません。また、外部回路の特許についても関知いたしません。
- ・本書の情報はインテル製品を使用できるようにする目的でのみ記載されています。
- ・インテルは、製品について「取引条件」で提示されている場合を除き、インテル製品の販売や使用に関して、いかなる特許または著作権の侵害をも含み、あらゆる責任を負わないものとします。
- ・いかなる形および方法によっても、インテルの文書による許可なく、この資料の一部またはすべてを複写することは禁じられています。

Intel、インテル、Intel ロゴ、Pentium は、米国およびその他の国における Intel Corporation またはその子会社の商標または登録商標です。

\* その他の社名、商品名などは、一般に各社の商標または登録商標です。

© 1994-2005, Intel Corporation. 無断での引用、転載を禁じます。

Portions © Copyright 2001 Hewlett-Packard Development Company, L.P.

Chapters 4 and 5 include derivative work portions that have been copyrighted  
© 1991, 1992, and 1998 by The Numerical Algorithms Group, Ltd.





# 目次

---

<b>第 1 章</b>	<b>概要</b>	
	本ソフトウェアについて .....	1-1
	技術サポート .....	1-2
	BLAS ルーチン .....	1-3
	スパース BLAS ルーチン .....	1-3
	LAPACK ルーチン .....	1-3
	ScaLAPACK ルーチン .....	1-4
	スパース・ソルバ・ルーチン .....	1-4
	VML 関数 .....	1-4
	VSL 関数 .....	1-5
	フーリエ変換関数 .....	1-5
	区間ソルバルーチン .....	1-5
	性能の改善 .....	1-5
	並列化 .....	1-6
	対応プラットフォーム .....	1-7
	本書について .....	1-7
	本書の対象読者 .....	1-7
	本書の構成 .....	1-7
	表記の規則 .....	1-9
	ルーチン名の省略表記 .....	1-9
	字体の規則 .....	1-9
<b>第 2 章</b>	<b>BLAS ルーチンとスパース BLAS ルーチン</b>	
	BLAS のルーチンと関数 .....	2-2

ルーチン命名規則.....	2-2
Fortran-95 インターフェイス規則.....	2-3
行列の格納形式.....	2-4
BLAS レベル 1 のルーチンと関数.....	2-5
?asum.....	2-6
?axpy.....	2-7
?copy.....	2-9
?dot.....	2-10
?sdot.....	2-12
?dotc.....	2-13
?dotu.....	2-15
?nrm2.....	2-16
?rot.....	2-17
?rotg.....	2-19
?rotm.....	2-21
?rotmg.....	2-23
?scal.....	2-25
?swap.....	2-26
i?amax.....	2-28
i?amin.....	2-29
dcabs1.....	2-30
BLAS レベル 2 のルーチン.....	2-31
?gbmv.....	2-32
?gemv.....	2-35
?ger.....	2-38
?gerc.....	2-40
?geru.....	2-42
?hbmV.....	2-44
?hemv.....	2-47
?her.....	2-49
?her2.....	2-51
?hpmv.....	2-54
?hpr.....	2-56
?hpr2.....	2-58

?sbmv .....	2-61
?spmv .....	2-64
?spr .....	2-66
?spr2 .....	2-68
?symv .....	2-70
?syr .....	2-73
?syr2 .....	2-75
?tbmv .....	2-77
?tbsv .....	2-80
?tpmv .....	2-83
?tpsv .....	2-86
?trmv .....	2-88
?trsv .....	2-91
BLAS レベル 3 のルーチン .....	2-94
インテル <sup>®</sup> MKL の対称型マルチプロセッシング・バージョン .....	2-94
?gemm .....	2-95
?hemm .....	2-99
?herk .....	2-102
?her2k .....	2-105
?symm .....	2-108
?syrk .....	2-111
?syr2k .....	2-115
?trmm .....	2-119
?trsm .....	2-122
スパース BLAS レベル 1 のルーチンと関数 .....	2-126
ベクトルの引数 .....	2-126
命名規則 .....	2-127
ルーチンとデータ型 .....	2-127
スパースベクトルで利用できる BLAS レベル 1 のルーチン .....	2-127
?axpyi .....	2-128
?doti .....	2-130
?dotci .....	2-131
?dotui .....	2-133
?gthr .....	2-134

?gthrz .....	2-136
?roti .....	2-137
?sctr .....	2-139
スパース BLAS レベル 2 およびレベル 3 .....	2-141
スパース BLAS レベル 2 およびレベル 3 における命名規則 .....	2-141
スパース行列のデータ構造 .....	2-142
ルーチンおよびサポートされる演算 .....	2-142
標準インターフェイスを備えたルーチン .....	2-143
簡易インターフェイスを備えたルーチン .....	2-143
インターフェイス考慮事項 .....	2-144
インテル MKL インターフェイスと NIST インターフェイスの相違点 .....	2-144
簡易インターフェイス .....	2-146
部分行列の演算 .....	2-146
三角ソルバルーチンの制限 .....	2-148
スパース BLAS レベル 2 およびレベル 3 のルーチン .....	2-148
mkl_dcsrsv .....	2-149
mkl_dcsrcmv .....	2-152
mkl_dcsrcsymv .....	2-154
mkl_dcscmv .....	2-156
mkl_dcoomv .....	2-158
mkl_dcoogmv .....	2-161
mkl_dcoosymv .....	2-163
mkl_ddiamv .....	2-165
mkl_ddiagmv .....	2-167
mkl_ddiasymv .....	2-169
mkl_dskymv .....	2-171
mkl_dcsrcsv .....	2-173
mkl_dcsrtrsv .....	2-175
mkl_dcscsv .....	2-178
mkl_dcoosv .....	2-180
mkl_dcootrsv .....	2-182
mkl_ddiasv .....	2-184
mkl_ddiatrsv .....	2-187
mkl_dskysv .....	2-189

mkl_dcsrmm .....	2-191
mkl_dcscmm.....	2-194
mkl_dcoomm .....	2-196
mkl_ddiamm.....	2-199
mkl_dskymm.....	2-202
mkl_dcsrsm.....	2-204
dcscsm .....	2-207
mkl_dcoosm.....	2-210
mkl_ddiasm .....	2-212
mkl_dskysm .....	2-215

### 第 3 章

#### LAPACK ルーチン : 1 次方程式

ルーチン命名規則 .....	3-2
Fortran-95 インターフェイス規則 .....	3-3
行列の格納形式 .....	3-5
数学的表記 .....	3-5
誤差の分析 .....	3-6
計算ルーチン .....	3-7
行列の因子分解用のルーチン .....	3-8
?getrf.....	3-9
?gbtrf.....	3-11
?gttrf.....	3-13
?potrf.....	3-15
?pptrf.....	3-18
?pbtrf.....	3-20
?pttrf.....	3-22
?sytrf.....	3-23
?hetrf.....	3-26
?sptrf.....	3-29
?hptrf.....	3-32
連立 1 次方程式を解くためのルーチン .....	3-35
?getrs .....	3-35
?gbtrs .....	3-37
?gttrs .....	3-40

?potrs .....	3-43
?pptrs .....	3-45
?pbtrs .....	3-48
?pttrs .....	3-51
?sytrs .....	3-53
?hetrs .....	3-55
?sptrs .....	3-58
?hptrs .....	3-60
?trtrs .....	3-63
?tptrs .....	3-65
?tbtrs .....	3-68
条件数を推定するためのルーチン .....	3-71
?gecon .....	3-71
?gbcon .....	3-73
?gtcon .....	3-76
?pocon .....	3-78
?ppcon .....	3-81
?pbcon .....	3-83
?ptcon .....	3-85
?sycon .....	3-87
?hecon .....	3-90
?spcon .....	3-92
?hpcon .....	3-94
?trcon .....	3-96
?tpcon .....	3-99
?tbcon .....	3-101
解の精度の改善と誤差の推定 .....	3-103
?gerfs .....	3-104
?gbrfs .....	3-107
?gtrfs .....	3-110
?porfs .....	3-113
?pprfs .....	3-116
?pbrfs .....	3-119
?ptrfs .....	3-122

---

?syrrfs.....	3-125
?herfs .....	3-128
?sprfs .....	3-131
?hprfs .....	3-134
?trrfss.....	3-137
?tprfs .....	3-140
?tbrfs .....	3-143
行列の反転用のルーチン .....	3-146
?getri .....	3-146
?potri .....	3-149
?pptri .....	3-151
?sytri.....	3-153
?hetri .....	3-155
?sptri .....	3-157
?hptri .....	3-159
?trtri.....	3-161
?tptri .....	3-163
行列の平衡化.....	3-165
?geequ.....	3-165
?gbequ.....	3-168
?poequ .....	3-170
?ppequ .....	3-172
?pbequ .....	3-175
ドライバルーチン .....	3-177
?gesv .....	3-178
?gesvx .....	3-180
?gbsv .....	3-186
?gbsvx .....	3-189
?gtsv .....	3-196
?gtsvx.....	3-198
?posv .....	3-203
?posvx.....	3-205
?ppsv .....	3-210
?ppsvx.....	3-212

?pbsv .....	3-217
?pbsvx .....	3-220
?ptsv .....	3-225
?ptsvx .....	3-227
?sysv .....	3-230
?sysvx .....	3-233
?hesv .....	3-238
?hesvx .....	3-241
?spsv .....	3-245
?spsvx .....	3-248
?hpsv .....	3-252
?hpsvx .....	3-255

## 第 4 章

### LAPACK ルーチン : 最小二乗問題および固有値問題

ルーチン命名規則 .....	4-3
行列の格納形式 .....	4-4
数学的表記 .....	4-4
計算ルーチン .....	4-5
直交因子分解 .....	4-5
?geqrf .....	4-7
?geqpf .....	4-10
?geqp3 .....	4-13
?orgqr .....	4-16
?ormqr .....	4-18
?ungqr .....	4-21
?unmqr .....	4-23
?gelqf .....	4-26
?orglq .....	4-29
?ormlq .....	4-31
?unglq .....	4-34
?unmlq .....	4-36
?geqlf .....	4-39
?orgql .....	4-41
?ungql .....	4-43



?ormql .....	4-45
?unmq1.....	4-48
?gerqf .....	4-51
?orgrq .....	4-53
?ungrq .....	4-55
?ormrq.....	4-57
?unmrq.....	4-60
?tzzzf .....	4-62
?ormrz .....	4-65
?unmrz .....	4-68
?ggqrf .....	4-71
?ggrqf .....	4-74
特異値分解 .....	4-78
?gebrd .....	4-80
?gbbrd .....	4-83
?orgbr .....	4-87
?ormbr .....	4-90
?ungbr .....	4-93
?unmbr .....	4-97
?bdsqr .....	4-100
?bdsdc.....	4-104
対称固有値問題.....	4-107
?sytrd .....	4-112
?orgtr .....	4-114
?ormtr .....	4-116
?hetrd .....	4-119
?ungtr .....	4-122
?unmtr .....	4-124
?sptrd .....	4-126
?opgtr .....	4-128
?opmtr .....	4-130
?hptrd .....	4-133
?upgtr .....	4-135
?upmtr .....	4-137

?sbtrd.....	4-139
?hbtrd.....	4-141
?sterf.....	4-144
?steqr.....	4-146
?stedc.....	4-150
?stegr.....	4-154
?pteqr.....	4-159
?stebz.....	4-163
?stein.....	4-167
?disna.....	4-169
汎用対称固有値問題.....	4-172
?sygst.....	4-173
?hegst.....	4-175
?spgst.....	4-177
?hpgst.....	4-179
?sbgst.....	4-182
?hbgst.....	4-184
?pbstf.....	4-187
非対称固有値問題.....	4-189
?gehrd.....	4-193
?orghr.....	4-195
?ormhr.....	4-198
?unghr.....	4-201
?unmhr.....	4-204
?gebal.....	4-207
?gebak.....	4-210
?hseqr.....	4-212
?hsein.....	4-217
?trevc.....	4-222
?trsna.....	4-227
?trexc.....	4-232
?trsen.....	4-235
?trsy.....	4-240
汎用非対称固有値問題.....	4-243

---

?gghrd .....	4-244
?ggbal .....	4-247
?ggbak .....	4-250
?hgeqz .....	4-252
?tgevc .....	4-259
?tgexc .....	4-264
?tgsen .....	4-267
?tgsyl .....	4-273
?tgsna .....	4-278
汎用特異値分解 .....	4-282
?ggsvp .....	4-283
?tgsja .....	4-287
ドライバルーチン .....	4-294
線形最小二乗 (LLS) 問題 .....	4-294
?gels .....	4-295
?gelsy .....	4-298
?gelss .....	4-302
?gelsd .....	4-305
汎用 LLS 問題 .....	4-309
?gglse .....	4-310
?ggglm .....	4-313
対称固有値問題 .....	4-315
?syev .....	4-316
?heev .....	4-319
?syevd .....	4-321
?heevd .....	4-324
?syevx .....	4-328
?heevx .....	4-332
?syevr .....	4-337
?heevr .....	4-342
?spev .....	4-348
?hpev .....	4-350
?spevd .....	4-352
?hpevd .....	4-355

?spevx.....	4-359
?hpevx.....	4-363
?sbev.....	4-367
?hbev.....	4-370
?sbevd.....	4-372
?hbevd.....	4-376
?sbevz.....	4-380
?hbevz.....	4-384
?stev.....	4-389
?stevd.....	4-391
?stevz.....	4-394
?stevr.....	4-398
非対称固有値問題.....	4-403
?gees.....	4-404
?geesx.....	4-408
?geev.....	4-414
?geevz.....	4-419
特異値分解.....	4-425
?gesvd.....	4-426
?gesdd.....	4-430
?ggsvd.....	4-434
汎用対称固有値問題.....	4-440
?sygv.....	4-441
?hegv.....	4-444
?sygvd.....	4-447
?hegvd.....	4-451
?sygvz.....	4-455
?hegvz.....	4-460
?spgv.....	4-466
?hpgv.....	4-469
?spgvd.....	4-472
?hpgvd.....	4-475
?spgvz.....	4-479
?hpgvz.....	4-484

?sbgv .....	4-489
?hbgv .....	4-491
?sbgvd .....	4-494
?hbgvd .....	4-497
?sbgvx .....	4-501
?hbgvx .....	4-506
汎用非対称固有値問題 .....	4-511
?gges .....	4-511
?ggesx .....	4-518
?ggev .....	4-525
?ggevx .....	4-530

## 第 5 章

### LAPACK 補助ルーチンとユーティリティ・ルーチン

補助ルーチン .....	5-1
?lacgv .....	5-10
?lacrm .....	5-10
?lacrt .....	5-12
?laesy .....	5-13
?rot .....	5-14
?spmv .....	5-15
?spr .....	5-17
?symv .....	5-19
?syr .....	5-21
i?max1 .....	5-22
?sum1 .....	5-23
?gbtf2 .....	5-24
?gebd2 .....	5-25
?gehd2 .....	5-28
?gelq2 .....	5-30
?geql2 .....	5-32
?geqr2 .....	5-33
?gerq2 .....	5-35
?gesc2 .....	5-37
?getc2 .....	5-38

?getf2 .....	5-40
?gtts2 .....	5-41
?labrd .....	5-42
?lacon .....	5-45
?lacpy .....	5-47
?ladiv .....	5-48
?lae2 .....	5-49
?laebz .....	5-50
?laed0 .....	5-55
?laed1 .....	5-57
?laed2 .....	5-59
?laed3 .....	5-62
?laed4 .....	5-64
?laed5 .....	5-66
?laed6 .....	5-67
?laed7 .....	5-68
?laed8 .....	5-72
?laed9 .....	5-75
?laeda .....	5-77
?laein .....	5-79
?laev2 .....	5-82
?laexc .....	5-84
?lag2 .....	5-85
?lags2 .....	5-87
?lagtf .....	5-89
?lagtm .....	5-91
?lagts .....	5-92
?lagv2 .....	5-94
?lahqr .....	5-96
?lahrd .....	5-99
?laic1 .....	5-101
?lain2 .....	5-104
?lals0 .....	5-107
?lalsa .....	5-111

?lalsd.....	5-114
?lamrg.....	5-117
?langb.....	5-118
?lange.....	5-119
?langt.....	5-121
?lanhs.....	5-122
?lansb.....	5-124
?lanhb.....	5-125
?lansp.....	5-127
?lanhp.....	5-129
?lanst/?lanht.....	5-130
?lansy.....	5-132
?lanhe.....	5-133
?lantb.....	5-135
?lantp.....	5-137
?lantr.....	5-138
?lanv2.....	5-140
?lapll.....	5-141
?lapmt.....	5-142
?lapy2.....	5-144
?lapy3.....	5-144
?laqgb.....	5-145
?laqge.....	5-147
?laqp2.....	5-149
?laqps.....	5-151
?laqsb.....	5-153
?laqsp.....	5-155
?laqsy.....	5-156
?laqtr.....	5-158
?lar1v.....	5-161
?lar2v.....	5-163
?larf.....	5-164
?larfb.....	5-166
?larfg.....	5-168

?larft .....	5-170
?larfx .....	5-173
?largv .....	5-174
?larnv .....	5-176
?larrb .....	5-177
?larre .....	5-179
?larrf .....	5-181
?larrv .....	5-183
?lartg .....	5-185
?lartv .....	5-187
?laruv .....	5-188
?larz .....	5-189
?larzb .....	5-191
?larzt .....	5-193
?las2 .....	5-197
?lascl .....	5-198
?lasd0 .....	5-200
?lasd1 .....	5-201
?lasd2 .....	5-204
?lasd3 .....	5-208
?lasd4 .....	5-211
?lasd5 .....	5-212
?lasd6 .....	5-214
?lasd7 .....	5-218
?lasd8 .....	5-222
?lasd9 .....	5-224
?lasda .....	5-226
?lasdq .....	5-229
?lasdt .....	5-232
?laset .....	5-233
?lasq1 .....	5-234
?lasq2 .....	5-235
?lasq3 .....	5-237
?lasq4 .....	5-239



?lasq5 .....	5-240
?lasq6 .....	5-242
?lasr .....	5-243
?lasrt .....	5-245
?lassq .....	5-246
?lasv2 .....	5-247
?laswp .....	5-249
?lasy2 .....	5-250
?lasyf .....	5-252
?lahef .....	5-255
?latbs .....	5-257
?latdf .....	5-260
?latps .....	5-262
?latrd .....	5-264
?latrs .....	5-267
?latrz .....	5-271
?lauu2 .....	5-273
?lauum .....	5-275
?org2l/?ung2l .....	5-276
?org2r/?ung2r .....	5-278
?orgl2/?ungl2 .....	5-279
?orgr2/?ungr2 .....	5-281
?orm2l/?unm2l .....	5-282
?orm2r/?unm2r .....	5-285
?orml2/?unml2 .....	5-287
?ormr2/?unmr2 .....	5-290
?ormr3/?unmr3 .....	5-292
?pbt2 .....	5-295
?potf2 .....	5-296
?ptts2 .....	5-298
?rscl .....	5-299
?sygs2/?hegs2 .....	5-300
?sytd2/?hetd2 .....	5-302
?sytf2 .....	5-304

?hetf2 .....	5-306
?tgex2 .....	5-308
?tgsy2 .....	5-310
?trti2 .....	5-314
ユーティリティ関数とルーチン .....	5-316
ilaenv .....	5-316
ieeeck .....	5-319
lsame .....	5-320
lsamen .....	5-320
?labad .....	5-321
?lamch .....	5-322
?lamc1 .....	5-323
?lamc2 .....	5-324
?lamc3 .....	5-325
?lamc4 .....	5-325
?lamc5 .....	5-326
second/dsecnd .....	5-327
xerbla .....	5-327

## 第 6 章

### ScaLAPACK ルーチン

概要 .....	6-2
ルーチン命名規則 .....	6-3
計算ルーチン .....	6-4
1 次方程式 .....	6-4
行列の因子分解用のルーチン .....	6-6
p?getrf .....	6-6
p?gbtrf .....	6-8
p?dbtrf .....	6-10
p?potrf .....	6-13
p?pbtrf .....	6-15
p?pttrf .....	6-17
p?dttrf .....	6-19
連立 1 次方程式を解くためのルーチン .....	6-22
p?getrs .....	6-22

---

p?gbtrs.....	6-24
p?potrs.....	6-27
p?pbtrs.....	6-29
p?pttrs.....	6-31
p?dttrs.....	6-34
p?dbtrs.....	6-37
p?trtrs.....	6-40
条件数を推定するためのルーチン.....	6-42
p?gecon.....	6-42
p?pocon.....	6-45
p?trcon.....	6-48
解の精度の改善と誤差の推定.....	6-51
p?gerfs.....	6-52
p?porfs.....	6-56
p?trrfs.....	6-60
行列の反転用のルーチン.....	6-64
p?getri.....	6-64
p?potri.....	6-66
p?trtri.....	6-68
行列の平衡化.....	6-70
p?geequ.....	6-70
p?poequ.....	6-72
直交因子分解.....	6-75
p?geqrf.....	6-75
p?geqpf.....	6-78
p?orgqr.....	6-81
p?ungqr.....	6-83
p?ormqr.....	6-85
p?unmqr.....	6-89
p?gelqf.....	6-92
p?orglq.....	6-95
p?unglq.....	6-97
p?ormlq.....	6-99
p?unmlq.....	6-103

p?geqlf .....	6-106
p?orgql .....	6-109
p?ungql .....	6-111
p?ormql .....	6-113
p?unmql .....	6-117
p?gerqf .....	6-120
p?orgrq .....	6-123
p?ungrq .....	6-125
p?ormrq .....	6-127
p?unmrq .....	6-131
p?tzzrf .....	6-134
p?ormrz .....	6-137
p?unmrz .....	6-141
p?ggqrf .....	6-145
p?ggrqf .....	6-149
対称固有値問題 .....	6-155
p?sytrd .....	6-156
p?ormtr .....	6-160
p?hetrd .....	6-163
p?unmtr .....	6-167
p?stebz .....	6-171
p?stein .....	6-175
非対称固有値問題 .....	6-179
p?gehrd .....	6-180
p?ormhr .....	6-184
p?unmhr .....	6-187
p?lahqr .....	6-190
特異値分解 .....	6-193
p?gebrd .....	6-193
p?ormbr .....	6-198
p?unmbr .....	6-203
汎用対称固有値問題 .....	6-208
p?sygst .....	6-208
p?hegst .....	6-210

ドライバルーチン .....	6-213
p?gesv .....	6-214
p?gesvx .....	6-216
p?gbsv .....	6-222
p?dbsv .....	6-225
p?dtsv .....	6-228
p?posv .....	6-231
p?posvx .....	6-233
p?pbsv .....	6-240
p?ptsv .....	6-242
p?gels .....	6-245
p?syev .....	6-249
p?syevx .....	6-252
p?heevx .....	6-259
p?gesvd .....	6-267
p?sygvx .....	6-271
p?hegvx .....	6-279

## 第 7 章

### ScaLAPACK 補助ルーチンとユーティリティ・ルーチン

補助ルーチン .....	7-1
p?lacgv .....	7-6
p?max1 .....	7-7
?combamax1 .....	7-8
p?sum1 .....	7-9
p?dbtrsv .....	7-10
p?dttrsv .....	7-13
p?gebd2 .....	7-17
p?gehd2 .....	7-21
p?gelq2 .....	7-24
p?geql2 .....	7-26
p?geqr2 .....	7-29
p?gerq2 .....	7-31
p?getf2 .....	7-34
p?labrd .....	7-35

p?lacon .....	7-39
p?laconsb .....	7-41
p?lcp2 .....	7-43
p?lcp3 .....	7-44
p?lacpy .....	7-46
p?laevswp .....	7-48
p?lahrd .....	7-50
p?laiect .....	7-53
p?lange .....	7-54
p?lanhs .....	7-56
p?lansy, p?lanhe .....	7-58
p?lantr .....	7-61
p?lapiv .....	7-63
p?laqge .....	7-66
p?laqsy .....	7-69
p?lared1d .....	7-71
p?lared2d .....	7-72
p?larf .....	7-74
p?larfb .....	7-77
p?larfc .....	7-80
p?larfg .....	7-83
p?larft .....	7-85
p?larz .....	7-88
p?larzb .....	7-92
p?larzc .....	7-96
p?larzt .....	7-99
p?lascl .....	7-103
p?laset .....	7-105
p?lasmsub .....	7-107
p?lassq .....	7-108
p?laswp .....	7-110
p?latra .....	7-112
p?latrd .....	7-113
p?latrs .....	7-117

---

p?latrz .....	7-119
p?lauu2 .....	7-122
p?lauum .....	7-124
p?lawil.....	7-125
p?org2l/p?ung2l .....	7-126
p?org2r/p?ung2r .....	7-129
p?orgl2/p?ungl2 .....	7-131
p?orgr2/p?ungr2 .....	7-134
p?orm2l/p?unm2l.....	7-136
p?orm2r/p?unm2r .....	7-140
p?orml2/p?unml2.....	7-144
p?ormr2/p?unmr2 .....	7-148
p?pbtrsv .....	7-152
p?pttrsv .....	7-156
p?potf2.....	7-159
p?rscl.....	7-161
p?sygs2/p?hegs2.....	7-162
p?sytd2/p?hetd2 .....	7-165
p?trti2.....	7-169
?lamsh .....	7-171
?laref.....	7-172
?lasorte.....	7-175
?lasrt2 .....	7-176
?stein2 .....	7-177
?dbtf2 .....	7-179
?dbtrf.....	7-181
?dttrf.....	7-183
?dttrsv .....	7-184
?pttrsv .....	7-186
?steqr2.....	7-188
ユーティリティ関数とルーチン .....	7-190
p?labad.....	7-190
p?lachkieee .....	7-191
p?lamch .....	7-192

p?lasnbt.....	7-193
pxerbla .....	7-194

## 第 8 章

### スパース・ソルバ・ルーチン

PARDISO – 並列化対応直接法スパース・ソルバ・インターフェイス.....	8-1
pardiso .....	8-3
直接法スパースソルバ (DSS) インターフェイス・ルーチン .....	8-14
インターフェイスの説明 .....	8-16
ルーチン・オプション .....	8-17
ユーザデータ配列 .....	8-17
DSS ルーチン .....	8-18
dss_create .....	8-18
dss_define_structure.....	8-19
dss_reorder .....	8-20
dss_factor_real、dss_factor_complex .....	8-21
dss_solve_real、dss_solve_complex .....	8-23
dss_delete.....	8-24
dss_statistics .....	8-25
mkl_cvt_to_null_terminated_str .....	8-28
実装の詳細 .....	8-28
メモリ割り当てとハンドル .....	8-29
リバース・コミュニケーション・インターフェイス (RCI ISS) に基づく反復	
法スパースソルバ.....	8-30
共役勾配ソルバ (RCI CG) .....	8-30
インターフェイスの説明 .....	8-33
ルーチン・オプション .....	8-33
ユーザデータ配列 .....	8-33
共通パラメータ .....	8-34
RCI CG ルーチン .....	8-37
dcg_init.....	8-37
dcg_check .....	8-38
dcg.....	8-39
dcg_get.....	8-41
実装の詳細 .....	8-42



---

C/C++ からのスパース・ソルバ・ルーチンの呼び出し .....	8-43
C ユーザが気をつけるべきこと .....	8-44
<b>第 9 章</b>	
<b>ベクトル数学関数</b>	
データ型と精度モード .....	9-2
関数命名規則 .....	9-2
関数インターフェイス .....	9-3
VML 数学関数 .....	9-3
Pack 関数 .....	9-4
Unpack 関数 .....	9-4
サービス関数 .....	9-4
入力パラメータ .....	9-5
出力パラメータ .....	9-5
ベクトル・インデックス方式 .....	9-6
エラー診断 .....	9-6
VML 数学関数 .....	9-7
Inv .....	9-9
Div .....	9-10
Sqrt .....	9-11
InvSqrt .....	9-12
Cbrt .....	9-13
InvCbrt .....	9-14
Pow .....	9-15
Powx .....	9-16
Exp .....	9-18
Ln .....	9-19
Log10 .....	9-20
Cos .....	9-21
Sin .....	9-22
SinCos .....	9-23
Tan .....	9-24
Acos .....	9-25
Asin .....	9-26
Atan .....	9-27

Atan2.....	9-28
Cosh.....	9-29
Sinh.....	9-30
Tanh.....	9-31
Acosh.....	9-32
Asinh.....	9-33
Atanh.....	9-34
Erf.....	9-35
Erfc.....	9-37
VML Pack/Unpack 関数.....	9-38
Pack.....	9-39
Unpack.....	9-41
VML サービス関数.....	9-43
SetMode.....	9-44
GetMode.....	9-46
SetErrStatus.....	9-47
GetErrStatus.....	9-48
ClearErrStatus.....	9-49
SetErrorCallBack.....	9-49
GetErrorCallBack.....	9-52
ClearErrorCallBack.....	9-53

## 第 10 章

### 統計関数

乱数生成器.....	10-1
規則.....	10-2
数学的表記.....	10-3
命名規則.....	10-4
基本生成器.....	10-8
BRNG パラメータの定義.....	10-9
ランダム・ストリーム.....	10-10
データ型.....	10-11
エラー報告.....	10-11
サービスルーチン.....	10-12
NewStream.....	10-14

NewStreamEx .....	10-15
iNewAbstractStream .....	10-17
dNewAbstractStream .....	10-19
sNewAbstractStream .....	10-22
DeleteStream .....	10-24
CopyStream .....	10-25
CopyStreamState .....	10-26
SaveStreamF .....	10-28
LoadStreamF .....	10-29
LeapfrogStream .....	10-31
SkipAheadStream .....	10-34
GetStreamStateBrng .....	10-37
GetNumRegBrngs .....	10-38
分布生成器 .....	10-39
連続分布 .....	10-40
Uniform .....	10-40
Gaussian .....	10-43
GaussianMV .....	10-45
Exponential .....	10-50
Laplace .....	10-52
Weibull .....	10-54
Cauchy .....	10-57
Rayleigh .....	10-59
Lognormal .....	10-61
Gumbel .....	10-64
Gamma .....	10-66
Beta .....	10-68
離散分布 .....	10-71
Uniform .....	10-71
UniformBits .....	10-73
Bernoulli .....	10-75
Geometric .....	10-77
Binomial .....	10-79
Hypergeometric .....	10-81

Poisson.....	10-83
PoissonV.....	10-85
NegBinomial.....	10-87
アドバンスト・サービス・ルーチン .....	10-89
データ型 .....	10-89
RegisterBrng .....	10-91
GetBrngProperties .....	10-92
ユーザ定義生成器のフォーマット .....	10-93
iBRng.....	10-95
sBRng .....	10-96
dBRng.....	10-97
畳み込みと相関 .....	10-97
概要 .....	10-97
命名規則 .....	10-99
データ型 .....	10-100
パラメータ .....	10-100
タスク・ステータス.....	10-102
タスク・コンストラクタ .....	10-103
NewTask .....	10-104
NewTask1D.....	10-106
NewTaskX .....	10-108
NewTaskX1D.....	10-111
タスクエディタ .....	10-114
SetMode.....	10-115
SetInternalPrecision .....	10-116
SetStart .....	10-118
SetDecimation.....	10-119
タスク実行ルーチン .....	10-121
Exec.....	10-122
Exec1D .....	10-124
ExecX.....	10-126
ExecX1D .....	10-128
タスク・ディストラクタ .....	10-130
DeleteTask.....	10-130

タスクコピー .....	10-131
CopyTask .....	10-131
使用法の例 .....	10-133
マルチスレッドの使用 .....	10-135
数学表記と定義 .....	10-136
線形畳み込み .....	10-137
線形相関 .....	10-137
データの割り当て .....	10-137
有限関数とデータベクトル .....	10-138
データベクトルの割り当て .....	10-139

## 第 11 章      フーリエ変換関数

DFT 関数 .....	11-1
DFT の計算 .....	11-3
DFT インターフェイス .....	11-3
ステータス確認関数 .....	11-5
ErrorClass .....	11-5
ErrorMessage .....	11-7
ディスクリプタ操作 .....	11-8
CreateDescriptor .....	11-8
CommitDescriptor .....	11-10
CopyDescriptor .....	11-11
FreeDescriptor .....	11-13
DFT 計算 .....	11-14
ComputeForward .....	11-14
ComputeBackward .....	11-16
ディスクリプタの構成 .....	11-19
SetValue .....	11-19
GetValue .....	11-21
構成設定 .....	11-24
変換精度 .....	11-27
順方向変換 .....	11-28
変換の次元と長さ .....	11-28
変換回数 .....	11-29

倍率.....	11-29
結果の配置.....	11-29
圧縮形式.....	11-29
格納体系.....	11-33
ユーザスレッド数.....	11-42
入力と出力との距離.....	11-42
ストライド.....	11-43
順序指定.....	11-45
転置.....	11-45
クラスタ DFT 関数.....	11-47
クラスタ DFT の計算.....	11-49
クラスタ DFT インターフェイス.....	11-52
ディスクリプタ操作.....	11-53
CreateDescriptorDM.....	11-53
CommitDescriptorDM.....	11-55
FreeDescriptorDM.....	11-57
DFT 計算.....	11-58
ComputeForwardDM.....	11-58
ComputeBackwardDM.....	11-60
ディスクリプタの構成.....	11-62
SetValueDM.....	11-63
GetValueDM.....	11-65
ステータス確認関数.....	11-67
高速フーリエ変換（非推奨）.....	11-68
1 次元 FFT.....	11-68
データ格納の型.....	11-68
データ構造の要件.....	11-69
複素数から複素数への 1 次元 FFT.....	11-70
cfft1d/zfft1d（非推奨）.....	11-71
cfft1dc/zfft1dc（非推奨）.....	11-72
実数から複素数への 1 次元 FFT.....	11-73
scfft1d/dzfft1d（非推奨）.....	11-75
scfft1dc/dzfft1dc（非推奨）.....	11-76
複素数から実数への 1 次元 FFT.....	11-78

csfft1d/zdfft1d (非推奨).....	11-79
csfft1dc/zdfft1dc (非推奨).....	11-81
2 次元 FFT.....	11-82
複素数から複素数への 2 次元 FFT.....	11-83
cfft2d/zfft2d (非推奨).....	11-84
cfft2dc/zfft2dc (非推奨).....	11-85
実数から複素数への 2 次元 FFT.....	11-86
scfft2d/dzfft2d (非推奨).....	11-87
scfft2dc/dzfft2dc (非推奨).....	11-90
複素数から実数への 2 次元 FFT.....	11-93
csfft2d/zdfft2d (非推奨).....	11-94
csfft2dc/zdfft2dc (非推奨).....	11-95

## 第 12 章

### 区間線形ソルバ

ルーチン命名規則.....	12-2
区間方程式を高速に解くためのルーチン.....	12-3
?trtrs.....	12-3
?gegas.....	12-5
?gehss.....	12-7
?gekws.....	12-8
?gegss.....	12-10
?gehbs.....	12-12
区間方程式の精密な解を求めるためのルーチン.....	12-13
?gepps.....	12-13
区間行列逆転用のルーチン.....	12-16
?trtri.....	12-16
?geszi.....	12-18
区間行列の特性確認用のルーチン.....	12-19
?gerbr.....	12-19
?gesvr.....	12-20
補助およびユーティリティ・ルーチン.....	12-23
?gemip.....	12-23

## 付録 A 線形ソルバの基礎

スパース連立 1 次方程式.....	A-1
行列の基礎事項.....	A-2
直接法.....	A-3
スパース行列のフィルインとリオーダーリング.....	A-4
スパース行列の格納形式.....	A-8
PARDISO ソルバの格納形式.....	A-8
スパース BLAS レベル 2-3 のスパース格納形式.....	A-11
CSR 形式.....	A-11
CSC 形式.....	A-13
座標形式.....	A-14
対角格納方式.....	A-15
スカイライン格納方式.....	A-16
区間線形方程式.....	A-16
区間.....	A-16
区間ベクトルと行列.....	A-18
区間線形方程式.....	A-19
前処理.....	A-22
区間行列の逆転.....	A-22

## 付録 B ルーチンおよび関数の引数

BLAS のベクトル引数.....	B-1
VML のベクトル引数.....	B-3
正増分インデックス.....	B-3
インデックス・ベクトル・インデックス.....	B-3
マスク・ベクトル・インデックス.....	B-3
行列引数.....	B-4

## 付録 C コード例

BLAS コードの例.....	C-1
PARDISO コードの例.....	C-7
スパース対称連立 1 次方程式の例.....	C-7
対称連立 1 次方程式の計算結果の例.....	C-7
スパース非対称連立 1 次方程式の例.....	C-17



---

非対称連立 1 次方程式の計算結果の例 .....	C-17
直接法スパース・ソルバのコード例.....	C-27
対称連立 1 次方程式の計算結果の例 .....	C-27
反復法スパース・ソルバのコード例.....	C-35
RCI (プリコンディショニング) 共役勾配ソルバの使用例 .....	C-35
DFT コードの例.....	C-40
DFT 計算でのマルチスレッディングの使用例.....	C-49
区間連立 1 次方程式ソルバのコード例 .....	C-55

## 付録 D

### BLAS に対する CBLAS インターフェイス

CBLAS の引数 .....	D-1
列挙型.....	D-2
レベル 1 の CBLAS.....	D-3
レベル 2 の CBLAS.....	D-5
レベル 3 の CBLAS.....	D-12
スパース CBLAS .....	D-16

## 用語集

## 参考文献

## 索引



インテル® マス・カーネル・ライブラリ (インテル® MKL) では、ベクトルおよびスパース行列や区間行列などの行列に対して広範囲にわたる演算を実行する Fortran のルーチンおよび関数を提供する。また、Fortran および C のインターフェイスを使用するベクトル数値演算関数とベクトル統計関数だけでなく、離散フーリエ変換関数も含んでいる。

インテル® クラスタ MKL と呼ばれるライブラリのバージョンはインテル MKL のスーパーセットで、分散メモリの並列処理コンピュータ上での個々の計算問題を解決するための ScaLAPACK ソフトウェアと Cluster DFT ソフトウェアが含まれる。

インテル MKL は最新のインテル® プロセッサ用に最適化されているため、MKL を使用することにより、アプリケーション・プログラムの性能が向上する。  
本章では、マス・カーネル・ライブラリを紹介するとともに、本マニュアルの構成について説明する。

## 本ソフトウェアについて

インテル・マス・カーネル・ライブラリには、以下に示すグループのルーチンが含まれる。

- BLAS (Basic Linear Algebra Subprograms)
  - ベクトル演算
  - 行列 - ベクトル演算
  - 行列 - 行列演算
- スパース BLAS レベル 1、2、および 3 (スパースベクトルと行列に対する基本演算)
- 連立 1 次方程式を解くための LAPACK ルーチン
- 最小二乗問題、固有値ならびに特異値問題、およびシルベスター式を解くための LAPACK ルーチン

- 補助 LAPACK ルーチン
- ScaLAPACK の計算、ドライバ、および 補助ルーチン ( インテル・クラスタ MKL のみ )
- 直接法および反復法スパース・ソルバ・ルーチン
- ベクトル引数に対する基本的な数値演算関数を計算するための VML (Vector Mathematical Library) 関数 (Fortran および C のインターフェイスを使用)
- さまざまな種類の統計的分布に従った擬似乱数ベクトルを生成および畳み込みや相関計算を実行するベクトル統計ライブラリ (VSL) 関数
- Fortran および C のインターフェイスを使用する一般的な離散フーリエ変換関数 (DFT) と高速フーリエ変換ルーチン (FFT) のサブセット
- クラスタ DFT 関数 ( インテル・クラスタ MKL のみ )
- 区間連立 1 次方程式を解くための区間ソルバルーチン

ライブラリの使用に関する詳細は、*MKL* リリースノートを参照のこと。

## 技術サポート

インテル MKL には、製品の特徴、ホワイトペーパー、技術記事など、製品に関する総合的な情報が適宜掲載される製品 Web サイトが用意されている。最新情報については、以下のサイトを参照のこと。

<http://www.intel.co.jp/jp/developer/software/products/>

インテルでは、基本操作のヒント、製品に関する確認済みの問題点、製品のエラッタ、ライセンス情報、ユーザ・フォーラムなど、大量のセルフヘルプ情報を利用できるサポート Web サイトも提供している (<http://support.intel.co.jp/> を参照)。

ユーザ登録を行うと、インテル® プレミア・サポートによる 1 年間の技術サポートと製品アップデートのサービスが受けられる。インテル・プレミア・サポートは、以下のサービスを提供する双方向型の問題管理 / コミュニケーション Web サイトである。

- 問題点の送信とその状態の検討
- 製品のアップデートのダウンロード (1 日 24 時間)

ユーザ登録、インテルへの問い合わせ、製品サポートについては、以下のサイトを参照のこと。

<http://www.intel.com/software/products/support> ( 英語 )

## BLAS ルーチン

BLAS のルーチンと関数は、実行する演算によって以下のグループに分類される。

- [BLAS レベル 1 のルーチンと関数](#)は、ベクトルデータに対して加算と減算を実行する。典型的な演算として、スケーリングや内積などがある。
- [BLAS レベル 2 のルーチン](#)は、行列 - ベクトルの乗算、階数 1 と階数 2 の行列の更新、三角法の計算などの行列 - ベクトル演算を実行する。
- [BLAS レベル 3 のルーチン](#)は、行列 - 行列の乗算、階数 k の更新、三角法の計算などの行列 - 行列演算を実行する。

インテル MKL 8.0 以降では、BLAS ルーチンに対する Fortran 95 インターフェイスもサポートしている。

## スパース BLAS ルーチン

[スパース BLAS レベル 1 のルーチンと関数](#) と [スパース BLAS レベル 2 およびレベル 3 ルーチン](#) および関数は、スパースベクトルと行列での演算を行う。これらのルーチンは、BLAS レベル 1、2、および 3 ルーチンと同じようなベクトル演算を実行する。スパース BLAS ルーチンは、ベクトルが疎であることを利用し、ベクトルの非零成分だけを格納できる。また、インテル MKL では、スパース BLAS ルーチンに対する Fortran 95 インターフェイスをサポートしている。

## LAPACK ルーチン

インテル・マス・カーネル・ライブラリは、LAPACK の計算、ドライバルーチン、補助ルーチン、およびユーティリティ・ルーチンをすべて含んでいる。

インテル MKL の一部の基となった LAPACK の原版は <http://www.netlib.org/lapack/index.html> から入手できる。LAPACK の開発は、E. Anderson、Z. Bai、C. Bischof、S. Blackford、J. Demmel、J. Dongarra、J. Du Croz、A. Greenbaum、S. Hammarling、A. McKenney、D. Sorensen らによって行われた。

LAPACK ルーチンは、実行する演算によって以下のグループに分類できる。

- 連立 1 次方程式を解くためのルーチンや、行列の因子分解ならびに逆行列の計算、条件数の推定などを実行するためのルーチン ([第 3 章](#)を参照)。
- 最小二乗問題、固有値ならびに特異値問題、およびシルベスター式を解くためのルーチン ([第 4 章](#)を参照)。
- 特定のサブタスク、共通のローレベル計算、または関連するタスクに使用される補助ルーチンおよびユーティリティ ([第 5 章](#)を参照)。

インテル MKL 8.0 以降では、LAPACK の計算およびドライバルーチンに対する Fortran 95 インターフェイスもサポートしている。このインターフェイスにより、必要な引数を少なくして LAPACK ルーチンの呼び出しを簡略化できる。

## ScaLAPACK ルーチン

ScaLAPACK パッケージ ( インテル・クラスタ MKL のみ。第 6 章および第 7 章を参照 ) は、分散メモリ・アーキテクチャ上で動作し、連立 1 次方程式の解の算出、線形最小二乗問題、固有値、特異値問題、およびそれらに関連する各種の計算タスクを実行するためのルーチンを提供する。

インテル・クラスタ MKL の一部の基となった ScaLAPACK の原版は <http://www.netlib.org/scalapack/index.html> から入手できる。ScaLAPACK の著者は、L. Blackford、J. Choi、A. Cleary、E. D'Azevedo、J. Demmel、I. Dhillon、J. Dongarra、S. Hammarling、G. Henry、A. Petitet、K. Stanley、D. Walker、および R. Whaley である。

ScaLAPACK のインテル・クラスタ MKL バージョンは、インテル・プロセッサ向けに最適化されており、MPI の MPICH バージョンを使用する。

## スパース・ソルバ・ルーチン

インテル MKL の直接法スパース・ソルバ・ルーチン ( 第 8 章を参照 ) は、実数または複素数係数の対称および対称構造スパース行列を解く。対称行列において、これらのインテル MKL サブルーチンは、正定値式と不定値式の両方を計算できる。インテル MKL には、ユーザからの呼び出しが可能な直接法スパース・ソルバ・ルーチンの代替セットだけでなく、PARDISO\* スパース・ソルバ・インターフェイスも含まれる。

また、スパース BLAS レベル 2 および 3 ルーチンを使用し、異なるスパースデータ形式で動作する反復法スパースソルバ ( 第 8 章を参照 ) も含まれる。

## VML 関数

VML (Vector Mathematical Library) 関数 ( 第 9 章を参照 ) には、実ベクトルの引数を操作する、大量の計算を必要とする一連の基本的な数値演算関数 ( べき乗、三角関数、指数関数、双曲線関数など ) の高度に最適化されたコードが含まれる。

## VSL 関数

ベクトル統計ライブラリ (VSL) には、2つの関数群のセットが含まれる (第10章を参照)。1つ目のセットは、基本的な連続分布または離散分布に対応した擬似乱数生成サブルーチン群で構成されている。VSL サブルーチンは高度に最適化された基本乱数生成器とベクトル数学関数ライブラリ VML を呼び出すことで最高レベルの性能を実現している。2つ目のセットは、広範囲にわたる畳み込みと相関演算に対応したサブルーチン群で構成されている。

## フーリエ変換関数

インテル MKL 多次元離散フーリエ変換関数の混合基数サポート (第11章を参照) は、DFT 計算機能の統一性を提供するとともに、多機能性と使いやすさの両立を実現する。Fortran と C のインターフェイス仕様が含まれる。また、分散メモリ・アーキテクチャで動作する DFT 関数のクラスタ・バージョンは、インテル・クラスタ MKL パッケージで提供される。

以前のバージョンとの互換性のために、インテル MKL では2の累乗の変換サイズをサポートする単純な1次元と2次元の高速フーリエ変換関数を提供している。これらのFFT関数については、現在その使用は推奨されず、機能やパフォーマンスは上記で説明したDFTのそれらとは異なる。

現在では、DFT およびクラスタ DFT 関数のみが継続して開発、最適化されているため、アプリケーションではFFTの代わりにこれらの関数を使用する。

## 区間ソルバルーチン

インテル MKL (第12章を参照) に含まれる区間ソルバルーチンは、区間連立1次方程式および関連する問題を解くために使用できる。

## 性能の改善

インテル・マス・カーネル・ライブラリは、プロセッサおよびシステム両方の機能と能力を利用することによって最適化される。これらのルーチンでは、その大半がキャッシュ管理テクノロジーのメリットを活かせるよう特に注意が払われている。そのメリットを最大限に活かせるのが、`dgemm()` などの行列 - 行列演算である。

また、整数演算ユニットと浮動小数点演算ユニットのスケジュールがプロセッサ内の結果に依存するのを最小限に抑えるため、コード最適化技法が適用されている。

ライブラリで使用される主な最適化技法には、次のものがある。

- ループの繰り返しを避けて、ループ管理コストを低減。
- データをブロック化して、データ再利用の可能性を改善。
- コピーにより、データがキャッシュから追い出される可能性を低減。
- データをプリフェッチして、メモリ・レイテンシをカバー。
- `dgemm` での内積などの複数の演算を同時に実行して、演算ユニットのパイプラインが原因で生じるストールを排除。
- 必要に応じて、SIMD 算術演算ユニットなどのハードウェア機能を使用。

これらの技法により、演算コードにおいてメリットを最大限に活用できる。

## 並列化

すでに説明した性能改善に加え、インテル MKL では対称型マルチプロセッシング (SMP) 機能の採用により可能になった並列化で、さらに性能を向上させている。以下に示す方法で性能改善を図れる。

- プログラム内のユーザ管理スレッドを基本とし、さらにデータ分解、領域分解、制御分解、あるいはその他の並列技法に基づいて複数のスレッドに処理を分配する。ライブラリはスレッドセーフとなるよう設計されているため、それぞれのスレッドで任意のインテル MKL 関数を使用できる。
- FFT ルーチンや BLAS レベル 3 ルーチンを使用する。これらのルーチンは並列化されているため、アプリケーションを変更しなくても多重処理による性能改善が得られる。BLAS レベル 3 で複数のプロセッサを使用した場合の性能は、プロセッサ数に比例して改善される。スレッドはライブラリ内で呼び出されて管理されるため、アプリケーションをスレッドセーフに再コンパイルする必要がない (第 2 章の [Fortran-95 インターフェイス規則](#) を参照)。
- 性能改善に役立つもう 1 つの方法は、機能調整した LAPACK ルーチンの使用である。現時点では、これらのルーチンには、一般行列の QR 因子分解、一般行列と対称正定値行列の三角因子分解、このような行列を使用しての連立方程式の解、対称固有値問題の解を得るための単精度と倍精度のルーチンが含まれている。

BLAS レベル 3 ルーチンや LAPACK ルーチンに対して使用可能なプロセッサ数を設定する方法については、インテル MKL テクニカル・ユーザ・ノートを参照のこと。



## 対応プラットフォーム

インテル・マス・カーネル・ライブラリ (MKL) には、多重処理をサポートするオペレーティング・システム上で稼働するインテル® プロセッサ・ベースのコンピュータ用に最適化された Fortran のルーチンと関数を含んでいる。また、インテル MKL には、Fortran インターフェイスに加え、ベクトル数値演算ライブラリ関数とベクトル統計ライブラリ関数だけでなく、離散フーリエ変換関数に対応する C 言語インターフェイスも含まれている。

インテル MKL を使用する際のハードウェアおよびソフトウェアの動作環境に関する詳細は、MKL リリースノートを参照のこと。

## 本書について

本書では、インテル MKL とインテル・クラスタ MKL のルーチンおよび関数について説明する。

リファレンスの各セクションでは、一般的に 4 つの基本的なデータ型 (単精度実数、倍精度実数、単精度複素数、倍精度複素数) で使用するルーチンを 1 つのルーチングループにまとめて説明する。

それぞれのルーチングループの説明では、名前と機能の簡単な説明とともに、グループ内の各ルーチンで使用するデータ型それぞれについての呼び出しシーケンスまたは構文を紹介する。また、以下のセクションも含まれる。

説明	1 つ以上の式を例に挙げて、グループ内の各ルーチンが実行する機能を説明する。引数のデータ型は、グループ全体に共通する一般項で定義する。
入力パラメータ	入力パラメータそれぞれについてデータ型を定義する。例えば、 a      REAL (saxpy の場合 ) DOUBLE PRECISION (daxpy の場合 )
出力パラメータ	終了時に結果として得られるパラメータを列挙する。

## 本書の対象読者

本書では、数値演算に精通し、線形代数、数理統計学、およびフーリエ変換の原理や用語についての知識があるプログラマを対象にしている。

## 本書の構成

本書は、以下の各章と付録で構成される。

- 第 1 章 「[概要](#)」。インテル・マス・カーネル・ライブラリ・ソフトウェアを紹介するとともに、マニュアルの構成や表記上の規則について説明する。
- 第 2 章 「[BLAS ルーチンと スパース BLAS ルーチン](#)」。BLAS ならびにスパース BLAS の関数とルーチンを説明する。
- 第 3 章 「[LAPACK ルーチン：1 次方程式](#)」。連立 1 次方程式の解の算出と、これに関連する多数の計算タスク（三角因子分解、逆行列の計算、行列の条件数の推定など）を実行するための LAPACK ルーチンについて説明する。
- 第 4 章 「[LAPACK ルーチン：最小二乗問題および 固有値問題](#)」。最小二乗問題、標準ならびに一般化された固有値問題、特異値問題、およびシルベスター式を解くための LAPACK ルーチンを説明する。
- 第 5 章 「[LAPACK 補助ルーチンとユーティリティ・ルーチン](#)」。特定のサブタスク、共通のローレベル計算を実行する、補助およびユーティリティ LAPACK ルーチンについて説明する。
- 第 6 章 「[ScaLAPACK ルーチン](#)」。ScaLAPACK の計算、ドライバルーチン（インテル・クラスタ MKL のみ）について説明する。
- 第 7 章 「[ScaLAPACK 補助ルーチンとユーティリティ・ルーチン](#)」。ScaLAPACK の補助ルーチン（インテル・クラスタ MKL のみ）について説明する。
- 第 8 章 「[スパース・ソルバ・ルーチン](#)」。対称および対称構造スパース行列を計算する直接法スパース・ソルバ・ルーチンについて説明する。また、反復法スパース・ソルバ・ルーチンについても説明する。
- 第 9 章 「[ベクトル数学関数](#)」。ベクトル引数に対する基本的な数値演算関数を計算するための VML 関数について説明する。
- 第 10 章 「[統計関数](#)」。擬似乱数ベクトルを生成し、畳み込みや相関計算を実行する VSL 関数について説明する。
- 第 11 章 「[フーリエ変換関数](#)」。離散フーリエ変換を計算する多次元関数について説明する。また、クラスタ DFT 関数（インテル・クラスタ MKL のみ）および簡略化した高速フーリエ変換 (FFT) 関数についても説明する。FFT 関数は、インテル MKL では使用されず、後方互換のみを目的として存在している。そのため、FFT 関数に代わり、DFT 関数の使用が推奨される。
- 第 12 章 「[区間線形ソルバ](#)」。区間連立 1 次方程式および関連する問題を解くために使用できるルーチンについて説明する。
- 付録 A 「[線形ソルバの基礎](#)」。連立 1 次方程式を解くための線形代数での基本定義とアプローチについて簡単に説明する。また、区間演算の基本概念およびスパースデータ格納形式についても説明する。

- 付録 B 「[ルーチンおよび関数の 引数](#)」。BLAS ルーチンと VML 関数の主要な引数 (ベクトル引数と行列引数) について説明する。
- 付録 C 「[コード例](#)」。さまざまな MKL 関数やルーチンを呼び出す場合のコード例を掲載する (BLAS、スパースソルバ、DFT)。
- 付録 D 「[BLAS に対する CBLAS インターフェイス](#)」。BLAS に対する C インターフェイスを掲載する。
- 本書には、「[参考文献](#)」、「[用語集](#)」、および「[索引](#)」も含まれる。

## 表記の規則

本書では、以下の表記上の規則を使用する。

- ルーチン名の省略表記 (cungqr/zungqr の代わりに ?ungqr と表記)。
- 本文とコードを区別するためのフォント表記規則。

### ルーチン名の省略表記

省略表記を行うため、特定のルーチングループの名前のキャラクタ・コードは疑問符 (?) で表している。この疑問符は、特定の関数についての各データ型用の別形を表している。次に例を示す。

?swap                      ベクトル - ベクトルの ?swap ルーチンの 4 つのデータ型すべて、つまり sswap、dswap、cswap、zswap を表す。

### 字体の規則

次の字体の表記規則を使用する。

UPPERCASE COURIER	Fortran インターフェイスの入力パラメータと出力パラメータの説明で使用するデータ型。例えば、 CHARACTER*1
lowercase courier	コード例 a(k+i,j) = matrix(i,j) および C インターフェイスのデータ型。例えば、const float*。
lowercase courier mixed with UpperCase courier	C インターフェイスの関数名。 例えば、vmlSetMode
lowercase courier italic	引数やパラメータの説明における変数。例えば、 incx。

\*

コード例や式で乗算記号として使用。また、Fortran  
の構文で必要な箇所に使用。

# BLAS ルーチンと スパース BLAS ルーチン

## 2

本章では、インテル<sup>®</sup> マス・カーネル・ライブラリ (インテル<sup>®</sup> MKL) の BLAS ルーチンとスパース BLAS ルーチンについて説明する。ルーチンの説明は、BLAS の演算レベルによって以下のセクションに分かれている。

- [BLAS レベル 1 のルーチンと関数](#) (ベクトル - ベクトル演算)
- [BLAS レベル 2 のルーチン](#) (行列 - ベクトル演算)
- [BLAS レベル 3 のルーチン](#) (行列 - 行列演算)
- [スパース BLAS レベル 1 のルーチンと関数](#) (ベクトル - ベクトル演算)
- [スパース BLAS レベル 2 およびレベル 3](#) (行列 - ベクトル演算および行列 - 行列演算)。

各セクションでは、ルーチンと関数のグループを、例えば、?asum グループや ?axpy グループなどのように、アルファベット順に説明する。グループ名にある疑問符は、データ型を示す各種のキャラクタ・コード (s、d、c、z、あるいはそれらの組み合わせ) に対応している。次のページの「[ルーチン命名規則](#)」を参照のこと。

BLAS ルーチンまたは Sparse BLAS ルーチンでエラーが発生した場合は、エラー報告ルーチンの [xerbla](#) が呼び出される。エラー報告を見るためには、コード内に xerbla を組み込む必要がある。xerbla 用のソースコードのコピーが、マス・カーネル・ライブラリに含まれている。

BLAS レベル 1 のグループ i?amax と i?amin には、キャラクタ・コードの前に「i」が付いており、ベクトル成分のインデックスに対応している。これらのグループは、BLAS レベル 1 のセクションの最後に収録している。

## BLAS のルーチンと関数

### ルーチン命名規則

BLAS ルーチンの名前は、次の構造になっている。

`<character code> <name> <mod> ( )`

`<character code>` は、データ型を示すキャラクタ・コードである。

s	実数、単精度	c	複素数、単精度
d	実数、倍精度	z	複素数、倍精度

一部のルーチンや関数では、`sc` や `dz` などの組み合わせられたキャラクタ・コードを持つことができる。例えば、関数 `scasum` は、入力として複素配列を使用し、出力として実数値を返す。

`<name>` フィールドは、**BLAS** レベル 1 では、演算のタイプを示す。例えば、**BLAS** レベル 1 ルーチンの `?dot`、`?rot`、`?swap` は、それぞれベクトルの内積、ベクトルの回転、ベクトルの交換を計算する。

**BLAS** レベル 2 と 3 では、`<name>` は行列の引数のタイプを表す。

ge	一般行列
gb	一般帯行列
sy	対称行列
sp	対称行列 ( 圧縮格納形式 )
sb	対称帯行列
he	エルミート行列
hp	エルミート行列 ( 圧縮格納形式 )
hb	エルミート帯行列
tr	三角行列
tp	三角行列 ( 圧縮格納形式 )
tb	三角帯行列

`<mod>` フィールド ( 存在する場合 ) では、演算をさらに詳しく指示する。

**BLAS** レベル 1 のルーチン名の `<mod>` フィールドには、次のキャラクタが入れられる。

c	共役ベクトル
u	非共役ベクトル
g	Givens 回転

**BLAS** レベル 2 のルーチン名の `<mod>` フィールドには、次のキャラクタが入れられる。

mv	行列 - ベクトルの積
----	-------------

sv 行列 - ベクトル演算による連立 1 次方程式の解  
 r 階数 1 の行列の更新  
 r2 階数 2 の行列の更新

BLAS レベル 3 のルーチン名の *<mod>* フィールドには、次のキャラクタが入れられる。

mm 行列 - 行列の積  
 sm 行列 - 行列演算による連立 1 次方程式の解  
 rk 階数  $k$  の行列の更新  
 r2k 階数  $2k$  の行列の更新

以下に、BLAS ルーチン名を解釈する方法の例を示す。

*<d> <dot>* ddot: 倍精度実数型のベクトル - ベクトルの内積  
*<c> <dot> <c>* cdotc: 単精度複素共役のベクトル - ベクトルの内積  
*<sc> <asum>* scasum: 単精度複素数型を入力とし、単精度実数型を出力とする  
 ベクトルの成分の大きさの合計  
*<c> <dot> <u>* cdotu: 単精度複素非共役のベクトル - ベクトルの内積  
*<s> <ge> <mv>* sgemv: 単精度一般行列の行列 - ベクトルの積  
*<z> <tr> <mm>* ztrmm: 倍精度複素数型三角行列の行列 - 行列の積

スパース BLAS における命名規則は、BLAS レベル 1 の規則と似ている。  
 詳細は、「[命名規則](#)」を参照。

## Fortran-95 インターフェイス規則

BLAS および Sparse BLAS レベル 1 ルーチンに対する Fortran-95 インターフェイスは、それぞれの Fortran-77 ルーチン呼び出すラッパーを通して実装される。このインターフェイスは、形状引継ぎ配列や、より少ない引数で簡略化した BLAS および Sparse BLAS レベル 1 ルーチンの呼び出しを提供するオプション引数などの Fortran-95 の機能を使用する。

Fortran-95 インターフェイスで使用される主な規則を以下に示す。

- Fortran-95 呼び出しで使用される引数名は、一般的にそれぞれの標準 (Fortran-77) インターフェイスと同じである。ただし、ライブラリで使用される引数名の数を減らすための、簡略化された引数名を以下に示す。

標準引数名	Fortran-95 引数名
<i>ap</i>	<i>a</i>

これらの正式引数名の変更はプログラムの動作には影響を与えず、また統一命名規則に従うものである。

- 配列次元などの入力引数は Fortran-95 では不要であり、呼び出しシーケンスからスキップされる。配列次元は、必要な配列形状に正確に従うユーザデータから再構築される。  
また、引数は、呼び出しシーケンス内の他の引数の有無により、その値が完全に定義される場合にスキップされる。復元値はスキップされた引数にとってのみ意味のある値である。
- 引数 *incx* および *incy* はスキップされる。あらゆる場合において、これらの引数の値は 1 であるとみなされる。*incx* および *incy* に 1 以外の値を設定するには、対応する Fortran-95 の機能を使用して、インデックスの増分を実引数で直接指定する。Fortran-77 の呼び出しを使用した場合も、同様の効果が得られる。
- いくつかの標準引数は Fortran-95 インターフェイスではオプションとして宣言され、呼び出しシーケンスから省略される場合がある。以下の条件のいずれかを満たす場合は、引数をオプションとして宣言できる。
  1. 入力引数が、ごくわずかの設定可能な値を持つ場合、オプションとして宣言できる。これらの引数のデフォルト値は、通常、リストの最初の値として設定される。この規則における例外のすべてはルーチンの説明で明示的に述べる。
  2. 入力引数が自然デフォルト値を持つ場合は、オプションとして宣言できる。これらのオプション引数のデフォルト値は、自然デフォルト値に設定される。
- Fortran-95 呼び出し構文では、オプション引数を大括弧 ([]) で表現する。

オプション・パラメータの値の再構築に使用される具体的な規定は、各ルーチンごとに特有で、ルーチン仕様の最後の各「Fortran-95 ノート」で詳細を説明する。各ルーチンに特有な規定が省略されると、指定されたルーチンの Fortran-95 インターフェイスは、対応する Fortran-77 インターフェイスと変わらない。

このインターフェイスは、現バージョンのスパース BLAS レベル 2 およびレベル 3 のルーチンでは実装されていない点に注意する。これらのルーチンに対する Fortran-95 インターフェイスは、ルーチン仕様の最後の各「インターフェイス - Fortran-95」に収録されている。

### 行列の格納形式

BLAS ルーチンの行列引数では、次の格納形式を使用できる。

- フル格納では、行列  $A$  は 2 次元配列  $a$  に格納され、行列成分  $a_{ij}$  は配列成分  $a(i, j)$  に格納される。



- 圧縮格納では、対称行列、エルミート行列、または三角行列をコンパクトに格納できる。行列の上三角または下三角が 1 次元配列の各列に圧縮される。
- 帯格納では、帯行列が 2 次元配列にコンパクトに格納される。行列の各列は配列の対応する列に格納され、行列の各対角成分は配列の各行に格納する。

行列の格納形式の詳細は、付録 B の「[行列引数](#)」を参照。

## BLAS レベル 1 のルーチンと関数

BLAS レベル 1 には、ベクトル - ベクトル演算を実行するルーチンと関数が含まれる。表 2-1 に、BLAS レベル 1 のルーチンと関数のグループと、それらに関連するデータ型を示す。

表 2-1 BLAS レベル 1 のルーチングループおよびそのデータ型

ルーチンまたは関数のグループ	データ型	説明
<a href="#">?asum</a>	s, d, sc, dz	ベクトルの大きさの合計 (関数)
<a href="#">?axpy</a>	s, d, c, z	スカラー - ベクトルの積 (ルーチン)
<a href="#">?copy</a>	s, d, c, z	ベクトルのコピー (ルーチン)
<a href="#">?dot</a>	s, d	内積 (関数)
<a href="#">?sdot</a>	sd, d	精度を拡張した内積 (関数)
<a href="#">?dotc</a>	c, z	共役の内積 (関数)
<a href="#">?dotu</a>	c, z	非共役の内積 (関数)
<a href="#">?nrm2</a>	s, d, sc, dz	正規ベクトルまたはヌルベクトルのベクトル 2- ノルム (ユークリッド・ノルム) (関数)
<a href="#">?rot</a>	s, d, cs, zd	点の面回転 (ルーチン)
<a href="#">?rotq</a>	s, d, c, z	点の Givens 回転 (ルーチン)
<a href="#">?rotm</a>	s, d	点の変形面回転
<a href="#">?rotmq</a>	s, d	点の Givens 変形面回転
<a href="#">?scal</a>	s, d, c, z, cs, zd	ベクトルのスケーリング (ルーチン)
<a href="#">?swap</a>	s, d, c, z	ベクトル - ベクトルの交換 (ルーチン)
<a href="#">i?amax</a>	s, d, c, z	ベクトルの最大値、すなわちベクトルの絶対最大成分 (関数)。i は、ベクトル配列内のこの値に対するインデックス
<a href="#">i?amin</a>	s, d, c, z	ベクトルの最小値、すなわちベクトルの絶対最小成分 (関数)。i は、ベクトル配列内のこの値に対するインデックス
<a href="#">dcabs1</a>	d	倍精度複素数 z の絶対値

### ?asum

ベクトル成分の大きさの合計を計算する。

---

#### 構文

##### Fortran 77:

```
res = sASUM( N, X, INCX )
res = scASUM( N, X, INCX )
res = dASUM( N, X, INCX )
res = dzASUM( N, X, INCX )
```

##### Fortran 95:

```
res = asum(x)
```

#### 説明

?asum 関数は、ベクトル  $x$  に対してその成分の大きさの合計を計算する。複素ベクトルの場合は、成分の実数部の大きさに虚数部の大きさを加算した合計を計算する。

$$res = |Rex(1)| + |Imx(1)| + |Rex(2)| + |Imx(2)| + \dots + |Rex(n)| + |Imx(n)|$$

$x$  は次数  $n$  のベクトルである。

#### 入力パラメータ

$n$	INTEGER。ベクトル $x$ の次数を指定する。
$x$	REAL (sasum の場合 ) DOUBLE PRECISION (dasum の場合 ) COMPLEX (scasum の場合 ) DOUBLE COMPLEX (dzasum の場合 )  配列、次元は $(1 + (n-1)*abs(incx))$ 以上。
$incx$	INTEGER。 $x$ の成分に対する増分を指定する。

## 出力パラメータ

`res`        REAL (sasum の場合 )  
              DOUBLE PRECISION (dasum の場合 )  
              REAL (scasum の場合 )  
              DOUBLE PRECISION (dzasum の場合 )

すべての成分の実数部の大きさに虚数部の大きさを加算した合計が格納される。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[Fortran-95 インターフェイス規則](#)を参照のこと。

ルーチン `asum` のインターフェイスの詳細を以下に示す。

`x`            サイズ (*n*) の配列を格納する。

---

## ?axpy

ベクトル-スカラー積を計算し、その結果をベクトルに加える。

---

### 構文

#### Fortran 77:

```
CALL SAXPY( N, A, X, INCX, Y, INCY )
CALL DAXPY( N, A, X, INCX, Y, INCY )
CALL CAXPY( N, A, X, INCX, Y, INCY )
CALL ZAXPY( N, A, X, INCX, Y, INCY )
```

#### Fortran 95:

```
call axpy(x, y [,a])
```

### 説明

?axpy ルーチンは、次のように定義されるベクトル-ベクトル演算を実行する。

$$y := a * x + y$$

$a$  はスカラ、

$x$  と  $y$  は次数  $n$  のベクトルである。

### 入力パラメータ

$n$             INTEGER。ベクトル  $x$  と  $y$  の次数を指定する。

$a$             REAL (saxpy の場合)  
              DOUBLE PRECISION (daxpy の場合)  
              COMPLEX (caxpy の場合)  
              DOUBLE COMPLEX (zaxpy の場合)  
  
              スカラ  $a$  を指定する。

$x$             REAL (saxpy の場合)  
              DOUBLE PRECISION (daxpy の場合)  
              COMPLEX (caxpy の場合)  
              DOUBLE COMPLEX (zaxpy の場合)  
  
              配列、次元は  $(1 + (n-1) * \text{abs}(\text{incx}))$  以上。

$\text{incx}$         INTEGER。  $x$  の成分に対する増分を指定する。

$y$             REAL (saxpy の場合)  
              DOUBLE PRECISION (daxpy の場合)  
              COMPLEX (caxpy の場合)  
              DOUBLE COMPLEX (zaxpy の場合)  
  
              配列、次元は  $(1 + (n-1) * \text{abs}(\text{incy}))$  以上。

$\text{incy}$         INTEGER。  $y$  の成分に対する増分を指定する。

### 出力パラメータ

$y$             更新されたベクトル  $y$  が格納される。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[Fortran-95 インターフェイス規則](#)を参照のこと。

ルーチン axpy のインターフェイスの詳細を以下に示す。

$x$             サイズ ( $n$ ) の配列を格納する。

$y$             サイズ ( $n$ ) の配列を格納する。  
 $a$             デフォルト値は '1' である。

---

## ?copy

ベクトルを別のベクトルにコピーする。

---

### 構文

#### Fortran 77:

```
CALL sCOPY( N, X, INCX, Y, INCY )  
CALL dCOPY( N, X, INCX, Y, INCY )  
CALL cCOPY( N, X, INCX, Y, INCY )  
CALL zCOPY( N, X, INCX, Y, INCY )
```

#### Fortran 95:

```
call copy(x, y)
```

### 説明

?copy ルーチンは、次のように定義されるベクトル - ベクトル演算を実行する。

$y = x$

$x$  と  $y$  はベクトルである。

### 入力パラメータ

$n$             INTEGER。ベクトル  $x$  と  $y$  の次数を指定する。  
 $x$             REAL (scopy の場合 )  
              DOUBLE PRECISION (dcopy の場合 )  
              COMPLEX (ccopy の場合 )  
              DOUBLE COMPLEX (zcopy の場合 )  
              配列、次元は  $(1 + (n-1) * \text{abs}(\text{incx}))$  以上。  
 $\text{incx}$         INTEGER。  $x$  の成分に対する増分を指定する。

$y$  REAL (scopy の場合)  
DOUBLE PRECISION (dcopy の場合)  
COMPLEX (ccopy の場合)  
DOUBLE COMPLEX (zcopy の場合)  
配列、次元は  $(1 + (n-1) * \text{abs}(\text{incy}))$  以上。  
 $\text{incy}$  INTEGER。  $y$  の成分に対する増分を指定する。

### 出力パラメータ

$y$   $n$  が正の場合は、ベクトル  $x$  のコピーが格納される。そうでない場合には、パラメータは変更されない。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[Fortran-95 インターフェイス規則](#)を参照のこと。

ルーチン copy のインターフェイスの詳細を以下に示す。

$x$  長さ ( $n$ ) のベクトルを格納する。  
 $y$  長さ ( $n$ ) のベクトルを格納する。

---

## ?dot

ベクトル-ベクトルの内積を計算する。

---

### 構文

#### Fortran 77:

```
res = sDOT( N, X, INCX, Y, INCY )  
res = dDOT( N, X, INCX, Y, INCY )
```

#### Fortran 95:

```
res = dot(x, y)
```

## 説明

?dot 関数は、次のように定義されるベクトル - ベクトルの縮退演算を実行する。

$$res = \sum (x * y)$$

$x$  と  $y$  はベクトルである。

## 入力パラメータ

$n$             INTEGER。ベクトル  $x$  と  $y$  の次数を指定する。

$x$             REAL (sdot の場合)  
              DOUBLE PRECISION (ddot の場合)

配列、次元は  $(1+(n-1)*abs(incx))$  以上。

$incx$         INTEGER。  $x$  の成分に対する増分を指定する。

$y$             REAL (sdot の場合)  
              DOUBLE PRECISION (ddot の場合)

配列、次元は  $(1+(n-1)*abs(incy))$  以上。

$incy$         INTEGER。  $y$  の成分に対する増分を指定する。

## 出力パラメータ

$res$         REAL (sdot の場合)  
              DOUBLE PRECISION (ddot の場合)

$n$  が正の場合は、  $x$  と  $y$  の内積の結果が格納される。そうでない場合には、 $res$  には 0 が格納される。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[Fortran-95 インターフェイス規則](#)を参照のこと。

ルーチン dot のインターフェイスの詳細を以下に示す。

$x$             長さ ( $n$ ) のベクトルを格納する。

$y$             長さ ( $n$ ) のベクトルを格納する。

### ?sdot

ベクトル - ベクトルの内積を拡張精度で計算する。

---

#### 構文

##### Fortran 77:

```
res = sdsDOT( N, sb, sX, INCX, sY, INCY )  
res = dsdot( N, sX, INCX, sY, INCY )
```

##### Fortran 95:

```
res = sdot(sx, sy)  
res = sdot(sx, sy, sb)
```

#### 説明

?sdot 関数は 2 つのベクトルの内積を拡張精度で計算する。どちらの関数も中間結果は拡張精度で蓄積されるが、sdsdot 関数は最終結果を単精度で出力し、dsdot は倍精度で出力する。また、sdsdot 関数では内積にスカラ値 *sb* が加算される。

#### 入力パラメータ

*n*            INTEGER。入力ベクトル *sx* と *sy* 内の成分の数を指定する。

*sb*            REAL。内積に加算する単精度スカラ (sdsdot 関数のみ)。

*sx, sy*        REAL。  
配列、次元はそれぞれ  $(1+(n-1)*abs(incx))$ 、 $(1+(n-1)*abs(incy))$  以上。入力単精度ベクトルが格納される。

*incx*         INTEGER。*sx* の成分に対する増分を指定する。

*incy*         INTEGER。*sy* の成分に対する増分を指定する。

#### 出力パラメータ

*res*            REAL (sdsdot の場合)  
                DOUBLE PRECISION (sdot の場合)

*n* が正の場合は *sx* と *sy* の内積の結果が格納される (sbsdot では *sb* が加算されている)。N が正でない場合は、sdsdot では *sb*、dsdot では 0 が *res* に格納される。



## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[Fortran-95 インターフェイス規則](#)を参照のこと。

ルーチン `sdot` のインターフェイスの詳細を以下に示す。

`sx`           長さ ( $n$ ) のベクトルを格納する。

`sy`           長さ ( $n$ ) のベクトルを格納する。



**注:** スカラ・パラメータ  $sb$  は、異なる精度の最終結果を出力する関数を区別するため、関数 `sdot` に対する Fortran-95 インターフェイスで、必須パラメータとして定義される。

## ?dotc

共役ベクトルと別のベクトルの内積を計算する。

### 構文

#### Fortran 77:

```
res = CDOTC( N, X, INCX, Y, INCY )
```

```
res = ZDOTC( N, X, INCX, Y, INCY )
```

#### Fortran 95:

```
res = dotc(x, y)
```

### 説明

?dotc 関数は、次のように定義されるベクトル - ベクトル演算を実行する。

$$res = \sum (conjg(x)*y)$$

$x$  と  $y$  は  $n$  個の成分を持つベクトルである。

### 入力パラメータ

<i>n</i>	INTEGER。ベクトル <i>x</i> と <i>y</i> の次数を指定する。
<i>x</i>	COMPLEX (cdotc の場合) DOUBLE COMPLEX (zdotc の場合) 配列、次元は $(1 + (n-1) * \text{abs}(\text{incx}))$ 以上。
<i>incx</i>	INTEGER。 <i>x</i> の成分に対する増分を指定する。
<i>y</i>	COMPLEX (cdotc の場合) DOUBLE COMPLEX (zdotc の場合) 配列、次元は $(1 + (n-1) * \text{abs}(\text{incy}))$ 以上。
<i>incy</i>	INTEGER。 <i>y</i> の成分に対する増分を指定する。

### 出力パラメータ

<i>res</i>	COMPLEX (cdotc の場合) DOUBLE COMPLEX (zdotc の場合) <i>n</i> が正の場合は、共役の <i>x</i> と非共役の <i>y</i> の内積の結果が格納される。そうでない場合には、 <i>res</i> には 0 が格納される。
------------	--

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[Fortran-95 インターフェイス規則](#)を参照のこと。

ルーチン dotc のインターフェイスの詳細を以下に示す。

<i>x</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>y</i>	長さ ( <i>n</i> ) のベクトルを格納する。

## ?dotu

ベクトル - ベクトルの内積を計算する。

### 構文

#### Fortran 77:

```
res = CDOTU( N, X, INCX, Y, INCY )
res = ZDOTU( N, X, INCX, Y, INCY )
```

#### Fortran 95:

```
res = dotu(x, y)
```

### 説明

?dotu 関数は、次のように定義されるベクトル - ベクトルの縮退演算を実行する。

$$res = \sum (x*y)$$

$x$  と  $y$  は  $n$  個の成分を持つベクトルである。

### 入力パラメータ

$n$             INTEGER。ベクトル  $x$  と  $y$  の次数を指定する。

$x$             COMPLEX (cdotu の場合 )  
               DOUBLE COMPLEX (zdotu の場合 )  
               配列、次元は  $(1 + (n-1)*abs(incx))$  以上。

$incx$         INTEGER。  $x$  の成分に対する増分を指定する。

$y$             COMPLEX (cdotu の場合 )  
               DOUBLE COMPLEX (zdotu の場合 )  
               配列、次元は  $(1 + (n-1)*abs(incy))$  以上。

$incy$         INTEGER。  $y$  の成分に対する増分を指定する。

### 出力パラメータ

$res$         COMPLEX (cdotu の場合 )  
               DOUBLE COMPLEX (zdotu の場合 )

$n$  が正の場合は、 $x$  と  $y$  の内積の結果が格納される。そうでない場合には、 $res$  には 0 が格納される。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[Fortran-95 インターフェイス規則](#)を参照のこと。

ルーチン `dotu` のインターフェイスの詳細を以下に示す。

$x$             長さ ( $n$ ) のベクトルを格納する。  
 $y$             長さ ( $n$ ) のベクトルを格納する。

---

## ?nrm2

ベクトルのユークリッド・ノルムを計算する。

---

### 構文

#### Fortran 77:

```
res = sNRM2( N, X, INCX )  
res = DNRM2( N, X, INCX )  
res = scNRM2( N, X, INCX )  
res = DzNRM2( N, X, INCX )
```

#### Fortran 95:

```
res = nrm2(x)
```

### 説明

?nrm2 関数は、次のように定義されるベクトルの縮退演算を実行する。

```
res = ||x||
```

$x$  はベクトル、

$res$  は  $x$  の成分のユークリッド・ノルムが格納される値である。

### 入力パラメータ

*n*            INTEGER。ベクトル *x* の次数を指定する。

*x*            REAL (snrm2 の場合)  
               DOUBLE PRECISION (dnrm2 の場合)  
               COMPLEX (scnrm2 の場合)  
               DOUBLE COMPLEX (dznrm2 の場合)  
               配列、次元は  $(1 + (n-1) * \text{abs}(\text{incx}))$  以上。

*incx*        INTEGER。 *x* の成分に対する増分を指定する。

### 出力パラメータ

*res*        REAL (snrm2 の場合)  
               DOUBLE PRECISION (dnrm2 の場合)  
               REAL (scnrm2 の場合)  
               DOUBLE PRECISION (dznrm2 の場合)  
               ベクトル *x* のユークリッド・ノルムが格納される。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[Fortran-95 インターフェイス規則](#)を参照のこと。

ルーチン nrm2 のインターフェイスの詳細を以下に示す。

*x*            長さ (*n*) のベクトルを格納する。

## ?rot

面における点の回転を実行する。

### 構文

#### Fortran 77:

```
CALL SROT( N, X, INCX, Y, INCY, C, S )
CALL dROT( N, X, INCX, Y, INCY, C, S )
```

```
CALL CSRROT( N, X, INCX, Y, INCY, C, S )
```

```
CALL zdROT( N, X, INCX, Y, INCY, C, S )
```

### Fortran 95:

```
call rot(x, y [,c] [,s])
```

### 説明

2つの複素ベクトル  $x$  と  $y$  が与えられたときに、これらのベクトルの各ベクトル成分が次のように置き換えられる。

$$x(i) = c \cdot x(i) + s \cdot y(i)$$
$$y(i) = c \cdot y(i) - s \cdot x(i)$$

### 入力パラメータ

$n$             INTEGER。ベクトル  $x$  と  $y$  の次数を指定する。

$x$             REAL (srot の場合)  
              DOUBLE PRECISION (drot の場合)  
              COMPLEX (csrot の場合)  
              DOUBLE COMPLEX (zdrot の場合)  
  
              配列、次元は  $(1 + (n-1) \cdot \text{abs}(\text{incx}))$  以上。

$\text{incx}$         INTEGER。  $x$  の成分に対する増分を指定する。

$y$             REAL (srot の場合)  
              DOUBLE PRECISION (drot の場合)  
              COMPLEX (csrot の場合)  
              DOUBLE COMPLEX (zdrot の場合)  
  
              配列、次元は  $(1 + (n-1) \cdot \text{abs}(\text{incy}))$  以上。

$\text{incy}$         INTEGER。  $y$  の成分に対する増分を指定する。

$c$             REAL (srot の場合)  
              DOUBLE PRECISION (drot の場合)  
              REAL (csrot の場合)  
              DOUBLE PRECISION (zdrot の場合)  
  
              スカラ。

$s$  REAL (srot の場合)  
 DOUBLE PRECISION (drot の場合)  
 REAL (csrot の場合)  
 DOUBLE PRECISION (zdrot の場合)  
 スカラ。

### 出力パラメータ

$x$  各成分は、 $c*x + s*y$  で置き換えられる。  
 $y$  各成分は、 $c*y - s*x$  で置き換えられる。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[Fortran-95 インターフェイス規則](#)を参照のこと。

ルーチン rot のインターフェイスの詳細を以下に示す。

$x$  長さ ( $n$ ) のベクトルを格納する。  
 $y$  長さ ( $n$ ) のベクトルを格納する。  
 $c$  デフォルト値は '1' である。  
 $s$  デフォルト値は '1' である。

## ?rotg

*Givens* 回転に対するパラメータを計算する。

### 構文

#### Fortran 77:

```
CALL sROTG( A, B, C, S )
CALL DROTG( A, B, C, S )
CALL cROTG( A, B, C, S )
CALL zROTG( A, B, C, S )
```

### Fortran 95:

```
call rotg(a, b, c, s)
```

### 説明

点  $p$  の直交座標  $(a, b)$  が与えられたときに、これらのルーチンはその点の  $y$  座標をゼロにする Givens 回転に関連付けられるパラメータ  $a$ 、 $b$ 、 $c$ 、 $s$  を返す。

より精度の高い LAPACK バージョンの [zlartg](#) を参照。

### 入力パラメータ

- $a$       REAL (srotg の場合)  
         DOUBLE PRECISION (drotg の場合)  
         COMPLEX (crotg の場合)  
         DOUBLE COMPLEX (zrotg の場合)  
  
         点  $p$  の  $x$  座標を与える。
- $b$       REAL (srotg の場合)  
         DOUBLE PRECISION (drotg の場合)  
         COMPLEX (crotg の場合)  
         DOUBLE COMPLEX (zrotg の場合)  
  
         点  $p$  の  $y$  座標を与える。

### 出力パラメータ

- $a$       Givens 回転に関連付けられたパラメータ  $r$  が格納される。
- $b$       Givens 回転に関連付けられたパラメータ  $z$  が格納される。
- $c$       REAL (srotg の場合)  
         DOUBLE PRECISION (drotg の場合)  
         REAL (crotg の場合)  
         DOUBLE PRECISION (zrotg の場合)  
  
         Givens 回転に関連付けられたパラメータ  $c$  が格納される。
- $s$       REAL (srotg の場合)  
         DOUBLE PRECISION (drotg の場合)  
         COMPLEX (crotg の場合)  
         DOUBLE COMPLEX (zrotg の場合)  
  
         Givens 回転に関連付けられたパラメータ  $s$  が格納される。



## ?rotm

変形面における点の回転を実行する。

### 構文

#### Fortran 77:

```
CALL SROTm( N, X, INCX, Y, INCY, param )
CALL dROTm( N, X, INCX, Y, INCY, param )
```

#### Fortran 95:

```
call rotm(x, y [,param])
```

### 説明

2つの複素ベクトル  $x$  と  $y$  が与えられたときに、これらのベクトルの各ベクトル成分が次のように置き換えられる。

$$x(i) = H \cdot x(i) + H \cdot y(i)$$

$$y(i) = H \cdot y(i) - H \cdot x(i)$$

$H$  は変形 Givens 変換行列で、その値は  $param(2)$  から  $param(5)$  の配列に格納される。  
 $param$  引数の説明を参照のこと。

### 入力パラメータ

$n$	INTEGER。ベクトル $x$ と $y$ の次数を指定する。
$x$	REAL (srotm の場合) DOUBLE PRECISION (drotm の場合) 配列、次元は $(1 + (n-1) \cdot \text{abs}(\text{incx}))$ 以上。
$\text{incx}$	INTEGER。 $x$ の成分に対する増分を指定する。
$y$	REAL (srotm の場合) DOUBLE PRECISION (drotm の場合) 配列、次元は $(1 + (n-1) \cdot \text{abs}(\text{incy}))$ 以上。
$\text{incy}$	INTEGER。 $y$ の成分に対する増分を指定する。

*param*      REAL (srotm の場合)  
              DOUBLE PRECISION (drotm の場合)  
              配列、次元は 5。  
  
              *param* 配列の成分は次のようになる。  
  
              *param*(1) にはスイッチ、*flag* が格納される。  
              *param*(2-5) にはそれぞれ、配列 *H* の成分である *h11*、*h21*、*h12*、*h22* が格納される。  
  
              *flag* の値により、*H* の成分は次のとおりに設定される。

$$flag = -1.: H = \begin{bmatrix} h11 & h12 \\ h21 & h22 \end{bmatrix}$$

$$flag = 0.: H = \begin{bmatrix} 1. & h12 \\ h21 & 1. \end{bmatrix}$$

$$flag = 1.: H = \begin{bmatrix} h11 & 1. \\ -1. & h22 \end{bmatrix}$$

$$flag = -2.: H = \begin{bmatrix} 1. & 0. \\ 0. & 1. \end{bmatrix}$$

上記で行列の成分が 1、-1、0 の場合には、*flag* の最後の 3 つの値に基づくものとみなされ、実際には *param* ベクトルにロードされない。

### 出力パラメータ

*x*           それぞれの成分が  $h11*x + h12*y$  で置き換えられる。  
*y*           それぞれの成分が  $h21*x + h22*y$  で置き換えられる。  
*H*           更新された Givens 変換行列。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[Fortran-95 インターフェイス規則](#)を参照のこと。

ルーチン *rotm* のインターフェイスの詳細を以下に示す。

*x*           長さ (*n*) のベクトルを格納する。

$y$            長さ ( $n$ ) のベクトルを格納する。  
 $param$         $param(1)$  のデフォルト値は -2 である。

## ?rotmg

*Givens* 回転に対する変形パラメータを計算する。

### 構文

#### Fortran 77:

```
CALL sROTmG( d1, d2, x1, y1, param )
CALL DROTmG( d1, d2, x1, y1, param )
```

#### Fortran 95:

```
call rotmg(x1, y1, param [,d1] [d2])
```

### 説明

入力ベクトルの直交座標 ( $x1, y1$ ) が与えられたときに、これらのルーチンは、結果として得られるベクトルの  $y$  成分をゼロにする変形 *Givens* 変換行列  $H$  の成分を計算する。

$$\begin{bmatrix} x \\ 0 \end{bmatrix} = H \begin{bmatrix} x1 \\ y1 \end{bmatrix}$$

### 入力パラメータ

$d1$            REAL (srotmg の場合)  
               DOUBLE PRECISION (drotmg の場合)  
               入力ベクトル ( $\sqrt{d1}x1$ ) の  $x$  座標に対する更新されたスケール係数を与える。

$d2$            REAL (srotmg の場合)  
               DOUBLE PRECISION (drotmg の場合)  
               入力ベクトル ( $\sqrt{d2}y1$ ) の  $y$  座標に対する更新されたスケール係数を与える。

$x1$            REAL (srotmg の場合)  
               DOUBLE PRECISION (drotmg の場合)  
               入力ベクトルの回転された  $x$  座標を与える。

*y1*            REAL (srotmg の場合)  
                DOUBLE PRECISION (drotmg の場合)  
                入力ベクトルの *y* 座標を与える。

### 出力パラメータ

*param*        REAL (srotmg の場合)  
                DOUBLE PRECISION (drotmg の場合)  
                配列、次元は 5。  
  
*param* 配列の成分は次のようになる。  
  
*param*(1) にはスイッチ、*flag* が格納される。  
*param*(2-5) にはそれぞれ、配列 *H* の成分である *h11*、*h21*、*h12*、*h22* が格納される。

*flag* の値により、*H* の成分は次のとおりに設定される。

$$flag = -1.: H = \begin{bmatrix} h11 & h12 \\ h21 & h22 \end{bmatrix}$$

$$flag = 0.: H = \begin{bmatrix} 1. & h12 \\ h21 & 1. \end{bmatrix}$$

$$flag = 1.: H = \begin{bmatrix} h11 & 1. \\ -1. & h22 \end{bmatrix}$$

$$flag = -2.: H = \begin{bmatrix} 1. & 0. \\ 0. & 1. \end{bmatrix}$$

上記で行列の成分が 1、-1、0 の場合には、*flag* の最後の 3 つの値に基づくものとみなされ、実際には *param* ベクトルにロードされない。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[Fortran-95 インターフェイス規則](#)を参照のこと。

ルーチン *rotmg* のインターフェイスの詳細を以下に示す。

*d1*            デフォルト値は '1.' である。

*d2*            デフォルト値は '1.' である。

---

## ?scal

ベクトルとスカラの積を計算する。

---

### 構文

#### Fortran 77:

```
CALL sSCAL( N, A, X, INCX )
CALL dSCAL( N, A, X, INCX )
CALL cSCAL( N, A, X, INCX )
CALL zSCAL( N, A, X, INCX )
CALL csSCAL( N, A, X, INCX )
CALL zdSCAL( N, A, X, INCX )
```

#### Fortran 95:

```
call scal(x, a)
```

### 説明

?scal ルーチンは、次のように定義されるベクトル - ベクトル演算を実行する。

$$x = a * x$$

$a$  はスカラ、 $x$  は  $n$  個の成分を持つベクトルである。

### 入力パラメータ

$n$	INTEGER。ベクトル $x$ の次数を指定する。
$a$	REAL(sscal、csscal の場合) DOUBLE PRECISION(dscal、zdscal の場合) REAL(cscal の場合) DOUBLE PRECISION(zscal の場合) スカラ $a$ を指定する。
$x$	REAL(sscal の場合) DOUBLE PRECISION(dscal の場合) COMPLEX(cscal、csscal の場合) DOUBLE COMPLEX(zscal、csscal の場合) 配列、次元は $(1 + (n-1) * \text{abs}(\text{incx}))$ 以上。

*incx*          INTEGER。 *x* の成分に対する増分を指定する。

### 出力パラメータ

*x*              更新されたベクトル *x* によって上書きされる。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[Fortran-95 インターフェイス規則](#)を参照のこと。

ルーチン `scal` のインターフェイスの詳細を以下に示す。

*x*              長さ (*n*) のベクトルを格納する。



**注：**スカラ・パラメータ *a* は、異なるデータ型の演算を行う関数を区別するため、関数 `scal` に対する Fortran-95 インターフェイスで、必須パラメータとして定義される。

---

## ?swap

ベクトルを別のベクトルと交換する。

---

### 構文

#### Fortran 77:

```
CALL sSWAP( N, X, INCX, Y, INCY )
```

```
CALL dSWAP( N, X, INCX, Y, INCY )
```

```
CALL cSWAP( N, X, INCX, Y, INCY )
```

```
CALL zSWAP( N, X, INCX, Y, INCY )
```

#### Fortran 95:

```
call swap(x, y)
```

## 説明

2つの複素ベクトル  $x$  と  $y$  が与えられたときに、`?swap` ルーチンは、相互に置き換えられたベクトル  $y$  と  $x$  を返す。

## 入力パラメータ

$n$             INTEGER。ベクトル  $x$  と  $y$  の次数を指定する。

$x$             REAL (`sswap` の場合)  
               DOUBLE PRECISION (`dswap` の場合)  
               COMPLEX (`cswap` の場合)  
               DOUBLE COMPLEX (`zswap` の場合)

配列、次元は  $(1 + (n-1) * \text{abs}(\text{incx}))$  以上。

$\text{incx}$         INTEGER。  $x$  の成分に対する増分を指定する。

$y$             REAL (`sswap` の場合)  
               DOUBLE PRECISION (`dswap` の場合)  
               COMPLEX (`cswap` の場合)  
               DOUBLE COMPLEX (`zswap` の場合)

配列、次元は  $(1 + (n-1) * \text{abs}(\text{incy}))$  以上。

$\text{incy}$         INTEGER。  $y$  の成分に対する増分を指定する。

## 出力パラメータ

$x$             結果として得られるベクトル  $x$  が格納される。

$y$             結果として得られるベクトル  $y$  が格納される。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[Fortran-95 インターフェイス規則](#)を参照のこと。

ルーチン `swap` のインターフェイスの詳細を以下に示す。

$x$             長さ ( $n$ ) のベクトルを格納する。

$y$             長さ ( $n$ ) のベクトルを格納する。

### i?amax

最大絶対値を持つベクトルの成分を検出する。

---

#### 構文

##### Fortran 77:

```
index = IsAMAX( N, X, INCX )  
index = IdAMAX( N, X, INCX )  
index = IcAMAX( N, X, INCX )  
index = IzAMAX( N, X, INCX )
```

##### Fortran 95:

```
res = iamax(x)
```

#### 説明

ベクトル  $x$  が与えられたときに、`i?amax` 関数は最大絶対値を持つベクトル成分  $x(i)$  の位置を返す。複素ベクトルの場合は、この関数は最大合計  $|\text{Re } x(i)| + |\text{Im } x(i)|$  を持つ成分の位置を返す。

$n$  が正でない場合は、0 が返される。

同じ最大絶対値を持つベクトル成分が 2 つ以上検出された場合は、最初に検出された成分のインデックスが返される。

#### 入力パラメータ

$n$             INTEGER。ベクトル  $x$  の次数を指定する。

$x$             REAL (`isamax` の場合)  
              DOUBLE PRECISION (`idamax` の場合)  
              COMPLEX (`icamax` の場合)  
              DOUBLE COMPLEX (`izamax` の場合)

              配列、次元は  $(1+(n-1)*\text{abs}(\text{incx}))$  以上。

$\text{incx}$         INTEGER。 $x$  の成分に対する増分を指定する。

#### 出力パラメータ

$\text{index}$         INTEGER。最大絶対値を持つベクトル成分  $x$  の位置が格納される。



## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[Fortran-95 インターフェイス規則](#)を参照のこと。

ルーチン `amax` のインターフェイスの詳細を以下に示す。

`x`           長さ ( $n$ ) のベクトルを格納する。

---

## i?amin

最小絶対値を持つベクトル成分を検出する。

---

### 構文

#### Fortran 77:

```
index = IsAMIN( N, X, INCX )  
index = IdAMIN( N, X, INCX )  
index = IcAMIN( N, X, INCX )  
index = IzAMIN( N, X, INCX )
```

#### Fortran 95:

```
res = iamin(x)
```

### 説明

ベクトル  $x$  が与えられたときに、`i?amin` 関数は最小絶対値を持つベクトル成分  $x(i)$  の位置を返す。複素ベクトルの場合は、この関数は最小合計  $|\text{Re}x(i)| + |\text{Im}x(i)|$  を持つ成分の位置を返す。

$n$  が正でない場合は、0 が返される。

同じ最小絶対値を持つベクトル成分が 2 つ以上検出された場合は、最初に検出した成分のインデックスが返される。

### 入力パラメータ

$n$            INTEGER。  $n$  にはベクトル  $x$  の次数を指定する。

$x$             REAL (isamin の場合)  
              DOUBLE PRECISION (idamin の場合)  
              COMPLEX (icamin の場合)  
              DOUBLE COMPLEX (izamin の場合)  
              配列、次元は  $(1+(n-1)*abs(incx))$  以上。  
 $incx$         INTEGER。  $x$  の成分に対する増分を指定する。

### 出力パラメータ

$index$         INTEGER。 最小絶対値を持つベクトル成分  $x$  の位置が格納される。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[Fortran-95 インターフェイス規則](#)を参照のこと。

ルーチン amin のインターフェイスの詳細を以下に示す。

$x$             長さ ( $n$ ) のベクトルを格納する。

---

## dcabs1

倍精度複素数の絶対値を計算する。

---

### 構文

#### Fortran 77:

```
res = dcabs1(z)
```

#### Fortran 95:

```
res = dcabs1(z)
```

### 説明

dcabs1 は、BLAS レベル 1 ルーチンの補助ルーチンである。この関数は、次のように定義される演算を実行する。

$$res = |\operatorname{Re}(z)| + |\operatorname{Im}(z)|$$

$z$  はスカラー、 $res$  は倍精度複素数  $z$  の絶対値が格納される値である。

### 入力パラメータ

$z$                     DOUBLE COMPLEX スカラー。

### 出力パラメータ

$res$                   DOUBLE PRECISION。倍精度複素数  $z$  の絶対値が格納される。

## BLAS レベル 2 のルーチン

このセクションでは、行列 - ベクトル演算を実行する BLAS レベル 2 のルーチンについて説明する。表 2-2 に、BLAS レベル 2 のルーチンのグループと、それらに関連するデータ型を示す。

表 2-2                  BLAS レベル 2 のルーチングループおよびそのデータ型

ルーチン グループ	データ型	説明
<a href="#">?gbmv</a>	s, d, c, z	一般帯行列での行列 - ベクトルの積
<a href="#">?gemv</a>	s, d, c, z	一般行列での行列 - ベクトルの積
<a href="#">?ger</a>	s, d	一般行列の階数 1 の更新
<a href="#">?gerc</a>	c, z	共役一般行列の階数 1 の更新
<a href="#">?geru</a>	c, z	非共役一般行列の階数 1 の更新
<a href="#">?hbm</a>	c, z	エルミート帯行列での行列 - ベクトルの積
<a href="#">?hemv</a>	c, z	エルミート行列での行列 - ベクトルの積
<a href="#">?her</a>	c, z	エルミート行列の階数 1 の更新
<a href="#">?her2</a>	c, z	エルミート行列の階数 2 の更新
<a href="#">?hpmv</a>	c, z	圧縮形式のエルミート行列での行列 - ベクトルの積
<a href="#">?hpr</a>	c, z	圧縮形式のエルミート行列の階数 1 の更新
<a href="#">?hpr2</a>	c, z	圧縮形式のエルミート行列の階数 2 の更新
<a href="#">?sbmv</a>	s, d	対称帯行列での行列 - ベクトルの積
<a href="#">?spmv</a>	s, d	圧縮形式の対称行列での行列 - ベクトルの積
<a href="#">?spr</a>	s, d	圧縮形式の対称行列の階数 1 の更新
<a href="#">?spr2</a>	s, d	圧縮形式の対称行列の階数 2 の更新
<a href="#">?symv</a>	s, d	対称行列での行列 - ベクトルの積
<a href="#">?syr</a>	s, d	対称行列の階数 1 の更新

表 2-2 BLAS レベル 2 のルーチングループおよびそのデータ型 ( 続き )

ルーチン グループ	データ型	説明
<a href="#">?syr2</a>	s, d	対称行列の階数 2 の更新
<a href="#">?tbmv</a>	s, d, c, z	三角帯行列での行列 - ベクトルの積
<a href="#">?tbsv</a>	s, d, c, z	三角帯行列の 1 次方程式の解
<a href="#">?tpmv</a>	s, d, c, z	圧縮形式の三角行列での行列 - ベクトルの積
<a href="#">?tpsv</a>	s, d, c, z	圧縮形式の三角行列の 1 次方程式の解
<a href="#">?trmv</a>	s, d, c, z	三角行列での行列 - ベクトルの積
<a href="#">?trsv</a>	s, d, c, z	三角行列の 1 次方程式の解

## ?gbmv

一般帯行列を使用して行列 - ベクトルの積を計算する。

### 構文

#### Fortran 77:

```
call sgbmv( trans, m, n, kl, ku, alpha, a, lda, x, inxc, beta, y, incy )
CALL dGBMV( TRANS, M, N, KL, KU, ALPHA, A, LDA, X, INCX, BETA, Y, INCY )
CALL CGBMV( TRANS, M, N, KL, KU, ALPHA, A, LDA, X, INCX, BETA, Y, INCY )
CALL ZGBMV( TRANS, M, N, KL, KU, ALPHA, A, LDA, X, INCX, BETA, Y, INCY )
```

#### Fortran 95:

```
call gbmw(a, x, y [,kl] [,m] [,alpha] [,beta] [,trans])
```

### 説明

?gbmv ルーチンは、次のように定義される行列 - ベクトル演算を実行する。

$$y := \alpha * a * x + \beta * y$$

または

$$y := \alpha * a' * x + \beta * y$$

または

$y := \alpha * \text{conjg}(a') * x + \beta * y$

$\alpha$  と  $\beta$  は、スカラである。

$x$  と  $y$  は、ベクトルである。

$a$  は、 $k_l$  個の劣対角成分と  $k_u$  個の優対角成分を持つ  $m \times n$  の帯行列である。

## 入力パラメータ

$trans$  CHARACTER\*1。次に示すように、実行する演算を指定する。

$trans$ の値	実行する演算
N または n	$y := \alpha * a * x + \beta * y$
T または t	$y := \alpha * a' * x + \beta * y$
C または c	$y := \alpha * \text{conjg}(a') * x + \beta * y$

$m$  INTEGER。行列  $a$  の行数を指定する。 $m$  の値は、ゼロ以上でなければならない。

$n$  INTEGER。行列  $a$  の列数を指定する。 $n$  の値は、ゼロ以上でなければならない。

$k_l$  INTEGER。行列  $a$  の劣対角成分の数を指定する。 $k_l$  の値は、 $0 \leq k_l$  を満たしていなければならない。

$k_u$  INTEGER。行列  $a$  の優対角成分の数を指定する。 $k_u$  の値は、 $0 \leq k_u$  を満たしていなければならない。

$\alpha$  REAL (sgbmV の場合)  
DOUBLE PRECISION (dgbmv の場合)  
COMPLEX (cgbmv の場合)  
DOUBLE COMPLEX (zgbmv の場合)  
スカラ  $\alpha$  を指定する。

$a$  REAL (sgbmV の場合)  
DOUBLE PRECISION (dgbmv の場合)  
COMPLEX (cgbmv の場合)  
DOUBLE COMPLEX (zgbmv の場合)

配列、次元は  $(lda, n)$ 。このルーチンに入る前に、配列  $a$  の先頭の  $(k_l + k_u + 1) \times n$  の部分に係数の行列を格納しなければならない。この行列は、行列の主対角成分が配列の行  $(k_u + 1)$  にくるように、また、最初の優対角成分が行  $k_u$  の位置 2 から始まり、最初の劣対角成分が行  $(k_u + 2)$  の位置 1 から始まるように、列ごとに与えなければならない。帯行列の成分に対応しない配列  $a$  の成分 (左上の  $k_u \times k_u$  の三角など) は参照されない。

次に示すプログラムの一部は、帯行列を通常のフル格納形式から帯格納形式に転送するためのものである。

```
do 20, j = 1, n
  k = ku + 1 - j
  do 10, i = max(1, j-ku), min(m, j+kl)
    a(k+i, j) = matrix(i, j)
  10 continue
20 continue
```

- lda* INTEGER。呼び出し元の(サブ)プログラムで宣言されているとおりに、*a*の第1次元を指定する。*lda*の値は、 $(kl + ku + 1)$ 以上でなければならない。
- x* REAL (sgbmv の場合)  
DOUBLE PRECISION (dgbmv の場合)  
COMPLEX (cgbmv の場合)  
DOUBLE COMPLEX (zgbmv の場合)
- 配列、次元は  $(1 + (n - 1) * \text{abs}(\text{incx}))$  以上 (*trans* = 'N' または 'n' の場合) または  $(1 + (m - 1) * \text{abs}(\text{incx}))$  以上 (それ以外の場合)。このルーチンに入る前に、増分された配列 *x* にベクトル *x* を格納しなければならない。
- incx* INTEGER。*x*の成分に対する増分を指定する。*incx*の値は、ゼロであってはならない。
- beta* REAL (sgbmv の場合)  
DOUBLE PRECISION (dgbmv の場合)  
COMPLEX (cgbmv の場合)  
DOUBLE COMPLEX (zgbmv の場合)
- スカラー *beta* を指定する。*beta* をゼロに設定した場合は、*y* を設定する必要はない。
- y* REAL (sgbmv の場合)  
DOUBLE PRECISION (dgbmv の場合)  
COMPLEX (cgbmv の場合)  
DOUBLE COMPLEX (zgbmv の場合)
- 配列、次元は  $(1 + (m - 1) * \text{abs}(\text{incy}))$  以上 (*trans* = 'N' または 'n' の場合) または  $(1 + (n - 1) * \text{abs}(\text{incy}))$  以上 (それ以外の場合)。このルーチンに入る前に、増分された配列 *y* にベクトル *y* を格納しなければならない。
- incy* INTEGER。*y*の成分に対する増分を指定する。*incy*の値は、ゼロであってはならない。

## 出力パラメータ

$y$  更新されたベクトル  $y$  によって上書きされる。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[Fortran-95 インターフェイス規則](#)を参照のこと。

ルーチン `gbmv` のインターフェイスの詳細を以下に示す。

$a$	サイズ $(k1+ku+1, n)$ の配列 $A$ を格納する。
$x$	長さ $(rx)$ のベクトルを格納する。 $trans = 'N'$ の場合は $rx = n$ 、それ以外の場合は $rx = m$ である。
$y$	長さ $(ry)$ のベクトルを格納する。 $trans = 'N'$ の場合は $ry = m$ 、それ以外の場合は $ry = n$ である。
$trans$	'N'、'C'、または 'T' でなければならない。デフォルト値は 'N' である。
$k1$	省略した場合、 $k1 = ku$ とみなす。
$ku$	$ku = lda - k1 - 1$ として復元する。
$m$	省略した場合、 $m = n$ とみなす。
$alpha$	デフォルト値は '1' である。
$beta$	デフォルト値は '1' である。

## ?gemv

一般行列を使用して行列-ベクトルの積を計算する。

## 構文

### Fortran 77:

```
CALL sGEMV( TRANS, M, N, ALPHA, A, LDA, X, INCX, BETA, Y, INCY )
CALL dGEMV( TRANS, M, N, ALPHA, A, LDA, X, INCX, BETA, Y, INCY )
CALL cGEMV( TRANS, M, N, ALPHA, A, LDA, X, INCX, BETA, Y, INCY )
CALL zGEMV( TRANS, M, N, ALPHA, A, LDA, X, INCX, BETA, Y, INCY )
```

## Fortran 95:

```
call gemv(a, x, y [,alpha] [,beta] [,trans])
```

## 説明

?gemv ルーチンは、次のように定義される行列 - ベクトル演算を実行する。

```
y := alpha*a*x + beta*y
```

または

```
y := alpha*a'*x + beta*y
```

または

```
y := alpha*conjg(a')*x + beta*y
```

alpha と beta は、スカラーである。

x と y は、ベクトルである。

a は、 $m \times n$  の行列である。

## 入力パラメータ

trans CHARACTER\*1。次に示すように、実行する演算を指定する。

trans の値	実行する演算
N または n	$y := \alpha * a * x + \beta * y$
T または t	$y := \alpha * a' * x + \beta * y$
C または c	$y := \alpha * \text{conjg}(a') * x + \beta * y$

m INTEGER。行列 a の行数を指定する。m の値は、ゼロ以上でなければならない。

n INTEGER。行列 a の列数を指定する。n の値は、ゼロ以上でなければならない。

alpha REAL (sgemv の場合)  
DOUBLE PRECISION (dgemv の場合)  
COMPLEX (cgemv の場合)  
DOUBLE COMPLEX (zgemv の場合)  
スカラー alpha を指定する。

a REAL (sgemv の場合)  
DOUBLE PRECISION (dgemv の場合)  
COMPLEX (cgemv の場合)  
DOUBLE COMPLEX (zgemv の場合)



配列、次元は  $(lda, n)$ 。このルーチンに入る前に、配列  $a$  の先頭の  $m \times n$  の部分に係数の行列を格納しなければならない。

- lda** INTEGER。呼び出し元の ( サブ ) プログラムで宣言されているとおりに、 $a$  の第 1 次元を指定する。lda の値は、 $\max(1, m)$  以上でなければならない。
- x** REAL (sgemv の場合 )  
DOUBLE PRECISION (dgemv の場合 )  
COMPLEX (cgemv の場合 )  
DOUBLE COMPLEX (zgemv の場合 )
- 配列、次元は  $trans = 'N'$  または  $'n'$  の場合  $(1 + (n - 1) * \text{abs}(incx))$  以上。それ以外の場合は  $(1 + (m - 1) * \text{abs}(incx))$  以上。このルーチンに入る前に、増分された配列  $x$  にベクトル  $x$  を格納しなければならない。
- incx** INTEGER。x の成分に対する増分を指定する。incx の値は、ゼロであってはならない。
- beta** REAL (sgemv の場合 )  
DOUBLE PRECISION (dgemv の場合 )  
COMPLEX (cgemv の場合 )  
DOUBLE COMPLEX (zgemv の場合 )
- スカラ  $\beta$  を指定する。 $\beta$  をゼロに設定した場合は、 $y$  を設定する必要はない。
- y** REAL (sgemv の場合 )  
DOUBLE PRECISION (dgemv の場合 )  
COMPLEX (cgemv の場合 )  
DOUBLE COMPLEX (zgemv の場合 )
- 配列、次元は  $trans = 'N'$  または  $'n'$  の場合  $(1 + (m - 1) * \text{abs}(incy))$  以上。それ以外の場合は  $(1 + (n - 1) * \text{abs}(incy))$  以上。 $\beta$  が非ゼロでこのルーチンに入る前に、増分された配列  $y$  にベクトル  $y$  を格納しなければならない。
- incy** INTEGER。y の成分に対する増分を指定する。incy の値は、ゼロであってはならない。

### 出力パラメータ

- y** 更新されたベクトル  $y$  によって上書きされる。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[Fortran-95 インターフェイス規則](#)を参照のこと。

ルーチン `gemv` のインターフェイスの詳細を以下に示す。

<code>a</code>	サイズ $(m, n)$ の行列 $A$ を格納する。
<code>x</code>	長さ $(rx)$ のベクトルを格納する。 <code>trans = 'N'</code> の場合は $rx = n$ 、それ以外の場合は $rx = m$ である。
<code>y</code>	長さ $(ry)$ のベクトルを格納する。 <code>trans = 'N'</code> の場合は $ry = m$ 、それ以外の場合は $ry = n$ である。
<code>trans</code>	'N'、'C'、または 'T' でなければならない。デフォルト値は 'N' である。
<code>alpha</code>	デフォルト値は '1' である。
<code>beta</code>	デフォルト値は '1' である。

---

## ?ger

一般行列の階数1 の更新を実行する。

---

### 構文

#### Fortran 77:

```
CALL sGER( M, N, ALPHA, X, INCX, Y, INCY, A, LDA )  
CALL dGER( M, N, ALPHA, X, INCX, Y, INCY, A, LDA )
```

#### Fortran 95:

```
call ger(a, x, y [,alpha])
```

### 説明

?ger ルーチンは、次のように定義される行列 - ベクトル演算を実行する。

$$a := \alpha * x * y' + a$$

`alpha` は、スカラーである。

`x` は、 $m$  個の成分を持つベクトルである。

$y$  は、 $n$  個の成分を持つベクトルである。

$a$  は、 $m \times n$  の行列である。

### 入力パラメータ

$m$	INTEGER。行列 $a$ の行数を指定する。 $m$ の値は、ゼロ以上でなければならない。
$n$	INTEGER。行列 $a$ の列数を指定する。 $n$ の値は、ゼロ以上でなければならない。
$alpha$	REAL (sger の場合) DOUBLE PRECISION (dger の場合) スカラ $alpha$ を指定する。
$x$	REAL (sger の場合) DOUBLE PRECISION (dger の場合) 配列、次元は $(1 + (m - 1) * \text{abs}(incx))$ 以上。このルーチンに入る前に、増分された配列 $x$ に $m$ 個の成分を持つベクトル $x$ を格納しなければならない。
$incx$	INTEGER。 $x$ の成分に対する増分を指定する。 $incx$ の値は、ゼロであってはならない。
$y$	REAL (sger の場合) DOUBLE PRECISION (dger の場合) 配列、次元は $(1 + (n - 1) * \text{abs}(incy))$ 以上。このルーチンに入る前に、増分された配列 $y$ に $n$ 個の成分を持つベクトル $y$ を格納しなければならない。
$incy$	INTEGER。 $y$ の成分に対する増分を指定する。 $incy$ の値は、ゼロであってはならない。
$a$	REAL (sger の場合) DOUBLE PRECISION (dger の場合) 配列、次元は $(lda, n)$ 。このルーチンに入る前に、配列 $a$ の先頭の $m \times n$ の部分に係数の行列を格納しなければならない。
$lda$	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 $a$ の第 1 次元を指定する。 $lda$ の値は、 $\max(1, m)$ 以上でなければならない。

### 出力パラメータ

$a$  更新された行列によって上書きされる。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[Fortran-95 インターフェイス規則](#)を参照のこと。

ルーチン `ger` のインターフェイスの詳細を以下に示す。

`a`            サイズ  $(m, n)$  の行列  $A$  を格納する。  
`x`            長さ  $(m)$  のベクトルを格納する。  
`y`            長さ  $(n)$  のベクトルを格納する。  
`alpha`       デフォルト値は '1' である。

---

## ?gerc

一般行列の階数1の更新(共役)を実行する。

---

### 構文

#### Fortran 77:

```
CALL CGERC( M, N, ALPHA, X, INCX, Y, INCY, A, LDA )  
CALL ZGERC( M, N, ALPHA, X, INCX, Y, INCY, A, LDA )
```

#### Fortran 95:

```
call gerc(a, x, y [,alpha])
```

### 説明

?gerc ルーチンは、次のように定義される行列 - ベクトル演算を実行する。

$$a := \alpha * x * \text{conjg}(y') + a$$

`alpha` は、スカラーである。

`x` は、 $m$  個の成分を持つベクトルである。

`y` は、 $n$  個の成分を持つベクトルである。

`a` は、 $m \times n$  の行列である。

### 入力パラメータ

<i>m</i>	INTEGER。行列 <i>a</i> の行数を指定する。 <i>m</i> の値は、ゼロ以上でなければならない。
<i>n</i>	INTEGER。行列 <i>a</i> の列数を指定する。 <i>n</i> の値は、ゼロ以上でなければならない。
<i>alpha</i>	SINGLE PRECISION COMPLEX (cgerc の場合) DOUBLE PRECISION COMPLEX (zgerc の場合)  スカラ <i>alpha</i> を指定する。
<i>x</i>	SINGLE PRECISION COMPLEX (cgerc の場合) DOUBLE PRECISION COMPLEX (zgerc の場合)  配列、次元は $(1 + (m - 1) * \text{abs}(\text{incx}))$ 以上。このルーチンに入る前に、増分された配列 <i>x</i> に <i>m</i> 個の成分を持つベクトル <i>x</i> を格納しなければならない。
<i>incx</i>	INTEGER。 <i>x</i> の成分に対する増分を指定する。 <i>incx</i> の値は、ゼロであってはならない。
<i>y</i>	COMPLEX (cgerc の場合) DOUBLE COMPLEX (zgerc の場合)  配列、次元は $(1 + (n - 1) * \text{abs}(\text{incy}))$ 以上。このルーチンに入る前に、増分された配列 <i>y</i> に <i>n</i> 個の成分を持つベクトル <i>y</i> を格納しなければならない。
<i>incy</i>	INTEGER。 <i>y</i> の成分に対する増分を指定する。 <i>incy</i> の値は、ゼロであってはならない。
<i>a</i>	COMPLEX (cgerc の場合) DOUBLE COMPLEX (zgerc の場合)  配列、次元は $(lda, n)$ 。このルーチンに入る前に、配列 <i>a</i> の先頭の $m \times n$ の部分に係数の行列を格納しなければならない。
<i>lda</i>	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 <i>a</i> の第 1 次元を指定する。 <i>lda</i> の値は、 $\max(1, m)$ 以上でなければならない。

### 出力パラメータ

<i>a</i>	更新された行列によって上書きされる。
----------	--------------------

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[Fortran-95 インターフェイス規則](#)を参照のこと。

ルーチン gerc のインターフェイスの詳細を以下に示す。

$a$	サイズ $(m, n)$ の行列 $A$ を格納する。
$x$	長さ $(m)$ のベクトルを格納する。
$y$	長さ $(n)$ のベクトルを格納する。
$alpha$	デフォルト値は '1' である。

---

### ?geru

一般行列の階数 1 の更新 (非共役) を実行する。

---

#### 構文

##### Fortran 77:

```
CALL CGERU( M, N, ALPHA, X, INCX, Y, INCY, A, LDA )  
CALL ZGERU( M, N, ALPHA, X, INCX, Y, INCY, A, LDA )
```

##### Fortran 95:

```
call geru(a, x, y [,alpha])
```

#### 説明

?geru ルーチンは、次のように定義される行列 - ベクトル演算を実行する。

$$a := alpha * x * y' + a$$

$alpha$  は、スカラーである。

$x$  は、 $m$  個の成分を持つベクトルである。

$y$  は、 $n$  個の成分を持つベクトルである。

$a$  は、 $m \times n$  の行列である。

#### 入力パラメータ

$m$	INTEGER。行列 $a$ の行数を指定する。 $m$ の値は、ゼロ以上でなければならない。
$n$	INTEGER。行列 $a$ の列数を指定する。 $n$ の値は、ゼロ以上でなければならない。
$alpha$	COMPLEX (cgeru の場合 ) DOUBLE COMPLEX (zgeru の場合 )

	スカラ $\alpha$ を指定する。
$x$	COMPLEX (cgeru の場合) DOUBLE COMPLEX (zgeru の場合)  配列、次元は $(1 + (m - 1) * \text{abs}(\text{incx}))$ 以上。このルーチンに入る前に、増分された配列 $x$ に $m$ 個の成分を持つベクトル $x$ を格納しなければならない。
$\text{incx}$	INTEGER。 $x$ の成分に対する増分を指定する。 $\text{incx}$ の値は、ゼロであってはならない。
$y$	COMPLEX (cgeru の場合) DOUBLE COMPLEX (zgeru の場合)  配列、次元は $(1 + (n - 1) * \text{abs}(\text{incy}))$ 以上。このルーチンに入る前に、増分された配列 $y$ に $n$ 個の成分を持つベクトル $y$ を格納しなければならない。
$\text{incy}$	INTEGER。 $y$ の成分に対する増分を指定する。 $\text{incy}$ の値は、ゼロであってはならない。
$a$	COMPLEX (cgeru の場合) DOUBLE COMPLEX (zgeru の場合)  配列、次元は $(\text{lda}, n)$ 。このルーチンに入る前に、配列 $a$ の先頭の $m \times n$ の部分に係数の行列を格納しなければならない。
$\text{lda}$	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 $a$ の第 1 次元を指定する。 $\text{lda}$ の値は、 $\max(1, m)$ 以上でなければならない。

### 出力パラメータ

$a$           更新された行列によって上書きされる。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[Fortran-95 インターフェイス規則](#)を参照のこと。

ルーチン `geru` のインターフェイスの詳細を以下に示す。

$a$	サイズ $(m, n)$ の行列 $A$ を格納する。
$x$	長さ $(m)$ のベクトルを格納する。
$y$	長さ $(n)$ のベクトルを格納する。
$\alpha$	デフォルト値は '1' である。

## ?hbmv

エルミート帯行列を使用して行列・ベクトルの積を計算する。

---

### 構文

#### Fortran 77:

```
CALL CHBMV( UPLO, N, K, ALPHA, A, LDA, X, INCX, BETA, Y, INCY )
CALL ZHBMV( UPLO, N, K, ALPHA, A, LDA, X, INCX, BETA, Y, INCY )
```

#### Fortran 95:

```
CALL HBMV(a, x, y [,uplo] [,alpha] [,beta])
```

### 説明

?hbmv ルーチンは、次のように定義される行列・ベクトル演算を実行する。

$y := \alpha * a * x + \beta * y$

$\alpha$  と  $\beta$  は、スカラーである。

$x$  と  $y$  は、 $n$  個の成分を持つベクトルである。

$a$  は、 $k$  個の優対角成分を持つ  $n \times n$  のエルミート帯行列である。

### 入力パラメータ

**uplo** CHARACTER\*1。次に示すように、帯行列  $a$  の上三角部分と下三角部分のどちらを与えるかを指定する。

uplo の値	与える行列 $a$ の部分
U または u	行列 $a$ の上三角部分を与える。
L または l	行列 $a$ の下三角部分を与える。

**n** INTEGER。行列  $a$  の次数を指定する。 $n$  の値は、ゼロ以上でなければならない。

**k** INTEGER。行列  $a$  の優対角成分の数を指定する。 $k$  の値は、 $0 \leq k$  を満たさなければならない。



*alpha*      COMPLEX (chbm<sub>v</sub> の場合 )  
 DOUBLE COMPLEX (zhbm<sub>v</sub> の場合 )  
 スカラ *alpha* を指定する。

*a*            COMPLEX (chbm<sub>v</sub> の場合 )  
 DOUBLE COMPLEX (zhbm<sub>v</sub> の場合 )

配列、次元は (*lda*, *n*)。uplo = 'U' または 'u' と指定した場合は、このルーチンに入る前に、配列 *a* の先頭の  $(k+1) \times n$  の部分に、エルミート行列の上三角帯部分を格納しなければならない。この行列には、行列の主対角成分が配列の行  $(k+1)$  にくるように、また最初の優対角成分が行 *k* の位置 2 から始まるように、列ごとに与えなければならない。配列 *a* の左上の  $k \times k$  の三角は、参照されない。

次に示すプログラムの一部は、エルミート帯行列の上三角部分を通常のフル格納形式の行列から帯格納形式に転送するためのものである。

```
do 20, j = 1, n
  m = k + 1 - j
  do 10, i = max(1, j - k), j
    a(m + i, j) = matrix(i, j)
  10 continue
20 continue
```

uplo = 'L' または 'l' と指定した場合は、このルーチンに入る前に、配列 *a* の先頭の  $(k+1) \times n$  の部分に、エルミート行列の下三角帯部分を格納しなければならない。この行列には、行列の主対角成分が配列の行 1 にくるように、また最初の劣対角成分が行 2 の位置 1 から始まるように、列ごとに与えなければならない。配列 *a* の右下の  $k \times k$  の三角は、参照されない。

次に示すプログラムの一部は、エルミート帯行列の下三角部分を通常のフル格納形式の行列から帯格納形式に転送するためのものである。

```
do 20, j = 1, n
  m = 1 - j
  do 10, i = j, min( n, j + k )
    a( m + i, j ) = matrix( i, j )
  10 continue
20 continue
```

対角成分の虚数部は設定する必要はなく、ゼロとみなされる。

*lda*            INTEGER。呼び出し元の ( サブ ) プログラムで宣言されているとおりに、*a* の第 1 次元を指定する。*lda* の値は、 $(k+1)$  以上でなければならない。

$x$	COMPLEX (chbm $v$ の場合) DOUBLE COMPLEX (zhbm $v$ の場合)  配列、次元は $(1 + (n - 1) * \text{abs}(\text{incx}))$ 以上。このルーチンに入る前に、増分された配列 $x$ にベクトル $x$ を格納しなければならない。
$\text{incx}$	INTEGER。 $x$ の成分に対する増分を指定する。 $\text{incx}$ の値はゼロであってはならない。
$\text{beta}$	COMPLEX (chbm $v$ の場合) DOUBLE COMPLEX (zhbm $v$ の場合)  スカラ $\text{beta}$ を指定する。
$y$	COMPLEX (chbm $v$ の場合) DOUBLE COMPLEX (zhbm $v$ の場合)  配列、次元は $(1 + (n - 1) * \text{abs}(\text{incy}))$ 以上。このルーチンに入る前に、増分された配列 $y$ にベクトル $y$ を格納しなければならない。
$\text{incy}$	INTEGER。 $y$ の成分に対する増分を指定する。 $\text{incy}$ の値は、ゼロであってはならない。

### 出力パラメータ

$y$             更新されたベクトル  $y$  によって上書きされる。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[Fortran-95 インターフェイス規則](#)を参照のこと。

ルーチン hbm $v$  のインターフェイスの詳細を以下に示す。

$a$	サイズ $(k+1, n)$ の配列 $A$ を格納する。
$x$	長さ $(n)$ のベクトルを格納する。
$y$	長さ $(n)$ のベクトルを格納する。
$\text{uplo}$	'U' または 'L' でなければならない。デフォルト値は 'U' である。
$\alpha$	デフォルト値は '1' である。
$\text{beta}$	デフォルト値は '1' である。

## ?hemv

エルミート行列を使用して行列-ベクトルの積を計算する。

### 構文

**Fortran 77:**

```
call chemv( UPLO, N, ALPHA, A, LDA, X, INCX, BETA, Y, incy )
CALL zHEMV( UPLO, N, ALPHA, A, LDA, X, INCX, BETA, Y, INCY )
```

**Fortran 95:**

```
CALL HeMV(a, x, y [,uplo] [,alpha] [,beta])
```

### 説明

?hemv ルーチンは、次のように定義される行列 - ベクトル演算を実行する。

$$y := \alpha * a * x + \beta * y$$

alpha と beta は、スカラである。

x と y は、n 個の成分を持つベクトルである。

a は、n × n のエルミート行列である。

### 入力パラメータ

uplo CHARACTER\*1。次に示すように、配列 a の上三角部分と下三角部分のどちらが参照されるかを指定する。

uplo の値	参照される配列 a の部分
U または u	配列 a の上三角部分が参照される。
L または l	配列 a の下三角部分が参照される。

n INTEGER。行列 a の次数を指定する。n の値は、ゼロ 以上でなければならない。

alpha COMPLEX (chemv の場合 )  
DOUBLE COMPLEX (zhemv の場合 )  
スカラ alpha を指定する。

<i>a</i>	COMPLEX (chemv の場合 ) DOUBLE COMPLEX (zhemv の場合 )  配列、次元は ( <i>lda</i> , <i>n</i> )。 <i>uplo</i> = 'U' または 'u' と指定した場合は、このルーチンに入る前に、配列 <i>a</i> の先頭の $n \times n$ の上三角部分に、エルミート行列の上三角部分を格納しなければならない。また <i>a</i> の厳密な下三角部分は参照されない。また、 <i>uplo</i> = 'L' または 'l' と指定した場合は、このルーチンに入る前に、配列 <i>a</i> の先頭の $n \times n$ の下三角部分に、エルミート行列の下三角部分を格納しなければならない。また <i>a</i> の厳密な上三角部分は参照されない。  対角成分の虚数部は設定する必要はなく、ゼロとみなされる。
<i>lda</i>	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 <i>a</i> の第 1 次元を指定する。 <i>lda</i> の値は、 $\max(1, n)$ 以上でなければならない。
<i>x</i>	COMPLEX (chemv の場合 ) DOUBLE COMPLEX (zhemv の場合 )  配列、次元は $(1 + (n - 1) * \text{abs}(\text{incx}))$ 以上。このルーチンに入る前に、増分された配列 <i>x</i> に <i>n</i> 個の成分を持つベクトル <i>x</i> を格納しなければならない。
<i>incx</i>	INTEGER。 <i>x</i> の成分に対する増分を指定する。 <i>incx</i> の値は、ゼロであってはならない。
<i>beta</i>	COMPLEX (chemv の場合 ) DOUBLE COMPLEX (zhemv の場合 )  スカラ <i>beta</i> を指定する。 <i>beta</i> をゼロに設定した場合は、 <i>y</i> を設定する必要はない。
<i>y</i>	COMPLEX (chemv の場合 ) DOUBLE COMPLEX (zhemv の場合 )  配列、次元は $(1 + (n - 1) * \text{abs}(\text{incy}))$ 以上。このルーチンに入る前に、増分された配列 <i>y</i> に <i>n</i> 個の成分を持つベクトル <i>y</i> を格納しなければならない。
<i>incy</i>	INTEGER。 <i>y</i> の成分に対する増分を指定する。 <i>incy</i> の値は、ゼロであってはならない。

### 出力パラメータ

<i>y</i>	更新されたベクトル <i>y</i> によって上書きされる。
----------	--------------------------------

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[Fortran-95 インターフェイス規則](#)を参照のこと。

ルーチン hemv のインターフェイスの詳細を以下に示す。

*a*            サイズ  $(n, n)$  の行列 *A* を格納する。  
*x*            長さ  $(n)$  のベクトルを格納する。  
*y*            長さ  $(n)$  のベクトルを格納する。  
*uplo*        'U' または 'L' でなければならない。デフォルト値は 'U' である。  
*alpha*       デフォルト値は '1' である。  
*beta*        デフォルト値は '1' である。

## ?her

エルミート行列の階数 1 の更新を実行する。

### 構文

#### Fortran 77:

```
CALL CHER( UPLO, N, ALPHA, X, INCX, A, LDA )
CALL ZHER( UPLO, N, ALPHA, X, INCX, A, LDA )
```

#### Fortran 95:

```
CALL Her(a, x [,uplo] [,alpha])
```

### 説明

?her ルーチンは、次のように定義される行列 - ベクトル演算を実行する。

$$a := \alpha * x * \text{conjg}(x') + a$$

*alpha* は、実数のスカラである。

*x* は、 $n$  個の成分を持つベクトルである。

*a* は、 $n \times n$  のエルミート行列である。

## 入力パラメータ

**uplo** CHARACTER\*1。次に示すように、配列 **a** の上三角部分と下三角部分のどちらが参照されるかを指定する。

<b>uplo</b> の値	参照される配列 <b>a</b> の部分
U または u	配列 <b>a</b> の上三角部分が参照される。
L または l	配列 <b>a</b> の下三角部分が参照される。

**n** INTEGER。行列 **a** の次数を指定する。**n** の値は、ゼロ以上でなければならない。

**alpha** REAL (cher の場合)  
DOUBLE PRECISION (zher の場合)  
スカラー **alpha** を指定する。

**x** COMPLEX (cher の場合)  
DOUBLE COMPLEX (zher の場合)

配列、次元は  $(1 + (n - 1) * \text{abs}(\text{incx}))$  以上。このルーチンに入る前に、増分された配列 **x** に **n** 個の成分を持つベクトル **x** を格納しなければならない。

**incx** INTEGER。**x** の成分に対する増分を指定する。**incx** の値は、ゼロであってはならない。

**a** COMPLEX (cher の場合)  
DOUBLE COMPLEX (zher の場合)

配列、次元は  $(lda, n)$ 。**uplo** = 'U' または 'u' と指定した場合は、このルーチンに入る前に、配列 **a** の先頭の  $n \times n$  の上三角部分に、エルミート行列の上三角部分を格納しなければならない、また **a** の厳密な下三角部分は参照されない。

**uplo** = 'L' または 'l' と指定した場合は、このルーチンに入る前に、配列 **a** の先頭の  $n \times n$  の下三角部分に、エルミート行列の下三角部分を格納しなければならない、また **a** の厳密な上三角部分は参照されない。

対角成分の虚数部は設定する必要はなく、ゼロとみなされる。

**lda** INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、**a** の第 1 次元を指定する。**lda** の値は、 $\max(1, n)$  以上でなければならない。

## 出力パラメータ

**a** **uplo** = 'U' または 'u' と指定した場合は、配列 **a** の上三角部分が、更新された行列の上三角部分によって上書きされる。

`uplo = 'L'` または `'l'` と指定した場合は、配列 `a` の下三角部分が、更新された行列の下三角部分によって上書きされる。

対角成分の虚数部は、ゼロに設定される。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[Fortran-95 インターフェイス規則](#)を参照のこと。

ルーチン `her` のインターフェイスの詳細を以下に示す。

`a`            サイズ  $(n,n)$  の行列  $A$  を格納する。  
`x`            長さ  $(n)$  のベクトルを格納する。  
`uplo`        `'U'` または `'L'` でなければならない。デフォルト値は `'U'` である。  
`alpha`      デフォルト値は `'1'` である。

## ?her2

エルミート行列の階数2 の更新を実行する。

### 構文

#### Fortran 77:

```
CALL CHER2( uplo, n, alpha, x, incx, y, incy, a, lda )
CALL ZHER2( uplo, n, alpha, x, incx, y, incy, a, lda )
```

#### Fortran 95:

```
CALL Her2(a, x, y [,uplo] [,alpha])
```

### 説明

?her2 ルーチンは、次のように定義される行列 - ベクトル演算を実行する。

$$a := \alpha * x * \text{conjg}(y') + \text{conjg}(\alpha) * y * \text{conjg}(x') + a$$

`alpha` は、スカラーである。

`x` と `y` は、 $n$  個の成分を持つベクトルである。

$a$  は、 $n \times n$  のエルミート行列である。

## 入力パラメータ

*uplo* CHARACTER\*1。次に示すように、配列  $a$  の上三角部分と下三角部分のどちらが参照されるかを指定する。

<i>uplo</i> の値	参照される配列 $a$ の部分
U または u	配列 $a$ の上三角部分が参照される。
L または l	配列 $a$ の下三角部分が参照される。

*n* INTEGER。行列  $a$  の次数を指定する。 $n$  の値は、ゼロ以上でなければならない。

*alpha* COMPLEX (cher2 の場合 )  
DOUBLE COMPLEX (zher2 の場合 )  
スカラー *alpha* を指定する。

*x* COMPLEX (cher2 の場合 )  
DOUBLE COMPLEX (zher2 の場合 )

配列、次元は  $(1 + (n - 1) * \text{abs}(\text{incx}))$  以上。このルーチンに入る前に、増分された配列  $x$  に  $n$  個の成分を持つベクトル  $x$  を格納しなければならない。

*incx* INTEGER。 $x$  の成分に対する増分を指定する。*incx* の値は、ゼロであってはならない。

*y* COMPLEX (cher2 の場合 )  
DOUBLE COMPLEX (zher2 の場合 )

配列、次元は  $(1 + (n - 1) * \text{abs}(\text{incy}))$  以上。このルーチンに入る前に、増分された配列  $y$  に  $n$  個の成分を持つベクトル  $y$  を格納しなければならない。

*incy* INTEGER。 $y$  の成分に対する増分を指定する。*incy* の値は、ゼロであってはならない。

*a* COMPLEX (cher2 の場合 )  
DOUBLE COMPLEX (zher2 の場合 )

配列、次元は  $(lda, n)$ 。*uplo* = 'U' または 'u' と指定した場合は、このルーチンに入る前に、配列  $a$  の先頭の  $n \times n$  の上三角部分に、エルミート行列の上三角部分を格納しなければならない、また  $a$  の厳密な下三角部分は参照されない。



`uplo = 'L'` または `'l'` と指定した場合は、このルーチンに入る前に、配列 `a` の先頭の  $n \times n$  の下三角部分に、エルミート行列の下三角部分を格納しなければならない。また `a` の厳密な上三角部分は参照されない。

対角成分の虚数部は設定する必要はなく、ゼロとみなされる。

`lda` INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、`a` の第 1 次元を指定する。`lda` の値は、 $\max(1, n)$  以上でなければならない。

### 出力パラメータ

`a` `uplo = 'U'` または `'u'` と指定した場合は、配列 `a` の上三角部分が、更新された行列の上三角部分によって上書きされる。

`uplo = 'L'` または `'l'` と指定した場合は、配列 `a` の下三角部分が、更新された行列の下三角部分によって上書きされる。

対角成分の虚数部は、ゼロに設定される。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[Fortran-95 インターフェイス規則](#)を参照のこと。

ルーチン `her2` のインターフェイスの詳細を以下に示す。

`a` サイズ  $(n, n)$  の行列 `A` を格納する。

`x` 長さ  $(n)$  のベクトルを格納する。

`y` 長さ  $(n)$  のベクトルを格納する。

`uplo` 'U' または 'L' でなければならない。デフォルト値は 'U' である。

`alpha` デフォルト値は '1' である。

## ?hpmv

圧縮形式のエルミート行列を使用して  
行列-ベクトルの積を計算する。

### 構文

#### Fortran 77:

```
CALL CHPMV( UPLO, N, ALPHA, AP, X, INCX, BETA, Y, INCY )
CALL ZHPMV( UPLO, N, ALPHA, AP, X, INCX, BETA, Y, INCY )
```

#### Fortran 95:

```
CALL Hpmv(a, x, y [,uplo] [,alpha] [,beta])
```

### 説明

?hpmv ルーチンは、次のように定義される行列-ベクトル演算を実行する。

$y := \alpha * a * x + \beta * y$

$\alpha$  と  $\beta$  は、スカラである。

$x$  と  $y$  は、 $n$  個の成分を持つベクトルである。

$a$  は、圧縮形式で与えられる  $n \times n$  のエルミート行列である。

### 入力パラメータ

**uplo** CHARACTER\*1。次に示すように、行列  $a$  の上三角部分と下三角部分のどちらを圧縮形式の配列  $ap$  において与えるかを指定する。

uplo の値	与える行列 $a$ の部分
U または u	行列 $a$ の上三角部分を $ap$ において与える。
L または l	行列 $a$ の下三角部分を $ap$ において与える。

**n** INTEGER。行列  $a$  の次数を指定する。 $n$  の値は、ゼロ以上でなければならない。

**alpha** COMPLEX (chpmv の場合 )  
DOUBLE COMPLEX (zhpmv の場合 )  
スカラ  $\alpha$  を指定する。

<i>ap</i>	COMPLEX (chpmv の場合 ) DOUBLE COMPLEX (zhpmv の場合 )  配列、次元は $((n*(n+1))/2)$ 以上。 <i>uplo</i> = 'U' または 'u' と指定した場合は、このルーチンに入る前に、 <i>ap</i> (1) に <i>a</i> (1, 1) が、 <i>ap</i> (2) に <i>a</i> (1, 2) が、 <i>ap</i> (3) に <i>a</i> (2, 2) がそれぞれ格納されるように、配列 <i>ap</i> には、順番に圧縮されたエルミート行列の上三角部分を列ごとに格納しなければならない。また、 <i>uplo</i> = 'L' または 'l' と指定した場合は、このルーチンに入る前に、 <i>ap</i> (1) に <i>a</i> (1, 1) が、 <i>ap</i> (2) に <i>a</i> (2, 1) が、 <i>ap</i> (3) に <i>a</i> (3, 1) がそれぞれ格納されるように、配列 <i>ap</i> には、順番に圧縮されたエルミート行列の下三角部分を列ごとに格納しなければならない。  対角成分の虚数部は設定する必要はなく、ゼロとみなされる。
<i>x</i>	COMPLEX (chpmv の場合 ) DOUBLE PRECISION COMPLEX (zhpmv の場合 )  配列、次元は $(1 + (n - 1)*abs(incx))$ 以上。このルーチンに入る前に、増分された配列 <i>x</i> に <i>n</i> 個の成分を持つベクトル <i>x</i> を格納しなければならない。
<i>incx</i>	INTEGER。 <i>x</i> の成分に対する増分を指定する。 <i>incx</i> の値は、ゼロであってはならない。
<i>beta</i>	COMPLEX (chpmv の場合 ) DOUBLE COMPLEX (zhpmv の場合 )  スカラー <i>beta</i> を指定する。 <i>beta</i> をゼロに設定した場合は、 <i>y</i> を設定する必要はない。
<i>y</i>	COMPLEX (chpmv の場合 ) DOUBLE COMPLEX (zhpmv の場合 )  配列、次元は $(1 + (n - 1)*abs(incy))$ 以上。このルーチンに入る前に、増分された配列 <i>y</i> に <i>n</i> 個の成分を持つベクトル <i>y</i> を格納しなければならない。
<i>incy</i>	INTEGER。 <i>y</i> の成分に対する増分を指定する。 <i>incy</i> の値は、ゼロであってはならない。

## 出力パラメータ

*y*            更新されたベクトル *y* によって上書きされる。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[Fortran-95 インターフェイス規則](#)を参照のこと。

ルーチン `hpmv` のインターフェイスの詳細を以下に示す。

<code>a</code>	サイズ $(n*(n+1)/2)$ の配列 <code>A</code> を格納する。
<code>x</code>	長さ $(n)$ のベクトルを格納する。
<code>y</code>	長さ $(n)$ のベクトルを格納する。
<code>uplo</code>	'U' または 'L' でなければならない。デフォルト値は 'U' である。
<code>alpha</code>	デフォルト値は '1' である。
<code>beta</code>	デフォルト値は '1' である。

---

### ?hpr

圧縮形式のエルミート行列の階数  $1$  の更新を実行する。

---

#### 構文

##### Fortran 77:

```
CALL CHPR( UPLO, N, ALPHA, X, INCX, AP )  
CALL zHPR( UPLO, N, ALPHA, X, INCX, AP )
```

##### Fortran 95:

```
CALL Hpr(a, x [,uplo] [,alpha])
```

#### 説明

?hpr ルーチンは、次のように定義される行列 - ベクトル演算を実行する。

```
a := alpha*x*conjg(x') + a
```

`alpha` は、実数のスカラーである。

`x` は、 $n$  個の成分を持つベクトルである。

`a` は、圧縮形式で与えられる  $n \times n$  のエルミート行列である。

## 入力パラメータ

**uplo** CHARACTER\*1。次に示すように、行列 *a* の上三角部分と下三角部分のどちらを圧縮形式の配列 *ap* において与えるかを指定する。

<i>uplo</i> の値	与える行列 <i>a</i> の部分
U または u	行列 <i>a</i> の上三角部分を <i>ap</i> において与える。
L または l	行列 <i>a</i> の下三角部分を <i>ap</i> において与える。

**n** INTEGER。行列 *a* の次数を指定する。*n* の値は、ゼロ以上でなければならない。

**alpha** REAL (chpr の場合)  
DOUBLE PRECISION (zhpr の場合)  
スカラ *alpha* を指定する。

**x** COMPLEX (chpr の場合)  
DOUBLE COMPLEX (zhpr の場合)

配列、次元は  $(1 + (n - 1) * \text{abs}(\text{incx}))$  以上。このルーチンに入る前に、増分された配列 *x* に *n* 個の成分を持つベクトル *x* を格納しなければならない。

**incx** INTEGER。*x* の成分に対する増分を指定する。*incx* の値は、ゼロであってはならない。

**ap** COMPLEX (chpr の場合)  
DOUBLE COMPLEX (zhpr の場合)

配列、次元は  $((n * (n + 1)) / 2)$  以上。*uplo* = 'U' または 'u' と指定した場合は、このルーチンに入る前に、*ap*(1) に *a*(1, 1) が、*ap*(2) に *a*(1, 2) が、*ap*(3) に *a*(2, 2) がそれぞれ格納されるように、配列 *ap* には、順番に圧縮されたエルミート行列の上三角部分を列ごとに格納しなければならない。

*uplo* = 'L' または 'l' と指定した場合は、このルーチンに入る前に、*ap*(1) に *a*(1, 1) が、*ap*(2) に *a*(2, 1) が、*ap*(3) に *a*(3, 1) がそれぞれ格納されるように、配列 *ap* には、順番に圧縮されたエルミート行列の下三角部分を列ごとに格納しなければならない。

対角成分の虚数部は設定する必要はなく、ゼロとみなされる。

### 出力パラメータ

`ap`            `uplo = 'U'` または `'u'` と指定した場合は、更新された行列の上三角部分によって上書きされる。

`uplo = 'L'` または `'l'` と指定した場合は、更新された行列の下三角部分によって上書きされる。

              対角成分の虚数部は、ゼロに設定される。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[Fortran-95 インターフェイス規則](#)を参照のこと。

ルーチン `hpr` のインターフェイスの詳細を以下に示す。

`a`            サイズ  $(n*(n+1)/2)$  の配列 `A` を格納する。

`x`            長さ  $(n)$  のベクトルを格納する。

`uplo`        `'U'` または `'L'` でなければならない。デフォルト値は `'U'` である。

`alpha`      デフォルト値は `'1'` である。

---

## ?hpr2

圧縮形式のエルミート行列の階数2 の更新を実行する。

---

### 構文

#### Fortran 77:

```
CALL CHPR2( UPLO, N, ALPHA, X, INCX, Y, INCY, AP )
CALL zHPR2( UPLO, N, ALPHA, X, INCX, Y, INCY, AP )
```

#### Fortran 95:

```
CALL Hpr2(a, x, y [,uplo] [,alpha])
```

## 説明

?hpr2 ルーチンは、次のように定義される行列 - ベクトル演算を実行する。

$$a := \alpha * x * \text{conjg}(y') + \text{conjg}(\alpha) * y * \text{conjg}(x') + a$$

$\alpha$  は、スカラーである。

$x$  と  $y$  は、 $n$  個の成分を持つベクトルである。

$a$  は、圧縮形式で与えられる  $n \times n$  のエルミート行列である。

## 入力パラメータ

*uplo* CHARACTER\*1。次に示すように、行列  $a$  の上三角部分と下三角部分のどちらを圧縮形式の配列  $ap$  において与えるかを指定する。

<i>uplo</i> の値	与える行列 $a$ の部分
U または u	行列 $a$ の上三角部分を $ap$ において与える。
L または l	行列 $a$ の下三角部分を $ap$ において与える。

$n$  INTEGER。行列  $a$  の次数を指定する。 $n$  の値は、ゼロ以上でなければならない。

$\alpha$  COMPLEX (chpr2 の場合)  
DOUBLE COMPLEX (zhpr2 の場合)  
スカラー  $\alpha$  を指定する。

$x$  COMPLEX (chpr2 の場合)  
DOUBLE COMPLEX (zhpr2 の場合)

配列、次元は  $(1 + (n - 1) * \text{abs}(\text{incx}))$  以上。このルーチンに入る前に、増分された配列  $x$  に  $n$  個の成分を持つベクトル  $x$  を格納しなければならない。

$\text{incx}$  INTEGER。 $x$  の成分に対する増分を指定する。 $\text{incx}$  の値は、ゼロであってはならない。

$y$  COMPLEX (chpr2 の場合)  
DOUBLE COMPLEX (zhpr2 の場合)

配列、次元は  $(1 + (n - 1) * \text{abs}(\text{incy}))$  以上。このルーチンに入る前に、増分された配列  $y$  に  $n$  個の成分を持つベクトル  $y$  を格納しなければならない。

$\text{incy}$  INTEGER。 $y$  の成分に対する増分を指定する。 $\text{incy}$  の値は、ゼロであってはならない。

`ap`            `COMPLEX (chpr2 の場合)`  
              `DOUBLE COMPLEX (zhpr2 の場合)`

配列、次元は  $((n*(n+1))/2)$  以上。 `uplo = 'U'` または `'u'` と指定した場合は、このルーチンに入る前に、`ap(1)` に `a(1, 1)` が、`ap(2)` に `a(1, 2)` が、`ap(3)` に `a(2, 2)` がそれぞれ格納されるように、配列 `ap` には、順番に圧縮されたエルミート行列の上三角部分を列ごとに格納しなければならない。

`uplo = 'L'` または `'l'` と指定した場合は、このルーチンに入る前に、`ap(1)` に `a(1, 1)` が、`ap(2)` に `a(2, 1)` が、`ap(3)` に `a(3, 1)` がそれぞれ格納されるように、配列 `ap` には、順番に圧縮されたエルミート行列の下三角部分を列ごとに格納しなければならない。

対角成分の虚数部は設定する必要はなく、ゼロとみなされる。

### 出力パラメータ

`ap`            `uplo = 'U'` または `'u'` と指定した場合は、更新された行列の上三角部分によって上書きされる。

`uplo = 'L'` または `'l'` と指定した場合は、更新された行列の下三角部分によって上書きされる。

対角成分の虚数部はゼロに設定される。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[Fortran-95 インターフェイス規則](#)を参照のこと。

ルーチン `hpr2` のインターフェイスの詳細を以下に示す。

`a`            サイズ  $(n*(n+1)/2)$  の配列 `A` を格納する。

`x`            長さ  $(n)$  のベクトルを格納する。

`y`            長さ  $(n)$  のベクトルを格納する。

`uplo`        `'U'` または `'L'` でなければならない。デフォルト値は `'U'` である。

`alpha`      デフォルト値は `'1'` である。



?sbmv

対称帯行列を使用して行列-ベクトルの積を計算する。

構文

Fortran 77:

```
CALL sSBMV( UPLO, N, K, ALPHA, A, LDA, X, INCX, BETA, Y, INCY )
CALL DSBMV( UPLO, N, K, ALPHA, A, LDA, X, INCX, BETA, Y, INCY )
```

Fortran 95:

```
CALL sbmv(a, x, y [,uplo] [,alpha] [,beta])
```

説明

?sbmv ルーチンは、次のように定義される行列-ベクトル演算を実行する。

$y := \alpha * a * x + \beta * y$

*alpha* と *beta* は、スカラである。

*x* と *y* は、*n* 個の成分を持つベクトルである。

*a* は、*k* 個の優対角成分を持つ  $n \times n$  の対称帯行列である。

入力パラメータ

*uplo* CHARACTER\*1。次に示すように、帯行列 *a* の上三角部分と下三角部分のどちらを与えるかを指定する。

<i>uplo</i> の値	与える行列 <i>a</i> の部分
U または u	行列 <i>a</i> の上三角部分を与える。
L または l	行列 <i>a</i> の下三角部分を与える。

*n* INTEGER。行列 *a* の次数を指定する。*n* の値は、ゼロ以上でなければならない。

*k* INTEGER。行列 *a* の優対角成分の数を指定する。*k* の値は、 $0 \leq k$  を満たさなければならない。

*alpha*      REAL (ssbm<sub>v</sub> の場合)  
               DOUBLE PRECISION (dsbm<sub>v</sub> の場合)  
               スカラ *alpha* を指定する。

*a*            REAL (ssbm<sub>v</sub> の場合)  
               DOUBLE PRECISION (dsbm<sub>v</sub> の場合)

配列、次元は (*lda*, *n*)。uplo = 'U' または 'u' と指定した場合は、このルーチンに入る前に、配列 *a* の先頭の  $(k+1) \times n$  の部分には、対称行列の上三角帯部分を格納しなければならない。この行列には、行列の主対角成分が配列の行  $(k+1)$  にくるように、また最初の優対角成分が行 *k* の位置 2 から始まるように、列ごとに与えなければならない。配列 *a* の左上の  $k \times k$  の三角は、参照されない。

次に示すプログラムの一部は、対称帯行列の上三角部分を通常のフル格納形式の行列から帯格納形式に転送するためのものである。

```
do 20, j = 1, n
  m = k + 1 - j
  do 10, i = max( 1, j - k ), j
    a( m + i, j ) = matrix( i, j )
  10 continue
20 continue
```

uplo = 'L' または 'l' と指定した場合は、このルーチンに入る前に、配列 *a* の先頭の  $(k+1) \times n$  の部分には、対称行列の下三角帯部分を格納しなければならない。この行列には、行列の主対角成分が配列の行 1 にくるように、また最初の劣対角成分が行 2 の位置 1 から始まるように、列ごとに与えなければならない。配列 *a* の右下の  $k \times k$  の三角は、参照されない。

次に示すプログラムの一部は、対称帯行列の下三角部分を通常のフル格納形式の行列から帯格納形式に転送するためのものである。

```
do 20, j = 1, n
  m = 1 - j
  do 10, i = j, min( n, j + k )
    a( m + i, j ) = matrix( i, j )
  10 continue
20 continue
```

*lda*          INTEGER。呼び出し元の ( サブ ) プログラムで宣言されているとおりに、*a* の第 1 次元を指定する。*lda* の値は、 $(k+1)$  以上でなければならない。

$x$	REAL (ssbmw の場合) DOUBLE PRECISION (dsbmw の場合)  配列、次元は $(1 + (n - 1) * \text{abs}(\text{incx}))$ 以上。このルーチンに入る前に、増分された配列 $x$ にベクトル $x$ を格納しなければならない。
$\text{incx}$	INTEGER。 $x$ の成分に対する増分を指定する。 $\text{incx}$ の値は、ゼロであってはならない。
$\text{beta}$	REAL (ssbmw の場合) DOUBLE PRECISION (dsbmw の場合)  スカラ $\text{beta}$ を指定する。
$y$	REAL (ssbmw の場合) DOUBLE PRECISION (dsbmw の場合)  配列、次元は $(1 + (n - 1) * \text{abs}(\text{incy}))$ 以上。このルーチンに入る前に、増分された配列 $y$ にベクトル $y$ を格納しなければならない。
$\text{incy}$	INTEGER。 $y$ の成分に対する増分を指定する。 $\text{incy}$ の値は、ゼロであってはならない。

### 出力パラメータ

$y$           更新されたベクトル  $y$  によって上書きされる。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[Fortran-95 インターフェイス規則](#)を参照のこと。

ルーチン sbmw のインターフェイスの詳細を以下に示す。

$a$	サイズ $(k+1, n)$ の配列 $A$ を格納する。
$x$	長さ $(n)$ のベクトルを格納する。
$y$	長さ $(n)$ のベクトルを格納する。
$\text{uplo}$	'U' または 'L' でなければならない。デフォルト値は 'U' である。
$\alpha$	デフォルト値は '1' である。
$\text{beta}$	デフォルト値は '1' である。

## ?spmv

圧縮形式の対称行列を使用して行列 - ベクトルの積を計算する。

### 構文

#### Fortran 77:

```
CALL sSPMV( UPLO, N, ALPHA, AP, X, INCX, BETA, Y, INCY )
CALL DSPMV( UPLO, N, ALPHA, AP, X, INCX, BETA, Y, INCY )
```

#### Fortran 95:

```
CALL spmv(a, x, y [,uplo] [,alpha] [,beta])
```

### 説明

?spmv ルーチンは、次のように定義される行列 - ベクトル演算を実行する。

$y := \alpha * a * x + \beta * y$

$\alpha$  と  $\beta$  は、スカラーである。

$x$  と  $y$  は、 $n$  個の成分を持つベクトルである。

$a$  は、圧縮形式で与えられる  $n \times n$  の対称行列である。

### 入力パラメータ

**uplo** CHARACTER\*1。次に示すように、行列  $a$  の上三角部分と下三角部分のどちらを圧縮形式の配列  $ap$  において与えるかを指定する。

uplo の値	与える行列 $a$ の部分
U または u	行列 $a$ の上三角部分を $ap$ において与える。
L または l	行列 $a$ の下三角部分を $ap$ において与える。

**n** INTEGER。行列  $a$  の次数を指定する。 $n$  の値は、ゼロ以上でなければならない。

**alpha** REAL (sspmv の場合 )  
DOUBLE PRECISION (dspmv の場合 )  
スカラー  $\alpha$  を指定する。

<i>ap</i>	REAL (sspmv の場合) DOUBLE PRECISION (dspmv の場合)  配列、次元は $((n*(n+1))/2)$ 以上。uplo = 'U' または 'u' と指定した場合は、このルーチンに入る前に、 <i>ap</i> (1) に <i>a</i> (1, 1) が、 <i>ap</i> (2) に <i>a</i> (1, 2) が、 <i>ap</i> (3) に <i>a</i> (2, 2) がそれぞれ格納されるように、配列 <i>ap</i> には、順番に圧縮された対称行列の上三角部分を列ごとに格納しなければならない。また、uplo = 'L' または 'l' と指定した場合は、このルーチンに入る前に、 <i>ap</i> (1) に <i>a</i> (1, 1) が <i>ap</i> (2) に <i>a</i> (2, 1) が、 <i>ap</i> (3) に <i>a</i> (3, 1) がそれぞれ格納されるように、配列 <i>ap</i> には、順番に圧縮された対称行列の下三角部分を列ごとに格納しなければならない。
<i>x</i>	REAL (sspmv の場合) DOUBLE PRECISION (dspmv の場合)  配列、次元は $(1 + (n - 1)*abs(incx))$ 以上。このルーチンに入る前に、増分された配列 <i>x</i> に <i>n</i> 個の成分を持つベクトル <i>x</i> を格納しなければならない。
<i>incx</i>	INTEGER。 <i>x</i> の成分に対する増分を指定する。 <i>incx</i> の値は、ゼロであってはならない。
<i>beta</i>	REAL (sspmv の場合) DOUBLE PRECISION (dspmv の場合)  スカラ <i>beta</i> を指定する。 <i>beta</i> をゼロに設定した場合は、 <i>y</i> を設定する必要はない。
<i>y</i>	REAL (sspmv の場合) DOUBLE PRECISION (dspmv の場合)  配列、次元は $(1 + (n - 1)*abs(incy))$ 以上。このルーチンに入る前に、増分された配列 <i>y</i> に <i>n</i> 個の成分を持つベクトル <i>y</i> を格納しなければならない。
<i>incy</i>	INTEGER。 <i>y</i> の成分に対する増分を指定する。 <i>incy</i> の値は、ゼロであってはならない。

## 出力パラメータ

*y*            更新されたベクトル *y* によって上書きされる。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[Fortran-95 インターフェイス規則](#)を参照のこと。

ルーチン `spmv` のインターフェイスの詳細を以下に示す。

<code>a</code>	サイズ $(n*(n+1)/2)$ の配列 <code>A</code> を格納する。
<code>x</code>	長さ $(n)$ のベクトルを格納する。
<code>y</code>	長さ $(n)$ のベクトルを格納する。
<code>uplo</code>	'U' または 'L' でなければならない。デフォルト値は 'U' である。
<code>alpha</code>	デフォルト値は '1' である。
<code>beta</code>	デフォルト値は '1' である。

---

### ?spr

圧縮形式の対称行列の階数1 の更新を実行する。

#### 構文

##### Fortran 77:

```
CALL SSPR( UPLO, N, ALPHA, X, INCX, AP )  
CALL dSPR( UPLO, N, ALPHA, X, INCX, AP )
```

##### Fortran 95:

```
CALL spr(a, x [,uplo] [,alpha])
```

#### 説明

?spr ルーチンは、次のように定義される行列 - ベクトル演算を実行する。

$a := \alpha * x * x' + a$

`alpha` は、実数のスカラーである。

`x` は、 $n$  個の成分を持つベクトルである。

`a` は、圧縮形式で与えられる  $n \times n$  の対称行列である。

## 入力パラメータ

`uplo` CHARACTER\*1。次に示すように、行列 `a` の上三角部分と下三角部分のどちらを圧縮形式の配列 `ap` において与えるかを指定する。

<code>uplo</code> の値	与える行列 <code>a</code> の部分
U または u	行列 <code>a</code> の上三角部分を <code>ap</code> において与える。
L または l	行列 <code>a</code> の下三角部分を <code>ap</code> において与える。

`n` INTEGER。行列 `a` の次数を指定する。`n` の値は、ゼロ以上でなければならない。

`alpha` REAL (sspr の場合)  
DOUBLE PRECISION (dspr の場合)  
スカラ `alpha` を指定する。

`x` REAL (sspr の場合)  
DOUBLE PRECISION (dspr の場合)

配列、次元は  $(1 + (n - 1) * \text{abs}(\text{incx}))$  以上。このルーチンに入る前に、増分された配列 `x` に `n` 個の成分を持つベクトル `x` を格納しなければならない。

`incx` INTEGER。`x` の成分に対する増分を指定する。`incx` の値は、ゼロであってはならない。

`ap` REAL (sspr の場合)  
DOUBLE PRECISION (dspr の場合)

配列、次元は  $((n * (n + 1)) / 2)$  以上。`uplo` = 'U' または 'u' と指定した場合は、このルーチンに入る前に、`ap(1)` に `a(1, 1)` が、`ap(2)` に `a(1, 2)` が、`ap(3)` に `a(2, 2)` がそれぞれ格納されるように、配列 `ap` には、順番に圧縮された対称行列の上三角部分を列ごとに格納しなければならない。

`uplo` = 'L' または 'l' と指定した場合は、このルーチンに入る前に、`ap(1)` に `a(1, 1)` が、`ap(2)` に `a(2, 1)` が、`ap(3)` に `a(3, 1)` がそれぞれ格納されるように、配列 `ap` には、順番に圧縮された対称行列の下三角部分を列ごとに格納しなければならない。

## 出力パラメータ

`ap` `uplo` = 'U' または 'u' と指定した場合は、更新された行列の上三角部分によって上書きされる。

`uplo` = 'L' または 'l' と指定した場合は、更新された行列の下三角部分によって上書きされる。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[Fortran-95 インターフェイス規則](#)を参照のこと。

ルーチン `spr` のインターフェイスの詳細を以下に示す。

`a`            サイズ  $(n*(n+1)/2)$  の配列 `A` を格納する。  
`x`            長さ  $(n)$  のベクトルを格納する。  
`uplo`        'U' または 'L' でなければならない。デフォルト値は 'U' である。  
`alpha`      デフォルト値は '1' である。

---

## ?spr2

圧縮形式の対称行列の階数2 の更新を実行する。

---

### 構文

#### Fortran 77:

```
CALL SSPR2( UPLO, N, ALPHA, X, INCX, Y, INCY, AP )  
CALL dSPR2( UPLO, N, ALPHA, X, INCX, Y, INCY, AP )
```

#### Fortran 95:

```
CALL spr2(a, x, y [,uplo] [,alpha])
```

### 説明

?spr2 ルーチンは、次のように定義される行列 - ベクトル演算を実行する。

$$a := \alpha * x * y' + \alpha * y * x' + a$$

`alpha` は、スカラーである。

`x` と `y` は、 $n$  個の成分を持つベクトルである。

`a` は、圧縮形式で与えられる  $n \times n$  の対称行列である。



## 入力パラメータ

`uplo` CHARACTER\*1。次に示すように、行列  $a$  の上三角部分と下三角部分のどちらを圧縮形式の配列  $ap$  において与えるかを指定する。

<code>uplo</code> の値	与える行列 $a$ の部分
U または u	行列 $a$ の上三角部分を $ap$ において与える。
L または l	行列 $a$ の下三角部分を $ap$ において与える。

`n` INTEGER。行列  $a$  の次数を指定する。`n` の値は、ゼロ以上でなければならない。

`alpha` REAL (sspr2 の場合)  
DOUBLE PRECISION (dspr2 の場合)  
スカラ `alpha` を指定する。

`x` REAL (sspr2 の場合)  
DOUBLE PRECISION (dspr2 の場合)

配列、次元は  $(1 + (n - 1) * \text{abs}(\text{incx}))$  以上。このルーチンに入る前に、増分された配列  $x$  に  $n$  個の成分を持つベクトル  $x$  を格納しなければならない。

`incx` INTEGER。 $x$  の成分に対する増分を指定する。`incx` の値は、ゼロであってはならない。

`y` REAL (sspr2 の場合)  
DOUBLE PRECISION (dspr2 の場合)

配列、次元は  $(1 + (n - 1) * \text{abs}(\text{incy}))$  以上。このルーチンに入る前に、増分された配列  $y$  に  $n$  個の成分を持つベクトル  $y$  を格納しなければならない。

`incy` INTEGER。 $y$  の成分に対する増分を指定する。`incy` の値は、ゼロであってはならない。

`ap` REAL (sspr2 の場合)  
DOUBLE PRECISION (dspr2 の場合)

配列、次元は  $((n * (n + 1)) / 2)$  以上。`uplo = 'U'` または `'u'` と指定した場合は、このルーチンに入る前に、`ap(1)` に  $a(1, 1)$  が、`ap(2)` に  $a(1, 2)$  が、`ap(3)` に  $a(2, 2)$  がそれぞれ格納されるように、配列  $ap$  には、順番に圧縮された対称行列の上三角部分を列ごとに格納しなければならない。

`uplo = 'L'` または `'l'` と指定した場合は、このルーチンに入る前に、`ap(1)` に  $a(1, 1)$  が、`ap(2)` に  $a(2, 1)$  が、`ap(3)` に  $a(3, 1)$  がそれぞれ格納されるように、配列  $ap$  には、順番に圧縮されたエルミート行列の下三角部分を列ごとに格納しなければならない。

### 出力パラメータ

`ap`            `uplo = 'U'` または `'u'` と指定した場合は、更新された行列の上三角部分によって上書きされる。

`uplo = 'L'` または `'l'` と指定した場合は、更新された行列の下三角部分によって上書きされる。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[Fortran-95 インターフェイス規則](#)を参照のこと。

ルーチン `spr2` のインターフェイスの詳細を以下に示す。

`a`            サイズ  $(n*(n+1)/2)$  の配列 `A` を格納する。

`x`            長さ  $(n)$  のベクトルを格納する。

`y`            長さ  $(n)$  のベクトルを格納する。

`uplo`        `'U'` または `'L'` でなければならない。デフォルト値は `'U'` である。

`alpha`      デフォルト値は `'1'` である。

---

## ?symv

対称行列に対して行列-ベクトルの積を計算する。

---

### 構文

#### Fortran 77:

```
ALL SSYMV( UPLO, N, ALPHA, A, LDA, X, INCX, BETA, Y, INCY )  
CALL dSYMV( UPLO, N, ALPHA, A, LDA, X, INCX, BETA, Y, INCY )
```

#### Fortran 95:

```
CALL symv(a, x, y [,uplo] [,alpha] [,beta])
```

## 説明

?symv ルーチンは、次のように定義される行列 - ベクトル演算を実行する。

$$y := \alpha * a * x + \beta * y$$

$\alpha$  と  $\beta$  は、スカラである。

$x$  と  $y$  は、 $n$  個の成分を持つベクトルである。

$a$  は、 $n \times n$  の対称行列である。

## 入力パラメータ

**uplo** CHARACTER\*1。次に示すように、配列  $a$  の上三角部分と下三角部分のどちらが参照されるかを指定する。

uplo の値	参照される配列 $a$ の部分
U または u	配列 $a$ の上三角部分が参照される。
L または l	配列 $a$ の下三角部分が参照される。

**n** INTEGER。行列  $a$  の次数を指定する。 $n$  の値は、ゼロ以上でなければならない。

**alpha** REAL (ssymv の場合)  
DOUBLE PRECISION (dsymv の場合)  
スカラ  $\alpha$  を指定する。

**a** REAL (ssymv の場合)  
DOUBLE PRECISION (dsymv の場合)

配列、次元は  $(lda, n)$ 。 $uplo = 'U'$  または  $'u'$  と指定した場合は、このルーチンに入る前に、配列  $a$  の先頭の  $n \times n$  の上三角部分には対称行列の上三角部分を格納しなければならない、 $a$  の厳密な下三角部分は参照されない。 $uplo = 'L'$  または  $'l'$  と指定した場合は、このルーチンに入る前に、配列  $a$  の先頭の  $n \times n$  の下三角部分には対称行列の下三角部分を格納しなければならない、 $a$  の厳密な上三角部分は参照されない。

**lda** INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 $a$  の第 1 次元を指定する。 $lda$  の値は、 $\max(1, n)$  以上でなければならない。

**x** REAL (ssymv の場合)  
DOUBLE PRECISION (dsymv の場合)

配列、次元は  $(1 + (n - 1) * \text{abs}(incx))$  以上。このルーチンに入る前に、増分された配列  $x$  に  $n$  個の成分を持つベクトル  $x$  を格納しなければならない。

<i>incx</i>	INTEGER。 <i>x</i> の成分に対する増分を指定する。 <i>incx</i> の値は、ゼロであってはならない。
<i>beta</i>	REAL (ssymv の場合) DOUBLE PRECISION (dsymv の場合)  スカラ <i>beta</i> を指定する。 <i>beta</i> をゼロに設定した場合は、 <i>y</i> を設定する必要はない。
<i>y</i>	REAL (ssymv の場合) DOUBLE PRECISION (dsymv の場合)  配列、次元は $(1 + (n - 1) * \text{abs}(\text{incy}))$ 以上。このルーチンに入る前に、増分された配列 <i>y</i> に <i>n</i> 個の成分を持つベクトル <i>y</i> を格納しなければならない。
<i>incy</i>	INTEGER。 <i>y</i> の成分に対する増分を指定する。 <i>incy</i> の値は、ゼロであってはならない。

### 出力パラメータ

*y*            更新されたベクトル *y* によって上書きされる。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[Fortran-95 インターフェイス規則](#)を参照のこと。

ルーチン symv のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ $(n, n)$ の行列 <i>A</i> を格納する。
<i>x</i>	長さ $(n)$ のベクトルを格納する。
<i>y</i>	長さ $(n)$ のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U' である。
<i>alpha</i>	デフォルト値は '1' である。
<i>beta</i>	デフォルト値は '1' である。

## ?syr

対称行列の階数1 の更新を実行する。

### 構文

#### Fortran 77:

```
CALL SSYR( UPLO, N, ALPHA, X, INCX, A, LDA )
CALL dSYR( UPLO, N, ALPHA, X, INCX, A, LDA )
```

#### Fortran 95:

```
CALL syr(a, x [,uplo] [,alpha])
```

### 説明

?syr ルーチンは、次のように定義される行列 - ベクトル演算を実行する。

$$a := \alpha * x * x' + a$$

$\alpha$  は、実数のスカラーである。

$x$  は、 $n$  個の成分を持つベクトルである。

$a$  は、 $n \times n$  の対称行列である。

### 入力パラメータ

**uplo** CHARACTER\*1。次に示すように、配列  $a$  の上三角部分と下三角部分のどちらが参照されるかを指定する。

uplo の値	参照される配列 $a$ の部分
U または u	配列 $a$ の上三角部分が参照される。
L または l	配列 $a$ の下三角部分が参照される。

**n** INTEGER。行列  $a$  の次数を指定する。 $n$  の値は、ゼロ以上でなければならない。

**alpha** REAL (ssyr の場合 )  
DOUBLE PRECISION (dsyr の場合 )  
スカラー  $\alpha$  を指定する。

$x$	REAL (ssyr の場合) DOUBLE PRECISION (dsyr の場合)  配列、次元は $(1 + (n - 1) * \text{abs}(\text{incx}))$ 以上。このルーチンに入る前に、増分された配列 $x$ に $n$ 個の成分を持つベクトル $x$ を格納しなければならない。
$\text{incx}$	INTEGER。 $x$ の成分に対する増分を指定する。 $\text{incx}$ の値は、ゼロであってはならない。
$a$	REAL (ssyr の場合) DOUBLE PRECISION (dsyr の場合)  配列、次元は $(\text{lda}, n)$ 。 $\text{uplo} = 'U'$ または $'u'$ と指定した場合は、このルーチンに入る前に、配列 $a$ の先頭の $n \times n$ の上三角部分には対称行列の上三角部分を格納しなければならない、 $a$ の厳密な下三角部分は参照されない。  また、 $\text{uplo} = 'L'$ または $'l'$ と指定した場合は、配列 $a$ の先頭の $n \times n$ の下三角部分には対称行列の下三角部分を格納しなければならない、 $a$ の厳密な上三角部分は参照されない。
$\text{lda}$	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 $a$ の第 1 次元を指定する。 $\text{lda}$ の値は、 $\max(1, n)$ 以上でなければならない。

### 出力パラメータ

$a$	$\text{uplo} = 'U'$ または $'u'$ と指定した場合は、配列 $a$ の上三角部分が、更新された行列の上三角部分によって上書きされる。  $\text{uplo} = 'L'$ または $'l'$ と指定した場合は、配列 $a$ の下三角部分が、更新された行列の下三角部分によって上書きされる。
-----	--

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[Fortran-95 インターフェイス規則](#)を参照のこと。

ルーチン `syr` のインターフェイスの詳細を以下に示す。

$a$	サイズ $(n, n)$ の行列 $A$ を格納する。
$x$	長さ $(n)$ のベクトルを格納する。
$\text{uplo}$	$'U'$ または $'L'$ でなければならない。デフォルト値は $'U'$ である。
$\alpha$	デフォルト値は $'1'$ である。

## ?syr2

対称行列の階数2 の更新を実行する。

### 構文

#### Fortran 77:

```
CALL SSYR2( UPLO, N, ALPHA, X, INCX, Y, INCY, A, LDA )
```

```
CALL dSYR2( UPLO, N, ALPHA, X, INCX, Y, INCY, A, LDA )
```

#### Fortran 95:

```
CALL syr2(a, x, y [,uplo] [,alpha])
```

### 説明

?syr2 ルーチンは、次のように定義される行列 - ベクトル演算を実行する。

$$a := \alpha * x * y' + \alpha * y * x' + a$$

$\alpha$  は、スカラーである。

$x$  と  $y$  は、 $n$  個の成分を持つベクトルである。

$a$  は、 $n \times n$  の対称行列である。

### 入力パラメータ

**uplo** CHARACTER\*1。次に示すように、配列  $a$  の上三角部分と下三角部分のどちらが参照されるかを指定する。

uplo の値	参照される配列 $a$ の部分
U または u	配列 $a$ の上三角部分が参照される。
L または l	配列 $a$ の下三角部分が参照される。

**n** INTEGER。行列  $a$  の次数を指定する。 $n$  の値は、ゼロ以上でなければならない。

**alpha** REAL (ssyr2 の場合 )  
DOUBLE PRECISION (dsyr2 の場合 )

スカラー  $\alpha$  を指定する。

$x$	REAL (ssyr2 の場合) DOUBLE PRECISION (dsyr2 の場合)  配列、次元は $(1 + (n - 1) * \text{abs}(\text{incx}))$ 以上。このルーチンに入る前に、増分された配列 $x$ に $n$ 個の成分を持つベクトル $x$ を格納しなければならない。
$\text{incx}$	INTEGER。 $x$ の成分に対する増分を指定する。 $\text{incx}$ の値は、ゼロであってはならない。
$y$	REAL (ssyr2 の場合) DOUBLE PRECISION (dsyr2 の場合)  配列、次元は $(1 + (n - 1) * \text{abs}(\text{incy}))$ 以上。このルーチンに入る前に、増分された配列 $y$ に $n$ 個の成分を持つベクトル $y$ を格納しなければならない。
$\text{incy}$	INTEGER。 $y$ の成分に対する増分を指定する。 $\text{incy}$ の値は、ゼロであってはならない。
$a$	REAL (ssyr2 の場合) DOUBLE PRECISION (dsyr2 の場合)  配列、次元は $(\text{lda}, n)$ 。 $\text{uplo} = 'U'$ または $'u'$ と指定した場合は、配列 $a$ の先頭の $n \times n$ の上三角部分には対称行列の上三角部分を格納しなければならない。  $\text{uplo} = 'L'$ または $'l'$ と指定した場合は、配列 $a$ の先頭の $n \times n$ の下三角部分には対称行列の下三角部分を格納しなければならない。  $\text{uplo} = 'U'$ または $'u'$ と指定した場合は、配列 $a$ の下三角部分は参照されない。  $\text{uplo} = 'L'$ または $'l'$ と指定した場合は、配列 $a$ の上三角部分は参照されない。
$\text{lda}$	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 $a$ の第 1 次元を指定する。 $\text{lda}$ の値は、 $\max(1, n)$ 以上でなければならない。

### 出力パラメータ

$a$	$\text{uplo} = 'U'$ または $'u'$ と指定した場合は、配列 $a$ の上三角部分が、更新された行列の上三角部分によって上書きされる。  $\text{uplo} = 'L'$ または $'l'$ と指定した場合は、配列 $a$ の下三角部分が、更新された行列の下三角部分によって上書きされる。
-----	--

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[Fortran-95 インターフェイス規則](#)を参照のこと。

ルーチン `syr2` のインターフェイスの詳細を以下に示す。



<i>a</i>	サイズ $(n, n)$ の行列 <i>A</i> を格納する。
<i>x</i>	長さ $(n)$ のベクトルを格納する。
<i>y</i>	長さ $(n)$ のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U' である。
<i>alpha</i>	デフォルト値は '1' である。

## ?tbmv

三角帯行列を使用して行列 - ベクトルの積を計算する。

### 構文

#### Fortran 77:

```
CALL sTBMV( UPLO, TRANS, DIAG, N, K, A, LDA, X, INCX )
CALL dTBMV( UPLO, TRANS, DIAG, N, K, A, LDA, X, INCX )
CALL cTBMV( UPLO, TRANS, DIAG, N, K, A, LDA, X, INCX )
CALL zTBMV( UPLO, TRANS, DIAG, N, K, A, LDA, X, INCX )
```

#### Fortran 95:

```
CALL tbmv(a, x [,uplo] [,trans] [,diag])
```

### 説明

?tbmv ルーチンは、次のように定義される行列 - ベクトル演算を実行する。

$x := a * x$ 、 $x := a' * x$ 、または  $x := \text{conjg}(a') * x$

*x* は、*n* 個の成分を持つベクトルである。

*a* は、 $n \times n$  単位あるいは非単位の、 $(k + 1)$  個の対角成分を持つ、上位または下位の三角帯行列である。

## 入力パラメータ

**uplo** CHARACTER\*1。次に示すように、行列が上三角行列と下三角行列のどちらであるかを指定する。

<i>uplo</i> の値	行列 <i>a</i>
U または u	上三角行列。
L または l	下三角行列。

**trans** CHARACTER\*1。次に示すように、実行する演算を指定する。

<i>trans</i> の値	実行する演算
N または n	$x := a * x$
T または t	$x := a' * x$
C または c	$x := \text{conjg}(a') * x$

**diag** CHARACTER\*1。次に示すように、*a* が単位三角行列であるかどうかを指定する。

<i>diag</i> の値	行列 <i>a</i>
U または u	行列 <i>a</i> は、単位三角行列とみなされる。
N または n	行列 <i>a</i> は、単位三角行列とはみなされない。

**n** INTEGER。行列 *a* の次数を指定する。*n* の値は、ゼロ以上でなければならない。

**k** INTEGER。*uplo* = 'U' または 'u' と指定した場合には、*k* には行列 *a* の優対角成分の数を指定する。*uplo* = 'L' または 'l' と指定した場合には、*k* には行列 *a* の劣対角成分の数を指定する。*k* の値は、 $0 \leq k$  を満たさなければならない。

**a** REAL (stbmv の場合)  
DOUBLE PRECISION (dtbmv の場合)  
COMPLEX (ctbmv の場合)  
DOUBLE COMPLEX (ztbmv の場合)

配列、次元は (*lda*, *n*)。*uplo* = 'U' または 'u' と指定した場合は、このルーチンに入る前に、配列 *a* の先頭の  $(k+1) \times n$  の部分には、係数行列の上三角帯部分を格納しなければならない。この行列には、行列の主対角成分が配列の行  $(k+1)$  にくるように、また最初の優対角成分が行 *k* の位置 2 から始まるように、列ごとに与えなければならない。配列 *a* の左上の  $k \times k$  の三角は、参照されない。次に示すプログラムの一部は、上三角帯行列を通常のフル格納形式の行列から帯格納形式に転送するためのものである。

```

do 20, j = 1, n
  m = k + 1 - j
  do 10, i = max(1, j - k), j
    a(m + i, j) = matrix(i, j)
  10 continue
20 continue

```

`uplo = 'L'` または `'l'` と指定した場合は、このルーチンに入る前に、配列 `a` の先頭の  $(k+1) \times n$  の部分には、係数行列の下三角帯部分を格納しなければならない。この行列には、行列の主対角成分が配列の行 1 にくるように、また最初の劣対角成分が行 2 の位置 1 から始まるように、列ごとに与えなければならない。配列 `a` の右下の  $k \times k$  の三角は、参照されない。次に示すプログラムの一部は、下三角帯行列を通常のフル格納形式の行列から帯格納形式に転送するためのものである。

```

do 20, j = 1, n
  m = 1 - j
  do 10, i = j, min(n, j + k)
    a(m + i, j) = matrix(i, j)
  10 continue
20 continue

```

`diag = 'U'` または `'u'` と指定した場合は、行列の対角成分に対応する配列 `a` の成分は参照されないが、1 とみなされる。

**lda** INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、`a` の第 1 次元を指定する。`lda` の値は、 $(k+1)$  以上でなければならない。

**x** REAL (stbmv の場合)  
 DOUBLE PRECISION (dtbmv の場合)  
 COMPLEX (ctbmv の場合)  
 DOUBLE COMPLEX (ztbmv の場合)

配列、次元は  $(1 + (n-1) * \text{abs}(\text{incx}))$  以上。このルーチンに入る前に、増分された配列 `x` に  $n$  個の成分を持つベクトル `x` を格納しなければならない。

**incx** INTEGER。`x` の成分に対する増分を指定する。`incx` の値は、ゼロであってはならない。

## 出力パラメータ

**x** 変換されたベクトル `x` によって上書きされる。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[Fortran-95 インターフェイス規則](#)を参照のこと。

ルーチン `tbmv` のインターフェイスの詳細を以下に示す。

<code>a</code>	サイズ $(k+1, n)$ の配列 $A$ を格納する。
<code>x</code>	長さ $(n)$ のベクトルを格納する。
<code>uplo</code>	'U' または 'L' でなければならない。デフォルト値は 'U' である。
<code>trans</code>	'N'、'C'、または 'T' でなければならない。デフォルト値は 'N' である。
<code>diag</code>	'N' または 'U' でなければならない。デフォルト値は 'N' である。

---

### ?tbsv

係数が三角帯行列に格納されている連立1次方程式を解く。

---

#### 構文

##### Fortran 77:

```
call stbsv( UPLO, TRANS, DIAG, N, K, A, LDA, X, INCX )
CALL dTBSV( UPLO, TRANS, DIAG, N, K, A, LDA, X, INCX )
CALL cTBSV( UPLO, TRANS, DIAG, N, K, A, LDA, X, INCX )
CALL zTBSV( UPLO, TRANS, DIAG, N, K, A, LDA, X, INCX )
```

##### Fortran 95:

```
CALL tbmv(a, x [,uplo] [,trans] [,diag])
```

#### 説明

?tbsv ルーチンは、次に示す連立方程式のいずれかを解く。

$a*x = b$ 、 $a'*x = b$ 、または  $\text{conjg}(a')*x = b$

$b$  と  $x$  は、 $n$  個の成分を持つベクトルである。

$a$  は、 $n \times n$  単位あるいは非単位の、 $(k+1)$  個の対角成分を持つ、上位または下位の三角帯行列である。

このルーチンは、特異点あるいは近似特異点はテストしない。この種のテストは、このルーチンを呼び出す前に実行しなければならない。

### 入力パラメータ

**uplo** CHARACTER\*1。次に示すように、行列が上三角行列と下三角行列のどちらであるかを指定する。

<b>uplo の値</b>	<b>行列 <math>a</math></b>
U または u	上三角行列。
L または l	下三角行列。

**trans** CHARACTER\*1。次に示すように、実行する演算を指定する。

<b>trans の値</b>	<b>実行する演算</b>
N または n	$a * x = b$
T または t	$a' * x = b$
C または c	$\text{conjg}(a') * x = b$

**diag** CHARACTER\*1。次に示すように、 $a$  が単位三角行列であるかどうかを指定する。

<b>diag の値</b>	<b>行列 <math>a</math></b>
U または u	行列 $a$ は、単位三角行列とみなされる。
N または n	行列 $a$ は、単位三角行列とはみなされない。

**n** INTEGER。行列  $a$  の次数を指定する。 $n$  の値は、ゼロ以上でなければならない。

**k** INTEGER。uplo = 'U' または 'u' と指定した場合のエントリでは、 $k$  は行列  $a$  の優対角成分の数を指定する。uplo = 'L' または 'l' と指定した場合のエントリでは、 $k$  は  $a$  の劣対角成分の数を指定する。 $k$  の値は、 $0 \leq k$  を満たさなければならない。

*a*            REAL (stbsv の場合)  
               DOUBLE PRECISION (dtbsv の場合)  
               COMPLEX (ctbsv の場合)  
               DOUBLE COMPLEX (ztbsv の場合)

配列、次元は (*lda*, *n*)。uplo = 'U' または 'u' と指定した場合は、このルーチンに入る前に、配列 *a* の先頭の  $(k+1) \times n$  の部分には、係数行列の上三角帯部分を格納しなければならない。この行列には、行列の主対角成分が配列の行  $(k+1)$  にくるように、また最初の優対角成分が行 *k* の位置 2 から始まるように、列ごとに与えなければならない。配列 *a* の左上の  $k \times k$  の三角は、参照されない。

次に示すプログラムの一部は、上三角帯行列を通常のフル格納形式の行列から帯格納形式に転送するためのものである。

```
do 20, j = 1, n
  m = k + 1 - j
  do 10, i = max(1, j - k), j
    a(m + i, j) = matrix (i, j)
  10 continue
20 continue
```

uplo = 'L' または 'l' と指定した場合は、このルーチンに入る前に、配列 *a* の先頭の  $(k+1) \times n$  の部分には、係数行列の下三角帯部分を格納しなければならない。この行列には、行列の主対角成分が配列の行 1 にくるように、また最初の劣対角成分が行 2 の位置 1 から始まるように、列ごとに与えなければならない。配列 *a* の右下の  $k \times k$  の三角は、参照されない。

次に示すプログラムの一部は、下三角帯行列を通常のフル格納形式の行列から帯格納形式に転送するためのものである。

```
do 20, j = 1, n
  m = 1 - j
  do 10, i = j, min(n, j + k)
    a(m + i, j) = matrix (i, j)
  10 continue
20 continue
```

diag = 'U' または 'u' と指定した場合、行列の対角成分に対応する配列 *a* の成分は参照されないが、1 とみなされる。

*lda*            INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、*a* の第 1 次元を指定する。*lda* の値は、 $(k+1)$  以上でなければならない。

$x$  REAL (stbsv の場合)  
 DOUBLE PRECISION (dtbsv の場合)  
 COMPLEX (ctbsv の場合)  
 DOUBLE COMPLEX (ztbsv の場合)

配列、次元は  $(1 + (n - 1) * \text{abs}(\text{incx}))$  以上。このルーチンに入る前に、増分された配列  $x$  に  $n$  個の成分からなる右辺のベクトル  $b$  を格納しなければならない。

$\text{incx}$  INTEGER。  $x$  の成分に対する増分を指定する。  $\text{incx}$  の値は、ゼロであってはならない。

### 出力パラメータ

$x$  解のベクトル  $x$  によって上書きされる。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[Fortran-95 インターフェイス規則](#)を参照のこと。

ルーチン tbsv のインターフェイスの詳細を以下に示す。

$a$  サイズ  $(k+1, n)$  の配列  $A$  を格納する。

$x$  長さ  $(n)$  のベクトルを格納する。

$\text{uplo}$  'U' または 'L' でなければならない。デフォルト値は 'U' である。

$\text{trans}$  'N'、'C'、または 'T' でなければならない。デフォルト値は 'N' である。

$\text{diag}$  'N' または 'U' でなければならない。デフォルト値は 'N' である。

## ?tpmv

圧縮形式の三角行列を使用して行列 - ベクトルの積を計算する。

### 構文

#### Fortran 77:

```
CALL sTPMV( UPLO, TRANS, DIAG, N, AP, X, INCX )
```

```
CALL dTPMV( UPLO, TRANS, DIAG, N, AP, X, INCX )
CALL cTPMV( UPLO, TRANS, DIAG, N, AP, X, INCX )
CALL zTPMV( UPLO, TRANS, DIAG, N, AP, X, INCX )
```

### Fortran 95:

```
CALL tpmv(a, x [,uplo] [,trans] [,diag])
```

### 説明

?tpmv ルーチンは、次のように定義される行列 - ベクトル演算を実行する。

$x := a * x$ 、 $x := a' * x$ 、または  $x := \text{conjg}(a') * x$

$x$  は、 $n$  個の成分を持つベクトルである。

$a$  は、 $n \times n$  単位あるいは非単位の、圧縮形式の上三角行列または下三角行列である。

### 入力パラメータ

**uplo** CHARACTER\*1。次に示すように、行列  $a$  が上三角行列と下三角行列のどちらであるかを指定する。

uplo の値	行列 $a$
U または u	上三角行列。
L または l	下三角行列。

**trans** CHARACTER\*1。次に示すように、実行する演算を指定する。

trans の値	実行する演算
N または n	$x := a * x$
T または t	$x := a' * x$
C または c	$x := \text{conjg}(a') * x$

**diag** CHARACTER\*1。次に示すように、 $a$  が単位三角行列であるかどうかを指定する。

diag の値	行列 $a$
U または u	行列 $a$ は、単位三角行列とみなされる。
N または n	行列 $a$ は、単位三角行列とはみなされない。

**n** INTEGER。行列  $a$  の次数を指定する。 $n$  の値は、ゼロ以上でなければならない。



- ap** REAL (stpmv の場合)  
DOUBLE PRECISION (dtpmv の場合)  
COMPLEX (ctpmv の場合)  
DOUBLE COMPLEX (ztpmv の場合)
- 配列、次元は  $((n*(n+1))/2)$  以上。uplo = 'U' または 'u' と指定した場合は、このルーチンに入る前に、ap(1) に a(1, 1) が、ap(2) に a(1, 2) が、ap(3) に a(2, 2) がそれぞれ格納されるように、配列 ap には、順番に圧縮された上三角行列を列ごとに格納しなければならない。また、uplo = 'L' または 'l' と指定した場合は、このルーチンに入る前に、ap(1) に a(1, 1) が、ap(2) に a(2, 1) が、ap(3) に a(3, 1) がそれぞれ格納されるように、配列 ap には、順番に圧縮された下三角行列を列ごとに格納しなければならない。diag = 'U' または 'u' と指定した場合、a の対角成分は参照されないが、1 とみなされる。
- x** REAL (stpmv の場合)  
DOUBLE PRECISION (dtpmv の場合)  
COMPLEX (ctpmv の場合)  
DOUBLE COMPLEX (ztpmv の場合)
- 配列、次元は  $(1 + (n-1)*abs(incx))$  以上。このルーチンに入る前に、増分された配列 x に n 個の成分を持つベクトル x を格納しなければならない。
- incx** INTEGER。x の成分に対する増分を指定する。incx の値は、ゼロであってはならない。

### 出力パラメータ

- x** 変換されたベクトル x によって上書きされる。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[Fortran-95 インターフェイス規則](#)を参照のこと。

ルーチン tpmv のインターフェイスの詳細を以下に示す。

- a** サイズ  $(n*(n+1)/2)$  の配列 A を格納する。
- x** 長さ (n) のベクトルを格納する。
- uplo** 'U' または 'L' でなければならない。デフォルト値は 'U' である。

*trans*        'N'、'C'、または 'T' でなければならない。デフォルト値は 'N' である。  
*diag*        'N' または 'U' でなければならない。デフォルト値は 'N' である。

---

### ?tpsv

係数が圧縮形式の三角行列に格納されている  
連立1次方程式を解く。

---

#### 構文

##### Fortran 77:

```
CALL sTPSV( UPLO, TRANS, DIAG, N, AP, X, INCX )  
CALL dTPSV( UPLO, TRANS, DIAG, N, AP, X, INCX )  
CALL cTPSV( UPLO, TRANS, DIAG, N, AP, X, INCX )  
CALL zTPSV( UPLO, TRANS, DIAG, N, AP, X, INCX )
```

##### Fortran 95:

```
CALL tpsv(a, x [,uplo] [,trans] [,diag])
```

#### 説明

?tpsv ルーチンは、次に示す連立方程式のいずれかを解く。

$a*x = b$ 、 $a'*x = b$ 、または  $\text{conjg}(a')*x = b$

$b$  と  $x$  は、 $n$  個の成分を持つベクトルである。

$a$  は、 $n \times n$  単位あるいは非単位の、圧縮格納形式の上三角行列または下三角行列である。

このルーチンは、特異点あるいは近似特異点はテストしない。この種のテストは、このルーチンを呼び出す前に実行しなければならない。

## 入力パラメータ

*uplo* CHARACTER\*1。次に示すように、行列 *a* が上三角行列と下三角行列のどちらであるかを指定する。

<i>uplo</i> の値	行列 <i>a</i>
U または u	上三角行列。
L または l	下三角行列。

*trans* CHARACTER\*1。次に示すように、実行する演算を指定する。

<i>trans</i> の値	実行する演算
N または n	$a * x = b$
T または t	$a' * x = b$
C または c	$\text{conjg}(a') * x = b$

*diag* CHARACTER\*1。次に示すように、*a* が単位三角行列であるかどうかを指定する。

<i>diag</i> の値	行列 <i>a</i>
U または u	行列 <i>a</i> は、単位三角行列とみなされる。
N または n	行列 <i>a</i> は、単位三角行列とはみなされない。

*n* INTEGER。行列 *a* の次数を指定する。*n* の値は、ゼロ以上でなければならない。

*ap* REAL (stpsv の場合 )  
 DOUBLE PRECISION (dtpsv の場合 )  
 COMPLEX (ctpsv の場合 )  
 DOUBLE COMPLEX (ztpsv の場合 )

配列、次元は  $((n*(n+1))/2)$  以上。*uplo* = 'U' または 'u' と指定した場合は、このルーチンに入る前に、*ap*(1) に *a*(1, 1) が、*ap*(2) に *a*(1, 2) が、*ap*(3) に *a*(2, 2) がそれぞれ格納されるように、配列 *ap* には、順番に圧縮された上三角行列を列ごとに格納しなければならない。また、*uplo* = 'L' または 'l' と指定した場合は、このルーチンに入る前に、*ap*(1) に *a*(1, 1) が、*ap*(2) に *a*(2, 1) が、*ap*(3) に *a*(3, 1) がそれぞれ格納されるように、配列 *ap* には、順番に圧縮された下三角行列を列ごとに格納しなければならない。*diag* = 'U' または 'u' と指定した場合、*a* の対角成分は参照されないが、1 とみなされる。

**x** REAL (stpsv の場合)  
DOUBLE PRECISION (dtpsv の場合)  
COMPLEX (ctpsv の場合)  
DOUBLE COMPLEX (ztpsv の場合)  
  
配列、次元は  $(1 + (n - 1) * \text{abs}(\text{incx}))$  以上。このルーチンに入る前に、増分された配列 **x** に *n* 個の成分からなる右辺のベクトル **b** を格納しなければならない。  
  
**incx** INTEGER。x の成分に対する増分を指定する。incx の値は、ゼロであってはならない。

### 出力パラメータ

**x** 解として得られたベクトル **x** によって上書きされる。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[Fortran-95 インターフェイス規則](#)を参照のこと。

ルーチン tpsv のインターフェイスの詳細を以下に示す。

**a** サイズ  $(n * (n + 1) / 2)$  の配列 **A** を格納する。  
**x** 長さ (*n*) のベクトルを格納する。  
**uplo** 'U' または 'L' でなければならない。デフォルト値は 'U' である。  
**trans** 'N'、'C'、または 'T' でなければならない。デフォルト値は 'N' である。  
**diag** 'N' または 'U' でなければならない。デフォルト値は 'N' である。

---

## ?trmv

三角行列を使用して行列-ベクトルの積を計算する。

---

### 構文

#### Fortran 77:

```
CALL sTRMV( UPLO, TRANS, DIAG, N, A, LDA, X, INCX )
```

```
CALL dTRMV( UPLO, TRANS, DIAG, N, A, LDA, X, INCX )
CALL cTRMV( UPLO, TRANS, DIAG, N, A, LDA, X, INCX )
CALL zTRMV( UPLO, TRANS, DIAG, N, A, LDA, X, INCX )
```

#### Fortran 95:

```
CALL trmv(a, x [,uplo] [,trans] [,diag])
```

#### 説明

?trmv ルーチンは、次のように定義される行列 - ベクトル演算のいずれかを実行する。

$x := a * x$ 、 $x := a' * x$ 、または  $x := \text{conjg}(a') * x$

$x$  は、 $n$  個の成分を持つベクトルである。

$a$  は、 $n \times n$  単位あるいは非単位の上三角行列または下三角行列である。

#### 入力パラメータ

**uplo** CHARACTER\*1。次に示すように、行列  $a$  が上三角行列と下三角行列のどちらであるかを指定する。

uplo の値	行列 $a$
U または u	上三角行列。
L または l	下三角行列。

**trans** CHARACTER\*1。次に示すように、実行する演算を指定する。

trans の値	実行する演算
N または n	$x := a * x$
T または t	$x := a' * x$
C または c	$x := \text{conjg}(a') * x$

**diag** CHARACTER\*1。次に示すように、 $a$  が単位三角行列であるかどうかを指定する。

diag の値	行列 $a$
U または u	行列 $a$ は、単位三角行列とみなされる。
N または n	行列 $a$ は、単位三角行列とはみなされない。

**n** INTEGER。行列  $a$  の次数を指定する。 $n$  の値は、ゼロ以上でなければならない。

- a** REAL (strmv の場合)  
DOUBLE PRECISION (dtrmv の場合)  
COMPLEX (ctrmv の場合)  
DOUBLE COMPLEX (ztrmv の場合)
- 配列、次元は  $(lda, n)$ 。uplo = 'U' または 'u' と指定した場合は、このルーチンに入る前に、配列 **a** の先頭の  $n \times n$  の上三角部分に上三角行列を格納しなければならない。また、uplo = 'L' または 'l' と指定した場合は、このルーチンに入る前に、配列 **a** の先頭の  $n \times n$  の下三角部分に下三角行列を格納しなければならない。また、**a** の厳密な上三角部分は参照されない。**diag** = 'U' または 'u' と指定した場合は、**a** の対角成分も参照されず、1 とみなされる。
- lda** INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、**a** の第 1 次元を指定する。**lda** の値は、 $\max(1, n)$  以上でなければならない。
- x** REAL (strmv の場合)  
DOUBLE PRECISION (dtrmv の場合)  
COMPLEX (ctrmv の場合)  
DOUBLE COMPLEX (ztrmv の場合)
- 配列、次元は  $(1 + (n - 1) * \text{abs}(\text{incx}))$  以上。このルーチンに入る前に、増分された配列 **x** に  $n$  個の成分を持つベクトル **x** を格納しなければならない。
- incx** INTEGER。**x** の成分に対する増分を指定する。**incx** の値は、ゼロであってはならない。

### 出力パラメータ

- x** 変換されたベクトル **x** によって上書きされる。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[Fortran-95 インターフェイス規則](#)を参照のこと。

ルーチン **trmv** のインターフェイスの詳細を以下に示す。

- a** サイズ  $(n, n)$  の行列 **A** を格納する。
- x** 長さ  $(n)$  のベクトルを格納する。
- uplo** 'U' または 'L' でなければならない。デフォルト値は 'U' である。

*trans* 'N'、'C'、または 'T' でなければならない。デフォルト値は 'N' である。

*diag* 'N' または 'U' でなければならない。デフォルト値は 'N' である。

## ?trsv

係数が三角行列に格納されている連立 1 次  
方程式を解く。

### 構文

#### Fortran 77:

```
call strsv( uplo, trans, diag, n, a, lda, x, incx )
call dtrsv( uplo, trans, diag, n, a, lda, x, incx )
call ctrsv( uplo, trans, diag, n, a, lda, x, incx )
call ztrsv( uplo, trans, diag, n, a, lda, x, incx )
```

#### Fortran 95:

```
CALL trsv(a, x [,uplo] [,trans] [,diag])
```

### 説明

?trsv ルーチンは、次に示す連立方程式のいずれかを解く。

$a \cdot x = b$ 、 $a' \cdot x = b$ 、または  $\text{conjg}(a') \cdot x = b$

$b$  と  $x$  は、 $n$  個の成分を持つベクトルである。

$a$  は、 $n \times n$  単位あるいは非単位の上三角行列または下三角行列である。

このルーチンは、特異点あるいは近似特異点はテストしない。この種のテストは、このルーチンを呼び出す前に実行しなければならない。

## 入力パラメータ

**uplo** CHARACTER\*1。次に示すように、行列が上三角行列と下三角行列のどちらであるかを指定する。

<b>uplo の値</b>	<b>行列 a</b>
U または u	上三角行列。
L または l	下三角行列。

**trans** CHARACTER\*1。次に示すように、実行する演算を指定する。

<b>trans の値</b>	<b>実行する演算</b>
N または n	$a * x = b$
T または t	$a' * x = b$
C または c	$\text{conjg}(a') * x = b$

**diag** CHARACTER\*1。次に示すように、a が単位三角行列であるかどうかを指定する。

<b>diag の値</b>	<b>行列 a</b>
U または u	行列 a は、単位三角行列とみなされる。
N または n	行列 a は、単位三角行列とはみなされない。

**n** INTEGER。行列 a の次数を指定する。n の値は、ゼロ以上でなければならない。

**a** REAL (strsv の場合)  
DOUBLE PRECISION (dtrsv の場合)  
COMPLEX (ctrsv の場合)  
DOUBLE COMPLEX (ztrsv の場合)

配列、次元は (lda, n)。uplo = 'U' または 'u' と指定した場合は、このルーチンに入る前に、配列 a の先頭の  $n \times n$  の上三角部分に上三角行列を格納しなければならず、a の厳密な下三角部分は参照されない。また、uplo = 'L' または 'l' と指定した場合は、このルーチンに入る前に、配列 a の先頭の  $n \times n$  の下三角部分に下三角行列を格納しなければならず、a の厳密な上三角部分は参照されない。diag = 'U' または 'u' と指定した場合は、a の対角成分も参照されず、1 とみなされる。

**lda** INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、a の第 1 次元を指定する。lda の値は、 $\max(1, n)$  以上でなければならない。



*x* REAL (strsv の場合)  
 DOUBLE PRECISION (dtrsv の場合)  
 COMPLEX (ctrsv の場合)  
 DOUBLE COMPLEX (ztrsv の場合)

配列、次元は  $(1 + (n - 1) * \text{abs}(\text{incx}))$  以上。このルーチンに入る前に、増分された配列 *x* に *n* 個の成分からなる右辺のベクトル *b* を格納しなければならない。

*incx* INTEGER。 *x* の成分に対する増分を指定する。 *incx* の値は、ゼロであってはならない。

### 出力パラメータ

*x* 解として得られたベクトル *x* によって上書きされる。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[Fortran-95 インターフェイス規則](#)を参照のこと。

ルーチン trsv のインターフェイスの詳細を以下に示す。

*a* サイズ  $(n, n)$  の行列 *A* を格納する。

*x* 長さ  $(n)$  のベクトルを格納する。

*uplo* 'U' または 'L' でなければならない。デフォルト値は 'U' である。

*trans* 'N'、'C'、または 'T' でなければならない。デフォルト値は 'N' である。

*diag* 'N' または 'U' でなければならない。デフォルト値は 'N' である。

### BLAS レベル 3 のルーチン

BLAS レベル 3 のルーチンは、行列 - 行列演算を実行する。表 2-3 に、BLAS レベル 3 のルーチンのグループと、それらに関連するデータ型を示す。

表 2-3 BLAS レベル 3 のルーチングループおよびそのデータ型

ルーチングループ	データ型	説明
<a href="#">?gemm</a>	s, d, c, z	一般行列での行列 - 行列の積
<a href="#">?hemm</a>	c, z	エルミート行列での行列 - 行列の積
<a href="#">?herk</a>	c, z	エルミート行列の階数 k の更新
<a href="#">?her2k</a>	c, z	エルミート行列の階数 2k の更新
<a href="#">?syrm</a>	s, d, c, z	対称行列での行列 - 行列の積
<a href="#">?syrk</a>	s, d, c, z	対称行列の階数 k の更新
<a href="#">?syr2k</a>	s, d, c, z	対称行列の階数 2k の更新
<a href="#">?trmm</a>	s, d, c, z	三角行列での行列 - 行列の積
<a href="#">?trsm</a>	s, d, c, z	三角行列に対する行列 - 行列による 1 次方程式の解

### インテル® MKL の対称型マルチプロセッシング・バージョン

多くのアプリケーションでは、BLAS レベル 3 のルーチンを実行するのに多くの時間を費やしている。インテル MKL に組み込まれている対称型マルチプロセッシング (SMP) 機能を使用すると、システム上で利用可能なプロセッサの数に応じて、この時間を短縮できる。ユーザ側でプログラミングしなくても、プロセッサの並行使用に基づいて性能を向上できる。

MKL は、次の手法を使用して、性能の向上を実現している。

- BLAS レベル 3 の行列 - 行列関数による演算により、データの参照をローカルで実行し、キャッシュ・メモリの利用度を改善し、メモリバスへの依存度を低減する方法でコードを再構築できる。
- 上に説明した方法でコードが効率よくブロック化された後、行列の 1 つが複数のプロセッサに分配され、2 番目の行列で乗算される。このような分配により、キャッシュ管理の効率が改善され、メモリバスへの依存度が低減され、プロセッサ数に応じた優れた結果が得られる。

---

## ?gemm

スカラ - 行列 - 行列の積を計算し、その結果を  
スカラ - 行列の積に加える。

---

### 構文

#### Fortran 77:

```
CALL sGEMM(TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB, BETA, C, LDC)
CALL dGEMM(TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB, BETA, C, LDC)
CALL cGEMM(TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB, BETA, C, LDC)
CALL zGEMM(TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB, BETA, C, LDC)
```

#### Fortran 95:

```
call gemm(a, b, c [,transa] [,transb] [,alpha] [,beta])
```

### 説明

?gemm ルーチンは、一般行列を使用して行列 - 行列演算を実行する。この演算は、次のように定義される。

$$c := \alpha * \text{op}(a) * \text{op}(b) + \beta * c$$

$\text{op}(x)$  は、 $\text{op}(x) = x$ 、 $\text{op}(x) = x'$ 、または  $\text{op}(x) = \text{conjg}(x')$  のいずれかである。

$\alpha$  と  $\beta$  は、スカラである。

$a$ 、 $b$ 、および  $c$  は、行列である。

$\text{op}(a)$  は、 $m \times k$  の行列である。

$\text{op}(b)$  は、 $k \times n$  の行列である。

$c$  は、 $m \times n$  の行列である。

### 入力パラメータ

*transa* CHARACTER\*1。次に示すように、行列の乗算で使用する  $\text{op}(a)$  の形式を指定する。

<i>transa</i> の値	$\text{op}(a)$ の形式
N または n	$\text{op}(a) = a$
T または t	$\text{op}(a) = a'$
C または c	$\text{op}(a) = \text{conjg}(a')$

*transb* CHARACTER\*1。次に示すように、行列の乗算で使用する  $\text{op}(b)$  の形式を指定する。

<i>transb</i> の値	$\text{op}(b)$ の形式
N または n	$\text{op}(b) = b$
T または t	$\text{op}(b) = b'$
C または c	$\text{op}(b) = \text{conjg}(b')$

*m* INTEGER。行列  $\text{op}(a)$  の行数と行列 *c* の行数を指定する。*m* の値は、ゼロ以上でなければならない。

*n* INTEGER。行列  $\text{op}(b)$  の列数と行列 *c* の列数を指定する。*n* の値はゼロ以上でなければならない。

*k* INTEGER。行列  $\text{op}(a)$  の列数と行列  $\text{op}(b)$  の行数を指定する。*k* の値はゼロ以上でなければならない。

*alpha* REAL (sgemm の場合)  
 DOUBLE PRECISION (dgemm の場合)  
 COMPLEX (cgemm の場合)  
 DOUBLE COMPLEX (zgemm の場合)  
 スカラ *alpha* を指定する。

- a** REAL (sgemm の場合 )  
 DOUBLE PRECISION (dgemm の場合 )  
 COMPLEX (cgemm の場合 )  
 DOUBLE COMPLEX (zgemm の場合 )
- 配列、次元は  $(lda, ka)$ 。  $ka$  は  $transa = 'N'$  または  $'n'$  の場合は  $k$  に、そうでない場合は  $m$  になる。  $transa = 'N'$  または  $'n'$  と指定した場合は、このルーチンに入る前に、配列  $a$  の先頭の  $m \times k$  の部分に行列  $a$  を格納しなければならない。そうでない場合は、配列  $a$  の先頭の  $k \times m$  の部分に行列  $a$  を格納しなければならない。
- lda** INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 $a$  の第 1 次元を指定する。  $transa = 'N'$  または  $'n'$  の場合、 $lda$  は  $\max(1, m)$  以上でなければならない。そうでない場合、 $lda$  は  $\max(1, k)$  以上でなければならない。
- b** REAL (sgemm の場合 )  
 DOUBLE PRECISION (dgemm の場合 )  
 COMPLEX (cgemm の場合 )  
 DOUBLE COMPLEX (zgemm の場合 )
- 配列、次元は  $(ldb, kb)$ 。  $kb$  は  $transb = 'N'$  または  $'n'$  の場合は  $n$  に、そうでない場合は  $k$  になる。  $transb = 'N'$  または  $'n'$  と指定した場合は、このルーチンに入る前に、配列  $b$  の先頭の  $k \times n$  の部分に行列  $b$  を格納しなければならない。そうでない場合は、配列  $b$  の先頭の  $n \times k$  の部分に行列  $b$  を格納しなければならない。
- ldb** INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 $b$  の第 1 次元を指定する。  $transb = 'N'$  または  $'n'$  の場合、 $ldb$  は  $\max(1, k)$  以上でなければならない。そうでない場合、 $ldb$  は  $\max(1, n)$  以上でなければならない。
- beta** REAL (sgemm の場合 )  
 DOUBLE PRECISION (dgemm の場合 )  
 COMPLEX (cgemm の場合 )  
 DOUBLE COMPLEX (zgemm の場合 )
- スカラー  $beta$  を指定する。  $beta$  にゼロを設定した場合には、 $c$  を設定する必要はない。

- c* REAL (sgemm の場合)  
DOUBLE PRECISION (dgemm の場合)  
COMPLEX (cgemm の場合)  
DOUBLE COMPLEX (zgemm の場合)
- 配列、次元は (*ldc*, *n*)。このルーチンに入る前に、配列 *c* の先頭の  $m \times n$  の部分に行列 *c* を格納しなければならない。ただし、*beta* がゼロの場合は、*c* を設定する必要はない。
- ldc* INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、*c* の第 1 次元を指定する。*ldc* の値は、 $\max(1, m)$  以上でなければならない。

### 出力パラメータ

- c*  $m \times n$  の行列 ( $\alpha * \text{op}(a) * \text{op}(b) + \beta * c$ ) によって上書きされる。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[Fortran-95 インターフェイス規則](#)を参照のこと。

ルーチン *gemm* のインターフェイスの詳細を以下に示す。

- a* サイズ (*ma*, *ka*) の行列 *A* を格納する。ここで、  
*transa* = 'N' の場合は *ka* = *k* に、それ以外の場合は *ka* = *m* になる。  
*transa* = 'N' の場合は *ma* = *m* に、それ以外の場合は *ma* = *k* になる。
- b* サイズ (*mb*, *kb*) の行列 *B* を格納する。ここで、  
*transb* = 'N' の場合 *kb* = *n* にそれ以外の場合は *kb* = *k* になる。  
*transb* = 'N' の場合 *mb* = *k* にそれ以外の場合は *mb* = *n* になる。
- c* サイズ (*m*, *n*) の行列 *C* を格納する。
- transa* 'N'、'C'、または 'T' でなければならない。デフォルト値は 'N' である。
- transb* 'N'、'C'、または 'T' でなければならない。デフォルト値は 'N' である。
- alpha* デフォルト値は '1' である。
- beta* デフォルト値は '1' である。

## ?hemm

スカラ - 行列 - 行列の積を計算し ( 行列オペランドのいずれかはエルミート行列)、その結果をスカラ - 行列の積に加える。

### 構文

#### Fortran 77:

```
call CHEMM( SIDE, UPLO, M, N, ALPHA, A, LDA, B, LDB, BETA, C, LDC )
call zHEMM( SIDE, UPLO, M, N, ALPHA, A, LDA, B, LDB, BETA, C, LDC )
```

#### Fortran 95:

```
call hemm(a, b, c [,side] [,uplo] [,alpha] [,beta])
```

### 説明

?hemm ルーチンは、エルミート行列を使用して行列 - 行列演算を実行する。この演算は、次のように定義される。

$$c := \alpha * a * b + \beta * c$$

または

$$c := \alpha * b * a + \beta * c$$

$\alpha$  と  $\beta$  は、スカラである。

$a$  は、エルミート行列である。

$b$  と  $c$  は、 $m \times n$  の行列である。

### 入力パラメータ

*side* CHARACTER\*1。次に示すように、演算でエルミート行列  $a$  が左と右のどちらにくるかを指定する。

<i>side</i> の値	実行する演算
L または l	$c := \alpha * a * b + \beta * c$
R または r	$c := \alpha * b * a + \beta * c$

**uplo** CHARACTER\*1。次に示すように、エルミート行列 *a* の上三角部分と下三角部分のどちらが参照されるかを指定する。

<i>uplo</i> の値	参照される行列 <i>a</i> の部分
U または u	エルミート行列の上三角部分だけが参照される。
L または l	エルミート行列の下三角部分だけが参照される。

**m** INTEGER。行列 *c* の行数を指定する。*m* の値は、ゼロ以上でなければならない。

**n** INTEGER。行列 *c* の列数を指定する。*n* の値はゼロ以上でなければならない。

**alpha** COMPLEX (chemm の場合)  
DOUBLE COMPLEX (zhemm の場合)  
スカラ *alpha* を指定する。

**a** COMPLEX (chemm の場合)  
DOUBLE COMPLEX (zhemm の場合)

配列、次元は (*lda*, *ka*)。 *ka* は *side* = 'L' または 'l' の場合に *m* に、そうでない場合は *n* になる。 *side* = 'L' または 'l' と指定した場合は、このルーチンに入る前に、次のように、配列 *a* の *m* × *m* の部分にエルミート行列を格納しなければならない。すなわち、*uplo* = 'U' または 'u' の場合は、配列 *a* の先頭の *m* × *m* の上三角部分にエルミート行列の上三角部分を格納しなければならない。また *a* の厳密な下三角部分は参照されない。 *uplo* = 'L' または 'l' の場合は、配列 *a* の先頭の *m* × *m* の下三角部分にエルミート行列の下三角部分を格納しなければならない。また *a* の厳密な上三角部分は参照されない。

*side* = 'R' または 'r' と指定した場合は、このルーチンに入る前に、次のように、配列 *a* の *n* × *n* の部分にエルミート行列を格納しなければならない。すなわち、*uplo* = 'U' または 'u' の場合は、配列 *a* の先頭の *n* × *n* の上三角部分にエルミート行列の上三角部分を格納しなければならない。また *a* の厳密な下三角部分は参照されない。 *uplo* = 'L' または 'l' の場合は、配列 *a* の先頭の *n* × *n* の下三角部分にエルミート行列の下三角部分を格納しなければならない。また *a* の厳密な上三角部分は参照されない。対角成分の虚数部は設定する必要はない。これらは、ゼロであるとみなされる。

**lda** INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、*a* の第 1 次元を指定する。 *side* = 'L' または 'l' の場合、*lda* は max(1, *m*) 以上でなければならない。そうでない場合、*lda* は max(1, *n*) 以上でなければならない。



<i>b</i>	COMPLEX (chemm の場合 ) DOUBLE COMPLEX (zhemm の場合 )  配列、次元は ( <i>ldb</i> , <i>n</i> )。このルーチンに入る前に、配列 <i>b</i> の先頭の $m \times n$ の部分に行列 <i>b</i> を格納しなければならない。
<i>ldb</i>	INTEGER。呼び出し元の ( サブ ) プログラムで宣言されているとおりに、 <i>b</i> の第 1 次元を指定する。 <i>ldb</i> の値は、 $\max(1, m)$ 以上でなければならない。
<i>beta</i>	COMPLEX (chemm の場合 ) DOUBLE COMPLEX (zhemm の場合 )  スカラ <i>beta</i> を指定する。 <i>beta</i> にゼロを設定した場合には、 <i>c</i> を設定する必要はない。
<i>c</i>	COMPLEX (chemm の場合 ) DOUBLE COMPLEX (zhemm の場合 )  配列、次元は ( <i>c</i> , <i>n</i> )。このルーチンに入る前に、配列 <i>c</i> の先頭の $m \times n$ の部分に行列 <i>c</i> を格納しなければならない。ただし、 <i>beta</i> がゼロの場合は、 <i>c</i> を設定する必要はない。
<i>ldc</i>	INTEGER。呼び出し元の ( サブ ) プログラムで宣言されているとおりに、 <i>c</i> の第 1 次元を指定する。 <i>ldc</i> の値は、 $\max(1, m)$ 以上でなければならない。

## 出力パラメータ

<i>c</i>	更新された $m \times n$ の行列によって上書きされる。
----------	-----------------------------------

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[Fortran-95 インターフェイス規則](#)を参照のこと。

ルーチン hemm のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ ( <i>k</i> , <i>k</i> ) の行列 <i>A</i> を格納する。ここで <i>side</i> = 'L' の場合 $k = m$ に、それ以外の場合は $k = n$ になる。
<i>b</i>	サイズ ( <i>m</i> , <i>n</i> ) の行列 <i>B</i> を格納する。
<i>c</i>	サイズ ( <i>m</i> , <i>n</i> ) の行列 <i>C</i> を格納する。
<i>side</i>	'L' または 'R' でなければならない。デフォルト値は 'L' である。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U' である。

*alpha* デフォルト値は '1' である。

*beta* デフォルト値は '1' である。

---

### ?herk

エルミート行列の階数  $n$  の更新を実行する。

---

#### 構文

##### Fortran 77:

```
CALL CHERK( UPLO, TRANS, N, K, ALPHA, A, LDA, BETA, C, LDC )  
CALL ZHERK( UPLO, TRANS, N, K, ALPHA, A, LDA, BETA, C, LDC )
```

##### Fortran 95:

```
call herk(a, c [,uplo] [,trans] [,alpha] [,beta])
```

#### 説明

?herk ルーチンは、エルミート行列を使用して行列 - 行列演算を実行する。この演算は、次のように定義される。

$$c := \alpha * a * \text{conjg}(a') + \beta * c$$

または

$$c := \alpha * \text{conjg}(a') * a + \beta * c$$

*alpha* と *beta* は、スカラーである。

*c* は、 $n \times n$  のエルミート行列である。

*a* は、1 番目の定義では  $n \times k$  の行列、2 番目の定義では  $k \times n$  の行列である。

#### 入力パラメータ

*uplo* CHARACTER\*1。次に示すように、配列 *c* の上三角部分と下三角部分のどちらが参照されるかを指定する。

<i>uplo</i> の値	参照される行列 <i>c</i> の部分
U または u	<i>c</i> の上三角部分だけが参照される。
L または l	<i>c</i> の下三角部分だけが参照される。

*trans* CHARACTER\*1。次に示すように、実行する演算を指定する。

<i>trans</i> の値	実行する演算
N または n	$c := \alpha * a * \text{conjg}(a') + \beta * c$
C または c	$c := \alpha * \text{conjg}(a') * a + \beta * c$

*n* INTEGER。行列 *c* の次数を指定する。*n* の値はゼロ以上でなければならない。

*k* INTEGER。*trans* = 'N' または 'n' と指定した場合は、*k* には行列 *a* の列数を指定する。また、*trans* = 'C' または 'c' と指定した場合は、*k* には行列 *a* の行数を指定する。*k* の値はゼロ以上でなければならない。

*alpha* REAL (cherk の場合)  
DOUBLE PRECISION (zherk の場合)  
スカラ *alpha* を指定する。

*a* COMPLEX (cherk の場合)  
DOUBLE COMPLEX (zherk の場合)  
配列、次元は (*lda*, *ka*)。 *ka* は *trans* = 'N' または 'n' の場合は *k* に、そうでない場合は *n* になる。*trans* = 'N' または 'n' と指定した場合は、このルーチンに入る前に、配列 *a* の先頭の  $n \times k$  の部分に行列 *a* を格納しなければならない。そうでない場合は、配列 *a* の先頭の  $k \times n$  の部分に行列 *a* を格納しなければならない。

*lda* INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、*a* の第 1 次元を指定する。*trans* = 'N' または 'n' の場合、*lda* は  $\max(1, n)$  以上でなければならない。そうでない場合、*lda* は  $\max(1, k)$  以上でなければならない。

*beta* REAL (cherk の場合)  
DOUBLE PRECISION (zherk の場合)  
スカラ *beta* を指定する。

- c*            COMPLEX (cherk の場合 )  
               DOUBLE COMPLEX (zherk の場合 )
- 配列、次元は (*ldc*, *n*)。 *uplo* = 'U' または 'u' と指定した場合は、このルーチンに入る前に、配列 *c* の先頭の  $n \times n$  の上三角部分にエルミート行列の上三角部分を格納しなければならない、また *c* の厳密な下三角部分は参照されない。
- uplo* = 'L' または 'l' と指定した場合は、このルーチンに入る前に、配列 *c* の先頭の  $n \times n$  の下三角部分にエルミート行列の下三角部分を格納しなければならない、また *c* の厳密な上三角部分は参照されない。
- 対角成分の虚数部は設定する必要はない。これらは、ゼロであるとみなされる。
- ldc*            INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、*c* の第 1 次元を指定する。*ldc* の値は、 $\max(1, n)$  以上でなければならない。

### 出力パラメータ

- c*            *uplo* = 'U' または 'u' と指定した場合は、配列 *c* の上三角部分が、更新された行列の上三角部分によって上書きされる。
- uplo* = 'L' または 'l' と指定した場合は、配列 *c* の下三角部分が、更新された行列の下三角部分によって上書きされる。
- 対角成分の虚数部は、ゼロに設定される。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[Fortran-95 インターフェイス規則](#)を参照のこと。

ルーチン *herk* のインターフェイスの詳細を以下に示す。

- a*            サイズ (*ma*, *ka*) の行列 *A* を格納する。ここで、  
               *transa* = 'N' の場合は *ka* = *k* に、それ以外の場合は *ka* = *n* になる。  
               *transa* = 'N' の場合は *ma* = *n* に、それ以外の場合は *ma* = *k* になる。
- c*            サイズ (*n*, *n*) の行列 *C* を格納する。
- uplo*        'U' または 'L' でなければならない。デフォルト値は 'U' である。
- trans*        'N' または 'C' でなければならない。デフォルト値は 'N' である。
- alpha*      デフォルト値は '1' である。
- beta*        デフォルト値は '1' である。

## ?her2k

エルミート行列の階数  $2k$  の更新を実行する。

### 構文

#### Fortran 77:

```
CALL CHER2K( UPLO, TRANS, N, K, ALPHA, A, LDA, b, ldb, BETA, C, LDC )
CALL ZHER2K( UPLO, TRANS, N, K, ALPHA, A, LDA, b, ldb, BETA, C, LDC )
```

#### Fortran 95:

```
call her2k(a, b, c [,uplo] [,trans] [,alpha] [,beta])
```

### 説明

?her2k ルーチンは、エルミート行列を使用して階数  $2k$  の行列 - 行列演算を実行する。この演算は、次のように定義される。

$$c := \alpha * a * \text{conjg}(b') + \text{conjg}(\alpha) * b * \text{conjg}(a') + \beta * c$$

または

$$c := \alpha * \text{conjg}(b') * a + \text{conjg}(\alpha) * \text{conjg}(a') * b + \beta * c$$

$\alpha$  はスカラー、 $\beta$  は実数のスカラーである。

$c$  は、 $n \times n$  のエルミート行列である。

$a$  と  $b$  は、1 番目の定義では  $n \times k$  の行列、2 番目の定義では  $k \times n$  の行列である。

### 入力パラメータ

**uplo** CHARACTER\*1。次に示すように、配列  $c$  の上三角部分と下三角部分のどちらが参照されるかを指定する。

uplo の値	参照される行列 $c$ の部分
U または u	$c$ の上三角部分だけが参照される。
L または l	$c$ の下三角部分だけが参照される。

*trans* CHARACTER\*1。次に示すように、実行する演算を指定する。

<i>trans</i> の値	実行する演算
N または n	$c := \alpha * a * \text{conjg}(b') + \alpha * b * \text{conjg}(a') + \beta * c$
C または c	$c := \alpha * \text{conjg}(a') * b + \alpha * \text{conjg}(b') * a + \beta * c$

*n* INTEGER。行列 *c* の次数を指定する。*n* の値はゼロ以上でなければならない。

*k* INTEGER。*trans* = 'N' または 'n' と指定した場合は、*k* には行列 *a* の列数を指定する。また、*trans* = 'C' または 'c' と指定した場合は、*k* には行列 *a* の行数を指定する。*k* の値はゼロ以上でなければならない。

*alpha* COMPLEX (cher2k の場合)  
DOUBLE COMPLEX (zher2k の場合)  
スカラ *alpha* を指定する。

*a* COMPLEX (cher2k の場合)  
DOUBLE COMPLEX (zher2k の場合)  
配列、次元は (*lda*, *ka*)。*ka* は *trans* = 'N' または 'n' の場合は *k* に、そうでない場合は *n* になる。*trans* = 'N' または 'n' と指定した場合は、このルーチンに入る前に、配列 *a* の先頭の *n* × *k* の部分に行列 *a* を格納しなければならない。そうでない場合は、配列 *a* の先頭の *k* × *n* の部分に行列 *a* を格納しなければならない。

*lda* INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、*a* の第 1 次元を指定する。*trans* = 'N' または 'n' の場合、*lda* は max(1, *n*) 以上でなければならない。そうでない場合、*lda* は max(1, *k*) 以上でなければならない。

*beta* REAL (cher2k の場合)  
DOUBLE PRECISION (zher2k の場合)  
スカラ *beta* を指定する。

- b**            COMPLEX (cher2k の場合 )  
               DOUBLE COMPLEX (zher2k の場合 )
- 配列、次元は  $(ldb, kb)$ 。  $kb$  は  $trans = 'N'$  または  $'n'$  の場合は  $k$  に、そうでない場合は  $n$  になる。  $trans = 'N'$  または  $'n'$  と指定した場合は、このルーチンに入る前に、配列  $b$  の先頭の  $n \times k$  の部分に行列  $b$  を格納しなければならない。そうでない場合は、配列  $b$  の先頭の  $k \times n$  の部分に行列  $b$  を格納しなければならない。
- ldb**            INTEGER。呼び出し元の ( サブ ) プログラムで宣言されているとおりに、 $b$  の第 1 次元を指定する。  $trans = 'N'$  または  $'n'$  の場合、 $ldb$  は  $\max(1, n)$  以上でなければならない。そうでない場合、 $ldb$  は  $\max(1, k)$  以上でなければならない。
- c**            COMPLEX (cher2k の場合 )  
               DOUBLE COMPLEX (zher2k の場合 )
- 配列、次元は  $(ldc, n)$ 。  $uplo = 'U'$  または  $'u'$  と指定した場合は、このルーチンに入る前に、配列  $c$  の先頭の  $n \times n$  の上三角部分にエルミート行列の上三角部分を格納しなければならず、また  $c$  の厳密な下三角部分は参照されない。
- $uplo = 'L'$  または  $'l'$  と指定した場合は、このルーチンに入る前に、配列  $c$  の先頭の  $n \times n$  の下三角部分にエルミート行列の下三角部分を格納しなければならず、また  $c$  の厳密な上三角部分は参照されない。
- 対角成分の虚数部は設定する必要はない。これらは、ゼロであるとみなされる。
- ldc**            INTEGER。呼び出し元の ( サブ ) プログラムで宣言されているとおりに、 $c$  の第 1 次元を指定する。  $ldc$  の値は、 $\max(1, n)$  以上でなければならない。

## 出力パラメータ

- c**             $uplo = 'U'$  または  $'u'$  と指定した場合は、配列  $c$  の上三角部分が、更新された行列の上三角部分によって上書きされる。
- $uplo = 'L'$  または  $'l'$  と指定した場合は、配列  $c$  の下三角部分が、更新された行列の下三角部分によって上書きされる。
- 対角成分の虚数部は、ゼロに設定される。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[Fortran-95 インターフェイス規則](#)を参照のこと。

ルーチン `her2k` のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ $(ma, ka)$ の行列 <i>A</i> を格納する。ここで、 <i>trans</i> = 'N' の場合は $ka = k$ に、それ以外の場合は $ka = n$ になる。 <i>trans</i> = 'N' の場合は $ma = n$ に、それ以外の場合は $ma = k$ になる。
<i>b</i>	サイズ $(mb, kb)$ の行列 <i>B</i> を格納する。ここで、 <i>trans</i> = 'N' の場合 $kb = k$ に、それ以外の場合は $kb = n$ になる。 <i>trans</i> = 'N' の場合 $mb = n$ に、それ以外の場合は $mb = k$ になる。
<i>c</i>	サイズ $(n, n)$ の行列 <i>C</i> を格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U' である。
<i>trans</i>	'N' または 'C' でなければならない。デフォルト値は 'N' である。
<i>alpha</i>	デフォルト値は '1' である。
<i>beta</i>	デフォルト値は '1' である。

---

## ?symm

スカラー - 行列 - 行列の積( 行列オペランドのいずれかは対称行列) を計算し、その結果をスカラー - 行列の積に加える。

---

### 構文

#### Fortran 77:

```
CALL sSYMM( SIDE, UPLO, M, N, ALPHA, A, LDA, B, LDB, BETA, C, LDC )
CALL dSYMM( SIDE, UPLO, M, N, ALPHA, A, LDA, B, LDB, BETA, C, LDC )
CALL cSYMM( SIDE, UPLO, M, N, ALPHA, A, LDA, B, LDB, BETA, C, LDC )
CALL zSYMM( SIDE, UPLO, M, N, ALPHA, A, LDA, B, LDB, BETA, C, LDC )
```

#### Fortran 95:

```
call symm(a, b, c [,side] [,uplo] [,alpha] [,beta])
```

### 説明

?symm ルーチンは、対称行列を使用して行列 - 行列演算を実行する。この演算は、次のように定義される。



$c := \alpha * a * b + \beta * c$

または

$c := \alpha * b * a + \beta * c$

$\alpha$  と  $\beta$  は、スカラである。

$a$  は、対称行列である。

$b$  と  $c$  は、 $m \times n$  の行列である。

## 入力パラメータ

*side* CHARACTER\*1。次に示すように、演算で対称行列  $a$  が左と右のどちらにくるかを指定する。

<i>side</i> の値	実行する演算
L または l	$c := \alpha * a * b + \beta * c$
R または r	$c := \alpha * b * a + \beta * c$

*uplo* CHARACTER\*1。次に示すように、対称行列  $a$  の上三角部分と下三角部分のどちらが参照されるかを指定する。

<i>uplo</i> の値	参照される配列 $a$ の部分
U または u	対称行列の上三角部分だけが参照される。
L または l	対称行列の下三角部分だけが参照される。

*m* INTEGER。行列  $c$  の行数を指定する。 $m$  の値はゼロ以上でなければならない。

*n* INTEGER。行列  $c$  の列数を指定する。 $n$  の値はゼロ以上でなければならない。

*alpha* REAL (ssymm の場合 )  
 DOUBLE PRECISION (dsymm の場合 )  
 COMPLEX (csymm の場合 )  
 DOUBLE COMPLEX (zsymm の場合 )  
 スカラ  $\alpha$  を指定する。

*a* REAL (ssymm の場合 )  
 DOUBLE PRECISION (dsymm の場合 )  
 COMPLEX (csymm の場合 )  
 DOUBLE COMPLEX (zsymm の場合 )

配列、次元は  $(lda, ka)$ 。  $ka$  は  $side = 'L'$  または  $'l'$  の場合は  $m$  に、そうでない場合は  $n$  になる。  $side = 'L'$  または  $'l'$  と指定した場合は、このルーチンに入る前に、次のように、配列  $a$  の  $m \times m$  の部分に対称行列を格納しなければならない。すなわち、 $uplo = 'U'$  または  $'u'$  の場合は、配列  $a$  の先頭の  $m \times m$  の上三角部分に対称行列の上三角部分を格納しなければならない。また  $a$  の厳密な下三角部分は参照されない。また、 $uplo = 'L'$  または  $'l'$  の場合は、配列  $a$  の先頭の  $m \times m$  の下三角部分に対称行列の下三角部分を格納しなければならない。また  $a$  の厳密な上三角部分は参照されない。

$side = 'R'$  または  $'r'$  と指定した場合は、このルーチンに入る前に、次のように、配列  $a$  の  $n \times n$  の部分に対称行列を格納しなければならない。すなわち、 $uplo = 'U'$  または  $'u'$  の場合は、配列  $a$  の先頭の  $n \times n$  の上三角部分に対称行列の上三角部分を格納しなければならない。また  $a$  の厳密な下三角部分は参照されない。また、 $uplo = 'L'$  または  $'l'$  の場合は、配列  $a$  の先頭の  $n \times n$  の下三角部分に対称行列の下三角部分を格納しなければならない。また  $a$  の厳密な上三角部分は参照されない。

**lda** INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 $a$  の第 1 次元を指定する。  $side = 'L'$  または  $'l'$  の場合、 $lda$  は  $\max(1, m)$  以上でなければならない。そうでない場合、 $lda$  は  $\max(1, n)$  以上でなければならない。

**b** REAL (ssymm の場合)  
DOUBLE PRECISION (dsymm の場合)  
COMPLEX (csymm の場合)  
DOUBLE COMPLEX (zsymm の場合)

配列、次元は  $(ldb, n)$ 。このルーチンに入る前に、配列  $b$  の先頭の  $m \times n$  の部分に行列  $b$  を格納しなければならない。

**ldb** INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 $b$  の第 1 次元を指定する。 $ldb$  の値は、 $\max(1, m)$  以上でなければならない。

**beta** REAL (ssymm の場合)  
DOUBLE PRECISION (dsymm の場合)  
COMPLEX (csymm の場合)  
DOUBLE COMPLEX (zsymm の場合)

スカラー  $beta$  を指定する。 $beta$  にゼロを設定した場合には、 $c$  を設定する必要はない。

**c** REAL (ssymm の場合)  
DOUBLE PRECISION (dsymm の場合)  
COMPLEX (csymm の場合)  
DOUBLE COMPLEX (zsymm の場合)

配列、次元は  $(ldc, n)$ 。このルーチンに入る前に、配列  $c$  の先頭の  $m \times n$  の部分に行列  $c$  を格納しなければならない。ただし、 $\beta$  がゼロの場合は、 $c$  を設定する必要はない。

$ldc$  INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 $c$  の第 1 次元を指定する。 $ldc$  の値は、 $\max(1, m)$  以上でなければならない。

### 出力パラメータ

$c$  更新された  $m \times n$  の行列によって上書きされる。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[Fortran-95 インターフェイス規則](#)を参照のこと。

ルーチン `symm` のインターフェイスの詳細を以下に示す。

$a$  サイズ  $(k, k)$  の行列  $A$  を格納する。ここで  $side = 'L'$  の場合は  $k = m$  に、それ以外の場合は  $k = n$  になる。

$b$  サイズ  $(m, n)$  の行列  $B$  を格納する。

$c$  サイズ  $(m, n)$  の行列  $C$  を格納する。

$side$  'L' または 'R' でなければならない。デフォルト値は 'L' である。

$uplo$  'U' または 'L' でなければならない。デフォルト値は 'U' である。

$\alpha$  デフォルト値は '1' である。

$\beta$  デフォルト値は '1' である。

## ?syrk

対称行列の階数  $n$  の更新を実行する。

### 構文

#### Fortran 77:

```
CALL sSYRK( UPLO, TRANS, N, K, ALPHA, A, LDA, BETA, C, LDC )
CALL dSYRK( UPLO, TRANS, N, K, ALPHA, A, LDA, BETA, C, LDC )
```

```
CALL cSYRK( UPLO, TRANS, N, K, ALPHA, A, LDA, BETA, C, LDC )
```

```
CALL zSYRK( UPLO, TRANS, N, K, ALPHA, A, LDA, BETA, C, LDC )
```

### Fortran 95:

```
call syrk(a, c [,uplo] [,trans] [,alpha] [,beta])
```

### 説明

?syrk ルーチンは、対称行列を使用して行列 - 行列演算を実行する。この演算は、次のように定義される。

$$c := \alpha * a * a' + \beta * c$$

または

$$c := \alpha * a' * a + \beta * c$$

$\alpha$  と  $\beta$  は、スカラーである。

$c$  は、 $n \times n$  の対称行列である。

$a$  は、1 番目の定義では  $n \times k$  の行列、2 番目の定義では  $k \times n$  の行列である。

### 入力パラメータ

**uplo** CHARACTER\*1。次に示すように、配列  $c$  の上三角部分と下三角部分のどちらが参照されるかを指定する。

uplo の値	参照される行列 $c$ の部分
U または u	$c$ の上三角部分だけが参照される。
L または l	$c$ の下三角部分だけが参照される。

**trans** CHARACTER\*1。次に示すように、実行する演算を指定する。

trans の値	実行する演算
N または n	$c := \alpha * a * a' + \beta * c$
T または t	$c := \alpha * a' * a + \beta * c$
C または c	$c := \alpha * a' * a + \beta * c$

**n** INTEGER。行列  $c$  の次数を指定する。 $n$  の値はゼロ以上でなければならない。

<i>k</i>	INTEGER。 <i>trans</i> = 'N' または 'n' と指定した場合は、 <i>k</i> には行列 <i>a</i> の列数を指定する。また、 <i>trans</i> = 'T' または 't'、あるいは 'C' または 'c' と指定した場合は、 <i>k</i> には行列 <i>a</i> の行数を指定する。 <i>k</i> の値はゼロ以上でなければならない。
<i>alpha</i>	REAL (ssyrk の場合) DOUBLE PRECISION (dsyrk の場合) COMPLEX (csyrk の場合) DOUBLE COMPLEX (zsyrk の場合)  スカラ <i>alpha</i> を指定する。
<i>a</i>	REAL (ssyrk の場合) DOUBLE PRECISION (dsyrk の場合) COMPLEX (csyrk の場合) DOUBLE COMPLEX (zsyrk の場合)  配列、次元は ( <i>lda</i> , <i>ka</i> )。 <i>ka</i> は <i>TRANS</i> = 'N' または 'n' の場合は <i>k</i> に、そうでない場合は <i>n</i> になる。 <i>trans</i> = 'N' または 'n' と指定した場合は、このルーチンに入る前に、配列 <i>a</i> の先頭の $n \times k$ の部分に行列 <i>a</i> を格納しなければならない。そうでない場合は、配列 <i>a</i> の先頭の $k \times n$ の部分に行列 <i>a</i> を格納しなければならない。
<i>lda</i>	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 <i>a</i> の第 1 次元を指定する。 <i>trans</i> = 'N' または 'n' の場合、 <i>lda</i> は $\max(1, n)$ 以上でなければならない。そうでない場合、 <i>lda</i> は $\max(1, k)$ 以上でなければならない。
<i>beta</i>	REAL (ssyrk の場合) DOUBLE PRECISION (dsyrk の場合) COMPLEX (csyrk の場合) DOUBLE COMPLEX (zsyrk の場合)  スカラ <i>beta</i> を指定する。
<i>c</i>	REAL (ssyrk の場合) DOUBLE PRECISION (dsyrk の場合) COMPLEX (csyrk の場合) DOUBLE COMPLEX (zsyrk の場合)  配列、次元は ( <i>ldc</i> , <i>n</i> )。 <i>uplo</i> = 'U' または 'u' と指定した場合は、このルーチンに入る前に、配列 <i>c</i> の先頭の $n \times n$ の上三角部分に対称行列の上三角部分を格納しなければならない。また <i>c</i> の厳密な下三角部分は参照されない。

`uplo = 'L'` または `'l'` と指定した場合は、このルーチンに入る前に、配列  $c$  の先頭の  $n \times n$  の下三角部分に対称行列の下三角部分を格納しなければならない。また  $c$  の厳密な上三角部分は参照されない。

`ldc`        `INTEGER`。呼び出し元の (サブ) プログラムで宣言されているとおりに、 $c$  の第 1 次元を指定する。`ldc` の値は、 $\max(1, n)$  以上でなければならない。

### 出力パラメータ

$c$         `uplo = 'U'` または `'u'` と指定した場合は、配列  $c$  の上三角部分が、更新された行列の上三角部分によって上書きされる。

`uplo = 'L'` または `'l'` と指定した場合は、配列  $c$  の下三角部分が、更新された行列の下三角部分によって上書きされる。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[Fortran-95 インターフェイス規則](#)を参照のこと。

ルーチン `syrk` のインターフェイスの詳細を以下に示す。

$a$         サイズ  $(ma, ka)$  の行列  $A$  を格納する。ここで、  
          `transa = 'N'` の場合は  $ka = k$  に、それ以外の場合は  $ka = n$  になる。  
          `transa = 'N'` の場合は  $ma = n$  に、それ以外の場合は  $ma = k$  になる。

$c$         サイズ  $(n, n)$  の行列  $C$  を格納する。

`uplo`        `'U'` または `'L'` でなければならない。デフォルト値は `'U'` である。

`trans`        `'N'`、`'C'`、または `'T'` でなければならない。デフォルト値は `'N'` である。

`alpha`        デフォルト値は `'1'` である。

`beta`        デフォルト値は `'1'` である。

## ?syr2k

対称行列の階数  $2k$  の更新を実行する。

### 構文

#### Fortran 77:

```
CALL sSYR2K( UPLO, TRANS, N, K, ALPHA, A, LDA, b, ldb, BETA, C, LDC )
CALL dSYR2K( UPLO, TRANS, N, K, ALPHA, A, LDA, b, ldb, BETA, C, LDC )
CALL cSYR2K( UPLO, TRANS, N, K, ALPHA, A, LDA, b, ldb, BETA, C, LDC )
CALL zSYR2K( UPLO, TRANS, N, K, ALPHA, A, LDA, b, ldb, BETA, C, LDC )
```

#### Fortran 95:

```
call syr2k(a, b, c [,uplo] [,trans] [,alpha] [,beta])
```

### 説明

?syr2k ルーチンは、対称行列を使用して階数  $2k$  の行列 - 行列演算を実行する。この演算は、次のように定義される。

$$c := \alpha * a * b' + \alpha * b * a' + \beta * c$$

または

$$c := \alpha * a' * b + \alpha * b' * a + \beta * c$$

$\alpha$  と  $\beta$  は、スカラーである。

$c$  は、 $n \times n$  の対称行列である。

$a$  と  $b$  は、1 番目の定義では  $n \times k$  の行列、2 番目の定義では  $k \times n$  の行列である。

### 入力パラメータ

**uplo** CHARACTER\*1。次に示すように、配列  $c$  の上三角部分と下三角部分のどちらが参照されるかを指定する。

uplo の値	参照される行列 $c$ の部分
U または u	$c$ の上三角部分だけが参照される。
L または l	$c$ の下三角部分だけが参照される。

*trans* CHARACTER\*1。次に示すように、実行する演算を指定する。

trans の値	実行する演算
N または n	$c := \alpha * a * b' + \alpha * b * a' + \beta * c$
T または t	$c := \alpha * a' * b + \alpha * b' * a + \beta * c$
C または c	$c := \alpha * a' * b + \alpha * b' * a + \beta * c$

*n* INTEGER。行列 *c* の次数を指定する。*n* の値はゼロ以上でなければならない。

*k* INTEGER。*trans* = 'N' または 'n' と指定した場合は、*k* には行列 *a* と *b* の列数を指定する。また、*trans* = 'T' または 't'、あるいは 'C' または 'c' と指定した場合は、*k* には行列 *a* と *b* の行数を指定する。*k* の値はゼロ以上でなければならない。

*alpha* REAL (ssyr2k の場合)  
DOUBLE PRECISION (dsyr2k の場合)  
COMPLEX (csyr2k の場合)  
DOUBLE COMPLEX (zsyr2k の場合)  
スカラ *alpha* を指定する。

*a* REAL (ssyr2k の場合)  
DOUBLE PRECISION (dsyr2k の場合)  
COMPLEX (csyr2k の場合)  
DOUBLE COMPLEX (zsyr2k の場合)

配列、次元は (*lda*, *ka*)。 *ka* は *TRANS* = 'N' または 'n' の場合は *k* に、そうでない場合は *n* になる。*trans* = 'N' または 'n' と指定した場合は、このルーチンに入る前に、配列 *a* の先頭の *n* × *k* の部分に行列 *a* を格納しなければならない。そうでない場合は、配列 *a* の先頭の *k* × *n* の部分に行列 *a* を格納しなければならない。

*lda* INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、*a* の第 1 次元を指定する。*trans* = 'N' または 'n' の場合、*lda* は max(1, *n*) 以上でなければならない。そうでない場合、*lda* は max(1, *k*) 以上でなければならない。



- b* REAL (ssyr2k の場合)  
DOUBLE PRECISION (dsyr2k の場合)  
COMPLEX (csyr2k の場合)  
DOUBLE COMPLEX (zsyr2k の場合)
- 配列、次元は (*ldb*, *kb*)。 *kb* は *trans* = 'N' または 'n' の場合は *k* に、そうでない場合は *n* になる。 *trans* = 'N' または 'n' と指定した場合は、このルーチンに入る前に、配列 *b* の先頭の  $n \times k$  の部分に行列 *b* を格納しなければならない。そうでない場合は、配列 *b* の先頭の  $k \times n$  の部分に行列 *b* を格納しなければならない。
- ldb* INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、*a* の第 1 次元を指定する。 *trans* = 'N' または 'n' の場合、*ldb* は  $\max(1, n)$  以上でなければならない。そうでない場合、*ldb* は  $\max(1, k)$  以上でなければならない。
- beta* REAL (ssyr2k の場合)  
DOUBLE PRECISION (dsyr2k の場合)  
COMPLEX (csyr2k の場合)  
DOUBLE COMPLEX (zsyr2k の場合)
- スカラ *beta* を指定する。
- c* REAL (ssyr2k の場合)  
DOUBLE PRECISION (dsyr2k の場合)  
COMPLEX (csyr2k の場合)  
DOUBLE COMPLEX (zsyr2k の場合)
- 配列、次元は (*ldc*, *n*)。 *uplo* = 'U' または 'u' と指定した場合は、このルーチンに入る前に、配列 *c* の先頭の  $n \times n$  の上三角部分に対称行列の上三角部分を格納しなければならない、また *c* の厳密な下三角部分は参照されない。
- uplo* = 'L' または 'l' と指定した場合は、このルーチンに入る前に、配列 *c* の先頭の  $n \times n$  の下三角部分に対称行列の下三角部分を格納しなければならない、また *c* の厳密な上三角部分は参照されない。
- ldc* INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、*c* の第 1 次元を指定する。 *ldc* の値は、 $\max(1, n)$  以上でなければならない。

### 出力パラメータ

- c*            *uplo* = 'U' または 'u' と指定した場合は、配列 *c* の上三角部分が、更新された行列の上三角部分によって上書きされる。
- uplo* = 'L' または 'l' と指定した場合は、配列 *c* の下三角部分が、更新された行列の下三角部分によって上書きされる。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[Fortran-95 インターフェイス規則](#)を参照のこと。

ルーチン *syr2k* のインターフェイスの詳細を以下に示す。

- a*            サイズ (*ma*, *ka*) の行列 *A* を格納する。ここで、*trans* = 'N' の場合は *ka* = *k* に、それ以外の場合は *ka* = *n* になる。  
              *trans* = 'N' の場合は *ma* = *n* に、それ以外の場合は *ma* = *k* になる。
- b*            サイズ (*mb*, *kb*) の行列 *B* を格納する。ここで、  
              *trans* = 'N' の場合 *kb* = *k* に、それ以外の場合は *kb* = *n* になる。  
              *trans* = 'N' の場合 *mb* = *n* に、それ以外の場合は *mb* = *k* になる。
- c*            サイズ (*n*, *n*) の行列 *C* を格納する。
- uplo*        'U' または 'L' でなければならない。デフォルト値は 'U' である。
- trans*        'N'、'C'、または 'T' でなければならない。デフォルト値は 'N' である。
- alpha*       デフォルト値は '1' である。
- beta*        デフォルト値は '1' である。

---

## ?trmm

スカラ - 行列 - 行列の積 ( 行列オペランドのいずれかは三角行列 ) を計算する。

---

### 構文

#### Fortran 77:

```
CALL sTRMM( SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A, LDA, B, LDB )
CALL dTRMM( SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A, LDA, B, LDB )
CALL cTRMM( SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A, LDA, B, LDB )
CALL zTRMM( SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A, LDA, B, LDB )
```

#### Fortran 95:

```
call trmm(a, b [,side] [,uplo] [,transa] [,diag] [,alpha])
```

### 説明

?trmm ルーチンは、三角行列を使用して行列 - 行列演算を実行する。この演算は、次のように定義される。

$b := \alpha * \text{op}(a) * b$

または

$b := \alpha * b * \text{op}(a)$

$\alpha$  は、スカラである。

$b$  は、 $m \times n$  の行列である。

$a$  は、単位または非単位の、上三角行列または下三角行列である。

$\text{op}(a)$  は、 $\text{op}(a) = a$ 、 $\text{op}(a) = a'$ 、または  $\text{op}(a) = \text{conjg}(a')$  のいずれかである。

### 入力パラメータ

*side* CHARACTER\*1。次に示すように、演算で  $\text{op}(a)$  が  $b$  を左側と右側のどちらから掛け合わせるかを指定する。

<i>side</i> の値	実行する演算
L または l	$b := \alpha * \text{op}(a) * b$
R または r	$b := \alpha * b * \text{op}(a)$

*uplo* CHARACTER\*1。次に示すように、行列  $a$  が上三角行列と下三角行列のどちらであるかを指定する。

<i>uplo</i> の値	行列 $a$
U または u	行列 $a$ は、上三角行列。
L または l	行列 $a$ は、下三角行列。

*transa* CHARACTER\*1。次に示すように、行列の乗算で使用する  $\text{op}(a)$  の形式を指定する。

<i>transa</i> の値	$\text{op}(a)$ の形式
N または n	$\text{op}(a) = a$
T または t	$\text{op}(a) = a'$
C または c	$\text{op}(a) = \text{conjg}(a')$

*diag* CHARACTER\*1。次に示すように、 $a$  が単位三角行列であるかどうかを指定する。

<i>diag</i> の値	行列 $a$
U または u	行列 $a$ は、単位三角行列とみなされる。
N または n	行列 $a$ は、単位三角行列とはみなされない。

*m* INTEGER。 $b$  の行数を指定する。 $m$  の値は、ゼロ以上でなければならない。

*n* INTEGER。 $b$  の列数を指定する。 $n$  の値は、ゼロ以上でなければならない。

*alpha* REAL (*strmm* の場合)  
 DOUBLE PRECISION (*dtrmm* の場合)  
 COMPLEX (*ctrmm* の場合)  
 DOUBLE COMPLEX (*ztrmm* の場合)

スカラー  $\alpha$  を指定する。 $\alpha$  がゼロの場合は  $a$  は参照されず、またこのルーチンに入る前に  $b$  を設定する必要はない。

- $a$       REAL (strmm の場合)  
           DOUBLE PRECISION (dtrmm の場合)  
           COMPLEX (ctrmm の場合)  
           DOUBLE COMPLEX (ztrmm の場合)
- 配列、次元は  $(lda, k)$ 。 $k$  は  $side = 'L'$  または  $'l'$  の場合は  $m$  に、 $side = 'R'$  または  $'r'$  の場合は  $n$  になる。 $uplo = 'U'$  または  $'u'$  と指定した場合は、このルーチンに入る前に、配列  $a$  の先頭の  $k \times k$  の上三角部分に上三角行列を格納しなければならない、また  $a$  の厳密な下三角部分は参照されない。
- $uplo = 'L'$  または  $'l'$  と指定した場合は、このルーチンに入る前に、配列  $a$  の先頭の  $k \times k$  の下三角部分に下三角行列を格納しなければならない、また  $a$  の厳密な上三角部分は参照されない。 $diag = 'U'$  または  $'u'$  と指定した場合は、 $a$  の対角成分も参照されず、1 とみなされる。
- $lda$       INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 $a$  の第 1 次元を指定する。 $side = 'L'$  または  $'l'$  の場合、 $lda$  は  $\max(1, m)$  以上でなければならない。 $side = 'R'$  または  $'r'$  の場合、 $lda$  は  $\max(1, n)$  以上でなければならない。
- $b$       real (strmm の場合)  
           DOUBLE PRECISION (dtrmm の場合)  
           COMPLEX (ctrmm の場合)  
           DOUBLE COMPLEX (ztrmm の場合)
- 配列、次元は  $(ldb, n)$ 。このルーチンに入る前に、配列  $b$  の先頭の  $m \times n$  の部分に行列  $b$  を格納しなければならない。
- $ldb$       INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 $b$  の第 1 次元を指定する。 $ldb$  の値は、 $\max(1, m)$  以上でなければならない。

## 出力パラメータ

- $b$       変換された行列によって上書きされる。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[Fortran-95 インターフェイス規則](#)を参照のこと。

ルーチン `trmm` のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ $(k,k)$ の行列 <i>A</i> を格納する。ここで <i>side</i> = 'L' の場合 $k=m$ に、それ以外の場合は $k=n$ になる。
<i>b</i>	サイズ $(m,n)$ の行列 <i>B</i> を格納する。
<i>side</i>	'L' または 'R' でなければならない。デフォルト値は 'L' である。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U' である。
<i>transa</i>	'N'、'C'、または 'T' でなければならない。デフォルト値は 'N' である。
<i>diag</i>	'N' または 'U' でなければならない。デフォルト値は 'N' である。
<i>alpha</i>	デフォルト値は '1' である。

---

### ?trsm

行列式( 行列オペランドのいずれかは三角行列)  
を解く。

---

#### 構文

##### Fortran 77:

```
CALL sTRSM( SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A, LDA, B, LDB )
CALL dTRSM( SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A, LDA, B, LDB )
CALL cTRSM( SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A, LDA, B, LDB )
CALL zTRSM( SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A, LDA, B, LDB )
```

##### Fortran 95:

```
call trsm(a, b [,side] [,uplo] [,transa] [,diag] [,alpha])
```

#### 説明

?trsm ルーチンは、次の行列式のいずれかを解く。

$$\text{op}(a) * x = \text{alpha} * b$$

または

$$x * \text{op}(a) = \text{alpha} * b$$

*alpha* は、スカラである。

$x$  と  $b$  は、 $m \times n$  の行列である。

$a$  は、単位または非単位の、上三角行列または下三角行列である。

$op(a)$  は、 $op(a) = a$ 、 $op(a) = a'$ 、または  $op(a) = conjg(a')$  のいずれかである。

行列  $x$  は、 $b$  に上書きされる。

## 入力パラメータ

*side* CHARACTER\*1。次に示すように、実行する演算で  $op(a)$  が  $x$  の左と右のどちらにくるかを指定する。

<i>side</i> の値	実行する演算
L または l	$op(a) * x = alpha * b$
R または r	$x * op(a) = alpha * b$

*uplo* CHARACTER\*1。次に示すように、行列  $a$  が上三角行列と下三角行列のどちらであるかを指定する。

<i>uplo</i> の値	行列 $a$
U または u	行列 $a$ は、上三角行列。
L または l	行列 $a$ は、下三角行列。

*transa* CHARACTER\*1。次に示すように、行列の乗算で使用する  $op(a)$  の形式を指定する。

<i>transa</i> の値	$op(a)$ の形式
N または n	$op(a) = a$
T または t	$op(a) = a'$
C または c	$op(a) = conjg(a')$

*diag* CHARACTER\*1。次に示すように、 $a$  が単位三角行列であるかどうかを指定する。

<i>diag</i> の値	行列 $a$
U または u	行列 $a$ は、単位三角行列とみなされる。
N または n	行列 $a$ は、単位三角行列とはみなされない。

*m* INTEGER。  $b$  の行数を指定する。  $m$  の値は、ゼロ以上でなければならない。

- n* INTEGER。 *b* の列数を指定する。 *n* の値はゼロ以上でなければならない。
- alpha* REAL (strsm の場合)  
DOUBLE PRECISION (dtrsm の場合)  
COMPLEX (ctrsm の場合)  
DOUBLE COMPLEX (ztrsm の場合)
- スカラー *alpha* を指定する。 *alpha* がゼロの場合は *a* は参照されず、またこのルーチンに入る前に *b* を設定する必要はない。
- a* REAL (strsm の場合)  
DOUBLE PRECISION (dtrsm の場合)  
COMPLEX (ctrsm の場合)  
DOUBLE COMPLEX (ztrsm の場合)
- 配列、次元は (*lda*, *k*)。 *k* は *side* = 'L' または 'l' の場合は *m* に、*side* = 'R' または 'r' の場合は *n* になる。 *uplo* = 'U' または 'u' と指定した場合は、このルーチンに入る前に、配列 *a* の先頭の *k* × *k* の上三角部分に上三角行列を格納しなければならない。また *a* の厳密な下三角部分は参照されない。
- uplo* = 'L' または 'l' と指定した場合は、このルーチンに入る前に、配列 *a* の先頭の *k* × *k* の下三角部分に下三角行列を格納しなければならない。また *a* の厳密な上三角部分は参照されない。 *diag* = 'U' または 'u' と指定した場合は、*a* の対角成分も参照されず、1 とみなされる。
- lda* INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、*a* の第 1 次元を指定する。 *side* = 'L' または 'l' の場合、*lda* は max(1, *m*) 以上でなければならない。 *side* = 'R' または 'r' の場合、*lda* は max(1, *n*) 以上でなければならない。
- b* REAL (strsm の場合)  
DOUBLE PRECISION (dtrsm の場合)  
COMPLEX (ctrsm の場合)  
DOUBLE COMPLEX (ztrsm の場合)
- 配列、次元は (*ldb*, *n*)。このルーチンに入る前に、配列 *b* の先頭の *m* × *n* の部分に右辺の行列 *b* を格納しなければならない。
- ldb* INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、*b* の第 1 次元を指定する。 *ldb* の値は、max(1, *m*) 以上でなければならない。

### 出力パラメータ

- b* 解の行列 *x* によって上書きされる。



### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[Fortran-95 インターフェイス規則](#)を参照のこと。

ルーチン `trsm` のインターフェイスの詳細を以下に示す。

- |               |   |
|---------------|---|
| <i>a</i>      | サイズ $(k, k)$ の行列 <i>A</i> を格納する。ここで <i>side</i> = 'L' の場合 $k = m$ に、それ以外の場合は $k = n$ になる。 |
| <i>b</i>      | サイズ $(m, n)$ の行列 <i>B</i> を格納する。  |
| <i>side</i>   | 'L' または 'R' でなければならない。デフォルト値は 'L' である。  |
| <i>uplo</i>   | 'U' または 'L' でなければならない。デフォルト値は 'U' である。  |
| <i>transa</i> | 'N'、'C'、または 'T' でなければならない。デフォルト値は 'N' である。  |
| <i>diag</i>   | 'N' または 'U' でなければならない。デフォルト値は 'N' である。  |
| <i>alpha</i>  | デフォルト値は '1' である。  |

### スパース BLAS レベル 1 のルーチンと関数

このセクションでは、BLAS レベル 1 の拡張であるスパース BLAS について説明する。スパース BLAS は、インテル MKL のリリース 2.1 からライブラリに組み込まれたもので、圧縮形式で格納されたスパースベクトルに対して多数の一般的なベクトル演算を実行するためのルーチンと関数のグループで構成されている。

スパースベクトルは、その成分の大半がゼロのベクトルである。スパース BLAS のルーチンと関数は、ベクトルが粗 (スパース) であるのを利用するために特に導入された。この特徴を利用すると、計算時間とメモリを大幅に節約することが可能である。ベクトルの非ゼロの成分の数が  $nz$  である場合、スパース BLAS の演算に必要な計算時間は  $O(nz)$  になる。

#### ベクトルの引数

**圧縮形式のスパースベクトル:**  $a$  を配列に格納されているベクトルとし、また  $a$  の非ゼロの成分を次のように仮定する。

$$a(k_1), a(k_2), a(k_3) \dots a(k_{nz})$$

$nz$  は、 $a$  内の非ゼロの成分の合計数である。

スパース BLAS では、このベクトルは 2 つの FORTRAN 配列、 $x$  (値) と  $indx$  (インデックス) を使って圧縮形式で表すことができる。それぞれの配列は、 $nz$  個の成分を持つ。

$$x(1)=a(k_1), x(2)=a(k_2), \dots x(nz)=a(k_{nz}),$$

$$indx(1)=k_1, indx(2)=k_2, \dots indx(nz)=k_{nz}.$$

したがって、スパースベクトルは 3 つの組 ( $nz, x, indx$ ) で完全に表せる。 $nz$  の値として負の数やゼロをスパース BLAS ルーチンに渡した場合は、サブルーチンはいずれの配列や変数も変更しない。

**フル格納形式のベクトル:** スパース BLAS のルーチンでは、1 つの FORTRAN 配列に完全に格納されているベクトル引数 (フル格納形式のベクトル) も使用できる。 $y$  がフル格納形式のベクトルとすると、この成分は連続して格納しなければならない。すなわち、最初の成分を  $y(1)$  に、2 番目の成分を  $y(2)$  に格納する。これは、BLAS レベル 1 の増分  $incy=1$  に対応する。フル格納形式のベクトルの増分値がスパース BLAS のルーチンや関数に引数で渡されることはない。

## 命名規則

BLAS ルーチンの場合と同じように、スパース BLAS のサブプログラムの名前の先頭には、関連するデータ型を決定する文字 ( 単精度と倍精度の実数に対しては `s` と `d`、単精度と倍精度の複素数に対しては `c` と `z`) がそれぞれに付く。

スパース BLAS のルーチンが「密」なルーチンの拡張である場合は、サブプログラムの名前は、対応する「密」なサブプログラムの名前の末尾に `i` (*indexed* を表す) が付く。例えば、スパース BLAS のルーチン `saxpyi` は、BLAS のルーチン `saxpy` に、またスパース BLAS の関数 `cdotci` は BLAS の関数 `cdotc` に対応する。

## ルーチンとデータ型

表 2-4 に、インテル MKL に導入されているスパース BLAS のルーチンとデータ型を示す。

表 2-4 スパース BLAS のルーチンおよびそのデータ型

ルーチン / 関数	データ型	説明
<a href="#">?axpyi</a>	s, d, c, z	スカラ - ベクトルの積にベクトルを加算 ( ルーチン )
<a href="#">?doti</a>	s, d	内積 ( 関数 )
<a href="#">?dotci</a>	c, z	複素共役の内積 ( 関数 )
<a href="#">?dotui</a>	c, z	複素非共役の内積 ( 関数 )
<a href="#">?gthr</a>	s, d, c, z	フル格納形式のスパースベクトルを圧縮形式 <code>nz</code> 、 <code>x</code> 、 <code>indx</code> に集積する
<a href="#">?gthrz</a>	s, d, c, z	フル格納形式のスパースベクトルを圧縮形式に集積し、フル格納形式のベクトルに集積した成分にゼロを割り当てる ( ルーチン )
<a href="#">?roti</a>	s, d	Givens 回転 ( ルーチン )
<a href="#">?sctr</a>	s, d, c, z	ベクトルを圧縮形式からフル格納形式に分散させる ( ルーチン )

## スパースベクトルで利用できる BLAS レベル 1 のルーチン

以下に示す BLAS レベル 1 のルーチンでは、圧縮形式の配列 `x` を渡した場合にも正しい結果が得られる ( 増分は、`incx = 1`)。

- [?asum](#)    ベクトル成分の絶対値の合計。
- [?copy](#)    ベクトルのコピー。
- [?nrm2](#)    ベクトルのユークリッド・ノルム。
- [?scal](#)    ベクトルのスケーリング。
- [i?amax](#)    最大絶対値を持つ成分のインデックス。あるいは、複素数の場合は、最大の合計  $|\text{Re}x(i)| + |\text{Im}x(i)|$  を持つ成分のインデックス。
- [i?amin](#)    最小絶対値を持つ成分のインデックス。あるいは、複素数の場合は、最小の合計  $|\text{Re}x(i)| + |\text{Im}x(i)|$  を持つ成分のインデックス。

`i?amax` と `i?amin` によって返される結果 `i` は、圧縮形式の配列のインデックスとして解釈しなければならない。つまり、最大(最小)値が `x(i)` で、これに対応するフル格納形式の配列のインデックスが `indx(i)` になる。

また、[?rotg](#) を呼び出して Givens 回転パラメータを計算し、その後でこれらのパラメータをスパース BLAS のルーチン [?roti](#) に渡すのも可能である。

---

### ?axpyi

圧縮形式のスパースベクトルのスカラ倍を、フル格納形式のベクトルに加える。

---

#### 構文

##### Fortran 77:

```
call sAxpypi( Nz, a, X, INdX, y )
call dAxpypi( Nz, a, X, INdX, y )
call cAxpypi( Nz, a, X, INdX, y )
call zAxpypi( Nz, a, X, INdX, y )
```

##### Fortran 95:

```
call axpyi(x, indx, y [,a])
```

#### 説明

?axpyi ルーチンは、次のように定義されるベクトル-ベクトル演算を実行する。

$$y := a * x + y$$

`a` はスカラである。

`(nz, x, indx)` は、圧縮形式で格納されたスパースベクトルである。

`y` は、フル格納形式のベクトルである。

?axpyi ルーチンは、インデックスが配列 `indx` に格納されている `y` の成分に対してのみ、参照あるいは変更を行う。`indx` の値は、明確に指定する。

#### 入力パラメータ

`nz`            INTEGER。 `x` と `indx` の成分の数。

$a$	<p>REAL (saxpyi の場合 )  DOUBLE PRECISION (daxpyi の場合 )  COMPLEX (caxpyi の場合 )  DOUBLE COMPLEX (zaxpyi の場合 )</p> <p>スカラ <math>a</math> を指定する。</p>
$x$	<p>REAL (saxpyi の場合 )  DOUBLE PRECISION (daxpyi の場合 )  COMPLEX (caxpyi の場合 )  DOUBLE COMPLEX (zaxpyi の場合 )</p> <p>配列、次元は <math>nz</math> 以上。</p>
$indx$	<p>INTEGER。 <math>x</math> の成分に対するインデックスを指定する。</p> <p>配列、次元は <math>nz</math> 以上。</p>
$y$	<p>REAL (saxpyi の場合 )  DOUBLE PRECISION (daxpyi の場合 )  COMPLEX (caxpyi の場合 )  DOUBLE COMPLEX (zaxpyi の場合 )</p> <p>配列、次元は <math>\max_i (indx(i))</math> 以上。</p>

### 出力パラメータ

$y$	更新されたベクトル $y$ が格納される。
-----	-----------------------

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[Fortran-95 インターフェイス規則](#)を参照のこと。

ルーチン `axpyi` のインターフェイスの詳細を以下に示す。

$x$	長さ ( $nz$ ) のベクトルを格納する。
$indx$	長さ ( $nz$ ) のベクトルを格納する。
$y$	長さ ( $nz$ ) のベクトルを格納する。
$a$	デフォルト値は '1' である。

### ?doti

圧縮形式の実数型スパースベクトルとフル格納形式の実ベクトルの内積を計算する。

---

#### 構文

##### Fortran 77:

```
res = sdoti( Nz, X, INdX, y )  
res = ddoti( Nz, X, INdX, y )
```

##### Fortran 95:

```
res = doti(x, indx, y)
```

#### 説明

?doti 関数は、次のように定義される  $x$  と  $y$  の内積を返す。

$$x(1)*y(indx(1)) + x(2)*y(indx(2)) + \dots + x(nz)*y(indx(nz))$$

3 つの組  $(nz, x, indx)$  には、圧縮形式で格納された実数型スパースベクトルを、 $y$  にはフル格納形式の実ベクトルを定義する。関数は、インデックスが配列  $indx$  に格納されている  $y$  の成分だけを参照する。 $indx$  の値は、明確に指定する。

#### 入力パラメータ

$nz$	INTEGER。 $x$ と $indx$ の成分の数。
$x$	REAL (sdoti の場合 ) DOUBLE PRECISION (ddoti の場合 ) 配列、次元は $nz$ 以上。
$indx$	INTEGER。 $x$ の成分に対するインデックスを指定する。 配列、次元は $nz$ 以上。
$y$	REAL (sdoti の場合 ) DOUBLE PRECISION (ddoti の場合 ) 配列、次元は $\max_i(indx(i))$ 以上。

## 出力パラメータ

`res`        `REAL` (`sdoti` の場合 )  
              `DOUBLE PRECISION` (`ddoti` の場合 )

`nz` が正の場合は、`x` と `y` の内積が格納される。そうでない場合には、`res` には 0 が格納される。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[Fortran-95 インターフェイス規則](#)を参照のこと。

ルーチン `doti` のインターフェイスの詳細を以下に示す。

`x`            長さ (`nz`) のベクトルを格納する。  
`indx`        長さ (`nz`) のベクトルを格納する。  
`y`            長さ (`nz`) のベクトルを格納する。

## ?dotci

圧縮形式の複素数型スパースベクトルとフル格納形式の複素ベクトルの共役内積を計算する。

### 構文

#### Fortran 77:

```
res = CDOTCi( Nz, X, INdX, Y )
res = zDOTCi( Nz, X, INdX, Y )
```

#### Fortran 95:

```
res = dotci(x, indx, y)
```

### 説明

?dotci 関数は、次のように定義される `x` と `y` の内積を返す。

$$\text{conjg}(x(1)) * y(\text{indx}(1)) + \dots + \text{conjg}(x(nz)) * y(\text{indx}(nz))$$

3つの組  $(nz, x, indx)$  には圧縮形式で格納された複素数型スパースベクトルを、 $y$  にはフル格納形式の複素ベクトルを定義する。関数は、インデックスが配列  $indx$  に格納されている  $y$  の成分だけを参照する。 $indx$  の値は、明確に指定する。

### 入力パラメータ

$nz$	INTEGER。 $x$ と $indx$ の成分の数。
$x$	COMPLEX (cdotci の場合 ) DOUBLE COMPLEX (zdotci の場合 ) 配列、次元は $nz$ 以上。
$indx$	INTEGER。 $x$ の成分に対するインデックスを指定する。 配列、次元は $nz$ 以上。
$y$	COMPLEX (cdotci の場合 ) DOUBLE COMPLEX (zdotci の場合 ) 配列、次元は $\max_i (indx(i))$ 以上。

### 出力パラメータ

$res$	COMPLEX (cdotci の場合 ) DOUBLE COMPLEX (zdotci の場合 )  $nz$ が正の場合は、 $x$ と $y$ の共役内積が格納される。そうでない場合は、 $res$ には 0 が格納される。
-------	--

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[Fortran-95 インターフェイス規則](#)を参照のこと。

ルーチン dotci のインターフェイスの詳細を以下に示す。

$x$	長さ ( $nz$ ) のベクトルを格納する。
$indx$	長さ ( $nz$ ) のベクトルを格納する。
$y$	長さ ( $nz$ ) のベクトルを格納する。



## ?dotui

圧縮形式の複素数型スパースベクトルとフル格納形式の複素ベクトルとの内積を計算する。

### 構文

#### Fortran 77:

```
res = CDOTui( Nz, X, INdX, Y )
res = ZDOTui( Nz, X, INdX, Y )
```

#### Fortran 95:

```
res = dotui(x, indx, y)
```

### 説明

?dotui 関数は、次のように定義される  $x$  と  $y$  の内積を返す。

$$x(1)*y(indx(1)) + x(2)*y(indx(2)) + \dots + x(nz)*y(indx(nz))$$

3 つの組  $(nz, x, indx)$  には圧縮形式で格納された複素数型スパースベクトルを、 $y$  にはフル格納形式の複素ベクトルを定義する。関数は、インデックスが配列  $indx$  に格納されている  $y$  の成分だけを参照する。 $indx$  の値は、明確に指定する。

### 入力パラメータ

$nz$	INTEGER。 $x$ と $indx$ の成分の数。
$x$	COMPLEX (cdotui の場合 ) DOUBLE COMPLEX (zdotui の場合 ) 配列、次元は $nz$ 以上。
$indx$	INTEGER。 $x$ の成分に対するインデックスを指定する。 配列、次元は $nz$ 以上。
$y$	COMPLEX (cdotui の場合 ) DOUBLE COMPLEX (zdotui の場合 ) 配列、次元は $\max_i(indx(i))$ 以上。

### 出力パラメータ

`res`            `COMPLEX` (`cdotui` の場合 )  
                 `DOUBLE COMPLEX` (`zdotui` の場合 )  
`nz` が正の場合は、`x` と `y` の内積が格納される。そうでない場合は、`res` には 0 が格納される。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[Fortran-95 インターフェイス規則](#)を参照のこと。

ルーチン `dotui` のインターフェイスの詳細を以下に示す。

`x`                長さ (`nz`) のベクトルを格納する。  
`indx`            長さ (`nz`) のベクトルを格納する。  
`y`                長さ (`nz`) のベクトルを格納する。

---

## ?gthr

フル格納形式のスパースベクトルの成分を圧縮形式に集積する。

---

### 構文

#### Fortran 77:

```
call sgthr( Nz, y, X, INdX )  
call dgthr( Nz, y, X, INdX )  
call cgthr( Nz, y, X, INdX )  
call zgthr( Nz, y, X, INdX )
```

#### Fortran 95:

```
res = gthr(x, indx, y)
```

## 説明

?gthr ルーチンは、フル格納形式のスパースベクトル  $y$  の指定の成分を、圧縮形式  $(nz, x, indx)$  に集積する。ルーチンは、インデックスが配列  $indx$  に格納されている  $y$  の成分だけを参照する。

$$x(i) = y(indx(i)), (i=1, 2, \dots, nz)$$

## 入力パラメータ

$nz$             INTEGER。集積する  $y$  の成分の数。

$indx$            INTEGER。集積する成分のインデックスを指定する。  
配列、次元は  $nz$  以上。

$y$               REAL (sgthr の場合)  
                DOUBLE PRECISION (dgthr の場合)  
                COMPLEX (cgthr の場合)  
                DOUBLE COMPLEX (zgthr の場合)  
配列、次元は  $\max_i (indx(i))$  以上。

## 出力パラメータ

$x$               REAL (sgthr の場合)  
                DOUBLE PRECISION (dgthr の場合)  
                COMPLEX (cgthr の場合)  
                DOUBLE COMPLEX (zgthr の場合)  
配列、次元は  $nz$  以上。  
  
圧縮形式に変換されたベクトルが格納される。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[Fortran-95 インターフェイス規則](#)を参照のこと。

ルーチン dthr のインターフェイスの詳細を以下に示す。

$x$               長さ ( $nz$ ) のベクトルを格納する。

$indx$            長さ ( $nz$ ) のベクトルを格納する。

$y$               長さ ( $nz$ ) のベクトルを格納する。

### ?gthrz

スパースベクトルの成分を圧縮形式に集積し、これらの成分をゼロで置き換える。

---

#### 構文

##### Fortran 77:

```
call sgthrz( Nz, y, X, INdX )
call dgthrz( Nz, y, X, INdX )
call cgthrz( Nz, y, X, INdX )
call zgthrz( Nz, y, X, INdX )
```

##### Fortran 95:

```
res = gthrz(x, indx, y)
```

#### 説明

?gthrz ルーチンは、配列 *indx* で指定したインデックスを持つ成分に対して、フル格納形式のベクトル *y* から圧縮形式 (*nz*, *x*, *indx*) に集積し、その集積した *y* の成分をゼロで上書きする。*y* の指定されていない成分に対しては、参照も変更も行わない ([?gthr](#) も参照)。

#### 入力パラメータ

<i>nz</i>	INTEGER。集積する <i>y</i> の成分の数。
<i>indx</i>	INTEGER。集積する成分のインデックスを指定する。配列、次元は <i>nz</i> 以上。
<i>y</i>	REAL (sgthrz の場合 ) DOUBLE PRECISION (dgthrz の場合 ) COMPLEX (cgthrz の場合 ) DOUBLE COMPLEX (zgthrz の場合 ) 配列、次元は $\max_i(\text{indx}(i))$ 以上。

### 出力パラメータ

$x$         REAL (sgthrz の場合)  
             DOUBLE PRECISION (dgthrz の場合)  
             COMPLEX (cgthrz の場合)  
             DOUBLE COMPLEX (zgthrz の場合)  
             配列、次元は  $nz$  以上。  
             圧縮形式に変換されたベクトルが格納される。

$y$         更新されたベクトル  $y$

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[Fortran-95 インターフェイス規則](#)を参照のこと。

ルーチン dthrz のインターフェイスの詳細を以下に示す。

$x$         長さ ( $nz$ ) のベクトルを格納する。

$indx$     長さ ( $nz$ ) のベクトルを格納する。

$y$         長さ ( $nz$ ) のベクトルを格納する。

## ?roti

一方が圧縮形式であるスパースベクトルに  
*Givens* 回転を適用する。

### 構文

#### Fortran 77:

```
CALL SROTi( Nz, X, INdX, Y, C, S )
CALL dROTi( Nz, X, INdX, Y, C, S )
```

#### Fortran 95:

```
call roti(x, indx, y [,c] [,s])
```

### 説明

?roti ルーチンは、2つの実数ベクトル、 $x$  (圧縮形式  $nz, x, indx$ ) と  $y$  (フル格納形式) の成分に Givens 回転を適用する。

$$\begin{aligned}x(i) &= c*x(i) + s*y(indx(i)) \\ y(indx(i)) &= c*y(indx(i)) - s*x(i)\end{aligned}$$

ルーチンは、インデックスが配列  $indx$  に格納されている  $y$  の成分だけを参照する。 $indx$  の値は、明確に指定する。

### 入力パラメータ

$nz$	INTEGER。 $x$ と $indx$ の成分の数。
$x$	REAL (sroti の場合) DOUBLE PRECISION (droti の場合) 配列、次元は $nz$ 以上。
$indx$	INTEGER。 $x$ の成分に対するインデックスを指定する。 配列、次元は $nz$ 以上。
$y$	REAL (sroti の場合) DOUBLE PRECISION (droti の場合) 配列、次元は $\max_i(indx(i))$ 以上。
$c$	スカラ。REAL (sroti の場合) DOUBLE PRECISION (droti の場合)
$s$	スカラ。REAL (sroti の場合) DOUBLE PRECISION (droti の場合)

### 出力パラメータ

$x$  および  $y$  更新された配列。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[Fortran-95 インターフェイス規則](#)を参照のこと。

ルーチン `roti` のインターフェイスの詳細を以下に示す。

$x$	長さ ( $nz$ ) のベクトルを格納する。
$indx$	長さ ( $nz$ ) のベクトルを格納する。

*y*           長さ (*nz*) のベクトルを格納する。  
*c*           デフォルト値は '1' である。  
*s*           デフォルト値は '1' である。

---

## ?sctr

圧縮形式のスパースベクトルをフル格納形式に変換する。

---

### 構文

#### Fortran 77:

```
call ssctr( Nz, X, INdX, y )  
call dsctr( Nz, X, INdX, y )  
call csctr( Nz, X, INdX, y )  
call zsctr( Nz, X, INdX, y )
```

#### Fortran 95:

```
call sctr(x, indx, y)
```

### 説明

?sctr ルーチンは、圧縮形式のスパースベクトル (*nz*, *x*, *indx*) の成分を、フル格納形式のベクトル *y* に分散する。 ルーチンは、インデックスが配列 *indx* に格納されている *y* の成分だけを変更する。

$y(indx(i)) = x(i), (i=1, 2, \dots, nz)$

### 入力パラメータ

*nz*           INTEGER。分散する *x* の成分の数。  
*indx*         INTEGER。分散する成分のインデックスを指定する。  
配列、次元は *nz* 以上。

$x$             REAL (ssctr の場合)  
              DOUBLE PRECISION (dsctr の場合)  
              COMPLEX (csctr の場合)  
              DOUBLE COMPLEX (zsctr の場合)  
              配列、次元は  $nz$  以上。  
              フル格納形式に変換するベクトルを格納する。

### 出力パラメータ

$y$             REAL (ssctr の場合)  
              DOUBLE PRECISION (dsctr の場合)  
              COMPLEX (csctr の場合)  
              DOUBLE COMPLEX (zsctr の場合)  
              配列、次元は  $\max_i(\text{indx}(i))$  以上。  
              更新された成分を持つベクトル  $y$  が格納される。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[Fortran-95 インターフェイス規則](#)を参照のこと。

ルーチン sctr のインターフェイスの詳細を以下に示す。

$x$             長さ ( $nz$ ) のベクトルを格納する。  
 $indx$         長さ ( $nz$ ) のベクトルを格納する。  
 $y$             長さ ( $nz$ ) のベクトルを格納する。



## スパース BLAS レベル 2 およびレベル 3

このセクションでは、インテル MKL に含まれるスパース BLAS レベル 2 およびレベル 3 について説明する。スパース BLAS レベル 2 は、スパース行列と密ベクトルに対して演算を実行するためのルーチンと関数のグループで構成されている。スパース BLAS レベル 3 は、スパース行列と密行列に対して演算を実行するためのルーチンと関数のグループで構成されている。

スパース行列は、その成分の大半がゼロの行列である。インテル MKL スパース BLAS のルーチンと関数は、行列が粗（スパース）であるのを利用するために特に導入された。この特徴を利用すると、計算時間とメモリを大幅に節約が可能である。スパース BLAS ルーチンは、本マニュアル第 8 章の「[リバース・コミュニケーション・インターフェイス \(RCI ISS\) に基づく反復法スパースソルバ](#)」の基礎といえる。

### スパース BLAS レベル 2 およびレベル 3 における命名規則

それぞれのスパース BLAS ルーチンは、プリフィックスが `mk1_` である 6 文字または 8 文字の基本名を持つ。標準インターフェイスを備えたルーチンの基本名は 6 文字で、簡易インターフェイスを備えたルーチンの基本名はテンプレートに従って 8 文字である。

```
mk1_<character code> <data> <operation>( )
```

```
mk1_<character code> <data> <mtype> <operation>( )
```

<character code> は、データ型を示すキャラクタ・コードである。

s 実数、単精度	c 複素数、単精度
d 実数、倍精度	z 複素数、倍精度



**注：**現在のバージョンのインテル MKL スパース BLAS では、実数データは倍精度のみサポートしている。

<data> フィールドにはスパース行列のデータ構造を指定する（「[スパース行列のデータ構造](#)」のセクションを参照）。

coo	座標形式
csr	行圧縮形式とその変形版
csc	列圧縮形式とその変形版
dia	対角形式
sky	輪郭格納形式

<operation> フィールドには演算のタイプを指定する。

mv 行列 - ベクトルの積 (レベル 2)  
mm 行列 - 行列の積 (レベル 3)  
sm 単一の三角法の解 (レベル 2)  
sm 複数の右辺を持つ三角法の解 (レベル 3)

任意のフィールド `<mtype>` には行列のタイプを指定する。このフィールドは簡易インターフェイス備えたルーチンで使用される。

ge 一般行列のスパース表現  
sy 対称行列の上または下三角部分のスパース表現  
tr 三角行列のスパース表現

### スパース行列のデータ構造

現在のバージョンのインテル MKL スパース BLAS レベル 2 およびレベル 3 では、次のスパース行列のデータ構造がサポートされる [Duff86]。

- 行圧縮形式 (CSR) とその変形版
- 列圧縮形式 (CSC)
- 座標形式
- 対角形式
- 輪郭格納形式

行列の格納形式の詳細は、付録 A の [スパース BLAS レベル 2-3 のスパース格納形式](#) を参照。

### ルーチンおよびサポートされる演算

このセクションでは、主要な 2 つのルーチンのタイプとサポートされる演算について説明する。ここでは次の表記を使用する。

$A$  - スパース行列  
 $B$  と  $C$  - 密行列  
 $D$  - スケーリング用の対角行列  
 $x$  と  $y$  - 密ベクトル  
 $\alpha$  と  $\beta$  - スカラ  
 $\text{op}(A)$  は次のいずれか。  
 $\text{op}(A) = A$   
 $\text{op}(A) = A' - A$  の転置  
 $\text{op}(A) = \text{conj}(A') - A$  の共役転置

[表 2-9](#) にすべてのルーチンの全リストを示す。

### 標準インターフェイスを備えたルーチン

インテル MKL スパース BLAS ルーチンは次の演算をサポートする。

#### レベル 2

- スパース行列 - 密ベクトルの積の計算  

$$y := \alpha * \text{op}(A) * x + \beta * y$$
- 単一の三角法の解の算出  

$$y := \alpha * \text{inv}(\text{op}(A)) * x$$

#### レベル 3

- スパース行列 - 密行列の積の計算  

$$C := \alpha * \text{op}(A) * B + \beta * C$$
- 複数の右辺を持つスパース三角法の解の算出  

$$C := \alpha * \text{inv}(\text{op}(A)) * B$$

これらのルーチンは、NIST スパース BLAS ライブラリで使用されるインターフェイスとは異なる、自然なインターフェイスを備えている [\[Rem05\]](#)。これらの違いの詳細は「[インターフェイス考慮事項](#)」を参照のこと。

### 簡易インターフェイスを備えたルーチン

一部のソフトウェア・パッケージとライブラリ（インテル MKL で使用している [PARDISO パッケージ](#)、[Sparskit 2](#) [\[Saad94\]](#)、[Compaq Extended Math Library \(CXML\)](#) [\[CXML01\]](#)）では、異なる（以前の）CSR 形式を使用しており、簡易インターフェイスを備えたレベル 2 の演算のみサポートしている。インテル MKL は、同様の簡易インターフェイスを備えたレベル 2 ルーチン群を提供している。それぞれのルーチンは固定タイプの行列に対して演算を行う。以下の演算をサポートしている。

$y := \text{op}(A) * x$  （一般および対称行列）  
 $y := \text{inv}(\text{op}(A)) * x$  （三角行列）

行列のタイプは、ルーチン名の `<mtype>` フィールドで指定される（「[スパース BLAS レベル 2 およびレベル 3 における命名規則](#)」のセクションを参照）。

これらのルーチンに対するインターフェイスの詳細は、「[インターフェイス考慮事項](#)」のセクションを参照のこと。

これらのルーチンは、次の3つのスパースデータ格納形式でのみ演算を行う。

PARDISO と CXML で受け入れられる CSR 形式  
 CXML で受け入れられる DIA 形式  
 COO 形式

上記のグループに属するルーチンは、特定の内部データ構造で動作する、同じ計算カーネルルーチンを使用する。

### インターフェイス考慮事項

#### インテル MKL インターフェイスと NIST インターフェイスの相違点

インテル MKL スパース BLAS レベル3ルーチンは、次のインターフェイスを備えている。

`mkl_xyyymm(transa, m, n, k, alpha, matdescra, arg(A), b, ldb, beta, c, ldc)` (行列 - 行列の積)

`mkl_xyyysm(transa, m, n, alpha, matdescra, arg(A), b, ldb, c, ldc)` (複数の右辺を持つ三角ソルバ)

これに相当する NIST スパース BLAS (NSB) ライブラリ・ルーチンは、次のインターフェイスを備えている。

`xyyyymm(transa, m, n, k, alpha, descra, arg(A), b, ldb, beta, c, ldc, work, lwork)` (行列 - 行列の積)

`xyyyysm(transa, m, n, unitd, dv, alpha, descra, arg(A), b, ldb, beta, c, ldc, work, lwork)` (複数の右辺を持つ三角ソルバ)

いくつかの類似した引数は両方のライブラリで使用される。引数 `transa` は行列を使用した演算方法を指定する。NSB ライブラリではやや異なる ([表 2-5](#) を参照)。引数 `m` と引数 `k` はそれぞれ、行列 `A` の行数と列数を示し、`n` は行列 `C` の列数を示す。引数 `alpha` と引数 `beta` はそれぞれ、スカラー `alpha` とスカラー `beta` を示す (`beta` はインテル MKL の三角ソルバで使用されていない)。引数 `b` と引数 `c` はそれぞれ、第1次元が `ldb` と `ldc` の矩形配列である。記号 `arg(A)` は、`A` のスパース表現を表す引数リストを指定する。

**表 2-5** パラメータ `transa`

	MKL インターフェイス	NSB インターフェイス	演算
データ型	CHARACTER*1	INTEGER	
値	N または n	0	$\text{op}(A) = A$
	T または t	1	$\text{op}(A) = A'$
	C または c	2	$\text{op}(A) = A'$

引数 *matdescra* は行列 *A* の関連する特性を示す。これは、NSB ライブラリの *descra* 引数に相当する (詳細は表 2-6 を参照)。

表 2-6 パラメータ *matdescra* (*descra*) に指定可能な値

	MKL インターフェイス	NSB インターフェイス	行列の性質
データ型	CHARACTER	INTEGER	
1 番目の成分	<i>matdescra</i> (1)	<i>descra</i> (1)	行列の構成
値	G	0	一般
	S	1	対称 ( $A=A'$ )
	H	2	エルミート ( $A=\text{conjg}(A')$ )
	T	3	三角
	A	4	非対称 ( $A=-A'$ )
	D	5	対角
2 番目の成分	<i>matdescra</i> (2)	<i>descra</i> (2)	上 / 下三角インジケータ
値	L	1	下三角
	U	2	上三角
3 番目の成分	<i>matdescra</i> (3)	<i>descra</i> (3)	主対角のタイプ
値	N	0	非単位
	U	1	単位

インテル MKL ルーチンでは、*matdescra* はいくつかの特徴を持っている。

行列 - 行列と行列 - ベクトルの演算を実行するルーチンでは次の点に注意する。

一般行列 (*matdescra*(1)='G') の場合、*matdescra*(2) と *matdescra*(3) の値は無視される。

非対称 - 対称行列 (*matdescra*(1)='A') の場合、*matdescra*(3) の値は無視される。

対角行列 (*matdescra*(1)='D') の場合、*matdescra*(2) の値は無視される。

*matdescra*(1) が 'G' または 'T' に設定されておらず、*matdescra*(2) と *matdescra*(3) が定義されていない場合、次のデフォルト値が割り当てられる。  
*matdescra*(2)='L' および *matdescra*(3)='N'

輪郭格納形式を使用するルーチンでは、*matdescra*(1)='G' はサポートされない。

三角ソルバの場合、*matdescra*(1)='D' ならば *matdescra*(2) は無視される。

三角ソルバに対してインテル MKL は *matdescra*(1)=T,D のみサポートする。

乗算ルーチンおよび三角ソルバでは、`matdescra(3)='U'` かつスパース行列が輪郭形式でない場合、非ゼロの対角成分を非単位であってもスパース表現で格納することができる。スパース行列が輪郭形式の場合、対角成分はゼロであってもスパース表現で格納しなければならない。

現バージョンの NSB ライブラリは、行列 - 行列の乗算では `descra(1)` のみサポートする。`descra(2)`、`descra(3)` は `descra(1)=3` の場合のみ三角ソルバでサポートされる。

引数 `work` は `work` 配列であり、`lwork` はその次元である。これらの引数は、インテル MKL では使用されない。

引数 `unitd` と引数 `dv` は、NSB 三角ソルバでのみ使用される。1 つ目は対角行列  $D$  がユニタリであるかどうかを指定する。`unitd=1` の場合、 $D$  は単位行列である。`unitd=2` ( $A$  の行がスケールされている) または `unitd=3` ( $A$  の列がスケールされている) の場合、線形配列 `dv` はスケール用の対角行列  $D$  を含む。

### 簡易インターフェイス

簡易インターフェイスを持つインテル MKL スパース BLAS レベル 2 ルーチンは、次のインターフェイスを備えている。

`mk1_xyyygemv(transa, m, arg(A), x, y)` (一般スパース行列に対する行列 - ベクトルの積)

`mk1_xyyysymv(uplo, transa, m, arg(A), x, y)` (対称スパース行列に対する行列 - ベクトルの積)

`mk1_xyyytrsv(uplo, transa, diag, m, arg(A), x, y)` (スパース三角行列を使用した連立方程式の解)

引数 `transa` は行列を使用した演算方法を指定する ([表 2-5](#) を参照)。引数 `uplo` はスパース行列の上三角と下三角のどちらを考慮するかを指定する。引数 `diag` は、 $A$  が単位三角行列であるかどうかを指定する。引数 `m` は、行列  $A$  内の行数であり、`arg(A)` は  $A$  のスパース表現を表す引数リストを指定する。配列 `x` は、入力ベクトルを格納し、配列 `y` は演算の実行結果を格納する。

行列 - ベクトルの乗算を行うすべてのルーチンは、行列  $A$  のスパース表現から三角や主対角を抽出することができる。

### 部分行列の演算

インテル MKL スパース BLAS ルーチンの特徴の 1 つは、パラメータ `matdescra` を指定する入力スパース行列の特定の部分 (三角や主対角) に対してのみ演算を実行できることである。スパース行列  $A$  は次のように分解されると仮定する。

$$A = L + D + U$$

$L$  は  $A$  の厳密な下三角、 $U$  は  $A$  の厳密な上三角、 $D$  は主対角である。

表 2-7 は行列 - 行列の乗算ルーチンの出力行列とスパース実行列  $A$  の *matdescra* の値の一致を示す。類似した一致は行列 - ベクトルの乗算ルーチンに対して存在する。

表 2-7 出力行列と *matdescra* の値の一致 (行列 - 行列の乗算ルーチン)

matdescra(1)	matdescra(2)	matdescra(3)	出力行列
G	無視される	無視される	$\alpha * \text{op}(A) * B + \beta * C$
S または H	L	N	$\alpha * \text{op}(L + D + L') * B + \beta * C$
S または H	L	U	$\alpha * \text{op}(L + I + L') * B + \beta * C$
S または H	U	N	$\alpha * \text{op}(U + D + U) * B + \beta * C$
S または H	U	U	$\alpha * \text{op}(U + I + U) * B + \beta * C$
T	L	U	$\alpha * \text{op}(L + I) * B + \beta * C$
T	L	N	$\alpha * \text{op}(L + D) * B + \beta * C$
T	U	U	$\alpha * \text{op}(U + I) * B + \beta * C$
T	U	N	$\alpha * \text{op}(U + D) * B + \beta * C$
A	L	無視される	$\alpha * \text{op}(L - L') * B + \beta * C$
A	U	無視される	$\alpha * \text{op}(U - U') * B + \beta * C$
D	無視される	N	$\alpha * D * B + \beta * C$
D	無視される	U	$\alpha * B + \beta * C$

表 2-8 は三角ソルバの出力行列とスパース実行列  $A$  の *matdescra* の値の一致を示す。

表 2-8 出力行列と *matdescra* の値の一致 (三角ソルバ)

matdescra(1)	matdescra(2)	matdescra(3)	出力行列
T	L	N	$\alpha * \text{inv}(\text{op}(L + D)) * B$
T	L	U	$\alpha * \text{inv}(\text{op}(L + I)) * B$
T	U	N	$\alpha * \text{inv}(\text{op}(U + D)) * B$
T	U	U	$\alpha * \text{inv}(\text{op}(U + I)) * B$
D	無視される	N	$\alpha * \text{inv}(D) * B$
D	無視される	U	$\alpha * B$

## 三角ソルバルーチンの制限

すべてのインテル MKL 三角ソルバには、次のような重要な制限がある。

行圧縮形式の列インデックスは、行ごとに昇順でソートされていなければならない。

列圧縮形式の行インデックスは、列ごとに昇順でソートされていなければならない。

対角形式では、スパース行列の非ゼロの対角成分の個数を格納する配列の成分は、昇順でソートされていなければならない。

## スパース BLAS レベル 2 およびレベル 3 のルーチン

このセクションで後述する、スパース BLAS レベル 2 およびレベル 3 のルーチンの一覧を表 2-9 に示す。

**表 2-9      スパース BLAS レベル 2 およびレベル 3 のルーチン**

ルーチン / 関数	説明
<b>レベル 2</b>	
<a href="#">mkl_dcsrsv</a>	CSR 形式で格納されているスパース行列の行列 - ベクトルの積を計算する。
<a href="#">mkl_dcsrcgemv</a>	CSR 形式 (PARDISO 版) で格納されているスパース一般行列の行列 - ベクトルの積を計算する。
<a href="#">mkl_dcsrcsymv</a>	CSR 形式 (PARDISO 版) で格納されているスパース対称行列の行列 - ベクトルの積を計算する。
<a href="#">mkl_dcscmv</a>	CSC 形式で格納されているスパース行列の行列 - ベクトルの積を計算する。
<a href="#">mkl_dcoomv</a>	座標形式で格納されているスパース行列の行列 - ベクトルの積を計算する。
<a href="#">mkl_dcoogemv</a>	座標形式で格納されているスパース一般行列の行列 - ベクトルの積を計算する。
<a href="#">mkl_dcoosymv</a>	座標形式で格納されているスパース対称行列の行列 - ベクトルの積を計算する。
<a href="#">mkl_ddiamv</a>	対角形式で格納されているスパース行列の行列 - ベクトルの積を計算する。
<a href="#">mkl_ddiagemv</a>	対角形式で格納されているスパース一般行列の行列 - ベクトルの積を計算する。
<a href="#">mkl_ddiasymv</a>	対角形式で格納されているスパース対称行列の行列 - ベクトルの積を計算する。
<a href="#">mkl_dskymv</a>	輪郭格納形式のスパース行列の行列 - ベクトルの積を計算する。
<a href="#">mkl_dcsrcsv</a>	CSR 形式のスパース行列について連立 1 次方程式を解く。
<a href="#">mkl_dcsrcrsv</a>	CSR 形式 (PARDISO 版) のスパース行列に対する簡易インターフェイスを備えた三角ソルバ。
<a href="#">mkl_dcscsv</a>	列圧縮形式のスパース行列について連立 1 次方程式を解く。
<a href="#">mkl_dcoosv</a>	座標形式のスパース行列について連立 1 次方程式を解く。
<a href="#">mkl_dcootrsv</a>	座標形式のスパース行列に対する簡易インターフェイスを備えた三角ソルバ。
<a href="#">mkl_ddiasv</a>	対角形式のスパース行列について連立 1 次方程式を解く。



表 2-9 スパース BLAS レベル 2 およびレベル 3 のルーチン ( 続き )

ルーチン / 関数	説明
<a href="#">mkl_ddiattrsv</a>	対角形式のスパース行列に対する簡易インターフェイスを備えた三角ソルバ。
<a href="#">mkl_dskysv</a>	輪郭形式のスパース行列について連立 1 次方程式を解く。
レベル 3	
<a href="#">mkl_dcsrcmm</a>	行圧縮形式で格納されたスパース行列の行列 - 行列の積を計算する。
<a href="#">mkl_dcscmm</a>	列圧縮形式で格納されたスパース行列の行列 - 行列の積を計算する。
<a href="#">mkl_dcoomm</a>	座標形式で格納されているスパース行列の行列 - 行列の積を計算する。
<a href="#">mkl_ddiamm</a>	対角形式で格納されているスパース行列の行列 - 行列の積を計算する。
<a href="#">mkl_dskymm</a>	輪郭形式で格納されているスパース行列の行列 - 行列の積を計算する。
<a href="#">mkl_dcsrcsm</a>	CSR 形式のスパース行列について連立 1 次行列方程式を解く。
<a href="#">dcscsm</a>	CSC 形式のスパース行列について連立 1 次行列方程式を解く。
<a href="#">mkl_dcoosm</a>	座標形式のスパース行列について連立 1 次行列方程式を解く。
<a href="#">mkl_ddiasm</a>	対角形式のスパース行列について連立 1 次行列方程式を解く。
<a href="#">mkl_dskysm</a>	輪郭格納形式で格納されたスパース行列について連立 1 次行列方程式を解く。

## mkl\_dcsrcmv

CSR 形式で格納されているスパース行列の  
行列 - ベクトルの積を計算する。

### 構文

#### Fortran:

```
call mkl_dcsrcmv(transa, m, k, alpha, matdescra, val, indx, pntrb, pntre,
  x, beta, y)
```

#### C:

```
mkl_dcsrcmv(&transa, &m, &k, &alpha, matdescra, val, indx, pntrb, pntre,
  x, &beta, y);
```

### 説明

mkl\_dcsrcmv ルーチンは、次のように定義される行列 - ベクトル演算を実行する。

$$y := \alpha * A * x + \beta * y$$

または

$y := \alpha * A' * x + \beta * y$

$\alpha$  と  $\beta$  はスカラ、 $x$  と  $y$  はベクトル、 $A$  は CSR 形式の  $m \times k$  のスパース行列、 $A'$  は  $A$  の転置を示す。

### 入力パラメータ

パラメータの説明は、Fortran 77 の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「インターフェイス」のセクションで説明する。

<code>transa</code>	CHARACTER*1。実行する演算を指定する。  $\text{transa} = 'N'$ または $'n'$ の場合、行列 - ベクトルの積は、次のように計算される。 $y := \alpha * A * x + \beta * y$  $\text{transa} = 'T'$ 、 $'t'$ または $'C'$ 、 $'c'$ の場合、行列 - ベクトルの積は次のように計算される。 $y := \alpha * A' * x + \beta * y$
<code>m</code>	INTEGER。行列 $A$ の行数。
<code>k</code>	INTEGER。行列 $A$ の列数。
<code>alpha</code>	REAL*8。スカラ $\alpha$ を指定する。
<code>matdescra</code>	CHARACTER。配列の 6 個の成分は、演算で使用する行列のプロパティを指定する。最初の 3 個の配列成分のみ使用される。 <a href="#">表 2-6</a> にそれぞれの配列成分に対して設定可能な値を示す。
<code>val</code>	REAL*8。行列 $A$ の非ゼロの成分を格納する配列。長さは、 $\text{pntrb}(m) - \text{pntrb}(1)$ である。詳細は、 <a href="#">CSR 形式</a> の <code>values</code> 配列の説明を参照のこと。
<code>indx</code>	INTEGER。行列 $A$ の非ゼロの各成分の列インデックスを格納する配列。長さは、 <code>val</code> 配列の長さと等しい。詳細は、 <a href="#">CSR 形式</a> の <code>columns</code> 配列の説明を参照のこと。
<code>pntrb</code>	INTEGER。長さ $m$ の配列。配列 <code>val</code> と配列 <code>indx</code> の行 $i$ の最初のインデックスが $\text{pntrb}(i) - \text{pntrb}(1) + 1$ である行インデックスを格納する。詳細は、 <a href="#">CSR 形式</a> の <code>pointerB</code> 配列の説明を参照のこと。
<code>pntrc</code>	INTEGER。長さ $m$ の配列。配列 <code>val</code> と配列 <code>indx</code> の行 $i$ の最後のインデックスが $\text{pntrc}(i) - \text{pntrb}(1)$ である行インデックスを格納する。詳細は、 <a href="#">CSR 形式</a> の <code>pointerE</code> 配列の説明を参照のこと。

- x* REAL\*8。配列、次元は *k* 以上 (*transa* = 'N' または 'n' の場合) または *m* 以上 (それ以外の場合)。このルーチンに入る前に、配列 *x* にベクトル *x* を格納しなければならない。
- beta* REAL\*8。スカラ *beta* を指定する。
- y* REAL\*8。配列、次元は *m* 以上 (*transa* = 'N' または 'n' の場合) または *k* 以上 (それ以外の場合)。このルーチンに入る前に、配列 *y* にベクトル *y* を格納しなければならない。

### 出力パラメータ

- y* 更新されたベクトル *y* によって上書きされる。

### インターフェイス

#### Fortran 77:

```
SUBROUTINE mkl_dcsrmmv(transa, m, k, alpha, matdescra, val, indx, pntreb,
pntre, x, beta, y)
  CHARACTER*1    transa
  CHARACTER      matdescra(*)
  INTEGER        m, k
  INTEGER        indx(*), pntreb(m), pntre(m)
  REAL*8         alpha, beta
  REAL*8         val(*), x(*), y(*)
```

#### Fortran 95:

```
SUBROUTINE mkl_dcsrmmv(transa, m, k, alpha, matdescra, val, indx, pntreb,
pntre, x, beta, y)
  CHARACTER(LEN=1), INTENT(IN):: transa
  INTEGER, INTENT(IN) :: m, k
  CHARACTER, INTENT(IN) :: matdescra(*)
  INTEGER, INTENT(IN) :: indx(*), pntreb(*), pntre(*)
  REAL(KIND(1.0D0)), INTENT(IN) :: alpha, beta
  REAL(KIND(1.0D0)), INTENT(IN) :: val(*), x(*)
  REAL(KIND(1.0D0)), INTENT(INOUT) :: y(*)
```

#### C:

```
void mkl_dcsrmmv(char *transa, int *m, int *k, double *alpha, char
  *matdescra, double *val, int *indx, int *pntreb, int *pntre, double
  *x, double *beta, double *y);
```

### mkl\_dcsrgev

CSR 形式 (PARDISO 版) で格納されている  
スパース一般行列の行列 - ベクトルの積を  
計算する。

---

#### 構文

##### Fortran:

```
call mkl_dcsrgev(transa, m, a, ia, ja, x, y)
```

##### C:

```
mkl_dcsrgev(&transa, &m, a, ia, ja, x, y);
```

#### 説明

mkl\_dcsrgev ルーチンは、次のように定義される行列 - ベクトル演算を実行する。

$y := A * x$

または

$y := A' * x$

$x$  と  $y$  はベクトル、 $A$  は CSR 形式 (PARDISO 版) の  $m \times m$  のスパース正方行列、 $A'$  は  $A$  の転置を示す。

#### 入力パラメータ

パラメータの説明は、Fortran 77 の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「**インターフェイス**」のセクションで説明する。

**transa** CHARACTER\*1。実行する演算を指定する。

**transa** = 'N' または 'n' の場合、行列 - ベクトルの積は、次のように計算される。

$y := A * x$

**transa** = 'T'、't' または 'C'、'c' の場合、行列 - ベクトルの積は次のように計算される。

$y := A' * x$

**m** INTEGER。行列  $A$  の行数。

- a** REAL\*8。行列  $A$  の非ゼロの成分を格納する配列。長さは、配列  $A$  内の非ゼロの成分の数と等しい。詳細は、[スパース行列の格納形式](#)の `values` 配列の説明を参照のこと。
- ia** INTEGER。長さ  $m+1$  の配列。配列  $a$  の成分のインデックスを格納する。 $ia(i)$  は配列  $a$  の行  $i$  の最初の非ゼロ成分のインデックスである。最後の成分  $ia(m+1)-1$  の値は、非ゼロの数+1 に等しい。詳細は、[スパース行列の格納形式](#)の `rowIndex` 配列の説明を参照のこと。
- ja** REAL\*8。行列  $A$  の非ゼロの各成分の列インデックスを格納する配列。長さは、配列  $a$  の長さと同じ。詳細は、[スパース行列の格納形式](#)の `columns` 配列の説明を参照のこと。
- x** REAL\*8。配列、次元は  $m$ 。このルーチンに入る前に、配列  $x$  にベクトル  $x$  を格納しなければならない。

### 出力パラメータ

- y** REAL\*8。配列、次元は  $m$  以上。終了時に、配列  $y$  にベクトル  $y$  を格納しなければならない。

### インターフェイス

#### Fortran 77:

```
SUBROUTINE mkl_dcsrgemv(transa, m, a, ia, ja, x, y)
  CHARACTER*1  transa
  INTEGER      m
  INTEGER      ia(*), ja(*)
  REAL*8       a(*), x(*), y(*)
```

#### Fortran 95:

```
SUBROUTINE mkl_dcsrgemv(transa, m, a, ia, ja, x, y)
  CHARACTER(LEN=1), INTENT(IN):: transa
  INTEGER, INTENT(IN) :: m
  INTEGER, INTENT(IN) :: ia(*), ja(*)
  REAL(KIND(1.0D0)), INTENT(IN) :: a(*), x(*)
  REAL(KIND(1.0D0)), INTENT(OUT) :: y(*)
```

#### C:

```
void mkl_dcsrgemv(char *transa, int *m, double *a, int *ia, int *ja,
  double *x, double *y);
```

### mkl\_dcsrsvmv

CSR 形式 (PARDISO 版) で格納されている  
スパース対称行列の行列 - ベクトルの積を  
計算する。

---

#### 構文

##### Fortran:

```
call mkl_dcsrsvmv(uplo, m, a, ia, ja, x, y)
```

##### C:

```
mkl_dcsrsvmv(&uplo, &m, a, ia, ja, x, y);
```

#### 説明

mkl\_dcsrsvmv ルーチンは、次のように定義される行列 - ベクトル演算を実行する。

$y := A * x$

または

$y := A' * x$

$x$  と  $y$  はベクトル、 $A$  は CSR 形式 (PARDISO 版) のスパース対称行列の上または下三角、 $A'$  は  $A$  の転置を示す。

#### 入力パラメータ

パラメータの説明は、Fortran 77 の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「**インターフェイス**」のセクションで説明する。

**uplo** CHARACTER\*1。行列  $A$  の上三角と下三角のどちらを使用するかを指定する。  
uplo = 'U' または 'u' の場合、行列  $A$  の上三角が使用される。  
uplo = 'L' または 'l' の場合、行列  $A$  の下三角が使用される。  
**m** INTEGER。行列  $A$  の行数。

- a** REAL\*8。行列  $A$  の非ゼロの成分を格納する配列。長さは、配列  $A$  内の非ゼロの成分の数と等しい。詳細は、[スパース行列の格納形式](#)の `values` 配列の説明を参照のこと。
- ia** INTEGER。長さ  $m+1$  の配列。配列  $a$  の成分のインデックスを格納する。 $ia(i)$  は配列  $a$  の行  $i$  の最初の非ゼロ成分のインデックスである。最後の成分  $ia(m+1)-1$  の値は、非ゼロの数+1 に等しい。詳細は、[スパース行列の格納形式](#)の `rowIndex` 配列の説明を参照のこと。
- ja** REAL\*8。行列  $A$  の非ゼロの各成分の列インデックスを格納する配列。長さは、配列  $a$  の長さと同じ。詳細は、[スパース行列の格納形式](#)の `columns` 配列の説明を参照のこと。
- x** REAL\*8。配列、次元は  $m$ 。このルーチンに入る前に、配列  $x$  にベクトル  $x$  を格納しなければならない。

### 出力パラメータ

- y** REAL\*8。配列、次元は  $m$  以上。終了時に、配列  $y$  にベクトル  $y$  を格納しなければならない。

### インターフェイス

#### Fortran 77:

```
SUBROUTINE mkl_dcsrsvmv(uplo, m, a, ia, ja, x, y)
  CHARACTER*1 uplo
  INTEGER m
  INTEGER ia(*), ja(*)
  REAL*8 a(*), x(*), y(*)
```

#### Fortran 95:

```
SUBROUTINE mkl_dcsrsvmv(uplo, m, a, ia, ja, x, y)
  CHARACTER(LEN=1), INTENT(IN):: uplo
  INTEGER, INTENT(IN) :: m
  INTEGER, INTENT(IN) :: ia(*), ja(*)
  REAL(KIND(1.0D0)), INTENT(IN) :: a(*), x(*)
  REAL(KIND(1.0D0)), INTENT(OUT) :: y(*)
```

#### C:

```
void mkl_dcsrsvmv(char *uplo, int *m, double *a, int *ia, int *ja, double
*x, double *y);
```

### mkl\_dcscmv

列圧縮形式のスパース行列の行列 - ベクトルの積を計算する。

---

#### 構文

##### Fortran:

```
call mkl_dcscmv(transa, m, k, alpha, matdescra, val, indx, pntreb, pntre,  
               x, beta, y)
```

##### C:

```
mkl_dcscmv(&transa, &m, &k, &alpha, matdescra, val, indx, pntreb, pntre,  
          x, &beta, y);
```

#### 説明

mkl\_dcscmv ルーチンは、次のように定義される行列 - ベクトル演算を実行する。

$$y := \alpha * A * x + \beta * y$$

または

$$y := \alpha * A' * x + \beta * y$$

$\alpha$  と  $\beta$  はスカラ、 $x$  と  $y$  はベクトル、 $A$  は列圧縮形式の  $m \times k$  のスパース行列、 $A'$  は  $A$  の転置を示す。

#### 入力パラメータ

パラメータの説明は、Fortran 77 の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「**インターフェイス**」のセクションで説明する。

**transa** CHARACTER\*1。実行する演算を指定する。

$\text{transa} = \text{'N'}$  または  $\text{'n'}$  の場合、行列 - ベクトルの積は、次のように計算される。

$$y := \alpha * A * x + \beta * y$$

$\text{transa} = \text{'T'}$ 、 $\text{'t'}$  または  $\text{'C'}$ 、 $\text{'c'}$  の場合、行列 - ベクトルの積は次のように計算される。

$$y := \alpha * A' * x + \beta * y$$

**m** INTEGER。行列  $A$  の行数。



<i>k</i>	INTEGER。行列 <i>A</i> の列数。
<i>alpha</i>	REAL*8。スカラ <i>alpha</i> を指定する。
<i>matdescra</i>	CHARACTER。配列の 6 個の成分は、演算で使用する行列のプロパティを指定する。最初の 3 個の配列成分のみ使用される。 <a href="#">表 2-6</a> にそれぞれの配列成分に対して設定可能な値を示す。
<i>val</i>	REAL*8。行列 <i>A</i> の非ゼロの成分を格納する配列。長さは、 <i>pntre(k) - pntrb(1)</i> である。詳細は、 <a href="#">CSC 形式</a> の <i>values</i> 配列の説明を参照のこと。
<i>indx</i>	INTEGER。行列 <i>A</i> の非ゼロの各成分の行インデックスを格納する配列。長さは、 <i>val</i> 配列の長さと等しい。詳細は、 <a href="#">CSC 形式</a> の <i>rows</i> 配列の説明を参照のこと。
<i>pntrb</i>	INTEGER。長さ <i>k</i> の配列。配列 <i>val</i> と配列 <i>indx</i> の列 <i>i</i> の最初のインデックスが <i>pntrb(i) - pntrb(1)+1</i> である行インデックスを格納する。詳細は、 <a href="#">CSC 形式</a> の <i>pointerB</i> 配列の説明を参照のこと。
<i>pntre</i>	INTEGER。長さ <i>k</i> の配列。配列 <i>val</i> と配列 <i>indx</i> の列 <i>i</i> の最後のインデックスが <i>pntre(i) - pntrb(1)</i> である行インデックスを格納する。詳細は、 <a href="#">CSC 形式</a> の <i>pointerE</i> 配列の説明を参照のこと。
<i>x</i>	REAL*8。配列、次元は <i>k</i> 以上 ( <i>transa</i> = 'N' または 'n' の場合) または <i>m</i> 以上 (それ以外の場合)。このルーチンに入る前に、配列 <i>x</i> にベクトル <i>x</i> を格納しなければならない。
<i>beta</i>	REAL*8。スカラ <i>beta</i> を指定する。
<i>y</i>	REAL*8。配列、次元は <i>m</i> 以上 ( <i>transa</i> = 'N' または 'n' の場合) または <i>k</i> 以上 (それ以外の場合)。このルーチンに入る前に、配列 <i>y</i> にベクトル <i>y</i> を格納しなければならない。

## 出力パラメータ

*y*            更新されたベクトル *y* によって上書きされる。

## インターフェイス

### Fortran 77:

```
SUBROUTINE mkl_dcscmv(transa, m, k, alpha, matdescra, val, indx, pntrb,
pntre, x, beta, y)
  CHARACTER*1  transa
  CHARACTER    matdescra(*)
  INTEGER      m, k, ldb, ldc
```

```
INTEGER      indx(*), pntrb(m), pntre(m)
REAL*8       alpha, beta
REAL*8       val(*), x(*), y(*)
```

### Fortran 95:

```
SUBROUTINE mkl_dcscmv(transa, m, k, alpha, matdescra, val, indx, pntrb,
pntre, x, beta, y)
  CHARACTER(LEN=1), INTENT(IN):: transa
  INTEGER, INTENT(IN) :: m, k
  CHARACTER, INTENT(IN) :: matdescra(*)
  INTEGER, INTENT(IN) :: indx(*), pntrb(*), pntre(*)
  REAL(KIND(1.0D0)), INTENT(IN) :: alpha, beta
  REAL(KIND(1.0D0)), INTENT(IN) :: val(*), x(*)
  REAL(KIND(1.0D0)), INTENT(INOUT) :: y(*)
```

### C:

```
void mkl_dcscmv(char *transa, int *m, int *k, double *alpha, char
    *matdescra, double *val, int *indx, int *pntrb, int *pntre, double
    *x, double *beta, double *y);
```

---

## mkl\_dcoomv

座標形式で格納されているスパース行列の  
行列-ベクトルの積を計算する。

---

### 構文

#### Fortran:

```
call mkl_dcoomv(transa, m, k, alpha, matdescra, val, rowind, colind, nnz,
    x, beta, y)
```

#### C:

```
mkl_dcoomv(&transa, &m, &k, &alpha, matdescra, val, rowind, colind, &nnz,
    x, &beta, y);
```

### 説明

mkl\_dcoomv ルーチンは、次のように定義される行列-ベクトル演算を実行する。

$$y := \alpha * A * x + \beta * y$$

または

$y := \alpha * A' * x + \beta * y$

$\alpha$  と  $\beta$  はスカラ、 $x$  と  $y$  はベクトル、 $A$  は圧縮座標形式の  $m \times k$  のスパース行列、 $A'$  は  $A$  の転置を示す。

## 入力パラメータ

パラメータの説明は、Fortran 77 の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「**インターフェイス**」のセクションで説明する。

<i>transa</i>	CHARACTER*1。実行する演算を指定する。  $\text{transa} = 'N'$ または $'n'$ の場合、行列 - ベクトルの積は、次のように計算される。 $y := \alpha * A * x + \beta * y$  $\text{transa} = 'T', 't'$ または $'C', 'c'$ の場合、行列 - ベクトルの積は次のように計算される。 $y := \alpha * A' * x + \beta * y$
<i>m</i>	INTEGER。行列 $A$ の行数。
<i>k</i>	INTEGER。行列 $A$ の列数。
<i>alpha</i>	REAL*8。スカラ $\alpha$ を指定する。
<i>matdescra</i>	CHARACTER。配列の 6 個の成分は、演算で使用する行列のプロパティを指定する。最初の 3 個の配列成分のみ使用される。 <a href="#">表 2-6</a> にそれぞれの配列成分に対して設定可能な値を示す。
<i>val</i>	REAL*8。長さ <i>nnz</i> の配列。行列 $A$ の非ゼロ成分を任意の順番で格納する。詳細は、 <a href="#">座標形式</a> の <i>values</i> 配列の説明を参照のこと。
<i>rowind</i>	INTEGER。長さ <i>nnz</i> の配列。行列 $A$ の各非ゼロ成分の行インデックスを格納する。詳細は、 <a href="#">座標形式</a> の <i>rows</i> 配列の説明を参照のこと。
<i>colind</i>	INTEGER。長さ <i>nnz</i> の配列。行列 $A$ の各非ゼロ成分の列インデックスを格納する。詳細は、 <a href="#">座標形式</a> の <i>columns</i> 配列の説明を参照のこと。
<i>nnz</i>	INTEGER。行列 $A$ の非ゼロの成分を指定する。詳細は、 <a href="#">座標形式</a> の <i>nnz</i> 配列の説明を参照のこと。
<i>x</i>	REAL*8。配列、次元は $k$ 以上 ( $\text{transa} = 'N'$ または $'n'$ の場合) または $m$ 以上 (それ以外の場合)。このルーチンに入る前に、配列 $x$ にベクトル $x$ を格納しなければならない。

*beta*            REAL\*8。スカラ *beta* を指定する。

*y*                REAL\*8。配列、次元は *m* 以上 (*transa* = 'N' または 'n' の場合) または *k* 以上 (それ以外の場合)。このルーチンに入る前に、配列 *y* にベクトル *y* を格納しなければならない。

### 出力パラメータ

*y*                更新されたベクトル *y* によって上書きされる。

### インターフェイス

#### Fortran 77:

```
SUBROUTINE mkl_dcoomv(transa, m, k, alpha, matdescra, val, rowind,
colind, nnz, x, beta, y)
    CHARACTER*1    transa
    CHARACTER      matdescra(*)
    INTEGER         m, k, nnz
    INTEGER         rowind(*), colind(*)
    REAL*8          alpha, beta
    REAL*8          val(*), x(*), y(*)
```

#### Fortran 95:

```
SUBROUTINE mkl_dcoomv(transa, m, k, alpha, matdescra, val, rowind,
colind, nnz, x, beta, y)
    CHARACTER(LEN=1), INTENT(IN) :: transa
    INTEGER, INTENT(IN) :: m, k, nnz
    CHARACTER, INTENT(IN) :: matdescra(*)
    INTEGER, INTENT(IN) :: rowind(*), colind(*)
    REAL(KIND(1.0D0)), INTENT(IN) :: alpha, beta
    REAL(KIND(1.0D0)), INTENT(IN) :: val(*), x(*)
    REAL(KIND(1.0D0)), INTENT(INOUT) :: y(*)
```

#### C:

```
void mkl_dcoomv(char *transa, int *m, int *k, double *alpha, char
    *matdescra, double *val, int *rowind, int *colind, int *nnz, double
    *x, double *beta, double *y);
```

## mkl\_dcoogemv

座標形式で格納されているスパース一般行列の  
行列-ベクトルの積を計算する。

### 構文

#### Fortran:

```
call mkl_dcoogemv(transa, m, val, rowind, colind, nnz, x, y)
```

#### C:

```
mkl_dcoogemv(&transa, &m, val, rowind, colind, &nnz, x, y);
```

### 説明

mkl\_dcoogemv ルーチンは、次のように定義される行列-ベクトル演算を実行する。

$y := A * x$

または

$y := A' * x$

$x$  と  $y$  はベクトル、 $A$  は座標形式の  $m \times m$  のスパース正方行列、 $A'$  は  $A$  の転置を示す。

### 入力パラメータ

パラメータの説明は、Fortran 77 の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「**インターフェイス**」のセクションで説明する。

**transa** CHARACTER\*1。実行する演算を指定する。

$transa = 'N'$  または  $'n'$  の場合、行列-ベクトルの積は、次のように計算される。

$y := A * x$

$transa = 'T'$ 、 $'t'$  または  $'C'$ 、 $'c'$  の場合、行列-ベクトルの積は次のように計算される。

$y := A' * x$

**m** INTEGER。行列  $A$  の行数。

**val** REAL\*8。長さ  $nnz$  の配列。行列  $A$  の非ゼロ成分を任意の順番で格納する。詳細は、[座標形式](#)の **values** 配列の説明を参照のこと。

*rowind*      INTEGER。長さ *nnz* の配列。行列 *A* の各非ゼロ成分の行インデックスを格納する。詳細は、[座標形式](#)の *rows* 配列の説明を参照のこと。

*colind*      INTEGER。長さ *nnz* の配列。行列 *A* の各非ゼロ成分の列インデックスを格納する。詳細は、[座標形式](#)の *columns* 配列の説明を参照のこと。

*nnz*          INTEGER。行列 *A* の非ゼロの成分を指定する。詳細は、[座標形式](#)の *nnz* 配列の説明を参照のこと。

*x*            REAL\*8。配列、次元は *m*。このルーチンに入る前に、配列 *x* にベクトル *x* を格納しなければならない。

### 出力パラメータ

*y*            REAL\*8。配列、次元は *m* 以上。終了時に、配列 *y* にベクトル *y* を格納しなければならない。

### インターフェイス

#### Fortran 77:

```
SUBROUTINE mkl_dcoogemv(transa, m, val, rowind, colind, nnz, x, y)
  CHARACTER*1  transa
  INTEGER      m, nnz
  INTEGER      rowind(*), colind(*)
  REAL*8       val(*), x(*), y(*)
```

#### Fortran 95:

```
SUBROUTINE mkl_dcoogemv(transa, m, val, rowind, colind, nnz, x, y)
  CHARACTER(LEN=1), INTENT(IN) :: transa
  INTEGER, INTENT(IN) :: m, nnz
  INTEGER, INTENT(IN) :: rowind(*), colind(*)
  REAL(KIND(1.0D0)), INTENT(IN) :: val(*), x(*)
  REAL(KIND(1.0D0)), INTENT(INOUT) :: y(*)
```

#### C:

```
void mkl_dcoogemv(char *transa, int *m, double *val, int *rowind, int
  *colind, int *nnz, double *x, double *y);
```

## mkl\_dcoosymv

座標形式で格納されているスパース対称行列の  
行列-ベクトルの積を計算する。

### 構文

#### Fortran:

```
call mkl_dcoosymv(uplo, m, val, rowind, colind, nnz, x, y)
```

#### C:

```
mkl_dcoosymv(&uplo, &m, val, rowind, colind, &nnz, x, y);
```

### 説明

mkl\_dcoosymv ルーチンは、次のように定義される行列 - ベクトル演算を実行する。

$$y := A * x$$

または

$$y := A' * x$$

$x$  と  $y$  はベクトル、 $A$  は座標形式のスパース対称行列の上または下三角、 $A'$  は  $A$  の転置を示す。

### 入力パラメータ

パラメータの説明は、Fortran 77 の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「**インターフェイス**」のセクションで説明する。

**uplo** CHARACTER\*1。行列  $A$  の上三角と下三角のどちらを使用するかを指定する。

$uplo = 'U'$  または  $'u'$  の場合、行列  $A$  の上三角が使用される。

$uplo = 'L'$  または  $'l'$  の場合、行列  $A$  の下三角が使用される。

**m** INTEGER。行列  $A$  の行数。

**val** REAL\*8。長さ  $nnz$  の配列。行列  $A$  の非ゼロ成分を任意の順番で格納する。詳細は、[座標形式](#)の **values** 配列の説明を参照のこと。

**rowind** INTEGER。長さ  $nnz$  の配列。行列  $A$  の各非ゼロ成分の行インデックスを格納する。詳細は、[座標形式](#)の **rows** 配列の説明を参照のこと。

*colind*      INTEGER。長さ *nnz* の配列。行列 *A* の各非ゼロ成分の列インデックスを格納する。詳細は、[座標形式](#)の *columns* 配列の説明を参照のこと。

*nnz*          INTEGER。行列 *A* の非ゼロの成分を指定する。詳細は、[座標形式](#)の *nnz* 配列の説明を参照のこと。

*x*            REAL\*8。配列、次元は *m*。このルーチンに入る前に、配列 *x* にベクトル *x* を格納しなければならない。

### 出力パラメータ

*y*            REAL\*8。配列、次元は *m* 以上。終了時に、配列 *y* にベクトル *y* を格納しなければならない。

### インターフェイス

#### Fortran 77:

```
SUBROUTINE mkl_dcoosymv(uplo, m, val, rowind, colind, nnz, x, y)
  CHARACTER*1    uplo
  INTEGER        m, nnz
  INTEGER        rowind(*), colind(*)
  REAL*8         val(*), x(*), y(*)
```

#### Fortran 95:

```
SUBROUTINE mkl_dcoosymv(uplo, m, val, rowind, colind, nnz, x, y)
  CHARACTER(LEN=1), INTENT(IN) :: uplo
  INTEGER, INTENT(IN) :: m, nnz
  INTEGER, INTENT(IN) :: rowind(*), colind(*)
  REAL(KIND(1.0D0)), INTENT(IN) :: val(*), x(*)
  REAL(KIND(1.0D0)), INTENT(INOUT) :: y(*)
```

#### C:

```
void mkl_dcoosymv(char *uplo, int *m, double *val, int *rowind, int
  *colind, int *nnz, double *x, double *y);
```



## mkl\_ddiamv

対角形式で格納されているスパース行列の  
行列-ベクトルの積を計算する。

### 構文

#### Fortran:

```
call mkl_ddiamv(transa, m, k, alpha, matdescra, val, lval, idiag, ndiag,
               x, beta, y)
```

#### C:

```
mkl_ddiamv(&transa, &m, &k, &alpha, matdescra, val, &lval, idiag, &ndiag,
           x, &beta, y);
```

### 説明

mkl\_ddiamv ルーチンは、次のように定義される行列 - ベクトル演算を実行する。

$$y := \alpha * A * x + \beta * y$$

または

$$y := \alpha * A' * x + \beta * y$$

$\alpha$  と  $\beta$  はスカラ、 $x$  と  $y$  はベクトル、 $A$  は対角形式の  $m \times k$  のスパース行列、 $A'$  は  $A$  の転置を示す。

### 入力パラメータ

パラメータの説明は、Fortran 77 の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「**インターフェイス**」のセクションで説明する。

**transa** CHARACTER\*1。実行する演算を指定する。

$\text{transa} = \text{'N'}$  または  $\text{'n'}$  の場合、行列 - ベクトルの積は、次のように計算される。

$$y := \alpha * A * x + \beta * y$$

$\text{transa} = \text{'T'}$ 、 $\text{'t'}$  または  $\text{'C'}$ 、 $\text{'c'}$  の場合、行列 - ベクトルの積は次のように計算される。

$$y := \alpha * A' * x + \beta * y$$

**m** INTEGER。行列  $A$  の行数。

<i>k</i>	INTEGER。行列 <i>A</i> の列数。
<i>alpha</i>	REAL*8。スカラ <i>alpha</i> を指定する。
<i>matdescra</i>	CHARACTER。配列の 6 個の成分は、演算で使用する行列のプロパティを指定する。最初の 3 個の配列成分のみ使用される。 <a href="#">表 2-6</a> にそれぞれの配列成分に対して設定可能な値を示す。
<i>val</i>	REAL*8。 <i>lval</i> × <i>ndiag</i> の 2 次元配列。行列 <i>A</i> の非ゼロ対角成分を格納する。詳細は、 <a href="#">対角格納方式</a> の <i>values</i> 配列の説明を参照のこと。
<i>lval</i>	INTEGER。 <i>val</i> , <i>lval</i> ≥ min( <i>m</i> , <i>k</i> ) のリーディング・ディメンジョン。詳細は、 <a href="#">対角格納方式</a> の <i>lval</i> 配列の説明を参照のこと。
<i>idiag</i>	INTEGER。長さ <i>ndiag</i> の配列。行列 <i>A</i> の主対角と非ゼロ対角の距離を格納する。詳細は、 <a href="#">対角格納方式</a> の <i>distance</i> 配列の説明を参照のこと。
<i>ndiag</i>	INTEGER。行列 <i>A</i> の非ゼロの対角成分を指定する。
<i>x</i>	REAL*8。配列、次元は <i>k</i> 以上 ( <i>transa</i> = 'N' または 'n' の場合) または <i>m</i> 以上 (それ以外の場合)。このルーチンに入る前に、配列 <i>x</i> にベクトル <i>x</i> を格納しなければならない。
<i>beta</i>	REAL*8。スカラ <i>beta</i> を指定する。
<i>y</i>	REAL*8。配列、次元は <i>m</i> 以上 ( <i>transa</i> = 'N' または 'n' の場合) または <i>k</i> 以上 (それ以外の場合)。このルーチンに入る前に、配列 <i>y</i> にベクトル <i>y</i> を格納しなければならない。

### 出力パラメータ

*y*            更新されたベクトル *y* によって上書きされる。

### インターフェイス

#### Fortran 77:

```
SUBROUTINE mkl_ddiamv(transa, m, k, alpha, matdescra, val, lval, idiag,
ndiag, x, beta, y)
  CHARACTER*1  transa
  CHARACTER    matdescra(*)
  INTEGER      m, k, lval, ndiag
  INTEGER      idiag(*)
  REAL*8       alpha, beta
  REAL*8       val(lval,*), x(*), y(*)
```

**Fortran 95:**

```

SUBROUTINE mkl_ddiamv(transa, m, k, alpha, matdescra, val, lval, idiag,
ndiag, x, beta, y)
    CHARACTER(LEN=1), INTENT(IN) :: transa
    INTEGER, INTENT(IN) :: m, k, lval, ndiag
    CHARACTER, INTENT(IN) :: matdescra(*)
    INTEGER, INTENT(IN) :: idiag(*)
    REAL(KIND(1.0D0)), INTENT(IN) :: alpha, beta
    REAL(KIND(1.0D0)), INTENT(IN) :: val(lval,*), x(*)
    REAL(KIND(1.0D0)), INTENT(INOUT) :: y(*)

```

**C:**

```

void mkl_ddiamv(char *transa, int *m, int *k, double *alpha, char
    *matdescra, double *val, int *lval, int *idiag, int *ndiag, double
    *x, double *beta, double *y);

```

## mkl\_ddiagemv

対角形式で格納されているスパース一般行列の  
行列-ベクトルの積を計算する。

**構文****Fortran:**

```
call mkl_ddiagemv(transa, m, val, lval, idiag, ndiag, x, y)
```

**C:**

```
mkl_ddiagemv(&transa, &m, val, &lval, idiag, &ndiag, x, y);
```

**説明**

mkl\_ddiagemv ルーチンは、次のように定義される行列 - ベクトル演算を実行する。

$$y := A * x$$

または

$$y := A' * x$$

$x$  と  $y$  はベクトル、 $A$  は対角格納形式の  $m \times m$  のスパース正方行列、 $A'$  は  $A$  の転置を示す。

### 入力パラメータ

パラメータの説明は、Fortran 77 の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「**インターフェイス**」のセクションで説明する。

<i>transa</i>	CHARACTER*1。実行する演算を指定する。  transa = 'N' または 'n' の場合、行列 - ベクトルの積は、次のように計算される。 y := A*x  transa= 'T'、't' または 'C'、'c' の場合、行列 - ベクトルの積は次のように計算される。 y := A'*x
<i>m</i>	INTEGER。行列 A の行数。
<i>val</i>	REAL*8。lval × ndiag の 2 次元配列。行列 A の非ゼロ対角成分を格納する。詳細は、 <a href="#">対角格納方式</a> の values 配列の説明を参照のこと。
<i>lval</i>	INTEGER。val, lval ≥ m のリーディング・ディメンジョン。詳細は、 <a href="#">対角格納方式</a> の lval 配列の説明を参照のこと。
<i>idiag</i>	INTEGER。長さ ndiag の配列。行列 A の主対角と非ゼロ対角の距離を格納する。詳細は、 <a href="#">対角格納方式</a> の distance 配列の説明を参照のこと。
<i>ndiag</i>	INTEGER。行列 A の非ゼロの対角成分を指定する。
<i>x</i>	REAL*8。配列、次元は m。このルーチンに入る前に、配列 x にベクトル x を格納しなければならない。

### 出力パラメータ

<i>y</i>	REAL*8。配列、次元は m 以上。終了時に、配列 y にベクトル y を格納しなければならない。
----------	--

### インターフェイス

#### Fortran 77:

```
SUBROUTINE mkl_ddiagmv(transa, m, val, lval, idiag, ndiag, x, y)
  CHARACTER*1  transa
  INTEGER       m, lval, ndiag
  INTEGER       idiag(*)
  REAL*8        val(lval,*), x(*), y(*)
```

**Fortran 95:**

```
SUBROUTINE mkl_ddiagemv(transa, m, val, lval, idiag, ndiag, x, y)
  CHARACTER(LEN=1), INTENT(IN) :: transa
  INTEGER, INTENT(IN) :: m, lval, ndiag
  INTEGER, INTENT(IN) :: idiag(*)
  REAL(KIND(1.0D0)), INTENT(IN) :: val(lval,*), x(*)
  REAL(KIND(1.0D0)), INTENT(OUT) :: y(*)
```

**C:**

```
void mkl_ddiagemv(char *transa, int *m, double *val, int *lval, int
  *idiag, int *ndiag, double *x, double *y);
```

**mkl\_ddiasymv**

対角形式で格納されているスパース対称行列の  
行列-ベクトルの積を計算する。

**構文****Fortran:**

```
call mkl_ddiasymv(uplo, m, val, lval, idiag, ndiag, x, y)
```

**C:**

```
mkl_ddiasymv(&uplo, &m, val, &lval, idiag, &ndiag, x, y);
```

**説明**

mkl\_ddiasymv ルーチンは、次のように定義される行列-ベクトル演算を実行する。

$y := A * x$

または

$y := A' * x$

$x$  と  $y$  はベクトル、 $A$  は対称スパース行列の上三角または下三角、 $A'$  は  $A$  の転置を示す。

**入力パラメータ**

パラメータの説明は、Fortran 77 の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「**インターフェイス**」のセクションで説明する。

<i>uplo</i>	CHARACTER*1。行列 <i>A</i> の上三角と下三角のどちらを使用するかを指定する。 <i>uplo</i> = 'U' または 'u' の場合、行列 <i>A</i> の上三角が使用される。 <i>uplo</i> = 'L' または 'l' の場合、行列 <i>A</i> の下三角が使用される。
<i>m</i>	INTEGER。行列 <i>A</i> の行数。
<i>val</i>	REAL*8。 <i>lval</i> × <i>ndiag</i> の 2 次元配列。行列 <i>A</i> の非ゼロ対角成分を格納する。 詳細は、 <a href="#">対角格納方式</a> の <i>values</i> 配列の説明を参照のこと。
<i>lval</i>	INTEGER。 <i>val</i> , <i>lval</i> ≥ <i>m</i> のリーディング・ディメンジョン。詳細は、 <a href="#">対角格納方式</a> の <i>lval</i> 配列の説明を参照のこと。
<i>idiag</i>	INTEGER。長さ <i>ndiag</i> の配列。行列 <i>A</i> の主対角と非ゼロ対角の距離を格納する。 詳細は、 <a href="#">対角格納方式</a> の <i>distance</i> 配列の説明を参照のこと。
<i>ndiag</i>	INTEGER。行列 <i>A</i> の非ゼロの対角成分を指定する。
<i>x</i>	REAL*8。配列、次元は <i>m</i> 。このルーチンに入る前に、配列 <i>x</i> にベクトル <i>x</i> を格納しなければならない。

### 出力パラメータ

<i>y</i>	REAL*8。配列、次元は <i>m</i> 以上。終了時に、配列 <i>y</i> にベクトル <i>y</i> を格納しなければならない。
----------	---

### インターフェイス

#### Fortran 77:

```
SUBROUTINE mkl_ddiasymv(uplo, m, val, lval, idiag, ndiag, x, y)
  CHARACTER*1    uplo
  INTEGER        m, lval, ndiag
  INTEGER        idiag(*)
  REAL*8         val(lval,*), x(*), y(*)
```

#### Fortran 95:

```
SUBROUTINE mkl_ddiasymv(uplo, m, val, lval, idiag, ndiag, x, y)
  CHARACTER(LEN=1), INTENT(IN) :: uplo
  INTEGER, INTENT(IN) :: m, lval, ndiag
  INTEGER, INTENT(IN) :: idiag(*)
  REAL(KIND(1.0D0)), INTENT(IN) :: val(lval,*), x(*)
  REAL(KIND(1.0D0)), INTENT(OUT) :: y(*)
```

**C:**

```
void mkl_ddiasymv(char *uplo, int *m, double *val, int *lval, int
    *idiag, int *ndiag, double *x, double *y);
```

---

## mkl\_dskymv

輪郭格納形式のスパース行列の行列-ベクトルの積を計算する。

---

### 構文

**Fortran:**

```
call mkl_dskymv(transa, m, k, alpha, matdescra, val, pntr, x, beta, y)
```

**C:**

```
mkl_dskymv(&transa, &m, &k, &alpha, matdescra, val, pntr, x, &beta, y);
```

### 説明

mkl\_dskymv ルーチンは、次のように定義される行列-ベクトル演算を実行する。

$$y := \alpha * A * x + \beta * y$$

または

$$y := \alpha * A' * x + \beta * y$$

$\alpha$  と  $\beta$  はスカラ、 $x$  と  $y$  はベクトル、 $A$  は輪郭格納形式を使用して格納された  $m \times k$  のスパース行列、 $A'$  は  $A$  の転置を示す。

### 入力パラメータ

パラメータの説明は、Fortran 77 の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「**インターフェイス**」のセクションで説明する。

**transa** CHARACTER\*1。実行する演算を指定する。

$\text{transa} = 'N'$  または  $'n'$  の場合、行列-ベクトルの積は、次のように計算される。

$$y := \alpha * A * x + \beta * y$$

$transa = 'T'$ 、 $'t'$  または  $'C'$ 、 $'c'$  の場合、行列 - ベクトルの積は次のように計算される。

$y := \alpha * A * x + \beta * y$

$m$	INTEGER。行列 $A$ の行数。
$k$	INTEGER。行列 $A$ の列数。
$\alpha$	REAL*8。スカラー $\alpha$ を指定する。
$matdescra$	CHARACTER。配列の 6 個の成分は、演算で使用する行列のプロパティを指定する。最初の 3 個の配列成分のみ使用される。 <a href="#">表 2-6</a> にそれぞれの配列成分に対して設定可能な値を示す。
$val$	REAL*8。行列 $A$ の成分のセット が輪郭プロファイル形式で格納された配列。  $matdescrsa(2) = 'L'$ の場合、 $val$ は行列 $A$ の下三角の成分を格納する。 $matdescrsa(2) = 'U'$ の場合、 $val$ は行列 $A$ の上三角の成分を格納する。  詳細は、 <a href="#">スカイライン格納方式</a> の $values$ 配列の説明を参照のこと。
$pntr$	INTEGER。下三角の場合は長さ $(m+1)$ 、上三角の場合は長さ $(k+1)$ の配列。 $val$ で指定される、行列 $A$ の各行 ( または列 ) の最初の成分の位置のインデックスを格納する。詳細は、 <a href="#">スカイライン格納方式</a> の $pointers$ 配列の説明を参照のこと。
$x$	REAL*8。配列、次元は $k$ 以上 ( $transa = 'N'$ または $'n'$ の場合) または $m$ 以上 ( それ以外の場合 )。このルーチンに入る前に、配列 $x$ にベクトル $x$ を格納しなければならない。
$\beta$	REAL*8。スカラー $\beta$ を指定する。
$y$	REAL*8。配列、次元は $m$ 以上 ( $transa = 'N'$ または $'n'$ の場合) または $k$ 以上 ( それ以外の場合 )。このルーチンに入る前に、配列 $y$ にベクトル $y$ を格納しなければならない。

### 出力パラメータ

$y$             更新されたベクトル  $y$  によって上書きされる。

### インターフェイス

#### Fortran 77:

```
SUBROUTINE mkl_dskymv(transa, m, k, alpha, matdescra, val, pntr, x, beta, y)
  CHARACTER*1  transa
  CHARACTER    matdescra(*)
  INTEGER      m, k
```



```

INTEGER      pntr(*)
REAL*8       alpha, beta
REAL*8       val(*), x(*), y(*)

```

#### Fortran 95:

```

SUBROUTINE mkl_dskymv(transa, m, k, alpha, matdescra, val, pntr, x, beta, y)
  CHARACTER(LEN=1), INTENT(IN) :: transa
  INTEGER, INTENT(IN) :: m, k
  CHARACTER, INTENT(IN) :: matdescra(*)
  INTEGER, INTENT(IN) :: pntr(*)
  REAL(KIND(1.0D0)), INTENT(IN) :: alpha, beta
  REAL(KIND(1.0D0)), INTENT(IN) :: val(*), x(*)
  REAL(KIND(1.0D0)), INTENT(INOUT) :: y(*)

```

#### C:

```

void mkl_dskymv (char *transa, int *m, int *k, double *alpha, char *matdescra,
  double *val, int *pntr, double *x, double *beta, double *y);

```

---

## mkl\_dcsrsv

*CSR 形式のスパース行列の連立1 次方程式を解く。*

---

### 構文

#### Fortran:

```

call mkl_dcsrsv(transa, m, alpha, matdescra, val, indx, pntrb, pntre, x, y)

```

#### C:

```

mkl_dcsrsv(&transa, &m, &alpha, matdescra, val, indx, pntrb, pntre, x, y);

```

### 説明

mkl\_dcsrsv ルーチンは、CSR 形式のスパース行列に対する行列 - ベクトル演算による連立 1 次方程式の解を算出する。

$y := \alpha * \text{inv}(A) * x$

または

$y := \alpha * \text{inv}(A') * x$

$\alpha$  はスカラ、 $x$  と  $y$  はベクトル、 $A$  は単位または非単位の主対角であるスパース上三角行列または下三角行列、 $A'$  は  $A$  の転置を示す。

### 入力パラメータ

パラメータの説明は、Fortran 77 の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「**インターフェイス**」のセクションで説明する。

<code>transa</code>	CHARACTER*1。実行する演算を指定する。  <code>transa = 'N'</code> または <code>'n'</code> の場合、 $y := \alpha * \text{inv}(A) * x$ <code>transa = 'T'</code> 、 <code>'t'</code> または <code>'C'</code> 、 <code>'c'</code> の場合、 $y := \alpha * \text{inv}(A') * x$
<code>m</code>	INTEGER。行列 $A$ の列数。
<code>alpha</code>	REAL*8。スカラ $\alpha$ を指定する。
<code>matdescra</code>	CHARACTER。配列の 6 個の成分は、演算で使用する行列のプロパティを指定する。最初の 3 個の配列成分のみ使用される。 <a href="#">表 2-6</a> にそれぞれの配列成分に対して設定可能な値を示す。
<code>val</code>	REAL*8。行列 $A$ の非ゼロの成分を格納する配列。長さは、 $\text{pntrb}(m) - \text{pntrb}(1)$ である。詳細は、 <a href="#">CSR 形式</a> の <code>values</code> 配列の説明を参照のこと。
<code>indx</code>	INTEGER。行列 $A$ の非ゼロの各成分の列インデックスを格納する配列。長さは、 <code>val</code> 配列の長さと等しい。詳細は、 <a href="#">CSR 形式</a> の <code>columns</code> 配列の説明を参照のこと。
<code>pntrb</code>	INTEGER。長さ $m$ の配列。配列 <code>val</code> と配列 <code>indx</code> の行 $i$ の最初のインデックスが $\text{pntrb}(i) - \text{pntrb}(1) + 1$ である行インデックスを格納する。詳細は、 <a href="#">CSR 形式</a> の <code>pointerB</code> 配列の説明を参照のこと。
<code>pntrb</code>	INTEGER。長さ $m$ の配列。配列 <code>val</code> と配列 <code>indx</code> の行 $i$ の最後のインデックスが $\text{pntrb}(i) - \text{pntrb}(1)$ である行インデックスを格納する。詳細は、 <a href="#">CSR 形式</a> の <code>pointerE</code> 配列の説明を参照のこと。
<code>x</code>	REAL*8。配列、次元は $m$ 以上。このルーチンに入る前に、配列 <code>x</code> にベクトル $x$ を格納しなければならない。成分には単位増分を使用してアクセスする。
<code>y</code>	REAL*8。配列、次元は $m$ 以上。このルーチンに入る前に、配列 <code>y</code> にベクトル $y$ を格納しなければならない。成分には単位増分を使用してアクセスする。

### 出力パラメータ

<code>y</code>	解として得られたベクトル $x$ を格納する。
----------------	-------------------------

## インターフェイス

### Fortran 77:

```
SUBROUTINE mkl_dcsrsv(transa, m, alpha, matdescra, val, indx, pntrb,
pntre, x, y)
  CHARACTER*1    transa
  CHARACTER       matdescra(*)
  INTEGER         m
  INTEGER         indx(*), pntrb(m), pntre(m)
  REAL*8         alpha
  REAL*8         val(*)
  REAL*8         x(*), y(*)
```

### Fortran 95:

```
SUBROUTINE mkl_dcsrsv(transa, m, alpha, matdescra, val, indx, pntrb,
pntre, x, y)
  CHARACTER(LEN=1), INTENT(IN):: transa
  INTEGER, INTENT(IN) :: m
  CHARACTER, INTENT(IN) :: matdescra(*)
  INTEGER, INTENT(IN) :: indx(*), pntrb(*), pntre(*)
  REAL(KIND(1.0D0)), INTENT(IN) :: alpha
  REAL(KIND(1.0D0)), INTENT(IN) :: val(*), x(*)
  REAL(KIND(1.0D0)), INTENT(INOUT) :: y(*)
```

### C:

```
void mkl_dcsrsv(char *transa, int *m, double *alpha, char *matdescra, double
*val, int *indx, int *pntrb, int *pntre, double *x, double *y);
```

---

## mkl\_dcsrtrsv

CSR 形式(PARDISO 版) のスパース行列に対する  
簡易インターフェイスを備えた三角ソルバ。

---

## 構文

### Fortran:

```
call mkl_dcsrtrsv(uplo, transa, diag, m, a, ia, ja, x, y)
```

**C:**

```
mkldcsrtrsv(&uplo, &transa, &diag, &m, a, ia, ja, x, y);
```

### 説明

mkldcsrtrsv ルーチンは、**PARDISO** で受け入れられる **CSR** 形式で格納されたスパース行列に対する行列 - ベクトル演算による連立 1 次方程式の解を算出する。

$$A * y = x$$

または

$$A' * y = x$$

$x$  と  $y$  はベクトル、 $A$  は単位または非単位の主対角であるスパース上三角行列または下三角行列、 $A'$  は  $A$  の転置を示す。

### 入力パラメータ

パラメータの説明は、**Fortran 77** の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「**インターフェイス**」のセクションで説明する。

**uplo** CHARACTER\*1。行列  $A$  の上三角と下三角のどちらを使用するかを指定する。

$uplo = 'U'$  または  $'u'$  の場合、行列  $A$  の上三角が使用される。

$uplo = 'L'$  または  $'l'$  の場合、行列  $A$  の下三角が使用される。

**transa** CHARACTER\*1。実行する演算を指定する。

$transa = 'N'$  または  $'n'$  の場合、 $A * y = x$

$transa = 'T'$ 、 $'t'$  または  $'C'$ 、 $'c'$  の場合、 $A' * y = x$

**diag** CHARACTER\*1。  $A$  が単位三角行列であるかどうかを指定する。

$diag = 'U'$  または  $'u'$  の場合、 $A$  は単位三角行列とみなされる。

$diag = 'N'$  または  $'n'$  の場合、 $A$  は単位三角行列とみなされない。

**m** INTEGER。行列  $A$  の行数。

**a** REAL\*8。行列  $A$  の非ゼロの成分を格納する配列。長さは、配列  $A$  内の非ゼロの成分の数と等しい。詳細は、[スパース行列の格納形式](#)の *values* 配列の説明を参照のこと。

- ia*        INTEGER。長さ  $m+1$  の配列。配列 *a* の成分のインデックスを格納する。  
*ia*(*i*) は配列 *a* の行 *i* の最初の非ゼロ成分のインデックスである。最後の成分 *ia*( $m+1$ )-1 の値は、非ゼロの数 +1 に等しい。詳細は、[スパース行列の格納形式](#) の *rowIndex* 配列の説明を参照のこと。
- ja*        REAL\*8。行列 *A* の非ゼロの各成分の列インデックスを格納する配列。長さは、配列 *a* の長さと同じ。詳細は、[スパース行列の格納形式](#) の *columns* 配列の説明を参照のこと。
- x*        REAL\*8。配列、次元は  $m$ 。このルーチンに入る前に、配列 *x* にベクトル *x* を格納しなければならない。

### 出力パラメータ

- y*        REAL\*8。配列、次元は  $m$  以上。ベクトル *y* が格納される。

### インターフェイス

#### Fortran 77:

```
SUBROUTINE mkl_dcsrtrsv(uplo, transa, diag, m, a, ia, ja, x, y)
  CHARACTER*1  uplo, transa, diag
  INTEGER      m
  INTEGER      ia(*), ja(*)
  REAL*8       a(*), x(*), y(*)
```

#### Fortran 95:

```
SUBROUTINE mkl_dcsrtrsv(uplo, transa, diag, m, a, ia, ja, x, y)
  CHARACTER(LEN=1), INTENT(IN):: uplo, transa, diag
  INTEGER, INTENT(IN) :: m
  INTEGER, INTENT(IN) :: ia(*), ja(*)
  REAL(KIND(1.0D0)), INTENT(IN) :: a(*), x(*)
  REAL(KIND(1.0D0)), INTENT(OUT) :: y(*)
```

#### C:

```
void mkl_dcsrtrsv(char *uplo, char *transa, char *diag, int *m, double *a,
  int *ia, int *ja, double *x, double *y);
```

### **mkl\_dcscsv**

*CSC 形式のスパース行列の連立1 次方程式を解く。*

---

#### **構文**

##### **Fortran:**

```
call mkl_dcscsv(transa, m, alpha, matdescra, val, indx, pntbr, pntre, x, y)
```

##### **C:**

```
mkl_dcscsv(&transa, &m, &alpha, matdescra, val, indx, pntbr, pntre, x, y);
```

#### **説明**

mkl\_dcscsv ルーチンは、CSC 形式のスパース行列に対する行列 - ベクトル演算による連立 1 次方程式の解を算出する。

$y := \alpha * \text{inverse\_of\_}(A) * x$

または

$y := \alpha * \text{inverse\_of\_}(A') * x$

$\alpha$  はスカラ、 $x$  と  $y$  はベクトル、 $A$  は単位または非単位の主対角であるスパース上三角行列または下三角行列、 $A'$  は  $A$  の転置を示す。

#### **入力パラメータ**

パラメータの説明は、Fortran 77 の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「**インターフェイス**」のセクションで説明する。

<b>transa</b>	CHARACTER*1。実行する演算を指定する。  $\text{transa} = \text{'N'}$ または $\text{'n'}$ の場合 $y := \alpha * \text{inv}(A) * x$ $\text{transa} = \text{'T'}$ 、 $\text{'t'}$ または $\text{'C'}$ 、 $\text{'c'}$ の場合、 $y := \alpha * \text{inv}(A') * x$
<b>m</b>	INTEGER。行列 $A$ の列数。
<b>alpha</b>	REAL*8。スカラ $\alpha$ を指定する。
<b>matdescra</b>	CHARACTER。配列の 6 個の成分は、演算で使用する行列のプロパティを指定する。最初の 3 個の配列成分のみ使用される。 <a href="#">表 2-6</a> にそれぞれの配列成分に対して設定可能な値を示す。

<i>val</i>	REAL*8。行列 <i>A</i> の非ゼロの成分を格納する配列。長さは、 <i>pntrb(m) - pntrb(1)</i> である。詳細は、 <a href="#">CSC 形式</a> の <i>values</i> 配列の説明を 参照のこと。
<i>indx</i>	INTEGER。行列 <i>A</i> の非ゼロの各成分の列インデックスを格納する配列。長 さは、 <i>val</i> 配列の長さと同じ。詳細は、 <a href="#">CSC 形式</a> の <i>columns</i> 配列の説明 を参照のこと。
<i>pntrb</i>	INTEGER。長さ <i>m</i> の配列。配列 <i>val</i> と配列 <i>indx</i> の行 <i>i</i> の最初のインデッ クスが <i>pntrb(i) - pntrb(1)+1</i> である行インデックスを格納する。詳細は、 <a href="#">CSC 形式</a> の <i>pointerB</i> 配列の説明を参照のこと。
<i>pntrb</i>	INTEGER。長さ <i>m</i> の配列。配列 <i>val</i> と配列 <i>indx</i> の行 <i>i</i> の最後のインデッ クスが <i>pntrb(i) - pntrb(1)</i> である行インデックスを格納する。詳細は、 <a href="#">CSC 形式</a> の <i>pointerE</i> 配列の説明を参照のこと。
<i>x</i>	REAL*8。配列、次元は <i>m</i> 以上。このルーチンに入る前に、配列 <i>x</i> にベクトル <i>x</i> を格納しなければならない。成分には単位増分を使用してアクセスする。
<i>y</i>	REAL*8。配列、次元は <i>m</i> 以上。このルーチンに入る前に、配列 <i>y</i> にベクトル <i>y</i> を格納しなければならない。成分には単位増分を使用してアクセスする。

## 出力パラメータ

*y* 解として得られたベクトル *x* を格納する。

## インターフェイス

### Fortran 77:

```
SUBROUTINE mkl_dcscsv(transa, m, alpha, matdescra, val, indx, pntrb,
pntrb, x, y)
  CHARACTER*1  transa
  CHARACTER    matdescra(*)
  INTEGER      m
  INTEGER      indx(*), pntrb(m), pntrb(m)
  REAL*8       alpha
  REAL*8       val(*)
  REAL*8       x(*), y(*)
```

### Fortran 95:

```
SUBROUTINE mkl_dcscsv(transa, m, alpha, matdescra, val, indx, pntrb,
pntrb, x, y)
  CHARACTER(LEN=1), INTENT(IN):: transa
  INTEGER, INTENT(IN) :: m
```

```
CHARACTER, INTENT(IN) :: matdescra(*)
INTEGER, INTENT(IN) :: indx(*), pntrb(*), pntre(*)
REAL(KIND(1.0D0)), INTENT(IN) :: alpha
REAL(KIND(1.0D0)), INTENT(IN) :: val(*), x(*)
REAL(KIND(1.0D0)), INTENT(INOUT) :: y(*)
```

**C:**

```
void mkl_dcscsv(char *transa, int *m, double *alpha, char *matdescra,
               double *val, int *indx, int *pntrb, int *pntre, double *x, double
               *y);
```

---

### mkl\_dcoosv

座標形式のスパース行列について連立1次方程式を解く。

---

#### 構文

**Fortran:**

```
call mkl_dcoosv(transa, m, k, alpha, matdescra, val, rowind, colind, nnz,
               x, y)
```

**C:**

```
mkl_dcoosv(&transa, &m, &k, &alpha, matdescra, val, rowind, colind, &nnz,
           x, y);
```

#### 説明

mkl\_dcoosv ルーチンは、座標形式のスパース行列に対する行列 - ベクトル演算による連立1次方程式の解を算出する。

$y := \alpha * \text{inverse\_of\_}(A) * x$

または

$y := \alpha * \text{inverse\_of\_}(A') * x$

$\alpha$  はスカラ、 $x$  と  $y$  はベクトル、 $A$  は単位または非単位の主対角であるスパース上三角行列または下三角行列、 $A'$  は  $A$  の転置を示す。



## 入力パラメータ

パラメータの説明は、Fortran 77 の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「**インターフェイス**」のセクションで説明する。

<i>transa</i>	CHARACTER*1。実行する演算を指定する。  $\begin{aligned} \text{transa} = 'N' \text{ または } 'n' \text{ の場合} & \quad y := \alpha * \text{inv}(A) * x \\ \text{transa} = 'T', 't' \text{ または } 'C', 'c' \text{ の場合、} & \quad y := \alpha * \text{inv}(A') * x \end{aligned}$
<i>m</i>	INTEGER。行列 <i>A</i> の行数。
<i>alpha</i>	REAL*8。スカラ <i>alpha</i> を指定する。
<i>matdescra</i>	CHARACTER。配列の 6 個の成分は、演算で使用する行列のプロパティを指定する。最初の 3 個の配列成分のみ使用される。 <a href="#">表 2-6</a> にそれぞれの配列成分に対して設定可能な値を示す。
<i>val</i>	REAL*8。長さ <i>nnz</i> の配列。行列 <i>A</i> の非ゼロ成分を任意の順番で格納する。詳細は、 <a href="#">座標形式</a> の <i>values</i> 配列の説明を参照のこと。
<i>rowind</i>	INTEGER。長さ <i>nnz</i> の配列。行列 <i>A</i> の各非ゼロ成分の行インデックスを格納する。詳細は、 <a href="#">座標形式</a> の <i>rows</i> 配列の説明を参照のこと。
<i>colind</i>	INTEGER。長さ <i>nnz</i> の配列。行列 <i>A</i> の各非ゼロ成分の列インデックスを格納する。詳細は、 <a href="#">座標形式</a> の <i>columns</i> 配列の説明を参照のこと。
<i>nnz</i>	INTEGER。行列 <i>A</i> の非ゼロの成分を指定する。詳細は、 <a href="#">座標形式</a> の <i>nnz</i> 配列の説明を参照のこと。
<i>x</i>	REAL*8。配列、次元は <i>m</i> 以上。このルーチンに入る前に、配列 <i>x</i> にベクトル <i>x</i> を格納しなければならない。成分には単位増分を使用してアクセスする。
<i>y</i>	REAL*8。配列、次元は <i>m</i> 以上。このルーチンに入る前に、配列 <i>y</i> にベクトル <i>y</i> を格納しなければならない。成分には単位増分を使用してアクセスする。

## 出力パラメータ

<i>y</i>	解として得られたベクトル <i>x</i> を格納する。
----------	------------------------------

## インターフェイス

### Fortran 77:

```
SUBROUTINE mkl_dcoosv(transa, m, alpha, matdescra, val, rowind, colind,
nnz, x, y)
    CHARACTER*1    transa
```

```
CHARACTER      matdescra(*)
INTEGER        m, nnz
INTEGER        rowind(*), colind(*)
REAL*8         alpha
REAL*8         val(*)
REAL*8         x(*), y(*)
```

### Fortran 95:

```
SUBROUTINE mkl_dcoosv(transa, m, alpha, matdescra, val, rowind, colind,
nnz, x, y)
  CHARACTER(LEN=1), INTENT(IN) :: transa
  INTEGER, INTENT(IN) :: m, nnz
  CHARACTER, INTENT(IN) :: matdescra(*)
  INTEGER, INTENT(IN) :: rowind(*), colind(*)
  REAL(KIND(1.0D0)), INTENT(IN) :: alpha
  REAL(KIND(1.0D0)), INTENT(IN) :: val(*), x(*)
  REAL(KIND(1.0D0)), INTENT(INOUT) :: y(*)
```

### C:

```
void mkl_dcoosv(char *transa, int *m, double *alpha, char *matdescra,
double *val, int *rowind, int *colind, int *nnz, double *x, double *y);
```

---

## mkl\_dcootrsv

座標形式のスパース行列に対する簡易インターフェイスを備えた三角ソルバ。

---

### 構文

#### Fortran:

```
call mkl_dcootrsv(uplo, transa, diag, m, val, rowind, colind, nnz, x, y)
```

#### C:

```
mkl_dcootrsv(&uplo, &transa, &diag, &m, val, rowind, colind, &nnz, x, y);
```

### 説明

mkl\_dcootrsv ルーチンは、座標形式で格納されたスパース行列に対する行列 - ベクトル演算による連立 1 次方程式の解を算出する。

$$A*y = x$$

または

$$A'*y = x$$

$x$  と  $y$  はベクトル、 $A$  は単位または非単位の主対角であるスパース上三角行列または下三角行列、 $A'$  は  $A$  の転置を示す。

### 入力パラメータ

パラメータの説明は、Fortran 77 の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「**インターフェイス**」のセクションで説明する。

- uplo** CHARACTER\*1。行列  $A$  の上三角と下三角のどちらを使用するかを指定する。  
     **uplo** = 'U' または 'u' の場合、行列  $A$  の上三角が使用される。  
     **uplo** = 'L' または 'l' の場合、行列  $A$  の下三角が使用される。
- transa** CHARACTER\*1。実行する演算を指定する。  
     **transa** = 'N' または 'n' の場合、 $A*y = x$   
     **transa** = 'T'、't' または 'C'、'c' の場合、 $A'*y = x$
- diag** CHARACTER\*1。 $A$  が単位三角行列であるかどうかを指定する。  
     **diag** = 'U' または 'u' の場合、 $A$  は単位三角行列とみなされる。  
     **diag** = 'N' または 'n' の場合、 $A$  は単位三角行列とみなされない。
- m** INTEGER。行列  $A$  の行数。
- val** REAL\*8。長さ  $nnz$  の配列。行列  $A$  の非ゼロ成分を任意の順番で格納する。詳細は、[座標形式](#)の **values** 配列の説明を参照のこと。
- rowind** INTEGER。長さ  $nnz$  の配列。行列  $A$  の各非ゼロ成分の行インデックスを格納する。詳細は、[座標形式](#)の **rows** 配列の説明を参照のこと。
- colind** INTEGER。長さ  $nnz$  の配列。行列  $A$  の各非ゼロ成分の列インデックスを格納する。詳細は、[座標形式](#)の **columns** 配列の説明を参照のこと。
- nnz** INTEGER。行列  $A$  の非ゼロの成分を指定する。詳細は、[座標形式](#)の **nnz** 配列の説明を参照のこと。
- x** REAL\*8。配列、次元は  $m$ 。このルーチンに入る前に、配列 **x** にベクトル  $x$  を格納しなければならない。

### 出力パラメータ

$y$  REAL\*8。配列、次元は  $m$  以上。ベクトル  $y$  が格納される。

### インターフェイス

#### Fortran 77:

```
SUBROUTINE mkl_dcootrsv(uplo, transa, diag, m, val, rowind, colind, nnz,
x, y)
  CHARACTER*1  uplo, transa, diag
  INTEGER      m, nnz
  INTEGER      rowind(*), colind(*)
  REAL*8       val(*), x(*), y(*)
```

#### Fortran 95:

```
SUBROUTINE mkl_dcootrsv(uplo, transa, diag, m, val, rowind, colind, nnz,
x, y)
  CHARACTER(LEN=1), INTENT(IN) :: uplo, transa, diag
  INTEGER, INTENT(IN) :: m, nnz
  INTEGER, INTENT(IN) :: rowind(*), colind(*)
  REAL(KIND(1.0D0)), INTENT(IN) :: val(*), x(*)
  REAL(KIND(1.0D0)), INTENT(INOUT) :: y(*)
```

#### C:

```
void mkl_dcootrsv(char *uplo, char *transa, char *diag, int *m, double
  *alpha, char *matdescra, double *val, int *rowind, int *colind, int
  *nnz, double *x, double *y);
```

---

## mkl\_ddiasv

対角形式のスパース行列について連立1次方程式を解く。

---

### 構文

#### Fortran:

```
call mkl_ddiasv(transa, m, alpha, matdescra, val, lval, iddiag, ndiag, x, y)
```

**C:**

```
mkl_ddiasv(&transa, &m, &alpha, matdescra, val, &lval, iddiag, &ndiag, x, y);
```

**説明**

mkl\_ddiasv ルーチンは、対角形式で格納されたスパース行列に対する行列 - ベクトル演算による連立 1 次方程式の解を算出する。

$$y := \alpha * \text{inverse\_of\_}(A) * x$$

または

$$y := \alpha * \text{inverse\_of\_}(A') * x$$

$\alpha$  はスカラ、 $x$  と  $y$  はベクトル、 $A$  は単位または非単位の主対角であるスパース上三角行列または下三角行列、 $A'$  は  $A$  の転置を示す。

**入力パラメータ**

パラメータの説明は、Fortran 77 の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「**インターフェイス**」のセクションで説明する。

<b>transa</b>	CHARACTER*1。実行する演算を指定する。 transa = 'N' または 'n' の場合 $y := \alpha * \text{inv}(A) * x$ transa = 'T'、't' または 'C'、'c' の場合、 $y := \alpha * \text{inv}(A') * x$
<b>m</b>	INTEGER。行列 $A$ の行数。
<b>alpha</b>	REAL*8。スカラ $\alpha$ を指定する。
<b>matdescra</b>	CHARACTER。配列の 6 個の成分は、演算で使用する行列のプロパティを指定する。最初の 3 個の配列成分のみ使用される。表 2-6 にそれぞれの配列成分に対して設定可能な値を示す。
<b>val</b>	REAL*8。lval × ndiag の 2 次元配列。行列 $A$ の非ゼロ対角成分を格納する。詳細は、対角格納方式の values 配列の説明を参照のこと。
<b>lval</b>	INTEGER。val, lval ≥ m のリーディング・ディメンジョン。詳細は、対角格納方式の lval 配列の説明を参照のこと。
<b>idiag</b>	INTEGER。長さ ndiag の配列。行列 $A$ の主対角と非ゼロ対角の距離を格納する。詳細は、対角格納方式の distance 配列の説明を参照のこと。
<b>ndiag</b>	INTEGER。行列 $A$ の非ゼロの対角成分を指定する。
<b>x</b>	REAL*8。配列、次元は m 以上。このルーチンに入る前に、配列 x にベクトル x を格納しなければならない。成分には単位増分を使用してアクセスする。

$y$  REAL\*8。配列、次元は  $m$  以上。このルーチンに入る前に、配列  $y$  にベクトル  $y$  を格納しなければならない。成分には単位増分を使用してアクセスする。

### 出力パラメータ

$y$  解として得られたベクトル  $x$  を格納する。

### インターフェイス

#### Fortran 77:

```
SUBROUTINE mkl_ddiasv(transa, m, alpha, matdescra, val, lval, idiag,
ndiag, x, y)
  CHARACTER*1    transa
  CHARACTER      matdescra(*)
  INTEGER         m, lval, ndiag
  INTEGER         idiag(*)
  REAL*8         alpha
  REAL*8         val(lval,*), x(*), y(*)
```

#### Fortran 95:

```
SUBROUTINE mkl_ddiasv(transa, m, alpha, matdescra, val, lval, idiag,
ndiag, x, y)
  CHARACTER(LEN=1), INTENT(IN) :: transa
  INTEGER, INTENT(IN) :: m, lval, ndiag
  CHARACTER, INTENT(IN) :: matdescra(*)
  INTEGER, INTENT(IN) :: idiag(*)
  REAL(KIND(1.0D0)), INTENT(IN) :: alpha
  REAL(KIND(1.0D0)), INTENT(IN) :: val(lval,*), x(*)
  REAL(KIND(1.0D0)), INTENT(INOUT) :: y(*)
```

#### C:

```
void mkl_ddiasv(char *transa, int *m, double *alpha, char *matdescra,
double *val, int *lval, int *idiag, int *ndiag, double *x, double *y);
```

## mkl\_ddiatsrv

対角形式のスパース行列に対する簡易インターフェイスを備えた三角ソルバ。

### 構文

#### Fortran:

```
call mkl_ddiatsrv(uplo, transa, diag, m, val, lval, idiag, ndiag, x, y)
```

#### C:

```
mkl_ddiatsrv(&uplo, &transa, &diag, &m, val, &lval, idiag, &ndiag, x, y);
```

### 説明

mkl\_ddiatsrv ルーチンは、対角形式で格納されたスパース行列に対する行列 - ベクトル演算による連立 1 次方程式の解を算出する。

$$A*y = x$$

または

$$A'*y = x$$

$x$  と  $y$  はベクトル、 $A$  は単位または非単位の主対角であるスパース上三角行列または下三角行列、 $A'$  は  $A$  の転置を示す。

### 入力パラメータ

パラメータの説明は、Fortran 77 の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「**インターフェイス**」のセクションで説明する。

**uplo** CHARACTER\*1。行列  $A$  の上三角と下三角のどちらを使用するかを指定する。

$uplo = 'U'$  または  $'u'$  の場合、行列  $A$  の上三角が使用される。

$uplo = 'L'$  または  $'l'$  の場合、行列  $A$  の下三角が使用される。

**transa** CHARACTER\*1。実行する演算を指定する。

$transa = 'N'$  または  $'n'$  の場合、 $A*y = x$

$transa = 'T'$ 、 $'t'$  または  $'C'$ 、 $'c'$  の場合、 $A'*y = x$

**diag** CHARACTER\*1。 $A$  が単位三角行列であるかどうかを指定する。

*diag* = 'U' または 'u' の場合、*A* は単位三角行列とみなされる。  
*diag* = 'N' または 'n' の場合、*A* は単位三角行列とみなされない。

*m*            INTEGER。行列 *A* の行数。

*val*           REAL\*8。 *lval* × *ndiag* の 2 次元配列。行列 *A* の非ゼロ対角成分を格納する。  
詳細は、[対角格納方式](#)の *values* 配列の説明を参照のこと。

*lval*           INTEGER。 *val*, *lval* ≥ *m* のリーディング・ディメンジョン。詳細は、[対角格納方式](#)の *lval* 配列の説明を参照のこと。

*idiag*          INTEGER。長さ *ndiag* の配列。行列 *A* の主対角と非ゼロ対角の距離を格納する。  
詳細は、[対角格納方式](#)の *distance* 配列の説明を参照のこと。

*ndiag*          INTEGER。行列 *A* の非ゼロの対角成分を指定する。

*x*            REAL\*8。配列、次元は *m*。このルーチンに入る前に、配列 *x* にベクトル *x* を格納しなければならない。

### 出力パラメータ

*y*            REAL\*8。配列、次元は *m* 以上。ベクトル *y* が格納される。

### インターフェイス

#### Fortran 77:

```
SUBROUTINE mkl_ddiattrsv(uplo, transa, diag, m, val, lval, idiag, ndiag,
x, y)
  CHARACTER*1    uplo, transa, diag
  INTEGER         m, lval, ndiag
  INTEGER         idiag(*)
  REAL*8          val(lval,*), x(*), y(*)
```

#### Fortran 95:

```
SUBROUTINE mkl_ddiattrsv(uplo, transa, diag, m, val, lval, idiag, ndiag,
x, y)
  CHARACTER(LEN=1), INTENT(IN) :: uplo, transa, diag
  INTEGER, INTENT(IN) :: m, lval, ndiag
  INTEGER, INTENT(IN) :: idiag(*)
  REAL(KIND(1.0D0)), INTENT(IN) :: alpha
  REAL(KIND(1.0D0)), INTENT(IN) :: val(lval,*), x(*)
  REAL(KIND(1.0D0)), INTENT(INOUT) :: y(*)
```



**C:**

```
void mkl_ddiattrsv(char *uplo, char *transa, char *diag, int *m, double
    *val, int *lval, int *idiag, int *ndiag, double *x, double *y);
```

---

## mkl\_dskysv

輪郭形式のスパース行列について連立 1 次方程式を解く。

---

### 構文

**Fortran:**

```
call mkl_dskysv(transa, m, alpha, matdescra, val, pntr, x, y)
```

**C:**

```
mkl_dskysv(&transa, &m, &alpha, matdescra, val, pntr, x, y);
```

### 説明

mkl\_dskysv ルーチンは、輪郭格納形式のスパース行列に対する行列 - ベクトル演算による連立 1 次方程式の解を算出する。

$$y := \alpha * \text{inv}(A) * x$$

または

$$y := \alpha * \text{inv}(A') * x$$

$\alpha$  はスカラ、 $x$  と  $y$  はベクトル、 $A$  は単位または非単位の主対角であるスパース上三角行列または下三角行列、 $A'$  は  $A$  の転置を示す。

### 入力パラメータ

パラメータの説明は、Fortran 77 の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「**インターフェイス**」のセクションで説明する。

**transa** CHARACTER\*1。実行する演算を指定する。

$\text{transa} = 'N'$  または  $'n'$  の場合  $y := \alpha * \text{inv}(A) * x$

$\text{transa} = 'T'$ 、 $'t'$  または  $'C'$ 、 $'c'$  の場合、  $y := \alpha * \text{inv}(A') * x$

**m** INTEGER。行列  $A$  の行数。

<i>alpha</i>	REAL*8。スカラ <i>alpha</i> を指定する。
<i>matdescra</i>	CHARACTER。配列の 6 個の成分は、演算で使用する行列のプロパティを指定する。最初の 3 個の配列成分のみ使用される。 <a href="#">表 2-6</a> にそれぞれの配列成分に対して設定可能な値を示す。
<i>val</i>	REAL*8。行列 <i>A</i> の成分のセットが輪郭プロファイル形式で格納された配列。 <i>matdescrsa</i> (2)= 'L' の場合、 <i>val</i> は行列 <i>A</i> の下三角の成分を格納する。 <i>matdescrsa</i> (2)= 'U' の場合、 <i>val</i> は行列 <i>A</i> の上三角の成分を格納する。 詳細は、 <a href="#">スカイライン格納方式</a> の <i>values</i> 配列の説明を参照のこと。
<i>pntr</i>	INTEGER。下三角の場合は長さ ( <i>m</i> +1)、上三角の場合は長さ ( <i>k</i> +1) の配列。 <i>val</i> で指定される、行列 <i>A</i> の各行 (または列) の最初の成分の位置のインデックスを格納する。詳細は、 <a href="#">対角格納方式</a> の <i>pointers</i> 配列の説明を参照のこと。
<i>x</i>	REAL*8。配列、次元は <i>m</i> 以上。このルーチンに入る前に、配列 <i>x</i> にベクトル <i>x</i> を格納しなければならない。成分には単位増分を使用してアクセスする。
<i>y</i>	REAL*8。配列、次元は <i>m</i> 以上。このルーチンに入る前に、配列 <i>y</i> にベクトル <i>y</i> を格納しなければならない。成分には単位増分を使用してアクセスする。

### 出力パラメータ

*y* 解として得られたベクトル *x* を格納する。

### インターフェイス

#### Fortran 77:

```
SUBROUTINE mkl_dskysv(transa, m, alpha, matdescra, val, pntr, x, y)
  CHARACTER*1  transa
  CHARACTER    matdescra(*)
  INTEGER      m
  INTEGER      pntr(*)
  REAL*8       alpha
  REAL*8       val(*), x(*), y(*)
```

#### Fortran 95:

```
SUBROUTINE mkl_dskysv(transa, m, alpha, matdescra, val, pntr, x, y)
  CHARACTER(LEN=1), INTENT(IN) :: transa
  INTEGER, INTENT(IN) :: m
  CHARACTER, INTENT(IN) :: matdescra(*)
  INTEGER, INTENT(IN) :: pntr(*)
```

```

REAL(KIND(1.0D0)), INTENT(IN) :: alpha
REAL(KIND(1.0D0)), INTENT(IN) :: val(*), x(*)
REAL(KIND(1.0D0)), INTENT(INOUT) :: y(*)

```

**C:**

```

void mkl_dskysv(char *transa, int *m, double *alpha, char *matdescra,
double *val, int *pntr, double *x, double *y);

```

---

## mkl\_dcsrmm

CSR 形式で格納されているスパース行列の  
行列- 行列の積を計算する。

---

### 構文

**Fortran:**

```

call mkl_dcsrmm(transa, m, n, k, alpha, matdescra, val, indx, pntrb,
pntrc, b, ldb, beta, c, ldc)

```

**C:**

```

mkl_dcsrmm(&transa, &m, &n, &k, &alpha, matdescra, val, indx, pntrb,
pntrc, b, &ldb, &beta, c, &ldc);

```

### 説明

mkl\_dcsrmm ルーチンは、次のように定義される行列 - 行列演算を実行する。

$$C := \alpha * A * B + \beta * C$$

または

$$C := \alpha * A' * B + \beta * C$$

$\alpha$  と  $\beta$  はスカラ、 $B$  と  $C$  は密行列、 $A$  は圧縮行形式の  $m \times k$  のスパース行列、 $A'$  は  $A$  の転置を示す。

### 入力パラメータ

パラメータの説明は、Fortran 77 の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「**インターフェイス**」のセクションで説明する。

*transa* CHARACTER\*1。実行する演算を指定する。

$transa = 'N'$  または  $'n'$  の場合、行列-行列の積は、次のように計算される。  
 $C := \alpha * A * B + \beta * C$

$transa = 'T'$ 、 $'t'$  または  $'C'$ 、 $'c'$  の場合、行列-ベクトルの積は次のように計算される。  
 $C := \alpha * A' * B + \beta * C$

<i>m</i>	INTEGER。行列 <i>A</i> の行数。
<i>n</i>	INTEGER。行列 <i>C</i> の列数。
<i>k</i>	INTEGER。行列 <i>A</i> の列数。
<i>alpha</i>	REAL*8。スカラ <i>alpha</i> を指定する。
<i>matdescra</i>	CHARACTER。配列の 6 個の成分は、演算で使用する行列のプロパティを指定する。最初の 3 個の配列成分のみ使用される。 <a href="#">表 2-6</a> にそれぞれの配列成分に対して設定可能な値を示す。
<i>val</i>	REAL*8。行列 <i>A</i> の非ゼロの成分を格納する配列。長さは、 $pntrb(m) - pntrb(1)$ である。詳細は、 <a href="#">CSR 形式</a> の <i>values</i> 配列の説明を参照のこと。
<i>indx</i>	INTEGER。行列 <i>A</i> の非ゼロの各成分の列インデックスを格納する配列。長さは、 <i>val</i> 配列の長さと同じ。詳細は、 <a href="#">CSR 形式</a> の <i>columns</i> 配列の説明を参照のこと。
<i>pntrb</i>	INTEGER。長さ <i>m</i> の配列。配列 <i>val</i> と配列 <i>indx</i> の行 <i>i</i> の最初のインデックスが $pntrb(i) - pntrb(1) + 1$ である行インデックスを格納する。詳細は、 <a href="#">CSR 形式</a> の <i>pointerB</i> 配列の説明を参照のこと。
<i>pntrc</i>	INTEGER。長さ <i>m</i> の配列。配列 <i>val</i> と配列 <i>indx</i> の行 <i>i</i> の最後のインデックスが $pntrc(i) - pntrb(1)$ である行インデックスを格納する。詳細は、 <a href="#">CSR 形式</a> の <i>pointerE</i> 配列の説明を参照のこと。
<i>b</i>	REAL*8。配列、次元は $(ldb, n)$ 。 $transa = 'N'$ または $'n'$ と指定した場合は、このルーチンに入る前に、配列 <i>b</i> の先頭の $k \times n$ の部分に行列 <i>B</i> を格納しなければならない。そうでない場合は、配列 <i>b</i> の先頭の $m \times n$ の部分に行列 <i>B</i> を格納しなければならない。
<i>ldb</i>	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 <i>b</i> の第 1 次元を指定する。
<i>beta</i>	REAL*8。スカラ <i>beta</i> を指定する。
<i>c</i>	REAL*8。配列、次元は $(ldc, n)$ 。このルーチンに入る前に、配列 <i>c</i> の先頭の $m \times n$ の部分に行列 <i>C</i> を格納しなければならない。そうでない場合は、配列 <i>c</i> の先頭の $k \times n$ の部分に行列 <i>C</i> を格納しなければならない。

*ldc* INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、*c* の第 1 次元を指定する。

### 出力パラメータ

*c* 行列  $(\alpha * A * B + \beta * C)$  または  $(\alpha * A' * B + \beta * C)$  によって上書きされる。

### インターフェイス

#### Fortran 77:

```
SUBROUTINE mkl_dcsrmm(transa, m, n, k, alpha, matdescra, val, indx,
pntrb, pntre, b, ldb, beta, c, ldc)
  CHARACTER*1  transa
  CHARACTER    matdescra(*)
  INTEGER      m, n, k, ldb, ldc
  INTEGER      indx(*), pntrb(m), pntre(m)
  REAL*8       alpha, beta
  REAL*8       val(*), b(ldb,*), c(ldc,*)
```

#### Fortran 95:

```
SUBROUTINE mkl_dcsrmm(transa, m, n, k, alpha, matdescra, val, indx,
pntrb, pntre, b, ldb, beta, c, ldc)
  CHARACTER(LEN=1), INTENT(IN):: transa
  INTEGER, INTENT(IN) :: m, n, k, ldb, ldc
  CHARACTER, INTENT(IN) :: matdescra(*)
  INTEGER, INTENT(IN) :: indx(*), pntrb(*), pntre(*)
  REAL(KIND(1.0D0)), INTENT(IN) :: alpha, beta
  REAL(KIND(1.0D0)), INTENT(IN) :: val(*), b(ldb,*)
  REAL(KIND(1.0D0)), INTENT(INOUT) :: c(ldc,*)
```

#### C:

```
void mkl_dcsrmm(char *transa, int *m, int *n, int *k, double *alpha, char
  *matdescra, double *val, int *indx, int *pntrb, int *pntre, double
  *b, int *ldb, double *beta, double *c, int *ldc,);
```

### mkl\_dcscmm

CSC 形式で格納されているスパース行列の  
行列 - 行列の積を計算する。

---

#### 構文

##### Fortran:

```
call mkl_dcscmm(transa, m, n, k, alpha, matdescra, val, indx, pntrb,  
                pntrc, b, ldb, beta, c, ldc)
```

##### C:

```
mkl_dcscmm(&transa, &m, &n, &k, &alpha, matdescra, val, indx, pntrb,  
            pntrc, b, &ldb, &beta, c, &ldc);
```

#### 説明

mkl\_dcscmm ルーチンは、次のように定義される行列 - 行列演算を実行する。

$C := \alpha * A * B + \beta * C$

または

$C := \alpha * A' * B + \beta * C$

$\alpha$  と  $\beta$  はスカラ、 $B$  と  $C$  は密行列、 $A$  は圧縮列形式の  $m \times k$  のスパース行列、 $A'$  は  $A$  の転置を示す。

#### 入力パラメータ

パラメータの説明は、Fortran 77 の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「**インターフェイス**」のセクションで説明する。

**transa** CHARACTER\*1。実行する演算を指定する。  
  
transa = 'N' または 'n' の場合、行列 - 行列の積は、次のように計算される。  
 $C := \alpha * A * B + \beta * C$   
  
transa = 'T'、't' または 'C'、'c' の場合、行列 - ベクトルの積は次のように計算される。  
 $C := \alpha * A' * B + \beta * C$

**m** INTEGER。行列  $A$  の行数。

<i>n</i>	INTEGER。行列 <i>C</i> の列数。
<i>k</i>	INTEGER。行列 <i>A</i> の列数。
<i>alpha</i>	REAL*8。スカラ <i>alpha</i> を指定する。
<i>matdescra</i>	CHARACTER。配列の 6 個の成分は、演算で使用する行列のプロパティを指定する。最初の 3 個の配列成分のみ使用される。 <a href="#">表 2-6</a> にそれぞれの配列成分に対して設定可能な値を示す。
<i>val</i>	REAL*8。行列 <i>A</i> の非ゼロの成分を格納する配列。長さは、 <i>pntrb</i> ( <i>k</i> ) - <i>pntrb</i> (1) である。詳細は、 <a href="#">CSC 形式</a> の <i>values</i> 配列の説明を参照のこと。
<i>indx</i>	INTEGER。行列 <i>A</i> の非ゼロの各成分の行インデックスを格納する配列。長さは、 <i>val</i> 配列の長さと等しい。詳細は、 <a href="#">CSC 形式</a> の <i>rows</i> 配列の説明を参照のこと。
<i>pntrb</i>	INTEGER。長さ <i>k</i> の配列。配列 <i>val</i> と配列 <i>indx</i> の列 <i>i</i> の最初のインデックスが <i>pntrb</i> ( <i>i</i> ) - <i>pntrb</i> (1)+1 である行インデックスを格納する。詳細は、 <a href="#">CSC 形式</a> の <i>pointerB</i> 配列の説明を参照のこと。
<i>pntrb</i>	INTEGER。長さ <i>k</i> の配列。配列 <i>val</i> と配列 <i>indx</i> の列 <i>i</i> の最後のインデックスが <i>pntrb</i> ( <i>i</i> ) - <i>pntrb</i> (1) である行インデックスを格納する。詳細は、 <a href="#">CSC 形式</a> の <i>pointerE</i> 配列の説明を参照のこと。
<i>b</i>	REAL*8。配列、次元は ( <i>ldb</i> , <i>n</i> )。 <i>transa</i> = 'N' または 'n' と指定した場合は、このルーチンに入る前に、配列 <i>b</i> の先頭の <i>k</i> × <i>n</i> の部分に行列 <i>B</i> を格納しなければならない。そうでない場合は、配列 <i>b</i> の先頭の <i>m</i> × <i>n</i> の部分に行列 <i>B</i> を格納しなければならない。
<i>ldb</i>	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 <i>b</i> の第 1 次元を指定する。
<i>beta</i>	REAL*8。スカラ <i>beta</i> を指定する。
<i>c</i>	REAL*8。配列、次元は ( <i>ldc</i> , <i>n</i> )。このルーチンに入る前に、配列 <i>c</i> の先頭の <i>m</i> × <i>n</i> の部分に行列 <i>C</i> を格納しなければならない。そうでない場合は、配列 <i>c</i> の先頭の <i>k</i> × <i>n</i> の部分に行列 <i>C</i> を格納しなければならない。
<i>ldc</i>	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 <i>c</i> の第 1 次元を指定する。

### 出力パラメータ

<i>c</i>	行列 ( $\alpha * A * B + \beta * C$ ) または ( $\alpha * A' * B + \beta * C$ ) によって上書きされる。
----------	---

### インターフェイス

#### Fortran 77:

```
SUBROUTINE mkl_dcscmm(transa, m, n, k, alpha, matdescra, val, indx,  
pntrb, pntre, b, ldb, beta, c, ldc)  
  CHARACTER*1  transa  
  CHARACTER    matdescra(*)  
  INTEGER      m, n, k, ldb, ldc  
  INTEGER      indx(*), pntrb(k), pntre(k)  
  REAL*8       alpha, beta  
  REAL*8       val(*), b(ldb,*), c(ldc,*)
```

#### Fortran 95:

```
SUBROUTINE mkl_dcscmm(transa, m, n, k, alpha, matdescra, val, indx,  
pntrb, pntre, b, ldb, beta, c, ldc)  
  CHARACTER(LEN=1), INTENT(IN):: transa  
  INTEGER, INTENT(IN) :: m, n, k, ldb, ldc  
  CHARACTER, INTENT(IN) :: matdescra(*)  
  INTEGER, INTENT(IN) :: indx(*), pntrb(*), pntre(*)  
  REAL(KIND(1.0D0)), INTENT(IN) :: alpha, beta  
  REAL(KIND(1.0D0)), INTENT(IN) :: val(*), b(ldb,*)  
  REAL(KIND(1.0D0)), INTENT(INOUT) :: c(ldc,*)
```

#### C:

```
void mkl_dcscmm(char *transa, int *m, int *n, int *k, double *alpha, char  
*matdescra, double *val, int *indx, int *pntrb, int *pntre, double  
*b, int *ldb, double *beta, double *c, int *ldc);
```

---

## mkl\_dcoomm

座標形式で格納されているスパース行列の  
行列-行列の積を計算する。

---

### 構文

#### Fortran:

```
call mkl_dcoomm(transa, m, n, k, alpha, matdescra, val, rowind, colind,  
nnz, b, ldb, beta, c, ldc)
```



**C:**

```

mkl_dcoomm(&transa, &m, &n, &k, &alpha, matdescra, val, rowind, colind,
           &nnz, b, &ldb, &beta, c, &ldc);

```

**説明**

mkl\_dcoomm ルーチンは、次のように定義される行列 - 行列演算を実行する。

$$C := \alpha * A * B + \beta * C$$

または

$$C := \alpha * A' * B + \beta * C$$

$\alpha$  と  $\beta$  はスカラ、 $B$  と  $C$  は密行列、 $A$  は座標形式の  $m \times k$  のスパース行列、 $A'$  は  $A$  の転置を示す。

**入力パラメータ**

パラメータの説明は、Fortran 77 の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「**インターフェイス**」のセクションで説明する。

**transa** CHARACTER\*1。実行する演算を指定する。

$\text{transa} = 'N'$  または  $'n'$  の場合、行列 - 行列の積は、次のように計算される。

$$C := \alpha * A * B + \beta * C$$

$\text{transa} = 'T'$ 、 $'t'$  または  $'C'$ 、 $'c'$  の場合、行列 - ベクトルの積は次のように計算される。

$$C := \alpha * A' * B + \beta * C$$

**m** INTEGER。行列  $A$  の行数。

**n** INTEGER。行列  $C$  の列数。

**k** INTEGER。行列  $A$  の列数。

**alpha** REAL\*8。スカラ  $\alpha$  を指定する。

**matdescra** CHARACTER。配列の 6 個の成分は、演算で使用する行列のプロパティを指定する。最初の 3 個の配列成分のみ使用される。[表 2-6](#) にそれぞれの配列成分に対して設定可能な値を示す。

**val** REAL\*8。長さ  $nnz$  の配列。行列  $A$  の非ゼロ成分を任意の順番で格納する。詳細は、[座標形式](#)の  $values$  配列の説明を参照のこと。

<i>rowind</i>	INTEGER。長さ <i>nnz</i> の配列。行列 <i>A</i> の各非ゼロ成分の行インデックスを格納する。詳細は、 <a href="#">座標形式</a> の <i>rows</i> 配列の説明を参照のこと。
<i>colind</i>	INTEGER。長さ <i>nnz</i> の配列。行列 <i>A</i> の各非ゼロ成分の列インデックスを格納する。詳細は、 <a href="#">座標形式</a> の <i>columns</i> 配列の説明を参照のこと。
<i>nnz</i>	INTEGER。行列 <i>A</i> の非ゼロの成分を指定する。詳細は、 <a href="#">座標形式</a> の <i>nnz</i> 配列の説明を参照のこと。
<i>b</i>	REAL*8。配列、次元は ( <i>ldb</i> , <i>n</i> )。 <i>transa</i> = 'N' または 'n' と指定した場合は、このルーチンに入る前に、配列 <i>b</i> の先頭の $k \times n$ の部分に行列 <i>B</i> を格納しなければならない。そうでない場合は、配列 <i>b</i> の先頭の $m \times n$ の部分に行列 <i>B</i> を格納しなければならない。
<i>ldb</i>	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 <i>b</i> の第 1 次元を指定する。
<i>beta</i>	REAL*8。スカラ <i>beta</i> を指定する。
<i>c</i>	REAL*8。配列、次元は ( <i>ldc</i> , <i>n</i> )。このルーチンに入る前に、配列 <i>c</i> の先頭の $m \times n$ の部分に行列 <i>C</i> を格納しなければならない。そうでない場合は、配列 <i>c</i> の先頭の $k \times n$ の部分に行列 <i>C</i> を格納しなければならない。
<i>ldc</i>	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 <i>c</i> の第 1 次元を指定する。

### 出力パラメータ

<i>c</i>	行列 ( $\alpha * A * B + \beta * C$ ) または ( $\alpha * A' * B + \beta * C$ ) によって上書きされる。
----------	---

### インターフェイス

#### Fortran 77:

```
SUBROUTINE mkl_dcoomm(transa, m, n, k, alpha, matdescra, val, rowind,
colind, nnz, b, ldb, beta, c, ldc)
  CHARACTER*1  transa
  CHARACTER    matdescra(*)
  INTEGER      m, n, k, ldb, ldc, nnz
  INTEGER      rowind(*), colind(*)
  REAL*8       alpha, beta
  REAL*8       val(*), b(ldb,*), c(ldc,*)
```

**Fortran 95:**

```

SUBROUTINE mkl_dcoomm(transa, m, n, k, alpha, matdescra, val, rowind,
colind, nnz, b, ldb, beta, c, ldc)
  CHARACTER(LEN=1), INTENT(IN):: transa
  INTEGER, INTENT(IN) :: m, n, k, ldb, ldc, nnz
  CHARACTER, INTENT(IN) :: matdescra(*)
  INTEGER, INTENT(IN) :: rowind(*), colind(*)
  REAL(KIND(1.0D0)), INTENT(IN) :: alpha, beta
  REAL(KIND(1.0D0)), INTENT(IN) :: val(*), b(ldb,*)
  REAL(KIND(1.0D0)), INTENT(INOUT) :: c(ldc,*)

```

**C:**

```

void mkl_dcoomm(char *transa, int *m, int *n, int *k, double *alpha, char
*matdescra, double *val, int *rowind, int *colind, int *nnz, double
*b, int *ldb, double *beta, double *c, int *ldc);

```

**mkl\_ddiamm**

対角形式で格納されているスパース行列の  
行列-行列の積を計算する。

**構文****Fortran:**

```

call mkl_ddiamm(transa, m, n, k, alpha, matdescra, val, lval, iddiag,
nddiag, b, ldb, beta, c, ldc)

```

**C:**

```

mkl_ddiamm(&transa, &m, &n, &k, &alpha, matdescra, val, &lval, iddiag,
&nddiag, b, &ldb, &beta, c, &ldc);

```

**説明**

mkl\_ddiamm ルーチンは、次のように定義される行列 - 行列演算を実行する。

$$C := \alpha * A * B + \beta * C$$

または

$$C := \alpha * A' * B + \beta * C$$

$\alpha$  と  $\beta$  はスカラー、 $B$  と  $C$  は密行列、 $A$  は対角形式の  $m \times k$  のスパース行列、 $A'$  は  $A$  の転置を示す。

### 入力パラメータ

パラメータの説明は、Fortran 77 の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「**インターフェイス**」のセクションで説明する。

<i>transa</i>	CHARACTER*1。実行する演算を指定する。  $transa = 'N'$ または $'n'$ の場合、行列-行列の積は、次のように計算される。 $C := \alpha * A * B + \beta * C$ $transa = 'T', 't'$ または $'C', 'c'$ の場合、行列-ベクトルの積は次のように計算される。 $C := \alpha * A' * B + \beta * C$
<i>m</i>	INTEGER。行列 $A$ の行数。
<i>n</i>	INTEGER。行列 $C$ の列数。
<i>k</i>	INTEGER。行列 $A$ の列数。
<i>alpha</i>	REAL*8。スカラー $\alpha$ を指定する。
<i>matdescra</i>	CHARACTER。配列の 6 個の成分は、演算で使用する行列のプロパティを指定する。最初の 3 個の配列成分のみ使用される。 <a href="#">表 2-6</a> にそれぞれの配列成分に対して設定可能な値を示す。
<i>val</i>	REAL*8。 $lval \times ndiag$ の 2 次元配列。行列 $A$ の非ゼロ対角成分を格納する。詳細は、 <a href="#">対角格納方式</a> の <i>values</i> 配列の説明を参照のこと。
<i>lval</i>	INTEGER。 $val$ , $lval \geq \min(m, k)$ のリーディング・ディメンジョン。詳細は、 <a href="#">対角格納方式</a> の <i>lval</i> 配列の説明を参照のこと。
<i>idiag</i>	INTEGER。長さ $ndiag$ の配列。行列 $A$ の主対角と非ゼロ対角の距離を格納する。詳細は、 <a href="#">対角格納方式</a> の <i>distance</i> 配列の説明を参照のこと。
<i>ndiag</i>	INTEGER。行列 $A$ の非ゼロの対角成分を指定する。
<i>b</i>	REAL*8。配列、次元は $(ldb, n)$ 。 $transa = 'N'$ または $'n'$ と指定した場合は、このルーチンに入る前に、配列 $b$ の先頭の $k \times n$ の部分に行列 $B$ を格納しなければならない。そうでない場合は、配列 $b$ の先頭の $m \times n$ の部分に行列 $B$ を格納しなければならない。
<i>ldb</i>	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 $b$ の第 1 次元を指定する。

*beta* REAL\*8。スカラ *beta* を指定する。

*c* REAL\*8。配列、次元は (*ldc*, *n*)。このルーチンに入る前に、配列 *c* の先頭の  $m \times n$  の部分に行列 *C* を格納しなければならない。そうでない場合は、配列 *c* の先頭の  $k \times n$  の部分に行列 *C* を格納しなければならない。

*ldc* INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、*c* の第 1 次元を指定する。

### 出力パラメータ

*c* 行列 ( $\alpha * A * B + \beta * C$ ) または ( $\alpha * A' * B + \beta * C$ ) によって上書きされる。

### インターフェイス

#### Fortran 77:

```
SUBROUTINE mkl_ddiamm(transa, m, n, k, alpha, matdescra, val, lval,
idiag, ndiag, b, ldb, beta, c, ldc)
  CHARACTER*1    transa
  CHARACTER       matdescra(*)
  INTEGER         m, n, k, ldb, ldc, lval, ndiag
  INTEGER         idiag(*)
  REAL*8          alpha, beta
  REAL*8          val(lval,*), b(ldb,*), c(ldc,*)
```

#### Fortran 95:

```
SUBROUTINE mkl_ddiamm(transa, m, n, k, alpha, matdescra, val, lval,
idiag, ndiag, b, ldb, beta, c, ldc)
  CHARACTER(LEN=1), INTENT(IN):: transa
  INTEGER, INTENT(IN) :: m, n, k, lval, ndiag, ldb, ldc
  CHARACTER, INTENT(IN) :: matdescra(*)
  INTEGER, INTENT(IN) :: idiag(*)
  REAL(KIND(1.0D0)), INTENT(IN) :: alpha, beta
  REAL(KIND(1.0D0)), INTENT(IN) :: val(*), b(ldb,*)
  REAL(KIND(1.0D0)), INTENT(INOUT) :: c(ldc,*)
```

#### C:

```
void mkl_ddiamm(char *transa, int *m, int *n, int *k, double *alpha, char
  *matdescra, double *val, int *lval, int *idiag, int *ndiag, double
  *b, int *ldb, double *beta, double *c, int *ldc);
```

### mkl\_dskymm

輪郭形式で格納されているスパース行列の  
行列-行列の積を計算する。

---

#### 構文

##### Fortran:

```
call mkl_dskymm(transa, m, n, k, alpha, matdescra, val, pntr, b, ldb,  
               beta, c, ldc)
```

##### C:

```
mkl_dskymm(&transa, &m, &n, &k, &alpha, matdescra, val, pntr, b, &ldb,  
           &beta, c, &ldc);
```

#### 説明

mkl\_dskymm ルーチンは、次のように定義される行列-行列演算を実行する。

$C := \alpha * A * B + \beta * C$

または

$C := \alpha * A' * B + \beta * C$

$\alpha$  と  $\beta$  はスカラ、 $B$  と  $C$  は密行列、 $A$  はスカイライン格納形式の  $m \times k$  のスパース行列、 $A'$  は  $A$  の転置を示す。

#### 入力パラメータ

パラメータの説明は、Fortran 77 の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「インターフェイス」のセクションで説明する。

<i>transa</i>	CHARACTER*1。実行する演算を指定する。  <i>transa</i> = 'N' または 'n' の場合、行列-行列の積は、次のように計算される。 $C := \alpha * A * B + \beta * C$  <i>transa</i> = 'T'、't' または 'C'、'c' の場合、行列-ベクトルの積は次のように計算される。 $C := \alpha * A' * B + \beta * C$
<i>m</i>	INTEGER。行列 $A$ の行数。

<i>n</i>	INTEGER。行列 <i>C</i> の列数。
<i>k</i>	INTEGER。行列 <i>A</i> の列数。
<i>alpha</i>	REAL*8。スカラ <i>alpha</i> を指定する。
<i>matdescra</i>	CHARACTER。配列の 6 個の成分は、演算で使用する行列のプロパティを指定する。最初の 3 個の配列成分のみ使用される。 <a href="#">表 2-6</a> にそれぞれの配列成分に対して設定可能な値を示す。
<i>val</i>	REAL*8。行列 <i>A</i> の成分のセット が輪郭プロファイル形式で格納された配列。 $matdescrsa(2) = 'L'$ の場合、 <i>val</i> は行列 <i>A</i> の下三角の成分を格納する。 $matdescrsa(2) = 'U'$ の場合、 <i>val</i> は行列 <i>A</i> の上三角の成分を格納する。 詳細は、 <a href="#">対角格納方式</a> の <i>values</i> 配列の説明を参照のこと。
<i>pntr</i>	INTEGER。下三角の場合は長さ ( $m+1$ )、上三角の場合は長さ ( $k+1$ ) の配列。 <i>val</i> で指定される、行列 <i>A</i> の各行 (または列) の最初の成分の位置のインデックスを格納する。詳細は、 <a href="#">対角格納方式</a> の <i>pointers</i> 配列の説明を参照のこと。
<i>b</i>	REAL*8。配列、次元は ( $ldb, n$ )。 $transa = 'N'$ または ' <i>n</i> ' と指定した場合は、このルーチンに入る前に、配列 <i>b</i> の先頭の $k \times n$ の部分に行列 <i>B</i> を格納しなければならない。そうでない場合は、配列 <i>b</i> の先頭の $m \times n$ の部分に行列 <i>B</i> を格納しなければならない。
<i>ldb</i>	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 <i>b</i> の第 1 次元を指定する。
<i>beta</i>	REAL*8。スカラ <i>beta</i> を指定する。
<i>c</i>	REAL*8。配列、次元は ( $ldc, n$ )。このルーチンに入る前に、配列 <i>c</i> の先頭の $m \times n$ の部分に行列 <i>C</i> を格納しなければならない。そうでない場合は、配列 <i>c</i> の先頭の $k \times n$ の部分に行列 <i>C</i> を格納しなければならない。
<i>ldc</i>	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 <i>c</i> の第 1 次元を指定する。

### 出力パラメータ

<i>c</i>	行列 ( $alpha * A * B + beta * C$ ) または ( $alpha * A' * B + beta * C$ ) によって上書きされる。
----------	---

### インターフェイス

#### Fortran 77:

```
SUBROUTINE mkl_dskymm(transa, m, n, k, alpha, matdescra, val, pntr, b,  
ldb, beta, c, ldc)  
  CHARACTER*1    transa  
  CHARACTER      matdescra(*)  
  INTEGER         m, n, k, ldb, ldc  
  INTEGER         pntr(*)  
  REAL*8         alpha, beta  
  REAL*8         val(*), b(ldb,*), c(ldc,*)
```

#### Fortran 95:

```
SUBROUTINE mkl_dskymm(transa, m, n, k, alpha, matdescra, val, pntr, b,  
ldb, beta, c, ldc)  
  CHARACTER(LEN=1), INTENT(IN):: transa  
  INTEGER, INTENT(IN) :: m, n, k, ldb, ldc  
  CHARACTER, INTENT(IN) :: matdescra(*)  
  INTEGER, INTENT(IN) :: pntr(*)  
  REAL(KIND(1.0D0)), INTENT(IN) :: alpha, beta  
  REAL(KIND(1.0D0)), INTENT(IN) :: val(*), b(ldb,*)  
  REAL(KIND(1.0D0)), INTENT(INOUT) :: c(ldc,*)
```

#### C:

```
void mkl_dskymm(char *transa, int *m, int *n, int *k, double *alpha, char  
  *matdescra, double *val, int *pntr, double *b, int *ldb, double  
  *beta, double *c, int *ldc);
```

---

## mkl\_dcsrsm

CSR 形式のスパース行列について連立 1 次行列  
方程式を解く。

---

### 構文

#### Fortran:

```
call mkl_dcsrsm(transa, m, n, alpha, matdescra, val, indx, pntrb, pntre,  
  b, ldb, c, ldc)
```



**C:**

```

mkl_dcsrsm(&transa, &m, &n, &alpha, matdescra, val, indx, pntrb, pntre,
           b, &ldb, c, &ldc);

```

**説明**

mkl\_dcsrsm ルーチンは、CSR 形式のスパース行列に対する行列 - 行列演算による連立 1 次方程式の解を算出する。

$$C := \alpha * \text{inv}(A) * B$$

または

$$C := \alpha * \text{inv}(A') * B$$

$\alpha$  はスカラ、 $B$  と  $C$  は密行列、 $A$  は単位または非単位の主対角であるスパース上三角行列または下三角行列、 $A'$  は  $A$  の転置を示す。

**入力パラメータ**

パラメータの説明は、Fortran 77 の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「**インターフェイス**」のセクションで説明する。

<i>transa</i>	CHARACTER*1。実行する演算を指定する。  $\text{transa} = 'N'$ または $'n'$ の場合、行列 - 行列の積は、次のように計算される。 $C := \alpha * \text{inv}(A) * B$ $\text{transa} = 'T'$ 、 $'t'$ または $'C'$ 、 $'c'$ の場合、行列 - ベクトルの積は次のように計算される。 $C := \alpha * \text{inv}(A') * B$
<i>m</i>	INTEGER。行列 $A$ の列数。
<i>n</i>	INTEGER。行列 $C$ の列数。
<i>alpha</i>	REAL*8。スカラ $\alpha$ を指定する。
<i>matdescra</i>	CHARACTER。配列の 6 個の成分は、演算で使用する行列のプロパティを指定する。最初の 3 個の配列成分のみ使用される。 <a href="#">表 2-6</a> にそれぞれの配列成分に対して設定可能な値を示す。
<i>val</i>	REAL*8。行列 $A$ の非ゼロの成分を格納する配列。長さは、 $\text{pntre}(m) - \text{pntrb}(1)$ である。詳細は、 <a href="#">CSR 形式</a> の <i>values</i> 配列の説明を参照のこと。

<i>indx</i>	INTEGER。行列 <i>A</i> の非ゼロの各成分の列インデックスを格納する配列。長さは、 <i>val</i> 配列の長さと等しい。詳細は、 <a href="#">CSR 形式</a> の <i>columns</i> 配列の説明を参照のこと。
<i>pntbrb</i>	INTEGER。長さ <i>m</i> の配列。配列 <i>val</i> と配列 <i>indx</i> の行 <i>i</i> の最初のインデックスが <i>pntbrb(i) - pntbrb(1)+1</i> である行インデックスを格納する。詳細は、 <a href="#">CSR 形式</a> の <i>pointerB</i> 配列の説明を参照のこと。
<i>pntre</i>	INTEGER。長さ <i>m</i> の配列。配列 <i>val</i> と配列 <i>indx</i> の行 <i>i</i> の最後のインデックスが <i>pntre(i) - pntbrb(1)</i> である行インデックスを格納する。詳細は、 <a href="#">CSR 形式</a> の <i>pointerE</i> 配列の説明を参照のこと。
<i>b</i>	REAL*8。配列、次元は ( <i>ldb, n</i> )。このルーチンに入る前に、配列 <i>b</i> の先頭の $m \times n$ の部分に行列 <i>B</i> を格納しなければならない。
<i>ldb</i>	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 <i>b</i> の第 1 次元を指定する。
<i>ldc</i>	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 <i>c</i> の第 1 次元を指定する。

### 出力パラメータ

<i>c</i>	REAL*8。配列、次元は ( <i>ldc, n</i> )。配列 <i>c</i> の先頭の $m \times n$ の部分に行列 <i>C</i> を格納する。
----------	--

### インターフェイス

#### Fortran 77:

```
SUBROUTINE mkl_dcsrsm(transa, m, n, alpha, matdescra, val, indx, pntbrb,
pntre, b, ldb, c, ldc)
  CHARACTER*1    transa
  CHARACTER       matdescra(*)
  INTEGER         m, n, ldb, ldc
  INTEGER         indx(*), pntbrb(m), pntre(m)
  REAL*8          alpha
  REAL*8          val(*), b(ldb,*), c(ldc,*)
```

#### Fortran 95:

```
SUBROUTINE mkl_dcsrsm(transa, m, n, alpha, matdescra, val, indx, pntbrb,
pntre, b, ldb, c, ldc)
  CHARACTER(LEN=1), INTENT(IN):: transa
  INTEGER, INTENT(IN) :: m, n, ldb, ldc
  CHARACTER, INTENT(IN) :: matdescra(*)
```

```

INTEGER, INTENT(IN) :: indx(*), pntrb(*), pntre(*)
REAL(KIND(1.0D0)), INTENT(IN) :: alpha
REAL(KIND(1.0D0)), INTENT(IN) :: val(*), b(ldb,*)
REAL(KIND(1.0D0)), INTENT(INOUT) :: c(ldc,*)

```

**C:**

```

void mkl_dcscsm(char *transa, int *m, int *n, double *alpha, char
    *matdescra, double *val, int *indx, int *pntrb, int *pntre, double
    *b, int *ldb, double *c, int *ldc);

```

---

## dcscsm

CSC 形式のスパース行列について連立 1 次行列  
方程式を解く。

---

### 構文

**Fortran:**

```

call mkl_dcscsm(transa, m, n, alpha, matdescra, val, indx, pntrb, pntre,
    b, ldb, c, ldc)

```

**C:**

```

mkl_dcscsm(&transa, &m, &n, &alpha, matdescra, val, indx, pntrb, pntre,
    b, &ldb, c, &ldc);

```

### 説明

mkl\_dcscsm ルーチンは、CSC 形式のスパース行列に対する行列 - 行列演算による連立 1 次方程式の解を算出する。

$C := \alpha \cdot \text{inv}(A) \cdot B$

または

$C := \alpha \cdot \text{inv}(A') \cdot B$

$\alpha$  はスカラ、 $B$  と  $C$  は密行列、 $A$  は単位または非単位の主対角であるスパース上三角行列または下三角行列、 $A'$  は  $A$  の転置を示す。

### 入力パラメータ

パラメータの説明は、Fortran 77 の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「**インターフェイス**」のセクションで説明する。

<i>transa</i>	CHARACTER*1。実行する演算を指定する。  <i>transa</i> = 'N' または 'n' の場合、行列-行列の積は、次のように計算される。 $C := \alpha * \text{inv}(A) * B$  <i>transa</i> = 'T'、't' または 'C'、'c' の場合、行列-ベクトルの積は次のように計算される。 $C := \alpha * \text{inv}(A') * B$
<i>m</i>	INTEGER。行列 <i>A</i> の列数。
<i>n</i>	INTEGER。行列 <i>C</i> の列数。
<i>alpha</i>	REAL*8。スカラ <i>alpha</i> を指定する。
<i>matdescra</i>	CHARACTER。配列の 6 個の成分は、演算で使用する行列のプロパティを指定する。最初の 3 個の配列成分のみ使用される。 <a href="#">表 2-6</a> にそれぞれの配列成分に対して設定可能な値を示す。
<i>val</i>	REAL*8。行列 <i>A</i> の非ゼロの成分を格納する配列。長さは、 <i>pntrb</i> ( <i>m</i> ) - <i>pntrb</i> (1) である。詳細は、 <a href="#">CSC 形式</a> の <i>values</i> 配列の説明を参照のこと。
<i>indx</i>	INTEGER。行列 <i>A</i> の非ゼロの各成分の行インデックスを格納する配列。長さは、 <i>val</i> 配列の長さと等しい。詳細は、 <a href="#">CSC 形式</a> の <i>rows</i> 配列の説明を参照のこと。
<i>pntrb</i>	INTEGER。長さ <i>k</i> の配列。配列 <i>val</i> と配列 <i>indx</i> の列 <i>i</i> の最初のインデックスが <i>pntrb</i> ( <i>i</i> ) - <i>pntrb</i> (1)+1 である行インデックスを格納する。詳細は、 <a href="#">CSC 形式</a> の <i>pointerB</i> 配列の説明を参照のこと。
<i>pntrb</i>	INTEGER。長さ <i>k</i> の配列。配列 <i>val</i> と配列 <i>indx</i> の列 <i>i</i> の最後のインデックスが <i>pntrb</i> ( <i>i</i> ) - <i>pntrb</i> (1) である行インデックスを格納する。詳細は、 <a href="#">CSC 形式</a> の <i>pointerE</i> 配列の説明を参照のこと。
<i>b</i>	REAL*8。配列、次元は ( <i>ldb</i> , <i>n</i> )。このルーチンに入る前に、配列 <i>b</i> の先頭の <i>m</i> × <i>n</i> の部分に行列 <i>B</i> を格納しなければならない。
<i>ldb</i>	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 <i>b</i> の第 1 次元を指定する。

*ldc* INTEGER。呼び出し元の ( サブ ) プログラムで宣言されているとおりに、*c* の第 1 次元を指定する。

### 出力パラメータ

*c* REAL\*8。配列、次元は (*ldc*, *n*)。配列 *c* の先頭の  $m \times n$  の部分に行列 *C* を格納する。

### インターフェイス

#### Fortran 77:

```
SUBROUTINE mkl_dcscsm(transa, m, n, alpha, matdescra, val, indx, pntrb,
  pntre, b, ldb, c, ldc)
  CHARACTER*1    transa
  CHARACTER      matdescra(*)
  INTEGER        m, n, ldb, ldc
  INTEGER        indx(*), pntrb(m), pntre(m)
  REAL*8         alpha
  REAL*8         val(*), b(ldb,*), c(ldc,*)
```

#### Fortran 95:

```
SUBROUTINE mkl_dcscsm(transa, m, n, alpha, matdescra, val, indx, pntrb,
  pntre, b, ldb, c, ldc)
  CHARACTER(LEN=1), INTENT(IN):: transa
  INTEGER, INTENT(IN) :: m, n, ldb, ldc
  CHARACTER, INTENT(IN) :: matdescra(*)
  INTEGER, INTENT(IN) :: indx(*), pntrb(*), pntre(*)
  REAL(KIND(1.0D0)), INTENT(IN) :: alpha
  REAL(KIND(1.0D0)), INTENT(IN) :: val(*), b(ldb,*)
  REAL(KIND(1.0D0)), INTENT(INOUT) :: c(ldc,*)
```

#### C:

```
void mkl_dcscsm(char *transa, int *m, int *n, double *alpha, char
  *matdescra, double *val, int *indx, int *pntrb, int *pntre, double
  *b, int *ldb, double *c, int *ldc);
```

### mkl\_dcoosm

座標形式のスパース行列について連立 1 次行列方程式を解く。

---

#### 構文

##### Fortran:

```
call mkl_dcoosm(transa, m, n, alpha, matdescra, val, rowind, colind,  
               nnz, b, ldb, c, ldc)
```

##### C:

```
mkl_dcoosm(&transa, &m, &n, &alpha, matdescra, val, rowind, colind,  
          &nnz, b, &ldb, c, &ldc);
```

#### 説明

mkl\_dcoosm ルーチンは、座標形式のスパース行列に対する行列 - 行列演算による連立 1 次方程式の解を算出する。

$C := \alpha * \text{inv}(A) * B$

または

$C := \alpha * \text{inv}(A') * B$

$\alpha$  はスカラ、 $B$  と  $C$  は密行列、 $A$  は単位または非単位の主対角であるスパース上三角行列または下三角行列、 $A'$  は  $A$  の転置を示す。

#### 入力パラメータ

パラメータの説明は、Fortran 77 の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「インターフェイス」のセクションで説明する。

**transa** CHARACTER\*1。実行する演算を指定する。

$\text{transa} = 'N'$  または  $'n'$  の場合、行列 - 行列の積は、次のように計算される。  
 $C := \alpha * \text{inv}(A) * B$

$\text{transa} = 'T'$ 、 $'t'$  または  $'C'$ 、 $'c'$  の場合、行列 - ベクトルの積は次のように計算される。  
 $C := \alpha * \text{inv}(A') * B$

**m** INTEGER。行列  $A$  の行数。

<i>n</i>	INTEGER。行列 <i>C</i> の列数。
<i>alpha</i>	REAL*8。スカラ <i>alpha</i> を指定する。
<i>matdescra</i>	CHARACTER。配列の 6 個の成分は、演算で使用する行列のプロパティを指定する。最初の 3 個の配列成分のみ使用される。 <a href="#">表 2-6</a> にそれぞれの配列成分に対して設定可能な値を示す。
<i>val</i>	REAL*8。長さ <i>nnz</i> の配列。行列 <i>A</i> の非ゼロ成分を任意の順番で格納する。詳細は、 <a href="#">座標形式</a> の <i>values</i> 配列の説明を参照のこと。
<i>rowind</i>	INTEGER。長さ <i>nnz</i> の配列。行列 <i>A</i> の各非ゼロ成分の行インデックスを格納する。詳細は、 <a href="#">座標形式</a> の <i>rows</i> 配列の説明を参照のこと。
<i>colind</i>	INTEGER。長さ <i>nnz</i> の配列。行列 <i>A</i> の各非ゼロ成分の列インデックスを格納する。詳細は、 <a href="#">座標形式</a> の <i>columns</i> 配列の説明を参照のこと。
<i>nnz</i>	INTEGER。行列 <i>A</i> の非ゼロの成分を指定する。詳細は、 <a href="#">座標形式</a> の <i>nnz</i> 配列の説明を参照のこと。
<i>b</i>	REAL*8。配列、次元は ( <i>ldb</i> , <i>n</i> )。このルーチンに入る前に、配列 <i>b</i> の先頭の $m \times n$ の部分に行列 <i>B</i> を格納しなければならない。
<i>ldb</i>	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 <i>b</i> の第 1 次元を指定する。
<i>ldc</i>	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 <i>c</i> の第 1 次元を指定する。

## 出力パラメータ

<i>c</i>	REAL*8。配列、次元は ( <i>ldc</i> , <i>n</i> )。配列 <i>c</i> の先頭の $m \times n$ の部分に行列 <i>C</i> を格納する。
----------	--

## インターフェイス

### Fortran 77:

```

SUBROUTINE mkl_dcoosm(transa, m, n, alpha, matdescra, val, rowind,
colind, nnz, b, ldb, c, ldc)
    CHARACTER*1    transa
    CHARACTER      matdescra(*)
    INTEGER        m, n, ldb, ldc, nnz
    INTEGER        rowind(*), colind(*)
    REAL*8         alpha
    REAL*8         val(*), b(ldb,*), c(ldc,*)

```

### Fortran 95:

```
SUBROUTINE mkl_dcoosm(transa, m, n, alpha, matdescra, val, rowind,
colind, nnz, b, ldb, c, ldc)
    CHARACTER(LEN=1), INTENT(IN):: transa
    INTEGER, INTENT(IN) :: m, n, ldb, ldc, nnz
    CHARACTER, INTENT(IN) :: matdescra(*)
    INTEGER, INTENT(IN) :: rowind(*), colind(*)
    REAL(KIND(1.0D0)), INTENT(IN) :: alpha
    REAL(KIND(1.0D0)), INTENT(IN) :: val(*), b(ldb,*)
    REAL(KIND(1.0D0)), INTENT(INOUT) :: c(ldc,*)
```

### C:

```
void mkl_dcoosm(char *transa, int *m, int *n, double *alpha, char
    *matdescra, double *val, int *rowind, int *colind, int *nnz, double
    *b, int *ldb, double *c, int *ldc);
```

---

## mkl\_ddiasm

対角形式のスパース行列について連立 1 次行列  
方程式を解く。

---

### 構文

#### Fortran:

```
call mkl_ddiasm(transa, m, n, alpha, matdescra, val, lval, iddiag, ndiag,
    b, ldb, c, ldc)
```

#### C:

```
mkl_ddiasm(&transa, &m, &n, &alpha, matdescra, val, &lval, iddiag, &ndiag,
    b, &ldb, c, &ldc);
```

### 説明

mkl\_ddiasm ルーチンは、対角形式のスパース行列に対する行列 - 行列演算による連立 1 次方程式の解を算出する。

$C := \alpha * \text{inv}(A) * B$

または

$C := \alpha * \text{inv}(A') * B$



$\alpha$  はスカラ、 $B$  と  $C$  は密行列、 $A$  は単位または非単位の主対角であるスパース上三角行列または下三角行列、 $A'$  は  $A$  の転置を示す。

## 入力パラメータ

パラメータの説明は、Fortran 77 の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「**インターフェイス**」のセクションで説明する。

<i>transa</i>	CHARACTER*1。実行する演算を指定する。  $transa = 'N'$ または $'n'$ の場合、行列-行列の積は、次のように計算される。 $C := \alpha * inv(A) * B$  $transa = 'T'$ 、 $'t'$ または $'C'$ 、 $'c'$ の場合、行列-ベクトルの積は次のように計算される。 $C := \alpha * inv(A') * B$
<i>m</i>	INTEGER。行列 $A$ の行数。
<i>n</i>	INTEGER。行列 $C$ の列数。
<i>alpha</i>	REAL*8。スカラ $\alpha$ を指定する。
<i>matdescra</i>	CHARACTER。配列の 6 個の成分は、演算で使用する行列のプロパティを指定する。最初の 3 個の配列成分のみ使用される。 <a href="#">表 2-6</a> にそれぞれの配列成分に対して設定可能な値を示す。
<i>val</i>	REAL*8。 $lval \times ndiag$ の 2 次元配列。行列 $A$ の非ゼロ対角成分を格納する。詳細は、 <a href="#">対角格納方式</a> の <i>values</i> 配列の説明を参照のこと。
<i>lval</i>	INTEGER。 <i>val</i> , $lval \geq m$ のリーディング・ディメンジョン。詳細は、 <a href="#">対角格納方式</a> の <i>lval</i> 配列の説明を参照のこと。
<i>idiag</i>	INTEGER。長さ <i>ndiag</i> の配列。行列 $A$ の主対角と非ゼロ対角の距離を格納する。詳細は、 <a href="#">対角格納方式</a> の <i>distance</i> 配列の説明を参照のこと。
<i>ndiag</i>	INTEGER。行列 $A$ の非ゼロの対角成分を指定する。
<i>b</i>	REAL*8。配列、次元は $(ldb, n)$ 。このルーチンに入る前に、配列 <i>b</i> の先頭の $m \times n$ の部分に行列 $B$ を格納しなければならない。
<i>ldb</i>	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 <i>b</i> の第 1 次元を指定する。
<i>ldc</i>	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 <i>c</i> の第 1 次元を指定する。

### 出力パラメータ

*c* REAL\*8。配列、次元は (*ldc*, *n*)。配列 *c* の先頭の  $m \times n$  の部分に行列 *C* を格納する。

### インターフェイス

#### Fortran 77:

```
SUBROUTINE mkl_ddiasm(transa, m, n, alpha, matdescra, val, lval, idiag,
ndiag, b, ldb, c, ldc)
    CHARACTER*1    transa
    CHARACTER       matdescra(*)
    INTEGER         m, n, ldb, ldc, lval, ndiag
    INTEGER         idiag(*)
    REAL*8          alpha
    REAL*8          val(lval,*), b(ldb,*), c(ldc,*)
```

#### Fortran 95:

```
SUBROUTINE mkl_ddiasm(transa, m, n, alpha, matdescra, val, lval, idiag,
ndiag, b, ldb, c, ldc)
    CHARACTER(LEN=1), INTENT(IN):: transa
    INTEGER, INTENT(IN) :: m, n, lval, ndiag, ldb, ldc
    CHARACTER, INTENT(IN) :: matdescra(*)
    INTEGER, INTENT(IN) :: idiag(*)
    REAL(KIND(1.0D0)), INTENT(IN) :: alpha
    REAL(KIND(1.0D0)), INTENT(IN) :: val(*), b(ldb,*)
    REAL(KIND(1.0D0)), INTENT(INOUT) :: c(ldc,*)
```

#### C:

```
void mkl_ddiasm(char *transa, int *m, int *n, double *alpha, char
    *matdescra, double *val, int *lval, int *idiag, int *ndiag, double
    *b, int *ldb, double *c, int *ldc);
```

## mkl\_dskysm

輪郭格納形式で格納されたスパース行列について  
連立 1 次行列方程式を解く。

### 構文

#### Fortran:

```
call mkl_dskysm(transa, m, n, alpha, matdescra, val, pntr, b, ldb, c, ldc)
```

#### C:

```
mkl_dskysm(&transa, &m, &n, &alpha, matdescra, val, pntr, b, &ldb, c, &ldc);
```

### 説明

mkl\_dskysm ルーチンは、輪郭格納形式のスパース行列に対する行列 - 行列演算による連立 1 次方程式の解を算出する。

$$C := \alpha * \text{inv}(A) * B$$

または

$$C := \alpha * \text{inv}(A') * B$$

$\alpha$  はスカラ、 $B$  と  $C$  は密行列、 $A$  は単位または非単位の主対角であるスパース上三角行列または下三角行列、 $A'$  は  $A$  の転置を示す。

### 入力パラメータ

パラメータの説明は、Fortran 77 の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「インターフェイス」のセクションで説明する。

<i>transa</i>	CHARACTER*1。実行する演算を指定する。  $\text{transa} = \text{'N'}$ または $\text{'n'}$ の場合、行列 - 行列の積は、次のように計算される。 $C := \alpha * \text{inv}(A) * B$ $\text{transa} = \text{'T'}$ 、 $\text{'t'}$ または $\text{'C'}$ 、 $\text{'c'}$ の場合、行列 - ベクトルの積は次のように計算される。 $C := \alpha * \text{inv}(A') * B$
<i>m</i>	INTEGER。行列 $A$ の行数。
<i>n</i>	INTEGER。行列 $C$ の列数。

<i>alpha</i>	REAL*8。スカラ <i>alpha</i> を指定する。
<i>matdescra</i>	CHARACTER。配列の 6 個の成分は、演算で使用する行列のプロパティを指定する。最初の 3 個の配列成分のみ使用される。 <a href="#">表 2-6</a> にそれぞれの配列成分に対して設定可能な値を示す。
<i>val</i>	REAL*8。行列 <i>A</i> の成分のセット が輪郭プロファイル形式で格納された配列。 <i>matdescrsa</i> (2)= 'L' の場合、 <i>val</i> は行列 <i>A</i> の下三角の成分を格納する。 <i>matdescrsa</i> (2)= 'U' の場合、 <i>val</i> は行列 <i>A</i> の上三角の成分を格納する。 詳細は、 <a href="#">対角格納方式</a> の <i>values</i> 配列の説明を参照のこと。
<i>pntr</i>	INTEGER。長さ ( <i>m</i> +1) の配列。 <i>val</i> で指定される、行列 <i>A</i> の各行 <i>i</i> (または列) の最初の非ゼロ成分の位置のインデックスを格納する。 <i>pointers</i> ( <i>i</i> )- <i>pointers</i> (1)+1 である。詳細は、 <a href="#">対角格納方式</a> の <i>pointers</i> 配列の説明を参照のこと。
<i>b</i>	REAL*8。配列、次元は ( <i>ldb</i> , <i>n</i> )。このルーチンに入る前に、配列 <i>b</i> の先頭の <i>m</i> × <i>n</i> の部分に行列 <i>B</i> を格納しなければならない。
<i>ldb</i>	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 <i>b</i> の第 1 次元を指定する。
<i>ldc</i>	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 <i>c</i> の第 1 次元を指定する。

### 出力パラメータ

<i>c</i>	REAL*8。配列、次元は ( <i>ldc</i> , <i>n</i> )。配列 <i>c</i> の先頭の <i>m</i> × <i>n</i> の部分に行列 <i>C</i> を格納する。
----------	---

### インターフェイス

#### Fortran 77:

```
SUBROUTINE mkl_dskysm(transa, m, n, alpha, matdescra, val, pntr, b, ldb,
c, ldc)
  CHARACTER*1    transa
  CHARACTER      matdescra(*)
  INTEGER        m, n, ldb, ldc
  INTEGER        pntr(*)
  REAL*8         alpha
  REAL*8         val(*), b(ldb,*), c(ldc,*)
```

**Fortran 95:**

```
SUBROUTINE mkl_dskysm(transa, m, n, alpha, matdescra, val, pntr, b, ldb,
c, ldc)
  CHARACTER(LEN=1), INTENT(IN):: transa
  INTEGER, INTENT(IN) :: m, n, ldb, ldc
  CHARACTER, INTENT(IN) :: matdescra(*)
  INTEGER, INTENT(IN) :: pntr(*)
  REAL(KIND(1.0D0)), INTENT(IN) :: alpha
  REAL(KIND(1.0D0)), INTENT(IN) :: val(*), b(ldb,*)
  REAL(KIND(1.0D0)), INTENT(INOUT) :: c(ldc,*)
```

**C:**

```
void mkl_dskysm(char *transa, int *m, int *n, double *alpha, char
  *matdescra, double *val, int *pntr, double *b, int *ldb, double *c,
  int *ldc,);
```



# LAPACK ルーチン： 1 次方程式

## 3

本章では、連立 1 次方程式の解の算出と、それに関連する一連の計算タスクを実行するための、LAPACK パッケージからインテル® マス・カーネル・ライブラリ (インテル® MKL) に組み込まれているルーチンについて説明する。このライブラリには、実データと複素データ用の LAPACK ルーチンが含まれる。

このライブラリは、以下のタイプの行列を持つ連立方程式用のルーチンをサポートしている。

- 一般行列
- 帯行列
- 対称 / エルミート正定値行列 (フル格納および圧縮格納形式)
- 対称 / エルミート正定値帯行列
- 対称 / エルミート不定値行列 (フル格納および圧縮格納形式)
- 対称 / エルミート不定値帯行列
- 三角行列 (フル格納および圧縮格納形式)
- 三角帯行列
- 三重対角行列

上記の行列タイプのそれぞれについて、ライブラリには、以下の計算を行うためのルーチンが用意されている。

- 行列の因子分解 (三角行列を除く)
- 行列の平衡化
- 連立 1 次方程式の解の算出
- 行列の条件数の推定
- 1 次方程式の解の精度の改善と誤差範囲の計算
- 行列の逆転の計算

特定の問題を解くために、2 つ以上の[計算ルーチン](#)を呼び出すか、1 回の呼び出しで複数のタスクを組み合わせる[ドライバルーチン](#) (例えば、因子分解と解の算出用の ?gesv) を呼び出せる。したがって、一般行列の連立 1 次方程式を解く場合、最初に ?getrf

(LU 因子分解) を呼び出し、次に ?getrs (解の算出) を呼び出せる。さらに、?gerfs を呼び出して、解の精度を改善し、誤差範囲を計算できる。代わりに、これらのタスクを 1 回の呼び出しで実行するドライバルーチン ?gesvx を使用することもできる。



**警告** :LAPACK ルーチンは、入力行列に INF 値または NaN 値が含まれないことを前提としている。LAPACK に対する入力データが適当でないとき、コンピュータのハングアップなどの問題が発生する場合がある。

インテル MKL 8.0 以降では、LAPACK 計算ルーチンおよびドライバルーチンに対する Fortran-77 インターフェイスとともに、より短い引数リストで簡略化したルーチン呼び出しを使用する Fortran-95 インターフェイスもサポートしている。Fortran-95 インターフェイスの呼び出しシーケンスは、ルーチン説明の「構文」セクションで、Fortran-77 呼び出しの説明の後に解説する。

### ルーチン命名規則

本章の各ルーチンについては、Fortran-77 プログラムから呼び出す際に、LAPACK 名を使用できる。

[表 3-1](#) と [表 3-2](#) に、**LAPACK 名** の一覧を示す。LAPACK 名は、以下に説明するように、?yyzzz または ?yyzz の構造になっている。

最初の記号 ? は、データ型を示す。

s	実数、単精度	c	複素数、単精度
d	実数、倍精度	z	複素数、倍精度

2 番目と 3 番目の文字 yy は、行列のタイプと格納形式を示す。

ge	一般行列
gb	一般帯行列
gt	一般三重対角行列
po	対称 / エルミート 正定値行列
pp	対称 / エルミート 正定値行列 (圧縮格納形式)
pb	対称 / エルミート 正定値帯行列
pt	対称 / エルミート 正定値三重対角行列
sy	対称不定値行列
sp	対称不定値行列 (圧縮格納形式)
he	エルミート不定値行列
hp	エルミート不定値行列 (圧縮格納形式)
tr	三角行列



tp 三角行列 (圧縮格納形式)  
 tb 三角帯行列

計算ルーチンでは、最後の 3 つの文字 `zzz` は、実行される処理を示す。

trf 三角行列の因子分解を実行する  
 trs 因子分解された行列を使って連立 1 次方程式を解く  
 con 行列の条件数を推定する  
 rfs 解の精度を改善し、誤差範囲を計算する  
 tri 因子分解を使用して逆行列を計算する  
 equ 行列を平衡化する

例えば、ルーチン `sgetrf` は、単精度実数型の一般行列の三角分解を実行する。これに対応する複素行列用のルーチンは、`cgetrf` である。

ドライバルーチンには、名前の最後に `-sv` (簡易ドライバ) または `-svx` (高度ドライバ) が付く。

インテル MKL において、Fortran-95 インターフェイスの LAPACK 計算ルーチン名およびドライバルーチン名は、最初の文字がデータ型を示す以外は Fortran-77 の名前と同じである。例えば、Fortran-95 インターフェイスで、実数型の一般行列の三角分解を実行するルーチン名は `getrf` である。異なるデータ型を扱うには、単精度および倍精度の名前付き定数でモジュールブロックを参照する特定の入力パラメータを定義して行う。

## Fortran-95 インターフェイス規則

LAPACK に対する Fortran-95 インターフェイスは、それぞれの Fortran-77 ルーチンを呼び出すラッパーを通して実装される。このインターフェイスは、形状引継ぎ配列や、より少ない引数で簡略化した LAPACK ルーチンの呼び出しを提供するオプション引数などの Fortran-95 の機能を使用する。

Fortran-95 インターフェイスで使用される主な規則を以下に示す。

- Fortran-95 呼び出しで使用される引数名は、一般的にそれぞれの標準 (Fortran-77) インターフェイスと同じである。ただし、ライブラリで使用される引数名の数を減らすための、簡略化された引数名を以下に示す。

標準引数名	Fortran-95 引数名
<i>ap</i>	<i>a</i>
<i>ab</i>	<i>a</i>
<i>afb</i>	<i>af</i>
<i>afp</i>	<i>af</i>

標準引数名	Fortran-95 引数名
<i>bp</i>	<i>b</i>
<i>bb</i>	<i>b</i>
<i>selctg</i>	<i>select</i>

これらの正式引数名の変更はプログラムの動作には影響を与えず、また統一規則に従うものである。

- 配列次元などの入力引数は Fortran-95 では不要であり、呼び出しシーケンスからスキップされる。配列次元は、必要な配列形状に正確に従うユーザデータから再構築される。

Fortran-95 インターフェイスでスキップされるその他の標準引数のタイプは、*work*、*rwork*、などのワークスペース配列を表す引数である。ただし、ワークスペース配列が出力時に重要な情報を返す場合は例外である。

また、引数は、呼び出しシーケンス内の他の引数の有無により、その値が完全に定義される場合にスキップされる。復元値はスキップされた引数にとってのみ意味のある値である。

- いくつかの標準引数は Fortran-95 インターフェイスではオプションとして宣言され、呼び出しシーケンスから省略される場合がある。以下の条件のいずれかを満たす場合は、引数をオプションとして宣言できる。
  - 引数の値が呼び出しシーケンス内の他の引数の有無により完全に定義される場合、オプションとして宣言できる。この場合のスキップされる引数との違いは、オプション引数はデフォルトで再構築される値とは異なる意味を持つことである。  
例えば、*jobz* という引数が 2 つの値を持ち、その中の 1 つの値が直接もう一方の引数の使用を暗黙的に示す場合、*jobz* の値は実際の 2 番目の引数の有無により一意的に再構築され、*jobz* は省略できる。
  - 入力引数が、ごくわずかの設定可能な値を持つ場合、オプションとして宣言できる。これらの引数のデフォルト値は、通常、リストの最初の値として設定される。この規則における例外のすべてはルーチンの説明で明示的に述べる。
  - 入力引数が自然デフォルト値を持つ場合は、オプションとして宣言できる。これらのオプション引数のデフォルト値は、自然デフォルト値に設定される。
- INFO* 引数は、Fortran-95 インターフェイスではオプションとして宣言される。呼び出しシーケンス内にある場合、*INFO* に設定された値の解釈を以下に示す。
  - 値が -1000 よりも大きい場合、Fortran-77 ルーチンと同様の意味を持つ。
  - 値が -1000 の場合、作業メモリの不足を意味する。

3. 値が -1001 の場合、矛盾した引数が呼び出しシーケンスに存在することを意味する。

- Fortran-95 呼び出し構文では、オプション引数を大括弧 ([]) で表現する。

オプション・パラメータの値の再構築に使用される具体的な規定は、各ルーチンごとに特有で、ルーチン仕様の最後の各「Fortran-95 ノート」で詳細を説明する。

## 行列の格納形式

LAPACK ルーチンでは、以下の行列格納形式を使用している。

- フル格納では、行列  $A$  は 2 次元配列  $a$  に格納され、行列成分  $a_{ij}$  は配列成分  $a(i, j)$  に格納される。
- 圧縮格納では、対称行列、エルミート行列、または三角行列をコンパクトに格納できる。行列の上三角または下三角が 1 次元配列の各列に圧縮される。
- 帯格納では、 $kl$  個の劣対角成分と  $ku$  個の優対角成分を持つ  $m \times n$  の帯行列は、 $(kl+ku+1)$  行  $n$  列の 2 次元配列  $ab$  にコンパクトに格納される。行列の各列は配列の対応する列に格納され、行列の各対角成分は配列の各行に格納される。

第 4 章と第 5 章では、圧縮格納形式で行列を格納する配列は名前の最後の文字を  $p$  で、帯格納形式で行列を  $i$  格納する配列には名前の最後の文字を  $b$  で示している。

行列の格納形式の詳細は、付録 B の「[行列引数](#)」を参照。

## 数学的表記

LAPACK ルーチンの説明では、以下の表記を使用している。

$Ax = b$	$n \times n$ の行列 $A = \{a_{ij}\}$ 、右辺のベクトル $b = \{b_i\}$ 、未知のベクトル $x = \{x_i\}$ を持つ連立 1 次方程式。
$AX = B$	共通の行列 $A$ と複数の右辺を持つ連立方程式の集合。 $B$ の各列は右辺のそれぞれに相当する。 $X$ の各列は対応する解である。
$ x $	成分 $ x_i $ ( $x_i$ の絶対値) を持つベクトル。
$ A $	成分 $ a_{ij} $ ( $a_{ij}$ の絶対値) を持つ行列。
$\ x\ _\infty = \max_i  x_i $	ベクトル $x$ の無限ノルム。
$\ A\ _\infty = \max_j \sum_i  a_{ij} $	行列 $A$ の無限ノルム。
$\ A\ _1 = \max_j \sum_i  a_{ij} $	行列 $A$ の 1- ノルム。 $\ A\ _1 = \ A^T\ _\infty = \ A^H\ _\infty$

$\kappa(A) = \|A\| \|A^{-1}\|$       行列  $A$  の条件数。

### 誤差の分析

実際には、ほとんどの計算で実行時に丸め誤差が発生する。また、連立方程式  $Ax = b$  で、データ ( $A$  と  $b$  の成分) が正確にわかっていない場合もある。したがって、データの誤差と丸め誤差が解  $x$  に与える影響を理解する必要がある。

**データの摂動。**  $x$  が  $Ax = b$  の正確な解であり、 $x + \delta x$  が摂動問題  $(A + \delta A)x = (b + \delta b)$  の正確な解である場合は、次の式が成り立つ。

$$\frac{\|\delta x\|}{\|x\|} \leq \kappa(A) \left( \frac{\|\delta A\|}{\|A\|} + \frac{\|\delta b\|}{\|b\|} \right), \quad \kappa(A) = \|A\| \|A^{-1}\|$$

つまり、 $A$  または  $b$  の相対誤差は、解のベクトル  $x$  内で係数  $\kappa(A) = \|A\| \|A^{-1}\|$  によって増幅される場合がある。この係数を、 $A$  の条件数と呼ぶ。

**丸め誤差** は、元のデータの相対摂動  $c(n)\varepsilon$  と同じ効果を持つ。 $\varepsilon$  はマシンの精度、 $c(n)$  は行列の次数  $n$  の適度な関数である。対応する解の誤差は、 $\|\delta x\|/\|x\| \leq c(n)\kappa(A)\varepsilon$  になる ( $c(n)$  の値が  $10n$  より大きくなることはほとんどない)。

したがって、行列  $A$  が悪条件である場合 (つまり、条件数  $\kappa(A)$  が非常に大きい場合) は、解  $x$  の誤差も大きくなる。ときには、精度が全く失われてしまう場合もある。

LAPACK には、 $\kappa(A)$  を推定するためのルーチンが含まれている (「[条件数を推定するためのルーチン](#)」を参照)。また、実際の解の誤差をより正確に推定する方法もある (「[解の精度の改善と誤差の推定](#)」を参照)。

## 計算ルーチン

表 3-1 に、実数行列の因子分解、平衡化、逆行列の計算、実数行列の条件数の推定、実数行列を持つ連立方程式の解の算出、解の精度の改善、解の誤差の推定を行うための LAPACK 計算ルーチン (Fortran-77 インターフェイス) を示す。表 3-2 に、これに対応する複素行列用のルーチンを示す。Fortran-95 インターフェイスのそれぞれのルーチン名には、最初の記号をつけない (「[ルーチン命名規則](#)」を参照)。

表 3-1 実数行列を持つ連立方程式用の計算ルーチン

行列のタイプ、 格納形式	行列の 因子分解	行列の 平衡化	連立 方程式 の解の 算出	条件数	誤差の 推定	逆行列の 計算
一般行列	<a href="#">?getrf</a>	<a href="#">?geequ</a>	<a href="#">?getrs</a>	<a href="#">?gecon</a>	<a href="#">?gerfs</a>	<a href="#">?getri</a>
一般帯	<a href="#">?gbtrf</a>	<a href="#">?gbequ</a>	<a href="#">?gbtrs</a>	<a href="#">?gbcon</a>	<a href="#">?gbrfs</a>	
一般三重対角	<a href="#">?gttrf</a>		<a href="#">?gttrs</a>	<a href="#">?gtcon</a>	<a href="#">?gtrfs</a>	
対称正定値	<a href="#">?potrf</a>	<a href="#">?poequ</a>	<a href="#">?potrs</a>	<a href="#">?pocon</a>	<a href="#">?porfs</a>	<a href="#">?potri</a>
対称正定値 圧縮格納	<a href="#">?pptrf</a>	<a href="#">?ppequ</a>	<a href="#">?pptrs</a>	<a href="#">?ppcon</a>	<a href="#">?pprfs</a>	<a href="#">?pptri</a>
対称正定値 帯	<a href="#">?pbtrf</a>	<a href="#">?pbequ</a>	<a href="#">?pbtrs</a>	<a href="#">?pbcon</a>	<a href="#">?pbrfs</a>	
対称正定値 三重対角	<a href="#">?pttrf</a>		<a href="#">?pttrs</a>	<a href="#">?ptcon</a>	<a href="#">?ptrfs</a>	
対称不定値	<a href="#">?sytrf</a>		<a href="#">?sytrs</a>	<a href="#">?sycon</a>	<a href="#">?syrrfs</a>	<a href="#">?sytri</a>
対称不定値 圧縮格納	<a href="#">?sptrf</a>		<a href="#">?sptrs</a>	<a href="#">?spcon</a>	<a href="#">?sprfs</a>	<a href="#">?sptri</a>
三角			<a href="#">?trtrs</a>	<a href="#">?trcon</a>	<a href="#">?trrrfs</a>	<a href="#">?trtri</a>
三角圧縮 格納			<a href="#">?tptrs</a>	<a href="#">?tpcon</a>	<a href="#">?tprfs</a>	<a href="#">?tptri</a>
三角帯			<a href="#">?tbtrs</a>	<a href="#">?tbcon</a>	<a href="#">?tbrfs</a>	

表中の ? は、Fortran-77 インターフェイスの **s** (単精度) または **d** (倍精度) を示す。

表 3-2 複素行列を持つ連立方程式用の計算ルーチン

行列のタイプ、 格納形式	行列の 因子分解	行列の 平衡化	連立 方程式 の解の 算出	条件数	誤差の 推定	逆行列 の計算
一般行列	<a href="#">?getrf</a>	<a href="#">?geequ</a>	<a href="#">?getrs</a>	<a href="#">?gecon</a>	<a href="#">?gerfs</a>	<a href="#">?getri</a>
一般帯	<a href="#">?gbtrf</a>	<a href="#">?gbequ</a>	<a href="#">?gbtrs</a>	<a href="#">?gbcon</a>	<a href="#">?gbrfs</a>	
一般三重対角	<a href="#">?gttrf</a>		<a href="#">?gttrs</a>	<a href="#">?gtcon</a>	<a href="#">?gtrfs</a>	
エルミート正定値	<a href="#">?potrf</a>	<a href="#">?poequ</a>	<a href="#">?potrs</a>	<a href="#">?pocon</a>	<a href="#">?porfs</a>	<a href="#">?potri</a>
エルミート正定値 圧縮格納	<a href="#">?pptrf</a>	<a href="#">?ppequ</a>	<a href="#">?pptrs</a>	<a href="#">?ppcon</a>	<a href="#">?pprfs</a>	<a href="#">?pptri</a>
エルミート正定値 帯	<a href="#">?pbtrf</a>	<a href="#">?pbequ</a>	<a href="#">?pbtrs</a>	<a href="#">?pbcon</a>	<a href="#">?pbrfs</a>	
エルミート正定値 三重対角	<a href="#">?pttrf</a>		<a href="#">?pttrs</a>	<a href="#">?ptcon</a>	<a href="#">?ptrfs</a>	
エルミート不定値	<a href="#">?hetrf</a>		<a href="#">?hetrs</a>	<a href="#">?hecon</a>	<a href="#">?herfs</a>	<a href="#">?hetri</a>
対称不定値	<a href="#">?sytrf</a>		<a href="#">?sytrs</a>	<a href="#">?sycon</a>	<a href="#">?syrrfs</a>	<a href="#">?sytri</a>
エルミート不定値 圧縮格納	<a href="#">?hptrf</a>		<a href="#">?hptrs</a>	<a href="#">?hpcon</a>	<a href="#">?hprfs</a>	<a href="#">?hptri</a>
対称不定値 圧縮格納	<a href="#">?sptrf</a>		<a href="#">?sptrs</a>	<a href="#">?spcon</a>	<a href="#">?sprfs</a>	<a href="#">?sptri</a>
三角			<a href="#">?trtrs</a>	<a href="#">?trcon</a>	<a href="#">?trrrfs</a>	<a href="#">?trtri</a>
三角圧縮 格納			<a href="#">?tptrs</a>	<a href="#">?tpcon</a>	<a href="#">?tprfs</a>	<a href="#">?tptri</a>
三角帯			<a href="#">?tbtrs</a>	<a href="#">?tbcon</a>	<a href="#">?tbrfs</a>	

表中の ? は、Fortran-77 インターフェイスの **c** (単精度複素数) または **z** (倍精度複素数) を示す。

## 行列の因子分解用のルーチン

この節では、行列の因子分解用の LAPACK ルーチンについて説明する。以下の因子分解をサポートしている。

- *LU* 因子分解
- 実対称正定値行列のコレスキー因子分解
- エルミート正定値行列のコレスキー因子分解

- 実対称行列と複素対称行列の Bunch-Kaufman 因子分解
- エルミート行列の Bunch-Kaufman 因子分解

$LU$  因子分解の計算には、フル格納と帯格納形式の行列を使用できる。コレスキー因子分解では、フル格納、圧縮格納、帯格納形式を使用できる。Bunch-Kaufman 因子分解では、フル格納と圧縮格納形式を使用できる。

---

## ?getrf

$m \times n$  の一般行列の  $LU$  因子分解を行う。

---

### 構文

#### Fortran 77:

```
call sgetrf(m, N, a, lda, Ipiv, info)
call dgetrf(m, N, a, lda, Ipiv, info)
call cgetrf(m, N, a, lda, Ipiv, info)
call zgetrf(m, N, a, lda, Ipiv, info)
```

#### Fortran 95:

```
call getrf(a [,ipiv] [,info])
```

### 説明

このルーチンは、次の式に従って、 $m \times n$  の一般行列  $A$  の  $LU$  因子分解を実行する。

$$A = PLU,$$

$P$  は置換行列、 $L$  は単位対角成分を含む下三角 ( $m > n$  の場合は下台形)、 $U$  は上三角 ( $m < n$  の場合は上台形) である。通常は、 $A$  は正方行列 ( $m = n$ ) で、 $L$  と  $U$  はいずれも三角行列である。このルーチンでは、行を交換し、部分的にピボット演算を行う。

### 入力パラメータ

$m$                     INTEGER。行列  $A$  の行数 ( $m \geq 0$ )。  
 $n$                     INTEGER。  $A$  の列数 ( $n \geq 0$ )。

*a* REAL (sgetrf の場合 )  
 DOUBLE PRECISION (dgetrf の場合 )  
 COMPLEX (cgetrf の場合 )  
 DOUBLE COMPLEX (zgetrf の場合 )。  
 配列、次元は (*lda*, \*)。行列 *A* を格納する。  
*a* の第 2 次元は、 $\max(1, n)$  以上でなければならない。

*lda* INTEGER。 *a* の第 1 次元。

## 出力パラメータ

*a* *L* と *U* によって上書きされる。*L* の単位対角成分は格納されない。

*ipiv* INTEGER。  
 配列、次元は  $\max(1, \min(m, n))$  以上。  
 ピボットのインデックス: 行 *i* は行 *ipiv*(*i*) と交換される。

*info* INTEGER。 *info* = 0 の場合、実行は正常に終了した。  
*info* = -*i* の場合、*i* 番目のパラメータの値が不正である。  
*info* = *i* の場合、*u*<sub>*ii*</sub> は 0 である。因子分解は完了したが *U* は完全に特異である。連立 1 次方程式の解の算出に係数 *U* を使用すると、0 による除算が発生する。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

getrf ルーチンのインターフェイス特有の詳細を以下に示す。

*a* サイズ (*m*, *n*) の行列 *A* を格納する。

*ipiv* 長さ  $\min(m, n)$  のベクトルを格納する。

## アプリケーション・ノート

算出された *L* と *U* は、摂動行列 *A* + *E* の正確な係数になる。  
 $|E| \leq c(\min(m, n))\epsilon P|L||U|$

*c*(*n*) は *n* の適度な 1 次関数で、 $\epsilon$  はマシンの精度である。

実数型の浮動小数点演算のおおよその回数は次のとおりである。

$(2/3)n^3$  (*m* = *n* の場合 )



$(1/3)n^2(3m-n)$  ( $m > n$  の場合)

$(1/3)m^2(3n-m)$  ( $m < n$  の場合)

複素数型の演算回数は、この 4 倍になる。

$m = n$  でこのルーチン呼び出した後、以下のルーチン呼び出せる。

[?getrs](#)  $AX = B$  または  $A^T X = B$  または  $A^H X = B$  を解く。

[?gecon](#)  $A$  の条件数を推定する。

[?getri](#)  $A$  の逆行列を計算する。

---

## ?gbtrf

$m \times n$  の一般帯行列の LU 因子分解を行う。

---

### 構文

#### Fortran 77:

```
call sgbtrf(m, N, kl, ku, ab, ldab, Ipiv, info)
call dgbtrf(m, N, kl, ku, ab, ldab, Ipiv, info)
call cgbtrf(m, N, kl, ku, ab, ldab, Ipiv, info)
call zgbtrf(m, N, kl, ku, ab, ldab, Ipiv, info)
```

#### Fortran 95:

```
call gbtrf(a [,kl] [,m] [,Ipiv] [,Info])
```

### 説明

このルーチンは、 $kl$  個の非ゼロの劣対角成分と  $ku$  個の非ゼロの優対角成分を含む  $m \times n$  の帯行列  $A$  の LU 因子分解を実行する。通常は、 $A$  は正方行列 ( $m = n$ ) で、次の式が成り立つ。

$$A = PLU$$

$P$  は置換行列である。 $L$  は、単位対角成分を含む、各列の非ゼロの成分の数が  $kl$  個以下の下三角行列である。 $U$  は、 $kl + ku$  個の優対角成分を含む上三角帯行列である。このルーチンは、行を交換して、部分的にピボット演算を行う ( 行の交換によって、 $U$  に  $kl$  個の優対角成分が追加される )。

## 入力パラメータ

<i>m</i>	INTEGER。行列 <i>A</i> の行数 ( $m \geq 0$ )。
<i>n</i>	INTEGER。 <i>A</i> の列数 ( $n \geq 0$ )。
<i>kl</i>	INTEGER。 <i>A</i> の帯内の劣対角成分の数 ( $kl \geq 0$ )。
<i>ku</i>	INTEGER。 <i>A</i> の帯内の優対角成分の数 ( $ku \geq 0$ )。
<i>ab</i>	REAL (sgbtrf の場合 ) DOUBLE PRECISION (dgbtrf の場合 ) COMPLEX (cgbtrf の場合 ) DOUBLE COMPLEX (zgbtrf の場合 )。 配列、次元は ( <i>ldab</i> , *)。 配列 <i>ab</i> には、行列 <i>A</i> が帯形式で格納される (「 <a href="#">行列の格納形式</a> 」を参照)。 <i>ab</i> の第 2 次元は $\max(1, n)$ 以上でなければならない。
<i>ldab</i>	INTEGER。配列 <i>ab</i> の第 1 次元 ( $ldab \geq 2kl + ku + 1$ )。

## 出力パラメータ

<i>ab</i>	<i>L</i> と <i>U</i> によって上書きされる。 <i>U</i> の対角成分と $kl + ku$ 個の優対角成分は、 <i>ab</i> の最初の $1 + kl + ku$ 行に格納される。 <i>L</i> の計算に使用される乗数は、次の <i>kl</i> 行に格納される。
<i>ipiv</i>	INTEGER。 配列、次元は $\max(1, \min(m, n))$ 以上。 ピボットのインデックス: 行 <i>i</i> は行 <i>ipiv</i> ( <i>i</i> ) と交換される。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了した。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正である。 <i>info</i> = <i>i</i> の場合、 $u_{ii}$ は 0 である。因子分解は完了したが <i>U</i> は完全に特異である。連立 1 次方程式の解の算出に係数 <i>U</i> を使用すると、0 による除算が発生する。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

gbtrf ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ab</i> を意味する。サイズ $(2*k1+ku+1, n)$ の配列 <i>A</i> を格納する。
<i>ipiv</i>	長さ $\min(m, n)$ のベクトルを格納する。
<i>k1</i>	省略した場合、 $k1 = ku$ とみなす。
<i>ku</i>	$ku = lda - 2*k1 - 1$ として復元する。
<i>m</i>	省略した場合、 $m = n$ とみなす。

### アプリケーション・ノート

算出された *L* と *U* は、摂動行列  $A + E$  の正確な係数になる。

$$|E| \leq c(k1 + ku + 1)\varepsilon P|L||U|$$

$c(k)$  は、 $k$  の適度な 1 次関数で、 $\varepsilon$  はマシンの精度である。

実数型の浮動小数点演算の合計回数は、約  $2n(ku+1)k1$  から  $2n(k1+ku+1)k1$  までの範囲である。複素数型の演算回数は、この 4 倍になる。これらの推定では、 $k1$  と  $ku$  は  $\min(m, n)$  よりはるかに小さいものとする。

$m = n$  でこのルーチン呼び出した後、以下のルーチン呼び出せる。

[?gbtrs](#)  $AX = B$  または  $A^T X = B$  または  $A^H X = B$  を解く。

[?gbcon](#) *A* の条件数を推定する。

---

## ?gttrf

三重対角行列の LU 因子分解を行う。

---

### 構文

#### Fortran 77:

```
call sgtrf(N, dl, d, du, du2, Ipiv, info)
call dgtrf(N, dl, d, du, du2, Ipiv, info)
call cgtrf(N, dl, d, du, du2, Ipiv, info)
call zgtrf(N, dl, d, du, du2, Ipiv, info)
```

#### Fortran 95:

```
call gttrf(dl, d, du, du2 [,ipiv] [,info])
```

## 説明

このルーチンは、次の形式で、実数または複素三重対角行列  $A$  の  $LU$  因子分解を実行する。

$$A = PLU,$$

$P$  は置換行列、 $L$  は単位対角成分を持つ下二重対角行列、 $U$  は主対角成分と最初の 2 つの優対角成分にのみ 0 でない値を持つ上三角行列である。このルーチンは、部分的なピボット演算で行を交換し、消去を実行する。

## 入力パラメータ

$n$  INTEGER。行列  $A$  の次数 ( $n \geq 0$ )。

$d1, d, du$  REAL (sgttrf の場合 )  
DOUBLE PRECISION (dgttrf の場合 )  
COMPLEX (cgttrf の場合 )  
DOUBLE COMPLEX (zgttrf の場合 )。  
 $A$  の成分を格納する配列。  
次元 ( $n - 1$ ) の配列  $d1$  は、 $A$  の劣対角成分を格納する。  
次元  $n$  の配列  $d$  は、 $A$  の対角成分を格納する。  
次元 ( $n - 1$ ) の配列  $du$  は、 $A$  の優対角成分を格納する。

## 出力パラメータ

$d1$   $A$  の  $LU$  因子分解で得られた行列  $L$  を定義する、( $n-1$ ) 個の乗数によって上書きされる。

$d$   $A$  の  $LU$  因子分解で得られた上三角行列  $U$  の  $n$  個の対角成分によって上書きされる。

$du$   $U$  の最初の優対角成分の ( $n-1$ ) 個の成分によって上書きされる。

$du2$  REAL (sgttrf の場合 )  
DOUBLE PRECISION (dgttrf の場合 )  
COMPLEX (cgttrf の場合 )  
DOUBLE COMPLEX (zgttrf の場合 )。  
配列、次元は ( $n-2$ )。終了時に、 $du2$  には  $U$  の 2 番目の優対角成分の ( $n-2$ ) 個の成分が格納される。

$ipiv$  INTEGER。  
配列、次元は ( $n$ )。  
ピボットのインデックス: 行  $i$  は行  $ipiv(i)$  と交換される。

*info* INTEGER。 *info* = 0 の場合、実行は正常に終了した。  
*info* = -*i* の場合、*i* 番目のパラメータの値が不正である。  
*info* = *i* の場合、 $u_{ii}$  は 0 である。因子分解は完了したが  $U$  は完全に特異である。連立 1 次方程式の解の算出に係数  $U$  を使用すると、0 による除算が発生する。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

gttrf ルーチンのインターフェイス特有の詳細を以下に示す。

*dl*                    長さ (n-1) のベクトルを格納する。  
*d*                     長さ (n) のベクトルを格納する。  
*du*                   長さ (n-1) のベクトルを格納する。  
*du2*                  長さ (n-2) のベクトルを格納する。  
*ipiv*                 長さ (n) のベクトルを格納する。

### アプリケーション・ノート

[?gbtrs](#)                 $AX=B$  または  $A^TX=B$  または  $A^HX=B$  を解く。

[?qbcon](#)                 $A$  の条件数を推定する。

---

## ?potrf

対称(エルミート)正定値行列のコレスキー  
 因子分解を行う。

---

### 構文

#### Fortran 77:

```
call spotrf(uplo, N, a, lda, info)
call dpotrf(uplo, N, a, lda, info)
call cpotrf(uplo, N, a, lda, info)
call zpotrf(uplo, N, a, lda, info)
```

## Fortran 95:

```
call potrf(a [,uplo] [,info])
```

## 説明

このルーチンは、対称 (複素データの場合はエルミート) 正定値行列  $A$  のコレスキー因子分解を実行する。

$$A = U^H U \quad (\text{uplo} = 'U' \text{ の場合})$$

$$A = L L^H \quad (\text{uplo} = 'L' \text{ の場合})$$

$L$  は下三角行列、 $U$  は上三角行列である。

## 入力パラメータ

**uplo** CHARACTER\*1。'U' または 'L' でなければならない。  
 $A$  の上三角部分と下三角部分のどちらを格納するか、また  $A$  をどのようにに因子分解するかを指定する。  
 $\text{uplo} = 'U'$  の場合、配列  $a$  には行列  $A$  の上三角部分が格納され、 $A$  は  $U^H U$  として因子分解される。  
 $\text{uplo} = 'L'$  の場合、配列  $a$  には行列  $A$  の下三角部分が格納され、 $A$  は  $L L^H$  として因子分解される。

**n** INTEGER。行列  $A$  の次数 ( $n \geq 0$ )。

**a** REAL (spotrf の場合)  
DOUBLE PRECISION (dpotrf の場合)  
COMPLEX (cpotrf の場合)  
DOUBLE COMPLEX (zpotrf の場合)。  
配列、次元は ( $lda, *$ )。  
配列  $a$  には、行列  $A$  の上三角部分または下三角部分を格納する ( $\text{uplo}$  を参照)。  
 $a$  の第 2 次元は、 $\max(1, n)$  以上でなければならない。

**lda** INTEGER。  $a$  の第 1 次元。

## 出力パラメータ

**a**  $\text{uplo}$  の指定に従って、 $a$  の上三角部分または下三角部分が、コレスキー係数  $U$  あるいは  $L$  によって上書きされる。

*info* INTEGER。 *info* = 0 の場合、実行は正常に終了した。  
*info* = -*i* の場合、*i* 番目のパラメータの値が不正である。  
*info* = *i* の場合、次数 *i* の先頭の小行列式 ( および行列 *A* そのもの ) が正定値でないため、因子分解を完了できない。これは、行列 *A* の構成に誤りがあることを示す。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

potrf ルーチンのインターフェイス特有の詳細を以下に示す。

*a* サイズ (*n*,*n*) の行列 *A* を格納する。

*uplo* 'U' または 'L' でなければならない。デフォルト値は 'U' である。

### アプリケーション・ノート

*uplo* = 'U' の場合、算出された係数 *U* は、摂動行列 *A* + *E* の正確な係数になる。*c*(*n*) は

$$|E| \leq c(n)\varepsilon \|U\| \|U\|, \quad |e_{ij}| \leq c(n)\varepsilon \sqrt{a_{ii}a_{jj}}$$

*n* の適度な 1 次関数で、 $\varepsilon$  はマシンの精度である。

*uplo* = 'L' の場合も、同様の推定が成り立つ。

浮動小数点演算の合計回数は、実数型の場合は約  $(1/3)n^3$ 、複素数型の場合は約  $(4/3)n^3$  になる。

このルーチンを呼び出した後、以下のルーチンを呼び出せる。

[?potrs](#) *AX* = *B* を解く。

[?pocon](#) *A* の条件数を推定する。

[?potri](#) *A* の逆行列を計算する。

## ?pptrf

圧縮格納形式による対称(エルミート)正定値行列のコレスキー因子分解を行う。

### 構文

#### Fortran 77:

```
call spptrf(uplo, N, ap, info)
call dpptrf(uplo, N, ap, info)
call cpptrf(uplo, N, ap, info)
call zpptrf(uplo, N, ap, info)
```

#### Fortran 95:

```
call pptrf(a [,uplo] [,info])
```

### 説明

このルーチンは、対称(複素データの場合はエルミート)正定値圧縮行列  $A$  のコレスキー因子分解を実行する。

$$A = U^H U \quad (\text{uplo} = 'U' \text{ の場合})$$

$$A = L L^H \quad (\text{uplo} = 'L' \text{ の場合})$$

$L$  は下三角行列、 $U$  は上三角行列である。

### 入力パラメータ

uplo	CHARACTER*1。'U' または 'L' でなければならない。 $A$ の上三角部分と下三角部分のどちらを配列 $ap$ にパックするか、また $A$ をどのように因子分解するかを指定する。 $uplo = 'U'$ の場合、配列 $ap$ には行列 $A$ の上三角部分が格納され、 $A$ は $U^H U$ として因子分解される。 $uplo = 'L'$ の場合、配列 $ap$ には行列 $A$ の下三角部分が格納され、 $A$ は $L L^H$ として因子分解される。
n	INTEGER。行列 $A$ の次数 ( $n \geq 0$ )。
ap	REAL (spptrf の場合) DOUBLE PRECISION (dpptrf の場合) COMPLEX (cpptrf の場合) DOUBLE COMPLEX (zpptrf の場合)。



配列、次元は  $\max(1, n(n+1)/2)$  以上。

配列 `ap` には、(`uplo` の指定に従って) 行列  $A$  の上三角部分または下三角部分を圧縮格納形式で格納する (「[行列の格納形式](#)」を参照)。

### 出力パラメータ

`ap` `uplo` の指定に従って、圧縮格納形式の  $A$  の上三角部分または下三角部分が、コレスキー係数  $U$  あるいは  $L$  によって上書きされる。

`info` INTEGER。 `info = 0` の場合、実行は正常に終了した。  
`info = -i` の場合、 $i$  番目のパラメータの値が不正である。  
`info = i` の場合、次数  $i$  の先頭の Minor 行列式 (および行列  $A$  そのもの) が正定値でないため、因子分解を完了できない。これは、行列  $A$  の構成に誤りがあることを示す。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

`pptrf` ルーチンのインターフェイス特有の詳細を以下に示す。

`a` Fortran 77 インターフェイスでの引数 `ap` を意味する。サイズ  $(n*(n+1)/2)$  の配列  $A$  を格納する。

`uplo` 'U' または 'L' でなければならない。デフォルト値は 'U' である。

### アプリケーション・ノート

`uplo = 'U'` の場合、算出された係数  $U$  は、摂動行列  $A + E$  の正確な係数になる。 $c(n)$  は

$$|E| \leq c(n)\varepsilon \|U\|, \quad |e_{ij}| \leq c(n)\varepsilon \sqrt{a_{ii}a_{jj}}$$

$n$  の適度な 1 次関数で、 $\varepsilon$  はマシンの精度である。

`uplo = 'L'` の場合も、同様の推定が成り立つ。

浮動小数点演算の合計回数は、実数型の場合は約  $(1/3)n^3$  で、複素数型の場合は約  $(4/3)n^3$  になる。

このルーチン呼び出し後、以下のルーチン呼び出せる。

[?pptrs](#)  $AX = B$  を解く。

[?ppcon](#)  $A$  の条件数を推定する。

[?pptri](#)  $A$  の逆行列を計算する。

---

## ?pbtrf

対称(エルミート) 正定値帯行列のコレスキー  
因子分解を行う。

---

### 構文

#### Fortran 77:

```
call spbtrf(uplo, N, kd, ab, ldab, info)
call dpbtrf(uplo, N, kd, ab, ldab, info)
call cpbtrf(uplo, N, kd, ab, ldab, info)
call zpbtrf(uplo, N, kd, ab, ldab, info)
```

#### Fortran 95:

```
call pbtrf(a [,uplo] [,info])
```

### 説明

このルーチンは、対称(複素データの場合はエルミート) 正定値帯行列  $A$  のコレスキー因子分解を実行する。

$$A = U^H U \quad (\text{uplo} = 'U' \text{ の場合})$$

$$A = L L^H \quad (\text{uplo} = 'L' \text{ の場合})$$

$L$  は下三角行列、 $U$  は上三角行列である。

### 入力パラメータ

$uplo$	CHARACTER*1。'U' または 'L' でなければならない。 $A$ の上三角部分と下三角部分のどちらを配列 $ab$ に格納するか、また $A$ をどのように因子分解するかを指定する。 $uplo = 'U'$ の場合、配列 $ab$ には行列 $A$ の上三角部分が格納され、 $A$ は $U^H U$ として因子分解される。 $uplo = 'L'$ の場合、配列 $ab$ には行列 $A$ の下三角部分が格納され、 $A$ は $L L^H$ として因子分解される。
$n$	INTEGER。行列 $A$ の次数 ( $n \geq 0$ )。
$kd$	INTEGER。 $A$ の優対角成分または劣対角成分の数 ( $kd \geq 0$ )。

*ab* REAL (spbtrf の場合 )  
 DOUBLE PRECISION (dpbtrf の場合 )  
 COMPLEX (cpbtrf の場合 )  
 DOUBLE COMPLEX (zpbtrf の場合 )。  
 配列、次元は (*ldab*, \*)。  
 配列 *ap* には、(*uplo* で指定された ) 行列 *A* の上三角部分または下三角部分  
 が帯形式で格納される (「[行列の格納形式](#)」を参照)。  
*ab* の第 2 次元は  $\max(1, n)$  以上でなければならない。

*ldab* INTEGER。配列 *ab* の第 1 次元 ( $ldab \geq kd + 1$ )。

### 出力パラメータ

*ap* *uplo* の指定に従って、帯格納形式の *A* の上三角部分または下三角部分  
 が、コレスキー係数 *U* あるいは *L* によって上書きされる。

*info* INTEGER。 *info* = 0 の場合、実行は正常に終了した。  
*info* = -*i* の場合、*i* 番目のパラメータの値が不正である。  
*info* = *i* の場合、次数 *i* の先頭の小行列式 ( および行列 *A* そのもの ) が  
 正定値でないため、因子分解を完了できない。これは、行列 *A* の構成  
 に誤りがあること示す。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの  
 引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関す  
 る詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

pbtrf ルーチンのインターフェイス特有の詳細を以下に示す。

*a* Fortran 77 インターフェイスでの引数 *ab* を意味する。サイズ ( $kd+1, n$ )  
 の配列 *A* を格納する。

*uplo* 'U' または 'L' でなければならない。デフォルト値は 'U' である。

### アプリケーション・ノート

*uplo* = 'U' の場合、算出された係数 *U* は、摂動行列  $A + E$  の正確な係数になる。 $c(n)$  は

$$|E| \leq c(kd+1)\varepsilon \|U^H\| \|U\|, \quad |e_{ij}| \leq c(kd+1)\varepsilon \sqrt{a_{ii}a_{jj}}$$

*n* の適度な 1 次関数で、 $\varepsilon$  はマシンの精度である。

*uplo* = 'L' の場合も、同様の推定が成り立つ。

実数型の場合は浮動小数点演算の合計回数は、約  $n(kd+1)^2$  になる。複素数型の演算回数は、この 4 倍になる。これらの推定では、 $kd$  は  $n$  よりはるかに小さいものとする。

このルーチン呼び出した後、以下のルーチン呼び出せる。

[?pbtrs](#)  $AX=B$  を解く。

[?pbcon](#)  $A$  の条件数を推定する。

---

## ?pttrf

対称 (エルミート) 正定値三重対角行列の  
因子分解を行う。

---

### 構文

#### Fortran 77:

```
call spttrf(N, d, e, info)
call dpttrf(N, d, e, info)
call cpttrf(N, d, e, info)
call zpttrf(N, d, e, info)
```

#### Fortran 95:

```
call pttrf(d, e [,info])
```

### 説明

このルーチンは、対称 (複素データの場合はエルミート) 正定値三重対角行列  $A$  の因子分解を実行する。

$A=LDL^H$  で、 $D$  は対角行列、 $L$  は単位下二重対角行列である。この因子分解の形式は、 $A=U^H D U$  とみなせる。 $D$  は単位上二重対角行列である。

### 入力パラメータ

$n$             INTEGER。行列  $A$  の次数 ( $n \geq 0$ )。  
 $d$             REAL (spttrf, cpttrf の場合)  
              DOUBLE PRECISION (dpttrf, zpttrf の場合)。  
              配列、次元は ( $n$ )。  $A$  の対角成分を格納する。

e REAL (spttrf の場合 )  
 DOUBLE PRECISION (dpttrf の場合 )  
 COMPLEX (cpttrf の場合 )  
 DOUBLE COMPLEX (zpttrf の場合 )。  
 配列、次元は  $(n-1)$ 。A の劣対角成分を格納する。

### 出力パラメータ

d A の  $LDL^H$  因子分解で得られた対角行列  $D$  の  $n$  個の対角成分によって上書きされる。

e A の因子分解で得られた単位二重対角係数  $L$  または  $U$  の  $(n-1)$  個の非対角成分によって上書きされる。

info INTEGER。info = 0 の場合、実行は正常に終了した。  
 info = -i の場合、i 番目のパラメータの値が不正である。  
 i = n の場合、因子分解は完了したが、 $d(n) = 0$  である。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

pttrf ルーチンのインターフェイス特有の詳細を以下に示す。

d 長さ  $(n)$  のベクトルを格納する。

e 長さ  $(n-1)$  のベクトルを格納する。

## ?sytrf

対称行列の Bunch-Kaufman 因子分解を行う。

### 構文

#### Fortran 77:

```
call ssytrf(uplo, N, a, lda, ipiv, work, lwork, info)
call dsytrf(uplo, N, a, lda, ipiv, work, lwork, info)
call csytrf(uplo, N, a, lda, ipiv, work, lwork, info)
call zsytrf(uplo, N, a, lda, ipiv, work, lwork, info)
```

## Fortran 95:

```
call sytrf(a [,uplo] [,ipiv] [,info])
```

## 説明

このルーチンは、対称行列の **Bunch-Kaufman** 因子分解を実行する。

$uplo='U'$  の場合、 $A = PUDU^TP^T$

$uplo='L'$  の場合、 $A = PLDL^TP^T$

$A$  は入力行列、 $P$  は置換行列、 $U$  と  $L$  は単位対角成分を含む上三角行列と下三角行列、 $D$  は  $1 \times 1$  と  $2 \times 2$  の対角ブロックを持つ対称ブロック対角行列である。 $U$  と  $L$  は、 $D$  の  $2 \times 2$  のブロックに対応する  $2 \times 2$  の単位対角ブロックを持っている。

## 入力パラメータ

**uplo** CHARACTER\*1。'U' または 'L' でなければならない。  
 $A$  の上三角部分と下三角部分のどちらを格納するか、また  $A$  をどのように因子分解するかを指定する。  
 $uplo='U'$  の場合、行列  $A$  の上三角部分を配列  $a$  に格納し、 $A$  を  $PUDU^TP^T$  として因子分解する。  
 $uplo='L'$  の場合、行列  $A$  の下三角部分を配列  $a$  に格納し、 $A$  を  $PLDL^TP^T$  として因子分解する。

**n** INTEGER。行列  $A$  の次数 ( $n \geq 0$ )。

**a** REAL (ssytrf の場合)  
 DOUBLE PRECISION (dsytrf の場合)  
 COMPLEX (csytrf の場合)  
 DOUBLE COMPLEX (zsytrf の場合)。  
 配列、次元は ( $lda, *$ )。  
 配列  $a$  には、行列  $A$  の上三角部分または下三角部分を格納する ( $uplo$  を参照)。  
 $a$  の第 2 次元は、 $\max(1, n)$  以上でなければならない。

**lda** INTEGER。  $a$  の第 1 次元。 $\max(1, n)$  以上でなければならない。

**work**  $a$  と同じタイプ。次元  $lwork$  のワークスペース配列。

**lwork** INTEGER。配列  $work$  のサイズ。 $lwork \geq n$ 。  
 $lwork = -1$  の場合はワークスペースのクエリとみなされ、ルーチンは  $work$  配列の最適サイズだけを計算し、その値を  $work$  配列の最初のエントリとして返し、 $xerbla$  は  $lwork$  に関するエラー・メッセージを生成しない。

$lwork$  の推奨値は、「[アプリケーション・ノート](#)」を参照。

## 出力パラメータ

<i>a</i>	<i>a</i> の上三角部分または下三角部分は、ブロック対角行列 <i>D</i> の各成分と、係数 <i>U</i> (または <i>L</i> ) の計算に使用された乗数によって上書きされる。
<i>work</i> (1)	<i>info</i> = 0 の場合、終了時に、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が <i>work</i> (1) に格納される。これ以降の実行には、この <i>lwork</i> の値を使用する。
<i>ipiv</i>	INTEGER。 配列、次元は $\max(1, n)$ 以上。 交換の結果と <i>D</i> のブロック構造の各成分が格納される。 <i>ipiv</i> ( <i>i</i> ) = <i>k</i> > 0 の場合、 <i>d</i> <sub><i>ii</i></sub> は 1 × 1 のブロックである。 <i>A</i> の <i>i</i> 番目の行と列は、 <i>k</i> 番目の行と列と交換される。 <i>uplo</i> = 'U' で <i>ipiv</i> ( <i>i</i> ) = <i>ipiv</i> ( <i>i</i> -1) = - <i>m</i> < 0 の場合、 <i>D</i> の <i>i</i> 行 ( <i>i</i> -1) 列に 2 × 2 のブロックが作成される。 <i>A</i> の ( <i>i</i> -1) 番目の行と列は、 <i>m</i> 番目の行と列と交換される。 <i>uplo</i> = 'L' で <i>ipiv</i> ( <i>i</i> ) = <i>ipiv</i> ( <i>i</i> +1) = - <i>m</i> < 0 の場合、 <i>D</i> の <i>i</i> 行 ( <i>i</i> +1) 列に 2 × 2 のブロックが作成される。 <i>A</i> の ( <i>i</i> +1) 番目の行と列は、 <i>m</i> 番目の行と列と交換される。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了した。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正である。 <i>info</i> = <i>i</i> の場合、 <i>d</i> <sub><i>ii</i></sub> は 0 である。因子分解は完了したが、 <i>D</i> は完全に特異で、ゼロである。連立 1 次方程式の解の算出に <i>D</i> を使用すると、0 による除算が発生する。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

sytrf ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	サイズ ( <i>n</i> , <i>n</i> ) の行列 <i>A</i> を格納する。
<i>ipiv</i>	長さ <i>n</i> のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U' である。

## アプリケーション・ノート

パフォーマンスを改善するには、*lwork* = *n* \* *blocksize* に設定する。*blocksize* は、ブロック・アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。

必要なワークスペースの大きさがわからない場合は、最初の実行では `lwork` を大きめの値に設定する。終了時に、`work(1)` の値を確認し、これ以降の実行にはその値を使用する。

$U$  と  $L$  の  $2 \times 2$  の単位対角ブロックと単位対角成分は格納されない。 $U$  と  $L$  のそれ以外の成分は、配列 `a` の対応する列に格納される。ただし、 $U$  または  $L$  を明示的に復元するには、さらに行の交換が必要である (その必要はほとんどない)。

すべての  $i = 1 \dots n$  について  $ipiv(i) = i$  が成り立つ場合は、 $U$  ( $L$ ) のすべての非対角成分は、配列 `a` の対応する成分に明示的に格納される。

`uplo = 'U'` の場合、算出された係数  $U$  と  $D$  は、摂動行列  $A + E$  の正確な係数になる。 $c(n)$  は  $n$  の適度な 1 次関数で、 $\varepsilon$  はマシンの精度である。

$$|E| \leq c(n) \varepsilon P|U||D||U^T|P^T$$

`uplo = 'L'` の場合も、算出された  $L$  と  $D$  について同様の推定が成り立つ。

浮動小数点演算の合計回数は、実数型の場合は約  $(1/3)n^3$ 、複素数型の場合は約  $(4/3)n^3$  になる。

このルーチンを呼び出した後、以下のルーチンを呼び出せる。

<a href="#">?sytrs</a>	$AX = B$ を解く。
<a href="#">?sycon</a>	$A$ の条件数を推定する。
<a href="#">?sytri</a>	$A$ の逆行列を計算する。

---

## ?hetrf

複素数型エルミート行列の *Bunch-Kaufman* 因子分解を行う。

---

### 構文

#### Fortran 77:

```
call chetrf(uplo, N, a, lda, ipiv, work, lwork, info)
call zhetrf(uplo, N, a, lda, ipiv, work, lwork, info)
```

#### Fortran 95:

```
call hetrf(a [,uplo] [,ipiv] [,info])
```



## 説明

このルーチンは、エルミート行列の **Bunch-Kaufman** 因子分解を実行する。

$uplo='U'$  の場合、 $A = PUDU^H P^T$

$uplo='L'$  の場合、 $A = PLDL^H P^T$

$A$  は入力行列、 $P$  は置換行列、 $U$  と  $L$  は単位対角成分を含む上三角行列と下三角行列、 $D$  は  $1 \times 1$  と  $2 \times 2$  の対角ブロックを持つエルミート・ブロック対角行列である。 $U$  と  $L$  は、 $D$  の  $2 \times 2$  のブロックに対応する  $2 \times 2$  の単位対角ブロックを持っている。

## 入力パラメータ

**uplo** CHARACTER\*1。'U' または 'L' でなければならない。  
 $A$  の上三角部分と下三角部分のどちらを格納するか、また  $A$  をどのように因子分解するかを指定する。  
 $uplo='U'$  の場合、行列  $A$  の上三角部分を配列  $a$  に格納し、 $A$  を  $PUDU^H P^T$  として因子分解する。  
 $uplo='L'$  の場合、行列  $A$  の下三角部分を配列  $a$  に格納し、 $A$  を  $PLDL^H P^T$  として因子分解する。

**n** INTEGER。行列  $A$  の次数 ( $n \geq 0$ )。

**a** COMPLEX (chetrf の場合)  
 DOUBLE COMPLEX (zhetrif の場合)。  
 配列、次元は ( $lda, *$ )。  
 配列  $a$  には、行列  $A$  の上三角部分または下三角部分を格納する ( $uplo$  を参照)。  
 $a$  の第 2 次元は、 $\max(1, n)$  以上でなければならない。

**lda** INTEGER。  $a$  の第 1 次元。  $\max(1, n)$  以上でなければならない。

**work**  $a$  と同じタイプ。次元  $lwork$  のワークスペース配列。

**lwork** INTEGER。配列  $work$  のサイズ。  $lwork \geq n$ 。  
 $lwork = -1$  の場合はワークスペースのクエリとみなされ、ルーチンは  $work$  配列の最適サイズだけを計算し、その値を  $work$  配列の最初のエントリとして返し、 $xerbla$  は  $lwork$  に関するエラー・メッセージを生成しない。  
 $lwork$  の推奨値は、「[アプリケーション・ノート](#)」を参照。

## 出力パラメータ

**a**  $a$  の上三角部分または下三角部分は、ブロック対角行列  $D$  の各成分と、係数  $U$  (または  $L$ ) の計算に使用された乗数によって上書きされる。

<i>work(1)</i>	<i>info</i> = 0 の場合、終了時に、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が <i>work(1)</i> に格納される。これ以降の実行には、この <i>lwork</i> の値を使用する。
<i>ipiv</i>	INTEGER。 配列、次元は $\max(1, n)$ 以上。 交換の結果と <i>D</i> のブロック構造の各成分が格納される。 <i>ipiv(i) = k &gt; 0</i> の場合、 <i>d<sub>ii</sub></i> は $1 \times 1$ のブロックである。 <i>A</i> の <i>i</i> 番目の行と列は、 <i>k</i> 番目の行と列と交換される。  <i>uplo</i> = 'U' で <i>ipiv(i) = ipiv(i-1) = -m &lt; 0</i> の場合、 <i>D</i> の <i>i</i> 行 ( <i>i-1</i> ) 列に $2 \times 2$ のブロックが作成される。 <i>A</i> の ( <i>i-1</i> ) 番目の行と列は、 <i>m</i> 番目の行と列と交換される。  <i>uplo</i> = 'L' で <i>ipiv(i) = ipiv(i+1) = -m &lt; 0</i> の場合、 <i>D</i> の <i>i</i> 行 ( <i>i+1</i> ) 列に $2 \times 2$ のブロックが作成される。 <i>A</i> の ( <i>i+1</i> ) 番目の行と列は、 <i>m</i> 番目の行と列と交換される。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了した。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正である。 <i>info</i> = <i>i</i> の場合、 <i>d<sub>ii</sub></i> は 0 である。因子分解は完了したが、 <i>D</i> は完全に特異で、ゼロである。連立 1 次方程式の解の算出に <i>D</i> を使用すると、0 による除算が発生する。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

hetrf ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	サイズ ( <i>n</i> , <i>n</i> ) の行列 <i>A</i> を格納する。
<i>ipiv</i>	長さ <i>n</i> のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U' である。

### アプリケーション・ノート

このルーチンは、正定値行列かどうかわからないエルミート行列に最適である。*A* が実際に正定値行列の場合、このルーチンは交換を実行せず、*D* 内に  $2 \times 2$  の対角ブロックは作成されない。

パフォーマンスを改善するには、 $lwork = n * blocksize$  に設定する。 $blocksize$  は、ブロック・アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。

必要なワークスペースの大きさがわからない場合は、最初の実行では  $lwork$  を大きめの値に設定する。終了時に、 $work(1)$  の値を確認し、これ以降の実行にはその値を使用する。

$U$  と  $L$  の  $2 \times 2$  の単位対角ブロックと単位対角成分は格納されない。 $U$  と  $L$  のそれ以外の成分は、配列  $a$  の対応する列に格納される。ただし、 $U$  または  $L$  を明示的に復元するには、さらに行の交換が必要である (その必要はほとんどない)。

すべての  $i = 1 \dots n$  について  $ipiv(i) = i$  が成り立つ場合は、 $U$  ( $L$ ) のすべての非対角成分は、配列  $a$  の対応する成分に明示的に格納される。

$uplo = 'U'$  の場合、算出された係数  $U$  と  $D$  は、摂動行列  $A + E$  の正確な係数になる。 $c(n)$  は  $n$  の適度な 1 次関数で、 $\varepsilon$  はマシンの精度である。

$$|E| \leq c(n) \varepsilon P |U| |D| |U^T| P^T$$

$uplo = 'L'$  の場合も、算出された  $L$  と  $D$  について同様の推定が成り立つ。

浮動小数点演算の合計回数は、約  $(4/3)n^3$  になる。

このルーチン呼び出した後、以下のルーチン呼び出せる。

[?hetrs](#)             $AX = B$  を解く。  
[?hecon](#)            $A$  の条件数を推定する。  
[?hetri](#)             $A$  の逆行列を計算する。

---

## ?sptf

圧縮格納形式による対称行列の *Bunch-Kaufman* 因子分解を行う。

---

### 構文

#### Fortran 77:

```
call ssptf(uplo, N, ap, ipiv, info)
call dsptf(uplo, N, ap, ipiv, info)
```

```
call csptftrf(uplo, N, ap, ipiv, info)
call zsptrf(uplo, N, ap, ipiv, info)
```

### Fortran 95:

```
call sptrf(a [,uplo] [,ipiv] [,info])
```

### 説明

このルーチンは、圧縮格納形式による対称行列  $A$  の Bunch-Kaufman 因子分解を実行する。

$uplo='U'$  の場合、 $A = PUDU^TP^T$

$uplo='L'$  の場合、 $A = PLDL^TP^T$

$P$  は置換行列、 $U$  と  $L$  は単位対角成分を含む上三角行列と下三角行列、 $D$  は  $1 \times 1$  と  $2 \times 2$  の対角ブロックを持つ対称ブロック対角行列である。 $U$  と  $L$  は、 $D$  の  $2 \times 2$  のブロックに対応する  $2 \times 2$  の単位対角ブロックを持っている。

### 入力パラメータ

**uplo** CHARACTER\*1。'U' または 'L' でなければならない。  
 $A$  の上三角部分と下三角部分のどちらを配列 **ap** にパックするか、また  $A$  をどのように因子分解するかを指定する。  
 $uplo='U'$  の場合、行列  $A$  の上三角部分を配列 **ap** に格納し、 $A$  を  $PUDU^TP^T$  として因子分解する。  
 $uplo='L'$  の場合、行列  $A$  の下三角部分を配列 **ap** に格納し、 $A$  を  $PLDL^TP^T$  として因子分解する。

**n** INTEGER。行列  $A$  の次数 ( $n \geq 0$ )。

**ap** REAL(ssptf の場合)  
DOUBLE PRECISION(dsptf の場合)  
COMPLEX(csptf の場合)  
DOUBLE COMPLEX(zsptrf の場合)。  
配列、次元は  $\max(1, n(n+1)/2)$  以上。  
配列 **ap** には、( $uplo$  の指定に従って) 行列  $A$  の上三角部分または下三角部分を圧縮格納形式で格納する (「[行列の格納形式](#)」を参照)。

## 出力パラメータ

<i>ap</i>	( <i>uplo</i> の指定に従って) $A$ の上三角部分または下三角部分は、ブロック対角行列 $D$ の各成分と、係数 $U$ (または $L$ ) の計算に使用された乗数によって上書きされる。
<i>ipiv</i>	INTEGER。 配列、次元は $\max(1, n)$ 以上。 交換の結果と $D$ のブロック構造の各成分が格納される。 $ipiv(i) = k > 0$ の場合、 $d_{ii}$ は $1 \times 1$ のブロックである。 $A$ の $i$ 番目の行と列は、 $k$ 番目の行と列と交換される。  $uplo = 'U'$ で $ipiv(i) = ipiv(i-1) = -m < 0$ の場合、 $D$ の $i$ 行 ( $i-1$ ) 列に $2 \times 2$ のブロックが作成される。 $A$ の ( $i-1$ ) 番目の行と列は、 $m$ 番目の行と列と交換される。  $uplo = 'L'$ で $ipiv(i) = ipiv(i+1) = -m < 0$ の場合、 $D$ の $i$ 行 ( $i+1$ ) 列に $2 \times 2$ のブロックが作成される。 $A$ の ( $i+1$ ) 番目の行と列は、 $m$ 番目の行と列と交換される。
<i>info</i>	INTEGER。 $info = 0$ の場合、実行は正常に終了した。 $info = -i$ の場合、 $i$ 番目のパラメータの値が不正である。 $info = i$ の場合、 $d_{ii}$ は 0 である。因子分解は完了したが、 $D$ は完全に特異で、ゼロである。連立 1 次方程式の解の算出に $D$ を使用すると、0 による除算が発生する。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

sptrf ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ap</i> を意味する。サイズ $(n*(n+1)/2)$ の配列 $A$ を格納する。
<i>ipiv</i>	長さ ( $n$ ) のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U' である。

## アプリケーション・ノート

$U$  と  $L$  の  $2 \times 2$  の単位対角ブロックと単位対角成分は格納されない。 $U$  と  $L$  のそれ以外の成分は、行列  $A$  の対応する列に格納される。ただし、 $U$  または  $L$  を明示的に復元するには、さらに行の交換が必要である (その必要はほとんどない)。

すべての  $i=1 \dots n$  で  $ipiv(i) = i$  が成り立つ場合は、 $U(L)$  のすべての非対角成分は、圧縮形式で明示的に格納される。

$uplo = 'U'$  の場合、算出された係数  $U$  と  $D$  は、摂動行列  $A + E$  の正確な係数になる。 $c(n)$  は  $n$  の適度な 1 次関数で、 $\varepsilon$  はマシンの精度である。

$$|E| \leq c(n) \varepsilon P |U| |D| |U^T| P^T$$

$uplo = 'L'$  の場合も、算出された  $L$  と  $D$  について同様の推定が成り立つ。

浮動小数点演算の合計回数は、実数型の場合は約  $(1/3) n^3$ 、複素数型の場合は約  $(4/3) n^3$  になる。

このルーチン呼び出した後、以下のルーチン呼び出せる。

<a href="#">?spttrs</a>	$AX = B$ を解く。
<a href="#">?spcon</a>	$A$ の条件数を推定する。
<a href="#">?sptri</a>	$A$ の逆行列を計算する。

---

## ?hptrf

圧縮格納形式による複素数型エルミート行列の  
*Bunch-Kaufman* 因子分解を行う。

---

### 構文

#### Fortran 77:

```
call chptrf(uplo, N, ap, ipiv, info)
call zhptrf(uplo, N, ap, ipiv, info)
```

#### Fortran 95:

```
call hptrf(a [,uplo] [,ipiv] [,info])
```

### 説明

このルーチンは、圧縮格納形式によるエルミート行列の **Bunch-Kaufman** 因子分解を実行する。

$uplo = 'U'$  の場合、 $A = PUDU^H P^T$

$uplo = 'L'$  の場合、 $A = PLDL^H P^T$

$A$  は入力行列、 $P$  は置換行列、 $U$  と  $L$  は単位対角成分を含む上三角行列と下三角行列、 $D$  は  $1 \times 1$  と  $2 \times 2$  の対角ブロックを持つエルミート・ブロック対角行列である。 $U$  と  $L$  は、 $D$  の  $2 \times 2$  のブロックに対応する  $2 \times 2$  の単位対角ブロックを持っている。

### 入力パラメータ

**uplo** CHARACTER\*1。'U' または 'L' でなければならない。  
 $A$  の上三角部分と下三角部分のどちらをパックするか、また  $A$  をどのように因子分解するかを指定する。  
**uplo** = 'U' の場合、行列  $A$  の上三角部分を配列 **ap** に格納し、 $A$  を  $PUDU^H P^T$  として因子分解する。  
**uplo** = 'L' の場合、行列  $A$  の下三角部分を配列 **ap** に格納し、 $A$  を  $PLDL^H P^T$  として因子分解する。

**n** INTEGER。行列  $A$  の次数 ( $n \geq 0$ )。

**ap** COMPLEX (chptrf の場合 )  
 DOUBLE COMPLEX (zhptrf の場合 )。  
 配列、次元は  $\max(1, n(n+1)/2)$  以上。  
 配列 **ap** には、(**uplo** の指定に従って) 行列  $A$  の上三角部分または下三角部分を圧縮格納形式で格納する (「[行列の格納形式](#)」を参照)。

### 出力パラメータ

**ap** (**uplo** の指定に従って)  $A$  の上三角部分または下三角部分は、ブロック対角行列  $D$  の各成分と、係数  $U$  (または  $L$ ) の計算に使用された乗数によって上書きされる。

**ipiv** INTEGER。  
 配列、次元は  $\max(1, n)$  以上。  
 交換の結果と  $D$  のブロック構造の各成分が格納される。  
**ipiv**( $i$ ) =  $k > 0$  の場合、 $d_{ii}$  は  $1 \times 1$  のブロックである。 $A$  の  $i$  番目の行と列は、 $k$  番目の行と列と交換される。  
**uplo** = 'U' で **ipiv**( $i$ ) = **ipiv**( $i-1$ ) =  $-m < 0$  の場合、 $D$  の  $i$  行 ( $i-1$ ) 列に  $2 \times 2$  のブロックが作成される。 $A$  の ( $i-1$ ) 番目の行と列は、 $m$  番目の行と列と交換される。  
**uplo** = 'L' で **ipiv**( $i$ ) = **ipiv**( $i+1$ ) =  $-m < 0$  の場合、 $D$  の  $i$  行 ( $i+1$ ) 列に  $2 \times 2$  のブロックが作成される。 $A$  の ( $i+1$ ) 番目の行と列は、 $m$  番目の行と列と交換される。

*info* INTEGER。 *info* = 0 の場合、実行は正常に終了した。  
*info* = -*i* の場合、*i* 番目のパラメータの値が不正である。  
*info* = *i* の場合、 $d_{ii}$  は 0 である。因子分解は完了したが、*D* は完全に特異で、ゼロである。連立 1 次方程式の解の算出に *D* を使用すると、0 による除算が発生する。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

hptrf ルーチンのインターフェイス特有の詳細を以下に示す。

*a* Fortran 77 インターフェイスでの引数 *ap* を意味する。サイズ  $(n*(n+1)/2)$  の配列 *A* を格納する。

*ipiv* 長さ (*n*) のベクトルを格納する。

*uplo* 'U' または 'L' でなければならない。デフォルト値は 'U' である。

### アプリケーション・ノート

*U* と *L* の  $2 \times 2$  の単位対角ブロックと単位対角成分は格納されない。*U* と *L* のそれ以外の成分は、配列 *a* の対応する列に格納される。ただし、*U* または *L* を明示的に復元するには、さらに行の交換が必要である (その必要はほとんどない)。

すべての  $i = 1 \dots n$  について  $ipiv(i) = i$  が成り立つ場合は、*U* (*L*) のすべての非対角成分は、配列 *a* の対応する成分に明示的に格納される。

*uplo* = 'U' の場合、算出された係数 *U* と *D* は、摂動行列  $A + E$  の正確な係数になる。 $c(n)$  は *n* の適度な 1 次関数で、 $\epsilon$  はマシンの精度である。

$$|E| \leq c(n)\epsilon P|U||D||U^T|P^T$$

*uplo* = 'L' の場合も、算出された *L* と *D* について同様の推定が成り立つ。

浮動小数点演算の合計回数は、約  $(4/3)n^3$  になる。

このルーチンを呼び出した後、以下のルーチンを呼び出せる。

[?hptrs](#) *AX = B* を解く。

[?hpcon](#) *A* の条件数を推定する。

[?hptri](#) *A* の逆行列を計算する。



## 連立 1 次方程式を解くためのルーチン

この節では、連立 1 次方程式を解くための LAPACK ルーチンについて説明する。通常は、これらのルーチンを呼び出す前に、連立方程式の行列を因子分解する必要がある (本章の「[行列の因子分解用のルーチン](#)」を参照)。ただし、解を求める連立方程式が三角行列を持つ場合は、因子分解の必要はない。

## ?getrs

LU 因子分解された正方行列を使って、複数の右辺を持つ連立 1 次方程式を解く。

### 構文

#### Fortran 77:

```
call sgetrs(trans, N, nrhs, a, lda, Ipiv, b, ldb, info)
call dgetrs(trans, N, nrhs, a, lda, Ipiv, b, ldb, info)
call cgetrs(trans, N, nrhs, a, lda, Ipiv, b, ldb, info)
call zgetrs(trans, N, nrhs, a, lda, Ipiv, b, ldb, info)
```

#### Fortran 95:

```
call getrs(a, ipiv, b [,trans] [,info])
```

### 説明

このルーチンは、以下の連立 1 次方程式を  $X$  について解く。

$AX = B$  (trans='N' の場合)

$A^T X = B$  (trans='T' の場合)

$A^H X = B$  (trans='C' の場合。ただし、複素行列のみ)

このルーチン呼び出す前に、[?getrf](#) を呼び出して、 $A$  の LU 因子分解を行う必要がある。

### 入力パラメータ

**trans** CHARACTER\*1。'N' または 'T' または 'C' でなければならない。  
方程式の形式を指定する。

	$trans = 'N'$ の場合、 $AX = B$ を $X$ について解く。
	$trans = 'T'$ の場合、 $A^T X = B$ を $X$ について解く。
	$trans = 'C'$ の場合、 $A^H X = B$ を $X$ について解く。
$n$	INTEGER。 $A$ の次数。 $B$ の行数 ( $n \geq 0$ )。
$nrhs$	INTEGER。 右辺の数 ( $nrhs \geq 0$ )。
$a, b$	REAL (sgetrs の場合 ) DOUBLE PRECISION (dgetrs の場合 ) COMPLEX (cgetrs の場合 ) DOUBLE COMPLEX (zgetrs の場合 )。 配列 : $a(lda, *)$ , $b(ldb, *)$ 。  配列 $a$ には、行列 $A$ が格納される。 配列 $b$ には、行列 $B$ が格納される。この行列の列は、連立方程式の右辺である。  $a$ の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $b$ の第 2 次元は、 $\max(1, nrhs)$ 以上でなければならない。
$lda$	INTEGER。 $a$ の第 1 次元。 $lda \geq \max(1, n)$ 。
$ldb$	INTEGER。 $b$ の第 1 次元。 $ldb \geq \max(1, n)$ 。
$ipiv$	INTEGER。 配列、次元は $\max(1, n)$ 以上。 <a href="#">?getrf</a> によって返される $ipiv$ 配列。

### 出力パラメータ

$b$	解の行列 $X$ によって上書きされる。
$info$	INTEGER。 $info = 0$ の場合、実行は正常に終了した。 $info = -i$ の場合、 $i$ 番目のパラメータの値が不正である。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

getrs ルーチンのインターフェイス特有の詳細を以下に示す。

$a$	サイズ $(n, n)$ の行列 $A$ を格納する。
$b$	サイズ $(n, nrhs)$ の行列 $B$ を格納する。

*ipiv*           長さ ( $n$ ) のベクトルを格納する。

*trans*           'N'、'C'、または 'T' でなければならない。デフォルト値は 'N' である。

### アプリケーション・ノート

各右辺  $b$  について算出された解は、摂動連立方程式  $(A + E)x = b$  の正確な解になる。  
 $|E| \leq c(n)\varepsilon P|L||U|$

$c(n)$  は  $n$  の適度な 1 次関数で、 $\varepsilon$  はマシンの精度である。

$x_0$  が真の解である場合、算出された解  $x$  は、次の誤差範囲を満たす。

$$\frac{\|x - x_0\|_\infty}{\|x\|_\infty} \leq c(n) \operatorname{cond}(A, x) \varepsilon$$

ただし、 $\operatorname{cond}(A, x) = \| |A^{-1}| |A| |x| \|_\infty / \|x\|_\infty \leq \|A^{-1}\|_\infty \|A\|_\infty = \kappa_\infty(A)$ 。

$\operatorname{cond}(A, x)$  は、 $\kappa_\infty(A)$  よりはるかに小さくなる場合がある。 $A^T$  と  $A^H$  の条件数は、 $\kappa_\infty(A)$  に等しいことも、等しくないこともある。

1 つの右辺ベクトル  $b$  の浮動小数点演算のおおよその回数は、実数型の場合は  $2n^2$ 、複素数型の場合は  $8n^2$  になる。

条件数  $\kappa_\infty(A)$  を推定するには、[?gecon](#) を呼び出す。

解の精度を改善して誤差を推定するには、[?gerfs](#) を呼び出す。

---

## ?gbtrs

LU 因子分解された帯行列を使って、複数の右辺を持つ連立 1 次方程式を解く。

---

### 構文

#### Fortran 77:

```
call sgbtrs(trans, n, kl, ku, nrhs, ab, ldab, ipiv, b, ldb, info)
call dgbtrs(trans, n, kl, ku, nrhs, ab, ldab, ipiv, b, ldb, info)
call cgbtrs(trans, n, kl, ku, nrhs, ab, ldab, ipiv, b, ldb, info)
```

```
call zgbtrs(trans, n, kl, ku, nrhs, ab, ldab, ipiv, b, ldb, info)
```

### Fortran 95:

```
call gbtrs(a, b, ipiv, [,kl] [,trans] [,info])
```

### 説明

このルーチンは、以下の連立 1 次方程式を  $X$  について解く。

$AX = B$  (trans='N' の場合)

$A^T X = B$  (trans='T' の場合)

$A^H X = B$  (trans='C' の場合。ただし、複素行列のみ)

$A$  は、 $kl$  個の非ゼロの劣対角成分と  $ku$  個の非ゼロの優対角成分を持つ、 $LU$  因子分解された  $n$  次の一般帯行列である。このルーチンを呼び出す前に、[?gbtrf](#) を呼び出して、 $A$  の  $LU$  因子分解を行う必要がある。

### 入力パラメータ

**trans** CHARACTER\*1. 'N' または 'T' または 'C' でなければならない。

**n** INTEGER.  $A$  の次数。  $B$  の行数 ( $n \geq 0$ )。

**kl** INTEGER.  $A$  の帯内の劣対角成分の数 ( $kl \geq 0$ )。

**ku** INTEGER.  $A$  の帯内の優対角成分の数 ( $ku \geq 0$ )。

**nrhs** INTEGER. 右辺の数 ( $nrhs \geq 0$ )。

**ab, b** REAL (sgbtrs の場合)  
DOUBLE PRECISION (dgbtrs の場合)  
COMPLEX (cgbtrs の場合)  
DOUBLE COMPLEX (zgbtrs の場合)。  
配列:  $ab(ldab, *)$ ,  $b(ldb, *)$ 。

配列  $ab$  には、行列  $A$  が帯形式で格納される (「[行列の格納形式](#)」を参照)。

配列  $b$  には、行列  $B$  が格納される。この行列の列は、連立方程式の右辺である。

$ab$  の第 2 次元は  $\max(1, n)$  以上でなければならない。

$b$  の第 2 次元は  $\max(1, nrhs)$  以上でなければならない。

**ldab** INTEGER. 配列  $ab$  の第 1 次元 ( $ldab \geq 2kl + ku + 1$ )。

**ldb** INTEGER.  $b$  の第 1 次元。  $ldb \geq \max(1, n)$ 。

*ipiv* INTEGER。配列、次元は  $\max(1, n)$  以上。  
[?gbtrf](#) によって返される *ipiv* 配列。

### 出力パラメータ

*b* 解の行列  $X$  によって上書きされる。

*info* INTEGER。 *info* = 0 の場合、実行は正常に終了した。  
*info* = -*i* の場合、*i* 番目のパラメータの値が不正である。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

gbtrs ルーチンのインターフェイス特有の詳細を以下に示す。

*a* Fortran 77 インターフェイスでの引数 *ab* を意味する。サイズ  $(2*k_l+ku+1, n)$  の配列  $A$  を格納する。

*b* サイズ  $(n, nrhs)$  の行列  $B$  を格納する。

*ipiv* 長さ  $\min(m, n)$  のベクトルを格納する。

*kl* 省略した場合、 $kl = ku$  とみなす。

*ku*  $lda-2*kl-1$  として復元する。

*trans* 'N'、'C'、または 'T' でなければならない。デフォルト値は 'N' である。

### アプリケーション・ノート

各右辺  $b$  について算出された解は、摂動連立方程式  $(A + E)x = b$  の正確な解になる。  
 $|E| \leq c(kl + ku + 1)\varepsilon P|L||U|$

$c(k)$  は、 $k$  の適度な 1 次関数で、 $\varepsilon$  はマシンの精度である。

$x_0$  が真の解である場合、算出された解  $x$  は、次の誤差範囲を満たす。

$$\frac{\|x - x_0\|_\infty}{\|x\|_\infty} \leq c(kl + ku + 1) \text{cond}(A, x)\varepsilon$$

ただし、 $\text{cond}(A, x) = \| |A^{-1}| |A| |x| \|_\infty / \|x\|_\infty \leq \|A^{-1}\|_\infty \|A\|_\infty = \kappa_\infty(A)$ 。

$\text{cond}(A, x)$  は、 $\kappa_{\infty}(A)$  よりはるかに小さくなる場合がある。 $A^T$  と  $A^H$  の条件数は、 $\kappa_{\infty}(A)$  に等しいことも、等しくないこともある。

1 つの右辺ベクトルの浮動小数点演算のおおよその回数は、実数型の場合は  $2n(ku + 2kl)$  になる。複素数型の演算回数は、この 4 倍になる。これらの推定では、 $kl$  と  $ku$  は  $\min(m, n)$  よりはるかに小さいものとする。

条件数  $\kappa_{\infty}(A)$  を推定するには、[?qbccon](#) を呼び出す。  
解の精度を改善して誤差を推定するには、[?qbrfs](#) を呼び出す。

---

## ?gttrs

?gttrf によって行われた LU 因子分解を使用して、三重対角行列を係数行列とする連立 1 次方程式を解く。

---

### 構文

#### Fortran 77:

```
call sgtrrs(trans, n, nrhs, dl, d, du, du2, ipiv, b, ldb, info)
call dgtrrs(trans, n, nrhs, dl, d, du, du2, ipiv, b, ldb, info)
call cgtrrs(trans, n, nrhs, dl, d, du, du2, ipiv, b, ldb, info)
call zgtrrs(trans, n, nrhs, dl, d, du, du2, ipiv, b, ldb, info)
```

#### Fortran 95:

```
call gttrs(dl, d, du, du2, b, ipiv [,trans] [,info])
```

### 説明

このルーチンは、複数の右辺を持つ以下の連立 1 次方程式を、 $X$  について解く。

$AX = B$  (trans='N' の場合)

$A^T X = B$  (trans='T' の場合)

$A^H X = B$  (trans='C' の場合。ただし、複素行列のみ)

このルーチンを呼び出す前に、[?gttrf](#) を呼び出して、 $A$  の LU 因子分解を行う必要がある。

## 入力パラメータ

<i>trans</i>	CHARACTER*1。'N' または 'T' または 'C' でなければならない。 方程式の形式を指定する。  <i>trans</i> = 'N' の場合、 $AX=B$ を $X$ について解く。 <i>trans</i> = 'T' の場合、 $A^TX=B$ を $X$ について解く。 <i>trans</i> = 'C' の場合、 $A^HX=B$ を $X$ について解く。
<i>n</i>	INTEGER。 $A$ の次数 ( $n \geq 0$ )。
<i>nrhs</i>	INTEGER。 右辺の数、すなわち、 $B$ の列数 ( $nrhs \geq 0$ )。
<i>d1, d, du, du2, b</i>	REAL (sgttrs の場合 ) DOUBLE PRECISION (dgttrs の場合 ) COMPLEX (cgttrs の場合 ) DOUBLE COMPLEX (zgttrf の場合 )。 配列 $d1(n-1), d(n), du(n-1), du2(n-2), b(ldb, nrhs)$ 。 配列 $d1$ には、 $A$ の $LU$ 因子分解で得られた行列 $L$ を定義する ( $n-1$ ) 個の乗数が格納される。 配列 $d$ には、 $A$ の $LU$ 因子分解で得られた上三角行列 $U$ の $n$ 個の対角成分が格納される。 配列 $du$ には、 $U$ の最初の優対角成分の ( $n-1$ ) 個の成分が格納される。 配列 $du2$ には、 $U$ の 2 番目の優対角成分の ( $n-2$ ) 個の成分が格納される。 配列 $b$ には、 行列 $B$ が格納される。この行列の列は、連立方程式の右辺である。
<i>ldb</i>	INTEGER。 $b$ のリーディング・ディメンジョン。 $ldb \geq \max(1, n)$ 。
<i>ipiv</i>	INTEGER。 配列、次元は ( $n$ )。 <a href="#">?gttrf</a> によって返される <i>ipiv</i> 配列。

## 出力パラメータ

<i>b</i>	解の行列 $X$ によって上書きされる。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了した。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正である。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

gttrs ルーチンのインターフェイス特有の詳細を以下に示す。

<i>d1</i>	長さ ( <i>n-1</i> ) のベクトルを格納する。
<i>d</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>du</i>	長さ ( <i>n-1</i> ) のベクトルを格納する。
<i>du2</i>	長さ ( <i>n-2</i> ) のベクトルを格納する。
<i>b</i>	サイズ ( <i>n,nrhs</i> ) の行列 <i>B</i> を格納する。
<i>ipiv</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>trans</i>	'N'、'C'、または 'T' でなければならない。デフォルト値は 'N' である。

## アプリケーション・ノート

各右辺 *b* について算出された解は、摂動連立方程式  $(A + E)x = b$  の正確な解になる。

$$|E| \leq c(n)\varepsilon P|L||U|$$

$c(n)$  は  $n$  の適度な 1 次関数で、 $\varepsilon$  はマシンの精度である。

$x_0$  が真の解である場合、算出された解  $x$  は、次の誤差範囲を満たす。

$$\frac{\|x - x_0\|_\infty}{\|x\|_\infty} \leq c(n) \text{ cond } (A, x) \varepsilon$$

ただし、 $\text{cond}(A, x) = \| |A^{-1}| |A| |x| \|_\infty / \|x\|_\infty \leq \|A^{-1}\|_\infty \|A\|_\infty = \kappa_\infty(A)$ 。

$\text{cond}(A, x)$  は、 $\kappa_\infty(A)$  よりはるかに小さくなる場合がある。 $A^T$  と  $A^H$  の条件数は、 $\kappa_\infty(A)$  に等しいことも、等しくないこともある。

1 つの右辺ベクトル *b* の浮動小数点演算のおおよその回数は、実数型の場合は  $2n^2$ 、複素数型の場合は  $8n^2$  になる。

条件数  $\kappa_\infty(A)$  を推定するには、[?gecon](#) を呼び出す。

解の精度を改善して誤差を推定するには、[?gerfs](#) を呼び出す。



## ?potrs

コレスキー因子分解された対称 (エルミート) 正定値行列を使って、連立 1 次方程式を解く。

### 構文

#### Fortran 77:

```
call spotrs(uplo, N, nrhs, a, lda, b, ldb, info)
call dpotrs(uplo, N, nrhs, a, lda, b, ldb, info)
call cpotrs(uplo, N, nrhs, a, lda, b, ldb, info)
call zpotrs(uplo, N, nrhs, a, lda, b, ldb, info)
```

#### Fortran 95:

```
call potrs(a, b [,uplo] [,info])
```

### 説明

このルーチンは、対称 (複素データの場合はエルミート) 正定値行列  $A$  を持つ連立 1 次方程式  $AX=B$  を、 $A$  のコレスキー因子分解に基づいて、 $X$  について解く。

$$A = U^H U \quad (\text{uplo} = 'U' \text{ の場合})$$

$$A = LL^H \quad (\text{uplo} = 'L' \text{ の場合})$$

$L$  は下三角行列、 $U$  は上三角行列である。行列  $B$  の各列に格納されている複数の右辺を使用して、この連立方程式を解く。

このルーチン呼び出す前に、[?potrf](#) を呼び出して、 $A$  のコレスキー因子分解を行う必要がある。

### 入力パラメータ

`uplo` CHARACTER\*1。'U' または 'L' でなければならない。

入力行列  $A$  の因子分解の方法を指定する。

`uplo` = 'U' の場合、配列 `a` には、コレスキー因子分解  $A = U^H U$  の係数  $U$  を格納する。

`uplo` = 'L' の場合、配列 `a` には、コレスキー因子分解  $A = LL^H$  の係数  $L$  を格納する。

<i>n</i>	INTEGER。行列 <i>A</i> の次数 ( $n \geq 0$ )。
<i>nrhs</i>	INTEGER。右辺の数 ( $nrhs \geq 0$ )。
<i>a</i> , <i>b</i>	REAL (spotrs の場合 ) DOUBLE PRECISION (dpotrs の場合 ) COMPLEX (cpotrs の場合 ) DOUBLE COMPLEX (zpotrs の場合 )。 配列 : <i>a</i> ( <i>lda</i> , *), <i>b</i> ( <i>ldb</i> , *)。 配列 <i>a</i> には、係数 <i>U</i> または <i>L</i> を格納する ( <i>uplo</i> を参照 )。 配列 <i>b</i> には、行列 <i>B</i> が格納される。この行列の列は、連立方程式の右辺である。  <i>a</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>b</i> の第 2 次元は、 $\max(1, nrhs)$ 以上でなければならない。
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $lda \geq \max(1, n)$ 。
<i>ldb</i>	INTEGER。 <i>b</i> の第 1 次元。 $ldb \geq \max(1, n)$ 。

### 出力パラメータ

<i>b</i>	解の行列 <i>X</i> によって上書きされる。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了した。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正である。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

potrs ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	サイズ ( <i>n</i> , <i>n</i> ) の行列 <i>A</i> を格納する。
<i>b</i>	サイズ ( <i>n</i> , <i>nrhs</i> ) の行列 <i>B</i> を格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U' である。

### アプリケーション・ノート

*uplo* = 'U' の場合、各右辺 *b* について算出された解は、摂動連立方程式  $(A + E)x = b$  の正確な解になる。*c*(*n*) は *n* の適度な 1 次関数で、 $\varepsilon$  はマシンの精度である。

$$|E| \leq c(n) \varepsilon |U^H| |U|$$

`uplo = 'L'` の場合も、同様の推定が成り立つ。  
 $x_0$  が真の解である場合、算出された解  $x$  は、次の誤差範囲を満たす。

$$\frac{\|x - x_0\|_\infty}{\|x\|_\infty} \leq c(n) \operatorname{cond}(A, x) \varepsilon$$

ただし、 $\operatorname{cond}(A, x) = \| |A^{-1}| |A| |x| \|_\infty / \|x\|_\infty \leq \|A^{-1}\|_\infty \|A\|_\infty = \kappa_\infty(A)$ 。

$\operatorname{cond}(A, x)$  は、 $\kappa_\infty(A)$  よりはるかに小さくなる場合がある。  
 1 つの右辺ベクトル  $b$  の浮動小数点演算のおおよその回数は、実数型の場合は  $2n^2$ 、複素数型の場合は  $8n^2$  になる。

条件数  $\kappa_\infty(A)$  を推定するには、[?pocon](#) を呼び出す。  
 解の精度を改善して誤差を推定するには、[?porfs](#) を呼び出す。

## ?pptrs

コレスキー因子分解された圧縮形式の対称(エルミート)正定値行列を使って、連立1次方程式を解く。

### 構文

#### Fortran 77:

```
call spptrs(uplo, N, nrhs, ap, b, ldb, info)
call dpptrs(uplo, N, nrhs, ap, b, ldb, info)
call cpptrs(uplo, N, nrhs, ap, b, ldb, info)
call zpptrs(uplo, N, nrhs, ap, b, ldb, info)
```

#### Fortran 95:

```
call pptrs(a, b [,uplo] [,info])
```

## 説明

このルーチンは、圧縮形式の対称 (複素データの場合はエルミート) 正定値行列  $A$  を持つ連立 1 次方程式  $AX = B$  を、 $A$  のコレスキー因子分解に基づいて、 $X$  について解く。

$$A = U^H U \quad (\text{uplo} = 'U' \text{ の場合})$$

$$A = LL^H \quad (\text{uplo} = 'L' \text{ の場合})$$

$L$  は下三角行列、 $U$  は上三角行列である。行列  $B$  の各列に格納されている複数の右辺を使用して、この連立方程式を解く。

このルーチンを呼び出す前に、[?pptrf](#) を呼び出して、 $A$  のコレスキー因子分解を行う必要がある。

## 入力パラメータ

<code>uplo</code>	CHARACTER*1。'U' または 'L' でなければならない。 入力行列 $A$ の因子分解の方法を指定する。  <code>uplo = 'U'</code> の場合、配列 $a$ には、コレスキー因子分解 $A = U^H U$ の係数 $U$ を圧縮形式で格納する。 <code>uplo = 'L'</code> の場合、配列 $a$ には、コレスキー因子分解 $A = LL^H$ の係数 $L$ を圧縮形式で格納する。
<code>n</code>	INTEGER。行列 $A$ の次数 ( $n \geq 0$ )。
<code>nrhs</code>	INTEGER。右辺の数 ( $nrhs \geq 0$ )。
<code>ap, b</code>	REAL (spptrs の場合) DOUBLE PRECISION (dpptrs の場合) COMPLEX (cpptrs の場合) DOUBLE COMPLEX (zpptrs の場合)。 配列、 $ap(*)$ , $b(ldb, *)$ $ap$ の次元は $\max(1, n(n+1)/2)$ 以上でなければならない。 配列 $ap$ には、 <code>uplo</code> の指定に従って、係数 $U$ または $L$ が圧縮格納形式で格納される (「 <a href="#">行列の格納形式</a> 」を参照)。  配列 $b$ には、行列 $B$ が格納される。この行列の列は、連立方程式の右辺である。 $b$ の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。
<code>ldb</code>	INTEGER。 $b$ の第 1 次元。 $ldb \geq \max(1, n)$ 。

### 出力パラメータ

- b* 解の行列  $X$  によって上書きされる。
- info* INTEGER。 *info* = 0 の場合、実行は正常に終了した。  
*info* = -*i* の場合、*i* 番目のパラメータの値が不正である。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

pptrs ルーチンのインターフェイス特有の詳細を以下に示す。

- a* Fortran 77 インターフェイスでの引数 *ap* を意味する。サイズ  $(n*(n+1)/2)$  の配列  $A$  を格納する。
- b* サイズ  $(n, nrhs)$  の行列  $B$  を格納する。
- uplo* 'U' または 'L' でなければならない。デフォルト値は 'U' である。

### アプリケーション・ノート

*uplo* = 'U' の場合、各右辺  $b$  について算出された解は、摂動連立方程式  $(A + E)x = b$  の正確な解になる。

$$|E| \leq c(n)\varepsilon |U^H||U|$$

$c(n)$  は  $n$  の適度な 1 次関数で、 $\varepsilon$  はマシンの精度である。

*uplo* = 'L' の場合も、同様の推定が成り立つ。

$x_0$  が真の解である場合、算出された解  $x$  は、次の誤差範囲を満たす。

$$\frac{\|x - x_0\|_\infty}{\|x\|_\infty} \leq c(n) \operatorname{cond}(A, x) \varepsilon$$

ただし、 $\operatorname{cond}(A, x) = \| |A^{-1}| |A| |x| \|_\infty / \|x\|_\infty \leq \|A^{-1}\|_\infty \|A\|_\infty = \kappa_\infty(A)$ 。

$\operatorname{cond}(A, x)$  は、 $\kappa_\infty(A)$  よりはるかに小さくなる場合がある。

1 つの右辺ベクトル  $b$  の浮動小数点演算のおおよその回数は、実数型の場合は  $2n^2$ 、複素数型の場合は  $8n^2$  になる。

条件数  $\kappa_{\infty}(A)$  を推定するには、[?ppcon](#) を呼び出す。

解の精度を改善して誤差を推定するには、[?pprfs](#) を呼び出す。

---

### ?pbtrs

コレスキー因子分解された対称 (エルミート)  
正定値帯行列を使って、連立 1 次方程式を解く。

---

#### 構文

##### Fortran 77:

```
call spbtrs(uplo, n, kd, nrhs, ab, ldab, b, ldb, info)
call dpbtrs(uplo, n, kd, nrhs, ab, ldab, b, ldb, info)
call cpbtrs(uplo, n, kd, nrhs, ab, ldab, b, ldb, info)
call zpbtrs(uplo, n, kd, nrhs, ab, ldab, b, ldb, info)
```

##### Fortran 95:

```
call pbtrs(a, b [,uplo] [,info])
```

#### 説明

このルーチンは、対称 (複素データの場合はエルミート) 正定値**帯行列**  $A$  を持つ連立 1 次方程式  $AX=B$  を、 $A$  のコレスキー因子分解に基づいて、 $X$  について解く。

$$A = U^H U \quad (\text{uplo='U' の場合})$$

$$A = LL^H \quad (\text{uplo='L' の場合})$$

$L$  は下三角行列、 $U$  は上三角行列である。行列  $B$  の各列に格納されている複数の右辺を使用して、この連立方程式を解く。

このルーチンを呼び出す前に、[?pbturf](#) を呼び出して、 $A$  のコレスキー因子分解を帯格納形式で計算する必要がある。

#### 入力パラメータ

**uplo** CHARACTER\*1。'U' または 'L' でなければならない。

入力行列  $A$  の因子分解の方法を指定する。

$uplo = 'U'$  の場合、配列  $a$  には、因子分解  $A = U^H U$  の係数  $U$  を帯格納形式で格納する。

$uplo = 'L'$  の場合、配列  $a$  には、因子分解  $A = L L^H$  の係数  $L$  を帯格納形式で格納する。

$n$  INTEGER。行列  $A$  の次数 ( $n \geq 0$ )。

$kd$  INTEGER。  $A$  の優対角成分または劣対角成分の数 ( $kd \geq 0$ )。

$nrhs$  INTEGER。 右辺の数 ( $nrhs \geq 0$ )。

$ab, b$  REAL (spbtrs の場合 )  
DOUBLE PRECISION (dpbtrs の場合 )  
COMPLEX (cpbtrs の場合 )  
DOUBLE COMPLEX (zpbtrs の場合 )。  
配列 :  $ab(ldab, *)$ ,  $b(l db, *)$ 。

配列  $ab$  には、因子分解ルーチンによって返されるコレスキー係数を帯格納形式で格納する。

配列  $b$  には、行列  $B$  が格納される。この行列の列は、連立方程式の右辺である。

$ab$  の第 2 次元は  $\max(1, n)$  以上でなければならない。

$b$  の第 2 次元は  $\max(1, nrhs)$  以上でなければならない。

$ldab$  INTEGER。 配列  $ab$  の第 1 次元 ( $ldab \geq kd + 1$ )。

$ldb$  INTEGER。  $b$  の第 1 次元。  $ldb \geq \max(1, n)$ 。

## 出力パラメータ

$b$  解の行列  $X$  によって上書きされる。

$info$  INTEGER。  $info = 0$  の場合、実行は正常に終了した。  
 $info = -i$  の場合、 $i$  番目のパラメータの値が不正である。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

pbtrs ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ab</i> を意味する。サイズ $(kd+1, n)$ の配列 <i>A</i> を格納する。
<i>b</i>	サイズ $(n, nrhs)$ の行列 <i>B</i> を格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U' である。

## アプリケーション・ノート

各右辺 *b* について算出された解は、摂動連立方程式  $(A + E)x = b$  の正確な解になる。

$$|E| \leq c(kd+1)\varepsilon P|U^H||U| \quad \text{および} \quad |E| \leq c(kd+1)\varepsilon P|L^H||L|$$

$c(k)$  は、 $k$  の適度な 1 次関数で、 $\varepsilon$  はマシンの精度である。

$x_0$  が真の解である場合、算出された解  $x$  は、次の誤差範囲を満たす。

$$\frac{\|x - x_0\|_\infty}{\|x\|_\infty} \leq c(kd+1) \text{cond}(A, x)\varepsilon$$

ただし、 $\text{cond}(A, x) = \|A^{-1}\|_\infty \|A\|_\infty / \|x\|_\infty \leq \|A^{-1}\|_\infty \|A\|_\infty = \kappa_\infty(A)$ 。

$\text{cond}(A, x)$  は、 $\kappa_\infty(A)$  よりはるかに小さくなる場合がある。

1 つの右辺ベクトルの浮動小数点演算のおおよその回数は、実数型の場合は  $4n*kd$ 、複素数型の場合は  $16n*kd$  になる。

条件数  $\kappa_\infty(A)$  を推定するには、[?pbcon](#) を呼び出す。

解の精度を改善して誤差を推定するには、[?pbrfs](#) を呼び出す。



## ?pttrs

?pttrf によって行われた因子分解を使用して、  
対称 (エルミート) 正定値三重対角行列を係数行列とする連立 1 次方程式を解く。

### 構文

#### Fortran 77:

```
call sppttrs(n, nrhs, d, e, b, ldb, info)
call dppttrs(n, nrhs, d, e, b, ldb, info)
call cppttrs(uplo, n, nrhs, d, e, b, ldb, info)
call zppttrs(uplo, n, nrhs, d, e, b, ldb, info)
```

#### Fortran 95:

```
call pttrs(d, e, b [,info])
call pttrs(d, e, b [,uplo] [,info])
```

### 説明

このルーチンは、対称 (エルミート) 正定値三重対角行列  $A$  を係数行列とする連立 1 次方程式  $AX=B$  を、 $X$  について解く。

このルーチンを呼び出す前に、[?pttrf](#) を呼び出して、 $A$  の  $LDL^H$  または  $U^H DU$  因子分解を行う必要がある。

### 入力パラメータ

<code>uplo</code>	CHARACTER*1。cppttrs/zppttrs でのみ使用される。 'U' または 'L' でなければならない。 三重対角行列 $A$ の優対角成分と劣対角成分のどちらを格納するかと、 $A$ の因子分解の方法を指定する。 <code>uplo = 'U'</code> の場合、配列 $e$ には $A$ の優対角成分が格納され、 $A$ は $U^H DU$ として因子分解される。 <code>uplo = 'L'</code> の場合、配列 $e$ には $A$ の劣対角成分が格納され、 $A$ は $LDL^H$ として因子分解される。
<code>n</code>	INTEGER。 $A$ の次数 ( $n \geq 0$ )。
<code>nrhs</code>	INTEGER。 右辺の数、すなわち、行列 $B$ の列数 ( $nrhs \geq 0$ )。

<i>d</i>	REAL (spttrs, cpttrs の場合 ) DOUBLE PRECISION (dpttrs, zpttrs の場合 )。 配列、次元は ( <i>n</i> )。 <a href="#">?pttrf</a> による因子分解で得られた対角行列 <i>D</i> の対角成分を格納する。
<i>e, b</i>	REAL (spttrs, cpttrs の場合 ) DOUBLE PRECISION (dpttrs の場合 ) COMPLEX (cpttrs の場合 ) DOUBLE COMPLEX (zpttrs の場合 )。 配列: <i>e</i> ( <i>n</i> - 1), <i>b</i> ( <i>ldb</i> , <i>nrhs</i> )。 配列 <i>e</i> には、 <a href="#">?pttrf</a> による因子分解で得られた単位二重対角係数 <i>U</i> または <i>L</i> の ( <i>n</i> - 1) 個の非対角成分が格納される ( <i>uplo</i> を参照 )。 配列 <i>b</i> には、行列 <i>B</i> が格納される。この行列の列は、連立方程式の右辺である。
<i>ldb</i>	INTEGER。 <i>b</i> のリーディング・ディメンジョン。 $ldb \geq \max(1, n)$ 。

### 出力パラメータ

<i>b</i>	解の行列 <i>X</i> によって上書きされる。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了した。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正である。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

pttrsrf ルーチンのインターフェイス特有の詳細を以下に示す。

<i>d</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>e</i>	長さ ( <i>n</i> -1) のベクトルを格納する。
<i>b</i>	サイズ ( <i>n</i> , <i>nrhs</i> ) の行列 <i>B</i> を格納する。
<i>uplo</i>	複素型でのみ使用される。'U' または 'L' でなければならない。デフォルト値は 'U' である。

## ?sytrs

UDU または LDL 因子分解された対称行列を使って、連立 1 次方程式を解く。

### 構文

#### Fortran 77:

```
call ssytrs(uplo, n, nrhs, a, lda, ipiv, b, ldb, info)
call dsytrs(uplo, n, nrhs, a, lda, ipiv, b, ldb, info)
call csytrs(uplo, n, nrhs, a, lda, ipiv, b, ldb, info)
call zsytrs(uplo, n, nrhs, a, lda, ipiv, b, ldb, info)
```

#### Fortran 95:

```
call sytrs(a, b, ipiv [,uplo] [,info])
```

### 説明

このルーチンは、対称行列  $A$  を持つ連立 1 次方程式  $AX=B$  を、 $A$  の Bunch-Kaufman 因子分解に基づいて、 $X$  について解く。

$uplo='U'$  の場合、 $A = PUDU^TP^T$

$uplo='L'$  の場合、 $A = PLDL^TP^T$

$P$  は置換行列、 $U$  と  $L$  は単位対角成分を含む上三角行列と下三角行列、 $D$  は対称ブロック対角行列である。行列  $B$  の各列に格納されている複数の右辺を使用して、この連立方程式を解く。このルーチンには、因子分解ルーチン [?sytrf](#) によって返される係数  $U$  (または  $L$ ) と配列  $ipiv$  を与える必要がある。

### 入力パラメータ

$uplo$	CHARACTER*1。'U' または 'L' でなければならない。 入力行列 $A$ の因子分解の方法を指定する。  $uplo='U'$ の場合、配列 $a$ には、因子分解 $A = PUDU^TP^T$ の上三角係数 $U$ を格納する。 $uplo='L'$ の場合、配列 $a$ には、因子分解 $A = PLDL^TP^T$ の下三角係数 $L$ を格納する。
$n$	INTEGER。行列 $A$ の次数 ( $n \geq 0$ )。

<i>nrhs</i>	INTEGER。右辺の数 ( $nrhs \geq 0$ )。
<i>ipiv</i>	INTEGER。配列、次元は $\max(1, n)$ 以上。 <a href="#">?sytrf</a> によって返される <i>ipiv</i> 配列。
<i>a, b</i>	REAL (ssytrs の場合 ) DOUBLE PRECISION (dsytrs の場合 ) COMPLEX (csytrs の場合 ) DOUBLE COMPLEX (zsytrs の場合 )。 配列 : $a(lda, *)$ , $b(ldb, *)$ 。 配列 <i>a</i> には、係数 <i>U</i> または <i>L</i> を格納する ( <i>uplo</i> を参照 )。 配列 <i>b</i> には、行列 <i>B</i> を格納する。この行列の列は、連立方程式の右辺である。  <i>a</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>b</i> の第 2 次元は、 $\max(1, nrhs)$ 以上でなければならない。
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $lda \geq \max(1, n)$ 。
<i>ldb</i>	INTEGER。 <i>b</i> の第 1 次元。 $ldb \geq \max(1, n)$ 。

### 出力パラメータ

<i>b</i>	解の行列 <i>X</i> によって上書きされる。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了した。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正である。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

sytrs ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	サイズ ( <i>n</i> , <i>n</i> ) の行列 <i>A</i> を格納する。
<i>b</i>	サイズ ( <i>n</i> , <i>nrhs</i> ) の行列 <i>B</i> を格納する。
<i>ipiv</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U' である。

## アプリケーション・ノート

各右辺  $b$  について算出された解は、摂動連立方程式  $(A + E)x = b$  の正確な解になる。  
 $c(n)$  は  $n$  の適度な 1 次関数で、 $\varepsilon$  はマシンの精度である。

$$|E| \leq c(n)\varepsilon P|U||D||U^T|P^T \quad \text{および} \quad |E| \leq c(n)\varepsilon P|L||D||L^T|P^T$$

$x_0$  が真の解である場合、算出された解  $x$  は、次の誤差範囲を満たす。

$$\frac{\|x - x_0\|_\infty}{\|x\|_\infty} \leq c(n) \operatorname{cond}(A, x) \varepsilon$$

ただし、 $\operatorname{cond}(A, x) = \| |A|^{-1} |A| |x| \|_\infty / \|x\|_\infty \leq \|A^{-1}\|_\infty \|A\|_\infty = \kappa_\infty(A)$ 。

$\operatorname{cond}(A, x)$  は、 $\kappa_\infty(A)$  よりはるかに小さくなる場合がある。

1 つの右辺ベクトルの浮動小数点演算の合計回数は、実数型の場合は約  $2n^2$ 、複素数型の場合は約  $8n^2$  になる。

条件数  $\kappa_\infty(A)$  を推定するには、[?sycon](#) を呼び出す。

解の精度を改善して誤差を推定するには、[?syrrfs](#) を呼び出す。

---

## ?hetrs

UDU または LDL 因子分解されたエルミート行列  
 を使って、連立 1 次方程式を解く。

---

### 構文

#### Fortran 77:

```
call chetrs(uplo, n, nrhs, a, lda, ipiv, b, ldb, info)
call zhetrs(uplo, n, nrhs, a, lda, ipiv, b, ldb, info)
```

#### Fortran 95:

```
call hetrs(a, b, ipiv [,uplo] [,info])
```

## 説明

このルーチンは、エルミート行列  $A$  を持つ連立 1 次方程式  $AX=B$  を、 $A$  の Bunch-Kaufman 因子分解に基づいて、 $X$  について解く。

$uplo = 'U'$  の場合、 $A = PUDU^H P^T$

$uplo = 'L'$  の場合、 $A = PLDL^H P^T$

$P$  は置換行列、 $U$  と  $L$  は単位対角成分を含む上三角行列と下三角行列、 $D$  は対称ブロック対角行列である。行列  $B$  の各列に格納されている複数の右辺を使用して、この連立方程式を解く。このルーチンには、因子分解ルーチン [?hetrf](#) によって返される係数  $U$  (または  $L$ ) と配列  $ipiv$  を与える必要がある。

## 入力パラメータ

$uplo$	CHARACTER*1。'U' または 'L' でなければならない。 入力行列 $A$ の因子分解の方法を指定する。  $uplo = 'U'$ の場合、配列 $a$ には、因子分解 $A = PUDU^H P^T$ の上三角係数 $U$ を格納する。  $uplo = 'L'$ の場合、配列 $a$ には、因子分解 $A = PLDL^H P^T$ の上三角係数 $L$ を格納する。
$n$	INTEGER。行列 $A$ の次数 ( $n \geq 0$ )。
$nrhs$	INTEGER。右辺の数 ( $nrhs \geq 0$ )。
$ipiv$	INTEGER。配列、次元は $\max(1, n)$ 以上。 <a href="#">?hetrf</a> によって返される $ipiv$ 配列。
$a, b$	COMPLEX (chetrs の場合)。 DOUBLE COMPLEX (zhetsrs の場合)。 配列: $a(lda, *)$ , $b(ldb, *)$ 。 配列 $a$ には、係数 $U$ または $L$ を格納する ( $uplo$ を参照)。 配列 $b$ には、行列 $B$ を格納する。この行列の列は、連立方程式の右辺である。  $a$ の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $b$ の第 2 次元は、 $\max(1, nrhs)$ 以上でなければならない。
$lda$	INTEGER。 $a$ の第 1 次元。 $lda \geq \max(1, n)$ 。
$ldb$	INTEGER。 $b$ の第 1 次元。 $ldb \geq \max(1, n)$ 。

### 出力パラメータ

$b$	解の行列 $X$ によって上書きされる。
$info$	INTEGER。 $info = 0$ の場合、実行は正常に終了した。 $info = -i$ の場合、 $i$ 番目のパラメータの値が不正である。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

hetrs ルーチンのインターフェイス特有の詳細を以下に示す。

$a$	サイズ $(n, n)$ の行列 $A$ を格納する。
$b$	サイズ $(n, nrhs)$ の行列 $B$ を格納する。
$ipiv$	長さ $(n)$ のベクトルを格納する。
$uplo$	'U' または 'L' でなければならない。デフォルト値は 'U' である。

### アプリケーション・ノート

各右辺  $b$  について算出された解は、摂動連立方程式  $(A + E)x = b$  の正確な解になる。 $c(n)$  は  $n$  の適度な 1 次関数で、 $\varepsilon$  はマシンの精度である。

$$|E| \leq c(n)\varepsilon P|U||D||U^H|P^T \quad \text{および} \quad |E| \leq c(n)\varepsilon P|L||D||L^H|P^T$$

$x_0$  が真の解である場合、算出された解  $x$  は、次の誤差範囲を満たす。

$$\frac{\|x - x_0\|_\infty}{\|x\|_\infty} \leq c(n) \operatorname{cond}(A, x) \varepsilon$$

ただし、 $\operatorname{cond}(A, x) = \| |A^{-1}| |A| |x| \|_\infty / \|x\|_\infty \leq \|A^{-1}\|_\infty \|A\|_\infty = \kappa_\infty(A)$ 。

$\operatorname{cond}(A, x)$  は、 $\kappa_\infty(A)$  よりはるかに小さくなる場合がある。

1 つの右辺ベクトルの浮動小数点演算の合計回数は、約  $8n^2$  になる。

条件数  $\kappa_{\infty}(A)$  を推定するには、[?hecon](#) を呼び出す。  
解の精度を改善して誤差を推定するには、[?herfs](#) を呼び出す。

---

### ?sptsr

UDU または LDL 因子分解された圧縮格納形式による対称行列を使って、連立 1 次方程式を解く。

---

#### 構文

##### Fortran 77:

```
call ssptsr(uplo, n, nrhs, ap, ipiv, b, ldb, info)
call dsptsr(uplo, n, nrhs, ap, ipiv, b, ldb, info)
call csptsr(uplo, n, nrhs, ap, ipiv, b, ldb, info)
call zsptsr(uplo, n, nrhs, ap, ipiv, b, ldb, info)
```

##### Fortran 95:

```
call sptsr(a, b, ipiv [,uplo] [,info])
```

#### 説明

このルーチンは、対称行列  $A$  を持つ連立 1 次方程式  $AX=B$  を、 $A$  の Bunch-Kaufman 因子分解に基づいて、 $X$  について解く。

$uplo='U'$  の場合、 $A = PUDU^T P^T$

$uplo='L'$  の場合、 $A = PLDL^T P^T$

$P$  は置換行列、 $U$  と  $L$  は単位対角成分を含む**圧縮形式**の上三角行列と下三角行列、 $D$  は対称ブロック対角行列である。行列  $B$  の各列に格納されている複数の右辺を使用して、この連立方程式を解く。このルーチンには、因子分解ルーチン [?sptsr](#) によって返される係数  $U$  (または  $L$ ) と配列  $ipiv$  を与える必要がある。

#### 入力パラメータ

**uplo** CHARACTER\*1。'U' または 'L' でなければならない。  
入力行列  $A$  の因子分解の方法を指定する。



	$uplo = 'U'$ の場合、配列 $ap$ には、因子分解 $A = PUDU^TP^T$ の圧縮形式の係数 $U$ を格納する。 $uplo = 'L'$ の場合、配列 $ap$ には、因子分解 $A = PLDL^TP^T$ の圧縮形式の係数 $L$ を格納する。
$n$	INTEGER。行列 $A$ の次数 ( $n \geq 0$ )。
$nrhs$	INTEGER。右辺の数 ( $nrhs \geq 0$ )。
$ipiv$	INTEGER。配列、次元は $\max(1, n)$ 以上。 <a href="#">?spturf</a> によって返される $ipiv$ 配列。
$ap, b$	REAL (ssptrs の場合) DOUBLE PRECISION (dsptrs の場合) COMPLEX (csptrs の場合) DOUBLE COMPLEX (zsptrs の場合)。 配列、 $ap(*), b(l, db, *)$ $ap$ の次元は $\max(1, n(n+1)/2)$ 以上でなければならない。 配列 $ap$ には、 $uplo$ の指定に従って、係数 $U$ または $L$ が圧縮格納形式で格納される (「 <a href="#">行列の格納形式</a> 」を参照)。  配列 $b$ には、行列 $B$ を格納する。この行列の列は、連立方程式の右辺である。 $b$ の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。
$ldb$	INTEGER。 $b$ の第 1 次元。 $ldb \geq \max(1, n)$ 。

### 出力パラメータ

$b$	解の行列 $X$ によって上書きされる。
$info$	INTEGER。 $info = 0$ の場合、実行は正常に終了した。 $info = -i$ の場合、 $i$ 番目のパラメータの値が不正である。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

spturs ルーチンのインターフェイス特有の詳細を以下に示す。

$a$	Fortran 77 インターフェイスでの引数 $ap$ を意味する。サイズ $(n*(n+1)/2)$ の配列 $A$ を格納する。
$b$	サイズ $(n, nrhs)$ の行列 $B$ を格納する。
$ipiv$	長さ $(n)$ のベクトルを格納する。

`uplo` 'U' または 'L' でなければならない。デフォルト値は 'U' である。

### アプリケーション・ノート

各右辺  $b$  について算出された解は、摂動連立方程式  $(A + E)x = b$  の正確な解になる。 $c(n)$  は  $n$  の適度な 1 次関数で、 $\varepsilon$  はマシンの精度である。

$$|E| \leq c(n)\varepsilon P|U||D||U^T|P^T \quad \text{および} \quad |E| \leq c(n)\varepsilon P|L||D||L^T|P^T$$

$x_0$  が真の解である場合、算出された解  $x$  は、次の誤差範囲を満たす。

$$\frac{\|x - x_0\|_\infty}{\|x\|_\infty} \leq c(n) \operatorname{cond}(A, x) \varepsilon$$

ただし、 $\operatorname{cond}(A, x) = \| |A^{-1}| |A| |x| \|_\infty / \|x\|_\infty \leq \|A^{-1}\|_\infty \|A\|_\infty = \kappa_\infty(A)$ 。

$\operatorname{cond}(A, x)$  は、 $\kappa_\infty(A)$  よりはるかに小さくなる場合がある。

1 つの右辺ベクトルの浮動小数点演算の合計回数は、実数型の場合は約  $2n^2$ 、複素数型の場合は約  $8n^2$  になる。

条件数  $\kappa_\infty(A)$  を推定するには、[`?spcon`](#) を呼び出す。

解の精度を改善して誤差を推定するには、[`?sprfs`](#) を呼び出す。

---

## ?hptrs

UDU または LDL 因子分解された圧縮格納形式によるエルミート行列を使って、連立 1 次方程式を解く。

---

### 構文

#### Fortran 77:

```
call chptrs(uplo, n, nrhs, ap, ipiv, b, ldb, info)
call zhptrs(uplo, n, nrhs, ap, ipiv, b, ldb, info)
```

**Fortran 95:**

```
call hptrs(a, b, ipiv [,uplo] [,info])
```

**説明**

このルーチンは、エルミート行列  $A$  を持つ連立 1 次方程式  $AX=B$  を、 $A$  の Bunch-Kaufman 因子分解に基づいて、 $X$  について解く。

$uplo='U'$  の場合、 $A = PUDU^H P^T$

$uplo='L'$  の場合、 $A = PLDL^H P^T$

$P$  は置換行列、 $U$  と  $L$  は単位対角成分を含む **圧縮形式** の上三角行列と下三角行列、 $D$  は対称ブロック対角行列である。行列  $B$  の各列に格納されている複数の右辺を使用して、この連立方程式を解く。

このルーチンには、( $U$  または  $L$  を格納する) 配列  $ap$  と、因子分解ルーチン [zhptrf](#) によって返される形式の配列  $ipiv$  を与える必要がある。

**入力パラメータ**

$uplo$	CHARACTER*1。'U' または 'L' でなければならない。 入力行列 $A$ の因子分解の方法を指定する。  $uplo='U'$ の場合、配列 $ap$ には、因子分解 $A = PUDU^H P^T$ の上三角係数 $U$ を格納する。 $uplo='L'$ の場合、配列 $ap$ には、因子分解 $A = PLDL^H P^T$ の上三角係数 $L$ を格納する。
$n$	INTEGER。行列 $A$ の次数 ( $n \geq 0$ )。
$nrhs$	INTEGER。右辺の数 ( $nrhs \geq 0$ )。
$ipiv$	INTEGER。配列、次元は $\max(1, n)$ 以上。 <a href="#">zhptrf</a> によって返される $ipiv$ 配列。
$ap, b$	COMPLEX (chptrs の場合)。 DOUBLE COMPLEX (zhptrs の場合)。 配列、 $ap(*)$ , $b(ldb, *)$ $ap$ の次元は $\max(1, n(n+1)/2)$ 以上でなければならない。 配列 $ap$ には、 $uplo$ の指定に従って、係数 $U$ または $L$ が <b>圧縮格納形式</b> で格納される (「 <a href="#">行列の格納形式</a> 」を参照)。  配列 $b$ には、行列 $B$ を格納する。この行列の列は、連立方程式の右辺である。 $b$ の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。
$ldb$	INTEGER。 $b$ の第 1 次元。 $ldb \geq \max(1, n)$ 。

## 出力パラメータ

$b$	解の行列 $X$ によって上書きされる。
$info$	INTEGER。 $info = 0$ の場合、実行は正常に終了した。 $info = -i$ の場合、 $i$ 番目のパラメータの値が不正である。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

hptrs ルーチンのインターフェイス特有の詳細を以下に示す。

$a$	Fortran 77 インターフェイスでの引数 $ap$ を意味する。サイズ $(n*(n+1)/2)$ の配列 $A$ を格納する。
$b$	サイズ $(n, nrhs)$ の行列 $B$ を格納する。
$ipiv$	長さ $(n)$ のベクトルを格納する。
$uplo$	'U' または 'L' でなければならない。デフォルト値は 'U' である。

## アプリケーション・ノート

各右辺  $b$  について算出された解は、摂動連立方程式  $(A + E)x = b$  の正確な解になる。 $c(n)$  は  $n$  の適度な 1 次関数で、 $\varepsilon$  はマシンの精度である。

$$|E| \leq c(n)\varepsilon P|U||D||U^H|P^T \quad \text{および} \quad |E| \leq c(n)\varepsilon P|L||D||L^H|P^T$$

$x_0$  が真の解である場合、算出された解  $x$  は、次の誤差範囲を満たす。

$$\frac{\|x - x_0\|_\infty}{\|x\|_\infty} \leq c(n) \text{cond}(A, x) \varepsilon$$

ただし、 $\text{cond}(A, x) = \|A^{-1}\|_\infty \|A\|_\infty / \|x\|_\infty \leq \|A^{-1}\|_\infty \|A\|_\infty = \kappa_\infty(A)$ 。

$\text{cond}(A, x)$  は、 $\kappa_\infty(A)$  よりはるかに小さくなる場合がある。

1 つの右辺ベクトルの浮動小数点演算の合計回数は、複素数型の場合は約  $8n^2$  になる。

条件数  $\kappa_{\infty}(A)$  を推定するには、[?hpccon](#) を呼び出す。  
 解の精度を改善して誤差を推定するには、[?hprfs](#) を呼び出す。

## ?trtrs

三角行列を使って、複数の右辺を持つ連立 1 次方程式を解く。

### 構文

#### Fortran 77:

```
call strtrs(uplo, trans, diag, n, nrhs, a, lda, b, ldb, info)
call dtrtrs(uplo, trans, diag, n, nrhs, a, lda, b, ldb, info)
call ctrtrs(uplo, trans, diag, n, nrhs, a, lda, b, ldb, info)
call ztrtrs(uplo, trans, diag, n, nrhs, a, lda, b, ldb, info)
```

#### Fortran 95:

```
call trtrs(a, b [,uplo] [,trans] [,diag] [,info])
```

### 説明

このルーチンは、行列  $B$  に格納された複数の右辺を使用して、三角行列  $A$  を持つ以下の連立 1 次方程式を  $X$  について解く。

$AX = B$  (trans='N' の場合)  
 $A^T X = B$  (trans='T' の場合)  
 $A^H X = B$  (trans='C' の場合。ただし、複素行列のみ)

### 入力パラメータ

**uplo** CHARACTER\*1。'U' または 'L' でなければならない。  
 $A$  が上三角行列か、下三角行列かを指定する。  
 uplo = 'U' の場合、 $A$  は上三角行列である。  
 uplo = 'L' の場合、 $A$  は下三角行列である。

**trans** CHARACTER\*1。'N' または 'T' または 'C' でなければならない。  
 trans = 'N' の場合、 $AX = B$  を  $X$  について解く。  
 trans = 'T' の場合、 $A^T X = B$  を  $X$  について解く。

<i>diag</i>	<p><i>trans</i> = 'C' の場合、<math>A^H X = B</math> を <math>X</math> について解く。          CHARACTER*1。'N' または 'U' でなければならない。</p> <p><i>diag</i> = 'N' の場合、<math>A</math> は単位三角行列ではない。  <i>diag</i> = 'U' の場合、<math>A</math> は単位三角行列である。<math>A</math> の対角成分は 1 とみなされ、配列 <i>a</i> 内で参照されない。</p>
<i>n</i>	INTEGER。 $A$ の次数。 $B$ の行数 ( $n \geq 0$ )。
<i>nrhs</i>	INTEGER。 右辺の数 ( $nrhs \geq 0$ )。
<i>a</i> , <i>b</i>	<p>REAL (<i>strtrs</i> の場合 )          DOUBLE PRECISION (<i>dtrtrs</i> の場合 )          COMPLEX (<i>ctrtrs</i> の場合 )          DOUBLE COMPLEX (<i>ztrtrs</i> の場合 )。          配列 : <i>a</i>(<i>lda</i>, *), <i>b</i>(<i>ldb</i>, *)。</p> <p>配列 <i>a</i> には、行列 <math>A</math> が格納される。          配列 <i>b</i> には、行列 <math>B</math> が格納される。この行列の列は、連立方程式の右辺である。</p> <p><math>a</math> の第 2 次元は、<math>\max(1, n)</math> 以上でなければならない。<math>b</math> の第 2 次元は、<math>\max(1, nrhs)</math> 以上でなければならない。</p>
<i>lda</i>	INTEGER。 $a$ の第 1 次元。 $lda \geq \max(1, n)$ 。
<i>ldb</i>	INTEGER。 $b$ の第 1 次元。 $ldb \geq \max(1, n)$ 。

## 出力パラメータ

<i>b</i>	解の行列 $X$ によって上書きされる。
<i>info</i>	<p>INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了した。  <i>info</i> = -<i>i</i> の場合、<i>i</i> 番目のパラメータの値が不正である。</p>

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

*trtrs* ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ap</i> を意味する。サイズ $(n*(n+1)/2)$ の行列 $A$ を格納する。
<i>b</i>	サイズ $(n, nrhs)$ の行列 $B$ を格納する。

`uplo` 'U' または 'L' でなければならない。デフォルト値は 'U' である。

`trans` 'N'、'C'、または 'T' でなければならない。デフォルト値は 'N' である。

`diag` 'N' または 'U' でなければならない。デフォルト値は 'N' である。

### アプリケーション・ノート

各右辺  $b$  について算出された解は、摂動連立方程式  $(A + E)x = b$  の正確な解になる。

$$|E| \leq c(n)\varepsilon |A|$$

$c(n)$  は  $n$  の適度な 1 次関数で、 $\varepsilon$  はマシンの精度である。

$x_0$  が真の解である場合、算出された解  $x$  は、次の誤差範囲を満たす。

$$\frac{\|x - x_0\|_\infty}{\|x\|_\infty} \leq c(n) \operatorname{cond}(A, x)\varepsilon, \quad c(n) \operatorname{cond}(A, x)\varepsilon < 1$$

ただし、 $\operatorname{cond}(A, x) = \| |A^{-1}| |A| |x| \|_\infty / \|x\|_\infty \leq \|A^{-1}\|_\infty \|A\|_\infty = \kappa_\infty(A)$ 。

$\operatorname{cond}(A, x)$  は、 $\kappa_\infty(A)$  よりはるかに小さくなる場合がある。 $A^T$  と  $A^H$  の条件数は、 $\kappa_\infty(A)$  に等しいことも、等しくないこともある。

1 つの右辺ベクトル  $b$  の浮動小数点演算のおおよその回数は、実数型の場合は  $n^2$ 、複素数型の場合は  $4n^2$  になる。

条件数  $\kappa_\infty(A)$  を推定するには、[?trcon](#) を呼び出す。

解の誤差を推定するには、[?trrfs](#) を呼び出す。

---

## ?tptrs

圧縮形式の三角行列を使って、複数の右辺を持つ連立 1 次方程式を解く。

---

### 構文

#### Fortran 77:

```
call stptrs(uplo, trans, diag, n, nrhs, ap, b, ldb, info)
call dtptrs(uplo, trans, diag, n, nrhs, ap, b, ldb, info)
```

```
call ctpttrs(uplo, trans, diag, n, nrhs, ap, b, ldb, info)
call ztpttrs(uplo, trans, diag, n, nrhs, ap, b, ldb, info)
```

### Fortran 95:

```
call tpttrs(a, b [,uplo] [,trans] [,diag] [,info])
```

### 説明

このルーチンは、行列  $B$  に格納された複数の右辺を使用して、圧縮形式の三角行列  $A$  を持つ以下の連立 1 次方程式を  $X$  について解く。

$AX = B$  (trans='N' の場合)  
 $A^T X = B$  (trans='T' の場合)  
 $A^H X = B$  (trans='C' の場合。ただし、複素行列のみ)

### 入力パラメータ

**uplo** CHARACTER\*1。'U' または 'L' でなければならない。  
 $A$  が上三角行列か、下三角行列かを指定する。  
  
 $uplo = 'U'$  の場合、 $A$  は上三角行列である。  
 $uplo = 'L'$  の場合、 $A$  は下三角行列である。

**trans** CHARACTER\*1。'N' または 'T' または 'C' でなければならない。  
  
 $trans = 'N'$  の場合、 $AX = B$  を  $X$  について解く。  
 $trans = 'T'$  の場合、 $A^T X = B$  を  $X$  について解く。  
 $trans = 'C'$  の場合、 $A^H X = B$  を  $X$  について解く。

**diag** CHARACTER\*1。'N' または 'U' でなければならない。  
  
 $diag = 'N'$  の場合、 $A$  は単位三角行列ではない。  
 $diag = 'U'$  の場合、 $A$  は単位三角行列である。対角成分は 1 とみなされ、配列  $ap$  内で参照されない。

**n** INTEGER。  $A$  の次数。  $B$  の行数 ( $n \geq 0$ )。

**nrhs** INTEGER。 右辺の数 ( $nrhs \geq 0$ )。

**ap, b** REAL (stpttrs の場合)  
DOUBLE PRECISION (dtpttrs の場合)  
COMPLEX (ctpttrs の場合)  
DOUBLE COMPLEX (ztpttrs の場合)。



配列、 $ap(*), b(ldb, *)$

$ap$  の次元は  $\max(1, n(n+1)/2)$  以上でなければならない。

配列  $ap$  は行列  $A$  を圧縮格納形式で格納する。(「[行列の格納形式](#)」を参照)。

配列  $b$  には、行列  $B$  を格納する。この行列の列は、連立方程式の右辺である。 $b$  の第 2 次元は  $\max(1, nrhs)$  以上でなければならない。

$ldb$                     INTEGER。  $b$  の第 1 次元。  $ldb \geq \max(1, n)$ 。

## 出力パラメータ

$b$                     解の行列  $X$  によって上書きされる。

$info$                   INTEGER。  $info = 0$  の場合、実行は正常に終了した。  
 $info = -i$  の場合、 $i$  番目のパラメータの値が不正である。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

tptrs ルーチンのインターフェイス特有の詳細を以下に示す。

$a$                     Fortran 77 インターフェイスでの引数  $ap$  を意味する。サイズ  $(n*(n+1)/2)$  の配列  $A$  を格納する。

$b$                     サイズ  $(n, nrhs)$  の行列  $B$  を格納する。

$uplo$                 'U' または 'L' でなければならない。デフォルト値は 'U' である。

$trans$                 'N'、'C'、または 'T' でなければならない。デフォルト値は 'N' である。

$diag$                 'N' または 'U' でなければならない。デフォルト値は 'N' である。

## アプリケーション・ノート

各右辺  $b$  について算出された解は、摂動連立方程式  $(A + E)x = b$  の正確な解になる。

$$|E| \leq c(n)\varepsilon |A|$$

$c(n)$  は  $n$  の適度な 1 次関数で、 $\varepsilon$  はマシンの精度である。

$x_0$  が真の解である場合、算出された解  $x$  は、次の誤差範囲を満たす。

ただし、 $\text{cond}(A, x) = \| |A^{-1}| |A| |x| \|_{\infty} / \|x\|_{\infty} \leq \|A^{-1}\|_{\infty} \|A\|_{\infty} = \kappa_{\infty}(A)$ 。

$$\frac{\|x - x_0\|_\infty}{\|x\|_\infty} \leq c(n) \operatorname{cond}(A, x) \varepsilon, \quad c(n) \operatorname{cond}(A, x) \varepsilon < 1$$

$\operatorname{cond}(A, x)$  は、 $\kappa_\infty(A)$  よりはるかに小さくなる場合がある。 $A^T$  と  $A^H$  の条件数は、 $\kappa_\infty(A)$  に等しいことも、等しくないこともある。

1 つの右辺ベクトル  $b$  の浮動小数点演算のおおよその回数は、実数型の場合は  $n^2$ 、複素数型の場合は  $4n^2$  になる。

条件数  $\kappa_\infty(A)$  を推定するには、[?tpcon](#) を呼び出す。

解の誤差を推定するには、[?tprfs](#) を呼び出す。

---

## ?tbtrs

帯三角行列を使って、複数の右辺を持つ連立 1 次方程式を解く。

---

### 構文

#### Fortran 77:

```
call stbtrs(uplo, trans, diag, n, kd, nrhs, ab, ldab, b, ldb, info)
call dtbtrs(uplo, trans, diag, n, kd, nrhs, ab, ldab, b, ldb, info)
call ctbtrs(uplo, trans, diag, n, kd, nrhs, ab, ldab, b, ldb, info)
call ztbtrs(uplo, trans, diag, n, kd, nrhs, ab, ldab, b, ldb, info)
```

#### Fortran 95:

```
call tbtrs(a, b [,uplo] [,trans] [,diag] [,info])
```

### 説明

このルーチンは、行列  $B$  に格納された複数の右辺を使用して、帯三角行列  $A$  を持つ以下の連立 1 次方程式を  $X$  について解く。

$AX = B$  (trans='N' の場合)

$A^T X = B$  (trans='T' の場合)

$A^H X = B$  (trans='C' の場合。ただし、複素行列のみ)

## 入力パラメータ

<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 A が上三角行列か、下三角行列かを指定する。  <i>uplo</i> = 'U' の場合、A は上三角行列である。 <i>uplo</i> = 'L' の場合、A は下三角行列である。
<i>trans</i>	CHARACTER*1。'N' または 'T' または 'C' でなければならない。  <i>trans</i> = 'N' の場合、 $AX=B$ を $X$ について解く。 <i>trans</i> = 'T' の場合、 $A^T X=B$ を $X$ について解く。 <i>trans</i> = 'C' の場合、 $A^H X=B$ を $X$ について解く。
<i>diag</i>	CHARACTER*1。'N' または 'U' でなければならない。  <i>diag</i> = 'N' の場合、A は単位三角行列ではない。  <i>diag</i> = 'U' の場合、A は単位三角行列である。対角成分は 1 とみなされ、配列 <i>ab</i> 内で参照されない。
<i>n</i>	INTEGER。A の次数。B の行数 ( $n \geq 0$ )。
<i>kd</i>	INTEGER。A の優対角成分または劣対角成分の数 ( $kd \geq 0$ )。
<i>nrhs</i>	INTEGER。右辺の数 ( $nrhs \geq 0$ )。
<i>ab, b</i>	REAL (stbtrs の場合) DOUBLE PRECISION (dtbtrs の場合) COMPLEX (ctbtrs 場合) DOUBLE COMPLEX (ztbtrs の場合)。 配列: <i>ab</i> ( <i>ldab</i> , *), <i>b</i> ( <i>ldb</i> , *)。  配列 <i>ab</i> には、行列 A を帯格納形式で格納する。  配列 <i>b</i> には、行列 B が格納される。この行列の列は、連立方程式の右辺である。  <i>ab</i> の第 2 次元は $\max(1, n)$ 以上でなければならない。 <i>b</i> の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。
<i>ldab</i>	INTEGER。ab の第 1 次元。 $ldab \geq kd + 1$ 。
<i>ldb</i>	INTEGER。b の第 1 次元。 $ldb \geq \max(1, n)$ 。

## 出力パラメータ

<i>b</i>	解の行列 $X$ によって上書きされる。
----------	----------------------

*info*                    INTEGER。 *info* = 0 の場合、実行は正常に終了した。  
*info* = -*i* の場合、*i* 番目のパラメータの値が不正である。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

tbtrs ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ab</i> を意味する。サイズ $(kd+1, n)$ の配列 <i>A</i> を格納する。
<i>b</i>	サイズ $(n, nrhs)$ の行列 <i>B</i> を格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U' である。
<i>trans</i>	'N'、'C'、または 'T' でなければならない。デフォルト値は 'N' である。
<i>diag</i>	'N' または 'U' でなければならない。デフォルト値は 'N' である。

### アプリケーション・ノート

各右辺 *b* について算出された解は、摂動連立方程式  $(A + E)x = b$  の正確な解になる。

$$|E| \leq c(n)\varepsilon |A|$$

$c(n)$  は  $n$  の適度な 1 次関数で、 $\varepsilon$  はマシンの精度である。  
 $x_0$  が真の解である場合、算出された解  $x$  は、次の誤差範囲を満たす。

$$\frac{\|x - x_0\|_\infty}{\|x\|_\infty} \leq c(n) \operatorname{cond}(A, x)\varepsilon, \quad c(n) \operatorname{cond}(A, x)\varepsilon < 1$$

ただし、 $\operatorname{cond}(A, x) = \| |A^{-1}| |A| |x| \|_\infty / \|x\|_\infty \leq \|A^{-1}\|_\infty \|A\|_\infty = \kappa_\infty(A)$ 。

$\operatorname{cond}(A, x)$  は、 $\kappa_\infty(A)$  よりはるかに小さくなる場合がある。 $A^T$  と  $A^H$  の条件数は、 $\kappa_\infty(A)$  に等しいことも、等しくないこともある。

1 つの右辺ベクトル *b* の浮動小数点演算のおおよその回数は、実数型の場合は  $2n \cdot kd$ 、複素数型の場合は  $8n \cdot kd$  になる。

条件数  $\kappa_\infty(A)$  を推定するには、[?tbcon](#) を呼び出す。  
解の誤差を推定するには、[?tbrfs](#) を呼び出す。

## 条件数を推定するためのルーチン

この節では、行列の条件数を推定するための LAPACK ルーチンについて説明する。条件数は、連立 1 次方程式の解の誤差の分析に使用される（「[誤差の分析](#)」を参照）。行列がほとんど特異である（ゼロに近い）場合は、条件数が非常に大きくなる場合がある。このため、これらのルーチンでは、実際には条件数の逆数を計算する。

## ?gecon

一般行列の条件数の逆数を、1- ノルムまたは無限ノルムによって推定する。

### 構文

#### Fortran 77:

```
call sgecon(norm, n, a, lda, anorm, rcond, work, iwork, info)
call dgecon(norm, n, a, lda, anorm, rcond, work, iwork, info)
call cgecon(norm, n, a, lda, anorm, rcond, work, rwork, info)
call zgecon(norm, n, a, lda, anorm, rcond, work, rwork, info)
```

#### Fortran 95:

```
call gecon(a, anorm, rcond [,norm] [,info])
```

### 説明

このルーチンは、次のように、一般行列  $A$  の条件数の逆数を、1- ノルムまたは無限ノルムによって推定する。

$$\kappa_1(A) = \|A\|_1 \|A^{-1}\|_1 = \kappa_\infty(A^T) = \kappa_\infty(A^H)$$

$$\kappa_\infty(A) = \|A\|_\infty \|A^{-1}\|_\infty = \kappa_1(A^T) = \kappa_1(A^H)$$

このルーチンを読み出す前に、以下の手順を実行する必要がある。

- $anorm$  を計算する ( $\|A\|_1 = \max_j \sum_i |a_{ij}|$  または  $\|A\|_\infty = \max_i \sum_j |a_{ij}|$ )。
- [?getrf](#) を呼び出して、 $A$  の  $LU$  因子分解を計算する。

### 入力パラメータ

$norm$  CHARACTER\*1. '1' または 'O' または 'I' でなければならない。

	$norm = '1'$ または $'0'$ の場合、ルーチンは $\kappa_1(A)$ を推定する。
	$norm = 'I'$ の場合、このルーチンは $\kappa_\infty(A)$ を推定する。
$n$	INTEGER。行列 $A$ の次数 ( $n \geq 0$ )。
$a, work$	REAL (sgecon の場合 ) DOUBLE PRECISION (dgecon の場合 ) COMPLEX (cgecon の場合 ) DOUBLE COMPLEX (zgecon の場合 )。 配列: $a(lda, *)$ , $work(*)$ 。  配列 $a$ には、 <a href="#">?getrf</a> によって返される、 $LU$ 因子分解された行列 $A$ を格納する。 $a$ の第 2 次元は $\max(1, n)$ 以上でなければならない。 配列 $work$ は、このルーチン用のワークスペースである。  $work$ の次元は $\max(1, 4*n)$ (実数型の場合) または $\max(1, 2*n)$ (複素数の場合) 以上でなければならない。
$anorm$	REAL (単精度の場合 ) DOUBLE PRECISION (倍精度の場合 )。 元の行列 $A$ のノルム (「 <a href="#">説明</a> 」を参照)。
$lda$	INTEGER。 $a$ の第 1 次元。 $lda \geq \max(1, n)$ 。
$iwork$	INTEGER。 ワークスペース配列、次元は $\max(1, n)$ 以上。
$rwork$	REAL (cgecon の場合 ) DOUBLE PRECISION (zgecon の場合 )。 ワークスペース配列、次元は $\max(1, 2*n)$ 以上。

## 出力パラメータ

$rcond$	REAL (単精度の場合 ) DOUBLE PRECISION (倍精度の場合 )。 条件数の逆数の推定値。 推定値のアンダーフローが発生した場合は、 $rcond = 0$ に設定される。 この場合、行列は有効な精度で特異になる。 ただし、 $rcond$ が (有効な精度で) 1.0 より小さい場合は、条件の良くない行列または特異な行列である。
$info$	INTEGER。 $info = 0$ の場合、実行は正常に終了した。 $info = -i$ の場合、 $i$ 番目のパラメータの値が不正である。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

gecon ルーチンのインターフェイス特有の詳細を以下に示す。

*a*                      サイズ  $(n,n)$  の行列 *A* を格納する。

*norm*                  '1'、'0'、または 'I' でなければならない。デフォルト値は '1' である。

### アプリケーション・ノート

計算された *rcond* は、 $\rho$  ( 真の条件数の逆数 ) より小さくはない。この値は、実際にはほぼ常に、 $10\rho$  より小さくなる。このルーチン呼び出す場合は、連立 1 次方程式  $Ax=b$  または  $A^Hx=b$  を繰り返し解く必要がある。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約  $2n^2$  で、複素数型の場合は約  $8n^2$  である。

---

## ?gbcon

帯行列の条件数の逆数を、1- ノルムまたは無限ノルムによって推定する。

---

### 構文

#### Fortran 77:

```
call sgbcon(norm, n, kl, ku, ab, ldab, ipiv, anorm, rcond, work, iwork, info)
call dgbcon(norm, n, kl, ku, ab, ldab, ipiv, anorm, rcond, work, iwork, info)
call cgbcon(norm, n, kl, ku, ab, ldab, ipiv, anorm, rcond, work, rwork, info)
call zgbcon(norm, n, kl, ku, ab, ldab, ipiv, anorm, rcond, work, rwork, info)
```

#### Fortran 95:

```
call gbcon(a, ipiv, anorm, rcond [,kl] [,norm] [,info])
```

## 説明

このルーチンは、次のように、一般帯行列  $A$  の条件数の逆数を、1- ノルムまたは無限ノルムによって推定する。

$$\kappa_1(A) = \|A\|_1 \|A^{-1}\|_1 = \kappa_\infty(A^T) = \kappa_\infty(A^H)$$

$$\kappa_\infty(A) = \|A\|_\infty \|A^{-1}\|_\infty = \kappa_1(A^T) = \kappa_1(A^H)$$

このルーチン呼び出す前に、以下の手順を実行する必要がある。

- `anorm` を計算する ( $\|A\|_1 = \max_j \sum_i |a_{ij}|$  または  $\|A\|_\infty = \max_i \sum_j |a_{ij}|$ )。
- [?gbtrf](#) を呼び出して、 $A$  の  $LU$  因子分解を計算する。

## 入力パラメータ

<code>norm</code>	CHARACTER*1。 '1' または 'O' または 'I' でなければならない。  <code>norm = '1'</code> または 'O' の場合、ルーチンは $\kappa_1(A)$ を推定する。 <code>norm = 'I'</code> の場合、このルーチンは $\kappa_\infty(A)$ を推定する。
<code>n</code>	INTEGER。 行列 $A$ の次数 ( $n \geq 0$ )。
<code>kl</code>	INTEGER。 $A$ の帯内の劣対角成分の数 ( $kl \geq 0$ )。
<code>ku</code>	INTEGER。 $A$ の帯内の優対角成分の数 ( $ku \geq 0$ )。
<code>ldab</code>	INTEGER。 配列 <code>ab</code> の第 1 次元 ( $ldab \geq 2kl + ku + 1$ )。
<code>ipiv</code>	INTEGER。 配列、次元は $\max(1, n)$ 以上。 <a href="#">?gbtrf</a> によって返される <code>ipiv</code> 配列。
<code>ab, work</code>	REAL (sgbcon の場合) DOUBLE PRECISION (dgbcon の場合) COMPLEX (cgbcon の場合) DOUBLE COMPLEX (zgbcon の場合)。  配列: <code>ab(ldab, *)</code> , <code>work(*)</code> 。  配列 <code>ab</code> には、 <a href="#">?gbtrf</a> によって返される、因子分解後の帯行列 $A$ を格納する。  <code>ab</code> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 配列 <code>work</code> は、このルーチン用のワークスペースである。  <code>work</code> の次元は、 $\max(1, 3*n)$ (実数型の場合) または $\max(1, 2*n)$ (複素数の場合) 以上でなければならない。
<code>anorm</code>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 元の行列 $A$ のノルム ( <a href="#">「説明」</a> を参照)。



<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, n)$ 以上。
<i>rwork</i>	REAL (cgbcon の場合 ) DOUBLE PRECISION (zgbcon の場合 )。 ワークスペース配列、次元は $\max(1, 2*n)$ 以上。

### 出力パラメータ

<i>rcond</i>	REAL ( 単精度の場合 ) DOUBLE PRECISION ( 倍精度の場合 )。 条件数の逆数の推定値。 推定値のアンダーフローが発生した場合は、 <i>rcond</i> = 0 に設定される。 この場合、行列は有効な精度で特異になる。 ただし、 <i>rcond</i> が ( 有効な精度で ) 1.0 より小さい場合は、条件の良くない行列または特異な行列である。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了した。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正である。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

gbcon ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ab</i> を意味する。サイズ $(2*k1+ku+1, n)$ の配列 <i>A</i> を格納する。
<i>ipiv</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>norm</i>	'1'、'O'、または 'I' でなければならない。デフォルト値は '1' である。
<i>k1</i>	省略した場合、 <i>k1</i> = <i>ku</i> とみなす。
<i>ku</i>	<i>ku</i> = <i>lda</i> - 2 * <i>k1</i> - 1 として復元する。

### アプリケーション・ノート

計算された *rcond* は、 $\rho$  ( 真の条件数の逆数 ) より小さくはない。この値は、実際にはほぼ常に、 $10\rho$  より小さくなる。このルーチンを呼び出す場合は、連立 1 次方程式  $Ax = b$  または  $A^H x = b$  を繰り返し解く必要がある。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約  $2n(ku + 2k1)$  で、複素数型の場合は約  $8n(ku + 2k1)$  である。

## ?gtcon

?gttrf によって行われた因子分解を使用して、  
三重対角行列の条件数の逆数を推定する。

---

### 構文

#### Fortran 77:

```
call sgtcon(norm, n, dl, d, du, du2, ipiv, anorm, rcond, work, iwork, info)
call dgtcon(norm, n, dl, d, du, du2, ipiv, anorm, rcond, work, iwork, info)
call cgtcon(norm, n, dl, d, du, du2, ipiv, anorm, rcond, work, info)
call zgtcon(norm, n, dl, d, du, du2, ipiv, anorm, rcond, work, info)
```

#### Fortran 95:

```
call gtcon(dl, d, du, du2, ipiv, anorm, rcond [,norm] [,info])
```

### 説明

このルーチンは、実数または複素三重対角行列  $A$  の (1- ノルムまたは無限ノルムの) 条件数の逆数を、次のように推定する。

$$\kappa_1(A) = \|A\|_1 \|A^{-1}\|_1$$
$$\kappa_\infty(A) = \|A\|_\infty \|A^{-1}\|_\infty$$

推定値は  $\|A^{-1}\|$  について得られる。条件数の逆数は、  
 $rcond = 1 / (\|A\| \|A^{-1}\|)$  として計算される。

このルーチン呼び出す前に、以下の手順を実行する必要がある。

- `anorm` を計算する ( $\|A\|_1 = \max_j \sum_i |a_{ij}|$  または  $\|A\|_\infty = \max_i \sum_j |a_{ij}|$ )。
- [?gttrf](#) を呼び出して、 $A$  の  $LU$  因子分解を計算する。

### 入力パラメータ

<code>norm</code>	CHARACTER*1。'1' または 'O' または 'I' でなければならない。 <code>norm = '1'</code> または 'O' の場合、ルーチンは $\kappa_1(A)$ を推定する。 <code>norm = 'I'</code> の場合、このルーチンは $\kappa_\infty(A)$ を推定する。
<code>n</code>	INTEGER。行列 $A$ の次数 ( $n \geq 0$ )。
<code>dl, d, du, du2</code>	REAL (sgtcon の場合) DOUBLE PRECISION (dgtcon の場合)

	COMPLEX (cgtcon の場合 ) DOUBLE COMPLEX (zgtcon の場合 )。 配列 : $d1(n-1)$ , $d(n)$ , $du(n-1)$ , $du2(n-2)$ 。 配列 $d1$ には、 <a href="#">?gttrf</a> による $A$ の $LU$ 因子分解で得られた行列 $L$ を定義する、 $(n-1)$ 個の乗数が格納される。 配列 $d$ には、 $A$ の $LU$ 因子分解で得られた上三角行列 $U$ の $n$ 個の対角成分が格納される。 配列 $du$ には、 $U$ の最初の優対角成分の $(n-1)$ 個の成分が格納される。 配列 $du2$ には、 $U$ の 2 番目の優対角成分の $(n-2)$ 個の成分が格納される。
<i>ipiv</i>	INTEGER。 配列、次元は $(n)$ 。 <a href="#">?gttrf</a> によって返される、ピボットのインデックスの配列。
<i>anorm</i>	REAL ( 単精度の場合 ) DOUBLE PRECISION ( 倍精度の場合 )。 元の行列 $A$ のノルム (「説明」を参照)。
<i>work</i>	REAL (sgtcon の場合 ) DOUBLE PRECISION (dgtcon の場合 ) COMPLEX (cgtcon の場合 ) DOUBLE COMPLEX (zgtcon の場合 )。 ワークスペース配列、次元は $(2*n)$ 。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $(n)$ 。 実数型でのみ使用される。

### 出力パラメータ

<i>rcond</i>	REAL ( 単精度の場合 ) DOUBLE PRECISION ( 倍精度の場合 )。 条件数の逆数の推定値。 推定値のアンダーフローが発生した場合は、 $rcond=0$ に設定される。 この場合、行列は有効な精度で特異になる。 ただし、 $rcond$ が ( 有効な精度で ) $1.0$ より小さい場合は、条件の良くない行列または特異な行列である。
<i>info</i>	INTEGER。 $info=0$ の場合、実行は正常に終了した。 $info=-i$ の場合、 $i$ 番目のパラメータの値が不正である。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

gtcon ルーチンのインターフェイス特有の詳細を以下に示す。

<i>d1</i>	長さ ( <i>n-1</i> ) のベクトルを格納する。
<i>d</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>du</i>	長さ ( <i>n-1</i> ) のベクトルを格納する。
<i>du2</i>	長さ ( <i>n-2</i> ) のベクトルを格納する。
<i>ipiv</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>norm</i>	'1'、'0'、または 'I' でなければならない。デフォルト値は '1' である。

### アプリケーション・ノート

計算された *rcond* は、 $\rho$  ( 真の条件数の逆数 ) より小さくはない。この値は、実際にはほぼ常に、 $10\rho$  より小さくなる。このルーチン呼び出すと、連立 1 次方程式  $Ax=b$  の解が複数回計算される。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約  $2n^2$  で、複素数型の場合は約  $8n^2$  である。

---

## ?pocon

対称( エルミート ) 正定値行列の条件数の逆数を推定する。

---

### 構文

#### Fortran 77:

```
call spocon(uplo, n, a, lda, anorm, rcond, work, iwork, info)
call dpocon(uplo, n, a, lda, anorm, rcond, work, iwork, info)
call cpocon(uplo, n, a, lda, anorm, rcond, work, rwork, info)
call zpocon(uplo, n, a, lda, anorm, rcond, work, rwork, info)
```

**Fortran 95:**

```
call pocon(a, anorm, rcond [,uplo] [,info])
```

**説明**

このルーチンは、次のように、対称 (エルミート) 正定値行列  $A$  の条件数の逆数を推定する。

$\kappa_1(A) = \|A\|_1 \|A^{-1}\|_1$  ( $A$  は対称またはエルミート行列であるため、 $\kappa_\infty(A) = \kappa_1(A)$ )。

このルーチン呼び出す前に、以下の手順を実行する必要がある。

- $anorm$  を計算する ( $\|A\|_1 = \max_j \sum_i |a_{ij}|$  または  $\|A\|_\infty = \max_i \sum_j |a_{ij}|$ )。
- [?potrf](#) を呼び出して、 $A$  のコレスキー因子分解を行う必要がある。

**入力パラメータ**

**uplo** CHARACTER\*1。'U' または 'L' でなければならない。  
入力行列  $A$  の因子分解の方法を指定する。

$uplo = 'U'$  の場合、配列  $a$  には、因子分解  $A = U^H U$  の上三角係数  $U$  を格納する。

$uplo = 'L'$  の場合、配列  $a$  には、因子分解  $A = LL^H$  の下三角係数  $L$  を格納する。

**n** INTEGER。行列  $A$  の次数 ( $n \geq 0$ )。

**a, work** REAL (spocon の場合)  
DOUBLE PRECISION (dpocon の場合)  
COMPLEX (cpocon の場合)  
DOUBLE COMPLEX (zpocon の場合)。  
配列:  $a(lda, *)$ ,  $work(*)$ 。

配列  $a$  には、[?potrf](#) によって返される、因子分解後の行列  $A$  を格納する。  
 $a$  の第 2 次元は  $\max(1, n)$  以上でなければならない。  
配列  $work$  は、このルーチン用のワークスペースである。

$work$  の次元は、 $\max(1, 3*n)$  (実数型の場合) または  $\max(1, 2*n)$  (複素数の場合) 以上でなければならない。

**lda** INTEGER。  $a$  の第 1 次元。  $lda \geq \max(1, n)$ 。

**anorm** REAL (単精度の場合)  
DOUBLE PRECISION (倍精度の場合)。  
元行列  $A$  のノルム (「説明」を参照)。

<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, n)$ 以上。
<i>rwork</i>	REAL (cpocon の場合 ) DOUBLE PRECISION (zpocon の場合 )。 ワークスペース配列、次元は $\max(1, n)$ 以上。

### 出力パラメータ

<i>rcond</i>	REAL ( 単精度の場合 ) DOUBLE PRECISION ( 倍精度の場合 )。 条件数の逆数の推定値。 推定値のアンダーフローが発生した場合は、 $rcond=0$ に設定される。 この場合、行列は有効な精度で特異になる。 ただし、 $rcond$ が ( 有効な精度で ) 1.0 より小さい場合は、条件の良くない行列または特異な行列である。
<i>info</i>	INTEGER。 $info=0$ の場合、実行は正常に終了した。 $info=-i$ の場合、 $i$ 番目のパラメータの値が不正である。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

pocon ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	サイズ $(n, n)$ の行列 $A$ を格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U' である。

### アプリケーション・ノート

計算された  $rcond$  は、 $\rho$  ( 真の条件数の逆数 ) より小さくはない。この値は、実際にはほぼ常に、 $10\rho$  より小さくなる。このルーチンを呼び出すと、連立 1 次方程式  $Ax=b$  の解が複数回計算される。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約  $2n^2$  で、複素数型の場合は約  $8n^2$  である。

## ?ppcon

圧縮格納形式の対称 (エルミート) 正定値行列の  
条件数の逆数を推定する。

### 構文

#### Fortran 77:

```
call sppcon(uplo, n, ap, anorm, rcond, work, iwork, info)
call dppcon(uplo, n, ap, anorm, rcond, work, iwork, info)
call cppcon(uplo, n, ap, anorm, rcond, work, rwork, info)
call zppcon(uplo, n, ap, anorm, rcond, work, rwork, info)
```

#### Fortran 95:

```
call ppcon(a, anorm, rcond [,uplo] [,info])
```

### 説明

このルーチンは、次のように、圧縮形式の対称 (エルミート) 正定値行列  $A$  の条件数の逆数を推定する。

- $\kappa_1(A) = \|A\|_1 \|A^{-1}\|_1$  ( $A$  は対称またはエルミート行列であるため、 $\kappa_\infty(A) = \kappa_1(A)$ )。  
このルーチンを呼び出す前に、以下の手順を実行する必要がある。
- $anorm$  を計算する ( $\|A\|_1 = \max_j \sum_i |a_{ij}|$  または  $\|A\|_\infty = \max_i \sum_j |a_{ij}|$ )。
  - [?pptrf](#) を呼び出して、 $A$  のコレスキー因子分解を行う必要がある。

### 入力パラメータ

**uplo** CHARACTER\*1。'U' または 'L' でなければならない。  
入力行列  $A$  の因子分解の方法を指定する。  
  
 $uplo = 'U'$  の場合、配列  $ap$  には、因子分解  $A = U^H U$  の上三角係数  $U$  を格納する。  
  
 $uplo = 'L'$  の場合、配列  $ap$  には、因子分解  $A = LL^H$  の下三角係数  $L$  を格納する。

**n** INTEGER。行列  $A$  の次数 ( $n \geq 0$ )。

<i>ap, work</i>	REAL (sppcon の場合 ) DOUBLE PRECISION (dppcon の場合 ) COMPLEX (cppcon の場合 ) DOUBLE COMPLEX (zppcon の場合 )。 配列 : <i>ap</i> (*), <i>work</i> (*)。  配列 <i>ap</i> には、 <a href="#">?pptrf</a> によって返される、因子分解後の圧縮形式の行列 <i>A</i> を格納する。 <i>ap</i> の次元は $\max(1, n(n+1)/2)$ 以上でなければならない。 配列 <i>work</i> は、このルーチン用のワークスペースである。  <i>work</i> の次元は、 $\max(1, 3*n)$ (実数型の場合) または $\max(1, 2*n)$ (複素数の場合) 以上でなければならない。
<i>anorm</i>	REAL (単精度の場合 ) DOUBLE PRECISION (倍精度の場合 )。 元の行列 <i>A</i> のノルム (説明を参照)。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, n)$ 以上。
<i>rwork</i>	REAL (cppcon の場合 ) DOUBLE PRECISION (zppcon の場合 )。 ワークスペース配列、次元は $\max(1, n)$ 以上。

### 出力パラメータ

<i>rcond</i>	REAL (単精度の場合 ) DOUBLE PRECISION (倍精度の場合 )。 条件数の逆数の推定値。 推定値のアンダーフローが発生した場合は、 <i>rcond</i> =0 に設定される。 この場合、行列は有効な精度で特異になる。 ただし、 <i>rcond</i> が (有効な精度で) 1.0 より小さい場合は、条件の良くない行列または特異な行列である。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了した。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正である。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ppcon ルーチンのインターフェイス特有の詳細を以下に示す。



**a** Fortran 77 インターフェイスでの引数 **ap** を意味する。サイズ  $(n*(n+1)/2)$  の配列 **A** を格納する。

**uplo** 'U' または 'L' でなければならない。デフォルト値は 'U' である。

### アプリケーション・ノート

計算された *rcond* は、 $\rho$  ( 真の条件数の逆数 ) より小さくはない。この値は、実際にはほぼ常に、 $10\rho$  より小さくなる。このルーチン呼び出すと、連立 1 次方程式  $Ax=b$  の解が複数回計算される。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約  $2n^2$  で、複素数型の場合は約  $8n^2$  である。

## ?pbcon

対称 ( エルミート ) 正定値帯行列の条件数の逆数を推定する。

### 構文

#### Fortran 77:

```
call spbcon(uplo, n, kd, ab, ldab, anorm, rcond, work, iwork, info)
call dpbcon(uplo, n, kd, ab, ldab, anorm, rcond, work, iwork, info)
call cpbcon(uplo, n, kd, ab, ldab, anorm, rcond, work, rwork, info)
call zpbcon(uplo, n, kd, ab, ldab, anorm, rcond, work, rwork, info)
```

#### Fortran 95:

```
call pbcon(a, anorm, rcond [,uplo] [,info])
```

### 説明

このルーチンは、次のように、対称 ( エルミート ) 正定値帯行列 **A** の条件数の逆数を推定する。 $\kappa_1(A) = \|A\|_1 \|A^{-1}\|_1$  (**A** は対称またはエルミート行列であるため、 $\kappa_\infty(A) = \kappa_1(A)$ )。このルーチン呼び出す前に、以下の手順を実行する必要がある。

- *anorm* を計算する ( $\|A\|_1 = \max_j \sum_i |a_{ij}|$  または  $\|A\|_\infty = \max_i \sum_j |a_{ij}|$ )。
- [?pbtrf](#) を呼び出して、**A** のコレスキー因子分解を行う必要がある。

## 入力パラメータ

<code>uplo</code>	CHARACTER*1。'U' または 'L' でなければならない。 入力行列 $A$ の因子分解の方法を指定する。 <code>uplo = 'U'</code> の場合、配列 <code>ab</code> には、コレスキー因子分解 $A = U^H U$ の上三角係数 $U$ を格納する。 <code>uplo = 'L'</code> の場合、配列 <code>ab</code> には、因子分解 $A = LL^H$ の下三角係数 $L$ を格納する。
<code>n</code>	INTEGER。行列 $A$ の次数 ( $n \geq 0$ )。
<code>kd</code>	INTEGER。 $A$ の優対角成分または劣対角成分の数 ( $kd \geq 0$ )。
<code>ldab</code>	INTEGER。配列 <code>ab</code> の第 1 次元 ( $ldab \geq kd + 1$ )。
<code>ab, work</code>	REAL (spbcon の場合 ) DOUBLE PRECISION (dpbcon の場合 ) COMPLEX (cpbcon の場合 ) DOUBLE COMPLEX (zpbcon の場合 )。  配列 : <code>ab(ldab, *)</code> , <code>work(*)</code> 。  配列 <code>ab</code> には、 <a href="#">?pbtrf</a> によって返される、因子分解後の帯形式の行列 $A$ を格納する。 <code>ab</code> の第 2 次元は $\max(1, n)$ 以上でなければならない。 配列 <code>work</code> は、このルーチン用のワークスペースである。 <code>work</code> の次元は、 $\max(1, 3 \cdot n)$ (実数型の場合) または $\max(1, 2 \cdot n)$ (複素数の場合) 以上でなければならない。
<code>anorm</code>	REAL (単精度の場合 ) DOUBLE PRECISION (倍精度の場合 )。 元の行列 $A$ のノルム (説明を参照)。
<code>iwork</code>	INTEGER。 ワークスペース配列、次元は $\max(1, n)$ 以上。
<code>rwork</code>	REAL (cpbcon の場合 ) DOUBLE PRECISION (zpbcon の場合 )。 ワークスペース配列、次元は $\max(1, n)$ 以上。

## 出力パラメータ

<code>rcond</code>	REAL (単精度の場合 ) DOUBLE PRECISION (倍精度の場合 )。 条件数の逆数の推定値。 推定値のアンダーフローが発生した場合は、 <code>rcond = 0</code> に設定される。
--------------------	---

この場合、行列は有効な精度で特異になる。  
 ただし、 $rcond$  が (有効な精度で) 1.0 より小さい場合は、条件の良くない行列または特異な行列である。

$info$  INTEGER。  $info = 0$  の場合、実行は正常に終了した。  
 $info = -i$  の場合、 $i$  番目のパラメータの値が不正である。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

pbcon ルーチンのインターフェイス特有の詳細を以下に示す。

$a$  Fortran 77 インターフェイスでの引数  $ab$  を意味する。サイズ  $(kd+1, n)$  の配列  $A$  を格納する。

$uplo$  'U' または 'L' でなければならない。デフォルト値は 'U' である。

### アプリケーション・ノート

計算された  $rcond$  は、 $\rho$  (真の条件数の逆数) より小さくはない。この値は、実際にはほぼ常に、 $10\rho$  より小さくなる。このルーチンを呼び出すと、連立 1 次方程式  $Ax = b$  の解が複数回計算される。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約  $4n(kd+1)$  で、複素数型の場合は約  $16n(kd+1)$  である。

## ?ptcon

対称(エルミート) 正定値三重対角行列の条件数の逆数を推定する。

### 構文

#### Fortran 77:

```
call sptcon(n, d, e, anorm, rcond, work, info)
call dptcon(n, d, e, anorm, rcond, work, info)
call cptcon(n, d, e, anorm, rcond, work, info)
call zptcon(n, d, e, anorm, rcond, work, info)
```

## Fortran 95:

```
call ptcon(d, e, anorm, rcond [,info])
```

## 説明

このルーチンは、[?pttrf](#) による因子分解  $A = LDL^H$  または  $A = U^H DU$  を使用して、実対称または複素エルミート正定値三重対角行列の (1- ノルムの) 条件数の逆数を、次のように計算する。

$$\kappa_1(A) = \|A\|_1 \|A^{-1}\|_1 \quad (A \text{ は対称またはエルミート行列であるため、} \kappa_\infty(A) = \kappa_1(A)).$$

ノルム  $\|A^{-1}\|$  は直接法で計算される。条件数の逆数は、 $rcond = 1 / (\|A\| \|A^{-1}\|)$  として計算される。

このルーチンを呼び出す前に、以下の手順を実行する必要がある。

- $anorm$  を  $\|A\|_1 = \max_j \sum_i |a_{ij}|$  として計算する。
- [?pttrf](#) を呼び出して、 $A$  の因子分解を計算する。

## 入力パラメータ

$n$	INTEGER。行列 $A$ の次数 ( $n \geq 0$ )。
$d, work$	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 配列、次元は ( $n$ )。 配列 $d$ には、 <a href="#">?pttrf</a> による $A$ の因子分解で得られた対角行列 $D$ の $n$ 個の対角成分が格納される。 $work$ はワークスペース配列である。
$e$	REAL (sptcon の場合) DOUBLE PRECISION (dptcon の場合) COMPLEX (cptcon の場合) DOUBLE COMPLEX (zptcon の場合)。 配列、次元は ( $n - 1$ )。 <a href="#">?pttrf</a> による因子分解で得られた単位二重対角係数 $U$ または $L$ の非対角成分を格納する。
$anorm$	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 元の行列 $A$ の 1- ノルム (説明を参照)。

## 出力パラメータ

$rcond$	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 条件数の逆数の推定値。
---------	--

推定値のアンダーフローが発生した場合は、 $rcond=0$  に設定される。  
 この場合、行列は有効な精度で特異になる。  
 ただし、 $rcond$  が (有効な精度で) 1.0 より小さい場合は、条件の良くない行列または特異な行列である。

*info*                    INTEGER。  
*info* = 0 の場合、実行は正常に終了した。  
*info* = -*i* の場合、*i* 番目のパラメータの値が不正である。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

gtcon ルーチンのインターフェイス特有の詳細を以下に示す。

*d*                    長さ (*n*) のベクトルを格納する。  
*e*                    長さ (*n*-1) のベクトルを格納する。

### アプリケーション・ノート

計算された  $rcond$  は、 $\rho$  (真の条件数の逆数) より小さくはない。この値は、実際にはほぼ常に、 $10\rho$  より小さくなる。このルーチン呼び出すと、連立 1 次方程式  $Ax=b$  の解が複数回計算される。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約  $4n(kd+1)$  で、複素数型の場合は約  $16n(kd+1)$  である。

---

## ?sycon

対称行列の条件数の逆数を推定する。

---

### 構文

#### Fortran 77:

```
call ssycon(uplo, n, a, lda, ipiv, anorm, rcond, work, iwork, info)
call dsycon(uplo, n, a, lda, ipiv, anorm, rcond, work, iwork, info)
call csycon(uplo, n, a, lda, ipiv, anorm, rcond, work, info)
call zsycon(uplo, n, a, lda, ipiv, anorm, rcond, work, info)
```

## Fortran 95:

```
call sycon(a, ipiv, anorm, rcond [,uplo] [,info])
```

## 説明

このルーチンは、次のように、対称行列  $A$  の条件数の逆数を推定する。

$$\kappa_1(A) = \|A\|_1 \|A^{-1}\|_1 \quad (A \text{ は対称行列であるため、} \kappa_\infty(A) = \kappa_1(A)).$$

このルーチン呼び出す前に、以下の手順を実行する必要がある。

- $anorm$  を計算する ( $\|A\|_1 = \max_j \sum_i |a_{ij}|$  または  $\|A\|_\infty = \max_i \sum_j |a_{ij}|$ )。
- [?sytrf](#) を呼び出して、 $A$  の因子分解を計算する。

## 入力パラメータ

$uplo$	CHARACTER*1。'U' または 'L' でなければならない。 入力行列 $A$ の因子分解の方法を指定する。 $uplo = 'U'$ の場合、配列 $a$ には、因子分解 $A = PUDU^T P^T$ の上三角係数 $U$ を格納する。 $uplo = 'L'$ の場合、配列 $a$ には、因子分解 $A = PLDL^T P^T$ の下三角係数 $L$ を格納する。
$n$	INTEGER。行列 $A$ の次数 ( $n \geq 0$ )。
$a, work$	REAL (ssycon の場合 ) DOUBLE PRECISION (dsycon の場合 ) COMPLEX (csycon の場合 ) DOUBLE COMPLEX (zsycon の場合 )。 配列: $a(lda, *)$ , $work(*)$ 。  配列 $a$ には、 <a href="#">?sytrf</a> によって返される、因子分解後の行列 $A$ を格納する。 $a$ の第 2 次元は $\max(1, n)$ 以上でなければならない。  配列 $work$ は、このルーチン用のワークスペースである。 $work$ の次元は $\max(1, 2*n)$ 以上でなければならない。
$lda$	INTEGER。 $a$ の第 1 次元。 $lda \geq \max(1, n)$ 。
$ipiv$	INTEGER。配列、次元は $\max(1, n)$ 以上。 <a href="#">?sytrf</a> によって返される $ipiv$ 配列。
$anorm$	REAL ( 単精度の場合 ) DOUBLE PRECISION ( 倍精度の場合 )。 元の行列 $A$ のノルム ( 説明を参照 )。

*iwork* INTEGER。  
ワークスペース配列、次元は  $\max(1, n)$  以上。

## 出力パラメータ

*rcond* REAL ( 単精度の場合 )  
DOUBLE PRECISION ( 倍精度の場合 )。  
条件数の逆数の推定値。  
推定値のアンダーフローが発生した場合は、*rcond* = 0 に設定される。  
この場合、行列は有効な精度で特異になる。  
ただし、*rcond* が ( 有効な精度で ) 1.0 より小さい場合は、条件の良くない行列または特異な行列である。

*info* INTEGER。  
*info* = 0 の場合、実行は正常に終了した。  
*info* = -*i* の場合、*i* 番目のパラメータの値が不正である。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

sycon ルーチンのインターフェイス特有の詳細を以下に示す。

*a* サイズ (*n*, *n*) の行列 *A* を格納する。  
*ipiv* 長さ (*n*) のベクトルを格納する。  
*uplo* 'U' または 'L' でなければならない。デフォルト値は 'U' である。

## アプリケーション・ノート

計算された *rcond* は、 $\rho$  ( 真の条件数の逆数 ) より小さくはない。この値は、実際にはほぼ常に、 $10\rho$  より小さくなる。このルーチン呼び出すと、連立 1 次方程式  $Ax = b$  の解が複数回計算される。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約  $2n^2$  で、複素数型の場合は約  $8n^2$  である。

## ?hecon

エルミート行列の条件数の逆数を計算する。

---

### 構文

#### Fortran 77:

```
call checon(uplo, n, a, lda, ipiv, anorm, rcond, work, info)
call zhecon(uplo, n, a, lda, ipiv, anorm, rcond, work, info)
```

#### Fortran 95:

```
call hecon(a, ipiv, anorm, rcond [,uplo] [,info])
```

### 説明

このルーチンは、次のように、エルミート行列  $A$  の条件数の逆数を推定する。

$$\kappa_1(A) = \|A\|_1 \|A^{-1}\|_1 \quad (A \text{ はエルミート行列のため、} \kappa_\infty(A) = \kappa_1(A)).$$

このルーチン呼び出す前に、以下の手順を実行する必要がある。

- $anorm$  を計算する ( $\|A\|_1 = \max_j \sum_i |a_{ij}|$  または  $\|A\|_\infty = \max_i \sum_j |a_{ij}|$ )。
- [?hetrf](#) を呼び出して、 $A$  の因子分解を計算する。

### 入力パラメータ

<code>uplo</code>	CHARACTER*1。'U' または 'L' でなければならない。 入力行列 $A$ の因子分解の方法を指定する。  <code>uplo = 'U'</code> の場合、配列 $a$ には、因子分解 $A = PUDU^H P^T$ の上三角係数 $U$ を格納する。  <code>uplo = 'L'</code> の場合、配列 $a$ には、因子分解 $A = PLDL^H P^T$ の上三角係数 $L$ を格納する。
<code>n</code>	INTEGER。行列 $A$ の次数 ( $n \geq 0$ )。
<code>a, work</code>	COMPLEX (checon の場合) DOUBLE COMPLEX (zhecon の場合)。 配列: <code>a(lda, *)</code> , <code>work(*)</code> 。  配列 $a$ には、 <a href="#">?hetrf</a> によって返される、因子分解後の行列 $A$ を格納する。 $a$ の第 2 次元は $\max(1, n)$ 以上でなければならない。



配列 *work* は、このルーチン用のワークスペースである。  
*work* の次元は  $\max(1, 2*n)$  以上でなければならない。

*lda* INTEGER。 *a* の第 1 次元。  $lda \geq \max(1, n)$ 。

*ipiv* INTEGER。 配列、次元は  $\max(1, n)$  以上。  
[?hetrf](#) によって返される *ipiv* 配列。

*anorm* REAL ( 単精度の場合 )  
 DOUBLE PRECISION ( 倍精度の場合 )。  
 元の行列 *A* のノルム ( 説明を参照 )。

### 出力パラメータ

*rcond* REAL ( 単精度の場合 )  
 DOUBLE PRECISION ( 倍精度の場合 )。  
 条件数の逆数の推定値。  
 推定値のアンダーフローが発生した場合は、*rcond*=0 に設定される。  
 この場合、行列は有効な精度で特異になる。  
 ただし、*rcond* が ( 有効な精度で ) 1.0 より小さい場合は、条件の良くない行列または特異な行列である。

*info* INTEGER。  
*info*=0 の場合、実行は正常に終了した。  
*info*=-*i* の場合、*i* 番目のパラメータの値が不正である。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

hecon ルーチンのインターフェイス特有の詳細を以下に示す。

*a* サイズ (*n*,*n*) の行列 *A* を格納する。

*ipiv* 長さ (*n*) のベクトルを格納する。

*uplo* 'U' または 'L' でなければならない。デフォルト値は 'U' である。

### アプリケーション・ノート

計算された *rcond* は、 $\rho$  ( 真の条件数の逆数 ) より小さくはない。この値は、実際にはほぼ常に、 $10\rho$  より小さくなる。このルーチンを呼び出す場合は、連立 1 次方程式  $Ax=b$  を繰り返し解く必要がある。その回数は、通常は 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、約  $8n^2$  である。

## ?spcon

圧縮格納形式の対称行列の条件数の逆数を推定する。

### 構文

#### Fortran 77:

```
call sspcon(uplo, n, ap, ipiv, anorm, rcond, work, iwork, info)
call dspcon(uplo, n, ap, ipiv, anorm, rcond, work, iwork, info)
call cspcon(uplo, n, ap, ipiv, anorm, rcond, work, info)
call zspcon(uplo, n, ap, ipiv, anorm, rcond, work, info)
```

#### Fortran 95:

```
call spcon(a, ipiv, anorm, rcond [,uplo] [,info])
```

### 説明

このルーチンは、次のように、圧縮格納形式の対称行列  $A$  の条件数の逆数を推定する。

$$\kappa_1(A) = \|A\|_1 \|A^{-1}\|_1 \quad (A \text{ は対称行列であるため、} \kappa_\infty(A) = \kappa_1(A)).$$

このルーチン呼び出す前に、以下の手順を実行する必要がある。

- $anorm$  を計算する ( $\|A\|_1 = \max_j \sum_i |a_{ij}|$  または  $\|A\|_\infty = \max_i \sum_j |a_{ij}|$ )。
- [?sptrf](#) を呼び出して、 $A$  の因子分解を計算する。

### 入力パラメータ

$uplo$	CHARACTER*1。'U' または 'L' でなければならない。 入力行列 $A$ の因子分解の方法を指定する。 $uplo = 'U'$ の場合、配列 $ap$ には、因子分解 $A = PUDU^T P^T$ の圧縮形式の上三角係数 $U$ を格納する。 $uplo = 'L'$ の場合、配列 $ap$ には、因子分解 $A = PLDL^T P^T$ の圧縮形式の下三角係数 $L$ を格納する。
$n$	INTEGER。行列 $A$ の次数 ( $n \geq 0$ )。

<i>ap, work</i>	<p>REAL (sspcon の場合 )  DOUBLE PRECISION (dspcon の場合 )  COMPLEX (cspcon の場合 )  DOUBLE COMPLEX (zspcon の場合 )。  配列 : <i>ap</i>( * ), <i>work</i>( * )。</p> <p>配列 <i>ap</i> には、<a href="#">?sptf</a> によって返される、因子分解後の圧縮形式の行列 <i>A</i> を格納する。  <i>ap</i> の次元は <math>\max(1, n(n+1)/2)</math> 以上でなければならない。</p> <p>配列 <i>work</i> は、このルーチン用のワークスペースである。  <i>work</i> の次元は <math>\max(1, 2*n)</math> 以上でなければならない。</p>
<i>ipiv</i>	<p>INTEGER。配列、次元は <math>\max(1, n)</math> 以上。  <a href="#">?sptf</a> によって返される <i>ipiv</i> 配列。</p>
<i>anorm</i>	<p>REAL ( 単精度の場合 )  DOUBLE PRECISION ( 倍精度の場合 )。  元の行列 <i>A</i> のノルム ( 説明を参照 )。</p>
<i>iwork</i>	<p>INTEGER。  ワークスペース配列、次元は <math>\max(1, n)</math> 以上。</p>

## 出力パラメータ

<i>rcond</i>	<p>REAL ( 単精度の場合 )  DOUBLE PRECISION ( 倍精度の場合 )。  条件数の逆数の推定値。  推定値のアンダーフローが発生した場合は、<i>rcond</i> = 0 に設定される。  この場合、行列は有効な精度で特異になる。  ただし、<i>rcond</i> が ( 有効な精度で ) 1.0 より小さい場合は、条件の良くない行列または特異な行列である。</p>
<i>info</i>	<p>INTEGER。  <i>info</i> = 0 の場合、実行は正常に終了した。  <i>info</i> = -<i>i</i> の場合、<i>i</i> 番目のパラメータの値が不正である。</p>

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

spcon ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ap</i> を意味する。サイズ $(n*(n+1)/2)$ の配列 <i>A</i> を格納する。
<i>ipiv</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U' である。

### アプリケーション・ノート

計算された *rcond* は、 $\rho$  ( 真の条件数の逆数 ) より小さくはない。この値は、実際にはほぼ常に、 $10\rho$  より小さくなる。このルーチン呼び出すと、連立 1 次方程式  $Ax=b$  の解が複数回計算される。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約  $2n^2$  で、複素数型の場合は約  $8n^2$  である。

---

## ?hpcon

圧縮格納形式のエルミート行列の条件数の逆数を推定する。

---

### 構文

#### Fortran 77:

```
call chpcon(uplo, n, ap, ipiv, anorm, rcond, work, info)
call zhpcon(uplo, n, ap, ipiv, anorm, rcond, work, info)
```

#### Fortran 95:

```
call hpcon(a, ipiv, anorm, rcond [,uplo] [,info])
```

### 説明

このルーチンは、次のように、エルミート行列 *A* の条件数の逆数を推定する。

$$\kappa_1(A) = \|A\|_1 \|A^{-1}\|_1 \quad (A \text{ はエルミート行列のため、}\kappa_\infty(A) = \kappa_1(A)).$$

このルーチン呼び出す前に、以下の手順を実行する必要がある。

- *anorm* を計算する ( $\|A\|_1 = \max_j \sum_i |a_{ij}|$  または  $\|A\|_\infty = \max_i \sum_j |a_{ij}|$ )。
- [?hptrf](#) を呼び出して、*A* の因子分解を計算する。

## 入力パラメータ

<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 入力行列 $A$ の因子分解の方法を指定する。  $uplo = 'U'$ の場合、配列 $ap$ には、因子分解 $A = PUDU^TP^T$ の圧縮形式の上三角係数 $U$ を格納する。  $uplo = 'L'$ の場合、配列 $ap$ には、因子分解 $A = PLDL^TP^T$ の圧縮形式の下三角係数 $L$ を格納する。
<i>n</i>	INTEGER。行列 $A$ の次数 ( $n \geq 0$ )。
<i>ap, work</i>	COMPLEX (chpcon の場合 ) DOUBLE COMPLEX (zhpcon の場合 )。 配列 : $ap(*)$ , $work(*)$ 。  配列 $ap$ には、 <a href="#">zhptrf</a> によって返される、因子分解後の圧縮形式の行列 $A$ を格納する。 $ap$ の次元は $\max(1, n(n+1)/2)$ 以上でなければならない。  配列 $work$ は、このルーチン用のワークスペースである。 $work$ の次元は $\max(1, 2*n)$ 以上でなければならない。
<i>ipiv</i>	INTEGER。配列、次元は $\max(1, n)$ 以上。 <a href="#">zhptrf</a> によって返される <i>ipiv</i> 配列。
<i>anorm</i>	REAL ( 単精度の場合 ) DOUBLE PRECISION ( 倍精度の場合 )。 元の行列 $A$ のノルム ( 説明を参照 )。

## 出力パラメータ

<i>rcond</i>	REAL ( 単精度の場合 ) DOUBLE PRECISION ( 倍精度の場合 )。 条件数の逆数の推定値。 推定値のアンダーフローが発生した場合は、 $rcond=0$ に設定される。 この場合、行列は有効な精度で特異になる。 ただし、 $rcond$ が ( 有効な精度で ) 1.0 より小さい場合は、条件の良くない行列または特異な行列である。
<i>info</i>	INTEGER。 $info = 0$ の場合、実行は正常に終了した。 $info = -i$ の場合、 $i$ 番目のパラメータの値が不正である。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

gbcon ルーチンのインターフェイス特有の詳細を以下に示す。

**a** Fortran 77 インターフェイスでの引数 **ap** を意味する。サイズ  $(n*(n+1)/2)$  の配列 **A** を格納する。

**ipiv** 長さ  $(n)$  のベクトルを格納する。

### アプリケーション・ノート

計算された  $rcond$  は、 $\rho$  ( 真の条件数の逆数 ) より小さくはない。この値は、実際にはほぼ常に、 $10\rho$  より小さくなる。このルーチン呼び出す場合は、連立 1 次方程式  $Ax=b$  を繰り返し解く必要がある。その回数は、通常は 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、約  $8n^2$  である。

---

## ?trcon

三角行列の条件数の逆数を推定する。

---

### 構文

#### Fortran 77:

```
call strcon(norm, uplo, diag, N, a, lda, rcond, work, iwork, info)
call dtrcon(norm, uplo, diag, N, a, lda, rcond, work, iwork, info)
call ctrcon(norm, uplo, diag, N, a, lda, rcond, work, rwork, info)
call ztrcon(norm, uplo, diag, N, a, lda, rcond, work, rwork, info)
```

#### Fortran 95:

```
call trcon(a, rcond [,uplo] [,diag] [,norm] [,info])
```

### 説明

このルーチンは、次のように、三角行列  $A$  の条件数の逆数を、1- ノルムまたは無限ノルムによって推定する。

$$\kappa_1(A) = \|A\|_1 \|A^{-1}\|_1 = \kappa_\infty(A^T) = \kappa_\infty(A^H)$$

$$\kappa_{\infty}(A) = \|A\|_{\infty} \|A^{-1}\|_{\infty} = \kappa_1(A^T) = \kappa_1(A^H)$$

### 入力パラメータ

<i>norm</i>	CHARACTER*1。'1' または 'O' または 'I' でなければならない。 <i>norm</i> = '1' または 'O' の場合、ルーチンは $\kappa_1(A)$ を推定する。 <i>norm</i> = 'I' の場合、このルーチンは $\kappa_{\infty}(A)$ を推定する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>A</i> が上三角行列か、下三角行列かを指定する。  <i>uplo</i> = 'U' の場合、配列 <i>a</i> には、 <i>A</i> の上三角部分を格納する。その他の配列成分は参照されない。  <i>uplo</i> = 'L' の場合、配列 <i>a</i> には、 <i>A</i> の下三角部分を格納する。その他の配列成分は参照されない。
<i>diag</i>	CHARACTER*1。'N' または 'U' でなければならない。  <i>diag</i> = 'N' の場合、 <i>A</i> は単位三角行列ではない。  <i>diag</i> = 'U' の場合、 <i>A</i> は単位三角行列である。対角成分は 1 とみなされ、配列 <i>a</i> 内で参照されない。
<i>n</i>	INTEGER。行列 <i>A</i> の次数 ( $n \geq 0$ )。
<i>a</i> , <i>work</i>	REAL ( <i>strcon</i> の場合) DOUBLE PRECISION ( <i>dtrcon</i> の場合) COMPLEX ( <i>ctrcon</i> の場合) DOUBLE COMPLEX ( <i>ztrcon</i> の場合)。 配列: <i>a</i> ( <i>lda</i> , *), <i>work</i> (*)。  配列 <i>a</i> には、行列 <i>A</i> を格納する。 <i>a</i> の第 2 次元は $\max(1, n)$ 以上でなければならない。 配列 <i>work</i> は、このルーチン用のワークスペースである。  <i>work</i> の次元は、 $\max(1, 3*n)$ (実数型の場合) または $\max(1, 2*n)$ (複素数の場合) 以上でなければならない。
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $lda \geq \max(1, n)$ 。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, n)$ 以上。
<i>rwork</i>	REAL ( <i>ctrcon</i> の場合) DOUBLE PRECISION ( <i>ztrcon</i> の場合)。 ワークスペース配列、次元は $\max(1, n)$ 以上。

### 出力パラメータ

<i>rcond</i>	REAL ( 単精度の場合 ) DOUBLE PRECISION ( 倍精度の場合 )。 条件数の逆数の推定値。 推定値のアンダーフローが発生した場合は、 <i>rcond</i> =0 に設定される。 この場合、行列は有効な精度で特異になる。 ただし、 <i>rcond</i> が ( 有効な精度で ) 1.0 より小さい場合は、条件の良くない行列または特異な行列である。
<i>info</i>	INTEGER。 <i>info</i> =0 の場合、実行は正常に終了した。 <i>info</i> =- <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正である。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

trcon ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	サイズ ( <i>n</i> , <i>n</i> ) の行列 <i>A</i> を格納する。
<i>norm</i>	'1'、'O'、または 'I' でなければならない。デフォルト値は '1' である。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U' である。
<i>diag</i>	'N' または 'U' でなければならない。デフォルト値は 'N' である。

### アプリケーション・ノート

計算された *rcond* は、 $\rho$  ( 真の条件数の逆数 ) より小さくはならない。この値は、実際にはほぼ常に、 $10\rho$  より小さくなる。このルーチン呼び出すと、連立 1 次方程式  $Ax=b$  の解が複数回計算される。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約  $n^2$  で、複素数型の場合は約  $4n^2$  である。



## ?tpcon

圧縮格納形式の三角行列の条件数の逆数を推定する。

### 構文

#### Fortran 77:

```
call stpcon(norm, uplo, diag, n, ap, rcond, work, iwork, info)
call dtpcon(norm, uplo, diag, n, ap, rcond, work, iwork, info)
call ctpcon(norm, uplo, diag, n, ap, rcond, work, rwork, info)
call ztpcon(norm, uplo, diag, n, ap, rcond, work, rwork, info)
```

#### Fortran 95:

```
call tpcon(a, rcond [,uplo] [,diag] [,norm] [,info])
```

### 説明

このルーチンは、次のように、圧縮格納形式の三角行列  $A$  の条件数の逆数を、1- ノルムまたは無限ノルムによって推定する。

$$\kappa_1(A) = \|A\|_1 \|A^{-1}\|_1 = \kappa_\infty(A^T) = \kappa_\infty(A^H)$$

$$\kappa_\infty(A) = \|A\|_\infty \|A^{-1}\|_\infty = \kappa_1(A^T) = \kappa_1(A^H)$$

### 入力パラメータ

<i>norm</i>	CHARACTER*1。'1' または 'O' または 'I' でなければならない。 <i>norm</i> = '1' または 'O' の場合、ルーチンは $\kappa_1(A)$ を推定する。 <i>norm</i> = 'I' の場合、このルーチンは $\kappa_\infty(A)$ を推定する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 $A$ が上三角行列か、下三角行列かを指定する。 <i>uplo</i> = 'U' の場合、配列 <i>ap</i> には、 $A$ の上三角部分を圧縮形式で格納する。 <i>uplo</i> = 'L' の場合、配列 <i>ap</i> には、 $A$ の下三角部分を圧縮形式で格納する。
<i>diag</i>	CHARACTER*1。'N' または 'U' でなければならない。 <i>diag</i> = 'N' の場合、 $A$ は単位三角行列ではない。

	<i>diag</i> = 'U' の場合、 <i>A</i> は単位三角行列である。対角成分は 1 とみなされ、配列 <i>ap</i> 内で参照されない。
<i>n</i>	INTEGER。行列 <i>A</i> の次数 ( $n \geq 0$ )。
<i>ap</i> , <i>work</i>	REAL ( <i>stpcon</i> の場合 ) DOUBLE PRECISION ( <i>dtpcon</i> の場合 ) COMPLEX ( <i>ctpcon</i> の場合 ) DOUBLE COMPLEX ( <i>ztpcon</i> の場合 )。 配列 : <i>ap</i> (*), <i>work</i> (*)。  配列 <i>ap</i> には、圧縮形式の行列 <i>A</i> を格納する。 <i>ap</i> の次元は $\max(1, n(n+1)/2)$ 以上でなければならない。 配列 <i>work</i> は、このルーチン用のワークスペースである。  <i>work</i> の次元は、 $\max(1, 3*n)$ (実数型の場合) または $\max(1, 2*n)$ (複素数の場合) 以上でなければならない。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, n)$ 以上。
<i>rwork</i>	REAL ( <i>ctpcon</i> の場合 ) DOUBLE PRECISION ( <i>ztpcon</i> の場合 )。 ワークスペース配列、次元は $\max(1, n)$ 以上。

### 出力パラメータ

<i>rcond</i>	REAL (単精度の場合 ) DOUBLE PRECISION (倍精度の場合 )。 条件数の逆数の推定値。 推定値のアンダーフローが発生した場合は、 <i>rcond</i> = 0 に設定される。 この場合、行列は有効な精度で特異になる。 ただし、 <i>rcond</i> が (有効な精度で) 1.0 より小さい場合は、条件の良くない行列または特異な行列である。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了した。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正である。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

tpcon ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ap</i> を意味する。サイズ $(n*(n+1)/2)$ の配列 <i>A</i> を格納する。
<i>norm</i>	'1'、'0'、または 'I' でなければならない。デフォルト値は '1' である。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U' である。
<i>diag</i>	'N' または 'U' でなければならない。デフォルト値は 'N' である。

### アプリケーション・ノート

計算された *rcond* は、 $\rho$  (真の条件数の逆数) より小さくはない。この値は、実際にはほぼ常に、 $10\rho$  より小さくなる。このルーチン呼び出すと、連立 1 次方程式  $Ax = b$  の解が複数回計算される。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約  $n^2$  で、複素数型の場合は約  $4n^2$  である。

## ?tbcon

三角帯行列の条件数の逆数を推定する。

### 構文

#### Fortran 77:

```
call stbcon(norm, uplo, diag, n, kd, ab, ldab, rcond, work, iwork, info)
call dtbcon(norm, uplo, diag, n, kd, ab, ldab, rcond, work, iwork, info)
call ctbcon(norm, uplo, diag, n, kd, ab, ldab, rcond, work, rwork, info)
call ztbcon(norm, uplo, diag, n, kd, ab, ldab, rcond, work, rwork, info)
```

#### Fortran 95:

```
call tbcon(a, rcond [,uplo] [,diag] [,norm] [,info])
```

### 説明

このルーチンは、次のように、三角帯行列 *A* の条件数の逆数を、1- ノルムまたは無限ノルムによって推定する。

$$\begin{aligned}\kappa_1(A) &= \|A\|_1 \|A^{-1}\|_1 = \kappa_\infty(A^T) = \kappa_\infty(A^H) \\ \kappa_\infty(A) &= \|A\|_\infty \|A^{-1}\|_\infty = \kappa_1(A^T) = \kappa_1(A^H)\end{aligned}$$

## 入力パラメータ

<i>norm</i>	CHARACTER*1。'1' または 'O' または 'I' でなければならない。 <i>norm</i> = '1' または 'O' の場合、ルーチンは $\kappa_1(A)$ を推定する。 <i>norm</i> = 'I' の場合、このルーチンは $\kappa_\infty(A)$ を推定する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>A</i> が上三角行列か、下三角行列かを指定する。 <i>uplo</i> = 'U' の場合、配列 <i>ap</i> には、 <i>A</i> の上三角部分を圧縮形式で格納する。 <i>uplo</i> = 'L' の場合、配列 <i>ap</i> には、 <i>A</i> の下三角部分を圧縮形式で格納する。
<i>diag</i>	CHARACTER*1。'N' または 'U' でなければならない。  <i>diag</i> = 'N' の場合、 <i>A</i> は単位三角行列ではない。  <i>diag</i> = 'U' の場合、 <i>A</i> は単位三角行列である。対角成分は 1 とみなされ、配列 <i>ab</i> 内で参照されない。
<i>n</i>	INTEGER。行列 <i>A</i> の次数 ( $n \geq 0$ )。
<i>kd</i>	INTEGER。 <i>A</i> の優対角成分または劣対角成分の数 ( $kd \geq 0$ )。
<i>ab, work</i>	REAL (stbcon の場合 ) DOUBLE PRECISION (dtbcon の場合 ) COMPLEX (ctbcon の場合 ) DOUBLE COMPLEX (ztbcon の場合 )。 配列 : <i>ab</i> ( <i>ldab</i> , *), <i>work</i> (*)。  配列 <i>ab</i> には、帯行列 <i>A</i> を格納する。 <i>ab</i> の第 2 次元は $\max(1, n)$ 以上でなければならない。 配列 <i>work</i> は、このルーチン用のワークスペースである。 <i>work</i> の次元は、 $\max(1, 3*n)$ (実数型の場合) または $\max(1, 2*n)$ (複素数の場合) 以上でなければならない。
<i>ldab</i>	INTEGER。配列 <i>ab</i> の第 1 次元 ( $ldab \geq kd + 1$ )。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, n)$ 以上。
<i>rwork</i>	REAL (ctbcon の場合 ) DOUBLE PRECISION (ztbcon の場合 )。 ワークスペース配列、次元は $\max(1, n)$ 以上。

**出力パラメータ**

<i>rcond</i>	REAL ( 単精度の場合 ) DOUBLE PRECISION ( 倍精度の場合 )。 条件数の逆数の推定値。 推定値のアンダーフローが発生した場合は、 <i>rcond</i> =0 に設定される。 この場合、行列は有効な精度で特異になる。 ただし、 <i>rcond</i> が ( 有効な精度で ) 1.0 より小さい場合は、条件の良くない行列または特異な行列である。
<i>info</i>	INTEGER。 <i>info</i> =0 の場合、実行は正常に終了した。 <i>info</i> =- <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正である。

**Fortran 95 インターフェイス・ノート**

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

tbcon ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ab</i> を意味する。サイズ $(kd+1, n)$ の配列 <i>A</i> を格納する。
<i>norm</i>	'1'、'0'、または 'I' でなければならない。デフォルト値は '1' である。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U' である。
<i>diag</i>	'N' または 'U' でなければならない。デフォルト値は 'N' である。

**アプリケーション・ノート**

計算された *rcond* は、 $\rho$  ( 真の条件数の逆数 ) より小さくはない。この値は、実際にはほぼ常に、 $10\rho$  より小さくなる。このルーチン呼び出すと、連立 1 次方程式  $Ax=b$  の解が複数回計算される。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約  $2n(kd+1)$  で、複素数型の場合は約  $8n(kd+1)$  である。

**解の精度の改善と誤差の推定**

この節では、算出された連立 1 次方程式の解の精度を改善し、解の誤差を推定するための LAPACK ルーチンについて説明する。これらのルーチン呼び出す前に、連立方程式の行列を因子分解し、解を計算する必要がある (「[行列の因子分解用のルーチン](#)」および「[連立 1 次方程式を解くためのルーチン](#)」を参照)。

## ?gerfs

一般行列による連立 1 次方程式の解の精度を改善し、解の誤差を推定する。

### 構文

#### Fortran 77:

```
call sgerfs(trans, n, nrhs, a, lda, af, ldaf, ipiv, b, ldb, x, ldx, ferr,
           berr, work, iwork, info)
call dgerfs(trans, n, nrhs, a, lda, af, ldaf, ipiv, b, ldb, x, ldx, ferr,
           berr, work, iwork, info)
call cgerfs(trans, n, nrhs, a, lda, af, ldaf, ipiv, b, ldb, x, ldx, ferr,
           berr, work, rwork, info)
call zgerfs(trans, n, nrhs, a, lda, af, ldaf, ipiv, b, ldb, x, ldx, ferr,
           berr, work, rwork, info)
```

#### Fortran 95:

```
call gerfs(a, af, ipiv, b, x [,trans] [,ferr] [,berr] [,info])
```

### 説明

このルーチンは、一般行列  $A$  による、複数の右辺を持つ連立 1 次方程式  $AX=B$  または  $A^T X=B$  または  $A^H X=B$  の解の精度の改善を繰り返し実行する。このルーチンは、算出されたそれぞれの解ベクトル  $x$  について、成分ごとに後退誤差  $\beta$  を計算する。次のように、 $x$  が摂動連立方程式の正確な解である場合、この誤差は、 $A$  と  $b$  の成分の最小相対摂動になる。

$$|\delta a_{ij}|/|a_{ij}| \leq \beta |a_{ij}|, |\delta b_i|/|b_i| \leq \beta |b_i| \text{ ただし } (A + \delta A)x = (b + \delta b)$$

最後に、このルーチンは、算出された解  $\|x - x_e\|_\infty / \|x\|_\infty$  の成分ごとに前進誤差を推定する ( $x_e$  は正確な解である)。

このルーチンを呼び出す前に、以下の手順を実行する必要がある。

- 因子分解ルーチン [?getrf](#) を呼び出す。
- 解の算出ルーチン [?getrs](#) を呼び出す。

### 入カパラメータ

`trans` CHARACTER\*1。'N' または 'T' または 'C' でなければならない。  
方程式の形式を指定する。

$trans = 'N'$  の場合、連立方程式の形式は  $AX = B$  である。  
 $trans = 'T'$  の場合、連立方程式の形式は  $A^T X = B$  である。  
 $trans = 'C'$  の場合、連立方程式の形式は  $A^H X = B$  である。

$n$                     INTEGER。行列  $A$  の次数 ( $n \geq 0$ )。

$nrhs$                 INTEGER。右辺の数 ( $nrhs \geq 0$ )。

$a, af, b, x, work$         REAL (sgerfs の場合 )  
                           DOUBLE PRECISION (dgerfs の場合 )  
                           COMPLEX (cgerfs の場合 )  
                           DOUBLE COMPLEX (zgerfs の場合 )。

配列 :

$a(lda, *)$  には、[?getrf](#) に入力として与えた元の行列  $A$  を格納する。  
 $af(ldaf, *)$  には、[?getrf](#) によって返された、因子分解後の行列  $A$  を格納する。  
 $b(l db, *)$  には、右辺の行列  $B$  が格納される。  
 $x(ldx, *)$  には、解の行列  $X$  が格納される。  
 $work(*)$  は、ワークスペース配列である。

$a$  と  $af$  の第 2 次元は  $\max(1, n)$  以上でなければならない。 $b$  と  $x$  の第 2 次元は  $\max(1, nrhs)$  以上でなければならない。 $work$  の次元は  $\max(1, 3 \cdot n)$  (実数型の場合) または  $\max(1, 2 \cdot n)$  (複素数の場合) 以上でなければならない。

$lda$                     INTEGER。  $a$  の第 1 次元。  $lda \geq \max(1, n)$ 。

$ldaf$                   INTEGER。  $af$  の第 1 次元。  $ldaf \geq \max(1, n)$ 。

$ldb$                     INTEGER。  $b$  の第 1 次元。  $ldb \geq \max(1, n)$ 。

$ldx$                     INTEGER。  $x$  の第 1 次元。  $ldx \geq \max(1, n)$ 。

$ipiv$                     INTEGER。  
                           配列、次元は  $\max(1, n)$  以上。  
                           [?getrf](#) によって返される  $ipiv$  配列。

$iwork$                   INTEGER。  
                           ワークスペース配列、次元は  $\max(1, n)$  以上。

$rwork$                   REAL (cgerfs の場合 )  
                           DOUBLE PRECISION (zgerfs の場合 )。  
                           ワークスペース配列、次元は  $\max(1, n)$  以上。

### 出力パラメータ

<i>x</i>	精度が改善された解の行列 <i>X</i> 。
<i>ferr</i> , <i>berr</i>	REAL ( 単精度の場合 ) DOUBLE PRECISION ( 倍精度の場合 )。 配列、次元は $\max(1, nrhs)$ 以上。各解ベクトルの成分ごとの前進誤差と後退誤差を格納する。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了した。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正である。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

gerfs ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	サイズ ( <i>n</i> , <i>n</i> ) の行列 <i>A</i> を格納する。
<i>af</i>	サイズ ( <i>n</i> , <i>n</i> ) の行列 <i>AF</i> を格納する。
<i>ipiv</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>b</i>	サイズ ( <i>n</i> , <i>nrhs</i> ) の行列 <i>B</i> を格納する。
<i>x</i>	サイズ ( <i>n</i> , <i>nrhs</i> ) の行列 <i>X</i> を格納する。
<i>ferr</i>	長さ ( <i>nrhs</i> ) のベクトルを格納する。
<i>berr</i>	長さ ( <i>nrhs</i> ) のベクトルを格納する。
<i>trans</i>	'N'、'C'、または 'T' でなければならない。デフォルト値は 'N' である。

### アプリケーション・ノート

*ferr* に返される誤差範囲は厳密ではないが、実際の誤差はほぼ常にこれより小さくなる。

各右辺について、後退誤差の計算に必要な浮動小数点演算の回数は、実数型の場合は  $4n^2$  以上、複素数型の場合は  $16n^2$  以上である。さらに、繰り返し行われる解の精度の改善の各段階で必要な演算の回数は、実数型の場合は  $6n^2$  で、複素数型の場合は  $24n^2$  である。繰り返される回数の範囲は 1 ～ 5 になる。前進誤差を推定するには、連立 1 次方



程式  $Ax = b$  を繰り返し解く必要がある。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約  $2n^2$  で、複素数型の場合は約  $8n^2$  である。

## ?gbrfs

一般帯行列による連立 1 次方程式の解の精度を改善し、解の誤差を推定する。

### 構文

#### Fortran 77:

```
call sgbtrfs(trans, n, kl, ku, nrhs, ab, ldab, afb, ldafb, ipiv, b, ldb, x, ldx,
             ferr, berr, work, iwork, info)
call dgbtrfs(trans, n, kl, ku, nrhs, ab, ldab, afb, ldafb, ipiv, b, ldb, x, ldx,
             ferr, berr, work, iwork, info)
call cgbtrfs(trans, n, kl, ku, nrhs, ab, ldab, afb, ldafb, ipiv, b, ldb, x, ldx,
             ferr, berr, work, rwork, info)
call zgbtrfs(trans, n, kl, ku, nrhs, ab, ldab, afb, ldafb, ipiv, b, ldb, x, ldx,
             ferr, berr, work, rwork, info)
```

#### Fortran 95:

```
call gbrfs(a, af, ipiv, b, x [,kl] [,trans] [,ferr] [,berr] [,info])
```

### 説明

このルーチンは、帯行列  $A$  による、複数の右辺を持つ連立 1 次方程式  $AX = B$  または  $A^T X = B$  または  $A^H X = B$  の解の精度の改善を繰り返し実行する。このルーチンは、算出されたそれぞれの解ベクトル  $x$  について、成分ごとに後退誤差  $\beta$  を計算する。次のように、 $x$  が摂動連立方程式の正確な解である場合、この誤差は、 $A$  と  $b$  の成分の最小相対摂動になる。

$$|\delta a_{ij}|/|a_{ij}| \leq \beta |a_{ij}|, |\delta b_i|/|b_i| \leq \beta |b_i| \quad \text{ただし } (A + \delta A)x = (b + \delta b)$$

最後に、このルーチンは、算出された解  $\|x - x_e\|_\infty / \|x\|_\infty$  の成分ごとに前進誤差を推定する ( $x_e$  は正確な解である)。

このルーチン呼び出す前に、以下の手順を実行する必要がある。

- 因子分解ルーチン [?gbtrf](#) を呼び出す。

- 解の算出ルーチン [?gbtrs](#) を呼び出す。

## 入力パラメータ

<i>trans</i>	CHARACTER*1。'N' または 'T' または 'C' でなければならない。 方程式の形式を指定する。 <i>trans</i> = 'N' の場合、連立方程式の形式は $AX = B$ である。 <i>trans</i> = 'T' の場合、連立方程式の形式は $A^T X = B$ である。 <i>trans</i> = 'C' の場合、連立方程式の形式は $A^H X = B$ である。
<i>n</i>	INTEGER。行列 <i>A</i> の次数 ( $n \geq 0$ )。
<i>kl</i>	INTEGER。 <i>A</i> の帯内の劣対角成分の数 ( $kl \geq 0$ )。
<i>ku</i>	INTEGER。 <i>A</i> の帯内の優対角成分の数 ( $ku \geq 0$ )。
<i>nrhs</i>	INTEGER。右辺の数 ( $nrhs \geq 0$ )。
<i>ab, afb, b, x, work</i>	REAL (sgbrfs の場合 ) DOUBLE PRECISION (dgbfrfs の場合 ) COMPLEX (cgbrfs の場合 ) DOUBLE COMPLEX (zgbrfs の場合 )。 配列： <i>ab</i> ( <i>ldab</i> , *) には、 <a href="#">?gbtrf</a> に入力として与えた元の帯行列 <i>A</i> を格納する。この行列は、行 1 ~ ( <i>kl</i> + <i>ku</i> + 1) に格納される。 <i>afb</i> ( <i>ldafb</i> , *) には、 <a href="#">?gbtrf</a> によって返された、因子分解後の帯行列 <i>A</i> を格納する。 <i>b</i> ( <i>ldb</i> , *) には、右辺の行列 <i>B</i> が格納される。 <i>x</i> ( <i>ldx</i> , *) には、解の行列 <i>X</i> が格納される。 <i>work</i> (*) は、ワークスペース配列である。 <i>ab</i> と <i>afb</i> の第 2 次元は $\max(1, n)$ 以上でなければならない。 <i>b</i> と <i>x</i> の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。 <i>work</i> の次元は $\max(1, 3*n)$ (実数型の場合) または $\max(1, 2*n)$ (複素数型の場合) 以上でなければならない。
<i>ldab</i>	INTEGER。 <i>ab</i> の第 1 次元。
<i>ldafb</i>	INTEGER。 <i>afb</i> の第 1 次元。
<i>ldb</i>	INTEGER。 <i>b</i> の第 1 次元。 $ldb \geq \max(1, n)$ 。
<i>ldx</i>	INTEGER。 <i>x</i> の第 1 次元。 $ldx \geq \max(1, n)$ 。

<i>ipiv</i>	INTEGER。 配列、次元は $\max(1, n)$ 以上。 <a href="#">zgbtrf</a> によって返される <i>ipiv</i> 配列。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, n)$ 以上。
<i>rwork</i>	REAL ( <i>cgbrfs</i> の場合 ) DOUBLE PRECISION ( <i>zgbfrfs</i> の場合 )。 ワークスペース配列、次元は $\max(1, n)$ 以上。

### 出力パラメータ

<i>x</i>	精度が改善された解の行列 <i>X</i> 。
<i>ferr</i> , <i>berr</i>	REAL ( 単精度の場合 ) DOUBLE PRECISION ( 倍精度の場合 )。 配列、次元は $\max(1, nrhs)$ 以上。各解ベクトルの成分ごとの前進誤差と後退誤差を格納する。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了した。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正である。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

*gbrfs* ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ab</i> を意味する。サイズ $(kl+ku+1, n)$ の配列 <i>A</i> を格納する。
<i>af</i>	Fortran 77 インターフェイスでの引数 <i>afb</i> を意味する。サイズ $(2*kl*ku+1, n)$ の配列 <i>AF</i> を格納する。
<i>ipiv</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>b</i>	サイズ $(n, nrhs)$ の行列 <i>B</i> を格納する。
<i>x</i>	サイズ $(n, nrhs)$ の行列 <i>X</i> を格納する。
<i>ferr</i>	長さ ( <i>nrhs</i> ) のベクトルを格納する。
<i>berr</i>	長さ ( <i>nrhs</i> ) のベクトルを格納する。

<code>trans</code>	'N'、'C'、または 'T' でなければならない。デフォルト値は 'N' である。
<code>k1</code>	省略した場合、 $k1 = ku$ とみなす。
<code>ku</code>	$ku = lda - k1 - 1$ として復元する。

### アプリケーション・ノート

`ferr` に返される誤差範囲は厳密ではないが、実際の誤差はほぼ常にこれより小さくなる。

各右辺について、後退誤差の計算に必要な浮動小数点演算の回数は、実数型の場合は  $4n(k1 + ku)$  以上、複素数型の場合は  $16n(k1 + ku)$  以上である。さらに、繰り返し行われる解の精度の改善の各段階で必要な演算の回数は、実数型の場合は  $2n(4k1 + 3ku)$  で、複素数型の場合は  $8n(4k1 + 3ku)$  である。繰り返される回数の範囲は 1 ～ 5 になる。前進誤差を推定する ... は、連立 1 次方程式  $Ax = b$  を繰り返し解く必要がある。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約  $2n^2$  で、複素数型の場合は約  $8n^2$  である。

---

## ?gtrfs

三重対角行列を係数行列とする連立 1 次方程式の解の精度を改善し、解の誤差を推定する。

---

### 構文

#### Fortran 77:

```
call sgtrfs(trans, n, nrhs, dl, d, du, dlf, df, duf, du2, ipiv, b, ldb, x, ldx,
  ferr, berr, work, iwork, info)
call dgtrfs(trans, n, nrhs, dl, d, du, dlf, df, duf, du2, ipiv, b, ldb, x, ldx,
  ferr, berr, work, iwork, info)
call cgtrfs(trans, n, nrhs, dl, d, du, dlf, df, duf, du2, ipiv, b, ldb, x, ldx,
  ferr, berr, work, rwork, info)
call zgtrfs(trans, n, nrhs, dl, d, du, dlf, df, duf, du2, ipiv, b, ldb, x, ldx,
  ferr, berr, work, rwork, info)
```

#### Fortran 95:

```
call gtrfs(dl, d, du, dlf, df, duf, du2, ipiv, b, x [,trans] [,ferr] [,berr]
  [,info])
```

## 説明

このルーチンは、三重対角行列  $A$  を係数行列とする、複数の右辺を持つ連立 1 次方程式  $AX=B$  または  $A^T X=B$  または  $A^H X=B$  の解の精度の改善を繰り返し実行する。このルーチンは、算出されたそれぞれの解ベクトル  $x$  について、成分ごとに後退誤差  $\beta$  を計算する。次のように、 $x$  が摂動連立方程式の正確な解である場合、この誤差は、 $A$  と  $b$  の成分の最小相対摂動になる。

$$|\delta a_{ij}|/|a_{ij}| \leq \beta |a_{ij}|, |\delta b_i|/|b_i| \leq \beta |b_i| \text{ ただし } (A + \delta A)x = (b + \delta b)$$

最後に、このルーチンは、算出された解  $\|x - x_e\|_\infty / \|x\|_\infty$  の成分ごとに前進誤差を推定する ( $x_e$  は正確な解である)。

このルーチン呼び出す前に、以下の手順を実行する必要がある。

- 因子分解ルーチン [?gtrf](#) を呼び出す。
- 解の算出ルーチン [?gtrs](#) を呼び出す。

## 入力パラメータ

*trans* CHARACTER\*1。'N' または 'T' または 'C' でなければならない。  
方程式の形式を指定する。  
*trans* = 'N' の場合、連立方程式の形式は  $AX=B$  である。  
*trans* = 'T' の場合、連立方程式の形式は  $A^T X=B$  である。  
*trans* = 'C' の場合、連立方程式の形式は  $A^H X=B$  である。

*n* INTEGER。行列  $A$  の次数 ( $n \geq 0$ )。

*nrhs* INTEGER。右辺の数、すなわち、行列  $B$  の列数 ( $nrhs \geq 0$ )。

*d1,d,du,d1f,df,duf,du2, b, x, work*  
REAL (sgtrfs の場合)  
DOUBLE PRECISION (dgtrfs の場合)  
COMPLEX (cgtrfs の場合)  
DOUBLE COMPLEX (zgtrfs の場合)。  
配列:  
*d1*、次元は  $(n-1)$ 。 $A$  の劣対角成分を格納する。  
*d*、次元は  $(n)$ 。 $A$  の対角成分を格納する。  
*du*、次元は  $(n-1)$ 。 $A$  の優対角成分を格納する。  
*d1f*、次元は  $(n-1)$  で、[?gtrf](#) による  $A$  の  $LU$  因子分解で得られた行列  $L$  を定義する  $(n-1)$  個の乗数を格納する。  
*df*、次元は  $(n)$  で、 $A$  の  $LU$  因子分解で得られた上三角行列  $U$  の  $n$  個の対角成分を格納する。

*duf*、次元は  $(n-1)$  で、 $U$  の最初の優対角成分の  $(n-1)$  個の成分を格納する。

*du2*、次元は  $(n-2)$  で、 $U$  の 2 番目の優対角成分の  $(n-2)$  個の成分を格納する。

*b(ldb, nrhs)* は、右辺の行列  $B$  を格納する。

*x(ldx, nrhs)* は、[?gttrs](#) によって計算された解の行列  $X$  を格納する。

*work(\*)* はワークスペース配列である。

*work* の次元は  $\max(1, 3*n)$  (実数型の場合) または  $\max(1, 2*n)$  (複素数型の場合) 以上でなければならない。

*ldb* INTEGER。 *b* の第 1 次元。  $ldb \geq \max(1, n)$ 。

*ldx* INTEGER。 *x* の第 1 次元。  $ldx \geq \max(1, n)$ 。

*ipiv* INTEGER。  
配列、次元は  $\max(1, n)$  以上。  
[?gttrf](#) によって返される *ipiv* 配列。

*iwork* INTEGER。  
ワークスペース配列、次元は  $(n)$ 。実数型でのみ使用される。

*rwork* REAL (cgtrfs の場合)  
DOUBLE PRECISION (zgtrfs の場合)。  
ワークスペース配列、次元は  $(n)$ 。複素数型でのみ使用される。

## 出力パラメータ

*x* 精度が改善された解の行列  $X$ 。

*ferr, berr* REAL (単精度の場合)  
DOUBLE PRECISION (倍精度の場合)。  
配列、次元は  $\max(1, nrhs)$  以上。各解ベクトルの成分ごとの前進誤差と後退誤差を格納する。

*info* INTEGER。  
*info* = 0 の場合、実行は正常に終了した。  
*info* = -*i* の場合、*i* 番目のパラメータの値が不正である。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

gtrfs ルーチンのインターフェイス特有の詳細を以下に示す。

<i>dl</i>	長さ ( $n-1$ ) のベクトルを格納する。
<i>d</i>	長さ ( $n$ ) のベクトルを格納する。
<i>du</i>	長さ ( $n-1$ ) のベクトルを格納する。
<i>dlf</i>	長さ ( $n-1$ ) のベクトルを格納する。
<i>df</i>	長さ ( $n$ ) のベクトルを格納する。
<i>duf</i>	長さ ( $n-1$ ) のベクトルを格納する。
<i>du2</i>	長さ ( $n-2$ ) のベクトルを格納する。
<i>ipiv</i>	長さ ( $n$ ) のベクトルを格納する。
<i>b</i>	サイズ ( $n, nrhs$ ) の行列 $B$ を格納する。
<i>x</i>	サイズ ( $n, nrhs$ ) の行列 $X$ を格納する。
<i>ferr</i>	長さ ( $nrhs$ ) のベクトルを格納する。
<i>berr</i>	長さ ( $nrhs$ ) のベクトルを格納する。
<i>trans</i>	'N'、'C'、または 'T' でなければならない。デフォルト値は 'N' である。

## ?porfs

対称(エルミート)正定値行列による連立1次方程式の解の精度を改善し、解の誤差を推定する。

### 構文

#### Fortran 77:

```
call sporfs(uplo, n, nrhs, a, lda, af, ldaf, b, ldb, x, ldx, ferr, berr, work,
            iwork, info)
call dporfs(uplo, n, nrhs, a, lda, af, ldaf, b, ldb, x, ldx, ferr, berr, work,
            iwork, info)
call cporfs(uplo, n, nrhs, a, lda, af, ldaf, b, ldb, x, ldx, ferr, berr, work,
            rwork, info)
call zporfs(uplo, n, nrhs, a, lda, af, ldaf, b, ldb, x, ldx, ferr, berr, work,
            rwork, info)
```

## Fortran 95:

```
call porfs(a, af, b, x [,uplo] [,ferr] [,berr] [,info])
```

### 説明

このルーチンは、対称 (エルミート) 正定値行列  $A$  による、複数の右辺を持つ連立 1 次方程式  $AX=B$  の解の精度の改善を繰り返し実行する。このルーチンは、算出されたそれぞれの解ベクトル  $x$  について、成分ごとに後退誤差  $\beta$  を計算する。次のように、 $x$  が摂動連立方程式の正確な解である場合、この誤差は、 $A$  と  $b$  の成分の最小相対摂動になる。

$$|\delta a_{ij}|/|a_{ij}| \leq \beta |a_{ij}|, |\delta b_i|/|b_i| \leq \beta |b_i| \quad \text{ただし } (A + \delta A)x = (b + \delta b)$$

最後に、このルーチンは、算出された解  $\|x - x_e\|_\infty / \|x\|_\infty$  の成分ごとに前進誤差を推定する ( $x_e$  は正確な解である)。

このルーチンと呼び出す前に、以下の手順を実行する必要がある。

- 因子分解ルーチン [?potrf](#) を呼び出す。
- 解の算出ルーチン [?potrs](#) を呼び出す。

### 入力パラメータ

**uplo** CHARACTER\*1。'U' または 'L' でなければならない。  
 入力行列  $A$  の因子分解の方法を指定する。  
 uplo = 'U' の場合、配列 af には、コレスキー因子分解  $A = U^H U$  の係数  $U$  を格納する。  
 uplo = 'L' の場合、配列 af には、コレスキー因子分解  $A = LL^H$  の係数  $L$  を格納する。

**n** INTEGER。行列  $A$  の次数 ( $n \geq 0$ )。

**nrhs** INTEGER。右辺の数 ( $nrhs \geq 0$ )。

**a, af, b, x, work**  
 REAL (sporfs の場合)  
 DOUBLE PRECISION (dporfs の場合)  
 COMPLEX (cporfs の場合)  
 DOUBLE COMPLEX (zporfs の場合)。  
 配列:  
 a(lda, \*) には、[?potrf](#) に入力として与えた元の行列  $A$  を格納する。  
 af(ldaf, \*) には、[?potrf](#) によって返された、因子分解後の行列  $A$  を格納する。  
 b ldb, \*) には、右辺の行列  $B$  が格納される。



$x(ldx, *)$  には、解の行列  $X$  が格納される。

$work(*)$  は、ワークスペース配列である。

$a$  と  $af$  の第 2 次元は  $\max(1, n)$  以上でなければならない。 $b$  と  $x$  の第 2 次元は  $\max(1, nrhs)$  以上でなければならない。 $work$  の次元は  $\max(1, 3*n)$  (実数型の場合) または  $\max(1, 2*n)$  (複素数の場合) 以上でなければならない。

$lda$             INTEGER。  $a$  の第 1 次元。  $lda \geq \max(1, n)$ 。  
 $ldaf$             INTEGER。  $af$  の第 1 次元。  $ldaf \geq \max(1, n)$ 。  
 $ldb$             INTEGER。  $b$  の第 1 次元。  $ldb \geq \max(1, n)$ 。  
 $ldx$             INTEGER。  $x$  の第 1 次元。  $ldx \geq \max(1, n)$ 。  
 $iwork$             INTEGER。  
                   ワークスペース配列、次元は  $\max(1, n)$  以上。  
 $rwork$             REAL (cporfs の場合)  
                   DOUBLE PRECISION (zporfs の場合)。  
                   ワークスペース配列、次元は  $\max(1, n)$  以上。

## 出力パラメータ

$x$                 精度が改善された解の行列  $X$ 。  
 $ferr, berr$         REAL (単精度の場合)  
                   DOUBLE PRECISION (倍精度の場合)。  
                   配列、次元は  $\max(1, nrhs)$  以上。各解ベクトルの成分ごとの前進誤差  
                   と後退誤差を格納する。  
 $info$             INTEGER。  
                    $info = 0$  の場合、実行は正常に終了した。  
                    $info = -i$  の場合、 $i$  番目のパラメータの値が不正である。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

porfs ルーチンのインターフェイス特有の詳細を以下に示す。

$a$                 サイズ  $(n, n)$  の行列  $A$  を格納する。  
 $af$               サイズ  $(n, n)$  の行列  $AF$  を格納する。  
 $b$                 サイズ  $(n, nrhs)$  の行列  $B$  を格納する。

<i>x</i>	サイズ ( <i>n,nrhs</i> ) の行列 <i>X</i> を格納する。
<i>ferr</i>	長さ ( <i>nrhs</i> ) のベクトルを格納する。
<i>berr</i>	長さ ( <i>nrhs</i> ) のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U' である。

### アプリケーション・ノート

*ferr* に返される誤差範囲は厳密ではないが、実際の誤差はほぼ常にこれより小さくなる。

各右辺について、後退誤差の計算に必要な浮動小数点演算の回数は、実数型の場合は  $4n^2$  以上、複素数型の場合は  $16n^2$  以上である。さらに、繰り返し行われる解の精度の改善の各段階に必要な演算の回数は、実数型の場合は  $6n^2$  で、複素数型の場合は  $24n^2$  である。繰り返される回数の範囲は 1 ～ 5 になる。前進誤差を推定するには、連立 1 次方程式  $Ax = b$  を繰り返し解く必要がある。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約  $2n^2$  で、複素数型の場合は約  $8n^2$  である。

---

## ?pprfs

圧縮形式の対称 (エルミート) 正定値行列による  
連立 1 次方程式の解の精度を改善し、解の誤差を  
推定する。

---

### 構文

#### Fortran 77:

```
call spprfs(uplo, n, nrhs, ap, afp, b, ldb, x, ldx, ferr, berr, work, iwork, info)
call dpprfs(uplo, n, nrhs, ap, afp, b, ldb, x, ldx, ferr, berr, work, iwork, info)
call cpprfs(uplo, n, nrhs, ap, afp, b, ldb, x, ldx, ferr, berr, work, rwork, info)
call zpprfs(uplo, n, nrhs, ap, afp, b, ldb, x, ldx, ferr, berr, work, rwork, info)
```

#### Fortran 95:

```
call pprfs(a, af, b, x [,uplo] [,ferr] [,berr] [,info])
```

## 説明

このルーチンは、圧縮形式の対称 (エルミート) 正定値行列  $A$  による、複数の右辺を持つ連立 1 次方程式  $AX = B$  の解の精度の改善を繰り返し実行する。このルーチンは、算出されたそれぞれの解ベクトル  $x$  について、成分ごとに後退誤差  $\beta$  を計算する。次のように、 $x$  が摂動連立方程式の正確な解である場合、この誤差は、 $A$  と  $b$  の成分の最小相対摂動になる。

$$|\delta a_{ij}|/|a_{ij}| \leq \beta |a_{ij}|, |\delta b_i|/|b_i| \leq \beta |b_i| \quad \text{ただし } (A + \delta A)x = (b + \delta b)$$

最後に、このルーチンは、算出された解  $\|x - x_e\|_\infty / \|x\|_\infty$  の成分ごとに前進誤差を推定する ( $x_e$  は正確な解である)。

このルーチン呼び出す前に、以下の手順を実行する必要がある。

- 因子分解ルーチン [?pptrf](#) を呼び出す。
- 解の算出ルーチン [?pptrs](#) を呼び出す。

## 入力パラメータ

**uplo** CHARACTER\*1。'U' または 'L' でなければならない。  
 入力行列  $A$  の因子分解の方法を指定する。  
 uplo = 'U' の場合、配列 **afp** には、コレスキー因子分解  $A = U^H U$  の圧縮形式の係数  $U$  を格納する。  
 uplo = 'L' の場合、配列 **afp** には、コレスキー因子分解  $A = LL^H$  の圧縮形式の係数  $L$  を格納する。

**n** INTEGER。行列  $A$  の次数 ( $n \geq 0$ )。

**nrhs** INTEGER。右辺の数 ( $nrhs \geq 0$ )。

**ap, afp, b, x, work**  
 REAL (spprfs の場合)  
 DOUBLE PRECISION (dpprfs の場合)  
 COMPLEX (cpprfs の場合)  
 DOUBLE COMPLEX (zpprfs の場合)。  
 配列:  
**ap**(\*) には、[?pptrf](#) に入力として与えた元の圧縮形式の行列  $A$  を格納する。  
**afp**(\*) には、[?pptrf](#) によって返された、因子分解後の圧縮形式の行列  $A$  を格納する。  
**b**(ldb, \*) には、右辺の行列  $B$  が格納される。  
**x**(ldx, \*) には、解の行列  $X$  が格納される。  
**work**(\*) は、ワークスペース配列である。

配列  $ap$  と  $afp$  の次元は  $\max(1, n(n+1)/2)$  以上でなければならない。 $b$  と  $x$  の第 2 次元は  $\max(1, nrhs)$  以上でなければならない。 $work$  の次元は  $\max(1, 3*n)$  (実数型の場合) または  $\max(1, 2*n)$  (複素数型の場合) 以上でなければならない。

<i>ldb</i>	INTEGER。 $b$ の第 1 次元。 $ldb \geq \max(1, n)$ 。
<i>ldx</i>	INTEGER。 $x$ の第 1 次元。 $ldx \geq \max(1, n)$ 。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, n)$ 以上。
<i>rwork</i>	REAL (cpprfs の場合) DOUBLE PRECISION (zpprfs の場合)。 ワークスペース配列、次元は $\max(1, n)$ 以上。

### 出力パラメータ

<i>x</i>	精度が改善された解の行列 $X$ 。
<i>ferr, berr</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 配列、次元は $\max(1, nrhs)$ 以上。各解ベクトルの成分ごとの前進誤差と後退誤差を格納する。
<i>info</i>	INTEGER。 $info = 0$ の場合、実行は正常に終了した。 $info = -i$ の場合、 $i$ 番目のパラメータの値が不正である。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

pprfs ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 $ap$ を意味する。サイズ $(n*(n+1)/2)$ の配列 $A$ を格納する。
<i>af</i>	Fortran 77 インターフェイスでの引数 $apf$ を意味する。サイズ $(n*(n+1)/2)$ の配列 $AF$ を格納する。
<i>b</i>	サイズ $(n, nrhs)$ の行列 $B$ を格納する。
<i>x</i>	サイズ $(n, nrhs)$ の行列 $X$ を格納する。
<i>ferr</i>	長さ $(nrhs)$ のベクトルを格納する。
<i>berr</i>	長さ $(nrhs)$ のベクトルを格納する。

`uplo` 'U' または 'L' でなければならない。デフォルト値は 'U' である。

### アプリケーション・ノート

`ferr` に返される誤差範囲は厳密ではないが、実際の誤差はほぼ常にこれより小さくなる。

各右辺について、後退誤差の計算に必要な浮動小数点演算の回数は、実数型の場合は  $4n^2$  以上、複素数型の場合は  $16n^2$  以上である。さらに、繰り返し行われる解の精度の改善の各段階で必要な演算の回数は、実数型の場合は  $6n^2$  で、複素数型の場合は  $24n^2$  である。繰り返される回数の範囲は 1 ~ 5 になる。

前進誤差を推定するには、連立 1 次方程式  $Ax = b$  を繰り返し解く必要がある。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約  $2n^2$  で、複素数型の場合は約  $8n^2$  である。

---

## ?pbrfs

対称(エルミート)正定値帯行列による連立 1 次方程式の解の精度を改善し、解の誤差を推定する。

---

### 構文

#### Fortran 77:

```
call spbrfs(uplo, n, kd, nrhs, ab, ldab, afb, ldafb, b, ldb, x, ldx, ferr, berr,
            work, iwork, info)
call dpbrfs(uplo, n, kd, nrhs, ab, ldab, afb, ldafb, b, ldb, x, ldx, ferr, berr,
            work, iwork, info)
call cpbrfs(uplo, n, kd, nrhs, ab, ldab, afb, ldafb, b, ldb, x, ldx, ferr, berr,
            work, rwork, info)
call zpbrfs(uplo, n, kd, nrhs, ab, ldab, afb, ldafb, b, ldb, x, ldx, ferr, berr,
            work, rwork, info)
```

#### Fortran 95:

```
call pbrfs(a, af, b, x [,uplo] [,ferr] [,berr] [,info])
```

## 説明

このルーチンは、対称 (エルミート) 正定値帯行列  $A$  による、複数の右辺を持つ連立 1 次方程式  $AX = B$  の解の精度の改善を繰り返し実行する。このルーチンは、算出されたそれぞれの解ベクトル  $x$  について、成分ごとに後退誤差  $\beta$  を計算する。次のように、 $x$  が摂動連立方程式の正確な解である場合、この誤差は、 $A$  と  $b$  の成分の最小相対摂動になる。

$$|\delta a_{ij}|/|a_{ij}| \leq \beta |a_{ij}|, |\delta b_i|/|b_i| \leq \beta |b_i| \quad \text{ただし } (A + \delta A)x = (b + \delta b)$$

最後に、このルーチンは、算出された解  $\|x - x_e\|_\infty / \|x\|_\infty$  の成分ごとに前進誤差を推定する ( $x_e$  は正確な解である)。

このルーチン呼び出す前に、以下の手順を実行する必要がある。

- 因子分解ルーチン [?pbturf](#) を呼び出す。
- 解の算出ルーチン [?pbtrs](#) を呼び出す。

## 入力パラメータ

**uplo** CHARACTER\*1。'U' または 'L' でなければならない。  
 入力行列  $A$  の因子分解の方法を指定する。  
 uplo = 'U' の場合、配列 **afb** には、コレスキー因子分解  $A = U^H U$  の係数  $U$  を格納する。  
 uplo = 'L' の場合、配列 **afb** には、コレスキー因子分解  $A = LL^H$  の係数  $L$  を格納する。

**n** INTEGER。行列  $A$  の次数 ( $n \geq 0$ )。

**kd** INTEGER。  $A$  の優対角成分または劣対角成分の数 ( $kd \geq 0$ )。

**nrhs** INTEGER。右辺の数 ( $nrhs \geq 0$ )。

**ab, afb, b, x, work**  
 REAL (spbrfs の場合)  
 DOUBLE PRECISION (dpbrfs の場合)  
 COMPLEX (cpbrfs の場合)  
 DOUBLE COMPLEX (zpbfrfs の場合)。  
 配列:  
**ab**(**ldab**, \*) には、[?pbturf](#) に入力として与えた元の帯行列  $A$  を格納する。  
**afb**(**ldafb**, \*) には、[?pbturf](#) によって返された、因子分解後の帯行列  $A$  を格納する。  
**b**(**ldb**, \*) には、右辺の行列  $B$  が格納される。

$x(ldx, *)$  には、解の行列  $X$  が格納される。

$work(*)$  は、ワークスペース配列である。

$ab$  と  $afb$  の第 2 次元は  $\max(1, n)$  以上でなければならない。 $b$  と  $x$  の第 2 次元は  $\max(1, nrhs)$  以上でなければならない。 $work$  の次元は  $\max(1, 3*n)$  (実数型の場合) または  $\max(1, 2*n)$  (複素数型の場合) 以上でなければならない。

<i>ldab</i>	INTEGER。 $ab$ の第 1 次元。 $ldab \geq kd + 1$ 。
<i>ldafb</i>	INTEGER。 $fb$ の第 1 次元。 $ldafb \geq kd + 1$ 。
<i>ldb</i>	INTEGER。 $b$ の第 1 次元。 $ldb \geq \max(1, n)$ 。
<i>ldx</i>	INTEGER。 $x$ の第 1 次元。 $ldx \geq \max(1, n)$ 。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, n)$ 以上。
<i>rwork</i>	REAL (cpbrfs の場合) DOUBLE PRECISION (zpbrfs の場合)。 ワークスペース配列、次元は $\max(1, n)$ 以上。

## 出力パラメータ

<i>x</i>	精度が改善された解の行列 $X$ 。
<i>ferr, berr</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 配列、次元は $\max(1, nrhs)$ 以上。各解ベクトルの成分ごとの前進誤差と後退誤差を格納する。
<i>info</i>	INTEGER。 $info = 0$ の場合、実行は正常に終了した。 $info = -i$ の場合、 $i$ 番目のパラメータの値が不正である。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

pbrfs ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 $ab$ を意味する。サイズ $(kd+1, n)$ の配列 $A$ を格納する。
<i>af</i>	Fortran 77 インターフェイスでの引数 $afb$ を意味する。サイズ $(kd+1, n)$ の配列 $AF$ を格納する。

<i>b</i>	サイズ ( <i>n,nrhs</i> ) の行列 <i>B</i> を格納する。
<i>x</i>	サイズ ( <i>n,nrhs</i> ) の行列 <i>X</i> を格納する。
<i>ferr</i>	長さ ( <i>nrhs</i> ) のベクトルを格納する。
<i>berr</i>	長さ ( <i>nrhs</i> ) のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U' である。

### アプリケーション・ノート

*ferr* に返される誤差範囲は厳密ではないが、実際の誤差はほぼ常にこれより小さくなる。

各右辺について、後退誤差の計算に必要な浮動小数点演算の回数は、実数型の場合は  $8n \cdot kd$  以上、複素数型の場合は  $32n \cdot kd$  以上である。さらに、繰り返し行われる解の精度の改善の各段階で必要な演算の回数は、実数型の場合は  $12n \cdot kd$  で、複素数型の場合は  $48n \cdot kd$  である。繰り返される回数の範囲は 1 ～ 5 になる。

前進誤差を推定するには、連立 1 次方程式  $Ax = b$  を繰り返し解く必要がある。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約  $4n \cdot kd$  で、複素数型の場合は約  $16n \cdot kd$  である。

---

## ?ptrfs

対称(エルミート)正定値三重対角行列を係数行列とする連立 1 次方程式の解の精度を改善し、解の誤差を推定する。

---

### 構文

#### Fortran 77:

```
call sptfrfs(n, nrhs, d, e, df, ef, b, ldb, x, ldx, ferr, berr, work, info)
call dptfrfs(n, nrhs, d, e, df, ef, b, ldb, x, ldx, ferr, berr, work, info)
call cptfrfs(uplo, n, nrhs, d, e, df, ef, b, ldb, x, ldx, ferr, berr, work,
             rwork, info)
call zptfrfs(uplo, n, nrhs, d, e, df, ef, b, ldb, x, ldx, ferr, berr, work,
             rwork, info)
```



**Fortran 95:**

```
call ptrfs(d, df, e, ef, b, x [,ferr] [,berr] [,info])
call ptrfs(d, df, e, ef, b, x [,uplo] [,ferr] [,berr] [,info])
```

**説明**

このルーチンは、対称 (エルミート) 正定値三重対角行列  $A$  を係数行列とし複数の右辺を持つ連立 1 次方程式  $AX=B$  の解の精度の改善を繰り返し実行する。このルーチンは、算出されたそれぞれの解ベクトル  $x$  について、成分ごとに後退誤差  $\beta$  を計算する。次のように、 $x$  が摂動連立方程式の正確な解である場合、この誤差は、 $A$  と  $b$  の成分の最小相対摂動になる。

$$|\delta a_{ij}|/|a_{ij}| \leq \beta |a_{ij}|, |\delta b_i|/|b_i| \leq \beta |b_i| \text{ ただし } (A + \delta A)x = (b + \delta b)$$

最後に、このルーチンは、算出された解  $\|x - x_e\|_\infty / \|x\|_\infty$  の成分ごとに前進誤差を推定する ( $x_e$  は正確な解である)。

このルーチン呼び出す前に、以下の手順を実行する必要がある。

- 因子分解ルーチン [?pttrf](#) を呼び出す。
- 解の算出ルーチン [?pttrs](#) を呼び出す。

**入力パラメータ**

**uplo** CHARACTER\*1. 複素数型でのみ使用される。  
'U' または 'L' でなければならない。

三重対角行列  $A$  の優対角成分と劣対角成分のどちらを格納するかと、 $A$  の因子分解の方法を指定する。  
 $uplo = 'U'$  の場合、配列  $e$  には  $A$  の優対角成分が格納され、 $A$  は  $U^H D U$  として因子分解される。  
 $uplo = 'L'$  の場合、配列  $e$  には  $A$  の劣対角成分が格納され、 $A$  は  $LDL^H$  として因子分解される。

**n** INTEGER. 行列  $A$  の次数 ( $n \geq 0$ )。

**nrhs** INTEGER. 右辺の数 ( $nrhs \geq 0$ )。

**d, df, rwork** REAL (単精度の場合)  
DOUBLE PRECISION (倍精度の場合)。  
配列:  $d(n)$ ,  $df(n)$ ,  $rwork(n)$ 。  
配列  $d$  には、三重対角行列  $A$  の  $n$  個の対角成分が格納される。  
配列  $df$  には、[?pttrf](#) による  $A$  の因子分解で得られた対角行列  $D$  の  $n$  個の対角成分が格納される。  
配列  $rwork$  は、複素数型でのみ使用されるワークスペース配列である。

*e, ef, b, x, work*

REAL (spttrfs の場合)

DOUBLE PRECISION (dpttrfs の場合)

COMPLEX (cpttrfs の場合)

DOUBLE COMPLEX (zpttrfs の場合)。

配列: *e*(*n* - 1), *ef*(*n* - 1), *b*(*ldb*, *nrhs*), *x*(*ldx*, *nrhs*), *work*(\*)。

配列 *e* には、三重対角行列 *A* の (*n* - 1) 個の非対角成分が格納される (*uplo* を参照)。

配列 *ef* には、[?pttrf](#) による因子分解で得られた単位二重対角係数 *U* または *L* の (*n* - 1) 個の非対角成分が格納される (*uplo* を参照)。

配列 *b* には、行列 *B* が格納される。この行列の列は、連立方程式の右辺である。

配列 *x* には、[?pttrs](#) によって計算された解の行列 *X* が格納される。

配列 *work* はワークスペース配列である。*work* の次元は、2 \* *n* (実数型の場合) または *n* (複素数型の場合) 以上でなければならない。

*ldb* INTEGER。 *b* のリーディング・ディメンジョン。 *ldb* ≥ max(1, *n*)。

*ldx* INTEGER。 *x* のリーディング・ディメンジョン。 *ldx* ≥ max(1, *n*)。

#### 出力パラメータ

*x* 精度が改善された解の行列 *X*。

*ferr, berr* REAL (単精度の場合)

DOUBLE PRECISION (倍精度の場合)。

配列、次元は max(1, *nrhs*) 以上。各解ベクトルの成分ごとの前進誤差と後退誤差を格納する。

*info* INTEGER。

*info* = 0 の場合、実行は正常に終了した。

*info* = -*i* の場合、*i* 番目のパラメータの値が不正である。

#### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

pttrfs ルーチンのインターフェイス特有の詳細を以下に示す。

*d* 長さ (*n*) のベクトルを格納する。

*df* 長さ (*n*) のベクトルを格納する。

*e* 長さ (*n* - 1) のベクトルを格納する。

<i>ef</i>	長さ ( <i>n</i> -1) のベクトルを格納する。
<i>b</i>	サイズ ( <i>n</i> , <i>nrhs</i> ) の行列 <i>B</i> を格納する。
<i>x</i>	サイズ ( <i>n</i> , <i>nrhs</i> ) の行列 <i>X</i> を格納する。
<i>ferr</i>	長さ ( <i>nrhs</i> ) のベクトルを格納する。
<i>berr</i>	長さ ( <i>nrhs</i> ) のベクトルを格納する。
<i>uplo</i>	複素型でのみ使用される。'U' または 'L' でなければならない。デフォルト値は 'U' である。

## ?syrrfs

対称行列による連立 1 次方程式の解の精度を改善し、解の誤差を推定する。

### 構文

#### Fortran 77:

```
call ssyrrfs(uplo, n, nrhs, a, lda, af, ldaf, ipiv, b, ldb, x, ldx, ferr, berr,
             work, iwork, info)
call dsyrrfs(uplo, n, nrhs, a, lda, af, ldaf, ipiv, b, ldb, x, ldx, ferr, berr,
             work, iwork, info)
call csyrrfs(uplo, n, nrhs, a, lda, af, ldaf, ipiv, b, ldb, x, ldx, ferr, berr,
             work, rwork, info)
call zsyrrfs(uplo, n, nrhs, a, lda, af, ldaf, ipiv, b, ldb, x, ldx, ferr, berr,
             work, rwork, info)
```

#### Fortran 95:

```
call syrrfs(a, af, ipiv, b, x [,uplo] [,ferr] [,berr] [,info])
```

### 説明

このルーチンは、フル格納形式の対称行列 *A* による、複数の右辺を持つ連立 1 次方程式  $AX=B$  の解の精度の改善を繰り返し実行する。このルーチンは、算出されたそれぞれの解ベクトル *x* について、成分ごとに後退誤差  $\beta$  を計算する。次のように、*x* が摂動連立方程式の正確な解である場合、この誤差は、*A* と *b* の成分の最小相対摂動になる。

$$|\delta a_{ij}|/|a_{ij}| \leq \beta |a_{ij}|, |\delta b_i|/|b_i| \leq \beta |b_i| \quad \text{ただし } (A + \delta A)x = (b + \delta b)$$

最後に、このルーチンは、算出された解  $\|x - x_e\|_\infty / \|x\|_\infty$  の成分ごとに前進誤差を推定する ( $x_e$  は正確な解である)。

このルーチン呼び出す前に、以下の手順を実行する必要がある。

- 因子分解ルーチン [?sytrf](#) を呼び出す。
- 解の算出ルーチン [?sytrs](#) を呼び出す。

## 入力パラメータ

<code>uplo</code>	CHARACTER*1。'U' または 'L' でなければならない。 入力行列 $A$ の因子分解の方法を指定する。 <code>uplo = 'U'</code> の場合、配列 <code>af</code> には、Bunch-Kaufman 因子分解 $A = PUDU^T P^T$ を格納する。 <code>uplo = 'L'</code> の場合、配列 <code>af</code> には、Bunch-Kaufman 因子分解 $A = PLDL^T P^T$ を格納する。
<code>n</code>	INTEGER。行列 $A$ の次数 ( $n \geq 0$ )。
<code>nrhs</code>	INTEGER。右辺の数 ( $nrhs \geq 0$ )。
<code>a, af, b, x, work</code>	REAL(ssyrfs の場合) DOUBLE PRECISION(dsyrfs の場合) COMPLEX(csyrfs の場合) DOUBLE COMPLEX(zsyrfs の場合)。  配列:  <code>a(lda, *)</code> には、 <a href="#">?sytrf</a> に入力として与えた元の行列 $A$ を格納する。 <code>af(ldaf, *)</code> には、 <a href="#">?sytrf</a> によって返された、因子分解後の行列 $A$ を格納する。 <code>b(ldb, *)</code> には、右辺の行列 $B$ が格納される。 <code>x(ldx, *)</code> には、解の行列 $X$ が格納される。 <code>work(*)</code> は、ワークスペース配列である。  <code>a</code> と <code>af</code> の第 2 次元は $\max(1, n)$ 以上でなければならない。 <code>b</code> と <code>x</code> の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。 <code>work</code> の次元は $\max(1, 3*n)$ (実数型の場合) または $\max(1, 2*n)$ (複素数の場合) 以上でなければならない。
<code>lda</code>	INTEGER。 <code>a</code> の第 1 次元。 $lda \geq \max(1, n)$ 。
<code>ldaf</code>	INTEGER。 <code>af</code> の第 1 次元。 $ldaf \geq \max(1, n)$ 。
<code>ldb</code>	INTEGER。 <code>b</code> の第 1 次元。 $ldb \geq \max(1, n)$ 。
<code>ldx</code>	INTEGER。 <code>x</code> の第 1 次元。 $ldx \geq \max(1, n)$ 。

<i>ipiv</i>	INTEGER。 配列、次元は $\max(1, n)$ 以上。 <a href="#">?sytrf</a> によって返される <i>ipiv</i> 配列。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, n)$ 以上。
<i>rwork</i>	REAL ( <i>csyrfs</i> の場合 ) DOUBLE PRECISION ( <i>zsyrfs</i> の場合 )。 ワークスペース配列、次元は $\max(1, n)$ 以上。

### 出力パラメータ

<i>x</i>	精度が改善された解の行列 <i>X</i> 。
<i>ferr</i> , <i>berr</i>	REAL ( 単精度の場合 ) DOUBLE PRECISION ( 倍精度の場合 )。 配列、次元は $\max(1, nrhs)$ 以上。各解ベクトルの成分ごとの前進誤差と後退誤差を格納する。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了した。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正である。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

*syrrfs* ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	サイズ $(n, n)$ の行列 <i>A</i> を格納する。
<i>af</i>	サイズ $(n, n)$ の行列 <i>AF</i> を格納する。
<i>ipiv</i>	長さ $(n)$ のベクトルを格納する。
<i>b</i>	サイズ $(n, nrhs)$ の行列 <i>B</i> を格納する。
<i>x</i>	サイズ $(n, nrhs)$ の行列 <i>X</i> を格納する。
<i>ferr</i>	長さ $(nrhs)$ のベクトルを格納する。
<i>berr</i>	長さ $(nrhs)$ のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U' である。

### アプリケーション・ノート

$ferr$  に返される誤差範囲は厳密ではないが、実際の誤差はほぼ常にこれより小さくなる。

各右辺について、後退誤差の計算に必要な浮動小数点演算の回数は、実数型の場合は  $4n^2$  以上、複素数型の場合は  $16n^2$  以上である。さらに、繰り返し行われる解の精度の改善の各段階で必要な演算の回数は、実数型の場合は  $6n^2$  で、複素数型の場合は  $24n^2$  である。繰り返される回数の範囲は 1 ~ 5 になる。前進誤差を推定するには、連立 1 次方程式  $Ax = b$  を繰り返し解く必要がある。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約  $2n^2$  で、複素数型の場合は約  $8n^2$  である。

---

## ?herfs

複素エルミート行列による連立 1 次方程式の解の精度を改善し、解の誤差を推定する。

---

### 構文

#### Fortran 77:

```
call cherfs(uplo, n, nrhs, a, lda, af, ldaf, ipiv, b, ldb, x, ldx, ferr, berr,  
            work, rwork, info)  
call zherfs(uplo, n, nrhs, a, lda, af, ldaf, ipiv, b, ldb, x, ldx, ferr, berr,  
            work, rwork, info)
```

#### Fortran 95:

```
call herfs(a, af, ipiv, b, x [,uplo] [,ferr] [,berr] [,info])
```

### 説明

このルーチンは、フル格納形式の複素エルミート行列  $A$  による、複数の右辺を持つ連立 1 次方程式  $AX = B$  の解の精度の改善を繰り返し実行する。このルーチンは、算出されたそれぞれの解ベクトル  $x$  について、成分ごとに後退誤差  $\beta$  を計算する。次のように、 $x$  が摂動連立方程式の正確な解である場合、この誤差は、 $A$  と  $b$  の成分の最小相対摂動になる。

$$|\delta a_{ij}|/|a_{ij}| \leq \beta |a_{ij}|, |\delta b_i|/|b_i| \leq \beta |b_i| \quad \text{ただし } (A + \delta A)x = (b + \delta b)$$

最後に、このルーチンは、算出された解  $\|x - x_e\|_\infty / \|x\|_\infty$  の成分ごとに前進誤差を推定する ( $x_e$  は正確な解である)。

このルーチンを呼び出す前に、以下の手順を実行する必要がある。

- 因子分解ルーチン [?hetrf](#) を呼び出す。
- 解の算出ルーチン [?hetrs](#) を呼び出す。

## 入力パラメータ

*uplo* CHARACTER\*1。'U' または 'L' でなければならない。  
 入力行列  $A$  の因子分解の方法を指定する。  
*uplo* = 'U' の場合、配列 *af* には、Bunch-Kaufman 因子分解  $A = PUDU^H P^T$  を格納する。  
*uplo* = 'L' の場合、配列 *af* には、Bunch-Kaufman 因子分解  $A = PLDL^H P^T$  を格納する。

*n* INTEGER。行列  $A$  の次数 ( $n \geq 0$ )。

*nrhs* INTEGER。右辺の数 ( $nrhs \geq 0$ )。

*a*, *af*, *b*, *x*, *work*  
 COMPLEX (cherfs の場合)  
 DOUBLE COMPLEX (zherfs の場合)。  
 配列:  
*a*(*lda*, \*) には、[?hetrf](#) に入力として与えた元の行列  $A$  を格納する。  
*af*(*ldaf*, \*) には、[?hetrf](#) によって返された、因子分解後の行列  $A$  を格納する。  
*b*(*ldb*, \*) には、右辺の行列  $B$  が格納される。  
*x*(*ldx*, \*) には、解の行列  $X$  が格納される。  
*work*(\*) は、ワークスペース配列である。  
*a* と *af* の第 2 次元は  $\max(1, n)$  以上でなければならない。*b* と *x* の第 2 次元は  $\max(1, nrhs)$  以上でなければならない。*work* の次元は  $\max(1, 2*n)$  以上でなければならない。

*lda* INTEGER。 *a* の第 1 次元。  $lda \geq \max(1, n)$ 。

*ldaf* INTEGER。 *af* の第 1 次元。  $ldaf \geq \max(1, n)$ 。

*ldb* INTEGER。 *b* の第 1 次元。  $ldb \geq \max(1, n)$ 。

*ldx* INTEGER。 *x* の第 1 次元。  $ldx \geq \max(1, n)$ 。

*ipiv* INTEGER。  
 配列、次元は  $\max(1, n)$  以上。  
[?hetrf](#) によって返される *ipiv* 配列。

*rwork* REAL (cherfs の場合 )  
DOUBLE PRECISION (zherfs の場合 )。  
ワークスペース配列、次元は  $\max(1, n)$  以上。

### 出力パラメータ

*x* 精度が改善された解の行列  $X$ 。  
*ferr*, *berr* REAL (cherfs の場合 )  
DOUBLE PRECISION (zherfs の場合 )。  
配列、次元は  $\max(1, nrhs)$  以上。各解ベクトルの成分ごとの前進誤差と後退誤差を格納する。  
*info* INTEGER。  
*info* = 0 の場合、実行は正常に終了した。  
*info* = -*i* の場合、*i* 番目のパラメータの値が不正である。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

herfs ルーチンのインターフェイス特有の詳細を以下に示す。

*a* サイズ ( $n, n$ ) の行列  $A$  を格納する。  
*af* サイズ ( $n, n$ ) の行列  $AF$  を格納する。  
*ipiv* 長さ ( $n$ ) のベクトルを格納する。  
*b* サイズ ( $n, nrhs$ ) の行列  $B$  を格納する。  
*x* サイズ ( $n, nrhs$ ) の行列  $X$  を格納する。  
*ferr* 長さ ( $nrhs$ ) のベクトルを格納する。  
*berr* 長さ ( $nrhs$ ) のベクトルを格納する。  
*uplo* 'U' または 'L' でなければならない。デフォルト値は 'U' である。

### アプリケーション・ノート

*ferr* に返される誤差範囲は厳密ではないが、実際の誤差はほぼ常にこれより小さくなる。

各右辺について、後退誤差の計算に必要な浮動小数点演算の回数は、 $16n^2$  以上である。さらに、繰り返し行われる解の精度の改善の各段階に必要な演算の回数は、 $24n^2$  である。繰り返される回数の範囲は 1 ～ 5 になる。



前進誤差を推定するには、連立 1 次方程式  $Ax = b$  を繰り返し解く必要がある。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、約  $8n^2$  である。

このルーチンに対応する実数型は、[ssyrfs](#) / [dsyrfs](#) である。

## ?sprfs

圧縮格納形式の対称行列による連立 1 次方程式の解の精度を改善し、解の誤差を推定する。

### 構文

#### Fortran 77:

```
call ssprfs(uplo, n, nrhs, ap, afp, ipiv, b, ldb, x, ldx, ferr, berr, work,
            iwork, info)
call dsprfs(uplo, n, nrhs, ap, afp, ipiv, b, ldb, x, ldx, ferr, berr, work,
            iwork, info)
call csprfs(uplo, n, nrhs, ap, afp, ipiv, b, ldb, x, ldx, ferr, berr, work,
            rwork, info)
call zsprfs(uplo, n, nrhs, ap, afp, ipiv, b, ldb, x, ldx, ferr, berr, work,
            rwork, info)
```

#### Fortran 95:

```
call sprfs(a, af, ipiv, b, x [,uplo] [,ferr] [,berr] [,info])
```

### 説明

このルーチンは、圧縮格納形式の対称行列  $A$  による、複数の右辺を持つ連立 1 次方程式  $AX = B$  の解の精度の改善を繰り返し実行する。このルーチンは、算出されたそれぞれの解ベクトル  $x$  について、成分ごとに後退誤差  $\beta$  を計算する。次のように、 $x$  が摂動連立方程式の正確な解である場合、この誤差は、 $A$  と  $b$  の成分の最小相対摂動になる。

$$|\delta a_{ij}|/|a_{ij}| \leq \beta |a_{ij}|, |\delta b_i|/|b_i| \leq \beta |b_i| \quad \text{ただし } (A + \delta A)x = (b + \delta b)$$

最後に、このルーチンは、算出された解  $\|x - x_e\|_\infty / \|x\|_\infty$  の成分ごとに前進誤差を推定する ( $x_e$  は正確な解である)。

このルーチンを呼び出す前に、以下の手順を実行する必要がある。

- 因子分解ルーチン [?spturf](#) を呼び出す。

- 解の算出ルーチン [?spttrs](#) を呼び出す。

## 入力パラメータ

<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 入力行列 <i>A</i> の因子分解の方法を指定する。 <i>uplo</i> = 'U' の場合、配列 <i>afp</i> には、圧縮形式の Bunch-Kaufman 因子分解 $A = PUDU^TP^T$ を格納する。 <i>uplo</i> = 'L' の場合、配列 <i>afp</i> には、圧縮形式の Bunch-Kaufman 因子分解 $A = PLDL^TP^T$ を格納する。
<i>n</i>	INTEGER。行列 <i>A</i> の次数 ( $n \geq 0$ )。
<i>nrhs</i>	INTEGER。右辺の数 ( $nrhs \geq 0$ )。
<i>ap</i> , <i>afp</i> , <i>b</i> , <i>x</i> , <i>work</i>	REAL (ssprfs の場合) DOUBLE PRECISION (dsprfs の場合) COMPLEX (csprfs の場合) DOUBLE COMPLEX (zsprfs の場合)。 配列: <i>ap</i> (*) には、 <a href="#">?spturf</a> に入力として与えた元の圧縮形式の行列 <i>A</i> を格納する。 <i>afp</i> (*) には、 <a href="#">?spturf</a> によって返された、因子分解後の圧縮形式の行列 <i>A</i> を格納する。 <i>b</i> ( <i>ldb</i> , *) には、右辺の行列 <i>B</i> が格納される。 <i>x</i> ( <i>ldx</i> , *) には、解の行列 <i>X</i> が格納される。 <i>work</i> (*) は、ワークスペース配列である。  配列 <i>ap</i> と <i>afp</i> の次元は $\max(1, n(n+1)/2)$ 以上でなければならない。 <i>b</i> と <i>x</i> の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。 <i>work</i> の次元は $\max(1, 3*n)$ (実数型の場合) または $\max(1, 2*n)$ (複素数型の場合) 以上でなければならない。
<i>ldb</i>	INTEGER。 <i>b</i> の第 1 次元。 $ldb \geq \max(1, n)$ 。
<i>ldx</i>	INTEGER。 <i>x</i> の第 1 次元。 $ldx \geq \max(1, n)$ 。
<i>ipiv</i>	INTEGER。 配列、次元は $\max(1, n)$ 以上。 <a href="#">?spturf</a> によって返される <i>ipiv</i> 配列。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, n)$ 以上。

*rwork* REAL (csprfs の場合 )  
DOUBLE PRECISION (zsprfs の場合 )。  
ワークスペース配列、次元は  $\max(1, n)$  以上。

### 出力パラメータ

*x* 精度が改善された解の行列  $X$ 。  
*ferr*, *berr* REAL ( 単精度の場合 )  
DOUBLE PRECISION ( 倍精度の場合 )。  
配列、次元は  $\max(1, nrhs)$  以上。各解ベクトルの成分ごとの前進誤差と後退誤差を格納する。  
*info* INTEGER。  
*info* = 0 の場合、実行は正常に終了した。  
*info* = -*i* の場合、*i* 番目のパラメータの値が不正である。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

sprfs ルーチンのインターフェイス特有の詳細を以下に示す。

*a* Fortran 77 インターフェイスでの引数 *ap* を意味する。  
サイズ  $(n*(n+1)/2)$  の配列  $A$  を格納する。  
*af* Fortran 77 インターフェイスでの引数 *afp* を意味する。  
サイズ  $(n*(n+1)/2)$  の配列  $AF$  を格納する。  
*ipiv* 長さ (*n*) のベクトルを格納する。  
*b* サイズ (*n*, *nrhs*) の行列  $B$  を格納する。  
*x* サイズ (*n*, *nrhs*) の行列  $X$  を格納する。  
*ferr* 長さ (*nrhs*) のベクトルを格納する。  
*berr* 長さ (*nrhs*) のベクトルを格納する。  
*uplo* 'U' または 'L' でなければならない。デフォルト値は 'U' である。

### アプリケーション・ノート

*ferr* に返される誤差範囲は厳密ではないが、実際の誤差はほぼ常にこれより小さくなる。

各右辺について、後退誤差の計算に必要な浮動小数点演算の回数は、実数型の場合は  $4n^2$  以上、複素数型の場合は  $16n^2$  以上である。さらに、繰り返し行われる解の精度の改善の各段階で必要な演算の回数は、実数型の場合は  $6n^2$  で、複素数型の場合は  $24n^2$  である。繰り返される回数の範囲は 1 ～ 5 になる。

前進誤差を推定するには、連立 1 次方程式  $Ax = b$  を繰り返し解く必要がある。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約  $2n^2$  で、複素数型の場合は約  $8n^2$  である。

---

### ?hprfs

圧縮格納形式の複素エルミート行列による連立  
1 次方程式の解の精度を改善し、解の誤差を推定  
する。

---

#### 構文

##### Fortran 77:

```
call chprfs(uplo, n, nrhs, ap, afp, ipiv, b, ldb, x, ldx, ferr, berr, work,  
            rwork, info)  
call zhprfs(uplo, n, nrhs, ap, afp, ipiv, b, ldb, x, ldx, ferr, berr, work,  
            rwork, info)
```

##### Fortran 95:

```
call hprfs(a, af, ipiv, b, x [,uplo] [,ferr] [,berr] [,info])
```

#### 説明

このルーチンは、圧縮格納形式の複素エルミート行列  $A$  による、複数の右辺を持つ連立 1 次方程式  $AX = B$  の解の精度の改善を繰り返し実行する。このルーチンは、算出されたそれぞれの解ベクトル  $x$  について、成分ごとに後退誤差  $\beta$  を計算する。次のように、 $x$  が摂動連立方程式の正確な解である場合、この誤差は、 $A$  と  $b$  の成分の最小相対摂動になる。

$$|\delta a_{ij}|/|a_{ij}| \leq \beta |a_{ij}|, |\delta b_i|/|b_i| \leq \beta |b_i| \text{ ただし } (A + \delta A)x = (b + \delta b)$$

最後に、このルーチンは、算出された解  $\|x - x_e\|_\infty / \|x\|_\infty$  の成分ごとに前進誤差を推定する ( $x_e$  は正確な解である)。

このルーチンを呼び出す前に、以下の手順を実行する必要がある。

- 因子分解ルーチン [?hptrf](#) を呼び出す。
- 解の算出ルーチン [?hptrs](#) を呼び出す。

### 入力パラメータ

<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 入力行列 $A$ の因子分解の方法を指定する。 <i>uplo</i> = 'U' の場合、配列 <i>afp</i> には、圧縮形式の Bunch-Kaufman 因子分解 $A = PUDU^H P^T$ を格納する。 <i>uplo</i> = 'L' の場合には、配列 <i>afp</i> は、圧縮形式の Bunch-Kaufman 因子分解 $A = PLDL^H P^T$ を格納する。
<i>n</i>	INTEGER。行列 $A$ の次数 ( $n \geq 0$ )。
<i>nrhs</i>	INTEGER。右辺の数 ( $nrhs \geq 0$ )。
<i>ap</i> , <i>afp</i> , <i>b</i> , <i>x</i> , <i>work</i>	COMPLEX (chprfs の場合) DOUBLE COMPLEX (zhprfs の場合)。 配列: <i>ap</i> (*) には、 <a href="#">?hptrf</a> に入力として与えた元の圧縮形式の行列 $A$ を格納する。 <i>afp</i> (*) には、 <a href="#">?hptrf</a> によって返された、因子分解後の圧縮形式の行列 $A$ を格納する。 <i>b</i> ( <i>ldb</i> , *) には、右辺の行列 $B$ が格納される。 <i>x</i> ( <i>ldx</i> , *) には、解の行列 $X$ が格納される。 <i>work</i> (*) は、ワークスペース配列である。  配列 <i>ap</i> と <i>afp</i> の次元は $\max(1, n(n+1)/2)$ 以上でなければならない。 <i>b</i> と <i>x</i> の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。 <i>work</i> の次元は $\max(1, 2*n)$ 以上でなければならない。
<i>ldb</i>	INTEGER。 <i>b</i> の第 1 次元。 $ldb \geq \max(1, n)$ 。
<i>ldx</i>	INTEGER。 <i>x</i> の第 1 次元。 $ldx \geq \max(1, n)$ 。
<i>ipiv</i>	INTEGER。 配列、次元は $\max(1, n)$ 以上。 <a href="#">?hptrf</a> によって返される <i>ipiv</i> 配列。
<i>rwork</i>	REAL (chprfs の場合) DOUBLE PRECISION (zhprfs の場合)。 ワークスペース配列、次元は $\max(1, n)$ 以上。

### 出力パラメータ

<i>x</i>	精度が改善された解の行列 <i>X</i> 。
<i>ferr</i> , <i>berr</i>	REAL ( <i>chprfs</i> の場合 )。 DOUBLE PRECISION ( <i>zhprfs</i> の場合 )。 配列、次元は $\max(1, nrhs)$ 以上。各解ベクトルの成分ごとの前進誤差と後退誤差を格納する。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了した。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正である。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

*hprfs* ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ap</i> を意味する。 サイズ $(n*(n+1)/2)$ の配列 <i>A</i> を格納する。
<i>af</i>	Fortran 77 インターフェイスでの引数 <i>afp</i> を意味する。 サイズ $(n*(n+1)/2)$ の配列 <i>AF</i> を格納する。
<i>ipiv</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>b</i>	サイズ ( <i>n</i> , <i>nrhs</i> ) の行列 <i>B</i> を格納する。
<i>x</i>	サイズ ( <i>n</i> , <i>nrhs</i> ) の行列 <i>X</i> を格納する。
<i>ferr</i>	長さ ( <i>nrhs</i> ) のベクトルを格納する。
<i>berr</i>	長さ ( <i>nrhs</i> ) のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U' である。

### アプリケーション・ノート

*ferr* に返される誤差範囲は厳密ではないが、実際の誤差はほぼ常にこれより小さくなる。

各右辺について、後退誤差の計算に必要な浮動小数点演算の回数は、 $16n^2$  以上である。さらに、繰り返し行われる解の精度の改善の各段階で必要な演算の回数は、 $24n^2$  である。繰り返される回数の範囲は 1 ～ 5 になる。

前進誤差を推定するには、連立 1 次方程式  $Ax = b$  を繰り返し解く必要がある。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、約  $8n^2$  である。

このルーチンに対応する実数型は、[ssprfs](#) / [dsprfs](#) である。

## ?trrfs

三角行列による連立 1 次方程式の解の誤差を推定する。

### 構文

#### Fortran 77:

```
call strrfs(uplo, trans, diag, n, nrhs, a, lda, b, ldb, x, ldx, ferr, berr,
            work, iwork, info)
call dtrrfs(uplo, trans, diag, n, nrhs, a, lda, b, ldb, x, ldx, ferr, berr,
            work, iwork, info)
call ctrrfs(uplo, trans, diag, n, nrhs, a, lda, b, ldb, x, ldx, ferr, berr,
            work, rwork, info)
call ztrrfs(uplo, trans, diag, n, nrhs, a, lda, b, ldb, x, ldx, ferr, berr,
            work, rwork, info)
```

#### Fortran 95:

```
call trrfs(a, b, x [,uplo] [,trans] [,diag] [,ferr] [,berr] [,info])
```

### 説明

このルーチンは、三角行列  $A$  による、複数の右辺を持つ連立 1 次方程式  $AX = B$  または  $A^T X = B$  または  $A^H X = B$  の解の誤差を推定する。このルーチンは、算出されたそれぞれの解ベクトル  $x$  について、成分ごとに後退誤差  $\beta$  を計算する。次のように、 $x$  が摂動連立方程式の正確な解である場合、この誤差は、 $A$  と  $b$  の成分の最小相対摂動になる。

$$|\delta a_{ij}|/|a_{ij}| \leq \beta |a_{ij}|, |\delta b_i|/|b_i| \leq \beta |b_i| \text{ ただし } (A + \delta A)x = (b + \delta b)$$

また、このルーチンは、算出された解  $\|x - x_e\|_\infty / \|x\|_\infty$  の成分ごとに前進誤差を推定する ( $x_e$  は正確な解である)。

このルーチンを呼び出す前に、解の算出ルーチン [?trtrs](#) を呼び出す。

## 入力パラメータ

<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 A が上三角行列か、下三角行列かを指定する。  <i>uplo</i> = 'U' の場合、A は上三角行列である。 <i>uplo</i> = 'L' の場合、A は下三角行列である。
<i>trans</i>	CHARACTER*1。'N' または 'T' または 'C' でなければならない。 方程式の形式を指定する。 <i>trans</i> = 'N' の場合、連立方程式の形式は $AX = B$ である。 <i>trans</i> = 'T' の場合、連立方程式の形式は $A^T X = B$ である。 <i>trans</i> = 'C' の場合、連立方程式の形式は $A^H X = B$ である。
<i>diag</i>	CHARACTER*1。'N' または 'U' でなければならない。 <i>diag</i> = 'N' の場合、A は単位三角行列ではない。  <i>diag</i> = 'U' の場合、A は単位三角行列である。A の対角成分は 1 とみなされ、配列 <i>a</i> 内で参照されない。
<i>n</i>	INTEGER。行列 A の次数 ( $n \geq 0$ )。
<i>nrhs</i>	INTEGER。右辺の数 ( $nrhs \geq 0$ )。
<i>a</i> , <i>b</i> , <i>x</i> , <i>work</i>	REAL( <i>strrrfs</i> の場合) DOUBLE PRECISION( <i>dtrrrfs</i> の場合) COMPLEX( <i>ctrrrfs</i> の場合) DOUBLE COMPLEX( <i>ztrrrfs</i> の場合)。  配列:  <i>a</i> ( <i>lda</i> , *) には、 <i>uplo</i> によって指定される上三角または下三角行列 A を格納する。  <i>b</i> ( <i>ldb</i> , *) には、右辺の行列 B が格納される。  <i>x</i> ( <i>ldx</i> , *) には、解の行列 X が格納される。  <i>work</i> (*) は、ワークスペース配列である。  <i>a</i> の第 2 次元は $\max(1, n)$ 以上でなければならない。 <i>b</i> と <i>x</i> の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。 <i>work</i> の次元は $\max(1, 3*n)$ (実数型の場合) または $\max(1, 2*n)$ (複素数型の場合) 以上でなければならない。
<i>lda</i>	INTEGER。a の第 1 次元。 $lda \geq \max(1, n)$ 。
<i>ldb</i>	INTEGER。b の第 1 次元。 $ldb \geq \max(1, n)$ 。
<i>ldx</i>	INTEGER。x の第 1 次元。 $ldx \geq \max(1, n)$ 。



<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, n)$ 以上。
<i>rwork</i>	REAL (ctrtrfs の場合 ) DOUBLE PRECISION (ztrtrfs の場合 )。 ワークスペース配列、次元は $\max(1, n)$ 以上。

### 出力パラメータ

<i>ferr</i> , <i>berr</i>	REAL ( 単精度の場合 ) DOUBLE PRECISION ( 倍精度の場合 )。 配列、次元は $\max(1, nrhs)$ 以上。各解ベクトルの成分ごとの前進誤差と後退誤差を格納する。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了した。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正である。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

trrfs ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	サイズ $(n, n)$ の行列 <i>A</i> を格納する。
<i>b</i>	サイズ $(n, nrhs)$ の行列 <i>B</i> を格納する。
<i>x</i>	サイズ $(n, nrhs)$ の行列 <i>X</i> を格納する。
<i>ferr</i>	長さ $(nrhs)$ のベクトルを格納する。
<i>berr</i>	長さ $(nrhs)$ のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U' である。
<i>trans</i>	'N'、'C'、または 'T' でなければならない。デフォルト値は 'N' である。
<i>diag</i>	'N' または 'U' でなければならない。デフォルト値は 'N' である。

### アプリケーション・ノート

*ferr* に返される誤差範囲は厳密ではないが、実際の誤差はほぼ常にこれより小さくなる。

このルーチン呼び出す場合は、各右辺について、連立 1 次方程式  $Ax = b$  を繰り返し解く必要がある。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約  $n^2$  で、複素数型の場合は約  $4n^2$  である。

---

### ?tprfs

圧縮格納形式の三角行列による連立 1 次方程式の解の誤差を推定する。

---

#### 構文

##### Fortran 77:

```
call stprfs(uplo, trans, diag, n, nrhs, ap, b, ldb, x, ldx, ferr, berr, work,
            iwork, info)
call dtprfs(uplo, trans, diag, n, nrhs, ap, b, ldb, x, ldx, ferr, berr, work,
            iwork, info)
call ctprfs(uplo, trans, diag, n, nrhs, ap, b, ldb, x, ldx, ferr, berr, work,
            rwork, info)
call ztprfs(uplo, trans, diag, n, nrhs, ap, b, ldb, x, ldx, ferr, berr, work,
            rwork, info)
```

##### Fortran 95:

```
call tprfs(a, b, x [,uplo] [,trans] [,diag] [,ferr] [,berr] [,info])
```

#### 説明

このルーチンは、圧縮格納形式の三角行列  $A$  による、複数の右辺を持つ連立 1 次方程式  $AX = B$  または  $A^T X = B$  または  $A^H X = B$  の解の誤差を推定する。このルーチンは、算出されたそれぞれの解ベクトル  $x$  について、成分ごとに後退誤差  $\beta$  を計算する。次のように、 $x$  が摂動連立方程式の正確な解である場合、この誤差は、 $A$  と  $b$  の成分の最小相対摂動になる。

$$|\delta a_{ij}|/|a_{ij}| \leq \beta |a_{ij}|, |\delta b_i|/|b_i| \leq \beta |b_i| \quad \text{ただし } (A + \delta A)x = (b + \delta b)$$

また、このルーチンは、算出された解  $\|x - x_e\|_\infty / \|x\|_\infty$  の成分ごとに前進誤差を推定する ( $x_e$  は正確な解である)。

このルーチン呼び出す前に、解の算出ルーチン [?tpttrs](#) を呼び出す。

## 入力パラメータ

<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 A が上三角行列か、下三角行列かを指定する。  <i>uplo</i> = 'U' の場合、A は上三角行列である。 <i>uplo</i> = 'L' の場合、A は下三角行列である。
<i>trans</i>	CHARACTER*1。'N' または 'T' または 'C' でなければならない。 方程式の形式を指定する。 <i>trans</i> = 'N' の場合、連立方程式の形式は $AX = B$ である。 <i>trans</i> = 'T' の場合、連立方程式の形式は $A^T X = B$ である。 <i>trans</i> = 'C' の場合、連立方程式の形式は $A^H X = B$ である。
<i>diag</i>	CHARACTER*1。'N' または 'U' でなければならない。 <i>diag</i> = 'N' の場合、A は単位三角行列ではない。  <i>diag</i> = 'U' の場合、A は単位三角行列である。A の対角成分は 1 とみなされ、配列 <i>ap</i> 内で参照されない。
<i>n</i>	INTEGER。行列 A の次数 ( $n \geq 0$ )。
<i>nrhs</i>	INTEGER。右辺の数 ( $nrhs \geq 0$ )。
<i>ap</i> , <i>b</i> , <i>x</i> , <i>work</i>	REAL (strrfs の場合) DOUBLE PRECISION (dtrrfs の場合) COMPLEX (ctrfs の場合) DOUBLE COMPLEX (ztrrfs の場合)。 配列: <i>ap</i> (*) には、 <i>uplo</i> によって指定される上三角または下三角行列 A を格納する。 <i>b</i> ( <i>ldb</i> , *) には、右辺の行列 B が格納される。 <i>x</i> ( <i>ldx</i> , *) には、解の行列 X が格納される。 <i>work</i> (*) は、ワークスペース配列である。  <i>ap</i> の次元は $\max(1, n(n+1)/2)$ 以上でなければならない。 <i>b</i> と <i>x</i> の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。 <i>work</i> の次元は $\max(1, 3*n)$ (実数型の場合) または $\max(1, 2*n)$ (複素数型の場合) 以上でなければならない。
<i>ldb</i>	INTEGER。b の第 1 次元。 $ldb \geq \max(1, n)$ 。
<i>ldx</i>	INTEGER。x の第 1 次元。 $ldx \geq \max(1, n)$ 。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, n)$ 以上。

*rwork* REAL (ctrdfs の場合 )  
DOUBLE PRECISION (ztrdfs の場合 )。  
ワークスペース配列、次元は  $\max(1, n)$  以上。

### 出力パラメータ

*ferr, berr* REAL ( 単精度の場合 )  
DOUBLE PRECISION ( 倍精度の場合 )。  
配列、次元は  $\max(1, nrhs)$  以上。各解ベクトルの成分ごとの前進誤差と後退誤差を格納する。

*info* INTEGER。  
*info* = 0 の場合、実行は正常に終了した。  
*info* = -*i* の場合、*i* 番目のパラメータの値が不正である。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

tpdfs ルーチンのインターフェイス特有の詳細を以下に示す。

*a* Fortran 77 インターフェイスでの引数 *ap* を意味する。サイズ  $(n*(n+1)/2)$  の配列 *A* を格納する。

*b* サイズ  $(n, nrhs)$  の行列 *B* を格納する。

*x* サイズ  $(n, nrhs)$  の行列 *X* を格納する。

*ferr* 長さ  $(nrhs)$  のベクトルを格納する。

*berr* 長さ  $(nrhs)$  のベクトルを格納する。

*uplo* 'U' または 'L' でなければならない。デフォルト値は 'U' である。

*trans* 'N'、'C'、または 'T' でなければならない。デフォルト値は 'N' である。

*diag* 'N' または 'U' でなければならない。デフォルト値は 'N' である。

### アプリケーション・ノート

*ferr* に返される誤差範囲は厳密ではないが、実際の誤差はほぼ常にこれより小さくなる。

このルーチンを呼び出す場合は、各右辺について、連立 1 次方程式  $Ax = b$  を繰り返し解く必要がある。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約  $n^2$  で、複素数型の場合は約  $4n^2$  である。

## ?tbrfs

三角帯行列による連立 1 次方程式の解の誤差を推定する。

### 構文

#### Fortran 77:

```
call stbrfs(uplo, trans, diag, n, kd, nrhs, ab, ldab, b, ldb, x, ldx, ferr,
            berr, work, iwork, info)
call dtbrfs(uplo, trans, diag, n, kd, nrhs, ab, ldab, b, ldb, x, ldx, ferr,
            berr, work, iwork, info)
call ctbrfs(uplo, trans, diag, n, kd, nrhs, ab, ldab, b, ldb, x, ldx, ferr,
            berr, work, rwork, info)
call ztbrfs(uplo, trans, diag, n, kd, nrhs, ab, ldab, b, ldb, x, ldx, ferr,
            berr, work, rwork, info)
```

#### Fortran 95:

```
call tbrfs(a, b, x [,uplo] [,trans] [,diag] [,ferr] [,berr] [,info])
```

### 説明

このルーチンは、三角帯行列  $A$  による、複数の右辺を持つ連立 1 次方程式  $AX = B$  または  $A^T X = B$  または  $A^H X = B$  の解の誤差を推定する。このルーチンは、算出されたそれぞれの解ベクトル  $x$  について、成分ごとに後退誤差  $\beta$  を計算する。次のように、 $x$  が摂動連立方程式の正確な解である場合、この誤差は、 $A$  と  $b$  の成分の最小相対摂動になる。

$$|\delta a_{ij}|/|a_{ij}| \leq \beta |a_{ij}|, |\delta b_i|/|b_i| \leq \beta |b_i| \quad \text{ただし } (A + \delta A)x = (b + \delta b)$$

また、このルーチンは、算出された解  $\|x - x_e\|_\infty / \|x\|_\infty$  の成分ごとに前進誤差を推定する ( $x_e$  は正確な解である)。

このルーチンを呼び出す前に、解の算出ルーチン [?tbtrs](#) を呼び出す。

## 入力パラメータ

<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 A が上三角行列か、下三角行列かを指定する。  <i>uplo</i> = 'U' の場合、A は上三角行列である。 <i>uplo</i> = 'L' の場合、A は下三角行列である。
<i>trans</i>	CHARACTER*1。'N' または 'T' または 'C' でなければならない。 方程式の形式を指定する。 <i>trans</i> = 'N' の場合、連立方程式の形式は $AX = B$ である。 <i>trans</i> = 'T' の場合、連立方程式の形式は $A^T X = B$ である。 <i>trans</i> = 'C' の場合、連立方程式の形式は $A^H X = B$ である。
<i>diag</i>	CHARACTER*1。'N' または 'U' でなければならない。 <i>diag</i> = 'N' の場合、A は単位三角行列ではない。  <i>diag</i> = 'U' の場合、A は単位三角行列である。A の対角成分は 1 とみなされ、配列 <i>ab</i> 内で参照されない。
<i>n</i>	INTEGER。行列 A の次数 ( $n \geq 0$ )。
<i>kd</i>	INTEGER。A の優対角成分または劣対角成分の数 ( $kd \geq 0$ )。
<i>nrhs</i>	INTEGER。右辺の数 ( $nrhs \geq 0$ )。
<i>ab, b, x, work</i>	REAL ( <i>stbrfs</i> の場合) DOUBLE PRECISION ( <i>dtbrfs</i> の場合) COMPLEX ( <i>ctbrfs</i> の場合) DOUBLE COMPLEX ( <i>ztbrfs</i> の場合)。  配列:  <i>ab</i> ( <i>ldab</i> , *) には、 <i>uplo</i> によって指定される上三角または下三角行列 A を帯格納形式で格納する。  <i>b</i> ( <i>ldb</i> , *) には、右辺の行列 B が格納される。  <i>x</i> ( <i>ldx</i> , *) には、解の行列 X が格納される。  <i>work</i> (*) は、ワークスペース配列である。  a の第 2 次元は $\max(1, n)$ 以上でなければならない。b と x の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。 <i>work</i> の次元は、 $\max(1, 3*n)$ (実数型の場合) または $\max(1, 2*n)$ (複素数の場合) 以上でなければならない。
<i>ldab</i>	INTEGER。配列 <i>ab</i> の第 1 次元 ( $ldab \geq kd + 1$ )。
<i>ldb</i>	INTEGER。b の第 1 次元。 $ldb \geq \max(1, n)$ 。
<i>ldx</i>	INTEGER。x の第 1 次元。 $ldx \geq \max(1, n)$ 。

*iwork*            INTEGER。  
ワークスペース配列、次元は  $\max(1, n)$  以上。

*rwork*            REAL (ctbrfs の場合 )  
DOUBLE PRECISION (ztbrfs の場合 )。  
ワークスペース配列、次元は  $\max(1, n)$  以上。

### 出力パラメータ

*ferr*, *berr*      REAL ( 単精度の場合 )  
DOUBLE PRECISION ( 倍精度の場合 )。  
配列、次元は  $\max(1, nrhs)$  以上。各解ベクトルの成分ごとの前進誤差  
と後退誤差を格納する。

*info*            INTEGER。  
*info* = 0 の場合、実行は正常に終了した。  
*info* = -*i* の場合、*i* 番目のパラメータの値が不正である。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

tbrfs ルーチンのインターフェイス特有の詳細を以下に示す。

*a*                Fortran 77 インターフェイスでの引数 *ab* を意味する。サイズ  $(kd+1, n)$  の配列 *A* を格納する。

*b*                サイズ  $(n, nrhs)$  の行列 *B* を格納する。

*x*                サイズ  $(n, nrhs)$  の行列 *X* を格納する。

*ferr*            長さ  $(nrhs)$  のベクトルを格納する。

*berr*            長さ  $(nrhs)$  のベクトルを格納する。

*uplo*            'U' または 'L' でなければならない。デフォルト値は 'U' である。

*trans*           'N'、'C'、または 'T' でなければならない。デフォルト値は 'N' である。

*diag*            'N' または 'U' でなければならない。デフォルト値は 'N' である。

### アプリケーション・ノート

*ferr* に返される誤差範囲は厳密ではないが、実際の誤差はほぼ常にこれより小さくなる。

このルーチン呼び出す場合は、各右辺について、連立 1 次方程式  $Ax=b$  を繰り返し解く必要がある。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約  $2n*kd$  で、複素数型の場合は約  $8n*kd$  である。

### 行列の反転用のルーチン

行列の逆行列を明示的に計算することは、実際にはほとんどない。  
特に、連立方程式  $Ax=b$  を解く場合に、最初に  $A^{-1}$  を計算してから、行列とベクトルの積  $x=A^{-1}b$  を計算してはならない。  
代わりに、ソルバルーチン呼び出す(「[連立 1 次方程式を解くためのルーチン](#)」を参照)。この方が効率が良く、高い精度の結果が得られる。

ただし、逆行列を求める必要が生じた場合に備えて、行列反転用のルーチンが用意されている。

---

## ?getri

LU 因子分解された一般行列の逆行列を計算する。

---

### 構文

#### Fortran 77:

```
call sgetri(n, a, lda, ipiv, work, lwork, info)
call dgetri(n, a, lda, ipiv, work, lwork, info)
call cgetri(n, a, lda, ipiv, work, lwork, info)
call zgetri(n, a, lda, ipiv, work, lwork, info)
```

#### Fortran 95:

```
call getri(a, ipiv [,info])
```



## 説明

このルーチンは、一般行列  $A$  の逆行列 ( $A^{-1}$ ) を計算する。  
 このルーチンを呼び出す前に、[?getrf](#) を呼び出して、 $A$  を因子分解する。

## 入力パラメータ

$n$  INTEGER。行列  $A$  の次数 ( $n \geq 0$ )。

$a, work$  REAL (sgetri の場合 )  
 DOUBLE PRECISION (dgetri の場合 )  
 COMPLEX (cgetri の場合 )  
 DOUBLE COMPLEX (zgetri の場合 )。  
 配列 :  $a(lda, *)$ ,  $work(lwork)$ 。  
 $a(lda, *)$  には、[?getrf](#) によって返される、行列  $A$  の因子分解  $A = PLU$  を格納する。  
 $a$  の第 2 次元は  $\max(1, n)$  以上でなければならない。  
 $work(lwork)$  は、ワークスペース配列である。

$lda$  INTEGER。  $a$  の第 1 次元。  $lda \geq \max(1, n)$ 。

$ipiv$  INTEGER。  
 配列、次元は  $\max(1, n)$  以上。  
[?getrf](#) によって返される  $ipiv$  配列。

$lwork$  INTEGER。 配列  $work$  のサイズ。  $lwork \geq n$ 。  
 $lwork = -1$  の場合はワークスペースのクエリとみなされ、ルーチンは  $work$  配列の最適サイズだけを計算し、その値を  $work$  配列の最初のエントリとして返し、 $xerbla$  は  $lwork$  に関するエラー・メッセージを生成しない。  
 $lwork$  の推奨値は、「アプリケーション・ノート」を参照。

## 出力パラメータ

$a$   $n \times n$  の行列  $A^{-1}$  によって上書きされる。

$work(1)$   $info = 0$  の場合、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の  $lwork$  の値が  $work(1)$  に出力される。  
 これ以降の実行には、この  $lwork$  の値を使用する。

*info* INTEGER。 *info* = 0 の場合、実行は正常に終了した。  
*info* = -*i* の場合、*i* 番目のパラメータの値が不正である。  
*info* = *i* の場合、係数 *U* の *i* 番目の対角成分がゼロで、*U* が特異になるため、行列の反転を完了できない。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

getri ルーチンのインターフェイス特有の詳細を以下に示す。

*a*                      サイズ (*n*,*n*) の行列 *A* を格納する。

*ipiv*                    長さ (*n*) のベクトルを格納する。

### アプリケーション・ノート

パフォーマンスを改善するには、*lwork* = *n*\**blocksize* に設定する。*blocksize* は、ブロック・アルゴリズムの最適なパフォーマンスを得るために必

要な値で、マシンによって異なる (通常は 16 ~ 64)。

必要なワークスペースの大きさがわからない場合は、最初の実行では *lwork* を大きめの値に設定する。終了時に、*work*(1) の値を確認し、これ以降の実行にはその値を使用する。

算出された逆行列 *X* は、以下の誤差範囲を満たす。

$$|XA - I| \leq c(n)\varepsilon |X|P|L||U|$$

*c*(*n*) は *n* の適度な 1 次関数で、 $\varepsilon$  はマシンの精度である。*I* は元の行列を示す。*P*、*L*、および *U* は、行列の因子分解  $A = PLU$  の係数である。

浮動小数点演算の合計回数は、実数型の場合は約  $(4/3)n^3$  で、複素数型の場合は約  $(16/3)n^3$  になる。

## ?potri

対称 (エルミート) 正定値行列の逆行列を計算する。

### 構文

#### Fortran 77:

```
call spotri(uplo, n, a, lda, info)
call dpotri(uplo, n, a, lda, info)
call cpotri(uplo, n, a, lda, info)
call zpotri(uplo, n, a, lda, info)
```

#### Fortran 95:

```
call potri(a [,uplo] [,info])
```

### 説明

このルーチンは、対称 (複素行列の場合はエルミート) 正定値行列  $A$  の逆行列 ( $A^{-1}$ ) を計算する。

このルーチンを呼び出す前に、[?potrf](#) を呼び出して、 $A$  を因子分解する。

### 入力パラメータ

**uplo** CHARACTER\*1。'U' または 'L' でなければならない。  
 入力行列  $A$  の因子分解の方法を指定する。  
 uplo = 'U' の場合、配列  $a$  には、コレスキー因子分解  $A = U^H U$  の係数  $U$  を格納する。  
 uplo = 'L' の場合、配列  $a$  には、コレスキー因子分解  $A = L L^H$  の係数  $L$  を格納する。

**n** INTEGER。行列  $A$  の次数 ( $n \geq 0$ )。

**a** REAL (spotri の場合)  
 DOUBLE PRECISION (dpotri の場合)  
 COMPLEX (cpotri の場合)  
 DOUBLE COMPLEX (zpotri の場合)。  
 配列:  $a(lda, *)$ 。  
[?potrf](#) によって返される、行列  $A$  の因子分解を格納する。

$a$  の第 2 次元は  $\max(1, n)$  以上でなければならない。

$lda$                     INTEGER。  $a$  の第 1 次元。  $lda \geq \max(1, n)$ 。

## 出力パラメータ

$a$                      $n \times n$  の行列  $A^{-1}$  によって上書きされる。

$info$                     INTEGER。  
 $info = 0$  の場合、実行は正常に終了した。  
 $info = -i$  の場合、 $i$  番目のパラメータの値が不正である。  
 $info = i$  の場合、コレスキー係数の  $i$  番目の対角成分 (したがって、  
 係数そのもの) がゼロのため、反転を完了できない。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

potri ルーチンのインターフェイス特有の詳細を以下に示す。

$a$                     サイズ  $(n, n)$  の行列  $A$  を格納する。

$uplo$                     'U' または 'L' でなければならない。デフォルト値は 'U' である。

## アプリケーション・ノート

算出された逆行列  $X$  は、以下の誤差範囲を満たす。

$$\|XA - I\|_2 \leq c(n)\varepsilon\kappa_2(A), \quad \|AX - I\|_2 \leq c(n)\varepsilon\kappa_2(A)$$

$c(n)$  は  $n$  の適度な 1 次関数で、 $\varepsilon$  はマシンの精度である。 $I$  は元の行列を示す。

行列  $A$  の 2- ノルム  $\|A\|_2$  は、 $\|A\|_2 = \max_{x \neq 0} (Ax \cdot Ax)^{1/2}$  によって定義される。条件数  $\kappa_2(A)$  は、 $\kappa_2(A) = \|A\|_2 \|A^{-1}\|_2$  によって定義される。

浮動小数点演算の合計回数は、実数型の場合は約  $(2/3)n^3$  で、複素数型の場合は約  $(8/3)n^3$  になる。

## ?pptri

圧縮格納形式の対称 (エルミート) 正定値行列の  
逆行列を計算する。

### 構文

#### Fortran 77:

```
call spptri(uplo, n, ap, info)
call dpptri(uplo, n, ap, info)
call cpptri(uplo, n, ap, info)
call zpptri(uplo, n, ap, info)
```

#### Fortran 95:

```
call pptri(a [,uplo] [,info])
```

### 説明

このルーチンは、*圧縮形式の対称* (複素行列の場合はエルミート) 正定値行列  $A$  の逆行列 ( $A^{-1}$ ) を計算する。このルーチンを呼び出す前に、[?pptrf](#) を呼び出して、 $A$  を因子分解する。

### 入力パラメータ

**uplo** CHARACTER\*1。'U' または 'L' でなければならない。  
 入力行列  $A$  の因子分解の方法を指定する。  
 uplo = 'U' の場合、配列 **ap** には、コレスキー因子分解  $A = U^H U$  の圧縮形式の係数  $U$  を格納する。  
 uplo = 'L' の場合、配列 **ap** には、コレスキー因子分解  $A = L L^H$  の圧縮形式の係数  $L$  を格納する。

**n** INTEGER。行列  $A$  の次数 ( $n \geq 0$ )。

**ap** REAL (spptri の場合)  
 DOUBLE PRECISION (dpptri の場合)  
 COMPLEX (cpptri の場合)  
 DOUBLE COMPLEX (zpptri の場合)。  
 配列、次元は  $\max(1, n(n+1)/2)$  以上。  
[?pptrf](#) によって返される、圧縮形式の行列  $A$  の因子分解を格納する。  
 ap の次元は  $\max(1, n(n+1)/2)$  以上でなければならない。

### 出力パラメータ

<i>ap</i>	圧縮形式の $n \times n$ の行列 $A^{-1}$ によって上書きされる。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了した。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正である。 <i>info</i> = <i>i</i> の場合、コレスキー係数の <i>i</i> 番目の対角成分 (したがって、係数そのもの) がゼロのため、反転を完了できない。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

pptri ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ap</i> を意味する。サイズ $(n*(n+1)/2)$ の配列 <i>A</i> を格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U' である。

### アプリケーション・ノート

算出された逆行列 *X* は、以下の誤差範囲を満たす。

$$\|XA - I\|_2 \leq c(n)\varepsilon\kappa_2(A), \quad \|AX - I\|_2 \leq c(n)\varepsilon\kappa_2(A)$$

$c(n)$  は  $n$  の適度な 1 次関数で、 $\varepsilon$  はマシンの精度である。*I* は元の行列を示す。

行列 *A* の 2- ノルム  $\|A\|_2$  は、 $\|A\|_2 = \max_{x \neq 0} (Ax \cdot Ax)^{1/2}$  によって定義される。条件数  $\kappa_2(A)$  は、 $\kappa_2(A) = \|A\|_2 \|A^{-1}\|_2$  によって定義される。

浮動小数点演算の合計回数は、実数型の場合は約  $(2/3)n^3$  で、複素数型の場合は約  $(8/3)n^3$  になる。

## ?sytri

対称行列の逆行列を計算する。

### 構文

#### Fortran 77:

```
call ssytri(uplo, n, a, lda, ipiv, work, info)
call dsytri(uplo, n, a, lda, ipiv, work, info)
call csytri(uplo, n, a, lda, ipiv, work, info)
call zsytri(uplo, n, a, lda, ipiv, work, info)
```

#### Fortran 95:

```
call sytri(a, ipiv [,uplo] [,info])
```

### 説明

このルーチンは、対称行列  $A$  の逆行列 ( $A^{-1}$ ) を計算する。  
 このルーチン呼び出す前に、[?sytrf](#) を呼び出して、 $A$  を因子分解する。

### 入力パラメータ

**uplo** CHARACTER\*1。'U' または 'L' でなければならない。  
 入力行列  $A$  の因子分解の方法を指定する。  
 uplo = 'U' の場合、配列  $a$  には、Bunch-Kaufman 因子分解  $A = PUDU^T P^T$  を格納する。  
 uplo = 'L' の場合、配列  $a$  には、Bunch-Kaufman 因子分解  $A = PLDL^T P^T$  を格納する。

**n** INTEGER。行列  $A$  の次数 ( $n \geq 0$ )。

**a, work** REAL (ssytri の場合)  
 DOUBLE PRECISION (dsytri の場合)  
 COMPLEX (csytri の場合)  
 DOUBLE COMPLEX (zsytri の場合)。  
 配列:  
 a(lda, \*) には、[?sytrf](#) によって返される、行列  $A$  の因子分解を格納する。  
 a の第 2 次元は  $\max(1, n)$  以上でなければならない。

*work*(\*) は、ワークスペース配列。  
*work* の次元は  $\max(1, 2*n)$  以上でなければならない。

*lda*                    INTEGER。 *a* の第 1 次元。  $lda \geq \max(1, n)$ 。

*ipiv*                    INTEGER。  
配列、次元は  $\max(1, n)$  以上。  
[?sytrf](#) によって返される *ipiv* 配列。

### 出力パラメータ

*a*                         $n \times n$  の行列  $A^{-1}$  によって上書きされる。

*info*                    INTEGER。  
*info* = 0 の場合、実行は正常に終了した。  
*info* = -*i* の場合、*i* 番目のパラメータの値が不正である。  
*info* = *i* の場合、*D* の *i* 番目の対角成分がゼロで、*D* が特異になるため、行列の反転を完了できない。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

sytri ルーチンのインターフェイス特有の詳細を以下に示す。

*a*                        サイズ (*n*,*n*) の行列 *A* を格納する。

*ipiv*                    長さ (*n*) のベクトルを格納する。

*uplo*                    'U' または 'L' でなければならない。デフォルト値は 'U' である。

### アプリケーション・ノート

算出された逆行列 *X* は、以下の誤差範囲を満たす。

$$|DU^T P^T XPU - I| \leq c(n)\varepsilon (|D||U^T|P^T|X|P|U| + |D||D^{-1}|)$$

*uplo* = 'U' の場合

$$|DL^T P^T XPL - I| \leq c(n)\varepsilon (|D||L^T|P^T|X|P|L| + |D||D^{-1}|)$$

*uplo* = 'L' の場合  $c(n)$  は *n* の適度な 1 次関数で、 $\varepsilon$  はマシンの精度である。*I* は元の行列を示す。



浮動小数点演算の合計回数は、実数型の場合は約  $(2/3)n^3$  で、複素数型の場合は約  $(8/3)n^3$  になる。

## ?hetri

複素エルミート行列の逆行列を計算する。

### 構文

#### Fortran 77:

```
call chetri(uplo, n, a, lda, ipiv, work, info)
call zhetri(uplo, n, a, lda, ipiv, work, info)
```

#### Fortran 95:

```
call hetri(a, ipiv [,uplo] [,info])
```

### 説明

このルーチンは、複素エルミート行列  $A$  の逆行列 ( $A^{-1}$ ) を計算する。  
このルーチン呼び出す前に、[?hetrf](#) を呼び出して、 $A$  を因子分解する。

### 入力パラメータ

<code>uplo</code>	CHARACTER*1。'U' または 'L' でなければならない。 入力行列 $A$ の因子分解の方法を指定する。 <code>uplo = 'U'</code> の場合、配列 <code>a</code> には、Bunch-Kaufman 因子分解 $A = PUDU^H P^T$ を格納する。 <code>uplo = 'L'</code> の場合、配列 <code>a</code> には、Bunch-Kaufman 因子分解 $A = PLDL^H P^T$ を格納する。
<code>n</code>	INTEGER。行列 $A$ の次数 ( $n \geq 0$ )。
<code>a, work</code>	COMPLEX (chetri の場合 ) DOUBLE COMPLEX (zhetri の場合 )。 配列 :  <code>a(lda, *)</code> には、 <a href="#">?hetrf</a> によって返される、行列 $A$ の因子分解を格納する。 <code>a</code> の第 2 次元は $\max(1, n)$ 以上でなければならない。

$work(*)$  は、ワークスペース配列。  
 $work$  の次元は  $\max(1, n)$  でなければならない。

$lda$                     INTEGER。  $a$  の第 1 次元。  $lda \geq \max(1, n)$ 。

$ipiv$                     INTEGER。  
 配列、次元は  $\max(1, n)$  以上。  
[zhetrf](#) によって返される  $ipiv$  配列。

#### 出力パラメータ

$a$                          $n \times n$  の行列  $A^{-1}$  によって上書きされる。

$info$                     INTEGER。  
 $info = 0$  の場合、実行は正常に終了した。  
 $info = -i$  の場合、 $i$  番目のパラメータの値が不正である。  
 $info = i$  の場合、 $D$  の  $i$  番目の対角成分がゼロで、 $D$  が特異になるため、行列の反転を完了できない。

#### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

hetri ルーチンのインターフェイス特有の詳細を以下に示す。

$a$                         サイズ  $(n, n)$  の行列  $A$  を格納する。

$ipiv$                     長さ  $(n)$  のベクトルを格納する。

$uplo$                     'U' または 'L' でなければならない。デフォルト値は 'U' である。

#### アプリケーション・ノート

算出された逆行列  $X$  は、以下の誤差範囲を満たす。

$$|DU^H P^T XPU - I| \leq c(n)\varepsilon (|D||U^H|P^T|X|P|U| + |D||D^{-1}|)$$

$uplo = 'U'$  の場合

$$|DL^H P^T XPL - I| \leq c(n)\varepsilon (|D||L^H|P^T|X|P|L| + |D||D^{-1}|)$$

$uplo = 'L'$  の場合  $c(n)$  は  $n$  の適度な 1 次関数で、 $\varepsilon$  はマシンの精度である。 $I$  は元の行列を示す。

浮動小数点演算の合計回数は、実数型の場合は約  $(2/3)n^3$  で、複素数型の場合は約  $(8/3)n^3$  になる。

このルーチンの実数版は、[?sytri](#) である。

## ?sptri

圧縮格納形式の対称行列の逆行列を計算する。

### 構文

#### Fortran 77:

```
call ssptri(uplo, n, ap, ipiv, work, info)
call dsptri(uplo, n, ap, ipiv, work, info)
call csptri(uplo, n, ap, ipiv, work, info)
call zsptri(uplo, n, ap, ipiv, work, info)
```

#### Fortran 95:

```
call sptri(a, ipiv [,uplo] [,info])
```

### 説明

このルーチンは、圧縮格納形式の対称行列  $A$  の逆行列 ( $A^{-1}$ ) を計算する。  
このルーチンを呼び出す前に、[?spturf](#) を呼び出して、 $A$  を因子分解する。

### 入力パラメータ

<code>uplo</code>	CHARACTER*1。'U' または 'L' でなければならない。 入力行列 $A$ の因子分解の方法を指定する。 <code>uplo = 'U'</code> の場合、配列 <code>ap</code> には、Bunch-Kaufman 因子分解 $A = PUDU^T P^T$ を格納する。 <code>uplo = 'L'</code> の場合、配列 <code>ap</code> には、Bunch-Kaufman 因子分解 $A = PLDL^T P^T$ を格納する。
<code>n</code>	INTEGER。行列 $A$ の次数 ( $n \geq 0$ )。

*ap, work* REAL (ssptri の場合 )  
DOUBLE PRECISION (dsptri の場合 )  
COMPLEX (csptri の場合 )  
DOUBLE COMPLEX (zsptri の場合 )。  
配列：  
*ap*(\*) には、[?spturf](#) によって返される、行列 *A* の因子分解を格納する。  
*ap* の次元は  $\max(1, n(n+1)/2)$  以上でなければならない。  
*work*(\*) は、ワークスペース配列。  
*work* の次元は  $\max(1, n)$  でなければならない。

*ipiv* INTEGER。  
配列、次元は  $\max(1, n)$  以上。  
[?spturf](#) によって返される *ipiv* 配列。

### 出力パラメータ

*ap* *ap* 圧縮形式の  $n \times n$  の行列  $A^{-1}$  によって上書きされる。

*info* INTEGER。  
*info* = 0 の場合、実行は正常に終了した。  
*info* = -*i* の場合、*i* 番目のパラメータの値が不正である。  
*info* = *i* の場合、*D* の *i* 番目の対角成分がゼロで、*D* が特異になるため、行列の反転を完了できない。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

sptri ルーチンのインターフェイス特有の詳細を以下に示す。

*a* Fortran 77 インターフェイスでの引数 *ap* を意味する。サイズ  $(n*(n+1)/2)$  の配列 *A* を格納する。

*ipiv* 長さ (*n*) のベクトルを格納する。

*uplo* 'U' または 'L' でなければならない。デフォルト値は 'U' である。

## アプリケーション・ノート

算出された逆行列  $X$  は、以下の誤差範囲を満たす。

$$|DU^T P^T XPU - I| \leq c(n)\varepsilon (|D||U^T|P^T|X|P|U| + |D||D^{-1}|)$$

$uplo = 'U'$  の場合

$$|DL^T P^T XPL - I| \leq c(n)\varepsilon (|D||L^T|P^T|X|P|L| + |D||D^{-1}|)$$

$uplo = 'L'$  の場合  $c(n)$  は  $n$  の適度な 1 次関数で、 $\varepsilon$  はマシンの精度である。 $I$  は元の行列を示す。

浮動小数点演算の合計回数は、実数型の場合は約  $(2/3)n^3$  で、複素数型の場合は約  $(8/3)n^3$  になる。

---

## ?hptri

圧縮格納形式の複素エルミート行列の逆行列を計算する。

---

### 構文

#### Fortran 77:

```
call chptri(uplo, n, ap, ipiv, work, info)
call zhptri(uplo, n, ap, ipiv, work, info)
```

#### Fortran 95:

```
call hptri(a, ipiv [,uplo] [,info])
```

### 説明

このルーチンは、圧縮格納形式の複素エルミート行列  $A$  の逆行列 ( $A^{-1}$ ) を計算する。  
このルーチン呼び出す前に、[?hptrf](#) を呼び出して、 $A$  を因子分解する。

### 入力パラメータ

**uplo** CHARACTER\*1。'U' または 'L' でなければならない。  
入行列  $A$  の因子分解の方法を指定する。

$uplo = 'U'$  の場合、配列  $ap$  には、圧縮形式の Bunch-Kaufman 因子分解  $A = PUDU^H P^T$  を格納する。

$uplo = 'L'$  の場合、配列  $ap$  には、圧縮形式の Bunch-Kaufman 因子分解  $A = PLDL^H P^T$  を格納する。

$n$  INTEGER。行列  $A$  の次数 ( $n \geq 0$ )。

$ap$  COMPLEX (chptri の場合)  
DOUBLE COMPLEX (zhptri の場合)。  
配列:

$ap(*)$  には、[?hptrf](#) によって返される、行列  $A$  の因子分解を格納する。

$ap$  の次元は  $\max(1, n(n+1)/2)$  以上でなければならない。

$work(*)$  は、ワークスペース配列。

$work$  の次元は  $\max(1, n)$  でなければならない。

$ipiv$  INTEGER。  
配列、次元は  $\max(1, n)$  以上。  
[?hptrf](#) によって返される  $ipiv$  配列。

### 出力パラメータ

$ap$   $n \times n$  の行列  $A^{-1}$  によって上書きされる。

$info$  INTEGER。  
 $info = 0$  の場合、実行は正常に終了した。  
 $info = -i$  の場合、 $i$  番目のパラメータの値が不正である。  
 $info = i$  の場合、 $D$  の  $i$  番目の対角成分がゼロで、 $D$  が特異になるため、行列の反転を完了できない。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

hptri ルーチンのインターフェイス特有の詳細を以下に示す。

$a$  Fortran 77 インターフェイスでの引数  $ap$  を意味する。サイズ  $(n*(n+1)/2)$  の配列  $A$  を格納する。

$ipiv$  長さ ( $n$ ) のベクトルを格納する。

$uplo$  'U' または 'L' でなければならない。デフォルト値は 'U' である。

## アプリケーション・ノート

算出された逆行列  $X$  は、以下の誤差範囲を満たす。

$$|DU^H P^T XPU - I| \leq c(n)\varepsilon (|D||U^H|P^T|X|P|U| + |D||D^{-1}|)$$

$uplo = 'U'$  の場合

$$|DL^H P^T XPL - I| \leq c(n)\varepsilon (|D||L^H|P^T|X|P|L| + |D||D^{-1}|)$$

$uplo = 'L'$  の場合  $c(n)$  は  $n$  の適度な 1 次関数で、 $\varepsilon$  はマシンの精度である。 $I$  は元の行列を示す。

浮動小数点演算の合計回数は、実数型の場合は約  $(2/3)n^3$  で、複素数型の場合は約  $(8/3)n^3$  になる。

このルーチンの実数版は、[?sptri](#) である。

---

## ?trtri

三角行列の逆行列を計算する。

---

### 構文

#### Fortran 77:

```
call strtri(uplo, diag, n, a, lda, info)
call dtrtri(uplo, diag, n, a, lda, info)
call ctrtri(uplo, diag, n, a, lda, info)
call ztrtri(uplo, diag, n, a, lda, info)
```

#### Fortran 95:

```
call trtri(a [,uplo] [,diag] [,info])
```

### 説明

このルーチンは、三角行列  $A$  の逆行列 ( $A^{-1}$ ) を計算する。

### 入力パラメータ

**uplo** CHARACTER\*1。'U' または 'L' でなければならない。  
 $A$  が上三角行列か、下三角行列かを指定する。

*diag*      *uplo* = 'U' の場合、*A* は上三角行列である。  
             *uplo* = 'L' の場合、*A* は下三角行列である。  
CHARACTER\*1。'N' または 'U' でなければならない。  
*diag* = 'N' の場合、*A* は単位三角行列ではない。  
  
*diag* = 'U' の場合、*A* は単位三角行列である。*A* の対角成分は 1 とみなされ、配列 *a* 内で参照されない。  
  
*n*            INTEGER。行列 *A* の次数 ( $n \geq 0$ )。  
  
*a*            REAL (*strtri* の場合 )  
             DOUBLE PRECISION (*dtrtri* の場合 )  
             COMPLEX (*ctrtri* の場合 )  
             DOUBLE COMPLEX (*ztrtri* の場合 )。  
  
             配列、次元は (*lda*, \*)。  
             行列 *A* を格納する。*a* の第 2 次元は  $\max(1, n)$  以上でなければならない。  
  
*lda*          INTEGER。*a* の第 1 次元。  $lda \geq \max(1, n)$ 。

### 出力パラメータ

*a*             $n \times n$  の行列  $A^{-1}$  によって上書きされる。  
  
*info*          INTEGER。  
             *info* = 0 の場合、実行は正常に終了した。  
             *info* = -*i* の場合、*i* 番目のパラメータの値が不正である。  
             *info* = *i* の場合、*A* の *i* 番目の対角成分がゼロで、*A* が特異になるため、反転を完了できない。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

*trtri* ルーチンのインターフェイス特有の詳細を以下に示す。

*a*            サイズ (*n*,*n*) の行列 *A* を格納する。  
  
*uplo*          'U' または 'L' でなければならない。デフォルト値は 'U' である。  
  
*diag*          'N' または 'U' でなければならない。デフォルト値は 'N' である。



## アプリケーション・ノート

算出された逆行列  $X$  は、以下の誤差範囲を満たす。

$$|XA - I| \leq c(n)\varepsilon |X||A|$$

$$|X - A^{-1}| \leq c(n)\varepsilon |A^{-1}||A||X|$$

$c(n)$  は  $n$  の適度な 1 次関数で、 $\varepsilon$  はマシンの精度である。

$I$  は元の行列を示す。

浮動小数点演算の合計回数は、実数型の場合は約  $(1/3)n^3$  で、複素数型の場合は約  $(4/3)n^3$  になる。

---

## ?tptri

圧縮格納形式の三角行列の逆行列を計算する。

---

### 構文

#### Fortran 77:

```
call stptri(uplo, diag, n, ap, info)
call dtptri(uplo, diag, n, ap, info)
call ctptri(uplo, diag, n, ap, info)
call ztptri(uplo, diag, n, ap, info)
```

#### Fortran 95:

```
call tptri(a [,uplo] [,diag] [,info])
```

### 説明

このルーチンは、圧縮格納形式の三角行列  $A$  の逆行列 ( $A^{-1}$ ) を計算する。

### 入力パラメータ

**uplo** CHARACTER\*1。'U' または 'L' でなければならない。  
 $A$  が上三角行列か、下三角行列かを指定する。  
  
 $uplo = 'U'$  の場合、 $A$  は上三角行列である。  
 $uplo = 'L'$  の場合、 $A$  は下三角行列である。

<i>diag</i>	CHARACTER*1。'N' または 'U' でなければならない。 <i>diag</i> = 'N' の場合、 <i>A</i> は単位三角行列ではない。 <i>diag</i> = 'U' の場合、 <i>A</i> は単位三角行列である。 <i>A</i> の対角成分は 1 とみなされ、配列 <i>ap</i> 内で参照されない。
<i>n</i>	INTEGER。行列 <i>A</i> の次数 ( $n \geq 0$ )。
<i>ap</i>	REAL ( <i>stptri</i> の場合 ) DOUBLE PRECISION ( <i>dtptri</i> の場合 ) COMPLEX ( <i>ctptri</i> の場合 ) DOUBLE COMPLEX ( <i>ztptri</i> の場合 )。  配列、次元は $\max(1, n(n+1)/2)$ 以上。 圧縮形式の三角行列 <i>A</i> を格納する。

### 出力パラメータ

<i>ap</i>	圧縮形式の $n \times n$ の行列 $A^{-1}$ によって上書きされる。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了した。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正である。 <i>info</i> = <i>i</i> の場合、 <i>A</i> の <i>i</i> 番目の対角成分がゼロで、 <i>A</i> が特異になるため、反転を完了できない。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

*tptri* ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ap</i> を意味する。サイズ $(n*(n+1)/2)$ の配列 <i>A</i> を格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U' である。
<i>diag</i>	'N' または 'U' でなければならない。デフォルト値は 'N' である。

### アプリケーション・ノート

算出された逆行列  $X$  は、以下の誤差範囲を満たす。

$$|XA - I| \leq c(n)\varepsilon |X||A|$$

$$|X - A^{-1}| \leq c(n)\varepsilon |A^{-1}||A||X|$$

$c(n)$  は  $n$  の適度な 1 次関数で、 $\varepsilon$  はマシンの精度である。

$I$  は元の行列を示す。

浮動小数点演算の合計回数は、実数型の場合は約  $(1/3)n^3$  で、複素数型の場合は約  $(4/3)n^3$  になる。

### 行列の平衡化

この節では、行列の平衡化に必要なスケール係数の計算に使用されるルーチンについて説明する。ただし、これらのルーチンが行列を実際にスケーリングするわけではない。

---

## ?geequ

行列を平衡化して、条件数を小さくするための  
行と列のスケール係数を計算する。

---

### 構文

#### Fortran 77:

```
call sgeequ(m, n, a, lda, r, c, rowcnd, colcnd, amax, info)
call dgeequ(m, n, a, lda, r, c, rowcnd, colcnd, amax, info)
call cgeequ(m, n, a, lda, r, c, rowcnd, colcnd, amax, info)
call zgeequ(m, n, a, lda, r, c, rowcnd, colcnd, amax, info)
```

#### Fortran 95:

```
call geequ(a, r, c [,rowcnd] [,colcnd] [,amax] [,info])
```

## 説明

このルーチンは、 $m \times n$  行列  $A$  を平衡化して条件数を小さくするための、行と列のスケール係数を計算する。出力配列  $r$  に行のスケール係数を返し、配列  $c$  に列のスケール係数を返す。これらの係数は、成分  $b_{ij}=r(i)*a_{ij}*c(j)$  を持つ行列  $B$  の各行と各列の最大の成分の絶対値が 1 になるように選択される。

?sgeequ により計算されるスケール係数を使用する [?laqge](#) 補助関数を参照のこと。

## 入力パラメータ

$m$  INTEGER。行列  $A$  の行数。  $m \geq 0$ 。

$n$  INTEGER。行列  $A$  の列数。  $n \geq 0$ 。

$a$  REAL (sgeequ の場合 )  
DOUBLE PRECISION (dgeequ の場合 )  
COMPLEX (cgeequ の場合 )  
DOUBLE COMPLEX (zgeequ の場合 )。

配列、次元は ( $lda$ , \*)。

平衡化係数を計算する  $m \times n$  行列  $A$  を格納する。  
 $a$  の第 2 次元は  $\max(1, n)$  以上でなければならない。

$lda$  INTEGER。  $a$  のリーディング・ディメンジョン。  $lda \geq \max(1, m)$ 。

## 出力パラメータ

$r, c$  REAL ( 単精度の場合 )  
DOUBLE PRECISION ( 倍精度の場合 )。  
配列 :  $r(m), c(n)$ 。  
 $info = 0$  または  $info > m$  の場合、配列  $r$  に行列  $A$  の行のスケール係数が格納される。  
 $info = 0$  の場合、配列  $c$  に行列  $A$  の列のスケール係数が格納される。

$rowcnd$  REAL ( 単精度の場合 )  
DOUBLE PRECISION ( 倍精度の場合 )。  
 $info = 0$  または  $info > m$  の場合、 $rowcnd$  に、最小の  $r(i)$  を最大の  $r(i)$  で割った値が格納される。

$colcnd$  REAL ( 単精度の場合 )  
DOUBLE PRECISION ( 倍精度の場合 )。  
 $info = 0$  の場合、 $colcnd$  に、最小の  $c(i)$  を最大の  $c(i)$  で割った値が格納される。

<i>amax</i>	REAL ( 単精度の場合 ) DOUBLE PRECISION ( 倍精度の場合 )。 行列 <i>A</i> の最大の成分の絶対値。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了した。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正である。 <i>info</i> = <i>i</i> 、かつ <i>i</i> ≤ <i>m</i> の場合、 <i>A</i> の <i>i</i> 番目の行が完全に 0 である。 <i>i</i> > <i>m</i> の場合、 <i>A</i> の ( <i>i</i> - <i>m</i> ) 番目の列が完全に 0 である。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

geequ ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	サイズ ( <i>m</i> , <i>n</i> ) の行列 <i>A</i> を格納する。
<i>r</i>	長さ ( <i>m</i> ) のベクトルを格納する。
<i>c</i>	長さ ( <i>n</i> ) のベクトルを格納する。

### アプリケーション・ノート

*r* と *c* のすべての成分は、SMLNUM = 最小の安全な数値と BIGNUM = 最大の安全な数値の範囲内に制限される。これらのスケール係数を使用しても、*A* の条件数が小さくなることは保証されないが、実際には有効に機能する。

*rowcnd* ≥ 0.1 であり、*amax* が大きすぎる値でも小さすぎる値でもない場合は、*r* によるスケールリングは不要である。*colcnd* ≥ 0.1 の場合、*c* によるスケールリングは不要である。

*amax* の値がオーバーフローまたはアンダーフローに非常に近い場合は、行列 *A* をスケールリングする必要がある。

### ?gbequ

帯行列を平衡化して条件数を小さくするための、  
行と列のスケール係数を計算する。

---

#### 構文

##### Fortran 77:

```
call sgbequ(m, n, kl, ku, ab, ldab, r, c, rowcnd, colcnd, amax, info)
call dgbequ(m, n, kl, ku, ab, ldab, r, c, rowcnd, colcnd, amax, info)
call cgbequ(m, n, kl, ku, ab, ldab, r, c, rowcnd, colcnd, amax, info)
call zgbequ(m, n, kl, ku, ab, ldab, r, c, rowcnd, colcnd, amax, info)
```

##### Fortran 95:

```
call gbequ(a, r, c [,kl] [,rowcnd] [,colcnd] [,amax] [,info])
```

#### 説明

このルーチンは、 $m \times n$  の帯行列  $A$  を平衡化して、条件数を小さくするための、行と列のスケール係数を計算する。出力配列  $r$  に行のスケール係数を返し、配列  $c$  に列のスケール係数を返す。これらの係数は、成分  $b_{ij}=r(i)*a_{ij}*c(j)$  を持つ行列  $B$  の各行と各列の最大の成分の絶対値が 1 になるように選択される。

?gbequ により計算されるスケール係数を使用する [?laggb](#) 補助関数を参照のこと。

#### 入力パラメータ

$m$	INTEGER。行列 $A$ の行数。 $m \geq 0$ 。
$n$	INTEGER。行列 $A$ の列数。 $n \geq 0$ 。
$kl$	INTEGER。 $A$ の帯内の劣対角成分の数 ( $kl \geq 0$ )。
$ku$	INTEGER。 $A$ の帯内の優対角成分の数 ( $ku \geq 0$ )。
$ab$	REAL (sgbequ の場合 ) DOUBLE PRECISION (dgbequ の場合 ) COMPLEX (cgbequ の場合 ) DOUBLE COMPLEX (zgbequ の場合 )。  配列、次元は ( $ldab, *$ )。 行 $1 \sim kl + ku + 1$ に格納される、元の帯行列 $A$ を格納する。  $ab$ の第 2 次元は $\max(1, n)$ 以上でなければならない。

*ldab* INTEGER。 *ab* のリーディング・ディメンジョン。  $ldab \geq kl+ku+1$ 。

### 出力パラメータ

*r, c* REAL ( 単精度の場合 )  
DOUBLE PRECISION ( 倍精度の場合 )。  
配列 :  $r(m), c(n)$ 。  
 $info = 0$  または  $info > m$  の場合、配列 *r* に行列 *A* の行のスケール係数が格納される。  
 $info = 0$  の場合、配列 *c* に行列 *A* の列のスケール係数が格納される。

*rowcnd* REAL ( 単精度の場合 )  
DOUBLE PRECISION ( 倍精度の場合 )。  
 $info = 0$  または  $info > m$  の場合、*rowcnd* に、最小の  $r(i)$  を最大の  $r(i)$  で割った値が格納される。

*colcnd* REAL ( 単精度の場合 )  
DOUBLE PRECISION ( 倍精度の場合 )。  
 $info = 0$  の場合、*colcnd* に、最小の  $c(i)$  を最大の  $c(i)$  で割った値が格納される。

*amax* REAL ( 単精度の場合 )  
DOUBLE PRECISION ( 倍精度の場合 )。  
行列 *A* の最大の成分の絶対値。

*info* INTEGER。  
 $info = 0$  の場合、実行は正常に終了した。  
 $info = -i$  の場合、*i* 番目のパラメータの値が不正である。  
 $info = i$ 、かつ  
     $i \leq m$  の場合、*A* の *i* 番目の行が完全に 0 である。  
     $i > m$  の場合、*A* の (*i-m*) 番目の列が完全に 0 である。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

gbequ ルーチンのインターフェイス特有の詳細を以下に示す。

*a* Fortran 77 インターフェイスでの引数 *ab* を意味する。サイズ  $(kl+ku+1, n)$  の配列 *A* を格納する。

*r* 長さ (*m*) のベクトルを格納する。

$c$                       長さ ( $n$ ) のベクトルを格納する。  
 $kl$                       省略した場合、 $kl = ku$  とみなす。  
 $ku$                        $ku = lda - kl - 1$  として復元する。

### アプリケーション・ノート

$r$  と  $c$  のすべての成分は、**SMLNUM** = 最小の安全な数値と **BIGNUM** = 最大の安全な数値の範囲内に制限される。これらのスケール係数を使用しても、 $A$  の条件数が小さくなることは保証されないが、実際には有効に機能する。

$rowcnd \geq 0.1$  であり、 $amax$  が大きすぎる値でも小さすぎる値でもない場合は、 $r$  によるスケールリングは不要である。 $colcnd \geq 0.1$  の場合、 $c$  によるスケールリングは不要である。

$amax$  の値がオーバーフローまたはアンダーフローに非常に近い場合は、行列  $A$  をスケールリングする必要がある。

---

## ?poequ

対称 (エルミート) 正定値行列を平衡化して条件数を小さくするための、行と列のスケール係数を計算する。

---

### 構文

#### Fortran 77:

```
call spoequ(n, a, lda, s, scond, amax, info)
call dpoequ(n, a, lda, s, scond, amax, info)
call cpoequ(n, a, lda, s, scond, amax, info)
call zpoequ(n, a, lda, s, scond, amax, info)
```

#### Fortran 95:

```
call poequ(a, s [,scond] [,amax] [,info])
```

### 説明

このルーチンは、対称 (エルミート) 正定値行列  $A$  を平衡化して (2- ノルムに関して) 条件数を小さくするための、行と列のスケール係数を計算する。出力配列  $s$  に、次のように計算されたスケール係数を返す。



$$s(i) = \frac{1}{\sqrt{a_{i,i}}}$$

これらの係数は、成分  $b_{ij}=s(i)*a_{ij}*s(j)$  を持つスケーリング後の行列  $B$  の対角成分が 1 に等しくなるように選択される。

$s$  をこのように選択すると、 $B$  の条件数は、すべての可能な対角スケーリングで、可能な最小の条件数に係数  $n$  を掛けた値の範囲内に収まる。

?poequ により計算されるスケール係数を使用する [?lagsy](#) 補助関数を参照のこと。

### 入力パラメータ

**n** INTEGER。行列  $A$  の次数 ( $n \geq 0$ )。

**a** REAL (spoequ の場合)  
DOUBLE PRECISION (dpoequ の場合)  
COMPLEX (cpoequ の場合)  
DOUBLE COMPLEX (zpoequ の場合)。

配列、次元は ( $lda, *$ )。  
スケール係数を計算する、 $n \times n$  の対称 / エルミート正定値行列  $A$  を格納する。 $A$  の対角成分だけが参照される。  
 $a$  の第 2 次元は  $\max(1, n)$  以上でなければならない。

**lda** INTEGER。  $a$  のリーディング・ディメンジョン。  $lda \geq \max(1, m)$ 。

### 出力パラメータ

**s** REAL (単精度の場合)  
DOUBLE PRECISION (倍精度の場合)。  
配列、次元は ( $n$ )。  
 $info = 0$  の場合、配列  $s$  に  $A$  のスケール係数が格納される。

**scond** REAL (単精度の場合)  
DOUBLE PRECISION (倍精度の場合)。  
 $info = 0$  の場合、 $scond$  に、最小の  $s(i)$  を最大の  $s(i)$  で割った値が格納される。

**amax** REAL (単精度の場合)  
DOUBLE PRECISION (倍精度の場合)。  
行列  $A$  の最大の成分の絶対値。

*info*                    INTEGER。  
                      *info* = 0 の場合、実行は正常に終了した。  
                      *info* = -*i* の場合、*i* 番目のパラメータの値が不正である。  
                      *info* = *i* の場合、*A* の *i* 番目の対角成分が正の値でない。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

poequ ルーチンのインターフェイス特有の詳細を以下に示す。

*a*                    サイズ (*n*,*n*) の行列 *A* を格納する。

*s*                    長さ (*n*) のベクトルを格納する。

### アプリケーション・ノート

*scond* ≥ 0.1 であり、*amax* が大きすぎる値でも小さすぎる値でもない場合は、*s* によるスケールリングは不要である。

*amax* の値がオーバーフローまたはアンダーフローに非常に近い場合は、行列 *A* をスケールリングする必要がある。

---

## ?ppequ

圧縮形式の対称(エルミート)正定値行列を  
平衡化して、条件数を小さくするための行と  
列のスケール係数を計算する。

---

### 構文

#### Fortran 77:

```
call sppequ(uplo, n, ap, s, acond, amax, info)
call dppequ(uplo, n, ap, s, acond, amax, info)
call cppequ(uplo, n, ap, s, acond, amax, info)
call zppequ(uplo, n, ap, s, acond, amax, info)
```

**Fortran 95:**

```
call ppequ(a, s [,scond] [,amax] [,uplo] [,info])
```

**説明**

このルーチンは、圧縮形式の対称 (エルミート) 正定値行列  $A$  を平衡化して (2- ノルムに関して) 条件数を小さくするための、行と列のスケール係数を計算する。出力配列  $s$  に、次のように計算されたスケール係数を返す。

$$s(i) = \frac{1}{\sqrt{a_{i,i}}}$$

これらの係数は、成分  $b_{ij}=s(i)*a_{ij}*s(j)$  を持つスケーリング後の行列  $B$  の対角成分が 1 に等しくなるように選択される。

$s$  をこのように選択すると、 $B$  の条件数は、すべての可能な対角スケーリングで、可能な最小の条件数に係数  $n$  を掛けた値の範囲内に収まる。

?ppequ により計算されるスケール係数を使用する [?lagsp](#) 補助関数を参照のこと。

**入力パラメータ**

**uplo** CHARACTER\*1。'U' または 'L' でなければならない。  
 $A$  の上三角部分と下三角部分のどちらを配列  $ap$  にパックするかを指定する。  
**uplo** = 'U' の場合、配列  $ap$  には行列  $A$  の上三角部分が格納される。  
**uplo** = 'L' の場合、配列  $ap$  には行列  $A$  の下三角部分が格納される。

**n** INTEGER。行列  $A$  の次数 ( $n \geq 0$ )。

**ap** REAL (sppequ の場合)  
 DOUBLE PRECISION (dppequ の場合)  
 COMPLEX (cppequ の場合)  
 DOUBLE COMPLEX (zppequ の場合)。  
 配列、次元は  $\max(1, n(n+1)/2)$  以上。  
 配列  $ap$  には、(**uplo** の指定に従って) 行列  $A$  の上三角部分または下三角部分を圧縮格納形式で格納する (「[行列の格納形式](#)」を参照)。

**出力パラメータ**

**s** REAL (単精度の場合)  
 DOUBLE PRECISION (倍精度の場合)。  
 配列、次元は ( $n$ )。  
**info** = 0 の場合、配列  $s$  に  $A$  のスケール係数が格納される。

<i>scond</i>	REAL ( 単精度の場合 ) DOUBLE PRECISION ( 倍精度の場合 )。 <i>info</i> = 0 の場合、 <i>scond</i> に、最小の <i>s</i> ( <i>i</i> ) を最大の <i>s</i> ( <i>i</i> ) で割った値が格納される。
<i>amax</i>	REAL ( 単精度の場合 ) DOUBLE PRECISION ( 倍精度の場合 )。 行列 <i>A</i> の最大の成分の絶対値。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了した。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正である。 <i>info</i> = <i>i</i> の場合、 <i>A</i> の <i>i</i> 番目の対角成分が正の値でない。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ppequ ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ap</i> を意味する。サイズ $(n*(n+1)/2)$ の配列 <i>A</i> を格納する。
<i>s</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U' である。

### アプリケーション・ノート

*scond*  $\geq 0.1$  であり、*amax* が大きすぎる値でも小さすぎる値でもない場合は、*s* によるスケーリングは不要である。

*amax* の値がオーバーフローまたはアンダーフローに非常に近い場合は、行列 *A* をスケーリングする必要がある。

## ?pbequ

対称 (エルミート) 正定値帯行列を平衡化して、条件数を小さくするための行と列のスケール係数を計算する。

### 構文

#### Fortran 77:

```
call spbequ(uplo, n, kd, ab, ldab, s, scond, amax, info)
call dpbequ(uplo, n, kd, ab, ldab, s, scond, amax, info)
call cpbequ(uplo, n, kd, ab, ldab, s, scond, amax, info)
call zpbequ(uplo, n, kd, ab, ldab, s, scond, amax, info)
```

#### Fortran 95:

```
call pbequ(a, s [,scond] [,amax] [,uplo] [,info])
```

### 説明

このルーチンは、圧縮形式の対称 (エルミート) 正定値行列  $A$  を平衡化して (2- ノルムに関して) 条件数を小さくするための、行と列のスケール係数を計算する。出力配列  $s$  に、次のように計算されたスケール係数を返す。

$$s(i) = \frac{1}{\sqrt{a_{i,i}}}$$

これらの係数は、成分  $b_{ij}=s(i)*a_{ij}*s(j)$  を持つスケーリング後の行列  $B$  の対角成分が 1 に等しくなるように選択される。

$s$  をこのように選択すると、 $B$  の条件数は、すべての可能な対角スケーリングで、可能な最小の条件数に係数  $n$  を掛けた値の範囲内に収まる。

?pbequ により計算されるスケール係数を使用する [?laqsb](#) 補助関数を参照のこと。

### 入力パラメータ

**uplo** CHARACTER\*1。'U' または 'L' でなければならない。  
 $A$  の上三角部分と下三角部分のどちらを配列 **ab** にパックするかを指定する。  
**uplo** = 'U' の場合、配列 **ab** には行列  $A$  の上三角部分が格納される。  
**uplo** = 'L' の場合、配列 **ab** には行列  $A$  の下三角部分が格納される。

<i>n</i>	INTEGER。行列 <i>A</i> の次数 ( $n \geq 0$ )。
<i>kd</i>	INTEGER。 <i>A</i> の優対角成分または劣対角成分の数 ( $kd \geq 0$ )。
<i>ab</i>	REAL (spbequ の場合 ) DOUBLE PRECISION (dpbequ の場合 ) COMPLEX (cpbequ の場合 ) DOUBLE COMPLEX (zpbequ の場合 )。 配列、次元は ( <i>ldab</i> , *)。 配列 <i>ap</i> には、( <i>uplo</i> で指定された ) 行列 <i>A</i> の上三角部分または下三角部分が <b>帯形式</b> で格納される (「 <a href="#">行列の格納形式</a> 」を参照)。 <i>ab</i> の第 2 次元は $\max(1, n)$ 以上でなければならない。
<i>ldab</i>	INTEGER。 <i>ab</i> のリーディング・ディメンジョン ( $ldab \geq kd + 1$ )。

### 出力パラメータ

<i>s</i>	REAL ( 単精度の場合 ) DOUBLE PRECISION ( 倍精度の場合 )。 配列、次元は ( <i>n</i> )。 <i>info</i> = 0 の場合、配列 <i>s</i> に <i>A</i> のスケール係数が格納される。
<i>scond</i>	REAL ( 単精度の場合 ) DOUBLE PRECISION ( 倍精度の場合 )。 <i>info</i> = 0 の場合、 <i>scond</i> に、最小の <i>s</i> ( <i>i</i> ) を最大の <i>s</i> ( <i>i</i> ) で割った値が格納される。
<i>amax</i>	REAL ( 単精度の場合 ) DOUBLE PRECISION ( 倍精度の場合 )。 行列 <i>A</i> の最大の成分の絶対値。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了した。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正である。 <i>info</i> = <i>i</i> の場合、 <i>A</i> の <i>i</i> 番目の対角成分が正の値でない。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

pbequ ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ab</i> を意味する。サイズ ( $kd+1, n$ ) の配列 <i>A</i> を格納する。
----------	---

*s*                      長さ (*n*) のベクトルを格納する。

*uplo*                  'U' または 'L' でなければならない。デフォルト値は 'U' である。

### アプリケーション・ノート

*scond*  $\geq 0.1$  であり、*amax* が大きすぎる値でも小さすぎる値でもない場合は、*s* によるスケーリングは不要である。

*amax* の値がオーバーフローまたはアンダーフローに非常に近い場合は、行列 *A* をスケーリングする必要がある。

## ドライバルーチン

表 3-3 に、実数または複素行列を係数行列とする連立 1 次方程式を解くための LAPACK ドライバルーチンを示す。

**表 3-3                  連立 1 次方程式の解の算出用のドライバルーチン**

行列のタイプ (格納形式)	簡易ドライバ	高度ドライバ
一般行列	<a href="#">?gesv</a>	<a href="#">?gesvx</a>
一般帯	<a href="#">?gbsv</a>	<a href="#">?gbsvx</a>
一般三重対角	<a href="#">?qtsv</a>	<a href="#">?qtsvx</a>
対称 / エルミート 正定値	<a href="#">?posv</a>	<a href="#">?posvx</a>
対称 / エルミート 正定値 (圧縮格納形式)	<a href="#">?ppsv</a>	<a href="#">?ppsvx</a>
対称 / エルミート 正定値 (帯形式)	<a href="#">?pbsv</a>	<a href="#">?pbsvx</a>
対称 / エルミート 正定値 (三重対角形式)	<a href="#">?ptsv</a>	<a href="#">?ptsvx</a>
対称 / エルミート 不定値	<a href="#">?sysv/?hesv</a>	<a href="#">?sysvx/?hesvx</a>
対称 / エルミート 不定値 (圧縮格納形式)	<a href="#">?spsv/?hpsv</a>	<a href="#">?spsvx/?hpsvx</a>
複素対称	<a href="#">?sysv</a>	<a href="#">?sysvx</a>
複素対称 (圧縮格納形式)	<a href="#">?spsv</a>	<a href="#">?spsvx</a>

表中の ? は、**s** (単精度実数)、**d** (倍精度実数)、**c** (単精度複素数)、または **z** (倍精度複素数) を示す。

## ?gesv

正方行列  $A$  を係数行列とする、複数の右辺を持つ  
連立 1 次方程式の解を計算する。

### 構文

#### Fortran 77:

```
call sgesv(n, nrhs, a, lda, ipiv, b, ldb, info)
call dgesv(n, nrhs, a, lda, ipiv, b, ldb, info)
call cgesv(n, nrhs, a, lda, ipiv, b, ldb, info)
call zgesv(n, nrhs, a, lda, ipiv, b, ldb, info)
```

#### Fortran 95:

```
call gesv(a, b [,ipiv] [,info])
```

### 説明

このルーチンは、連立 1 次方程式  $AX=B$  を  $X$  について解く。 $A$  は  $n \times n$  行列、行列  $B$  の列は個々の右辺、 $X$  の列はそれに対応する解である。

部分的なピボット演算と行の交換による  $LU$  分解を使用して、 $A$  を  $A=PLU$  として因子分解する。 $P$  は置換行列、 $L$  は単位下三角行列、 $U$  は上三角行列である。次に、 $A$  の因子分解された形式を使用して、連立方程式  $AX=B$  を解く。

### 入力パラメータ

$n$	INTEGER。 $A$ の次数。 $B$ の行数 ( $n \geq 0$ )。
$nrhs$	INTEGER。 右辺の数。 $B$ の列数 ( $nrhs \geq 0$ )。
$a, b$	REAL (sgesv の場合 ) DOUBLE PRECISION (dgesv の場合 ) COMPLEX (cgesv の場合 ) DOUBLE COMPLEX (zgesv の場合 )。 配列 : $a(lda, *)$ , $b(ldb, *)$ 。  配列 $a$ には、行列 $A$ が格納される。配列 $b$ には、行列 $B$ が格納される。この行列の列は、連立方程式の右辺である。 $a$ の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $b$ の第 2 次元は、 $\max(1, nrhs)$ 以上でなければならない。



*lda* INTEGER。 *a* の第 1 次元。  $lda \geq \max(1, n)$ 。

*ldb* INTEGER。 *b* の第 1 次元。  $ldb \geq \max(1, n)$ 。

### 出力パラメータ

*a*  $A = PLU$  の因子分解で得られた係数  $L$  と  $U$  によって上書きされる。 $L$  の単位対角成分は格納されない。

*b* 解の行列  $X$  によって上書きされる。

*ipiv* INTEGER。  
配列、次元は  $\max(1, n)$  以上。  
置換行列  $P$  を定義するピボットのインデックス。行列の行  $i$  は、行  $ipiv(i)$  と交換される。

*info* INTEGER。  $info = 0$  の場合、実行は正常に終了した。  
 $info = -i$  の場合、 $i$  番目のパラメータの値が不正である。  
 $info = i$  の場合、 $U(i, i)$  が完全に 0 である。因子分解は完了したが、係数  $U$  が完全に特異であるため、解を計算できなかった。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

gesv ルーチンのインターフェイス特有の詳細を以下に示す。

*a* サイズ  $(n, n)$  の行列  $A$  を格納する。

*b* サイズ  $(n, nrhs)$  の行列  $B$  を格納する。

*ipiv* 長さ  $(n)$  のベクトルを格納する。

## ?gesvx

正方行列  $A$  を係数行列とする、複数の右辺を持つ連立 1 次方程式の解を計算し、解の誤差範囲を示す。

---

### 構文

#### Fortran 77:

```
call sgesvx(FACT, TRANS, N, NRHS, A, LDA, AF, LDAF, IPIV, EQUED, R, C, B,  
            LDB, X, LDX, RCOND, FERR, BERR, WORK, IWORK, INFO)  
call dgesvx(FACT, TRANS, N, NRHS, A, LDA, AF, LDAF, IPIV, EQUED, R, C, B,  
            LDB, X, LDX, RCOND, FERR, BERR, WORK, IWORK, INFO)  
call cgesvx(FACT, TRANS, N, NRHS, A, LDA, AF, LDAF, IPIV, EQUED, R, C, B,  
            LDB, X, LDX, RCOND, FERR, BERR, WORK, rWORK, INFO)  
call zgesvx(FACT, TRANS, N, NRHS, A, LDA, AF, LDAF, IPIV, EQUED, R, C, B,  
            LDB, X, LDX, RCOND, FERR, BERR, WORK, rWORK, INFO)
```

#### Fortran 95:

```
call gesvx(a, b, x [,af] [,ipiv] [,fact] [,trans] [,equed] [,r] [,c]  
            [,ferr] [,berr] [,rcond] [,rpvgrw] [,info])
```

### 説明

このルーチンは、 $LU$  因子分解を使用して、実数または複素連立 1 次方程式  $AX=B$  の解を計算する。 $A$  は  $n \times n$  行列、行列  $B$  の列は個々の右辺、 $X$  の列はそれに対応する解である。

このルーチンは、解の誤差範囲と条件数の推定値も示す。

ルーチン ?gesvx は、以下の手順を実行する。

1.  $fact = 'E'$  の場合、連立方程式を平衡化するための実スケール係数  $r$  と  $c$  を計算する。

$trans = 'N'$ :  $diag(r) * A * diag(c) * diag(c)^{-1} * X = diag(r) * B$

$trans = 'T'$ :  $(diag(r) * A * diag(c))^T * diag(r)^{-1} * X = diag(c) * B$

$trans = 'C'$ :  $(diag(r) * A * diag(c))^H * diag(r)^{-1} * X = diag(c) * B$

連立方程式が平衡化されるかどうかは、行列  $A$  のスケーリングによって決まる。平衡化が行われる場合は、 $A$  は  $\text{diag}(r)*A*\text{diag}(c)$  によって上書きされ、 $B$  は  $\text{diag}(r)*B$  ( $\text{trans}='N'$  の場合) または  $\text{diag}(c)*B$  ( $\text{trans}='T'$  または  $'C'$  の場合) によって上書きされる。

2.  $\text{fact}='N'$  または  $'E'$  の場合、 $LU$  分解を使用して、( $\text{fact}='E'$  の場合は平衡化の後に) 行列  $A$  を  $A=PLU$  として因子分解する。 $P$  は置換行列、 $L$  は単位下三角行列、 $U$  は上三角行列である。

3.  $U_{i,i}=0$  の場合、つまり  $U$  が完全に特異である場合は、ルーチンは  $\text{info}=i$  を返す。それ以外の場合、 $A$  の因子分解された形式を使用して、行列  $A$  の条件数を推定する。条件数の逆数がマシンの精度より小さい場合は、警告として  $\text{info}=n+1$  を返すが、後で説明するようにルーチンは続行され、 $X$  について解を算出し、誤差範囲を計算する。

4.  $A$  の因子分解された形式を使用して、連立方程式を  $X$  について解く。

5. 計算された解の行列の精度を繰り返し改善し、解の誤差範囲と後退誤差の推定値を計算する。

6. 平衡化が行われている場合は、平衡化が行われる前の元の連立方程式を解くために、 $\text{diag}(c)$  ( $\text{trans}='N'$  の場合) または  $\text{diag}(r)$  ( $\text{trans}='T'$  または  $'C'$  の場合) によって、行列  $X$  を事前に乗算する。

## 入力パラメータ

**fact** CHARACTER\*1. 'F'、'N'、または 'E' でなければならない。  
ルーチンの開始時に行列  $A$  の因子分解された形式が与えられるかどうか、与えられない場合は、行列  $A$  を因子分解する前に  $A$  を平衡化するかどうかを指定する。  
 $\text{fact}='F'$  の場合、開始時に、 $af$  と  $ipiv$  に  $A$  の因子分解された形式が格納される。 $\text{equed}$  が 'N' でない場合は、行列  $A$  は、 $r$  と  $c$  で指定されたスケール係数によって平衡化されている。  
 $a$ 、 $af$ 、 $ipiv$  は変更されない。  
 $\text{fact}='N'$  の場合、行列  $A$  を  $af$  にコピーし、因子分解を実行する。  
 $\text{fact}='E'$  の場合、必要に応じて行列  $A$  を平衡化してから、 $af$  にコピーし、因子分解を実行する。

**trans** CHARACTER\*1. 'N'、'T'、または 'C' でなければならない。  
連立方程式の形式を指定する。

$trans = 'N'$  の場合、連立方程式の形式は  $AX = B$  (転置なし) である。  
 $trans = 'T'$  の場合、連立方程式の形式は  $A^T X = B$  (転置) である。  
 $trans = 'C'$  の場合、連立方程式の形式は  $A^H X = B$  (共役転置) である。

$n$  INTEGER。1 次方程式の数。行列  $A$  の次数 ( $n \geq 0$ )。

$nrhs$  INTEGER。右辺の数。行列  $B$  と  $X$  の列数 ( $nrhs \geq 0$ )。

$a, af, b, work$  REAL (sgesvx の場合)  
DOUBLE PRECISION (dgesvx の場合)  
COMPLEX (cgesvx の場合)  
DOUBLE COMPLEX (zgesvx の場合)。  
配列:  $a(lda, *)$ ,  $af(ldaf, *)$ ,  $b(l db, *)$ ,  $work(*)$ 。

配列  $a$  には、行列  $A$  が格納される。 $fact = 'F'$  で、 $equed$  が 'N' でない場合は、 $A$  は、 $r$  または  $c$ 、あるいはその両方のスケール係数によって平衡化されている。 $a$  の第 2 次元は  $\max(1, n)$  以上でなければならない。

配列  $af$  は、 $fact = 'F'$  の場合は入力引数になる。  
この配列には、行列  $A$  の因子分解された形式、すなわち、[?getrf](#) による因子分解  $A = PLU$  で得られた係数  $L$  と  $U$  が格納される。 $equed$  が 'N' でない場合は、 $af$  は、平衡化された行列  $A$  の因子分解された形式になる。 $af$  の第 2 次元は  $\max(1, n)$  以上でなければならない。

配列  $b$  には、行列  $B$  が格納される。この行列の列は、連立方程式の右辺である。

$b$  の第 2 次元は  $\max(1, nrhs)$  以上でなければならない。

$work(*)$  は、ワークスペース配列。

$work$  の次元は  $\max(1, 4*n)$  (実数型の場合) または  $\max(1, 2*n)$  (複素数型の場合) 以上でなければならない。

$lda$  INTEGER。  $a$  の第 1 次元。  $lda \geq \max(1, n)$ 。

$ldaf$  INTEGER。  $af$  の第 1 次元。  $ldaf \geq \max(1, n)$ 。

$ldb$  INTEGER。  $b$  の第 1 次元。  $ldb \geq \max(1, n)$ 。

$ipiv$  INTEGER。

配列、次元は  $\max(1, n)$  以上。

配列  $ipiv$  は、 $fact = 'F'$  の場合は入力引数になる。

この配列には、[?getrf](#) による因子分解  $A = PLU$  で得られたピボットのインデックスが格納される。行列の行  $i$  は、行  $ipiv(i)$  と交換される。

<i>equed</i>	<p>CHARACTER*1。'N'、'R'、'C'、または 'B' でなければならない。  <i>equed</i> は、<i>fact</i> = 'F' の場合は入力引数になる。この引数は、実行された平衡化の形式を指定する。  <i>equed</i> = 'N' の場合、平衡化は行われていない (<i>fact</i> = 'N') の場合は常に真)。  <i>equed</i> = 'R' の場合、行の平衡化が行われ、<i>A</i> は <i>diag</i>(<i>r</i>) によって事前に乗算されている。  <i>equed</i> = 'C' の場合、列の平衡化が行われ、<i>A</i> は <i>diag</i>(<i>c</i>) によって事後に乗算されている。  <i>equed</i> = 'B' の場合、行と列の平衡化が行われ、<i>A</i> は <i>diag</i>(<i>r</i>)*<i>A</i>*<i>diag</i>(<i>c</i>) で置き換えられている。</p>
<i>r</i> , <i>c</i>	<p>REAL ( 単精度の場合 )  DOUBLE PRECISION ( 倍精度の場合 )。  配列 : <i>r</i>(<i>n</i>), <i>c</i>(<i>n</i>)。  配列 <i>r</i> には <i>A</i> の行のスケール係数が格納され、配列 <i>c</i> には <i>A</i> の列のスケール係数が格納される。これらの配列は、<i>fact</i> = 'F' の場合にのみ入力引数になり、それ以外の場合は出力引数になる。  <i>equed</i> = 'R' または 'B' の場合、<i>A</i> は <i>diag</i>(<i>r</i>) によって左辺で乗算される。<i>equed</i> = 'N' または 'C' の場合、<i>r</i> は使用されない。  <i>fact</i> = 'F' で、<i>equed</i> = 'R' または 'B' の場合、<i>r</i> の各成分は正の値でなければならない。    <i>equed</i> = 'C' または 'B' の場合、<i>A</i> は <i>diag</i>(<i>c</i>) によって右辺で乗算される。<i>equed</i> = 'N' または 'R' の場合、<i>c</i> は使用されない。  <i>fact</i> = 'F' で、<i>equed</i> = 'C' または 'B' の場合、<i>c</i> の各成分は正の値でなければならない。</p>
<i>ldx</i>	<p>INTEGER。出力配列 <i>x</i> の第 1 次元。 <math>ldx \geq \max(1, n)</math>。</p>
<i>iwork</i>	<p>INTEGER。  ワークスペース配列、次元は <math>\max(1, n)</math> 以上。実数型でのみ使用される。</p>
<i>rwork</i>	<p>REAL ( 単精度の場合 )  DOUBLE PRECISION ( 倍精度の場合 )。  ワークスペース配列、次元は <math>\max(1, 2*n)</math> 以上。複素数型でのみ使用される。</p>

## 出力パラメータ

<i>x</i>	<p>REAL (sgesvx の場合 )  DOUBLE PRECISION (dgesvx の場合 )  COMPLEX (cgesvx の場合 )  DOUBLE COMPLEX (zgesvx の場合 )。  配列、次元は (<i>ldx</i>, *)。</p> <p><i>info</i> = 0 または <i>info</i> = <i>n</i>+1 の場合、配列 <i>x</i> には、元の連立方程式の解の行列 <i>X</i> が格納される。  ただし、<i>equed</i> ≠ 'N' の場合、<i>A</i> と <i>B</i> は終了時に修正され、平衡化された連立方程式の解は次のようになる。  <math>\text{diag}(c)^{-1} * X</math> (<i>trans</i> = 'N' で、<i>equed</i> = 'C' または 'B' の場合 )  <math>\text{diag}(r)^{-1} * X</math> (<i>trans</i> = 'T' または 'C' で、<i>equed</i> = 'R' または 'B' の場合 )。  <i>x</i> の第 2 次元は max(1, <i>nrhs</i>) 以上でなければならない。</p>
<i>a</i>	<p>配列 <i>a</i> は、<i>fact</i> = 'F' または 'N' の場合、または <i>fact</i> = 'E' で <i>equed</i> = 'N' の場合、終了時に変更されない。  <i>equed</i> ≠ 'N' の場合、<i>A</i> は終了時に次のようにスケーリングされる。  <i>equed</i> = 'R': <math>A = \text{diag}(r) * A</math>  <i>equed</i> = 'C': <math>A = A * \text{diag}(c)</math>  <i>equed</i> = 'B': <math>A = \text{diag}(r) * A * \text{diag}(c)</math></p>
<i>af</i>	<p><i>fact</i> = 'N' または 'E' の場合、<i>af</i> は出力引数になる。この引数は、終了時に、元の行列 <i>A</i> (<i>fact</i> = 'N' の場合 ) または平衡化された行列 <i>A</i> (<i>fact</i> = 'E' の場合 ) の因子分解 <math>A = PLU</math> で得られた係数 <i>L</i> と <i>U</i> を返す。平衡化された行列の形式は、<i>a</i> の説明を参照のこと。</p>
<i>b</i>	<p><i>trans</i> = 'N' で、<i>equed</i> = 'R' または 'B' の場合、<math>\text{diag}(r) * B</math> によって上書きされる。  <i>trans</i> = 'T' で、<i>equed</i> = 'C' または 'B' の場合、<math>\text{diag}(c) * B</math> によって上書きされる。  <i>equed</i> = 'N' の場合は変更されない。</p>
<i>r</i> , <i>c</i>	<p>これらの配列は、<i>fact</i> ≠ 'F' の場合は出力引数になる。  「入力引数」セクションの <i>r</i>, <i>c</i> の説明を参照のこと。</p>
<i>rcond</i>	<p>REAL ( 単精度の場合 )  DOUBLE PRECISION ( 倍精度の場合 )。  平衡化後の行列 <i>A</i> の条件数の逆数の推定値 ( 平衡化が行われる場合 )。  推定値のアンダーフローが発生した場合は、<i>rcond</i> = 0 に設定される。</p>

	<p>この場合、行列は有効な精度で特異になる。  ただし、<i>rcond</i> が (有効な精度で) 1.0 より小さい場合は、条件の良くない行列または特異な行列である。</p>
<i>ferr, berr</i>	<p>REAL (単精度の場合)  DOUBLE PRECISION (倍精度の場合)。  配列、次元は <math>\max(1, nrhs)</math> 以上。各解ベクトルの成分ごとの前進誤差と相対後退誤差を格納する。</p>
<i>ipiv</i>	<p><i>fact</i> = 'N' または 'E' の場合、<i>ipiv</i> は出力引数になる。この引数には、終了時に、元の行列 <i>A</i> (<i>fact</i> = 'N' の場合) または平衡化された行列 <i>A</i> (<i>fact</i> = 'E' の場合) の因子分解 <math>A = PLU</math> で得られたピボットのインデックスが格納される。</p>
<i>equed</i>	<p><i>fact</i> ≠ 'F' の場合、<i>equed</i> は出力引数になる。この引数は、実行された平衡化の形式を指定する (「入力/数」セクションの <i>equed</i> の説明を参照)。</p>
<i>work, rwork</i>	<p>終了時に、<i>work</i>(1) (実数型の場合) または <i>rwork</i>(1) (複素数型の場合) には、ピボット成長係数の逆数 <math>\text{norm}(A)/\text{norm}(U)</math> が格納される。「最大絶対成分」ノルムが使用される。<i>work</i>(1) (実数型の場合) または <i>rwork</i>(1) (複素数型の場合) が 1 よりはるかに小さい場合は、(平衡化された) 行列 <i>A</i> の LU 因子分解の安定性は低くなる。このため、解 <i>x</i>、条件推定子 <i>rcond</i>、前進誤差範囲 <i>ferr</i> の信頼性も低くなる。  <math>0 &lt; info \leq n</math> で因子分解に失敗した場合は、<i>work</i>(1) (実数型の場合) または <i>rwork</i>(1) (複素数型の場合) には、<i>A</i> の先頭の <i>info</i> 列のピボット演算成長係数の逆数が格納される。</p>
<i>info</i>	<p>INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了した。  <i>info</i> = -<i>i</i> の場合、<i>i</i> 番目のパラメータの値が不正である。  <i>info</i> = <i>i</i> で、<math>i \leq n</math> の場合、<i>U</i>(<i>i</i>, <i>i</i>) が完全に 0 である。因子分解は完了したが、係数 <i>U</i> が完全に特異であるため、解と誤差範囲は計算できなかった。<i>rcond</i> = 0 が返される。  <i>info</i> = <i>i</i> で、<math>i = n + 1</math> の場合、<i>U</i> は特異でないが、<i>rcond</i> がマシンの精度より小さいため、行列は有効な精度で特異になる。しかし、この場合は、解と誤差範囲が計算される。これは、計算された解の精度が、<i>rcond</i> の値から想定される精度より高くなる場合があるためである。</p>

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

gesvx ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	サイズ $(n,n)$ の行列 <i>A</i> を格納する。
<i>b</i>	サイズ $(n,nrhs)$ の行列 <i>B</i> を格納する。
<i>x</i>	サイズ $(n,nrhs)$ の行列 <i>X</i> を格納する。
<i>af</i>	サイズ $(n,n)$ の行列 <i>AF</i> を格納する。
<i>ipiv</i>	長さ $(n)$ のベクトルを格納する。
<i>r</i>	長さ $(n)$ のベクトルを格納する。各成分のデフォルト値は、 $r(i) = 1.0\_WP$ である。
<i>c</i>	長さ $(n)$ のベクトルを格納する。各成分のデフォルト値は、 $c(i) = 1.0\_WP$ である。
<i>ferr</i>	長さ $(nrhs)$ のベクトルを格納する。
<i>berr</i>	長さ $(nrhs)$ のベクトルを格納する。
<i>fact</i>	'N'、'E'、または 'F' でなければならない。デフォルト値は 'N' である。 <i>fact</i> = 'F' の場合、引数 <i>af</i> および <i>ipiv</i> の両方を指定しなければならない。指定しない場合は、エラーが返される。
<i>trans</i>	'N'、'C'、または 'T' でなければならない。デフォルト値は 'N' である。
<i>equed</i>	'N'、'B'、'C'、または 'R' でなければならない。デフォルト値は 'N' である。
<i>rpvgrw</i>	実数値。ピボット成長係数の逆数 $\text{norm}(A)/\text{norm}(U)$ が格納される。

---

## ?gbsv

帯行列 *A* を係数行列とする、複数の右辺を持つ  
連立 1 次方程式の解を計算する。

---

### 構文

#### Fortran 77:

```
call sgbsv(n, kl, ku, nrhs, ab, ldab, ipiv, b, ldb, info)  
call dgbsv(n, kl, ku, nrhs, ab, ldab, ipiv, b, ldb, info)
```



```
call cgbstv(n, kl, ku, nrhs, ab, ldab, ipiv, b, ldb, info)
call zgbstv(n, kl, ku, nrhs, ab, ldab, ipiv, b, ldb, info)
```

#### Fortran 95:

```
call gbsv(a, b [,kl] [,ipiv] [,info])
```

#### 説明

このルーチンは、実数または複素連立 1 次方程式  $AX=B$  を、 $X$  について解く。 $A$  は  $kl$  個の劣対角成分と  $ku$  個の優対角成分を持つ  $n \times n$  の帯行列、行列  $B$  の列は個々の右辺、 $X$  の列はそれに対応する解である。

部分的なピボット演算と行の交換による  $LU$  分解を使用して、 $A$  を  $A=LU$  として因子分解する。 $L$  は置換行列と  $kl$  個の劣対角成分を持つ単位下三角行列の積、 $U$  は  $kl+ku$  個の優対角成分を持つ上三角行列である。次に、 $A$  の因子分解された形式を使用して、連立方程式  $AX=B$  を解く。

#### 入力パラメータ

<i>n</i>	INTEGER。 $A$ の次数。 $B$ の行数 ( $n \geq 0$ )。
<i>kl</i>	INTEGER。 $A$ の帯内の劣対角成分の数 ( $kl \geq 0$ )。
<i>ku</i>	INTEGER。 $A$ の帯内の優対角成分の数 ( $ku \geq 0$ )。
<i>nrhs</i>	INTEGER。 右辺の数。 $B$ の列数 ( $nrhs \geq 0$ )。
<i>ab, b</i>	REAL (sgbsv の場合) DOUBLE PRECISION (dgbstv の場合) COMPLEX (cgbsv の場合) double complex (zgbstv の場合)。 配列: <i>ab</i> ( <i>ldab</i> , *), <i>b</i> ( <i>ldb</i> , *)。 配列 <i>ab</i> には、行列 $A$ が帯形式で格納される (「 <a href="#">行列の格納形式</a> 」を参照)。 <i>ab</i> の第 2 次元は $\max(1, n)$ 以上でなければならない。 配列 <i>b</i> には、行列 $B$ が格納される。この行列の列は、連立方程式の右辺である。 <i>b</i> の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。
<i>ldab</i>	INTEGER。 配列 <i>ab</i> の第 1 次元 ( $ldab \geq 2kl + ku + 1$ )。
<i>ldb</i>	INTEGER。 <i>b</i> の第 1 次元。 $ldb \geq \max(1, n)$ 。

### 出力パラメータ

<i>ab</i>	<i>L</i> と <i>U</i> によって上書きされる。 <i>U</i> の対角成分と $k_l + k_u$ 個の優対角成分は、 <i>ab</i> の最初の $1 + k_l + k_u$ 行に格納される。 <i>L</i> の計算に使用される乗数は、次の $k_l$ 行に格納される。
<i>b</i>	解の行列 <i>X</i> によって上書きされる。
<i>ipiv</i>	INTEGER。 配列、次元は $\max(1, n)$ 以上。 ピボットのインデックス: 行 <i>i</i> は行 <i>ipiv</i> ( <i>i</i> ) と交換される。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了した。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正である。 <i>info</i> = <i>i</i> の場合、 <i>U</i> ( <i>i</i> , <i>i</i> ) が完全に 0 である。因子分解は完了したが、係数 <i>U</i> が完全に特異であるため、解を計算できなかった。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

gbsv ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ab</i> を意味する。 サイズ $(2 * k_l + k_u + 1, n)$ の配列 <i>A</i> を格納する。
<i>b</i>	サイズ $(n, nrhs)$ の行列 <i>B</i> を格納する。
<i>ipiv</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>kl</i>	省略した場合、 $k_l = k_u$ とみなす。
<i>ku</i>	$k_u = lda - 2 * k_l - 1$ として復元する。

## ?gbsvx

帯行列  $A$  を係数行列とする、複数の右辺を持つ  
実数または複素連立 1 次方程式の解を計算し、  
解の誤差範囲を示す。

### 構文

#### Fortran 77:

```
call sgbsvx(FACT, TRANS, N, kl, ku, NRHS, Ab, LDAb, AFb, LDAFb, IPIV,
            EQUED, R, C, B, LDB, X, LDX, RCOND, FERR, BERR, WORK, IWORK, INFO)
call dgbsvx(FACT, TRANS, N, kl, ku, NRHS, Ab, LDAb, AFb, LDAFb, IPIV,
            EQUED, R, C, B, LDB, X, LDX, RCOND, FERR, BERR, WORK, IWORK, INFO)
call cgbsvx(FACT, TRANS, N, kl, ku, NRHS, Ab, LDAb, AFb, LDAFb, IPIV,
            EQUED, R, C, B, LDB, X, LDX, RCOND, FERR, BERR, WORK, rWORK, INFO)
call zgbsvx(FACT, TRANS, N, kl, ku, NRHS, Ab, LDAb, AFb, LDAFb, IPIV,
            EQUED, R, C, B, LDB, X, LDX, RCOND, FERR, BERR, WORK, rWORK, INFO)
```

#### Fortran 95:

```
call gbsvx (a, b, x [,kl] [,af] [,ipiv] [,fact] [,trans] [,equed] [,r]
            [,c] [,ferr] [,berr] [,rcond] [,rpvgrw] [,info])
```

### 説明

このルーチンは、 $LU$  因子分解を使用して、実数または複素連立 1 次方程式  $AX=B$ 、 $A^T X=B$ 、または  $A^H X=B$  の解を計算する。 $A$  は  $kl$  個の劣対角成分と  $ku$  個の優対角成分を持つ次数  $n$  の帯行列、行列  $B$  の列は個々の右辺、 $X$  の列はそれに対応する解である。

このルーチンは、解の誤差範囲と条件数の推定値も示す。

ルーチン ?gbsvx は、以下の手順を実行する。

1.  $fact = 'E'$  の場合、連立方程式を平衡化するための実スケール係数  $r$  と  $c$  を計算する。

$trans = 'N'$ :  $diag(r) * A * diag(c) * diag(c)^{-1} * X = diag(r) * B$

$trans = 'T'$ :  $(diag(r) * A * diag(c))^T * diag(r)^{-1} * X = diag(c) * B$

$trans = 'C'$ :  $(diag(r) * A * diag(c))^H * diag(r)^{-1} * X = diag(c) * B$

連立方程式が平衡化されるかどうかは、行列  $A$  のスケーリングによって決まる。平衡化が行われる場合は、 $A$  は  $\text{diag}(r)*A*\text{diag}(c)$  によって上書きされ、 $B$  は  $\text{diag}(r)*B$  ( $\text{trans}='N'$  の場合) または  $\text{diag}(c)*B$  ( $\text{trans}='T'$  または  $'C'$  の場合) によって上書きされる。

2.  $\text{fact}='N'$  または  $'E'$  の場合、 $LU$  分解を使用して、( $\text{fact}='E'$  の場合は平衡化の後に) 行列  $A$  を  $A=LU$  として因子分解する。 $L$  は置換行列と  $k_l$  個の劣対角成分を持つ単位下三角行列の積、 $U$  は  $k_l+k_u$  個の優対角成分を持つ上三角行列である。

3.  $U_{i,i}=0$  の場合、つまり  $U$  が完全に特異である場合は、ルーチンは  $\text{info}=i$  を返す。それ以外の場合、 $A$  の因子分解された形式を使用して、行列  $A$  の条件数を推定する。条件数の逆数がマシンの精度より小さい場合は、警告として  $\text{info}=n+1$  を返すが、後で説明するようにルーチンは続行され、 $X$  について解を算出し、誤差範囲を計算する。

4.  $A$  の因子分解された形式を使用して、連立方程式を  $X$  について解く。

5. 計算された解の行列の精度を繰り返し改善し、解の誤差範囲と後退誤差の推定値を計算する。

6. 平衡化が行われている場合は、平衡化が行われる前の元の連立方程式を解くために、 $\text{diag}(c)$  ( $\text{trans}='N'$  の場合) または  $\text{diag}(r)$  ( $\text{trans}='T'$  または  $'C'$  の場合) によって、行列  $X$  を事前に乗算する。

### 入力パラメータ

**fact** CHARACTER\*1。'F'、'N'、または'E' でなければならない。  
ルーチンの開始時に行列  $A$  の因子分解された形式が与えられるかどうか、与えられない場合は、行列  $A$  を因子分解する前に  $A$  を平衡化するかどうかを指定する。  
 $\text{fact}='F'$  の場合、開始時に、 $\text{afb}$  と  $\text{ipiv}$  に  $A$  の因子分解された形式が格納される。 $\text{equed}$  が 'N' でない場合は、行列  $A$  は、 $r$  と  $c$  で指定されたスケール係数によって平衡化されている。  
 $\text{ab}$ 、 $\text{afb}$ 、 $\text{ipiv}$  は変更されない。  
 $\text{fact}='N'$  の場合、行列  $A$  を  $\text{afb}$  にコピーし、因子分解を実行する。  
 $\text{fact}='E'$  の場合、必要に応じて行列  $A$  を平衡化してから、 $\text{afb}$  にコピーし、因子分解を実行する。

**trans** CHARACTER\*1。'N'、'T'、または'C' でなければならない。  
連立方程式の形式を指定する。

$trans = 'N'$  の場合、連立方程式の形式は  $AX = B$  (転置なし) である。  
 $trans = 'T'$  の場合、連立方程式の形式は  $A^T X = B$  (転置) である。  
 $trans = 'C'$  の場合、連立方程式の形式は  $A^H X = B$  (共役転置) である。

$n$  INTEGER。1 次方程式の数。行列  $A$  の次数 ( $n \geq 0$ )。

$k1$  INTEGER。  $A$  の帯内の劣対角成分の数 ( $k1 \geq 0$ )。

$ku$  INTEGER。  $A$  の帯内の優対角成分の数 ( $ku \geq 0$ )。

$nrhs$  INTEGER。 右辺の数。行列  $B$  と  $X$  の列数 ( $nrhs \geq 0$ )。

$ab, afb, b, work$

REAL (sgesvx の場合 )  
 DOUBLE PRECISION (dgesvx の場合 )  
 COMPLEX (cgsvx の場合 )  
 DOUBLE COMPLEX (zgesvx の場合 )。  
 配列 :  $a(lda, *)$ ,  $af(ldaf, *)$ ,  $b ldb, *)$ ,  $work(*)$ 。  
 配列  $ab$  には、行列  $A$  が帯形式で格納される (「[行列の格納形式](#)」を参照)。  
 $ab$  の第 2 次元は  $\max(1, n)$  以上でなければならない。  
 $fact = 'F'$  で、 $equed$  が 'N' でない場合は、 $A$  は、 $r$  または  $c$ 、あるいはその両方のスケール係数によって平衡化されている。  
 配列  $afb$  は、 $fact = 'F'$  の場合は入力引数になる。  
 $afb$  の第 2 次元は  $\max(1, n)$  以上でなければならない。  
 この配列には、行列  $A$  の因子分解された形式、すなわち、[zgbtrf](#) による因子分解  $A = LU$  で得られた係数  $L$  と  $U$  が格納される。  
 $U$  は、 $k1 + ku$  個の優対角成分を持つ上三角帯行列として、 $afb$  の最初の  $1 + k1 + ku$  行に格納される。因子分解に使用した乗数は、次の  $k1$  行に格納される。  
 $equed$  が 'N' でない場合は、 $afb$  は、平衡化された行列  $A$  の因子分解された形式になる。

配列  $b$  には、行列  $B$  が格納される。この行列の列は、連立方程式の右辺である。  
 $b$  の第 2 次元は  $\max(1, nrhs)$  以上でなければならない。

$work(*)$  は、ワークスペース配列。  
 $work$  の次元は  $\max(1, 3*n)$  (実数型の場合) または  $\max(1, 2*n)$  (複素数型の場合) 以上でなければならない。

$ldab$  INTEGER。  $ab$  の第 1 次元。  $ldab \geq k1 + ku + 1$ 。

$ldafb$  INTEGER。  $afb$  の第 1 次元。  
 $ldafb \geq 2*k1 + ku + 1$ 。

<i>ldb</i>	INTEGER。 <i>b</i> の第 1 次元。 $ldb \geq \max(1, n)$ 。
<i>ipiv</i>	INTEGER。 配列、次元は $\max(1, n)$ 以上。 配列 <i>ipiv</i> は、 <i>fact</i> = 'F' の場合は入力引数になる。 この配列には、 <a href="#">?gbtrf</a> による因子分解 $A = LU$ で得られたピボットのインデックスが格納される。行列の行 <i>i</i> は、行 <i>ipiv</i> ( <i>i</i> ) と交換される。
<i>equed</i>	CHARACTER*1。 'N'、 'R'、 'C'、 または 'B' でなければならない。 <i>equed</i> は、 <i>fact</i> = 'F' の場合は入力引数になる。この引数は、実行された平衡化の形式を指定する。 <i>equed</i> = 'N' の場合、平衡化は行われていない ( <i>fact</i> = 'N') の場合は常に真)。 <i>equed</i> = 'R' の場合、行の平衡化が行われ、 <i>A</i> は <i>diag</i> ( <i>r</i> ) によって事前に乗算されている。 <i>equed</i> = 'C' の場合、列の平衡化が行われ、 <i>A</i> は <i>diag</i> ( <i>c</i> ) によって事後に乗算されている。 <i>equed</i> = 'B' の場合、行と列の平衡化が行われ、 <i>A</i> は <i>diag</i> ( <i>r</i> )* <i>A</i> * <i>diag</i> ( <i>c</i> ) で置き換えられている。
<i>r</i> , <i>c</i>	REAL ( 単精度の場合 ) DOUBLE PRECISION ( 倍精度の場合 )。 配列 : <i>r</i> ( <i>n</i> ), <i>c</i> ( <i>n</i> )。 配列 <i>r</i> には <i>A</i> の行のスケール係数が格納され、配列 <i>c</i> には <i>A</i> の列のスケール係数が格納される。これらの配列は、 <i>fact</i> = 'F' の場合にのみ入力引数になり、それ以外の場合は出力引数になる。 <i>equed</i> = 'R' または 'B' の場合、 <i>A</i> は <i>diag</i> ( <i>r</i> ) によって左辺で乗算される。 <i>equed</i> = 'N' または 'C' の場合、 <i>r</i> は使用されない。 <i>fact</i> = 'F' で、 <i>equed</i> = 'R' または 'B' の場合、 <i>r</i> の各成分は正の値でなければならない。 <i>equed</i> = 'C' または 'B' の場合、 <i>A</i> は <i>diag</i> ( <i>c</i> ) によって右辺で乗算される。 <i>equed</i> = 'N' または 'R' の場合、 <i>c</i> は使用されない。 <i>fact</i> = 'F' で、 <i>equed</i> = 'C' または 'B' の場合、 <i>c</i> の各成分は正の値でなければならない。
<i>ldx</i>	INTEGER。 出力配列 <i>x</i> の第 1 次元。 $ldx \geq \max(1, n)$ 。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, n)$ 以上。実数型でのみ使用される。

*rwork* REAL ( 単精度の場合 )  
 DOUBLE PRECISION ( 倍精度の場合 )。  
 ワークスペース配列、次元は  $\max(1, n)$  以上。複素数型でのみ使用される。

## 出力パラメータ

*x* REAL (sgbsvx の場合 )  
 DOUBLE PRECISION (dgbsvx の場合 )  
 COMPLEX (cgbsvx の場合 )  
 DOUBLE COMPLEX (zgbsvx の場合 )。  
 配列、次元は (*ldx*, \*)。  
  
*info* = 0 または *info* = *n*+1 の場合、配列 *x* には、元の連立方程式の解の行列 *X* が格納される。  
 ただし、*equed* ≠ 'N' の場合、*A* と *B* は終了時に修正され、平衡化された連立方程式の解は次のようになる。  
 $\text{diag}(c)^{-1} * X$  (*trans* = 'N' で、*equed* = 'C' または 'B' の場合 )  
 $\text{diag}(r)^{-1} * X$  (*trans* = 'T' または 'C' で、*equed* = 'R' または 'B' の場合 )。  
*x* の第 2 次元は  $\max(1, \text{nrhs})$  以上でなければならない。

*ab* 配列 *ab* は、*fact* = 'F' または 'N' の場合、あるいは *fact* = 'E' で *equed* = 'N' の場合、終了時に変更されない。  
*equed* ≠ 'N' の場合、*A* は終了時に次のようにスケーリングされる。  
*equed* = 'R':  $A = \text{diag}(r) * A$   
*equed* = 'C':  $A = A * \text{diag}(c)$   
*equed* = 'B':  $A = \text{diag}(r) * A * \text{diag}(c)$

*afb* *fact* = 'N' または 'E' の場合、*afb* は出力引数になる。この引数は、終了時に、元の行列 *A* (*fact* = 'N' の場合 ) または平衡化された行列 *A* (*fact* = 'E' の場合 ) の LU 因子分解の詳細を返す。  
 平衡化された行列の形式については、*ab* の説明を参照のこと。

*b* *trans* = 'N' で、*equed* = 'R' または 'B' の場合、 $\text{diag}(r) * b$  によって書きされる。  
*trans* = 'T' で、*equed* = 'C' または 'B' の場合、 $\text{diag}(c) * b$  によって書きされる。  
*equed* = 'N' の場合は変更されない。

*r, c* これらの配列は、*fact* ≠ 'F' の場合は出力引数になる。  
 「入力引数」セクションの *r, c* の説明を参照のこと。

<i>rcond</i>	REAL ( 単精度の場合 ) DOUBLE PRECISION ( 倍精度の場合 )。 平衡化後の行列 <i>A</i> の条件数の逆数の推定値 ( 平衡化が行われる場合 )。 <i>rcond</i> がマシンの精度より小さい場合 ( 特に、 <i>rcond</i> = 0 の場合 ) は、行列は有効な精度で特異になる。この条件は、 <i>info</i> > 0 のリターンコードで示される。
<i>ferr, berr</i>	REAL ( 単精度の場合 ) DOUBLE PRECISION ( 倍精度の場合 )。 配列、次元は $\max(1, nrhs)$ 以上。各解ベクトルの成分ごとの前進誤差と相対後退誤差を格納する。
<i>ipiv</i>	<i>fact</i> = 'N' または 'E' の場合、 <i>ipiv</i> は出力引数になる。この引数には、終了時に、元の行列 <i>A</i> ( <i>fact</i> = 'N' の場合) または平衡化された行列 <i>A</i> ( <i>fact</i> = 'E' の場合) の因子分解 $A = LU$ で得られたピボットのインデックスが格納される。
<i>equed</i>	<i>fact</i> ≠ 'F' の場合、 <i>equed</i> は出力引数になる。この引数は、実行された平衡化の形式を指定する ( 「入力引数」 セクションの <i>equed</i> の説明を参照 )。
<i>work, rwork</i>	終了時に、 <i>work</i> (1) ( 実数型の場合 ) または <i>rwork</i> (1) ( 複素数型の場合 ) には、ピボット成長係数の逆数 $\text{norm}(A)/\text{norm}(U)$ が格納される。「最大絶対成分」 ノルムが使用される。 <i>work</i> (1) ( 実数型の場合 ) または <i>rwork</i> (1) ( 複素数型の場合 ) が 1 よりはるかに小さい場合は、( 平衡化された ) 行列 <i>A</i> の <i>LU</i> 因子分解の安定性は低くなる。このため、解 <i>x</i> 、条件推定子 <i>rcond</i> 、前進誤差範囲 <i>ferr</i> の信頼性も低くなる。0 < <i>info</i> ≤ <i>n</i> で因子分解に失敗した場合は、 <i>work</i> (1) ( 実数型の場合 ) または <i>rwork</i> (1) ( 複素数型の場合 ) には、 <i>A</i> の先頭の <i>info</i> 列のピボット演算成長係数の逆数が格納される。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了した。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正である。 <i>info</i> = <i>i</i> で、 <i>i</i> ≤ <i>n</i> の場合、 <i>U</i> ( <i>i</i> , <i>i</i> ) が完全に 0 である。因子分解は完了したが、係数 <i>U</i> が完全に特異であるため、解と誤差範囲は計算できなかった。 <i>rcond</i> = 0 が返される。 <i>info</i> = <i>i</i> で、 <i>i</i> = <i>n</i> + 1 の場合、 <i>U</i> は特異でないが、 <i>rcond</i> がマシンの精度より小さいため、行列は有効な精度で特異になる。しかし、この場合は、解と誤差範囲が計算される。これは、計算された解の精度が、 <i>rcond</i> の値から想定される精度より高くなる場合があるためである。



## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

gbsvx ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ab</i> を意味する。 サイズ $(kl+ku+1, n)$ の配列 <i>A</i> を格納する。
<i>b</i>	サイズ $(n, nrhs)$ の行列 <i>B</i> を格納する。
<i>x</i>	サイズ $(n, nrhs)$ の行列 <i>X</i> を格納する。
<i>af</i>	Fortran 77 インターフェイスでの引数 <i>ab</i> を意味する。 サイズ $(2*kl+ku+1, n)$ の配列 <i>AF</i> を格納する。
<i>ipiv</i>	長さ $(n)$ のベクトルを格納する。
<i>r</i>	長さ $(n)$ のベクトルを格納する。各成分のデフォルト値は、 $r(i) = 1.0\_WP$ である。
<i>c</i>	長さ $(n)$ のベクトルを格納する。各成分のデフォルト値は、 $c(i) = 1.0\_WP$ である。
<i>ferr</i>	長さ $(nrhs)$ のベクトルを格納する。
<i>berr</i>	長さ $(nrhs)$ のベクトルを格納する。
<i>trans</i>	'N'、'C'、または 'T' でなければならない。デフォルト値は 'N' である。
<i>equed</i>	'N'、'B'、'C'、または 'R' でなければならない。デフォルト値は 'N' である。
<i>fact</i>	'N'、'E'、または 'F' でなければならない。デフォルト値は 'N' である。 <i>fact</i> = 'F' の場合、引数 <i>af</i> および <i>ipiv</i> の両方を指定しなければならない。指定しない場合は、エラーが返される。
<i>rpvgrw</i>	実数値。ピボット成長係数の逆数 $\text{norm}(A)/\text{norm}(U)$ が格納される。
<i>kl</i>	省略した場合、 $kl = ku$ とみなす。
<i>ku</i>	$ku = lda - kl - 1$ として復元する。

## ?gtsv

三重対角行列  $A$  を係数行列とする、複数の右辺を持つ連立 1 次方程式の解を計算する。

### 構文

#### Fortran 77:

```
call sgtsv(n, nrhs, dl, d, du, b, ldb, info)
call dgtsv(n, nrhs, dl, d, du, b, ldb, info)
call cgtsv(n, nrhs, dl, d, du, b, ldb, info)
call zgtsv(n, nrhs, dl, d, du, b, ldb, info)
```

#### Fortran 95:

```
call gtsv(dl, d, du, b [,info])
```

### 説明

このルーチンは、連立 1 次方程式  $AX=B$  を  $X$  について解く。 $A$  は  $n \times n$  の三重対角行列、行列  $B$  の列は個々の右辺、 $X$  の列はそれに対応する解である。

このルーチンは、部分的なピボット演算によるガウス消去を使用する。

方程式  $A^T X = B$  の解は、引数  $du$  と  $dl$  の順序を入れ替えて求められる。

### 入力パラメータ

$n$	INTEGER。 $A$ の次数。 $B$ の行数 ( $n \geq 0$ )。
$nrhs$	INTEGER。 右辺の数。 $B$ の列数 ( $nrhs \geq 0$ )。
$dl, d, du, b$	REAL (sgtsv の場合 ) DOUBLE PRECISION (dgtsv の場合 ) COMPLEX (cgtsv の場合 ) DOUBLE COMPLEX (zgtsv の場合 )。 配列 : $dl(n-1), d(n), du(n-1), b(ldb, *)$ 。 配列 $dl$ には、 $A$ の $(n-1)$ 個の劣対角成分が格納される。 配列 $d$ には、 $A$ の対角成分が格納される。 配列 $du$ には、 $A$ の $(n-1)$ 個の優対角成分が格納される。 配列 $b$ には、行列 $B$ が格納される。この行列の列は、連立方程式の右辺である。 $b$ の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。

*ldb* INTEGER。 *b* の第 1 次元。  $ldb \geq \max(1, n)$ 。

### 出力パラメータ

*d1* A の LU 因子分解で得られた上三角行列 *U* の 2 番目の優対角成分の (n-2) 個の成分によって上書きされる。これらの成分は、*d1*(1), ..., *d1*(n-2) に格納される。

*d* *U* の *n* 個の対角成分によって上書きされる。

*du* *U* の最初の優対角成分の (n-1) 個の成分によって上書きされる。

*b* 解の行列 *X* によって上書きされる。

*info* INTEGER。 *info* = 0 の場合、実行は正常に終了した。  
*info* = -*i* の場合、*i* 番目のパラメータの値が不正である。  
*info* = *i* の場合、*U*(*i*, *i*) が完全に 0 であるため、解は計算されなかった。*i* = *n* でない限り、因子分解は完了していない。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

gtsv ルーチンのインターフェイス特有の詳細を以下に示す。

*d1* 長さ (n-1) のベクトルを格納する。

*d* 長さ (n) のベクトルを格納する。

*d1* 長さ (n-1) のベクトルを格納する。

*b* サイズ (n, nrhs) の行列 *B* を格納する。

## ?gtsvx

三重対角行列  $A$  を係数行列とする、複数の右辺を持つ実数または複素連立 1 次方程式の解を計算し、解の誤差範囲を示す。

### 構文

#### Fortran 77:

```
call sgtsvx(FACT, TRANS, N, NRHS, dl, d, du, dlf, df, duf, du2, IPIV, B,  
            LDB, X, LDX, RCOND, FERR, BERR, WORK, IWORK, INFO)  
call dgtsvx(FACT, TRANS, N, NRHS, dl, d, du, dlf, df, duf, du2, IPIV, B,  
            LDB, X, LDX, RCOND, FERR, BERR, WORK, IWORK, INFO)  
call cgtsvx(FACT, TRANS, N, NRHS, dl, d, du, dlf, df, duf, du2, IPIV, B,  
            LDB, X, LDX, RCOND, FERR, BERR, WORK, rWORK, INFO)  
call zgtsvx(FACT, TRANS, N, NRHS, dl, d, du, dlf, df, duf, du2, IPIV, B,  
            LDB, X, LDX, RCOND, FERR, BERR, WORK, rWORK, INFO)
```

#### Fortran 95:

```
call gtsvx (dl, d, du, b, x [,dlf] [,df] [,duf] [,du2] [,ipiv] [,fact]  
            [,trans] [,ferr] [,berr] [,rcond] [,info])
```

### 説明

このルーチンは、 $LU$  因子分解を使用して、実数または複素連立 1 次方程式  $AX=B$ 、 $A^T X=B$ 、または  $A^H X=B$  の解を計算する。 $A$  は次数  $n$  の三重対角行列、行列  $B$  の列は個々の右辺、 $X$  の列はそれに対応する解である。

このルーチンは、解の誤差範囲と条件数の推定値も示す。

ルーチン ?gtsvx は、以下の手順を実行する。

1.  $fact = 'N'$  の場合、 $LU$  分解を使用して、行列  $A$  を  $A=LU$  として因子分解する。 $L$  は置換行列と単位下二重対角行列の積で、 $U$  は主対角成分と最初の 2 つの優対角成分にのみ 0 でない値を持つ上三角行列である。

2.  $U_{i,i} = 0$  の場合、つまり  $U$  が完全に特異である場合は、ルーチンは  $info = i$  を返す。それ以外の場合、 $A$  の因子分解された形式を使用して、行列  $A$  の条件数を推定する。条件数の逆数がマシンの精度より小さい場合は、警告として  $info = n + 1$  を返すが、後で説明するようにルーチンは続行され、 $X$  について解を算出し、誤差範囲を計算する。

3.  $A$  の因子分解された形式を使用して、連立方程式を  $X$  について解く。
4. 計算された解の行列の精度を繰り返し改善し、解の誤差範囲と後退誤差の推定値を計算する。

## 入力パラメータ

*fact* CHARACTER\*1. 'F' または 'N' でなければならない。  
 ルーチンの開始時に行列  $A$  の因子分解された形式が与えられるかどうかを指定する。  
*fact* = 'F' の場合、開始時に、*d1f*、*df*、*duf*、*du2*、*ipiv* に、 $A$  の因子分解された形式が格納される。配列 *d1*、*d*、*du*、*d1f*、*df*、*duf*、*du2*、*ipiv* は変更されない。  
*fact* = 'N' の場合、行列  $A$  を *d1f*、*df*、*duf* にコピーし、因子分解を実行する。

*trans* CHARACTER\*1. 'N'、'T'、または 'C' でなければならない。  
 連立方程式の形式を指定する。  
*trans* = 'N' の場合、連立方程式の形式は  $AX=B$  (転置なし) である。  
*trans* = 'T' の場合、連立方程式の形式は  $A^T X=B$  (転置) である。  
*trans* = 'C' の場合、連立方程式の形式は  $A^H X=B$  (共役転置) である。

*n* INTEGER. 1 次方程式の数。行列  $A$  の次数 ( $n \geq 0$ )。

*nrhs* INTEGER. 右辺の数。行列  $B$  と  $X$  の列数 ( $nrhs \geq 0$ )。

*d1*, *d*, *du*, *d1f*, *df*,  
*duf*, *du2*, *b*, *x*, *work*  
 REAL (sgtsvx の場合)  
 DOUBLE PRECISION (dgtsvx の場合)  
 COMPLEX (cgtsvx の場合)  
 DOUBLE COMPLEX (zgtsvx の場合)。  
 配列:  
*d1*、次元は  $(n-1)$ 。 $A$  の劣対角成分を格納する。  
*d*、次元は  $(n)$ 。 $A$  の対角成分を格納する。  
*du*、次元は  $(n-1)$ 。 $A$  の優対角成分を格納する。  
*d1f*、次元は  $(n-1)$ 。*fact* = 'F' の場合、*d1f* は入力引数になる。この引数には、開始時に、[?gttrf](#) による  $A$  の  $LU$  因子分解で得られた行列  $L$  を定義する、 $(n-1)$  個の乗数が格納される。

*df*、次元は (*n*)。 *fact* = 'F' の場合、*df* は入力引数になる。この引数には、開始時に、*A* の *LU* 因子分解で得られた上三角行列 *U* の *n* 個の対角成分が格納される。

*duf*、次元は (*n* - 1)。 *fact* = 'F' の場合、*duf* は入力引数になる。この引数には、開始時に、*U* の最初の優対角成分の (*n* - 1) 個の成分が格納される。

*du2*、次元は (*n* - 2)。 *fact* = 'F' の場合、*du2* は入力引数になる。この引数には、開始時に、*U* の 2 番目の優対角成分の (*n* - 2) 個の成分が格納される。

*b*(*ldb*, \*) には、右辺の行列 *B* が格納される。*b* の第 2 次元は *max*(1, *nrhs*) 以上でなければならない。

*x*(*ldx*, \*) には、解の行列 *X* が格納される。*x* の第 2 次元は *max*(1, *nrhs*) 以上でなければならない。

*work* (\*) はワークスペース配列である。

*work* の次元は *max*(1, 3\**n*) (実数型の場合) または *max*(1, 2\**n*) (複素数型の場合) 以上でなければならない。

*ldb* INTEGER。 *b* の第 1 次元。 *ldb* ≥ *max*(1, *n*)。

*ldx* INTEGER。 *x* の第 1 次元。 *ldx* ≥ *max*(1, *n*)。

*ipiv* INTEGER。

配列、次元は *max*(1, *n*) 以上。 *fact* = 'F' の場合、*ipiv* は入力引数になる。この配列には、開始時に、[?gttrf](#) によって返されたピボットのインデックスが格納される。

*iwork* INTEGER。

ワークスペース配列、次元は (*n*)。実数型でのみ使用される。

*rwork* REAL (cgtsvx の場合)

DOUBLE PRECISION (zgtsvx の場合)。

ワークスペース配列、次元は (*n*)。複素数型でのみ使用される。

## 出力パラメータ

*x* REAL (sgtsvx の場合)

DOUBLE PRECISION (dgtsvx の場合)

COMPLEX (cgtsvx の場合)

DOUBLE COMPLEX (zgtsvx の場合)。

配列、次元は (*ldx*, \*)。

	$info = 0$ または $info = n+1$ の場合、配列 $x$ には、解の行列 $X$ が格納される。 $x$ の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。
<i>d1f</i>	$fact = 'N'$ の場合、 <i>d1f</i> は出力引数になる。この引数には、終了時に、 $A$ の $LU$ 因子分解で得られた行列 $L$ を定義する、 $(n-1)$ 個の乗数が格納される。
<i>df</i>	$fact = 'N'$ の場合、 <i>df</i> は出力引数になる。この引数には、終了時に、 $A$ の $LU$ 因子分解で得られた上三角行列 $U$ の $n$ 個の対角成分が格納される。
<i>duf</i>	$fact = 'N'$ の場合、 <i>duf</i> は出力引数になる。この引数には、終了時に、 $U$ の最初の優対角成分の $(n-1)$ 個の成分が格納される。
<i>du2</i>	$fact = 'N'$ の場合、 <i>du2</i> は出力引数になる。この引数には、終了時に、 $U$ の 2 番目の優対角成分の $(n-2)$ 個の成分が格納される。
<i>ipiv</i>	配列 <i>ipiv</i> は、 $fact = 'N'$ の場合は出力引数になる。この配列は、終了時に、因子分解 $A = LU$ で得られたピボットのインデックスが格納される。行列の行 $i$ は、行 $ipiv(i)$ と交換される。 $ipiv(i)$ の値は、常に $i$ または $i+1$ になる。 $ipiv(i)=i$ は、行の交換が不要であったことを示す。
<i>rcond</i>	REAL ( 単精度の場合 ) DOUBLE PRECISION ( 倍精度の場合 )。 行列 $A$ の条件数の逆数の推定値。 $rcond$ がマシンの精度より小さい場合 ( 特に、 $rcond = 0$ の場合 ) は、行列は有効な精度で特異になる。この条件は、 $info > 0$ のリターンコードで示される。
<i>ferr, berr</i>	REAL ( 単精度の場合 ) DOUBLE PRECISION ( 倍精度の場合 )。 配列、次元は $\max(1, nrhs)$ 以上。各解ベクトルの成分ごとの前進誤差と後退誤差を格納する。
<i>info</i>	INTEGER。 $info = 0$ の場合、実行は正常に終了した。 $info = -i$ の場合、 $i$ 番目のパラメータの値が不正である。 $info = i$ で、 $i \leq n$ の場合、 $U(i, i)$ が完全に 0 である。 $i = n$ でない限り、因子分解は完了していない。係数 $U$ が完全に特異であるため、解と誤差範囲は計算できなかった。 $rcond = 0$ が返される。 $info = i$ で、 $i = n+1$ の場合、 $U$ は特異でないが、 $rcond$ がマシンの精度より小さいため、行列は有効な精度で特異になる。しかし、この場合は、解と誤差範囲が計算される。これは、計算された解の精度が、 $rcond$ の値から想定される精度より高くなる場合があるためである。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

gtsvx ルーチンのインターフェイス特有の詳細を以下に示す。

<i>dl</i>	長さ ( <i>n-1</i> ) のベクトルを格納する。
<i>d</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>du</i>	長さ ( <i>n-1</i> ) のベクトルを格納する。
<i>b</i>	サイズ ( <i>n,nrhs</i> ) の行列 <i>B</i> を格納する。
<i>x</i>	サイズ ( <i>n,nrhs</i> ) の行列 <i>X</i> を格納する。
<i>dlf</i>	長さ ( <i>n-1</i> ) のベクトルを格納する。
<i>df</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>duf</i>	長さ ( <i>n-1</i> ) のベクトルを格納する。
<i>du2</i>	長さ ( <i>n-2</i> ) のベクトルを格納する。
<i>ipiv</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>ferr</i>	長さ ( <i>nrhs</i> ) のベクトルを格納する。
<i>berr</i>	長さ ( <i>nrhs</i> ) のベクトルを格納する。
<i>fact</i>	'N' または 'F' でなければならない。デフォルト値は 'N' である。 <i>fact</i> = 'F' の場合、引数 <i>dlf</i> 、 <i>df</i> 、 <i>duf</i> 、 <i>du2</i> 、および <i>ipiv</i> を指定しなければならない。指定しない場合は、エラーが返される。
<i>trans</i>	'N'、'C'、または 'T' でなければならない。デフォルト値は 'N' である。



## ?posv

対称/エルミート正定値行列  $A$  を係数行列とする、複数の右辺を持つ連立 1 次方程式を解く。

### 構文

#### Fortran 77:

```
call sposv(uplo, n, nrhs, a, lda, b, ldb, info)
call dposv(uplo, n, nrhs, a, lda, b, ldb, info)
call cposv(uplo, n, nrhs, a, lda, b, ldb, info)
call zposv(uplo, n, nrhs, a, lda, b, ldb, info)
```

#### Fortran 95:

```
call posv(a, b [,uplo] [,info])
```

### 説明

このルーチンは、実数または複素数連立 1 次方程式  $AX=B$  を、 $X$  について解く。 $A$  は  $n \times n$  の対称/エルミート正定値行列、行列  $B$  の列は個々の右辺、 $X$  の列はそれに対応する解である。

コレスキー因子分解を使用して、 $A$  を  $A=U^H U$  ( $uplo='U'$  の場合)

または  $A=LL^H$  ( $uplo='L'$  の場合) として因子分解する。 $U$  は上三角行列、 $L$  は下三角行列である。次に、 $A$  の因子分解された形式を使用して、連立方程式  $AX=B$  を解く。

### 入力パラメータ

<code>uplo</code>	CHARACTER*1。'U' または 'L' でなければならない。 $A$ の上三角部分と下三角部分のどちらを格納するか、また $A$ をどのように因子分解するかを指定する。 $uplo='U'$ の場合、配列 $a$ には行列 $A$ の上三角部分が格納され、 $A$ は $U^H U$ として因子分解される。 $uplo='L'$ の場合、配列 $a$ には行列 $A$ の下三角部分が格納され、 $A$ は $LL^H$ として因子分解される。
<code>n</code>	INTEGER。行列 $A$ の次数 ( $n \geq 0$ )。
<code>nrhs</code>	INTEGER。右辺の数。 $B$ の列数 ( $nrhs \geq 0$ )。

<i>a</i> , <i>b</i>	REAL (sposv の場合 ) DOUBLE PRECISION (dposv の場合 ) COMPLEX (cposv の場合 ) DOUBLE COMPLEX (zposv の場合 )。 配列 : <i>a</i> ( <i>lda</i> , *), <i>b</i> ( <i>ldb</i> , *)。 配列 <i>a</i> には、行列 <i>A</i> の上三角部分または下三角部分を格納する ( <i>uplo</i> を参照 )。 <i>a</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 配列 <i>b</i> には、行列 <i>B</i> が格納される。この行列の列は、連立方程式の右辺である。 <i>b</i> の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $lda \geq \max(1, n)$ 。
<i>ldb</i>	INTEGER。 <i>b</i> の第 1 次元。 $ldb \geq \max(1, n)$ 。

### 出力パラメータ

<i>a</i>	<i>info</i> = 0 の場合、 <i>uplo</i> の指定に従って、 <i>a</i> の上三角部分または下三角部分が、コレスキー係数 <i>U</i> あるいは <i>L</i> によって上書きされる。
<i>b</i>	解の行列 <i>X</i> によって上書きされる。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了した。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正である。 <i>info</i> = <i>i</i> の場合、次数 <i>i</i> の先頭の Minor 行列式 (したがって、行列 <i>A</i> そのもの) が正定値になっていないため、因子分解を実行できず、解は計算されなかった。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

posv ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	サイズ ( <i>n</i> , <i>n</i> ) の行列 <i>A</i> を格納する。
<i>b</i>	サイズ ( <i>n</i> , <i>nrhs</i> ) の行列 <i>B</i> を格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U' である。

## ?posvx

コレスキー因子分解を使用して、対称/エルミート正定値行列  $A$  を係数行列とする連立 1 次方程式の解を計算し、解の誤差範囲を示す。

### 構文

#### Fortran 77:

```
call sposvx(FACT, uplo, N, NRHS, A, LDA, AF, LDAF, EQUED, s, B, LDB, X,
            LDX, RCOND, FERR, BERR, WORK, IWORK, INFO)
call dposvx(FACT, uplo, N, NRHS, A, LDA, AF, LDAF, EQUED, s, B, LDB, X,
            LDX, RCOND, FERR, BERR, WORK, IWORK, INFO)
call cposvx(FACT, uplo, N, NRHS, A, LDA, AF, LDAF, EQUED, s, B, LDB, X,
            LDX, RCOND, FERR, BERR, WORK, rWORK, INFO)
call zposvx(FACT, uplo, N, NRHS, A, LDA, AF, LDAF, EQUED, s, B, LDB, X,
            LDX, RCOND, FERR, BERR, WORK, rWORK, INFO)
```

#### Fortran 95:

```
call posvx (a, b, x [,uplo] [,af] [,fact] [,equed] [,s] [,ferr] [,berr]
            [,rcond] [,info])
```

### 説明

このルーチンは、コレスキー因子分解  $A=U^H U$  または  $A=LL^H$  を使用して、実数または複素数連立 1 次方程式  $AX=B$  の解を計算する。 $A$  は  $n \times n$  の実対称/エルミート正定値行列、行列  $B$  の列は個々の右辺、 $X$  の列はそれに対応する解である。

このルーチンは、解の誤差範囲と条件数の推定値も示す。

ルーチン ?posvx は、以下の手順を実行する。

1.  $fact = 'E'$  の場合、連立方程式を平衡化するための実スケール係数  $s$  を計算する。

$$\text{diag}(s) * A * \text{diag}(s) * \text{diag}(s)^{-1} * X = \text{diag}(s) * B$$

連立方程式が平衡化されるかどうかは、行列  $A$  のスケーリングによって決まる。平衡化が行われる場合は、 $A$  は  $\text{diag}(s) * A * \text{diag}(s)$  によって上書きされ、 $B$  は  $\text{diag}(s) * B$  によって上書きされる。

2.  $fact = 'N'$  または  $'E'$  の場合、コレスキー分解を使用して、( $fact = 'E'$  の場合は平衡化の後に) 行列  $A$  を次のように因子分解する。

$A = U^H U$  ( $uplo = 'U'$  の場合)、または  
 $A = L L^H$  ( $uplo = 'L'$  の場合)。  
 $U$  は上三角行列で、 $L$  は下三角行列である。

3. 先頭の  $i \times i$  の主小行列式が正定値でない場合は、ルーチンは  $info = i$  を返す。それ以外の場合、 $A$  の因子分解された形式を使用して、行列  $A$  の条件数を推定する。条件数の逆数がマシンの精度より小さい場合は、警告として  $info = n + 1$  を返すが、後で説明するようにルーチンは続行され、 $X$  について解を算出し、誤差範囲を計算する。

4.  $A$  の因子分解された形式を使用して、連立方程式を  $X$  について解く。

5. 計算された解の行列の精度を繰り返し改善し、解の誤差範囲と後退誤差の推定値を計算する。

6. 平衡化が行われている場合は、平衡化が行われる前の元の連立方程式を解くために、 $diag(s)$  によって行列  $X$  を事前に乗算する。

## 入力パラメータ

$fact$	CHARACTER*1。'F'、'N'、または 'E' でなければならない。  ルーチンの開始時に行列 $A$ の因子分解された形式が与えられるかどうか、与えられない場合は、行列 $A$ を因子分解する前に $A$ を平衡化するかどうかを指定する。  $fact = 'F'$ の場合、開始時に、 $af$ に $A$ の因子分解された形式が格納される。 $eques = 'Y'$ の場合、行列 $A$ は、 $s$ で指定されたスケール係数によって平衡化されている。 $a$ と $af$ は変更されない。  $fact = 'N'$ の場合、行列 $A$ を $af$ にコピーし、因子分解を実行する。 $fact = 'E'$ の場合、必要に応じて行列 $A$ を平衡化してから、 $af$ にコピーし、因子分解を実行する。
$uplo$	CHARACTER*1。'U' または 'L' でなければならない。 $A$ の上三角部分と下三角部分のどちらを格納するか、また $A$ をどのように因子分解するかを指定する。 $uplo = 'U'$ の場合、配列 $a$ には行列 $A$ の上三角部分が格納され、 $A$ は $U^H U$ として因子分解される。 $uplo = 'L'$ の場合、配列 $a$ には行列 $A$ の下三角部分が格納され、 $A$ は $LL^H$ として因子分解される。
$n$	INTEGER。行列 $A$ の次数 ( $n \geq 0$ )。

*nrhs* INTEGER。右辺の数。 *B* の列数 ( $nrhs \geq 0$ )。

*a*, *af*, *b*, *work*

REAL (sposvx の場合 )  
 DOUBLE PRECISION (dposvx の場合 )  
 COMPLEX (cposvx の場合 )  
 DOUBLE COMPLEX (zposvx の場合 )。  
 配列 : *a*(*lda*, \*), *af*(*ldaf*, \*), *b*(*ldb*, \*), *work*(\*)。

配列 *a* には、*uplo* の指定に従って、行列 *A* が格納される。  
*fact* = 'F' で *equed* = 'Y' の場合、*A* は *s* のスケール係数によって平衡化され、*a* には平衡化された行列  $\text{diag}(s) * A * \text{diag}(s)$  が格納されている。*a* の第 2 次元は  $\max(1, n)$  以上でなければならない。

配列 *af* は、*fact* = 'F' の場合は入力引数になる。  
 この配列には、*A* のコレスキー因子分解で得られた三角係数 *U* または *L* が、*A* と同じ格納形式で格納される。*equed* が 'N' でない場合は、*af* は、平衡化された行列  $\text{diag}(s) * A * \text{diag}(s)$  の因子分解された形式になる。*af* の第 2 次元は  $\max(1, n)$  以上でなければならない。

配列 *b* には、行列 *B* が格納される。この行列の列は、連立方程式の右辺である。  
*b* の第 2 次元は  $\max(1, nrhs)$  以上でなければならない。

*work*(\*) は、ワークスペース配列。  
*work* の次元は  $\max(1, 3 * n)$  (実数型の場合) または  $\max(1, 2 * n)$  (複素数型の場合) 以上でなければならない。

*lda* INTEGER。 *a* の第 1 次元。  $lda \geq \max(1, n)$ 。

*ldaf* INTEGER。 *af* の第 1 次元。  $ldaf \geq \max(1, n)$ 。

*ldb* INTEGER。 *b* の第 1 次元。  $ldb \geq \max(1, n)$ 。

*equed* CHARACTER\*1。 'N' または 'Y' でなければならない。  
*equed* は、*fact* = 'F' の場合は入力引数になる。この引数は、実行された平衡化の形式を指定する。  
*equed* = 'N' の場合、平衡化は行われていない (*fact* = 'N' の場合は常に真)。  
*equed* = 'Y' の場合、平衡化が行われ、*A* は  $\text{diag}(s) * A * \text{diag}(s)$  で置き換えられている。

*s* REAL (単精度の場合 )  
 DOUBLE PRECISION (倍精度の場合 )。  
 配列、次元は (*n*)。  
 配列 *s* には、*A* のスケール係数が格納される。この配列は、*fact* = 'F'

の場合にのみ入力引数になり、それ以外の場合は出力引数になる。  
 $equed = 'N'$  の場合、 $s$  は使用されない。  
 $fact = 'F'$  で、 $equed = 'Y'$  の場合、 $s$  の各成分は正の値でなければならない。

$ldx$  INTEGER。出力配列  $x$  の第 1 次元。  $ldx \geq \max(1, n)$ 。  
 $iwork$  INTEGER。  
 ワークスペース配列、次元は  $\max(1, n)$  以上。実数型でのみ使用される。  
 $rwork$  REAL ( $cposvx$  の場合 )  
 DOUBLE PRECISION ( $zposvx$  の場合 )。  
 ワークスペース配列、次元は  $\max(1, n)$  以上。複素数型でのみ使用される。

## 出力パラメータ

$x$  REAL ( $sposvx$  の場合 )  
 DOUBLE PRECISION ( $dposvx$  の場合 )  
 COMPLEX ( $cposvx$  の場合 )  
 DOUBLE COMPLEX ( $zposvx$  の場合 )。  
 配列、次元は  $(ldx, *)$ 。  
 $info = 0$  または  $info = n+1$  の場合、配列  $x$  には、元の連立方程式の解の行列  $X$  が格納される。  
 ただし、 $equed = 'Y'$  の場合、 $A$  と  $B$  は終了時に修正され、平衡化された連立方程式の解は  $diag(s)^{-1} * X$  になる。  
 $x$  の第 2 次元は  $\max(1, nrhs)$  以上でなければならない。

$a$  配列  $a$  は、 $fact = 'F'$  または  $'N'$  の場合、または  
 $fact = 'E'$  で  $equed = 'N'$  の場合、終了時に変更されない。  
 $fact = 'E'$  で  $equed = 'Y'$  の場合、 $A$  は  $diag(s) * A * diag(s)$  によって上書きされる。

$af$   $fact = 'N'$  または  $'E'$  の場合、 $af$  は出力引数になる。この引数は終了時に、元の行列  $A$  ( $fact = 'N'$  の場合 ) または平衡化された行列  $A$  ( $fact = 'E'$  の場合 ) のコレスキー因子分解  $A = U^H U$  または  $A = L L^H$  で得られた三角係数  $U$  または  $L$  を返す。平衡化された行列の形式は、 $a$  の説明を参照のこと。

$b$   $equed = 'Y'$  の場合、 $diag(s) * B$  によって上書きされる。  
 $equed = 'N'$  の場合は変更されない。

<i>s</i>	この配列は、 <i>fact</i> ≠ 'F' の場合は出力引数になる。 「入力引数」セクションの <i>s</i> の説明を参照。
<i>rcond</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 平衡化後の行列 <i>A</i> の条件数の逆数の推定値 (平衡化が行われる場合)。 <i>rcond</i> がマシンの精度より小さい場合 (特に、 <i>rcond</i> = 0 の場合) は、行列は有効な精度で特異になる。この条件は、 <i>info</i> > 0 のリターンコードで示される。
<i>ferr</i> , <i>berr</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 配列、次元は $\max(1, nrhs)$ 以上。各解ベクトルの成分ごとの前進誤差と相対後退誤差を格納する。
<i>equed</i>	<i>fact</i> ≠ 'F' の場合、 <i>equed</i> は出力引数になる。この引数は、実行された平衡化の形式を指定する (「入力引数」セクションの <i>equed</i> の説明を参照)。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了した。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正である。 <i>info</i> = <i>i</i> で、 <i>i</i> ≤ <i>n</i> の場合、次数 <i>i</i> の先頭の小行列式 (したがって、行列 <i>A</i> そのもの) が正定値になっていないため、因子分解を実行できず、解と誤差範囲は計算できなかった。 <i>rcond</i> = 0 が返される。 <i>info</i> = <i>i</i> で、 <i>i</i> = <i>n</i> + 1 の場合、 <i>U</i> は特異でないが、 <i>rcond</i> がマシンの精度より小さいため、行列は有効な精度で特異になる。しかし、この場合は、解と誤差範囲が計算される。これは、計算された解の精度が、 <i>rcond</i> の値から想定される精度より高くなる場合があるためである。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

posvx ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	サイズ ( <i>n</i> , <i>n</i> ) の行列 <i>A</i> を格納する。
<i>b</i>	サイズ ( <i>n</i> , <i>nrhs</i> ) の行列 <i>B</i> を格納する。
<i>x</i>	サイズ ( <i>n</i> , <i>nrhs</i> ) の行列 <i>X</i> を格納する。
<i>af</i>	サイズ ( <i>n</i> , <i>n</i> ) の行列 <i>AF</i> を格納する。

<i>s</i>	長さ ( <i>n</i> ) のベクトルを格納する。各成分のデフォルト値は、 $s(i) = 1.0\_WP$ である。
<i>ferr</i>	長さ ( <i>nrhs</i> ) のベクトルを格納する。
<i>berr</i>	長さ ( <i>nrhs</i> ) のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U' である。
<i>fact</i>	'N'、'E'、または 'F' でなければならない。デフォルト値は 'N' である。 <i>fact</i> = 'F' の場合、 <i>af</i> を指定しなければならない。指定しない場合はエラーが返される。
<i>equed</i>	'N' または 'Y' でなければならない。デフォルト値は 'N' である。

---

### ?ppsv

圧縮形式の対称 (エルミート) 正定値行列 *A* を係数行列とする、複数の右辺を持つ連立 1 次方程式を解く。

---

#### 構文

##### Fortran 77:

```
call sppsv(uplo, n, nrhs, ap, b, ldb, info)
call dppsv(uplo, n, nrhs, ap, b, ldb, info)
call cppsv(uplo, n, nrhs, ap, b, ldb, info)
call zppsv(uplo, n, nrhs, ap, b, ldb, info)
```

##### Fortran 95:

```
call ppsv(a, b [,uplo] [,info])
```

#### 説明

このルーチンは、実数または複素数連立 1 次方程式  $AX=B$  を、*X* について解く。*A* は圧縮形式で格納される  $n \times n$  の実対称 / エルミート正定値行列、行列 *B* の列は個々の右辺、*X* の列はそれに対応する解である。

コレスキー因子分解を使用して、*A* を  $A = U^H U$  (*uplo* = 'U' の場合)



または  $A = LL^H$  ( $uplo = 'L'$  の場合) として因子分解する。 $U$  は上三角行列、 $L$  は下三角行列である。次に、 $A$  の因子分解された形式を使用して、連立方程式  $AX = B$  を解く。

### 入力パラメータ

<code>uplo</code>	CHARACTER*1。'U' または 'L' でなければならない。 $A$ の上三角部分と下三角部分のどちらを格納するか、また $A$ をどのように因子分解するかを指定する。 $uplo = 'U'$ の場合、配列 $a$ には行列 $A$ の上三角部分が格納され、 $A$ は $U^H U$ として因子分解される。 $uplo = 'L'$ の場合、配列 $a$ には行列 $A$ の下三角部分が格納され、 $A$ は $LL^H$ として因子分解される。
<code>n</code>	INTEGER。行列 $A$ の次数 ( $n \geq 0$ )。
<code>nrhs</code>	INTEGER。右辺の数。 $B$ の列数 ( $nrhs \geq 0$ )。
<code>ap, b</code>	REAL (sppsv の場合) DOUBLE PRECISION (dppsv の場合) COMPLEX (cpcsv の場合) DOUBLE COMPLEX (zpcsv の場合)。 配列: $ap(*)$ , $b(ldb, *)$ 。 配列 $ap$ には、( $uplo$ の指定に従って) 行列 $A$ の上三角部分または下三角部分を圧縮格納形式で格納する (「 <a href="#">行列の格納形式</a> 」を参照)。 $ap$ の次元は $\max(1, n(n+1)/2)$ 以上でなければならない。 配列 $b$ には、行列 $B$ が格納される。この行列の列は、連立方程式の右辺である。 $b$ の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。
<code>ldb</code>	INTEGER。 $b$ の第 1 次元。 $ldb \geq \max(1, n)$ 。

### 出力パラメータ

<code>ap</code>	$info = 0$ の場合、 $uplo$ の指定に従って、圧縮格納形式の $A$ の上三角部分または下三角部分が、コレスキー係数 $U$ あるいは $L$ によって上書きされる。
<code>b</code>	解の行列 $X$ によって上書きされる。
<code>info</code>	INTEGER。 $info = 0$ の場合、実行は正常に終了した。 $info = -i$ の場合、 $i$ 番目のパラメータの値が不正である。 $info = i$ の場合、次数 $i$ の先頭の Minor 行列式 (したがって、行列 $A$ そのもの) が正定値になっていないため、因子分解を実行できず、解は計算されなかった。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ppsv ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ap</i> を意味する。サイズ $(n*(n+1)/2)$ の配列 <i>A</i> を格納する。
<i>b</i>	サイズ $(n,nrhs)$ の行列 <i>B</i> を格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U' である。

---

### ?ppsvx

コレスキー因子分解を使用して、圧縮形式の対称 (エルミート) 正定値行列 *A* を係数行列とする連立 1 次方程式の解を計算し、解の誤差範囲を示す。

---

#### 構文

##### Fortran 77:

```
call sppsvx(FACT, uplo, N, NRHS, Ap, Afp, EQUED, s, B, LDB, X, LDX,  
            RCOND, FERR, BERR, WORK, IWORK, INFO)  
call dppsvx(FACT, uplo, N, NRHS, Ap, Afp, EQUED, s, B, LDB, X, LDX,  
            RCOND, FERR, BERR, WORK, IWORK, INFO)  
call cppsvx(FACT, uplo, N, NRHS, Ap, Afp, EQUED, s, B, LDB, X, LDX,  
            RCOND, FERR, BERR, WORK, rWORK, INFO)  
call zppsvx(FACT, uplo, N, NRHS, Ap, Afp, EQUED, s, B, LDB, X, LDX,  
            rCOND, FERR, BERR, WORK, rWORK, INFO)
```

##### Fortran 95:

```
call ppsvx (a, b, x [,uplo] [,af] [,fact] [,equed] [,s] [,ferr] [,berr]  
            [,rcond] [,info])
```

## 説明

このルーチンは、コレスキー因子分解  $A=U^H U$  または  $A=LL^H$  を使用して、実数または複素数連立 1 次方程式  $AX=B$  の解を計算する。 $A$  は圧縮形式で格納される  $n \times n$  の対称 / エルミート正定値行列、行列  $B$  の列は個々の右辺、 $X$  の列はそれに対応する解である。

このルーチンは、解の誤差範囲と条件数の推定値も示す。

ルーチン `?ppsvx` は、以下の手順を実行する。

1.  $fact = 'E'$  の場合、連立方程式を平衡化するための実スケール係数  $s$  を計算する。

$$\text{diag}(s) * A * \text{diag}(s) * \text{diag}(s)^{-1} * X = \text{diag}(s) * B$$

連立方程式が平衡化されるかどうかは、行列  $A$  のスケーリングによって決まる。平衡化が行われる場合は、 $A$  は  $\text{diag}(s) * A * \text{diag}(s)$  によって上書きされ、 $B$  は  $\text{diag}(s) * B$  によって上書きされる。

2.  $fact = 'N'$  または  $'E'$  の場合、コレスキー分解を使用して、( $fact = 'E'$  の場合は平衡化の後に) 行列  $A$  を次のように因子分解する。

$A = U^H U$  ( $uplo = 'U'$  の場合)、または

$A = L L^H$  ( $uplo = 'L'$  の場合)。

$U$  は上三角行列で、 $L$  は下三角行列である。

3. 先頭の  $i \times i$  の主小行列式が正定値でない場合は、ルーチンは  $info = i$  を返す。それ以外の場合、 $A$  の因子分解された形式を使用して、行列  $A$  の条件数を推定する。条件数の逆数がマシンの精度より小さい場合は、警告として  $info = n + 1$  を返すが、後で説明するようにルーチンは続行され、 $X$  について解を算出し、誤差範囲を計算する。

4.  $A$  の因子分解された形式を使用して、連立方程式を  $X$  について解く。

5. 計算された解の行列の精度を繰り返し改善し、解の誤差範囲と後退誤差の推定値を計算する。

6. 平衡化が行われている場合は、平衡化が行われる前の元の連立方程式を解くために、 $\text{diag}(s)$  によって行列  $X$  を事前に乗算する。

## 入力パラメータ

**fact** CHARACTER\*1. 'F'、'N'、または 'E' でなければならない。

ルーチンの開始時に行列  $A$  の因子分解された形式が与えられるかどうか、与えられない場合は、行列  $A$  を因子分解する前に  $A$  を平衡化するかどうかを指定する。

$fact = 'F'$  の場合、開始時に、 $afp$  に  $A$  の因子分解された形式が格納される。 $equed = 'Y'$  の場合、行列  $A$  は、 $s$  で指定されたスケール係数によって平衡化されている。

$ap$  と  $afp$  は変更されない。

$fact = 'N'$  の場合、行列  $A$  を  $afp$  にコピーし、因子分解を実行する。 $fact = 'E'$  の場合、必要に応じて行列  $A$  を平衡化してから、 $afp$  にコピーし、因子分解を実行する。

$uplo$

CHARACTER\*1。'U' または 'L' でなければならない。

$A$  の上三角部分と下三角部分のどちらを格納するか、また  $A$  をどのように因子分解するかを指定する。

$uplo = 'U'$  の場合、配列  $ap$  には行列  $A$  の上三角部分が格納され、 $A$  は  $U^H U$  として因子分解される。

$uplo = 'L'$  の場合、配列  $ap$  には行列  $A$  の下三角部分が格納され、 $A$  は  $LL^H$  として因  $q$  分解される。

$n$

INTEGER。行列  $A$  の次数 ( $n \geq 0$ )。

$nrhs$

INTEGER。右辺の数。  $B$  の列数 ( $nrhs \geq 0$ )。

$ap, afp, b, work$

REAL (sppsvx の場合)

DOUBLE PRECISION (dppsvx の場合)

COMPLEX (cspsvx の場合)

DOUBLE COMPLEX (zppsvx の場合)。

配列:  $ap(*)$ ,  $afp(*)$ ,  $b(lb, *)$ ,  $work(*)$ 。

配列  $ap$  には、元の対称 / エルミート行列  $A$  の上三角部分または下三角部分が圧縮格納形式で格納される (「[行列の格納形式](#)」を参照)。

$fact = 'F'$  で  $equed = 'Y'$  の場合、 $ap$  には平衡化された行列

$diag(s)*A*diag(s)$  が格納されている。

配列  $afp$  は、 $fact = 'F'$  の場合は入力引数になる。この配列には、 $A$  のコレスキー因子分解で得られた三角係数  $U$  または  $L$  が、 $A$  と同じ格納形式で格納される。 $equed$  が 'N' でない場合は、 $afp$  は、平衡化された行列  $A$  の因子分解された形式になる。

配列  $b$  には、行列  $B$  が格納される。この行列の列は、連立方程式の右辺である。

$work(*)$  は、ワークスペース配列である。

配列  $ap$  と  $afp$  の次元は  $\max(1, n(n+1)/2)$  以上でなければならない。 $b$  の第 2 次元は  $\max(1, nrhs)$  以上でなければならない。 $work$  の次元は  $\max(1, 3*n)$  (実数型の場合) または  $\max(1, 2*n)$  (複素数型の場合) 以上でなければならない。

<i>ldb</i>	INTEGER。 <i>b</i> の第 1 次元。 $ldb \geq \max(1, n)$ 。
<i>equed</i>	CHARACTER*1。 'N' または 'Y' でなければならない。 <i>equed</i> は、 <i>fact</i> = 'F' の場合は入力引数になる。 この引数は、実行された平衡化の形式を指定する。 <i>equed</i> = 'N' の場合、平衡化は行われていない ( <i>fact</i> = 'N' の場合は常に真)。 <i>equed</i> = 'Y' の場合、平衡化が行われ、 <i>A</i> は $\text{diag}(s) * A * \text{diag}(s)$ で置き換えられている。
<i>s</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 配列、次元は ( <i>n</i> )。 配列 <i>s</i> には、 <i>A</i> のスケール係数が格納される。 この配列は、 <i>fact</i> = 'F' の場合にのみ入力引数になり、それ以外の場合は出力引数になる。 <i>equed</i> = 'N' の場合、 <i>s</i> は使用されない。 <i>fact</i> = 'F' で、 <i>equed</i> = 'Y' の場合、 <i>s</i> の各成分は正の値でなければならない。
<i>ldx</i>	INTEGER。 出力配列 <i>x</i> の第 1 次元。 $ldx \geq \max(1, n)$ 。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, n)$ 以上。 実数型でのみ使用される。
<i>rwork</i>	REAL (cppsux の場合) DOUBLE PRECISION (zppsux の場合)。 ワークスペース配列、次元は $\max(1, n)$ 以上。 複素数型でのみ使用される。

## 出力パラメータ

<i>x</i>	REAL (sppsux の場合) DOUBLE PRECISION (dppsux の場合) COMPLEX (cppsux の場合) DOUBLE COMPLEX (zppsux の場合)。 配列、次元は ( <i>ldx</i> , *)。  <i>info</i> = 0 または <i>info</i> = <i>n</i> +1 の場合、配列 <i>x</i> には、元の連立方程式の解の行列 <i>X</i> が格納される。 ただし、 <i>equed</i> = 'Y' の場合、 <i>A</i> と <i>B</i> は終了時に修正され、平衡化された連立方程式の解は $\text{diag}(s)^{-1} * X$ になる。 <i>x</i> の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。
----------	---

<i>ap</i>	配列 <i>ap</i> は、 <i>fact</i> = 'F' または 'N' の場合、または <i>fact</i> = 'E' で <i>equed</i> = 'N' の場合、終了時に変更されない。 <i>fact</i> = 'E' で <i>equed</i> = 'Y' の場合、 <i>A</i> は $\text{diag}(s) * A * \text{diag}(s)$ によって上書きされる。
<i>afp</i>	<i>fact</i> = 'N' または 'E' の場合、 <i>afp</i> は出力引数になる。 この引数は終了時に、元の行列 <i>A</i> ( <i>fact</i> = 'N' の場合) または平衡化された行列 <i>A</i> ( <i>fact</i> = 'E' の場合) のコレスキー因子分解 $A = U^H U$ または $A = LL^H$ で得られた三角係数 <i>U</i> または <i>L</i> を返す。平衡化された行列の形式は、 <i>ap</i> の説明を参照のこと。
<i>b</i>	<i>equed</i> = 'Y' の場合、 $\text{diag}(s) * B$ によって上書きされる。 <i>equed</i> = 'N' の場合は変更されない。
<i>s</i>	この配列は、 <i>fact</i> ≠ 'F' の場合は出力引数になる。 「入力引数」セクションの <i>s</i> の説明を参照。
<i>rcond</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 平衡化後の行列 <i>A</i> の条件数の逆数の推定値 (平衡化が行われる場合)。 <i>rcond</i> がマシンの精度より小さい場合 (特に、 <i>rcond</i> = 0 の場合) は、行列は有効な精度で特異になる。この条件は、 <i>info</i> > 0 のリターンコードで示される。
<i>ferr, berr</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 配列、次元は $\max(1, \text{nrhs})$ 以上。各解ベクトルの成分ごとの前進誤差と相対後退誤差を格納する。
<i>equed</i>	<i>fact</i> ≠ 'F' の場合、 <i>equed</i> は出力引数になる。この引数は、実行された平衡化の形式を指定する (「入力引数」セクションの <i>equed</i> の説明を参照)。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了した。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正である。 <i>info</i> = <i>i</i> で、 $i \leq n$ の場合、次数 <i>i</i> の先頭の Minor 行列式 (したがって、行列 <i>A</i> そのもの) が正定値になっていないため、因子分解を実行できず、解と誤差範囲は計算できなかった。 <i>rcond</i> = 0 が返される。 <i>info</i> = <i>i</i> で、 $i = n + 1$ の場合、 <i>U</i> は特異でないが、 <i>rcond</i> がマシンの精度より小さいため、行列は有効な精度で特異になる。しかし、この場合は、解と誤差範囲が計算される。これは、計算された解の精度が、 <i>rcond</i> の値から想定される精度より高くなる場合があるためである。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ppsvx ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ap</i> を意味する。 サイズ $(n*(n+1)/2)$ の配列 <i>A</i> を格納する。
<i>b</i>	サイズ $(n, nrhs)$ の行列 <i>B</i> を格納する。
<i>x</i>	サイズ $(n, nrhs)$ の行列 <i>X</i> を格納する。
<i>af</i>	Fortran 77 インターフェイスでの引数 <i>afp</i> を意味する。 サイズ $(n*(n+1)/2)$ の配列 <i>AF</i> を格納する。
<i>s</i>	長さ $(n)$ のベクトルを格納する。各成分のデフォルト値は、 $s(i) = 1.0\_WP$ である。
<i>ferr</i>	長さ $(nrhs)$ のベクトルを格納する。
<i>berr</i>	長さ $(nrhs)$ のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U' である。
<i>fact</i>	'N'、'E'、または 'F' でなければならない。デフォルト値は 'N' である。 <i>fact</i> = 'F' の場合、 <i>af</i> を指定しなければならない。指定しない場合はエラーが返される。
<i>equad</i>	'N' または 'Y' でなければならない。デフォルト値は 'N' である。

## ?pbsv

対称/エルミート正定値帯行列 *A* を係数行列とする、複数の右辺を持つ連立 1 次方程式を解く。

### 構文

#### Fortran 77:

```
call spbsv(uplo, n, kd, nrhs, ab, ldab, b, ldb, info)
call dpbsv(uplo, n, kd, nrhs, ab, ldab, b, ldb, info)
```

```
call cpbsv(uplo, n, kd, nrhs, ab, ldab, b, ldb, info)
call zpbsv(uplo, n, kd, nrhs, ab, ldab, b, ldb, info)
```

## Fortran 95:

```
call pbsv(a, b [,uplo] [,info])
```

## 説明

このルーチンは、実数または複素数連立 1 次方程式  $AX=B$  を、 $X$  について解く。 $A$  は  $n \times n$  の対称 / エルミート 正定値帯行列、行列  $B$  の列は個々の右辺、 $X$  の列はそれに対応する解である。

コレスキー因子分解を使用して、 $A$  を  $A=U^H U$  ( $uplo='U'$  の場合)

または  $A=LL^H$  ( $uplo='L'$  の場合) として因子分解する。 $U$  は  $A$  と同じ数の優対角成分を持つ上三角帯行列で、 $L$  は  $A$  と同じ数の劣対角成分を持つ下三角帯行列である。次に、 $A$  の因子分解された形式を使用して、連立方程式  $AX=B$  を解く。

## 入力パラメータ

<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 $A$ の上三角部分と下三角部分のどちらを配列 <i>ab</i> に格納するか、また $A$ をどのように因子分解するかを指定する。 <i>uplo</i> = 'U' の場合、配列 <i>ab</i> には行列 $A$ の上三角部分が格納され、 $A$ は $U^H U$ として因子分解される。 <i>uplo</i> = 'L' の場合、配列 <i>ab</i> には行列 $A$ の下三角部分が格納され、 $A$ は $LL^H$ として因子分解される。
<i>n</i>	INTEGER。行列 $A$ の次数 ( $n \geq 0$ )。
<i>kd</i>	INTEGER。行列 $A$ の優対角成分の数 ( <i>uplo</i> = 'U' の場合) または劣対角成分の数 ( <i>uplo</i> = 'L' の場合)。 ( $kd \geq 0$ )。
<i>nrhs</i>	INTEGER。右辺の数。 $B$ の列数 ( $nrhs \geq 0$ )。
<i>ab, b</i>	REAL (spbsv の場合) DOUBLE PRECISION (dpbsv の場合) COMPLEX (cpbsv の場合) DOUBLE COMPLEX (zpbsv の場合)。 配列: <i>ab</i> ( <i>ldab</i> , *), <i>b</i> ( <i>ldb</i> , *)。 配列 <i>ab</i> には、 <i>uplo</i> の指定に従って、行列 $A$ の上三角部分または下三角部分が帯形式で格納される (「 <a href="#">行列の格納形式</a> 」を参照)。 <i>ab</i> の第 2 次元は $\max(1, n)$ 以上でなければならない。配列 <i>b</i> には、行列 $B$ が格納される。この行列の列は、連立方程式の右辺である。 <i>b</i> の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。



*ldab* INTEGER。配列 *ab* の第 1 次元 ( $ldab \geq kd+1$ )。

*ldb* INTEGER。 *b* の第 1 次元。  $ldb \geq \max(1, n)$ 。

### 出力パラメータ

*ab* *uplo* の指定に従って、( 帯形式の ) *A* の上三角部分または下三角部分が、コレスキー係数 *U* または *L* によって、*A* と同じ格納形式で上書きされる。

*b* 解の行列 *X* によって上書きされる。

*info* INTEGER。 *info* = 0 の場合、実行は正常に終了した。  
*info* = -*i* の場合、*i* 番目のパラメータの値が不正である。  
*info* = *i* の場合、次数 *i* の先頭の Minor 行列式 (したがって、行列 *A* そのもの) が正定値になっていないため、因子分解を実行できず、解は計算されなかった。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

pbsv ルーチンのインターフェイス特有の詳細を以下に示す。

*a* Fortran 77 インターフェイスでの引数 *ab* を意味する。サイズ  $(kd+1, n)$  の配列 *A* を格納する。

*b* サイズ  $(n, nrhs)$  の行列 *B* を格納する。

*uplo* 'U' または 'L' でなければならない。デフォルト値は 'U' である。

## ?pbsvx

コレスキー因子分解を使用して、対称(エルミート)正定値帯行列  $A$  を係数行列とする連立1次方程式の解を計算し、解の誤差範囲を示す。

---

### 構文

#### Fortran 77:

```
call spbsvx(FACT, uplo, N, kd, NRHS, Ab, ldab, Afb, ldafb, EQUED, s, B,
            LDB, X, LDX, RCOND, FERR, BERR, WORK, IWORK, INFO)
call dpbsvx(FACT, uplo, N, kd, NRHS, Ab, ldab, Afb, ldafb, EQUED, s, B,
            LDB, X, LDX, RCOND, FERR, BERR, WORK, IWORK, INFO)
call cpbsvx(FACT, uplo, N, kd, NRHS, Ab, ldab, Afb, ldafb, EQUED, s, B,
            LDB, X, LDX, RCOND, FERR, BERR, WORK, IWORK, INFO)
call zpbsvx(FACT, uplo, N, kd, NRHS, Ab, ldab, Afb, ldafb, EQUED, s, B,
            LDB, X, LDX, RCOND, FERR, BERR, WORK, IWORK, INFO)
```

#### Fortran 95:

```
call pbsvx(a, b, x [,uplo] [,af] [,fact] [,equed] [,s] [,ferr] [,berr]
            [,rcond] [,info])
```

### 説明

このルーチンは、コレスキー因子分解  $A=U^H U$  または  $A=LL^H$  を使用して、実数または複素数連立1次方程式  $AX=B$  の解を計算する。 $A$  は  $n \times n$  の対称/エルミート正定値帯行列、行列  $B$  の列は個々の右辺、 $X$  の列はそれに対応する解である。

このルーチンは、解の誤差範囲と条件数の推定値も示す。

ルーチン ?pbsvx は、以下の手順を実行する。

1.  $fact = 'E'$  の場合、連立方程式を平衡化するための実スケール係数  $s$  を計算する。

$$\text{diag}(s) * A * \text{diag}(s) * \text{diag}(s)^{-1} * X = \text{diag}(s) * B$$

連立方程式が平衡化されるかどうかは、行列  $A$  のスケーリングによって決まる。平衡化が行われる場合は、 $A$  は  $\text{diag}(s) * A * \text{diag}(s)$  によって上書きされ、 $B$  は  $\text{diag}(s) * B$  によって上書きされる。

2.  $fact = 'N'$  または  $'E'$  の場合、コレスキー分解を使用して、( $fact = 'E'$  の場合は平衡化の後に) 行列  $A$  を次のように因子分解する。

$A = U^H U$  ( $uplo = 'U'$  の場合)、または  
 $A = L L^H$  ( $uplo = 'L'$  の場合)。  
 $U$  は上三角帯行列で、 $L$  は下三角帯行列である。

3. 先頭の  $i \times i$  の主小行列式が正定値でない場合は、ルーチンは  $info = i$  を返す。それ以外の場合、 $A$  の因子分解された形式を使用して、行列  $A$  の条件数を推定する。条件数の逆数がマシンの精度より小さい場合は、警告として  $info = n + 1$  を返すが、後で説明するようにルーチンは続行され、 $X$  について解を算出し、誤差範囲を計算する。

4.  $A$  の因子分解された形式を使用して、連立方程式を  $X$  について解く。

5. 計算された解の行列の精度を繰り返し改善し、解の誤差範囲と後退誤差の推定値を計算する。

6. 平衡化が行われている場合は、平衡化が行われる前の元の連立方程式を解くために、 $diag(s)$  によって行列  $X$  を事前に乗算する。

## 入力パラメータ

**fact** CHARACTER\*1。'F'、'N'、または 'E' でなければならない。  
 ルーチンの開始時に行列  $A$  の因子分解された形式が与えられるかどうか、与えられない場合は、行列  $A$  を因子分解する前に  $A$  を平衡化するかどうかを指定する。  
 $fact = 'F'$  の場合、開始時に、 $afb$  に  $A$  の因子分解された形式が格納される。 $equet = 'Y'$  の場合、行列  $A$  は、 $s$  で指定されたスケール係数によって平衡化されている。  
 $ab$  と  $afb$  は変更されない。  
 $fact = 'N'$  の場合、行列  $A$  を  $afb$  にコピーし、因子分解を実行する。  
 $fact = 'E'$  の場合、必要に応じて行列  $A$  を平衡化してから、 $afb$  にコピーし、因子分解を実行する。

**uplo** CHARACTER\*1。'U' または 'L' でなければならない。  
 $A$  の上三角部分と下三角部分のどちらを格納するか、また  $A$  をどのように因子分解するかを指定する。  
 $uplo = 'U'$  の場合、配列  $ab$  には行列  $A$  の上三角部分が格納され、 $A$  は  $U^H U$  として因子分解される。  
 $uplo = 'L'$  の場合、配列  $ab$  には行列  $A$  の下三角部分が格納され、 $A$  は  $LL^H$  として因子分解される。

**n** INTEGER。行列  $A$  の次数 ( $n \geq 0$ )。

<i>kd</i>	INTEGER。 <i>A</i> の優対角成分または劣対角成分の数 ( $kd \geq 0$ )。
<i>nrhs</i>	INTEGER。 右辺の数。 <i>B</i> の列数 ( $nrhs \geq 0$ )。
<i>ab</i> , <i>afb</i> , <i>b</i> , <i>work</i>	<p>REAL (spbsvx の場合 )  DOUBLE PRECISION (dpbsvx の場合 )  COMPLEX (cpbsvx の場合 )  DOUBLE COMPLEX (zpbsvx の場合 )。</p> <p>配列 : <i>ab</i>(<i>ldab</i>, *), <i>afb</i>(<i>ldab</i>, *), <i>b</i>(<i>ldb</i>, *), <i>work</i>(*)。</p> <p>配列 <i>ab</i> には、行列 <i>A</i> の上三角部分または下三角部分が帯形式で格納される (「<a href="#">行列の格納形式</a>」を参照)。  <i>fact</i> = 'F' で <i>equed</i> = 'Y' の場合、<i>ab</i> には平衡化された行列 <i>diag(s)*A*diag(s)</i> が格納されている。<i>ab</i> の第 2 次元は <math>\max(1, n)</math> 以上でなければならない。</p> <p>配列 <i>afb</i> は、<i>fact</i> = 'F' の場合は入力引数になる。  この配列には、帯行列 <i>A</i> のコレスキー因子分解で得られた三角係数 <i>U</i> または <i>L</i> が、<i>A</i> と同じ格納形式で格納される。<i>equed</i> = 'Y' の場合、<i>afb</i> は、平衡化された行列 <i>A</i> の因子分解された形式になる。  <i>afb</i> の第 2 次元は <math>\max(1, n)</math> 以上でなければならない。</p> <p>配列 <i>b</i> には、行列 <i>B</i> が格納される。この行列の列は、連立方程式の右辺である。  <i>b</i> の第 2 次元は <math>\max(1, nrhs)</math> 以上でなければならない。</p> <p><i>work</i>(*) は、ワークスペース配列。  <i>work</i> の次元は <math>\max(1, 3*n)</math> (実数型の場合) または <math>\max(1, 2*n)</math> (複素数型の場合) 以上でなければならない。</p>
<i>ldab</i>	INTEGER。 <i>ab</i> の第 1 次元。 $ldab \geq kd+1$ 。
<i>ldaafb</i>	INTEGER。 <i>afb</i> の第 1 次元。 $ldaafb \geq kd+1$ 。
<i>ldb</i>	INTEGER。 <i>b</i> の第 1 次元。 $ldb \geq \max(1, n)$ 。
<i>equed</i>	<p>CHARACTER*1。 'N' または 'Y' でなければならない。</p> <p><i>equed</i> は、<i>fact</i> = 'F' の場合は入力引数になる。この引数は、実行された平衡化の形式を指定する。</p> <p><i>equed</i> = 'N' の場合、平衡化は行われていない (<i>fact</i> = 'N' の場合は常に真)。</p> <p><i>equed</i> = 'Y' の場合、平衡化が行われ、<i>A</i> は <i>diag(s)*A*diag(s)</i> で置き換えられている。</p>

<i>s</i>	<p>REAL ( 単精度の場合 )  DOUBLE PRECISION ( 倍精度の場合 )。  配列、次元は (<i>n</i>)。  配列 <i>s</i> には、<i>A</i> のスケール係数が格納される。この配列は、<i>fact</i> = 'F' の場合にのみ入力引数になり、それ以外の場合は出力引数になる。  <i>equed</i> = 'N' の場合、<i>s</i> は使用されない。  <i>fact</i> = 'F' で、<i>equed</i> = 'Y' の場合、<i>s</i> の各成分は正の値でなければならない。</p>
<i>ldx</i>	INTEGER。出力配列 <i>x</i> の第 1 次元。 $ldx \geq \max(1, n)$ 。
<i>iwork</i>	<p>INTEGER。  ワークスペース配列、次元は <math>\max(1, n)</math> 以上。実数型でのみ使用される。</p>
<i>rwork</i>	<p>REAL (cpbsvx の場合 )  DOUBLE PRECISION (zpbsvx の場合 )。  ワークスペース配列、次元は <math>\max(1, n)</math> 以上。複素数型でのみ使用される。</p>

### 出力パラメータ

<i>x</i>	<p>REAL (spbsvx の場合 )  DOUBLE PRECISION (dpbsvx の場合 )  COMPLEX (cpbsvx の場合 )  DOUBLE COMPLEX (zpbsvx の場合 )。  配列、次元は (<i>ldx</i>, *)。</p> <p><i>info</i> = 0 または <i>info</i> = <i>n</i>+1 の場合、配列 <i>x</i> には、元の連立方程式の解の行列 <i>X</i> が格納される。  ただし、<i>equed</i> = 'Y' の場合、<i>A</i> と <i>B</i> は終了時に修正され、平衡化された連立方程式の解は <math>\text{diag}(s)^{-1} * X</math> になる。  <i>x</i> の第 2 次元は <math>\max(1, nrhs)</math> 以上でなければならない。</p>
<i>ab</i>	終了時に、 <i>fact</i> = 'E' で <i>equed</i> = 'Y' の場合、 <i>A</i> は $\text{diag}(s) * A * \text{diag}(s)$ によって上書きされる。
<i>afb</i>	<p><i>fact</i> = 'N' または 'E' の場合、<i>afb</i> は出力引数になる。この引数は終了時に、元の行列 <i>A</i> (<i>fact</i> = 'N' の場合) または平衡化された行列 <i>A</i> (<i>fact</i> = 'E' の場合) のコレスキー因子分解 <math>A = U^H U</math> または <math>A = L L^H</math> で得られた三角係数 <i>U</i> または <i>L</i> を返す。平衡化された行列の形式については、<i>ab</i> の説明を参照のこと。</p>
<i>b</i>	<p><i>equed</i> = 'Y' の場合、<math>\text{diag}(s) * B</math> によって上書きされる。  <i>equed</i> = 'N' の場合は変更されない。</p>

<i>s</i>	この配列は、 <i>fact</i> ≠ 'F' の場合は出力引数になる。 「入力引数」セクションの <i>s</i> の説明を参照。
<i>rcond</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 平衡化後の行列 <i>A</i> の条件数の逆数の推定値 (平衡化が行われる場合)。 <i>rcond</i> がマシンの精度より小さい場合 (特に、 <i>rcond</i> = 0 の場合) は、行列は有効な精度で特異になる。この条件は、 <i>info</i> > 0 のリターンコードで示される。
<i>ferr, berr</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 配列、次元は $\max(1, nrhs)$ 以上。各解ベクトルの成分ごとの前進誤差と相対後退誤差を格納する。
<i>equed</i>	<i>fact</i> ≠ 'F' の場合、 <i>equed</i> は出力引数になる。この引数は、実行された平衡化の形式を指定する (「入力引数」セクションの <i>equed</i> の説明を参照)。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了した。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正である。 <i>info</i> = <i>i</i> で、 <i>i</i> ≤ <i>n</i> の場合、次数 <i>i</i> の先頭の小行列式 (したがって、行列 <i>A</i> そのもの) が正定値になっていないため、因子分解を実行できず、解と誤差範囲は計算できなかった。 <i>rcond</i> = 0 が返される。 <i>info</i> = <i>i</i> で、 <i>i</i> = <i>n</i> + 1 の場合、 <i>U</i> は特異でないが、 <i>rcond</i> がマシンの精度より小さいため、行列は有効な精度で特異になる。しかし、この場合は、解と誤差範囲が計算される。これは、計算された解の精度が、 <i>rcond</i> の値から想定される精度より高くなる場合があるためである。

#### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

pbsvx ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ab</i> を意味する。 サイズ ( <i>kd</i> +1, <i>n</i> ) の配列 <i>A</i> を格納する。
<i>b</i>	サイズ ( <i>n</i> , <i>nrhs</i> ) の行列 <i>B</i> を格納する。
<i>x</i>	サイズ ( <i>n</i> , <i>nrhs</i> ) の行列 <i>X</i> を格納する。

<i>af</i>	Fortran 77 インターフェイスでの引数 <i>afb</i> を意味する。 サイズ $(kd+1, n)$ の配列 <i>AF</i> を格納する。
<i>s</i>	長さ ( <i>n</i> ) のベクトルを格納する。各成分のデフォルト値は、 $s(i) = 1.0\_WP$ である。
<i>ferr</i>	長さ ( <i>nrhs</i> ) のベクトルを格納する。
<i>berr</i>	長さ ( <i>nrhs</i> ) のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U' である。
<i>fact</i>	'N'、'E'、または 'F' でなければならない。デフォルト値は 'N' である。 <i>fact</i> = 'F' の場合、 <i>af</i> を指定しなければならない。指定しない場合はエラーが返される。
<i>equed</i>	'N' または 'Y' でなければならない。デフォルト値は 'N' である。

## ?ptsv

対称/エルミート正定値三重対角行列 *A* を係数行列とする、複数の右辺を持つ連立 1 次方程式を解く。

### 構文

#### Fortran 77:

```
call sptsv(n, nrhs, d, e, b, ldb, info)
call dptsv(n, nrhs, d, e, b, ldb, info)
call cptsv(n, nrhs, d, e, b, ldb, info)
call zptsv(n, nrhs, d, e, b, ldb, info)
```

#### Fortran 95:

```
call ptsv(d, e, b [,info])
```

### 説明

このルーチンは、実数または複素連立 1 次方程式  $AX=B$  を、*X* について解く。  
*A* は  $n \times n$  の対称/エルミート正定値三重対角行列、行列 *B* の列は個々の右辺、*X* の列はそれに対応する解である。

$A$  を  $A = LDL^H$  として因子分解する。次に、 $A$  の因子分解された形式を使用して、連立方程式  $AX = B$  を解く。

### 入力パラメータ

$n$	INTEGER。行列 $A$ の次数 ( $n \geq 0$ )。
$nrhs$	INTEGER。右辺の数。 $B$ の列数 ( $nrhs \geq 0$ )。
$d$	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 配列、次元は $\max(1, n)$ 以上。三重対角行列 $A$ の対角成分を格納する。
$e, b$	REAL (sptsv の場合) DOUBLE PRECISION (dptsv の場合) COMPLEX (cptsv の場合) DOUBLE COMPLEX (zptsv の場合)。 配列: $e(n-1), b(ldb, *)$ 。 配列 $e$ には、 $A$ の $(n-1)$ 個の劣対角成分が格納される。 配列 $b$ には、行列 $B$ が格納される。この行列の列は、連立方程式の右辺である。 $b$ の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。
$ldb$	INTEGER。 $b$ の第 1 次元。 $ldb \geq \max(1, n)$ 。

### 出力パラメータ

$d$	$A$ の $LDL^H$ 因子分解で得られた対角行列 $D$ の $n$ 個の対角成分によって上書きされる。
$e$	$A$ の因子分解で得られた単位二重対角係数 $L$ の $(n-1)$ 個の劣対角成分によって上書きされる。
$b$	解の行列 $X$ によって上書きされる。
$info$	INTEGER。 $info = 0$ の場合、実行は正常に終了した。 $info = -i$ の場合、 $i$ 番目のパラメータの値が不正である。 $info = i$ の場合、次数 $i$ の先頭の Minor 行列式 (したがって、行列 $A$ そのもの) が正定値になっていないため、解は計算されなかった。 $i = n$ でない限り、因子分解は完了していない。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。



ptsv ルーチンのインターフェイス特有の詳細を以下に示す。

- $d$                    長さ ( $n$ ) のベクトルを格納する。
- $e$                    長さ ( $n-1$ ) のベクトルを格納する。
- $b$                    サイズ ( $n, nrhs$ ) の行列  $B$  を格納する。

## ?ptsvx

因子分解  $A=LDL^H$  を使用して、対称 (エルミート) 正定値三重対角行列  $A$  を係数行列とする連立 1 次方程式の解を計算し、解の誤差範囲を示す。

### 構文

#### Fortran 77:

```
call sptsvx(FACT, N, NRHS, d, e, df, ef, B, LDB, X, LDX, RCOND, FERR,
            BERR, WORK, INFO)
call dptsvx(FACT, N, NRHS, d, e, df, ef, B, LDB, X, LDX, RCOND, FERR,
            BERR, WORK, INFO)
call cptsvx(FACT, N, NRHS, d, e, df, ef, B, LDB, X, LDX, RCOND, FERR,
            BERR, WORK, rWORK, INFO)
call zptsvx(FACT, N, NRHS, d, e, df, ef, B, LDB, X, LDX, RCOND, FERR,
            BERR, WORK, rWORK, INFO)
```

#### Fortran 95:

```
call ptsvx(d, e, b, x [,df] [,ef] [,fact] [,ferr] [,berr] [,rcond]
            [,info])
```

### 説明

このルーチンは、コレスキー因子分解  $A=LDL^H$  を使用して、実数または複素連立 1 次方程式  $AX=B$  の解を計算する。 $A$  は  $n \times n$  の対称 / エルミート正定値三重対角行列、行列  $B$  の列は個々の右辺、 $X$  の列はそれに対応する解である。

このルーチンは、解の誤差範囲と条件数の推定値も示す。

ルーチン ?ptsvx は、以下の手順を実行する。

1.  $fact = 'N'$  の場合、行列  $A$  を  $A = LDL^H$  として因子分解する。 $L$  は単位下二重対角行列で、 $D$  は対角行列である。この因子分解の形式は、 $A = U^H D U$  とみなせる。
2. 先頭の  $i \times i$  の主小行列式が正定値でない場合は、ルーチンは  $info = i$  を返す。それ以外の場合、 $A$  の因子分解された形式を使用して、行列  $A$  の条件数を推定する。条件数の逆数がマシンの精度より小さい場合は、警告として  $info = n + 1$  を返すが、後で説明するようにルーチンは続行され、 $X$  について解を算出し、誤差範囲を計算する。
3.  $A$  の因子分解された形式を使用して、連立方程式を  $X$  について解く。
4. 計算された解の行列の精度を繰り返し改善し、解の誤差範囲と後退誤差の推定値を計算する。

## 入力パラメータ

<i>fact</i>	CHARACTER*1。'F' または 'N' でなければならない。  ルーチンの開始時に行列 $A$ の因子分解された形式が与えられるかどうかを指定する。  $fact = 'F'$ の場合、開始時に、 $df$ と $ef$ に $A$ の因子分解された形式が格納される。配列 $d$ 、 $e$ 、 $df$ 、 $ef$ は変更されない。  $fact = 'N'$ の場合、行列 $A$ を $df$ と $ef$ にコピーし、因子分解を実行する。
<i>n</i>	INTEGER。行列 $A$ の次数 ( $n \geq 0$ )。
<i>nrhs</i>	INTEGER。右辺の数。 $B$ の列数 ( $nrhs \geq 0$ )。
<i>d</i> , <i>df</i> , <i>rwork</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 配列: $d(n)$ , $df(n)$ , $rwork(n)$ 。 配列 $d$ には、三重対角行列 $A$ の $n$ 個の対角成分が格納される。 配列 $df$ は、 $fact = 'F'$ の場合は入力引数になる。この配列には、開始時に、 $A$ の $LDL^H$ 因子分解で得られた対角行列 $D$ の $n$ 個の対角成分が格納される。 配列 $rwork$ は、複素数型でのみ使用されるワークスペース配列である。
<i>e</i> , <i>ef</i> , <i>b</i> , <i>work</i>	REAL (sptsvx の場合) DOUBLE PRECISION (dptsvx の場合) COMPLEX (cptsvx の場合) DOUBLE COMPLEX (zptsvx の場合)。 配列: $e(n-1)$ , $ef(n-1)$ , $b(ldb, *)$ , $work(*)$ 。 配列 $e$ には、三重対角行列 $A$ の $(n-1)$ 個の劣対角成分が格納される。

配列 *ef* は、*fact* = 'F' の場合は入力引数になる。この配列には、開始時に、*A* の  $LDL^H$  因子分解で得られた単位二重対角係数 *L* の  $(n-1)$  個の劣対角成分が格納される。

配列 *b* には、行列 *B* が格納される。この行列の列は、連立方程式の右辺である。

配列 *work* はワークスペース配列である。*work* の次元は、 $2*n$  (実数型の場合) または  $n$  (複素数型の場合) 以上でなければならない。

*ldb* INTEGER。 *b* のリーディング・ディメンジョン。  $ldb \geq \max(1, n)$ 。  
*ldx* INTEGER。 *x* のリーディング・ディメンジョン。  $ldx \geq \max(1, n)$ 。

### 出力パラメータ

*x* REAL (sptsvx の場合)  
DOUBLE PRECISION (dptsvx の場合)  
COMPLEX (cptsvx の場合)  
DOUBLE COMPLEX (zptsvx の場合)。  
配列、次元は (*ldx*, \*)。  
  
*info* = 0 または *info* =  $n+1$  の場合、配列 *x* には、連立方程式の解の行列 *X* が格納される。*x* の第 2 次元は  $\max(1, nrhs)$  以上でなければならない。  
  
*df*, *ef* これらの配列は、*fact* = 'N' の場合は出力引数になる。  
「入力引数」セクションの *df*, *ef* の説明を参照。  
  
*rcond* REAL (単精度の場合)  
DOUBLE PRECISION (倍精度の場合)。  
平衡化後の行列 *A* の条件数の逆数の推定値 (平衡化が行われる場合)。  
*rcond* がマシンの精度より小さい場合 (特に、*rcond* = 0 の場合) は、行列は有効な精度で特異になる。この条件は、*info* > 0 のリターンコードで示される。  
  
*ferr*, *berr* REAL (単精度の場合)  
DOUBLE PRECISION (倍精度の場合)。  
配列、次元は  $\max(1, nrhs)$  以上。各解ベクトルの成分ごとの前進誤差と相対後退誤差を格納する。  
  
*info* INTEGER。 *info* = 0 の場合、実行は正常に終了した。  
*info* = -*i* の場合、*i* 番目のパラメータの値が不正である。  
*info* = *i* で、 $i \leq n$  の場合、次数 *i* の先頭の Minor 行列式 (したがって、行列 *A* そのもの) が正定値になっていないため、因子分解を実行できず、解と誤差範囲は計算できなかった。*rcond* = 0 が返される。  
*info* = *i* で、 $i = n+1$  の場合、*U* は特異でないが、*rcond* がマシンの精度より小さいため、行列は有効な精度で特異になる。しかし、この

場合は、解と誤差範囲が計算される。これは、計算された解の精度が、*rcond* の値から想定される精度より高くなる場合があるためである。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ptsvx ルーチンのインターフェイス特有の詳細を以下に示す。

<i>d</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>e</i>	長さ ( <i>n</i> -1) のベクトルを格納する。
<i>b</i>	サイズ ( <i>n</i> , <i>nrhs</i> ) の行列 <i>B</i> を格納する。
<i>x</i>	サイズ ( <i>n</i> , <i>nrhs</i> ) の行列 <i>X</i> を格納する。
<i>df</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>ef</i>	長さ ( <i>n</i> -1) のベクトルを格納する。
<i>ferr</i>	長さ ( <i>nrhs</i> ) のベクトルを格納する。
<i>berr</i>	長さ ( <i>nrhs</i> ) のベクトルを格納する。
<i>fact</i>	'N' または 'F' でなければならない。デフォルト値は 'N' である。 <i>fact</i> = 'F' の場合、引数 <i>af</i> および <i>ipiv</i> の両方を指定しなければならない。指定しない場合は、エラーが返される。

---

## ?sysv

実対称または複素対称行列 *A* を係数行列とする、  
複数の右辺を持つ連立 1 次方程式の解を計算する。

---

### 構文

#### Fortran 77:

```
call ssysv(uplo, n, nrhs, a, lda, ipiv, b, ldb, work, lwork, info)
call dsysv(uplo, n, nrhs, a, lda, ipiv, b, ldb, work, lwork, info)
```

```
call csysv(uplo, n, nrhs, a, lda, ipiv, b, ldb, work, lwork, info)
call zsysv(uplo, n, nrhs, a, lda, ipiv, b, ldb, work, lwork, info)
```

#### Fortran 95:

```
call sysv(a, b [,uplo] [,ipiv] [,info])
```

#### 説明

このルーチンは、実数または複素連立 1 次方程式  $AX=B$  を、 $X$  について解く。  
 $A$  は  $n \times n$  の対称行列、行列  $B$  の列は個々の右辺、 $X$  の列はそれに対応する解である。

対角ピボット演算法を使用して、 $A$  を  $A=UDU^T$  または  $A=LDL^T$  として因子分解する。 $U$  (または  $L$ ) は置換行列と単位上 (下) 三角行列の積で、 $D$  は  $1 \times 1$  と  $2 \times 2$  の対角ブロックを持つ対称ブロック対角行列である。

次に、 $A$  の因子分解された形式を使用して、連立方程式  $AX=B$  を解く。

#### 入力パラメータ

<code>uplo</code>	CHARACTER*1。'U' または 'L' でなければならない。 $A$ の上三角部分と下三角部分のどちらを格納するか、また $A$ をどのように因子分解するかを指定する。 <code>uplo = 'U'</code> の場合、配列 <code>a</code> には行列 $A$ の上三角部分が格納され、 $A$ は $UDU^T$ として因子分解される。 <code>uplo = 'L'</code> の場合、配列 <code>a</code> には行列 $A$ の下三角部分が格納され、 $A$ は $LDL^T$ として因子分解される。
<code>n</code>	INTEGER。行列 $A$ の次数 ( $n \geq 0$ )。
<code>nrhs</code>	INTEGER。右辺の数。 $B$ の列数 ( $nrhs \geq 0$ )。
<code>a, b, work</code>	REAL (ssysv の場合) DOUBLE PRECISION (dsysv の場合) COMPLEX (csysv の場合) DOUBLE COMPLEX (zsysv の場合)。 配列: <code>a(lda, *)</code> , <code>b(ldb, *)</code> , <code>work(lwork)</code> 。 配列 <code>a</code> には、対称行列 $A$ の上三角部分または下三角部分が格納される ( <code>uplo</code> を参照)。 <code>a</code> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 配列 <code>b</code> には、行列 $B$ が格納される。この行列の列は、連立方程式の右辺である。 <code>b</code> の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。 <code>work(lwork)</code> は、ワークスペース配列である。
<code>lda</code>	INTEGER。 <code>a</code> の第 1 次元。 $lda \geq \max(1, n)$ 。

*ldb* INTEGER。 *b* の第 1 次元。  $ldb \geq \max(1, n)$ 。

*lwork* INTEGER。 配列 *work* のサイズ、  $lwork \geq 1$ 。

*lwork* = -1 の場合はワークスペースのクエリとみなされ、ルーチンは *work* 配列の最適サイズだけを計算し、その値を *work* 配列の最初のエントリとして返し、*xerbla* は *lwork* に関するエラー・メッセージを生成しない。*lwork* の推奨値は、「アプリケーション・ノート」を参照。

### 出力パラメータ

*a* *info* = 0 の場合、*a* は [?sytrf](#) による *A* の因子分解で得られたブロック対角行列 *D* と、係数 *U* (または *L*) の計算に使用された乗数によって上書きされる。

*b* *info* = 0 の場合、*b* は解の行列 *X* によって上書きされる。

*ipiv* INTEGER。  
配列、次元は  $\max(1, n)$  以上。  
[?sytrf](#) によって決まる、交換操作と *D* のブロック構造の詳細を格納する。  
*ipiv*(*i*) = *k* > 0 の場合、*d<sub>ii</sub>* は 1 × 1 の対角ブロックである。*A* の *i* 番目の行と列は、*k* 番目の行と列に交換される。  
*uplo* = 'U' で *ipiv*(*i*) = *ipiv*(*i*-1) = -*m* < 0 の場合、*D* の *i* 行 (*i*-1) 列に 2 × 2 のブロックが作成される。*A* の (*i*-1) 番目の行と列は、*m* 番目の行と列と交換される。  
*uplo* = 'L' で *ipiv*(*i*) = *ipiv*(*i*+1) = -*m* < 0 の場合、*D* の *i* 行 (*i*+1) 列に 2 × 2 のブロックが作成される。*A* の (*i*+1) 番目の行と列は、*m* 番目の行と列と交換される。

*work*(1) *info* = 0 の場合、終了時に、最適なパフォーマンスを得るために必要な *lwork* の最小値が *work*(1) に格納される。これ以降の実行には、この *lwork* の値を使用する。

*info* INTEGER。 *info* = 0 の場合、実行は正常に終了した。  
*info* = -*i* の場合、*i* 番目のパラメータの値が不正である。  
*info* = *i* の場合、*d<sub>ii</sub>* は 0 である。因子分解は完了したが、*D* が完全に特異であるため、解は計算されなかった。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

*sysv* ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	サイズ $(n,n)$ の行列 <i>A</i> を格納する。
<i>b</i>	サイズ $(n,nrhs)$ の行列 <i>B</i> を格納する。
<i>ipiv</i>	長さ $(n)$ のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U' である。

### アプリケーション・ノート

パフォーマンスを改善するには、 $lwork = n * blocksize$  に設定する。*blocksize* は、ブロック・アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。

必要なワークスペースの大きさがわからない場合は、最初の実行では  $lwork = -1$  に設定する。この設定は、ワークスペースのクエリとみなされる。ルーチンは、*work* 配列の最適なサイズだけを計算し、この値を *work* 配列の最初のエントリ *work*(1) として返す。[xerbla](#) は、*lwork* に関するエラー・メッセージを生成しない。終了時に、*work*(1) の値を確認し、これ以降の実行にはその値を使用する。

## ?sysvx

対角ピボット演算による因子分解を使用して、  
実対称または複素対称行列 *A* を係数行列とする  
連立 1 次方程式の解を計算し、解の誤差範囲を  
示す。

### 構文

#### Fortran 77:

```
call ssysvx(FACT, uplo, N, NRHS, A, lda, Af, ldaf, ipiv, B, LDB, X, LDX,
            RCOND, FERR, BERR, WORK, lWORK, iWORK, INFO)
call dsysvx(FACT, uplo, N, NRHS, A, lda, Af, ldaf, ipiv, B, LDB, X, LDX,
            RCOND, FERR, BERR, WORK, lWORK, iWORK, INFO)
call csysvx(FACT, uplo, N, NRHS, A, lda, Af, ldaf, ipiv, B, LDB, X, LDX,
            RCOND, FERR, BERR, WORK, lWORK, rWORK, INFO)
call zsysvx(FACT, uplo, N, NRHS, A, lda, Af, ldaf, ipiv, B, LDB, X, LDX,
            RCOND, FERR, BERR, WORK, lWORK, rWORK, INFO)
```

#### Fortran 95:

```
call sysvx(a, b, x[,uplo][,af][,ipiv][,fact][,ferr][,berr][,rcond]
[,info])
```

## 説明

このルーチンは、対角ピボット演算による因子分解を使用して、実数または複素連立 1 次方程式  $AX=B$  の解を計算する。A は  $n \times n$  の対称行列、行列 B の列は個々の右辺、X の列はそれに対応する解である。

このルーチンは、解の誤差範囲と条件数の推定値も示す。

ルーチン `?sysvx` は、以下の手順を実行する。

1.  $fact = 'N'$  の場合、対角ピボット演算法を使用して、行列 A を因子分解する。因子分解の形式は、 $A = U D U^T$  または  $A = L D L^T$  である。U (または L) は置換行列と単位上 (下) 三角行列の積で、D は  $1 \times 1$  と  $2 \times 2$  の対角ブロックを持つ対称ブロック対角行列である。
2.  $d_{i,i} = 0$  の場合、つまり D が完全に特異である場合は、ルーチンは  $info = i$  を返す。それ以外の場合、A の因子分解された形式を使用して、行列 A の条件数を推定する。条件数の逆数がマシンの精度より小さい場合は、警告として  $info = n + 1$  を返すが、後で説明するようにルーチンは続行され、X について解を算出し、誤差範囲を計算する。
3. A の因子分解された形式を使用して、連立方程式を X について解く。
4. 計算された解の行列の精度を繰り返し改善し、解の誤差範囲と後退誤差の推定値を計算する。

## 入力パラメータ

<i>fact</i>	CHARACTER*1。'F' または 'N' でなければならない。  ルーチンの開始時に行列 A の因子分解された形式が与えられるかどうかを指定する。  $fact = 'F'$ の場合、開始時に、 <i>af</i> と <i>ipiv</i> に A の因子分解された形式が格納される。配列 <i>a</i> 、 <i>af</i> 、 <i>ipiv</i> は変更されない。  $fact = 'N'$ の場合、行列 A を <i>af</i> にコピーし、因子分解を実行する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 A の上三角部分と下三角部分のどちらを格納するか、また A をどのように因子分解するかを指定する。



$uplo = 'U'$  の場合、配列  $a$  には対称行列  $A$  の上三角部分が格納され、 $A$  は  $UDU^T$  として因子分解される。  
 $uplo = 'L'$  の場合、配列  $a$  には対称行列  $A$  の下三角部分が格納され、 $A$  は  $LDL^T$  として因子分解される。

$n$  INTEGER。行列  $A$  の次数 ( $n \geq 0$ )。

$nrhs$  INTEGER。右辺の数。  $B$  の列数 ( $nrhs \geq 0$ )。

$a, af, b, work$  REAL (ssysvx の場合 )  
 DOUBLE PRECISION (dsysvx の場合 )  
 COMPLEX (csysvx の場合 ) DOUBLE COMPLEX (zsysvx の場合 )。  
 配列 :  $a(lda, *)$ ,  $af(ldaf, *)$ ,  $b ldb, *)$ ,  $work(*)$ 。

配列  $a$  には、対称行列  $A$  の上三角部分または下三角部分が格納される ( $uplo$  を参照)。  
 $a$  の第 2 次元は  $\max(1, n)$  以上でなければならない。

配列  $af$  は、 $fact = 'F'$  の場合は入力引数になる。  
 この配列には、[?sytrf](#) による因子分解  $A = UDU^T$  または  $A = LDL^T$  で得られた、ブロック対角行列  $D$  と、係数  $U$  または  $L$  の計算に使用された乗数が格納される。  
 $af$  の第 2 次元は  $\max(1, n)$  以上でなければならない。

配列  $b$  には、行列  $B$  が格納される。この行列の列は、連立方程式の右辺である。  
 $b$  の第 2 次元は  $\max(1, nrhs)$  以上でなければならない。

$work(*)$  は、次元 ( $lwork$ ) のワークスペース配列。

$lda$  INTEGER。  $a$  の第 1 次元。  $lda \geq \max(1, n)$ 。

$ldaf$  INTEGER。  $af$  の第 1 次元。  $ldaf \geq \max(1, n)$ 。

$ldb$  INTEGER。  $b$  の第 1 次元。  $ldb \geq \max(1, n)$ 。

$ipiv$  INTEGER。  
 配列、次元は  $\max(1, n)$  以上。  
 配列  $ipiv$  は、 $fact = 'F'$  の場合は入力引数になる。  
 この配列には、[?sytrf](#) によって決まる、交換操作と  $D$  のブロック構造の詳細が格納される。  
 $ipiv(i) = k > 0$  の場合、 $d_{ii}$  は  $1 \times 1$  の対角ブロックである。  $A$  の  $i$  番目の行と列は、 $k$  番目の行と列に交換される。  
 $uplo = 'U'$  で  $ipiv(i) = ipiv(i-1) = -m < 0$  の場合、 $D$  の  $i$  行 ( $i-1$ ) 列に  $2 \times 2$  のブロックが作成される。  $A$  の ( $i-1$ ) 番目の行と列は、 $m$  番目の行と列と交換される。

$uplo = 'L'$  で  $ipiv(i) = ipiv(i+1) = -m < 0$  の場合、 $D$  の  $i$  行 ( $i+1$ ) 列に  $2 \times 2$  のブロックが作成される。 $A$  の ( $i+1$ ) 番目の行と列は、 $m$  番目の行と列と交換される。

<i>ldx</i>	INTEGER。出力配列 $x$ のリーディング・ディメンジョン。 $ldx \geq \max(1, n)$ 。
<i>lwork</i>	INTEGER。work 配列のサイズ。 $lwork = -1$ の場合はワークスペースのクエリとみなされ、ルーチンは work 配列の最適サイズだけを計算し、その値を work 配列の最初のエントリとして返し、xerbla は <i>lwork</i> に関するエラー・メッセージを生成しない。 <i>lwork</i> の推奨値は、「アプリケーション・ノート」を参照。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, n)$ 以上。実数型でのみ使用される。
<i>rwork</i>	REAL (csysvx の場合 ) DOUBLE PRECISION (zsysvx の場合 )。 ワークスペース配列、次元は $\max(1, n)$ 以上。複素数型でのみ使用される。

## 出力パラメータ

<i>x</i>	REAL (ssysvx の場合 ) DOUBLE PRECISION (dsysvx の場合 ) COMPLEX (csysvx の場合 ) DOUBLE COMPLEX (zsysvx の場合 )。 配列、次元は ( <i>ldx</i> , *)。  $info = 0$ または $info = n+1$ の場合、配列 $x$ には、連立方程式の解の行列 $X$ が格納される。 $x$ の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。
<i>af</i> , <i>ipiv</i>	これらの配列は、 $fact = 'N'$ の場合は出力引数になる。 「入力引数」セクションの <i>af</i> , <i>ipiv</i> の説明を参照。
<i>rcond</i>	REAL (単精度の場合 ) DOUBLE PRECISION (倍精度の場合 )。 行列 $A$ の条件数の逆数の推定値。 <i>rcond</i> がマシンの精度より小さい場合 (特に、 $rcond = 0$ の場合) は、行列は有効な精度で特異になる。この条件は、 $info > 0$ のリターンコードで示される。

<i>ferr, berr</i>	REAL ( 単精度の場合 ) DOUBLE PRECISION ( 倍精度の場合 )。 配列、次元は $\max(1, nrhs)$ 以上。各解ベクトルの成分ごとの前進誤差と相対後退誤差を格納する。
<i>work(1)</i>	<i>info</i> = 0 の場合、終了時に、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が <i>work(1)</i> に格納される。これ以降の実行には、この <i>lwork</i> の値を使用する。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了した。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正である。 <i>info</i> = <i>i</i> で、 $i \leq n$ の場合、 $d_{ii}$ は完全に 0 である。因子分解は完了したが、ブロック対角行列 <i>D</i> が完全に特異であるため、解と誤差範囲は計算できなかった。 <i>rcond</i> = 0 が返される。 <i>info</i> = <i>i</i> で、 $i = n + 1$ の場合、 <i>D</i> は特異でないが、 <i>rcond</i> がマシンの精度より小さいため、行列は有効な精度で特異になる。しかし、この場合は、解と誤差範囲が計算される。これは、計算された解の精度が、 <i>rcond</i> の値から想定される精度より高くなる場合があるためである。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

sysvx ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	サイズ ( <i>n</i> , <i>n</i> ) の行列 <i>A</i> を格納する。
<i>b</i>	サイズ ( <i>n</i> , <i>nrhs</i> ) の行列 <i>B</i> を格納する。
<i>x</i>	サイズ ( <i>n</i> , <i>nrhs</i> ) の行列 <i>X</i> を格納する。
<i>af</i>	サイズ ( <i>n</i> , <i>n</i> ) の行列 <i>AF</i> を格納する。
<i>ipiv</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>ferr</i>	長さ ( <i>nrhs</i> ) のベクトルを格納する。
<i>berr</i>	長さ ( <i>nrhs</i> ) のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U' である。
<i>fact</i>	'N' または 'F' でなければならない。デフォルト値は 'N' である。 <i>fact</i> = 'F' の場合、引数 <i>af</i> および <i>ipiv</i> の両方を指定しなければならない。指定しない場合は、エラーが返される。

### アプリケーション・ノート

*lwork* は、実数型の場合は  $3 \times n$  以上、複素数型の場合は  $2 \times n$  以上でなければならない。パフォーマンスを向上させるには、

*lwork* =  $n \times \text{blocksize}$  に設定する。*blocksize* は、`?sytrf` に最適なブロックサイズである。

必要なワークスペースの大きさがわからない場合は、最初の実行では *lwork* = -1 に設定する。この設定は、ワークスペースのクエリとみなされる。ルーチンは、*work* 配列の最適なサイズだけを計算し、この値を *work* 配列の最初のエン트리 *work*(1) として返す。[xerbla](#) は、*lwork* に関するエラー・メッセージを生成しない。終了時に、*work*(1) の値を確認し、これ以降の実行にはその値を使用する。

---

## ?hesv

エルミート行列 *A* を係数行列とする、複数の右辺を持つ連立 1 次方程式の解を計算する。

---

### 構文

#### Fortran 77:

```
call chesv(uplo, n, nrhs, a, lda, ipiv, b, ldb, work, lwork, info)
call zhesv(uplo, n, nrhs, a, lda, ipiv, b, ldb, work, lwork, info)
```

#### Fortran 95:

```
call hesv(a, b [,uplo] [,ipiv] [,info])
```

### 説明

このルーチンは、実数または複素連立 1 次方程式  $AX=B$  を、*X* について解く。*A* は  $n \times n$  の対称行列、行列 *B* の列は個々の右辺、*X* の列はそれに対応する解である。

対角ピボット演算法を使用して、*A* を  $A=UDU^H$  または  $A=LDL^H$  として因子分解する。*U* (または *L*) は置換行列と単位上 (下) 三角行列の積で、*D* は  $1 \times 1$  と  $2 \times 2$  の対角ブロックを持つエルミート・ブロック対角行列である。

次に、*A* の因子分解された形式を使用して、連立方程式  $AX=B$  を解く。

### 入力パラメータ

*uplo* CHARACTER\*1. 'U' または 'L' でなければならない。

	$A$ の上三角部分と下三角部分のどちらを格納するか、また $A$ をどのように因子分解するかを指定する。
	$uplo = 'U'$ の場合、配列 $a$ には行列 $A$ の上三角部分が格納され、 $A$ は $UDU^H$ として因子分解される。
	$uplo = 'L'$ の場合、配列 $a$ には行列 $A$ の下三角部分が格納され、 $A$ は $LDL^H$ として因子分解される。
$n$	INTEGER。行列 $A$ の次数 ( $n \geq 0$ )。
$nrhs$	INTEGER。右辺の数。 $B$ の列数 ( $nrhs \geq 0$ )。
$a, b, work$	COMPLEX (chesv の場合 ) DOUBLE COMPLEX (zheshv の場合 )。 配列 : $a(lda, *)$ , $b(ldb, *)$ , $work(lwork)$ 。 配列 $a$ には、エルミート行列 $A$ の上三角部分または下三角部分が格納される ( $uplo$ を参照 )。 $a$ の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 配列 $b$ には、行列 $B$ が格納される。この行列の列は、連立方程式の右辺である。 $b$ の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。 $work(lwork)$ は、ワークスペース配列である。
$lda$	INTEGER。 $a$ の第 1 次元。 $lda \geq \max(1, n)$ 。
$ldb$	INTEGER。 $b$ の第 1 次元。 $ldb \geq \max(1, n)$ 。
$lwork$	INTEGER。配列 $work$ のサイズ。 $lwork \geq 1$ 。 $lwork = -1$ の場合はワークスペースのクエリとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエントリとして返し、 $xerbla$ は $lwork$ に関するエラー・メッセージを生成しない。 $lwork$ の推奨値は、「アプリケーション・ノート」を参照。

### 出力パラメータ

$a$	$info = 0$ の場合、 $a$ は <a href="#">?hetrf</a> による $A$ の因子分解で得られたブロック対角行列 $D$ と、係数 $U$ (または $L$ ) の計算に使用された乗数によって上書きされる。
$b$	$info = 0$ の場合、 $b$ は解の行列 $X$ によって上書きされる。
$ipiv$	INTEGER。 配列、次元は $\max(1, n)$ 以上。 <a href="#">?hetrf</a> によって決まる、交換操作と $D$ のブロック構造の詳細を格納

する。

$ipiv(i) = k > 0$  の場合、 $d_{ii}$  は  $1 \times 1$  の対角ブロックである。 $A$  の  $i$  番目の行と列は、 $k$  番目の行と列に交換される。

$uplo = 'U'$  で  $ipiv(i) = ipiv(i-1) = -m < 0$  の場合、 $D$  の  $i$  行 ( $i-1$ ) 列に  $2 \times 2$  のブロックが作成される。 $A$  の ( $i-1$ ) 番目の行と列は、 $m$  番目の行と列と交換される。

$uplo = 'L'$  で  $ipiv(i) = ipiv(i+1) = -m < 0$  の場合、 $D$  の  $i$  行 ( $i+1$ ) 列に  $2 \times 2$  のブロックが作成される。 $A$  の ( $i+1$ ) 番目の行と列は、 $m$  番目の行と列と交換される。

**work(1)**  $info = 0$  の場合、終了時に、最適なパフォーマンスを得るために必要な  $lwork$  の最小値が **work(1)** に格納される。これ以降の実行には、この  $lwork$  の値を使用する。

**info** **INTEGER**。  $info = 0$  の場合、実行は正常に終了した。  
 $info = -i$  の場合、 $i$  番目のパラメータの値が不正である。  
 $info = i$  の場合、 $d_{ii}$  は  $0$  である。因子分解は完了したが、 $D$  が完全に特異であるため、解は計算されなかった。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

hesv ルーチンのインターフェイス特有の詳細を以下に示す。

**a**                    サイズ  $(n,n)$  の行列  $A$  を格納する。

**b**                    サイズ  $(n,nrhs)$  の行列  $B$  を格納する。

**ipiv**                長さ  $(n)$  のベクトルを格納する。

**uplo**                'U' または 'L' でなければならない。デフォルト値は 'U' である。

## アプリケーション・ノート

パフォーマンスを改善するには、 $lwork = n * blocksize$  に設定する。 $blocksize$  は、ブロック・アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64) である。

必要なワークスペースの大きさがわからない場合は、最初の実行では  $lwork = -1$  に設定する。この設定は、ワークスペースのクエリとみなされる。ルーチンは、 $work$  配列の最適なサイズだけを計算し、この値を  $work$  配列の最初のエントリ  $work(1)$  として返す。[xerbla](#) は、 $lwork$  に関するエラー・メッセージを生成しない。終了時に、 $work(1)$  の値を確認し、これ以降の実行にはその値を使用する。

## ?hesvx

対角ピボット演算による因子分解を使用して、  
エルミート行列  $A$  を係数行列とする複素連立 1 次  
方程式の解を計算し、解の誤差範囲を示す。

### 構文

#### Fortran 77:

```
call chesvx(FACT, uplo, N, NRHS, A, lda, Af, ldaf, ipiv, B, LDB, X, LDX,
            RCOND, FERR, BERR, WORK, lWORK, rWORK, INFO)
call zhesvx(FACT, uplo, N, NRHS, A, lda, Af, ldaf, ipiv, B, LDB, X, LDX,
            RCOND, FERR, BERR, WORK, lWORK, rWORK, INFO)
```

#### Fortran 95:

```
call hesvx(a, b, x [,uplo] [,af] [,ipiv] [,fact] [,ferr] [,berr] [,rcond]
           [,info])
```

### 説明

このルーチンは、対角ピボット演算による因子分解を使用して、複素連立 1 次方程式  $AX=B$  の解を計算する。 $A$  は  $n \times n$  のエルミート行列、行列  $B$  の列は個々の右辺、 $X$  の列はそれに対応する解である。

このルーチンは、解の誤差範囲と条件数の推定値も示す。

ルーチン ?hesvx は、以下の手順を実行する。

1.  $fact = 'N'$  の場合、対角ピボット演算法を使用して、行列  $A$  を因子分解する。因子分解の形式は、 $A = U D U^H$  または  $A = L D L^H$  である。 $U$  (または  $L$ ) は置換行列と単位上 (下) 三角行列の積で、 $D$  は  $1 \times 1$  と  $2 \times 2$  の対角ブロックを持つエルミート・ブロック対角行列である。

2.  $d_{i,i} = 0$  の場合、つまり  $D$  が完全に特異である場合は、ルーチンは  $info = i$  を返す。それ以外の場合、 $A$  の因子分解された形式を使用して、行列  $A$  の条件数を推定する。条件数の逆数がマシンの精度より小さい場合は、警告として  $info = n + 1$  を返すが、後で説明するようにルーチンは続行され、 $X$  について解を算出し、誤差範囲を計算する。

3.  $A$  の因子分解された形式を使用して、連立方程式を  $X$  について解く。

4. 計算された解の行列の精度を繰り返し改善し、解の誤差範囲と後退誤差の推定値を計算する。

## 入力パラメータ

<i>fact</i>	<p>CHARACTER*1。'F' または 'N' でなければならない。</p> <p>ルーチンの開始時に行列 <i>A</i> の因子分解された形式が与えられるかどうかを指定する。</p> <p><i>fact</i> = 'F' の場合、開始時に、<i>af</i> と <i>ipiv</i> に <i>A</i> の因子分解された形式が格納される。配列 <i>a</i>、<i>af</i>、<i>ipiv</i> は変更されない。</p> <p><i>fact</i> = 'N' の場合、行列 <i>A</i> を <i>af</i> にコピーし、因子分解を実行する。</p>
<i>uplo</i>	<p>CHARACTER*1。'U' または 'L' でなければならない。</p> <p><i>A</i> の上三角部分と下三角部分のどちらを格納するか、また <i>A</i> をどのように因子分解するかを指定する。</p> <p><i>uplo</i> = 'U' の場合、配列 <i>a</i> にはエルミート行列 <i>A</i> の上三角部分が格納され、<i>A</i> は <math>UDU^H</math> として因子分解される。</p> <p><i>uplo</i> = 'L' の場合、配列 <i>a</i> にはエルミート行列 <i>A</i> の下三角部分が格納され、<i>A</i> は <math>LDL^H</math> として因子分解される。</p>
<i>n</i>	INTEGER。行列 <i>A</i> の次数 ( $n \geq 0$ )。
<i>nrhs</i>	INTEGER。右辺の数。 <i>B</i> の列数 ( $nrhs \geq 0$ )。
<i>a</i> , <i>af</i> , <i>b</i> , <i>work</i>	<p>COMPLEX (chesvx の場合)</p> <p>DOUBLE COMPLEX (zhesvx の場合)。</p> <p>配列: <i>a</i>(<i>lda</i>, *), <i>af</i>(<i>ldaf</i>, *), <i>b</i>(<i>ldb</i>, *), <i>work</i>(*)。</p> <p>配列 <i>a</i> には、エルミート行列 <i>A</i> の上三角部分または下三角部分が格納される (<i>uplo</i> を参照)。</p> <p><i>a</i> の第 2 次元は <math>\max(1, n)</math> 以上でなければならない。</p> <p>配列 <i>af</i> は、<i>fact</i> = 'F' の場合は入力引数になる。</p> <p>この配列には、<a href="#">?hetrf</a> による因子分解 <math>A = UDU^H</math> または <math>A = LDL^H</math> で得られた、ブロック対角行列 <i>D</i> と、係数 <i>U</i> または <i>L</i> の計算に使用された乗数が格納される。</p> <p><i>af</i> の第 2 次元は <math>\max(1, n)</math> 以上でなければならない。</p> <p>配列 <i>b</i> には、行列 <i>B</i> が格納される。この行列の列は、連立方程式の右辺である。</p> <p><i>b</i> の第 2 次元は <math>\max(1, nrhs)</math> 以上でなければならない。</p> <p><i>work</i>(*) は、次元 (<i>lwork</i>) のワークスペース配列。</p>
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $lda \geq \max(1, n)$ 。



<i>ldaf</i>	INTEGER。 <i>af</i> の第 1 次元。 $ldaf \geq \max(1, n)$ 。
<i>ldb</i>	INTEGER。 <i>b</i> の第 1 次元。 $ldb \geq \max(1, n)$ 。
<i>ipiv</i>	INTEGER。 配列、次元は $\max(1, n)$ 以上。 配列 <i>ipiv</i> は、 <i>fact</i> = 'F' の場合は入力引数になる。 この配列には、 <a href="#">?hetrf</a> によって決まる、交換操作と <i>D</i> のブロック構造の詳細が格納される。 $ipiv(i) = k > 0$ の場合、 $d_{ii}$ は $1 \times 1$ の対角ブロックである。 <i>A</i> の <i>i</i> 番目の行と列は、 <i>k</i> 番目の行と列に交換される。 $uplo = 'U'$ で $ipiv(i) = ipiv(i-1) = -m < 0$ の場合、 <i>D</i> の <i>i</i> 行 ( <i>i</i> -1) 列に $2 \times 2$ のブロックが作成される。 <i>A</i> の ( <i>i</i> -1) 番目の行と列は、 <i>m</i> 番目の行と列と交換される。 $uplo = 'L'$ で $ipiv(i) = ipiv(i+1) = -m < 0$ の場合、 <i>D</i> の <i>i</i> 行 ( <i>i</i> +1) 列に $2 \times 2$ のブロックが作成される。 <i>A</i> の ( <i>i</i> +1) 番目の行と列は、 <i>m</i> 番目の行と列と交換される。
<i>ldx</i>	INTEGER。 出力配列 <i>x</i> のリーディング・ディメンジョン。 $ldx \geq \max(1, n)$ 。
<i>lwork</i>	INTEGER。 <i>work</i> 配列のサイズ。 <i>lwork</i> = -1 の場合はワークスペースのクエリとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリとして返し、 <i>xerbla</i> は <i>lwork</i> に関するエラー・メッセージを生成しない。 <i>lwork</i> の詳細と推奨値は、以下の「アプリケーション・ノート」を参照。
<i>rwork</i>	REAL ( <i>chesvx</i> の場合 ) DOUBLE PRECISION ( <i>zhesvx</i> の場合 )。 ワークスペース配列、次元は $\max(1, n)$ 以上。

### 出力パラメータ

<i>x</i>	COMPLEX ( <i>chesvx</i> の場合 ) DOUBLE COMPLEX ( <i>zhesvx</i> の場合 )。 配列、次元は ( <i>ldx</i> , *)。  <i>info</i> = 0 または <i>info</i> = <i>n</i> +1 の場合、配列 <i>x</i> には、連立方程式の解の行列 <i>X</i> が格納される。 <i>x</i> の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。
<i>af</i> , <i>ipiv</i>	これらの配列は、 <i>fact</i> = 'N' の場合は出力引数になる。 「入力引数」セクションの <i>af</i> , <i>ipiv</i> の説明を参照。

<i>rcond</i>	REAL (chesvx の場合 ) DOUBLE PRECISION (zhesvx の場合 )。 行列 <i>A</i> の条件数の逆数の推定値。 <i>rcond</i> がマシンの精度より小さい場合 (特に、 <i>rcond</i> = 0 の場合 ) は、行列は有効な精度で特異になる。この条件は、 <i>info</i> > 0 のリターンコードで示される。
<i>ferr</i> , <i>berr</i>	REAL (chesvx の場合 ) DOUBLE PRECISION (zhesvx の場合 )。 配列、次元は max(1, <i>nrhs</i> ) 以上。各解ベクトルの成分ごとの前進誤差と相対後退誤差を格納する。
<i>work</i> (1)	<i>info</i> = 0 の場合、終了時に、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が <i>work</i> (1) に格納される。これ以降の実行には、この <i>lwork</i> の値を使用する。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了した。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正である。 <i>info</i> = <i>i</i> で、 <i>i</i> ≤ <i>n</i> の場合、 <i>d<sub>ii</sub></i> は完全に 0 である。因子分解は完了したが、ブロック対角行列 <i>D</i> が完全に特異であるため、解と誤差範囲は計算できなかった。 <i>rcond</i> = 0 が返される。 <i>info</i> = <i>i</i> で、 <i>i</i> = <i>n</i> + 1 の場合、 <i>D</i> は特異でないが、 <i>rcond</i> がマシンの精度より小さいため、行列は有効な精度で特異になる。しかし、この場合は、解と誤差範囲が計算される。これは、計算された解の精度が、 <i>rcond</i> の値から想定される精度より高くなる場合があるためである。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

hesvx ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	サイズ ( <i>n</i> , <i>n</i> ) の行列 <i>A</i> を格納する。
<i>b</i>	サイズ ( <i>n</i> , <i>nrhs</i> ) の行列 <i>B</i> を格納する。
<i>x</i>	サイズ ( <i>n</i> , <i>nrhs</i> ) の行列 <i>X</i> を格納する。
<i>af</i>	サイズ ( <i>n</i> , <i>n</i> ) の行列 <i>AF</i> を格納する。
<i>ipiv</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>ferr</i>	長さ ( <i>nrhs</i> ) のベクトルを格納する。
<i>berr</i>	長さ ( <i>nrhs</i> ) のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U' である。

*fact* 'N' または 'F' でなければならない。デフォルト値は 'N' である。  
*fact* = 'F' の場合、引数 *af* および *ipiv* の両方を指定しなければならない。指定しない場合は、エラーが返される。

### アプリケーション・ノート

*lwork* の値は、 $2 \times n$  以上でなければならない。パフォーマンスを向上させるには、 $lwork = n \times \text{blocksize}$  に設定する。*blocksize* は、`?hetrf` に最適なブロックサイズである。

必要なワークスペースの大きさがわからない場合は、最初の実行では  $lwork = -1$  に設定する。この設定は、ワークスペースのクエリとみなされる。ルーチンは、*work* 配列の最適なサイズだけを計算し、この値を *work* 配列の最初のエントリ *work*(1) として返す。[xerbla](#) は、*lwork* に関するエラー・メッセージを生成しない。終了時に、*work*(1) の値を確認し、これ以降の実行にはその値を使用する。

## ?spsv

圧縮形式で格納される実対称または複素対称行列 *A* を係数行列とする、複数の右辺を持つ連立 1 次方程式の解を計算する。

### 構文

#### Fortran 77:

```
call sspsv(uplo, n, nrhs, ap, ipiv, b, ldb, info)
call dspsv(uplo, n, nrhs, ap, ipiv, b, ldb, info)
call cspsv(uplo, n, nrhs, ap, ipiv, b, ldb, info)
call zspsv(uplo, n, nrhs, ap, ipiv, b, ldb, info)
```

#### Fortran 95:

```
call spsv(a, b [,uplo] [,ipiv] [,info])
```

### 説明

このルーチンは、実数または複素連立 1 次方程式  $AX=B$  を、*X* について解く。*A* は圧縮形式で格納される  $n \times n$  の対称行列、行列 *B* の列は個々の右辺、*X* の列はそれに対応する解である。

対角ピボット演算法を使用して、 $A$  を  $A = U D U^T$  または  $A = L D L^T$  として因子分解する。 $U$  (または  $L$ ) は置換行列と単位上 (下) 三角行列の積で、 $D$  は  $1 \times 1$  と  $2 \times 2$  の対角ブロックを持つ対称ブロック対角行列である。

次に、 $A$  の因子分解された形式を使用して、連立方程式  $AX = B$  を解く。

## 入力パラメータ

<code>uplo</code>	CHARACTER*1。'U' または 'L' でなければならない。 $A$ の上三角部分と下三角部分のどちらを格納するか、また $A$ をどのように因子分解するかを指定する。 <code>uplo = 'U'</code> の場合、配列 <code>ap</code> には行列 $A$ の上三角部分が格納され、 $A$ は $UDU^T$ として因子分解される。 <code>uplo = 'L'</code> の場合、配列 <code>ap</code> には行列 $A$ の下三角部分が格納され、 $A$ は $LDL^T$ として因子分解される。
<code>n</code>	INTEGER。行列 $A$ の次数 ( $n \geq 0$ )。
<code>nrhs</code>	INTEGER。右辺の数。 $B$ の列数 ( $nrhs \geq 0$ )。
<code>ap, b</code>	REAL (sspsv の場合 ) DOUBLE PRECISION (dspsv の場合 ) COMPLEX (cspsv の場合 ) DOUBLE COMPLEX (zspsv の場合 )。 配列 : <code>ap(*)</code> , <code>b(ldb, *)</code> 。 <code>ap</code> の次元は $\max(1, n(n+1)/2)$ 以上でなければならない。 配列 <code>ap</code> には、 <code>uplo</code> の指定に従って、係数 $U$ または $L$ が圧縮格納形式で格納される (「 <a href="#">行列の格納形式</a> 」を参照)。 配列 <code>b</code> には、行列 $B$ が格納される。この行列の列は、連立方程式の右辺である。 <code>b</code> の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。
<code>ldb</code>	INTEGER。 <code>b</code> の第 1 次元。 $ldb \geq \max(1, n)$ 。

## 出力パラメータ

<code>ap</code>	<a href="#">?spttrf</a> による $A$ の因子分解で得られた、ブロック対角行列 $D$ と、係数 $U$ (または $L$ ) の計算に使用された乗数が、 $A$ と同じ格納形式で、圧縮形式の三角行列として格納される。
<code>b</code>	<code>info = 0</code> の場合、 <code>b</code> は解の行列 $X$ によって上書きされる。
<code>ipiv</code>	INTEGER。 配列、次元は $\max(1, n)$ 以上。 <a href="#">?spttrf</a> によって決まる、交換操作と $D$ のブロック構造の詳細を格納

する。

$ipiv(i) = k > 0$  の場合、 $d_{ii}$  は  $1 \times 1$  のブロックである。A の  $i$  番目の行と列は、 $k$  番目の行と列と交換される。

$uplo = 'U'$  で  $ipiv(i) = ipiv(i-1) = -m < 0$  の場合、 $D$  の  $i$  行 ( $i-1$ ) 列に  $2 \times 2$  のブロックが作成される。A の ( $i-1$ ) 番目の行と列は、 $m$  番目の行と列と交換される。

$uplo = 'L'$  で  $ipiv(i) = ipiv(i+1) = -m < 0$  の場合、 $D$  の  $i$  行 ( $i+1$ ) 列に  $2 \times 2$  のブロックが作成される。A の ( $i+1$ ) 番目の行と列は、 $m$  番目の行と列と交換される。

*info* INTEGER。 *info* = 0 の場合、実行は正常に終了した。  
*info* = - $i$  の場合、 $i$  番目のパラメータの値が不正である。  
*info* =  $i$  の場合、 $d_{ii}$  は 0 である。因子分解は完了したが、 $D$  が完全に特異であるため、解は計算されなかった。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

spsv ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ap</i> を意味する。 サイズ $(n*(n+1)/2)$ の配列 <i>A</i> を格納する。
<i>b</i>	サイズ $(n, nrhs)$ の行列 <i>B</i> を格納する。
<i>ipiv</i>	長さ $(n)$ のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U' である。

## ?spsvx

対角ピボット演算による因子分解を使用して、  
圧縮形式で格納される実対称または複素対称行列  
 $A$  を係数行列とする連立 1 次方程式の解を計算し、  
解の誤差範囲を示す。

---

### 構文

#### Fortran 77:

```
call sspsvx(FACT, uplo, N, NRHS, Ap, Afp, ipiv, B, LDB, X, LDX, RCOND,  
            FERR, BERR, WORK, iWORK, INFO)  
call dspsvx(FACT, uplo, N, NRHS, Ap, Afp, ipiv, B, LDB, X, LDX, RCOND,  
            FERR, BERR, WORK, iWORK, INFO)  
call cspsvx(FACT, uplo, N, NRHS, Ap, Afp, ipiv, B, LDB, X, LDX, RCOND,  
            FERR, BERR, WORK, rWORK, INFO)  
call zspsvx(FACT, uplo, N, NRHS, Ap, Afp, ipiv, B, LDB, X, LDX, RCOND,  
            FERR, BERR, WORK, rWORK, INFO)
```

#### Fortran 95:

```
call spsvx(a, b, x [,uplo] [,af] [,ipiv] [,fact] [,ferr] [,berr] [,rcond]  
            [,info])
```

### 説明

このルーチンは、対角ピボット演算による因子分解を使用して、実数または複素連立 1 次方程式  $AX=B$  の解を計算する。 $A$  は圧縮形式で格納される  $n \times n$  の対称行列、行列  $B$  の列は個々の右辺、 $X$  の列はそれに対応する解である。

このルーチンは、解の誤差範囲と条件数の推定値も示す。

ルーチン ?spsvx は、以下の手順を実行する。

1.  $fact = 'N'$  の場合、対角ピボット演算法を使用して、行列  $A$  を因子分解する。因子分解の形式は、 $A = U D U^T$  または  $A = L D L^T$  である。 $U$  (または  $L$ ) は置換行列と単位上 (下) 三角行列の積で、 $D$  は  $1 \times 1$  と  $2 \times 2$  の対角ブロックを持つ対称ブロック対角行列である。

2.  $d_{i,i} = 0$  の場合、つまり  $D$  が完全に特異である場合は、ルーチンは  $info = i$  を返す。それ以外の場合、 $A$  の因子分解された形式を使用して、行列  $A$  の条件数を推定する。条件数の逆数がマシンの精度より小さい場合は、警告として  $info = n + 1$  を返すが、後で説明するようにルーチンは続行され、 $X$  について解を算出し、誤差範囲を計算する。

3.  $A$  の因子分解された形式を使用して、連立方程式を  $X$  について解く。

4. 計算された解の行列の精度を繰り返し改善し、解の誤差範囲と後退誤差の推定値を計算する。

## 入力パラメータ

*fact* CHARACTER\*1。'F' または 'N' でなければならない。  
ルーチンの開始時に行列  $A$  の因子分解された形式が与えられるかどうかを指定する。  
  
*fact* = 'F' の場合、開始時に、*afp* と *ipiv* に  $A$  の因子分解された形式が格納される。配列 *ap*、*afp*、*ipiv* は変更されない。  
  
*fact* = 'N' の場合、行列  $A$  を *afp* にコピーし、因子分解を実行する。

*uplo* CHARACTER\*1。'U' または 'L' でなければならない。  
 $A$  の上三角部分と下三角部分のどちらを格納するか、また  $A$  をどのように因子分解するかを指定する。  
*uplo* = 'U' の場合、配列 *ap* には対称行列  $A$  の上三角部分が格納され、 $A$  は  $UDU^T$  として因子分解される。  
*uplo* = 'L' の場合、配列 *ap* には対称行列  $A$  の下三角部分が格納され、 $A$  は  $LDL^T$  として因子分解される。

*n* INTEGER。行列  $A$  の次数 ( $n \geq 0$ )。

*nrhs* INTEGER。右辺の数。  $B$  の列数 ( $nrhs \geq 0$ )。

*ap*, *afp*, *b*, *work*  
REAL (sspsvx の場合)  
DOUBLE PRECISION (dspsvx の場合)  
COMPLEX (cspsvx の場合)  
DOUBLE COMPLEX (zspsvx の場合)。  
配列: *ap*(\*), *afp*(\*), *b*(ldb, \*), *work*(\*)。  
  
配列 *ap* には、対称行列  $A$  の上三角部分または下三角部分が圧縮格納形式で格納される(「[行列の格納形式](#)」を参照)。

配列 *afp* は、*fact* = 'F' の場合は入力引数になる。この配列には、[?spturf](#) による因子分解  $A = U D U^T$  または  $A = L D L^T$  で得られた、ブロック対角行列 *D* と、係数 *U* または *L* の計算に使用された乗数が、*A* と同じ格納形式で格納される。

配列 *b* には、行列 *B* が格納される。この行列の列は、連立方程式の右辺である。

*work* (\*) は、ワークスペース配列である。

配列 *ap* と *afp* の次元は  $\max(1, n(n+1)/2)$  以上でなければならない。*b* の第 2 次元は  $\max(1, nrhs)$  以上でなければならない。*work* の次元は  $\max(1, 3*n)$  (実数型の場合) または  $\max(1, 2*n)$  (複素数型の場合) 以上でなければならない。

<i>ldb</i>	INTEGER。 <i>b</i> の第 1 次元。 $ldb \geq \max(1, n)$ 。
<i>ipiv</i>	INTEGER。 配列、次元は $\max(1, n)$ 以上。 配列 <i>ipiv</i> は、 <i>fact</i> = 'F' の場合は入力引数になる。 この配列には、 <a href="#">?spturf</a> によって決まる、交換操作と <i>D</i> のブロック構造の詳細が格納される。 $ipiv(i) = k > 0$ の場合、 $d_{ii}$ は $1 \times 1$ の対角ブロックである。 <i>A</i> の <i>i</i> 番目の行と列は、 <i>k</i> 番目の行と列に交換される。 $uplo = 'U'$ で $ipiv(i) = ipiv(i-1) = -m < 0$ の場合、 <i>D</i> の <i>i</i> 行 ( <i>i</i> -1) 列に $2 \times 2$ のブロックが作成される。 <i>A</i> の ( <i>i</i> -1) 番目の行と列は、 <i>m</i> 番目の行と列と交換される。 $uplo = 'L'$ で $ipiv(i) = ipiv(i+1) = -m < 0$ の場合、 <i>D</i> の <i>i</i> 行 ( <i>i</i> +1) 列に $2 \times 2$ のブロックが作成される。 <i>A</i> の ( <i>i</i> +1) 番目の行と列は、 <i>m</i> 番目の行と列と交換される。
<i>ldx</i>	INTEGER。出力配列 <i>x</i> のリーディング・ディメンジョン。 $ldx \geq \max(1, n)$ 。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, n)$ 以上。実数型でのみ使用される。
<i>rwork</i>	REAL (cspsvx の場合) DOUBLE PRECISION (zspsvx の場合)。 ワークスペース配列、次元は $\max(1, n)$ 以上。複素数型でのみ使用される。



## 出力パラメータ

<i>x</i>	REAL (sspsvx の場合 ) DOUBLE PRECISION (dpsvx の場合 ) COMPLEX (cspvx の場合 ) DOUBLE COMPLEX (zspvx の場合 )。 配列、次元は ( <i>ldx</i> , *)。  <i>info</i> = 0 または <i>info</i> = <i>n</i> + 1 の場合、配列 <i>x</i> には、連立方程式の解の行列 <i>X</i> が格納される。 <i>x</i> の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。
<i>afp</i> , <i>ipiv</i>	これらの配列は、 <i>fact</i> = 'N' の場合は出力引数になる。 「入力引数」セクションの <i>afp</i> , <i>ipiv</i> の説明を参照。
<i>rcond</i>	REAL (単精度の場合 ) DOUBLE PRECISION (倍精度の場合 )。 行列 <i>A</i> の条件数の逆数の推定値。 <i>rcond</i> がマシンの精度より小さい場合 (特に、 <i>rcond</i> = 0 の場合) は、行列は有効な精度で特異になる。この条件は、 <i>info</i> > 0 のリターンコードで示される。
<i>ferr</i> , <i>berr</i>	REAL (単精度の場合 ) DOUBLE PRECISION (倍精度の場合 )。 配列、次元は $\max(1, nrhs)$ 以上。各解ベクトルの成分ごとの前進誤差と相対後退誤差を格納する。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了した。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正である。 <i>info</i> = <i>i</i> で、 <i>i</i> ≤ <i>n</i> の場合、 <i>d<sub>ii</sub></i> は完全に 0 である。因子分解は完了したが、ブロック対角行列 <i>D</i> が完全に特異であるため、解と誤差範囲は計算できなかった。 <i>rcond</i> = 0 が返される。 <i>info</i> = <i>i</i> で、 <i>i</i> = <i>n</i> + 1 の場合、 <i>D</i> は特異でないが、 <i>rcond</i> がマシンの精度より小さいため、行列は有効な精度で特異になる。しかし、この場合は、解と誤差範囲が計算される。これは、計算された解の精度が、 <i>rcond</i> の値から想定される精度より高くなる場合があるためである。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

spsvx ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ap</i> を意味する。 サイズ $(n*(n+1)/2)$ の配列 <i>A</i> を格納する。
<i>b</i>	サイズ $(n,nrhs)$ の行列 <i>B</i> を格納する。
<i>x</i>	サイズ $(n,nrhs)$ の行列 <i>X</i> を格納する。
<i>af</i>	Fortran 77 インターフェイスでの引数 <i>afp</i> を意味する。 サイズ $(n*(n+1)/2)$ の配列 <i>AF</i> を格納する。
<i>ipiv</i>	長さ $(n)$ のベクトルを格納する。
<i>ferr</i>	長さ $(nrhs)$ のベクトルを格納する。
<i>berr</i>	長さ $(nrhs)$ のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U' である。
<i>fact</i>	'N' または 'F' でなければならない。デフォルト値は 'N' である。 <i>fact</i> = 'F' の場合、引数 <i>af</i> および <i>ipiv</i> の両方を指定しなければならない。指定しない場合は、エラーが返される。

---

### ?hpsv

圧縮形式で格納されるエルミート行列 *A* を係数行列とする、複数の右辺を持つ連立 1 次方程式の解を計算する。

---

#### 構文

##### Fortran 77:

```
call chpsv(uplo, n, nrhs, ap, ipiv, b, ldb, info)
call zhpsv(uplo, n, nrhs, ap, ipiv, b, ldb, info)
```

##### Fortran 95:

```
call hpsv(a, b [,uplo] [,ipiv] [,info])
```

#### 説明

このルーチンは、連立 1 次方程式  $AX=B$  を *X* について解く。*A* は圧縮形式で格納される  $n \times n$  のエルミート行列、行列 *B* の列は個々の右辺、*X* の列はそれに対応する解である。

対角ピボット演算法を使用して、 $A$  を  $A = U D U^H$  または  $A = L D L^H$  として因子分解する。 $U$  (または  $L$ ) は置換行列と単位上 (下) 三角行列の積で、 $D$  は  $1 \times 1$  と  $2 \times 2$  の対角ブロックを持つエルミート・ブロック対角行列である。

次に、 $A$  の因子分解された形式を使用して、連立方程式  $AX = B$  を解く。

### 入力パラメータ

<code>uplo</code>	CHARACTER*1。'U' または 'L' でなければならない。 $A$ の上三角部分と下三角部分のどちらを格納するか、また $A$ をどのように因子分解するかを指定する。 <code>uplo = 'U'</code> の場合、配列 <code>ap</code> には行列 $A$ の上三角部分が格納され、 $A$ は $UDU^H$ として因子分解される。 <code>uplo = 'L'</code> の場合、配列 <code>ap</code> には行列 $A$ の下三角部分が格納され、 $A$ は $LDL^H$ として因子分解される。
<code>n</code>	INTEGER。行列 $A$ の次数 ( $n \geq 0$ )。
<code>nrhs</code>	INTEGER。右辺の数。 $B$ の列数 ( $nrhs \geq 0$ )。
<code>ap, b</code>	COMPLEX (chpsv の場合) DOUBLE COMPLEX (zhpsv の場合)。 配列 : <code>ap(*)</code> , <code>b(ldb, *)</code> 。 <code>ap</code> の次元は $\max(1, n(n+1)/2)$ 以上でなければならない。 配列 <code>ap</code> には、 <code>uplo</code> の指定に従って、係数 $U$ または $L$ が圧縮格納形式で格納される (「 <a href="#">行列の格納形式</a> 」を参照)。 配列 <code>b</code> には、行列 $B$ が格納される。この行列の列は、連立方程式の右辺である。 <code>b</code> の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。
<code>ldb</code>	INTEGER。 <code>b</code> の第 1 次元。 $ldb \geq \max(1, n)$ 。

### 出力パラメータ

<code>ap</code>	<a href="#">?hptrf</a> による $A$ の因子分解で得られた、ブロック対角行列 $D$ と、係数 $U$ (または $L$ ) の計算に使用された乗数が、 $A$ と同じ格納形式で、圧縮形式の三角行列として格納される。
<code>b</code>	<code>info = 0</code> の場合、 <code>b</code> は解の行列 $X$ によって上書きされる。
<code>ipiv</code>	INTEGER。 配列、次元は $\max(1, n)$ 以上。 <a href="#">?hptrf</a> によって決まる、交換操作と $D$ のブロック構造の詳細を格納する。 <code>ipiv(i) = k &gt; 0</code> の場合、 $d_{ii}$ は $1 \times 1$ のブロックである。 $A$ の $i$ 番目の行と列は、 $k$ 番目の行と列と交換される。

$uplo = 'U'$  で  $ipiv(i) = ipiv(i-1) = -m < 0$  の場合、 $D$  の  $i$  行 ( $i-1$ ) 列に  $2 \times 2$  のブロックが作成される。 $A$  の ( $i-1$ ) 番目の行と列は、 $m$  番目の行と列と交換される。

$uplo = 'L'$  で  $ipiv(i) = ipiv(i+1) = -m < 0$  の場合、 $D$  の  $i$  行 ( $i+1$ ) 列に  $2 \times 2$  のブロックが作成される。 $A$  の ( $i+1$ ) 番目の行と列は、 $m$  番目の行と列と交換される。

*info* INTEGER。 *info* = 0 の場合、実行は正常に終了した。  
*info* = -*i* の場合、*i* 番目のパラメータの値が不正である。  
*info* = *i* の場合、 $d_{ii}$  は 0 である。因子分解は完了したが、 $D$  が完全に特異であるため、解は計算されなかった。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

hpsv ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ap</i> を意味する。 サイズ $(n*(n+1)/2)$ の配列 $A$ を格納する。
<i>b</i>	サイズ $(n, nrhs)$ の行列 $B$ を格納する。
<i>ipiv</i>	長さ $(n)$ のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U' である。

## ?hpsvx

対角ピボット演算による因子分解を使用して、  
圧縮形式で格納されるエルミート行列  $A$  を係数行列とする連立 1 次方程式の解を計算し、解の誤差範囲を示す。

### 構文

#### Fortran 77:

```
call chpsvx(FACT, uplo, N, NRHS, Ap, Afp, ipiv, B, LDB, X, LDX, RCOND,
            FERR, BERR, WORK, rWORK, INFO)
call zhpsvx(FACT, uplo, N, NRHS, Ap, Afp, ipiv, B, LDB, X, LDX, RCOND,
            FERR, BERR, WORK, rWORK, INFO)
```

#### Fortran 95:

```
call hpsvx(a, b, x [,uplo] [,af] [,ipiv] [,fact] [,ferr] [,berr] [,rcond]
            [,info])
```

### 説明

このルーチンは、対角ピボット演算による因子分解を使用して、複素連立 1 次方程式  $AX=B$  の解を計算する。 $A$  は圧縮形式で格納される  $n \times n$  のエルミート行列、行列  $B$  の列は個々の右辺、 $X$  の列はそれに対応する解である。

このルーチンは、解の誤差範囲と条件数の推定値も示す。

ルーチン ?hpsvx は、以下の手順を実行する。

1.  $fact = 'N'$  の場合、対角ピボット演算法を使用して、行列  $A$  を因子分解する。因子分解の形式は、 $A = U D U^H$  または  $A = L D L^H$  である。 $U$  (または  $L$ ) は置換行列と単位上 (下) 三角行列の積で、 $D$  は  $1 \times 1$  と  $2 \times 2$  の対角ブロックを持つエルミート・ブロック対角行列である。
2.  $d_{i,i} = 0$  の場合、つまり  $D$  が完全に特異である場合は、ルーチンは  $info = i$  を返す。それ以外の場合、 $A$  の因子分解された形式を使用して、行列  $A$  の条件数を推定する。条件数の逆数がマシンの精度より小さい場合は、警告として  $info = n + 1$  を返すが、後で説明するようにルーチンは続行され、 $X$  について解を算出し、誤差範囲を計算する。
3.  $A$  の因子分解された形式を使用して、連立方程式を  $X$  について解く。

4. 計算された解の行列の精度を繰り返し改善し、解の誤差範囲と後退誤差の推定値を計算する。

## 入力パラメータ

<i>fact</i>	CHARACTER*1。'F' または 'N' でなければならない。  ルーチンの開始時に行列 <i>A</i> の因子分解された形式が与えられるかどうかを指定する。  <i>fact</i> = 'F' の場合、開始時に、 <i>afp</i> と <i>ipiv</i> に <i>A</i> の因子分解された形式が格納される。配列 <i>ap</i> 、 <i>afp</i> 、 <i>ipiv</i> は変更されない。  <i>fact</i> = 'N' の場合、行列 <i>A</i> を <i>afp</i> にコピーし、因子分解を実行する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>A</i> の上三角部分と下三角部分のどちらを格納するか、また <i>A</i> をどのように因子分解するかを指定する。 <i>uplo</i> = 'U' の場合、配列 <i>ap</i> にはエルミート行列 <i>A</i> の上三角部分が格納され、 <i>A</i> は $UDU^H$ として因子分解される。 <i>uplo</i> = 'L' の場合、配列 <i>ap</i> にはエルミート行列 <i>A</i> の下三角部分が格納され、 <i>A</i> は $LDL^H$ として因子分解される。
<i>n</i>	INTEGER。行列 <i>A</i> の次数 ( $n \geq 0$ )。
<i>nrhs</i>	INTEGER。右辺の数。 <i>B</i> の列数 ( $nrhs \geq 0$ )。
<i>ap</i> , <i>afp</i> , <i>b</i> , <i>work</i>	COMPLEX (chpsvx の場合) DOUBLE COMPLEX (zhpsvx の場合)。 配列: <i>ap</i> (*), <i>afp</i> (*), <i>b</i> (ldb, *), <i>work</i> (*)。  配列 <i>ap</i> には、エルミート行列 <i>A</i> の上三角部分または下三角部分が圧縮格納形式で格納される (「 <a href="#">行列の格納形式</a> 」を参照)。 配列 <i>afp</i> は、 <i>fact</i> = 'F' の場合は入力引数になる。この配列には、 <a href="#">?hptrf</a> による因子分解 $A = U D U^H$ または $A = L D L^H$ で得られた、ブロック対角行列 <i>D</i> と、係数 <i>U</i> または <i>L</i> の計算に使用された乗数が、 <i>A</i> と同じ格納形式で格納される。 配列 <i>b</i> には、行列 <i>B</i> が格納される。この行列の列は、連立方程式の右辺である。 <i>work</i> (*) は、ワークスペース配列である。  配列 <i>ap</i> と <i>afp</i> の次元は $\max(1, n(n+1)/2)$ 以上でなければならない。 <i>b</i> の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。 <i>work</i> の次元は $\max(1, 2*n)$ 以上でなければならない。
<i>ldb</i>	INTEGER。 <i>b</i> の第 1 次元。 $ldb \geq \max(1, n)$ 。

<i>ipiv</i>	<p>INTEGER。</p> <p>配列、次元は <math>\max(1, n)</math> 以上。</p> <p>配列 <i>ipiv</i> は、<i>fact</i> = 'F' の場合は入力引数になる。</p> <p>この配列には、<a href="#">?hptrf</a> によって決まる、交換操作と <i>D</i> のブロック構造の詳細が格納される。</p> <p><math>ipiv(i) = k &gt; 0</math> の場合、<math>d_{ii}</math> は <math>1 \times 1</math> の対角ブロックである。<i>A</i> の <i>i</i> 番目の行と列は、<i>k</i> 番目の行と列に交換される。</p> <p><i>uplo</i> = 'U' で <math>ipiv(i) = ipiv(i-1) = -m &lt; 0</math> の場合、<i>D</i> の <i>i</i> 行 (<i>i</i>-1) 列に <math>2 \times 2</math> のブロックが作成される。<i>A</i> の (<i>i</i>-1) 番目の行と列は、<i>m</i> 番目の行と列と交換される。</p> <p><i>uplo</i> = 'L' で <math>ipiv(i) = ipiv(i+1) = -m &lt; 0</math> の場合、<i>D</i> の <i>i</i> 行 (<i>i</i>+1) 列に <math>2 \times 2</math> のブロックが作成される。<i>A</i> の (<i>i</i>+1) 番目の行と列は、<i>m</i> 番目の行と列と交換される。</p>
<i>ldx</i>	<p>INTEGER。出力配列 <i>x</i> のリーディング・ディメンジョン。</p> <p><math>ldx \geq \max(1, n)</math>。</p>
<i>rwork</i>	<p>REAL (chpsvx の場合 )</p> <p>DOUBLE PRECISION (zhpsvx の場合 )。</p> <p>ワークスペース配列、次元は <math>\max(1, n)</math> 以上。</p>

### 出力パラメータ

<i>x</i>	<p>COMPLEX (chpsvx の場合 )</p> <p>DOUBLE COMPLEX (zhpsvx の場合 )。</p> <p>配列、次元は (<i>ldx</i>, *)。</p> <p><i>info</i> = 0 または <i>info</i> = <i>n</i>+1 の場合、配列 <i>x</i> には、連立方程式の解の行列 <i>X</i> が格納される。<i>x</i> の第 2 次元は <math>\max(1, nrhs)</math> 以上でなければならない。</p>
<i>afp</i> , <i>ipiv</i>	<p>これらの配列は、<i>fact</i> = 'N' の場合は出力引数になる。</p> <p>「入力引数」セクションの <i>afp</i>, <i>ipiv</i> の説明を参照。</p>
<i>rcond</i>	<p>REAL (chpsvx の場合 )</p> <p>DOUBLE PRECISION (zhpsvx の場合 )。</p> <p>行列 <i>A</i> の条件数の逆数の推定値。<i>rcond</i> がマシンの精度より小さい場合 (特に、<i>rcond</i> = 0 の場合) は、行列は有効な精度で特異になる。この条件は、<i>info</i> &gt; 0 のリターンコードで示される。</p>
<i>ferr</i> , <i>berr</i>	<p>REAL (chpsvx の場合 )</p> <p>DOUBLE PRECISION (zhpsvx の場合 )。</p> <p>配列、次元は <math>\max(1, nrhs)</math> 以上。各解ベクトルの成分ごとの前進誤差と相対後退誤差を格納する。</p>

*info* INTEGER。 *info* = 0 の場合、実行は正常に終了した。  
*info* = -*i* の場合、*i* 番目のパラメータの値が不正である。  
*info* = *i* で、 $i \leq n$  の場合、 $d_{ii}$  は完全に 0 である。因子分解は完了したが、ブロック対角行列 *D* が完全に特異であるため、解と誤差範囲は計算できなかった。*rcond* = 0 が返される。  
*info* = *i* で、 $i = n + 1$  の場合、*D* は特異でないが、*rcond* がマシンの精度より小さいため、行列は有効な精度で特異になる。しかし、この場合は、解と誤差範囲が計算される。これは、計算された解の精度が、*rcond* の値から想定される精度より高くなる場合があるためである。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

hpsvx ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ap</i> を意味する。 サイズ $(n*(n+1)/2)$ の配列 <i>A</i> を格納する。
<i>b</i>	サイズ $(n, nrhs)$ の行列 <i>B</i> を格納する。
<i>x</i>	サイズ $(n, nrhs)$ の行列 <i>X</i> を格納する。
<i>af</i>	Fortran 77 インターフェイスでの引数 <i>ap</i> を意味する。 サイズ $(n*(n+1)/2)$ の配列 <i>AF</i> を格納する。
<i>ipiv</i>	長さ $(n)$ のベクトルを格納する。
<i>ferr</i>	長さ $(nrhs)$ のベクトルを格納する。
<i>berr</i>	長さ $(nrhs)$ のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U' である。
<i>fact</i>	'N' または 'F' でなければならない。デフォルト値は 'N' である。 <i>fact</i> = 'F' の場合、引数 <i>af</i> および <i>ipiv</i> の両方を指定しなければならない。指定しない場合は、エラーが返される。



# LAPACK ルーチン： 最小二乗問題および 固有値問題

## 4

この章では、線形の最小二乗問題、固有値および特異値問題の解決や、それらに関連する一連の計算タスクを実行するための、LAPACK パッケージからインテル<sup>®</sup> マス・カーネル・ライブラリに組み込まれているルーチンについて説明する。

この章の各セクションで、LAPACK の[計算ルーチン](#)と[ドライバルーチン](#)について説明する。LAPACK ルーチンの完全なリファレンスと関連情報は、[\[LUG\]](#) を参照。

**最小二乗問題：**典型的な最小二乗問題とは、行列  $A$  とベクトル  $b$  が与えられたときに、二乗和  $\sum_i ((Ax)_i - b_i)^2$  を最小にするベクトル  $x$  を見つける、または 2- ノルム  $\|Ax - b\|_2$  を最小にするベクトル  $x$  を見つけることである。

通常、 $A$  は  $m \times n$  の行列 ( $m \geq n$ ) で、 $\text{rank}(A) = n$  である。また、この問題は、*優決定* の連立 1 次方程式 (未知数より方程式の個数の方が多い) に対して**最小二乗解**を見つけないこととも言える。この問題を解くには、行列  $A$  の **QR 因子分解** ([QR 因子分解](#)を参照) を使用する。

$m < n$  で  $\text{rank}(A) = m$  の場合、 $Ax = b$  を満たす (つまり、ノルム  $\|Ax - b\|_2$  が最小になる) 解  $x$  が無限に存在する。この場合、 $\|x\|_2$  を最小にする一意な解を見つけない方が有用である。この問題は、*劣決定* の連立 1 次方程式 (方程式より未知数の個数の方が多い) に対して**最小ノルム解**を見つけないこととも言える。この問題を解くには、行列  $A$  の **LQ 因子分解** ([LQ 因子分解](#)を参照) を使用する。

一般には、 $\text{rank}(A) < \min(m, n)$  となり、*階数不足の最小二乗問題*になるときがあるので、 $\|x\|_2$  も  $\|Ax - b\|_2$  も最小にする**最小ノルム最小二乗解**を見つけない必要がある。その場合 (または  $A$  の階数が不明な場合) は、**QR 因子分解**と、ピボット演算または**特異値分解** ([特異値分解](#)を参照) を併用する。

**固有値問題：**固有値 (eigenvalue) 問題 (*eigen* は、" 固有 " の意味のドイツ語) は、「行列  $A$  が与えられたときに、下記のいずれかの方程式を満たす**固有値** $\lambda$  と、それに対応する**固有ベクトル** $z$ を見つけないこと」である。

$$Az = \lambda z \text{ (右固有ベクトル } z)$$

または

$$z^H A = \lambda z^H \text{ (左固有ベクトル } z)$$

$A$  が実対称 / 複素エルミート行列の場合、上記の 2 つの方程式が等価になり、上記の問題は対称固有値問題と呼ばれる。このタイプの問題を解決するためのルーチンについては、この章の[対称固有値問題](#)のセクションで説明する。

非対称 / 非エルミート行列を使って固有値問題を解決するためのルーチンについては、この章の[非対称固有値問題](#)のセクションで説明する。

本ライブラリには、汎用対称固有値問題( 次のいずれかの方程式を満たす固有値  $\lambda$  と、それに対応する固有ベクトル  $z$  を見つけること ) を処理するためのルーチンも含まれている。

$$Az = \lambda Bz, ABz = \lambda z, \text{ または } BAz = \lambda z$$

$A$  は、対称 / エルミート行列である。 $B$  は、対称 / エルミート正定値行列である。これらの問題を標準的な対称固有値問題にするためのルーチンについては、この章の[汎用対称固有値問題](#)のセクションで説明する。

個々の問題を解決するには、通常、いくつかの計算ルーチンを呼び出す。本章のルーチンを、第 3 章で説明している他の LAPACK ルーチンや第 2 章で説明している BLAS ルーチンと組み合わせて使用しなければならない場合もある。

例えば、与えられた行列  $B$  のすべての列  $b$  に対して  $\|Ax - b\|_2$  が最小になる一連の最小二乗問題 ( $A$  も  $B$  も実行列) を解くには、`?geqrf` を呼び出して  $A = QR$  の因子分解を行った後、`?ormqr` を呼び出して  $C = Q^H B$  を求める。最後に、BLAS ルーチンの `?trsm` を呼び出して、連立方程式  $RX = C$  を  $X$  について解く。

あるいは、適切なドライバルーチンを使用すれば、複数のタスクを一度に実行できる。例えば、最小二乗問題を解く場合は、ドライバルーチン `?gels` を使用する。



**警告:** LAPACK ルーチンは、入力行列が INF 値または NaN 値を格納していないことを前提としている。LAPACK に対する入力データが適当でないとき、コンピュータのハングアップなどの問題が発生する可能性がある。

インテル MKL 8.0 以降では、LAPACK 計算ルーチンおよびドライバルーチンに対する Fortran-77 インターフェイスとともに、より短い引数リストで簡略化したルーチン呼び出しを使用する Fortran-95 インターフェイスもサポートしている。Fortran-95 インターフェイスの呼び出しシーケンスは、ルーチン説明の「構文」セクションで、Fortran-77 呼び出しの説明の後に解説する。

## ルーチン命名規則

この章の各ルーチンについて、Fortran-77 プログラムからの呼び出し時に LAPACK 名を使用できる。

**LAPACK** 名は、以下に示すように、`xyyzzz` 構造になっている。

最初の文字 `x` は、データ型を示す。

<code>s</code>	実数、単精度	<code>c</code>	複素数、単精度
<code>d</code>	実数、倍精度	<code>z</code>	複素数、倍精度

2 番目と 3 番目の文字 `yy` は、行列のタイプと格納形式を示す。

<code>bd</code>	二重対角行列
<code>ge</code>	一般行列
<code>gb</code>	一般帯行列
<code>hs</code>	上 <b>Hessenberg</b> 行列
<code>or</code>	(実数) 直交行列
<code>op</code>	(実数) 直交行列 (圧縮格納形式)
<code>un</code>	(複素数) ユニタリ行列
<code>up</code>	(複素数) ユニタリ行列 (圧縮格納形式)
<code>pt</code>	対称 / エルミート 正定値 三重対角行列
<code>sy</code>	対称行列
<code>sp</code>	対称行列 (圧縮格納形式)
<code>sb</code>	(実数) 対称帯行列
<code>st</code>	(実数) 対称 三重対角行列
<code>he</code>	エルミート行列
<code>hp</code>	エルミート行列 (圧縮格納形式)
<code>hb</code>	(複素数) エルミート帯行列
<code>tr</code>	三角または準三角行列

最後の 3 文字 `zzz` は、実行される処理を示す。

<code>qrf</code>	<i>QR</i> 因子分解を行う。
<code>lqf</code>	<i>LQ</i> 因子分解を行う。

例えば、ルーチン `sgeqrf` は、単精度の一般実行列の *QR* 因子分解を行う。これに対応する複素行列用のルーチンは、`cgeqrf` である。

インテル MKL において、Fortran-95 インターフェイスの LAPACK 計算ルーチン名およびドライバルーチン名は、最初の文字がデータ型を示す以外は Fortran-77 の名前と同じである。例えば、Fortran-95 インターフェイスで、一般実行列の *QR* 因子分解を行うルーチン名は `geqrf` である。異なるデータ型を扱うには、単精度および倍精度の名前付き定数でモジュールブロックを参照する特定の入力パラメータを定義して行う。

インテル MKL に含まれている LAPACK 計算ルーチンとドライバルーチン用の Fortran-95 インターフェイスの詳細、およびオプションの引数の使用についての一般的な情報は、第 3 章の [Fortran-95 インターフェイス規則](#) を参照。

### 行列の格納形式

LAPACK ルーチンでは、以下の行列格納形式を使用している。

- フル格納では、行列  $A$  は 2 次元配列  $a$  に格納され、行列成分  $a_{ij}$  は配列成分  $a(i, j)$  に格納される。
- 圧縮格納では、対称行列、エルミート行列、または三角行列をコンパクトに格納できる。行列の上三角または下三角が 1 次元配列の各列に圧縮される。
- 帯格納では、 $k_l$  個の劣対角成分と  $k_u$  個の優対角成分を持つ  $m \times n$  の帯行列は、 $(k_l + k_u + 1)$  行  $n$  列の 2 次元配列  $ab$  にコンパクトに格納される。行列の各列は配列の対応する列に格納され、行列の各対角成分は配列の各行に格納される。

第 3 章と第 4 章では、圧縮格納形式で行列を格納する配列は名前の最後の文字を  $p$  で、帯格納形式で行列を格納する配列には名前の最後の文字を  $b$  で示している。行列の格納形式の詳細は、付録 B の「[行列引数](#)」を参照。

### 数学的表記

これまでの章で使用した数学表記以外に、本章では以下に示す表記も使用している。

$\lambda_i$	行列 $A$ の固有値 (固有値の定義は、 <a href="#">固有値問題</a> を参照)。
$\sigma_i$	行列 $A$ の特異値。 $A^H A$ の固有値の平方根に等しい (詳細は、 <a href="#">特異値分解</a> を参照)。
$\ x\ _2$	ベクトル $x$ の 2- ノルム $x$ : $\ x\ _2 = (\sum_i  x_i ^2)^{1/2} = \ x\ _E$
$\ A\ _2$	行列 $A$ の 2- ノルム (またはスペクトル・ノルム)。 $\ A\ _2 = \max_i \sigma_i$ , $\ A\ _2^2 = \max_{ x =1} (Ax, Ax)$
$\ A\ _E$	行列 $A$ のユークリッド・ノルム。 $\ A\ _E^2 = \sum_i \sum_j  a_{ij} ^2$ (ベクトルの場合、ユークリッド・ノルムと 2- ノルムが等しくなるので、 $\ x\ _E = \ x\ _2$ )。
$q(x, y)$	ベクトル $x$ と $y$ の間の鋭角。 $\cos q(x, y) =  xy  / (\ x\ _2 \ y\ _2)$ 。

## 計算ルーチン

以下の各セクションでは、LAPACK 計算ルーチンについて説明する。LAPACK 計算ルーチンは、次の目的の計算タスクを個別に実行する。

[直交因子分解](#)

[特異値分解](#)

[対称固有値問題](#)

[汎用対称固有値問題](#)

[非対称固有値問題](#)

[汎用非対称固有値問題](#)

[汎用特異値分解](#)

各[ドライバルーチン](#)も参照のこと。

## 直交因子分解

このセクションでは、行列の  $QR$  ( $RQ$ ) 因子分解と  $LQ$  ( $QL$ ) 因子分解を行うための LAPACK ルーチンについて説明する。一般化された  $QR$  因子分解と  $RQ$  因子分解と同様に  $RZ$  因子分解用のルーチンも含まれている。

**QR 因子分解：**  $A$  を、因子分解する  $m \times n$  の行列とする。

$m \geq n$  の場合、 $QR$  因子分解は次の式で与えられる。 $A = Q \begin{pmatrix} R \\ 0 \end{pmatrix} = (Q_1, Q_2) \begin{pmatrix} R \\ 0 \end{pmatrix}$

$R$  は、実数型の対角成分を持つ  $n \times n$  の上三角行列である。 $Q$  は、 $m \times m$  の直交 (またはユニタリ) 行列である。

$QR$  因子分解を用いると、「 $A$  が  $m \times n$  の最大階数の行列 ( $m \geq n$ ) である場合に、 $\|Ax - b\|_2$  を最小の値にする」という最小二乗問題が解ける。該当する行列の因子分解を行ってから、 $Rx = (Q_1)^T b$  を解いて、解  $x$  を求める。

$m < n$  の場合、 $QR$  因子分解は次の式で与えられる。

$$A = QR = Q(R_1 R_2)$$

$R$  は台形行列、 $R_1$  は上三角行列、 $R_2$  は矩形行列である。

LAPACK ルーチンでは、行列  $Q$  を明示的な形式で表現しない。代わりに、 $Q$  は、 $\min(m, n)$  個の基本リフレクタの積として表現される。一連のルーチンでは、この形式で表現される  $Q$  を操作する。

**LQ 因子分解** : LQ 因子分解  $A$  を  $m \times n$  の行列とすると、 $m \leq n$  の場合、次の式で与えられる。

$$A = (L, 0)Q = (L, 0) \begin{pmatrix} Q_1 \\ Q_2 \end{pmatrix} = LQ_1$$

$L$  は、実数型の対角成分を持つ  $m \times m$  の下三角行列である。 $Q$  は、 $n \times n$  の直交 (またはユニタリ) 行列である。

$m > n$  の場合、 $LQ$  因子分解は次の式で与えられる。

$$A = \begin{pmatrix} L_1 \\ L_2 \end{pmatrix} Q$$

$L_1$  は  $n \times n$  の下三角行列、 $L_2$  は矩形行列、 $Q$  は  $n \times n$  の直交 (またはユニタリ) 行列である。

$LQ$  因子分解を用いると、劣決定の連立 1 次方程式  $Ax = b$  ( $A$  は、階数  $m$  ( $m < n$ ) の  $m \times n$  の行列) の最小ノルム解を求めることができる。該当する行列の因子分解を行ってから、 $Ly = b$  を  $y$  について解いた後、 $x = (Q_1)^H y$  を計算し、解ベクトル  $x$  を求める。

[表 4-1](#) に、行列の直交因子分解を行うための LAPACK ルーチン (Fortran-77 インターフェイス) を示す。Fortran-95 インターフェイスのルーチン名はそれぞれ、先頭の記号がない ([ルーチン命名規則](#) を参照)。

**表 4-1 直交因子分解用の計算ルーチン**

行列のタイプ、 因子分解	因子分解 (ピボット演算なし)	因子分解 (ピボット演算あり)	行列 Q の生成	行列 Q の適用
一般行列、 QR 因子分解	<a href="#">?geqrf</a>	<a href="#">?geqpf</a> <a href="#">?geqp3</a>	<a href="#">?orgqr</a> <a href="#">?ungqr</a>	<a href="#">?ormqr</a> <a href="#">?unmqr</a>
一般行列、 RQ 因子分解	<a href="#">?qerqf</a>		<a href="#">?orgrq</a> <a href="#">?ungrq</a>	<a href="#">?ormrq</a> <a href="#">?unmrq</a>
一般行列、 LQ 因子分解	<a href="#">?gelqf</a>		<a href="#">?orglq</a> <a href="#">?unglq</a>	<a href="#">?ormlq</a> <a href="#">?unmlq</a>
一般行列、 QL 因子分解	<a href="#">?qeqlf</a>		<a href="#">?orgql</a> <a href="#">?ungql</a>	<a href="#">?ormql</a> <a href="#">?unmql</a>
台形行列、 RZ 因子分解	<a href="#">?tzzrf</a>			<a href="#">?ormrz</a> <a href="#">?unmrz</a>
行列のペア、 汎用 QR 因子分解	<a href="#">?ggqrf</a>			
行列のペア、 汎用 RQ 因子分解	<a href="#">?qgrqf</a>			

## ?geqrf

$m \times n$  の一般行列の  $QR$  因子分解を行う。

### 構文

#### Fortran 77:

```
call sgeqrf(m, n, a, lda, tau, work, lwork, info)
call dgeqrf(m, n, a, lda, tau, work, lwork, info)
call cgeqrf(m, n, a, lda, tau, work, lwork, info)
call zgeqrf(m, n, a, lda, tau, work, lwork, info)
```

#### Fortran 95:

```
call geqrf(a [,tau] [,info])
```

### 説明

このルーチンは、 $m \times n$  の一般行列  $A$  の  $QR$  因子分解を行う ([直交因子分解](#)を参照)。ピボット演算は行わない。

このルーチンでは、行列  $Q$  を明示的な形式で表現しない。代わりに、 $Q$  は、 $\min(m, n)$  個の基本リフレクタの積として表現される。一連のルーチンでは、この形式で表現される  $Q$  を操作する。

### 入力パラメータ

$m$	INTEGER。行列 $A$ の行数 ( $m \geq 0$ )。
$n$	INTEGER。行列 $A$ の列数 ( $n \geq 0$ )。
$a, work$	REAL (sgeqrf の場合 ) DOUBLE PRECISION (dgeqrf の場合 ) COMPLEX (cgeqrf の場合 ) DOUBLE COMPLEX (zgeqrf の場合 )。 配列： $a(lda, *)$ には、行列 $A$ を格納する。 $a$ の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $work(lwork)$ は、ワークスペース配列である。
$lda$	INTEGER。 $a$ の第 1 次元。 $\max(1, m)$ 以上。

*lwork* INTEGER。配列 *work* のサイズ ( $lwork \geq n$ )。  
*lwork* = -1 の場合、ワークスペースのクエリとみなされ、ルーチンは *work* 配列の最適サイズだけを計算し、その値を *work* 配列の最初のエントリとして返す。[xerbla](#) は *lwork* に関するエラー・メッセージを生成しない。  
*lwork* の推奨値は、「[アプリケーション・ノート](#)」を参照。

### 出力パラメータ

*a* 因子分解のデータによって、次のように上書きされる。  
 $m \geq n$  の場合、対角線より下の成分は、ユニタリ行列  $Q$  の各成分によって上書きされる。上三角は、上三角行列  $R$  の対応する成分によって上書きされる。  
 $m < n$  の場合、厳密な下三角部分は、ユニタリ行列  $Q$  の各成分によって上書きされる。残りの成分は、 $m \times n$  の上台形行列  $R$  の対応する成分によって上書きされる。

*tau* REAL (sgeqrf の場合)  
DOUBLE PRECISION (dgeqrf の場合)  
COMPLEX (cgeqrf の場合)  
DOUBLE COMPLEX (zgeqrf の場合)。  
配列、次元は  $\max(1, \min(m, n))$  以上。  
行列  $Q$  に関するその他の情報も格納される。

*work(1)* *info* = 0 の場合、終了時に、最適なパフォーマンスを得るために必要な *lwork* の最小値が *work(1)* に格納される。以降の実行では、この *lwork* 値を使用する。

*info* INTEGER。  
*info* = 0 の場合、実行は正常に終了したことを示す。  
*info* = -*i* の場合、*i* 番目のパラメータの値が不正であったことを示す。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *geqrf* のインターフェイスの詳細を以下に示す。

*a* サイズ ( $m, n$ ) の行列  $A$  を格納する。



`tau`                      長さ  $\min(m, n)$  のベクトルを格納する。

### アプリケーション・ノート

パフォーマンスを改善するには、 $lwork = n * blocksize$  に設定する。 $blocksize$  は、ブロック・アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。

必要なワークスペースの大きさがわからない場合は、最初の実行では  $lwork$  を大きめの値に設定する。終了時に、`work(1)` の値を調べ、以降はその値を使用する。

計算で求めた因子分解は、行列  $A + E$  ( $\|E\|_2 = O(\epsilon) \|A\|_2$ ) を因子分解したそのものである。

実数型の浮動小数点演算のおおよその回数は次のとおりである。

$$(4/3)n^3 \quad (m = n \text{ の場合}),$$

$$(2/3)n^2(3m - n) \quad (m > n \text{ の場合}),$$

$$(2/3)m^2(3n - m) \quad (m < n \text{ の場合}).$$

複素数型の演算回数は、この 4 倍になる。

与えられた行列  $B$  のすべての列  $b$  に対して  $\|Ax - b\|_2$  が最小にする一連の最小二乗問題を解くには、以下の手順でルーチン呼び出す。

`?geqrf` (このルーチン)                      因子分解  $A = QR$  を行う。

[`?ormqr`](#)                                       $C = Q^T B$  を計算する (実行列の場合)。

[`?unmqr`](#)                                       $C = Q^H B$  を計算する (複素行列の場合)。

[`?trsm`](#) (BLAS ルーチン)                       $RX = C$  を解く。

(計算で求めた  $X$  の列は、最小二乗の解ベクトル  $x$  である。)

$Q$  の成分を明示的に計算するには、次のルーチン呼び出す。

[`?orgqr`](#)                                      (実行列の場合)

[`?ungqr`](#)                                      (複素行列の場合)

### ?geqpf

$m \times n$  の一般行列の  $QR$  因子分解をピボット演算を用いて行う。

---

#### 構文

##### Fortran 77:

```
call sgeqpf(m, n, a, lda, jpvt, tau, work, info)
call dgeqpf(m, n, a, lda, jpvt, tau, work, info)
call cgeqpf(m, n, a, lda, jpvt, tau, work, rwork, info)
call zgeqpf(m, n, a, lda, jpvt, tau, work, rwork, info)
```

##### Fortran 95:

```
call geqpf(a, jpvt [,tau] [,info])
```

#### 説明

このルーチンは [?geqp3](#) に置き換えられている。

このルーチン ?geqpf は、 $m \times n$  の一般行列  $A$  の  $QR$  因子分解  $AP = QR$  を列ピボット演算を用いて行う ([直交因子分解](#)を参照)。  $P$  は、 $n \times n$  の置換行列である。

このルーチンでは、行列  $Q$  を明示的な形式で表現しない。代わりに、 $Q$  は、 $\min(m, n)$  個の基本リフレクタの積として表現される。一連のルーチンでは、この形式で表現される  $Q$  を操作する。

#### 入力パラメータ

$m$	INTEGER。行列 $A$ の行数 ( $m \geq 0$ )。
$n$	INTEGER。行列 $A$ の列数 ( $n \geq 0$ )。
$a, work$	REAL (sgeqpf の場合) DOUBLE PRECISION (dgeqpf の場合) COMPLEX (cgeqpf の場合) DOUBLE COMPLEX (zgeqpf の場合)。 配列: $a(lda,*)$ には、行列 $A$ を格納する。 $a$ の第 2 次元は、 $\max(1, n)$ 以上でなければならない。

	$work(lwork)$ は、ワークスペース配列である。
$lda$	INTEGER。 $a$ の第 1 次元。 $\max(1, m)$ 以上。
$lwork$	INTEGER。 配列 $work$ のサイズ。 $\max(1, 3*n)$ 以上でなければならない。
$jpvt$	INTEGER。 配列、次元は $\max(1, n)$ 以上。 呼び出し時に、 $jpvt(i) > 0$ の場合、計算の開始前に、 $A$ の $i$ 番目の列が $AP$ の先頭に移され、計算中はその位置に保持される。 $jpvt(i) = 0$ の場合、 $A$ の $i$ 番目の列はフリー列なので、計算中に他のフリー列と交換される場合がある。
$rwork$	REAL (cgeqpf の場合 ) DOUBLE PRECISION (zgeqpf の場合 )。 ワークスペース配列、次元は $\max(1, 2*n)$ 以上。

### 出力パラメータ

$a$	因子分解のデータによって、次のように上書きされる。  $m \geq n$ の場合、対角線より下の成分は、ユニタリ (直交) 行列 $Q$ の各成分によって上書きされる。上三角は、上三角行列 $R$ の対応する成分によって上書きされる。  $m < n$ の場合、厳密な下三角部分は、行列 $Q$ の各成分によって上書きされる。残りの成分は、 $m \times n$ の上台形行列 $R$ の対応する成分によって上書きされる。
$tau$	REAL (sgeqpf の場合 ) DOUBLE PRECISION (dgeqpf の場合 ) COMPLEX (cgeqpf の場合 ) DOUBLE COMPLEX (zgeqpf の場合 )。 配列、次元は $\max(1, \min(m, n))$ 以上。 行列 $Q$ に関するその他の情報も格納される。
$jpvt$	因子分解 $AP = QR$ における置換行列 $P$ の各成分によって上書きされる。すなわち、 $AP$ の列は、 $A$ の列を次の順に並べたものになる。 $jpvt(1), jpvt(2), \dots, jpvt(n)$ 。
$info$	INTEGER。 $info = 0$ の場合、実行は正常に終了したことを示す。 $info = -i$ の場合、 $i$ 番目のパラメータの値が不正であったことを示す。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `geqpf` のインターフェイスの詳細を以下に示す。

`a`                      サイズ  $(m, n)$  の行列  $A$  を格納する。  
`jpvt`                    長さ  $(n)$  のベクトルを格納する。  
`tau`                     長さ  $\min(m, n)$  のベクトルを格納する。

## アプリケーション・ノート

計算で求めた因子分解は、行列  $A + E$  ( $\|E\|_2 = O(\epsilon) \|A\|_2$ ) を因子分解したそのものである。

実数型の浮動小数点演算のおおよその回数は次のとおりである。

$(4/3)n^3$                       ( $m = n$  の場合)、  
 $(2/3)n^2(3m - n)$             ( $m > n$  の場合)、  
 $(2/3)m^2(3n - m)$             ( $m < n$  の場合)。

複素数型の演算回数は、この 4 倍になる。

与えられた行列  $B$  のすべての列  $b$  に対して  $\|Ax - b\|_2$  が最小にする一連の最小二乗問題を解くには、以下の手順でルーチンを呼び出す。

`?geqpf` (このルーチン)                      因子分解  $AP = QR$  を行う。  
[?ormqr](#)     $C = Q^T B$  を計算する (実行列の場合)。  
[?unmqr](#)     $C = Q^H B$  を計算する (複素行列の場合)。  
[?trsm](#) (BLAS ルーチン)                       $RX = C$  を解く。

(計算で求めた  $X$  の列は、最小二乗解ベクトル  $x$  が置換されたものである。置換順序は、配列 `jpvt` に出力される。)

$Q$  の成分を明示的に計算するには、次のルーチンを呼び出す。

[?orgqr](#)    (実行列の場合)  
[?ungqr](#)    (複素行列の場合)

## ?geqp3

レベル 3 BLAS を使用して、 $m \times n$  の一般行列の  $QR$  因子分解をピボット演算を用いて行う。

### 構文

#### Fortran 77:

```
call sgeqp3(m, n, a, lda, jpvt, tau, work, lwork, info)
call dgeqp3(m, n, a, lda, jpvt, tau, work, lwork, info)
call cgeqp3(m, n, a, lda, jpvt, tau, work, lwork, rwork, info)
call zgeqp3(m, n, a, lda, jpvt, tau, work, lwork, rwork, info)
```

#### Fortran 95:

```
call gep3(a, jpvt [,tau] [,info])
```

### 説明

このルーチンは、レベル 3 BLAS を使用して、 $m \times n$  の一般行列  $A$  の  $QR$  因子分解 ( $AP = QR$ 、[直交因子分解](#)を参照) をピボット演算を用いて行う。 $P$  は、 $n \times n$  の置換行列である。このルーチンを使用すると、?geqpf より高いパフォーマンスが得られる。

このルーチンでは、行列  $Q$  を明示的な形式で表現しない。代わりに、 $Q$  は、 $\min(m, n)$  個の基本リフレクタの積として表現される。一連のルーチンでは、この形式で表現される  $Q$  を操作する。

### 入力パラメータ

$m$	INTEGER。行列 $A$ の行数 ( $m \geq 0$ )。
$n$	INTEGER。行列 $A$ の列数 ( $n \geq 0$ )。
$a, work$	REAL (sgeqp3 の場合 ) DOUBLE PRECISION (dgeqp3 の場合 ) COMPLEX (cgeqp3 の場合 ) DOUBLE COMPLEX (zgeqp3 の場合 )。 配列： $a(lda,*)$ には、行列 $A$ を格納する。 $a$ の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $work(lwork)$ は、ワークスペース配列である。

<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $\max(1, m)$ 以上。
<i>lwork</i>	INTEGER。 配列 <i>work</i> のサイズ。 $\max(1, 3*n+1)$ 以上 (実数型の場合) または $\max(1, n+1)$ 以上 (複素数型の場合) でなければならない。  <i>lwork</i> = -1 の場合、ワークスペースのクエリとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリとして返す。 <i>xerbla</i> は <i>lwork</i> に関するエラー・メッセージを生成しない。
<i>jpvt</i>	INTEGER。 配列、次元は $\max(1, n)$ 以上。  呼び出し時に、 <i>jpvt</i> ( <i>i</i> ) $\neq 0$ の場合、計算の開始前に、 <i>A</i> の <i>i</i> 番目の列が <i>AP</i> の先頭に移され、計算中はその位置に保持される。 <i>jpvt</i> ( <i>i</i> ) = 0 の場合、 <i>A</i> の <i>i</i> 番目の列はフリー列なので、計算中に他のフリー列と交換される場合がある。
<i>rwork</i>	REAL ( <i>cgeqp3</i> の場合) DOUBLE PRECISION ( <i>zgeqp3</i> の場合)。 ワークスペース配列、次元は $\max(1, 2*n)$ 以上。複素数型でのみ使用される。

## 出力パラメータ

<i>a</i>	因子分解のデータによって、次のように上書きされる。  $m \geq n$ の場合、対角線より下の成分は、ユニタリ (直交) 行列 <i>Q</i> の各成分によって上書きされる。上三角は、上三角行列 <i>R</i> の対応する成分によって上書きされる。  $m < n$ の場合、厳密な下三角部分は、行列 <i>Q</i> の各成分によって上書きされる。残りの成分は、 $m \times n$ の上台形行列 <i>R</i> の対応する成分によって上書きされる。
<i>tau</i>	REAL ( <i>sgeqp3</i> の場合) DOUBLE PRECISION ( <i>dgeqp3</i> の場合) COMPLEX ( <i>cgeqp3</i> の場合) DOUBLE COMPLEX ( <i>zgeqp3</i> の場合)。 配列、次元は $\max(1, \min(m, n))$ 以上。 行列 <i>Q</i> に対する基本リフレクタのスカラ係数が格納される。
<i>jpvt</i>	因子分解 $AP = QR$ における置換行列 <i>P</i> の各成分によって上書きされる。すなわち、 <i>AP</i> の列は、 <i>A</i> の列を次の順に並べたものになる。 <i>jpvt</i> (1), <i>jpvt</i> (2), ..., <i>jpvt</i> ( <i>n</i> )。

*info*                    INTEGER。  
                         *info* = 0 の場合、実行は正常に終了したことを示す。  
                         *info* = -*i* の場合、*i* 番目のパラメータの値が不正であったことを示す。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `geqp3` のインターフェイスの詳細を以下に示す。

*a*                        サイズ (*m*,*n*) の行列 *A* を格納する。  
*jpvt*                    長さ (*n*) のベクトルを格納する。  
*tau*                     長さ  $\min(m, n)$  のベクトルを格納する。

### アプリケーション・ノート

与えられた行列 *B* のすべての列 *b* に対して  $\|Ax - b\|_2$  が最小にする一連の最小二乗問題を解くには、以下の手順でルーチンを呼び出す。

`?geqp3` ( このルーチン )                    因子分解  $AP = QR$  を行う。  
[?ormqr](#)                                         $C = Q^T B$  を計算する ( 実行列の場合 )。  
[?unmqr](#)                                         $C = Q^H B$  を計算する ( 複素行列の場合 )。  
[?trsm](#) (BLAS ルーチン)                     $RX = C$  を解く。

( 計算で求めた *X* の列は、最小二乗解ベクトル *x* が置換されたものである。置換順序は、配列 *jpvt* に出力される。 )

*Q* の成分を明示的に計算するには、次のルーチンを呼び出す。

[?orgqr](#)                                        ( 実行列の場合 )  
[?ungqr](#)                                        ( 複素行列の場合 )

### ?orgqr

?geqrf で求めた  $QR$  因子分解の実直交行列  $Q$  を生成する。

---

#### 構文

##### Fortran 77:

```
call sorgqr(m, n, k, a, lda, tau, work, lwork, info)
call dorgqr(m, n, k, a, lda, tau, work, lwork, info)
```

##### Fortran 95:

```
call orgqr(a, tau [,info])
```

#### 説明

このルーチンは、ルーチン [sgeqrf/dgeqrf](#) または [sgeqpf/dgeqpf](#) によって求めた  $QR$  因子分解の直交行列  $Q$  ( $m \times m$ ) の全部または一部を生成する。このルーチンは、[sgeqrf/dgeqrf](#) または [sgeqpf/dgeqpf](#) を呼び出した後で使用する。

$Q$  は、通常、 $m \times p$  の行列  $A$  ( $m \geq p$ ) の  $QR$  因子分解によって決定される。行列  $Q$  の全体を計算するには、次のように呼び出す。

```
call ?orgqr(m, m, p, a, lda, tau, work, lwork, info)
```

$Q$  の先頭から  $p$  個の列 ( $A$  の列で張られた空間内で直交基を形成) を計算するには、次のように呼び出す。

```
call ?orgqr(m, p, p, a, lda, tau, work, lwork, info)
```

$A$  の先頭から  $k$  個の列に関して  $QR$  因子分解の行列  $Q^k$  を求めるには、次のように呼び出す。

```
call ?orgqr(m, m, k, a, lda, tau, work, lwork, info)
```

$Q^k$  の先頭から  $k$  個の列 ( $A$  の先頭から  $k$  個の列で張られた空間内で直交基を形成) を求めるには、次のように呼び出す。

```
call ?orgqr(m, k, k, a, lda, tau, work, lwork, info)
```

#### 入力パラメータ

$m$                     INTEGER。直交行列  $Q$  の次数 ( $m \geq 0$ )。

$n$                     INTEGER。計算する  $Q$  の列数 ( $0 \leq n \leq m$ )。



$k$	INTEGER。その積が行列 $Q$ となる基本リフレクタの個数 ( $0 \leq k \leq n$ )。
$a, \tau, work$	REAL (sorgqr の場合) DOUBLE PRECISION (dorgqr の場合)。 配列： $a(lda,*)$ と $\tau(*)$ は、sgeqrf / dgeqrf または sgeqpf / dgeqpf から返された配列である。 $a$ の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $\tau$ の次元は、 $\max(1, k)$ 以上でなければならない。  $work(lwork)$ は、ワークスペース配列である。
$lda$	INTEGER。 $a$ の第 1 次元。 $\max(1, m)$ 以上。
$lwork$	INTEGER。配列 $work$ のサイズ ( $lwork \geq n$ )。 $lwork = -1$ の場合、ワークスペースのクエリとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエントリとして返す。xerbla は $lwork$ に関するエラー・メッセージを生成しない。  $lwork$ の推奨値は、「アプリケーション・ノート」を参照。

### 出力パラメータ

$a$	$m \times m$ の直交行列 $Q$ の先頭から $n$ 個の列によって上書きされる。
$work(1)$	$info = 0$ の場合、終了時に、最適なパフォーマンスを得るために必要な $lwork$ の最小値が $work(1)$ に格納される。以降の実行では、この $lwork$ 値を使用する。
$info$	INTEGER。 $info = 0$ の場合、実行は正常に終了したことを示す。 $info = -i$ の場合、 $i$ 番目のパラメータの値が不正であったことを示す。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `orgqr` のインターフェイスの詳細を以下に示す。

$a$	サイズ $(m, n)$ の行列 $A$ を格納する。
$\tau$	長さ $(k)$ のベクトルを格納する。

### アプリケーション・ノート

パフォーマンスを改善するには、 $lwork = n * blocksize$  に設定する。 $blocksize$  は、ブロック・アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。必要なワークスペースの大きさがわからない場合は、最初の実行では  $lwork$  を大きめの値に設定する。終了時に、 $work(1)$  の値を調べ、以降はその値を使用する。

計算で求めた  $Q$  は、正確な直交行列と行列  $E$  ( $\|E\|_2 = O(\varepsilon) \|A\|_2$ ,  $\varepsilon$  はマシン精度) だけ異なる。

浮動小数演算のおおよその総数は、  
 $4 * m * n * k - 2 * (m + n) * k^2 + (4/3) * k^3$  である。  
 $n = k$  の場合、この値は約  $(2/3) * n^2 * (3m - n)$  になる。

このルーチンの複素数版は、[?ungqr](#) である。

---

## ?ormqr

実行列に `?geqrf` または `?geqpf` で求めた  
 $QR$  因子分解の直交行列  $Q$  を掛ける。

---

### 構文

#### Fortran 77:

```
call sormqr(side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork, info)
call dormqr(side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork, info)
```

#### Fortran 95:

```
call ormqr(a, tau, c [,side] [,trans] [,info])
```

### 説明

このルーチンは、実行列  $C$  に  $Q$  または  $Q^T$  を掛ける。 $Q$  は、ルーチン [sgeqrf/dgeqrf](#) または [sgeqpf/dgeqpf](#) で求めた  $QR$  因子分解の直交行列  $Q$  である。

このルーチンを使用すれば、パラメータ  $side$  と  $trans$  の値に従って、 $QC$ 、 $Q^TC$ 、 $CQ$ 、または  $CQ^T$  のいずれかの行列積を求めることができる ( $C$  の値は上書きされる)。

**入力パラメータ**

<i>side</i>	CHARACTER*1。'L' または 'R' でなければならない。 <i>side</i> = 'L' の場合、 $Q$ または $Q^T$ は、 $C$ に左側から適用される。 <i>side</i> = 'R' の場合、 $Q$ または $Q^T$ は、 $C$ に右側から適用される。
<i>trans</i>	CHARACTER*1。'N' または 'T' でなければならない。 <i>trans</i> = 'N' の場合、 $C$ に $Q$ を掛ける。 <i>trans</i> = 'T' の場合、 $C$ に $Q^T$ を掛ける。
<i>m</i>	INTEGER。行列 $C$ の行数 ( $m \geq 0$ )。
<i>n</i>	INTEGER。行列 $C$ の列数 ( $n \geq 0$ )。
<i>k</i>	INTEGER。その積が行列 $Q$ となる基本リフレクタの個数。次の制約がある。 $0 \leq k \leq m$ ( <i>side</i> = 'L' の場合)。 $0 \leq k \leq n$ ( <i>side</i> = 'R' の場合)。
<i>a, work, tau, c</i>	REAL (sgeqrf の場合) DOUBLE PRECISION (dgeqrf の場合)。 配列： <i>a</i> ( <i>lda</i> ,*) と <i>tau</i> (*) は、sgeqrf / dgeqrf または sgeqp / dgeqp から返された配列である。 <i>a</i> の第 2 次元は、 $\max(1, k)$ 以上でなければならない。 <i>tau</i> の次元は、 $\max(1, k)$ 以上でなければならない。  <i>c</i> ( <i>ldc</i> ,*) には、行列 $C$ を格納する。 <i>c</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。  <i>work</i> ( <i>lwork</i> ) は、ワークスペース配列である。
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。次の制約がある。 $lda \geq \max(1, m)$ ( <i>side</i> = 'L' の場合)。 $lda \geq \max(1, n)$ ( <i>side</i> = 'R' の場合)。
<i>ldc</i>	INTEGER。 <i>c</i> の第 1 次元。次の制約がある： $ldc \geq \max(1, m)$ 。
<i>lwork</i>	INTEGER。配列 <i>work</i> のサイズ。次の制約がある。 $lwork \geq \max(1, n)$ ( <i>side</i> = 'L' の場合)。 $lwork \geq \max(1, m)$ ( <i>side</i> = 'R' の場合)。 <i>lwork</i> = -1 の場合、ワークスペースのクエリとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリとして返す。xerbla は <i>lwork</i> に関するエラー・メッセージを生成しない。

*lwork* の推奨値は、「アプリケーション・ノート」を参照。

## 出力パラメータ

<i>c</i>	( <i>side</i> と <i>trans</i> の値に従って) $QC$ 、 $Q^T C$ 、 $CQ$ 、または $CQ^T$ のいずれかの積によって上書きされる。
<i>work</i> (1)	<i>info</i> = 0 の場合、終了時に、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が <i>work</i> (1) に格納される。以降の実行では、この <i>lwork</i> 値を使用する。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正であったことを示す。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `ormqr` のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ ( <i>r</i> , <i>k</i> ) の行列 <i>A</i> を格納する。 <i>r</i> = <i>m</i> ( <i>side</i> = 'L' の場合)。 <i>r</i> = <i>n</i> ( <i>side</i> = 'R' の場合)。
<i>tau</i>	長さ ( <i>k</i> ) のベクトルを格納する。
<i>c</i>	サイズ ( <i>m</i> , <i>n</i> ) の行列 <i>C</i> を格納する。
<i>side</i>	'L' または 'R' でなければならない。デフォルト値は 'L'。
<i>trans</i>	'N' または 'T' でなければならない。デフォルト値は 'N'。

## アプリケーション・ノート

パフォーマンスを改善するには、*lwork* = *n*\**blocksize* (*side* = 'L' の場合) または *lwork* = *m*\**blocksize* (*side* = 'R' の場合) に設定する。*blocksize* は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。必要なワークスペースの大きさがわからない場合は、最初の実行では *lwork* を大きめの値に設定する。終了時に、*work*(1) の値を調べ、以降はその値を使用する。

このルーチンの複素数版は、[?unmqr](#) である。

---

## ?ungqr

?geqrf で求めた  $QR$  因子分解の複素ユニタリ行列  $Q$  を生成する。

---

### 構文

#### Fortran 77:

```
call cungqr(m, n, k, a, lda, tau, work, lwork, info)
call zungqr(m, n, k, a, lda, tau, work, lwork, info)
```

#### Fortran 95:

```
call ungqr(a, tau [,info])
```

### 説明

このルーチンは、ルーチン [cgeqrf/zgeqrf](#) または [cgeqpf/zgeqpf](#) によって求めた  $QR$  因子分解のユニタリ行列  $Q$  ( $m \times m$ ) の全部または一部を生成する。このルーチンは、[cgeqrf/zgeqrf](#) または [cgeqpf/zgeqpf](#) を呼び出した後で使用する。

$Q$  は、通常、 $m \times p$  の行列  $A$  ( $m \geq p$ ) の  $QR$  因子分解によって決定される。行列  $Q$  の全体を計算するには、次のように呼び出す。

```
call ?ungqr(m, m, p, a, lda, tau, work, lwork, info)
```

$Q$  の先頭から  $p$  個の列 ( $A$  の列で張られた空間内で直交基を形成) を計算するには、次のように呼び出す。

```
call ?ungqr(m, p, p, a, lda, tau, work, lwork, info)
```

$A$  の先頭から  $k$  個の列に関して  $QR$  因子分解の行列  $Q^k$  を求めるには、次のように呼び出す。

```
call ?ungqr(m, m, k, a, lda, tau, work, lwork, info)
```

$Q^k$  の先頭から  $k$  個の列 ( $A$  の先頭から  $k$  個の列で張られた空間内で直交基を形成) を求めるには、次のように呼び出す。

```
call ?ungqr(m, k, k, a, lda, tau, work, lwork, info)
```

### 入力パラメータ

$m$	INTEGER。ユニタリ行列 $Q$ の次数 ( $m \geq 0$ )。
$n$	INTEGER。計算する $Q$ の列数 ( $0 \leq n \leq m$ )。

$k$	INTEGER。その積が行列 $Q$ となる基本リフレクタの個数 ( $0 \leq k \leq n$ )。
$a, \tau, work$	COMPLEX (cungqr の場合) DOUBLE COMPLEX (zungqr の場合)。 配列: $a(lda,*)$ と $\tau(*)$ は、cgeqrf/zgeqrf または cgeqpf/zgeqpf から返された配列である。 $a$ の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $\tau$ の次元は、 $\max(1, k)$ 以上でなければならない。  $work(lwork)$ は、ワークスペース配列である。
$lda$	INTEGER。 $a$ の第 1 次元。 $\max(1, m)$ 以上。
$lwork$	INTEGER。配列 $work$ のサイズ ( $lwork \geq n$ )。 $lwork = -1$ の場合、ワークスペースのクエリとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエントリとして返す。xerbla は $lwork$ に関するエラー・メッセージを生成しない。  $lwork$ の推奨値は、「アプリケーション・ノート」を参照。

### 出力パラメータ

$a$	$m \times m$ のユニタリ行列 $Q$ の先頭から $n$ 個の列によって上書きされる。
$work(1)$	$info = 0$ の場合、終了時に、最適なパフォーマンスを得るために必要な $lwork$ の最小値が $work(1)$ に格納される。以降の実行では、この $lwork$ 値を使用する。
$info$	INTEGER。 $info = 0$ の場合、実行は正常に終了したことを示す。 $info = -i$ の場合、 $i$ 番目のパラメータの値が不正であったことを示す。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `ungqr` のインターフェイスの詳細を以下に示す。

$a$	サイズ $(m, n)$ の行列 $A$ を格納する。
$\tau$	長さ $(k)$ のベクトルを格納する。

## アプリケーション・ノート

パフォーマンスを改善するには、 $lwork = n * blocksize$  に設定する。 $blocksize$  は、ブロック・アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。必要なワークスペースの大きさがわからない場合は、最初の実行では  $lwork$  を大きめの値に設定する。終了時に、 $work(1)$  の値を調べ、以降はその値を使用する。

計算で求めた  $Q$  は、正確なユニタリ行列と行列  $E$  ( $\|E\|_2 = O(\epsilon) \|A\|_2$ ,  $\epsilon$  はマシン精度) だけ異なる。

浮動小数演算のおおよその総数は、  
 $16 * m * n * k - 8 * (m + n) * k^2 + (16/3) * k^3$  である。  
 $n = k$  の場合、この値は約  $(8/3) * n^2 * (3m - n)$  になる。

このルーチンの実数版は、[?orgqr](#) である。

## ?unmqr

複素行列に [?geqrf](#) で求めた  $QR$  因子分解のユニタリ行列  $Q$  を掛ける。

### 構文

#### Fortran 77:

```
call cunmqr(side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork, info)
call zunmqr(side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork, info)
```

#### Fortran 95:

```
call unmqr(a, tau, c [,side] [,trans] [,info])
```

### 説明

このルーチンは、矩形の複素行列  $C$  に  $Q$  または  $Q^H$  を掛ける。 $Q$  は、ルーチン [cgeqrf/zgeqrf](#) または [cgeqpf/zgeqpf](#) で求めた  $QR$  因子分解のユニタリ行列  $Q$  である。

このルーチンを使用すれば、パラメータ  $side$  と  $trans$  の値に従って、 $QC$ 、 $Q^H C$ 、 $CQ$ 、または  $CQ^H$  のいずれかの行列積を求めることができる ( $C$  の値は上書きされる)。

## 入力パラメータ

<i>side</i>	<p>CHARACTER*1。'L' または 'R' でなければならない。  <i>side</i> = 'L' の場合、<math>Q</math> または <math>Q^H</math> は、<math>C</math> に左側から適用される。  <i>side</i> = 'R' の場合、<math>Q</math> または <math>Q^H</math> は、<math>C</math> に右側から適用される。</p>
<i>trans</i>	<p>CHARACTER*1。'N' または 'C' でなければならない。  <i>trans</i> = 'N' の場合、<math>C</math> に <math>Q</math> を掛ける。  <i>trans</i> = 'C' の場合、<math>C</math> に <math>Q^H</math> を掛ける。</p>
<i>m</i>	INTEGER。行列 $C$ の行数 ( $m \geq 0$ )。
<i>n</i>	INTEGER。行列 $C$ の列数 ( $n \geq 0$ )。
<i>k</i>	<p>INTEGER。その積が行列 <math>Q</math> となる基本リフレクタの個数。次の制約がある。  <math>0 \leq k \leq m</math> (<i>side</i> = 'L' の場合)。  <math>0 \leq k \leq n</math> (<i>side</i> = 'R' の場合)。</p>
<i>a, work, tau, c</i>	<p>COMPLEX (cgeqrf の場合)  DOUBLE COMPLEX (zgeqrf の場合)。  配列:  <i>a</i>(<i>lda</i>,*) と <i>tau</i>(*) は、cgeqrf / zgeqrf または cgeqp / zgeqp から返された配列である。  <i>a</i> の第 2 次元は、<math>\max(1, k)</math> 以上でなければならない。  <i>tau</i> の次元は、<math>\max(1, k)</math> 以上でなければならない。    <i>c</i>(<i>ldc</i>,*) には、行列 <math>C</math> を格納する。  <i>c</i> の第 2 次元は、<math>\max(1, n)</math> 以上でなければならない。    <i>work</i>(<i>lwork</i>) は、ワークスペース配列である。</p>
<i>lda</i>	<p>INTEGER。a の第 1 次元。次の制約がある。  <math>lda \geq \max(1, m)</math> (<i>side</i> = 'L' の場合)。  <math>lda \geq \max(1, n)</math> (<i>side</i> = 'R' の場合)。</p>
<i>ldc</i>	<p>INTEGER。c の第 1 次元。次の制約がある:  <math>ldc \geq \max(1, m)</math>。</p>
<i>lwork</i>	<p>INTEGER。配列 <i>work</i> のサイズ。次の制約がある。  <math>lwork \geq \max(1, n)</math> (<i>side</i> = 'L' の場合)。  <math>lwork \geq \max(1, m)</math> (<i>side</i> = 'R' の場合)。  <i>lwork</i> = -1 の場合、ワークスペースのクエリとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリとして返す。xerbla は <i>lwork</i> に関するエラー・メッセージを生成しない。</p>



$lwork$  の推奨値は、「アプリケーション・ノート」を参照。

### 出力パラメータ

$c$	( $side$ と $trans$ の値に従って) $QC$ 、 $Q^H C$ 、 $CQ$ 、または $CQ^H$ のいずれかの積によって上書きされる。
$work(1)$	$info=0$ の場合、終了時に、最適なパフォーマンスを得るために必要な $lwork$ の最小値が $work(1)$ に格納される。以降の実行では、この $lwork$ 値を使用する。
$info$	INTEGER。 $info=0$ の場合、実行は正常に終了したことを示す。 $info=-i$ の場合、 $i$ 番目のパラメータの値が不正であったことを示す。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `unmqr` のインターフェイスの詳細を以下に示す。

$a$	サイズ $(r,k)$ の行列 $A$ を格納する。 $r = m(side = 'L')$ の場合)。 $r = n(side = 'R')$ の場合)。
$\tau$	長さ $(k)$ のベクトルを格納する。
$c$	サイズ $(m,n)$ の行列 $C$ を格納する。
$side$	'L' または 'R' でなければならない。デフォルト値は 'L'。
$trans$	'N' または 'C' でなければならない。デフォルト値は 'N'。

### アプリケーション・ノート

パフォーマンスを改善するには、 $lwork = n * blocksize(side = 'L')$  の場合) または  $lwork = m * blocksize(side = 'R')$  の場合) に設定する。 $blocksize$  は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。必要なワークスペースの大きさがわからない場合は、最初の実行では  $lwork$  を大きめの値に設定する。終了時に、 $work(1)$  の値を調べ、以降はその値を使用する。

このルーチンの実数版は、[zormqr](#) である。

### ?gelqf

$m \times n$  の一般行列の  $LQ$  因子分解を行う。

---

#### 構文

##### Fortran 77:

```
call sgelqf(m, n, a, lda, tau, work, lwork, info)
call dgelqf(m, n, a, lda, tau, work, lwork, info)
call cgelqf(m, n, a, lda, tau, work, lwork, info)
call zgelqf(m, n, a, lda, tau, work, lwork, info)
```

##### Fortran 95:

```
call gelqf(a [,tau] [,info])
```

#### 説明

このルーチンは、 $m \times n$  の一般行列  $A$  の  $LQ$  因子分解を行う ([4-5 ページ](#)の直交因子分解を参照)。ピボット演算は行わない。

このルーチンでは、行列  $Q$  を明示的な形式で表現しない。代わりに、 $Q$  は、 $\min(m, n)$  個の基本リフレクタの積として表現される。一連のルーチンでは、この形式で表現される  $Q$  を操作する。

#### 入力パラメータ

$m$	INTEGER。行列 $A$ の行数 ( $m \geq 0$ )。
$n$	INTEGER。行列 $A$ の列数 ( $n \geq 0$ )。
$a, work$	REAL (sgelqf の場合 ) DOUBLE PRECISION (dgelqf の場合 ) COMPLEX (cgelqf の場合 ) DOUBLE COMPLEX (zgelqf の場合 )。 配列: $a(lda,*)$ には、行列 $A$ を格納する。 $a$ の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $work(lwork)$ は、ワークスペース配列である。
$lda$	INTEGER。 $a$ の第 1 次元。 $\max(1, m)$ 以上。

*lwork* INTEGER。配列 *work* のサイズ。 $\max(1, m)$  以上。  
*lwork* = -1 の場合、ワークスペースのクエリとみなされ、ルーチンは *work* 配列の最適サイズだけを計算し、その値を *work* 配列の最初のエントリとして返す。xerbla は *lwork* に関するエラー・メッセージを生成しない。  
*lwork* の推奨値は、「アプリケーション・ノート」を参照。

## 出力パラメータ

*a* 因子分解のデータによって、次のように上書きされる。  
 $m \leq n$  の場合、対角線より上の成分は、ユニタリ (直交) 行列  $Q$  の各成分によって上書きされる。下三角は、下三角行列  $L$  の対応する成分によって上書きされる。  
 $m > n$  の場合、厳密な上三角部分は、行列  $Q$  の各成分によって上書きされる。残りの成分は、 $m \times n$  の下台形行列  $L$  の対応する成分によって上書きされる。

*tau* REAL (sgelqf の場合)  
DOUBLE PRECISION (dgelqf の場合)  
COMPLEX (cgelqf の場合)  
DOUBLE COMPLEX (zgelqf の場合)。  
配列、次元は  $\max(1, \min(m, n))$  以上。  
行列  $Q$  に関するその他の情報も格納される。

*work(1)* *info* = 0 の場合、終了時に、最適なパフォーマンスを得るために必要な *lwork* の最小値が *work(1)* に格納される。以降の実行では、この *lwork* 値を使用する。

*info* INTEGER。  
*info* = 0 の場合、実行は正常に終了したことを示す。  
*info* = -*i* の場合、*i* 番目のパラメータの値が不正であったことを示す。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン gelsqf のインターフェイスの詳細を以下に示す。

*a* サイズ ( $m, n$ ) の行列  $A$  を格納する。

`tau`                      長さ  $\min(m, n)$  のベクトルを格納する。

## アプリケーション・ノート

パフォーマンスを改善するには、 $lwork = m * blocksize$  に設定する。 $blocksize$  は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。

必要なワークスペースの大きさがわからない場合は、最初の実行では  $lwork$  を大きめの値に設定する。終了時に、`work(1)` の値を調べ、以降はその値を使用する。

計算で求めた因子分解は、行列  $A + E$  ( $\|E\|_2 = O(\epsilon) \|A\|_2$ ) を因子分解したそのものである。

実数型の浮動小数点演算のおおよその回数は次のとおりである。

$(4/3)n^3$                       ( $m = n$  の場合)、

$(2/3)n^2(3m - n)$               ( $m > n$  の場合)、

$(2/3)m^2(3n - m)$               ( $m < n$  の場合)。

複素数型の演算回数は、この 4 倍になる。

与えられた行列  $B$  のすべての列  $b$  に対して  $\|Ax - b\|_2$  が最小になるような劣決定最小二乗問題の最小ノルム解を見つけるには、以下の手順でルーチン呼び出す。

`?gelqf` (このルーチン)              因子分解  $A = LQ$  を行う。

`?trsm` (BLAS ルーチン)               $LY = B$  を  $Y$  について解く。

`?ormlq`                                   $X = (Q_1)^T Y$  を計算する (実行列の場合)。

`?unmlq`                                   $X = (Q_1)^H Y$  を計算する (複素行列の場合)。

(計算で求めた  $X$  の列は、最小ノルムの解ベクトル  $x$  である。ここで、 $A$  は、 $m \times n$  の行列 ( $m < n$ ) である。 $Q_1$  は、 $Q$  の先頭から  $m$  個の列を表す)。

$Q$  の成分を明示的に計算するには、次のルーチン呼び出す。

`?orglq`                                  (実行列の場合)

`?unglq`                                  (複素行列の場合)

---

## ?orglq

?gelqf で求めた  $LQ$  因子分解の実直交行列  $Q$  を生成する。

---

### 構文

#### Fortran 77:

```
call sorglq(m, n, k, a, lda, tau, work, lwork, info)
call dorglq(m, n, k, a, lda, tau, work, lwork, info)
```

#### Fortran 95:

```
call orglq(a, tau [,info])
```

### 説明

このルーチンは、ルーチン [sgelqf/dgelqf](#) によって求めた  $LQ$  因子分解の直交行列  $Q$  ( $n \times n$ ) の全部または一部を生成する。このルーチンは、sgelqf/dgelqf を呼び出した後で使用する。

$Q$  は、通常、 $p \times n$  の行列  $A$  ( $n \geq p$ ) の  $LQ$  因子分解によって決定される。行列  $Q$  の全体を計算するには、次のように呼び出す。

```
call ?orglq(n, n, p, a, lda, tau, work, lwork, info)
```

$Q$  の先頭から  $p$  個の行 ( $A$  の行で張られた空間内で直交基を形成) を計算するには、次のように呼び出す。

```
call ?orglq(p, n, p, a, lda, tau, work, lwork, info)
```

$A$  の先頭から  $k$  個の行に関して  $LQ$  因子分解の行列  $Q^k$  を求めるには、次のように呼び出す。

```
call ?orglq(n, n, k, a, lda, tau, work, lwork, info)
```

$Q^k$  の先頭から  $k$  個の行 ( $A$  の先頭から  $k$  個の行で張られた空間内で直交基を形成) を求めるには、次のように呼び出す。

```
call ?orgqr(k, n, k, a, lda, tau, work, lwork, info)
```

### 入力パラメータ

$m$                     INTEGER。計算する  $Q$  の行数 ( $0 \leq m \leq n$ )。

$n$                     INTEGER。直交行列  $Q$  の次数 ( $n \geq m$ )。

$k$	INTEGER。その積が行列 $Q$ となる基本リフレクタの個数 ( $0 \leq k \leq m$ )。
$a, \tau, work$	REAL (sorglq の場合 ) DOUBLE PRECISION (dorglq の場合 )。 配列 : $a(lda,*)$ と $\tau(*)$ は、sgelqf/dgelqf から返された配列である。 $a$ の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $\tau$ の次元は、 $\max(1, k)$ 以上でなければならない。  $work(lwork)$ は、ワークスペース配列である。
$lda$	INTEGER。 $a$ の第 1 次元。 $\max(1, m)$ 以上。
$lwork$	INTEGER。配列 $work$ のサイズ。 $\max(1, m)$ 以上。 $lwork = -1$ の場合、ワークスペースのクエリとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエントリとして返す。xerbla は $lwork$ に関するエラー・メッセージを生成しない。  $lwork$ の推奨値は、「アプリケーション・ノート」を参照。

### 出力パラメータ

$a$	$n \times n$ の直交行列 $Q$ の先頭から $m$ 個の行によって上書きされる。
$work(1)$	$info = 0$ の場合、終了時に、最適なパフォーマンスを得るために必要な $lwork$ の最小値が $work(1)$ に格納される。以降の実行では、この $lwork$ 値を使用する。
$info$	INTEGER。 $info = 0$ の場合、実行は正常に終了したことを示す。 $info = -i$ の場合、 $i$ 番目のパラメータの値が不正であったことを示す。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン sorglq のインターフェイスの詳細を以下に示す。

$a$	サイズ $(m, n)$ の行列 $A$ を格納する。
$\tau$	長さ $(k)$ のベクトルを格納する。

## アプリケーション・ノート

パフォーマンスを改善するには、 $lwork = m * blocksize$  に設定する。 $blocksize$  は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。必要なワークスペースの大きさがわからない場合は、最初の実行では  $lwork$  を大きめの値に設定する。終了時に、 $work(1)$  の値を調べ、以降はその値を使用する。

計算で求めた  $Q$  は、正確な直交行列と行列  $E$  ( $\|E\|_2 = O(\varepsilon) \|A\|_2$ ,  $\varepsilon$  はマシン精度) だけ異なる。

浮動小数演算のおおよその総数は、  
 $4 * m * n * k - 2 * (m + n) * k^2 + (4/3) * k^3$  である。  
 $m = k$  の場合、この値は約  $(2/3) * m^2 * (3n - m)$  になる。

このルーチンの複素数版は、[?unglq](#) である。

## ?ormlq

実行列に [?gelqf](#) で求めた  $LQ$  因子分解の直交行列  $Q$  を掛ける。

### 構文

#### Fortran 77:

```
call sormlq(side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork, info)
call dormlq(side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork, info)
```

#### Fortran 95:

```
call ormlq(a, tau, c [,side] [,trans] [,info])
```

### 説明

このルーチンは、 $m \times n$  の実行列  $C$  に  $Q$  または  $Q^T$  を掛ける。 $Q$  は、ルーチン [sgelqf/dgelqf](#) で求めた  $LQ$  因子分解の直交行列  $Q$  である。

このルーチンを使用すれば、パラメータ  $side$  と  $trans$  の値に従って、 $QC$ 、 $Q^TC$ 、 $CQ$ 、または  $CQ^T$  のいずれかの行列積を求めることができる ( $C$  の値は上書きされる)。

## 入力パラメータ

<i>side</i>	CHARACTER*1。'L' または 'R' でなければならない。 <i>side</i> = 'L' の場合、 $Q$ または $Q^T$ は、 $C$ に左側から適用される。 <i>side</i> = 'R' の場合、 $Q$ または $Q^T$ は、 $C$ に右側から適用される。
<i>trans</i>	CHARACTER*1。'N' または 'T' でなければならない。 <i>trans</i> = 'N' の場合、 $C$ に $Q$ を掛ける。 <i>trans</i> = 'T' の場合、 $C$ に $Q^T$ を掛ける。
<i>m</i>	INTEGER。行列 $C$ の行数 ( $m \geq 0$ )。
<i>n</i>	INTEGER。行列 $C$ の列数 ( $n \geq 0$ )。
<i>k</i>	INTEGER。その積が行列 $Q$ となる基本リフレクタの個数。次の制約がある。 $0 \leq k \leq m$ ( <i>side</i> = 'L' の場合)。 $0 \leq k \leq n$ ( <i>side</i> = 'R' の場合)。
<i>a, work, tau, c</i>	REAL (sormlq の場合) DOUBLE PRECISION (dormlq の場合)。 配列: <i>a</i> ( <i>lda</i> ,*) と <i>tau</i> (*) は、?gelqf から返された配列である。 <i>a</i> の第 2 次元は、 $\max(1, m)$ 以上 ( <i>side</i> = 'L' の場合) または $\max(1, n)$ 以上 ( <i>side</i> = 'R' の場合) でなければならない。 <i>tau</i> の次元は、 $\max(1, k)$ 以上でなければならない。  <i>c</i> ( <i>ldc</i> ,*) には、行列 $C$ を格納する。 <i>c</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。  <i>work</i> ( <i>lwork</i> ) は、ワークスペース配列である。
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $lda \geq \max(1, k)$ 。
<i>ldc</i>	INTEGER。 <i>c</i> の第 1 次元。 $ldc \geq \max(1, m)$ 。
<i>lwork</i>	INTEGER。配列 <i>work</i> のサイズ。次の制約がある。 $lwork \geq \max(1, n)$ ( <i>side</i> = 'L' の場合)。 $lwork \geq \max(1, m)$ ( <i>side</i> = 'R' の場合)。 <i>lwork</i> = -1 の場合、ワークスペースのクエリとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリとして返す。xerbla は <i>lwork</i> に関するエラー・メッセージを生成しない。  <i>lwork</i> の推奨値は、「アプリケーション・ノート」を参照。



**出力パラメータ**

<i>c</i>	( <i>side</i> と <i>trans</i> の値に従って) $QC$ 、 $Q^TC$ 、 $CQ$ 、または $CQ^T$ のいずれかの積によって上書きされる。
<i>work</i> (1)	<i>info</i> = 0 の場合、終了時に、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が <i>work</i> (1) に格納される。以降の実行では、この <i>lwork</i> 値を使用する。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正であったことを示す。

**Fortran 95 インターフェイス・ノート**

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `ormlq` のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ ( <i>k</i> , <i>m</i> ) の行列 <i>A</i> を格納する。
<i>tau</i>	長さ ( <i>k</i> ) のベクトルを格納する。
<i>c</i>	サイズ ( <i>m</i> , <i>n</i> ) の行列 <i>C</i> を格納する。
<i>side</i>	'L' または 'R' でなければならない。デフォルト値は 'L'。
<i>trans</i>	'N' または 'T' でなければならない。デフォルト値は 'N'。

**アプリケーション・ノート**

パフォーマンスを改善するには、 $lwork = n * blocksize$  (*side* = 'L' の場合) または  $lwork = m * blocksize$  (*side* = 'R' の場合) に設定する。*blocksize* は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。必要なワークスペースの大きさがわからない場合は、最初の実行では *lwork* を大きめの値に設定する。終了時に、*work*(1) の値を調べ、以降はその値を使用する。

このルーチンの複素数版は、[?unmlq](#) である。

### ?unglq

?gelqf で求めた  $LQ$  因子分解の複素ユニタリ行列  $Q$  を生成する。

---

#### 構文

##### Fortran 77:

```
call cunglq(m, n, k, a, lda, tau, work, lwork, info)
call zunglq(m, n, k, a, lda, tau, work, lwork, info)
```

##### Fortran 95:

```
call unglq(a, tau [,info])
```

#### 説明

このルーチンは、ルーチン [cgelqf/zgelqf](#) で求めた  $LQ$  因子分解のユニタリ行列  $Q$  ( $n \times n$ ) の全部または一部を生成する。このルーチンは、[cgelqf/zgelqf](#) を呼び出した後で使用する。

$Q$  は、通常、 $p \times n$  の行列  $A$  ( $n \geq p$ ) の  $LQ$  因子分解によって決定される。行列  $Q$  の全体を計算するには、次のように呼び出す。

```
call ?unglq(n, n, p, a, lda, tau, work, lwork, info)
```

$Q$  の先頭から  $p$  個の行 ( $A$  の行で張られた空間内で直交基を形成) を計算するには、次のように呼び出す。

```
call ?unglq(p, n, p, a, lda, tau, work, lwork, info)
```

$A$  の先頭から  $k$  個の行に関して  $LQ$  因子分解の行列  $Q^k$  を求めるには、次のように呼び出す。

```
call ?unglq(n, n, k, a, lda, tau, work, lwork, info)
```

$Q^k$  の先頭から  $k$  個の行 ( $A$  の先頭から  $k$  個の行で張られた空間内で直交基を形成) を求めるには、次のように呼び出す。

```
call ?ungqr(k, n, k, a, lda, tau, work, lwork, info)
```

#### 入力パラメータ

$m$                     INTEGER。計算する  $Q$  の行数 ( $0 \leq m \leq n$ )。

$n$                     INTEGER。ユニタリ行列  $Q$  の次数 ( $n \geq m$ )。

$k$	INTEGER。その積が行列 $Q$ となる基本リフレクタの個数 ( $0 \leq k \leq m$ )。
$a, \tau, work$	COMPLEX (cunglq の場合) DOUBLE COMPLEX (zunglq の場合)。 配列： $a(lda,*)$ と $\tau(*)$ は、sgelqf/dgelqf から返された配列である。 $a$ の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $\tau$ の次元は、 $\max(1, k)$ 以上でなければならない。  $work(lwork)$ は、ワークスペース配列である。
$lda$	INTEGER。 $a$ の第 1 次元。 $\max(1, m)$ 以上。
$lwork$	INTEGER。配列 $work$ のサイズ。 $\max(1, m)$ 以上。 $lwork = -1$ の場合、ワークスペースのクエリとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエントリとして返す。xerbla は $lwork$ に関するエラー・メッセージを生成しない。  $lwork$ の推奨値は、「アプリケーション・ノート」を参照。

### 出力パラメータ

$a$	$n \times n$ のユニタリ行列 $Q$ の先頭から $m$ 個の行によって上書きされる。
$work(1)$	$info = 0$ の場合、終了時に、最適なパフォーマンスを得るために必要な $lwork$ の最小値が $work(1)$ に格納される。以降の実行では、この $lwork$ 値を使用する。
$info$	INTEGER。 $info = 0$ の場合、実行は正常に終了したことを示す。 $info = -i$ の場合、 $i$ 番目のパラメータの値が不正であったことを示す。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `unglq` のインターフェイスの詳細を以下に示す。

$a$	サイズ $(m, n)$ の行列 $A$ を格納する。
$\tau$	長さ $(k)$ のベクトルを格納する。

### アプリケーション・ノート

パフォーマンスを改善するには、 $lwork = m * blocksize$  に設定する。 $blocksize$  は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。必要なワークスペースの大きさがわからない場合は、最初の実行では  $lwork$  を大きめの値に設定する。終了時に、 $work(1)$  の値を調べ、以降はその値を使用する。

計算で求めた  $Q$  は、正確なユニタリ行列と行列  $E$  ( $\|E\|_2 = O(\epsilon) \|A\|_2$ 、 $\epsilon$  はマシン精度) だけ異なる。

浮動小数演算のおおよその総数は、  
 $16 * m * n * k - 8 * (m + n) * k^2 + (16/3) * k^3$  である。  
 $m = k$  の場合、この値は約  $(8/3) * m^2 * (3n - m)$  になる。

このルーチンの実数版は、[?orgqlq](#) である。

---

## ?unmlq

複素行列に [?gelqf](#) で求めた  $LQ$  因子分解のユニタリ行列  $Q$  を掛ける。

---

### 構文

#### Fortran 77:

```
call cunmlq(side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork, info)
call zunmlq(side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork, info)
```

#### Fortran 95:

```
call unmlq(a, tau, c [,side] [,trans] [,info])
```

### 説明

このルーチンは、 $m \times n$  の複素行列  $C$  に  $Q$  または  $Q^H$  を掛ける。 $Q$  は、ルーチン [cgelqf](#)/[zgelqf](#) で求めた  $LQ$  因子分解のユニタリ行列  $Q$  である。

このルーチンを使用すれば、パラメータ  $side$  と  $trans$  の値に従って、 $QC$ 、 $Q^H C$ 、 $CQ$ 、または  $CQ^H$  のいずれかの行列積を求めることができる ( $C$  の値は上書きされる)。

**入力パラメータ**

<i>side</i>	CHARACTER*1。'L' または 'R' でなければならない。 <i>side</i> = 'L' の場合、 $Q$ または $Q^H$ は、 $C$ に左側から適用される。 <i>side</i> = 'R' の場合、 $Q$ または $Q^H$ は、 $C$ に右側から適用される。
<i>trans</i>	CHARACTER*1。'N' または 'C' でなければならない。 <i>trans</i> = 'N' の場合、 $C$ に $Q$ を掛ける。 <i>trans</i> = 'C' の場合、 $C$ に $Q^H$ を掛ける。
<i>m</i>	INTEGER。行列 $C$ の行数 ( $m \geq 0$ )。
<i>n</i>	INTEGER。行列 $C$ の列数 ( $n \geq 0$ )。
<i>k</i>	INTEGER。その積が行列 $Q$ となる基本リフレクタの個数。次の制約がある。 $0 \leq k \leq m$ ( <i>side</i> = 'L' の場合)。 $0 \leq k \leq n$ ( <i>side</i> = 'R' の場合)。
<i>a, work, tau, c</i>	COMPLEX (cunmlq の場合) DOUBLE COMPLEX (zunmlq の場合)。 配列： <i>a</i> ( <i>lda</i> ,*) と <i>tau</i> (*) は、?gelqf から返された配列である。 <i>a</i> の第 2 次元は、 $\max(1, m)$ 以上 ( <i>side</i> = 'L' の場合) または $\max(1, n)$ 以上 ( <i>side</i> = 'R' の場合) でなければならない。 <i>tau</i> の次元は、 $\max(1, k)$ 以上でなければならない。  <i>c</i> ( <i>ldc</i> ,*) には、行列 $C$ を格納する。 <i>c</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。  <i>work</i> ( <i>lwork</i> ) は、ワークスペース配列である。
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $lda \geq \max(1, k)$ 。
<i>ldc</i>	INTEGER。 <i>c</i> の第 1 次元。 $ldc \geq \max(1, m)$ 。
<i>lwork</i>	INTEGER。配列 <i>work</i> のサイズ。次の制約がある。 $lwork \geq \max(1, n)$ ( <i>side</i> = 'L' の場合)。 $lwork \geq \max(1, m)$ ( <i>side</i> = 'R' の場合)。 <i>lwork</i> = -1 の場合、ワークスペースのクエリとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリとして返す。xerbla は <i>lwork</i> に関するエラー・メッセージを生成しない。  <i>lwork</i> の推奨値は、「アプリケーション・ノート」を参照。

## 出力パラメータ

<i>c</i>	( <i>side</i> と <i>trans</i> の値に従って) $QC$ 、 $Q^H C$ 、 $CQ$ 、または $CQ^H$ のいずれかの積によって上書きされる。
<i>work</i> (1)	<i>info</i> = 0 の場合、終了時に、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が <i>work</i> (1) に格納される。以降の実行では、この <i>lwork</i> 値を使用する。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正であったことを示す。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `unmlq` のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ ( <i>k</i> , <i>m</i> ) の行列 <i>A</i> を格納する。
<i>tau</i>	長さ ( <i>k</i> ) のベクトルを格納する。
<i>c</i>	サイズ ( <i>m</i> , <i>n</i> ) の行列 <i>C</i> を格納する。
<i>side</i>	'L' または 'R' でなければならない。デフォルト値は 'L'。
<i>trans</i>	'N' または 'C' でなければならない。デフォルト値は 'N'。

## アプリケーション・ノート

パフォーマンスを改善するには、 $lwork = n * blocksize$  (*side* = 'L' の場合) または  $lwork = m * blocksize$  (*side* = 'R' の場合) に設定する。*blocksize* は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。必要なワークスペースの大きさがわからない場合は、最初の実行では *lwork* を大きめの値に設定する。終了時に、*work*(1) の値を調べ、以降はその値を使用する。

このルーチンの実数版は、[?ormlq](#) である。

---

## ?geqlf

$m \times n$  の一般行列の  $QL$  因子分解を行う。

---

### 構文

#### Fortran 77:

```
call sgeqlf(m, n, a, lda, tau, work, lwork, info)
call dgeqlf(m, n, a, lda, tau, work, lwork, info)
call cgeqlf(m, n, a, lda, tau, work, lwork, info)
call zgeqlf(m, n, a, lda, tau, work, lwork, info)
```

#### Fortran 95:

```
call geqlf(a [,tau] [,info])
```

### 説明

このルーチンは、 $m \times n$  の一般行列  $A$  の  $QL$  因子分解を行う。  
ピボット演算は行わない。

このルーチンでは、行列  $Q$  を明示的な形式で表現しない。代わりに、 $Q$  は、 $\min(m, n)$  個の基本リフレクタの積として表現される。一連のルーチンでは、この形式で表現される  $Q$  を操作する。

### 入力パラメータ

$m$	INTEGER。行列 $A$ の行数 ( $m \geq 0$ )。
$n$	INTEGER。行列 $A$ の列数 ( $n \geq 0$ )。
$a, work$	REAL (sgeqlf の場合 ) DOUBLE PRECISION (dgeqlf の場合 ) COMPLEX (cgeqlf の場合 ) DOUBLE COMPLEX (zgeqlf の場合 )。 配列： $a(lda, *)$ には、行列 $A$ を格納する。 $a$ の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $work(lwork)$ は、ワークスペース配列である。
$lda$	INTEGER。 $a$ の第 1 次元。 $\max(1, m)$ 以上。

*lwork* INTEGER。配列 *work* のサイズ。 $\max(1, n)$  以上。  
*lwork* = -1 の場合、ワークスペースのクエリとみなされ、ルーチンは *work* 配列の最適サイズだけを計算し、その値を *work* 配列の最初のエントリとして返す。xerbla は *lwork* に関するエラー・メッセージを生成しない。  
*lwork* の推奨値は、「アプリケーション・ノート」を参照。

## 出力パラメータ

*a* 終了時に、因子分解データによって次のように上書きされる。  
 $m \geq n$  の場合、部分配列  $a(m-n+1:m, 1:n)$  の下三角部分に、 $n \times n$  の下三角行列 *L* が格納される。  
 $m \leq n$  の場合、 $(n-m)$  番目の優対角成分とその下の各成分に、 $m \times n$  の下台形行列 *L* が格納される。  
 どちらの場合も、残りの成分は、配列 *tau* とともに、基本リフレクタの積として直交/ユニタリ行列 *Q* を表現する。

*tau* REAL (sgeqlf の場合 )  
 DOUBLE PRECISION (dgeqlf の場合 )  
 COMPLEX (cgeqlf の場合 )  
 DOUBLE COMPLEX (zgeqlf の場合 )。  
 配列、次元は  $\max(1, \min(m, n))$  以上。  
 行列 *Q* に対する基本リフレクタのスカラー係数が格納される。

*work(1)* *info* = 0 の場合、終了時に、最適なパフォーマンスを得るために必要な *lwork* の最小値が *work(1)* に格納される。

*info* INTEGER。  
*info* = 0 の場合、実行は正常に終了したことを示す。  
*info* = -*i* の場合、*i* 番目のパラメータの値が不正であったことを示す。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *geqlf* のインターフェイスの詳細を以下に示す。

*a* サイズ  $(m, n)$  の行列 *A* を格納する。

*tau* 長さ  $\min(m, n)$  のベクトルを格納する。



## アプリケーション・ノート

パフォーマンスを改善するには、 $lwork = n * blocksize$  に設定する。 $blocksize$  は、ブロック・アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる（通常は 16 ～ 64）。必要なワークスペースの大きさがわからない場合は、最初の実行では  $lwork$  を大きめの値に設定する。終了時に、 $work(1)$  の値を調べ、以降はその値を使用する。

関連ルーチン：

<a href="#">?orgql</a>	行列 $Q$ を生成する（実行列の場合）。
<a href="#">?ungql</a>	行列 $Q$ を生成する（複素行列の場合）。
<a href="#">?ormql</a>	行列 $Q$ を適用する（実行列の場合）。
<a href="#">?unmql</a>	行列 $Q$ を適用する（複素行列の場合）。

---

## ?orgql

?geqlf で求めた  $QL$  因子分解の実行列  $Q$  を生成する。

---

### 構文

#### Fortran 77:

```
call sorgql(m, N, k, a, lda, tau, work, lwork, info)
call dorgql(m, N, k, a, lda, tau, work, lwork, info)
```

#### Fortran 95:

```
call orgql(a, tau [,info])
```

### 説明

このルーチンは、直交列を持つ  $m \times n$  の実行列  $Q$  を生成する。これは、ルーチン [sgeqlf/dgeqlf](#) から返された  $k$  個の基本リフレクタ  $H_i$ （次数は  $m$ ）の積の最終  $n$  列で定義される ( $Q = H_k \cdots H_2 H_1$ )。このルーチンは、[sgeqlf/dgeqlf](#) を呼び出した後で使用する。

### 入力パラメータ

$m$	INTEGER。行列 $Q$ の行数 ( $m \geq 0$ )。
$n$	INTEGER。行列 $Q$ の列数 ( $m \geq n \geq 0$ )。

$k$	INTEGER。その積が行列 $Q$ となる基本リフレクタの個数 ( $n \geq k \geq 0$ )。
$a, \tau, work$	REAL (sorgql の場合 ) DOUBLE PRECISION (dorgql の場合 )。 配列 : $a(lda, *)$ , $\tau(*)$ , $work(lwork)$ 。  呼び出し時に、sgeqlf/dgeqlf で配列引数 $a$ の最終 $k$ 列に返されたとおりに、 $a$ の $(n - k + i)$ 番目の列に、基本リフレクタ $H_i$ (ただし、 $i = 1, 2, \dots, k$ ) を定義するベクトルが格納されていなければならない。 $\tau(i)$ には、sgeqlf/dgeqlf から返されたとおりに、基本リフレクタ $H_i$ のスカラー係数が格納されていなければならない。  $a$ の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $\tau$ の次元は、 $\max(1, k)$ 以上でなければならない。  $work(lwork)$ は、ワークスペース配列である。
$lda$	INTEGER。 $a$ の第 1 次元。 $\max(1, m)$ 以上。
$lwork$	INTEGER。配列 $work$ のサイズ。 $\max(1, n)$ 以上。 $lwork = -1$ の場合、ワークスペースのクエリとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエントリとして返す。xerbla は $lwork$ に関するエラー・メッセージを生成しない。  $lwork$ の推奨値は、「アプリケーション・ノート」を参照。

## 出力パラメータ

$a$	$m \times n$ の行列 $Q$ によって上書きされる。
$work(1)$	$info = 0$ の場合、終了時に、最適なパフォーマンスを得るために必要な $lwork$ の最小値が $work(1)$ に格納される。以降の実行では、この $lwork$ 値を使用する。
$info$	INTEGER。 $info = 0$ の場合、実行は正常に終了したことを示す。 $info = -i$ の場合、 $i$ 番目のパラメータの値が不正であったことを示す。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `orgql` のインターフェイスの詳細を以下に示す。

`a`                      サイズ  $(m, n)$  の行列  $A$  を格納する。

`tau`                     長さ  $(k)$  のベクトルを格納する。

### アプリケーション・ノート

パフォーマンスを改善するには、`lwork = n * blocksize` に設定する。`blocksize` は、ブロック・アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。必要なワークスペースの大きさがわからない場合は、最初の実行では `lwork` を大きめの値に設定する。終了時に、`work(1)` の値を調べ、以降はその値を使用する。

このルーチンの複素数版は、[?ungql](#) である。

---

## ?ungql

`?geqlf` で求めた  $QL$  因子分解の複素行列を生成する。

---

### 構文

#### Fortran 77:

```
call cungql(m, n, k, a, lda, tau, work, lwork, info)
call zungql(m, n, k, a, lda, tau, work, lwork, info)
```

#### Fortran 95:

```
call ungql(a, tau [,info])
```

### 説明

このルーチンは、直交列を持つ  $m \times n$  の複素行列  $Q$  を生成する。これは、ルーチン [cgeqlf/zgeqlf](#) から返された  $k$  個の基本リフレクタ  $H_1$  (次数は  $m$ ) の積の最終  $n$  列で定義される ( $Q = H_k \cdots H_2 H_1$ )。このルーチンは、`cgeqlf/zgeqlf` を呼び出した後で使用する。

### 入力パラメータ

`m`                      INTEGER。行列  $Q$  の行数 ( $m \geq 0$ )。

<i>n</i>	INTEGER。行列 <i>Q</i> の列数 ( $m \geq n \geq 0$ )。
<i>k</i>	INTEGER。その積が行列 <i>Q</i> となる基本リフレクタの個数 ( $n \geq k \geq 0$ )。
<i>a</i> , <i>tau</i> , <i>work</i>	COMPLEX (cungql の場合) DOUBLE COMPLEX (zungql の場合)。 配列: <i>a</i> ( <i>lda</i> ,*), <i>tau</i> (*), <i>work</i> ( <i>lwork</i> )。  呼び出し時に、cgeqlf/zgeqlf で配列引数 <i>a</i> の最終 <i>k</i> 列に返されたとおりに、 <i>a</i> の ( <i>n</i> - <i>k</i> + <i>i</i> ) 番目の列に、基本リフレクタ $H_i$ (ただし、 <i>i</i> = 1, 2, ..., <i>k</i> ) を定義するベクトルが格納されていなければならない。 <i>tau</i> ( <i>i</i> ) には、cgeqlf/zgeqlf から返されたとおりに、基本リフレクタ $H_i$ のスカラー係数が格納されていなければならない。  <i>a</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>tau</i> の次元は、 $\max(1, k)$ 以上でなければならない。  <i>work</i> ( <i>lwork</i> ) は、ワークスペース配列である。
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $\max(1, m)$ 以上。
<i>lwork</i>	INTEGER。配列 <i>work</i> のサイズ。 $\max(1, n)$ 以上。 <i>lwork</i> = -1 の場合、ワークスペースのクエリとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリとして返す。xerbla は <i>lwork</i> に関するエラー・メッセージを生成しない。  <i>lwork</i> の推奨値は、「アプリケーション・ノート」を参照。

## 出力パラメータ

<i>a</i>	$m \times n$ の行列 <i>Q</i> によって上書きされる。
<i>work</i> (1)	<i>info</i> = 0 の場合、終了時に、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が <i>work</i> (1) に格納される。以降の実行では、この <i>lwork</i> 値を使用する。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正であったことを示す。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `ungql` のインターフェイスの詳細を以下に示す。

`a`                      サイズ  $(m,n)$  の行列  $A$  を格納する。

`tau`                     長さ  $(k)$  のベクトルを格納する。

### アプリケーション・ノート

パフォーマンスを改善するには、`lwork=n*blocksize` に設定する。`blocksize` は、ブロック・アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。必要なワークスペースの大きさがわからない場合は、最初の実行では `lwork` を大きめの値に設定する。終了時に、`work(1)` の値を調べ、以降はその値を使用する。

このルーチンの実数版は、[?orgql](#) である。

---

## ?ormql

実行列に `?geqlf` で求めた  $QL$  因子分解の直交行列  $Q$  を掛ける。

---

### 構文

#### Fortran 77:

```
call sormql(side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork, info)
call dormql(side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork, info)
```

#### Fortran 95:

```
call ormql(a, tau, c [,side] [,trans] [,info])
```

### 説明

このルーチンは、 $m \times n$  の実行列  $C$  に  $Q$  または  $Q^T$  を掛ける。 $Q$  は、ルーチン [sgeqlf](#)/[dgeqlf](#) で求めた  $QL$  因子分解の直交行列  $Q$  である。

このルーチンを使用すれば、パラメータ `side` と `trans` の値に従って、 $QC$ 、 $Q^TC$ 、 $CQ$ 、または  $CQ^T$  のいずれかの行列積を求めることができる ( $C$  の値は上書きされる)。

## 入力パラメータ

<i>side</i>	<p>CHARACTER*1。'L' または 'R' でなければならない。  <i>side</i> = 'L' の場合、<math>Q</math> または <math>Q^T</math> は、<math>C</math> に左側から適用される。  <i>side</i> = 'R' の場合、<math>Q</math> または <math>Q^T</math> は、<math>C</math> に右側から適用される。</p>
<i>trans</i>	<p>CHARACTER*1。'N' または 'T' でなければならない。  <i>trans</i> = 'N' の場合、<math>C</math> に <math>Q</math> を掛ける。  <i>trans</i> = 'T' の場合、<math>C</math> に <math>Q^T</math> を掛ける。</p>
<i>m</i>	INTEGER。行列 $C$ の行数 ( $m \geq 0$ )。
<i>n</i>	INTEGER。行列 $C$ の列数 ( $n \geq 0$ )。
<i>k</i>	<p>INTEGER。その積が行列 <math>Q</math> となる基本リフレクタの個数。次の制約がある。  <math>0 \leq k \leq m</math> (<i>side</i> = 'L' の場合)。  <math>0 \leq k \leq n</math> (<i>side</i> = 'R' の場合)。</p>
<i>a, tau, c, work</i>	<p>REAL (sormql の場合)  DOUBLE PRECISION (dormql の場合)。  配列: <math>a(lda, *)</math>, <math>tau(*)</math>, <math>c ldc, *)</math>, <math>work(lwork)</math></p> <p>呼び出し時に、sgeqlf/dgeqlf で配列引数 <math>a</math> の最終 <math>k</math> 列に返されたとおりに、<math>a</math> の <math>i</math> 番目の列に、基本リフレクタ <math>H_i</math> (ただし、<math>i = 1, 2, \dots, k</math>) を定義するベクトルが格納されていなければならない。  <math>a</math> の第 2 次元は、<math>\max(1, k)</math> 以上でなければならない。</p> <p><math>tau(i)</math> には、sgeqlf/dgeqlf から返されたとおりに、基本リフレクタ <math>H_i</math> のスカラー係数が格納されていなければならない。  <math>tau</math> の次元は、<math>\max(1, k)</math> 以上でなければならない。</p> <p><math>c(ldc, *)</math> には、<math>m \times n</math> の行列 <math>C</math> を格納する。  <math>c</math> の第 2 次元は、<math>\max(1, n)</math> 以上でなければならない。</p> <p><math>work(lwork)</math> は、ワークスペース配列である。</p>
<i>lda</i>	<p>INTEGER。 <math>a</math> の第 1 次元。</p> <p><i>side</i> = 'L' の場合、<math>lda \geq \max(1, m)</math>。  <i>side</i> = 'R' の場合、<math>lda \geq \max(1, n)</math>。</p>
<i>ldc</i>	INTEGER。 $c$ の第 1 次元。 $ldc \geq \max(1, m)$ 。
<i>lwork</i>	<p>INTEGER。配列 <math>work</math> のサイズ。次の制約がある。  <math>lwork \geq \max(1, n)</math> (<i>side</i> = 'L' の場合)。  <math>lwork \geq \max(1, m)</math> (<i>side</i> = 'R' の場合)。  <math>lwork = -1</math> の場合、ワークスペースのクエリとみなされ、ルーチンは</p>

*work* 配列の最適サイズだけを計算し、その値を *work* 配列の最初のエントリとして返す。xerbla は *lwork* に関するエラー・メッセージを生成しない。

*lwork* の推奨値は、「アプリケーション・ノート」を参照。

### 出力パラメータ

<i>c</i>	( <i>side</i> と <i>trans</i> の値に従って) $QC$ 、 $Q^TC$ 、 $CQ$ 、または $CQ^T$ のいずれかの積によって上書きされる。
<i>work</i> (1)	<i>info</i> = 0 の場合、終了時に、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が <i>work</i> (1) に格納される。以降の実行では、この <i>lwork</i> 値を使用する。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正であったことを示す。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン ormql のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ ( <i>r</i> , <i>k</i> ) の行列 <i>A</i> を格納する。 $r = m$ ( <i>side</i> = 'L' の場合)。 $r = n$ ( <i>side</i> = 'R' の場合)。
<i>tau</i>	長さ ( <i>k</i> ) のベクトルを格納する。
<i>c</i>	サイズ ( <i>m</i> , <i>n</i> ) の行列 <i>C</i> を格納する。
<i>side</i>	'L' または 'R' でなければならない。デフォルト値は 'L'。
<i>trans</i>	'N' または 'T' でなければならない。デフォルト値は 'N'。

### アプリケーション・ノート

パフォーマンスを改善するには、 $lwork = n * blocksize$  ( $side = 'L'$  の場合) または  $lwork = m * blocksize$  ( $side = 'R'$  の場合) に設定する。 $blocksize$  は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。必要なワークスペースの大きさがわからない場合は、最初の実行では  $lwork$  を大きめの値に設定する。終了時に、 $work(1)$  の値を調べ、以降はその値を使用する。

このルーチンの複素数版は、[?unmql](#) である。

---

## ?unmql

複素行列に `?geqlf` で求めた  $QL$  因子分解のユニタリ行列  $Q$  を掛ける。

---

### 構文

#### Fortran 77:

```
call cunmql(side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork, info)
call zunmql(side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork, info)
```

#### Fortran 95:

```
call unmql(a, tau, c [,side] [,trans] [,info])
```

### 説明

このルーチンは、 $m \times n$  の複素行列  $C$  に  $Q$  または  $Q^H$  を掛ける。 $Q$  は、ルーチン [cgeqlf](#)/[zgeqlf](#) で求めた  $QL$  因子分解のユニタリ行列  $Q$  である。

このルーチンを使用すれば、パラメータ  $side$  と  $trans$  の値に従って、 $QC$ 、 $Q^H C$ 、 $CQ$ 、または  $CQ^H$  のいずれかの行列積を求めることができる ( $C$  の値は上書きされる)。

### 入力パラメータ

$side$	CHARACTER*1。'L' または 'R' でなければならない。 $side = 'L'$ の場合、 $Q$ または $Q^H$ は、 $C$ に左側から適用される。 $side = 'R'$ の場合、 $Q$ または $Q^H$ は、 $C$ に右側から適用される。
$trans$	CHARACTER*1。'N' または 'C' でなければならない。 $trans = 'N'$ の場合、 $C$ に $Q$ を掛ける。 $trans = 'C'$ の場合、 $C$ に $Q^H$ を掛ける。



<i>m</i>	INTEGER。行列 <i>C</i> の行数 ( $m \geq 0$ )。
<i>n</i>	INTEGER。行列 <i>C</i> の列数 ( $n \geq 0$ )。
<i>k</i>	INTEGER。その積が行列 <i>Q</i> となる基本リフレクタの個数。次の制約がある。 $0 \leq k \leq m$ ( <i>side</i> = 'L' の場合)。 $0 \leq k \leq n$ ( <i>side</i> = 'R' の場合)。
<i>a, tau, c, work</i>	COMPLEX (cunmql の場合) DOUBLE COMPLEX (zunmql の場合)。 配列: <i>a</i> ( <i>lda</i> , *), <i>tau</i> (*), <i>c</i> ( <i>ldc</i> , *), <i>work</i> ( <i>lwork</i> ) 呼び出し時に、cgeqlf/zgeqlf で配列引数 <i>a</i> の最終 <i>k</i> 列に返されたとおりに、 <i>a</i> の <i>i</i> 番目の列に、基本リフレクタ $H_i$ (ただし、 $i = 1, 2, \dots, k$ ) を定義するベクトルが格納されていなければならない。 <i>a</i> の第 2 次元は、 $\max(1, k)$ 以上でなければならない。  <i>tau</i> ( <i>i</i> ) には、cgeqlf/zgeqlf から返されたとおりに、基本リフレクタ $H_i$ のスカラ係数が格納されていなければならない。 <i>tau</i> の次元は、 $\max(1, k)$ 以上でなければならない。  <i>c</i> ( <i>ldc</i> , *) には、 $m \times n$ の行列 <i>C</i> を格納する。 <i>c</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。  <i>work</i> ( <i>lwork</i> ) は、ワークスペース配列である。
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。  <i>side</i> = 'L' の場合、 $lda \geq \max(1, m)$ 。 <i>side</i> = 'R' の場合、 $lda \geq \max(1, n)$ 。
<i>ldc</i>	INTEGER。 <i>c</i> の第 1 次元。 $ldc \geq \max(1, m)$ 。
<i>lwork</i>	INTEGER。配列 <i>work</i> のサイズ。次の制約がある。 $lwork \geq \max(1, n)$ ( <i>side</i> = 'L' の場合)。 $lwork \geq \max(1, m)$ ( <i>side</i> = 'R' の場合)。 <i>lwork</i> = -1 の場合、ワークスペースのクエリとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリとして返す。xerbla は <i>lwork</i> に関するエラー・メッセージを生成しない。  <i>lwork</i> の推奨値は、「アプリケーション・ノート」を参照。

## 出力パラメータ

<i>c</i>	( <i>side</i> と <i>trans</i> の値に従って) $QC$ 、 $Q^H C$ 、 $CQ$ 、または $CQ^H$ のいずれかの積によって上書きされる。
<i>work</i> (1)	<i>info</i> = 0 の場合、終了時に、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が <i>work</i> (1) に格納される。以降の実行では、この <i>lwork</i> 値を使用する。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正であったことを示す。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `unmql` のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ ( <i>r</i> , <i>k</i> ) の行列 <i>A</i> を格納する。 <i>r</i> = <i>m</i> ( <i>side</i> = 'L' の場合)。 <i>r</i> = <i>n</i> ( <i>side</i> = 'R' の場合)。
<i>tau</i>	長さ ( <i>k</i> ) のベクトルを格納する。
<i>c</i>	サイズ ( <i>m</i> , <i>n</i> ) の行列 <i>C</i> を格納する。
<i>side</i>	'L' または 'R' でなければならない。デフォルト値は 'L'。
<i>trans</i>	'N' または 'C' でなければならない。デフォルト値は 'N'。

## アプリケーション・ノート

パフォーマンスを改善するには、*lwork* = *n* \* *blocksize* (*side* = 'L' の場合) または *lwork* = *m* \* *blocksize* (*side* = 'R' の場合) に設定する。*blocksize* は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。必要なワークスペースの大きさがわからない場合は、最初の実行では *lwork* を大きめの値に設定する。終了時に、*work*(1) の値を調べ、以降はその値を使用する。

このルーチンの実数版は、[?ormql](#) である。

---

## ?gerqf

$m \times n$  の一般行列の  $RQ$  因子分解を行う。

---

### 構文

#### Fortran 77:

```
call sgerqf(m, n, a, lda, tau, work, lwork, info)
call dgerqf(m, n, a, lda, tau, work, lwork, info)
call cgerqf(m, n, a, lda, tau, work, lwork, info)
call zgerqf(m, n, a, lda, tau, work, lwork, info)
```

#### Fortran 95:

```
call gerqf(a [,tau] [,info])
```

### 説明

このルーチンは、 $m \times n$  の一般行列  $A$  の  $RQ$  因子分解を行う。ピボット演算は行わない。

このルーチンでは、行列  $Q$  を明示的な形式で表現しない。代わりに、 $Q$  は、 $\min(m, n)$  個の基本リフレクタの積として表現される。一連のルーチンでは、この形式で表現される  $Q$  を操作する。

### 入力パラメータ

$m$	INTEGER。行列 $A$ の行数 ( $m \geq 0$ )。
$n$	INTEGER。行列 $A$ の列数 ( $n \geq 0$ )。
$a, work$	REAL (sgerqf の場合 ) DOUBLE PRECISION (dgerqf の場合 ) COMPLEX (cgerqf の場合 ) DOUBLE COMPLEX (zgerqf の場合 )。 配列： $a(lda, *)$ には、 $m \times n$ の行列 $A$ を格納する。 $a$ の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $work(lwork)$ は、ワークスペース配列である。
$lda$	INTEGER。 $a$ の第 1 次元。 $\max(1, m)$ 以上。

**lwork** INTEGER。配列 *work* のサイズ。  $lwork \geq \max(1, m)$ 。  
 $lwork = -1$  の場合、ワークスペースのクエリとみなされ、ルーチンは *work* 配列の最適サイズだけを計算し、その値を *work* 配列の最初のエントリとして返す。xerbla は *lwork* に関するエラー・メッセージを生成しない。  
*lwork* の推奨値は、「[アプリケーション・ノート](#)」を参照。

### 出力パラメータ

**a** 終了時に、因子分解データによって次のように上書きされる。  
 $m \leq n$  の場合、部分配列  $a(1:m, n-m+1:n)$  の上三角部分に、 $m \times m$  の上三角行列  $R$  が格納される。  
 $m \geq n$  の場合、 $(m-n)$  番目の劣対角成分とその上の各成分に、 $m \times n$  の上台形行列  $R$  が格納される。  
 どちらの場合も、残りの成分は、配列 *tau* とともに、 $\min(m, n)$  個の基本リフレクタの積として直交 / ユニタリ行列  $Q$  を表現する。

**tau** REAL (sgerqf の場合 )  
 DOUBLE PRECISION (dgerqf の場合 )  
 COMPLEX (cgerqf の場合 )  
 DOUBLE COMPLEX (zgerqf の場合 )。  
 配列、次元は  $\max(1, \min(m, n))$  以上。  
 行列  $Q$  に対する基本リフレクタのスカラ係数が格納される。

**work(1)**  $info = 0$  の場合、終了時に、最適なパフォーマンスを得るために必要な *lwork* の最小値が *work(1)* に格納される。

**info** INTEGER。  
 $info = 0$  の場合、実行は正常に終了したことを示す。  
 $info = -i$  の場合、 $i$  番目のパラメータの値が不正であったことを示す。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *gerqf* のインターフェイスの詳細を以下に示す。

**a** サイズ  $(m, n)$  の行列  $A$  を格納する。

**tau** 長さ  $\min(m, n)$  のベクトルを格納する。

## アプリケーション・ノート

パフォーマンスを改善するには、 $lwork = m * blocksize$  に設定する。 $blocksize$  は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる（通常は 16 ～ 64）。必要なワークスペースの大きさがわからない場合は、最初の実行では  $lwork$  を大きめの値に設定する。終了時に、 $work(1)$  の値を調べ、以降はその値を使用する。

関連ルーチン：

<a href="#">?orgqr</a>	行列 $Q$ を生成する（実行列の場合）。
<a href="#">?ungqr</a>	行列 $Q$ を生成する（複素行列の場合）。
<a href="#">?ormqr</a>	行列 $Q$ を適用する（実行列の場合）。
<a href="#">?unmqr</a>	行列 $Q$ を適用する（複素行列の場合）。

## ?orgqr

?gerqf で求めた  $RQ$  因子分解の実行列  $Q$  を生成する。

### 構文

#### Fortran 77:

```
call sorgqr(m, n, k, a, lda, tau, work, lwork, info)
call dorgqr(m, n, k, a, lda, tau, work, lwork, info)
```

#### Fortran 95:

```
call orgqr(a, tau [,info])
```

### 説明

このルーチンは、直交行を持つ  $m \times n$  の実行列  $Q$  を生成する。これは、ルーチン [sgerqf/dgerqf](#) から返された  $k$  個の基本リフレクタ  $H_1$ （次数は  $n$ ）の積の最終  $m$  行で定義される ( $Q = H_1 H_2 \cdots H_k$ )。このルーチンは、[sgerqf/dgerqf](#) を呼び出した後で使用する。

### 入力パラメータ

$m$	INTEGER。行列 $Q$ の行数 ( $m \geq 0$ )。
$n$	INTEGER。行列 $Q$ の列数 ( $n \geq m$ )。

$k$	INTEGER。その積が行列 $Q$ となる基本リフレクタの個数 ( $m \geq k \geq 0$ )。
$a, \tau, work$	REAL (sorghr の場合) DOUBLE PRECISION (dorghr の場合)。 配列: $a(lda, *)$ , $\tau(*)$ , $work(lwork)$ 。  呼び出し時に、sorghr/dorghr で配列引数 $a$ の最終 $k$ 行に返されたとおりに、 $a$ の $(m - k + i)$ 番目の行に、基本リフレクタ $H_i$ (ただし、 $i = 1, 2, \dots, k$ ) を定義するベクトルが格納されていなければならない。 $\tau(i)$ には、sorghr/dorghr から返されたとおりに、基本リフレクタ $H_i$ のスカラー係数が格納されていなければならない。  $a$ の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $\tau$ の次元は、 $\max(1, k)$ 以上でなければならない。  $work(lwork)$ は、ワークスペース配列である。
$lda$	INTEGER。 $a$ の第 1 次元。 $\max(1, m)$ 以上。
$lwork$	INTEGER。配列 $work$ のサイズ。 $\max(1, m)$ 以上。  $lwork = -1$ の場合、ワークスペースのクエリとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエントリとして返す。xerbla は $lwork$ に関するエラー・メッセージを生成しない。  $lwork$ の推奨値は、「アプリケーション・ノート」を参照。

## 出力パラメータ

$a$	$m \times n$ の行列 $Q$ によって上書きされる。
$work(1)$	$info = 0$ の場合、終了時に、最適なパフォーマンスを得るために必要な $lwork$ の最小値が $work(1)$ に格納される。以降の実行では、この $lwork$ 値を使用する。
$info$	INTEGER。 $info = 0$ の場合、実行は正常に終了したことを示す。 $info = -i$ の場合、 $i$ 番目のパラメータの値が不正であったことを示す。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `orgrq` のインターフェイスの詳細を以下に示す。

`a`                      サイズ  $(m, n)$  の行列  $A$  を格納する。

`tau`                     長さ  $(k)$  のベクトルを格納する。

### アプリケーション・ノート

パフォーマンスを改善するには、`lwork = m * blocksize` に設定する。`blocksize` は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。必要なワークスペースの大きさがわからない場合は、最初の実行では `lwork` を大きめの値に設定する。終了時に、`work(1)` の値を調べ、以降はその値を使用する。

このルーチンの複素数版は、[?ungrq](#) である。

---

## ?ungrq

`?gerqf` で求めた  $RQ$  因子分解の複素行列  $Q$  を生成する。

---

### 構文

#### Fortran 77:

```
call cungrq(m, n, k, a, lda, tau, work, lwork, info)
call zungrq(m, n, k, a, lda, tau, work, lwork, info)
```

#### Fortran 95:

```
call ungrq(a, tau [,info])
```

### 説明

このルーチンは、直交行を持つ  $m \times n$  の複素行列  $Q$  を生成する。これは、ルーチン [sgerqf](#)/[dgerqf](#) から返された  $k$  個の基本リフレクタ  $H_i$  (次数は  $n$ ) の積の最終  $m$  行で定義される ( $Q = H_1^H H_2^H \cdots H_k^H$ )。このルーチンは、`sgerqf`/`dgerqf` を呼び出した後で使用する。

### 入力パラメータ

`m`                      INTEGER。行列  $Q$  の行数 ( $m \geq 0$ )。

<i>n</i>	INTEGER。行列 <i>Q</i> の列数 ( $n \geq m$ )。
<i>k</i>	INTEGER。その積が行列 <i>Q</i> となる基本リフレクタの個数 ( $m \geq k \geq 0$ )。
<i>a</i> , <i>tau</i> , <i>work</i>	REAL (cungrq の場合 ) DOUBLE PRECISION (zungrq の場合 )。 配列 : <i>a</i> ( <i>lda</i> ,*), <i>tau</i> (*), <i>work</i> ( <i>lwork</i> )。  呼び出し時に、sgerqf/dgerqf で配列引数 <i>a</i> の最終 <i>k</i> 行に返されたとおりに、 <i>a</i> の ( <i>m</i> - <i>k</i> + <i>i</i> ) 番目の行に、基本リフレクタ $H_i$ (ただし、 <i>i</i> = 1, 2, ..., <i>k</i> ) を定義するベクトルが格納されていなければならない。 <i>tau</i> ( <i>i</i> ) には、sgerqf/dgerqf から返されたとおりに、基本リフレクタ $H_i$ のスカラー係数が格納されていなければならない。  <i>a</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>tau</i> の次元は、 $\max(1, k)$ 以上でなければならない。  <i>work</i> ( <i>lwork</i> ) は、ワークスペース配列である。
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $\max(1, m)$ 以上。
<i>lwork</i>	INTEGER。配列 <i>work</i> のサイズ。 $\max(1, m)$ 以上。 <i>lwork</i> = -1 の場合、ワークスペースのクエリとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリとして返す。xerbla は <i>lwork</i> に関するエラー・メッセージを生成しない。  <i>lwork</i> の推奨値は、「アプリケーション・ノート」を参照。

### 出力パラメータ

<i>a</i>	$m \times n$ の行列 <i>Q</i> によって上書きされる。
<i>work</i> (1)	<i>info</i> = 0 の場合、終了時に、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が <i>work</i> (1) に格納される。以降の実行では、この <i>lwork</i> 値を使用する。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正であったことを示す。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。



ルーチン `ungrq` のインターフェイスの詳細を以下に示す。

`a`                      サイズ  $(m, n)$  の行列  $A$  を格納する。

`tau`                     長さ  $(k)$  のベクトルを格納する。

### アプリケーション・ノート

パフォーマンスを改善するには、`lwork = m * blocksize` に設定する。`blocksize` は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。必要なワークスペースの大きさがわからない場合は、最初の実行では `lwork` を大きめの値に設定する。終了時に、`work(1)` の値を調べ、以降はその値を使用する。

このルーチンの実数版は、[?orgrq](#) である。

---

## ?ormrq

実行列に `?gerqf` で求めた  $RQ$  因子分解の直交行列  $Q$  を掛ける。

---

### 構文

#### Fortran 77:

```
call sormrq(side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork, info)
call dormrq(side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork, info)
```

#### Fortran 95:

```
call ormrmq(a, tau, c [,side] [,trans] [,info])
```

### 説明

このルーチンは、 $m \times n$  の実行列  $C$  に  $Q$  または  $Q^T$  を掛ける。 $Q$  は、 $RQ$  因子分解ルーチン [sgerqf/dgerqf](#) から返された  $k$  個の基本リフレクタ  $H_1$  の積で定義される直交行列である ( $Q = H_1 H_2 \cdots H_k$ )。

このルーチンを使用すれば、パラメータ `side` と `trans` の値に従って、 $QC$ 、 $Q^T C$ 、 $CQ$ 、または  $CQ^T$  のいずれかの行列積を求めることができる ( $C$  の値は上書きされる)。

## 入力パラメータ

<i>side</i>	CHARACTER*1。'L' または 'R' でなければならない。 <i>side</i> = 'L' の場合、 $Q$ または $Q^T$ は、 $C$ に左側から適用される。 <i>side</i> = 'R' の場合、 $Q$ または $Q^T$ は、 $C$ に右側から適用される。
<i>trans</i>	CHARACTER*1。'N' または 'T' でなければならない。 <i>trans</i> = 'N' の場合、 $C$ に $Q$ を掛ける。 <i>trans</i> = 'T' の場合、 $C$ に $Q^T$ を掛ける。
<i>m</i>	INTEGER。行列 $C$ の行数 ( $m \geq 0$ )。
<i>n</i>	INTEGER。行列 $C$ の列数 ( $n \geq 0$ )。
<i>k</i>	INTEGER。その積が行列 $Q$ となる基本リフレクタの個数。次の制約がある。 $0 \leq k \leq m$ ( <i>side</i> = 'L' の場合)。 $0 \leq k \leq n$ ( <i>side</i> = 'R' の場合)。
<i>a, tau, c, work</i>	REAL (sormrq の場合) DOUBLE PRECISION (dormrq の場合)。 配列: <i>a</i> ( <i>lda</i> , *), <i>tau</i> (*), <i>c</i> ( <i>ldc</i> , *), <i>work</i> ( <i>lwork</i> )  呼び出し時に、sgerqf/dgerqf で配列引数 <i>a</i> の最終 <i>k</i> 行に返されたとおりに、 <i>a</i> の <i>i</i> 番目の行に、基本リフレクタ $H_i$ (ただし、 $i = 1, 2, \dots, k$ ) を定義するベクトルが格納されていなければならない。 <i>a</i> の第 2 次元は、 $\max(1, m)$ 以上 ( <i>side</i> = 'L' の場合) または $\max(1, n)$ 以上 ( <i>side</i> = 'R' の場合) でなければならない。  <i>tau</i> ( <i>i</i> ) には、sgerqf/dgerqf から返されたとおりに、基本リフレクタ $H_i$ のスカラー係数が格納されていなければならない。 <i>tau</i> の次元は、 $\max(1, k)$ 以上でなければならない。  <i>c</i> ( <i>ldc</i> , *) には、 $m \times n$ の行列 $C$ を格納する。 <i>c</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。  <i>work</i> ( <i>lwork</i> ) は、ワークスペース配列である。
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $lda \geq \max(1, k)$ 。
<i>ldc</i>	INTEGER。 <i>c</i> の第 1 次元。 $ldc \geq \max(1, m)$ 。
<i>lwork</i>	INTEGER。配列 <i>work</i> のサイズ。次の制約がある。 $lwork \geq \max(1, n)$ ( <i>side</i> = 'L' の場合)。 $lwork \geq \max(1, m)$ ( <i>side</i> = 'R' の場合)。 $lwork = -1$ の場合、ワークスペースのクエリとみなされ、ルーチンは

*work* 配列の最適サイズだけを計算し、その値を *work* 配列の最初のエントリとして返す。xerbla は *lwork* に関するエラー・メッセージを生成しない。

*lwork* の推奨値は、「アプリケーション・ノート」を参照。

### 出力パラメータ

<i>c</i>	( <i>side</i> と <i>trans</i> の値に従って) $QC$ 、 $Q^TC$ 、 $CQ$ 、または $CQ^T$ のいずれかの積によって上書きされる。
<i>work</i> (1)	<i>info</i> = 0 の場合、終了時に、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が <i>work</i> (1) に格納される。以降の実行では、この <i>lwork</i> 値を使用する。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正であったことを示す。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン ormrq のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ ( <i>k</i> , <i>m</i> ) の行列 <i>A</i> を格納する。
<i>tau</i>	長さ ( <i>k</i> ) のベクトルを格納する。
<i>c</i>	サイズ ( <i>m</i> , <i>n</i> ) の行列 <i>C</i> を格納する。
<i>side</i>	'L' または 'R' でなければならない。デフォルト値は 'L'。
<i>trans</i>	'N' または 'T' でなければならない。デフォルト値は 'N'。

### アプリケーション・ノート

パフォーマンスを改善するには、 $lwork = n * blocksize$  (*side* = 'L' の場合) または  $lwork = m * blocksize$  (*side* = 'R' の場合) に設定する。*blocksize* は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。必要なワークスペースの大きさがわからない場合は、最初の実行では *lwork* を大きめの値に設定する。終了時に、*work*(1) の値を調べ、以降はその値を使用する。

このルーチンの複素数版は、[?unmrq](#) である。

## ?unmrq

複素行列に?gerqf で求めた  $RQ$  因子分解のユニタリ行列  $Q$  を掛ける。

### 構文

#### Fortran 77:

```
call cunmrq(side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork, info)
call zunmrq(side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork, info)
```

#### Fortran 95:

```
call unmrq(a, tau, c [,side] [,trans] [,info])
```

### 説明

このルーチンは、 $m \times n$  の複素行列  $C$  に  $Q$  または  $Q^H$  を掛ける。 $Q$  は、 $RQ$  因子分解ルーチン [cgerqf/zgerqf](#) から返された  $k$  個の基本リフレクタ  $H_i$  の積で定義される複素ユニタリ行列である ( $Q = H_1^H H_2^H \dots H_k^H$ )。

このルーチンを使用すれば、パラメータ *side* と *trans* の値に従って、 $QC$ 、 $Q^H C$ 、 $CQ$ 、または  $CQ^H$  のいずれかの行列積を求めることができる ( $C$  の値は上書きされる)。

### 入力パラメータ

<i>side</i>	CHARACTER*1。'L' または 'R' でなければならない。 <i>side</i> = 'L' の場合、 $Q$ または $Q^H$ は、 $C$ に左側から適用される。 <i>side</i> = 'R' の場合、 $Q$ または $Q^H$ は、 $C$ に右側から適用される。
<i>trans</i>	CHARACTER*1。'N' または 'C' でなければならない。 <i>trans</i> = 'N' の場合、 $C$ に $Q$ を掛ける。 <i>trans</i> = 'C' の場合、 $C$ に $Q^H$ を掛ける。
<i>m</i>	INTEGER。行列 $C$ の行数 ( $m \geq 0$ )。
<i>n</i>	INTEGER。行列 $C$ の列数 ( $n \geq 0$ )。
<i>k</i>	INTEGER。その積が行列 $Q$ となる基本リフレクタの個数。次の制約がある。 $0 \leq k \leq m$ ( <i>side</i> = 'L' の場合)。 $0 \leq k \leq n$ ( <i>side</i> = 'R' の場合)。

*a, tau, c, work*    COMPLEX (cunmrq の場合)  
                       DOUBLE COMPLEX (zunmrq の場合)。  
 配列: *a(lda,\*)*, *tau(\*)*, *c ldc,\*)*, *work(lwork)*

呼び出し時に、cgerqf/zgerqf で配列引数 *a* の最終 *k* 行に返されたとおりに、*a* の *i* 番目の行に、基本リフレクタ  $H_i$  (ただし、 $i = 1, 2, \dots, k$ ) を定義するベクトルが格納されていなければならない。  
*a* の第 2 次元は、 $\max(1, m)$  以上 (*side* = 'L' の場合) または  $\max(1, n)$  以上 (*side* = 'R' の場合) でなければならない。

*tau(i)* には、cgerqf/zgerqf から返されたとおりに、基本リフレクタ  $H_i$  のスカラー係数が格納されていなければならない。  
*tau* の次元は、 $\max(1, k)$  以上でなければならない。

*c ldc,\*)* には、 $m \times n$  の行列 *C* を格納する。  
*c* の第 2 次元は、 $\max(1, n)$  以上でなければならない。

*work(lwork)* は、ワークスペース配列である。

*lda*                INTEGER。 *a* の第 1 次元。  $lda \geq \max(1, k)$ 。

*ldc*                INTEGER。 *c* の第 1 次元。  $ldc \geq \max(1, m)$ 。

*lwork*             INTEGER。 配列 *work* のサイズ。次の制約がある。  
 $lwork \geq \max(1, n)$  (*side* = 'L' の場合)。  
 $lwork \geq \max(1, m)$  (*side* = 'R' の場合)。  
*lwork* = -1 の場合、ワークスペースのクエリとみなされ、ルーチンは *work* 配列の最適サイズだけを計算し、その値を *work* 配列の最初のエントリとして返す。xerbla は *lwork* に関するエラー・メッセージを生成しない。

*lwork* の推奨値は、「アプリケーション・ノート」を参照。

### 出力パラメータ

*c*                    (*side* と *trans* の値に従って)  $QC$ 、 $Q^H C$ 、 $CQ$ 、または  $CQ^H$  のいずれかの積によって上書きされる。

*work(1)*            *info* = 0 の場合、終了時に、最適なパフォーマンスを得るために必要な *lwork* の最小値が *work(1)* に格納される。以降の実行では、この *lwork* 値を使用する。

*info*                INTEGER。  
*info* = 0 の場合、実行は正常に終了したことを示す。  
*info* = -*i* の場合、*i* 番目のパラメータの値が不正であったことを示す。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `unmrq` のインターフェイスの詳細を以下に示す。

<code>a</code>	サイズ $(k,m)$ の行列 $A$ を格納する。
<code>tau</code>	長さ $(k)$ のベクトルを格納する。
<code>c</code>	サイズ $(m,n)$ の行列 $C$ を格納する。
<code>side</code>	'L' または 'R' でなければならない。デフォルト値は 'L'。
<code>trans</code>	'N' または 'C' でなければならない。デフォルト値は 'N'。

### アプリケーション・ノート

パフォーマンスを改善するには、`lwork = n*blocksize` (`side = 'L'` の場合) または `lwork = m*blocksize` (`side = 'R'` の場合) に設定する。`blocksize` は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。必要なワークスペースの大きさがわからない場合は、最初の実行では `lwork` を大きめの値に設定する。終了時に、`work(1)` の値を調べ、以降はその値を使用する。

このルーチンの実数版は、[?ormrq](#) である。

---

## ?tzrzf

上台形行列  $A$  を上三角形式に縮退させる。

---

### 構文

#### Fortran 77:

```
call stzrzf(m, n, a, lda, tau, work, lwork, info)
call dtzrzf(m, n, a, lda, tau, work, lwork, info)
call ctzrzf(m, n, a, lda, tau, work, lwork, info)
call ztzrzf(m, n, a, lda, tau, work, lwork, info)
```

#### Fortran 95:

```
call tzrzf(a [,tau] [,info])
```

**説明**

このルーチンは、直交 / ユニタリ変換を用いて、 $m \times n$  ( $m \leq n$ ) の実 / 複素上台形行列  $A$  を、上三角形式に縮退させる。上台形行列  $A$  は、次のように因子分解される。

$$A = (R \ 0) * Z$$

$Z$  は  $n \times n$  の直交 / ユニタリ行列、 $R$  は  $m \times m$  の上三角行列である。

?tzrzf が返したとおりの基本リフレクタを一般行列に適用する。[?larz](#) を参照。

**入力パラメータ**

$m$  INTEGER。行列  $A$  の行数 ( $m \geq 0$ )。

$n$  INTEGER。行列  $A$  の列数 ( $n \geq m$ )。

$a, work$  REAL (stzrzf の場合 )  
DOUBLE PRECISION (dtzrzf の場合 )  
COMPLEX (ctzrzf の場合 )  
DOUBLE COMPLEX (ztzrzf の場合 )。  
配列:  $a(lda, *)$ ,  $work(lwork)$ 。  
配列  $a$  の先頭の  $m \times n$  の上台形部分に、因子分解する行列  $A$  を格納する。  
 $a$  の第 2 次元は、 $\max(1, n)$  以上でなければならない。  
 $work$  は、ワークスペース配列である。

$lda$  INTEGER。  $a$  の第 1 次元。  $\max(1, m)$  以上。

$lwork$  INTEGER。配列  $work$  のサイズ。  
 $lwork \geq \max(1, m)$ 。  
 $lwork = -1$  の場合、ワークスペースのクエリとみなされ、ルーチンは  $work$  配列の最適サイズだけを計算し、その値を  $work$  配列の最初のエントリとして返す。xerbla は  $lwork$  に関するエラー・メッセージを生成しない。  
 $lwork$  の推奨値は、「アプリケーション・ノート」を参照。

**出力パラメータ**

$a$  終了時に、因子分解データによって次のように上書きされる。  
 $a$  の先頭の  $m \times m$  の上三角部分に、上三角行列  $R$  が格納される。 $a$  の最初の  $m$  行の  $m+1$  から  $n$  までの成分は、配列  $tau$  とともに、基本リフレクタ  $m$  の積として直交行列  $Z$  を表現する。

<code>tau</code>	REAL (stzrzf の場合 ) DOUBLE PRECISION (dtzrzf の場合 ) COMPLEX (ctzrzf の場合 ) DOUBLE COMPLEX (ztzrzf の場合 )。 配列、次元は $\max(1, m)$ 以上。 行列 $Z$ に対する基本リフレクタのスカラ係数が格納される。
<code>work(1)</code>	<code>info = 0</code> の場合、終了時に、最適なパフォーマンスを得るために必要な <code>lwork</code> の最小値が <code>work(1)</code> に格納される。以降の実行では、この <code>lwork</code> 値を使用する。
<code>info</code>	INTEGER。 <code>info = 0</code> の場合、実行は正常に終了したことを示す。 <code>info = -i</code> の場合、 $i$ 番目のパラメータの値が不正であったことを示す。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `tzrzf` のインターフェイスの詳細を以下に示す。

<code>a</code>	サイズ $(m, n)$ の行列 $A$ を格納する。
<code>tau</code>	長さ $(m)$ のベクトルを格納する。

### アプリケーション・ノート

パフォーマンスを改善するには、`lwork = m * blocksize` に設定する。`blocksize` は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。

必要なワークスペースの大きさがわからない場合は、最初の実行では `lwork` を大きめの値に設定する。終了時に、`work(1)` の値を調べ、以降はその値を使用する。

関連ルーチン:

<a href="#">?ormrz</a>	行列 $Q$ を適用する (実行列の場合)。
<a href="#">?unmrz</a>	行列 $Q$ を適用する (複素行列の場合)。



## ?ormrz

実行列に ?tzzrzf で求めた直交行列を掛ける。

### 構文

#### Fortran 77:

```
call sormrz(side, trans, m, n, k, l, a, lda, tau, c, ldc, work, lwork, info)
call dormrz(side, trans, m, n, k, l, a, lda, tau, c, ldc, work, lwork, info)
```

#### Fortran 95:

```
call ormz(a, tau, c, l [,side] [,trans] [,info])
```

### 説明

このルーチンは、 $m \times n$  の実行列  $C$  に  $Q$  または  $Q^T$  を掛ける。 $Q$  は、ルーチン [stzrzf/dtzrzf](#) から返された  $k$  個の基本リフレクタ  $H_i$  の積で定義される直交実行列である ( $Q = H_1 H_2 \cdots H_k$ )。

このルーチンを使用すれば、パラメータ  $side$  と  $trans$  の値に従って、 $QC$ 、 $Q^T C$ 、 $CQ$ 、または  $CQ^T$  のいずれかの行列積を求めることができる ( $C$  の値は上書きされる)。

行列  $Q$  の次数は、 $m$  ( $side = 'L'$  の場合) または  $n$  ( $side = 'R'$  の場合)。

### 入力パラメータ

$side$	CHARACTER*1。'L' または 'R' でなければならない。 $side = 'L'$ の場合、 $Q$ または $Q^T$ は、 $C$ に左側から適用される。 $side = 'R'$ の場合、 $Q$ または $Q^T$ は、 $C$ に右側から適用される。
$trans$	CHARACTER*1。'N' または 'T' でなければならない。 $trans = 'N'$ の場合、 $C$ に $Q$ を掛ける。 $trans = 'T'$ の場合、 $C$ に $Q^T$ を掛ける。
$m$	INTEGER。行列 $C$ の行数 ( $m \geq 0$ )。
$n$	INTEGER。行列 $C$ の列数 ( $n \geq 0$ )。
$k$	INTEGER。その積が行列 $Q$ となる基本リフレクタの個数。次の制約がある。 $0 \leq k \leq m$ ( $side = 'L'$ の場合)。 $0 \leq k \leq n$ ( $side = 'R'$ の場合)。

<i>l</i>	<p>INTEGER。</p> <p>Householder リフレクタとして意味を持った部分が格納されている行列 <i>A</i> の列数。次の制約がある。</p> <p><math>0 \leq l \leq m</math> (<i>side</i> = 'L' の場合)。</p> <p><math>0 \leq l \leq n</math> (<i>side</i> = 'R' の場合)。</p>
<i>a, tau, c, work</i>	<p>REAL (<i>sormrz</i> の場合)</p> <p>DOUBLE PRECISION (<i>dormrz</i> の場合)。</p> <p>配列: <i>a(lda,*)</i>, <i>tau(*)</i>, <i>c(ldc,*)</i>, <i>work(lwork)</i></p> <p>呼び出し時に、<i>stzrzf/dtzrzf</i> で配列引数 <i>a</i> の最終 <i>k</i> 行に返されたとおりに、<i>a</i> の <i>i</i> 番目の行に、基本リフレクタ <math>H_i</math> (ただし、<math>i = 1, 2, \dots, k</math>) を定義するベクトルが格納されていなければならない。</p> <p><i>a</i> の第 2 次元は、<math>\max(1, m)</math> 以上 (<i>side</i> = 'L' の場合) または <math>\max(1, n)</math> 以上 (<i>side</i> = 'R' の場合) でなければならない。</p> <p><i>tau(i)</i> には、<i>stzrzf/dtzrzf</i> から返されたとおりに、基本リフレクタ <math>H_i</math> のスカラー係数が格納されていなければならない。</p> <p><i>tau</i> の次元は、<math>\max(1, k)</math> 以上でなければならない。</p> <p><i>c(ldc,*)</i> には、<math>m \times n</math> の行列 <i>C</i> を格納する。</p> <p><i>c</i> の第 2 次元は、<math>\max(1, n)</math> 以上でなければならない。</p> <p><i>work(lwork)</i> は、ワークスペース配列である。</p>
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $lda \geq \max(1, k)$ 。
<i>ldc</i>	INTEGER。 <i>c</i> の第 1 次元。 $ldc \geq \max(1, m)$ 。
<i>lwork</i>	<p>INTEGER。 配列 <i>work</i> のサイズ。 次の制約がある。</p> <p><math>lwork \geq \max(1, n)</math> (<i>side</i> = 'L' の場合)。</p> <p><math>lwork \geq \max(1, m)</math> (<i>side</i> = 'R' の場合)。</p> <p><i>lwork</i> = -1 の場合、ワークスペースのクエリとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリとして返す。 <i>xerbla</i> は <i>lwork</i> に関するエラー・メッセージを生成しない。</p> <p><i>lwork</i> の推奨値は、「アプリケーション・ノート」を参照。</p>

## 出力パラメータ

<i>c</i>	( <i>side</i> と <i>trans</i> の値に従って) $QC$ 、 $Q^T C$ 、 $CQ$ 、または $CQ^T$ のいずれかの積によって上書きされる。
----------	---

<i>work(1)</i>	<i>info</i> = 0 の場合、終了時に、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が <i>work(1)</i> に格納される。以降の実行では、この <i>lwork</i> 値を使用する。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正であったことを示す。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `ormrz` のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ ( <i>k</i> , <i>m</i> ) の行列 <i>A</i> を格納する。
<i>tau</i>	長さ ( <i>k</i> ) のベクトルを格納する。
<i>c</i>	サイズ ( <i>m</i> , <i>n</i> ) の行列 <i>C</i> を格納する。
<i>side</i>	'L' または 'R' でなければならない。デフォルト値は 'L'。
<i>trans</i>	'N' または 'T' でなければならない。デフォルト値は 'N'。

### アプリケーション・ノート

パフォーマンスを改善するには、 $lwork = n * blocksize$  (*side* = 'L' の場合) または  $lwork = m * blocksize$  (*side* = 'R' の場合) に設定する。*blocksize* は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。必要なワークスペースの大きさがわからない場合は、最初の実行では *lwork* を大きめの値に設定する。終了時に、*work(1)* の値を調べ、以降はその値を使用する。

このルーチンの複素数版は、[?unmrz](#) である。

## ?unmrz

複素行列に?tzrzf で求めた因子分解のユニタリ行列  $Q$  を掛ける。

### 構文

#### Fortran 77:

```
call cunmrz(side, trans, m, n, k, l, a, lda, tau, c, ldc, work, lwork, info)
call zunmrz(side, trans, m, n, k, l, a, lda, tau, c, ldc, work, lwork, info)
```

#### Fortran 95:

```
call unmrz(a, tau, c, l [,side] [,trans] [,info])
```

### 説明

このルーチンは、 $m \times n$  の複素行列  $C$  に  $Q$  または  $Q^H$  を掛ける。 $Q$  は、ルーチン [ctzrzf/ztzrzf](#) から返された  $k$  個の基本リフレクタ  $H_i$  の積で定義されるユニタリ行列である ( $Q = H_1^H H_2^H \dots H_k^H$ )。

このルーチンを使用すれば、パラメータ *side* と *trans* の値に従って、 $QC$ 、 $Q^H C$ 、 $CQ$ 、または  $CQ^H$  のいずれかの行列積を求めることができる ( $C$  の値は上書きされる)。

行列  $Q$  の次数は、 $m$  (*side* = 'L' の場合) または  $n$  (*side* = 'R' の場合)。

### 入力パラメータ

<i>side</i>	CHARACTER*1。'L' または 'R' でなければならない。 <i>side</i> = 'L' の場合、 $Q$ または $Q^H$ は、 $C$ に左側から適用される。 <i>side</i> = 'R' の場合、 $Q$ または $Q^H$ は、 $C$ に右側から適用される。
<i>trans</i>	CHARACTER*1。'N' または 'C' でなければならない。 <i>trans</i> = 'N' の場合、 $C$ に $Q$ を掛ける。 <i>trans</i> = 'C' の場合、 $C$ に $Q^H$ を掛ける。
<i>m</i>	INTEGER。行列 $C$ の行数 ( $m \geq 0$ )。
<i>n</i>	INTEGER。行列 $C$ の列数 ( $n \geq 0$ )。

$k$	<p>INTEGER。その積が行列 <math>Q</math> となる基本リフレクタの個数。次の制約がある。</p> <p><math>0 \leq k \leq m</math> (<math>side = 'L'</math> の場合)。</p> <p><math>0 \leq k \leq n</math> (<math>side = 'R'</math> の場合)。</p>
$l$	<p>INTEGER。</p> <p>Householder リフレクタとして意味を持った部分が格納されている行列 <math>A</math> の列数。次の制約がある。</p> <p><math>0 \leq l \leq m</math> (<math>side = 'L'</math> の場合)。</p> <p><math>0 \leq l \leq n</math> (<math>side = 'R'</math> の場合)。</p>
$a, \tau, c, work$	<p>COMPLEX (cunmrz の場合)</p> <p>DOUBLE COMPLEX (zunmrz の場合)。</p> <p>配列: <math>a(lda, *)</math>, <math>\tau(*)</math>, <math>c ldc, *)</math>, <math>work(lwork)</math></p> <p>呼び出し時に、ctzrzf/ztzrzf で配列引数 <math>a</math> の最終 <math>k</math> 行に返されたとおりに、<math>a</math> の <math>i</math> 番目の行に、基本リフレクタ <math>H_i</math> (ただし、<math>i = 1, 2, \dots, k</math>) を定義するベクトルが格納されていなければならない。</p> <p><math>a</math> の第 2 次元は、<math>\max(1, m)</math> 以上 (<math>side = 'L'</math> の場合) または <math>\max(1, n)</math> 以上 (<math>side = 'R'</math> の場合) でなければならない。</p> <p><math>\tau(i)</math> には、ctzrzf/ztzrzf から返されたとおりに、基本リフレクタ <math>H_i</math> のスカラー係数が格納されていなければならない。</p> <p><math>\tau</math> の次元は、<math>\max(1, k)</math> 以上でなければならない。</p> <p><math>c(ldc, *)</math> には、<math>m \times n</math> の行列 <math>C</math> を格納する。</p> <p><math>c</math> の第 2 次元は、<math>\max(1, n)</math> 以上でなければならない。</p> <p><math>work(lwork)</math> は、ワークスペース配列である。</p>
$lda$	<p>INTEGER。 <math>a</math> の第 1 次元。 <math>lda \geq \max(1, k)</math>。</p>
$ldc$	<p>INTEGER。 <math>c</math> の第 1 次元。 <math>ldc \geq \max(1, m)</math>。</p>
$lwork$	<p>INTEGER。配列 <math>work</math> のサイズ。次の制約がある。</p> <p><math>lwork \geq \max(1, n)</math> (<math>side = 'L'</math> の場合)。</p> <p><math>lwork \geq \max(1, m)</math> (<math>side = 'R'</math> の場合)。</p> <p><math>lwork = -1</math> の場合、ワークスペースのクエリとみなされ、ルーチンは <math>work</math> 配列の最適サイズだけを計算し、その値を <math>work</math> 配列の最初のエントリとして返す。xerbla は <math>lwork</math> に関するエラー・メッセージを生成しない。</p> <p><math>lwork</math> の推奨値は、「アプリケーション・ノート」を参照。</p>

## 出力パラメータ

<i>c</i>	( <i>side</i> と <i>trans</i> の値に従って) $QC$ 、 $Q^H C$ 、 $CQ$ 、または $CQ^H$ のいずれかの積によって上書きされる。
<i>work</i> (1)	<i>info</i> = 0 の場合、終了時に、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が <i>work</i> (1) に格納される。以降の実行では、この <i>lwork</i> 値を使用する。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正であったことを示す。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `unmrz` のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ ( <i>k</i> , <i>m</i> ) の行列 <i>A</i> を格納する。
<i>tau</i>	長さ ( <i>k</i> ) のベクトルを格納する。
<i>c</i>	サイズ ( <i>m</i> , <i>n</i> ) の行列 <i>C</i> を格納する。
<i>side</i>	'L' または 'R' でなければならない。デフォルト値は 'L'。
<i>trans</i>	'N' または 'C' でなければならない。デフォルト値は 'N'。

## アプリケーション・ノート

パフォーマンスを改善するには、 $lwork = n * blocksize$  (*side* = 'L' の場合) または  $lwork = m * blocksize$  (*side* = 'R' の場合) に設定する。*blocksize* は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。必要なワークスペースの大きさがわからない場合は、最初の実行では *lwork* を大きめの値に設定する。終了時に、*work*(1) の値を調べ、以降はその値を使用する。

このルーチンの実数版は、[?ormrz](#) である。

**?ggqrf**

2 つの行列に対して汎用 *QR* 因子分解を行う。

**構文****Fortran 77:**

```
call sggqrf(n, m, p, a, lda, taua, b, ldb, taub, work, lwork, info)
call dggqrf(n, m, p, a, lda, taua, b, ldb, taub, work, lwork, info)
call cggqrf(n, m, p, a, lda, taua, b, ldb, taub, work, lwork, info)
call zggqrf(n, m, p, a, lda, taua, b, ldb, taub, work, lwork, info)
```

**Fortran 95:**

```
call ggqrf(a, b [,taua] [,taub] [,info])
```

**説明**

このルーチンは、 $n \times m$  の行列  $A$  と  $n \times p$  の行列  $B$  に対して、 $A = QR$ ,  $B = QTZ$  で表される汎用 *QR* 因子分解を行う。 $Q$  は  $n \times n$  の直交 / ユニタリ行列、 $Z$  は  $p \times p$  の直交 / ユニタリ行列である。また、 $R$  と  $T$  は次のどちらかの形式である。

$$R = \begin{matrix} & & m \\ & m & \\ n-m & \begin{pmatrix} R_{11} \\ 0 \end{pmatrix} \end{matrix}, \quad (n \geq m \text{ の場合})$$

または

$$R = \begin{matrix} & n & m-n \\ n & (R_{11} \quad R_{12}) \end{matrix}, \quad (n < m \text{ の場合})$$

$R_{11}$  は上三角行列である。

$$T = \begin{matrix} & p-n & n \\ n & \begin{pmatrix} 0 & T_{12} \end{pmatrix} \end{matrix}, \quad (n \leq p \text{ の場合}), \text{ または}$$

$$T = \begin{matrix} & & p \\ n-p & & \begin{pmatrix} T_{11} \\ T_{21} \end{pmatrix} \\ & p & \end{matrix}, \quad (n > p \text{ の場合})$$

$T_{12}$  または  $T_{21}$  は  $p \times p$  の上三角行列である。

特に、行列  $B$  が正方かつ非特異の場合、 $A$  と  $B$  の  $GQR$  因子分解によって、 $B^{-1}A$  の  $QR$  因子分解が次のように得られる。

$$B^{-1}A = Z^H (T^{-1} R)$$

### 入力パラメータ

$n$	INTEGER。行列 $A$ と $B$ の行数 ( $n \geq 0$ )。
$m$	INTEGER。行列 $A$ の列数 ( $m \geq 0$ )。
$p$	INTEGER。行列 $B$ の列数 ( $p \geq 0$ )。
$a, b, work$	REAL (sggqrf の場合) DOUBLE PRECISION (dggqrf の場合) COMPLEX (cggqrf の場合) DOUBLE COMPLEX (zggqrf の場合)。 配列: $a(lda, *)$ には、行列 $A$ を格納する。 $a$ の第 2 次元は、 $\max(1, m)$ 以上でなければならない。 $b(l db, *)$ には、行列 $B$ を格納する。 $b$ の第 2 次元は、 $\max(1, p)$ 以上でなければならない。 $work(lwork)$ は、ワークスペース配列である。
$lda$	INTEGER。 $a$ の第 1 次元。 $\max(1, n)$ 以上。
$ldb$	INTEGER。 $b$ の第 1 次元。 $\max(1, n)$ 以上。
$lwork$	INTEGER。配列 $work$ のサイズ。 $\max(1, n, m, p)$ 以上でなければならない。 $lwork = -1$ の場合、ワークスペースのクエリとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエントリとして返す。xerbla は $lwork$ に関するエラー・メッセージを生成しない。 $lwork$ の推奨値は、「アプリケーション・ノート」を参照。



**出力パラメータ**

<i>a</i> , <i>b</i>	<p>因子分解データによって次のように上書きされる。</p> <p>終了時に、配列 <i>a</i> の対角成分とその上の成分は、<math>\min(n, m) \times m</math> の上台形行列 <i>R</i> (<math>n \geq m</math> の場合、<i>R</i> は上三角行列) によって上書きされる。対角成分よりも下の成分は、配列 <i>taua</i> とともに、<math>\min(n, m)</math> 個の基本リフレクタの積として直交/ユニタリ行列 <i>Q</i> を表現する。</p> <p><math>n \leq p</math> の場合、部分配列 <i>b</i>(1:<i>n</i>, <i>p</i>-<i>n</i>+1:<i>p</i>) の上三角部分に、<math>n \times n</math> の上三角行列 <i>T</i> が格納される。</p> <p><math>n &gt; p</math> の場合、(<i>n</i>-<i>p</i>) 番目の劣対角成分とその上の各成分に、<math>n \times p</math> の上台形行列 <i>T</i> が格納される。残りの各成分は、配列 <i>taub</i> とともに、基本リフレクタの積として直交/ユニタリ行列 <i>Z</i> を表現する。</p>
<i>taua</i> , <i>taub</i>	<p>REAL (sggqrf の場合)</p> <p>DOUBLE PRECISION (dggqrf の場合)</p> <p>COMPLEX (cggqrf の場合)</p> <p>DOUBLE COMPLEX (zggqrf の場合)。</p> <p>配列、次元は <math>\max(1, \min(n, m))</math> 以上 (<i>taua</i> の場合) または <math>\max(1, \min(n, p))</math> 以上 (<i>taub</i> の場合)。</p> <p>配列 <i>taua</i> には、直交/ユニタリ行列 <i>Q</i> を表す基本リフレクタのスカラー係数が格納される。</p> <p>配列 <i>taub</i> には、直交/ユニタリ行列 <i>Z</i> を表す基本リフレクタのスカラー係数が格納される。</p>
<i>work</i> (1)	<p><i>info</i> = 0 の場合、終了時に、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が <i>work</i>(1) に格納される。以降の実行では、この <i>lwork</i> 値を使用する。</p>
<i>info</i>	<p>INTEGER。</p> <p><i>info</i> = 0 の場合、実行は正常に終了したことを示す。</p> <p><i>info</i> = -<i>i</i> の場合、<i>i</i> 番目のパラメータの値が不正であったことを示す。</p>

**Fortran 95 インターフェイス・ノート**

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン gggqrf のインターフェイスの詳細を以下に示す。

*a*                    サイズ (*n*,*m*) の行列 *A* を格納する。

<i>b</i>	サイズ $(n, p)$ の行列 <i>B</i> を格納する。
<i>taua</i>	長さ $\min(n, m)$ のベクトルを格納する。
<i>taub</i>	長さ $\min(n, p)$ のベクトルを格納する。

### アプリケーション・ノート

パフォーマンスを改善するには、*lwork* を次のように設定する。

$lwork \geq \max(n, m, p) * \max(nb1, nb2, nb3)$ 。

*nb1* は  $n \times m$  の行列の *QR* 因子分解における最適ブロックサイズ、*nb2* は  $n \times p$  の行列の *RQ* 因子分解における最適ブロックサイズ、*nb3* は [?ormqr](#)/[?unmqr](#) を呼び出すときの最適ブロックサイズである。

---

## ?ggrqf

2 つの行列に対して汎用 *RQ* 因子分解を行う。

---

### 構文

#### Fortran 77:

```
call sggrqf (m, p, n, a, lda, taua, b, ldb, taub, work, lwork, info)
call dggrqf (m, p, n, a, lda, taua, b, ldb, taub, work, lwork, info)
call cggrqf (m, p, n, a, lda, taua, b, ldb, taub, work, lwork, info)
call zggrqf (m, p, n, a, lda, taua, b, ldb, taub, work, lwork, info)
```

#### Fortran 95:

```
call ggrqf(a, b [,taua] [,taub] [,info])
```

### 説明

このルーチンは、 $m \times n$  の行列 *A* と  $p \times n$  の行列 *B* に対して、 $A = RQ$ 、 $B = ZTQ$  で表される汎用 *RQ* 因子分解を行う。*Q* は  $n \times n$  の直交 / ユニタリ行列、*Z* は  $p \times p$  の直交 / ユニタリ行列である。また、*R* と *T* は次のどちらかの形式である。

$$\begin{matrix} n-m & m \\ \hline \end{matrix}$$

$$R = \begin{pmatrix} m & 0 & R_{12} \end{pmatrix}, (m \leq n \text{ の場合})$$

または

$$R = \begin{pmatrix} m-n & n \\ R_{11} & \\ & R_{21} \end{pmatrix} \quad , (m > n \text{ の場合})$$

$R_{11}$  と  $R_{21}$  は上三角行列である。

$$T = \begin{pmatrix} n & n \\ T_{11} & \\ p-n & 0 \end{pmatrix} \quad , (p \geq n \text{ の場合})$$

または

$$T = \begin{pmatrix} p & n-p \\ T_{11} & T_{12} \end{pmatrix} \quad , (p < n \text{ の場合})$$

$T_{11}$  は上三角行列である。

特に、行列  $B$  が正方かつ非特異の場合、 $A$  と  $B$  の  $GRQ$  因子分解によって、 $AB^{-1}$  の  $RQ$  因子分解が次のように得られる。

$$AB^{-1} = (R \ T^{-1}) Z^H$$

### 入力パラメータ

$m$	INTEGER。行列 $A$ の行数 ( $m \geq 0$ )。
$p$	INTEGER。行列 $B$ の行数 ( $p \geq 0$ )。
$n$	INTEGER。行列 $A$ と $B$ の列数 ( $n \geq 0$ )。
$a, b, work$	REAL (sggrqf の場合) DOUBLE PRECISION (dggrqf の場合) COMPLEX (cggrqf の場合) DOUBLE COMPLEX (zggrqf の場合)。 配列： $a(lda,*)$ には、 $m \times n$ の行列 $A$ を格納する。 $a$ の第 2 次元は、 $\max(1, n)$ 以上でなければならない。

$b(l\delta b, *)$  には、 $p \times n$  の行列  $B$  を格納する。  
 $b$  の第 2 次元は、 $\max(1, n)$  以上でなければならない。

$work(lwork)$  は、ワークスペース配列である。

$lda$  INTEGER。  $a$  の第 1 次元。  $\max(1, m)$  以上。

$ldb$  INTEGER。  $b$  の第 1 次元。  $\max(1, p)$  以上。

$lwork$  INTEGER。 配列  $work$  のサイズ。  $\max(1, n, m, p)$  以上でなければならない。

$lwork = -1$  の場合、ワークスペースのクエリとみなされ、ルーチンは  $work$  配列の最適サイズだけを計算し、その値を  $work$  配列の最初のエントリとして返す。  $xerbla$  は  $lwork$  に関するエラー・メッセージを生成しない。

$lwork$  の推奨値は、「アプリケーション・ノート」を参照。

## 出力パラメータ

$a, b$  因子分解データによって次のように上書きされる。

終了時に、 $m \leq n$  の場合、部分配列  $a(1:m, n-m+1:n)$  の上三角部分は、 $m \times m$  の上三角行列  $R$  によって上書きされる。  
 $m > n$  の場合、 $(m-n)$  番目の劣対角成分とその上の成分は、 $m \times n$  の上台形行列  $R$  によって上書きされる。残りの各成分は、配列  $\tau_{aua}$  とともに、基本リフレクタの積として直交 / ユニタリ行列  $Q$  を表現する。  
配列  $b$  の対角成分とその上の成分は、 $\min(p, n) \times n$  の上台形行列  $T$  ( $p \geq n$  の場合、 $T$  は上三角行列) によって上書きされる。対角成分よりも下の成分は、配列  $\tau_{aub}$  とともに、基本リフレクタの積として直交 / ユニタリ行列  $Z$  を表現する。

$\tau_{aua}, \tau_{aub}$  REAL (sggrqf の場合)

DOUBLE PRECISION (dggrqf の場合)

COMPLEX (cggrqf の場合)

DOUBLE COMPLEX (zggrqf の場合)。

配列、次元は  $\max(1, \min(m, n))$  以上 ( $\tau_{aua}$  の場合) または

$\max(1, \min(p, n))$  以上 ( $\tau_{aub}$  の場合)。

配列  $\tau_{aua}$  には、直交 / ユニタリ行列  $Q$  を表す基本リフレクタのスカラー係数が格納される。

配列  $\tau_{aub}$  には、直交 / ユニタリ行列  $Z$  を表す基本リフレクタのスカラー係数が格納される。

<i>work</i> (1)	<i>info</i> = 0 の場合、終了時に、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が <i>work</i> (1) に格納される。以降の実行では、この <i>lwork</i> 値を使用する。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正であったことを示す。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *ggrqf* のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ ( <i>m</i> , <i>n</i> ) の行列 <i>A</i> を格納する。
<i>b</i>	サイズ ( <i>p</i> , <i>n</i> ) の行列 <i>A</i> を格納する。
<i>taua</i>	長さ $\min(m, n)$ のベクトルを格納する。
<i>taub</i>	長さ $\min(p, n)$ のベクトルを格納する。

### アプリケーション・ノート

パフォーマンスを改善するには、 $lwork \geq \max(n, m, p) * \max(nb1, nb2, nb3)$  のように設定する。

*nb1* は  $m \times n$  の行列の *RQ* 因子分解における最適なブロックサイズ、*nb2* は  $p \times n$  の行列の *QR* 因子分解における最適なブロックサイズ、*nb3* は *?ormrq*/*?unmrq* を呼び出すときの最適なブロックサイズである。

必要なワークスペースの大きさがわからない場合は、最初の実行では *lwork* を大きめの値に設定する。終了時に、*work*(1) の値を調べ、以降はその値を使用する。

## 特異値分解

このセクションでは、一般行列  $A$  ( $m \times n$ ) を次のように特異値分解(SVD)するための LAPACK ルーチンについて説明する。

$$A = U \Sigma V^H$$

この分解処理では、 $U$  と  $V$  は、ユニタリ行列 ( $A$  が複素行列の場合) または直交行列 ( $A$  が実行列の場合) である。 $\Sigma$  は、次に示す対角成分  $\sigma_i$  を持つ  $m \times n$  の対角行列である。

$$\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_{\min(m, n)} \geq 0$$

対角成分  $\sigma_i$  は、 $A$  の特異値である。行列  $U$  と  $V$  の先頭から  $\min(m, n)$  の列は、それぞれ、 $A$  の左特異ベクトルと右特異ベクトルである。対応する特異値と特異ベクトルは、次の条件を満たす。

$$A v_i = \sigma_i u_i \quad \text{かつ} \quad A^H u_i = \sigma_i v_i$$

$u_i$  と  $v_i$  は、それぞれ、 $U$  と  $V$  の  $i$  番目の成分である。

一般行列  $A$  の SVD を求めるには、まず、LAPACK ルーチンの ?gebrd または ?gbbbrd を呼び出し、ユニタリ (直交) 変換  $A = QBP^H$  によって  $A$  を二重対角行列  $B$  に縮退させる。次に ?bdsqr を呼び出し、二重対角行列  $B$  から  $B = U_1 \Sigma V_1^H$  となるような SVD を求める。

そのため、求めるべき  $A$  の SVD は、 $A = U \Sigma V^H = (QU_1) \Sigma (V_1^H P^H)$  のように表現できる。

表 4-2 に、行列の特異値分解を実行するための LAPACK ルーチン (Fortran-77 インターフェイス) を示す。Fortran-95 インターフェイスのルーチン名はそれぞれ、先頭の記号がない ([ルーチン命名規則](#) を参照)。

表 4-2 特異値分解 (SVD) 用のルーチン

演算	実行列	複素行列
$A$ を二重対角行列 $B$ に縮退させる。 $A = QBP^H$ (フル格納)	<a href="#">?gebrd</a>	<a href="#">?gebrd</a>
$A$ を二重対角行列 $B$ に縮退させる。 $A = QBP^H$ (帯格納)	<a href="#">?gbbbrd</a>	<a href="#">?gbbbrd</a>
直交 (ユニタリ) 行列 $Q$ または $P$ を生成する	<a href="#">?orgbr</a>	<a href="#">?ungbr</a>
直交 (ユニタリ) 行列 $Q$ または $P$ を適用する	<a href="#">?ormbr</a>	<a href="#">?unmbr</a>
二重対角行列 $B$ から次の特異値分解を行う。 $B = U \Sigma V^H$	<a href="#">?bdsqr</a> <a href="#">?bdsdc</a>	<a href="#">?bdsqr</a>

図 4-1 デシジョン・ツリー：特異値分解

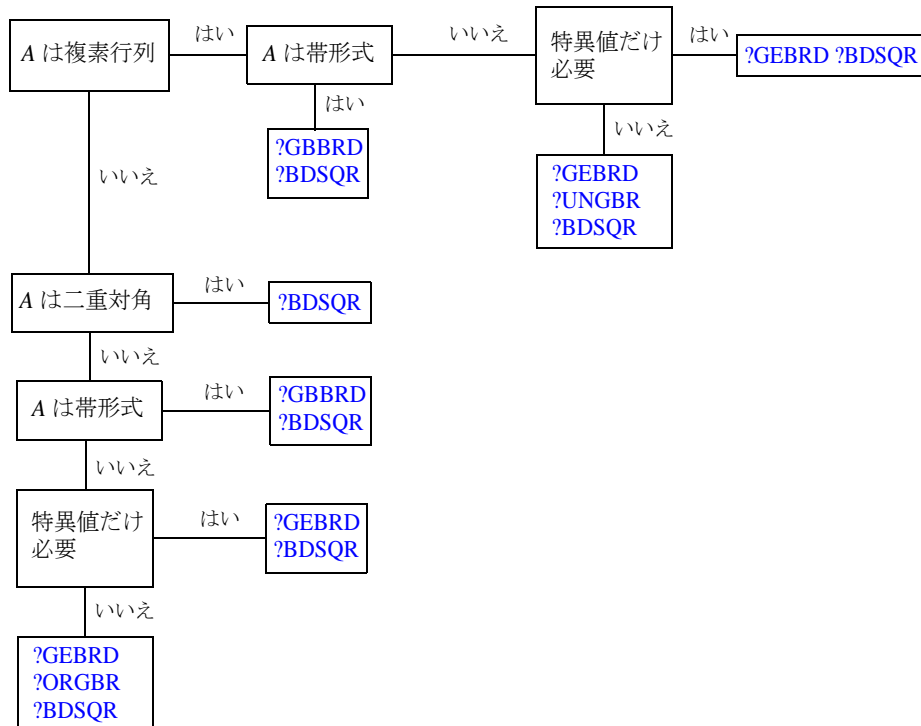


図 4-1 のデシジョン・ツリーを参考にすれば、特異値だけ必要な場合、特異値と特異ベクトルの両方が必要な場合、あるいは  $A$  が実数または複素数のどちらであるかなどといった条件に応じて、SVD を求めるための一連のルーチンを正しく選択できる。

SVD を使用すれば、 $\|Ax - b\|_2$  が最小になるような階数不足の最小二乗問題に対する最小ノルム解を得られる（可能な場合）。行列  $A$  の有効な階数  $k$  は、適切なしきい値を超えるような特異値の個数によって決定できる。最小ノルム解は、次のとおりである。

$$x = V_k(\Sigma_k)^{-1}c$$

$\Sigma_k$  は、 $\Sigma$  の先頭から  $k \times k$  の部分行列である。行列  $V_k$  は  $V = PV_1$  の先頭から  $k$  個の列で構成され、ベクトル  $c$  は  $U^H b = U_1^H Q^H b$  の先頭から  $k$  個の成分で構成される。

## ?gebrd

一般行列を二重対角形式に縮退させる。

---

### 構文

#### Fortran 77:

```
call sgebrd(m, n, a, lda, d, e, tauq, taup, work, lwork, info)
call dgebrd(m, n, a, lda, d, e, tauq, taup, work, lwork, info)
call cgebrd(m, n, a, lda, d, e, tauq, taup, work, lwork, info)
call zgebrd(m, n, a, lda, d, e, tauq, taup, work, lwork, info)
```

#### Fortran 95:

```
call gebrd(a [,d] [,e] [,tauq] [,taup] [,info])
```

### 説明

このルーチンは、直交 (ユニタリ) 変換によって、 $m \times n$  の一般行列  $A$  を二重対角行列  $B$  に縮退させる。

$m \geq n$  の場合、縮退は次の式で与えられる。 $A = QBP^H = Q \begin{pmatrix} B_1 \\ 0 \end{pmatrix} P^H = Q_1 B_1 P_1^H$ ,

$B_1$  は、 $n \times n$  の上三角行列である。 $Q$  と  $P$  は、直交行列またはユニタリ行列 ( $A$  が複素行列の場合) である。 $Q_1$  は、 $Q$  の先頭から  $n$  個の列によって構成される。

$m < n$  の場合、縮退は次の式で与えられる。

$$A = QBP^H = Q(B_1 0)P^H = Q_1 B_1 P_1^H,$$

$B_1$  は、 $m \times m$  の下対角行列である。 $Q$  と  $P$  は、直交行列またはユニタリ行列 ( $A$  が複素行列の場合) である。 $P_1$  は、 $P$  の先頭から  $m$  個の行によって構成される。

このルーチンでは、行列  $Q$  と  $P$  を明示的な形式では表現せず、基本リフレクタの積として表現する。一連のルーチンでは、この形式で表現される行列  $Q$  と  $P$  を操作する。

行列  $A$  が実数型の場合、

- $Q$  と  $P$  を明示的に求めるには、[?orgbr](#) を呼び出す。
- 一般行列に  $Q$  または  $P$  を掛けるには、[?ormbr](#) を呼び出す。

行列  $A$  が複素数型の場合、



- $Q$  と  $P$  を明示的に求めるには、[?ungbr](#) を呼び出す。
- 一般行列に  $Q$  または  $P$  を掛けるには、[?unmbr](#) を呼び出す。

### 入力パラメータ

$m$	INTEGER。行列 $A$ の行数 ( $m \geq 0$ )。
$n$	INTEGER。行列 $A$ の列数 ( $n \geq 0$ )。
$a, work$	REAL (sgebrd の場合) DOUBLE PRECISION (dgebrd の場合) COMPLEX (cgebrd の場合) DOUBLE COMPLEX (zgebrd の場合)。  配列： $a(lda, *)$ には、行列 $A$ を格納する。 $a$ の第 2 次元は、 $\max(1, n)$ 以上でなければならない。  $work(lwork)$ は、ワークスペース配列である。
$lda$	INTEGER。 $a$ の第 1 次元。 $\max(1, m)$ 以上。
$lwork$	INTEGER。 $work$ の次元。 $\max(1, m, n)$ 以上。 $lwork = -1$ の場合、ワークスペースのクエリとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエントリとして返す。xerbla は $lwork$ に関するエラー・メッセージを生成しない。  $lwork$ の推奨値は、「アプリケーション・ノート」を参照。

### 出力パラメータ

$a$	$m \geq n$ の場合、 $a$ の対角線と第 1 の優対角成分は、上二重対角行列 $B$ によって上書きされる。対角線より下の成分は $Q$ の各成分によって、残りの成分は $P$ の各成分によって上書きされる。  $m < n$ の場合、 $a$ の対角線と第 1 の劣対角成分は、下二重対角行列 $B$ によって上書きされる。対角線より上の成分は $P$ の各成分によって、残りの成分は $Q$ の各成分によって上書きされる。
$d$	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。配列、次元は $\max(1, \min(m, n))$ 以上。 $B$ の対角成分が格納される。

<i>e</i>	REAL ( 単精度の場合 ) DOUBLE PRECISION ( 倍精度の場合 )。 配列、次元は $\max(1, \min(m, n) - 1)$ 以上。 <i>B</i> の非対角成分が格納される。
<i>tauq, tauq</i>	REAL (sgebrd の場合 ) DOUBLE PRECISION (dgebrd の場合 ) COMPLEX (cgebrd の場合 ) DOUBLE COMPLEX (zgebrd の場合 )。 配列、次元は $(1, \min(m, n))$ 以上。 行列 <i>Q</i> と <i>P</i> の各成分が格納される。
<i>work(1)</i>	<i>info</i> = 0 の場合、終了時に、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が <i>work(1)</i> に格納される。以降の実行では、この <i>lwork</i> 値を使用する。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正であったことを示す。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *gebrd* のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ $(m, n)$ の行列 <i>A</i> を格納する。
<i>d</i>	長さ $\min(m, n)$ のベクトルを格納する。
<i>e</i>	長さ $\min(m, n) - 1$ のベクトルを格納する。
<i>tauq</i>	長さ $\min(m, n)$ のベクトルを格納する。
<i>taup</i>	長さ $\min(m, n)$ のベクトルを格納する。

## アプリケーション・ノート

パフォーマンスを改善するには、 $lwork = (m + n) * blocksize$  に設定する。*blocksize* は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる ( 通常は 16 ~ 64 )。

必要なワークスペースの大きさがわからない場合は、最初の実行では `lwork` を大きめの値に設定する。終了時に、`work(1)` の値を調べ、以降はその値を使用する。

計算で求めた行列  $Q$ 、 $B$ 、 $P$  は、 $QBP^H = A + E$  を満たす。

$\|E\|_2 = c(n)\varepsilon \|A\|_2$  であり、 $c(n)$  は漸増する  $n$  の関数で、 $\varepsilon$  はマシンによって異なる値である。

実数型の浮動小数点演算のおおよその回数は次のとおりである。

$(4/3) \cdot n^2 \cdot (3 \cdot m - n)$  ( $m \geq n$  の場合) 、

$(4/3) \cdot m^2 \cdot (3 \cdot n - m)$  ( $m < n$  の場合) 。

複素数型の場合、この 4 倍になる。

$n$  が  $m$  に比べてはるかに小さい場合は、まず [?gqgrf](#) を呼び出して  $A$  の  $QR$  因子分解を求めた後、因数  $R$  を二重対角形式に縮退した方が効率が良くなる。

その場合、約  $2 \cdot n^2 \cdot (m + n)$  回の浮動小数演算が必要になる。

一方、 $m$  が  $n$  に比べてはるかに小さい場合は、まず [?gelqf](#) を呼び出して  $A$  の  $LQ$  因子分解を求めた後、因数  $L$  を二重対角形式に縮退した方が効率が良くなる。

その場合、約  $2 \cdot m^2 \cdot (m + n)$  回の浮動小数演算が必要になる。

---

## ?gbbbrd

一般帯行列を二重対角形式に縮退させる。

---

### 構文

#### Fortran 77:

```
call sgbbrd(vect, m, n, ncc, kl, ku, ab, ldab, d, e, q, ldq, pt,
           ldpt, c, ldc, work, info)
call dgbbrd(vect, m, n, ncc, kl, ku, ab, ldab, d, e, q, ldq, pt,
           ldpt, c, ldc, work, info)
call cgbbrd(vect, m, n, ncc, kl, ku, ab, ldab, d, e, q, ldq, pt,
           ldpt, c, ldc, work, rwork, info)
call zgbbrd(vect, m, n, ncc, kl, ku, ab, ldab, d, e, q, ldq, pt,
           ldpt, c, ldc, work, rwork, info)
```

#### Fortran 95:

```
call gbbbrd(a [,c] [,d] [,e] [,q] [,pt] [,kl] [,m] [,info])
```

## 説明

このルーチンは、 $m \times n$  の帯行列  $A$  を  $A = QBP^H$  に縮退させ、上二重対角行列  $B$  を求める。行列  $Q$  と  $P$  は、直交行列 ( $A$  が実数の場合) またはユニタリ行列 ( $A$  が複素数の場合) である。これらは、Givens 回転行列の積として得られる。そのため、必要に応じて、このルーチンを使って明示的に求めることができる。このルーチンでは、 $C = Q^H C$  として行列  $C$  の更新もできる。

## 入カパラメータ

**vect** CHARACTER\*1。'N'、'Q'、'P'、または 'B' のいずれかでなければならない。  
**vect** = 'N' の場合、 $Q$  も  $P^H$  も生成されない。  
**vect** = 'Q' の場合、行列  $Q$  が生成される。  
**vect** = 'P' の場合、行列  $P^H$  が生成される。  
**vect** = 'B' の場合、 $Q$  も  $P^H$  も生成される。

**m** INTEGER。行列  $A$  の行数 ( $m \geq 0$ )。

**n** INTEGER。行列  $A$  の列数 ( $n \geq 0$ )。

**ncc** INTEGER。行列  $C$  の列数 ( $ncc \geq 0$ )。

**k1** INTEGER。  $A$  の帯内の劣対角成分の数 ( $k1 \geq 0$ )。

**ku** INTEGER。  $A$  の帯内の優対角成分の数 ( $ku \geq 0$ )。

**ab,c,work** REAL (sgbbrd の場合)  
 DOUBLE PRECISION (dgbbrd の場合)  
 COMPLEX (cgbbrd の場合)  
 DOUBLE COMPLEX (zgbbrd の場合)。  
 配列:  
**ab(ldab,\*)** には、帯格納形式の行列  $A$  ([行列の格納形式](#)を参照) を格納する。  
**a** の第 2 次元は、 $\max(1, n)$  以上でなければならない。  
**c ldc, \*** には、 $m \times ncc$  の行列  $C$  を格納する。  
**ncc = 0** の場合、配列 **c** は参照されない。**c** の第 2 次元は、 $\max(1, ncc)$  以上でなければならない。  
**work(\*)** は、ワークスペース配列である。  
**work** の次元は、 $2 * \max(m, n)$  以上 (実数型の場合) または  $\max(m, n)$  以上 (複素数型の場合) でなければならない。

**ldab** INTEGER。配列 **ab** の第 1 次元。 ( $ldab \geq k1 + ku + 1$ )。

<i>ldq</i>	INTEGER。出力配列 <i>q</i> の第 1 次元。 <i>vect</i> = 'Q' または 'B' の場合、 $ldq \geq \max(1, m)$ そうでない場合、 $ldq \geq 1$ 。
<i>ldpt</i>	INTEGER。出力配列 <i>pt</i> の第 1 次元。 <i>vect</i> = 'P' または 'B' の場合、 $ldpt \geq \max(1, n)$ そうでない場合、 $ldpt \geq 1$ 。
<i>ldc</i>	INTEGER。配列 <i>c</i> の第 1 次元。 $ldc \geq \max(1, m)$ ( $ncc > 0$ の場合)、 $ldc \geq 1$ ( $ncc = 0$ の場合)。
<i>rwork</i>	REAL (cgbbbrd の場合) DOUBLE PRECISION (zgbbrd の場合)。 ワークスペース配列、次元は $\max(m, n)$ 以上。

### 出力パラメータ

<i>ab</i>	縮退中に生成された値によって上書きされる。
<i>d</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 配列、次元は $\max(1, \min(m, n))$ 以上。 行列 <i>B</i> の対角成分が格納される。
<i>e</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 配列、次元は $\max(1, \min(m, n) - 1)$ 以上。 <i>B</i> の非対角成分が格納される。
<i>q, pt</i>	REAL (sgebrd の場合) DOUBLE PRECISION (dgebrd の場合) COMPLEX (cgebrd の場合) DOUBLE COMPLEX (zgebrd の場合)。 配列：  <i>q</i> ( <i>ldq</i> ,*) には、 $m \times m$ の出力行列 <i>Q</i> が格納される。 <i>q</i> の第 2 次元は、 $\max(1, m)$ 以上でなければならない。  <i>p</i> ( <i>ldpt</i> ,*) には、 $n \times n$ の出力行列 <i>P<sup>H</sup></i> が格納される。 <i>pt</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正であったことを示す。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `gbbrd` のインターフェイスの詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ab</i> を意味する。 サイズ $(k1+ku+1, n)$ の配列 <i>A</i> を格納する。
<i>c</i>	サイズ $(m, ncc)$ の行列 <i>C</i> を格納する。
<i>d</i>	長さ $\min(m, n)$ のベクトルを格納する。
<i>e</i>	長さ $\min(m, n)-1$ のベクトルを格納する。
<i>q</i>	サイズ $(m, m)$ の行列 <i>Q</i> を格納する。
<i>pt</i>	サイズ $(n, n)$ の行列 <i>PT</i> を格納する。
<i>m</i>	省略した場合、 $m = n$ とみなす。
<i>k1</i>	省略した場合、 $k1 = ku$ とみなす。
<i>ku</i>	$ku = lda - k1 - 1$ として復元する。
<i>vect</i>	引数 <i>q</i> および <i>pt</i> の存在に基づいて以下のように復元される。 <i>vect</i> = 'B' ( <i>q</i> と <i>pt</i> の両方が存在する場合)、 <i>vect</i> = 'Q' ( <i>q</i> が存在し、 <i>pt</i> が省略された場合)、 <i>vect</i> = 'P' ( <i>q</i> が省略され、 <i>pt</i> が存在する場合)、 <i>vect</i> = 'N' ( <i>q</i> と <i>pt</i> の両方が省略された場合)。

## アプリケーション・ノート

計算で求めた行列 *Q*、*B*、*P* は、 $QBP^H = A + E$  を満たす。

$\|E\|_2 = c(n)\varepsilon \|A\|_2$  であり、 $c(n)$  は漸増する  $n$  の関数で、 $\varepsilon$  はマシンによって異なる値である。

$m = n$  の場合、実数型の場合の浮動小数演算のおおよその総数は、次の合計になる。

$6 * n^2 * (k1 + ku)$	( <i>vect</i> = 'N' かつ $ncc = 0$ の場合)
$3 * n^2 * ncc * (k1 + ku - 1) / (k1 + ku)$	( <i>C</i> を更新する場合)
$3 * n^3 * (k1 + ku - 1) / (k1 + ku)$	( <i>Q</i> または <i>PH</i> を生成する場合) (両方とも生成する場合は、2 倍の値になる)。

複素数型の場合の演算回数を見積もる場合は、同じ式で係数を (6 と 3 ではなく) 20 と 10 とする。

---

## ?orgbr

?gebrd で求めた実直交行列  $Q$  または  $P^T$  を生成する。

---

### 構文

#### Fortran 77:

```
call sorgbr(vect, m, n, k, a, lda, tau, work, lwork, info)
call dorgbr(vect, m, n, k, a, lda, tau, work, lwork, info)
```

#### Fortran 95:

```
call orgbr(a, tau [,vect] [,info])
```

### 説明

このルーチンは、ルーチン [sgebrd/dgebrd](#) で求めた直交行列  $Q$  と  $P^T$  の全部または一部を生成する。このルーチンは、sgebrd/dgebrd を呼び出した後で使用する。引数の有効な組み合わせについては、入力パラメータを参照。ほとんどの場合、次のように呼び出す必要がある。

$m \times m$  の行列  $Q$  の全体を求めるには、  
call ?orgbr ( 'Q', m, m, n, a ... )  
(配列  $a$  には、列が  $m$  個以上なければならない)。

$Q$  の先頭から  $n$  列を求めるには ( $m > n$  の場合)、  
call ?orgbr ( 'Q', m, n, n, a ... )

$n \times n$  の行列  $P^T$  の全体を求めるには、  
call ?orgbr ( 'P', n, n, m, a ... )  
(配列  $a$  には、行が  $n$  個以上なければならない)。

$P^T$  の先頭から  $m$  行を求めるには ( $m < n$  の場合)、  
call ?orgbr ( 'P', m, n, m, a ... )

## 入力パラメータ

<code>vect</code>	CHARACTER*1。'Q' または 'P' でなければならない。 <code>vect = 'Q'</code> の場合、行列 $Q$ が生成される。 <code>vect = 'P'</code> の場合、行列 $P^T$ が生成される。
<code>m</code>	INTEGER。 $Q$ または $P^T$ に必要な行数。
<code>n</code>	INTEGER。 $Q$ または $P^T$ に必要な列数。
<code>k</code>	INTEGER。 ?gebrd から返された $A$ の次元のいずれか。 <code>vect = 'Q'</code> の場合、 $A$ の列数。 <code>vect = 'P'</code> の場合、 $A$ の行数。  次の制約がある。 $m \geq 0, n \geq 0, k \geq 0$ 。 <code>vect = 'Q'</code> の場合、 $k \leq n \leq m$ ( $m > k$ の場合) または $m = n$ ( $m \leq k$ の場合)。 <code>vect = 'P'</code> の場合、 $k \leq m \leq n$ ( $n > k$ の場合) または $m = n$ ( $n \leq k$ の場合)。
<code>a, work</code>	REAL (sorgbr の場合) DOUBLE PRECISION (dorgbr の場合)。 配列: <code>a(lda,*)</code> は、?gebrd から返された配列 <code>a</code> である。 <code>a</code> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。  <code>work(lwork)</code> は、ワークスペース配列である。
<code>lda</code>	INTEGER。 <code>a</code> の第 1 次元。 $\max(1, m)$ 以上。
<code>tau</code>	REAL (sorgbr の場合) DOUBLE PRECISION (dorgbr の場合)。 <code>vect = 'Q'</code> の場合、 <code>tauq</code> は ?gebrd から返された配列である。 <code>vect = 'P'</code> の場合、 <code>taup</code> は ?gebrd から返された配列である。 <code>tau</code> の次元は、 $\max(1, \min(m, k))$ 以上 ( <code>vect = 'Q'</code> の場合)、または $\max(1, \min(m, k))$ 以上 ( <code>vect = 'P'</code> の場合) でなければならない。
<code>lwork</code>	INTEGER。配列 <code>work</code> のサイズ。 <code>lwork = -1</code> の場合、ワークスペースのクエリとみなされ、ルーチンは <code>work</code> 配列の最適サイズだけを計算し、その値を <code>work</code> 配列の最初のエントリとして返す。xerbla は <code>lwork</code> に関するエラー・メッセージを生成しない。  <code>lwork</code> の推奨値は、「アプリケーション・ノート」を参照。



**出力パラメータ**

<i>a</i>	<i>vect</i> 、 <i>m</i> 、 <i>n</i> の値に従って、直交行列 <i>Q</i> または $P^T$ (あるいは対応する先頭からの一連の行または列) によって上書きされる。
<i>work</i> (1)	<i>info</i> = 0 の場合、終了時に、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が <i>work</i> (1) に格納される。以降の実行では、この <i>lwork</i> 値を使用する。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正であったことを示す。

**Fortran 95 インターフェイス・ノート**

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *orgbr* のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ ( <i>m</i> , <i>n</i> ) の行列 <i>A</i> を格納する。
<i>tau</i>	長さ $\min(m, k)$ のベクトルを格納する。ここで $k = m$ ( <i>vect</i> = 'P' の場合)、 $k = n$ ( <i>vect</i> = 'Q' の場合)。
<i>vect</i>	'Q' または 'P' でなければならない。デフォルト値は 'Q'。

**アプリケーション・ノート**

パフォーマンスを改善するには、 $lwork = \min(m, n) * blocksize$  に設定する。*blocksize* は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。必要なワークスペースの大きさがわからない場合は、最初の実行では *lwork* を大きめの値に設定する。終了時に、*work*(1) の値を調べ、以降はその値を使用する。計算で求めた行列 *Q* は、正確な直交行列と行列 *E* ( $\|E\|_2 = O(\epsilon)$ ) だけ異なる。

説明で示した場合ごとの浮動小数演算のおおよその総数は、次のようになる。

*Q* の全体を求める場合、

$$\begin{aligned} (4/3)n(3m^2 - 3m*n + n^2) & \quad (m > n \text{ の場合}), \\ (4/3)m^3 & \quad (m \leq n \text{ の場合}). \end{aligned}$$

$Q$  の先頭から  $n$  列を求める場合 ( $m > n$ )、

$$(2/3)n^2(3m - n^2) \quad (m > n \text{ の場合})。$$

$P^T$  の全体を求める場合、

$$(4/3)n^3 \quad (m \geq n \text{ の場合})、$$

$$(4/3)m(3n^2 - 3m*n + m^2) \quad (m < n \text{ の場合})。$$

$P^T$  の先頭から  $m$  列を求める場合 ( $m < n$ )、

$$(2/3)n^2(3m - n^2) \quad (m > n \text{ の場合})。$$

このルーチンの複素数版は、[?ungbr](#) である。

---

### ?ormbr

任意の実行列に ?gebrd で求めた実直交行列  $Q$   
または  $P^T$  を掛ける。

---

#### 構文

##### Fortran 77:

```
call sormbr(vect, side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork, info)
call dormbr(vect, side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork, info)
```

##### Fortran 95:

```
call ormbr(a, tau, c [,vect] [,side] [,trans] [,info])
```

#### 説明

このルーチンは、任意の実行列  $C$  に対して、 $QC$ 、 $Q^TC$ 、 $CQ$ 、 $CQ^T$ 、 $PC$ 、 $P^TC$ 、 $CP$ 、または  $CP^T$  のいずれかの行列積を生成する。 $Q$  と  $P$  は、ルーチン [sgebrd/dgebrd](#) を呼び出して求めた直交行列である。このルーチンを実行すると、 $C$  の積が上書きされる。

#### 入力パラメータ

下記の説明で、 $r$  は  $Q$  または  $P^T$  の次数を表す。  
 $side = 'L'$  の場合、 $r = m$ 、 $side = 'R'$  の場合、 $r = n$ 。

<i>vect</i>	CHARACTER*1。'Q' または 'P' でなければならない。 <i>vect</i> ='Q' の場合、 $Q$ または $Q^T$ が $C$ に適用される。 <i>vect</i> ='P' の場合、 $P$ または $P^T$ が $C$ に適用される。
<i>side</i>	CHARACTER*1。'L' または 'R' でなければならない。 <i>side</i> ='L' の場合、乗算は $C$ に左側から適用される。 <i>side</i> ='R' の場合、乗算は $C$ に右側から適用される。
<i>trans</i>	CHARACTER*1。'N' または 'T' でなければならない。 <i>trans</i> ='N' の場合、 $Q$ または $P$ が $C$ に適用される。 <i>trans</i> ='T' の場合、 $Q^T$ または $P^T$ が $C$ に適用される。
<i>m</i>	INTEGER。 $C$ の行数。
<i>n</i>	INTEGER。 $C$ の列数。
<i>k</i>	INTEGER。 ?gebrd から返された $A$ の次元のいずれか。 <i>vect</i> ='Q' の場合、 $A$ の列数。 <i>vect</i> ='P' の場合、 $A$ の行数。  次の制約がある。 $m \geq 0, n \geq 0, k \geq 0$ 。
<i>a, c, work</i>	REAL (sormbr の場合 ) DOUBLE PRECISION (dormbr の場合 )。 配列： $a(la,*)$ は、?gebrd から返された配列 $a$ である。 その第 2 次元は、 $\max(1, \min(r, k))$ 以上 ( <i>vect</i> ='Q' の場合 )、または $\max(1, r)$ 以上 ( <i>vect</i> ='P' の場合 ) でなければならない。  $c ldc,*)$ には、行列 $C$ を格納する。 その第 2 次元は、 $\max(1, n)$ 以上でなければならない。  $work(lwork)$ は、ワークスペース配列である。
<i>lda</i>	INTEGER。 $a$ の第 1 次元。次の制約がある。 $lda \geq \max(1, r)$ ( <i>vect</i> ='Q' の場合 )。 $lda \geq \max(1, \min(r, k))$ ( <i>vect</i> ='P' の場合 )。
<i>ldc</i>	INTEGER。 $c$ の第 1 次元。 $ldc \geq \max(1, m)$ 。
<i>tau</i>	REAL (sormbr の場合 ) DOUBLE PRECISION (dormbr の場合 )。 配列、次元は $\max(1, \min(r, k))$ 以上。 <i>vect</i> ='Q' の場合、 $\tau_{aq}$ は ?gebrd から返された配列である。 <i>vect</i> ='P' の場合、 $\tau_{ap}$ は ?gebrd から返された配列である。

*lwork* INTEGER。配列 *work* のサイズ。次の制約がある。  
 $lwork \geq \max(1, n)$  (*side* = 'L' の場合)。  
 $lwork \geq \max(1, m)$  (*side* = 'R' の場合)。  
*lwork* = -1 の場合、ワークスペースのクエリとみなされ、ルーチンは *work* 配列の最適サイズだけを計算し、その値を *work* 配列の最初のエントリとして返す。xerbla は *lwork* に関するエラー・メッセージを生成しない。  
*lwork* の推奨値は、「アプリケーション・ノート」を参照。

### 出力パラメータ

*c* (*vect*, *side*, および *trans* の値に従って)  $QC$ ,  $Q^TC$ ,  $CQ$ ,  $CQ^T$ ,  $PC$ ,  $P^TC$ ,  $CP$ , または  $CP^T$  のいずれかの積によって上書きされる。  
*work*(1) *info* = 0 の場合、終了時に、最適なパフォーマンスを得るために必要な *lwork* の最小値が *work*(1) に格納される。以降の実行では、この *lwork* 値を使用する。  
*info* INTEGER。  
*info* = 0 の場合、実行は正常に終了したことを示す。  
*info* = -*i* の場合、*i* 番目のパラメータの値が不正であったことを示す。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン ormbrr のインターフェイスの詳細を以下に示す。

*a* サイズ ( $r, \min(nq, k)$ ) の行列 *A* を格納する。ここで  
 $r = nq$  (*vect* = 'Q' の場合)  
 $r = \min(nq, k)$  (*vect* = 'P' の場合)  
 $nq = m$  (*side* = 'L' の場合)  
 $nq = n$  (*side* = 'R' の場合)  
 $k = m$  (*vect* = 'P' の場合)  
 $k = n$  (*vect* = 'Q' の場合)。  
*tau* 長さ  $\min(nq, k)$  のベクトルを格納する。  
*c* サイズ ( $m, n$ ) の行列 *C* を格納する。  
*vect* 'Q' または 'P' でなければならない。デフォルト値は 'Q'。

*side* 'L' または 'R' でなければならない。デフォルト値は 'L'。

*trans* 'N' または 'T' でなければならない。デフォルト値は 'N'。

## アプリケーション・ノート

パフォーマンスを改善するには、

$lwork = n * blocksize$  (*side* = 'L' の場合) または

$lwork = m * blocksize$  (*side* = 'R' の場合)

に設定する。*blocksize* は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。

必要なワークスペースの大きさがわからない場合は、最初の実行では *lwork* を大きめの値に設定する。終了時に、*work*(1) の値を調べ、以降はその値を使用する。

計算で求めた積は、正確な積と行列  $E$  ( $\|E\|_2 = O(\epsilon) \|C\|_2$ ) だけ異なる。

浮動小数演算のおおよその総数は、

$2 * n * k (2 * m - k)$  (*side* = 'L' かつ  $m \geq k$  の場合)、

$2 * m * k (2 * n - k)$  (*side* = 'R' かつ  $n \geq k$  の場合)、

$2 * m^2 * n$  (*side* = 'L' かつ  $m < k$  の場合)、

$2 * n^2 * m$  (*side* = 'R' かつ  $n < k$  の場合)。

このルーチンの複素数版は、[?unmbr](#) である。

---

## ?ungbr

?gebrd で求めた複素ユニタリ行列  $Q$  または  $P^H$  を生成する。

---

### 構文

#### Fortran 77:

```
call cungbr(vect, m, n, k, a, lda, tau, work, lwork, info)
```

```
call zungbr(vect, m, n, k, a, lda, tau, work, lwork, info)
```

#### Fortran 95:

```
call ungbr(a, tau [,vect] [,info])
```

## 説明

このルーチンは、ルーチン [cgebrd/zgebrd](#) で求めたユニタリ行列  $Q$  と  $P^H$  の全部または一部を生成する。このルーチンは、[cgebrd/zgebrd](#) を呼び出した後で使用する。引数の有効な組み合わせについては、入力パラメータを参照。ほとんどの場合、次のように呼び出す必要がある。

$m \times m$  の行列  $Q$  の全体を求めるには、  
`call ?ungbr ('Q', m, m, n, a ...)`  
 (配列  $a$  には、列が  $m$  個以上なければならない)。

$Q$  の先頭から  $n$  列を求めるには ( $m > n$  の場合)、  
`call ?ungbr ('Q', m, n, n, a ...)`

$n \times n$  の行列  $P^H$  の全体を求めるには、  
`call ?ungbr ('P', n, n, m, a ...)`  
 (配列  $a$  には、行が  $n$  個以上なければならない)。

$P^H$  の先頭から  $m$  行を求めるには ( $m < n$  の場合)、  
`call ?ungbr ('P', m, n, m, a ...)`

## 入力パラメータ

<code>vect</code>	CHARACTER*1。'Q' または 'P' でなければならない。 <code>vect = 'Q'</code> の場合、行列 $Q$ が生成される。 <code>vect = 'P'</code> の場合、行列 $P^H$ が生成される。
<code>m</code>	INTEGER。 $Q$ または $P^H$ に必要な行数。
<code>n</code>	INTEGER。 $Q$ または $P^H$ に必要な列数。
<code>k</code>	INTEGER。?gebrd から返された $A$ の次元のいずれか。 <code>vect = 'Q'</code> の場合、 $A$ の列数。 <code>vect = 'P'</code> の場合、 $A$ の行数。

次の制約がある。 $m \geq 0, n \geq 0, k \geq 0$ 。  
`vect = 'Q'` の場合、 $k \leq n \leq m$  ( $m > k$  の場合) または  $m = n$  ( $m \leq k$  の場合)。  
`vect = 'P'` の場合、 $k \leq m \leq n$  ( $n > k$  の場合) または  $m = n$  ( $n \leq k$  の場合)。

<i>a, work</i>	COMPLEX (cungbr の場合 ) DOUBLE COMPLEX (zungbr の場合 )。 配列： <i>a</i> ( <i>lda</i> ,*) は、?gebrd から返された配列 <i>a</i> である。 <i>a</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>work</i> ( <i>lwork</i> ) は、ワークスペース配列である。
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $\max(1, m)$ 以上。
<i>tau</i>	COMPLEX (cungbr の場合 ) DOUBLE COMPLEX (zungbr の場合 )。 <i>vect</i> = 'Q' の場合、 <i>tauq</i> は ?gebrd から返された配列である。 <i>vect</i> = 'P' の場合、 <i>taup</i> は ?gebrd から返された配列である。 <i>tau</i> の次元は、 $\max(1, \min(m, k))$ 以上 ( <i>vect</i> = 'Q' の場合 )、または $\max(1, \min(m, k))$ 以上 ( <i>vect</i> = 'P' の場合 ) でなければならない。
<i>lwork</i>	INTEGER。 配列 <i>work</i> のサイズ。 次の制約がある。 $lwork \geq \max(1, \min(m, n))$ 。 <i>lwork</i> = -1 の場合、ワークスペースのクエリとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエン トリとして返す。xerbla は <i>lwork</i> に関するエラー・メッセージを 生成しない。 <i>lwork</i> の推奨値は、「アプリケーション・ノート」を参照。

## 出力パラメータ

<i>a</i>	<i>vect</i> 、 <i>m</i> 、 <i>n</i> の値に従って、直交行列 <i>Q</i> または $P^T$ (あるいは対応する先頭からの一連の行または列) によって上書きされる。
<i>work</i> (1)	<i>info</i> = 0 の場合、終了時に、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が <i>work</i> (1) に格納される。以降の実行では、この <i>lwork</i> 値を使用する。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正であったことを示す。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `ungbr` のインターフェイスの詳細を以下に示す。

`a`                      サイズ  $(m, n)$  の行列  $A$  を格納する。

`tau`                    長さ  $\min(m, k)$  のベクトルを格納する。ここで  
                           $k = m$  (`vect = 'P'` の場合)、  
                           $k = n$  (`vect = 'Q'` の場合)。

`vect`                    'Q' または 'P' でなければならない。デフォルト値は 'Q'。

### アプリケーション・ノート

パフォーマンスを改善するには、 $lwork = \min(m, n) * blocksize$  に設定する。 $blocksize$  は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。必要なワークスペースの大きさがわからない場合は、最初の実行では  $lwork$  を大きめの値に設定する。終了時に、`work(1)` の値を調べ、以降はその値を使用する。計算で求めた行列  $Q$  は、正確な直交行列と行列  $E$  ( $\|E\|_2 = O(\epsilon)$ ) だけ異なる。

説明で示した場合ごとの浮動小数演算のおおよその総数は、次のようになる。

$Q$  の全体を求める場合、

$$(16/3)n(3m^2 - 3m*n + n^2) \quad (m > n \text{ の場合}),$$

$$(16/3)m^3 \quad (m \leq n \text{ の場合}).$$

$Q$  の先頭から  $n$  列を求める場合 ( $m > n$ )、

$$(8/3)n^2(3m - n^2) \quad (m > n \text{ の場合}).$$

$P^T$  の全体を求める場合、

$$(16/3)n^3 \quad (m \geq n \text{ の場合}),$$

$$(16/3)m(3n^2 - 3m*n + m^2) \quad (m < n \text{ の場合}).$$

$P^T$  の先頭から  $m$  列を求める場合 ( $m < n$ )、

$$(8/3)n^2(3m - n^2) \quad (m > n \text{ の場合}).$$

このルーチンの実数版は、[?orgbr](#) である。



## ?unmbr

任意の複素行列に ?gebrd で求めた  
ユニタリ行列  $Q$  または  $P$  を掛ける。

### 構文

#### Fortran 77:

```
call cunmbr(vect, side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork, info)
call zunmbr(vect, side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork, info)
```

#### Fortran 95:

```
call unmbr(a, tau, c [,vect] [,side] [,trans] [,info])
```

### 説明

このルーチンは、任意の複素行列  $C$  に対して、 $QC$ 、 $Q^HC$ 、 $CQ$ 、 $CQ^H$ 、 $PC$ 、 $P^HC$ 、 $CP$ 、または  $CP^H$  のいずれかの行列積を生成する。 $Q$  と  $P$  は、ルーチン [cgebrd/zgebrd](#) を呼び出して求めた直交行列である。このルーチンを実行すると、 $C$  の積が上書きされる。

### 入力パラメータ

下記の説明で、 $r$  は  $Q$  または  $P^H$  の次数を表す。

$side = 'L'$  の場合、 $r = m$ 、 $side = 'R'$  の場合、 $r = n$ 。

<i>vect</i>	CHARACTER*1。'Q' または 'P' でなければならない。 $vect = 'Q'$ の場合、 $Q$ または $Q^H$ が $C$ に適用される。 $vect = 'P'$ の場合、 $P$ または $P^H$ が $C$ に適用される。
<i>side</i>	CHARACTER*1。'L' または 'R' でなければならない。 $side = 'L'$ の場合、乗算は $C$ に左側から適用される。 $side = 'R'$ の場合、乗算は $C$ に右側から適用される。
<i>trans</i>	CHARACTER*1。'N' または 'C' でなければならない。 $trans = 'N'$ の場合、 $Q$ または $P$ が $C$ に適用される。 $trans = 'C'$ の場合、 $Q^H$ または $P^H$ が $C$ に適用される。
<i>m</i>	INTEGER。 $C$ の行数。
<i>n</i>	INTEGER。 $C$ の列数。

<i>k</i>	<p>INTEGER。?gebrd から返された <i>A</i> の次元のいずれか。  <i>vect</i> = 'Q' の場合、<i>A</i> の列数。  <i>vect</i> = 'P' の場合、<i>A</i> の行数。</p> <p>次の制約がある。<math>m \geq 0, n \geq 0, k \geq 0</math>。</p>
<i>a, c, work</i>	<p>COMPLEX (cunmbr の場合 )  DOUBLE COMPLEX (zunmbr の場合 )。  配列 :</p> <p><i>a</i>(<i>lda</i>,*) は、?gebrd から返された配列 <i>a</i> である。  その第 2 次元は、<math>\max(1, \min(r, k))</math> 以上 (<i>vect</i> = 'Q' の場合 )、または  <math>\max(1, r)</math> 以上 (<i>vect</i> = 'P' の場合 ) でなければならない。</p> <p><i>c</i>(<i>ldc</i>,*) には、行列 <i>C</i> を格納する。  その第 2 次元は、<math>\max(1, n)</math> 以上でなければならない。</p> <p><i>work</i>(<i>lwork</i>) は、ワークスペース配列である。</p>
<i>lda</i>	<p>INTEGER。 <i>a</i> の第 1 次元。次の制約がある。  <math>lda \geq \max(1, r)</math> (<i>vect</i> = 'Q' の場合 )。  <math>lda \geq \max(1, \min(r, k))</math> (<i>vect</i> = 'P' の場合 )。</p>
<i>ldc</i>	<p>INTEGER。 <i>c</i> の第 1 次元。 <math>ldc \geq \max(1, m)</math>。</p>
<i>tau</i>	<p>COMPLEX (cunmbr の場合 )  DOUBLE COMPLEX (zunmbr の場合 )。  配列、次元は <math>\max(1, \min(r, k))</math> 以上。  <i>vect</i> = 'Q' の場合、<i>tauq</i> は ?gebrd から返された配列である。  <i>vect</i> = 'P' の場合、<i>taup</i> は ?gebrd から返された配列である。</p>
<i>lwork</i>	<p>INTEGER。配列 <i>work</i> のサイズ。次の制約がある。  <math>lwork \geq \max(1, n)</math> (<i>side</i> = 'L' の場合 )。  <math>lwork \geq \max(1, m)</math> (<i>side</i> = 'R' の場合 )。  <i>lwork</i> = -1 の場合、ワークスペースのクエリとみなされ、ルーチンは  <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエン  トリとして返す。xerbla は <i>lwork</i> に関するエラー・メッセージを  生成しない。</p> <p><i>lwork</i> の推奨値は、「アプリケーション・ノート」を参照。</p>

## 出力パラメータ

<i>c</i>	<p>(<i>vect</i>、<i>side</i>、および <i>trans</i> の値に従って) <math>QC</math>、<math>Q^H C</math>、<math>CQ</math>、<math>CQ^H</math>、  <math>PC</math>、<math>P^H C</math>、<math>CP</math>、または <math>CP^H</math> のいずれかの積によって上書きされる。</p>
----------	---

<i>work</i> (1)	<i>info</i> = 0 の場合、終了時に、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が <i>work</i> (1) に格納される。以降の実行では、この <i>lwork</i> 値を使用する。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正であったことを示す。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *unmbr* のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ ( $r, \min(nq, k)$ ) の行列 <i>A</i> を格納する。ここで $r = nq$ ( <i>vect</i> = 'Q' の場合) $r = \min(nq, k)$ ( <i>vect</i> = 'P' の場合) $nq = m$ ( <i>side</i> = 'L' の場合) $nq = n$ ( <i>side</i> = 'R' の場合) $k = m$ ( <i>vect</i> = 'P' の場合) $k = n$ ( <i>vect</i> = 'Q' の場合)。
<i>tau</i>	長さ $\min(nq, k)$ のベクトルを格納する。
<i>c</i>	サイズ ( $m, n$ ) の行列 <i>C</i> を格納する。
<i>vect</i>	'Q' または 'P' でなければならない。デフォルト値は 'Q'。
<i>side</i>	'L' または 'R' でなければならない。デフォルト値は 'L'。
<i>trans</i>	'N' または 'C' でなければならない。デフォルト値は 'N'。

### アプリケーション・ノート

パフォーマンスを改善するには、

$lwork = n * blocksize$  (*side* = 'L' の場合) または  
 $lwork = m * blocksize$  (*side* = 'R' の場合)

に設定する。*blocksize* は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。

必要なワークスペースの大きさがわからない場合は、最初の実行では `lwork` を大きめの値に設定する。終了時に、`work(1)` の値を調べ、以降はその値を使用する。

計算で求めた積は、正確な積と行列  $E$  ( $\|E\|_2 = O(\epsilon) \|C\|_2$ ) だけ異なる。

浮動小数演算のおおよその総数は、

$8 \cdot n \cdot k(2 \cdot m - k)$	( <code>side = 'L'</code> かつ $m \geq k$ の場合)、
$8 \cdot m \cdot k(2 \cdot n - k)$	( <code>side = 'R'</code> かつ $n \geq k$ の場合)、
$8 \cdot m^2 \cdot n$	( <code>side = 'L'</code> かつ $m < k$ の場合)、
$8 \cdot n^2 \cdot m$	( <code>side = 'R'</code> かつ $n < k$ の場合)。

このルーチンの実数版は、[?ormbr](#) である。

---

## ?bdsqr

二重対角形式に縮退された一般行列の  
特異値分解を求める。

---

### 構文

#### Fortran 77:

```
call sbdsqr(uplo, n, ncv, nru, ncc, d, e, vt, ldvt, u, ldu,  
            c, ldc, work, info)  
call dbdsqr(uplo, n, ncv, nru, ncc, d, e, vt, ldvt, u, ldu,  
            c, ldc, work, info)  
call cbdsqr(uplo, n, ncv, nru, ncc, d, e, vt, ldvt, u, ldu,  
            c, ldc, work, info)  
call zbdsqr(uplo, n, ncv, nru, ncc, d, e, vt, ldvt, u, ldu,  
            c, ldc, work, info)
```

#### Fortran 95:

```
call rbdqr(d, e [,vt] [,u] [,c] [,uplo] [,info])  
call bdsqr(d, e [,vt] [,u] [,c] [,uplo] [,info])
```

## 説明

このルーチンは、暗黙的ゼロシフト  $QR$  アルゴリズムを使用して、 $n \times n$  の実上 / 下二重対角行列  $B$  に対して [特異値分解](#) (SVD) を行い、その特異値を求める。オプションで、右と左の特異ベクトルを求めることができる。行列  $B$  の SVD は、 $B = Q * S * P^H$  と表せられる。 $S$  は特異値を持つ対角行列、 $Q$  は左特異ベクトルを持つ直交行列、 $P$  は右特異ベクトルを持つ直交行列である。このサブルーチンは、与えられた実 / 複素入力行列  $U$  と  $VT$  に関して、左特異ベクトルが要求されると  $Q$  ではなく  $U * Q$  を返し、右特異ベクトルが要求されると  $P^H$  ではなく  $P^H * VT$  を返す。 $U$  と  $VT$  が、`?gebrd` で計算されるような、一般行列  $A$  を対角形式に縮退させた直交 / ユニタリ行列の場合、( $A = U * B * VT$ )、 $A$  の SVD は次のようになる。 $A = (U * Q) * S * (P^H * VT)$  このサブルーチンは、オプションで、実 / 複素入力行列  $C$  に対する  $Q^H * C$  を計算できる。

このルーチンで使用される [?lasq1](#)、[?lasq2](#)、[?lasq3](#)、[?lasq4](#)、[?lasq5](#)、および [?lasq6](#) を参照。

## 入力パラメータ

**uplo** CHARACTER\*1. 'U' または 'L' でなければならない。  
**uplo** = 'U' の場合、 $B$  は上二重対角行列である。  
**uplo** = 'L' の場合、 $B$  は下二重対角行列である。

**n** INTEGER. 行列  $B$  の次数 ( $n \geq 0$ )。

**ncvt** INTEGER. 行列  $VT$  の列数、つまり、右特異ベクトルの個数 ( $ncvt \geq 0$ )。右特異ベクトルが不要な場合、**ncvt** = 0 に設定する。

**nru** INTEGER.  $U$  の行数、つまり左特異ベクトルの個数 ( $nru \geq 0$ )。左特異ベクトルが不要な場合、**nru** = 0 に設定する。

**ncc** INTEGER. 積  $Q^H C$  を計算するために使用する行列  $C$  の列数 ( $ncc \geq 0$ )。行列  $C$  を指定しない場合、**ncc** = 0 に設定する。

**d, e, work** REAL (単精度の場合)  
 DOUBLE PRECISION (倍精度の場合)。  
 配列：  
**d**(\*) には、 $B$  の対角成分を格納する。  
**d** の次元は、 $\max(1, n)$  以上でなければならない。  
**e**(\*) には、 $B$  の ( $n-1$ ) の非対角成分を格納する。  
**e** の次元は、 $\max(1, n)$  以上でなければならない。  
**e**( $n$ ) はワークスペース用に使用される。  
**work**(\*) は、ワークスペース配列である。  
**work** の次元は、 $\max(1, 2*n)$  以上 (**ncvt** = **nru** = **ncc** = 0 の場合) または  $\max(1, 4*(n-1))$  以上 (それ以外の場合) でなければならない。

$vt, u, c$	<p>REAL (sbdsqr の場合 )  DOUBLE PRECISION (dbdsqr の場合 )  COMPLEX (cbdsqr の場合 )  DOUBLE COMPLEX (zbdbsqr の場合 )。  配列 :  <math>vt(ldvt, *)</math> には、<math>n \times ncv</math> の行列 <math>VT</math> を格納する。  <math>vt</math> の第 2 次元は、<math>\max(1, ncv)</math> 以上でなければならない。  <math>ncv = 0</math> の場合、<math>vt</math> は参照されない。</p> <p><math>u(ldu, *)</math> には、<math>nru \times n</math> の単位行列 <math>U</math> を格納する。  <math>u</math> の第 2 次元は、<math>\max(1, n)</math> 以上でなければならない。  <math>nru = 0</math> の場合、<math>u</math> は参照されない。</p> <p><math>c ldc, *)</math> には、積 <math>Q^H * C</math> を計算するための行列 <math>C</math> を格納する。<math>c</math> の  第 2 次元は、<math>\max(1, ncc)</math> 以上でなければならない。<math>ncc = 0</math> の場合、  対応する配列は参照されない。</p>
$ldvt$	<p>INTEGER。 <math>vt</math> の第 1 次元。次の制約がある。  <math>ldvt \geq \max(1, n)</math> (<math>ncv &gt; 0</math> の場合 )。  <math>ldvt \geq 1</math> (<math>ncv = 0</math> の場合 )。</p>
$ldu$	<p>INTEGER。 <math>u</math> の第 1 次元。次の制約がある。  <math>ldu \geq \max(1, nru)</math>。</p>
$ldc$	<p>INTEGER。 <math>c</math> の第 1 次元。次の制約がある。  <math>ldc \geq \max(1, n)</math> (<math>ncc &gt; 0</math> の場合 )。  <math>ldc \geq 1</math> (その他の場合 )。</p>

## 出力パラメータ

$d$	終了時に、 $info = 0$ の場合、特異値によって降順に上書きされる ( $info$ を参照)。
$e$	終了時に、 $info = 0$ の場合、 $e$ は破棄される。後述の $info$ も参照。
$c$	積 $Q^H * C$ によって上書きされる。
$vt$	終了時に、 $P^H * VT$ によって上書きされる。
$u$	終了時に、 $U * Q$ によって上書きされる。
$info$	<p>INTEGER。  <math>info = 0</math> の場合、実行は正常に終了したことを示す。  <math>info = -i</math> の場合、<math>i</math> 番目のパラメータの値が不正であったことを示す。  <math>info = i</math> の場合、アルゴリズムが収束していない。</p>

$i$  は、収束しなかった非対角成分の個数を示す。  
 その場合、終了時に、 $d$  と  $e$  に、二重対角行列 ( $B$  の直交行列と等価) の対角成分と非対角成分が格納される。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `bdsqr` のインターフェイスの詳細を以下に示す。

$d$	長さ ( $n$ ) のベクトルを格納する。
$e$	長さ ( $n$ ) のベクトルを格納する。
$vt$	サイズ ( $n, ncv$ ) の行列 $VT$ を格納する。
$u$	サイズ ( $nru, n$ ) の行列 $U$ を格納する。
$c$	サイズ ( $n, ncc$ ) の行列 $C$ を格納する。
$uplo$	'U' または 'L' でなければならない。デフォルト値は 'U'。
$ncv$	引数 $vt$ を指定する場合、 $ncv$ は行列 $VT$ の列数と同じ。それ以外の場合、 $ncv$ にはゼロが設定される。
$nru$	引数 $u$ を指定する場合、 $nru$ は行列 $U$ の行数と同じ。それ以外の場合、 $nru$ にはゼロが設定される。
$ncc$	引数 $c$ を指定する場合、 $ncc$ は行列 $C$ の列数と同じ。それ以外の場合、 $ncc$ にはゼロが設定される。

$vt$ 、 $u$ 、および  $c$  が省略されると、実数および複素数の場合に曖昧な選択が行われるため、`bdsqr` ルーチンには 2 つの Fortran 95 インターフェイスが必要な点に注意する。したがって、実数の場合 (単精度または倍精度) は `rbdsqr`、複素数の場合 (単精度または倍精度) は `bdsqr` を使用する。

### アプリケーション・ノート

それぞれの特異値や特異ベクトルは、比較的高い精度で計算される。ただし、(このルーチンが呼び出される前に実行される) 二重対角形式への縮退によって、特異値の絶対値が大きく変動するような場合には、元の行列の特異値が小さいときに相対精度が低下する場合もある。

$\sigma_i$  が  $B$  の正確な特異値で、 $s_i$  が対応する計算で求めた値の場合、次のようになる。

$$|s_i - \sigma_i| \leq p(m, n)\varepsilon\sigma_i$$

$p(m, n)$  は、漸増する  $m$  と  $n$  の関数で、 $\varepsilon$  はマシン精度である。特異値だけを計算するほうが、特異ベクトルも一緒に計算する（つまり、関数  $p(m, n)$  が小さい）よりも結果は正確である。

$u_i$  が  $B$  の対応する正確な左特異ベクトルで、 $w_i$  が対応する計算で求めた左特異ベクトルの場合、それらの間の角  $\theta(u_i, w_i)$  は次のようになる。

$$\theta(u_i, w_i) \leq p(m, n)\varepsilon / \min_{i \neq j} (|\sigma_i - \sigma_j| / |\sigma_i + \sigma_j|)$$

$\min_{i \neq j} (|\sigma_i - \sigma_j| / |\sigma_i + \sigma_j|)$  は、 $\sigma_i$  と他の特異値との間の相対誤差である。右特異ベクトルに関しても、これと同様に誤差が制限される。

特異値だけを計算する場合、実数型の浮動小数演算のおおよその総数は、 $n^2$  に比例する。左特異ベクトルを計算する場合は約  $6n^2 \cdot nru$ （複素数型の場合は約  $12n^2 \cdot nru$ ）回だけ、右特異ベクトルを計算する場合は約  $6n^2 \cdot ncvt$ （複素数型の場合は約  $12n^2 \cdot ncvt$ ）回だけ、余分な計算が必要になる。

## ?bdsdc

分割統治法を使用して、実二重対角行列の特異値分解を実行する。

### 構文

#### Fortran 77:

```
call sbdsdc(uplo, compq, n, d, e, u, ldu, vt, ldvt, q, iq, work, iwork, info)
call dbdsdc(uplo, compq, n, d, e, u, ldu, vt, ldvt, q, iq, work, iwork, info)
```

#### Fortran 95:

```
call bdsdc(d, e [,u] [,vt] [,q] [,iq] [,uplo] [,info])
```

### 説明

このルーチンは、分割統治法を使用して、 $n \times n$  の実上 / 二重対角行列  $B$  に対する [特異値分解](#) (SVD)、すなわち  $B = U \Sigma V^T$  を計算する。 $\Sigma$  は非負の対角成分 ( $B$  の特異値) を持つ対角行列、 $U$  と  $V$  は、それぞれ左と右の特異ベクトルを持つ直交行列である。`?bdsdc` を使用して、すべての特異値を計算し、オプションで特異ベクトル（またはコンパクト形式の特異ベクトル）を求めることができる。



このルーチンで使用する [?lasd0](#)、[?lasd1](#)、[?lasd2](#)、[?lasd3](#)、[?lasd4](#)、[?lasd5](#)、[?lasd6](#)、[?lasd7](#)、[?lasd8](#)、[?lasd9](#)、[?lasda](#)、[?lasdg](#)、および [?lasdt](#) を参照。

### 入力パラメータ

<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> = 'U' の場合、 <i>B</i> は上二重対角行列である。 <i>uplo</i> = 'L' の場合、 <i>B</i> は下二重対角行列である。
<i>compq</i>	CHARACTER*1。'N'、'P'、または 'I' でなければならない。 <i>compq</i> = 'N' の場合、特異値のみを計算する。 <i>compq</i> = 'P' の場合、特異値とコンパクト形式の特異ベクトルを計算する。 <i>compq</i> = 'I' の場合、特異値と特異ベクトルを計算する。
<i>n</i>	INTEGER。行列 <i>B</i> の次数 ( $n \geq 0$ )。
<i>d</i> , <i>e</i> , <i>work</i>	REAL (sbdsdc の場合) DOUBLE PRECISION (sbdsdc の場合)。 配列: <i>d</i> (*) には、二重対角行列 <i>B</i> の <i>n</i> 個の対角成分を格納する。 <i>d</i> の次元は、 $\max(1, n)$ 以上でなければならない。 <i>e</i> (*) には、二重対角行列 <i>B</i> の非対角成分を格納する。 <i>e</i> の次元は、 $\max(1, n)$ 以上でなければならない。 <i>work</i> (*) は、ワークスペース配列である。 <i>work</i> の次元は、 $\max(1, 4*n)$ 以上 ( <i>compq</i> = 'N' の場合) または $\max(1, 6*n)$ 以上 ( <i>compq</i> = 'P' の場合) または $\max(1, 3*n^2+4*n)$ 以上 ( <i>compq</i> = 'I' の場合) でなければならない。
<i>ldu</i>	INTEGER。出力配列 <i>u</i> の第 1 次元。 $ldu \geq 1$ 。特異ベクトルも要求する場合、 $ldu \geq \max(1, n)$ 。
<i>ldvt</i>	INTEGER。出力配列 <i>vt</i> の第 1 次元。 $ldvt \geq 1$ 。特異ベクトルも要求する場合、 $ldvt \geq \max(1, n)$ 。
<i>iwork</i>	INTEGER。ワークスペース配列。サイズは $\max(1, 8*n)$ 以上。

### 出力パラメータ

<i>d</i>	<i>info</i> = 0 の場合、 <i>B</i> の特異値によって上書きされる。
<i>e</i>	終了時に、 <i>e</i> は上書きされる。

<i>u, vt, q</i>	<p>REAL (sbdsdc の場合) DOUBLE PRECISION (dbdsdc の場合)。 配列: <i>u(ldu,*)</i>, <i>vt(ldvt,*)</i>, <i>q(*)</i>。 <i>compq</i> = 'I' の場合、終了時に、二重対角行列 <i>B</i> の左特異ベクトルが <i>u</i> に格納される (ただし、<i>info</i> ≠ 0 でない場合 (<i>info</i> を参照))。 <i>compq</i> が他の値の場合、<i>u</i> は参照されない。 <i>u</i> の第 2 次元は、<math>\max(1, n)</math> 以上でなければならない。</p> <p><i>compq</i> = 'I' の場合、終了時に、二重対角行列 <i>B</i> の右特異ベクトルが <i>vt</i> に格納される (ただし、<i>info</i> ≠ 0 でない場合 (<i>info</i> を参照))。 <i>compq</i> が他の値の場合、<i>vt</i> は参照されない。 <i>vt</i> の第 2 次元は、<math>\max(1, n)</math> 以上でなければならない。</p> <p><i>compq</i> = 'P' の場合、終了時に、<i>info</i> = 0 の場合、<i>q</i> と <i>iq</i> に左と右の特異ベクトルがコンパクト形式で格納される。 <i>q</i> には、特異ベクトルのすべての REAL データ (sbdsdc の場合) または DOUBLE PRECISION データ (dbdsdc の場合) が格納される。 <i>compq</i> が他の値の場合、<i>q</i> は参照されない。詳細は、「アプリケーション・ノート」を参照。</p>
<i>iq</i>	<p>INTEGER。 配列: <i>iq(*)</i>。 <i>compq</i> = 'P' の場合、終了時に、<i>info</i> = 0 の場合、<i>q</i> と <i>iq</i> に左と右の特異ベクトルがコンパクト形式で格納される。 <i>iq</i> には、特異ベクトルのすべての INTEGER データが格納される。 <i>compq</i> が他の値の場合、<i>iq</i> は参照されない。詳細は、「アプリケーション・ノート」を参照。</p>
<i>info</i>	<p>INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = -<i>i</i> の場合、<i>i</i> 番目のパラメータの値が不正であったことを示す。 <i>info</i> = <i>i</i> の場合、このアルゴリズムによる特異値計算に失敗したことを示す。分割統治法の更新プロセスに失敗した。</p>

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *bdsdc* のインターフェイスの詳細を以下に示す。

<i>d</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>e</i>	長さ ( <i>n</i> ) のベクトルを格納する。

$u$	サイズ $(n, n)$ の行列 $U$ を格納する。
$vt$	サイズ $(n, n)$ の行列 $VT$ を格納する。
$q$	長さ $(ldq)$ のベクトルを格納する。ここで $ldq \geq n * (11 + 2 * smlsiz + 8 * \text{int}(\log_2(n / (smlsiz + 1))))$ および $smlsiz$ は $laenv$ によって返された値で、計算ツリーが一番下の部分問題の最大サ サイズ (通常は約 25) と等しい。
$compq$	引数 $u$ 、 $vt$ 、 $q$ 、および $iq$ の存在に基づいて以下のように復元される。 $compq = 'N'$ ( $u$ 、 $vt$ 、 $q$ 、および $iq$ がすべて存在しない場合)、 $compq = 'I'$ ( $u$ と $vt$ の両方が存在する場合)。引数 $u$ と $vt$ は、両方存 在するか、両方省略されるかのいずれかでなければならない。 $compq = 'P'$ ( $q$ と $iq$ の両方が存在する場合)。引数 $q$ と $iq$ は、両方存 在するか、両方省略されるかのいずれかでなければならない。  引数 $u$ 、 $vt$ 、 $q$ 、および $iq$ がすべて同時に存在する場合、エラー条件 がセットされる。

## 対称固有値問題

対称固有値問題は、「 $n \times n$  の実対称行列または複素エルミート行列  $A$  が与えられたとき  
に、次の方程式を満たす固有値  $\lambda$  と、それに対応する固有ベクトル  $z$  を見つけること」  
である。

$$Az = \lambda z \quad (\text{または } z^H A = \lambda z^H)$$

このような固有値問題の場合、実対称行列だけでなく複素エルミート行列  $A$  に関しても、 $n$  個の固有値はすべて実数型になる。そして、 $n$  個の固有ベクトルから構成される  
正規直交系が 1 つ存在する。 $A$  が対称 / エルミート正定値行列である場合は、どの固有  
値も正になる。

LAPACK を使って対称固有値問題を解く場合は、通常、対応する行列をまず三重対角形  
式に縮退した後、得られた三重対角行列を使って固有値問題を解く必要がある。

LAPACK には、直交 (またはユニタリ) の相似変換  $A = QTQ^H$  によって行列を三重対角  
形式に縮退させるためのルーチンだけでなく、三重対角の対称固有値問題を解くための  
ルーチンも含まれている。それらのルーチン (Fortran-77 インターフェイス) を、[表 4-3](#)  
に示す。

Fortran-95 インターフェイスのルーチン名はそれぞれ、先頭の記号がない ([ルーチン命  
名規則](#)を参照)。

対称固有値問題を解くためのルーチンには、固有ベクトルの全部あるいは一部が必要な  
場合は、固有値のみが必要な場合、行列  $A$  が正定値であるかどうかなどに応じて、さま  
ざまなものが用意されている。

これらのルーチンは、対称問題で固有値と固有ベクトルを計算する 3 つの主要アルゴリズムに基づいている。3 つの主要アルゴリズムとは、分割統治アルゴリズム、QR アルゴリズム、二分法と逆反復法の組み合わせである。一般に、分割統治アルゴリズムが最も効率が良いので、固有値と固有ベクトルをすべて計算するときには、このアルゴリズムが推奨される。

また、分割統治アルゴリズムを使用して固有値問題を解く場合は、ルーチンを 1 つだけ呼び出せばよい。一般に、QR アルゴリズム、または二分法と逆反復法の組み合わせを使用すると、複数のルーチンを呼び出す必要がある。

[図 4-2](#) のデシジョン・ツリーを参考にすれば、実対称行列の固有値問題を解くための正しいルーチン (1 つまたは複数個) を簡単に選べる。[図 4-3](#) は、複素エルミート行列の場合のデシジョン・ツリーである。

図 4-2 デシジョン・ツリー：実対称行列の固有値問題

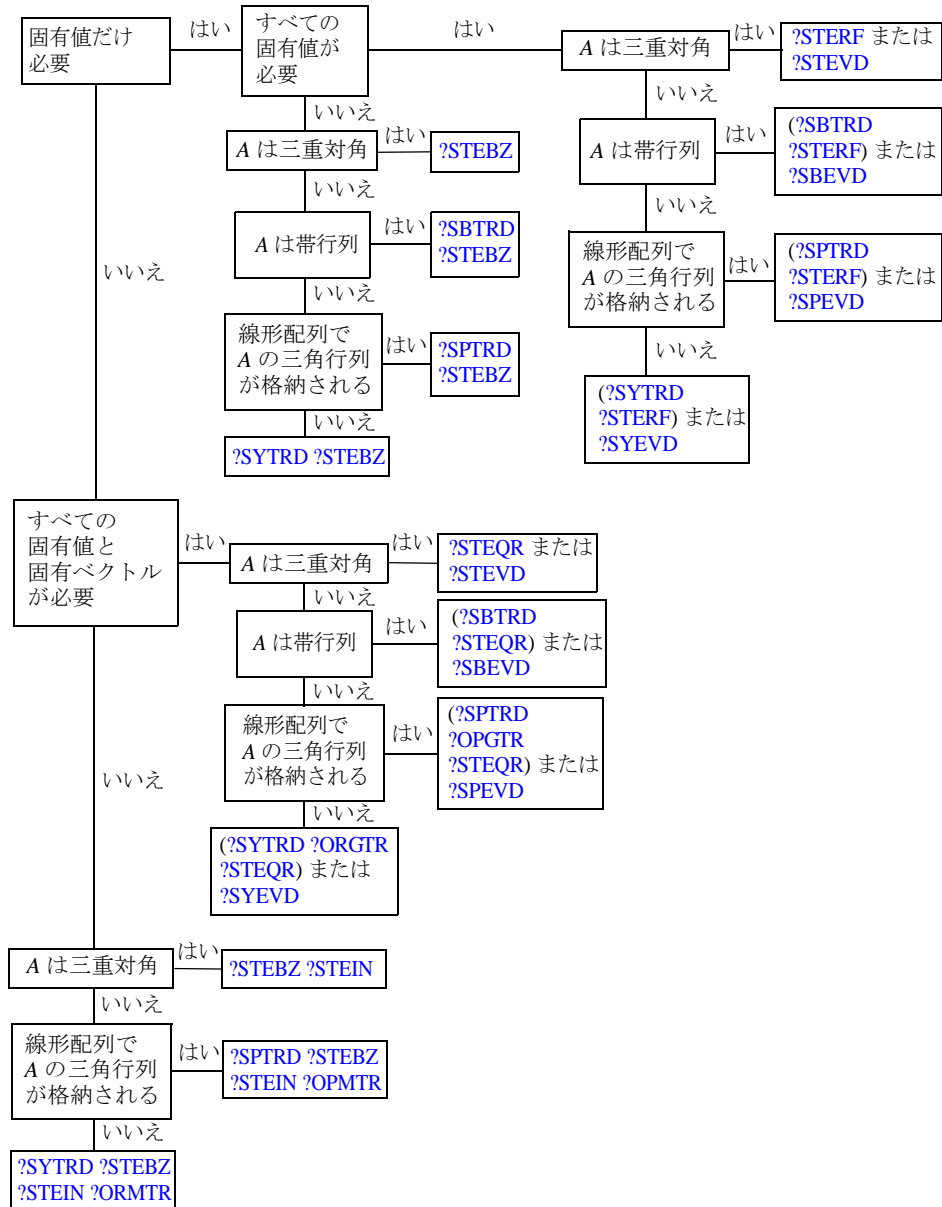


図 4-3 デシジョン・ツリー：複素エルミート行列の固有値問題

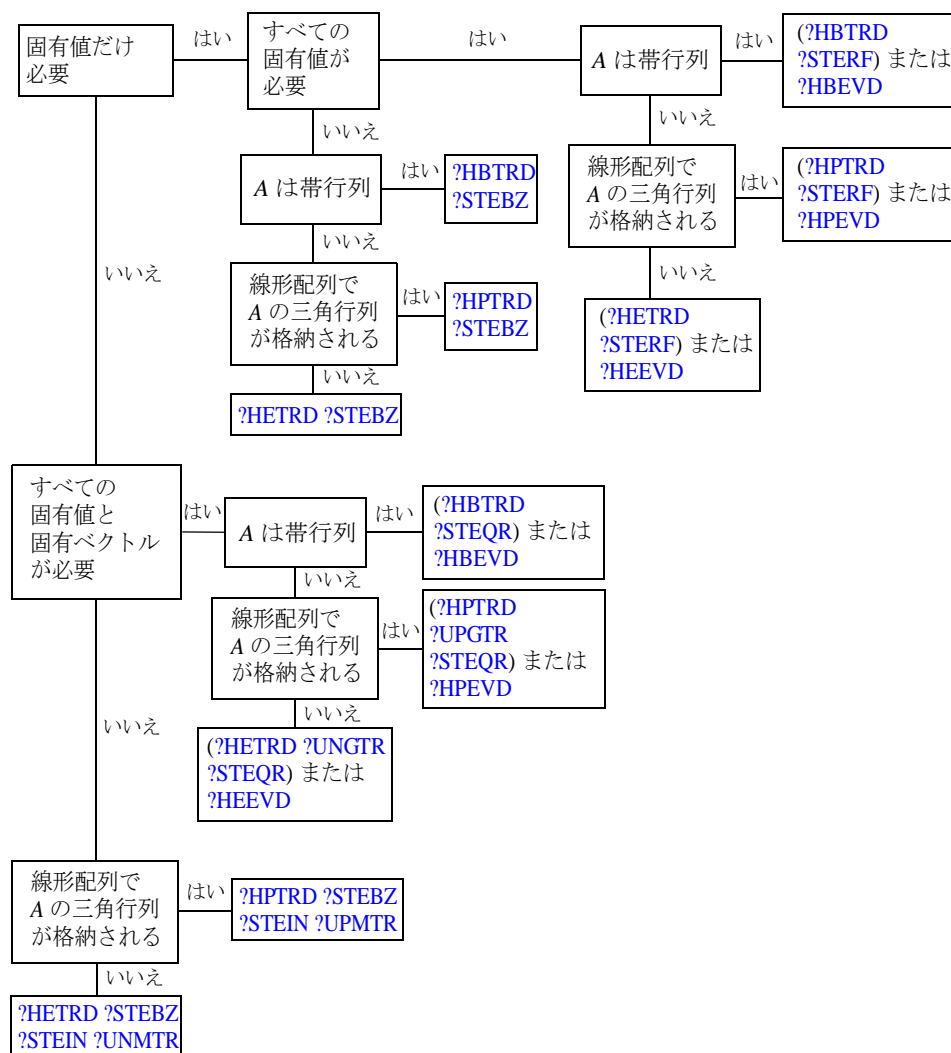


表 4-3 対称固有値問題用の計算ルーチン

演算	対称実行列		複素エルミート行列	
三重対角形式に縮退させる $A = QTQ^H$ (フル格納)	<a href="#">?sytrd</a>		<a href="#">?hetrd</a>	
三重対角形式に縮退させる $A = QTQ^H$ (圧縮格納)	<a href="#">?sptrd</a>		<a href="#">?hptrd</a>	
三重対角形式に縮退させる $A = QTQ^H$ (帯格納)	<a href="#">?sbtrd</a>		<a href="#">?hbtrd</a>	
行列 $Q$ を生成する (フル格納)	<a href="#">?orgtr</a>		<a href="#">?ungtr</a>	
行列 $Q$ を生成する (圧縮格納)	<a href="#">?opgtr</a>		<a href="#">?upgtr</a>	
行列 $Q$ を適用する (フル格納)	<a href="#">?ormtr</a>		<a href="#">?unmtr</a>	
行列 $Q$ を適用する (圧縮格納)	<a href="#">?opmtr</a>		<a href="#">?upmtr</a>	
三重対角行列 $T$ のすべての固有値を見つける	<a href="#">?sterf</a>			
三重対角行列 $T$ のすべての固有値と固有ベクトルを見つける	<a href="#">?stegr</a>	<a href="#">?stedc</a>	<a href="#">?stegr</a>	<a href="#">?stedc</a>
三重対角正定値行列 $T$ のすべての固有値と固有ベクトルを見つける	<a href="#">?ptegr</a>		<a href="#">?ptegr</a>	
三重対角行列 $T$ の指定された固有値を求める	<a href="#">?stebz</a> <a href="#">?stegr</a>		<a href="#">?stegr</a>	
三重対角行列 $T$ の指定された固有ベクトルを求める	<a href="#">?stein</a> <a href="#">?stegr</a>		<a href="#">?stein</a> <a href="#">?stegr</a>	
固有ベクトルの相互条件数を計算する	<a href="#">?disna</a>		<a href="#">?disna</a>	

### ?sytrd

実対称行列を三重対角形式に縮退させる。

---

#### 構文

##### Fortran 77:

```
call ssytrd(uplo, n, a, lda, d, e, tau, work, lwork, info)
call dsytrd(uplo, n, a, lda, d, e, tau, work, lwork, info)
```

##### Fortran 95:

```
call sytrd(a, tau [,uplo] [,info])
```

#### 説明

このルーチンは、直交相似変換  $A = QTQ^T$  によって、実対称行列  $A$  を対称三重対角形式  $T$  に縮退させる。直交行列  $Q$  は、明示的な形式では表現されず、 $(n-1)$  個の基本リフレクタの積として表現される。一連のルーチンでは、この形式で表現される  $Q$  を操作する (このセクションで後述)。

このルーチン [?latrd](#) で、直交相似変換によって、実対称行列を三重対角形式  $T$  に縮退させる。

#### 入力パラメータ

<code>uplo</code>	CHARACTER*1。'U' または 'L' でなければならない。 <code>uplo = 'U'</code> の場合、 <code>a</code> には $A$ の上三角部分を格納する。 <code>uplo = 'L'</code> の場合、 <code>a</code> には $A$ の下三角部分を格納する。
<code>n</code>	INTEGER。行列 $A$ の次数 ( $n \geq 0$ )。
<code>a, work</code>	REAL (ssytrd の場合) DOUBLE PRECISION (dsytrd の場合)。 <code>a(lda,*)</code> は、 <code>uplo</code> の値に従って、行列 $A$ の上三角部分または下三角部分を格納する配列である。 <code>a</code> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。  <code>work(lwork)</code> は、ワークスペース配列である。
<code>lda</code>	INTEGER。 <code>a</code> の第 1 次元。 $\max(1, n)$ 以上。



*lwork* INTEGER。配列 *work* のサイズ ( $lwork \geq n$ )。  
*lwork* = -1 の場合、ワークスペースのクエリとみなされ、ルーチンは *work* 配列の最適サイズだけを計算し、その値を *work* 配列の最初のエントリとして返す。xerbla は *lwork* に関するエラー・メッセージを生成しない。  
*lwork* の推奨値は、「アプリケーション・ノート」を参照。

## 出力パラメータ

*a* *uplo* の値に従って、三重対角行列 *T* と直交行列 *Q* の各成分によって上書きされる。

*d*, *e*, *tau* REAL (ssytrd の場合)  
DOUBLE PRECISION (dsytrd の場合)。  
配列：  
*d*(\*) には、行列 *T* の対角成分が格納される。  
*d* の次元は、 $\max(1, n)$  以上でなければならない。  
*e*(\*) には、*T* の非対角成分が格納される。  
*e* の次元は、 $\max(1, n-1)$  以上でなければならない。  
*tau*(\*) には、直交行列 *Q* の各成分が格納される。*tau* の次元は、 $\max(1, n-1)$  以上でなければならない。

*work*(1) *info* = 0 の場合、終了時に、最適なパフォーマンスを得るために必要な *lwork* の最小値が *work*(1) に格納される。以降の実行では、この *lwork* 値を使用する。

*info* INTEGER。  
*info* = 0 の場合、実行は正常に終了したことを示す。  
*info* = -*i* の場合、*i* 番目のパラメータの値が不正であったことを示す。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *sytrd* のインターフェイスの詳細を以下に示す。

*a* サイズ ( $n, n$ ) の行列 *A* を格納する。  
*tau* 長さ ( $n-1$ ) のベクトルを格納する。

*d*                      長さ (*n*) のベクトルを格納する。  
*e*                      長さ (*n*-1) のベクトルを格納する。  
*uplo*                  'U' または 'L' でなければならない。デフォルト値は 'U'。

### アプリケーション・ノート

パフォーマンスを改善するには、*lwork* = *n*\**blocksize* に設定する。*blocksize* は、ブロック・アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。必要なワークスペースの大きさがわからない場合は、最初の実行では *lwork* を大きめの値に設定する。終了時に、*work*(1) の値を調べ、以降はその値を使用する。

計算で求めた行列 *T* は、行列 *A* + *E* の近似値である。ここで、 $\|E\|_2 = c(n)\varepsilon \|A\|_2$ 、*c*(*n*) は漸増する *n* の関数、 $\varepsilon$  はマシンによって異なる値である。

浮動小数演算のおおよその総数は、 $(4/3)n^3$  である。

このルーチン呼び出した後は、以下のルーチン呼び出せる。

[?orgtr](#)                      計算で求める行列 *Q* を明示的に生成する場合

[?ormtr](#)                      実行列に *Q* を掛ける場合

このルーチンの複素数版は、[?hetrd](#) である。

---

## ?orgtr

?sytrd で求めた実直交行列 *Q* を生成する。

---

### 構文

#### Fortran 77:

```
call sorgtr(uplo, n, a, lda, tau, work, lwork, info)
call dorgtr(uplo, n, a, lda, tau, work, lwork, info)
```

#### Fortran 95:

```
call orgtr(a, tau [,uplo] [,info])
```

## 説明

このルーチンは、実対称行列  $A$  を三重対角行列の形式  $A = QTQ^T$  に縮退させるために [?sytrd](#) によって求めた  $n \times n$  の直交行列  $Q$  を明示的に生成する。このルーチンは、[?sytrd](#) を呼び出した後で使用する。

## 入力パラメータ

**uplo** CHARACTER\*1. 'U' または 'L' でなければならない。  
?sytrd に指定した **uplo** と同じ値を使用する。

**n** INTEGER。行列  $Q$  の次数 ( $n \geq 0$ )。

**a, tau, work** REAL (sorgtr の場合)  
DOUBLE PRECISION (dorgtr の場合)。  
配列:  
**a**(**lda**,\*) は、?sytrd から返された配列 **a** である。  
**a** の第 2 次元は、 $\max(1, n)$  以上でなければならない。  
  
**tau**(\*) は、?sytrd から返された配列 **tau** である。  
**tau** の次元は、 $\max(1, n-1)$  以上でなければならない。  
  
**work**(**lwork**) は、ワークスペース配列である。

**lda** INTEGER。 **a** の第 1 次元。  $\max(1, n)$  以上。

**lwork** INTEGER。 配列 **work** のサイズ ( $lwork \geq n$ )。  
 $lwork = -1$  の場合、ワークスペースのクエリとみなされ、ルーチンは **work** 配列の最適サイズだけを計算し、その値を **work** 配列の最初のエントリとして返す。xerbla は **lwork** に関するエラー・メッセージを生成しない。  
  
**lwork** の推奨値は、「アプリケーション・ノート」を参照。

## 出力パラメータ

**a** 直交行列  $Q$  によって上書きされる。

**work(1)** **info** = 0 の場合、終了時に、最適なパフォーマンスを得るために必要な **lwork** の最小値が **work(1)** に格納される。以降の実行では、この **lwork** 値を使用する。

**info** INTEGER。  
**info** = 0 の場合、実行は正常に終了したことを示す。  
**info** = -*i* の場合、*i* 番目のパラメータの値が不正であったことを示す。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `orgtr` のインターフェイスの詳細を以下に示す。

`a`                    サイズ  $(n,n)$  の行列  $A$  を格納する。  
`tau`                    長さ  $(n-1)$  のベクトルを格納する。  
`uplo`                    'U' または 'L' でなければならない。デフォルト値は 'U'。

### アプリケーション・ノート

パフォーマンスを改善するには、 $lwork = (n-1) * blocksize$  に設定する。 $blocksize$  は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。

必要なワークスペースの大きさがわからない場合は、最初の実行では  $lwork$  を大きめの値に設定する。終了時に、`work(1)` の値を調べ、以降はその値を使用する。

計算で求めた  $Q$  は、正確な直交行列と行列  $E$  ( $\|E\|_2 = O(\epsilon)$ 、 $\epsilon$  はマシン精度) だけ異なる。

浮動小数演算のおおよその総数は、 $(4/3)n^3$  である。

このルーチンの複素数版は、[?ungtr](#) である。

---

## ?ormtr

実行列に `?sytrd` で求めた実直交行列  $Q$  を掛ける。

---

### 構文

#### Fortran 77:

```
call sormtr(side, uplo, trans, m, n, a, lda, tau, c, ldc, work, lwork, info)
call dormtr(side, uplo, trans, m, n, a, lda, tau, c, ldc, work, lwork, info)
```

#### Fortran 95:

```
call ormtr(a, tau, c [,side] [,uplo] [,trans] [,info])
```

## 説明

このルーチンは、実行列  $C$  に  $Q$  または  $Q^T$  を掛ける。 $Q$  は、実対称行列  $A$  を三重対角形式  $A = QTQ^T$  に縮退させるために [?sytrd](#) によって求めた直交行列  $Q$  である。このルーチンは、`?sytrd` を呼び出した後で使用する。

このルーチンを使用すれば、パラメータ `side` と `trans` の値に従って、 $QC$ 、 $Q^TC$ 、 $CQ$ 、または  $CQ^T$  のいずれかの行列積を求めることができる ( $C$  の値は上書きされる)。

## 入力パラメータ

下記の説明で、 $r$  は  $Q$  の次数を表す。

`side = 'L'` の場合、 $r = m$ 、`side = 'R'` の場合、 $r = n$ 。

<code>side</code>	CHARACTER*1。'L' または 'R' でなければならない。 <code>side = 'L'</code> の場合、 $Q$ または $Q^T$ は、 $C$ に左側から適用される。 <code>side = 'R'</code> の場合、 $Q$ または $Q^T$ は、 $C$ に右側から適用される。
<code>uplo</code>	CHARACTER*1。'U' または 'L' でなければならない。 <code>?sytrd</code> に指定した <code>uplo</code> と同じ値を使用する。
<code>trans</code>	CHARACTER*1。'N' または 'T' でなければならない。 <code>trans = 'N'</code> の場合、 $C$ に $Q$ を掛ける。 <code>trans = 'T'</code> の場合、 $C$ に $Q^T$ を掛ける。
<code>m</code>	INTEGER。行列 $C$ の行数 ( $m \geq 0$ )。
<code>n</code>	INTEGER。行列 $C$ の列数 ( $n \geq 0$ )。
<code>a, work, tau, c</code>	REAL (sormtr の場合) DOUBLE PRECISION (dormtr の場合)。 <code>a(lda,*)</code> と <code>tau</code> は、 <code>?sytrd</code> から返された配列である。 $a$ の第 2 次元は、 $\max(1, r)$ 以上でなければならない。 $\tau$ の次元は、 $\max(1, r-1)$ 以上でなければならない。 <code>c(ldc,*)</code> には、行列 $C$ を格納する。 $c$ の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <code>work(lwork)</code> は、ワークスペース配列である。
<code>lda</code>	INTEGER。 $a$ の第 1 次元。 $lda \geq \max(1, r)$ 。
<code>ldc</code>	INTEGER。 $c$ の第 1 次元。 $ldc \geq \max(1, n)$ 。
<code>lwork</code>	INTEGER。配列 <code>work</code> のサイズ。次の制約がある。 $lwork \geq \max(1, n)$ ( <code>side = 'L'</code> の場合)。 $lwork \geq \max(1, m)$ ( <code>side = 'R'</code> の場合)。

$lwork = -1$  の場合、ワークスペースのクエリとみなされ、ルーチンは  $work$  配列の最適サイズだけを計算し、その値を  $work$  配列の最初のエントリとして返す。xerbla は  $lwork$  に関するエラー・メッセージを生成しない。

$lwork$  の推奨値は、「アプリケーション・ノート」を参照。

### 出力パラメータ

$c$	( $side$ と $trans$ の値に従って) $QC$ 、 $Q^TC$ 、 $CQ$ 、または $CQ^T$ のいずれかの積によって上書きされる。
$work(1)$	$info = 0$ の場合、終了時に、最適なパフォーマンスを得るために必要な $lwork$ の最小値が $work(1)$ に格納される。以降の実行では、この $lwork$ 値を使用する。
$info$	INTEGER。 $info = 0$ の場合、実行は正常に終了したことを示す。 $info = -i$ の場合、 $i$ 番目のパラメータの値が不正であったことを示す。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン ormtr のインターフェイスの詳細を以下に示す。

$a$	サイズ $(r, r)$ の行列 $A$ を格納する。 $r = m(side = 'L' \text{ の場合 })$ 。 $r = n(side = 'R' \text{ の場合 })$ 。
$tau$	長さ $(r-1)$ のベクトルを格納する。
$c$	サイズ $(m, n)$ の行列 $C$ を格納する。
$side$	'L' または 'R' でなければならない。デフォルト値は 'L'。
$uplo$	'U' または 'L' でなければならない。デフォルト値は 'U'。
$trans$	'N' または 'T' でなければならない。デフォルト値は 'N'。

## アプリケーション・ノート

パフォーマンスを改善するには、 $lwork = n * blocksize$  ( $side = 'L'$  の場合) または  $lwork = m * blocksize$  ( $side = 'R'$  の場合) に設定する。 $blocksize$  は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。必要なワークスペースの大きさがわからない場合は、最初の実行では  $lwork$  を大きめの値に設定する。終了時に、 $work(1)$  の値を調べ、以降はその値を使用する。

計算で求めた積は、正確な積と行列  $E$  ( $\|E\|_2 = O(\epsilon) \|C\|_2$ ) だけ異なる。

浮動小数演算のおおよその総数は、 $2 * m^2 * n$  ( $side = 'L'$  の場合)、または  $2 * n^2 * m$  ( $side = 'R'$  の場合) である。

このルーチンの複素数版は、[?unmtr](#) である。

## ?hetrd

複素エルミート行列を三重対角形式に縮退させる。

### 構文

#### Fortran 77:

```
call chetrd(uplo, n, a, lda, d, e, tau, work, lwork, info)
call zhetrd(uplo, n, a, lda, d, e, tau, work, lwork, info)
```

#### Fortran 95:

```
call hetrd(a, tau [,uplo] [,info])
```

### 説明

このルーチンは、ユニタリ相似変換  $A = QTQ^H$  によって、複素エルミート行列  $A$  を対称な三重対角形式  $T$  に縮退させる。ユニタリ行列  $Q$  は、明示的な形式では表現されず、 $(n-1)$  個の基本リフレクタの積として表現される。一連のルーチンでは、この形式で表現される  $Q$  を操作する。(このセクションで後述)。

このルーチンは、[?latrd](#) を呼び出して、ユニタリ相似変換によって、複素エルミート行列  $A$  をエルミート三重対角形式に縮退させる。

## 入力パラメータ

<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> = 'U' の場合、 <i>a</i> には <i>A</i> の上三角部分を格納する。 <i>uplo</i> = 'L' の場合、 <i>a</i> には <i>A</i> の下三角部分を格納する。
<i>n</i>	INTEGER。行列 <i>A</i> の次数 ( $n \geq 0$ )。
<i>a</i> , <i>work</i>	COMPLEX (chetrd の場合 ) DOUBLE COMPLEX (zheterd の場合 )。 <i>a</i> ( <i>lda</i> ,*) は、 <i>uplo</i> の値に従って、行列 <i>A</i> の上三角部分または下三角部分を格納する配列である。 <i>a</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。  <i>work</i> ( <i>lwork</i> ) は、ワークスペース配列である。
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $\max(1, n)$ 以上。
<i>lwork</i>	INTEGER。配列 <i>work</i> のサイズ ( $lwork \geq n$ )。 <i>lwork</i> = -1 の場合、ワークスペースのクエリとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリとして返す。xerbla は <i>lwork</i> に関するエラー・メッセージを生成しない。  <i>lwork</i> の推奨値は、「アプリケーション・ノート」を参照。

## 出力パラメータ

<i>a</i>	<i>uplo</i> の値に従って、三重対角行列 <i>T</i> とユニタリ行列 <i>Q</i> の各成分によって上書きされる。
<i>d</i> , <i>e</i>	REAL (chetrd の場合 ) DOUBLE PRECISION (zheterd の場合 )。 配列： <i>d</i> (*) には、行列 <i>T</i> の対角成分が格納される。 <i>d</i> の次元は、 $\max(1, n)$ 以上でなければならない。  <i>e</i> (*) には、 <i>T</i> の非対角成分が格納される。 <i>e</i> の次元は、 $\max(1, n-1)$ 以上でなければならない。
<i>tau</i>	COMPLEX (chetrd の場合 ) DOUBLE COMPLEX (zheterd の場合 )。 配列、次元は $\max(1, n-1)$ 以上。 ユニタリ行列 <i>Q</i> の各成分が格納される。



`work(1)`      `info = 0` の場合、終了時に、最適なパフォーマンスを得るために必要な `lwork` の最小値が `work(1)` に格納される。以降の実行では、この `lwork` 値を使用する。

`info`          INTEGER。  
                  `info = 0` の場合、実行は正常に終了したことを示す。  
                  `info = -i` の場合、*i* 番目のパラメータの値が不正であったことを示す。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `hetrd` のインターフェイスの詳細を以下に示す。

`a`              サイズ  $(n, n)$  の行列 *A* を格納する。

`tau`            長さ  $(n-1)$  のベクトルを格納する。

`d`              長さ  $(n)$  のベクトルを格納する。

`e`              長さ  $(n-1)$  のベクトルを格納する。

`uplo`          'U' または 'L' でなければならない。デフォルト値は 'U'。

### アプリケーション・ノート

パフォーマンスを改善するには、`lwork = n * blocksize` に設定する。`blocksize` は、ブロック・アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる（通常は 16 ～ 64）。必要なワークスペースの大きさがわからない場合は、最初の実行では `lwork` を大きめの値に設定する。終了時に、`work(1)` の値を調べ、以降はその値を使用する。

計算で求めた行列 *T* は、行列 *A* + *E* の近似値である。ここで、 $\|E\|_2 = c(n)\varepsilon \|A\|_2$ 、*c(n)* は漸増する *n* の関数、 $\varepsilon$  はマシンによって異なる値である。

浮動小数演算のおおよその総数は、 $(16/3)n^3$  である。

このルーチンを呼び出した後は、以下のルーチンを呼び出せる。

[?ungtr](#)              計算で求める行列 *Q* を明示的に生成する場合

[?unmtr](#)            複素行列に *Q* を掛ける場合

このルーチンの実数版は、[?sytrd](#) である。

## ?ungtr

?hetrd で求めた複素ユニタリ行列  $Q$  を生成する。

### 構文

#### Fortran 77:

```
call cungr(uplo, n, a, lda, tau, work, lwork, info)
call zungr(uplo, n, a, lda, tau, work, lwork, info)
```

#### Fortran 95:

```
call ungtr(a, tau [,uplo] [,info])
```

このルーチンは、複素エルミート行列  $A$  を三重対角形式  $A = QTQ^H$  に縮退するために [?hetrd](#) によって求めた  $n \times n$  のユニタリ行列  $Q$  を明示的に生成する。このルーチンは、[?hetrd](#) を呼び出した後で使用する。

### 入力パラメータ

table {
uplo	CHARACTER\*1。'U' または 'L' でなければならない。 ?hetrd に指定した uplo と同じ値を使用する。
n	INTEGER。行列  $Q$  の次数 ( $n \geq 0$ )。
a, tau, work	COMPLEX (cungr の場合) DOUBLE COMPLEX (zungr の場合)。 配列: a(lda,\*) は、?hetrd から返された配列 a である。 a の第 2 次元は、max(1, n) 以上でなければならない。  tau(\*) は、?hetrd から返された配列 tau である。 tau の次元は、max(1, n-1) 以上でなければならない。  work(lwork) は、ワークスペース配列である。
lda	INTEGER。a の第 1 次元。max(1, n) 以上。
lwork	INTEGER。配列 work のサイズ ( $lwork \geq n$ )。  $lwork = -1$  の場合、ワークスペースのクエリとみなされ、ルーチンは work 配列の最適サイズだけを計算し、その値を work 配列の最初のエントリとして返す。xerbla は lwork に関するエラー・メッセージを生成しない。

*lwork* の推奨値は、「アプリケーション・ノート」を参照。

### 出力パラメータ

<i>a</i>	ユニタリ行列 $Q$ によって上書きされる。
<i>work</i> (1)	<i>info</i> = 0 の場合、終了時に、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が <i>work</i> (1) に格納される。以降の実行では、この <i>lwork</i> 値を使用する。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正であったことを示す。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `ungtr` のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ $(n, n)$ の行列 $A$ を格納する。
<i>tau</i>	長さ $(n-1)$ のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。

### アプリケーション・ノート

パフォーマンスを改善するには、 $lwork = (n-1) * blocksize$  に設定する。*blocksize* は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。

必要なワークスペースの大きさがわからない場合は、最初の実行では *lwork* を大きめの値に設定する。終了時に、*work*(1) の値を調べ、以降はその値を使用する。

計算で求めた  $Q$  は、正確なユニタリ行列と行列  $E$  ( $\|E\|_2 = O(\epsilon)$ ,  $\epsilon$  はマシン精度) だけ異なる。

浮動小数演算のおおよその総数は、 $(16/3)n^3$  である。

このルーチンの実数版は、[?orgtr](#) である。

### ?unmtr

複素行列に ?hetrd で求めた複素ユニタリ行列  $Q$  を掛ける。

---

#### 構文

##### Fortran 77:

```
call cunmtr(side, uplo, trans, m, n, a, lda, tau, c, ldc, work, lwork, info)
call zunmtr(side, uplo, trans, m, n, a, lda, tau, c, ldc, work, lwork, info)
```

##### Fortran 95:

```
call unmtr(a, tau, c [,side] [,uplo] [,trans] [,info])
```

#### 説明

このルーチンは、複素行列  $C$  に  $Q$  または  $Q^H$  を掛ける。 $Q$  は、複素エルミート行列  $A$  を三重対角形式  $A = QTQ^H$  に縮退するために [?hetrd](#) によって求めたユニタリ行列  $Q$  である。このルーチンは、?hetrd を呼び出した後で使用する。

このルーチンを使用すれば、パラメータ  $side$  と  $trans$  の値に従って、 $QC$ 、 $Q^HC$ 、 $CQ$ 、または  $CQ^H$  のいずれかの行列積を求めることができる ( $C$  の値は上書きされる)。

#### 入力パラメータ

下記の説明で、 $r$  は  $Q$  の次数を表す。

$side = 'L'$  の場合、 $r = m$ 、 $side = 'R'$  の場合、 $r = n$ 。

$side$	CHARACTER*1。'L' または 'R' でなければならない。 $side = 'L'$ の場合、 $Q$ または $Q^H$ は、 $C$ に左側から適用される。 $side = 'R'$ の場合、 $Q$ または $Q^H$ は、 $C$ に右側から適用される。
$uplo$	CHARACTER*1。'U' または 'L' でなければならない。 ?hetrd に指定した $uplo$ と同じ値を使用する。
$trans$	CHARACTER*1。'N' または 'T' でなければならない。 $trans = 'N'$ の場合、 $C$ に $Q$ を掛ける。 $trans = 'T'$ の場合、 $C$ に $Q^H$ を掛ける。
$m$	INTEGER。行列 $C$ の行数 ( $m \geq 0$ )。
$n$	INTEGER。行列 $C$ の列数 ( $n \geq 0$ )。

*a, work, tau, c*    COMPLEX (cunmtr の場合)  
                       DOUBLE COMPLEX (zunmtr の場合)。  
*a(lda,\*)* と *tau* は、?hetrd から返された配列である。  
*a* の第 2 次元は、 $\max(1, r)$  以上でなければならない。  
*tau* の次元は、 $\max(1, r-1)$  以上でなければならない。  
*c(ldc,\*)* には、行列 *C* を格納する。*c* の第 2 次元は、 $\max(1, n)$  以上でなければならない。  
*work(lwork)* は、ワークスペース配列である。

*lda*                INTEGER。 *a* の第 1 次元。  $lda \geq \max(1, r)$ 。

*ldc*                INTEGER。 *c* の第 1 次元。  $ldc \geq \max(1, n)$ 。

*lwork*             INTEGER。 配列 *work* のサイズ。 次の制約がある。  
                        $lwork \geq \max(1, n)$  (*side* = 'L' の場合)。  
                        $lwork \geq \max(1, m)$  (*side* = 'R' の場合)。  
                       *lwork* = -1 の場合、ワークスペースのクエリとみなされ、ルーチンは  
                       *work* 配列の最適サイズだけを計算し、その値を *work* 配列の最初のエン  
                       トリーとして返す。xerbla は *lwork* に関するエラー・メッセージを  
                       生成しない。  
                       *lwork* の推奨値は、「アプリケーション・ノート」を参照。

### 出力パラメータ

*c*                    (*side* と *trans* の値に従って)  $QC$ 、 $Q^H C$ 、 $CQ$ 、または  $CQ^H$  のいずれか  
                       の積によって上書きされる。

*work(1)*           *info* = 0 の場合、終了時に、最適なパフォーマンスを得るために必要  
                       な *lwork* の最小値が *work(1)* に格納される。以降の実行では、この  
                       *lwork* 値を使用する。

*info*                INTEGER。  
                       *info* = 0 の場合、実行は正常に終了したことを示す。  
                       *info* = -*i* の場合、*i* 番目のパラメータの値が不正であったことを示  
                       す。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `unmtr` のインターフェイスの詳細を以下に示す。

<code>a</code>	サイズ $(r, r)$ の行列 $A$ を格納する。 $r = m$ ( <code>side = 'L'</code> の場合)。 $r = n$ ( <code>side = 'R'</code> の場合)。
<code>tau</code>	長さ $(r-1)$ のベクトルを格納する。
<code>c</code>	サイズ $(m, n)$ の行列 $C$ を格納する。
<code>side</code>	'L' または 'R' でなければならない。デフォルト値は 'L'。
<code>uplo</code>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<code>trans</code>	'N' または 'C' でなければならない。デフォルト値は 'N'。

### アプリケーション・ノート

パフォーマンスを改善するには、 $lwork = n * blocksize$  (`side = 'L'` の場合) または  $lwork = m * blocksize$  (`side = 'R'` の場合) に設定する。`blocksize` は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。必要なワークスペースの大きさがわからない場合は、最初の実行では `lwork` を大きめの値に設定する。終了時に、`work(1)` の値を調べ、以降はその値を使用する。

計算で求めた積は、正確な積と行列  $E$  ( $\|E\|_2 = O(\varepsilon) \|C\|_2$ 、 $\varepsilon$  はマシン精度) だけ異なる。

浮動小数演算のおおよその総数は、 $8 * m^2 * n$  (`side = 'L'` の場合)、または  $8 * n^2 * m$  (`side = 'R'` の場合) である。

このルーチンの実数版は、[?ormtr](#) である。

---

## ?sptdr

圧縮格納形式の実対称行列を三重対角形式に縮退させる。

---

### 構文

#### Fortran 77:

```
call ssptdr(uplo, n, ap, d, e, tau, info)
call dsptdr(uplo, n, ap, d, e, tau, info)
```

**Fortran 95:**

```
call sptrd(a, tau [,uplo] [,info])
```

**説明**

このルーチンは、直交相似変換  $A = QTQ^T$  によって、圧縮格納形式の実対称行列  $A$  を対称三重対角形式  $T$  に縮退させる。直交行列  $Q$  は、明示的な形式では表現されず、 $(n-1)$  個の基本リフレクタの積として表現される。一連のルーチンでは、この形式で表現される  $Q$  を操作する (このセクションで後述)。

**入力パラメータ**

**uplo** CHARACTER\*1。'U' または 'L' でなければならない。  
**uplo**='U' の場合、**ap** に  $A$  の上三角行列 (圧縮形式) を格納する。  
**uplo**='L' の場合、**ap** に  $A$  の下三角行列 (圧縮形式) を格納する。

**n** INTEGER。行列  $A$  の次数 ( $n \geq 0$ )。

**ap** REAL (ssptrd の場合)  
 DOUBLE PRECISION (dsptdr の場合)。  
 配列、次元は  $\max(1, n(n+1)/2)$  以上。  
 (**uplo** の値に従って) 圧縮形式の  $A$  の上三角行列または下三角行列を格納する。

**出力パラメータ**

**ap** **uplo** の値に従って、三重対角行列  $T$  と直交行列  $Q$  の各成分によって上書きされる。

**d, e, tau** REAL (ssptrd の場合)  
 DOUBLE PRECISION (dsptdr の場合)。  
 配列:  
**d**(\*) には、行列  $T$  の対角成分が格納される。  
**d** の次元は、 $\max(1, n)$  以上でなければならない。  
**e**(\*) には、 $T$  の非対角成分が格納される。  
**e** の次元は、 $\max(1, n-1)$  以上でなければならない。  
**tau**(\*) には、行列  $Q$  の各成分が格納される。  
**tau** の次元は、 $\max(1, n-1)$  以上でなければならない。

**info** INTEGER。  
**info** = 0 の場合、実行は正常に終了したことを示す。  
**info** = - $i$  の場合、 $i$  番目のパラメータの値が不正であったことを示す。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `sptd` のインターフェイスの詳細を以下に示す。

<code>a</code>	Fortran 77 インターフェイスでの引数 <code>ap</code> を意味する。 サイズ $(n*(n+1)/2)$ の配列 <code>A</code> を格納する。
<code>tau</code>	長さ $(n-1)$ のベクトルを格納する。
<code>d</code>	長さ $(n)$ のベクトルを格納する。
<code>e</code>	長さ $(n-1)$ のベクトルを格納する。
<code>uplo</code>	'U' または 'L' でなければならない。デフォルト値は 'U'。

### アプリケーション・ノート

計算で求めた行列  $T$  は、行列  $A + E$  の近似値である。ここで、 $\|E\|_2 = c(n)\varepsilon \|A\|_2$ 、 $c(n)$  は漸増する  $n$  の関数、 $\varepsilon$  はマシンによって異なる値である。浮動小数演算のおおよその総数は、 $(4/3)n^3$  である。

このルーチンを呼び出した後は、以下のルーチンを呼び出せる。

[?opgtr](#) 計算で求める行列  $Q$  を明示的に生成する場合

[?opmtr](#) 実行列に  $Q$  を掛ける場合

このルーチンの複素数版は、[?hptd](#) である。

---

## ?opgtr

?sptd で求めた実直交行列  $Q$  を生成する。

---

### 構文

#### Fortran 77:

```
call sptd(uplo, n, ap, tau, q, ldq, work, info)
call dopgtr(uplo, n, ap, tau, q, ldq, work, info)
```



**Fortran 95:**

```
call opgtr(a, tau, q [,uplo] [,info])
```

**説明**

このルーチンは、圧縮形式の実対称行列  $A$  を三重対角行列の形式  $A = QTQ^T$  に縮退させるために [?sptdrd](#) によって求めた  $n \times n$  の直交行列  $Q$  を明示的に生成する。このルーチンは、[?sptdrd](#) を呼び出した後で使用する。

**入力パラメータ**

**uplo** CHARACTER\*1。'U' または 'L' でなければならない。  
[?sptdrd](#) に指定した **uplo** と同じ値を使用する。

**n** INTEGER。行列  $Q$  の次数 ( $n \geq 0$ )。

**ap, tau** REAL ([sopgtr](#) の場合 )  
 DOUBLE PRECISION ([dopgtr](#) の場合 )。  
 配列 **ap** と **tau** は、[?sptdrd](#) から返された配列である。  
**ap** の次元は、 $\max(1, n(n+1)/2)$  以上でなければならない。  
**tau** の次元は、 $\max(1, n-1)$  以上でなければならない。

**ldq** INTEGER。出力配列 **q** の第 1 次元は、 $\max(1, n)$  以上でなければならない。

**work** REAL ([sopgtr](#) の場合 )  
 DOUBLE PRECISION ([dopgtr](#) の場合 )。  
 ワークスペース配列、次元は  $\max(1, n-1)$  以上。

**出力パラメータ**

**q** REAL ([sopgtr](#) の場合 )  
 DOUBLE PRECISION ([dopgtr](#) の場合 )。  
 配列、次元は (**ldq**, \*)。  
 計算で求めた行列  $Q$  が格納される。  
**q** の第 2 次元は、 $\max(1, n)$  以上でなければならない。

**info** INTEGER。  
**info** = 0 の場合、実行は正常に終了したことを示す。  
**info** = -*i* の場合、*i* 番目のパラメータの値が不正であったことを示す。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `opgtr` のインターフェイスの詳細を以下に示す。

<code>a</code>	Fortran 77 インターフェイスでの引数 <code>ap</code> を意味する。 サイズ $(n*(n+1)/2)$ の配列 $A$ を格納する。
<code>tau</code>	長さ $(n-1)$ のベクトルを格納する。
<code>q</code>	サイズ $(n,n)$ の行列 $Q$ を格納する。
<code>uplo</code>	'U' または 'L' でなければならない。デフォルト値は 'U'。

### アプリケーション・ノート

計算で求めた  $Q$  は、正確な直交行列と行列  $E$  ( $\|E\|_2 = O(\epsilon)$ 、 $\epsilon$  はマシン精度) だけ異なる。

浮動小数演算のおおよその総数は、 $(4/3)n^3$  である。

このルーチンの複素数版は、[?upgtr](#) である。

---

## ?opmtr

実行列に `?sptdr` で求めた実直交行列  $Q$  を掛ける。

---

### 構文

#### Fortran 77:

```
call sopmtr(side, uplo, trans, m, n, ap, tau, c, ldc, work, info)
call dopmtr(side, uplo, trans, m, n, ap, tau, c, ldc, work, info)
```

#### Fortran 95:

```
call opmtr(a, tau, c [,side] [,uplo] [,trans] [,info])
```

**説明**

このルーチンは、実行列  $C$  に  $Q$  または  $Q^T$  を掛ける。 $Q$  は、圧縮形式の実対称行列  $A$  を三重対角形式  $A = QTQ^T$  に縮退させるために [?sptdrd](#) によって求めた直交行列  $Q$  である。このルーチンは、[?sptdrd](#) を呼び出した後で使用する。

このルーチンを使用すれば、パラメータ  $side$  と  $trans$  の値に従って、 $QC$ 、 $Q^TC$ 、 $CQ$ 、または  $CQ^T$  のいずれかの行列積を求めることができる ( $C$  の値は上書きされる)。

**入力パラメータ**

下記の説明で、 $r$  は  $Q$  の次数を表す。

$side = 'L'$  の場合、 $r = m$ 、 $side = 'R'$  の場合、 $r = n$ 。

$side$	CHARACTER*1。'L' または 'R' でなければならない。 $side = 'L'$ の場合、 $Q$ または $Q^T$ は、 $C$ に左側から適用される。 $side = 'R'$ の場合、 $Q$ または $Q^T$ は、 $C$ に右側から適用される。
$uplo$	CHARACTER*1。'U' または 'L' でなければならない。 <a href="#">?sptdrd</a> に指定した $uplo$ と同じ値を使用する。
$trans$	CHARACTER*1。'N' または 'T' でなければならない。 $trans = 'N'$ の場合、 $C$ に $Q$ を掛ける。 $trans = 'T'$ の場合、 $C$ に $Q^T$ を掛ける。
$m$	INTEGER。行列 $C$ の行数 ( $m \geq 0$ )。
$n$	INTEGER。行列 $C$ の列数 ( $n \geq 0$ )。
$ap, work, tau, c$	REAL ( <a href="#">sopmtr</a> の場合 ) DOUBLE PRECISION ( <a href="#">dopmtr</a> の場合 )。 $ap$ と $tau$ は、 <a href="#">?sptdrd</a> から返された配列である。 $ap$ の次元は、 $\max(1, r(r+1)/2)$ 以上でなければならない。 $tau$ の次元は、 $\max(1, r-1)$ 以上でなければならない。 $c(ldc, *)$ には、行列 $C$ を格納する。 $c$ の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $work(*)$ は、ワークスペース配列である。 $work$ の次元は、 $\max(1, n)$ 以上 ( $side = 'L'$ の場合 ) または $\max(1, m)$ 以上 ( $side = 'R'$ の場合 ) でなければならない。
$ldc$	INTEGER。 $c$ の第 1 次元。 $ldc \geq \max(1, n)$ 。

## 出力パラメータ

<i>c</i>	( <i>side</i> と <i>trans</i> の値に従って) $QC$ 、 $Q^TC$ 、 $CQ$ 、または $CQ^T$ のいずれかの積によって上書きされる。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正であったことを示す。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `opmtr` のインターフェイスの詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ap</i> を意味する。 サイズ $(r*(r+1)/2)$ の配列 <i>A</i> を格納する。ここで $r = m$ ( <i>side</i> = 'L' の場合)、 $r = n$ ( <i>side</i> = 'R' の場合)。
<i>tau</i>	長さ $(r-1)$ のベクトルを格納する。
<i>c</i>	サイズ $(m,n)$ の行列 <i>C</i> を格納する。
<i>side</i>	'L' または 'R' でなければならない。デフォルト値は 'L'。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>trans</i>	'N'、'C'、または 'T' でなければならない。デフォルト値は 'N'。

## アプリケーション・ノート

計算で求めた積は、正確な積と行列  $E$  ( $\|E\|_2 = O(\epsilon) \|C\|_2$ 、 $\epsilon$  はマシン精度) だけ異なる。

浮動小数演算のおおよその総数は、 $2*m^2*n$  (*side* = 'L' の場合)、または  $2*n^2*m$  (*side* = 'R' の場合) である。

このルーチンの複素数版は、[?upmtr](#) である。

## ?hptrd

圧縮格納形式の複素エルミート行列を  
三重対角形式に縮退させる。

### 構文

#### Fortran 77:

```
call chptrd(uplo, n, ap, d, e, tau, info)
call zhptrd(uplo, n, ap, d, e, tau, info)
```

#### Fortran 95:

```
call hptrd(a, tau [,uplo] [,info])
```

### 説明

このルーチンは、ユニタリ相似変換  $A = QTQ^H$  によって、圧縮形式の複素エルミート行列  $A$  を対称三重対角形式  $T$  に縮退させる。ユニタリ行列  $Q$  は、明示的な形式では表現されず、 $(n-1)$  個の基本リフレクタの積として表現される。一連のルーチンでは、この形式で表現される  $Q$  を操作する (このセクションで後述)。

### 入力パラメータ

<code>uplo</code>	CHARACTER*1。'U' または 'L' でなければならない。 <code>uplo = 'U'</code> の場合、 <code>ap</code> に $A$ の上三角行列 (圧縮形式) を格納する。 <code>uplo = 'L'</code> の場合、 <code>ap</code> に $A$ の下三角行列 (圧縮形式) を格納する。
<code>n</code>	INTEGER。行列 $A$ の次数 ( $n \geq 0$ )。
<code>ap</code>	COMPLEX (chptrd の場合) DOUBLE COMPLEX (zhptrd の場合)。 配列、次元は $\max(1, n(n+1)/2)$ 以上。 ( <code>uplo</code> の値に従って) 圧縮形式の $A$ の上三角行列または下三角行列を格納する。

### 出力パラメータ

<code>ap</code>	<code>uplo</code> の値に従って、三重対角行列 $T$ と直交行列 $Q$ の各成分によって上書きされる。
-----------------	---

<i>d, e</i>	REAL (chptrd の場合 ) DOUBLE PRECISION (zhptrd の場合 )。 配列 : <i>d</i> (*) には、行列 <i>T</i> の対角成分が格納される。 <i>d</i> の次元は、 $\max(1, n)$ 以上でなければならない。  <i>e</i> (*) には、 <i>T</i> の非対角成分が格納される。 <i>e</i> の次元は、 $\max(1, n-1)$ 以上でなければならない。
<i>tau</i>	COMPLEX (chptrd の場合 ) DOUBLE COMPLEX (zhptrd の場合 )。 配列、次元は $\max(1, n-1)$ 以上。 直交行列 <i>Q</i> の各成分が格納される。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正であったことを示す。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン hptrd のインターフェイスの詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ap</i> を意味する。 サイズ $(n*(n+1)/2)$ の配列 <i>A</i> を格納する。
<i>tau</i>	長さ $(n-1)$ のベクトルを格納する。
<i>d</i>	長さ $(n)$ のベクトルを格納する。
<i>e</i>	長さ $(n-1)$ のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。

## アプリケーション・ノート

計算で求めた行列 *T* は、行列 *A* + *E* の近似値である。ここで、 $\|E\|_2 = c(n)\epsilon \|A\|_2$ 、*c*(*n*) は漸増する *n* の関数、 $\epsilon$  はマシンによって異なる値である。

浮動小数演算のおおよその総数は、 $(16/3)n^3$  である。

このルーチンを呼び出した後は、以下のルーチンを呼び出せる。

[?upgtr](#) 計算で求める行列  $Q$  を明示的に生成する場合

[?upmtr](#) 複素行列に  $Q$  を掛ける場合

このルーチンの実数版は、[?sptrd](#) である。

## ?upgtr

?hptrd で求めた複素ユニタリ行列  $Q$  を生成する。

### 構文

#### Fortran 77:

```
call cupgtr(uplo, n, ap, tau, q, ldq, work, info)
call zupgtr(uplo, n, ap, tau, q, ldq, work, info)
```

#### Fortran 95:

```
call upgtr(a, tau, q [,uplo] [,info])
```

### 説明

このルーチンは、圧縮形式の複素エルミート行列  $A$  を三重対角形式  $A = QTQ^H$  に縮退するために [?hptrd](#) によって求めた  $n \times n$  のユニタリ行列  $Q$  を明示的に生成する。このルーチンは、?hptrd を呼び出した後で使用する。

### 入力パラメータ

<code>uplo</code>	CHARACTER*1。'U' または 'L' でなければならない。 ?sptrd に指定した <code>uplo</code> と同じ値を使用する。
<code>n</code>	INTEGER。行列 $Q$ の次数 ( $n \geq 0$ )。
<code>ap, tau</code>	COMPLEX (cupgtr の場合 ) DOUBLE COMPLEX (zupgtr の場合 )。 配列 <code>ap</code> と <code>tau</code> は、?hptrd から返された配列である。 <code>ap</code> の次元は、 $\max(1, n(n+1)/2)$ 以上でなければならない。 <code>tau</code> の次元は、 $\max(1, n-1)$ 以上でなければならない。

<i>ldq</i>	INTEGER。出力配列 <i>q</i> の第 1 次元は、 $\max(1, n)$ 以上でなければならない。
<i>work</i>	COMPLEX (cupgtr の場合 ) DOUBLE COMPLEX (zupgtr の場合 )。 ワークスペース配列、次元は $\max(1, n-1)$ 以上。

## 出力パラメータ

<i>q</i>	COMPLEX (cupgtr の場合 ) DOUBLE COMPLEX (zupgtr の場合 )。 配列、次元は ( <i>ldq</i> , *)。 計算で求めた行列 <i>Q</i> が格納される。 <i>q</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正であったことを示す。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン upgtr のインターフェイスの詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ap</i> を意味する。 サイズ $(n*(n+1)/2)$ の配列 <i>A</i> を格納する。
<i>tau</i>	長さ $(n-1)$ のベクトルを格納する。
<i>q</i>	サイズ $(n, n)$ の行列 <i>Q</i> を格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。

## アプリケーション・ノート

計算で求めた *Q* は、正確な直交行列と行列 *E* ( $\|E\|_2 = O(\epsilon)$ 、 $\epsilon$  はマシン精度) だけ異なる。

浮動小数演算のおおよその総数は、 $(16/3)n^3$  である。

このルーチンの実数版は、[?opgtr](#) である。



## ?upmtr

複素行列に ?hptrd で求めたユニタリ行列  $Q$  を掛ける。

### 構文

#### Fortran 77:

```
call cupmtr(side, uplo, trans, m, n, ap, tau, c, ldc, work, info)
call zupmtr(side, uplo, trans, m, n, ap, tau, c, ldc, work, info)
```

#### Fortran 95:

```
call upmtr(a, tau, c [,side] [,uplo] [,trans] [,info])
```

### 説明

このルーチンは、圧縮形式の複素行列  $C$  と  $Q$  または  $Q^H$  を掛ける。 $Q$  は、複素エルミート行列  $A$  を三重対角形式  $A = QTQ^H$  に縮退するために [?hptrd](#) によって求めたユニタリ行列  $Q$  である。このルーチンは、?hptrd を呼び出した後で使用する。

このルーチンを使用すれば、パラメータ  $side$  と  $trans$  の値に従って、 $QC$ 、 $Q^HC$ 、 $CQ$ 、または  $CQ^H$  のいずれかの行列積を求めることができる ( $C$  の値は上書きされる)。

### 入力パラメータ

下記の説明で、 $r$  は  $Q$  の次数を表す。

$side = 'L'$  の場合、 $r = m$ 、 $side = 'R'$  の場合、 $r = n$ 。

$side$	CHARACTER*1。'L' または 'R' でなければならない。 $side = 'L'$ の場合、 $Q$ または $Q^H$ は、 $C$ に左側から適用される。 $side = 'R'$ の場合、 $Q$ または $Q^H$ は、 $C$ に右側から適用される。
$uplo$	CHARACTER*1。'U' または 'L' でなければならない。 ?hptrd に指定した $uplo$ と同じ値を使用する。
$trans$	CHARACTER*1。'N' または 'T' でなければならない。 $trans = 'N'$ の場合、 $C$ に $Q$ を掛ける。 $trans = 'T'$ の場合、 $C$ に $Q^H$ を掛ける。
$m$	INTEGER。行列 $C$ の行数 ( $m \geq 0$ )。
$n$	INTEGER。行列 $C$ の列数 ( $n \geq 0$ )。

*ap, tau, c, work* COMPLEX (cupmtr の場合)  
 DOUBLE COMPLEX (zupmtr の場合)。  
*ap* と *tau* は、?hptrd から返された配列である。  
*ap* の次元は、 $\max(1, r(r+1)/2)$  以上でなければならない。  
*tau* の次元は、 $\max(1, r-1)$  以上でなければならない。  
*c(ldc,\*)* には、行列 *C* を格納する。*c* の第 2 次元は、 $\max(1, n)$  以上でなければならない。  
*work(\*)* は、ワークスペース配列である。  
*work* の次元は、 $\max(1, n)$  以上 (*side* = 'L' の場合) または  $\max(1, m)$  以上 (*side* = 'R' の場合) でなければならない。  
*ldc* INTEGER。 *c* の第 1 次元。  $ldc \geq \max(1, n)$ 。

## 出力パラメータ

*c* (*side* と *trans* の値に従って)  $QC$ 、 $Q^H C$ 、 $CQ$ 、または  $CQ^H$  のいずれかの積によって上書きされる。  
*info* INTEGER。  
*info* = 0 の場合、実行は正常に終了したことを示す。  
*info* = -*i* の場合、*i* 番目のパラメータの値が不正であったことを示す。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン upmtr のインターフェイスの詳細を以下に示す。

*a* Fortran 77 インターフェイスでの引数 *ap* を意味する。  
 サイズ  $(r*(r+1)/2)$  の配列 *A* を格納する。ここで  
 $r = m$  (*side* = 'L' の場合)、  
 $r = n$  (*side* = 'R' の場合)。  
*tau* 長さ  $(r-1)$  のベクトルを格納する。  
*c* サイズ  $(m, n)$  の行列 *C* を格納する。  
*side* 'L' または 'R' でなければならない。デフォルト値は 'L'。  
*uplo* 'U' または 'L' でなければならない。デフォルト値は 'U'。  
*trans* 'N' または 'C' でなければならない。デフォルト値は 'N'。

## アプリケーション・ノート

計算で求めた積は、正確な積と行列  $E$  ( $\|E\|_2 = O(\varepsilon) \|C\|_2$ ,  $\varepsilon$  はマシン精度) だけ異なる。

浮動小数演算のおおよその総数は、 $8 \cdot m^2 \cdot n$  ( $side = 'L'$  の場合)、または  $8 \cdot n^2 \cdot m$  ( $side = 'R'$  の場合) である。

このルーチンの実数版は、[?opmtr](#) である。

## ?sbtrd

実対称帯行列を三重対角形式に縮退させる。

### 構文

#### Fortran 77:

```
call ssbtrd(vect, uplo, n, kd, ab, ldab, d, e, q, ldq, work, info)
call dsbtrd(vect, uplo, n, kd, ab, ldab, d, e, q, ldq, work, info)
```

#### Fortran 95:

```
call sbtrd(a [,q] [,vect] [,uplo] [,info])
```

### 説明

このルーチンは、直交相似変換  $A = QTQ^T$  によって、実対称帯行列  $A$  を対称三重対角形式  $T$  に縮退させる。直交行列  $Q$  は、Givens 回転行列の積として得られる。そのため、必要に応じて、このルーチンを使用して、行列  $Q$  を明示的に求めることができる。

### 入力パラメータ

<b>vect</b>	CHARACTER*1。'V' または 'N' でなければならない。 vect = 'V' の場合、行列 $Q$ が明示的に返される。 vect = 'N' の場合、 $Q$ は返されない。
<b>uplo</b>	CHARACTER*1。'U' または 'L' でなければならない。 uplo = 'U' の場合、ab に $A$ の上三角部分を格納する。 uplo = 'L' の場合、ab に $A$ の下三角部分を格納する。
<b>n</b>	INTEGER。行列 $A$ の次数 ( $n \geq 0$ )。
<b>kd</b>	INTEGER。 $A$ の優対角成分または劣対角成分の数 ( $kd \geq 0$ )。

*ab, work* REAL (ssbtrd の場合 )  
DOUBLE PRECISION (dsbtrd の場合 )。  
*ab(ldab,\*)* は、(*uplo* の値に従って ) 行列 *A* の上三角部分または下三角部分を帯形式で格納する配列である。  
*ab* の第 2 次元は、 $\max(1, n)$  以上でなければならない。  
*work(\*)* は、ワークスペース配列である。  
*work* の次元は、 $\max(1, n)$  以上でなければならない。

*ldab* INTEGER。 *ab* の第 1 次元。  $kd+1$  以上。

*ldq* INTEGER。 *q* の第 1 次元。 次の制約がある。  
 $ldq \geq \max(1, n)$  (*vect* = 'V' の場合 )。  
 $ldq \geq 1$  (*vect* = 'N' の場合 )。

## 出力パラメータ

*ab* 終了時に、*ab* は上書きされる。

*d, e, q* REAL (ssbtrd の場合 )  
DOUBLE PRECISION (dsbtrd の場合 )。  
配列：  
*d(\*)* には、行列 *T* の対角成分が格納される。  
*d* の次元は、 $\max(1, n)$  以上でなければならない。  
*e(\*)* には、*T* の非対角成分が格納される。  
*e* の次元は、 $\max(1, n-1)$  以上でなければならない。  
*vect* = 'N' の場合、*q(ldq,\*)* は参照されない。  
*vect* = 'V' の場合、*q* には  $n \times n$  の行列 *Q* が格納される。  
*q* の第 2 次元は、 $\max(1, n)$  以上 (*vect* = 'V' の場合 ) または 1 以上 (*vect* = 'N' の場合 ) でなければならない。

*info* INTEGER。  
*info* = 0 の場合、実行は正常に終了したことを示す。  
*info* = -*i* の場合、*i* 番目のパラメータの値が不正であったことを示す。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン sbtrd のインターフェイスの詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ab</i> を意味する。サイズ $(kd+1, n)$ の配列 <i>A</i> を格納する。
<i>q</i>	サイズ $(n, n)$ の行列 <i>Q</i> を格納する。
<i>d</i>	長さ $(n)$ のベクトルを格納する。
<i>e</i>	長さ $(n-1)$ のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>vect</i>	省略された場合、この引数は、引数 <i>q</i> の存在に基づいて以下のように復元される。 <i>vect</i> = 'V' ( <i>q</i> が存在する場合)、 <i>vect</i> = 'N' ( <i>q</i> が省略された場合)。 存在する場合、 <i>vect</i> は 'V' または 'U' と等しくなければならず、引数 <i>q</i> も存在しなければならない。 <i>vect</i> が存在し、 <i>q</i> が省略された場合、エラー条件がセットされる。

### アプリケーション・ノート

計算で求めた行列 *T* は、行列  $A + E$  の近似値である。ここで、 $\|E\|_2 = c(n)\varepsilon \|A\|_2$ 、 $c(n)$  は漸増する  $n$  の関数、 $\varepsilon$  はマシンによって異なる値である。計算で求めた行列 *Q* は、正確な直交行列と行列 *E* ( $\|E\|_2 = O(\varepsilon)$ ) だけ異なる。

浮動小数演算のおおよその総数は、*vect* = 'N' の場合、 $6n^2 * kd$  である。*vect* = 'V' の場合、 $3n^3 * (kd-1)/kd$  だけ余分な演算が必要になる。

このルーチンの複素数版は、[?hbtrd](#) である。

---

## ?hbtrd

複素エルミート帯行列を三重対角形式に縮退させる。

---

### 構文

#### Fortran 77:

```
call chbtrd(vect, uplo, n, kd, ab, ldab, d, e, q, ldq, work, info)
call zhbtrd(vect, uplo, n, kd, ab, ldab, d, e, q, ldq, work, info)
```

## Fortran 95:

```
call hbttrd(a [,q] [,vect] [,uplo] [,info])
```

## 説明

このルーチンは、ユニタリ相似変換  $A = QTQ^H$  によって、複素エルミート帯行列  $A$  を対称三重対角形式  $T$  に縮退させる。ユニタリ行列  $Q$  は、Givens 回転行列の積として得られる。そのため、必要に応じて、このルーチンを使用して、行列  $Q$  を明示的に求めることができる。

## 入力パラメータ

<b>vect</b>	CHARACTER*1。'V' または 'N' でなければならない。 vect = 'V' の場合、行列 $Q$ が明示的に返される。 vect = 'N' の場合、 $Q$ は返されない。
<b>uplo</b>	CHARACTER*1。'U' または 'L' でなければならない。 uplo = 'U' の場合、 $ab$ に $A$ の上三角部分を格納する。 uplo = 'L' の場合、 $ab$ に $A$ の下三角部分を格納する。
<b>n</b>	INTEGER。行列 $A$ の次数 ( $n \geq 0$ )。
<b>kd</b>	INTEGER。 $A$ の優対角成分または劣対角成分の数 ( $kd \geq 0$ )。
<b>ab, work</b>	COMPLEX (chbttrd の場合 ) DOUBLE COMPLEX (zhbttrd の場合 )。 $ab(ldab,*)$ は、(uplo の値に従って) 行列 $A$ の上三角部分または下三角部分を帯形式で格納する配列である。 $ab$ の第 2 次元は、 $\max(1, n)$ 以上でなければならない。  $work(*)$ は、ワークスペース配列である。 $work$ の次元は、 $\max(1, n)$ 以上でなければならない。
<b>ldab</b>	INTEGER。 $ab$ の第 1 次元。 $kd+1$ 以上。
<b>ldq</b>	INTEGER。 $q$ の第 1 次元。次の制約がある。 $ldq \geq \max(1, n)$ (vect = 'V' の場合 )。 $ldq \geq 1$ (vect = 'N' の場合 )。

## 出力パラメータ

<b>ab</b>	終了時に、 $ab$ は上書きされる。
-----------	---------------------

<i>d, e</i>	REAL (chbtrd の場合 ) DOUBLE PRECISION (zhbtrd の場合 )。 配列： <i>d</i> (*) には、行列 <i>T</i> の対角成分が格納される。 <i>d</i> の次元は、 $\max(1, n)$ 以上でなければならない。  <i>e</i> (*) には、 <i>T</i> の非対角成分が格納される。 <i>e</i> の次元は、 $\max(1, n-1)$ 以上でなければならない。
<i>q</i>	COMPLEX (chbtrd の場合 ) DOUBLE COMPLEX (zhbtrd の場合 )。 配列、次元は ( <i>ldq</i> ,*)。 <i>vect</i> = 'N' の場合、 <i>q</i> は参照されない。 <i>vect</i> = 'V' の場合、 <i>q</i> には $n \times n$ の行列 <i>Q</i> が格納される。 <i>q</i> の第 2 次元は、 $\max(1, n)$ 以上 ( <i>vect</i> = 'V' の場合 ) または 1 以上 ( <i>vect</i> = 'N' の場合 ) でなければならない。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正であったことを示す。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン hbtrd のインターフェイスの詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ab</i> を意味する。サイズ ( <i>kd</i> +1, <i>n</i> ) の配列 <i>A</i> を格納する。
<i>q</i>	サイズ ( <i>n</i> , <i>n</i> ) の行列 <i>Q</i> を格納する。
<i>d</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>e</i>	長さ ( <i>n</i> -1) のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>vect</i>	省略された場合、この引数は、引数 <i>q</i> の存在に基づいて以下のように復元される。 <i>vect</i> = 'V' ( <i>q</i> が存在する場合 )、 <i>vect</i> = 'N' ( <i>q</i> が省略された場合 )。

存在する場合、`vect` は 'V' または 'U' と等しくなければならず、引数  $q$  も存在しなければならない。  
`vect` が存在し、 $q$  が省略された場合、エラー条件がセットされる。

### アプリケーション・ノート

計算で求めた行列  $T$  は、行列  $A + E$  の近似値である。ここで、 $\|E\|_2 = c(n)\varepsilon \|A\|_2$ 、 $c(n)$  は漸増する  $n$  の関数、 $\varepsilon$  はマシンによって異なる値である。計算で求めた行列  $Q$  は、正確な直交行列と行列  $E$  ( $\|E\|_2 = O(\varepsilon)$ ) だけ異なる。

浮動小数演算のおおよその総数は、`vect = 'N'` の場合、 $20n^2 * kd$  である。`vect = 'V'` の場合、 $10n^3 * (kd-1)/kd$  だけ余分な演算が必要になる。

このルーチンの実数版は、[?sbtrd](#) である。

---

## ?sterf

*QR* アルゴリズムを使用して、実対称三重対角行列の固有値をすべて計算する。

---

### 構文

#### Fortran 77:

```
call ssterf(n, d, e, info)
call dsterf(n, d, e, info)
```

#### Fortran 95:

```
call sterf(d, e [,info])
```

### 説明

このルーチンは、実対称三重対角行列  $T$  (対称 / エルミート行列を対角形式に縮退させて得られる) の固有値をすべて計算する。このルーチンは、平方根なしの *QR* アルゴリズムを使用する。

固有値だけでなく固有ベクトルも必要な場合は、[?stegr](#) を使用する。

### 入力パラメータ

$n$                       INTEGER。行列  $T$  の次数 ( $n \geq 0$ )。



*d*, *e*                REAL (sssterf の場合 )  
                       DOUBLE PRECISION (dsterf の場合 )。  
                       配列：  
                       *d*(\*) には、*T* の対角成分を格納する。  
                       *d* の次元は、 $\max(1, n)$  以上でなければならない。  
                       *e*(\*) には、*T* の非対角成分が格納される。  
                       *e* の次元は、 $\max(1, n-1)$  以上でなければならない。

### 出力パラメータ

*d*                    *n* 個の固有値が、昇順に格納される (*info* > 0 でない場合 )。  
                       *info* も参照のこと。

*e*                    終了時に上書きされる。 *info* を参照。

*info*                INTEGER。  
                       *info* = 0 の場合、実行は正常に終了したことを示す。  
                       *info* = *i* の場合、このアルゴリズムで  $30n$  回の処理を繰り返した結果、  
                       すべての固有値を見つけられなかった (*i* 個の非対角成分がゼロに収束  
                       していない) ことを示す。終了時に、*d* と *e* にそれぞれ、三重対角行  
                       列 ( 直交性が *T* に近似している ) の対角成分と非対角成分が格納され  
                       る。  
                       *info* = -*i* の場合、*i* 番目のパラメータの値が不正であったことを示  
                       す。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの  
 引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する  
 詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `sterf` のインターフェイスの詳細を以下に示す。

*d*                    長さ (*n*) のベクトルを格納する。

*e*                    長さ (*n*-1) のベクトルを格納する。

### アプリケーション・ノート

計算で求めた固有値と固有ベクトルは、正確には、 $\|E\|_2 = O(\varepsilon) \|T\|_2$  のような行列  $T + E$   
 ( $\varepsilon$  はマシン精度) のものである。

$\lambda_i$  が正確な固有値で、 $\mu_i$  が対応する計算で求めた値の場合、次のようになる。

$$|\mu_i - \lambda_i| \leq c(n)\varepsilon \|T\|_2$$

$c(n)$  は、 $n$  の漸増関数である。

浮動小数演算の総数は、このアルゴリズムの収束の速さによって決まる。一般に、約  $14n^2$  である。

---

### ?steqr

三重対角形式に縮退させた対称/エルミート  
行列の固有値と固有ベクトルをすべて計算する  
(QR アルゴリズム)。

---

#### 構文

##### Fortran 77:

```
call ssteqr(compz, n, d, e, z, ldz, work, info)
call dsteqr(compz, n, d, e, z, ldz, work, info)
call csteqr(compz, n, d, e, z, ldz, work, info)
call zsteqr(compz, n, d, e, z, ldz, work, info)
```

##### Fortran 95:

```
call rsteqr(d, e [,z] [,compz] [,info])
call steqr(d, e [,z] [,compz] [,info])
```

#### 説明

このルーチンは、実対称三重対角行列  $T$  の固有値と (オプションで) 固有ベクトルをすべて計算する。すなわち、スペクトル因子分解  $T = \mathbf{Z}\mathbf{\Lambda}\mathbf{Z}^T$  を計算できる。 $\mathbf{\Lambda}$  は、対角成分が固有値  $\lambda_i$  である対角行列で、 $\mathbf{Z}$  は列が固有ベクトルである直交行列である。つまり、次のように表現できる。

$$Tz_i = \lambda_i z_i \text{ for } i = 1, 2, \dots, n$$

(このルーチンでは、 $\|z_i\|_2 = 1$  となるように、固有ベクトルを正規化する。)

このルーチンを使用すれば、三重対角形式  $T (A = \mathbf{Q}T\mathbf{Q}^H)$  に縮退させた任意の実対称 (または複素エルミート) 行列  $A$  の固有値と固有ベクトルも計算できる。その場合、スペクトル因子分解は、 $A = \mathbf{Q}T\mathbf{Q}^H = (\mathbf{Q}\mathbf{Z})\mathbf{\Lambda}(\mathbf{Q}\mathbf{Z})^H$  になる。`?steqr` を呼び出す場合は、事前に  $A$  を三重対角形式に縮退させ、次のルーチンを呼び出すことによって行列  $\mathbf{Q}$  を明示的に生成しなければならない。

	実行列の場合	複素行列の場合
フル格納形式	?sytrd, ?orgtr	?hetrd, ?ungtr
圧縮格納形式	?sptrd, ?opgtr	?hptrd, ?upgtr
帯格納形式	?sbtrd (vect='V')	?hbtrd (vect='V')

固有値のみ必要な場合は、[?sterf](#) を呼び出した方が効率は良くなる。 $T$  が正定値の場合、[?steqr](#) よりも [?pteqr](#) の方が、小さな固有値を正確に計算できる。

1 つのルーチン呼び出しでこの問題を解くには、分割統治ルーチンを使用する。すなわち、実対称行列の場合、[?stevd](#)、[?syevd](#)、[?spevd](#)、または [?sbevd](#) のいずれか、複素エルミート行列の場合、[?heevd](#)、[?hpevd](#)、または [?hbevd](#) のいずれかを使用する。

## 入力パラメータ

<i>compz</i>	CHARACTER*1。'N'、'I'、または 'V' のいずれかでなければならない。 <i>compz</i> = 'N' の場合、固有値のみを計算する。 <i>compz</i> = 'I' の場合、三重対角行列 $T$ の固有値と固有ベクトルを計算する。 <i>compz</i> = 'V' の場合、 $A$ の固有値と固有ベクトルを計算する（呼び出し時に、配列 $z$ には、あらかじめ行列 $Q$ を格納しておかなければならない）。
<i>n</i>	INTEGER。行列 $T$ の次数 ( $n \geq 0$ )。
<i>d,e,work</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 配列： <i>d</i> (*) には、 $T$ の対角成分を格納する。 <i>d</i> の次元は、 $\max(1, n)$ 以上でなければならない。  <i>e</i> (*) には、 $T$ の非対角成分が格納される。 <i>e</i> の次元は、 $\max(1, n-1)$ 以上でなければならない。  <i>work</i> (*) は、ワークスペース配列である。 <i>work</i> の次元は、1 以上 ( <i>compz</i> = 'N' の場合) または $\max(1, 2*n-2)$ 以上 ( <i>compz</i> = 'V' または 'I' の場合) でなければならない。
<i>z</i>	REAL (ssteqr の場合) DOUBLE PRECISION (dsteqr の場合) COMPLEX (csteqr の場合) DOUBLE COMPLEX (zsteqr の場合)。 配列、次元は ( <i>ldz</i> , *)。 <i>compz</i> = 'N' または 'I' の場合、 $z$ を設定する必要はない。

$vect = 'V'$  の場合、 $z$  には  $n \times n$  の行列  $Q$  を格納しなければならない。  
 $z$  の第 2 次元は、1 以上 ( $compz = 'N'$  の場合) または  $\max(1, n)$  以上 ( $compz = 'V'$  または  $'I'$  の場合) でなければならない。

$work(lwork)$  は、ワークスペース配列である。

$ldz$             INTEGER。  $z$  の第 1 次元。次の制約がある。  
 $ldz \geq 1$  ( $compz = 'N'$  の場合)。  
 $ldz \geq \max(1, n)$  ( $compz = 'V'$  または  $'I'$  の場合)。

### 出力パラメータ

$d$              $n$  個の固有値が、昇順に格納される ( $info > 0$  でない場合)。  
 $info$  も参照のこと。

$e$             終了時に上書きされる。 $info$  を参照。

$z$              $info = 0$  の場合、 $n$  個の正規直交固有ベクトルが、列ごとに格納される ( $i$  番目の列は、 $i$  番目の固有値に対応する)。

$info$             INTEGER。  
 $info = 0$  の場合、実行は正常に終了したことを示す。  
 $info = i$  の場合、このアルゴリズムで  $30n$  回の処理を繰り返した結果、すべての固有値を見つけられなかった ( $i$  個の非対角成分がゼロに収束していない) ことを示す。終了時に、 $d$  と  $e$  にそれぞれ、三重対角行列 (直交性が  $T$  に近似している) の対角成分と非対角成分が格納される。  
 $info = -i$  の場合、 $i$  番目のパラメータの値が不正であったことを示す。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `steqr` のインターフェイスの詳細を以下に示す。

$d$             長さ ( $n$ ) のベクトルを格納する。

$e$             長さ ( $n-1$ ) のベクトルを格納する。

$z$             サイズ ( $n, n$ ) の行列  $Z$  を格納する。

`compz` 省略された場合、この引数は、引数  $z$  の存在に基づいて以下のように復元される。  
`compz = 'I'` ( $z$  が存在する場合)、  
`compz = 'N'` ( $z$  が省略された場合)。  
 存在する場合、`compz` は 'I' または 'V' と等しくなければならず、引数  $z$  も存在しなければならない。  
`compz` が存在し、 $z$  が省略された場合、エラー条件がセットされる。

$z$  が省略されると、実数および複素数の場合に曖昧な選択が行われるため、`steqr` ルーチンには 2 つの Fortran 95 インターフェイスが必要な点に注意する。したがって、実数の場合 (単精度または倍精度) は `rsteqr`、複素数の場合 (単精度または倍精度) は `steqr` を使用する。

### アプリケーション・ノート

計算で求めた固有値と固有ベクトルは、正確には、 $\|E\|_2 = O(\epsilon) \|T\|_2$  のような行列  $T + E$  ( $\epsilon$  はマシン精度) のものである。

$\lambda_i$  が正確な固有値で、 $\mu_i$  が対応する計算で求めた値の場合、次のようになる。

$$|\mu_i - \lambda_i| \leq c(n)\epsilon \|T\|_2$$

$c(n)$  は、 $n$  の漸増関数である。

$z_i$  が対応する正確な固有ベクトルで、 $w_i$  が対応する計算ベクトルの場合、それらの間の角度  $\theta(z_i, w_i)$  は次のようになる。 $\theta(z_i, w_i) \leq c(n)\epsilon \|T\|_2 / \min_{i \neq j} |\lambda_i - \lambda_j|$

浮動小数演算の総数は、このアルゴリズムの収束の速さによって決まる。通常は、ほぼ次の値になる。

$24n^2$  (`compz = 'N'` の場合)。

$7n^3$  (複素数型の場合、 $14n^3$ ) (`compz = 'V'` または 'I' の場合)。

### ?stedc

分割統治法を使用して、対称三重行列の固有値と固有ベクトルをすべて計算する。

---

#### 構文

##### Fortran 77:

```
call sstedc(compz, n, d, e, z, ldz, work, lwork, iwork, liwork, info)
call dstedc(compz, n, d, e, z, ldz, work, lwork, iwork, liwork, info)
call cstedc(compz, n, d, e, z, ldz, work, lwork, rwork, lrwork,
            iwork, liwork, info)
call zstedc(compz, n, d, e, z, ldz, work, lwork, rwork, lrwork,
            iwork, liwork, info)
```

##### Fortran 95:

```
call rstedc(d, e [,z] [,compz] [,info])
call stedc(d, e [,z] [,compz] [,info])
```

#### 説明

このルーチンは、分割統治法を使用して、対称三重行列の固有値と（オプションで）固有ベクトルをすべて計算する。

フル形式または帯形式の実対称行列または複素エルミート行列の固有ベクトルは、[?sytrd/?hetrd](#)、[?sptird/?hptrd](#)、[?sbtrd/?hbtrd](#) のいずれかを使用して、この行列を三重対角形式に縮退させて得ることもできる。

この関数で使用する [?laed0](#)、[?laed1](#)、[?laed2](#)、[?laed3](#)、[?laed4](#)、[?laed5](#)、[?laed6](#)、[?laed7](#)、[?laed8](#)、[?laed9](#)、および [?laeda](#) を参照。

#### 入力パラメータ

**compz** CHARACTER\*1。'N'、'I'、または 'V' のいずれかでなければならない。  
compz = 'N' の場合、固有値のみを計算する。  
compz = 'I' の場合、三重対角行列の固有値と固有ベクトルを計算する。  
compz = 'V' の場合、元の対称 / エルミート行列の固有値と固有ベクトル

ルを計算する。呼び出し時に、配列  $z$  には、元の行列を三重対角形式に縮退させるのに使用した直交 / ユニタリ行列が格納されていなければならない。

$n$	INTEGER。対称三重対角行列の次数 ( $n \geq 0$ )。
$d, e, rwork$	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 配列： $d(*)$ には、三重対角行列の対角成分を格納する。 $d$ の次元は、 $\max(1, n)$ 以上でなければならない。 $e(*)$ には、三重対角行列の劣対角成分を格納する。 $e$ の次元は、 $\max(1, n-1)$ 以上でなければならない。 $rwork(lrwork)$ は、複素数型でのみ使用されるワークスペース配列である。
$z, work$	REAL (sstedc の場合) DOUBLE PRECISION (dstedc の場合) COMPLEX (cstedc の場合) DOUBLE COMPLEX (zstedc の場合)。 配列: $z(ldz, *)$ , $work(*)$ 。 $compz = 'V'$ の場合、呼び出し時に、配列 $z$ には、元の行列を三重対角形式に縮退させるのに使用した直交 / ユニタリ行列が格納されていなければならない。 $z$ の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $work(lwork)$ は、ワークスペース配列である。
$ldz$	INTEGER。 $z$ の第 1 次元。次の制約がある。 $ldz \geq 1$ ( $compz = 'N'$ の場合)。 $ldz \geq \max(1, n)$ ( $compz = 'V'$ または $'I'$ の場合)。
$lwork$	INTEGER。配列 $work$ の次元。 $lwork = -1$ の場合、ワークスペースのクエリとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエントリとして返す。 $xerbla$ は $lwork$ に関するエラー・メッセージを生成しない。 $lwork$ に必要な値は、「アプリケーション・ノート」を参照。
$lrwork$	INTEGER。配列 $rwork$ の次元 (複素数型でのみ使用される)。 $lrwork = -1$ の場合、ワークスペースのクエリとみなされ、ルーチンは $rwork$ 配列の最適サイズだけを計算し、その値を $rwork$ 配列の最初の

エントリとして返す。xerbla は *lwork* に関するエラー・メッセージを生成しない。

*lwork* に必要な値は、「アプリケーション・ノート」を参照。

*iwork*                    INTEGER。ワークスペース配列、次元は (*liwork*)。

*liwork*                  INTEGER。配列 *iwork* の次元。  
*liwork* = -1 の場合、ワークスペースのクエリとみなされ、ルーチンは *iwork* 配列の最適サイズだけを計算し、その値を *iwork* 配列の最初のエントリとして返す。xerbla は *liwork* に関するエラー・メッセージを生成しない。  
*liwork* に必要な値は、「アプリケーション・ノート」を参照。

### 出力パラメータ

*d*                        *n* 個の固有値が、昇順に格納される (*info* ≠ 0 でない場合)。  
*info* も参照のこと。

*e*                        終了時に上書きされる。*info* を参照。

*z*                        *info* = 0 の場合、*compz* = 'V' であれば、元の対称 / エルミート行列の正規直交固有ベクトルが *z* に格納され、*compz* = 'I' であれば、対称三重対角行列の正規直交固有ベクトルが *z* に格納される。*compz* = 'N' であれば、*z* は参照されない。

*work(1)*                終了時に、*info* = 0 の場合、*work(1)* に最適な *lwork* 値が返される。

*rwork(1)*               終了時に、*info* = 0 の場合、*rwork(1)* に最適な *lrwork* 値が返される (複素数型の場合のみ)。

*iwork(1)*               終了時に、*info* = 0 の場合、*iwork(1)* に最適な *liwork* 値が返される。

*info*                    INTEGER。  
*info* = 0 の場合、実行は正常に終了したことを示す。  
*info* = -*i* の場合、*i* 番目のパラメータの値が不正であったことを示す。*info* = *i* の場合、このアルゴリズムが、*i*/(*n*+1) から mod(*i*, *n*+1) までの行と列からなる部分行列の操作中に、固有値の計算に失敗したことを示す。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。



ルーチン `stedc` のインターフェイスの詳細を以下に示す。

<code>d</code>	長さ ( $n$ ) のベクトルを格納する。
<code>e</code>	長さ ( $n-1$ ) のベクトルを格納する。
<code>z</code>	サイズ ( $n, n$ ) の行列 $Z$ を格納する。
<code>compz</code>	省略された場合、この引数は、引数 $z$ の存在に基づいて以下のように復元される。 <code>compz = 'I'</code> ( $z$ が存在する場合)、 <code>compz = 'N'</code> ( $z$ が省略された場合)。 存在する場合、 <code>compz</code> は ' <code>I</code> ' または ' <code>V</code> ' と等しくなければならず、引数 $z$ も存在しなければならない。 <code>compz</code> が存在し、 $z$ が省略された場合、エラー条件がセットされる。

$z$  と `work` が省略されると、実数および複素数の場合に曖昧な選択が行われるため、`stedc` ルーチンには 2 つの Fortran 95 インターフェイスが必要な点に注意する。したがって、実数の場合 (単精度または倍精度) は `rstedc`、複素数の場合 (単精度または倍精度) は `zstedc` を使用する。

## アプリケーション・ノート

ワークスペース配列に必要なサイズは次のようになる。

`sstedc/dstedc` の場合

`compz = 'N'` または  $n \leq 1$  の場合、`lwork` は 1 以上でなければならない。  
`compz = 'V'` および  $n > 1$  の場合、`lwork` は  $(1 + 3n + 2n \cdot \lg n + 3n^2)$  以上でなければならない。ここで、 $\lg(n)$  は  $2^k \geq n$  となる最小の整数  $k$  である。

`compz = 'I'` および  $n > 1$  の場合、`lwork` は  $(1 + 4n + n^2)$  以上でなければならない。

`compz = 'N'` または  $n \leq 1$  の場合、`liwork` は 1 以上でなければならない。  
`compz = 'V'` および  $n > 1$  の場合、`liwork` は  $(6 + 6n + 5n \cdot \lg n)$  以上でなければならない。  
`compz = 'I'` および  $n > 1$  の場合、`liwork` は  $(3 + 5n)$  以上でなければならない。

`cstedc/zstedc` の場合

`compz = 'N'` または '`I`' の場合、または  $n \leq 1$  の場合、`lwork` は 1 以上でなければならない。

`compz = 'V'` および  $n > 1$  の場合、`lwork` は  $n^2$  以上でなければならない。

`compz = 'N'` または  $n \leq 1$  の場合、`lwork` は 1 以上でなければならない。  
`compz = 'V'` および  $n > 1$  の場合、`lwork` は  $(1 + 3n + 2n \cdot \lg n + 3n^2)$  以上でなければならない。ここで、 $\lg(n)$  は  $2^k \geq n$  となるような最小の整数  $k$  である。

`compz = 'I'` および  $n > 1$  の場合、`lwork` は  $(1 + 4n + 2n^2)$  以上でなければならない。

複素数型の場合に `liwork` に必要な値は、実数型の場合と同じである。

---

### ?stegr

実対称三重対角行列の固有値と ( オプションで )  
固有ベクトルを選択的に計算する。

---

#### 構文

##### Fortran 77:

```
call sstegr(jobz, range, n, d, e, vl, vu, il, iu, abstol, m, w, z,  
           ldz, isuppz, work, lwork, iwork, liwork, info)  
call dstegr(jobz, range, n, d, e, vl, vu, il, iu, abstol, m, w, z,  
           ldz, isuppz, work, lwork, iwork, liwork, info)  
call cstegr(jobz, range, n, d, e, vl, vu, il, iu, abstol, m, w, z,  
           ldz, isuppz, work, lwork, iwork, liwork, info)  
call zstegr(jobz, range, n, d, e, vl, vu, il, iu, abstol, m, w, z,  
           ldz, isuppz, work, lwork, iwork, liwork, info)
```

##### Fortran 95:

```
call rstegr(d, e, w [,z] [,vl] [,vu] [,il] [,iu] [,m] [,isuppz] [,abstol]  
           [,info])  
call stegr(d, e, w [,z] [,vl] [,vu] [,il] [,iu] [,m] [,isuppz] [,abstol]  
           [,info])
```

#### 説明

このルーチンは、実対称三重対角行列  $T$  の固有値と ( オプションで ) 固有ベクトルを選択的に計算する。固有値と固有ベクトルを選択するには、希望する固有値の値の範囲かインデックスの範囲を指定する。固有値は *dqds* アルゴリズムで計算するが、直交固有

ベクトルは種々の「良好な」 $LDL^T$  表現（比較的安定した表現とも呼ばれる）で計算する。グラム-シュミットの直交化はできる限り避ける。アルゴリズムの各ステップを詳しく説明すると次のようになる。 $T$  のまだ縮退されていない  $i$  番目のブロックに対して、

- (a)  $L_i D_i L_i^T$  が比較的安定な表現になるように、 $T - s_i = L_i D_i L_i^T$  を計算する。
- (b)  $dqds$  アルゴリズムによって、 $L_i D_i L_i^T$  の固有値  $\lambda_j$  を高い相対精度で計算する。
- (c) 値が近い固有値からなるクラスタが存在する場合は、そのクラスタに近い  $s_i$  を「選択」し、ステップ (a) に戻る。
- (d)  $L_i D_i L_i^T$  の近似的な固有値  $\lambda_j$  に対する固有ベクトルを、階数表示のツイスト分解によって求める。

出力値に必要な精度は、入力パラメータ `abstol` で指定できる。

このルーチンで使用される補助ルーチン [?lasq2](#)、[?lasq5](#)、および [?lasq6](#) を参照。

### 入力パラメータ

<code>jobz</code>	CHARACTER*1。'N' または 'V' でなければならない。 <code>job='N'</code> の場合、固有値のみを計算する。 <code>job='V'</code> の場合、固有値と固有ベクトルを計算する。
<code>range</code>	CHARACTER*1。'A'、'V'、または 'I' のいずれかでなければならない。 <code>range='A'</code> の場合、固有値をすべて計算する。 <code>range='V'</code> の場合、半開区間 $v_l < \lambda_i \leq v_u$ における固有値 $\lambda_i$ を計算する。 <code>range='I'</code> の場合、ルーチンはインデックスが $i_l \sim i_u$ である固有値を計算する。
<code>n</code>	INTEGER。行列 $T$ の次数 ( $n \geq 0$ )。
<code>d, e, work</code>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 配列： <code>d(*)</code> には、 $T$ の対角成分を格納する。 $d$ の次元は、 $\max(1, n)$ 以上でなければならない。 <code>e(*)</code> の $1 \sim n-1$ の成分に、 $T$ の劣対角成分を格納する。 <code>e(n)</code> に値を設定する必要はない。 $e$ の次元は、 $\max(1, n)$ 以上でなければならない。 <code>work(lwork)</code> は、ワークスペース配列である。

<i>vl, vu</i>	<p>REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 <i>range</i> = 'V' の場合、固有値を検索する区間の上限と下限。 次の制約がある。 <math>vl &lt; vu</math>。</p> <p><i>range</i> = 'A' または 'I' の場合、<i>vl</i> と <i>vu</i> は参照されない。</p>
<i>il, iu</i>	<p>INTEGER。 <i>range</i> = 'I' の場合、返される最小固有値と最大固有値に対する昇順のインデックス。 次の制約がある。 <math>1 \leq il \leq iu \leq n</math> (<math>n &gt; 0</math> の場合)。 <math>il=1</math> かつ <math>iu=0</math> (<math>n=0</math> の場合)。</p> <p><i>range</i> = 'A' または 'V' の場合、<i>il</i> と <i>iu</i> は参照されない。</p>
<i>abstol</i>	<p>REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 各固有値と固有ベクトルに許される絶対許容値。 <i>jobz</i> = 'V' の場合、出力される固有値と固有ベクトルの誤差ノルムが <i>abstol</i> 以内になる。また、異なる固有ベクトル間の内積も <i>abstol</i> 以内になる。<math>abstol &lt; n\epsilon\ T\ _1</math> の場合、<math>n\epsilon\ T\ _1</math> が代わりに使用される (<math>\epsilon</math> はマシン精度)。固有値は、<i>abstol</i> とは無関係に、<math>\epsilon\ T\ _1</math> の精度で計算される。高い相対精度が必要な場合、<i>abstol</i> を <a href="#">?lamch</a> (「安全な最小値」) に設定する。</p>
<i>ldz</i>	<p>INTEGER。出力配列 <i>z</i> のリーディング・ディメンション。 次の制約がある。 <math>ldz \geq 1</math> (<i>jobz</i> = 'N' の場合)。 <math>ldz \geq \max(1, n)</math> (<i>jobz</i> = 'V' の場合)。</p>
<i>lwork</i>	<p>INTEGER。配列 <i>work</i> の次元。 <math>lwork \geq \max(1, 18n)</math>。 <i>lwork</i> = -1 の場合、ワークスペースのクエリとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリとして返す。<i>xerbla</i> は <i>lwork</i> に関するエラー・メッセージを生成しない。</p>
<i>iwork</i>	<p>INTEGER。 ワークスペース配列、次元は (<i>liwork</i>)。</p>
<i>liwork</i>	<p>INTEGER。配列 <i>iwork</i> の次元。 <math>liwork \geq \max(1, 10n)</math>。 <i>liwork</i> = -1 の場合、ワークスペースのクエリとみなされ、ルーチンは</p>

*iwork* 配列の最適サイズだけを計算し、その値を *iwork* 配列の最初のエントリとして返す。xerbla は *liwork* に関するエラー・メッセージを生成しない。

### 出力パラメータ

<i>d, e</i>	終了時に、 <i>d</i> と <i>e</i> は上書きされる。
<i>m</i>	INTEGER。検出された固有値の総数。( $0 \leq m \leq n$ )。 <i>range</i> = 'A' の場合、 <i>m</i> = <i>n</i> 、 <i>range</i> = 'I' の場合、 <i>m</i> = <i>iu-il</i> +1。
<i>w</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 配列、次元は $\max(1, n)$ 以上。 選択された固有値が昇順に <i>w</i> (1) ~ <i>w</i> ( <i>m</i> ) に格納される。
<i>z</i>	REAL (sstegr の場合) DOUBLE PRECISION (dstegr の場合) COMPLEX (cstegr の場合) DOUBLE COMPLEX (zstegr の場合)。 配列 <i>z</i> ( <i>ldz</i> , *)。 <i>z</i> の第 2 次元は、 $\max(1, m)$ 以上でなければならない。  <i>jobz</i> = 'V' の場合、 <i>info</i> = 0 であれば、 <i>z</i> の先頭の <i>m</i> 列に、行列 <i>T</i> の選択された固有値に対応する正規直交固有ベクトルが格納される。すなわち、 <i>w</i> ( <i>i</i> ) に対応する固有ベクトルが、 <i>z</i> の <i>i</i> 番目の列に格納される。 <i>jobz</i> = 'N' の場合、 <i>z</i> は参照されない。 注：配列 <i>z</i> には、 $\max(1, m)$ 以上の列が提供されなければならない。 <i>range</i> = 'V' の場合、 <i>m</i> の正確な値が事前にわからないため、上限値を使用する必要がある。
<i>isuppz</i>	INTEGER。 配列、次元は $2 * \max(1, m)$ 以上。  <i>z</i> に格納されている固有ベクトルのサポート情報、すなわち <i>z</i> に格納されている非ゼロの成分を示すインデックス。 <i>i</i> 番目の固有ベクトルは、 <i>isuppz</i> (2 <i>i</i> -1) から <i>isuppz</i> (2 <i>i</i> ) までの成分のみ非ゼロである。
<i>work</i> (1)	終了時に、 <i>info</i> = 0 の場合、 <i>work</i> (1) は <i>lwork</i> の必要最小サイズを返す。
<i>iwork</i> (1)	終了時に、 <i>info</i> = 0 の場合、 <i>iwork</i> (1) は <i>liwork</i> の必要最小サイズを返す。

*info*                    INTEGER。  
*info* = 0 の場合、実行は正常に終了したことを示す。  
*info* = -*i* の場合、*i* 番目のパラメータの値が不正であったことを示す。  
*info* = 1 の場合、slarre で内部エラーが発生したことを示す。  
*info* = 2 の場合、?larrv で内部エラーが発生したことを示す。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *stegr* のインターフェイスの詳細を以下に示す。

<i>d</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>e</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>w</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>z</i>	サイズ ( <i>n,m</i> ) の行列 <i>Z</i> を格納する。
<i>isuppz</i>	長さ (2*m) のベクトルを格納する。
<i>v1</i>	この引数のデフォルト値は、 <i>v1</i> = -HUGE( <i>v1</i> ) であり、ここで HUGE( <i>a</i> ) は引数 <i>a</i> と同じ精度のマシン最大値を意味する。
<i>vu</i>	この引数のデフォルト値は、 <i>vu</i> = HUGE( <i>v1</i> ) である。
<i>i1</i>	この引数のデフォルト値は、 <i>i1</i> = 1 である。
<i>iu</i>	この引数のデフォルト値は、 <i>iu</i> = <i>n</i> である。
<i>abstol</i>	この引数のデフォルト値は、 <i>abstol</i> = 0.0_WP である。
<i>jobz</i>	引数 <i>z</i> の存在に基づいて以下のように復元される。 <i>jobz</i> = 'V' ( <i>z</i> が存在する場合)、 <i>jobz</i> = 'N' ( <i>z</i> が省略された場合)。
<i>range</i>	引数 <i>v1</i> 、 <i>vu</i> 、 <i>i1</i> 、および <i>iu</i> の存在に基づいて以下のように復元される。 <i>range</i> = 'V' ( <i>v1</i> と <i>vu</i> のいずれかまたは両方が存在する場合)、 <i>range</i> = 'I' ( <i>i1</i> と <i>iu</i> のいずれかまたは両方が存在する場合)、 <i>range</i> = 'A' ( <i>v1</i> 、 <i>vu</i> 、 <i>i1</i> 、および <i>iu</i> がすべて存在しない場合)。 <i>v1</i> と <i>vu</i> のいずれかまたは両方が存在し、同時に <i>i1</i> と <i>iu</i> のいずれかまたは両方が存在する場合、エラー条件がセットされる。

$z$  が省略されると、実数および複素数の場合に曖昧な選択が行われるため、stegr ルーチンには 2 つの Fortran 95 インターフェイスが必要な点に注意する。したがって、実数の場合 (単精度または倍精度) は rstegr、複素数の場合 (単精度または倍精度) は stegr を使用する。

### アプリケーション・ノート

現在、?stegr は、 $O(n^2)$  時間内で  $T$  のすべての固有値  $n$  と固有ベクトルを見つけ出すように設定されている。すなわち、`range='A'` のみサポートしている。

現在、上記のステップ (c) で適切な  $\sigma_i$  が選択できない場合、ルーチン ?stein が呼び出される。固有値同士が近い値であれば、?stein が変形グラム-シュミットを呼び出す。

?stegr は、無限大と NaN の処理方法が、IEEE-754 の浮動小数点標準に準拠しているマシンでのみ正しく機能する。通常の ?stegr 実行で無限大と NaN が発生するため、IEEE-754 標準に準拠しない環境で実行すると、浮動小数点例外によって異常終了する可能性がある。

---

## ?pteqr

実対称正定値三重対角行列の固有値と (オプションで) 固有ベクトルをすべて計算する。

---

### 構文

#### Fortran 77:

```
call spteqr(compz, N, d, e, z, ldz, work, info)
call dpteqr(compz, N, d, e, z, ldz, work, info)
call cpteqr(compz, N, d, e, z, ldz, work, info)
call zpteqr(compz, N, d, e, z, ldz, work, info)
```

#### Fortran 95:

```
call rpteqr(d, e [,z] [,compz] [,info])
call pteqr(d, e [,z] [,compz] [,info])
```

## 説明

このルーチンは、実対称正定値三重対角行列  $T$  の固有値と (オプションで) 固有ベクトルをすべて計算する。すなわち、スペクトル因子分解  $T = Z\Lambda Z^T$  を計算できる。 $\Lambda$  は、対角成分が固有値  $\lambda_i$  である対角行列で、 $Z$  は列が固有ベクトルである直交行列である。つまり、次のように表現できる。

$$Tz_i = \lambda_i z_i \text{ for } i = 1, 2, \dots, n$$

(このルーチンでは、 $\|z_i\|_2 = 1$  となるように、固有ベクトルを正規化する。)

このルーチンを使用すれば、三重対角形式  $T (A = QTQ^H)$  に縮退させた実対称 (または複素エルミート) 正定値行列  $A$  の固有値と固有ベクトルも計算できる。その場合、スペクトル因子分解は、 $A = QTQ^H = (QZ)\Lambda(QZ)^H$  になる。`?pteqr` を呼び出す場合は、事前に  $A$  を三重対角形式に縮退させ、次のルーチンを読み出すことによって行列  $Q$  を明示的に生成しなければならない。

	実行列の場合	複素行列の場合
フル格納形式	<code>?sytrd, ?orgtr</code>	<code>?hetrd, ?ungtr</code>
圧縮格納形式	<code>?sptrd, ?opgtr</code>	<code>?hptrd, ?upgtr</code>
帯格納形式	<code>?sbtrd (vect='V')</code>	<code>?hbtrd (vect='V')</code>

このルーチンは、最初に、 $T$  を  $LDL^H$  として因子分解する ( $L$  は下側の単位二重対角行列で、 $D$  は対角行列である)。次に、二重対角行列  $B = LD^{1/2}$  を求めた後、`?bdsqr` を呼び出して  $B$  の特異値 ( $T$  の固有値と同じ) を求める。

## 入力パラメータ

<code>compz</code>	CHARACTER*1。'N'、'I'、または 'V' のいずれかでなければならない。 <code>compz = 'N'</code> の場合、固有値のみを計算する。 <code>compz = 'I'</code> の場合、三重対角行列 $T$ の固有値と固有ベクトルを計算する。 <code>compz = 'V'</code> の場合、 $A$ の固有値と固有ベクトルを計算する (呼び出し時に、配列 $z$ には、あらかじめ行列 $Q$ を格納しておかなければならない)。
<code>n</code>	INTEGER。行列 $T$ の次数 ( $n \geq 0$ )。
<code>d,e,work</code>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 配列: $d(*)$ には、 $T$ の対角成分を格納する。 $d$ の次元は、 $\max(1, n)$ 以上でなければならない。



	$e(*)$ には、 $T$ の非対角成分が格納される。 $e$ の次元は、 $\max(1, n-1)$ 以上でなければならない。
	$work(*)$ は、ワークスペース配列である。 $work$ の次元は、1 以上 ( $compz = 'N'$ の場合) または $\max(1, 2*n-2)$ 以上 ( $compz = 'V'$ または $'I'$ の場合) でなければならない。
$z$	REAL (spteqr の場合) DOUBLE PRECISION (dpteqr の場合) COMPLEX (cpteqr の場合) DOUBLE COMPLEX (zpteqr の場合)。 配列、次元は $(ldz, *)$ 。 $compz = 'N'$ または $'I'$ の場合、 $z$ を設定する必要はない。 $vect = 'V'$ の場合、 $z$ には $n \times n$ の行列 $Q$ を格納しなければならない。 $z$ の第 2 次元は、1 以上 ( $compz = 'N'$ の場合) または $\max(1, n)$ 以上 ( $compz = 'V'$ または $'I'$ の場合) でなければならない。
$ldz$	INTEGER。 $z$ の第 1 次元。次の制約がある。 $ldz \geq 1$ ( $compz = 'N'$ の場合)。 $ldz \geq \max(1, n)$ ( $compz = 'V'$ または $'I'$ の場合)。

### 出力パラメータ

$d$	$n$ 個の固有値が、降順に格納される ( $info > 0$ でない場合)。 $info$ も参照のこと。
$e$	終了時に上書きされる。
$z$	$info = 0$ の場合、 $n$ 個の正規直交固有ベクトルが、列ごとに格納される ( $i$ 番目の列は、 $i$ 番目の固有値に対応する)。
$info$	INTEGER。 $info = 0$ の場合、実行は正常に終了したことを示す。 $info = i$ の場合、先頭から次数 $i$ の小行列 (したがって $T$ そのもの) は正定値ではないことを示す。 $info = n + i$ の場合、特異値を計算するためのアルゴリズムが収束していない ( $i$ 個の非対角成分がゼロに収束していない) ことを示す。 $info = -i$ の場合、 $i$ 番目のパラメータの値が不正であったことを示す。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `ptegr` のインターフェイスの詳細を以下に示す。

<code>d</code>	長さ ( $n$ ) のベクトルを格納する。
<code>e</code>	長さ ( $n-1$ ) のベクトルを格納する。
<code>z</code>	サイズ ( $n, n$ ) の行列 $Z$ を格納する。
<code>compz</code>	省略された場合、この引数は、引数 $z$ の存在に基づいて以下のように復元される。 <code>compz = 'I'</code> ( $z$ が存在する場合)、 <code>compz = 'N'</code> ( $z$ が省略された場合)。 存在する場合、 <code>compz</code> は ' <code>I</code> ' または ' <code>V</code> ' と等しくなければならず、引数 $z$ も存在しなければならない。 <code>compz</code> が存在し、 $z$ が省略された場合、エラー条件がセットされる。

$z$  が省略されると、実数および複素数の場合に曖昧な選択が行われるため、`ptegr` ルーチンには 2 つの Fortran 95 インターフェイスが必要な点に注意する。したがって、実数の場合 (単精度または倍精度) は `rptegr`、複素数の場合 (単精度または倍精度) は `ptegr` を使用する。

## アプリケーション・ノート

$\lambda_i$  が正確な固有値で、 $\mu_i$  が対応する計算で求めた値の場合、次のようになる。

$$|\mu_i - \lambda_i| \leq c(n)\varepsilon K \lambda_i$$

$c(n)$  は  $n$  の漸増する関数、 $\varepsilon$  はマシン精度、 $K = \|DTD\|_2 \|(DTD)^{-1}\|_2$ 、および  $D$  は  $d_{ii} = t_{ii}^{-1/2}$  の対角成分である。

$z_i$  が対応する正確な固有ベクトルで、 $w_i$  が対応する計算で求めたベクトルの場合、それらの間の角度  $\theta(z_i, w_i)$  は次のようになる。 $\theta(u_i, w_i) \leq c(n)\varepsilon K / \min_{i \neq j} (|\lambda_i - \lambda_j| / |\lambda_i + \lambda_j|)$

$\min_{i \neq j} (|\lambda_i - \lambda_j| / |\lambda_i + \lambda_j|)$  は、 $\lambda_i$  ともう一方の固有値との間の相対誤差である。

浮動小数演算の総数は、このアルゴリズムの収束の速さによって決まる。通常は、ほぼ次の値になる。

$30n^2$  (`compz = 'N'` の場合)。

$6n^3$  (複素数型の場合、 $12n^3$ ) (`compz = 'V'` または '`I`' の場合)。

## ?stebz

実対称三重対角行列の選択された固有値を二分法で計算する。

### 構文

#### Fortran 77:

```
call sstebz (range, order, n, vl, vu, il, iu, abstol,
             d, e, m, nsplit, w, iblock, isplit, work, iwork, info)
call dstebz (range, order, n, vl, vu, il, iu, abstol,
             d, e, m, nsplit, w, iblock, isplit, work, iwork, info)
```

#### Fortran 95:

```
call stebz(d, e, m, nsplit, w, iblock, isplit [,order] [,vl] [,vu] [,il] [,iu]
           [,abstol] [,info])
```

### 説明

このルーチンは、実対称三重対角行列  $T$  の固有値の一部 (または全部) を、二分法によって計算する。このルーチンでは、ゼロまたは無視できる程度の非対角成分を検索し、 $T$  をブロック - 対角形式  $T = \text{diag}(T_1, T_2, \dots)$  に分解できるかどうかを調べる。次に、 $T_i$  の各ブロックに対して二分法を実行し、計算で求めた各固有値のブロックのインデックスを返す。そのため、[?stein](#) を呼び出した後に、続けてこのブロック構造を利用できる。

[?laebz](#) も参照のこと。

### 入力パラメータ

<code>range</code>	CHARACTER*1。'A'、'V'、または 'I' のいずれかでなければならない。 <code>range = 'A'</code> の場合、固有値をすべて計算する。 <code>range = 'V'</code> の場合、半開区間 $vl < \lambda_i \leq vu$ における固有値 $\lambda_i$ を計算する。 <code>range = 'I'</code> の場合、ルーチンはインデックスが $il \sim iu$ である固有値を計算する。
<code>order</code>	CHARACTER*1。'B' または 'E' でなければならない。

$order='B'$  の場合、一連の固有値は、分解された各ブロック内で最小のものから昇順に配置される。

$order='E'$  の場合、固有値が行列全体で最小のものから昇順に配置される。

$n$	INTEGER。行列 $T$ の次数 ( $n \geq 0$ )。
$vl, vu$	REAL (sstebz の場合 ) DOUBLE PRECISION (dstebz の場合 )。 $range='V'$ の場合、半開区間 $vl < \lambda_i \leq vu$ における固有値 $\lambda_i$ を計算する。  $range='A'$ または $'I'$ の場合、 $vl$ と $vu$ は参照されない。
$il, iu$	INTEGER。次の制約がある。 $1 \leq il \leq iu \leq n$ 。 $range='I'$ の場合、 $il \leq i \leq iu$ となるような固有値 $\lambda_i$ を計算する (固有値 $\lambda_i$ は昇順とする)。  $range='A'$ または $'V'$ の場合、 $il$ と $iu$ は参照されない。
$abstol$	REAL (sstebz の場合 ) DOUBLE PRECISION (dstebz の場合 )。 各固有値に対する絶対許容値は、必ず指定しなければならない。固有値 (または集積値) は、幅 $abstol$ の区間内に存在している場合に、収束しているものとみなされる。 $abstol \leq 0.0$ の場合、許容値は $\epsilon \ T\ _1$ ( $\epsilon$ はマシン精度) になる。
$d, e, work$	REAL (sstebz の場合 ) DOUBLE PRECISION (dstebz の場合 )。 配列: $d(*)$ には、 $T$ の対角成分を格納する。 $d$ の次元は、 $\max(1, n)$ 以上でなければならない。  $e(*)$ には、 $T$ の非対角成分が格納される。 $e$ の次元は、 $\max(1, n-1)$ 以上でなければならない。  $work(*)$ は、ワークスペース配列である。 $work$ の次元は、 $\max(1, 4n)$ 以上でなければならない。
$iwork$	INTEGER。ワークスペース 配列、次元は $\max(1, 3n)$ 以上。

## 出力パラメータ

$m$	INTEGER。見つかった固有値の実際の個数。
$nsplit$	INTEGER。 $T$ で見つかった対角ブロックの個数。

<i>w</i>	REAL (sstebz の場合 ) DOUBLE PRECISION (dstebz の場合 )。 配列、次元は $\max(1, n)$ 以上。 計算で求めた固有値が、 $w(1) \sim w(m)$ に格納される。
<i>iblock, isplit</i>	INTEGER。 配列、次元は $\max(1, n)$ 以上。 正値 $iblock(i)$ は、 $w(i)$ に格納されている固有値のブロック番号である ( <i>info</i> を参照)。 <i>isplit</i> の先頭から <i>nsplit</i> 個の成分には、 ブロック $T_1$ に $1 \sim isplit(1)$ の行列を格納し、 ブロック $T_2$ に $isplit(1)+1 \sim isplit(2)$ の行列を格納するというように、 $T$ をブロック $T_i$ に分割した点が格納される。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = 1 の場合、 <i>range</i> = 'A' または 'V' であれば、所定の精度で計算できなかった固有値がある。 $iblock(i) < 0$ の場合、 $w(i)$ に格納されている固有値が収束していない。 <i>info</i> = 2 の場合、 <i>range</i> = 'I' であれば、計算できなかった固有値がある。 <i>range</i> = 'A' と指定して、このルーチンを再度呼び出すこと。 <i>info</i> = 3 の場合、 <i>range</i> = 'A' または 'V' であれば、 <i>info</i> = 1 と同じである。 <i>range</i> = 'I' であれば、 <i>info</i> = 2 と同じである。 <i>info</i> = 4 の場合、固有値を計算することはできない。コンピュータ上で、浮動小数演算が期待通りに動作していない。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正であったことを示す。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *stebz* のインターフェイスの詳細を以下に示す。

<i>d</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>e</i>	長さ ( <i>n</i> -1) のベクトルを格納する。
<i>w</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>iblock</i>	長さ ( <i>n</i> ) のベクトルを格納する。

<i>isplit</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>order</i>	'B' または 'E' でなければならない。デフォルト値は 'B'。
<i>v1</i>	この引数のデフォルト値は、 $v1 = -HUGE(v1)$ であり、ここで $HUGE(a)$ は引数 <i>a</i> と同じ精度のマシン最大値を意味する。
<i>vu</i>	この引数のデフォルト値は、 $vu = HUGE(v1)$ である。
<i>i1</i>	この引数のデフォルト値は、 $i1 = 1$ である。
<i>iu</i>	この引数のデフォルト値は、 $iu = n$ である。
<i>abstol</i>	この引数のデフォルト値は、 $abstol = 0.0\_WP$ である。
<i>range</i>	引数 <i>v1</i> 、 <i>vu</i> 、 <i>i1</i> 、および <i>iu</i> の存在に基づいて以下のように復元される。 $range = 'V'$ ( <i>v1</i> と <i>vu</i> のいずれかまたは両方が存在する場合)、 $range = 'I'$ ( <i>i1</i> と <i>iu</i> のいずれかまたは両方が存在する場合)、 $range = 'A'$ ( <i>v1</i> 、 <i>vu</i> 、 <i>i1</i> 、および <i>iu</i> がすべて存在しない場合)。 <i>v1</i> と <i>vu</i> のいずれかまたは両方が存在し、同時に <i>i1</i> と <i>iu</i> のいずれかまたは両方が存在する場合、エラー条件がセットされる。

### アプリケーション・ノート

*T* の固有値が比較的高い精度で計算される場合は、その絶対値の変動が大きくても、標準の *QR* 法を使う場合などに比べて、小さな固有値がすべて正確に計算される。ただし、(このルーチンが呼び出される前に実行される) 三重対角形式への縮退によって、固有値の絶対値が大きく変動するような場合には、元の行列の固有値が小さいときに相対精度が低下する場合もある。

## ?stein

実対称三重対角行列の指定された固有値に対応する固有ベクトルを計算する。

### 構文

#### Fortran 77:

```
call sstein(n, d, e, m, w, iblock, isplit, z, ldz, work, iwork, ifailv, info)
call dstein(n, d, e, m, w, iblock, isplit, z, ldz, work, iwork, ifailv, info)
call cstein(n, d, e, m, w, iblock, isplit, z, ldz, work, iwork, ifailv, info)
call zstein(n, d, e, m, w, iblock, isplit, z, ldz, work, iwork, ifailv, info)
```

#### Fortran 95:

```
call stein(d, e, w, iblock, isplit, z [,ifailv] [,info])
```

### 説明

このルーチンは、実対称三重対角行列  $T$  の指定された固有値に対応する固有ベクトルを、反転（逆行列の計算）を繰り返して求める。このルーチンは特に、`order='B'` と指定して `?stebz` を呼び出し、指定の固有値を計算した後で使うように設計されている。ただし、このルーチンは他のルーチンを使って固有値を計算した後に使用できる。`?stebz` の後でこのルーチンを使う場合は、各  $T_i$  ブロックごとに反転を個々に繰り返して、そのブロック構造を利用でき、行列  $T$  の全体を使うよりも、効率が良くなる。

対称 / エルミート行列  $A$  の全体を三重対角形式に縮退させて  $T$  を求めた場合は、`?ormtr` または `?opmtr`（実数型の場合）、`?unmtr` または `?upmtr`（複素数型の場合）を呼び出し、 $T$  の固有ベクトルを  $A$  の固有ベクトルに変換できる。

### 入力パラメータ

$n$                     INTEGER。行列  $T$  の次数 ( $n \geq 0$ )。

$m$                     INTEGER。返すべき固有ベクトルの個数。

$d, e, w$             REAL (単精度の場合)  
                       DOUBLE PRECISION (倍精度の場合)。  
                       配列：  
                        $d(*)$  には、 $T$  の対角成分を格納する。  
                        $d$  の次元は、 $\max(1, n)$  以上でなければならない。

$e(*)$  には、 $T$  の非対角成分が格納される。

$e$  の次元は、 $\max(1, n-1)$  以上でなければならない。

$w(*)$  は、 $w(1) \sim w(m)$  に、[?stebz](#) から返された  $T$  の固有値を格納する。 $T_1$  の固有値、 $T_2$  の固有値の順番 (降順ではない) で指定する。次の制約がある。

$iblock(i) = iblock(i+1)$  の場合、

$w(i) \leq w(i+1)$ 。

$w$  の次元は、 $\max(1, n)$  以上でなければならない。

*iblock, isplit* INTEGER。

配列、次元は  $\max(1, n)$  以上。

配列 *iblock* と *isplit* は、*order*='B' と指定して [?stebz](#) を呼び出し、その出力として返された配列である。

*order*='B' と指定して [?stebz](#) を呼び出さなかった場合、*iblock* の全成分に 1 を、*isplit*(1) に  $n$  を設定する。

*ldz* INTEGER。出力配列  $z$  の第 1 次元。  $ldz \geq \max(1, n)$ 。

*work* REAL (単精度の場合)

DOUBLE PRECISION (倍精度の場合)。ワークスペース配列、次元は  $\max(1, 5n)$  以上。

*iwork* INTEGER。

ワークスペース配列、次元は  $\max(1, n)$  以上。

## 出力パラメータ

*z* REAL (*sstein* の場合)

DOUBLE PRECISION (*dstein* の場合)

COMPLEX (*cstein* の場合)

DOUBLE COMPLEX (*zstein* の場合)。

配列、次元は (*ldz*,\*)。

*info*=0 の場合、 $z$  には、 $m$  個の正規直交固有ベクトルが、列ごとに格納される ( $i$  番目の列は、 $i$  番目に指定した固有値に対応する)。

*ifailv* INTEGER。配列、次元は  $\max(1, m)$  以上。

*info*= $i > 0$  の場合、*ifailv* の先頭から  $i$  個の成分に、収束しなかったすべての固有ベクトルのインデックスが格納される。

*info* INTEGER。

*info*=0 の場合、実行は正常に終了したことを示す。

*info*= $i$  の場合、 $i$  個の固有ベクトル (パラメータ *ifailv* によって指定) が、5 回の繰り返しによっても収束しなかったことを示す。現時



点の繰り返し計算の結果は、配列  $z$  の対応する列に格納される。  
 $info = -i$  の場合、 $i$  番目のパラメータの値が不正であったことを示す。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `stein` のインターフェイスの詳細を以下に示す。

$d$	長さ ( $n$ ) のベクトルを格納する。
$e$	長さ ( $n$ ) のベクトルを $i$ 納する。
$w$	長さ ( $n$ ) のベクトルを格納する。
$iblock$	長さ ( $n$ ) のベクトルを格納する。
$isplit$	長さ ( $n$ ) のベクトルを格納する。
$z$	サイズ ( $n, m$ ) の行列 $Z$ を格納する。
$ifailv$	長さ ( $m$ ) のベクトルを格納する。

### アプリケーション・ノート

計算で求めた各固有ベクトル  $z_i$  は、行列  $T + E_i$  の正確な固有ベクトルである ( $\|E_i\|_2 = O(\epsilon) \|T\|_2$ )。ただし、このルーチンで求める一連の固有ベクトルは、`?steqr` によって求める固有ベクトルと比較すると、直交性の面で精度が低くなる場合もある。

---

## ?disna

対称/エルミート行列の固有ベクトル、または一般行列の左または右の特異ベクトルに対して、条件数の逆数を計算する。

---

### 構文

#### Fortran 77:

```
call sdisna(job, m, n, d, sep, info)
call ddisna(job, m, n, d, sep, info)
```

## Fortran 95:

```
call disna(d, sep [,job] [,minmn] [,info])
```

## 説明

このルーチンは、実対称 / 複素エルミート行列の固有ベクトル、または  $m \times n$  の一般行列の左または右の特異ベクトルに対して、条件数の逆数を計算する。

条件数の逆数とは、最も近い固有値 (または特異値) 間の「ギャップ」である。

計算で求めた  $i$  番目のベクトルの誤差範囲 (ラジアン単位の角度で測定) は、次のように表される。

$$\text{slamch}('E') * (\text{anorm} / \text{sep}(i))$$

ここで、 $\text{anorm} = \|A\|_2 = \max(|d(j)|)$ 。誤差範囲を限定するには、 $\text{sep}(i)$  が  $\text{slamch}('E') * \text{anorm}$  より小さくてはならない。

?disna は、汎用対称固有値問題における固有ベクトルの誤差範囲の計算にも使用できる。

## 入力パラメータ

<i>job</i>	CHARACTER*1。'E'、'L'、または 'R' でなければならない。 どの問題に対して条件数の逆数を計算するか指定する。 <i>job</i> ='E': 対称 / エルミート行列の固有ベクトル。 <i>job</i> ='L': 一般行列の左特異ベクトル。 <i>job</i> ='R': 一般行列の右特異ベクトル。
<i>m</i>	INTEGER。行列の行数 ( $m \geq 0$ )。
<i>n</i>	INTEGER。 <i>job</i> ='L' または 'R' の場合、行列の列数 ( $n \geq 0$ )。 <i>job</i> ='E' の場合は無視される。
<i>d</i>	REAL (sdisna の場合) DOUBLE PRECISION (ddisna の場合)。 配列、次元は $\max(1, m)$ 以上 ( <i>job</i> ='E' の場合) または $\max(1, \min(m, n))$ 以上 ( <i>job</i> ='L' または 'R' の場合)。 この配列には、行列の固有値 ( <i>job</i> ='E' の場合) または特異値 ( <i>job</i> ='L' または 'R' の場合) が、昇順または降順に格納されていないなければならない。特異値は、非負でなければならない。

## 出力パラメータ

<i>sep</i>	REAL (sdisna の場合 ) DOUBLE PRECISION (ddisna の場合 )。 配列、次元は $\max(1,m)$ 以上 ( $job='E'$ の場合 ) または $\max(1, \min(m,n))$ 以上 ( $job='L'$ または $'R'$ の場合 )。 ベクトルの条件数の逆数。
<i>info</i>	INTEGER。 $info=0$ の場合、実行は正常に終了したことを示す。 $info=-i$ の場合、 $i$ 番目のパラメータの値が不正であったことを示す。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `disna` のインターフェイスの詳細を以下に示す。

<i>d</i>	長さ $\min(m,n)$ のベクトルを格納する。
<i>sep</i>	長さ $\min(m,n)$ のベクトルを格納する。
<i>job</i>	'E'、'L'、または 'R' でなければならない。デフォルト値は 'E'。
<i>minmn</i>	$m$ または $n$ で値の小さい方を示す。'M' または 'N' でなければならない。デフォルト値は 'M'。 $job='E'$ の場合、この引数は余分である。 $job='L'$ または 'R' の場合、この引数はルーチンで使用される。

## 汎用対称固有値問題

汎用対称固有値問題は、「次のいずれかの方程式を満たすような固有値  $\lambda$  とそれに対応する固有ベクトル  $z$  を求めること」である。

$$Az = \lambda Bz, ABz = \lambda z, \text{ または } BAz = \lambda z$$

$A$  は  $n \times n$  の対称行列またはエルミート行列、 $B$  は  $n \times n$  の対称正定値行列またはエルミート正定値行列である。

このような固有値問題では、実数型の固有値に対応する実数型の固有ベクトルが  $n$  個存在している (エルミート行列  $A$  と  $B$  が複素型の場合でも実数型になる)。

ここで説明する一連のルーチンを使えば、上記の一般化された問題を標準の対称固有値問題  $Cy = \lambda y$  に縮退させられる。縮退させた後の標準の対称固有値問題は、この章ですでに説明した一連の LAPACK ルーチン ([4-107 ページ](#)を参照) を呼び出して解くことができる。

種々のルーチンで、行列を従来の形式または圧縮形式で格納できる。縮退の前に、まず正定値行列  $B$  を [?potrf](#) または [?pptrf](#) を使用して因子分解する必要がある。

帯行列  $A$  と  $B$  の縮退ルーチンは、分割コレスキー因子分解を使用するが、このための専用ルーチン [?pbstf](#) が提供されている。このルーチンを使用することにより、行列  $C$  の生成に必要な作業量が半分で済む。

表 4-4 に、汎用対称固有値問題を解くために使用できる LAPACK ルーチン (Fortran-77 インターフェイス) を示す。Fortran-95 インターフェイスのルーチン名はそれぞれ、先頭の記号がない ([ルーチン命名規則](#)を参照)。

**表 4-4 汎用固有値問題から標準固有値問題への縮退用の計算ルーチン**

行列の タイプ	標準問題への縮退 (フル格納形式)	標準問題への縮退 (圧縮格納形式)	標準問題への縮退 (帯行列)	帯行列の 因子分解
実対称 行列	<a href="#">?syqst</a>	<a href="#">?spqst</a>	<a href="#">?sbqst</a>	<a href="#">?pbstf</a>
複素 エルミート 行列	<a href="#">?hegst</a> /	<a href="#">?hpqst</a>	<a href="#">?hbqst</a>	<a href="#">?pbstf</a>

## ?sygst

実対称の汎用固有値問題を標準形式に縮退させる。

### 構文

#### Fortran 77:

```
call ssygst(itype, uplo, n, a, lda, b, ldb, info)
call dsygst(itype, uplo, n, a, lda, b, ldb, info)
```

#### Fortran 95:

```
call sygst(a, b [,itype] [,uplo] [,info])
```

### 説明

このルーチンは、汎用対称固有値問題

$$Az = \lambda Bz, ABz = \lambda z, \text{ または } BAz = \lambda z$$

を、標準形式  $Cy = \lambda y$  に縮退させる。 $A$  は実対称行列で、 $B$  は実対称正定値行列である。このルーチン呼び出す前に、[?potrf](#) を呼び出し、コレスキー因子分解  $B = U^T U$  または  $B = LL^T$  を計算する。

### 入力パラメータ

*itype*                    INTEGER。1、2、または3のいずれかでなければならない。  
*itype* = 1 の場合、汎用固有値問題は  $Az = \lambda Bz$ 。  
     *uplo* = 'U' の場合、 $C = U^{-T}AU^{-1}$ 、 $z = U^{-1}y$ 。  
     *uplo* = 'L' の場合、 $C = L^{-1}AL^{-T}$ 、 $z = L^{-T}y$ 。  
*itype* = 2 の場合、汎用固有値問題は  $ABz = \lambda z$ 。  
     *uplo* = 'U' の場合、 $C = UAU^T$ 、 $z = U^{-1}y$ 。  
     *uplo* = 'L' の場合、 $C = L^TAL$ 、 $z = L^{-T}y$ 。  
*itype* = 3 の場合、汎用固有値問題は  $BAz = \lambda z$ 。  
     *uplo* = 'U' の場合、 $C = UAU^T$ 、 $z = U^T y$ 。  
     *uplo* = 'L' の場合、 $C = L^TAL$ 、 $z = Ly$ 。

<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> ='U' の場合、配列 <i>a</i> に <i>A</i> の上三角行列を格納する。 <i>B</i> は、 $B = U^T U$ に因子分解した形式で指定しなければならない。 <i>uplo</i> ='L' の場合、配列 <i>a</i> に <i>A</i> の下三角行列を格納する。 <i>B</i> は、 $B = LL^T$ に因子分解した形式で指定しなければならない。
<i>n</i>	INTEGER。行列 <i>A</i> と <i>B</i> の次数 ( $n \geq 0$ )。
<i>a</i> , <i>b</i>	REAL(ssygst の場合) DOUBLE PRECISION(dsygst の場合)。 配列: <i>a</i> ( <i>lda</i> ,*) には、 <i>A</i> の上三角行列または下三角行列を格納する。 <i>a</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。  <i>b</i> ( <i>ldb</i> ,*) には、次のコレスキー因子分解された行列 <i>B</i> を格納する。 $B = U^T U$ または $B = LL^T$ (?potrf から返された行列)。 <i>b</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $\max(1, n)$ 以上。
<i>ldb</i>	INTEGER。 <i>b</i> の第 1 次元。 $\max(1, n)$ 以上。

## 出力パラメータ

<i>a</i>	引数 <i>itype</i> と <i>uplo</i> の値に従って、 <i>A</i> の上三角行列または下三角行列が、 <i>C</i> の上三角行列または下三角行列によって上書きされる。
<i>info</i>	INTEGER。 <i>info</i> =0 の場合、実行は正常に終了したことを示す。 <i>info</i> =- <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正であったことを示す。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *sygst* のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ ( <i>n</i> , <i>n</i> ) の行列 <i>A</i> を格納する。
<i>b</i>	サイズ ( <i>n</i> , <i>n</i> ) の行列 <i>B</i> を格納する。
<i>itype</i>	1、2、または 3 でなければならない。デフォルト値は 1。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。

## アプリケーション・ノート

縮退させた行列  $C$  を得るのは、安定した処理である。ただし、 $B^{-1}$  ( $itype = 1$  の場合) または  $B$  ( $itype = 2$  または  $3$  の場合) が、暗黙的に乗算される。このルーチンを元の問題の固有値および固有ベクトルを計算する際の 1 ステップとして使う場合、反転処理に関して  $B$  の条件が悪いと、精度が大きく低下するときがある。

浮動小数演算のおおよその総数は、 $n^3$  である。

## ?hegst

複素エルミートの汎用固有値問題を標準形式に縮退させる。

### 構文

#### Fortran 77:

```
call chegst(itype, uplo, n, a, lda, b, ldb, info)
call zhegst(itype, uplo, n, a, lda, b, ldb, info)
```

#### Fortran 95:

```
call hegst(a, b [,itype] [,uplo] [,info])
```

### 説明

このルーチンは、次の複素エルミートの汎用固有値問題

$$Az = \lambda Bz, ABz = \lambda z, \text{ または } BAz = \lambda z$$

を、標準形式  $Cy = \lambda y$  に縮退させる。行列  $A$  は複素エルミート行列で、 $B$  は複素エルミート正定値行列である。このルーチン呼び出す前に、[?potrf](#) を呼び出し、コレスキー因子分解  $B = U^H U$  または  $B = LL^H$  を計算する。

### 入力パラメータ

**itype** INTEGER。1、2、または 3 のいずれかでなければならない。  
 $itype = 1$  の場合、汎用固有値問題は  $Az = \lambda Bz$ 。  
      $uplo = 'U'$  の場合、 $C = U^{-H} A U^{-1}$ 、 $z = U^{-1} y$ 。  
      $uplo = 'L'$  の場合、 $C = L^{-1} A L^{-H}$ 、 $z = L^{-H} y$ 。  
 $itype = 2$  の場合、汎用固有値問題は  $ABz = \lambda z$ 。

$uplo = 'U'$  の場合、 $C = UAU^H$ 、 $z = U^{-1}y$ 。  
 $uplo = 'L'$  の場合、 $C = L^HAL$ 、 $z = L^{-H}y$ 。  
 $itype = 3$  の場合、汎用固有値問題は  $BAz = \lambda z$ 。  
 $uplo = 'U'$  の場合、 $C = UAU^H$ 、 $z = U^Hy$ 。  
 $uplo = 'L'$  の場合、 $C = L^HAL$ 、 $z = Ly$ 。

**uplo** CHARACTER\*1。'U' または 'L' でなければならない。  
 $uplo = 'U'$  の場合、配列 *a* に *A* の上三角行列を格納する。  
*B* は、 $B = U^HU$  に因子分解した形式で指定しなければならない。  
 $uplo = 'L'$  の場合、配列 *a* に *A* の下三角行列を格納する。  
*B* は、 $B = LL^H$  に因子分解した形式で指定しなければならない。

**n** INTEGER。行列 *A* と *B* の次数 ( $n \geq 0$ )。

**a, b** COMPLEX (chegst の場合)  
 DOUBLE COMPLEX (zhegst の場合)。  
 配列:  
*a*(*lda*,\*) には、*A* の上三角行列または下三角行列を格納する。  
*a* の第 2 次元は、 $\max(1, n)$  以上でなければならない。  
*b*(*ldb*,\*) には、次のコレスキー因子分解された行列 *B* を格納する。  
 $B = U^HU$  または  $B = LL^H$  (?potrf から返された行列)。  
*b* の第 2 次元は、 $\max(1, n)$  以上でなければならない。

**lda** INTEGER。 *a* の第 1 次元。  $\max(1, n)$  以上。

**ldb** INTEGER。 *b* の第 1 次元。  $\max(1, n)$  以上。

## 出力パラメータ

**a** 引数 *itype* と *uplo* の値に従って、*A* の上三角行列または下三角行列が、*C* の上三角行列または下三角行列によって上書きされる。

**info** INTEGER。  
*info* = 0 の場合、実行は正常に終了したことを示す。  
*info* = -*i* の場合、*i* 番目のパラメータの値が不正であったことを示す。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *hegst* のインターフェイスの詳細を以下に示す。



<i>a</i>	サイズ $(n,n)$ の行列 <i>A</i> を格納する。
<i>b</i>	サイズ $(n,n)$ の行列 <i>B</i> を格納する。
<i>itype</i>	1、2、または 3 でなければならない。デフォルト値は 1。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。

### アプリケーション・ノート

縮退させた行列 *C* を得るのは、安定した処理である。ただし、 $B^{-1}$  (*itype* = 1 の場合) または *B* (*itype* = 2 または 3 の場合) が、暗黙的に乗算される。このルーチンを元の問題の固有値および固有ベクトルを計算する際の 1 ステップとして使う場合、反転処理に関して *B* の条件が悪いと、精度が大きく低下するときがある。

浮動小数演算のおおよその総数は、 $n^3$  である。

---

## ?spgst

圧縮格納形式の実対称の汎用固有値問題を  
標準形式に縮退させる。

---

### 構文

#### Fortran 77:

```
call sspgst(itype, uplo, n, ap, bp, info)
call dspgst(itype, uplo, n, ap, bp, info)
```

#### Fortran 95:

```
call spgst(a, b [,itype] [,uplo] [,info])
```

### 説明

このルーチンは、汎用対称固有値問題

$$Az = \lambda Bz, ABz = \lambda z, \text{ または } BAz = \lambda z$$

を、標準形式  $Cy = \lambda y$  に縮退させる。*A* は実対称行列で、*B* は実対称正定値行列である。このルーチンを呼び出す前に、[?pptrf](#) を呼び出し、コレスキー因子分解  $B = U^T U$  または  $B = LL^T$  を計算する。

## 入力パラメータ

<i>itype</i>	<p>INTEGER。1、2、または3のいずれかでなければならない。</p> <p><i>itype</i> = 1 の場合、汎用固有値問題は <math>Az = \lambda Bz</math>。</p> <p>    <i>uplo</i> = 'U' の場合、<math>C = U^{-T}AU^{-1}</math>、<math>z = U^{-1}y</math>。</p> <p>    <i>uplo</i> = 'L' の場合、<math>C = L^{-1}AL^{-T}</math>、<math>z = L^{-T}y</math>。</p> <p><i>itype</i> = 2 の場合、汎用固有値問題は <math>ABz = \lambda z</math>。</p> <p>    <i>uplo</i> = 'U' の場合、<math>C = UAU^T</math>、<math>z = U^{-1}y</math>。</p> <p>    <i>uplo</i> = 'L' の場合、<math>C = L^TAL</math>、<math>z = L^{-T}y</math>。</p> <p><i>itype</i> = 3 の場合、汎用固有値問題は <math>BAz = \lambda z</math>。</p> <p>    <i>uplo</i> = 'U' の場合、<math>C = UAU^T</math>、<math>z = U^T y</math>。</p> <p>    <i>uplo</i> = 'L' の場合、<math>C = L^TAL</math>、<math>z = Ly</math>。</p>
<i>uplo</i>	<p>CHARACTER*1。'U' または 'L' でなければならない。</p> <p><i>uplo</i> = 'U' の場合、<i>ap</i> に <i>A</i> の上三角行列 (圧縮形式) を格納する。 <i>B</i> は、<math>B = U^T U</math> に因子分解した形式で指定しなければならない。</p> <p><i>uplo</i> = 'L' の場合、<i>ap</i> に <i>A</i> の下三角行列 (圧縮形式) を格納する。 <i>B</i> は、<math>B = LL^T</math> に因子分解した形式で指定しなければならない。</p>
<i>n</i>	INTEGER。行列 <i>A</i> と <i>B</i> の次数 ( $n \geq 0$ )。
<i>ap</i> , <i>bp</i>	<p>REAL (sspgst の場合 )</p> <p>DOUBLE PRECISION (dspgst の場合 )。</p> <p>配列 :</p> <p><i>ap</i>(*) には、<i>A</i> の上三角行列または下三角行列 (圧縮形式) を格納する。 <i>ap</i> の次元は、<math>\max(1, n*(n+1)/2)</math> 以上でなければならない。</p> <p><i>bp</i>(*) には、<i>B</i> (同一の <i>uplo</i> 値を指定して ?pptrf から返されたもの) のコレスキー因数 (圧縮形式) を格納する。 <i>bp</i> の次元は、<math>\max(1, n*(n+1)/2)</math> 以上でなければならない。</p>

## 出力パラメータ

<i>ap</i>	引数 <i>itype</i> と <i>uplo</i> の値に従って、 <i>A</i> の上三角行列または下三角行列が、 <i>C</i> の上三角行列または下三角行列によって上書きされる。
<i>info</i>	<p>INTEGER。</p> <p><i>info</i> = 0 の場合、実行は正常に終了したことを示す。</p> <p><i>info</i> = -<i>i</i> の場合、<i>i</i> 番目のパラメータの値が不正であったことを示す。</p>

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `spgst` のインターフェイスの詳細を以下に示す。

<code>a</code>	Fortran 77 インターフェイスでの引数 <code>ap</code> を意味する。 サイズ $(n*(n+1)/2)$ の配列 $A$ を格納する。
<code>b</code>	Fortran 77 インターフェイスでの引数 <code>bp</code> を意味する。 サイズ $(n*(n+1)/2)$ の配列 $B$ を格納する。
<code>itype</code>	1、2、または 3 でなければならない。デフォルト値は 1。
<code>uplo</code>	'U' または 'L' でなければならない。デフォルト値は 'U'。

## アプリケーション・ノート

縮退させた行列  $C$  を得るのは、安定した処理である。ただし、 $B^{-1}$  ( $itype = 1$  の場合) または  $B$  ( $itype = 2$  または 3 の場合) が、暗黙的に乗算される。このルーチンを元の問題の固有値および固有ベクトルを計算する際の 1 ステップとして使う場合、反転処理に関して  $B$  の条件が悪いと、精度が大きく低下するときがある。

浮動小数演算のおおよその総数は、 $n^3$  である。

---

## ?hpgst

圧縮格納形式の複素エルミート行列の  
汎用固有値問題を標準形式に縮退させる。

---

### 構文

#### Fortran 77:

```
call chpgst(itype, uplo, n, ap, bp, info)
call zhpst(itype, uplo, n, ap, bp, info)
```

#### Fortran 95:

```
call hpgst(a, b [,itype] [,uplo] [,info])
```

## 説明

このルーチンは、汎用対称固有値問題

$$Az = \lambda Bz, ABz = \lambda z, \text{ または } BAz = \lambda z$$

を、標準形式  $Cy = \lambda y$  に縮退させる。  $A$  は実対称行列で、  $B$  は実対称正定値行列である。このルーチンを呼び出す前に、 [?pptrf](#) を呼び出し、コレスキー因子分解  $B = U^H U$  または  $B = LL^H$  を計算する。

## 入力パラメータ

**itype** INTEGER。 1、 2、 または 3 のいずれかでなければならない。  
**itype** = 1 の場合、汎用固有値問題は  $Az = \lambda Bz$ 。  
     **uplo** = 'U' の場合、  $C = U^{-H} A U^{-1}$ 、  $z = U^{-1} y$ 。  
     **uplo** = 'L' の場合、  $C = L^{-1} A L^{-H}$ 、  $z = L^{-H} y$ 。  
**itype** = 2 の場合、汎用固有値問題は  $ABz = \lambda z$ 。  
     **uplo** = 'U' の場合、  $C = U A U^H$ 、  $z = U^{-1} y$ 。  
     **uplo** = 'L' の場合、  $C = L^H A L$ 、  $z = L^{-H} y$ 。  
**itype** = 3 の場合、汎用固有値問題は  $BAz = \lambda z$ 。  
     **uplo** = 'U' の場合、  $C = U A U^H$ 、  $z = U^H y$ 。  
     **uplo** = 'L' の場合、  $C = L^H A L$ 、  $z = L y$ 。  
**uplo** CHARACTER\*1。 'U' または 'L' でなければならない。  
     **uplo** = 'U' の場合、 **ap** に  $A$  の上三角行列 (圧縮形式) を格納する。  
     **B** は、  $B = U^H U$  に因子分解した形式で指定しなければならない。  
     **uplo** = 'L' の場合、 **ap** に  $A$  の下三角行列 (圧縮形式) を格納する。  
     **B** は、  $B = LL^H$  に因子分解した形式で指定しなければならない。  
**n** INTEGER。 行列  $A$  と  $B$  の次数 ( $n \geq 0$ )。  
**ap, bp** COMPLEX (chpgst の場合 )  
     DOUBLE COMPLEX (zhpgst の場合 )。  
     配列 :  
     **ap**(\*) には、  $A$  の上三角行列または下三角行列 (圧縮形式) を格納する。  
     **a** の次元は、  $\max(1, n*(n+1)/2)$  以上でなければならない。  
     **bp**(\*) には、  $B$  ( 同一の **uplo** 値を指定して [?pptrf](#) から返されたもの )  
     のコレスキー因数 (圧縮形式) を格納する。  
     **b** の次元は、  $\max(1, n*(n+1)/2)$  以上でなければならない。

## 出力パラメータ

**ap** 引数 **itype** と **uplo** の値に従って、  $A$  の上三角行列または下三角行列が、  $C$  の上三角行列または下三角行列によって上書きされる。

*info*                    INTEGER。  
                         *info* = 0 の場合、実行は正常に終了したことを示す。  
                         *info* = -*i* の場合、*i* 番目のパラメータの値が不正であったことを示す。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `hpgst` のインターフェイスの詳細を以下に示す。

*a*                    Fortran 77 インターフェイスでの引数 *ap* を意味する。  
                         サイズ  $(n*(n+1)/2)$  の配列 *A* を格納する。

*b*                    Fortran 77 インターフェイスでの引数 *bp* を意味する。  
                         サイズ  $(n*(n+1)/2)$  の配列 *B* を格納する。

*itype*                1、2、または 3 でなければならない。デフォルト値は 1。

*uplo*                'U' または 'L' でなければならない。デフォルト値は 'U'。

### アプリケーション・ノート

縮退させた行列 *C* を得るのは、安定した処理である。ただし、 $B^{-1}$  (*itype* = 1 の場合) または *B* (*itype* = 2 または 3 の場合) が、暗黙的に乗算される。このルーチンを元の問題の固有値および固有ベクトルを計算する際の 1 ステップとして使う場合、反転処理に関して *B* の条件が悪いと、精度が大きく低下するときがある。

浮動小数演算のおおよその総数は、 $n^3$  である。

## ?sbgst

?pbstf による因子分解を使用して、  
帯形式の実対称行列の汎用固有値問題を  
標準形式に縮退させる。

### 構文

#### Fortran 77:

```
call ssbgst(vect, uplo, n, ka, kb, ab, ldab, bb, ldbb, x, ldx, work, info)
call dsbgst(vect, uplo, n, ka, kb, ab, ldab, bb, ldbb, x, ldx, work, info)
```

#### Fortran 95:

```
call sbgst(a, b [,x] [,uplo] [,info])
```

### 説明

実対称汎用固有値問題 ( $Az = \lambda Bz$ ) を標準形式 ( $Cy = \lambda y$ ) に縮退させる ( $A$ 、 $B$ 、および  $C$  は帯行列) には、このルーチンに先立って、正定値行列  $B$  の分割コレスキー因子分解 ( $B: B = S^T S$ ) を行う [spbstf](#)/[dpbstf](#) を呼び出す必要がある。分割コレスキー因子分解は、通常のコレスキー因子分解と比べて、作業量が半分で済む。

このルーチンは、 $A$  を  $C = X^T A X$  で上書きする。ここで、 $X = S^{-1} Q$  と  $Q$  は  $A$  の帯幅を保持するために (暗黙的に) 選択された直交行列である。

また、このルーチンには、 $X$  を蓄積するオプションがある。その場合、 $z$  が  $C$  の固有ベクトルであれば、 $Xz$  は元の問題の固有ベクトルとなる。

### 入力パラメータ

<b>vect</b>	CHARACTER*1。'N' または 'V' でなければならない。 <i>vect</i> = 'N' の場合、行列 $X$ は返されない。 <i>vect</i> = 'V' の場合、行列 $X$ が返される。
<b>uplo</b>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> = 'U' の場合、 <i>ab</i> に $A$ の上三角部分を格納する。 <i>uplo</i> = 'L' の場合、 <i>ab</i> に $A$ の下三角部分を格納する。
<b>n</b>	INTEGER。行列 $A$ と $B$ の次数 ( $n \geq 0$ )。
<b>ka</b>	INTEGER。 $A$ の優対角成分または劣対角成分の数 ( $ka \geq 0$ )。
<b>kb</b>	INTEGER。 $B$ の優対角成分または劣対角成分の数 ( $ka \geq kb \geq 0$ )。

*ab, bb, work* REAL (ssbgst の場合)  
 DOUBLE PRECISION (dsbgst の場合)  
*ab(ldab,\*)* は、(*uplo* の値に従って) 対称行列 *A* の上三角部分または下三角部分を帯形式で格納する配列である。配列 *ab* の第 2 次元は、 $\max(1, n)$  以上でなければならない。  
*bb(lddb,\*)* は、(*uplo*、*n*、および *kb* の値に従って) [spbstf/dpbstf](#) から返された分割コレスキー因子 *B* を帯形式で格納する配列である。配列 *bb* の第 2 次元は、 $\max(1, n)$  以上でなければならない。  
*work(\*)* は、ワークスペース配列、次元は  $\max(1, 2*n)$  以上。

*ldab* INTEGER。配列 *ab* の第 1 次元。*ka*+1 以上でなければならない。

*ldbb* INTEGER。配列 *bb* の第 1 次元。*kb*+1 以上でなければならない。

*ldx* 出力配列 *x* の第 1 次元。次の制約がある。  
*vect*='N' の場合、*ldx* ≥ 1。  
*vect*='V' の場合、*ldx* ≥  $\max(1, n)$ 。

### 出力パラメータ

*ab* 終了時に、*uplo* の値に従って、*C* の上または下三角部分で上書きされる。

*x* REAL (ssbgst の場合)  
 DOUBLE PRECISION (dsbgst の場合)  
 配列：  
*vect*='V' の場合、*x(ldx,\*)* に  $n \times n$  の行列  $X = S^{-1}Q$  が格納される。  
*vect*='N' の場合、*x* は参照されない。  
*x* の第 2 次元は、 $\max(1, n)$  以上 (*vect*='V' の場合) または 1 以上 (*vect*='N' の場合) でなければならない。

*info* INTEGER。  
*info*=0 の場合、実行は正常に終了したことを示す。  
*info*=-*i* の場合、*i* 番目のパラメータの値が不正であったことを示す。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *sbgst* のインターフェイスの詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ab</i> を意味する。 サイズ $(ka+1, n)$ の配列 <i>A</i> を格納する。
<i>b</i>	Fortran 77 インターフェイスでの引数 <i>bb</i> を意味する。 サイズ $(kd+1, n)$ の配列 <i>B</i> を格納する。
<i>x</i>	サイズ $(n, n)$ の行列 <i>X</i> を格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>vect</i>	引数 <i>x</i> の存在に基づいて以下のように復元される。 <i>vect</i> = 'V' ( <i>x</i> が存在する場合)、 <i>vect</i> = 'N' ( <i>x</i> が省略された場合)。

### アプリケーション・ノート

縮退した行列 *C* を生成するには、暗黙的に  $B^{-1}$  が乗算される。このルーチンを元の問題の固有値および固有ベクトルを計算する際の 1 ステップとして使う場合、反転処理に関して *B* の条件が悪いと、精度が大きく低下するときがある。  
浮動小数点演算の総数は、*vect* = 'N' の場合、約  $6n^2 \cdot kb$  になる。*vect* = 'V' の場合、さらに  $(3/2)n^3 \cdot (kb/ka)$  回の演算が必要になる。ただしこれらの値は、*ka* と *kb* がどちらも *n* と比べて十分に小さい場合の推定値である。

---

## ?hbgst

?pbstf による因子分解を使用して、帯形式の複素エルミート行列の汎用固有値問題を標準形式に縮退させる。

---

### 構文

#### Fortran 77:

```
call chbgst(vect, uplo, n, ka, kb, ab, ldab, bb, ldbb, x, ldx, work, rwork, info)
call zhbgst(vect, uplo, n, ka, kb, ab, ldab, bb, ldbb, x, ldx, work, rwork, info)
```

#### Fortran 95:

```
call hbgst(a, b [,x] [,uplo] [,info])
```



## 説明

複素エルミート行列の汎用固有値問題 ( $Az = \lambda Bz$ ) を標準形式 ( $Cy = \lambda y$ ) に縮退させる ( $A$ ,  $B$ ,  $C$  は帯行列) には、このルーチンに先立って、正定値行列  $B$  の分割コレスキー因子分解 ( $B: B = S^H S$ ) を行う [cpbstf/zpbstf](#) を呼び出す必要がある。分割コレスキー因子分解は、通常のコレスキー因子分解と比べて、作業量が半分で済む。

このルーチンは、 $A$  を  $C = X^H A X$  で上書きする。ここで、 $X = S^{-1} Q$  と  $Q$  は  $A$  の帯幅を保持するために (暗黙的に) 選択されたユニタリ行列である。

また、このルーチンには、 $X$  を蓄積するオプションがある。その場合、 $z$  が  $C$  の固有ベクトルであれば、 $Xz$  は元の問題の固有ベクトルとなる。

## 入力パラメータ

<i>vect</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>vect</i> = 'N' の場合、行列 $X$ は返されない。 <i>vect</i> = 'V' の場合、行列 $X$ が返される。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> = 'U' の場合、 <i>ab</i> に $A$ の上三角部分を格納する。 <i>uplo</i> = 'L' の場合、 <i>ab</i> に $A$ の下三角部分を格納する。
<i>n</i>	INTEGER。行列 $A$ と $B$ の次数 ( $n \geq 0$ )。
<i>ka</i>	INTEGER。 $A$ の優対角成分または劣対角成分の数 ( $ka \geq 0$ )。
<i>kb</i>	INTEGER。 $B$ の優対角成分または劣対角成分の数 ( $ka \geq kb \geq 0$ )。
<i>ab, bb, work</i>	COMPLEX (chbgvx の場合) DOUBLE COMPLEX (zhbgvx の場合) 配列: <i>ab</i> ( <i>ldab</i> ,*) は、( <i>uplo</i> の値に従って) エルミート行列 $A$ の上三角部分または下三角部分を帯格納形式で格納する配列である。配列 <i>ab</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>bb</i> ( <i>ldbb</i> ,*) は、( <i>uplo</i> 、 <i>n</i> 、および <i>kb</i> の値に従って) <a href="#">cpbstf/zpbstf</a> から返された分割コレスキー因子 $B$ を帯形式で格納する配列である。 配列 <i>bb</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>work</i> (*) は、ワークスペース配列、次元は $\max(1, n)$ 以上。
<i>ldab</i>	INTEGER。配列 <i>ab</i> の第 1 次元。 $ka+1$ 以上でなければならない。
<i>ldbb</i>	INTEGER。配列 <i>bb</i> の第 1 次元。 $kb+1$ 以上でなければならない。
<i>ldx</i>	出力配列 <i>x</i> の第 1 次元。次の制約がある。 <i>vect</i> = 'N' の場合、 $ldx \geq 1$ 。 <i>vect</i> = 'V' の場合、 $ldx \geq \max(1, n)$ 。

*rwork* REAL (chbgst の場合)  
DOUBLE PRECISION (zhbgst の場合)  
ワークスペース配列、次元は  $\max(1, n)$  以上。

## 出力パラメータ

*ab* 終了時に、*uplo* の値に従って、*C* の上または下三角部分で上書きされる。

*x* COMPLEX (chbgst の場合)  
DOUBLE COMPLEX (zhbgst の場合)  
配列：  
*vect* = 'V' の場合、*x*(*ldx*, \*) に  $n \times n$  の行列  $X = S^{-1}Q$  が格納される。  
*vect* = 'N' の場合、*x* は参照されない。  
*x* の第 2 次元は、 $\max(1, n)$  以上 (*vect* = 'V' の場合) または 1 以上 (*vect* = 'N' の場合) でなければならない。

*info* INTEGER。  
*info* = 0 の場合、実行は正常に終了したことを示す。  
*info* = -*i* の場合、*i* 番目のパラメータの値が不正であったことを示す。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン hbgst のインターフェイスの詳細を以下に示す。

*a* Fortran 77 インターフェイスでの引数 *ab* を意味する。サイズ ( $ka+1, n$ ) の配列 *A* を格納する。

*b* Fortran 77 インターフェイスでの引数 *bb* を意味する。サイズ ( $kd+1, n$ ) の配列 *B* を格納する。

*x* サイズ ( $n, n$ ) の行列 *X* を格納する。

*uplo* 'U' または 'L' でなければならない。デフォルト値は 'U'。

*vect* 引数 *x* の存在に基づいて以下のように復元される。  
*vect* = 'V' (*x* が存在する場合)、  
*vect* = 'N' (*x* が省略された場合)。

## アプリケーション・ノート

縮退した行列  $C$  を生成するには、暗黙的に  $B^{-1}$  が乗算される。このルーチンを元の問題の固有値および固有ベクトルを計算する際の 1 ステップとして使う場合、反転処理に関して  $B$  の条件が悪いと、精度が大きく低下するときがある。

浮動小数点演算の総数は、 $\text{vect} = 'N'$  の場合、約  $20n^2 * kb$  になる。 $\text{vect} = 'V'$  の場合、さらに  $5n^3 * (kb/ka)$  回の演算が必要になる。ただしこれらの値は、 $ka$  と  $kb$  がどちらも  $n$  と比べて十分に小さい場合の推定値である。

## ?pbstf

?sbgst/?hbgst で使用するために、帯形式の実対称 / 複素エルミート正定値行列の分割コレスキー因子分解を行う。

### 構文

#### Fortran 77:

```
call spbstf(uplo, n, kb, bb, ldbb, info)
call dpbstf(uplo, n, kb, bb, ldbb, info)
call cpbstf(uplo, n, kb, bb, ldbb, info)
call zpbstf(uplo, n, kb, bb, ldbb, info)
```

#### Fortran 95:

```
call pbstf(b [,uplo] [,info])
```

### 説明

このルーチンは、帯形式の実対称 / 複素エルミート正定値行列  $B$  の分割コレスキー因子分解を行う。このルーチンは、[?sbgst](#)/[?hbgst](#) とともに使用する。

分割コレスキー因子分解は、 $B = S^T S$  (複素数型の場合、 $B = S^H S$ ) と表される。ここで、 $S$  は、帯幅が  $B$  と同じ帯行列で、最初の  $(n+kb)/2$  行は上三角行列、残りの行は下三角行列である。

### 入力パラメータ

`uplo` CHARACTER\*1。'U' または 'L' でなければならない。

	<code>uplo='U'</code> の場合、 <code>bb</code> に $B$ の上三角部分を格納する。 <code>uplo='L'</code> の場合、 <code>bb</code> に $B$ の下三角部分を格納する。
<code>n</code>	INTEGER。行列 $B$ の次数 ( $n \geq 0$ )。
<code>kb</code>	INTEGER。 $B$ の優対角成分または劣対角成分の数 ( $kb \geq 0$ )。
<code>bb</code>	REAL ( <code>spbstf</code> の場合 ) DOUBLE PRECISION ( <code>dpbstf</code> の場合 ) COMPLEX ( <code>cpbstf</code> の場合 ) DOUBLE COMPLEX ( <code>zpbstf</code> の場合 )。 <code>bb(ldbb,*)</code> は、( <code>uplo</code> の値に従って ) 行列 $B$ の上三角部分または下三角部分を帯形式で格納する配列である。 配列 <code>bb</code> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。
<code>ldbb</code>	INTEGER。 <code>bb</code> の第 1 次元。 $kb+1$ 以上でなければならない。

### 出力パラメータ

<code>bb</code>	終了時に、分割コレスキー因子 $S$ の各成分で上書きされる。
<code>info</code>	INTEGER。 <code>info = 0</code> の場合、実行は正常に終了したことを示す。 <code>info = i</code> の場合、更新しようとした成分 $b_{ii}$ が負値の平方根になるため、因子分解を完了できなかったことを示す。すなわち、行列 $B$ が正定値でない。 <code>info = -i</code> の場合、 $i$ 番目のパラメータの値が不正であったことを示す。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `pbstf` のインターフェイスの詳細を以下に示す。

<code>b</code>	Fortran 77 インターフェイスでの引数 <code>bb</code> を意味する。サイズ $(kd+1, n)$ の配列 $B$ を格納する。
<code>uplo</code>	'U' または 'L' でなければならない。デフォルト値は 'U'。

## アプリケーション・ノート

計算で求めた因子  $S$  は、正確には摂動行列  $B + E$  の因子である。  
 $c(n)$  は  $n$  の適度な線形関数、 $\varepsilon$  はマシン精度。

$$|E| \leq c(kb+1)\varepsilon|S^H||S|, \quad |e_{ij}| \leq c(kb+1)\varepsilon\sqrt{b_{ii}b_{jj}}$$

浮動小数点演算の総数は、実数型の場合は約  $n(kb+1)^2$  である。複素数型の演算回数は、この 4 倍になる。これらの値は、 $kb$  が  $n$  と比べて十分に小さい場合の推定値である。

このルーチンを呼び出した後、[?sbgst/?hbgst](#) を呼び出して汎用固有値問題 ( $Az = \lambda Bz$ 、ただし、 $A$  と  $B$  は帯形式、 $B$  は正定値) を解くことができる。

## 非対称固有値問題

このセクションでは、非対称固有値問題の解決、一般行列の Schur 因子分解の計算、およびそれらに関連する各種の計算タスクを実行するための LAPACK ルーチンについて説明する。

**非対称固有値問題**は、「非対称 (または非エルミート) の行列  $A$  が与えられたときに、次のいずれかの方程式を満たす固有値  $\lambda$  と、それに対応する固有ベクトル  $z$  を求めること」である。

$$Az = \lambda z \text{ (右固有ベクトル } z\text{)}$$

または

$$z^H A = \lambda z^H \text{ (左固有ベクトル } z\text{)}$$

非対称固有値問題には、次の特性がある。

- 固有ベクトルの個数が、行列の次数より少ない場合がある (ただし、 $A$  の相異なる固有値の個数以上である)。
- 実行列  $A$  に対して、固有値が複素数の場合もある。
- 実非対称行列が、固有ベクトル  $z$  に対応する固有値として複素数  $a + bi$  を持っている場合は、 $a - bi$  も固有値である。  
 固有値  $a - bi$  は、 $z$  の成分に対して複素共役になっている成分の固有ベクトルに対応する。

LAPACK を使って非対称固有値問題を解くには、通常、行列を上 Hessenberg 形式に縮退させた後、得られた Hessenberg 行列を使って固有値問題を解く必要がある。[表 4-5](#) に示されている LAPACK ルーチン (Fortran-77 インターフェイス) を利用すれば、直交 (またはユニタリ) の相似変換  $A = QHQ^H$  を使って、該当行列を上 Hessenberg 形式に縮退さ

せることができる。また、Hessenberg 行列を使って固有値問題を解き、行列の Schur 因子分解を求め、対応する条件数を計算できる。

Fortran-95 インターフェイスのルーチン名はそれぞれ、先頭の記号がない ([ルーチン命名規則](#) を参照)。

[図 4-4](#) のデシジョン・ツリーを参照すれば、実非対称行列を使って固有値問題を解くためのルーチン (1 つまたは複数個) を簡単に選べる。

複素非エルミート行列を使って固有値問題を解く場合は、[図 4-5](#) のデシジョン・ツリーを参照する。

**表 4-5 非対称固有値問題を解くための計算ルーチン**

機能	実行列用のルーチン	複素実行列用のルーチン
Hessenberg 形式 $A = QHQ^H$ に縮退させる	<a href="#">?gehrd</a>	<a href="#">?gehrd</a>
行列 Q を生成する	<a href="#">?orghr</a>	<a href="#">?unghr</a>
行列 Q を適用する	<a href="#">?ormhr</a>	<a href="#">?unmhr</a>
行列の平衡化	<a href="#">?gebal</a>	<a href="#">?gebal</a>
平衡化した行列の固有ベクトルを元の行列の固有ベクトルに変換する	<a href="#">?gebak</a>	<a href="#">?gebak</a>
固有値と Schur 因子分解を求める (QR アルゴリズム)	<a href="#">?hsegr</a>	<a href="#">?hsegr</a>
Hessenberg 形式から固有ベクトルを求める (反転繰り返し法)	<a href="#">?hsein</a>	<a href="#">?hsein</a>
Schur の因子分解から固有ベクトルを求める	<a href="#">?trevc</a>	<a href="#">?trevc</a>
固有値と固有ベクトルの条件数を見積もる	<a href="#">?trsna</a>	<a href="#">?trsna</a>
Schur の因子分解の順序を変更する	<a href="#">?trexc</a>	<a href="#">?trexc</a>
Schur の因子分解の順序変更、不変部分空間の検出、条件数の見積もり	<a href="#">?trsen</a>	<a href="#">?trsen</a>
シルベスター式を解く	<a href="#">?trsyl</a>	<a href="#">?trsyl</a>

図 4-4 デシジョン・ツリー：実非対称固有値問題

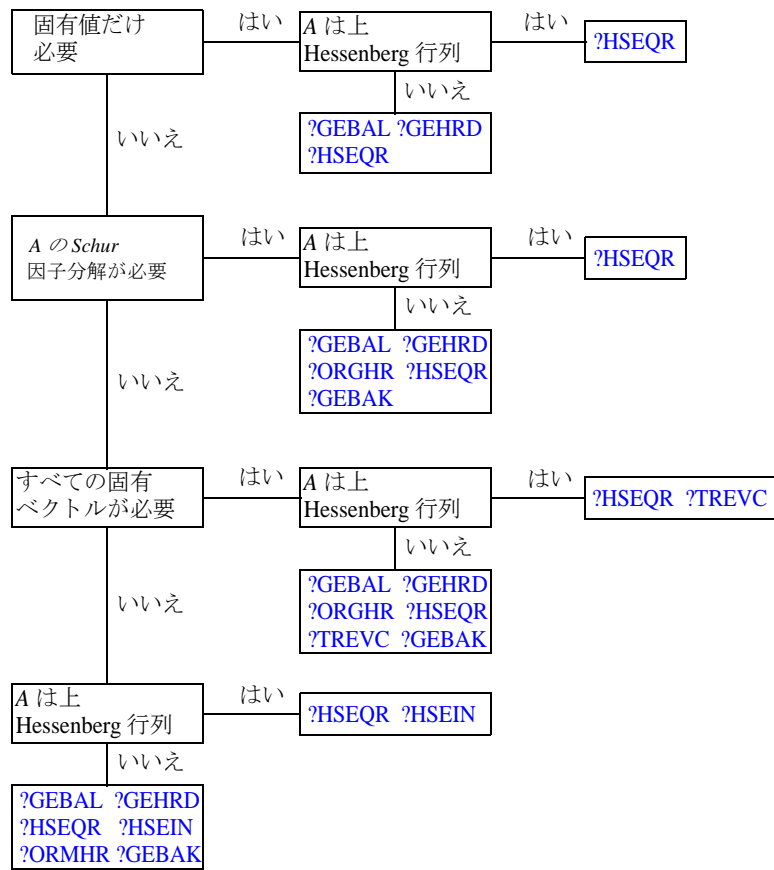
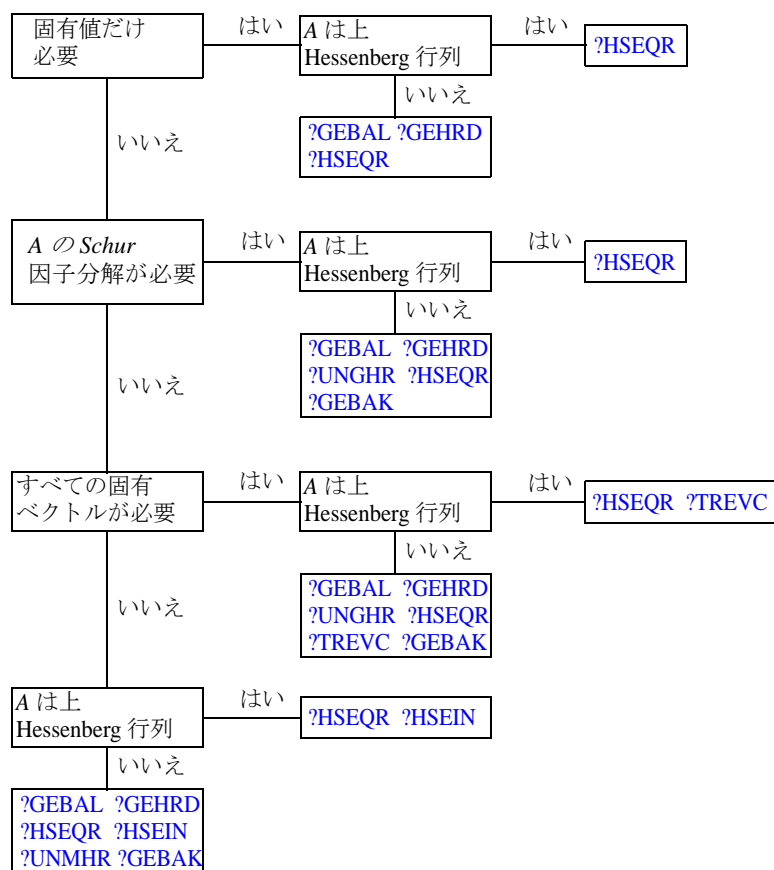


図 4-5 デシジョン・ツリー: 複素非エルミート固有値問題





## ?gehrd

一般行列を上 *Hessenberg* 形式に縮退させる。

### 構文

#### Fortran 77:

```
call sgehrd(n, ilo, ihi, a, lda, tau, work, lwork, info)
call dgehrd(n, ilo, ihi, a, lda, tau, work, lwork, info)
call cgehrd(n, ilo, ihi, a, lda, tau, work, lwork, info)
call zgehrd(n, ilo, ihi, a, lda, tau, work, lwork, info)
```

#### Fortran 95:

```
call gehrd(a [,tau] [,ilo] [,ihi] [,info])
```

### 説明

このルーチンは、直交またはユニタリの相似変換  $A = QHQ^H$  によって、一般行列  $A$  を上 *Hessenberg* 形式  $H$  に縮退させる。 $H$  は、実数の対角成分を持つ。

このルーチンでは、行列  $Q$  を明示的な形式で表現しない。代わりに、 $Q$  は基本リフレクタの積として表現される。一連のルーチンでは、この形式で表現される  $Q$  を操作する。

### 入力パラメータ

$n$	INTEGER。行列 $A$ の次数 ( $n \geq 0$ )。
$ilo, ihi$	INTEGER。 $A$ が ?gebal から出力された場合、 $ilo$ と $ihi$ に、そのルーチンから返された値を設定しなければならない。そうでない場合、 $ilo = 1$ かつ $ihi = n$ 。 ( $n > 0$ の場合、 $1 \leq ilo \leq ihi \leq n$ 。 $n = 0$ の場合、 $ilo = 1$ かつ $ihi = 0$ )。
$a, work$	REAL (sgehrd の場合 ) DOUBLE PRECISION (dgehrd の場合 ) COMPLEX (cgehrd の場合 ) DOUBLE COMPLEX (zgehrd の場合 )。 配列： $a(lda,*)$ には、行列 $A$ を格納する。 $a$ の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $work(lwork)$ は、ワークスペース配列である。

<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $\max(1, n)$ 以上。
<i>lwork</i>	INTEGER。 配列 <i>work</i> のサイズ。 $\max(1, n)$ 以上。 <i>lwork</i> = -1 の場合、ワークスペースのクエリとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリとして返す。 <i>xerbla</i> は <i>lwork</i> に関するエラー・メッセージを生成しない。 <i>lwork</i> の推奨値は、「アプリケーション・ノート」を参照。

### 出力パラメータ

<i>a</i>	上 Hessenberg 行列 <i>H</i> と行列 <i>Q</i> の各成分によって上書きされる。 <i>H</i> の劣対角成分は、実数である。
<i>tau</i>	REAL (sgehrd の場合) DOUBLE PRECISION (dgehrd の場合) COMPLEX (cgehrd の場合) DOUBLE COMPLEX (zgehrd の場合)。 配列、次元は $\max(1, n-1)$ 以上。 行列 <i>Q</i> に関するその他の情報も格納される。
<i>work</i> (1)	<i>info</i> = 0 の場合、終了時に、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が <i>work</i> (1) に格納される。以降の実行では、この <i>lwork</i> 値を使用する。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正であったことを示す。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *gehrd* のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ ( <i>n</i> , <i>n</i> ) の行列 <i>A</i> を格納する。
<i>tau</i>	長さ ( <i>n</i> -1) のベクトルを格納する。
<i>ilo</i>	この引数のデフォルト値は、 <i>ilo</i> = 1 である。
<i>ihi</i>	この引数のデフォルト値は、 <i>ihi</i> = <i>n</i> である。

## アプリケーション・ノート

パフォーマンスを改善するには、 $lwork = n * blocksize$  に設定する。 $blocksize$  は、ブロック・アルゴリズムの最適なパフォーマンスに必要な値で、マシンによって異なる（通常は 16 ～ 64）。必要なワークスペースの大きさがわからない場合は、最初の実行では  $lwork$  を大きめの値に設定する。終了時に、 $work(1)$  の値を調べ、以降はその値を使用する。

計算で求めた **Hessenberg** 行列  $H$  は、近似行列  $A + E$  の近似値である。ここで、 $\|E\|_2 < c(n)\varepsilon\|A\|_2$ 、 $c(n)$  は漸増する  $n$  の関数、 $\varepsilon$  はマシンによって異なる値である。

実数型の場合の浮動小数演算のおおよその総数は、 $(2/3)(ihi - ilo)^2(2ihi + 2ilo + 3n)$  である。複素数型の場合、この 4 倍になる。

## ?orghr

`?gehrd` で求めた実直交行列  $Q$  を生成する。

### 構文

#### Fortran 77:

```
call sorghr(n, ilo, ihi, a, lda, tau, work, lwork, info)
call dorghr(n, ilo, ihi, a, lda, tau, work, lwork, info)
```

#### Fortran 95:

```
call orghr(a, tau [,ilo] [,ihi] [,info])
```

### 説明

このルーチンは、`sgehrd/dgehrd` を呼び出して求めた直交行列  $Q$  を明示的に生成する（ルーチン `?gehrd` は、直交相似変換  $A = QHQ^T$  によって一般実行列  $A$  を上 **Hessenberg** 形式  $H$  に縮退させ、行列  $Q$  を  $(ihi - ilo)$  個の基本リフレクタの積として表現する。 $ilo$  と  $ihi$  は、行列を平衡化する際に `sgebal/dgebal` で求めた値である。行列が平衡化されていない場合、 $ilo = 1$  かつ  $ihi = n$  になる）。

?orghr によって生成される行列  $Q$  は、次の構造になる。

$$Q = \begin{bmatrix} I & 0 & 0 \\ 0 & Q_{22} & 0 \\ 0 & 0 & I \end{bmatrix}$$

$Q_{22}$  は、 $ilo \sim ihi$  の行と列を占める。

## 入力パラメータ

$n$	INTEGER。行列 $Q$ の次数 ( $n \geq 0$ )。
$ilo, ihi$	INTEGER。これらの値は、それぞれ ?gehrd に指定した $ilo$ および $ihi$ と同じでなければならない。( $n > 0$ の場合、 $1 \leq ilo \leq ihi \leq n$ 。 $n = 0$ の場合、 $ilo = 1$ かつ $ihi = 0$ )。
$a, tau, work$	REAL (sorghr の場合) DOUBLE PRECISION (dorghr の場合) 配列: $a(lda, *)$ には、?gehrd から返された一連のベクトル (基本リフレクタを定義) の各成分を格納する。 $a$ の第 2 次元は、 $\max(1, n)$ 以上でなければならない。  $tau(*)$ には、?gehrd から返された基本リフレクタの各成分を格納する。 $tau$ の次元は、 $\max(1, n-1)$ 以上でなければならない。  $work(lwork)$ は、ワークスペース配列である。
$lda$	INTEGER。 $a$ の第 1 次元。 $\max(1, n)$ 以上。
$lwork$	INTEGER。配列 $work$ のサイズ。 $lwork \geq \max(1, ihi-ilo)$ 。 $lwork = -1$ の場合、ワークスペースのクエリとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエントリとして返す。xerbla は $lwork$ に関するエラー・メッセージを生成しない。  $lwork$ の推奨値は、「アプリケーション・ノート」を参照。

## 出力パラメータ

$a$	$n \times n$ の直交行列 $Q$ によって上書きされる。
-----	------------------------------------

<code>work(1)</code>	<code>info = 0</code> の場合、終了時に、最適なパフォーマンスを得るために必要な <code>lwork</code> の最小値が <code>work(1)</code> に格納される。以降の実行では、この <code>lwork</code> 値を使用する。
<code>info</code>	INTEGER。 <code>info = 0</code> の場合、実行は正常に終了したことを示す。 <code>info = -i</code> の場合、 <code>i</code> 番目のパラメータの値が不正であったことを示す。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `orghr` のインターフェイスの詳細を以下に示す。

<code>a</code>	サイズ $(n, n)$ の行列 $A$ を格納する。
<code>tau</code>	長さ $(n-1)$ のベクトルを格納する。
<code>ilo</code>	この引数のデフォルト値は、 <code>ilo = 1</code> である。
<code>ihi</code>	この引数のデフォルト値は、 <code>ihi = n</code> である。

### アプリケーション・ノート

パフォーマンスを改善するには、`lwork = (ihi-ilo)*blocksize` に設定する。`blocksize` は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。必要なワークスペースの大きさがわからない場合は、最初の実行では `lwork` を大きめの値に設定する。終了時に、`work(1)` の値を調べ、以降はその値を使用する。

計算で求めた行列  $Q$  は、正確な値と行列  $E$  ( $\|E\|_2 = O(\epsilon)$ 、 $\epsilon$  はマシン精度) だけ異なる。

浮動小数演算のおおよその総数は、 $(4/3)(ihi-ilo)^3$  である。

このルーチンの複素数版は、[?ungahr](#) である。

### ?ormhr

任意の実行列  $C$  に ?gehrd で求めた  
実直交行列  $Q$  を掛ける。

---

#### 構文

##### Fortran 77:

```
call sormhr(side, trans, m, n, ilo, ihi, a, lda, tau, c, ldc,  
            work, lwork, info)  
call dormhr(side, trans, m, n, ilo, ihi, a, lda, tau, c, ldc,  
            work, lwork, info)
```

##### Fortran 95:

```
call ormhr(a, tau, c [,ilo] [,ihi] [,side] [,trans] [,info])
```

#### 説明

このルーチンは、行列  $C$  に sgehrd/dgehrd を呼び出して求めた直交行列  $Q$  を掛ける  
(ルーチン ?gehrd は、直交相似変換  $A = QHQ^T$  によって一般実行列  $A$  を上 Hessenberg 形  
式  $H$  に縮退させ、行列  $Q$  を  $(ihi-ilo)$  個の基本リフレクタの積として表現する。 $ilo$  と  
 $ihi$  は、行列を平衡化する際に sgebal/dgebal で求めた値である。行列が平衡化されて  
いない場合、 $ilo = 1$  かつ  $ihi = n$  になる)。

?ormhr を使用すれば、 $QC$ 、 $Q^TC$ 、 $CQ$ 、または  $CQ^T$  のいずれかの行列積を求めること  
ができる。その場合、 $C$  (実矩形行列) の値が上書きされる。

?ormhr は、 $H$  の固有ベクトルで構成される行列  $V$  を、 $A$  の固有ベクトルで構成される  
行列  $QV$  に変換する際によく使用する。

#### 入力パラメータ

side	CHARACTER*1。'L' または 'R' でなければならない。 side = 'L' の場合、 $QC$ または $Q^TC$ を求める。 side = 'R' の場合、 $CQ$ または $CQ^T$ を求める。
trans	CHARACTER*1。'N' または 'T' でなければならない。 trans = 'N' の場合、 $Q$ が $C$ に適用される。 trans = 'T' の場合、 $Q^T$ が $C$ に適用される。
m	INTEGER。行列 $C$ の行数 ( $m \geq 0$ )。

<i>n</i>	INTEGER。行列 <i>C</i> の列数 ( $n \geq 0$ )。
<i>ilo, ihi</i>	INTEGER。これらの値は、それぞれ ?gehrd に指定した <i>ilo</i> および <i>ihi</i> と同じでなければならない。 $m > 0$ かつ <i>side</i> = 'L' の場合、 $1 \leq ilo \leq ihi \leq m$ 。 $m = 0$ かつ <i>side</i> = 'L' の場合、 $ilo = 1$ かつ $ihi = 0$ 。 $n > 0$ かつ <i>side</i> = 'R' の場合、 $1 \leq ilo \leq ihi \leq n$ 。 $n = 0$ かつ <i>side</i> = 'R' の場合、 $ilo = 1$ かつ $ihi = 0$ 。
<i>a, tau, c, work</i>	REAL (sormhr の場合) DOUBLE PRECISION (dormhr の場合) 配列： <i>a</i> ( <i>lda</i> ,*) には、?gehrd から返された一連のベクトル (基本リフレクタを定義) の各成分を格納する。 <i>a</i> の第 2 次元は、 $\max(1, m)$ 以上 ( <i>side</i> = 'L' の場合) または $\max(1, n)$ 以上 ( <i>side</i> = 'R' の場合) でなければならない。 <i>tau</i> (*) には、?gehrd から返された基本リフレクタの各成分を格納する。 <i>tau</i> の次元は、 $\max(1, m-1)$ 以上 ( <i>side</i> = 'L' の場合) または $\max(1, n-1)$ 以上 ( <i>side</i> = 'R' の場合) でなければならない。 <i>c</i> ( <i>ldc</i> ,*) には、 $m \times n$ の行列 <i>C</i> を格納する。 <i>c</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>work</i> ( <i>lwork</i> ) は、ワークスペース配列である。
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $\max(1, m)$ 以上 ( <i>side</i> = 'L' の場合) または $\max(1, n)$ ( <i>side</i> = 'R' の場合)。
<i>ldc</i>	INTEGER。 <i>c</i> の第 1 次元。 $\max(1, m)$ 以上。
<i>lwork</i>	INTEGER。配列 <i>work</i> のサイズ。 <i>side</i> = 'L' の場合、 $lwork \geq \max(1, n)$ 。 <i>side</i> = 'R' の場合、 $lwork \geq \max(1, m)$ 。 <i>lwork</i> = -1 の場合、ワークスペースのクエリとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリとして返す。xerbla は <i>lwork</i> に関するエラー・メッセージを生成しない。 <i>lwork</i> の推奨値は、「アプリケーション・ノート」を参照。

### 出力パラメータ

<i>c</i>	<i>C</i> は、 <i>side</i> と <i>trans</i> の値に従って、 $QC$ 、 $Q^TC$ 、 $CQ^T$ 、または $CQ$ のいずれかによって上書きされる。
----------	--

<i>work(1)</i>	<i>info</i> = 0 の場合、終了時に、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が <i>work(1)</i> に格納される。以降の実行では、この <i>lwork</i> 値を使用する。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正であったことを示す。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *ormhr* のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ ( <i>r</i> , <i>r</i> ) の行列 <i>A</i> を格納する。 <i>r</i> = <i>m</i> ( <i>side</i> = 'L' の場合)。 <i>r</i> = <i>n</i> ( <i>side</i> = 'R' の場合)。
<i>tau</i>	長さ ( <i>r</i> -1) のベクトルを格納する。
<i>c</i>	サイズ ( <i>m</i> , <i>n</i> ) の行列 <i>C</i> を格納する。
<i>ilo</i>	この引数のデフォルト値は、 <i>ilo</i> = 1 である。
<i>ihi</i>	この引数のデフォルト値は、 <i>ihi</i> = <i>n</i> である。
<i>side</i>	'L' または 'R' でなければならない。デフォルト値は 'L'。
<i>trans</i>	'N' または 'T' でなければならない。デフォルト値は 'N'。

### アプリケーション・ノート

パフォーマンスを改善するには、*lwork* を、*side* = 'L' の場合は  $n \cdot \text{blocksize}$  以上に、*side* = 'R' の場合は  $m \cdot \text{blocksize}$  以上に設定する。*blocksize* は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。必要なワークスペースの大きさがわからない場合は、最初の実行では *lwork* を大きめの値に設定する。終了時に、*work(1)* の値を調べ、以降はその値を使用する。

計算で求めた行列 *Q* は、正確な値と行列 *E* ( $\|E\|_2 = O(\varepsilon)\|C\|_2$ 、 $\varepsilon$  はマシン精度) だけ異なる。

浮動小数演算のおおよその総数は、次のとおりである。

$2n(ihi-ilo)^2$  (*side* = 'L' の場合)、  
 $2m(ihi-ilo)^2$  (*side* = 'R' の場合)。



このルーチンの複素数版は、[?unmhr](#) である。

## ?unghr

?gehrd で求めた複素ユニタリ行列  $Q$  を生成する。

### 構文

#### Fortran 77:

```
call cunghr(n, ilo, ihi, a, lda, tau, work, lwork, info)
call zunghr(n, ilo, ihi, a, lda, tau, work, lwork, info)
```

#### Fortran 95:

```
call unghr(a, tau [,ilo] [,ihi] [,info])
```

このルーチンは、cgehrd/zgehrd (ユニタリ相似変換  $A = QHQ^H$  によって、複素行列  $A$  を上 Hessenberg 形式  $H$  に縮退させる) を呼び出した後で使用する。?gehrd は、行列  $Q$  を  $(ihi-ilo)$  個の基本リフレクタの積として表現する。 $ilo$  と  $ihi$  は、行列を平衡化する際に cgebal/zgebal で求めた値である。行列が平衡化されていない場合は、 $ilo=1$  かつ  $ihi=n$  になる)。

$Q$  を正方行列として明示的に生成するには、ルーチン ?unghr を使う。行列  $Q$  は、次の構造になる。

$$Q = \begin{bmatrix} I & 0 & 0 \\ 0 & Q_{22} & 0 \\ 0 & 0 & I \end{bmatrix}$$

$Q_{22}$  は、 $ilo \sim ihi$  の行と列を占める。

### 入力パラメータ

$n$	INTEGER。行列 $Q$ の次数 ( $n \geq 0$ )。
$ilo, ihi$	INTEGER。これらの値は、それぞれ ?gehrd に指定した $ilo$ および $ihi$ と同じでなければならない。( $n > 0$ の場合、 $1 \leq ilo \leq ihi \leq n$ 。 $n = 0$ の場合、 $ilo = 1$ かつ $ihi = 0$ )。

*a*, *tau*, *work*    COMPLEX (cunghr の場合)  
                       DOUBLE COMPLEX (zunghr の場合)。  
 配列:  
*a*(*lda*,\*) には、?gehrd から返された一連のベクトル (基本リフレクタを定義) の各成分を格納する。  
*a* の第 2 次元は、 $\max(1, n)$  以上でなければならない。  
*tau*(\*) には、?gehrd から返された基本リフレクタの各成分を格納する。  
*tau* の次元は、 $\max(1, n-1)$  以上でなければならない。  
*work*(*lwork*) は、ワークスペース配列である。

*lda*                INTEGER。 *a* の第 1 次元。  $\max(1, n)$  以上。

*lwork*            INTEGER。 配列 *work* のサイズ。  
 $lwork \geq \max(1, ihi-ilo)$ 。  
*lwork* = -1 の場合、ワークスペースのクエリとみなされ、ルーチンは *work* 配列の最適サイズだけを計算し、その値を *work* 配列の最初のエントリとして返す。xerbla は *lwork* に関するエラー・メッセージを生成しない。  
*lwork* の推奨値は、「アプリケーション・ノート」を参照。

## 出力パラメータ

*a*                 $n \times n$  のユニタリ行列 *Q* によって上書きされる。

*work*(1)        *info* = 0 の場合、終了時に、最適なパフォーマンスを得るために必要な *lwork* の最小値が *work*(1) に格納される。以降の実行では、この *lwork* 値を使用する。

*info*            INTEGER。  
*info* = 0 の場合、実行は正常に終了したことを示す。  
*info* = -*i* の場合、*i* 番目のパラメータの値が不正であったことを示す。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン unghr のインターフェイスの詳細を以下に示す。

*a*                サイズ (*n*,*n*) の行列 *A* を格納する。

*tau*                    長さ  $(n-1)$  のベクトルを格納する。  
*ilo*                    この引数のデフォルト値は、 $ilo = 1$  である。  
*ihi*                    この引数のデフォルト値は、 $ihi = n$  である。

### アプリケーション・ノート

パフォーマンスを改善するには、 $lwork = (ihi-ilo)*blocksize$  に設定する。*blocksize* は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。必要なワークスペースの大きさがわからない場合は、最初の実行では *lwork* を大きめの値に設定する。終了時に、*work*(1) の値を調べ、以降はその値を使用する。

計算で求めた行列  $Q$  は、正確な値と行列  $E$  ( $\|E\|_2 = O(\epsilon)$ 、 $\epsilon$  はマシン精度) だけ異なる。

実数型の場合の浮動小数演算のおおよその総数は、 $(16/3)(ihi-ilo)^3$  である。

このルーチンの実数版は、[?orgqr](#) である。

### ?unmhr

任意の複素行列  $C$  に ?gehrd で求めた  
複素ユニタリ行列  $Q$  を掛ける。

---

#### 構文

##### Fortran 77:

```
call cunmhr(side, trans, m, n, ilo, ihi, a, lda, tau, c, ldc,  
            work, lwork, info)  
call zunmhr(side, trans, m, n, ilo, ihi, a, lda, tau, c, ldc,  
            work, lwork, info)
```

##### Fortran 95:

```
call unmhr(a, tau, c [,ilo] [,ihi] [,side] [,trans] [,info])
```

#### 説明

このルーチンは、行列  $C$  に cgehrd/zgehrd を呼び出して求めたユニタリ行列  $Q$  を掛ける (ルーチン ?gehrd は、直交相似変換  $A = QHQ^H$  によって一般実行列  $A$  を上 Hessenberg 形式  $H$  に縮退させ、行列  $Q$  を  $(ihi-ilo)$  個の基本リフレクタの積として表現する。  $ilo$  と  $ihi$  は、行列を平衡化する際に cgebal/zgebal で求めた値である。行列が平衡化されていない場合は、  $ilo = 1$  かつ  $ihi = n$  になる)。

?unmhr を使用すれば、  $QC$ 、  $Q^HC$ 、  $CQ$ 、または  $CQ^H$  のいずれかの行列積を求めることができる。その場合、  $C$  (複素矩形行列) の値が上書きされる。このルーチンは、  $H$  の固有ベクトルで構成される行列  $V$  を、  $A$  の固有ベクトルで構成される行列  $QV$  に変換する際によく使われる。

#### 入力パラメータ

side	CHARACTER*1。 'L' または 'R' でなければならない。 side = 'L' の場合、 $QC$ または $Q^HC$ を求める。 side = 'R' の場合、 $CQ$ または $CQ^H$ を求める。
trans	CHARACTER*1。 'N' または 'C' でなければならない。 trans = 'N' の場合、 $Q$ が $C$ に適用される。 trans = 'T' の場合、 $Q^H$ が $C$ に適用される。
m	INTEGER。 行列 $C$ の行数 ( $m \geq 0$ )。
n	INTEGER。 行列 $C$ の列数 ( $n \geq 0$ )。

<i>ilo, ihi</i>	<p>INTEGER。これらの値は、それぞれ ?gehrd に指定した <i>ilo</i> および <i>ihi</i> と同じでなければならない。</p> <p><math>m &gt; 0</math> かつ <i>side</i> = 'L' の場合、<math>1 \leq ilo \leq ihi \leq m</math>。</p> <p><math>m = 0</math> かつ <i>side</i> = 'L' の場合、<math>ilo = 1</math> かつ <math>ihi = 0</math>。</p> <p><math>n &gt; 0</math> かつ <i>side</i> = 'R' の場合、<math>1 \leq ilo \leq ihi \leq n</math>。</p> <p><math>n = 0</math> かつ <i>side</i> = 'R' の場合、<math>ilo = 1</math> かつ <math>ihi = 0</math>。</p>
<i>a, tau, c, work</i>	<p>COMPLEX (cunmhr の場合)</p> <p>DOUBLE COMPLEX (zunmhr の場合)。</p> <p>配列：</p> <p><i>a</i>(<i>lda</i>,*) には、?gehrd から返された一連のベクトル (基本リフレクタを定義) の各成分を格納する。</p> <p><i>a</i> の第 2 次元は、<math>\max(1, m)</math> 以上 (<i>side</i> = 'L' の場合) または <math>\max(1, n)</math> 以上 (<i>side</i> = 'R' の場合) でなければならない。</p> <p><i>tau</i>(*) には、?gehrd から返された基本リフレクタの各成分を格納する。</p> <p><i>tau</i> の次元は、<math>\max(1, m-1)</math> 以上 (<i>side</i> = 'L' の場合) または <math>\max(1, n-1)</math> 以上 (<i>side</i> = 'R' の場合) でなければならない。</p> <p><i>c</i>(<i>ldc</i>,*) には、<math>m \times n</math> の行列 <i>C</i> を格納する。</p> <p><i>c</i> の第 2 次元は、<math>\max(1, n)</math> 以上でなければならない。</p> <p><i>work</i>(<i>lwork</i>) は、ワークスペース配列である。</p>
<i>lda</i>	<p>INTEGER。 <i>a</i> の第 1 次元。 <math>\max(1, m)</math> 以上 (<i>side</i> = 'L' の場合) または <math>\max(1, n)</math> (<i>side</i> = 'R' の場合)。</p>
<i>ldc</i>	<p>INTEGER。 <i>c</i> の第 1 次元。 <math>\max(1, m)</math> 以上。</p>
<i>lwork</i>	<p>INTEGER。 配列 <i>work</i> のサイズ。</p> <p><i>side</i> = 'L' の場合、<math>lwork \geq \max(1, n)</math>。</p> <p><i>side</i> = 'R' の場合、<math>lwork \geq \max(1, m)</math>。</p> <p><i>lwork</i> = -1 の場合、ワークスペースのクエリとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリとして返す。xerbla は <i>lwork</i> に関するエラー・メッセージを生成しない。</p> <p><i>lwork</i> の推奨値は、「アプリケーション・ノート」を参照。</p>

## 出力パラメータ

<i>c</i>	<p><i>C</i> は、<i>side</i> と <i>trans</i> の値に従って、<math>QC</math>、<math>Q^H C</math>、<math>CQ^H</math>、または <math>CQ</math> のいずれかによって上書きされる。</p>
----------	---

<i>work(1)</i>	<i>info</i> = 0 の場合、終了時に、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が <i>work(1)</i> に格納される。以降の実行では、この <i>lwork</i> 値を使用する。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正であったことを示す。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *unmhr* のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ ( <i>r</i> , <i>r</i> ) の行列 <i>A</i> を格納する。 <i>r</i> = <i>m</i> ( <i>side</i> = 'L' の場合)。 <i>r</i> = <i>n</i> ( <i>side</i> = 'R' の場合)。
<i>tau</i>	長さ ( <i>r</i> -1) のベクトルを格納する。
<i>c</i>	サイズ ( <i>m</i> , <i>n</i> ) の行列 <i>C</i> を格納する。
<i>ilo</i>	この引数のデフォルト値は、 <i>ilo</i> = 1 である。
<i>ihi</i>	この引数のデフォルト値は、 <i>ihi</i> = <i>n</i> である。
<i>side</i>	'L' または 'R' でなければならない。デフォルト値は 'L'。
<i>trans</i>	'N' または 'C' でなければならない。デフォルト値は 'N'。

### アプリケーション・ノート

パフォーマンスを改善するには、*lwork* を、*side* = 'L' の場合は  $n \cdot \text{blocksize}$  以上に、*side* = 'R' の場合は  $m \cdot \text{blocksize}$  以上に設定する。*blocksize* は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。必要なワークスペースの大きさがわからない場合は、最初の実行では *lwork* を大きめの値に設定する。終了時に、*work(1)* の値を調べ、以降はその値を使用する。

計算で求めた行列 *Q* は、正確な値と行列 *E* ( $\|E\|_2 = O(\varepsilon)\|C\|_2$ ,  $\varepsilon$  はマシン精度) だけ異なる。

浮動小数演算のおおよその総数は、次のとおりである。

$$8n(ihi-ilo)^2 \text{ (} side = 'L' \text{ の場合)},$$

$$8m(ihi-ilo)^2 \text{ (} side = 'R' \text{ の場合)}.$$

このルーチンの実数版は、[?ormhr](#) である。

## ?gebal

一般行列を平衡化し、固有値および  
固有ベクトルの計算精度を改善する。

### 構文

#### Fortran 77:

```
call sgebal(job, n, a, lda, ilo, ihi, scale, info)
call dgebal(job, n, a, lda, ilo, ihi, scale, info)
call cgebal(job, n, a, lda, ilo, ihi, scale, info)
call zgebal(job, n, a, lda, ilo, ihi, scale, info)
```

#### Fortran 95:

```
call gebal(a [,scale] [,ilo] [,ihi] [,job] [,info])
```

### 説明

このルーチンは、次の 2 つの相似変換の一方または両方を実行して、行列  $A$  を平衡化する。

(1) まず、 $A$  をブロック上三角形式に置換してみる。 $P$  は置換行列、 $A'_{11}$  と  $A'_{33}$  は上三角

$$PAP^T = A' = \begin{bmatrix} A'_{11} & A'_{12} & A'_{13} \\ 0 & A'_{22} & A'_{23} \\ 0 & 0 & A'_{33} \end{bmatrix}$$

行列である。 $A'_{11}$  と  $A'_{33}$  の対角成分は、 $A$  の固有値である。 $A$  の残りの固有値は、中央の対角ブロック  $A'_{22}$  の固有値 ( $ilo \sim ihi$  の行と列) である。 $A$  の固有値 (またはその Schur 因子分解) を計算するためのこれ以降の演算は、この範囲の行と列に対して適用するだけで済む。そのため、 $ilo > 1$  かつ  $ihi < n$  の場合、作業を大幅に節約できる。適切な置換行列が存在しない場合 (存在しない場合が多い) は、 $ilo = 1$  かつ  $ihi = n$  と設定され、 $A'_{22}$  が  $A$  の全体になる。

(2) 対角相似変換を  $A'$  に適用し、 $A'_{22}$  の行と列をノルム内でできるだけ近づける。

$$A'' = DA'D^{-1} = \begin{bmatrix} I & 0 & 0 \\ 0 & D_{22} & 0 \\ 0 & 0 & I \end{bmatrix} \times \begin{bmatrix} A'_{11} & A'_{12} & A'_{13} \\ 0 & A'_{22} & A'_{23} \\ 0 & 0 & A'_{33} \end{bmatrix} \times \begin{bmatrix} I & 0 & 0 \\ 0 & D_{22}^{-1} & 0 \\ 0 & 0 & I \end{bmatrix}$$

このスケーリングによって、この行列のノルムが縮退される（つまり、 $\|A'_{22}\| < \|A'_{22}\|$ ）。その結果、固有値と固有ベクトルの計算精度に対する丸め誤差の影響が減る。

## 入力パラメータ

<i>job</i>	CHARACTER*1。'N'、'P'、'S'、または 'B' のいずれかでなければならない。 <i>job</i> ='N' の場合、 <i>A</i> の置換およびスケーリングは実行されない（ただし、 <i>ilo</i> 、 <i>ihi</i> 、 <i>scale</i> は、入力された値を受け取る）。 <i>job</i> ='P' の場合、 <i>A</i> の置換が実行され、スケーリングは実行されない。 <i>job</i> ='S' の場合、 <i>A</i> のスケーリングが実行され、置換は実行されない。 <i>job</i> ='B' の場合、 <i>A</i> のスケーリングと置換が両方とも実行される。
<i>n</i>	INTEGER。行列 <i>A</i> の次数 ( $n \geq 0$ )。
<i>a</i>	REAL (sgebal の場合) DOUBLE PRECISION (dgebal の場合) COMPLEX (cgebal の場合) DOUBLE COMPLEX (zgebal の場合)。 配列： <i>a</i> ( <i>lda</i> ,*) には、行列 <i>A</i> を格納する。 <i>a</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>job</i> ='N' の場合、 <i>a</i> は参照されない。
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $\max(1, n)$ 以上。

## 出力パラメータ

<i>a</i>	平衡化された行列によって上書きされる ( <i>job</i> ='N' の場合、 <i>a</i> は参照されない)。
<i>ilo</i> , <i>ihi</i>	INTEGER。 <i>ilo</i> および <i>ihi</i> は、終了時に、 <i>a</i> ( <i>i</i> , <i>j</i> ) が 0 となるような値 ( $i > j$ かつ $1 \leq j < ilo$ あるいは $ihi < i \leq n$ の場合。 <i>job</i> ='N' または 'S' の場合、 <i>ilo</i> =1、 <i>ihi</i> = <i>n</i> 。)



<i>scale</i>	<p>REAL ( 単精度の場合 )  DOUBLE PRECISION ( 倍精度の場合 )  配列、次元は <math>\max(1, n)</math> 以上。</p> <p>置換とスケーリング係数の各成分が格納される。</p> <p>もう少し詳しく説明すると、<math>p_j</math> が行と列 <math>j</math> と交換された行と列のインデックスであり、<math>d_j</math> が行と列 <math>j</math> を平衡化する際に使ったスケーリング係数である場合は、次のようになる。  <math>scale(j) = p_j (j = 1, 2, \dots, ilo-1, ihi+1, \dots, n)</math>。  <math>scale(j) = d_j (j = ilo, ilo+1, \dots, ihi)</math>。  交換の実行順序は、<math>n \sim ihi+1</math>、次に、<math>1 \sim ilo-1</math> になる。</p>
<i>info</i>	<p>INTEGER。</p> <p><math>info = 0</math> の場合、実行は正常に終了したことを示す。  <math>info = -i</math> の場合、<math>i</math> 番目のパラメータの値が不正であったことを示す。</p>

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `gebal` のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ $(n, n)$ の行列 $A$ を格納する。
<i>scale</i>	長さ $(n)$ のベクトルを格納する。
<i>ilo</i>	この引数のデフォルト値は、 $ilo = 1$ である。
<i>ihi</i>	この引数のデフォルト値は、 $ihi = n$ である。
<i>job</i>	'B'、'S'、'P'、または 'N' でなければならない。デフォルト値は 'B'。

### アプリケーション・ノート

誤差は、これ以降の計算における誤差に比べて無視できる程度である。

行列  $A$  をこのルーチンによって平衡化した場合は、以降に計算する固有ベクトルはすべて、行列  $A''$  の固有ベクトルになる。そのため、[?gebak](#) を呼び出して、それらを  $A$  の固有ベクトルに変換しなければならない。

$A$  の Schur ベクトルが必要な場合は、 $job = 'S'$  または  $'B'$  と指定してこのルーチン呼び出してはならない。つまり、これらを指定すると、平衡化のための変換が直交 (複素数型の場合はユニタリ) にならないからである。 $job = 'P'$  と指定してこのルーチン呼び出すと、以降に計算する Schur ベクトルはすべて、行列  $A''$  の Schur ベクトルになる。そのため、[?gebak](#) ( $side = 'R'$  と指定) を呼び出して、それらを  $A$  の Schur ベクトルに変換し直す必要がある。

浮動小数演算の総数は、 $n^2$  に比例する。

---

### ?gebak

平衡化後の行列の固有ベクトルを、元の非対称行列の固有ベクトルに変換する。

---

#### 構文

##### Fortran 77:

```
call sgebak(job, side, n, ilo, ihi, scale, m, v, ldv, info)
call dgebak(job, side, n, ilo, ihi, scale, m, v, ldv, info)
call cgebak(job, side, n, ilo, ihi, scale, m, v, ldv, info)
call zgebak(job, side, n, ilo, ihi, scale, m, v, ldv, info)
```

##### Fortran 95:

```
call gebak(v, scale [,ilo] [,ihi] [,job] [,side] [,info])
```

#### 説明

このルーチンは、[?gebal](#) を呼び出して行列  $A$  を平衡化した後、平衡化後の行列  $A''_{22}$  の固有ベクトルを計算してから使用する。

平衡化については、[?gebal](#) を参照。平衡化後の行列  $A''$  は、 $A'' = DPAP^T D^{-1}$  として得られる ( $P$  は置換行列、 $D$  はスケーリング用の対角行列)。このルーチンでは、固有ベクトルを次のように変換する。

$x$  が  $A''$  の右固有ベクトルである場合、 $P^T D^{-1} x$  が  $A$  の右固有ベクトルになる。

$x$  が  $A''$  の左固有ベクトルである場合、 $P^T D y$  が  $A$  の左固有ベクトルになる。

**入力パラメータ**

<i>job</i>	CHARACTER*1. 'N'、'P'、'S'、または 'B' のいずれかでなければならない。 ?gebal に指定した <i>job</i> と同じ値を使用する。
<i>side</i>	CHARACTER*1. 'L' または 'R' でなければならない。 <i>side</i> ='L' の場合、左固有ベクトルが変換される。 <i>side</i> ='R' の場合、右固有ベクトルが変換される。
<i>n</i>	INTEGER. 固有ベクトルで構成される行列の行数 ( $n \geq 0$ ).
<i>ilo, ihi</i>	INTEGER. <i>ilo</i> と <i>ihi</i> の値は、?gebal から返された値である。 ( $n > 0$ の場合、 $1 \leq ilo \leq ihi \leq n$ 。 $n = 0$ の場合、 $ilo = 1$ かつ $ihi = 0$ ).
<i>scale</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列、次元は $\max(1, n)$ 以上。  元の一般行列を平衡化するために使った置換やスケーリング係数の各成分 (?gebal から返されたもの) を格納する。
<i>m</i>	INTEGER. 固有ベクトルで構成される <i>s</i> 列の列数 ( $m \geq 0$ ).
<i>v</i>	REAL (sgebak の場合) DOUBLE PRECISION (dgebak の場合) COMPLEX (cgebak の場合) DOUBLE COMPLEX (zgebak の場合)。 配列： <i>v</i> ( <i>ldv</i> ,*) には、変換対象の左または右の固有ベクトルで構成される行列を格納する。 <i>v</i> の第 2 次元は、 $\max(1, m)$ 以上でなければならない。
<i>ldv</i>	INTEGER. <i>v</i> の第 1 次元。 $\max(1, n)$ 以上。

**出力パラメータ**

<i>v</i>	変換後の固有ベクトルによって上書きされる。
<i>info</i>	INTEGER. <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正であったことを示す。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `gebak` のインターフェイスの詳細を以下に示す。

<code>v</code>	サイズ $(n,m)$ の行列 $V$ を格納する。
<code>scale</code>	長さ $(n)$ のベクトルを格納する。
<code>ilo</code>	この引数のデフォルト値は、 <code>ilo = 1</code> である。
<code>ihi</code>	この引数のデフォルト値は、 <code>ihi = n</code> である。
<code>job</code>	'B'、'S'、'P'、または 'N' でなければならない。デフォルト値は 'B'。
<code>side</code>	'L' または 'R' でなければならない。デフォルト値は 'L'。

### アプリケーション・ノート

このルーチンでの誤差は、無視できる程度である。

浮動小数演算のおおよその総数は、 $m \cdot n$  に比例する。

---

## ?hseqr

*Hessenberg* 形式に縮退された行列の固有値を  
すべて計算し、( オプションで ) *Schur* 因子分解を  
行う。

---

### 構文

#### Fortran 77:

```
call shseqr(job, compz, n, ilo, ihi, h, ldh, wr, wi, z, ldz, work, lwork, info)
call dhseqr(job, compz, n, ilo, ihi, h, ldh, wr, wi, z, ldz, work, lwork, info)
call chseqr(job, compz, n, ilo, ihi, h, ldh, w, z, ldz, work, lwork, info)
call zhseqr(job, compz, n, ilo, ihi, h, ldh, w, z, ldz, work, lwork, info)
```

**Fortran 95:**

```
call hseqr(h, wr, wi [,ilo] [,ihi] [,z] [,job] [,compz] [,info])
call hseqr(h, w [,ilo] [,ihi] [,z] [,job] [,compz] [,info])
```

**説明**

このルーチンは、上 Hessenberg 行列  $H: H = ZTZ^H$  の固有値をすべて計算し、オプションで、Schur 因子分解を行う。 $T$  は、上三角 (実数型の場合は準三角) 行列 ( $H$  の Schur 形式) である。 $Z$  は、ユニタリ行列または直交行列で、列は Schur ベクトル  $z_i$  である。

このルーチンを使用すれば、上 Hessenberg 形式  $H: A = QHQ^H$  ( $Q$  はユニタリ行列で、実数型の場合は直交行列) に縮退されている一般行列  $A$  の Schur 因子分解を求めることができる。

$A = (QZ)T(QZ)^H$  である。

この場合、[?gehrd](#) によって  $A$  を Hessenberg 形式に縮退させた後、[?orghr](#) を呼び出して  $Q$  を明示的に求め、`compz = 'V'` と指定して  $Q$  を [?hseqr](#) に渡す。

[?gebal](#) を呼び出して、[?hseqr](#) を使って Hessenberg 形式に縮退させる前に、元の行列を平衡化することもできる。Hessenberg の行列  $H$  は、次の構造を持つ。

$$\begin{bmatrix} H_{11} & H_{12} & H_{13} \\ 0 & H_{22} & H_{23} \\ 0 & 0 & H_{33} \end{bmatrix}$$

$H_{11}$  と  $H_{33}$  は、上三角行列である。

この場合、中央の対角ブロック  $H_{22}$  ( $ilo \sim ihi$  の行と列) だけをさらに Schur 形式に縮退させる必要がある ( $H_{12}$  と  $H_{23}$  のブロックも影響を受ける)。したがって、 $ilo$  と  $ihi$  の値を、[?hseqr](#) に直接与えられる。また、このルーチンを呼び出した後は、[?gebak](#) を呼び出して、平衡化後の行列の Schur ベクトルを置換して、元の行列の Schur ベクトルにしなければならない。

ただし、[?gebal](#) を呼び出さなかった場合は、 $ilo$  に 1 を、 $ihi$  に  $n$  を設定しなければならない。 $A$  の Schur 因子分解が必要な場合は、`job = 'S'` または `'B'` を指定して [?gebal](#) を呼び出してはならない。つまり、これらを指定すると、平衡化のための変換がユニタリ (実数型の場合は直交) にならないからである。

?hseqr では、上 Hessenberg の *QR* アルゴリズムのマルチシフト形式を使う。Schur ベクトルは、 $\|z\|_2 = 1$  になるように正規化される。ただし、絶対値が 1 となる複素数の係数 (実数型の場合は係数  $\pm 1$ ) の範囲に収まる。

## 入力パラメータ

<i>job</i>	CHARACTER*1。'E' または 'S' でなければならない。 <i>job</i> ='E' の場合、固有値だけを求める。 <i>job</i> ='S' の場合、Schur 形式 <i>T</i> を求める。
<i>compz</i>	CHARACTER*1。'N'、'I'、または 'V' のいずれかでなければならない。 <i>compz</i> ='N' の場合、Schur ベクトルは計算されない (配列 <i>z</i> は参照されない)。 <i>compz</i> ='I' の場合、 <i>H</i> の Schur ベクトルが計算される (配列 <i>z</i> は、このルーチンによって初期化される)。 <i>compz</i> ='V' の場合、 <i>A</i> の Schur ベクトルが計算される (呼び出し時 ...、配列 <i>z</i> には、あらかじめ行列 <i>Q</i> を格納しておかなければならない)。
<i>n</i>	INTEGER。行列 <i>H</i> の次数 ( $n \geq 0$ )。
<i>ilo</i> , <i>ihi</i>	INTEGER。 <i>A</i> の平衡化を ?gebal によって実行した場合は、 <i>ilo</i> と <i>ihi</i> に、?gebal から返された値を設定しなければならない。そうでない場合は、 <i>ilo</i> に 1 を、 <i>ihi</i> に <i>n</i> を設定しなければならない。
<i>h</i> , <i>z</i> , <i>work</i>	REAL (shseqr の場合) DOUBLE PRECISION (dhseqr の場合) COMPLEX (chseqr の場合) DOUBLE COMPLEX (zhseqr の場合)。 配列: <i>h</i> ( <i>ldh</i> ,*) $n \times n$ の上 Hessenberg 行列 <i>H</i> 。 <i>h</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。  <i>z</i> ( <i>ldz</i> ,*) <i>compz</i> ='V' の場合、 <i>z</i> には、Hessenberg 形式に縮退させた行列 <i>Q</i> を設定しなければならない。 <i>compz</i> ='I' の場合、 <i>z</i> を設定する必要はない。 <i>compz</i> ='N' の場合、 <i>z</i> は参照されない。 <i>z</i> の第 2 次元は、 $\max(1, n)$ 以上 ( <i>compz</i> ='V' または 'I' の場合) または 1 以上 ( <i>compz</i> ='N' の場合) でなければならない。  <i>work</i> ( <i>lwork</i> ) は、ワークスペース配列である。 <i>work</i> の次元は、 $\max(1, n)$ 以上でなければならない。
<i>ldh</i>	INTEGER。 <i>h</i> の第 1 次元。 $\max(1, n)$ 以上。

<i>ldz</i>	INTEGER。 <i>z</i> の第 1 次元。 <i>compz</i> = 'N' の場合、 $ldz \geq 1$ 。 <i>compz</i> = 'V' または 'I' の場合、 $ldz \geq \max(1, n)$ 。
<i>lwork</i>	INTEGER。 配列 <i>work</i> の次元。 $lwork \geq \max(1, n)$ <i>lwork</i> = -1 の場合、ワークスペースのクエリとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリとして返す。xerbla は <i>lwork</i> に関するエラー・メッセージを生成しない。

## 出力パラメータ

<i>w</i>	COMPLEX (chseqr の場合 ) DOUBLE COMPLEX (zhseqr の場合 )。 配列、次元は $\max(1, n)$ 以上。 <i>info</i> > 0 でない場合、計算で求めた固有値が格納される。一連の固有値は、Schur 形式 <i>T</i> ( 求めるように指定した場合 ) の対角成分と同じ順序で格納される。
<i>wr</i> , <i>wi</i>	REAL (shseqr の場合 ) DOUBLE PRECISION (dhseqr の場合 ) 配列、次元はそれぞれ $\max(1, n)$ 以上。 <i>info</i> > 0 でない場合、計算で求めた固有値の実数部と虚数部がそれぞれ格納される。固有値の複素共役ペアは、虚数部が正である固有値に続いて格納される。一連の固有値は、Schur 形式 <i>T</i> ( 求めるように指定した場合 ) の対角成分と同じ順序で格納される。
<i>z</i>	<i>compz</i> = 'V' または 'I' の場合、 <i>info</i> > 0 でない限り、Schur ベクトルを求めるように指定したときには、Schur ベクトルのユニタリ ( 直交 ) 行列が <i>z</i> に格納される。 <i>compz</i> = 'N' の場合、 <i>z</i> は参照されない。
<i>work</i> (1)	終了時に、 <i>info</i> = 0 の場合、 <i>work</i> (1) に最適な <i>lwork</i> 値が返される。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正であったことを示す。 <i>info</i> > 0 の場合、このアルゴリズムで 30 ( <i>ihi</i> - <i>ilo</i> + 1) 回の処理を繰り返した結果、すべての固有値を見つけられなかった。 <i>info</i> = <i>i</i> の場合、見つかった固有値の実数部と虚数部が、 <i>wr</i> と <i>wi</i> の 1, 2, ..., <i>ilo</i> - 1 および <i>i</i> + 1, <i>i</i> + 2, ..., <i>n</i> の成分に格納される。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `hseqr` のインターフェイスの詳細を以下に示す。

<code>h</code>	サイズ $(n, n)$ の行列 $H$ を格納する。
<code>wr</code>	長さ $(n)$ のベクトルを格納する。実数型でのみ使用される。
<code>wi</code>	長さ $(n)$ のベクトルを格納する。実数型でのみ使用される。
<code>w</code>	長さ $(n)$ のベクトルを格納する。複素数型でのみ使用される。
<code>z</code>	サイズ $(n, n)$ の行列 $Z$ を格納する。
<code>job</code>	'E' または 'S' でなければならない。デフォルト値は 'E'。
<code>compz</code>	省略された場合、この引数は、引数 $z$ の存在に基づいて以下のように復元される。 <code>compz = 'I'</code> ( $z$ が存在する場合)、 <code>compz = 'N'</code> ( $z$ が省略された場合)。 存在する場合、 <code>compz</code> は 'I' または 'V' と等しくなければならず、引数 $z$ も存在しなければならない。 <code>compz</code> が存在し、 $z$ が省略された場合、エラー条件がセットされる。

## アプリケーション・ノート

計算で求めた Schur 因子分解は、近似行列  $H + E$  の正確な因子分解である。ここで、 $\|E\|_2 < O(\epsilon) \|H\|_2/s_1$ 、 $\epsilon$  はマシン精度である。

$\lambda_i$  が正確な固有値で、 $\mu_i$  が対応する計算で求めた値の場合、 $|\lambda_i - \mu_i| \leq c(n)\epsilon \|H\|_2/s_1$  になる。 $c(n)$  は、 $n$  の漸増する関数である。 $s_1$  は、 $\lambda_i$  の条件数の逆数である。条件数  $s_1$  は、[?trsna](#) を呼び出して、求められる。

浮動小数演算の総数は、このアルゴリズムの収束の速さによって決まり、通常は、次の数値になる。

固有値のみを計算する場合：	$7n^3$ (実数型の場合) $25n^3$ (複素数型の場合)
Schur 形式を計算する場合：	$10n^3$ (実数型の場合) $35n^3$ (複素数型の場合)
Schur の因子分解をすべて計算する場合：	$20n^3$ (実数型の場合) $70n^3$ (複素数型の場合)



## ?hsein

指定された固有値に対応する上 *Hessenberg* 行列の固有ベクトルを選択して計算する。

### 構文

#### Fortran 77:

```
call shsein(job, eigsrc, initv, select, n, h, ldh, wr, wi, vl,
            ldvl, vr, ldvr, mm, m, work, ifaill, ifailr, info)
call dhsein(job, eigsrc, initv, select, n, h, ldh, wr, wi, vl,
            ldvl, vr, ldvr, mm, m, work, ifaill, ifailr, info)
call chsein(job, eigsrc, initv, select, n, h, ldh, w, vl,
            ldvl, vr, ldvr, mm, m, work, rwork, ifaill, ifailr, info)
call zhsein(job, eigsrc, initv, select, n, h, ldh, w, vl,
            ldvl, vr, ldvr, mm, m, work, rwork, ifaill, ifailr, info)
```

#### Fortran 95:

```
call hsein(h, wr, wi, select [,vl] [,vr] [,ifaill] [,ifailr] [,initv] [,eigsrc]
[,m] [,info])
call hsein(h, w, select [,vl] [,vr] [,ifaill] [,ifailr] [,initv] [,eigsrc] [,m]
[,info])
```

### 説明

このルーチンは、選択された固有値に対応する、上 *Hessenberg* 行列  $H$  の左または右の固有ベクトルを計算する。

右の固有ベクトル  $x$  と左の固有ベクトル  $y$  (固有値  $\lambda$  に対応) は、 $Hx = \lambda x$  と  $y^H H = \lambda y^H$  (または  $H^H y = \lambda^* y$ ) として定義される。  
 $\lambda^*$  は、 $\lambda$  の共役を表す。

一連の固有ベクトルは、反転を繰り返すことによって計算される。そして、実数型の固有ベクトル  $x$  の場合は  $\max |x_i| = 1$  で、複素数型の固有ベクトルの場合は  $\max(|\operatorname{Re} x_i| + |\operatorname{Im} x_i|) = 1$  になるようにスケールリングされる。

一般行列  $A$  を上 *Hessenberg* 形式に縮退させて  $H$  を求めた場合は、[?ormhr](#) または [?unmhr](#) によって、 $H$  の固有ベクトルを  $A$  の固有ベクトルに変換できる。

## 入力パラメータ

<i>job</i>	CHARACTER*1。'R'、'L'、または 'B' でなければならない。 <i>job</i> ='R' の場合、右の固有ベクトルだけが計算される。 <i>job</i> ='L' の場合、左の固有ベクトルだけが計算される。 <i>job</i> ='B' の場合、すべての固有ベクトルが計算される。
<i>eigsrc</i>	CHARACTER*1。'Q' または 'N' でなければならない。 <i>eigsrc</i> ='Q' の場合、 <a href="#">zhseqr</a> を使って、 <i>H</i> の固有値が求められている。そのため、 <i>H</i> の劣対角成分の中にゼロのものがある (つまり、ブロック三角である) 場合、 <i>j</i> 番目の固有値を <i>j</i> 番目の行と列が格納されているブロックの固有値とみなせる。このルーチンでは、この特性により、1 つの対角ブロックに関してだけ反転の繰り返し処理を実行できる。 <i>eigsrc</i> ='N' の場合、そのような仮定を行わず、このルーチンでは行列全体を使って反転の繰り返し処理を実行する。
<i>initv</i>	CHARACTER*1。'N' または 'U' でなければならない。 <i>initv</i> ='N' の場合、選択した固有ベクトルに対して初期の見積もりを指定しない。 <i>initv</i> ='U' の場合、選択した固有ベクトルに対して初期の見積もりを <i>vl</i> や <i>vr</i> で指定する。
<i>select</i>	LOGICAL。 配列、次元は $\max(1, n)$ 以上。 計算の対象となる固有ベクトルを指定する。 <b>実数型の場合:</b> 実数の固有値 $w_r(j)$ に対応する実数型の固有ベクトルを求めるには、 <i>select(j)</i> に .TRUE. を設定する。 複素数の固有値 ( $w_r(j), w_i(j)$ ) (複素共役は ( $w_r(j+1), w_i(j+1)$ )) に対応する複素数型の固有ベクトルを選択するには、 <i>select(j)</i> や <i>select(j+1)</i> に .TRUE. を設定する。ペアの最初の固有値に対応する固有ベクトルが計算される。 <b>複素数型の場合:</b> 固有値 $w(j)$ に対応する固有ベクトルを選択するには、 <i>select(j)</i> に .TRUE. を設定する。
<i>n</i>	INTEGER。行列 <i>H</i> の次数 ( $n \geq 0$ )。
<i>h, vl, vr, work</i>	REAL (shsein の場合) DOUBLE PRECISION (dhsein の場合) COMPLEX (chsein の場合) DOUBLE COMPLEX (zhsein の場合)。

配列：

$h(ldh,*)$   $n \times n$  の上 Hessenberg 行列  $H$ 。  
 $h$  の第 2 次元は、 $\max(1, n)$  以上でなければならない。

$v1(ldv1,*)$

$initv='V'$  かつ  $job='L'$  または  $'B'$  の場合、左固有ベクトルの反転の繰り返し処理用の開始ベクトルを  $v1$  に格納しなければならない。それぞれの開始ベクトルは、対応する固有ベクトルを格納する際に使うのと同じ列 (1 つまたは複数個) に格納しなければならない。

$initv='N'$  の場合、 $v1$  を設定する必要はない。

$v1$  の第 2 次元は、 $\max(1, mm)$  以上 ( $job='L'$  または  $'B'$  の場合) または 1 以上 ( $job='R'$  の場合) でなければならない。

$job='R'$  の場合、配列  $v1$  は参照されない。

$vr(ldvr,*)$

$initv='V'$  かつ  $job='R'$  または  $'B'$  の場合、右固有ベクトルの反転の繰り返し処理用の開始ベクトルを  $vr$  に格納しなければならない。それぞれの開始ベクトルは、対応する固有ベクトルを格納する際に使うのと同じ列 (1 つまたは複数個) に格納しなければならない。

$initv='N'$  の場合、 $vr$  を設定する必要はない。

$vr$  の第 2 次元は、 $\max(1, mm)$  以上 ( $job='R'$  または  $'B'$  の場合) または 1 以上 ( $job='L'$  の場合) でなければならない。

$job='L'$  の場合、配列  $vr$  は参照されない。

$work(*)$  は、ワークスペース配列である。

次元は  $\max(1, n*(n+2))$  以上 (実数型の場合) または  $\max(1, n*n)$  以上 (複素数型の場合)。

$ldh$  INTEGER。  $h$  の第 1 次元。  $\max(1, n)$  以上。

$w$  COMPLEX (chsein の場合)  
 DOUBLE COMPLEX (zhsein の場合)。  
 配列、次元は  $\max(1, n)$  以上。  
 行列  $H$  の固有値を格納する。  
 $eigsrc='Q'$  の場合、この配列は ?hseqr から返されたものと同じでなければならない。

$wr, wi$  REAL (shsein の場合)  
 DOUBLE PRECISION (dhsein の場合)  
 配列、次元はそれぞれ  $\max(1, n)$  以上。  
 行列  $H$  の固有値の実数部と虚数部をそれぞれ格納する。複素共役ペアの値は、この配列の連続する成分に格納しなければならない。  
 $eigsrc='Q'$  の場合、この配列は、?hseqr から返されたものと同じでなければならない。

<i>ldvl</i>	INTEGER。 <i>vl</i> の第 1 次元。 <i>job</i> ='L' または 'B' の場合、 $ldvl \geq \max(1, n)$ 。 <i>job</i> ='R' の場合、 $ldvl \geq 1$ 。
<i>ldvr</i>	INTEGER。 <i>vr</i> の第 1 次元。 <i>job</i> ='R' または 'B' の場合、 $ldvr \geq \max(1, n)$ 。 <i>job</i> ='L' の場合、 $ldvr \geq 1$ 。
<i>mm</i>	INTEGER。 <i>vl</i> や <i>vr</i> の列数。 <i>m</i> (実際に必要な列数。下記の出力パラメータを参照) 以上でなければならない。 実数型の場合、選択された実数の固有値ごとに 1 を、選択された複素数の固有値ごとに 2 をカウントすれば、 <i>m</i> が得られる ( <i>select</i> を参照)。 複素数型の場合、選択された固有ベクトルの個数が <i>m</i> になる ( <i>select</i> を参照)。次の制約がある。 $0 \leq mm \leq n$ 。
<i>rwork</i>	REAL ( <i>chsein</i> の場合) DOUBLE PRECISION ( <i>zhsein</i> の場合)。 配列、次元は $\max(1, n)$ 以上。

## 出力パラメータ

<i>select</i>	実数型の場合だけ、上書きされる。上記のように複素数の固有値が選択された場合は、 <i>select(j)</i> に .TRUE. が、 <i>select(j+1)</i> に .FALSE. が設定される。
<i>w</i>	<i>w</i> の一部の成分の実数部が変動する場合がある。これは、独立した固有ベクトルを検索する際に、近似した固有値がわずかに摂動するからである。
<i>wr</i>	<i>w</i> の一部の成分が変動する場合がある。これは、独立した固有ベクトルを検索する際に、近似した固有値がわずかに摂動するからである。
<i>vl, vr</i>	<i>job</i> ='L' または 'B' の場合、 <i>vl</i> には、( <i>select</i> の値に従って) 計算で求めた左の固有ベクトルが格納される。 <i>job</i> ='R' または 'B' の場合、 <i>vr</i> には、( <i>select</i> の値に従って) 計算で求めた右の固有ベクトルが格納される。  一連の固有ベクトルは、対応する固有値と同じ順序で、この配列の列に連続して格納される。 実数型の場合: 選択された実数の固有値に対応する実数型の固有ベクトルによって列を 1 つ占める。選択された固有値に対応する複素数型の固有ベクトルによって列を 2 つ占める。最初の列には実数部が、2 番目の列には虚数部が格納される。

<i>m</i>	INTEGER。実数型の場合: 選択された固有ベクトルの格納に必要な <i>v1</i> や <i>vr</i> の列数。 複素数型の場合: 選択した固有ベクトルの個数。
<i>ifail1, ifailr</i>	INTEGER。 配列、次元はそれぞれ $\max(1, mm)$ 以上。 <i>ifail1</i> ( <i>i</i> ) = 0 ( <i>v1</i> の <i>i</i> 番目の列が収束した場合)。 <i>ifail1</i> ( <i>i</i> ) = <i>j</i> > 0 ( <i>v1</i> の <i>i</i> 番目の列に格納されている固有ベクトル ( <i>j</i> 番目の固有値に対応) が収束しなかった場合)。 <i>ifailr</i> ( <i>i</i> ) = 0 ( <i>vr</i> の <i>i</i> 番目の列が収束した場合)。 <i>ifailr</i> ( <i>i</i> ) = <i>j</i> > 0 ( <i>vr</i> の <i>i</i> 番目の列に格納されている固有ベクトル ( <i>j</i> 番目の固有値に対応) が収束しなかった場合)。 実数型の場合: <i>v1</i> の <i>i</i> 番目と ( <i>i</i> +1) 番目の列に選択した複素数型の固有ベクトルが格納されている場合は、 <i>ifail1</i> ( <i>i</i> ) と <i>ifail1</i> ( <i>i</i> +1) に同じ値が設定される。 <i>vr</i> と <i>ifailr</i> にも、これと同様の規則が適用される。  <i>job</i> = 'R' の場合、配列 <i>ifail1</i> は参照されない。 <i>job</i> = 'L' の場合、配列 <i>ifailr</i> は参照されない。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正であったことを示す。 <i>info</i> > 0 の場合、 <i>i</i> 個の固有ベクトル (上記のパラメータ <i>ifail1</i> や <i>ifailr</i> で指定) が、収束していない。 <i>v1</i> や <i>vr</i> の対応する列に、有用な情報が格納されていない。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *hsein* のインターフェイスの詳細を以下に示す。

<i>h</i>	サイズ ( <i>n</i> , <i>n</i> ) の行列 <i>H</i> を格納する。
<i>wr</i>	長さ ( <i>n</i> ) のベクトルを格納する。実数型でのみ使用される。
<i>wi</i>	長さ ( <i>n</i> ) のベクトルを格納する。実数型でのみ使用される。
<i>w</i>	長さ ( <i>n</i> ) のベクトルを格納する。複素数型でのみ使用される。
<i>select</i>	長さ ( <i>n</i> ) のベクトルを格納する。

<code>vl</code>	サイズ $(n, mm)$ の行列 <code>VL</code> を格納する。
<code>vr</code>	サイズ $(n, mm)$ の行列 <code>VR</code> を格納する。
<code>ifaill</code>	長さ $(mm)$ のベクトルを格納する。 <code>ifaill</code> が存在し、 <code>vl</code> が省略された場合、エラー条件がセットされる。
<code>ifailr</code>	長さ $(mm)$ のベクトルを格納する。 <code>ifailr</code> が存在し、 <code>vr</code> が省略された場合、エラー条件がセットされる。
<code>initv</code>	'N' または 'U' でなければならない。デフォルト値は 'N'。
<code>eigsrc</code>	'N' または 'Q' でなければならない。デフォルト値は 'N'。
<code>job</code>	引数 <code>vl</code> および <code>vr</code> の存在に基づいて以下のように復元される。 <code>job</code> = 'B' ( <code>vl</code> と <code>vr</code> の両方が存在する場合)、 <code>job</code> = 'L' ( <code>vl</code> が存在し、 <code>vr</code> が省略された場合)、 <code>job</code> = 'R' ( <code>vl</code> が省略され、 <code>vr</code> が存在する場合)。 <code>vl</code> と <code>vr</code> の両方が省略された場合、エラー条件がセットされる。

### アプリケーション・ノート

計算で求めた右固有ベクトル  $x_i$  はそれぞれ、近似行列  $A + E_i$  の正確な固有ベクトル ( $\|E_i\| < O(\epsilon)\|A\|$ ) である。そのため、誤差は小さいものになる ( $\|Ax_i - \lambda_i x_i\| = O(\epsilon)\|A\|$ )。

ただし、近似したあるいは一致した固有値に対応する固有ベクトルが、関連する部分空間に正確に張られていない場合がある。

計算で求めた左固有ベクトルについても、これと同様の注意が適用される。

---

## ?trevc

?hseqr で求めた上( 準) 三角行列の固有ベクトルを選択して計算する。

---

### 構文

#### Fortran 77:

```
call strevc(side, howmny, select, n, t, ldt, vl, ldvl, vr, ldvr,  
            mm, m, work, info)  
call dtrevc(side, howmny, select, n, t, ldt, vl, ldvl, vr, ldvr,  
            mm, m, work, info)
```

```
call ctrevc(side, howmny, select, n, t, ldt, vl, ldvl, vr, ldvr,
            mm, m, work, rwork, info)
call ztrevc(side, howmny, select, n, t, ldt, vl, ldvl, vr, ldvr,
            mm, m, work, rwork, info)
```

**Fortran 95:**

```
call trevc(t [,howmny] [,select] [,vl] [,vr] [,m] [,info])
```

**説明**

このルーチンは、上三角行列  $T$  (実数型の場合、上準三角行列  $T$ ) の左あるいは右の一部またはすべての固有ベクトルを計算する。このタイプの行列は、一般行列の Schur 因子分解 ( $A = QTQ^H$ ) で生成される ([?hsegr](#) で計算される)。

$T$  の固有値  $w$  に対応する右固有ベクトル  $x$  と左固有ベクトル  $y$  は、次のように表される。  
 $Tx = wx$  ,  $y^H T = wy^H$   
 $y^H$  は  $y$  の共役転置を表す。

固有値は、このルーチンに入力されず、 $T$  の対角ブロックから直接読み込まれる。

このルーチンは、 $T$  の右と左固有ベクトルが入った行列  $X$  と  $Y$  を返すか、積  $QX$  と積  $QY$  を返す ( $Q$  は入力行列)。

$Q$  が、行列  $A$  を Schur 形式  $T$  に縮退する直交 / ユニタリ行列の場合、 $QX$  と  $QY$  は、 $A$  の右と左の固有ベクトルからなる行列である。

**入力パラメータ**

<i>side</i>	CHARACTER*1。'R'、'L'、または 'B' でなければならない。 <i>side</i> = 'R' の場合、右の固有ベクトルだけが計算される。 <i>side</i> = 'L' の場合、左の固有ベクトルだけが計算される。 <i>side</i> = 'B' の場合、すべての固有ベクトルが計算される。
<i>howmny</i>	CHARACTER*1。'A'、'B'、または 'S' のいずれかでなければならない。 <i>howmny</i> = 'A' の場合、( <i>side</i> の値に従って) すべての固有ベクトルが計算される。 <i>howmny</i> = 'B' の場合、( <i>side</i> の値に従って) すべての固有ベクトルが計算され、 <i>vl</i> と <i>vr</i> で与えられる行列によって逆変換される。 <i>howmny</i> = 'S' の場合、( <i>side</i> と <i>select</i> の値に従って) 選択した固有ベクトルが計算される。
<i>select</i>	LOGICAL。 配列、次元は $\max(1, n)$ 以上。 <i>howmny</i> = 'S' の場合、 <i>select</i> は計算する固有ベクトルを指定する。

*howmny*='A' または 'B' の場合、*select* は参照されない。

実数型の場合:

$\omega_j$  が実固有値の場合、*select*(*j*) が .TRUE. であれば、それに対応する実固有ベクトルが計算される。

$\omega_j$  と  $\omega_{j+1}$  が複素固有値の実部と虚部の場合、*select*(*j*) または *select*(*j*+1) が .TRUE. であれば、それに対応する複素固有ベクトルが計算され、終了時に、*select*(*j*) は .TRUE.、*select*(*j*+1) は .FALSE. に設定される。

複素数型の場合:

*select*(*j*) が .TRUE. であれば、*j* 番目の固有値に対応する固有ベクトルが計算される。

*n* INTEGER。行列 *T* の次数 ( $n \geq 0$ )。

*t, vl, vr, work* REAL (*strevc* の場合 )  
DOUBLE PRECISION (*dtrevc* の場合 )  
COMPLEX (*ctrevc* の場合 )  
DOUBLE COMPLEX (*ztrevc* の場合 )。

配列:

*t*(*ldt*,\*) には、 $n \times n$  の行列 *T* を標準の Schur 形式で格納する。  
*t* の第 2 次元は、 $\max(1, n)$  以上でなければならない。

*vl*(*ldvl*,\*)

*howmny*='B' かつ *side*='L' または 'B' の場合、*vl* には、 $n \times n$  の行列 *Q* (通常は、?hseqr から返された Schur ベクトルから成る行列) を格納しなければならない。

*howmny*='A' または 'S' の場合、*vl* を設定する必要はない。

*vl* の第 2 次元は、 $\max(1, mm)$  以上 (*side*='L' または 'B' の場合) または 1 以上 (*side*='R' の場合) でなければならない。

*side*='R' の場合、配列 *vl* は参照されない。

*vr*(*ldvr*,\*)

*howmny*='B' かつ *side*='R' または 'B' の場合、*vr* には、 $n \times n$  の行列 *Q* (通常は、?hseqr から返された Schur ベクトルから成る行列) を格納しなければならない。

*howmny*='A' または 'S' の場合、*vr* を設定する必要はない。

*vr* の第 2 次元は、 $\max(1, mm)$  以上 (*side*='R' または 'B' の場合) または 1 以上 (*side*='L' の場合) でなければならない。

*side*='L' の場合、配列 *vr* は参照されない。

*work*(\*) は、ワークスペース配列である。

次元は、 $\max(1, 3*n)$  以上 (実数型の場合) または  $\max(1, 2*n)$  以上 (複素数型の場合)。



<i>ldt</i>	INTEGER。 <i>t</i> の第 1 次元。 $\max(1, n)$ 以上。
<i>ldvl</i>	INTEGER。 <i>vl</i> の第 1 次元。 <i>side</i> = 'L' または 'B' の場合、 $ldvl \geq \max(1, n)$ 。 <i>side</i> = 'R' の場合、 $ldvl \geq 1$ 。
<i>ldvr</i>	INTEGER。 <i>vr</i> の第 1 次元。 <i>side</i> = 'R' または 'B' の場合、 $ldvr \geq \max(1, n)$ 。 <i>side</i> = 'L' の場合、 $ldvr \geq 1$ 。
<i>mm</i>	INTEGER。 配列 <i>vl</i> や <i>vr</i> の列数。 <i>m</i> ( 必要な列の正確な個数 ) 以上でなければならない。 <i>howmny</i> = 'A' または 'B' の場合、 $m = n$ 。 <i>howmny</i> = 'S' の場合： <i>実数型の場合</i> 、 選択された実数の固有ベクトルごとに 1 を、 選択された複素数の固有ベクトルごとに 2 をカウントすれば、 <i>m</i> が得られる。 <i>複素数型の場合</i> 、 選択された固有ベクトルの個数が <i>m</i> になる ( <i>select</i> を参照 )。 次の制約がある。 $0 \leq m \leq n$ 。
<i>rwork</i>	REAL ( <i>ctrevc</i> の場合 ) DOUBLE PRECISION ( <i>ztrevc</i> の場合 )。 ワークスペース配列、 次元は $\max(1, n)$ 以上。

### 出力パラメータ

<i>select</i>	上記のように実行列の複素数型の固有ベクトルを選択した場合は、 <i>select(j)</i> に .TRUE. が、 <i>select(j+1)</i> に .FALSE. が設定される。
<i>vl, vr</i>	<i>side</i> = 'L' または 'B' の場合、 <i>vl</i> に、 ( <i>howmny</i> と <i>select</i> の値に従って ) 計算で求めた左固有ベクトルが格納される。 <i>side</i> = 'R' または 'B' の場合、 <i>vr</i> に、 ( <i>howmny</i> と <i>select</i> の値に従って ) 計算で求めた右固有ベクトルが格納される。  一連の固有ベクトルは、 対応する固有値と同じ順序で、 この配列の列に連続して格納される。 <i>実数型の場合</i> ： 実数の固有値のそれぞれに実数の固有ベクトルが 1 つ対応し、 列を 1 つ占める。 固有値の複素共役ペアのそれぞれに複素数の固有ベクトルが 1 つ対応し、 列を 2 つ占める。 最初の列には実数部、 2 番目の列には虚数部が格納される。
<i>m</i>	INTEGER。 <i>複素数型の場合</i> ： 選択した固有ベクトルの個数。 <i>howmny</i> = 'A' または 'B' の場合、 <i>m</i> には <i>n</i> が設定される。

実数型の場合: 選択した固有ベクトルの格納に実際に使用される  $v_l$  や  $vr$  の列数。

$howmny = 'A'$  または  $'B'$  の場合、 $m$  には  $n$  が設定される。

$info$  INTEGER。  $info = 0$  の場合、実行は正常に終了したことを示す。  
 $info = -i$  の場合、 $i$  番目のパラメータの値が不正であったことを示す。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `trevc` のインターフェイスの詳細を以下に示す。

$t$	サイズ $(n, n)$ の行列 $T$ を格納する。
$select$	長さ $(n)$ のベクトルを格納する。
$v_l$	サイズ $(n, mm)$ の行列 $VL$ を格納する。
$vr$	サイズ $(n, mm)$ の行列 $VR$ を格納する。
$side$	省略された場合、この引数は、引数 $v_l$ と $vr$ の存在に基づいて以下のよう復元される。 $side = 'B'$ ( $v_l$ と $vr$ の両方が存在する場合)、 $side = 'L'$ ( $vr$ が省略された場合)、 $side = 'R'$ ( $v_l$ が省略された場合)。 $v_l$ と $vr$ の両方が省略された場合、エラー条件がセットされる。
$howmny$	省略された場合、この引数は、引数 $select$ の存在に基づいて以下のよう復元される。 $howmny = 'V'$ ( $select$ が存在する場合)、 $howmny = 'N'$ ( $select$ が省略された場合)。 存在する場合、 $vect = 'V'$ または $'U'$ で、引数 $select$ も存在しなければならない。 $select$ と $howmny$ の両方が省略された場合、エラー条件がセットされる。

## アプリケーション・ノート

$x_i$  が正確な固有ベクトルで、 $y_i$  が対応する計算で求めた固有ベクトルの場合、それらの間の角  $\theta(y_i, x_i)$  は次のようになる。 $\theta(y_i, x_i) \leq (c(n)\epsilon\|T\|_2)/sep_i$  ( $sep_i$  は  $x_i$  の条件数の逆数)。条件数  $sep_i$  は、`?trsna` を呼び出すことによって求められる。

## ?trsna

上(準)三角行列の指定された固有値および  
右固有ベクトルに関して条件数を見積もる。

### 構文

#### Fortran 77:

```
call strsna(job, howmny, select, n, t, ldt, vl, ldvl, vr, ldvr,
  s, sep, mm, m, work, ldwork, iwork, info)
call dtrsna(job, howmny, select, n, t, ldt, vl, ldvl, vr, ldvr,
  s, sep, mm, m, work, ldwork, iwork, info)
call ctrsna(job, howmny, select, n, t, ldt, vl, ldvl, vr, ldvr,
  s, sep, mm, m, work, ldwork, rwork, info)
call ztrsna(job, howmny, select, n, t, ldt, vl, ldvl, vr, ldvr,
  s, sep, mm, m, work, ldwork, rwork, info)
```

#### Fortran 95:

```
call trsna(t [,s] [,sep] [,vl] [,vr] [,select] [,m] [,info])
```

### 説明

このルーチンは、上三角行列  $T$  (実数型の場合、標準形の Schur 形式の上準三角行列  $T$ ) の指定された固有値や右固有ベクトルに関して、条件数を見積もる。これらは、元の行列  $A = ZTZ^H$  ( $Z$  はユニタリ行列。ただし、実数型の場合は直交行列) の固有値と右固有ベクトルの条件数と同じである。ここから  $T$  を導き出すことができる。

このルーチンでは、固有値  $\lambda_i$  の条件数の逆数を  $s_i = |v^H u| / (\|u\|_E \|v\|_E)$  として計算する。 $u$  と  $v$  はそれぞれ、 $\lambda_i$  に対応する  $T$  の右と左の固有ベクトルである。この条件数の逆数は、常に、ゼロ (悪い条件の場合) と 1 (良い条件の場合) の間になる。

計算で求めた固有値  $\lambda_i$  のおおよその誤差は、 $\varepsilon \|T\| / s_i$  ( $\varepsilon$  はマシン精度) と見積もれる。

$\lambda_i$  に対応する右固有ベクトルの条件数の逆数を見積もるために、このルーチンではまず [?trexc](#) を呼び出して一連の固有値の順序を変更し、 $\lambda_i$  が先頭の位置にする。

$$T = Q \begin{bmatrix} \lambda_i & C^H \\ 0 & T_{22} \end{bmatrix} Q^H$$

次に、この固有ベクトルの条件数の逆数を、 $sep_i$  (行列  $T_{22}$  の最小の特異値  $-\lambda_i I$ ) として見積もる。この値は、ゼロ (悪い条件の場合) から非常に大きな値 (良い条件の場合) の間になる。

$\lambda_i$  に対応して計算で求めた右固有ベクトル  $u$  のおおよその誤差は、 $\varepsilon \|T\|/sep_i$  と見積もれる。

## 入力パラメータ

<i>job</i>	<p>CHARACTER*1。'E'、'V'、または 'B' のいずれかでなければならない。</p> <p><i>job</i>='E' の場合、固有値だけの条件数が計算される。</p> <p><i>job</i>='V' の場合、固有ベクトルだけの条件数が計算される。</p> <p><i>job</i>='B' の場合、固有値と固有ベクトルの両方の条件数が計算される。</p>
<i>howmny</i>	<p>CHARACTER*1。'A' または 'S' でなければならない。</p> <p><i>howmny</i>='A' の場合、すべての固有ペアの条件数が計算される。</p> <p><i>howmny</i>='S' の場合、選択した固有ペア (<i>select</i> で指定) の条件数が計算される。</p>
<i>select</i>	<p>LOGICAL。</p> <p>配列、次元は <math>\max(1, n)</math> 以上 (<i>howmny</i>='S' の場合) または 1 以上 (それ以外の場合)。</p> <p><i>howmny</i>='S' の場合、条件数を計算する固有ペアを指定する。</p> <p><b>実数型の場合:</b></p> <p>実数の固有値 <math>\lambda_j</math> に対応する固有ペアの条件数を選択するには、<i>select</i>(<i>j</i>) に .TRUE. を設定しなければならない。</p> <p>固有値 <math>\lambda_j</math> と <math>\lambda_{j+1}</math> の複素共役ペアに対応する固有ペアの条件数を選択するには、<i>select</i>(<i>j</i>) や <i>select</i>(<i>j</i>+1) に .TRUE. を設定しなければならない。</p> <p><b>複素数の場合:</b></p> <p>固有値 <math>\lambda_j</math> に対応する固有ペアの条件数を選択するには、<i>select</i>(<i>j</i>) に .TRUE. を設定しなければならない。 <i>howmny</i>='A' の場合、<i>select</i> は参照されない。</p>

$n$	INTEGER。行列 $T$ の次数 ( $n \geq 0$ )。
$t, v1, vr, work$	<p>REAL (strsna の場合 )  DOUBLE PRECISION (dtrsna の場合 )  COMPLEX (ctrsna の場合 )  DOUBLE COMPLEX (ztrsna の場合 )。</p> <p>配列：  <math>t(ldt, *)</math> には、<math>n \times n</math> の行列 <math>T</math> を格納する。  <math>t</math> の第 2 次元は、<math>\max(1, n)</math> 以上でなければならない。</p> <p><math>v1(ldv1, *)</math>  <math>job = 'E'</math> または <math>'B'</math> の場合、<math>v1</math> には、<i>howmny</i> と <i>select</i> で指定した固有ペアに対応する <math>T</math> ( または <math>Q</math> がユニタリ行列か直交行列のような任意の行列 <math>QTQ^H</math> ) の左固有ベクトルを格納しなければならない。この固有ベクトルは、<a href="#">?trevc</a> や <a href="#">?hsein</a> で返される通りに、<math>v1</math> の連続した列に格納されていなければならない。  <math>v1</math> の第 2 次元は、<math>\max(1, mm)</math> 以上 (<math>job = 'E'</math> または <math>'B'</math> の場合 ) または 1 以上 (<math>job = 'V'</math> の場合 ) でなければならない。  <math>job = 'V'</math> の場合、配列 <math>v1</math> は参照されない。</p> <p><math>vr(ldvr, *)</math>  <math>job = 'E'</math> または <math>'B'</math> の場合、<math>vr</math> には、<i>howmny</i> と <i>select</i> で指定した固有ペアに対応する <math>T</math> ( または <math>Q</math> がユニタリ行列か直交行列のような任意の行列 <math>QTQ^H</math> ) の右固有ベクトルを格納しなければならない。この固有ベクトルは、<a href="#">?trevc</a> や <a href="#">?hsein</a> で返される通りに、<math>vr</math> の連続した列に格納されていなければならない。  <math>vr</math> の第 2 次元は、<math>\max(1, mm)</math> 以上 (<math>job = 'E'</math> または <math>'B'</math> の場合 ) または 1 以上 (<math>job = 'V'</math> の場合 ) でなければならない。  <math>job = 'V'</math> の場合、配列 <math>vr</math> は参照されない。</p> <p><math>work(ldwork, *)</math> は、ワークスペース配列である。  <math>work</math> の第 2 次元は、<math>\max(1, n+1)</math> 以上 ( 複素数型の場合 ) または <math>\max(1, n+6)</math> 以上 ( 実数型の場合 ) (<math>job = 'V'</math> または <math>'B'</math> の場合 ) または 1 以上 (<math>job = 'E'</math> の場合 ) でなければならない。  <math>job = 'E'</math> の場合、配列 <math>work</math> は参照されない。</p>
$ldt$	INTEGER。 $t$ の第 1 次元。 $\max(1, n)$ 以上。
$ldv1$	<p>INTEGER。 <math>v1</math> の第 1 次元。  <math>job = 'E'</math> または <math>'B'</math> の場合、<math>ldv1 \geq \max(1, n)</math>。  <math>job = 'V'</math> の場合、<math>ldv1 \geq 1</math>。</p>

<i>ldvr</i>	INTEGER。 <i>vr</i> の第 1 次元。 <i>job</i> ='E' または 'B' の場合、 $ldvr \geq \max(1, n)$ 。 <i>job</i> ='R' の場合、 $ldvr \geq 1$ 。
<i>mm</i>	INTEGER。 配列 <i>s</i> と <i>sep</i> の成分の個数、または <i>v1</i> と <i>vr</i> の列数 ( 使う場合 )。 <i>m</i> ( 必要とする正確な個数 ) 以上でなければならない。 <i>howmny</i> ='A' の場合、 $m = n$ 。 <i>howmny</i> ='S' の場合、実数型の場合、選択された実数の固有値ごとに 1 を、固有値の選択された複素共役ペアごとに 2 をカウントすれば、 <i>m</i> が得られる。 複素数型の場合、選択された固有ペアの個数が <i>m</i> になる ( <i>select</i> を参照)。次の制約がある。 $0 \leq m \leq n$ 。
<i>ldwork</i>	INTEGER。 <i>work</i> の第 1 次元。 <i>job</i> ='V' または 'B' の場合、 $ldwork \geq \max(1, n)$ 。 <i>job</i> ='E' の場合、 $ldwork \geq 1$ 。
<i>rwork</i>	REAL ( <i>ctrсна</i> 、 <i>ztrsna</i> の場合)。 配列、次元は $\max(1, n)$ 以上。
<i>iwork</i>	INTEGER ( <i>strсна</i> 、 <i>dtrsna</i> の場合)。 配列、次元は $\max(1, n)$ 以上。

## 出力パラメータ

<i>s</i>	REAL ( 単精度の場合 ) DOUBLE PRECISION ( 倍精度の場合 )。 配列、次元は $\max(1, mm)$ 以上 ( <i>job</i> ='E' または 'B' の場合) または 1 以上 ( <i>job</i> ='V' の場合)。 <i>job</i> ='E' または 'B' の場合、選択した固有値の条件数の逆数が、この配列の連続する成分に格納される。そのため、 <i>s</i> ( <i>j</i> )、 <i>sep</i> ( <i>j</i> )、および <i>v1</i> と <i>vr</i> の <i>j</i> 番目の列はすべて、同じ固有ペアに対応している (ただし、選択しなかった固有ペアが存在する場合は、一般に、 <i>j</i> 番目の固有ペアにはならない)。実数型の場合: 固有値の複素共役ペアでは、 <i>S</i> の 2 つの連続する成分に同じ値が設定される。 <i>job</i> ='V' の場合、配列 <i>s</i> は参照されない。
<i>sep</i>	REAL ( 単精度の場合 ) DOUBLE PRECISION ( 倍精度の場合 )。 配列、次元は $\max(1, mm)$ 以上 ( <i>job</i> ='V' または 'B' の場合) または 1 以上 ( <i>job</i> ='E' の場合)。 <i>job</i> ='V' または 'B' の場合、選択した右固有ベクトルについての見積もりの条件数の逆数が、この配列の連続する成分に格納される。 実数型の場合: 複素数の固有ベクトルのときは、 <i>sep</i> の 2 つの連続す

る成分に同じ値が設定される。 $sep(j)$  を計算するために一連の固有値の順序を変更できない場合は、 $sep(j)$  にゼロが設定される。これは、真の値が非常に小さい場合にだけ発生する可能性がある。

$job = 'E'$  の場合、配列  $sep$  は参照されない。

$m$                     INTEGER。  
                       複素数型の場合：選択した固有ペアの個数。 $howmny = 'A'$  の場合、 $m$  は  $n$  に設定される。  
                       実数型の場合：条件数の見積もり値の格納に実際に使用される  $s$  や  $sep$  の成分の個数。 $howmny = 'A'$  の場合、 $m$  は  $n$  に設定される。

$info$                 INTEGER。  
                        $info = 0$  の場合、実行は正常に終了したことを示す。  
                        $info = -i$  の場合、 $i$  番目のパラメータの値が不正であったことを示す。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `trsna` のインターフェイスの詳細を以下に示す。

$t$                     サイズ  $(n, n)$  の行列  $T$  を格納する。

$s$                     長さ  $(mm)$  のベクトルを格納する。

$sep$                 長さ  $(mm)$  のベクトルを格納する。

$vl$                 サイズ  $(n, mm)$  の行列  $VL$  を格納する。

$vr$                 サイズ  $(n, mm)$  の行列  $VR$  を格納する。

$select$             長さ  $(n)$  のベクトルを格納する。

$job$                 引数  $s$  と  $sep$  の存在に基づいて以下のように復元される。  
                        $job = 'B'$  ( $s$  と  $sep$  の両方が存在する場合)、  
                        $job = 'E'$  ( $s$  が存在し、 $sep$  が省略された場合)、  
                        $job = 'V'$  ( $s$  が省略され、 $sep$  が存在する場合)。  
                        $s$  と  $sep$  の両方が省略された場合、エラー条件がセットされる。

$howmny$             引数  $select$  の存在に基づいて以下のように復元される。  
                        $howmny = 'S'$  ( $select$  が存在する場合)、  
                        $howmny = 'A'$  ( $select$  が省略された場合)。

引数  $s$ 、 $v1$ 、および  $vr$  は、すべて存在するか、すべて省略されなければならない。そうでない場合、エラー条件がセットされる。

### アプリケーション・ノート

計算で求めた値  $sep_i$  は、真の値を過大見積もりする場合もあるが、3 を超える係数になるのはほとんどない。

---

## ?trexc

一般行列の *Schur* 因子分解の順序を変更する。

---

### 構文

#### Fortran 77:

```
call strexc(compq, n, t, ldt, q, ldq, ifst, ilst, work, info)
call dtrexc(compq, n, t, ldt, q, ldq, ifst, ilst, work, info)
call ctrexc(compq, n, t, ldt, q, ldq, ifst, ilst, info)
call ztrexc(compq, n, t, ldt, q, ldq, ifst, ilst, info)
```

#### Fortran 95:

```
call trexc(t, ifst, ilst [,q] [,info])
```

### 説明

このルーチンは、一般行列  $A = QTQ^H$  の *Schur* 因子分解の順序を変更し、 $T$  の行インデックス  $ifst$  の対角成分または対角ブロックを行  $ilst$  に移動する。

順序を変更した後の *Schur* 形式  $S$  は、ユニタリ (実数型の場合は直交) 相似変換  $S = Z^H T Z$  によって計算される。また、*Schur* ベクトルの更新後の行列  $P$  を、 $P = QZ$  ( $A = PSP^H$ ) として計算できる。

### 入力パラメータ

<i>compq</i>	CHARACTER*1。'V' または 'N' でなければならない。 <i>compq</i> = 'V' の場合、 <i>Schur</i> ベクトル ( $Q$ ) が更新される。 <i>compq</i> = 'N' の場合、 <i>Schur</i> ベクトルは更新されない。
<i>n</i>	INTEGER。行列 $T$ の次数 ( $n \geq 0$ )。



$t, q$	<p>REAL (strexc の場合 )  DOUBLE PRECISION (dtrexc の場合 )  COMPLEX (ctrexc の場合 )  DOUBLE COMPLEX (ztrexc の場合 )。  配列：  <math>t(ldt, *)</math> には、<math>n \times n</math> の行列 <math>T</math> を格納する。  <math>t</math> の第 2 次元は、<math>\max(1, n)</math> 以上でなければならない。</p> <p><math>q(ldq, *)</math>  <math>compq = 'V'</math> の場合、<math>q</math> には <math>Q</math> (Schur ベクトル) を格納しなければならない。  <math>compq = 'N'</math> の場合、<math>q</math> は参照されない。</p> <p><math>q</math> の第 2 次元は、<math>\max(1, n)</math> 以上 (<math>compq = 'V'</math> の場合 ) または 1 以上 (<math>compq = 'N'</math> の場合 ) でなければならない。</p>
$ldt$	INTEGER。 $t$ の第 1 次元。 $\max(1, n)$ 以上。
$ldq$	INTEGER。 $q$ の第 1 次元。 $compq = 'N'$ の場合、 $ldq \geq 1$ 。 $compq = 'V'$ の場合、 $ldq \geq \max(1, n)$ 。
$ifst, ilst$	<p>INTEGER。 <math>1 \leq ifst \leq n; 1 \leq ilst \leq n</math>。  行列 <math>T</math> の対角成分 (実数型の場合は対角ブロック) の順序の変更を指定する。行インデックスが <math>ifst</math> の成分 (またはブロック) は、隣接する成分 (またはブロック) 間での一連の交換処理により、行 <math>ilst</math> に移される。</p>
$work$	<p>REAL (strexc の場合 )  DOUBLE PRECISION (dtrexc の場合 )。  配列、次元は <math>\max(1, n)</math> 以上。</p>

### 出力パラメータ

$t$	更新後の行列 $S$ によって上書きされる。
$q$	$compq = 'V'$ の場合、 $q$ に、Schur ベクトルで構成される更新後の行列が格納される。
$ifst, ilst$	<p>実数型の場合にのみ上書きされる。  呼び出し時に、<math>ifst</math> が <math>2 \times 2</math> のブロックの 2 行目を指している場合は、先頭行を指すように変更される。<math>ilst</math> は常に、最終位置 (入力値から <math>\pm 1</math> だけ変動している場合もある) のブロックの先頭行を指す。</p>

*info*                    INTEGER。  
*info* = 0 の場合、実行は正常に終了したことを示す。  
*info* = -*i* の場合、*i* 番目のパラメータの値が不正であったことを示す。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `trexc` のインターフェイスの詳細を以下に示す。

*t*                    サイズ (*n*,*n*) の行列 *T* を格納する。  
*q*                    サイズ (*n*,*n*) の行列 *Q* を格納する。  
*compq*              引数 *q* の存在に基づいて以下のように復元される。  
                     *compq* = 'V' (*q* が存在する場合)、  
                     *compq* = 'N' (*q* が省略された場合)。

## アプリケーション・ノート

計算で求めた行列 *S* は、行列 *T* + *E* の近似値である。ここで、 $\|E\|_2 = O(\epsilon) \|T\|_2$ 、 $\epsilon$  はマシン精度である。

2 × 2 の対角ブロックが順序変更に関与する場合は、一般に、その非対角成分が変更される。そのブロックの条件が非常に悪い場合を除き、そのブロックの対角成分と固有値は変更されない。条件が非常に悪い場合は、それらが明確に変更される場合もある。2 × 2 のブロックは、2 つの 1 × 1 のブロックに分割できる。つまり、複素数の固有値ペアを、純粋な実数にできる。

ただし、固有値の値が、順序の変更によって変更されたりはしない。

浮動小数演算のおおよその総数は、次のようになる。

実数型の場合：               $6n(\text{ifst-ilst})(\text{compq} = \text{'N'} \text{ の場合})$   
                                   $12n(\text{ifst-ilst})(\text{compq} = \text{'V'} \text{ の場合})$   
 複素数型の場合：             $20n(\text{ifst-ilst})(\text{compq} = \text{'N'} \text{ の場合})$   
                                   $40n(\text{ifst-ilst})(\text{compq} = \text{'V'} \text{ の場合})$

## ?trsen

行列の Schur 因子分解の順序を変更し、( オプションで) 固有値の選択した束についての条件数の逆数と不変部分空間を計算する。

### 構文

#### Fortran 77:

```
call strsen(job, compq, select, n, t, ldt, q, ldq, wr, wi, m, s,
            sep, work, lwork, iwork, liwork, info)
call dtrsen(job, compq, select, n, t, ldt, q, ldq, wr, wi, m, s,
            sep, work, lwork, iwork, liwork, info)
call ctrsen(job, compq, select, n, t, ldt, q, ldq, w, m, s,
            sep, work, lwork, info)
call ztrsen(job, compq, select, n, t, ldt, q, ldq, w, m, s,
            sep, work, lwork, info)
```

#### Fortran 95:

```
call trsen(t, select [,wr] [,wi] [,m] [,s] [,sep] [,q] [,info])
call trsen(t, select [,w] [,m] [,s] [,sep] [,q] [,info])
```

### 説明

このルーチンは、一般行列  $A = QTQ^H$  の Schur 因子分解の順序を変更して、固有値の選択した束が Schur 形式の主対角成分 (実数型の場合は対角ブロック) にくるようにする。順序を変更した後の Schur 形式  $R$  は、ユニタリ (直交) 相似変換  $R = Z^H T Z$  によって計算される。また、Schur ベクトルの更新後の行列  $P$  を、 $P = QZ$  ( $A = PRP^H$ ) として計算できる。

次のように仮定し、

$$R = \begin{bmatrix} T_{11} & T_{12} \\ 0 & T_{13} \end{bmatrix}$$

選択した固有値が、厳密な意味で  $m \times m$  の先頭の部分行列  $T_{11}$  の固有値であるとする。 $P$  を  $(Q_1 \ Q_2)$  のように対応させて分割し、 $Q_1$  が  $Q$  の先頭から  $m$  列を構成させる。これで、 $AQ_1 = Q_1 T_{11}$  になり、 $Q_1$  の  $m$  列は固有値の選択した束に対応する不変部分空間の直交基になる。

また、このルーチンでは、固有値の束と不変部分空間の平均の条件数の逆数を見積もれる。

## 入力パラメータ

<i>job</i>	<p>CHARACTER*1。'N'、'E'、'V'、または 'B' のいずれかでなければならない。</p> <p><i>job</i>='N' の場合、条件数は計算されない。</p> <p><i>job</i>='E' の場合、固有値の束の条件数だけが計算される。</p> <p><i>job</i>='V' の場合、不変部分空間の条件数だけが計算される。</p> <p><i>job</i>='B' の場合、束と不変部分空間の条件数が計算される。</p>
<i>compq</i>	<p>CHARACTER*1。'V' または 'N' でなければならない。</p> <p><i>compq</i>='V' の場合、Schur ベクトルの <i>Q</i> が更新される。</p> <p><i>compq</i>='N' の場合、Schur ベクトルは更新されない。</p>
<i>select</i>	<p>LOGICAL。</p> <p>配列、次元は <math>\max(1, n)</math> 以上。</p> <p>選択するクラスタに属する固有値を指定する。</p> <p>固有値 <math>\lambda_j</math> を選択するには、<i>select</i>(<i>j</i>) が .TRUE. でなければならない。</p> <p>実数型の場合: 固有値 <math>\lambda_j</math> と <math>\lambda_{j+1}</math> (対応する <math>2 \times 2</math> の対角ブロック) の複素共役ペアを選択するには、<i>select</i>(<i>j</i>) や <i>select</i>(<i>j</i>+1) が .TRUE. でなければならない。複素共役 <math>\lambda_j</math> と <math>\lambda_{j+1}</math> は、束に両方とも含まれているか、あるいは両方とも束から除外されていなければならない。</p>
<i>n</i>	<p>INTEGER。行列 <i>T</i> の次数 (<math>n \geq 0</math>)。</p>
<i>t, q, work</i>	<p>REAL (strsen の場合 )</p> <p>DOUBLE PRECISION (dtrsen の場合 )</p> <p>COMPLEX (ctrsen の場合 )</p> <p>DOUBLE COMPLEX (ztrsen の場合 )。</p> <p>配列:</p> <p><i>t</i>(<i>ldt</i>,*)。 <math>n \times n</math> の <i>T</i>。</p> <p><i>t</i> の第 2 次元は、<math>\max(1, n)</math> 以上でなければならない。</p> <p><i>q</i>(<i>ldq</i>,*)</p> <p><i>compq</i>='V' の場合、<i>q</i> に Schur ベクトルの <i>Q</i> を格納しなければならない。</p> <p><i>compq</i>='N' の場合、<i>q</i> は参照されない。</p> <p><i>q</i> の第 2 次元は、<math>\max(1, n)</math> 以上 (<i>compq</i>='V' の場合)、あるいは 1 以上 (<i>compq</i>='N' の場合) でなければならない。</p>

*work(lwork)* は、ワークスペース配列である。

複素数の場合: *job*='N' の場合、配列 *work* は参照されない。

必要なワークスペースの実際の量は、 $n^2/4$  (*job*='E' の場合)、あるいは  $n^2/2$  (*job*='V' または 'B' の場合) を超えることはできない。

*ldt* INTEGER。 *t* の第 1 次元。  $\max(1, n)$  以上。

*ldq* INTEGER。 *q* の第 1 次元。  
*compq*='N' の場合、 $ldq \geq 1$ 。  
*compq*='V' の場合、 $ldq \geq \max(1, n)$ 。

*lwork* INTEGER。 配列 *work* の次元。  
*job*='V' または 'B' の場合、  
 $lwork \geq \max(1, 2m(n-m))$ 。  
*job*='E' の場合、 $lwork \geq \max(1, m(n-m))$   
*job*='N' の場合、 $lwork \geq 1$  (複素数型の場合) または  $lwork \geq \max(1, n)$  (実数型の場合)。  
*lwork* = -1 の場合、ワークスペースのクエリとみなされ、ルーチンは *work* 配列の最適サイズだけを計算し、その値を *work* 配列の最初のエントリとして返す。xerbla は *lwork* に関するエラー・メッセージを生成しない。

*iwork* INTEGER。  
*iwork(liwork)* は、ワークスペース配列である。  
*job*='N' または 'E' の場合、配列 *iwork* は参照されない。  
*job*='V' または 'B' の場合、必要なワークスペースの実際の量は  $n^2/2$  を超えることはできない。

*liwork* INTEGER。  
配列 *iwork* の次元。  
*job*='V' または 'B' の場合、  
 $liwork \geq \max(1, 2m(n-m))$ 。  
*job*='E' または 'E' の場合、 $liwork \geq 1$ 。  
*liwork* = -1 の場合、ワークスペースのクエリとみなされ、ルーチンは *iwork* 配列の最適サイズだけを計算し、その値を *iwork* 配列の最初のエントリとして返す。xerbla は *liwork* に関するエラー・メッセージを生成しない。

## 出力パラメータ

*t* 更新後の行列 *R* によって上書きされる。

$q$	$compq = 'V'$ の場合、 $q$ には、Schur ベクトルで構成される更新後の行列が格納される。 $Q$ の先頭から $m$ 個の列は、指定した不変部分空間の直交基を形成する。
$w$	COMPLEX (ctrsen の場合) DOUBLE COMPLEX (ztrsen の場合)。 配列、次元は $\max(1, n)$ 以上。 格納される $R$ の固有値。一連の固有値は、 $R$ の対角成分と同じ順序で格納される。
$wr, wi$	REAL (strsen の場合) DOUBLE PRECISION (dtrsen の場合) 配列、次元は $\max(1, n)$ 以上。 $R$ の順序変更後の固有値の実数部と虚数部が、それぞれ格納される。一連の固有値は、 $R$ の対角成分と同じ順序で格納される。ただし、複素数の固有値の条件が非常に悪い場合には、その値が順序変更前から大幅に変更される場合がある。
$m$	INTEGER。 複素数型の場合: 指定した不変部分空間の個数であり、選択した固有値の個数と同じである ( <i>select</i> を参照)。 実数型の場合: 指定した不変部分空間の次元。選択した実数の固有値ごとに 1 を、固有値の選択した複素共役ペアごとに 2 をカウントすれば、 $m$ の値が得られる ( <i>select</i> を参照)。  次の制約がある。 $0 \leq m \leq n$ 。
$s$	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 $job = 'E'$ または $'B'$ の場合、 $s$ は、固有値の選択した束の平均の条件数の逆数の下限になる。 $m = 0$ または $n$ の場合、 $s = 1$ 。 実数型の場合: $info = 1$ の場合、 $s$ はゼロに設定される。 $job = 'N'$ または $'V'$ の場合、 $s$ は参照されない。
$sep$	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 $job = 'V'$ または $'B'$ の場合、 $sep$ は、指定した不変部分空間の条件数の逆数の見積もりである。 $m = 0$ または $n$ の場合、 $sep = \ T\ $ 。 実数型の場合: $info = 1$ の場合、 $sep$ はゼロに設定される。 $job = 'N'$ または $'E'$ の場合、 $sep$ は参照されない。
$work(1)$	終了時に、 $info = 0$ の場合、 $work(1)$ は $lwork$ の必要最小サイズを返す。

<i>iwork</i> (1)	終了時に、 <i>info</i> = 0 の場合、 <i>iwork</i> (1) は <i>liwork</i> の必要最小サイズを返す。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正であったことを示す。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *trsen* のインターフェイスの詳細を以下に示す。

<i>t</i>	サイズ ( <i>n</i> , <i>n</i> ) の行列 <i>T</i> を格納する。
<i>select</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>wr</i>	長さ ( <i>n</i> ) のベクトルを格納する。実数型でのみ使用される。
<i>wi</i>	長さ ( <i>n</i> ) のベクトルを格納する。実数型でのみ使用される。
<i>w</i>	長さ ( <i>n</i> ) のベクトルを格納する。複素数型でのみ使用される。
<i>q</i>	サイズ ( <i>n</i> , <i>n</i> ) の行列 <i>Q</i> を格納する。
<i>compq</i>	引数 <i>q</i> の存在に基づいて以下のように復元される。 <i>compq</i> = 'V' ( <i>q</i> が存在する場合)、 <i>compq</i> = 'N' ( <i>q</i> が省略された場合)。
<i>job</i>	引数 <i>s</i> と <i>sep</i> の存在に基づいて以下のように復元される。 <i>job</i> = 'B' ( <i>s</i> と <i>sep</i> の両方が存在する場合)、 <i>job</i> = 'E' ( <i>s</i> が存在し、 <i>sep</i> が省略された場合)、 <i>job</i> = 'V' ( <i>s</i> が省略され、 <i>sep</i> が存在する場合)、 <i>job</i> = 'N' ( <i>s</i> と <i>sep</i> の両方が省略された場合)。

### アプリケーション・ノート

計算で求めた行列 *R* は、行列 *T* + *E* の近似値である。ここで、 $\|E\|_2 = O(\epsilon)\|T\|_2$ 、 $\epsilon$  はマシン精度である。

計算で求めた *s* は、 $(\min(m, n-m))^{1/2}$  を超える係数で条件数の真の逆数を過小評価できない。*sep* は、真の値から  $(m*n-m^2)^{1/2}$  だけ異なっている場合がある。計算で求めた不変部分空間と真の部分空間の間の角は、 $O(\epsilon)\|A\|_2/sep$  である。

2 × 2 の対角ブロックが順序変更に関与する場合は、一般に、その非対角成分が変更さ

れる。そのブロックの条件が非常に悪い場合を除き、そのブロックの対角成分と固有値は変更されない。条件が非常に悪い場合は、それらが明確に変更される場合もある。 $2 \times 2$  のブロックは、2つの  $1 \times 1$  のブロックに分割できる。つまり、複素数の固有値ペアを、純粋な実数にできる。ただし、固有値の値が、順序の変更によって変更されたりはしない。

---

### ?trsyl

実準三角行列または複素三角行列に関するシルベスター式を解く。

---

#### 構文

##### Fortran 77:

```
call strsyl(trana, tranb, isgn, m, n, a, lda, b, ldb, c, ldc, scale, info)
call dtrsyl(trana, tranb, isgn, m, n, a, lda, b, ldb, c, ldc, scale, info)
call ctrsyl(trana, tranb, isgn, m, n, a, lda, b, ldb, c, ldc, scale, info)
call ztrsyl(trana, tranb, isgn, m, n, a, lda, b, ldb, c, ldc, scale, info)
```

##### Fortran 95:

```
call trsyl(a, b, c, scale [,trana] [,tranb] [,isgn] [,info])
```

#### 説明

このルーチンは、シルベスターの行列方程式  $\text{op}(A)X \pm X\text{op}(B) = \alpha C$  を解く。 $\text{op}(A) = A$  または  $A^H$  であり、行列  $A$  と  $B$  は上三角行列 (実数型の場合は標準形の Schur 形式の上準三角行列) である。 $\alpha \leq 1$  は、 $X$  のオーバーフローを防止するためにこのルーチンで求めるスケール係数である。 $A$  は  $m \times m$ 、 $B$  は  $n \times n$ 、 $C$  と  $X$  はどちらも  $m \times n$  である。行列  $X$  は、後方置換によって簡単に求められる。

この方程式は、 $\alpha_i \pm \beta_i \neq 0$  の場合にだけ、一意な解を持つ。 $\{\alpha_i\}$  と  $\{\beta_i\}$  はそれぞれ、 $A$  と  $B$  の固有値である。符合 (+ または -) は、解くべき方程式の中で使われているものと同じである。



**入力パラメータ**

<i>trana</i>	CHARACTER*1. 'N'、'T'、または 'C' のいずれかでなければならない。 <i>trana</i> ='N' の場合、 $\text{op}(A) = A$ 。 <i>trana</i> ='T' の場合、 $\text{op}(A) = A^T$ (実数型の場合のみ)。 <i>trana</i> ='C' の場合、 $\text{op}(A) = A^H$ 。
<i>tranb</i>	CHARACTER*1. 'N'、'T'、または 'C' のいずれかでなければならない。 <i>tranb</i> ='N' の場合、 $\text{op}(B) = B$ 。 <i>tranb</i> ='T' の場合、 $\text{op}(B) = B^T$ (実数型の場合のみ)。 <i>tranb</i> ='C' の場合、 $\text{op}(B) = B^H$ 。
<i>isgn</i>	INTEGER。シルベスター式の形式を指定する。 <i>isgn</i> = +1 の場合、 $\text{op}(A)X + X\text{op}(B) = \alpha C$ 。 <i>isgn</i> = -1 の場合、 $\text{op}(A)X - X\text{op}(B) = \alpha C$ 。
<i>m</i>	INTEGER。 <i>A</i> の次数、および <i>X</i> と <i>C</i> ( $m \geq 0$ ) の行数。
<i>n</i>	INTEGER。 <i>B</i> の次数、および <i>X</i> と <i>C</i> ( $n \geq 0$ ) の行数。
<i>a, b, c</i>	REAL (strsyl の場合) DOUBLE PRECISION (dtrsyl の場合) COMPLEX (ctrsyl の場合) DOUBLE COMPLEX (ztrsyl の場合)。 配列： <i>a</i> ( <i>lda</i> ,*) には、行列 <i>A</i> を格納する。 <i>a</i> の第 2 次元は、 $\max(1, m)$ 以上でなければならない。 <i>b</i> ( <i>ldb</i> ,*) には、行列 <i>B</i> を格納する。 <i>b</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>c</i> ( <i>ldc</i> ,*) には、行列 <i>C</i> を格納する。 <i>c</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $\max(1, m)$ 以上。
<i>ldb</i>	INTEGER。 <i>b</i> の第 1 次元。 $\max(1, n)$ 以上。
<i>ldc</i>	INTEGER。 <i>c</i> の第 1 次元。 $\max(1, n)$ 以上でなければならない。

**出力パラメータ**

<i>c</i>	解行列 <i>X</i> によって上書きされる。
----------	--------------------------

<i>scale</i>	REAL ( 単精度の場合 ) DOUBLE PRECISION ( 倍精度の場合 )。 スケール係数 $\alpha$ の値。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。  <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正であったことを示す。  <i>info</i> = 1 の場合、 <i>A</i> と <i>B</i> は共通または近似の固有値を持ち、その摂動値を使ってこの方程式が解かれる。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `trsyl` のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ ( <i>m</i> , <i>m</i> ) の行列 <i>A</i> を格納する。
<i>b</i>	サイズ ( <i>n</i> , <i>n</i> ) の行列 <i>B</i> を格納する。
<i>c</i>	サイズ ( <i>m</i> , <i>n</i> ) の行列 <i>C</i> を格納する。
<i>trana</i>	'N'、'C'、または 'T' でなければならない。デフォルト値は 'N'。
<i>tranb</i>	'N'、'C'、または 'T' でなければならない。デフォルト値は 'N'。
<i>isgn</i>	+1 または -1 でなければならない。デフォルト値は +1 である。

## アプリケーション・ノート

*X* が正確な解、*Y* が対応する計算で求めた解、*R* が誤差行列であり、 $R = C - (AY \pm YB)$  とする。その場合の誤差は、常に小さくなる。

$$\|R\|_F = O(\epsilon) (\|A\|_F + \|B\|_F) \|Y\|_F$$

ただし、*Y* は、わずかに摂動した方程式の場合、正確な解とならない場合もある。つまり、この解は後方安定ではない。

前方誤差に関しては、次の制限が成立する。

$$\|Y - X\|_F \leq \|R\|_F / \text{sep}(A, B)$$

ただし、これは、大幅な過大評価である場合もある。 $\text{sep}(A, B)$  の定義については、[\[Golub96\]](#) を参照。

実数型の場合の浮動小数演算のおおよその総数は、 $m \cdot n \cdot (m + n)$  である。複素数型の場合、この 4 倍になる。

## 汎用非対称固有値問題

このセクションでは、汎用非対称固有値問題を解いたり、行列ペアの Schur 因子分解を順序変更したり、関連する種々の計算タスクを実行するための LAPACK ルーチンについて説明する。

汎用非対称固有値問題は、次のように説明できる。 $n \times n$  の非対称行列 (または非エルミート行列) のペア  $A$  と  $B$  があるとき、次の 2 つの式を満たす汎用固有値  $\lambda$  と、それに対応する汎用固有ベクトル  $x$  と  $y$  を見つけ出す。

$$Ax = \lambda Bx \quad (\text{右汎用固有ベクトル } x)$$

$$y^H A = \lambda y^H B \quad (\text{左汎用固有ベクトル } y)$$

[表 4-6](#) に、汎用非対称固有値問題と汎用シルベスター式を解くための LAPACK ルーチン (Fortran-77 インターフェイス) を示す。

Fortran-95 インターフェイスのルーチン名はそれぞれ、先頭の記号がない ([ルーチン命名規則](#) を参照)。

**表 4-6 汎用非対称固有値問題を解く計算ルーチン**

ルーチン名	機能
<a href="#">?qghrd</a>	直交 / ユニタリ変換を使用して、行列ペアを汎用上 Hessenberg 形式に縮退する。
<a href="#">?qgbal</a>	一般の実 / 複素行列ペアを平衡化する。
<a href="#">?qgbak</a>	汎用固有値問題の右または左固有ベクトルを求める。
<a href="#">?hgeqz</a>	QZ 法により行列ペア (H, T) の汎用固有値を求める。
<a href="#">?tgevc</a>	上三角行列ペアから右または左汎用固有ベクトルの一部または全部を求める。
<a href="#">?tgexc</a>	行列ペア (A, B) において、ある対角ブロックが別の行インデックスに移動するように、(A, B) の汎用 Schur 分解の順序を変更する。
<a href="#">?tgsen</a>	選択した固有値クラスが行列ペア (A, B) の主対角ブロックに移動するように、(A, B) の汎用 Schur 分解の順序を変更する。
<a href="#">?tgsyl</a>	汎用シルベスター式を解く。
<a href="#">?tgsna</a>	汎用実 Schur 標準形の行列ペアに関して、指定した固有値と固有ベクトルの条件数の逆数を見積もる。

## ?gghrd

直交/ユニタリ変換を使用して、行列ペアを汎用上 *Hessenberg* 形式に縮退する。

### 構文

#### Fortran 77:

```
call sgghrd(compq, compz, n, ilo, ihi, a, lda, b, ldb, q, ldq, z, ldz, info)
call dgghrd(compq, compz, n, ilo, ihi, a, lda, b, ldb, q, ldq, z, ldz, info)
call cgghrd(compq, compz, n, ilo, ihi, a, lda, b, ldb, q, ldq, z, ldz, info)
call zgghrd(compq, compz, n, ilo, ihi, a, lda, b, ldb, q, ldq, z, ldz, info)
```

#### Fortran 95:

```
call gghrd(a, b [,ilo] [,ihi] [,q] [,z] [,compq] [,compz] [,info])
```

### 説明

このルーチンは、直交/ユニタリ変換を使用して、実/複素行列ペア ( $A, B$ ) を、汎用上 *Hessenberg* 形式に縮退する。ここで、 $A$  は一般行列、 $B$  は上三角行列である。汎用固有値問題は、 $Ax = \lambda Bx$  で表される。一般に、 $B$  は、 $QR$  因子分解を計算し、直交行列  $Q$  を式の左辺に移動すると、上三角行列になる。

このルーチンは、 $A$  を次のように *Hessenberg* 行列  $H$  に縮退し、  

$$Q^H A Z = H$$

同時に  $B$  を次のように別の上三角行列  $T$  に変換すると、  

$$Q^H B Z = T$$

元の問題を標準形式  $Hy = \lambda Ty$  (ただし、 $y = Z^H x$ ) に縮退する。

直交/ユニタリ行列  $Q$  と  $Z$  は、*Givens* 回転の積で求められる。この2つの行列は、明示的に求められ、次のように入力行列  $Q_I$  と  $Z_I$  に後からも掛けられる。

$$Q_I A Z_I^H = (Q_I Q) H (Z_I Z)^H$$

$$Q_I B Z_I^H = (Q_I Q) T (Z_I Z)^H$$

$Q_I$  が、元の式  $Ax = \lambda Bx$  における  $B$  の  $QR$  因子分解で得た直交行列であれば、?gghrd は元の問題を汎用 *Hessenberg* 形式に縮退する。

**入力パラメータ**

<i>compq</i>	<p>CHARACTER*1. 'N'、'I'、または 'V' のいずれかでなければならない。</p> <p><i>compq</i>='N' の場合、行列 <math>Q</math> は計算されない。</p> <p><i>compq</i>='I' の場合、<math>Q</math> は単位行列に初期化され、直交 / ユニタリ行列 <math>Q</math> が返される。</p> <p><i>compq</i>='V' の場合、呼び出し時に、<math>Q</math> には、直交 / ユニタリ行列 <math>Q_1</math> が格納されていなければならない。終了時に、積 <math>Q_1 Q</math> が返される。</p>
<i>compz</i>	<p>CHARACTER*1. 'N'、'I'、または 'V' のいずれかでなければならない。</p> <p><i>compz</i>='N' の場合、行列 <math>Z</math> は計算されない。</p> <p><i>compz</i>='I' の場合、<math>Z</math> は単位行列に初期化され、直交 / ユニタリ行列 <math>Z</math> が返される。</p> <p><i>compz</i>='V' の場合、<math>Z</math> には、呼び出し時に、直交 / ユニタリ行列 <math>Z_1</math> が格納されていなければならない。終了時に、積 <math>Z_1 Z</math> が返される。</p>
<i>n</i>	INTEGER. 行列 $A$ と $B$ の次数 ( $n \geq 0$ ).
<i>ilo, ihi</i>	<p>INTEGER. <i>ilo</i> と <i>ihi</i> は、<math>A</math> のどの行と列を縮退するかをマークする。<math>A</math> において、行と列が <math>1:i\text{lo}-1</math> と <math>i\text{hi}+1:n</math> の部分は、すでに上三角形になっているものとする。一般に、<i>ilo</i> と <i>ihi</i> の値は、先に呼び出した <a href="#">?qgbal</a> によって設定されるが、そうでない場合はそれぞれ 1 と <math>n</math> に設定する。次の制約がある。</p> <p><math>n &gt; 0</math> の場合、<math>1 \leq i\text{lo} \leq i\text{hi} \leq n</math>。</p> <p><math>n = 0</math> の場合、<math>i\text{lo} = 1</math> かつ <math>i\text{hi} = 0</math>。</p>
<i>a, b, q, z</i>	<p>REAL (sgghrd の場合 )</p> <p>DOUBLE PRECISION (dgghrd の場合 )</p> <p>COMPLEX (cgghrd の場合 )</p> <p>DOUBLE COMPLEX (zgghrd の場合 )。</p> <p>配列 :</p> <p><math>a(l\text{da}, *)</math> <math>n \times n</math> の一般行列 <math>A</math> を格納する。</p> <p><math>a</math> の第 2 次元は、<math>\max(1, n)</math> 以上でなければならない。</p> <p><math>b(l\text{db}, *)</math> <math>n \times n</math> の上三角行列 <math>B</math> を格納する。</p> <p><math>b</math> の第 2 次元は、<math>\max(1, n)</math> 以上でなければならない。</p> <p><math>q(l\text{dq}, *)</math></p> <p><i>compq</i>='N' の場合、<math>q</math> は参照されない。</p> <p><i>compq</i>='I' の場合、呼び出し時に、<math>q</math> を設定する必要はない。</p> <p><i>compq</i>='V' の場合、<math>q</math> には、直交 / ユニタリ行列 <math>Q_1</math> ( 通常は <math>B</math> の <math>QR</math> 因子分解 ) が格納されていなければならない。</p> <p><math>q</math> の第 2 次元は、<math>\max(1, n)</math> 以上でなければならない。</p>

	$z(ldz, *)$ $compq = 'N'$ の場合、 $z$ は参照されない。 $compq = 'I'$ の場合、呼び出し時に、 $z$ を設定する必要はない。 $compq = 'V'$ の場合、 $z$ には、直交 / ユニタリ行列 $Z_1$ が格納されていなければならない。 $z$ の第 2 次元は、 $\max(1, n)$ 以上でなければならない。
$lda$	INTEGER。 $a$ の第 1 次元。 $\max(1, n)$ 以上。
$ldb$	INTEGER。 $b$ の第 1 次元。 $\max(1, n)$ 以上。
$ldq$	INTEGER。 $q$ の第 1 次元。 $compq = 'N'$ の場合、 $ldq \geq 1$ 。 $compq = 'I'$ または $'V'$ の場合、 $ldq \geq \max(1, n)$ 。
$ldz$	INTEGER。 $z$ の第 1 次元。 $compq = 'N'$ の場合、 $ldz \geq 1$ 。 $compq = 'I'$ または $'V'$ の場合、 $ldz \geq \max(1, n)$ 。

## 出力パラメータ

$a$	終了時に、 $A$ の上三角部分と最初の劣対角成分が上 Hessenberg 行列 $H$ で上書きされ、残りの成分はゼロに設定される。
$b$	終了時に、上三角行列 $T = Q^H B Z$ で上書きされる。対角成分より下の各成分はゼロに設定される。
$q$	$compq = 'I'$ の場合、 $q$ に、直交 / ユニタリ行列 $Q$ ( $Q^H$ は、 $A$ と $B$ の左側に適用される Givens 変換の積) が格納される。 $compq = 'V'$ の場合、 $q$ は積 $Q_1 Q$ で上書きされる。
$z$	$compq = 'I'$ の場合、 $z$ に、直交 / ユニタリ行列 $Z$ ( $A$ と $B$ の右側に適用される Givens 変換の積) が格納される。 $compq = 'V'$ の場合、 $z$ は積 $Z_1 Z$ で上書きされる。
$info$	INTEGER。 $info = 0$ の場合、実行は正常に終了したことを示す。 $info = -i$ の場合、 $i$ 番目のパラメータの値が不正であったことを示す。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `gghrd` のインターフェイスの詳細を以下に示す。

<code>a</code>	サイズ $(n,n)$ の行列 $A$ を格納する。
<code>b</code>	サイズ $(n,n)$ の行列 $B$ を格納する。
<code>q</code>	サイズ $(n,n)$ の行列 $Q$ を格納する。
<code>z</code>	サイズ $(n,n)$ の行列 $Z$ を格納する。
<code>ilo</code>	この引数のデフォルト値は、 <code>ilo = 1</code> である。
<code>ihi</code>	この引数のデフォルト値は、 <code>ihi = n</code> である。
<code>compq</code>	省略された場合、この引数は、引数 <code>q</code> の存在に基づいて以下のように復元される。 <code>compq = 'I'</code> ( <code>q</code> が存在する場合)、 <code>compq = 'N'</code> ( <code>q</code> が省略された場合)。 存在する場合、 <code>compz</code> は ' <code>I</code> ' または ' <code>V</code> ' と等しくなければならず、引数 <code>q</code> も存在しなければならない。 <code>compz</code> が存在し、 <code>q</code> が省略された場合、エラー条件がセットされる。
<code>compz</code>	省略された場合、この引数は、引数 <code>z</code> の存在に基づいて以下のように復元される。 <code>compz = 'I'</code> ( <code>z</code> が存在する場合)、 <code>compz = 'N'</code> ( <code>z</code> が省略された場合)。 存在する場合、 <code>compz</code> は ' <code>I</code> ' または ' <code>V</code> ' と等しくなければならず、引数 <code>z</code> も存在しなければならない。 <code>compz</code> が存在し、 <code>z</code> が省略された場合、エラー条件がセットされる。

## ?ggbal

一般の実/複素行列ペアを平衡化する。

### 構文

#### Fortran 77:

```
call sggbal(job, n, a, lda, b, ldb, ilo, ihi, lscale, rscale, work, info)
call dggbal(job, n, a, lda, b, ldb, ilo, ihi, lscale, rscale, work, info)
call cggbal(job, n, a, lda, b, ldb, ilo, ihi, lscale, rscale, work, info)
call zggbal(job, n, a, lda, b, ldb, ilo, ihi, lscale, rscale, work, info)
```

## Fortran 95:

```
call ggbal(a, b [,ilo] [,ihi] [,lscale] [,rscale] [,job] [,info])
```

### 説明

このルーチンは、一般の実/複素行列ペア  $(A, B)$  を平衡化する。すなわち、まず相似変換によって  $A$  と  $B$  を置換し、固有値を対角線上の最初の  $1 \sim ilo-1$  と最後の  $ihi+1 \sim n$  の成分に分離する。次に、 $ilo \sim ihi$  の行と列に対角相似変換を適用して、これらの行と列のノルムができる限り近くなるようにする。この2つのステップはオプションである。

平衡化により、行列の 1- ノルムが縮退され、汎用固有値問題  $Ax = \lambda Bx$  における固有値と固有ベクトルの計算精度が向上する。

### 入力パラメータ

<i>job</i>	CHARACTER*1。 $A$ と $B$ に対して実行する演算を指定する。'N'、'P'、'S'、または 'B' のいずれかでなければならない。 $job='N'$ の場合、演算は行わず、 $ilo = 1$ 、 $ihi = n$ 、 $lscale(i) = 1.0$ 、 $rscale(i) = 1.0$ (ただし、 $i = 1, \dots, n$ ) に設定する。 $job='P'$ の場合、置換のみ。 $job='S'$ の場合、スケーリングのみ。 $job='B'$ の場合、置換とスケーリング。
<i>n</i>	INTEGER。行列 $A$ と $B$ の次数 ( $n \geq 0$ )。
<i>a, b</i>	REAL (sggbal の場合) DOUBLE PRECISION (dggbal の場合) COMPLEX (cggbal の場合) DOUBLE COMPLEX (zggbal の場合)。 配列: $a(lda, *)$ には、行列 $A$ を格納する。 $a$ の第2次元は、 $\max(1, n)$ 以上でなければならない。 $b(l db, *)$ には、行列 $B$ を格納する。 $b$ の第2次元は、 $\max(1, n)$ 以上でなければならない。
<i>lda</i>	INTEGER。 $a$ の第1次元。 $\max(1, n)$ 以上。
<i>ldb</i>	INTEGER。 $b$ の第1次元。 $\max(1, n)$ 以上。
<i>work</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 ワークスペース配列、次元は $\max(1, 6n)$ 以上。



**出力パラメータ**

<i>a</i> , <i>b</i>	平衡化した行列 <i>A</i> と <i>B</i> でそれぞれが上書きされる。 <i>job</i> ='N' の場合、 <i>a</i> と <i>b</i> は参照されない。
<i>ilo</i> , <i>ihi</i>	INTEGER。終了時に、 <i>ilo</i> と <i>ihi</i> は、 $i > j$ で $j=1, \dots, ilo-1$ または $i=ihi+1, \dots, n$ であれば、 $a(i, j)=0$ と $b(i, j)=0$ が満たされるような整数に設定される。  <i>job</i> ='N' または 'S' の場合、 <i>ilo</i> = 1、 <i>ihi</i> = <i>n</i> に設定される。
<i>lscale</i> , <i>rscale</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 配列、次元は $\max(1, n)$ 以上。  <i>lscale</i> には、 <i>A</i> と <i>B</i> の左側に適用される置換およびスケーリング係数の各成分が格納される。 行 <i>j</i> と交換する行のインデックスを $P_j$ とし、行 <i>j</i> に適用されるスケール係数を $D_j$ とすると、  $lscale(j) = P_j \text{ (} j = 1, \dots, ilo-1 \text{ の場合)}$ $= D_j \text{ (} j = ilo, \dots, ihi \text{ の場合)}$ $= P_j \text{ (} j = ihi+1, \dots, n \text{ の場合)}$ <i>rscale</i> には、 <i>A</i> と <i>B</i> の右側に適用される置換およびスケーリング係数の各成分が格納される。 列 <i>j</i> と交換する列のインデックスを $P_j$ とし、列 <i>j</i> に適用されるスケール係数を $D_j$ とすると、  $rscale(j) = P_j \text{ (} j = 1, \dots, ilo-1 \text{ の場合)}$ $= D_j \text{ (} j = ilo, \dots, ihi \text{ の場合)}$ $= P_j \text{ (} j = ihi+1, \dots, n \text{ の場合)}$ 交換は、まず <i>n</i> から <i>ihi</i> +1 の順、次に 1 から <i>ilo</i> -1 の順に行われる。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正であったことを示す。

**Fortran 95 インターフェイス・ノート**

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `ggbal` のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ $(n,n)$ の行列 <i>A</i> を格納する。
<i>b</i>	サイズ $(n,n)$ の行列 <i>B</i> を格納する。
<i>lscale</i>	長さ $(n)$ のベクトルを格納する。
<i>rscale</i>	長さ $(n)$ のベクトルを格納する。
<i>ilo</i>	この引数のデフォルト値は、 <i>ilo</i> = 1 である。
<i>ihi</i>	この引数のデフォルト値は、 <i>ihi</i> = <i>n</i> である。
<i>job</i>	'B'、'S'、'P'、または 'N' でなければならない。デフォルト値は 'B'。

---

### ?ggbak

汎用固有値問題の右または左固有ベクトルを求める。

---

#### 構文

##### Fortran 77:

```
call sggbak(job, side, n, ilo, ihi, lscale, rscale, m, v, ldv, info)
call dggbak(job, side, n, ilo, ihi, lscale, rscale, m, v, ldv, info)
call cggbak(job, side, n, ilo, ihi, lscale, rscale, m, v, ldv, info)
call zggbak(job, side, n, ilo, ihi, lscale, rscale, m, v, ldv, info)
```

##### Fortran 95:

```
call ggbak(v [,ilo] [,ihi] [,lscale] [,rscale] [,job] [,info])
```

#### 説明

このルーチンは、平衡化した行列ペア ([?ggbal](#) から返されたもの) で計算した固有ベクトルを逆変換して、実数 / 複素数の汎用固有値問題  $Ax = \lambda Bx$  における右または左固有ベクトルを求める。

#### 入力パラメータ

*job* CHARACTER\*1。必要な逆変換のタイプを指定する、'N'、'P'、'S'、または 'B' でなければならない。  
*job* = 'N' の場合、演算を行わずにリターンする。

*job*='P' の場合、置換に対する逆変換のみ行う。  
*job*='S' の場合、スケーリングに対する逆変換のみ行う。  
*job*='B' の場合、置換とスケーリングに対する逆変換を行う。  
 この引数は、?ggbal に指定した引数 *job* と同じ値でなければならない。

*side* CHARACTER\*1。'L' または 'R' でなければならない。  
*side*='L' の場合、*v* に左固有ベクトルが格納される。  
*side*='R' の場合、*v* に右固有ベクトルが格納される。

*n* INTEGER。行列 *V* の行数 ( $n \geq 0$ )。

*ilo, ihi* INTEGER。?gebal で決定された整数 *ilo* と *ihi*。次の制約がある。  
 $n > 0$  の場合、 $1 \leq ilo \leq ihi \leq n$ 。  
 $n = 0$  の場合、 $ilo = 1$  かつ  $ihi = 0$ 。

*lscale, rscale* REAL (単精度の場合)  
 DOUBLE PRECISION (倍精度の場合)。  
 配列、次元は  $\max(1, n)$  以上。

配列 *lscale* には、*A* と *B* の左側に適用する置換およびスカラ係数の詳細 (?ggbal から返された値) を格納する。

配列 *rscale* には、*A* と *B* の右側に適用する置換およびスカラ係数の詳細 (?ggbal から返された値) を格納する。

*m* INTEGER。行列 *V* の列数 ( $m \geq 0$ )。

*v* REAL (sggbak の場合)  
 DOUBLE PRECISION (dggbak の場合)  
 COMPLEX (cggbak の場合)  
 DOUBLE COMPLEX (zggbak の場合)。  
 配列 *v*(*ldv*,\*)。変換する右または左固有ベクトルが入った行列 ([?tgevc](#) から返された値) を格納する。  
*v* の第 2 次元は、 $\max(1, m)$  以上でなければならない。

*ldv* INTEGER。*v* の第 1 次元。  $\max(1, n)$  以上。

## 出力パラメータ

*v* 変換後の固有ベクトルで上書きされる。

*info* INTEGER。  
*info* = 0 の場合、実行は正常に終了したことを示す。  
*info* = -*i* の場合、*i* 番目のパラメータの値が不正であったことを示す。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `ggbak` のインターフェイスの詳細を以下に示す。

<code>v</code>	サイズ $(n,m)$ の行列 $V$ を格納する。
<code>lscale</code>	長さ $(n)$ のベクトルを格納する。
<code>rscale</code>	長さ $(n)$ のベクトルを格納する。
<code>ilo</code>	この引数のデフォルト値は、 $ilo = 1$ である。
<code>ihi</code>	この引数のデフォルト値は、 $ihi = n$ である。
<code>job</code>	'B'、'S'、'P'、または 'N' でなければならない。デフォルト値は 'B'。
<code>side</code>	省略された場合、この引数は、引数 <code>lscale</code> と <code>rscale</code> の存在に基づいて以下のように復元される。 <code>side = 'L'</code> ( <code>lscale</code> が存在し、 <code>rscale</code> が省略された場合)、 <code>side = 'R'</code> ( <code>lscale</code> が省略され、 <code>rscale</code> が存在する場合)。 <code>lscale</code> と <code>rscale</code> の両方が存在するか、両方が省略された場合、エラー条件がセットされる。

---

## ?hgeqz

*QZ* 法により行列ペア  $(H, T)$  の汎用固有値を求める。

---

### 構文

#### Fortran 77:

```
call shgeqz(job, compq, compz, n, ilo, ihi, h, ldh, t, ldt, alphas,
            alphai, beta, q, ldq, z, ldz, work, lwork, info)
call dhgeqz(job, compq, compz, n, ilo, ihi, h, ldh, t, ldt, alphas,
            alphai, beta, q, ldq, z, ldz, work, lwork, info)
call chgeqz(job, compq, compz, n, ilo, ihi, h, ldh, t, ldt, alpha,
            beta, q, ldq, z, ldz, work, lwork, rwork, info)
```

```
call zhgeqz(job, compq, compz, n, ilo, ihi, h, ldh, t, ldt, alpha,
            beta, q, ldq, z, ldz, work, lwork, rwork, info)
```

**Fortran 95:**

```
call hgeqz(h, t [,ilo] [,ihi] [,alphan] [,alphai] [,beta] [,q] [,z] [,job]
[,compq] [,compz] [,info])
call hgeqz(h, t [,ilo] [,ihi] [,alpha] [,beta] [,q] [,z] [,job] [,compq]
[,compz] [,info])
```

**説明**

このルーチンは、ダブルシフト版 (実数型の場合) またはシングルシフト版 (複素数型の場合) の  $QZ$  法を使用して、実 / 複素行列ペア  $(H, T)$  の固有値を計算する。ただし、 $H$  は上 Hessenberg 行列、 $T$  は上三角行列である。  
このタイプの行列ペアは、実 / 複素行列ペア  $(A, B)$  の汎用上 Hessenberg 形式への縮退で得られる。

$$A = Q_1 H Z_1^H, \quad B = Q_1 T Z_1^H$$

?gghrd によって計算される。

**実数型の場合:**

$job='s'$  の場合、Hessenberg- 三角行列ペア  $(H, T)$  も、次のような汎用 Schur 形式に縮退される。

$$H = Q S Z^T, \quad T = Q P Z^T,$$

$Q$  と  $Z$  は直交行列、 $P$  は上三角行列、 $S$  は、 $1 \times 1$  と  $2 \times 2$  の対角ブロックを持つ準三角行列である。

$1 \times 1$  のブロックは、行列ペア  $(H, T)$  の実固有値に対応し、 $2 \times 2$  のブロックは固有値の複素共役ペアに対応する。

また、 $S$  の  $2 \times 2$  のブロックに対応する  $P$  の  $2 \times 2$  の上三角対角ブロックは、正値対角形式に縮退される。すなわち、 $S(j+1, j)$  がゼロでなければ、 $P(j+1, j) = P(j, j+1) = 0$ 、 $P(j, j) > 0$ 、 $P(j+1, j+1) > 0$  となる。

**複素数型の場合:**

$job='s'$  の場合、Hessenberg- 三角行列ペア  $(H, T)$  も、次のような汎用 Schur 形式に縮退される。

$$H = Q S Z^H, \quad T = Q P Z^H$$

$Q$  と  $Z$  はユニタリ行列、 $S$  と  $P$  は上三角行列である。

すべての場合:

オプションで、汎用 Schur 因子分解から得た直交 / ユニタリ行列  $Q$  を入力行列  $Q_I$  の後に掛け、直交 / ユニタリ行列  $Z$  を入力行列  $Z_I$  の後に掛けられる。 $Q_I$  と  $Z_I$  が、?gghrd により行列ペア  $(A, B)$  を汎用上 Hessenberg 形式に縮退した直交 / ユニタリ行列であれば、出力行列  $Q_I Q$  と  $Z_I Z$  は、 $(A, B)$  の汎用 Schur 因子分解で得た直交 / ユニタリ因子である。

$$A = (Q_I Q) S (Z_I Z)^H, \quad B = (Q_I Q) P (Z_I Z)^H.$$

オーバーフローを避けるために、行列ペア  $(H, T)$  の固有値 (すなわち、 $(A, B)$  の固有値) は、値のペア  $(\alpha, \beta)$  として計算される。chgeqz/zhgeqz の場合、 $\alpha$  と  $\beta$  は複素数である。shgeqz/dhgeqz の場合、 $\alpha$  は複素数、 $\beta$  は実数である。 $\beta$  がゼロでなければ、 $\lambda = \alpha / \beta$  は、次の汎用非対称固有値問題 (GNEP) の固有値である。

$$Ax = \lambda Bx$$

また、 $\alpha$  がゼロでなければ、 $\mu = \beta / \alpha$  は、次のように別の形式で表された GNEP の固有値である。

$$\mu Ay = By$$

実固有値 (実数型の場合)、または  $i$  番目の固有値を表す  $\alpha$  と  $\beta$  (複素数型の場合) は、次のように汎用 Schur 形式から直接読み込める。

$$\alpha = S(i, i), \quad \beta = P(i, i).$$

## 入力パラメータ

<code>job</code>	CHARACTER*1。実行する演算を指定する。'E' または 'S' でなければならない。 <code>job</code> ='E' の場合、固有値のみを計算する。 <code>job</code> ='S' の場合、固有値と Schur 形式を計算する。
<code>compq</code>	CHARACTER*1。'N'、'I'、または 'V' のいずれかでなければならない。 <code>compq</code> ='N' の場合、左 Schur ベクトル ( $q$ ) は計算されない。 <code>compq</code> ='I' の場合、 $q$ は単位行列に初期化され、 $(H, T)$ の左 Schur ベクトルからなる行列が返される。 <code>compq</code> ='V' の場合、 $q$ には、呼び出し時に、直交 / ユニタリ行列 $Q_I$ が格納されていなければならない。終了時に、積 $Q_I Q$ が返される。
<code>compz</code>	CHARACTER*1。'N'、'I'、または 'V' のいずれかでなければならない。 <code>compz</code> ='N' の場合、左 Schur ベクトル ( $q$ ) は計算されない。 <code>compz</code> ='I' の場合、 $z$ は単位行列に初期化され、 $(H, T)$ の右 Schur ベクトルからなる行列が返される。 <code>compz</code> ='V' の場合、 $z$ には、呼び出し時に、直交 / ユニタリ行列 $Z_I$ が格納されていなければならない。終了時に、積 $Z_I Z$ が返される。
<code>n</code>	INTEGER。行列 $H$ 、 $T$ 、 $Q$ 、および $Z$ の次数 ( $n \geq 0$ )。

<i>ilo, ihi</i>	<p>INTEGER。 <i>ilo</i> と <i>ihi</i> は、<math>H</math> のどの行と列が Hessenberg 形式かをマークする。<math>H</math> において、行と列が <math>1:ilo-1</math> と <math>ihi+1:n</math> の範囲は、すでに上三角形式になっているものとする。次の制約がある。</p> <p><math>n &gt; 0</math> の場合、<math>1 \leq ilo \leq ihi \leq n</math>。</p> <p><math>n = 0</math> の場合、<math>ilo = 1</math> かつ <math>ihi = 0</math>。</p>
<i>h, t, q, z, work</i>	<p>REAL (shgeqz の場合 )</p> <p>DOUBLE PRECISION (dhgeqz の場合 )</p> <p>COMPLEX (chgeqz の場合 )</p> <p>DOUBLE COMPLEX (zhgeqz の場合 )。</p> <p>配列 :</p> <p>呼び出し時に、<math>h(ldh, *)</math> には、<math>n \times n</math> の上 Hessenberg 行列 <math>H</math> を格納する。</p> <p><math>h</math> の第 2 次元は、<math>\max(1, n)</math> 以上でなければならない。</p> <p>呼び出し時に、<math>t(ldt, *)</math> には、<math>n \times n</math> の上三角行列 <math>T</math> を格納する。</p> <p><math>t</math> の第 2 次元は、<math>\max(1, n)</math> 以上でなければならない。</p> <p><math>q(ldq, *)</math>:</p> <p>呼び出し時に、<math>compq = 'V'</math> の場合、<math>(A, B)</math> を汎用 Hessenberg 形式に縮退するのに使用した直交 / ユニタリ行列 <math>Q_l</math> を格納する。</p> <p><math>compq = 'N'</math> の場合、<math>q</math> は参照されない。</p> <p><math>q</math> の第 2 次元は、<math>\max(1, n)</math> 以上でなければならない。</p> <p><math>z(ldz, *)</math>:</p> <p>呼び出し時に、<math>compz = 'V'</math> の場合、<math>(A, B)</math> を汎用 Hessenberg 形式に縮退するのに使用した直交 / ユニタリ行列 <math>Z_l</math> を格納する。</p> <p><math>compz = 'N'</math> の場合、<math>z</math> は参照されない。</p> <p><math>z</math> の第 2 次元は、<math>\max(1, n)</math> 以上でなければならない。</p> <p><math>work(lwork)</math> は、ワークスペース配列である。</p>
<i>ldh</i>	INTEGER。 $h$ の第 1 次元。 $\max(1, n)$ 以上。
<i>ldt</i>	INTEGER。 $t$ の第 1 次元。 $\max(1, n)$ 以上。
<i>ldq</i>	<p>INTEGER。 <math>q</math> の第 1 次元。</p> <p><math>compq = 'N'</math> の場合、<math>ldq \geq 1</math>。</p> <p><math>compq = 'I'</math> または <math>'V'</math> の場合、<math>ldq \geq \max(1, n)</math>。</p>
<i>ldz</i>	<p>INTEGER。 <math>z</math> の第 1 次元。</p> <p><math>compz = 'N'</math> の場合、<math>ldz \geq 1</math>。</p> <p><math>compz = 'I'</math> または <math>'V'</math> の場合、<math>ldz \geq \max(1, n)</math>。</p>

<i>lwork</i>	INTEGER。配列 <i>work</i> の次元。 $lwork \geq \max(1, n)$ <i>lwork</i> = -1 の場合、ワークスペースのクエリとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリとして返す。xerbla は <i>lwork</i> に関するエラー・メッセージを生成しない。
<i>rwork</i>	REAL (chgeqz の場合 ) DOUBLE PRECISION (zhgeqz の場合 )。 ワークスペース配列、次元は $\max(1, n)$ 以上。複素数型でのみ使用される。

## 出力パラメータ

<i>h</i>	実数型の場合: <i>job</i> = 'S' の場合、終了時に、汎用 Schur 因子分解で得た上準三角行列 <i>S</i> が <i>h</i> に格納される。 $2 \times 2$ の対角ブロック ( 固有値の複素共役ペアに対応 ) が標準形で返され、 $h(i, i) = h(i+1, i+1)$ および $h(i+1, i) * h(i, i+1) < 0$ となる。 <i>job</i> = 'E' の場合、終了時に、 <i>h</i> の対角ブロックは <i>S</i> と同じになるが、 <i>h</i> のその他の部分は設定されない。  複素数型の場合: <i>job</i> = 'S' の場合、終了時に、汎用 Schur 因子分解で得た上三角行列 <i>S</i> が <i>h</i> に格納される。 <i>job</i> = 'E' の場合、終了時に、 <i>h</i> の対角成分は <i>S</i> と同じになるが、 <i>h</i> のその他の部分は設定されない。
<i>t</i>	<i>job</i> = 'S' の場合、終了時に、汎用 Schur 因子分解で得た上三角行列 <i>P</i> が <i>t</i> に格納される。 実数型の場合: <i>S</i> の $2 \times 2$ のブロックに対応する <i>P</i> の $2 \times 2$ の対角ブロックが、正値対角形式に縮退される。すなわち、 $h(j+1, j)$ がゼロでない場合、 $t(j+1, j) = t(j, j+1) = 0$ 、 $t(j, j)$ と $t(j+1, j+1)$ が正になる。  <i>job</i> = 'E' の場合、終了時に、 <i>t</i> の対角ブロックは <i>P</i> と同じになるが、 <i>t</i> のその他の部分は設定されない。  複素数型の場合: <i>job</i> = 'E' の場合、終了時に、 <i>t</i> の対角成分は <i>P</i> と同じになるが、 <i>t</i> のその他の部分は設定されない。



<i>alphar, alphai</i>	<p>REAL (shgeqz の場合 )  DOUBLE PRECISION (dhgeqz の場合 )。  配列、次元は <math>\max(1, n)</math> 以上。  汎用非対称固有値問題において固有値を定義する各スカラ <i>alpha</i> の実部と虚部。</p> <p><i>alphai(j)</i> がゼロの場合、<i>j</i> 番目の固有値は実数である。正の場合、<i>j</i> 番目と (<i>j</i>+1) 番目の固有値は、<math>\text{alphai}(j+1) = -\text{alphai}(j)</math> であるような複素共役ペアである。</p>
<i>alpha</i>	<p>COMPLEX (chgeqz の場合 )  DOUBLE COMPLEX (zhgeqz の場合 )。  配列、次元は <math>\max(1, n)</math> 以上。  汎用非対称固有値問題で固有値を定義する複素スカラ <i>alpha</i>。  汎用 Schur 因子分解で、<math>\text{alphai}(i) = S(i, i)</math></p>
<i>beta</i>	<p>REAL (shgeqz の場合 )  DOUBLE PRECISION (dhgeqz の場合 )  COMPLEX (chgeqz の場合 )  DOUBLE COMPLEX (zhgeqz の場合 )。  配列、次元は <math>\max(1, n)</math> 以上。  実数型の場合：  汎用非対称固有値問題で固有値を定義するスカラ <i>beta</i>。  <math>\alpha = (\text{alphar}(j), \text{alphai}(j))</math> と <math>\beta = \text{beta}(j)</math> とで、行列ペア (<i>A</i>, <i>B</i>) の <i>j</i> 番目の固有値が、<math>\lambda = \alpha/\beta</math>、<math>\mu = \beta/\alpha</math> のどちらかの形式で表される。<math>\lambda</math> または <math>\mu</math> がオーバーフローする可能性があるので、一般にその計算は行わない。</p> <p>複素数型の場合：  汎用非対称固有値問題で固有値を定義する、非負の実スカラ <i>beta</i>。汎用 Schur 因子分解で、<math>\text{beta}(i) = P(i, i)</math>。  <math>\alpha = \text{alpha}(j)</math> と <math>\beta = \text{beta}(j)</math> とで、行列ペア (<i>A</i>, <i>B</i>) の <i>j</i> 番目の固有値が、<math>\lambda = \alpha/\beta</math>、<math>\mu = \beta/\alpha</math> のどちらかの形式で表される。<math>\lambda</math> または <math>\mu</math> がオーバーフローする可能性があるので、一般にその計算は行わない。</p>
<i>q</i>	<p>終了時に、<i>compq</i>='I' の場合、<i>q</i> は、行列ペア (<i>H</i>, <i>T</i>) の左 Schur ベクトルからなる直交 / ユニタリ行列で上書きされる。<i>compq</i>='V' の場合、行列ペア (<i>A</i>, <i>B</i>) の左 Schur ベクトルからなる直交 / ユニタリ行列で <i>q</i> は上書きされる。</p>

<i>z</i>	終了時に、 <i>compz</i> ='I' の場合、 <i>z</i> は、行列ペア ( <i>H</i> , <i>T</i> ) の右 Schur ベクトルからなる直交 / ユニタリ行列で上書きされる。 <i>compz</i> ='V' の場合、行列ペア ( <i>A</i> , <i>B</i> ) の右 Schur ベクトルからなる直交 / ユニタリ行列で <i>z</i> は上書きされる。
<i>work</i> (1)	<i>info</i> ≥ 0 の場合、終了時に、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が <i>work</i> (1) に格納される。以降の実行では、この <i>lwork</i> 値を使用する。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正であったことを示す。 <i>info</i> = 1, ..., <i>n</i> の場合、 <i>QZ</i> 反復で収束しなかった。 ( <i>H</i> , <i>T</i> ) は Schur 形式ではないが、 <i>alphar</i> ( <i>i</i> )、 <i>alphai</i> ( <i>i</i> ) (実数型の場合)、 <i>alpha</i> ( <i>i</i> ) (複素数型の場合)、 <i>beta</i> ( <i>i</i> ) は正しい値である (ただし、 <i>i</i> = <i>info</i> +1,..., <i>n</i> )  <i>info</i> = <i>n</i> +1, ..., 2 <i>n</i> の場合、シフト計算に失敗した。 ( <i>H</i> , <i>T</i> ) は Schur 形式ではないが、 <i>alphar</i> ( <i>i</i> )、 <i>alphai</i> ( <i>i</i> ) (実数型の場合)、 <i>alpha</i> ( <i>i</i> ) (複素数型の場合)、 <i>beta</i> ( <i>i</i> ) は正しい値である (ただし、 <i>i</i> = <i>info</i> - <i>n</i> +1,..., <i>n</i> )。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *hgeqz* のインターフェイスの詳細を以下に示す。

<i>h</i>	サイズ ( <i>n</i> , <i>n</i> ) の行列 <i>H</i> を格納する。
<i>t</i>	サイズ ( <i>n</i> , <i>n</i> ) の行列 <i>T</i> を格納する。
<i>alphar</i>	長さ ( <i>n</i> ) のベクトルを格納する。実数型でのみ使用される。
<i>alphai</i>	長さ ( <i>n</i> ) のベクトルを格納する。実数型でのみ使用される。
<i>alpha</i>	長さ ( <i>n</i> ) のベクトルを格納する。複素数型でのみ使用される。
<i>beta</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>q</i>	サイズ ( <i>n</i> , <i>n</i> ) の行列 <i>Q</i> を格納する。
<i>z</i>	サイズ ( <i>n</i> , <i>n</i> ) の行列 <i>Z</i> を格納する。

<i>ilo</i>	この引数のデフォルト値は、 <i>ilo</i> = 1 である。
<i>ihi</i>	この引数のデフォルト値は、 <i>ihi</i> = <i>n</i> である。
<i>job</i>	'E' または 'S' でなければならない。デフォルト値は 'E'。
<i>compq</i>	省略された場合、この引数は、引数 <i>q</i> の存在に基づいて以下のように復元される。 <i>compq</i> = 'I' ( <i>q</i> が存在する場合)、 <i>compq</i> = 'N' ( <i>q</i> が省略された場合)。 存在する場合、 <i>compz</i> は 'I' または 'V' と等しくなければならず、引数 <i>q</i> も存在しなければならない。 <i>compz</i> が存在し、 <i>q</i> が省略された場合、エラー条件がセットされる。
<i>compz</i>	省略された場合、この引数は、引数 <i>z</i> の存在に基づいて以下のように復元される。 <i>compz</i> = 'I' ( <i>z</i> が存在する場合)、 <i>compz</i> = 'N' ( <i>z</i> が省略された場合)。 存在する場合、 <i>compz</i> は 'I' または 'V' と等しくなければならず、引数 <i>z</i> も存在しなければならない。 <i>compz</i> が存在し、 <i>z</i> が省略された場合、エラー条件がセットされる。

## ?tgevc

上三角行列ペアの右と左汎用固有ベクトルの一部または全部を求める。

### 構文

#### Fortran 77:

```
call stgevc(side, howmny, select, n, s, lds, p, ldp, vl, ldvl, vr,
            ldvr, mm, m, work, info)
call dtgevc(side, howmny, select, n, s, lds, p, ldp, vl, ldvl, vr,
            ldvr, mm, m, work, info)
call ctgevc(side, howmny, select, n, s, lds, p, ldp, vl, ldvl, vr,
            ldvr, mm, m, work, rwork, info)
call ztgevc(side, howmny, select, n, s, lds, p, ldp, vl, ldvl, vr,
            ldvr, mm, m, work, rwork, info)
```

## Fortran 95:

```
call tgevc(s, p [,howmny] [,select] [,v1] [,vr] [,m] [,info])
```

### 説明

このルーチンは、実 / 複素行列ペア  $(S, P)$  の右と左固有ベクトルの一部または全部を求める。 $S$  は準三角行列 (実数型の場合) または上三角行列 (複素数型の場合)、 $P$  は上三角行列である。

このタイプの行列ペアは、実 / 複素行列ペア  $(A, B)$  の汎用 Schur 因子分解で得られる。

$$A = Q S Z^H, \quad B = Q P Z^H$$

?gghrd と ?hgeqz によって計算される。

固有値  $w$  に対応する、 $(S, P)$  の右固有ベクトル  $x$  と左固有ベクトル  $y$  は次のように定義される。

$$Sx = wPx, \quad y^H S = w y^H P$$

固有値は、このルーチンに入力されず、 $S$  と  $P$  の対角ブロックまたは対角成分から直接計算される。

このルーチンは、 $(S, P)$  の右と左の固有ベクトルからなる行列  $X$  と  $Y$ 、または積  $ZX$  と  $QY$  ( $Z$  と  $Q$  は入力行列) を返す。

$Q$  と  $Z$  が行列ペア  $(A, B)$  の汎用 Schur 因子分解で得た直交 / ユニタリ因子であれば、 $ZX$  と  $QY$  は  $(A, B)$  の右と左の固有ベクトルからなる行列である。

### 入力パラメータ

<i>side</i>	CHARACTER*1。'R'、'L'、または 'B' でなければならない。 <i>side</i> ='R' の場合、右固有ベクトルのみを計算する。 <i>side</i> ='L' の場合、左固有ベクトルのみを計算する。 <i>side</i> ='B' の場合、右と左の固有ベクトルを両方計算する。
<i>howmny</i>	CHARACTER*1。'A'、'B'、または 'S' でなければならない。 <i>howmny</i> ='A' の場合、右と左の固有ベクトルをすべて計算する。 <i>howmny</i> ='B' の場合、 <i>vr</i> と <i>v1</i> 内の行列を逆変換して、右と左の固有ベクトルをすべて計算する。 <i>howmny</i> ='S' の場合、論理配列 <i>select</i> で指定した右と左の固有ベクトルのみを計算する。
<i>select</i>	LOGICAL。 配列、次元は $\max(1, n)$ 以上。 <i>howmny</i> ='S' の場合、 <i>select</i> は、計算する固有値を $w$ 定する。

*howmny*='A' または 'B' の場合、*select* は参照されない。

実数型の場合：

$\omega_j$  が実固有値の場合、*select*(*j*) が .TRUE. であれば、それに対応する実固有ベクトルが計算される。

$\omega_j$  と  $\omega_{j+1}$  が複素固有値の実部と虚部の場合、*select*(*j*) または *select*(*j*+1) が .TRUE. であれば、それに対応する複素固有ベクトルが計算され、終了時に、*select*(*j*) は .TRUE.、*select*(*j*+1) は .FALSE. に設定される。

複素数型の場合：

*select*(*j*) が .TRUE. であれば、*j* 番目の固有値に対応する固有ベクトルが計算される。

*n* INTEGER。行列 *A* と *B* の次数 ( $n \geq 0$ )。

*s,p,vl,vr,work* REAL (stgevc の場合 )  
DOUBLE PRECISION (dtgevc の場合 )  
COMPLEX (ctgevc の場合 )  
DOUBLE COMPLEX (ztgevc の場合 )。  
配列：

*s*(*lds*,\*) には、?hgeqz による汎用 Schur 因子分解で得た行列 *S* を格納する。この行列は、上準三角行列 (実数型の場合) または三角行列 (複素数型の場合) である。

*s* の第 2 次元は、 $\max(1, n)$  以上でなければならない。

*p*(*ldp*,\*) には、?hgeqz

による汎用 Schur 因子分解で得た行列 *P* を格納する。実数型の場合、*S* の  $2 \times 2$  のブロックに対応する *P* の  $2 \times 2$  の対角ブロックが、正値対角形式でなければならない。

複素数型の場合、*P* は実対角成分を持たなければならない。

*p* の第 2 次元は、 $\max(1, n)$  以上でなければならない。

*side*='L' または 'B' で *howmny*='B' の場合、

*vl*(*ldvl*,\*) には、 $n \times n$  の行列 *Q* (通常は、?hgeqz から返された、左 Schur ベクトルからなる直交 / ユニタリ行列 *Q*) が格納されていなければならない。*vl* の第 2 次元は、 $\max(1, mm)$  以上でなければならない。

*side*='R' の場合、*vl* は参照されない。

*side*='R' または 'B' で *howmny*='B' の場合、

*vr*(*ldvr*,\*) には、 $n \times n$  の行列 *Z* (通常は、?hgeqz から返された、右 Schur ベクトルからなる直交 / ユニタリ行列 *Z*) が格納されていなければならない。*vr* の第 2 次元は、 $\max(1, mm)$  以上でなければならない。

*side*='L' の場合、*vr* は参照されない。

$work(*)$  は、ワークスペース配列である。  
次元は  $\max(1, 6*n)$  以上 (実数型の場合) または  $\max(1, 2*n)$  以上 (複素数型の場合)。

$lds$	INTEGER。 $s$ の第 1 次元。 $\max(1, n)$ 以上。
$ldp$	INTEGER。 $p$ の第 1 次元。 $\max(1, n)$ 以上。
$ldvl$	INTEGER。 $vl$ の第 1 次元。 $side = 'L'$ または $'B'$ の場合、 $ldvl \geq \max(1, n)$ 。 $side = 'R'$ の場合、 $ldvl \geq 1$ 。
$ldvr$	INTEGER。 $vr$ の第 1 次元。 $side = 'R'$ または $'B'$ の場合、 $ldvr \geq \max(1, n)$ 。 $side = 'L'$ の場合、 $ldvr \geq 1$ 。
$mm$	INTEGER。 配列 $vl$ と $vr$ の列数 ( $mm \geq m$ )。
$rwork$	REAL (ctgevc の場合) DOUBLE PRECISION (ztgevc の場合)。 ワークスペース配列、次元は $\max(1, 2*n)$ 以上。複素数型でのみ使用される。

## 出力パラメータ

$vl$	終了時に、 $side = 'L'$ または $'B'$ の場合、 $vl$ に次の値が格納される。 $howmny = 'A'$ の場合、 $(S, P)$ の左固有ベクトルからなる行列 $Y$ 。 $howmny = 'B'$ の場合、 行列 $QY$ 。 $howmny = 'S'$ の場合、 $select$ で指定した $(S, P)$ の左固有ベクトルが、固有値と同じ順に、 $vl$ の連続した列に格納される。 実数型の場合： 複素固有値に対応する複素固有ベクトルが、連続した 2 つの列に格納される (最初の列に実部、2 番目の列に虚部)。
$vr$	終了時に、 $side = 'R'$ または $'B'$ の場合、 $vr$ に次の値が格納される。 $howmny = 'A'$ の場合、 $(S, P)$ の右固有ベクトルからなる行列 $X$ 。 $howmny = 'B'$ の場合、 行列 $ZX$ 。 $howmny = 'S'$ の場合、 $select$ で指定した $(S, P)$ の右固有ベクトルが、固有値と同じ順に、 $vr$ の連続した列に格納される。 実数型の場合： 複素固有値に対応する複素固有ベクトルが、連続した 2 つの列に格納される (最初の列に実部、2 番目の列に虚部)。

<i>m</i>	INTEGER。実際に固有値の格納に使用された配列 <i>v1</i> と <i>vr</i> の列数。 <i>howmny</i> = 'A' または 'B' の場合、 <i>m</i> には <i>n</i> が設定される。 実数型の場合： 選択された実固有ベクトル 1 つに対して 1 列、選択された複素固有ベクトル 1 つに対して 2 列が必要である。 複素数型の場合： 選択された固有ベクトル 1 つに対して 1 列必要である。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正であったことを示す。 実数型の場合： <i>info</i> = <i>i</i> > 0 の場合、 $2 \times 2$ のブロック ( <i>i</i> : <i>i</i> +1) に複素固有値が格納されていない。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *tgevc* のインターフェイスの詳細を以下に示す。

<i>s</i>	サイズ ( <i>n</i> , <i>n</i> ) の行列 <i>S</i> を格納する。
<i>p</i>	サイズ ( <i>n</i> , <i>n</i> ) の行列 <i>P</i> を格納する。
<i>select</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>v1</i>	サイズ ( <i>n</i> , <i>mm</i> ) の行列 <i>VL</i> を格納する。
<i>vr</i>	サイズ ( <i>n</i> , <i>mm</i> ) の行列 <i>VR</i> を格納する。
<i>side</i>	引数 <i>v1</i> および <i>vr</i> の存在に基づいて以下のように復元される。 <i>side</i> = 'B' ( <i>v1</i> と <i>vr</i> の両方が存在する場合)、 <i>side</i> = 'L' ( <i>v1</i> が存在し、 <i>vr</i> が省略された場合)、 <i>side</i> = 'R' ( <i>v1</i> が省略され、 <i>vr</i> が存在する場合)。 <i>v1</i> と <i>vr</i> の両方が省略された場合、エラー条件がセットされる。
<i>howmny</i>	省略された場合、この引数は、引数 <i>select</i> の存在に基づいて以下のように復元される。 <i>howmny</i> = 'S' ( <i>select</i> が存在する場合)、 <i>howmny</i> = 'A' ( <i>select</i> が省略された場合)。 存在する場合、 <i>howmny</i> は 'A' または 'B' と等しくなければならず、

引数 *select* も存在しなければならない。  
*howmny* と *select* の両方が存在する場合、エラー条件がセットされる。

---

### ?tgexc

行列ペア  $(A, B)$  において、ある対角ブロック  
が別の行インデックスに移動するように、  
 $(A, B)$  の汎用 Schur 分解の順序を変更する。

---

#### 構文

##### Fortran 77:

```
call stgexc(wantq, wantz, n, a, lda, b, ldb, q, ldq, z, ldz,  
           ifst, ilst, work, lwork, info)  
call dtgexc(wantq, wantz, n, a, lda, b, ldb, q, ldq, z, ldz,  
           ifst, ilst, work, lwork, info)  
call ctgexc(wantq, wantz, n, a, lda, b, ldb, q, ldq, z, ldz,  
           ifst, ilst, info)  
call ztgexc(wantq, wantz, n, a, lda, b, ldb, q, ldq, z, ldz,  
           ifst, ilst, info)
```

##### Fortran 95:

```
call tgexc(a, b [,ifst] [,ilst] [,z] [,q] [,info])
```

#### 説明

このルーチンは、直交 / ユニタリ等価変換を使用して、実 / 複素行列ペア  $(A, B)$  の汎用実 Schur/Schur 分解の順序を変更する。

$$(A, B) = Q (A, B) Z^H$$

このとき、 $(A, B)$  において、行インデックス *ifst* の対角ブロックが行 *ilst* に移動するように変更する。

行列ペア  $(A, B)$  は、汎用実 Schur/Schur 標準形 ([?qges](#) から返される形式)、すなわち、 $A$  は  $1 \times 1$  と  $2 \times 2$  の対角ブロックを持つブロック上三角行列、 $B$  は上三角行列でなければならない。

オプションで、汎用 Schur ベクトルからなる行列  $Q$  と  $Z$  を更新できる。



$$Q(\text{in}) * A(\text{in}) * Z(\text{in})' = Q(\text{out}) * A(\text{out}) * Z(\text{out})'$$

$$Q(\text{in}) * B(\text{in}) * Z(\text{in})' = Q(\text{out}) * B(\text{out}) * Z(\text{out})'$$

## 入力パラメータ

`wantq, wantz` LOGICAL。  
`wantq=.TRUE.` の場合、左側の変換行列  $Q$  を更新する。  
`wantq=.FALSE.` の場合、 $Q$  を更新しない。  
`wantz=.TRUE.` の場合、右側の変換行列  $Z$  を更新する。  
`wantz=.FALSE.` の場合、 $Z$  を更新しない。

`n` INTEGER。行列  $A$  と  $B$  の次数 ( $n \geq 0$ )。

`a, b, q, z` REAL (stgexc の場合 )  
DOUBLE PRECISION (dtgexc の場合 )  
COMPLEX (ctgexc の場合 )  
DOUBLE COMPLEX (ztgexc の場合 )。  
配列：  
`a(lda,*)` には、行列  $A$  を格納する。  
 $a$  の第 2 次元は、 $\max(1, n)$  以上でなければならない。  
`b(ldb,*)` には、行列  $B$  を格納する。  
 $b$  の第 2 次元は、 $\max(1, n)$  以上でなければならない。  
`q(ldq,*)`  
`wantq=.FALSE.` の場合、 $q$  は参照されない。  
`wantq=.TRUE.` の場合、 $q$  に直交 / ユニタリ行列  $Q$  が格納されていなければならない。  
 $q$  の第 2 次元は、 $\max(1, n)$  以上でなければならない。  
`z(ldz,*)`  
`wantz=.FALSE.` の場合、 $z$  は参照されない。  
`wantz=.TRUE.` の場合、 $z$  に直交 / ユニタリ行列  $Z$  が格納されていなければならない。  
 $z$  の第 2 次元は、 $\max(1, n)$  以上でなければならない。

`lda` INTEGER。  $a$  の第 1 次元。  $\max(1, n)$  以上。

`ldb` INTEGER。  $b$  の第 1 次元。  $\max(1, n)$  以上。

`ldq` INTEGER。  $q$  の第 1 次元。  
`wantq=.FALSE.` の場合、 $ldq \geq 1$ 。  
`wantq=.TRUE.` の場合、 $ldq \geq \max(1, n)$ 。

<i>ldz</i>	INTEGER。 <i>z</i> の第 1 次元。 <i>wantz</i> = .FALSE. の場合、 $ldz \geq 1$ 。 <i>wantz</i> = .TRUE. の場合、 $ldz \geq \max(1, n)$ 。
<i>ifst, ilst</i>	INTEGER。 ( <i>A, B</i> ) の対角ブロックの順序変更を指定する。行インデックス <i>ifst</i> のブロックが、隣接ブロックの連続的交換により、行 <i>ilst</i> に移動される。次の制約がある。 $1 \leq ifst, ilst \leq n$ 。
<i>work</i>	REAL ( <i>stgexc</i> の場合 ) DOUBLE PRECISION ( <i>dtgexc</i> の場合 )。 ワークスペース配列、次元は ( <i>lwork</i> )。実数型でのみ使用される。
<i>lwork</i>	INTEGER。 <i>work</i> の次元。 $4n+16$ 以上でなければならない。 <i>lwork</i> = -1 の場合、ワークスペースのクエリとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリとして返す。 <i>xerbla</i> は <i>lwork</i> に関するエラー・メッセージを生成しない。

### 出力パラメータ

<i>a, b</i>	更新された行列 <i>A</i> と <i>B</i> で上書きされる。
<i>ifst, ilst</i>	実数型の場合にのみ上書きされる。 呼び出し時に、 <i>ifst</i> が $2 \times 2$ のブロックの 2 行目を指している場合は、先頭行を指すように変更される。 <i>ilst</i> は常に、最終位置 (入力値から $\pm 1$ だけ変動している場合もある) のブロックの先頭行を指す。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正であったことを示す。 <i>info</i> = 1 の場合、変換後の行列ペア ( <i>A, B</i> ) が汎用 Schur 形式から遠すぎる。すなわち、この問題は悪条件である。 ( <i>A, B</i> ) は部分的に順序変更されている可能性があり、 <i>ilst</i> は移動中のブロックの現在位置の第 1 行を指している。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *tgexc* のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ ( <i>n, n</i> ) の行列 <i>A</i> を格納する。
----------	---

<i>b</i>	サイズ $(n,n)$ の行列 <i>B</i> を格納する。
<i>z</i>	サイズ $(n,n)$ の行列 <i>Z</i> を格納する。
<i>q</i>	サイズ $(n,n)$ の行列 <i>Q</i> を格納する。
<i>wantq</i>	引数 <i>q</i> の存在に基づいて以下のように復元される。 <i>wantq</i> = .TRUE ( <i>q</i> が存在する場合)、 <i>wantq</i> = .FALSE ( <i>q</i> が省略された場合)。
<i>wantz</i>	引数 <i>z</i> の存在に基づいて以下のように復元される。 <i>wantz</i> = .TRUE ( <i>z</i> が存在する場合)、 <i>wantz</i> = .FALSE ( <i>z</i> が省略された場合)。

---

## ?tgsen

選択した固有値クラスが行列ペア  $(A, B)$  の  
主対角ブロックに現れるように、 $(A, B)$  の  
汎用 Schur 分解の順序を変更する。

---

### 構文

#### Fortran 77:

```
call stgsen(ijob, wantq, wantz, select, n, a, lda, b, ldb, alphas,  
           alphai, beta, q, ldq, z, ldz, m, pl, pr, dif, work,  
           lwork, iwork, liwork, info)  
call dtgsen(ijob, wantq, wantz, select, n, a, lda, b, ldb, alphas,  
           alphai, beta, q, ldq, z, ldz, m, pl, pr, dif, work,  
           lwork, iwork, liwork, info)  
call ctgsen(ijob, wantq, wantz, select, n, a, lda, b, ldb, alpha,  
           beta, q, ldq, z, ldz, m, pl, pr, dif, work,  
           lwork, iwork, liwork, info)  
call ztgsen(ijob, wantq, wantz, select, n, a, lda, b, ldb, alpha,  
           beta, q, ldq, z, ldz, m, pl, pr, dif, work,  
           lwork, iwork, liwork, info)
```

## Fortran 95:

```
call tgsen(a, b, select [,alphar] [,alphai] [,beta] [,ijob] [,q] [,z] [,p1]
[,pr] [,dif] [,m] [,info])
call tgsen(a, b, select [,alpha] [,beta] [,ijob] [,q] [,z] [,p1] [,pr] [,dif]
[,m] [,info])
```

## 説明

このルーチンは、実/複素行列ペア  $(A, B)$  の汎用実 Schur/Schur 分解の順序を変更し (直交/ユニタリ等価変換  $Q' * (A, B) * Z$  を使用)、選択した固有値クラスが、行列ペア  $(A, B)$  の主対角ブロックに現れるようにする。

$Q$  と  $Z$  の先頭列は、対応する左と右の固有空間 (収縮部分空間) の正規直交/ユニタリ基底を形成する。

$(A, B)$  は、汎用実 Schur/Schur 標準形 ([?qges](#) から返される形式)、すなわち  $A$  と  $B$  がともに上三角行列でなければならない。

?tgsen は、順序変更後の

行列ペア  $(A, B)$

$\omega_j = (\text{alphar}(j) + \text{alphai}(j)*i)/\text{beta}(j)$  (実数型の場合)

$\omega_j = \text{alpha}(j)/\text{beta}(j)$  (複素数型の場合)

の汎用固有値も計算する。

このルーチンは、オプションで、固有値と固有空間の条件数の逆数を見積もる。すなわち、 $\text{Difu}[(A_{11}, B_{11}), (A_{22}, B_{22})]$  と  $\text{Difl}[(A_{11}, B_{11}), (A_{22}, B_{22})]$  を計算する。これは、行列ペア  $(A_{11}, B_{11})$  と  $(A_{22}, B_{22})$  (選択したクラスタに対応するものとそれ以外の固有値に対応するもの) の差と、(1,1) ブロック内の選択したクラスタに対する左と右固有空間への「投影」のノルムの差である。

## 入力パラメータ

**ijob** INTEGER。固有値クラスタに関する条件数 ( $p1$  と  $pr$ )、または収縮部分空間  $\text{Difu}$  と  $\text{Difl}$  を求めるかどうか指定する。  
*ijob* = 0 の場合、*select* に対する順序変更のみ。  
*ijob* = 1 の場合、選択したクラスタに関する左と右の固有空間への「投影」のノルムの逆数 ( $p1$  と  $pr$ )。  
*ijob* = 2 の場合、F- ノルムベースの推定値を使用して、 $\text{Difu}$  と  $\text{Difl}$  の上限値を計算する (*dif* (1:2))。  
*ijob* = 3 の場合、1- ノルムベースの推定値を使用して、 $\text{Difu}$  と  $\text{Difl}$  の推定値を計算する (*dif* (1:2))。このオプションは、*ijob* = 2 の 5 倍の負荷がかかる。  
*ijob* = 4 の場合、 $p1$ 、 $pr$ 、*dif* を計算する (上記のオプション 0、1、2

の組み合わせ)。すべてを計算する効率のよいオプション。  
 $ijob=5$  の場合、 $p1$ 、 $pr$ 、 $dif$  を計算する (上記のオプション 0、1、3 の組み合わせ)。

$wantq, wantz$  LOGICAL。  
 $wantq=.TRUE.$  の場合、左側の変換行列  $Q$  を更新する。  
 $wantq=.FALSE.$  の場合、 $Q$  を更新しない。  
 $wantz=.TRUE.$  の場合、右側の変換行列  $Z$  を更新する。  
 $wantz=.FALSE.$  の場合、 $Z$  を更新しない。

$select$  LOGICAL。  
 配列、次元は  $\max(1, n)$  以上。  
 選択するクラスタに属する固有値を指定する。  
 固有値  $\omega_j$  を選択するには、 $select(j)$  を  $.TRUE.$  に設定する。実数型の場合：複素共役固有値のペア  $\omega_j$  と  $\omega_{j+1}$  ( $2 \times 2$  の対角ブロックに対応) を選択するには、 $select(j)$  と  $select(j+1)$  のどちらかを  $.TRUE.$  に設定する。複素共役固有値  $\omega_j$  と  $\omega_{j+1}$  は、ともにクラスタに含めるか、ともにクラスタから除外する必要がある。

$n$  INTEGER。行列  $A$  と  $B$  の次数 ( $n \geq 0$ )。

$a, b, q, z, work$  REAL (stgsen の場合)  
 DOUBLE PRECISION (dtgsen の場合)  
 COMPLEX (ctgsen の場合)  
 DOUBLE COMPLEX (ztgsen の場合)。  
 配列：  
 $a(lda, *)$  には、行列  $A$  を格納する。  
 実数型の場合： $A$  は上準三角行列で、 $(A, B)$  は汎用実 Schur 標準形。  
 複素数型の場合： $A$  は上三角行列で、汎用 Schur 標準形。  
 $a$  の第 2 次元は、 $\max(1, n)$  以上でなければならない。

$b(lb, *)$  には、行列  $B$  を格納する。  
 実数型の場合： $B$  は上三角行列で、 $(A, B)$  は汎用実 Schur 標準形。  
 複素数型の場合： $B$  は上三角行列で、汎用 Schur 標準形。  
 $b$  の第 2 次元は、 $\max(1, n)$  以上でなければならない。

$q(ldq, *)$   
 $wantq=.TRUE.$  の場合、 $q$  は  $n \times n$  の行列。  
 $wantq=.FALSE.$  の場合、 $q$  は参照されない。  
 $q$  の第 2 次元は、 $\max(1, n)$  以上でなければならない。

	$z(ldz, *)$ $wantz = .TRUE.$ の場合、 $z$ は $n \times n$ の行列。 $wantz = .FALSE.$ の場合、 $z$ は参照されない。 $z$ の第 2 次元は、 $\max(1, n)$ 以上でなければならない。  $work(lwork)$ は、ワークスペース配列である。 $ijob = 0$ の場合、 $work$ は参照されない。
$lda$	INTEGER。 $a$ の第 1 次元。 $\max(1, n)$ 以上。
$ldb$	INTEGER。 $b$ の第 1 次元。 $\max(1, n)$ 以上。
$ldq$	INTEGER。 $q$ の第 1 次元。 $ldq \geq 1$ 。 $wantq = .TRUE.$ の場合、 $ldq \geq \max(1, n)$ 。
$ldz$	INTEGER。 $z$ の第 1 次元。 $ldz \geq 1$ 。 $wantz = .TRUE.$ の場合、 $ldz \geq \max(1, n)$ 。
$lwork$	INTEGER。 配列 $work$ の次元。 実数型の場合: $ijob = 1, 2, 4$ の場合、 $lwork \geq \max(4n+16, 2m(n-m))$ 。 $ijob = 3$ または $5$ の場合、 $lwork \geq \max(4n+16, 4m(n-m))$ 。 複素数型の場合: $ijob = 1, 2, 4$ の場合、 $lwork \geq \max(1, 2m(n-m))$ 。 $ijob = 3$ または $5$ の場合、 $lwork \geq \max(1, 4m(n-m))$ 。  $lwork = -1$ の場合、ワークスペースのクエリとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエントリとして返す。xerbla は $lwork$ に関するエラー・メッセージを生成しない。
$iwork$	INTEGER。 ワークスペース配列、次元は $(liwork)$ 。 $ijob = 0$ の場合、 $iwork$ は参照されない。
$liwork$	INTEGER。 配列 $iwork$ の次元。 実数型の場合: $ijob = 1, 2, 4$ の場合、 $liwork \geq n+6$ 。 $ijob = 3$ または $5$ の場合、 $liwork \geq \max(n+6, 2m(n-m))$ 。 複素数型の場合: $ijob = 1, 2, 4$ の場合、 $liwork \geq n+2$ 。 $ijob = 3$ または $5$ の場合、 $liwork \geq \max(n+2, 2m(n-m))$ 。  $liwork = -1$ の場合、ワークスペースのクエリとみなされ、ルーチンは $iwork$ 配列の最適サイズだけを計算し、その値を $iwork$ 配列の最初のエントリとして返す。xerbla は $liwork$ に関するエラー・メッセージを生成しない。

**出力パラメータ**

$a, b$	順序変更された行列 $A$ と $B$ でそれぞれ上書きされる。
$alphar, alphai$	REAL (stgsen の場合 ) DOUBLE PRECISION (dtgsen の場合 )。 配列、次元は $\max(1, n)$ 以上。実数型の汎用固有値を形成する値が格納される。 $\beta$ を参照。
$\alpha$	COMPLEX (ctgsen の場合 ) DOUBLE COMPLEX (ztgsen の場合 )。 配列、次元は $\max(1, n)$ 以上。複素数型の汎用固有値を形成する値が格納される。 $\beta$ を参照。
$\beta$	REAL (stgsen の場合 ) DOUBLE PRECISION (dtgsen の場合 ) COMPLEX (ctgsen の場合 ) DOUBLE COMPLEX (ztgsen の場合 )。 配列、次元は $\max(1, n)$ 以上。 実数型の場合： 終了時に、 $(\alpha_r(j) + \alpha_i(j)*i)/\beta(j)$ 、 $j=1, \dots, n$ は、汎用固有値になる。 $\alpha_r(j) + \alpha_i(j)*i$ と $\beta(j)$ 、 $j=1, \dots, n$ は、 $(A, B)$ の実数汎用 Schur 形式の $2 \times 2$ の対角ブロックを複素数のユニタリ変換を使用して三角形式に更に縮退する場合に生じる複素数 Schur 形式 $(S, T)$ の対角である。 $\alpha_i(j)$ がゼロの場合、 $j$ 番目の固有値が実数であり、正の場合、 $j$ 番目と $(j+1)$ 番目の固有値が複素共役ペアである ( $\alpha_i(j+1)$ は負)。 複素数型の場合： $(A, B)$ を汎用 Schur 形式に縮退させたときの $A$ と $B$ の対角成分。 $\alpha(i)/\beta(i)$ 、 $i=1, \dots, n$ は、汎用固有値。
$q$	$wantq = .TRUE.$ の場合、終了時に、 $(A, B)$ の順序を変更する左直交変換行列が $Q$ に掛けられる。 $Q$ の先頭 $m$ 列は、指定された左固有空間 (収縮部分空間) のペアに対する正規直交基底を形成する。
$z$	$wantz = .TRUE.$ の場合、終了時に、 $(A, B)$ の順序を変更する左直交変換行列が $Z$ に掛けられる。 $Z$ の先頭 $m$ 列は、指定された左固有空間 (収縮部分空間) のペアに対する正規直交基底を形成する。
$m$	INTEGER。左と右固有空間 (収縮部分空間) のペアの次数、 $0 \leq m \leq n$ 。

<i>p1, pr</i>	<p>REAL ( 単精度の場合 )</p> <p>DOUBLE PRECISION ( 倍精度の場合 )。</p> <p><i>ijob</i> = 1、4、5 の場合、<i>p1</i> と <i>pr</i> は、選択したクラスタに関する左と右固有空間への「投影」のノルムの逆数の下限値。</p> <p><math>0 &lt; p1, pr \leq 1</math>。 <i>m</i> = 0 または <i>m</i> = <i>n</i> の場合、<i>p1</i> = <i>pr</i> = 1。</p> <p><i>ijob</i> = 0、2、3 の場合、<i>p1</i> と <i>pr</i> は参照されない。</p>
<i>dif</i>	<p>REAL ( 単精度の場合 )</p> <p>DOUBLE PRECISION ( 倍精度の場合 )。</p> <p>配列、次元は (2)。</p> <p><i>ijob</i> ≥ 2 の場合、<i>dif</i>(1:2) に Difu と Difl の推定値が格納される。</p> <p><i>ijob</i> = 2 または 4 の場合、<i>dif</i>(1:2) は、F- ノルムベースの Difu と Difl の上限値。</p> <p><i>ijob</i> = 3 または 5 の場合、<i>dif</i>(1:2) は、1- ノルムベースの Difu と Difl の推定値。 <i>m</i> = 0 または <i>n</i> の場合、<i>dif</i>(1:2) = F-norm([<i>A</i>, <i>B</i>])。</p> <p><i>ijob</i> = 0 または 1 の場合、<i>dif</i> は参照されない。</p>
<i>work</i> (1)	<i>ijob</i> ≠ 0 かつ <i>info</i> = 0 の場合、終了時に、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が <i>work</i> (1) に格納される。以降の実行では、この <i>lwork</i> 値を使用する。
<i>iwork</i> (1)	<i>ijob</i> ≠ 0 かつ <i>info</i> = 0 の場合、終了時に、最適なパフォーマンスを得るために必要な <i>liwork</i> の最小値が <i>iwork</i> (1) に格納される。以降の実行では、この <i>liwork</i> 値を使用する。
<i>info</i>	<p>INTEGER。</p> <p><i>info</i> = 0 の場合、実行は正常に終了したことを示す。</p> <p><i>info</i> = -<i>i</i> の場合、<i>i</i> 番目のパラメータの値が不正であったことを示す。</p> <p><i>info</i> = 1 の場合、変換後の行列ペア (<i>A</i>, <i>B</i>) が汎用 Schur 形式から遠すぎるため、(<i>A</i>, <i>B</i>) の順序変更に失敗した。すなわち、この問題は悪条件である。(<i>A</i>, <i>B</i>) は部分的に順序変更されている可能性がある。</p> <p><i>dif</i>(*)、<i>p1</i>、<i>pr</i> が要求されている場合、0 が返される。</p>

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *tgsen* のインターフェイスの詳細を以下に示す。

*a*                      サイズ (*n*,*n*) の行列 *A* を格納する。



<i>b</i>	サイズ $(n,n)$ の行列 $B$ を格納する。
<i>select</i>	長さ $(n)$ のベクトルを格納する。
<i>alphan</i>	長さ $(n)$ のベクトルを格納する。実数型でのみ使用される。
<i>alphai</i>	長さ $(n)$ のベクトルを格納する。実数型でのみ使用される。
<i>alpha</i>	長さ $(n)$ のベクトルを格納する。複素数型でのみ使用される。
<i>beta</i>	長さ $(n)$ のベクトルを格納する。
<i>q</i>	サイズ $(n,n)$ の行列 $Q$ を格納する。
<i>z</i>	サイズ $(n,n)$ の行列 $Z$ を格納する。
<i>dif</i>	長さ $(2)$ のベクトルを格納する。
<i>ijob</i>	0、1、2、3、4、または 5 でなければならない。デフォルト値は 0。
<i>wantq</i>	引数 $q$ の存在に基づいて以下のように復元される。 $wantq = .TRUE$ ( $q$ が存在する場合)、 $wantq = .FALSE$ ( $q$ が省略された場合)。
<i>wantz</i>	引数 $z$ の存在に基づいて以下のように復元される。 $wantz = .TRUE$ ( $z$ が存在する場合)、 $wantz = .FALSE$ ( $z$ が省略された場合)。

---

## ?tgsyl

汎用シルベスター式を解く。

---

### 構文

#### Fortran 77:

```
call stgsyl(trans, ijob, m, n, a, lda, b, ldb, c, ldc, d, ldd, e,  
           lde, f, ldf, scale, dif, work, lwork, iwork, info)  
call dtgsyl(trans, ijob, m, n, a, lda, b, ldb, c, ldc, d, ldd, e,  
           lde, f, ldf, scale, dif, work, lwork, iwork, info)  
call ctgsyl(trans, ijob, m, n, a, lda, b, ldb, c, ldc, d, ldd, e,  
           lde, f, ldf, scale, dif, work, lwork, iwork, info)  
call ztgsyl(trans, ijob, m, n, a, lda, b, ldb, c, ldc, d, ldd, e,  
           lde, f, ldf, scale, dif, work, lwork, iwork, info)
```

## Fortran 95:

```
call tgsyl(a, b, c, d, e, f [,ijob] [,trans] [,scale] [,dif] [,info])
```

## 説明

このルーチンは、次の汎用シルベスター式を解く。

$$A R - L B = scale * C$$

$$D R - L E = scale * F$$

$R$  と  $L$  は未知の  $m \times n$  の行列、 $(A, D)$ 、 $(B, E)$ 、 $(C, F)$  はそれぞれ  $m \times m$ 、 $n \times n$ 、 $m \times n$  の与えられた行列で、実 / 複素成分を持っている。 $(A, D)$  と  $(B, E)$  は、汎用実 Schur/Schur 標準形でなければならない。すなわち、 $A$  と  $B$  は上準三角行列 / 上三角行列、 $D$  と  $E$  は上三角行列でなければならない。

$(C, F)$  は、解  $(R, L)$  で上書きされる。 $scale$  ( $0 \leq scale \leq 1$ ) は、オーバーフローを避けるために選択された出力スケール係数である。

上の式を行列表記すると次のようになる。

$Zx = scale * b$  を解く。ただし、 $Z$  は

$$Z = \begin{pmatrix} kron(I_n, A) - kron(B', I_m) \\ kron(I_n, D) - kron(E', I_m) \end{pmatrix}$$

$I_k$  はサイズが  $k$  の単位行列、 $X'$  は  $X$  の転置 / 共役転置行列である。 $kron(X, Y)$  は、行列  $X$  と  $Y$  の Kronecker 積である。

$trans = 'T'$  (実数型) または  $trans = 'C'$  (複素数型) の場合、ルーチン `?tgsyl` は、転置 / 共役転置式  $Z' y = scale * b$  を解く。これは、次の  $R$  と  $L$  を解くのと等価である。

$$A' R + D' L = scale * C$$

$$R B' + L E' = scale * (-F)$$

このケース (`stgsyl/dtgsyl` の場合は  $trans = 'T'$ 、`ctgsyl/ztgsyl` の場合は  $trans = 'C'$ ) を使用して、 $Dif[(A, D), (B, E)]$  の 1- ノルムベースの推定値、すなわち行列ペア  $(A, D)$  と  $(B, E)$  の隔たりを [slacon](#)/[clacon](#) を使用して計算する。

$ijob \geq 1$  の場合、`?tgsyl` は、 $Dif[(A, D), (B, E)]$  の Frobenius ノルムベースの推定値を計算する。すなわち、 $Z$  の最小特異値の逆数に対する下限値の逆数である。これはレベル 3 BLAS アルゴリズムである。

## 入力パラメータ

<i>trans</i>	CHARACTER*1. 'N'、'T'、または 'C' でなければならない。 <i>trans</i> ='N' の場合、汎用シルベスター式を解く。 <i>trans</i> ='T' の場合、「転置」式を解く（実数型の場合のみ）。 <i>trans</i> ='C' の場合、「共役転置」式を解く（複素数型の場合のみ）。
<i>ijob</i>	INTEGER。実行する機能を指定する。 <i>ijob</i> =0 の場合、汎用シルベスター式のみ解く。 <i>ijob</i> =1 の場合、 <i>ijob</i> =0 と <i>ijob</i> =3 を組み合わせた機能を実行する。 <i>ijob</i> =2 の場合、 <i>ijob</i> =0 と <i>ijob</i> =4 を組み合わせた機能を実行する。 <i>ijob</i> =3 の場合、 $\text{Dif}[A, D], (B, E)]$ の推定値のみ求める（先読み法を使用する）。 <i>ijob</i> =4 の場合、 $\text{Dif}[A, D], (B, E)]$ の推定値のみ求める（部分式に対して <a href="#">?gecon</a> を使用する）。 <i>trans</i> ='T' または 'C' の場合、 <i>ijob</i> は参照されない。
<i>m</i>	INTEGER。 行列 <i>A</i> と <i>D</i> の次数、行列 <i>C</i> 、 <i>F</i> 、 <i>R</i> 、および <i>L</i> の行のサイズ。
<i>n</i>	INTEGER。 行列 <i>B</i> と <i>E</i> の次数、行列 <i>C</i> 、 <i>F</i> 、 <i>R</i> 、および <i>L</i> の列のサイズ。
<i>a, b, c, d</i> <i>e, f, work</i>	REAL (stgsyl の場合) DOUBLE PRECISION (dtgsyl の場合) COMPLEX (ctgsyl の場合) DOUBLE COMPLEX (ztgsyl の場合)。  配列： <i>a</i> ( <i>lda</i> ,*) 行列 <i>A</i> （実数型の場合は上準三角行列、複素数型の場合は上三角行列）を格納する。 <i>a</i> の第 2 次元は、 $\max(1, m)$ 以上でなければならない。  <i>b</i> ( <i>ldb</i> ,*) 行列 <i>B</i> （実数型の場合は上準三角行列、複素数型の場合は上三角行列）を格納する。 <i>b</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。  <i>c</i> ( <i>ldc</i> ,*) 汎用シルベスター式の最初の行列式の右辺を格納する ( <i>trans</i> で指定)。 <i>c</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。  <i>d</i> ( <i>ldd</i> ,*) 上三角行列 <i>D</i> を格納する。 <i>d</i> の第 2 次元は、 $\max(1, m)$ 以上でなければならない。  <i>e</i> ( <i>lde</i> ,*) 上三角行列 <i>E</i> を格納する。 <i>e</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。

$f(ldf,*)$  汎用シルベスター式の 2 番目の行列式の右辺を格納する (*trans* で指定)。

$f$  の第 2 次元は、 $\max(1, n)$  以上でなければならない。

$work(lwork)$  は、ワークスペース配列である。 $ijob=0$  の場合、 $work$  は参照されない。

<i>lda</i>	INTEGER。 $a$ の第 1 次元。 $\max(1, m)$ 以上。
<i>ldb</i>	INTEGER。 $b$ の第 1 次元。 $\max(1, n)$ 以上。
<i>ldc</i>	INTEGER。 $c$ の第 1 次元。 $\max(1, m)$ 以上。
<i>ldd</i>	INTEGER。 $d$ の第 1 次元。 $\max(1, m)$ 以上。
<i>lde</i>	INTEGER。 $e$ の第 1 次元。 $\max(1, n)$ 以上。
<i>ldf</i>	INTEGER。 $f$ の第 1 次元。 $\max(1, m)$ 以上。
<i>lwork</i>	INTEGER。 配列 $work$ の次元。 $lwork \geq 1$ 。 $ijob=1$ または $2$ で、 $trans='N'$ の場合、 $lwork \geq 2mn$ 。 $lwork=-1$ の場合、ワークスペースのクエリとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエントリとして返す。 $xerbla$ は $lwork$ に関するエラー・メッセージを生成しない。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $(m+n+6)$ 以上 (実数型の場合) または $(m+n+2)$ 以上 (複素数型の場合)。 $ijob=0$ の場合、 $iwork$ は参照されない。

## 出力パラメータ

$c$	$ijob=0, 1, 2$ の場合、解 $R$ で上書きされる。 $ijob=3$ または $4$ で $trans='N'$ の場合、 $c$ に $R$ (Dif 推定値の計算中に求められた解) が格納される。
$f$	$ijob=0, 1, 2$ の場合、解 $L$ で上書きされる。 $ijob=3$ または $4$ および $trans='N'$ の場合、 $f$ に $L$ (Dif 推定値の計算中に求められた解) が格納される。
<i>dif</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 終了時に、 <i>dif</i> は、Dif 関数の逆数の下限の逆数に設定される。つまり、 <i>dif</i> は、 $\text{Dif}[(A,D), (B,E)] = \text{sigma\_min}(Z)$ の上限に設定される。 ここで、 $Z$ は (2)。 $ijob=0$ 、または $trans='T'$ (実数型) または $trans='C'$ (複素数型) の場合、 <i>dif</i> は設定されない。

<i>scale</i>	REAL ( 単精度の場合 ) DOUBLE PRECISION ( 倍精度の場合 )。 終了時に、 <i>scale</i> には、汎用シルベスター式のスケール係数が格納される。 $0 < scale < 1$ の場合、 <i>c</i> と <i>f</i> にはそれぞれ、わずかに摂動する系の解 <i>R</i> と <i>L</i> が入るが、入力行列 <i>A</i> 、 <i>B</i> 、 <i>D</i> 、および <i>E</i> は変更されない。 <i>scale</i> = 0 の場合、 <i>c</i> と <i>f</i> にはそれぞれ、 $C = F = 0$ の均一な系の解 <i>R</i> と <i>L</i> が入る。通常は、 <i>scale</i> = 1。
<i>work</i> (1)	<i>ijob</i> ≠ 0 かつ <i>info</i> = 0 の場合、終了時に、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が <i>work</i> (1) に格納される。以降の実行では、この <i>lwork</i> 値を使用する。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正であったことを示す。 <i>info</i> > 0 の場合、( <i>A</i> , <i>D</i> ) と ( <i>B</i> , <i>E</i> ) は、同じか近い固有値を持つ。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *tgssyl* のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ ( <i>m</i> , <i>m</i> ) の行列 <i>A</i> を格納する。
<i>b</i>	サイズ ( <i>n</i> , <i>n</i> ) の行列 <i>B</i> を格納する。
<i>c</i>	サイズ ( <i>m</i> , <i>n</i> ) の行列 <i>C</i> を格納する。
<i>d</i>	サイズ ( <i>m</i> , <i>m</i> ) の行列 <i>D</i> を格納する。
<i>e</i>	サイズ ( <i>n</i> , <i>n</i> ) の行列 <i>E</i> を格納する。
<i>f</i>	サイズ ( <i>m</i> , <i>n</i> ) の行列 <i>F</i> を格納する。
<i>ijob</i>	0、1、2、3、または 4 でなければならない。デフォルト値は 0。
<i>trans</i>	'N' または 'T' でなければならない。デフォルト値は 'N'。

### ?tgsna

汎用実 Schur 標準形の行列ペアに関して、指定した固有値と固有ベクトルの条件数の逆数を見積もる。

---

#### 構文

##### Fortran 77:

```
call stgsna(job, howmny, select, n, a, lda, b, ldb, vl, ldvl, vr,  
           ldvr, s, dif, mm, m, work, lwork, iwork, info)  
call dtgsna(job, howmny, select, n, a, lda, b, ldb, vl, ldvl, vr,  
           ldvr, s, dif, mm, m, work, lwork, iwork, info)  
call ctgsna(job, howmny, select, n, a, lda, b, ldb, vl, ldvl, vr,  
           ldvr, s, dif, mm, m, work, lwork, iwork, info)  
call ztgsna(job, howmny, select, n, a, lda, b, ldb, vl, ldvl, vr,  
           ldvr, s, dif, mm, m, work, lwork, iwork, info)
```

##### Fortran 95:

```
call tgsna(a, b [,s] [,dif] [,vl] [,vr] [,select] [,m] [,info])
```

#### 説明

実数型の場合 (stgsna/dtgsna の場合) は、汎用実 Schur 標準形の行列ペア  $(A, B)$ 、または直交行列  $Q$  と  $Z$  を持つ任意の行列ペア  $(QA Z^T, QB Z^T)$  において、指定された固有値と固有ベクトルの条件数の逆数を見積もる。

$(A, B)$  は、汎用実 Schur 形式 ([sgges](#)/[dggss](#) から返される形式) でなければならない。すなわち、 $A$  は  $1 \times 1$  と  $2 \times 2$  の対角ブロックを持つブロック上三角行列、 $B$  は上三角行列でなければならない。

複素数型の場合 (ctgsna/ztgsna の場合) は、行列ペア  $(A, B)$  において、指定された固有値と固有ベクトルの条件数の逆数を見積もる。 $(A, B)$  は汎用 Schur 標準形でなければならない。すなわち、 $A$  と  $B$  はどちらも上三角行列でなければならない。

#### 入力パラメータ

**job** CHARACTER\*1。固有値と固有ベクトルに対して条件数の逆数を計算するかどうかを指定する。  
'E'、'V'、または 'B' のいずれかでなければならない。

	$job='E'$ の場合、固有値のみ ( $s$ を計算する)。 $job='V'$ の場合、固有ベクトルのみ ( $dif$ を計算する)。 $job='B'$ の場合、固有値と固有ベクトルの両方 ( $s$ と $dif$ を計算する)。
<i>howmny</i>	CHARACTER*1。'A' または 'S' でなければならない。 $howmny='A'$ の場合、すべての固有ペアに対して条件数を計算する。 $howmny='S'$ の場合、論理配列 <i>select</i> で指定した固有ペアに対してのみ条件数を計算する。
<i>select</i>	LOGICAL。 配列、次元は $\max(1, n)$ 以上。 $howmny='S'$ の場合、 <i>select</i> は、条件数を計算する固有ペアを指定する。 $howmny='A'$ の場合、 <i>select</i> は参照されない。 実数型の場合： 実固有値 $\omega_j$ に対応する固有ペアの条件数を選択するには、 <i>select(j)</i> を <i>.TRUE.</i> に設定しなければならない。複素共役固有値ペア $\omega_j$ と $\omega_{j+1}$ の条件数を選択するには、 <i>select(j)</i> と <i>select(j+1)</i> のどちらかを <i>.TRUE.</i> に設定しなければならない。 複素数型の場合： $j$ 番目の固有値と固有ベクトルの条件数を選択するには、 <i>select(j)</i> を <i>.TRUE.</i> に設定しなければならない。
<i>n</i>	INTEGER。正方行列ペア ( $A, B$ ) の次数 ( $n \geq 0$ )。
<i>a, b, vl, vr, work</i>	REAL ( <i>stgsna</i> の場合) DOUBLE PRECISION ( <i>dtgsna</i> の場合) COMPLEX ( <i>ctgsna</i> の場合) DOUBLE COMPLEX ( <i>ztgsna</i> の場合)。 配列： $a(lda, *)$ 行列ペア ( $A, B$ ) における上準三角 (実数型の場合) または上三角 (複素数型の場合) 行列 $A$ を格納する。 $a$ の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $b(l db, *)$ 行列ペア ( $A, B$ ) における上三角行列 $B$ を格納する。 $b$ の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $job='E'$ または $'B'$ の場合、 $vl(ldvl, *)$ には、 <i>howmny</i> と <i>select</i> で指定した固有ペアに対応する、( $A, B$ ) の左固有ベクトルが格納されていなければならない。この固有ベクトルは、 <i>?tgevc</i> で返される通りに、 $vl$ の連続した列に格納されていなければならない。 $job='V'$ の場合、 $vl$ は参照されない。 $vl$ の第 2 次元は、 $\max(1, m)$ 以上でなければならない。

$job='E'$  または  $'B'$  の場合、 $vr(ldvr,*)$  には、 $howmny$  と  $select$  で指定した固有ペアに対応する、 $(A, B)$  の右固有ベクトルが格納されていなければならない。この固有ベクトルは、 $?tgevc$  で返される通りに、 $vr$  の連続した列に格納されていなければならない。

$job='V'$  の場合、 $vr$  は参照されない。

$vr$  の第 2 次元は、 $\max(1, m)$  以上でなければならない。

$work(lwork)$  は、ワークスペース配列である。 $job='E'$  の場合、 $work$  は参照されない。

$lda$	INTEGER。 $a$ の第 1 次元。 $\max(1, n)$ 以上。
$ldb$	INTEGER。 $b$ の第 1 次元。 $\max(1, n)$ 以上。
$ldv1$	INTEGER。 $v1$ の第 1 次元。 $ldv1 \geq 1$ 。 $job='E'$ または $'B'$ の場合、 $ldv1 \geq \max(1, n)$ 。
$ldvr$	INTEGER。 $vr$ の第 1 次元。 $ldvr \geq 1$ 。 $job='E'$ または $'B'$ の場合、 $ldvr \geq \max(1, n)$ 。
$mm$	INTEGER。 配列 $s$ と $dif$ の成分の個数 ( $mm \geq m$ )。
$lwork$	INTEGER。 配列 $work$ の次元。 実数型の場合: $lwork \geq n$ 。 $job='V'$ または $'B'$ の場合、 $lwork \geq 2n(n+2)+16$ 。 複素数型の場合: $lwork \geq 1$ 。 $job='V'$ または $'B'$ の場合、 $lwork \geq 2n^2$ 。 $lwork=-1$ の場合、ワークスペースのクエリとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエントリとして返す。 $xerbla$ は $lwork$ に関するエラー・メッセージを生成しない。
$iwork$	INTEGER。 ワークスペース配列、次元は $(n+6)$ 以上 (実数型の場合) または $(n+2)$ 以上 (複素数型の場合)。 $ijob='E'$ の場合、 $iwork$ は参照されない。

## 出力パラメータ

$s$	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 配列、次元は $(mm)$ 。 $job='E'$ または $'B'$ の場合、選択した固有値の条件数の逆数が、配列の連続成分に格納される。
-----	--



*job*='V' の場合、*s* は参照されない。

実数型の場合：

固有値の複素共役ペアに対して、*s* の 2 つの連続成分が同じ値に設定される。したがって、*s*(*j*)、*dif*(*j*)、*v1* と *vr* の *j* 番目の列が、同じ固有ペアに対応することになる（ただし、すべての固有ペアが選択される場合を除き、*j* 番目の固有ペアに対応することにはならない）。

*dif*

REAL ( 単精度の場合 )

DOUBLE PRECISION ( 倍精度の場合 )。

配列、次元は (*mm*)。

*job*='V' または 'B' の場合、選択した固有ベクトルの条件数の逆数の推定値が、配列の連続成分に格納される。*dif*(*j*) を計算するための固有値の順序変更ができなければ、*dif*(*j*) が 0 に設定される。これは、True 値が非常に小さい場合にのみ起こる。

*job*='E' の場合、*dif* は参照されない。

実数型の場合：

1 つの複素固有ベクトルに対して、*dif* の 2 つの連続成分が同じ値に設定される。

複素数型の場合：

*select* で指定された 1 つの固有値 / 固有ベクトルに対して、*Difl* の Frobenius ノルムベースの推定値が *dif* に格納される。

*m*

INTEGER。配列 *s* と *dif* において、指定された条件数を格納するのに使用した成分の個数。選択された 1 つの固有値に対して、1 つの成分が使用される。

*howmny*='A' の場合、*m* は *n* に設定される。

*work*(1)

*job* ≠ 'E' かつ *info* = 0 の場合、終了時に、最適なパフォーマンスを得るために必要な *lwork* の最小値が *work*(1) に格納される。以降の実行では、この *lwork* 値を使用する。

*info*

INTEGER。

*info* = 0 の場合、実行は正常に終了したことを示す。

*info* = -*i* の場合、*i* 番目のパラメータの値が不正であったことを示す。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *tgssna* のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ $(n,n)$ の行列 <i>A</i> を格納する。
<i>b</i>	サイズ $(n,n)$ の行列 <i>B</i> を格納する。
<i>s</i>	長さ $(mm)$ のベクトルを格納する。
<i>dif</i>	長さ $(mm)$ のベクトルを格納する。
<i>vl</i>	サイズ $(n,mm)$ の行列 <i>VL</i> を格納する。
<i>vr</i>	サイズ $(n,mm)$ の行列 <i>VR</i> を格納する。
<i>select</i>	長さ $(n)$ のベクトルを格納する。
<i>howmny</i>	引数 <i>select</i> の存在に基づいて以下のように復元される。 <i>howmny</i> = 'S' ( <i>select</i> が存在する場合)、 <i>howmny</i> = 'A' ( <i>select</i> が省略された場合)。
<i>job</i>	引数 <i>s</i> および <i>dif</i> の存在に基づいて以下のように復元される。 <i>job</i> = 'B' ( <i>s</i> と <i>dif</i> の両方が存在する場合)、 <i>job</i> = 'L' ( <i>s</i> が存在し、 <i>dif</i> が省略された場合)、 <i>job</i> = 'R' ( <i>s</i> が省略され、 <i>dif</i> が存在する場合)。 <i>s</i> と <i>dif</i> の両方が省略された場合、エラー条件がセットされる。

## 汎用特異値分解

このセクションでは、2つの行列 *A* と *B* に対して次のような汎用特異値分解 (GSVD) を行う LAPACK 計算ルーチンについて説明する。

$$U^H A Q = D_1 * (0 \ R),$$

$$V^H B Q = D_2 * (0 \ R),$$

*U*、*V*、および *Q* は直交 / ユニタリ行列、*R* は非特異上三角行列、*D*<sub>1</sub>、*D*<sub>2</sub> は「対角」行列 (詳しい構造については、各ルーチンの説明を参照) である。

表 4-7 に、行列の特異値分解を実行するための LAPACK ルーチン (Fortran-77 インターフェイス) を示す。Fortran-95 インターフェイスのルーチン名はそれぞれ、先頭の記号がない ([ルーチン命名規則](#) を参照)。

**表 4-7 汎用特異値分解のための計算ルーチン**

ルーチン名	機能
<a href="#">?ggsvp</a>	汎用 SVD のための予備的な分解を行う。
<a href="#">?tgsja</a>	2つの上三角行列または台形行列の汎用 SVD を求める。

一般三角行列ペアの **GSVD** を得るには、上記のルーチンを使用することも、ドライバルーチン [?ggsvd](#) を使用することもできる。

## ?ggsvp

汎用 **SVD** のための予備的な分解を行う。

### 構文

#### Fortran 77:

```
call sggsvp(jobu, jobv, jobq, m, p, n, a, lda, b, ldb, tola, tolb,
            k, l, u, ldu, v, ldv, q, ldq, iwork, tau, work, info)
call dggsvp(jobu, jobv, jobq, m, p, n, a, lda, b, ldb, tola, tolb,
            k, l, u, ldu, v, ldv, q, ldq, iwork, tau, work, info)
call cggsvp(jobu, jobv, jobq, m, p, n, a, lda, b, ldb, tola, tolb,
            k, l, u, ldu, v, ldv, q, ldq, iwork, rwork, tau, work, info)
call zggsvp(jobu, jobv, jobq, m, p, n, a, lda, b, ldb, tola, tolb,
            k, l, u, ldu, v, ldv, q, ldq, iwork, rwork, tau, work, info)
```

#### Fortran 95:

```
call ggsvp(a, b, tola, tolb [,k] [,l] [,u] [,v] [,q] [,info])
```

### 説明

このルーチンは、次のような直交行列  $U$ 、 $V$ 、および  $Q$  を求める。

$$U^H A Q = \begin{matrix} & \begin{matrix} n-k-l & k & l \end{matrix} \\ \begin{matrix} k \\ l \\ m-k-l \end{matrix} & \begin{pmatrix} 0 & A_{12} & A_{13} \\ 0 & 0 & A_{23} \\ 0 & 0 & 0 \end{pmatrix} \end{matrix}, \quad (m-k-l \geq 0 \text{ の場合})$$

$$= \begin{matrix} & \begin{matrix} n-k-l & k & l \end{matrix} \\ \begin{matrix} k \\ m-k \end{matrix} & \begin{pmatrix} 0 & A_{12} & A_{13} \\ 0 & 0 & A_{23} \end{pmatrix} \end{matrix}, \quad (m-k-l < 0 \text{ の場合})$$

$$V^H B Q = \begin{matrix} & n-k-1 & k & 1 \\ \begin{matrix} 1 \\ p-1 \end{matrix} & \begin{pmatrix} 0 & 0 & B_{13} \\ 0 & 0 & 0 \end{pmatrix} \end{matrix}$$

$k \times k$  の行列  $A_{12}$  と  $1 \times 1$  の行列  $B_{13}$  は、非特異上三角行列である。 $A_{23}$  は、 $m-k-1 \geq 0$  の場合は  $1 \times 1$  の上三角行列、それ以外の場合、 $A_{23}$  は  $(m-k) \times 1$  の上台形行列である。 $k+1$  の和は、 $(m+p) \times n$  の行列  $(A^H, B^H)^H$  の有効階数に等しい。

この分解は、汎用特異値分解 (GSVD) の予備ステップとして行われる。サブルーチン [?sggsvd](#) を参照。

## 入力パラメータ

<i>jobu</i>	CHARACTER*1。'U' または 'N' でなければならない。 <i>jobu</i> ='U' の場合、直交 / ユニタリ行列 <i>U</i> が計算される。 <i>jobu</i> ='N' の場合、 <i>U</i> は計算されない。
<i>jobv</i>	CHARACTER*1。'V' または 'N' でなければならない。 <i>jobv</i> ='V' の場合、直交 / ユニタリ行列 <i>V</i> が計算される。 <i>jobv</i> ='N' の場合、 <i>V</i> は計算されない。
<i>jobq</i>	CHARACTER*1。'Q' または 'N' でなければならない。 <i>jobq</i> ='Q' の場合、直交 / ユニタリ行列 <i>Q</i> が計算される。 <i>jobq</i> ='N' の場合、 <i>Q</i> は計算されない。
<i>m</i>	INTEGER。行列 <i>A</i> の行数 ( $m \geq 0$ )。
<i>p</i>	INTEGER。行列 <i>B</i> の行数 ( $p \geq 0$ )。
<i>n</i>	INTEGER。行列 <i>A</i> と <i>B</i> の列数 ( $n \geq 0$ )。
<i>a, b, tau, work</i>	REAL (sggsvp の場合 ) DOUBLE PRECISION (dggsvp の場合 ) COMPLEX (cggsvp の場合 ) DOUBLE COMPLEX (zggsvp の場合 )。 配列 : <i>a</i> ( <i>lda</i> ,*) には、 $m \times n$ の行列 <i>A</i> を格納する。 <i>a</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。  <i>b</i> ( <i>ldb</i> ,*) には、 $p \times n$ の行列 <i>B</i> を格納する。 <i>b</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。

	$\tau(*)$ は、ワークスペース配列である。 $\tau$ の次元は、 $\max(1, n)$ 以上でなければならない。
	$work(*)$ は、ワークスペース配列である。 $work$ の次元は、 $\max(1, 3n, m, p)$ 以上でなければならない。
$lda$	INTEGER。 $a$ の第 1 次元。 $\max(1, m)$ 以上。
$ldb$	INTEGER。 $b$ の第 1 次元。 $\max(1, p)$ 以上。
$tola, tol b$	REAL ( 単精度の場合 ) DOUBLE PRECISION ( 倍精度の場合 )。 $tola$ と $tol b$ は、行列 $B$ と部分ブロック $A$ の有効階数を決定するためのしきい値である。一般に、次のように設定される。 $tola = \max(m, n) * \ A\  * \text{MACHEPS}$ $tol b = \max(p, n) * \ B\  * \text{MACHEPS}$ $tola$ と $tol b$ のサイズが、分解の後退誤差のサイズに影響する可能性がある。
$ldu$	INTEGER。 出力配列 $u$ の第 1 次元。 $jobu = 'U'$ の場合、 $ldu \geq \max(1, m)$ 、それ以外の場合、 $ldu \geq 1$ 。
$ldv$	INTEGER。 出力配列 $v$ の第 1 次元。 $jobv = 'V'$ の場合、 $ldv \geq \max(1, p)$ 、それ以外の場合、 $ldv \geq 1$ 。
$ldq$	INTEGER。 出力配列 $q$ の第 1 次元。 $jobq = 'Q'$ の場合、 $ldq \geq \max(1, n)$ 、それ以外の場合、 $ldq \geq 1$ 。
$iwork$	INTEGER。 ワークスペース配列、次元は $\max(1, n)$ 以上。
$rwork$	REAL (cggsvp の場合 ) DOUBLE PRECISION ( zggsvp の場合 )。 ワークスペース配列、次元は $\max(1, 2n)$ 以上。複素数型でのみ使用される。

## 出力パラメータ

$a$	説明で示した三角 ( 台形 ) 行列で上書きされる。
$b$	説明で示した三角行列で上書きされる。
$k, l$	INTEGER。 終了時に、 $k$ と $l$ に部分ブロックのサイズが格納される。 $k+l$ は、 $(A^H, B^H)^H$ の有効階数に一致する。

*u, v, q* REAL (sggsvp の場合 )  
 DOUBLE PRECISION (dggsvp の場合 )  
 COMPLEX (cggsvp の場合 )  
 DOUBLE COMPLEX (zggsvp の場合 )。  
 配列:  
*jobu*='U' の場合、*u*(*ldu*,\*) に、直交 / ユニタリ行列 *U* が格納される。  
*u* の第 2 次元は、 $\max(1, m)$  以上でなければならない。  
*jobu*='N' の場合、*u* は参照されない。  
  
*jobv*='V' の場合、*v*(*ldv*,\*) に、直交 / ユニタリ行列 *V* が格納される。  
*v* の第 2 次元は、 $\max(1, m)$  以上でなければならない。  
*jobv*='N' の場合、*v* は参照されない。  
  
*jobq*='Q' の場合、*q*(*ldq*,\*) に、直交 / ユニタリ行列 *Q* が格納される。  
*q* の第 2 次元は、 $\max(1, n)$  以上でなければならない。  
*jobq*='N' の場合、*q* は参照されない。

*info* INTEGER。  
*info*=0 の場合、実行は正常に終了したことを示す。  
*info*=-*i* の場合、*i* 番目のパラメータの値が不正であったことを示す。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン ggsvp のインターフェイスの詳細を以下に示す。

*a* サイズ (*m*,*n*) の行列 *A* を格納する。  
*b* サイズ (*p*,*n*) の行列 *B* を格納する。  
*u* サイズ (*m*,*m*) の行列 *U* を格納する。  
*v* サイズ (*p*,*m*) の行列 *V* を格納する。  
*q* サイズ (*n*,*n*) の行列 *Q* を格納する。  
*jobu* 引数 *u* の存在に基づいて以下のように復元される。  
*jobu* = 'U' (*u* が存在する場合 )、  
*jobu* = 'N' (*u* が省略された場合 )。

*jobv*            引数 *v* の存在に基づいて以下のように復元される。  
*jobv* = 'V' (*v* が存在する場合)、  
*jobv* = 'N' (*v* が省略された場合)。

*jobq*            引数 *q* の存在に基づいて以下のように復元される。  
*jobq* = 'Q' (*q* が存在する場合)、  
*jobq* = 'N' (*q* が省略された場合)。

## ?tgsja

2 つの上三角行列または台形行列の  
汎用 SVD を計算する。

### 構文

#### Fortran 77:

```
call stgsja(jobu, jobv, jobq, m, p, n, k, l, a, lda, b, ldb, tola,
            tolb, alpha, beta, u, ldu, v, ldv, q, ldq, work, ncycle, info)
call dtgsja(jobu, jobv, jobq, m, p, n, k, l, a, lda, b, ldb, tola,
            tolb, alpha, beta, u, ldu, v, ldv, q, ldq, work, ncycle, info)
call ctgsja(jobu, jobv, jobq, m, p, n, k, l, a, lda, b, ldb, tola,
            tolb, alpha, beta, u, ldu, v, ldv, q, ldq, work, ncycle, info)
call ztgsja(jobu, jobv, jobq, m, p, n, k, l, a, lda, b, ldb, tola,
            tolb, alpha, beta, u, ldu, v, ldv, q, ldq, work, ncycle, info)
```

#### Fortran 95:

```
call tgsja(a, b, tola, tolb, k, l [,u] [,v] [,q] [,jobu] [,jobv] [,jobq]
[,alpha] [,beta] [,ncycle] [,info])
```

### 説明

このルーチンは、2 つの実 / 複素上三角行列 (または台形行列) *A* と *B* に対して、汎用特異値分解 (GSVD) を計算する。呼び出し時に、行列 *A* と *B* は次の形式であるとみなされる。この形式は、 $m \times n$  の一般行列 *A* と  $p \times n$  の一般行列 *B* を、予備的サブルーチン [?ggsvp](#) で前処理して得られる。

$$\begin{matrix} n-k-l & k & l \end{matrix}$$

$$A = \begin{matrix} & k & & \\ & \left( \begin{array}{ccc} 0 & A_{12} & A_{13} \\ 0 & 0 & A_{23} \\ 0 & 0 & 0 \end{array} \right) & \\ m-k-l & & \end{matrix}, \quad (m-k-l \geq 0 \text{ の場合})$$

$$= \begin{matrix} & n-k-l & k & l \\ & \left( \begin{array}{ccc} 0 & A_{12} & A_{13} \\ 0 & 0 & A_{23} \end{array} \right) & \\ m-k & & \end{matrix}, \quad (m-k-l < 0 \text{ の場合})$$

$$B = \begin{matrix} & n-k-l & k & l \\ & \left( \begin{array}{ccc} 0 & 0 & B_{13} \\ 0 & 0 & 0 \end{array} \right) & \\ p-l & & \end{matrix}$$

$k \times k$  の行列  $A_{12}$  と  $l \times l$  の行列  $B_{13}$  は、非特異上三角行列である。 $A_{23}$  は、 $m-k-l \geq 0$  の場合は  $l \times l$  の上三角行列、それ以外の場合、 $A_{23}$  は  $(m-k) \times l$  の上台形行列である。

終了時は、

$$U^H A Q = D_1 * (0 \ R), \quad V^H B Q = D_2 * (0 \ R)$$

$U$ 、 $V$ 、および  $Q$  は直交 / ユニタリ行列、 $R$  は非特異上三角行列、 $D_1$  と  $D_2$  は次の構造を持つ「対角」行列である。

$m-k-l \geq 0$  の場合、

$$D_1 = \begin{matrix} & k & & l \\ & \left( \begin{array}{cc} I & 0 \\ 0 & C \end{array} \right) & \\ m-k-l & & \end{matrix}$$

$$D_2 = \begin{matrix} & k & & l \\ & \left( \begin{array}{cc} 0 & S \\ 0 & 0 \end{array} \right) & \\ p-l & & \end{matrix}$$



$$\begin{pmatrix} 0 & R \end{pmatrix} = \begin{matrix} & n-k-l & k & l \\ \begin{matrix} k \\ 1 \end{matrix} & \begin{pmatrix} 0 & R_{11} & R_{12} \\ 0 & 0 & R_{22} \end{pmatrix} \end{matrix}$$

$C = \text{diag}(\alpha(k+1), \dots, \alpha(k+l))$

$S = \text{diag}(\beta(k+1), \dots, \beta(k+l))$

$C^2 + S^2 = I$

終了時に、 $R$  は  $a(1:k+l, n-k-l+1:n)$  に格納される。

$m-k-l < 0$  の場合、

$$D_1 = \begin{matrix} & k & m-k & k+l-m \\ \begin{matrix} k \\ m-k \end{matrix} & \begin{pmatrix} I & 0 & 0 \\ 0 & C & 0 \end{pmatrix} \end{matrix}$$

$$D_2 = \begin{matrix} & k & m-k & k+l-m \\ \begin{matrix} m-k \\ k+l-m \\ p-l \end{matrix} & \begin{pmatrix} 0 & S & 0 \\ 0 & 0 & I \\ 0 & 0 & 0 \end{pmatrix} \end{matrix}$$

$$\begin{pmatrix} 0 & R \end{pmatrix} = \begin{matrix} & n-k-l & k & m-k & k+l-m \\ \begin{matrix} k \\ m-k \\ k+l-m \end{matrix} & \begin{pmatrix} 0 & R_{11} & R_{12} & R_{13} \\ 0 & 0 & R_{22} & R_{23} \\ 0 & 0 & 0 & R_{33} \end{pmatrix} \end{matrix}$$

$$C = \text{diag}(\alpha(k+1), \dots, \alpha(m)),$$

$$S = \text{diag}(\beta(k+1), \dots, \beta(m)),$$

$$C^2 + S^2 = I$$

終了時に、 $\begin{pmatrix} R_{11} & R_{12} & R_{13} \\ 0 & R_{22} & R_{23} \end{pmatrix}$  は  $a(1:m, n-k-l+1:n)$  に格納され、 $R_{33}$  は

$b(m-k+1:l, n+m-k-l+1:n)$  に格納される。

オプションで、直交 / ユニタリ変換行列  $U$ 、 $V$ 、および  $Q$  を計算できる。これらの行列は明示的に生成することができる。また、入力行列  $U_1$ 、 $V_1$ 、および  $Q_1$  に後から掛けることもできる。

## 入力パラメータ

<i>jobu</i>	CHARACTER*1。'U'、'I'、または 'N' でなければならない。 <i>jobu</i> ='U' の場合、 <i>u</i> には、呼び出し時に、直交 / ユニタリ行列 $U_l$ が格納されていなければならない。 <i>jobu</i> ='I' の場合、 <i>u</i> は単位行列に初期化される。 <i>jobu</i> ='N' の場合、 <i>u</i> は計算されない。
<i>jobv</i>	CHARACTER*1。'V'、'I'、または 'N' でなければならない。 <i>jobv</i> ='V' の場合、 <i>v</i> には、呼び出し時に、直交 / ユニタリ行列 $V_l$ が格納されていなければならない。 <i>jobv</i> ='I' の場合、 <i>v</i> は単位行列に初期化される。 <i>jobv</i> ='N' の場合、 <i>v</i> は計算されない。
<i>jobq</i>	CHARACTER*1。'Q'、'I'、または 'N' でなければならない。 <i>jobq</i> ='Q' の場合、 <i>q</i> には、呼び出し時に、直交 / ユニタリ行列 $Q_l$ が格納されていなければならない。 <i>jobq</i> ='I' の場合、 <i>q</i> は単位行列に初期化される。 <i>jobq</i> ='N' の場合、 <i>q</i> は計算されない。
<i>m</i>	INTEGER。行列 $A$ の行数 ( $m \geq 0$ )。
<i>p</i>	INTEGER。行列 $B$ の行数 ( $p \geq 0$ )。
<i>n</i>	INTEGER。行列 $A$ と $B$ の列数 ( $n \geq 0$ )。
<i>k, l</i>	INTEGER。入力行列 $A$ と $B$ の部分ブロックを指定する。?tgsja は、これから GSVD を計算する

$a, b, u, v, q, work$  REAL (stgsja の場合)  
 DOUBLE PRECISION (dtgsja の場合)  
 COMPLEX (ctgsja の場合)  
 DOUBLE COMPLEX (ztgsja の場合)。  
 配列:  
 $a(lda, *)$  には、 $m \times n$  の行列  $A$  を格納する。  
 $a$  の第 2 次元は、 $\max(1, n)$  以上でなければならない。  
 $b(ldb, *)$  には、 $p \times n$  の行列  $B$  を格納する。  
 $b$  の第 2 次元は、 $\max(1, n)$  以上でなければならない。  
 $jobu = 'U'$  の場合、 $u(ldu, *)$  には、行列  $U_I$  (通常は、?ggsvp から返された直交 / ユニタリ行列) が格納されていなければならない。  
 $u$  の第 2 次元は、 $\max(1, m)$  以上でなければならない。  
 $jobv = 'V'$  の場合、 $v(ldv, *)$  には、行列  $V_I$  (通常は、?ggsvp から返された直交 / ユニタリ行列) が格納されていなければならない。  
 $v$  の第 2 次元は、 $\max(1, p)$  以上でなければならない。  
 $jobq = 'Q'$  の場合、 $q(ldq, *)$  には、行列  $Q_I$  (通常は、?ggsvp から返された直交 / ユニタリ行列) が格納されていなければならない。  
 $q$  の第 2 次元は、 $\max(1, n)$  以上でなければならない。  
 $work(*)$  は、ワークスペース配列である。 $work$  の次元は、 $\max(1, 2n)$  以上でなければならない。  
  
 $lda$  INTEGER。  $a$  の第 1 次元。  $\max(1, m)$  以上。  
 $ldb$  INTEGER。  $b$  の第 1 次元。  $\max(1, p)$  以上。  
 $ldu$  INTEGER。 配列  $u$  の第 1 次元。  
 $jobu = 'U'$  の場合、 $ldu \geq \max(1, m)$ 、それ以外の場合、 $ldu \geq 1$ 。  
 $ldv$  INTEGER。 配列  $v$  の第 1 次元。  
 $jobv = 'V'$  の場合、 $ldv \geq \max(1, p)$ 、それ以外の場合、 $ldv \geq 1$ 。  
 $ldq$  INTEGER。 配列  $q$  の第 1 次元。  
 $jobq = 'Q'$  の場合、 $ldq \geq \max(1, n)$ 、それ以外の場合、 $ldq \geq 1$ 。  
  
 $tola, tol b$  REAL (単精度の場合)  
 DOUBLE PRECISION (倍精度の場合)。  
 $tola$  と  $tol b$  は、Jacobi-Kogbetliantz 反復法における収束基準である。  
 一般に、?ggsvp で使用した値と同じ値にする。  
 $tola = \max(m, n) * \|A\| * \text{MACHEPS}$   
 $tol b = \max(p, n) * \|B\| * \text{MACHEPS}$

## 出力パラメータ

<i>a</i>	終了時に、 $a(n-k+1:n, 1:\min(k+1, m))$ に三角行列 $R$ または $R$ の一部が格納される。
<i>b</i>	終了時に、必要な場合、 $b(m-k+1:1, n+m-k-1+1:n)$ に $R$ の一部が格納される。
<i>alpha, beta</i>	<p>REAL ( 単精度の場合 )  DOUBLE PRECISION ( 倍精度の場合 )。  配列、次元は <math>\max(1, n)</math> 以上。  <math>A</math> と <math>B</math> の汎用特異値ペアが、次のように格納される。</p> <p><math>\alpha(1:k) = 1</math>  <math>\beta(1:k) = 0</math></p> <p><math>m-k-1 \geq 0</math> の場合、  <math>\alpha(k+1:k+1) = \text{diag}(C)</math>,  <math>\beta(k+1:k+1) = \text{diag}(S)</math></p> <p><math>m-k-1 &lt; 0</math> の場合、  <math>\alpha(k+1:m) = C</math>, <math>\alpha(m+1:k+1) = 0</math>  <math>\beta(k+1:m) = S</math>, <math>\beta(m+1:k+1) = 1</math></p> <p><math>k+1 &lt; n</math> の場合、  <math>\alpha(k+1+1:n) = 0</math>  <math>\beta(k+1+1:n) = 0</math></p>
<i>u</i>	<p><math>jobu = 'I'</math> の場合、<math>u</math> には、直交 / ユニタリ行列 <math>U</math> が格納される。  <math>jobu = 'U'</math> の場合、<math>u</math> には、積 <math>U_I U</math> が格納される。  <math>jobu = 'N'</math> の場合、<math>u</math> は参照されない。</p>
<i>v</i>	<p><math>jobv = 'I'</math> の場合、<math>v</math> には、直交 / ユニタリ行列 <math>U</math> が格納される。  <math>jobv = 'V'</math> の場合、<math>v</math> には、積 <math>V_I V</math> が格納される。  <math>jobv = 'N'</math> の場合、<math>v</math> は参照されない。</p>
<i>q</i>	<p><math>jobq = 'I'</math> の場合、<math>q</math> には、直交 / ユニタリ行列 <math>U</math> が格納される。  <math>jobq = 'Q'</math> の場合、<math>q</math> には、積 <math>Q_I Q</math> が格納される。  <math>jobq = 'N'</math> の場合、<math>q</math> は参照されない。</p>
<i>ncycle</i>	INTEGER。収束するのに要したサイクル数。
<i>info</i>	<p>INTEGER。  <math>info = 0</math> の場合、実行は正常に終了したことを示す。  <math>info = -i</math> の場合、<math>i</math> 番目のパラメータの値が不正であったことを示す。  <math>info = 1</math> の場合、この方法では MAXIT サイクル以内に収束しなかったことを示す。</p>

**Fortran 95 インターフェイス・ノート**

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `tgsgja` のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ $(m,n)$ の行列 <i>A</i> を格納する。
<i>b</i>	サイズ $(p,n)$ の行列 <i>B</i> を格納する。
<i>u</i>	サイズ $(m,m)$ の行列 <i>U</i> を格納する。
<i>v</i>	サイズ $(p,p)$ の行列 <i>V</i> を格納する。
<i>q</i>	サイズ $(n,n)$ の行列 <i>Q</i> を格納する。
<i>alpha</i>	長さ $(n)$ のベクトルを格納する。
<i>beta</i>	長さ $(n)$ のベクトルを格納する。
<i>jobu</i>	省略された場合、この引数は、引数 <i>u</i> の存在に基づいて以下のように復元される。 <i>jobu</i> = 'U' ( <i>u</i> が存在する場合)、 <i>jobu</i> = 'N' ( <i>u</i> が省略された場合)。 存在する場合、 <i>jobu</i> は 'I' または 'U' と等しくなければならず、引数 <i>u</i> も存在しなければならない。 <i>jobu</i> が存在し、 <i>u</i> が省略された場合、エラー条件がセットされる。
<i>jobv</i>	省略された場合、この引数は、引数 <i>v</i> の存在に基づいて以下のように復元される。 <i>jobv</i> = 'V' ( <i>v</i> が存在する場合)、 <i>jobv</i> = 'N' ( <i>v</i> が省略された場合)。 存在する場合、 <i>jobv</i> は 'I' または 'V' と等しくなければならず、引数 <i>v</i> も存在しなければならない。 <i>jobv</i> が存在し、 <i>v</i> が省略された場合、エラー条件がセットされる。
<i>jobq</i>	省略された場合、この引数は、引数 <i>q</i> の存在に基づいて以下のように復元される。 <i>jobq</i> = 'Q' ( <i>q</i> が存在する場合)、 <i>jobq</i> = 'N' ( <i>q</i> が省略された場合)。 存在する場合、 <i>jobq</i> は 'I' または 'Q' と等しくなければならず、引数 <i>q</i> も存在しなければならない。 <i>jobq</i> が存在し、 <i>q</i> が省略された場合、エラー条件がセットされる。

## ドライバルーチン

LAPACK ドライバルーチンを使用すると、ある問題を完全に解くのに、1 つのルーチン  
を呼び出すだけで済む。  
各ドライバルーチンは一般に、一連の[計算ルーチン](#)を適切に組み合わせて呼び出す。  
以下の各セクションで、ドライバルーチンについて説明する。

[線形最小二乗 \(LLS\) 問題](#)  
[汎用 LLS 問題](#)  
[対称固有値問題](#)  
[非対称固有値問題](#)  
[特異値分解](#)  
[汎用対称固有値問題](#)  
[汎用非対称固有値問題](#)

### 線形最小二乗 (LLS) 問題

このセクションでは、線形最小二乗問題を解くための LAPACK ドライバルーチンについて説明する。表 4-8 に、Fortran-77 インターフェイスのルーチンを示す。Fortran-95 インターフェイスのルーチン名はそれぞれ、先頭の記号がない ([ルーチン命名規則](#)を参照)。

表 4-8      **LLS 問題を解くためのドライバルーチン**

---

ルーチン名	機能
<a href="#">?gels</a>	QR または LQ 因子分解を使用して、最大階数の行列で表される、優決定または劣決定の線形問題を解く。
<a href="#">?gelsy</a>	A の完全直交因子分解を使用して、線形最小二乗問題の最小ノルム解を求める。
<a href="#">?gelss</a>	A の特異値分解を使用して、線形最小二乗問題の最小ノルム解を求める。
<a href="#">?gelstd</a>	A の特異値分解と分割統治法を使用して、線形最小二乗問題の最小ノルム解を求める。

## ?gels

*QR* または *LQ* 因子分解を使用して、大階数の行列で表される、決定または劣決定の線形問題を解く。

### 構文

#### Fortran 77:

```
call sgels(trans, m, n, nrhs, a, lda, b, ldb, work, lwork, info)
call dgels(trans, m, n, nrhs, a, lda, b, ldb, work, lwork, info)
call cgels(trans, m, n, nrhs, a, lda, b, ldb, work, lwork, info)
call zgels(trans, m, n, nrhs, a, lda, b, ldb, work, lwork, info)
```

#### Fortran 95:

```
call gels(a, b [,trans] [,info])
```

### 説明

このルーチンは、 $A$  の *QR* または *LQ* 因子分解を使用して、 $m \times n$  の行列  $A$  ( またはその転置 / 共役転置行列 ) で表される、実数 / 複素数の優決定 / 劣決定の線形問題を解く。ただし、 $A$  は最大階数の行列とする。

次のオプションが提供されている。

1.  $trans = 'N'$  で、 $m \geq n$  の場合：優決定の最小二乗解を求める。すなわち次の最小二乗問題を解く。

$$\text{minimize } \|b - Ax\|_2$$

2.  $trans = 'N'$  で、 $m < n$  の場合：劣決定の問題  $AX = B$  の最小ノルム解を求める。

3.  $trans = 'T'$  または  $'C'$  で、 $m \geq n$  の場合：劣決定の問題  $A^H X = B$  の最小ノルム解を求める。

4.  $trans = 'T'$  または  $'C'$  で、 $m < n$  の場合：優決定の最小二乗解を求める。すなわち次の最小二乗問題を解く。

$$\text{minimize } \|b - A^H x\|_2$$

複数の右辺のベクトル  $b$  と解ベクトル  $x$  を、一度の呼び出しで処理できる。これらのベクトルは、 $m \times nrhs$  の右辺の行列  $B$  と  $n \times nrh$  の解行列  $X$  の各列に格納される。

## 入力パラメータ

<i>trans</i>	CHARACTER*1。'N'、'T'、または 'C' でなければならない。 <i>trans</i> ='N' の場合、行列 <i>A</i> で表される線形問題。 <i>trans</i> ='T' の場合、転置行列 $A^T$ で表される線形問題 (実数型の場合のみ)。 <i>trans</i> ='C' の場合、共役転置行列 $A^H$ で表される線形問題 (複素数型の場合のみ)。
<i>m</i>	INTEGER。行列 <i>A</i> の行数 ( $m \geq 0$ )。
<i>n</i>	INTEGER。行列 <i>A</i> の列数 ( $n \geq 0$ )。
<i>nrhs</i>	INTEGER。右辺の数。 <i>B</i> の列数 ( $nrhs \geq 0$ )。
<i>a</i> , <i>b</i> , <i>work</i>	REAL (sgels の場合) DOUBLE PRECISION (dgels の場合) COMPLEX (cgels の場合) DOUBLE COMPLEX (zgels の場合)。 配列: <i>a</i> ( <i>lda</i> ,*) には、 $m \times n$ の行列 <i>A</i> を格納する。 <i>a</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。  <i>b</i> ( <i>ldb</i> ,*) には、右辺のベクトルが各列に入った行列 <i>B</i> を格納する。 <i>B</i> は、 <i>trans</i> ='N' の場合は $m \times nrhs$ 、 <i>trans</i> ='T' または 'C' の場合は $n \times nrhs$ 。 <i>b</i> の第 2 次元は、 $\max(1, nrhs)$ 以上でなければならない。  <i>work</i> ( <i>lwork</i> ) は、ワークスペース配列である。
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $\max(1, m)$ 以上。
<i>ldb</i>	INTEGER。 <i>b</i> の第 1 次元。 $\max(1, m, n)$ 以上でなければならない。
<i>lwork</i>	INTEGER。配列 <i>work</i> のサイズ。 $\min(m, n) + \max(1, m, n, nrhs)$ 以上でなければならない。 <i>lwork</i> = -1 の場合、ワークスペースのクエリとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリとして返す。 <i>xerbla</i> は <i>lwork</i> に関するエラー・メッセージを生成しない。  <i>lwork</i> の推奨値は、「アプリケーション・ノート」を参照。

## 出力パラメータ

<i>a</i>	終了時に、因子分解データで次のように上書きされる。
----------	---------------------------



	$m \geq n$ の場合、行列 $A$ の $QR$ 因子分解の詳細 ( <a href="#">?qgegrf</a> から返された値) が、配列 $a$ に格納される。 $m < n$ の場合、行列 $A$ の $LQ$ 因子分解の詳細 ( <a href="#">?gelqf</a> から返された値) が、配列 $a$ に格納される。
$b$	各列が解ベクトルで上書きされる。 $trans = 'N'$ で $m \geq n$ の場合、 $b$ の $1 \sim n$ 行に、最小二乗解のベクトルが格納される。各列の解の二乗和の誤差は、それぞれの列の $n+1 \sim m$ の成分の二乗和で与えられる。 $trans = 'N'$ で $m < n$ の場合、 $b$ の $1 \sim n$ 行に、最小ノルム解のベクトルが格納される。 $trans = 'T'$ または $'C'$ で $m \geq n$ の場合、 $b$ の $1 \sim m$ 行に、最小ノルム解のベクトルが格納される。 $trans = 'T'$ または $'C'$ で $m < n$ の場合、 $b$ の $1 \sim m$ 行に、最小ノルム解のベクトルが格納される。各列の解の二乗和の誤差は、それぞれの列の $m+1 \sim n$ の成分の二乗和で与えられる。
$work(1)$	$info = 0$ の場合、終了時に、最適なパフォーマンスを得るために必要な $lwork$ の最小値が $work(1)$ に格納される。以降の実行では、この $lwork$ 値を使用する。
$info$	INTEGER。 $info = 0$ の場合、実行は正常に終了したことを示す。 $info = -i$ の場合、 $i$ 番目のパラメータの値が不正であったことを示す。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `gels` のインターフェイスの詳細を以下に示す。

$a$	サイズ $(m, n)$ の行列 $A$ を格納する。
$b$	サイズ $\max(m, n) \times nrhs$ の行列を格納する。 $trans = 'N'$ の場合、呼び出し時に、 $b$ のサイズは $m \times nrhs$ である。 $trans = 'T'$ の場合、呼び出し時に、 $b$ のサイズは $n \times nrhs$ である。
$trans$	$'N'$ または $'T'$ でなければならない。デフォルト値は $'N'$ 。

### アプリケーション・ノート

パフォーマンスを改善するには、 $lwork = \min(m, n) + \max(1, m, n, nrhs) * blocksize$  に設定する。 $blocksize$  は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。

必要なワークスペースの大きさがわからない場合は、最初の実行では `lwork` を大きめの値に設定する。終了時に、`work(1)` の値を調べ、以降はその値を使用する。

---

### ?gelsy

*A* の完全直交因子分解を使用して、線形  
小二乗問題の最小ノルム解を求める。

---

#### 構文

##### Fortran 77:

```
call sgelsy(m, n, nrhs, a, lda, b, ldb, jpvt, rcond, rank, work,  
            lwork, info)  
call dgelsy(m, n, nrhs, a, lda, b, ldb, jpvt, rcond, rank, work,  
            lwork, info)  
call cgelsy(m, n, nrhs, a, lda, b, ldb, jpvt, rcond, rank, work,  
            lwork, rwork, info)  
call zgelsy(m, n, nrhs, a, lda, b, ldb, jpvt, rcond, rank, work,  
            lwork, rwork, info)
```

##### Fortran 95:

```
call gelsy(a, b [,rank] [,jpvt] [,rcond] [,info])
```

#### 説明

このルーチンは、実数 / 複素数の線形最小二乗問題、すなわち

$$\text{minimize } \|b - Ax\|_2$$

の最小ノルム解を、*A* の完全直交因子分解を使用して求める。*A* は  $m \times n$  の行列で、階数不足の可能性ある。

複数の右辺のベクトル *b* と解ベクトル *x* を、一度の呼び出しで処理できる。これらのベクトルは、 $m \times nrhs$  の右辺の行列 *B* と  $n \times nrhs$  の解行列 *X* の各列に格納される。

このルーチンは、まず列ピボット演算を用いた *QR* 因子分解を行う。

$$AP = Q \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix}$$

$R_{11}$  は最大の見出し部分行列で、条件数は  $1/rcond$  以下と見積もられる。 $R_{11}$  の次数 (*rank*) が、 $A$  の有効階数である。

次に、 $R_{22}$  は無視できるとみなし、 $R_{12}$  を右からの直交 / ユニタリ変換でゼロにして、次のような完全直交因子分解を得る。

$$AP = Q \begin{pmatrix} T_{11} & 0 \\ 0 & 0 \end{pmatrix} Z$$

最小ノルム解は次のようになる。

$$x = PZ^H \begin{pmatrix} T_{11}^{-1} Q_1^H b \\ 0 \end{pmatrix}$$

$Q_1$  は、 $Q$  の先頭 *rank* 列である。このルーチンは、元の `?gelsx` と基本的に同じであるが、次の点が異なる。

- サブルーチン [?geqp3](#) ではなく、サブルーチン [?geqp3](#) を呼び出している。このサブルーチンは、ピボット演算を用いた **QR** 因子分解の BLAS-3 版である。
- 行列  $B$  (右辺) が BLAS-3 で更新される。
- 行列  $B$  (右辺) の置換が、より高速かつ単純になっている。

### 入力パラメータ

<i>m</i>	INTEGER。行列 $A$ の行数 ( $m \geq 0$ )。
<i>n</i>	INTEGER。行列 $A$ の列数 ( $n \geq 0$ )。
<i>nrhs</i>	INTEGER。右辺の数 $BB$ の列数 ( $nrhs \geq 0$ )。
<i>a</i> , <i>b</i> , <i>work</i>	REAL (sgelsy の場合) DOUBLE PRECISION (dgelsy の場合) COMPLEX (cgelsy の場合) DOUBLE COMPLEX (zgelsy の場合)。 配列: $a(lda,*)$ には、 $m \times n$ の行列 $A$ を格納する。 $a$ の第 2 次元は、 $\max(1, n)$ 以上でなければならない。

	<p><math>b(l\delta b,*)</math> には、<math>m \times nrhs</math> の右辺の行列 <math>B</math> を格納する。  <math>b</math> の第 2 次元は、<math>\max(1, nrhs)</math> 以上でなければならない。</p> <p><math>work(lwork)</math> は、ワークスペース配列である。</p>
$lda$	INTEGER。 $a$ の第 1 次元。 $\max(1, m)$ 以上。
$ldb$	INTEGER。 $b$ の第 1 次元。 $\max(1, m, n)$ 以上でなければならない。
$jpvt$	<p>INTEGER。 配列、次元は <math>\max(1, n)</math> 以上。</p> <p>呼び出し時に、<math>jpvt(i) \neq 0</math> の場合、<math>A</math> の <math>i</math> 番目の列が <math>AP</math> の先頭に置換される。それ以外の場合、<math>A</math> の <math>i</math> 番目の列がフリー列になる。</p>
$rcond$	<p>REAL (単精度の場合)          DOUBLE PRECISION (倍精度の場合)。</p> <p><math>rcond</math> は、<math>A</math> の有効階数を決定するのに使用する。有効階数は、<math>A</math> のピボット演算を用いた <math>QR</math> 因子分解で得られた、先頭の最大三角部分行列 <math>R_{11}</math> の次数で定義される。その条件数の推定値 <math>&lt; 1/rcond</math> である。</p>
$lwork$	<p>INTEGER。 配列 <math>work</math> のサイズ。</p> <p><math>lwork = -1</math> の場合、ワークスペースのクエリとみなされ、ルーチンは <math>work</math> 配列の最適サイズだけを計算し、その値を <math>work</math> 配列の最初のエントリとして返す。xerbla は <math>lwork</math> に関するエラー・メッセージを生成しない。</p> <p><math>lwork</math> の推奨値は、「アプリケーション・ノート」を参照。</p>
$rwork$	<p>REAL (cgelsy の場合)          DOUBLE PRECISION (zgelsy の場合)。</p> <p>ワークスペース配列、次元は <math>\max(1, 2n)</math> 以上。複素数型でのみ使用される。</p>

## 出力パラメータ

$a$	終了時に、 $A$ の完全直交因子分解の詳細で上書きされる。
$b$	$n \times nrhs$ の解行列 $X$ で上書きされる。
$jpvt$	終了時に、 $jpvt(i) = k$ の場合、 $AP$ の $i$ 番目の列が $A$ の $k$ 番目の列である。
$rank$	<p>INTEGER。</p> <p><math>A</math> の有効階数、すなわち部分行列 <math>R_{11}</math> の次数。これは、<math>A</math> の完全直交因子分解における部分行列 <math>T_{11}</math> の次数と同じである。</p>

*info* INTEGER。  
*info* = 0 の場合、実行は正常に終了したことを示す。  
*info* = -*i* の場合、*i* 番目のパラメータの値が不正であったことを示す。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *gelsy* のインターフェイスの詳細を以下に示す。

*a*                    サイズ (*m*, *n*) の行列 *A* を格納する。  
*b*                    サイズ  $\max(m, n) \times nrhs$  の行列を格納する。  
                       呼び出し時に、 $m \times nrhs$  の右辺の行列 *B* を格納する。  
                       終了時に、 $n \times nrhs$  の解の行列 *X* によって上書きされる。  
*jpvt*                長さ (*n*) のベクトルを格納する。この成分のデフォルト値は、*jpvt*(*i*)  
                       である。  
*rcond*                この成分のデフォルト値は  $rcond = 100 * EPSILON(1.0\_WP)$  である。

### アプリケーション・ノート

実数型の場合：

ブロック化アルゴリズムを使用しない場合は、次のワークスペースが必要である。

$lwork \geq \max(mn + 3n + 1, 2 * mn + nrhs)$

ただし、 $mn = \min(m, n)$  である。

ブロック化アルゴリズムを使用する場合は、次のようになる。

$lwork \geq \max(mn + 2n + nb * (n + 1), 2 * mn + nb * nrhs)$ 、

ここで、*nb* は、[ilaenv](#) がルーチン *sgeqp3/dgeqp3*、*stzrzf/dtzrzf*、*stzrqf/dtzrqf*、*sormqr/dormqr*、*sormrz/dormrz* に対して返したブロックサイズの上限值である。

複素数型の場合：

ブロック化アルゴリズムを使用しない場合は、次のワークスペースが必要である。

$lwork \geq mn + \max(2 * mn, n + 1, mn + nrhs)$

ただし、 $mn = \min(m, n)$  である。

ブロック化アルゴリズムを使用する場合は、次のようになる。

$lwork \geq mn + \max(2*mn, nb*(n+1), mn+mn*nb, mn+nb*nrhs)$

ここで、 $nb$  は、[ilaenv](#) がルーチン `cgeqp3/zgeqp3`、`ctzrzf/ztzrzf`、`ctzrqf/ztzrqf`、`cunmqr/zunmqr`、`cunmrz/zunmrz` に対して返したブロックサイズの上限值である。

---

### ?gelss

$A$  の特異値分解を使用して、線形最小乗問題の最小ノルム解を求める。

---

#### 構文

##### Fortran 77:

```
call sgelss(m, n, nrhs, a, lda, b, ldb, s, rcond, rank, work,
            lwork, info)
call dgelss(m, n, nrhs, a, lda, b, ldb, s, rcond, rank, work,
            lwork, info)
call cgelss(m, n, nrhs, a, lda, b, ldb, s, rcond, rank, work,
            lwork, rwork, info)
call zgelss(m, n, nrhs, a, lda, b, ldb, s, rcond, rank, work,
            lwork, rwork, info)
```

##### Fortran 95:

```
call gelss(a, b [,rank] [,s] [,rcond] [,info])
```

#### 説明

このルーチンは、実数の線形最小二乗問題、すなわち  
$$\text{minimize } \|b - Ax\|_2$$

の最小ノルム解を、 $A$  の特異値分解 (SVD) を使用して求める。 $A$  は  $m \times n$  の行列で、階数不足の可能性がある。

複数の右辺のベクトル  $b$  と解ベクトル  $x$  を、一度の呼び出しで処理できる。これらのベクトルは、 $m \times nrhs$  の右辺の行列  $B$  と  $n \times nrhs$  の解行列  $X$  の各列に格納される。

$A$  の有効階数は、最大特異値の  $rcond$  倍より小さい特異値をゼロとして扱うことで決定される。

**入力パラメータ**

<i>m</i>	INTEGER。行列 <i>A</i> の行数 ( $m \geq 0$ )。
<i>n</i>	INTEGER。行列 <i>A</i> の列数 ( $n \geq 0$ )。
<i>nrhs</i>	INTEGER。右辺の数。 <i>B</i> の列数 ( $nrhs \geq 0$ )。
<i>a</i> , <i>b</i> , <i>work</i>	REAL (sgelss の場合 ) DOUBLE PRECISION (dgelss の場合 ) COMPLEX (cgelss の場合 ) DOUBLE COMPLEX (zgelss の場合 )。 配列： <i>a</i> ( <i>lda</i> ,*) には、 $m \times n$ の行列 <i>A</i> を格納する。 <i>a</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。  <i>b</i> ( <i>ldb</i> ,*) には、 $m \times nrhs$ の右辺の行列 <i>B</i> を格納する。 <i>b</i> の第 2 次元は、 $\max(1, nrhs)$ 以上でなければならない。  <i>work</i> ( <i>lwork</i> ) は、ワークスペース配列である。
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $\max(1, m)$ 以上。
<i>ldb</i>	INTEGER。 <i>b</i> の第 1 次元。 $\max(1, m, n)$ 以上でなければならない。
<i>rcond</i>	REAL ( 単精度の場合 ) DOUBLE PRECISION ( 倍精度の場合 )。  <i>rcond</i> は、 <i>A</i> の有効階数を決定するのに使用する。 特異値 $s(i) \leq rcond * s(1)$ をゼロとして扱う。 <i>rcond</i> < 0 の場合、 代わりにマシン精度を使用する。
<i>lwork</i>	INTEGER。配列 <i>work</i> のサイズ、 $lwork \geq 1$ 。 <i>lwork</i> = -1 の場合、ワークスペースのクエリとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエン トリとして返す。 <i>xerbla</i> は <i>lwork</i> に関するエラー・メッセージを 生成しない。  <i>lwork</i> の推奨値は、「アプリケーション・ノート」を参照。
<i>rwork</i>	REAL (cgelss の場合 ) DOUBLE PRECISION (zgelss の場合 )。 複素数型でのみ使用されるワークスペース配列である。 次元は $\max(1, 5 * \min(m, n))$ 以上。

## 出力パラメータ

<i>a</i>	終了時に、 <i>A</i> の先頭から $\min(m, n)$ 個の各行に、右特異ベクトルが格納される。
<i>b</i>	$n \times nrhs$ の解行列 <i>X</i> で上書きされる。  $m \geq n$ で $rank = n$ の場合、 <i>i</i> 番目の列の解の二乗和の誤差は、その列の $n+1:m$ の成分の二乗和で与えられる。
<i>s</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 配列、次元は $\max(1, \min(m, n))$ 以上。 <i>A</i> の特異値が降順で格納される。 <i>A</i> の 2- ノルムの条件数は、 $k_2(A) = s(1) / s(\min(m, n))$ 。
<i>rank</i>	INTEGER。 <i>A</i> の有効階数、すなわち、 $rcond * s(1)$ より大きい特異値の個数。
<i>work(1)</i>	<i>info</i> = 0 の場合、終了時に、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が <i>work(1)</i> に格納される。以降の実行では、この <i>lwork</i> 値を使用する。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正であったことを示す。 <i>info</i> = <i>i</i> の場合、SVD の計算アルゴリズムが収束に失敗したことを示す。 <i>i</i> は、中間の二重対角形式の非対角成分で、ゼロに収束しなかった成分の個数を表す。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *gelss* のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ $(m, n)$ の行列 <i>A</i> を格納する。
<i>b</i>	サイズ $\max(m, n) \times nrhs$ の行列を格納する。 呼び出し時に、 $m \times nrhs$ の右辺の行列 <i>B</i> を格納する。 終了時に、 $n \times nrhs$ の解の行列 <i>X</i> によって上書きされる。
<i>s</i>	長さ $\min(m, n)$ のベクトルを格納する。



*rcond*                      この成分のデフォルト値は  $rcond = 100 * \text{EPSILON}(1.0\_WP)$  である。

## アプリケーション・ノート

実数型の場合：

$$lwork \geq 3 * \min(m, n) + \max(2 * \min(m, n), \max(m, n), nrhs)$$

複素数型の場合：

$$lwork \geq 2 * \min(m, n) + \max(m, n, nrhs)$$

パフォーマンスを改善するには、一般に *lwork* に上記以上の値が必要である。必要なワークスペースの大きさがわからない場合は、最初の実行では *lwork* を大きめの値に設定する。終了時に、*work(1)* の値を調べ、以降はその値を使用する。

---

## ?gelsd

*A* の特異値分解と分割統治法を使用して、  
線形最小二乗問題の最小ノルム解を求める。

---

### 構文

#### Fortran 77:

```
call sgelsd(m, n, nrhs, a, lda, b, ldb, s, rcond, rank, work,  
            lwork, iwork, info)  
call dgelsd(m, n, nrhs, a, lda, b, ldb, s, rcond, rank, work,  
            lwork, iwork, info)  
call cgelsd(m, n, nrhs, a, lda, b, ldb, s, rcond, rank, work,  
            lwork, rwork, iwork, info)  
call zgelsd(m, n, nrhs, a, lda, b, ldb, s, rcond, rank, work,  
            lwork, rwork, iwork, info)
```

#### Fortran 95:

```
call gelsd(a, b [,rank] [,s] [,rcond] [,info])
```

## 説明

このルーチンは、実数の線形最小二乗問題、すなわち

$$\text{minimize } \|b - Ax\|_2$$

の最小ノルム解を、 $A$  の特異値分解 (SVD) を使用して求める。 $A$  は  $m \times n$  の行列で、階数不足の可能性はある。

複数の右辺のベクトル  $b$  と解ベクトル  $x$  を、一度の呼び出しで処理できる。これらのベクトルは、 $m \times nrhs$  の右辺の行列  $B$  と  $n \times nrhs$  の解行列  $X$  の各列に格納される。

この問題は、次の 3 ステップで解く。

1. Householder 変換により係数行列  $A$  を二重対角形式に縮退し、元の問題を「二重対角形式の最小二乗問題 (BLS)」に縮退する。
2. 分割統治法を用いて、BLS を解く。
3. すべての Householder 変換を逆に適用して、元の最小二乗問題を解く。

$A$  の有効階数は、最大特異値の  $rcond$  倍より小さい特異値をゼロとして扱うことで決定される。

このルーチンは、補助ルーチン [?lals0](#) および [?lalsa](#) を使用する。

## 入力パラメータ

$m$	INTEGER。行列 $A$ の行数 ( $m \geq 0$ )。
$n$	INTEGER。行列 $A$ の列数 ( $n \geq 0$ )。
$nrhs$	INTEGER。右辺の数。 $B$ の列数 ( $nrhs \geq 0$ )。
$a, b, work$	REAL (sgelsd の場合) DOUBLE PRECISION (dgelsd の場合) COMPLEX (cgelsd の場合) DOUBLE COMPLEX (zgelsd の場合)。 配列: $a(lda,*)$ には、 $m \times n$ の行列 $A$ を格納する。 $a$ の第 2 次元は、 $\max(1, n)$ 以上でなければならない。  $b ldb,*)$ には、 $m \times nrhs$ の右辺の行列 $B$ を格納する。 $b$ の第 2 次元は、 $\max(1, nrhs)$ 以上でなければならない。  $work(lwork)$ は、ワークスペース配列である。
$lda$	INTEGER。 $a$ の第 1 次元。 $\max(1, m)$ 以上。

<i>ldb</i>	INTEGER。 <i>b</i> の第 1 次元。 $\max(1, m, n)$ 以上でなければならない。
<i>rcond</i>	REAL ( 単精度の場合 ) DOUBLE PRECISION ( 倍精度の場合 )。  <i>rcond</i> は、 <i>A</i> の有効階数を決定するのに使用する。 特異値 $s(i) \leq rcond * s(1)$ をゼロとして扱う。 $rcond < 0$ の場合、 代わりにマシン精度を使用する。
<i>lwork</i>	INTEGER。 配列 <i>work</i> のサイズ、 $lwork \geq 1$ 。 <i>lwork</i> = -1 の場合、ワークスペースのクエリとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエン トリとして返す。 <i>xerbla</i> は <i>lwork</i> に関するエラー・メッセージを 生成しない。  <i>lwork</i> の推奨値は、「アプリケーション・ノート」を参照。
<i>iwork</i>	INTEGER。 ワークスペース配列。 <i>iwork</i> の推奨値は、「アプリケーション ・ノート」を参照。
<i>rwork</i>	REAL (cgelsd の場合 ) DOUBLE PRECISION (zgelsd の場合 )  複素数型でのみ使用されるワークスペース配列である。  <i>rwork</i> の推奨値は、「アプリケーション・ノート」を参照。

### 出力パラメータ

<i>a</i>	終了時に、 <i>A</i> が上書きされる。
<i>b</i>	$n \times nrhs$ の解行列 <i>X</i> で上書きされる。  $m \geq n$ で $rank = n$ の場合、 <i>i</i> 番目の列の解の二乗和の誤差は、その列の $n+1:m$ の成分の二乗和で与えられる。
<i>s</i>	REAL ( 単精度の場合 ) DOUBLE PRECISION ( 倍精度の場合 )。 配列、次元は $\max(1, \min(m, n))$ 以上。 <i>A</i> の特異値が降順で格納される。 <i>A</i> の 2- ノルムの条件数は、 $k_2(A) = s(1) / s(\min(m, n))$ 。
<i>rank</i>	INTEGER。 <i>A</i> の有効階数、すなわち、 $rcond * s(1)$ より大きい特異値の個数。
<i>work(1)</i>	<i>info</i> = 0 の場合、終了時に、最適なパフォーマンスを得るために必要 な <i>lwork</i> の最小値が <i>work(1)</i> に格納される。以降の実行では、この <i>lwork</i> 値を使用する。

*info* INTEGER。  
*info* = 0 の場合、実行は正常に終了したことを示す。  
*info* = -*i* の場合、*i* 番目のパラメータの値が不正であったことを示す。  
*info* = *i* の場合、SVD の計算アルゴリズムが収束に失敗したことを示す。*i* は、中間の二重対角形式の非対角成分で、ゼロに収束しなかった成分の個数を表す。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `gelsd` のインターフェイスの詳細を以下に示す。

*a* サイズ (*m*, *n*) の行列 *A* を格納する。  
*b* サイズ  $\max(m, n) \times nrhs$  の行列を格納する。  
 呼び出し時に、 $m \times nrhs$  の右辺の行列 *B* を格納する。  
 終了時に、 $n \times nrhs$  の解の行列 *X* によって上書きされる。  
*s* 長さ  $\min(m, n)$  のベクトルを格納する。  
*rcond* この成分のデフォルト値は  $rcond = 100 * EPSILON(1.0\_WP)$  である。

## アプリケーション・ノート

分割統治法は、浮動小数点演算において、非常に緩い仮定を行う。すなわち、加減算においてガード桁を持つマシンで機能する。ガード桁を持たない 16 進マシンや 10 進マシンでは異常終了するのも考えられるが、そのような前例はない。

必要なワークスペースの最小値は、*m*、*n*、*nrhs* によって決まる。ワークスペース配列 *work* のサイズ *lwork* は、次に示す値以上でなければならない。

実数型の場合：

$m \geq n$  の場合、  
 $lwork \geq 12n + 2n * smlsiz + 8n * nlvl + n * nrhs + (smlsiz + 1)^2$

$m < n$  の場合、  
 $lwork \geq 12m + 2m * smlsiz + 8m * nlvl + m * nrhs + (smlsiz + 1)^2$

複素数型の場合：

$m \geq n$  の場合、  
 $lwork \geq 2n + n*nrhs$

$m < n$  の場合、  
 $lwork \geq 2m + m*nrhs$

$smlsiz$  は `ilaenv` に返された値で、計算ツリーの一層下の部分問題の最大サイズ (通常は約 25) で、 $nlvl = \text{INT}(\log_2(\min(m, n)/(smlsiz+1))) + 1$ 。

パフォーマンスを改善するには、一般に  $lwork$  に上記以上の値が必要である。必要なワークスペースの大きさがわからない場合は、最初の実行では  $lwork$  を大きめの値に設定する。終了時に、 $work(1)$  の値を調べ、以降はその値を使用する。

ワークスペース配列  $iwork$  のサイズは、 $3*\min(m, n)*nlvl + 11*\min(m, n)$  以上でなければならない。

ワークスペース配列  $rwork$  (複素数型の場合) のサイズ  $lrwork$  は、次の値でなければならない。 $m \geq n$  の場合、  
 $lrwork \geq 10n + 2n*smlsiz + 8n*nlvl + 3*smlsiz*nrhs + (smlsiz+1)^2$

$m < n$  の場合、  
 $lrwork \geq 10m + 2m*smlsiz + 8m*nlvl + 3*smlsiz*nrhs + (smlsiz+1)^2$

## 汎用 LLS 問題

このセクションでは、汎用線形最小二乗問題を解くための LAPACK ドライバルーチンについて説明する。[表 4-9](#) に、Fortran-77 インターフェイスのルーチンを示す。

Fortran-95 インターフェイスのルーチン名はそれぞれ、先頭の記号がない ([ルーチン命名規則](#)を参照)。

**表 4-9 汎用 LLS 問題を解くためのドライバルーチン**

ルーチン名	機能
<a href="#">?gqlse</a>	汎用 RQ 因子分解を使用して、均衡制約を持つ線形最小二乗問題を解く。
<a href="#">?ggglm</a>	汎用 QR 因子分解を使用して、Gauss-Markov 線形モデル問題を解く。

### sgglsse

汎用  $RQ$  因子分解を使用して、均衡制約を持つ線形最小二乗問題を解く。

---

#### 構文

##### Fortran 77:

```
call sgglsse(m, n, p, a, lda, b, ldb, c, d, x, work, lwork, info)
call dgglsse(m, n, p, a, lda, b, ldb, c, d, x, work, lwork, info)
call cgglsse(m, n, p, a, lda, b, ldb, c, d, x, work, lwork, info)
call zgglsse(m, n, p, a, lda, b, ldb, c, d, x, work, lwork, info)
```

##### Fortran 95:

```
call ggsse(a, b, c, d, x [,info])
```

#### 説明

このルーチンは、次のような均衡制約を持つ線形最小二乗 (LSE) 問題を解く。

$$\text{minimize } \|c - Ax\|_2 \quad \text{ただし、} Bx = d$$

$A$  は  $m \times n$  の行列、 $B$  は  $p \times n$  の行列、 $c$  は与えられた  $m$ -ベクトル、 $d$  は与えられた  $p$ -ベクトルである。

ただし、 $p \leq n \leq m+p$ 、

$$\text{rank}(B) = p, \text{rank} \begin{pmatrix} A \\ B \end{pmatrix} = n \text{ と仮定する。}$$

これらの条件により、この LSE 問題は、行列  $B$  と  $A$  の汎用  $RQ$  因子分解により、一意の解が得られる問題になる。

#### 入力パラメータ

$m$	INTEGER。行列 $A$ の行数 ( $m \geq 0$ )。
$n$	INTEGER。行列 $A$ と $B$ の列数 ( $n \geq 0$ )。
$p$	INTEGER。行列 $B$ の行数 ( $0 \leq p \leq n \leq m+p$ )。

$a, b, c, d, work$  REAL (sgglse の場合)  
 DOUBLE PRECISION (dgglse の場合)  
 COMPLEX (cgglse の場合)  
 DOUBLE COMPLEX (zgglse の場合)

配列:  
 $a(lda, *)$  には、 $m \times n$  の行列  $A$  を格納する。  
 $a$  の第 2 次元は、 $\max(1, n)$  以上でなければならない。

$b(ldb, *)$  には、 $p \times n$  の行列  $B$  を格納する。  
 $b$  の第 2 次元は、 $\max(1, n)$  以上でなければならない。

$c(*)$  は、サイズが  $\max(1, m)$  以上の配列で、LSE 問題の最小二乗部分に関する右辺のベクトルを格納する。  
 $d(*)$  は、サイズが  $\max(1, p)$  以上の配列で、制約式の右辺のベクトルを格納する。  
 $work(lwork)$  は、ワークスペース配列である。

$lda$  INTEGER。  $a$  の第 1 次元。  $\max(1, m)$  以上。

$ldb$  INTEGER。  $b$  の第 1 次元。  $\max(1, p)$  以上。

$lwork$  INTEGER。 配列  $work$  のサイズ。  $lwork \geq \max(1, m+n+p)$ 。  
 $lwork = -1$  の場合、ワークスペースのクエリとみなされ、ルーチンは  $work$  配列の最適サイズだけを計算し、その値を  $work$  配列の最初のエントリとして返す。  $xerbla$  は  $lwork$  に関するエラー・メッセージを生成しない。

$lwork$  の推奨値は、「アプリケーション・ノート」を参照。

### 出力パラメータ

$x$  REAL (sgglse の場合)  
 DOUBLE PRECISION (dgglse の場合)  
 COMPLEX (cgglse の場合)  
 DOUBLE COMPLEX (zgglse の場合)。  
 配列、次元は  $\max(1, n)$  以上。  
 終了時に、LSE 問題の解が格納される。

$a, b, d$  終了時に、これらの配列が上書きされる。

$c$  終了時に、解の二乗和の誤差が、ベクトル  $c$  の  $n-p+1 \sim m$  の成分の二乗和で与えられる。

<i>work(1)</i>	<i>info</i> = 0 の場合、終了時に、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が <i>work(1)</i> に格納される。以降の実行では、この <i>lwork</i> 値を使用する。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正であったことを示す。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *gg1se* のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ ( <i>m</i> , <i>n</i> ) の行列 <i>A</i> を格納する。
<i>b</i>	サイズ ( <i>p</i> , <i>n</i> ) の行列 <i>B</i> を格納する。
<i>c</i>	長さ ( <i>m</i> ) のベクトルを格納する。
<i>d</i>	長さ ( <i>p</i> ) のベクトルを格納する。
<i>x</i>	長さ ( <i>n</i> ) のベクトルを格納する。

### アプリケーション・ノート

最適なパフォーマンスを得るには、次の値を使用する。

$$lwork \geq p + \min(m, n) + \max(m, n) * nb$$

ここで、*nb* は、*?gegrf*、*?gerqf*、*?ormqr*/*?unmqr*、および *?ormrq*/*?unmrq* に対する最適ブロックサイズの上限值である。



## ?ggglm

汎用 *QR* 因子分解を使用して、*Gauss-Markov* 線形モデル問題を解く。

### 構文

#### Fortran 77:

```
call sggglm(n, m, p, a, lda, b, ldb, d, x, y, work, lwork, info)
call dggglm(n, m, p, a, lda, b, ldb, d, x, y, work, lwork, info)
call cggglm(n, m, p, a, lda, b, ldb, d, x, y, work, lwork, info)
call zggglm(n, m, p, a, lda, b, ldb, d, x, y, work, lwork, info)
```

#### Fortran 95:

```
call gggglm(a, b, d, x, y [,info])
```

### 説明

このルーチンは、次のような一般 *Gauss-Markov* 線形モデル (GLM) 問題を解く。

$$\text{minimize}_x \|y\|_2 \quad \text{ただし、} d = Ax + By$$

ここで、 $A$  は  $n \times m$  の行列、 $B$  は  $n \times p$  の行列、 $d$  は与えられた  $n$ -ベクトルである。  
 $m \leq n \leq m+p$ 、

$$\text{rank}(A) = m, \text{rank}(A \ B) = n \text{ と仮定する。}$$

これらの仮定により、制約式が常に成立し、 $A$  と  $B$  の汎用 *QR* 因子分解により、一意の解  $x$  と最小 2- ノルム解  $y$  が得られる。

特に、行列  $B$  が正方非特異行列の場合、この GLM 問題は、次のような重み付け線形最小二乗問題と等価になる。

$$\text{minimize}_x \|B^{-1}(d - Ax)\|_2$$

### 入力パラメータ

$n$	INTEGER。行列 $A$ と $B$ の行数 ( $n \geq 0$ )。
$m$	INTEGER。行列 $A$ の列数 ( $m \geq 0$ )。
$p$	INTEGER。行列 $B$ の列数 ( $p \geq n - m$ )。
$a, b, d, work$	REAL (sggglm の場合) DOUBLE PRECISION (dggglm の場合) COMPLEX (cggglm の場合) DOUBLE COMPLEX (zggglm の場合)

配列:

$a(lda,*)$  には、 $n \times m$  の行列  $A$  を格納する。  
 $a$  の第 2 次元は、 $\max(1, m)$  以上でなければならない。

$b(ldb,*)$  には、 $n \times p$  の行列  $B$  を格納する。  
 $b$  の第 2 次元は、 $\max(1, p)$  以上でなければならない。

$d(*)$  は、サイズが  $\max(1, n)$  以上の配列で、GLM 式の左辺を格納する。

$work(lwork)$  は、ワークスペース配列である。

$lda$  INTEGER。  $a$  の第 1 次元。  $\max(1, n)$  以上。

$ldb$  INTEGER。  $b$  の第 1 次元。  $\max(1, n)$  以上。

$lwork$  INTEGER。 配列  $work$  のサイズ。  $lwork \geq \max(1, n+m+p)$ 。  
 $lwork = -1$  の場合、ワークスペースのクエリとみなされ、ルーチンは  $work$  配列の最適サイズだけを計算し、その値を  $work$  配列の最初のエントリとして返す。  $xerbla$  は  $lwork$  に関するエラー・メッセージを生成しない。

$lwork$  の推奨値は、「アプリケーション・ノート」を参照。

## 出力パラメータ

$x, y$  REAL (sggglm の場合 )  
DOUBLE PRECISION (dggglm の場合 )  
COMPLEX (cggglm の場合 )  
DOUBLE COMPLEX (zggglm の場合 )。  
配列  $x(*)$ 、 $y(*)$ 。 次元は  $\max(1, m)$  以上 ( $x$  の場合 ) または  $\max(1, p)$  以上 ( $y$  の場合 )。  
終了時に、 $x$  と  $y$  が GLM 問題の解になる。

$a, b, d$  終了時に、これらの配列が上書きされる。

$work(1)$   $info = 0$  の場合、終了時に、最適なパフォーマンスを得るために必要な  $lwork$  の最小値が  $work(1)$  に格納される。

$info$  INTEGER。  
 $info = 0$  の場合、実行は正常に終了したことを示す。  
 $info = -i$  の場合、 $i$  番目のパラメータの値が不正であったことを示す。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `ggglm` のインターフェイスの詳細を以下に示す。

- `a`                    サイズ  $(n, m)$  の行列  $A$  を格納する。
- `b`                    サイズ  $(n, p)$  の行列  $B$  を格納する。
- `d`                    長さ  $(n)$  のベクトルを格納する。
- `x`                    長さ  $(m)$  のベクトルを格納する。
- `y`                    長さ  $(p)$  のベクトルを格納する。

## アプリケーション・ノート

最適なパフォーマンスを得るには、次の値を使用する。

$$lwork \geq m + \min(n, p) + \max(n, p) * nb$$

ここで、 $nb$  は、`?geqrf`、`?gerqf`、`?ormqr`/`?unmqr`、および `?ormrq`/`?unmrq` に対する最適ブロックサイズの上限值である。

## 対称固有値問題

このセクションでは、対称固有値問題を解くための LAPACK ドライバルーチンについて説明する。これらの問題を解くために呼び出すことが可能な[計算ルーチン](#)も参照のこと。

[表 4-10](#) に Fortran-77 インターフェイスのすべてのドライバルーチンを示す。Fortran-95 インターフェイスのルーチン名はそれぞれ、先頭の記号がない ([ルーチン命名規則](#)を参照)。

**表 4-10      対称固有値問題を解くためのドライバルーチン**

ルーチン名	機能
<a href="#">?syev</a> / <a href="#">?heev</a>	実対称 / エルミート行列の固有値と (オプションで) 固有ベクトルをすべて計算する。
<a href="#">?syevd</a> / <a href="#">?heevd</a>	分割統治アルゴリズムを使用して、実対称 / エルミート行列の固有値と (オプションで) 固有ベクトルをすべて計算する。
<a href="#">?syevx</a> / <a href="#">?heevx</a>	対称 / エルミート行列の固有値と (オプションで) 固有ベクトルを選択的に計算する。

**表 4-10 対称固有値問題を解くためのドライバルーチン**

ルーチン名	機能
<a href="#">?syevr</a> / <a href="#">?heevr</a>	「比較的安定な表現」を使用して、実対称 / エルミート行列の固有値と (オプションで) 固有ベクトルを選択的に計算する。
<a href="#">?spev</a> / <a href="#">?hpev</a>	圧縮形式の実対称 / エルミート行列の固有値と (オプションで) 固有ベクトルをすべて計算する。
<a href="#">?spevd</a> / <a href="#">?hpevd</a>	分割統治アルゴリズムを使用して、圧縮形式の実対称 / エルミート行列の固有値と (オプションで) 固有ベクトルをすべて計算する。
<a href="#">?spevx</a> / <a href="#">?hpevx</a>	圧縮形式の実対称 / エルミート行列の固有値と (オプションで) 固有ベクトルを選択的に計算する。
<a href="#">?sbev</a> / <a href="#">?hbev</a>	帯形式の実対称 / エルミート行列の固有値と (オプションで) 固有ベクトルをすべて計算する。
<a href="#">?sbevd</a> / <a href="#">?hbevd</a>	分割統治アルゴリズムを使用して、帯形式の実対称 / エルミート行列の固有値と (オプションで) 固有ベクトルをすべて計算する。
<a href="#">?sbevx</a> / <a href="#">?hbevx</a>	帯形式の実対称 / エルミート行列の固有値と (オプションで) 固有ベクトルを選択的に計算する。
<a href="#">?stev</a>	実対称三重対角行列の固有値と (オプションで) 固有ベクトルをすべて計算する。
<a href="#">?stevd</a>	分割統治アルゴリズムを使用して、実対称三重対角行列の固有値と (オプションで) 固有ベクトルをすべて計算する。
<a href="#">?stevx</a>	実対称三重対角行列の固有値と固有ベクトルを選択的に計算する。
<a href="#">?stevr</a>	「比較的安定な表現」を使用して、実対称三重対角行列の固有値と (オプションで) 固有ベクトルを選択的に計算する。

## ?syev

実対称行列の固有値と (オプションで)  
固有ベクトルをすべて計算する。

### 構文

#### Fortran 77:

```
call ssyev(jobz, uplo, n, a, lda, w, work, lwork, info)
call dsyev(jobz, uplo, n, a, lda, w, work, lwork, info)
```

#### Fortran 95:

```
call syev(a, w [,jobz] [,uplo] [,info])
```

## 説明

このルーチンは、実対称行列  $A$  のすべての固有値と (オプションで) 固有ベクトルをすべて計算する。

ほとんどの実対称固有値問題では、基本となるアルゴリズムがより高速でワークスペースの使用が少なくなるように、デフォルトで [?syevr](#) 関数を使用する。

## 入力パラメータ

<i>jobz</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobz</i> = 'N' の場合、固有値のみを計算する。 <i>jobz</i> = 'V' の場合、固有値と固有ベクトルを計算する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> = 'U' の場合、 $a$ には $A$ の上三角部分を格納する。 <i>uplo</i> = 'L' の場合、 $a$ には $A$ の下三角部分を格納する。
<i>n</i>	INTEGER。行列 $A$ の次数 ( $n \geq 0$ )。
<i>a, work</i>	REAL (ssyev の場合) DOUBLE PRECISION (dsyev の場合) 配列: $a(lda,*)$ は、 <i>uplo</i> の値に従って、対称行列 $A$ の上または下三角部分を格納する配列である。 $a$ の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $work(lwork)$ は、ワークスペース配列である。
<i>lda</i>	INTEGER。配列 $a$ の第 1 次元。 $\max(1, n)$ 以上でなければならない。
<i>lwork</i>	INTEGER。配列 $work$ の次元。 次の制約がある。 $lwork \geq \max(1, 3n-1)$ 。 $lwork = -1$ の場合、ワークスペースのクエリとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエントリとして返す。xerbla は $lwork$ に関するエラー・メッセージを生成しない。  $lwork$ の推奨値は、「アプリケーション・ノート」を参照。

## 出力パラメータ

<i>a</i>	<i>jobz</i> ='V' の場合、終了時に、 <i>info</i> =0 であれば、行列 <i>A</i> の正規直交固有ベクトルが、配列 <i>a</i> に格納される。 <i>jobz</i> ='N' の場合、終了時に、行列 <i>A</i> の下三角部分 ( <i>uplo</i> ='L' の場合) または上三角部分 ( <i>uplo</i> ='U' の場合) が、対角成分を含めて上書きされる。
<i>w</i>	REAL (ssyev の場合) DOUBLE PRECISION (dsyev の場合) 配列、次元は $\max(1, n)$ 以上。 <i>info</i> =0 の場合、行列 <i>A</i> の固有値が昇順に格納される。
<i>work</i> (1)	終了時に、 <i>lwork</i> >0 の場合、 <i>work</i> (1) は <i>lwork</i> の必要最小サイズを返す。
<i>info</i>	INTEGER。 <i>info</i> =0 の場合、実行は正常に終了したことを示す。 <i>info</i> =- <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正であったことを示す。 <i>info</i> = <i>i</i> の場合、このアルゴリズムが収束に失敗したことを示す。 <i>i</i> は、中間の三重対角形式で、ゼロに収束しなかった成分の個数を表す。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *syev* のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ ( <i>n</i> , <i>n</i> ) の行列 <i>A</i> を格納する。
<i>w</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>job</i>	'N' または 'V' でなければならない。デフォルト値は 'N'。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。

## アプリケーション・ノート

最適なパフォーマンスを得るには、次の値を使用する。

$$lwork \geq (nb+2)*n$$

ここで、*nb* は、*ilaenv* が *?sytrd* に対して返したブロックサイズである。  
必要なワークスペースの大きさがわからない場合は、最初の実行では *lwork* を大きめの値に設定する。終了時に、*work(1)* の値を調べ、以降はその値を使用する。

---

## ?heev

エルミート行列の固有値と (オプションで)  
固有ベクトルをすべて計算する。

---

### 構文

#### Fortran 77:

```
call cheev(jobz, uplo, n, a, lda, w, work, lwork, rwork, info)
call zheev(jobz, uplo, n, a, lda, w, work, lwork, rwork, info)
```

#### Fortran 95:

```
call heev(a, w [,jobz] [,uplo] [,info])
```

### 説明

このルーチンは、エルミート行列 *A* のすべての固有値と (オプションで) 固有ベクトルをすべて計算する。

ほとんどの複素エルミート固有値問題では、基本となるアルゴリズムがより高速でワークスペースの使用が少なくなるように、デフォルトで [?heevr](#) 関数を使用する。

### 入力パラメータ

<i>jobz</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobz</i> = 'N' の場合、固有値のみを計算する。 <i>jobz</i> = 'V' の場合、固有値と固有ベクトルを計算する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> = 'U' の場合、 <i>a</i> には <i>A</i> の上三角部分を格納する。 <i>uplo</i> = 'L' の場合、 <i>a</i> には <i>A</i> の下三角部分を格納する。
<i>n</i>	INTEGER。行列 <i>A</i> の次数 ( $n \geq 0$ )。
<i>a</i> , <i>work</i>	COMPLEX ( <i>cheev</i> の場合) DOUBLE COMPLEX ( <i>zheev</i> の場合) 配列:

$a(lda,*)$  は、 $uplo$  の値に従って、エルミート行列  $A$  の上または下三角部分を格納する配列である。

$a$  の第 2 次元は、 $\max(1, n)$  以上でなければならない。

$work(lwork)$  は、ワークスペース配列である。

$lda$	INTEGER。配列 $a$ の第 1 次元。 $\max(1, n)$ 以上でなければならない。
$lwork$	INTEGER。配列 $work$ の次元。 次の制約がある。 $lwork \geq \max(1, 2n-1)$ 。 $lwork = -1$ の場合、ワークスペースのクエリとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエントリとして返す。 $xerbla$ は $lwork$ に関するエラー・メッセージを生成しない。  $lwork$ の推奨値は、「アプリケーション・ノート」を参照。
$rwork$	REAL (cheev の場合 ) DOUBLE PRECISION (zheev の場合 )。 ワークスペース配列、次元は $\max(1, 3n-2)$ 以上。

## 出力パラメータ

$a$	$jobz = 'V'$ の場合、終了時に、 $info = 0$ であれば、行列 $A$ の正規直交固有ベクトルが、配列 $a$ に格納される。 $jobz = 'N'$ の場合、終了時に、行列 $A$ の下三角部分 ( $uplo = 'L'$ の場合) または上三角部分 ( $uplo = 'U'$ の場合) が、対角成分を含めて上書きされる。
$w$	REAL (cheev の場合 ) DOUBLE PRECISION (zheev の場合 ) 配列、次元は $\max(1, n)$ 以上。 $info = 0$ の場合、行列 $A$ の固有値が昇順に格納される。
$work(1)$	終了時に、 $lwork > 0$ の場合、 $work(1)$ は $lwork$ の必要最小サイズを返す。
$info$	INTEGER。 $info = 0$ の場合、実行は正常に終了したことを示す。 $info = -i$ の場合、 $i$ 番目のパラメータの値が不正であったことを示す。 $info = i$ の場合、このアルゴリズムが収束に失敗したことを示す。 $i$ は、中間の三重対角形式で、ゼロに収束しなかった成分の個数を表す。



### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `heev` のインターフェイスの詳細を以下に示す。

<code>a</code>	サイズ $(n,n)$ の行列 $A$ を格納する。
<code>w</code>	長さ $(n)$ のベクトルを格納する。
<code>job</code>	'N' または 'V' でなければならない。デフォルト値は 'N'。
<code>uplo</code>	'U' または 'L' でなければならない。デフォルト値は 'U'。

### アプリケーション・ノート

最適なパフォーマンスを得るには、次の値を使用する。

$$lwork \geq (nb+1)*n$$

ここで、 $nb$  は、`ilaenv` が `?hetrd` に対して返したブロックサイズである。  
必要なワークスペースの大きさがわからない場合は、最初の実行では `lwork` を大きめの値に設定する。終了時に、`work(1)` の値を調べ、以降はその値を使用する。

---

## ?syevd

分割統治アルゴリズムを使用して、  
実対称行列の固有値と ( オプションで )  
固有ベクトルをすべて計算する。

---

### 構文

#### Fortran 77:

```
call ssyevd(job, uplo, n, a, lda, w, work, lwork, iwork, liwork, info)
call dsyevd(job, uplo, n, a, lda, w, work, lwork, iwork, liwork, info)
```

#### Fortran 95:

```
call syevd(a, w [,jobz] [,uplo] [,info])
```

## 説明

このルーチンは、分割統治アルゴリズムを使用して、実対称行列  $A$  の固有値と (オプションで) 固有ベクトルをすべて計算する。言い換えると、 $A$  のスペクトル因子分解  $A = Z\Lambda Z^T$  を求めることができる。 $\Lambda$  は対角成分が固有値  $\lambda_i$  であるような対角行列、 $Z$  は各列が固有ベクトル  $z_i$  であるような直交行列である。つまり、次のように表現できる。

$$Az_i = \lambda_i z_i \text{ ただし } i = 1, 2, \dots, n$$

このルーチンは、固有ベクトルも要求された場合は、分割統治アルゴリズムを使用して、固有値と固有ベクトルを計算する。ただし、固有値のみ要求された場合は、Pal-Walker-Kahan 変形版の  $QL$  または  $QR$  アルゴリズムを使用して、固有値を計算する。

ほとんどの実対称固有値問題では、基本となるアルゴリズムがより高速でワークスペースの使用が少なくなるように、デフォルトで [?syevr](#) 関数を使用する。[?syevd](#) は、より多くのワークスペースが必要となるが、特定のケース (特に大きな行列の場合) ではより高速である。

## 入力パラメータ

<i>job</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>job</i> = 'N' の場合、固有値のみを計算する。 <i>job</i> = 'V' の場合、固有値と固有ベクトルを計算する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> = 'U' の場合、 <i>a</i> には $A$ の上三角部分を格納する。 <i>uplo</i> = 'L' の場合、 <i>a</i> には $A$ の下三角部分を格納する。
<i>n</i>	INTEGER。行列 $A$ の次数 ( $n \geq 0$ )。
<i>a</i>	REAL (ssyevd の場合 ) DOUBLE PRECISION (dsyevd の場合 ) 配列、次元は ( <i>lda</i> ,*)。 <i>a</i> ( <i>lda</i> ,*) は、 <i>uplo</i> の値に従って、対称行列 $A$ の上三角部分または下三角部分を格納する配列である。 <i>a</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。
<i>lda</i>	INTEGER。配列 <i>a</i> の第 1 次元。 $\max(1, n)$ 以上でなければならない。
<i>work</i>	REAL (ssyevd の場合 ) DOUBLE PRECISION (dsyevd の場合 )。 ワークスペース配列、次元は <i>lwork</i> 以上。

<i>lwork</i>	INTEGER。配列 <i>work</i> の次元。 次の制約がある。 $n \leq 1$ の場合、 $lwork \geq 1$ 。 $job = 'N'$ で $n > 1$ の場合、 $lwork \geq 2n+1$ 。 $job = 'V'$ で $n > 1$ の場合、 $lwork \geq 3n^2 + (5+2k) * n + 1$ (ここで、 $k$ は、 $2^k \geq n$ を満たす最小の整数)。 $lwork = -1$ の場合、ワークスペースのクエリとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリとして返す。xerbla は <i>lwork</i> に関するエラー・メッセージを生成しない。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は <i>liwork</i> 以上。
<i>liwork</i>	INTEGER。配列 <i>iwork</i> の次元。 次の制約がある。 $n \leq 1$ の場合、 $liwork \geq 1$ 。 $job = 'N'$ で $n > 1$ の場合、 $liwork \geq 1$ 。 $job = 'V'$ で $n > 1$ の場合、 $liwork \geq 5n+2$ 。  $liwork = -1$ の場合、ワークスペースのクエリとみなされ、ルーチンは <i>iwork</i> 配列の最適サイズだけを計算し、その値を <i>iwork</i> 配列の最初のエントリとして返す。xerbla は <i>liwork</i> に関するエラー・メッセージを生成しない。

## 出力パラメータ

<i>w</i>	REAL (ssyevd の場合) DOUBLE PRECISION (dsyevd の場合) 配列、次元は $\max(1, n)$ 以上。 $info = 0$ の場合、行列 <i>A</i> の固有値が昇順に格納される。 <i>info</i> も参照のこと。
<i>a</i>	$job = 'V'$ の場合、終了時に、 <i>A</i> の固有ベクトルが入った直交行列 <i>Z</i> で上書きされる。
<i>work(1)</i>	終了時に、 $lwork > 0$ の場合、 <i>work(1)</i> は <i>lwork</i> の必要最小サイズを返す。
<i>iwork(1)</i>	終了時に、 $liwork > 0$ の場合、 <i>iwork(1)</i> は <i>liwork</i> の必要最小サイズを返す。
<i>info</i>	INTEGER。 $info = 0$ の場合、実行は正常に終了したことを示す。 $info = i$ の場合、このアルゴリズムが収束に失敗したことを示す。

$i$  は、中間の三重対角形式で、ゼロに収束しなかった成分の個数を表す。  
 $info = -i$  の場合、 $i$  番目のパラメータの値が不正であったことを示す。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `syevd` のインターフェイスの詳細を以下に示す。

<code>a</code>	サイズ $(n,n)$ の行列 $A$ を格納する。
<code>w</code>	長さ $(n)$ のベクトルを格納する。
<code>job</code>	'N' または 'V' でなければならない。デフォルト値は 'N'。
<code>uplo</code>	'U' または 'L' でなければならない。デフォルト値は 'U'。

### アプリケーション・ノート

計算で求めた固有値と固有ベクトルは、正確には、 $\|E\|_2 = O(\varepsilon) \|T\|_2$  のような行列  $T + E$  ( $\varepsilon$  はマシン精度) のものである。

このルーチンの複素数版は、[?heevd](#) である。

---

## ?heevd

分割統治アルゴリズムを使用して、複素エルミート行列の固有値と(オプションで)固有ベクトルをすべて計算する。

---

### 構文

#### Fortran 77:

```
call cheevd(job, uplo, n, a, lda, w, work, lwork, rwork, lrwork,  
            iwork, liwork, info)  
call zheevd(job, uplo, n, a, lda, w, work, lwork, rwork, lrwork,  
            iwork, liwork, info)
```

**Fortran 95:**

```
call heevd(a, w [,job] [,uplo] [,info])
```

**説明**

このルーチンは、分割統治アルゴリズムを使用して、複素エルミート行列  $A$  の固有値と (オプションで) 固有ベクトルをすべて計算する。言い換えると、 $A$  のスペクトル因子分解  $A = Z\Lambda Z^H$  を求めることができる。 $\Lambda$  は対角成分が固有値  $\lambda_i$  であるような実対角行列、 $Z$  は各列が固有ベクトル  $z_i$  であるような (複素) ユニタリ行列である。つまり、次のように表現できる。

$$Az_i = \lambda_i z_i \text{ ただし } i = 1, 2, \dots, n$$

このルーチンは、固有ベクトルも要求された場合は、分割統治アルゴリズムを使用して、固有値と固有ベクトルを計算する。ただし、固有値のみ要求された場合は、Pal-Walker-Kahan 変形版の  $QL$  または  $QR$  アルゴリズムを使用して、固有値を計算する。

ほとんどの複素エルミート固有値問題では、基本となるアルゴリズムがより高速でワークスペースの使用が少なくなるように、デフォルトで [?heevr](#) 関数を使用する。[?heevd](#) は、より多くのワークスペースが必要となるが、特定のケース (特に大きな行列の場合) ではより高速である。

**入力パラメータ**

<i>job</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>job</i> ='N' の場合、固有値のみを計算する。 <i>job</i> ='V' の場合、固有値と固有ベクトルを計算する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> ='U' の場合、 <i>a</i> には $A$ の上三角部分を格納する。 <i>uplo</i> ='L' の場合、 <i>a</i> には $A$ の下三角部分を格納する。
<i>n</i>	INTEGER。行列 $A$ の次数 ( $n \geq 0$ )。
<i>a</i>	COMPLEX (cheevd の場合) DOUBLE COMPLEX (zheevd の場合) 配列、次元は ( <i>lda</i> ,*)。 <i>a</i> ( <i>lda</i> ,*) は、 <i>uplo</i> の値に従って、エルミート行列 $A$ の上三角部分または下三角部分を格納する配列である。 <i>a</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。
<i>lda</i>	INTEGER。配列 <i>a</i> の第 1 次元。 $\max(1, n)$ 以上でなければならない。

<i>work</i>	COMPLEX (cheevd の場合 ) DOUBLE COMPLEX (zheevd の場合 )。 ワークスペース配列、次元は <i>lwork</i> 以上。
<i>lwork</i>	INTEGER。配列 <i>work</i> の次元。 次の制約がある。 $n \leq 1$ の場合、 $lwork \geq 1$ 。 $job = 'N'$ で $n > 1$ の場合、 $lwork \geq n+1$ 。 $job = 'V'$ で $n > 1$ の場合、 $lwork \geq n^2+2n$ 。 $lwork = -1$ の場合、ワークスペースのクエリとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリとして返す。xerbla は <i>lwork</i> に関するエラー・メッセージを生成しない。
<i>rwork</i>	REAL (cheevd の場合 ) DOUBLE PRECISION (zheevd の場合 ) ワークスペース配列、次元は <i>lrwork</i> 以上。
<i>lrwork</i>	INTEGER。配列 <i>rwork</i> の次元。 次の制約がある。 $n \leq 1$ の場合、 $lrwork \geq 1$ 。 $job = 'N'$ で $n > 1$ の場合、 $lrwork \geq n$ 。 $job = 'V'$ で $n > 1$ の場合、 $lrwork \geq 3n^2 + (4+2k) * n + 1$ (ここで、 $k$ は、 $2^k \geq n$ を満たす最小の整数)。 $lrwork = -1$ の場合、ワークスペースのクエリとみなされ、ルーチンは <i>rwork</i> 配列の最適サイズだけを計算し、その値を <i>rwork</i> 配列の最初のエントリとして返す。xerbla は <i>lrwork</i> に関するエラー・メッセージを生成しない。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は <i>liwork</i> 以上。
<i>liwork</i>	INTEGER。配列 <i>iwork</i> の次元。 次の制約がある。 $n \leq 1$ の場合、 $liwork \geq 1$ 。 $job = 'N'$ で $n > 1$ の場合、 $liwork \geq 1$ 。 $job = 'V'$ で $n > 1$ の場合、 $liwork \geq 5n+2$ 。  $liwork = -1$ の場合、ワークスペースのクエリとみなされ、ルーチンは <i>iwork</i> 配列の最適サイズだけを計算し、その値を <i>iwork</i> 配列の最初のエントリとして返す。xerbla は <i>liwork</i> に関するエラー・メッセージを生成しない。

**出力パラメータ**

<i>w</i>	REAL (cheevd の場合 ) DOUBLE PRECISION (zheevd の場合 ) 配列、次元は $\max(1, n)$ 以上。 <i>info</i> = 0 の場合、行列 <i>A</i> の固有値が昇順に格納される。 <i>info</i> も参照のこと。
<i>a</i>	<i>job</i> = 'V' の場合、終了時に、 <i>A</i> の固有ベクトルが入ったユニタリ行列 <i>Z</i> で上書きされる。
<i>work</i> (1)	終了時に、 <i>lwork</i> > 0 の場合、 <i>work</i> (1) の実数部は <i>lwork</i> の必要最小サイズを返す。
<i>rwork</i> (1)	終了時に、 <i>lrwork</i> > 0 の場合、 <i>rwork</i> (1) は <i>lrwork</i> の必要最小サイズを返す。
<i>iwork</i> (1)	終了時に、 <i>liwork</i> > 0 の場合、 <i>iwork</i> (1) は <i>liwork</i> の必要最小サイズを返す。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = <i>i</i> の場合、このアルゴリズムが収束に失敗したことを示す。 <i>i</i> は、中間の三重対角形式で、ゼロに収束しなかった成分の個数を表す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正であったことを示す。

**Fortran 95 インターフェイス・ノート**

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン heevd のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ ( <i>n</i> , <i>n</i> ) の行列 <i>A</i> を格納する。
<i>w</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>job</i>	'N' または 'V' でなければならない。デフォルト値は 'N'。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。

### アプリケーション・ノート

計算で求めた固有値と固有ベクトルは、正確には、 $\|E\|_2 = O(\varepsilon) \|A\|_2$  のような行列  $A + E$  ( $\varepsilon$  はマシン精度) のものである。

このルーチンの実数版は、[?syevd](#) である。

圧縮形式の行列に対する [?hpevd](#) と、帯形式の行列に関する [?hbevd](#) も参照のこと。

---

## ?syevx

対称行列の固有値と ( オプションで )  
固有ベクトルを選択的に計算する。

---

### 構文

#### Fortran 77:

```
call ssyevx(jobz, range, uplo, n, a, lda, vl, vu, il, iu, abstol,  
            m, w, z, ldz, work, lwork, iwork, ifail, info)  
call dsyevx(jobz, range, uplo, n, a, lda, vl, vu, il, iu, abstol,  
            m, w, z, ldz, work, lwork, iwork, ifail, info)
```

#### Fortran 95:

```
call syevx(a, w [,uplo] [,z] [,vl] [,vu] [,il] [,iu] [,m] [,ifail] [,abstol]  
[,info])
```

### 説明

このルーチンは、実対称行列  $A$  の固有値と ( オプションで ) 固有ベクトルを選択的に計算する。固有値と固有ベクトルを選択するには、希望する固有値の値の範囲かインデックスの範囲を指定する。

ほとんどの実対称固有値問題では、基本となるアルゴリズムがより高速でワークスペースの使用が少なくなるように、デフォルトで [?syevr](#) 関数を使用する。[?syevx](#) は、選択する固有値が少ない場合、より高速である。

### 入力パラメータ

**jobz** CHARACTER\*1。'N' または 'V' でなければならない。  
jobz = 'N' の場合、固有値のみを計算する。  
jobz = 'V' の場合、固有値と固有ベクトルを計算する。



<i>range</i>	CHARACTER*1。'A'、'V'、または 'I' でなければならない。 <i>range</i> = 'A' の場合、固有値をすべて計算する。 <i>range</i> = 'V' の場合、半開区間 ( <i>v1</i> , <i>vu</i> ] に含まれる固有値をすべて計算する。 <i>range</i> = 'I' の場合、インデックスが <i>il</i> ~ <i>iu</i> の固有値を計算する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> = 'U' の場合、 <i>a</i> には <i>A</i> の上三角部分を格納する。 <i>uplo</i> = 'L' の場合、 <i>a</i> には <i>A</i> の下三角部分を格納する。
<i>n</i>	INTEGER。行列 <i>A</i> の次数 ( $n \geq 0$ )。
<i>a</i> , <i>work</i>	REAL (ssyevx の場合 ) DOUBLE PRECISION (dsyevx の場合 )。 配列： <i>a</i> ( <i>lda</i> ,*) は、 <i>uplo</i> の値に従って、対称行列 <i>A</i> の上三角部分または下三角部分を格納する配列である。 <i>a</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。  <i>work</i> ( <i>lwork</i> ) は、ワークスペース配列である。
<i>lda</i>	INTEGER。配列 <i>a</i> の第 1 次元。 $\max(1, n)$ 以上でなければならない。
<i>v1</i> , <i>vu</i>	REAL (ssyevx の場合 ) DOUBLE PRECISION (dsyevx の場合 )。 <i>range</i> = 'V' の場合、固有値を探す区間の下限値と上限値。 $v1 \leq vu$ 。 <i>range</i> = 'A' または 'I' の場合は参照されない。
<i>il</i> , <i>iu</i>	INTEGER。 <i>range</i> = 'I' の場合、求める固有値の最小インデックスと最大インデックス。 次の制約がある。 $1 \leq il \leq iu \leq n$ ( $n > 0$ の場合 )。 $il=1$ かつ $iu=0$ ( $n = 0$ の場合 )。 <i>range</i> = 'A' または 'V' の場合は参照されない。
<i>abstol</i>	REAL (ssyevx の場合 ) DOUBLE PRECISION (dsyevx の場合 )。 固有値に対する絶対エラー許容値。 詳細は、「アプリケーション・ノート」を参照。
<i>ldz</i>	INTEGER。出力配列 <i>z</i> の第 1 次元。 $ldz \geq 1$ 。 <i>jobz</i> = 'V' の場合、 $ldz \geq \max(1, n)$ 。

<i>lwork</i>	INTEGER。配列 <i>work</i> の次元。 次の制約がある。 $lwork \geq \max(1, 8n)$ 。 $lwork = -1$ の場合、ワークスペースのクエリとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリとして返す。xerbla は <i>lwork</i> に関するエラー・メッセージを生成しない。  <i>lwork</i> の推奨値は、「アプリケーション・ノート」を参照。
<i>iwork</i>	INTEGER。ワークスペース配列、次元は $\max(1, 5n)$ 以上。

## 出力パラメータ

<i>a</i>	終了時に、行列 <i>A</i> の下三角部分 ( <i>uplo</i> = 'L' の場合) または上三角部分 ( <i>uplo</i> = 'U' の場合) が、対角成分を含めて上書きされる。
<i>m</i>	INTEGER。検出された固有値の総数。 $(0 \leq m \leq n)$ 。 <i>range</i> = 'A' の場合は $m = n$ 、 <i>range</i> = 'I' の場合は $m = iu - il + 1$ 。
<i>w</i>	REAL (ssyevx の場合) DOUBLE PRECISION (dsyevx の場合) 配列、次元は $\max(1, n)$ 以上。 先頭の <i>m</i> 個の成分に、行列 <i>A</i> の選択された固有値が昇順に格納される。
<i>z</i>	REAL (ssyevx の場合) DOUBLE PRECISION (dsyevx の場合)。 配列 <i>z</i> ( <i>ldz</i> ,*) には、固有ベクトルが格納される。 <i>z</i> の第 2 次元は、 $\max(1, m)$ 以上でなければならない。  <i>jobz</i> = 'V' の場合、 <i>info</i> = 0 であれば、 <i>z</i> の先頭の <i>m</i> 列に、行列 <i>A</i> の選択された固有値に対応する正規直交固有ベクトルが格納される。すなわち、 <i>w</i> ( <i>i</i> ) に対応する固有ベクトルが、 <i>z</i> の <i>i</i> 番目の列に格納される。固有ベクトルが収束できない場合、 <i>z</i> のその列には、その固有ベクトルに対する最新の近似値が格納され、固有ベクトルのインデックスは <i>ifail</i> に返される。 <i>jobz</i> = 'N' の場合、 <i>z</i> は参照されない。 注: 配列 <i>z</i> には、 $\max(1, m)$ 以上の列が提供されなければならない。 <i>range</i> = 'V' の場合、 <i>m</i> の正確な値が事前にわからないため、上限値を使用する必要がある。
<i>work</i> (1)	終了時に、 $lwork > 0$ の場合、 <i>work</i> (1) は <i>lwork</i> の必要最小サイズを返す。

<i>ifail</i>	INTEGER。配列、次元は $\max(1, n)$ 以上。 <i>jobz</i> = 'V' の場合、 <i>info</i> = 0 であれば、 <i>ifail</i> の先頭から <i>m</i> 個の成分はゼロになる。 <i>info</i> > 0 であれば、収束に失敗した固有ベクトルのインデックスが <i>ifail</i> に格納される。 <i>jobz</i> = 'V' の場合、 <i>ifail</i> は参照されない。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正であったことを示す。 <i>info</i> = <i>i</i> の場合、 <i>i</i> 番目の固有ベクトルが収束に失敗した。その固有ベクトルのインデックスが配列 <i>ifail</i> に格納される。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *syevx* のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ $(n, n)$ の行列 <i>A</i> を格納する。
<i>w</i>	長さ $(n)$ のベクトルを格納する。
<i>a</i>	サイズ $(m, n)$ の行列 <i>A</i> を格納する。
<i>ifail</i>	長さ $(n)$ のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>v1</i>	この成分のデフォルト値は、 <i>v1</i> = -HUGE( <i>v1</i> ) である。
<i>vu</i>	この成分のデフォルト値は、 <i>vu</i> = HUGE( <i>v1</i> ) である。
<i>il</i>	この引数のデフォルト値は、 <i>il</i> = 1 である。
<i>iu</i>	この引数のデフォルト値は、 <i>iu</i> = <i>n</i> である。
<i>abstol</i>	この成分のデフォルト値は、 <i>abstol</i> = 0.0_WP である。
<i>jobz</i>	引数 <i>z</i> の存在に基づいて以下のように復元される。 <i>jobz</i> = 'V' ( <i>z</i> が存在する場合)、 <i>jobz</i> = 'N' ( <i>z</i> が省略された場合)。 <i>ifail</i> が存在し、 <i>z</i> が省略された場合、エラー条件がセットされる。

*range*                    引数 *vl*、*vu*、*il*、および *iu* の存在に基づいて以下のように復元される。

*range* = 'V' (*vl* と *vu* のいずれかまたは両方が存在する場合)、  
*range* = 'I' (*il* と *iu* のいずれかまたは両方が存在する場合)、  
*range* = 'A' (*vl*、*vu*、*il*、および *iu* がすべて存在しない場合。  
*vl* と *vu* のいずれかまたは両方が存在し、同時に *il* と *iu* のいずれかまたは両方が存在する場合、エラー条件がセットされる。

### アプリケーション・ノート

最適なパフォーマンスを得るには、次の値を使用する。

$$lwork \geq (nb+3)*n$$

ここで、*nb* は、*ilaenv* が ?sytrd と ?ormtr に対して返したブロックサイズである。必要なワークスペースの大きさがわからない場合は、最初の実行では *lwork* を大きめの値に設定する。終了時に、*work*(1) の値を調べ、以降はその値を使用する。

$abstol + \epsilon * \max(|a|, |b|)$  以下の幅の区間  $[a, b]$  内に存在していると判別された場合 ( $\epsilon$  はマシン精度)、近似固有値は、収束したものとして受け入れられる。*abstol* がゼロまたは負の場合、代わりに  $\epsilon * |T|$  を使用する。 $|T|$  は、*A* を三重対角形式に縮退して得られた三重対角行列の 1- ノルムである。

固有値が最も正確に計算されるのは、*abstol* がゼロでなく、アンダーフローのしきい値の 2 倍、 $2 * slamch('S')$  に設定されたときである。このルーチンで *info* > 0 が返された場合、固有ベクトルのいくつか収束に失敗したことを意味するので、*abstol* を  $2 * slamch('S')$  に設定して、やり直してみる。

---

## ?heevx

エルミート行列の固有値と ( オプションで )  
固有ベクトルを選択的に計算する。

---

### 構文

#### Fortran 77:

```
call cheevx(jobz, range, uplo, n, a, lda, vl, vu, il, iu, abstol,  
            m, w, z, ldz, work, lwork, rwork, iwork, ifail, info)  
call zheevx(jobz, range, uplo, n, a, lda, vl, vu, il, iu, abstol,  
            m, w, z, ldz, work, lwork, rwork, iwork, ifail, info)
```

**Fortran 95:**

```
call heevx(a, w [,uplo] [,z] [,vl] [,vu] [,il] [,iu] [,m] [,ifail] [,abstol]
[,info])
```

**説明**

このルーチンは、複素エルミート行列  $A$  の固有値と (オプションで) 固有ベクトルを選択的に計算する。固有値と固有ベクトルを選択するには、希望する固有値の値の範囲かインデックスの範囲を指定する。

ほとんどの複素エルミート固有値問題では、基本となるアルゴリズムがより高速でワークスペースの使用が少なくなるように、デフォルトで [zheevr](#) 関数を使用する。`?heevx` は、選択する固有値が少ない場合、より高速である。

**入力パラメータ**

<i>jobz</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobz</i> = 'N' の場合、固有値のみを計算する。 <i>jobz</i> = 'V' の場合、固有値と固有ベクトルを計算する。
<i>range</i>	CHARACTER*1。'A'、'V'、または 'I' でなければならない。 <i>range</i> = 'A' の場合、固有値をすべて計算する。 <i>range</i> = 'V' の場合、半开区間 ( <i>vl</i> , <i>vu</i> ] に含まれる固有値をすべて計算する。 <i>range</i> = 'I' の場合、インデックスが <i>il</i> ~ <i>iu</i> の固有値を計算する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> = 'U' の場合、 <i>a</i> には $A$ の上三角部分を格納する。 <i>uplo</i> = 'L' の場合、 <i>a</i> には $A$ の下三角部分を格納する。
<i>n</i>	INTEGER。行列 $A$ の次数 ( $n \geq 0$ )。
<i>a</i> , <i>work</i>	COMPLEX ( <code>cheevx</code> の場合) DOUBLE COMPLEX ( <code>zheevx</code> の場合)。 配列: <i>a</i> ( <i>lda</i> ,*) は、 <i>uplo</i> の値に従って、エルミート行列 $A$ の上三角部分または下三角部分を格納する配列である。 <i>a</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>work</i> ( <i>lwork</i> ) は、ワークスペース配列である。
<i>lda</i>	INTEGER。配列 <i>a</i> の第 1 次元。 $\max(1, n)$ 以上でなければならない。

<i>vl, vu</i>	REAL (cheevx の場合 ) DOUBLE PRECISION (zheevx の場合 )。 <i>range</i> = 'V' の場合、固有値を探す区間の下限値と上限値。 $vl \leq vu$ 。 <i>range</i> = 'A' または 'I' の場合は参照されない。
<i>il, iu</i>	INTEGER。 <i>range</i> = 'I' の場合、求める固有値の最小インデックスと最大インデックス。 次の制約がある。 $1 \leq il \leq iu \leq n$ ( $n > 0$ の場合 )。 $il=1$ かつ $iu=0$ ( $n = 0$ の場合 )。 <i>range</i> = 'A' または 'V' の場合は参照されない。
<i>abstol</i>	REAL (cheevx の場合 ) DOUBLE PRECISION (zheevx の場合 )。 固有値に対する絶対エラー許容値。 詳細は、「アプリケーション・ノート」を参照。
<i>ldz</i>	INTEGER。 出力配列 <i>z</i> の第 1 次元。 $ldz \geq 1$ 。 <i>jobz</i> = 'V' の場合、 $ldz \geq \max(1, n)$ 。
<i>lwork</i>	INTEGER。 配列 <i>work</i> の次元。 次の制約がある。 $lwork \geq \max(1, 2n-1)$ 。 <i>lwork</i> = -1 の場合、ワークスペースのクエリとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリとして返す。xerbla は <i>lwork</i> に関するエラー・メッセージを生成しない。  <i>lwork</i> の推奨値は、「アプリケーション・ノート」を参照。
<i>rwork</i>	REAL (cheevx の場合 ) DOUBLE PRECISION (zheevx の場合 )。 ワークスペース配列、次元は $\max(1, 7n)$ 以上。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, 5n)$ 以上。

## 出力パラメータ

<i>a</i>	終了時に、行列 <i>A</i> の下三角部分 ( <i>uplo</i> = 'L' の場合 ) または上三角部分 ( <i>uplo</i> = 'U' の場合 ) が、対角成分を含めて上書きされる。
<i>m</i>	INTEGER。 検出された固有値の総数。 $(0 \leq m \leq n)$ 。 <i>range</i> = 'A' の場合は $m = n$ 、 <i>range</i> = 'I' の場合は $m = iu - il + 1$ 。

<i>w</i>	<p>REAL (cheevx の場合 )  DOUBLE PRECISION (zheevx の場合 )  配列、次元は <math>\max(1, n)</math> 以上。  先頭の <math>m</math> 個の成分に、行列 <math>A</math> の選択された固有値が昇順に格納される。</p>
<i>z</i>	<p>COMPLEX (cheevx の場合 )  DOUBLE COMPLEX (zheevx の場合 )。  配列 <math>z(ldz, *)</math> には、固有ベクトルが格納される。  <math>z</math> の第 2 次元は、<math>\max(1, m)</math> 以上でなければならない。</p> <p><math>jobz = 'V'</math> の場合、<math>info = 0</math> であれば、<math>z</math> の先頭の <math>m</math> 列に、行列 <math>A</math> の選択された固有値に対応する正規直交固有ベクトルが格納される。すなわち、<math>w(i)</math> に対応する固有ベクトルが、<math>z</math> の <math>i</math> 番目の列に格納される。固有ベクトルが収束できない場合、<math>z</math> のその列には、その固有ベクトルに対する最新の近似値が格納され、固有ベクトルのインデックスは <i>ifail</i> に返される。  <math>jobz = 'N'</math> の場合、<math>z</math> は参照されない。  注：配列 <math>z</math> には、<math>\max(1, m)</math> 以上の列が提供されなければならない。  <math>range = 'V'</math> の場合、<math>m</math> の正確な値が事前にわからないため、上限値を使用する必要がある。</p>
<i>work(1)</i>	<p>終了時に、<math>lwork &gt; 0</math> の場合、<i>work(1)</i> は <math>lwork</math> の必要最小サイズを返す。</p>
<i>ifail</i>	<p>INTEGER。配列、次元は <math>\max(1, n)</math> 以上。  <math>jobz = 'V'</math> の場合、<math>info = 0</math> であれば、<i>ifail</i> の先頭から <math>m</math> 個の成分はゼロになる。<math>info &gt; 0</math> であれば、収束に失敗した固有ベクトルのインデックスが <i>ifail</i> に格納される。  <math>jobz = 'V'</math> の場合、<i>ifail</i> は参照されない。</p>
<i>info</i>	<p>INTEGER。  <math>info = 0</math> の場合、実行は正常に終了したことを示す。  <math>info = -i</math> の場合、<math>i</math> 番目のパラメータの値が不正であったことを示す。  <math>info = i</math> の場合、<math>i</math> 番目の固有ベクトルが収束に失敗した。その固有ベクトルのインデックスが配列 <i>ifail</i> に格納される。</p>

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `heevx` のインターフェイスの詳細を以下に示す。

<code>a</code>	サイズ $(n,n)$ の行列 $A$ を格納する。
<code>w</code>	長さ $(n)$ のベクトルを格納する。
<code>z</code>	サイズ $(n,n)$ の行列 $Z$ を格納する。
<code>ifail</code>	長さ $(n)$ のベクトルを格納する。
<code>uplo</code>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<code>v1</code>	この成分のデフォルト値は、 $v1 = -HUGE(v1)$ である。
<code>vu</code>	この成分のデフォルト値は、 $vu = HUGE(v1)$ である。
<code>il</code>	この引数のデフォルト値は、 $il = 1$ である。
<code>iu</code>	この引数のデフォルト値は、 $iu = n$ である。
<code>abstol</code>	この成分のデフォルト値は、 $abstol = 0.0\_WP$ である。
<code>jobz</code>	引数 $z$ の存在に基づいて以下のように復元される。 $jobz = 'V'$ ( $z$ が存在する場合)、 $jobz = 'N'$ ( $z$ が省略された場合)。 $ifail$ が存在し、 $z$ が省略された場合、エラー条件がセットされる。
<code>range</code>	引数 $v1$ 、 $vu$ 、 $il$ 、および $iu$ の存在に基づいて以下のように復元される。 $range = 'V'$ ( $v1$ と $vu$ のいずれかまたは両方が存在する場合)、 $range = 'I'$ ( $il$ と $iu$ のいずれかまたは両方が存在する場合)、 $range = 'A'$ ( $v1$ 、 $vu$ 、 $il$ 、および $iu$ がすべて存在しない場合)。 $v1$ と $vu$ のいずれかまたは両方が存在し、同時に $il$ と $iu$ のいずれかまたは両方が存在する場合、エラー条件がセットされる。

## アプリケーション・ノート

最適なパフォーマンスを得るには、次の値を使用する。

$$lwork \geq (nb+1)*n$$

ここで、 $nb$  は、`ilaenv` が `?hetrd` と `?unmtr` に対して返したブロックサイズである。必要なワークスペースの大きさがわからない場合は、最初の実行では  $lwork$  を大きめの値に設定する。終了時に、`work(1)` の値を調べ、以降はその値を使用する。

$abstol + \epsilon * \max(|a|, |b|)$  以下の幅の区間  $[a, b]$  内に存在していると判別された場合 ( $\epsilon$  はマシン精度)、近似固有値は、収束したものとして受け入れられる。 $abstol$  がゼロまたは負の場合、代わりに  $\epsilon * |T|$  を使用する。 $|T|$  は、 $A$  を三重対角形式に縮退して得られた三重対角行列の 1- ノルムである。



固有値が最も正確に計算されるのは、`abstol` がゼロでなく、アンダーフローのしきい値の 2 倍、`2*slamch('S')` に設定されたときである。このルーチンで `info > 0` が返された場合、固有ベクトルのいくつか収束に失敗したことを意味するので、`abstol` を `2*slamch('S')` に設定して、やり直してみる。

---

## ?syevr

「比較的安定な表現」を使用して、実対称行列の固有値と (オプションで) 固有ベクトルを選択的に計算する。

---

### 構文

#### Fortran 77:

```
call ssyevr(jobz, range, uplo, n, a, lda, vl, vu, il, iu, abstol,  
            m, w, z, ldz, isuppz, work, lwork, iwork, liwork, info)  
call dsyevr(jobz, range, uplo, n, a, lda, vl, vu, il, iu, abstol,  
            m, w, z, ldz, isuppz, work, lwork, iwork, liwork, info)
```

#### Fortran 95:

```
call syevr(a, w [,uplo] [,z] [,vl] [,vu] [,il] [,iu] [,m] [,isuppz] [,abstol]  
           [,info])
```

### 説明

このルーチンは、実対称行列  $T$  の固有値と (オプションで) 固有ベクトルを選択的に計算する。固有値と固有ベクトルを選択するには、希望する固有値の値の範囲かインデックスの範囲を指定する。

可能であれば、?syevr は [sstegr/dstegr](#) を呼び出し、「比較的安定な表現」を使用して、固有スペクトルを計算する。?stegr は、固有値の計算には *dqds* アルゴリズムを使用するが、直交固有ベクトルは、種々の「良好な」 $LDL^T$  表現 (「比較的安定な表現」と

も呼ばれる) から計算する。グラム - シュミットの直交化はできる限り避ける。アルゴリズムの各ステップを詳しく説明すると次のようになる。

$T$  のまだ縮退されていない  $i$  番目のブロックに対して、

- (a)  $L_i D_i L_i^T$  が比較的安定な表現になるように、 $T - s_i = L_i D_i L_i^T$  を計算する。
- (b)  $dqds$  アルゴリズムによって、 $L_i D_i L_i^T$  の固有値  $\lambda_j$  を高い相対精度で計算する。
- (c) 値が近い固有値からなるクラスタが存在する場合は、そのクラスタに近い  $s_i$  を「選択」し、ステップ (a) に戻る。
- (d)  $L_i D_i L_i^T$  の近似的な固有値  $\lambda_j$  に対する固有ベクトルを、階数表示のツイスト分解によって求める。

出力値に必要な精度は、入力パラメータ `abstol` で指定できる。

ルーチン `?syevr` は、IEEE-754 の浮動小数点標準に準拠しているマシンでフル・スペクトルが要求された場合は、[sstegr/dstegr](#) を呼び出す。`?syevr` は、IEEE-754 に準拠しないマシンの場合や、部分スペクトルが要求された場合に、[sstebz/dstebz](#) と [sstein/dstein](#) を呼び出す。

ほとんどの実対称固有値問題では、基本となるアルゴリズムがより高速でワークスペースの使用が少なくなるように、`?syevr` を使用する。

## 入力パラメータ

<code>jobz</code>	CHARACTER*1。'N' または 'V' でなければならない。 <code>jobz = 'N'</code> の場合、固有値のみを計算する。 <code>jobz = 'V'</code> の場合、固有値と固有ベクトルを計算する。
<code>range</code>	CHARACTER*1。'A'、'V'、または 'I' のいずれかでなければならない。 <code>range = 'A'</code> の場合、固有値をすべて計算する。 <code>range = 'V'</code> の場合、半開区間 $v_l < \lambda_i \leq v_u$ における固有値 $\lambda_i$ を計算する。 <code>range = 'I'</code> の場合、ルーチンはインデックスが $i_l \sim i_u$ である固有値を計算する。  <code>range = 'V'</code> または 'I' で、 $i_u - i_l < n - 1$ の場合、 <a href="#">sstebz/dstebz</a> と <a href="#">sstein/dstein</a> を呼び出す。
<code>uplo</code>	CHARACTER*1。'U' または 'L' でなければならない。 <code>uplo = 'U'</code> の場合、 $a$ には $A$ の上三角部分を格納する。 <code>uplo = 'L'</code> の場合、 $a$ には $A$ の下三角部分を格納する。
<code>n</code>	INTEGER。行列 $A$ の次数 ( $n \geq 0$ )。

<i>a, work</i>	REAL (ssyevr の場合 ) DOUBLE PRECISION (dsyevr の場合 )。 配列 : <i>a</i> ( <i>lda</i> ,*) は、 <i>uplo</i> の値に従って、対称行列 <i>A</i> の上三角部分または下三角部分を格納する配列である。 <i>a</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>work</i> ( <i>lwork</i> ) は、ワークスペース配列である。
<i>lda</i>	INTEGER。配列 <i>a</i> の第 1 次元。 $\max(1, n)$ 以上でなければならない。
<i>vl, vu</i>	REAL (ssyevr の場合 ) DOUBLE PRECISION (dsyevr の場合 )。 <i>range</i> = 'V' の場合、固有値を検索する区間の上限と下限。 次の制約がある。 $vl < vu$ 。  <i>range</i> = 'A' または 'I' の場合、 <i>vl</i> と <i>vu</i> は参照されない。
<i>il, iu</i>	INTEGER。 <i>range</i> = 'I' の場合、返される最小固有値と最大固有値に対する昇順のインデックス。 次の制約がある。 $1 \leq il \leq iu \leq n$ ( $n > 0$ の場合 )。 $il=1$ かつ $iu=0$ ( $n=0$ の場合 )。  <i>range</i> = 'A' または 'V' の場合、 <i>il</i> と <i>iu</i> は参照されない。
<i>abstol</i>	REAL (ssyevr の場合 ) DOUBLE PRECISION (dsyevr の場合 )。 各固有値および固有ベクトルに許される絶対誤差許容値。 <i>jobz</i> = 'V' の場合、出力される固有値と固有ベクトルの誤差ノルムが <i>abstol</i> 以内になる。また、異なる固有ベクトル間の内積も <i>abstol</i> 以内になる。 $abstol < n\epsilon\ T\ _1$ の場合、 $n\epsilon\ T\ _1$ が代わりに使用される ( $\epsilon$ はマシン精度)。固有値は、 <i>abstol</i> とは無関係に、 $\epsilon\ T\ _1$ の精度で計算される。高い相対精度が必要な場合は、 <i>abstol</i> を ?lamch('S') に設定する。
<i>ldz</i>	INTEGER。出力配列 <i>z</i> のリーディング・ディメンジョン。次の制約がある。 $ldz \geq 1$ ( <i>jobz</i> = 'N' の場合 )。 $ldz \geq \max(1, n)$ ( <i>jobz</i> = 'V' の場合 )。

*lwork* INTEGER。配列 *work* の次元。  
次の制約がある。 $lwork \geq \max(1, 26n)$ 。  
 $lwork = -1$  の場合、ワークスペースのクエリとみなされ、ルーチンは *work* 配列の最適サイズだけを計算し、その値を *work* 配列の最初のエントリとして返す。xerbla は *lwork* に関するエラー・メッセージを生成しない。

*lwork* の推奨値は、「アプリケーション・ノート」を参照。

*iwork* INTEGER。ワークスペース配列、次元は (*liwork*)。

*liwork* INTEGER。配列 *iwork* の次元。 $lwork \geq \max(1, 10n)$

$liwork = -1$  の場合、ワークスペースのクエリとみなされ、ルーチンは *iwork* 配列の最適サイズだけを計算し、その値を *iwork* 配列の最初のエントリとして返す。xerbla は *liwork* に関するエラー・メッセージを生成しない。

## 出力パラメータ

*a* 終了時に、行列 *A* の下三角部分 (*uplo* = 'L' の場合) または上三角部分 (*uplo* = 'U' の場合) が、対角成分を含めて上書きされる。

*m* INTEGER。検出された固有値の総数。 $(0 \leq m \leq n)$ 。 *range* = 'A' の場合、 $m = n$ 、 *range* = 'I' の場合、 $m = iu - il + 1$ 。

*w*, *z* REAL (ssyevr の場合)  
DOUBLE PRECISION (dsyevr の場合)。  
配列:  
*w*(\*)、次元は  $\max(1, n)$  以上。選択された固有値が昇順に  $w(1) \sim w(m)$  に格納される。

*z*(*ldz*, \*)。 *z* の第 2 次元は、 $\max(1, m)$  以上でなければならない。  
*jobz* = 'V' の場合、*info* = 0 であれば、*z* の先頭の *m* 列に、行列 *T* の選択された固有値に対応する正規直交固有ベクトルが格納される。すなわち、*w*(*i*) に対応する固有ベクトルが、*z* の *i* 番目の列に格納される。  
*jobz* = 'N' の場合、*z* は参照されない。  
注: 配列 *z* には、 $\max(1, m)$  以上の列が提供されなければならない。  
*range* = 'V' の場合、*m* の正確な値が事前にわからないため、上限値を使用する必要がある。

*isuppz* INTEGER。  
配列、次元は  $2 * \max(1, m)$  以上。

$z$  に格納されている固有ベクトルのサポート情報、すなわち  $z$  に格納されている非ゼロの成分を示すインデックス。 $i$  番目の固有ベクトルは、 $isuppz(2i-1)$  から  $isuppz(2i)$  までの成分のみ非ゼロである。 $range = 'A'$  または  $range = 'I'$  で  $iu-il = n-1$  の場合にのみ有効である。

$work(1)$  終了時に、 $info = 0$  の場合、 $work(1)$  は  $lwork$  の必要最小サイズを返す。

$iwork(1)$  終了時に、 $info = 0$  の場合、 $iwork(1)$  は  $liwork$  の必要最小サイズを返す。

$info$  INTEGER。  
 $info = 0$  の場合、実行は正常に終了したことを示す。  
 $info = -i$  の場合、 $i$  番目のパラメータの値が不正であったことを示す。  
 $info = i$  の場合、内部エラーが発生したことを示す。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `syevr` のインターフェイスの詳細を以下に示す。

$a$  サイズ  $(n,n)$  の行列  $A$  を格納する。

$w$  長さ  $(n)$  のベクトルを格納する。

$z$  サイズ  $(n,m)$  の行列  $Z$  を格納する。ここで、値  $n$  と  $m$  は有意である。

$isuppz$  長さ  $(2*m)$  のベクトルを格納する。ここで、値  $(2*m)$  は有意である。

$uplo$  'U' または 'L' でなければならない。デフォルト値は 'U'。

$v1$  この成分のデフォルト値は、 $v1 = -HUGE(v1)$  である。

$vu$  この成分のデフォルト値は、 $vu = HUGE(v1)$  である。

$il$  この引数のデフォルト値は、 $il = 1$  である。

$iu$  この引数のデフォルト値は、 $iu = n$  である。

$abstol$  この成分のデフォルト値は、 $abstol = 0.0\_WP$  である。

<i>jobz</i>	引数 <i>z</i> の存在に基づいて以下のように復元される。 <i>jobz</i> = 'V' ( <i>z</i> が存在する場合)、 <i>jobz</i> = 'N' ( <i>z</i> が省略された場合)。 <i>isuppz</i> が存在し、 <i>z</i> が省略された場合、エラー条件がセットされる。
<i>range</i>	引数 <i>vl</i> 、 <i>vu</i> 、 <i>il</i> 、および <i>iu</i> の存在に基づいて以下のように復元される。 <i>range</i> = 'V' ( <i>vl</i> と <i>vu</i> のいずれかまたは両方が存在する場合)、 <i>range</i> = 'I' ( <i>il</i> と <i>iu</i> のいずれかまたは両方が存在する場合)、 <i>range</i> = 'A' ( <i>vl</i> 、 <i>vu</i> 、 <i>il</i> 、および <i>iu</i> がすべて存在しない場合)。 <i>vl</i> と <i>vu</i> のいずれかまたは両方が存在し、同時に <i>il</i> と <i>iu</i> のいずれかまたは両方が存在する場合、エラー条件がセットされる。

### アプリケーション・ノート

最適なパフォーマンスを得るには、次の値を使用する。

$$lwork \geq (nb+6)*n$$

ここで、*nb* は、*ilaenv* が *?sytrd* と *?ormtr* に対して返したブロックサイズである。  
必要なワークスペースの大きさがわからない場合は、最初の実行では *lwork* を大きめの値に設定する。終了時に、*work(1)* の値を調べ、以降はその値を使用する。

通常の *?stegr* の実行で NaN と無限大が発生するため、NaN と無限大が IEEE 標準のデフォルト方法以外で処理される環境では、浮動小数点例外によって異常終了する可能性がある。

---

## ?heevr

「比較的安定な表現」を使用して、エルミート行列の固有値と (オプションで) 固有ベクトルを選択的に計算する。

---

### 構文

#### Fortran 77:

```
call cheevr(jobz, range, uplo, n, a, lda, vl, vu, il, iu, abstol, m, w, z, ldz,  
            isuppz, work, lwork, rwork, lrwork, iwork, liwork, info)  
call zheevr(jobz, range, uplo, n, a, lda, vl, vu, il, iu, abstol, m, w, z, ldz,  
            isuppz, work, lwork, rwork, lrwork, iwork, liwork, info)
```

**Fortran 95:**

```
call heevr(a, w [,uplo] [,z] [,vl] [,vu] [,il] [,iu] [,m] [,isuppz] [,abstol]
[,info])
```

**説明**

このルーチンは、複素エルミート行列  $T$  の固有値と (オプションで) 固有ベクトルを選択的に計算する。固有値と固有ベクトルを選択するには、希望する固有値の値の範囲かインデックスの範囲を指定する。

可能であれば、`?heevr` は [cstegr/zstegr](#) を呼び出し、「比較的安定な表現」を使用して、固有スペクトルを計算する。`?stegr` は、固有値の計算には *dqds* アルゴリズムを使用するが、直交固有ベクトルは、種々の「良好な」 $LDL^T$  表現 (「比較的安定な表現」とも呼ばれる) から計算する。グラム-シュミットの直交化はできる限り避ける。アルゴリズムの各ステップを詳しく説明すると次のようになる。 $T$  のまだ縮退されていない  $i$  番目のブロックに対して、

- (a)  $L_i D_i L_i^T$  が比較的安定な表現になるように、 $T - s_i = L_i D_i L_i^T$  を計算する。
- (b) *dqds* アルゴリズムによって、 $L_i D_i L_i^T$  の固有値  $\lambda_j$  を高い相対精度で計算する。
- (c) 値が近い固有値からなるクラスタが存在する場合は、そのクラスタに近い  $s_i$  を「選択」し、ステップ (a) に戻る。
- (d)  $L_i D_i L_i^T$  の近似的な固有値  $\lambda_j$  に対する固有ベクトルを、階数表示のツイスト分解によって求める。

出力値に必要な精度は、入力パラメータ `abstol` で指定できる。

ルーチン `?heevr` は、IEEE-754 の浮動小数点標準に準拠しているマシンにおいてフル・スペクトルが要求された場合は、[cstegr/zstegr](#) を呼び出す。`?heevr` は、IEEE-754 に準拠しないマシンの場合や、部分スペクトルが要求された場合に、[sstebz/dstebz](#) と [cstein/zstein](#) を呼び出す。

ほとんどの複素エルミート固有値問題では、基本となるアルゴリズムがより高速でワークスペースの使用が少なくなるように、`?heevr` を使用する。

**入力パラメータ**

<code>jobz</code>	CHARACTER*1。'N' または 'V' でなければならない。 <code>job='N'</code> の場合、固有値のみを計算する。 <code>job='V'</code> の場合、固有値と固有ベクトルを計算する。
<code>range</code>	CHARACTER*1。'A'、'V'、または 'I' のいずれかでなければならない。

	<p><math>range = 'A'</math> の場合、固有値をすべて計算する。</p> <p><math>range = 'V'</math> の場合、半開区間 <math>v1 &lt; \lambda_i \leq vu</math> における固有値 <math>\lambda_i</math> を計算する。</p> <p><math>range = 'I'</math> の場合、ルーチンはインデックスが <math>i1 \sim iu</math> である固有値を計算する。</p> <p><math>range = 'V'</math> または <math>'I'</math> の場合、<code>sstebz/dstebz</code> と <code>cstein/zstein</code> を呼び出す。</p>
<code>uplo</code>	<p>CHARACTER*1。'U' または 'L' でなければならない。</p> <p><math>uplo = 'U'</math> の場合、<math>a</math> には <math>A</math> の上三角部分を格納する。</p> <p><math>uplo = 'L'</math> の場合、<math>a</math> には <math>A</math> の下三角部分を格納する。</p>
<code>n</code>	INTEGER。行列 $A$ の次数 ( $n \geq 0$ )。
<code>a, work</code>	<p>COMPLEX (cheevr の場合 )</p> <p>DOUBLE COMPLEX (zheevr の場合 )。</p> <p>配列 :</p> <p><math>a(lda,*)</math> は、<math>uplo</math> の値に従って、エルミート行列 <math>A</math> の上三角部分または下三角部分を格納する配列である。</p> <p><math>a</math> の第 2 次元は、<math>\max(1, n)</math> 以上でなければならない。</p> <p><math>work(lwork)</math> は、ワークスペース配列である。</p>
<code>lda</code>	<p>INTEGER。配列 <math>a</math> の第 1 次元。</p> <p><math>\max(1, n)</math> 以上でなければならない。</p>
<code>v1, vu</code>	<p>REAL (cheevr の場合 )</p> <p>DOUBLE PRECISION (zheevr の場合 )。</p> <p><math>range = 'V'</math> の場合、固有値を検索する区間の上限と下限。次の制約がある。 <math>v1 &lt; vu</math>。</p> <p><math>range = 'A'</math> または <math>'I'</math> の場合、<math>v1</math> と <math>vu</math> は参照されない。</p>
<code>i1, iu</code>	<p>INTEGER。</p> <p><math>range = 'I'</math> の場合、返される最小固有値と最大固有値に対する昇順のインデックス。</p> <p>次の制約がある。</p> <p><math>1 \leq i1 \leq iu \leq n</math> (<math>n &gt; 0</math> の場合 )。</p> <p><math>i1=1</math> かつ <math>iu=0</math> (<math>n=0</math> の場合 )。</p> <p><math>range = 'A'</math> または <math>'V'</math> の場合、<math>i1</math> と <math>iu</math> は参照されない。</p>
<code>abstol</code>	<p>REAL (cheevr の場合 )</p> <p>DOUBLE PRECISION (zheevr の場合 )。</p> <p>各固有値および固有ベクトルに許される絶対誤差許容値。</p>



$jobz = 'V'$  の場合、出力される固有値と固有ベクトルの誤差ノルムが  $abstol$  以内になる。また、異なる固有ベクトル間の内積も  $abstol$  以内になる。 $abstol < n\epsilon\|T\|_1$  の場合、 $n\epsilon\|T\|_1$  が代わりに使用される ( $\epsilon$  はマシン精度)。固有値は、 $abstol$  とは無関係に、 $\epsilon\|T\|_1$  の精度で計算される。高い相対精度が必要な場合は、 $abstol$  を `?lamch('S')` に設定する。

<i>ldz</i>	INTEGER。出力配列 <i>z</i> のリーディング・ディメンジョン。次の制約がある。 $ldz \geq 1$ ( $jobz = 'N'$ の場合)。 $ldz \geq \max(1, n)$ ( $jobz = 'V'$ の場合)。
<i>lwork</i>	INTEGER。配列 <i>work</i> の次元。 次の制約がある。 $lwork \geq \max(1, 2n)$ 。 $lwork = -1$ の場合、ワークスペースのクエリとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエン트리として返す。 <i>xerbla</i> は <i>lwork</i> に関するエラー・メッセージを生成しない。  <i>lwork</i> の推奨値は、「アプリケーション・ノート」を参照。
<i>rwork</i>	REAL ( <i>cheevr</i> の場合) DOUBLE PRECISION ( <i>zheevr</i> の場合)。 ワークスペース配列、次元は ( <i>lrwork</i> )。
<i>lrwork</i>	INTEGER。配列 <i>rwork</i> の次元。 $lwork \geq \max(1, 24n)$ 。 $lrwork = -1$ の場合、ワークスペースのクエリとみなされ、ルーチンは <i>rwork</i> 配列の最適サイズだけを計算し、その値を <i>rwork</i> 配列の最初のエン트리として返す。 <i>xerbla</i> は <i>lrwork</i> に関するエラー・メッセージを生成しない。
<i>iwork</i>	INTEGER。ワークスペース配列、次元は ( <i>liwork</i> )。
<i>liwork</i>	INTEGER。配列 <i>iwork</i> の次元。 $lwork \geq \max(1, 10n)$ 。 $liwork = -1$ の場合、ワークスペースのクエリとみなされ、ルーチンは <i>iwork</i> 配列の最適サイズだけを計算し、その値を <i>iwork</i> 配列の最初のエン트리として返す。 <i>xerbla</i> は <i>liwork</i> に関するエラー・メッセージを生成しない。

## 出力パラメータ

<i>a</i>	終了時に、行列 <i>A</i> の下三角部分 ( $uplo = 'L'$ の場合) または上三角部分 ( $uplo = 'U'$ の場合) が、対角成分を含めて上書きされる。
----------	--

<i>m</i>	INTEGER。検出された固有値の総数。 $(0 \leq m \leq n)$ 。 <i>range</i> = 'A' の場合、 $m = n$ 、 <i>range</i> = 'I' の場合、 $m = iu - il + 1$ 。
<i>w</i>	REAL (cheevr の場合 ) DOUBLE PRECISION (zheevr の場合 )。 配列、次元は $\max(1, n)$ 以上。 選択された固有値が昇順に $w(1) \sim w(m)$ に格納される。
<i>z</i>	COMPLEX (cheevr の場合 ) DOUBLE COMPLEX (zheevr の場合 )。 配列 $z(ldz, *)$ 。 <i>z</i> の第 2 次元は、 $\max(1, m)$ 以上でなければならない。 <i>jobz</i> = 'V' の場合、 <i>info</i> = 0 であれば、 <i>z</i> の先頭の <i>m</i> 列に、行列 <i>T</i> の選択された固有値に対応する正規直交固有ベクトルが格納される。すなわち、 $w(i)$ に対応する固有ベクトルが、 <i>z</i> の <i>i</i> 番目の列に格納される。 <i>jobz</i> = 'N' の場合、 <i>z</i> は参照されない。 注：配列 <i>z</i> には、 $\max(1, m)$ 以上の列が提供されなければならない。 <i>range</i> = 'V' の場合、 <i>m</i> の正確な値が事前にわからないため、上限値を使用する必要がある。
<i>isuppz</i>	INTEGER。 配列、次元は $2 * \max(1, m)$ 以上。  <i>z</i> に格納されている固有ベクトルのサポート情報、すなわち <i>z</i> に格納されている非ゼロの成分を示すインデックス。 <i>i</i> 番目の固有ベクトルは、 $isuppz(2i-1)$ から $isuppz(2i)$ までの成分のみ非ゼロである。
<i>work(1)</i>	終了時に、 <i>info</i> = 0 の場合、 <i>work(1)</i> は、 <i>lwork</i> の必要最小サイズを返す。
<i>rwork(1)</i>	終了時に、 <i>info</i> = 0 の場合、 <i>rwork(1)</i> は、 <i>lrwork</i> の必要最小サイズを返す。
<i>iwork(1)</i>	終了時に、 <i>info</i> = 0 の場合、 <i>iwork(1)</i> は、 <i>liwork</i> の必要最小サイズを返す。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正であったことを示す。 <i>info</i> = <i>i</i> の場合、内部エラーが発生したことを示す。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `heevr` のインターフェイスの詳細を以下に示す。

<code>a</code>	サイズ $(n,n)$ の行列 $A$ を格納する。
<code>w</code>	長さ $(n)$ のベクトルを格納する。
<code>z</code>	サイズ $(n,m)$ の行列 $Z$ を格納する。ここで、値 $n$ と $m$ は有意である。
<code>isuppz</code>	長さ $(2*m)$ のベクトルを格納する。ここで、値 $(2*m)$ は有意である。
<code>uplo</code>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<code>v1</code>	この成分のデフォルト値は、 <code>v1 = -HUGE(v1)</code> である。
<code>vu</code>	この成分のデフォルト値は、 <code>vu = HUGE(v1)</code> である。
<code>il</code>	この引数のデフォルト値は、 <code>il = 1</code> である。
<code>iu</code>	この引数のデフォルト値は、 <code>iu = n</code> である。
<code>abstol</code>	この成分のデフォルト値は、 <code>abstol = 0.0_WP</code> である。
<code>jobz</code>	引数 $z$ の存在に基づいて以下のように復元される。 <code>jobz = 'V'</code> ( $z$ が存在する場合)、 <code>jobz = 'N'</code> ( $z$ が省略された場合)。 <code>isuppz</code> が存在し、 $z$ が省略された場合、エラー条件がセットされる。
<code>range</code>	引数 $v1$ 、 $vu$ 、 $il$ 、および $iu$ の存在に基づいて以下のように復元される。 <code>range = 'V'</code> ( $v1$ と $vu$ のいずれかまたは両方が存在する場合)、 <code>range = 'I'</code> ( $il$ と $iu$ のいずれかまたは両方が存在する場合)、 <code>range = 'A'</code> ( $v1$ 、 $vu$ 、 $il$ 、および $iu$ がすべて存在しない場合)。 $v1$ と $vu$ のいずれかまたは両方が存在し、同時に $il$ と $iu$ のいずれかまたは両方が存在する場合、エラー条件がセットされる。

## アプリケーション・ノート

最適なパフォーマンスを得るには、次の値を使用する。

$$lwork \geq (nb+1)*n$$

ここで、 $nb$  は、[ilaenv](#) が `?hetrd` と `?unmtr` に対して返したブロックサイズである。必要なワークスペースの大きさがわからない場合は、最初の実行では `lwork` を大きめの値に設定する。終了時に、`work(1)` の値を調べ、以降はその値を使用する。

通常の ?stegr の実行で NaN と無限大が発生するため、NaN と無限大が IEEE 標準のデフォルト方法以外で処理される環境では、浮動小数点例外によって異常終了する可能性がある。

---

### ?spev

圧縮形式の実対称行列の固有値と ( オプションで )  
固有ベクトルをすべて計算する。

---

#### 構文

##### Fortran 77:

```
call sspev(jobz, uplo, n, ap, w, z, ldz, work, info)
call dspev(jobz, uplo, n, ap, w, z, ldz, work, info)
```

##### Fortran 95:

```
call spev(a, w [,uplo] [,z] [,info])
```

#### 説明

このルーチンは、圧縮形式の実対称行列  $A$  の固有値と ( オプションで ) 固有ベクトルをすべて計算する。

#### 入力パラメータ

jobz	CHARACTER*1。'N' または 'V' でなければならない。 job='N' の場合、固有値のみを計算する。 job='V' の場合、固有値と固有ベクトルを計算する。
uplo	CHARACTER*1。'U' または 'L' でなければならない。 uplo='U' の場合、ap に、圧縮形式の $A$ の上三角部分を格納する。 uplo='L' の場合、ap に、圧縮形式の $A$ の下三角部分を格納する。
n	INTEGER。行列 $A$ の次数 ( $n \geq 0$ )。
ap,work	REAL (sspev の場合 ) DOUBLE PRECISION (dspev の場合 ) 配列 : ap(*) には、uplo の値に従って、対称行列 $A$ の上または下三角部分を圧縮形式で格納する。ap の次元は、 $\max(1, n*(n+1)/2)$ 以上でなければならない。

*work*(\*) は、ワークスペース配列、次元は  $\max(1, 3n)$  以上。

*ldz*            INTEGER。出力配列 *z* のリーディング・ディメンジョン。  
 次の制約がある。  
*jobz* = 'N' の場合、 $ldz \geq 1$ 。  
*jobz* = 'V' の場合、 $ldz \geq \max(1, n)$ 。

## 出力パラメータ

*w*, *z*            REAL (sspev の場合)  
 DOUBLE PRECISION (dspev の場合)  
 配列：  
*w*(\*)、次元は  $\max(1, n)$  以上。  
*info* = 0 の場合、*w* に、行列 *A* の固有値が昇順に格納される。  
*z*(*ldz*,\*)。 *z* の第 2 次元は、 $\max(1, n)$  以上でなければならない。  
*jobz* = 'V' の場合、*info* = 0 であれば、*z* に、行列 *A* の正規直交固有ベクトルが格納される。そのとき、*w*(*i*) に対応する固有ベクトルが、*z* の *i* 番目の列に格納される。  
*jobz* = 'N' の場合、*z* は参照されない。

*ap*              終了時に、三重対角形式に縮退させたときに生成された値で上書きされる。すなわち、生成された三重対角行列の対角成分と非対角成分によって、*A* の対応する成分が上書きされる。

*info*            INTEGER。  
*info* = 0 の場合、実行は正常に終了したことを示す。  
*info* = -*i* の場合、*i* 番目のパラメータの値が不正であったことを示す。  
*info* = *i* の場合、このアルゴリズムが収束に失敗したことを示す。  
*i* は、中間の三重対角形式で、ゼロに収束しなかった成分の個数を表す。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *spev* のインターフェイスの詳細を以下に示す。

*a*                Fortran 77 インターフェイスでの引数 *ap* を意味する。サイズ  $(n*(n+1)/2)$  の配列 *A* を格納する。

<i>w</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>z</i>	サイズ ( <i>n,n</i> ) の行列 <i>Z</i> を格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>jobz</i>	引数 <i>z</i> の存在に基づいて以下のように復元される。 <i>jobz</i> = 'V' ( <i>z</i> が存在する場合)、 <i>jobz</i> = 'N' ( <i>z</i> が省略された場合)。

---

### ?hpev

圧縮形式のエルミート行列の固有値と ( オプションで ) 固有ベクトルをすべて計算する。

---

#### 構文

##### Fortran 77:

```
call chpev(jobz, uplo, n, ap, w, z, ldz, work, rwork, info)
call zhpev(jobz, uplo, n, ap, w, z, ldz, work, rwork, info)
```

##### Fortran 95:

```
call hpev(a, w [,uplo] [,z] [,info])
```

#### 説明

このルーチンは、圧縮形式のエルミート行列 *A* の固有値と ( オプションで ) 固有ベクトルをすべて計算する。

#### 入力パラメータ

<i>jobz</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>job</i> = 'N' の場合、固有値のみを計算する。 <i>job</i> = 'V' の場合、固有値と固有ベクトルを計算する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> = 'U' の場合、 <i>ap</i> に、圧縮形式の <i>A</i> の上三角部分を格納する。 <i>uplo</i> = 'L' の場合、 <i>ap</i> に、圧縮形式の <i>A</i> の下三角部分を格納する。
<i>n</i>	INTEGER。行列 <i>A</i> の次数 ( <i>n</i> ≥ 0)。

<i>ap, work</i>	COMPLEX (chpev の場合 ) DOUBLE COMPLEX (zhpev の場合 )。 配列： <i>ap</i> (*) には、 <i>uplo</i> の値に従って、エルミート行列 <i>A</i> の上または下三角部分を圧縮形式で格納する。 <i>ap</i> の次元は、 $\max(1, n*(n+1)/2)$ 以上でなければならない。  <i>work</i> (*) は、ワークスペース配列、次元は $\max(1, 2n-1)$ 以上。
<i>ldz</i>	INTEGER。出力配列 <i>z</i> のリーディング・ディメンジョン。 次の制約がある。 <i>jobz</i> ='N' の場合、 $ldz \geq 1$ 。 <i>jobz</i> ='V' の場合、 $ldz \geq \max(1, n)$ 。
<i>rwork</i>	REAL (chpev の場合 ) DOUBLE PRECISION (zhpev の場合 )。 ワークスペース配列、次元は $\max(1, 3n-2)$ 以上。

### 出力パラメータ

<i>w</i>	REAL (chpev の場合 ) DOUBLE PRECISION (zhpev の場合 )。 配列、次元は $\max(1, n)$ 以上。 <i>info</i> =0 の場合、 <i>w</i> に、行列 <i>A</i> の固有値が昇順に格納される。
<i>z</i>	COMPLEX (chpev の場合 ) DOUBLE COMPLEX (zhpev の場合 )。 配列 <i>z</i> ( <i>ldz</i> ,*)。 <i>z</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>jobz</i> ='V' の場合、 <i>info</i> =0 であれば、 <i>z</i> に、行列 <i>A</i> の正規直交固有ベクトルが格納される。そのとき、 <i>w</i> ( <i>i</i> ) に対応する固有ベクトルが、 <i>z</i> の <i>i</i> 番目の列に格納される。 <i>jobz</i> ='N' の場合、 <i>z</i> は参照されない。
<i>ap</i>	終了時に、三重対角形式に縮退させたときに生成された値で上書きされる。すなわち、生成された三重対角行列の対角成分と非対角成分によって、 <i>A</i> の対応する成分が上書きされる。
<i>info</i>	INTEGER。 <i>info</i> =0 の場合、実行は正常に終了したことを示す。 <i>info</i> =- <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正であったことを示す。 <i>info</i> = <i>i</i> の場合、このアルゴリズムが収束に失敗したことを示す。 <i>i</i> は、中間の三重対角形式で、ゼロに収束しなかった成分の個数を表す。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `hpev` のインターフェイスの詳細を以下に示す。

<code>a</code>	Fortran 77 インターフェイスでの引数 <code>ap</code> を意味する。 サイズ $(n*(n+1)/2)$ の配列 <code>A</code> を格納する。
<code>w</code>	長さ $(n)$ のベクトルを格納する。
<code>z</code>	サイズ $(n,n)$ の行列 <code>Z</code> を格納する。
<code>uplo</code>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<code>jobz</code>	引数 <code>z</code> の存在に基づいて以下のように復元される。 <code>jobz = 'V'</code> ( <code>z</code> が存在する場合)、 <code>jobz = 'N'</code> ( <code>z</code> が省略された場合)。

---

## ?spevd

分割統治アルゴリズムを使用して、圧縮形式の実対称行列の固有値と (オプションで) 固有ベクトルをすべて計算する。

---

### 構文

#### Fortran 77:

```
call sspevd(job, uplo, n, ap, w, z, ldz, work, lwork, iwork, liwork, info)
call dspevd(job, uplo, n, ap, w, z, ldz, work, lwork, iwork, liwork, info)
```

#### Fortran 95:

```
call spevd(a, w [,uplo] [,z] [,info])
```

### 説明

このルーチンは、圧縮形式の実対称行列  $A$  の固有値と (オプションで) 固有ベクトルをすべて計算する。言い換えると、 $A$  のスペクトル因子分解  $A = Z\Lambda Z^T$  を求めることができる。 $\Lambda$  は対角成分が固有値  $\lambda_i$  であるような対角行列、 $Z$  は各列が固有ベクトル  $z_i$  であるような直交行列である。つまり、次のように表現できる。



$$Az_i = \lambda_i z_i \text{ ただし } i = 1, 2, \dots, n$$

このルーチンは、固有ベクトルも要求された場合は、分割統治アルゴリズムを使用して、固有値と固有ベクトルを計算する。ただし、固有値のみ要求された場合は、Pal-Walker-Kahan 変形版の *QL* または *QR* アルゴリズムを使用して、固有値を計算する。

### 入力パラメータ

<i>job</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>job</i> ='N' の場合、固有値のみを計算する。 <i>job</i> ='V' の場合、固有値と固有ベクトルを計算する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> ='U' の場合、 <i>ap</i> に、圧縮形式の <i>A</i> の上三角部分を格納する。 <i>uplo</i> ='L' の場合、 <i>ap</i> に、圧縮形式の <i>A</i> の下三角部分を格納する。
<i>n</i>	INTEGER。行列 <i>A</i> の次数 ( $n \geq 0$ )。
<i>ap,work</i>	REAL (sspevd の場合) DOUBLE PRECISION (dspevd の場合) 配列： <i>ap</i> (*) には、 <i>uplo</i> の値に従って、対称行列 <i>A</i> の上または下三角部分を圧縮形式で格納する。 <i>ap</i> のサイズは、 $\max(1, n*(n+1)/2)$ 以上でなければならない。 <i>work</i> (*) は、ワークスペース配列、次元は <i>lwork</i> 以上。
<i>ldz</i>	INTEGER。出力配列 <i>z</i> のリーディング・ディメンジョン。 次の制約がある。 <i>job</i> ='N' の場合、 $ldz \geq 1$ 。 <i>job</i> ='V' の場合、 $ldz \geq \max(1, n)$ 。
<i>lwork</i>	INTEGER。配列 <i>work</i> の次元。 次の制約がある。 $n \leq 1$ の場合、 $lwork \geq 1$ 。 <i>job</i> ='N' で $n > 1$ の場合、 $lwork \geq 2n$ 。 <i>job</i> ='V' で $n > 1$ の場合、 $lwork \geq 2n^2 + (5+2k)*n + 1$ (ここで、 <i>k</i> は、 $2^k \geq n$ を満たす最小の整数)。 <i>lwork</i> = -1 の場合、ワークスペースのクエリとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリとして返す。 <i>xerbla</i> は <i>lwork</i> に関するエラー・メッセージを生成しない。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は <i>liwork</i> 以上。

*liwork* INTEGER。配列 *iwork* の次元。  
 次の制約がある。  
 $n \leq 1$  の場合、 $liwork \geq 1$ 。  
 $job = 'N'$  で  $n > 1$  の場合、 $liwork \geq 1$ 。  
 $job = 'V'$  で  $n > 1$  の場合、 $liwork \geq 5n+3$ 。  
 $liwork = -1$  の場合、ワークスペースのクエリとみなされ、ルーチンは *iwork* 配列の最適サイズだけを計算し、その値を *iwork* 配列の最初のエントリとして返す。xerbla は *liwork* に関するエラー・メッセージを生成しない。

## 出力パラメータ

*w, z* REAL (sspevd の場合)  
 DOUBLE PRECISION (dspevd の場合)  
 配列:  
 $w(*)$ 、次元は  $\max(1, n)$  以上。  
 $info = 0$  の場合、行列 *A* の固有値が昇順に格納される。*info* も参照のこと。  
 $z(ldz, *)$ 。 $z$  の第 2 次元は、1 以上 ( $job = 'N'$  の場合) または  $\max(1, n)$  以上 ( $job = 'V'$  の場合) でなければならない。  
 $job = 'V'$  の場合、この配列は、*A* の固有ベクトルの入った直交行列 *Z* で上書きされる。 $job = 'N'$  の場合、 $z$  は参照されない。

*ap* 終了時に、三重対角形式に縮退させたときに生成された値で上書きされる。すなわち、生成された三重対角行列の対角成分と非対角成分によって、*A* の対応する成分が上書きされる。

*work(1)* 終了時に、 $info = 0$  の場合、*work(1)* に最適な *lwork* 値が返される。

*iwork(1)* 終了時に、 $info = 0$  の場合、*iwork(1)* に最適な *liwork* 値が返される。

*info* INTEGER。  
 $info = 0$  の場合、実行は正常に終了したことを示す。  
 $info = i$  の場合、このアルゴリズムが収束に失敗したことを示す。  
*i* は、中間の三重対角形式で、ゼロに収束しなかった成分の個数を表す。  
 $info = -i$  の場合、*i* 番目のパラメータの値が不正であったことを示す。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `spevd` のインターフェイスの詳細を以下に示す。

<code>a</code>	Fortran 77 インターフェイスでの引数 <code>ap</code> を意味する。 サイズ $(n*(n+1)/2)$ の配列 <code>A</code> を格納する。
<code>w</code>	長さ $(n)$ のベクトルを格納する。
<code>z</code>	サイズ $(n,n)$ の行列 <code>Z</code> を格納する。
<code>uplo</code>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<code>jobz</code>	引数 <code>z</code> の存在に基づいて以下のように復元される。 <code>jobz = 'V'</code> ( <code>z</code> が存在する場合)、 <code>jobz = 'N'</code> ( <code>z</code> が省略された場合)。

## アプリケーション・ノート

計算で求めた固有値と固有ベクトルは、正確には、 $\|E\|_2 = O(\varepsilon) \|T\|_2$  のような行列  $T + E$  ( $\varepsilon$  はマシン精度) のものである。

このルーチンの複素数版は、[?hpevd](#) である。

フル形式の行列に対する [?syevd](#) と、帯形式の行列に関する [?sbevd](#) も参照のこと。

---

## ?hpevd

分割統治アルゴリズムを使用して、圧縮形式の複素エルミート行列の固有値と(オプションで)固有ベクトルをすべて計算する。

---

### 構文

#### Fortran 77:

```
call chpevd(job, uplo, n, ap, w, z, ldz, work, lwork, rwork, lrwork, iwork,
            liwork, info)
call zhpevd(job, uplo, n, ap, w, z, ldz, work, lwork, rwork, lrwork, iwork,
            liwork, info)
```

## Fortran 95:

```
call hpevd(a, w [,uplo] [,z] [,info])
```

### 説明

このルーチンは、圧縮形式の複素エルミート行列  $A$  の固有値と (オプションで) 固有ベクトルをすべて計算する。言い換えると、 $A$  のスペクトル因子分解  $A = Z\Lambda Z^H$  を求めることができる。 $\Lambda$  は対角成分が固有値  $\lambda_i$  であるような実対角行列、 $Z$  は各列が固有ベクトル  $z_i$  であるような (複素) ユニタリ行列である。つまり、次のように表現できる。

$$Az_i = \lambda_i z_i \quad \text{ただし } i = 1, 2, \dots, n$$

このルーチンは、固有ベクトルも要求された場合は、分割統治アルゴリズムを使用して、固有値と固有ベクトルを計算する。ただし、固有値のみ要求された場合は、Pal-Walker-Kahan 変形版の  $QL$  または  $QR$  アルゴリズムを使用して、固有値を計算する。

### 入力パラメータ

<i>job</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>job</i> ='N' の場合、固有値のみを計算する。 <i>job</i> ='V' の場合、固有値と固有ベクトルを計算する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> ='U' の場合、 <i>ap</i> に、圧縮形式の $A$ の上三角部分を格納する。 <i>uplo</i> ='L' の場合、 <i>ap</i> に、圧縮形式の $A$ の下三角部分を格納する。
<i>n</i>	INTEGER。行列 $A$ の次数 ( $n \geq 0$ )。
<i>ap,work</i>	COMPLEX (chpevd の場合) DOUBLE COMPLEX (zhpevd の場合) 配列: <i>ap</i> (*) には、 <i>uplo</i> の値に従って、エルミート行列 $A$ の上または下三角部分を圧縮形式で格納する。 <i>ap</i> のサイズは、 $\max(1, n*(n+1)/2)$ 以上でなければならない。 <i>work</i> (*) は、ワークスペース配列、次元は <i>lwork</i> 以上。
<i>ldz</i>	INTEGER。出力配列 $z$ のリーディング・ディメンション。 次の制約がある。 <i>job</i> ='N' の場合、 $ldz \geq 1$ 。 <i>job</i> ='V' の場合、 $ldz \geq \max(1, n)$ 。
<i>lwork</i>	INTEGER。配列 <i>work</i> の次元。 次の制約がある。 $n \leq 1$ の場合、 $lwork \geq 1$ 。 <i>job</i> ='N' で $n > 1$ の場合、 $lwork \geq n$ 。

$job='V'$  で  $n > 1$  の場合、 $lwork \geq 2n$ 。

$lwork = -1$  の場合、ワークスペースのクエリとみなされ、ルーチンは  $work$  配列の最適サイズだけを計算し、その値を  $work$  配列の最初のエントリとして返す。xerbla は  $lwork$  に関するエラー・メッセージを生成しない。

*rwork* REAL (chpevd の場合 )  
DOUBLE PRECISION (zhpevd の場合 )  
ワークスペース配列、次元は  $lrwork$  以上。

*lrwork* INTEGER。配列 *rwork* の次元。  
次の制約がある。  
 $n \leq 1$  の場合、 $lrwork \geq 1$ 。  
 $job='N'$  で  $n > 1$  の場合、 $lrwork \geq n$ 。  
 $job='V'$  で  $n > 1$  の場合、 $lrwork \geq 3n^2 + (4+2k) * n + 1$   
(ここで、 $k$  は、 $2^k \geq n$  を満たす最小の整数)。  
 $lrwork = -1$  の場合、ワークスペースのクエリとみなされ、ルーチンは  $rwork$  配列の最適サイズだけを計算し、その値を  $rwork$  配列の最初のエントリとして返す。xerbla は  $lrwork$  に関するエラー・メッセージを生成しない。

*iwork* INTEGER。  
ワークスペース配列、次元は  $liwork$  以上。

*liwork* INTEGER。配列 *iwork* の次元。  
次の制約がある。  
 $n \leq 1$  の場合、 $liwork \geq 1$ 。  
 $job='N'$  で  $n > 1$  の場合、 $liwork \geq 1$ 。  
 $job='V'$  で  $n > 1$  の場合、 $liwork \geq 5n + 2$ 。  
 $liwork = -1$  の場合、ワークスペースのクエリとみなされ、ルーチンは *iwork* 配列の最適サイズだけを計算し、その値を *iwork* 配列の最初のエントリとして返す。xerbla は *liwork* に関するエラー・メッセージを生成しない。

## 出力パラメータ

*w* REAL (chpevd の場合 )  
DOUBLE PRECISION (zhpevd の場合 )  
配列、次元は  $\max(1, n)$  以上。  
 $info = 0$  の場合、行列 *A* の固有値が昇順に格納される。*info* も参照のこと。

<i>z</i>	COMPLEX (chpevd の場合) DOUBLE COMPLEX (zhpevd の場合) 配列、次元は ( <i>ldz</i> ,*)。 <i>z</i> の第 2 次元は、1 以上 ( <i>job</i> ='N' の場合) または $\max(1, n)$ 以上 ( <i>job</i> ='V' の場合) でなければならない。 <i>job</i> ='V' の場合、この配列は、A の固有ベクトルの入ったユニタリ行列 <i>Z</i> で上書きされる。 <i>job</i> ='N' の場合、 <i>z</i> は参照されない。
<i>ap</i>	終了時に、三重対角形式に縮退させたときに生成された値で上書きされる。すなわち、生成された三重対角行列の対角成分と非対角成分によって、A の対応する成分が上書きされる。
<i>work(1)</i>	終了時に、 <i>lwork</i> > 0 の場合、 <i>work(1)</i> の実数部は <i>lwork</i> の必要最小サイズを返す。
<i>rwork(1)</i>	終了時に、 <i>lrwork</i> > 0 の場合、 <i>rwork(1)</i> は <i>lrwork</i> の必要最小サイズを返す。
<i>iwork(1)</i>	終了時に、 <i>liwork</i> > 0 の場合、 <i>iwork(1)</i> は <i>liwork</i> の必要最小サイズを返す。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = <i>i</i> の場合、このアルゴリズムが収束に失敗したことを示す。 <i>i</i> は、中間の三重対角形式で、ゼロに収束しなかった成分の個数を表す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正であったことを示す。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン hpevd のインターフェイスの詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ap</i> を意味する。 サイズ ( $n*(n+1)/2$ ) の配列 <i>A</i> を格納する。
<i>w</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>z</i>	サイズ ( <i>n</i> , <i>n</i> ) の行列 <i>Z</i> を格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。

`jobz`            引数  $z$  の存在に基づいて以下のように復元される。  
`jobz = 'V'` ( $z$  が存在する場合)、  
`jobz = 'N'` ( $z$  が省略された場合)。

### アプリケーション・ノート

計算で求めた固有値と固有ベクトルは、正確には、 $\|E\|_2 = O(\varepsilon) \|T\|_2$  のような行列  $T + E$  ( $\varepsilon$  はマシン精度) のものである。

このルーチンの実数版は、[?spevd](#) である。

フル形式の行列に対する [?heevd](#) と、帯形式の行列に関する [?hbevd](#) も参照のこと。

---

## ?spevx

圧縮形式の実対称行列の固有値と (オプションで)  
固有ベクトルを選択的に計算する。

---

### 構文

#### Fortran 77:

```
call sspevx(jobz, range, uplo, n, ap, vl, vu, il, iu, abstol, m, w, z, ldz,  
            work, iwork, ifail, info)  
call dspevx(jobz, range, uplo, n, ap, vl, vu, il, iu, abstol, m, w, z, ldz,  
            work, iwork, ifail, info)
```

#### Fortran 95:

```
call spevx(a, w [,uplo] [,z] [,vl] [,vu] [,il] [,iu] [,m] [,ifail] [,abstol]  
            [,info])
```

### 説明

圧縮形式の実対称行列  $A$  の固有値と (オプションで) 固有ベクトルを選択的に計算する。  
固有値と固有ベクトルを選択するには、希望する固有値の値の範囲かインデックスの範囲を指定する。

## 入力パラメータ

<i>jobz</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>job</i> ='N' の場合、固有値のみを計算する。 <i>job</i> ='V' の場合、固有値と固有ベクトルを計算する。
<i>range</i>	CHARACTER*1。'A'、'V'、または 'I' のいずれかでなければならない。 <i>range</i> ='A' の場合、固有値をすべて計算する。 <i>range</i> ='V' の場合、半開区間 $v_l < \lambda_i \leq v_u$ における固有値 $\lambda_i$ を計算する。 <i>range</i> ='I' の場合、ルーチンはインデックスが $i_l \sim i_u$ である固有値を計算する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> ='U' の場合、 <i>ap</i> に、圧縮形式の <i>A</i> の上三角部分を格納する。 <i>uplo</i> ='L' の場合、 <i>ap</i> に、圧縮形式の <i>A</i> の下三角部分を格納する。
<i>n</i>	INTEGER。行列 <i>A</i> の次数 ( $n \geq 0$ )。
<i>ap, work</i>	REAL (sspevx の場合 ) DOUBLE PRECISION (dspevx の場合 ) 配列 : <i>ap</i> (*) には、 <i>uplo</i> の値に従って、対称行列 <i>A</i> の上または下三角部分を圧縮形式で格納する。 <i>ap</i> の次元は、 $\max(1, n*(n+1)/2)$ 以上でなければならない。  <i>work</i> (*) は、ワークスペース配列、次元は $\max(1, 8n)$ 以上。
<i>vl, vu</i>	REAL (sspevx の場合 ) DOUBLE PRECISION (dspevx の場合 ) <i>range</i> ='V' の場合、固有値を探す区間の下限値と上限値。 次の制約がある。 $v_l < v_u$ 。 <i>range</i> ='A' または 'I' の場合、 <i>vl</i> と <i>vu</i> は参照されない。
<i>il, iu</i>	INTEGER。 <i>range</i> ='I' の場合、返される最小固有値と最大固有値に対する昇順のインデックス。 次の制約がある。 $1 \leq i_l \leq i_u \leq n$ ( $n > 0$ の場合 )。 $i_l=1$ かつ $i_u=0$ ( $n=0$ の場合 )。 <i>range</i> ='A' または 'V' の場合、 <i>il</i> と <i>iu</i> は参照されない。



<i>abstol</i>	REAL (sspevx の場合 ) DOUBLE PRECISION (dspevx の場合 ) 各固有値に許される絶対誤差許容値。誤差許容値の詳細は、「アプリケーション・ノート」を参照。
<i>ldz</i>	INTEGER。出力配列 <i>z</i> のリーディング・ディメンジョン。 次の制約がある。 <i>jobz</i> = 'N' の場合、 $ldz \geq 1$ 。 <i>jobz</i> = 'V' の場合、 $ldz \geq \max(1, n)$ 。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, 5n)$ 以上。

### 出力パラメータ

<i>ap</i>	終了時に、三重対角形式に縮退させたときに生成された値で上書きされる。すなわち、生成された三重対角行列の対角成分と非対角成分によって、 <i>A</i> の対応する成分が上書きされる。
<i>m</i>	INTEGER。検出された固有値の総数。 $(0 \leq m \leq n)$ 。 <i>range</i> = 'A' の場合、 $m = n$ 、 <i>range</i> = 'I' の場合、 $m = iu - il + 1$ 。
<i>w, z</i>	REAL (sspevx の場合 ) DOUBLE PRECISION (dspevx の場合 ) 配列： <i>w</i> (*)、次元は $\max(1, n)$ 以上。 <i>info</i> = 0 の場合、選択された <i>A</i> の固有値が昇順に格納される。 <i>z</i> ( <i>ldz</i> , *)。 <i>z</i> の第 2 次元は、 $\max(1, m)$ 以上でなければならない。 <i>jobz</i> = 'V' の場合、 <i>info</i> = 0 であれば、 <i>z</i> の先頭の <i>m</i> 列に、行列 <i>A</i> の選択された固有値に対応する正規直交固有ベクトルが格納される。すなわち、 <i>w</i> ( <i>i</i> ) に対応する固有ベクトルが、 <i>z</i> の <i>i</i> 番目の列に格納される。固有ベクトルが収束できない場合、 <i>z</i> のその列には、その固有ベクトルに対する最新の近似値が格納され、固有ベクトルのインデックスは <i>ifail</i> に返される。 <i>jobz</i> = 'N' の場合、 <i>z</i> は参照されない。 注：配列 <i>z</i> には、 $\max(1, m)$ 以上の列が提供されなければならない。 <i>range</i> = 'V' の場合、 <i>m</i> の正確な値が事前にわからないため、上限値を使用する必要がある。
<i>ifail</i>	INTEGER。配列、次元は $\max(1, n)$ 以上。 <i>jobz</i> = 'V' の場合、 <i>info</i> = 0 であれば、 <i>ifail</i> の先頭から <i>m</i> 個の成分はゼロになる。 <i>info</i> > 0 であれば、収束に失敗した固有ベクトルのインデックスが <i>ifail</i> に格納される。 <i>jobz</i> = 'N' の場合、 <i>ifail</i> は参照されない。

*info*                    INTEGER。  
*info* = 0 の場合、実行は正常に終了したことを示す。  
*info* = -*i* の場合、*i* 番目のパラメータの値が不正であったことを示す。  
*info* = *i* の場合、*i* 番目の固有ベクトルが収束に失敗した。その固有ベクトルのインデックスが配列 *ifail* に格納される。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *spevx* のインターフェイスの詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ap</i> を意味する。 サイズ $(n*(n+1)/2)$ の配列 <i>A</i> を格納する。
<i>w</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>z</i>	サイズ ( <i>n,m</i> ) の行列 <i>Z</i> を格納する。ここで、値 <i>n</i> と <i>m</i> は有意である。
<i>ifail</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>v1</i>	この成分のデフォルト値は、 <i>v1</i> = -HUGE( <i>v1</i> ) である。
<i>vu</i>	この成分のデフォルト値は、 <i>vu</i> = HUGE( <i>v1</i> ) である。
<i>il</i>	この引数のデフォルト値は、 <i>il</i> = 1 である。
<i>iu</i>	この引数のデフォルト値は、 <i>iu</i> = <i>n</i> である。
<i>abstol</i>	この成分のデフォルト値は、 <i>abstol</i> = 0.0_WP である。
<i>jobz</i>	引数 <i>z</i> の存在に基づいて以下のように復元される。 <i>jobz</i> = 'V' ( <i>z</i> が存在する場合)、 <i>jobz</i> = 'N' ( <i>z</i> が省略された場合)。 <i>ifail</i> が存在し、 <i>z</i> が省略された場合、エラー条件がセットされる。
<i>range</i>	引数 <i>v1</i> 、 <i>vu</i> 、 <i>il</i> 、および <i>iu</i> の存在に基づいて以下のように復元される。 <i>range</i> = 'V' ( <i>v1</i> と <i>vu</i> のいずれかまたは両方が存在する場合)、 <i>range</i> = 'I' ( <i>il</i> と <i>iu</i> のいずれかまたは両方が存在する場合)、

`range = 'A'` (`vl`, `vu`, `il`, および `iu` がすべて存在しない場合。  
`vl` と `vu` のいずれかまたは両方が存在し、同時に `il` と `iu` のいずれか  
または両方が存在する場合、エラー条件がセットされる。

### アプリケーション・ノート

$abstol + \varepsilon * \max(|a|, |b|)$  以下の幅の区間  $[a, b]$  内に存在していると判別された場合 ( $\varepsilon$  はマシン精度)、近似固有値は、収束したものとして受け入れられる。 $abstol$  がゼロ以下の場合、 $\varepsilon * \|T\|_1$  は所定の場所で使用される ( $T$  は、 $A$  を三重対角形式に縮退して獲得した三重対角行列)。

固有値が最も正確に計算されるのは、 $abstol$  がゼロでなく、アンダーフローのしきい値の 2 倍、 $2 * \lambda_{\text{mach}}('S')$  に設定されたときである。このルーチンで  $info > 0$  が返された場合、固有ベクトルのいくつか収束に失敗したことを意味するので、 $abstol$  を  $2 * \lambda_{\text{mach}}('S')$  に設定して、やり直してみる。

---

## ?hpevx

圧縮形式のエルミート行列の固有値と (オプションで) 固有ベクトルを選択的に計算する。

---

### 構文

#### Fortran 77:

```
call chpevx(jobz, range, uplo, n, ap, vl, vu, il, iu, abstol, m, w, z, ldz,
            work, rwork, iwork, ifail, info)
call zhpevx(jobz, range, uplo, n, ap, vl, vu, il, iu, abstol, m, w, z, ldz,
            work, rwork, iwork, ifail, info)
```

#### Fortran 95:

```
call hpevx(a, w [,uplo] [,z] [,vl] [,vu] [,il] [,iu] [,m] [,ifail] [,abstol]
            [,info])
```

### 説明

このルーチンは、圧縮形式の複素エルミート行列  $A$  の固有値と (オプションで) 固有ベクトルを選択的に計算する。固有値と固有ベクトルを選択するには、希望する固有値の値の範囲かインデックスの範囲を指定する。

## 入力パラメータ

<i>jobz</i>	CHARACTER*1. 'N' または 'V' でなければならない。 <i>job</i> ='N' の場合、固有値のみを計算する。 <i>job</i> ='V' の場合、固有値と固有ベクトルを計算する。
<i>range</i>	CHARACTER*1. 'A'、'V'、または 'I' のいずれかでなければならない。 <i>range</i> ='A' の場合、固有値をすべて計算する。 <i>range</i> ='V' の場合、半開区間 $v_l < \lambda_i \leq v_u$ における固有値 $\lambda_i$ を計算する。 <i>range</i> ='I' の場合、ルーチンはインデックスが <i>i</i> 1 ~ <i>i</i> u である固有値を計算する。
<i>uplo</i>	CHARACTER*1. 'U' または 'L' でなければならない。 <i>uplo</i> ='U' の場合、 <i>ap</i> に、圧縮形式の <i>A</i> の上三角部分を格納する。 <i>uplo</i> ='L' の場合、 <i>ap</i> に、圧縮形式の <i>A</i> の下三角部分を格納する。
<i>n</i>	INTEGER。行列 <i>A</i> の次数 ( $n \geq 0$ )。
<i>ap, work</i>	COMPLEX (chpevx の場合) DOUBLE COMPLEX (zhpevx の場合) 配列: <i>ap</i> (*) には、 <i>uplo</i> の値に従って、エルミート行列 <i>A</i> の上または下三角部分を圧縮形式で格納する。 <i>ap</i> の次元は、 $\max(1, n*(n+1)/2)$ 以上でなければならない。  <i>work</i> (*) は、ワークスペース配列、次元は $\max(1, 2n)$ 以上。
<i>v1, vu</i>	REAL (chpevx の場合) DOUBLE PRECISION (zhpevx の場合) <i>range</i> ='V' の場合、固有値を探す区間の下限値と上限値。 次の制約がある。 $v_l < v_u$ 。 <i>range</i> ='A' または 'I' の場合、 <i>v1</i> と <i>vu</i> は参照されない。
<i>i1, iu</i>	INTEGER。 <i>range</i> ='I' の場合、返される最小固有値と最大固有値に対する昇順のインデックス。 次の制約がある。 $1 \leq i_1 \leq i_u \leq n$ ( $n > 0$ の場合)。 <i>i</i> 1=1 かつ <i>i</i> u=0 ( $n=0$ の場合)。 <i>range</i> ='A' または 'V' の場合、 <i>i1</i> と <i>i</i> u は参照されない。

<i>abstol</i>	REAL (chpevx の場合 ) DOUBLE PRECISION (zhpevx の場合 ) 各固有値に許される絶対誤差許容値。誤差許容値の詳細は、「アプリケーション・ノート」を参照。
<i>ldz</i>	INTEGER。出力配列 <i>z</i> のリーディング・ディメンジョン。 次の制約がある。 <i>jobz</i> = 'N' の場合、 $ldz \geq 1$ 。 <i>jobz</i> = 'V' の場合、 $ldz \geq \max(1, n)$ 。
<i>rwork</i>	REAL (chpevx の場合 ) DOUBLE PRECISION (zhpevx の場合 ) ワークスペース配列、次元は $\max(1, 7n)$ 以上。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, 5n)$ 以上。

### 出力パラメータ

<i>ap</i>	終了時に、三重対角形式に縮退させたときに生成された値で上書きされる。すなわち、生成された三重対角行列の対角成分と非対角成分によって、 <i>A</i> の対応する成分が上書きされる。
<i>m</i>	INTEGER。検出された固有値の総数。( $0 \leq m \leq n$ )。 <i>range</i> = 'A' の場合、 $m = n$ 、 <i>range</i> = 'I' の場合、 $m = iu - il + 1$ 。
<i>w</i>	REAL (chpevx の場合 ) DOUBLE PRECISION (zhpevx の場合 ) 配列、次元は $\max(1, n)$ 以上。 <i>info</i> = 0 の場合、選択された <i>A</i> の固有値が昇順に格納される。
<i>z</i>	COMPLEX (chpevx の場合 ) DOUBLE COMPLEX (zhpevx の場合 ) 配列 <i>z</i> ( <i>ldz</i> ,*)。 <i>z</i> の第 2 次元は、 $\max(1, m)$ 以上でなければならない。 <i>jobz</i> = 'V' の場合、 <i>info</i> = 0 であれば、 <i>z</i> の先頭の <i>m</i> 列に、行列 <i>A</i> の選択された固有値に対応する正規直交固有ベクトルが格納される。すなわち、 <i>w</i> ( <i>i</i> ) に対応する固有ベクトルが、 <i>z</i> の <i>i</i> 番目の列に格納される。固有ベクトルが収束できない場合、 <i>z</i> のその列には、その固有ベクトルに対する最新の近似値が格納され、固有ベクトルのインデックスは <i>ifail</i> に返される。 <i>jobz</i> = 'N' の場合、 <i>z</i> は参照されない。 注：配列 <i>z</i> には、 $\max(1, m)$ 以上の列が提供されなければならない。 <i>range</i> = 'V' の場合、 <i>m</i> の正確な値が事前にわからないため、上限値を使用する必要がある。

<i>ifail</i>	INTEGER。配列、次元は $\max(1, n)$ 以上。 <i>jobz</i> ='V' の場合、 <i>info</i> =0 であれば、 <i>ifail</i> の先頭から <i>m</i> 個の成分はゼロになる。 <i>info</i> >0 であれば、収束に失敗した固有ベクトルのインデックスが <i>ifail</i> に格納される。 <i>jobz</i> ='N' の場合、 <i>ifail</i> は参照されない。
<i>info</i>	INTEGER。 <i>info</i> =0 の場合、実行は正常に終了したことを示す。 <i>info</i> =- <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正であったことを示す。 <i>info</i> = <i>i</i> の場合、 <i>i</i> 番目の固有ベクトルが収束に失敗した。その固有ベクトルのインデックスが配列 <i>ifail</i> に格納される。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *hpevx* のインターフェイスの詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ap</i> を意味する。 サイズ $(n*(n+1)/2)$ の配列 <i>A</i> を格納する。
<i>w</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>z</i>	サイズ ( <i>n,m</i> ) の行列 <i>Z</i> を格納する。ここで、値 <i>n</i> と <i>m</i> は有意である。
<i>ifail</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>v1</i>	この成分のデフォルト値は、 <i>v1</i> = -HUGE( <i>v1</i> ) である。
<i>vu</i>	この成分のデフォルト値は、 <i>vu</i> = HUGE( <i>v1</i> ) である。
<i>il</i>	この引数のデフォルト値は、 <i>il</i> = 1 である。
<i>iu</i>	この引数のデフォルト値は、 <i>iu</i> = <i>n</i> である。
<i>abstol</i>	この成分のデフォルト値は、 <i>abstol</i> = 0.0_WP である。
<i>jobz</i>	引数 <i>z</i> の存在に基づいて以下のように復元される。 <i>jobz</i> = 'V' ( <i>z</i> が存在する場合)、 <i>jobz</i> = 'N' ( <i>z</i> が省略された場合)。 <i>ifail</i> が存在し、 <i>z</i> が省略された場合、エラー条件がセットされる。

*range*            引数 *vl*、*vu*、*il*、および *iu* の存在に基づいて以下のように復元される。

*range* = 'V' (*vl* と *vu* のいずれかまたは両方が存在する場合)、  
*range* = 'I' (*il* と *iu* のいずれかまたは両方が存在する場合)、  
*range* = 'A' (*vl*、*vu*、*il*、および *iu* がすべて存在しない場合。  
*vl* と *vu* のいずれかまたは両方が存在し、同時に *il* と *iu* のいずれかまたは両方が存在する場合、エラー条件がセットされる。

## アプリケーション・ノート

$abstol + \varepsilon * \max(|a|, |b|)$  以下の幅の区間  $[a, b]$  内に存在していると判別された場合 ( $\varepsilon$  はマシン精度)、近似固有値は、収束したものとして受け入れられる。*abstol* がゼロ以下の場合、 $\varepsilon * \|T\|_1$  は所定の場所で使用される ( $T$  は、 $A$  を三重対角形式に縮退して獲得した三重対角行列)。

固有値が最も正確に計算されるのは、*abstol* がゼロでなく、アンダーフローのしきい値の 2 倍、 $2 * \lambda_{\text{mach}}('S')$  に設定されたときである。このルーチンで *info* > 0 が返された場合、固有ベクトルのいくつか収束に失敗したことを意味するので、*abstol* を  $2 * \lambda_{\text{mach}}('S')$  に設定して、やり直してみる。

## ?sbev

帯形式の実対称行列の固有値と (オプションで)  
固有ベクトルをすべて計算する。

### 構文

#### Fortran 77:

```
call ssbev(jobz, uplo, n, kd, ab, ldab, w, z, ldz, work, info)
call dsbev(jobz, uplo, n, kd, ab, ldab, w, z, ldz, work, info)
```

#### Fortran 95:

```
call sbev(a, w [,uplo] [,z] [,info])
```

### 説明

このルーチンは、帯形式の実対称行列  $A$  の固有値と (オプションで) 固有ベクトルをすべて計算する。

## 入力パラメータ

<i>jobz</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobz</i> = 'N' の場合、固有値のみを計算する。 <i>jobz</i> = 'V' の場合、固有値と固有ベクトルを計算する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> = 'U' の場合、 <i>ab</i> に <i>A</i> の上三角部分を格納する。 <i>uplo</i> = 'L' の場合、 <i>ab</i> に <i>A</i> の下三角部分を格納する。
<i>n</i>	INTEGER。行列 <i>A</i> の次数 ( $n \geq 0$ )。
<i>kd</i>	INTEGER。 <i>A</i> の優対角成分または劣対角成分の数 ( $kd \geq 0$ )。
<i>ab, work</i>	REAL (ssbev の場合 ) DOUBLE PRECISION (dsbev の場合 )。 配列： <i>ab</i> ( <i>ldab</i> ,*) は、( <i>uplo</i> の値に従って ) 対称行列 <i>A</i> の上または下三角部分を帯形式で格納する配列である。 <i>ab</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。  <i>work</i> (*) は、ワークスペース配列である。 <i>work</i> の次元は、 $\max(1, 3n-2)$ 以上でなければならない。
<i>ldab</i>	INTEGER。 <i>ab</i> のリーディング・ディメンジョンのサイズ。 $kd+1$ 以上でなければならない。
<i>ldz</i>	INTEGER。出力配列 <i>z</i> のリーディング・ディメンジョン。 次の制約がある。 <i>jobz</i> = 'N' の場合、 $ldz \geq 1$ 。 <i>jobz</i> = 'V' の場合、 $ldz \geq \max(1, n)$ 。

## 出力パラメータ

<i>w, z</i>	REAL (ssbev の場合 ) DOUBLE PRECISION (dsbev の場合 ) 配列： <i>w</i> (*)、次元は $\max(1, n)$ 以上。 <i>info</i> = 0 の場合、行列 <i>A</i> の固有値が昇順に格納される。  <i>z</i> ( <i>ldz</i> ,*)。 <i>z</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>jobz</i> = 'V' の場合、 <i>info</i> = 0 であれば、 <i>z</i> に、行列 <i>A</i> の正規直交固有ベクトルが格納される。そのとき、 <i>w</i> ( <i>i</i> ) に対応する固有ベクトルが、 <i>z</i> の <i>i</i> 番目の列に格納される。 <i>jobz</i> = 'N' の場合、 <i>z</i> は参照されない。
-------------	--



<i>ab</i>	終了時に、三重対角形式に縮退させたときに生成された値で上書きされる。 <i>uplo</i> = 'U' の場合、三重対角行列 <i>T</i> の第一優対角成分と対角成分が、 <i>ab</i> の 2 つの行 ( <i>kd</i> と <i>kd</i> +1) に返される。 <i>uplo</i> = 'L' の場合、 <i>T</i> の対角成分と第一劣対角成分が、 <i>ab</i> の先頭 2 行に返される。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正であったことを示す。 <i>info</i> = <i>i</i> の場合、このアルゴリズムが収束に失敗したことを示す。 <i>i</i> は、中間の三重対角形式で、ゼロに収束しなかった成分の個数を表す。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *sbev* のインターフェイスの詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ab</i> を意味する。サイズ ( <i>kd</i> +1, <i>n</i> ) の配列 <i>A</i> を格納する。
<i>w</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>z</i>	サイズ ( <i>n</i> , <i>n</i> ) の行列 <i>Z</i> を格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>jobz</i>	引数 <i>z</i> の存在に基づいて以下のように復元される。 <i>jobz</i> = 'V' ( <i>z</i> が存在する場合)、 <i>jobz</i> = 'N' ( <i>z</i> が省略された場合)。

## ?hbev

帯形式のエルミート行列の固有値と ( オプションで ) 固有ベクトルをすべて計算する。

### 構文

#### Fortran 77:

```
call chbev(jobz, uplo, n, kd, ab, ldab, w, z, ldz, work, rwork, info)
call zhbev(jobz, uplo, n, kd, ab, ldab, w, z, ldz, work, rwork, info)
```

#### Fortran 95:

```
call hbev(a, w [,uplo] [,z] [,info])
```

### 説明

このルーチンは、帯形式の複素エルミート行列  $A$  の固有値と ( オプションで ) 固有ベクトルをすべて計算する。

### 入力パラメータ

<i>jobz</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobz</i> = 'N' の場合、固有値のみを計算する。 <i>jobz</i> = 'V' の場合、固有値と固有ベクトルを計算する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> = 'U' の場合、 <i>ab</i> に $A$ の上三角部分を格納する。 <i>uplo</i> = 'L' の場合、 <i>ab</i> に $A$ の下三角部分を格納する。
<i>n</i>	INTEGER。行列 $A$ の次数 ( $n \geq 0$ )。
<i>kd</i>	INTEGER。 $A$ の優対角成分または劣対角成分の数 ( $kd \geq 0$ )。
<i>ab, work</i>	COMPLEX (chbev の場合 ) DOUBLE COMPLEX (zhbev の場合 )。 配列 : <i>ab</i> ( <i>ldab</i> ,*) は、( <i>uplo</i> の値に従って ) エルミート行列 $A$ の上または下三角部分を帯形式で格納する配列である。 <i>ab</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。  <i>work</i> (*) は、ワークスペース配列である。 <i>work</i> の次元は、 $\max(1, n)$ 以上でなければならない。

<i>ldab</i>	INTEGER。 <i>ab</i> のリーディング・ディメンジョンのサイズ。 $kd+1$ 以上でなければならない。
<i>ldz</i>	INTEGER。 出力配列 <i>z</i> のリーディング・ディメンジョン。 次の制約がある。 <i>jobz</i> = 'N' の場合、 $ldz \geq 1$ 。 <i>jobz</i> = 'V' の場合、 $ldz \geq \max(1, n)$ 。
<i>rwork</i>	REAL ( <i>chbev</i> の場合 ) DOUBLE PRECISION ( <i>zhbev</i> の場合 ) ワークスペース配列、次元は $\max(1, 3n-2)$ 以上。

### 出力パラメータ

<i>w</i>	REAL ( <i>chbev</i> の場合 ) DOUBLE PRECISION ( <i>zhbev</i> の場合 ) 配列、次元は $\max(1, n)$ 以上。 <i>info</i> = 0 の場合、固有値が昇順で格納される。
<i>z</i>	COMPLEX ( <i>chbev</i> の場合 ) DOUBLE COMPLEX ( <i>zhbev</i> の場合 )。 配列 <i>z</i> ( <i>ldz</i> ,*)。 <i>z</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>jobz</i> = 'V' の場合、 <i>info</i> = 0 であれば、 <i>z</i> に、行列 <i>A</i> の正規直交固有ベクトルが格納される。そのとき、 <i>w</i> ( <i>i</i> ) に対応する固有ベクトルが、 <i>z</i> の <i>i</i> 番目の列に格納される。 <i>jobz</i> = 'N' の場合、 <i>z</i> は参照されない。
<i>ab</i>	終了時に、三重対角形式に縮退させたときに生成された値で上書きされる。 <i>uplo</i> = 'U' の場合、三重対角行列 <i>T</i> の第一優対角成分と対角成分が、 <i>ab</i> の 2 つの行 ( <i>kd</i> と <i>kd+1</i> ) に返される。 <i>uplo</i> = 'L' の場合、 <i>T</i> の対角成分と第一劣対角成分が、 <i>ab</i> の先頭 2 行に返される。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正であったことを示す。 <i>info</i> = <i>i</i> の場合、このアルゴリズムが収束に失敗したことを示す。 <i>i</i> は、中間の三重対角形式で、ゼロに収束しなかった成分の個数を表す。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `hbevd` のインターフェイスの詳細を以下に示す。

<code>a</code>	Fortran 77 インターフェイスでの引数 <code>ab</code> を意味する。 サイズ $(kd+1, n)$ の配列 $A$ を格納する。
<code>w</code>	長さ $(n)$ のベクトルを格納する。
<code>z</code>	サイズ $(n, n)$ の行列 $Z$ を格納する。
<code>uplo</code>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<code>jobz</code>	引数 <code>z</code> の存在に基づいて以下のように復元される。 <code>jobz = 'V'</code> ( <code>z</code> が存在する場合)、 <code>jobz = 'N'</code> ( <code>z</code> が省略された場合)。

---

### ?sbevd

分割統治アルゴリズムを使用して、帯形式の実対称行列の固有値と (オプションで) 固有ベクトルをすべて計算する。

---

#### 構文

##### Fortran 77:

```
call ssbevd(job, uplo, n, kd, ab, ldab, w, z, ldz, work, lwork,  
            iwork, liwork, info)  
call dsbevd(job, uplo, n, kd, ab, ldab, w, z, ldz, work, lwork,  
            iwork, liwork, info)
```

##### Fortran 95:

```
call sbevd(a, w [,uplo] [,z] [,info])
```

#### 説明

このルーチンは、帯形式の実対称行列  $A$  の固有値と (オプションで) 固有ベクトルをすべて計算する。言い換えると、次のような  $A$  のスペクトル因子分解を求めることができる。

$$A = Z\Lambda Z^T$$

ここで、 $\Lambda$  は対角成分が固有値  $\lambda_i$  であるような対角行列、 $Z$  は各列が固有ベクトル  $z_i$  であるような直交行列である。つまり、次のように表現できる。

$$Az_i = \lambda_i z_i \quad \text{ただし } i = 1, 2, \dots, n$$

このルーチンは、固有ベクトルも要求された場合は、分割統治アルゴリズムを使用して、固有値と固有ベクトルを計算する。ただし、固有値のみ要求された場合は、Pal-Walker-Kahan 変形版の  $QL$  または  $QR$  アルゴリズムを使用して、固有値を計算する。

### 入力パラメータ

<i>job</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>job</i> ='N' の場合、固有値のみを計算する。 <i>job</i> ='V' の場合、固有値と固有ベクトルを計算する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> ='U' の場合、 <i>ab</i> に $A$ の上三角部分を格納する。 <i>uplo</i> ='L' の場合、 <i>ab</i> に $A$ の下三角部分を格納する。
<i>n</i>	INTEGER。行列 $A$ の次数 ( $n \geq 0$ )。
<i>kd</i>	INTEGER。 $A$ の優対角成分または劣対角成分の数 ( $kd \geq 0$ )。
<i>ab, work</i>	REAL (ssbevd の場合 ) DOUBLE PRECISION (dsbevd の場合 )。 配列： <i>ab</i> ( <i>ldab</i> ,*) は、( <i>uplo</i> の値に従って ) 対称行列 $A$ の上または下三角部分を帯形式で格納する配列である。 <i>ab</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。  <i>work</i> (*) は、ワークスペース配列である。 <i>work</i> の次元は、 <i>lwork</i> 以上でなければならない。
<i>ldab</i>	INTEGER。 <i>ab</i> のリーディング・ディメンジョンのサイズ。 $kd+1$ 以上でなければならない。
<i>ldz</i>	INTEGER。出力配列 $z$ のリーディング・ディメンジョン。 次の制約がある。 <i>job</i> ='N' の場合、 $ldz \geq 1$ 。 <i>job</i> ='V' の場合、 $ldz \geq \max(1, n)$ 。
<i>lwork</i>	INTEGER。配列 <i>work</i> の次元。 次の制約がある。 $n \leq 1$ の場合、 $lwork \geq 1$ 。 <i>job</i> ='N' で $n > 1$ の場合、 $lwork \geq 2n$ 。 <i>job</i> ='V' で $n > 1$ の場合、 $lwork \geq 3n^2 + (4+2k) * n + 1$ ( ここで、 $k$ は、 $2^k \geq n$ を満たす最小の整数 )。 <i>lwork</i> = -1 の場合、ワークスペースのクエリとみなされ、ルーチンは

*work* 配列の最適サイズだけを計算し、その値を *work* 配列の最初のエントリとして返す。xerbla は *lwork* に関するエラー・メッセージを生成しない。

*iwork* INTEGER。  
ワークスペース配列、次元は *liwork* 以上。

*liwork* INTEGER。配列 *iwork* の次元。  
次の制約がある。  
 $n \leq 1$  の場合、 $liwork \geq 1$ 。  
 $job = 'N'$  で  $n > 1$  の場合、 $liwork \geq 1$ 。  
 $job = 'V'$  で  $n > 1$  の場合、 $liwork \geq 5n+2$ 。

*liwork* = -1 の場合、ワークスペースのクエリとみなされ、ルーチンは *iwork* 配列の最適サイズだけを計算し、その値を *iwork* 配列の最初のエントリとして返す。xerbla は *liwork* に関するエラー・メッセージを生成しない。

## 出力パラメータ

*w, z* REAL (ssbevd の場合 )  
DOUBLE PRECISION (dsbevd の場合 )  
配列 :  
*w*(\*)、次元は  $\max(1, n)$  以上。  
*info* = 0 の場合、行列 *A* の固有値が昇順に格納される。*info* も参照のこと。  
*z*(*ldz*,\*)。 *z* の第 2 次元は、1 以上 ( $job = 'N'$  の場合 ) または  $\max(1, n)$  以上 ( $job = 'V'$  の場合 ) でなければならない。  
 $job = 'V'$  の場合、この配列は、*A* の固有ベクトルの入った直交行列 *Z* で上書きされる。すなわち、固有値 *w*(*i*) に対応する固有ベクトルが、*Z* の *i* 番目の列に格納される。  
 $job = 'N'$  の場合、*z* は参照されない。

*ab* 終了時に、三重対角形式に縮退させたときに生成された値で上書きされる。

*work*(1) 終了時に、*lwork* > 0 の場合、*work*(1) は *lwork* の必要最小サイズを返す。

*iwork*(1) 終了時に、*liwork* > 0 の場合、*iwork*(1) は *liwork* の必要最小サイズを返す。

*info* INTEGER。  
*info* = 0 の場合、実行は正常に終了したことを示す。  
*info* = *i* の場合、このアルゴリズムが収束に失敗したことを示す。  
*i* は、中間の三重対角形式で、ゼロに収束しなかった成分の個数を表す。  
*info* = -*i* の場合、*i* 番目のパラメータの値が不正であったことを示す。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `sbevd` のインターフェイスの詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ab</i> を意味する。サイズ $(kd+1, n)$ の配列 <i>A</i> を格納する。
<i>w</i>	長さ $(n)$ のベクトルを格納する。
<i>z</i>	サイズ $(n, n)$ の行列 <i>Z</i> を格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>jobz</i>	引数 <i>z</i> の存在に基づいて以下のように復元される。 <i>jobz</i> = 'V' ( <i>z</i> が存在する場合)、 <i>jobz</i> = 'N' ( <i>z</i> が省略された場合)。

### アプリケーション・ノート

計算で求めた固有値と固有ベクトルは、正確には、 $\|E\|_2 = O(\epsilon) \|T\|_2$  のような行列  $T + E$  ( $\epsilon$  はマシン精度) のものである。

このルーチンの複素数版は、[?hbevd](#) である。

フル形式の行列に対する [?syevd](#) と、圧縮形式の行列に関する [?spevd](#) も参照のこと。

### ?hbevd

分割統治アルゴリズムを使用して、帯形式の複素エルミート行列の固有値と (オプションで) 固有ベクトルをすべて計算する。

---

#### 構文

##### Fortran 77:

```
call chbevd(job, uplo, n, kd, ab, ldab, w, z, ldz, work, lwork,  
            rwork, lrwork, iwork, liwork, info)  
call zhbevd(job, uplo, n, kd, ab, ldab, w, z, ldz, work, lwork,  
            rwork, lrwork, iwork, liwork, info)
```

##### Fortran 95:

```
call hbevd(a, w [,uplo] [,z] [,info])
```

#### 説明

このルーチンは、帯形式の複素エルミート行列  $A$  の固有値と (オプションで) 固有ベクトルをすべて計算する。言い換えると、 $A$  のスペクトル因子分解  $A = Z\Lambda Z^H$  を求めることができる。 $\Lambda$  は対角成分が固有値  $\lambda_i$  であるような実対角行列、 $Z$  は各列が固有ベクトル  $z_i$  であるような (複素) ユニタリ行列である。つまり、次のように表現できる。

$$Az_i = \lambda_i z_i \quad \text{ただし } i = 1, 2, \dots, n$$

このルーチンは、固有ベクトルも要求された場合は、分割統治アルゴリズムを使用して、固有値と固有ベクトルを計算する。ただし、固有値のみ要求された場合は、Pal-Walker-Kahan 変形版の  $QL$  または  $QR$  アルゴリズムを使用して、固有値を計算する。

#### 入力パラメータ

<i>job</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>job</i> = 'N' の場合、固有値のみを計算する。 <i>job</i> = 'V' の場合、固有値と固有ベクトルを計算する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> = 'U' の場合、 <i>ab</i> に $A$ の上三角部分を格納する。 <i>uplo</i> = 'L' の場合、 <i>ab</i> に $A$ の下三角部分を格納する。
<i>n</i>	INTEGER。行列 $A$ の次数 ( $n \geq 0$ )。



<i>kd</i>	INTEGER。 <i>A</i> の優対角成分または劣対角成分の数 ( $kd \geq 0$ )。
<i>ab, work</i>	COMPLEX (chbevd の場合 ) DOUBLE COMPLEX (zhbevd の場合 )。 配列 : <i>ab</i> ( <i>ldab</i> ,*) は、( <i>uplo</i> の値に従って ) エルミート行列 <i>A</i> の上または下三角部分を帯形式で格納する配列である。 <i>ab</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。  <i>work</i> (*) は、ワークスペース配列である。 <i>work</i> の次元は、 <i>lwork</i> 以上でなければならない。
<i>ldab</i>	INTEGER。 <i>ab</i> のリーディング・ディメンジョンのサイズ。 $kd+1$ 以上でなければならない。
<i>ldz</i>	INTEGER。 出力配列 <i>z</i> のリーディング・ディメンジョン。 次の制約がある。 <i>job</i> ='N' の場合、 $ldz \geq 1$ 。 <i>job</i> ='V' の場合、 $ldz \geq \max(1, n)$ 。
<i>lwork</i>	INTEGER。 配列 <i>work</i> の次元。 次の制約がある。 $n \leq 1$ の場合、 $lwork \geq 1$ 。 <i>job</i> ='N' で $n > 1$ の場合、 $lwork \geq n$ 。 <i>job</i> ='V' で $n > 1$ の場合、 $lwork \geq 2n^2$ 。 <i>lwork</i> = -1 の場合、ワークスペースのクエリとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリとして返す。xerbla は <i>lwork</i> に関するエラー・メッセージを生成しない。
<i>rwork</i>	REAL (chbevd の場合 ) DOUBLE PRECISION (zhbevd の場合 ) ワークスペース配列、次元は <i>lrwork</i> 以上。
<i>lrwork</i>	INTEGER。 配列 <i>rwork</i> の次元。 次の制約がある。 $n \leq 1$ の場合、 $lrwork \geq 1$ 。 <i>job</i> ='N' で $n > 1$ の場合、 $lrwork \geq n$ 。 <i>job</i> ='V' で $n > 1$ の場合、 $lrwork \geq 3n^2 + (4+2k) * n + 1$ ( ここで、 <i>k</i> は、 $2^k \geq n$ を満たす最小の整数 )。 <i>lrwork</i> = -1 の場合、ワークスペースのクエリとみなされ、ルーチンは <i>rwork</i> 配列の最適サイズだけを計算し、その値を <i>rwork</i> 配列の最初のエントリとして返す。xerbla は <i>lrwork</i> に関するエラー・メッセージを生成しない。

*iwork* INTEGER。  
ワークスペース配列、次元は *liwork* 以上。

*liwork* INTEGER。配列 *iwork* の次元。  
次の制約がある。  
*job*='N' または  $n \leq 1$  の場合、 $liwork \geq 1$ 。  
*job*='V' で  $n > 1$  の場合、 $liwork \geq 5n+2$ 。

*liwork* = -1 の場合、ワークスペースのクエリとみなされ、ルーチンは *iwork* 配列の最適サイズだけを計算し、その値を *iwork* 配列の最初のエントリとして返す。xerbla は *liwork* に関するエラー・メッセージを生成しない。

## 出力パラメータ

*w* REAL (chbevd の場合 )  
DOUBLE PRECISION (zhbevd の場合 )  
配列、次元は  $\max(1, n)$  以上。  
*info* = 0 の場合、行列 *A* の固有値が昇順に格納される。*info* も参照のこと。

*z* COMPLEX (chbevd の場合 )  
DOUBLE COMPLEX (zhbevd の場合 )  
配列、次元は (*ldz*, \* )。 *z* の第 2 次元は、1 以上 (*job*='N' の場合 ) または  $\max(1, n)$  以上 (*job*='V' の場合 ) でなければならない。  
*job*='V' の場合、この配列は、*A* の固有ベクトルの入ったユニタリ行列 *Z* で上書きされる。すなわち、固有値 *w*(*i*) に対応する固有ベクトルが、*Z* の *i* 番目の列に格納される。  
*job*='N' の場合、*z* は参照されない。

*ab* 終了時に、三重対角形式に縮退させたときに生成された値で上書きされる。

*work*(1) 終了時に、*lwork* > 0 の場合、*work*(1) の実数部は *lwork* の必要最小サイズを返す。

*rwork*(1) 終了時に、*lrwork* > 0 の場合、*rwork*(1) は *lrwork* の必要最小サイズを返す。

*iwork*(1) 終了時に、*liwork* > 0 の場合、*iwork*(1) は *liwork* の必要最小サイズを返す。

*info* INTEGER。  
*info* = 0 の場合、実行は正常に終了したことを示す。  
*info* = *i* の場合、このアルゴリズムが収束に失敗したことを示す。

$i$  は、中間の三重対角形式で、ゼロに収束しなかった成分の個数を表す。

$info = -i$  の場合、 $i$  番目のパラメータの値が不正であったことを示す。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `hbevd` のインターフェイスの詳細を以下に示す。

<code>a</code>	Fortran 77 インターフェイスでの引数 <code>ab</code> を意味する。サイズ $(kd+1, n)$ の配列 $A$ を格納する。
<code>w</code>	長さ $(n)$ のベクトルを格納する。
<code>z</code>	サイズ $(n, n)$ の行列 $Z$ を格納する。
<code>uplo</code>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<code>jobz</code>	引数 $z$ の存在に基づいて以下のように復元される。 <code>jobz = 'V'</code> ( $z$ が存在する場合)、 <code>jobz = 'N'</code> ( $z$ が省略された場合)。

### アプリケーション・ノート

計算で求めた固有値と固有ベクトルは、正確には、 $\|E\|_2 = O(\epsilon) \|T\|_2$  のような行列  $T + E$  ( $\epsilon$  はマシン精度) のものである。

このルーチンの実数版は、[?sbevd](#) である。

フル形式の行列に対する [?heevd](#) と、圧縮形式の行列に関する [?hpevd](#) も参照のこと。

### ?sbevx

帯形式の実対称行列の固有値と ( オプションで )  
固有ベクトルを選択的に計算する。

---

#### 構文

##### Fortran 77:

```
call ssbevx(jobz, range, uplo, n, kd, ab, ldab, q, ldq, vl, vu, il, iu, abstol,  
            m, w, z, ldz, work, iwork, ifail, info)  
call dsbevx(jobz, range, uplo, n, kd, ab, ldab, q, ldq, vl, vu, il, iu, abstol,  
            m, w, z, ldz, work, iwork, ifail, info)
```

##### Fortran 95:

```
call sbevx(a, w [,uplo] [,z] [,vl] [,vu] [,il] [,iu] [,m] [,ifail] [,q]  
            [,abstol] [,info])
```

#### 説明

このルーチンは、帯形式の実対称行列  $A$  の固有値と ( オプションで ) 固有ベクトルを選択的に計算する。固有値と固有ベクトルを選択するには、希望する固有値の値の範囲かインデックスの範囲を指定する。

#### 入力パラメータ

<i>jobz</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobz</i> = 'N' の場合、固有値のみを計算する。 <i>jobz</i> = 'V' の場合、固有値と固有ベクトルを計算する。
<i>range</i>	CHARACTER*1。'A'、'V'、または 'I' のいずれかでなければならない。 <i>range</i> = 'A' の場合、固有値をすべて計算する。 <i>range</i> = 'V' の場合、半開区間 $vl < \lambda_i \leq vu$ における固有値 $\lambda_i$ を計算する。 <i>range</i> = 'I' の場合、ルーチンはインデックスが <i>il</i> ~ <i>iu</i> である固有値を計算する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> = 'U' の場合、 <i>ab</i> に $A$ の上三角部分を格納する。 <i>uplo</i> = 'L' の場合、 <i>ab</i> に $A$ の下三角部分を格納する。
<i>n</i>	INTEGER。行列 $A$ の次数 ( $n \geq 0$ )。

<i>kd</i>	INTEGER。 <i>A</i> の優対角成分または劣対角成分の数 ( $kd \geq 0$ )。
<i>ab, work</i>	REAL (ssbevz の場合 ) DOUBLE PRECISION (dsbevz の場合 )。 配列： <i>ab</i> ( <i>ldab</i> ,*) は、( <i>uplo</i> の値に従って ) 対称行列 <i>A</i> の上または下三角部分を帯形式で格納する配列である。 <i>ab</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。  <i>work</i> (*) は、ワークスペース配列である。 <i>work</i> の次元は、 $\max(1, 7n)$ 以上でなければならない。
<i>ldab</i>	INTEGER。 <i>ab</i> のリーディング・ディメンジョンのサイズ。 $kd+1$ 以上でなければならない。
<i>v1, vu</i>	REAL (ssbevz の場合 ) DOUBLE PRECISION (dsbevz の場合 )。 <i>range</i> = 'V' の場合、固有値を検索する区間の上限と下限。 次の制約がある。 $v1 < vu$ 。 <i>range</i> = 'A' または 'I' の場合、 <i>v1</i> と <i>vu</i> は参照されない。
<i>il, iu</i>	INTEGER。 <i>range</i> = 'I' の場合、返される最小固有値と最大固有値に対する昇順のインデックス。 次の制約がある。 $1 \leq il \leq iu \leq n$ ( $n > 0$ の場合 )。 <i>il</i> =1 かつ <i>iu</i> =0 ( $n = 0$ の場合 )。 <i>range</i> = 'A' または 'V' の場合、 <i>il</i> と <i>iu</i> は参照されない。
<i>abstol</i>	REAL (chpevx の場合 ) DOUBLE PRECISION (zhpevx の場合 ) 各固有値に許される絶対誤差許容値。誤差許容値の詳細は、「アプリケーション・ノート」を参照。
<i>ldq, ldz</i>	INTEGER。出力配列 <i>q</i> と <i>z</i> のリーディング・ディメンジョン。次の制約がある。 $ldq \geq 1, ldz \geq 1$ 。 <i>jobz</i> = 'V' の場合、 $ldq \geq \max(1, n)$ および $ldz \geq \max(1, n)$ 。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, 5n)$ 以上。

## 出力パラメータ

<i>q</i>	<p>REAL (ssbevx の場合 )  DOUBLE PRECISION (dsbevx の場合 )。  配列、次元は (<i>ldz</i>, <i>n</i>)。  <i>jobz</i>='V' の場合、<math>n \times n</math> の直交行列が三重対角形式への縮退に使用される。  <i>jobz</i>='N' の場合、配列 <i>q</i> は参照されない。</p>
<i>m</i>	<p>INTEGER。検出された固有値の総数。(<math>0 \leq m \leq n</math>)。 <i>range</i>='A' の場合、<math>m = n</math>、 <i>range</i>='I' の場合、<math>m = iu - il + 1</math>。</p>
<i>w</i> , <i>z</i>	<p>REAL (ssbevx の場合 )  DOUBLE PRECISION (dsbevx の場合 )  配列 :  <i>w</i>(*)、サイズは <math>\max(1, n)</math> 以上。  <i>w</i> の先頭 <i>m</i> 成分に、行列 <i>A</i> の選択された固有値が昇順に格納される。    <i>z</i>(<i>ldz</i>,*)。 <i>z</i> の第 2 次元は、<math>\max(1, m)</math> 以上でなければならない。  <i>jobz</i>='V' の場合、 <i>info</i>=0 であれば、 <i>z</i> の先頭の <i>m</i> 列に、行列 <i>A</i> の選択された固有値に対応する正規直交固有ベクトルが格納される。すなわち、 <i>w</i>(<i>i</i>) に対応する固有ベクトルが、 <i>z</i> の <i>i</i> 番目の列に格納される。固有ベクトルが収束できない場合、 <i>z</i> のその列には、その固有ベクトルに対する最新の近似値が格納され、固有ベクトルのインデックスは <i>ifail</i> に返される。  <i>jobz</i>='N' の場合、 <i>z</i> は参照されない。  注 : 配列 <i>z</i> には、<math>\max(1, m)</math> 以上の列が提供されなければならない。  <i>range</i>='V' の場合、 <i>m</i> の正確な値が事前にわからないため、上限値を使用する必要がある。</p>
<i>ab</i>	<p>終了時に、三重対角形式に縮退させたときに生成された値で上書きされる。 <i>uplo</i>='U' の場合、三重対角行列 <i>T</i> の第一優対角成分と対角成分が、 <i>ab</i> の 2 つの行 (<i>kd</i> と <i>kd</i>+1) に返される。 <i>uplo</i>='L' の場合、 <i>T</i> の対角成分と第一劣対角成分が、 <i>ab</i> の先頭 2 行に返される。</p>
<i>ifail</i>	<p>INTEGER。  配列、次元は <math>\max(1, n)</math> 以上。  <i>jobz</i>='V' の場合、 <i>info</i>=0 であれば、 <i>ifail</i> の先頭から <i>m</i> 個の成分はゼロになる。 <i>info</i>&gt;0 であれば、収束に失敗した固有ベクトルのインデックスが <i>ifail</i> に格納される。  <i>jobz</i>='N' の場合、 <i>ifail</i> は参照されない。</p>

*info* INTEGER。  
*info* = 0 の場合、実行は正常に終了したことを示す。  
*info* = -*i* の場合、*i* 番目のパラメータの値が不正であったことを示す。  
*info* = *i* の場合、*i* 番目の固有ベクトルが収束に失敗した。その固有ベクトルのインデックスが配列 *ifail* に格納される。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *sbev*x のインターフェイスの詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ab</i> を意味する。サイズ $(kd+1, n)$ の配列 <i>A</i> を格納する。
<i>w</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>z</i>	サイズ $(n, m)$ の行列 <i>Z</i> を格納する。ここで、値 <i>n</i> と <i>m</i> は有意である。
<i>ifail</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>q</i>	サイズ $(n, n)$ の行列 <i>Q</i> を格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>v1</i>	この成分のデフォルト値は、 $v1 = -HUGE(v1)$ である。
<i>vu</i>	この成分のデフォルト値は、 $vu = HUGE(v1)$ である。
<i>i1</i>	この引数のデフォルト値は、 $i1 = 1$ である。
<i>iu</i>	この引数のデフォルト値は、 $iu = n$ である。
<i>abstol</i>	この成分のデフォルト値は、 $abstol = 0.0\_WP$ である。
<i>jobz</i>	引数 <i>z</i> の存在に基づいて以下のように復元される。 <i>jobz</i> = 'V' ( <i>z</i> が存在する場合)、 <i>jobz</i> = 'N' ( <i>z</i> が省略された場合)。 <i>ifail</i> または <i>q</i> のいずれかが存在し、 <i>z</i> が省略された場合、エラー条件がセットされる。
<i>range</i>	引数 <i>v1</i> 、 <i>vu</i> 、 <i>i1</i> 、および <i>iu</i> の存在に基づいて以下のように復元される。 <i>range</i> = 'V' ( <i>v1</i> と <i>vu</i> のいずれかまたは両方が存在する場合)、 <i>range</i> = 'I' ( <i>i1</i> と <i>iu</i> のいずれかまたは両方が存在する場合)、

`range = 'A'` (`vl`、`vu`、`il`、および `iu` がすべて存在しない場合。  
`vl` と `vu` のいずれかまたは両方が存在し、同時に `il` と `iu` のいずれか  
または両方が存在する場合、エラー条件がセットされる。

### アプリケーション・ノート

$abstol + \epsilon * \max(|a|, |b|)$  以下の幅の区間  $[a, b]$  内に存在していると判別された場合 ( $\epsilon$  はマシン精度)、近似固有値は、収束したものとして受け入れられる。`abstol` がゼロ以下の場合、 $\epsilon * \|T\|_1$  は所定の場所で使用される ( $T$  は、 $A$  を三重対角形式に縮退して獲得した三重対角行列)。

固有値が最も正確に計算されるのは、`abstol` がゼロでなく、アンダーフローのしきい値の 2 倍、`2*?lamch('S')` に設定されたときである。このルーチンで `info > 0` が返された場合、固有ベクトルのいくつか収束に失敗したことを意味するので、`abstol` を `2*?lamch('S')` に設定して、やり直してみる。

---

## ?hbevz

帯形式のエルミート行列の固有値と (オプションで) 固有ベクトルを選択的に計算する。

---

### 構文

#### Fortran 77:

```
call chbevz(jobz, range, uplo, n, kd, ab, ldab, q, ldq, vl, vu, il, iu, abstol,  
            m, w, z, ldz, work, rwork, iwork, ifail, info)  
call zhbevz(jobz, range, uplo, n, kd, ab, ldab, q, ldq, vl, vu, il, iu, abstol,  
            m, w, z, ldz, work, rwork, iwork, ifail, info)
```

#### Fortran 95:

```
call hbevz(a, w [,uplo] [,z] [,vl] [,vu] [,il] [,iu] [,m] [,ifail] [,q]  
            [,abstol] [,info])
```

### 説明

このルーチンは、帯形式の複素エルミート行列  $A$  の固有値と (オプションで) 固有ベクトルを選択的に計算する。固有値と固有ベクトルを選択するには、希望する固有値の値の範囲かインデックスの範囲を指定する。



**入力パラメータ**

<i>jobz</i>	CHARACTER*1. 'N' または 'V' でなければならない。 <i>job</i> ='N' の場合、固有値のみを計算する。 <i>job</i> ='V' の場合、固有値と固有ベクトルを計算する。
<i>range</i>	CHARACTER*1. 'A'、'V'、または 'I' のいずれかでなければならない。 <i>range</i> ='A' の場合、固有値をすべて計算する。 <i>range</i> ='V' の場合、半開区間 $v_l < \lambda_i \leq v_u$ における固有値 $\lambda_i$ を計算する。 <i>range</i> ='I' の場合、ルーチンはインデックスが $i_l \sim i_u$ である固有値を計算する。
<i>uplo</i>	CHARACTER*1. 'U' または 'L' でなければならない。 <i>uplo</i> ='U' の場合、 <i>ab</i> に <i>A</i> の上三角部分を格納する。 <i>uplo</i> ='L' の場合、 <i>ab</i> に <i>A</i> の下三角部分を格納する。
<i>n</i>	INTEGER. 行列 <i>A</i> の次数 ( $n \geq 0$ ).
<i>kd</i>	INTEGER. <i>A</i> の優対角成分または劣対角成分の数 ( $kd \geq 0$ ).
<i>ab, work</i>	COMPLEX (chbevz の場合) DOUBLE COMPLEX (zhbevz の場合). 配列: <i>ab</i> ( <i>ldab</i> ,*) は、( <i>uplo</i> の値に従って) エルミート行列 <i>A</i> の上または下三角部分を帯形式で格納する配列である。 <i>ab</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>work</i> (*) は、ワークスペース配列である。 <i>work</i> の次元は、 $\max(1, n)$ 以上でなければならない。
<i>ldab</i>	INTEGER. <i>ab</i> のリーディング・ディメンジョンのサイズ。 $kd+1$ 以上でなければならない。
<i>v_l, v_u</i>	REAL (chbevz の場合) DOUBLE PRECISION (zhbevz の場合). <i>range</i> ='V' の場合、固有値を検索する区間の上限と下限。 次の制約がある。 $v_l < v_u$ . <i>range</i> ='A' または 'I' の場合、 <i>v_l</i> と <i>v_u</i> は参照されない。
<i>i_l, i_u</i>	INTEGER. <i>range</i> ='I' の場合、返される最小固有値と最大固有値に対する昇順のインデックス。 次の制約がある。

	$1 \leq il \leq iu \leq n$ ( $n > 0$ の場合)。 $il=1$ かつ $iu=0$ ( $n=0$ の場合)。 $range='A'$ または $'V'$ の場合、 $il$ と $iu$ は参照されない。
<i>abstol</i>	REAL (chbevz の場合) DOUBLE PRECISION (zhbevz の場合)。 各固有値に許される絶対誤差許容値。誤差許容値の詳細は、「アプリケーション・ノート」を参照。
<i>ldq, ldz</i>	INTEGER。出力配列 <i>q</i> と <i>z</i> のリーディング・ディメンション。次の制約がある。 $ldq \geq 1, ldz \geq 1$ 。 $jobz='V'$ の場合、 $ldq \geq \max(1, n)$ および $ldz \geq \max(1, n)$ 。
<i>rwork</i>	REAL (chbevz の場合) DOUBLE PRECISION (zhbevz の場合) ワークスペース配列、次元は $\max(1, 7n)$ 以上。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, 5n)$ 以上。

## 出力パラメータ

<i>q</i>	COMPLEX (chbevz の場合) DOUBLE COMPLEX (zhbevz の場合)。 配列、次元は $(ldz, n)$ 。 $jobz='V'$ の場合、 $n \times n$ のユニタリ行列が三重対角形式への縮退に使用される。 $jobz='N'$ の場合、配列 <i>q</i> は参照されない。
<i>m</i>	INTEGER。検出された固有値の総数。( $0 \leq m \leq n$ )。 $range='A'$ の場合、 $m=n$ 、 $range='I'$ の場合、 $m=iu-il+1$ 。
<i>w</i>	REAL (chbevz の場合) DOUBLE PRECISION (zhbevz の場合) 配列、次元は $\max(1, n)$ 以上。 先頭の <i>m</i> 個の成分に、行列 <i>A</i> の選択された固有値が昇順に格納される。
<i>z</i>	COMPLEX (chbevz の場合) DOUBLE COMPLEX (zhbevz の場合)。 配列 <i>z</i> ( <i>ldz</i> ,*)。 <i>z</i> の第 2 次元は、 $\max(1, m)$ 以上でなければならない。 $jobz='V'$ の場合、 <i>info</i> =0 であれば、 <i>z</i> の先頭の <i>m</i> 列に、行列 <i>A</i> の選択された固有値に対応する正規直交固有ベクトルが格納される。すなわち、 <i>w</i> ( <i>i</i> ) に対応する固有ベクトルが、 <i>z</i> の <i>i</i> 番目の列に格納される。

固有ベクトルが収束できない場合、 $z$  のその列には、その固有ベクトルに対する最新の近似値が格納され、固有ベクトルのインデックスは *ifail* に返される。

*jobz* = 'N' の場合、 $z$  は参照されない。

注：配列  $z$  には、 $\max(1, m)$  以上の列が提供されなければならない。

*range* = 'V' の場合、 $m$  の正確な値が事前にわからないため、上限値を使用する必要がある。

<i>ab</i>	終了時に、三重対角形式に縮退させたときに生成された値で上書きされる。 <i>uplo</i> = 'U' の場合、三重対角行列 $T$ の第一優対角成分と対角成分が、 <i>ab</i> の 2 つの行 ( $kd$ と $kd+1$ ) に返される。 <i>uplo</i> = 'L' の場合、 $T$ の対角成分と第一劣対角成分が、 <i>ab</i> の先頭 2 行に返される。
<i>ifail</i>	INTEGER。 配列、次元は $\max(1, n)$ 以上。 <i>jobz</i> = 'V' の場合、 <i>info</i> = 0 であれば、 <i>ifail</i> の先頭から $m$ 個の成分はゼロになる。 <i>info</i> > 0 であれば、収束しなかった固有ベクトルのインデックスが <i>ifail</i> に格納される。 <i>jobz</i> = 'N' の場合、 <i>ifail</i> は参照されない。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - $i$ の場合、 $i$ 番目のパラメータの値が不正であったことを示す。 <i>info</i> = $i$ の場合、 $i$ 番目の固有ベクトルが収束に失敗した。その固有ベクトルのインデックスが配列 <i>ifail</i> に格納される。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *hbev*x のインターフェイスの詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ab</i> を意味する。サイズ $(kd+1, n)$ の配列 $A$ を格納する。
<i>w</i>	長さ $(n)$ のベクトルを格納する。
<i>z</i>	サイズ $(n, m)$ の行列 $Z$ を格納する。ここで、値 $n$ と $m$ は有意である。
<i>ifail</i>	長さ $(n)$ のベクトルを格納する。
<i>q</i>	サイズ $(n, n)$ の行列 $Q$ を格納する。

<code>uplo</code>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<code>v1</code>	この成分のデフォルト値は、 $v1 = -HUGE(v1)$ である。
<code>vu</code>	この成分のデフォルト値は、 $vu = HUGE(v1)$ である。
<code>i1</code>	この引数のデフォルト値は、 $i1 = 1$ である。
<code>iu</code>	この引数のデフォルト値は、 $iu = n$ である。
<code>abstol</code>	この成分のデフォルト値は、 $abstol = 0.0\_WP$ である。
<code>jobz</code>	引数 $z$ の存在に基づいて以下のように復元される。 $jobz = 'V'$ ( $z$ が存在する場合)、 $jobz = 'N'$ ( $z$ が省略された場合)。 $ifail$ または $q$ のいずれかが存在し、 $z$ が省略された場合、エラー条件がセットされる。
<code>range</code>	引数 $v1$ 、 $vu$ 、 $i1$ 、および $iu$ の存在に基づいて以下のように復元される。 $range = 'V'$ ( $v1$ と $vu$ のいずれかまたは両方が存在する場合)、 $range = 'I'$ ( $i1$ と $iu$ のいずれかまたは両方が存在する場合)、 $range = 'A'$ ( $v1$ 、 $vu$ 、 $i1$ 、および $iu$ がすべて存在しない場合)。 $v1$ と $vu$ のいずれかまたは両方が存在し、同時に $i1$ と $iu$ のいずれかまたは両方が存在する場合、エラー条件がセットされる。

## アプリケーション・ノート

$abstol + \epsilon * \max(|a|, |b|)$  以下の幅の区間  $[a, b]$  内に存在していると判別された場合 ( $\epsilon$  はマシン精度)、近似固有値は、収束したものとして受け入れられる。 $abstol$  がゼロ以下の場合、 $\epsilon * \|T\|_1$  は所定の場所で使用される ( $T$  は、 $A$  を三重対角形式に縮退して獲得した三重対角行列)。

固有値が最も正確に計算されるのは、 $abstol$  がゼロでなく、アンダーフローのしきい値の 2 倍、 $2 * ?lamch('S')$  に設定されたときである。このルーチンで  $info > 0$  が返された場合、固有ベクトルのいくつか収束に失敗したことを意味するので、 $abstol$  を  $2 * ?lamch('S')$  に設定して、やり直してみる。

## ?stev

実対称三重対角行列の固有値と ( オプションで )  
固有ベクトルをすべて計算する。

### 構文

#### Fortran 77:

```
call sstev(jobz, n, d, e, z, ldz, work, info)
call dstev(jobz, n, d, e, z, ldz, work, info)
```

#### Fortran 95:

```
call stev(d, e [,z] [,info])
```

### 説明

このルーチンは、実対称三重対角行列  $A$  の固有値と ( オプションで ) 固有ベクトルをすべて計算する。

### 入力パラメータ

<i>jobz</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobz</i> = 'N' の場合、固有値のみを計算する。 <i>jobz</i> = 'V' の場合、固有値と固有ベクトルを計算する。
<i>n</i>	INTEGER。行列 $A$ の次数 ( $n \geq 0$ )。
<i>d, e, work</i>	REAL (sstev の場合 ) DOUBLE PRECISION (dstev の場合 )。 配列 : <i>d</i> (*) には、三重対角行列 $A$ の $n$ 個の対角成分を格納する。 <i>d</i> の次元は、 $\max(1, n)$ 以上でなければならない。 <i>e</i> (*) には、三重対角行列 $A$ の $n-1$ 個の劣対角成分を格納する。 <i>e</i> の次元は、 $\max(1, n)$ 以上でなければならない。この配列の $n$ 番目の成分は、ワークスペースとして使用される。 <i>work</i> (*) は、ワークスペース配列である。 <i>work</i> の次元は、 $\max(1, 2n-2)$ 以上でなければならない。 <i>jobz</i> = 'N' の場合、 <i>work</i> は参照されない。

*ldz* INTEGER。出力配列 *z* のリーディング・ディメンジョン。  $ldz \geq 1$ 。  
*jobz* = 'V' の場合、  $ldz \geq \max(1, n)$ 。

### 出力パラメータ

*d* 終了時に、 *info* = 0 の場合、行列 *A* の固有値が昇順に格納される。

*z* REAL (sstev の場合 )  
DOUBLE PRECISION (dstev の場合 )  
配列、次元は (*ldz*, \*)。  
*z* の第 2 次元は、  $\max(1, n)$  以上でなければならない。  
*jobz* = 'V' の場合、 *info* = 0 であれば、行列 *A* の正規直交固有ベクトルが *z* に格納される。すなわち、 *d*(*i*) に対応する固有ベクトルが、*z* の *i* 番目の列に格納される。  
*job* = 'N' の場合、 *z* は参照されない。

*e* 終了時に、中間結果で上書きされる。

*info* INTEGER。  
*info* = 0 の場合、実行は正常に終了したことを示す。  
*info* = -*i* の場合、*i* 番目のパラメータの値が不正であったことを示す。  
*info* = *i* の場合、このアルゴリズムが収束に失敗したことを示す。  
*e* の *i* 個の成分が、ゼロに収束しなかった。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *stev* のインターフェイスの詳細を以下に示す。

*d* 長さ (*n*) のベクトルを格納する。

*e* 長さ (*n*) のベクトルを格納する。

*z* サイズ (*n*, *n*) の行列 *Z* を格納する。

*jobz* 引数 *z* の存在に基づいて以下のように復元される。  
*jobz* = 'V' (*z* が存在する場合)、  
*jobz* = 'N' (*z* が省略された場合)。

## ?stevd

分割統治アルゴリズムを使用して、実対称三重対角行列の固有値と (オプションで) 固有ベクトルをすべて計算する。

### 構文

#### Fortran 77:

```
call sstevd(job, n, d, e, z, ldz, work, lwork, iwork, liwork, info)
call dstevd(job, n, d, e, z, ldz, work, lwork, iwork, liwork, info)
```

#### Fortran 95:

```
call stevd(d, e [,z] [,info])
```

### 説明

このルーチンは、実対称三重対角行列  $T$  の固有値と (オプションで) 固有ベクトルをすべて計算する。言い換えると、 $T$  のスペクトル因子分解  $T = Z\Lambda Z^T$  を求めることができる。 $\Lambda$  は対角成分が固有値  $\lambda_i$  であるような対角行列、 $Z$  は各列が固有ベクトル  $z_i$  であるような直交行列である。つまり、次のように表現できる。

$$Tz_i = \lambda_i z_i \text{ for } i = 1, 2, \dots, n$$

このルーチンは、固有ベクトルも要求された場合は、分割統治アルゴリズムを使用して、固有値と固有ベクトルを計算する。ただし、固有値のみ要求された場合は、Pal-Walker-Kahan 変形版の  $QL$  または  $QR$  アルゴリズムを使用して、固有値を計算する。

このルーチンの複素数版はない。

### 入力パラメータ

<i>job</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>job</i> ='N' の場合、固有値のみを計算する。 <i>job</i> ='V' の場合、固有値と固有ベクトルを計算する。
<i>n</i>	INTEGER。行列 $T$ の次数 ( $n \geq 0$ )。

<i>d, e, work</i>	<p>REAL (sstevd の場合 )</p> <p>DOUBLE PRECISION (dstevd の場合 )</p> <p>配列 :</p> <p><i>d</i>(*) には、三重対角行列 <i>T</i> の <i>n</i> 個の対角成分を格納する。  <i>d</i> の次元は、<math>\max(1, n)</math> 以上でなければならない。</p> <p><i>e</i>(*) には、<i>T</i> の <i>n</i>-1 個の非対角成分を格納する。  <i>e</i> の次元は、<math>\max(1, n)</math> 以上でなければならない。この配列の <i>n</i> 番目の成分は、ワークスペースとして使用される。</p> <p><i>work</i>(*) は、ワークスペース配列である。  <i>work</i> の次元は、<i>lwork</i> 以上でなければならない。</p>
<i>ldz</i>	<p>INTEGER。出力配列 <i>z</i> のリーディング・ディメンジョン。次の制約がある。</p> <p><math>ldz \geq 1</math> (<i>job</i>='N' の場合 )。</p> <p><math>ldz \geq \max(1, n)</math> (<i>job</i>='V' の場合 )。</p>
<i>lwork</i>	<p>INTEGER。配列 <i>work</i> の次元。          次の制約がある。</p> <p><i>job</i>='N' または <math>n \leq 1</math> の場合、<math>lwork \geq 1</math>。</p> <p><i>job</i>='V' で <math>n &gt; 1</math> の場合、<math>lwork \geq 2n^2 + (3+2k) * n + 1</math>          (ここで、<i>k</i> は、<math>2^k \geq n</math> を満たす最小の整数)。</p> <p><i>lwork</i> = -1 の場合、ワークスペースのクエリとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリとして返す。xerbla は <i>lwork</i> に関するエラー・メッセージを生成しない。</p>
<i>iwork</i>	<p>INTEGER。          ワークスペース配列、次元は <i>liwork</i> 以上。</p>
<i>liwork</i>	<p>INTEGER。配列 <i>iwork</i> の次元。          次の制約がある。</p> <p><i>job</i>='N' または <math>n \leq 1</math> の場合、<math>liwork \geq 1</math>。</p> <p><i>job</i>='V' で <math>n &gt; 1</math> の場合、<math>liwork \geq 5n + 2</math>。</p> <p><i>liwork</i> = -1 の場合、ワークスペースのクエリとみなされ、ルーチンは <i>iwork</i> 配列の最適サイズだけを計算し、その値を <i>iwork</i> 配列の最初のエントリとして返す。xerbla は <i>liwork</i> に関するエラー・メッセージを生成しない。</p>



**出力パラメータ**

<i>d</i>	終了時に、 <i>info</i> = 0 の場合、行列 <i>T</i> の固有値が昇順に格納される。 <i>info</i> も参照のこと。
<i>z</i>	REAL (sstevd の場合 ) DOUBLE PRECISION (dstevd の場合 ) 配列、次元は ( <i>ldz</i> , * )。 <i>z</i> の第 2 次元は、1 以上 ( <i>job</i> = 'N' の場合 ) または max(1, <i>n</i> ) 以上 ( <i>job</i> = 'V' の場合 ) でなければならない。  <i>job</i> = 'V' の場合、 <i>T</i> の固有ベクトルが格納された直交行列 <i>Z</i> で上書き される。 <i>job</i> = 'N' の場合、 <i>z</i> は参照されない。
<i>e</i>	終了時に、中間結果で上書きされる。
<i>work</i> (1)	終了時に、 <i>lwork</i> > 0 の場合、 <i>work</i> (1) は <i>lwork</i> の必要最小サイズを 返す。
<i>iwork</i> (1)	終了時に、 <i>liwork</i> > 0 の場合、 <i>iwork</i> (1) は <i>liwork</i> の必要最小サイズ を返す。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = <i>i</i> の場合、このアルゴリズムが収束に失敗したことを示す。 <i>i</i> は、中間の三重対角形式で、ゼロに収束しなかった成分の個数を表 す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正であったことを示 す。

**Fortran 95 インターフェイス・ノート**

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *stevd* のインターフェイスの詳細を以下に示す。

<i>d</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>e</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>z</i>	サイズ ( <i>n</i> , <i>n</i> ) の行列 <i>Z</i> を格納する。
<i>jobz</i>	引数 <i>z</i> の存在に基づいて以下のように復元される。 <i>jobz</i> = 'V' ( <i>z</i> が存在する場合 )、 <i>jobz</i> = 'N' ( <i>z</i> が省略された場合 )。

### アプリケーション・ノート

計算で求めた固有値と固有ベクトルは、正確には、 $\|E\|_2 = O(\varepsilon) \|T\|_2$  のような行列  $T + E$  ( $\varepsilon$  はマシン精度) のものである。

$\lambda_i$  が正確な固有値で、 $\mu_i$  が対応する計算で求めた値の場合、次のようになる。

$$|\mu_i - \lambda_i| \leq c(n)\varepsilon \|T\|_2$$

$c(n)$  は、 $n$  の漸増関数である。

$z_i$  が対応する正確な固有ベクトルで、 $w_i$  が対応する計算ベクトルの場合、それらの間の角度  $\theta(z_i, w_i)$  は次のようになる。 $\theta(z_i, w_i) \leq c(n)\varepsilon \|T\|_2 / \min_{i \neq j} |\lambda_i - \lambda_j|$

このように、計算された固有ベクトルの精度は、それが対応する固有値と、他のすべての固有値との隔たりに依存するのがわかる。

---

## ?stevx

実対称三重対角行列の固有値と (オプションで)  
固有ベクトルを選択的に計算する。

---

### 構文

#### Fortran 77:

```
call sstevx(jobz, range, n, d, e, vl, vu, il, iu, abstol, m, w, z, ldz, work,  
            iwork, ifail, info)  
call dstevx(jobz, range, n, d, e, vl, vu, il, iu, abstol, m, w, z, ldz, work,  
            iwork, ifail, info)
```

#### Fortran 95:

```
call stevx(d, e, w [,z] [,vl] [,vu] [,il] [,iu] [,m] [,ifail] [,abstol] [,info])
```

### 説明

このルーチンは、実対称三重対角行列  $A$  の固有値と (オプションで) 固有ベクトルを選択的に計算する。固有値と固有ベクトルを選択するには、希望する固有値の値の範囲かインデックスの範囲を指定する。

## 入力パラメータ

<i>jobz</i>	CHARACTER*1. 'N' または 'V' でなければならない。 <i>job</i> ='N' の場合、固有値のみを計算する。 <i>job</i> ='V' の場合、固有値と固有ベクトルを計算する。
<i>range</i>	CHARACTER*1. 'A'、'V'、または 'I' のいずれかでなければならない。 <i>range</i> ='A' の場合、固有値をすべて計算する。 <i>range</i> ='V' の場合、半開区間 $v_l < \lambda_i \leq v_u$ における固有値 $\lambda_i$ を計算する。 <i>range</i> ='I' の場合、ルーチンはインデックスが $i_l \sim i_u$ である固有値を計算する。
<i>n</i>	INTEGER。行列 <i>A</i> の次数 ( $n \geq 0$ )。
<i>d, e, work</i>	REAL (sstevx の場合 ) DOUBLE PRECISION (dstevx の場合 )。 配列： <i>d</i> (*) には、三重対角行列 <i>A</i> の <i>n</i> 個の対角成分を格納する。 <i>d</i> の次元は、 $\max(1, n)$ 以上でなければならない。  <i>e</i> (*) には、 <i>A</i> の <i>n</i> -1 個の劣対角成分を格納する。 <i>e</i> の次元は、 $\max(1, n)$ 以上でなければならない。この配列の <i>n</i> 番目の成分は、ワークスペースとして使用される。  <i>work</i> (*) は、ワークスペース配列である。 <i>work</i> の次元は、 $\max(1, 5n)$ 以上でなければならない。
<i>vl, vu</i>	REAL (sstevx の場合 ) DOUBLE PRECISION (dstevx の場合 )。 <i>range</i> ='V' の場合、固有値を検索する区間の上限と下限。 次の制約がある。 $v_l < v_u$ 。 <i>range</i> ='A' または 'I' の場合、 <i>vl</i> と <i>vu</i> は参照されない。
<i>il, iu</i>	INTEGER。 <i>range</i> ='I' の場合、返される最小固有値と最大固有値に対する昇順のインデックス。 次の制約がある。 $1 \leq i_l \leq i_u \leq n$ ( $n > 0$ の場合 )。 $i_l=1$ かつ $i_u=0$ ( $n=0$ の場合 )。 <i>range</i> ='A' または 'V' の場合、 <i>il</i> と <i>iu</i> は参照されない。

<i>abstol</i>	REAL (sstevx の場合 ) DOUBLE PRECISION (dstevx の場合 )。 各固有値に許される絶対誤差許容値。誤差許容値の詳細は、「アプリケーション・ノート」を参照。
<i>ldz</i>	INTEGER。出力配列 <i>z</i> のリーディング・ディメンジョン。 $ldz \geq 1$ 。 <i>jobz</i> = 'V' の場合、 $ldz \geq \max(1, n)$ 。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, 5n)$ 以上。

## 出力パラメータ

<i>m</i>	INTEGER。検出された固有値の総数。 $(0 \leq m \leq n)$ 。 <i>range</i> = 'A' の場合、 <i>m</i> = <i>n</i> 、 <i>range</i> = 'I' の場合、 <i>m</i> = <i>iu-il</i> +1。
<i>w</i> , <i>z</i>	REAL (sstevx の場合 ) DOUBLE PRECISION (dstevx の場合 )。 配列： <i>w</i> (*)、次元は $\max(1, n)$ 以上。 <i>w</i> の先頭 <i>m</i> 成分に、行列 <i>A</i> の選択された固有値が昇順に格納される。  <i>z</i> ( <i>ldz</i> ,*)。 <i>z</i> の第 2 次元は、 $\max(1, m)$ 以上でなければならない。 <i>jobz</i> = 'V' の場合、 <i>info</i> = 0 であれば、 <i>z</i> の先頭の <i>m</i> 列に、行列 <i>A</i> の選択された固有値に対応する正規直交固有ベクトルが格納される。すなわち、 <i>w</i> ( <i>i</i> ) に対応する固有ベクトルが、 <i>z</i> の <i>i</i> 番目の列に格納される。固有ベクトルが収束できない場合、 <i>z</i> のその列には、その固有ベクトルに対する最新の近似値が格納され、固有ベクトルのインデックスは <i>ifail</i> に返される。 <i>jobz</i> = 'N' の場合、 <i>z</i> は参照されない。 注：配列 <i>z</i> には、 $\max(1, m)$ 以上の列が提供されなければならない。 <i>range</i> = 'V' の場合、 <i>m</i> の正確な値が事前にわからないため、上限値を使用する必要がある。
<i>d</i> , <i>e</i>	終了時に、固有値を計算するときのオーバーフローまたはアンダーフローを避けるために、選択された定係数が掛けられることがある。
<i>ifail</i>	INTEGER。 配列、次元は $\max(1, n)$ 以上。 <i>jobz</i> = 'V' の場合、 <i>info</i> = 0 であれば、 <i>ifail</i> の先頭から <i>m</i> 個の成分はゼロになる。 <i>info</i> > 0 であれば、収束しなかった固有ベクトルのインデックスが <i>ifail</i> に格納される。 <i>jobz</i> = 'N' の場合、 <i>ifail</i> は参照されない。

*info* INTEGER。  
*info* = 0 の場合、実行は正常に終了したことを示す。  
*info* = -*i* の場合、*i* 番目のパラメータの値が不正であったことを示す。  
*info* = *i* の場合、*i* 番目の固有ベクトルが収束に失敗した。その固有ベクトルのインデックスが配列 *ifail* に格納される。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *stevx* のインターフェイスの詳細を以下に示す。

<i>d</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>e</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>w</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>z</i>	サイズ ( <i>n,m</i> ) の行列 <i>Z</i> を格納する。ここで、値 <i>n</i> と <i>m</i> は有意である。
<i>ifail</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>v1</i>	この成分のデフォルト値は、 <i>v1</i> = -HUGE( <i>v1</i> ) である。
<i>vu</i>	この成分のデフォルト値は、 <i>vu</i> = HUGE( <i>v1</i> ) である。
<i>i1</i>	この引数のデフォルト値は、 <i>i1</i> = 1 である。
<i>iu</i>	この引数のデフォルト値は、 <i>iu</i> = <i>n</i> である。
<i>abstol</i>	この成分のデフォルト値は、 <i>abstol</i> = 0.0_WP である。
<i>jobz</i>	引数 <i>z</i> の存在に基づいて以下のように復元される。 <i>jobz</i> = 'V' ( <i>z</i> が存在する場合)、 <i>jobz</i> = 'N' ( <i>z</i> が省略された場合)。 <i>ifail</i> が存在し、 <i>z</i> が省略された場合、エラー条件がセットされる。
<i>range</i>	引数 <i>v1</i> 、 <i>vu</i> 、 <i>i1</i> 、および <i>iu</i> の存在に基づいて以下のように復元される。 <i>range</i> = 'V' ( <i>v1</i> と <i>vu</i> のいずれかまたは両方が存在する場合)、 <i>range</i> = 'I' ( <i>i1</i> と <i>iu</i> のいずれかまたは両方が存在する場合)、 <i>range</i> = 'A' ( <i>v1</i> 、 <i>vu</i> 、 <i>i1</i> 、および <i>iu</i> がすべて存在しない場合)。 <i>v1</i> と <i>vu</i> のいずれかまたは両方が存在し、同時に <i>i1</i> と <i>iu</i> のいずれかまたは両方が存在する場合、エラー条件がセットされる。

### アプリケーション・ノート

$abstol + \epsilon * \max(|a|, |b|)$  以下の幅の区間  $[a, b]$  内に存在していると判別された場合 ( $\epsilon$  はマシン精度)、近似固有値は、収束したものとして受け入れられる。 $abstol$  がゼロまたは負の場合、代わりに  $\epsilon * \|A\|_1$  を使用する。  
固有値が最も正確に計算されるのは、 $abstol$  がゼロでなく、アンダーフローのしきい値の 2 倍、 $2 * \text{?lamch}('S')$  に設定されたときである。このルーチンで  $info > 0$  が返された場合、固有ベクトルのいくつか収束に失敗したことを意味するので、 $abstol$  を  $2 * \text{?lamch}('S')$  に設定して、やり直してみる。

---

## ?stevr

「比較的安定な表現」を使用して、実対称三重対角行列の固有値と (オプションで) 固有ベクトルを選択的に計算する。

---

### 構文

#### Fortran 77:

```
call sstevr(jobz, range, n, d, e, vl, vu, il, iu, abstol, m, w, z, ldz, isuppz,
           work, lwork, iwork, liwork, info)
call dstevr(jobz, range, n, d, e, vl, vu, il, iu, abstol, m, w, z, ldz, isuppz,
           work, lwork, iwork, liwork, info)
```

#### Fortran 95:

```
call stevr(d, e, w [,z] [,vl] [,vu] [,il] [,iu] [,m] [,isuppz] [,abstol]
           [,info])
```

### 説明

このルーチンは、実対称三重対角行列  $T$  の固有値と (オプションで) 固有ベクトルを選択的に計算する。固有値と固有ベクトルを選択するには、希望する固有値の値の範囲かインデックスの範囲を指定する。

可能であれば、?stevr は [sstegr/dstegr](#) を呼び出し、「比較的安定な表現」を使用して、固有スペクトルを計算する。?stegr は、固有値の計算には *dqds* アルゴリズムを使用するが、直交固有ベクトルは、種々の「良好な」 $LDL^T$  表現 (「比較的安定な表現」と

も呼ばれる)から計算する。グラム-シュミットの直交化はできる限り避ける。アルゴリズムの各ステップを詳しく説明すると次のようになる。 $T$  のまだ縮退されていない  $i$  番目のブロックに対して、

- (a)  $L_i D_i L_i^T$  が比較的安定な表現になるように、 $T - s_i = L_i D_i L_i^T$  を計算する。
- (b) *dqds* アルゴリズムによって、 $L_i D_i L_i^T$  の固有値  $\lambda_j$  を高い相対精度で計算する。
- (c) 値が近い固有値からなるクラスタが存在する場合は、そのクラスタに近い  $s_i$  を「選択」し、ステップ (a) に戻る。
- (d)  $L_i D_i L_i^T$  の近似的な固有値  $\lambda_j$  に対する固有ベクトルを、階数表示のツイスト分解によって求める。

出力値に必要な精度は、入力パラメータ *abstol* で指定できる。

ルーチン *?stevr* は、IEEE-754 の浮動小数点標準に準拠しているマシンにおいてフル・スペクトルが要求された場合、[sstegr/dstegr](#) を呼び出す。*?stevr* は、IEEE-754 に準拠しないマシンの場合や、部分スペクトルが要求された場合、[sstebz/dstebz](#) と [sstein/dstein](#) を呼び出す。

## 入力パラメータ

<i>jobz</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobz</i> = 'N' の場合、固有値のみを計算する。 <i>jobz</i> = 'V' の場合、固有値と固有ベクトルを計算する。
<i>range</i>	CHARACTER*1。'A'、'V'、または 'I' のいずれかでなければならない。 <i>range</i> = 'A' の場合、固有値をすべて計算する。 <i>range</i> = 'V' の場合、半開区間 $vl < \lambda_i \leq vu$ における固有値 $\lambda_i$ を計算する。 <i>range</i> = 'I' の場合、ルーチンはインデックスが $il \sim iu$ である固有値を計算する。  <i>range</i> = 'V' または 'I' で、 $iu - il < n - 1$ の場合、 <i>sstebz/dstebz</i> と <i>sstein/dstein</i> を呼び出す。
<i>n</i>	INTEGER。行列 $T$ の次数 ( $n \geq 0$ )。
<i>d, e, work</i>	REAL ( <i>sstevr</i> の場合) DOUBLE PRECISION ( <i>dstevr</i> の場合)。 配列: <i>d</i> (*) には、三重対角行列 $T$ の $n$ 個の対角成分を格納する。 <i>d</i> の次元は、 $\max(1, n)$ 以上でなければならない。

	<p><math>e(*)</math> には、<math>A</math> の <math>n-1</math> 個の劣対角成分を格納する。  <math>e</math> の次元は、<math>\max(1, n)</math> 以上でなければならない。この配列の <math>n</math> 番目の成分は、ワークスペースとして使用される。</p> <p><math>work(lwork)</math> は、ワークスペース配列である。</p>
$vl, vu$	<p>REAL(ssstevr の場合 )          DOUBLE PRECISION(dstevr の場合 )。  <math>range = 'V'</math> の場合、固有値を検索する区間の上限と下限。          次の制約がある。 <math>vl &lt; vu</math>。</p> <p><math>range = 'A'</math> または <math>'I'</math> の場合、<math>vl</math> と <math>vu</math> は参照されない。</p>
$il, iu$	<p>INTEGER。  <math>range = 'I'</math> の場合、返される最小固有値と最大固有値に対する昇順のインデックス。          次の制約がある。  <math>1 \leq il \leq iu \leq n</math> (<math>n &gt; 0</math> の場合 )。  <math>il=1</math> かつ <math>iu=0</math> (<math>n=0</math> の場合 )。</p> <p><math>range = 'A'</math> または <math>'V'</math> の場合、<math>il</math> と <math>iu</math> は参照されない。</p>
$abstol$	<p>REAL(ssyevr の場合 )          DOUBLE PRECISION(dsyevr の場合 )。          各固有値および固有ベクトルに許される絶対誤差許容値。  <math>jobz = 'V'</math> の場合、出力される固有値と固有ベクトルの誤差ノルムが <math>abstol</math> 以内になる。また、異なる固有ベクトル間の内積も <math>abstol</math> 以内になる。<math>abstol &lt; n\epsilon\ T\ _1</math> の場合、<math>n\epsilon\ T\ _1</math> が代わりに使用される (<math>\epsilon</math> はマシン精度)。固有値は、<math>abstol</math> とは無関係に、<math>\epsilon\ T\ _1</math> の精度で計算される。高い相対精度が必要な場合は、<math>abstol</math> を <math>?lamch('S')</math> に設定する。</p>
$ldz$	<p>INTEGER。出力配列 <math>z</math> のリーディング・ディメンジョン。次の制約がある。  <math>ldz \geq 1</math> (<math>jobz = 'N'</math> の場合 )。  <math>ldz \geq \max(1, n)</math> (<math>jobz = 'V'</math> の場合 )。</p>
$lwork$	<p>INTEGER。配列 <math>work</math> の次元。          次の制約がある。 <math>lwork \geq \max(1, 20n)</math>。  <math>lwork = -1</math> の場合、ワークスペースのクエリとみなされ、ルーチンは <math>work</math> 配列の最適サイズだけを計算し、その値を <math>work</math> 配列の最初のエントリとして返す。xerbla は <math>lwork</math> に関するエラー・メッセージを生成しない。</p>



<i>iwork</i>	INTEGER。 ワークスペース配列、次元は ( <i>liwork</i> )。
<i>liwork</i>	INTEGER。配列 <i>iwork</i> の次元。 $liwork \geq \max(1, 10n)$ 。

*liwork* = -1 の場合、ワークスペースのクエリとみなされ、ルーチンは *iwork* 配列の最適サイズだけを計算し、その値を *iwork* 配列の最初のエントリとして返す。xerbla は *liwork* に関するエラー・メッセージを生成しない。

### 出力パラメータ

<i>m</i>	INTEGER。検出された固有値の総数。( $0 \leq m \leq n$ )。 <i>range</i> = 'A' の場合、 $m = n$ 、 <i>range</i> = 'I' の場合、 $m = iu - il + 1$ 。
<i>w</i> , <i>z</i>	REAL (sstevr の場合 ) DOUBLE PRECISION (dstevr の場合 )。 配列： <i>w</i> (*)、次元は $\max(1, n)$ 以上。 <i>w</i> の先頭 <i>m</i> 成分に、行列 <i>T</i> の選択された固有値が昇順に格納される。  <i>z</i> ( <i>ldz</i> ,*)。 <i>z</i> の第 2 次元は、 $\max(1, m)$ 以上でなければならない。 <i>jobz</i> = 'V' の場合、 <i>info</i> = 0 であれば、 <i>z</i> の先頭の <i>m</i> 列に、行列 <i>T</i> の選択された固有値に対応する正規直交固有ベクトルが格納される。すなわち、 <i>w</i> ( <i>i</i> ) に対応する固有ベクトルが、 <i>z</i> の <i>i</i> 番目の列に格納される。 <i>jobz</i> = 'N' の場合、 <i>z</i> は参照されない。 注：配列 <i>z</i> には、 $\max(1, m)$ 以上の列が提供されなければならない。 <i>range</i> = 'V' の場合、 <i>m</i> の正確な値が事前にわからないため、上限値を使用する必要がある。
<i>d</i> , <i>e</i>	終了時に、固有値を計算するときのオーバーフローまたはアンダーフローを避けるために、選択された定係数が掛けられることがある。
<i>isuppz</i>	INTEGER。 配列、次元は $2 * \max(1, m)$ 以上。  <i>z</i> に格納されている固有ベクトルのサポート情報、すなわち <i>z</i> に格納されている非ゼロの成分を示すインデックス。 <i>i</i> 番目の固有ベクトルは、 <i>isuppz</i> ( $2i - 1$ ) から <i>isuppz</i> ( $2i$ ) までの成分のみ非ゼロである。 <i>range</i> = 'A' または <i>range</i> = 'I' で $iu - il = n - 1$ の場合にのみ有効である。
<i>work</i> (1)	終了時に、 <i>info</i> = 0 の場合、 <i>work</i> (1) は <i>lwork</i> の必要最小サイズを返す。

<i>iwork(1)</i>	終了時に、 <i>info</i> = 0 の場合、 <i>iwork(1)</i> は <i>liwork</i> の必要最小サイズを返す。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正であったことを示す。 <i>info</i> = <i>i</i> の場合、内部エラーが発生したことを示す。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *stevr* のインターフェイスの詳細を以下に示す。

<i>d</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>e</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>w</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>z</i>	サイズ ( <i>n,m</i> ) の行列 <i>Z</i> を格納する。ここで、値 <i>n</i> と <i>m</i> は有意である。
<i>isuppz</i>	長さ (2*m) のベクトルを格納する。ここで、値 (2*m) は有意である。
<i>v1</i>	この成分のデフォルト値は <i>v1</i> = -HUGE( <i>v1</i> ) である。
<i>vu</i>	この成分のデフォルト値は <i>vu</i> = HUGE( <i>v1</i> ) である。
<i>i1</i>	この引数のデフォルト値は、 <i>i1</i> = 1 である。
<i>iu</i>	この引数のデフォルト値は、 <i>iu</i> = <i>n</i> である。
<i>abstol</i>	この成分のデフォルト値は <i>abstol</i> = 0.0_WP である。
<i>jobz</i>	引数 <i>z</i> の存在に基づいて以下のように復元される。 <i>jobz</i> = 'V' ( <i>z</i> が存在する場合)、 <i>jobz</i> = 'N' ( <i>z</i> が省略された場合)。 <i>ifail</i> が存在し、 <i>z</i> が省略された場合、エラー条件がセットされる。
<i>range</i>	引数 <i>v1</i> 、 <i>vu</i> 、 <i>i1</i> 、および <i>iu</i> の存在に基づいて以下のように復元される。 <i>range</i> = 'V' ( <i>v1</i> と <i>vu</i> のいずれかまたは両方が存在する場合)、 <i>range</i> = 'I' ( <i>i1</i> と <i>iu</i> のいずれかまたは両方が存在する場合)、

`range = 'A'` (`v1`、`vu`、`i1`、および `iu` がすべて存在しない場合。  
`v1` と `vu` のいずれかまたは両方が存在し、同時に `i1` と `iu` のいずれか  
または両方が存在する場合、エラー条件がセットされる。

## アプリケーション・ノート

通常の `?stegr` の実行で NaN と無限大が発生するため、NaN と無限大が IEEE 標準のデフォルト方法以外で処理される環境では、浮動小数点例外によって異常終了する可能性がある。

## 非対称固有値問題

このセクションでは、非対称固有値問題を解くために使用する LAPACK ドライバルーチンについて説明する。これらの問題を解くために呼び出すことが可能な[計算ルーチン](#)も参照のこと。

[表 4-11](#) に Fortran-77 インターフェイスのすべてのドライバルーチンを示す。Fortran-95 インターフェイスのルーチン名はそれぞれ、先頭の記号がない ([ルーチン命名規則](#)を参照)。

**表 4-11**      非対称固有値問題を解くためのドライバルーチン

ルーチン名	機能
<a href="#">?gees</a>	一般行列の固有値と Schur 因子分解を計算し、指定された固有値が Schur 形式の左上にくるように因子分解を順序付けする。
<a href="#">?geesx</a>	一般行列の固有値と Schur 因子分解を計算し、因子分解の順序付けと、条件数の逆数の計算を実行する。
<a href="#">?geev</a>	一般行列の固有値と左 / 右の固有ベクトルを計算する。
<a href="#">?geevx</a>	あらかじめ行列を平衡化して、一般行列の固有値と左 / 右の固有ベクトル、固有値と右固有ベクトルに対する条件数の逆数を計算する。

### ?gees

一般行列の固有値と *Schur* 因子分解を計算し、指定された固有値が *Schur* 形式の左上にくるように因子分解を順序付けする。

---

#### 構文

##### Fortran 77:

```
call sgees(jobvs, sort, select, n, a, lda, sdim, wr, wi, vs, ldvs, work, lwork,  
          bwork, info)  
call dgees(jobvs, sort, select, n, a, lda, sdim, wr, wi, vs, ldvs, work, lwork,  
          bwork, info)  
call cgees(jobvs, sort, select, n, a, lda, sdim, w, vs, ldvs, work, lwork,  
          rwork, bwork, info)  
call zgees(jobvs, sort, select, n, a, lda, sdim, w, vs, ldvs, work, lwork,  
          rwork, bwork, info)
```

##### Fortran 95:

```
call gees(a, wr, wi [,vs] [,select] [,sdim] [,info])  
call gees(a, w [,vs] [,select] [,sdim] [,info])
```

#### 説明

このルーチンは、 $n \times n$  の実 / 複素非対称行列  $A$  について、固有値、実数 *Schur* 形式  $T$  と (オプションで) *Schur* ベクトル  $Z$  の行列を計算する。これにより、*Schur* 因子分解  $A = ZTZ^H$  が提供される。

また、このルーチンは、指定された固有値が左上にくるように、実数 *Schur*/*Schur* 形式の対角上の固有値を順序付けることもできる。 $Z$  の先頭の列は、指定された固有値に対応する不変部分空間の直交基を形成する。

$1 \times 1$  ブロックと  $2 \times 2$  ブロックを持つ上準三角行列の場合、実数行列は、実数 *Schur* 形式となる。 $2 \times 2$  ブロックは、次の形式で標準化される。

$$\begin{pmatrix} a & b \\ c & a \end{pmatrix}$$

ここで、 $b \cdot c < 0$ 。このようなブロックの固有値は、 $a \pm \sqrt{bc}$  となる。

複素行列は、上三角の場合は **Schur** 形式になる。

### 入力パラメータ

<i>jobvs</i>	CHARACTER*1. 'N' または 'V' でなければならない。 <i>jobvs</i> ='N' の場合、 <b>Schur</b> ベクトルは計算されない。 <i>jobvs</i> ='V' の場合、 <b>Schur</b> ベクトルは計算される。
<i>sort</i>	CHARACTER*1. 'N' または 'S' でなければならない。 <b>Schur</b> 形式の対角上の固有値を順序付けるかどうかを指定する。  <i>sort</i> ='N' の場合、固有値は順序付けされない。 <i>sort</i> ='S' の場合、固有値は順序付けされる ( <i>select</i> を参照)。
<i>select</i>	実数型の場合、2 つの REAL 引数の LOGICAL FUNCTION。 複素数型の場合、1 つの COMPLEX 引数の LOGICAL FUNCTION。  <i>select</i> は、呼び出しサブルーチン内で EXTERNAL として宣言する必要がある。  <i>sort</i> ='S' の場合、 <i>select</i> は、 <b>Schur</b> 形式の左上に対してソートを実行する固有値を選択するために使用される。 <i>sort</i> ='N' の場合、 <i>select</i> は参照されない。 実数型の場合： <i>select</i> ( <i>wr</i> ( <i>j</i> ), <i>wi</i> ( <i>j</i> )) が真の場合、固有値 $wr(j) + \sqrt{-1} * wi(j)$ が選択される。つまり、固有値の複素共役ペアの一方を選択すると、複素数の固有値が両方選択される。そのため、順序付けを実行した後、選択した複素数の固有値は、 <i>select</i> ( <i>wr</i> ( <i>j</i> ), <i>wi</i> ( <i>j</i> ))=.TRUE. を満たさないときがある。この場合、 <i>info</i> には、 <i>n</i> +2 が設定される (以下の <i>info</i> を参照)。 複素数型の場合： <i>select</i> ( <i>w</i> ( <i>j</i> )) が真の場合、固有値 <i>w</i> ( <i>j</i> ) が選択される。
<i>n</i>	INTEGER。行列 <i>A</i> の次数 ( <i>n</i> ≥ 0)。
<i>a</i> , <i>work</i>	REAL ( <i>sgees</i> の場合) DOUBLE PRECISION ( <i>dgees</i> の場合) COMPLEX ( <i>cgees</i> の場合) DOUBLE COMPLEX ( <i>zgees</i> の場合)。 配列： <i>a</i> ( <i>lda</i> ,*) は、 <i>n</i> × <i>n</i> の行列 <i>A</i> を格納する配列である。 <i>a</i> の第 2 次元は、max(1, <i>n</i> ) 以上でなければならない。  <i>work</i> ( <i>lwork</i> ) は、ワークスペース配列である。
<i>lda</i>	INTEGER。配列 <i>a</i> の第 1 次元。 max(1, <i>n</i> ) 以上でなければならない。

<i>ldvs</i>	INTEGER。出力配列 <i>vs</i> のリーディング・ディメンジョン。 次の制約がある。 $ldvs \geq 1$ 。 $ldvs \geq \max(1, n)$ ( <i>jobvs</i> = 'V' の場合)。
<i>lwork</i>	INTEGER。配列 <i>work</i> の次元。 次の制約がある。 $lwork \geq \max(1, 3n)$ (実数型の場合)。 $lwork \geq \max(1, 2n)$ (複素数型の場合)。 <i>lwork</i> = -1 の場合、ワークスペースのクエリとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリとして返す。 <i>xerbla</i> は <i>lwork</i> に関するエラー・メッセージを生成しない。
<i>rwork</i>	REAL ( <i>cgees</i> の場合) DOUBLE PRECISION ( <i>zgees</i> の場合) ワークスペース配列、次元は $\max(1, n)$ 以上。複素数型でのみ使用される。
<i>bwork</i>	LOGICAL。 ワークスペース配列、次元は $\max(1, n)$ 以上。 <i>sort</i> = 'N' の場合は参照されない。

## 出力パラメータ

<i>a</i>	終了時に、実数 Schur/Schur 形式 <i>T</i> で上書きされる。
<i>sdim</i>	INTEGER。 <i>sort</i> = 'N' の場合、 <i>sdim</i> = 0 である。 <i>sort</i> = 'S' の場合、 <i>sdim</i> は、 <i>select</i> が真の (ソート後の) 固有値の数に等しくなる。 実数型の場合、どちらかの固有値について <i>select</i> が真の複素共役ペアは、2 としてカウントされるのに注意する必要がある。
<i>wr, wi</i>	REAL ( <i>sgees</i> の場合) DOUBLE PRECISION ( <i>dgees</i> の場合) 配列、次元はそれぞれ $\max(1, n)$ 以上。 出力実数 Schur 形式 <i>T</i> の対角上に格納されるのと同じ順序で、計算された固有値の実数部と虚数部をそれぞれ格納する。固有値の複素共役ペアは、虚数部が正である固有値に続いて格納される。

<i>w</i>	<p>COMPLEX (cgees の場合 )  DOUBLE COMPLEX (zgees の場合 )。  配列、次元は <math>\max(1, n)</math> 以上。  計算された固有値が格納される。一連の固有値は、出力 Schur 形式 <i>T</i> の対角上に格納されるのと同じ順序で格納される。</p>
<i>vs</i>	<p>REAL (sgees の場合 )  DOUBLE PRECISION (dgees の場合 )  COMPLEX (cgees の場合 )  DOUBLE COMPLEX (zgees の場合 )。  配列 <i>vs</i>(<i>ldvs</i>,*)。 <i>vs</i> の第 2 次元は、<math>\max(1, n)</math> 以上でなければならない。    <i>jobvs</i> = 'V' の場合、 <i>vs</i> には、Schur ベクトルの直交 / ユニタリ行列 <i>Z</i> が格納される。  <i>jobvs</i> = 'N' の場合、 <i>vs</i> は参照されない。</p>
<i>work</i> (1)	<p>終了時に、 <i>info</i> = 0 の場合、 <i>work</i>(1) は <i>lwork</i> の必要最小サイズを返す。</p>
<i>info</i>	<p>INTEGER。  <i>info</i> = 0 の場合、実行は正常に終了したことを示す。    <i>info</i> = -<i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正であったことを示す。    <i>info</i> = <i>i</i>、かつ  <math>i \leq n</math> :    <b>QR</b> アルゴリズムが固有値をすべて計算できなかった。収束した固有値は、<i>wr</i> と <i>wi</i> ( 実数型の場合 ) または <i>w</i> ( 複素数型の場合 ) の 1:<i>i</i><i>lo</i>-1 と <i>i</i>+1:<i>n</i> の成分に格納される。 <i>jobvs</i> = 'V' の場合、 <i>vs</i> には、<i>A</i> をその部分収束 Schur 形式に縮退させる行列を格納する。    <i>i</i> = <i>n</i>+1 :    いくつかの固有値が接近しすぎて分離できなかったため、固有値を順序変更できなかった ( 非常に悪い条件が問題 )。    <i>i</i> = <i>n</i>+2 :    順序変更後、丸め操作によりいくつかの複素数の固有値の値が変更されたため、Schur 形式の先頭の固有値が <i>select</i> = .TRUE. を満たさなくなった。これは、スケーリングによるアンダーフローによっても引き起こされる。</p>

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `gees` のインターフェイスの詳細を以下に示す。

<code>a</code>	サイズ $(n,n)$ の行列 <code>A</code> を格納する。
<code>wr</code>	長さ $(n)$ のベクトルを格納する。実数型でのみ使用される。
<code>wi</code>	長さ $(n)$ のベクトルを格納する。実数型でのみ使用される。
<code>w</code>	長さ $(n)$ のベクトルを格納する。複素数型でのみ使用される。
<code>vs</code>	サイズ $(n,n)$ の行列 <code>VS</code> を格納する。
<code>jobvs</code>	引数 <code>vs</code> の存在に基づいて以下のように復元される。 <code>jobvs = 'V'</code> ( <code>vs</code> が存在する場合)、 <code>jobvs = 'N'</code> ( <code>vs</code> が省略された場合)。
<code>sort</code>	引数 <code>select</code> の存在に基づいて以下のように復元される。 <code>sort = 'S'</code> ( <code>select</code> が存在する場合)、 <code>sort = 'N'</code> ( <code>select</code> が省略された場合)。

### アプリケーション・ノート

配列 `work` に必要なワークスペースの大きさがわからない場合は、最初の実行で `lwork` を大きめの値に設定する。終了時に、`work(1)` の値を調べ、以降はその値を使用する。

---

## ?geesx

一般行列の固有値と *Schur* 因子分解を計算し、  
因子分解の順序付けと、条件数の逆数の計算を  
実行する。

---

### 構文

#### Fortran 77:

```
call sgeesx(jobvs, sort, select, sense, n, a, lda, sdim, wr, wi, vs, ldvs,  
            rconde, rcondv, work, lwork, iwork, liwork, bwork, info)
```



```
call dgeesx(jobvs, sort, select, sense, n, a, lda, sdim, wr, wi, vs, ldvs,
           rconde, rcondv, work, lwork, iwork, liwork, bwork, info)
call cgeesx(jobvs, sort, select, sense, n, a, lda, sdim, w, vs, ldvs, rconde,
           rcondv, work, lwork, rwork, bwork, info)
call zgeesx(jobvs, sort, select, sense, n, a, lda, sdim, w, vs, ldvs, rconde,
           rcondv, work, lwork, rwork, bwork, info)
```

**Fortran 95:**

```
call geesx(a, wr, wi [,vs] [,select] [,sdim] [,rconde] [,rcondv] [,info])
call geesx(a, w [,vs] [,select] [,sdim] [,rconde] [,rcondv] [,info])
```

**説明**

このルーチンは、 $n \times n$  の実 / 複素非対称行列  $A$  について、固有値、実数 Schur/Schur 形式  $T$  と (オプションで) Schur ベクトル  $Z$  の行列を計算する。これにより、Schur 因子分解  $A = ZTZ^H$  が提供される。

このルーチンは、指定された固有値が左上にくるように、実数 Schur/Schur 形式の対角上の固有値を順序付けもできる。また、指定した固有値の平均に対する条件数の逆数 ( $rconde$ ) を計算したり、指定した固有値に対応する右不変部分空間に対する条件数の逆数 ( $rcondv$ ) も計算できる。 $Z$  の先頭の列は、この不変部分空間の直交基を形成する。

条件数の逆数  $rconde$ 、 $rcondv$  の詳細は、4.10 の [\[LUG\]](#) を参照のこと (これらの大きさはそれぞれ、 $s$  と  $sep$  と呼ぶ)。

$1 \times 1$  ブロックと  $2 \times 2$  ブロックを持つ上準三角行列の場合、実数行列は、実数 Schur 形式となる。 $2 \times 2$  ブロックは、次の形式で標準化される。

$$\begin{pmatrix} a & b \\ c & a \end{pmatrix}$$

ここで、 $b^*c < 0$ 。このようなブロックの固有値は、 $a \pm \sqrt{bc}$  となる。

複素行列は、上三角の場合は Schur 形式になる。

**入力パラメータ**

<i>jobvs</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobvs</i> ='N' の場合、Schur ベクトルは計算されない。 <i>jobvs</i> ='V' の場合、Schur ベクトルは計算される。
<i>sort</i>	CHARACTER*1。'N' または 'S' でなければならない。 Schur 形式の対角上の固有値を順序付けるかどうかを指定する。

	<p><code>sort = 'N'</code> の場合、固有値は順序付けされない。</p> <p><code>sort = 'S'</code> の場合、固有値は順序付けされる (<code>select</code> を参照)。</p>
<code>select</code>	<p>実数型の場合、2 つの REAL 引数の LOGICAL FUNCTION。</p> <p>複素数型の場合、1 つの COMPLEX 引数の LOGICAL FUNCTION。</p> <p><code>select</code> は、呼び出しサブルーチン内で EXTERNAL として宣言する必要がある。</p> <p><code>sort = 'S'</code> の場合、<code>select</code> は、Schur 形式の左上に対してソートを実行する固有値を選択するために使用される。</p> <p><code>sort = 'N'</code> の場合、<code>select</code> は参照されない。</p> <p>実数型の場合:</p> <p><code>select(wr(j), wi(j))</code> が真の場合、固有値 <math>wr(j) + \sqrt{-1} * wi(j)</math> が選択される。つまり、固有値の複素共役ペアの一方を選択すると、複素数の固有値が両方選択される。そのため、順序付けを実行した後、選択した複素数の固有値は、<code>select(wr(j), wi(j)) = .TRUE.</code> を満たさないときがある。この場合、<code>info</code> には、<code>n+2</code> が設定される (以下の <code>info</code> を参照)。</p> <p>複素数型の場合:</p> <p><code>select(w(j))</code> が真の場合、固有値 <code>w(j)</code> が選択される。</p>
<code>sense</code>	<p>CHARACTER*1。'N'、'E'、'V'、または 'B' でなければならない。</p> <p>どの条件数の逆数を計算するのかを決定する。</p> <p><code>sense = 'N'</code> の場合、計算は実行されない。</p> <p><code>sense = 'E'</code> の場合、指定された固有値の平均についてのみ計算が実行される。</p> <p><code>sense = 'V'</code> の場合、指定された右不変部分空間についてのみ計算が実行される。</p> <p><code>sense = 'B'</code> の場合、両方について計算が実行される。</p> <p><code>sense</code> が 'E'、'V'、または 'B' の場合、<code>sort</code> は 'S' に等しくなければならない。</p>
<code>n</code>	INTEGER。行列 <code>A</code> の次数 ( $n \geq 0$ )。
<code>a, work</code>	<p>REAL (<code>sgeesx</code> の場合)</p> <p>DOUBLE PRECISION (<code>dgeesx</code> の場合)</p> <p>COMPLEX (<code>cgeesx</code> の場合)</p> <p>DOUBLE COMPLEX (<code>zgeesx</code> の場合)。</p> <p>配列:</p> <p><code>a(lda,*)</code> は、<math>n \times n</math> の行列 <code>A</code> を格納する配列である。</p> <p><code>a</code> の第 2 次元は、<math>\max(1, n)</math> 以上でなければならない。</p> <p><code>work(lwork)</code> は、ワークスペース配列である。</p>

<i>lda</i>	INTEGER。配列 <i>a</i> の第 1 次元。 $\max(1, n)$ 以上でなければならない。
<i>ldvs</i>	INTEGER。出力配列 <i>vs</i> のリーディング・ディメンジョン。 次の制約がある。 $ldvs \geq 1$ 。 $ldvs \geq \max(1, n)$ ( <i>jobvs</i> = 'V' の場合)。
<i>lwork</i>	INTEGER。配列 <i>work</i> の次元。 次の制約がある。 $lwork \geq \max(1, 3n)$ (実数型の場合)。 $lwork \geq \max(1, 2n)$ (複素数型の場合)。  また、 <i>sense</i> = 'E'、'V'、または 'B' の場合、次のようになる。 $lwork \geq n+2*sdim*(n-sdim)$ (実数型の場合)。 $lwork \geq 2*sdim*(n-sdim)$ (複素数型の場合)。 ここで、 <i>sdim</i> は、このルーチンで計算される指定された固有値の数である。また、 $2*sdim*(n-sdim) \leq n*n/2$ である。  パフォーマンスを改善するには、一般に <i>lwork</i> に上記以上の値が必要である。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は ( <i>liwork</i> )。実数型でのみ使用される。 <i>sense</i> = 'N' または 'E' の場合は参照されない。
<i>liwork</i>	INTEGER。配列 <i>iwork</i> の次元。実数型でのみ使用される。 次の制約がある。 $liwork \geq 1$ 。 <i>sense</i> = 'V' または 'B' の場合、 $liwork \geq sdim*(n-sdim)$ 。
<i>rwork</i>	REAL ( <i>cgeesx</i> の場合) DOUBLE PRECISION ( <i>zgeesx</i> の場合) ワークスペース配列、次元は $\max(1, n)$ 以上。複素数型でのみ使用される。
<i>bwork</i>	LOGICAL。 ワークスペース配列、次元は $\max(1, n)$ 以上。 <i>sort</i> = 'N' の場合は参照されない。

## 出力パラメータ

<i>a</i>	終了時に、実数 Schur/Schur 形式 <i>T</i> で上書きされる。
----------	--

<i>sdim</i>	<p>INTEGER。</p> <p><i>sort</i>='N' の場合、<i>sdim</i>=0 である。</p> <p><i>sort</i>='S' の場合、<i>sdim</i> は、<i>select</i> が真の ( ソート後の ) 固有値の数に等しくなる。</p> <p>実数型の場合、どちらかの固有値について <i>select</i> が真の複素共役ペアは、2 としてカウントされるのに注意する必要がある。</p>
<i>wr, wi</i>	<p>REAL (<i>sgeesx</i> の場合 )</p> <p>DOUBLE PRECISION (<i>dgeesx</i> の場合 )</p> <p>配列、次元はそれぞれ <math>\max(1, n)</math> 以上。</p> <p>出力実数 Schur 形式 <i>T</i> の対角上に格納されるのと同じ順序で、計算された固有値の実数部と虚数部をそれぞれ格納する。固有値の複素共役ペアは、虚数部が正である固有値に続いて格納される。</p>
<i>w</i>	<p>COMPLEX (<i>cgeesx</i> の場合 )</p> <p>DOUBLE COMPLEX (<i>zgeesx</i> の場合 )。</p> <p>配列、次元は <math>\max(1, n)</math> 以上。</p> <p>計算された固有値が格納される。一連の固有値は、出力 Schur 形式 <i>T</i> の対角上に格納されるのと同じ順序で格納される。</p>
<i>vs</i>	<p>REAL (<i>sgeesx</i> の場合 )</p> <p>DOUBLE PRECISION (<i>dgeesx</i> の場合 )</p> <p>COMPLEX (<i>cgeesx</i> の場合 )</p> <p>DOUBLE COMPLEX (<i>zgeesx</i> の場合 )。</p> <p>配列 <i>vs</i>(<i>ldvs</i>,*)。 <i>vs</i> の第 2 次元は、<math>\max(1, n)</math> 以上でなければならない。</p> <p><i>jobvs</i>='V' の場合、<i>vs</i> には、Schur ベクトルの直交 / ユニタリ行列 <i>Z</i> が格納される。</p> <p><i>jobvs</i>='N' の場合、<i>vs</i> は参照されない。</p>
<i>rconde, rcondv</i>	<p>REAL ( 単精度の場合 )</p> <p>DOUBLE PRECISION ( 倍精度の場合 )。</p> <p><i>sense</i>='E' または 'B' の場合、<i>rconde</i> には、指定した固有値の平均に対する条件数の逆数が格納される。<i>sense</i>='N' または 'V' の場合、<i>rconde</i> は参照されない。</p> <p><i>sense</i>='V' または 'B' の場合、<i>rcondv</i> には、指定した右不変部分空間に対する条件数の逆数が格納される。<i>sense</i>='N' または 'E' の場合、<i>rcondv</i> は参照されない。</p>
<i>work(1)</i>	<p>終了時に、<i>info</i>=0 の場合、<i>work(1)</i> は <i>lwork</i> の必要最小サイズを返す。</p>

*info* INTEGER。  
*info* = 0 の場合、実行は正常に終了したことを示す。  
*info* = -*i* の場合、*i* 番目のパラメータの値が不正であったことを示す。  
*info* = *i*、かつ  
*i* ≤ *n* :  
 QR アルゴリズムが固有値をすべて計算できなかった。収束した固有値は、*wr* と *wi* (実数型の場合) または *w* (複素数型の場合) の 1:*ilo*-1 と *i*+1:*n* の成分に格納される。*jobvs* = 'V' の場合、*vs* には、*A* をその部分収束 Schur 形式に縮退させる変換を格納する。  
*i* = *n*+1 :  
 いくつかの固有値が接近しすぎて分離できなかったため、固有値を順序変更できなかった (非常に悪い条件が問題)。  
*i* = *n*+2 :  
 順序変更後、丸め操作によりいくつかの複素数の固有値の値が変更されたため、Schur 形式の先頭の固有値が *select* = .TRUE. を満たさなくなった。これは、スケーリングによるアンダーフローによっても引き起こされる。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *geesx* のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ ( <i>n</i> , <i>n</i> ) の行列 <i>A</i> を格納する。
<i>wr</i>	長さ ( <i>n</i> ) のベクトルを格納する。実数型でのみ使用される。
<i>wi</i>	長さ ( <i>n</i> ) のベクトルを格納する。実数型でのみ使用される。
<i>w</i>	長さ ( <i>n</i> ) のベクトルを格納する。複素数型でのみ使用される。
<i>vs</i>	サイズ ( <i>n</i> , <i>n</i> ) の行列 <i>VS</i> を格納する。
<i>jobvs</i>	引数 <i>vs</i> の存在に基づいて以下のように復元される。 <i>jobvs</i> = 'V' ( <i>vs</i> が存在する場合)、 <i>jobvs</i> = 'N' ( <i>vs</i> が省略された場合)。

<i>sort</i>	引数 <i>select</i> の存在に基づいて以下のように復元される。 <i>sort</i> = 'S' ( <i>select</i> が存在する場合)、 <i>sort</i> = 'N' ( <i>select</i> が省略された場合)。
<i>sense</i>	引数 <i>rconde</i> と <i>rcondv</i> の存在に基づいて以下のように復元される。 <i>sense</i> = 'B' ( <i>rconde</i> と <i>rcondv</i> の両方が存在する場合)、 <i>sense</i> = 'E' ( <i>rconde</i> が存在し、 <i>rcondv</i> が省略された場合)、 <i>sense</i> = 'V' ( <i>rconde</i> が省略され、 <i>rcondv</i> が存在する場合)、 <i>sense</i> = 'N' ( <i>rconde</i> と <i>rcondv</i> の両方が省略された場合)。

### アプリケーション・ノート

配列 *work* に必要なワークスペースの大きさがわからない場合は、最初の実行で *lwork* を大きめの値に設定する。終了時に、*work(1)* の値を調べ、以降はその値を使用する。

---

## ?geev

一般行列の固有値と ( オプションで )  
左 / 右の固有ベクトルを計算する。

---

### 構文

#### Fortran 77:

```
call sgeev(jobvl, jobvr, n, a, lda, wr, wi, vl, ldvl, vr, ldvr, work, lwork,  
           info)  
call dgeev(jobvl, jobvr, n, a, lda, wr, wi, vl, ldvl, vr, ldvr, work, lwork,  
           info)  
call cgeev(jobvl, jobvr, n, a, lda, w, vl, ldvl, vr, ldvr, work, lwork, rwork,  
           info)  
call zgeev(jobvl, jobvr, n, a, lda, w, vl, ldvl, vr, ldvr, work, lwork, rwork,  
           info)
```

#### Fortran 95:

```
call geev(a, wr, wi [,vl] [,vr] [,info])  
call geev(a, w [,vl] [,vr] [,info])
```

**説明**

このルーチンは、 $n \times n$  の実 / 複素非対称行列  $A$  について、固有値と（オプションで）左 / 右の固有ベクトルを計算する。 $A$  の右固有ベクトル  $v(j)$  は、以下を満たす。

$$A * v(j) = \lambda(j) * v(j)$$

$\lambda(j)$  は、その固有値である。

$A$  の左固有ベクトル  $u(j)$  は、以下を満たす。

$$u(j)^H * A = \lambda(j) * u(j)^H$$

$u(j)^H$  は、 $u(j)$  の共役転置を示す。

計算された固有ベクトルは、1 に等しいユークリッド・ノルムと最大コンポーネント実数を持つように正規化される。

**入力パラメータ**

<i>jobvl</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobvl</i> ='N' の場合、 $A$ の左固有ベクトルは計算されない。 <i>jobvl</i> ='V' の場合、 $A$ の左固有ベクトルは計算される。
<i>jobvr</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobvr</i> ='N' の場合、 $A$ の右固有ベクトルは計算されない。 <i>jobvr</i> ='V' の場合、 $A$ の右固有ベクトルは計算される。
<i>n</i>	INTEGER。行列 $A$ の次数 ( $n \geq 0$ )。
<i>a, work</i>	REAL (sggeev の場合) DOUBLE PRECISION (dgeev の場合) COMPLEX (cggeev の場合) DOUBLE COMPLEX (zggeev の場合)。 配列： $a(lda, *)$ は、 $n \times n$ の行列 $A$ を格納する配列である。 $a$ の第 2 次元は、 $\max(1, n)$ 以上でなければならない。  $work(lwork)$ は、ワークスペース配列である。
<i>lda</i>	INTEGER。配列 $a$ の第 1 次元。 $\max(1, n)$ 以上でなければならない。

<i>ldvl, ldvr</i>	<p>INTEGER。それぞれ、出力配列 <i>v1</i> と <i>vr</i> のリーディング・ディメンジョン。次の制約がある。</p> <p><math>ldvl \geq 1</math>、<math>ldvr \geq 1</math>。</p> <p><i>jobvl</i>='V' の場合、<math>ldvl \geq \max(1, n)</math>。</p> <p><i>jobvr</i>='V' の場合、<math>ldvr \geq \max(1, n)</math>。</p>
<i>lwork</i>	<p>INTEGER。配列 <i>work</i> の次元。</p> <p>次の制約がある。</p> <p><math>lwork \geq \max(1, 3n)</math>。</p> <p><i>jobvl</i>='V' または <i>jobvr</i>='V' の場合、<math>lwork \geq \max(1, 4n)</math> (実数型の場合)。</p> <p><math>lwork \geq \max(1, 2n)</math> (複素数型の場合)。</p> <p>パフォーマンスを改善するには、一般に <i>lwork</i> に上記以上の値が必要である。</p> <p><i>lwork</i> = -1 の場合、ワークスペースのクエリとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリとして返す。xerbla は <i>lwork</i> に関するエラー・メッセージを生成しない。</p>
<i>rwork</i>	<p>REAL (cggeev の場合)</p> <p>DOUBLE PRECISION (zggeev の場合)</p> <p>ワークスペース配列、次元は <math>\max(1, 2n)</math> 以上。複素数型でのみ使用される。</p>

## 出力パラメータ

<i>a</i>	終了時に、中間結果で上書きされる。
<i>wr, wi</i>	<p>REAL (sggeev の場合)</p> <p>DOUBLE PRECISION (dsggeev の場合)</p> <p>配列、次元はそれぞれ <math>\max(1, n)</math> 以上。</p> <p>計算された固有値の実数部と虚数部がそれぞれ格納される。固有値の複素共役ペアは、虚数部が正である固有値に続いて格納される。</p>
<i>w</i>	<p>COMPLEX (cggeev の場合)</p> <p>DOUBLE COMPLEX (zggeev の場合)。</p> <p>配列、次元は <math>\max(1, n)</math> 以上。</p> <p>計算された固有値が格納される。</p>
<i>v1, vr</i>	<p>REAL (sggeev の場合)</p> <p>DOUBLE PRECISION (dsggeev の場合)</p> <p>COMPLEX (cggeev の場合)</p>



DOUBLE COMPLEX (zggev の場合)。

配列：

$v1(ldv1,*)$ 。 $v1$  の第 2 次元は、 $\max(1, n)$  以上でなければならない。

$jobv1='V'$  の場合、左固有ベクトル  $u(j)$  は、それらの固有値と同じ順序で、 $v1$  の列に次々と格納される。 $jobv1='N'$  の場合、 $v1$  は参照されない。

実数型の場合：

$j$  番目の固有値が実数の場合、 $u(j) = v1(:, j)$  ( $v1$  の  $j$  番目の列) である。

$j$  番目と  $(j+1)$  番目の固有値が複素共役ペアとなる場合、

$u(j) = v1(:, j) + i * v1(:, j+1)$ 、 $u(j+1) = v1(:, j) - i * v1(:, j+1)$  である。

ここで、 $i = \sqrt{-1}$  である。

複素数型の場合：

$u(j) = v1(:, j)$  ( $v1$  の  $j$  番目の列) である。

$vr(ldvr,*)$ 。 $vr$  の第 2 次元は、 $\max(1, n)$  以上でなければならない。

$jobvr='V'$  の場合、右固有ベクトル  $v(j)$  は、それらの固有値と同じ順序で、 $vr$  の列に次々と格納される。 $jobvr='N'$  の場合、 $vr$  は参照されない。

実数型の場合：

$j$  番目の固有値が実数の場合、 $v(j) = vr(:, j)$  ( $vr$  の  $j$  番目の列) である。

$j$  番目と  $(j+1)$  番目の固有値が複素共役ペアとなる場合、

$v(j) = vr(:, j) + i * vr(:, j+1)$ 、 $v(j+1) = vr(:, j) - i * vr(:, j+1)$  である。

ここで、 $i = \sqrt{-1}$  である。

複素数型の場合：

$v(j) = vr(:, j)$  ( $vr$  の  $j$  番目の列) である。

$work(1)$  終了時に、 $info = 0$  の場合、 $work(1)$  は  $lwork$  の必要最小サイズを返す。

$info$

INTEGER。

$info = 0$  の場合、実行は正常に終了したことを示す。

$info = -i$  の場合、 $i$  番目のパラメータの値が不正であったことを示す。

$info = i$  の場合、**QR** アルゴリズムが固有値をすべて計算できず、固有ベクトルが計算されなかったことを示す。収束した固有値は、 $wr$  と  $wi$  (実数型の場合) または  $w$  (複素数型の場合) の  $i+1:n$  の成分に格納される。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `geev` のインターフェイスの詳細を以下に示す。

<code>a</code>	サイズ $(n,n)$ の行列 $A$ を格納する。
<code>wr</code>	長さ $(n)$ のベクトルを格納する。実数型でのみ使用される。
<code>wi</code>	長さ $(n)$ のベクトルを格納する。実数型でのみ使用される。
<code>w</code>	長さ $(n)$ のベクトルを格納する。複素数型でのみ使用される。
<code>vl</code>	サイズ $(n,n)$ の行列 $VL$ を格納する。
<code>vr</code>	サイズ $(n,n)$ の行列 $VR$ を格納する。
<code>jobvl</code>	引数 <code>vl</code> の存在に基づいて以下のように復元される。 <code>jobvl = 'V'</code> ( <code>vl</code> が存在する場合)、 <code>jobvl = 'N'</code> ( <code>vl</code> が省略された場合)。
<code>jobvr</code>	引数 <code>vr</code> の存在に基づいて以下のように復元される。 <code>jobvr = 'V'</code> ( <code>vr</code> が存在する場合)、 <code>jobvr = 'N'</code> ( <code>vr</code> が省略された場合)。

### アプリケーション・ノート

配列 `work` に必要なワークスペースの大きさがわからない場合は、最初の実行で `lwork` を大きめの値に設定する。終了時に、`work(1)` の値を調べ、以降はその値を使用する。

## ?geevx

あらかじめ行列を平衡化して、一般行列の固有値と左/右の固有ベクトル、固有値と右固有ベクトルに対する条件数の逆数を計算する。

### 構文

#### Fortran 77:

```
call sgeevx(balanc, jobvl, jobvr, sense, n, a, lda, wr, wi, vl, ldvl, vr, ldvr,
  ilo, ihi, scale, abnrm, rconde, rcondv, work, lwork, iwork, info)
call dgeevx(balanc, jobvl, jobvr, sense, n, a, lda, wr, wi, vl, ldvl, vr, ldvr,
  ilo, ihi, scale, abnrm, rconde, rcondv, work, lwork, iwork, info)
call cgeevx(balanc, jobvl, jobvr, sense, n, a, lda, w, vl, ldvl, vr, ldvr, ilo,
  ihi, scale, abnrm, rconde, rcondv, work, lwork, rwork, info)
call zgeevx(balanc, jobvl, jobvr, sense, n, a, lda, w, vl, ldvl, vr, ldvr, ilo,
  ihi, scale, abnrm, rconde, rcondv, work, lwork, rwork, info)
```

#### Fortran 95:

```
call geevx(a, wr, wi [,vl] [,vr] [,balanc] [,ilo] [,ihi] [,scale] [,abnrm]
  [,rconde] [,rcondv] [,info])
call geevx(a, w [,vl] [,vr] [,balanc] [,ilo] [,ihi] [,scale] [,abnrm] [,rconde]
  [,rcondv] [,info])
```

### 説明

このルーチンは、 $n \times n$  の実 / 複素非対称行列  $A$  について、固有値と（オプションで）左 / 右の固有ベクトルを計算する。

また、このルーチンは、平衡化変換を計算して、固有値と固有ベクトル (*ilo*、*ihi*、*scale*、*abnrm*)、固有値に対する条件数の逆数 (*rconde*)、右固有ベクトルに対する条件数の逆数 (*rcondv*) の条件付けも改善できる。

$A$  の右固有ベクトル  $v(j)$  は、以下を満たす。

$$A * v(j) = \lambda(j) * v(j)$$

$\lambda(j)$  は、その固有値である。

$A$  の左固有ベクトル  $u(j)$  は、以下を満たす。

$$u(j)^H * A = \lambda(j) * u(j)^H$$

$u(j)^H$  は、 $u(j)$  の共役転置を示す。  
計算された固有ベクトルは、1 に等しいユークリッド・ノルムと最大コンポーネント実数を持つように正規化される。

行列の平衡化は、行と列を置換して行列を上三角により近づけ、対角相似変換  $D A D^{-1}$  ( $D$  は対角行列) を適用して行と列をノルム内でより近づけ、その固有値と固有ベクトルの条件数をより小さくすることを意味する。計算された条件数の逆数は、平衡化された行列に一致する。

行と列を置換しても条件数は変更されないが ( 正確な演算において )、対角をスケールリングすると条件数は変更される。平衡化の詳細は、4.10 の [\[LUG\]](#) を参照。

## 入力パラメータ

**balanc** CHARACTER\*1。'N'、'P'、'S'、または 'B' でなければならない。  
入力行列を対角的にスケールリングする方法、または入力行列を置換してその固有値の条件付けを改善する方法を指定する。

**balanc** = 'N' の場合、対角的なスケールリングまたは置換は実行されない。

**balanc** = 'P' の場合、置換を実行して行列を上三角により近づける。対角的なスケールリングは実行されない。

**balanc** = 'S' の場合、行列に対して対角的なスケールリングが実行される。つまり、 $A$  が  $D A D^{-1}$  で置き換えられる ( $D$  は、 $A$  の行と列をノルム内でより等しくするために選択された対角行列)。置換は実行されない。

**balanc** = 'B' の場合、対角的なスケールリングが実行されるとともに、 $A$  が置換される。

計算された条件数の逆数は、平衡化または置換を実行後の行列に使用される。置換を実行しても条件数は変更されないが ( 正確な演算では )、平衡化を実行すると条件数が変更される。

**jobvl** CHARACTER\*1。'N' または 'V' でなければならない。  
**jobvl** = 'N' の場合、 $A$  の左固有ベクトルは計算されない。  
**jobvl** = 'V' の場合、 $A$  の左固有ベクトルは計算される。  
**sense** = 'E' または 'B' の場合、**jobvl** は 'V' でなければならない。

<i>jobvr</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobvr</i> ='N' の場合、 <i>A</i> の右固有ベクトルは計算されない。 <i>jobvr</i> ='V' の場合、 <i>A</i> の右固有ベクトルは計算される。 <i>sense</i> ='E' または 'B' の場合、 <i>jobvr</i> は 'V' でなければならない。
<i>sense</i>	CHARACTER*1。'N'、'E'、'V'、または 'B' でなければならない。 どの条件数の逆数を計算するのかを決定する。  <i>sense</i> ='N' の場合、計算は実行されない。 <i>sense</i> ='E' の場合、固有値についてのみ計算が実行される。 <i>sense</i> ='V' の場合、右固有ベクトルについてのみ計算が実行される。 <i>sense</i> ='B' の場合、固有値と右固有ベクトルについて計算が実行される。  <i>sense</i> が 'E' または 'B' の場合、左固有ベクトルと右固有ベクトルの両方についても計算を実行する必要がある ( <i>jobvl</i> ='V' と <i>jobvr</i> ='V')。
<i>n</i>	INTEGER。行列 <i>A</i> の次数 ( $n \geq 0$ )。
<i>a</i> , <i>work</i>	REAL ( <i>sgeevx</i> の場合 ) DOUBLE PRECISION ( <i>dgeevx</i> の場合 ) COMPLEX ( <i>cgeevx</i> の場合 ) DOUBLE COMPLEX ( <i>zgeevx</i> の場合 )。 配列： <i>a</i> ( <i>lda</i> ,*) は、 $n \times n$ の行列 <i>A</i> を格納する配列である。 <i>a</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。  <i>work</i> ( <i>lwork</i> ) は、ワークスペース配列である。
<i>lda</i>	INTEGER。配列 <i>a</i> の第 1 次元。 $\max(1, n)$ 以上でなければならない。
<i>ldvl</i> , <i>ldvr</i>	INTEGER。それぞれ、出力配列 <i>vl</i> と <i>vr</i> のリーディング・ディメンジョン。次の制約がある。 $ldvl \geq 1$ 、 $ldvr \geq 1$ 。 <i>jobvl</i> ='V' の場合、 $ldvl \geq \max(1, n)$ 。 <i>jobvr</i> ='V' の場合、 $ldvr \geq \max(1, n)$ 。
<i>lwork</i>	INTEGER。配列 <i>work</i> の次元。 実数型の場合： <i>sense</i> ='N' または 'E' の場合、 $lwork \geq \max(1, 2n)$ 、 <i>jobvl</i> ='V' または <i>jobvr</i> ='V' の場合、 $lwork \geq 3n$ 、

$sense = 'V'$  または  $'B'$  の場合、 $lwork \geq n(n+6)$ 。  
パフォーマンスを改善するには、一般に  $lwork$  に上記以上の値が必要である。

複素数型の場合:

$sense = 'N'$  または  $'E'$  の場合、 $lwork \geq \max(1, 2n)$ 、

$sense = 'V'$  または  $'B'$  の場合、 $lwork \geq n^2 + 2n$ 。

パフォーマンスを改善するには、一般に  $lwork$  に上記以上の値が必要である。

$lwork = -1$  の場合、ワークスペースのクエリとみなされ、ルーチンは  $work$  配列の最適サイズだけを計算し、その値を  $work$  配列の最初のエントリとして返す。xerbla は  $lwork$  に関するエラー・メッセージを生成しない。

$rwork$	REAL (cggeevx の場合 ) DOUBLE PRECISION (zggeevx の場合 ) ワークスペース配列、次元は $\max(1, 2n)$ 以上。複素数型でのみ使用される。
$iwork$	INTEGER。 ワークスペース配列、次元は $\max(1, 2n-2)$ 以上。実数型でのみ使用される。 $sense = 'N'$ または $'E'$ の場合は参照されない。

## 出力パラメータ

$a$	終了時に上書きされる。 $jobvl = 'V'$ または $jobvr = 'V'$ の場合、平衡化した入力行列 $A$ の実数 Schur/Schur 形式が格納される。
$wr, wi$	REAL (sggeevx の場合 ) DOUBLE PRECISION (dsggeevx の場合 ) 配列、次元はそれぞれ $\max(1, n)$ 以上。 計算された固有値の実数部と虚数部がそれぞれ格納される。固有値の複素共役ペアは、虚数部が正である固有値に続いて格納される。
$w$	COMPLEX (cggeevx の場合 ) DOUBLE COMPLEX (zggeevx の場合 )。 配列、次元は $\max(1, n)$ 以上。 計算された固有値が格納される。
$vl, vr$	REAL (sggeevx の場合 ) DOUBLE PRECISION (dsggeevx の場合 ) COMPLEX (cggeevx の場合 )

DOUBLE COMPLEX (zggevz の場合)

配列:

$v1(ldv1,*)$ 。 $v1$  の第 2 次元は、 $\max(1, n)$  以上でなければならない。

$jobv1='V'$  の場合、左固有ベクトル  $u(j)$  は、それらの固有値と同じ順序で、 $v1$  の列に次々と格納される。 $jobv1='N'$  の場合、 $v1$  は参照されない。

実数型の場合:

$j$  番目の固有値が実数の場合、 $u(j) = v1(:, j)$  ( $v1$  の  $j$  番目の列) である。

$j$  番目と  $(j+1)$  番目の固有値が複素共役ペアとなる場合、

$u(j) = v1(:, j) + i*v1(:, j+1)$ 、 $u(j+1) = v1(:, j) - i*v1(:, j+1)$  である。

ここで、 $i = \sqrt{-1}$  である。

複素数型の場合:

$u(j) = v1(:, j)$  ( $v1$  の  $j$  番目の列) である。

$vr(ldvr,*)$ 。 $vr$  の第 2 次元は、 $\max(1, n)$  以上でなければならない。

$jobvr='V'$  の場合、右固有ベクトル  $v(j)$  は、それらの固有値と同じ順序で、 $vr$  の列に次々と格納される。 $jobvr='N'$  の場合、 $vr$  は参照されない。

実数型の場合:

$j$  番目の固有値が実数の場合、 $v(j) = vr(:, j)$  ( $vr$  の  $j$  番目の列) である。

$j$  番目と  $(j+1)$  番目の固有値が複素共役ペアとなる場合、

$v(j) = vr(:, j) + i*vr(:, j+1)$ 、 $v(j+1) = vr(:, j) - i*vr(:, j+1)$  である。

ここで、 $i = \sqrt{-1}$  である。

複素数型の場合:

$v(j) = vr(:, j)$  ( $vr$  の  $j$  番目の列) である。

$ilo, ihi$

INTEGER。

$ilo$  と  $ihi$  は、 $A$  を平衡化したときに決定される整数値である。

$i > j$  と  $j = 1, \dots, ilo-1$  または  $i = ihi+1, \dots, n$  の場合、平衡化した  $A(i, j) = 0$  である。

$balanc='N'$  または  $'S'$  の場合、 $ilo = 1$  と  $ihi = n$  である。

$scale$

REAL (単精度の場合)

DOUBLE PRECISION (倍精度の場合)。

配列、次元は  $\max(1, n)$  以上。

$A$  の平衡化時に適用された置換とスケーリング係数の各成分が格納される。 $P(j)$  が、行と列  $j$  で交換された行と列のインデックスである場合、 $D(j)$  は、行と列  $j$  に適用されたスケーリング係数であり、

$scale(j) = P(j)$ 、( $j = 1, \dots, ilo-1$ )

$= D(j)$ 、( $j = ilo, \dots, ihi$ )

$= P(j)$ 、( $j = ihi+1, \dots, n$ ) である。

交換を実行する順序は、 $n \sim ihi+1$ 、次に  $1 \sim ilo-1$  である。

*abnrm*

REAL (単精度の場合)

DOUBLE PRECISION (倍精度の場合)

平衡化した行列の 1- ノルム (任意の列の成分に対する絶対値の合計の最大値)。

*rconde, rcondv*

REAL (単精度の場合)

DOUBLE PRECISION (倍精度の場合)。

配列、次元はそれぞれ  $\max(1, n)$  以上。

*rconde(j)* は、 $j$  番目の固有値の条件数の逆数である。

*rcondv(j)* は、 $j$  番目の右固有ベクトルの条件数の逆数である。

*work(1)*

終了時に、*info* = 0 の場合、*work(1)* は *lwork* の必要最小サイズを返す。

*info*

INTEGER。

*info* = 0 の場合、実行は正常に終了したことを示す。

*info* = -*i* の場合、*i* 番目のパラメータの値が不正であったことを示す。

*info* = *i* の場合、QR アルゴリズムが固有値をすべて計算できず、固有ベクトルまたは条件数が計算されなかったことを示す。収束した固有値は、*wr* と *wi* (実数型の場合) または *w* (複素数型の場合) の  $1:ilo-1$  と  $i+1:n$  の成分に格納される。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *geevx* のインターフェイスの詳細を以下に示す。

*a*

サイズ ( $n, n$ ) の行列 *A* を格納する。

*wr*

長さ ( $n$ ) のベクトルを格納する。実数型でのみ使用される。

*wi*

長さ ( $n$ ) のベクトルを格納する。実数型でのみ使用される。

*w*

長さ ( $n$ ) のベクトルを格納する。複素数型でのみ使用される。

*vl*

サイズ ( $n, n$ ) の行列 *VL* を格納する。

*vr*

サイズ ( $n, n$ ) の行列 *VR* を格納する。



<i>scale</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>rconde</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>rcondv</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>balanc</i>	'N'、'B'、'P'、または 'S' のいずれかでなければならない。デフォルト値は 'N'。
<i>jobvl</i>	引数 <i>vl</i> の存在に基づいて以下のように復元される。 <i>jobvl</i> = 'V' ( <i>vl</i> が存在する場合)、 <i>jobvl</i> = 'N' ( <i>vl</i> が省略された場合)。
<i>jobvr</i>	引数 <i>vr</i> の存在に基づいて以下のように復元される。 <i>jobvr</i> = 'V' ( <i>vr</i> が存在する場合)、 <i>jobvr</i> = 'N' ( <i>vr</i> が省略された場合)。
<i>sense</i>	引数 <i>rconde</i> と <i>rcondv</i> の存在に基づいて以下のように復元される。 <i>sense</i> = 'B' ( <i>rconde</i> と <i>rcondv</i> の両方が存在する場合)、 <i>sense</i> = 'E' ( <i>rconde</i> が存在し、 <i>rcondv</i> が省略された場合)、 <i>sense</i> = 'V' ( <i>rconde</i> が省略され、 <i>rcondv</i> が存在する場合)、 <i>sense</i> = 'N' ( <i>rconde</i> と <i>rcondv</i> の両方が省略された場合)。

## アプリケーション・ノート

配列 *work* に必要なワークスペースの大きさがわからない場合は、最初の実行で *lwork* を大きめの値に設定する。終了時に、*work*(1) の値を調べ、以降はその値を使用する。

## 特異値分解

このセクションでは、特異値を解くために使用する LAPACK ドライバルーチンについて説明する。これらの問題を解くために呼び出すことが可能な[計算ルーチン](#)も参照のこと。

[表 4-12](#) に Fortran-77 インターフェイスのすべてのドライバルーチンを示す。Fortran-95 インターフェイスのルーチン名はそれぞれ、先頭の記号がない ([ルーチン命名規則](#)を参照)。

**表 4-12**      **特異値分解 (SVD) 用のドライバルーチン**

ルーチン名	機能
<a href="#">?gesvd</a>	一般矩形行列の特異値分解を計算する。
<a href="#">?gesdd</a>	分割統治法を使用して、一般矩形行列の特異値分解を計算する。
<a href="#">?ggsvd</a>	一般矩形行列ペアの汎用特異値分解を計算する。

## ?gesvd

一般矩形行列の特異値分解を計算する。

### 構文

#### Fortran 77:

```
call sgesvd(jobu, jobvt, m, n, a, lda, s, u, ldu, vt, ldvt, work, lwork, info)
call dgesvd(jobu, jobvt, m, n, a, lda, s, u, ldu, vt, ldvt, work, lwork, info)
call cgesvd(jobu, jobvt, m, n, a, lda, s, u, ldu, vt, ldvt, work, lwork, rwork,
            info)
call zgesvd(jobu, jobvt, m, n, a, lda, s, u, ldu, vt, ldvt, work, lwork, rwork,
            info)
```

#### Fortran 95:

```
call gesvd(a, s [,u] [,vt] [,ww] [,job] [,info])
```

### 説明

このルーチンは、実 / 複素行列  $A$  ( $m \times n$ ) の特異値分解 (SVD) と (オプションで) 左特異ベクトルまたは右特異ベクトルを計算する。SVD は、次のように記述される。

$$A = U \Sigma V^H$$

ここで、 $\Sigma$  はその  $\min(m, n)$  対角成分を除いてゼロの  $m \times n$  の行列、 $U$  は  $m \times m$  の直交 / ユニタリ行列、 $V$  は  $n \times n$  の直交 / ユニタリ行列である。 $\Sigma$  の対角成分は、 $A$  の特異値である。これらは、実数かつ非負であり、降順で返される。 $U$  と  $V$  の先頭から  $\min(m, n)$  の列は、 $A$  の左特異ベクトルと右特異ベクトルである。ルーチンは、 $V$  ではなく、 $V^H$  を戻すことに注意が必要である。

### 入力パラメータ

**jobu** CHARACTER\*1。'A'、'S'、'O'、または 'N' でなければならない。  
行列  $U$  のすべてまたは一部を計算するオプションを指定する。  
  
**jobu = 'A'** の場合、 $U$  の  $m$  個の列がすべて配列  $u$  に返される。  
**jobu = 'S'** の場合、 $U$  の先頭から  $\min(m, n)$  の列 (左特異ベクトル) が配列  $u$  に返される。  
**jobu = 'O'** の場合、 $U$  の先頭から  $\min(m, n)$  の列 (左特異ベクトル) が配列  $a$  上で上書きされる。  
**jobu = 'N'** の場合、 $U$  の列 (左特異ベクトル) は計算されない。

<i>jobvt</i>	<p>CHARACTER*1。'A'、'S'、'O'、または 'N' でなければならない。          行列 <math>V^H</math> のすべてまたは一部を計算するオプションを指定する。</p> <p><i>jobvt</i>='A' の場合、<math>V^H</math> の <math>n</math> 個の行がすべて配列 <i>vt</i> に返される。  <i>jobvt</i>='S' の場合、<math>V^H</math> の先頭から <math>\min(m, n)</math> の行 (右特異ベクトル) が配列 <i>vt</i> に返される。  <i>jobvt</i>='O' の場合、<math>V^H</math> の先頭から <math>\min(m, n)</math> の行 (右特異ベクトル) が配列 <i>a</i> 上で書き込まれる。  <i>jobvt</i>='N' の場合、<math>V^H</math> の行 (右特異ベクトル) は計算されない。</p> <p><i>jobvt</i> と <i>jobu</i> は、どちらも 'O' にできない。</p>
<i>m</i>	INTEGER。行列 <i>A</i> の行数 ( $m \geq 0$ )。
<i>n</i>	INTEGER。行列 <i>A</i> の列数 ( $n \geq 0$ )。
<i>a, work</i>	<p>REAL (sgesvd の場合)          DOUBLE PRECISION (dgesvd の場合)          COMPLEX (cgesvd の場合)          DOUBLE COMPLEX (zgesvd の場合)。          配列:  <i>a</i>(<i>lda</i>,*) は、<math>m \times n</math> の行列 <i>A</i> を格納する配列である。  <i>a</i> の第 2 次元は、<math>\max(1, n)</math> 以上でなければならない。</p> <p><i>work</i>(<i>lwork</i>) は、ワークスペース配列である。</p>
<i>lda</i>	<p>INTEGER。配列 <i>a</i> の第 1 次元。  <math>\max(1, m)</math> 以上でなければならない。</p>
<i>ldu, ldvt</i>	<p>INTEGER。それぞれ、出力配列 <i>u</i> と <i>vt</i> のリーディング・ディメンジョン。次の制約がある。  <math>ldu \geq 1</math>; <math>ldvt \geq 1</math>。  <i>jobu</i>='S' または 'A' の場合、<math>ldu \geq m</math>。  <i>jobvt</i>='A' の場合、<math>ldvt \geq n</math>。  <i>jobvt</i>='S' の場合、<math>ldvt \geq \min(m, n)</math>。</p>
<i>lwork</i>	<p>INTEGER。配列 <i>work</i> の次元。 <math>lwork \geq 1</math>。          次の制約がある。  <math>lwork \geq \max(3 * \min(m, n) + \max(m, n), 5 * \min(m, n))</math> (実数型の場合)。  <math>lwork \geq 2 * \min(m, n) + \max(m, n)</math> (複素数型の場合)。          パフォーマンスを改善するには、一般に <i>lwork</i> に上記以上の値が必要である。  <math>lwork = -1</math> の場合、ワークスペースのクエリとみなされ、ルーチンは</p>

*work* 配列の最適サイズだけを計算し、その値を *work* 配列の最初のエントリとして返す。xerbla は *lwork* に関するエラー・メッセージを生成しない。

*rwork* REAL (cgesvd の場合 )  
DOUBLE PRECISION (zgesvd の場合 )  
ワークスペース配列、次元は  $\max(1, 5 \cdot \min(m, n))$  以上。複素数型でのみ使用される。

## 出力パラメータ

*a* 終了時に、  
*jobu*='O' の場合、*a* は、*U* の先頭から  $\min(m, n)$  の列 ( 列方向に格納される左特異ベクトル ) で上書きされる。  
*jobvt*='O' の場合、*a* は、*V<sup>H</sup>* の先頭から  $\min(m, n)$  の行 ( 行方向に格納される右特異ベクトル ) で上書きされる。  
*jobu* ≠ 'O' かつ *jobvt* ≠ 'O' の場合、*a* の内容は破棄される。

*s* REAL ( 単精度の場合 )  
DOUBLE PRECISION ( 倍精度の場合 )。  
配列、次元は  $\max(1, \min(m, n))$  以上。  
*s*(*i*) ≥ *s*(*i*+1) となるようにソートされて *A* の特異値が格納される。

*u*, *vt* REAL (sgesvd の場合 )  
DOUBLE PRECISION (dgesvd の場合 )  
COMPLEX (cgesvd の場合 )  
DOUBLE COMPLEX (zgesvd の場合 )。  
配列 :  
*u*(*ldu*,\*)。 *u* の第 2 次元は、 $\max(1, m)$  以上 (*jobu*='A' の場合 ) または  $\max(1, \min(m, n))$  (*jobu*='S' の場合 ) 以上でなければならない。  
*jobu*='A' の場合、*u* には、 $m \times m$  の直交 / ユニタリ行列 *U* が格納される。  
*jobu*='S' の場合、*u* には、*U* の先頭から  $\min(m, n)$  の列 ( 列方向に格納される左特異ベクトル ) が格納される。  
*jobu*='N' または 'O' の場合、*u* は参照されない。  
*vt*(*ldvt*,\*)。 *vt* の第 2 次元は、 $\max(1, n)$  以上でなければならない。  
*jobvt*='A' の場合、*vt* には、 $n \times n$  の直交 / ユニタリ行列 *V<sup>H</sup>* が格納される。  
*jobvt*='S' の場合、*vt* には、*V<sup>H</sup>* の先頭から  $\min(m, n)$  の行 ( 行方向に格納される右特異ベクトル ) が格納される。  
*jobvt*='N' または 'O' の場合、*vt* は参照されない。

<i>work</i>	終了時に、 <i>info</i> = 0 の場合、 <i>work</i> (1) は <i>lwork</i> の必要最小サイズを返す。 実数型の場合： <i>info</i> > 0 の場合、 <i>work</i> (2:min( <i>m</i> , <i>n</i> )) には、対角が (必ずしもソートされていない) <i>s</i> 内にある上二重対角行列 <i>B</i> の非収束優対角成分が格納される。 <i>B</i> は、 $A = u * B * vt$ を満たすため、 <i>A</i> と同じ特異値を持ち、 <i>u</i> と <i>vt</i> で関連付けられた特異ベクトルを持つ。
<i>rwork</i>	終了時 (複素数型の場合)、 <i>info</i> > 0 であれば、 <i>rwork</i> (1:min( <i>m</i> , <i>n</i> )-1) には、対角が (必ずしもソートされていない) <i>s</i> 内にある上二重対角行列 <i>B</i> の非収束優対角成分が格納される。 <i>B</i> は、 $A = u * B * vt$ を満たすため、 <i>A</i> と同じ特異値を持ち、 <i>u</i> と <i>vt</i> で関連付けられた特異ベクトルを持つ。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正であったことを示す。 <i>info</i> = <i>i</i> で、?bdsqr が収束していなかった場合、 <i>i</i> は、ゼロに収束しなかった中間二重対角形式 <i>B</i> の優対角の数である。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン gesvd のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ ( <i>m</i> , <i>n</i> ) の行列 <i>A</i> を格納する。
<i>s</i>	長さ min( <i>m</i> , <i>n</i> ) のベクトルを格納する。
<i>u</i>	サイズ ( <i>m</i> , min( <i>m</i> , <i>n</i> )) の行列 <i>U</i> を格納する。
<i>vt</i>	サイズ (min( <i>m</i> , <i>n</i> ), <i>n</i> ) の行列 <i>VT</i> を格納する。
<i>ww</i>	長さ (min( <i>m</i> , <i>n</i> )-1) のベクトルを格納する。
<i>ww</i>	長さ min( <i>m</i> , <i>n</i> )-1 のベクトルを格納する。 <i>ww</i> には、対角が (必ずしもソートされていない) <i>s</i> 内にある上二重対角行列 <i>B</i> の非収束優対角成分が格納される。 <i>B</i> は、 $A = U * B * VT$ を満たすため、 <i>A</i> と同じ特異値を持ち、 <i>U</i> と <i>VT</i> で関連付けられた特異ベクトルを持つ。

<i>job</i>	'N'、'U'、または 'V' でなければならない。デフォルト値は 'N'。 <i>job</i> = 'U' で、 <i>u</i> が存在しない場合、 <i>u</i> は配列 <i>a</i> に返される。 <i>job</i> = 'V' で、 <i>vt</i> が存在しない場合、 <i>vt</i> は配列 <i>a</i> に返される。
<i>jobu</i>	引数 <i>u</i> の存在、 <i>job</i> の値、配列 <i>u</i> と <i>a</i> のサイズに基づいて以下のように復元される。 <i>jobu</i> = 'A' ( <i>u</i> が存在し、 <i>u</i> の列数が <i>a</i> の行数と等しい場合)、 <i>jobu</i> = 'S' ( <i>u</i> が存在し、 <i>u</i> の列数が <i>a</i> の行数と等しくない場合)、 <i>jobu</i> = 'O' ( <i>u</i> が省略され、 <i>job</i> が 'U' と等しい場合)、 <i>jobu</i> = 'N' ( <i>u</i> が省略され、 <i>job</i> が 'U' と等しくない場合)。
<i>jobvt</i>	引数 <i>vt</i> の存在、 <i>job</i> の値、配列 <i>vt</i> と <i>a</i> のサイズに基づいて以下のように復元される。 <i>jobvt</i> = 'A' ( <i>vt</i> が存在し、 <i>vt</i> の列数が <i>a</i> の行数と等しい場合)、 <i>jobvt</i> = 'S' ( <i>vt</i> が存在し、 <i>vt</i> の列数が <i>a</i> の行数と等しくない場合)、 <i>jobvt</i> = 'O' ( <i>vt</i> が省略され、 <i>job</i> が 'V' と等しい場合)、 <i>jobu</i> = 'N' ( <i>vt</i> が省略され、 <i>job</i> が 'V' と等しくない場合)。

### アプリケーション・ノート

配列 *work* に必要なワークスペースの大きさがわからない場合は、最初の実行で *lwork* を大きめの値に設定する。終了時に、*work(1)* の値を調べ、以降はその値を使用する。

---

## ?gesdd

分割統治法を使用して、一般矩形行列の特異値分解を計算する。

---

### 構文

#### Fortran 77:

```
call sgesdd(jobz, m, n, a, lda, s, u, ldu, vt, ldvt, work, lwork, iwork, info)
call dgesdd(jobz, m, n, a, lda, s, u, ldu, vt, ldvt, work, lwork, iwork, info)
call cgesdd(jobz, m, n, a, lda, s, u, ldu, vt, ldvt, work, lwork, rwork, iwork,
            info)
call zgesdd(jobz, m, n, a, lda, s, u, ldu, vt, ldvt, work, lwork, rwork, iwork,
            info)
```

**Fortran 95:**

```
call gesdd(a, s [,u] [,vt] [,jobz] [,info])
```

**説明**

このルーチンは、実 / 複素行列  $A$  ( $m \times n$ ) の特異値分解 (SVD) と (オプションで) 左特異ベクトルまたは右特異ベクトルを計算する。特異ベクトルを計算する場合、分割統治アルゴリズムを使用する。

SVD は、次のように記述される。

$$A = U \Sigma V^H$$

ここで、 $\Sigma$  はその  $\min(m, n)$  対角成分を除いてゼロの  $m \times n$  の行列、 $U$  は  $m \times m$  の直交 / ユニタリ行列、 $V$  は  $n \times n$  の直交 / ユニタリ行列である。 $\Sigma$  の対角成分は、 $A$  の特異値である。これらは、実数かつ非負であり、降順で返される。 $U$  と  $V$  の先頭から  $\min(m, n)$  の列は、 $A$  の左特異ベクトルと右特異ベクトルである。

ルーチンは、 $V$  ではなく、 $V^H$  を戻すことに注意が必要である。

**入力パラメータ**

**jobz** CHARACTER\*1。'A'、'S'、'O'、または 'N' でなければならない。  
行列  $U$  のすべてまたは一部を計算するオプションを指定する。

**jobz = 'A'** の場合、 $U$  の  $m$  個の列、 $V^T$  の  $n$  個の行は、すべて配列  $u$  と  $vt$  に返される。

**jobz = 'S'** の場合、 $U$  の先頭から  $\min(m, n)$  の列、 $V^T$  の先頭から  $\min(m, n)$  の行は、配列  $u$  と  $vt$  に返される。

**jobz = 'O'** の場合、

$m \geq n$  であれば、 $U$  の先頭から  $n$  個の列が配列  $a$  上で上書きされ、 $V^T$  の行がすべて配列  $vt$  に返される。

$m < n$  であれば、 $U$  の列がすべて配列  $u$  に返され、 $V^T$  の先頭から  $m$  個の行が配列  $vt$  内で上書きされる。

**jobz = 'N'** の場合、 $U$  の列、または  $V^T$  の行は計算されない。

**m** INTEGER。行列  $A$  の行数 ( $m \geq 0$ )。

**n** INTEGER。行列  $A$  の列数 ( $n \geq 0$ )。

**a, work** REAL (sgesdd の場合 )  
DOUBLE PRECISION (dgesdd の場合 )  
COMPLEX (cgesdd の場合 )  
DOUBLE COMPLEX (zgesdd の場合 )。

配列:

$a(lda,*)$  は、 $m \times n$  の行列  $A$  を格納する配列である。  
 $a$  の第 2 次元は、 $\max(1, n)$  以上でなければならない。

$work(lwork)$  は、ワークスペース配列である。

$lda$	INTEGER。配列 $a$ の第 1 次元。 $\max(1, m)$ 以上でなければならない。
$ldu, ldvt$	INTEGER。それぞれ、出力配列 $u$ と $vt$ のリーディング・ディメンジョン。次の制約がある。 $ldu \geq 1; ldvt \geq 1$ 。 $jobz = 'S'$ または $'A'$ の場合、または $jobz = 'O'$ の場合、 $m < n$ であれば、 $ldu \geq m$ 。 $jobz = 'A'$ 、または $jobz = 'O'$ の場合、 $m \geq n$ であれば、 $ldvt \geq n$ 。 $jobz = 'S'$ の場合、 $ldvt \geq \min(m, n)$ である。
$lwork$	INTEGER。配列 $work$ の次元。 $lwork \geq 1$ 。 $lwork = -1$ の場合、ワークスペースのクエリとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエントリとして返す。 $xerbla$ は $lwork$ に関するエラー・メッセージを生成しない。 $lwork$ の推奨値は、「アプリケーション・ノート」を参照。
$rwork$	REAL (cgesdd の場合) DOUBLE PRECISION (zgesdd の場合) $jobz = 'N'$ の場合、ワークスペース配列、次元は $\max(1, 5 * \min(m, n))$ 以上。それ以外の場合、 $rwork$ の次元は、 $5 * (\min(m, n))^2 + 7 * \min(m, n)$ 以上でなければならない。この配列は、複素数型でのみ使用される。
$iwork$	INTEGER。ワークスペース配列、次元は $\max(1, 8 * \min(m, n))$ 以上。

## 出力パラメータ

$a$	終了時に、 $jobz = 'O'$ の場合、 $m \geq n$ であれば、 $a$ は、 $U$ の先頭から $n$ 個の列 (列方向に格納される左特異ベクトル) で上書きされる。 $m < n$ であれば、 $a$ は、 $V^T$ の先頭から $m$ 個の行 (行方向に格納される右特異ベクトル) で上書きされる。 $jobz \neq 'O'$ の場合、 $a$ の内容は破棄される。
$s$	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 配列、次元は $\max(1, \min(m, n))$ 以上。 $s(i) \geq s(i+1)$ となるようにソートされて $A$ の特異値が格納される。



<i>u, vt</i>	<p>REAL (sgesdd の場合 )  DOUBLE PRECISION (dgesdd の場合 )  COMPLEX (cgesdd の場合 )  DOUBLE COMPLEX (zgesdd の場合 )。  配列：  <i>u(ldu,*)</i>。 <i>u</i> の第 2 次元は、<math>\max(1, m)</math> 以上でなければならない  (<i>jobz</i>='A' または <i>jobz</i>='O' であり、<math>m &lt; n</math> の場合 )。  <i>jobz</i>='S' の場合、 <i>u</i> の第 2 次元は、<math>\max(1, \min(m, n))</math> 以上でなければならない。</p> <p><i>jobz</i>='A' または <i>jobz</i>='O' の場合、 <math>m &lt; n</math> であれば、 <i>u</i> には、 <math>m \times m</math> の直交 / ユニタリ行列 <i>U</i> が格納される。  <i>jobz</i>='S' の場合、 <i>u</i> には、 <i>U</i> の先頭から <math>\min(m, n)</math> の列 ( 列方向に格納される左特異ベクトル ) が格納される。  <i>jobz</i>='O' かつ <math>m \geq n</math>、 または <i>jobz</i>='N' の場合、 <i>u</i> は参照されない。</p> <p><i>vt(ldvt,*)</i>。 <i>vt</i> の第 2 次元は、<math>\max(1, n)</math> 以上でなければならない。</p> <p><i>jobz</i>='A' または <i>jobz</i>='O' の場合、 <math>m \geq n</math> であれば、 <i>vt</i> には、 <math>n \times n</math> の直交 / ユニタリ行列 <i>V<sup>T</sup></i> が格納される。  <i>jobz</i>='S' の場合、 <i>vt</i> には、 <i>V<sup>T</sup></i> の先頭から <math>\min(m, n)</math> の行 ( 行方向に格納される右特異ベクトル ) が格納される。  <i>jobz</i>='O' かつ <math>m &lt; n</math>、 または <i>jobz</i>='N' の場合、 <i>vt</i> は参照されない。</p>
<i>work(1)</i>	終了時に、 <i>info</i> = 0 の場合、 <i>work(1)</i> は <i>lwork</i> の必要最小サイズを返す。
<i>info</i>	<p>INTEGER。  <i>info</i> = 0 の場合、実行は正常に終了したことを示す。  <i>info</i> = -<i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正であったことを示す。  <i>info</i> = <i>i</i> で、 ?bdsdc が収束していなかった場合、更新プロセスが失敗したことを示す。</p>

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン gesdd のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ ( <i>m, n</i> ) の行列 <i>A</i> を格納する。
<i>s</i>	長さ $\min(m, n)$ のベクトルを格納する。

<code>u</code>	サイズ $(m, \min(m, n))$ の行列 $U$ を格納する。
<code>vt</code>	サイズ $(\min(m, n), n)$ の行列 $VT$ を格納する。
<code>jobz</code>	'N'、'A'、'S'、または 'O' でなければならない。デフォルト値は 'N'。

### アプリケーション・ノート

実数型の場合:

`jobz = 'N'` の場合、 $lwork \geq 3 * \min(m, n) + \max(\max(m, n), 6 * \min(m, n))$ 。

`jobz = 'O'` の場合、 $lwork \geq 3 * (\min(m, n))^2 + \max(\max(m, n), 5 * (\min(m, n))^2 + 4 * \min(m, n))$ 。

`jobz = 'S'` または 'A' の場合、

$lwork \geq 3 * (\min(m, n))^2 + \max(\max(m, n), 4 * (\min(m, n))^2 + 4 * \min(m, n))$ 。

複素数型の場合:

`jobz = 'N'` の場合、 $lwork \geq 2 * \min(m, n) + \max(m, n)$ 。

`jobz = 'O'` の場合、 $lwork \geq 2 * (\min(m, n))^2 + \max(m, n) + 2 * \min(m, n)$ 。

`jobz = 'S'` または 'A' の場合、 $lwork \geq (\min(m, n))^2 + \max(m, n) + 2 * \min(m, n)$ 。

パフォーマンスを改善するには、一般に `lwork` に上記以上の値が必要である。

配列 `work` に必要なワークスペースの大きさがわからない場合は、最初の実行で `lwork` を大きめの値に設定する。終了時に、`work(1)` の値を調べ、以降はその値を使用する。

---

## ?ggsvd

一般矩形行列ペアの汎用特異値分解を計算する。

---

### 構文

#### Fortran 77:

```
call sggsvd(jobu, jobv, jobq, m, n, p, k, l, a, lda, b, ldb, alpha,
            beta, u, ldu, v, ldv, q, ldq, work, iwork, info)
call dggsvd(jobu, jobv, jobq, m, n, p, k, l, a, lda, b, ldb, alpha,
            beta, u, ldu, v, ldv, q, ldq, work, iwork, info)
call cggsvd(jobu, jobv, jobq, m, n, p, k, l, a, lda, b, ldb, alpha,
            beta, u, ldu, v, ldv, q, ldq, work, rwork, iwork, info)
call zggsvd(jobu, jobv, jobq, m, n, p, k, l, a, lda, b, ldb, alpha,
            beta, u, ldu, v, ldv, q, ldq, work, rwork, iwork, info)
```

**Fortran 95:**

```
call ggsvd(a, b, alpha, beta [,k] [,l] [,u] [,v] [,q] [,iwork] [,info])
```

**説明**

このルーチンは、 $m \times n$  の実 / 複素行列  $A$  と  $p \times n$  の実 / 複素行列  $B$  の汎用特異値分解 (GSVD) を計算する。

$$U^H A Q = D_1 * (0 \ R), \quad V^H B Q = D_2 * (0 \ R)$$

ここで、 $U$ 、 $V$ 、および  $Q$  は直交 / ユニタリ行列である。

$k+1=$  が行列  $(A^H, B^H)^H$  の有効な数値ランクの場合、 $R$  は、 $(k+1) \times (k+1)$  の非特異上三角行列となり、 $D_1$  と  $D_2$  は、 $m \times (k+1)$  の "対角" 行列と  $p \times (k+1)$  の "対角" 行列となる。 $D_1$  と  $D_2$  はそれぞれ、次の構造で示される。

$m-k-1 \geq 0$  の場合、

$$D_1 = \begin{matrix} & k & 1 \\ & \begin{pmatrix} I & 0 \\ 0 & C \end{pmatrix} \\ m-k-1 & \begin{pmatrix} 0 & 0 \end{pmatrix} \end{matrix}$$

$$D_2 = \begin{matrix} & k & 1 \\ 1 & \begin{pmatrix} 0 & S \\ 0 & 0 \end{pmatrix} \\ p-1 & \end{matrix}$$

$$(0 \ R) = \begin{matrix} & n-k-1 & k & 1 \\ k & \begin{pmatrix} 0 & R_{11} & R_{12} \\ 0 & 0 & R_{22} \end{pmatrix} \\ 1 & \end{matrix}$$

$$C = \text{diag}(\alpha(k+1), \dots, \alpha(k+1))$$

$$S = \text{diag}(\beta(k+1), \dots, \beta(k+1))$$

$$C^2 + S^2 = I$$

$R$  は、終了時に、 $a(1:k+1, n-k-1+1:n)$  に格納される。

$m-k-l < 0$  の場合、

$$D_1 = \begin{matrix} & k & m-k & k+l-m \\ m-k & \begin{pmatrix} I & 0 & 0 \\ 0 & C & 0 \end{pmatrix} \end{matrix}$$

$$D_2 = \begin{matrix} & k & m-k & k+l-m \\ m-k & \begin{pmatrix} 0 & S & 0 \\ 0 & 0 & I \\ p-l & 0 & 0 \end{pmatrix} \end{matrix}$$

$$\begin{pmatrix} 0 & R \end{pmatrix} = \begin{matrix} & n-k-l & k & m-k & k+l-m \\ k & \begin{pmatrix} 0 & R_{11} & R_{12} & R_{13} \\ 0 & 0 & R_{22} & R_{23} \\ 0 & 0 & 0 & R_{33} \end{pmatrix} \\ m-k & \\ k+l-m & \end{matrix}$$

$$\begin{aligned} C &= \text{diag}(\alpha(k+1), \dots, \alpha(m)), \\ S &= \text{diag}(\beta(k+1), \dots, \beta(m)), \\ C^2 + S^2 &= I \end{aligned}$$

終了時に、 $\begin{pmatrix} R_{11} & R_{12} & R_{13} \\ 0 & R_{22} & R_{23} \end{pmatrix}$  は  $a(1:m, n-k-l+1:n)$  に格納され、 $R_{33}$  は

$b(m-k+1:l, n+m-k-l+1:n)$  に格納される。

このルーチンは、 $C$ 、 $S$ 、 $R$  と (オプションで) 直交 / ユニタリ変換行列  $U$ 、 $V$ 、 $Q$  を計算する。

特に、 $B$  が  $n \times n$  の非特異行列の場合、 $A$  と  $B$  の GSVD は、暗黙的に  $AB^{-1}$  の SVD を提供する。

$$AB^{-1} = U(D_1 D_2^{-1}) V^H$$

$(A^H, B^H)^H$  が直交列を持っている場合、 $A$  と  $B$  の GSVD も  $A$  と  $B$  の CS 分解に等しくなる。更に、GSVD を使用すると、固有値問題の解を導き出すことができる。

$$A^H A x = \lambda B^H B x$$

### 入力パラメータ

<i>jobu</i>	CHARACTER*1。'U' または 'N' でなければならない。 <i>jobu</i> ='U' の場合、直交 / ユニタリ行列 $U$ が計算される。 <i>jobu</i> ='N' の場合、 $U$ は計算されない。
<i>jobv</i>	CHARACTER*1。'V' または 'N' でなければならない。 <i>jobv</i> ='V' の場合、直交 / ユニタリ行列 $V$ が計算される。 <i>jobv</i> ='N' の場合、 $V$ は計算されない。
<i>jobq</i>	CHARACTER*1。'Q' または 'N' でなければならない。 <i>jobq</i> ='Q' の場合、直交 / ユニタリ行列 $Q$ が計算される。 <i>jobq</i> ='N' の場合、 $Q$ は計算されない。
<i>m</i>	INTEGER。行列 $A$ の行数 ( $m \geq 0$ )。
<i>n</i>	INTEGER。行列 $A$ と $B$ の列数 ( $n \geq 0$ )。
<i>p</i>	INTEGER。行列 $B$ の行数 ( $p \geq 0$ )。
<i>a, b, work</i>	REAL (sggsvd の場合) DOUBLE PRECISION (dggsvd の場合) COMPLEX (cggsvd の場合) DOUBLE COMPLEX (zggsvd の場合)。 配列： $a(lda, *)$ には、 $m \times n$ の行列 $A$ を格納する。 $a$ の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $b(l db, *)$ には、 $p \times n$ の行列 $B$ を格納する。 $b$ の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $work(*)$ は、ワークスペース配列である。 $work$ の次元は、 $\max(3n, m, p)+n$ 以上でなければならない。
<i>lda</i>	INTEGER。 $a$ の第 1 次元。 $\max(1, m)$ 以上。
<i>ldb</i>	INTEGER。 $b$ の第 1 次元。 $\max(1, p)$ 以上。
<i>ldu</i>	INTEGER。配列 $u$ の第 1 次元。 <i>jobu</i> ='U' の場合、 $ldu \geq \max(1, m)$ 、それ以外の場合、 $ldu \geq 1$ 。
<i>ldv</i>	INTEGER。配列 $v$ の第 1 次元。 <i>jobv</i> ='V' の場合、 $ldv \geq \max(1, p)$ 、それ以外の場合、 $ldv \geq 1$ 。

<i>ldq</i>	INTEGER。配列 <i>q</i> の第 1 次元。 <i>jobq</i> ='Q' の場合、 $ldq \geq \max(1, n)$ 、それ以外の場合、 $ldq \geq 1$ 。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, n)$ 以上。
<i>rwork</i>	REAL (cggsvd の場合 ) DOUBLE PRECISION (zggsvd の場合 )。 ワークスペース配列、次元は $\max(1, 2n)$ 以上。複素数型でのみ使用される。

## 出力パラメータ

<i>k, l</i>	INTEGER。終了時に、 <i>k</i> と <i>l</i> は、サブブロックの次元を指定する。 <i>k + l</i> の合計は、 $(A^H, B^H)^H$ の有効数値ランクに等しい。
<i>a</i>	終了時に、 <i>a</i> には、三角行列 <i>R</i> または <i>R</i> の一部が格納される。
<i>b</i>	終了時に、 $m-k-l < 0$ の場合、 <i>b</i> には、三角行列 <i>R</i> の一部が格納される。
<i>alpha, beta</i>	REAL ( 単精度の場合 ) DOUBLE PRECISION ( 倍精度の場合 )。 配列、次元はそれぞれ $\max(1, n)$ 以上。 <i>A</i> と <i>B</i> の汎用特異値ペアが、次のように格納される。  $\alpha(1:k) = 1$ $\beta(1:k) = 0$ および、 $m-k-l \geq 0$ の場合、 $\alpha(k+1:k+l) = C、$ $\beta(k+1:k+l) = S、$ $m-k-l < 0$ の場合、 $\alpha(k+1:m) = C, \alpha(m+1:k+l) = 0$ $\beta(k+1:m) = S, \beta(m+1:k+l) = 1$ および、 $\alpha(k+l+1:n) = 0$ $\beta(k+l+1:n) = 0$
<i>u, v, q</i>	REAL (sggsvd の場合 ) DOUBLE PRECISION (dggsvd の場合 ) COMPLEX (cggsvd の場合 ) DOUBLE COMPLEX (zggsvd の場合 )。 配列：

$u(ldu,*)$ 。  $u$  の第 2 次元は、 $\max(1, m)$  以上でなければならない。  
 $jobu='U'$  の場合、 $u$  には、 $m \times m$  の直交 / ユニタリ行列  $U$  が格納される。  
 $jobu='N'$  の場合、 $u$  は参照されない。  
 $v(ldv,*)$ 。  $v$  の第 2 次元は、 $\max(1, p)$  以上でなければならない。  
 $jobv='V'$  の場合、 $v$  には、 $p \times p$  の直交 / ユニタリ行列  $V$  が格納される。  
 $jobv='N'$  の場合、 $v$  は参照されない。  
 $q(ldq,*)$ 。  $q$  の第 2 次元は、 $\max(1, n)$  以上でなければならない。  
 $jobq='Q'$  の場合、 $q$  には、 $n \times n$  の直交 / ユニタリ行列  $Q$  が格納される。  
 $jobq='N'$  の場合、 $q$  は参照されない。  
*iwork*                    終了時に、*iwork* には、ソート情報が格納される。  
*info*                    INTEGER。  
                          *info* = 0 の場合、実行は正常に終了したことを示す。  
                          *info* = -*i* の場合、*i* 番目のパラメータの値が不正であったことを示す。  
                          *info* = 1 の場合、Jacobi 型プロシージャが収束できなかったことを示す。詳細は、サブルーチン [ztgsja](#) を参照。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン ggsvd のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ ( $m, n$ ) の行列 $A$ を格納する。
<i>b</i>	サイズ ( $p, n$ ) の行列 $B$ を格納する。
<i>alpha</i>	長さ ( $n$ ) のベクトルを格納する。
<i>beta</i>	長さ ( $n$ ) のベクトルを格納する。
<i>u</i>	サイズ ( $m, m$ ) の行列 $U$ を格納する。
<i>v</i>	サイズ ( $p, p$ ) の行列 $V$ を格納する。
<i>q</i>	サイズ ( $n, n$ ) の行列 $Q$ を格納する。
<i>iwork</i>	長さ ( $n$ ) のベクトルを格納する。

<i>jobu</i>	引数 <i>u</i> の存在に基づいて以下のように復元される。 <i>jobu</i> = 'U' ( <i>u</i> が存在する場合)、 <i>jobu</i> = 'N' ( <i>u</i> が省略された場合)。
<i>jobv</i>	引数 <i>v</i> の存在に基づいて以下のように復元される。 <i>jobv</i> = 'V' ( <i>v</i> が存在する場合)、 <i>jobv</i> = 'N' ( <i>v</i> が省略された場合)。
<i>jobq</i>	引数 <i>q</i> の存在に基づいて以下のように復元される。 <i>jobq</i> = 'Q' ( <i>q</i> が存在する場合)、 <i>jobq</i> = 'N' ( <i>q</i> が省略された場合)。

## 汎用対称固有値問題

このセクションでは、汎用対称固有値問題を解くために使用する LAPACK ドライバルーチンについて説明する。これらの問題を解くために呼び出すことが可能な[計算ルーチン](#)も参照のこと。

[表 4-13](#) に Fortran-77 インターフェイスのすべてのドライバルーチンを示す。Fortran-95 インターフェイスのルーチン名はそれぞれ、先頭の記号がない ([ルーチン命名規則](#)を参照)。

**表 4-13 汎用対称固有値問題を解くためのドライバルーチン**

ルーチン名	機能
<a href="#">?sygv</a> / <a href="#">?hegv</a>	実数 / 複素数の汎用対称 / エルミート固有値問題について、固有値と (オプションで) 固有ベクトルをすべて計算する。
<a href="#">?sygvd</a> / <a href="#">?hegvd</a>	実数 / 複素数の汎用対称 / エルミート固有値問題について、固有値と (オプションで) 固有ベクトルをすべて計算する。固有ベクトルを計算する場合は、分割統治法を使用する。
<a href="#">?sygvx</a> / <a href="#">?hegvx</a>	実数 / 複素数の汎用対称 / エルミート固有値問題について、固有値と (オプションで) 固有ベクトルを選択的に計算する。
<a href="#">?spgv</a> / <a href="#">?hpgv</a>	圧縮格納形式の行列に関する実数 / 複素数の汎用対称 / エルミート固有値問題について、固有値と (オプションで) 固有ベクトルをすべて計算する。
<a href="#">?spgvd</a> / <a href="#">?hpgvd</a>	圧縮格納形式の行列に関する実数 / 複素数の汎用対称 / エルミート固有値問題について、固有値と (オプションで) 固有ベクトルをすべて計算する。固有ベクトルを計算する場合は、分割統治法を使用する。
<a href="#">?spgvx</a> / <a href="#">?hpgvx</a>	圧縮格納形式の行列に関する実数 / 複素数の汎用対称 / エルミート固有値問題について、固有値と (オプションで) 固有ベクトルを選択的に計算する。
<a href="#">?sbqv</a> / <a href="#">?hbqv</a>	帯行列に関する実数 / 複素数の汎用対称 / エルミート固有値問題について、固有値と (オプションで) 固有ベクトルをすべて計算する。



表 4-13 汎用対称固有値問題を解くためのドライバルーチン

ルーチン名	機能
<a href="#">?sbgvd</a> / <a href="#">?hbgvd</a>	帯行列に関する実数 / 複素数の汎用対称 / エルミート固有値問題について、固有値と (オプションで) 固有ベクトルをすべて計算する。固有ベクトルを計算する場合は、分割統治法を使用する。
<a href="#">?sbgvx</a> / <a href="#">?hbgvx</a>	帯行列に関する実数 / 複素数の汎用対称 / エルミート固有値問題について、固有値と (オプションで) 固有ベクトルを選択的に計算する。

?sygv

実数の汎用対称固有値問題について、固有値と (オプションで) 固有ベクトルをすべて計算する。

構文

Fortran 77:

```
call ssygv(itype, jobz, uplo, n, a, lda, b, ldb, w, work, lwork, info)
call dsygv(itype, jobz, uplo, n, a, lda, b, ldb, w, work, lwork, info)
```

Fortran 95:

```
call sygv(a, b, w [,itype] [,jobz] [,uplo] [,info])
```

説明

このルーチンは、次の形式で示される実数の汎用対称固有値問題について、固有値と (オプションで) 固有ベクトルをすべて計算する。

$Ax = \lambda Bx$ ,  $ABx = \lambda x$ , または  $BAx = \lambda x$

$A$  と  $B$  は対称であると仮定する。また、 $B$  は正定値である。

入力パラメータ

*itype*                    INTEGER。1、2、または 3 のいずれかでなければならない。  
                          解決する問題のタイプを指定する。  
                          *itype* = 1 の場合、問題のタイプは  $Ax = \lambda Bx$  である。  
                          *itype* = 2 の場合、問題のタイプは  $ABx = \lambda x$  である。  
                          *itype* = 3 の場合、問題のタイプは  $BAx = \lambda x$  である。

<i>jobz</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobz</i> ='N' の場合、固有値のみを計算する。 <i>jobz</i> ='V' の場合、固有値と固有ベクトルを計算する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> ='U' の場合、配列 <i>a</i> と <i>b</i> は、 <i>A</i> と <i>B</i> の上三角を格納する。 <i>uplo</i> ='L' の場合、配列 <i>a</i> と <i>b</i> は、 <i>A</i> と <i>B</i> の下三角を格納する。
<i>n</i>	INTEGER。行列 <i>A</i> と <i>B</i> の次数 ( $n \geq 0$ )。
<i>a</i> , <i>b</i> , <i>work</i>	REAL(ssygv の場合) DOUBLE PRECISION(dsygv の場合)。 配列: <i>a</i> ( <i>lda</i> ,*) には、 <i>uplo</i> の値に従って、対称行列 <i>A</i> の上三角または下三角を格納する。 <i>a</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。  <i>b</i> ( <i>ldb</i> ,*) には、 <i>uplo</i> の値に従って、対称正定値行列 <i>B</i> の上三角または下三角を格納する。 <i>b</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。  <i>work</i> ( <i>lwork</i> ) は、ワークスペース配列である。
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $\max(1, n)$ 以上。
<i>ldb</i>	INTEGER。 <i>b</i> の第 1 次元。 $\max(1, n)$ 以上。
<i>lwork</i>	INTEGER。配列 <i>work</i> の次元。 $lwork \geq \max(1, 3n-1)$ 。 <i>lwork</i> = -1 の場合、ワークスペースのクエリとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリとして返す。 <i>xerbla</i> は <i>lwork</i> に関するエラー・メッセージを生成しない。  <i>lwork</i> の推奨値は、「アプリケーション・ノート」を参照。

## 出力パラメータ

<i>a</i>	終了時に、 <i>jobz</i> ='V' の場合、 <i>info</i> =0 であれば、 <i>a</i> には、固有ベクトルの行列 <i>Z</i> が格納される。固有ベクトルは、次のように正規化される。 <i>itype</i> = 1 または 2 の場合、 $Z^T B Z = I$ 。 <i>itype</i> = 3 の場合、 $Z^T B^{-1} Z = I$ 。  <i>jobz</i> ='N' の場合、終了時に、 <i>A</i> の上三角 ( <i>uplo</i> ='U' の場合)、または下三角 ( <i>uplo</i> ='L' の場合) が対角を含めて破棄される。
----------	---

<i>b</i>	終了時に、 $info \leq n$ の場合、行列を格納している <i>b</i> の一部が、コレスキー因子分解 $B = U^T U$ または $B = L L^T$ からの三角係数 <i>U</i> または <i>L</i> で上書きされる。
<i>w</i>	REAL (ssygv の場合 ) DOUBLE PRECISION (dsygv の場合 )。 配列、次元は $\max(1, n)$ 以上。 $info = 0$ の場合、固有値が昇順で格納される。
<i>work(1)</i>	終了時に、 $info = 0$ の場合、 <i>work(1)</i> は <i>lwork</i> の必要最小サイズを返す。
<i>info</i>	INTEGER。 $info = 0$ の場合、実行は正常に終了したことを示す。 $info = -i$ の場合、 <i>i</i> 番目の引数が不正であったことを示す。 $info > 0$ の場合、spotrf/dpotrf と ssyev/dsyev によって、エラーコードが返される。  $info = i \leq n$ の場合、ssyev/dsyev が収束せず、中間三重対角の <i>i</i> 個の対角外の成分がゼロに収束しなかったことを示す。 $info = n + i$ ( $1 \leq i \leq n$ ) の場合、 <i>B</i> について先頭から次数 <i>i</i> の小行列が正定値でないことを示す。また、 <i>B</i> の因子分解が完了できず、固有値または固有ベクトルが計算されなかった。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン sygv のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ ( <i>n</i> , <i>n</i> ) の行列 <i>A</i> を格納する。
<i>b</i>	サイズ ( <i>n</i> , <i>n</i> ) の行列 <i>B</i> を格納する。
<i>w</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>itype</i>	1、2、または 3 でなければならない。デフォルト値は 1。
<i>jobz</i>	'N' または 'V' でなければならない。デフォルト値は 'N'。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。

### アプリケーション・ノート

最適なパフォーマンスを得るには、次の値を使用する。

$$lwork \geq (nb+2)*n$$

ここで、 $nb$  は、`ilaenv` が `ssytrd/dsytrd` に対して返したブロックサイズである。  
配列 `work` に必要なワークスペースの大きさがわからない場合は、最初の実行で `lwork` を大きめの値に設定する。終了時に、`work(1)` の値を調べ、以降はその値を使用する。

---

## ?hegv

複素数の汎用エルミート固有値問題について、  
固有値と (オプションで) 固有ベクトルをすべて  
計算する。

---

### 構文

#### Fortran 77:

```
call chegv(itype, jobz, uplo, n, a, lda, b, ldb, w, work, lwork, rwork, info)
call zhegv(itype, jobz, uplo, n, a, lda, b, ldb, w, work, lwork, rwork, info)
```

#### Fortran 95:

```
call hegv(a, b, w [,itype] [,jobz] [,uplo] [,info])
```

### 説明

このルーチンは、次の形式で示される複素数の汎用エルミート固有値問題について、固有値と (オプションで) 固有ベクトルをすべて計算する。

$$Ax = \lambda Bx, \quad ABx = \lambda x, \quad \text{または} \quad BAx = \lambda x$$

$A$  と  $B$  はエルミートであると仮定する。また、 $B$  は正定値である。

### 入力パラメータ

`itype`            INTEGER。1、2、または3のいずれかでなければならない。  
                  解決する問題のタイプを指定する。  
                  `itype = 1` の場合、問題のタイプは  $Ax = \lambda Bx$  である。  
                  `itype = 2` の場合、問題のタイプは  $ABx = \lambda x$  である。  
                  `itype = 3` の場合、問題のタイプは  $BAx = \lambda x$  である。

<i>jobz</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobz</i> ='N' の場合、固有値のみを計算する。 <i>jobz</i> ='V' の場合、固有値と固有ベクトルを計算する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> ='U' の場合、配列 <i>a</i> と <i>b</i> は、 <i>A</i> と <i>B</i> の上三角を格納する。 <i>uplo</i> ='L' の場合、配列 <i>a</i> と <i>b</i> は、 <i>A</i> と <i>B</i> の下三角を格納する。
<i>n</i>	INTEGER。行列 <i>A</i> と <i>B</i> の次数 ( $n \geq 0$ )。
<i>a</i> , <i>b</i> , <i>work</i>	COMPLEX ( <i>chegv</i> の場合 ) DOUBLE COMPLEX ( <i>zhegv</i> の場合 )。 配列： <i>a</i> ( <i>lda</i> ,*) には、 <i>uplo</i> の値に従って、エルミート行列 <i>A</i> の上三角または下三角を格納する。 <i>a</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。  <i>b</i> ( <i>ldb</i> ,*) には、 <i>uplo</i> の値に従って、エルミート正定値行列 <i>B</i> の上三角または下三角を格納する。 <i>b</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。  <i>work</i> ( <i>lwork</i> ) は、ワークスペース配列である。
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $\max(1, n)$ 以上。
<i>ldb</i>	INTEGER。 <i>b</i> の第 1 次元。 $\max(1, n)$ 以上。
<i>lwork</i>	INTEGER。 配列 <i>work</i> の次元。 $lwork \geq \max(1, 2n-1)$ 。 <i>lwork</i> = -1 の場合、ワークスペースのクエリとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリとして返す。 <i>xerbla</i> は <i>lwork</i> に関するエラー・メッセージを生成しない。  <i>lwork</i> の推奨値は、「アプリケーション・ノート」を参照。
<i>rwork</i>	REAL ( <i>chegv</i> の場合 ) DOUBLE PRECISION ( <i>zhegv</i> の場合 )。 ワークスペース配列、次元は $\max(1, 3n-2)$ 以上。

## 出力パラメータ

<i>a</i>	終了時に、 <i>jobz</i> ='V' の場合、 <i>info</i> =0 であれば、 <i>a</i> には、固有ベクトルの行列 <i>Z</i> が格納される。固有ベクトルは、次のように正規化される。 $itype = 1$ または $2$ の場合、 $Z^H B Z = I$ 。 $itype = 3$ の場合、 $Z^H B^{-1} Z = I$ 。 <i>jobz</i> ='N' の場合、終了時に、 <i>A</i> の上三角 ( <i>uplo</i> ='U' の場合)、または下三角 ( <i>uplo</i> ='L' の場合) が対角を含めて破棄される。
<i>b</i>	終了時に、 <i>info</i> ≤ <i>n</i> の場合、行列を格納している <i>b</i> の一部が、コレスキー因子分解 $B = U^H U$ または $B = L L^H$ からの三角係数 <i>U</i> または <i>L</i> で上書きされる。
<i>w</i>	REAL ( <i>chegv</i> の場合) DOUBLE PRECISION ( <i>zhegv</i> の場合)。 配列、次元は max(1, <i>n</i> ) 以上。 <i>info</i> = 0 の場合、固有値が昇順で格納される。
<i>work(1)</i>	終了時に、 <i>info</i> = 0 の場合、 <i>work(1)</i> は <i>lwork</i> の必要最小サイズを返す。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目の引数が不正であったことを示す。 <i>info</i> > 0 の場合、 <i>cpotrf/zpotrf</i> と <i>cheev/zheev</i> によって、エラーコードが返される。  <i>info</i> = <i>i</i> ≤ <i>n</i> の場合、 <i>cheev/zheev</i> が収束せず、中間三重対角の <i>i</i> 個の対角外の成分がゼロに収束しなかったことを示す。 <i>info</i> = <i>n</i> + <i>i</i> (1 ≤ <i>i</i> ≤ <i>n</i> ) の場合、 <i>B</i> について先頭から次数 <i>i</i> の小行列が正定値でないことを示す。また、 <i>B</i> の因子分解が完了できず、固有値または固有ベクトルが計算されなかった。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *hegv* のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ ( <i>n</i> , <i>n</i> ) の行列 <i>A</i> を格納する。
<i>b</i>	サイズ ( <i>n</i> , <i>n</i> ) の行列 <i>B</i> を格納する。

*w*                   長さ (*n*) のベクトルを格納する。

*itype*               1、2、または3 でなければならない。デフォルト値は 1。

*jobz*               'N' または 'V' でなければならない。デフォルト値は 'N'。

*uplo*               'U' または 'L' でなければならない。デフォルト値は 'U'。

### アプリケーション・ノート

最適なパフォーマンスを得るには、次の値を使用する。

$$lwork \geq (nb+1)*n$$

ここで、*nb* は、*ilaenv* が *chetrd/zhetrd* に対して返したブロックサイズである。  
配列 *work* に必要なワークスペースの大きさがわからない場合は、最初の実行で *lwork* を大きめの値に設定する。終了時に、*work(1)* の値を調べ、以降はその値を使用する。

---

## ?sygvd

実数の汎用対称固有値問題について、固有値と  
( オプションで ) 固有ベクトルをすべて計算する。  
固有ベクトルを計算する場合は、分割統治法を  
使用する。

---

### 構文

#### Fortran 77:

```
call ssygvd(itype, jobz, uplo, n, a, lda, b, ldb, w, work, lwork, iwork, liwork,
            info)
call dsygvd(itype, jobz, uplo, n, a, lda, b, ldb, w, work, lwork, iwork, liwork,
            info)
```

#### Fortran 95:

```
call sygvd(a, b, w [,itype] [,jobz] [,uplo] [,info])
```

### 説明

このルーチンは、次の形式で示される実数の汎用対称固有値問題について、固有値と  
( オプションで ) 固有ベクトルをすべて計算する。

$$Ax = \lambda Bx, ABx = \lambda x, \text{ または } BAx = \lambda x$$

*A* と *B* は対称であると仮定する。また、*B* は正定値である。

固有ベクトルを計算する場合は、分割統治法を使用する。

## 入力パラメータ

<i>itype</i>	INTEGER。1、2、または3のいずれかでなければならない。 解決する問題のタイプを指定する。 <i>itype</i> = 1 の場合、問題のタイプは $Ax = \lambda Bx$ である。 <i>itype</i> = 2 の場合、問題のタイプは $ABx = \lambda x$ である。 <i>itype</i> = 3 の場合、問題のタイプは $B Ax = \lambda x$ である。
<i>jobz</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobz</i> = 'N' の場合、固有値のみを計算する。 <i>jobz</i> = 'V' の場合、固有値と固有ベクトルを計算する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> = 'U' の場合、配列 <i>a</i> と <i>b</i> は、 <i>A</i> と <i>B</i> の上三角を格納する。 <i>uplo</i> = 'L' の場合、配列 <i>a</i> と <i>b</i> は、 <i>A</i> と <i>B</i> の下三角を格納する。
<i>n</i>	INTEGER。行列 <i>A</i> と <i>B</i> の次数 ( $n \geq 0$ )。
<i>a</i> , <i>b</i> , <i>work</i>	REAL (ssygvd の場合 ) DOUBLE PRECISION (dsygvd の場合 )。 配列: <i>a</i> ( <i>lda</i> ,*) には、 <i>uplo</i> の値に従って、対称行列 <i>A</i> の上三角または下三角を格納する。 <i>a</i> の第2次元は、 $\max(1, n)$ 以上でなければならない。  <i>b</i> ( <i>ldb</i> ,*) には、 <i>uplo</i> の値に従って、対称正定値行列 <i>B</i> の上三角または下三角を格納する。 <i>b</i> の第2次元は、 $\max(1, n)$ 以上でなければならない。  <i>work</i> ( <i>lwork</i> ) は、ワークスペース配列である。
<i>lda</i>	INTEGER。 <i>a</i> の第1次元。 $\max(1, n)$ 以上。
<i>ldb</i>	INTEGER。 <i>b</i> の第1次元。 $\max(1, n)$ 以上。
<i>lwork</i>	INTEGER。配列 <i>work</i> の次元。  次の制約がある。 $n \leq 1$ の場合、 $lwork \geq 1$ 。 <i>jobz</i> = 'N' かつ $n > 1$ の場合、 $lwork \geq 2n+1$ 。 <i>jobz</i> = 'V' で $n > 1$ の場合、 $lwork \geq 2n^2+6n+1$ 。 <i>lwork</i> = -1 の場合、ワークスペースのクエリとみなされ、ルーチンは



*work* 配列の最適サイズだけを計算し、その値を *work* 配列の最初のエントリとして返す。xerbla は *lwork* に関するエラー・メッセージを生成しない。

*iwork* INTEGER。  
ワークスペース配列、次元は (*liwork*)。

*liwork* INTEGER。配列 *iwork* の次元。  
次の制約がある。  
 $n \leq 1$  の場合、 $liwork \geq 1$ 。  
 $jobz = 'N'$  で  $n > 1$  の場合、 $liwork \geq 1$ 。  
 $jobz = 'V'$  で  $n > 1$  の場合、 $liwork \geq 5n+3$ 。

*liwork* = -1 の場合、ワークスペースのクエリとみなされ、ルーチンは *iwork* 配列の最適サイズだけを計算し、その値を *iwork* 配列の最初のエントリとして返す。xerbla は *liwork* に関するエラー・メッセージを生成しない。

### 出力パラメータ

*a* 終了時に、 $jobz = 'V'$  の場合、 $info = 0$  であれば、*a* には、固有ベクトルの行列 *Z* が格納される。固有ベクトルは、次のように正規化される。

$itype = 1$  または  $2$  の場合、 $Z^T B Z = I$ 。  
 $itype = 3$  の場合、 $Z^T B^{-1} Z = I$ 。

$jobz = 'N'$  の場合、終了時に、*A* の上三角 ( $uplo = 'U'$  の場合)、または下三角 ( $uplo = 'L'$  の場合) が対角を含めて破棄される。

*b* 終了時に、 $info \leq n$  の場合、行列を格納している *b* の一部が、コレスキー因子分解  $B = U^T U$  または  $B = L L^T$  からの三角係数 *U* または *L* で上書きされる。

*w* REAL (ssygvd の場合 )  
DOUBLE PRECISION (dsygvd の場合 )。  
配列、次元は  $\max(1, n)$  以上。  
 $info = 0$  の場合、固有値が昇順で格納される。

*work(1)* 終了時に、 $info = 0$  の場合、*work(1)* は *lwork* の必要最小サイズを返す。

*iwork(1)* 終了時に、 $info = 0$  の場合、*iwork(1)* は *liwork* の必要最小サイズを返す。

*info* INTEGER。  
*info* = 0 の場合、実行は正常に終了したことを示す。  
*info* = -*i* の場合、*i* 番目の引数が不正であったことを示す。  
*info* > 0 の場合、spotrf/dpotrf と ssyev/dsyev によって、エラーコードが返される。  
  
*info* = *i* ≤ *n* の場合、ssyev/dsyev が収束せず、中間三重対角の *i* 個の対角外の成分がゼロに収束しなかったことを示す。  
*info* = *n* + *i* (1 ≤ *i* ≤ *n*) の場合、*B* について先頭から次数 *i* の小行列が正定値でないことを示す。また、*B* の因子分解が完了できず、固有値または固有ベクトルが計算されなかった。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン sygvd のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ ( <i>n</i> , <i>n</i> ) の行列 <i>A</i> を格納する。
<i>b</i>	サイズ ( <i>n</i> , <i>n</i> ) の行列 <i>B</i> を格納する。
<i>w</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>itype</i>	1、2、または 3 でなければならない。デフォルト値は 1。
<i>jobz</i>	'N' または 'V' でなければならない。デフォルト値は 'N'。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。

## ?hegvd

複素数の汎用エルミート固有値問題について、固有値と (オプションで) 固有ベクトルをすべて計算する。固有ベクトルを計算する場合は、分割統治法を使用する。

### 構文

#### Fortran 77:

```
call chegvd(itype, jobz, uplo, n, a, lda, b, ldb, w, work, lwork, rwork, lrwork,  
            iwork, liwork, info)  
call zhegvd(itype, jobz, uplo, n, a, lda, b, ldb, w, work, lwork, rwork, lrwork,  
            iwork, liwork, info)
```

#### Fortran 95:

```
call hegvd(a, b, w [,itype] [,jobz] [,uplo] [,info])
```

### 説明

このルーチンは、次の形式で示される複素数の汎用エルミート固有値問題について、固有値と (オプションで) 固有ベクトルをすべて計算する。

$$Ax = \lambda Bx, \quad ABx = \lambda x, \quad \text{または} \quad BAx = \lambda x$$

$A$  と  $B$  はエルミートであると仮定する。また、 $B$  は正定値である。  
固有ベクトルを計算する場合は、分割統治法を使用する。

### 入力パラメータ

<i>itype</i>	INTEGER。1、2、または3のいずれかでなければならない。 解決する問題のタイプを指定する。 $itype = 1$ の場合、問題のタイプは $Ax = \lambda Bx$ である。 $itype = 2$ の場合、問題のタイプは $ABx = \lambda x$ である。 $itype = 3$ の場合、問題のタイプは $BAx = \lambda x$ である。
<i>jobz</i>	CHARACTER*1。'N' または 'V' でなければならない。 $jobz = 'N'$ の場合、固有値のみを計算する。 $jobz = 'V'$ の場合、固有値と固有ベクトルを計算する。

<i>uplo</i>	<p>CHARACTER*1。'U' または 'L' でなければならない。  <i>uplo</i>='U' の場合、配列 <i>a</i> と <i>b</i> は、<i>A</i> と <i>B</i> の上三角を格納する。  <i>uplo</i>='L' の場合、配列 <i>a</i> と <i>b</i> は、<i>A</i> と <i>B</i> の下三角を格納する。</p>
<i>n</i>	<p>INTEGER。行列 <i>A</i> と <i>B</i> の次数 (<math>n \geq 0</math>)。</p>
<i>a</i> , <i>b</i> , <i>work</i>	<p>COMPLEX (chegvd の場合 )  DOUBLE COMPLEX (zhegvd の場合 )。  配列:  <i>a</i>(<i>lda</i>,*) には、<i>uplo</i> の値に従って、エルミート行列 <i>A</i> の上三角または下三角を格納する。  <i>a</i> の第 2 次元は、<math>\max(1, n)</math> 以上でなければならない。    <i>b</i>(<i>ldb</i>,*) には、<i>uplo</i> の値に従って、エルミート正定値行列 <i>B</i> の上三角または下三角を格納する。  <i>b</i> の第 2 次元は、<math>\max(1, n)</math> 以上でなければならない。    <i>work</i>(<i>lwork</i>) は、ワークスペース配列である。</p>
<i>lda</i>	<p>INTEGER。 <i>a</i> の第 1 次元。 <math>\max(1, n)</math> 以上。</p>
<i>ldb</i>	<p>INTEGER。 <i>b</i> の第 1 次元。 <math>\max(1, n)</math> 以上。</p>
<i>lwork</i>	<p>INTEGER。 配列 <i>work</i> の次元。  次の制約がある。  <math>n \leq 1</math> の場合、<math>lwork \geq 1</math>。  <i>jobz</i>='N' で <math>n &gt; 1</math> の場合、<math>lwork \geq n+1</math>。  <i>jobz</i>='V' で <math>n &gt; 1</math> の場合、<math>lwork \geq n^2+2n</math>。    <i>lwork</i> = -1 の場合、ワークスペースのクエリとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリとして返す。xerbla は <i>lwork</i> に関するエラー・メッセージを生成しない。</p>
<i>rwork</i>	<p>REAL (chegvd の場合 )  DOUBLE PRECISION (zhegvd の場合 )。  ワークスペース配列、次元は (<i>lrwork</i>)。</p>
<i>lrwork</i>	<p>INTEGER。 配列 <i>rwork</i> の次元。  次の制約がある。  <math>n \leq 1</math> の場合、<math>lrwork \geq 1</math>。  <i>jobz</i>='N' で <math>n &gt; 1</math> の場合、<math>lrwork \geq n</math>。  <i>jobz</i>='V' で <math>n &gt; 1</math> の場合、<math>lrwork \geq 2n^2+5n+1</math>。</p>

$lrwork = -1$  の場合、ワークスペースのクエリとみなされ、ルーチンは  $rwork$  配列の最適サイズだけを計算し、その値を  $rwork$  配列の最初のエントリとして返す。xerbla は  $lrwork$  に関するエラー・メッセージを生成しない。

$iwork$  INTEGER。  
ワークスペース配列、次元は ( $liwork$ )。

$liwork$  INTEGER。配列  $iwork$  の次元。  
次の制約がある。  
 $n \leq 1$  の場合、 $liwork \geq 1$ 。  
 $jobz = 'N'$  で  $n > 1$  の場合、 $liwork \geq 1$ 。  
 $jobz = 'V'$  で  $n > 1$  の場合、 $liwork \geq 5n+3$ 。

$liwork = -1$  の場合、ワークスペースのクエリとみなされ、ルーチンは  $iwork$  配列の最適サイズだけを計算し、その値を  $iwork$  配列の最初のエントリとして返す。xerbla は  $liwork$  に関するエラー・メッセージを生成しない。

### 出力パラメータ

$a$  終了時に、 $jobz = 'V'$  の場合、 $info = 0$  であれば、 $a$  には、固有ベクトルの行列  $Z$  が格納される。固有ベクトルは、次のように正規化される。  
 $itype = 1$  または  $2$  の場合、 $Z^H B Z = I$ 。  
 $itype = 3$  の場合、 $Z^H B^{-1} Z = I$ 。  
 $jobz = 'N'$  の場合、終了時に、 $A$  の上三角 ( $uplo = 'U'$  の場合)、または下三角 ( $uplo = 'L'$  の場合) が対角を含めて破棄される。

$b$  終了時に、 $info \leq n$  の場合、行列を格納している  $b$  の一部が、コレスキー因子分解  $B = U^H U$  または  $B = L L^H$  からの三角係数  $U$  または  $L$  で上書きされる。

$w$  REAL (chegvd の場合 )  
DOUBLE PRECISION (zhegvd の場合 )。  
配列、次元は  $\max(1, n)$  以上。  
 $info = 0$  の場合、固有値が昇順で格納される。

$work(1)$  終了時に、 $info = 0$  の場合、 $work(1)$  は、 $lwork$  の必要最小サイズを返す。

$rwork(1)$  終了時に、 $info = 0$  の場合、 $rwork(1)$  は、 $lrwork$  の必要最小サイズを返す。

<i>iwork(1)</i>	終了時に、 <i>info</i> = 0 の場合、 <i>iwork(1)</i> は <i>liwork</i> の必要最小サイズを返す。
<i>info</i>	<p>INTEGER。</p> <p><i>info</i> = 0 の場合、実行は正常に終了したことを示す。</p> <p><i>info</i> = -<i>i</i> の場合、<i>i</i> 番目の引数が不正であったことを示す。</p> <p><i>info</i> &gt; 0 の場合、<i>cpotrf/zpotrf</i> と <i>cheev/zheev</i> によって、エラーコードが返される。</p> <p><i>info</i> = <i>i</i> ≤ <i>n</i> の場合、<i>cheev/zheev</i> が収束せず、中間三重対角の <i>i</i> 個の対角外の成分がゼロに収束しなかったことを示す。</p> <p><i>info</i> = <i>n</i> + <i>i</i> (1 ≤ <i>i</i> ≤ <i>n</i>) の場合、<i>B</i> について先頭から次数 <i>i</i> の小行列が正定値でないことを示す。また、<i>B</i> の因子分解が完了できず、固有値または固有ベクトルが計算されなかった。</p>

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *hegv*d のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ ( <i>n</i> , <i>n</i> ) の行列 <i>A</i> を格納する。
<i>b</i>	サイズ ( <i>n</i> , <i>n</i> ) の行列 <i>B</i> を格納する。
<i>w</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>itype</i>	1、2、または 3 でなければならない。デフォルト値は 1。
<i>jobz</i>	'N' または 'V' でなければならない。デフォルト値は 'N'。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。

## ?sygvx

実数の汎用対称固有値問題について、固有値と (オプションで) 固有ベクトルを選択的に計算する。

### 構文

#### Fortran 77:

```
call ssygvx(itype, jobz, range, uplo, n, a, lda, b, ldb, vl, vu, il, iu, abstol,
            m, w, z, ldz, work, lwork, iwork, ifail, info)
call dsygvx(itype, jobz, range, uplo, n, a, lda, b, ldb, vl, vu, il, iu, abstol,
            m, w, z, ldz, work, lwork, iwork, ifail, info)
```

#### Fortran 95:

```
call sygvx(a, b, w [,itype] [,uplo] [,z] [,vl] [,vu] [,il] [,iu] [,m] [,ifail]
           [,abstol] [,info])
```

### 説明

このルーチンは、次の形式で示される実数の汎用対称固有値問題について、固有値と (オプションで) 固有ベクトルを選択的に計算する。

$$Ax = \lambda Bx, ABx = \lambda x, \text{ または } BAx = \lambda x$$

$A$  と  $B$  は対称であると仮定する。また、 $B$  は正定値である。

固有値と固有ベクトルを選択するには、希望する固有値の値の範囲かインデックスの範囲を指定する。

### 入力パラメータ

<i>itype</i>	INTEGER。1、2、または3のいずれかでなければならない。 解決する問題のタイプを指定する。 <i>itype</i> = 1 の場合、問題のタイプは $Ax = \lambda Bx$ である。 <i>itype</i> = 2 の場合、問題のタイプは $ABx = \lambda x$ である。 <i>itype</i> = 3 の場合、問題のタイプは $BAx = \lambda x$ である。
<i>jobz</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobz</i> = 'N' の場合、固有値のみを計算する。 <i>jobz</i> = 'V' の場合、固有値と固有ベクトルを計算する。
<i>range</i>	CHARACTER*1。'A'、'V'、または 'I' のいずれかでなければならない。

	<p><math>range = 'A'</math> の場合、固有値をすべて計算する。</p> <p><math>range = 'V'</math> の場合、半開区間 <math>v1 &lt; \lambda_i \leq vu</math> における固有値 <math>\lambda_i</math> を計算する。</p> <p><math>range = 'I'</math> の場合、ルーチンはインデックスが <math>i1 \sim iu</math> である固有値を計算する。</p>
<i>uplo</i>	<p>CHARACTER*1。'U' または 'L' でなければならない。</p> <p><math>uplo = 'U'</math> の場合、配列 <i>a</i> と <i>b</i> は、<i>A</i> と <i>B</i> の上三角を格納する。</p> <p><math>uplo = 'L'</math> の場合、配列 <i>a</i> と <i>b</i> は、<i>A</i> と <i>B</i> の下三角を格納する。</p>
<i>n</i>	INTEGER。行列 <i>A</i> と <i>B</i> の次数 ( $n \geq 0$ )。
<i>a, b, work</i>	<p>REAL (ssygvd の場合)</p> <p>DOUBLE PRECISION (dsygvd の場合)。</p> <p>配列:</p> <p><math>a(lda,*)</math> には、<i>uplo</i> の値に従って、対称行列 <i>A</i> の上三角または下三角を格納する。</p> <p><i>a</i> の第 2 次元は、<math>\max(1, n)</math> 以上でなければならない。</p> <p><math>b(l db,*)</math> には、<i>uplo</i> の値に従って、対称正定値行列 <i>B</i> の上三角または下三角を格納する。</p> <p><i>b</i> の第 2 次元は、<math>\max(1, n)</math> 以上でなければならない。</p> <p><math>work(lwork)</math> は、ワークスペース配列である。</p>
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $\max(1, n)$ 以上。
<i>ldb</i>	INTEGER。 <i>b</i> の第 1 次元。 $\max(1, n)$ 以上。
<i>v1, vu</i>	<p>REAL (ssygvx の場合)</p> <p>DOUBLE PRECISION (dsygvx の場合)。</p> <p><math>range = 'V'</math> の場合、固有値を検索する区間の上限と下限。次の制約がある。 <math>v1 &lt; vu</math>。</p> <p><math>range = 'A'</math> または 'I' の場合、<i>v1</i> と <i>vu</i> は参照されない。</p>
<i>i1, iu</i>	<p>INTEGER。</p> <p><math>range = 'I'</math> の場合、返される最小固有値と最大固有値に対する昇順のインデックス。</p> <p>次の制約がある。</p> <p><math>1 \leq i1 \leq iu \leq n</math> (<math>n &gt; 0</math> の場合)。</p> <p><math>i1=1</math> かつ <math>iu=0</math> (<math>n=0</math> の場合)。</p> <p><math>range = 'A'</math> または 'V' の場合、<i>i1</i> と <i>iu</i> は参照されない。</p>



<i>abstol</i>	REAL (ssygvx の場合 ) DOUBLE PRECISION (dsygvx の場合 )。 固有値に対する絶対エラー許容値。 詳細は、「アプリケーション・ノート」を参照。
<i>ldz</i>	INTEGER。出力配列 <i>z</i> のリーディング・ディメンジョン。 次の制約がある。 $ldz \geq 1$ 。 <i>jobz</i> = 'V' の場合、 $ldz \geq \max(1, n)$ 。
<i>lwork</i>	INTEGER。配列 <i>work</i> の次元。 $lwork \geq \max(1, 8n)$ 。 <i>lwork</i> = -1 の場合、ワークスペースのクエリとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリとして返す。xerbla は <i>lwork</i> に関するエラー・メッセージを生成しない。  <i>lwork</i> の推奨値は、「アプリケーション・ノート」を参照。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, 5n)$ 以上。

### 出力パラメータ

<i>a</i>	終了時に、 <i>A</i> の上三角 ( <i>uplo</i> = 'U' の場合 )、または下三角 ( <i>uplo</i> = 'L' の場合 ) が対角を含めて上書きされる。
<i>b</i>	終了時に、 $info \leq n$ の場合、行列を格納している <i>b</i> の一部が、コレスキー因子分解 $B = U^T U$ または $B = L L^T$ からの三角係数 <i>U</i> または <i>L</i> で上書きされる。
<i>m</i>	INTEGER。検出された固有値の総数。 ( $0 \leq m \leq n$ )。 <i>range</i> = 'A' の場合、 $m = n$ 、 <i>range</i> = 'I' の場合、 $m = iu - il + 1$ 。
<i>w, z</i>	REAL (ssygvx の場合 ) DOUBLE PRECISION (dsygvx の場合 )。 配列： <i>w</i> (*)、次元は $\max(1, n)$ 以上。 <i>w</i> の先頭から <i>m</i> 個の成分には、指定された固有値が昇順で格納される。  <i>z</i> ( <i>ldz</i> ,*)。 <i>z</i> の第 2 次元は、 $\max(1, m)$ 以上でなければならない。 <i>jobz</i> = 'V' の場合、 $info = 0$ であれば、 <i>z</i> の先頭の <i>m</i> 列に、行列 <i>A</i> の選択された固有値に対応する正規直交固有ベクトルが格納される。すなわち、 <i>w</i> ( <i>i</i> ) に対応する固有ベクトルが、 <i>z</i> の <i>i</i> 番目の列に格納される。

固有ベクトルは、次のように正規化される。

$itype = 1$  または  $2$  の場合、 $Z^T B Z = I$ 。

$itype = 3$  の場合、 $Z^T B^{-1} Z = I$ 。

$jobz = 'N'$  の場合、 $z$  は参照されない。

固有ベクトルが収束できない場合、 $z$  のその列には、その固有ベクトルに対する最新の近似値が格納され、固有ベクトルのインデックスは  $ifail$  に返される。

注：配列  $z$  には、 $\max(1, m)$  以上の列が提供されなければならない。

$range = 'V'$  の場合、 $m$  の正確な値が事前にわからないため、上限値を使用する必要がある。

$work(1)$  終了時に、 $info = 0$  の場合、 $work(1)$  は  $lwork$  の必要最小サイズを返す。

$ifail$  INTEGER。

配列、次元は  $\max(1, n)$  以上。

$jobz = 'V'$  の場合、 $info = 0$  であれば、 $ifail$  の先頭から  $m$  個の成分はゼロになる。 $info > 0$  であれば、収束しなかった固有ベクトルのインデックスが  $ifail$  に格納される。

$jobz = 'N'$  の場合、 $ifail$  は参照されない。

$info$  INTEGER。

$info = 0$  の場合、実行は正常に終了したことを示す。

$info = -i$  の場合、 $i$  番目の引数が不正であったことを示す。

$info > 0$  の場合、 $spotrf/dpotrf$  と  $ssyevx/dsyevx$  によって、エラーコードが返される。

$info = i \leq n$  の場合、 $ssyevx/dsyevx$  が収束せず、 $i$  個の固有ベクトルが収束しなかったことを示す。これらのインデックスは、配列  $ifail$  に格納される。

$info = n + i$  ( $1 \leq i \leq n$ ) の場合、 $B$  について先頭から次数  $i$  の小行列が正定値でないことを示す。また、 $B$  の因子分解が完了できず、固有値または固有ベクトルが計算されなかった。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `sygvx` のインターフェイスの詳細を以下に示す。

$a$                       サイズ  $(n, n)$  の行列  $A$  を格納する。

<i>b</i>	サイズ $(n,n)$ の行列 <i>B</i> を格納する。
<i>w</i>	長さ $(n)$ のベクトルを格納する。
<i>z</i>	サイズ $(n,m)$ の行列 <i>Z</i> を格納する。ここで、値 <i>n</i> と <i>m</i> は有意である。
<i>ifail</i>	長さ $(n)$ のベクトルを格納する。
<i>itype</i>	1、2、または 3 でなければならない。デフォルト値は 1。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>v1</i>	この成分のデフォルト値は、 $v1 = -HUGE(v1)$ である。
<i>vu</i>	この成分のデフォルト値は、 $vu = HUGE(v1)$ である。
<i>il</i>	この引数のデフォルト値は、 $il = 1$ である。
<i>iu</i>	この引数のデフォルト値は、 $iu = n$ である。
<i>abstol</i>	この成分のデフォルト値は、 $abstol = 0.0\_WP$ である。
<i>jobz</i>	引数 <i>z</i> の存在に基づいて以下のように復元される。 <i>jobz</i> = 'V' ( <i>z</i> が存在する場合)、 <i>jobz</i> = 'N' ( <i>z</i> が省略された場合)。 <i>ifail</i> が存在し、 <i>z</i> が省略された場合、エラー条件がセットされる。
<i>range</i>	引数 <i>v1</i> 、 <i>vu</i> 、 <i>il</i> 、および <i>iu</i> の存在に基づいて以下のように復元される。 <i>range</i> = 'V' ( <i>v1</i> と <i>vu</i> のいずれかまたは両方が存在する場合)、 <i>range</i> = 'I' ( <i>il</i> と <i>iu</i> のいずれかまたは両方が存在する場合)、 <i>range</i> = 'A' ( <i>v1</i> 、 <i>vu</i> 、 <i>il</i> 、および <i>iu</i> がすべて存在しない場合)。 <i>v1</i> と <i>vu</i> のいずれかまたは両方が存在し、同時に <i>il</i> と <i>iu</i> のいずれかまたは両方が存在する場合、エラー条件がセットされる。

### アプリケーション・ノート

$abstol + \varepsilon * \max(|a|, |b|)$  以下の幅の区間  $[a, b]$  内に存在していると判別された場合 ( $\varepsilon$  はマシン精度)、近似固有値は、収束したものとして受け入れられる。*abstol* がゼロ以下の場合、 $\varepsilon * \|T\|_1$  は所定の場所で使用される (*T* は、*A* を三重対角形式に縮退して獲得した三重対角行列)。

固有値が最も正確に計算されるのは、*abstol* がゼロでなく、アンダーフローのしきい値の 2 倍、 $2 * ?lamch('S')$  に設定されたときである。このルーチンで *info* > 0 が返された場合、固有ベクトルのいくつかは収束に失敗したことを意味するので、*abstol* を  $2 * ?lamch('S')$  に設定して、やり直してみる。

最適なパフォーマンスを得るには、次の値を使用する。

$$lwork \geq (nb+3)*n$$

ここで、 $nb$  は、`ilaenv` が `ssytrd/dsytrd` に対して返したブロックサイズである。  
配列 `work` に必要なワークスペースの大きさがわからない場合は、最初の実行で `lwork` を大きめの値に設定する。終了時に、`work(1)` の値を調べ、以降はその値を使用する。

---

### ?hegvx

複素数の汎用エルミート固有値問題について、  
固有値と (オプションで) 固有ベクトルを選択的に  
計算する。

---

#### 構文

##### Fortran 77:

```
call chegvx(itype, jobz, range, uplo, n, a, lda, b, ldb, vl, vu, il, iu, abstol,  
            m, w, z, ldz, work, lwork, rwork, iwork, ifail, info)  
call zhegvx(itype, jobz, range, uplo, n, a, lda, b, ldb, vl, vu, il, iu, abstol,  
            m, w, z, ldz, work, lwork, rwork, iwork, ifail, info)
```

##### Fortran 95:

```
call hegvx(a, b, w [,itype] [,uplo] [,z] [,vl] [,vu] [,il] [,iu] [,m] [,ifail]  
            [,abstol] [,info])
```

#### 説明

このルーチンは、次の形式で示される複素数の汎用エルミート固有値問題について、固有値と (オプションで) 固有ベクトルを選択的に計算する。

$$Ax = \lambda Bx, ABx = \lambda x, \text{ または } BAx = \lambda x$$

$A$  と  $B$  はエルミートであると仮定する。また、 $B$  は正定値である。  
固有値と固有ベクトルを選択するには、希望する固有値の値の範囲かインデックスの範囲を指定する。

**入力パラメータ**

<i>itype</i>	INTEGER。1、2、または3のいずれかでなければならない。 解決する問題のタイプを指定する。 <i>itype</i> = 1 の場合、問題のタイプは $Ax = \lambda Bx$ である。 <i>itype</i> = 2 の場合、問題のタイプは $ABx = \lambda x$ である。 <i>itype</i> = 3 の場合、問題のタイプは $BAx = \lambda x$ である。
<i>jobz</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobz</i> = 'N' の場合、固有値のみを計算する。 <i>jobz</i> = 'V' の場合、固有値と固有ベクトルを計算する。
<i>range</i>	CHARACTER*1。'A'、'V'、または 'I' のいずれかでなければならない。 <i>range</i> = 'A' の場合、固有値をすべて計算する。 <i>range</i> = 'V' の場合、半開区間 $v_l < \lambda_i \leq v_u$ における固有値 $\lambda_i$ を計算する。 <i>range</i> = 'I' の場合、ルーチンはインデックスが $i_l \sim i_u$ である固有値を計算する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> = 'U' の場合、配列 <i>a</i> と <i>b</i> は、 <i>A</i> と <i>B</i> の上三角を格納する。 <i>uplo</i> = 'L' の場合、配列 <i>a</i> と <i>b</i> は、 <i>A</i> と <i>B</i> の下三角を格納する。
<i>n</i>	INTEGER。行列 <i>A</i> と <i>B</i> の次数 ( $n \geq 0$ )。
<i>a</i> , <i>b</i> , <i>work</i>	COMPLEX (chegvx の場合) DOUBLE COMPLEX (zhegvx の場合)。 配列： <i>a</i> ( <i>lda</i> ,*) には、 <i>uplo</i> の値に従って、エルミート行列 <i>A</i> の上三角または下三角を格納する。 <i>a</i> の第2次元は、 $\max(1, n)$ 以上でなければならない。 <i>b</i> ( <i>ldb</i> ,*) には、 <i>uplo</i> の値に従って、エルミート正定値行列 <i>B</i> の上三角または下三角を格納する。 <i>b</i> の第2次元は、 $\max(1, n)$ 以上でなければならない。 <i>work</i> ( <i>lwork</i> ) は、ワークスペース配列である。
<i>lda</i>	INTEGER。 <i>a</i> の第1次元。 $\max(1, n)$ 以上。
<i>ldb</i>	INTEGER。 <i>b</i> の第1次元。 $\max(1, n)$ 以上。

<i>vl, vu</i>	<p>REAL (chegvx の場合 )  DOUBLE PRECISION (zhegvx の場合 )。  <code>range = 'V'</code> の場合、固有値を検索する区間の上限と下限。  次の制約がある。 <math>vl &lt; vu</math>。</p> <p><code>range = 'A'</code> または <code>'I'</code> の場合、<i>vl</i> と <i>vu</i> は参照されない。</p>
<i>il, iu</i>	<p>INTEGER。  <code>range = 'I'</code> の場合、返される最小固有値と最大固有値に対する昇順のインデックス。  次の制約がある。  <math>1 \leq il \leq iu \leq n</math> (<math>n &gt; 0</math> の場合 )。  <math>il=1</math> かつ <math>iu=0</math> (<math>n = 0</math> の場合 )。</p> <p><code>range = 'A'</code> または <code>'V'</code> の場合、<i>il</i> と <i>iu</i> は参照されない。</p>
<i>abstol</i>	<p>REAL (chegvx の場合 )  DOUBLE PRECISION (zhegvx の場合 )。  固有値に対する絶対エラー許容値。  詳細は、「アプリケーション・ノート」を参照。</p>
<i>ldz</i>	<p>INTEGER。出力配列 <i>z</i> のリーディング・ディメンション。  次の制約がある。  <math>ldz \geq 1</math>。  <code>jobz = 'V'</code> の場合、<math>ldz \geq \max(1, n)</math>。</p>
<i>lwork</i>	<p>INTEGER。配列 <i>work</i> の次元。  <math>lwork \geq \max(1, 2n-1)</math>。  <math>lwork = -1</math> の場合、ワークスペースのクエリとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリとして返す。xerbla は <i>lwork</i> に関するエラー・メッセージを生成しない。</p> <p><i>lwork</i> の推奨値は、「アプリケーション・ノート」を参照。</p>
<i>rwork</i>	<p>REAL (chegvx の場合 )  DOUBLE PRECISION (zhegvx の場合 )。  ワークスペース配列、次元は <math>\max(1, 7n)</math> 以上。</p>
<i>iwork</i>	<p>INTEGER。  ワークスペース配列、次元は <math>\max(1, 5n)</math> 以上。</p>

## 出力パラメータ

<i>a</i>	<p>終了時に、<i>A</i> の上三角 (<code>uplo = 'U'</code> の場合 )、または下三角 (<code>uplo = 'L'</code> の場合 ) が対角を含めて上書きされる。</p>
----------	---

<i>b</i>	終了時に、 $info \leq n$ の場合、行列を格納している <i>b</i> の一部が、コレスキー因子分解 $B = U^H U$ または $B = L L^H$ からの三角係数 <i>U</i> または <i>L</i> で上書きされる。
<i>m</i>	INTEGER。検出された固有値の総数。 $(0 \leq m \leq n)$ 。 $range = 'A'$ の場合、 $m = n$ 、 $range = 'I'$ の場合、 $m = iu - il + 1$ 。
<i>w</i>	REAL (chegvx の場合 ) DOUBLE PRECISION (zhegvx の場合 )。 配列、次元は $\max(1, n)$ 以上。 <i>w</i> の先頭から <i>m</i> 個の成分には、指定された固有値が昇順で格納される。
<i>z</i>	COMPLEX (chegvx の場合 ) DOUBLE COMPLEX (zhegvx の場合 )。 配列 $z(ldz, *)$ 。 <i>z</i> の第 2 次元は、 $\max(1, m)$ 以上でなければならない。  $jobz = 'V'$ の場合、 $info = 0$ であれば、 <i>z</i> の先頭の <i>m</i> 列に、行列 <i>A</i> の選択された固有値に対応する正規直交固有ベクトルが格納される。すなわち、 $w(i)$ に対応する固有ベクトルが、 <i>z</i> の <i>i</i> 番目の列に格納される。固有ベクトルは、次のように正規化される。 $itype = 1$ または $2$ の場合、 $Z^H B Z = I$ 。 $itype = 3$ の場合、 $Z^H B^{-1} Z = I$ 。 $jobz = 'N'$ の場合、 <i>z</i> は参照されない。 固有ベクトルが収束できない場合、 <i>z</i> のその列には、その固有ベクトルに対する最新の近似値が格納され、固有ベクトルのインデックスは <i>ifail</i> に返される。 注：配列 <i>z</i> には、 $\max(1, m)$ 以上の列が提供されなければならない。 $range = 'V'$ の場合、 <i>m</i> の正確な値が事前にわからないため、上限値を使用する必要がある。
<i>work(1)</i>	終了時に、 $info = 0$ の場合、 <i>work(1)</i> は <i>lwork</i> の必要最小サイズを返す。
<i>ifail</i>	INTEGER。 配列、次元は $\max(1, n)$ 以上。 $jobz = 'V'$ の場合、 $info = 0$ であれば、 <i>ifail</i> の先頭から <i>m</i> 個の成分はゼロになる。 $info > 0$ であれば、収束しなかった固有ベクトルのインデックスが <i>ifail</i> に格納される。 $jobz = 'N'$ の場合、 <i>ifail</i> は参照されない。

*info* INTEGER。  
*info* = 0 の場合、実行は正常に終了したことを示す。  
*info* = -*i* の場合、*i* 番目の引数が不正であったことを示す。  
*info* > 0 の場合、cpotrf/zpotrf と cheevx/zheevx によって、エラーコードが返される。  
*info* = *i* ≤ *n* の場合、cheevx/zheevx が収束せず、*i* 個の固有ベクトルが収束しなかったことを示す。これらのインデックスは、配列 *ifail* に格納される。  
*info* = *n* + *i* (1 ≤ *i* ≤ *n*) の場合、*B* について先頭から次数 *i* の小行列が正定値でないことを示す。また、*B* の因子分解が完了できず、固有値または固有ベクトルが計算されなかった。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン hegvx のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ ( <i>n</i> , <i>n</i> ) の行列 <i>A</i> を格納する。
<i>b</i>	サイズ ( <i>n</i> , <i>n</i> ) の行列 <i>B</i> を格納する。
<i>w</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>z</i>	サイズ ( <i>n</i> , <i>m</i> ) の行列 <i>Z</i> を格納する。ここで、値 <i>n</i> と <i>m</i> は有意である。
<i>ifail</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>itype</i>	1、2、または 3 でなければならない。デフォルト値は 1。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>v1</i>	この成分のデフォルト値は、 <i>v1</i> = -HUGE( <i>v1</i> ) である。
<i>vu</i>	この成分のデフォルト値は、 <i>vu</i> = HUGE( <i>v1</i> ) である。
<i>il</i>	この引数のデフォルト値は、 <i>il</i> = 1 である。
<i>iu</i>	この引数のデフォルト値は、 <i>iu</i> = <i>n</i> である。
<i>abstol</i>	この成分のデフォルト値は、 <i>abstol</i> = 0.0_WP である。
<i>jobz</i>	引数 <i>z</i> の存在に基づいて以下のように復元される。 <i>jobz</i> = 'V' ( <i>z</i> が存在する場合)、 <i>jobz</i> = 'N' ( <i>z</i> が省略された場合)。 <i>ifail</i> が存在し、 <i>z</i> が省略された場合、エラー条件がセットされる。



*range*                    引数 *v1*, *vu*, *i1*, および *iu* の存在に基づいて以下のように復元される。

*range* = 'V' (*v1* と *vu* のいずれかまたは両方が存在する場合)、  
*range* = 'I' (*i1* と *iu* のいずれかまたは両方が存在する場合)、  
*range* = 'A' (*v1*, *vu*, *i1*, および *iu* がすべて存在しない場合。  
*v1* と *vu* のいずれかまたは両方が存在し、同時に *i1* と *iu* のいずれかまたは両方が存在する場合、エラー条件がセットされる。

### アプリケーション・ノート

$abstol + \varepsilon * \max(|a|, |b|)$  以下の幅の区間  $[a, b]$  内に存在していると判別された場合 ( $\varepsilon$  はマシン精度)、近似固有値は、収束したものとして受け入れられる。*abstol* がゼロ以下の場合、 $\varepsilon * \|T\|_1$  は所定の場所で使用される (*T* は、*A* を三重対角形式に縮退して獲得した三重対角行列)。

固有値が最も正確に計算されるのは、*abstol* がゼロでなく、アンダーフローのしきい値の 2 倍、 $2 * \lambda_{\text{mach}}('S')$  に設定されたときである。このルーチンで *info* > 0 が返された場合、固有ベクトルのいくつか収束に失敗したことを意味するので、*abstol* を  $2 * \lambda_{\text{mach}}('S')$  に設定して、やり直してみる。

最適なパフォーマンスを得るには、次の値を使用する。

$$lwork \geq (nb+1)*n$$

ここで、*nb* は、*ilaenv* が *chetrd/zhetrd* に対して返したブロックサイズである。配列 *work* に必要なワークスペースの大きさがわからない場合は、最初の実行で *lwork* を大きめの値に設定する。終了時に、*work*(1) の値を調べ、以降はその値を使用する。

### ?spgv

圧縮格納形式の行列に関する実数の汎用対称  
固有値問題について、固有値と(オプションで)  
固有ベクトルをすべて計算する。

---

#### 構文

##### Fortran 77:

```
call sspgv(itype, jobz, uplo, n, ap, bp, w, z, ldz, work, info)
call dspgv(itype, jobz, uplo, n, ap, bp, w, z, ldz, work, info)
```

##### Fortran 95:

```
call spgv(a, b, w [,itype] [,uplo] [,z] [,info])
```

#### 説明

このルーチンは、次の形式で示される実数の汎用対称固有値問題について、固有値と(オプションで)固有ベクトルをすべて計算する。

$$Ax = \lambda Bx, ABx = \lambda x, \text{ または } BAx = \lambda x$$

$A$  と  $B$  は対称であり、圧縮形式で格納されると仮定する。また、 $B$  は正定値である。

#### 入力パラメータ

<i>itype</i>	INTEGER。1、2、または3のいずれかでなければならない。 解決する問題のタイプを指定する。 <i>itype</i> = 1 の場合、問題のタイプは $Ax = \lambda Bx$ である。 <i>itype</i> = 2 の場合、問題のタイプは $ABx = \lambda x$ である。 <i>itype</i> = 3 の場合、問題のタイプは $BAx = \lambda x$ である。
<i>jobz</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobz</i> = 'N' の場合、固有値のみを計算する。 <i>jobz</i> = 'V' の場合、固有値と固有ベクトルを計算する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> = 'U' の場合、配列 <i>ap</i> と <i>bp</i> は、 $A$ と $B$ の上三角を格納する。 <i>uplo</i> = 'L' の場合、配列 <i>ap</i> と <i>bp</i> は、 $A$ と $B$ の下三角を格納する。
<i>n</i>	INTEGER。行列 $A$ と $B$ の次数 ( $n \geq 0$ )。

*ap*, *bp*, *work* REAL (sspgv の場合 )  
 DOUBLE PRECISION (dspgv の場合 )。  
 配列：  
*ap*(\*) には、*uplo* の値に従って、対称行列 *A* の上三角または下三角を圧縮形式で格納する。*ap* の次元は、 $\max(1, n*(n+1)/2)$  以上でなければならない。  
*bp*(\*) には、*uplo* の値に従って、対称行列 *B* の上三角または下三角を圧縮形式で格納する。*bp* の次元は、 $\max(1, n*(n+1)/2)$  以上でなければならない。  
*work*(\*) は、ワークスペース配列、次元は  $\max(1, 3n)$  以上。  
*ldz* INTEGER。出力配列 *z* のリーディング・ディメンジョン。 $ldz \geq 1$ 。  
*jobz*='V' の場合、 $ldz \geq \max(1, n)$ 。

### 出力パラメータ

*ap* 終了時に、*ap* の内容は上書きされる。  
*bp* 終了時に、コレスキー因子分解  $B = U^T U$  または  $B = L L^T$  からの三角係数 *U* または *L* が、*B* と同じ格納形式で格納される。  
*w*, *z* REAL (sspgv の場合 )  
 DOUBLE PRECISION (dspgv の場合 )。  
 配列：  
*w*(\*)、次元は  $\max(1, n)$  以上。  
*info*=0 の場合、固有値が昇順で格納される。  
*z*(*ldz*,\*)。 *z* の第 2 次元は、 $\max(1, n)$  以上でなければならない。  
*jobz*='V' を指定した場合に *info*=0 であると、*z* には、固有ベクトルの行列 *Z* が格納される。固有ベクトルは、次のように正規化される。  
*itype*=1 または 2 の場合、 $Z^T B Z = I$ 。  
*itype*=3 の場合、 $Z^T B^{-1} Z = I$ 。  
*jobz*='N' の場合、*z* は参照されない。  
*info* INTEGER。  
*info*=0 の場合、実行は正常に終了したことを示す。  
*info*=-*i* の場合、*i* 番目の引数が不正であったことを示す。  
*info*>0 の場合、*spptrf*/*dpptf* と *sspev*/*dspev* によって、エラーコードが返される。

$info = i \leq n$  の場合、sspev/dspev が収束せず、中間三重対角の  $i$  個の対角外の成分がゼロに収束しなかったことを示す。

$info = n + i$  ( $1 \leq i \leq n$ ) の場合、 $B$  について先頭から次数  $i$  の小行列が正定値でないことを示す。また、 $B$  の因子分解が完了できず、固有値または固有ベクトルが計算されなかった。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン spgv のインターフェイスの詳細を以下に示す。

$a$	Fortran 77 インターフェイスでの引数 $ap$ を意味する。 サイズ $(n*(n+1)/2)$ の配列 $A$ を格納する。
$b$	Fortran 77 インターフェイスでの引数 $bp$ を意味する。 サイズ $(n*(n+1)/2)$ の配列 $B$ を格納する。
$w$	長さ $(n)$ のベクトルを格納する。
$z$	サイズ $(n, n)$ の行列 $Z$ を格納する。
$itype$	1、2、または 3 でなければならない。デフォルト値は 1。
$uplo$	'U' または 'L' でなければならない。デフォルト値は 'U'。
$jobz$	引数 $z$ の存在に基づいて以下のように復元される。 $jobz = 'V'$ ( $z$ が存在する場合)、 $jobz = 'N'$ ( $z$ が省略された場合)。

## ?hpgv

圧縮格納形式の行列に関する複素数の汎用エルミート固有値問題について、固有値と ( オプションで ) 固有ベクトルをすべて計算する。

### 構文

#### Fortran 77:

```
call chpgv(itype, jobz, uplo, n, ap, bp, w, z, ldz, work, rwork, info)
call zhpgv(itype, jobz, uplo, n, ap, bp, w, z, ldz, work, rwork, info)
```

#### Fortran 95:

```
call hpgv(a, b, w [,itype] [,uplo] [,z] [,info])
```

### 説明

このルーチンは、次の形式で示される複素数の汎用エルミート固有値問題について、固有値と ( オプションで ) 固有ベクトルをすべて計算する。

$$Ax = \lambda Bx, \quad ABx = \lambda x, \quad \text{または} \quad BAx = \lambda x$$

$A$  と  $B$  はエルミートであり、圧縮形式で格納されると仮定する。また、 $B$  は正定値である。

### 入力パラメータ

<i>itype</i>	INTEGER。1、2、または3のいずれかでなければならない。 解決する問題のタイプを指定する。 <i>itype</i> = 1 の場合、問題のタイプは $Ax = \lambda Bx$ である。 <i>itype</i> = 2 の場合、問題のタイプは $ABx = \lambda x$ である。 <i>itype</i> = 3 の場合、問題のタイプは $BAx = \lambda x$ である。
<i>jobz</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobz</i> = 'N' の場合、固有値のみを計算する。 <i>jobz</i> = 'V' の場合、固有値と固有ベクトルを計算する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> = 'U' の場合、配列 <i>ap</i> と <i>bp</i> は、 $A$ と $B$ の上三角を格納する。 <i>uplo</i> = 'L' の場合、配列 <i>ap</i> と <i>bp</i> は、 $A$ と $B$ の下三角を格納する。
<i>n</i>	INTEGER。行列 $A$ と $B$ の次数 ( $n \geq 0$ )。

*ap*, *bp*, *work*    COMPLEX (chpgv の場合 )  
                          DOUBLE COMPLEX (zhpgv の場合 )。  
 配列:  
*ap*(\*) には、*uplo* の値に従って、エルミート行列 *A* の上三角または下三角を圧縮形式で格納する。*ap* の次元は、 $\max(1, n*(n+1)/2)$  以上でなければならない。  
*bp*(\*) には、*uplo* の値に従って、エルミート行列 *B* の上三角または下三角を圧縮形式で格納する。*bp* の次元は、 $\max(1, n*(n+1)/2)$  以上でなければならない。  
*work*(\*) は、ワークスペース配列、次元は  $\max(1, 2n-1)$  以上。  
*ldz*                 INTEGER。出力配列 *z* のリーディング・ディメンジョン。 $ldz \geq 1$ 。  
                          *jobz* = 'V' の場合、 $ldz \geq \max(1, n)$ 。  
*rwork*               REAL (chpgv の場合 )  
                          DOUBLE PRECISION (zhpgv の場合 )。  
                          ワークスペース配列、次元は  $\max(1, 3n-2)$  以上。

## 出力パラメータ

*ap*                    終了時に、*ap* の内容は上書きされる。  
*bp*                    終了時に、コレスキー因子分解  $B = U^T U$  または  $B = L L^T$  からの三角係数 *U* または *L* が、*B* と同じ格納形式で格納される。  
*w*                     REAL (chpgv の場合 )  
                          DOUBLE PRECISION (zhpgv の場合 )。  
                          配列、次元は  $\max(1, n)$  以上。  
                          *info* = 0 の場合、固有値が昇順で格納される。  
*z*                     COMPLEX (chpgv の場合 )  
                          DOUBLE COMPLEX (zhpgv の場合 )。  
                          配列 *z*(*ldz*,\*)。 *z* の第 2 次元は、 $\max(1, n)$  以上でなければならない。  
                          *jobz* = 'V' を指定した場合に *info* = 0 であると、*z* には、固有ベクトルの行列 *Z* が格納される。固有ベクトルは、次のように正規化される。  
                          *itype* = 1 または 2 の場合、 $Z^H B Z = I$ 。  
                          *itype* = 3 の場合、 $Z^H B^{-1} Z = I$ 。  
                          *jobz* = 'N' の場合、*z* は参照されない。

*info* INTEGER。  
*info* = 0 の場合、実行は正常に終了したことを示す。  
*info* = -*i* の場合、*i* 番目の引数が不正であったことを示す。  
*info* > 0 の場合、cpptrf/zpptrf と chpev/zhpev によって、エラーコードが返される。

*info* = *i* ≤ *n* の場合、chpev/zhpev が収束せず、中間三重対角の *i* 個の対角外の成分がゼロに収束しなかったことを示す。  
*info* = *n* + *i* (1 ≤ *i* ≤ *n*) の場合、*B* について先頭から次数 *i* の小行列が正定値でないことを示す。また、*B* の因子分解が完了できず、固有値または固有ベクトルが計算されなかった。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン hpgv のインターフェイスの詳細を以下に示す。

*a* Fortran 77 インターフェイスでの引数 *ap* を意味する。  
 サイズ  $(n \cdot (n+1)/2)$  の配列 *A* を格納する。

*b* Fortran 77 インターフェイスでの引数 *bp* を意味する。  
 サイズ  $(n \cdot (n+1)/2)$  の配列 *B* を格納する。

*w* 長さ (*n*) のベクトルを格納する。

*z* サイズ (*n*, *n*) の行列 *Z* を格納する。

*itype* 1、2、または 3 でなければならない。デフォルト値は 1。

*uplo* 'U' または 'L' でなければならない。デフォルト値は 'U'。

*jobz* 引数 *z* の存在に基づいて以下のように復元される。  
*jobz* = 'V' (*z* が存在する場合)、  
*jobz* = 'N' (*z* が省略された場合)。

### ?spgvd

圧縮格納形式の行列に関する実数の汎用対称固有値問題について、固有値と (オプションで) 固有ベクトルをすべて計算する。固有ベクトルを計算する場合は、分割統治法を使用する。

---

#### 構文

##### Fortran 77:

```
call sspgvd(itype, jobz, uplo, n, ap, bp, w, z, ldz, work, lwork, iwork, liwork, info)
call dspgvd(itype, jobz, uplo, n, ap, bp, w, z, ldz, work, lwork, iwork, liwork, info)
```

##### Fortran 95:

```
call spgvd(a, b, w [,itype] [,uplo] [,z] [,info])
```

#### 説明

このルーチンは、次の形式で示される実数の汎用対称固有値問題について、固有値と (オプションで) 固有ベクトルをすべて計算する。

$$Ax = \lambda Bx, ABx = \lambda x, \text{ または } BAx = \lambda x$$

$A$  と  $B$  は対称であり、圧縮形式で格納されると仮定する。また、 $B$  は正定値である。固有ベクトルを計算する場合は、分割統治法を使用する。

#### 入力パラメータ

<b>itype</b>	INTEGER。1、2、または3のいずれかでなければならない。 解決する問題のタイプを指定する。 $itype = 1$ の場合、問題のタイプは $Ax = \lambda Bx$ である。 $itype = 2$ の場合、問題のタイプは $ABx = \lambda x$ である。 $itype = 3$ の場合、問題のタイプは $BAx = \lambda x$ である。
<b>jobz</b>	CHARACTER*1。'N' または 'V' でなければならない。 $jobz = 'N'$ の場合、固有値のみを計算する。 $jobz = 'V'$ の場合、固有値と固有ベクトルを計算する。



<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> ='U' の場合、配列 <i>ap</i> と <i>bp</i> は、 <i>A</i> と <i>B</i> の上三角を格納する。 <i>uplo</i> ='L' の場合、配列 <i>ap</i> と <i>bp</i> は、 <i>A</i> と <i>B</i> の下三角を格納する。
<i>n</i>	INTEGER。行列 <i>A</i> と <i>B</i> の次数 ( $n \geq 0$ )。
<i>ap</i> , <i>bp</i> , <i>work</i>	REAL (sspgvd の場合 ) DOUBLE PRECISION (dspgvd の場合 )。 配列： <i>ap</i> (*) には、 <i>uplo</i> の値に従って、対称行列 <i>A</i> の上三角または下三角を圧縮形式で格納する。 <i>ap</i> の次元は、 $\max(1, n*(n+1)/2)$ 以上でなければならない。  <i>bp</i> (*) には、 <i>uplo</i> の値に従って、対称行列 <i>B</i> の上三角または下三角を圧縮形式で格納する。 <i>bp</i> の次元は、 $\max(1, n*(n+1)/2)$ 以上でなければならない。  <i>work</i> ( <i>lwork</i> ) は、ワークスペース配列である。
<i>ldz</i>	INTEGER。出力配列 <i>z</i> のリーディング・ディメンジョン。 $ldz \geq 1$ 。 <i>jobz</i> ='V' の場合、 $ldz \geq \max(1, n)$ 。
<i>lwork</i>	INTEGER。配列 <i>work</i> の次元。  次の制約がある。 $n \leq 1$ の場合、 $lwork \geq 1$ 。 <i>jobz</i> ='N' で $n > 1$ の場合、 $lwork \geq 2n$ 。 <i>jobz</i> ='V' で $n > 1$ の場合、 $lwork \geq 2n^2+6n+1$ 。  $lwork = -1$ の場合、ワークスペースのクエリとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリとして返す。 <i>xerbla</i> は <i>lwork</i> に関するエラー・メッセージを生成しない。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は ( <i>liwork</i> )。
<i>liwork</i>	INTEGER。配列 <i>iwork</i> の次元。  次の制約がある。 $n \leq 1$ の場合、 $liwork \geq 1$ 。 <i>jobz</i> ='N' で $n > 1$ の場合、 $liwork \geq 1$ 。 <i>jobz</i> ='V' で $n > 1$ の場合、 $liwork \geq 5n+3$ 。  $liwork = -1$ の場合、ワークスペースのクエリとみなされ、ルーチンは

*iwork* 配列の最適サイズだけを計算し、その値を *iwork* 配列の最初のエントリとして返す。xerbla は *liwork* に関するエラー・メッセージを生成しない。

## 出力パラメータ

<i>ap</i>	終了時に、 <i>ap</i> の内容は上書きされる。
<i>bp</i>	終了時に、コレスキー因子分解 $B = U^T U$ または $B = L L^T$ からの三角係数 $U$ または $L$ が、 $B$ と同じ格納形式で格納される。
<i>w, z</i>	<p>REAL (sspgv の場合 )            DOUBLE PRECISION (dspgv の場合 )。            配列：  <math>w(*)</math>、次元は <math>\max(1, n)</math> 以上。  <math>info = 0</math> の場合、固有値が昇順で格納される。</p> <p><math>z(ldz, *)</math>。 <math>z</math> の第 2 次元は、<math>\max(1, n)</math> 以上でなければならない。  <math>jobz = 'V'</math> を指定した場合に <math>info = 0</math> であると、<math>z</math> には、固有ベクトルの行列 <math>Z</math> が格納される。固有ベクトルは、次のように正規化される。  <math>itype = 1</math> または <math>2</math> の場合、<math>Z^T B Z = I</math>。  <math>itype = 3</math> の場合、<math>Z^T B^{-1} Z = I</math>。  <math>jobz = 'N'</math> の場合、<math>z</math> は参照されない。</p>
<i>work(1)</i>	終了時に、 $info = 0$ の場合、 <i>work(1)</i> は <i>lwork</i> の必要最小サイズを返す。
<i>iwork(1)</i>	終了時に、 $info = 0$ の場合、 <i>iwork(1)</i> は <i>liwork</i> の必要最小サイズを返す。
<i>info</i>	<p>INTEGER。  <math>info = 0</math> の場合、実行は正常に終了したことを示す。  <math>info = -i</math> の場合、<math>i</math> 番目の引数が不正であったことを示す。  <math>info &gt; 0</math> の場合、spptrf/dpptrf と sspevd/dspevd によって、エラーコードが返される。</p> <p><math>info = i \leq n</math> の場合、sspevd/dspevd が収束せず、中間三重対角の <math>i</math> 個の対角外の成分がゼロに収束しなかったことを示す。  <math>info = n + i</math> (<math>1 \leq i \leq n</math>) の場合、<math>B</math> について先頭から次数 <math>i</math> の小行列が正定値でないことを示す。また、<math>B</math> の因子分解が完了できず、固有値または固有ベクトルが計算されなかった。</p>

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `spgvd` のインターフェイスの詳細を以下に示す。

<code>a</code>	Fortran 77 インターフェイスでの引数 <code>ap</code> を意味する。 サイズ $(n*(n+1)/2)$ の配列 $A$ を格納する。
<code>b</code>	Fortran 77 インターフェイスでの引数 <code>bp</code> を意味する。 サイズ $(n*(n+1)/2)$ の配列 $B$ を格納する。
<code>w</code>	長さ $(n)$ のベクトルを格納する。
<code>z</code>	サイズ $(n,n)$ の行列 $Z$ を格納する。
<code>itype</code>	1、2、または 3 でなければならない。デフォルト値は 1。
<code>uplo</code>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<code>jobz</code>	引数 $z$ の存在に基づいて以下のように復元される。 <code>jobz = 'V'</code> ( $z$ が存在する場合)、 <code>jobz = 'N'</code> ( $z$ が省略された場合)。

---

## ?hpgvd

圧縮格納形式の行列に関する複素数の汎用エルミート固有値問題について、固有値と(オプションで)固有ベクトルをすべて計算する。固有ベクトルを計算する場合は、分割統治法を使用する。

---

### 構文

#### Fortran 77:

```
call chpgvd(itype, jobz, uplo, n, ap, bp, w, z, ldz, work, lwork, rwork, lrwork,  
            iwork, liwork, info)  
call zhpgvd(itype, jobz, uplo, n, ap, bp, w, z, ldz, work, lwork, rwork, lrwork,  
            iwork, liwork, info)
```

## Fortran 95:

```
call hpgvd(a, b, w [,itype] [,uplo] [,z] [,info])
```

### 説明

このルーチンは、次の形式で示される複素数の汎用エルミート固有値問題について、固有値と (オプションで) 固有ベクトルをすべて計算する。

$$Ax = \lambda Bx, \quad ABx = \lambda x, \quad \text{または} \quad BAx = \lambda x$$

$A$  と  $B$  はエルミートであり、圧縮形式で格納されると仮定する。また、 $B$  は正定値である。固有ベクトルを計算する場合は、分割統治法を使用する。

### 入力パラメータ

<i>itype</i>	INTEGER。1、2、または3のいずれかでなければならない。 解決する問題のタイプを指定する。 <i>itype</i> = 1 の場合、問題のタイプは $Ax = \lambda Bx$ である。 <i>itype</i> = 2 の場合、問題のタイプは $ABx = \lambda x$ である。 <i>itype</i> = 3 の場合、問題のタイプは $BAx = \lambda x$ である。
<i>jobz</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobz</i> = 'N' の場合、固有値のみを計算する。 <i>jobz</i> = 'V' の場合、固有値と固有ベクトルを計算する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> = 'U' の場合、配列 <i>ap</i> と <i>bp</i> は、 $A$ と $B$ の上三角を格納する。 <i>uplo</i> = 'L' の場合、配列 <i>ap</i> と <i>bp</i> は、 $A$ と $B$ の下三角を格納する。
<i>n</i>	INTEGER。行列 $A$ と $B$ の次数 ( $n \geq 0$ )。
<i>ap</i> , <i>bp</i> , <i>work</i>	COMPLEX (chpgvd の場合) DOUBLE COMPLEX (zhpgvd の場合)。 配列: <i>ap</i> (*) には、 <i>uplo</i> の値に従って、エルミート行列 $A$ の上三角または下三角を圧縮形式で格納する。 <i>ap</i> の次元は、 $\max(1, n*(n+1)/2)$ 以上でなければならない。  <i>bp</i> (*) には、 <i>uplo</i> の値に従って、エルミート行列 $B$ の上三角または下三角を圧縮形式で格納する。 <i>bp</i> の次元は、 $\max(1, n*(n+1)/2)$ 以上でなければならない。  <i>work</i> ( <i>lwork</i> ) は、ワークスペース配列である。
<i>ldz</i>	INTEGER。出力配列 <i>z</i> のリーディング・ディメンジョン。 $ldz \geq 1$ 。 <i>jobz</i> = 'V' の場合、 $ldz \geq \max(1, n)$ 。

<i>lwork</i>	INTEGER。配列 <i>work</i> の次元。  次の制約がある。 $n \leq 1$ の場合、 $lwork \geq 1$ 。 $jobz = 'N'$ で $n > 1$ の場合、 $lwork \geq n$ 。 $jobz = 'V'$ で $n > 1$ の場合、 $lwork \geq 2n$ 。  $lwork = -1$ の場合、ワークスペースのクエリとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリとして返す。xerbla は <i>lwork</i> に関するエラー・メッセージを生成しない。
<i>rwork</i>	REAL (chpgvd の場合) DOUBLE PRECISION (zhpgvd の場合)。 ワークスペース配列、次元は ( <i>lrwork</i> )。
<i>lrwork</i>	INTEGER。配列 <i>rwork</i> の次元。  次の制約がある。 $n \leq 1$ の場合、 $lrwork \geq 1$ 。 $jobz = 'N'$ で $n > 1$ の場合、 $lrwork \geq n$ 。 $jobz = 'V'$ で $n > 1$ の場合、 $lrwork \geq 2n^2 + 5n + 1$ 。  $lrwork = -1$ の場合、ワークスペースのクエリとみなされ、ルーチンは <i>rwork</i> 配列の最適サイズだけを計算し、その値を <i>rwork</i> 配列の最初のエントリとして返す。xerbla は <i>lrwork</i> に関するエラー・メッセージを生成しない。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は ( <i>liwork</i> )。
<i>liwork</i>	INTEGER。配列 <i>iwork</i> の次元。  次の制約がある。 $n \leq 1$ の場合、 $liwork \geq 1$ 。 $jobz = 'N'$ で $n > 1$ の場合、 $liwork \geq 1$ 。 $jobz = 'V'$ で $n > 1$ の場合、 $liwork \geq 5n + 3$ 。  $liwork = -1$ の場合、ワークスペースのクエリとみなされ、ルーチンは <i>iwork</i> 配列の最適サイズだけを計算し、その値を <i>iwork</i> 配列の最初のエントリとして返す。xerbla は <i>liwork</i> に関するエラー・メッセージを生成しない。

## 出力パラメータ

<i>ap</i>	終了時に、 <i>ap</i> の内容は上書きされる。
-----------	-----------------------------

<i>bp</i>	終了時に、コレスキー因子分解 $B = U^T U$ または $B = L L^T$ からの三角係数 $U$ または $L$ が、 $B$ と同じ格納形式で格納される。
<i>w</i>	REAL (chpgvd の場合 ) DOUBLE PRECISION (zhpgvd の場合 )。 配列、次元は $\max(1, n)$ 以上。 $info = 0$ の場合、固有値が昇順で格納される。
<i>z</i>	COMPLEX (chpgvd の場合 ) DOUBLE COMPLEX (zhpgvd の場合 )。 配列 $z(ldz, *)$ 。 $z$ の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $jobz = 'V'$ を指定した場合に $info = 0$ であると、 $z$ には、固有ベクトルの行列 $Z$ が格納される。固有ベクトルは、次のように正規化される。 $itype = 1$ または $2$ の場合、 $Z^H B Z = I$ 。 $itype = 3$ の場合、 $Z^H B^{-1} Z = I$ 。 $jobz = 'N'$ の場合、 $z$ は参照されない。
<i>work(1)</i>	終了時に、 $info = 0$ の場合、 <i>work(1)</i> は <i>lwork</i> の必要最小サイズを返す。
<i>rwork(1)</i>	終了時に、 $info = 0$ の場合、 <i>rwork(1)</i> は、 <i>lrwork</i> の必要最小サイズを返す。
<i>iwork(1)</i>	終了時に、 $info = 0$ の場合、 <i>iwork(1)</i> は <i>liwork</i> の必要最小サイズを返す。
<i>info</i>	INTEGER。 $info = 0$ の場合、実行は正常に終了したことを示す。 $info = -i$ の場合、 $i$ 番目の引数が不正であったことを示す。 $info > 0$ の場合、cpptrf/zpptrf と chpevd/zhpevd によって、エラーコードが返される。  $info = i \leq n$ の場合、chpevd/zhpevd が収束せず、中間三重対角の $i$ 個の対角外の成分がゼロに収束しなかったことを示す。 $info = n + i$ ( $1 \leq i \leq n$ ) の場合、 $B$ について先頭から次数 $i$ の小行列が正定値でないことを示す。また、 $B$ の因子分解が完了できず、固有値または固有ベクトルが計算されなかった。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `hpgvd` のインターフェイスの詳細を以下に示す。

<code>a</code>	Fortran 77 インターフェイスでの引数 <code>ap</code> を意味する。 サイズ $(n*(n+1)/2)$ の配列 <i>A</i> を格納する。
<code>b</code>	Fortran 77 インターフェイスでの引数 <code>bp</code> を意味する。 サイズ $(n*(n+1)/2)$ の配列 <i>B</i> を格納する。
<code>w</code>	長さ $(n)$ のベクトルを格納する。
<code>z</code>	サイズ $(n,n)$ の行列 <i>Z</i> を格納する。
<code>itype</code>	1、2、または 3 でなければならない。デフォルト値は 1。
<code>uplo</code>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<code>jobz</code>	引数 <i>z</i> の存在に基づいて以下のように復元される。 <code>jobz = 'V'</code> ( <i>z</i> が存在する場合)、 <code>jobz = 'N'</code> ( <i>z</i> が省略された場合)。

## ?spgvx

圧縮格納形式の行列に関する実数の汎用対称  
固有値問題について、固有値と(オプションで)  
固有ベクトルを選択的に計算する。

### 構文

#### Fortran 77:

```
call sspgvx(itype, jobz, range, uplo, n, ap, bp, vl, vu, il, iu, abstol, m, w,
            z, ldz, work, iwork, ifail, info)
call dspgvx(itype, jobz, range, uplo, n, ap, bp, vl, vu, il, iu, abstol, m, w,
            z, ldz, work, iwork, ifail, info)
```

#### Fortran 95:

```
call spgvx(a, b, w [,itype] [,uplo] [,z] [,vl] [,vu] [,il] [,iu] [,m] [,ifail]
            [,abstol] [,info])
```

## 説明

このルーチンは、次の形式で示される実数の汎用対称固有値問題について、固有値と (オプションで) 固有ベクトルを選択的に計算する。

$$Ax = \lambda Bx, ABx = \lambda x, \text{ または } BAx = \lambda x$$

$A$  と  $B$  は対称であり、圧縮形式で格納されると仮定する。また、 $B$  は正定値である。固有値と固有ベクトルを選択するには、希望する固有値の値の範囲かインデックスの範囲を指定する。

## 入力パラメータ

<i>itype</i>	INTEGER。1、2、または3のいずれかでなければならない。 解決する問題のタイプを指定する。 <i>itype</i> = 1 の場合、問題のタイプは $Ax = \lambda Bx$ である。 <i>itype</i> = 2 の場合、問題のタイプは $ABx = \lambda x$ である。 <i>itype</i> = 3 の場合、問題のタイプは $BAx = \lambda x$ である。
<i>jobz</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobz</i> = 'N' の場合、固有値のみを計算する。 <i>jobz</i> = 'V' の場合、固有値と固有ベクトルを計算する。
<i>range</i>	CHARACTER*1。'A'、'V'、または 'I' のいずれかでなければならない。 <i>range</i> = 'A' の場合、固有値をすべて計算する。 <i>range</i> = 'V' の場合、半開区間 $vl < \lambda_i \leq vu$ における固有値 $\lambda_i$ を計算する。 <i>range</i> = 'I' の場合、ルーチンはインデックスが $il \sim iu$ である固有値を計算する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> = 'U' の場合、配列 <i>ap</i> と <i>bp</i> は、 $A$ と $B$ の上三角を格納する。 <i>uplo</i> = 'L' の場合、配列 <i>ap</i> と <i>bp</i> は、 $A$ と $B$ の下三角を格納する。
<i>n</i>	INTEGER。行列 $A$ と $B$ の次数 ( $n \geq 0$ )。
<i>ap</i> , <i>bp</i> , <i>work</i>	REAL (sspgvx の場合) DOUBLE PRECISION (dspgvx の場合)。 配列: <i>ap</i> (*) には、 <i>uplo</i> の値に従って、対称行列 $A$ の上三角または下三角を圧縮形式で格納する。 <i>ap</i> の次元は、 $\max(1, n*(n+1)/2)$ 以上でなければならない。



$bp(*)$  には、 $uplo$  の値に従って、対称行列  $B$  の上三角または下三角を圧縮形式で格納する。 $bp$  の次元は、 $\max(1, n*(n+1)/2)$  以上でなければならない。

$work(*)$  は、ワークスペース配列、次元は  $\max(1, 8n)$  以上。

$vl, vu$	<p>REAL (sspgvx の場合 )  DOUBLE PRECISION (dspgvx の場合 )。  <math>range = 'V'</math> の場合、固有値を検索する区間の上限と下限。  次の制約がある。<math>vl &lt; vu</math>。</p> <p><math>range = 'A'</math> または <math>'I'</math> の場合、<math>vl</math> と <math>vu</math> は参照されない。</p>
$il, iu$	<p>INTEGER。  <math>range = 'I'</math> の場合、返される最小固有値と最大固有値に対する昇順のインデックス。  次の制約がある。  <math>1 \leq il \leq iu \leq n</math> (<math>n &gt; 0</math> の場合 )。  <math>il=1</math> かつ <math>iu=0</math> (<math>n=0</math> の場合 )。</p> <p><math>range = 'A'</math> または <math>'V'</math> の場合、<math>il</math> と <math>iu</math> は参照されない。</p>
$abstol$	<p>REAL (sspgvx の場合 )  DOUBLE PRECISION (dspgvx の場合 )。  固有値に対する絶対エラー許容値。  詳細は、「アプリケーション・ノート」を参照。</p>
$ldz$	<p>INTEGER。出力配列 <math>z</math> のリーディング・ディメンション。  次の制約がある。  <math>ldz \geq 1</math>。  <math>jobz = 'V'</math> の場合、<math>ldz \geq \max(1, n)</math>。</p>
$iwork$	<p>INTEGER。  ワークスペース配列、次元は <math>\max(1, 5n)</math> 以上。</p>

### 出力パラメータ

$ap$	終了時に、 $ap$ の内容は上書きされる。
$bp$	終了時に、コレスキー因子分解 $B = U^T U$ または $B = L L^T$ からの三角係数 $U$ または $L$ が、 $B$ と同じ格納形式で格納される。
$m$	INTEGER。検出された固有値の総数。 $(0 \leq m \leq n)$ 。 $range = 'A'$ の場合、 $m = n$ 、 $range = 'I'$ の場合、 $m = iu - il + 1$ 。

<i>w, z</i>	<p>REAL (sspgvx の場合 )  DOUBLE PRECISION (dspgvx の場合 )。  配列 :  <math>w(*)</math>、次元は <math>\max(1, n)</math> 以上。  <math>info = 0</math> の場合、固有値が昇順で格納される。</p> <p><math>z(ldz, *)</math>。 <math>z</math> の第 2 次元は、<math>\max(1, n)</math> 以上でなければならない。  <math>jobz = 'V'</math> の場合、<math>info = 0</math> であれば、<math>z</math> の先頭の <math>m</math> 列に、行列 <math>A</math> の選択された固有値に対応する正規直交固有ベクトルが格納される。すなわち、<math>w(i)</math> に対応する固有ベクトルが、<math>z</math> の <math>i</math> 番目の列に格納される。固有ベクトルは、次のように正規化される。  <math>itype = 1</math> または <math>2</math> の場合、<math>Z^T B Z = I</math>。  <math>itype = 3</math> の場合、<math>Z^T B^{-1} Z = I</math>。</p> <p><math>jobz = 'N'</math> の場合、<math>z</math> は参照されない。  固有ベクトルが収束できない場合、<math>z</math> のその列には、その固有ベクトルに対する最新の近似値が格納され、固有ベクトルのインデックスは <i>ifail</i> に返される。  注: 配列 <math>z</math> には、<math>\max(1, m)</math> 以上の列が提供されなければならない。  <math>range = 'V'</math> の場合、<math>m</math> の正確な値が事前にわからないため、上限値を使用する必要がある。</p>
<i>ifail</i>	<p>INTEGER。  配列、次元は <math>\max(1, n)</math> 以上。  <math>jobz = 'V'</math> の場合、<math>info = 0</math> であれば、<i>ifail</i> の先頭から <math>m</math> 個の成分はゼロになる。<math>info &gt; 0</math> であれば、収束しなかった固有ベクトルのインデックスが <i>ifail</i> に格納される。  <math>jobz = 'N'</math> の場合、<i>ifail</i> は参照されない。</p>
<i>info</i>	<p>INTEGER。  <math>info = 0</math> の場合、実行は正常に終了したことを示す。  <math>info = -i</math> の場合、<math>i</math> 番目の引数が不正であったことを示す。  <math>info &gt; 0</math> の場合、spptrf/dpptrf と sspevx/dspevx によって、エラーコードが返される。</p> <p><math>info = i \leq n</math> の場合、sspevx/dspevx が収束せず、<math>i</math> 個の固有ベクトルが収束しなかったことを示す。これらのインデックスは、配列 <i>ifail</i> に格納される。  <math>info = n + i</math> (<math>1 \leq i \leq n</math>) の場合、<math>B</math> について先頭から次数 <math>i</math> の小行列が正定値でないことを示す。また、<math>B</math> の因子分解が完了できず、固有値または固有ベクトルが計算されなかった。</p>

**Fortran 95 インターフェイス・ノート**

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `spgvx` のインターフェイスの詳細を以下に示す。

<code>a</code>	Fortran 77 インターフェイスでの引数 <code>ap</code> を意味する。 サイズ $(n*(n+1)/2)$ の配列 $A$ を格納する。
<code>b</code>	Fortran 77 インターフェイスでの引数 <code>bp</code> を意味する。 サイズ $(n*(n+1)/2)$ の配列 $B$ を格納する。
<code>w</code>	長さ $(n)$ のベクトルを格納する。
<code>z</code>	サイズ $(n,m)$ の行列 $Z$ を格納する。ここで、値 $n$ と $m$ は有意である。
<code>ifail</code>	長さ $(n)$ のベクトルを格納する。
<code>itype</code>	1、2、または 3 でなければならない。デフォルト値は 1。
<code>uplo</code>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<code>v1</code>	この成分のデフォルト値は、 <code>v1 = -HUGE(v1)</code> である。
<code>vu</code>	この成分のデフォルト値は、 <code>vu = HUGE(v1)</code> である。
<code>i1</code>	この引数のデフォルト値は、 <code>i1 = 1</code> である。
<code>iu</code>	この引数のデフォルト値は、 <code>iu = n</code> である。
<code>abstol</code>	この成分のデフォルト値は、 <code>abstol = 0.0_WP</code> である。
<code>jobz</code>	引数 $z$ の存在に基づいて以下のように復元される。 <code>jobz = 'V'</code> ( $z$ が存在する場合)、 <code>jobz = 'N'</code> ( $z$ が省略された場合)。 <code>ifail</code> が存在し、 $z$ が省略された場合、エラー条件がセットされる。
<code>range</code>	引数 $v1$ 、 $vu$ 、 $i1$ 、および $iu$ の存在に基づいて以下のように復元される。 <code>range = 'V'</code> ( $v1$ と $vu$ のいずれかまたは両方が存在する場合)、 <code>range = 'I'</code> ( $i1$ と $iu$ のいずれかまたは両方が存在する場合)、 <code>range = 'A'</code> ( $v1$ 、 $vu$ 、 $i1$ 、および $iu$ がすべて存在しない場合)。 $v1$ と $vu$ のいずれかまたは両方が存在し、同時に $i1$ と $iu$ のいずれかまたは両方が存在する場合、エラー条件がセットされる。

### アプリケーション・ノート

$abstol + \varepsilon * \max(|a|, |b|)$  以下の幅の区間  $[a, b]$  内に存在していると判別された場合 ( $\varepsilon$  はマシン精度)、近似固有値は、収束したものとして受け入れられる。 $abstol$  がゼロ以下の場合、 $\varepsilon * \|T\|_1$  は所定の場所で使用される ( $T$  は、 $A$  を三重対角形式に縮退して獲得した三重対角行列)。

固有値が最も正確に計算されるのは、 $abstol$  がゼロでなく、アンダーフローのしきい値の 2 倍、 $2 * \lambda_{\text{lamch}}('S')$  に設定されたときである。このルーチンで  $info > 0$  が返された場合、固有ベクトルのいくつか収束に失敗したことを意味するので、 $abstol$  を  $2 * \lambda_{\text{lamch}}('S')$  に設定して、やり直してみる。

---

## ?hpgvx

圧縮格納形式の行列に関する汎用エルミート  
固有値問題について、固有値と (オプションで)  
固有ベクトルを選択的に計算する。

---

### 構文

#### Fortran 77:

```
call chpgvx(itype, jobz, range, uplo, n, ap, bp, vl, vu, il, iu, abstol, m, w,  
            z, ldz, work, rwork, iwork, ifail, info)  
call zhpgvx(itype, jobz, range, uplo, n, ap, bp, vl, vu, il, iu, abstol, m, w,  
            z, ldz, work, rwork, iwork, ifail, info)
```

#### Fortran 95:

```
call hpgvx(a, b, w [,itype] [,uplo] [,z] [,vl] [,vu] [,il] [,iu] [,m] [,ifail]  
            [,abstol] [,info])
```

### 説明

このルーチンは、次の形式で示される複素数の汎用エルミート固有値問題について、固有値と (オプションで) 固有ベクトルを選択的に計算する。

$$Ax = \lambda Bx, ABx = \lambda x, \text{ または } BAx = \lambda x$$

$A$  と  $B$  はエルミートであり、圧縮形式で格納されると仮定する。また、 $B$  は正定値である。

固有値と固有ベクトルを選択するには、希望する固有値の値の範囲かインデックスの範囲を指定する。

## 入力パラメータ

<i>itype</i>	INTEGER。1、2、または3のいずれかでなければならない。 解決する問題のタイプを指定する。 <i>itype</i> = 1 の場合、問題のタイプは $Ax = \lambda Bx$ である。 <i>itype</i> = 2 の場合、問題のタイプは $ABx = \lambda x$ である。 <i>itype</i> = 3 の場合、問題のタイプは $BAx = \lambda x$ である。
<i>jobz</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobz</i> = 'N' の場合、固有値のみを計算する。 <i>jobz</i> = 'V' の場合、固有値と固有ベクトルを計算する。
<i>range</i>	CHARACTER*1。'A'、'V'、または 'I' のいずれかでなければならない。 <i>range</i> = 'A' の場合、固有値をすべて計算する。 <i>range</i> = 'V' の場合、半開区間 $v1 < \lambda_i \leq vu$ における固有値 $\lambda_i$ を計算する。 <i>range</i> = 'I' の場合、ルーチンはインデックスが $i1 \sim iu$ である固有値を計算する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> = 'U' の場合、配列 <i>ap</i> と <i>bp</i> は、 <i>A</i> と <i>B</i> の上三角を格納する。 <i>uplo</i> = 'L' の場合、配列 <i>ap</i> と <i>bp</i> は、 <i>A</i> と <i>B</i> の下三角を格納する。
<i>n</i>	INTEGER。行列 <i>A</i> と <i>B</i> の次数 ( $n \geq 0$ )。
<i>ap</i> , <i>bp</i> , <i>work</i>	COMPLEX (chpgvx の場合) DOUBLE COMPLEX (zhpgvx の場合)。 配列： <i>ap</i> (*) には、 <i>uplo</i> の値に従って、エルミート行列 <i>A</i> の上三角または下三角を圧縮形式で格納する。 <i>ap</i> の次元は、 $\max(1, n*(n+1)/2)$ 以上でなければならない。  <i>bp</i> (*) には、 <i>uplo</i> の値に従って、エルミート行列 <i>B</i> の上三角または下三角を圧縮形式で格納する。 <i>bp</i> の次元は、 $\max(1, n*(n+1)/2)$ 以上でなければならない。  <i>work</i> (*) は、ワークスペース配列、次元は $\max(1, 2n)$ 以上。
<i>v1</i> , <i>vu</i>	REAL (chpgvx の場合) DOUBLE PRECISION (zhpgvx の場合)。 <i>range</i> = 'V' の場合、固有値を検索する区間の上限と下限。 次の制約がある。 $v1 < vu$ 。  <i>range</i> = 'A' または 'I' の場合、 <i>v1</i> と <i>vu</i> は参照されない。

<i>il, iu</i>	INTEGER。 <i>range</i> = 'I' の場合、返される最小固有値と最大固有値に対する昇順のインデックス。 次の制約がある。 $1 \leq il \leq iu \leq n$ ( $n > 0$ の場合)。 $il=1$ かつ $iu=0$ ( $n=0$ の場合)。 <i>range</i> = 'A' または 'V' の場合、 <i>il</i> と <i>iu</i> は参照されない。
<i>abstol</i>	REAL (chpgvx の場合) DOUBLE PRECISION (zhpgvx の場合)。 固有値に対する絶対エラー許容値。 詳細は、「アプリケーション・ノート」を参照。
<i>ldz</i>	INTEGER。出力配列 <i>z</i> のリーディング・ディメンション。 $ldz \geq 1$ 。 <i>jobz</i> = 'V' の場合、 $ldz \geq \max(1, n)$ 。
<i>rwork</i>	REAL (chpgvx の場合) DOUBLE PRECISION (zhpgvx の場合)。 ワークスペース配列、次元は $\max(1, 7n)$ 以上。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, 5n)$ 以上。

## 出力パラメータ

<i>ap</i>	終了時に、 <i>ap</i> の内容は上書きされる。
<i>bp</i>	終了時に、コレスキー因子分解 $B = U^T U$ または $B = L L^T$ からの三角係数 <i>U</i> または <i>L</i> が、 <i>B</i> と同じ格納形式で格納される。
<i>m</i>	INTEGER。検出された固有値の総数。 $(0 \leq m \leq n)$ 。 <i>range</i> = 'A' の場合、 $m = n$ 、 <i>range</i> = 'I' の場合、 $m = iu - il + 1$ 。
<i>w</i>	REAL (chpgvx の場合) DOUBLE PRECISION (zhpgvx の場合)。 配列、次元は $\max(1, n)$ 以上。 <i>info</i> = 0 の場合、固有値が昇順で格納される。
<i>z</i>	COMPLEX (chpgvx の場合) DOUBLE COMPLEX (zhpgvx の場合)。 配列 <i>z</i> ( <i>ldz</i> ,*)。 <i>z</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>jobz</i> = 'V' の場合、 <i>info</i> = 0 であれば、 <i>z</i> の先頭の <i>m</i> 列に、行列 <i>A</i> の選択された固有値に対応する正規直交固有ベクトルが格納される。すなわち、 <i>w</i> ( <i>i</i> ) に対応する固有ベクトルが、 <i>z</i> の <i>i</i> 番目の列に格納される。

固有ベクトルは、次のように正規化される。

$itype = 1$  または  $2$  の場合、 $Z^H B Z = I$ 。

$itype = 3$  の場合、 $Z^H B^{-1} Z = I$ 。

$jobz = 'N'$  の場合、 $z$  は参照されない。

固有ベクトルが収束できない場合、 $z$  のその列には、その固有ベクトルに対する最新の近似値が格納され、固有ベクトルのインデックスは  $ifail$  に返される。

注：配列  $z$  には、 $\max(1, m)$  以上の列が提供されなければならない。

$range = 'V'$  の場合、 $m$  の正確な値が事前にわからないため、上限値を使用する必要がある。

*ifail*

INTEGER。

配列、次元は  $\max(1, n)$  以上。

$jobz = 'V'$  の場合、 $info = 0$  であれば、 $ifail$  の先頭から  $m$  個の成分はゼロになる。 $info > 0$  であれば、収束しなかった固有ベクトルのインデックスが  $ifail$  に格納される。

$jobz = 'N'$  の場合、 $ifail$  は参照されない。

*info*

INTEGER。

$info = 0$  の場合、実行は正常に終了したことを示す。

$info = -i$  の場合、 $i$  番目の引数が不正であったことを示す。

$info > 0$  の場合、 $cpptrf/zpptrf$  と  $chpevx/zhpevx$  によって、エラーコードが返される。

$info = i \leq n$  の場合、 $chpevx/zhpevx$  が収束せず、 $i$  個の固有ベクトルが収束しなかったことを示す。これらのインデックスは、配列  $ifail$  に格納される。

$info = n + i$  ( $1 \leq i \leq n$ ) の場合、 $B$  について先頭から次数  $i$  の小行列が正定値でないことを示す。また、 $B$  の因子分解が完了できず、固有値または固有ベクトルが計算されなかった。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `hpgvx` のインターフェイスの詳細を以下に示す。

*a*

Fortran 77 インターフェイスでの引数 `ap` を意味する。

サイズ  $(n*(n+1)/2)$  の配列  $A$  を格納する。

<i>b</i>	Fortran 77 インターフェイスでの引数 <i>bp</i> を意味する。 サイズ $(n*(n+1)/2)$ の配列 <i>B</i> を格納する。
<i>w</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>z</i>	サイズ ( <i>n,m</i> ) の行列 <i>Z</i> を格納する。ここで、値 <i>n</i> と <i>m</i> は有意である。
<i>ifail</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>itype</i>	1、2、または 3 でなければならない。デフォルト値は 1。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>v1</i>	この成分のデフォルト値は、 $v1 = -HUGE(v1)$ である。
<i>vu</i>	この成分のデフォルト値は、 $vu = HUGE(v1)$ である。
<i>il</i>	この引数のデフォルト値は、 $il = 1$ である。
<i>iu</i>	この引数のデフォルト値は、 $iu = n$ である。
<i>abstol</i>	この成分のデフォルト値は、 $abstol = 0.0\_WP$ である。
<i>jobz</i>	引数 <i>z</i> の存在に基づいて以下のように復元される。 $jobz = 'V'$ ( <i>z</i> が存在する場合)、 $jobz = 'N'$ ( <i>z</i> が省略された場合)。 <i>ifail</i> が存在し、 <i>z</i> が省略された場合、エラー条件がセットされる。
<i>range</i>	引数 <i>v1</i> 、 <i>vu</i> 、 <i>il</i> 、および <i>iu</i> の存在に基づいて以下のように復元される。 $range = 'V'$ ( <i>v1</i> と <i>vu</i> のいずれかまたは両方が存在する場合)、 $range = 'I'$ ( <i>il</i> と <i>iu</i> のいずれかまたは両方が存在する場合)、 $range = 'A'$ ( <i>v1</i> 、 <i>vu</i> 、 <i>il</i> 、および <i>iu</i> がすべて存在しない場合)。 <i>v1</i> と <i>vu</i> のいずれかまたは両方が存在し、同時に <i>il</i> と <i>iu</i> のいずれかまたは両方が存在する場合、エラー条件がセットされる。

## アプリケーション・ノート

$abstol + \varepsilon * \max(|a|, |b|)$  以下の幅の区間  $[a, b]$  内に存在していると判別された場合 ( $\varepsilon$  はマシン精度)、近似固有値は、収束したものとして受け入れられる。 $abstol$  がゼロ以下の場合、 $\varepsilon * \|T\|_1$  は所定の場所で使用される ( $T$  は、 $A$  を三重対角形式に縮退して獲得した三重対角行列)。

固有値が最も正確に計算されるのは、 $abstol$  がゼロでなく、アンダーフローのしきい値の 2 倍、 $2 * ?lamch('S')$  に設定されたときである。このルーチンで  $info > 0$  が返された場合、固有ベクトルのいくつか収束に失敗したことを意味するので、 $abstol$  を  $2 * ?lamch('S')$  に設定して、やり直してみる。



## ?sbgv

帯行列に関する実数の汎用対称固有値問題について、固有値と(オプションで)固有ベクトルをすべて計算する。

### 構文

#### Fortran 77:

```
call ssbgv(jobz, uplo, n, ka, kb, ab, ldab, bb, ldbb, w, z, ldz, work, info)
call dsbgv(jobz, uplo, n, ka, kb, ab, ldab, bb, ldbb, w, z, ldz, work, info)
```

#### Fortran 95:

```
call sbgv(a, b, w [,uplo] [,z] [,info])
```

### 説明

このルーチンは、 $Ax = \lambda Bx$  の形式で示される帯行列に関する実数の汎用対称固有値問題について、固有値と(オプションで)固有ベクトルをすべて計算する。 $A$  と  $B$  は対称で帯であると仮定する。また、 $B$  は正定値である。

### 入力パラメータ

<i>jobz</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobz</i> ='N' の場合、固有値のみを計算する。 <i>jobz</i> ='V' の場合、固有値と固有ベクトルを計算する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> ='U' の場合、配列 <i>ab</i> と <i>bb</i> は、 $A$ と $B$ の上三角を格納する。 <i>uplo</i> ='L' の場合、配列 <i>ab</i> と <i>bb</i> は、 $A$ と $B$ の下三角を格納する。
<i>n</i>	INTEGER。行列 $A$ と $B$ の次数 ( $n \geq 0$ )。
<i>ka</i>	INTEGER。 $A$ の優対角成分または劣対角成分の数 ( $ka \geq 0$ )。
<i>kb</i>	INTEGER。 $B$ の優対角成分または劣対角成分の数 ( $kb \geq 0$ )。
<i>ab,bb,work</i>	REAL (ssbgv の場合) DOUBLE PRECISION (dsbgv の場合) 配列:

$ab(ldab,*)$  は、( $uplo$  の値に従って) 対称行列  $A$  の上三角部分または下三角部分を帯格納形式で格納する配列である。

配列  $ab$  の第 2 次元は、 $\max(1, n)$  以上でなければならない。

$bb(lddb,*)$  は、( $uplo$  の値に従って) 対称行列  $B$  の上三角部分または下三角部分を帯格納形式で格納する配列である。

配列  $bb$  の第 2 次元は、 $\max(1, n)$  以上でなければならない。

$work(*)$  はワークスペース配列、次元は  $\max(1, 3n)$  以上。

$ldab$  INTEGER。配列  $ab$  の第 1 次元。  $ka+1$  以上でなければならない。

$ldbb$  INTEGER。配列  $bb$  の第 1 次元。  $kb+1$  以上でなければならない。

$ldz$  INTEGER。出力配列  $z$  のリーディング・ディメンジョン。  $ldz \geq 1$ 。  
 $jobz = 'V'$  の場合、  $ldz \geq \max(1, n)$ 。

## 出力パラメータ

$ab$  終了時に、 $ab$  の内容は上書きされる。

$bb$  終了時に、[spbstf/dpbstf](#) によって返された分割コレスキー因子分解  $B = S^T S$  からの係数  $S$  が格納される。

$w, z$  REAL (ssbgv の場合)  
DOUBLE PRECISION (dsbgv の場合)  
配列:  
 $w(*)$ 、次元は  $\max(1, n)$  以上。  
 $info = 0$  の場合、固有値が昇順で格納される。

$z(ldz,*)$ 。 $z$  の第 2 次元は、 $\max(1, n)$  以上でなければならない。  
 $jobz = 'V'$  を指定した場合に  $info = 0$  であると、 $z$  には、 $w(i)$  に関連する固有ベクトルを格納している  $z$  の  $i$  番目の列とともに、固有ベクトルの行列  $Z$  が格納される。固有ベクトルは、 $Z^T B Z = I$  のように正規化される。  
 $jobz = 'N'$  の場合、 $z$  は参照されない。

$info$  INTEGER。  
 $info = 0$  の場合、実行は正常に終了したことを示す。  
 $info = -i$  の場合、 $i$  番目の引数が不正であったことを示す。  
 $info > 0$  で、  
 $i \leq n$  の場合、アルゴリズムが収束せず、中間三重対角の  $i$  個の対角外の成分がゼロに収束しなかったことを示す。  
 $info = n + i$  ( $1 \leq i \leq n$ ) の場合、[spbstf/dpbstf](#) によって  $info = i$  が返され、 $B$  が正定値でないことを示す。また、 $B$  の因子分解が完了できず、固有値または固有ベクトルが計算されなかった。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `sbgv` のインターフェイスの詳細を以下に示す。

<code>a</code>	Fortran 77 インターフェイスでの引数 <code>ab</code> を意味する。サイズ $(ka+1, n)$ の配列 $A$ を格納する。
<code>b</code>	Fortran 77 インターフェイスでの引数 <code>bb</code> を意味する。サイズ $(kd+1, n)$ の配列 $B$ を格納する。
<code>w</code>	長さ $(n)$ のベクトルを格納する。
<code>z</code>	サイズ $(n, n)$ の行列 $Z$ を格納する。
<code>uplo</code>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<code>jobz</code>	引数 $z$ の存在に基づいて以下のように復元される。 <code>jobz = 'V'</code> ( $z$ が存在する場合)、 <code>jobz = 'N'</code> ( $z$ が省略された場合)。

---

## ?hbgv

帯行列に関する複素数の汎用エルミート固有値問題について、固有値と(オプションで)固有ベクトルをすべて計算する。

---

### 構文

#### Fortran 77:

```
call chbgv(jobz, uplo, n, ka, kb, ab, ldab, bb, ldbb, w, z, ldz, work, rwork, info)
call zhbgv(jobz, uplo, n, ka, kb, ab, ldab, bb, ldbb, w, z, ldz, work, rwork, info)
```

#### Fortran 95:

```
call hbgv(a, b, w [,uplo] [,z] [,info])
```

## 説明

このルーチンは、 $Ax = \lambda Bx$  の形式で示される帯行列に関する複素数の汎用エルミート固有値問題について、固有値と (オプションで) 固有ベクトルをすべて計算する。 $A$  と  $B$  はエルミートで帯であると仮定する。また、 $B$  は正定値である。

## 入力パラメータ

<i>jobz</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobz</i> ='N' の場合、固有値のみを計算する。 <i>jobz</i> ='V' の場合、固有値と固有ベクトルを計算する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> ='U' の場合、配列 <i>ab</i> と <i>bb</i> は、 $A$ と $B$ の上三角を格納する。 <i>uplo</i> ='L' の場合、配列 <i>ab</i> と <i>bb</i> は、 $A$ と $B$ の下三角を格納する。
<i>n</i>	INTEGER。行列 $A$ と $B$ の次数 ( $n \geq 0$ )。
<i>ka</i>	INTEGER。 $A$ の優対角成分または劣対角成分の数 ( $ka \geq 0$ )。
<i>kb</i>	INTEGER。 $B$ の優対角成分または劣対角成分の数 ( $kb \geq 0$ )。
<i>ab,bb,work</i>	COMPLEX (chbgv の場合 ) DOUBLE COMPLEX (zhbgv の場合 ) 配列 : <i>ab</i> ( <i>ldab</i> ,*) は、( <i>uplo</i> の値に従って ) エルミート行列 $A$ の上三角部分または下三角部分を帯格納形式で格納する配列である。 配列 <i>ab</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>bb</i> ( <i>ldbb</i> ,*) は、( <i>uplo</i> の値に従って ) エルミート行列 $B$ の上三角部分または下三角部分を帯格納形式で格納する配列である。 配列 <i>bb</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>work</i> (*) は、ワークスペース配列、次元は $\max(1, n)$ 以上。
<i>ldab</i>	INTEGER。配列 <i>ab</i> の第 1 次元。 $ka+1$ 以上でなければならない。
<i>ldbb</i>	INTEGER。配列 <i>bb</i> の第 1 次元。 $kb+1$ 以上でなければならない。
<i>ldz</i>	INTEGER。出力配列 <i>z</i> のリーディング・ディメンジョン。 $ldz \geq 1$ 。 <i>jobz</i> ='V' の場合、 $ldz \geq \max(1, n)$ 。
<i>rwork</i>	REAL (chbgv の場合 ) DOUBLE PRECISION (zhbgv の場合 )。 ワークスペース配列、次元は $\max(1, 3n)$ 以上。

## 出力パラメータ

<i>ab</i>	終了時に、 <i>ab</i> の内容は上書きされる。
<i>bb</i>	終了時に、 <a href="#">cpbstf/zpbstf</a> によって返された分割コレスキー因子分解 $B = S^H S$ からの係数 $S$ が格納される。
<i>w</i>	REAL (chbgv の場合 ) DOUBLE PRECISION (zhbgv の場合 )。 配列、次元は $\max(1, n)$ 以上。 <i>info</i> = 0 の場合、固有値が昇順で格納される。
<i>z</i>	COMPLEX (chbgv の場合 ) DOUBLE COMPLEX (zhbgv の場合 ) 配列 <i>z</i> ( <i>ldz</i> ,*)。 <i>z</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>jobz</i> = 'V' を指定した場合に <i>info</i> = 0 であると、 <i>z</i> には、 <i>w</i> ( <i>i</i> ) に関連する固有ベクトルを格納している <i>z</i> の <i>i</i> 番目の列とともに、固有ベクトルの行列 $Z$ が格納される。固有ベクトルは、 $Z^H B Z = I$ のように正規化される。 <i>jobz</i> = 'N' の場合、 <i>z</i> は参照されない。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目の引数が不正であったことを示す。 <i>info</i> > 0 で、  <i>i</i> ≤ <i>n</i> の場合、アルゴリズムが収束せず、中間三重対角の <i>i</i> 個の対角外の成分がゼロに収束しなかったことを示す。 <i>info</i> = <i>n</i> + <i>i</i> (1 ≤ <i>i</i> ≤ <i>n</i> ) の場合、 <a href="#">cpbstf/zpbstf</a> によって <i>info</i> = <i>i</i> が返され、 $B$ が正定値でないことを示す。また、 $B$ の因子分解が完了できず、固有値または固有ベクトルが計算されなかった。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン hbgv のインターフェイスの詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ab</i> を意味する。サイズ ( <i>ka</i> +1, <i>n</i> ) の配列 $A$ を格納する。
<i>b</i>	Fortran 77 インターフェイスでの引数 <i>bb</i> を意味する。サイズ ( <i>kd</i> +1, <i>n</i> ) の配列 $B$ を格納する。

<code>w</code>	長さ ( $n$ ) のベクトルを格納する。
<code>z</code>	サイズ ( $n,n$ ) の行列 $Z$ を格納する。
<code>uplo</code>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<code>jobz</code>	引数 $z$ の存在に基づいて以下のように復元される。 <code>jobz = 'V'</code> ( $z$ が存在する場合)、 <code>jobz = 'N'</code> ( $z$ が省略された場合)。

---

### ?sbgvd

帯行列に関する実数の汎用対称固有値問題について、固有値と (オプションで) 固有ベクトルをすべて計算する。固有ベクトルを計算する場合は、分割統治法を使用する。

---

#### 構文

##### Fortran 77:

```
call ssbgvd(jobz, uplo, n, ka, kb, ab, ldab, bb, ldbb, w, z, ldz, work, lwork,  
            iwork, liwork, info)  
call dsbgvd(jobz, uplo, n, ka, kb, ab, ldab, bb, ldbb, w, z, ldz, work, lwork,  
            iwork, liwork, info)
```

##### Fortran 95:

```
call sbgvd(a, b, w [,uplo] [,z] [,info])
```

#### 説明

このルーチンは、 $Ax = \lambda Bx$  の形式で示される帯行列に関する実数の汎用対称固有値問題について、固有値と (オプションで) 固有ベクトルをすべて計算する。 $A$  と  $B$  は対称で帯であると仮定する。また、 $B$  は正定値である。固有ベクトルを計算する場合は、分割統治法を使用する。

#### 入力パラメータ

`jobz` CHARACTER\*1。'N' または 'V' でなければならない。  
`jobz = 'N'` の場合、固有値のみを計算する。  
`jobz = 'V'` の場合、固有値と固有ベクトルを計算する。

<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> ='U' の場合、配列 <i>ab</i> と <i>bb</i> は、 <i>A</i> と <i>B</i> の上三角を格納する。 <i>uplo</i> ='L' の場合、配列 <i>ab</i> と <i>bb</i> は、 <i>A</i> と <i>B</i> の下三角を格納する。
<i>n</i>	INTEGER。行列 <i>A</i> と <i>B</i> の次数 ( $n \geq 0$ )。
<i>ka</i>	INTEGER。 <i>A</i> の優対角成分または劣対角成分の数 ( $ka \geq 0$ )。
<i>kb</i>	INTEGER。 <i>B</i> の優対角成分または劣対角成分の数 ( $kb \geq 0$ )。
<i>ab,bb,work</i>	REAL (ssbgvd の場合) DOUBLE PRECISION (dsbgvd の場合) 配列： <i>ab</i> ( <i>ldab</i> ,*) は、( <i>uplo</i> の値に従って) 対称行列 <i>A</i> の上三角部分または下三角部分を帯格納形式で格納する配列である。 配列 <i>ab</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。  <i>bb</i> ( <i>ldbb</i> ,*) は、( <i>uplo</i> の値に従って) 対称行列 <i>B</i> の上三角部分または下三角部分を帯格納形式で格納する配列である。 配列 <i>bb</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。  <i>work</i> ( <i>lwork</i> ) は、ワークスペース配列である。
<i>ldab</i>	INTEGER。配列 <i>ab</i> の第 1 次元。 $ka+1$ 以上でなければならない。
<i>ldbb</i>	INTEGER。配列 <i>bb</i> の第 1 次元。 $kb+1$ 以上でなければならない。
<i>ldz</i>	INTEGER。出力配列 <i>z</i> のリーディング・ディメンジョン。 $ldz \geq 1$ 。 <i>jobz</i> ='V' の場合、 $ldz \geq \max(1, n)$ 。
<i>lwork</i>	INTEGER。配列 <i>work</i> の次元。  次の制約がある。 $n \leq 1$ の場合、 $lwork \geq 1$ 。 <i>jobz</i> ='N' で $n > 1$ の場合、 $lwork \geq 3n$ 。 <i>jobz</i> ='V' で $n > 1$ の場合、 $lwork \geq 2n^2+5n+1$ 。  $lwork = -1$ の場合、ワークスペースのクエリとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリとして返す。xerbla は <i>lwork</i> に関するエラー・メッセージを生成しない。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は ( <i>liwork</i> )。
<i>liwork</i>	INTEGER。配列 <i>iwork</i> の次元。 次の制約がある。 $n \leq 1$ の場合、 $liwork \geq 1$ 。

$jobz = 'N'$  で  $n > 1$  の場合、 $liwork \geq 1$ 。  
 $jobz = 'V'$  で  $n > 1$  の場合、 $liwork \geq 5n+3$ 。

$liwork = -1$  の場合、ワークスペースのクエリとみなされ、ルーチンは  $iwork$  配列の最適サイズだけを計算し、その値を  $iwork$  配列の最初のエントリとして返す。xerbla は  $liwork$  に関するエラー・メッセージを生成しない。

## 出力パラメータ

$ab$	終了時に、 $ab$ の内容は上書きされる。
$bb$	終了時に、 <a href="#">spbstf/dpbstf</a> によって返された分割コレスキー因子分解 $B = S^T S$ からの係数 $S$ が格納される。
$w, z$	REAL (ssbgvd の場合) DOUBLE PRECISION (dsbgvd の場合) 配列: $w(*)$ 、次元は $\max(1, n)$ 以上。 $info = 0$ の場合、固有値が昇順で格納される。  $z(ldz, *)$ 。 $z$ の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $jobz = 'V'$ を指定した場合に $info = 0$ であると、 $z$ には、 $w(i)$ に関連する固有ベクトルを格納している $z$ の $i$ 番目の列とともに、固有ベクトルの行列 $Z$ が格納される。固有ベクトルは、 $Z^T B Z = I$ のように正規化される。 $jobz = 'N'$ の場合、 $z$ は参照されない。
$work(1)$	終了時に、 $info = 0$ の場合、 $work(1)$ は $lwork$ の必要最小サイズを返す。
$iwork(1)$	終了時に、 $info = 0$ の場合、 $iwork(1)$ は $liwork$ の必要最小サイズを返す。
$info$	INTEGER。 $info = 0$ の場合、実行は正常に終了したことを示す。 $info = -i$ の場合、 $i$ 番目の引数が不正であったことを示す。 $info > 0$ で、  $i \leq n$ の場合、アルゴリズムが収束せず、中間三重対角の $i$ 個の対角外の成分がゼロに収束しなかったことを示す。 $info = n + i$ ( $1 \leq i \leq n$ ) の場合、 <a href="#">spbstf/dpbstf</a> によって $info = i$ が返され、 $B$ が正定値でないことを示す。また、 $B$ の因子分解が完了できず、固有値または固有ベクトルが計算されなかった。



**Fortran 95 インターフェイス・ノート**

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `sbgvd` のインターフェイスの詳細を以下に示す。

<code>a</code>	Fortran 77 インターフェイスでの引数 <code>ab</code> を意味する。サイズ $(ka+1, n)$ の配列 $A$ を格納する。
<code>b</code>	Fortran 77 インターフェイスでの引数 <code>bb</code> を意味する。サイズ $(kd+1, n)$ の配列 $B$ を格納する。
<code>w</code>	長さ $(n)$ のベクトルを格納する。
<code>z</code>	サイズ $(n, n)$ の行列 $Z$ を格納する。
<code>uplo</code>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<code>jobz</code>	引数 $z$ の存在に基づいて以下のように復元される。 <code>jobz = 'V'</code> ( $z$ が存在する場合)、 <code>jobz = 'N'</code> ( $z$ が省略された場合)。

**?hbgvd**

帯行列に関する複素数の汎用エルミート固有値問題について、固有値と(オプションで)固有ベクトルをすべて計算する。固有ベクトルを計算する場合は、分割統治法を使用する。

**構文****Fortran 77:**

```
call chbgvd(jobz, uplo, n, ka, kb, ab, ldab, bb, ldbb, w, z, ldz, work, lwork,
           rwork, lrwork, iwork, liwork, info)
call zhbgvd(jobz, uplo, n, ka, kb, ab, ldab, bb, ldbb, w, z, ldz, work, lwork,
           rwork, lrwork, iwork, liwork, info)
```

**Fortran 95:**

```
call hbgvd(a, b, w [,uplo] [,z] [,info])
```

## 説明

このルーチンは、 $Ax = \lambda Bx$  の形式で示される帯行列に関する複素数の汎用エルミート固有値問題について、固有値と (オプションで) 固有ベクトルをすべて計算する。 $A$  と  $B$  はエルミートで帯であると仮定する。また、 $B$  は正定値である。固有ベクトルを計算する場合は、分割統治法を使用する。

## 入力パラメータ

<i>jobz</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobz</i> ='N' の場合、固有値のみを計算する。 <i>jobz</i> ='V' の場合、固有値と固有ベクトルを計算する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> ='U' の場合、配列 <i>ab</i> と <i>bb</i> は、 $A$ と $B$ の上三角を格納する。 <i>uplo</i> ='L' の場合、配列 <i>ab</i> と <i>bb</i> は、 $A$ と $B$ の下三角を格納する。
<i>n</i>	INTEGER。行列 $A$ と $B$ の次数 ( $n \geq 0$ )。
<i>ka</i>	INTEGER。 $A$ の優対角成分または劣対角成分の数 ( $ka \geq 0$ )。
<i>kb</i>	INTEGER。 $B$ の優対角成分または劣対角成分の数 ( $kb \geq 0$ )。
<i>ab,bb,work</i>	COMPLEX (chbgvd の場合) DOUBLE COMPLEX (zhbgvd の場合) 配列: <i>ab</i> ( <i>ldab</i> ,*) は、( <i>uplo</i> の値に従って) エルミート行列 $A$ の上三角部分または下三角部分を帯格納形式で格納する配列である。 配列 <i>ab</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。  <i>bb</i> ( <i>ldbb</i> ,*) は、( <i>uplo</i> の値に従って) エルミート行列 $B$ の上三角部分または下三角部分を帯格納形式で格納する配列である。 配列 <i>bb</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。  <i>work</i> ( <i>lwork</i> ) は、ワークスペース配列である。
<i>ldab</i>	INTEGER。配列 <i>ab</i> の第 1 次元。 $ka+1$ 以上でなければならない。
<i>ldbb</i>	INTEGER。配列 <i>bb</i> の第 1 次元。 $kb+1$ 以上でなければならない。
<i>ldz</i>	INTEGER。出力配列 <i>z</i> のリーディング・ディメンジョン。 $ldz \geq 1$ 。 <i>jobz</i> ='V' の場合、 $ldz \geq \max(1, n)$ 。
<i>lwork</i>	INTEGER。配列 <i>work</i> の次元。

次の制約がある。

$n \leq 1$  の場合、 $lwork \geq 1$ 。

$jobz = 'N'$  で  $n > 1$  の場合、 $lwork \geq n$ 。

$jobz = 'V'$  で  $n > 1$  の場合、 $lwork \geq 2n^2$ 。

$lwork = -1$  の場合、ワークスペースのクエリとみなされ、ルーチンは  $work$  配列の最適サイズだけを計算し、その値を  $work$  配列の最初のエントリとして返す。xerbla は  $lwork$  に関するエラー・メッセージを生成しない。

*rwork*      REAL (chbgvd の場合 )  
DOUBLE PRECISION (zhbgvd の場合 )。  
ワークスペース配列、次元は (*lrwork*)。

*lrwork*      INTEGER。配列 *rwork* の次元。

次の制約がある。

$n \leq 1$  の場合、 $lrwork \geq 1$ 。

$jobz = 'N'$  で  $n > 1$  の場合、 $lrwork \geq n$ 。

$jobz = 'V'$  で  $n > 1$  の場合、 $lrwork \geq 2n^2 + 5n + 1$ 。

$lrwork = -1$  の場合、ワークスペースのクエリとみなされ、ルーチンは  $rwork$  配列の最適サイズだけを計算し、その値を  $rwork$  配列の最初のエントリとして返す。xerbla は  $lrwork$  に関するエラー・メッセージを生成しない。

*iwork*      INTEGER。  
ワークスペース配列、次元は (*liwork*)。

*liwork*      INTEGER。配列 *iwork* の次元。

次の制約がある。

$n \leq 1$  の場合、 $liwork \geq 1$ 。

$jobz = 'N'$  で  $n > 1$  の場合、 $liwork \geq 1$ 。

$jobz = 'V'$  で  $n > 1$  の場合、 $liwork \geq 5n + 3$ 。

$liwork = -1$  の場合、ワークスペースのクエリとみなされ、ルーチンは  $iwork$  配列の最適サイズだけを計算し、その値を  $iwork$  配列の最初のエントリとして返す。xerbla は  $liwork$  に関するエラー・メッセージを生成しない。

## 出力パラメータ

*ab*              終了時に、*ab* の内容は上書きされる。

<i>bb</i>	終了時に、 <a href="#">cpbstf/zpbstf</a> によって返された分割コレスキー因子分解 $B = S^H S$ からの係数 $S$ が格納される。
<i>w</i>	REAL (chbgvd の場合 ) DOUBLE PRECISION (zhbgvd の場合 )。 配列、次元は $\max(1, n)$ 以上。 $info = 0$ の場合、固有値が昇順で格納される。
<i>z</i>	COMPLEX (chbgvd の場合 ) DOUBLE COMPLEX (zhbgvd の場合 ) 配列 $z(ldz, *)$ 。 $z$ の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $jobz = 'V'$ の場合、 $info = 0$ であれば、 $z$ には、 $w(i)$ に関連する固有ベクトルを格納している $z$ の $i$ 番目の列とともに、固有ベクトルの行列 $Z$ が格納される。固有ベクトルは、 $Z^H B Z = I$ のように正規化される。 $jobz = 'N'$ の場合、 $z$ は参照されない。
<i>work(1)</i>	終了時に、 $info = 0$ の場合、 <i>work(1)</i> は <i>lwork</i> の必要最小サイズを返す。
<i>rwork(1)</i>	終了時に、 $info = 0$ の場合、 <i>rwork(1)</i> は、 <i>lrwork</i> の必要最小サイズを返す。
<i>iwork(1)</i>	終了時に、 $info = 0$ の場合、 <i>iwork(1)</i> は <i>liwork</i> の必要最小サイズを返す。
<i>info</i>	INTEGER。 $info = 0$ の場合、実行は正常に終了したことを示す。 $info = -i$ の場合、 $i$ 番目の引数が不正であったことを示す。 $info > 0$ で、 $i \leq n$ の場合、アルゴリズムが収束せず、中間三重対角の $i$ 個の対角外の成分がゼロに収束しなかったことを示す。 $info = n + i$ ( $1 \leq i \leq n$ ) の場合、 <a href="#">cpbstf/zpbstf</a> によって $info = i$ が返され、 $B$ が正定値でないことを示す。また、 $B$ の因子分解が完了できず、固有値または固有ベクトルが計算されなかった。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン hbgvd のインターフェイスの詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ab</i> を意味する。サイズ $(ka+1, n)$ の配列 <i>A</i> を格納する。
<i>b</i>	Fortran 77 インターフェイスでの引数 <i>bb</i> を意味する。サイズ $(kd+1, n)$ の配列 <i>B</i> を格納する。
<i>w</i>	長さ $(n)$ のベクトルを格納する。
<i>z</i>	サイズ $(n, n)$ の行列 <i>Z</i> を格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>jobz</i>	引数 <i>z</i> の存在に基づいて以下のように復元される。 <i>jobz</i> = 'V' ( <i>z</i> が存在する場合)、 <i>jobz</i> = 'N' ( <i>z</i> が省略された場合)。

## ?sbgvx

帯行列に関する実数の汎用対称固有値問題について、固有値と(オプションで)固有ベクトルを選択的に計算する。

### 構文

#### Fortran 77:

```
call ssbgvx(jobz, range, uplo, n, ka, kb, ab, ldab, bb, ldbb, q, ldq, vl, vu,
            il, iu, abstol, m, w, z, ldz, work, iwork, ifail, info)
call dsbgvx(jobz, range, uplo, n, ka, kb, ab, ldab, bb, ldbb, q, ldq, vl, vu,
            il, iu, abstol, m, w, z, ldz, work, iwork, ifail, info)
```

#### Fortran 95:

```
call sbgvx(a, b, w [,uplo] [,z] [,vl] [,vu] [,il] [,iu] [,m] [,ifail] [,q]
            [,abstol] [,info])
```

### 説明

このルーチンは、 $Ax = \lambda Bx$  の形式で示される帯行列に関する実数の汎用対称固有値問題について、固有値と(オプションで)固有ベクトルを選択的に計算する。*A* と *B* は対称で帯であると仮定する。また、*B* は正定値である。  
固有値と固有ベクトルを選択するには、希望する固有値について、すべての固有値、値の範囲、またはインデックスの範囲を指定する。

## 入力パラメータ

<i>jobz</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobz</i> ='N' の場合、固有値のみを計算する。 <i>jobz</i> ='V' の場合、固有値と固有ベクトルを計算する。
<i>range</i>	CHARACTER*1。'A'、'V'、または 'I' のいずれかでなければならない。 <i>range</i> ='A' の場合、固有値をすべて計算する。 <i>range</i> ='V' の場合、半開区間 $v_l < \lambda_i \leq v_u$ における固有値 $\lambda_i$ を計算する。 <i>range</i> ='I' の場合、ルーチンはインデックスが $i_l \sim i_u$ である固有値を計算する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> ='U' の場合、配列 <i>ab</i> と <i>bb</i> は、 <i>A</i> と <i>B</i> の上三角を格納する。 <i>uplo</i> ='L' の場合、配列 <i>ab</i> と <i>bb</i> は、 <i>A</i> と <i>B</i> の下三角を格納する。
<i>n</i>	INTEGER。行列 <i>A</i> と <i>B</i> の次数 ( $n \geq 0$ )。
<i>ka</i>	INTEGER。 <i>A</i> の優対角成分または劣対角成分の数 ( $ka \geq 0$ )。
<i>kb</i>	INTEGER。 <i>B</i> の優対角成分または劣対角成分の数 ( $kb \geq 0$ )。
<i>ab,bb,work</i>	REAL (ssbgvd の場合 ) DOUBLE PRECISION (dsbgvd の場合 ) 配列 : <i>ab</i> ( <i>ldab</i> ,*) は、( <i>uplo</i> の値に従って ) 対称行列 <i>A</i> の上三角部分または下三角部分を帯格納形式で格納する配列である。 配列 <i>ab</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>bb</i> ( <i>ldbb</i> ,*) は、( <i>uplo</i> の値に従って ) 対称行列 <i>B</i> の上三角部分または下三角部分を帯格納形式で格納する配列である。 配列 <i>bb</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>work</i> (*) は、ワークスペース配列、次元は $\max(1, 7n)$ 以上。
<i>ldab</i>	INTEGER。配列 <i>ab</i> の第 1 次元。 $ka+1$ 以上でなければならない。
<i>ldbb</i>	INTEGER。配列 <i>bb</i> の第 1 次元。 $kb+1$ 以上でなければならない。
<i>v1, vu</i>	REAL (ssbgvx の場合 ) DOUBLE PRECISION (dsbgvx の場合 )。 <i>range</i> ='V' の場合、固有値を検索する区間の上限と下限。 次の制約がある。 $v_l < v_u$ 。 <i>range</i> ='A' または 'I' の場合、 <i>v1</i> と <i>vu</i> は参照されない。

<i>il, iu</i>	INTEGER。 <i>range</i> = 'I' の場合、返される最小固有値と最大固有値に対する昇順のインデックス。 次の制約がある。 $1 \leq il \leq iu \leq n$ ( $n > 0$ の場合)。 $il=1$ かつ $iu=0$ ( $n=0$ の場合)。 <i>range</i> = 'A' または 'V' の場合、 <i>il</i> と <i>iu</i> は参照されない。
<i>abstol</i>	REAL ( <i>ssbgvx</i> の場合) DOUBLE PRECISION ( <i>dsbgvx</i> の場合)。 固有値に対する絶対エラー許容値。 詳細は、「アプリケーション・ノート」を参照。
<i>ldz</i>	INTEGER。出力配列 <i>z</i> のリーディング・ディメンジョン。 $ldz \geq 1$ 。 <i>jobz</i> = 'V' の場合、 $ldz \geq \max(1, n)$ 。
<i>ldq</i>	INTEGER。出力配列 <i>q</i> のリーディング・ディメンジョン。 $ldq \geq 1$ 。 <i>jobz</i> = 'V' の場合、 $ldq \geq \max(1, n)$ 。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, 5n)$ 以上。

### 出力パラメータ

<i>ab</i>	終了時に、 <i>ab</i> の内容は上書きされる。
<i>bb</i>	終了時に、 <a href="#">spbstf</a> / <a href="#">dpbstf</a> によって返された分割コレスキー因子分解 $B = S^T S$ からの係数 <i>S</i> が格納される。
<i>m</i>	INTEGER。検出された固有値の総数。 ( $0 \leq m \leq n$ )。 <i>range</i> = 'A' の場合、 $m = n$ 、 <i>range</i> = 'I' の場合、 $m = iu - il + 1$ 。
<i>w, z, q</i>	REAL ( <i>ssbgvx</i> の場合) DOUBLE PRECISION ( <i>dsbgvx</i> の場合) 配列： <i>w</i> (*)、次元は $\max(1, n)$ 以上。 <i>info</i> = 0 の場合、固有値が昇順で格納される。  <i>z</i> ( <i>ldz</i> ,*)。 <i>z</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>jobz</i> = 'V' の場合、 <i>info</i> = 0 であれば、 <i>z</i> には、 <i>w</i> ( <i>i</i> ) に関連する固有ベクトルを格納している <i>z</i> の <i>i</i> 番目の列とともに、固有ベクトルの行列 <i>Z</i> が格納される。固有ベクトルは、 $Z^T B Z = I$ のように正規化される。 <i>jobz</i> = 'N' の場合、 <i>z</i> は参照されない。 <i>q</i> ( <i>ldq</i> ,*)。 <i>q</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>jobz</i> = 'V' の場合、 <i>q</i> には、 $Ax = \lambda Bx$ を標準形式、すなわち $Cx = \lambda x$ に

縮退させて、 $C$  を三重対角形式に縮退させるために使用する  $n \times n$  の行列が格納される。

$jobz = 'N'$  の場合、 $q$  は参照されない。

<i>ifail</i>	<p>INTEGER。</p> <p>配列、次元は <math>\max(1, n)</math> 以上。</p> <p><math>jobz = 'V'</math> の場合、<math>info = 0</math> であれば、<i>ifail</i> の先頭から <math>m</math> 個の成分はゼロになる。<math>info &gt; 0</math> であれば、収束しなかった固有ベクトルのインデックスが <i>ifail</i> に格納される。</p> <p><math>jobz = 'N'</math> の場合、<i>ifail</i> は参照されない。</p>
<i>info</i>	<p>INTEGER。</p> <p><math>info = 0</math> の場合、実行は正常に終了したことを示す。</p> <p><math>info = -i</math> の場合、<math>i</math> 番目の引数が不正であったことを示す。</p> <p><math>info &gt; 0</math> で、</p> <p><math>i \leq n</math> の場合、アルゴリズムが収束せず、中間三重対角の <math>i</math> 個の対角外の成分がゼロに収束しなかったことを示す。</p> <p><math>info = n + i</math> (<math>1 \leq i \leq n</math>) の場合、<a href="#">spbstf</a>/<a href="#">dpbstf</a> によって <math>info = i</math> が返され、<math>B</math> が正定値でないことを示す。また、<math>B</math> の因子分解が完了できず、固有値または固有ベクトルが計算されなかった。</p>

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `sbgvx` のインターフェイスの詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ab</i> を意味する。サイズ $(ka+1, n)$ の配列 $A$ を格納する。
<i>b</i>	Fortran 77 インターフェイスでの引数 <i>bb</i> を意味する。サイズ $(kd+1, n)$ の配列 $B$ を格納する。
<i>w</i>	長さ $(n)$ のベクトルを格納する。
<i>z</i>	サイズ $(n, n)$ の行列 $Z$ を格納する。
<i>ifail</i>	長さ $(n)$ のベクトルを格納する。
<i>q</i>	サイズ $(n, n)$ の行列 $Q$ を格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>v1</i>	この成分のデフォルト値は $v1 = -HUGE(v1)$ である。



<i>vu</i>	この成分のデフォルト値は、 $vu = \text{HUGE}(vl)$ である。
<i>il</i>	この引数のデフォルト値は、 $il = 1$ である。
<i>iu</i>	この引数のデフォルト値は、 $iu = n$ である。
<i>abstol</i>	この成分のデフォルト値は、 $abstol = 0.0\_WP$ である。
<i>jobz</i>	引数 $z$ の存在に基づいて以下のように復元される。 $jobz = 'V'$ ( $z$ が存在する場合)、 $jobz = 'N'$ ( $z$ が省略された場合)。 $ifail$ または $q$ が存在し、 $z$ が省略された場合、エラー条件がセットされる。
<i>range</i>	引数 $vl$ 、 $vu$ 、 $il$ 、および $iu$ の存在に基づいて以下のように復元される。 $range = 'V'$ ( $vl$ と $vu$ のいずれかまたは両方が存在する場合)、 $range = 'I'$ ( $il$ と $iu$ のいずれかまたは両方が存在する場合)、 $range = 'A'$ ( $vl$ 、 $vu$ 、 $il$ 、および $iu$ がすべて存在しない場合)。 $vl$ と $vu$ のいずれかまたは両方が存在し、同時に $il$ と $iu$ のいずれかまたは両方が存在する場合、エラー条件がセットされる。

### アプリケーション・ノート

$abstol + \epsilon * \max(|a|, |b|)$  以下の幅の区間  $[a, b]$  内に存在していると判別された場合 ( $\epsilon$  はマシン精度)、近似固有値は、収束したものとして受け入れられる。 $abstol$  がゼロ以下の場合、 $\epsilon * \|T\|_1$  は所定の場所で使用される ( $T$  は、 $A$  を三重対角形式に縮退して獲得した三重対角行列)。

固有値が最も正確に計算されるのは、 $abstol$  がゼロでなく、アンダーフローのしきい値の 2 倍、 $2 * \text{?lamch}('S')$  に設定されたときである。このルーチンで  $info > 0$  が返された場合、固有ベクトルのいくつか収束に失敗したことを意味するので、 $abstol$  を  $2 * \text{?lamch}('S')$  に設定して、やり直してみる。

### ?hbgvx

帯行列に関する複素数の汎用エルミート固有値問題について、固有値と ( オプションで ) 固有ベクトルを選択的に計算する。

---

#### 構文

##### Fortran 77:

```
call chbgvx(jobz, range, uplo, n, ka, kb, ab, ldab, bb, ldbb, q, ldq, vl, vu,  
            il, iu, abstol, m, w, z, ldz, work, rwork, iwork, ifail, info)  
call zhbgvx(jobz, range, uplo, n, ka, kb, ab, ldab, bb, ldbb, q, ldq, vl, vu,  
            il, iu, abstol, m, w, z, ldz, work, rwork, iwork, ifail, info)
```

##### Fortran 95:

```
call hbgvx(a, b, w [,uplo] [,z] [,vl] [,vu] [,il] [,iu] [,m] [,ifail] [,q]  
            [,abstol] [,info])
```

#### 説明

このルーチンは、 $Ax = \lambda Bx$  の形式で示される帯行列に関する複素数の汎用エルミート固有値問題について、固有値と ( オプションで ) 固有ベクトルを選択的に計算する。 $A$  と  $B$  はエルミートで帯であると仮定する。また、 $B$  は正定値である。

固有値と固有ベクトルを選択するには、希望する固有値について、すべての固有値、値の範囲、またはインデックスの範囲を指定する。

#### 入力パラメータ

jobz	CHARACTER*1。'N' または 'V' でなければならない。 jobz='N' の場合、固有値のみを計算する。 jobz='V' の場合、固有値と固有ベクトルを計算する。
range	CHARACTER*1。'A'、'V'、または 'I' のいずれかでなければならない。 range='A' の場合、固有値をすべて計算する。 range='V' の場合、半開区間 $vl < \lambda_i \leq vu$ における固有値 $\lambda_i$ を計算する。 range='I' の場合、ルーチンはインデックスが $il \sim iu$ である固有値を計算する。
uplo	CHARACTER*1。'U' または 'L' でなければならない。

	$uplo = 'U'$ の場合、配列 $ab$ と $bb$ は、 $A$ と $B$ の上三角を格納する。 $uplo = 'L'$ の場合、配列 $ab$ と $bb$ は、 $A$ と $B$ の下三角を格納する。
$n$	INTEGER。行列 $A$ と $B$ の次数 ( $n \geq 0$ )。
$ka$	INTEGER。 $A$ の優対角成分または劣対角成分の数 ( $ka \geq 0$ )。
$kb$	INTEGER。 $B$ の優対角成分または劣対角成分の数 ( $kb \geq 0$ )。
$ab, bb, work$	COMPLEX (chbgvx の場合 ) DOUBLE COMPLEX (zhbgvx の場合 ) 配列： $ab(ldab, *)$ は、( $uplo$ の値に従って ) エルミート行列 $A$ の上三角部分 または下三角部分を帯格納形式で格納する配列である。 配列 $ab$ の第 2 次元は、 $\max(1, n)$ 以上でなければならない。  $bb(lddb, *)$ は、( $uplo$ の値に従って ) エルミート行列 $B$ の上三角部分 または下三角部分を帯格納形式で格納する配列である。 配列 $bb$ の第 2 次元は、 $\max(1, n)$ 以上でなければならない。  $work(*)$ は、ワークスペース配列、次元は $\max(1, n)$ 以上。
$ldab$	INTEGER。配列 $ab$ の第 1 次元。 $ka+1$ 以上でなければならない。
$ldbb$	INTEGER。配列 $bb$ の第 1 次元。 $kb+1$ 以上でなければならない。
$vl, vu$	REAL (chbgvx の場合 ) DOUBLE PRECISION (zhbgvx の場合 )。 $range = 'V'$ の場合、固有値を検索する区間の上限と下限。 次の制約がある。 $vl < vu$ 。  $range = 'A'$ または $'I'$ の場合、 $vl$ と $vu$ は参照されない。
$il, iu$	INTEGER。 $range = 'I'$ の場合、返される最小固有値と最大固有値に対する昇順の インデックス。 次の制約がある。 $1 \leq il \leq iu \leq n$ ( $n > 0$ の場合 )。 $il=1$ かつ $iu=0$ ( $n=0$ の場合 )。 $range = 'A'$ または $'V'$ の場合、 $il$ と $iu$ は参照されない。
$abstol$	REAL (chbgvx の場合 ) DOUBLE PRECISION (zhbgvx の場合 )。 固有値に対する絶対エラー許容値。 詳細は、「アプリケーション・ノート」を参照。

<i>ldz</i>	INTEGER。出力配列 <i>z</i> のリーディング・ディメンジョン。 $ldz \geq 1$ 。 <i>jobz</i> = 'V' の場合、 $ldz \geq \max(1, n)$ 。
<i>ldq</i>	INTEGER。出力配列 <i>q</i> のリーディング・ディメンジョン。 $ldq \geq 1$ 。 <i>jobz</i> = 'V' の場合、 $ldq \geq \max(1, n)$ 。
<i>rwork</i>	REAL (chbgvx の場合 ) DOUBLE PRECISION (zhbgvx の場合 )。 ワークスペース配列、次元は $\max(1, 7n)$ 以上。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, 5n)$ 以上。

## 出力パラメータ

<i>ab</i>	終了時に、 <i>ab</i> の内容は上書きされる。
<i>bb</i>	終了時に、 <a href="#">cpbstf/zpbstf</a> によって返された分割コレスキー因子分解 $B = S^H S$ からの係数 <i>S</i> が格納される。
<i>m</i>	INTEGER。検出された固有値の総数。 ( $0 \leq m \leq n$ )。 <i>range</i> = 'A' の場合、 <i>m</i> = <i>n</i> 、 <i>range</i> = 'I' の場合、 <i>m</i> = <i>iu</i> - <i>il</i> + 1。
<i>w</i>	REAL (chbgvx の場合 ) DOUBLE PRECISION (zhbgvx の場合 )。 配列 <i>w</i> (*)、次元は $\max(1, n)$ 以上。 <i>info</i> = 0 の場合、固有値が昇順で格納される。
<i>z, q</i>	COMPLEX (chbgvx の場合 ) DOUBLE COMPLEX (zhbgvx の場合 ) 配列： <i>z</i> ( <i>ldz</i> ,*)。 <i>z</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>jobz</i> = 'V' を指定した場合に <i>info</i> = 0 であると、 <i>z</i> には、 <i>w</i> ( <i>i</i> ) に関連する固有ベクトルを格納している <i>z</i> の <i>i</i> 番目の列とともに、固有ベクトルの行列 <i>Z</i> が格納される。固有ベクトルは、 $Z^H B Z = I$ のように正規化される。 <i>jobz</i> = 'N' の場合、 <i>z</i> は参照されない。 <i>q</i> ( <i>ldq</i> ,*)。 <i>q</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>jobz</i> = 'V' の場合、 <i>q</i> には、 $Ax = \lambda Bx$ を標準形式、すなわち $Cx = \lambda x$ に縮退させて、 <i>C</i> を三重対角形式に縮退させるために使用する $n \times n$ の行列が格納される。 <i>jobz</i> = 'N' の場合、 <i>q</i> は参照されない。

<i>ifail</i>	INTEGER。 配列、次元は $\max(1, n)$ 以上。 <i>jobz</i> ='V' の場合、 <i>info</i> =0 であれば、 <i>ifail</i> の先頭から <i>m</i> 個の成分はゼロになる。 <i>info</i> >0 であれば、収束しなかった固有ベクトルのインデックスが <i>ifail</i> に格納される。 <i>jobz</i> ='N' の場合、 <i>ifail</i> は参照されない。
<i>info</i>	INTEGER。 <i>info</i> =0 の場合、実行は正常に終了したことを示す。 <i>info</i> =- <i>i</i> の場合、 <i>i</i> 番目の引数が不正であったことを示す。 <i>info</i> >0 で、  <i>i</i> ≤ <i>n</i> の場合、アルゴリズムが収束せず、中間三重対角の <i>i</i> 個の対角外の成分がゼロに収束しなかったことを示す。 <i>info</i> = <i>n</i> + <i>i</i> ( $1 \leq i \leq n$ ) の場合、 <a href="#">cpbstf/zpbstf</a> によって <i>info</i> = <i>i</i> が返され、 <i>B</i> が正定値でないことを示す。また、 <i>B</i> の因子分解が完了できず、固有値または固有ベクトルが計算されなかった。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン hbgvx のインターフェイスの詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ab</i> を意味する。サイズ ( <i>ka</i> +1, <i>n</i> ) の配列 <i>A</i> を格納する。
<i>b</i>	Fortran 77 インターフェイスでの引数 <i>bb</i> を意味する。サイズ ( <i>kd</i> +1, <i>n</i> ) の配列 <i>B</i> を格納する。
<i>w</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>z</i>	サイズ ( <i>n</i> , <i>n</i> ) の行列 <i>Z</i> を格納する。
<i>ifail</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>q</i>	サイズ ( <i>n</i> , <i>n</i> ) の行列 <i>Q</i> を格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>v1</i>	この成分のデフォルト値は、 <i>v1</i> = -HUGE( <i>v1</i> ) である。
<i>vu</i>	この成分のデフォルト値は、 <i>vu</i> = HUGE( <i>v1</i> ) である。
<i>il</i>	この引数のデフォルト値は、 <i>il</i> = 1 である。

<i>iu</i>	この引数のデフォルト値は、 <i>iu</i> = <i>n</i> である。
<i>abstol</i>	この成分のデフォルト値は、 <i>abstol</i> = 0.0_WP である。
<i>jobz</i>	引数 <i>z</i> の存在に基づいて以下のように復元される。 <i>jobz</i> = 'V' ( <i>z</i> が存在する場合)、 <i>jobz</i> = 'N' ( <i>z</i> が省略された場合)。 <i>ifail</i> または <i>q</i> が存在し、 <i>z</i> が省略された場合、エラー条件がセットされる。
<i>range</i>	引数 <i>vl</i> 、 <i>vu</i> 、 <i>il</i> 、および <i>iu</i> の存在に基づいて以下のように復元される。 <i>range</i> = 'V' ( <i>vl</i> と <i>vu</i> のいずれかまたは両方が存在する場合)、 <i>range</i> = 'I' ( <i>il</i> と <i>iu</i> のいずれかまたは両方が存在する場合)、 <i>range</i> = 'A' ( <i>vl</i> 、 <i>vu</i> 、 <i>il</i> 、および <i>iu</i> がすべて存在しない場合)。 <i>vl</i> と <i>vu</i> のいずれかまたは両方が存在し、同時に <i>il</i> と <i>iu</i> のいずれかまたは両方が存在する場合、エラー条件がセットされる。

## アプリケーション・ノート

$abstol + \epsilon * \max(|a|, |b|)$  以下の幅の区間  $[a, b]$  内に存在していると判別された場合 ( $\epsilon$  はマシン精度)、近似固有値は、収束したものとして受け入れられる。*abstol* がゼロ以下の場合、 $\epsilon * \|T\|_1$  は所定の場所で使用される (*T* は、*A* を三重対角形式に縮退して獲得した三重対角行列)。

固有値が最も正確に計算されるのは、*abstol* がゼロでなく、アンダーフローのしきい値の 2 倍、 $2 * ?lamch('S')$  に設定されたときである。このルーチンで *info* > 0 が返された場合、固有ベクトルのいくつか収束に失敗したことを意味するので、*abstol* を  $2 * ?lamch('S')$  に設定して、やり直してみる。

汎用非対称固有値問題

このセクションでは、汎用非対称固有値問題を解くために使用する LAPACK ドライバ  
ルーチンについて説明する。これらの問題を解くために呼び出すことが可能な[計算ルー  
チン](#)も参照のこと。

表 4-14 に Fortran-77 インターフェイスのすべてのドライバルーチンを示す。Fortran-95  
インターフェイスのルーチン名はそれぞれ、先頭の記号がない ([ルーチン命名規則](#)を参  
照)。

表 4-14      汎用非対称固有値問題を解くためのドライバルーチン

ルーチン名	機能
<a href="#">?qges</a>	非対称行列のペアについて、汎用固有値、Schur 形式、左 / 右の Schur ベ クトルを計算する。
<a href="#">?qgesx</a>	汎用固有値、Schur 形式と ( オプションで ) Schur ベクトルの左 / 右の行列 を計算する。
<a href="#">?qgeev</a>	非対称行列のペアについて、汎用固有値、左 / 右の汎用固有ベクトルを計 算する。
<a href="#">?qgevx</a>	汎用固有値と ( オプションで ) 左 / 右の汎用固有ベクトルを計算する。

?qges

非対称行列のペアについて、汎用固有値、Schur  
形式、左 / 右の Schur ベクトルを計算する。

構文

**Fortran 77:**

```
call sgges(jobvsl, jobvsr, sort, selctg, n, a, lda, b, ldb, sdim, alphas,
           alphas, beta, vsl, ldvsl, vsr, ldvsr, work, lwork, bwork, info)
call dgges(jobvsl, jobvsr, sort, selctg, n, a, lda, b, ldb, sdim, alphas,
           alphas, beta, vsl, ldvsl, vsr, ldvsr, work, lwork, bwork, info)
call cgges(jobvsl, jobvsr, sort, selctg, n, a, lda, b, ldb, sdim, alpha, beta,
           vsl, ldvsl, vsr, ldvsr, work, lwork, rwork, bwork, info)
call zgges(jobvsl, jobvsr, sort, selctg, n, a, lda, b, ldb, sdim, alpha, beta,
           vsl, ldvsl, vsr, ldvsr, work, lwork, rwork, bwork, info)
```

## Fortran 95:

```
call gges(a, b, alphas, alphas, beta [,vs1] [,vsr] [,select] [,sdim] [,info])
call gges(a, b, alpha, beta [,vs1] [,vsr] [,select] [,sdim] [,info])
```

## 説明

このルーチンは、 $n \times n$  の実 / 複素非対称行列  $(A, B)$  のペアについて、汎用固有値、汎用実数 / 複素数 Schur 形式  $(S, T)$  と (オプションで) Schur ベクトルの左 / 右の行列 ( $vs1$  と  $vsr$ ) を計算する。これにより、汎用 Schur 因子分解

$$(A, B) = (vs1 * S * vsr^H, vs1 * T * vsr^H)$$

が提供される。

また、このルーチンは、固有値の選択した束が上準三角行列  $S$  と上三角行列  $T$  の主対角ブロックにくるように固有値を順序付けられる。 $vs1$  と  $vsr$  の先頭の列は、対応する左と右の固有空間 (収縮部分空間) に対する直交 / ユニタリ基を形成する。

(汎用固有値だけが必要な場合、より高速なドライバ [zggev](#) を代わりに使用できる)。行列  $(A, B)$  のペアの汎用固有値は、 $A - w * B$  が特異であるようなスカラー  $w$  または比  $alpha / beta = w$  である。 $beta=0$  または両方がゼロである妥当な解釈が存在するため、通常は、ペア  $(alpha, beta)$  として表現される。

$T$  が非負の対角を持つ上三角で  $S$  が  $1 \times 1$  と  $2 \times 2$  のブロックを持つブロック上三角である場合、行列  $(S, T)$  のペアは、汎用実数 Schur 形式である。 $1 \times 1$  ブロックは実数の汎用固有値に対応するが、 $S$  の

$2 \times 2$  ブロックは、 $T$  の対応する成分を次の形式にすることで「標準化」される。

$$\begin{pmatrix} a & 0 \\ 0 & b \end{pmatrix}$$

また、 $S$  と  $T$  の対応する  $2 \times 2$  ブロックのペアは、汎用固有値の複素共役ペアを持つ。 $S$  と  $T$  が上三角であり、 $T$  の対角が非負の実数の場合、行列  $(S, T)$  のペアは、汎用複素数 Schur 形式になる。

## 入力パラメータ

<code>jobvs1</code>	CHARACTER*1。'N' または 'V' でなければならない。 <code>jobvs1='N'</code> の場合、左 Schur ベクトルは計算されない。 <code>jobvs1='V'</code> の場合、左 Schur ベクトルは計算される。
<code>jobvsr</code>	CHARACTER*1。'N' または 'V' でなければならない。 <code>jobvsr='N'</code> の場合、右 Schur ベクトルは計算されない。 <code>jobvsr='V'</code> の場合、右 Schur ベクトルは計算される。



<i>sort</i>	<p>CHARACTER*1。'N' または 'S' でなければならない。 汎用 Schur 形式の対角上の固有値を順序付けるかどうかを指定する。</p> <p><i>sort</i>='N' の場合、固有値は順序付けされない。 <i>sort</i>='S' の場合、固有値は順序付けされる (<i>select</i> を参照)。</p>
<i>selctg</i>	<p>3 つの REAL 引数の LOGICAL FUNCTION (実数型の場合)。 2 つの COMPLEX 引数の LOGICAL FUNCTION (複素数型の場合)。</p> <p><i>selctg</i> は、呼び出しサブルーチン内で EXTERNAL として宣言する必要がある。</p> <p><i>sort</i>='S' の場合、<i>selctg</i> は、Schur 形式の左上に対してソートを実行する固有値を選択するために使用される。 <i>sort</i>='N' の場合、<i>selctg</i> は参照されない。</p> <p>実数型の場合： <i>selctg</i>(<i>alphan</i>(<i>j</i>), <i>alpha</i>(<i>j</i>), <i>beta</i>(<i>j</i>)) が真の場合、 固有値 (<i>alphan</i>(<i>j</i>) + <i>alpha</i>(<i>j</i>))/<i>beta</i>(<i>j</i>) が選択される。つまり、固有値の複素共役ペアの一方が選択されると、複素数の固有値が両方選択される。 悪条件の場合、指定された複素数の固有値は、順序付けの実行後、 <i>selctg</i>(<i>alphan</i>(<i>j</i>), <i>alpha</i>(<i>j</i>), <i>beta</i>(<i>j</i>)) = .TRUE. を満たさないことがある。その場合、<i>info</i> に <i>n</i>+2 を設定する。</p> <p>複素数型の場合： <i>selctg</i>(<i>alpha</i>(<i>j</i>), <i>beta</i>(<i>j</i>)) が真の場合、固有値 <i>alpha</i>(<i>j</i>) / <i>beta</i>(<i>j</i>) が選択される。 順序付けを行うと、複素数の固有値の値が変更される可能性がある (特に、固有値が悪い条件の場合)。そのため、順序付けを実行した後、指定された複素数の固有値は、<i>selctg</i>(<i>alpha</i>(<i>j</i>), <i>beta</i>(<i>j</i>)) = .TRUE. を満たさないことがある。その場合、<i>info</i> に <i>n</i>+2 が設定される (以下の <i>info</i> を参照)。</p>
<i>n</i>	INTEGER。行列 <i>A</i> 、 <i>B</i> 、 <i>vs1</i> 、および <i>vsr</i> の次数 ( <i>n</i> ≥ 0)。
<i>a</i> , <i>b</i> , <i>work</i>	<p>REAL (<i>sgges</i> の場合) DOUBLE PRECISION (<i>dgges</i> の場合) COMPLEX (<i>cgges</i> の場合) DOUBLE COMPLEX (<i>zgges</i> の場合)。</p> <p>配列： <i>a</i>(<i>lda</i>,*) は、<i>n</i> × <i>n</i> の行列 <i>A</i> (行列のペアの 1 番目) を格納する配列である。 <i>a</i> の第 2 次元は、max(1, <i>n</i>) 以上でなければならない。</p>

$b(ldb,*)$  は、 $n \times n$  の行列  $B$  ( 行列のペアの 2 番目 ) を格納する配列である。

$b$  の第 2 次元は、 $\max(1, n)$  以上でなければならない。

$work(lwork)$  は、ワークスペース配列である。

$lda$  INTEGER。配列  $a$  の第 1 次元。  
 $\max(1, n)$  以上でなければならない。

$ldb$  INTEGER。配列  $b$  の第 1 次元。  
 $\max(1, n)$  以上でなければならない。

$ldvsl, ldvsr$  INTEGER。それぞれ、出力行列  $vsl$  と  $vsr$  の第 1 次元。  
次の制約がある。  
 $ldvsl \geq 1$ 。  $jobvsl = 'V'$  の場合、 $ldvsl \geq \max(1, n)$ 。  
 $ldvsr \geq 1$ 。  $jobvsr = 'V'$  の場合、 $ldvsr \geq \max(1, n)$ 。

$lwork$  INTEGER。配列  $work$  の次元。  
  
 $lwork \geq \max(1, 8n+16)$  ( 実数型の場合 )。  
 $lwork \geq \max(1, 2n)$  ( 複素数型の場合 )。  
パフォーマンスを改善するには、一般に  $lwork$  に上記以上の値が必要である。

$lwork = -1$  の場合、ワークスペースのクエリとみなされ、ルーチンは  $work$  配列の最適サイズだけを計算し、その値を  $work$  配列の最初のエントリとして返す。xerbla は  $lwork$  に関するエラー・メッセージを生成しない。

$rwork$  REAL (cgges の場合 )  
DOUBLE PRECISION (zgges の場合 )  
ワークスペース配列、次元は  $\max(1, 8n)$  以上。  
この配列は、複素数型でのみ使用される。

$bwork$  LOGICAL。  
ワークスペース配列、次元は  $\max(1, n)$  以上。  
 $sort = 'N'$  の場合は参照されない。

## 出力パラメータ

$a$  終了時に、その汎用 Schur 形式  $S$  で上書きされる。

$b$  終了時に、その汎用 Schur 形式  $T$  で上書きされる。

$sdim$  INTEGER。  
 $sort = 'N'$  の場合、 $sdim = 0$  である。  
 $sort = 'S'$  の場合、 $sdim$  は、 $selctg$  が真の ( ソート後の ) 固有値の数

に等しくなる。

実数型の場合、どちらかの固有値について *selctg* が真の複素共役ペアは、2 としてカウントされるのに注意が必要である。

<i>alphar, alphai</i>	<p>REAL (sgges の場合 )            DOUBLE PRECISION (dgges の場合 )。            配列、次元はそれぞれ <math>\max(1, n)</math> 以上。実数型の汎用固有値を形成する値が格納される。  <i>beta</i> を参照。</p>
<i>alpha</i>	<p>COMPLEX (cgges の場合 )            DOUBLE COMPLEX (zgges の場合 )。            配列、次元は <math>\max(1, n)</math> 以上。複素数型の汎用固有値を形成する値が格納される。<i>beta</i> を参照。</p>
<i>beta</i>	<p>REAL (sgges の場合 )            DOUBLE PRECISION (dgges の場合 )            COMPLEX (cgges の場合 )            DOUBLE COMPLEX (zgges の場合 )。            配列、次元は <math>\max(1, n)</math> 以上。            実数型の場合:            終了時に、<math>(\text{alphar}(j) + \text{alphai}(j)*i)/\text{beta}(j)</math>、<math>j=1, \dots, n</math> は、汎用固有値になる。  <math>\text{alphar}(j) + \text{alphai}(j)*i</math> と <math>\text{beta}(j)</math>、<math>j=1, \dots, n</math> は、<math>(A, B)</math> の実数汎用 Schur 形式の <math>2 \times 2</math> の対角ブロックを複素数のユニタリ変換を使用して三角形式に更に縮退する場合に生じる複素数 Schur 形式 <math>(S, T)</math> の対角である。<math>\text{alphai}(j)</math> がゼロの場合、<math>j</math> 番目の固有値が実数であり、正の場合、<math>j</math> 番目と <math>(j+1)</math> 番目の固有値が複素共役ペアである (<math>\text{alphai}(j+1)</math> は負)。            複素数型の場合:            終了時に、<math>\text{alpha}(j)/\text{beta}(j)</math>、<math>j=1, \dots, n</math> は、汎用固有値になる。  <math>\text{alpha}(j)</math>、<math>j=1, \dots, n</math> と <math>\text{beta}(j)</math>、<math>j=1, \dots, n</math> は、cgges/zgges による複素数の Schur 形式 <math>(S, T)</math> 出力の対角である。<i>beta</i>(j) は、非負の実数になる。</p> <p>次の「アプリケーション・ノート」も参照のこと。</p>
<i>vs1, vsr</i>	<p>REAL (sgges の場合 )            DOUBLE PRECISION (dgges の場合 )            COMPLEX (cgges の場合 )            DOUBLE COMPLEX (zgges の場合 )。            配列:  <math>\text{vs1}(\text{ldvs1}, *)</math>、<i>vs1</i> の第 2 次元は、<math>\max(1, n)</math> 以上でなければならな</p>

い。  
`jobvsl='V'` の場合、左 Schur ベクトルが格納される。  
`jobvsl='N'` の場合、`vsl` は参照されない。

`vsr(ldvsr,*)`、`vsr` の第 2 次元は、 $\max(1, n)$  以上でなければならない。  
`jobvsr='V'` の場合、右 Schur ベクトルが格納される。  
`jobvsr='N'` の場合、`vsr` は参照されない。

`work(1)` 終了時に、`info=0` の場合、`work(1)` は `lwork` の必要最小サイズを返す。

`info` INTEGER。  
`info=0` の場合、実行は正常に終了したことを示す。  
`info=-i` の場合、`i` 番目のパラメータの値が不正であったことを示す。  
`info=i`、かつ  
 $i \leq n$  :

`QZ` の繰り返し失敗した。`(A, B)` は Schur 形式でないが、`alphar(j)`、`alpha(j)` (実数型の場合)、または `alpha(j)` (複素数型の場合)、および `beta(j)`、 $j=info+1, \dots, n$  は正しい。

$i > n$  の場合：一般に LAPACK の問題を示すエラー。

$i = n+1$  の場合：[?hgeqz](#) で `QZ` の繰り返し以外の操作に失敗した。

$i = n+2$  の場合：順序変更後、丸め操作によりいくつかの複素数の固有値の値が変更され、その結果、汎用 Schur 形式の先頭の固有値が `selctg=.TRUE.` を満たさなくなった。これは、スケーリングによっても引き起こされる。

$i = n+3$  の場合：[?tqsen](#) で順序変更失敗した。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `gges` のインターフェイスの詳細を以下に示す。

<code>a</code>	サイズ $(n, n)$ の行列 <code>A</code> を格納する。
<code>b</code>	サイズ $(n, n)$ の行列 <code>B</code> を格納する。
<code>alphar</code>	長さ $(n)$ のベクトルを格納する。実数型でのみ使用される。

<i>alphai</i>	長さ ( <i>n</i> ) のベクトルを格納する。実数型でのみ使用される。
<i>alpha</i>	長さ ( <i>n</i> ) のベクトルを格納する。複素数型でのみ使用される。
<i>beta</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>vs1</i>	サイズ ( <i>n,n</i> ) の行列 <i>VSL</i> を格納する。
<i>vsr</i>	サイズ ( <i>n,n</i> ) の行列 <i>VSR</i> を格納する。
<i>jobvs1</i>	引数 <i>vs1</i> の存在に基づいて以下のように復元される。 <i>jobvs1</i> = 'V' ( <i>vs1</i> が存在する場合)、 <i>jobvs1</i> = 'N' ( <i>vs1</i> が省略された場合)。
<i>jobvsr</i>	引数 <i>vsr</i> の存在に基づいて以下のように復元される。 <i>jobvsr</i> = 'V' ( <i>vsr</i> が存在する場合)、 <i>jobvsr</i> = 'N' ( <i>vsr</i> が省略された場合)。
<i>sort</i>	引数 <i>select</i> の存在に基づいて以下のように復元される。 <i>sort</i> = 'S' ( <i>select</i> が存在する場合)、 <i>sort</i> = 'N' ( <i>select</i> が省略された場合)。

### アプリケーション・ノート

配列 *work* に必要なワークスペースの大きさがわからない場合は、最初の実行で *lwork* を大きめの値に設定する。終了時に、*work*(1) の値を調べ、以降はその値を使用する。

一般に、商 *alphas*(*j*)/*beta*(*j*) と *alphai*(*j*)/*beta*(*j*) は、簡単にオーバーフローまたはアンダーフローし、*beta*(*j*) がゼロになる場合もある。したがって、単純に比を計算するのは避けなければならない。ただし、*alphas* と *alphai* は、大きさがノルム (*A*) より常に小さくなり、通常はノルム (*A*) と比較できる。また、*beta* は、ノルム (*B*) より常に小さくなり、通常はノルム (*B*) と比較できる。

## ?ggesx

汎用固有値、Schur 形式と ( オプションで )  
Schur ベクトルの左 / 右の行列を計算する。

---

### 構文

#### Fortran 77:

```
call sggesx (jobvsl, jobvsr, sort, selctg, sense, n, a, lda, b, ldb, sdim,  
            alphas, alphas, beta, vsl, ldvsl, vsr, ldvsr, rconde, rcondv, work, lwork,  
            iwork, liwork, bwork, info)  
call dggesx (jobvsl, jobvsr, sort, selctg, sense, n, a, lda, b, ldb, sdim,  
            alphas, alphas, beta, vsl, ldvsl, vsr, ldvsr, rconde, rcondv, work, lwork,  
            iwork, liwork, bwork, info)  
call cggesx (jobvsl, jobvsr, sort, selctg, sense, n, a, lda, b, ldb, sdim,  
            alpha, beta, vsl, ldvsl, vsr, ldvsr, rconde, rcondv, work, lwork, rwork,  
            iwork, liwork, bwork, info)  
call zggesx (jobvsl, jobvsr, sort, selctg, sense, n, a, lda, b, ldb, sdim,  
            alpha, beta, vsl, ldvsl, vsr, ldvsr, rconde, rcondv, work, lwork, rwork,  
            iwork, liwork, bwork, info)
```

#### Fortran 95:

```
call ggesx(a, b, alphas, alphas, beta [,vsl] [,vsr] [,select] [,sdim] [,rconde]  
          [,rcondv] [,info])  
call ggesx(a, b, alpha, beta [,vsl] [,vsr] [,select] [,sdim] [,rconde] [,rcondv]  
          [,info])
```

### 説明

このルーチンは、 $n \times n$  の実 / 複素非対称行列  $(A, B)$  のペアについて、汎用固有値、汎用実数 / 複素数 Schur 形式  $(S, T)$  と ( オプションで ) Schur ベクトルの左 / 右の行列 ( $vsl$  と  $vsr$ ) を計算する。これにより、汎用 Schur 因子分解

$$(A, B) = (vsl * S * vsr^H, vsl * T * vsr^H)$$

が提供される。

このルーチンは、固有値の選択した束が上準三角行列  $S$  と上三角行列  $T$  の主対角ブロックにくるように固有値を順序付けられる。また、指定された固有値の平均に対する条件数の逆数 ( $rconde$ ) を計算し、指定された固有値に対応する右と左の収縮部分空間に対する条件数の逆数 ( $rcondv$ ) を計算することもできる。 $vs1$  と  $vsr$  の先頭の列は、対応する左と右の固有空間 (収縮部分空間) に対する直交 / ユニタリ基を形成する。

行列  $(A, B)$  のペアの汎用固有値は、 $A - w*B$  が特異であるようなスカラ  $w$  または比  $\alpha / \beta = w$  である。 $\beta=0$  または両方がゼロである妥当な解釈が存在するため、通常は、ペア  $(\alpha, \beta)$  として表現される。

$T$  が非負の対角を持つ上三角で  $S$  が  $1 \times 1$  と  $2 \times 2$  のブロックを持つブロック上三角である場合、行列  $(S, T)$  のペアは、汎用実数 Schur 形式である。 $1 \times 1$  ブロックは実数の汎用固有値に対応するが、 $S$  の  $2 \times 2$  ブロックは、 $T$  の対応する成分を次の形式にすることで「標準化」される。

$$\begin{pmatrix} a & 0 \\ 0 & b \end{pmatrix}$$

また、 $S$  と  $T$  の対応する  $2 \times 2$  ブロックのペアは、汎用固有値の複素共役ペアを持つ。 $S$  と  $T$  が上三角であり、 $T$  の対角が非負の実数の場合、行列  $(S, T)$  のペアは、汎用複素数 Schur 形式になる。

### 入力パラメータ

<i>jobvs1</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobvs1</i> ='N' の場合、左 Schur ベクトルは計算されない。 <i>jobvs1</i> ='V' の場合、左 Schur ベクトルは計算される。
<i>jobvsr</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobvsr</i> ='N' の場合、右 Schur ベクトルは計算されない。 <i>jobvsr</i> ='V' の場合、右 Schur ベクトルは計算される。
<i>sort</i>	CHARACTER*1。'N' または 'S' でなければならない。 汎用 Schur 形式の対角上の固有値を順序付けるかどうかを指定する。  <i>sort</i> ='N' の場合、固有値は順序付けされない。 <i>sort</i> ='S' の場合、固有値は順序付けされる ( <i>select</i> を参照)。
<i>selectg</i>	3 つの REAL 引数の LOGICAL FUNCTION (実数型の場合)。 2 つの COMPLEX 引数の LOGICAL FUNCTION (複素数型の場合)。

*selctg* は、呼び出しサブルーチン内で **EXTERNAL** として宣言する必要がある。

*sort*='S' の場合、*selctg* は、Schur 形式の左上に対してソートを実行する固有値を選択するために使用される。

*sort*='N' の場合、*selctg* は参照されない。

**実数型の場合:**

*selctg*(*alphan*(*j*), *alphai*(*j*), *beta*(*j*)) が真の場合、固有値 (*alphan*(*j*) + *alphai*(*j*))/*beta*(*j*) が選択される。つまり、固有値の複素共役ペアの一方が選択されると、複素数の固有値が両方選択される。

悪条件の場合、指定された複素数の固有値は、順序付けの実行後、*selctg*(*alphan*(*j*), *alphai*(*j*), *beta*(*j*)) = .TRUE. を満たさないことがある。その場合、*info* に *n*+2 を設定する。

**複素数型の場合:**

*selctg*(*alpha*(*j*), *beta*(*j*)) が真の場合、固有値 *alpha*(*j*) / *beta*(*j*) が選択される。

順序付けを行うと、複素数の固有値の値が変更される可能性がある (特に、固有値が悪い条件の場合)。そのため、順序付けを実行した後、指定された複素数の固有値は、*selctg*(*alpha*(*j*), *beta*(*j*)) = .TRUE. を満たさないことがある。その場合、*info* に *n*+2 が設定される (以下の *info* を参照)。

*sense* CHARACTER\*1. 'N'、'E'、'V'、または 'B' でなければならない。どの条件数の逆数を計算するのかが決定する。

*sense*='N' の場合、計算は実行されない。

*sense*='E' の場合、指定された固有値の平均についてのみ計算が実行される。

*sense*='V' の場合、指定された収縮部分空間についてのみ計算が実行される。

*sense*='B' の場合、両方について計算が実行される。

*sense* が 'E'、'V'、または 'B' の場合、*sort* は 'S' に等しくなければならない。

*n* INTEGER。行列 *A*、*B*、*vs1*、および *vsr* の次数 (*n* ≥ 0)。

*a*, *b*, *work* REAL (sggesx の場合)  
DOUBLE PRECISION (dggex の場合)  
COMPLEX (cggesx の場合)  
DOUBLE COMPLEX (zggesx の場合)。  
配列:



	<p><math>a(lda,*)</math> は、<math>n \times n</math> の行列 <math>A</math> ( 行列のペアの 1 番目 ) を格納する配列である。</p> <p><math>a</math> の第 2 次元は、<math>\max(1, n)</math> 以上でなければならない。</p> <p><math>b ldb,*)</math> は、<math>n \times n</math> の行列 <math>B</math> ( 行列のペアの 2 番目 ) を格納する配列である。</p> <p><math>b</math> の第 2 次元は、<math>\max(1, n)</math> 以上でなければならない。</p> <p><math>work(lwork)</math> は、ワークスペース配列である。</p>
<i>lda</i>	INTEGER。配列 $a$ の第 1 次元。 $\max(1, n)$ 以上でなければならない。
<i>ldb</i>	INTEGER。配列 $b$ の第 1 次元。 $\max(1, n)$ 以上でなければならない。
<i>ldvsl, ldvsr</i>	INTEGER。それぞれ、出力行列 $vs1$ と $vsr$ の第 1 次元。次の制約がある。 $ldvsl \geq 1$ 。 $jobvsl = 'V'$ の場合、 $ldvsl \geq \max(1, n)$ 。 $ldvsr \geq 1$ 。 $jobvsr = 'V'$ の場合、 $ldvsr \geq \max(1, n)$ 。
<i>lwork</i>	INTEGER。配列 $work$ の次元。 実数型の場合： $lwork \geq \max(1, 8(n+1)+16)$ 。 $sense = 'E', 'V',$ または $'B'$ の場合、 $lwork \geq \max( 8(n+1)+16, 2*sdim*(n-sdim))$ 。 複素数型の場合： $lwork \geq \max(1, 2n)$ 。 $sense = 'E', 'V',$ または $'B'$ の場合、 $lwork \geq \max( 2n, 2*sdim*(n-sdim))$ 。  パフォーマンスを改善するには、一般に $lwork$ に上記以上の値が必要である。
<i>rwork</i>	REAL (cggesx の場合 ) DOUBLE PRECISION (zggesx の場合 ) ワークスペース配列、次元は $\max(1, 8n)$ 以上。 この配列は、複素数型でのみ使用される。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は ( $liwork$ )。 $sense = 'N'$ の場合は参照されない。
<i>liwork</i>	INTEGER。配列 $iwork$ の次元。

$liwork \geq n+6$  (実数型の場合)。  
 $liwork \geq n+2$  (複素数型の場合)。

*bwork* LOGICAL。  
 ワークスペース配列、次元は  $\max(1, n)$  以上。  
 $sort = 'N'$  の場合は参照されない。

## 出力パラメータ

*a* 終了時に、その汎用 Schur 形式  $S$  で上書きされる。

*b* 終了時に、その汎用 Schur 形式  $T$  で上書きされる。

*sdim* INTEGER。  
 $sort = 'N'$  の場合、 $sdim = 0$  である。  
 $sort = 'S'$  の場合、 $sdim$  は、 $selctg$  が真の (ソート後の) 固有値の数に等しくなる。  
 実数型の場合、どちらかの固有値について  $selctg$  が真の複素共役ペアは、2 としてカウントされるのに注意が必要である。

*alphan, alphai* REAL (sggesx の場合)  
 DOUBLE PRECISION (dggex の場合)。  
 配列、次元はそれぞれ  $\max(1, n)$  以上。実数型の汎用固有値を形成する値が格納される。  
*beta* を参照。

*alpha* COMPLEX (cggesx の場合)  
 DOUBLE COMPLEX (zggesx の場合)。  
 配列、次元は  $\max(1, n)$  以上。複素数型の汎用固有値を形成する値が格納される。*beta* を参照。

*beta* REAL (sggesx の場合)  
 DOUBLE PRECISION (dggex の場合)  
 COMPLEX (cggesx の場合)  
 DOUBLE COMPLEX (zggesx の場合)。  
 配列、次元は  $\max(1, n)$  以上。  
 実数型の場合:  
 終了時に、 $(alphan(j) + alphai(j)*i)/beta(j)$ 、 $j=1, \dots, n$  は、汎用固有値になる。  
 $alphan(j) + alphai(j)*i$  と  $beta(j)$ 、 $j=1, \dots, n$  は、 $(A, B)$  の実数汎用 Schur 形式の  $2 \times 2$  の対角ブロックを複素数のユニタリ変換を使用して三角形式に更に縮退する場合に生じる複素数 Schur 形式  $(S, T)$  の対角である。 $alphai(j)$  がゼロの場合、 $j$  番目の固有値が実数であり、正の場合、 $j$  番目と  $(j+1)$  番目の固有値が複素共役ペアである ( $alphai(j+1)$  は

負)。

複素数型の場合：

終了時に、 $\alpha(j)/\beta(j)$ 、 $j=1, \dots, n$  は、汎用固有値になる。

$\alpha(j)$ 、 $j=1, \dots, n$  と  $\beta(j)$ 、 $j=1, \dots, n$  は、cggesx/zggesx による複素数の Schur 形式 ( $S, T$ ) 出力の対角である。 $\beta(j)$  は、非負の実数になる。

次の「アプリケーション・ノート」も参照のこと。

*vs1, vsr*      REAL (sggesx の場合)  
                   DOUBLE PRECISION (dggesx の場合)  
                   COMPLEX (cggesx の場合)  
                   DOUBLE COMPLEX (zggesx の場合)。

配列：

$vs1(ldvs1, *)$ 、 $vs1$  の第 2 次元は、 $\max(1, n)$  以上でなければならない。

$jobvs1='V'$  の場合、左 Schur ベクトルが格納される。

$jobvs1='N'$  の場合、 $vs1$  は参照されない。

$vsr(ldvsr, *)$ 、 $vsr$  の第 2 次元は、 $\max(1, n)$  以上でなければならない。

$jobvsr='V'$  の場合、右 Schur ベクトルが格納される。

$jobvsr='N'$  の場合、 $vsr$  は参照されない。

*rconde, rcondv*    REAL (単精度の場合)  
                   DOUBLE PRECISION (倍精度の場合)。  
                   配列は、次元はそれぞれ (2)

$sense='E'$  または  $'B'$  の場合、 $rconde(1)$  と  $rconde(2)$  には、指定された固有値の平均に対する条件数の逆数が格納される。

$sense='N'$  または  $'V'$  の場合は参照されない。

$sense='V'$  または  $'B'$  の場合、 $rcondv(1)$  と  $rcondv(2)$  には、指定された収縮部分空間に対する条件数の逆数が格納される。

$sense='N'$  または  $'E'$  の場合は参照されない。

*work(1)*          終了時に、 $info=0$  の場合、 $work(1)$  は  $lwork$  の必要最小サイズを返す。

*info*             INTEGER。  
                    $info=0$  の場合、実行は正常に終了したことを示す。  
                    $info=-i$  の場合、 $i$  番目のパラメータの値が不正であったことを示

す。

$info = i$ 、かつ  
 $i \leq n$ :

$QZ$  の繰り返しが失敗した。( $A, B$ ) は Schur 形式でないが、 $alphar(j)$ 、 $alphai(j)$  (実数型の場合)、または  $alpha(j)$  (複素数型の場合)、および  $beta(j)$ 、 $j=info+1, \dots, n$  は正しい。

$i > n$  の場合: 一般に LAPACK の問題を示すエラー。

$i = n+1$  の場合: [zhgeqz](#) で  $QZ$  の繰り返し以外の操作に失敗した。

$i = n+2$  の場合: 順序変更後、丸め操作によりいくつかの複素数の固有値の値が変更され、その結果、汎用 Schur 形式の先頭の固有値が  $selectg = .TRUE.$  を満たさなくなった。これは、スケーリングによっても引き起こされる。

$i = n+3$  の場合: [ztgsen](#) で順序変更に失敗した。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `ggesx` のインターフェイスの詳細を以下に示す。

<code>a</code>	サイズ $(n, n)$ の行列 $A$ を格納する。
<code>b</code>	サイズ $(n, n)$ の行列 $B$ を格納する。
<code>alphar</code>	長さ $(n)$ のベクトルを格納する。実数型でのみ使用される。
<code>alphai</code>	長さ $(n)$ のベクトルを格納する。実数型でのみ使用される。
<code>alpha</code>	長さ $(n)$ のベクトルを格納する。複素数型でのみ使用される。
<code>beta</code>	長さ $(n)$ のベクトルを格納する。
<code>vs1</code>	サイズ $(n, n)$ の行列 $VSL$ を格納する。
<code>vsr</code>	サイズ $(n, n)$ の行列 $VSR$ を格納する。
<code>rconde</code>	長さ $(2)$ のベクトルを格納する。
<code>rcondv</code>	長さ $(2)$ のベクトルを格納する。
<code>jobvs1</code>	引数 <code>vs1</code> の存在に基づいて以下のように復元される。 <code>jobvs1 = 'V'</code> ( <code>vs1</code> が存在する場合)、 <code>jobvs1 = 'N'</code> ( <code>vs1</code> が省略された場合)。

<i>jobvsr</i>	引数 <i>vsr</i> の存在に基づいて以下のように復元される。 <i>jobvsr</i> = 'V' ( <i>vsr</i> が存在する場合)、 <i>jobvsr</i> = 'N' ( <i>vsr</i> が省略された場合)。
<i>sort</i>	引数 <i>select</i> の存在に基づいて以下のように復元される。 <i>sort</i> = 'S' ( <i>select</i> が存在する場合)、 <i>sort</i> = 'N' ( <i>select</i> が省略された場合)。
<i>sense</i>	引数 <i>rconde</i> と <i>rcondv</i> の存在に基づいて以下のように復元される。 <i>sense</i> = 'B' ( <i>rconde</i> と <i>rcondv</i> の両方が存在する場合)、 <i>sense</i> = 'E' ( <i>rconde</i> が存在し、 <i>rcondv</i> が省略された場合)、 <i>sense</i> = 'V' ( <i>rconde</i> が省略され、 <i>rcondv</i> が存在する場合)、 <i>sense</i> = 'N' ( <i>rconde</i> と <i>rcondv</i> の両方が省略された場合)。

*rconde* または *rcondv* が存在し、*select* が省略された場合、エラー条件がセットされる。

### アプリケーション・ノート

配列 *work* に必要なワークスペースの大きさがわからない場合は、最初の実行で *lwork* を大きめの値に設定する。終了時に、*work(1)* の値を調べ、以降はその値を使用する。

一般に、商  $\text{alphan}(j)/\text{betan}(j)$  と  $\text{alphai}(j)/\text{betai}(j)$  は、簡単にオーバーフローまたはアンダーフローし、 $\text{betan}(j)$  がゼロになる場合もある。したがって、単純に比を計算するのは避けなければならない。ただし、 $\text{alphan}$  と  $\text{alphai}$  は、大きさがノルム (A) より常に小さくなり、通常はノルム (A) と比較できる。また、 $\text{betan}$  は、ノルム (B) より常に小さくなり、通常はノルム (B) と比較できる。

---

## ?ggeev

非対称行列のペアについて、汎用固有値、左/右の汎用固有ベクトルを計算する。

---

### 構文

#### Fortran 77:

```
call sggev(jobvl, jobvr, n, a, lda, b, ldb, alphan, alphai, betan, vl, ldvl, vr,
           ldvr, work, lwork, info)
call dggev(jobvl, jobvr, n, a, lda, b, ldb, alphan, alphai, betan, vl, ldvl, vr,
           ldvr, work, lwork, info)
```

```
call cggev(jobvl, jobvr, n, a, lda, b, ldb, alpha, beta, vl, ldvl, vr, ldvr,  
work, lwork, rwork, info)
```

```
call zggev(jobvl, jobvr, n, a, lda, b, ldb, alpha, beta, vl, ldvl, vr, ldvr,  
work, lwork, rwork, info)
```

### Fortran 95:

```
call ggev(a, b, alphas, alphai, beta [,vl] [,vr] [,info])
```

```
call ggev(a, b, alpha, beta [,vl] [,vr] [,info])
```

### 説明

このルーチンは、 $n \times n$  の実 / 複素非対称行列  $(A, B)$  のペアについて、汎用固有値と (オプションで) 左 / 右の汎用固有ベクトルを計算する。

行列  $(A, B)$  のペアの汎用固有値は、 $A - \lambda * B$  が特異であるようなスカラー  $\lambda$  または比  $\alpha / \beta = \lambda$  である。  $\beta=0$  および両方がゼロである妥当な解釈が存在するため、通常は、ペア  $(\alpha, \beta)$  として表現される。

$(A, B)$  の汎用固有値  $\lambda(j)$  に対応する右汎用固有ベクトル  $v(j)$  は、以下を満たす。

$$A * v(j) = \lambda(j) * B * v(j)$$

$(A, B)$  の汎用固有値  $\lambda(j)$  に対応する左汎用固有ベクトル  $u(j)$  は、以下を満たす。

$$u(j)^H * A = \lambda(j) * u(j)^H * B$$

$u(j)^H$  は、 $u(j)$  の共役転置を示す。

### 入力パラメータ

<i>jobvl</i>	CHARACTER*1。 'N' または 'V' でなければならない。 <i>jobvl</i> ='N' の場合、左汎用固有ベクトルは計算されない。 <i>jobvl</i> ='V' の場合、左汎用固有ベクトルは計算される。
<i>jobvr</i>	CHARACTER*1。 'N' または 'V' でなければならない。 <i>jobvr</i> ='N' の場合、右汎用固有ベクトルは計算されない。 <i>jobvr</i> ='V' の場合、右汎用固有ベクトルは計算される。
<i>n</i>	INTEGER。 行列 $A$ 、 $B$ 、 $vl$ 、 $vr$ の次数 ( $n \geq 0$ )。
<i>a</i> , <i>b</i> , <i>work</i>	REAL (sggev の場合 ) DOUBLE PRECISION (dggev の場合 ) COMPLEX (cggev の場合 ) DOUBLE COMPLEX (zggev の場合 )。 配列 :

$a(lda,*)$  は、 $n \times n$  の行列  $A$  (行列のペアの 1 番目) を格納する配列である。

$a$  の第 2 次元は、 $\max(1, n)$  以上でなければならない。

$b(ldb,*)$  は、 $n \times n$  の行列  $B$  (行列のペアの 2 番目) を格納する配列である。

$b$  の第 2 次元は、 $\max(1, n)$  以上でなければならない。

$work(lwork)$  は、ワークスペース配列である。

$lda$	INTEGER。配列 $a$ の第 1 次元。 $\max(1, n)$ 以上でなければならない。
$ldb$	INTEGER。配列 $b$ の第 1 次元。 $\max(1, n)$ 以上でなければならない。
$ldvl, ldvr$	INTEGER。それぞれ、出力行列 $v1$ と $vr$ の第 1 次元。次の制約がある。 $ldvl \geq 1$ 。 $jobvl = 'V'$ の場合、 $ldvl \geq \max(1, n)$ 。 $ldvr \geq 1$ 。 $jobvr = 'V'$ の場合、 $ldvr \geq \max(1, n)$ 。
$lwork$	INTEGER。配列 $work$ の次元。  $lwork \geq \max(1, 8n+16)$ (実数型の場合)。 $lwork \geq \max(1, 2n)$ (複素数型の場合)。 パフォーマンスを改善するには、一般に $lwork$ に上記以上の値が必要である。  $lwork = -1$ の場合、ワークスペースのクエリとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエントリとして返す。 $xerbla$ は $lwork$ に関するエラー・メッセージを生成しない。
$rwork$	REAL (cggev の場合) DOUBLE PRECISION (zggev の場合) ワークスペース配列、次元は $\max(1, 8n)$ 以上。 この配列は、複素数型でのみ使用される。

## 出力パラメータ

$a, b$	終了時に上書きされる。
$alphar, alphas$	REAL (sggev の場合) DOUBLE PRECISION (dggev の場合)。 配列、次元はそれぞれ $\max(1, n)$ 以上。実数型の汎用固有値を形成する値が格納される。 $beta$ を参照。

<i>alpha</i>	<p>COMPLEX (cggev の場合 )  DOUBLE COMPLEX (zggev の場合 )。  配列、次元は <math>\max(1, n)</math> 以上。複素数型の汎用固有値を形成する値が格納される。<i>beta</i> を参照。</p>
<i>beta</i>	<p>REAL (sggev の場合 )  DOUBLE PRECISION (dggev の場合 )  COMPLEX (cggev の場合 )  DOUBLE COMPLEX (zggev の場合 )。  配列、次元は <math>\max(1, n)</math> 以上。  実数型の場合:  終了時に、<math>(\alpha_{\text{r}}(j) + \alpha_{\text{i}}(j)*i)/\beta(j)</math>、<math>j=1, \dots, n</math> は、汎用固有値になる。  <math>\alpha_{\text{i}}(j)</math> がゼロの場合、<math>j</math> 番目の固有値が実数であり、正の場合、<math>j</math> 番目と <math>(j+1)</math> 番目の固有値が複素共役ペアである (<math>\alpha_{\text{i}}(j+1)</math> は負)。  複素数型の場合:  終了時に、<math>\alpha(j)/\beta(j)</math>、<math>j=1, \dots, n</math> は、汎用固有値になる。  次の「アプリケーション・ノート」も参照のこと。</p>
<i>v1, vr</i>	<p>REAL (sggev の場合 )  DOUBLE PRECISION (dggev の場合 )  COMPLEX (cggev の場合 )  DOUBLE COMPLEX (zggev の場合 )。  配列:  <math>v1(ldv1, *)</math>。<math>v1</math> の第 2 次元は、<math>\max(1, n)</math> 以上でなければならない。  <math>jobv1='V'</math> の場合、左汎用固有ベクトル <math>u(j)</math> は、それらの固有値と同じ順序で、<math>v1</math> の列に次々と格納される。各固有ベクトルは、最大のコンポーネントが <math>\text{abs}(\text{Re}) + \text{abs}(\text{Im}) = 1</math> となるようにスケーリングされる。<math>jobv1='N'</math> の場合、<math>v1</math> は参照されない。  実数型の場合:  <math>j</math> 番目の固有値が実数の場合、<math>u(j) = v1(:, j)</math> (<math>v1</math> の <math>j</math> 番目の列) である。  <math>j</math> 番目と <math>(j+1)</math> 番目の固有値が複素共役ペアとなる場合、  <math>u(j) = v1(:, j) + i*v1(:, j+1)</math>、<math>u(j+1) = v1(:, j) - i*v1(:, j+1)</math> である。  ここで、<math>i = \sqrt{-1}</math> である。  複素数型の場合:  <math>u(j) = v1(:, j)</math> (<math>v1</math> の <math>j</math> 番目の列) である。  <math>vr(ldvr, *)</math>。<math>vr</math> の第 2 次元は、<math>\max(1, n)</math> 以上でなければならない。</p>



$jobv = 'V'$  の場合、右汎用固有ベクトル  $v(j)$  は、それらの固有値と同じ順序で、 $vr$  の列に次々と格納される。各固有ベクトルは、最大のコンポーネントが  $\text{abs}(\text{Re}) + \text{abs}(\text{Im}) = 1$  となるようにスケーリングされる。 $jobv = 'N'$  の場合、 $vr$  は参照されない。

実数型の場合：

$j$  番目の固有値が実数の場合、 $v(j) = vr(:, j)$  ( $vr$  の  $j$  番目の列) である。 $j$  番目と  $(j+1)$  番目の固有値が複素共役ペアとなる場合は、 $v(j) = vr(:, j) + i * vr(:, j+1)$ 、 $v(j+1) = vr(:, j) - i * vr(:, j+1)$  である。

複素数型の場合：

$v(j) = vr(:, j)$  ( $vr$  の  $j$  番目の列) である。

$work(1)$  終了時に、 $info = 0$  の場合、 $work(1)$  は  $lwork$  の必要最小サイズを返す。

$info$  INTEGER。  
 $info = 0$  の場合、実行は正常に終了したことを示す。  
 $info = -i$  の場合、 $i$  番目のパラメータの値が不正であったことを示す。  
 $info = i$ 、かつ  
 $i \leq n$ ：

$QZ$  の繰り返しが失敗した。固有ベクトルは計算されなかったが、 $\alpha_r(j)$ 、 $\alpha_{hi}(j)$  (実数型の場合)、または  $\alpha(j)$  (複素数型の場合)、および  $\beta(j)$ 、 $j = info+1, \dots, n$  は正しい。

$i > n$  の場合：一般に LAPACK の問題を示すエラー。

$i = n+1$  の場合：[?hgeqz](#) で  $QZ$  の繰り返し以外の操作に失敗した。

$i = n+2$  の場合：[?tgevc](#) からエラーが返された。

## Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `ggeev` のインターフェイスの詳細を以下に示す。

$a$	サイズ $(n, n)$ の行列 $A$ を格納する。
$b$	サイズ $(n, n)$ の行列 $B$ を格納する。
$\alpha_r$	長さ $(n)$ のベクトルを格納する。実数型でのみ使用される。
$\alpha_{hi}$	長さ $(n)$ のベクトルを格納する。実数型でのみ使用される。

<i>alpha</i>	長さ ( <i>n</i> ) のベクトルを格納する。複素数型でのみ使用される。
<i>beta</i>	長さ ( <i>n</i> ) のベクトルを格納する。
<i>vl</i>	サイズ ( <i>n,n</i> ) の行列 <i>VL</i> を格納する。
<i>vr</i>	サイズ ( <i>n,n</i> ) の行列 <i>VR</i> を格納する。
<i>jobvl</i>	引数 <i>vl</i> の存在に基づいて以下のように復元される。 <i>jobvl</i> = 'V' ( <i>vl</i> が存在する場合)、 <i>jobvl</i> = 'N' ( <i>vl</i> が省略された場合)。
<i>jobvr</i>	引数 <i>vr</i> の存在に基づいて以下のように復元される。 <i>jobvr</i> = 'V' ( <i>vr</i> が存在する場合)、 <i>jobvr</i> = 'N' ( <i>vr</i> が省略された場合)。

### アプリケーション・ノート

配列 *work* に必要なワークスペースの大きさがわからない場合は、最初の実行で *lwork* を大きめの値に設定する。終了時に、*work(1)* の値を調べ、以降はその値を使用する。

一般に、商 *alphar(j)/beta(j)* と *alphai(j)/beta(j)* は、簡単にオーバーフローまたはアンダーフローし、*beta(j)* がゼロになる場合もある。したがって、単純に比を計算するのは避けなければならない。ただし、*alphar* と *alphai* (実数型の場合) または *alpha* (複素数型の場合) は、大きさがノルム (*A*) より常に小さくなり、通常はノルム (*A*) と比較できる。また、*beta* は、ノルム (*B*) より常に小さくなり、通常はノルム (*B*) と比較できる。

---

## ?ggev

汎用固有値と (オプションで) 左/右の汎用固有ベクトルを計算する。

---

### 構文

#### Fortran 77:

```
call sggev(balanc, jobvl, jobvr, sense, n, a, lda, b, ldb, alphar, alphai,  
          beta, vl, ldvl, vr, ldvr, ilo, ihi, lscale, rscale, abnrm, bbnrm, rconde,  
          rcondv, work, lwork, iwork, bwork, info)
```

```
call dggevx(balanc, jobvl, jobvr, sense, n, a, lda, b, ldb, alphas, alphas,
  beta, vl, ldvl, vr, ldvr, ilo, ihi, lscale, rscale, abnrm, bbnrm, rconde,
  rcondv, work, lwork, iwork, bwork, info)

call cggvx(balanc, jobvl, jobvr, sense, n, a, lda, b, ldb, alpha, beta, vl,
  ldvl, vr, ldvr, ilo, ihi, lscale, rscale, abnrm, bbnrm, rconde, rcondv,
  work, lwork, rwork, iwork, bwork, info)

call zggvx(balanc, jobvl, jobvr, sense, n, a, lda, b, ldb, alpha, beta, vl,
  ldvl, vr, ldvr, ilo, ihi, lscale, rscale, abnrm, bbnrm, rconde, rcondv,
  work, lwork, rwork, iwork, bwork, info)
```

**Fortran 95:**

```
call ggevx(a, b, alphas, alphas, beta [,vl] [,vr] [,balanc] [,ilo] [,ihi]
  [,lscale] [,rscale] [,abnrm] [,bbnrm] [,rconde] [,rcondv] [,info])
call ggevx(a, b, alpha, beta [,vl] [,vr] [,balanc] [,ilo] [,ihi] [,lscale]
  [,rscale] [,abnrm] [,bbnrm] [,rconde] [,rcondv] [,info])
```

**説明**

このルーチンは、 $n \times n$  の実 / 複素非対称行列  $(A, B)$  のペアについて、汎用固有値と (オプションで) 左 / 右の汎用固有ベクトルを計算する。

また、このルーチンは、平衡化のための変換を計算し、固有値と固有ベクトル ( $ilo$ 、 $ihi$ 、 $lscale$ 、 $rscale$ 、 $abnrm$ 、 $bbnrm$ )、固有値に対する条件数の逆数 ( $rconde$ )、右固有ベクトルに対する条件数の逆数 ( $rcondv$ ) の条件付けを改善できる。

行列  $(A, B)$  のペアの汎用固有値は、 $A - \lambda B$  が特異であるようなスカラー  $\lambda$  または比  $\alpha / \beta = \lambda$  である。 $\beta=0$  および両方がゼロである妥当な解釈が存在するため、通常は、ペア  $(\alpha, \beta)$  として表現される。

$(A, B)$  の汎用固有値  $\lambda(j)$  に対応する右汎用固有ベクトル  $v(j)$  は、以下を満たす。

$$A * v(j) = \lambda(j) * B * v(j)$$

$(A, B)$  の汎用固有値  $\lambda(j)$  に対応する左汎用固有ベクトル  $u(j)$  は、以下を満たす。

$$u(j)^H * A = \lambda(j) * u(j)^H * B$$

$u(j)^H$  は、 $u(j)$  の共役転置を示す。

**入力パラメータ**

**balanc** CHARACTER\*1。'N'、'P'、'S'、または 'B' でなければならない。  
実行する平衡化オプションを指定する。

*balanc* = 'N' の場合、対角的なスケーリングまたは置換は実行されない。

*balanc* = 'P' の場合、置換のみが実行される。

*balanc* = 'S' の場合、スケーリングのみが実行される。

*balanc* = 'B' の場合、置換とスケーリングがともに実行される。

計算された条件数の逆数は、平衡化または置換を実行後の行列に使用される。置換を実行しても条件数は変更されないが ( 正確な演算では )、平衡化を実行すると条件数が変更される。

<i>jobvl</i>	CHARACTER*1。 'N' または 'V' でなければならない。 <i>jobvl</i> = 'N' の場合、左汎用固有ベクトルは計算されない。 <i>jobvl</i> = 'V' の場合、左汎用固有ベクトルは計算される。
<i>jobvr</i>	CHARACTER*1。 'N' または 'V' でなければならない。 <i>jobvr</i> = 'N' の場合、右汎用固有ベクトルは計算されない。 <i>jobvr</i> = 'V' の場合、右汎用固有ベクトルは計算される。
<i>sense</i>	CHARACTER*1。 'N'、'E'、'V'、または 'B' でなければならない。 どの条件数の逆数を計算するのかを決定する。  <i>sense</i> = 'N' の場合、計算は実行されない。 <i>sense</i> = 'E' の場合、固有値についてのみ計算が実行される。 <i>sense</i> = 'V' の場合、固有ベクトルについてのみ計算が実行される。 <i>sense</i> = 'B' の場合、固有値と固有ベクトルについて計算が実行される。
<i>n</i>	INTEGER。 行列 <i>A</i> 、 <i>B</i> 、 <i>vl</i> 、 <i>vr</i> の次数 ( $n \geq 0$ )。
<i>a</i> , <i>b</i> , <i>work</i>	REAL (sggevx の場合 ) DOUBLE PRECISION (dggevx の場合 ) COMPLEX (cggevx の場合 ) DOUBLE COMPLEX (zggevx の場合 )。 配列 : <i>a</i> ( <i>lda</i> ,*) は、 $n \times n$ の行列 <i>A</i> ( 行列のペアの 1 番目 ) を格納する配列である。 <i>a</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。  <i>b</i> ( <i>ldb</i> ,*) は、 $n \times n$ の行列 <i>B</i> ( 行列のペアの 2 番目 ) を格納する配列である。 <i>b</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。  <i>work</i> ( <i>lwork</i> ) は、ワークスペース配列である。
<i>lda</i>	INTEGER。 配列 <i>a</i> の第 1 次元。 $\max(1, n)$ 以上でなければならない。

<i>ldb</i>	INTEGER。配列 <i>b</i> の第 1 次元。 $\max(1, n)$ 以上でなければならない。
<i>ldvl</i> , <i>ldvr</i>	INTEGER。それぞれ、出力行列 <i>vl</i> と <i>vr</i> の第 1 次元。次の制約がある。 $ldvl \geq 1$ 。 <i>jobvl</i> = 'V' の場合、 $ldvl \geq \max(1, n)$ 。 $ldvr \geq 1$ 。 <i>jobvr</i> = 'V' の場合、 $ldvr \geq \max(1, n)$ 。
<i>lwork</i>	INTEGER。配列 <i>work</i> の次元。 実数型の場合： $lwork \geq \max(1, 6n)$ 。 <i>sense</i> = 'E' の場合、 $lwork \geq 12n$ 。 <i>sense</i> = 'V' または 'B' の場合、 $lwork \geq 2n^2 + 12n + 16$ 。 複素数型の場合： $lwork \geq \max(1, 2n)$ 。 <i>sense</i> = 'N' または 'E' の場合、 $lwork \geq 2n$ 。 <i>sense</i> = 'V' または 'B' の場合、 $lwork \geq 2n^2 + 2n$ 。  <i>lwork</i> = -1 の場合、ワークスペースのクエリとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリとして返す。 <i>xerbla</i> は <i>lwork</i> に関するエラー・メッセージを生成しない。
<i>rwork</i>	REAL ( <i>cggevx</i> の場合) DOUBLE PRECISION ( <i>zggevx</i> の場合) ワークスペース配列、次元は $\max(1, 6n)$ 以上。 この配列は、複素数型でのみ使用される。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $(n+6)$ 以上 (実数型の場合) または $(n+2)$ 以上 (複素数型の場合)。 <i>sense</i> = 'E' の場合は参照されない。
<i>bwork</i>	LOGICAL。 ワークスペース配列、次元は $\max(1, n)$ 以上。 <i>sense</i> = 'N' の場合は参照されない。

## 出力パラメータ

<i>a</i> , <i>b</i>	終了時に上書きされる。  <i>jobvl</i> = 'V' または <i>jobvr</i> = 'V' (またはその両方) の場合、 <i>a</i> には、平衡化した入力 <i>A</i> と <i>B</i> の実数 Schur 形式の 1 番目の部分が格納され、 <i>b</i> には、その 2 番目の部分が格納される。
---------------------	---

*alphar, alphai* REAL (sggevx の場合 )  
DOUBLE PRECISION (dggevx の場合 )。  
配列、次元はそれぞれ  $\max(1, n)$  以上。実数型の汎用固有値を形成する値が格納される。  
*beta* を参照。

*alpha* COMPLEX (cggevx の場合 )  
DOUBLE COMPLEX (zggevx の場合 )。  
配列、次元は  $\max(1, n)$  以上。複素数型の汎用固有値を形成する値が格納される。*beta* を参照。

*beta* REAL (sggevx の場合 )  
DOUBLE PRECISION (dggevx の場合 )  
COMPLEX (cggevx の場合 )  
DOUBLE COMPLEX (zggevx の場合 )。  
配列、次元は  $\max(1, n)$  以上。  
実数型の場合:  
終了時に、 $(\text{alphar}(j) + \text{alphai}(j)*i)/\text{beta}(j)$ 、 $j=1, \dots, n$  は、汎用固有値になる。  
 $\text{alphai}(j)$  がゼロの場合、 $j$  番目の固有値が実数であり、正の場合、 $j$  番目と  $(j+1)$  番目の固有値が複素共役ペアである ( $\text{alphai}(j+1)$  は負 )。  
複素数型の場合:  
終了時に、 $\text{alpha}(j)/\text{beta}(j)$ 、 $j=1, \dots, n$  は、汎用固有値になる。  
次の「アプリケーション・ノート」も参照のこと。

*v1, vr* REAL (sggevx の場合 )  
DOUBLE PRECISION (dggevx の場合 )  
COMPLEX (cggevx の場合 )  
DOUBLE COMPLEX (zggevx の場合 )。  
配列:  
 $v1(ldv1, *)$ 。 $v1$  の第 2 次元は、 $\max(1, n)$  以上でなければならない。  
 $jobv1='V'$  の場合、左汎用固有ベクトル  $u(j)$  は、それらの固有値と同じ順序で、 $v1$  の列に次々と格納される。各固有ベクトルは、最大のコンポーネントが  $\text{abs}(\text{Re}) + \text{abs}(\text{Im}) = 1$  となるようにスケーリングされる。 $jobv1='N'$  の場合、 $v1$  は参照されない。  
実数型の場合:  
 $j$  番目の固有値が実数の場合、 $u(j) = v1(:, j)$  ( $v1$  の  $j$  番目の列) である。  
 $j$  番目と  $(j+1)$  番目の固有値が複素共役ペアとなる場合、  
 $u(j) = v1(:, j) + i*v1(:, j+1)$ 、 $u(j+1) = v1(:, j) - i*v1(:, j+1)$  である。  
ここで、 $i = \sqrt{-1}$  である。

複素数型の場合：

$u(j) = vl(:, j)$  ( $vl$  の  $j$  番目の列) である。

$vr(ldvr, *)$ 。 $vr$  の第 2 次元は、 $\max(1, n)$  以上でなければならない。

$jobvr = 'V'$  の場合、右汎用固有ベクトル  $v(j)$  は、それらの固有値と同じ順序で、 $vr$  の列に次々と格納される。各固有ベクトルは、最大のコンポーネントが  $\text{abs}(\text{Re}) + \text{abs}(\text{Im}) = 1$  となるようにスケーリングされる。 $jobvr = 'N'$  の場合、 $vr$  は参照されない。

実数型の場合：

$j$  番目の固有値が実数の場合、 $v(j) = vr(:, j)$  ( $vr$  の  $j$  番目の列) である。

$j$  番目と  $(j+1)$  番目の固有値が複素共役ペアとなる場合は、

$v(j) = vr(:, j) + i * vr(:, j+1)$ 、 $v(j+1) = vr(:, j) - i * vr(:, j+1)$  である。

複素数型の場合：

$v(j) = vr(:, j)$  ( $vr$  の  $j$  番目の列) である。

$ilo, ihi$

INTEGER。

$ilo$  と  $ihi$  は、終了時に、 $A(i, j) = 0$  および  $B(i, j) = 0$  ( $i > j$  かつ  $j = 1, \dots, ilo-1$  または  $i = ihi+1, \dots, n$  の場合) であるような整数値である。

$balanc = 'N'$  または  $'S'$  の場合、 $ilo = 1$  と  $ihi = n$  である。

$lscale, rscale$  REAL (単精度の場合)

DOUBLE PRECISION (倍精度の場合)。

配列、次元はそれぞれ  $\max(1, n)$  以上。

$lscale$  には、 $A$  と  $B$  の左側に適用される置換およびスケーリング係数の各成分が格納される。

$PL(j)$  は行  $j$  と交換される行のインデックスであり、 $DL(j)$  は行  $j$  に適用されるスケーリング係数である。したがって、

$lscale(j) = PL(j)$ 、 $j = 1, \dots, ilo-1$

$= DL(j)$ 、 $j = ilo, \dots, ihi$

$= PL(j)$ 、 $j = ihi+1, \dots, n$

交換を実行する順序は、 $n \sim ihi+1$ 、次に  $1 \sim ilo-1$  である。

$rscale$  には、 $A$  と  $B$  の右側に適用される置換およびスケーリング係数の各成分が格納される。

$PR(j)$  は列  $j$  と交換される列のインデックスであり、 $DR(j)$  は列  $j$  に適用されるスケーリング係数である。したがって、

$rscale(j) = PR(j)$ 、 $j = 1, \dots, ilo-1$

$= DR(j)$ 、 $j = ilo, \dots, ihi$

$= PR(j), j = ihi+1, \dots, n$

交換を実行する順序は、 $n \sim ihi+1$ 、次に  $1 \sim ilo-1$  である。

*abnrm, bbnrm* REAL (単精度の場合)  
DOUBLE PRECISION (倍精度の場合)。

それぞれ、平衡化した行列 *A* と *B* の 1- ノルムである。

*rconde, rcondv* REAL (単精度の場合)  
DOUBLE PRECISION (倍精度の場合)。  
配列、次元はそれぞれ  $\max(1, n)$  以上。

*sense* = 'E' または 'B' の場合、*rconde* には、指定された固有値の条件数の逆数を格納する。これらは、配列の連続する成分に格納される。固有値の複素共役ペアの場合、*rconde* の 2 つの連続する成分には同じ値が設定される。したがって、*rconde(j)*、*rcondv(j)*、*vl* と *vr* の *j* 番目の列はすべて、同じ固有ペアに対応している (ただし、選択しなかった固有ペアが存在する場合は、一般に、*j* 番目の固有ペアにはならない)。

*sense* = 'V' の場合、*rconde* は参照されない。

*sense* = 'V' または 'B' の場合、*rcondv* には、指定された固有ベクトルについての見積もりの条件数の逆数が格納される。これらは、配列の連続する成分に格納される。複素数の固有ベクトルの場合、*rcondv* の 2 つの連続する成分には同じ値が設定される。*rcondv(j)* を計算するために固有値の順序を変更できない場合は、*rcondv(j)* にゼロが設定される。これは、真の値が非常に小さい場合にだけ発生する可能性がある。

*sense* = 'E' の場合、*rcondv* は参照されない。

*work(1)* 終了時に、*info* = 0 の場合、*work(1)* は *lwork* の必要最小サイズを返す。

*info* INTEGER。  
*info* = 0 の場合、実行は正常に終了したことを示す。  
*info* = -*i* の場合、*i* 番目のパラメータの値が不正であったことを示す。  
*info* = *i*、かつ  
 $i \leq n$  :  
*QZ* の繰り返しが失敗した。固有ベクトルは計算されなかったが、*alphar(j)*、*alpha(i)(j)* (実数型の場合)、または *alpha(j)* (複素数型の場合)、および *beta(j)*、 $j = info+1, \dots, n$  は正しい。  
 $i > n$  の場合：一般に LAPACK の問題を示すエラー。



$i = n+1$  の場合：[?hgeqz](#) で  $QZ$  の繰り返し以外の操作に失敗した。

$i = n+2$  の場合：[?tgevc](#) からエラーが返された。

### Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `ggevx` のインターフェイスの詳細を以下に示す。

<code>a</code>	サイズ $(n, n)$ の行列 $A$ を格納する。
<code>b</code>	サイズ $(n, n)$ の行列 $B$ を格納する。
<code>alphar</code>	長さ $(n)$ のベクトルを格納する。実数型でのみ使用される。
<code>alphai</code>	長さ $(n)$ のベクトルを格納する。実数型でのみ使用される。
<code>alpha</code>	長さ $(n)$ のベクトルを格納する。複素数型でのみ使用される。
<code>beta</code>	長さ $(n)$ のベクトルを格納する。
<code>v1</code>	サイズ $(n, n)$ の行列 $VL$ を格納する。
<code>vr</code>	サイズ $(n, n)$ の行列 $VR$ を格納する。
<code>lscale</code>	長さ $(n)$ のベクトルを格納する。
<code>rscale</code>	長さ $(n)$ のベクトルを格納する。
<code>rconde</code>	長さ $(n)$ のベクトルを格納する。
<code>rcondv</code>	長さ $(n)$ のベクトルを格納する。
<code>balanc</code>	'N'、'B'、または 'P' でなければならない。デフォルト値は 'N'。
<code>jobv1</code>	引数 <code>v1</code> の存在に基づいて以下のように復元される。 <code>jobv1</code> = 'V' ( <code>v1</code> が存在する場合)、 <code>jobv1</code> = 'N' ( <code>v1</code> が省略された場合)。
<code>jobvr</code>	引数 <code>vr</code> の存在に基づいて以下のように復元される。 <code>jobvr</code> = 'V' ( <code>vr</code> が存在する場合)、 <code>jobvr</code> = 'N' ( <code>vr</code> が省略された場合)。

*sense*                      引数 *rconde* と *rcondv* の存在に基づいて以下のように復元される。  
*sense* = 'B' (*rconde* と *rcondv* の両方が存在する場合)、  
*sense* = 'E' (*rconde* が存在し、*rcondv* が省略された場合)、  
*sense* = 'V' (*rconde* が省略され、*rcondv* が存在する場合)、  
*sense* = 'N' (*rconde* と *rcondv* の両方が省略された場合)。

### アプリケーション・ノート

配列 *work* に必要なワークスペースの大きさがわからない場合は、最初の実行で *lwork* を大きめの値に設定する。終了時に、*work(1)* の値を調べ、以降はその値を使用する。

一般に、商 *alphan(j)/beta(j)* と *alphai(j)/beta(j)* は、簡単にオーバーフローまたはアンダーフローし、*beta(j)* がゼロになる場合もある。したがって、単純に比を計算するのは避けなければならない。ただし、*alphan* と *alphai* (実数型の場合) または *alpha* (複素数型の場合) は、大きさがノルム (A) より常に小さくなり、通常はノルム (A) と比較できる。また、*beta* は、ノルム (B) より常に小さくなり、通常はノルム (B) と比較できる。

# LAPACK 補助ルーチンとユーティリティ・ルーチン

## 5

この章では LAPACK [補助ルーチン](#)と[ユーティリティ・ルーチン](#)に関するインテル® マス・カーネル・ライブラリの実装について説明する。ライブラリは実数と複素の両方のデータに対応した補助ルーチンで構成される。

## 補助ルーチン

LAPACK 補助ルーチンで使用されているルーチン名の規則、数学表記、行列の格納形式は、これまでの章に記載されているドライバおよび計算ルーチンのものと同一である。

次の表は、利用可能な LAPACK 補助ルーチンについての情報をまとめたものである。

表 5-1 [LAPACK 補助ルーチン](#)

ルーチン名	データ型	説明
<a href="#">?lacgv</a>	c, z	複素ベクトルを共役する。
<a href="#">?lacrm</a>	c, z	複素行列に正方実数行列を乗算する。
<a href="#">?lacrt</a>	c, z	複素ベクトル対の線形変換を実行する。
<a href="#">?laesy</a>	c, z	2 × 2 の複素数対称行列の固有値と固有ベクトルを計算する。
<a href="#">?prot</a>	c, z	複素ベクトル対に実余弦と複素正弦を用いて面回転を適用する。
<a href="#">?spmv</a>	c, z	複素対称圧縮行列を使用して複素ベクトルに対する行列 - ベクトル積を計算する。
<a href="#">?spr</a>	c, z	対称圧縮行列の複素数対称階数 1 の更新を実行する。
<a href="#">?symv</a>	c, z	複素対称行列に対して行列 - ベクトル積を計算する。
<a href="#">?syr</a>	c, z	複素対称行列の対称階数 1 の更新を実行する。
<a href="#">i?max1</a>	c, z	実数部分が最大絶対値を持つベクトルの成分のインデックスを検出する。
<a href="#">?sum1</a>	sc, dz	真の絶対値を用いて複素ベクトルの 1- ノルムを実行する。

表 5-1 LAPACK 補助ルーチン ( 続き )

ルーチン名	データ型	説明
<a href="#">?gbtff2</a>	s, d, c, z	非ブロック化バージョンのアルゴリズムを使って一般帯行列の LU 因子分解を計算する。
<a href="#">?gebd2</a>	s, d, c, z	非ブロック化アルゴリズムを使って一般行列を二重対角形式に縮退させる。
<a href="#">?gehd2</a>	s, d, c, z	非ブロック化アルゴリズムを使用して一般正方行列を上 Hessenberg 形式に縮退させる。
<a href="#">?gelq2</a>	s, d, c, z	非ブロック化アルゴリズムを使用して一般矩形行列の LQ 因子分解を計算する。
<a href="#">?geql2</a>	s, d, c, z	非ブロック化アルゴリズムを使用して一般矩形行列の QL 因子分解を計算する。
<a href="#">?geqr2</a>	s, d, c, z	非ブロックアルゴリズムを使用して一般矩形行列の QR 因子分解を計算する。
<a href="#">?gerq2</a>	s, d, c, z	非ブロック化アルゴリズムを使用して一般矩形行列の RQ 因子分解を計算する。
<a href="#">?gesc2</a>	s, d, c, z	?getc2 で計算された完全ピボット演算による LU 因子分解を使用して連立 1 次方程式を解く。
<a href="#">?getc2</a>	s, d, c, z	一般 $n \times n$ 行列の完全ピボット演算による LU 因子分解を計算する。
<a href="#">?getf2</a>	s, d, c, z	一般 $m \times n$ 行列の LU 因子分解を、行交換を伴う部分ピボット演算を用いて計算する ( 非ブロック化アルゴリズム )。
<a href="#">?gtts2</a>	s, d, c, z	?gttrf によって行われた LU 因子分解を使用して、三重対角行列を係数行列とする連立 1 次方程式を解く。
<a href="#">?labrd</a>	s, d, c, z	一般行列の最初の $nb$ 行と列を二重対角形式に縮退させる。
<a href="#">?lacon</a>	s, d, c, z	行列 - ベクトル積の評価にリバース・コミュニケーションを使用して正方行列の 1- ノルムを推定する。
<a href="#">?lacpy</a>	s, d, c, z	1 つの 2 次元配列の一部または全部を他にコピーする。
<a href="#">?ladiv</a>	s, d, c, z	不必要なオーバーフローを回避して、実数演算で複素除算を実行する。
<a href="#">?lae2</a>	s, d	$2 \times 2$ 対称行列の固有値を計算する。
<a href="#">?laebz</a>	s, d	実数対称三重対角行列の指定された値以下の固有値の個数を計算し、また ?stebz ルーチンが必要とする他の処理を実行する。
<a href="#">?laed0</a>	s, d, c, z	?stedc で使用される。縮退されていない対称三重対角行列の固有値と対応する固有ベクトルを分割統治法を使用してすべて求める。

表 5-1 LAPACK 補助ルーチン ( 続き )

ルーチン名	データ型	説明
<a href="#">?laed1</a>	s,d	sstedc/dstedc で使用される。階数 1 対称行列による更新後に対角行列の更新された固有連立方程式を計算する。元の行列が三重対角の場合に使用される。
<a href="#">?laed2</a>	s,d	sstedc/dstedc で使用される。固有値を併合し永年方程式を収縮させる。元の行列が三重対角の場合に使用される。
<a href="#">?laed3</a>	s,d	sstedc/dstedc で使用される。永年方程式の根を探し固有ベクトルを更新する。元の行列が三重対角の場合に使用される。
<a href="#">?laed4</a>	s,d	sstedc/dstedc で使用される。永年方程式の単一根を見つける。
<a href="#">?laed5</a>	s,d	sstedc/dstedc で使用される。2×2 の永年方程式を解く。
<a href="#">?laed6</a>	s,d	sstedc/dstedc で使用される。永久方程式の解の中から Newton ステップの 1 つを計算する。
<a href="#">?laed7</a>	s,d,c,z	?stedc で使用される。階数 1 対称行列による更新後に対角行列の更新された固有連立方程式を計算する。元の行列が密の場合に使用される。
<a href="#">?laed8</a>	s,d,c,z	?stedc で使用される。固有値を併合し永年方程式を収縮させる。元の行列が密の場合に使用される。
<a href="#">?laed9</a>	s,d	sstedc/dstedc で使用される。永年方程式の根を探し固有ベクトルを更新する。元の行列が密の場合に使用される。
<a href="#">?laeda</a>	s,d	?stedc で使用される。対角行列の階数 1 更新を決定する Z ベクトルを計算する。元の行列が密の場合に使用される。
<a href="#">?laein</a>	s,d,c,z	上 Hessenberg 行列の指定された右または左固有ベクトルを逆反復法を用いて計算する。
<a href="#">?laev2</a>	s,d,c,z	2×2 の対称 / エルミート行列の固有値と固有ベクトルを計算する。
<a href="#">?laexc</a>	s,d	Schur 標準形となっている実数の上準三角行列の隣接対角ブロックを、直交相似変換によって交換する。
<a href="#">?lag2</a>	s,d	2×2 一般固有値問題の固有値を、オーバーフロー / アンダーフローが発生しないように必要に応じてスケーリングを行い計算する。
<a href="#">?lags2</a>	s,d	2×2 直交行列 $U$ 、 $V$ 、 $Q$ を計算し、変換された $A$ と $B$ の行が平行であるような $A$ と $B$ にそれらを適用する。
<a href="#">?lagtf</a>	s,d	行交換を伴う部分ピボット演算を用いて行列 $T\lambda I$ の LU 因子分解を計算する。 $T$ は一般三重対角行列、 $\lambda$ はスカラー。
<a href="#">?lagtm</a>	s,d,c,z	$C = \alpha AB + \beta C$ の形式で示される行列 = 行列積を実行し、ここで $A$ は三重対角行列、 $B$ と $C$ は矩形行列、 $\alpha$ と $\beta$ はスカラーで 0、1、または -1 である。

表 5-1 LAPACK 補助ルーチン ( 続き )

ルーチン名	データ型	説明
<a href="#">?lagts</a>	s, d	?lagtf で計算された $\Lambda Y$ 因子分解を用いて連立方程式 $(T-I)x = y$ または $(T-\lambda I)^T x = y$ を解く。ここで $T$ は一般三重対角行列、 $\lambda$ はスカラ。
<a href="#">?lagv2</a>	s, d	実数の $2 \times 2$ 行列束 (A, B) の汎用 Schur 因子分解を計算する。ここで B は上三角である。
<a href="#">?lahqrx</a>	s, d, c, z	ダブルシフト / シングルシフト QR アルゴリズムを用いて、上 Hessenberg 行列の固有値と Schur 因子分解を計算する。
<a href="#">?lahrd</a>	s, d, c, z	$k$ 番目の劣対角よりも下の成分がゼロになるように一般矩形行列の最初の $nb$ 列を縮退させ、A の縮退されていない部分に変換を適用するために必要となる補助行列を返す。
<a href="#">?laic1</a>	s, d, c, z	増加条件推定を 1 ステップ適用する。
<a href="#">?laln2</a>	s, d	指定された形式の $1 \times 1$ または $2 \times 2$ の線形連立方程式を解く。
<a href="#">?lals0</a>	s, d, c, z	最小二乗問題を分割統治 SVD 法を使用して解く過程で乗率を逆に適用する。?gelsd で使用される。
<a href="#">?lalsa</a>	s, d, c, z	コンパクト形式で係数行列の SVD を計算する。?gelsd で使用される。
<a href="#">?lalsd</a>	s, d, c, z	最小二乗問題を解くために A の特異値分解を使用する。
<a href="#">?lamrg</a>	s, d	2 つの独立するソートされたセットの成分を単一のソートされたセットに昇順で併合する置換リストを生成する。
<a href="#">?langb</a>	s, d, c, z	一般帯行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。
<a href="#">?lange</a>	s, d, c, z	一般矩形行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。
<a href="#">?langt</a>	s, d, c, z	一般三重対角行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。
<a href="#">?lanhs</a>	s, d, c, z	上 Hessenberg 行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。
<a href="#">?lansb</a>	s, d, c, z	対称帯行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。
<a href="#">?lanhb</a>	c, z	エルミート帯行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。
<a href="#">?lansp</a>	s, d, c, z	圧縮形式で与えられる対称行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。
<a href="#">?lanhp</a>	c, z	圧縮形式で与えられる複素エルミート行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。

表 5-1 LAPACK 補助ルーチン ( 続き )

ルーチン名	データ型	説明
<a href="#">?lanst/?lanht</a>	s,d/c,z	実数対称または複素エルミート三重対角行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。
<a href="#">?lansy</a>	s,d,c,z	実数 / 複素対称行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。
<a href="#">?lanhe</a>	c,z	複素エルミート行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。
<a href="#">?lanth</a>	s,d,c,z	三角帯行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。
<a href="#">?lantp</a>	s,d,c,z	圧縮形式で与えられる三角行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。
<a href="#">?lantr</a>	s,d,c,z	台形または三角行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。
<a href="#">?lanv2</a>	s,d	2 × 2 実数非対称行列の Schur 因子分解を標準形式で計算する。
<a href="#">?lap1l</a>	s,d,c,z	2 つのベクトルの線形従属性を測定する。
<a href="#">?lapmt</a>	s,d,c,z	行列の列に対して順方向または逆方向置換を実行する。
<a href="#">?lapy2</a>	s,d	$\sqrt{x^2+y^2}$ を返す。
<a href="#">?lapy3</a>	s,d	$\sqrt{x^2+y^2+z^2}$ を返す。
<a href="#">?laqgb</a>	s,d,c,z	?gbequ で計算された行と列のスケール係数を使って一般帯行列をスケールリングする。
<a href="#">?laqge</a>	s,d,c,z	?geequ で計算された行と列のスケール係数を使って一般矩形行列をスケールリングする。
<a href="#">?laqp2</a>	s,d,c,z	行列ブロックに対して列ピボット演算を用いた QR 因子分解を計算する。
<a href="#">?laqps</a>	s,d,c,z	BLAS レベル 3 を使って実数 m × n 行列 A に対して列ピボット演算を用いた QR 因子分解のステップを計算する。
<a href="#">?laqsb</a>	s,d,c,z	?pbequ で計算されたスケール係数を用いて対称/エルミート帯行列をスケールリングする。
<a href="#">?laqsp</a>	s,d,c,z	?ppequ で計算されたスケール係数を用いて圧縮格納形式にある対称 / エルミート行列をスケールリングする。
<a href="#">?laqsy</a>	s,d,c,z	?poequ で計算されたスケール係数を用いて対称/エルミート行列をスケールリングする。
<a href="#">?laqtr</a>	s,d	実数準三角連立方程式または特殊な形式の複素準三角連立方程式を実数演算で解く。
<a href="#">?larlv</a>	s,d,c,z	三重対角行列 $LDL^T - \sigma I$ の行 $b1$ から $bn$ にある部分行列に対して逆行列の ( スケールされた ) $r$ 番目の列を計算する。

表 5-1 LAPACK 補助ルーチン ( 続き )

ルーチン名	データ型	説明
<a href="#">?lar2v</a>	s,d,c,z	一連の $2 \times 2$ 対称 / エルミート行列に、実数余弦と実数 / 複素正弦を持つ面回転のベクトルを両側から適用する。
<a href="#">?larf</a>	s,d,c,z	一般矩形行列に基本リフレクタを適用する。
<a href="#">?larfb</a>	s,d,c,z	ブロック・リフレクタまたはその転置 / 共役転置を一般矩形行列に適用する。
<a href="#">?larfg</a>	s,d,c,z	基本リフレクタ (Householder 行列) を生成する
<a href="#">?larft</a>	s,d,c,z	ブロック・リフレクタ $H = I - VTV^H$ の三角係数 $T$ を生成する。
<a href="#">?larfx</a>	s,d,c,z	リフレクタが次数 $\leq 10$ を持つ場合、ループの繰り返しを避けながら、基本リフレクタを一般矩形行列に適用する。
<a href="#">?largv</a>	s,d,c,z	実数余弦および実数 / 複素正弦を持つ面回転のベクトルを生成する。
<a href="#">?larnv</a>	s,d,c,z	乱数ベクトルを一様分布または正規分布で返す。
<a href="#">?larrb</a>	s,d	より高い精度で固有値を見つけるための限定二分法を与える。
<a href="#">?larre</a>	s,d	三重対角行列 $T$ が与えられたとき、小さい非対角成分をゼロに設定し、縮退されていないブロック $T_i$ に対して本表現と固有値を探す。
<a href="#">?larrrf</a>	s,d	少なくとも 1 つの固有値が相対的に分離されているような新しい比較的安定な表現を探す。
<a href="#">?larrrv</a>	s,d,c,z	$L$ 、 $D$ 、 $L D L^T$ の固有値が与えられたとき、三重対角行列 $T = L D L^T$ の固有ベクトルを計算する。
<a href="#">?lartg</a>	s,d,c,z	実数余弦と実数 / 複素正弦を持つ面回転を生成する。
<a href="#">?lartv</a>	s,d,c,z	実数余弦と実数 / 複素制限を持つ面回転のベクトルをベクトル対の成分に適用する。
<a href="#">?laruv</a>	s,d	$n$ 個の実数乱数のベクトルを一様分布で返す。
<a href="#">?larz</a>	s,d,c,z	(?tzzrf が返したとおりの) 基本リフレクタを一般行列に適用する。
<a href="#">?larzb</a>	s,d,c,z	ブロック・リフレクタまたはその転置 / 共役転置を一般矩形行列に適用する。
<a href="#">?larzt</a>	s,d,c,z	ブロック・リフレクタ $H = I - VTV^H$ の三角係数 $T$ を生成する。
<a href="#">?las2</a>	s,d	$2 \times 2$ 三角行列の特異値を計算する。
<a href="#">?lascl</a>	s,d,c,z	一般矩形行列に $C_{to}/C_{from}$ として定義される実数スカラを乗算する。
<a href="#">?lasd0</a>	s,d	対角 $d$ と対角外 $e$ を持つ実数上二重対角 $n \times m$ 行列 $B$ の特異値を計算する。?bdsdc で使用される。



表 5-1 LAPACK 補助ルーチン ( 続き )

ルーチン名	データ型	説明
<a href="#">?lasd1</a>	s, d	指定サイズの上二重対角行列 B の SVD を計算する。?bdsdc で使用される。
<a href="#">?lasd2</a>	s, d	2 つの特異値のセットをソートされた単一のセットに併合する。?bdsdc で使用される。
<a href="#">?lasd3</a>	s, d	D と Z 内の値によって定義されている永年方程式に対して根のすべての平方根を求め、次に行列乗算によって特異ベクトルを更新する。?bdsdc で使用される。
<a href="#">?lasd4</a>	s, d	正値対角行列に対する正値対称階数 1 更新の、i 番目の更新された固有値の平方根を計算する。?bdsdc で使用される。
<a href="#">?lasd5</a>	s, d	2 × 2 対角行列の正値対称階数 1 更新から、i 番目の固有値の平方根を計算する。?bdsdc で使用される。
<a href="#">?lasd6</a>	s, d	2 つの小さい行列を行追加によって併合して得た、更新された上二重対角行列の SVD を計算する。?bdsdc で使用される。
<a href="#">?lasd7</a>	s, d	2 つの特異値のセットをソートされた単一のセットに併合する。続いて、問題の大きさの収縮を試みる。?bdsdc で使用される。
<a href="#">?lasd8</a>	s, d	永年方程式の根の平方根を求め、D の各成分に対して最も近い 2 つの極までの距離を格納する。?bdsdc で使用される。
<a href="#">?lasd9</a>	s, d	永年方程式の根の平方根を求め、D の各成分に対して最も近い 2 つの極までの距離を格納する。?bdsdc で使用される。
<a href="#">?lasda</a>	s, d	対角 d と対角外 e を持つ実数上二重対角行列の特異値分解 (SVD) を計算する。?bdsdc で使用される。
<a href="#">?lasdq</a>	s, d	対角 d と対角外 e を持つ実二重対角行列を計算する。?bdsdc で使用される。
<a href="#">?lasdt</a>	s, d	二重対角分割統治に対する部分問題のツリーを生成する。?bdsdc で使用される。
<a href="#">?laset</a>	s, d, c, z	行列の非対角成分と対角成分を指定された値に初期化する。
<a href="#">?lasq1</a>	s, d	実数平方二重対角行列の特異値を計算する。?bdsqr で使用される。
<a href="#">?lasq2</a>	s, d	相対的に高い精度で、qd 配列 z に関する対称正定値三重対角行列のすべての固有値を計算する。?bdsqr と ?stegr で使用される。
<a href="#">?lasq3</a>	s, d	収縮をチェックし、シフトを計算し、dqds を呼び出す。?bdsqr で使用される。
<a href="#">?lasq4</a>	s, d	以前の変換で得られた d の値を用いて最小固有値に対する近似を計算する。?bdsqr で使用される。
<a href="#">?lasq5</a>	s, d	ping-pong 形式で dqds 変換を 1 つ計算する。?bdsqr と ?stegr で使用される。

表 5-1 LAPACK 補助ルーチン ( 続き )

ルーチン名	データ型	説明
<a href="#">?lasq6</a>	s,d	ping-pong 形式で $dqd$ 変換を 1 つ計算する。 <a href="#">?bdsqr</a> と <a href="#">?stegr</a> で使用される。
<a href="#">?lasr</a>	s,d,c,z	一般矩形行列に一連の面回転を適用する。
<a href="#">?lasrt</a>	s,d	数を昇順または降順でソートする。
<a href="#">?lassq</a>	s,d,c,z	スケーリング形式で表現された二乗和を更新する。
<a href="#">?lasv2</a>	s,d	$2 \times 2$ 三角行列の特異値分解を計算する。
<a href="#">?laswp</a>	s,d,c,z	一般矩形行列に対して一連の行交換を実行する。
<a href="#">?lasy2</a>	s,d	行列の次数が 1 または 2 のシルベスター行列式を解く。
<a href="#">?lasyf</a>	s,d,c,z	対角ピボット演算法を使って実数 / 複素対称行列の部分因子分解を計算する。
<a href="#">?lahef</a>	c,z	対角ピボット演算法を使って複素エルミート無限行列の部分因子分解を計算する。
<a href="#">?latbs</a>	s,d,c,z	三角帯連立方程式を解く。
<a href="#">?latdf</a>	s,d,c,z	<a href="#">?getc2</a> による $n \times n$ 行列の LU 因子分解を使い、Dif 推定値の逆数に対する影響を計算する。
<a href="#">?latps</a>	s,d,c,z	圧縮形式で格納されている行列を持った三角連立方程式を解く。
<a href="#">?latrd</a>	s,d,c,z	直交 / ユニタリ相似変換を用いて、対称 / エルミート行列 A の最初の $nb$ 列と $nb$ 行を実数三重対角形式に縮退する。
<a href="#">?latrs</a>	s,d,c,z	オーバフローを防ぐために設定されたスケール係数を持つ三角連立方程式を解く。
<a href="#">?latrz</a>	s,d,c,z	上台形行列を直交 / ユニタリ変換によって因子分解する。
<a href="#">?lauu2</a>	s,d,c,z	積 $UU^H$ または $L^H L$ を計算する。ここで $U$ と $L$ は上三角または下三角行列 ( 非ブロック化アルゴリズム )。
<a href="#">?lauum</a>	s,d,c,z	積 $UU^H$ または $L^H L$ を計算する。ここで $U$ と $L$ は上三角または下三角行列 ( ブロック化アルゴリズム )。
<a href="#">?org2l/?ung2l</a>	s,d/c,z	<a href="#">?geqlf</a> で求めた QL 因子分解から、全部または一部の直交 / ユニタリ行列 Q を生成する ( 非ブロック化アルゴリズム )。
<a href="#">?org2r/?ung2r</a>	s,d/c,z	<a href="#">?geqrf</a> で求めた QR 因子分解から、全部または一部の直交 / ユニタリ行列 Q を生成する ( 非ブロック化アルゴリズム )。
<a href="#">?orgl2/?ungl2</a>	s,d/c,z	<a href="#">?gelqf</a> で求めた LQ 因子分解から、全部または一部の直交 / ユニタリ行列 Q を生成する ( 非ブロック化アルゴリズム )。
<a href="#">?org2/?ungr2</a>	s,d/c,z	<a href="#">?gerqf</a> で求めた RQ 因子分解から、全部または一部の直交 / ユニタリ行列 Q を生成する ( 非ブロック化アルゴリズム )。
<a href="#">?orm2l/?unm2l</a>	s,d/c,z	一般行列と <a href="#">?geqlf</a> で求めた QL 因子分解の直交 / ユニタリ行列を乗算する ( 非ブロック化アルゴリズム )。

表 5-1 LAPACK 補助ルーチン ( 続き )

ルーチン名	データ型	説明
<a href="#">?orm2r/?unm2r</a>	s,d/c,z	一般行列と ?geqrf で求めた QR 因子分解の直交 / ユニタリ行列とを乗算する ( 非ブロック化アルゴリズム )。
<a href="#">?orml2/?unml2</a>	s,d/c,z	一般行列と ?gelqf で求めた LQ 因子分解の直交 / ユニタリ行列とを乗算する ( 非ブロック化アルゴリズム )。
<a href="#">?ormr2/?unmr2</a>	s,d/c,z	一般行列と ?gerqf で求めた RQ 因子分解の直交 / ユニタリ行列とを乗算する ( 非ブロック化アルゴリズム )。
<a href="#">?ormr3/?unmr3</a>	s,d/c,z	一般行列と ?tzzrf で求めた RZ 因子分解の直交 / ユニタリ行列とを乗算する ( 非ブロック化アルゴリズム )。
<a href="#">?pbtf2</a>	s,d,c,z	対称 / エルミート正定値帯行列のコレスキー因子分解を計算する ( 非ブロック化アルゴリズム )。
<a href="#">?potf2</a>	s,d,c,z	対称 / エルミート正定値行列のコレスキー因子分解を行う ( 非ブロック化アルゴリズム )。
<a href="#">?ptts2</a>	s,d,c,z	?pttrf で計算した $L$ 、 $D$ 、 $L^H$ 因子分解を用いて、形式 $AX=B$ の三重対角連立方程式を解く。
<a href="#">?rscl</a>	s,d,cs, zd	ベクトルに実数スカラの逆数を乗算する。
<a href="#">?sygs2/?hegs2</a>	s,d/c,z	?potrf で得られた因子分解の結果を用いて、対称 / エルミート汎用固有値問題を標準形式に縮退させる ( 非ブロック化アルゴリズム )。
<a href="#">?sytd2/?hetd2</a>	s,d/c,z	直交 / ユニタリ相似変換を用いて、対称 / エルミート行列を実数対称三重対角形式に縮退させる ( 非ブロック化アルゴリズム )。
<a href="#">?sytf2</a>	s,d,c,z	対角ピボット演算法を使用して、実数 / 複素不定値対称行列の因子分解を計算する ( 非ブロック化アルゴリズム )。
<a href="#">?hetf2</a>	c,z	対角ピボット演算法を使用して、複素エルミート行列の因子分解を計算する ( 非ブロック化アルゴリズム )。
<a href="#">?tgex2</a>	s,d,c,z	直交 / ユニタリ等価変換を使用して、( 準 ) 上三角行列のペア中の隣接対角ブロックを交換する。
<a href="#">?tgsy2</a>	s,d,c,z	汎用シルベスター式を解く ( 非ブロック化アルゴリズム )。
<a href="#">?trti2</a>	s,d,c,z	三角行列の逆行列を計算する ( 非ブロック化アルゴリズム )。

### ?lacgv

複素ベクトルを共役する。

---

#### 構文

```
call clacgv( n, x, incx )
call zlacgv( n, x, incx )
```

#### 説明

このルーチンは、長さ  $n$ 、増分  $incx$  の複素ベクトル  $x$  を共役する ( 付録 B の「[BLAS のベクトル引数](#)」を参照 )。

#### 入力パラメータ

$n$	INTEGER。ベクトル $x$ の長さ ( $n \geq 0$ )。
$x$	COMPLEX (clacgv の場合) COMPLEX*16 (zlacgv の場合) 配列、次元は $(1+(n-1)* incx )$ 。 共役する長さ $n$ のベクトルを格納する。
$incx$	INTEGER。 $x$ の連続する成分の間隔。

#### 出力パラメータ

$x$	conjg(x) で上書きされる。
-----	-------------------

### ?lacrm

複素行列に正方実数行列を乗算する。

---

#### 構文

```
call clacrm( m, n, a, lda, b, ldb, c, ldc, rwork )
call zlacrm( m, n, a, lda, b, ldb, c, ldc, rwork )
```

## 説明

このルーチンは、次の形式の単純な行列 - 行列乗算を実行する。

$$C = A * B$$

ここで、 $A$  は  $m \times n$  の複素行列、 $B$  は  $n \times n$  の実行列、 $C$  は  $m \times n$  の複素行列である。

## 入力パラメータ

$m$	INTEGER。行列 $A$ の行数と行列 $C$ の行数 ( $m \geq 0$ )。
$n$	INTEGER。行列 $B$ の列数と行数および行列 $C$ の列数 ( $n \geq 0$ )。
$a$	COMPLEX (clacrm の場合) COMPLEX*16 (zlacrm の場合)  配列、次元は ( $lda, n$ )。 $m \times n$ 行列 $A$ が格納される。
$lda$	INTEGER。配列 $a$ のリーディング・ディメンジョン。 $lda \geq \max(1, m)$ 。
$b$	REAL (clacrm の場合) DOUBLE PRECISION (zlacrm の場合)  配列、次元は ( $ldb, n$ )。 $n \times n$ 行列 $B$ が格納される。
$ldb$	INTEGER。配列 $b$ のリーディング・ディメンジョン。 $ldb \geq \max(1, n)$
$ldc$	INTEGER。出力配列 $c$ のリーディング・ディメンジョン。 $ldc \geq \max(1, n)$
$rwork$	REAL (clacrm の場合) DOUBLE PRECISION (zlacrm の場合)  ワークスペース配列、次元は ( $2*m*n$ )。

## 出力パラメータ

$c$	COMPLEX (clacrm の場合) COMPLEX*16 (zlacrm の場合)  配列、次元は ( $ldc, n$ )。 $m \times n$ 行列 $C$ が格納される。
-----	---

## ?lacrt

複素ベクトル対の線形変換を実行する。

---

### 構文

```
call clacrt( n, cx, incx, cy, incy, c, s )
call zlacrt( n, cx, incx, cy, incy, c, s )
```

### 説明

このルーチンは次の変換を実行する。

$$\begin{pmatrix} c & s \\ -s & c \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \Rightarrow \begin{pmatrix} x \\ y \end{pmatrix}$$

$C$  と  $S$  は複素スカラ、 $x$  と  $y$  は複素ベクトルである。

### 入力パラメータ

$n$	INTEGER。ベクトル $cx$ とベクトル $cy$ の成分の数 ( $n \geq 0$ )。
$cx, cy$	COMPLEX (clacrt の場合) COMPLEX*16 (zlacrt の場合)  配列、次元は ( $n$ )。 それぞれ入力ベクトル $x$ と $y$ を格納する。
$incx$	INTEGER。 $cx$ の連続する成分間の増分。
$incy$	INTEGER。 $cy$ の連続する成分間の増分。
$c, s$	COMPLEX (clacrt の場合) COMPLEX*16 (zlacrt の場合)  変換行列を定義する複素スカラ

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix}$$

### 出力パラメータ

$cx$	$c*x + s*y$ で上書きされる。
------	----------------------

$cy$   $-s*x + c*y$  で上書きされる。

## ?laesy

$2 \times 2$  の複素対称行列の固有値と固有ベクトルを計算し、固有ベクトルの行列のノルムがしきい値より大きいことを確認する。

### 構文

```
call claesy (a, b, c, rt1, rt2, evscal, cs1, sn1)
call zlaesy (a, b, c, rt1, rt2, evscal, cs1, sn1)
```

### 説明

このルーチンは、固有ベクトルの行列のノルムがしきい値より大きければ、次の  $2 \times 2$  対称行列の固有分解を実行する。

$$\begin{bmatrix} a & b \\ b & c \end{bmatrix}$$

$rt1$  は絶対値が大きい方の固有値、 $rt2$  は絶対値が小さい方の固有値である。固有ベクトルが計算されると、 $(cs1, sn1)$  は  $rt1$  の単位固有ベクトルで上書きされ、ゆえに、

$$\begin{bmatrix} cs1 & sn1 \\ -sn1 & cs1 \end{bmatrix} \cdot \begin{bmatrix} a & b \\ b & c \end{bmatrix} \cdot \begin{bmatrix} cs1 & -sn1 \\ sn1 & cs1 \end{bmatrix} = \begin{bmatrix} rt1 & 0 \\ 0 & rt2 \end{bmatrix}$$

### 入力パラメータ

$a, b, c$       COMPLEX (claesy の場合)  
                  COMPLEX\*16 (zlaesy の場合)  
                  入力行列の成分。

### 出力パラメータ

<i>rt1, rt2</i>	COMPLEX (claesy の場合) COMPLEX*16 (zlaesy の場合)  それぞれ、大きい方 (モジュラス) の固有値、小さい方 (モジュラス) の固有値。
<i>evscal</i>	COMPLEX (claesy の場合) COMPLEX*16 (zlaesy の場合)  固有ベクトル行列を正規直交化するためにスケーリングした複素値。 <i>evscal</i> がゼロの場合、固有ベクトルは計算されていない。これには 2 つの理由が考えられる。2×2 行列を対角化できなかったか、スケーリング前の固有ベクトルの行列のノルムがしきい値 <i>thresh</i> (0.1E0 に設定) よりも大きかったからである。
<i>cs1, sn1</i>	COMPLEX (claesy の場合) COMPLEX*16 (zlaesy の場合)  <i>evscal</i> がゼロでない場合、( <i>cs1, sn1</i> ) は <i>rt1</i> の単位右固有ベクトルである。

---

## ?rot

複素ベクトル対に実余弦と複素正弦を用いて  
面回転を適用する。

---

### 構文

```
call crot( n, cx, incx, cy, incy, c, s )  
call zrot( n, cx, incx, cy, incy, c, s )
```

### 説明

このルーチンは面回転を適用する。ここで余弦 (*c*) は実数、正弦 (*s*) は複素数、ベクトル *cx* と *cy* は複素数である。このルーチンと実数において等価な関数が BLAS にある (第 2 章の [?rot](#) を参照)。

### 入力パラメータ

*n*                    INTEGER。ベクトル *cx* とベクトル *cy* の成分の数。



<i>cx</i> , <i>cy</i>	COMPLEX (crot の場合 ) COMPLEX*16 (zrot の場合 ) 次元 ( <i>n</i> ) の配列で、それぞれ入力ベクトル <i>x</i> と <i>y</i> を格納する。
<i>incx</i>	INTEGER。 <i>cx</i> の連続する成分間の増分。
<i>incy</i>	INTEGER。 <i>cy</i> の連続する成分間の増分。
<i>c</i>	REAL (crot の場合 ) DOUBLE PRECISION (zrot の場合 )
<i>s</i>	COMPLEX (crot の場合 ) COMPLEX*16 (zrot の場合 ) 回転を定義する値で、 $\begin{bmatrix} c & s \\ -\text{conjg}(s) & c \end{bmatrix}$ $c*c + s*\text{conjg}(s) = 1.0$

### 出力パラメータ

<i>cx</i>	$c*x + s*y$ で上書きされる。
<i>cy</i>	$-\text{conjg}(s)*x + c*y$ で上書きされる。

## ?spmv

複素対称圧縮行列を使用して複素ベクトルに  
対する行列-ベクトル積を計算する。

### 構文

```
call cspmv( uplo, n, alpha, ap, x, incx, beta, y, incy )
call zspmv( uplo, n, alpha, ap, x, incx, beta, y, incy )
```

### 説明

これらのルーチンは、次のように定義される行列 - ベクトル演算を実行する。

$y := \text{alpha} * a * x + \text{beta} * y,$

ここで、

$\alpha$  と  $\beta$  は、複素スカラーである。

$x$  と  $y$  は  $n$  個の成分を持つ複素ベクトルである。

$a$  は、圧縮形式で与える  $n \times n$  の対称行列である。

このルーチンと実数において等価な関数が BLAS にある ( 第 2 章の [?spmv](#) を参照 )。

### 入力パラメータ

<code>uplo</code>	CHARACTER*1。次に示すように、行列 $a$ の上三角部分と下三角部分のどちらを圧縮形式の配列 $ap$ に与えるかを指定する。  <code>uplo = 'U'</code> または <code>'u'</code> の場合、行列 $a$ の上三角部分を配列 $ap$ に格納する。 <code>uplo = 'L'</code> または <code>'l'</code> の場合、行列 $a$ の下三角部分を配列 $ap$ に格納する。
<code>n</code>	INTEGER。行列 $a$ の次数を指定する。 $n$ の値はゼロ以上でなければならない。
<code>alpha, beta</code>	COMPLEX (cspmv の場合) COMPLEX*16 (zspmv の場合)  複素スカラー $\alpha$ と $\beta$ を指定する。 $\beta$ をゼロに設定した場合は、 $y$ を設定する必要はない。
<code>ap</code>	COMPLEX (cspmv の場合) COMPLEX*16 (zspmv の場合)  配列、次元は $((n*(n+1))/2)$ 以上。 <code>uplo = 'U'</code> または <code>'u'</code> と指定した場合は、 $ap(1)$ に $a(1, 1)$ が、 $ap(2)$ に $a(1, 2)$ が、 $ap(3)$ に $a(2, 2)$ がそれぞれ格納されるように、配列 $ap$ には順番にパックされた対称行列の上三角部分を列ごとに格納しなければならない。 また、 <code>uplo = 'L'</code> または <code>'l'</code> と指定した場合は、 $ap(1)$ に $a(1, 1)$ が、 $ap(2)$ に $a(2, 1)$ が、 $ap(3)$ に $a(3, 1)$ がそれぞれ格納されるように、配列 $ap$ には順番にパックされた対称行列の下三角部分を列ごとに格納しなければならない。
<code>x</code>	COMPLEX (cspmv の場合) COMPLEX*16 (zspmv の場合)  配列、次元は $(1 + (n - 1)*abs(incx))$ 以上。このルーチンに入る前に、増分された配列 $x$ に $n$ 個の成分を持つベクトル $x$ を格納しなければならない。

<i>incx</i>	INTEGER。x の成分に対する増分を指定する。 <i>incx</i> の値はゼロであってはならない。
<i>y</i>	COMPLEX (cspmv の場合) COMPLEX*16 (zspmv の場合)  配列、次元は $(1 + (n - 1) * \text{abs}(\text{incy}))$ 以上。このルーチンに入る前に、増分された配列 <i>y</i> に <i>n</i> 個の成分を持つベクトル <i>y</i> を格納しなければならない。
<i>incy</i>	INTEGER。 <i>y</i> の成分に対する増分を指定する。 <i>incy</i> の値はゼロであってはならない。

### 出力パラメータ

<i>y</i>	更新されたベクトル <i>y</i> によって上書きされる。
----------	--------------------------------

## ?spr

対称圧縮行列の複素数対称階数 *l* の更新を実行する。

### 構文

```
call cspr( uplo, n, alpha, x, incx, ap )
call zspr( uplo, n, alpha, x, incx, ap )
```

### 説明

?spr ルーチンは、次のように定義される行列 - ベクトル演算を実行する。

$$a := \alpha * x * \text{conjg}(x') + a$$

*alpha* は複素数のスカラである。

*x* は、*n* 個の成分を持つ複素数ベクトルである。

*a* は、圧縮形式で与える  $n \times n$  の対称行列である。

このルーチンと実数において等価な関数が BLAS にある ( 第 2 章の [?spr](#) を参照 )。

## 入力パラメータ

<i>uplo</i>	<p>CHARACTER*1。次に示すように、行列 <i>a</i> の上三角部分と下三角部分のどちらを圧縮形式の配列 <i>ap</i> に与えるかを指定する。</p> <p><i>uplo</i> = 'U' または 'u' の場合、行列 <i>a</i> の上三角部分を配列 <i>ap</i> に格納する。</p> <p><i>uplo</i> = 'L' または 'l' の場合、行列 <i>a</i> の下三角部分を配列 <i>ap</i> に格納する。</p>
<i>n</i>	<p>INTEGER。行列 <i>a</i> の次数を指定する。<i>n</i> の値はゼロ以上でなければならない。</p>
<i>alpha</i>	<p>COMPLEX (cspr の場合) COMPLEX*16 (zspr の場合)</p> <p>スカラ <i>alpha</i> を指定する。</p>
<i>x</i>	<p>COMPLEX (cspr の場合) COMPLEX*16 (zspr の場合)</p> <p>配列、次元は <math>(1 + (n - 1) * \text{abs}(\text{incx}))</math> 以上。このルーチンに入る前に、増分された配列 <i>x</i> に <i>n</i> 個の成分を持つベクトル <i>x</i> を格納しなければならない。</p>
<i>incx</i>	<p>INTEGER。<i>x</i> の成分に対する増分を指定する。<i>incx</i> の値はゼロであってはならない。</p>
<i>ap</i>	<p>COMPLEX (cspr の場合) COMPLEX*16 (zspr の場合)</p> <p>配列、次元は <math>((n * (n + 1)) / 2)</math> 以上。<i>uplo</i> = 'U' または 'u' と指定した場合は、このルーチンに入る前に、<i>ap</i>(1) に <i>a</i>(1, 1) が、<i>ap</i>(2) に <i>a</i>(1, 2) が、<i>ap</i>(3) に <i>a</i>(2, 2) がそれぞれ格納されるように、配列 <i>ap</i> には、順番にパックされた対称行列の上三角部分を列ごとに格納しなければならない。</p> <p><i>uplo</i> = 'L' または 'l' と指定した場合は、このルーチンに入る前に、<i>ap</i>(1) に <i>a</i>(1, 1) が、<i>ap</i>(2) に <i>a</i>(2, 1) が、<i>ap</i>(3) に <i>a</i>(3, 1) がそれぞれ格納されるように、配列 <i>ap</i> には、順番にパックされた対称行列の下三角部分を列ごとに格納しなければならない。</p> <p>直交成分の虚数部分はゼロとみなすため設定する必要はなく、また、出力ではゼロに設定される。</p>

### 出力パラメータ

*ap*                    *uplo* = 'U' または 'u' と指定した場合は、更新された行列の上三角部分によって上書きされる。

*uplo* = 'L' または 'l' の場合は更新された行列の下三角部分によって上書きされる。

## ?symv

複素対称行列に対して行列 - ベクトル積を計算する。

### 構文

```
call csymv( uplo, n, alpha, a, lda, x, incx, beta, y, incy )
call zsymv( uplo, n, alpha, a, lda, x, incx, beta, y, incy )
```

### 説明

これらのルーチンは、次のように定義される行列 - ベクトル演算を実行する。

$$y := \alpha * a * x + \beta * y,$$

*alpha* と *beta* は、複素スカラである。

*x* と *y* は *n* 個の成分を持つ複素ベクトルである。

*a* は、 $n \times n$  の対称複素行列である。

このルーチンと実数において等価な関数が BLAS にある ( 第 2 章の [?symv](#) を参照 )。

### 入力パラメータ

*uplo*                    CHARACTER\*1。次に示すように、配列 *a* の上三角部分と下三角部分のどちらが参照されるかを指定する。

*uplo* = 'U' または 'u' の場合、配列 *a* の上三角部分が参照される。

*uplo* = 'L' または 'l' の場合、配列 *a* の下三角部分が参照される。

*n*                        INTEGER。行列 *a* の次数を指定する。*n* の値はゼロ以上でなければならない。

<i>alpha, beta</i>	COMPLEX (csymv の場合) COMPLEX*16 (zsymv の場合)  スカラ <i>alpha</i> と <i>beta</i> を指定する。 <i>beta</i> をゼロに設定した場合は、 <i>y</i> を設定する必要はない。
<i>a</i>	COMPLEX (csymv の場合) COMPLEX*16 (zsymv の場合)  配列、次元は ( <i>lda</i> , <i>n</i> )。 <i>uplo</i> = 'U' または 'u' と指定した場合は、このルーチンに入る前に、配列 <i>a</i> の先頭の $n \times n$ の上三角部分には対称行列の上三角部分を格納しなければならない、 <i>a</i> の厳密な下三角部分は参照されない。 <i>uplo</i> = 'L' または 'l' と指定した場合は、このルーチンに入る前に、配列 <i>a</i> の先頭の $n \times n$ の下三角部分には対称行列の下三角部分を格納しなければならない、 <i>a</i> の厳密な上三角部分は参照されない。
<i>lda</i>	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 <i>a</i> の第 1 次元を指定する。 <i>lda</i> の値は、 $\max(1, n)$ 以上でなければならない。
<i>x</i>	COMPLEX (csymv の場合) COMPLEX*16 (zsymv の場合)  配列、次元は $(1 + (n - 1) * \text{abs}(\text{incx}))$ 以上。このルーチンに入る前に、増分された配列 <i>x</i> に <i>n</i> 個の成分を持つベクトル <i>x</i> を格納しなければならない。
<i>incx</i>	INTEGER。 <i>x</i> の成分に対する増分を指定する。 <i>incx</i> の値はゼロであってはならない。
<i>y</i>	COMPLEX (csymv の場合) COMPLEX*16 (zsymv の場合)  配列、次元は $(1 + (n - 1) * \text{abs}(\text{incy}))$ 以上。このルーチンに入る前に、増分された配列 <i>y</i> に <i>n</i> 個の成分を持つベクトル <i>y</i> を格納しなければならない。
<i>incy</i>	INTEGER。 <i>y</i> の成分の増分を指定する。 <i>incy</i> の値はゼロであってはならない。

## 出力パラメータ

<i>y</i>	更新されたベクトル <i>y</i> によって上書きされる。
----------	--------------------------------

## ?syr

複素対称行列の対称階数 1 の更新を実行する。

### 構文

```
call csyr( uplo, n, alpha, x, incx, a, lda )
call zsyr( uplo, n, alpha, x, incx, a, lda )
```

### 説明

このルーチンは、次のように定義される対称階数 1 の演算を実行する。

$$a := \alpha * x * x' + a$$

ここで、

$\alpha$  は複素数のスカラーである。

$x$  は、 $n$  個の成分を持つ複素数ベクトルである。

$a$  は、 $n \times n$  の複素対称行列である。

このルーチンと実数において等価な関数が BLAS にある ( 第 2 章の [?syr](#) を参照 )。

### 入力パラメータ

$uplo$	CHARACTER*1。次に示すように、配列 $a$ の上三角部分と下三角部分のどちらが参照されるかを指定する。  $uplo = 'U'$ または $'u'$ の場合、配列 $a$ の上三角部分が参照される。 $uplo = 'L'$ または $'l'$ の場合、配列 $a$ の下三角部分が参照される。
$n$	INTEGER。行列 $a$ の次数を指定する。 $n$ の値は、ゼロ以上でなければならない。
$\alpha$	COMPLEX (csyr の場合) COMPLEX*16 (zsyr の場合)  スカラー $\alpha$ を指定する。
$x$	COMPLEX (csyr の場合) COMPLEX*16 (zsyr の場合)

配列、次元は  $(1 + (n - 1) * \text{abs}(\text{incx}))$  以上。このルーチンに入る前に、増分された配列  $x$  に  $n$  個の成分を持つベクトル  $x$  を格納しなければならない。

*incx*            INTEGER。  $x$  の成分に対する増分を指定する。*incx* の値はゼロであってはならない。

*a*                COMPLEX (csyr の場合)  
COMPLEX\*16 (zsyr の場合)

配列、次元は  $(lda, n)$ 。*uplo* = 'U' または 'u' と指定した場合は、このルーチンに入る前に、配列  $A$  の先頭の  $n \times n$  の上三角部分には対称行列の上三角部分を格納しなければならない、 $A$  の厳密な下三角部分は参照されない。

*uplo* = 'L' または 'l' と指定した場合は、このルーチンに入る前に、配列  $a$  の先頭の  $n \times n$  の下三角部分には対称行列の下三角部分を格納しなければならない、 $A$  の厳密な上三角部分は参照されない。

*lda*             INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 $a$  の第 1 次元を指定する。*lda* の値は、 $\max(1, n)$  以上でなければならない。

### 出力パラメータ

*a*                *uplo* = 'U' または 'u' と指定した場合は、配列  $a$  の上三角部分が、更新された行列の上三角部分によって上書きされる。

*uplo* = 'L' または 'l' と指定した場合は、配列  $A$  の下三角部分が、更新された行列の下三角部分によって上書きされる。

---

## i?max1

実数部分が最大絶対値を持つベクトルの成分のインデックスを検出する。

---

### 構文

```
index = icmax1( n, cx, incx )
```

```
index = izmax1( n, cx, incx )
```



## 説明

複素ベクトル  $cx$  が与えられたとき、 $i?max1$  関数は実数部分が最大絶対値を持つベクトル成分のインデックスを返す。これら関数は **BLAS** 関数 `icamax/izamax` を元になっているが、実数部分の絶対値を使用している。`clacon/zlacon` との併用を意図している。

## 入力パラメータ

$n$  INTEGER。ベクトル  $cx$  の成分の数を指定する。

$cx$  COMPLEX( $icmax1$  の場合)  
COMPLEX\*16( $izmax1$  の場合)

配列、次元は  $(1+(n-1)*abs(incx))$  以上。

入力ベクトルが格納される。

$incx$  INTEGER。 $cx$  の連続する成分の間隔を指定する。

## 出力パラメータ

$index$  INTEGER。実数部分が最大絶対値を持つベクトル成分のインデックスが格納される。

---

## ?sum1

真の絶対値を用いて複素ベクトルの 1- ノルムを実行する。

---

## 構文

```
res = sctsum1( n, cx, incx )  
res = dzctsum1( n, cx, incx )
```

## 説明

複素ベクトル  $cx$  が与えられたとき、`sctsum1/dzctsum1` 関数は、ベクトル成分の絶対値の合計を取り、それぞれ単精度 / 倍精度の結果を返す。これら関数はレベル 1 **BLAS** の [scasum/dzasum](#) を元になっているが、真の絶対値を使用し、[clacon/zlacon](#) と併せて使用するよう設計されている。

### 入力パラメータ

<i>n</i>	INTEGER。ベクトル <i>cx</i> の成分の数を指定する。
<i>cx</i>	COMPLEX (scsum1 の場合) COMPLEX*16 (dzsum1) の場合  配列、次元は $(1+(n-1)*abs(incx))$ 以上。  成分を合計する入力ベクトルを格納する。
<i>incx</i>	INTEGER。 <i>cx</i> の連続する成分間の間隔を指定する ( <i>incx</i> > 0)。

### 出力パラメータ

<i>res</i>	REAL (scsum1 の場合) DOUBLE PRECISION (dzsum1 の場合)  絶対値の合計が格納される。
------------	---

---

## ?gbtf2

非ブロック化バージョンのアルゴリズムを使って  
一般帯行列の LU 因子分解を計算する。

---

### 構文

```
call sgbtf2( m, n, kl, ku, ab, ldab, ipiv, info )  
call dgbtf2( m, n, kl, ku, ab, ldab, ipiv, info )  
call cgbtf2( m, n, kl, ku, ab, ldab, ipiv, info )  
call zgbtf2( m, n, kl, ku, ab, ldab, ipiv, info )
```

### 説明

このルーチンは、*kl* 個の劣対角と *ku* 個の優対角を含む  $m \times n$  の一般実数 / 複素帯行列 *A* の LU 因子分解を実行する。このルーチンは行交換を伴う部分ピボット演算を使用しており、また、非ブロック化ルーチンのアルゴリズムを実装している。レベル 2 BLAS を呼び出す。[?gbtrf](#) も参照のこと。

### 入力パラメータ

<i>m</i>	INTEGER。行列 <i>A</i> の行数 ( $m \geq 0$ )。
<i>n</i>	INTEGER。 <i>A</i> の列の数 ( $n \geq 0$ )。

`kl` INTEGER。  $A$  の帯内の劣対角成分の数 ( $kl \geq 0$ )。  
`ku` INTEGER。  $A$  の帯内の優対角成分の数 ( $ku \geq 0$ )。  
`ab` REAL (sgbtf2 の場合 )  
 DOUBLE PRECISION (dgbtf2 の場合 )  
 COMPLEX (cgbtf2 の場合 )  
 COMPLEX\*16 (zgbtf2 の場合 )  
 配列、次元は (`ldab`, \*)。  
 配列 `ab` には、行列  $A$  が帯形式で格納される ([行列引数](#)を参照)。  
`ab` の第 2 次元は  $\max(1, n)$  以上でなければならない。  
  
`ldab` INTEGER。 配列 `ab` の第 1 次元。  
 ( $ldab \geq 2kl + ku + 1$ )

### 出力パラメータ

`ab` 因子分解の詳細で上書きされる。 $U$  の対角成分と  $kl + ku$  個の優対角成分は、`ab` の最初の  $1 + kl + ku$  行に格納される。因子分解に使用した乗数は、次の  $kl$  行に格納される。  
  
`ipiv` INTEGER。  
 配列、次元は  $\max(1, \min(m, n))$  以上。  
 ピボットのインデックス: 行  $i$  は行  $ipiv(i)$  と交換される。  
  
`info` INTEGER。 `info = 0` の場合、正常に終了したことを示す。  
`info = -i` の場合、 $i$  番目のパラメータの値が不正であったことを示す。  
`info = i` の場合、 $u_{ii}$  は 0 である。因子分解は完了したが  $U$  は完全に特異である。連立 1 次方程式の解の算出に係数  $U$  を使用すると、0 による除算が発生する。

## ?gebd2

非ブロック化アルゴリズムを使って一般行列を二重対角形式に縮退させる。

### 構文

```

call sgebd2( m, n, a, lda, d, e, tauq, taup, work, info )
call dgebd2( m, n, a, lda, d, e, tauq, taup, work, info )
call cgebd2( m, n, a, lda, d, e, tauq, taup, work, info )

```

```
call zgebd2( m, n, a, lda, d, e, tauq, taup, work, info )
```

## 説明

このルーチンは直交 / ユニタリ変換を用いて、 $m \times n$  の一般行列  $A$  を上または下二重対角形式  $B$  に縮退させる。 $Q' A P = B$

$m \geq n$  の場合、 $B$  は上二重対角である。 $m < n$  の場合、 $B$  は下二重対角である。

このルーチンでは、行列  $Q$  と  $P$  を明示的な形式では表現せず、基本リフレクタの積として表現する。 $m \geq n$  の場合、

$$Q = H(1)H(2)...H(n) \quad \text{および} \quad P = G(1)G(2)...G(n-1)$$

$m < n$  の場合、

$$Q = H(1)H(2)...H(m-1) \quad \text{および} \quad P = G(1)G(2)...G(m)$$

それぞれの  $H(i)$  と  $G(i)$  は次のような形式を持つ。

$$H(i) = I - \tau_{i,q} v v' \quad \text{および} \quad G(i) = I - \tau_{i,p} u u'$$

$\tau_{i,q}$  と  $\tau_{i,p}$  はスカラー (sgebd2/dgebd2 では実数、cgebd2/zgebd2 では複素)、 $v$  と  $u$  はベクトルである (sgebd2/dgebd2 では実数、cgebd2/zgebd2 では複素)。

## 入力パラメータ

$m$	INTEGER。行列 $A$ の行の数 ( $m \geq 0$ )。
$n$	INTEGER。 $A$ の列数 ( $n \geq 0$ )。
$a, work$	REAL (sgebd2 の場合 ) DOUBLE PRECISION (dgebd2 の場合 ) COMPLEX (cgebd2 の場合 ) COMPLEX*16 (zgebd2 の場合 )  配列 : $a(lda, *)$ には縮退する $m \times n$ の一般行列 $A$ を格納する。 $a$ の第 2 次元は、 $\max(1, n)$ 以上でなければならない。  $work(*)$ はワークスペース配列、 $work$ の値は $\max(1, m, n)$ 以上でなければならない。
$lda$	INTEGER。 $a$ の第 1 次元。 $\max(1, m)$ 以上であること。

## 出力パラメータ

- a**  $m \geq n$  の場合、**a** の対角および第 1 の優対角成分は上二重対角行列 **B** によって上書きされる。対角よりも下の成分は、配列 **tauq** とともに、基本リフレクタの積として直交/ユニタリの行列 **Q** を表現する。また、第 1 の優対角よりも上の成分は、配列 **taup** とともに、基本リフレクタの積として直交/ユニタリの行列 **P** を表現する。
- $m < n$  の場合、**a** の対角線と第 1 の劣対角成分が、下二重対角行列 **B** によって上書きされる。第 1 の劣対角よりも下の成分は、配列 **tauq** とともに、基本リフレクタの積として直交/ユニタリの行列 **Q** を表現する。また、対角よりも上の成分は、配列 **taup** とともに、基本リフレクタの積として直交/ユニタリの行列 **P** を表現する。
- d** REAL ( 単精度の場合 )  
DOUBLE PRECISION ( 倍精度の場合 ) 配列、次元は  $\max(1, \min(m, n))$  以上。  
二重対角行列 **B** の対角成分 ( $d(i) = a(i, i)$ ) が格納される。
- e** REAL ( 単精度の場合 )  
DOUBLE PRECISION ( 倍精度の場合 ) 配列、次元は  $\max(1, \min(m, n) - 1)$  以上。  
二重対角行列 **B** の対角外の成分が次のように格納される。
- $m \geq n$  のとき、 $i = 1, 2, \dots, n-1$  に対して  $e(i) = a(i, i+1)$ 、  
 $m < n$  のとき、 $i = 1, 2, \dots, m-1$  に対して  $e(i) = a(i+1, i)$
- tauq, taup** REAL (sgebd2 の場合 )  
DOUBLE PRECISION (dgebd2 の場合 )  
COMPLEX (cgebd2 の場合 )  
COMPLEX\*16 (zgebd2 の場合 )  
配列、次元は  $(1, \min(m, n))$  以上。  
直交/ユニタリ行列 **Q** と **P** をそれぞれ表す基本リフレクタのスカラ係数が格納される。
- info** INTEGER。  
 $info = 0$  の場合、正常に終了したことを示す。  
 $info = -i$  の場合、 $i$  番目のパラメータの値が不正であったことを示す。

## ?gehd2

非ブロック化アルゴリズムを使用して一般正方行列を上 Hessenberg 形式に縮退させる。

### 構文

```
call sgehd2( n, ilo, ihi, a, lda, tau, work, info )
call dgehd2( n, ilo, ihi, a, lda, tau, work, info )
call cgehd2( n, ilo, ihi, a, lda, tau, work, info )
call zgehd2( n, ilo, ihi, a, lda, tau, work, info )
```

### 説明

このルーチンは、直交またはユニタリの相似変換  $Q'AQ = H$  によって、実数 / 複素一般行列  $A$  を上 Hessenberg 形式  $H$  に縮退させる。

このルーチンでは、行列  $Q$  を明示的な形式では表現しない。代わりに、 $Q$  は基本リフレクタの積として表現される。

### 入力パラメータ

<i>n</i>	INTEGER。行列 $A$ の次数 ( $n \geq 0$ )。
<i>ilo, ihi</i>	INTEGER。 $A$ はすでに行 $1:ilo-1$ と列 $ihi+1:n$ で上三角になっていると仮定する。 $A$ が ?gebal から出力された場合、 <i>ilo</i> と <i>ihi</i> に、そのルーチンから返された値を設定しなければならない。それ以外の場合は $ilo = 1$ かつ $ihi = n$ を設定する。制約: $1 \leq ilo \leq ihi \leq \max(1, n)$
<i>a, work</i>	REAL (sgehd2 の場合) DOUBLE PRECISION (dgehd2 の場合) COMPLEX (cgehd2 の場合) COMPLEX*16 (zgehd2 の場合) 配列: $a(lda, *)$ には縮退させる $n \times n$ の行列 $A$ を格納する。 $a$ の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $work(n)$ は、ワークスペース配列。
<i>lda</i>	INTEGER。 $a$ の第 1 次元。 $\max(1, n)$ 以上でなければならない。

## 出力パラメータ

<i>a</i>	<i>A</i> の上三角と第 1 の劣対角は、上 Hessenberg 行列 <i>H</i> および第 1 の劣対角よりも下の成分で上書きされ、配列 <i>tau</i> とともに、基本リフレクタの積として直交 / ユニタリの行列 <i>Q</i> を表現する。次のアプリケーション・ノートを参照。
<i>tau</i>	REAL (sgehd2 の場合) DOUBLE PRECISION (dgehd2 の場合) COMPLEX (cgehd2 の場合) COMPLEX*16 (zgehd2 の場合) 配列、次元は $\max(1, n-1)$ 以上。 基本リフレクタのスカラー係数が格納される。次のアプリケーション・ノートを参照。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正であったことを示す。

## アプリケーション・ノート

?gehrd は、行列 *Q* を (*ihi-ilo*) 個の基本リフレクタの積として表現する。

$$Q = H(ilo) H(ilo+1) \dots H(ihi-1)$$

それぞれの *H*(*i*) は次のような形式を持つ。

$$H(i) = I - \tau * v * v'$$

*tau* は実数 / 複素スカラー、*v* は実数 / 複素ベクトルで、  
 $v(1:i) = 0$ ,  $v(i+1) = 1$  および  $v(ihi+1:n) = 0$  である。

ルーチン終了時には、 $v(i+2:ihi)$  は  $a(i+2:ihi, i)$  に格納され、*tau* は *tau*(*i*) に格納される。

$n=7$ 、 $ilo=2$ 、 $ihi=6$  の場合、*a* の内容を次の例に示す。

入力

$$\begin{bmatrix} a & a & a & a & a & a & a \\ & a & a & a & a & a & a \\ & & a & a & a & a & a \\ & & & a & a & a & a \\ & & & & a & a & a \\ & & & & & a & a \\ & & & & & & a \end{bmatrix}$$

出力

$$\begin{bmatrix} a & a & h & h & h & h & a \\ & a & h & h & h & h & a \\ & & h & h & h & h & h \\ & & & v_2 & h & h & h \\ & & & & v_2 & v_3 & h \\ & & & & & v_2 & v_3 & v_4 \\ & & & & & & & h \\ & & & & & & & & a \end{bmatrix}$$

ここで  $a$  は元の行列  $A$  の成分を表し、 $h$  は上 Hessenberg 行列  $H$  の更新された成分を表し、 $v_i$  は  $H(i)$  を定義するベクトルの成分を表している。

## ?sgelq2

非ブロック化アルゴリズムを使用して一般矩形行列の  $LQ$  因子分解を計算する。

### 構文

```
call sgelq2( m, n, a, lda, tau, work, info )
call dgelq2( m, n, a, lda, tau, work, info )
call cgelq2( m, n, a, lda, tau, work, info )
call zgelq2( m, n, a, lda, tau, work, info )
```

### 説明

このルーチンは実数 / 複素の  $m \times n$  行列  $A$  の  $LQ$  因子分解を  $A = LQ$  として計算する。

このルーチンでは、行列  $Q$  を明示的な形式では表現しない。代わりに、 $Q$  は  $\min(m, n)$  の基本リフレクタの積として表現される。

$Q = H(k) \dots H(2) H(1)$  ( または複素の場合、 $Q = H(k)' \dots H(2)' H(1)'$  )、ここで  $k = \min(m, n)$

それぞれの  $H(i)$  は次のような形式を持つ。

$$H(i) = I - \tau v v'$$



$\tau$  は  $\tau(i)$  に格納される実数 / 複素スカラー、 $v$  は  $v(1:i-1) = 0$  および  $v(i) = 1$  を持つ実数 / 複素ベクトルである。

ルーチン終了時に  $v(i+1:n)$  は  $a(i, i+1:n)$  に格納される。

### 入力パラメータ

$m$  INTEGER。行列  $A$  の行の数 ( $m \geq 0$ )。

$n$  INTEGER。  $A$  の列数 ( $n \geq 0$ )。

$a, work$  REAL (sgelq2 の場合)  
DOUBLE PRECISION (dgelq2 の場合)  
COMPLEX (cgelq2 の場合)  
COMPLEX\*16 (zgelq2 の場合)  
配列:  
 $a(lda, *)$  には、 $m \times n$  の行列  $A$  を格納する。  
 $a$  の第 2 次元は、 $\max(1, n)$  以上でなければならない。  
 $work(m)$  は、ワークスペース配列。

$lda$  INTEGER。  $a$  の第 1 次元。  $\max(1, m)$  以上であること。

### 出力パラメータ

$a$  因子分解のデータによって、次のように上書きされる。  
  
配列  $a$  の対角成分とその下の成分は  $m \times \min(n, m)$  の下台形行列  $L$  ( $n \geq m$  の場合、 $L$  は下三角行列) で上書きされる。対角よりも上の成分は、配列  $\tau$  とともに、 $\min(n, m)$  の基本リフレクタ積として直交 / ユニタリの行列  $Q$  を表現する。

$\tau$  REAL (sgelq2 の場合)  
DOUBLE PRECISION (dgelq2 の場合)  
COMPLEX (cgelq2 の場合)  
COMPLEX\*16 (zgelq2 の場合)  
配列、次元は  $\max(1, \min(m, n))$  以上。  
基本リフレクタのスカラー係数が入る。

$info$  INTEGER。  
 $info = 0$  の場合、正常に終了したことを示す。  
 $info = -i$  の場合、 $i$  番目のパラメータの値が不正であったことを示す。

## ?geql2

非ブロック化アルゴリズムを使用して一般矩形行列の  $QL$  因子分解を計算する。

### 構文

```
call sgeql2( m, n, a, lda, tau, work, info )
call dgeql2( m, n, a, lda, tau, work, info )
call cgeql2( m, n, a, lda, tau, work, info )
call zgeql2( m, n, a, lda, tau, work, info )
```

### 説明

このルーチンは実数 / 複素の  $m \times n$  行列  $A$  の  $QL$  因子分解を  $A = QL$  として計算する。

このルーチンでは、行列  $Q$  を明示的な形式では表現しない。代わりに、 $Q$  は  $\min(m, n)$  の基本リフレクタの積として表現される。

$Q = H(k) \dots H(2) H(1)$ 、ここで  $k = \min(m, n)$

それぞれの  $H(i)$  は次のような形式を持つ。

$$H(i) = I - \tau v v'$$

$\tau$  は  $\tau(i)$  に格納される実数 / 複素スカラー、 $v$  は  $v(m-k+i+1:m) = 0$  および  $v(m-k+i) = 1$  を持つ実数 / 複素ベクトルである。

ルーチン終了時には  $v(1:m-k+i-1)$  は  $a(1:m-k+i-1, n-k+i)$  に格納される。

### 入力パラメータ

$m$	INTEGER。行列 $A$ の行の数 ( $m \geq 0$ )。
$n$	INTEGER。 $A$ の列数 ( $n \geq 0$ )。
$a, work$	REAL (sgeql2 の場合 ) DOUBLE PRECISION (dgeql2 の場合 ) COMPLEX (cgeql2 の場合 ) COMPLEX*16 (zgeql2 の場合 ) 配列: $a(lda, *)$ には、 $m \times n$ の行列 $A$ を格納する。 $a$ の第 2 次元は、 $\max(1, n)$ 以上でなければならない。

$work(m)$  は、ワークスペース配列。

$lda$  INTEGER。  $a$  の第 1 次元。  $\max(1, m)$  以上であること。

### 出力パラメータ

$a$  因子分解のデータによって、次のように上書きされる。  
 $m \geq n$  の場合は、部分配列  $a(m-n+1:m, 1:n)$  の下三角部分に  $n \times n$  の下三角行列  $L$  が格納される。  
 $m < n$  の場合は、 $(n-m)$  番目の優対角成分とその下の各成分に  $m \times n$  の下台形行列  $L$  が格納される。残りの成分は、配列  $tau$  とともに、基本リフレクタの積として直交 / ユニタリ行列  $Q$  を表現する。

$tau$  REAL (sgeql2 の場合)  
 DOUBLE PRECISION (dgeql2 の場合)  
 COMPLEX (cgeql2 の場合)  
 COMPLEX\*16 (zgeql2 の場合)  
 配列、次元は  $\max(1, \min(m, n))$  以上。  
 基本リフレクタのスカラー係数が入る。

$info$  INTEGER。  
 $info = 0$  の場合、正常に終了したことを示す。  
 $info = -i$  の場合、 $i$  番目のパラメータの値が不正であったことを示す。

## ?geqr2

非ブロックアルゴリズムを使用して一般  
 矩形行列の  $QR$  因子分解を計算する。

### 構文

```
call sgeqr2( m, n, a, lda, tau, work, info )
call dgeqr2( m, n, a, lda, tau, work, info )
call cgeqr2( m, n, a, lda, tau, work, info )
call zgeqr2( m, n, a, lda, tau, work, info )
```

### 説明

このルーチンは実数 / 複素の  $m \times n$  行列  $A$  の  $QR$  因子分解を  $A = QR$  として計算する。

このルーチンでは、行列  $Q$  を明示的な形式では表現しない。代わりに、 $Q$  は  $\min(m, n)$  の基本リフレクタの積として表現される。

$Q = H(1)H(2) \dots H(k)$ 、ここで  $k = \min(m, n)$

それぞれの  $H(i)$  は次のような形式を持つ。

$$H(i) = I - \tau v v'$$

$\tau$  は  $\tau(i)$  に格納される実数 / 複素スカラ、 $v$  は  $v(1:i-1) = 0$  および  $v(i) = 1$  を持つ実数 / 複素ベクトルである。

ルーチン終了時に  $v(i+1:m)$  は  $a(i+1:m, i)$  に格納される。

## 入力パラメータ

$m$	INTEGER。行列 $A$ の行の数 ( $m \geq 0$ )。
$n$	INTEGER。 $A$ の列数 ( $n \geq 0$ )。
$a, work$	REAL (sgeqr2 の場合 ) DOUBLE PRECISION (dgeqr2 の場合 ) COMPLEX (cgeqr2 の場合 ) COMPLEX*16 (zgeqr2 の場合 ) 配列 : $a(lda, *)$ には、 $m \times n$ の行列 $A$ を格納する。 $a$ の第 2 次元は、 $\max(1, n)$ 以上でなければならない。  $work(n)$ は、ワークスペース配列。
$lda$	INTEGER。 $a$ の第 1 次元。 $\max(1, m)$ 以上であること。

## 出力パラメータ

$a$	因子分解のデータによって、次のように上書きされる。  行列の対角成分とその上の成分は、 $\min(n, m) \times n$ の上台形行列 $R$ によって上書きされる ( $m \geq n$ のとき $R$ は上三角行列)。対角よりも下の成分は、配列 $\tau$ とともに、基本リフレクタの積として直交 / ユニタリ行列 $Q$ を表現する。
$\tau$	REAL (sgeqr2 の場合 ) DOUBLE PRECISION (dgeqr2 の場合 ) COMPLEX (cgeqr2 の場合 ) COMPLEX*16 (zgeqr2 の場合 ) 配列、次元は $\max(1, \min(m, n))$ 以上。 基本リフレクタのスカラ係数が入る。

*info* INTEGER。  
*info* = 0 の場合、正常に終了したことを示す。  
*info* = -*i* の場合、*i* 番目のパラメータの値が不正であったことを示す。

## ?gerq2

非ブロック化アルゴリズムを使用して一般矩形行列の *RQ* 因子分解を計算する。

### 構文

```
call sgerq2( m, n, a, lda, tau, work, info )
call dgerq2( m, n, a, lda, tau, work, info )
call cgerq2( m, n, a, lda, tau, work, info )
call zgerq2( m, n, a, lda, tau, work, info )
```

### 説明

このルーチンは実数 / 複素の  $m \times n$  行列  $A$  の *RQ* 因子分解を  $A = QR$  として計算する。

このルーチンでは、行列  $Q$  を明示的な形式では表現しない。代わりに、 $Q$  は  $\min(m, n)$  の基本リフレクタの積として表現される。

$Q = H(1)H(2) \dots H(k)$ 、ここで  $k = \min(m, n)$

それぞれの  $H(i)$  は次のような形式を持つ。

$$H(i) = I - \tau v v'$$

$\tau$  は  $\tau(i)$  に格納される実数 / 複素スカラ、 $v$  は  $v(n-k+i+1:n) = 0$  および  $v(n-k+i) = 1$  を持つ実数 / 複素ベクトルである。

ルーチン終了時に  $v(1:n-k+i-1)$  は  $a(m-k+i, 1:n-k+i-1)$  に格納される。

### 入力パラメータ

*m* INTEGER。行列  $A$  の行の数 ( $m \geq 0$ )。  
*n* INTEGER。  $A$  の列数 ( $n \geq 0$ )。

*a, work*            REAL (sgerq2 の場合 )  
                       DOUBLE PRECISION (dgerq2 の場合 )  
                       COMPLEX (cgerq2 の場合 )  
                       COMPLEX\*16 (zgerq2 の場合 )  
                       配列:  
                       *a*(*lda*, \*) には、 $m \times n$  の行列 *A* を格納する。  
                       *a* の第 2 次元は、 $\max(1, n)$  以上でなければならない。  
                       *work*(*m*) は、ワークスペース配列。

*lda*                 INTEGER。 *a* の第 1 次元。  $\max(1, m)$  以上であること。

## 出力パラメータ

*a*                    因子分解のデータによって、次のように上書きされる。

$m \leq n$  の場合は、部分配列 *a*(1:*m*, *n*-*m*+1:*n*) の上三角部分に、 $m \times m$  の上三角行列 *R* が格納される。

$m > n$  の場合は、(*m*-*n*) 番目の劣対角成分とその上の各成分に、 $m \times n$  の上台形行列 *R* が格納される。残りの成分は、配列 *tau* とともに、 $\min(m, n)$  個の基本リフレクタの積として直交/ユニタリ行列 *Q* を表現する。

*tau*                 REAL (sgerq2 の場合 )  
                       DOUBLE PRECISION (dgerq2 の場合 )  
                       COMPLEX (cgerq2 の場合 )  
                       COMPLEX\*16 (zgerq2 の場合 )  
                       配列、次元は  $\max(1, \min(m, n))$  以上。  
                       基本リフレクタのスカラ係数が入る。

*info*                INTEGER。  
                       *info* = 0 の場合、正常に終了したことを示す。  
                       *info* = -*i* の場合、*i* 番目のパラメータの値が不正であったことを示す。

## ?gesc2

?getc2 で計算された完全ピボット演算による  
 $LU$  因子分解を使用して連立 1 次方程式を解く。

### 構文

```
call sgetc2( n, a, lda, rhs, ipiv, jpiv, scale )
call dgetc2( n, a, lda, rhs, ipiv, jpiv, scale )
call cgetc2( n, a, lda, rhs, ipiv, jpiv, scale )
call zgetc2( n, a, lda, rhs, ipiv, jpiv, scale )
```

### 説明

このルーチンは、次の連立 1 次方程式を解く。

$$AX = scale * RHS$$

[?getc2](#) で計算された完全ピボット演算による  $LU$  因子分解を使用して、一般  $n \times n$  行列  $A$  について解く。

### 入力パラメータ

$n$	INTEGER。行列 $A$ の次数。
$a, rhs$	REAL (sgesc2 の場合) DOUBLE PRECISION (dgetc2 の場合) COMPLEX (cgetc2 の場合) COMPLEX*16 (zgetc2 の場合) 配列: $a(lda, *)$ には ?getc2 で計算された $n \times n$ 行列 $A$ の因子分解の $LU$ 部分を格納する。 $A = PLUQ$ $a$ の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $rhs(n)$ には連立 1 次方程式の右辺ベクトルを格納する。
$lda$	INTEGER。 $a$ の第 1 次元。 $\max(1, n)$ 以上でなければならない。

<i>ipiv</i>	INTEGER。 配列、次元は $\max(1, n)$ 以上。 ピボットのインデックス。 $1 \leq i \leq n$ で、行列の行 $i$ は行 $ipiv(i)$ と交換されている。
<i>jpiv</i>	INTEGER 配列、次元は $\max(1, n)$ 以上。 ピボットのインデックス。 $1 \leq j \leq n$ で、行列の列 $j$ は列 $jpiv(j)$ と交換されている。

### 出力パラメータ

<i>rhs</i>	解のベクトル $X$ で上書きされる。
<i>scale</i>	REAL ( <i>sgesc2/cgesc2</i> の場合 ) DOUBLE PRECISION ( <i>dgesc2/zgesc2</i> の場合 ) スケール係数が格納される。 <i>scale</i> は解におけるオーバーフローを防ぐため $0 \leq scale \leq 1$ の範囲で選択される。

---

## ?getc2

一般  $n \times n$  行列の完全ピボット演算による  $LU$  因子分解を計算する。

---

### 構文

```
call sgetc2( n, a, lda, ipiv, jpiv, info )
call dgetc2( n, a, lda, ipiv, jpiv, info )
call cgetc2( n, a, lda, ipiv, jpiv, info )
call zgetc2( n, a, lda, ipiv, jpiv, info )
```

### 説明

このルーチンは一般  $n \times n$  行列  $A$  の完全ピボット演算による  $LU$  因子分解を計算する。因子分解は形式  $A = P * L * U * Q$  を持ち、 $P$  と  $Q$  は置換行列、 $L$  は単位対角成分を持つ下三角、 $U$  は上三角である。

このルーチンで計算される  $LU$  因子分解は、[?latdf](#) より Dif 推定値の逆数に対する影響を計算するのに用いられる。



## 入力パラメータ

<i>n</i>	INTEGER。行列 <i>A</i> の次数 ( $n \geq 0$ )。
<i>a</i>	REAL (sgetc2 の場合 ) DOUBLE PRECISION (dgetc2 の場合 ) COMPLEX (cgetc2 の場合 ) COMPLEX*16 (zgetc2 の場合 ) 配列 <i>a</i> ( <i>lda</i> , *) には因子分解する $n \times n$ 行列 <i>A</i> を格納する。 <i>a</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $\max(1, n)$ 以上でなければならない。

## 出力パラメータ

<i>a</i>	因子分解 $A = P * L * U * Q$ で得られた係数 <i>L</i> と <i>U</i> で上書きされる。 <i>L</i> の単位対角成分は格納されない。 <i>U</i> ( <i>k</i> , <i>k</i> ) が <i>smin</i> 未満の場合、 <i>U</i> ( <i>k</i> , <i>k</i> ) が <i>smin</i> の値として与えられる。すなわち、非特異の摂動連立方程式を与える。
<i>ipiv</i>	INTEGER。 配列、次元は $\max(1, n)$ 以上。 ピボットのインデックス。 $1 \leq i \leq n$ で、行列の行 <i>i</i> は行 <i>ipiv</i> ( <i>i</i> ) と交換されている。
<i>jpiv</i>	INTEGER 配列、次元は $\max(1, n)$ 以上。 ピボットのインデックス。 $1 \leq j \leq n$ で、行列の列 <i>j</i> は列 <i>jpiv</i> ( <i>j</i> ) と交換されている。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = <i>k</i> > 0 の場合、 $Ax = b$ で <i>x</i> を解こうとすると <i>U</i> ( <i>k</i> , <i>k</i> ) はおそらくオーバーフローを生む。そこで、オーバーフローを防止するために <i>U</i> は摂動となる。

### ?getf2

一般  $m \times n$  行列の  $LU$  因子分解を、行交換を伴う部分ピボット演算を用いて計算する (非ブロック化アルゴリズム)。

---

#### 構文

```
call sgetf2( m, n, a, lda, ipiv, info )
call dgetf2( m, n, a, lda, ipiv, info )
call cgetf2( m, n, a, lda, ipiv, info )
call zgetf2( m, n, a, lda, ipiv, info )
```

#### 説明

このルーチンは一般  $m \times n$  行列  $A$  の  $LU$  因子分解を、行交換を伴う部分ピボット演算を用いて計算する。因子分解は次の形式を持つ。

$$A = PLU,$$

$P$  は置換行列、 $L$  は単位対角成分を含む下三角 ( $m > n$  の場合は下台形)、 $U$  は上三角 ( $m < n$  の場合は上台形) である。

#### 入力パラメータ

$m$	INTEGER。行列 $A$ の行の数 ( $m \geq 0$ )。
$n$	INTEGER。 $A$ の列数 ( $n \geq 0$ )。
$a$	REAL (sgetf2 の場合) DOUBLE PRECISION (dgetf2 の場合) COMPLEX (cgetf2 の場合) COMPLEX*16 (zgetf2 の場合) 配列、次元は ( $lda, *$ )。因子分解の対象となる行列 $A$ が格納される。 $a$ の第 2 次元は、 $\max(1, n)$ 以上でなければならない。
$lda$	INTEGER。 $a$ の第 1 次元。 $\max(1, m)$ 以上であること。

#### 出力パラメータ

$a$	$L$ と $U$ によって上書きされる。 $L$ の単位対角成分は格納されない。
-----	---

<i>ipiv</i>	INTEGER。 配列、次元は $\max(1, \min(m, n))$ 以上。 ピボットのインデックス。 $1 \leq i \leq n$ で、行列の行 $i$ は行 $ipiv(i)$ と交換されている。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了した。 <i>info</i> = - $i$ の場合、 $i$ 番目のパラメータの値が不正であったことを示す。 <i>info</i> = $i > 0$ の場合、 $u_{ii}$ はゼロである。因子分解は完了したが $U$ は完全に特異である。連立 1 次方程式の解の算出に係数 $U$ を使用すると、0 による除算が発生する。

## ?gtts2

?gttrf によって行われた LU 因子分解を使用して、三重対角行列を係数行列とする連立 1 次方程式を解く。

### 構文

```
call sgtts2( itrans, n, nrhs, dl, d, du, du2, ipiv, b, ldb )
call dgtts2( itrans, n, nrhs, dl, d, du, du2, ipiv, b, ldb )
call cgtts2( itrans, n, nrhs, dl, d, du, du2, ipiv, b, ldb )
call zgtts2( itrans, n, nrhs, dl, d, du, du2, ipiv, b, ldb )
```

### 説明

このルーチンは、以下に示す複数の右辺を持つ以下の連立 1 次方程式のうち 1 つを  $X$  について解く。

$AX=B$      $A^T X=B$     または     $A^H X=B$  ( 複素行列のみ )、  
 $A$  は三重対角行列で [?gttrf](#) で計算した LU 因子分解を使用する。

### 入力パラメータ

*itrans*    INTEGER。ゼロ、1、または 2 のいずれかでなければならない。  
 方程式の形式を指定する。  
*itrans* = 0 の場合、 $AX=B$  ( 転置なし )  
*itrans* = 1 の場合、 $A^T X=B$  ( 転置 )  
*itrans* = 2 の場合、 $A^H X=B$  ( 共役転置 )

<i>n</i>	INTEGER。行列 <i>A</i> の次数 ( $n \geq 0$ )。
<i>nrhs</i>	INTEGER。右辺の数、すなわち、 <i>B</i> の列の数 ( $nrhs \geq 0$ )。
<i>d1, d, du, du2, b</i>	REAL (sgtts2 の場合) DOUBLE PRECISION (dgtts2 の場合) COMPLEX (cgtts2 の場合) COMPLEX*16 (zgtts2 の場合) 配列: <i>d1</i> ( $n-1$ ), <i>d</i> ( $n$ ), <i>du</i> ( $n-1$ ), <i>du2</i> ( $n-2$ ), <i>b</i> ( <i>ldb</i> , <i>nrhs</i> ) 配列 <i>d1</i> には、 <i>A</i> の LU 因子分解で得られた行列 <i>L</i> を定義する ( $n-1$ ) 個の乗数が格納される。 配列 <i>d</i> には、 <i>A</i> の LU 因子分解で得られた上三角行列 <i>U</i> の $n$ 個の対角成分が格納される。 配列 <i>du</i> には、 <i>U</i> の最初の優対角成分の ( $n-1$ ) 個の成分が格納される。 配列 <i>du2</i> には、 <i>U</i> の 2 番目の優対角成分の ( $n-2$ ) 個の成分が格納される。 配列 <i>b</i> には、行列 <i>B</i> が格納される。この行列の列は、連立方程式の右辺である。
<i>ldb</i>	INTEGER。配列 <i>b</i> のリーディング・ディメンジョン。 $ldb \geq \max(1, n)$
<i>ipiv</i>	INTEGER。 配列、次元は ( $n$ )。 <a href="#">?gttrf</a> によって返されるピボットのインデックス配列。

## 出力パラメータ

<i>b</i>	解の行列 <i>X</i> によって上書きされる。
----------	---------------------------

---

## ?labrd

一般行列の最初の *nb* 行と列を二重対角形式に縮退させる。

---

### 構文

```
call slabrd( m, n, nb, a, lda, d, e, tauq, taup, x, ldx, y, ldy )
call dlabrd( m, n, nb, a, lda, d, e, tauq, taup, x, ldx, y, ldy )
call clabrd( m, n, nb, a, lda, d, e, tauq, taup, x, ldx, y, ldy )
call zlabrd( m, n, nb, a, lda, d, e, tauq, taup, x, ldx, y, ldy )
```

## 説明

このルーチンは、直交/ユニタリ変換  $Q'AP$  によって一般  $m \times n$  行列  $A$  の最初の  $nb$  行と列を上または下二重対角形式に縮退させ、 $A$  の縮退されていない部分に変換を適用するために必要な行列  $X$  と  $Y$  を返す。

$m \geq n$  の場合、 $A$  は上二重対角形式に縮退され、 $m < n$  の場合、下二重対角形式に縮退される。

行列  $Q$  と  $P$  は基本リフレクタの積として表現される。

$Q = H(1) H(2) \dots H(nb)$  および  $P = G(1) G(2) \dots G(nb)$

それぞれの  $H(i)$  と  $G(i)$  は次のような形式を持つ。

$H(i) = I - \tau_{au} v v'$  および  $G(i) = I - \tau_{ap} u u'$

$\tau_{au}$  と  $\tau_{ap}$  はスカラ、 $v$  と  $u$  はベクトルである。

ベクトル  $v$  と  $u$  の成分は  $m \times nb$  の行列  $V$  と  $nb \times n$  の行列  $U'$  を形成する。行列  $A$  の縮退されていない部分に次の形式のブロック更新を使って変換を適用するために、これらの行列と行列  $X$  および  $Y$  が必要となる。 $A := A - V * Y' - X * U'$

このルーチンは [?gebrd](#) から呼び出される補助ルーチンである。

## 入力パラメータ

$m$	INTEGER。行列 $A$ の行の数 ( $m \geq 0$ )。
$n$	INTEGER。 $A$ の列数 ( $n \geq 0$ )。
$nb$	INTEGER。縮退の対象となる $A$ の先頭の行と列の数。
$a$	REAL (slabrd の場合 ) DOUBLE PRECISION (dlabrd の場合 ) COMPLEX (clabrd の場合 ) COMPLEX*16 (zlabrd の場合 )  配列 $a(lda, *)$ には縮退される行列 $A$ を格納する。 $a$ の第 2 次元は、 $\max(1, n)$ 以上でなければならない。
$lda$	INTEGER。 $a$ の第 1 次元。 $\max(1, m)$ 以上であること。
$ldx$	INTEGER。出力配列 $x$ の第 1 次元のサイズ。 $\max(1, m)$ 以上でなければならない。
$ldy$	INTEGER。出力配列 $y$ の第 1 次元のサイズ。 $\max(1, n)$ 以上でなければならない。

## 出力パラメータ

<i>a</i>	<p>行列の最初の <i>nb</i> 行と列は上書きされる。配列の残りの部分は変更されない。</p> <p><math>m \geq n</math> の場合、最初の <i>nb</i> 列の対角線上とその下の成分は、配列 <i>taug</i> とともに、基本リフレクタの積として直交 / ユニタリ行列 <i>Q</i> を表現する。また、最初の <i>nb</i> 行の対角よりも上の成分は、配列 <i>taup</i> とともに、基本リフレクタの積として直交 / ユニタリ行列 <i>P</i> を表現する。</p> <p><math>m &lt; n</math> の場合、最初の <i>nb</i> 列の対角よりも下の成分は、配列 <i>taug</i> とともに、基本リフレクタの積として直交 / ユニタリ行列 <i>Q</i> を表現する。また、最初の <i>nb</i> 行の対角線上とその上の成分は、配列 <i>taup</i> とともに、基本リフレクタの積として直交 / ユニタリ行列 <i>P</i> を表現する。</p>
<i>d</i> , <i>e</i>	<p>REAL ( 単精度の場合 )</p> <p>DOUBLE PRECISION ( 倍精度の場合 ) 配列、次元はそれぞれ (<i>nb</i>)。</p> <p>配列 <i>d</i> には縮退された行列の最初の <i>nb</i> 行と列の対角成分が格納される。</p> <p><math>d(i) = a(i, i)</math></p> <p>配列 <i>e</i> には縮退された行列の最初の <i>nb</i> 行と列の対角以外の成分が格納される。</p>
<i>taug</i> , <i>taup</i>	<p>REAL (slabrd の場合 )</p> <p>DOUBLE PRECISION (dlabrd の場合 )</p> <p>COMPLEX (clabrd の場合 )</p> <p>COMPLEX*16 (zlabrd の場合 )</p> <p>配列、次元はそれぞれ (<i>nb</i>)。</p> <p>直交 / ユニタリ行列 <i>Q</i> と <i>P</i> をそれぞれ表す基本リフレクタのスカラ係数が格納される。</p>
<i>x</i> , <i>y</i>	<p>REAL (slabrd の場合 )</p> <p>DOUBLE PRECISION (dlabrd の場合 )</p> <p>COMPLEX (clabrd の場合 )</p> <p>COMPLEX*16 (zlabrd の場合 )</p> <p>配列、次元は <math>x(ldx, nb)</math>, <math>y(ldy, nb)</math>。</p> <p>配列 <i>x</i> には、<i>A</i> の縮退されていない部分を更新するために必要となる <math>m \times nb</math> の行列 <i>X</i> が格納される。</p> <p>配列 <i>y</i> には、<i>A</i> の縮退されていない部分を更新するために必要となる <math>n \times nb</math> の行列 <i>Y</i> が格納される。</p>

**アプリケーション・ノート**

$m \geq n$  の場合、基本リフレクタ  $H(i)$  と  $G(i)$  に対して、

$v(1:i-1) = 0$ 、 $v(i) = 1$ 、 $v(i:m)$  は  $a(i:m, i)$  に格納される。  
 $u(1:i) = 0$ 、 $u(i+1) = 1$ 、 $u(i+1:n)$  は  $a(i, i+1:n)$  に格納される。  
 $tauq$  は  $tauq(i)$  に、 $taup$  は  $taup(i)$  に格納される。

$m < n$  の場合、

$v(1:i) = 0$ 、 $v(i+1) = 1$ 、 $v(i+1:m)$  は  $a(i+2:m, i)$  に格納される。  
 $u(1:i-1) = 0$ 、 $u(i) = 1$ 、 $u(i:n)$  は  $a(i, i+1:n)$  に格納される。  
 $tauq$  は  $tauq(i)$  に、 $taup$  は  $taup(i)$  に格納される。

ルーチン終了後の  $a$  の内容は、 $nb = 2$  において次の例に示される。

$m = 6, n = 5 (m > n)$

$m = 5, n = 6 (m < n)$

$$\begin{bmatrix} 1 & 1 & u_1 & u_1 & u_1 \\ v_1 & 1 & 1 & u_2 & u_2 \\ v_1 & v_2 & a & a & a \\ v_1 & v_2 & a & a & a \\ v_1 & v_2 & a & a & a \\ v_1 & v_2 & a & a & a \end{bmatrix}$$

$$\begin{bmatrix} 1 & u_1 & u_1 & u_1 & u_1 & u_1 \\ 1 & 1 & u_2 & u_2 & u_2 & u_2 \\ v_1 & 1 & a & a & a & a \\ v_1 & v_2 & a & a & a & a \\ v_1 & v_2 & a & a & a & a \end{bmatrix}$$

$a$  は元の行列の変更されていない成分を意味し、 $v_i$  は  $H(i)$  を定義するベクトルの成分を意味し、 $u_i$  は  $G(i)$  を定義するベクトルの成分を意味する。

**?lacon**

行列-ベクトル積の評価にリバーズ・コミュニケーションを使用して正行列の1-ノルムを推定する。

**構文**

```
call slacon( n, v, x, isgn, est, kase, jmax, jump, iter )
```

```
call dlacon( n, v, x, isgn, est, kase, jmax, jump, iter )
call clacon( n, v, x, est, kase, jmax, jump, iter )
call zlacon( n, v, x, est, kase, jmax, jump, iter )
```

## 説明

このルーチンは実数 / 複素の正方行列  $A$  の 1- ノルムを推定する。行列 - ベクトル積の評価にはリバース・コミュニケーションが使用される。

## 入力パラメータ

<i>n</i>	INTEGER。行列 $A$ の次数 ( $n \geq 1$ )。
<i>v, x</i>	REAL (slacon の場合 ) DOUBLE PRECISION (dlacon の場合 ) COMPLEX (clacon の場合 ) COMPLEX*16 (zlacon の場合 )  配列、次元はそれぞれ、( $n$ )。 $v$ は、ワークスペース配列。 $x$ は中間戻り後は入力として使用される。
<i>isgn</i>	INTEGER。ワークスペース配列、次元は ( $n$ )、実数の場合のみ使用される。
<i>kase</i>	INTEGER。 $?lacon$ を初めて呼び出すときは <i>kase</i> はゼロでなければならない。
<i>jmax, jump, iter</i>	INTEGER。ワークスペース、初めての $?lacon$ ( $kase = 0$ ) の呼び出しのため内部データを保持する。変更してはいけない。

## 出力パラメータ

<i>est</i>	REAL (slacon/clacon の場合 ) DOUBLE PRECISION (dlacon/zlacon の場合 ) ノルム ( $A$ ) の推定 ( 下限 )。
<i>kase</i>	中間戻りでは、 <i>kase</i> は 1 か 2 となり、 $x$ が $A * x$ または $A' * x$ によって上書きされるかどうかを示す。 $?lacon$ からの最終戻りでは、 <i>kase</i> は再びゼロになる。
<i>v</i>	最終戻りでは $v = A * w$ 、ここで $est = \text{norm}(v) / \text{norm}(w)$ ( $w$ は返されない)。



$x$  中間戻りでは、 $x$  は次の値で上書きされる。  
 $A * x$ 、 $kase = 1$  の場合、  
 $A' * x$ 、 $kase = 2$  の場合、  
(複素では  $A'$  は  $A$  の共役転置)、また、他のパラメータは変更しない  
状態で `?lacon` を再度呼び出さなければならない。

## ?lacpy

1 つの 2 次元配列の一部または全部を他に  
コピーする。

### 構文

```
call slacpy( uplo, m, n, a, lda, b, ldb )
call dlacpy( uplo, m, n, a, lda, b, ldb )
call clacpy( uplo, m, n, a, lda, b, ldb )
call zlacpy( uplo, m, n, a, lda, b, ldb )
```

### 説明

このルーチンは 2 次元行列  $A$  の一部または全部を他の行列  $B$  にコピーする。

### 入力パラメータ

**uplo** CHARACTER\*1。  
行列  $A$  のうち、 $B$  にコピーする部分を指定する。  
 $uplo = 'U'$  の場合、 $A$  の上三角部分をコピーする。  
 $uplo = 'L'$  の場合、 $A$  の下三角部分をコピーする。  
それ以外の場合は、行列  $A$  の全部がコピーする。

**m** INTEGER。行列  $A$  の行の数 ( $m \geq 0$ )。

**n** INTEGER。  $A$  の列数 ( $n \geq 0$ )。

**a** REAL (slacpy の場合 )  
DOUBLE PRECISION (dlacpy の場合 )  
COMPLEX (clacpy の場合 )  
COMPLEX\*16 (zlacpy の場合 )  
配列  $a(lda, *)$  には  $m \times n$  の行列  $A$  を格納する。

$a$  の第 2 次元は  $\max(1, n)$  以上でなければならない。  
 $uplo = 'U'$  の場合、三角または台形の上側だけがアクセスされる。  
 $uplo = 'L'$  の場合、三角または台形の下側だけがアクセスされる。

$lda$  INTEGER。  $a$  の第 1 次元。  $lda \geq \max(1, m)$

$ldb$  INTEGER。 出力配列  $b$  の第 1 次元のサイズ。  $ldb \geq \max(1, m)$

### 出力パラメータ

$b$  REAL (slacpy の場合 )  
DOUBLE PRECISION (dlacpy の場合 )  
COMPLEX (clacpy の場合 )  
COMPLEX\*16 (zlacpy の場合 )  
配列  $b(ldb, *)$  には  $m \times n$  の行列  $B$  が格納される。  
 $b$  の第 2 次元は  $\max(1, n)$  以上でなければならない。  
 $uplo$  で指定された部分において  $B = A$  で上書きされる。

---

## ?ladiv

不必要なオーバーフローを回避して、  
実数演算で複素除算を実行する。

---

### 構文

```
call sladiv( a, b, c, d, p, q )  
call dladiv( a, b, c, d, p, q )  
res = cladiv( x, y )  
res = zladiv( x, y )
```

### 説明

ルーチン `sladiv/dladiv` は、次式として複素除算を実数演算で実行する。

$$p + iq = \frac{a + ib}{c + id}$$

複素関数 `cladiv/zladiv` は結果を次式として計算する。

$$res = x/y$$

$x$  と  $y$  は複素である。 $x/y$  の計算は、結果がオーバーフローしない限り中間過程でオーバーフローは生じない。

### 入力パラメータ

$a, b, c, d$       REAL (sladiv の場合)  
                   DOUBLE PRECISION (dladiv の場合)  
                   上の式にあるスカラ  $a, b, c, d$  (実数型のみ)

$x, y$             COMPLEX (cladiv の場合)  
                   COMPLEX\*16 (zladiv の場合)  
                   複素スカラ  $x$  と  $y$  (複素型のみ)

### 出力パラメータ

$p, q$             REAL (sladiv の場合)  
                   DOUBLE PRECISION (dladiv の場合)  
                   上の式にあるスカラ  $p$  と  $q$  (実数型のみ)

$res$             COMPLEX (cladiv の場合)  
                   DOUBLE COMPLEX (zladiv の場合)  
                    $x/y$  の除算の結果が格納される。

## ?lae2

$2 \times 2$  対称行列の固有値を計算する。

### 構文

```
call slae2( a, b, c, rt1, rt2 )
call dlae2( a, b, c, rt1, rt2 )
```

### 説明

ルーチン slae2/dlae2 は  $2 \times 2$  対称行列の固有値を計算する。

$$\begin{bmatrix} a & b \\ b & c \end{bmatrix}$$

出力において、*rt1* は大きい方の絶対値の固有値、*rt2* は小さい方の絶対値の固有値で上書きされる。

### 入力パラメータ

*a*, *b*, *c*            REAL(*slae2* の場合 )  
                      DOUBLE PRECISION(*dlae2* の場合 )  
                      上記  $2 \times 2$  行列の成分 *a*、*b*、*c*

### 出力パラメータ

*rt1*, *rt2*            REAL(*slae2* の場合 )  
                      DOUBLE PRECISION(*dlae2* の場合 )  
                      大きい方の絶対値、小さい方の絶対値に対してそれぞれ計算された固有値。

### アプリケーション・ノート

*rt1* はオーバーフロー/アンダーフローが起これなければ、わずかな *ulp* ( 最小位置単位 ) に対して正確である。*rt2* は行列式  $a*c-b*b$  で、大幅な桁落ちがある場合に不正確となる可能性がある。*rt2* を精度高く計算するには、あらゆる場合で高精度または適切な丸めまたは適切な切捨て計算が必要になる。

オーバーフローは、*rt1* がオーバーフローに係数 5 を掛けた値の範囲内にあるときに起こり得る。アンダーフローは入力データがゼロか *underflow\_threshold* / *macheps* を超えた場合には影響とはならない。

---

## ?laebz

実数対称三重対角行列の指定された値以下の固有値の個数を計算し、また ?stebz ルーチンが必要とする他の処理を実行する。

---

### 構文

```
call slaebz( ijob, nitmax, n, mmax, minp, nbmin, abstol,
            reltol, pivmin, d, e, e2, nval, ab, c, mout, nab,
            work, iwork, info )
call dlaebz( ijob, nitmax, n, mmax, minp, nbmin, abstol,
            reltol, pivmin, d, e, e2, nval, ab, c, mout, nab,
            work, iwork, info )
```

## 説明

ルーチン `?laebz` は関数  $N(w)$  を使った反復ループで構成される。関数  $N(w)$  は、対称三重対角行列  $T$  から、引数  $w$  以下の固有値を計数する。ループには 2 つのタイプがある。

$ijob = 1$ 、このあと続けて、

$ijob = 2$ : 区間リストを入力として受け取り、元の区間の和集合と同一の固有値の和集合を持つ十分小さい区間のリストを返す。

入力区間は  $(ab(j,1), ab(j,2)]$ ,  $j=1, \dots, minp$  である。出力区間  $(ab(j,1), ab(j,2)]$  は固有値  $nab(j,1)+1, \dots, nab(j,2)$  を含み、ここで  $1 \leq j \leq mout$  である。

$ijob = 3$ : 各入力区間  $(ab(j,1), ab(j,2)]$  内で  $N(w(j))=nval(j)$  となる  $w(j)$  点に対してバイナリサーチを実行し、サーチでは  $c(j)$  を開始点として使用する。そのような  $w(j)$  が見つかった場合、出力は  $ab(j,1)=ab(j,2)=w$  となる。 $w(j)$  が見つからなかった場合、出力  $(ab(j,1), ab(j,2)]$  は、初期区間外に点がない限り  $N(w)$  が  $nval(j)$  を飛び越える点を含んだ小区間となる。前記区間はどのような場合でも半开区間である  $B$

すなわち、形式  $(a,b]$  において  $b$  は含まれるが  $a$  は含まれないことに注意する。

アンダーフローを防ぐために、行列を、その最大の成分が  $overflow^{**}(1/2) * underflow^{**}(1/4)$  よりも絶対値において超えないようにスケールしなければならない。小さい固有値の計算を最も精度高く行うには、行列を上記条件よりもはるかに小さくスケールしてはならない。

注: 通常、引数の妥当性は確認されない。

## 入力パラメータ

$ijob$	INTEGER。処理方法を指定する。 = 1: 初期区間に対して $nab$ を計算する。 = 2: $T$ の固有値を探すために二分法の反復を実行する。 = 3: $N(w)$ を反転するため、すなわち、指定した個数の $T$ の固有値を左辺に持つ点を探すため、二分法の反復を実行する。 そのほかの値の場合、 <code>?laebz</code> は $info = -1$ を返す。
$nitmax$	INTEGER 実行する二分法の「レベル」の最大数で、すなわち、幅 $W$ の区間を $2^{(-nitmax)} * W$ よりも小さくしない。 $nitmax$ 反復後にすべての区間が収束しなかった場合、 $info$ には非収束区間数が設定される。
$n$	INTEGER。 三重対角行列 $T$ の次元 $n$ 。この値は 1 以上でなければならない。

<i>mmax</i>	INTEGER。 区間の最大数。 <i>mmax</i> より大きい区間が生成された場合、 <i>?laebz</i> は <i>info = mmax+1</i> を返して終了する。
<i>minp</i>	INTEGER。 区間の初期数。 <i>mmax</i> よりも大きくてはならない。
<i>nbmin</i>	INTEGER。 ベクトルループを使って処理されるべき、区間の最小値。ゼロの場合、スカラループのみが使用される。
<i>abstol</i>	REAL( <i>slaebz</i> の場合) DOUBLE PRECISION( <i>dlaebz</i> の場合) 区間の最小(絶対)幅。区間が <i>abstol</i> よりも狭い場合、または(大きさで)大きい方の終点の <i>reltol</i> 倍より狭い場合、区間は十分に小さい、すなわち収束としてみなされる。この値は 0 以上でなければならない。
<i>reltol</i>	REAL( <i>slaebz</i> の場合) DOUBLE PRECISION( <i>dlaebz</i> の場合) 区間の最小相対幅。区間が <i>abstol</i> よりも狭い場合、または(大きさで)大きい方の終点の <i>reltol</i> 倍より狭い場合、区間は十分に小さい、すなわち収束としてみなされる。 <i>radix*machine epsilon</i> 以上でなければならない。
<i>pivmin</i>	REAL( <i>slaebz</i> の場合) DOUBLE PRECISION( <i>dlaebz</i> の場合) Sturm シーケンス・ループにおける「ピボット」の最小絶対値。この値は $ e(j)  * 2 * \text{safe\_min}$ 以上で、かつ <i>safe_min</i> 以上でなければならない、ここで <i>safe_min</i> はオーバーフローなしで除算できる最小値である。
<i>d, e, e2</i>	REAL( <i>slaebz</i> の場合) DOUBLE PRECISION( <i>dlaebz</i> の場合) 配列、次元はそれぞれ、( <i>n</i> ) 配列 <i>d</i> には三重対角行列 <i>T</i> の対角成分を格納する。  配列 <i>e</i> には 1 から <i>n-1</i> の位置にある三重対角行列 <i>T</i> の非対角成分を格納する。 <i>e(n)</i> は任意である。  配列 <i>e2</i> には三重対角行列 <i>T</i> の非対角成分の正方を格納する。 <i>e2(n)</i> は無視される。
<i>nval</i>	INTEGER。 配列、次元は ( <i>minp</i> ) <i>ijob = 1</i> または <i>2</i> の場合は参照されない。 <i>ijob = 3</i> の場合、 <i>N(w)</i> の求める値。

<i>ab</i>	<p>REAL (slaebz の場合 )</p> <p>DOUBLE PRECISION (dlaebz の場合 )</p> <p>配列、次元は (<i>mmax</i>, 2)</p> <p>区間の終点。<i>ab</i>(<i>j</i>, 1) は <i>j</i> 番目の区間の左側終点 <i>a</i>(<i>j</i>)、<i>ab</i>(<i>j</i>, 2) は <i>j</i> 番目の区間の右側終点 <i>b</i>(<i>j</i>)</p>
<i>c</i>	<p>REAL (slaebz の場合 )</p> <p>DOUBLE PRECISION (dlaebz の場合 )</p> <p>配列、次元は (<i>mmax</i>)</p> <p><i>ijob</i> = 1 の場合は無視される。</p> <p><i>ijob</i> = 2 の場合、ワークスペース。</p> <p><i>ijob</i> = 3 の場合、<i>c</i>(<i>j</i>) はバイナリサーチにおける最初の検索点に初期化されなければならない。</p>
<i>nab</i>	<p>INTEGER。</p> <p>配列、次元は (<i>mmax</i>, 2)</p> <p><i>ijob</i> = 2 の場合は <i>nab</i>(<i>i</i>, <i>j</i>) を設定しなければならない。次の条件を満たす必要がある。</p> <p><math>N(ab(i,1)) \leq nab(i,1) \leq nab(i,2) \leq N(ab(i,2))</math>、これは区間 <i>i</i> で、固有値 <i>nab</i>(<i>i</i>,1)+1,...,<i>nab</i>(<i>i</i>,2) のみが考慮されることを意味する。通常、先に <i>ijob</i> = 1 で ?laebz を呼び出しておけば、<i>nab</i>(<i>i</i>, <i>j</i>) = <i>N</i>(<i>ab</i>(<i>i</i>, <i>j</i>)) となる。</p> <p><i>ijob</i> = 3 の場合、?laebz を呼び出す前に、<i>nab</i> に別の値を通常は設定しなければならない。</p>
<i>work</i>	<p>REAL (slaebz の場合 )</p> <p>DOUBLE PRECISION (dlaebz の場合 )</p> <p>ワークスペース配列、次元は (<i>mmax</i>)。</p>
<i>iwork</i>	<p>INTEGER。</p> <p>ワークスペース配列、次元は (<i>mmax</i>)。</p>

### 出力パラメータ

<i>nval</i>	<p><i>nval</i> の成分は <i>ab</i> 内の区間に対応して順序が変更される。そのため、出力の <i>nval</i>(<i>j</i>) は一般に入力の <i>nval</i>(<i>j</i>) とは同一ではないが、出力の区間 (<i>ab</i>(<i>j</i>, 1), <i>ab</i>(<i>j</i>, 2)] に対応する。</p>
<i>ab</i>	<p>入力区間は一般に、計算によって、変更、分割、または順序の変更が生じる。</p>

<i>mout</i>	INTEGER。 <i>ijob</i> = 1 の場合、区間内の固有値の個数。 <i>ijob</i> = 2 または 3 の場合、区間出力の個数 <i>ijob</i> = 3 の場合、 <i>mout</i> は <i>minp</i> と等しい。
<i>nab</i>	<i>ijob</i> = 1 の場合、 <i>nab</i> ( <i>i,j</i> ) は <i>N</i> ( <i>ab</i> ( <i>i,j</i> )) に設定される。 <i>ijob</i> = 2 の場合、 <i>nab</i> ( <i>i,j</i> ) には $\max(na(k), \min(nb(k), N(ab(i,j))))$ が格納され、 <i>k</i> は出力区間 ( <i>ab</i> ( <i>j</i> ,1), <i>ab</i> ( <i>j</i> ,2)] をもたらした入力区間のインデックス、 <i>na</i> ( <i>k</i> ) と <i>nb</i> ( <i>k</i> ) は <i>nab</i> ( <i>k</i> ,1) と <i>nab</i> ( <i>k</i> ,2) の入力値である。 <i>ijob</i> = 3 の場合、 <i>nab</i> ( <i>i</i> , 1) が変更されない (すなわち出力値は入力値と同じ (モジュロの順序変更、 <i>nval</i> と <i>ab</i> 参照)) すべてのサーチ点 <i>w</i> に対して $N(w) > nval(i)$ でない限り、または、 <i>nab</i> ( <i>i</i> , 2) が変更されないすべてのサーチ点 <i>w</i> に対して $N(w) < nval(i)$ でない限り、 <i>nab</i> ( <i>i</i> , <i>j</i> ) には <i>N</i> ( <i>ab</i> ( <i>i</i> , <i>j</i> )) が格納される。
<i>info</i>	INTEGER。 0: すべての区間は収束した。 1 ~ <i>mmax</i> : 最後の <i>info</i> 区間は収束しなかった。 <i>mmax</i> +1: <i>mmax</i> を超える区間が生成された。

## アプリケーション・ノート

このルーチンは他の LAPACK ルーチンからの呼び出しのみを想定しているため、インターフェイスが使いにくくなっている。目的は次の 2 つである。

(a) 固有値を探す。?laebz は 1 つ以上の初期区間を *ab* に持っていなければならない、また ?laebz を *ijob*=1 で呼び出さなければならない。これは *nab* を設定し、また固有値をカウントする。固有値を持たない区間は通常この時点で捨てられる。また、区間 *i* のすべての固有値が必要ではない場合、*nab*(*i*, 1) を増やすか *nab*(*i*, 2) を減らせる。例えば、最大固有値を得るために *nab*(*i*, 1)=*nab*(*i*, 2)-1 と設定する。次に、*ijob*=2 とし、*mmax* を *ijob*=1 の呼び出しで得た *mout* 値以上にして ?laebz を呼び出す。*ijob*=2 の呼び出し後、固有値 *nab*(*i*, 1)+1 から *nab*(*i*, 2) は、*abstol* と *reltol* で指定された許容値で、およそ *ab*(*i*, 1) (または *ab*(*i*, 2)) となる。

(b) 固有値 *w*(*f*), ..., *w*(*l*) を含む区間 (*a*', *b*'] を探す。ここでは Gershgorin interval (*a*, *b*) 区間から開始する。*ab* に 2 個のサーチ区間を設定し、どちらも最初は (*a*, *b*) とする。*nval* の 1 つの成分は *f*-1 を、その他の成分には 1 を格納しなければならない一方で、*c* には *a* と *b* がそれぞれ格納されなければならない。求めた区間が (*a*, *b*) にない場合をエラーとするため、*nab*(*i*, 1) は -1 でなければならない、*nab*(*i*, 2) は *n*+1 でなければならない。そして、*ijob*=3 で ?laebz を呼び出す。  
出力で、*w*(*f*-1) < *w*(*f*) ならば、区間の 1 つ *j* は *ab*(*j*, 1)=*ab*(*j*, 2) と *nab*(*j*, 1)=*nab*(*j*, 2)=*f*-1 を



持ち、一方、指定された許容値に対して、 $w(f-k)=\dots=w(f+r)$ 、 $k > 0$  および  $r \geq 0$  ならば、区間は  $N(ab(j, 1))=nab(j, 1)=f-k$  と  $N(ab(j, 2))=nab(j, 2)=f+r$  を持つ。 $w(l) < w(l+1)$  と  $w(l-r)=\dots=w(l+k)$  は同様に扱われる。

## ?laed0

?stedc で使用される。縮退されていない対称三重対角行列の固有値と対応する固有ベクトルを分割統治法を使用してすべて求める。

### 構文

```
call slaed0( icompg, qsiz, n, d, e, q, ldq, qstore, ldqs, work, iwork, info )
call dlaed0( icompg, qsiz, n, d, e, q, ldq, qstore, ldqs, work, iwork, info )
call claed0( qsiz, n, d, e, q, ldq, qstore, ldqs, rwork, iwork, info )
call zlaed0( qsiz, n, d, e, q, ldq, qstore, ldqs, rwork, iwork, info )
```

### 説明

実数型の本ルーチンは、対称三重対角行列の固有値と ( オプションとして ) 対応する固有ベクトルを分割統治法を使用してすべて求める。

複素型の claed0/zlaed0 は、密または帯エルミート行列を縮退して得られる 1 個の対角ブロックである対称三重対角行列のすべての固有値と、密または帯行列の対応する固有ベクトルを求める

### 入力パラメータ

<i>icompg</i>	INTEGER。実数型でのみ使用される。 <i>icompg</i> = 0 の場合、固有値のみ計算する。 <i>icompg</i> = 1 の場合、元の密対称行列の固有ベクトルも計算する。配列 <i>q</i> には、元の行列から三重対角形式への縮退に使用した直交行列を格納しなければならない。 <i>icompg</i> = 2 の場合、三重対角行列の固有値と固有ベクトルを計算する。
<i>qsiz</i>	INTEGER。 フル行列から三重対角行列への縮退に使用した直交 / ユニタリ行列の次元で、 $qsiz \geq n$ ( 実数型の場合。 <i>icompg</i> = 1 の場合は $qsiz \geq n$ )
<i>n</i>	INTEGER。対称三重対角行列の次元 ( $n \geq 0$ )。

<i>d, e, rwork</i>	<p>REAL ( 単精度の場合 )</p> <p>DOUBLE PRECISION ( 倍精度の場合 )</p> <p>配列 :</p> <p><i>d</i>(*) には三重対角行列の主対角を格納する。<i>d</i> の次元は、<math>\max(1, n)</math> 以上でなければならない。</p> <p><i>e</i>(*) には三重対角行列の対角外の成分を格納する。<i>e</i> のサイズは、<math>\max(1, n-1)</math> 以上でなければならない。</p> <p><i>rwork</i>(*) は複素型でのみ使用されるワークスペース配列である。<i>rwork</i> の次元は <math>(1 + 3n + 2n \lg(n) + 3n^2)</math> 以上でなければならない、ここで <math>\lg(n) = 2^k \geq n</math> となる最小の整数 <i>k</i></p>
<i>q, qstore</i>	<p>REAL (slaed0 の場合 )</p> <p>DOUBLE PRECISION (dlaed0 の場合 )</p> <p>COMPLEX (claed0 の場合 )</p> <p>COMPLEX*16 (zlaed0 の場合 )</p> <p>配列 : <i>q</i>(<i>ldq</i>, *), <i>qstore</i>(<i>ldqs</i>, *)。これらの配列の第 2 次元は <math>\max(1, n)</math> 以上でなければならない。</p> <p>実数型の場合は、<i>icompq</i> = 0 ならば <i>q</i> は参照されない。</p> <p><i>icompq</i> = 1 の場合、<i>q</i> は、現時点で分解の対象となるフル行列の部分集合に対応する、フル行列から三重対角形式への縮退に使用する直交行列の列の部分集合を格納する。</p> <p><i>icompq</i> = 2 の場合、<i>q</i> は単位行列を格納する。</p> <p>配列 <i>qstore</i> はワークスペース配列で、<i>icomp</i> = 1 の場合のみ参照される。更新行列乗算の実行時に固有ベクトル行列の部分を格納するために使用される。</p> <p>複素型の場合 :</p> <p><i>q</i> には列がユニタリに直交している <i>qsiz</i> × <i>n</i> 行列を格納しなければならない。これは、フル密エルミート行列を ( 縮退可能な ) 対称三重対角行列に縮退させるユニタリ行列の一部分である。</p> <p>配列 <i>qstore</i> は、更新行列乗算の実行時に固有ベクトル行列の部分を格納するためにワークスペースとして使用される。</p>
<i>ldq</i>	<p>INTEGER。配列 <i>q</i> の第 1 次元。</p> <p><math>ldq \geq \max(1, n)</math></p>
<i>ldqs</i>	<p>INTEGER。配列 <i>qstore</i> の第 1 次元。 <math>ldqs \geq \max(1, n)</math></p>
<i>work</i>	<p>REAL (slaed0 の場合 )</p> <p>DOUBLE PRECISION (dlaed0 の場合 )</p> <p>ワークスペース配列で実数型でのみ使用される。</p>

$icmpq = 0$  または  $1$  の場合、 $work$  の次元は  $(1 + 3n + 2n \lg(n) + 2n^2)$  以上でなければならない、ここで  $\lg(n) = 2^k \geq n$  となる最小の整数  $k$  である。  
 $icmpq = 2$  の場合、 $work$  の次元は  $(4n + n^2)$  以上でなければならない。

$iwork$  INTEGER。  
 ワークスペース配列。  
 実数型で  $icmp = 0$  または  $1$  の場合、および複素型の場合、 $iwork$  の次元は  $(6 + 6n + 5n \lg(n))$  以上でなければならない。  
 実数型の場合は、 $icmpq = 2$  ならば  $iwork$  の次元は  $(3 + 5n)$  以上でなければならない。

### 出力パラメータ

$d$  昇順に並べられた固有値によって上書きされる。  
 $e$  配列を削除する。  
 $q$   $icmpq = 2$  の場合、 $q$  は三重対角行列の固有ベクトルで上書きされる。  
 $info$  INTEGER。  
 $info = 0$  の場合、正常に終了したことを示す。  
 $info = -i$  の場合、 $i$  番目のパラメータの値が不正であったことを示す。  
 $info = i > 0$  の場合、 $i/(n+1)$  から  $\text{mod}(i, n+1)$  の行と列からなる部分行列の操作中に、アルゴリズムが固有値の計算に失敗したことを示す。

## ?laed1

sstedc/dstedc で使用される。階数 1 対称行列による更新後に対角行列の更新された固有連立方程式を計算する。元の行列が三重対角の場合に使用される。

### 構文

```
call slaed1( n, d, q, ldq, indxq, rho, cutpnt, work, iwork, info )
call dlaed1( n, d, q, ldq, indxq, rho, cutpnt, work, iwork, info )
```

## 説明

ルーチン `?laed1` は、階数 1 対称行列による更新後に対角行列の更新された固有連立方程式を計算する。このルーチンは、三重対角行列のすべての固有値と固有ベクトルを必要とする固有問題に対してのみ使用される。[?laed7](#) は、フル対称行列の固有値のみを必要とする場合、または ( 三重対角形式に縮退された ) 固有値と固有ベクトルを必要とする場合を取り扱う。

$$T = Q(\text{in}) ( D(\text{in}) + \text{rho} * Z * Z' ) Q'(\text{in}) = Q(\text{out}) * D(\text{out}) * Q'(\text{out})$$

$Z = Q'u$  で、 $u$  は `cutpnt` 番目と `(cutpnt + 1)` 番目の成分が 1 で残りがゼロの長さ  $n$  のベクトルである。元の行列の固有ベクトルは  $Q$  に格納され、固有値は  $D$  に格納される。アルゴリズムは次の 3 過程で構成される。

最初の過程では、固有値が複数ある場合、または  $Z$  内にゼロがある場合、問題の大きさの収縮が行われる。そのような場合ごとに、永年方程式問題の次元は 1 だけ縮小される。この過程はルーチン [?laed2](#) によって実行される。

次の過程では更新された固有値を計算する。これは、ルーチン [?laed4](#) ([?laed3](#) として呼び出される) を介して永年方程式の根を見つけることによって行われる。このルーチンは現在の問題の固有ベクトルも計算する。

最後の過程では、更新された固有値を直接使用して更新された固有ベクトルを計算する。現在の問題に対する固有ベクトルは、全体問題から得られる固有ベクトルで乗算される。

## 入力パラメータ

$n$	INTEGER。対称三重対角行列の次元 ( $n \geq 0$ )。
$d, q, work$	REAL( <code>s1aed1</code> の場合) DOUBLE PRECISION( <code>d1aed1</code> の場合) 配列: $d(*)$ には階数 1 摂動行列の固有値を格納する。 $d$ の次元は、 $\max(1, n)$ 以上でなければならない。 $q(ldq, *)$ には階数 1 摂動行列の固有ベクトルを格納する。 $q$ の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $work(*)$ は、ワークスペース配列、次元は $(4n+n^2)$ 以上。
$ldq$	INTEGER。配列 $q$ の第 1 次元。 $ldq \geq \max(1, n)$
$indxq$	INTEGER。配列、次元は $(n)$ 。 $d$ に入っている 2 つの部分問題を昇順で別々にソートする置換。

<i>work</i>	REAL (slaed1 の場合) DOUBLE PRECISION (dlaed1 の場合) 階数 1 更新を生成するために使用する劣対角エントリ。
<i>cutpnt</i>	INTEGER。 先頭の部分行列にある最後の固有値の位置。 $\min(1, n) \leq \text{cutpnt} \leq n/2$
<i>iwork</i>	INTEGER。ワークスペース配列、次元は $(4n)$ 。

### 出力パラメータ

<i>d</i>	修復された行列の固有値で上書きされる。
<i>q</i>	<i>q</i> は修復された三重対角行列の固有ベクトルで上書きされる。
<i>indxq</i>	ソートされた順に部分問題を再統合する置換で上書きされ、すなわち、 $d(\text{indxq}(i=1, n))$ は昇順となる。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正であったことを示す。 <i>info</i> = 1 の場合、固有値は収束しなかったことを示す。

## ?laed2

sstedc/dstedc で使用される。固有値を併合し永年方程式を収縮させる。元の行列が三重対角の場合に使用される。

### 構文

```
call slaed2( k, n, n1, d, q, ldq, indxq, rho, z, dlamda, w, q2, indx, indxc,
            indxp, coltyp, info )
call dlaed2( k, n, n1, d, q, ldq, indxq, rho, z, dlamda, w, q2, indx, indxc,
            indxp, coltyp, info )
```

## 説明

このルーチン `?laed2` は 2 つの固有値のセットをソートされた単一のセットに併合する。続いて、問題の大きさの収縮を試みる。収縮が起こり得るには 2 つの場合がある。2 個以上の固有値が互いに近い場合、または、 $z$  ベクトルに微小エントリがある場合である。そのような場合ごとに、関連する永年方程式問題の次元は 1 だけ縮小される。

## 入力パラメータ

<code>k</code>	INTEGER。収縮されていない固有値の個数、および関連する永年方程式の次数 ( $0 \leq k \leq n$ )。
<code>n</code>	INTEGER。対称三重対角行列の次元 ( $n \geq 0$ )。
<code>n1</code>	INTEGER。先頭の部分行列にある最後の固有値の位置で、 $\min(1, n) \leq n1 \leq n/2$
<code>d, q, z</code>	REAL( <code>slaed2</code> の場合) DOUBLE PRECISION( <code>dlaed2</code> の場合) <code>d(*)</code> には結合対象となる 2 つの部分行列の固有値を格納する。 <code>d</code> の次元は、 $\max(1, n)$ 以上でなければならない。  <code>q(ldq, *)</code> には、 $(1,1)$ , $(n1,n1)$ と $(n1+1,n1+1)$ , $(n,n)$ の隅にある 2 つの平方ブロック内の 2 個の部分行列の固有ベクトルを格納する。 <code>q</code> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <code>z(*)</code> には更新ベクトルを格納する (最初部分固有ベクトル行列の最後の行と、2 番目の部分固有ベクトル行列の最初の行)。
<code>ldq</code>	INTEGER。配列 <code>q</code> の第 1 次元。 $ldq \geq \max(1, n)$
<code>indxq</code>	INTEGER。配列、次元は $(n)$ 。 <code>d</code> に入っている 2 つの部分問題を昇順で別々にソートする置換。この置換の後半にある成分は、それら値に加算された <code>n1</code> を最初に持っていないなければならない。
<code>rho</code>	REAL( <code>slaed2</code> の場合) DOUBLE PRECISION( <code>dlaed2</code> の場合) 今回再結合の対象となっている 2 つの部分行列を、そもそも分割する階数 1 切断に関連した非対角成分。
<code>indx, indxp</code>	INTEGER。  ワークスペース配列、次元はそれぞれ、 $(n)$ 。 配列 <code>indx</code> には <code>d1amda</code> の内容を昇順にソートするために使用した置換を格納する。

配列 *indx<sub>p</sub>* には *d* の収縮された値を配列の終わりに配置するために使用した置換を格納する。

*indx<sub>p</sub>*(1:*k*) は収縮されていない *d* 値を指し、*indx<sub>p</sub>*(*k*+1:*n*) は収縮された固有値を指す。

*coltyp* INTEGER。ワークスペース配列、次元は (*n*)。  
 実行中に、*q2* 行列中の次のタイプの列を示すラベルである。  
 1: 上半分のみ非ゼロ  
 2: 密  
 3: 下半分のみ非ゼロ  
 4: 収縮

### 出力パラメータ

*d* *d* には後続の (*n*-*k*) 個の更新された固有値 (収縮された) が昇順で上書きされる。

*q* *q* の最後の *n*-*k* 列には、後続の (*n*-*k*) 個の更新された固有値 (収縮された) が上書きされる。

*indx<sub>q</sub>* 壊される。

*rho* ?*laed3* が必要とした値に *rho* は変更されている。

*d<sub>lamda</sub>*, *w*, *q2* REAL(*s<sub>laed2</sub>* の場合)  
 DOUBLE PRECISION(*d<sub>laed2</sub>* の場合)  
 配列: *d<sub>lamda</sub>*(*n*), *w*(*n*), *q2*(*n1*<sup>2</sup>+(*n*-*n1*)<sup>2</sup>)

配列 *d<sub>lamda</sub>* には最初の *k* 個の固有値のコピーが格納され、永年方程式を形成するために ?*laed3* で使用される。

配列 *w* には収縮更新された最終的な *z* ベクトルの最初の *k* 個の値が格納され、?*laed3* に渡される。

配列 *q2* には最初の *k* 個の固有ベクトルのコピーが格納され、新しい固有ベクトルを解くために ?*laed3* の行列乗算 (*sgemm*/*dgemm*) で使用される。

*indx<sub>c</sub>* INTEGER。配列、次元は (*n*)。  
 収縮された *q* 行列の列を 3 つのグループに整理するために使用された置換。第 1 のグループには *n1* とその上にのみ非ゼロの成分が格納され、第 2 のグループには *n1* より下にのみ非ゼロの成分が格納され、第 3 のグループは密である。

*coltyp* *coltyp*(*i*) は、*i* = 1 から 4 のみに対するタイプ *i* の列数で上書きされる (タイプの定義については入力パラメータの *coltyp* の説明を参照)。

*info*                    INTEGER。  
                         *info* = 0 の場合、正常に終了したことを示す。  
                         *info* = -*i* の場合、*i* 番目のパラメータの値が不正であったことを示す。

---

### ?laed3

sstedc/dstedc で使用される。永年方程式の根を探し固有ベクトルを更新する。元の行列が三重対角の場合に使用される。

---

#### 構文

```
call slaed3( k, n, n1, d, q, ldq, rho, dlamda, q2, indx, ctot, w, s, info )
call dlaed3( k, n, n1, d, q, ldq, rho, dlamda, q2, indx, ctot, w, s, info )
```

#### 説明

ルーチン ?laed3 は、1 と *k* の範囲に対して、*d*、*w*、*rho* の値によって定義されているとおり、永年方程式の根を探す。?laed4 を適切に呼び出し、ここで解かれる  $k \times k$  連立方程式の固有ベクトルの行列によって結合される固有方程式ペアから固有ベクトルの行列を乗算して、その固有ベクトルを更新する。

このコードは浮動小数点演算に対してきわめて緩い仮定を行う。加減算においてガード桁を持つマシン、または Cray X-MP、Cray Y-MP、Cray C-90、または Cray-2 のような減算のガード桁を持たないバイナリマシンで動作する。ガード桁を持たない 16 進マシンや 10 進マシンでは異常終了することが考えられるが、そのような前例はない。

#### 入力パラメータ

*k*                    INTEGER。?laed4 で解かれる有利関数の項目数 ( $k \geq 0$ )。  
*n*                    INTEGER。*q* 行列の行と列の数。 $n \geq k$  (収縮で  $v > k$  になる場合がある)。  
*n1*                   INTEGER。先頭の部分行列にある最後の固有値の位置で、  
                          $\min(1,n) \leq n1 \leq n/2$ 。



$q$	<p>REAL (slaed3 の場合 )</p> <p>DOUBLE PRECISION (dlaed3 の場合 )</p> <p>配列 <math>q(ldq, *)</math>。<math>q</math> の第 2 次元は、<math>\max(1, n)</math> 以上でなければならない。 初期時には、この配列の最初の <math>k</math> 列はワークスペースとして使用される。</p>
$ldq$	<p>INTEGER。配列 <math>q</math> の第 1 次元。 <math>ldq \geq \max(1, n)</math></p>
$\rho$	<p>REAL (slaed3 の場合 )</p> <p>DOUBLE PRECISION (dlaed3 の場合 )</p> <p>階数 1 の更新方程式にあるパラメータの値。<math>\rho \geq 0</math> が必要。</p>
$d\lambda, q_2, w$	<p>REAL (slaed3 の場合 )</p> <p>DOUBLE PRECISION (dlaed3 の場合 )</p> <p>配列 : <math>d\lambda(k), q_2(ldq_2, *), w(k)</math></p> <p>配列 <math>d\lambda</math> の最初の <math>k</math> 個の成分には収縮された更新問題の古い根を格納する。これらは永年方程式の極である。</p> <p>配列 <math>q_2</math> の最初の <math>k</math> 列には、分割問題に対する収縮されていない固有ベクトルを格納する。<math>q_2</math> の第 2 次元は <math>\max(1, n)</math> 以上でなければならない。</p> <p>配列 <math>w</math> の最初の <math>k</math> 個の成分には、収縮調整更新ベクトルの成分を格納する。</p>
$indx$	<p>INTEGER。配列、次元は <math>(n)</math>。</p> <p>収縮された <math>q</math> 行列の列を 3 つのグループに整理するために使用された置換 (?laed2 参照)。?laed4 によって求められた固有ベクトルの行は、行列乗算が行われる前に同様に置換されなければならない。</p>
$ctot$	<p>INTEGER。配列、次元は <math>(4)</math>。</p> <p><math>indx</math> に記述されている、<math>q</math> に含まれる列のタイプのそれぞれの合計。 4 番目の列タイプは収縮された任意の列である。</p>
$s$	<p>REAL (slaed3 の場合 )</p> <p>DOUBLE PRECISION (dlaed3 の場合 )</p> <p>ワークスペース配列、次元は <math>(n+1)*k</math>。</p> <p>連立方程式を更新するために過去に蓄積された固有ベクトルで乗算される、修復された行列の固有ベクトルを格納する。</p>

### 出力パラメータ

$d$	REAL (slaed3 の場合 ) DOUBLE PRECISION (dlaed3 の場合 ) 配列、次元は $\max(1, n)$ 以上。 $d(i)$ には $1 \leq i \leq k$ に対する更新された固有値が格納される。
$q$	$q$ の列 1 から $k$ は更新された固有ベクトルで上書きされる。
$d\lambda mda$	前述のとおり、Cray X-MP、Cray Y-MP、Cray-2、または Cray C-90 では、ゼロに設定された最下位ビットを持っていると変更される場合がある。
$w$	削除される。
$info$	INTEGER。 $info = 0$ の場合、正常に終了したことを示す。 $info = -i$ の場合、 $i$ 番目のパラメータの値が不正であったことを示す。 $info = 1$ の場合、固有値は収束しなかったことを示す。

---

## ?laed4

sstedc/dstedc で使用される。  
永年方程式の単一根を見つける。

---

### 構文

```
call slaed4(n, i, d, z, delta, rho, dlam, info )  
call dlaed4(n, i, d, z, delta, rho, dlam, info )
```

### 説明

このサブルーチンは、成分が配列  $d$  で与えられている対角行列に対する、対称階数 1 の更新の  $i$  番目の更新された固有値を計算し、

ここで、 $i < j$  では  $D(i) < D(j)$ 、

および  $\rho > 0$  である。これは呼び出しルーチンによって配置され、一般性が失われることはない。階数 1 の更新された連立方程式はゆえに、

$$\text{diag}(D) + \rho * Z * \text{transpose}(Z)$$

$Z$  のユークリッド・ノルムを 1 とする。

この方法は、単純補間の有利関数による永久方程式内の有利関数の近似を含む。

### 入力パラメータ

<i>n</i>	INTEGER。全配列の長さ。
<i>i</i>	INTEGER。計算される固有値のインデックスで、 $1 \leq i \leq n$
<i>d</i> , <i>z</i>	REAL (slaed4 の場合) DOUBLE PRECISION (dlaed4 の場合) 配列、次元はそれぞれ ( <i>n</i> )。 配列 <i>d</i> には元の固有値を格納する。それらは、 $i < j$ では $d(i) < d(j)$ のように順序どおりに並べられていると仮定される。  配列 <i>z</i> には更新ベクトル $Z$ の成分を格納する。
<i>rho</i>	REAL (slaed4 の場合) DOUBLE PRECISION (dlaed4 の場合) 対称更新式におけるスカラ。

### 出力パラメータ

<i>delta</i>	REAL (slaed4 の場合) DOUBLE PRECISION (dlaed4 の場合) 配列、次元は ( <i>n</i> )。 $n \neq 1$ の場合、 <i>delta</i> の <i>j</i> 番目の成分に $(d(j) - \lambda_i)$ が格納される。 $n = 1$ の場合、 <i>delta</i> (1) = 1。ベクトル <i>delta</i> には固有ベクトルを構成するために必要な情報が格納される。
<i>diam</i>	REAL (slaed4 の場合) DOUBLE PRECISION (dlaed4 の場合) 計算された $\lambda_i$ 、 <i>i</i> 番目の更新された固有値。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = 1 の場合、更新処理に失敗したことを示す。

### ?laed5

sstedc/dstedc で使用される。  
 $2 \times 2$  の永年方程式を解く。

---

#### 構文

```
call slaed5(i, d, z, delta, rho, dlam )  
call dlaed5(i, d, z, delta, rho, dlam )
```

#### 説明

このサブルーチンは、 $2 \times 2$  対角行列の対称階数 1 の更新の  $i$  番目の固有値を計算する。

$\text{diag}(D) + \rho * Z * \text{transpose}(Z)$ 。

配列  $D$  における対角成分は満たされるとみなされる。

ここで、 $i < j$  では  $D(i) < D(j)$ 、

さらに、 $\rho > 0$ 、ベクトル  $Z$  のユークリッド・ノルムを 1 と仮定する。

#### 入力パラメータ

$i$	INTEGER。計算される固有値のインデックスで $1 \leq i \leq 2$
$d, z$	REAL (slaed5 の場合 ) DOUBLE PRECISION (dlaed5 の場合 ) 配列、次元はそれぞれ、(2)。 配列 $d$ には元の固有値を格納する。 $d(1) < d(2)$ と仮定する。 配列 $z$ には更新ベクトルの成分を格納する。
$\rho$	REAL (slaed5 の場合 ) DOUBLE PRECISION (dlaed5 の場合 ) 対称更新式におけるスカラ。

### 出力パラメータ

<i>delta</i>	REAL (slaed5 の場合 ) DOUBLE PRECISION (dlaed5 の場合 ) 配列、次元は (2)。 ベクトル <i>delta</i> には固有ベクトルを構成するために必要な情報が格納される。
<i>dlam</i>	REAL (slaed5 の場合 ) DOUBLE PRECISION (dlaed5 の場合 ) 計算された <i>lambda_i</i> 、 <i>i</i> 番目の更新された固有値。

## ?laed6

sstedc/dstedc で使用される。  
永久方程式の解の中から *Newton* ステップの  
1 つを計算する。

### 構文

```
call slaed6( kniter, orgati, rho, d, z, finit, tau, info )
call dlaed6( kniter, orgati, rho, d, z, finit, tau, info )
```

### 説明

このルーチンは次の式の正または負の根 ( 元にもっとも近い ) を計算する。

$$f(x) = \text{rho} + \frac{z(1)}{d(1) - x} + \frac{z(2)}{d(2) - x} + \frac{z(3)}{d(3) - x}$$

*orgati* = .TRUE の場合、根は *d(2)* と *d(3)* の間にあると仮定される。それ以外では *d(1)* と *d(2)* の間にあると仮定される。このルーチンは ?laed4 から必要に応じて呼び出される。多くの場合、根の探索は大きさにおいて最も小さいが、きわめて稀な状況ではないと考えられる。

### 入力パラメータ

*kniter*            INTEGER  
大きさに関して ?laed4 を参照。

<i>orgati</i>	LOGICAL  <i>orgati</i> = .TRUE. の場合、必要となる根は $d(2)$ と $d(3)$ の間にあり、それ以外では $d(1)$ と $d(2)$ の間にある。詳細は ?laed4 を参照。
<i>rho</i>	REAL (slaed6 の場合 ) DOUBLE PRECISION (dlaed6 の場合 ) 上記の $f(x)$ の式を参照。
<i>d, z</i>	REAL (slaed6 の場合 ) DOUBLE PRECISION (dlaed6 の場合 ) 配列、次元はそれぞれ、(3)。  $d(1) < d(2) < d(3)$ を満たす配列 $d$  配列 $z$ の各成分は正でなければならない。
<i>finit</i>	REAL (slaed6 の場合 ) DOUBLE PRECISION (dlaed6 の場合 ) ゼロにおける $f(x)$ の値。このルーチン内部を評価した値よりも正確であること ( そのようなことを望む場合 )。

### 出力パラメータ

<i>tau</i>	REAL (slaed6 の場合 ) DOUBLE PRECISION (dlaed6 の場合 ) $f(x)$ に対する式の根。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = 1 の場合、収束に失敗したことを示す。

---

## ?laed7

?stedc で使用される。階数 1 対称行列による更新後に対角行列の更新された固有連立方程式を計算する。元の行列が密の場合に使用される。

---

### 構文

```
call slaed7(  icompq, n, qsiz, tlvls, curlvl, curpbm, d, q, ldq,  
              indxq, rho, cutpnt, qstore, qptra, prmptra, perm, givptr, givcol,  
              givnum, work, iwork, info )
```

```
call dlaed7( icipq, n, qsiz, tlvls, curlvl, curpbm, d, q, ldq,
             indxq, rho, cutpnt, qstore, qptra, prmptr, perm, givptr, givcol,
             givnum, work, iwork, info )
call claed7( n, cutpnt, qsiz, tlvls, curlvl, curpbm, d, q, ldq, rho,
             indxq, qstore, qptra, prmptr, perm, givptr, givcol, givnum,
             work, rwork, iwork, info )
call zlaed7( n, cutpnt, qsiz, tlvls, curlvl, curpbm, d, q, ldq, rho,
             indxq, qstore, qptra, prmptr, perm, givptr, givcol, givnum,
             work, rwork, iwork, info )
```

## 説明

ルーチン `?laed7` は、階数 1 対称行列による更新後に対角行列の更新された固有連立方程式を計算する。このルーチンは、三重対角形式に縮退された密対称 / エルミート行列のすべての固有値と、オプションで固有ベクトルを必要とする固有問題に対してのみ使用される。実数型では、`slaed1/dlaed1` は、対称三重対角行列のすべての固有値と固有ベクトルが必要な場合を取り扱う。

$$T = Q(\text{in}) ( D(\text{in}) + \rho * Z * Z' ) Q'(\text{in}) = Q(\text{out}) * D(\text{out}) * Q'(\text{out})$$

$Z = Q'u$  で、 $u$  は `cutpnt` 番目と `(cutpnt + 1)` 番目の成分が 1 で残りがゼロの長さ  $n$  のベクトルである。元の行列の固有ベクトルは  $Q$  に格納され、固有値は  $D$  に格納される。アルゴリズムは次の 3 過程で構成される。

最初の過程では、固有値が複数ある場合、または  $Z$  内にゼロがある場合、問題の大きさの収縮が行われる。そのような場合ごとに、永年方程式問題の次元は 1 だけ縮小される。この過程はルーチン `slaed8/dlaed8` (実数型)、またはルーチン `slaed2/dlaed2` (複素型) によって実行される。

次の過程では更新された固有値を計算する。これは、ルーチン `?laed4` (`?laed3` または `?laed9` として呼び出される) を介して永年方程式の根を見つけることによって行われる。このルーチンは現在の問題の固有ベクトルも計算する。

最後の過程では、更新された固有値を直接使用して更新された固有ベクトルを計算する。現在の問題に対する固有ベクトルは、全体問題から得られる固有ベクトルで乗算される。

## 入力パラメータ

`icipq` INTEGER。実数型でのみ使用される。  
`icipq = 0` の場合、固有値のみ計算する。  
`icipq = 1` の場合、元の密対称行列の固有ベクトルも計算する。配列  $q$  には、元の行列から三重対角形式への縮退に使用した直交行列を格納しなければならない。

<i>n</i>	INTEGER。対称三重対角行列の次元 ( $n \geq 0$ )。
<i>cutpnt</i>	INTEGER。先頭の部分行列にある最後の固有値の位置。 $\min(1, n) \leq \text{cutpnt} \leq n$
<i>qsiz</i>	INTEGER。フル行列から三重対角行列への縮退に使用した直交 / ユニタリ行列の次元で、 $qsiz \geq n$ (実数型の場合。 $icompq = 1$ の場合は $qsiz \geq n$ )
<i>tlvls</i>	INTEGER。分割統治ツリー全体の併合レベルの合計数。
<i>curlvl</i>	INTEGER。併合ルーチン全体の現在のレベルで、 $0 \leq \text{curlvl} \leq \text{tlvls}$
<i>curpbm</i>	INTEGER。併合ルーチン全体の現在のレベルにおける現在の問題 (左上から右下に向かって計数)。
<i>d</i>	REAL (slaed7/claed7 の場合) DOUBLE PRECISION (dlaed7/zlaed7 の場合)  配列、次元は $\max(1, n)$ 以上。 配列 <i>d</i> (*) には階数 1 置換行列の固有値を格納する。
<i>q, work</i>	REAL (slaed7 の場合) DOUBLE PRECISION (dlaed7 の場合) COMPLEX (claed7 の場合) COMPLEX*16 (zlaed7 の場合) 配列: <i>q</i> ( <i>ldq</i> , *) には階数 1 置換行列の固有ベクトルを格納する。 <i>q</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。  <i>work</i> (*) はワークスペース配列、次元は、実数の場合は $(3n+qsiz*n)$ 以上、複素数の場合は $(qsiz*n)$ 以上。
<i>ldq</i>	INTEGER。配列 <i>q</i> の第 1 次元。 $ldq \geq \max(1, n)$
<i>rho</i>	REAL (slaed7/claed7 の場合) DOUBLE PRECISION (dlaed7/zlaed7 の場合) 階数 1 更新を生成するために使用された部分対角成分。
<i>qstore</i>	REAL (slaed7/claed7 の場合) DOUBLE PRECISION (dlaed7/zlaed7 の場合) 配列、次元は $(n^2+1)$ 。出力パラメータとしても働く。 分割統治中に遭遇した部分行列の固有ベクトルを圧縮して格納する。 <i>qptr</i> は部分行列の始まりを指す。



<i>qp</i> <i>tr</i>	INTEGER。配列、次元は $(n+2)$ 。出力パラメータとしても働く。 <i>qstore</i> に格納されている部分行列の始まりを指すインデックス・リスト。部分行列は、分割統治ツリーの左下を先頭にして、左から右へ、下から上へ番号が振られる。
<i>prmptr</i> , <i>perm</i> , <i>givptr</i>	INTEGER。配列、次元はそれぞれ、 $(n \lg n)$ 。  配列 <i>prmptr</i> (*) には、あるレベルの置換が <i>perm</i> 内のどこに格納されているかを示すポインタリストを格納する。 <i>prmptr</i> ( <i>i</i> +1) - <i>prmptr</i> ( <i>i</i> ) は置換のサイズを表し、また収縮されていない全問題の大きさも表す。  配列 <i>perm</i> (*) には各固有ブロックに適用される (収縮とソートによる) 置換を格納する。  配列 <i>givptr</i> (*) には、あるレベルの Givens 回転が <i>givcol</i> 内のどこに格納されているかを示すポインタリストを格納する。 <i>givptr</i> ( <i>i</i> +1) - <i>givptr</i> ( <i>i</i> ) は Givens 回転の回数を示す。
<i>givcol</i>	INTEGER。配列、次元は $(2, n \lg n)$ 。 それぞれの数のペアは Givens 回転で対象となる列ペアを示す。
<i>givnum</i>	REAL ( <i>slaed7</i> / <i>claed7</i> の場合) DOUBLE PRECISION ( <i>dlaed7</i> / <i>zlaed7</i> の場合) 配列、次元は $(2, n \lg n)$ 。 それぞれの数は対応する Givens 回転で使用される <i>S</i> 値を示す。
<i>iwork</i>	INTEGER。ワークスペース配列、次元は $(4n)$ 。
<i>rwork</i>	REAL ( <i>claed7</i> の場合) DOUBLE PRECISION ( <i>zlaed7</i> の場合) ワークスペース配列、次元は $(3n+2qsiz*n)$ 。複素型でのみ使用される。

### 出力パラメータ

<i>d</i>	修復された行列の固有値で上書きされる。
<i>q</i>	<i>q</i> は修復された三重対角行列の固有ベクトルで上書きされる。
<i>indxq</i>	INTEGER。配列、次元は $(n)$ 。 ソートされた順に部分問題を再統合する置換で上書きされ、すなわち、 <i>d</i> ( <i>indxq</i> ( <i>i</i> = 1, <i>n</i> )) は昇順となる。

*info*                    INTEGER。  
*info* = 0 の場合、正常に終了したことを示す。  
*info* = -*i* の場合、*i* 番目のパラメータの値が不正であったことを示す。  
*info* = 1 の場合、固有値は収束しなかったことを示す。

---

### ?laed8

?stedc で使用される。固有値を併合し永年方程式を収縮させる。元の行列が密の場合に使用される。

---

#### 構文

```
call slaed8( icompq, k, n, qsiz, d, q, ldq, indxq, rho, cutpnt, z,  
            dlamda, q2, ldq2, w, perm, givptr, givcol, givnum, indx,  
            info )  
  
call dlaed8( icompq, k, n, qsiz, d, q, ldq, indxq, rho, cutpnt, z,  
            dlamda, q2, ldq2, w, perm, givptr, givcol, givnum, indx,  
            info )  
  
call claed8( k, n, qsiz, q, ldq, d, rho, cutpnt, z, dlamda, q2,  
            ldq2, w, indx, indx, indxq, perm, givptr, givcol, givnum,  
            info )  
  
call zlaed8( k, n, qsiz, q, ldq, d, rho, cutpnt, z, dlamda, q2,  
            ldq2, w, indx, indx, indxq, perm, givptr, givcol, givnum,  
            info )
```

#### 説明

このルーチンは2つの固有値のセットをソートされた単一のセットに併合する。続いて、問題の大きさの収縮を試みる。収縮が起こり得るには2つの場合がある。2個以上の固有値が互いに近い場合、または、**z** ベクトルに微小エントリがある場合である。そのような場合ごとに、関連する永年方程式問題の次元は1だけ縮小される。

#### 入力パラメータ

*icompq*                INTEGER。実数型でのみ使用される。

	<p><math>icompg = 0</math> の場合、固有値のみ計算する。</p> <p><math>icompg = 1</math> の場合、元の密対称行列の固有ベクトルも計算する。配列 <math>q</math> には、元の行列から三重対角形式への縮退に使用した直交行列を格納しなければならない。</p>
$n$	INTEGER。対称三重対角行列の次元 ( $n \geq 0$ )。
$cutpnt$	INTEGER。先頭の部分行列にある最後の固有値の位置。 $\min(1, n) \leq cutpnt \leq n$
$qsiz$	INTEGER。フル行列から三重対角行列への縮退に使用した直交 / ユニタリ行列の次元で、 $qsiz \geq n$ (実数型の場合。 $icompg = 1$ の場合は $qsiz \geq n$ )
$d, z$	<p>REAL (slaed8/claed8 の場合)</p> <p>DOUBLE PRECISION (dlaed8/zlaed8 の場合)</p> <p>配列、次元はそれぞれ、<math>\max(1, n)</math> 以上。</p> <p>配列 <math>d(*)</math> には結合する 2 つの部分行列の固有値を格納する。</p> <p><math>z(*)</math> には更新ベクトルを格納する (最初部分固有ベクトル行列の最後の行と、2 番目の部分固有ベクトル行列の最初の行)。 <math>z</math> の内容は更新過程で壊される。</p>
$q$	<p>REAL (slaed8 の場合)</p> <p>DOUBLE PRECISION (dlaed8 の場合)</p> <p>COMPLEX (claed8 の場合)</p> <p>COMPLEX*16 (zlaed8 の場合)</p> <p>配列 <math>q(ldq, *)</math>。 <math>q</math> の第 2 次元は、<math>\max(1, n)</math> 以上でなければならない。</p> <p><math>q</math> には、他の部分的に解かれた固有連立方程式との行列乗算によって、すでに更新された部分的に解かれた連立方程式の固有ベクトルを格納する。</p> <p>実数型の場合は、<math>icompg = 0</math> ならば <math>q</math> は参照されない。</p>
$ldq$	INTEGER。配列 $q$ の第 1 次元。 $ldq \geq \max(1, n)$
$ldq2$	INTEGER。出力配列 $q2$ の第 1 次元のサイズ。 $ldq2 \geq \max(1, n)$
$indxq$	<p>INTEGER。配列、次元は (<math>n</math>)。</p> <p><math>d</math> に入っている 2 つの部分問題を昇順で別々にソートする置換。この置換の後半にある成分は、精度を得るために、それら値に加算された <math>cutpnt</math> を最初に持っていなければならない。</p>

*rho* REAL (slaed8/claed8 の場合)  
DOUBLE PRECISION (dlaed8/zlaed8 の場合)  
今回再結合の対象となっている 2 つの部分行列を、そもそも分割する  
階数 1 切断に関連した非対角成分。

## 出力パラメータ

*k* INTEGER。非収縮の固有値の数、関連永年方程式の次数。

*d* 後続の  $(n-k)$  個の更新された固有値 (収縮された) が昇順で上書きされる。

*q* *q* の最後の  $n-k$  列には、後続の  $(n-k)$  個の更新された固有値 (収縮された) が上書きされる。

*rho* ?laed3 が必要とした値に *rho* は変更されている。

*dlambda, w* REAL (slaed8/claed8 の場合)  
DOUBLE PRECISION (dlaed8/zlaed8 の場合)  
配列、次元はそれぞれ、 $(n)$   
配列 *dlambda*(\*) には最初の *k* 個の固有値のコピーが格納され、永年方程式を形成するために ?laed3 で使用される。  
配列 *w*(\*) は収縮更新された最終的な *z* ベクトルの最初の *k* 個の値を保持し、?laed3 に渡される。

*q2* REAL (slaed8 の場合)  
DOUBLE PRECISION (dlaed8 の場合)  
COMPLEX (claed8 の場合)  
COMPLEX\*16 (zlaed8 の場合)  
配列 *q2*(1:*dq2*, \*)。 *q2* の第 2 次元は  $\max(1, n)$  以上でなければならない。  
新しい固有値の更新のために、slaed7/dlaed7 の行列乗算 (sgemm/dgemm) で使用される最初の *k* 個の固有ベクトルのコピーが格納される。  
実数型の場合は、*icompq* = 0 ならば *q2* は参照されない。

*indx, indx* INTEGER。ワークスペース配列、次元はそれぞれ、 $(n)$ 。  
配列 *indx*(\*) には、*d* の収縮した値を配列の終わりに配置するために使用した置換を格納する。*indx*(1:*k*) は収縮されていない *d* 値を指し、*indx*(*k*+1:*n*) は収縮された固有値を指す。  
配列 *indx*(\*) には *d* の内容を昇順にソートするために使用した置換を格納する。

<i>perm</i>	INTEGER。配列、次元は $(n)$ 。 各固有ブロックに適用される (収縮とソートによる) 置換が格納される。
<i>givptr</i>	INTEGER。この部分問題で実行された <b>Givens</b> 回転の回数が格納される。
<i>givcol</i>	INTEGER。配列、次元は $(2, n)$ 。 それぞれの数のペアは <b>Givens</b> 回転で対象となる列ペアを示す。
<i>givnum</i>	REAL (slaed8/claed8 の場合) DOUBLE PRECISION (dlaed8/zlaed8 の場合) 配列、次元は $(2, n)$ 。 それぞれの数は対応する <b>Givens</b> 回転で使用される $S$ 値を示す。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正であったことを示す。

## ?laed9

sstedc/dstedc で使用される。  
永年方程式の根を探し固有ベクトルを更新する。  
元の行列が密の場合に使用される。

### 構文

```
call slaed9( k, kstart, kstop, n, d, q, ldq, rho, dlamda, w, s, lds, info )
call dlaed9( k, kstart, kstop, n, d, q, ldq, rho, dlamda, w, s, lds, info )
```

### 説明

ルーチン ?laed3 は、*kstart* と *kstop* の範囲で、*d*、*w*、*rho* の値によって定義されているとおり、永年方程式の根を探す。slaed4/dlaed4 への適切な呼び出しを行い、続いて、*z* ベクトルの次のレベルの計算で用いる固有ベクトルの新しい行列を格納する。

### 入力パラメータ

*k* INTEGER。slaed4/dlaed4 で解かれる有利関数の項目数 ( $k \geq 0$ )。

<i>kstart</i> , <i>kstop</i>	INTEGER。更新された固有値 <i>lambda</i> ( <i>i</i> )、 $kstart \leq i \leq kstop$ で計算される。 $1 \leq kstart \leq kstop \leq k$
<i>n</i>	INTEGER。 <i>Q</i> 行列の行と列の数。 $n \geq k$ (収縮で $n > k$ になる場合がある)。
<i>q</i>	REAL (slaed9 の場合) DOUBLE PRECISION (dlaed9 の場合) ワークスペース配列、次元は ( <i>ldq</i> , *)。 <i>q</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。
<i>ldq</i>	INTEGER。配列 <i>q</i> の第 1 次元。 $ldq \geq \max(1, n)$
<i>rho</i>	REAL (slaed9 の場合) DOUBLE PRECISION (dlaed9 の場合) 階数 1 の更新方程式にあるパラメータの値。 $\rho \geq 0$ が必要。
<i>dlambda</i> , <i>w</i>	REAL (slaed9 の場合) DOUBLE PRECISION (dlaed9 の場合) 配列、次元はそれぞれ ( <i>k</i> )。 配列 <i>dlambda</i> (*) の最初の <i>k</i> 個の成分には収縮された更新問題の古い根を格納する。これらは永年方程式の極である。  配列 <i>w</i> (*) の最初の <i>k</i> 個の成分には、収縮調整更新ベクトルの成分を格納する。
<i>lds</i>	INTEGER。出力配列 <i>s</i> の第 1 次元のサイズ。 $lds \geq \max(1, k)$

## 出力パラメータ

<i>d</i>	REAL (slaed9 の場合) DOUBLE PRECISION (dlaed9 の場合) 配列、次元は ( <i>n</i> )。 <i>d</i> ( <i>i</i> ) には $kstart \leq i \leq kstop$ に対する更新された固有値が格納される。
<i>s</i>	REAL (slaed9 の場合) DOUBLE PRECISION (dlaed9 の場合) 配列、次元は ( <i>lds</i> , *)。 <i>s</i> の第 2 次元は $\max(1, k)$ 以上でなければならない。 続く <i>z</i> ベクトル計算のために格納され、また、連立方程式の更新のために過去に蓄積された固有ベクトルで乗算される、修復された行列の固有ベクトルが格納される。

*info* INTEGER。  
*info* = 0 の場合、正常に終了したことを示す。  
*info* = -*i* の場合、*i* 番目のパラメータの値が不正であったことを示す。  
*info* = 1 の場合、固有値は収束しなかったことを示す。

## ?laeda

?stedc で使用される。対角行列の階数 1 更新を決定する Z ベクトルを計算する。元の行列が密の場合に使用される。

### 構文

```
call slaeda( n, tlvls, curlvl, curpbm, prmptr, perm, givptr, givcol,
             givnum, q, qptr, z, ztemp, info )
call dlaeda( n, tlvls, curlvl, curpbm, prmptr, perm, givptr, givcol,
             givnum, q, qptr, z, ztemp, info )
```

### 説明

?laeda ルーチンは、*curpbm* 番目問題に対する *tlvls* ステップを用いて、併合過程の *curlvl* 番目のステップ中の併合ステップに対応した Z ベクトルを計算する。

### 入力パラメータ

*n* INTEGER。対称三重対角行列の次元 ( $n \geq 0$ )。  
*tlvls* INTEGER。分割統治ツリー全体の併合レベルの合計数。  
*curlvl* INTEGER。併合ルーチン全体の現在のレベルで、 $0 \leq \text{curlvl} \leq \text{tlvls}$   
*curpbm* INTEGER。併合ルーチン全体の現在のレベルにおける現在の問題 (左上から右下に向かって計数)。  
*prmptr*, *perm*, *givptr* INTEGER。配列、次元はそれぞれ、( $n \lg n$ )。  
配列 *prmptr*(\*) には、あるレベルの置換が *perm* 内のどこに格納されているかを示すポインタリストを格納する。*prmptr*(*i*+1) - *prmptr*(*i*) は置換のサイズを表し、また収縮されていない全問題の大きさも表す。

配列 *perm(\*)* には各固有ブロックに適用される (収縮とソートによる) 置換を格納する。

配列 *givptr(\*)* には、あるレベルの Givens 回転が *givcol* 内のどこに格納されているかを示すポインタリストを格納する。  
*givptr(i+1) - givptr(i)* は Givens 回転の回数を示す。

<i>givcol</i>	INTEGER。配列、次元は $(2, n \lg n)$ それぞれの数のペアは Givens 回転で対象となる列ペアを示す。
<i>givnum</i>	REAL ( <i>slaeda</i> の場合) DOUBLE PRECISION ( <i>dlaeda</i> の場合) 配列、次元は $(2, n \lg n)$ 。 それぞれの数は対応する Givens 回転で使用する <i>S</i> 値を示す。
<i>q</i>	REAL ( <i>slaed0</i> の場合) DOUBLE PRECISION ( <i>dlaed0</i> の場合) 配列、次元は $(n^2)$ 。 以前のレベルから得られる平方固有ブロックを格納する。ブロックの開始位置は <i>qptra</i> で与えられる。
<i>qptra</i>	INTEGER。配列、次元は $(n+2)$ 。固有ブロックが <i>q</i> のどこに格納されているかを示すポインタのリストを格納する。 $\text{sqrt}(\text{qptra}(i+1) - \text{qptra}(i))$ はブロックの大きさを示す。
<i>ztemp</i>	REAL ( <i>slaeda</i> の場合) DOUBLE PRECISION ( <i>dlaeda</i> の場合) ワークスペース配列、次元は $(n)$ 。

## 出力パラメータ

<i>z</i>	REAL ( <i>slaeda</i> の場合) DOUBLE PRECISION ( <i>dlaeda</i> の場合) 配列、次元は $(n)$ 。更新ベクトルが格納される (最初の部分固有ベクトル行列の最後の行と、2 番目の部分固有ベクトル行列の最初の行)。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正であったことを示す。



## ?laein

上 Hessenberg 行列の指定された右または左固有ベクトルを逆反復法を用いて計算する。

### 構文

```
call slaein( rightv, noinit, n, h, ldh, wr, wi, vr, vi, b, ldb, work, eps3,
            smlnum, bignum, info )
call dlaein( rightv, noinit, n, h, ldh, wr, wi, vr, vi, b, ldb, work, eps3,
            smlnum, bignum, info )
call claein( rightv, noinit, n, h, ldh, w, v, b, ldb, rwork, eps3, smlnum,
            info )
call zlaein( rightv, noinit, n, h, ldh, w, v, b, ldb, rwork, eps3, smlnum,
            info )
```

### 説明

?laein ルーチンは、実数の上 Hessenberg 行列  $H$  (実数型 slaein/dlaein) の固有値 ( $wr, wi$ )、または複素の上 Hessenberg 行列  $H$  (複素型 claein/zlaein) の固有値  $w$  に対応した右または左固有ベクトルを逆反復法を使用して探す。

### 入力パラメータ

<i>rightv</i>	LOGICAL。 <i>rightv</i> = .TRUE. の場合、右固有ベクトルを計算する。 <i>rightv</i> = .FALSE. の場合、左固有ベクトルを計算する。
<i>noinit</i>	LOGICAL。 <i>noinit</i> = .TRUE. の場合、( <i>vr, vi</i> ) または <i>v</i> (複素型) において初期ベクトルは与えられない。 <i>noinit</i> = .FALSE. の場合、( <i>vr, vi</i> ) または <i>v</i> (複素型) において初期ベクトルが与えられる。
<i>n</i>	INTEGER。行列 $H$ の次数 ( $n \geq 0$ )。
<i>h</i>	REAL (slaein の場合) DOUBLE PRECISION (dlaein の場合) COMPLEX (claein の場合) COMPLEX*16 (zlaein の場合) 配列 $h(ldh, *)$ 。 $h$ の第 2 次元は $\max(1, n)$ 以上でなければならない。 Hessenberg 上行列 $H$ が格納される。

<i>ldh</i>	INTEGER。配列 <i>h</i> の第 1 次元。 $ldh \geq \max(1, n)$
<i>wr, wi</i>	REAL (slaein の場合) DOUBLE PRECISION (dlaein の場合) その対応する右または左の固有ベクトルが、計算対象となっている <i>H</i> の固有値の実数部分および虚数部分 (実数型ルーチン)。
<i>w</i>	COMPLEX (claein の場合) COMPLEX*16 (zlaein の場合) その対応する右または左の固有ベクトルが計算対象となっている <i>H</i> の固有値 (複素型ルーチン)。
<i>vr, vi</i>	REAL (slaein の場合) DOUBLE PRECISION (dlaein の場合) 配列、次元はそれぞれ、( <i>n</i> )。実数型でのみ使用される。 <i>noinit</i> = .FALSE. かつ <i>wi</i> = 0.0 の場合、実数固有値 <i>wr</i> を使った逆反復のために、 <i>vr</i> には実数開始ベクトルを格納しなければならない。 <i>noinit</i> = .FALSE. かつ <i>wi</i> ≠ 0.0 の場合、複素固有値 ( <i>wr, wi</i> ) を使った逆反復のために、 <i>vr</i> と <i>vi</i> には複素開始ベクトルの実数と虚数部分を格納しなければならない。それ以外の場合は <i>vr</i> と <i>vi</i> を設定する必要はない。
<i>v</i>	COMPLEX (claein の場合) COMPLEX*16 (zlaein の場合) 配列、次元は ( <i>n</i> )。複素型でのみ使用される。 <i>noinit</i> = .FALSE. 場合、 <i>v</i> には逆反復のために開始ベクトルを格納しなければならない。それ以外の場合は <i>v</i> を設定する必要はない。
<i>b</i>	REAL (slaein の場合) DOUBLE PRECISION (dlaein の場合) COMPLEX (claein の場合) COMPLEX*16 (zlaein の場合) ワークスペース配列 <i>b</i> ( <i>ldb, *</i> )。 <i>b</i> の第 2 次元は $\max(1, n)$ 以上でなければならない。
<i>ldb</i>	INTEGER。配列 <i>b</i> の第 1 次元。 実数型は $ldb \geq n+1$ 複素型は $ldb \geq \max(1, n)$
<i>work</i>	REAL (slaein の場合) DOUBLE PRECISION (dlaein の場合) ワークスペース配列、次元は ( <i>n</i> )。実数型でのみ使用される。

<i>rwork</i>	REAL (claein の場合 ) DOUBLE PRECISION (zlaein の場合 ) ワークスペース配列、次元は (n)。複素型でのみ使用される。
<i>eps3, smlnum</i>	REAL (slaein/claein の場合 ) DOUBLE PRECISION (dlaein/zlaein の場合 ) <i>eps3</i> は小さなマシン依存値で、値が近い固有値を摂動するため、およびゼロピボットを交換するために使われる。 <i>smlnum</i> はマシン依存値でアンダーフローしきい値に近い。
<i>bignum</i>	REAL (slaein の場合 ) DOUBLE PRECISION (dlaein の場合 ) <i>bignum</i> はマシン依存値でオーバーフローしきい値に近い。実数型でのみ使用される。

### 出力パラメータ

<i>vr, vi</i>	<i>wi</i> = 0.0 (実数固有値) の場合、 <i>vr</i> は計算された実数固有ベクトルで上書きされ、 <i>wi</i> ≠ 0.0 (複素固有値) の場合、 <i>vr</i> と <i>vi</i> は計算された複素固有ベクトルの実数部と虚数部で上書きされる。固有ベクトルは最大の大きさの成分が大きさ 1 を持つように正規化される。ここで、複素値 (x,y) の大きさは $ x  +  y $ として求められる。 <i>wi</i> = 0.0 の場合、 <i>vi</i> は参照されない。
<i>v</i>	<i>v</i> は、最大の大きさの成分が大きさ 1 を持つように正規化された、計算された固有ベクトルで上書きされる。ここで複素値 (x,y) の大きさは $ x  +  y $ として求められる。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = 1 の場合、逆反復は収束しなかったことを示す。実数型の場合、 <i>vr</i> は最後の反復が設定され、 <i>vi</i> も <i>wi</i> ≠ 0.0 の場合に同じである。複素型の場合、 <i>v</i> には最後の反復が設定される。

## ?laev2

2 × 2 の対称/エルミート行列の固有値と固有ベクトルを計算する。

### 構文

```
call slaev2( a, b, c, rt1, rt2, cs1, sn1 )
call dlaev2( a, b, c, rt1, rt2, cs1, sn1 )
call claev2( a, b, c, rt1, rt2, cs1, sn1 )
call zlaev2( a, b, c, rt1, rt2, cs1, sn1 )
```

### 説明

このルーチンは、固有ベクトルの行列のノルムがしきい値より大きければ、次の 2 × 2 対称行列の固有分解を実行する。

$$\begin{bmatrix} a & b \\ b & c \end{bmatrix} \text{ (slaev2/dlaev2 の場合)、またはエルミート行列 } \begin{bmatrix} a & b \\ \text{conjg}(b) & c \end{bmatrix}$$

(claev2/zlaev2 の場合) に対して固有分解を実行する。

rt1 は大きい方の絶対値の固有値で上書きされ、rt2 は小さい方の絶対値の固有値で上書きされ、(cs1, sn1) は rt1 に対する単位右固有ベクトルで次の分解を与える。

$$\begin{bmatrix} cs1 & sn1 \\ -sn1 & cs1 \end{bmatrix} \cdot \begin{bmatrix} a & b \\ b & c \end{bmatrix} \cdot \begin{bmatrix} cs1 & -sn1 \\ sn1 & cs1 \end{bmatrix} = \begin{bmatrix} rt1 & 0 \\ 0 & rt2 \end{bmatrix}$$

(slaev2/dlaev2 の場合)、  
または

$$\begin{bmatrix} cs1 & \text{conjg}(sn1) \\ -sn1 & cs1 \end{bmatrix} \cdot \begin{bmatrix} a & b \\ \text{conjg}(b) & c \end{bmatrix} \cdot \begin{bmatrix} cs1 & -\text{conjg}(sn1) \\ sn1 & cs1 \end{bmatrix} = \begin{bmatrix} rt1 & 0 \\ 0 & rt2 \end{bmatrix}$$

(claev2/zlaev2 の場合)

### 入力パラメータ

*a*, *b*, *c*      REAL (slaev2 の場合)  
                   DOUBLE PRECISION (dlaev2 の場合)  
                   COMPLEX (claev2 の場合)  
                   COMPLEX\*16 (zlaev2 の場合)  
                   入力行列の成分。

### 出力パラメータ

*rt1*, *rt2*      REAL (slaev2/claev2 の場合)  
                   DOUBLE PRECISION (dlaev2/zlaev2 の場合)  
                   それぞれ、大きい方および小さい方の値の固有値。

*cs1*            REAL (slaev2/claev2 の場合)  
                   DOUBLE PRECISION (dlaev2/zlaev2 の場合)

*sn1*            REAL (slaev2 の場合)  
                   DOUBLE PRECISION (dlaev2 の場合)  
                   COMPLEX (claev2 の場合)  
                   COMPLEX\*16 (zlaev2 の場合)  
                   ベクトル (*cs1*, *sn1*) は *rt1* に対する単位右固有ベクトル。

### アプリケーション・ノート

*rt1* はオーバーフロー/アンダーフローが起こらなければ、わずかな **ulp** (最小位置単位) に対して正確である。*rt2* は行列式  $a*c-b*b$  で、大幅な桁落ちがある場合に不正確となる可能性がある。*rt2* を精度高く計算するには、あらゆる場合で高精度または適切な丸めまたは適切な切捨て計算が必要になる。*cs1* と *sn1* はオーバーフロー/アンダーフローが起こらなければわずかな **ulp** に対して正確である。オーバーフローは、*rt1* がオーバーフローに係数 5 を掛けた値の範囲内にあるときに起こり得る。アンダーフローは入力データがゼロか *underflow\_threshold* / *macheps* を超えた場合には影響とはならない。

### ?laexc

*Schur* 標準形となっている実数の上準三角行列の隣接対角ブロックを、直交相似変換によって交換する。

---

#### 構文

```
call slaexc( wantq, n, t, ldt, q, ldq, j1, n1, n2, work, info )
call dlaexc( wantq, n, t, ldt, q, ldq, j1, n1, n2, work, info )
```

#### 説明

このルーチンは、上準三角行列  $T$  にある次数 1 または 2 の隣接対角ブロック  $T_{11}$  と  $T_{22}$  を、直交相似変換によって交換する。

$T$  は *Schur* 標準形でなければならない、すなわち、 $1 \times 1$  と  $2 \times 2$  の対角ブロックを持つブロック上三角である。それぞれの  $2 \times 2$  対角ブロックは、対角成分が等しく、かつ非対角成分が逆の符号を持つ。

#### 入力パラメータ

wantq	LOGICAL。 wantq = .TRUE. の場合、行列 $Q$ 内の変換を蓄積する。 wantq = .FALSE. の場合、変換を蓄積しない。
n	INTEGER。行列 $T$ の次数 ( $n \geq 0$ )。
t, q	REAL (slaexc の場合 ) DOUBLE PRECISION (dlaexc の場合 ) 配列 : $t(ldt, *)$ には、上準三角行列 $T$ を <i>Schur</i> 標準形で格納する。 $t$ の第 2 次元は $\max(1, n)$ 以上でなければならない。  wantq = .TRUE. の場合、 $q(ldq, *)$ には直交行列 $Q$ を格納する。 wantq = .FALSE. の場合、 $q$ は参照されない。 $q$ の第 2 次元は、 $\max(1, n)$ 以上でなければならない。
ldt	INTEGER。 $t$ の第 1 次元。 $\max(1, n)$ 以上。
ldq	INTEGER。 $q$ の第 1 次元。 wantq = .FALSE. の場合、 $ldq \geq 1$ wantq = .TRUE. の場合、 $ldq \geq \max(1, n)$
j1	INTEGER。 第 1 のブロック $T_{11}$ の第 1 行のインデックス。

*n1* INTEGER。第 1 のブロック  $T_{11}$  の次数 ( $n1 = 0, 1$ , または  $2$ )。  
*n2* INTEGER。第 2 のブロック  $T_{22}$  の次数 ( $n2 = 0, 1$ , または  $2$ )。  
*work* REAL (slaexc の場合 )  
 DOUBLE PRECISION (dlaexc の場合 )  
 ワークスペース配列、次元は ( $n$ )。

### 出力パラメータ

*t* schur 標準形を持つ更新された行列  $T$  で上書きされる。  
*q* `wantq = .TRUE.` の場合、更新された行列  $Q$  で上書きされる。  
*info* INTEGER。  
 $info = 0$  の場合、正常に終了したことを示す。  
 $info = 1$  の場合、変換された行列  $T$  は Schur 形式に遠すぎたことを示す。ブロックは交換されず、また  $T$  と  $Q$  は変更されない。

## ?lag2

$2 \times 2$  一般固有値問題の固有値を、オーバーフロー/アンダーフローが発生しないように必要に応じてスケーリングを行い計算する。

### 構文

```
call slag2( a, lda, b, ldb, safmin, scale1, scale2, wr1, wr2, wi )
call dlag2( a, lda, b, ldb, safmin, scale1, scale2, wr1, wr2, wi )
```

### 説明

このルーチンは、 $2 \times 2$  一般固有値問題  $A - wB$  の固有値を、オーバーフロー/アンダーフローが発生しないように必要に応じてスケーリングを行い計算する。スケール係数  $s$  によって固有値の式は次のように変更される。

$$sA - wB$$

$s$  は、 $w$ 、 $wB$ 、 $sA$  がオーバーフローを起こさないように、かつ可能ならばアンダーフローも起こさないように選ばれた非負のスケール係数である。

## 入力パラメータ

<i>a, b</i>	<p>REAL (slag2 の場合 )</p> <p>DOUBLE PRECISION (dlag2 の場合 )</p> <p>配列 :</p> <p><i>a</i>(<i>lda</i>, 2) には <math>2 \times 2</math> の行列 <i>A</i> を格納する。この 1- ノルムは <i>1/safmin</i> よりも小さいとする。sqrt(<i>safmin</i>)*norm(<i>A</i>) より小さい成分はゼロとして取り扱われる。</p> <p><i>b</i>(<i>ldb</i>, 2) には <math>2 \times 2</math> 上三角行列 <i>B</i> を格納する。この 1- ノルムは <i>1/safmin</i> よりも小さいとする。対角は <i>B</i> の (絶対値で) 最大成分の sqrt(<i>safmin</i>) 倍以上でなければならない。もし対角がこれよりも小さい場合は、対角が変わって +/- sqrt(<i>safmin</i>) が使用される。</p>
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 <i>lda</i> $\geq 2$
<i>ldb</i>	INTEGER。 <i>b</i> の第 1 次元。 <i>ldb</i> $\geq 2$
<i>safmin</i>	<p>REAL (slag2 の場合 )</p> <p>DOUBLE PRECISION (dlag2 の場合 )</p> <p><i>1/safmin</i> をオーバーフローさせない最も小さな正の値。(これは常に ?lamch('S') でなければならない。?lamch を頻繁に呼び出すのを避ける引数である。)</p>

## 出力パラメータ

<i>scale1</i>	<p>REAL (slag2 の場合 )</p> <p>DOUBLE PRECISION (dlag2 の場合 )</p> <p>第 1 の固有値を定義する固有値の式で、オーバーフロー/アンダーフローを防ぐために使用されたスケール係数。固有値が複素の場合、固有値は</p> <p><math>(wr1 \pm wi i) / scale1</math> ( マシンの指数範囲を超えるときがある ) で、<i>scale1</i>=<i>scale2</i> となり、<i>scale1</i> は常に正となる。</p> <p>固有値が実数の場合、第 1( 実数 ) の固有値は</p> <p><i>wr1 / scale1</i> となるが、これはオーバーフローかアンダーフローを起こすときがあり、実際、正確な固有値が十分に大きい場合、<i>scale1</i> はゼロかアンダーフローしきい値未満になることがある。</p>
<i>scale2</i>	<p>REAL (slag2 の場合 )</p> <p>DOUBLE PRECISION (dlag2 の場合 )</p> <p>第 2 の固有値を定義する固有値の式で、オーバーフロー/アンダーフローを防ぐために使用されたスケール係数。固有値が複素の場合、<i>scale1</i>=<i>scale2</i> となる。固有値が実数の場合、第 2( 実数 ) の固有値は</p>



$wr2/scale2$  となるが、これはオーバーフローかアンダーフローを起こすときがあり、実際、正確な固有値が十分に大きいと、 $scale2$  はゼロかアンダーフローしきい値未満になることがある。

$wr1$  REAL(slag2 の場合)  
DOUBLE PRECISION(dlag2 の場合)  
固有値が実数の場合、 $wr1$  は  $AB^{-1}$  の (2, 2) 成分に最も近い固有値の  $scale1$  倍となる。固有値が複素の場合、 $wr1$  は固有値の実数部の  $scale1$  倍となり  $wr1 = wr2$  である。

$wr2$  REAL(slag2 の場合)  
DOUBLE PRECISION(dlag2 の場合)  
固有値が実数の場合、 $wr2$  はその他の固有値の  $scale2$  倍になる。固有値が複素の場合、 $wr1$  は固有値の実数部の  $scale1$  倍となり  $wr1 = wr2$  である。

$wi$  REAL(slag2 の場合)  
DOUBLE PRECISION(dlag2 の場合)  
固有値が実数の場合、 $wi$  はゼロである。固有値が複素の場合、 $wi$  は固有値の虚数部の  $scale1$  倍となる。 $wi$  は常に非負である。

## ?lags2

$2 \times 2$  直交行列  $U$ 、 $V$ 、 $Q$  を計算し、変換された  $A$  と  $B$  の行が平行であるような  $A$  と  $B$  にそれらを適用する。

### 構文

```
call slags2( upper, a1, a2, a3, b1, b2, b3, csu, snu, csv, snv, csq, snq )
call dlags2( upper, a1, a2, a3, b1, b2, b3, csu, snu, csv, snv, csq, snq )
```

### 説明

このルーチンは  $2 \times 2$  直交行列  $U$ 、 $V$ 、 $Q$  を計算し、 $upper = .TRUE.$  であれば、

$$U' * A * Q = U' * \begin{bmatrix} A_1 & A_2 \\ 0 & A_3 \end{bmatrix} * Q = \begin{bmatrix} x & 0 \\ x & x \end{bmatrix}$$

および

$$V' * B * Q = V' * \begin{bmatrix} B_1 & B_2 \\ 0 & B_3 \end{bmatrix} * Q = \begin{bmatrix} x & 0 \\ x & x \end{bmatrix}$$

または `upper = .FALSE.` であれば、

$$U' * A * Q = U' * \begin{bmatrix} A_1 & 0 \\ A_2 & A_3 \end{bmatrix} * Q = \begin{bmatrix} x & x \\ 0 & x \end{bmatrix}$$

および

$$V' * B * Q = V' * \begin{bmatrix} B_1 & 0 \\ B_2 & B_3 \end{bmatrix} * Q = \begin{bmatrix} x & x \\ 0 & x \end{bmatrix}$$

変換された  $A$  と  $B$  の行は平行で、ここで

$$U = \begin{bmatrix} csu & snu \\ -snu & csu \end{bmatrix}, V = \begin{bmatrix} csv & snv \\ -snv & csv \end{bmatrix}, Q = \begin{bmatrix} csq & snq \\ -snq & csq \end{bmatrix}$$

$Z'$  は  $Z$  の転置を表わす。

## 入力パラメータ

<code>upper</code>	LOGICAL。 <code>upper = .TRUE.</code> の場合、入力行列 $A$ と $B$ は上三角である。 <code>upper = .FALSE.</code> の場合、入力行列 $A$ と $B$ は下三角である。
<code>a1, a2, a3</code>	REAL (slags2 の場合) DOUBLE PRECISION (dlags2 の場合) <code>a1, a2, a3</code> には入力 $2 \times 2$ 上 (下) 三角行列 $A$ の成分を格納する。
<code>b1, b2, b3</code>	REAL (slags2 の場合) DOUBLE PRECISION (dlags2 の場合) <code>b1, b2, b3</code> には入力 $2 \times 2$ 上 (下) 三角行列 $B$ の成分を格納する。

## 出力パラメータ

<i>csu, snu</i>	REAL (slags2 の場合 ) DOUBLE PRECISION (dlags2 の場合 ) 目的の直交行列 $U$ 。
<i>csv, snv</i>	REAL (slags2 の場合 ) DOUBLE PRECISION (dlags2 の場合 ) 目的の直交行列 $V$ 。
<i>csq, snq</i>	REAL (slags2 の場合 ) DOUBLE PRECISION (dlags2 の場合 ) 目的の直交行列 $Q$ 。

## ?lagtf

行交換を伴う部分ピボット演算を用いて行列  $T - \lambda I$  の LU 因子分解を計算する。 $T$  は一般三重対角行列、 $\lambda$  はスカラ。

### 構文

```
call slagtf( n, a, lambda, b, c, tol, d, in, info )
call dlagtf( n, a, lambda, b, c, tol, d, in, info )
```

### 説明

このルーチンは次のように行列  $(T - \lambda I)$  を因子分解する。 $T$  は  $n \times n$  三重対角行列、 $\lambda$  はスカラである。

$$T - \lambda I = P L U$$

$P$  は置換行列、 $L$  は非ゼロの劣対角成分を列あたり多くても 1 つしか持たない単位下三重対角行列、 $U$  は非ゼロの優対角成分を列あたり多くても 1 つしか持たない上三重対角行列である。因子分解は部分ピボット演算と黙示的な行スケーリングを用いたガウス消去によって実行される。 $T$  の固有ベクトルを逆反復によって取得するために、?lagtf と ?lagts が使用できるようにパラメータ  $\lambda$  はルーチンに格納されている。

## 入力パラメータ

$n$                     INTEGER。行列  $T$  の次数 ( $n \geq 0$ )。

*a*, *b*, *c*      REAL (slagtf の場合 )  
 DOUBLE PRECISION (dlagtf の場合 )  
 配列、次元は  $a(n)$ ,  $b(n-1)$ ,  $c(n-1)$ :  
*a*(\*) には行列  $T$  の対角成分を格納しなければならない。  
*b*(\*) には行列  $T$  の  $(n-1)$  個の優対角成分を格納しなければならない。  
*c*(\*) には行列  $T$  の  $(n-1)$  個の劣対角成分を格納しなければならない。

*tol*      REAL (slagtf の場合 )  
 DOUBLE PRECISION (dlagtf の場合 )  
 行列  $(T - \lambda I)$  がほとんど特異値かどうかを示すために使われる  
 相対許容値。*tol* は通常、 $T$  の成分におけるおよその最大相対誤差と  
 して選択される。例えば、 $T$  の成分がおよそ 4 桁の有効数字で正しい  
 場合、*tol* はおよそ  $5 \times 10^{-4}$  に設定しなければならない。*tol* が相対マ  
 シン精度 *eps* よりも小さく与えられた場合は *tol* の代わりに *eps* の値  
 が使われる。

## 出力パラメータ

*a*       $T$  を因子分解した上三角行列  $U$  の  $n$  個の対角成分で上書きされる。

*b*       $T$  を因子分解した行列  $U$  の  $n-1$  個の優対角成分で上書きされる。

*c*       $T$  を因子分解した行列  $L$  の  $n-1$  個の劣対角成分で上書きされる。

*d*      REAL (slagtf の場合 )  
 DOUBLE PRECISION (dlagtf の場合 )  
 配列、次元は  $(n-2)$ 。  
 $T$  を因子分解した行列  $U$  の  $n-2$  個の第 2 の優対角成分で上書きされ  
 る。

*in*      INTEGER。  
 配列、次元は  $(n)$ 。  
*in* には置換行列  $P$  の詳細が格納される。消去の  $k$  番目の過程で交換が  
 起こった場合は  $in(k) = 1$  となり、それ以外では  $in(k) = 0$  となる。成  
 分  $in(n)$  は以下を満たすような最小の正の整数  $j$  を返す。  

$$\text{abs}(u(j,j)) \leq \text{norm}((T - \lambda I)(j)) * \text{tol},$$
 $\text{norm}(A(j))$  は行列  $A$  の  $j$  行目の絶対値の総和を示す。そのような  $j$  が  
 存在しない場合は  $in(n)$  はゼロとして返される。*in*( $n$ ) が正で返された  
 場合、 $U$  の対角成分は小さく、 $(T - \lambda I)$  が特異値またはほとんど  
 特異値であることを示す。

*info*      INTEGER。  
*info* = 0 の場合、正常に終了したことを示す。  
*info* =  $-k$  の場合、 $k$  番目のパラメータの値が不正だったことを示す。

## ?lagtm

$C = \alpha AB + \beta C$  の形式で示される行列=行列積を  
実行し、ここで  $A$  は三重対角行列、 $B$  と  $C$  は矩形  
行列、 $\alpha$  と  $\beta$  はスカラで 0、1、または -1 である。

### 構文

```
call slagtm( trans, n, nrhs, alpha, dl, d, du, x, ldx, beta, b, ldb )
call dlagtm( trans, n, nrhs, alpha, dl, d, du, x, ldx, beta, b, ldb )
call clagtm( trans, n, nrhs, alpha, dl, d, du, x, ldx, beta, b, ldb )
call zlagtm( trans, n, nrhs, alpha, dl, d, du, x, ldx, beta, b, ldb )
```

### 説明

このルーチンは次の形式で示される行列 - ベクトル積を実行する。

$$B := \alpha A * X + \beta B$$

ここで  $A$  は次数  $n$  の三重対角行列、 $B$  と  $X$  は  $n \times nrhs$  行列、 $\alpha$  と  $\beta$  は実数のスカラで、それぞれ 0、1、または -1 である。

### 入力パラメータ

<i>trans</i>	CHARACTER*1。'N' または 'T' または 'C' でなければならない。 方程式の形式を指定する。 <i>trans</i> = 'N' の場合、 $B := \alpha A * X + \beta B$ (転置なし) <i>trans</i> = 'T' の場合、 $B := \alpha A^T * X + \beta B$ (転置) <i>trans</i> = 'C' の場合、 $B := \alpha A^H * X + \beta B$ (共役転置)
<i>n</i>	INTEGER。行列 $A$ の次数 ( $n \geq 0$ )。
<i>nrhs</i>	INTEGER。右辺の個数。すなわち、 $X$ と $B$ にある列の数 ( $nrhs \geq 0$ )。
<i>alpha, beta</i>	REAL (slagtm/clagtm の場合) DOUBLE PRECISION (dlagtm/zlagtm の場合) スカラ $\alpha$ と $\beta$ 。alpha は 0、1、または -1 のいずれかでなければならず、それ以外の場合は 0 とみなされる。beta は 0、1、または -1 のいずれかでなければならず、それ以外の場合は 1 とみなされる。

$dl, d, du$	<p>REAL (slagtm の場合 )</p> <p>DOUBLE PRECISION (dlagtm の場合 )</p> <p>COMPLEX (clagtm の場合 )</p> <p>COMPLEX*16 (zlagtm の場合 )</p> <p>配列: <math>dl(n-1), d(n), du(n-1)</math>。</p> <p>配列 <math>dl</math> には <math>T</math> の <math>(n-1)</math> 個の劣対角成分を格納する。</p> <p>配列 <math>d</math> には <math>T</math> の <math>n</math> 個の対角成分を格納する。</p> <p>配列 <math>du</math> には <math>T</math> の <math>(n-1)</math> 個の優対角成分を格納する。</p>
$x, b$	<p>REAL (slagtm の場合 )</p> <p>DOUBLE PRECISION (dlagtm の場合 )</p> <p>COMPLEX (clagtm の場合 )</p> <p>COMPLEX*16 (zlagtm の場合 )</p> <p>配列:</p> <p><math>x(ldx, *)</math> には <math>n \times nrhs</math> の行列 <math>X</math> を格納する。<math>x</math> の第 2 次元は <math>\max(1, nrhs)</math> 以上でなければならない。</p> <p><math>b(ldb, *)</math> には <math>n \times nrhs</math> の行列 <math>B</math> を格納する。<math>b</math> の第 2 次元は <math>\max(1, nrhs)</math> 以上でなければならない。</p>
$ldx$	INTEGER。配列 $x$ のリーディング・ディメンジョン。 $ldx \geq \max(1, n)$
$ldb$	INTEGER。配列 $b$ のリーディング・ディメンジョン。 $ldb \geq \max(1, n)$

## 出力パラメータ

$b$	<p>次の行列式で上書きされる。</p> $B := \alpha * A * X + \beta * B$
-----	--

---

## ?lagts

?lagtf で計算された LU 因子分解を用いて連立方程式  $(T - \lambda I)x = y$  または  $(T - \lambda I)^T x = y$  を解く。ここで  $T$  は一般三重対角行列、 $\lambda$  はスカラー。

---

### 構文

```
call slagts( job, n, a, b, c, d, in, y, tol, info )
call dlagts( job, n, a, b, c, d, in, y, tol, info )
```

## 説明

このルーチンは次のどちらかの連立方程式を  $x$  について解くために使用される。

$(T - \lambda I)x = y$  または  $(T - \lambda I)'x = y$ 、  
ここで  $T$  は  $n \times n$  の三重対角行列で、ルーチン [?lagtf](#) で、 $T - \lambda I = PLU$  として  
計算された  $(T - \lambda I)$  の因子分解から続く。

解く式は引数  $job$  によって選択し、どちらの場合もゼロへの摂動か  $U$  のきわめて小さい対角成分のオプションがあり、このオプションは逆反復のようなアプリケーションでの使用を意図したものである。

## 入力パラメータ

**job** INTEGER。?lagts で実行される内容を次の中から指定する。  
 = 1 の場合、式  $(T - \lambda I)x = y$  を解くが  $U$  の対角成分を摂動させない。  
 = -1 の場合、式  $(T - \lambda I)x = y$  を解き、かつ、オーバーフローが起きるなら  $U$  の対角成分を摂動させる。下記の引数  $tol$  を参照のこと。  
 = 2 の場合、式  $(T - \lambda I)'x = y$  を解くが  $U$  の対角成分を摂動させない。  
 = -2 の場合、 $(T - \lambda I)'x = y$  を解き、かつ、オーバーフローが起きるなら  $U$  の対角成分を摂動させる。下記の引数  $tol$  を参照のこと。

**n** INTEGER。行列  $T$  の次数 ( $n \geq 0$ )。

**a, b, c, d** REAL (slagts の場合)  
 DOUBLE PRECISION (dlagts の場合)  
 配列、次元は  $a(n)$ 、 $b(n-1)$ 、 $c(n-1)$ 、 $d(n-2)$ ：  
 $a(*)$  には、?lagtf から返されたとおり、 $U$  の対角成分を格納しなければならない。  
 $b(*)$  には、?lagtf から返されたとおり、 $U$  の第 1 優対角成分を格納しなければならない。  
 $c(*)$  には、?lagtf から返されたとおり、 $L$  の劣対角成分を格納しなければならない。  
 $d(*)$  には、?lagtf から返されたとおり、 $U$  の第 2 優対角成分を格納しなければならない。

**in** INTEGER。  
 配列、次元は  $(n)$ 。  
 $in(*)$  には、?lagtf から返されたとおり、行列  $P$  の詳細を格納しなければならない。

<i>y</i>	REAL (slagts の場合 ) DOUBLE PRECISION (dlagts の場合 ) 配列、次元は ( <i>n</i> )。ベクトル <i>y</i> の右辺である。
<i>tol</i>	REAL (slagtf の場合 )DOUBLE PRECISION (dlagtf の場合 ) <i>job</i> < 0 の場合、 <i>tol</i> は、 <i>U</i> のきわめて小さな対角成分に対して作られる最小摂動でなければならない。 <i>tol</i> は通常およそ <i>eps</i> *norm( <i>U</i> ) として選ばれ、ここで <i>eps</i> は相対マシン精度であるが、 <i>tol</i> が 0 未満として与えられた場合は <i>eps</i> *max(abs( <i>u</i> ( <i>i</i> , <i>j</i> ))) に設定しなおされる。 <i>job</i> > 0 の場合、 <i>tol</i> は参照されない。

### 出力パラメータ

<i>y</i>	<i>y</i> は解ベクトル <i>x</i> によって上書きされる。
<i>tol</i>	上記入力パラメータのセクションで説明したとおり、 <i>tol</i> が入力において 0 未満の場合のみ <i>tol</i> は変更される。それ以外では <i>tol</i> は変更されない。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正であったことを示す。 <i>info</i> = <i>i</i> > 0 の場合、解ベクトル <i>x</i> の <i>i</i> 番目の成分の計算でオーバーフローが起こったことを示す。これは <i>job</i> が正として与えられ、 <i>U</i> の対角成分がきわめて小さいかベクトル <i>y</i> の右辺の成分がきわめて大きい場合にのみ起こる

---

## ?lagv2

実数の  $2 \times 2$  行列束 (*A*, *B*) の汎用 Schur 因子分解を計算する。ここで *B* は上三角である。

---

### 構文

```
call slagv2( a, lda, b, ldb, alphas, alphai, beta, csl, snl, csr, snr )  
call dlagv2( a, lda, b, ldb, alphas, alphai, beta, csl, snl, csr, snr )
```



## 説明

このルーチンは  $2 \times 2$  行列束  $(A, B)$  の汎用 Schur 因子分解を計算する。ここで  $B$  は上三角である。ルーチンは、次を満たすような  $cs1, sn1$  と  $csr, snr$  で与えられる直交 (回転) 行列を計算する。

1) 束  $(A, B)$  が 2 個の実数固有値 (0/0 または 1/0 タイプを含む) を持つ場合、

$$\begin{bmatrix} a_{11} & a_{12} \\ 0 & a_{22} \end{bmatrix} = \begin{bmatrix} cs1 & sn1 \\ -sn1 & cs1 \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} csr & -snr \\ snr & csr \end{bmatrix}$$

$$\begin{bmatrix} b_{11} & b_{12} \\ 0 & b_{22} \end{bmatrix} = \begin{bmatrix} cs1 & sn1 \\ -sn1 & cs1 \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ 0 & b_{22} \end{bmatrix} \begin{bmatrix} csr & -snr \\ snr & csr \end{bmatrix}$$

2) 束  $(A, B)$  が複素共役固有値のペアを持つ場合、

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = \begin{bmatrix} cs1 & sn1 \\ -sn1 & cs1 \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} csr & -snr \\ snr & csr \end{bmatrix}$$

$$\begin{bmatrix} b_{11} & 0 \\ 0 & b_{22} \end{bmatrix} = \begin{bmatrix} cs1 & sn1 \\ -sn1 & cs1 \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ 0 & b_{22} \end{bmatrix} \begin{bmatrix} csr & -snr \\ snr & csr \end{bmatrix}$$

ここで  $b_{11} \geq b_{22} > 0$ 。

## 入力パラメータ

$a, b$	REAL (slagv2 の場合) DOUBLE PRECISION (dlagv2 の場合) 配列: $a(lda, 2)$ には $2 \times 2$ 行列 $A$ を格納する。 $b(l db, 2)$ には上三角 $2 \times 2$ 行列 $B$ を格納する。
$lda$	INTEGER。配列 $a$ のリーディング・ディメンジョン。 $lda \geq 2$
$ldb$	INTEGER。配列 $b$ のリーディング・ディメンジョン。 $ldb \geq 2$

### 出力パラメータ

<i>a</i>	<i>a</i> は汎用 Schur 形式の "A- 部分 " で上書きされる。
<i>b</i>	<i>b</i> は汎用 Schur 形式の "B- 部分 " で上書きされる。
<i>alphar, alphai,</i> <i>beta</i>	REAL (slagv2 の場合 ) DOUBLE PRECISION (dlagv2 の場合 ) 配列、次元はそれぞれ、(2)。  $(\text{alphar}(k) + i * \text{alphai}(k)) / \text{beta}(k)$ は束 ( <i>A, B</i> ) の固有値で、 <i>k</i> =1,2 および $i = \sqrt{-1}$ である。 <i>beta</i> ( <i>k</i> ) はゼロとなることがある点に注意する。
<i>cs1, sn1</i>	REAL (slagv2 の場合 ) DOUBLE PRECISION (dlagv2 の場合 ) それぞれ、左回転行列の余弦と正弦である。
<i>csr, snr</i>	REAL (slagv2 の場合 ) DOUBLE PRECISION (dlagv2 の場合 ) それぞれ、右回転行列の余弦と正弦である。

---

## ?lahqr

ダブルシフト / シングルシフト *QR* アルゴリズム  
を用いて、上 Hessenberg 行列の固有値と Schur 因子分解を計算する。

---

### 構文

```
call slahqr( wantt, wantz, n, ilo, ihi, h, ldh, wr, wi, iloz, ihiz, z, ldz,  
            info )  
call dlahqr( wantt, wantz, n, ilo, ihi, h, ldh, wr, wi, iloz, ihiz, z, ldz,  
            info )  
call clahqr( wantt, wantz, n, ilo, ihi, h, ldh, w, iloz, ihiz, z, ldz, info )  
call zlahqr( wantt, wantz, n, ilo, ihi, h, ldh, w, iloz, ihiz, z, ldz, info )
```

### 説明

このルーチンは [?hseqr](#) から呼び出される補助ルーチンで、*ilo* から *ihi* の行と列にある Hessenberg 部分行列を対象として、?hseqr によって過去に計算された固有値と Schur 分解を更新する。

## 入力パラメータ

<code>wantt</code>	LOGICAL。 <code>wantt = .TRUE.</code> の場合、すべての Schur 形式 $T$ が必要である。 <code>wantt = .FALSE.</code> の場合、固有値のみが必要である。
<code>wantz</code>	LOGICAL。 <code>wantz = .TRUE.</code> の場合、Schur ベクトル $Z$ の行列が必要である。 <code>wantz = .FALSE.</code> の場合、Schur ベクトルは必要ない。
<code>n</code>	INTEGER。行列 $H$ の次数 ( $n \geq 0$ )。
<code>ilo, ihi</code>	INTEGER。 $H$ は、行と列が $ihi+1:n$ の上準三角になっているとする。また、 $H(ilo, ilo-1) = 0$ ( $ilo = 1$ でない場合) であるとする。ルーチン <code>?lahqr</code> は基本的に行と列が $ilo$ から $ihi$ の Hessenberg 部分行列を対象とするが、 <code>wantt = .TRUE.</code> の場合は $H$ のすべてに変換が適用される。 制約: $1 \leq ilo \leq \max(1, ihi); ihi \leq n$
<code>h, z</code>	REAL ( <code>slahqr</code> の場合) DOUBLE PRECISION ( <code>dlahqr</code> の場合) COMPLEX ( <code>clahqr</code> の場合) COMPLEX*16 ( <code>zlahqr</code> の場合) 配列: $h(ldh, *)$ には上 Hessenberg 行列 $H$ が格納される。 $h$ の第 2 次元は $\max(1, n)$ 以上でなければならない。 $z(ldz, *)$ <code>wantz = .TRUE.</code> の場合、 $z$ には <code>?hseqr</code> で蓄積された変換の現在の行列 $Z$ を格納しなければならない。 <code>wantz = .FALSE.</code> の場合、 $z$ は参照されない。 $z$ の第 2 次元は、 $\max(1, n)$ 以上でなければならない。
<code>ldh</code>	INTEGER。 $h$ の第 1 次元。 $\max(1, n)$ 以上。
<code>ldz</code>	INTEGER。 $z$ の第 1 次元。 $\max(1, n)$ 以上であること。
<code>iloz, ihiz</code>	INTEGER。 <code>wantz = .TRUE.</code> の場合、変換が適用されなければならない $Z$ の行を指定する。 $1 \leq iloz \leq ilo; ihi \leq ihiz \leq n$ 。

## 出力パラメータ

<i>h</i>	<i>wantt</i> = .TRUE. の場合、 <i>H</i> は行と列が <i>ilo:ihi</i> にある上準三角 (複素型では上三角) で、標準形式の $2 \times 2$ 対角ブロックを持つ。 <i>wantt</i> = .FALSE. の場合、 <i>H</i> の内容は規定されない。
<i>wr, wi</i>	REAL ( <i>slahqr</i> の場合) DOUBLE PRECISION ( <i>dlahqr</i> の場合) 配列、次元はそれぞれ $\max(1, n)$ 以上。実数型でのみ使用される。 計算された固有値 <i>ilo</i> から <i>ihi</i> の実数部分と虚数部分はそれぞれ <i>wr</i> と <i>wi</i> の対応する成分に格納される。2 個の固有値が複素共役ペアとして計算された場合は、それらは <i>wr</i> と <i>wi</i> の連続する成分に格納され、すなわち <i>i</i> 番目と ( <i>i</i> +1) 番目で $wi(i) > 0$ かつ $wi(i+1) < 0$ である。 <i>wantt</i> = .TRUE. の場合、固有値は <i>H</i> に返された Schur 形式の対角と同じ順序で格納され、 $wr(i) = H(i, i)$ 、また $2 \times 2$ 対角ブロックでは $H(i:i+1, i:i+1)$ で、 $wi(i) = \sqrt{H(i+1, i) * H(i, i+1)}$ および $wi(i+1) = -wi(i)$ である。
<i>w</i>	COMPLEX ( <i>clahqr</i> の場合) COMPLEX*16 ( <i>zlahqr</i> の場合) 配列、次元は $\max(1, n)$ 以上。複素型でのみ使用される。計算された固有値 <i>ilo</i> から <i>ihi</i> は対応する <i>w</i> の成分に格納される。 <i>wantt</i> = .TRUE. の場合、固有値は <i>H</i> に返された Schur 形式の対角と同じ順序で格納され、 $w(i) = H(i, i)$ 。
<i>z</i>	<i>wantz</i> = .TRUE. の場合、 <i>z</i> は更新されている。変換は部分行列 $Z(ilo:ihiz, ilo:ihi)$ にのみ適用されている。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = <i>i</i> > 0 の場合、 <i>slahqr</i> はすべての固有値 <i>ilo</i> から <i>ihi</i> を、合計 $30 * (ihi - ilo + 1)$ 回以内の反復で計算できなかったことを示す。 <i>wr</i> と <i>wi</i> ( <i>slahqr/dlahqr</i> の場合) または <i>w</i> ( <i>clahqr/zlahqr</i> の場合) の成分 <i>i+1:ihi</i> には、正常に終了したそれら固有値が格納される。

## ?lahrd

$k$  番目の劣対角よりも下の成分がゼロになるように一般矩形行列の最初の  $nb$  列を縮退させ、 $A$  の縮退されていない部分に変換を適用するために必要となる補助行列を返す。

### 構文

```
call slahrd( n, k, nb, a, lda, tau, t, ldt, y, ldy )
call dlahrd( n, k, nb, a, lda, tau, t, ldt, y, ldy )
call clahrd( n, k, nb, a, lda, tau, t, ldt, y, ldy )
call zlahrd( n, k, nb, a, lda, tau, t, ldt, y, ldy )
```

### 説明

このルーチンは、 $k$  番目の劣対角よりも下の成分がゼロになるように、実数 / 複素一般  $n \times (n-k+1)$  行列  $A$  の最初の  $nb$  列を縮退させる。この縮退は直交 / ユニタリ相似変換  $Q'AQ$  によって実行される。ルーチンは、ブロック・リフレクタ  $I - VTV'$  として  $Q$  を定義する行列  $V$  と  $T$ 、および行列  $Y = AVT$  を返す。

行列  $Q$  は  $nb$  個の基本リフレクタの積として表現される。  
 $Q = H(1)H(2) \dots H(nb)$

それぞれの  $H(i)$  は次のような形式を持つ。

$$H(i) = I - \tau v v'$$

$\tau$  は実数 / 複素スカラー、 $v$  は実数 / 複素の  $(n-1)$  成分で構成されるベクトルである。

このルーチンは [?gehrd](#) から呼び出される補助ルーチンである。

### 入力パラメータ

$n$	INTEGER。行列 $A$ の次数 ( $n \geq 0$ )。
$k$	INTEGER。縮退に対するオフセット。最初の $nb$ 列にある $k$ 番目の劣対角よりも下の成分がゼロに縮退される。
$nb$	INTEGER。縮退させる列の数。

$a$	REAL (slahrd の場合 ) DOUBLE PRECISION (dlahrd の場合 ) COMPLEX (clahrd の場合 ) COMPLEX*16 (zlahrd の場合 )  $a(lda, n-k+1)$ には縮退対象の $n \times (n-k+1)$ 一般行列 $A$ を格納する。
$lda$	INTEGER。 $a$ の第 1 次元。 $\max(1, n)$ 以上でなければならない。
$ldt$	INTEGER。 出力配列 $t$ の第 1 次元のサイズ。 $\max(1, nb)$ 以上でなければならない。
$ldy$	INTEGER。 出力配列 $y$ の第 1 次元のサイズ。 $\max(1, n)$ 以上でなければならない。

## 出力パラメータ

$a$	最初の $nb$ 列にある $k$ 番目の劣対角成分とその上の成分は縮退された行列の対応する成分で上書きされる。 $k$ 番目の劣対角よりも下の成分は配列 $\tau$ で上書きされ、基本リフレクタの積として行列 $Q$ を表現する。そのほかの列は変更されない。 次のアプリケーション・ノートを参照。
$\tau$	REAL (slahrd の場合 ) DOUBLE PRECISION (dlahrd の場合 ) COMPLEX (clahrd の場合 ) COMPLEX*16 (zlahrd の場合 )  配列、次元は $(nb)$ 。 基本リフレクタのスカラー係数が入る。
$t, y$	REAL (slahrd の場合 ) DOUBLE PRECISION (dlahrd の場合 ) COMPLEX (clahrd の場合 ) COMPLEX*16 (zlahrd の場合 )  配列、次元は $t(ldt, nb), y(ldy, nb)$ 。 配列 $t$ には上三角行列 $T$ が格納される。  配列 $y$ には $n \times nb$ の行列 $Y$ が格納される。

## アプリケーション・ノート

基本リフレクタ  $H(i)$  に対して、

ルーチン終了時に  $v(1:i+k-1) = 0$ ,  $v(i+k) = 1$ ;  $v(i+k+1:n)$  は  $a(i+k+1:n, i)$  に格納され、 $\tau$  は  $\tau(i)$  に格納される。

ベクトル  $v$  の成分は、行列の縮退されていない部分に変換を適用するために、次の形式の更新を用いて、 $T$  と  $Y$  とともに必要な  $(n-k+1) \times nb$  の行列  $V$  を形成する。

$$A := (I - V T V') * (A - Y V')$$

$n = 7$ ,  $k = 3$ ,  $nb = 2$  におけるルーチン終了時の  $A$  の内容を以下に示す。

$$\begin{bmatrix} a & h & a & a & a \\ a & h & a & a & a \\ a & h & a & a & a \\ h & h & a & a & a \\ v_1 & h & a & a & a \\ v_1 & v_2 & a & a & a \\ v_1 & v_2 & a & a & a \end{bmatrix}$$

$a$  は元の行列  $A$  の成分、 $h$  は上 Hessenberg 行列  $H$  の変更された成分、 $v_i$  は  $H(i)$  を定義するベクトルの成分を示す。

## ?laic1

増加条件推定を 1 ステップ適用する。

### 構文

```
call slaic1( job, j, x, sest, w, gamma, sestpr, s, c )
call dlaic1( job, j, x, sest, w, gamma, sestpr, s, c )
call claic1( job, j, x, sest, w, gamma, sestpr, s, c )
call zlaic1( job, j, x, sest, w, gamma, sestpr, s, c )
```

### 説明

ルーチン ?laic1 は最も簡単な形で増加条件推定を 1 ステップ適用する。

$\|x\|_2 = 1$  ( $\|a\|_2$  とは  $a$  の 2- ノルムを示す) の  $x$  を、次を満たすような  $j \times j$  の下三角行列  $L$  のおおよその特異ベクトルとする。

$$\|L^*x\|_2 = sest$$

次に ?laic1 は、おおよその特異ベクトルを次の意味で満たす  $sestpr$ 、 $s$ 、 $c$  を計算する。

$$xhat = \begin{bmatrix} s^*x \\ c \end{bmatrix}$$

$$Lhat = \begin{bmatrix} L & 0 \\ w' & gamma \end{bmatrix}$$

$$\|Lhat^*xhat\|_2 = sestpr$$

$job$  によって最大または最小特異値に対する推定が計算される。

$[s \ c]'$  と  $sestpr^2$  は、次の連立方程式の固有ペアである点に注意する (slaic1/ciaic の場合)

$$\text{diag}(sest^*sest, 0) + \begin{bmatrix} alpha & gamma \end{bmatrix} * \begin{bmatrix} alpha \\ gamma \end{bmatrix}$$

ここで、 $alpha = x'^*w$ 、

または、次の連立方程式の固有ペアである (claic1/zlaic の場合)

$$\text{diag}(sest^*sest, 0) + \begin{bmatrix} alpha & gamma \end{bmatrix} * \begin{bmatrix} \text{conjg}(alpha) \\ \text{conjg}(gamma) \end{bmatrix}$$

ここで、 $alpha = \text{conjg}(x)'*w$



## 入力パラメータ

<i>job</i>	INTEGER。 <i>job</i> = 1 の場合、最大特異値に対する推定が計算される。 <i>job</i> = 2 の場合、最小特異値に対する推定が計算される。
<i>j</i>	INTEGER。 <i>x</i> と <i>w</i> の長さ。
<i>x</i> , <i>w</i>	REAL ( <i>slaic1</i> の場合) DOUBLE PRECISION ( <i>dlaic1</i> の場合) COMPLEX ( <i>claic1</i> の場合) COMPLEX*16 ( <i>zlaic1</i> の場合) 配列、次元はそれぞれ、( <i>j</i> ) それぞれ、ベクトル <i>x</i> と <i>w</i> を格納する。
<i>sest</i>	REAL ( <i>slaic1/claic1</i> の場合) DOUBLE PRECISION ( <i>dlaic1/zlaic1</i> の場合) <i>j</i> × <i>j</i> の行列 <i>L</i> の推定された特異値。
<i>gamma</i>	REAL ( <i>slaic1</i> の場合) DOUBLE PRECISION ( <i>dlaic1</i> の場合) COMPLEX ( <i>claic1</i> の場合) COMPLEX*16 ( <i>zlaic1</i> の場合) 対角成分 <i>gamma</i>

## 出力パラメータ

<i>sestpr</i>	REAL ( <i>slaic1/claic1</i> の場合) DOUBLE PRECISION ( <i>dlaic1/zlaic1</i> の場合) ( <i>j</i> +1) × ( <i>j</i> +1) の行列 <i>Lhat</i> の推定された特異値。
<i>s</i> , <i>c</i>	REAL ( <i>slaic1</i> の場合) DOUBLE PRECISION ( <i>dlaic1</i> の場合) COMPLEX ( <i>claic1</i> の場合) COMPLEX*16 ( <i>zlaic1</i> の場合) <i>xhat</i> の構成に必要な正弦と余弦。

## ?laln2

指定された形式の  $1 \times 1$  または  $2 \times 2$  の線形連立方程式を解く。

### 構文

```
call slaln2( ltrans, na, nw, smin, ca, a, lda, d1, d2, b, ldb, wr, wi, x, ldx,  
            scale, xnorm, info )  
call dlaln2( ltrans, na, nw, smin, ca, a, lda, d1, d2, b, ldb, wr, wi, x, ldx,  
            scale, xnorm, info )
```

### 説明

このルーチンは次の形式の連立方程式を、可能なスケーリング ( $s$ ) と  $A$  の摂動 ( $A'$  は  $A$  の転置) を使って解く。

$$(ca A - w D) X = s B \quad \text{または} \quad (ca A' - w D) X = s B$$

$A$  は  $na \times na$  の実数行列、 $ca$  は実数スカラ、 $D$  は  $na \times na$  の実数対角行列、 $w$  は実数または複素値、 $X$  と  $B$  は  $na \times 1$  の行列で  $w$  が実数の場合は実数、 $w$  が複素の場合は複素である。パラメータ  $na$  は 1 か 2 である。

$w$  が複素の場合、 $X$  と  $B$  は  $na \times 2$  の行列として表現され、それぞれの第 1 の列が実数部、第 2 の列が虚数部となる。

このルーチンは  $X$  をオーバーフローせずに計算できるようにスケール係数  $s (\leq 1)$  を選択する。 $X$  は、 $\text{norm}(ca A - w D) * \text{norm}(X)$  がオーバーフロー未満になるように、必要に応じてさらにスケーリングされる。

$(ca A - w D)$  の両方の特異値が  $smin$  よりも小さい場合、 $smin * I$  (ここで  $I$  は単位を表わす) が  $(ca A - w D)$  の代わりに使用される。1 つの固有値が  $smin$  よりも小さい場合は、最小特異値がおおよそ  $smin$  になるように  $(ca A - w D)$  の 1 つの成分が摂動される。両方の特異値が  $smin$  以上ならば、 $(ca A - w D)$  は摂動されない。

どのような場合でも、摂動は、大きくとも  $\max(smin, ulp * \text{norm}(ca A - w D))$  の小さな倍

数となる。  
特異値は無限ノルム近似によって計算され、そのため、2 程度の係数に対してのみ正確となる。



**注:** 入力の大きさは、妥当な係数によってオーバーフローよりも小さいと仮定される (*bignum* を参照)。

## 入力パラメータ

<i>trans</i>	LOGICAL。 <i>trans</i> = .TRUE. の場合、A- 転置が使用される。 <i>trans</i> = .FALSE. の場合、A が使用される (転置なし)。
<i>na</i>	INTEGER。行列 A のサイズである。1 か 2 のみを取り得る。
<i>nw</i>	INTEGER。 <i>w</i> が実数の場合、このパラメータは 1、 <i>w</i> が複素数の場合、パラメータは 2 でなければならない。1 か 2 のみを取り得る。
<i>smin</i>	REAL (slaln2 の場合) DOUBLE PRECISION (dlaln2 の場合) A の特異値における目的の下限。これはアンダーフローまたはオーバーフローに対して安全な距離を持っていなければならない。例えば、( <i>underflow/machine_precision</i> ) と ( <i>machine_precision * overflow</i> ) の間とする。( <i>bignum</i> と <i>ulp</i> を参照)
<i>ca</i>	REAL (slaln2 の場合) DOUBLE PRECISION (dlaln2 の場合) A に乗算される係数。
<i>a</i>	REAL (slaln2 の場合) DOUBLE PRECISION (dlaln2 の場合) 配列、次元は ( <i>lda</i> , <i>na</i> )。 $na \times na$ の行列 A
<i>lda</i>	INTEGER。 <i>a</i> のリーディング・ディメンジョン。 <i>na</i> 以上でなければならない。
<i>d1</i> , <i>d2</i>	REAL (slaln2 の場合) DOUBLE PRECISION (dlaln2 の場合) それぞれ対角行列 D にある (1, 1) と (2, 2) の成分。 <i>nw</i> = 1 の場合は <i>d2</i> は使用されない。

<i>b</i>	REAL (slaln2 の場合 ) DOUBLE PRECISION (dlaln2 の場合 ) 配列、次元は ( <i>ldb</i> , <i>nw</i> )。 $na \times nw$ の行列 <i>B</i> ( 右辺 )。 $nw = 2$ ( <i>w</i> が複素 ) の場合、列 1 には <i>B</i> の実数部分を格納し、列 2 には虚数部分を格納する。
<i>ldb</i>	INTEGER。 <i>b</i> のリーディング・ディメンジョン。 <i>na</i> 以上でなければならない。
<i>wr</i> , <i>wi</i>	REAL (slaln2 の場合 ) DOUBLE PRECISION (dlaln2 の場合 ) それぞれスカラ <i>w</i> の実数部と虚数部。 $nw = 1$ の場合は <i>wi</i> は参照されない。
<i>ldx</i>	INTEGER。 出力配列 <i>x</i> のリーディング・ディメンジョン。 <i>na</i> 以上でなければならない。

## 出力パラメータ

<i>x</i>	REAL (slaln2 の場合 ) DOUBLE PRECISION (dlaln2 の場合 ) 配列、次元は ( <i>ldx</i> , <i>nw</i> )。 このルーチンによって計算された $na \times nw$ の行列 <i>X</i> ( 未知 )。 $nw = 2$ の場合 ( <i>w</i> が複素 )、列 1 には <i>X</i> の実数部が格納され、列 2 には虚数部が格納される。
<i>scale</i>	REAL (slaln2 の場合 ) DOUBLE PRECISION (dlaln2 の場合 ) <i>X</i> の計算でオーバーフローを起こさないように <i>B</i> に乗算されるべきスケール係数。 そのため、( <i>ca A</i> - <i>w D</i> ) <i>X</i> は <i>B</i> ではなく <i>scale*B</i> となる ( <i>A</i> の摂動は無視 )。 大きくとも 1 である。
<i>xnorm</i>	REAL (slaln2 の場合 ) DOUBLE PRECISION (dlaln2 の場合 ) <i>X</i> が $na \times nw$ の実数行列としてみなされる場合、 <i>X</i> の無限ノルム。
<i>info</i>	INTEGER。 エラーフラグ。 エラーが生じなかった場合はゼロに、引数にエラーがあった場合は負に、( <i>ca A</i> - <i>w D</i> ) を摂動しなければならない場合は正になる。 取り得る値は次のとおりである。 <i>info</i> = 0 の場合、正常に終了したことを示し、

( $ca A - w D$ ) を摂動する必要がなかった。  
 $info = 1$  の場合、最小 (または唯一の) 特異値を  $smin$  よりも大きくするために、( $ca A - w D$ ) を摂動する必要があったことを示す。



**注:** 実行速度を考慮してこのルーチンでは入力のエラーをチェックしない。

## ?lals0

最小二乗問題を分割統治 SVD 法を使用して解く過程で乗率を逆に適用する。`?gelsd` で使用される。

### 構文

```
call slals0( icompg, nl, nr, sqre, nrhs, b, ldb, bx, ldbx, perm,
             givptr, givcol, ldgcol, givnum, ldgnum, poles, difl, difr, z,
             k, c, s, work, info )
call dlals0( icompg, nl, nr, sqre, nrhs, b, ldb, bx, ldbx, perm,
             givptr, givcol, ldgcol, givnum, ldgnum, poles, difl, difr, z,
             k, c, s, work, info )
call clals0( icompg, nl, nr, sqre, nrhs, b, ldb, bx, ldbx, perm,
             givptr, givcol, ldgcol, givnum, ldgnum, poles, difl, difr, z,
             k, c, s, rwork, info )
call zlals0( icompg, nl, nr, sqre, nrhs, b, ldb, bx, ldbx, perm,
             givptr, givcol, ldgcol, givnum, ldgnum, poles, difl, difr, z,
             k, c, s, rwork, info )
```

### 説明

このルーチンは、分割統治 SVD 法を使用して最小二乗問題を解く過程で、右辺行列  $B$  に行を追加した対角行列の左または右特異ベクトル行列のいずれかの乗率を逆に適用する。

左特異ベクトル行列において、3 タイプの直交行列が関係する。

(1L) Givens 回転。回転の回数は `givptr` に格納する。回転が適用された列 / 行のペアは `givcol` に格納する。これら回転の  $c$  値と  $s$  値は `givnum` に格納する。

(2L) 置換。  $B$  の  $(n1+1)$  番目の行が第 1 の行に移動される行であり、  $j = 2:n$  において、  $B$  の  $perm(j)$  番目の行が  $j$  番目の行に移動される行である。

(3L) 残りの行列の左特異ベクトル行列。

右特異ベクトル行列において、 4 タイプの直交行列が関係する。

(1R) 残りの行列の右特異ベクトル行列。

(2R)  $scre = 1$  の場合、 右ヌル空間を生成するために 1 回の追加 Givens 回転。

(3R) (2L) の逆変換。

(4R) (1L) の逆変換。

## 入力パラメータ

<i>icompq</i>	INTEGER。 特異ベクトルを因子分解された形式で計算するかを指定する。 <i>icompq</i> = 0 の場合、 左特異ベクトル行列。 <i>icompq</i> = 1 の場合、 右特異ベクトル行列。
<i>n1</i>	INTEGER。 上ブロックの行次元。 $n1 \geq 1$
<i>nr</i>	INTEGER。 下ブロックの行次元。 $nr \geq 1$
<i>scre</i>	INTEGER。 <i>scre</i> = 0 の場合、 下ブロックは $nr \times nr$ の正方行列。 <i>scre</i> = 1 の場合、 下ブロックは $nr \times (nr+1)$ の矩形行列。 二重対角行列は行次元 $n = n1 + nr + 1$ 、 列次元 $m = n + scre$ を持つ。
<i>nrhs</i>	INTEGER。 $b$ と $bx$ の列の数。 1 以上でなければならない。
<i>b</i>	REAL (slals0 の場合 ) DOUBLE PRECISION (dlals0 の場合 ) COMPLEX (clals0 の場合 ) COMPLEX*16 (zlals0 の場合 ) 配列、 次元は $(ldb, nrhs)$ 。 最小二乗問題の右边を行 1 から $m$ に格納する。
<i>ldb</i>	INTEGER。 $b$ のリーディング・ディメンジョン。 $\max(1, \max(m, n))$ 以上でなければならない。

<i>bx</i>	REAL (slals0 の場合) DOUBLE PRECISION (dlals0 の場合) COMPLEX (clals0 の場合) COMPLEX*16 (zlals0 の場合) ワークスペース配列、次元は ( <i>ldbx</i> , <i>nrhs</i> )。
<i>ldbx</i>	INTEGER。 <i>bx</i> のリーディング・ディメンジョン。
<i>perm</i>	INTEGER。 配列、次元は ( <i>n</i> )。2 個のブロックに適用される (収縮とソートによる) 置換。
<i>givptr</i>	INTEGER。この部分問題で実行された Givens 回転の回数が格納される。
<i>givcol</i>	INTEGER。配列、次元は ( <i>ldgcol</i> , 2)。Givens 回転に関与した行 / 列のペアを示す数値のペア。
<i>ldgcol</i>	INTEGER。 <i>givcol</i> のリーディング・ディメンジョン。 <i>n</i> 以上でなければならない。
<i>givnum</i>	REAL (slals0 / clals0 の場合) DOUBLE PRECISION (dlals0 / zlals0 の場合) 配列、次元は ( <i>ldgnum</i> , 2)。対応する Givens 回転で使用された <i>c</i> 値または <i>s</i> 値を示す数。
<i>ldgnum</i>	INTEGER。配列 <i>difl</i> 、 <i>poles</i> 、 <i>givnum</i> のリーディング・ディメンジョン。 <i>k</i> 以上でなければならない。
<i>poles</i>	REAL (slals0 / clals0 の場合) DOUBLE PRECISION (dlals0 / zlals0 の場合) 配列、次元は ( <i>ldgnum</i> , 2)。 <i>poles</i> (1: <i>k</i> , 1) には永年方程式を解いて得られた新しい特異値を格納し、 <i>poles</i> (1: <i>k</i> , 2) には永年方程式内の極を含む配列を格納する。
<i>difl</i>	REAL (slals0 / clals0 の場合) DOUBLE PRECISION (dlals0 / zlals0 の場合) 配列、次元は ( <i>k</i> )。 <i>difl</i> ( <i>i</i> ) には <i>i</i> 番目の更新された (収縮されていない) 特異値と <i>i</i> 番目の (収縮されていない) 古い特異値の距離を格納する。
<i>difr</i>	REAL (slals0 / clals0 の場合) DOUBLE PRECISION (dlals0 / zlals0 の場合) 配列、次元は ( <i>ldgnum</i> , 2)。 <i>difr</i> ( <i>i</i> , 1) には、 <i>i</i> 番目の更新された (収

縮されていない) 特異値と  $i+1$  番目の ( 収縮されていない) 古い特異値の距離を格納する。  $difr(i, 2)$  には  $i$  番目の右特異ベクトルに対する正規化係数を格納する。

$z$	REAL (slals0 / clals0 の場合 ) DOUBLE PRECISION (dlals0 / zlals0 の場合 ) 配列、次元は $(k)$ 。収縮調整された更新行ベクトルの成分を格納する。
$k$	INTEGER。非収縮行列の次元を格納する。これは関連する永年方程式の次数である。 $1 \leq k \leq n$
$c$	REAL (slals0 / clals0 の場合 ) DOUBLE PRECISION (dlals0 / zlals0 の場合 ) $sqre = 0$ の場合はガーベッジを格納し、 $sqre = 1$ の場合は右ヌル空間に 関係する Givens 回転の $c$ 値を格納する。
$s$	REAL (slals0 / clals0 の場合 ) DOUBLE PRECISION (dlals0 / zlals0 の場合 ) $sqre = 0$ の場合はガーベッジを格納し、 $sqre = 1$ の場合は右ヌル空間に 関係する Givens 回転の $s$ 値を格納する。
$work$	REAL (slals0 の場合 ) DOUBLE PRECISION (dlals0 の場合 ) ワークスペース配列、次元は $(k)$ 。実数型でのみ使用される。
$rwork$	REAL (clals0 の場合 ) DOUBLE PRECISION (zlals0 の場合 ) ワークスペース配列、次元は $(k*(1+nrhs) + 2*nrhs)$ 。複素型でのみ使用される。

## 出力パラメータ

$b$	行 1 から $n$ は解 $X$ で上書きされる。
$info$	INTEGER。 $info = 0$ の場合、正常に終了したことを示す。 $info = -i < 0$ の場合、 $i$ 番目の引数の値が不正だったことを示す。



## ?lalsa

コンパクト形式で係数行列の SVD を計算する。  
 ?gelsd で使用される。

### 構文

```
call slalsa( icipq, smlsiz, n, nrhs, b, ldb, bx, ldbx, u, ldu, vt, k, difl,
             difr, z, poles, givptr, givcol, ldgcol, perm, givnum, c, s, work, iwork,
             info )
call dlalsa( icipq, smlsiz, n, nrhs, b, ldb, bx, ldbx, u, ldu, vt, k, difl,
             difr, z, poles, givptr, givcol, ldgcol, perm, givnum, c, s, work, iwork,
             info )
call clalsa( icipq, smlsiz, n, nrhs, b, ldb, bx, ldbx, u, ldu, vt, k, difl,
             difr, z, poles, givptr, givcol, ldgcol, perm, givnum, c, s, rwork, iwork,
             info )
call zlalsa( icipq, smlsiz, n, nrhs, b, ldb, bx, ldbx, u, ldu, vt, k, difl,
             difr, z, poles, givptr, givcol, ldgcol, perm, givnum, c, s, rwork, iwork,
             info )
```

### 説明

このルーチンは最小二乗問題を解く中間の過程であり、コンパクト形式で係数行列の SVD を計算する。特異ベクトルは単純な直交行列の積として計算される。

ルーチン ?lalsa は、`icipq = 0` の場合は上二重対角行列の左特異ベクトル行列の逆を右辺に適用する。`icipq = 1` の場合は右特異ベクトル行列を右辺に適用する。特異ベクトル行列は ?lalsa によってコンパクト形式で生成される。

### 入力パラメータ

<code>icipq</code>	INTEGER。左または右のどちらの特異ベクトル行列を対象とするか指定する。 <code>icipq = 0</code> の場合、左特異ベクトル行列を使用する <code>icipq = 1</code> の場合、右特異ベクトル行列を使用する。
<code>smlsiz</code>	INTEGER。計算ツリーが一番下の部分問題の最大サイズ。
<code>n</code>	INTEGER。上二重対角行列の行と列の次元。
<code>nrhs</code>	INTEGER。 <code>b</code> と <code>bx</code> の列の数。1 以上でなければならない。

<i>b</i>	REAL (slalsa の場合) DOUBLE PRECISION (dlalsa の場合) COMPLEX (clalsa の場合) COMPLEX*16 (zlalsa の場合) 配列、次元は ( <i>ldb</i> , <i>nrhs</i> )。最小二乗問題の右辺を行 1 から <i>m</i> に格納する。
<i>ldb</i>	INTEGER。呼び出し元のサブプログラムの <i>b</i> のリーディング・ディメンジョン。max(1, max( <i>m</i> , <i>n</i> )) 以上でなければならない。
<i>ldb<sub>x</sub></i>	INTEGER。出力配列 <i>b<sub>x</sub></i> のリーディング・ディメンジョン。
<i>u</i>	REAL (slalsa/clalsa の場合) DOUBLE PRECISION (dlalsa/zlalsa の場合) 配列、次元は ( <i>ldu</i> , <i>smlsiz</i> )。 <i>u</i> には、一番下のレベルにあるすべての部分問題の左特異ベクトル行列を格納する。
<i>ldu</i>	INTEGER、 <i>ldu</i> ≥ <i>n</i> 。配列 <i>u</i> 、 <i>vt</i> 、 <i>difl</i> 、 <i>difr</i> 、 <i>poles</i> 、 <i>givnum</i> 、 <i>z</i> のリーディング・ディメンジョン。
<i>vt</i>	REAL (slalsa/clalsa の場合) DOUBLE PRECISION (dlalsa/zlalsa の場合) 配列、次元は ( <i>ldu</i> , <i>smlsiz</i> +1 )。一番下のレベルにあるすべての部分問題の右特異ベクトル行列を格納する。
<i>k</i>	INTEGER。配列、次元は ( <i>n</i> )。
<i>difl</i>	REAL (slalsa/clalsa の場合) DOUBLE PRECISION (dlalsa/zlalsa の場合) 配列、次元は ( <i>ldu</i> , <i>nlvl</i> )。 <i>nlvl</i> = int(log <sub>2</sub> ( <i>n</i> /( <i>smlsiz</i> +1))) + 1。
<i>difr</i>	REAL (slalsa/clalsa の場合) DOUBLE PRECISION (dlalsa/zlalsa の場合) 配列、次元は ( <i>ldu</i> , 2* <i>nlvl</i> )。 <i>difl</i> (*, <i>i</i> ) と <i>difr</i> (*, 2 <i>i</i> -1) には、 <i>i</i> 番目のレベルの特異値と ( <i>i</i> -1) 番目のレベルの特異値の距離を格納する。 <i>difr</i> (*, 2 <i>i</i> ) には <i>i</i> 番目のレベルの部分問題の右特異ベクトル行列に対する正規化係数を格納する。
<i>z</i>	REAL (slalsa/clalsa の場合) DOUBLE PRECISION (dlalsa/zlalsa の場合) 配列、次元は ( <i>ldu</i> , <i>nlvl</i> )。 <i>z</i> (1, <i>i</i> ) には <i>i</i> 番目レベルの部分問題に対する収縮調整された更新行ベクトルの成分を格納する。

<i>poles</i>	REAL (slalsa/clalsa の場合) DOUBLE PRECISION (dlalsa/zlalsa の場合) 配列、次元は ( <i>ldu</i> , 2* <i>nlvl</i> )。 <i>poles</i> (*, 2 <i>i</i> -1: 2 <i>i</i> ) には <i>i</i> 番目レベルの永年方程式に関係する新しい特異値と古い特異値を格納する。
<i>givptr</i>	INTEGER。 配列、次元は ( <i>n</i> )。 <i>givptr</i> ( <i>i</i> ) には計算ツリーの <i>i</i> 番目問題で実行された Givens 回転の回数を格納する。
<i>givcol</i>	INTEGER。 配列、次元は ( <i>ldgcol</i> , 2* <i>nlvl</i> )。 それぞれの <i>i</i> で、 <i>i</i> GIVCOL(*, 2 <i>i</i> -1: 2 <i>i</i> ) には計算ツリーの <i>i</i> 番目レベルで実行された Givens 回転の位置を格納する。
<i>ldgcol</i>	INTEGER、 <i>ldgcol</i> ≥ <i>n</i> 。配列 <i>givcol</i> 、 <i>perm</i> のリーディング・ディメンジョン。
<i>perm</i>	INTEGER。 配列、次元は ( <i>ldgcol</i> , <i>nlvl</i> )。 <i>perm</i> (*, <i>i</i> ) には計算ツリーの <i>i</i> 番目レベルで実行された置換を格納する。
<i>givnum</i>	REAL (slalsa/clalsa の場合) DOUBLE PRECISION (dlalsa/zlalsa の場合) 配列、次元は ( <i>ldu</i> , 2* <i>nlvl</i> )。 <i>givnum</i> (*, 2 <i>i</i> -1: 2 <i>i</i> ) には計算ツリーの <i>i</i> 番目レベルで実行された Givens 回転の <i>c</i> 値と <i>s</i> 値を格納する。
<i>c</i>	REAL (slalsa/clalsa の場合) DOUBLE PRECISION (dlalsa/zlalsa の場合) 配列、次元は ( <i>n</i> )。 <i>i</i> 番目の部分問題が正方ではない場合、 <i>c</i> ( <i>i</i> ) には <i>i</i> 番目の部分問題の右ヌル空間に関する Givens 回転の <i>c</i> 値を格納する。
<i>s</i>	REAL (slalsa/clalsa の場合) DOUBLE PRECISION (dlalsa/zlalsa の場合) 配列、次元は ( <i>n</i> )。 <i>i</i> 番目の部分問題が正方ではない場合、 <i>s</i> ( <i>i</i> ) には <i>i</i> 番目の部分問題の右ヌル空間に関する Givens 回転の <i>s</i> 値を格納する。
<i>work</i>	REAL (slalsa の場合) DOUBLE PRECISION (dlalsa の場合) ワークスペース配列、次元は ( <i>n</i> ) 以上。実数型でのみ使用される。

*rwork* REAL (clalsa の場合)  
DOUBLE PRECISION (zlalsa の場合)  
ワークスペース配列、次元は  
 $\max(n, (smlsiz+1)*nrhs*3)$  以上。複素型でのみ使用される。

*iwork* INTEGER。  
ワークスペース配列、次元は  $(3n)$  以上。

### 出力パラメータ

*b* 行 1 から  $n$  は解  $X$  で上書きされる。

*bx* REAL (slalsa の場合)  
DOUBLE PRECISION (DLALSA の場合)  
COMPLEX (CLALSA の場合)  
COMPLEX\*16 (ZLALSA の場合)  
配列、次元は  $(ldb, nrhs)$ 。 $b$  に対する左または右の特異ベクトル行列  
の適用結果で上書きされる。

*info* INTEGER。  
 $info = 0$  の場合、正常に終了したことを示す。  
 $info = -i < 0$  の場合、 $i$  番目の引数の値が不正だったことを示す。

---

## ?lalsd

最小二乗問題を解くために  $A$  の特異値分解を  
使用する。

---

### 構文

```
call slalsd( uplo, smlsiz, n, nrhs, d, e, b, ldb, rcond, rank, work, iwork,  
            info )  
call dlalsd( uplo, smlsiz, n, nrhs, d, e, b, ldb, rcond, rank, work, iwork,  
            info )  
call clalsd( uplo, smlsiz, n, nrhs, d, e, b, ldb, rcond, rank, work, rwork,  
            iwork, info )  
call zlalsd( uplo, smlsiz, n, nrhs, d, e, b, ldb, rcond, rank, work, rwork,  
            iwork, info )
```

## 説明

このルーチンは、 $AX=B$  の各列のユークリッド・ノルムを最小化するために  $X$  を求める最小二乗問題を  $A$  の特異値分解を使用して解き、ここで  $A$  は  $n \times n$  の上二重対角、 $X$  と  $B$  は  $n \times nrhs$  である。解  $X$  は  $B$  を上書きする。

最大特異値の  $rcond$  倍より小さな  $A$  の特異値は、最小二乗問題を解く過程でゼロとして扱われる。この場合は最小ノルム解が返される。実際の特異値は  $d$  に昇順で格納され返される。

このコードは浮動小数点演算に対してきわめて緩い仮定を行う。加減算においてガード桁を持つマシン、または Cray X-MP、Cray Y-MP、Cray C-90、または Cray-2 のような減算のガード桁を持たないバイナリマシンで動作する。

ガード桁を持たない 16 進マシンや 10 進マシンでは異常終了するのも考えられるが、そのような前例はない。

## 入力パラメータ

<i>uplo</i>	CHARACTER*1。 <i>uplo</i> = 'U' の場合、 $d$ と $e$ は上二重対角行列を定義する。 <i>uplo</i> = 'L' の場合、 $d$ と $e$ は下二重対角行列を定義する。
<i>smlsiz</i>	INTEGER。計算ツリーの一番下の部分問題の最大サイズ。
<i>n</i>	INTEGER。二重対角行列の次元。 $n \geq 0$
<i>nrhs</i>	INTEGER。 $B$ の列の数。1 以上でなければならない。
<i>d</i>	REAL (slalsd/clalsd の場合) DOUBLE PRECISION (dlalsd/zlalsd の場合) 配列、次元は $(n)$ 。 $d$ には二重対角行列の主対角を格納する。
<i>e</i>	REAL (slalsd/clalsd の場合) DOUBLE PRECISION (dlalsd/zlalsd の場合) 配列、次元は $(n-1)$ 。入力パラメータとして、二重対角行列の優対角成分を格納する。出力において $e$ の内容は壊される。
<i>b</i>	REAL (slalsd の場合) DOUBLE PRECISION (dlalsd の場合) COMPLEX (clalsd の場合) COMPLEX*16 (zlalsd の場合) 配列、次元は $(ldb,nrhs)$ 。入力パラメータとして $b$ には最小二乗問題の右辺を格納する。出力パラメータとして $b$ には解 $X$ が格納される。

<i>ldb</i>	INTEGER。呼び出し元のサブプログラムの <i>b</i> のリーディング・ディメンション。max(1, <i>n</i> ) 以上でなければならない。
<i>rcond</i>	REAL (slalsd/clalsd の場合) DOUBLE PRECISION (dlalsd/zlalsd の場合) 最大特異値の <i>rcond</i> 倍より小さいか等しい <i>A</i> の特異値は、最小二乗問題を解く過程でゼロとして扱われる。 <i>rcond</i> が負の場合はマシン精度が代わりに使用される。 例えば、 $\text{diag}(S) * X = B$ が最小二乗問題で $\text{diag}(S)$ が特異値の二重対角行列の場合、解は $S(i)$ が $rcond * \max(S)$ より大きい場合は $X(i) = B(i) / S(i)$ に、 $S(i)$ が $rcond * \max(S)$ より小さいか等しい場合は $X(i) = 0$ になる。
<i>rank</i>	INTEGER。最大特異値の <i>rcond</i> 倍より大きい <i>A</i> の特異値の個数。
<i>work</i>	REAL (slalsd の場合) DOUBLE PRECISION (dlalsd の場合) COMPLEX (clalsd の場合) COMPLEX*16 (zlalsd の場合) ワークスペース配列。 実数型の場合、次元は $(9n + 2n * smlsiz + 8n * nlvl + n * nrhs + (smlsiz + 1)^2)$ 以上。 ここで、 $nlvl = \max(0, \text{int}(\log_2(n / (smlsiz + 1))) + 1)$ 複素型の場合、次元は $(n * nrhs)$ 以上。
<i>rwork</i>	REAL (clalsd の場合) DOUBLE PRECISION (zlalsd の場合) ワークスペース配列、複素型でのみ使用される。 次元は $(9n + 2n * smlsiz + 8n * nlvl + 3 * mlsiz * nrhs + (smlsiz + 1)^2)$ 以上。 ここで $nlvl = \max(0, \text{int}(\log_2(\min(m, n) / (smlsiz + 1))) + 1)$
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $(3n * nlvl + 11n)$ 以上。

## 出力パラメータ

<i>d</i>	<i>info</i> = 0 の場合、 <i>d</i> には二重対角行列の特異値が格納される。
<i>b</i>	<i>b</i> には解 <i>X</i> が格納される。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> < 0 の場合、 <i>i</i> 番目の引数の値が不正であったことを示す。

$info = i > 0$  の場合、 $info/(n+1)$  から  $\text{mod}(info, n+1)$  までの行と列にある部分行列の操作中に、アルゴリズムが特異値の計算に失敗したことを示す。

## ?lamrg

2 つの独立するソートされたセットの成分を  
単一のソートされたセットに昇順で併合する  
置換リストを生成する。

### 構文

```
call slamrg( n1, n2, a, strd1, strd2, index )
call dlamrg( n1, n2, a, strd1, strd2, index )
```

### 説明

このルーチンは、 $a$  (2 つの独立したソートされたセットで構成) の成分を単一のソートされたセットに昇順で併合する置換リストを生成する。

### 入力パラメータ

$n1, n2$	INTEGER。 これら引数には、併合対象となる 2 つのソートされたリストのそれぞれの長さを格納する。
$a$	REAL (slamrg の場合 ) DOUBLE PRECISION (dlamrg の場合 ) 配列、次元は $(n1+n2)$ $a$ の最初の $n1$ 個の成分には、昇順または降順のどちらかでソートされる数値のリストを格納する。最後の $n2$ 個の成分も同様である。
$strd1, strd2$	INTEGER。 配列 $a$ を介して取られるストライドである。可能なストライドは 1 と -1 である。これらは $a$ の部分集合が昇順 ( $strdx = 1$ )、または降順 ( $strdx = -1$ ) でソートされることを示す。

### 出力パラメータ

`index`                    INTEGER。  
配列、次元は  $(n1+n2)$   
 $i = 1, n1+n2$  に対して  $b(i) = a(index(i))$  ならば、この配列には置換が格納され、 $b$  は昇順でソートされる。

---

## ?langb

一般帯行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。

---

### 構文

```
val = slangb( norm, n, kl, ku, ab, ldab, work )  
val = dlangb( norm, n, kl, ku, ab, ldab, work )  
val = clangb( norm, n, kl, ku, ab, ldab, work )  
val = zlangb( norm, n, kl, ku, ab, ldab, work )
```

### 説明

この関数は、 $kl$  個の劣対角と  $ku$  個の優対角を持つ  $n \times n$  帯行列  $A$  の、1- ノルムの値、または Frobenius ノルムの値、または無限ノルムの値、または最大絶対値の成分を返す。

関数の戻り値は `val` に格納される。

`val = max(abs( $A_{ij}$ ))`、`norm = 'M'` または `'m'` の場合

    = `norm1( $A$ )`、`norm = '1'` または `'O'` または `'o'` の場合

    = `normI( $A$ )`、`norm = 'I'` または `'i'` の場合

    = `normF( $A$ )`、`norm = 'F', 'f', 'E'` または `'e'` の場合

`norm1` は行列の 1- ノルム ( 最大列合計 )、`normI` は行列の無限ノルム ( 最大行合計 )、`normF` は行列の Frobenius ノルム ( 二乗和の平方根 ) を示す。`max(abs( $A_{ij}$ ))` は行列ノルムではない点に注意する。

### 入力パラメータ

`norm`                    CHARACTER\*1。ルーチンが返す値を上述のように指定する。



<i>n</i>	INTEGER。行列 <i>A</i> の次数。 $n \geq 0$ $n = 0$ と指定した場合、 <i>?langb</i> はゼロに設定される。
<i>kl</i>	INTEGER。行列 <i>A</i> の劣対角成分の数。 $kl \geq 0$
<i>ku</i>	INTEGER。行列 <i>A</i> の優対角成分の数。 $ku \geq 0$
<i>ab</i>	REAL ( <i>slangb</i> の場合 ) DOUBLE PRECISION ( <i>dlangb</i> の場合 ) COMPLEX ( <i>clangb</i> の場合 ) COMPLEX*16 ( <i>zlangb</i> の場合 ) 配列、次元は ( <i>ldab</i> , <i>n</i> )。帯行列 <i>A</i> で、行 1 から $kl+ku+1$ に格納する。 <i>A</i> の <i>j</i> 番目の列を配列 <i>ab</i> の <i>j</i> 番目の列に次のように格納する。 $\max(1, j-ku) \leq i \leq \min(n, j+kl)$ に対して $ab(ku+1+i-j, j) = a(i, j)$ の場合。
<i>ldab</i>	INTEGER。配列 <i>ab</i> のリーディング・ディメンジョン。 $ldab \geq kl+ku+1$
<i>work</i>	REAL ( <i>slangb</i> / <i>clangb</i> の場合 ) DOUBLE PRECISION ( <i>dlangb</i> / <i>zlangb</i> の場合 ) ワークスペース配列、次元は ( <i>lwork</i> ) <i>norm</i> = 'I' のとき $lwork \geq n$ 。それ以外では <i>work</i> は参照されない。

### 出力パラメータ

<i>val</i>	REAL ( <i>slangb</i> / <i>clangb</i> の場合 ) DOUBLE PRECISION ( <i>dlangb</i> / <i>zlangb</i> の場合 ) 関数の戻り値。
------------	---

## ?lange

一般矩形行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。

### 構文

```
val = slangb( norm, m, n, a, lda, work )
val = dlangb( norm, m, n, a, lda, work )
val = clangb( norm, m, n, a, lda, work )
```

```
val = zlange( norm, m, n, a, lda, work )
```

## 説明

関数 ?lange は、実数 / 複素行列  $A$  の、1- ノルムの値、または Frobenius ノルムの値、または無限ノルムの値、または最大絶対値の成分を返す。

関数の戻り値は `val` に格納される。

`val = max(abs(Aij))`、`norm = 'M'` または `'m'` の場合

`= norm1(A)`、`norm = '1'` または `'O'` または `'o'` の場合

`= normI(A)`、`norm = 'I'` または `'i'` の場合

`= normF(A)`、`norm = 'F', 'f', 'E'` または `'e'` の場合

`norm1` は行列の 1- ノルム ( 最大列合計 )、`normI` は行列の無限ノルム ( 最大行合計 )、`normF` は行列の Frobenius ノルム ( 二乗和の平方根 ) を示す。 `max(abs(Aij))` は行列ノルムではない点に注意する。

## 入力パラメータ

<code>norm</code>	CHARACTER*1。ルーチン ?lange が返す値を上述のように指定する。
<code>m</code>	INTEGER。行列 $A$ の行数。 $m \geq 0$ $m = 0$ と指定した場合、?lange はゼロに設定される。
<code>n</code>	INTEGER。行列 $A$ の列数。 $n \geq 0$ $n = 0$ と指定した場合、?lange はゼロに設定される。
<code>a</code>	REAL (slange の場合 ) DOUBLE PRECISION (dlange の場合 ) COMPLEX (clange の場合 ) COMPLEX*16 (zlange の場合 ) 配列、次元は (lda,n)。 $m \times n$ の行列 $A$
<code>lda</code>	INTEGER。配列 <code>a</code> のリーディング・ディメンジョン。 $lda \geq \max(m, 1)$
<code>work</code>	REAL (slange と clange の場合 ) DOUBLE PRECISION (dlange と zlange の場合 ) ワークスペース配列、次元は (lwork)。 <code>norm = 'I'</code> のとき $lwork \geq m$ 。 それ以外では <code>work</code> は参照されない。

## 出力パラメータ

`val` REAL (`slange/clange` の場合 )  
 DOUBLE PRECISION (`dlange/zlange` の場合 )  
 関数の戻り値。

## ?langt

一般三重対角行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。

### 構文

```
val = slangt( norm, n, dl, d, du )
val = dlangt( norm, n, dl, d, du )
val = clangt( norm, n, dl, d, du )
val = zlangt( norm, n, dl, d, du )
```

### 説明

このルーチンは、実数 / 複素三重対角行列  $A$  の、1- ノルムの値、または Frobenius ノルムの値、または無限ノルムの値、または最大絶対値の成分を返す。

関数の戻り値は `val` に格納される。

`val = max(abs( $A_{ij}$ ))`、`norm = 'M'` または `'m'` の場合

`= norm1( $A$ )`、`norm = '1'` または `'O'` または `'o'` の場合

`= normI( $A$ )`、`norm = 'I'` または `'i'` の場合

`= normF( $A$ )`、`norm = 'F', 'f', 'E'` または `'e'` の場合

`norm1` は行列の 1- ノルム ( 最大列合計 )、`normI` は行列の無限ノルム ( 最大行合計 )、`normF` は行列の Frobenius ノルム ( 二乗和の平方根 ) を示す。`max(abs( $A_{ij}$ ))` は行列ノルムではない点に注意する。

## 入力パラメータ

`norm` CHARACTER\*1。ルーチン `?langt` が返す値を上述のように指定する。

$n$  INTEGER。行列  $A$  の次数。  
 $n \geq 0$   $n = 0$  と指定した場合、 $?langt$  はゼロに設定される。

$dl, d, du$  REAL( $slangt$  の場合 )  
 DOUBLE PRECISION( $dlangt$  の場合 )  
 COMPLEX( $clangt$  の場合 )  
 COMPLEX\*16( $zlangt$  の場合 )  
 配列:  $dl(n-1), d(n), du(n-1)$ 。  
 配列  $dl$  には  $A$  の  $(n-1)$  個の劣対角成分を格納する。  
 配列  $d$  には  $A$  の対角成分を格納する。  
 配列  $du$  には  $A$  の  $(n-1)$  個の優対角成分を格納する。

## 出力パラメータ

$val$  REAL( $slangt/clangt$  の場合 )  
 DOUBLE PRECISION( $dlangt/zlangt$  の場合 )  
 関数の戻り値。

---

## ?lanhs

上 Hessenberg 行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。

---

### 構文

```
val = slanh( norm, n, a, lda, work )
val = dlanhs( norm, n, a, lda, work )
val = clanhs( norm, n, a, lda, work )
val = zlanhs( norm, n, a, lda, work )
```

### 説明

関数  $?lanhs$  は、Hessenberg 行列  $A$  の、1- ノルムの値、または Frobenius ノルムの値、または無限ノルムの値、または最大絶対値の成分を返す。

関数の戻り値は  $val$  に格納される。

$val = \max(\text{abs}(A_{ij}))$ 、 $norm = 'M'$  または  $'m'$  の場合  
 $= \text{norm1}(A)$ 、 $norm = '1'$  または  $'O'$  または  $'o'$  の場合

= normI(A)、norm='I' または 'i' の場合

= normF(A)、norm='F', 'f', 'E' または 'e' の場合

norm1 は行列の 1- ノルム ( 最大列合計 )、normI は行列の無限ノルム ( 最大行合計 )、normF は行列の Frobenius ノルム ( 二乗和の平方根 ) を示す。max(abs(A<sub>ij</sub>)) は行列ノルムではない点に注意する。

## 入力パラメータ

*norm* CHARACTER\*1。ルーチン ?lanhs が返す値を上述のように指定する。

*n* INTEGER。行列 *A* の次数。  
 $n \geq 0$   $n=0$  と指定した場合、?lanhs はゼロに設定される。

*a* REAL(slanhs の場合 )  
 DOUBLE PRECISION(dlanhs の場合 )  
 COMPLEX(clanhs の場合 )  
 COMPLEX\*16(zlanhs の場合 )  
 配列、次元は (lda,n)。 $n \times n$  の上 Hessenberg 行列 *A*。*A* のうち最初の  
 劣対角から下の部分は参照されない。

*lda* INTEGER。配列 *a* のリーディング・ディメンション。  
 $lda \geq \max(n, 1)$

*work* REAL(slanhs と clanhs の場合 )  
 DOUBLE PRECISION(dlange と zlange の場合 )  
 ワークスペース配列、次元は (lwork)。norm='I' のとき  $lwork \geq n$ 。  
 それ以外では work は参照されない。

## 出力パラメータ

*val* REAL(slanhs/clanhs の場合 )  
 DOUBLE PRECISION(dlanhs/zlanhs の場合 )  
 関数の戻り値。

## ?lansb

対称帯行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。

### 構文

```
val = slansb( norm, uplo, n, k, ab, ldab, work )
val = dlansb( norm, uplo, n, k, ab, ldab, work )
val = clansb( norm, uplo, n, k, ab, ldab, work )
val = zlansb( norm, uplo, n, k, ab, ldab, work )
```

### 説明

関数 ?lansb は、 $k$  個の優対角を持つ  $n \times n$  実数 / 複素対称帯行列  $A$  の、1- ノルムの値、または Frobenius ノルムの値、または無限ノルムの値、または最大絶対値の成分を返す。

関数の戻り値は `val` に格納される。

`val = max(abs( $A_{ij}$ ))`、`norm = 'M'` または `'m'` の場合  
= `norm1(A)`、`norm = '1'` または `'O'` または `'o'` の場合  
= `normI(A)`、`norm = 'I'` または `'i'` の場合  
= `normF(A)`、`norm = 'F', 'f', 'E'` または `'e'` の場合

`norm1` は行列の 1- ノルム ( 最大列合計 )、`normI` は行列の無限ノルム ( 最大行合計 )、`normF` は行列の Frobenius ノルム ( 二乗和の平方根 ) を示す。`max(abs( $A_{ij}$ ))` は行列ノルムではない点に注意する。

### 入力パラメータ

<code>norm</code>	CHARACTER*1。ルーチン ?lansb が返す値を上述のように指定する。
<code>uplo</code>	CHARACTER*1。帯行列 $A$ の上三角部分または下三角部分のどちらを与えるか指定する。 <code>uplo = 'U'</code> の場合、上三角部分を与える。 <code>uplo = 'L'</code> の場合、下三角部分を与える。
<code>n</code>	INTEGER。行列 $A$ の次数。 $n \geq 0$ $n = 0$ と指定した場合、?lansb はゼロに設定される。

<i>k</i>	INTEGER。帯行列 <i>A</i> の優対角または劣対角の個数。 $k \geq 0$
<i>ab</i>	REAL (slansb の場合 ) DOUBLE PRECISION (dlansb の場合 ) COMPLEX (clansb の場合 ) COMPLEX*16 (zlansb の場合 ) 配列、次元は ( <i>ldab</i> , <i>n</i> )。対称帯行列 <i>A</i> の上三角または下三角で、 <i>ab</i> の最初の <i>k</i> +1 行に格納する。 <i>A</i> の <i>j</i> 番目の列は配列 <i>ab</i> の <i>j</i> 番目の列に次のように格納する。 <i>uplo</i> ='U' の場合、 $\max(1, j-k) \leq i \leq j$ に対して $ab(k+1+i-j, j) = a(i, j)$ <i>uplo</i> ='L' の場合、 $j \leq i \leq \min(n, j+k)$ に対して $ab(1+i-j, j) = a(i, j)$
<i>ldab</i>	INTEGER。配列 <i>ab</i> のリーディング・ディメンジョン。 $ldab \geq k+1$
<i>work</i>	REAL (slansb と clansb の場合 ) DOUBLE PRECISION (dlansb と zlansb の場合 ) ワークスペース配列、次元は ( <i>lwork</i> )。 <i>norm</i> ='I'、'1'、'O' のいずれかであるとき $lwork \geq n$ 。それ以外では <i>work</i> は参照されない。

### 出力パラメータ

<i>val</i>	REAL (slansb/clansb の場合 ) DOUBLE PRECISION (dlansb/zlansb の場合 ) 関数の戻り値。
------------	---

## ?lanhb

エルミート帯行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。

### 構文

```
val = clanhb( norm, uplo, n, k, ab, ldab, work )
val = zlanhb( norm, uplo, n, k, ab, ldab, work )
```

## 説明

このルーチンは、 $k$  個の優対角を持つ  $n \times n$  エルミート帯行列  $A$  の、1- ノルムの値、または Frobenius ノルムの値、または無限ノルムの値、または最大絶対値の成分を返す。

関数の戻り値は `val` に格納される。

`val = max(abs(Aij))`、`norm = 'M'` または `'m'` の場合

`= norm1(A)`、`norm = '1'` または `'O'` または `'o'` の場合

`= normI(A)`、`norm = 'I'` または `'i'` の場合

`= normF(A)`、`norm = 'F'`、`'f'`、`'E'` または `'e'` の場合

`norm1` は行列の 1- ノルム ( 最大列合計 )、`normI` は行列の無限ノルム ( 最大行合計 )、`normF` は行列の Frobenius ノルム ( 二乗和の平方根 ) を示す。`max(abs(Aij))` は行列ノルムではない点に注意する。

## 入力パラメータ

<code>norm</code>	CHARACTER*1。ルーチン <code>?lanhnb</code> が返す値を上述のように指定する。
<code>uplo</code>	CHARACTER*1。帯行列 $A$ の上三角部分または下三角部分のどちらを与えるか指定する。 <code>uplo = 'U'</code> の場合、上三角部分を与える。 <code>uplo = 'L'</code> の場合、下三角部分を与える。
<code>n</code>	INTEGER。行列 $A$ の次数。 $n \geq 0$ $n = 0$ と指定した場合、 <code>?lanhnb</code> はゼロに設定される。
<code>k</code>	INTEGER。帯行列 $A$ の優対角または劣対角の個数。 $k \geq 0$
<code>ab</code>	COMPLEX ( <code>clanhb</code> の場合 ) COMPLEX*16 ( <code>zlanhb</code> の場合 ) 配列、次元は $(ldab, n)$ 。対称帯行列 $A$ の上三角または下三角で、 <code>ab</code> の最初の $k+1$ 行に格納する。 $A$ の $j$ 番目の列は配列 <code>ab</code> の $j$ 番目の列に次のように格納する。 <code>uplo = 'U'</code> の場合、 $\max(1, j-k) \leq i \leq j$ に対して <code>ab(k+1+i-j, j) = a(i, j)</code> <code>uplo = 'L'</code> の場合、 $j \leq i \leq \min(n, j+k)$ に対して <code>ab(1+i-j, j) = a(i, j)</code> 対角成分の虚数部分は設定する必要はなくゼロとして仮定される点に注意する。
<code>ldab</code>	INTEGER。配列 <code>ab</code> のリーディング・ディメンション。 $ldab \geq k+1$



`work` REAL (`clanhb` の場合)  
 DOUBLE PRECISION (`zlanhb` の場合)  
 ワークスペース配列、次元は (`lwork`)。  
`norm = 'I', '1', 'O'` のいずれかであるとき  $lwork \geq n$ 。それ以外では  
`work` は参照されない。

### 出力パラメータ

`val` REAL (`slanhb/clanhb` の場合)  
 DOUBLE PRECISION (`dlanhb/zlanhb` の場合)  
 関数の戻り値。

## ?lansp

圧縮形式で与えられる対称行列の 1- ノルムの値、  
 Frobenius ノルムの値、無限ノルムの値、あるいは  
 最大絶対値の成分を返す。

### 構文

```
val = slansp( norm, uplo, n, ap, work )
val = dlansp( norm, uplo, n, ap, work )
val = clansp( norm, uplo, n, ap, work )
val = zlansp( norm, uplo, n, ap, work )
```

### 説明

関数 ?lansp は、圧縮形式で与えられる実数 / 複素対称行列  $A$  の、1- ノルムの値、または Frobenius ノルムの値、または無限ノルムの値、または最大絶対値の成分を返す。

関数の戻り値は `val` に格納される。

`val = max(abs( $A_{ij}$ ))`、`norm = 'M'` または `'m'` の場合  
     = `norm1(A)`、`norm = '1'` または `'O'` または `'o'` の場合  
     = `normI(A)`、`norm = 'I'` または `'i'` の場合  
     = `normF(A)`、`norm = 'F', 'f', 'E'` または `'e'` の場合

`norm1` は行列の 1- ノルム ( 最大列合計 )、`normI` は行列の無限ノルム ( 最大行合計 )、`normF` は行列の Frobenius ノルム ( 二乗和の平方根 ) を示す。 `max(abs(Aij))` は行列ノルムではない点に注意する。

### 入力パラメータ

<code>norm</code>	CHARACTER*1。ルーチン <code>?lansp</code> が返す値を上述のように指定する。
<code>uplo</code>	CHARACTER*1。対称行列 $A$ の上三角部分または下三角部分のどちらを与えるか指定する。 <code>uplo = 'U'</code> の場合、 $A$ の上三角部分を与える。 <code>uplo = 'L'</code> の場合、 $A$ の下三角部分を与える。
<code>n</code>	INTEGER。行列 $A$ の次数。 $n \geq 0$ $n = 0$ と指定した場合、 <code>?lansp</code> はゼロに設定される。
<code>ap</code>	REAL ( <code>slansp</code> の場合 ) DOUBLE PRECISION ( <code>dlansp</code> の場合 ) COMPLEX ( <code>clansp</code> の場合 ) COMPLEX*16 ( <code>zlansp</code> の場合 ) 配列、次元は $(n(n+1)/2)$ 。線形配列に列方向に圧縮された対称行列 $A$ の上三角または下三角。 $A$ の $j$ 番目の列は配列 <code>ap</code> に次のように格納する。 <code>uplo = 'U'</code> の場合、 $1 \leq i \leq j$ に対して $ap(i + (j-1)j/2) = A(i, j)$ <code>uplo = 'L'</code> の場合、 $j \leq i \leq n$ に対して $ap(i + (j-1)(2n-j)/2) = A(i, j)$
<code>work</code>	REAL ( <code>slansp</code> と <code>clansp</code> の場合 ) DOUBLE PRECISION ( <code>dlansp</code> と <code>zlansp</code> の場合 ) ワークスペース配列、次元は ( <code>lwork</code> )。 <code>norm = 'I'</code> 、 <code>'1'</code> 、 <code>'O'</code> のいずれかであるとき $lwork \geq n$ 。それ以外では <code>work</code> は参照されない。

### 出力パラメータ

<code>val</code>	REAL ( <code>slansp/clansp</code> の場合 ) DOUBLE PRECISION ( <code>dlansp/zlansp</code> の場合 ) 関数の戻り値。
------------------	---

## ?lanhp

圧縮形式で与えられる複素エルミート行列の 1-ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。

### 構文

```
val = clanhp( norm, uplo, n, ap, work )
val = zlanhp( norm, uplo, n, ap, work )
```

### 説明

関数 ?lanhp は、圧縮形式で与えられる複素エルミート行列  $A$  の、1- ノルムの値、または Frobenius ノルムの値、または無限ノルムの値、または最大絶対値の成分を返す。

関数の戻り値は `val` に格納される。

`val = max(abs( $A_{ij}$ ))`、`norm = 'M'` または `'m'` の場合  
     = `norm1(A)`、`norm = '1'` または `'O'` または `'o'` の場合  
     = `normI(A)`、`norm = 'I'` または `'i'` の場合  
     = `normF(A)`、`norm = 'F'`、`'f'`、`'E'` または `'e'` の場合

`norm1` は行列の 1- ノルム ( 最大列合計 )、`normI` は行列の無限ノルム ( 最大行合計 )、`normF` は行列の Frobenius ノルム ( 二乗和の平方根 ) を示す。`max(abs( $A_{ij}$ ))` は行列ノルムではない点に注意する。

### 入力パラメータ

`norm` CHARACTER\*1。ルーチン ?lanhp が返す値を上述のように指定する。

`uplo` CHARACTER\*1。エルミート行列  $A$  の上三角部分または下三角部分のどちらを与えるか指定する。  
     `uplo = 'U'` の場合、 $A$  の上三角部分を与える。  
     `uplo = 'L'` の場合、 $A$  の下三角部分を与える。

`n` INTEGER。行列  $A$  の次数。  
      $n \geq 0$   $n = 0$  と指定した場合、?lanhp はゼロに設定される。

**ap**                    `COMPLEX` (`clanhp` の場合 )  
                          `COMPLEX*16` (`zlanhp` の場合 )  
 配列、次元は  $(n(n+1)/2)$ 。線形配列に列方向に圧縮されたエルミート行列  $A$  の上三角または下三角。 $A$  の  $j$  番目の列は配列  $ap$  に次のように格納する。  
        $uplo = 'U'$  の場合、 $1 \leq i \leq j$  に対して  $ap(i + (j-1)j/2) = A(i, j)$   
        $uplo = 'L'$  の場合、 $j \leq i \leq n$  に対して  $ap(i + (j-1)(2n-j)/2) = A(i, j)$

**work**                `REAL` (`clanhp` の場合 )  
                          `DOUBLE PRECISION` (`zlanhp` の場合 )  
 ワークスペース配列、次元は ( $lwork$ )。  
 $norm = 'I'$ 、 $'1'$ 、 $'O'$  のいずれかであるとき  $lwork \geq n$ 。それ以外では  $work$  は参照されない。

## 出力パラメータ

**val**                    `REAL` (`clanhp` の場合 )  
                          `DOUBLE PRECISION` (`zlanhp` の場合 )  
 関数の戻り値。

---

## ?lanst/?lanht

実数対称または複素エルミート三重対角行列の  
 $1$ - ノルムの値、*Frobenius* ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。

---

### 構文

```
val = slanst( norm, n, d, e )
val = dlanst( norm, n, d, e )
val = clanht( norm, n, d, e )
val = zlanht( norm, n, d, e )
```

### 説明

関数 ?lanst/?lanht は、実数対称または複素エルミート三重対角行列  $A$  の、 $1$ - ノルムの値、または *Frobenius* ノルムの値、または無限ノルムの値、または最大絶対値の成分を返す。

関数の戻り値は `val` に格納される。

$val = \max(\text{abs}(A_{ij}))$ 、 $norm = 'M'$  または  $'m'$  の場合  
 $= \text{norm1}(A)$ 、 $norm = '1'$  または  $'O'$  または  $'o'$  の場合  
 $= \text{normI}(A)$ 、 $norm = 'I'$  または  $'i'$  の場合  
 $= \text{normF}(A)$ 、 $norm = 'F', 'f', 'E'$  または  $'e'$  の場合

$\text{norm1}$  は行列の 1- ノルム ( 最大列合計 )、 $\text{normI}$  は行列の無限ノルム ( 最大行合計 )、 $\text{normF}$  は行列の Frobenius ノルム ( 二乗和の平方根 ) を示す。 $\max(\text{abs}(A_{ij}))$  は行列ノルムではない点に注意する。

### 入力パラメータ

$norm$  CHARACTER\*1。ルーチン  $?lanst/?lanht$  が返す値を上述のように指定する。  
 $n$  INTEGER。行列  $A$  の次数。  
 $n \geq 0$   $n = 0$  と指定した場合、 $?lanst/?lanht$  はゼロに設定される。  
 $d$  REAL( $slanst/clanht$  の場合 )  
 DOUBLE PRECISION( $dlanst/zlanht$  の場合 )  
 配列、次元は  $(n)$ 。 $A$  の対角成分。  
 $e$  REAL( $slanst$  の場合 )  
 DOUBLE PRECISION( $dlanst$  の場合 )  
 COMPLEX( $clanht$  の場合 )  
 COMPLEX\*16( $zlanht$  の場合 )  
 配列、次元は  $(n-1)$ 。 $A$  の  $(n-1)$  個の劣対角成分または優対角成分。

### 出力パラメータ

$val$  REAL( $slanst/clanht$  の場合 )  
 DOUBLE PRECISION( $dlanst/zlanht$  の場合 )  
 関数の戻り値。

### ?lansy

実数 / 複素対称行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。

---

#### 構文

```
val = slansy( norm, uplo, n, a, lda, work )  
val = dlansy( norm, uplo, n, a, lda, work )  
val = clansy( norm, uplo, n, a, lda, work )  
val = zlansy( norm, uplo, n, a, lda, work )
```

#### 説明

関数 ?lansy は、実数 / 複素対称行列  $A$  の、1- ノルムの値、または Frobenius ノルムの値、または無限ノルムの値、または最大絶対値の成分を返す。

関数の戻り値は `val` に格納される。

`val = max(abs( $A_{ij}$ ))`、`norm = 'M'` または `'m'` の場合

`= norm1( $A$ )`、`norm = '1'` または `'O'` または `'o'` の場合

`= normI( $A$ )`、`norm = 'I'` または `'i'` の場合

`= normF( $A$ )`、`norm = 'F', 'f', 'E'` または `'e'` の場合

`norm1` は行列の 1- ノルム ( 最大列合計 )、`normI` は行列の無限ノルム ( 最大行合計 )、`normF` は行列の Frobenius ノルム ( 二乗和の平方根 ) を示す。`max(abs( $A_{ij}$ ))` は行列ノルムではない点に注意する。

#### 入力パラメータ

<code>norm</code>	CHARACTER*1。ルーチン ?lansy が返す値を上述のように指定する。
<code>uplo</code>	CHARACTER*1。対称行列 $A$ の上三角部分と下三角部分のどちらを参照させるか指定する。 <code>= 'U'</code> の場合、 $A$ の上三角部分を参照させる。 <code>= 'L'</code> の場合、 $A$ の下三角部分を参照させる。
<code>n</code>	INTEGER。行列 $A$ の次数。 $n \geq 0$ $n = 0$ と指定した場合、?lansy はゼロに設定される。

<code>a</code>	<p>REAL (slansy の場合 )</p> <p>DOUBLE PRECISION (dlansy の場合 )</p> <p>COMPLEX (clansy の場合 )</p> <p>COMPLEX*16 (zlansy の場合 )</p> <p>配列、次元は (<code>lda</code>, <math>N</math>)。対称行列 <math>A</math>。 <code>uplo='U'</code> の場合、<code>a</code> の先頭の <math>n \times n</math> 上三角部分に行列 <math>A</math> の上三角部分を格納する。<code>a</code> の厳密な下三角部分は参照されない。</p> <p><code>uplo='L'</code> の場合、<code>a</code> の先頭の <math>n \times n</math> 下三角部分に行列 <math>A</math> の下三角部分を格納する。<code>a</code> の厳密な上三角部分は参照されない。</p>
<code>lda</code>	<p>INTEGER。配列 <code>a</code> のリーディング・ディメンジョン。</p> <p><math>lda \geq \max(n, 1)</math></p>
<code>work</code>	<p>REAL (slansy と clansy の場合 )</p> <p>DOUBLE PRECISION (dlansy と zlansy の場合 )</p> <p>ワークスペース配列、次元は (<code>lwork</code>)。</p> <p><code>norm='I'</code>、<code>'1'</code>、<code>'O'</code> のいずれかであるとき <math>lwork \geq n</math>。それ以外では <code>work</code> は参照されない。</p>

## 出力パラメータ

<code>val</code>	<p>REAL (slansy/clansy の場合 )</p> <p>DOUBLE PRECISION (dlansy/zlansy の場合 )</p> <p>関数の戻り値。</p>
------------------	--

## ?lanhe

複素エルミート行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。

### 構文

```
val = clanhe( norm, uplo, n, a, lda, work )
val = zlanhe( norm, uplo, n, a, lda, work )
```

### 説明

関数 ?lanhe は、複素エルミート行列  $A$  の、1- ノルムの値、または Frobenius ノルムの値、または無限ノルムの値、または最大絶対値の成分を返す。

関数の戻り値は `val` に格納される。

`val = max(abs(Aij))`、`norm = 'M'` または `'m'` の場合  
`= norm1(A)`、`norm = '1'` または `'O'` または `'o'` の場合  
`= normI(A)`、`norm = 'I'` または `'i'` の場合  
`= normF(A)`、`norm = 'F'`、`'f'`、`'E'` または `'e'` の場合

`norm1` は行列の 1- ノルム ( 最大列合計 )、`normI` は行列の無限ノルム ( 最大行合計 )、`normF` は行列の Frobenius ノルム ( 二乗和の平方根 ) を示す。`max(abs(Aij))` は行列ノルムではない点に注意する。

## 入力パラメータ

<code>norm</code>	CHARACTER*1。ルーチン <code>?lanhe</code> が返す値を上述のように指定する。
<code>uplo</code>	CHARACTER*1。エルミート行列 $A$ の上三角部分と下三角部分のどちらを参照させるか指定する。 <code>= 'U'</code> の場合、 $A$ の上三角部分を参照させる。 <code>= 'L'</code> の場合、 $A$ の下三角部分を参照させる。
<code>n</code>	INTEGER。行列 $A$ の次数。 $n \geq 0$ $n = 0$ と指定した場合、 <code>?lanhe</code> はゼロに設定される。
<code>a</code>	COMPLEX ( <code>clanhe</code> の場合 ) COMPLEX*16 ( <code>zlanhe</code> の場合 ) 配列、次元は $(lda, n)$ 。エルミート行列 $A$ 。 <code>uplo = 'U'</code> の場合、 $a$ の先頭の $n \times n$ 上三角部分に行列 $A$ の上三角部分を格納する。 $a$ の厳密な下三角部分は参照されない。 <code>uplo = 'L'</code> の場合、 $a$ の先頭の $n \times n$ 下三角部分に行列 $A$ の下三角部分を格納する。 $a$ の厳密な上三角部分は参照されない。
<code>lda</code>	INTEGER。配列 $a$ のリーディング・ディメンジョン。 $lda \geq \max(n, 1)$
<code>work</code>	REAL ( <code>clanhe</code> の場合 ) DOUBLE PRECISION ( <code>zlanhe</code> の場合 ) ワークスペース配列、次元は $(lwork)$ 。 <code>norm = 'I'</code> 、 <code>'1'</code> 、 <code>'O'</code> のいずれかであるとき $lwork \geq n$ 。それ以外では <code>work</code> は参照されない。



## 出力パラメータ

`val`                `REAL` (`clanhe` の場合 )  
                      `DOUBLE PRECISION` (`zlanhe` の場合 )  
                      関数の戻り値。

## ?lantb

三角帯行列の 1- ノルムの値、*Frobenius* ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。

### 構文

```
val = slantb( norm, uplo, diag, n, k, ab, ldab, work )
val = dlantb( norm, uplo, diag, n, k, ab, ldab, work )
val = clantb( norm, uplo, diag, n, k, ab, ldab, work )
val = zlantb( norm, uplo, diag, n, k, ab, ldab, work )
```

### 説明

関数 ?lantb は、 $(k+1)$  個の対角を持つ  $n \times n$  三角帯行列  $A$  の、1- ノルムの値、または Frobenius ノルムの値、または無限ノルムの値、または最大絶対値の成分を返す。

関数の戻り値は `val` に格納される。

`val = max(abs( $A_{ij}$ ))`、`norm = 'M'` または `'m'` の場合  
       = `norm1(A)`、`norm = '1'` または `'O'` または `'o'` の場合  
       = `normI(A)`、`norm = 'I'` または `'i'` の場合  
       = `normF(A)`、`norm = 'F'`、`'f'`、`'E'` または `'e'` の場合

`norm1` は行列の 1- ノルム ( 最大列合計 )、`normI` は行列の無限ノルム ( 最大行合計 )、`normF` は行列の Frobenius ノルム ( 二乗和の平方根 ) を示す。`max(abs( $A_{ij}$ ))` は行列ノルムではない点に注意する。

## 入力パラメータ

`norm`                `CHARACTER*1`。ルーチン ?lantb が返す値を上述のように指定する。

<i>uplo</i>	CHARACTER*1。行列 <i>A</i> が上三角か下三角かを指定する。 = 'U' の場合、上三角 = 'L' の場合、下三角。
<i>diag</i>	CHARACTER*1。行列 <i>A</i> が単位三角かどうかを指定する。 = 'N' の場合、単位三角ではない。 = 'U' の場合、単位三角。
<i>n</i>	INTEGER。行列 <i>A</i> の次数。 $n \geq 0$ $n = 0$ と指定した場合、 <i>?lantb</i> はゼロに設定される。
<i>k</i>	INTEGER。 <i>uplo</i> = 'U' の場合は行列 <i>A</i> の優対角成分の数、 <i>uplo</i> = 'L' の場合は行列 <i>A</i> の劣対角成分の数。 $k \geq 0$
<i>ab</i>	REAL( <i>slantb</i> の場合) DOUBLE PRECISION( <i>dlantb</i> の場合) COMPLEX( <i>clantb</i> の場合) COMPLEX*16( <i>zlantb</i> の場合) 配列、次元は ( <i>ldab</i> , <i>n</i> )。上または下三角帯行列 <i>A</i> で、 <i>ab</i> の最初の <i>k</i> +1 行に格納する。 <i>A</i> の <i>j</i> 番目の列は配列 <i>ab</i> の <i>j</i> 番目の列に次のように格納する。 <i>uplo</i> = 'U' の場合、 $\max(1, j-k) \leq i \leq j$ に対して $ab(k+1+i-j, j) = a(i, j)$ <i>uplo</i> = 'L' の場合、 $j \leq i \leq \min(n, j+k)$ に対して $ab(1+i-j, j) = a(i, j)$ <i>diag</i> = 'U' の場合、行列 <i>A</i> の対角成分に対応する配列 <i>ab</i> の成分は参照されないが、1 として仮定される。
<i>ldab</i>	INTEGER。配列 <i>ab</i> のリーディング・ディメンション。 $ldab \geq k+1$
<i>work</i>	REAL( <i>slantb</i> と <i>clantb</i> の場合) DOUBLE PRECISION( <i>dlantb</i> と <i>zlantb</i> の場合) ワークスペース配列、次元は ( <i>lwork</i> )。norm='I' のとき $lwork \geq n$ 。 それ以外では <i>work</i> は参照されない。

## 出力パラメータ

<i>val</i>	REAL( <i>slantb</i> / <i>clantb</i> の場合) DOUBLE PRECISION( <i>dlantb</i> / <i>zlantb</i> の場合) 関数の戻り値。
------------	---

## ?lantp

圧縮形式で与えられる対称行列の 1- ノルムの値、  
Frobenius ノルムの値、無限ノルムの値、あるいは  
最大絶対値の成分を返す。

### 構文

```
val = slantp( norm, uplo, diag, n, ap, work )
val = dlantp( norm, uplo, diag, n, ap, work )
val = clantp( norm, uplo, diag, n, ap, work )
val = zlantp( norm, uplo, diag, n, ap, work )
```

### 説明

関数 ?lantp は、圧縮形式で与えられる三角行列  $A$  の、1- ノルムの値、または Frobenius ノルムの値、または無限ノルムの値、または最大絶対値の成分を返す。

関数の戻り値は `val` に格納される。

`val = max(abs( $A_{ij}$ ))`、`norm = 'M'` または `'m'` の場合

= `norm1(A)`、`norm = '1'` または `'O'` または `'o'` の場合

= `normI(A)`、`norm = 'I'` または `'i'` の場合

= `normF(A)`、`norm = 'F', 'f', 'E'` または `'e'` の場合

`norm1` は行列の 1- ノルム ( 最大列合計 )、`normI` は行列の無限ノルム ( 最大行合計 )、`normF` は行列の Frobenius ノルム ( 二乗和の平方根 ) を示す。`max(abs( $A_{ij}$ ))` は行列ノルムではない点に注意する。

### 入力パラメータ

<code>norm</code>	CHARACTER*1。ルーチン ?lantp が返す値を上述のように指定する。
<code>uplo</code>	CHARACTER*1。行列 $A$ が上三角か下三角かを指定する。 = 'U' の場合、上三角 = 'L' の場合、下三角。
<code>diag</code>	CHARACTER*1。行列 $A$ が単位三角かどうかを指定する。 = 'N' の場合、単位三角ではない。 = 'U' の場合、単位三角。

<code>n</code>	INTEGER。行列 $A$ の次数。 $n \geq 0$ $n = 0$ と指定した場合、 <code>?lantrp</code> はゼロに設定される。
<code>ap</code>	REAL( <code>slantrp</code> の場合 ) DOUBLE PRECISION( <code>dlantrp</code> の場合 ) COMPLEX( <code>clantrp</code> の場合 ) COMPLEX*16( <code>zlantrp</code> の場合 ) 配列、次元は $(n(n+1)/2)$ 。上三角または下三角帯行列 $A$ で、線形配列にカラム方向に圧縮されている。 $A$ の $j$ 番目の列は配列 <code>ap</code> に次のように格納する。 <code>uplo = 'U'</code> の場合、 $1 \leq i \leq j$ に対して $ap(i + (j-1)j/2) = A(i,j)$ <code>uplo = 'L'</code> の場合、 $j \leq i \leq n$ に対して $ap(i + (j-1)(2n-j)/2) = A(i,j)$ <code>diag = 'U'</code> の場合、行列 $A$ の対角成分に対応する配列 <code>ap</code> の成分は参照されないが、1 として仮定される。
<code>work</code>	REAL( <code>slantrp</code> と <code>clantrp</code> の場合 ) DOUBLE PRECISION( <code>dlantrp</code> と <code>zlantrp</code> の場合 ) ワークスペース配列、次元は $(lwork)$ 。 <code>norm = 'I'</code> のとき $lwork \geq n$ 。 それ以外では <code>work</code> は参照されない。

## 出力パラメータ

<code>val</code>	REAL( <code>slantrp/clantrp</code> の場合 ) DOUBLE PRECISION( <code>dlantrp/zlantrp</code> の場合 ) 関数の戻り値。
------------------	---

---

## ?lantr

台形または三角行列の  $l_1$ - ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。

---

### 構文

```
val = slantr( norm, uplo, diag, m, n, a, lda, work )
val = dlantr( norm, uplo, diag, m, n, a, lda, work )
val = clantr( norm, uplo, diag, m, n, a, lda, work )
val = zlantr( norm, uplo, diag, m, n, a, lda, work )
```

## 説明

関数 `?lantr` は、台形または三角行列  $A$  の、1- ノルムの値、または Frobenius ノルムの値、または無限ノルムの値、または最大絶対値の成分を返す。

関数の戻り値は `val` に格納される。

`val = max(abs(Aij))`、`norm = 'M'` または `'m'` の場合

`= norm1(A)`、`norm = '1'` または `'O'` または `'o'` の場合

`= normI(A)`、`norm = 'I'` または `'i'` の場合

`= normF(A)`、`norm = 'F', 'f', 'E'` または `'e'` の場合

`norm1` は行列の 1- ノルム ( 最大列合計 )、`normI` は行列の無限ノルム ( 最大行合計 )、`normF` は行列の Frobenius ノルム ( 二乗和の平方根 ) を示す。`max(abs(Aij))` は行列ノルムではない点に注意する。

## 入力パラメータ

`norm` CHARACTER\*1。ルーチン `?lantr` が返す値を上述のように指定する。

`uplo` CHARACTER\*1。行列  $A$  が上三角か下三角かを指定する。  
`= 'U'` の場合、上台形  
`= 'L'` の場合、上台形。  
 $m = n$  の場合、 $A$  は台形ではなく三角であることを注意する。

`diag` CHARACTER\*1。行列  $A$  が単位三角かどうかを指定する。  
`= 'N'` の場合、単位対角ではない。  
`= 'U'` の場合、単位対角。

`m` INTEGER。行列  $A$  の行数。 $m \geq 0$ 、`uplo = 'U'` の場合はさらに  $m \leq n$   $m = 0$  と指定した場合、`?lantr` はゼロに設定される。

`n` INTEGER。行列  $A$  の列数。 $n \geq 0$ 、`uplo = 'L'` の場合はさらに  $n \leq m$   $n = 0$  と指定した場合、`?lantr` はゼロに設定される。

`a` REAL ( `slantr` の場合 )  
DOUBLE PRECISION ( `dlantr` の場合 )  
COMPLEX ( `clantr` の場合 )  
COMPLEX\*16 ( `zlantr` の場合 )  
配列、次元は  $(lda, N)$ 。  
台形行列  $A$  ( $m = n$  の場合  $A$  は三角 )。  
`uplo = 'U'` の場合、配列 `a` の先頭の  $m \times n$  上台形部分には上台形行列を格納する。厳密な下三角部分は参照されない。

`uplo='L'` の場合、配列 `a` の先頭の  $m \times n$  上台形部分には上台形行列を格納する。厳密な下三角部分は参照されない。`diag='U'` の場合、`a` の対角成分は参照されず 1 と仮定される点に注意する。

`lda` INTEGER。配列 `a` のリーディング・ディメンジョン。  
 $lda \geq \max(m, 1)$

`work` REAL (`slantr/clantrp` の場合)  
 DOUBLE PRECISION (`dlantr/zlantr` の場合)  
 ワークスペース配列、次元は (`lwork`)。 `norm='I'` のとき  $lwork \geq m$ 。  
 それ以外では `work` は参照されない。

## 出力パラメータ

`val` REAL (`slantr/clantrp` の場合)  
 DOUBLE PRECISION (`dlantr/zlantr` の場合)  
 関数の戻り値。

## ?lanv2

$2 \times 2$  実数非対称行列の Schur 因子分解を標準形式で計算する。

### 構文

```
call slanv2( a, b, c, d, rt1r, rt1i, rt2r, rt2i, cs, sn )
call dlanv2( a, b, c, d, rt1r, rt1i, rt2r, rt2i, cs, sn )
```

### 説明

このルーチンは  $2 \times 2$  実数非対称行列の Schur 因子分解を標準形式で計算する。

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} cs & -sn \\ sn & cs \end{bmatrix} \begin{bmatrix} aa & bb \\ cc & dd \end{bmatrix} \begin{bmatrix} cs & sn \\ -sn & cs \end{bmatrix}$$

ここで、次のいずれかである。

1.  $aa$  と  $dd$  が行列の実数固有値となるように  $cc = 0$ 、または、
2.  $aa \sqrt{bb*cc}$  が複素共役固有値となるように  $aa = dd$  と  $bb*cc < 0$  ±

このルーチンは、実数固有値の計算において桁落ち誤差のリスクを低減するために、また可能ならば  $\text{abs}(\text{rt1r}) \geq \text{abs}(\text{rt2r})$  を保証するために調整されている。

### 入力パラメータ

$a, b, c, d$       REAL (slanv2 の場合)  
                   DOUBLE PRECISION (dlanv2 の場合)  
                   入力行列の成分。

### 出力パラメータ

$a, b, c, d$       標準化された Schur 形式の成分で上書きされる。  
 $\text{rt1r}, \text{rt1i},$   
                    $\text{rt2r}, \text{rt2i},$   
                   REAL (slanv2 の場合)  
                   DOUBLE PRECISION (dlanv2 の場合)  
                   固有値の実数部と虚数部。固有値が複素共役ペアの場合、 $\text{rt1i} > 0$   
  
 $cs, sn$           REAL (slanv2 の場合)  
                   DOUBLE PRECISION (dlanv2 の場合)  
                   回転行列のパラメータ

---

## ?lapll

2 つのベクトルの線形従属性を測定する。

---

### 構文

```
call slapll( n, x, incx, y, incy, ssmin )
call dlapll( n, x, incx, y, incy, ssmin )
call clapll( n, x, incx, y, incy, ssmin )
call zlapll( n, x, incx, y, incy, ssmin )
```

### 説明

長さ  $n$  の 2 つの列ベクトル  $x$  と  $y$  が与えられ、  
 $A = (x \ y)$  を  $n \times 2$  行列とする。

ルーチン `?lap11` は最初に  $A$  の  $QR$  因子分解を  $A = QR$  として計算し、次に  $2 \times 2$  上三角行列  $R$  の SVD を計算する。 $R$  の小さい方の特異値は `ssmin` で返され、ベクトル  $x$  と  $y$  の線形従属性の測定結果として使用される。

### 入力パラメータ

<code>n</code>	INTEGER。ベクトル $x$ およびベクトル $y$ の長さ。
<code>x</code>	REAL ( <code>slap11</code> の場合 ) DOUBLE PRECISION ( <code>dlap11</code> の場合 ) COMPLEX ( <code>clap11</code> の場合 ) COMPLEX*16 ( <code>zlap11</code> の場合 ) 配列、次元は $(1+(n-1)incx)$ 。 $x$ には $n$ -ベクトル $x$ を格納する。
<code>y</code>	REAL ( <code>slap11</code> の場合 ) DOUBLE PRECISION ( <code>dlap11</code> の場合 ) COMPLEX ( <code>clap11</code> の場合 ) COMPLEX*16 ( <code>zlap11</code> の場合 ) 配列、次元は $(1+(n-1)incy)$ 。 $y$ には $n$ -ベクトル $y$ を格納する。
<code>incx</code>	INTEGER。 $x$ の連続する成分間の増分で、 $incx > 0$
<code>incy</code>	INTEGER。 $y$ の連続する成分間の増分で、 $incy > 0$

### 出力パラメータ

<code>x</code>	$x$ は上書きされる。
<code>y</code>	$y$ は上書きされる。
<code>ssmin</code>	REAL ( <code>slap11/clap11</code> の場合 ) DOUBLE PRECISION ( <code>dlap11/zlap11</code> の場合 ) $n \times 2$ の行列 $A = (x \ y)$ の最小固有値。

---

## ?lapmt

行列の列に対して順方向または逆方向置換を実行する。

---

### 構文

```
call slapmt( forwrd, m, n, x, ldx, k )
```



```
call dlapmt( forwrd, m, n, x, ldx, k )
call clapmt( forwrd, m, n, x, ldx, k )
call zlapmt( forwrd, m, n, x, ldx, k )
```

## 説明

ルーチン `?lapmt` は、整数  $1, \dots, n$  の置換  $k(1), k(2), \dots, k(n)$  で指定されたとおりに、 $m \times n$  行列  $X$  の列を再配置する。

`forwrd = .TRUE.` の場合、順方向置換である。

$j = 1, 2, \dots, n$  では、 $X(*, k(j))$  は  $X(*, j)$  に移動される。

`forwrd = .FALSE.` の場合、逆方向置換である。

$j = 1, 2, \dots, n$  では、 $X(*, j)$  は  $X(*, k(j))$  に移動される。

## 入力パラメータ

<code>forwrd</code>	LOGICAL。 <code>forwrd = .TRUE.</code> の場合、順方向置換。 <code>forwrd = .FALSE.</code> の場合、逆方向置換。
<code>m</code>	INTEGER。行列 $X$ の行数。 $m \geq 0$
<code>n</code>	INTEGER。行列 $X$ の列数。 $n \geq 0$
<code>x</code>	REAL ( <code>slapmt</code> の場合) DOUBLE PRECISION ( <code>dlapmt</code> の場合) COMPLEX ( <code>clapmt</code> の場合) COMPLEX*16 ( <code>zlapmt</code> の場合) 配列、次元は $(ldx, n)$ 。 $m \times n$ 行列 $X$ を格納する。
<code>ldx</code>	INTEGER。配列 $x$ のリーディング・ディメンジョン。 $ldx \geq \max(1, m)$
<code>k</code>	INTEGER。 配列、次元は $(n)$ 。 $k$ には置換ベクトルを格納する。

## 出力パラメータ

`x`  $x$  は置換された行列  $X$  で上書きされる。

### ?lapy2

$\text{sqrt}(x^2+y^2)$  を返す。

---

#### 構文

```
val = slapy2( x, y )  
val = dlapy2( x, y )
```

#### 説明

関数 ?lapy2 は、無用なオーバーフローや有害なアンダーフローを発生させないようにしながら、 $\text{sqrt}(x^2+y^2)$  を返す。

#### 入力パラメータ

$x, y$	REAL (slapy2 の場合 ) DOUBLE PRECISION (dlapy2 の場合 ) 入力値 $x$ と $y$ を指定する。
--------	--

#### 出力パラメータ

val	REAL (slapy2 の場合 ) DOUBLE PRECISION (dlapy2 の場合 ) 関数の戻り値。
-----	---

### ?lapy3

$\text{sqrt}(x^2+y^2+z^2)$  を返す。

---

#### 構文

```
val = slapy3( x, y, z )  
val = dlapy3( x, y, z )
```

#### 説明

関数 ?lapy3 は、無用なオーバーフローや有害なアンダーフローを発生させないようにしながら、 $\text{sqrt}(x^2+y^2+z^2)$  を返す。

### 入力パラメータ

*x*, *y*, *z*            REAL (slapy3 の場合 )  
                      DOUBLE PRECISION (dlapy3 の場合 )  
                      入力値 *x*、*y*、*z* を指定する。

### 出力パラメータ

*val*                 REAL (slapy3 の場合 )  
                      DOUBLE PRECISION (dlapy3 の場合 )  
                      関数の戻り値。

---

## ?laqgb

?gbequ で計算された行と列のスケール係数を使って一般帯行列をスケールリングする。

---

### 構文

```
call slaqgb( m, n, kl, ku, ab, ldab, r, c, rowcnd, colcnd, amax, equed )  
call dlaqgb( m, n, kl, ku, ab, ldab, r, c, rowcnd, colcnd, amax, equed )  
call claqgb( m, n, kl, ku, ab, ldab, r, c, rowcnd, colcnd, amax, equed )  
call zlaqgb( m, n, kl, ku, ab, ldab, r, c, rowcnd, colcnd, amax, equed )
```

### 説明

このルーチンは、*kl* 個の劣対角と *ku* 個の優対角を持つ一般  $m \times n$  帯行列 *A* を、ベクトル *r* と *c* に格納されている行と列のスケール係数を用いて平衡化する。

### 入力パラメータ

*m*                    INTEGER。行列 *A* の行数。  
                       $m \geq 0$

*n*                    INTEGER。行列 *A* の列数。  
                       $n \geq 0$

*kl*                   INTEGER。 *A* の帯内にある劣対角の数。  $kl \geq 0$

*ku*                   INTEGER。 *A* の帯内にある優対角の数。  $ku \geq 0$

<i>ab</i>	<p>REAL (slaqgb の場合 )</p> <p>DOUBLE PRECISION (dlaqgb の場合 )</p> <p>COMPLEX (claqgb の場合 )</p> <p>COMPLEX*16 (zlaqgb の場合 )</p> <p>配列、次元は (<i>ldab</i>,<i>n</i>)。行列 <i>A</i> を帯格納形式で行 1 から <i>kl+ku+1</i> に格納する。<i>A</i> の <i>j</i> 番目の列は配列 <i>ab</i> の <i>j</i> 番目の列に次のように格納する。</p> <p><math>\max(1, j-ku) \leq i \leq \min(m, j+kl)</math> に対して <math>ab(ku+1+i-j, j) = A(i, j)</math></p>
<i>ldab</i>	<p>INTEGER。配列 <i>ab</i> のリーディング・ディメンジョン。</p> <p><math>lda \geq kl+ku+1</math></p>
<i>amax</i>	<p>REAL (slaqgb/claqgb の場合 )</p> <p>DOUBLE PRECISION (dlaqgb/zlaqgb の場合 )</p> <p>最大行列成分の絶対値。</p>

## 出力パラメータ

<i>ab</i>	<p><i>A</i> と同じ格納形式の平衡化行列で上書きされる。</p> <p>平衡化された行列の形式は <i>equed</i> を参照のこと。</p>
<i>r, c</i>	<p>REAL (slaqgb/claqgb の場合 )</p> <p>DOUBLE PRECISION (dlaqgb/zlaqgb の場合 )</p> <p>配列 <i>r</i>(<i>m</i>), <i>c</i>(<i>n</i>)。それぞれ、<i>A</i> に対する行と列のスケール係数。</p>
<i>rowcnd</i>	<p>REAL (slaqgb/claqgb の場合 )</p> <p>DOUBLE PRECISION (dlaqgb/zlaqgb の場合 )</p> <p>最小の <i>r</i>(<i>i</i>)<i>t</i> を最大の <i>r</i>(<i>i</i>) で割った値。</p>
<i>colcnd</i>	<p>REAL (slaqgb/claqgb の場合 )</p> <p>DOUBLE PRECISION (dlaqgb/zlaqgb の場合 )</p> <p>最小の <i>c</i>(<i>i</i>)<i>t</i> を最大の <i>c</i>(<i>i</i>) で割った値。</p>
<i>equed</i>	<p>CHARACTER*1</p> <p>実行された平衡化の形式を表わす。</p> <p><i>equed</i> = 'N' の場合、平衡化は行われていない。</p> <p><i>equed</i> = 'R' の場合、行の平衡化、すなわち、<i>A</i> は <i>diag</i>(<i>r</i>) で事前乗算された。</p> <p><i>equed</i> = 'C' の場合、列の平衡化、すなわち、<i>A</i> は <i>diag</i>(<i>c</i>) で事後乗算された。</p> <p><i>equed</i> = 'B' の場合、行と列の平衡化、すなわち、<i>A</i> は <i>diag</i>(<i>r</i>)*<i>A</i>*<i>diag</i>(<i>c</i>) で置き換えられた。</p>

## アプリケーション・ノート

このルーチンは内部パラメータ *thresh*、*large*、*small* を使用する。*thresh* はしきい値で、行または列のスケール係数の比率にもとづいた行または列のスケーリングの実行判定に使用される。

*rowcnd* < *thresh* の場合は行スケーリングが実行され、*colcnd* < *thresh* の場合は列スケーリングが実行される。*large* と *small* はそれぞれしきい値で、最大行列成分の絶対サイズにもとづいた行スケーリングの実行判定に使用される。*amax* > *large* または *amax* < *small* の場合に行スケーリングが実行される。

## ?laqge

?geequ で計算された行と列のスケール係数を使って一般矩形行列をスケーリングする。

### 構文

```
call slaqge( m, n, a, lda, r, c, rowcnd, colcnd, amax, equed )
call dlaqge( m, n, a, lda, r, c, rowcnd, colcnd, amax, equed )
call claqge( m, n, a, lda, r, c, rowcnd, colcnd, amax, equed )
call zlaqge( m, n, a, lda, r, c, rowcnd, colcnd, amax, equed )
```

### 説明

このルーチンは、一般  $m \times n$  行列 *A* を、ベクトル *r* と *c* に格納されている行と列のスケール係数を用いて平衡化する。

### 入力パラメータ

<i>m</i>	INTEGER。行列 <i>A</i> の行数。 $m \geq 0$
<i>n</i>	INTEGER。行列 <i>A</i> の列数。 $n \geq 0$
<i>a</i>	REAL (slaqge の場合) DOUBLE PRECISION (dlaqge の場合) COMPLEX (claqge の場合) COMPLEX*16 (zlaqge の場合) 配列、次元は ( <i>lda</i> , <i>n</i> )。 $m \times n$ の行列 <i>A</i> を格納する。

<i>lda</i>	INTEGER。配列 <i>a</i> のリーディング・ディメンジョン。 $lda \geq \max(m, 1)$
<i>r</i>	REAL (slanqge/claqge の場合 ) DOUBLE PRECISION (dlaqge/zlaqge の場合 ) 配列、次元は ( <i>m</i> )。A に対する行スケール係数。
<i>c</i>	REAL (slanqge/claqge の場合 ) DOUBLE PRECISION (dlaqge/zlaqge の場合 ) 配列、次元は ( <i>n</i> )。A に対する列スケール係数。
<i>rowcnd</i>	REAL (slanqge/claqge の場合 ) DOUBLE PRECISION (dlaqge/zlaqge の場合 ) 最小の <i>r</i> ( <i>i</i> ) を最大の <i>r</i> ( <i>i</i> ) で割った値。
<i>colcnd</i>	REAL (slanqge/claqge の場合 ) DOUBLE PRECISION (dlaqge/zlaqge の場合 ) 最小の <i>c</i> ( <i>i</i> ) を最大の <i>c</i> ( <i>i</i> ) で割った値。
<i>amax</i>	REAL (slanqge/claqge の場合 ) DOUBLE PRECISION (dlaqge/ZLAQGE の場合 ) 最大行列成分の絶対値。

## 出力パラメータ

<i>a</i>	平衡化された行列で上書きされる。 平衡化された行列の形式は <i>equed</i> を参照のこと。
<i>equed</i>	CHARACTER*1 実行された平衡化の形式を表わす。 <i>equed</i> = 'N' の場合、平衡化は行われていない。 <i>equed</i> = 'R' の場合、行の平衡化、すなわち、A は <i>diag</i> ( <i>r</i> ) で事前乗算された。 <i>equed</i> = 'C' の場合、列の平衡化、すなわち、A は <i>diag</i> ( <i>c</i> ) で事後乗算された。 <i>equed</i> = 'B' の場合、行と列の平衡化、すなわち、A は <i>diag</i> ( <i>r</i> )*A* <i>diag</i> ( <i>c</i> ) で置き換えられた。

## アプリケーション・ノート

このルーチンは内部パラメータ *thresh*、*large*、*small* を使用する。*thresh* はしきい値で、行または列のスケール係数の比率にもとづいた行または列のスケールリングの実行判定に使用される。

*rowcnd* < *thresh* の場合は行スケールリングが実行され、*colcnd* < *thresh* の場合は列ス

ケーリングが実行される。*large* と *small* はそれぞれしきい値で、最大行列成分の絶対サイズにもとづいた行スケーリングの実行判定に使用される。*amax* > *large* または *amax* < *small* の場合に行スケーリングが実行される。

## ?laqp2

行列ブロックに対して列ピボット演算を用いた *QR* 因子分解を計算する。

### 構文

```
call slaqp2( m, n, offset, a, lda, jpvt, tau, vn1, vn2, work )
call dlaqp2( m, n, offset, a, lda, jpvt, tau, vn1, vn2, work )
call claqp2( m, n, offset, a, lda, jpvt, tau, vn1, vn2, work )
call zlaqp2( m, n, offset, a, lda, jpvt, tau, vn1, vn2, work )
```

### 説明

このルーチンはブロック  $A(offset+1:m, 1:n)$  に対して列ピボット演算を用いた *QR* 因子分解を計算する。ブロック  $A(1:offset, 1:n)$  はピボットされるが因子分解はされない。

### 入力パラメータ

<i>m</i>	INTEGER。行列 <i>A</i> の行数。 $m \geq 0$
<i>n</i>	INTEGER。行列 <i>A</i> の列数。 $n \geq 0$
<i>offset</i>	INTEGER。因子分解はしないがピボット演算すべき行列 <i>A</i> の行数。 $offset \geq 0$
<i>a</i>	REAL (slaqp2 の場合) DOUBLE PRECISION (dlaqp2 の場合) COMPLEX (claqp2 の場合) COMPLEX*16 (zlaqp2 の場合) 配列、次元は ( <i>lda</i> , <i>n</i> )。 $m \times n$ の行列 <i>A</i> を格納する。
<i>lda</i>	INTEGER。配列 <i>A</i> のリーディング・ディメンジョン。 $lda \geq \max(1, m)$

<i>jpvt</i>	INTEGER 配列、次元は ( <i>n</i> )。呼び出し時に、 <i>jpvt</i> ( <i>i</i> ) ≠ 0 の場合、 <i>A</i> の <i>i</i> 番目の列が <i>A</i> * <i>P</i> の先頭 (先頭の列) に置換される。 <i>jpvt</i> ( <i>i</i> ) = 0 の場合、 <i>A</i> の <i>i</i> 番目の列はフリー列になる。
<i>vn1, vn2</i>	REAL ( <i>slaqp2</i> / <i>claqp2</i> の場合 ) DOUBLE PRECISION ( <i>dlaqp2</i> / <i>zlaqp2</i> の場合 ) 配列、次元はそれぞれ、( <i>n</i> )。それぞれ、部分的な列ノルムと完全な列ノルムを持つベクトルを格納する。
<i>work</i>	REAL ( <i>slaqp2</i> の場合 ) DOUBLE PRECISION ( <i>dlaqp2</i> の場合 ) COMPLEX ( <i>claqp2</i> の場合 ) COMPLEX*16 ( <i>zlaqp2</i> の場合 ) ワークスペース配列、次元は ( <i>n</i> )。

## 出力パラメータ

<i>a</i>	ブロック <i>A</i> ( <i>offset</i> +1: <i>m</i> , 1: <i>n</i> ) の上三角には得られた三角係数が上書きされる。対角より下のブロック <i>A</i> ( <i>offset</i> +1: <i>m</i> , 1: <i>n</i> ) の成分は、配列 <i>tau</i> とともに、基本リフレクタの積として直交行列 <i>Q</i> を表現する。ブロック <i>A</i> (1: <i>offset</i> , 1: <i>n</i> ) はピボットされるが因子分解はされない。
<i>jpvt</i>	<i>jpvt</i> ( <i>i</i> ) = <i>k</i> の場合、 <i>A</i> の <i>k</i> 番目の列が <i>A</i> * <i>P</i> の <i>i</i> 番目の列になる。
<i>tau</i>	REAL ( <i>slaqp2</i> の場合 ) DOUBLE PRECISION ( <i>dlaqp2</i> の場合 ) COMPLEX ( <i>claqp2</i> の場合 ) COMPLEX*16 ( <i>zlaqp2</i> の場合 ) 配列、次元は (min( <i>m</i> , <i>n</i> ))。基本リフレクタのスカラ係数。
<i>vn1, vn2</i>	それぞれ部分的な列ノルムと完全な列ノルムを持つベクトルが格納される。



## ?laqps

BLAS レベル 3 を使って実数  $m \times n$  行列  $A$  に対して列ピボット演算を用いた  $QR$  因子分解のステップを計算する。

### 構文

```
call slaqps( m, n, offset, nb, kb, a, lda, jpvt, tau, vn1, vn2, auxv, f, ldf )
call dlaqps( m, n, offset, nb, kb, a, lda, jpvt, tau, vn1, vn2, auxv, f, ldf )
call claqps( m, n, offset, nb, kb, a, lda, jpvt, tau, vn1, vn2, auxv, f, ldf )
call zlaqps( m, n, offset, nb, kb, a, lda, jpvt, tau, vn1, vn2, auxv, f, ldf )
```

### 説明

このルーチンは、BLAS レベル 3 を使って、実数  $m \times n$  行列  $A$  に対して列ピボット演算を用いた  $QR$  因子分解を計算する。このルーチンは、行  $offset+1$  から始めて  $A$  の  $nb$  列の因子分解を試み、BLAS レベル 3 ルーチン ?gemm を使って行列全体を更新する。

大幅な桁落ちの発生によって ?laqps が  $nb$  列の因子分解を実行できない場合がある。実際に因子分解された列数は  $kb$  に返される。

ブロック  $A(1:offset, 1:n)$  はピボットされるが因子分解はされない。

### 入力パラメータ

$m$	INTEGER。行列 $A$ の行数。 $m \geq 0$
$n$	INTEGER。行列 $A$ の列数。 $n \geq 0$
$offset$	INTEGER。前のステップですでに因子分解された $A$ の行数。
$nb$	INTEGER。因子分解する列数。
$a$	REAL (slaqps の場合 ) DOUBLE PRECISION (dlaqps の場合 ) COMPLEX (claqps の場合 ) COMPLEX*16 (zlaqps の場合 ) 配列、次元は $(lda, n)$ 。 $m \times n$ の行列 $A$ を格納する。

<i>lda</i>	INTEGER。配列 <i>a</i> のリーディング・ディメンジョン。 $lda \geq \max(1, m)$
<i>jpvt</i>	INTEGER。 配列、次元は ( <i>n</i> )。 <i>jpvt</i> ( <i>i</i> ) = <i>k</i> の場合、フル行列 <i>A</i> の列 <i>k</i> は <i>AP</i> の位置 <i>i</i> に置換されている。
<i>vn1, vn2</i>	REAL( <i>slaqps</i> / <i>claqps</i> の場合) DOUBLE PRECISION( <i>dlaqps</i> / <i>zlaqps</i> の場合) 配列、次元はそれぞれ、( <i>n</i> )。それぞれ、部分的な列ノルムと完全な列ノルムを持つベクトルを格納する。
<i>a</i>	REAL( <i>slaqps</i> の場合) DOUBLE PRECISION( <i>dlaqps</i> の場合) COMPLEX( <i>claqps</i> の場合) COMPLEX*16( <i>zlaqps</i> の場合) 配列、次元は ( <i>nb</i> )。補助ベクトル。
<i>f</i>	REAL( <i>slaqps</i> の場合) DOUBLE PRECISION( <i>dlaqps</i> の場合) COMPLEX( <i>claqps</i> の場合) COMPLEX*16( <i>zlaqps</i> の場合) 配列、次元は ( <i>ldf</i> , <i>nb</i> )。行列 $F' = L * Y' * A$ 。
<i>ldf</i>	INTEGER。配列 <i>f</i> のリーディング・ディメンジョン。 $ldf \geq \max(1, n)$ 。

## 出力パラメータ

<i>kb</i>	INTEGER。実際に因子分解された列数。
<i>a</i>	ブロック <i>A</i> ( <i>offset</i> +1: <i>m</i> , 1: <i>kb</i> ) は得られた三角係数が上書きされる。また、ブロック <i>A</i> (1: <i>offset</i> , 1: <i>n</i> ) はピボットされているが因子分解はされていない。行列の残りの部分ブロック <i>A</i> ( <i>offset</i> +1: <i>m</i> , <i>kb</i> +1: <i>n</i> ) は更新されている。
<i>jpvt</i>	INTEGER。配列、次元は ( <i>n</i> )。 <i>jpvt</i> ( <i>i</i> ) = <i>k</i> の場合、フル行列 <i>A</i> の列 <i>k</i> は <i>AP</i> の位置 <i>i</i> に置換されている。
<i>tau</i>	REAL( <i>slaqps</i> の場合) DOUBLE PRECISION( <i>dlaqps</i> の場合) COMPLEX( <i>claqps</i> の場合) COMPLEX*16( <i>zlaqps</i> の場合) 配列、次元は ( <i>kb</i> )。基本リフレクタのスカラ係数。
<i>vn1, vn2</i>	それぞれ、部分的な列ノルムと完全な列ノルムを持つベクトル。

`auxv`            補助ベクトル。  
`f`                行列  $F' = L * Y' * A$ 。

## ?laqsb

?pbequ で計算されたスケール係数を用いて対称/  
 エルミート帯行列をスケールリングする。

### 構文

```
call slaqsb( uplo, n, kd, ab, ldab, s, scond, amax, equed )
call dlaqsb( uplo, n, kd, ab, ldab, s, scond, amax, equed )
call claqsb( uplo, n, kd, ab, ldab, s, scond, amax, equed )
call zlaqsb( uplo, n, kd, ab, ldab, s, scond, amax, equed )
```

### 説明

このルーチンはベクトル `s` に格納されているスケール係数を使って対称帯行列 `A` を平衡化する。

### 入力パラメータ

`uplo`            CHARACTER\*1。対称行列 `A` の上三角部分または下三角部分のどちらが格納されているかを指定する。  
                  `uplo = 'U'` の場合は、上三角。  
                  `uplo = 'L'` の場合は、下三角。

`n`                INTEGER。行列 `A` の次数。  
                   $n \geq 0$

`kd`              INTEGER。 `uplo = 'U'` の場合は行列 `A` の優対角成分の数、`uplo = 'L'` の場合は行列 `A` の劣対角成分の数。  $kd \geq 0$

`ab`              REAL (slaqsb の場合 )  
                  DOUBLE PRECISION (dlaqsb の場合 )  
                  COMPLEX (claqsb の場合 )  
                  COMPLEX\*16 (zlaqsb の場合 )  
                  配列、次元は  $(ldab, n)$ 。対称帯行列 `A` の上三角または下三角で、配列の最初の  $kd+1$  行に格納する。`A` の  $j$  番目の列は配列 `ab` の番目の列に次のように格納する。

	$uplo = 'U'$ の場合、 $\max(1, j-kd) \leq i \leq j$ に対して $ab(kd+1+i-j, j) = A(i, j)$ $uplo = 'L'$ の場合、 $j \leq i \leq \min(n, j+kd)$ に対して $ab(1+i-j, j) = A(i, j)$
<i>ldab</i>	INTEGER。配列 <i>ab</i> のリーディング・ディメンジョン。 $ldab \geq kd+1$ 。
<i>scond</i>	REAL (slaqsб/claqsb の場合 ) DOUBLE PRECISION (dlaqsб/ZLAQSB の場合 ) 最小の $s(i)$ を最大の $s(i)$ で割った値。
<i>amax</i>	REAL (slaqsб/claqsb の場合 ) DOUBLE PRECISION (dlaqsб/ZLAQSB の場合 ) 最大行列成分の絶対値。

## 出力パラメータ

<i>ab</i>	<i>info</i> = 0 の場合、帯行列 <i>A</i> のコレスキー因子分解 $A = U'U$ または $A = LL'$ で得られた三角係数 <i>U</i> または <i>L</i> が、 <i>A</i> と同じ格納形式で格納される。
<i>s</i>	REAL (slaqsб/claqsb の場合 ) DOUBLE PRECISION (dlaqsб/zlaqsб の場合 ) 配列、次元は ( <i>n</i> )。 <i>A</i> のスケール係数。
<i>equed</i>	CHARACTER*1 平衡化が行われたかどうかを示す。 <i>equed</i> = 'N' の場合は、平衡化は行われていない。 <i>equed</i> = 'Y' の場合は、平衡化が行われ、 <i>A</i> は $\text{diag}(s) * A * \text{diag}(s)$ で置き換えられている。

## アプリケーション・ノート

このルーチンは内部パラメータ *thresh*、*large*、*small* を使用する。*thresh* はしきい値で、スケール係数の比率にもとづいたスケーリングの実行判定に使用される。*scond* < *thresh* の場合にスケーリングが実行される。*large* と *small* はそれぞれしきい値で、最大行列成分の絶対サイズにもとづいたスケーリングの実行判定に使用される。*amax* > *large* または *amax* < *small* の場合にスケーリングが実行される。

## ?laqsp

?ppequ で計算されたスケール係数を用いて  
圧縮格納形式にある対称/エルミート行列を  
スケールリングする。

### 構文

```
call slaqsp( uplo, n, ap, s, scond, amax, equed )
call dlaqsp( uplo, n, ap, s, scond, amax, equed )
call claqsp( uplo, n, ap, s, scond, amax, equed )
call zlaqsp( uplo, n, ap, s, scond, amax, equed )
```

### 説明

ルーチン ?laqsp は、ベクトル *s* に格納されているスケール係数を使って対称行列 *A* を平衡化する。

### 入力パラメータ

<i>uplo</i>	CHARACTER*1。対称行列 <i>A</i> の上三角部分または下三角部分のどちらが格納されているかを指定する。 <i>uplo</i> = 'U' の場合は、上三角。 <i>uplo</i> = 'L' の場合は、下三角。
<i>n</i>	INTEGER。行列 <i>A</i> の次数。 $n \geq 0$
<i>ap</i>	REAL (slaqsp の場合) DOUBLE PRECISION (dlaqsp の場合) COMPLEX (claqsp の場合) COMPLEX*16 (zlaqsp の場合) 配列、次元は $(n(n+1)/2)$ 。線形配列に列方向に圧縮された対称行列 <i>A</i> の上三角または下三角を格納する。 <i>A</i> の <i>j</i> 番目の列は配列 <i>ap</i> に次のように格納する。 UPLO = 'U' の場合、 $1 \leq i \leq j$ に対して $ap(i + (j-1)j/2) = A(i,j)$ UPLO = 'L' の場合、 $j \leq i \leq n$ に対して $ap(i + (j-1)(2n-j)/2) = A(i,j)$
<i>s</i>	REAL (slaqsp/claqsp の場合) DOUBLE PRECISION (dlaqsp/zlaqsp の場合) 配列、次元は $(n)$ 。 <i>A</i> のスケール係数。

<i>scond</i>	REAL (slaqsp/claqsp の場合 ) DOUBLE PRECISION (dlaqsp/zlaqsp の場合 ) 最小の $s(i)$ を最大の $s(i)$ で割った値。
<i>amax</i>	REAL (slaqsp/claqsp の場合 ) DOUBLE PRECISION (dlaqsp/zlaqsp の場合 ) 最大行列成分の絶対値。

### 出力パラメータ

<i>ap</i>	A と同じ形式で平衡化された行列 $\text{diag}(s)*A*\text{diag}(s)$ で上書きされる。
<i>equed</i>	CHARACTER*1。 平衡化が行われたかどうかを示す。 <i>equed</i> = 'N' の場合は、平衡化は行われていない。 <i>equed</i> = 'Y' の場合は、平衡化が行われ、A は $\text{diag}(s)*A*\text{diag}(s)$ で置き換えられている。

### アプリケーション・ノート

このルーチンは内部パラメータ *thresh*、*large*、*small* を使用する。*thresh* はしきい値で、スケール係数の比率にもとづいたスケーリングの実行判定に使用される。*scond* < *thresh* の場合にスケーリングが実行される。*large* と *small* はそれぞれしきい値で、最大行列成分の絶対サイズにもとづいたスケーリングの実行判定に使用される。*amax* > *large* または *amax* < *small* の場合にスケーリングが実行される。

---

## ?laqsy

?poequ で計算されたスケール係数を用いて  
対称/エルミート行列をスケーリングする。

---

### 構文

```
call slaqsy( uplo, n, a, lda, s, scond, amax, equed )
call dlaqsy( uplo, n, a, lda, s, scond, amax, equed )
call claqsy( uplo, n, a, lda, s, scond, amax, equed )
call zlaqsy( uplo, n, a, lda, s, scond, amax, equed )
```

## 説明

このルーチンは、ベクトル  $s$  に格納されているスケール係数を使って対称行列  $A$  を平衡化する。

## 入力パラメータ

<i>uplo</i>	CHARACTER*1。対称行列 $A$ の上三角部分または下三角部分のどちらが格納されているかを指定する。 <i>uplo</i> = 'U' の場合は、上三角。 <i>uplo</i> = 'L' の場合は、下三角。
<i>n</i>	INTEGER。行列 $A$ の次数。 $n \geq 0$
<i>a</i>	REAL ( <i>slaqsy</i> の場合) DOUBLE PRECISION ( <i>dlaqsy</i> の場合) COMPLEX ( <i>claqsy</i> の場合) COMPLEX*16 ( <i>zlaqsy</i> の場合) 配列、次元は ( <i>lda</i> , <i>n</i> )。対称行列 $A$ を格納する。 <i>uplo</i> = 'U' の場合、 $a$ の先頭の $n \times n$ 上三角部分に行列 $A$ の上三角部分を格納する。 $a$ の厳密な下三角部分は参照されない。 <i>uplo</i> = 'L' の場合、 $a$ の先頭の $n \times n$ 下三角部分に行列 $A$ の下三角部分を格納する。 $a$ の厳密な上三角部分は参照されない。
<i>lda</i>	INTEGER。配列 $a$ のリーディング・ディメンジョン。 $lda \geq \max(n, 1)$
<i>s</i>	REAL ( <i>slaqsy/claqsy</i> の場合) DOUBLE PRECISION ( <i>dlaqsy/zlaqsy</i> の場合) 配列、次元は ( <i>n</i> )。 $A$ のスケール係数。
<i>scond</i>	REAL ( <i>slaqsy/claqsy</i> の場合) DOUBLE PRECISION ( <i>dlaqsy/zlaqsy</i> の場合) 最小の $s(i)$ を最大の $s(i)$ で割った値。
<i>amax</i>	REAL ( <i>slaqsy/claqsy</i> の場合) DOUBLE PRECISION ( <i>dlaqsy/zlaqsy</i> の場合) 最大行列成分の絶対値。

## 出力パラメータ

<i>a</i>	<i>eques</i> = 'Y' の場合、平衡化された行列、 $\text{diag}(s) * A * \text{diag}(s)$ が格納される。
----------	--

`equed` CHARACTER\*1。  
平衡化が行われたかどうかを示す。  
`equed = 'N'` の場合は、平衡化は行われていない。  
`equed = 'Y'` の場合は、平衡化が行われ、 $A$  は  $\text{diag}(s)*A*\text{diag}(s)$  で置き換えられている。

### アプリケーション・ノート

このルーチンは内部パラメータ `thresh`、`large`、`small` を使用する。`thresh` はしきい値で、スケール係数の比率にもとづいたスケーリングの実行判定に使用される。`scond < thresh` の場合にスケーリングが実行される。`large` と `small` はそれぞれしきい値で、最大行列成分の絶対サイズにもとづいたスケーリングの実行判定に使用される。`amax > large` または `amax < small` の場合にスケーリングが実行される。

---

## ?laqtr

実数準三角連立方程式または特殊な形式の  
複素準三角連立方程式を実数演算で解く。

---

### 構文

```
call slaqtr( ltran, lreal, n, t, ldt, b, w, scale, x, work, info )
call dlaqtr( ltran, lreal, n, t, ldt, b, w, scale, x, work, info )
```

### 説明

ルーチン ?laqtr は実数準三角連立方程式、  
 $\text{op}(T) * p = \text{scale} * c$ 、`lreal = .TRUE.` の場合

または複素準三角連立方程式、  
 $\text{op}(T + iB) * (p + iq) = \text{scale} * (c + id)$ 、`lreal = .FALSE.` の場合を実数演算で解き、  
ここで  $T$  は上準三角である。

`lreal = .FALSE.` の場合、 $T$  の最初の対角ブロックは  $1 \times 1$  でなければならない、  
また、 $B$  は次のような特殊な構造の行列である。



$$B = \begin{bmatrix} b_1 & b_2 & \dots & \dots & b_n \\ & w & & & \\ & & w & & \\ & & & \dots & \\ & & & & w \end{bmatrix}$$

$\text{op}(A) = A$  または  $A'$ 。ここで  $A'$  は行列  $A$  の共役転置である。

入力では、

$$x = \begin{bmatrix} c \\ d \end{bmatrix}, \text{ 出力では、 } x = \begin{bmatrix} p \\ q \end{bmatrix}$$

このルーチンはルーチン `?trsna` における条件数推定に用いられる。

### 入力パラメータ

<i>ltran</i>	LOGICAL。 <i>ltran</i> は共役転置のオプションを指定する。 = .FALSE. の場合、 $\text{op}(T + iB) = T + iB$ 、 = .TRUE. の場合、 $\text{op}(T + iB) = (T + iB)'$ 。
<i>lreal</i>	LOGICAL。 <i>lreal</i> は入力行列の構造を指定する。 = .FALSE. の場合、入力は複素、 = .TRUE. の場合、入力は実数。
<i>n</i>	INTEGER。 <i>n</i> は $T + iB$ の次数を指定する。 $n \geq 0$
<i>t</i>	REAL (slaqtr の場合 ) DOUBLE PRECISION (dlaqtr の場合 ) 配列、次元は ( <i>ldt</i> , <i>n</i> )。 <i>t</i> には行列を Schur 標準形で格納する。 <i>lreal</i> = .FALSE. の場合、 <i>t</i> の最初の対角ブロックは $1 \times 1$ でなければならない。
<i>ldt</i>	INTEGER。 行列 $T$ のリーディング・ディメンジョン。 $ldt \geq \max(1, n)$

<i>b</i>	REAL (slaqtr の場合 ) DOUBLE PRECISION (dlaqtr の場合 ) 配列、次元は ( <i>n</i> )。 <i>b</i> には上述のように行列 <i>B</i> を形成するための成分を格納する。 <i>lreal</i> = .TRUE. の場合、 <i>b</i> は参照されない。
<i>w</i>	REAL (slaqtr の場合 ) DOUBLE PRECISION (dlaqtr の場合 ) <i>w</i> には行列 <i>B</i> の対角成分を格納する。 <i>lreal</i> = .TRUE. の場合、 <i>w</i> は参照されない。
<i>x</i>	REAL (slaqtr の場合 ) DOUBLE PRECISION (dlaqtr の場合 ) 配列、次元は ( <i>2n</i> )。 <i>x</i> には連立方程式の右辺を格納する。
<i>work</i>	REAL (slaqtr の場合 ) DOUBLE PRECISION (dlaqtr の場合 ) ワークスペース配列、次元は ( <i>n</i> )。

## 出力パラメータ

<i>scale</i>	REAL (slaqtr の場合 ) DOUBLE PRECISION (dlaqtr の場合 ) <i>scale</i> にはスケール係数が格納される。
<i>x</i>	<i>x</i> は解で上書きされる。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = 1 の場合、非特異点を維持するために、ある対角 1×1 ブロックが小さい数 <i>smin</i> によって摂動されたことを示す。 <i>info</i> = 2 の場合、非特異点を維持するために、ある対角 2×2 ブロックが $\epsilon_1 \alpha \ln 2$ に含まれる小さい数によって摂動されたことを示す。




---

**注：**実行速度を考慮してこのルーチンでは入力のエラーをチェックしない。

---

## ?lar1v

三重対角行列  $ldlt - \sigma * I$  の行  $b1$  から  $bn$  にある部分行列に対して逆行列の ( スケールされた )  $r$  番目の列を計算する。

### 構文

```
call slar1v( n, b1, bn, sigma, d, l, ld, lld, gersch, z, ztz, mingma, r, isuppz,
            work )
call dlar1v( n, b1, bn, sigma, d, l, ld, lld, gersch, z, ztz, mingma, r, isuppz,
            work )
call clar1v( n, b1, bn, sigma, d, l, ld, lld, gersch, z, ztz, mingma, r, isuppz,
            work )
call zlar1v( n, b1, bn, sigma, d, l, ld, lld, gersch, z, ztz, mingma, r, isuppz,
            work )
```

### 説明

ルーチン ?lar1v は、三重対角行列  $LDL^T - \sigma * I$  の行  $b1$  から  $bn$  にある部分行列に対して、逆行列の ( スケールされた )  $r$  番目の列を計算する。  
この計算は次のようなステップで実行される。

1. 定常  $qd$  変換、 $LDL^T - \sigma * I = L(+ ) D(+ ) L(+ )^T$
2. 進行  $qd$  変換、 $LDL^T - \sigma * I = U(- ) D(- ) U(- )^T$ 、
3. 記変換の組み合わせと、大きさが最大 ( のうちの 1 つ ) となる逆行列の対角成分の場所を示すインデックスとして  $r$  を選択すれば、 $LDL^T - \sigma * I$  の逆行列の対角成分を計算する。
4. 逆行列の ( スケールされた )  $r$  番目の列を、定常変換の上部分と進行変換の下部分との組み合わせで得られるツイスト分解を使用して計算する。

### 入力パラメータ

$n$                     INTEGER。 行列  $LDL^T$  の次数。  
 $b1$                     INTEGER。  $LDL^T$  の部分行列の最初のインデックス。  
 $bn$                     INTEGER。  $LDL^T$  の部分行列の最後のインデックス。

<i>sigma</i>	REAL (slar1v/clar1v の場合 ) DOUBLE PRECISION (dlar1v/zlar1v の場合 ) シフト。初期時点の $r=0$ の場合、 <i>sigma</i> は $LDL^T$ の固有値の良好な近似でなければならない。
<i>l</i>	REAL (slar1v/clar1v の場合 ) DOUBLE PRECISION (dlar1v/zlar1v の場合 ) 配列、次元は $(n-1)$ 。単位二重対角行列 $L$ の $(n-1)$ 個の劣対角成分を成分 1 から $n-1$ に格納する。
<i>d</i>	REAL (slar1v/clar1v の場合 ) DOUBLE PRECISION (dlar1v/zlar1v の場合 ) 配列、次元は $(n)$ 。対角行列 $D$ の $n$ 個の対角成分。
<i>ld</i>	REAL (slar1v/clar1v の場合 ) DOUBLE PRECISION (dlar1v/zlar1v の場合 ) 配列、次元は $(n-1)$ 。 $n-1$ 個の成分 $L_i * D_i$
<i>lld</i>	REAL (slar1v/clar1v の場合 ) DOUBLE PRECISION (dlar1v/zlar1v の場合 ) 配列、次元は $(n-1)$ 。 $n-1$ 個の成分 $L_i * L_i * D_i$ 。
<i>gersch</i>	REAL (slar1v/clar1v の場合 ) DOUBLE PRECISION (dlar1v/zlar1v の場合 ) 配列、次元は $(2n)$ 。 $n$ 個の Gerschgorin 間隔。 $r$ がゼロとして入力された場合、これらは $r$ の初期検索を制限するために使用される。
<i>r</i>	INTEGER。 最初の入力では $r$ はゼロとする。出力では大きさにおいて最大の逆行列の対角成分の場所を示すインデックスとなる。その後の反復では出力と同じ値を入力に与えなければならない。
<i>work</i>	REAL (slar1v/clar1v の場合 ) DOUBLE PRECISION (dlar1v/zlar1v の場合 ) ワークスペース配列、次元は $(4n)$ 。

## 出力パラメータ

<i>z</i>	REAL (slar1v の場合 ) DOUBLE PRECISION (dlar1v の場合 ) COMPLEX (clar1v の場合 ) COMPLEX*16 (zlar1v の場合 ) 配列、次元 $(n)$ 。逆行列の ( スケールされた ) $r$ 番目の列。 $z(r)$ は 1 として返される。
----------	--

<code>ztz</code>	REAL (slar1v/clar1v の場合 ) DOUBLE PRECISION (dlar1v/zlar1v の場合 ) $z$ のノルムの平方。
<code>mingma</code>	REAL (slar1v/clar1v の場合 ) DOUBLE PRECISION (dlar1v/zlar1v の場合 ) $LDL^T - \sigma * I$ の逆行列の ( 大きさにおいて ) 最大の対角成分の逆数。
<code>r</code>	大きさでは最大となる逆行列の対角成分の場所を示すインデックス。
<code>isuppz</code>	INTEGER 配列、次元は (2)。 $z$ にあるベクトルのサポート情報、すなわち、ベクトル $z$ は成分 <code>isuppz(1)</code> から <code>isuppz(2)</code> のみ非ゼロ。

## ?lar2v

一連の  $2 \times 2$  対称/エルミート行列に、実数余弦と実数/複素正弦を持つ面回転のベクトルを両側から適用する。

### 構文

```
call slar2v( n, x, y, z, incx, c, s, incc )
call dlar2v( n, x, y, z, incx, c, s, incc )
call clar2v( n, x, y, z, incx, c, s, incc )
call zlar2v( n, x, y, z, incx, c, s, incc )
```

### 説明

ルーチン ?lar2v は、ベクトル  $x$ 、 $y$ 、 $z$  の成分で定義される一連の  $2 \times 2$  実数対称または複素エルミート行列に、実数余弦を持つ実数/複素面回転のベクトルを両側から適用する。 $i = 1, 2, \dots, n$  では、

$$\begin{bmatrix} x_i & z_i \\ \text{conjg}(z_i) & y_i \end{bmatrix} := \begin{bmatrix} c(i) & \text{conjg}(s(i)) \\ -s(i) & c(i) \end{bmatrix} \begin{bmatrix} x_i & z_i \\ \text{conjg}(z_i) & y_i \end{bmatrix} \begin{bmatrix} c(i) & -\text{conjg}(s(i)) \\ s(i) & c(i) \end{bmatrix}$$

### 入力パラメータ

$n$  INTEGER。適用する面回転の回数。

$x, y, z$	REAL (slar2v の場合 ) DOUBLE PRECISION (dlar2v の場合 ) COMPLEX (clar2v の場合 ) COMPLEX*16 (zlar2v の場合 ) 配列、次元は それぞれ、 $(1+(n-1)*incx)$ 。それぞれ、ベクトル $x, y, z$ を格納する。 $?lar2v$ のすべての型で $x$ と $y$ は実数とする。
$incx$	INTEGER。 $x, y, z$ の成分間の増分。 $incx > 0$
$c$	REAL (slar2v/clar2v の場合 ) DOUBLE PRECISION (dlar2v/zlar2v の場合 ) 配列、次元は $(1+(n-1)*incc)$ 。面回転の余弦。
$s$	REAL (slar2v の場合 ) DOUBLE PRECISION (dlar2v の場合 ) COMPLEX (clar2v の場合 ) COMPLEX*16 (zlar2v の場合 ) 配列、次元は $(1+(n-1)*incc)$ 。面回転の正弦。
$incc$	INTEGER。 $c$ と $s$ の成分間の増分。 $incc > 0$

### 出力パラメータ

$x, y, z$            ベクトル  $x, y, z$  は変換結果で上書きされる。

---

## ?larf

一般矩形行列に基本リフレクタを適用する。

---

### 構文

```
call slarf( side, m, n, v, incv, tau, c, ldc, work )
call dlarf( side, m, n, v, incv, tau, c, ldc, work )
call clarf( side, m, n, v, incv, tau, c, ldc, work )
call zlarf( side, m, n, v, incv, tau, c, ldc, work )
```

## 説明

このルーチンは、実数 / 複素基本リフレクタ  $H$  を、実数 / 複素  $m \times n$  行列  $C$  に、左または右のいずれかから適用する。 $H$  は次の形式で表現される。

$$H = I - \tau * v * v'$$

$\tau$  は実数 / 複素スカラーで、 $v$  は実数 / 複素ベクトルである。

$\tau = 0$  の場合、 $H$  は単位行列をとる。

clarf/zlarf で  $H'$  ( $H$  の共役転置) を適用するには  $\tau$  の代わりに  $\text{conjg}(\tau)$  を与える。

## 入力パラメータ

<i>side</i>	CHARACTER*1。 <i>side</i> = 'L' の場合、形式 $H * C$ <i>side</i> = 'R' の場合、形式 $C * H$ 。
<i>m</i>	INTEGER。行列 $C$ の行数。
<i>n</i>	INTEGER。行列 $C$ の列数。
<i>v</i>	REAL (slarf の場合) DOUBLE PRECISION (dlarf の場合) COMPLEX (clarf の場合) COMPLEX*16 (zlarf の場合) 配列、次元は $(1 + (m-1) * \text{abs}(\text{incv}))$ ( <i>side</i> = 'L' の場合) $(1 + (n-1) * \text{abs}(\text{incv}))$ ( <i>side</i> = 'R' の場合) $H$ の表現におけるベクトル $v$ 。 $\tau = 0$ の場合、 $v$ は使用されない。
<i>incv</i>	INTEGER。 $v$ の成分間の増分。 $\text{incv} \neq 0$
<i>tau</i>	REAL (slarf の場合) DOUBLE PRECISION (dlarf の場合) COMPLEX (clarf の場合) COMPLEX*16 (zlarf の場合) $H$ の表現における値 $\tau$ 。
<i>v</i>	REAL (slarf の場合) DOUBLE PRECISION (dlarf の場合) COMPLEX (clarf の場合) COMPLEX*16 (zlarf の場合) 配列、次元は $(ldc, n)$ 。 $m \times n$ の行列 $C$ を格納する。

<i>ldc</i>	INTEGER。配列 <i>c</i> のリーディング・ディメンジョン。 $ldc \geq \max(1, m)$
<i>work</i>	REAL (slarf の場合) DOUBLE PRECISION (dlarf の場合) COMPLEX (clarf の場合) COMPLEX*16 (zlarf の場合) ワークスペース配列、次元は ( <i>n</i> ) ( <i>side</i> = 'L' の場合) または ( <i>m</i> ) ( <i>side</i> = 'R' の場合)

### 出力パラメータ

<i>c</i>	<i>side</i> = 'L' の場合は行列 $H * C$ で、 <i>side</i> = 'R' の場合は行列 $C * H$ で上書きされる。
----------	---

---

## ?larfb

ブロック・リフレクタまたはその転置/共役転置  
を一般矩形行列に適用する。

---

### 構文

```
call slarfb( side, trans, direct, storev, m, n, k, v, ldv, t, ldt, c, ldc, work,  
            ldwork )  
call dlarfb( side, trans, direct, storev, m, n, k, v, ldv, t, ldt, c, ldc, work,  
            ldwork )  
call clarfb( side, trans, direct, storev, m, n, k, v, ldv, t, ldt, c, ldc, work,  
            ldwork )  
call zlarfb( side, trans, direct, storev, m, n, k, v, ldv, t, ldt, c, ldc, work,  
            ldwork )
```

### 説明

ルーチン ?larfb は、複素ブロック・リフレクタ  $H$  またはその転置  $H'$  を、複素  $m \times n$  行列  $C$  に、左または右のいずれかから適用する。



## 入力パラメータ

<i>side</i>	CHARACTER*1。 <i>side</i> = 'L' の場合、 $H$ または $H'$ を左から適用する。 <i>side</i> = 'R' の場合、 $H$ または $H'$ を右から適用する。
<i>trans</i>	CHARACTER*1。 <i>trans</i> = 'N' の場合、 $H$ を適用する (転置なし)。 <i>trans</i> = 'C' の場合、 $H'$ を適用する (共役転置)
<i>direct</i>	CHARACTER*1。基本リフレクタの積からどのように $H$ が表現されているかを示す。 <i>direct</i> = 'F' の場合、 $H = H(1) H(2) \dots H(k)$ (順方向) <i>direct</i> = 'B' の場合、 $H = H(k) \dots H(2) H(1)$ (逆方向)
<i>storev</i>	CHARACTER*1。リフレクタを定義するベクトルがどのように格納されているかを示す。 <i>storev</i> = 'C' の場合、列方向 <i>storev</i> = 'R' の場合、行方向。
<i>m</i>	INTEGER。行列 $C$ の行数。
<i>n</i>	INTEGER。行列 $C$ の列数。
<i>k</i>	INTEGER。行列 $T$ の次数 (積がブロック・リフレクタを定義する基本リフレクタの個数に等しい)。
<i>v</i>	REAL (slarfb の場合) DOUBLE PRECISION (dlarfb の場合) COMPLEX (clarfb の場合) COMPLEX*16 (zlarfb の場合) 配列、次元は <i>storev</i> = 'C' の場合では $(ldv, k)$ <i>storev</i> = 'R' および <i>side</i> = 'L' の場合では $(ldv, m)$ <i>storev</i> = 'R' および <i>side</i> = 'R' の場合では $(ldv, n)$ 行列 $V$ 。
<i>ldv</i>	INTEGER。 配列 $v$ のリーディング・ディメンジョン。 <i>storev</i> = 'C' および <i>side</i> = 'L' の場合、 $ldv \geq \max(1, m)$ ; <i>storev</i> = 'C' および <i>side</i> = 'R' の場合、 $ldv \geq \max(1, n)$ ; <i>storev</i> = 'R' の場合、 $ldv \geq k$
<i>t</i>	REAL (slarfb の場合) DOUBLE PRECISION (dlarfb の場合) COMPLEX (clarfb の場合)

	COMPLEX*16 (zlarfb の場合) 配列、次元は $(ldt, k)$ 。 ブロック・リフレクタの表現にある $k \times k$ 行列 $T$ の三角を格納する。
$ldt$	INTEGER。配列 $t$ のリーディング・ディメンジョン。 $ldt \geq k$
$c$	REAL (slarfb の場合) DOUBLE PRECISION (dlarfb の場合) COMPLEX (clarfb の場合) COMPLEX*16 (zlarfb の場合) 配列、次元は $(ldc, n)$ 。 $m \times n$ の行列 $C$ を格納する。
$ldc$	INTEGER。配列 $c$ のリーディング・ディメンジョン。 $ldc \geq \max(1, m)$
$work$	REAL (slarfb の場合) DOUBLE PRECISION (dlarfb の場合) COMPLEX (clarfb の場合) COMPLEX*16 (zlarfb の場合) ワークスペース配列、次元は $(ldwork, k)$ 。
$ldwork$	INTEGER。配列 $work$ のリーディング・ディメンジョン。 $side = 'L'$ の場合、 $ldwork \geq \max(1, n)$ 、 $side = 'R'$ の場合、 $ldwork \geq \max(1, m)$

## 出力パラメータ

$c$	終了時に、 $c$ は $H * C$ または $H' * C$ あるいは $C * H$ または $C * H'$ によって上書きされる。
-----	--

---

## ?larfg

基本リフレクタ (Householder 行列) を生成する

---

### 構文

```
call slarfg( n, alpha, x, incx, tau )
call dlarfg( n, alpha, x, incx, tau )
call clarfg( n, alpha, x, incx, tau )
```

```
call zlarfg( n, alpha, x, incx, tau )
```

## 説明

ルーチン `zlarfg` は、次を満たすような次数  $n$  の実数 / 複素基本リフレクタ  $H$  を生成する。

$$H' * \begin{bmatrix} \alpha \\ x \end{bmatrix} = \begin{bmatrix} \beta \\ 0 \end{bmatrix} \quad H' * H = I$$

$\alpha$  と  $\beta$  は、スカラ ( $\beta$  はすべての型で実数)、 $x$  は  $(n-1)$  成分で構成される実数 / 複素ベクトルである。 $H$  は次の形式で表現される。

$$H = I - \tau * \begin{bmatrix} 1 \\ v \end{bmatrix} * \begin{bmatrix} 1 & v' \end{bmatrix}$$

$\tau$  は実数 / 複素スカラ、 $v$  は実数 / 複素の  $(n-1)$  成分で構成されるベクトルである。  
`clarfg/zlarfg` では、 $H$  はエルミートではない点に注意する。

$x$  の成分がすべてゼロの場合 (かつ、複素型で  $\alpha$  が実数)、 $\tau = 0$  となり、 $H$  は単位行列をとる。

それ以外では、 $1 \leq \tau \leq 2$  (実数型の場合)、または、 $1 \leq \text{Re}(\tau) \leq 2$  かつ  $\text{abs}(\tau-1) \leq 1$  (複素型の場合)

## 入力パラメータ

$n$	INTEGER。基本リフレクタの次数。
$\alpha$	REAL (slarfg の場合) DOUBLE PRECISION (dlarfg の場合) COMPLEX (clarfg の場合) COMPLEX*16 (zlarfg の場合) $\alpha$ の値を格納する。
$x$	REAL (slarfg の場合) DOUBLE PRECISION (dlarfg の場合) COMPLEX (clarfg の場合)

COMPLEX\*16 (zlarfg の場合)  
 配列、次元は  $(1+(n-2)*abs(incx))$ 。  
 ベクトル  $x$  を格納する。

*incx*                    INTEGER。  
                           $x$  の成分間の増分。  $incx > 0$

## 出力パラメータ

*alpha*                  *beta* の値で上書きされる。  
*x*                        ベクトル  $v$  で上書きされる。  
*tau*                     REAL (slarfg の場合)  
                          DOUBLE PRECISION (dlarfg の場合)  
                          COMPLEX (clarfg の場合)  
                          COMPLEX\*16 (zlarfg の場合)  
                          *tau* の値を格納する。

---

## ?larft

ブロック・リフレクタ  $H = I - VTV^H$  の  
 三角係数  $T$  を生成する。

---

### 構文

```
call slarft( direct, storev, n, k, v, ldv, tau, t, ldt )
call dlarft( direct, storev, n, k, v, ldv, tau, t, ldt )
call clarft( direct, storev, n, k, v, ldv, tau, t, ldt )
call zlarft( direct, storev, n, k, v, ldv, tau, t, ldt )
```

### 説明

ルーチン ?larft は、次数  $n$  の実 / 複素ブロック・リフレクタ  $H$  の三角係数  $T$  を生成する。ブロック・リフレクタ  $H$  は  $k$  個の基本リフレクタの積として定義される

$direct = 'F'$  の場合、 $H = H(1)H(2) \dots H(k)$  と  $T$  は上三角である。

$direct = 'B'$  の場合、 $H = H(k) \dots H(2)H(1)$  と  $T$  は下三角である。

$storev = 'C'$  の場合、基本リフレクタ  $H(i)$  を定義するベクトルは配列  $v$  の  $i$  番目の列に格納され、 $H = I - V^*T^*V$  である。

$storev = 'R'$  の場合、基本リフレクタ  $H(i)$  を定義するベクトルは配列  $v$  の  $i$  番目の行に格納され、 $H = I - V^* T V$  である。

### 入力パラメータ

<i>direct</i>	CHARACTER*1。ブロック・リフレクタを生成するために基本リフレクタを乗算する次数を指定する。 = 'F': $H = H(1) H(2) \dots H(k)$ (順方向) = 'B': $H = H(k) \dots H(2) H(1)$ (逆方向)
<i>storev</i>	CHARACTER*1。基本リフレクタを定義するベクトルがどのように格納されているかを指定する。(次のアプリケーション・ノートも参照): = 'C' の場合、列方向。 = 'R' の場合、行方向。
<i>n</i>	INTEGER。ブロック・リフレクタ $H$ の次数。 $n \geq 0$
<i>k</i>	INTEGER。三角係数 $T$ の次数 (基本リフレクタの個数に等しい)。 $k \geq 1$
<i>v</i>	REAL (slarft の場合) DOUBLE PRECISION (dlarft の場合) COMPLEX (clarft の場合) COMPLEX*16 (zlarft の場合) 配列、次元は $storev = 'C'$ の場合は $(ldv, k)$ 、または $storev = 'R'$ の場合は $(ldv, n)$ 。 行列 $V$ 。
<i>ldv</i>	INTEGER。配列 $v$ のリーディング・ディメンジョン。 $storev = 'C'$ の場合、 $ldv \geq \max(1, n)$ 、 $storev = 'R'$ の場合、 $ldv \geq k$
<i>tau</i>	REAL (slarft の場合) DOUBLE PRECISION (dlarft の場合) COMPLEX (clarft の場合) COMPLEX*16 (zlarft の場合) 配列、次元は $(k)$ 。 $tau(i)$ には、基本リフレクタ $H(i)$ のスカラ係数が入っていないといけない。
<i>ldt</i>	INTEGER。出力配列 $t$ のリーディング・ディメンジョン。 $ldt \geq k$

## 出力パラメータ

$t$  REAL (slarft の場合)  
 DOUBLE PRECISION (dlarft の場合)  
 COMPLEX (clarft の場合)  
 COMPLEX\*16 (zlarft の場合)  
 配列、次元は  $(ldt, k)$ 。ブロック・リフレクタの  $k \times k$  三角係数  $T$ 。  
 $direct = 'F'$  の場合、 $T$  は上三角。 $direct = 'B'$  の場合、 $T$  は下三角。配  
 列の残りの部分は使用されない。

$v$  行列  $V$ 。

## アプリケーション・ノート

行列  $V$  の形状と  $H(i)$  を定義するベクトルの格納形式を、 $n=5$ 、 $k=3$  において次の図に  
 示す。1 に等しい成分は格納されない。対応する配列成分は変更されるが、終了時に復  
 元される。配列の残りの部分は使用されない。

$direct = 'F'$  および  $storev = 'C'$      $direct = 'F'$  および  $storev = 'R'$

$$\begin{bmatrix} 1 \\ v_1 & 1 \\ v_1 & v_2 & 1 \\ v_1 & v_2 & v_3 \\ v_1 & v_2 & v_3 \end{bmatrix} \qquad \begin{bmatrix} 1 & v_1 & v_1 & v_1 & v_1 \\ & 1 & v_2 & v_2 & v_2 \\ & & 1 & v_3 & v_3 \end{bmatrix}$$

$direct = 'B'$  および  $storev = 'C'$      $direct = 'B'$  および  $storev = 'R'$

$$\begin{bmatrix} v_1 & v_2 & v_3 \\ v_1 & v_2 & v_3 \\ 1 & v_2 & v_3 \\ & 1 & v_3 \\ & & 1 \end{bmatrix} \qquad \begin{bmatrix} v_1 & v_1 & 1 \\ v_2 & v_2 & v_2 & 1 \\ v_3 & v_3 & v_3 & v_3 & 1 \end{bmatrix}$$

## ?larfx

リフレクタが次数 $\leq 10$ を持つ場合、ループの繰返しを避けながら、基本リフレクタを一般矩形行列に適用する。

### 構文

```
call slarfx( side, m, n, v, tau, c, ldc, work )
call dlarfx( side, m, n, v, tau, c, ldc, work )
call clarfx( side, m, n, v, tau, c, ldc, work )
call zlarfx( side, m, n, v, tau, c, ldc, work )
```

### 説明

ルーチン ?larfx は、実数 / 複素基本リフレクタ  $H$  を、実数 / 複素  $m \times n$  行列  $C$  に、左辺または右辺から適用する。

$H$  は次の形式で表現される。

$$H = I - \tau * v * v'$$

$\tau$  は実数 / 複素スカラーで、 $v$  は実数 / 複素ベクトルである。

$\tau = 0$  の場合、 $H$  は単位行列をとる。

### 入力パラメータ

<i>side</i>	CHARACTER*1。 <i>side</i> ='L' の場合、形式 $H * C$ <i>side</i> ='R' の場合、形式 $C * H$ 。
<i>m</i>	INTEGER。行列 $C$ の行数。
<i>n</i>	INTEGER。行列 $C$ の列数。
<i>v</i>	REAL (slarfx の場合 ) DOUBLE PRECISION (dlarfx の場合 ) COMPLEX (clarfx の場合 ) COMPLEX*16 (zlarfx の場合 ) 配列、次元は <i>side</i> ='L' の場合 ( <i>m</i> ) <i>side</i> ='R' の場合 ( <i>n</i> )。 $H$ の表現におけるベクトル $v$ 。

<i>tau</i>	REAL (slarfx の場合 ) DOUBLE PRECISION (dlarfx の場合 ) COMPLEX (clarfx の場合 ) COMPLEX*16 (zlarfx の場合 ) <i>H</i> の表現における値 <i>tau</i> 。
<i>c</i>	REAL (slarfx の場合 ) DOUBLE PRECISION (dlarfx の場合 ) COMPLEX (clarfx の場合 ) COMPLEX*16 (zlarfx の場合 ) 配列、次元は ( <i>ldc</i> , <i>n</i> )。 $m \times n$ の行列 <i>c</i> を格納する。
<i>ldc</i>	INTEGER。配列 <i>c</i> のリーディング・ディメンション。 $lda \geq (1, m)$
<i>work</i>	REAL (slarfx の場合 ) DOUBLE PRECISION (dlarfx の場合 ) COMPLEX (clarfx の場合 ) COMPLEX*16 (zlarfx の場合 ) ワークスペース配列、次元は <i>side</i> = 'L' の場合 ( <i>n</i> ) <i>side</i> = 'R' の場合 ( <i>m</i> )。 <i>H</i> が次数 < 11 を持つ場合は <i>work</i> は参照されない。

## 出力パラメータ

<i>c</i>	<i>side</i> = 'L' の場合は行列 $h * C$ で、 <i>side</i> = 'R' の場合は行列 $C * H$ で上書きされる。
----------	---

---

## ?largv

実数余弦および実数 / 複素正弦を持つ  
面回転のベクトルを生成する。

---

### 構文

```
call slargv( n, x, incx, y, incy, c, incc )
call dlargv( n, x, incx, y, incy, c, incc )
call clargv( n, x, incx, y, incy, c, incc )
call zlargv( n, x, incx, y, incy, c, incc )
```



## 説明

このルーチンは、実数 / 複素ベクトル  $x$  と  $y$  の成分で決定される、実数余弦を持った実数 / 複素面回転のベクトルを生成する。

slargv/dlargv の場合

$$\begin{bmatrix} c(i) & s(i) \\ -s(i) & c(i) \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} = \begin{bmatrix} a_i \\ 0 \end{bmatrix}, \quad i = 1, 2, \dots, n \text{ の場合}$$

clargv/zlargv の場合

$$\begin{bmatrix} c(i) & s(i) \\ -\text{conjg}(s(i)) & c(i) \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} = \begin{bmatrix} r_i \\ 0 \end{bmatrix}, \quad i = 1, 2, \dots, n \text{ の場合}$$

ここで  $c(i)^2 + \text{abs}(s(i))^2 = 1$ 。また、次の規則が使用される (clartg/zlartg と同じだが、BLAS レベル 1 ルーチン crotg/zrotg とは異なる)。  $y_i = 0$  の場合、  $c(i) = 0$  かつ  $s(i) = 0$   $x_i = 0$  の場合、  $c(i) = 0$ 、かつ  $r_i$  が実数となるように  $s(i)$  が選択される。

## 入力パラメータ

$n$	INTEGER。適用する面回転の回数。
$x, y$	REAL (slargv の場合 ) DOUBLE PRECISION (dlargv の場合 ) COMPLEX (clargv の場合 ) COMPLEX*16 (zlargv の場合 ) 配列、次元はそれぞれ $(1 + (n-1)*incx)$ と $(1 + (n-1)*incy)$ 。 ベクトル $x, y$ を格納する。
$incx$	INTEGER。 $x$ の成分間の増分。 $incx > 0$
$incy$	INTEGER。 $y$ の成分間の増分。 $incy > 0$
$incc$	INTEGER。出力配列 $c$ の分間の増分。 $incc > 0$

### 出力パラメータ

$x$	$i = 1, \dots, n$ に対して、 $x(i)$ は $a_i$ (実数型) または $r_i$ (複素型) で上書きされる。
$y$	面回転の正弦 $s(i)$ で上書きされる。
$c$	REAL (slargv/clargv の場合) DOUBLE PRECISION (dlargv/zlargv の場合) 配列、次元は $(1+(n-1)*incc)$ 。面回転の余弦。

---

## ?larnv

乱数ベクトルを一様分布または正規分布で返す。

---

### 構文

```
call slarnv( idist, iseed, n, x )  
call dlarnv( idist, iseed, n, x )  
call clarnv( idist, iseed, n, x )  
call zlarnv( idist, iseed, n, x )
```

### 説明

ルーチン ?larnv は、 $n$  個の実数 / 複素乱数のベクトルを一様分布または正規分布で返す。

このルーチンは、一様分布 (0, 1) の実数乱数を生成するために、ベクトル化可能なコードを使った最大 128 のバッチ処理中に補助ルーチン [?laruv](#) を呼び出す。一様分布から正規分布への数値変換は Box-Muller 法を使用する。

### 入力パラメータ

$idist$             INTEGER。乱数の分布を指定する。  
                  slarnv と dlanrv の場合。  
                  = 1 の場合、一様分布 (0, 1)  
                  = 2 の場合、一様分布 (-1, 1)  
                  = 3 の場合、正規分布 (0, 1)。  
                  clarnv と zlanrv の場合。  
                  = 1 の場合、実数部分と虚数部分がそれぞれ一様分布 (0, 1)  
                  = 2 の場合、実数部分と虚数部分がそれぞれ一様分布 (-1, 1)

= 3 の場合、実数部分と虚数部分がそれぞれ正規分布 (0, 1)

= 4 の場合、円盤  $\text{abs}(z) < 1$  上で一様分布

= 5 の場合、円  $\text{abs}(z) = 1$  上で一様分布。

*iseed* INTEGER。  
配列、次元は (4)。  
乱数生成器の種 (シード) を与える。配列成分は 0 から 4095 の間でなければならない。また *iseed*(4) は奇数でなければならない。

*n* INTEGER。生成する乱数の個数。

### 出力パラメータ

*x* REAL (slarnv の場合)  
DOUBLE PRECISION (dlarnv の場合)  
COMPLEX (clarnv の場合)  
COMPLEX\*16 (zlarnv の場合)  
配列、次元は (*n*)。生成された乱数。

*iseed* 更新された種。

## ?larrb

より高い精度で固有値を見つけるための  
限定二分法を与える。

### 構文

```
call slarrb( n, d, l, ld, lld, ifirst, ilast, sigma, reltol, w, wgap, werr,
            work, iwork, info )
```

```
call dlarrb( n, d, l, ld, lld, ifirst, ilast, sigma, reltol, w, wgap, werr,
            work, iwork, info )
```

### 説明

このルーチンは、比較的安定な表現 (RRR) の  $LDL^T$  が与えられた状態で、より高い精度で  $LDL^T$  の固有値  $w(\text{ifirst})$  から  $w(\text{ilast})$  をを見つけるために、「限定された」二分法を実行する。間隔 [ 左, 右 ] は、それらの中点と部分幅を配列 *w* と *werr* にそれぞれ格納することによって維持される。

## 入力パラメータ

<i>n</i>	INTEGER。行列の次数。
<i>d</i>	REAL (slarrb の場合 ) DOUBLE PRECISION (dlarrb の場合 ) 配列、次元は ( <i>n</i> )。対角行列 <i>D</i> の <i>n</i> 個の対角成分。
<i>l</i>	REAL (slarrb の場合 ) DOUBLE PRECISION (dlarrb の場合 ) 配列、次元は ( <i>n</i> -1)。単位二重対角行列 <i>L</i> の <i>n</i> -1 個の劣対角成分。
<i>ld</i>	REAL (slarrb の場合 ) DOUBLE PRECISION (dlarrb の場合 ) 配列、次元は ( <i>n</i> -1)。 <i>n</i> -1 個の成分 $L_i * D_i$
<i>lld</i>	REAL (slarrb の場合 ) DOUBLE PRECISION (dlarrb の場合 ) 配列、次元は ( <i>n</i> -1)。 <i>n</i> -1 個の成分 $L_i * L_i * D_i$ 。
<i>ifirst</i>	INTEGER。クラスタ内の最初の固有値のインデックス。
<i>ilast</i>	INTEGER。クラスタ内の最後の固有値のインデックス。
<i>sigma</i>	REAL (slarrb の場合 ) DOUBLE PRECISION (dlarrb の場合 ) $LDL^T$ を形成するために使用されたシフト。(?larrrf を参照)
<i>reltol</i>	REAL (slarrb の場合 ) DOUBLE PRECISION (dlarrb の場合 ) 相対許容値。
<i>w</i>	REAL (slarrb の場合 ) DOUBLE PRECISION (dlarrb の場合 ) 配列、次元は ( <i>n</i> )。 <i>w</i> ( <i>ifirst</i> ) から <i>w</i> ( <i>ilast</i> ) には $LDL^T$ の固有値に対応した推定を格納する。
<i>wgap</i>	REAL (slarrb の場合 ) DOUBLE PRECISION (dlarrb の場合 ) 配列、次元は ( <i>n</i> )。 $LDL^T$ の固有値間の隔たり。
<i>werr</i>	real (slarrb の場合 ) DOUBLE PRECISION (dlarrb の場合 ) 配列、次元は ( <i>n</i> )。 <i>werr</i> ( <i>ifirst</i> ) から <i>werr</i> ( <i>ilast</i> ) には推定 <i>w</i> ( <i>ifirst</i> ) から <i>w</i> ( <i>ilast</i> ) の誤差を格納する。

<code>work</code>	REAL (slarrb の場合 ) DOUBLE PRECISION (dlarrb の場合 ) ワークスペース配列。このパラメータはルーチン内では使用されない。
<code>iwork</code>	INTEGER。 ワークスペース配列、次元は (2n)

### 出力パラメータ

<code>w</code>	固有値の推定の精度を改善した結果が上書きされる。
<code>wgap</code>	微小な隔たりが変更される。
<code>werr</code>	推定 $w(\text{ifirst})$ から $w(\text{ilast})$ の誤差に対して精度を改善した結果が上書きされる。
<code>info</code>	INTEGER。 エラーフラグ。ルーチンではこのパラメータは設定されない。

## ?larre

三重対角行列  $T$  が与えられたとき、小さい非対角成分をゼロに設定し、縮退されていないブロック  $T_i$  に対して基本表現と固有値を探す。

### 構文

```
call slarre( n, d, e, tol, nsplit, isplit, m, w, woff, gersch, work, info )
call dlarre( n, d, e, tol, nsplit, isplit, m, w, woff, gersch, work, info )
```

### 説明

三重対角行列  $T$  が与えられたとき、このルーチンは「小さい」非対角成分をゼロに設定し、縮退されていないブロック  $T_i$  に対して次を探す。

- 数  $\sigma_i$
- 基本表現  $T_i - \sigma_i I = L_i D_i L_i^T$
- 各  $L_i D_i L_i^T$  の固有値

求められた表現と固有値は対称三重対角行列の固有値を計算するために ?stegr で使用される。現時点で基本表現は正定値または負定値として制限されており、定値行列の固有値は *dqds* アルゴリズムによって求められる ( サブルーチン ?lasq2)。また ?larre は、 $L_i D_i L_i^T$  に対する  $n$  個の Gerschgorin 間隔を出力する利点を持つ。

## 入力パラメータ

<i>n</i>	INTEGER。行列の次数。
<i>d</i>	REAL (slarre の場合 ) DOUBLE PRECISION (dlarre の場合 ) 配列、次元は ( $n$ )。三重対角行列 $T$ の $n$ 個の対角成分を格納する。
<i>e</i>	REAL (slarre の場合 ) DOUBLE PRECISION (dlarre の場合 ) 配列、次元は ( $n$ )。三重対角行列 $T$ の ( $n-1$ ) 個の劣対角成分を格納する。 $e(n)$ は設定する必要はない。
<i>tol</i>	REAL (slarre の場合 ) DOUBLE PRECISION (dlarre の場合 ) 分割のしきい値。入力において $ e(i)  < tol$ の場合、行列 $T$ は小さいブロックに分割される。
<i>nsplit</i>	INTEGER。ブロック $T$ を分割する数。 $1 \leq nsplit \leq n$ 。
<i>work</i>	REAL (slarre の場合 ) DOUBLE PRECISION (dlarre の場合 ) ワークスペース配列、次元は ( $4*n$ )。

## 出力パラメータ

<i>d</i>	対角行列 $D_i$ の $n$ 個の対角成分で上書きされる。
<i>e</i>	単位二重対角行列 $L_i$ の劣対角成分で上書きされる。
<i>isplit</i>	INTEGER。 配列、次元は ( $2n$ )。 $T$ を部分行列に分割した分割点。第 1 の部分行列は 1 から <i>isplit</i> (1) の行 / 列で構成され、第 2 の部分行列は <i>isplit</i> (1)+1 から <i>isplit</i> (2) の行 / 列で構成され、以下同様であり、 <i>nsplit</i> 番目は <i>isplit</i> ( <i>nsplit</i> -1)+1 から <i>isplit</i> ( <i>nsplit</i> ) = $n$ の行 / 列で構成される。
<i>m</i>	INTEGER。求められた ( すべての $L_i D_i L_i^T$ ) 固有値の総数。

<i>w</i>	REAL (slarre の場合 ) DOUBLE PRECISION (dlarre の場合 ) 配列、次元は (n)。最初の <i>m</i> 個の成分には固有値が格納される。各ブロック $L_i D_i L_i^T$ の各固有値は昇順でソートされる。
<i>woff</i>	REAL (slarre の場合 ) DOUBLE PRECISION (dlarre の場合 ) 配列、次元は (n)。 <i>nsplit</i> 基本点 $\sigma_i$
<i>gersch</i>	REAL (slarre の場合 ) DOUBLE PRECISION (dlarre の場合 ) 配列、次元は (2n)。n 個の Gerschgorin 間隔。
<i>info</i>	INTEGER。?lasq2 のエラーコード。

## ?larrf

少なくとも 1 つの固有値が相対的に分離されているような新しい比較的安定な表現を探す。

### 構文

```
call slarrf( n, d, l, ld, lld, ifirst, ilast, w, dplus, lplus, work, iwork,
            info )
call dlarrf( n, d, l, ld, lld, ifirst, ilast, w, dplus, lplus, work, iwork,
            info )
```

### 説明

初期表現  $LDL^T$  と ( 相対的に ) 値に近いそれらの固有値のクラス  $w(\text{ifirst})$ ,  $w(\text{ifirst}+1), \dots, w(\text{ilast})$  が与えられたとき、ルーチン ?larrf は、新しい比較的安定な表現を探す。

$$LDL^T - \sigma_i I = L(+ )D(+ )L(+ )^T$$

$L(+ )D(+ )L(+ )^T$  の少なくとも 1 つの固有値が相対的に分離されている。

### 入力パラメータ

<i>n</i>	INTEGER。行列の次数。
<i>d</i>	REAL (slarrf の場合 ) DOUBLE PRECISION (dlarrf の場合 ) 配列、次元は (n)。対角行列 <i>D</i> の n 個の対角成分。

<i>l</i>	REAL (slarrf の場合 ) DOUBLE PRECISION (dlarrf の場合 ) 配列、次元は (n-1)。単位二重対角行列 $L$ の (n-1) 個の劣対角成分。
<i>ld</i>	REAL (slarrf の場合 ) DOUBLE PRECISION (dlarrf の場合 ) 配列、次元は (n-1)。n-1 個の成分 $L_i * D_i$
<i>lld</i>	REAL (slarrf の場合 ) DOUBLE PRECISION (dlarrf の場合 ) 配列、次元は (n-1)。n-1 個の成分 $L_i * L_i * D_i$
<i>ifirst</i>	INTEGER。クラスタ内の最初の固有値のインデックス。
<i>ilast</i>	INTEGER。クラスタ内の最後の固有値のインデックス。
<i>w</i>	REAL (slarrf の場合 ) DOUBLE PRECISION (dlarrf の場合 ) 配列、次元は (n)。 $LDL^T$ の固有値を昇順で格納する。w( <i>ifirst</i> ) から w( <i>ilast</i> ) が比較的安定な表現のクラスタを形成する。
<i>sigma</i>	REAL (slarrf の場合 ) DOUBLE PRECISION (dlarrf の場合 ) $L(+)D(+)L(+)^T$ を形成するために使用されたシフト。
<i>work</i>	REAL (slarrf の場合 ) DOUBLE PRECISION (dlarrf の場合 ) ワークスペース配列。

## 出力パラメータ

<i>w</i>	w( <i>ifirst</i> ) から w( <i>ilast</i> ) は、 $L(+)D(+)L(+)^T$ の対応する固有値の推定で上書きされる。
<i>dplus</i>	REAL (slarrf の場合 ) DOUBLE PRECISION (dlarrf の場合 ) 配列、次元は (n)。対角行列 $D(+)$ の n 個の対角成分。
<i>lplus</i>	REAL (slarrf の場合 ) DOUBLE PRECISION (dlarrf の場合 ) 配列、次元は (n)。lplus の最初の (n-1) 個の成分には単位二重対角行列 $L(+)$ の劣対角成分が格納される。lplus(n) は <i>sigma</i> に設定される。



## ?larrv

$L$ 、 $D$ 、 $LDL^T$  の固有値が与えられたとき、三重対角行列  $T = LDL^T$  の固有ベクトルを計算する。

### 構文

```
call slarrv( n, d, l, isplit, m, w, iblock, gersch, tol, z, ldz, isuppz, work,
            iwork, info )
call dlarrv( n, d, l, isplit, m, w, iblock, gersch, tol, z, ldz, isuppz, work,
            iwork, info )
call clarrv( n, d, l, isplit, m, w, iblock, gersch, tol, z, ldz, isuppz, work,
            iwork, info )
call zlarrv( n, d, l, isplit, m, w, iblock, gersch, tol, z, ldz, isuppz, work,
            iwork, info )
```

### 説明

ルーチン ?larrv は、 $L$ 、 $D$ 、 $LDL^T$  の固有値が与えられたとき、三重対角行列  $T = LDL^T$  の固有ベクトルを計算する。入力固有値は  $L$  と  $D$  の成分に対して相対的に高い精度を持っていなければならない。目的とする出力精度は入力パラメータ *tol* で指定できる。

### 入力パラメータ

<i>n</i>	INTEGER。行列の次数。 $n \geq 0$
<i>d</i>	REAL (slarrv/clarrv の場合) DOUBLE PRECISION (dlarrv/zlarrv の場合) 配列、次元は ( <i>n</i> )。対角行列 $D$ の <i>n</i> 個の対角成分を格納する。
<i>l</i>	REAL (slarrv/clarrv の場合) DOUBLE PRECISION (dlarrv/zlarrv の場合) 配列、次元は ( <i>n</i> -1)。単位対角行列 $L$ の ( <i>n</i> -1) 個の劣対角成分を 1 の成分 1 から <i>n</i> -1 に格納する。 $l(n)$ を設定する必要はない。
<i>isplit</i>	INTEGER。 配列、次元は ( <i>n</i> )。 $T$ を部分行列に分割した分割点。第 1 の部分行列は 1 から <i>isplit</i> (1) の行 / 列で構成され、第 2 の部分行列は <i>isplit</i> (1)+1 から <i>isplit</i> (2) の行 / 列で構成され、以下同様。

<i>tol</i>	REAL (slarrv/clarrv の場合 ) DOUBLE PRECISION (dlarrv/zlarrv の場合 ) 固有値 / 固有ベクトルに対する絶対誤差許容値。 入力固有値の誤差は <i>tol</i> を上限としなければならない。 固有ベクトル出力は <i>tol</i> を上限とする誤差ノルムを持ち、また、異なる固有ベクトル間の内積は <i>tol</i> に制限される。 <i>tol</i> は $n * \text{eps} *  T $ 以上でなければならない、ここで <i>eps</i> はマシン精度、 $ T $ は三重対角行列の 1- ノルムである。
<i>m</i>	INTEGER。求められた固有値の個数。 $0 \leq m \leq n$ 。 <i>range</i> = 'A' の場合は $m = n$ 、 <i>range</i> = 'I' の場合は $m = iu - il + 1$
<i>w</i>	REAL (slarrv/clarrv の場合 ) DOUBLE PRECISION (dlarrv/zlarrv の場合 ) 配列、次元は ( <i>n</i> )。 <i>w</i> の最初の <i>m</i> 個の成分には固有ベクトルを計算する固有値を格納する。固有値は分割されたブロックごとにグループ分けし、ブロック内で最小から最大に並べられていなければならない ( <a href="#">?larre</a> の出力配列 <i>w</i> がここで要求される )。 <i>w</i> の誤差は <i>tol</i> を上限としなければならない。
<i>iblock</i>	INTEGER。 配列、次元は ( <i>n</i> )。 <i>w</i> 内の対応する固有値に関係した部分行列インデックス。固有値 $w(i)$ が第 1 の部分行列の先頭から属する場合は $iblock(i) = 1$ 、 $w(i)$ が第 2 の行列に属する場合は $iblock(i) = 2$ 、以下同様。
<i>ldz</i>	INTEGER。出力配列 <i>z</i> のリーディング・ディメンション。 $ldz \geq 1$ 、 <i>jobz</i> = 'V' の場合は $ldz \geq \max(1, n)$ 。
<i>work</i>	REAL (slarrv/clarrv の場合 ) DOUBLE PRECISION (dlarrv/zlarrv の場合 ) ワークスペース配列、次元は (13 <i>n</i> )。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は (6 <i>n</i> )。

## 出力パラメータ

<i>d</i>	<i>d</i> は上書きされることがある。
<i>l</i>	<i>l</i> は上書きされる。

*z* REAL (slarrv の場合)  
 DOUBLE PRECISION (dlarrv の場合)  
 COMPLEX (clarrv の場合)  
 COMPLEX\*16 (zlarrv の場合)  
 配列、次元は (ldz, max(1,m))。  
*jobz* = 'V' の場合、*info*=0 ならば、行列 *T* の正規直交固有ベクトルのうち、選択された固有値に対応するものが *z* の最初の *m* 列に格納される。すなわち、*w*(*i*) に対応する固有ベクトルが *z* の *i* 番目の列に入る。  
*jobz* = 'N' の場合、*z* は参照されない。



**注:** 配列 *z* には、max(1, *m*) 以上の列が提供されなければならない。  
*range* = 'V' の場合は、*m* の正確な値が事前にわからないため、上限値を使用する必要がある。

*isuppz* INTEGER。  
 配列、次元は (2\*max(1, *m*))。 *z* に入っている固有ベクトルのサポート情報、すなわち *z* に入っている非ゼロの成分を示すインデックス。*i* 番目の固有ベクトルは *isuppz*(2*i*-1) から *isuppz*(2*i*) までの成分のみ非ゼロである。

*info* INTEGER。  
*info*=0 の場合、正常に終了したことを示す。  
*info*=-*i* < 0 の場合、*i* 番目の引数の値が不正だったことを示す。  
*info* > 0 の場合、*info*=1 ならば ?larrb で内部エラーが発生したことを示す。  
*info*=2 ならば ?stein で内部エラーが発生したことを示す。

## ?lartg

実数余弦と実数/複素正弦を持つ面回転を生成する。

### 構文

```
call slartg( f, g, cs, sn, r )
```

```
call dlartg( f, g, cs, sn, r )
call clartg( f, g, cs, sn, r )
call zlartg( f, g, cs, sn, r )
```

## 説明

このルーチンは、次のように面回転を生成する。

$$\begin{bmatrix} cs & sn \\ -\text{conjg}(sn) & cs \end{bmatrix} \cdot \begin{bmatrix} f \\ g \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix}$$

ここで  $cs^2 + |sn|^2 = 1$

このルーチンは BLAS レベル 1 ルーチン [?rotg](#) の実行速度が遅く精度が高いバージョンであるが、次の点で相違がある。

slartg/dlartg の場合。

- $f$  と  $g$  は出力時に変更されない。
- $g = 0$  の場合、 $cs = 1$  と  $sn = 0$
- $f = 0$  かつ  $g \neq 0$  の場合、浮動小数点演算は実行されず  $cs = 0$  と  $sn = 1$  (対角にゼロがある場合、?bdsqr の動作を省く)。
- $f$  が大きさで  $g$  を超えた場合、 $cs$  は正となる。

clartg/zlartg の場合。

- $f$  と  $g$  は出力時に変更されない。
- $g = 0$  の場合、 $cs = 1$  と  $sn = 0$
- $f = 0$  の場合、 $cs = 0$ 、と  $r$  が実数になるように  $sn$  が選択される。

## 入力パラメータ

$f, g$	REAL (slartg の場合 )
	DOUBLE PRECISION (dlartg の場合 )
	COMPLEX ( lartg の場合 )
	COMPLEX*16 (zlartg の場合 )
	回転対象のベクトルの第 1 および第 2 成分。

**出力パラメータ**

<i>cs</i>	REAL (slartg/clartg の場合) DOUBLE PRECISION (dlartg/zlartg の場合) 回転の余弦。
<i>sn</i>	REAL (slartg の場合) DOUBLE PRECISION (dlartg の場合) COMPLEX (clartg の場合) COMPLEX*16 (zlartg の場合) 回転の正弦。
<i>r</i>	REAL (slartg の場合) DOUBLE PRECISION (dlartg の場合) COMPLEX (clartg の場合) COMPLEX*16 (zlartg の場合) 回転されたベクトルの非ゼロの成分。

**?lartv**

実数余弦と実数 / 複素制限を持つ面回転の  
ベクトルをベクトル対の成分に適用する。

**構文**

```
call slartv( n, x, incx, y, incy, c, s, incc )
call dlartv( n, x, incx, y, incy, c, s, incc )
call clartv( n, x, incx, y, incy, c, s, incc )
call zlartv( n, x, incx, y, incy, c, s, incc )
```

**説明**

このルーチンは、実数余弦を持つ実数 / 複素面回転を実数 / 複素ベクトル  $x$  と  $y$  の成分に適用する。  $i = 1, 2, \dots, n$  では、

$$\begin{bmatrix} x_i \\ y_i \end{bmatrix} := \begin{bmatrix} c(i) & s(i) \\ -\text{conjg}(s(i)) & c(i) \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix}$$

### 入力パラメータ

<i>n</i>	INTEGER。適用する面回転の回数。
<i>x</i> , <i>y</i>	REAL (slartv の場合 ) DOUBLE PRECISION (dlartv の場合 ) COMPLEX (clartv の場合 ) COMPLEX*16 (zlartv の場合 ) 配列、次元はそれぞれ $(1+(n-1)*incx)$ と $(1+(n-1)*incy)$ 入力ベクトル <i>x</i> と <i>y</i>
<i>incx</i>	INTEGER。 <i>x</i> の成分間の増分。 <i>incx</i> > 0
<i>incy</i>	INTEGER。 <i>y</i> の成分間の増分。 <i>incy</i> > 0
<i>c</i>	REAL (slartv/clartv の場合 ) DOUBLE PRECISION (dlartv/zlartv の場合 ) 配列、次元は $(1+(n-1)*incc)$ 。面回転の余弦。
<i>s</i>	REAL (slartv の場合 ) DOUBLE PRECISION (dlartv の場合 ) COMPLEX (clartv の場合 ) COMPLEX*16 (zlartv の場合 ) 配列、次元は $(1+(n-1)*incc)$ 。面回転の正弦。
<i>incc</i>	INTEGER。 <i>c</i> と <i>s</i> の成分間の増分。 <i>incc</i> > 0

### 出力パラメータ

<i>x</i> , <i>y</i>	回転されたベクトル <i>x</i> と <i>y</i>
---------------------	-------------------------------

---

## ?laruv

*n* 個の実数乱数のベクトルを一様分布で返す。

---

### 構文

```
call slaruv( iseed, n, x )  
call dlaruv( iseed, n, x )
```

## 説明

ルーチン `?laruv` は  $n$  個の実数乱数のベクトルを一様分布  $(0, 1)$  で返す ( $n \leq 128$ )。

このルーチンは [?larnv](#) から呼び出される補助ルーチンである。

## 入力パラメータ

<code>iseed</code>	INTEGER。 配列、次元は (4)。乱数生成器の種 (シード) を与える。配列成分は 0 から 4095 の間でなければならない。また <code>iseed(4)</code> は奇数でなければならない。
<code>n</code>	INTEGER。生成する乱数の個数。 $n \leq 128$

## 出力パラメータ

<code>x</code>	REAL ( <code>slaruv</code> の場合) DOUBLE PRECISION ( <code>dlaruv</code> の場合) 配列、次元は ( $n$ )。生成された乱数。
<code>seed</code>	種は更新される。

## ?larz

(`?tzrzf` が返したとおりの) 基本リフレクタを一般行列に適用する。

## 構文

```
call slarz( side, m, n, l, v, incv, tau, c, ldc, work )
call dlarz( side, m, n, l, v, incv, tau, c, ldc, work )
call clarz( side, m, n, l, v, incv, tau, c, ldc, work )
call zlarz( side, m, n, l, v, incv, tau, c, ldc, work )
```

## 説明

ルーチン `?larz` は、実数 / 複素基本リフレクタ  $H$  を、実数 / 複素  $m \times n$  行列  $C$  に、左または右のいずれかから適用する。 $H$  は次の形式で表現される。 $H = I - \tau * v * v'$ 、 $\tau$  は実数 / 複素スカラー、 $v$  は実数 / 複素ベクトルである。

$\tau = 0$  の場合、 $H$  は単位行列をとる。  
複素型で  $H'$  ( $H$  の共役転置) を適用するには  $\tau$  の代わりに  $\text{conjg}(\tau)$  を与える。  
 $H$  は [?tzrzf](#) から返されたとおり、 $k$  個の基本リフレクタの積である。

## 入力パラメータ

<i>side</i>	CHARACTER*1。 <i>side</i> = 'L' の場合、形式 $H * C$ <i>side</i> = 'R' の場合、形式 $C * H$ 。
<i>m</i>	INTEGER。行列 $C$ の行数。
<i>n</i>	INTEGER。行列 $C$ の列数。
<i>l</i>	INTEGER。Householder ベクトルとして意味を持った部分が格納されているベクトル $v$ の成分数。 <i>side</i> = 'L' の場合、 $m \geq l \geq 0$ 、 <i>side</i> = 'R' の場合、 $n \geq l \geq 0$
<i>v</i>	REAL (slarz の場合) DOUBLE PRECISION (dlarz の場合) COMPLEX (clarz の場合) COMPLEX*16 (zlarz の場合) 配列、次元は $(1+(n-1)*\text{abs}(\text{incv}))$ 。?tzrzf から返されたとおり、 $H$ の表現中にあるベクトル $v$ 。 $\tau = 0$ の場合、 $v$ は使用されない。
<i>incv</i>	INTEGER。 $v$ の成分間の増分。 $\text{incv} \neq 0$
<i>tau</i>	REAL (slarz の場合) DOUBLE PRECISION (dlarz の場合) COMPLEX (clarz の場合) COMPLEX*16 (zlarz の場合) $H$ の表現における値 $\tau$ 。
<i>v</i>	REAL (slarz の場合) DOUBLE PRECISION (dlarz の場合) COMPLEX (clarz の場合) COMPLEX*16 (zlarz の場合) 配列、次元は $(ldc, n)$ 。 $m \times n$ の行列 $C$ を格納する。
<i>ldc</i>	INTEGER。配列 $C$ のリーディング・ディメンジョン。 $ldc \geq \max(1, m)$



work            REAL (slarz の場合 )  
                  DOUBLE PRECISION (dlarz の場合 )  
                  COMPLEX (clarz の場合 )  
                  COMPLEX\*16 (zlarz の場合 )  
                  ワークスペース配列、次元は  
                   $side = 'L'$  の場合  $(n)$   
                   $side = 'R'$  の場合  $(m)$ 。

### 出力パラメータ

$c$              $side = 'L'$  の場合は行列  $H * C$  で、 $side = 'R'$  の場合は行列  $C * H$  で上書きされる。

## ?larzb

ブロック・リフレクタまたはその転置 / 共役転置  
 を一般矩形行列に適用する。

```
call slarzb( side, trans, direct, storev, m, n, k, l, v, ldv, t, ldt, c, ldc,
            work, ldwork )
call dlarzb( side, trans, direct, storev, m, n, k, l, v, ldv, t, ldt, c, ldc,
            work, ldwork )
call clarzb( side, trans, direct, storev, m, n, k, l, v, ldv, t, ldt, c, ldc,
            work, ldwork )
call zlarzb( side, trans, direct, storev, m, n, k, l, v, ldv, t, ldt, c, ldc,
            work, ldwork )
```

### 説明

このルーチンは実数 / 複素ブロック・リフレクタ  $H$  またはその転置  $H^T$  ( または複素型の場合  $H^H$  ) を、実数 / 複素分布  $m \times n$  行列  $C$  に左または右のいずれかから適用する。  
 現時点で、 $storev = 'R'$  と  $direct = 'B'$  のみがサポートされている。

### 入力パラメータ

$side$             CHARACTER\*1  
                   $side = 'L'$  の場合、 $H$  または  $H^T$  を左から適用する。  
                   $side = 'R'$  の場合、 $H$  または  $H^T$  を右から適用する。

<i>trans</i>	CHARACTER*1。 <i>trans</i> = 'N' の場合、 <i>H</i> を適用する (転置なし)。 <i>trans</i> = 'C' の場合、 <i>H'</i> を適用する (転置 / 共役転置)
<i>direct</i>	CHARACTER*1。基本リフレクタの積からどのように <i>H</i> が表現されているかを示す。 = 'F': $H = H(1) H(2) \dots H(k)$ (順方向、現時点でこの機能はサポートされていない) = 'B': $H = H(k) \dots H(2) H(1)$ (逆方向)
<i>storev</i>	CHARACTER*1。リフレクタを定義するベクトルがどのように格納されているかを示す。 = 'C' の場合、列方向 現時点でこの機能はサポートされていない)。 = 'R' の場合、行方向。
<i>m</i>	INTEGER。行列 <i>C</i> の行数。
<i>n</i>	INTEGER。行列 <i>C</i> の列数。
<i>k</i>	INTEGER。行列 <i>T</i> の次数 (積がブロック・リフレクタを定義する基本リフレクタの個数に等しい)。
<i>l</i>	INTEGER。Householder ベクトルとして意味を持った部分が格納されている行列 <i>V</i> の列数。 <i>side</i> = 'L' の場合、 $m \geq l \geq 0$ 、 <i>side</i> = 'R' の場合、 $n \geq l \geq 0$
<i>v</i>	REAL (slarzb の場合) DOUBLE PRECISION (dlarzb の場合) COMPLEX (clarzb の場合) COMPLEX*16 (zlarzb の場合) 配列、次元は (ldv,nv)。 <i>storev</i> = 'C' の場合は $nv = k$ 、 <i>storev</i> = 'R' の場合は $nv = l$
<i>ldv</i>	INTEGER。配列 <i>v</i> のリーディング・ディメンジョン。 <i>storev</i> = 'C' の場合 $ldv \geq l$ 、 <i>storev</i> = 'R' の場合 $ldv \geq k$
<i>t</i>	REAL (slarzb の場合) DOUBLE PRECISION (dlarzb の場合) COMPLEX (clarzb の場合) COMPLEX*16 (zlarzb の場合) 配列、次元は (ldt,k)。ブロック・リフレクタの表現中にある三角 $k \times k$ 行列 <i>T</i>
<i>ldt</i>	INTEGER。配列 <i>t</i> のリーディング・ディメンジョン。 $ldt \geq k$

<i>c</i>	REAL (slarzb の場合 ) DOUBLE PRECISION (dlarzb の場合 ) COMPLEX (clarzb の場合 ) COMPLEX*16 (zlarzb の場合 ) 配列、次元は ( <i>ldc</i> , <i>n</i> )。 $m \times n$ の行列 <i>C</i> を格納する。
<i>ldc</i>	INTEGER。 配列 <i>c</i> のリーディング・ディメンジョン。 $ldc \geq \max(1, m)$
<i>work</i>	REAL (slarzb の場合 ) DOUBLE PRECISION (dlarzb の場合 ) COMPLEX (clarzb の場合 ) COMPLEX*16 (zlarzb の場合 ) ワークスペース配列、次元は ( <i>ldwork</i> , <i>k</i> )。
<i>ldwork</i>	INTEGER。 配列 <i>work</i> のリーディング・ディメンジョン。 <i>side</i> = 'L' の場合、 $ldwork \geq \max(1, n)$ 、 <i>side</i> = 'R' の場合、 $ldwork \geq \max(1, m)$

### 出力パラメータ

<i>c</i>	<i>c</i> は、 $H^*C$ 、または $H^*C$ 、または $C^*H$ 、または $C^*H$ で上書きされる。
----------	---

## ?larzt

ブロック・リフレクタ  $H = I - VTV^H$  の  
三角係数 *T* を生成する。

### 構文

```
call slarzt( direct, storev, n, k, v, ldv, tau, t, ldt )
call dlarzt( direct, storev, n, k, v, ldv, tau, t, ldt )
call clarzt( direct, storev, n, k, v, ldv, tau, t, ldt )
call zlarzt( direct, storev, n, k, v, ldv, tau, t, ldt )
```

## 説明

このルーチンは、 $k$  個の基本リフレクタの積として定義されている次数  $> n$  の実 / 複素ブロック・リフレクタ  $H$  の三角係数  $T$  を生成する。

$direct = 'F'$  の場合、 $H = H(1) H(2) \dots H(k)$  と  $T$  は上三角である。

$direct = 'B'$  の場合、 $H = H(k) \dots H(2) H(1)$  と  $T$  は下三角である。

$storev = 'C'$  の場合、基本リフレクタ  $H(i)$  を定義するベクトルは配列  $v$  の  $i$  番目の列に格納され、 $H = I - V * T * V'$  である。

$storev = 'R'$  の場合、基本リフレクタ  $H(i)$  を定義するベクトルは配列  $v$  の  $i$  番目の行に格納され、 $H = I - V' * T * V$  である。

現時点で、 $storev = 'R'$  と  $direct = 'B'$  のみがサポートされている。

## 入力パラメータ

<i>direct</i>	CHARACTER*1。ブロック・リフレクタを生成するために基本リフレクタを乗算する次数を指定する。 $direct = 'F'$ の場合、 $H = H(1) H(2) \dots H(k)$ (順方向、現時点でこの機能はサポートされていない) $direct = 'B'$ の場合、 $H = H(k) \dots H(2) H(1)$ (逆方向)
<i>storev</i>	CHARACTER*1。基本リフレクタを定義するベクトルがどのように格納されているかを指定する。(次のアプリケーション・ノートも参照): $storev = 'C'$ の場合、列方向 (現時点でこの機能はサポートされていない)。 $storev = 'R'$ の場合、行方向。
<i>n</i>	INTEGER。ブロック・リフレクタ $H$ の次数。 $n \geq 0$
<i>k</i>	INTEGER。三角係数 $T$ の次数 (基本リフレクタの個数に等しい)。 $k \geq 1$
<i>v</i>	REAL (slarzt の場合) DOUBLE PRECISION (dlarzt の場合) COMPLEX (clarzt の場合) COMPLEX*16 (zlarzt の場合) 配列、次元は $storev = 'C'$ の場合では $(ldv, k)$ $storev = 'R'$ の場合では $(ldv, n)$ 行列 $V$ 。
<i>ldv</i>	INTEGER。配列 $v$ のリーディング・ディメンジョン。 $storev = 'C'$ の場合、 $ldv \geq \max(1, n)$ 、 $storev = 'R'$ の場合、 $ldv \geq k$

$\tau$	REAL (slarzt の場合 ) DOUBLE PRECISION (dlarzt の場合 ) COMPLEX (clarzt の場合 ) COMPLEX*16 (zlarzt の場合 ) 配列、次元は $(k)$ 。 $\tau(i)$ には、基本リフレクタ $H(i)$ のスカラー係数が入っていないなければならない。
$ldt$	INTEGER。出力配列 $t$ のリーディング・ディメンジョン。 $ldt \geq k$

### 出力パラメータ

$t$	REAL (slarzt の場合 ) DOUBLE PRECISION (dlarzt の場合 ) COMPLEX (clarzt の場合 ) COMPLEX*16 (zlarzt の場合 ) 配列、次元は $(ldt, k)$ 。ブロック・リフレクタの $k \times k$ 三角係数 $T$ 。 $direct = 'F'$ の場合、 $T$ は上三角。 $direct = 'B'$ の場合、 $T$ は下三角。配列の残りの部分は使用されない。
$v$	行列 $V$ 。次のアプリケーション・ノートを参照。

## アプリケーション・ノート

行列  $V$  の形状と  $H(i)$  を定義するベクトルの格納形式の例を、 $n=5$ 、 $k=3$  において次の図に示す。1 に等しい成分は格納されない。対応する配列成分は変更されるが、終了時に復元される。配列の残りの部分は使用されない。

$direct = 'F'$  および  $storev = 'C'$ :       $direct = 'F'$  および  $storev = 'R'$ :

$$V = \begin{bmatrix} v_1 & v_2 & v_3 \\ v_1 & v_2 & v_3 \\ v_1 & v_2 & v_3 \\ v_1 & v_2 & v_3 \\ v_1 & v_2 & v_3 \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ 1 & \cdot & \cdot \\ \cdot & 1 & \cdot \\ \cdot & \cdot & 1 \end{bmatrix} \quad \begin{array}{c} \text{---}V\text{---} \\ / \qquad \backslash \end{array} \begin{bmatrix} v_1 & v_1 & v_1 & v_1 & v_1 & \cdot & \cdot & \cdot & 1 \\ v_2 & v_2 & v_2 & v_2 & v_2 & \cdot & \cdot & \cdot & 1 \\ v_3 & v_3 & v_3 & v_3 & v_3 & \cdot & \cdot & \cdot & 1 \end{bmatrix}$$

$direct = 'B'$  および  $storev = 'C'$ :       $direct = 'B'$  および  $storev = 'R'$ :

$$V = \begin{bmatrix} 1 \\ \cdot & 1 \\ \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ v_1 & v_2 & v_3 \\ v_1 & v_2 & v_3 \\ v_1 & v_2 & v_3 \\ v_1 & v_2 & v_3 \\ v_1 & v_2 & v_3 \end{bmatrix} \quad \begin{array}{c} \text{---}V\text{---} \\ / \qquad \backslash \end{array} \begin{bmatrix} 1 & \cdot & \cdot & \cdot & \cdot & v_1 & v_1 & v_1 & v_1 & v_1 \\ \cdot & 1 & \cdot & \cdot & \cdot & v_2 & v_2 & v_2 & v_2 & v_2 \\ \cdot & \cdot & 1 & \cdot & \cdot & v_3 & v_3 & v_3 & v_3 & v_3 \end{bmatrix}$$

## ?las2

$2 \times 2$  三角行列の特異値を計算する。

### 構文

```
call slas2( f, g, h, ssmin, ssmax )
call dlas2( f, g, h, ssmin, ssmax )
```

### 説明

ルーチン ?las2 は  $2 \times 2$  の行列、の特異値を計算する。

$$\begin{bmatrix} f & g \\ 0 & h \end{bmatrix}$$

出力では、ssmin に小さい方の特異値、ssmax に大きい方の特異値が格納される。

### 入力パラメータ

f, g, h      REAL (slas2 の場合 )  
             DOUBLE PRECISION (dlas2 の場合 )  
それぞれ、 $2 \times 2$  行列の (1,1)、(1,2)、(2,2) の成分。

### 出力パラメータ

ssmin, ssmax    REAL (slas2 の場合 )  
                 DOUBLE PRECISION (dlas2 の場合 )  
それぞれ、小さい方の特異値と大きい方の特異値。

### アプリケーション・ノート

オーバーフロー / アンダーフローの場合を除いて、加算 / 減算にガード桁がなくとも、すべての出力の大きさは最後の位置からわずかな単位以内 (*ulp*) で正確である。

IEEE 演算で、1 つの行列成分が無限でもコードは正しく動作する。

最大特異値自身がオーバーフローしない限り、あるいは最大特異値がオーバーフローからわずかな *ulp* 範囲内にある限り、オーバーフローは起こらない。(Cray のように部分オーバーフローを持つマシンでは、最大特異値がオーバーフローに対して 2 の因子以内にあるとオーバーフローが起こることがある。)

アンダーフローは緩やかであればアンダーフローは無害である。そうでない場合、結果は、アンダーフローしきい値に近いサイズの摂動によって変更された行列に一致することがある。

---

### ?lascl

一般矩形行列に  $c_{to}/c_{from}$  として定義される  
実数スカラを乗算する。

---

#### 構文

```
call slascl( type, kl, ku, cfrom, cto, m, n, a, lda, info )
call dlascl( type, kl, ku, cfrom, cto, m, n, a, lda, info )
call clascl( type, kl, ku, cfrom, cto, m, n, a, lda, info )
call zlascl( type, kl, ku, cfrom, cto, m, n, a, lda, info )
```

#### 説明

ルーチン ?lascl は  $m \times n$  実数 / 複素行列  $A$  に実数スカラ  $cto/cfrom$  を乗算する。最終結果  $cto * A(i,j) / cfrom$  がオーバーフロー / アンダーフローしない限り、演算はオーバーフロー / アンダーフローなしで実行される。

$A$  の形式は `type` によって、フル、上三角、下三角、上 Hessenberg、または帯から指定される。

#### 入力パラメータ

<code>type</code>	CHARACTER*1。 <code>type</code> は入力行列の格納形式を示す。 = 'G' の場合、 $A$ はフル行列。 = 'L' の場合、 $A$ は下三角行列。 = 'U' の場合、 $A$ は上三角行列。 = 'H' の場合、 $A$ は上 Hessenberg 行列。 = 'B' の場合、 $A$ は下バンド幅 $kl$ と上バンド幅 $ku$ を持つ対称帯行列で、下半分のみ格納される。= 'Q' の場合、 $A$ は下バンド幅 $kl$ と上バンド幅 $ku$ を持つ対称帯行列で、上半分のみ格納される。 = 'Z' の場合、 $A$ は下バンド幅 $kl$ と上バンド幅 $ku$ を持つ対称帯行列。
<code>kl</code>	INTEGER。 $A$ の下バンド幅。 <code>type</code> = 'B'、'Q'、または 'Z' の場合のみ参照される。



<i>ku</i>	INTEGER。 <i>A</i> の上バンド幅。 <i>type</i> = 'B'、'Q'、または 'Z' の場合のみ参照される。
<i>cfrom</i> , <i>cto</i>	REAL (slascl/clascl の場合 ) DOUBLE PRECISION (dlascl/zlascl の場合 )  行列 <i>A</i> に乗算する <i>cto/cfrom</i> 。最終結果 <i>cto</i> * <i>A</i> ( <i>i</i> , <i>j</i> )/ <i>cfrom</i> がオーバーフロー/アンダーフローなしで表現される場合、 <i>A</i> ( <i>i</i> , <i>j</i> ) はオーバーフロー/アンダーフローなしで計算される。 <i>cfrom</i> は非ゼロでなければならない。
<i>m</i>	INTEGER。 行列 <i>A</i> の行数。 $m \geq 0$
<i>n</i>	INTEGER。 行列 <i>A</i> の列数。 $n \geq 0$
<i>a</i>	REAL (slascl の場合 ) DOUBLE PRECISION (dlascl の場合 ) COMPLEX (clascl の場合 ) COMPLEX*16 (zlascl の場合 ) 配列、次元は ( <i>lda</i> , <i>m</i> )。 <i>cto/cfrom</i> が乗算される行列。格納形式については <i>type</i> を参照。
<i>lda</i>	INTEGER。 配列 <i>a</i> のリーディング・ディメンジョン。 $lda \geq \max(1, m)$

## 出力パラメータ

<i>a</i>	乗算された行列 <i>A</i>
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> < 0 の場合、 <i>i</i> 番目の引数の値が不正だったことを示す。

### ?lasd0

対角  $d$  と対角外  $e$  を持つ実数上二重対角  $n \times m$  行列  $B$  の特異値を計算する。  
`?bdsdc` で使用される。

---

#### 構文

```
call slasd0( n, sqre, d, e, u, ldu, vt, ldvt, smlsiz, iwork, work, info )
call dlasd0( n, sqre, d, e, u, ldu, vt, ldvt, smlsiz, iwork, work, info )
```

#### 説明

ルーチン `?lasd0` は、分割統治法を使って、対角  $d$  と対角外  $e$  を持つ実数上二重対角  $n \times m$  行列  $B$  の特異値分解 (SVD) を計算する。ここで  $m = n + sqre$

このアルゴリズムは、 $B = U * S * VT$  であるような直交行列  $U$  と  $VT$  を計算する。特異値  $S$  は  $d$  に上書きされる。

関連するサブルーチン [?lasda](#) は、特異値のみを計算し、オプションで特異ベクトルをコンパクト形式で計算する。

#### 入力パラメータ

<code>n</code>	INTEGER。上二重対角行列の行次元を格納する。主対角配列 $d$ の次元でもある。
<code>sqre</code>	INTEGER。二重対角行列の列のサイズを指定する。 <code>sqre = 0</code> の場合、二重対角行列は列次元 $m = n$ を持つ。 <code>sqre = 1</code> の場合、二重対角行列は列次元 $m = n+1$ を持つ。
<code>d</code>	REAL( <code>slasd0</code> の場合) DOUBLE PRECISION( <code>dlasd0</code> の場合) 配列、次元は $(n)$ 。 $d$ には二重対角行列の主対角を格納する。
<code>e</code>	REAL( <code>slasd0</code> の場合) DOUBLE PRECISION( <code>dlasd0</code> の場合) 配列、次元は $(m-1)$ 。二重対角行列の劣対角成分を格納する。出力において $e$ の内容は壊される。
<code>ldu</code>	INTEGER。出力配列 $u$ のリーディング・ディメンジョンを格納する。
<code>ldvt</code>	INTEGER。出力配列 $vt$ のリーディング・ディメンジョンを格納する。

<i>smlsiz</i>	INTEGER。計算ツリーの一番下の部分問題の最大サイズを格納する。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $(8n)$ 以上でなければならない。
<i>work</i>	REAL (slasd0 の場合 ) DOUBLE PRECISION (dlasd0 の場合 ) ワークスペース配列、次元は $(3m_2 + 2m)$ 以上でなければならない。

### 出力パラメータ

<i>d</i>	<i>info</i> = 0 の場合、 <i>d</i> には二重対角行列の特異値が上書きされる。
<i>u</i>	REAL (slasd0 の場合 ) DOUBLE PRECISION (dlasd0 の場合 ) 配列、次元は $(ldq, n)$ 以上。 <i>u</i> には左特異値が格納される。
<i>vt</i>	REAL (slasd0 の場合 ) DOUBLE PRECISION (dlasd0 の場合 ) 配列、次元は $(ldvt, m)$ 以上。 <i>vt'</i> には右特異値が格納される。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = $-i < 0$ の場合、 <i>i</i> 番目の引数の値が不正だったことを示す。 <i>info</i> = 1 の場合、特異値が収束しなかったことを示す。

## ?lasd1

指定サイズの上二重対角行列 *B* の SVD を計算する。*?bdsdc* で使用される。

### 構文

```
call slasd1( nl, nr, sqre, d, alpha, beta, u, ldu, vt, ldvt, idxq, iwork, work,
            info )
call dlasd1( nl, nr, sqre, d, alpha, beta, u, ldu, vt, ldvt, idxq, iwork, work,
            info )
```

### 説明

このルーチンは上二重対角  $n \times m$  行列 *B* の SVD を計算する。ここで  $n = nl + nr + 1$  および  $m = n + sqre$ 。ルーチン *?lasd1* は [?lasd0](#) から呼び出される。

関連サブルーチン [?lasd7](#) は特異値 ( および因子分解形式の特異ベクトル ) を目的とする場合を扱う。

?lasd1 は次のように SVD を計算する。

$$B = U(in) * \begin{bmatrix} D1(in) & 0 & 0 & 0 \\ Z1' & a & Z2' & b \\ 0 & 0 & D2(in) & 0 \end{bmatrix} * VT(in)$$

$$= U(out) * (D(out) \quad 0) * VT(out)$$

$Z = (Z1' a Z2' b) = u' VT'$ 、 $u$  は次元  $m$  のベクトルで、 $n1+1$  番目と  $n1+2$  番目の成分に  $alpha$  と  $beta$  を持ち他はゼロである。 $sgre=0$  の場合、成分  $b$  は空である。

元の行列の左特異ベクトルは  $u$  に格納され、右特異ベクトルの転置は  $vt$  に格納され、特異値は  $a$  に格納される。アルゴリズムは次の 3 過程で構成される。

第 1 の過程では、複数の特異値がある場合、または  $Z$  ベクトルにゼロがある場合、問題の大きさの収縮を行う。そのような場合ごとに、永年方程式問題の次元は 1 だけ縮小される。この過程はルーチン [?lasd2](#) によって実行される。

第 2 の過程では更新された特異値を計算する。これは、ルーチン [?lasd4](#) ([?lasd3](#) として呼び出される) を介して永年方程式の根の平方根を見つけることによって行われる。このルーチンは現在の問題の特異ベクトルも計算する。

最後の過程では、更新された特異値を直接使用して更新された特異ベクトルを計算する。現在の問題に対する特異ベクトルは、全体問題から得られる特異ベクトルで乗算される。

## 入力パラメータ

$n1$	INTEGER。上ブロックの行次元。 $n1 \geq 1$
$nr$	INTEGER。下ブロックの行次元。 $nr \geq 1$
$sgre$	INTEGER。 $sgre = 0$ の場合、下ブロックは $nr \times nr$ の正方行列。 $sgre = 1$ の場合、下ブロックは $nr \times (nr+1)$ の矩形行列。二重対角行列は行次元 $n = n1 + nr + 1$ 、列次元 $m = n + sgre$ を持つ。

<i>d</i>	REAL (slasd1 の場合 ) DOUBLE PRECISION (dlasd1 の場合 ) 配列、次元は $(n = n1 + nr + 1)$ 。 $d(1:n1, 1:n1)$ には上ブロックの特異値を格納し、 $d(n1+2:n)$ には下ブロックの特異値を格納する。
<i>alpha</i>	REAL (slasd1 の場合 ) DOUBLE PRECISION (dlasd1 の場合 ) 追加した行に関連する対角成分。
<i>beta</i>	REAL (slasd1 の場合 ) DOUBLE PRECISION (dlasd1 の場合 ) 追加した行に関連する非対角成分。
<i>u</i>	REAL (slasd1 の場合 ) DOUBLE PRECISION (dlasd1 の場合 ) 配列、次元は $(ldu, n)$ 。 $u(1:n1, 1:n1)$ には上ブロックの左特異ベクトルを格納し、 $u(n1+2:n, n1+2:n)$ には下ブロックの左特異ベクトルを格納する。
<i>ldu</i>	INTEGER。 配列 <i>u</i> のリーディング・ディメンジョン。 $ldu \geq \max(1, n)$
<i>vt</i>	REAL (slasd1 の場合 ) DOUBLE PRECISION (dlasd1 の場合 ) 配列、次元は $(ldvt, m)$ 、ここで $m = n + sqre$ 。 $vt(1:n1+1, 1:n1+1)$ には上ブロックの右特異ベクトルを格納し、 $vt(n1+2:m, n1+2:m)$ には下ブロックの右特異ベクトルを格納する。
<i>ldvt</i>	INTEGER。 配列 <i>vt</i> のリーディング・ディメンジョン。 $ldvt \geq \max(1, m)$
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $(4n)$
<i>work</i>	REAL (slasd1 の場合 ) DOUBLE PRECISION (dlasd1 の場合 ) ワークスペース配列、次元は $(3m2 + 2m)$ 。

### 出力パラメータ

<i>d</i>	$d(1:n)$ には、変更された行列の特異値が格納される。
<i>u</i>	<i>u</i> には二重対角行列の左特異ベクトルが格納される。
<i>vt</i>	<i>vt'</i> には二重対角行列の右特異ベクトルが格納される。

<i>idxq</i>	INTEGER 配列、次元は ( <i>n</i> )。解かれたばかりの部分問題をソートされた順に再統合する置換で上書きされる。すなわち、 <i>d(idxq(i = 1, n))</i> は昇順となる。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> < 0 の場合、 <i>i</i> 番目の引数の値が不正であったことを示す。 <i>info</i> = 1 の場合、特異値が収束しなかったことを示す。

---

### ?lasd2

2 つの特異値のセットをソートされた単一のセットに併合する。`?bdsdc` で使用される。

---

#### 構文

```
call slasd2( nl, nr, sqre, k, d, z, alpha, beta, u, ldu, vt, ldvt, dsigma, u2,  
            ldu2, vt2, ldvt2, idxp, idx, idxc, idxq, coltyp, info )  
call dlasd2( nl, nr, sqre, k, d, z, alpha, beta, u, ldu, vt, ldvt, dsigma, u2,  
            ldu2, vt2, ldvt2, idxp, idx, idxc, idxq, coltyp, info )
```

#### 説明

ルーチン `?lasd2` は 2 つの特異値のセットをソートされた単一のセットに併合する。続いて、問題の大きさの収縮を試みる。収縮が起こるには 2 つの場合がある。2 個以上の特異値が互いに近い場合、または、*Z* ベクトルに微小成分がある場合である。そのような場合ごとに、関連する永年方程式問題の次元は 1 だけ縮小される。

ルーチン `?lasd2` は [?lasd1](#) から呼び出される。

#### 入力パラメータ

<i>nl</i>	INTEGER。上ブロックの行次元。 <i>nl</i> ≥ 1
<i>nr</i>	INTEGER。下ブロックの行次元。 <i>nr</i> ≥ 1

<i>sqre</i>	INTEGER。 <i>sqre</i> = 0 の場合、下ブロックは $nr \times nr$ 平方行列である。 <i>sqre</i> = 1 の場合、下ブロックは $nr \times (nr+1)$ 矩形行列である。 二重対角行列は行次元 $n = n1 + nr + 1$ と列次元 $m = n + sqre \geq n$ を持つ。
<i>d</i>	REAL (slasd2 の場合) DOUBLE PRECISION (dlasd2 の場合) 配列、次元は ( <i>n</i> )。 <i>d</i> には結合を行う 2 つの部分行列の特異値を格納する。
<i>alpha</i>	REAL (slasd2 の場合) DOUBLE PRECISION (dlasd2 の場合) 追加した行に関連する対角成分。
<i>beta</i>	REAL (slasd2 の場合) DOUBLE PRECISION (dlasd2 の場合) 追加した行に関連する非対角成分。
<i>u</i>	REAL (slasd2 の場合) DOUBLE PRECISION (dlasd2 の場合) 配列、次元は ( <i>ldu</i> , <i>n</i> )。 <i>u</i> には、(1, 1), ( <i>n1</i> , <i>n1</i> ) と ( <i>n1</i> +2, <i>n1</i> +2), ( <i>n</i> , <i>n</i> ) の偶にある 2 つの平方ブロック内の 2 個の部分行列の左特異ベクトルを格納する。
<i>ldu</i>	INTEGER。 配列 <i>u</i> のリーディング・ディメンジョン。 $ldu \geq n$
<i>ldu2</i>	INTEGER。 出力配列 <i>u2</i> のリーディング・ディメンジョン。 $ldu2 \geq n$
<i>vt</i>	REAL (slasd2 の場合) DOUBLE PRECISION (dlasd2 の場合) 配列、次元は ( <i>ldvt</i> , <i>m</i> )。 <i>vt'</i> には、 (1, 1), ( <i>n1</i> +1, <i>n1</i> +1) と ( <i>n1</i> +2, <i>n1</i> +2), ( <i>m</i> , <i>m</i> ) の偶にある 2 つの平方ブロック内の 2 個の部分行列の右特異ベクトルを格納する。
<i>ldvt</i>	INTEGER。 配列 <i>vt</i> のリーディング・ディメンジョン。 $ldvt \geq m$
<i>ldvt2</i>	INTEGER。 出力配列 <i>vt2</i> のリーディング・ディメンジョン。 $ldvt2 \geq m$

<i>idxp</i>	INTEGER。 ワークスペース配列、次元は (n)。この配列には、収縮された <i>d</i> の値を配列の終わりに配置した置換を格納する。出力で、 <i>idxp</i> (2:k) は収縮されていない <i>d</i> 値を指し、 <i>idxp</i> (k+1:n) は収縮された特異値を指す。
<i>idx</i>	INTEGER。 ワークスペース配列、次元は (n)。 <i>d</i> の内容を昇順でソートしたときに使用した置換を格納する。
<i>coltyp</i>	INTEGER。 ワークスペース配列、次元は (n)。ワークスペースとして、 <i>u</i> 2 行列内の列、または <i>vt</i> 2 行列内の行を示すラベルを次のタイプから格納する。 1: 上半分のみ非ゼロ 2: 下半分のみ非ゼロ 3: 密 4: 収縮
<i>idxq</i>	INTEGER。 配列、次元は (n)。 <i>d</i> に入っている 2 つの部分問題を昇順で別々にソートする置換。この置換の前半にある成分は、最初に 1 つの位置だけ後ろに動かされなければならない。また、後半にある成分は、最初にそれら値に <i>n</i> l+1 を加算しなければならない。

## 出力パラメータ

<i>k</i>	INTEGER。収縮されていない行列の次元が格納される。 $1 \leq k \leq n$
<i>d</i>	<i>d</i> は、後続の (n-k) 個の更新された特異値 (収縮された) の昇順で上書きされる。
<i>u</i>	<i>u</i> の最後の n-k 個の列は、後続の (n-k) 個の更新された左特異値 (収縮された) で上書きされる。
<i>z</i>	REAL (slasd2 の場合) DOUBLE PRECISION (dlasd2 の場合) 配列、次元は (n)。 <i>z</i> は永年方程式内の更新列ベクトルで上書きされる。
<i>dsigma</i>	REAL (slasd2 の場合) DOUBLE PRECISION (dlasd2 の場合) 配列、次元は (n)。永年方程式内の対角成分 (k-1 個の特異値と 1 個のゼロ) のコピーが格納される。



<i>u2</i>	REAL (slasd2 の場合 ) DOUBLE PRECISION (dlasd2 の場合 ) 配列、次元は ( <i>ldu2</i> , <i>n</i> )。最初の <i>k</i> -1 個の左特異ベクトルのコピーが格納される。これは新しい左特異ベクトルを解くために ?lasd3 での行列乗算 (?gemm) で使用される。 <i>u2</i> は 4 つのブロックに配置される。第 1 のブロックは列 <i>n1</i> +1 が 1 で他はゼロである。第 2 のブロックには <i>n1</i> とその上のみに非ゼロの成分が格納される。第 3 のブロックには <i>n1</i> +1 よりも下のみに非ゼロの成分が格納される。第 4 のブロックは密。
<i>vt</i>	<i>vt'</i> の最後の <i>n-k</i> 個の列は、後続の ( <i>n-k</i> ) 個の更新された右特異値 (収縮された) で上書きされる。 <i>sqre</i> = 1 の場合、 <i>vt</i> の最後の行は右ヌル空間にかかる。
<i>vt2</i>	REAL (slasd2 の場合 ) DOUBLE PRECISION (dlasd2 の場合 ) 配列、次元は ( <i>ldvt2</i> , <i>n</i> )。 <i>vt2'</i> には最初の <i>k</i> 個の右特異ベクトルのコピーが格納される。これは新しい右特異ベクトルを解くために ?lasd3 での行列乗算 (?gemm) で使用される。 <i>vt2</i> は 3 つのブロックに配置される。第 1 のブロックには <i>sigma</i> 内の特殊 0 対角成分に対応する行が格納される。第 2 のブロックには <i>n1</i> +1 とその前のみに非ゼロが格納される。第 3 のブロックには <i>n1</i> +2 とその後ろのみに非ゼロが格納される。
<i>idxc</i>	INTEGER。 配列、次元は ( <i>n</i> )。収縮された <i>U</i> 行列の列を 3 つのグループに配置する置換が格納される。第 1 のグループには <i>n1</i> とその上に非ゼロの成分が格納され、第 2 のグループには <i>n1</i> +2 よりも下のみに非ゼロの成分が格納され、第 3 は密である。
<i>coltyp</i>	<i>coltyp</i> ( <i>i</i> ) が <i>i</i> 番目のタイプ列の次元となる次元 4 の配列で上書きされる。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> < 0 の場合、 <i>i</i> 番目の引数の値が不正だったことを示す。

## ?lasd3

$D$  と  $Z$  内の値によって定義されている永年方程式に対して根のすべての平方根を求め、次に行列乗算によって特異ベクトルを更新する。  
`?bdsdc` で使用される。

---

### 構文

```
call slasd3( nl, nr, sqre, k, d, q, ldq, dsigma, u, ldu, u2, ldu2, vt, ldvt,
            vt2, ldvt2, idxc, ctot, z, info )
call dlasd3( nl, nr, sqre, k, d, q, ldq, dsigma, u, ldu, u2, ldu2, vt, ldvt,
            vt2, ldvt2, idxc, ctot, z, info )
```

### 説明

ルーチン `?lasd3` は、 $D$  と  $Z$  内の値によって定義されている永年方程式に対して根のすべての平方根を求める。[?lasd4](#) に対する適切な呼び出しを実行し、続いて行列乗算によって特異ベクトルを更新する。

ルーチン `?lasd3` は [?lasdl](#) から呼び出される。

### 入力パラメータ

<code>nl</code>	INTEGER。上ブロックの行次元。 $nl \geq 1$
<code>nr</code>	INTEGER。下ブロックの行次元。 $nr \geq 1$
<code>sqre</code>	INTEGER。 $sqre = 0$ の場合、下ブロックは $nr \times nr$ の正方行列。 $sqre = 1$ の場合、下ブロックは $nr \times (nr+1)$ 矩形行列である。 二重対角行列は行次元 $n = nl + nr + 1$ と列次元 $m = n + sqre \geq n$ を持つ。
<code>k</code>	INTEGER。永年方程式の大きさで、 $1 \leq k \leq n$
<code>q</code>	REAL( <code>slasd3</code> の場合 ) DOUBLE PRECISION( <code>dlasd3</code> の場合 ) ワークスペース配列、次元は $(ldq, k)$ 以上。

<i>ldq</i>	INTEGER。配列 <i>q</i> のリーディング・ディメンジョン。 $ldq \geq k$
<i>dsigma</i>	REAL (slasd3 の場合) DOUBLE PRECISION (dlasd3 の場合) 配列、次元は ( <i>k</i> )。この配列の先頭の <i>k</i> 個の成分には収縮された更新問題の古い根を格納する。これらは永年方程式の極である。
<i>u</i>	REAL (slasd3 の場合) DOUBLE PRECISION (dlasd3 の場合) 配列、次元は ( <i>ldu</i> , <i>n</i> )。この行列の最後の <i>n-k</i> 個の列には収縮された左特異ベクトルを格納する。
<i>ldu</i>	INTEGER。配列 <i>u</i> のリーディング・ディメンジョン。 $ldu \geq n$
<i>u2</i>	REAL (slasd3 の場合) DOUBLE PRECISION (dlasd3 の場合) 配列、次元は ( <i>ldu2</i> , <i>n</i> )。この行列の先頭の <i>k</i> 個の列には分割問題に対する収縮されていない左特異ベクトルを格納する。
<i>ldu2</i>	INTEGER。配列 <i>u2</i> のリーディング・ディメンジョン。 $ldu2 \geq n$
<i>vt</i>	REAL (slasd3 の場合) DOUBLE PRECISION (dlasd3 の場合) 配列、次元は ( <i>ldvt</i> , <i>m</i> )。vt' の最後の <i>m-k</i> 個の列には収縮された右特異ベクトルを格納する。
<i>ldvt</i>	INTEGER。配列 <i>vt</i> のリーディング・ディメンジョン。 $ldvt \geq n$
<i>vt2</i>	REAL (slasd3 の場合) DOUBLE PRECISION (dlasd3 の場合) 配列、次元は ( <i>ldvt2</i> , <i>n</i> )。vt2' の先頭の <i>k</i> 個の列には分割問題に対する収縮されていない右特異ベクトルを格納する。
<i>ldvt2</i>	INTEGER。配列 <i>vt2</i> のリーディング・ディメンジョン。 $ldvt2 \geq n$
<i>idxc</i>	INTEGER。 配列、次元は ( <i>n</i> )。u の列 (および vt の行) を 3 つのグループに配置する置換。第 1 のグループには <i>n1+1</i> とその上 (または前) のみに非ゼロの成分が格納される。第 2 のグループには <i>n1+2</i> とその下 (または後ろ) にのみ非ゼロの成分が格納される。第 3 のグループは密である。

ただし、*u* の最初の列と *vt* の最初の行は別々に扱われる。*?lasd4* で求められた特異ベクトルの行は、行列乗算が実行される前に、同様に置換されなければならない。

*ctot*                    INTEGER。  
配列、次元は (4)。*idxc* に記述されているとおり、*u* に含まれる列 (または *vt* に含まれる行) のタイプのそれぞれの合計。4 番目の列タイプは収縮された任意の列である。

*z*                      REAL (*slasd3* の場合 )  
DOUBLE PRECISION (*dlasd3* の場合 )  
配列、次元は (*k*)。この配列の先頭の *k* 個の成分には収縮調整された更新行ベクトルの成分を格納する。

### 出力パラメータ

*d*                      REAL (*slasd3* の場合 )  
DOUBLE PRECISION (*dlasd3* の場合 )  
配列、次元は (*k*)。永年方程式の根の平方根が昇順で上書きされる。

*info*                    INTEGER。  
*info* = 0 の場合、正常に終了したことを示す。  
*info* = -*i* < 0 の場合、*i* 番目の引数の値が不正だったことを示す。  
*info* = 1 の場合、特異値が収束しなかったことを示す。

### アプリケーション・ノート

このコードは浮動小数点演算に対してきわめて緩い仮定を行う。加減算においてガード桁を持つマシン、または Cray X-MP、Cray Y-MP、Cray C-90、または Cray-2 のような減算のガード桁を持たないバイナリマシンで動作する。ガード桁を持たない 16 進マシンや 10 進マシンでは異常終了するのも考えられるが、そのような前例はない。

## ?lasd4

正値対角行列に対する正値対称階数 1 更新の、  
 $i$  番目の更新された固有値の平方根を計算する。  
 ?bdsdc で使用される。

### 構文

```
call slasd4( n, i, d, z, delta, rho, sigma, work, info )
call dlasd4( n, i, d, z, delta, rho, sigma, work, info )
```

### 説明

このルーチンは、正値対角行列に対する正値対称階数 1 の更新の、 $i$  番目の更新された固有値の平方根を計算する。正値対角行列の成分は、配列  $d$  中の対応する成分の平方として与えられ、 $i < j$  では  $0 \leq d(i) < d(j)$ 、および  $\rho > 0$  である。これは呼び出しルーチンによって配置され、一般性が失われることはない。階数 1 の更新された連立方程式はゆえに、

$$\text{diag}(d) * \text{diag}(d) + \rho * Z * Z_{\text{transpose}}$$

$Z$  のユークリッド・ノルムを 1 とする。この方法には、単純補間の有利関数を用いた永久方程式内の有利関数の近似が含まれる。

### 入力パラメータ

$n$	INTEGER。全配列の長さ。
$i$	INTEGER。計算する固有値のインデックス。 $1 \leq i \leq n$
$d$	REAL (slasd4 の場合 ) DOUBLE PRECISION (dlasd4 の場合 ) 配列、次元は $(n)$ 。 元の固有値。 $i < j$ では $0 \leq d(i) < d(j)$ のように順序どおりに並べられていると仮定される。
$z$	REAL (slasd4 の場合 ) DOUBLE PRECISION (DLASD4 の場合 ) 配列、次元は $(n)$ 。 更新ベクトルの成分。
$\rho$	REAL (slasd4 の場合 ) DOUBLE PRECISION (dlasd4 の場合 ) 対称更新式におけるスカラ。

*work*                    *real* (slasd4 の場合 )  
DOUBLE PRECISION (dlasd4 の場合 )  
ワークスペース配列、次元は (*n*)。  
*n* ≠ 1 の場合、*work* の *j* 番目の成分に (*d*(*j*) + *sigma\_i*) を格納する。  
*n* = 1 の場合、*work*(1) = 1

### 出力パラメータ

*delta*                    REAL (slasd4 の場合 )  
DOUBLE PRECISION (DLASD4 の場合 )  
配列、次元は (*n*)。  
*n* ≠ 1 の場合、*delta* の *j* 番目の成分に (*d*(*j*) - *sigma\_i*) が格納される。  
*n* = 1 の場合、*delta*(1) = 1。ベクトル *delta* には (特異) 固有ベクトルを構成するために必要な情報が格納される。

*sigma*                    REAL (slasd4 の場合 )  
DOUBLE PRECISION (dlasd4 の場合 )  
計算された  $\lambda_i$ 、*i* 番目の更新された固有値。

*info*                    INTEGER。  
= 0 の場合、正常に終了したことを示す。  
> 0 の場合で *info* = 1 ならば、更新処理に失敗したことを示す。

---

## ?lasd5

2 × 2 対角行列の正値対称階数 1 更新から、  
*i* 番目の固有値の平方根を計算する。  
?bdsdc で使用される。

---

### 構文

```
call slasd5( i, d, z, delta, rho, dsigma, work )  
call dlasd5( i, d, z, delta, rho, dsigma, work )
```

### 説明

このルーチンは、2 × 2 対角行列の正値対称階数 1 更新から、*i* 番目の固有値の平方根を計算する。

$\text{diag}(d) * \text{diag}(d) + \text{rho} * Z * Z_{\text{transpose}}$

配列  $d$  の対角成分は  $i < j$  において  $0 \leq d(i) < d(j)$  を満たすものとする。合わせて、 $\rho > 0$ 、ベクトル  $Z$  のユークリッド・ノルムは 1 であるとする。

### 入力パラメータ

$i$	INTEGER。計算する固有値のインデックス。 $i = 1$ または $i = 2$
$d$	REAL (slasd5 の場合) DOUBLE PRECISION (dlasd5 の場合) 配列、次元は (2)。 元の固有値。 $0 \leq d(1) < d(2)$ とする。
$z$	REAL (slasd5 の場合) DOUBLE PRECISION (dlasd5 の場合) 配列、次元は (2)。 更新ベクトルの成分。
$\rho$	REAL (slasd5 の場合) DOUBLE PRECISION (dlasd5 の場合) 対称更新式におけるスカラ。
$work$	REAL (slaed0 の場合) DOUBLE PRECISION (dlaed0 の場合) ワークスペース配列、次元は (2) $work$ の $j$ 番目の成分には $(d(j) + \sigma_i)$ を格納する。

### 出力パラメータ

$\delta$	REAL (slasd5 の場合) DOUBLE PRECISION (dlasd5 の場合) 配列、次元は (2)。 $\delta$ の $j$ 番目の成分に $(d(j) - \lambda_i)$ が格納される。ベクトル $\delta$ には固有ベクトルを構成するために必要な情報が格納される。
$d\sigma$	REAL (slasd5 の場合) DOUBLE PRECISION (dlasd5 の場合) 計算された $\lambda_i$ 、 $i$ 番目の更新された固有値。

## ?lasd6

2 つの小さい行列を行追加によって併合して得た、更新された上二重対角行列の SVD を計算する。  
?bdsdc で使用される。

### 構文

```
call slasd6(  icompq, nl, nr, sqre, d, vf, vl, alpha, beta, idxq, perm, givptr,
              givcol, ldgcol, givnum, ldgnum, poles, difl, difr, z, k, c, s, work, iwork,
              info )
call dlasd6(  icompq, nl, nr, sqre, d, vf, vl, alpha, beta, idxq, perm, givptr,
              givcol, ldgcol, givnum, ldgnum, poles, difl, difr, z, k, c, s, work, iwork,
              info )
```

### 説明

ルーチン ?lasd6 は、2 つの小さい行列を行追加によって併合して得た、更新された上二重対角行列  $B$  の SVD を計算する。このルーチンは、特異値すべてと、オプションで因子分解形式の特異ベクトル行列を必要とする問題でのみ使用される。 $B$  は  $n \times m$  の行列で、 $n = nl + nr + 1$  かつ  $m = n + sqre$  である。関連サブルーチン [?lasd1](#) は、二重対角行列のすべての特異値と特異ベクトルが必要な場合を扱う。?lasd6 は次のように SVD を計算する。

$$B = U(in) * \begin{bmatrix} D1(in) & 0 & 0 & 0 \\ Z1' & a & Z2' & b \\ 0 & 0 & D2(in) & 0 \end{bmatrix} * VT(in)$$

$$= U(out) * (D(out) \quad 0) * VT(out)$$

$Z = (Z1' a Z2' b) = u' VT$ 、 $u$  は次元  $m$  のベクトルで、 $n1+1$  番目と  $n1+2$  番目の成分に  $alpha$  と  $beta$  を持ち他はゼロである。 $sqre = 0$  の場合、成分  $b$  は空である。

$B$  の特異値は、下ブロックの全右特異ベクトルの最初の成分  $D1$  と、上ブロックの全右特異ベクトルの最後の成分  $D2$  を使って計算できる。これら成分は ?lasd6 の中で、それぞれ  $vf$  と  $vl$  に格納され更新される。そのため  $U$  と  $VT$  は明示的には参照されない。特異値は  $D$  に格納される。アルゴリズムは 2 つの過程で構成される。



第 1 の過程では、複数の特異値がある場合、または  $Z$  ベクトルにゼロがある場合、問題の大きさの収縮を行う。そのような場合ごとに、永年方程式問題の次元は 1 だけ縮小される。この過程はルーチン [?lasd7](#) によって実行される。

第 2 の過程では更新された特異値を計算する。これは、ルーチン [?lasd4](#) ([?lasd8](#) として呼び出される) を介して永年方程式の根を見つけることによって行われる。また、このルーチンは、 $vf$  と  $vl$  を更新し、更新された特異値と古い特異値の距離を計算する。[?lasd6](#) は [?lasda](#) から呼び出される。

## 入力パラメータ

<i>icompq</i>	INTEGER。因子分解形式で特異ベクトルを計算するかどうかを指定する。 = 0 の場合、特異値のみ計算する。 = 1 の場合、合わせて特異ベクトルを因子分解形式で計算する。
<i>nl</i>	INTEGER。上ブロックの行次元。 $nl \geq 1$
<i>nr</i>	INTEGER。下ブロックの行次元。 $nr \geq 1$
<i>sqre</i>	INTEGER。 $sqre = 0$ の場合、下ブロックは $nr \times nr$ の正方行列。 $= 1$ の場合、下ブロックは $nr \times (nr+1)$ 矩形行列である。 二重対角行列は行次元 $n = nl + nr + 1$ 、列次元 $m = n + sqre$ を持つ。
<i>d</i>	REAL ( <a href="#">slasd6</a> の場合) DOUBLE PRECISION ( <a href="#">dlasd6</a> の場合) 配列、次元は $(nl+nr+1)$ 。 $d(1:nl, 1:nl)$ には上ブロックの特異値を格納し、 $d(nl+2:n)$ には下ブロックの特異値を格納する。
<i>vf</i>	REAL ( <a href="#">slasd6</a> の場合) DOUBLE PRECISION ( <a href="#">dlasd6</a> の場合) 配列、次元は $(m)$ 。 $vf(1:nl+1)$ には上ブロックの全右特異ベクトルの最初の成分を格納し、 $vf(nl+2:m)$ には下ブロックの全右左特異ベクトルの最初の成分を格納する。
<i>vl</i>	REAL ( <a href="#">slasd6</a> の場合) DOUBLE PRECISION ( <a href="#">dlasd6</a> の場合) 配列、次元は $(m)$ 。 $vl(1:nl+1)$ には上ブロックの全右特異ベクトルの最後の成分を格納し、 $vl(nl+2:m)$ には下ブロックの全右左特異ベクトルの最後の成分を格納する。

<i>alpha</i>	REAL (slasd6 の場合 ) DOUBLE PRECISION (dlasd6 の場合 ) 追加した行に関連する対角成分。
<i>beta</i>	REAL (slasd6 の場合 ) DOUBLE PRECISION (dlasd6 の場合 ) 追加した行に関連する非対角成分。
<i>ldgcol</i>	INTEGER。出力配列 <i>givcol</i> のリーディング・ディメンジョン。 <i>n</i> 以上でなければならない。
<i>ldgnum</i>	INTEGER。出力配列 <i>givnum</i> 、 <i>poles</i> のリーディング・ディメンジョン。 <i>n</i> 以上でなければならない。
<i>work</i>	REAL (slasd6 の場合 ) DOUBLE PRECISION (DLASD6 の場合 ) ワークスペース配列、次元は $(4m)$ 。
<i>iwork</i>	INTEGER ワークスペース 配列、次元は $(3n)$

## 出力パラメータ

<i>d</i>	$d(1:n)$ は更新された行列の固有値で上書きされる。
<i>vf</i>	<i>vf</i> は二重対角行列の全右特異ベクトルの最初の成分で上書きされる。
<i>v1</i>	<i>v1</i> は二重対角行列の全右特異ベクトルの最後の成分で上書きされる。
<i>idxq</i>	INTEGER。 配列、次元は $(n)$ 。解かれたばかりの部分問題をソートされた順に再統合する置換で上書きされる。すなわち、 $d(idxq(i = 1, n))$ は昇順となる。
<i>perm</i>	INTEGER。 配列、次元は $(n)$ 。各固有ブロックに適用される (収縮とソートによる) 置換が格納される。 <i>icompq</i> = 0 の場合は参照されない。
<i>givptr</i>	INTEGER。この部分問題で実行された Givens 回転の回数が格納される。 <i>icompq</i> = 0 の場合は参照されない。
<i>givcol</i>	INTEGER。 配列、次元は $(ldgcol, 2)$ 。それぞれの数のペアは Givens 回転で対象となる列ペアを示す。 <i>icompq</i> = 0 の場合は参照されない。

<i>givnum</i>	<p>REAL (slasd6 の場合 )</p> <p>DOUBLE PRECISION (DLASD6 の場合 )</p> <p>配列、次元は (<i>ldgnum</i>, 2)。それぞれの値は対応する Givens 回転で使用される <i>C</i> 値または <i>S</i> 値を示す。<i>icompg</i> = 0 の場合は参照されない。</p>
<i>poles</i>	<p>REAL (slasd6 の場合 )</p> <p>DOUBLE PRECISION (dlasd6 の場合 )</p> <p>配列、次元は (<i>ldgnum</i>, 2)。配列 <i>poles</i>(1, *) には永年方程式を解いて得られた新しい特異値が格納される。配列 <i>poles</i>(2, *) には永年方程式内の極が格納される。<i>icompg</i> = 0 の場合は参照されない。</p>
<i>difl</i>	<p>REAL (slasd6 の場合 )</p> <p>DOUBLE PRECISION (dlasd6 の場合 )</p> <p>配列、次元は (<i>n</i>)。 <i>difl</i>(<i>i</i>) には <i>i</i> 番目の更新された (収縮されていない) 特異値と <i>i</i> 番目の (収縮されていない) 古い特異値との距離が格納される。</p>
<i>difr</i>	<p>REAL (slasd6 の場合 )</p> <p>DOUBLE PRECISION (dlasd6 の場合 )</p> <p>配列、次元は (<i>ldgnum</i>, 2) (<i>icompg</i> = 1 の場合)、次元は (<i>n</i>) (<i>icompg</i> = 0 の場合)。</p> <p><i>difr</i>(<i>i</i>, 1) には <i>i</i> 番目の更新された (収縮されていない) 特異値と <i>i</i>+1 番目の (収縮されていない) 古い特異値との距離が格納される。</p> <p><i>icompg</i> = 1 の場合、配列 <i>difr</i>(1:<i>k</i>, 2) には右特異ベクトル行列に対する正規化係数が格納される。</p> <p><i>difl</i> と <i>difr</i> の詳細は ?lasd8 を参照のこと。</p>
<i>z</i>	<p>real (slasd6 の場合 )</p> <p>DOUBLE PRECISION (DLASD6 の場合 )</p> <p>配列、次元は (<i>m</i>)。</p> <p>この配列の先頭成分には収縮調整された更新行ベクトルが格納される。</p>
<i>k</i>	<p>INTEGER。非収縮行列の次元を格納する。これは関連する永年方程式の次数である。<math>1 \leq k \leq n</math>。</p>
<i>c</i>	<p>REAL (slasd6 の場合 )</p> <p>DOUBLE PRECISION (dlasd6 の場合 )</p> <p><i>sqre</i> = 0 の場合はガーベッジを格納し、<i>sqre</i> = 1 の場合は右ヌル空間に関する Givens 回転の <i>C</i> 値を格納する。</p>

*s* REAL (slasd6 の場合)  
DOUBLE PRECISION (dlasd6 の場合)  
*sqre*=0 の場合はガーベッジを格納し、*sqre*=1 の場合は右スル空間  
に関する Givens 回転の *S* 値を格納する。

*info* INTEGER。  
= 0 の場合、正常に終了したことを示す。  
< 0 の場合、*info* = -*i* は *i* 番目の引数の値が不正だったことを示す。  
> 0 の場合、*info* = 1 は固有値が収束しなかったことを示す。

---

### ?lasd7

2 つの特異値のセットをソートされた単一のセッ  
トに併合する。続いて、問題の大きさの収縮を試  
みる。*?bdsdc* で使用される。

---

#### 構文

```
call slasd7( icompg, nl, nr, sqre, k, d, z, zw, vf, vfw, vl, vlw, alpha, beta,  
            dsigma, idx, idxp, idxq, perm, givptr, givcol, ldgcol, givnum, ldgnum, c, s,  
            info )  
call dlasd7( icompg, nl, nr, sqre, k, d, z, zw, vf, vfw, vl, vlw, alpha, beta,  
            dsigma, idx, idxp, idxq, perm, givptr, givcol, ldgcol, givnum, ldgnum, c, s,  
            info )
```

#### 説明

ルーチン *?lasd7* は 2 つの特異値のセットをソートされた単一のセットに併合する。続  
いて、問題の大きさの収縮を試みる。収縮が起こり得るには 2 つの場合がある。2 個以  
上の特異値が互いに近い場合、または、*Z* ベクトルに微小成分がある場合である。その  
ような条件が発生するごとに、関連する永年方程式問題の次元は 1 だけ縮小される。  
*?lasd7* は [?lasd6](#) から呼び出される。

#### 入力パラメータ

*icompg* INTEGER。コンパクト形式で特異ベクトルを計算するかどうかを指定  
する。  
= 0 の場合、特異値のみ計算する。

	<p>= 1 の場合、上二重対角行列の特異ベクトルをコンパクト形式で計算する。</p>
<i>nl</i>	<p>INTEGER。上ブロックの行次元。 <math>nl \geq 1</math></p>
<i>nr</i>	<p>INTEGER。下ブロックの行次元。 <math>nr \geq 1</math></p>
<i>sqre</i>	<p>INTEGER。 = 0 の場合、下ブロックは <math>nr \times nr</math> 平方行列である。 = 1 の場合、下ブロックは <math>nr \times (nr+1)</math> 矩形行列である。二重対角行列は行次元 <math>n = nl + nr + 1</math> と列次元 <math>m = n + sqre \geq n</math> を持つ。</p>
<i>d</i>	<p>REAL (slasd7 の場合) DOUBLE PRECISION (dlasd7 の場合) 配列、次元は (n)。d には結合を行う 2 つの部分行列の特異値を格納する。</p>
<i>zw</i>	<p>REAL (slasd7 の場合) DOUBLE PRECISION (dlasd7 の場合) 配列、次元は (m)。z 用のワークスペース。</p>
<i>vf</i>	<p>REAL (slasd7 の場合) DOUBLE PRECISION (dlasd7 の場合) 配列、次元は (m)。vf(1:nl+1) には上ブロックの全右特異ベクトルの最初の成分を格納し、vf(nl+2:m) には下ブロックの全右左特異ベクトルの最初の成分を格納する。</p>
<i>vfw</i>	<p>REAL (slasd7 の場合) DOUBLE PRECISION (dlasd7 の場合) 配列、次元は (m)。vf 用のワークスペース。</p>
<i>vl</i>	<p>REAL (slasd7 の場合) DOUBLE PRECISION (dlasd7 の場合) 配列、次元は (m)。vl(1:nl+1) には上ブロックの全右特異ベクトルの最後の成分を格納し、vl(nl+2:m) には下ブロックの全右左特異ベクトルの最後の成分を格納する。</p>
<i>vlw</i>	<p>REAL (slasd7 の場合) DOUBLE PRECISION (dlasd7 の場合) 配列、次元は (m)。vl 用のワークスペース。</p>

<i>alpha</i>	REAL (slasd7 の場合 ) DOUBLE PRECISION (dlasd7 の場合 ) 追加した行に関連する対角成分。
<i>beta</i>	REAL (slasd7 の場合 ) DOUBLE PRECISION (dlasd7 の場合 ) 追加した行に関連する非対角成分。
<i>idx</i>	INTEGER。 ワークスペース配列、次元は (n)。d の内容を昇順でソートしたときに使用した置換を格納する。
<i>idxp</i>	INTEGER。 ワークスペース配列、次元は (n)。収縮した d の値を配列の終わりに配置した置換を格納する。
<i>idxq</i>	INTEGER。 配列、次元は (n)。d に入っている 2 つの部分問題を昇順で別々にソートする置換。この置換の前半にある成分は、最初に 1 つの位置だけ後ろに動かされなければならない。また、この置換の後半にある成分は、最初にそれら値に n1+1 を加算しなければならない。
<i>ldgcol</i>	INTEGER。出力配列 <i>givcol</i> のリーディング・ディメンジョン。 n 以上でなければならない。
<i>ldgnum</i>	INTEGER。出力配列 <i>givnum</i> のリーディング・ディメンジョン。 n 以上でなければならない。

## 出力パラメータ

<i>k</i>	INTEGER。収縮されていない行列の次元が格納される。これは関連する永年方程式の次数である。 $1 \leq k \leq n$
<i>d</i>	d は、後続の (n-k) 個の更新された特異値 (収縮された) の昇順で上書きされる。
<i>z</i>	REAL (slasd7 の場合 ) DOUBLE PRECISION (dlasd7 の場合 ) 配列、次元は (m)。z は永年方程式内の更新列ベクトルで上書きされる。
<i>vf</i>	vf は二重対角行列の全右特異値の最初の成分で上書きされる。
<i>v1</i>	v1 は二重対角行列の全右特異ベクトルの最後の成分で上書きされる。

<i>dsigma</i>	REAL (slasd7 の場合 ) DOUBLE PRECISION (dlasd7 の場合 ) 配列、次元は (n)。永年方程式内の対角成分 (k-1 個の特異値と 1 個のゼロ) のコピーが格納される。
<i>idxp</i>	<i>idxp</i> (2:k) は収縮されていない <i>d</i> 値を指し、 <i>idxp</i> (k+1:n) は収縮された特異値を指す。
<i>perm</i>	INTEGER。 配列、次元は (n)。各固有ブロックに適用される (収縮とソートによる) 置換が格納される。 <i>icompq</i> = 0 の場合は参照されない。
<i>givptr</i>	INTEGER。この部分問題で実行された Givens 回転の回数が格納される。 <i>icompq</i> = 0 の場合は参照されない。
<i>givcol</i>	INTEGER。 配列、次元は (ldgcol, 2)。それぞれの数のペアは Givens 回転で対象となる列ペアを示す。 <i>icompq</i> = 0 の場合は参照されない。
<i>givnum</i>	REAL (slasd7 の場合 ) DOUBLE PRECISION (dlasd7 の場合 ) 配列、次元は (ldgnum, 2)。それぞれの値は対応する Givens 回転で使用する <i>C</i> 値または <i>S</i> 値を示す。 <i>icompq</i> = 0 の場合は参照されない。
<i>c</i>	REAL (slasd7 の場合 ) DOUBLE PRECISION (dlasd7 の場合 ) <i>sqre</i> = 0 の場合はガーベッジが格納され、 <i>sqre</i> = 1 の場合は右ヌル空間に関する Givens 回転の <i>C</i> 値が <i>c</i> に格納される。
<i>s</i>	REAL (slasd7 の場合 ) DOUBLE PRECISION (dlasd7 の場合 ) <i>sqre</i> = 0 の場合はガーベッジが格納され、 <i>sqre</i> = 1 の場合は右ヌル空間に関する Givens 回転の <i>S</i> 値が <i>s</i> に格納される。
<i>info</i>	INTEGER。 = 0 の場合、正常に終了したことを示す。 < 0 の場合、 <i>info</i> = - <i>i</i> は <i>i</i> 番目の引数の値が不正だったことを示す。

### ?lasd8

永年方程式の根の平方根を求め、 $D$  の各成分に対して最も近い2つの極までの距離を格納する。  
?bdsdc で使用される。

---

#### 構文

```
call slasd8(  icompq, k, d, z, vf, vl, difl, difr, lddifr, dsigma, work, info )
call dlasd8(  icompq, k, d, z, vf, vl, difl, difr, lddifr, dsigma, work, info )
```

#### 説明

ルーチン ?lasd8 は、 $dsigma$  と  $z$  の値によって定義されている永年方程式に対して根の平方根を求める。[?lasd4](#) に対する適切な呼び出しを実行し、続いて  $d$  の各成分に対して最も近い2つの極までの距離 ( $dsigma$  内の成分) を格納する。また、元の二重対角行列の全右特異ベクトルの最初と最後の成分である配列  $vf$  と  $vl$  を更新する。?lasd8 は [?lasd6](#) から呼び出される。

#### 入力パラメータ

<i>icompq</i>	INTEGER。呼び出されたルーチン内で、因子分解形式で特異ベクトルを計算するかどうかを指定する。 = 0 の場合、特異値のみ計算する。 = 1 の場合、合わせて特異ベクトルを因子分解形式で計算する。
<i>k</i>	INTEGER。?lasd4 で解かれる有利関数の項目数。 $k \geq 1$
<i>z</i>	REAL (slasd8 の場合) DOUBLE PRECISION (dlasd8 の場合) 配列、次元は ( $k$ )。この配列の先頭の $k$ 個の成分には収縮調整された更新行ベクトルの成分を格納する。
<i>vf</i>	REAL (slasd8 の場合) DOUBLE PRECISION (dlasd8 の場合) 配列、次元は ( $k$ )。 <i>vf</i> には <i>dbede8</i> から渡された情報を格納する。
<i>vl</i>	REAL (slasd8 の場合) DOUBLE PRECISION (dlasd8 の場合) 配列、次元は ( $k$ )。 <i>vl</i> には <i>dbede8</i> から渡された情報を格納する。



<i>lddifr</i>	INTEGER。出力配列 <i>difr</i> のリーディング・ディメンジョン。 $k$ 以上でなければならない。
<i>dsigma</i>	REAL (slasd8 の場合) DOUBLE PRECISION (dlasd8 の場合) 配列、次元は $(k)$ 。この配列の先頭の $k$ 個の成分には収縮された更新問題の古い根を格納する。これらは永年方程式の極である。
<i>work</i>	REAL (slasd8 の場合) DOUBLE PRECISION (dlasd8 の場合) ワークスペース配列、次元は $(3k)$ 以上。

### 出力パラメータ

<i>d</i>	REAL (slasd8 の場合) DOUBLE PRECISION (dlasd8 の場合) 配列、次元は $(k)$ 。 <i>d</i> は更新された特異ベクトルで上書きされる。
<i>vf</i>	<i>vf</i> は二重対角行列の全右特異ベクトルの最初の成分の先頭の $k$ 個の成分で上書きされる。
<i>v1</i>	<i>v1</i> は二重対角行列の全右特異ベクトルの最後の成分の先頭の $k$ 個の成分で上書きされる。
<i>difl</i>	REAL (slasd8 の場合) DOUBLE PRECISION (dlasd8 の場合) 配列、次元は $(k)$ 。 $difl(i) = d(i) - dsigma(i)$ が格納される。
<i>difr</i>	REAL (slasd8 の場合) DOUBLE PRECISION (dlasd8 の場合) 配列、 DIMENSION ( <i>lddifr</i> , 2 ) ( <i>icompq</i> = 1 の場合 ) または 次元は $(k)$ ( <i>icompq</i> = 0 の場合 )。 $difr(i, 1) = d(i) - dsigma(i+1)$ が格納される。 <i>difr</i> ( $k, 1$ ) は定義されて なく参照されない。 <i>icompq</i> = 1 の場合、配列 <i>difr</i> (1: $k, 2$ ) には右特異ベクトル行列に対する 正規化係数が格納される。
<i>info</i>	INTEGER。 = 0 の場合、正常に終了したことを示す。 < 0 の場合、 <i>info</i> = - <i>i</i> は <i>i</i> 番目の引数の値が不正だったことを示す。 > 0 の場合、 <i>info</i> = 1 は固有値が収束しなかったことを示す。

## ?lasd9

永年方程式の根の平方根を求め、 $D$  の各成分に対して最も近い2つの極までの距離を格納する。  
?bdsdc で使用される。

### 構文

```
call slasd9(  icompq, ldu, k, d, z, vf, vl, difl, difr, dsigma, work, info )  
call dlasd9(  icompq, ldu, k, d, z, vf, vl, difl, difr, dsigma, work, info )
```

### 説明

ルーチン ?lasd9 は、 $dsigma$  と  $z$  の値によって定義されている永年方程式に対して根の平方根を求める。[?lasd4](#) に対する適切な呼び出しを実行し、続いて  $d$  の各成分に対して最も近い2つの極までの距離 ( $dsigma$  内の成分) を格納する。また、元の二重対角行列の全右特異ベクトルの最初と最後の成分である配列  $vf$  と  $vl$  を更新する。?lasd9 は [?lasd7](#) から呼び出される。

### 入力パラメータ

<i>icompq</i>	INTEGER。呼び出されたルーチン内で、因子分解形式で特異ベクトルを計算するかどうかを指定する。 <i>icompq</i> = 0 の場合、特異値のみ計算する。 <i>icompq</i> = 1 の場合、合わせて特異ベクトルを因子分解形式で計算する。
<i>k</i>	INTEGER。slasd4 で解かれる有利関数の項目数。 $k \geq 1$
<i>dsigma</i>	REAL (slasd9 の場合) DOUBLE PRECISION (dlasd9 の場合) 配列、次元は ( $k$ )。この配列の先頭の $k$ 個の成分には収縮された更新問題の古い根を格納する。これらは永年方程式の極である。
<i>z</i>	REAL (slasd9 の場合) DOUBLE PRECISION (dlasd9 の場合) 配列、次元は ( $k$ )。この配列の先頭の $k$ 個の成分には収縮調整された更新行ベクトルの成分を格納する。

<i>vf</i>	REAL (slasd9 の場合 ) DOUBLE PRECISION (dlasd9 の場合 ) 配列、次元は (k)。vf には sbede8 から渡された情報を格納する。
<i>vl</i>	REAL (slasd9 の場合 ) DOUBLE PRECISION (dlasd9 の場合 ) 配列、次元は (k)。vl には sbede8 から渡された情報を格納する。
<i>work</i>	REAL (slasd9 の場合 ) DOUBLE PRECISION (dlasd9 の場合 ) ワークスペース配列、次元は (3k) 以上。

### 出力パラメータ

<i>d</i>	REAL (slasd9 の場合 ) DOUBLE PRECISION (dlasd9 の場合 ) 配列、次元は (k)。d(i) は更新された特異ベクトルで上書きされる。
<i>vf</i>	vf は二重対角行列の全右特異ベクトルの最初の成分の先頭の k 個の成分で上書きされる。
<i>vl</i>	vl は二重対角行列の全右特異ベクトルの最後の成分の先頭の k 個の成分で上書きされる。
<i>difl</i>	REAL (slasd9 の場合 ) DOUBLE PRECISION (dlasd9 の場合 ) 配列、次元は (k)。 $difl(i) = d(i) - d\sigma(i)$ が格納される。
<i>difr</i>	REAL (slasd9 の場合 ) DOUBLE PRECISION (dlasd9 の場合 ) 配列、次元は (ldu, 2) (icompq=1 の場合 ) または (k) (icompq=0 の場合 ) $difr(i, 1) = d(i) - d\sigma(i+1)$ が格納される。difr(k, 1) は定義され なく参照されない。 icompq=1 の場合、配列 difr(1:k, 2) には右特異ベクトル行列に対す る正規化係数が格納される。
<i>info</i>	INTEGER。 = 0 の場合、正常に終了したことを示す。 < 0 の場合、info = -i は i 番目の引数の値が不正だったことを示す。 > 0 の場合、info = 1 は固有値が収束しなかったことを示す。

## ?lasda

対角  $d$  と対角外  $e$  を持つ実数上二重対角行列の特異値分解 (SVD) を計算する。  
?bdsdc で使用される。

### 構文

```
call slasda( icompg, smlsiz, n, sqre, d, e, u, ldu, vt, k, difl, difr, z, poles,  
            givptr, givcol, ldgcol, perm, givnum, c, s, work, iwork, info )  
call dlasda( icompg, smlsiz, n, sqre, d, e, u, ldu, vt, k, difl, difr, z, poles,  
            givptr, givcol, ldgcol, perm, givnum, c, s, work, iwork, info )
```

### 説明

ルーチン ?lasda は、分割統治法を使って、対角  $d$  と対角外  $e$  を持つ実数上二重対角  $n \times m$  行列  $B$  の特異値分解 (SVD) を計算する。ここで  $m = n + sqre$ 。このアルゴリズムは SVD で特異値  $B = U * S * VT$  を計算する。直交行列  $U$  と  $VT$  の計算はオプションであり、コンパクト形式で計算される。関連するサブルーチン [?lasd0](#) は特異値と特異ベクトルを黙示的な形式で計算する。

### 入力パラメータ

<i>icompg</i>	INTEGER。コンパクト形式で特異ベクトルを計算するかどうかを指定する。 = 0 の場合、特異値のみ計算する。 = 1 の場合、上二重対角行列の特異ベクトルをコンパクト形式で計算する。
<i>smlsiz</i>	INTEGER。計算ツリーの一番下の部分問題の最大サイズ。
<i>n</i>	INTEGER。上二重対角行列の行次元。これはまた、主対角配列 $d$ の次元でもある。
<i>sqre</i>	INTEGER。二重対角行列の列のサイズを指定する。 $sqre = 0$ の場合、二重対角行列は列次元 $m = n$ を持つ。 $sqre = 1$ の場合、二重対角行列は列次元 $m = n + 1$ を持つ。
<i>d</i>	REAL (slasda の場合) DOUBLE PRECISION (dlasda の場合) 配列、次元は $(n)$ 。 $d$ には二重対角行列の主対角を格納する。

<i>e</i>	REAL (slasda の場合 ) DOUBLE PRECISION (dlasda の場合 ) 配列、次元は $(m-1)$ 。二重対角行列の劣対角成分を格納する。出力において <i>e</i> の内容は壊される。
<i>ldu</i>	INTEGER。配列 <i>u</i> 、 <i>vt</i> 、 <i>difl</i> 、 <i>difr</i> 、 <i>poles</i> 、 <i>givnum</i> 、 <i>z</i> のリーディング・ディメンジョン。 $ldu \geq n$ 。
<i>ldgcol</i>	INTEGER。配列 <i>givcol</i> 、 <i>perm</i> のリーディング・ディメンジョン。 $ldgcol \geq n$
<i>work</i>	REAL (slasda の場合 ) DOUBLE PRECISION (dlasda の場合 ) ワークスペース配列、次元は $(6n + (smlsiz + 1)^2)$
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $(7n)$ 以上でなければならない。

### 出力パラメータ

<i>d</i>	<i>info</i> = 0 の場合、二重対角行列の特異値が格納される。
<i>u</i>	REAL (slasda の場合 ) DOUBLE PRECISION (dlasda の場合 ) 配列、次元は $(ldu, smlsiz)$ ( <i>icompq</i> = 1 の場合 ) <i>icompq</i> = 0 の場合は参照されない。 <i>icompq</i> = 1 の場合、 <i>u</i> には一番下のレベルの全部分問題の左特異ベクトル行列が格納される。
<i>vt</i>	REAL (slasda の場合 ) DOUBLE PRECISION (dlasda の場合 ) 配列、次元は $(ldu, smlsiz+1)$ ( <i>icompq</i> = 1 の場合 )。 <i>icompq</i> = 0 の場合は参照されない。 <i>icompq</i> = 1 の場合、 <i>vt</i> には一番下のレベルの全部分問題の右特異ベクトル行列が格納される。
<i>k</i>	INTEGER。 配列、 次元は $(n)$ ( <i>icompq</i> = 1 の場合 )。 次元は $(1)$ ( <i>icompq</i> = 0 の場合 )。 <i>icompq</i> = 1 の場合、 <i>k(i)</i> には計算ツリーにおける <i>i</i> 番目の永年方程式の次元が格納される。
<i>difl</i>	REAL (slasda の場合 ) DOUBLE PRECISION (dlasda の場合 ) 配列、次元は $(ldu, nlv1)$ 、ここで $nlv1 = \text{floor}(\log_2(n/smlsiz))$ 。

<i>difr</i>	<p>REAL (slasda の場合 )  DOUBLE PRECISION (dlasda の場合 )  配列、次元は <math>(ldu, 2 \cdot nlv1)</math> (<math>icompq = 1</math> の場合 ) または <math>(n)</math> (<math>icompq = 0</math> の場合 )。  <math>icompq = 1</math> の場合、<math>difl(1:n, i)</math> と <math>difr(1:n, 2i-1)</math> には、<math>i</math> 番目のレベルの特異値と <math>(i-1)</math> 番目のレベルの特異値との距離が記録され、<math>difr(1:n, 2i)</math> には右特異ベクトル行列に対する正規化係数が格納される。詳細は ?lasd8 を参照のこと。</p>
<i>z</i>	<p>REAL (slasda の場合 )  DOUBLE PRECISION (dlasda の場合 )  配列、次元は <math>(ldu, nlv1)</math> (<math>icompq = 1</math> の場合 ) または <math>(n)</math> (<math>icompq = 0</math> の場合 )。  <math>z(1, i)</math> の最初の <math>k</math> 個の成分には、<math>i</math> 番目のレベルの部分問題に対する収縮調整された更新行ベクトルの成分が格納される。</p>
<i>poles</i>	<p>REAL (slasda の場合 )  DOUBLE PRECISION (dlasda の場合 )  配列、次元は <math>(ldu, 2 \cdot nlv1)</math> (<math>icompq = 1</math> の場合 )。 <math>icompq = 0</math> の場合は参照されない。 <math>icompq = 1</math> の場合、<math>poles(1, 2i-1)</math> と <math>poles(1, 2i)</math> には <math>i</math> 番目レベルの永年方程式に関係する新しい特異値と古い特異値が格納される。</p>
<i>givptr</i>	<p>INTEGER。  配列、次元は <math>(n)</math> (<math>icompq = 1</math> の場合 )。 <math>icompq = 0</math> の場合は参照されない。 <math>icompq = 1</math> の場合、<math>givptr(i)</math> には計算ツリーの <math>i</math> 番目問題で実行された Givens 回転の回数が格納される。</p>
<i>givcol</i>	<p>INTEGER。  配列、次元は <math>(ldgcol, 2 \cdot nlv1)</math> (<math>icompq = 1</math> の場合 )。 <math>icompq = 0</math> の場合は参照されない。 <math>icompq = 1</math> の場合、<math>givcol(1, 2i-1)</math> と <math>givcol(1, 2i)</math> には計算ツリーの <math>i</math> 番目問題で実行された Givens 回転の位置が格納される。</p>
<i>perm</i>	<p>INTEGER  配列、次元は <math>(ldgcol, nlv1)</math> (<math>icompq = 1</math> の場合 )。 <math>icompq = 0</math> の場合は参照されない。 <math>icompq = 1</math> の場合、<math>perm(1, i)</math> には計算ツリーの <math>i</math> 番目問題で実行された置換が格納される。</p>
<i>givnum</i>	<p>REAL (slasda の場合 )  DOUBLE PRECISION (dlasda の場合 )  配列、次元は <math>(ldu, 2 \cdot nlv1)</math> (<math>icompq = 1</math> の場合 )。 <math>icompq = 0</math> の場合</p>

は参照されない。 $icompq=1$  の場合、 $givnum(1, 2i-1)$  と  $givnum(1, 2i)$  には計算ツリーの  $i$  番目問題で実行された Givens 回転の  $C$  値と  $S$  値が格納される。

$c$	REAL (slasda の場合 ) DOUBLE PRECISION (dlasda の場合 ) 配列、 次元は $(N)$ ( $icompq=1$ の場合 ) または 次元は $(1)$ ( $icompq=0$ の場合 )。 $icompq=1$ で $i$ 番目の部分問題が正方ではない場合、 $c(i)$ には $i$ 番目の部分問題の右ヌル空間に関する Givens 回転の $C$ 値が格納される。
$s$	REAL (slasda の場合 ) DOUBLE PRECISION (dlasda の場合 ) 配列、次元は $(n)$ ( $icompq=1$ の場合 )。 $(1)$ ( $icompq=0$ の場合 )。 $icompq=1$ で $i$ 番目の部分問題が正方ではない場合、 $s(i)$ には $i$ 番目の部分問題の右ヌル空間に関する Givens 回転の $S$ 値が格納される。
$info$	INTEGER。 $=0$ の場合、正常に終了したことを示す。 $<0$ の場合、 $info=-i$ は $i$ 番目の引数の値が不正だったことを示す。 $>0$ の場合、 $info=1$ は固有値が収束しなかったことを示す。

## ?lasdq

対角  $d$  と対角外  $e$  を持つ実二重対角行列を  
計算する。?bdsdc で使用される。

### 構文

```
call slasdq( uplo, sqre, n, ncv, nru, ncc, d, e, vt, ldvt, u, ldu, c, ldc,
            work, info )
call dlasdq( uplo, sqre, n, ncv, nru, ncc, d, e, vt, ldvt, u, ldu, c, ldc,
            work, info )
```

### 説明

ルーチン ?lasdq は、対角  $d$  と対角外  $e$  を持つ実数 ( 上または下 ) 二重対角行列の特異値分解 (svd) を、必要に応じて変換の蓄積を行いながら計算する。 $B$  を入力 of 二重対角行列としたとき、このアルゴリズムは  $B = Q S P'$  であるような直交行列  $Q$  と  $P$  を計算する ( $P'$  は  $P$  の転置)。特異値  $S$  は  $d$  に上書きされる。

必要に応じて入力行列  $U$  は  $UQ$  に変更される。  
 必要に応じて入力行列  $VT$  は  $P'VT$  に変更される。  
 必要に応じて入力行列  $C$  は  $Q'C$  に変更される。

## 入力パラメータ

<code>uplo</code>	CHARACTER*1。 <code>uplo</code> は入力二重対角行列が上二重対角か下二重対角かを指定する。 <code>uplo = 'U'</code> または <code>'u'</code> の場合、 $B$ は上二重対角。 <code>uplo = 'L'</code> または <code>'l'</code> の場合、 $B$ は下二重対角。
<code>sqre</code>	INTEGER。 $= 0$ の場合、入力行列は $n \times n$ $= 1$ の場合、 <code>uplu = 'U'</code> ならば入力行列は $n \times (n+1)$ 、 <code>uplu = 'L'</code> ならば $(n+1) \times n$ 。二重対角行列は行次元 $n = n_l + n_r + 1$ と列次元 $m = n + sqre \geq n$ を持つ。
<code>n</code>	INTEGER。 $n$ は行列内の行数と列数を指定する。 $n$ は 0 以上でなければならない。
<code>ncvt</code>	INTEGER。 <code>ncvt</code> は行列 $VT$ の列数を指定する。 <code>ncvt</code> は 0 以上でなければならない。
<code>nru</code>	INTEGER。 <code>nru</code> は行列 $U$ の行数を指定する。 <code>nru</code> は 0 以上でなければならない。
<code>ncc</code>	INTEGER。 <code>ncc</code> は行列 $C$ の列数を指定する。 <code>ncc</code> は 0 以上でなければならない。
<code>d</code>	REAL (slasdq の場合 ) DOUBLE PRECISION (dlasdq の場合 ) 配列、次元は $(n)$ 。 $d$ には、SVD の実行対象である二重対角行列の対角成分を格納する。
<code>e</code>	REAL (slasdq の場合 ) DOUBLE PRECISION (dlasdq の場合 ) 配列、次元は、 <code>sqre = 0</code> の場合は $(n-1)$ 、 <code>sqre = 1</code> の場合は $(n)$ 。 $e$ の成分には、SVD の実行対象である二重対角行列の非対角成分を格納する。
<code>vt</code>	REAL (slasdq の場合 ) DOUBLE PRECISION (dlasdq の場合 ) 配列、次元は $(ldvt, ncvt)$ 。出力時に $P'$ によって事前乗算される行列



	を格納する。 $sGRE = 0$ ならば次元は $n \times ncvT$ 、 $sGRE = 1$ ならば次元は $(n+1) \times ncvT$ ( $ncvT = 0$ の場合は参照されない)。
<i>ldvt</i>	INTEGER。呼び出し (サブ) プログラムで宣言されたとおり、 <i>ldvt</i> には <i>vt</i> のリーディング・ディメンジョンを格納する。 <i>ldvt</i> の値は 1 以上でなければならない。さらに <i>ncvt</i> が非ゼロならば、 <i>ldvt</i> は <i>n</i> 以上でなければならない。
<i>u</i>	REAL ( <i>sLasdq</i> の場合) DOUBLE PRECISION ( <i>dLasdq</i> の場合) 配列、次元は ( <i>ldu</i> , <i>n</i> )。出力時に <i>Q</i> によって事後乗算される行列を格納する。 $sGRE = 0$ ならば次元は $nru \times n$ 、 $sGRE = 1$ ならば次元は $nru \times (n+1)$ ( $nru = 0$ の場合は参照されない)。
<i>ldu</i>	INTEGER。呼び出し (サブ) プログラムで宣言されたとおり、 <i>ldu</i> には <i>u</i> のリーディング・ディメンジョンを格納する。 <i>ldu</i> は $\max(1, nru)$ 以上でなければならない。
<i>c</i>	REAL ( <i>sLasdq</i> の場合) DOUBLE PRECISION ( <i>dLasdq</i> の場合) 配列、次元は ( <i>ldc</i> , <i>ncc</i> )。出力時に <i>q'</i> によって事前乗算される $n \times NCC$ の行列を格納する。 $sGRE = 0$ ならば次元は $n \times ncc$ 、 $sGRE = 1$ ならば次元は $(n+1) \times ncc$ ( $ncc = 0$ の場合は参照されない)。
<i>ldc</i>	INTEGER。呼び出し (サブ) プログラムで宣言されたとおり、 <i>ldc</i> には <i>c</i> のリーディング・ディメンジョンを格納する。 <i>ldc</i> は 1 以上でなければならない。さらに <i>ncc</i> が非ゼロならば、 <i>ldc</i> は <i>n</i> 以上でなければならない。
<i>work</i>	REAL ( <i>sLasdq</i> の場合) DOUBLE PRECISION ( <i>dLasdq</i> の場合) 配列、次元は ( $4n$ )。ワークスペース配列。 <i>ncvt</i> 、 <i>nru</i> 、または <i>ncc</i> のうち 1 つが非ゼロで、かつ、 <i>n</i> が 2 以上の場合にのみ参照される。

## 出力パラメータ

<i>d</i>	<i>d</i> には特異値が昇順で上書きされる。
<i>e</i>	通常終了時は <i>e</i> には 0 が格納される。アルゴリズムが収束しない場合、 <i>d</i> と <i>e</i> には、入力として与えられた行列と直交的に等価な二重対角行列の対角と劣対角成分が格納される。
<i>vt</i>	終了時に、行列は <i>P'</i> によって事前に乗算されている。

<i>u</i>	終了時に、行列は $Q$ によって事後に乗算されている。
<i>c</i>	終了時に、行列は $Q'$ によって事前に乗算されている。
<i>info</i>	INTEGER。ゼロは正常に終了したことを示す。 <i>info</i> < 0 の場合、- <i>info</i> 番目の引数の値が不正であったことを示す。 <i>info</i> > 0 の場合、アルゴリズムが収束しなかったことを示し、 <i>info</i> は収束しなかった優対角の個数を表わす。

---

### ?lasdt

二重対角分割統治に対する部分問題のツリーを生成する。?bdsdc で使用される。

---

#### 構文

```
call slasdt( n, lvl, nd, inode, ndiml, ndimr, msub )  
call dlasdt( n, lvl, nd, inode, ndiml, ndimr, msub )
```

#### 説明

このルーチンは二重対角分割統治に対する部分問題のツリーを生成する。

#### 入力パラメータ

<i>n</i>	INTEGER。二重対角行列の対角成分の個数を格納する。
<i>msub</i>	INTEGER。一番下のツリーにある各部分問題が取り得る最大行次元を格納する。

#### 出力パラメータ

<i>lvl</i>	INTEGER。計算ツリーのレベル数が格納される。
<i>nd</i>	INTEGER。ツリーのノード数が格納される。
<i>inode</i>	INTEGER。 配列、次元は ( <i>n</i> )。部分問題の中央が格納される。
<i>ndiml</i>	INTEGER。 配列、次元は ( <i>n</i> )。左の子の列次元が格納される。
<i>ndimr</i>	INTEGER。 配列、次元は ( <i>n</i> )。右の子の列次元が格納される。

## ?laset

行列の非対角成分と対角成分を指定された値に初期化する。

### 構文

```
call slaset( uplo, m, n, alpha, beta, a, lda )
call dlaset( uplo, m, n, alpha, beta, a, lda )
call claset( uplo, m, n, alpha, beta, a, lda )
call zlaset( uplo, m, n, alpha, beta, a, lda )
```

### 説明

このルーチンは、 $m \times n$  行列  $A$  で、対角を  $\beta$  に、対角外を  $\alpha$  に初期化する。

### 入力パラメータ

<code>uplo</code>	CHARACTER*1。設定対象の行列の部分指定する。 <code>uplo = 'U'</code> の場合、上三角部分が設定される。 $A$ の厳密な下三角部分は変更されない。 <code>uplo = 'L'</code> の場合、下三角部分が設定される。 $A$ の厳密な上三角部分は変更されない。 それ以外では行列 $A$ のすべての部分が設定される。
<code>m</code>	INTEGER。行列 $A$ の行数。 $m \geq 0$
<code>n</code>	INTEGER。行列 $A$ の列数。 $n \geq 0$
<code>alpha, beta</code>	REAL (slaset の場合) DOUBLE PRECISION (dlaset の場合) COMPLEX (claset の場合) COMPLEX*16 (zlaset の場合) それぞれ、非対角成分と対角成分に設定する定数。
<code>a</code>	REAL (slaset の場合) DOUBLE PRECISION (dlaset の場合) COMPLEX (claset の場合) COMPLEX*16 (zlaset の場合) 配列、次元は $(lda, n)$ 。 $m \times n$ の行列 $A$ を格納する。

*lda*                    INTEGER。配列 *A* のリーディング・ディメンジョン。  
 $lda \geq \max(1, m)$

### 出力パラメータ

*a*                    *A* の先頭の  $m \times n$  部分行列は次のように設定される。  
*uplo* = 'U' の場合、 $A(i, j) = \alpha$ 、 $1 \leq i \leq j-1$ 、 $1 \leq j \leq n$ 、  
*uplo* = 'L' の場合、 $A(i, j) = \alpha$ 、 $j+1 \leq i \leq m$ 、 $1 \leq j \leq n$ 、  
それ以外の場合、 $A(i, j) = \alpha$ 、 $1 \leq i \leq m$ 、 $1 \leq j \leq n$ 、 $i \neq j$ 、  
また、すべての *uplo* で  $A(i, i) = \beta$ 、 $1 \leq i \leq \min(m, n)$

---

## ?lasq1

実数平方二重対角行列の特異値を計算する。  
*?bdsqr* で使用される。

---

### 構文

```
call slasq1( n, d, e, work, info )  
call dlasq1( n, d, e, work, info )
```

### 説明

ルーチン *?lasq1* は、対角 *d* と対角外 *e* を持つ実数  $n \times n$  二重対角行列の特異値を計算する。特異値は相対的に高い精度で計算され、非正規化、アンダーフロー、オーバーフローは発生しない。

### 入力パラメータ

*n*                    INTEGER。行列内の行と列の数。  $n \geq 0$

*d*                    REAL(*slasq1* の場合 )  
                     DOUBLE PRECISION(*dlasq1* の場合 )  
配列、次元は (*n*)。 *d* には、*SVD* の実行対象である二重対角行列の対角成分を格納する。

*e*                    REAL(*slasq1* の場合 )  
                     DOUBLE PRECISION(*dlasq1* の場合 )  
配列、次元は (*n*)。成分 *e*(1:*n*-1) には、*SVD* の実行対象である二重対角行列の非対角成分を格納する。

`work` REAL (slasq1 の場合)  
 DOUBLE PRECISION (dlasq1 の場合)  
 ワークスペース配列、次元は  $(4n)$

### 出力パラメータ

`d`  $d$  には特異値が降順で上書きされる。

`e`  $e$  は上書きされる。

`info` INTEGER。  
 $= 0$  の場合、正常に終了したことを示す。  
 $< 0$  の場合、 $info = -i$  は  $i$  番目の引数の値が不正だったことを示す。  
 $> 0$  の場合、アルゴリズムが失敗したことを示す。  
 $= 1$  の場合、分割は  $e$  に格納されている正の値でマークされた。  
 $= 2$  の場合、 $z$  の現在のブロックは  $30*n$  回の反復 (内側の while ループ) によって対角化されなかった。  
 $= 3$  の場合、外側の while ループの終了条件を満たさなかった (プログラムは  $n$  を超える縮退されていないブロックを生成した)。

## ?lasq2

相対的に高い精度で、 $qd$  配列  $z$  に関する対称  
 正定値三重対角行列のすべての固有値を計算す  
 る。`?bdsqr` と `?stegr` で使用される。

### 構文

```
call slasq2( n, z, info )
call dlasq2( n, z, info )
```

### 説明

ルーチン `?lasd2` は、 $qd$  配列  $z$  に関する対称正値三重対角行列のすべての固有値を計算する。固有値は相対的に高い精度で計算され、非正規化、アンダーフロー、オーバーフローは発生しない。

三重対角行列に対する  $z$  の関係を理解するには、 $L$  を劣対角  $z(2, 4, 6, \dots)$  を持つ単位下二重対角行列とし、 $U$  を対角  $z(1, 3, 5, \dots)$  とその上に 1 を持つ上二重対角行列とする。三重対角は  $LU$  となる。または、類似した対称三重対角としてもよい。

## 入力パラメータ

<i>n</i>	INTEGER。行列内の行と列の数。 $n \geq 0$
<i>z</i>	REAL (slasq2 の場合 ) DOUBLE PRECISION (dlasq2 の場合 ) 配列、次元は $(4n)$ 。 <i>z</i> には <i>qd</i> 配列を格納する。

## 出力パラメータ

<i>z</i>	成分 1 から <i>n</i> には固有値が降順で、 <i>z</i> (2 <i>n</i> +1) には対角和が、 <i>z</i> (2 <i>n</i> +2) には固有値の合計が格納される。 <i>n</i> > 2 の場合、 <i>z</i> (2 <i>n</i> +3) には反復回数が、 <i>z</i> (2 <i>n</i> +4) には <i>ndivs</i> / <i>nin</i> <sup>2</sup> が、 <i>z</i> (2 <i>n</i> +5) には失敗したシフトのパーセントが格納される。
<i>info</i>	INTEGER。 = 0 の場合、正常に終了したことを示す。 < 0 の場合、 <i>i</i> 番目の引数がスカラで値が不正だった場合は <i>info</i> = - <i>i</i> 、 <i>i</i> 番目の引数が配列で <i>j</i> 番目の成分の値が不正だった場合は <i>info</i> = -( <i>i</i> *100+ <i>j</i> ) > 0 の場合、アルゴリズムが失敗したことを示す。 = 1 の場合、分割は <i>e</i> に格納されている正値でマークされた。 = 2 の場合、 <i>z</i> の現在のブロックは 30* <i>n</i> 回の反復 ( 内側 while ループ ) によって対角化されなかった。 = 3 の場合、外側 while ループの終了条件を満たさなかった ( プログラムは <i>n</i> を超える縮退されていないブロックを生成した )。

## アプリケーション・ノート

ルーチン ?lasq2 は論理変数 *ieee* を定義する。マシンが IEEE-754 浮動小数点スタンダードに従って無限および NaN を取り扱う場合は .TRUE. となり、そうでない場合は .FALSE. となる。変数は [?lasq3](#) に渡される。

## ?lasq3

収縮をチェックし、シフトを計算し、*dqds* を呼び出す。*?bdsqr* で使用される。

### 構文

```
call slasq3( i0, n0, z, pp, dmin, sigma, desig, qmax, nfail, iter, ndiv, ieee,
            ttype, dmin1, dmin2, dn, dn1, dn2, tau )
call dlasq3( i0, n0, z, pp, dmin, sigma, desig, qmax, nfail, iter, ndiv, ieee,
            ttype, dmin1, dmin2, dn, dn1, dn2, tau )
```

### 説明

ルーチン *?lasq3* は、収縮をチェックし、シフト (*tau*) を計算し、*dqds* を呼び出す。失敗した場合はシフトを変更して出力が正になるまで繰り返す。

### 入力パラメータ

<i>i0</i>	INTEGER。最初のインデックス。
<i>n0</i>	INTEGER。最後のインデックス。
<i>z</i>	REAL ( <i>slasq3</i> の場合 ) DOUBLE PRECISION ( <i>dlasq3</i> の場合 ) 配列、次元は $(4n)$ 。 <i>z</i> には <i>qd</i> 配列を格納する。
<i>pp</i>	INTEGER。 <i>ping</i> のとき <i>pp</i> = 0、 <i>pong</i> のとき <i>pp</i> = 1
<i>desig</i>	REAL ( <i>slasq3</i> の場合 ) DOUBLE PRECISION ( <i>dlasq3</i> の場合 ) <i>sigma</i> の下位次数部分。
<i>qmax</i>	REAL ( <i>slasq3</i> の場合 ) DOUBLE PRECISION ( <i>dlasq3</i> の場合 ) <i>q</i> の最大値。
<i>ieee</i>	LOGICAL。IEEE または非 IEEE 演算を示すフラグ ( <i>?lasq5</i> に渡される )。
<i>dmin1</i>	REAL ( <i>slasq3</i> の場合 ) DOUBLE PRECISION ( <i>dlasq3</i> の場合 ) <i>d(n0)</i> を除く <i>d</i> の最小値。最初の反復では 0 であり、変更されてはいけない。

<i>dmin2</i>	REAL (slasq3 の場合 ) DOUBLE PRECISION (dlasq3 の場合 ) <i>d</i> ( <i>n</i> 0) と <i>d</i> ( <i>n</i> 0-1) を除く <i>d</i> の最小値。最初の反復では 0 であり、変更されてはいけない。
<i>dn</i>	REAL (slasq3 の場合 ) DOUBLE PRECISION (dlasq3 の場合 ) <i>d</i> ( <i>n</i> ) が格納される。最初の反復では 0 であり、変更されてはいけない。
<i>dn1</i>	REAL (slasq3 の場合 ) DOUBLE PRECISION (dlasq3 の場合 ) <i>d</i> ( <i>n</i> -1) が格納される。最初の反復では 0 であり、変更されてはいけない。
<i>dn2</i>	REAL (slasq3 の場合 ) DOUBLE PRECISION (dlasq3 の場合 ) <i>d</i> ( <i>n</i> -2) が格納される。最初の反復では 0 であり、変更されてはいけない。

## 出力パラメータ

<i>dmin</i>	REAL (slasq3 の場合 ) DOUBLE PRECISION (dlasq3 の場合 ) <i>d</i> の最小値。
<i>sigma</i>	REAL (slasq3 の場合 ) DOUBLE PRECISION (dlasq3 の場合 ) 現在のセグメントで使用されるシフトの合計。
<i>desig</i>	<i>sigma</i> の下位次数部分。
<i>nfail</i>	INTEGER。シフト回数が大きすぎたことを示す。
<i>iter</i>	INTEGER。反復の回数。
<i>ndiv</i>	INTEGER。除算の回数。
<i>ttype</i>	INTEGER。シフトのタイプ。
<i>dmin1</i>	<i>d</i> ( <i>n</i> 0) を除く <i>d</i> の最小値。
<i>dmin2</i>	<i>d</i> ( <i>n</i> 0) と <i>d</i> ( <i>n</i> 0-1) を除く <i>d</i> の最小値。
<i>dn</i>	<i>d</i> ( <i>n</i> )。
<i>dn1</i>	<i>d</i> ( <i>n</i> -1)。
<i>dn2</i>	<i>d</i> ( <i>n</i> -2)。



`tau` REAL (slasq3 の場合)  
DOUBLE PRECISION (dlasq3 の場合)  
シフト。

## ?lasq4

以前の変換で得られた  $d$  の値を用いて最小固有値  
に対する近似を計算する。  
?bdsqr で使用される。

### 構文

```
call slasq4( i0, n0, z, pp, n0in, dmin, dmin1, dmin2, dn, dn1, dn2, tau,
            ttype, g )
call dlasq4( i0, n0, z, pp, n0in, dmin, dmin1, dmin2, dn, dn1, dn2, tau,
            ttype, g )
```

### 説明

このルーチンは、以前の変換で得られた  $d$  の値を用いて、最小固有値に対する近似  $tau$  を計算する。

### 入力パラメータ

`i0` INTEGER。最初のインデックス。  
`n0` INTEGER。最後のインデックス。  
`z` REAL (slasq4 の場合)  
DOUBLE PRECISION (dlasq4 の場合)  
配列、次元は  $(4n)$ 。 $z$  には  $qd$  配列を格納する。  
`pp` INTEGER。ping のとき  $pp = 0$ 、pong のとき  $pp = 1$   
`noin` INTEGER。eigtest の開始にある  $n0$  の値。  
`dmin` REAL (slasq4 の場合)  
DOUBLE PRECISION (dlasq4 の場合)  
 $d$  の最小値。  
`dmin1` REAL (slasq4 の場合)  
DOUBLE PRECISION (dlasq4 の場合)  
 $d(n0)$  を除く  $d$  の最小値。

<i>dmin2</i>	REAL (slasq4 の場合 ) DOUBLE PRECISION (dlasq4 の場合 ) <i>d</i> ( <i>n0</i> ) と <i>d</i> ( <i>n0</i> -1) を除く <i>d</i> の最小値。
<i>dn</i>	REAL (slasq4 の場合 ) DOUBLE PRECISION (dlasq4 の場合 ) <i>d</i> ( <i>n</i> ) が格納される。
<i>dn1</i>	REAL (slasq4 の場合 ) DOUBLE PRECISION (dlasq4 の場合 ) <i>d</i> ( <i>n</i> -1) が格納される。
<i>dn2</i>	REAL (slasq4 の場合 ) DOUBLE PRECISION (dlasq4 の場合 ) <i>d</i> ( <i>n</i> -2) が格納される。
<i>g</i>	REAL (slasq4 の場合 ) DOUBLE PRECISION (dlasq4 の場合 ) シフト係数、入力では 0 である。

### 出力パラメータ

<i>tau</i>	REAL (slasq4 の場合 ) DOUBLE PRECISION (dlasq4 の場合 ) シフト。
<i>ttype</i>	INTEGER。シフトのタイプ。

---

## ?lasq5

*ping-pong* 形式で *dqds* 変換を 1 つ計算する。  
*?bdsqr* と *?stegr* で使用される。

---

### 構文

```
call slasq5(i0, n0, z, pp, tau, dmin, dmin1, dmin2, dn, dnm1, dnm2, ieee)  
call dlasq5(i0, n0, z, pp, tau, dmin, dmin1, dmin2, dn, dnm1, dnm2, ieee)
```

### 説明

このルーチンは *ping-pong* 形式で *dqds* 変換を 1 つ計算する。IEEE マシンと非 IEEE マシンに対応する。

## 入力パラメータ

<i>i0</i>	INTEGER。最初のインデックス。
<i>n0</i>	INTEGER。最後のインデックス。
<i>z</i>	REAL (slasq5 の場合 ) DOUBLE PRECISION (dlasq5 の場合 ) 配列、次元は (4 <i>n</i> )。 <i>z</i> には <i>qd</i> 配列を格納する。余分な引数を防ぐため <i>z</i> (4* <i>n0</i> ) に <i>emin</i> を格納する。
<i>pp</i>	INTEGER。 ping のとき <i>pp</i> = 0、pong のとき <i>pp</i> = 1
<i>tau</i>	REAL (slasq5 の場合 ) DOUBLE PRECISION (dlasq5 の場合 ) シフトである。
<i>ieee</i>	LOGICAL。 IEEE または非 IEEE 演算を示すフラグ。

## 出力パラメータ

<i>dmin</i>	REAL (slasq5 の場合 ) DOUBLE PRECISION (dlasq5 の場合 ) <i>d</i> の最小値。
<i>dmin1</i>	REAL (slasq5 の場合 ) DOUBLE PRECISION (dlasq5 の場合 ) <i>d</i> ( <i>n0</i> ) を除く <i>d</i> の最小値。
<i>dmin2</i>	REAL (slasq5 の場合 ) DOUBLE PRECISION (dlasq5 の場合 ) <i>d</i> ( <i>n0</i> ) と <i>d</i> ( <i>n0</i> -1) を除く <i>d</i> の最小値。
<i>dn</i>	REAL (slasq5 の場合 ) DOUBLE PRECISION (dlasq5 の場合 ) <i>d</i> の最後の値である <i>d</i> ( <i>n0</i> ) が格納される。
<i>dnm1</i>	REAL (slasq5 の場合 ) DOUBLE PRECISION (dlasq5 の場合 ) <i>d</i> ( <i>n0</i> -1) が格納される。
<i>dnm2</i>	REAL (slasq5 の場合 ) DOUBLE PRECISION (dlasq5 の場合 ) <i>d</i> ( <i>n0</i> -2) が格納される。

## ?lasq6

ping-pong 形式で *dqd* 変換を 1 つ計算する。  
?bdsqr と ?stegr で使用される。

---

### 構文

```
call slasq6( i0, n0, z, pp, dmin, dmin1, dmin2, dn, dnm1, dnm2 )  
call dlasq6( i0, n0, z, pp, dmin, dmin1, dmin2, dn, dnm1, dnm2 )
```

### 説明

ルーチン ?lasq6 は、アンダーフローとオーバーフローに対する保護を行いながら、ping-pong 形式で *dqd* (ゼロに等しいシフト) 変換を 1 つ計算する。

### 入力パラメータ

<i>i0</i>	INTEGER。最初のインデックス。
<i>n0</i>	INTEGER。最後のインデックス。
<i>z</i>	REAL (slasq6 の場合 ) DOUBLE PRECISION (dlasq6 の場合 ) 配列、次元は (4 <i>n</i> )。 <i>z</i> には <i>qd</i> 配列を格納する。余分な引数を防ぐため <i>z</i> (4* <i>n0</i> ) に <i>emin</i> を格納する。
<i>pp</i>	INTEGER。 ping のとき <i>pp</i> = 0、pong のとき <i>pp</i> = 1

### 出力パラメータ

<i>dmin</i>	REAL (slasq6 の場合 ) DOUBLE PRECISION (dlasq6 の場合 ) <i>d</i> の最小値。
<i>dmin1</i>	REAL (slasq6 の場合 ) DOUBLE PRECISION (dlasq6 の場合 ) <i>d</i> ( <i>n0</i> ) を除く <i>d</i> の最小値。
<i>dmin2</i>	REAL (slasq6 の場合 ) DOUBLE PRECISION (dlasq6 の場合 ) <i>d</i> ( <i>n0</i> ) と <i>d</i> ( <i>n0</i> -1) を除く <i>d</i> の最小値。

<i>dn</i>	REAL (slasq6 の場合 ) DOUBLE PRECISION (dlasq6 の場合 ) <i>d</i> の最後の値である <i>d</i> ( <i>n0</i> ) が格納される。
<i>dnm1</i>	REAL (slasq6 の場合 ) DOUBLE PRECISION (dlasq6 の場合 ) <i>d</i> ( <i>n0-1</i> ) が格納される。
<i>dnm2</i>	REAL (slasq6 の場合 ) DOUBLE PRECISION (dlasq6 の場合 ) <i>d</i> ( <i>n0-2</i> ) が格納される。

## ?lasr

一般矩形行列に一連の面回転を適用する。

### 構文

```
call slasr( side, pivot, direct, m, n, c, s, a, lda )
call dlasr( side, pivot, direct, m, n, c, s, a, lda )
call clasr( side, pivot, direct, m, n, c, s, a, lda )
call zlasr( side, pivot, direct, m, n, c, s, a, lda )
```

### 説明

このルーチンは次の変換を実行する。

*side* = 'L' または 'l' のとき  $A := P A$  ( 左辺側 )

*side* = 'R' または 'r' のとき  $A := A P$  ( 右辺側 )

*A* は  $m \times n$  の実数行列、*P* は直交行列で、パラメータ *pivot* と *direct* によって次のように定義される一連の面回転で構成される (*side* = 'L' または 'l' のとき  $z = m$ 、*side* = 'R' または 'r' のとき  $z = n$ )。

*direct* = 'F' または 'f' のとき ( 順方向順 )、 $P = P(z-1) \dots P(2) P(1)$ 、

*direct* = 'B' または 'b' のとき ( 逆方向順 )、 $P = P(1) P(2) \dots P(z-1)$ 、

*P*(*k*) は次の面に対する面回転行列である。

*pivot* = 'V' または 'v' ( 可変ピボット ) のとき、面 (*k*, *k*+1)

*pivot* = 'T' または 't' ( 上ピボット ) のとき、面 (1, *k*+1)

*pivot* = 'B' または 'b' ( 下ピボット ) のとき、面 (*k*, *z*)

$c(k)$  と  $s(k)$  には  $P(k)$  を定義する余弦と正弦が格納されていなければならない。行列  $P(k)$  の  $2 \times 2$  面回転部分  $R(k)$  は、次の形式とみなす。

$$R(k) = \begin{bmatrix} c(k) & s(k) \\ -s(k) & c(k) \end{bmatrix}$$

## 入力パラメータ

<i>side</i>	CHARACTER*1。面回転行列 $P$ を $A$ の左または右に適用するかを指定する。 = 'L' の場合、左、 $A := PA$ を計算する。 = 'R' の場合、右、 $A := AP$ を計算する。
<i>direct</i>	CHARACTER*1。 $P$ が面回転の順方向順または逆方向順かを指定する。 = 'F' の場合、順方向、 $P = P(z-1) \dots P(2) P(1)$ = 'B' の場合、逆方向、 $P = P(1) P(2) \dots P(z-1)$
<i>pivot</i>	CHARACTER*1。面回転行列 $P(k)$ の面を指定する。 = 'V' の場合、可変ピボット、面 $(k, k+1)$ = 'T' の場合、上ピボット、面 $(1, k+1)$ = 'B' の場合、下ピボット、面 $(k, z)$
<i>m</i>	INTEGER。行列 $A$ の行数。 $m \leq 1$ の場合、イミディエイト・リターンが有効。
<i>n</i>	INTEGER。行列 $A$ の列数。 $n \leq 1$ の場合、イミディエイト・リターンが有効。
<i>c, s</i>	REAL (slasr/clasr の場合) DOUBLE PRECISION (dlasr/zlasr の場合) 配列、次元は、 <i>side</i> = 'L' の場合 $(m-1)$ <i>side</i> = 'R' の場合 $(n-1)$ $c(k)$ と $s(k)$ には上述のように行列 $P(k)$ を定義する余弦と正弦を格納する。
<i>a</i>	REAL (slasr の場合) DOUBLE PRECISION (dlasr の場合) COMPLEX (clasr の場合) COMPLEX*16 (zlasr の場合) 配列、次元は $(lda, n)$ 。 $m \times n$ の行列 $A$
<i>lda</i>	INTEGER。配列 $A$ のリーディング・ディメンジョン。 $lda \geq \max(1, m)$

### 出力パラメータ

*a* *side* = 'R' の場合は *pa*、*side* = 'I' の場合は *AP* で上書きされる。

## ?lasrt

数を昇順または降順でソートする。

### 構文

```
call slasrt( id, n, d, info )
```

```
call dlasrt( id, n, d, info )
```

### 説明

?lasrt は、*d* に格納されている数を昇順 (*id* = 'I' の場合) または降順 (*id* = 'D' の場合) でソートする。このルーチンはクイックソートを使用するが、*size* ≤ 20 の配列では挿入ソートが使われる。スタックの次元によって *n* はおよそ  $2^{32}$  に制限される。

### 入力パラメータ

*id* CHARACTER\*1。  
= 'I' の場合、*d* を昇順でソートする。  
= 'D' の場合、*d* を降順でソートする。

*n* INTEGER。配列 *d* の長さ。

*d* REAL (slasrt の場合)  
DOUBLE PRECISION (dlasrt の場合)  
ソートする配列を格納する。

### 出力パラメータ

*d* *id* の設定によって、*d* は昇順 ( $d(1) \leq \dots \leq d(n)$ ) または降順 ( $d(1) \geq \dots \geq d(n)$ ) となる。

*info* INTEGER。  
< 0 の場合、*info* = -*i* は *i* 番目の引数の値が不正だったことを示す。

## ?lassq

スケーリング形式で表現された二乗和を更新する。

---

### 構文

```
call slassq( n, x, incx, scale, sumsq )
call dlassq( n, x, incx, scale, sumsq )
call classq( n, x, incx, scale, sumsq )
call zlassq( n, x, incx, scale, sumsq )
```

### 説明

実数ルーチン `slassq/dlassq` は、次のように値 `scl` と `sumsq` を返す。

$$scl^2 * sumsq = x(1)^2 + \dots + x(n)^2 + scale^2 * sumsq$$

ここで  $x(i) = x(1 + (i - 1) incx)$ 。

`sumsq` の値は非負であるとし、`scl` は次の値を返す。

$$scl = \max( scale, \text{abs}(x(i)) )$$

`scale` と `sumsq` は `scale` と `sumsq` によって与えられなければならない。また `scl` と `sumsq` は `scale` と `sumsq` にそれぞれ上書きされる。

複素ルーチン `classq/zlassq` は、次のように値 `scl` と `ssq` を返す。

$$scl^2 * ssq = x(1)^2 + \dots + x(n)^2 + scale^2 * sumsq$$

ここで  $x(i) = \text{abs}(x(1 + (i - 1) incx))$ 。

`sumsq` の値は少なくともユニティであるとみなされ、`ssq` の値は、次のように満たされる。  $1.0 \leq ssq \leq sumsq + 2n$

`scale` の値は非負であるとし、`scl` は次の値を返す。

$$scl = \max_i( scale, \text{abs}(\text{real}(x(i))), \text{abs}(\text{aimag}(x(i))) )$$

`scale` と `sumsq` は `scale` と `sumsq` によって与えられなければならない。また `scl` と `ssq` は `scale` と `sumsq` にそれぞれ上書きされる。

?lassq の全ルーチンは唯一、ベクトル `x` を出力に渡す。



### 入力パラメータ

<i>n</i>	INTEGER。ベクトル <i>x</i> で使用する成分の個数。
<i>x</i>	REAL (slassq の場合 ) DOUBLE PRECISION (dlassq の場合 ) COMPLEX (classq の場合 ) COMPLEX*16 (zlassq の場合 ) スケーリングされた二乗和を計算するベクトル。 $x(i) = x(1 + (i - 1) incx), 1 \leq i \leq n$
<i>incx</i>	INTEGER。ベクトル <i>x</i> の連続する値の増分。 $incx > 0$
<i>scale</i>	REAL (slassq/classq の場合 ) DOUBLE PRECISION (dlassq/zlassq の場合 ) 上の式で示した <i>scale</i> の値を格納する。
<i>sumsq</i>	REAL (slassq/classq の場合 ) DOUBLE PRECISION (dlassq/zlassq の場合 ) 上の式で示した <i>sumsq</i> の値を格納する。

### 出力パラメータ

<i>scale</i>	<i>scale</i> は、二乗和に対するスケール係数 <i>scl</i> で上書きされる。
<i>sumsq</i>	実数型の場合。 <i>sumsq</i> は上の式で示した <i>sumsq</i> の値で上書きされる。 複素型の場合。 <i>sumsq</i> は上の式で示した <i>ssq</i> の値で上書きされる。

---

## ?lasv2

2 × 2 三角行列の特異値分解を計算する。

---

### 構文

```
call slasv2( f, g, h, ssmin, ssmax, snr, csr, snl, csl )
call dlasv2( f, g, h, ssmin, ssmax, snr, csr, snl, csl )
```

## 説明

ルーチン `?lasv2` は  $2 \times 2$  三角行列の特異値分解を計算する。

$$\begin{bmatrix} f & g \\ 0 & h \end{bmatrix}$$

出力において、`abs(ssmax)` には大きいほうの特異値が、`abs(ssmin)` には小さいほうの特異値が、`(csl, snl)` と `(csr, snr)` には `abs(ssmax)` に対する左と右の特異ベクトルが格納され、分解を与える。

$$\begin{bmatrix} csl & snl \\ -snl & csl \end{bmatrix} \begin{bmatrix} f & g \\ 0 & h \end{bmatrix} \begin{bmatrix} csr & -snr \\ snr & csr \end{bmatrix} = \begin{bmatrix} ssmax & 0 \\ 0 & ssmin \end{bmatrix}$$

## 入力パラメータ

`f, g, h`            REAL (`slasv2` の場合)  
                      DOUBLE PRECISION (`dlasv2` の場合)  
                      それぞれ、 $2 \times 2$  行列の (1,1)、(1,2)、(2,2) の成分。

## 出力パラメータ

`ssmin, ssmax`    REAL (`slasv2` の場合)  
                      DOUBLE PRECISION (`dlasv2` の場合)  
                      それぞれ、`abs(ssmin)` と `abs(ssmax)` は小さいほうの固有値および大きいほうの特異値。

`snl, csl`        REAL (`slasv2` の場合)  
                      DOUBLE PRECISION (`dlasv2` の場合)  
                      ベクトル (`csl, snl`) は特異値 `abs(ssmax)` に対する単位左特異ベクトル。

`snr, csr`        REAL (`slasv2` の場合)  
                      DOUBLE PRECISION (`dlasv2` の場合)  
                      ベクトル (`csr, sn`) は特異値 `abs(ssmax)` に対する単位右特異ベクトル。

## アプリケーション・ノート

入力パラメータは、出力パラメータにエイリアスされることがある。

オーバーフロー / アンダーフローの場合を除いて、減算にガード桁がなくとも、すべての出力の大きさは最後の位置からわずかな単位以内 (ulp) で正確である。

IEEE 演算で、1 つの行列成分が無限でもコードは正しく動作する。  
最大特異値自身がオーバーフローしない限り、あるいは最大特異値がオーバーフローからわずかな ulp 範囲内にある限り、オーバーフローは起こらない。(Cray のように部分オーバーフローを持つマシンでは、最大特異値がオーバーフローに対して 2 の因子以内にあるとオーバーフローが起こることがある。)  
アンダーフローは緩やかであればアンダーフローは無害である。そうでない場合、結果は、アンダーフローしきい値に近いサイズの摂動によって変更された行列に一致することがある。

---

## ?laswp

一般矩形行列に対して一連の行交換を実行する。

### 構文

```
call slaswp( n, a, lda, k1, k2, ipiv, incx )
call dlaswp( n, a, lda, k1, k2, ipiv, incx )
call claswp( n, a, lda, k1, k2, ipiv, incx )
call zlaswp( n, a, lda, k1, k2, ipiv, incx )
```

### 説明

このルーチンは行列 A に一連の行交換を実行する。行交換は a の行 k1 から k2 のそれぞれに対して開始される、

### 入力パラメータ

n	INTEGER。行列 A の列数。
a	REAL (slaswp の場合 ) DOUBLE PRECISION (dlaswp の場合 ) COMPLEX (claswp の場合 )

	COMPLEX*16 (zlaswp の場合) 配列、次元は (lda, n)。 行交換を適用する列次元 n の行列を格納する。
lda	INTEGER。配列 a のリーディング・ディメンジョン。
k1	INTEGER。ipiv の行交換を適用する最初の成分。
k2	INTEGER。ipiv の行交換を適用する最後の成分。
ipiv	INTEGER。 配列、次元は (m* abs(incx)) ピボットのインデックスのベクトル。ipiv の位置 k1 から k2 にある成分のみがアクセスされる。 ipiv(k) = l には行 k と l が交換される意味がある。
incx	INTEGER。ipiv の連続する値の増分。ipiv が負の場合、ピボットは逆順に適用される。

### 出力パラメータ

a	置換された行列で上書きされる。
---	-----------------

---

## ?lasy2

行列の次数が 1 または 2 のシルベスター行列式を解く。

---

### 構文

```
call slasy2( ltranl, ltranr, isgn, n1, n2, t1, ldt1, tr, ldtr, b, ldb, scale, x,  
            ldx, xnorm, info )  
call dlasy2( ltranl, ltranr, isgn, n1, n2, t1, ldt1, tr, ldtr, b, ldb, scale, x,  
            ldx, xnorm, info )
```

### 説明

このルーチンは、次式で  $n1 \times n2$  行列  $X$  を解く。  $1 \leq n1$ 、 $n2 \leq 2$ 、

$$\text{op}(TL) * X + \text{isgn} * X * \text{op}(TR) = \text{scale} * B$$

ここで

$TL$  は  $n1 \times n1$ 、

$TR$  は  $n2 \times n2$ 、

$B$  は  $n1 \times n2$ 、

また、 $isgn = 1$  または  $-1$ 、 $op(T) = T$  または  $T'$ 、ここで  $T'$  は  $T$  の転置を表す。

## 入力パラメータ

<i>ltranl</i>	LOGICAL。 <i>ltranl</i> は $op(TL)$ を指定する。 = .FALSE. の場合、 $op(TL) = TL$ 、 = .TRUE. の場合、 $op(TL) = TL'$ 。
<i>ltrandr</i>	LOGICAL。 <i>ltrandr</i> は $op(TR)$ を指定する。 = .FALSE. の場合 $op(TR) = TR$ 、 = .TRUE. の場合 $op(TR) = TR'$ 。
<i>isgn</i>	INTEGER。 <i>isgn</i> は前述のとおり式の符号を指定する。 <i>isgn</i> は 1 または -1 を取り得る。
<i>n1</i>	INTEGER。 <i>n1</i> は行列 $TL$ の次数を指定する。 <i>n1</i> は、0、1、2 を取り得る。
<i>n2</i>	INTEGER。 <i>n2</i> は行列 $TR$ の次数を指定する。 <i>n2</i> は、0、1、2 を取り得る。
<i>t1</i>	REAL (slasy2 の場合 ) DOUBLE PRECISION (dlasy2 の場合 ) 配列、次元は ( <i>ldt1</i> , 2)。 <i>t1</i> には $n1 \times n1$ の行列 $TL$ を格納する。
<i>ldt1</i>	INTEGER。 行列 <i>t1</i> のリーディング・ディメンジョン。 $ldt1 \geq \max(1, n1)$
<i>tr</i>	REAL (slasy2 の場合 ) DOUBLE PRECISION (dlasy2 の場合 ) 配列、次元は ( <i>ldtr</i> , 2)。 <i>tr</i> には $n2 \times n2$ の行列 $TR$ を格納する。
<i>ldtr</i>	INTEGER。 行列 <i>tr</i> のリーディング・ディメンジョン。 $ldtr \geq \max(1, n2)$
<i>b</i>	REAL (slasy2 の場合 ) DOUBLE PRECISION (dlasy2 の場合 ) 配列、次元は ( <i>ldb</i> , 2)。 $n1 \times n2$ の行列 <i>b</i> には式の右辺を格納する。

*ldb* INTEGER。  
行列 *b* のリーディング・ディメンジョン。  
 $ldb \geq \max(1, n1)$

*ldx* INTEGER。  
出力行列 *x* のリーディング・ディメンジョン。  
 $ldx \geq \max(1, n1)$

### 出力パラメータ

*scale* REAL (slasy2 の場合 )  
DOUBLE PRECISION (dlasy2 の場合 )  
終了時に、*scale* にはスケール係数が格納される。  
*scale* は 1 以下に選択され解のオーバーフローを防ぐ。

*x* REAL (slasy2 の場合 )  
DOUBLE PRECISION (dlasy2 の場合 )  
配列、次元は (*ldx*,2)。 *x* には  $n1 \times n2$  の解が格納される。

*xnorm* REAL (slasy2 の場合 )  
DOUBLE PRECISION (dlasy2 の場合 )  
*xnorm* には解の無限ノルムが格納される。

*info* INTEGER。 *info* には次の値が設定される。  
0 の場合、正常に終了したことを示す。  
1 の場合、*TL* と *TR* は近すぎる固有値を持つため、*TL* と *TR* は非特異方程式を得るために摂動された。



---

**注：**実行速度を考慮してこのルーチンでは入力のエラーをチェックしない。

---

---

## ?lasyf

対角ピボット演算法を使って実数/複素対称行列  
の部分因子分解を計算する。

---

### 構文

```
call slasyf( uplo, n, nb, kb, a, lda, ipiv, w, ldw, info )
```

```
call dlasyf( uplo, n, nb, kb, a, lda, ipiv, w, ldw, info )
call clasyf( uplo, n, nb, kb, a, lda, ipiv, w, ldw, info )
call zlasyf( uplo, n, nb, kb, a, lda, ipiv, w, ldw, info )
```

## 説明

ルーチン `?lasyf` は、**Bunch-Kaufman** 対角ピボット演算法を使って、実数 / 複素対称行列  $A$  の部分因子分解を計算する。部分因子分解は次の形式を持つ。

$$A = \begin{bmatrix} I & U_{12} \\ 0 & U_{22} \end{bmatrix} \begin{bmatrix} A_{11} & 0 \\ 0 & D \end{bmatrix} \begin{bmatrix} I & 0 \\ U_{12}' & U_{22}' \end{bmatrix} \quad \text{uplo='U' の場合、または}$$

$$A = \begin{bmatrix} L_{11} & 0 \\ L_{21} & I \end{bmatrix} \begin{bmatrix} D & 0 \\ 0 & A_{22} \end{bmatrix} \begin{bmatrix} L_{11}' & L_{21}' \\ 0 & I \end{bmatrix} \quad \text{uplo='L' の場合}$$

ここで  $D$  の次数は大きくとも  $nb$  である。実際の次数は引数  $kb$  で返され、 $nb$  か  $nb-1$ 、あるいは  $n \leq nb$  の場合は  $n$  である。

このルーチンは `?sytrf` から呼び出される補助ルーチンである。部分行列  $A_{11}$  (uplo='U' の場合) または  $A_{22}$  (uplo='L' の場合) を更新するためにブロック化コード (レベル 3 BLAS の呼び出し) が使用されている。

## 入力パラメータ

<code>uplo</code>	CHARACTER*1。 対称行列 $A$ の上三角部分または下三角部分のどちらが格納されているかを指定する。 = 'U' の場合、上三角 = 'L' の場合、下三角
<code>n</code>	INTEGER。 行列 $A$ の次数。 $n \geq 0$
<code>nb</code>	INTEGER。 因子分解すべき行列 $A$ の最大列。 $2 \times 2$ ピボットブロックを許すために、 $nb$ は小さくとも 2 でなければならない。

<i>a</i>	REAL (slasyf の場合 ) DOUBLE PRECISION (dlasyf の場合 ) COMPLEX (clasyf の場合 ) COMPLEX*16 (zlasyf の場合 ) 配列、次元は ( <i>lda</i> , <i>n</i> )。対称行列 <i>A</i> を格納する。uplo='U' の場合、 <i>a</i> の先頭の $n \times n$ 上三角部分に行列 <i>A</i> の上三角部分を格納する。 <i>a</i> の厳密な下三角部分は参照されない。uplo='L' の場合、 <i>a</i> の先頭の $n \times n$ 下三角部分に行列 <i>A</i> の下三角部分を格納する。 <i>a</i> の厳密な上三角部分は参照されない。
<i>lda</i>	INTEGER。 配列 <i>a</i> のリーディング・ディメンジョン。 $lda \geq \max(1, n)$ 。
<i>w</i>	REAL (slasyf の場合 ) DOUBLE PRECISION (dlasyf の場合 ) COMPLEX (clasyf の場合 ) COMPLEX*16 (zlasyf の場合 ) ワークスペース配列、次元は ( <i>ldw</i> , <i>nb</i> )
<i>ldw</i>	INTEGER。 配列 <i>a</i> のリーディング・ディメンジョン。 $ldw \geq \max(1, n)$ 。

## 出力パラメータ

<i>kb</i>	INTEGER。 実際に因子分解された <i>A</i> の列数。 <i>kb</i> は <i>nb</i> -1 か <i>nb</i> 、あるいは $n \leq nb$ の場合は <i>n</i> である。
<i>a</i>	<i>a</i> は部分因子分解の詳細で上書きされる。
<i>ipiv</i>	INTEGER。 配列、次元は ( <i>n</i> )。交換の結果と <i>D</i> のブロック構造の各成分が格納される。 uplo='U' の場合、 <i>ipiv</i> の最後の <i>kb</i> 個の成分のみが設定される。 uplo='L' の場合、最初の <i>kb</i> 個の成分のみが設定される。  <i>ipiv</i> ( <i>k</i> ) > 0 の場合、 <i>k</i> 番目の行と列と <i>ipiv</i> ( <i>k</i> ) 番目の行と列は交換された。 <i>D</i> ( <i>k</i> , <i>k</i> ) は $1 \times 1$ の対角ブロックである。 uplo='U' かつ <i>ipiv</i> ( <i>k</i> ) = <i>ipiv</i> ( <i>k</i> -1) < 0 の場合、 <i>k</i> -1 番目の行と列と - <i>ipiv</i> ( <i>k</i> ) 番目の行と列は交換された。 <i>D</i> ( <i>k</i> -1: <i>k</i> , <i>k</i> -1: <i>k</i> ) は $2 \times 2$ の対角ブロックである。 uplo='L' かつ <i>ipiv</i> ( <i>k</i> ) = <i>ipiv</i> ( <i>k</i> +1) < 0 の場合、 <i>k</i> +1 番目の行と列と - <i>ipiv</i> ( <i>k</i> ) 番目の行と列は交換された。 <i>D</i> ( <i>k</i> : <i>k</i> +1, <i>k</i> : <i>k</i> +1) は $2 \times 2$ の対角ブロックである。



*info* INTEGER。  
 = 0 の場合、正常に終了したことを示す。  
 > 0 の場合、*info* = *k* ならば  $D(k, k)$  は完全にゼロである。因子分解は完了したが、ブロック対角行列  $D$  は完全に特異である。

## ?lahef

対角ピボット演算法を使って複素エルミート無限  
 行列の部分因子分解を計算する。

### 構文

```
call clahef( uplo, n, nb, kb, a, lda, ipiv, w, ldw, info )
call zlahef( uplo, n, nb, kb, a, lda, ipiv, w, ldw, info )
```

### 説明

ルーチン ?lahef は、Bunch-Kaufman 対角ピボット演算法を使って、複素エルミート行列  $A$  の部分因子分解を計算する。部分因子分解は次の形式を持つ。

$$A = \begin{bmatrix} I & U_{12} \\ 0 & U_{22} \end{bmatrix} \begin{bmatrix} A_{11} & 0 \\ 0 & D \end{bmatrix} \begin{bmatrix} I & 0 \\ U_{12}' & U_{22}' \end{bmatrix} \quad \text{uplo='U' の場合、または}$$

$$A = \begin{bmatrix} L_{11} & 0 \\ L_{21} & I \end{bmatrix} \begin{bmatrix} D & 0 \\ 0 & A_{22} \end{bmatrix} \begin{bmatrix} L_{11}' & L_{21}' \\ 0 & I \end{bmatrix} \quad \text{uplo='L' の場合}$$

ここで  $D$  の次数は大きくとも  $nb$  である。実際の次数は引数  $kb$  で返され、 $nb$  か  $nb-1$ 、あるいは  $n \leq nb$  の場合は  $n$  である。  
 なお、 $U'$  は  $U$  の共役転置を表わす。

このルーチンは ?hetrf から呼び出される補助ルーチンである。部分行列  $A_{11}$  (uplo='U' の場合) または  $A_{22}$  (uplo='L' の場合) を更新するためにブロック化コード (レベル 3 BLAS の呼び出し) が使用されている。

## 入力パラメータ

<code>uplo</code>	CHARACTER*1。 エルミート行列 $A$ の上三角部分と下三角部分のどちらが格納されているかを指定する。 = 'U' の場合、上三角 = 'L' の場合、下三角
<code>n</code>	INTEGER。 行列 $A$ の次数。 $n \geq 0$
<code>nb</code>	INTEGER。 因子分解すべき行列 $A$ の最大列。 $2 \times 2$ ピボットブロックを許すために、 <code>nb</code> は小さくとも 2 でなければならない。
<code>a</code>	COMPLEX (clahef の場合) COMPLEX*16 (zlahef の場合) 配列、次元は $(lda, n)$ 。 エルミート行列 $A$ を格納する。 <code>uplo</code> = 'U' の場合、 <code>a</code> の先頭の $n \times n$ 上三角部分に行列 $A$ の上三角部分を格納する。 <code>a</code> の厳密な下三角部分は参照されない。 <code>uplo</code> = 'L' の場合、 <code>a</code> の先頭の $n \times n$ 下三角部分に行列 $A$ の下三角部分を格納する。 <code>a</code> の厳密な上三角部分は参照されない。
<code>lda</code>	INTEGER。 配列 <code>a</code> のリーディング・ディメンジョン。 $lda \geq \max(1, n)$ 。
<code>w</code>	COMPLEX (clahef の場合) COMPLEX*16 (zlahef の場合) ワークスペース配列、次元は $(ldw, nb)$
<code>ldw</code>	INTEGER。 配列 <code>a</code> のリーディング・ディメンジョン。 $ldw \geq \max(1, n)$ 。

## 出力パラメータ

<code>kb</code>	INTEGER。 実際に因子分解された $A$ の列数。 <code>kb</code> は <code>nb-1</code> か <code>nb</code> 、あるいは $n \leq nb$ の場合は $n$ である。
<code>a</code>	<code>a</code> は部分因子分解の詳細で上書きされる。

*ipiv* INTEGER。  
 配列、次元は  $(n)$ 。交換の結果と  $D$  のブロック構造の各成分が格納される。  
*uplo* = 'U' の場合、*ipiv* の最後の  $kb$  個の成分のみが設定される。  
*uplo* = 'L' の場合、最初の  $kb$  個の成分のみが設定される。

*ipiv*( $k$ ) > 0 の場合、 $k$  番目の行と列と *ipiv*( $k$ ) 番目の行と列は交換された。 $D(k, k)$  は  $1 \times 1$  の対角ブロックである。  
*uplo* = 'U' かつ *ipiv*( $k$ ) = *ipiv*( $k-1$ ) < 0 の場合、 $k-1$  番目の行と列と -*ipiv*( $k$ ) 番目の行と列は交換された。 $D(k-1:k, k-1:k)$  は  $2 \times 2$  の対角ブロックである。  
*uplo* = 'L' かつ *ipiv*( $k$ ) = *ipiv*( $k+1$ ) < 0 の場合、 $k+1$  番目の行と列と -*ipiv*( $k$ ) 番目の行と列は交換された。 $D(k:k+1, k:k+1)$  は  $2 \times 2$  の対角ブロックである。

*info* INTEGER。  
 = 0 の場合、正常に終了したことを示す。  
 > 0 の場合、*info* =  $k$  ならば  $D(k, k)$  は完全にゼロである。因子分解は完了したが、ブロック対角行列  $D$  は完全に特異である。

## ?latbs

三角帯連立方程式を解く。

### 構文

```
call slatbs( uplo, trans, diag, normin, n, kd, ab, ldab, x, scale, cnorm, info )
call dlatbs( uplo, trans, diag, normin, n, kd, ab, ldab, x, scale, cnorm, info )
call clatbs( uplo, trans, diag, normin, n, kd, ab, ldab, x, scale, cnorm, info )
call zlatbs( uplo, trans, diag, normin, n, kd, ab, ldab, x, scale, cnorm, info )
```

### 説明

このルーチンは、次の三角法をオーバーフローを防ぐためにスケーリングして解く。

$AX = s b$ 、または  $A^T x = S B$ 、または  $A^H x = s b$  (複素型の場合)

$A$  は上または下三角帯行列である。 $A^T$  は  $A$  の転置を表わし、 $A^H$  は  $A$  の共役転置を表わし、 $x$  と  $b$  は  $n$  成分のベクトル、 $s$  はスケール係数で、 $x$  の成分がオーバーフローしきい値未満になるように通常は 1 以下に選択される。スケーリングされていない問題がオー

オーバーフローを起こさない場合は、レベル 2 の BLAS ルーチン `?tbsv` が呼び出される。行列  $A$  が特異 ( 任意の  $j$  に対して  $A(j, j) = 0$  ) の場合、 $s$  は 0 に設定され  $Ax = 0$  に対するゼロでない解が返される

## 入力パラメータ

<code>uplo</code>	CHARACTER*1 行列 $A$ が上三角か下三角かを指定する。 = 'U' の場合、上三角 = 'L' の場合、下三角。
<code>trans</code>	CHARACTER*1 $A$ に適用する演算を指定する。 = 'N' の場合、 $Ax = s b$ を解く ( 転置なし ) = 'T' の場合、 $A^T x = s b$ を解く ( 転置 ) = 'C' の場合、 $A^H x = s b$ を解く ( 共役転置 )
<code>diag</code>	CHARACTER*1。 行列 $A$ が単位三角かどうかを指定する。 = 'N' の場合、単位三角ではない = 'U' の場合、単位三角
<code>normin</code>	CHARACTER*1。 <code>cnorm</code> を設定したかどうかを指定する。 = 'Y' の場合、 <code>cnorm</code> には列ノルムを格納した。 = 'N' の場合、 <code>cnorm</code> は設定していない。終了時に、ノルムが計算され、 <code>cnorm</code> に格納される。
<code>n</code>	INTEGER。 行列 $A$ の次数。 $n \geq 0$
<code>kd</code>	INTEGER。 三角行列 $A$ の劣対角または優対角の個数。 $kd \geq 0$
<code>ab</code>	REAL ( <code>slatbs</code> の場合 ) DOUBLE PRECISION ( <code>dlatbs</code> の場合 ) COMPLEX ( <code>clatbs</code> の場合 ) COMPLEX*16 ( <code>zlatbs</code> の場合 ) 配列、次元は $(ldab, n)$ 。上三角または下三角帯行列 $A$ で、配列の最初の $kd+1$ 行に格納する。 $A$ の $j$ 番目の列は配列 <code>ab</code> の番目の列に次のように格納する。 $uplo = 'U'$ の場合、 $\max(1, j-kd) \leq i \leq j$ に対して $ab(kd+1+i-j, j) = A(i, j)$ $uplo = 'L'$ の場合、 $j \leq i \leq \min(n, j+kd)$ に対して $ab(1+i-j, j) = A(i, j)$

<i>ldab</i>	INTEGER。 配列 <i>ab</i> のリーディング・ディメンジョン。 $ldab \geq kd+1$ 。
<i>x</i>	REAL (slatbs の場合 ) DOUBLE PRECISION (dlatbs の場合 ) COMPLEX (clatbs の場合 ) COMPLEX*16 (zlatbs の場合 ) 配列、次元は ( <i>n</i> )。 三角法の右辺 <i>b</i> を格納する。
<i>cnorm</i>	REAL (slatbs/clatbs の場合 ) DOUBLE PRECISION (dlatbs/zlatbs の場合 ) 配列、次元は ( <i>n</i> )。 <i>normin</i> = 'Y' の場合、 <i>cnorm</i> は入力引数となり <i>cnorm</i> ( <i>j</i> ) には <i>A</i> の <i>j</i> 番 目列の非対角部分を格納する。 <i>trans</i> = 'N' の場合、 <i>cnorm</i> ( <i>j</i> ) は無限ノ ルムに等しいか大きくなければならない。 <i>trans</i> = 'T' または 'C' の場 合、 <i>cnorm</i> ( <i>j</i> ) は 1- ノルムに等しいか大きくなければならない。

### 出力パラメータ

<i>scale</i>	REAL (slatbs/clatbs の場合 ) DOUBLE PRECISION (dlatbs/zlatbs の場合 ) 上述のとおり三角法に対するスケール係数 <i>s</i> 。 <i>scale</i> = 0 は行列 <i>A</i> が特異であるか不適切にスケーリングされたこと を示し、ベクトル <i>x</i> は $Ax = 0$ に対する完全または近似解となる。
<i>cnorm</i>	<i>normin</i> = 'N' の場合、 <i>cnorm</i> は出力引数となり <i>cnorm</i> ( <i>j</i> ) は <i>A</i> の <i>J</i> 番目 列の非対角部分の 1- ノルムを返す。
<i>info</i>	INTEGER。 < 0 の場合、 <i>info</i> = - <i>k</i> は <i>k</i> 番目の引数の値が不正だったことを示す。

## ?latdf

?getc2 による  $n \times n$  行列の  $LU$  因子分解を使い、  
 $Dif$  推定値の逆数に対する影響を計算する。

### 構文

```
call slatdf( ijob, n, z, ldz, rhs, rdsum, rdscal, ipiv, jpiv )
call dlatdf( ijob, n, z, ldz, rhs, rdsum, rdscal, ipiv, jpiv )
call clatdf( ijob, n, z, ldz, rhs, rdsum, rdscal, ipiv, jpiv )
call zlatdf( ijob, n, z, ldz, rhs, rdsum, rdscal, ipiv, jpiv )
```

### 説明

ルーチン ?latdf は、[?getc2](#) で計算された  $n \times n$  行列  $Z$  の  $LU$  因子分解を使い、 $x$  に対して  $Zx = b$  を解き、さらに  $x$  のノルムが可能な限り大きくなるような右辺  $b$  を選択して、 $Dif$  推定値の逆数に対する影響を計算する。入力において、 $rhs = b$  にはこれまでに部分式から解いた影響を格納し、出力において  $rhs = x$  となる。

?getc2 で返される  $Z$  の因子分解は形式  $Z = PLUQ$  を持っており、 $P$  と  $Q$  は置換行列である。 $L$  は単位対角成分を持つ下三角、 $U$  は上三角である。

### 入力パラメータ

<i>ijob</i>	INTEGER。 $ijob = 2$ の場合、最初に ?gecon を使って $Z$ のおおよそのノルムベクトル $e$ を計算し、続いて $e$ は正規化され、最後に 2- ノルム ( $x$ ) に大きな値を与える符号を使って $Zx = \pm e - f$ を解く。このオプションはデフォルトよりも 5 倍程度負荷が高い。 $ijob \neq 2$ (デフォルト) の場合、右辺 $b$ の全成分が +1 または -1 のいずれかとして選択される局所的先読み法。
<i>n</i>	INTEGER。 行列 $Z$ の列数。
<i>z</i>	REAL (slatdf/clatdf の場合) DOUBLE PRECISION (dlatdf/zlatdf の場合) 配列、次元は ( <i>ldz</i> , <i>n</i> ) $n \times n$ 行列 $Z$ を ?getc2 で計算した因子分解の $LU$ 部分を格納する。 $Z = PLUQ$

<i>ldz</i>	INTEGER。 配列 <i>z</i> のリーディング・ディメンジョン。 $lda \geq \max(1, n)$ 。
<i>rhs</i>	REAL (slatdf/clatdf の場合) DOUBLE PRECISION (dlatdf/zlatdf の場合) 配列、次元は ( <i>n</i> )。 <i>rhs</i> には他の部分式で得られた影響を格納する。
<i>rdsum</i>	REAL (slatdf/clatdf の場合) DOUBLE PRECISION (dlatdf/zlatdf の場合) ?tgsyl で得られた <i>Dif</i> 推定に対する、計算された影響の二乗和を格納する。ここでスケール係数 <i>rdscal</i> は因子分解されている。 <i>trans</i> = 'T' の場合、 <i>rdsum</i> は使われない。 <i>rdsum</i> は ?tgsy2 が ?tgsyl から呼び出されたときのみ意味を持つことに注意する。
<i>rdscal</i>	REAL (slatdf/clatdf の場合) DOUBLE PRECISION (dlatdf/zlatdf の場合) <i>rdsum</i> のオーバーフローを防ぐために使用されるスケール係数を格納する。 <i>trans</i> = 'T' の場合、 <i>rdscal</i> は使われない。  <i>rdscal</i> は ?tgsy2 が ?tgsyl から呼び出されたときのみ意味を持つことに注意する。
<i>ipiv</i>	INTEGER。 配列、次元は ( <i>n</i> )。 ピボットのインデックス。 $1 \leq i \leq n$ で、行列の行 <i>i</i> は行 <i>ipiv(i)</i> と交換されている。
<i>jpiv</i>	INTEGER。 配列、次元は ( <i>n</i> )。 ピボットのインデックス。 $1 \leq j \leq n$ で、行列の列 <i>j</i> は列 <i>jpiv(j)</i> と交換されている。

### 出力パラメータ

<i>rhs</i>	<i>rhs</i> には、 <i>ijob</i> の値に対応する成分で構成される、部分式の解が書きされる
<i>rdsum</i>	対応する二乗和は、現在の部分式から得られた影響によって更新される。 <i>trans</i> = 'T' の場合、 <i>rdsum</i> は使われない。
<i>rdscal</i>	<i>rdscal</i> は、 <i>rdsum</i> に格納されている現在の影響に関して更新される。 <i>trans</i> = 'T' の場合、 <i>rdsum</i> は使われない。

### ?latps

圧縮形式で格納されている行列を持った三角連立方程式を解く。

---

#### 構文

```
call slatps( uplo, trans, diag, normin, n, ap, x, scale, cnorm, info )
call dlatps( uplo, trans, diag, normin, n, ap, x, scale, cnorm, info )
call clatps( uplo, trans, diag, normin, n, ap, x, scale, cnorm, info )
call zlatps( uplo, trans, diag, normin, n, ap, x, scale, cnorm, info )
```

#### 説明

ルーチン ?latps は、次の三角法のうち 1 つをオーバーフローを防ぐためにスケーリングして解く。

$AX = s b$ 、または  $A^T x = S B$ 、または  $A^H x = s b$  (複素型の場合)

$A$  は圧縮形式で格納されている上または下三角行列である。 $A^T$  は  $A$  の転置を表わし、 $A^H$  は  $A$  の共役転置を表わし、 $x$  と  $b$  は  $n$  成分のベクトル、 $s$  はスケール係数で、 $x$  の成分がオーバーフローしきい値未満になるように通常は 1 以下に選択される。スケーリングされていない問題がオーバーフローを起こさない場合は、レベル 2 の BLAS ルーチン [?tpsv](#) が呼び出される。行列  $A$  が特異 (任意の  $j$  に対して  $A(j, j) = 0$ ) の場合、 $s$  は 0 に設定され  $Ax = 0$  に対するゼロでない解が返される

#### 入力パラメータ

<code>uplo</code>	CHARACTER*1。 行列 $A$ が上三角か下三角かを指定する。 = 'U' の場合、上三角 = 'L' の場合、下三角。
<code>trans</code>	CHARACTER*1。 $A$ に適用する演算を指定する。 = 'N' の場合、 $Ax = s b$ を解く (転置なし) = 'T' の場合、 $A^T x = s b$ を解く (転置) = 'C' の場合、 $A^H x = s b$ を解く (共役転置)



<i>diag</i>	<p>CHARACTER*1。          行列 <math>A</math> が単位三角かどうかを指定する。          = 'N' の場合、単位三角ではない。          = 'U' の場合、単位三角。</p>
<i>normin</i>	<p>CHARACTER*1。  <i>cnorm</i> を設定したかどうかを指定する。          = 'Y' の場合、<i>cnorm</i> には列ノルムを格納した。          = 'N' の場合、<i>cnorm</i> は設定していない。終了時に、ノルムが計算され、<i>cnorm</i> に格納される。</p>
<i>n</i>	<p>INTEGER。          行列 <math>A</math> の次数。 <math>n \geq 0</math></p>
<i>ap</i>	<p>REAL (slatps の場合 )          DOUBLE PRECISION (dlatps の場合 )          COMPLEX (clatps の場合 )          COMPLEX*16 (zlatps の場合 )          配列、次元は <math>(n(n+1)/2)</math>。上三角または下三角帯行列 <math>A</math> で、線形配列にカラム方向に圧縮されている。<math>A</math> の <math>j</math> 番目の列は配列 <i>ap</i> に次のように格納する。  <math>uplo = 'U'</math> の場合、 <math>1 \leq i \leq j</math> に対して <math>ap(i + (j-1)j/2) = A(i, j)</math>  <math>uplo = 'L'</math> の場合、 <math>j \leq i \leq n</math> に対して <math>ap(i + (j-1)(2n-j)/2) = A(i, j)</math></p>
<i>x</i>	<p>REAL (slatps の場合 )          DOUBLE PRECISION (dlatps の場合 )          COMPLEX (clatps の場合 )          COMPLEX*16 (zlatps の場合 )          配列、次元は <math>(n)</math>。三角法の右辺 <math>b</math> を格納する。</p>
<i>cnorm</i>	<p>REAL (slatps/clatps の場合 )          DOUBLE PRECISION (dlatps/zlatps の場合 )          配列、次元は <math>(n)</math>。  <i>normin</i> = 'Y' の場合、<i>cnorm</i> は入力引数となり <i>cnorm</i>(<math>j</math>) には <math>A</math> の <math>j</math> 番目列の非対角部分を格納する。<i>trans</i> = 'N' の場合、<i>cnorm</i>(<math>j</math>) は無限ノルムに等しいか大きくなければならない。<i>trans</i> = 'T' または 'C' の場合、<i>cnorm</i>(<math>j</math>) は 1- ノルムに等しいか大きくなければならない。</p>

## 出力パラメータ

<i>x</i>	終了時に、 <i>x</i> は解のベクトル $x$ によって上書きされる。
----------	--

<i>scale</i>	REAL (slatps/clatps の場合 ) DOUBLE PRECISION (dlatps/zlatps の場合 ) 上述のとおり三角法に対するスケール係数 $s$ 。 $scale=0$ は行列 $A$ が特異であるか不適切にスケーリングされたことを示し、ベクトル $x$ は $Ax=0$ に対する完全または近似解となる。
<i>cnorm</i>	$normin='N'$ の場合、 <i>cnorm</i> は出力引数となり $cnorm(j)$ は $A$ の $J$ 番目列の非対角部分の 1- ノルムを返す。
<i>info</i>	INTEGER。 = 0 の場合、正常に終了したことを示す。 < 0 の場合、 $info=-k$ は $k$ 番目の引数の値が不正だったことを示す。

---

### ?latrd

直交/ユニタリ相似変換を用いて、対称/エルミート行列  $A$  の最初の  $NB$  列と  $nb$  行を実数三重対角形式に縮退する。

---

#### 構文

```
call slatrd( uplo, n, nb, a, lda, e, tau, w, ldw )  
call dlatrd( uplo, n, nb, a, lda, e, tau, w, ldw )  
call clatrd( uplo, n, nb, a, lda, e, tau, w, ldw )  
call zlatrd( uplo, n, nb, a, lda, e, tau, w, ldw )
```

#### 説明

ルーチン ?latrd は、実数対称または複素エルミート行列  $a$  の  $NB$  行と  $nb$  列を、直交/ユニタリ相似変換  $Q^H A Q$  によって対称/エルミート実数三重対角形式に縮退し、 $A$  の縮退されていない部分に変換を適用するために必要となる  $V$  と  $W$  を返す。

$uplo='U'$  の場合、?latrd は、上三角が与えられる行列に対して最後の  $nb$  行と  $nb$  列の縮退を実行する。 $uplo='L'$  の場合、?latrd は、下三角が与えられる行列に対して最初の  $nb$  行と  $nb$  列の縮退を実行する。

このルーチンは ?sytrd/?hetrd から呼び出される補助ルーチンである。

## 入力パラメータ

<code>uplo</code>	CHARACTER エルミート行列 $A$ の上三角部分と下三角部分のどちらが格納されているかを指定する。 = 'U' の場合、上三角 = 'L' の場合、下三角
<code>n</code>	INTEGER。 行列 $A$ の次数。
<code>nb</code>	INTEGER。 縮退する行数と列数。
<code>a</code>	REAL (slatrd の場合) DOUBLE PRECISION (dlatrd の場合) COMPLEX (clatrd の場合) COMPLEX*16 (zlatrd の場合) 配列、次元は $(lda, n)$ 。 対称 / エルミート行列 $A$ を格納する。 <code>uplo</code> = 'U' の場合、 $a$ の先頭の $n \times n$ 上三角部分に行列 $A$ の上三角部分を格納する。 $a$ の厳密な下三角部分は参照されない。 <code>uplo</code> = 'L' の場合、 $a$ の先頭の $n \times n$ 下三角部分に行列 $A$ の下三角部分を格納する。 $a$ の厳密な上三角部分は参照されない。
<code>lda</code>	INTEGER。 配列 $a$ のリーディング・ディメンジョン。 $lda \geq (1, n)$ 。
<code>ldw</code>	INTEGER。 出力配列 $w$ のリーディング・ディメンジョン。 $ldw \geq \max(1, n)$ 。

## 出力パラメータ

<code>a</code>	<code>uplo</code> = 'U' の場合、最後の $nb$ 列が三重対角形式に縮退され、 $a$ の対角成分は三重対角形式の対角成分で上書きされる。対角よりも上の成分は配列 $\tau$ とともに基本リフレクタの積として直交 / ユニタリの行列 $Q$ を表現する。 <code>uplo</code> = 'L' の場合、最初の $nb$ 列が三重対角形式に縮退され、 $a$ の対角成分は三重対角形式の対角成分で上書きされる。対角よりも下の成分は配列 $\tau$ とともに基本リフレクタの積として直交 / ユニタリの行列 $Q$ を表現する。
----------------	--

<b>e</b>	<p>REAL (slatrd/clatrd の場合 )</p> <p>DOUBLE PRECISION (dlatrd/zlatrd の場合 )</p> <p>uplo = 'U' の場合、e(n-nb:n-1) には縮退された行列の最後の nb 列の優対角成分が格納される。</p> <p>uplo = 'L' の場合、e(1:nb) には縮退された行列の最初の nb 列の劣対角成分が格納される。</p>
<b>tau</b>	<p>REAL (slatrd の場合 )</p> <p>DOUBLE PRECISION (dlatrd の場合 )</p> <p>COMPLEX (clatrd の場合 )</p> <p>COMPLEX*16 (zlatrd の場合 )</p> <p>配列、次元は (lda, n)。</p> <p>基本リフレクタのスケール係数で、uplo = 'U' の場合は tau(n-nb:n-1) に、uplo = 'L' の場合は tau(1:nb) に格納される。</p>
<b>w</b>	<p>REAL (slatrd の場合 )</p> <p>DOUBLE PRECISION (dlatrd の場合 )</p> <p>COMPLEX (clatrd の場合 )</p> <p>COMPLEX*16 (zlatrd の場合 )</p> <p>配列、次元は (lda, n)。</p> <p>A の縮退されていない部分の更新に必要な <math>n \times nb</math> 行列 W</p>

## アプリケーション・ノート

uplo = 'U' の場合、行列  $Q$  は次の基本リフレクタの積として表現される。

$$Q = H(n) H(n-1) \dots H(n-nb+1)$$

それぞれの  $H(i)$  は次のような形式を持つ。  $H(i) = I - \tau v v^T$   $\tau$  は実数 / 複素スカラ、 $v$  は実数 / 複素ベクトルで  $v(i:n) = 0$  と  $v(i-1) = 1$ 。出力において、 $v(1:i-1)$  は  $a(1:i-1, i)$  に格納され、 $\tau$  は  $\tau(i-1)$  に格納される。

uplo = 'L' の場合、行列  $Q$  は次の基本リフレクタの積として表現される。

$$Q = H(1) H(2) \dots H(nb)$$

それぞれの  $H(i)$  は次のような形式を持つ。

$$H(i) = I - \tau v v^T$$

$\tau$  は実数 / 複素スカラ、 $v$  は実数 / 複素ベクトルで  $v(1:i) = 0$  と  $v(i+1) = 1$ 。出力において、 $v(i+1:n)$  は  $a(i+1:n, i)$  に格納され、 $\tau$  は  $\tau(i)$  に格納される。

ベクトル  $v$  の成分は、行列の縮退されていない部分に変換を適用するために、行列  $W$  とともに必要となる  $n \times nb$  の行列  $V$  を、形式の対称 / エルミート階数 -2k 更新を使用して形成する。  $A := A - V W^T - W V^T$ 。

$a$  の出力の例を、 $n=5$ 、 $nb=2$  で以下に示す。

$uplo = 'U'$  の場合       $uplo = 'L'$  の場合

$$\begin{bmatrix} a & a & a & v_4 & v_5 \\ & a & a & v_4 & v_5 \\ & & a & 1 & v_5 \\ & & & d & 1 \\ & & & & d \end{bmatrix} \qquad \begin{bmatrix} d \\ 1 & d \\ v_1 & 1 & a \\ v_1 & v_2 & a & a \\ v_1 & v_2 & a & a & a \end{bmatrix}$$

$d$  は縮退された行列の対角成分、 $a$  は変更されていない元の行列の成分、 $v_i$  は  $H(i)$  を定義するベクトルの成分を表わす。

## ?latrs

オーバーフローを防ぐために設定された  
スケール係数を持つ三角連立方程式を  
解く。

### 構文

```
call slatrs( uplo, trans, diag, normin, n, a, lda, x, scale, cnorm, info )
call dlatrs( uplo, trans, diag, normin, n, a, lda, x, scale, cnorm, info )
call clatrs( uplo, trans, diag, normin, n, a, lda, x, scale, cnorm, info )
call zlatrs( uplo, trans, diag, normin, n, a, lda, x, scale, cnorm, info )
```

### 説明

このルーチンは、次の三角法をオーバーフローを防ぐためにスケーリングして解く。

$AX = s b$ 、または  $A^T x = S B$ 、または  $A^H x = s b$  (複素型の場合)

$A$  は上または下三角行列である。 $A^T$  は  $A$  の転置を表わし、 $A^H$  は  $A$  の共役転置を表わし、 $x$  と  $b$  は  $n$  成分のベクトル、 $s$  はスケール係数で、 $x$  の成分がオーバーフローしきい値未満になるように通常は 1 以下に選択される。スケーリングされていない問題がオーバー

フローを起こさない場合は、レベル 2 の BLAS ルーチン `?trsv` が呼び出される。行列  $A$  が特異 (任意の  $j$  に対して  $A(j, j) = 0$ ) の場合、 $s$  は 0 に設定され  $Ax = 0$  に対するゼロでない解が返される

## 入力パラメータ

<code>uplo</code>	CHARACTER*1。 行列 $A$ が上三角か下三角かを指定する。 = 'U' の場合、上三角 = 'L' の場合、下三角。
<code>trans</code>	CHARACTER*1。 $A$ に適用する演算を指定する。 = 'N' の場合、 $Ax = s b$ を解く (転置なし) = 'T' の場合、 $A^T x = s b$ を解く (転置) = 'C' の場合、 $A^H x = s b$ を解く (共役転置)
<code>diag</code>	CHARACTER*1。 行列 $A$ が単位三角かどうかを指定する。 = 'N' の場合、単位三角ではない。 = 'U' の場合、単位三角。
<code>normin</code>	CHARACTER*1。 <code>cnorm</code> を設定したかどうかを指定する。 = 'Y' の場合、 <code>cnorm</code> には列ノルムが格納される。= 'N' の場合、 <code>cnorm</code> は設定していない。終了時に、ノルムが計算され、 <code>cnorm</code> に格納される。
<code>n</code>	INTEGER。 行列 $A$ の次数。 $n \geq 0$
<code>a</code>	REAL (slatrs の場合) DOUBLE PRECISION (dlatrs の場合) COMPLEX (clatrs の場合) COMPLEX*16 (zlatrs の場合) 配列、次元は $(lda, n)$ 。三角行列 $A$ を格納する。 <code>uplo</code> = 'U' の場合、配列 $a$ の先頭の $n \times n$ 上三角部分に上三角行列を格納する。 $a$ の厳密な下三角部分は参照されない。 <code>uplo</code> = 'L' の場合、配列 $a$ の先頭の $n \times n$ 下三角部分に下三角行列を格納する。 $a$ の厳密な上三角部分は参照されない。 <code>diag</code> = 'U' の場合も対角成分は参照されず 1 とみなされる。
<code>lda</code>	INTEGER。 配列 $a$ のリーディング・ディメンジョン。 $lda \geq \max(1, n)$ 。

$x$	REAL (slatrs の場合 ) DOUBLE PRECISION (dlatrs の場合 ) COMPLEX (clatrs の場合 ) COMPLEX*16 (zlatrs の場合 ) 配列、次元は $(n)$ 。三角法の右边 $b$ を格納する。
$cnorm$	REAL (slatrs/clatrs の場合 ) DOUBLE PRECISION (dlatrs/zlatrs の場合 ) 配列、次元は $(n)$ 。 $normin = 'y'$ の場合、 $cnorm$ は入力引数となり $cnorm(j)$ には $A$ の $J$ 番目列の非対角部分を格納する。 $trans = 'n'$ の場合、 $cnorm(j)$ は無限ノルムに等しいか大きくなければならない。 $trans = 't'$ または $'c'$ の場合、 $cnorm(j)$ は 1- ノルムに等しいか大きくなければならない。

### 出力パラメータ

$x$	終了時に、 $x$ は解のベクトル $x$ によって上書きされる。
$scale$	REAL (slatrs/clatrs の場合 ) DOUBLE PRECISION (dlatrs/zlatrs の場合 ) 配列、次元は $(lda, n)$ 。 上述のとおり三角法に対するスケール係数 $s$ 。 $scale = 0$ は行列 $A$ が特異であるか不適切にスケールされたことを示し、ベクトル $x$ は $Ax = 0$ に対する完全または近似解となる。
$cnorm$	$normin = 'n'$ の場合、 $cnorm$ は出力引数となり $cnorm(j)$ は $A$ の $j$ 番目列の非対角部分の 1- ノルムを返す。
$info$	INTEGER。 $< 0$ の場合、 $info = -k$ は $k$ 番目の引数の値が不正だったことを示す。

### アプリケーション・ノート

$x$  に対するおおまかな制限が計算される。これがオーバーフロー未満の場合は `?trsv` が呼び出され、そうでなければ演算ごとにオーバーフローまたはゼロ除算をチェックする特定のコードが使われる。

$Ax = b$  を解くうえで列方向の格納形式が使用される。 $A$  が下三角の場合、基本アルゴリズムは次のようになる。

```

x[1:n] := b[1:n]
j = 1, ..., n の場合
x(j) := x(j) / A(j,j)
x[j+1:n] := x[j+1:n] - x(j)*A[j+1:n,j]
end

```

ループを  $j$  回反復した後、 $x$  の成分に対する制限を定義する。

$M(j) = x[1:j]$  に対する制限

$G(j) = x[j+1:n]$  に対する制限

初期時は、 $M(0) = 0$  and  $G(0) = \max\{x(i), i=1, \dots, n\}$  とする。

$j+1$  回目の反復で、以下を得る。

$M(j+1) \leq G(j) / |A(j+1, j+1)|$

$G(j+1) \leq G(j) + M(j+1) * |A[j+2:n, j+1]|$

$\leq G(j) (1 + cnorm(j+1) / |A(j+1, j+1)|)$

$cnorm(j+1)$  は、対角を含めない状態で  $A$  の列  $J+1$  の無限ノルムに等しいか大きい。ゆえに、

$$G(j) \leq G(0) \prod_{1 \leq i \leq j} (1 + cnorm(i) / |A(i, i)|)$$

$$|x(j)| \leq (G(0) / |A(j, j)|) \prod_{1 \leq i \leq j} (1 + cnorm(i) / |A(i, i)|)$$

$|x(j)| \leq M(j)$  であるから、 $j=1, \dots, n$  における最大  $M(j)$  の逆数が  $\max(\text{underflow}, 1/\text{overflow})$  よりも大きい場合はレベル 2 の BLAS ルーチン `?trsv` を使用する。

$x(j)$  に対する制限もまた、オーバーフローの可能性のない、列方向の格納形式でのステップの実行判断に使用される。計算された制限が大きな定数よりも大きい場合、オーバーフローを防ぐために  $x$  はスケールリングされる。しかし、制限がオーバーフローする場合、 $x$  は 0 に、 $x(j)$  は 1 に、 $scale$  は 0 にそれぞれ設定され、 $Ax = 0$  に対するゼロでない解が返される。

同様に、列方向の格納形式が  $A^T x = b$  または  $A^H x = b$  を解くために使用される。 $A$  が上三角の場合、基本アルゴリズムは次のようになる。



$j = 1, \dots, n$  の場合、  
 $x(j) := (b(j) - A[1:j-1, j]' x[1:j-1]) / A(j, j)$   
 end

2 つの制限が同時に計算される。

$G(j) = (b(i) - A[1:i-1, i]' x[1:i-1])$  に対する制限、 $1 \leq i \leq j$

$M(j) = x(i)$  に対する制限、 $1 \leq i \leq j$

初期値は  $G(0) = 0$ 、 $M(0) = \max\{b(i), i=1, \dots, n\}$  で、ここで  $j \geq 1$  に対し  $G(j) \geq G(j-1)$  と  $M(j) \geq M(j-1)$  の制約を追加する。

よって  $x(j)$  に対する制限は、

$$M(j) \leq M(j-1) * (1 + cnorm(j)) / |A(j, j)|$$

$$\leq M(0) \prod_{1 \leq i \leq j} (1 + cnorm(i) / |A(i, i)|)$$

これによって、 $1/M(n)$  と  $1/G(n)$  の両方が  $\max(\text{underflow}, 1/\text{overflow})$  を超えている場合でも安全に ?trsv を呼び出すことができる。

## ?latrz

上台形行列を直交 / ユニタリ変換によって  
 因子分解する。

### 構文

```
call slatz( m, n, l, a, lda, tau, work )
call dlatrz( m, n, l, a, lda, tau, work )
call clatz( m, n, l, a, lda, tau, work )
call zlatrz( m, n, l, a, lda, tau, work )
```

### 説明

ルーチン ?latrz は、次の  $m \times (m+1)$  の実数 / 複素上台形行列を、  
 $[A1 \ A2] = [A(1:m, 1:m) \ A(1:m, m+1:n)]$

$(R \ 0) * Z$  として、直交 / ユニタリ変換を使用して因子分解する。Z は  $(m+1) \times (m+1)$  直交 / ユニタリ行列、R と A1 は  $m \times m$  の上三角行列である。

## 入力パラメータ

<i>m</i>	INTEGER。 行列 <i>A</i> の行数。 $m \geq 0$
<i>n</i>	INTEGER。 行列 <i>A</i> の列数。 $n \geq 0$
<i>l</i>	INTEGER。 Householder ベクトルとして意味を持った部分が格納されている行列 <i>A</i> の列数。 $n-m \geq l \geq 0$
<i>a</i>	REAL (slatz の場合) DOUBLE PRECISION (dlatz の場合) COMPLEX (clatz の場合) COMPLEX*16 (zlatrz の場合) 配列、次元は ( <i>lda</i> , <i>n</i> )。 配列 <i>a</i> 先頭の $m \times n$ 上三角形部分には因子分解する行列を格納しておかなければならない。
<i>lda</i>	INTEGER。 配列 <i>a</i> のリーディング・ディメンジョン。 $lda \geq \max(1, m)$
<i>work</i>	REAL (slatz の場合) DOUBLE PRECISION (dlatz の場合) COMPLEX (clatz の場合) COMPLEX*16 (zlatrz の場合) ワークスペース配列、次元は ( <i>m</i> )

## 出力パラメータ

<i>a</i>	<i>a</i> の先頭の $m \times m$ 上三角部分には上三角行列 <i>R</i> が格納される。 <i>a</i> の最初の <i>m</i> 行の <i>n-l+1</i> から <i>n</i> までの成分は、配列 <i>tau</i> とともに、基本リフレクタの積として直交/ユニタリ行列 <i>Z</i> を表現する。
<i>tau</i>	REAL (slatz の場合) DOUBLE PRECISION (dlatz の場合) COMPLEX (clatz の場合) COMPLEX*16 (zlatrz の場合) 配列、次元は ( <i>m</i> )。基本リフレクタのスカラ係数。

## アプリケーション・ノート

因子分解は Householder 法を用いて得る。 $A$  の  $(m - k + 1)$  番目の行にゼロを導入するために使用される  $k$  番目の変換行列  $Z(k)$  は次の形式で与えられる。

$$Z(k) = \begin{bmatrix} I & 0 \\ 0 & T(k) \end{bmatrix}$$

$$T(k) = I - \tau u(k) u(k)' \quad u(k) = \begin{bmatrix} 1 \\ 0 \\ z(k) \end{bmatrix}$$

$\tau$  はスカラ、 $z(k)$  は 1 成分のベクトルである。 $\tau$  と  $z(k)$  は、 $A2$  の  $k$  番目の行の成分をゼロにするために選択される。

スカラ  $\tau$  は  $\tau$  の  $k$  番目の成分で返され、ベクトル  $u(k)$  は、 $z(k)$  の成分が  $a(k, l+1), \dots, a(k, n)$  に格納されるような  $A2$  の  $k$  番目の行で返される。 $R$  の成分は  $A1$  の上三角部分で返される。 $Z$  は次式で与えられる。

$$Z = Z(1) Z(2) \dots Z(m)$$

## ?lauu2

積  $UU^H$  または  $L^H L$  を計算する。ここで  $U$  と  $L$  は上三角または下三角行列(非ブロック化アルゴリズム)。

### 構文

```
call slauu2( uplo, n, a, lda, info )
call dlauu2( uplo, n, a, lda, info )
call clauu2( uplo, n, a, lda, info )
call zlauu2( uplo, n, a, lda, info )
```

## 説明

ルーチン `?lauu2` は積  $UU'$  または  $LL'$  を計算する。ここで、三角係数  $U$  または  $L$  は、それぞれ配列  $a$  の上三角または下三角部分に格納される。

`uplo='U'` または `'u'` の場合、 $a$  の係数  $U$  は結果の上三角で上書きされる。

`uplo='L'` または `'l'` の場合、 $a$  の係数  $L$  は結果の下三角で上書きされる。

このルーチンはアルゴリズムの非ブロック化形式で、[BLAS レベル 2 のルーチン](#) を呼び出す。

## 入力パラメータ

<code>uplo</code>	CHARACTER*1。 配列 $a$ に格納されている三角係数が上三角か下三角かを指定する。 = 'U' の場合、上三角 = 'L' の場合、下三角。
<code>n</code>	INTEGER。 三角係数 $U$ または $L$ の次数。 $n \geq 0$
<code>a</code>	REAL ( <code>slauu2</code> の場合) DOUBLE PRECISION ( <code>dlauu2</code> の場合) COMPLEX ( <code>clauu2</code> の場合) COMPLEX*16 ( <code>zlauu2</code> の場合) 配列、次元は $(lda, n)$ 。三角係数 $U$ または $L$ を格納する。
<code>lda</code>	INTEGER。 配列 $a$ のリーディング・ディメンジョン。 $lda \geq \max(1, n)$ 。

## 出力パラメータ

<code>a</code>	<code>uplo='U'</code> の場合、 $a$ の上三角は積 $UU'$ の上三角で上書きされる。 <code>uplo='L'</code> の場合、 $a$ の下三角は積 $LL'$ の下三角で上書きされる。
<code>info</code>	INTEGER。 = 0 の場合、正常に終了したことを示す。 < 0 の場合、 <code>info=-k</code> は $k$ 番目の引数の値が不正だったことを示す。

## ?lauum

積  $UU^H$  または  $L^H L$  を計算する。ここで  $U$  と  $L$  は上三角または下三角行列(ブロック化アルゴリズム)。

### 構文

```
call slauum( uplo, n, a, lda, info )
call dlauum( uplo, n, a, lda, info )
call clauum( uplo, n, a, lda, info )
call zlauum( uplo, n, a, lda, info )
```

### 説明

ルーチン ?lauum は積  $UU^H$  または  $L^H L$  を計算する。三角係数  $U$  または  $L$  は、配列  $a$  の上三角または下三角部分に格納される。

uplo='U' または 'u' の場合、 $a$  の係数  $U$  は結果の上三角で上書きされる。

uplo='L' または 'l' の場合、 $a$  の係数  $L$  は結果の下三角で上書きされる。

このルーチンはアルゴリズムのブロック化形式で、[BLAS レベル 3 のルーチン](#) を呼び出す。

### 入力パラメータ

uplo	CHARACTER*1。 配列 $a$ に格納されている三角係数が上三角か下三角かを指定する。 = 'U' の場合、上三角 = 'L' の場合、下三角。
n	INTEGER。 三角係数 $U$ または $L$ の次数。 $n \geq 0$
a	REAL (slauum の場合 ) DOUBLE PRECISION (dlauum の場合 ) COMPLEX (clauum の場合 ) COMPLEX*16 (zlauum の場合 ) 配列、次元は (lda, n)。三角係数 $U$ または $L$ を格納する。
lda	INTEGER。 配列 $a$ のリーディング・ディメンジョン。 $lda \geq \max(1, n)$ 。

### 出力パラメータ

<i>a</i>	<i>uplo</i> = 'U' の場合、 <i>a</i> の上三角は積 <i>UU'</i> の上三角で上書きされる。 <i>uplo</i> = 'L' の場合、 <i>a</i> の下三角は積 <i>LL'</i> の下三角で上書きされる。
<i>info</i>	INTEGER。 = 0 の場合、正常に終了したことを示す。 < 0 の場合、 <i>info</i> = - <i>k</i> は <i>k</i> 番目の引数の値が不正だったことを示す。

---

## ?org2l/?ung2l

?geqlf で求めた *QL* 因子分解から、全部または一部の直交/ユニタリ行列 *Q* を生成する (非ブロック化アルゴリズム)。

---

### 構文

```
call sorg2l( m, n, k, a, lda, tau, work, info )
call dorg2l( m, n, k, a, lda, tau, work, info )
call cung2l( m, n, k, a, lda, tau, work, info )
call zung2l( m, n, k, a, lda, tau, work, info )
```

### 説明

ルーチン ?org2l/?ung2l は、直交列を持つ  $m \times n$  の実数 / 複素行列 *Q* を生成する。これは、次数 *m* の *k* 個の基本リフレクタの積の最後の *n* 列として定義される。

[?geqlf](#) によって返される  $Q = H(k) \dots H(2) H(1)$

### 入力パラメータ

<i>m</i>	INTEGER。 行列 <i>Q</i> の行数。 $m \geq 0$
<i>n</i>	INTEGER。 行列 <i>Q</i> の列数。 $m \geq n \geq 0$
<i>k</i>	INTEGER。 その積が行列 <i>Q</i> を定義する基本リフレクタの個数。 $n \geq k \geq 0$

<i>a</i>	<p>REAL (sorg21 の場合 )</p> <p>DOUBLE PRECISION (dorg21 の場合 )</p> <p>COMPLEX (cung21 の場合 )</p> <p>COMPLEX*16 (zung21 の場合 )</p> <p>配列、次元は (<i>lda</i>, <i>n</i>)</p> <p><i>a</i> の (<i>n</i>-<i>k</i>+<i>i</i>) 番目の列には、?geqlf によって配列引数 <i>a</i> の最後の <i>k</i> 列に返されたとおりに、基本リフレクタ <math>H(i)</math> を定義するベクトルを格納しておかなければならない。ここで、<math>i = 1, 2, \dots, k</math></p>
<i>lda</i>	<p>INTEGER。</p> <p>配列 <i>a</i> の第 1 次元。 <math>lda \geq \max(1, m)</math></p>
<i>tau</i>	<p>REAL (sorg21 の場合 )</p> <p>DOUBLE PRECISION (dorg21 の場合 )</p> <p>COMPLEX (cung21 の場合 )</p> <p>COMPLEX*16 (zung21 の場合 )</p> <p>配列、次元は (<i>k</i>)。</p> <p><math>\tau(i)</math> には、?geqlf から返されたとおりに、基本リフレクタ <math>H(i)</math> のスカラー係数を格納しておかなければならない。</p>
<i>work</i>	<p>REAL (sorg21 の場合 )</p> <p>DOUBLE PRECISION (dorg21 の場合 )</p> <p>COMPLEX (cung21 の場合 )</p> <p>COMPLEX*16 (zung21 の場合 )</p> <p>ワークスペース配列、次元は (<i>n</i>)。</p>

## 出力パラメータ

<i>a</i>	$m \times n$ の行列 $Q$ で上書きされる。
<i>info</i>	<p>INTEGER。</p> <p>= 0 の場合、正常に終了したことを示す。</p> <p>&lt; 0 の場合、<math>info = -i</math> は <i>i</i> 番目の引数の値が不正だったことを示す。</p>

---

## ?org2r/?ung2r

?geqrf で求めた  $QR$  因子分解から、全部または一部の直交/ユニタリ行列  $Q$  を生成する(非ブロック化アルゴリズム)。

---

### 構文

```
call sorg2r( m, n, k, a, lda, tau, work, info )
call dorg2r( m, n, k, a, lda, tau, work, info )
call cung2r( m, n, k, a, lda, tau, work, info )
call zung2r( m, n, k, a, lda, tau, work, info )
```

### 説明

ルーチン ?org2r/?ung2r は、直交列を持つ  $m \times n$  の実数/複素行列  $Q$  を生成する。これは、次数  $m$  の  $k$  個の基本リフレクタの積の最初の  $n$  列として定義される。

$$Q = H(1)H(2) \dots H(k)$$

([?geqrf](#) によって返される。)

### 入力パラメータ

$m$	INTEGER。 行列 $Q$ の行数。 $m \geq 0$
$n$	INTEGER。 行列 $Q$ の列数。 $m \geq n \geq 0$
$k$	INTEGER。 その積が行列 $Q$ を定義する基本リフレクタの個数。 $n \geq k \geq 0$
$a$	REAL(sorg2r の場合) DOUBLE PRECISION(dorg2r の場合) COMPLEX(cung2r の場合) COMPLEX*16(zung2r の場合) 配列、次元は $(lda, n)$ 。 $a$ の $i$ 番目の列には、?geqrf によって配列引数 $a$ の最初の $k$ 列に返されたとおりに、基本リフレクタ $H(i)$ を定義するベクトルを格納しておかなければならない。ここで、 $i = 1, 2, \dots, k$



<i>lda</i>	INTEGER。 配列 <i>a</i> の第 1 次元。 $lda \geq \max(1, m)$
<i>tau</i>	REAL (sorg2r の場合 ) DOUBLE PRECISION (dorg2r の場合 ) COMPLEX (cung2r の場合 ) COMPLEX*16 (zung2r の場合 ) 配列、次元は ( <i>k</i> )。 <i>tau(i)</i> には、?geqrf から返されたとおりに、基本リフレクタ $H(i)$ のス カラ係数を格納しておかなければならない。
<i>work</i>	REAL (sorg2r の場合 ) DOUBLE PRECISION (dorg2r の場合 ) COMPLEX (cung2r の場合 ) COMPLEX*16 (zung2r の場合 ) ワークスペース配列、次元は ( <i>n</i> )。

### 出力パラメータ

<i>a</i>	$m \times n$ の行列 $Q$ で上書きされる。
<i>info</i>	INTEGER。 = 0 の場合、正常に終了したことを示す。 < 0 の場合、 <i>info</i> = - <i>i</i> は <i>i</i> 番目の引数の値が不正だったことを示す。

## ?orgl2/?ungl2

?gelqf で求めた  $LQ$  因子分解から、全部または一部の直交/ユニタリ行列  $Q$  を生成する ( 非ブ  
ロック化アルゴリズム )。

### 構文

```
call sorgl2( m, n, k, a, lda, tau, work, info )
call dorgl2( m, n, k, a, lda, tau, work, info )
call cungl2( m, n, k, a, lda, tau, work, info )
call zungl2( m, n, k, a, lda, tau, work, info )
```

## 説明

ルーチン `?org21/?ung21` は、直交行を持つ  $m \times n$  の実数 / 複素行列  $Q$  を生成する。これは、次数  $n$  の  $k$  個の基本リフレクタの積の最初の  $m$  行として定義される。

$$Q = H(k) \dots H(2)H(1) \text{ または } Q = H(k)' \dots H(2)'H(1)'$$

([?gelqf](#) によって返される。)

## 入力パラメータ

$m$	INTEGER。 行列 $Q$ の行数。 $m \geq 0$
$n$	INTEGER。 行列 $Q$ の列数。 $n \geq m$
$k$	INTEGER。 その積が行列 $Q$ を定義する基本リフレクタの個数。 $m \geq k \geq 0$
$a$	REAL ( <code>sorg12</code> の場合 ) DOUBLE PRECISION ( <code>dorg12</code> の場合 ) COMPLEX ( <code>cung12</code> の場合 ) COMPLEX*16 ( <code>zung12</code> の場合 ) 配列、次元は $(lda, n)$ 。 $a$ の $i$ 番目の行には、 <code>?gelqf</code> によって配列引数 $a$ の最初の $k$ 行に返されたとおりに、基本リフレクタ $H(i)$ を定義するベクトルを格納しておかなければならない。ここで、 $i = 1, 2, \dots, k$
$lda$	INTEGER。 配列 $a$ の第 1 次元。 $lda \geq \max(1, m)$
$tau$	REAL ( <code>sorg12</code> の場合 ) DOUBLE PRECISION ( <code>dorg12</code> の場合 ) COMPLEX ( <code>cung12</code> の場合 ) COMPLEX*16 ( <code>zung12</code> の場合 ) 配列、次元は $(k)$ 。 $tau(i)$ には、 <code>?gelqf</code> から返されたとおりに、基本リフレクタ $H(i)$ のスカラー係数を格納しておかなければならない。
$work$	REAL ( <code>sorg12</code> の場合 ) DOUBLE PRECISION ( <code>dorg12</code> の場合 ) COMPLEX ( <code>cung12</code> の場合 ) COMPLEX*16 ( <code>zung12</code> の場合 ) ワークスペース配列、次元は $(m)$

### 出力パラメータ

*a*  $m \times n$  の行列  $Q$  で上書きされる。

*info* INTEGER。  
 = 0 の場合、正常に終了したことを示す。  
 < 0 の場合、*info* = -*i* は *i* 番目の引数の値が不正だったことを示す。

## ?orgr2/?ungr2

?gerqf で求めた  $RQ$  因子分解から、全部  
 または一部の直交/ユニタリ行列  $Q$  を生成する  
 (非ブロック化アルゴリズム)。

### 構文

```
call sorgr2( m, n, k, a, lda, tau, work, info )
call dorgr2( m, n, k, a, lda, tau, work, info )
call cungr2( m, n, k, a, lda, tau, work, info )
call zungr2( m, n, k, a, lda, tau, work, info )
```

### 説明

ルーチン ?orgr2/?ungr2 は、直交行を持つ  $m \times n$  の実数の行列  $Q$  を生成する。これは、[?gerqf](#) から返されたとおり、次数  $n$  の  $k$  個の基本リフレクタの積の最後の  $m$  行として定義される。 $Q = H(1)H(2) \dots H(k)$  または  $Q = H(1)'H(2)' \dots H(k)'$

### 入力パラメータ

*m* INTEGER。行列  $Q$  の行数。  $m \geq 0$

*n* INTEGER。  
 行列  $Q$  の列数。  $n \geq m$

*k* INTEGER。  
 その積が行列  $Q$  を定義する基本リフレクタの個数。  $m \geq k \geq 0$

*a* REAL (sorgr2 の場合)  
 DOUBLE PRECISION (dorgr2 の場合)  
 COMPLEX (cungr2 の場合)  
 COMPLEX\*16 (zungr2 の場合) 配列、次元は (*lda*, *n*)。

$a$  の  $(m-k+i)$  番目の行には、?gerqf によって配列引数  $a$  の最後の  $k$  行に返されたとおりに、基本リフレクタ  $H(i)$  を定義するベクトルを格納しておかなければならない。ここで、 $i = 1, 2, \dots, k$

<i>lda</i>	INTEGER。 配列 $a$ の第 1 次元。 $lda \geq \max(1, m)$
<i>tau</i>	REAL (sorgr2 の場合 ) DOUBLE PRECISION (dorgr2 の場合 ) COMPLEX (cungr2 の場合 ) COMPLEX*16 (zungr2 の場合 ) 配列、次元は $(k)$ 。 $tau(i)$ には、?gerqf から返されたとおりに、基本リフレクタ $H(i)$ のスカラ係数を格納しておかなければならない。
<i>work</i>	REAL (sorgr2 の場合 ) DOUBLE PRECISION (dorgr2 の場合 ) COMPLEX (cungr2 の場合 ) COMPLEX*16 (zungr2 の場合 ) ワークスペース配列、次元は $(m)$

### 出力パラメータ

<i>a</i>	$m \times n$ の行列 $Q$ で上書きされる。
<i>info</i>	INTEGER。 = 0 の場合、正常に終了したことを示す。 < 0 の場合、 $info = -i$ は $i$ 番目の引数の値が不正だったことを示す。

---

## ?orm2l/?unm2l

一般行列と ?geqlf で求めた  $QL$  因子分解の直交/  
ユニタリ行列を乗算する ( 非ブロック化アルゴリズム )。

---

### 構文

```
call sorm2l( side, trans, m, n, k, a, lda, tau, c, ldc, work, info )
call dorm2l( side, trans, m, n, k, a, lda, tau, c, ldc, work, info )
call cunm2l( side, trans, m, n, k, a, lda, tau, c, ldc, work, info )
call zunm2l( side, trans, m, n, k, a, lda, tau, c, ldc, work, info )
```

## 説明

ルーチン `?orm2l/?unm2l` は一般実数 / 複素  $m \times n$  行列  $C$  を以下で上書きする。

$side = 'L'$  かつ  $trans = 'N'$  の場合、 $Q * C$ 、  
 $side = 'L'$  かつ、 $trans = 'T'$  (実数型) または  
 $trans = 'C'$  (複素型) の場合、 $Q' * C$ 、  
 $side = 'R'$  かつ  $trans = 'N'$  の場合、 $C * Q$ 、  
 $side = 'R'$  かつ、 $trans = 'T'$  (実数型) または  
 $trans = 'C'$  (複素型) の場合、 $C * Q'$

$Q$  は、 $k$  個の基本リフレクタの積として定義される実数直交または複素ユニタリ行列である。

$$Q = H(k) \dots H(2) H(1)$$

([?geqlf](#) によって返される。)  $Q$  の次数は、 $side = 'L'$  の場合は  $m$ 、 $side = 'R'$  の場合は  $n$  である。

## 入力パラメータ

**side** CHARACTER\*1。  
 = 'L' の場合、 $Q$  または  $Q'$  を左から適用する。  
 = 'R' の場合、 $Q$  または  $Q'$  を右から適用する。

**trans** CHARACTER\*1。  
 = 'N' の場合、 $Q$  を適用する (転置なし)  
 = 'T' の場合、 $Q'$  を適用する (転置、実数型の場合)  
 = 'C' の場合、 $Q'$  を適用する (共役転置、複素型の場合)

**m** INTEGER。  
 行列  $C$  の行数。  $m \geq 0$

**n** INTEGER。  
 行列  $C$  の列数。  $n \geq 0$

**k** INTEGER。  
 積が行列  $Q$  を定義する基本リフレクタの個数。  
 $side = 'L'$  の場合、 $m \geq k \geq 0$ 、  
 $side = 'R'$  の場合、 $n \geq k \geq 0$

**a** REAL (sorm2l の場合)  
 DOUBLE PRECISION (dorm2l の場合)  
 COMPLEX (cunm2l の場合)  
 COMPLEX\*16 (zunm2l の場合) 配列、次元は (lda, k)

$a$  の  $i$  番目の列には、?geqlf によって配列引数  $a$  の最後の  $k$  列に返されたとおりに、基本リフレクタ  $H(i)$  を定義するベクトルを格納しておかなければならない。ここで、 $i = 1, 2, \dots, k$ 。配列  $a$  はルーチンにより変更されるが終了時に復元される。

<i>lda</i>	INTEGER。 配列 $A$ のリーディング・ディメンジョン。 <i>side</i> = 'L' の場合 $lda \geq \max(1, m)$ 、 <i>side</i> = 'R' の場合 $lda \geq \max(1, n)$
<i>tau</i>	REAL (sorm21 の場合 ) DOUBLE PRECISION (dorm21 の場合 ) COMPLEX (cunm21 の場合 ) COMPLEX*16 (zunm21 の場合 ) 配列、次元は $(k)$ 。 $\tau(i)$ には、?geqlf から返されたとおりに、基本リフレクタ $h(i)$ のスカラー係数を格納しておかなければならない。
<i>c</i>	REAL (sorm21 の場合 ) DOUBLE PRECISION (dorm21 の場合 ) COMPLEX (cunm21 の場合 ) COMPLEX*16 (zunm21 の場合 ) 配列、次元は $(ldc, n)$ 。 $m \times n$ の行列 $C$ を格納する。
<i>ldc</i>	INTEGER。 配列 $C$ のリーディング・ディメンジョン。 $ldc \geq \max(1, m)$
<i>work</i>	REAL (sorm21 の場合 ) DOUBLE PRECISION (dorm21 の場合 ) COMPLEX (cunm21 の場合 ) COMPLEX*16 (zunm21 の場合 ) ワークスペース配列、次元は <i>side</i> = 'L' の場合、 $(n)$ <i>side</i> = 'R' の場合、 $(m)$

## 出力パラメータ

<i>c</i>	$c$ は $QC$ または $Q'C$ 、あるいは $CQ'$ または $CQ$ によって上書きされる。
<i>info</i>	INTEGER。 = 0 の場合、正常に終了したことを示す。 < 0 の場合、 <i>info</i> = - $i$ は $i$ 番目の引数の値が不正だったことを示す。

## ?orm2r/?unm2r

一般行列と [?geqrf](#) で求めた *QR* 因子分解の直交/  
ユニタリ行列とを乗算する ( 非ブロック化アルゴ  
リズム )。

### 構文

```
call sorm2r( side, trans, m, n, k, a, lda, tau, c, ldc, work, info )
call dorm2r( side, trans, m, n, k, a, lda, tau, c, ldc, work, info )
call cunm2r( side, trans, m, n, k, a, lda, tau, c, ldc, work, info )
call zunm2r( side, trans, m, n, k, a, lda, tau, c, ldc, work, info )
```

### 説明

ルーチン [?orm2l/?unm2l](#) は一般実数 / 複素  $m \times n$  行列  $C$  を以下で上書きする。

$side = 'L'$  かつ  $trans = 'N'$  の場合、 $Q \cdot C$ 、  
 $side = 'L'$  かつ、 $trans = 'T'$  ( 実数型 ) または  
 $trans = 'c'$  ( 複素型 ) の場合、 $Q' \cdot C$ 、  
 $side = 'R'$  かつ  $trans = 'N'$  の場合、 $C \cdot Q$ 、  
 $side = 'R'$  かつ、 $trans = 'T'$  ( 実数型 ) または  
 $trans = 'c'$  ( 複素型 ) の場合、 $C \cdot Q'$

$Q$  は、 $k$  個の基本リフレクタの積として定義される実数直交または複素ユニタリ行列である。

$$Q = H(1) H(2) \dots H(k)$$

([?geqrf](#) によって返される。)  $Q$  の次数は、 $side = 'L'$  の場合は  $m$ 、 $side = 'R'$  の場合は  $n$  である。

### 入力パラメータ

$side$  CHARACTER\*1。  
 = 'L' の場合、 $Q$  または  $Q'$  を左から適用する。  
 = 'R' の場合、 $Q$  または  $Q'$  を右から適用する。

$trans$  CHARACTER\*1。  
 = 'N' の場合、 $Q$  を適用する ( 転置なし )  
 = 'T' の場合、 $Q'$  を適用する ( 転置、実数型の場合 )  
 = 'c' の場合、 $Q'$  を適用する ( 共役転置、複素型の場合 )

<i>m</i>	INTEGER。 行列 <i>C</i> の行数。 $m \geq 0$
<i>n</i>	INTEGER。 行列 <i>C</i> の列数。 $n \geq 0$
<i>k</i>	INTEGER。 その積が行列 <i>Q</i> を定義する基本リフレクタの個数。 <i>side</i> = 'L' の場合、 $m \geq k \geq 0$ 、 <i>side</i> = 'R' の場合、 $n \geq k \geq 0$
<i>a</i>	REAL (sorm2r の場合 ) DOUBLE PRECISION (dorm2r の場合 ) COMPLEX (cunm2r の場合 ) COMPLEX*16 (zunm2r の場合 ) 配列、次元は ( <i>lda</i> , <i>k</i> ) <i>a</i> の <i>i</i> 番目の列には、?geqrf によって配列引数 <i>a</i> の最初の <i>k</i> 列に返されたとおりに、基本リフレクタ <i>H</i> ( <i>i</i> ) を定義するベクトルを格納しておかなければならない。ここで、 $i = 1, 2, \dots, k$ 。配列 <i>a</i> はルーチンにより変更されるが終了時に復元される。
<i>lda</i>	INTEGER。 配列 <i>a</i> のリーディング・ディメンジョン。 <i>side</i> = 'L' の場合 $lda \geq \max(1, m)$ 、 <i>side</i> = 'R' の場合 $lda \geq \max(1, n)$
<i>tau</i>	REAL (sorm2r の場合 ) DOUBLE PRECISION (dorm2r の場合 ) COMPLEX (cunm2r の場合 ) COMPLEX*16 (zunm2r の場合 ) 配列、次元は ( <i>k</i> )。 <i>tau</i> ( <i>i</i> ) には、?geqrf から返されたとおりに、基本リフレクタ <i>H</i> ( <i>i</i> ) のスカラー係数を格納しておかなければならない。
<i>c</i>	REAL (sorm2r の場合 ) DOUBLE PRECISION (dorm2r の場合 ) COMPLEX (cunm2r の場合 ) COMPLEX*16 (zunm2r の場合 ) 配列、次元は ( <i>ldc</i> , <i>n</i> )。 $m \times n$ の行列 <i>C</i> を格納する。
<i>ldc</i>	INTEGER。 配列 <i>C</i> のリーディング・ディメンジョン。 $ldc \geq \max(1, m)$



`work` REAL (sorm2r の場合 )  
 DOUBLE PRECISION (dorm2r の場合 )  
 COMPLEX (cunm2r の場合 )  
 COMPLEX\*16 (zunm2r の場合 )  
 ワークスペース配列、次元は  
 $side = 'L'$  の場合、 $(n)$   
 $side = 'R'$  の場合、 $(m)$

### 出力パラメータ

`c`  $c$  は  $QC$  または  $Q'C$ 、あるいは  $CQ'$  または  $CQ$  によって上書きされる。

`info` INTEGER。  
 $= 0$  の場合、正常に終了したことを示す。  
 $< 0$  の場合、`info` =  $-i$  は  $i$  番目の引数の値が不正だったことを示す。

## ?orml2/?unml2

一般行列と ?gelqf で求めた  $LQ$  因子分解の直交/  
 ユニタリ行列とを乗算する ( 非ブロック化アルゴ  
 リズム )。

### 構文

```
call sorml2( side, trans, m, n, k, a, lda, tau, c, ldc, work, info )
call dorml2( side, trans, m, n, k, a, lda, tau, c, ldc, work, info )
call cunml2( side, trans, m, n, k, a, lda, tau, c, ldc, work, info )
call zunml2( side, trans, m, n, k, a, lda, tau, c, ldc, work, info )
```

### 説明

ルーチン ?orml2/?unml2 は一般実数 / 複素  $m \times n$  行列  $C$  を以下で上書きする。

$side = 'L'$  かつ  $trans = 'N'$  の場合、 $Q \star C$ 、  
 $side = 'L'$  かつ、 $trans = 'T'$  ( 実数型 ) または  
 $trans = 'C'$  ( 複素型 ) の場合、 $Q' \star C$ 、  
 $side = 'R'$  かつ  $trans = 'N'$  の場合、 $C \star Q$ 、  
 $side = 'R'$  かつ、 $trans = 'T'$  ( 実数型 ) または  
 $trans = 'C'$  ( 複素型 ) の場合、 $C \star Q'$

$Q$  は、 $k$  個の基本リフレクタの積として定義される実数直交または複素ユニタリ行列である。

$$Q = H(k) \dots H(2) H(1) \text{ または } Q = H(k)' \dots H(2)' H(1)'$$

([?gelqf](#) によって返される。)  $Q$  の次数は、 $side = 'L'$  の場合は  $m$ 、 $side = 'R'$  の場合は  $n$  である。

## 入力パラメータ

<i>side</i>	CHARACTER*1。 = 'L' の場合、 $Q$ または $Q'$ を左から適用する。 = 'R' の場合、 $Q$ または $Q'$ を右から適用する。
<i>trans</i>	CHARACTER*1。 = 'N' の場合、 $Q$ を適用する (転置なし)。 = 'T' の場合、 $Q'$ を適用する (転置、実数型の場合) = 'C' の場合、 $Q'$ を適用する (共役転置、複素型の場合)
<i>m</i>	INTEGER。 行列 $C$ の行数。 $m \geq 0$
<i>n</i>	INTEGER。 行列 $C$ の列数。 $n \geq 0$
<i>k</i>	INTEGER。 基本リフレクタ (その積で行列 $Q$ を定義) の個数。 $side = 'L'$ の場合、 $m \geq k \geq 0$ 、 $side = 'R'$ の場合、 $n \geq k \geq 0$
<i>a</i>	REAL (sorml2 の場合) DOUBLE PRECISION (dorml2 の場合) COMPLEX (cunml2 の場合) COMPLEX*16 (zunml2 の場合) 配列、次元は $side = 'L'$ の場合、 $(lda, m)$ $side = 'R'$ の場合、 $(lda, n)$ $a$ の $i$ 番目の行には、 <a href="#">?gelqf</a> によって配列引数 $a$ の最初の $k$ 行に返されたとおりに、基本リフレクタ $H(i)$ を定義するベクトルを格納しておかなければならない。ここで、 $i = 1, 2, \dots, k$ 。配列 $a$ はルーチンにより変更されるが終了時に復元される。
<i>lda</i>	INTEGER。 配列 $A$ のリーディング・ディメンジョン。 $lda \geq \max(1, k)$

<i>tau</i>	<p>REAL (sorml2 の場合 )</p> <p>DOUBLE PRECISION (dorml2 の場合 )</p> <p>COMPLEX (cunml2 の場合 )</p> <p>COMPLEX*16 (zunml2 の場合 ) 配列、次元は <math>(k)</math>。</p> <p><math>\tau(i)</math> には、?gelqf から返されたとおりに、基本リフレクタ <math>H(i)</math> のスカラー係数を格納しておかなければならない。</p>
<i>c</i>	<p>REAL (sorml2 の場合 )</p> <p>DOUBLE PRECISION (dorml2 の場合 )</p> <p>COMPLEX (cunml2 の場合 )</p> <p>COMPLEX*16 (zunml2 の場合 )</p> <p>配列、次元は <math>(ldc, n)</math>。</p> <p><math>m \times n</math> の行列 <math>C</math> を格納する。</p>
<i>ldc</i>	<p>INTEGER。</p> <p>配列 <math>c</math> のリーディング・ディメンジョン。 <math>ldc \geq \max(1, m)</math></p>
<i>work</i>	<p>REAL (sorml2 の場合 )</p> <p>DOUBLE PRECISION (dorml2 の場合 )</p> <p>COMPLEX (cunml2 の場合 )</p> <p>COMPLEX*16 (zunml2 の場合 )</p> <p>ワークスペース配列、次元は</p> <p><math>side = 'L'</math> の場合、 <math>(n)</math></p> <p><math>side = 'R'</math> の場合、 <math>(m)</math></p>

### 出力パラメータ

<i>c</i>	$c$ は $QC$ または $Q'C$ 、あるいは $CQ'$ または $CQ$ によって上書きされる。
<i>info</i>	<p>INTEGER。</p> <p><math>= 0</math> の場合、正常に終了したことを示す。</p> <p><math>&lt; 0</math> の場合、<math>info = -i</math> は <math>i</math> 番目の引数の値が不正だったことを示す。</p>

## ?ormr2/?unmr2

一般行列と [?gerqf](#) で求めた  $RQ$  因子分解の直交/  
ユニタリ行列とを乗算する ( 非ブロック化アルゴ  
リズム )。

### 構文

```
call sormr2( side, trans, m, n, k, a, lda, tau, c, ldc, work, info )
call dormr2( side, trans, m, n, k, a, lda, tau, c, ldc, work, info )
call cunmr2( side, trans, m, n, k, a, lda, tau, c, ldc, work, info )
call zunmr2( side, trans, m, n, k, a, lda, tau, c, ldc, work, info )
```

### 説明

ルーチン [?ormr2/?unmr2](#) は一般実数 / 複素  $m \times n$  行列  $C$  を以下で上書きする。

$side = 'L'$  かつ  $trans = 'N'$  の場合、 $Q * C$ 、  
 $side = 'L'$  かつ、 $trans = 'T'$  ( 実数型 ) または  
 $trans = 'c'$  ( 複素型 ) の場合、 $Q' * C$ 、  
 $side = 'R'$  かつ  $trans = 'N'$  の場合、 $C * Q$ 、  
 $side = 'R'$  かつ、 $trans = 'T'$  ( 実数型 ) または  
 $trans = 'c'$  ( 複素型 ) の場合、 $C * Q'$

$Q$  は、 $k$  個の基本リフレクタの積として定義される実数直交または複素ユニタリ行列である。

$$Q = H(1) H(2) \dots H(k) \text{ または } Q = H(1)' H(2)' \dots H(k)'$$

([?gerqf](#) によって返される。)  $Q$  の次数は、 $side = 'L'$  の場合は  $m$ 、 $side = 'R'$  の場合は  $n$  である。

### 入力パラメータ

**side** CHARACTER\*1。  
= 'L' の場合、 $Q$  または  $Q'$  を左から適用する。  
= 'R' の場合、 $Q$  または  $Q'$  を右から適用する。

**trans** CHARACTER\*1。  
= 'N' の場合、 $Q$  を適用する ( 転置なし )。  
= 'T' の場合、 $Q'$  を適用する ( 転置、実数型の場合 )  
= 'c' の場合、 $Q'$  を適用する ( 共役転置、複素型の場合 )

<i>m</i>	INTEGER。 行列 <i>C</i> の行数。 $m \geq 0$
<i>n</i>	INTEGER。 行列 <i>C</i> の列数。 $n \geq 0$
<i>k</i>	INTEGER。 その積が行列 <i>Q</i> を定義する基本リフレクタの個数。 <i>side</i> = 'L' の場合、 $m \geq k \geq 0$ 、 <i>side</i> = 'R' の場合、 $n \geq k \geq 0$
<i>a</i>	REAL (sormr2 の場合 ) DOUBLE PRECISION (dormr2 の場合 ) COMPLEX (cunmr2 の場合 ) COMPLEX*16 (zunmr2 の場合 ) 配列、次元は <i>side</i> = 'L' の場合、 ( <i>lda</i> , <i>m</i> ) <i>side</i> = 'R' の場合、 ( <i>lda</i> , <i>n</i> ) <i>a</i> の <i>i</i> 番目の行には、?gerqf によって配列引数 <i>A</i> の最後の <i>k</i> 行に返されたとおりに、基本リフレクタ <i>H</i> ( <i>i</i> ) を定義するベクトルを格納しておかなければならない。ここで、 $i = 1, 2, \dots, k$ 。配列 <i>a</i> はルーチンにより変更されるが終了時に復元される。
<i>lda</i>	INTEGER。 配列 <i>A</i> のリーディング・ディメンジョン。 $lda \geq \max(1, k)$
<i>tau</i>	REAL (sormr2 の場合 ) DOUBLE PRECISION (dormr2 の場合 ) COMPLEX (cunmr2 の場合 ) COMPLEX*16 (zunmr2 の場合 ) 配列、次元は ( <i>k</i> )。 <i>tau</i> ( <i>i</i> ) には、?gerqf から返されたとおりに、基本リフレクタ <i>H</i> ( <i>i</i> ) のスカラー係数を格納しておかなければならない。
<i>c</i>	REAL (sormr2 の場合 ) DOUBLE PRECISION (dormr2 の場合 ) COMPLEX (cunmr2 の場合 ) COMPLEX*16 (zunmr2 の場合 ) 配列、次元は ( <i>ldc</i> , <i>n</i> )。 $m \times n$ の行列 <i>C</i> を格納する。
<i>ldc</i>	INTEGER。 配列 <i>C</i> のリーディング・ディメンジョン。 $ldc \geq \max(1, m)$

`work` REAL (sormr2 の場合 )  
DOUBLE PRECISION (dormr2 の場合 )  
COMPLEX (cunmr2 の場合 )  
COMPLEX\*16 (zunmr2 の場合 )  
ワークスペース配列、次元は  
`side = 'L'` の場合、(n)  
`side = 'R'` の場合、(m)

### 出力パラメータ

`c`  $C$  は  $QC$  または  $Q'C$ 、あるいは  $CQ'$  または  $CQ$  によって上書きされる。

`info` INTEGER。  
= 0 の場合、正常に終了したことを示す。  
< 0 の場合、`info = -i` は  $i$  番目の引数の値が不正だったことを示す。

---

## ?ormr3/?unmr3

一般行列と ?tzrzf で求めた RZ 因子分解の直交/  
ユニタリ行列とを乗算する ( 非ブロック化アルゴ  
リズム )。

---

### 構文

```
call sormr3( side, trans, m, n, k, l, a, lda, tau, c, ldc, work, info )
call dormr3( side, trans, m, n, k, l, a, lda, tau, c, ldc, work, info )
call cunmr3( side, trans, m, n, k, l, a, lda, tau, c, ldc, work, info )
call zunmr3( side, trans, m, n, k, l, a, lda, tau, c, ldc, work, info )
```

### 説明

ルーチン ?ormr3/?unmr3 は一般実数 / 複素  $m \times n$  行列  $C$  を以下で上書きする。

`side = 'L'` かつ `trans = 'N'` の場合、 $Q \star C$ 、  
`side = 'L'` かつ、`trans = 'T'` ( 実数型 ) または  
`trans = 'C'` ( 複素型 ) の場合、 $Q' \star C$ 、  
`side = 'R'` かつ `trans = 'N'` の場合、 $C \star Q$ 、  
`side = 'R'` かつ、`trans = 'T'` ( 実数型 ) または  
`trans = 'C'` ( 複素型 ) の場合、 $C \star Q'$

$Q$  は、 $k$  個の基本リフレクタの積として定義される実数直交または複素ユニタリ行列である。

$$Q = H(1) H(2) \dots H(k)$$

([?tzrzf](#) によって返される。)  $Q$  の次数は、 $side = 'L'$  の場合は  $m$ 、 $side = 'R'$  の場合は  $n$  である。

## 入力パラメータ

<i>side</i>	CHARACTER*1。 = 'L' の場合、 $Q$ または $Q'$ を左から適用する。 = 'R' の場合、 $Q$ または $Q'$ を右から適用する。
<i>trans</i>	CHARACTER*1。 = 'N' の場合、 $Q$ を適用する (転置なし)。 = 'T' の場合、 $Q'$ を適用する (転置、実数型の場合) = 'C' の場合、 $Q'$ を適用する (共役転置、複素型の場合)
<i>m</i>	INTEGER。 行列 $C$ の行数。 $m \geq 0$
<i>n</i>	INTEGER。 行列 $C$ の列数。 $n \geq 0$
<i>k</i>	INTEGER。 その積が行列 $Q$ を定義する基本リフレクタの個数。 $side = 'L'$ の場合、 $m \geq k \geq 0$ 、 $side = 'R'$ の場合、 $n \geq k \geq 0$
<i>l</i>	INTEGER。 Householder リフレクタとして意味を持った部分が格納されている行列 $A$ の列数。 $side = 'L'$ の場合、 $m \geq l \geq 0$ 、 $side = 'R'$ の場合、 $n \geq l \geq 0$
<i>a</i>	REAL (sormr3 の場合) DOUBLE PRECISION (dormr3 の場合) COMPLEX (cunmr3 の場合) COMPLEX*16 (zunmr3 の場合) 配列、次元は $side = 'L'$ の場合、 $(lda, m)$ $side = 'R'$ の場合、 $(lda, n)$ $a$ の $i$ 番目の行には、 <a href="#">?tzrzf</a> によって配列引数 $a$ の最後の $k$ 行に返さ

れたとおりに、基本リフレクタ  $H(i)$  を定義するベクトルを格納しておかなければならない。ここで、 $i = 1, 2, \dots, k$ 。配列  $a$  はルーチンにより変更されるが終了時に復元される。

<i>lda</i>	INTEGER。 配列 $A$ のリーディング・ディメンジョン。 $lda \geq \max(1, k)$
<i>tau</i>	REAL (sormr3 の場合 ) DOUBLE PRECISION (dormr3 の場合 ) COMPLEX (cunmr3 の場合 ) COMPLEX*16 (zunmr3 の場合 ) 配列、次元は $(k)$ 。 $\tau(i)$ には、?tzrzf から返されたとおりに、基本リフレクタ $H(i)$ のスカラー係数を格納しておかなければならない。
<i>a</i>	REAL (sormr3 の場合 ) DOUBLE PRECISION (dormr3 の場合 ) COMPLEX (cunmr3 の場合 ) COMPLEX*16 (zunmr3 の場合 ) 配列、次元は $(ldc, n)$ 。 $m \times n$ の行列 $C$ を格納する。
<i>ldc</i>	INTEGER。 配列 $c$ のリーディング・ディメンジョン。 $ldc \geq \max(1, m)$
<i>work</i>	REAL (sormr3 の場合 ) DOUBLE PRECISION (dormr3 の場合 ) COMPLEX (cunmr3 の場合 ) COMPLEX*16 (zunmr3 の場合 ) ワークスペース配列、次元は $side = 'L'$ の場合、 $(n)$ $side = 'R'$ の場合、 $(m)$

## 出力パラメータ

<i>c</i>	$c$ は $QC$ または $Q'C$ 、あるいは $CQ'$ または $CQ$ によって上書きされる。
<i>info</i>	INTEGER。 = 0 の場合、正常に終了したことを示す。 < 0 の場合、 $info = -i$ は $i$ 番目の引数の値が不正だったことを示す。



## ?pbtf2

対称 / エルミート 正定値帯行列のコレスキー因子分解を計算する ( 非ブロック化アルゴリズム )。

### 構文

```
call spbtf2( uplo, n, kd, ab, ldab, info )
call dpbtf2( uplo, n, kd, ab, ldab, info )
call cpbtf2( uplo, n, kd, ab, ldab, info )
call zpbtf2( uplo, n, kd, ab, ldab, info )
```

### 説明

このルーチンは、実対称 / 複素エルミート 正定値帯行列  $A$  に対してコレスキー因子分解を行う。因子分解は以下の形式を持つ。

$A = U^H U$ 、 $uplo = 'U'$  の場合、または  
 $A = L L^H$ 、 $uplo = 'L'$  の場合

$U$  は上三角行列、 $U^H$  は  $U$  の転置、 $L$  は下三角である。

このルーチンはアルゴリズムの非ブロック化形式で、[BLAS レベル 2 のルーチン](#)を呼び出す。

### 入力パラメータ

<code>uplo</code>	CHARACTER*1。 対称 / エルミート行列 $A$ の上三角部分と下三角部分のどちらが格納されているかを指定する。 = 'U' の場合、上三角 = 'L' の場合、下三角
<code>n</code>	INTEGER。 行列 $A$ の次数。 $n \geq 0$
<code>kd</code>	INTEGER。 $uplo = 'U'$ の場合は行列 $A$ の優対角成分の個数、 $uplo = 'L'$ の場合は行列 $A$ の劣対角成分の個数。 $kd \geq 0$
<code>ab</code>	REAL (spbtf2 の場合 ) DOUBLE PRECISION (dpbtf2 の場合 ) COMPLEX (cpbtf2 の場合 )

COMPLEX\*16 (zpbtf2 の場合)

配列、次元は (*ldab*, *n*)。

対称 / エルミート 帯行列 *A* の上三角または下三角を、配列の最初の *kd*+1 行に格納する。*A* の *J* 番目の列は配列 *ab* の番目の列に次のように格納する。

*uplo* = 'u' の場合、 $\max(1, j-kd) \leq i \leq j$  に対して

$AB(kd+1+i-j, j) = A(i, j)$

*UPLO* = 'l' の場合、 $j \leq i \leq \min(v, j+kd)$  に対して  $AB(1+i-j, j) = A(i, j)$

*ldab*

INTEGER。

配列 *ab* のリーディング・ディメンジョン。  $ldab \geq kd+1$ 。

### 出力パラメータ

*ab*

*info* = 0 の場合、帯行列 *A* のコレスキー因子分解  $A = U^T U$  または  $A = L L^T$  で得られた三角係数 *U* または *L* が、*A* と同じ格納形式で格納される。

*info*

INTEGER。

= 0 の場合、正常に終了したことを示す。

< 0 の場合、*info* = -*k* は *k* 番目の引数の値が不正だったことを示す。

> 0 の場合、*info* = *k* は次数 *k* の先頭の小行列式が正定値でないため因子分解を完了できなかったことを示す。

---

## ?potf2

対称 / エルミート 正定値行列のコレスキー因子分解を行う ( 非ブロック化アルゴリズム ) 。

---

### 構文

```
call spotf2( uplo, n, a, lda, info )
```

```
call dpotf2( uplo, n, a, lda, info )
```

```
call cpotf2( uplo, n, a, lda, info )
```

```
call zpotf2( uplo, n, a, lda, info )
```

## 説明

ルーチン `?potf2` は、実対称 / 複素エルミート正定値行列  $A$  に対してコレスキー因子分解を行う。因子分解は以下の形式を持つ。

$A = U'U$ 、`uplo='u'` の場合、または

$A = LL'$ 、`uplo='l'` の場合

ここで  $U$  は上三角行列で、 $L$  は下三角である。

このルーチンはアルゴリズムの非ブロック化形式で、[BLAS レベル 2 のルーチン](#) を呼び出す。

## 入力パラメータ

<code>uplo</code>	CHARACTER*1。 対称 / エルミート行列 $A$ の上三角または下三角部分のどちらを格納するか指定する。 = 'u' の場合、上三角 = 'l' の場合、下三角。
<code>n</code>	INTEGER。 行列 $A$ の次数。 $n \geq 0$
<code>a</code>	REAL ( <code>spotf2</code> の場合) DOUBLE PRECISION ( <code>dpotf2</code> の場合) COMPLEX ( <code>cpotf2</code> の場合) COMPLEX*16 ( <code>zpotf2</code> の場合) 配列、次元は $(lda, n)$ 。 対称 / エルミート行列 $A$ を格納する。 <code>uplo='u'</code> の場合、 $a$ の先頭の $n \times n$ 上三角部分に行列 $A$ の上三角部分を格納する。 $a$ の厳密な下三角部分は参照されない。 <code>uplo='l'</code> の場合、 $a$ の先頭の $n \times n$ 下三角部分に行列 $A$ の下三角部分を格納する。 $a$ の厳密な上三角部分は参照されない。
<code>lda</code>	INTEGER。 配列 $a$ のリーディング・ディメンジョン。 $lda \geq \max(1, n)$ 。

## 出力パラメータ

<code>a</code>	<code>info=0</code> の場合、コレスキー因子分解 $A = U'U$ または $A = LL'$ により得られた係数 $U$ または $L$ で上書きされる。
----------------	--

*info* INTEGER。  
= 0 の場合、正常に終了したことを示す。  
< 0 の場合、*info* = -*k* は *k* 番目の引数の値が不正だったことを示す。  
> 0 の場合、*info* = *k* は次数 *k* の先頭の Minor 行列が正定値でないため  
因子分解を完了できなかったことを示す。

---

### ?ptts2

?pttrf で計算した  $L$ 、 $D$ 、 $L^H$  因子分解を用いて、  
形式  $AX = B$  の三重対角連立方程式を解く。

---

#### 構文

```
call sptts2( n, nrhs, d, e, b, ldb )  
call dptts2( n, nrhs, d, e, b, ldb )  
call cptts2( iuplo, n, nrhs, d, e, b, ldb )  
call zptts2( iuplo, n, nrhs, d, e, b, ldb )
```

#### 説明

ルーチン ?ptts2 は次の形式の三重対角連立方程式を解く。

$$AX = B$$

実数型の sptts2/dptts2 は [spttrf/dpttrf](#) で計算された  $A$  の  $LDL'$  因子分解を使用する。複素数型の cptts2/zptts2 は [cpttrf/zpttrf](#) で計算された  $A$  の  $U'DU$  または  $LDL'$  因子分解を使用する。

$D$  はベクトル  $d$  で指定される対角行列、 $U$  (または  $L$ ) はその優対角成分 (劣対角成分) がベクトル  $e$  で指定されている単位二重対角行列、 $X$  と  $B$  は  $n \times \text{nrhs}$  の行列である。

#### 入力パラメータ

*iuplo* INTEGER。複素型でのみ使用される。  
因子分解の形式と、ベクトル  $e$  が上二重単位対角係数  $U$  の優対角成分  
か下二重単位対角係数  $L$  の劣対角成分のどちらであるかを指定する。  
= 1 の場合、 $A = U'DU$ 、 $e$  は  $U$  の優対角成分。  
= 0 の場合、 $A = LDL'$ 、 $e$  は  $L$  の劣対角成分。

*n* INTEGER。  
三重対角行列  $A$  の次数。  $n \geq 0$

<i>nrhs</i>	INTEGER。 右辺の数、すなわち、行列 <i>B</i> の列数。 $nrhs \geq 0$ 。
<i>d</i>	REAL (sptts2/cptts2 の場合 ) DOUBLE PRECISION (dptts2/zptts2 の場合 ) 配列、次元は ( <i>n</i> )。 <i>A</i> の因子分解で得られる対角行列 <i>D</i> の <i>n</i> 対角成分
<i>e</i>	REAL (sptts2 の場合 ) DOUBLE PRECISION (dptts2 の場合 ) COMPLEX (cptts2 の場合 ) COMPLEX*16 (zptts2 の場合 ) 配列、次元は ( <i>n</i> -1)。 <i>A</i> の <i>LDL'</i> 因子分解から得られる単位二重対角係数 <i>L</i> の ( <i>n</i> -1) 個の劣対角成分を格納する ( 実数型、また <i>iuplo</i> = 0 の場合は複素型 )。 複素型において <i>iuplo</i> = 1 の場合、 <i>e</i> には因子分解 $A = U'DU$ から得られる単位二重対角係数 <i>U</i> の ( <i>n</i> -1) 個の優対角成分を格納する。
<i>b</i>	REAL (sptts2/cptts2 の場合 ) DOUBLE PRECISION (dptts2/zptts2 の場合 ) 配列、次元は ( <i>ldb</i> , <i>nrhs</i> )。 連立方程式に対するベクトル <i>B</i> の右辺。
<i>ldb</i>	INTEGER。 配列 <i>B</i> のリーディング・ディメンジョン。 $ldb \geq \max(1, n)$ 。

## 出力パラメータ

<i>b</i>	解ベクトル <i>X</i> で上書きされる。
----------	-------------------------

---

## ?rscl

ベクトルに実数スカラの逆数を乗算する。

---

### 構文

```
call srscl( n, sa, sx, incx )
call drscl( n, sa, sx, incx )
call csrscl( n, SA, SX, INCX )
call zdrscl( n, SA, SX, INCX )
```

### 説明

ルーチン `?rscl` は、 $n$  成分の実数 / 複素ベクトル  $x$  に実数スカラー  $1/a$  を乗算する。最終結果  $x/a$  がオーバーフローまたはアンダアフローを起こさない限り、演算はオーバーフローまたはアンダアフローを起こすことなく実行される。

### 入力パラメータ

$n$	INTEGER。 ベクトル $x$ の成分の個数。
$sa$	REAL( <code>srscl</code> / <code>csrscl</code> の場合 ) DOUBLE PRECISION ( <code>drsc1</code> / <code>zdrsc1</code> の場合 ) ベクトル $x$ の各成分の除算に使用されるスカラー $a$ 。 $sa$ は $\geq 0$ でなければならない。そうしないとサブルーチンでゼロの除算が発生する。
$sx$	REAL ( <code>srscl</code> の場合 ) DOUBLE PRECISION ( <code>drsc1</code> の場合 ) COMPLEX ( <code>csrscl</code> の場合 ) COMPLEX*16 ( <code>zdrsc1</code> の場合 ) 配列、次元は $(1+(n-1)*abs(incx))$ 。 ベクトル $x$ の $n$ 成分。
$incx$	INTEGER。 ベクトル $sx$ の連続する値の間の増加分。 $incx > 0$ の場合、 $sx(i) = x(i)$ 、 $1 < i \leq n$ に対して $sx(1+(i-1)*incx) = x(i)$

### 出力パラメータ

$sx$	結果 $x/a$ で上書きされる。
------	-------------------

---

## ?sygs2/?hegs2

`?potrf` で得られた因子分解の結果を用いて、  
対称 / エルミート汎用固有値問題を標準形式に  
縮退させる ( 非ブロック化アルゴリズム )。

---

### 構文

```
call ssygs2( itype, uplo, n, a, lda, b, ldb, info )
```

```
call dsygs2( itype, uplo, n, a, lda, b, ldb, info )
call chegs2( itype, uplo, n, a, lda, b, ldb, info )
call zhegs2( itype, uplo, n, a, lda, b, ldb, info )
```

## 説明

ルーチン `?sygs2/?hegs2` は、実数対称または複素エルミートの汎用固有値問題を標準化形式に縮退させる。

`itype = 1` の場合、問題は

$$Ax = \lambda Bx$$

となり、 $A$  は  $\text{inv}(U') * A * \text{inv}(U)$  または  $\text{inv}(L) * A * \text{inv}(L')$  で上書きされる。

`itype = 2` または `3` の場合、問題は

$$ABx = \lambda x \text{ または } B Ax = \lambda x$$

となり、 $A$  は  $UAU'$  または  $L'AL$  で上書きされる。 $B$  は、 $U'U$  または  $LL'$  として、[?potrf](#) によって事前に因子分解されていないといけない。

## 入力パラメータ

`itype`            INTEGER。  
                    $= 1$  の場合、 $\text{inv}(U') * A * \text{inv}(U)$  または  $\text{inv}(L) * A * \text{inv}(L')$  を計算する。  
                    $= 2$  または  $3$  の場合、 $UAU'$  または  $L'AL$  を計算する。

`uplo`            CHARACTER  
                   対称 / エルミート行列  $A$  の上三角部分または下三角部分のどちらが格納されているか、および  $B$  がどのように因子分解されたかを指定する。  
                    $= 'U'$  の場合、上三角  
                    $= 'L'$  の場合、下三角。

`n`                INTEGER。  
                   行列  $A$  と  $B$  の次数。  $n \geq 0$

`a`                REAL (ssygs2 の場合)  
                   DOUBLE PRECISION (dsygs2 の場合)  
                   COMPLEX (chegs2 の場合)  
                   COMPLEX\*16 (zhegs2 の場合)  
                   配列、次元は  $(lda, n)$ 。  
                   対称 / エルミート行列  $A$  を格納する。  
                   `uplo = 'U'` の場合、 $A$  の先頭の  $n \times n$  上三角部分に行列  $a$  の上三角部分を格納する。 $a$  の厳密な下三角部分は参照されない。`uplo = 'L'` の場合、 $a$  の先頭の  $n \times n$  下三角部分に行列  $A$  の下三角部分を格納する。 $a$  の厳密な上三角部分は参照されない。

<i>lda</i>	INTEGER。 配列 <i>a</i> のリーディング・ディメンジョン。 $lda \geq \max(1,n)$ 。
<i>b</i>	REAL (ssygs2 の場合 ) DOUBLE PRECISION (dsygs2 の場合 ) COMPLEX (chegs2 の場合 ) COMPLEX*16 (zhegs2 の場合 ) 配列、次元は ( <i>ldb</i> , <i>n</i> )。 ?potrf によって返された、 <i>B</i> のコレスキー因子分解で得られた三角係数。
<i>ldb</i>	INTEGER。 配列 <i>B</i> のリーディング・ディメンジョン。 $ldb \geq \max(1,n)$ 。

### 出力パラメータ

<i>a</i>	<i>info</i> = 0 の場合、変換後の行列が <i>A</i> と同じ形式で格納される。
<i>info</i>	INTEGER。 = 0 の場合、正常に終了したことを示す。 < 0 の場合、 <i>info</i> = - <i>i</i> は <i>i</i> 番目の引数の値が不正だったことを示す。

---

## ?sytd2/?hetd2

直交/ユニタリ相似変換を用いて、対称/エルミート行列を実数対称三重対角形式に縮退させる  
(非ブロック化アルゴリズム)。

---

### 構文

```
call ssytd2( uplo, n, a, lda, d, e, tau, info )
call dsytd2( uplo, n, a, lda, d, e, tau, info )
call chetd2( uplo, n, a, lda, d, e, tau, info )
call zhetd2( uplo, n, a, lda, d, e, tau, info )
```

### 説明

ルーチン ?sytd2/?hetd2 は、直交/ユニタリ相似変換  $Q' A Q = T$  を用いて、実数対称/複素エルミート行列 *A* を実数対称三重対角形式 *T* に縮退させる。



## 入力パラメータ

<code>uplo</code>	CHARACTER*1。 対称 / エルミート行列 $A$ の上三角部分と下三角部分のどちらが格納されているかを指定する。 = 'U' の場合、上三角 = 'L' の場合、下三角
<code>n</code>	INTEGER。 行列 $A$ の次数。 $n \geq 0$
<code>a</code>	REAL (ssytd2 の場合) DOUBLE PRECISION (dsytd2 の場合) COMPLEX (chetd2 の場合) COMPLEX*16 (zhetd2 の場合) 配列、次元は $(lda, n)$ 。 対称 / エルミート行列 $A$ を格納する。 $uplo = 'U'$ の場合、 $A$ の先頭の $n \times n$ 上三角部分に行列 $a$ の上三角部分を格納する。 $a$ の厳密な下三角部分は参照されない。 $uplo = 'L'$ の場合、 $a$ の先頭の $n \times n$ 下三角部分に行列 $A$ の下三角部分を格納する。 $a$ の厳密な上三角部分は参照されない。
<code>lda</code>	INTEGER。 配列 $a$ のリーディング・ディメンジョン。 $lda \geq \max(1, n)$ 。

## 出力パラメータ

<code>a</code>	$uplo = 'U'$ の場合、 $a$ の対角成分と最初の優対角成分は三重対角行列 $T$ の対応する成分で上書きされる。最初の優対角成分より上の成分は、配列 <code>tau</code> とともに、基本リフレクタの積として直交 / ユニタリ行列 $Q$ を表現する。 $uplo = 'L'$ の場合、 $a$ の対角成分と最初の劣対角成分は三重対角行列 $T$ の対応する成分で上書きされる。最初の劣対角成分より下の成分は、配列 <code>tau</code> とともに、基本リフレクタの積として直交 / ユニタリ行列 $Q$ を表現する。
<code>d</code>	REAL (ssytd2/chetd2 の場合) DOUBLE PRECISION (dsytd2/zhetd2 の場合) 配列、次元は $(n)$ 。 三重対角行列 $T$ の対角成分。 $d(i) = a(i, i)$
<code>e</code>	REAL (ssytd2/chetd2 の場合) DOUBLE PRECISION (dsytd2/zhetd2 の場合) 配列、次元は $(n-1)$ 。

三重対角行列  $T$  の非対角成分。  
 $e(i) = a(i, i+1)$  ( $uplo = 'U'$  の場合)  
 $e(i) = a(i+1, i)$  ( $uplo = 'L'$  の場合)  
  
 $tau$             REAL (ssytd2 の場合)  
                DOUBLE PRECISION (dsytd2 の場合)  
                COMPLEX (chetd2 の場合)  
                COMPLEX\*16 (zhetd2 の場合)  
                配列、次元は  $(n-1)$ 。  
                基本リフレクタのスカラー係数。  
  
 $info$             INTEGER。  
                 $= 0$  の場合、正常に終了したことを示す。  
                 $< 0$  の場合、 $info = -i$  は  $i$  番目の引数の値が不正だったことを示す。

---

### ?sytf2

対角ピボット演算法を使用して、実数 / 複素不定値対称行列の因子分解を計算する ( 非ブロック化アルゴリズム )。

---

#### 構文

```
call ssytf2( uplo, n, a, lda, ipiv, info )  
call dsytf2( uplo, n, a, lda, ipiv, info )  
call csytf2( uplo, n, a, lda, ipiv, info )  
call zsytf2( uplo, n, a, lda, ipiv, info )
```

#### 説明

ルーチン ?sytf2 は、Bunch-Kaufman 対角ピボット演算法を使用して、実数 / 複素不定値対称行列  $A$  の因子分解を計算する。

$$A = U D U' \text{ または } A = L D L'$$

$U$  (または  $L$ ) は置換行列と単位上 (下) 三角行列の積、 $U'$  は  $U$  の転置、 $D$  は  $1 \times 1$  と  $2 \times 2$  の対角ブロックを持つ対称ブロック対角行列である。

このルーチンはアルゴリズムの非ブロック化形式で、[BLAS レベル 2 のルーチン](#) を呼び出す。

## 入力パラメータ

<code>uplo</code>	CHARACTER*1。 対称行列 $A$ の上三角部分または下三角部分のどちらが格納されているかを指定する。 = 'U' の場合、上三角 = 'L' の場合、下三角
<code>n</code>	INTEGER。 行列 $A$ の次数。 $n \geq 0$
<code>a</code>	REAL (ssytf2 の場合) DOUBLE PRECISION (dsytf2 の場合) COMPLEX (csytf2 の場合) COMPLEX*16 (zsytf2 の場合) 配列、次元は $(lda, n)$ 。 対称行列 $A$ を格納する。 $uplo = 'U'$ の場合、 $A$ の先頭の $n \times n$ 上三角部分に行列 $a$ の上三角部分を格納する。 $a$ の厳密な下三角部分は参照されない。 $uplo = 'L'$ の場合、 $a$ の先頭の $n \times n$ 下三角部分に行列 $A$ の下三角部分を格納する。 $a$ の厳密な上三角部分は参照されない。
<code>lda</code>	INTEGER。 配列 $a$ のリーディング・ディメンジョン。 $lda \geq \max(1, n)$ 。

## 出力パラメータ

<code>a</code>	ブロック対角行列 $D$ と、係数 $U$ または $L$ を得るために使用された乗数が上書きされる。
<code>ipiv</code>	INTEGER。 配列、次元は $(n)$ 。 交換の結果と $D$ のブロック構造の各成分が格納される。 $ipiv(k) > 0$ の場合、 $k$ 番目の行と列と $ipiv(k)$ 番目の行と列は交換された。 $D(k, k)$ は $1 \times 1$ の対角ブロックである。 $uplo = 'U'$ かつ $ipiv(k) = ipiv(k-1) < 0$ の場合、 $k-1$ 番目の行と列と $-ipiv(k)$ 番目の行と列は交換された。 $D(k-1:k, k-1:k)$ は $2 \times 2$ の対角ブロックである。 $uplo = 'L'$ かつ $ipiv(k) = ipiv(k+1) < 0$ の場合、 $k+1$ 番目の行と列と $-ipiv(k)$ 番目の行と列は交換された。 $D(k:k+1, k:k+1)$ は $2 \times 2$ の対角ブロックである。

*info* INTEGER。  
= 0 の場合、正常に終了したことを示す。  
< 0 の場合、*info* = -*k* は *k* 番目の引数の値が不正だったことを示す。  
> 0 の場合、*info* = *k* は  $D(k, k)$  が完全に 0 であることを示す。因子分解は完了したが、ブロック対角行列  $D$  は完全に特異で、連立方程式の解の算出に  $D$  を使用すると 0 での除算が発生する。

---

## ?hetf2

対角ピボット演算法を使用して、複素エルミート行列の因子分解を計算する ( 非ブロック化アルゴリズム )。

---

### 構文

```
call chetf2( uplo, n, a, lda, ipiv, info )  
call zhetf2( uplo, n, a, lda, ipiv, info )
```

### 説明

このルーチンは、Bunch-Kaufman 対角ピボット演算法を使用して、複素エルミート行列  $A$  の因子分解を計算する。

$$A = U D U' \text{ または } A = L D L'$$

$U$  ( または  $L$  ) は置換行列と単位上 ( 下 ) 三角行列の積、 $U'$  は  $U$  の共役転置、 $D$  は  $1 \times 1$  と  $2 \times 2$  の対角ブロックを持つエルミート・ブロック対角行列である。

このルーチンはアルゴリズムの非ブロック化形式で、[BLAS レベル 2 のルーチン](#)を呼び出す。

### 入力パラメータ

*uplo* CHARACTER\*1。  
エルミート行列  $A$  の上三角部分と下三角部分のどちらが格納されているかを指定する。  
= 'U' の場合、上三角  
= 'L' の場合、下三角

*n* INTEGER。  
行列  $A$  の次数。  $n \geq 0$

<i>a</i>	COMPLEX (chetf2 の場合) COMPLEX*16 (zhetf2 の場合) 配列、次元は ( <i>lda</i> , <i>n</i> )。 エルミート行列 <i>A</i> を格納する。 <i>uplo</i> = 'U' の場合、 <i>a</i> の先頭の $n \times n$ 上三角部分に行列 <i>A</i> の上三角部分を格納する。 <i>a</i> の厳密な下三角部分は参照されない。 <i>uplo</i> = 'L' の場合、 <i>a</i> の先頭の $n \times n$ 下三角部分に行列 <i>A</i> の下三角部分を格納する。 <i>a</i> の厳密な上三角部分は参照されない。
<i>lda</i>	INTEGER。 配列 <i>a</i> のリーディング・ディメンジョン。 $lda \geq \max(1, n)$ 。

### 出力パラメータ

<i>a</i>	ブロック対角行列 <i>D</i> と、係数 <i>U</i> または <i>L</i> を得るために使用された乗数が上書きされる。
<i>ipiv</i>	INTEGER。 配列、次元は ( <i>n</i> )。 交換の結果と <i>D</i> のブロック構造の各成分が格納される。 <i>ipiv</i> ( <i>k</i> ) > 0 の場合、 <i>k</i> 番目の行と列と <i>ipiv</i> ( <i>k</i> ) 番目の行と列は交換された。 <i>D</i> ( <i>k</i> , <i>k</i> ) は $1 \times 1$ の対角ブロックである。 <i>uplo</i> = 'U' かつ <i>ipiv</i> ( <i>k</i> ) = <i>ipiv</i> ( <i>k</i> -1) < 0 の場合、 <i>k</i> -1 番目の行と列と - <i>ipiv</i> ( <i>k</i> ) 番目の行と列は交換された。 <i>D</i> ( <i>k</i> -1: <i>k</i> , <i>k</i> -1: <i>k</i> ) は $2 \times 2$ の対角ブロックである。 <i>uplo</i> = 'L' かつ <i>ipiv</i> ( <i>k</i> ) = <i>ipiv</i> ( <i>k</i> +1) < 0 の場合、 <i>k</i> +1 番目の行と列と - <i>ipiv</i> ( <i>k</i> ) 番目の行と列は交換された。 <i>D</i> ( <i>k</i> : <i>k</i> +1, <i>k</i> : <i>k</i> +1) は $2 \times 2$ の対角ブロックである。
<i>info</i>	INTEGER。 = 0 の場合、正常に終了したことを示す。 < 0 の場合、 <i>info</i> = - <i>k</i> は <i>k</i> 番目の引数の値が不正だったことを示す。 > 0 の場合、 <i>info</i> = <i>k</i> は <i>D</i> ( <i>k</i> , <i>k</i> ) が完全に 0 であることを示す。因子分解は完了したが、ブロック対角行列 <i>D</i> は完全に特異で、連立方程式の解の算出に <i>D</i> を使用すると 0 での除算が発生する。

## ?tgex2

直交/ユニタリ等価変換を使用して、( 準 ) 上三角行列のペア中の隣接対角ブロックを交換する。

### 構文

```
call stgex2( wantq, wantz, n, a, lda, b, ldb, q, ldq, z, ldz, j1, n1, n2, work,
            lwork, info )
call dtgex2( wantq, wantz, n, a, lda, b, ldb, q, ldq, z, ldz, j1, n1, n2, work,
            lwork, info )
call ctgex2( wantq, wantz, n, a, lda, b, ldb, q, ldq, z, ldz, j1, info )
call ztgex2( wantq, wantz, n, a, lda, b, ldb, q, ldq, z, ldz, j1, info )
```

### 説明

実数ルーチン stgex2/dtgex2 は、直交等価変換を用いて、( 準 ) 上三角行列のペア ( $A$ ,  $B$ ) の中の  $1 \times 1$  の隣接対角ブロック ( $A_{11}$ ,  $B_{11}$ ) と  $2 \times 2$  の隣接対角ブロック ( $A_{22}$ ,  $B_{22}$ ) を交換する。( $A$ ,  $B$ ) は汎用実数 Schur 標準形 (sgges/dgges により返されたとおり) でなければならない。すなわち、 $A$  は  $1 \times 1$  と  $2 \times 2$  の対角ブロックを持つ上三角ブロックである。 $B$  は上三角である。

複素ルーチン ctgex2/ztgex2 は、ユニタリ等価変換を用いて、上三角行列のペア ( $A$ ,  $B$ ) の中の  $1 \times 1$  の隣接対角ブロック ( $A_{11}$ ,  $B_{11}$ ) と  $2 \times 2$  の隣接対角ブロック ( $A_{22}$ ,  $B_{22}$ ) を交換する。( $A$ ,  $B$ ) は汎用 Schur 標準形でなければならない。すなわち、 $A$  と  $B$  はどちらも上三角である。

すべてのルーチンは、汎用 Schur ベクトルの  $Q$  と  $Z$  をオプションで更新する。

$$Q(\text{in}) * A(\text{in}) * Z(\text{in})' = Q(\text{out}) * A(\text{out}) * Z(\text{out})'$$

$$Q(\text{in}) * B(\text{in}) * Z(\text{in})' = Q(\text{out}) * B(\text{out}) * Z(\text{out})'$$

### 入力パラメータ

wantq	LOGICAL。 wantq = .TRUE. の場合、左変換行列 $Q$ を更新する。 wantq = .FALSE. の場合、 $Q$ を更新しない。
wantz	LOGICAL。 wantz = .TRUE. の場合、右変換行列 $Z$ を更新する。 wantz = .FALSE. の場合、 $Z$ を更新しない。

<i>n</i>	INTEGER。 行列 <i>A</i> と <i>B</i> の次数。 $n \geq 0$
<i>a</i> , <i>b</i>	REAL (stgex2 の場合) DOUBLE PRECISION (dtgex2 の場合) COMPLEX (ctgex2 の場合) COMPLEX*16 (ztgex2 の場合) 配列、次元はそれぞれ、( <i>lda</i> , <i>n</i> ) および ( <i>ldb</i> , <i>n</i> ) ペア ( <i>A</i> , <i>B</i> ) となっている行列 <i>A</i> と <i>B</i> を格納する。
<i>lda</i>	INTEGER。 配列 <i>a</i> のリーディング・ディメンジョン。 $lda \geq \max(1, n)$ 。
<i>ldb</i>	INTEGER。 配列 <i>b</i> のリーディング・ディメンジョン。 $ldb \geq \max(1, n)$ 。
<i>q</i> , <i>z</i>	REAL (stgex2 の場合) DOUBLE PRECISION (dtgex2 の場合) COMPLEX (ctgex2 の場合) COMPLEX*16 (ztgex2 の場合) 配列、次元はそれぞれ、( <i>ldq</i> , <i>n</i> ) および ( <i>ldz</i> , <i>n</i> ) <i>wantq</i> = .TRUE. の場合は <i>q</i> に直交 / ユニタリ行列 <i>Q</i> を格納し、 <i>wantz</i> = .TRUE. の場合は <i>z</i> に直交 / ユニタリ行列 <i>Z</i> を格納する。
<i>ldq</i>	INTEGER。 配列 <i>q</i> のリーディング・ディメンジョン。 $ldq \geq 1$ <i>wantq</i> = .TRUE. の場合、 $ldq \geq n$
<i>ldz</i>	INTEGER。 配列 <i>z</i> のリーディング・ディメンジョン。 $ldz \geq 1$ <i>wantz</i> = .TRUE. の場合、 $ldz \geq n$
<i>j1</i>	INTEGER 最初のブロック ( <i>A11</i> , <i>B11</i> ) へのインデックス。 $1 \leq j1 \leq n$
<i>n1</i>	INTEGER。実数型でのみ使用される。 最初のブロック ( <i>A11</i> , <i>B11</i> ) の次数。 $n1 = 0, 1$ または 2
<i>n2</i>	INTEGER。実数型でのみ使用される。 2 番目のブロック ( <i>A22</i> , <i>B22</i> ) の次数。 $n2 = 0, 1$ または 2
<i>work</i>	REAL (stgex2 の場合) DOUBLE PRECISION (dtgex2 の場合) ワークスペース配列、次元は ( <i>lwork</i> ) 実数型でのみ使用される。

*lwork*                    INTEGER  
配列 *work* の次元。  
 $lwork \geq \max( n*(n2+n1), 2*(n2+n1)^2 )$

### 出力パラメータ

*a*                    更新された行列 *A* で上書きされる。

*b*                    更新された行列 *B* で上書きされる。

*q*                    更新された行列 *Q* で上書きされる。  
*wantq* = .FALSE. の場合は参照されない。

*z*                    更新された行列 *Z* で上書きされる。  
*wantz* = .FALSE. の場合は参照されない。

*info*                INTEGER。  
= 0 の場合、正常に終了したことを示す。  
*stgex2/dtgex2* の場合。  
*info* = 1 の場合、変換された行列 (*A*, *B*) が汎用 Schur 形式からかけ離れていることを示す。ブロックは交換されず、(*A*, *B*) と (*Q*, *Z*) は変更されない。すなわち、この問題は悪条件である。*info* = -16 の場合、*lwork* が小さすぎることを示す。*lwork* の適切な値は *work*(1) に返される。

*ctgex2/ztgex2* の場合。  
*info* = 1 の場合、変換後の行列ペア (*A*, *B*) が汎用 Schur 形式から遠すぎる。すなわち、この問題は悪条件である。(*A*, *B*) は部分的に順序が変更されている可能性があり、*ilst* は移動対象ブロックの現在の位置の最初の行を指している。

---

## ?tgsy2

汎用シルベスター式を解く ( 非ブロック化アルゴリズム )。

---

### 構文

```
call stgsy2( trans, ijob, m, n, a, lda, b, ldb, c, ldc, d, ldd, e, lde, f, ldf,  
            scale, rdsum, rdscal, iwork, pq, info )  
call dtgsy2( trans, ijob, m, n, a, lda, b, ldb, c, ldc, d, ldd, e, lde, f, ldf,  
            scale, rdsum, rdscal, iwork, pq, info )
```



```
call ctgsy2( trans, ijob, m, n, a, lda, b, ldb, c, ldc, d, ldd, e, lde, f, ldf,
            scale, rdsum, rdscal, iwork, pq, info )
call ztgsy2( trans, ijob, m, n, a, lda, b, ldb, c, ldc, d, ldd, e, lde, f, ldf,
            scale, rdsum, rdscal, iwork, pq, info )
```

## 説明

ルーチン ?tgsy2 は、レベル 1 BLAS とレベル 2 BLAS を使用して、次の汎用シルベスター式を解く。

$$\begin{aligned} AR - LB &= \text{scale} * C \\ DR - LE &= \text{scale} * F, \end{aligned} \quad (1)$$

$R$  と  $L$  は未知の  $m \times n$  行列、 $(A, D)$ 、 $(B, E)$ 、 $(C, F)$  はそれぞれ  $m \times m$ 、 $n \times n$ 、 $m \times n$  の与えられた行列のペアである。

stgsy2/dtgsy2 の場合、ペア  $(A, D)$  と  $(B, E)$  は汎用 Schur 標準形でなければならない。すなわち、 $A$  と  $B$  は上準三角行列で、 $D$  と  $E$  は上三角行列でなければならない。

ctgsy2/ztgsy2 の場合、行列  $A$  と  $B$  と  $D$  と  $E$  は上三角行列である (すなわち、 $(A, D)$  と  $(B, E)$  は汎用 Schur 形)。

$(C, F)$  は解  $(R, L)$  で上書きされる。 $0 \leq \text{scale} \leq 1$  はオーバーフローを避けるために選択された出力スケール係数である。

行列表記では、方程式 (1) を解くことは次式を解くことに対応する。

$$Zx = \text{scale} * b$$

ただし、 $Z$  は以下のように定義される。

$$Z = \begin{bmatrix} \text{kron}(I_n, A) & -\text{kron}(B', I_m) \\ \text{kron}(I_n, D) & -\text{kron}(E', I_m) \end{bmatrix} \quad (2)$$

$I_k$  はサイズが  $k$  の単位行列、 $X'$  は  $X$  の転置である。

$\text{kron}(X, Y)$  は行列  $X$  と  $Y$  の Kronecker 積を表わす。

$\text{trans} = 'T'$  の場合、次の転置 (共役転置) 式を  $y$  について解く。

$$Zy = \text{scale} * b$$

これは、以下の式を  $R, L$  に対して解くことと等価である。

$$\begin{aligned} A' R + D' L &= \text{scale} * C \\ R B' + L E' &= \text{scale} * (-F) \end{aligned} \quad (3)$$

上の事例は、[?lacon](#) による逆コミュニケーションを使用した、 $\text{Dif}[(A, D), (B, E)] = \text{sigma\_min}(Z)$  の推定の計算に用いられる。

?tgsy2 はまた ( $ijob \geq 1$  に対して)、?tgsy1 における 2 つの行列ペア間の隔たりの上限計算に影響する。入力 ( $A, D$ )、( $B, e$ ) は、?tgsy1 において行列ペアの部分行列束である。詳細は [?tgsy1](#) を参照のこと。

## 入力パラメータ

<i>trans</i>	CHARACTER $trans = 'N'$ の場合、汎用シルベスター方程式 (1) を解く。 $trans = 'T'$ の場合、転置された式 (3) を解く。
<i>ijob</i>	INTEGER。 実行する機能を指定する。 $ijob = 0$ の場合、(1) のみを解く。 $ijob = 1$ の場合、この部分式を始点として、2 個の行列ペア間の隔たりの Frobenius ノルムにもとづく推定までの影響が計算される (先読み法が用いられる)。 $ijob = 2$ の場合、この部分式を始点として、2 個の行列ペア間の隔たりの Frobenius ノルムにもとづく推定までの影響が計算される (部分式に対して ?gecon が用いられる)。 $trans = 'T'$ の場合は参照されない。
<i>m</i>	INTEGER。 $m$ は、 $A$ と $D$ の次数、および $C, F, R, L$ の行次元を指定する。
<i>n</i>	INTEGER。 $n$ は、 $B$ と $E$ の次数、および $C, F, R, L$ の列次元を指定する。
<i>a, b</i>	REAL (stgsy2 の場合) DOUBLE PRECISION (dtgsy2 の場合) COMPLEX (ctgsy2 の場合) COMPLEX*16 (ztgsy2 の場合) 配列、次元はそれぞれ、( $lda, m$ ) および ( $ldb, n$ )。 $a$ には上 (準) 三角行列 $A$ を、 $b$ には上 (準) 三角行列 $B$ を格納する。
<i>lda</i>	INTEGER。 配列 $a$ のリーディング・ディメンジョン。 $lda \geq \max(1, m)$ 。
<i>ldb</i>	INTEGER。 配列 $b$ のリーディング・ディメンジョン。 $ldb \geq \max(1, n)$ 。
<i>c, f</i>	REAL (stgsy2 の場合) DOUBLE PRECISION (dtgsy2 の場合) COMPLEX (ctgsy2 の場合) COMPLEX*16 (ztgsy2 の場合)

配列、次元はそれぞれ、 $(l_{dc}, n)$  および  $(l_{df}, n)$ 。 $c$  には (1) の最初の行列方程式の右辺を、 $f$  には (1) の 2 番目の行列方程式の右辺を格納する。

$l_{dc}$	INTEGER。 配列 $c$ のリーディング・ディメンジョン。 $l_{dc} \geq \max(1, m)$ 。
$d, e$	REAL (stgsy2 の場合 ) DOUBLE PRECISION (dtgsy2 の場合 ) COMPLEX (ctgsy2 の場合 ) COMPLEX*16 (ztgsy2 の場合 ) 配列、次元はそれぞれ、 $(l_{dd}, m)$ および $(l_{de}, n)$ 。 $d$ には上三角行列 $D$ を、 $e$ には上三角行列 $E$ を格納する。
$l_{dd}$	INTEGER。 配列 $d$ のリーディング・ディメンジョン。 $l_{dd} \geq \max(1, m)$ 。
$l_{de}$	INTEGER。 配列 $e$ のリーディング・ディメンジョン。 $l_{de} \geq \max(1, n)$ 。
$l_{df}$	INTEGER。 配列 $f$ のリーディング・ディメンジョン。 $l_{df} \geq \max(1, m)$ 。
$rdsum$	REAL (stgsy2/ctgsy2 の場合 ) DOUBLE PRECISION (dtgsy2/ztgsy2 の場合 ) ?tgsy1 で得られた Dif 推定に対する、計算された影響の二乗和を格納する。ここでスケール係数 $rdscal$ は因子分解されている。
$rdscal$	REAL (stgsy2/ctgsy2 の場合 ) DOUBLE PRECISION (dtgsy2/ztgsy2 の場合 ) $rdsum$ のオーバーフローを防ぐために使用されるスケール係数を格納する。
$iwork$	INTEGER。実数型でのみ使用される。 ワークスペース配列、次元は $(m+n+2)$

### 出力パラメータ

$c$	$ijob = 0$ の場合、 $c$ は解 $R$ で上書きされる。
$f$	$ijob = 0$ の場合、 $f$ は解 $L$ で上書きされる。
$scale$	REAL (stgsy2/ctgsy2 の場合 ) DOUBLE PRECISION (dtgsy2/ztgsy2 の場合 ) $0 \leq scale \leq 1$ が格納される。 $0 < scale < 1$ の場合、解 $R$ と解 $L$ (入力

	時は $C$ と $F$ ) にはそれぞれ、わずかに摂動する系の解が入るが、入力行列 $A$ 、 $B$ 、 $D$ 、 $E$ は変更されない。 $scale=0$ の場合、 $R$ と $L$ にはそれぞれ $C=F=0$ の均一な系の解が入る。通常は $scale=1$ である。
<i>rdsum</i>	対応する二乗和は、現在の部分式から得られた影響によって更新される。 <i>trans</i> = 'T' の場合、 <i>rdsum</i> は使われない。 <i>rdsum</i> は ?tgsy2 が ?tgsy1 から呼び出されたときのみ意味を持つことに注意する。
<i>rdscal</i>	<i>rdscal</i> は、 <i>rdsum</i> に格納されている現在の影響に関して更新される。 <i>trans</i> = 'T' の場合、 <i>rdsum</i> は使われない。 <i>rdscal</i> は ?tgsy2 が ?tgsy1 から呼び出されたときのみ意味を持つことに注意する。
<i>pq</i>	INTEGER。実数型でのみ使用される。 ルーチン <i>stgsy2/dtgsy2</i> で解かれた部分式 ( $2 \times 2$ 、 $4 \times 4$ 、 $8 \times 8$ のサイズ) の個数が格納される。
<i>info</i>	INTEGER。 <i>info</i> には次の値が設定される。 0 の場合、正常に終了したことを示す。 <0 の場合、 <i>info</i> = - <i>i</i> は <i>i</i> 番目の引数の値が不正だったことを示す。 >0 の場合、行列のペア ( $A, D$ ) と ( $B, E$ ) は、同じ固有値か、きわめて近い固有値を持つ。

---

## ?trti2

三角行列の逆行列を計算する ( 非ブロック化アルゴリズム )。

---

### 構文

```
call strti2( uplo, diag, n, a, lda, info )
call dtrti2( uplo, diag, n, a, lda, info )
call ctrti2( uplo, diag, n, a, lda, info )
call ztrti2( uplo, diag, n, a, lda, info )
```

## 説明

ルーチン `?trti2` は実数 / 複素の上三角または下三角行列の逆行列を計算する。

このルーチンは、アルゴリズムの Level 2 BLAS バージョンである。

## 入力パラメータ

<code>uplo</code>	CHARACTER*1。 行列 $A$ が上三角か下三角かを指定する。 = 'U' の場合、上三角 = 'L' の場合、下三角。
<code>diag</code>	CHARACTER*1。 行列 $A$ が単位三角かどうかを指定する。 = 'N' の場合、単位三角ではない。 = 'U' の場合、単位三角。
<code>n</code>	INTEGER。 行列 $A$ の次数。 $n \geq 0$
<code>a</code>	REAL( <code>strti2</code> の場合) DOUBLE PRECISION( <code>dtrti2</code> の場合) COMPLEX( <code>ctrti2</code> の場合) COMPLEX*16( <code>ztrti2</code> の場合) 配列、次元は $(lda, n)$ 。 三角行列 $A$ を格納する。 <code>uplo</code> = 'U' の場合、配列 $a$ の先頭の $n \times n$ 上三角部分に上三角行列を格納する。厳密な下三角部分は参照されない。 <code>uplo</code> = 'L' の場合、配列 $a$ の先頭の $n \times n$ 下三角部分に下三角行列を格納する。厳密な上三角部は参照されない。 <code>diag</code> = 'U' の場合も対角成分は参照されず 1 とみなされる。
<code>lda</code>	INTEGER。 配列 $a$ のリーディング・ディメンジョン。 $lda \geq \max(1, n)$ 。

## 出力パラメータ

<code>a</code>	終了時に、元の行列の (三角) 逆行列で、同じ格納形式で上書きされる。
<code>info</code>	INTEGER。 = 0 の場合、正常に終了したことを示す。 < 0 の場合、 <code>info</code> = $-k$ は $k$ 番目の引数の値が不正だったことを示す。

ユーティリティ関数とルーチン

このセクションでは、LAPACK ユーティリティ関数とルーチンについて説明する。次の表はこれらのルーチンについてまとめたものである。

表 5-2 LAPACK ユーティリティ・ルーチン

ルーチン名	データ型	説明
<a href="#">ilaenv</a>		アルゴリズム性能のチューニング値を戻す環境問い合わせ関数。
<a href="#">ieeeck</a>		無限大演算および NaN 演算が安全かどうかを確認する。ilaenv により呼び出される。
<a href="#">lsame</a>		2 つの文字の同一性を大文字 / 小文字に関わらずテストする。
<a href="#">lsamen</a>		2 つの文字列の同一性を大文字 / 小文字に関わらずテストする。
<a href="#">?labad</a>	s,d	指数範囲がきわめて大きい場合にアンダーフローおよびオーバーフローしきい値の平方根を返す。
<a href="#">?lamch</a>	s,d	浮動小数点演算に対するマシン・パラメータを決定する。
<a href="#">?lamc1</a>	s,d	?lamc2 から呼び出される。beta, t, rnd, ieee1 で与えられるマシン・パラメータを決定する。
<a href="#">?lamc2</a>	s,d	?lamch で使用される。引数リストで指定されたマシン・パラメータを決定する。
<a href="#">?lamc3</a>	s,d	?lamc1 ~ ?lamc5 から呼び出される。a と b の加算を実行する前に a と b を格納させる。
<a href="#">?lamc4</a>	s,d	このルーチンは ?lamc2 のサービスルーチンである。
<a href="#">?lamc5</a>	s,d	?lamc2 から呼び出される。オーバーフローしないマシンの最大浮動小数点値の計算を試みる。
<a href="#">second/d secnd</a>		プロセスのユーザ時間を返す。
<a href="#">xerbla</a>		LAPACK ルーチンから呼び出されるエラー処理ルーチン。

ilaenv

アルゴリズム性能のチューニング値を戻す  
環境問い合わせ関数。

構文

value = ilaenv( ispec, name, opts, n1, n2, n3, n4 )

## 説明

問い合わせ関数 `ilaenv` はローカル環境に対する問題依存パラメータを選択するために LAPACK ルーチンから呼び出される。パラメータの説明は `ispec` を参照のこと。

このバージョンは、現在の多くのコンピュータ上で最適ではないが、良好な性能を得る一連のパラメータを与える。ユーザは、引数中のオプションと問題サイズ情報を使って、使用対象のマシンに対するチューニング・パラメータを設定するために、このサブルーチンを変更することが奨励される。

このルーチンはすべて小文字にすると正しく機能しない。すべて大文字で使用することは許される。

## 入力パラメータ

`ispec`                    INTEGER。 `ilaenv` の戻り値として得たいパラメータを指定する。

= 1: 最適ブロックサイズ。戻り値が 1 のとき、非ブロック化アルゴリズムで最適な性能が得られる。

= 2: ブロックルーチンを使用すべき最小ブロックサイズ。使用できるブロックサイズがこの値未満の場合は、非ブロック化ルーチンを使用すべきである。

= 3: クロスオーバー点 (ブロックルーチンでは、この値未満の  $N$  に対して、非ブロック化ルーチンを使用すべきである)。

= 4: シフト数。非対称の固有値ルーチンで使用される。

= 5: ブロック化が使用される最小の列次元。矩形ブロックは  $k \times m$  以上の次元を持っていなければならない、ここで  $k$  は `ilaenv(2, ...)` で与えられ、 $m$  は `ilaenv(5, ...)` で与えられる。

= 6: SVD に対するクロスオーバー点 ( $m \times n$  の行列を二重対角形式に縮退する場合、 $\max(m, n)/\min(m, n)$  がこの値を超えたなら、行列を三角形式に縮退するために最初に  $QR$  因子分解が使用される)。

= 7: プロセッサの個数。

= 8: 非対称固有値問題に対するマルチ。シフト  $QR$  および  $QZ$  法のクロスオーバー点

= 9: 分割統治アルゴリズムにおける計算ツリーの一番下にある部分問題の最大サイズ (`?gelsd` と `?gesdd` で使用する)。

= 10: トラップを生じないと信頼される IEEE NaN 演算。

= 11: トラップを生じないと信頼される無限大演算。

*name* CHARACTER\*(\*)。呼び出しサブルーチンの名称で、大文字でも小文字でもよい。

*opts* CHARACTER\*(\*)。サブルーチン *name* に対する文字オプションを単一文字列に連結したもの。例えば、*uplo* = 'U'、*trans* = 'T'、*diag* = 'N' の三角ルーチンは *opts* = 'UTN' として指定してもよい。

*n1*, *n2*, *n3*, *n4* INTEGER。サブルーチン *name* に対する問題の次元。すべてのパラメータが必要でない場合もある。

### 出力パラメータ

*value* INTEGER。  
*value* ≥ 0 の場合、*ispec* で指定されたパラメータ値が返される。  
*value* = -*k* < 0 の場合、*k* 番目の引数が不正な値であることを示す。

### アプリケーション・ノート

LAPACK ルーチンから *ilaenv* を呼び出すときには以下の規則が使用される。

- 1) *opts* は、*ispec* で指定したパラメータ値を求めるために使用しない場合でも、*name* の引数リストと同じ順番でサブルーチン *name* のすべての文字オプションを連結したものである。
- 2) 問題次元 *n1*, *n2*, *n3*, *n4* は *name* の引数リストと同じ順番で指定する。*n1* が最初に使用され、次に *n2* が使われ、以下同様に続く。使用されない問題次元には値 -1 が渡される。
- 3) *ilaenv* が戻すパラメータ値は呼び出しサブルーチン内で有効性をチェックする。例えば、*strtri* の最適ブロックサイズを検索するには *ilaenv* を次のように使用する。  

```
nb = ilaenv( 1, 'strtri', uplo // diag, n, -1, -1, -1 )  
if( nb.le.1 ) nb = max( 1, n )
```



---

## ieeeck

無限大演算およびNaN 演算が安全かどうかを確認する。`ilaenv` により呼び出される。

---

### 構文

```
ival = ieeeck( ispec, zero, one )
```

### 説明

関数の `ieeeck` は、[ilaenv](#) により呼び出され、無限大および、おそらく NaN 演算が安全であること (トラップが生じない) を確認する。

### 入力パラメータ

<code>ispec</code>	INTEGER。無限大演算のみ、または無限大演算と NaN 演算の両方を検証するかどうかを指定する。 <code>ispec = 0</code> の場合、無限大演算のみを確認する。 <code>ispec = 1</code> の場合、無限大演算と NaN 演算を確認する。
<code>zero</code>	REAL。値 0.0 を含まなければならない。 これはこのコードがコンパイラによって最適化されることがないように渡される。
<code>one</code>	REAL。値 1.0 を含まなければならない。 これはこのコードがコンパイラによって最適化されることがないように渡される。

### 出力値

<code>ndimr</code>	INTEGER。 <code>ival = 0</code> の場合、演算は失敗し、正確な結果を得られなかったを示す。 <code>ival = 1</code> の場合、演算により正確な結果が得られたことを示す。
--------------------	---

---

## lsame

2 つの文字の同一性を大文字 / 小文字に関わらずテストする。

---

### 構文

```
val = lsame( ca, cb )
```

### 説明

この論理関数は、*ca* と *cb* が大文字 / 小文字に関わらず同一の文字のときに `.TRUE.` を返す。

### 入力パラメータ

*ca*, *cb*                      CHARACTER\*1。比較する単一文字を指定する。

### 出力パラメータ

*val*                          LOGICAL。比較結果を示す。

---

## lsamen

2 つの文字列の同一性を大文字 / 小文字に関わらずテストする。

---

### 構文

```
val = lsamen( n, ca, cb )
```

### 説明

この論理関数は、文字列 *ca* の最初の *n* 文字と文字列 *cb* の最初の *n* 文字が大文字 / 小文字に関わらず、等しいかどうかをテストする。`lsamen` 関数は大文字 / 小文字に関わらず、*ca* と *cb* が等価のときに `.true.` を返し、それ以外では `.FALSE.` を返す。`len(ca)` または `len(cb)` が *n* 未満のときにも、`lsamen` は `.false.` を返す。

### 入力パラメータ

*n* INTEGER。 *ca* と *cb* を比較する文字数。

*ca*, *cb* CHARACTER\*(\*)。長さ *n* 以上の 2 つの比較する文字列を指定する。  
それぞれの文字列は先頭の *n* 文字のみアクセスされる。

### 出力パラメータ

*val* LOGICAL。比較結果を示す。

---

## ?labad

指数範囲がきわめて大きい場合にアンダーフロー  
およびオーバーフローしきい値の平方根を返す。

---

### 構文

```
call slabad( small, large )  
call dlabad( small, large )
```

### 説明

このルーチンは、アンダーフローとオーバーフローに対して *slamch*/*dlamch* によって計算された値を入力として受け取り、*large* のログが十分に大きい場合にそれら値のそれぞれの平方根を返す。このサブルーチンは、Cray のような大きな指数範囲を持つマシンを判別するために使用し、アンダーフローとオーバーフローのリミット値を *?lamch* で計算された値の平方根に再定義する。Cray に見られるような上半分の指数範囲における計算能力が弱い場合でも *?lamch* は補正を行わないため、このサブルーチンが必要となる。

### 入力パラメータ

*small* REAL(*slabad* の場合 )  
DOUBLE PRECISION(*dlabad* の場合 )  
*?lamch* によって計算されたアンダーフローしきい値。

*large* REAL(*slabad* の場合 )  
DOUBLE PRECISION(*dlabad* の場合 )  
*?lamch* によって計算されたオーバーフローしきい値。

### 出力パラメータ

<i>small</i>	$\log_{10}(\textit{large})$ が十分に大きい場合は <i>small</i> の平方根で上書きされ、そうでない場合は変更されない。
<i>large</i>	$\log_{10}(\textit{large})$ が十分に大きい場合は <i>large</i> の平方根で上書きされ、そうでない場合は変更されない。

---

## ?lamch

浮動小数点演算に対するマシン・パラメータを決定する。

---

### 構文

```
val = slamch( cmach )  
val = dlamch( cmach )
```

### 説明

関数 ?lamch は、単精度および倍精度のマシン・パラメータを決定する。

### 入力パラメータ

*cmach* CHARACTER\*1。?lamch が返す値を指定する。  
= 'E' または 'e', *val* = *eps*  
= 'S' または 's', *val* = *sfmin*  
= 'B' または 'b', *val* = *base*  
= 'P' または 'p', *val* = *eps*\**base*  
= 'N' または 'n', *val* = *t*  
= 'R' または 'r', *val* = *rnd*  
= 'M' または 'm', *val* = *emin*  
= 'U' または 'u', *val* = *rmin*  
= 'L' または 'l', *val* = *emax*  
= 'O' または 'o', *val* = *rmax*  
ここで  
*eps* = 相対マシン精度。 *sfmin* = 1/*sfmin* がオーバーフローしないような安全な最小値。  
*base* = マシンの基数。  
*prec* = *eps*\**base*

$t$  = 仮数における ( 基数 ) 桁数。  
 $rnd$  = 丸めが追加で発生した場合 1.0、それ以外では 0.0  
 $emin$  = ( 緩やかに ) アンダーフローを起こす前の最小指数。  
 $rmin = \text{underflow\_threshold} - \text{base}^{**}(\text{emin}-1)$   
 $emax$  = オーバーフローを起こす前の最大指数  
 $rmax = \text{overflow\_threshold} - (\text{base}^{**}\text{emax})*(1-\text{eps})$

### 出力パラメータ

$val$                     REAL (slamch の場合 )  
                         DOUBLE PRECISION (dlamch の場合 )  
                         関数の戻り値。

## ?lamc1

?lamc2 から呼び出される。  
 $\text{beta}$ ,  $t$ ,  $rnd$ ,  $ieee1$  で与えられるマシン・パラメータを決定する。

### 構文

```
call slamc1( beta, t, rnd, ieee1 )
call dlamc1( beta, t, rnd, ieee1 )
```

### 説明

ルーチン ?lamc1 は  $\text{beta}$ ,  $t$ ,  $rnd$ ,  $ieee1$  で与えられるマシン・パラメータを決定する。

### 出力パラメータ

$\text{beta}$                     INTEGER。 マシンの基数。  
 $t$                         INTEGER。 仮数における ( $\text{beta}$ ) 桁数。  
 $rnd$                     LOGICAL。  
                         適切な丸め ( $rnd = .TRUE.$ ) または切捨て  
                         ( $rnd = .FALSE.$ ) が追加で発生したかを示す。 マシンの演算性能を決  
                         める信頼性のある指標ではない。  
 $ieee1$                   LOGICAL。  
                          $ieee$  の「round to nearest」形式で丸めが発生したかどうかを示す。

## ?lamc2

?lamch で使用される。  
引数リストで指定されたマシン・パラメータを  
決定する。

---

### 構文

```
call slamc2( beta, t, rnd, eps, emin, rmin, emax, rmax )  
call dlamc2( beta, t, rnd, eps, emin, rmin, emax, rmax )
```

### 説明

ルーチン ?lamc2 は引数リストで指定されたマシン・パラメータを決定する。

### 出力パラメータ

<i>beta</i>	INTEGER。マシンの基数。
<i>t</i>	INTEGER。仮数における ( <i>beta</i> ) 桁数。
<i>rnd</i>	LOGICAL。 適切な丸め ( <i>rnd</i> = .TRUE.) または切捨て ( <i>rnd</i> = .FALSE.) が追加で発生したかを示す。マシンの演算性能を決 める信頼性のある指標ではない。
<i>eps</i>	REAL (slamc2 の場合 ) DOUBLE PRECISION (dlamc2 の場合 ) 以下を満たすような最小の正の 値。 $fl(1.0 - eps) < 1.0$ 、 <i>fl</i> は計算された値を示す。
<i>emin</i>	INTEGER。(緩やかに) アンダーフローを起こす前の最小指数。
<i>rmin</i>	REAL (slamc2 の場合 ) DOUBLE PRECISION (dlamc2 の場合 ) $base^{emin-1}$ で与えられ、 <i>base</i> は <i>beta</i> の浮動小数点値である。
<i>emax</i>	INTEGER。オーバーフローを起こす前の最大指数
<i>rmax</i>	REAL (slamc2 の場合 ) DOUBLE PRECISION (dlamc2 の場合 ) マシンに対する最大の正の数で、 $base^{emax(1-eps)}$ で与えられ、 <i>base</i> は <i>beta</i> の浮動小数点値である。

---

## ?lamc3

?lamc1 ~ ?lamc5 から呼び出される。a と b の加算を実行する前に a と b を格納させる。

---

### 構文

```
val = slamc3( a, b )  
val = dlamc3( a, b )
```

### 説明

このルーチンは、a と b の加算を実行する前に a と b を格納させる。オブティマイザがこれらの値をレジスタに保持している場合に使用する。

### 入力パラメータ

a, b	REAL (slamc3 の場合 )
	DOUBLE PRECISION (dlamc3 の場合 )
	a と b の値。

### 出力パラメータ

val	REAL (slamc3 の場合 )
	DOUBLE PRECISION (dlamc3 の場合 )
	a と b を加算した結果。

---

## ?lamc4

?lamc2 のサービスルーチンである。

---

### 構文

```
call slamc4( emin, start, base )  
call dlamc4( emin, start, base )
```

### 説明

このルーチンは [?lamc2](#) のサービスルーチンである。

### 入力パラメータ

<i>start</i>	REAL (slamc4 の場合 ) DOUBLE PRECISION (dlamc4 の場合 ) <i>emin</i> を決定する開始点。
<i>base</i>	INTEGER。マシンの基数。

### 出力パラメータ

<i>emin</i>	INTEGER。(緩やかに)アンダーフローを起こす前の最小指数で、 $a = start$ と設定し、前の $a$ が回復できなくなるまで基数による除算 を行って計算される。
-------------	---

---

## ?lamc5

?lamc2 から呼び出される。  
オーバーフローしないマシンの最大浮動小数点値  
の計算を試みる。

---

### 構文

```
call slamc5( beta, p, emin, iieee, emax, rmax )  
call dlamc5( beta, p, emin, iieee, emax, rmax )
```

### 説明

ルーチン ?lamc5 は、オーバーフローしない最大マシン浮動小数点値 *rmax* の計算を試みる。 $emax + \text{abs}(emin)$  の合計はおおよそ 2 のべき乗であると仮定する。  
例えば、Cyber 205 ( $emin = -28625$ ,  $emax = 28718$ ) のように、この仮定が満たされない場合、マシンでの実行は失敗する。*emin* にオーバーフローするであろう大きすぎる値 (すなわちゼロに近い) を与えても同様にマシンでの実行は失敗する。

### 入力パラメータ

<i>beta</i>	INTEGER。浮動小数点演算の基数。
<i>p</i>	INTEGER。浮動小数点演算の仮数における基数 <i>beta</i> の桁数。
<i>emin</i>	INTEGER。(緩やかに)アンダーフローを起こす前の最小指数。
<i>ieee</i>	LOGICAL。演算システムが IEEE スタANDARD に準拠していると考えられるかどうかを示す論理フラグ。



### 出力パラメータ

<i>emax</i>	INTEGER。オーバーフローを起こす前の最大指数。
<i>rmax</i>	REAL (slamc5 の場合 ) DOUBLE PRECISION (dlamc5 の場合 ) 最大マシン浮動小数点値。

---

## second/dsecnd

プロセスのユーザ時間を返す。

---

### 構文

```
val = second()  
val = dsecnd()
```

### 説明

関数 second/dsecnd はプロセスのユーザ時間 ( 秒単位 ) を返す。これらの関数のバージョンはシステム関数の etime から時間を取得する。相違点は dsecnd が結果を倍精度で返すことである。

### 出力パラメータ

<i>val</i>	REAL (second の場合 ) DOUBLE PRECISION (dsecnd の場合 ) プロセスのユーザ時間。
------------	---

---

## xerbla

BLAS ルーチン、LAPACK ルーチン、VML ルーチンから呼び出されるエラー処理ルーチン。

---

### 構文

```
call xerbla( srname, info )
```

### 説明

ルーチン `xerbla` は **BLAS** ルーチン、**LAPACK** ルーチン、**VML** ルーチンのエラー処理ルーチンである。**BLAS** ルーチン、**LAPACK** ルーチンまたは **VML** ルーチンで不正な入力パラメータの値があったときに呼び出される。  
メッセージを出力し実行を停止する。  
ルーチンを導入する場合に、システム固有の例外処理機能を呼び出すのであれば、`stop` 文の変更を考慮するとよい。

### 入力パラメータ

<i>srname</i>	CHARACTER*6 <code>xerbla</code> を呼び出したルーチンの名前。
<i>info</i>	INTEGER。 呼び出しルーチンのパラメータ・リストにおいて、不正なパラメータの位置。

# ScaLAPACK ルーチン

## 6

この章では、分散型メモリ・アーキテクチャ用の ScaLAPACK ルーチンに関するインテル<sup>®</sup> マス・カーネル・ライブラリ (インテル<sup>®</sup> MKL) の実装について説明する。

ScaLAPACK ルーチンは、実数と複素数の密行列および帯行列をサポートし、連立 1 次方程式、線形の最小二乗問題、固有値と特異値問題を解くタスクに加えて、さまざまな関連する計算タスクを実行する。すべてのルーチンは、単精度と倍精度の両方で使用できる。



---

**注** :ScaLAPACK ルーチンは、インテル MKL のスーパーセットであるインテル<sup>®</sup> クラスタ・マス・カーネル・ライブラリ (インテル<sup>®</sup> クラスタ MKL) で提供されている。

---

この章の各セクションで、ScaLAPACK の[計算ルーチン](#) (計算タスクを個別に実行) と[ドライバルーチン](#) (1 回の呼び出しで標準的な問題の解を算出) について説明する。

一般に、ScaLAPACK は、メッセージ・パッシング・レイヤとして MPI を使用するコンピュータのネットワーク上で実行され、事前に構築された 1 セットのコミュニケーション・サブプログラム (BLACS) と対象のアーキテクチャ用に最適化された BLAS のセットを使用する。ScaLAPACK のインテル・クラスタ MKL 版は、インテル<sup>®</sup> プロセッサ用に最適化されている。詳細なシステム要件と環境要件は、「インテル MKL リリースノート」および「インテル MKL テクニカル・ユーザ・ノート」を参照。

ScaLAPACK ルーチンの完全なリファレンスと関連情報は、[\[SLUG\]](#) を参照。

## 概要

ScaLAPACK 用計算環境のモデルは、プロセスの 1 次元配列 ( 帯行列または三角対角行列の計算の場合 ) または 2 次元のプロセスグリッド ( 密行列の計算の場合 ) として表される。ScaLAPACK を使用するには、すべてのグローバル行列またはベクトルは、ScaLAPACK ルーチンを呼び出す前に、この配列またはグリッド上に分割されなければならない。

ScaLAPACK は、密行列計算用のレイアウトとして、2 次元のブロック・サイクリック・データ分割を使用する。この分割により、利用可能なプロセッサ間で作業の平衡化が行われ、BLAS レベル 3 のルーチンを最適なローカル計算で使うことが可能になる。各グローバル配列と対応するプロセスおよびメモリ位置間のマッピングを確立するために必要なデータ分割に関する情報は、各グローバル配列と関連する *配列ディスクリプタ* に格納される。

配列ディスクリプタ構造の例を [表 6-1](#) に示す。

**表 6-1 密行列用の配列ディスクリプタの内容**

配列成分 #	名前	定義
1	<i>dtype</i>	ディスクリプタのタイプ ( 密行列の場合は 1 )
2	<i>ctxt</i>	プロセスグリッドの BLACS コンテキスト・ハンドル
3	<i>m</i>	グローバル配列の行数
4	<i>n</i>	グローバル配列の列数
5	<i>mb</i>	行ブロック化係数
6	<i>nb</i>	列ブロック化係数
7	<i>rsrc</i>	グローバル配列の最初の行が分割されるプロセス列
8	<i>csrc</i>	グローバル配列の最初の列が分割されるプロセス行
9	<i>lld</i>	ローカル配列のリーディング・ディメンジョン

データ分割後にグリッドの特定のプロセスが受け取るグローバル密行列の行数と列数は、 $LOC_r()$  と  $LOC_c()$  で示される。これらの数は、ScaLAPACK ルーチン `numroc` を使用して計算できる。

グローバル・データのブロック・サイクリック分割が完了したら、グローバル行列 *A* の部分行列で実行する操作を選択できる。操作は、グローバル部分配列 `sub(A)` を含み、以下の 6 つの値で定義される ( 密行列の場合 )。

<i>m</i>	<code>sub(A)</code> の行数
<i>n</i>	<code>sub(A)</code> の列数

<i>a</i>	全体のグローバル配列 <i>A</i> を含むローカル配列へのポインタ
<i>ia</i>	グローバル配列における <i>sub(A)</i> の行インデックス
<i>ja</i>	グローバル配列における <i>sub(A)</i> の列インデックス
<i>desca</i>	グローバル配列の配列ディスクリプタ

## ルーチン命名規則

この章の各ルーチンについて、ScaLAPACK 名を使用できる。ScaLAPACK ルーチンの命名規則は、LAPACK ルーチン ( 第 4 章の [ルーチン命名規則](#) を参照 ) に使用されているものと似ている。一般に、ScaLAPACK のルーチン名は、LAPACK 名の先頭に文字 *p* が追加された名前である。

**ScaLAPACK 名** は、以下に示すように、*p?yyzzz* または *p?yyzz* という構造になっている。

先頭の文字 *p* は、ScaLAPACK ルーチン用の特殊なプリフィックスであり、個々のルーチンに追加される。

2 番目の記号 *?* は、データ型を示す。

<i>s</i> 実数、単精度	<i>c</i> 複素数、単精度
<i>d</i> 実数、倍精度	<i>z</i> 複素数、倍精度

3 番目と 4 番目の文字 *yy* は、行列のタイプを示す。

<i>ge</i>	一般行列
<i>gb</i>	一般帯行列
<i>gg</i>	一般行列のペア ( 汎用問題用 )
<i>dt</i>	一般三重対角行列 ( 対角優位 )
<i>db</i>	一般帯行列 ( 対角優位 )
<i>po</i>	対称 / エルミート 正定値行列
<i>pb</i>	対称 / エルミート 正定値帯行列
<i>pt</i>	対称 / エルミート 正定値三重対角行列
<i>sy</i>	対称行列
<i>st</i>	対称三重対角行列 ( 実数 )
<i>he</i>	エルミート行列
<i>or</i>	直交行列
<i>tr</i>	三角 ( または準三角 ) 行列
<i>tz</i>	台形行列
<i>un</i>	ユニタリ行列

計算ルーチンでは、最後の 3 つの文字 **zzz** は、実行される処理を示し、LAPACK ルーチンと同じ意味を持つ。

ドライバルーチンでは、最後の 2 つの文字 **zz** または 3 つの文字 **zzz** は次の意味を持つ。

sv    1 次方程式を解くための簡易ドライバ  
svx   1 次方程式を解くための高度ドライバ  
ls    線形最小二乗問題を解くためのドライバ  
ev    対称固有値問題を解くための簡易ドライバ  
evx   対称固有値問題を解くための高度ドライバ  
svd   特異値分解計算用のドライバ  
gvx   汎用対称固有値問題を解くための高度ドライバ

簡易ドライバは、汎用問題のみを解くドライバである。高度ドライバは、より用途が広く、その他の関連する演算（例えば、1 次方程式を解いた後の、解の精度の改善と誤差範囲の計算など）を行うこともできる。

## 計算ルーチン

以下の各セクションでは、ScaLAPACK 計算ルーチンについて説明する。LAPACK 計算ルーチンは、次の目的の計算タスクを個別に実行する。

- [連立 1 次方程式を解く](#)
- [直交因子分解 および LLS 問題](#)
- [対称固有値問題](#)
- [非対称固有値問題](#)
- [特異値分解](#)
- [汎用対称固有値問題](#)

各[ドライバルーチン](#)も参照のこと。

### 1 次方程式

ScaLAPACK は、以下のタイプの行列を持つ連立方程式用のルーチンをサポートしている。

- 一般行列
- 一般帯行列
- 一般対角優位帯行列（一般三重対角行列を含む）
- 対称 / エルミート 正定値行列
- 対称 / エルミート 正定値帯行列
- 対称 / エルミート 正定値三重対角行列

対角優位行列は、この行列の LU 因子分解にピボット演算が不要であることがあらかじめわかっている行列として定義される。

これらの各行列タイプについて、ライブラリには、行列の因子分解、行列の平衡化、連立 1 次方程式の解の算出、行列の条件数の推定、1 次方程式の解の精度の改善と誤差範囲の計算、行列の逆転を計算するためのルーチンが用意されている。いくつかの行列タイプでは、一部の計算ルーチンのみが提供されている点に注意する（例えば、解の精度を改善するルーチンは帯行列や三角行列では提供されていない）。使用可能なルーチンの完全なリストは、表 6-2 を参照のこと。

特定の問題を解くために、2 つ以上の計算ルーチン呼び出しか、1 回の呼び出しで複数のタスクが組み合わさった対応する [ドライバルーチン](#) を呼び出せる。したがって、一般行列の連立 1 次方程式を解く場合、最初に `p?getrf` (LU 因子分解) を呼び出し、次に `p?getrs` (解の算出) を呼び出せる。さらに、`p?gerfs` を呼び出して、解の精度を改善し、誤差範囲を計算できる。代わりに、これらのタスクを 1 回の呼び出しで実行するドライバルーチン `p?gesvx` を使用することもできる。

表 6-2 に、実数行列の因子分解、平衡化、行列の逆転、条件数の推定、連立 1 次方程式の解の算出、解の精度の改善、誤差の推定を行うための ScaLAPACK 計算ルーチンを示す。

表 6-2 連立 1 次方程式用の計算ルーチン

行列のタイプ、 格納形式	行列の 因子分解	行列の 平衡化	方程式の 解の算出	条件数	誤差の 推定	逆行列の 計算
一般 (部分ピボット演算)	<a href="#">p?getrf</a>	<a href="#">p?geequ</a>	<a href="#">p?getrs</a>	<a href="#">p?gecon</a>	<a href="#">p?gerfs</a>	<a href="#">p?getri</a>
一般帯 (部分ピボット演算)	<a href="#">p?gbtrf</a>		<a href="#">p?gbtrs</a>			
一般帯 (ピボット演算なし)	<a href="#">p?dbtrf</a>		<a href="#">p?dbtrs</a>			
一般三重対角 (ピボット演算なし)	<a href="#">p?dttrf</a>		<a href="#">p?dttrs</a>			
対称 / エルミート正 定値	<a href="#">p?potrf</a>	<a href="#">p?poequ</a>	<a href="#">p?potrs</a>	<a href="#">p?pocon</a>	<a href="#">p?porfs</a>	<a href="#">p?potri</a>
対称 / エルミート正 定値 (帯形式)	<a href="#">p?pbtrf</a>		<a href="#">p?pbtrs</a>			
対称 / エルミート正 定値 (三重対角形式)	<a href="#">p?pttrf</a>		<a href="#">p?pttrs</a>			
三角			<a href="#">p?trtrs</a>	<a href="#">p?trcon</a>	<a href="#">p?trrfs</a>	<a href="#">p?trtri</a>

表中の ? は、s (単精度実数)、d (倍精度実数)、c (単精度複素数)、または z (倍精度複素数) を示す。

### 行列の因子分解用のルーチン

このセクションでは、行列の因子分解用の ScaLAPACK ルーチンについて説明する。以下の因子分解をサポートしている。

- 一般行列の LU 因子分解
- 対角優位行列の LU 因子分解
- 実対称 / 複素エルミート正定値行列のコレスキー因子分解

因子分解の計算には、フル格納と帯格納形式の行列を使用できる。

---

### p?getrf

$m \times n$  の一般分割行列の LU 因子分解を行う。

---

#### 構文

```
call psgetrf( m, n, a, ia, ja, desca, ipiv, info )
call pdgetrf( m, n, a, ia, ja, desca, ipiv, info )
call pcgetrf( m, n, a, ia, ja, desca, ipiv, info )
call pzgetrf( m, n, a, ia, ja, desca, ipiv, info )
```

#### 説明

このルーチンは、次の式に従って、 $m \times n$  の一般分割行列  $\text{sub}(A) = A(ia:ia+n-1, ja:ja+n-1)$  の LU 因子分解を行う。

$$A = P L U$$

ここで、 $P$  は置換行列、 $L$  は単位対角成分を含む下三角 ( $m > n$  の場合は下台形)、 $U$  は上三角 ( $m < n$  の場合上台形) である。 $L$  と  $U$  は  $\text{sub}(A)$  に格納される。

このルーチンでは、行を交換し、部分的にピボット演算を行う。

#### 入力パラメータ

$m$	(グローバル) INTEGER。 分散部分行列 $\text{sub}(A)$ の行数 ( $m \geq 0$ )。
$n$	(グローバル) INTEGER。 分散部分行列 $\text{sub}(A)$ の列数 ( $n \geq 0$ )。



<i>a</i>	(ローカル) REAL (psgetrf の場合) DOUBLE PRECISION (pdgetrf の場合) COMPLEX (pcgetrf の場合) DOUBLE COMPLEX (pzgetrf の場合)。 ローカルメモリにある、ローカル次元 ( $lld\_a$ , $LOC_c(ja+n-1)$ ) の配列へのポインタ。 因子分解する分散行列 $\text{sub}(A)$ のローカル部分を含む。
<i>ia, ja</i>	(グローバル) INTEGER。 それぞれ、部分行列 $A(ia:ia+n-1, ja:ja+n-1)$ の最初の行と最初の列を示すグローバル配列 $A$ の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。 配列、次元は ( $dlen\_$ )。分散行列 $A$ の配列ディスクリプタ。

### 出力パラメータ

<i>a</i>	因子分解 $A = PLU$ で得られた係数 $L$ と $U$ のローカル部分によって上書きされる。 $L$ の単位対角成分は格納されない。
<i>ipiv</i>	(ローカル) INTEGER。配列。 $ipiv$ の次元は ( $LOC_r(m\_a) + mb\_a$ )。 この配列には、ピボット演算情報が格納される。ローカル行 $i$ は、グローバル行 $ipiv(i)$ と交換される。この配列は、分散行列 $A$ に関連付けられる。
<i>info</i>	(グローバル) INTEGER。 $info = 0$ の場合、実行は正常に終了したことを示す。 $info < 0$ の場合： $i$ 番目の引数が配列で、 $j$ 番目の値が不正だった場合、 $info = -(i*100+j)$ 。 $i$ 番目の引数がスカラで値が不正だった場合、 $info = -i$ 。 $info = i$ の場合、 $u_{ii}$ は 0 である。因子分解は完了したが、係数 $U$ は完全に特異である。連立 1 次方程式の解の算出に係数 $U$ を使用すると、0 による除算が発生する。

### p?gbtrf

$n \times n$  の一般帯分割行列の LU 因子分解を行う。

---

#### 構文

```
call psgbtrf(n, bwl, bwu, a, ja, desca, ipiv, af, laf, work, lwork, info)
call pdgbtrf(n, bwl, bwu, a, ja, desca, ipiv, af, laf, work, lwork, info)
call pcgbtrf(n, bwl, bwu, a, ja, desca, ipiv, af, laf, work, lwork, info)
call pzgbtrf(n, bwl, bwu, a, ja, desca, ipiv, af, laf, work, lwork, info)
```

#### 説明

このルーチンは、行交換を伴う部分ピボット演算を用いて、 $n \times n$  の一般実 / 複素帯分散行列  $A(1:n, ja:ja+n-1)$  の LU 因子分解を行う。

ここで行われる因子分解は、LAPACK ルーチン [?gbtrf](#) から返される因子分解とは異なる。並列用の行列では、追加の置換が実行される。

因子分解の形式は次のとおりである。

$$A(1:n, ja:ja+n-1) = P L U Q$$

ここで、 $P$  と  $Q$  は置換行列、 $L$  は下三角行列、 $U$  は上三角行列である。行列  $Q$  は並列化のために列の順序を変更することを表し、 $P$  は数の安定のために部分的なピボット演算を使用して行の順序を変更することを表す。

#### 入力パラメータ

$n$	(グローバル) INTEGER。 分散部分行列 $A(1:n, ja:ja+n-1)$ の行と列の数 ( $n \geq 0$ )。
$bwl$	(グローバル) INTEGER。 $A$ の帯内の劣対角成分の数 ( $0 \leq bwl \leq n-1$ )。
$bwu$	(グローバル) INTEGER。 $A$ の帯内の優対角成分の数 ( $0 \leq bwu \leq n-1$ )。
$a$	(ローカル) REAL (psgbtrf の場合) DOUBLE PRECISION (pdgbtrf の場合) COMPLEX (pcgbtrf の場合)

DOUBLE COMPLEX (pzgbtrf の場合)。  
 ローカルメモリにある、ローカル次元 ( $lld\_a$ ,  $LOC_c(ja+n-1)$ ) の配列へのポインタ。ここで、 $lld\_a \geq 2*bw1 + 2*bwu + 1$  である。  
 因子分解する  $n \times n$  分散帯行列  $A(1:n, ja:ja+n-1)$  のローカル部分を含む。

*ja* (グローバル) INTEGER。  
 ( $A$  のすべて、または  $A$  の部分行列で) 処理される行列の先頭を指すグローバル配列  $A$  のインデックス。

*desca* (グローバルおよびローカル) INTEGER。  
 配列、次元は ( $dlen\_$ )。分散行列  $A$  の配列ディスクリプタ。  
 $desca(dtype\_)$  = 501 の場合、 $dlen\_ \geq 7$ 。  
 $desca(dtype\_)$  = 1 の場合、 $dlen\_ \geq 9$ 。

*laf* (ローカル) INTEGER。配列 *af* の次元。  
 $laf \geq (NB+bwu)*(bw1+bwu)+6*(bw1+bwu)*(bw1+2*bwu)$  でなければならない。

*laf* が小さすぎる場合、エラーコードが返され、エラーにならない最小のサイズが *af*(1) に返される。

*work* (ローカル) *a* と同じタイプ。ワークスペース配列、次元は *lwork*。

*lwork* (ローカルまたはグローバル) INTEGER。  
 配列 *work* のサイズ ( $lwork \geq 1$ )。 *lwork* が小さすぎる場合、エラーコードが返され、エラーにならない最小のサイズが *work*(1) に返される。

## 出力パラメータ

*a* 終了時に、この配列には因子分解の詳細が格納される。行列で追加の置換が行われるため、返される係数は LAPACK から返される係数とは異なる点に注意する。

*ipiv* (ローカル) INTEGER。配列。  
*ipiv* の次元は  $desca(NB)$  以上でなければならない。  
 ローカル因子分解用のピボットのインデックスが格納される。因子分解および解の算出を行う間、この配列の内容は変更すべきでない点に注意する。

<i>af</i>	(ローカル) REAL (psgbtrf の場合) DOUBLE PRECISION (pdgbtrf の場合) COMPLEX (pcgbtrf の場合) DOUBLE COMPLEX (pzgbtrf の場合)。  配列、次元は ( <i>laf</i> )。 補助非零要素空間。非零要素は、因子分解ルーチン <i>p?gbtrf</i> で作成され、 <i>af</i> に格納される。 因子分解ルーチンの後に <i>p?gbtrs</i> を使用して 1 次方程式の解を算出する場合、 <i>af</i> は因子分解の後に変更されてはならない点に注意する。
<i>work(1)</i>	終了時に、 <i>work(1)</i> には、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が格納される。
<i>info</i>	(グローバル) INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> < 0 の場合：  <i>i</i> 番目の引数が配列で、 <i>j</i> 番目の値が不正だった場合、 <i>info</i> = -( <i>i</i> *100+ <i>j</i> )。 <i>i</i> 番目の引数がスカラで値が不正だった場合、 <i>info</i> = - <i>i</i> 。  <i>info</i> > 0 の場合： <i>info</i> = <i>k</i> ≤ NPROCS の場合、プロセッサ <i>info</i> に格納され、ローカルに因子分解された部分行列が非特異でないため、因子分解を完了できなかったことを示す。 <i>info</i> = <i>k</i> > NPROCS の場合、他のプロセッサとの対話を表すプロセッサ <i>info</i> -NPROCS に格納された部分行列が非特異でないため、因子分解を完了できなかったことを示す。

---

### p?dbtrf

$n \times n$  の対角優位帯分散行列の LU 因子分解を行う。

---

#### 構文

```
call psdbtrf( n, bwl, bwu, a, ja, desca, af, laf, work, lwork, info )
call pddbtrf( n, bwl, bwu, a, ja, desca, af, laf, work, lwork, info )
call pcdbtrf( n, bwl, bwu, a, ja, desca, af, laf, work, lwork, info )
call pzdbtrf( n, bwl, bwu, a, ja, desca, af, laf, work, lwork, info )
```

## 説明

このルーチンは、ピボット演算を使用しないで、 $n \times n$  の実 / 複素対角優位帯分散行列  $A(1:n, ja:ja+n-1)$  の  $LU$  因子分解を行う。

因子分解の結果は、LAPACK から返される因子分解とは異なる点に注意する。並列用の行列では、追加の置換が実行される。

## 入力パラメータ

$n$	(グローバル) INTEGER。 分散部分行列 $A(1:n, ja:ja+n-1)$ の行と列の数 ( $n \geq 0$ )。
$bw1$	(グローバル) INTEGER。 $A$ の帯内の劣対角成分の数 ( $0 \leq bw1 \leq n-1$ )。
$bwu$	(グローバル) INTEGER。 $A$ の帯内の優対角成分の数 ( $0 \leq bwu \leq n-1$ )。
$a$	(ローカル) REAL (psdbtrf の場合) DOUBLE PRECISION (pddbtrf の場合) COMPLEX (pcdbtrf の場合) DOUBLE COMPLEX (pzdbtrf の場合)。 ローカルメモリにある、ローカル次元 ( $lld\_a$ , $LOC_c(ja+n-1)$ ) の配列へのポインタ。 因子分解する $n \times n$ 分散帯行列 $A(1:n, ja:ja+n-1)$ のローカル部分を含む。
$ja$	(グローバル) INTEGER。 ( $A$ のすべて、または $A$ の部分行列で) 処理される行列の先頭を指すグローバル配列 $A$ のインデックス。
$desca$	(グローバルおよびローカル) INTEGER。配列、次元は ( $dlen\_$ )。 分散行列 $A$ の配列ディスクリプタ。 $desca(dtype\_)$ = 501 の場合、 $dlen\_ \geq 7$ 。 $desca(dtype\_)$ = 1 の場合、 $dlen\_ \geq 9$ 。
$laf$	(ローカル) INTEGER。配列 $af$ の次元。 $laf \geq NB * (bw1 + bwu) + 6 * (\max(bw1, bwu))^2$ でなければならない。 $laf$ が小さすぎる場合、エラーコードが返され、エラーにならない最小のサイズが $af(1)$ に返される。
$work$	(ローカル) $a$ と同じタイプ。ワークスペース配列、次元は $lwork$ 。

*lwork* (ローカルまたはグローバル) INTEGER。配列 *work* のサイズ。  
 $lwork \geq (\max(bw1, bwu))^2$  でなければならない。*lwork* が小さすぎる場合、エラーコードが返され、エラーにならない最小のサイズが *work*(1) に返される。

### 出力パラメータ

*a* 終了時に、この配列には因子分解の詳細が格納される。行列で追加の置換が行われるため、返される係数は LAPACK から返される係数とは異なる点に注意する。

*af* (ローカル)  
 REAL (psdbtrf の場合)  
 DOUBLE PRECISION (pddbtrf の場合)  
 COMPLEX (pcdbtrf の場合)  
 DOUBLE COMPLEX (pzdbtrf の場合)。

配列、次元は (*laf*)。

補助非零要素空間。非零要素は、因子分解ルーチン *p?dbtrf* で作成され、*af* に格納される。

因子分解ルーチンの後に *p?dbtrs* を使用して 1 次方程式の解を算出する場合、*af* は因子分解の後に変更されてはならない点に注意する。

*work*(1) 終了時に、*work*(1) には、最適なパフォーマンスを得るために必要な *lwork* の最小値が格納される。

*info* (グローバル) INTEGER。  
*info* = 0 の場合、実行は正常に終了したことを示す。  
*info* < 0 の場合：  
*i* 番目の引数が配列で、*j* 番目の値が不正だった場合、  
*info* = -(*i*\*100+*j*)。 *i* 番目の引数がスカラで値が不正だった場合、  
*info* = -*i*。  
*info* > 0 の場合：  
*info* = *k* ≤ NPROCS の場合、プロセッサ *info* に格納され、ローカルに因子分解された部分行列が対角優位でないため、因子分解を完了できなかったことを示す。*info* = *k* > NPROCS の場合、他のプロセッサとの対話を表すプロセッサ *info*-NPROCS に格納された部分行列が非特異でないため、因子分解を完了できなかったことを示す。

## p?potrf

対称 (エルミート) 正定値分散行列のコレスキー  
因子分解を行う。

### 構文

```
call pspotrf( uplo, n, a, ia, ja, desca, info )
call pdpotrf( uplo, n, a, ia, ja, desca, info )
call pcpotrf( uplo, n, a, ia, ja, desca, info )
call pzpotrf( uplo, n, a, ia, ja, desca, info )
```

### 説明

このルーチンは、 $n \times n$  の実対称 / 複素エルミート正定値分散行列  
 $A(ia:ia+n-1, ja:ja+n-1)$  のコレスキー因子分解を行う。行列は、以下  $\text{sub}(A)$  として表す。

因子分解の形式は次のとおりである。

$\text{sub}(A) = U^H U$  ( $\text{uplo} = 'U'$  の場合)、または

$\text{sub}(A) = LL^H$  ( $\text{uplo} = 'L'$  の場合)

ここで、 $L$  は下三角行列、 $U$  は上三角行列である。

### 入力パラメータ

**uplo** (グローバル) CHARACTER\*1。  
'U' または 'L' でなければならない。  
 $\text{sub}(A)$  の上三角部分と下三角部分のどちらが格納されるかを指定する。  
 $\text{uplo} = 'U'$  の場合、配列  $a$  には行列  $\text{sub}(A)$  の上三角部分が格納され、 $\text{sub}(A)$  は  $U^H U$  として因子分解される。  
 $\text{uplo} = 'L'$  の場合、配列  $a$  には行列  $\text{sub}(A)$  の下三角部分が格納され、 $\text{sub}(A)$  は  $LL^H$  として因子分解される。

**n** (グローバル) INTEGER。  
分散部分行列  $\text{sub}(A)$  の次数 ( $n \geq 0$ )。

<i>a</i>	<p>(ローカル)</p> <p>REAL (pspotrf の場合)</p> <p>DOUBLE PRECISION (pdpotrf の場合)</p> <p>COMPLEX (pcpotrf の場合)</p> <p>DOUBLE COMPLEX (pzpotrf の場合)。</p> <p>ローカルメモリにある、次元 (<math>lld\_a</math>, <math>LOC_c(ja+n-1)</math>) の配列へのポインタ。</p> <p>呼び出し時に、この配列は、因子分解する <math>n \times n</math> の対称 / エルミート分散行列 <b>sub(A)</b> のローカル部分を含む。</p> <p><i>uplo</i> に応じて、配列 <i>a</i> は、行列 <b>sub(A)</b> の上三角部分または下三角部分を含む (<i>uplo</i> を参照)。</p>
<i>ia, ja</i>	<p>(グローバル) INTEGER。</p> <p>それぞれ、部分行列 <b>sub(A)</b> の最初の行と最初の列を示す、グローバル配列 <i>A</i> の行インデックスと列インデックス。</p>
<i>desca</i>	<p>(グローバルおよびローカル) INTEGER。配列、次元は (<i>dlen_</i>)。</p> <p>分散行列 <i>A</i> の配列ディスクリプタ。</p>

### 出力パラメータ

<i>a</i>	<i>uplo</i> の指定に従って、 <i>a</i> の上三角部分または下三角部分が、コレスキー係数 <i>U</i> あるいは <i>L</i> によって上書きされる。
<i>info</i>	<p>(グローバル) INTEGER。</p> <p><i>info</i> = 0 の場合、実行は正常に終了したことを示す。</p> <p><i>info</i> &lt; 0 の場合：</p> <p><i>i</i> 番目の引数が配列で、<i>j</i> 番目の値が不正だった場合、  <i>info</i> = -(<i>i</i>*100+<i>j</i>)。 <i>i</i> 番目の引数がスカラで値が不正だった場合、  <i>info</i> = -<i>i</i>。</p> <p><i>info</i> = <i>k</i> &gt; 0 の場合、次数 <i>k</i> の先頭の小行列式 <i>A</i>(<i>ia:ia+k-1, ja:ja+k-1</i>) が正定値でないため、因子分解を完了できなかったことを示す。</p>



## p?pbtrf

対称/エルミート正定値帯分散行列のコレスキー  
因子分解を行う。

### 構文

```
call pspbtrf( uplo, n, bw, a, ja, desca, af, laf, work, lwork, info )
call pdpbtrf( uplo, n, bw, a, ja, desca, af, laf, work, lwork, info )
call pcpbtrf( uplo, n, bw, a, ja, desca, af, laf, work, lwork, info )
call pzpbtrf( uplo, n, bw, a, ja, desca, af, laf, work, lwork, info )
```

### 説明

このルーチンは、 $n \times n$  の実対称 / 複素エルミート正定値帯分散行列  $A(1:n, ja:ja+n-1)$  のコレスキー因子分解を行う。

因子分解の結果は、LAPACK から返される因子分解とは異なる。並列用の行列では、追加の置換が実行される。

因子分解の形式は次のとおりである。

$A(1:n, ja:ja+n-1) = P U^H U P^T$  ( $uplo = 'U'$  の場合)、または

$A(1:n, ja:ja+n-1) = P L L^H P^T$  ( $uplo = 'L'$  の場合)

ここで、 $P$  は置換行列、 $U$  は上三角行列、 $L$  は下三角行列である。

### 入力パラメータ

**uplo** (グローバル) CHARACTER\*1。  
'U' または 'L' でなければならない。  
  
 $uplo = 'U'$  の場合、 $A(1:n, ja:ja+n-1)$  の上三角部分が格納される。  
 $uplo = 'L'$  の場合、 $A(1:n, ja:ja+n-1)$  の下三角部分が格納される。

**n** (グローバル) INTEGER。  
分散部分行列  $A(1:n, ja:ja+n-1)$  の次数 ( $n \geq 0$ )。

**bw** (グローバル) INTEGER。  
分散行列の優対角成分の数 ( $uplo = 'U'$  の場合) または  
劣対角成分の数 ( $uplo = 'L'$  の場合) ( $bw \geq 0$ )。

<i>a</i>	(ローカル) REAL (pspbtrf の場合) DOUBLE PRECISION (pdspbtrf の場合) COMPLEX (pcpbtrf の場合) DOUBLE COMPLEX (pzpbtrf の場合)。  ローカルメモリにある、次元 ( $lld\_a$ , $LOC_c(ja+n-1)$ ) の配列へのポインタ。  呼び出し時に、この配列は、因子分解する対称/エルミート帯分散行列 $A(1:n, ja:ja+n-1)$ の上三角または下三角のローカル部分を含む。
<i>ja</i>	(グローバル) INTEGER。 ( $A$ のすべて、または $A$ の部分行列で) 処理される行列の先頭を指すグローバル配列 $A$ のインデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( $dlen\_$ )。 分散行列 $A$ の配列ディスクリプタ。 $desca(dtype\_)=501$ の場合、 $dlen\_ \geq 7$ 。 $desca(dtype\_)=1$ の場合、 $dlen\_ \geq 9$ 。
<i>laf</i>	(ローカル) INTEGER。配列 <i>af</i> の次元。 $laf \geq (NB+2*bw)*bw$ でなければならない。  $laf$ が小さすぎる場合、エラーコードが返され、エラーにならない最小のサイズが $af(1)$ に返される。
<i>work</i>	(ローカル) <i>a</i> と同じタイプ。ワークスペース配列、次元は <i>lwork</i> 。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。配列 <i>work</i> のサイズ。 $lwork \geq bw^2$ でなければならない。

## 出力パラメータ

<i>a</i>	終了時に、 $info=0$ の場合、 <i>uplo</i> の指定に従って、 帯行列 $A(1:n, ja:ja+n-1)$ のコレスキー因子分解で得られた三角係数 $U$ または $L$ が格納される。
<i>work(1)</i>	終了時に、 <i>work(1)</i> には、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が格納される。
<i>info</i>	(グローバル) INTEGER。  $info=0$ の場合、実行は正常に終了したことを示す。 $info < 0$ の場合： $i$ 番目の引数が配列で、 $j$ 番目の値が不正だった場合、

$info = -(i*100+j)$ 。  $i$  番目の引数がスカラで値が不正だった場合、  
 $info = -i$ 。  
 $info > 0$  の場合：

$info = k \leq NPROCS$  の場合、プロセッサ  $info$  に格納され、ローカルに  
 因子分解された部分行列が正定値でないため、因子分解を完了できな  
 かったことを示す。

$info = k > NPROCS$  の場合、他のプロセッサとの対話を表すプロセッサ  
 $info-NPROCS$  に格納された部分行列が非特異でないため、因子分解を  
 完了できなかったことを示す。

## p?pttrf

対称/エルミート正定値三重対角分散行列の  
 コレスキー因子分解を行う。

### 構文

```
call pspttrf( n, d, e, ja, desca, af, laf, work, lwork, info )
call pdpttrf( n, d, e, ja, desca, af, laf, work, lwork, info )
call pcpttrf( n, d, e, ja, desca, af, laf, work, lwork, info )
call pzpttrf( n, d, e, ja, desca, af, laf, work, lwork, info )
```

### 説明

このルーチンは、 $n \times n$  の実対称 / 複素エルミート正定値三重対角分散行列  
 $A(1:n, ja:ja+n-1)$  のコレスキー因子分解を行う。

因子分解の結果は、LAPACK から返される因子分解とは異なる。並列用の行列では、追  
 加の置換が実行される。

因子分解の形式は次のとおりである。

$$A(1:n, ja:ja+n-1) = P L D L^H P^T \text{ または}$$

$$A(1:n, ja:ja+n-1) = P U^H D U P^T$$

ここで、 $P$  は置換行列、 $U$  は上三角行列、 $L$  は下三角行列である。

## 入力パラメータ

<i>n</i>	(グローバル) INTEGER。 分散部分行列 $A(1:n, ja:ja+n-1)$ の次数 ( $n \geq 0$ )。
<i>d, e</i>	(ローカル) REAL (pspttrf の場合) DOUBLE PRECISION (pdpttrf の場合) COMPLEX (pcpttrf の場合) DOUBLE COMPLEX (pzpttrf の場合)。  それぞれ、ローカルメモリにある、次元 ( <i>desca</i> ( <i>nb_</i> )) の配列へのポインタ。  呼び出し時に、配列 <i>d</i> は、分散行列 <i>A</i> の主対角を格納するグローバル・ベクトルのローカル部分を含む。  呼び出し時に、配列 <i>e</i> は、分散行列 <i>A</i> の上対角を格納するグローバル・ベクトルのローカル部分を含む。
<i>ja</i>	(グローバル) INTEGER。 ( <i>A</i> のすべて、または <i>A</i> の部分行列で) 処理される行列の先頭を指すグローバル配列 <i>A</i> のインデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。 分散行列 <i>A</i> の配列ディスクリプタ。 <i>desca</i> ( <i>dtype_</i> ) = 501 の場合、 <i>dlen_</i> $\geq 7$ 。 <i>desca</i> ( <i>dtype_</i> ) = 1 の場合、 <i>dlen_</i> $\geq 9$ 。
<i>laf</i>	(ローカル) INTEGER。配列 <i>af</i> の次元。 <i>laf</i> $\geq \text{NB}+2$ でなければならない。  <i>laf</i> が小さすぎる場合、エラーコードが返され、エラーにならない最小のサイズが <i>af</i> (1) に返される。
<i>work</i>	(ローカル) <i>d</i> および <i>e</i> と同じタイプ。 ワークスペース配列、次元は <i>lwork</i> 。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。配列 <i>work</i> のサイズ。 <i>lwork</i> $\geq 8 * \text{NPCOL}$ でなければならない。

## 出力パラメータ

<i>d, e</i>	終了時に、因子分解の詳細によって上書きされる。
-------------	-------------------------

<i>af</i>	<p>(ローカル)</p> <p>REAL (pspttrf の場合)</p> <p>DOUBLE PRECISION (pdpttrf の場合)</p> <p>COMPLEX (pcpttrf の場合)</p> <p>DOUBLE COMPLEX (pzpttrf の場合)。</p> <p>配列、次元は (<i>laf</i>)。</p> <p>補助非零要素空間。非零要素は、因子分解ルーチン <i>p?pttrf</i> で作成され、<i>af</i> に格納される。</p> <p>因子分解ルーチンの後に <a href="#">p?pttrs</a> を使用して 1 次方程式の解を算出する場合、<i>af</i> は因子分解の後に変更されてはならない点に注意する。</p>
<i>work(1)</i>	終了時に、 <i>work(1)</i> には、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が格納される。
<i>info</i>	<p>(グローバル) INTEGER。</p> <p><i>info</i> = 0 の場合、実行は正常に終了したことを示す。</p> <p><i>info</i> &lt; 0 の場合：</p> <p><i>i</i> 番目の引数が配列で、<i>j</i> 番目の値が不正だった場合、  <i>info</i> = -(<i>i</i>*100+<i>j</i>)。 <i>i</i> 番目の引数がスカラで値が不正だった場合、  <i>info</i> = -<i>i</i>。</p> <p><i>info</i> &gt; 0 の場合：</p> <p><i>info</i> = <i>k</i> ≤ NPROCS の場合、プロセッサ <i>info</i> に格納され、ローカルに因子分解された部分行列が正定値でないため、因子分解を完了できなかったことを示す。</p> <p><i>info</i> = <i>k</i> &gt; NPROCS の場合、他のプロセッサとの対話を表すプロセッサ <i>info</i>-NPROCS に格納された部分行列が非特異でないため、因子分解を完了できなかったことを示す。</p>

## p?dttrf

対角優位三重対角分散行列の LU 因子分解を行う。

### 構文

```
call psdttrf( n, dl, d, du, ja, desca, af, laf, work, lwork, info )
call pdttrf( n, dl, d, du, ja, desca, af, laf, work, lwork, info )
call pcdttrf( n, dl, d, du, ja, desca, af, laf, work, lwork, info )
call pzdtttrf( n, dl, d, du, ja, desca, af, laf, work, lwork, info )
```

## 説明

このルーチンは、ピボット演算を使用しないで、 $n \times n$  の実 / 複素対角優位三重対角分散行列  $A(1:n, ja:ja+n-1)$  の  $LU$  因子分解を行う。

因子分解の結果は、LAPACK から返される因子分解とは異なる。並列用の行列では、追加の置換が実行される。

因子分解の形式は次のとおりである。

$$A(1:n, ja:ja+n-1) = P L U P^T$$

ここで、 $P$  は置換行列、 $U$  は上三角行列、 $L$  は下三角行列である。

## 入力パラメータ

$n$	(グローバル) INTEGER。処理される行数と列数。 すなわち、分散行列 $A(1:n, ja:ja+n-1)$ の次数 ( $n \geq 0$ )。
$d1, d, du$	(ローカル) REAL (pspttrf の場合) DOUBLE PRECISION (pdpttrf の場合) COMPLEX (pcpttrf の場合) DOUBLE COMPLEX (pzpttrf の場合)。  それぞれ、次元 ( $desca(nb\_)$ ) のローカル配列へのポインタ。  呼び出し時に、配列 $d1$ は、行列の劣対角成分を格納するグローバル・ベクトルのローカル部分を含む。全体的に、 $d1(1)$ は参照されず、 $d1$ は $d$ とアライメントされなければならない。  呼び出し時に、配列 $d$ は、行列の対角成分を格納するグローバル・ベクトルのローカル部分を含む。  呼び出し時に、配列 $du$ は、行列の優対角成分を格納するグローバル・ベクトルのローカル部分を含む。 $du(n)$ は参照されず、 $du$ は $d$ とアライメントされなければならない。
$ja$	(グローバル) INTEGER。 ( $A$ のすべて、または $A$ の部分行列で) 処理される行列の先頭を指すグローバル配列 $A$ のインデックス。
$desca$	(グローバルおよびローカル) INTEGER。配列、次元は ( $dlen\_$ )。 分散行列 $A$ の配列ディスクリプタ。 $desca(dtype\_)$ = 501 の場合、 $dlen\_ \geq 7$ 。 $desca(dtype\_)$ = 1 の場合、 $dlen\_ \geq 9$ 。

<i>laf</i>	(ローカル) INTEGER。配列 <i>af</i> の次元。 $laf \geq 2*(NB+2)$ でなければならない。  <i>laf</i> が小さすぎる場合、エラーコードが返され、エラーにならない最小のサイズが <i>af</i> (1) に返される。
<i>work</i>	(ローカル) <i>d</i> と同じタイプ。ワークスペース配列、次元は <i>lwork</i> 。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。配列 <i>work</i> のサイズ。 $lwork \geq 8*NPCOL$ でなければならない。

### 出力パラメータ

<i>dl, d, du</i>	終了時に、行列の係数を含む情報によって上書きされる。
<i>af</i>	(ローカル) REAL (psdttrf の場合) DOUBLE PRECISION (pddttrf の場合) COMPLEX (pcdttrf の場合) DOUBLE COMPLEX (pzdttrf の場合)。  配列、次元は ( <i>laf</i> )。 補助非零要素空間。非零要素は、因子分解ルーチン <i>p?dttrf</i> で作成され、 <i>af</i> に格納される。 因子分解ルーチンの後に <a href="#">p?dttrs</a> を使用して 1 次方程式の解を算出する場合、 <i>af</i> は因子分解の後に変更されてはならない点に注意する。
<i>work</i> (1)	終了時に、 <i>work</i> (1) には、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が格納される。
<i>info</i>	(グローバル) INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> < 0 の場合： <i>i</i> 番目の引数が配列で、 <i>j</i> 番目の値が不正だった場合、 $info = -(i*100+j)$ 。 <i>i</i> 番目の引数がスカラで値が不正だった場合、 $info = -i$ 。  <i>info</i> > 0 の場合： $info = k \leq NPROCS$ の場合、プロセッサ <i>info</i> に格納され、ローカルに因子分解された部分行列が対角優位でないため、因子分解を完了できなかったことを示す。 $info = k > NPROCS$ の場合、他のプロセッサとの対話を表すプロセッサ <i>info</i> - <i>NPROCS</i> に格納された部分行列が非特異でないため、因子分解を完了できなかったことを示す。

### 連立 1 次方程式を解くためのルーチン

このセクションでは、連立 1 次方程式を解くための ScaLAPACK ルーチンについて説明する。通常は、これらのルーチン呼び出す前に、連立方程式の行列を因子分解する必要がある (この章の[行列の因子分解用のルーチン](#)を参照)。ただし、解を求める連立方程式が三角行列を持つ場合は、因子分解の必要はない。

---

### p?getrs

p?getrf によって行われた LU 因子分解を使用して、一般正方行列の分散連立 1 次方程式を解く。

---

#### 構文

```
call psgetrs(trans, n, nrhs, a, ia, ja, desca, ipiv, b, ib, jb, descb,
             info)
call pdgetrs(trans, n, nrhs, a, ia, ja, desca, ipiv, b, ib, jb, descb,
             info)
call pcgetrs(trans, n, nrhs, a, ia, ja, desca, ipiv, b, ib, jb, descb,
             info)
call pzgetrs(trans, n, nrhs, a, ia, ja, desca, ipiv, b, ib, jb, descb,
             info)
```

#### 説明

このルーチンは、[p?getrf](#) によって行われた LU 因子分解を使用して、 $n \times n$  の一般分散行列  $\text{sub}(A) = A(\text{ia}:\text{ia}+n-1, \text{ja}:\text{ja}+n-1)$  の分散連立 1 次方程式を解く。

*trans* ルーチンで指定される連立方程式の形式は、次のいずれかである。

$\text{sub}(A) * X = \text{sub}(B)$  (転置なし)、

$\text{sub}(A)^T * X = \text{sub}(B)$  (転置あり)、

$\text{sub}(A)^H * X = \text{sub}(B)$  (共役転置)

ここで、 $\text{sub}(B) = B(\text{ib}:\text{ib}+n-1, \text{jb}:\text{jb}+\text{nrhs}-1)$  である。

このルーチン呼び出す前に、p?getrf を呼び出して、 $\text{sub}(A)$  の LU 因子分解を行う必要がある。



## 入力パラメータ

<i>trans</i>	(グローバル) CHARACTER*1。 'N'、'T'、または 'C' のいずれかでなければならない。 方程式の形式を指定する。  <i>trans</i> = 'N' の場合、 $\text{sub}(A) * X = \text{sub}(B)$ を $X$ について解く。 <i>trans</i> = 'T' の場合、 $\text{sub}(A)^T * X = \text{sub}(B)$ を $X$ について解く。 <i>trans</i> = 'C' の場合、 $\text{sub}(A)^H * X = \text{sub}(B)$ を $X$ について解く。
<i>n</i>	(グローバル) INTEGER。 1 次方程式の数。部分行列 $\text{sub}(A)$ の次数 ( $n \geq 0$ )。
<i>nrhs</i>	(グローバル) INTEGER。 右辺の数。分散部分行列 $\text{sub}(B)$ の列数 ( $nrhs \geq 0$ )。
<i>a, b</i>	(グローバル) REAL (psgetrs の場合) DOUBLE PRECISION (pdgetrs の場合) COMPLEX (pcgetrs の場合) DOUBLE COMPLEX (pzgetrs の場合)。 それぞれ、ローカルメモリにある、ローカル次元の配列 $a(lld\_a, LOC_c(ja+n-1))$ と $b(lld\_b, LOC_c(jb+nrhs-1))$ へのポインタ。 呼び出し時に、配列 $a$ は因子分解 $\text{sub}(A) = PLU$ で得られた係数 $L$ と $U$ のローカル部分を含む。 $L$ の単位対角成分は格納されない。 呼び出し時に、配列 $b$ は右辺 $\text{sub}(B)$ を含む。
<i>ia, ja</i>	(グローバル) INTEGER。 それぞれ、部分行列 $\text{sub}(A)$ の最初の行と最初の列を示す、グローバル配列 $A$ の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。 分散行列 $A$ の配列ディスクリプタ。
<i>ipiv</i>	(ローカル) INTEGER。配列。 <i>ipiv</i> の次元は $(LOC_r(m\_a) + mb\_a)$ 。 この配列は、ピボット演算情報を含む。行列のローカル行 $i$ は、グローバル行 $ipiv(i)$ と交換される。 この配列は、分散行列 $A$ に関連付けられる。
<i>ib, jb</i>	(グローバル) INTEGER。 それぞれ、部分行列 $\text{sub}(B)$ の最初の行と最初の列を示す、グローバル配列 $B$ の行インデックスと列インデックス。

*descb* (グローバルおよびローカル) INTEGER。配列、次元は (*dlen\_*)。分散行列 *B* の配列ディスクリプタ。

### 出力パラメータ

*b* 終了時に、解の分散行列 *X* によって上書きされる。

*info* INTEGER。 *info* = 0 の場合、実行は正常に終了したことを示す。  
*info* < 0 の場合：  
*i* 番目の引数が配列で、*j* 番目の値が不正だった場合、  
*info* = -(*i*\*100+*j*)。 *i* 番目の引数がスカラで値が不正だった場合、  
*info* = -*i*。

---

## p?gbtrs

p?gbtrf によって行われた *LU* 因子分解を使用して、一般帯行列の分散連立 1 次方程式を解く。

---

### 構文

```
call psgbtrs(trans, n, bwl, bwu, nrhs, a, ja, desca, ipiv, b, ib, descb,  
             af, laf, work, lwork, info)  
call pdgbtrs(trans, n, bwl, bwu, nrhs, a, ja, desca, ipiv, b, ib, descb,  
             af, laf, work, lwork, info)  
call pcgbtrs(trans, n, bwl, bwu, nrhs, a, ja, desca, ipiv, b, ib, descb,  
             af, laf, work, lwork, info)  
call pzgbtrs(trans, n, bwl, bwu, nrhs, a, ja, desca, ipiv, b, ib, descb,  
             af, laf, work, lwork, info)
```

### 説明

このルーチンは、p?gbtrf によって行われた *LU* 因子分解を使用して、一般帯行列  $\text{sub}(A) = A(1:n, ja:ja+n-1)$  の分散連立 1 次方程式を解く。

*trans* ルーチンで指定される連立方程式の形式は、次のいずれかである。

$\text{sub}(A) * X = \text{sub}(B)$  (転置なし)、

$\text{sub}(A)^T * X = \text{sub}(B)$  (転置あり)、

$\text{sub}(A)^H * X = \text{sub}(B)$  ( 共役転置 )

ここで、 $\text{sub}(B) = B(ib:ib+n-1, 1:nrhs)$  である。

このルーチンを呼び出す前に、[p?gbtrf](#) を呼び出して、 $\text{sub}(A)$  の  $LU$  因子分解を行う必要がある。

## 入力パラメータ

<i>trans</i>	( グローバル ) CHARACTER*1。 'N'、'T'、または 'C' のいずれかでなければならない。 方程式の形式を指定する。 <i>trans</i> = 'N' の場合、 $\text{sub}(A)*X = \text{sub}(B)$ を $X$ について解く。 <i>trans</i> = 'T' の場合、 $\text{sub}(A)^T * X = \text{sub}(B)$ を $X$ について解く。 <i>trans</i> = 'C' の場合、 $\text{sub}(A)^H * X = \text{sub}(B)$ を $X$ について解く。
<i>n</i>	( グローバル ) INTEGER。 1 次方程式の数。分散部分行列 $\text{sub}(A)$ の次数 ( $n \geq 0$ )。
<i>bwl</i>	( グローバル ) INTEGER。 $A$ の帯内の劣対角成分の数 ( $0 \leq bwl \leq n-1$ )。
<i>bwu</i>	( グローバル ) INTEGER。 $A$ の帯内の優対角成分の数 ( $0 \leq bwu \leq n-1$ )。
<i>nrhs</i>	( グローバル ) INTEGER。 右辺の数。分散部分行列 $\text{sub}(B)$ の列数 ( $nrhs \geq 0$ )。
<i>a, b</i>	( グローバル ) REAL (psgbtrs の場合 ) DOUBLE PRECISION (pdgbtrs の場合 ) COMPLEX (pcgbtrs の場合 ) DOUBLE COMPLEX (pzgbtrs の場合 )。 それぞれ、ローカルメモリにある、ローカル次元の配列 $a(lld\_a, LOC_c(ja+n-1))$ と $b(lld\_b, LOC_c(nrhs))$ へのポインタ。 配列 $a$ は、分散帯行列 $A$ の $LU$ 因子分解の詳細を含む。 呼び出し時に、配列 $b$ は右辺 $B(ib:ib+n-1, 1:nrhs)$ のローカル部分を含む。
<i>ja</i>	( グローバル ) INTEGER。 ( $A$ のすべて、または $A$ の部分行列で ) 処理される行列の先頭を指すグローバル配列 $A$ のインデックス。

<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。 分散行列 <i>A</i> の配列ディスクリプタ。 <i>desca</i> ( <i>dtype_</i> ) = 501 の場合、 <i>dlen_</i> ≥ 7。 <i>desca</i> ( <i>dtype_</i> ) = 1 の場合、 <i>dlen_</i> ≥ 9。
<i>ib</i>	(グローバル) INTEGER。 ( <i>A</i> のすべて、または <i>A</i> の部分行列で) 処理される行列の先頭を指すグローバル配列 <i>A</i> のインデックス。
<i>descb</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。 分散行列 <i>A</i> の配列ディスクリプタ。 <i>desca</i> ( <i>dtype_</i> ) = 501 の場合、 <i>dlen_</i> ≥ 7。 <i>desca</i> ( <i>dtype_</i> ) = 1 の場合、 <i>dlen_</i> ≥ 9。
<i>laf</i>	(ローカル) INTEGER。配列 <i>af</i> の次元。 $laf \geq NB * (bwl + bwu) + 6 * (bwl + bwu) * (bwl + 2 * bwu)$ でなければならない。 <i>laf</i> が小さすぎる場合、エラーコードが返され、エラーにならない最小のサイズが <i>af</i> (1) に返される。
<i>work</i>	(ローカル) <i>a</i> と同じタイプ。ワークスペース配列、次元は <i>lwork</i> 。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。配列 <i>work</i> のサイズ。 $lwork \geq nrhs * (NB + 2 * bwl + 4 * bwu)$ でなければならない。

## 出力パラメータ

<i>ipiv</i>	(ローカル) INTEGER。配列。 <i>ipiv</i> の次元は <i>desca</i> (NB) 以上でなければならない。 ローカル因子分解用のピボットのインデックスが格納される。因子分解および解の算出を行う間、この配列の内容は変更すべきでない点に注意する。
<i>b</i>	終了時に、解の分散行列 <i>X</i> のローカル部分によって上書きされる。
<i>af</i>	(ローカル) REAL ( <i>psgbtrs</i> の場合) DOUBLE PRECISION ( <i>pdgbtrs</i> の場合) COMPLEX ( <i>pcgbtrs</i> の場合) DOUBLE COMPLEX ( <i>pzgbtrs</i> の場合)。  配列、次元は ( <i>laf</i> )。 補助非零要素空間。非零要素は、因子分解ルーチン <i>p?gbtrf</i> で作成され、 <i>af</i> に格納される。 因子分解ルーチンの後に <i>p?gbtrs</i> を使用して 1 次方程式の解を算出する場合、 <i>af</i> は因子分解の後に変更されてはならない点に注意する。

<code>work(1)</code>	終了時に、 <code>work(1)</code> には、最適なパフォーマンスを得るために必要な <code>lwork</code> の最小値が格納される。
<code>info</code>	INTEGER。 <code>info = 0</code> の場合、実行は正常に終了したことを示す。 <code>info &lt; 0</code> の場合： <code>i</code> 番目の引数が配列で、 <code>j</code> 番目の値が不正だった場合、 <code>info = -(i*100+j)</code> 。 <code>i</code> 番目の引数がスカラで値が不正だった場合、 <code>info = -i</code> 。

## p?potrs

コレスキー因子分解された対称/エルミート  
分散正定値行列の連立1次方程式を解く。

### 構文

```
call pspotrs( uplo, n, nrhs, a, ia, ja, desca, b, ib, jb, descb, info )
call pdpotrs( uplo, n, nrhs, a, ia, ja, desca, b, ib, jb, descb, info )
call pcpotrs( uplo, n, nrhs, a, ia, ja, desca, b, ib, jb, descb, info )
call pzpotsr( uplo, n, nrhs, a, ia, ja, desca, b, ib, jb, descb, info )
```

### 説明

ルーチン `p?potrs` は、次の分散連立1次方程式を  $X$  について解く。

$$\text{sub}(A) * X = \text{sub}(B)$$

ここで、 $\text{sub}(A) = A(\text{ia}:\text{ia}+n-1, \text{ja}:\text{ja}+n-1)$  は  $n \times n$  の実対称 / 複素エルミート正定値分散行列、 $\text{sub}(B)$  は分散行列  $B(\text{ib}:\text{ib}+n-1, \text{jb}:\text{jb}+\text{nrhs}-1)$  である。  
このルーチンは、次のコレスキー因子分解を使用する。

$$\text{sub}(A) = U^H U \quad \text{または} \quad \text{sub}(A) = L L^H$$

因子分解は、[p?potrf](#) によって行われる。

### 入力パラメータ

`uplo` (グローバル) CHARACTER\*1。  
'U' または 'L' でなければならない。

	$uplo = 'U'$ の場合、 $sub(A)$ の上三角部分が格納される。 $uplo = 'L'$ の場合、 $sub(A)$ の下三角部分が格納される。
$n$	(グローバル) INTEGER。 分散部分行列 $sub(A)$ の次数 ( $n \geq 0$ )。
$nrhs$	(グローバル) INTEGER。 右辺の数。分散部分行列 $sub(B)$ の列数 ( $nrhs \geq 0$ )。
$a, b$	(ローカル) REAL (pspotrs の場合) DOUBLE PRECISION (pdpotrs の場合) COMPLEX (pcpotrs の場合) DOUBLE COMPLEX (pzpotrs の場合)。 それぞれ、ローカルメモリにある、ローカル次元の配列 $a(lld\_a, LOC_c(ja+n-1))$ と $b(lld\_b, LOC_c(jb+nrhs-1))$ へのポインタ。
	配列 $a$ は、 $p?potrf$ によって行われたコレスキー因子分解 $sub(A) = LL^H$ または $sub(A) = U^H U$ で得られた係数 $L$ または $U$ を含む。 呼び出し時に、配列 $b$ は右辺 $sub(B)$ のローカル部分を含む。
$ia, ja$	(グローバル) INTEGER。 それぞれ、部分行列 $sub(A)$ の最初の行と最初の列を示す、グローバル配列 $A$ の行インデックスと列インデックス。
$desca$	(グローバルおよびローカル) INTEGER。配列、次元は ( $dlen\_$ )。 分散行列 $A$ の配列ディスクリプタ。
$ib, jb$	(グローバル) INTEGER。 それぞれ、部分行列 $sub(B)$ の最初の行と最初の列を示す、グローバル配列 $B$ の行インデックスと列インデックス。
$descb$	(グローバルおよびローカル) INTEGER。配列、次元は ( $dlen\_$ )。 分散行列 $B$ の配列ディスクリプタ。

## 出力パラメータ

$b$	解の行列 $X$ のローカル部分によって上書きされる。
$info$	INTEGER。 $info = 0$ の場合、実行は正常に終了したことを示す。 $info < 0$ の場合 : $i$ 番目の引数が配列で、 $j$ 番目の値が不正だった場合、 $info = -(i*100+j)$ 。 $i$ 番目の引数がスカラで値が不正だった場合、 $info = -i$ 。

## p?pbtrs

コレスキー因子分解された対称/エルミート  
正定値帯行列の連立1次方程式を解く。

### 構文

```
call pspbtrs( uplo, n, bw, nrhs, a, ja, desca, b, ib, descb, af, laf,
              work, lwork, info )
call pdpbtrs( uplo, n, bw, nrhs, a, ja, desca, b, ib, descb, af, laf,
              work, lwork, info )
call pcpbtrs( uplo, n, bw, nrhs, a, ja, desca, b, ib, descb, af, laf,
              work, lwork, info )
call pzpbtrs( uplo, n, bw, nrhs, a, ja, desca, b, ib, descb, af, laf,
              work, lwork, info )
```

### 説明

ルーチン p?pbtrs は、次の分散連立1次方程式を  $X$  について解く。

$$\text{sub}(A) * X = \text{sub}(B)$$

ここで、 $\text{sub}(A) = A(1:n, ja:ja+n-1)$  は  $n \times n$  の実対称 / 複素エルミート正定値帯分散行列、 $\text{sub}(B)$  は分散行列  $B(ib:ib+n-1, 1:nrhs)$  である。  
このルーチンは、次のコレスキー因子分解を使用する。

$$\text{sub}(A) = P U^H U P^T \quad \text{または} \quad \text{sub}(A) = P L L^H P^T$$

因子分解は、[p?pbtrf](#) によって行われる。

### 入力パラメータ

**uplo** (グローバル) CHARACTER\*1。  
'U' または 'L' でなければならない。  
  
 $uplo = 'U'$  の場合、 $\text{sub}(A)$  の上三角部分が格納される。  
 $uplo = 'L'$  の場合、 $\text{sub}(A)$  の下三角部分が格納される。

**n** (グローバル) INTEGER。  
分散部分行列  $\text{sub}(A)$  の次数 ( $n \geq 0$ )。

<i>bw</i>	(グローバル) INTEGER。 分散行列の優対角成分の数 ( <i>uplo</i> = 'U' の場合) または劣対角成分の数 ( <i>uplo</i> = 'L' の場合) ( <i>bw</i> ≥ 0)。
<i>nrhs</i>	(グローバル) INTEGER。 右辺の数。分散部分行列 <i>sub(B)</i> の列数 ( <i>nrhs</i> ≥ 0)。
<i>a, b</i>	(ローカル) REAL ( <i>pspbtrs</i> の場合) DOUBLE PRECISION ( <i>pdpbtrs</i> の場合) COMPLEX ( <i>pcpbtrs</i> の場合) DOUBLE COMPLEX ( <i>pzpbtrs</i> の場合)。 それぞれ、ローカルメモリにある、ローカル次元の配列 <i>a</i> ( <i>lld_a</i> , <i>LOC<sub>c</sub></i> ( <i>ja</i> + <i>n</i> -1)) と <i>b</i> ( <i>lld_b</i> , <i>LOC<sub>c</sub></i> ( <i>nrhs</i> -1)) へのポインタ。 配列 <i>a</i> は、 <i>p?pbtrf</i> によって返される、帯行列 <i>A</i> のコレスキー因子分解 $\text{sub}(A) = P U^H U P^T$ または $\text{sub}(A) = P L L^H P^T$ で得られた三角係数 <i>U</i> あるいは <i>L</i> を含む。  呼び出し時に、配列 <i>b</i> は、 <i>n</i> × <i>nrhs</i> の右辺の分散行列 <i>sub(B)</i> のローカル部分を含む。
<i>ja</i>	(グローバル) INTEGER。 ( <i>A</i> のすべて、または <i>A</i> の部分行列で) 処理される行列の先頭を指すグローバル配列 <i>A</i> のインデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。 分散行列 <i>A</i> の配列ディスクリプタ。 <i>desca</i> ( <i>dtype_</i> ) = 501 の場合、 <i>dlen_</i> ≥ 7。 <i>desca</i> ( <i>dtype_</i> ) = 1 の場合、 <i>dlen_</i> ≥ 9。
<i>ib</i>	(グローバル) INTEGER。 部分行列 <i>sub(B)</i> の最初の行を示す、グローバル配列 <i>B</i> の行インデックス。
<i>descb</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。 分散行列 <i>B</i> の配列ディスクリプタ。  <i>descb</i> ( <i>dtype_</i> ) = 502 の場合、 <i>dlen_</i> ≥ 7。 <i>descb</i> ( <i>dtype_</i> ) = 1 の場合、 <i>dlen_</i> ≥ 9。
<i>af, work</i>	(ローカル) 配列、 <i>a</i> と同じタイプ。 配列 <i>af</i> は、次元 ( <i>laf</i> ) の配列。補助非零要素空間を含む。非零要素は、因子分解ルーチン <a href="#">p?dbtrf</a> で作成され、 <i>af</i> に格納される。  配列 <i>work</i> は、次元 <i>lwork</i> のワークスペース配列。



<i>laf</i>	(ローカル) INTEGER。配列 <i>af</i> の次元。 $laf \geq nrhs * bw$ でなければならない。 <i>laf</i> が小さすぎる場合、エラーコードが返され、エラーにならない最小のサイズが <i>af</i> (1) に返される。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。配列 <i>work</i> のサイズ。 $lwork \geq bw^2$ でなければならない。

### 出力パラメータ

<i>b</i>	終了時に、 <i>info</i> =0 の場合、この配列には、 $n \times nrhs$ の解の分散行列 <i>X</i> のローカル部分が格納される。
<i>work</i> (1)	終了時に、 <i>work</i> (1) には、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が格納される。
<i>info</i>	INTEGER。 <i>info</i> =0 の場合、実行は正常に終了したことを示す。 <i>info</i> <0 の場合： <i>i</i> 番目の引数が配列で、 <i>j</i> 番目の値が不正だった場合、 $info = -(i * 100 + j)$ 。 <i>i</i> 番目の引数がスカラで値が不正だった場合、 $info = -i$ 。

## p?pttrs

p?pttrf によって行われた因子分解を使用して、  
対称(エルミート)正定値三重対角分散行列の  
連立1次方程式を解く。

### 構文

```
call pspttrs( n, nrhs, d, e, ja, desca, b, ib, descb, af, laf, work,
             lwork, info )
call pdpttrs( n, nrhs, d, e, ja, desca, b, ib, descb, af, laf, work,
             lwork, info )
call pcpttrs( uplo, n, nrhs, d, e, ja, desca, b, ib, descb, af, laf,
             work, lwork, info )
call pzpttrs( uplo, n, nrhs, d, e, ja, desca, b, ib, descb, af, laf,
             work, lwork, info )
```

## 説明

ルーチン `p?pttrs` は、次の分散連立 1 次方程式を  $X$  について解く。

$$\text{sub}(A) * X = \text{sub}(B)$$

ここで、 $\text{sub}(A) = A(1:n, ja:ja+n-1)$  は  $n \times n$  の実対称 / 複素エルミート正定値三重対角分散行列、 $\text{sub}(B)$  は分散行列  $B(ib:ib+n-1, 1:nrhs)$  である。  
このルーチンは、次の因子分解を使用する。

$$\text{sub}(A) = P L D L^H P^T \text{ または } \text{sub}(A) = P U^H D U P^T$$

因子分解は、[p?pttrf](#) によって行われる。

## 入力パラメータ

<code>uplo</code>	( グローバル、複素数型の場合のみ使用される ) CHARACTER*1。'U' または 'L' でなければならない。  <code>uplo = 'U'</code> の場合、 $\text{sub}(A)$ の上三角部分が格納される。 <code>uplo = 'L'</code> の場合、 $\text{sub}(A)$ の下三角部分が格納される。
<code>n</code>	( グローバル ) INTEGER。 分散部分行列 $\text{sub}(A)$ の次数 ( $n \geq 0$ )。
<code>nrhs</code>	( グローバル ) INTEGER。 右辺の数。分散部分行列 $\text{sub}(B)$ の列数 ( $nrhs \geq 0$ )。
<code>d, e</code>	( ローカル ) REAL ( <code>pspttrs</code> の場合 ) DOUBLE PRECISION ( <code>pdpttrs</code> の場合 ) COMPLEX ( <code>pcpttrs</code> の場合 ) DOUBLE COMPLEX ( <code>pzpttrs</code> の場合 )。  それぞれ、ローカルメモリにある、次元 ( <code>desca(nb_)</code> ) の配列へのポインタ。  これらの配列は、 <code>p?pttrf</code> によって返される因子分解の詳細を含む。
<code>ja</code>	( グローバル ) INTEGER。 ( $A$ のすべて、または $A$ の部分行列で ) 処理される行列の先頭を指すグローバル配列 $A$ のインデックス。
<code>desca</code>	( グローバルおよびローカル ) INTEGER。配列、次元は ( <code>dlen_</code> )。 分散行列 $A$ の配列ディスクリプタ。 <code>desca(dtype_) = 501</code> または <code>502</code> の場合、 <code>dlen_ <math>\geq 7</math></code> 。 <code>desca(dtype_) = 1</code> の場合、 <code>dlen_ <math>\geq 9</math></code> 。

<i>b</i>	(ローカル) <i>d</i> , <i>e</i> と同じタイプ。 ローカルメモリにある、ローカル次元 $b(1:d_b, LOC_c(nrhs))$ の配列へのポインタ。 呼び出し時に、配列 <i>b</i> は、 $n \times nrhs$ の右辺の分散行列 $\text{sub}(B)$ のローカル部分を含む。
<i>ib</i>	(グローバル) INTEGER。 ( <i>B</i> のすべて、または <i>B</i> の部分行列で) 処理される行列の最初の行を指すグローバル配列 <i>B</i> の行インデックス。
<i>descb</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。 分散行列 <i>B</i> の配列ディスクリプタ。  <i>descb(dtype_)</i> = 502 の場合、 <i>dlen_</i> $\geq 7$ 。 <i>descb(dtype_)</i> = 1 の場合、 <i>dlen_</i> $\geq 9$ 。
<i>af</i> , <i>work</i>	(ローカル) REAL (pspttrs の場合) DOUBLE PRECISION (pdpttrs の場合) COMPLEX (pcpttrs の場合) DOUBLE COMPLEX (pzpttrs の場合)。  それぞれ、次元 ( <i>laf</i> ) と ( <i>lwork</i> ) の配列。 配列 <i>af</i> は、補助非零要素空間を含む。非零要素は、因子分解ルーチン <i>p?pttrf</i> で作成され、 <i>af</i> に格納される。  配列 <i>work</i> はワークスペース配列である。
<i>laf</i>	(ローカル) INTEGER。配列 <i>af</i> の次元。 <i>laf</i> $\geq NB+2$ でなければならない。  <i>laf</i> が小さすぎる場合、エラーコードが返され、エラーにならない最小のサイズが <i>af</i> (1) に返される。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。配列 <i>work</i> のサイズ。 <i>lwork</i> $\geq (10+2*\min(100, nrhs))*NPCOL+4*nrhs$ でなければならない。

### 出力パラメータ

<i>b</i>	終了時に、この配列には解の分散行列 <i>X</i> のローカル部分が格納される。
<i>work</i> (1)	終了時に、 <i>work</i> (1) には、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が格納される。

*info*                    INTEGER。 *info* = 0 の場合、実行は正常に終了したことを示す。  
*info* < 0 の場合：  
    *i* 番目の引数が配列で、*j* 番目の値が不正だった場合、  
    *info* = -(*i*\*100+*j*)。 *i* 番目の引数がスカラーで値が不正だった場合、  
    *info* = -*i*。

---

### p?dttrs

p?dttrf によって行われた因子分解を使用して、  
対角優位三重対角分散行列の連立1次方程式を  
解く。

---

#### 構文

```
call psdttrs( trans, n, nrhs, dl, d, du, ja, desca, b, ib, descb, af,  
             laf, work, lwork, info )  
call pddttrs( trans, n, nrhs, dl, d, du, ja, desca, b, ib, descb, af,  
             laf, work, lwork, info )  
call pcdttrs( trans, n, nrhs, dl, d, du, ja, desca, b, ib, descb, af,  
             laf, work, lwork, info )  
call pzdttrs( trans, n, nrhs, dl, d, du, ja, desca, b, ib, descb, af,  
             laf, work, lwork, info )
```

#### 説明

ルーチン p?dttrs は、以下のいずれかの連立方程式を  $X$  について解く。

$$\text{sub}(A) * X = \text{sub}(B),$$

$$(\text{sub}(A))^T * X = \text{sub}(B), \text{ または }$$

$$(\text{sub}(A))^H * X = \text{sub}(B)$$

ここで、 $\text{sub}(A) = A(1:n, ja:ja+n-1)$  は対角優位三重対角分散行列、 $\text{sub}(B)$  は分散行列  
 $B(ib:ib+n-1, 1:nrhs)$  である。

このルーチンは、[p?dttrf](#) によって行われた  $LU$  因子分解を使用する。

## 入力パラメータ

<i>trans</i>	(グローバル) CHARACTER*1。 'N'、'T'、または 'C' のいずれかでなければならない。 方程式の形式を指定する。  <i>trans</i> = 'N' の場合、 $\text{sub}(A) * X = \text{sub}(B)$ を $X$ について解く。 <i>trans</i> = 'T' の場合、 $\text{sub}(A)^T * X = \text{sub}(B)$ を $X$ について解く。 <i>trans</i> = 'C' の場合、 $\text{sub}(A)^H * X = \text{sub}(B)$ を $X$ について解く。
<i>n</i>	(グローバル) INTEGER。 分散部分行列 $\text{sub}(A)$ の次数 ( $n \geq 0$ )。
<i>nrhs</i>	(グローバル) INTEGER。 右辺の数。分散部分行列 $\text{sub}(B)$ の列数 ( $nrhs \geq 0$ )。
<i>dl, d, du</i>	(ローカル) REAL (psdttrs の場合) DOUBLE PRECISION (pddttrs の場合) COMPLEX (pcdttrs の場合) DOUBLE COMPLEX (pzdttrs の場合)。  それぞれ、次元 ( $\text{desca}(nb\_)$ ) のローカル配列へのポインタ。  呼び出し時に、これらの配列は因子分解の詳細を含む。全体的に、 <i>dl</i> (1) および <i>du</i> ( <i>n</i> ) は参照されず、 <i>dl</i> および <i>du</i> は <i>d</i> とアライメントされなければならない。
<i>ja</i>	(グローバル) INTEGER。 ( <i>A</i> のすべて、または <i>A</i> の部分行列で) 処理される行列の先頭を指すグローバル配列 <i>A</i> のインデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。 分散行列 <i>A</i> の配列ディスクリプタ。 <i>desca</i> ( <i>dtype_</i> ) = 501 または 502 の場合、 <i>dlen_</i> $\geq 7$ 。 <i>desca</i> ( <i>dtype_</i> ) = 1 の場合、 <i>dlen_</i> $\geq 9$ 。
<i>b</i>	(ローカル) <i>d</i> と同じタイプ。  ローカルメモリにある、ローカル次元 $b(11d\_b, LOC_c(nrhs))$ の配列へのポインタ。 呼び出し時に、配列 <i>b</i> は、 $n \times nrhs$ の右辺の分散行列 $\text{sub}(B)$ のローカル部分を含む。

<i>ib</i>	(グローバル) INTEGER。 ( <i>B</i> のすべて、または <i>B</i> の部分行列で) 処理される行列の最初の行を指すグローバル配列 <i>B</i> の行インデックス。
<i>descb</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。 分散行列 <i>B</i> の配列ディスクリプタ。  <i>descb(dtype_)</i> = 502 の場合、 <i>dlen_</i> ≥ 7。 <i>descb(dtype_)</i> = 1 の場合、 <i>dlen_</i> ≥ 9。
<i>af, work</i>	(ローカル) REAL ( <i>psdttrs</i> の場合) DOUBLE PRECISION ( <i>pddttrs</i> の場合) COMPLEX ( <i>pcdttrs</i> の場合) DOUBLE COMPLEX ( <i>pzdttrs</i> の場合)。  それぞれ、次元 ( <i>laf</i> ) と ( <i>lwork</i> ) の配列。 配列 <i>af</i> は、補助非零要素空間を含む。非零要素は、因子分解ルーチン <i>p?dttrf</i> で作成され、 <i>af</i> に格納される。因子分解ルーチンの後に <i>p?dttrs</i> を使用して 1 次方程式の解を算出する場合、 <i>af</i> は因子分解の後に変更されてはならない点に注意する。  配列 <i>work</i> はワークスペース配列である。
<i>laf</i>	(ローカル) INTEGER。配列 <i>af</i> の次元。 <i>laf</i> ≥ NB*( <i>bw1</i> + <i>bwu</i> )+6*( <i>bw1</i> + <i>bwu</i> )*( <i>bw1</i> +2* <i>bwu</i> ) でなければならない。  <i>laf</i> が小さすぎる場合、エラーコードが返され、エラーにならない最小のサイズが <i>af</i> (1) に返される。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。配列 <i>work</i> のサイズ。 <i>lwork</i> ≥ 10*NPCOL+4*nrhs でなければならない。

## 出力パラメータ

<i>b</i>	終了時に、この配列には解の分散行列 <i>X</i> のローカル部分が格納される。
<i>work</i> (1)	終了時に、 <i>work</i> (1) には、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が格納される。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> < 0 の場合：  <i>i</i> 番目の引数が配列で、 <i>j</i> 番目の値が不正だった場合、 <i>info</i> = -( <i>i</i> *100+ <i>j</i> )。 <i>i</i> 番目の引数がスカラーで値が不正だった場合、 <i>info</i> = - <i>i</i> 。

## p?dbtrs

p?dbtrf によって行われた因子分解を使用して、  
対角優位帯分散行列の連立 1 次方程式を解く。

### 構文

```
call psdbtrs( trans, n, bwl, bwu, nrhs, a, ja, desca, b, ib, descb, af,
              laf, work, lwork, info )
call pddbtrs( trans, n, bwl, bwu, nrhs, a, ja, desca, b, ib, descb, af,
              laf, work, lwork, info )
call pcdbtrs( trans, n, bwl, bwu, nrhs, a, ja, desca, b, ib, descb, af,
              laf, work, lwork, info )
call pzdbtrs( trans, n, bwl, bwu, nrhs, a, ja, desca, b, ib, descb, af,
              laf, work, lwork, info )
```

### 説明

ルーチン p?dbtrs は、以下のいずれかの連立方程式を  $X$  について解く。

$$\text{sub}(A) * X = \text{sub}(B),$$

$$(\text{sub}(A))^T * X = \text{sub}(B), \text{ または}$$

$$(\text{sub}(A))^H * X = \text{sub}(B)$$

ここで、 $\text{sub}(A) = A(1:n, ja:ja+n-1)$  は対角優位帯分散行列、 $\text{sub}(B)$  は分散行列  $B(ib:ib+n-1, 1:nrhs)$  である。

このルーチンは、[p?dbtrf](#) によって行われた  $LU$  因子分解を使用する。

### 入力パラメータ

*trans* (グローバル) CHARACTER\*1。  
'N'、'T'、または 'C' のいずれかでなければならない。  
方程式の形式を指定する。  
*trans* = 'N' の場合、 $\text{sub}(A) * X = \text{sub}(B)$  を  $X$  について解く。  
*trans* = 'T' の場合、 $\text{sub}(A)^T * X = \text{sub}(B)$  を  $X$  について解く。  
*trans* = 'C' の場合、 $\text{sub}(A)^H * X = \text{sub}(B)$  を  $X$  について解く。

<i>n</i>	(グローバル) INTEGER。 分散部分行列 <i>sub(A)</i> の次数 ( $n \geq 0$ )。
<i>bw1</i>	(グローバル) INTEGER。 <i>A</i> の帯内の劣対角成分の数 ( $0 \leq bw1 \leq n-1$ )。
<i>bwu</i>	(グローバル) INTEGER。 <i>A</i> の帯内の優対角成分の数 ( $0 \leq bwu \leq n-1$ )。
<i>nrhs</i>	(グローバル) INTEGER。 右辺の数。分散部分行列 <i>sub(B)</i> の列数 ( $nrhs \geq 0$ )。
<i>a, b</i>	(ローカル)  REAL (psdbtrs の場合) DOUBLE PRECISION (pddbtrs の場合) COMPLEX (pcdbtrs の場合) DOUBLE COMPLEX (pzdbtrs の場合)。  それぞれ、ローカルメモリにある、ローカル次元の配列 <i>a(lld_a, LOC_c(ja+n-1))</i> と <i>b(lld_b, LOC_c(nrhs))</i> へのポインタ。  呼び出し時に、配列 <i>a</i> は、p?dbtrf によって行われた帯行列 <i>A</i> の <i>LU</i> 因子分解の詳細を含む。  呼び出し時に、配列 <i>b</i> は右辺の分散行列 <i>sub(B)</i> のローカル部分を含 む。
<i>ja</i>	(グローバル) INTEGER。 ( <i>A</i> のすべて、または <i>A</i> の部分行列で) 処理される行列の先頭を指すグ ローバル配列 <i>A</i> のインデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。 分散行列 <i>A</i> の配列ディスクリプタ。 <i>desca(dtype_) = 501</i> の場合、 <i>dlen_</i> $\geq 7$ 。 <i>desca(dtype_) = 1</i> の場合、 <i>dlen_</i> $\geq 9$ 。
<i>ib</i>	(グローバル) INTEGER。 ( <i>B</i> のすべて、または <i>B</i> の部分行列で) 処理される行列の最初の行を指 すグローバル配列 <i>B</i> の行インデックス。
<i>descb</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。 分散行列 <i>B</i> の配列ディスクリプタ。 <i>descb(dtype_) = 502</i> の場合、 <i>dlen_</i> $\geq 7$ 。 <i>descb(dtype_) = 1</i> の場合、 <i>dlen_</i> $\geq 9$ 。



<i>af, work</i>	<p>(ローカル)</p> <p>REAL (psdbtrs の場合)</p> <p>DOUBLE PRECISION (pddbtrs の場合)</p> <p>COMPLEX (pcdbtrs の場合)</p> <p>DOUBLE COMPLEX (pzdbtrs の場合)。</p> <p>それぞれ、次元 (<i>laf</i>) と (<i>lwork</i>) の配列。</p> <p>配列 <i>af</i> は、補助非零要素空間を含む。非零要素は、因子分解ルーチン <i>p?dbtrf</i> で作成され、<i>af</i> に格納される。</p> <p>配列 <i>work</i> はワークスペース配列である。</p>
<i>laf</i>	<p>(ローカル) INTEGER。配列 <i>af</i> の次元。</p> <p><math>laf \geq NB * (bwl + bwu) + 6 * (\max(bwl, bwu))^2</math> でなければならない。</p> <p><i>laf</i> が小さすぎる場合、エラーコードが返され、エラーにならない最小のサイズが <i>af</i>(1) に返される。</p>
<i>lwork</i>	<p>(ローカルまたはグローバル) INTEGER。配列 <i>work</i> のサイズ。</p> <p><math>lwork \geq (\max(bwl, bwu))^2</math> でなければならない。</p>

### 出力パラメータ

<i>b</i>	終了時に、この配列には解の分散行列 <i>X</i> のローカル部分が格納される。
<i>work</i> (1)	終了時に、 <i>work</i> (1) には、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が格納される。
<i>info</i>	<p>INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。</p> <p><i>info</i> &lt; 0 の場合：</p> <p><i>i</i> 番目の引数が配列で、<i>j</i> 番目の値が不正だった場合、  <i>info</i> = -(<i>i</i>*100+<i>j</i>)。 <i>i</i> 番目の引数がスカラで値が不正だった場合、  <i>info</i> = -<i>i</i>。</p>

## p?trtrs

三角分散行列の連立1次方程式を解く。

### 構文

```
call pstrtrs(uplo, trans, diag, n, nrhs, a, ia, ja, desca, b, ib, jb,
             descb, info)
call pdtrtrs(uplo, trans, diag, n, nrhs, a, ia, ja, desca, b, ib, jb,
             descb, info)
call pcttrtrs(uplo, trans, diag, n, nrhs, a, ia, ja, desca, b, ib, jb,
              descb, info)
call pztrtrs(uplo, trans, diag, n, nrhs, a, ia, ja, desca, b, ib, jb,
             descb, info)
```

### 説明

このルーチンは、以下のいずれかの連立方程式を  $X$  について解く。

$$\text{sub}(A) * X = \text{sub}(B),$$

$$(\text{sub}(A))^T * X = \text{sub}(B), \text{ または }$$

$$(\text{sub}(A))^H * X = \text{sub}(B)$$

ここで、 $\text{sub}(A) = A(\text{ia}:\text{ia}+n-1, \text{ja}:\text{ja}+n-1)$  は次数  $n$  の三角分散行列、 $\text{sub}(B)$  は分散行列  $B(\text{ib}:\text{ib}+n-1, \text{jb}:\text{jb}+\text{nrhs}-1)$  である。

$\text{sub}(A)$  が非特異であることを確認するチェックが行われる。

### 入力パラメータ

**uplo** (グローバル) CHARACTER\*1。  
 'U' または 'L' でなければならない。  
 $\text{sub}(A)$  が上三角と下三角のどちらであることを示す。  
 uplo = 'U' の場合、 $\text{sub}(A)$  は上三角である。  
 uplo = 'L' の場合、 $\text{sub}(A)$  は下三角である。

**trans** (グローバル) CHARACTER\*1。  
 'N'、'T'、または 'C' のいずれかでなければならない。  
 方程式の形式を指定する。

	$trans = 'N'$ の場合、 $sub(A)*X = sub(B)$ を $X$ について解く。 $trans = 'T'$ の場合、 $sub(A)^T * X = sub(B)$ を $X$ について解く。 $trans = 'C'$ の場合、 $sub(A)^H * X = sub(B)$ を $X$ について解く。
<i>diag</i>	(グローバル) CHARACTER*1。 'N' または 'U' でなければならない。  $diag = 'N'$ の場合、 $sub(A)$ は単位三角ではない。 $diag = 'U'$ の場合、 $sub(A)$ は単位三角である。
<i>n</i>	(グローバル) INTEGER。 分散部分行列 $sub(A)$ の次数 ( $n \geq 0$ )。
<i>nrhs</i>	(グローバル) INTEGER。 右辺の数。すなわち、分散行列 $sub(B)$ の列数 ( $nrhs \geq 0$ )。
<i>a, b</i>	(ローカル) REAL ( $pstrtrs$ の場合) DOUBLE PRECISION ( $pdtrtrs$ の場合) COMPLEX ( $pctrtrs$ の場合) DOUBLE COMPLEX ( $pztrtrs$ の場合)。  それぞれ、ローカルメモリにある、ローカル次元の配列 $a(lld\_a, LOC_c(ja+n-1))$ と $b(lld\_b, LOC_c(jb+nrhs-1))$ へのポインタ。  配列 $a$ は、三角分散行列 $sub(A)$ のローカル部分を含む。 $uplo = 'U'$ の場合、 $sub(A)$ の先頭の $n \times n$ 上三角部分は上三角行列を含む。 $sub(A)$ の厳密な下三角部分は参照されない。 $uplo = 'L'$ の場合、 $sub(A)$ の先頭の $n \times n$ 下三角部分は下三角行列を含む。 $sub(A)$ の厳密な上三角部分は参照されない。 $diag = 'U'$ の場合、 $sub(A)$ の対角成分も参照されず、1 と仮定される。  呼び出し時に、配列 $b$ は右辺の分散行列 $sub(B)$ のローカル部分を含む。
<i>ia, ja</i>	(グローバル) INTEGER。 それぞれ、部分行列 $sub(A)$ の最初の行と最初の列を示す、グローバル配列 $A$ の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( $dlen\_$ )。 分散行列 $A$ の配列ディスクリプタ。

*ib, jb* (グローバル) INTEGER。  
それぞれ、部分行列  $\text{sub}(B)$  の最初の行と最初の列を示す、グローバル配列  $B$  の行インデックスと列インデックス。

*descb* (グローバルおよびローカル) INTEGER。配列、次元は (*dlen\_*)。  
分散行列  $B$  の配列ディスクリプタ。

### 出力パラメータ

*b* 終了時に、*info* = 0 の場合、 $\text{sub}(B)$  は解の行列  $X$  によって上書きされる。

*info* INTEGER。 *info* = 0 の場合、実行は正常に終了したことを示す。  
*info* < 0 の場合：  
*i* 番目の引数が配列で、*j* 番目の値が不正だった場合、  
*info* = -(*i*\*100+*j*)。 *i* 番目の引数がスカラで値が不正だった場合、  
*info* = -*i*。  
*info* > 0 の場合：  
*info* = *i* の場合、 $\text{sub}(A)$  の *i* 番目の対角成分はゼロで、部分行列が特異であり、解  $X$  が計算されていないことを示す。

### 条件数を推定するためのルーチン

このセクションでは、行列の条件数を推定するための ScaLAPACK ルーチンについて説明する。条件数は、連立 1 次方程式の解の誤差を解析するために使用される。行列がほとんど特異である (ゼロに近い) 場合、条件数が非常に大きくなることがあるため、これらのルーチンでは、実際には条件数の逆数を計算する。

---

## p?gecon

一般分散行列の条件数の逆数を、1- ノルムまたは無限ノルムのいずれかで推定する。

---

### 構文

```
call psgecon( norm, n, a, ia, ja, desca, anorm, rcond, work, lwork,  
             iwork, liwork, info )
```

```

call pdgecon( norm, n, a, ia, ja, desca, anorm, rcond, work, lwork,
              iwork, liwork, info )
call pcgecon( norm, n, a, ia, ja, desca, anorm, rcond, work, lwork,
              rwork, lrwork, info )
call pzgecon( norm, n, a, ia, ja, desca, anorm, rcond, work, lwork,
              rwork, lrwork, info )

```

## 説明

このルーチンは、[p?getrf](#) によって行われた LU 因子分解を使用して、一般分散実 / 複素行列  $\text{sub}(A) = A(ia:ia+n-1, ja:ja+n-1)$  の条件数の逆数を、1- ノルムまたは無限ノルムのいずれかで推定する。

推定値が  $\|(\text{sub}(A))^{-1}\|$  について得られる。次に、条件数の逆数が、

$$rcond = \text{として計算される。} \quad \frac{1}{\|\text{sub}(A)\| \times \|(\text{sub}(A))^{-1}\|}$$

## 入力パラメータ

**norm** (グローバル) CHARACTER\*1。  
 '1'、'0'、または 'I' のいずれかでなければならない。  
 1- ノルムの条件数と無限ノルムの条件数のどちらを使用するかを指定する。  
 norm = '1' または '0' の場合、1- ノルムが使用される。  
 norm = 'I' の場合、無限ノルムが使用される。

**n** (グローバル) INTEGER。  
 分散部分行列  $\text{sub}(A)$  の次数 ( $n \geq 0$ )。

**a** (ローカル)  
 REAL (psgecon の場合)  
 DOUBLE PRECISION (pdgecon の場合)  
 COMPLEX (pcgecon の場合)  
 DOUBLE COMPLEX (pzgecon の場合)。  
 ローカルメモリにある、次元  $a(1:d_a, LOC_c(ja+n-1))$  の配列へのポインタ。  
 配列 **a** は、因子分解  $\text{sub}(A) = P L U$  で得られた係数  $L$  と  $U$  のローカル部分を含む。 $L$  の単位対角成分は格納されない。

<i>ia, ja</i>	(グローバル) INTEGER。 それぞれ、部分行列 <i>sub(A)</i> の最初の行と最初の列を示す、グローバル配列 <i>A</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。 分散行列 <i>A</i> の配列ディスクリプタ。
<i>anorm</i>	(グローバル) REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 <i>norm</i> = '1' または '0' の場合、元の分散行列 <i>sub(A)</i> の 1- ノルム。 <i>norm</i> = 'I' の場合、元の分散行列 <i>sub(A)</i> の無限ノルム。
<i>work</i>	(ローカル) REAL ( <i>psgecon</i> の場合) DOUBLE PRECISION ( <i>pdgecon</i> の場合) COMPLEX ( <i>pcgecon</i> の場合) DOUBLE COMPLEX ( <i>pzgecon</i> の場合)。 次元 ( <i>lwork</i> ) の配列 <i>work</i> はワークスペース配列。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。配列 <i>work</i> の次元。 実数型の場合: $lwork \geq 2 * LOC_r(n + \text{mod}(ia - 1, mb\_a)) +$ $2 * LOC_c(n + \text{mod}(ja - 1, nb\_a)) +$ $\max(2, \max(nb\_a * \max(1, \text{ceil}(NPROW - 1, NPCOL)),$ $LOC_c(n + \text{mod}(ja - 1, nb\_a)) + nb\_a * \max(1, \text{ceil}(NPCOL - 1,$ $NPROW)))$ でなければならない。 複素数型の場合: $lwork \geq 2 * LOC_r(n + \text{mod}(ia - 1, mb\_a)) +$ $\max(2, \max(nb\_a * \text{ceil}(NPROW - 1, NPCOL),$ $LOC_c(n + \text{mod}(ja - 1, nb\_a)) + nb\_a * \text{ceil}(NPCOL - 1, NPROW)))$ でなければならない。 <i>LOC<sub>r</sub></i> および <i>LOC<sub>c</sub></i> の値は、ScaLAPACK ツール関数 <i>numroc</i> を使用して計算できる。NPROW および NPCOL は、サブルーチン <i>blacs_gridinfo</i> を呼び出して決定できる。
<i>iwork</i>	(ローカル) INTEGER。 ワークスペース配列、次元は ( <i>liwork</i> )。実数型でのみ使用される。
<i>liwork</i>	(ローカルまたはグローバル) INTEGER。 配列 <i>iwork</i> の次元。実数型でのみ使用される。 <i>liwork</i> $\geq LOC_r(n + \text{mod}(ia - 1, mb\_a))$ でなければならない。

*rwork* (ローカル) REAL (pcgecon の場合)  
DOUBLE PRECISION (pzgecon の場合)  
ワークスペース配列、次元は (*lrwork*)。複素数型でのみ使用される。

*lrwork* (ローカルまたはグローバル) INTEGER。  
配列 *rwork* の次元。複素数型でのみ使用される。  
 $lrwork \geq 2 * LOC_c(n + \text{mod}(ja-1, nb\_a))$  でなければならない。

### 出力パラメータ

*rcond* (グローバル) REAL (単精度の場合)。  
DOUBLE PRECISION (倍精度の場合)。  
分散行列 *sub(A)* の条件数の逆数。説明を参照。

*work(1)* 終了時に、*work(1)* には、最適なパフォーマンスを得るために必要な *lwork* の最小値が格納される。

*iwork(1)* 終了時に、*iwork(1)* には、最適なパフォーマンスを得るために必要な *liwork* の最小値が格納される (実数型の場合)。

*rwork(1)* 終了時に、*rwork(1)* には、最適なパフォーマンスを得るために必要な *lrwork* の最小値が格納される (複素数型の場合)。

*info* (グローバル) INTEGER。 *info* = 0 の場合、実行は正常に終了したことを示す。

*info* < 0 の場合：

*i* 番目の引数が配列で、*j* 番目の値が不正だった場合、  
*info* = -(*i*\*100+*j*)。 *i* 番目の引数がスカラーで値が不正だった場合、  
*info* = -*i*。

## p?pocon

対称/エルミート正定値分散行列の条件数の  
逆数を (1- ノルムで) 推定する。

### 構文

```
call pspocon( uplo, n, a, ia, ja, desca, anorm, rcond, work, lwork,
              iwork, liwork, info )
call pdpocon( uplo, n, a, ia, ja, desca, anorm, rcond, work, lwork,
              iwork, liwork, info )
```

```
call pcpocon( uplo, n, a, ia, ja, desca, anorm, rcond, work, lwork,
             rwork, lrwork, info )

call pzpocon( uplo, n, a, ia, ja, desca, anorm, rcond, work, lwork,
             rwork, lrwork, info )
```

## 説明

このルーチンは、[p?potrf](#) によって行われたコレスキー因子分解  $\text{sub}(A) = U^H U$  または  $\text{sub}(A) = LL^H$  を使用して、実対称 / 複素エルミート正定値分散行列  $\text{sub}(A) = A(ia:ia+n-1, ja:ja+n-1)$  の条件数の逆数を (1- ノルムで) 推定する。

推定値が  $\|(\text{sub}(A))^{-1}\|$  について得られる。次に、条件数の逆数が、

$$rcond = \frac{1}{\|\text{sub}(A)\| \times \|(\text{sub}(A))^{-1}\|}$$

## 入力パラメータ

**uplo** (グローバル) CHARACTER\*1。  
'U' または 'L' でなければならない。

**sub(A)** に格納されている係数が上三角と下三角のどちらであるかを指定する。

**uplo = 'U'** の場合、**sub(A)** には、コレスキー因子分解  $\text{sub}(A) = U^H U$  の上三角係数  $U$  が格納される。

**uplo = 'L'** の場合、**sub(A)** には、コレスキー因子分解  $\text{sub}(A) = LL^H$  の下三角係数  $L$  が格納される。

**n** (グローバル) INTEGER。  
分散部分行列 **sub(A)** の次数 ( $n \geq 0$ )。

**a** (ローカル)  
REAL (pspocon の場合)  
DOUBLE PRECISION (pdpocon の場合)  
COMPLEX (pcpocon の場合)  
DOUBLE COMPLEX (pzpocon の場合)。

ローカルメモリにある、次元  $a(11d\_a, LOC_c(ja+n-1))$  の配列へのポインタ。

配列 **a** は、[p?potrf](#) によって行われたコレスキー因子分解  $\text{sub}(A) = U^H U$  または  $\text{sub}(A) = LL^H$  で得られた係数  $U$  または  $L$  を含む。



<i>ia, ja</i>	(グローバル) INTEGER。 それぞれ、部分行列 <i>sub(A)</i> の最初の行と最初の列を示す、グローバル配列 <i>A</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。 配列、次元は ( <i>dlen_</i> )。分散行列 <i>A</i> の配列ディスクリプタ。
<i>anorm</i>	(グローバル) REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 対称/エルミート分散行列 <i>sub(A)</i> の 1- ノルム。
<i>work</i>	(ローカル) REAL ( <i>pspocon</i> の場合) DOUBLE PRECISION ( <i>pdpocon</i> の場合) COMPLEX ( <i>pcpocon</i> の場合) DOUBLE COMPLEX ( <i>pzpocon</i> の場合)。  次元 ( <i>lwork</i> ) の配列 <i>work</i> はワークスペース配列。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。配列 <i>work</i> の次元。 実数型の場合: $lwork \geq 2*LOC_r(n+\text{mod}(ia-1, mb\_a)) +$ $2*LOC_c(n+\text{mod}(ja-1, nb\_a)) +$ $\max(2, \max(nb\_a*\text{ceil}(NPROW-1, NPCOL),$ $LOC_c(n+\text{mod}(ja-1, nb\_a)) + nb\_a*\text{ceil}(NPCOL-1, NPROW)))$ でなければならない。 複素数型の場合: $lwork \geq 2*LOC_r(n+\text{mod}(ia-1, mb\_a)) +$ $\max(2, \max(nb\_a*\max(1, \text{ceil}(NPROW-1, NPCOL)),$ $LOC_c(n+\text{mod}(ja-1, nb\_a)) +$ $nb\_a*\max(1, \text{ceil}(NPCOL-1, NPROW))))$ でなければならない。
<i>iwork</i>	(ローカル) INTEGER。 ワークスペース配列、次元は ( <i>liwork</i> )。実数型でのみ使用される。
<i>liwork</i>	(ローカルまたはグローバル) INTEGER。 配列 <i>iwork</i> の次元。実数型でのみ使用される。 $liwork \geq LOC_r(n+\text{mod}(ia-1, mb\_a))$ でなければならない。
<i>rwork</i>	(ローカル) REAL ( <i>pcpocon</i> の場合) DOUBLE PRECISION ( <i>pzpocon</i> の場合) ワークスペース配列、次元は ( <i>lrwork</i> )。複素数型でのみ使用される。

*lrwork* (ローカルまたはグローバル) INTEGER。  
配列 *rwork* の次元。複素数型でのみ使用される。  
 $lrwork \geq 2 * LOC_c(n + \text{mod}(ja-1, nb\_a))$  でなければならない。

### 出力パラメータ

*rcond* (グローバル) REAL (単精度の場合)。  
DOUBLE PRECISION (倍精度の場合)。  
分散行列 *sub(A)* の条件数の逆数。

*work(1)* 終了時に、*work(1)* には、最適なパフォーマンスを得るために必要な *lwork* の最小値が格納される。

*iwork(1)* 終了時に、*iwork(1)* には、最適なパフォーマンスを得るために必要な *liwork* の最小値が格納される (実数型の場合)。

*rwork(1)* 終了時に、*rwork(1)* には、最適なパフォーマンスを得るために必要な *lrwork* の最小値が格納される (複素数型の場合)。

*info* (グローバル) INTEGER。 *info* = 0 の場合、実行は正常に終了したことを示す。

*info* < 0 の場合：

*i* 番目の引数が配列で、*j* 番目の値が不正だった場合、  
 $info = -(i * 100 + j)$ 。*i* 番目の引数がスカラで値が不正だった場合、  
 $info = -i$ 。

---

## p?trcon

三角分散行列の条件数の逆数を、1- ノルムまたは無限ノルムのいずれかで推定する。

---

### 構文

```
call pstrcon( norm, uplo, diag, n, a, ia, ja, desca, rcond, work, lwork,  
             iwork, liwork, info )  
call pdtrcon( norm, uplo, diag, n, a, ia, ja, desca, rcond, work, lwork,  
             iwork, liwork, info )  
call pctrcon( norm, uplo, diag, n, a, ia, ja, desca, rcond, work, lwork,  
             rwork, lrwork, info )
```

```
call pztrcon( norm, uplo, diag, n, a, ia, ja, desca, rcond, work, lwork,
             rwork, lrwork, info )
```

## 説明

このルーチンは、三角分散行列  $\text{sub}(A) = A(ia:ia+n-1, ja:ja+n-1)$  の条件数の逆数を、1- ノルムまたは無限ノルムのいずれかで推定する。

$\text{sub}(A)$  のノルムが計算され、推定値が  $\|(\text{sub}(A))^{-1}\|$  に対して得られる。次に、条件数の逆数が

$$rcond = \text{として計算される。} \quad \frac{1}{\|\text{sub}(A)\| \times \|(\text{sub}(A))^{-1}\|}$$

## 入力パラメータ

*norm* (グローバル) CHARACTER\*1。  
 '1'、'O'、または 'I' のいずれかでなければならない。  
 1- ノルムの条件数と無限ノルムの条件数のどちらを使用するかを指定する。  
*norm* = '1' または 'O' の場合、1- ノルムが使用される。  
*norm* = 'I' の場合、無限ノルムが使用される。

*uplo* (グローバル) CHARACTER\*1。  
 'U' または 'L' でなければならない。  
*uplo* = 'U' の場合、 $\text{sub}(A)$  は上三角部分である。  
*uplo* = 'L' の場合、 $\text{sub}(A)$  は下三角部分である。

*diag* (グローバル) CHARACTER\*1。  
 'N' または 'U' でなければならない。  
*diag* = 'N' の場合、 $\text{sub}(A)$  は単位三角ではない。  
*diag* = 'U' の場合、 $\text{sub}(A)$  は単位三角である。

*n* (グローバル) INTEGER。  
 分散部分行列  $\text{sub}(A)$  の次数 ( $n \geq 0$ )。

*a* (ローカル)  
 REAL (pztrcon の場合)  
 DOUBLE PRECISION (pdtrcon の場合)  
 COMPLEX (pctrcon の場合)  
 DOUBLE COMPLEX (pztrcon の場合)。

ローカルメモリにある、次元  $a(lld\_a, LOC_c(ja+n-1))$  の配列へのポインタ。

配列  $a$  は、三角分散行列  $sub(A)$  のローカル部分を含む。  
 $uplo = 'U'$  の場合、この分散行列の先頭の  $n \times n$  上三角部分は上三角行列を含む。厳密な下三角部分は参照されない。

$uplo = 'L'$  の場合、この分散行列の先頭の  $n \times n$  下三角部分は下三角行列を含む。厳密な上三角部分は参照されない。

$diag = 'U'$  の場合、 $sub(A)$  の対角成分も参照されず、1 と仮定される。

$ia, ja$	(グローバル) INTEGER。 それぞれ、部分行列 $sub(A)$ の最初の行と最初の列を示す、グローバル配列 $A$ の行インデックスと列インデックス。
$desca$	(グローバルおよびローカル) INTEGER。配列、次元は ( $dlen\_$ )。分散行列 $A$ の配列ディスクリプタ。
$work$	(ローカル) REAL ( $pstrcon$ の場合) DOUBLE PRECISION ( $pdtrcon$ の場合) COMPLEX ( $pctrcon$ の場合) DOUBLE COMPLEX ( $pztrcon$ の場合)。 次元 ( $lwork$ ) の配列 $work$ はワークスペース配列。
$lwork$	(ローカルまたはグローバル) INTEGER。配列 $work$ の次元。

実数型の場合:

$lwork$  は  

$$lwork \geq 2 * LOC_r(n + \text{mod}(ia-1, mb\_a)) +$$

$$LOC_c(n + \text{mod}(ja-1, nb\_a)) +$$

$$\max(2, \max(nb\_a * \max(1, \text{ceil}(NPROW-1, NPCOL)),$$

$$LOC_c(n + \text{mod}(ja-1, nb\_a)) +$$

$$nb\_a * \max(1, \text{ceil}(NPCOL-1, NPROW)))$$

でなければならない。

複素数型の場合:

$lwork$  は  

$$lwork \geq 2 * LOC_r(n + \text{mod}(ia-1, mb\_a)) +$$

$$\max(2, \max(nb\_a * \text{ceil}(NPROW-1, NPCOL),$$

$$LOC_c(n + \text{mod}(ja-1, nb\_a)) +$$

$$nb\_a * \text{ceil}(NPCOL-1, NPROW)))$$

でなければならない。

<i>iwork</i>	(ローカル) INTEGER。 ワークスペース配列、次元は ( <i>liwork</i> )。実数型でのみ使用される。
<i>liwork</i>	(ローカルまたはグローバル) INTEGER。 配列 <i>iwork</i> の次元。実数型でのみ使用される。 $liwork \geq LOC_j(n + \text{mod}(ia-1, mb\_a))$ でなければならない。
<i>rwork</i>	(ローカル) REAL (pcpocon の場合) DOUBLE PRECISION (pzpocon の場合) ワークスペース配列、次元は ( <i>lrwork</i> )。複素数型でのみ使用される。
<i>lrwork</i>	(ローカルまたはグローバル) INTEGER。 配列 <i>rwork</i> の次元。複素数型でのみ使用される。 $lrwork \geq LOC_c(n + \text{mod}(ja-1, nb\_a))$ でなければならない。

### 出力パラメータ

<i>rcond</i>	(グローバル) REAL (単精度の場合)。 DOUBLE PRECISION (倍精度の場合)。 分散行列 $\text{sub}(A)$ の条件数の逆数。
<i>work</i> (1)	終了時に、 <i>work</i> (1) には、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が格納される。
<i>iwork</i> (1)	終了時に、 <i>iwork</i> (1) には、最適なパフォーマンスを得るために必要な <i>liwork</i> の最小値が格納される (実数型の場合)。
<i>rwork</i> (1)	終了時に、 <i>rwork</i> (1) には、最適なパフォーマンスを得るために必要な <i>lrwork</i> の最小値が格納される (複素数型の場合)。
<i>info</i>	(グローバル) INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。  <i>info</i> < 0 の場合： <i>i</i> 番目の引数が配列で、 <i>j</i> 番目の値が不正だった場合、 $info = -(i*100+j)$ 。 <i>i</i> 番目の引数がスカラで値が不正だった場合、 $info = -i$ 。

### 解の精度の改善と誤差の推定

このセクションでは、算出された連立 1 次方程式の解の精度を改善し、解の誤差を推定するための ScaLAPACK ルーチンについて説明する。これらのルーチン呼び出す前に、連立方程式の行列を因子分解し、解を計算する必要がある ([「行列の因子分解用のルーチン」](#) および [「連立 1 次方程式を解くためのルーチン」](#) を参照)。

## p?gerfs

連立 1 次方程式の算出された解を改善し、  
解の誤差範囲と後退誤差推定を提供する。

### 構文

```
call psgerfs(trans, n, nrhs, a, ia, ja, desca, af, iaf, jaf, descaf,
             ipiv, b, ib, jb, descb, x, ix, jx, descx, ferr, berr, work, lwork,
             iwork, liwork, info)

call pdgerfs(trans, n, nrhs, a, ia, ja, desca, af, iaf, jaf, descaf,
             ipiv, b, ib, jb, descb, x, ix, jx, descx, ferr, berr, work, lwork,
             iwork, liwork, info)

call pcgerfs(trans, n, nrhs, a, ia, ja, desca, af, iaf, jaf, descaf,
             ipiv, b, ib, jb, descb, x, ix, jx, descx, ferr, berr, work, lwork,
             rwork, lrwork, info)

call pzgerfs(trans, n, nrhs, a, ia, ja, desca, af, iaf, jaf, descaf,
             ipiv, b, ib, jb, descb, x, ix, jx, descx, ferr, berr, work, lwork,
             rwork, lrwork, info)
```

### 説明

このルーチンは、以下のいずれかの連立 1 次方程式の算出された解を改善し、解の誤差範囲と後退誤差推定を提供する。

$$\begin{aligned} \text{sub}(A) * \text{sub}(X) &= \text{sub}(B), \\ \text{sub}(A)^T * \text{sub}(X) &= \text{sub}(B), \text{ または} \\ \text{sub}(A)^T * \text{sub}(X) &= \text{sub}(B) \end{aligned}$$

ここで、 $\text{sub}(A) = A(\text{ia}:\text{ia}+\text{n}-1, \text{ja}:\text{ja}+\text{n}-1)$ 、 $\text{sub}(B) = B(\text{ib}:\text{ib}+\text{n}-1, \text{jb}:\text{jb}+\text{nrhs}-1)$ 、  
および  $\text{sub}(X) = X(\text{ix}:\text{ix}+\text{n}-1, \text{jx}:\text{jx}+\text{nrhs}-1)$  である。

### 入力パラメータ

*trans* (グローバル) CHARACTER\*1。  
'N'、'T'、または'C'のいずれかでなければならない。  
連立方程式の形式を指定する。  
  
*trans* = 'N' の場合、連立方程式の形式は、  
 $\text{sub}(A) * \text{sub}(X) = \text{sub}(B)$  (転置なし) である。

	$trans = 'T'$ の場合、連立方程式の形式は、 $sub(A)^T * sub(X) = sub(B)$ (転置あり) である。 $trans = 'C'$ の場合、連立方程式の形式は、 $sub(A)^H * sub(X) = sub(B)$ (共役転置) である。
$n$	(グローバル) INTEGER。 分散部分行列 $sub(A)$ の次数 ( $n \geq 0$ )。
$nrhs$	(グローバル) INTEGER。右辺の数。 すなわち、 $sub(B)$ と $sub(X)$ の列数 ( $nrhs \geq 0$ )。
$a, af, b, x$	(ローカル) REAL (psgerfs の場合) DOUBLE PRECISION (pdgerfs の場合) COMPLEX (pcgerfs の場合) DOUBLE COMPLEX (pzgerfs の場合)。 それぞれ、ローカルメモリにある、ローカル次元の配列 $a(lld\_a, LOC_c(ja+n-1))$ 、 $af(lld\_af, LOC_c(jaf+n-1))$ 、 $b(lld\_b, LOC_c(jb+nrhs-1))$ 、および $x(lld\_x, LOC_c(jx+nrhs-1))$ へのポインタ。 配列 $a$ は、分散行列 $sub(A)$ のローカル部分を含む。 配列 $af$ は、 <a href="#">p?getrf</a> によって計算された行列 $sub(A) = PLU$ の分散要素のローカル部分を含む。 配列 $b$ は、右辺の分散行列 $sub(B)$ のローカル部分を含む。 呼び出し時に、配列 $x$ は解の分散行列 $sub(X)$ のローカル部分を含む。
$ia, ja$	(グローバル) INTEGER。 それぞれ、部分行列 $sub(A)$ の最初の行と最初の列を示す、グローバル配列 $A$ の行インデックスと列インデックス。
$desca$	(グローバルおよびローカル) INTEGER。配列、次元は ( $dlen\_$ )。 分散行列 $A$ の配列ディスクリプタ。
$iaf, jaf$	(グローバル) INTEGER。 それぞれ、部分行列 $sub(AF)$ の最初の行と最初の列を示す、グローバル配列 $AF$ の行インデックスと列インデックス。
$descaf$	(グローバルおよびローカル) INTEGER。配列、次元は ( $dlen\_$ )。 分散行列 $AF$ の配列ディスクリプタ。

<i>ib, jb</i>	(グローバル) INTEGER。 それぞれ、部分行列 $\text{sub}(B)$ の最初の行と最初の列を示す、グローバル配列 $B$ の行インデックスと列インデックス。
<i>descb</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。 分散行列 $B$ の配列ディスクリプタ。
<i>ix, jx</i>	(グローバル) INTEGER。 それぞれ、部分行列 $\text{sub}(X)$ の最初の行と最初の列を示す、グローバル配列 $X$ の行インデックスと列インデックス。
<i>descx</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。 分散行列 $X$ の配列ディスクリプタ。
<i>ipiv</i>	(ローカル) INTEGER。配列、次元は $LOC_r(m\_af) + mb\_af$ 。 この配列は、 <a href="#">p?getrf</a> によって計算されたピボット情報を含む。 $ipiv(i)=j$ の場合、ローカル行 $i$ は、グローバル行 $j$ と交換される。 この配列は、分散行列 $A$ に関連付けられる。
<i>work</i>	(ローカル) REAL (psgerfs の場合) DOUBLE PRECISION (pdgerfs の場合) COMPLEX (pcgerfs の場合) DOUBLE COMPLEX (pzgerfs の場合)。  次元 ( <i>lwork</i> ) の配列 <i>work</i> はワークスペース配列。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。配列 <i>work</i> の次元。 実数型の場合: $lwork \geq 3 * LOC_r(n + \text{mod}(ia-1, mb\_a))$ でなければならない。  複素数型の場合: $lwork \geq 2 * LOC_r(n + \text{mod}(ia-1, mb\_a))$ でなければならない。
<i>iwork</i>	(ローカル) INTEGER。 ワークスペース配列、次元は ( <i>liwork</i> )。実数型でのみ使用される。
<i>liwork</i>	(ローカルまたはグローバル) INTEGER。 配列 <i>iwork</i> の次元。実数型でのみ使用される。 $liwork \geq LOC_r(n + \text{mod}(ib-1, mb\_b))$ でなければならない。
<i>rwork</i>	(ローカル) REAL (pcgerfs の場合) DOUBLE PRECISION (pzgerfs の場合) ワークスペース配列、次元は ( <i>lrwork</i> )。複素数型でのみ使用される。



*lwork* (ローカルまたはグローバル) INTEGER。  
 配列 *rwork* の次元。複素数型でのみ使用される。  
 $lwork \geq LOC_r(n + \text{mod}(ib-1, mb\_b))$  でなければならない。

## 出力パラメータ

*x* 終了時に、改善された解ベクトルが格納される。

*ferr*, *berr* REAL (単精度の場合)。  
 DOUBLE PRECISION (倍精度の場合)。  
 配列、次元はそれぞれ  $LOC_c(jb + nrhs - 1)$ 。  
 配列 *ferr* には、 $\text{sub}(X)$  の各解ベクトル用に推定された前進誤差範囲が格納される。  
 XTRUE が  $\text{sub}(X)$  に対応する真の解の場合、*ferr* は、 $(\text{sub}(X) - \text{XTRUE})$  の最大成分の大きさを  $\text{sub}(X)$  の最大成分の大きさで割った推定上限である。この推定値は *rcond* に対する推定値と同程度の信頼性があり、ほとんどの場合、実際の誤差よりも少し多めに推定される。  
 この配列は、分散行列 *X* に関連付けられる。  
 配列 *berr* は、各解ベクトルの成分ごとの相対後退誤差を含む (すなわち、 $\text{sub}(X)$  が正確な解となる  $\text{sub}(A)$  または  $\text{sub}(B)$  の任意のエントリにおける最小相対変化)。この配列は、分散行列 *X* に関連付けられる。

*work*(1) 終了時に、*work*(1) には、最適なパフォーマンスを得るために必要な *lwork* の最小値が格納される。

*iwork*(1) 終了時に、*iwork*(1) には、最適なパフォーマンスを得るために必要な *liwork* の最小値が格納される (実数型の場合)。

*rwork*(1) 終了時に、*rwork*(1) には、最適なパフォーマンスを得るために必要な *lwork* の最小値が格納される (複素数型の場合)。

*info* (グローバル) INTEGER。 *info* = 0 の場合、実行は正常に終了したことを示す。  
*info* < 0 の場合：  
*i* 番目の引数が配列で、*j* 番目の値が不正だった場合、  
 $info = -(i * 100 + j)$ 。*i* 番目の引数がスカラーで値が不正だった場合、  
 $info = -i$ 。

## p?porfs

対称 / エルミート 正定値分散行列の連立 1 次方程式の算出された解を改善し、解の誤差範囲と後退誤差推定を提供する。

### 構文

```
call psporfs(uplo, n, nrhs, a, ia, ja, desca, af, iaf, jaf, descaf, b,
             ib, jb, descb, x, ix, jx, descx, ferr, berr, work, lwork, iwork,
             liwork, info)

call pdporfs(uplo, n, nrhs, a, ia, ja, desca, af, iaf, jaf, descaf, b,
             ib, jb, descb, x, ix, jx, descx, ferr, berr, work, lwork, iwork,
             liwork, info)

call pcporfs(uplo, n, nrhs, a, ia, ja, desca, af, iaf, jaf, descaf, b,
             ib, jb, descb, x, ix, jx, descx, ferr, berr, work, lwork, rwork,
             lrwork, info)

call pzporfs(uplo, n, nrhs, a, ia, ja, desca, af, iaf, jaf, descaf, b,
             ib, jb, descb, x, ix, jx, descx, ferr, berr, work, lwork, rwork,
             lrwork, info)
```

### 説明

ルーチン p?porfs は、連立 1 次方程式  

$$\text{sub}(A) * \text{sub}(X) = \text{sub}(B)$$
の算出された解を改善する。

ここで、 $\text{sub}(A) = A(\text{ia}:\text{ia}+\text{n}-1, \text{ja}:\text{ja}+\text{n}-1)$  は実対称 / 複素エルミート 正定値分散行列、  
 $\text{sub}(B) = B(\text{ib}:\text{ib}+\text{n}-1, \text{jb}:\text{jb}+\text{nrhs}-1)$ 、  
 $\text{sub}(X) = X(\text{ix}:\text{ix}+\text{n}-1, \text{jx}:\text{jx}+\text{nrhs}-1)$   
はそれぞれ、右辺と解の部分行列である。  
また、このルーチンは、解の誤差範囲と後退誤差推定を提供する。

### 入力パラメータ

**uplo** (グローバル) CHARACTER\*1。  
'U' または 'L' でなければならない。  
対称 / エルミート行列 **sub(A)** の上三角部分と下三角部分のどちらが格納されるかを指定する。

	$uplo = 'U'$ の場合、 $sub(A)$ は上三角部分である。 $uplo = 'L'$ の場合、 $sub(A)$ は下三角部分である。
$n$	(グローバル) INTEGER。 分散行列 $sub(A)$ の次数 ( $n \geq 0$ )。
$nrhs$	(グローバル) INTEGER。右辺の数。 すなわち、 $sub(B)$ と $sub(X)$ の列数 ( $nrhs \geq 0$ )。
$a, af, b, x$	(ローカル) REAL (psporfs の場合) DOUBLE PRECISION (pdporfs の場合) COMPLEX (pcporfs の場合) DOUBLE COMPLEX (pzporfs の場合)。  それぞれ、ローカルメモリにある、ローカル次元の配列 $a(lld\_a, LOC_c(ja+n-1))$ 、 $af(lld\_af, LOC_c(ja+n-1))$ 、 $b(lld\_b, LOC_c(jb+nrhs-1))$ 、および $x(lld\_x, LOC_c(jx+nrhs-1))$ へのポインタ。  配列 $a$ は、 $n \times n$ の対称 / エルミート分散行列 $sub(A)$ のローカル部分を含む。 $uplo = 'U'$ の場合、 $sub(A)$ の先頭の $n \times n$ 上三角部分は行列の上三角部分を含む。厳密な下三角部分は参照されない。 $uplo = 'L'$ の場合、 $sub(A)$ の先頭の $n \times n$ 下三角部分は行列の下三角部分を含む。厳密な上三角部分は参照されない。  配列 $af$ は、p?potrf によって行われたコレスキー因子分解 $sub(A) = LL^H$ または $sub(A) = U^H U$ で得られた係数 $L$ または $U$ を含む。  呼び出し時に、配列 $b$ は右辺の分散行列 $sub(B)$ のローカル部分を含む。  呼び出し時に、配列 $x$ は解のベクトル $sub(X)$ のローカル部分を含む。
$ia, ja$	(グローバル) INTEGER。 それぞれ、部分行列 $sub(A)$ の最初の行と最初の列を示す、グローバル配列 $A$ の行インデックスと列インデックス。
$desca$	(グローバルおよびローカル) INTEGER。配列、次元は ( $dlen\_$ )。 分散行列 $A$ の配列ディスクリプタ。
$iaf, jaf$	(グローバル) INTEGER。 それぞれ、部分行列 $sub(AF)$ の最初の行と最初の列を示す、グローバル配列 $AF$ の行インデックスと列インデックス。

<i>descaf</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。分散行列 <i>AF</i> の配列ディスクリプタ。
<i>ib, jb</i>	(グローバル) INTEGER。 それぞれ、部分行列 <i>sub(B)</i> の最初の行と最初の列を示す、グローバル配列 <i>B</i> の行インデックスと列インデックス。
<i>descb</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。分散行列 <i>B</i> の配列ディスクリプタ。
<i>ix, jx</i>	(グローバル) INTEGER。 それぞれ、部分行列 <i>sub(X)</i> の最初の行と最初の列を示す、グローバル配列 <i>X</i> の行インデックスと列インデックス。
<i>descx</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。分散行列 <i>X</i> の配列ディスクリプタ。
<i>work</i>	(ローカル) REAL ( <i>psporfs</i> の場合) DOUBLE PRECISION ( <i>pdporfs</i> の場合) COMPLEX ( <i>pcporfs</i> の場合) DOUBLE COMPLEX ( <i>pzporfs</i> の場合)。 次元 ( <i>lwork</i> ) の配列 <i>work</i> はワークスペース配列。
<i>lwork</i>	(ローカル) INTEGER。配列 <i>work</i> の次元。 実数型の場合: $lwork \geq 3 * LOC_r(n + \text{mod}(ia - 1, mb\_a))$ でなければならない。 複素数型の場合: $lwork \geq 2 * LOC_r(n + \text{mod}(ia - 1, mb\_a))$ でなければならない。
<i>iwork</i>	(ローカル) INTEGER。 ワークスペース配列、次元は ( <i>liwork</i> )。実数型でのみ使用される。
<i>liwork</i>	(ローカルまたはグローバル) INTEGER。 配列 <i>iwork</i> の次元。実数型でのみ使用される。 $liwork \geq LOC_r(n + \text{mod}(ib - 1, mb\_b))$ でなければならない。
<i>rwork</i>	(ローカル) REAL ( <i>pcporfs</i> の場合) DOUBLE PRECISION ( <i>pzporfs</i> の場合) ワークスペース配列、次元は ( <i>lrwork</i> )。複素数型でのみ使用される。

*lrwork* (ローカルまたはグローバル) INTEGER。  
 配列 *rwork* の次元。複素数型でのみ使用される。  
 $lrwork \geq LOC_r(n + \text{mod}(ib-1, mb\_b))$  でなければならない。

## 出力パラメータ

*x* 終了時に、改善された解ベクトルが格納される。

*ferr*, *berr* REAL (単精度の場合)。  
 DOUBLE PRECISION (倍精度の場合)。  
 配列、次元はそれぞれ  $LOC_c(jb + nrhs - 1)$ 。  
 配列 *ferr* には、*sub(X)* の各解ベクトル用に推定された前進誤差範囲が格納される。  
 XTRUE が *sub(X)* に対応する真の解の場合、*ferr* は、(*sub(X)* - XTRUE) の最大成分の大きさを *sub(X)* の最大成分の大きさで割った推定上限である。この推定値は *rcond* に対する推定値と同程度の信頼性があり、ほとんどの場合、実際の誤差よりも少し多めに推定される。  
 この配列は、分散行列 *X* に関連付けられる。  
 配列 *berr* は、各解ベクトルの成分ごとの相対後退誤差を含む (すなわち、*sub(X)* が正確な解となる *sub(A)* または *sub(B)* の任意のエントリにおける最小相対変化)。この配列は、分散行列 *X* に関連付けられる。

*work*(1) 終了時に、*work*(1) には、最適なパフォーマンスを得るために必要な *lwork* の最小値が格納される。

*iwork*(1) 終了時に、*iwork*(1) には、最適なパフォーマンスを得るために必要な *liwork* の最小値が格納される (実数型の場合)。

*rwork*(1) 終了時に、*rwork*(1) には、最適なパフォーマンスを得るために必要な *lrwork* の最小値が格納される (複素数型の場合)。

*info* (グローバル) INTEGER。 *info* = 0 の場合、実行は正常に終了したことを示す。  
*info* < 0 の場合：  
*i* 番目の引数が配列で、*j* 番目の値が不正だった場合、  
 $info = -(i * 100 + j)$ 。*i* 番目の引数がスカラーで値が不正だった場合、  
 $info = -i$ 。

## p?trrfs

分散三角係数行列の連立1次方程式の解の誤差範囲と後退誤差推定を提供する。

### 構文

```
call pstrrfs(uplo, trans, diag, n, nrhs, a, ia, ja, desca, b, ib, jb,
             descb, x, ix, jx, descx, ferr, berr, work, lwork, iwork, liwork, info)
call pdtrrfs(uplo, trans, diag, n, nrhs, a, ia, ja, desca, b, ib, jb,
             descb, x, ix, jx, descx, ferr, berr, work, lwork, iwork, liwork, info)
call pctrfs(uplo, trans, diag, n, nrhs, a, ia, ja, desca, b, ib, jb,
            descb, x, ix, jx, descx, ferr, berr, work, lwork, rwork, lrwork, info)
call pztrrfs(uplo, trans, diag, n, nrhs, a, ia, ja, desca, b, ib, jb,
            descb, x, ix, jx, descx, ferr, berr, work, lwork, rwork, lrwork, info)
```

### 説明

ルーチン p?trrfs は、以下のいずれかの連立1次方程式の解の誤差範囲と後退誤差推定を提供する。

$$\begin{aligned} \text{sub}(A) * \text{sub}(X) &= \text{sub}(B), \\ \text{sub}(A)^T * \text{sub}(X) &= \text{sub}(B), \text{または} \\ \text{sub}(A)^T * \text{sub}(X) &= \text{sub}(B) \end{aligned}$$

ここで、 $\text{sub}(A) = A(\text{ia}:\text{ia}+n-1, \text{ja}:\text{ja}+n-1)$  は三角行列、  
 $\text{sub}(B) = B(\text{ib}:\text{ib}+n-1, \text{jb}:\text{jb}+\text{nrhs}-1)$ 、および  
 $\text{sub}(X) = X(\text{ix}:\text{ix}+n-1, \text{jx}:\text{jx}+\text{nrhs}-1)$  である。

解の行列  $X$  は、このルーチンに入る前に、p?trtrs または他の手段によって計算されていなければならない。後退誤差が改善されないことがあるため、ルーチン p?trrfs は精度の改善を繰り返して行わない。

### 入力パラメータ

**uplo** (グローバル) CHARACTER\*1。  
 'U' または 'L' でなければならない。  
 uplo = 'U' の場合、sub(A) は上三角である。  
 uplo = 'L' の場合、sub(A) は下三角である。

<i>trans</i>	<p>(グローバル) CHARACTER*1。 'N'、'T'、または 'C' のいずれかでなければならない。 連立方程式の形式を指定する。</p> <p><i>trans</i> = 'N' の場合、連立方程式の形式は、 <math>\text{sub}(A) * \text{sub}(X) = \text{sub}(B)</math> (転置なし) である。</p> <p><i>trans</i> = 'T' の場合、連立方程式の形式は、 <math>\text{sub}(A)^T * \text{sub}(X) = \text{sub}(B)</math> (転置あり) である。</p> <p><i>trans</i> = 'C' の場合、連立方程式の形式は、 <math>\text{sub}(A)^H * \text{sub}(X) = \text{sub}(B)</math> (共役転置) である。</p>
<i>diag</i>	<p>CHARACTER*1。 'N' または 'U' でなければならない。</p> <p><i>diag</i> = 'N' の場合、<i>sub</i>(<i>A</i>) は単位三角ではない。</p> <p><i>diag</i> = 'U' の場合、<i>sub</i>(<i>A</i>) は単位三角である。</p>
<i>n</i>	<p>(グローバル) INTEGER。 分散行列 <i>sub</i>(<i>A</i>) の次数 (<math>n \geq 0</math>)。</p>
<i>nrhs</i>	<p>(グローバル) INTEGER。右辺の数。 すなわち、<i>sub</i>(<i>B</i>) と <i>sub</i>(<i>X</i>) の列数 (<math>nrhs \geq 0</math>)。</p>
<i>a</i> , <i>b</i> , <i>x</i>	<p>(ローカル) REAL (<i>pstrrfs</i> の場合) DOUBLE PRECISION (<i>pdtrrfs</i> の場合) COMPLEX (<i>pctrfs</i> の場合) DOUBLE COMPLEX (<i>pztrfs</i> の場合)。</p> <p>それぞれ、ローカルメモリにある、ローカル次元の配列 <i>a</i>(<i>lld_a</i>, <i>LOC<sub>c</sub></i>(<i>ja</i>+<i>n</i>-1))、<i>b</i>(<i>lld_b</i>, <i>LOC<sub>c</sub></i>(<i>jb</i>+<i>nrhs</i>-1))、および <i>x</i>(<i>lld_x</i>, <i>LOC<sub>c</sub></i>(<i>jx</i>+<i>nrhs</i>-1)) へのポインタ。</p> <p>配列 <i>a</i> は、元の三角分散行列 <i>sub</i>(<i>A</i>) のローカル部分を含む。 <i>uplo</i> = 'U' の場合、<i>sub</i>(<i>A</i>) の先頭の <math>n \times n</math> 上三角部分は行列の上三角部分を含む。厳密な下三角部分は参照されない。 <i>uplo</i> = 'L' の場合、<i>sub</i>(<i>A</i>) の先頭の <math>n \times n</math> 下三角部分は行列の下三角部分を含む。厳密な上三角部分は参照されない。 <i>diag</i> = 'U' の場合、<i>sub</i>(<i>A</i>) の対角成分も参照されず、1 と仮定される。</p> <p>呼び出し時に、配列 <i>b</i> は右辺の分散行列 <i>sub</i>(<i>B</i>) のローカル部分を含む。</p> <p>呼び出し時に、配列 <i>x</i> は解のベクトル <i>sub</i>(<i>X</i>) のローカル部分を含む。</p>

<i>ia, ja</i>	(グローバル) INTEGER。 それぞれ、部分行列 <i>sub(A)</i> の最初の行と最初の列を示す、グローバル配列 <i>A</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。 分散行列 <i>A</i> の配列ディスクリプタ。
<i>ib, jb</i>	(グローバル) INTEGER。 それぞれ、部分行列 <i>sub(B)</i> の最初の行と最初の列を示す、グローバル配列 <i>B</i> の行インデックスと列インデックス。
<i>descb</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。 分散行列 <i>B</i> の配列ディスクリプタ。
<i>ix, jx</i>	(グローバル) INTEGER。 それぞれ、部分行列 <i>sub(X)</i> の最初の行と最初の列を示す、グローバル配列 <i>X</i> の行インデックスと列インデックス。
<i>descx</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。 分散行列 <i>X</i> の配列ディスクリプタ。
<i>work</i>	(ローカル) REAL ( <i>pstrrfs</i> の場合) DOUBLE PRECISION ( <i>pdtrrfs</i> の場合) COMPLEX ( <i>pctrfs</i> の場合) DOUBLE COMPLEX ( <i>pztrfs</i> の場合)。  次元 ( <i>lwork</i> ) の配列 <i>work</i> はワークスペース配列。
<i>lwork</i>	(ローカル) INTEGER。配列 <i>work</i> の次元。 実数型の場合: $lwork \geq 3 * LOC_r(n + \text{mod}(ia-1, mb_a))$ でなければならない。  複素数型の場合: $lwork \geq 2 * LOC_r(n + \text{mod}(ia-1, mb_a))$ でなければならない。
<i>iwork</i>	(ローカル) INTEGER。 ワークスペース配列、次元は ( <i>liwork</i> )。実数型でのみ使用される。
<i>liwork</i>	(ローカルまたはグローバル) INTEGER。 配列 <i>iwork</i> の次元。実数型でのみ使用される。 $liwork \geq LOC_r(n + \text{mod}(ib-1, mb_b))$ でなければならない。



*rwork* (ローカル) REAL (pctrrfs の場合)  
DOUBLE PRECISION (pztrrfss の場合)  
ワークスペース配列、次元は (*lrwork*)。複素数型でのみ使用される。

*lrwork* (ローカルまたはグローバル) INTEGER。  
配列 *rwork* の次元。複素数型でのみ使用される。  
 $lrwork \geq LOC_r(n + \text{mod}(ib-1, mb\_b))$  でなければならない。

## 出力パラメータ

*ferr*, *berr* REAL (単精度の場合)。  
DOUBLE PRECISION (倍精度の場合)。  
配列、次元はそれぞれ  $LOC_c(jb + nrhs - 1)$ 。

配列 *ferr* には、*sub(X)* の各解ベクトル用に推定された前進誤差範囲が格納される。  
XTRUE が *sub(X)* に対応する真の解の場合、*ferr* は、(*sub(X)* - XTRUE) の最大成分の大きさを *sub(X)* の最大成分の大きさで割った推定上限である。この推定値は *rcond* に対する推定値と同程度の信頼性があり、ほとんどの場合、実際の誤差よりも少し多めに推定される。  
この配列は、分散行列 *X* に関連付けられる。

配列 *berr* は、各解ベクトルの成分ごとの相対後退誤差を含む (すなわち、*sub(X)* が正確な解となる *sub(A)* または *sub(B)* の任意のエントリにおける最小相対変化)。この配列は、分散行列 *X* に関連付けられる。

*work(1)* 終了時に、*work(1)* には、最適なパフォーマンスを得るために必要な *lwork* の最小値が格納される。

*iwork(1)* 終了時に、*iwork(1)* には、最適なパフォーマンスを得るために必要な *liwork* の最小値が格納される (実数型の場合)。

*rwork(1)* 終了時に、*rwork(1)* には、最適なパフォーマンスを得るために必要な *lrwork* の最小値が格納される (複素数型の場合)。

*info* (グローバル) INTEGER。 *info* = 0 の場合、実行は正常に終了したことを示す。

*info* < 0 の場合：

*i* 番目の引数が配列で、*j* 番目の値が不正だった場合、  
 $info = -(i * 100 + j)$ 。*i* 番目の引数がスカラで値が不正だった場合、  
 $info = -i$ 。

### 行列の反転用のルーチン

このセクションでは、以前に得られた因子分解に基づいて行列の逆行列を計算する ScaLAPACK ルーチンについて説明する。連立方程式  $Ax = b$  を解く場合、最初に  $A^{-1}$  を計算し、次に行列・ベクトルの積  $x = A^{-1}b$  を計算しないように注意する。代わりに、ソルバルーチン呼び出す(「[連立1次方程式を解くためのルーチン](#)」を参照)。この方が効率が良く、高い精度の結果が得られる。

---

### p?getri

*LU* 因子分解された分散行列の逆行列を計算する。

---

#### 構文

```
call psgetri(n, a, ia, ja, desca, ipiv, work, lwork, iwork, liwork, info)
call pdgetri(n, a, ia, ja, desca, ipiv, work, lwork, iwork, liwork, info)
call pcgetri(n, a, ia, ja, desca, ipiv, work, lwork, iwork, liwork, info)
call pzgetri(n, a, ia, ja, desca, ipiv, work, lwork, iwork, liwork, info)
```

#### 説明

このルーチンは、p?getrf によって行われた *LU* 因子分解を使用して、一般分散行列  $\text{sub}(A) = A(ia:ia+n-1, ja:ja+n-1)$  の逆行列を計算する。この手法は、 $U$  を反転した後で方程式を解いて、 $\text{InvA}$  で示された  $\text{sub}(A)$  の逆行列を計算する。

$$\text{InvA} * L = U^{-1}$$

( $\text{InvA}$  の場合)。

#### 入力パラメータ

<i>n</i>	(グローバル) INTEGER。処理される行数と列数。 すなわち、分散部分行列 $\text{sub}(A)$ の次数 ( $n \geq 0$ )。
<i>a</i>	(ローカル) REAL (psgetri の場合) DOUBLE PRECISION (pdgetri の場合) COMPLEX (pcgetri の場合) DOUBLE COMPLEX (pzgetri の場合)。

ローカルメモリにある、ローカル次元  $a(lld\_a, LOC_c(ja+n-1))$  の配列へのポインタ。

呼び出し時に、配列  $a$  は、`p?getrf` によって行われた因子分解  $\text{sub}(A) = PLU$  で得られた係数  $L$  と  $U$  のローカル部分を含む。

<i>ia, ja</i>	(グローバル) INTEGER。 それぞれ、部分行列 $\text{sub}(A)$ の最初の行と最初の列を示す、グローバル配列 $A$ の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。分散行列 $A$ の配列ディスクリプタ。
<i>work</i>	(ローカル) REAL ( <code>psgetri</code> の場合) DOUBLE PRECISION ( <code>pdgetri</code> の場合) COMPLEX ( <code>pcgetri</code> の場合) DOUBLE COMPLEX ( <code>pzgetri</code> の場合)。  次元 ( <i>lwork</i> ) の配列 <i>work</i> はワークスペース配列。
<i>lwork</i>	(ローカル) INTEGER。配列 <i>work</i> の次元。 $lwork \geq LOC_r(n + \text{mod}(ia-1, mb\_a)) * nb\_a$ でなければならない。 配列 <i>work</i> は、 $\text{sub}(A)$ の列ブロック全体を維持するために使用される。
<i>iwork</i>	(ローカル) INTEGER。 ピボットの物理的な転置に使用されるワークスペース配列、次元は ( <i>liwork</i> )。
<i>liwork</i>	(ローカルまたはグローバル) INTEGER。配列 <i>iwork</i> の次元。 最小値 <i>liwork</i> は、以下のコードによって決定される。 <pre> If NPROW == NPCOL then   liwork = LOCc(n_a + mod(ja-1, nb_a)) + nb_a Else   liwork = LOCc(n_a + mod(ja-1, nb_a)) + max(ceil(ceil(LOCr(m_a)/nb_a)/(lcm/NPROW)), nb_a) End if </pre> ここで、 <i>lcm</i> はプロセス列とプロセス行 ( <i>NPROW</i> および <i>NPCOL</i> ) の最小公倍数である。

### 出力パラメータ

<code>ipiv</code>	(ローカル) INTEGER。 配列、次元は $(LOC_r(m\_a) + mb\_a)$ 。 この配列には、ピボット情報が格納される。 <code>ipiv(i)=j</code> の場合、ローカル行 <i>i</i> は、グローバル行 <i>j</i> と交換される。 この配列は、分散行列 <i>A</i> に関連付けられる。
<code>work(1)</code>	終了時に、 <code>work(1)</code> には、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が格納される。
<code>iwork(1)</code>	終了時に、 <code>iwork(1)</code> には、最適なパフォーマンスを得るために必要な <i>liwork</i> の最小値が格納される。
<code>info</code>	(グローバル) INTEGER。 <code>info=0</code> の場合、実行は正常に終了したことを示す。  <code>info &lt; 0</code> の場合： <i>i</i> 番目の引数が配列で、 <i>j</i> 番目の値が不正だった場合、 <code>info = -(i*100+j)</code> 。 <i>i</i> 番目の引数がスカラで値が不正だった場合、 <code>info = -i</code> 。  <code>info &gt; 0</code> の場合： <code>info = i</code> の場合、 <i>U</i> ( <i>i</i> , <i>i</i> ) は完全にゼロである。因子分解は完了したが、 係数 <i>U</i> は完全に特異で、連立方程式の解の算出に使用するとゼロ除算が発生する。

---

## p?potri

対称/エルミート正定値分散行列の逆行列を計算する。

---

### 構文

```
call pspotri(uplo, n, a, ia, ja, desca, info)
call pdpotri(uplo, n, a, ia, ja, desca, info)
call pcpotri(uplo, n, a, ia, ja, desca, info)
call pzpotri(uplo, n, a, ia, ja, desca, info)
```

## 説明

このルーチンは、p?potrf によって行われたコレスキー因子分解  $\text{sub}(A) = U^H U$  または  $\text{sub}(A) = LL^H$  を使用して、実対称 / 複素エルミート正定値分散行列  $\text{sub}(A) = A(ia:ia+n-1, ja:ja+n-1)$  の逆行列を計算する。

## 入力パラメータ

- uplo** (グローバル) CHARACTER\*1。  
'U' または 'L' でなければならない。  
  
対称 / エルミート行列  $\text{sub}(A)$  の上三角部分と下三角部分のどちらが格納されるかを指定する。  
  
 $\text{uplo} = 'U'$  の場合、 $\text{sub}(A)$  の上三角部分が格納される。  
 $\text{uplo} = 'L'$  の場合、 $\text{sub}(A)$  の下三角部分が格納される。
- n** (グローバル) INTEGER。処理される行数と列数。  
すなわち、分散部分行列  $\text{sub}(A)$  の次数 ( $n \geq 0$ )。
- a** (ローカル)  
REAL (pspotri の場合)  
DOUBLE PRECISION (pdpotri の場合)  
COMPLEX (pcpotri の場合)  
DOUBLE COMPLEX (pzpotri の場合)。  
  
ローカルメモリにある、ローカル次元  $a(1:d_a, LOC_c(ja+n-1))$  の配列へのポインタ。  
  
呼び出し時に、配列  $a$  は、p?potrf によって行われたコレスキー因子分解  $\text{sub}(A) = U^H U$  または  $\text{sub}(A) = LL^H$  で得られた係数  $U$  または  $L$  を含む。
- ia, ja** (グローバル) INTEGER。  
それぞれ、部分行列  $\text{sub}(A)$  の最初の行と最初の列を示す、グローバル配列  $A$  の行インデックスと列インデックス。
- desca** (グローバルおよびローカル) INTEGER。配列、次元は (dlen\_)。  
分散行列  $A$  の配列ディスクリプタ。

## 出力パラメータ

- a** 終了時に、 $\text{sub}(A)$  の (対称 / エルミート) 逆行列の上三角または下三角のローカル部分によって上書きされる。
- info** (グローバル) INTEGER。info = 0 の場合、実行は正常に終了したことを示す。

*info* < 0 の場合 :

*i* 番目の引数が配列で、*j* 番目の値が不正だった場合、  
*info* = -(*i*\*100+*j*)。 *i* 番目の引数がスカラで値が不正だった場合、  
*info* = -*i*。

*info* > 0 の場合 :

*info* = *i* の場合、係数 *U* または *L* の (*i*,*i*) 成分がゼロで、逆行列は計算できない。

---

### p?trtri

三角分散行列の逆行列を計算する。

---

#### 構文

```
call pstrtri(uplo, diag, n, a, ia, ja, desca, info)
call pdtrtri(uplo, diag, n, a, ia, ja, desca, info)
call pctrtri(uplo, diag, n, a, ia, ja, desca, info)
call pztrtri(uplo, diag, n, a, ia, ja, desca, info)
```

#### 説明

このルーチンは、実 / 複素上 / 下三角分散行列  $\text{sub}(A) = A(ia:ia+n-1, ja:ja+n-1)$  の逆行列を計算する。

#### 入力パラメータ

*uplo* (グローバル) CHARACTER\*1。  
'U' または 'L' でなければならない。  
分散行列  $\text{sub}(A)$  が上三角と下三角のどちらであるかを指定する。  
*uplo* = 'U' の場合、 $\text{sub}(A)$  は上三角である。  
*uplo* = 'L' の場合、 $\text{sub}(A)$  は下三角である。

*diag* CHARACTER\*1。  
'N' または 'U' でなければならない。  
分散行列  $\text{sub}(A)$  が単位三角かどうかを指定する。  
*diag* = 'N' の場合、 $\text{sub}(A)$  は単位三角ではない。  
*diag* = 'U' の場合、 $\text{sub}(A)$  は単位三角である。

<i>n</i>	(グローバル) INTEGER。処理される行数と列数。 すなわち、分散部分行列 $\text{sub}(A)$ の次数 ( $n \geq 0$ )。
<i>a</i>	(ローカル) REAL (pstrtri の場合) DOUBLE PRECISION (pdtrtri の場合) COMPLEX (pctrtri の場合) DOUBLE COMPLEX (pztrtri の場合)。  ローカルメモリにある、ローカル次元 $a(\text{lld\_a}, \text{LOC}_c(\text{ja}+n-1))$ の 配列へのポインタ。  配列 <i>a</i> は、三角分散行列 $\text{sub}(A)$ のローカル部分を含む。 <i>uplo</i> = 'U' の場合、 $\text{sub}(A)$ の先頭の $n \times n$ 上三角部分は反転する上三角 行列を含む。 $\text{sub}(A)$ の厳密な下三角部分は参照されない。 <i>uplo</i> = 'L' の場合、 $\text{sub}(A)$ の先頭の $n \times n$ 下三角部分は下三角行列を含 む。 $\text{sub}(A)$ の厳密な上三角部分は参照されない。
<i>ia, ja</i>	(グローバル) INTEGER。 それぞれ、部分行列 $\text{sub}(A)$ の最初の行と最初の列を示す、グローバル 配列 <i>A</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。 分散行列 <i>A</i> の配列ディスクリプタ。

### 出力パラメータ

<i>a</i>	終了時に、元の行列の (三角) 逆行列によって上書きされる。
<i>info</i>	(グローバル) INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。  <i>info</i> < 0 の場合： <i>i</i> 番目の引数が配列で、 <i>j</i> 番目の値が不正だった場合、 <i>info</i> = -( <i>i</i> *100+ <i>j</i> )。 <i>i</i> 番目の引数がスカラで値が不正だった場合、 <i>info</i> = - <i>i</i> 。  <i>info</i> > 0 の場合： <i>info</i> = <i>k</i> の場合、 $A(\text{ia}+k-1, \text{ja}+k-1)$ は完全にゼロである。三角関数 $\text{sub}(A)$ は特異で、逆行列は計算できない。

### 行列の平衡化

このセクションでは、行列の平衡化に必要なスケール係数の計算に使用される ScaLAPACK ルーチンについて説明する。ただし、これらのルーチンが行列を実際にスケールリングするわけではない。

---

### p?geequ

一般矩形分散行列を平衡化して条件数を小さくするための、行と列のスケール係数を計算する。

---

#### 構文

```
call psgeequ(m, n, a, ia, ja, desca, r, c, rowcnd, colcnd, amax, info)
call pdgeequ(m, n, a, ia, ja, desca, r, c, rowcnd, colcnd, amax, info)
call pcgeequ(m, n, a, ia, ja, desca, r, c, rowcnd, colcnd, amax, info)
call pzgeequ(m, n, a, ia, ja, desca, r, c, rowcnd, colcnd, amax, info)
```

#### 説明

このルーチンは、 $m \times n$  の分散行列  $\text{sub}(A) = A(ia:ia+m-1, ja:ja+n-1)$  を平衡化して条件数を小さくするための、行と列のスケール係数を計算する。出力配列  $r$  に行のスケール係数を返し、配列  $c$  に列のスケール係数を返す。これらの係数は、成分  $b_{ij}=r(i)*a_{ij}*c(j)$  の行列  $B$  の各行と各列の最大成分の絶対値が 1 になるように選択される。

$r(i)$  と  $c(j)$  は、 $SMLNUM$  = 最小の安全な数値と  $BIGNUM$  = 最大の安全な数値の範囲内に制限される。これらのスケール係数を使用しても  $\text{sub}(A)$  の条件数が小さくなることは保証されていないが、実際には有効に機能する。

補助関数 [p?lagge](#) は、p?geequ によって計算されたスケール係数を使用して、一般矩形行列をスケールリングする。

#### 入力パラメータ

- |     |   |
|-----|---|
| $m$ | (グローバル) INTEGER。処理される行数。<br>すなわち、分散部分行列 $\text{sub}(A)$ の行数 ( $m \geq 0$ )。 |
| $n$ | (グローバル) INTEGER。処理される列数。<br>すなわち、分散部分行列 $\text{sub}(A)$ の列数 ( $n \geq 0$ )。 |



<i>a</i>	(ローカル) REAL (psgeequ の場合) DOUBLE PRECISION (pdgeequ の場合) COMPLEX (pcgeequ の場合) DOUBLE COMPLEX (pzgeequ の場合)。  ローカルメモリにある、ローカル次元 $a(lld\_a, LOC_c(ja+n-1))$ の配列へのポインタ。  配列 <i>a</i> は、平衡化係数を計算する $m \times n$ の分散行列のローカル部分を含む。
<i>ia, ja</i>	(グローバル) INTEGER。 それぞれ、部分行列 $\text{sub}(A)$ の最初の行と最初の列を示す、グローバル配列 <i>A</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。分散行列 <i>A</i> の配列ディスクリプタ。

### 出力パラメータ

<i>r, c</i>	(ローカル) REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 配列、次元はそれぞれ $LOC_r(m\_a)$ および $LOC_c(n\_a)$ 。 <i>info</i> = 0 または <i>info</i> > <i>ia</i> + <i>m</i> -1 の場合、配列 <i>r</i> ( <i>ia</i> : <i>ia</i> + <i>m</i> -1) には、 $\text{sub}(A)$ の行スケール係数が格納される。 <i>r</i> は分散行列 <i>A</i> とアライメントされ、すべてのプロセス列に複製される。 <i>r</i> は、分散行列 <i>A</i> に関連付けられる。 <i>info</i> = 0 の場合、配列 <i>c</i> ( <i>ja</i> : <i>ja</i> + <i>n</i> -1) には、 $\text{sub}(A)$ の列スケール係数が格納される。 <i>c</i> は分散行列 <i>A</i> とアライメントされ、すべてのプロセス行に複製される。 <i>c</i> は、分散行列 <i>A</i> に関連付けられる。
<i>rowcnd, colcnd</i>	(グローバル) REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 <i>info</i> = 0 または <i>info</i> > <i>ia</i> + <i>m</i> -1 の場合、 <i>rowcnd</i> には最小の <i>r</i> ( <i>i</i> ) を最大の <i>r</i> ( <i>i</i> ) で割った値が格納される ( $ia \leq i \leq ia+m-1$ )。 <i>rowcnd</i> $\geq 0.1$ かつ <i>amax</i> が大きすぎる値でも小さすぎる値でもない場合、 <i>r</i> ( <i>ia</i> : <i>ia</i> + <i>m</i> -1) によるスケーリングは不要である。  <i>info</i> = 0 の場合、 <i>colcnd</i> には最小の <i>c</i> ( <i>j</i> ) を最大の <i>c</i> ( <i>j</i> ) で割った値が格納される ( $ja \leq j \leq ja+n-1$ )。 <i>colcnd</i> $\geq 0.1$ の場合、 <i>c</i> ( <i>ja</i> : <i>ja</i> + <i>n</i> -1) によるスケーリングは不要である。

<i>amax</i>	(グローバル) REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 最大行列成分の絶対値。 <i>amax</i> がオーバーフローまたはアンダーフローに非常に近い場合、行列をスケーリングする必要がある。
<i>info</i>	(グローバル) INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。  <i>info</i> < 0 の場合： <i>i</i> 番目の引数が配列で、 <i>j</i> 番目の値が不正だった場合、 <i>info</i> = -( <i>i</i> *100+ <i>j</i> )。 <i>i</i> 番目の引数がスカラで値が不正だった場合、 <i>info</i> = - <i>i</i> 。  <i>info</i> > 0 の場合： <i>info</i> = <i>i</i> で <i>i</i> ≤ <i>m</i> の場合、分散行列 sub( <i>A</i> ) の <i>i</i> 番目の行が完全にゼロである。 <i>i</i> > <i>m</i> の場合、分散行列 sub( <i>A</i> ) の ( <i>i</i> - <i>m</i> ) 番目の列が完全にゼロである。

---

### p?poequ

対称(エルミート) 正定値分散行列を平衡化して  
条件数を小さくするための行と列のスケール係数  
を計算する。

---

#### 構文

```
call pspoequ(n, a, ia, ja, desca, sr, sc, scond, amax, info)
call pdpoequ(n, a, ia, ja, desca, sr, sc, scond, amax, info)
call pcpcpoequ(n, a, ia, ja, desca, sr, sc, scond, amax, info)
call pzpcpoequ(n, a, ia, ja, desca, sr, sc, scond, amax, info)
```

#### 説明

このルーチンは、実対称 / 複素エルミート正定値分散行列  
sub(*A*) = *A*(*ia*:*ia*+*n*-1, *ja*:*ja*+*n*-1) を平衡化して (2- ノルムに関して) 条件数を小さくする  
ための行と列のスケール係数を計算する。出力配列 *sr* に行スケール係数を返し、*sc*  
に列スケール係数を返す。

$$s(i) = \frac{1}{\sqrt{a_{i,i}}}$$

これらの係数は、スケーリング後に成分  $b_{ij}=s(i)*a_{ij}*s(j)$  の行列  $B$  の対角成分が同じになるように選択される。

$sr$  および  $sc$  をこのように選択すると、 $B$  の条件数は、すべての可能な対角スケーリングで、可能な最小の条件数に係数  $n$  を掛けた値の範囲内に収まる。

補助関数 [p?lagsy](#) は、[p?geequ](#) によって計算されたスケール係数を使用して、一般矩形行列をスケーリングする。

## 入力パラメータ

- $n$  (グローバル) INTEGER。処理される行数と列数。  
すなわち、分散部分行列  $\text{sub}(A)$  の次数 ( $n \geq 0$ )。
- $a$  (ローカル)  
REAL ([pspoequ](#) の場合)  
DOUBLE PRECISION ([pdpoequ](#) の場合)  
COMPLEX ([pcpoequ](#) の場合)  
DOUBLE COMPLEX ([pzpoequ](#) の場合)。  
  
ローカルメモリにある、ローカル次元  $a(\text{lld\_a}, LOC_c(\text{ja}+n-1))$  の配列へのポインタ。  
  
配列  $a$  は、スケール係数を計算する  $n \times n$  の対称 / エルミート正定値分散行列  $\text{sub}(A)$  を含む。 $\text{sub}(A)$  の対角成分だけが参照される。
- $ia, ja$  (グローバル) INTEGER。それぞれ、部分行列  $\text{sub}(A)$  の最初の行と最初の列を示す、グローバル配列  $A$  の行インデックスと列インデックス。
- $desca$  (グローバルおよびローカル) INTEGER。配列、次元は ( $\text{dlen\_}$ )。分散行列  $A$  の配列ディスクリプタ。

## 出力パラメータ

- $sr, sc$  (ローカル) REAL (単精度の場合)  
DOUBLE PRECISION (倍精度の場合)。  
配列、次元はそれぞれ  $LOC_r(m\_a)$  および  $LOC_c(n\_a)$ 。  
 $\text{info} = 0$  の場合、配列  $sr(ia:ia+n-1)$  には、 $\text{sub}(A)$  の行スケール係数が格納される。 $sr$  は分散行列  $A$  とアライメントされ、すべてのプロセス列に複製される。 $sr$  は、分散行列  $A$  に関連付けられる。

	<p><math>info = 0</math> の場合、配列 <math>sc(ja:ja+n-1)</math> には、<math>sub(A)</math> の列スケール係数が格納される。<math>sc</math> は分散行列 <math>A</math> とアライメントされ、すべてのプロセス行に複製される。<math>sc</math> は、分散行列 <math>A</math> に関連付けられる。</p>
<i>scond</i>	<p>(グローバル) REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 <math>info = 0</math> の場合、<i>scond</i> には最小の <math>sr(i)</math> (または <math>sc(j)</math>) を最大の <math>sr(i)</math> (または <math>sc(j)</math>) で割った値が格納される (<math>ia \leq i \leq ia+n-1</math> および <math>ja \leq j \leq ja+n-1</math>)。 <math>scond \geq 0.1</math> かつ <i>amax</i> が大きすぎる値でも小さすぎる値でもない場合、<math>sr</math> (または <math>sc</math>) によるスケーリングは不要である。</p>
<i>amax</i>	<p>(グローバル) REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 最大行列成分の絶対値。<i>amax</i> がオーバーフローまたはアンダーフローに非常に近い場合、行列をスケーリングする必要がある。</p>
<i>info</i>	<p>(グローバル) INTEGER。 <math>info = 0</math> の場合、実行は正常に終了したことを示す。</p> <p><math>info &lt; 0</math> の場合： <math>i</math> 番目の引数が配列で、<math>j</math> 番目の値が不正だった場合、<math>info = -(i*100+j)</math>。<math>i</math> 番目の引数がスカラで値が不正だった場合、<math>info = -i</math>。</p> <p><math>info &gt; 0</math> の場合： <math>info = k</math> の場合、<math>sub(A)</math> の <math>k</math> 番目の対角エントリが正の値ではない。</p>

## 直交因子分解

このセクションでは、行列の  $QR$  ( $RQ$ ) 因子分解と  $LQ$  ( $QL$ ) 因子分解を行うための ScaLAPACK ルーチンについて説明する。一般化された  $QR$  因子分解と  $RQ$  因子分解と同様に  $RZ$  因子分解用のルーチンも含まれている。因子分解の数学的な定義については、それぞれの LAPACK セクションを参照するか、[SLUG] を参照のこと。

表 5-1 に、行列の直交因子分解を実行するための ScaLAPACK ルーチンを示す。

表 6-3 直交因子分解用の計算ルーチン

行列のタイプ、因子分解	因子分解 (ピボット演算なし)	因子分解 (ピボット演算あり)	行列 Q の生成	行列 Q の適用
一般行列、 QR 因子分解	<a href="#">p?geqrf</a>	<a href="#">p?geqpf</a>	<a href="#">p?orgqr</a> <a href="#">p?ungqr</a>	<a href="#">p?ormqr</a> <a href="#">p?unmqr</a>
一般行列、 RQ 因子分解	<a href="#">p?gerqf</a>		<a href="#">p?orgrq</a> <a href="#">p?ungrq</a>	<a href="#">p?ormrq</a> <a href="#">p?unmrq</a>
一般行列、 LQ 因子分解	<a href="#">p?gelqf</a>		<a href="#">p?orglq</a> <a href="#">p?unglq</a>	<a href="#">p?ormlq</a> <a href="#">p?unmlq</a>
一般行列、 QL 因子分解	<a href="#">p?geqlf</a>		<a href="#">p?orgql</a> <a href="#">p?ungql</a>	<a href="#">p?ormql</a> <a href="#">p?unmql</a>
台形行列、 RZ 因子分解	<a href="#">p?tzzrf</a>			<a href="#">p?ormrz</a> <a href="#">p?unmrz</a>
行列のペア、 汎用 QR 因子分解	<a href="#">p?ggqrf</a>			
行列のペア、 汎用 RQ 因子分解	<a href="#">p?ggrqf</a>			

## p?geqrf

$m \times n$  の一般行列の  $QR$  因子分解を行う。

### 構文

```
call psgeqrf( m, n, a, ia, ja, desca, tau, work, lwork, info )
call pdgeqrf( m, n, a, ia, ja, desca, tau, work, lwork, info )
call pcgeqrf( m, n, a, ia, ja, desca, tau, work, lwork, info )
call pzgeqrf( m, n, a, ia, ja, desca, tau, work, lwork, info )
```

## 説明

このルーチンは、次の式に従って、 $m \times n$  の一般分散行列  $\text{sub}(A) = A(\text{ia}:\text{ia}+m-1, \text{ja}:\text{ja}+n-1)$  の  $QR$  因子分解を行う。

$$A = QR$$

## 入力パラメータ

<i>m</i>	(グローバル) INTEGER。 分散部分行列 $\text{sub}(A)$ の行数 ( $m \geq 0$ )。
<i>n</i>	(グローバル) INTEGER。 分散部分行列 $\text{sub}(A)$ の列数 ( $n \geq 0$ )。
<i>a</i>	(ローカル) REAL (psgeqrf の場合) DOUBLE PRECISION (pdgeqrf の場合) COMPLEX (pcgeqrf の場合) DOUBLE COMPLEX (pzgeqrf の場合)。 ローカルメモリにある、ローカル次元 ( <i>lld_a</i> , <i>LOCc(ja+n-1)</i> ) の配列へのポインタ。 因子分解する分散行列 $\text{sub}(A)$ のローカル部分を含む。
<i>ia, ja</i>	(グローバル) INTEGER。 それぞれ、部分行列 $A(\text{ia}:\text{ia}+m-1, \text{ja}:\text{ja}+n-1)$ の最初の行と最初の列を示すグローバル配列 <i>a</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。 分散行列 <i>A</i> の配列ディスクリプタ。
<i>work</i>	(ローカル)。 REAL (psgeqrf の場合) DOUBLE PRECISION (pdgeqrf の場合) COMPLEX (pcgeqrf の場合) DOUBLE COMPLEX (pzgeqrf の場合) 次元 <i>lwork</i> のワークスペース配列。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 <i>work</i> の次元は $\text{lwork} \geq \text{nb\_a} * (\text{mp0} + \text{nq0} + \text{nb\_a})$ でなければならない。 ここで、  $\text{irow} = \text{mod}(\text{ia}-1, \text{mb\_a})$ 、 $\text{icoff} = \text{mod}(\text{ja}-1, \text{nb\_a})$ 、 $\text{iarow} = \text{indxg2p}(\text{ia}, \text{mb\_a}, \text{MYROW}, \text{rsrc\_a}, \text{NPROW})$ 、 $\text{iacol} = \text{indxg2p}(\text{ja}, \text{nb\_a}, \text{MYCOL}, \text{csrc\_a}, \text{NPCOL})$ 、

$mp0 = \text{numroc}(m + iroff, mb\_a, MYROW, iarow, NPROW)$ 、  
 $np0 = \text{numroc}(n + icoff, nb\_a, MYCOL, iacol, NPCOL)$   
 $\text{numroc}$  および  $\text{indxg2p}$  は、ScaLAPACK ツール関数である。  
 $MYROW$ 、 $MYCOL$ 、 $NPROW$  および  $NPCOL$  は、サブルーチン  
 $\text{blacs\_gridinfo}$  を呼び出して決定できる。

$lwork = -1$  の場合、 $lwork$  はグローバル入力であり、ワークスペース  
のクエリとみなされ、ルーチンはすべての  $work$  配列の最小かつ最適  
なサイズだけを計算する。各値は該当する  $work$  配列の最初のエン  
トリとして返され、[pxerbla](#) はエラー・メッセージを生成しない。

## 出力パラメータ

$a$	$\text{sub}(A)$ の対角成分とその上の成分は、 $\min(m, n) \times n$ の上台形行列 $R$ ( $m \geq n$ の場合、 $R$ は上三角行列) によって上書きされる。対角より下 の成分は、配列 $\text{tau}$ とともに、基本リフレクタの積として直交/ユニ タリ行列 $Q$ を表す。(次の「アプリケーション・ノート」を参照)。
$\text{tau}$	(ローカル) $\text{REAL}$ ( $\text{psgeqrf}$ の場合) $\text{DOUBLE PRECISION}$ ( $\text{pdgeqrf}$ の場合) $\text{COMPLEX}$ ( $\text{pcgeqrf}$ の場合) $\text{DOUBLE COMPLEX}$ ( $\text{pzgeqrf}$ の場合)。 配列、次元は $LOCc(ja + \min(m, n) - 1)$ 。 基本リフレクタのスカラー係数 $\text{tau}$ が格納される。 $\text{tau}$ は、分散行列 $A$ に関連付けられる。
$\text{work}(1)$	終了時に、 $\text{work}(1)$ には、最適なパフォーマンスを得るために必要な $lwork$ の最小値が格納される。
$\text{info}$	(グローバル) $\text{INTEGER}$ 。 $\text{info} = 0$ の場合、実行は正常に終了したことを示す。 $\text{info} < 0$ の場合: $i$ 番目の引数が配列で、 $j$ 番目の値が不正だった場合、 $\text{info} = -(i * 100 + j)$ 。 $i$ 番目の引数がスカラーで値が不正だった場合、 $\text{info} = -i$ 。

## アプリケーション・ノート

行列  $Q$  は、以下の基本リフレクタの積として表現される。

$$Q = H(ja) H(ja+1) \dots H(ja+k-1)$$

ここで、 $k = \min(m, n)$  である。

それぞれの  $H(i)$  は次のような形式を持つ。

$$H(j) = I - \tau * v * v'$$

ここで、 $\tau$  は実 / 複素スカラ、 $v$  は実 / 複素ベクトルで、 $v(1:i-1) = 0$  および  $v(i) = 1$ 。  
終了時に、 $v(i+1:m)$  は  $A(ia+i:ia+m-1, ja+i-1)$  に、 $\tau$  は  $\tau(ja+i-1)$  に格納される。

---

### p?geqpf

$m \times n$  の一般行列の  $QR$  因子分解をピボット演算付きで行う。

---

#### 構文

```
call psgeqpf( m, n, a, ia, ja, desca, ipiv, tau, work, lwork, info )
call pdgeqpf( m, n, a, ia, ja, desca, ipiv, tau, work, lwork, info )
call pcgeqpf( m, n, a, ia, ja, desca, ipiv, tau, work, lwork, info )
call pzgeqpf( m, n, a, ia, ja, desca, ipiv, tau, work, lwork, info )
```

#### 説明

このルーチンは、次の式に従って、 $m \times n$  の一般分散行列  $\text{sub}(A) = A(ia:ia+m-1, ja:ja+n-1)$  の列ピボット演算を用いた  $QR$  因子分解を行う。

$$\text{sub}(A) P = Q R$$

#### 入力パラメータ

$m$	(グローバル) INTEGER。 部分行列 $\text{sub}(A)$ の行数 ( $m \geq 0$ )。
$n$	(グローバル) INTEGER。 部分行列 $\text{sub}(A)$ の列数 ( $n \geq 0$ )。
$a$	(ローカル) REAL (psgeqpf の場合) DOUBLE PRECISION (pdgeqpf の場合) COMPLEX (pcgeqpf の場合) DOUBLE COMPLEX (pzgeqpf の場合)。



	ローカルメモリにある、ローカル次元 ( $lld\_a$ , $LOCc(ja+n-1)$ ) の配列へのポインタ。 因子分解する分散行列 $sub(A)$ のローカル部分を含む。
$ia, ja$	(グローバル) INTEGER。 それぞれ、部分行列 $A(ia:ia+m-1, JA:JA+n-1)$ の最初の行と最初の列を示すグローバル配列 $a$ の行インデックスと列インデックス。
$desca$	(グローバルおよびローカル) INTEGER。配列、次元は ( $dlen\_$ )。 分散行列 $A$ の配列ディスクリプタ。
$work$	(ローカル)。 REAL (psgeqpf の場合) DOUBLE PRECISION (pdgeqpf の場合) COMPLEX (pcgeqpf の場合) DOUBLE COMPLEX (pzgeqpf の場合) 次元 $lwork$ のワークスペース配列。
$lwork$	(ローカルまたはグローバル) INTEGER。 $work$ の次元は 実数型の場合、 $lwork \geq \max(3, mp0+nq0) + LOCc(ja+n-1) + nq0$ でなければならない。 複素数型の場合、 $lwork \geq \max(3, mp0+nq0)$ でなければならない。  ここで、 $iroff = \text{mod}(ia-1, mb\_a)$ , $icoff = \text{mod}(ja-1, nb\_a)$ , $iarow = \text{indxg2p}(ia, mb\_a, MYROW, rsrc\_a, NPROW)$ , $iacol = \text{indxg2p}(ja, nb\_a, MYCOL, csrc\_a, NPCOL)$ , $mp0 = \text{numroc}(m+iroff, mb\_a, MYROW, iarow, NPROW)$ , $nq0 = \text{numroc}(n+icoff, nb\_a, MYCOL, iacol, NPCOL)$ , $LOCc(ja+n-1) = \text{numroc}(ja+n-1, nb\_a, MYCOL, csrc\_a, NPCOL)$ $\text{numroc}$ および $\text{indxg2p}$ は、ScaLAPACK ツール関数である。 $MYROW$ , $MYCOL$ , $NPROW$ および $NPCOL$ は、サブルーチン $\text{blacs\_gridinfo}$ を呼び出して決定できる。  $lwork = -1$ の場合、 $lwork$ はグローバル入力であり、ワークスペースのクエリとみなされ、ルーチンはすべての $work$ 配列の最小かつ最適なサイズだけを計算する。各値は該当する $work$ 配列の最初のエン트리として返され、 <a href="#">pxerbla</a> はエラー・メッセージを生成しない。

## 出力パラメータ

<i>a</i>	<i>sub(A)</i> の対角成分とその上の成分は、 $\min(m,n) \times n$ の上台形行列 <i>R</i> ( $M \geq n$ の場合、 <i>r</i> は上三角行列) によって上書きされる。対角より下の成分は、配列 <i>tau</i> とともに、基本リフレクタの積として直交 / ユニタリ行列 <i>Q</i> を表す。(次の「アプリケーション・ノート」を参照)。
<i>ipiv</i>	(ローカル) INTEGER。配列、次元は <i>LOCc(ja+n-1)</i> 。  <i>ipiv(i) = k</i> 、 <i>sub(A)*P</i> の <i>i</i> 番目のローカル行は、 <i>sub(A)</i> の <i>k</i> 番目のグローバル行。 <i>ipiv</i> は、分散行列 <i>A</i> に関連付けられる。
<i>tau</i>	(ローカル) REAL ( <i>psgeqpf</i> の場合) DOUBLE PRECISION ( <i>pdgeqpf</i> の場合) COMPLEX ( <i>pcgeqpf</i> の場合) DOUBLE COMPLEX ( <i>pzgeqpf</i> の場合)。 配列、次元は <i>LOCc(ja+min(m,n)-1)</i> 。 基本リフレクタのスカラー係数 <i>tau</i> が格納される。 <i>tau</i> は、分散行列 <i>A</i> に関連付けられる。
<i>work(1)</i>	終了時に、 <i>work(1)</i> には、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が格納される。
<i>info</i>	(グローバル) INTEGER。 <i>info = 0</i> の場合、実行は正常に終了したことを示す。 <i>info &lt; 0</i> の場合： <i>i</i> 番目の引数が配列で、 <i>j</i> 番目の値が不正だった場合、 <i>info = -(i*100+j)</i> 。 <i>i</i> 番目の引数がスカラーで値が不正だった場合、 <i>info = -i</i> 。

## アプリケーション・ノート

行列 *Q* は、以下の基本リフレクタの積として表現される。  
 $Q = H(1) H(2) \dots H(n)$

それぞれの *H(i)* は次の形式を持つ。  
 $H = I - \tau * v * v'$

ここで、*tau* は実 / 複素スカラー、*v* は実 / 複素ベクトルで、 $v(1:i-1) = 0$  および  $v(i) = 1$ 。  
 終了時に、 $v(i+1:m)$  は  $A(ia+i:ia+m-1, ja+i-1)$  に格納される。

行列 *P* は、*jpvt* で次のように表現される：*jpvt(j) = i* の場合、*P* の *j* 番目の列は *i* 番目の正当な単位ベクトルである。

## p?orgqr

p?geqrf で求めた  $QR$  因子分解の直交行列  $Q$  を生成する。

### 構文

```
call psorgqr( m, n, k, a, ia, ja, desca, tau, work, lwork, info )
call pdorgqr( m, n, k, a, ia, ja, desca, tau, work, lwork, info )
```

### 説明

このルーチンは、直交列を持つ  $m \times n$  の実分散行列  $Q$  の全体または一部を生成する。これは、以下の次数  $m$  の  $k$  個の基本リフレクタによる積の最初の  $n$  列として定義される。ここで、 $Q$  は  $A(ia:ia+m-1, ja:ja+n-1)$  である。

$$Q = H(1) H(2) \dots H(k)$$

[p?geqrf](#) によって返される。

### 入力パラメータ

- $m$  (グローバル) INTEGER。  
部分行列  $\text{sub}(Q)$  の行数 ( $m \geq 0$ )。
- $n$  (グローバル) INTEGER。  
部分行列  $\text{sub}(Q)$  の列数 ( $m \geq n \geq 0$ )。
- $k$  (グローバル) INTEGER。  
その積が行列  $Q$  となる基本リフレクタの個数 ( $n \geq k \geq 0$ )。
- $a$  (ローカル)  
REAL (psorgqr の場合)  
DOUBLE PRECISION (pdorgqr の場合)  
ローカルメモリにある、ローカル次元 ( $lld\_a$ ,  $LOC(ja+n-1)$ ) の配列へのポインタ。  $j$  番目の列は、p?geqrf によって分散行列引数  $a(ia:*, ja:ja+k-1)$  の  $k$  列に返される、基本リフレクタ  $H(j)$  ( $ja \leq j \leq ja+k-1$ ) を定義するベクトルを含んでいなければならない。
- $ia, ja$  (グローバル) INTEGER。  
それぞれ、部分行列  $A(ia:ia+m-1, ja:ja+n-1)$  の最初の行と最初の列を示すグローバル配列  $a$  の行インデックスと列インデックス。

<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。分散行列 <i>A</i> の配列ディスクリプタ。
<i>tau</i>	(ローカル) REAL (psorgqr の場合) DOUBLE PRECISION (pdorgqr の場合) 配列、次元は <i>LOCc(ja+k-1)</i> 。 <a href="#">p?geqrf</a> によって返される、基本リフレクタ <i>H(j)</i> のスカラ係数 <i>tau(j)</i> を含む。 <i>tau</i> は、分散行列 <i>A</i> に関連付けられる。
<i>work</i>	(ローカル) REAL (psorgqr の場合) DOUBLE PRECISION (pdorgqr の場合) 次元 <i>lwork</i> のワークスペース配列。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 <i>work</i> の次元。 <i>lwork</i> ≥ <i>nb_a</i> * ( <i>nqa0</i> + <i>mpa0</i> + <i>nb_a</i> ) でなければならない。 ここで、  $iroffa = \text{mod}(ia-1, mb\_a), \quad icoffa = \text{mod}(ja-1, nb\_a),$ $iarow = \text{indxg2p}(ia, mb\_a, MYROW, rsrc\_a, NPROW),$ $iacol = \text{indxg2p}(ja, nb\_a, MYCOL, csrc\_a, NPCOL),$ $mpa0 = \text{numroc}(m+iroffa, mb\_a, MYROW, iarow, NPROW),$ $nqa0 = \text{numroc}(n+icoffa, nb\_a, MYCOL, iacol, NPCOL)$ <i>indxg2p</i> および <i>numroc</i> は、ScaLAPACK ツール関数である。MYROW、MYCOL、NPROW および NPCOL は、サブルーチン <i>blacs_gridinfo</i> を呼び出して決定できる。  <i>lwork</i> = -1 の場合、 <i>lwork</i> はグローバル入力であり、ワークスペースのクエリとみなされ、ルーチンはすべての <i>work</i> 配列の最小かつ最適なサイズだけを計算する。各値は該当する <i>work</i> 配列の最初のエントリとして返され、 <a href="#">pxerbla</a> はエラー・メッセージを生成しない。

### 出力パラメータ

<i>a</i>	$m \times n$ の分散行列 <i>Q</i> のローカル部分が格納される。
<i>work(1)</i>	終了時に、 <i>work(1)</i> には、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が格納される。

*info* (グローバル) INTEGER。  
*info* = 0 の場合、実行は正常に終了したことを示す。  
*info* < 0 の場合：  
*i* 番目の引数が配列で、*j* 番目の値が不正だった場合、  
*info* = -(*i*\*100+*j*)。 *i* 番目の引数がスカラで値が不正だった場合、  
*info* = -*i*。

## p?ungqr

p?geqrf で求めた *QR* 因子分解の複素ユニタリ  
 行列 *Q* を生成する。

### 構文

```
call pcungqr( m, n, k, a, ia, ja, desca, tau, work, lwork, info )
call pzungqr( m, n, k, a, ia, ja, desca, tau, work, lwork, info )
```

### 説明

このルーチンは、直交列を持つ  $m \times n$  の複素分散行列 *Q* の全体または一部を生成する。  
 これは、以下の次数 *m* の *k* 個の基本リフレクタの積による最初の *n* 列として定義される。  
 ここで、*Q* は  $A(ia:ia+m-1, ja:ja+n-1)$  である。

$$Q = H(1) H(2) \dots H(k)$$

[p?geqrf](#) によって返される。

### 入力パラメータ

*m* (グローバル) INTEGER。  
 部分行列  $\text{sub}(Q)$  の行数 ( $m \geq 0$ )。

*n* (グローバル) INTEGER。  
 部分行列  $\text{sub}(Q)$  の列数 ( $m \geq n \geq 0$ )。

*k* (グローバル) INTEGER。  
 その積が行列 *Q* となる基本リフレクタの個数 ( $n \geq k \geq 0$ )。

*a* (ローカル)  
 COMPLEX (pcungqr の場合)  
 DOUBLE COMPLEX (pzungqr の場合)  
 ローカルメモリにある、次元 ( $lld\_a, LOCc(ja+n-1)$ ) の配列へのポ

	<p>インタ。 <math>j</math> 番目の列は、<code>p?geqrf</code> によって分散行列引数 <math>a(ia:*,ja:ja+k-1)</math> の <math>k</math> 列に返される、基本リフレクタ <math>H(j)</math> (<math>ja \leq j \leq ja+k-1</math>) を定義するベクトルを含んでいなければならない。</p>
<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 $A$ の最初の行と最初の列を示す、グローバル配列 $a$ の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。分散行列 $A$ の配列ディスクリプタ。
<i>tau</i>	<p>(ローカル)</p> <p>COMPLEX (<code>pcungqr</code> の場合)</p> <p>DOUBLE COMPLEX (<code>pzungqr</code> の場合) 配列、次元は <math>LOCc(ja+k-1)</math>。  <a href="#">p?geqrf</a> によって返される、基本リフレクタ <math>H(j)</math> のスカラー係数 <math>\tau(j)</math> を含む。<math>\tau</math> は、分散行列 <math>A</math> に関連付けられる。</p>
<i>work</i>	<p>(ローカル)</p> <p>COMPLEX (<code>pcungqr</code> の場合)</p> <p>DOUBLE COMPLEX (<code>pzungqr</code> の場合)</p> <p>次元 <i>lwork</i> のワークスペース配列。</p>
<i>lwork</i>	<p>(ローカルまたはグローバル) INTEGER。</p> <p><i>work</i> の次元は <math>lwork \geq nb\_a * (nqa0 + mpa0 + nb\_a)</math> でなければならない。ここで、</p> <p><math>iroffa = \text{mod}(ia-1, mb\_a)</math>、</p> <p><math>icoffa = \text{mod}(ja-1, nb\_a)</math>、</p> <p><math>iarow = \text{indxg2p}(ia, mb\_a, MYROW, rsrc\_a, NPROW)</math>、</p> <p><math>iacol = \text{indxg2p}(ja, nb\_a, MYCOL, csrc\_a, NPCOL)</math>、</p> <p><math>mpa0 = \text{numroc}(m+iroffa, mb\_a, MYROW, iarow, NPROW)</math>、</p> <p><math>nqa0 = \text{numroc}(n+icoffa, nb\_a, MYCOL, iacol, NPCOL)</math></p> <p><code>indxg2p</code> および <code>numroc</code> は、ScaLAPACK ツール関数である。MYROW、MYCOL、NPROW、および NPCOL は、サブルーチン <code>blacs_gridinfo</code> を呼び出して決定できる。</p> <p><math>lwork = -1</math> の場合、<i>lwork</i> はグローバル入力であり、ワークスペースのクエリとみなされ、ルーチンはすべての <i>work</i> 配列の最小かつ最適なサイズだけを計算する。各値は該当する <i>work</i> 配列の最初のエントリとして返され、<a href="#">pxerbla</a> はエラー・メッセージを生成しない。</p>

### 出力パラメータ

<i>a</i>	$m \times n$ の分散行列 $Q$ のローカル部分が格納される。
<i>work</i> (1)	終了時に、 <i>work</i> (1) には、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が格納される。
<i>info</i>	(グローバル) INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> < 0 の場合： <i>i</i> 番目の引数が配列で、 <i>j</i> 番目の値が不正だった場合、 <i>info</i> = -( <i>i</i> *100+ <i>j</i> )。 <i>i</i> 番目の引数がスカラで値が不正だった場合、 <i>info</i> = - <i>i</i> 。

## p?ormqr

一般行列に p?geqrf によって求めた  $QR$  因子  
 分解の直交行列  $Q$  を掛ける。

### 構文

```
call psormqr( side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc,
              descc, work, lwork, info )
call pdormqr( side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc,
              descc, work, lwork, info )
```

### 説明

このルーチンは、 $m \times n$  の一般実分散行列  
 $\text{sub}(C) = C(\text{ic}:\text{ic}+m-1, \text{jc}:\text{jc}+n-1)$  を以下により上書きする。

	<i>side</i> = 'L'	<i>side</i> = 'R'
<i>trans</i> = 'N':	$Q \text{sub}(C)$	$\text{sub}(C) Q$
<i>trans</i> = 'T':	$Q^T \text{sub}(C)$	$\text{sub}(C) Q^T$

ここで、 $Q$  は、 $k$  個の以下の基本リフレクタの積として定義される実直交分散行列である。

$$Q = H(1) H(2) \dots H(k)$$

[p?geqrf](#) によって返される。 $Q$  の次数は、 $side = 'L'$  の場合は  $m$ 、 $side = 'R'$  の場合は  $n$  である。

## 入力パラメータ

<i>side</i>	(グローバル) CHARACTER = 'L' の場合、 $Q$ または $Q^T$ は左側から適用される。 = 'R' の場合、 $Q$ または $Q^T$ は右側から適用される。
<i>trans</i>	(グローバル) CHARACTER = 'N' の場合、 $Q$ が適用される (転置なし)。 = 'T' の場合、 $Q^T$ が適用される (転置)。
<i>m</i>	(グローバル) INTEGER。 分散行列 $\text{sub}(C)$ の行数 ( $m \geq 0$ )。
<i>n</i>	(グローバル) INTEGER。 分散行列 $\text{sub}(C)$ の列数 ( $n \geq 0$ )。
<i>k</i>	(グローバル) INTEGER。 その積が行列 $Q$ を定義する基本リフレクタの個数。次の制約がある。 $side = 'L'$ の場合、 $m \geq k \geq 0$ 、 $side = 'R'$ の場合、 $n \geq k \geq 0$ 。
<i>a</i>	(ローカル) REAL (psormqr の場合) DOUBLE PRECISION (pdormqr の場合)。 ローカルメモリにある、次元 ( $lld\_a$ , $LOCc(ja+k-1)$ ) の配列へのポインタ。 $j$ 番目の列は、 <a href="#">p?geqrf</a> によって分散行列引数 $a(ia:*,ja:ja+k-1)$ の $k$ 列に返される、基本リフレクタ $H(j)$ ( $ja \leq j \leq ja+k-1$ ) を定義するベクトルを含んでいなければならない。 $a(ia:*,ja:ja+k-1)$ はルーチンにより変更されるが終了時に復元される。  $side = 'L'$ の場合、 $lld\_a \geq \max(1, LOCr(ia+m-1))$ $side = 'R'$ の場合、 $lld\_a \geq \max(1, LOCr(ia+n-1))$
<i>ia, ja</i>	(グローバル) INTEGER。 それぞれ、部分行列 $A$ の最初の行と最初の列を示す、グローバル配列 $a$ の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( $dlen\_$ )。 分散行列 $A$ の配列ディスクリプタ。



<i>tau</i>	<p>(ローカル)</p> <p>REAL (psormqr の場合)</p> <p>DOUBLE PRECISION (pdormqr の場合)</p> <p>配列、次元は <math>LOCc(ja+k-1)</math>。</p> <p><a href="#">p?geqrf</a> によって返される、基本リフレクタ <math>H(j)</math> のスカラー係数 <math>tau(j)</math> を含む。<math>tau</math> は、分散行列 <math>A</math> に関連付けられる。</p>
<i>c</i>	<p>(ローカル)</p> <p>REAL (psormqr の場合)</p> <p>DOUBLE PRECISION (pdormqr の場合)</p> <p>ローカルメモリにある、ローカル次元 (<math>lld\_c</math>, <math>LOCc(jc+n-1)</math>) の配列へのポインタ。</p> <p>因子分解する分散行列 <math>sub(C)</math> のローカル部分を含む。</p>
<i>ic, jc</i>	<p>(グローバル) INTEGER。</p> <p>それぞれ、部分行列 <math>C</math> の最初の行と最初の列を示す、グローバル配列 <math>c</math> の行インデックスと列インデックス。</p>
<i>descc</i>	<p>(グローバルおよびローカル) INTEGER。配列、次元は (<math>dlen\_</math>)。</p> <p>分散行列 <math>C</math> の配列ディスクリプタ。</p>
<i>work</i>	<p>(ローカル)</p> <p>REAL (psormqr の場合)</p> <p>DOUBLE PRECISION (pdormqr の場合)。次元 <math>lwork</math> のワークスペース配列。</p>
<i>lwork</i>	<p>(ローカルまたはグローバル) INTEGER。</p> <p><math>work</math> の次元は</p> <p><math>side = 'L'</math> の場合、  <math>lwork \geq \max ((nb\_a*(nb\_a-1))/2, (nqc0 + mpc0)*nb\_a) + nb\_a * nb\_a</math></p> <p><math>side = 'R'</math> の場合、  <math>lwork \geq \max ((nb\_a*(nb\_a-1))/2, (nqc0 + \max (npa0 + \text{numroc}(\text{numroc}(n+icoffc, nb\_a, 0, 0, NPCOL), nb\_a, 0, 0, lcmq), mpc0))*nb\_a) + nb\_a * nb\_a</math></p> <p>でなければならない。</p> <p>ここで、</p> <p><math>lcmq = lcm / NPCOL</math> で <math>lcm = ilcm(NPROW, NPCOL)</math>、</p> <p><math>iroffa = \text{mod}(ia-1, mb\_a)</math>、</p> <p><math>icoffa = \text{mod}(ja-1, nb\_a)</math>、</p>

```

iarow = indxg2p (ia, mb_a, MYROW, rsrc_a, NPROW),
npa0 = numroc(n+iroffa, mb_a, MYROW, iarow, NPROW),
iroffc = mod(ic-1, mb_c),
icoffc = mod(jc-1, nb_c),
icrow = indxg2p(ic, mb_c, MYROW, rsrc_c, NPROW),
iccol = indxg2p(jc, nb_c, MYCOL, csrc_c, NPCOL),
mpc0 = numroc(m+iroffc, mb_c, MYROW, icrow, NPROW),
nqc0 = numroc(n+icoffc, nb_c, MYCOL, iccol, NPCOL)

```

ilcm、indxg2p および numroc は、ScaLAPACK ツール関数である。  
MYROW、MYCOL、NPROW および NPCOL は、サブルーチン  
blacs\_gridinfo を呼び出して決定できる。

lwork = -1 の場合、lwork はグローバル入力であり、ワークスペース  
のクエリとみなされ、ルーチンはすべての work 配列の最小かつ最適  
なサイズだけを計算する。各値は該当する work 配列の最初のエン  
トリとして返され、[pxerbla](#) はエラー・メッセージを生成しない。

### 出力パラメータ

<i>c</i>	積 $Q * \text{sub}(C)$ 、 $Q^T \text{sub}(C)$ 、 $\text{sub}(C) * Q^T$ 、または $\text{sub}(C) * Q$ のいずれかに よって上書きされる。
<i>work(1)</i>	終了時に、 <i>work(1)</i> には、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が格納される。
<i>info</i>	(グローバル) INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> < 0 の場合： <i>i</i> 番目の引数が配列で、 <i>j</i> 番目の値が不正だった場合、 <i>info</i> = -( <i>i</i> *100+ <i>j</i> )。 <i>i</i> 番目の引数がスカラで値が不正だった場合、 <i>info</i> = - <i>i</i> 。

## p?unmqr

複素行列に p?geqrf で求めた  $QR$  因子分解のユニタリ行列  $Q$  を掛ける。

### 構文

```
call cunmqr( side,trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc,
             descc, work, lwork, info )
call zunmqr( side,trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc,
             descc, work, lwork, info )
```

### 説明

このルーチンは、 $m \times n$  の一般複素分散行列  $\text{sub}(C) = C(ic:ic+m-1, jc:jc+n-1)$  を以下により上書きする。

	$side = 'L'$	$side = 'R'$
$trans = 'N':$	$Q \text{ sub}(C)$	$\text{sub}(C) Q$
$trans = 'T':$	$Q^H \text{ sub}(C)$	$\text{sub}(C) Q^H$

ここで、 $Q$  は、 $k$  個の以下の基本リフレクタの積として定義される複素ユニタリ分散行列である。

$$Q = H(1) H(2) \dots H(k)$$

[p?geqrf](#) によって返される。 $Q$  の次数は、 $side = 'L'$  の場合は  $m$ 、 $side = 'R'$  の場合は  $n$  である。

### 入力パラメータ

**side** (グローバル) CHARACTER  
 $= 'L'$  の場合、 $Q$  または  $Q^H$  は左側から適用される。  
 $= 'R'$  の場合、 $Q$  または  $Q^H$  は右側から適用される。

**trans** (グローバル) CHARACTER  
 $= 'N'$  の場合、 $Q$  が適用される (転置なし)。  
 $= 'C'$  の場合、 $Q^H$  が適用される (共役転置)。

**m** (グローバル) INTEGER。  
分散行列  $\text{sub}(C)$  の行数 ( $m \geq 0$ )。

$n$	(グローバル) INTEGER。 分散行列 $\text{sub}(C)$ の列数 ( $n \geq 0$ )。
$k$	(グローバル) INTEGER。その積が行列 $Q$ を定義する基本リフレクタの個数。次の制約がある。 $\text{side} = 'L'$ の場合、 $m \geq k \geq 0$ 、 $\text{side} = 'R'$ の場合、 $n \geq k \geq 0$ 。
$a$	(ローカル) COMPLEX (pcunmqr の場合) DOUBLE COMPLEX (pzunmqr の場合)。 ローカルメモリにある、次元 ( $l1d\_a$ , $LOCc(ja+k-1)$ ) の配列へのポインタ。 $j$ 番目の列は、 <a href="#">p?gegrf</a> によって分散行列引数 $a(ia:*,ja:ja+k-1)$ の $k$ 列に返される、基本リフレクタ $H(j)$ ( $ja \leq j \leq ja+k-1$ ) を定義するベクトルを含んでいなければならない。 $a(ia:*,ja:ja+k-1)$ はルーチンにより変更されるが終了時に復元される。  $\text{side} = 'L'$ の場合、 $l1d\_a \geq \max(1, LOCr(ia+m-1))$ $\text{side} = 'R'$ の場合、 $l1d\_a \geq \max(1, LOCr(ia+n-1))$
$ia, ja$	(グローバル) INTEGER。 それぞれ、部分行列 $A$ の最初の行と最初の列を示す、グローバル配列 $a$ の行インデックスと列インデックス。
$desca$	(グローバルおよびローカル) INTEGER。配列、次元は ( $dlen\_$ )。 分散行列 $A$ の配列ディスクリプタ。
$\tau$	(ローカル) COMPLEX (pcunmqr の場合) DOUBLE COMPLEX (pzunmqr の場合) 配列、次元は $LOCc(ja+k-1)$ 。 <a href="#">p?gegrf</a> によって返される、基本リフレクタ $H(j)$ のスカラー係数 $\tau(j)$ を含む。 $\tau$ は、分散行列 $A$ に関連付けられる。
$c$	(ローカル) COMPLEX (pcunmqr の場合) DOUBLE COMPLEX (pzunmqr の場合)。 ローカルメモリにある、ローカル次元 ( $l1d\_c$ , $LOCc(jc+n-1)$ ) の配列へのポインタ。  因子分解する分散行列 $\text{sub}(C)$ のローカル部分を含む。
$ic, jc$	(グローバル) INTEGER。 それぞれ、部分行列 $C$ の最初の行と最初の列を示す、グローバル配列 $c$ の行インデックスと列インデックス。

*descc* (グローバルおよびローカル) INTEGER。配列、次元は (*dlen\_*)。分散行列 *C* の配列ディスクリプタ。

*work* (ローカル)  
COMPLEX (*pcunmqr* の場合)  
DOUBLE COMPLEX (*pzunmqr* の場合)。次元 *lwork* のワークスペース配列。

*lwork* (ローカルまたはグローバル) INTEGER。  
*work* の次元は

*side* = 'L' の場合、  
 $lwork \geq \max ((nb\_a * (nb\_a - 1)) / 2, (nqc0 + mpc0) * nb\_a) + nb\_a * nb\_a$

*side* = 'R' の場合、  
 $lwork \geq \max ((nb\_a * (nb\_a - 1)) / 2, (nqc0 + \max (npa0 + \text{numroc}(\text{numroc}(n + iroffa, nb\_a, 0, 0, NPCOL), nb\_a, 0, 0, lcmq), mpc0)) * nb\_a) + nb\_a * nb\_a$

でなければならない。

ここで、

$lcmq = lcm / NPCOL$  で  $lcm = ilcm(NPROW, NPCOL)$ 、  
 $iroffa = \text{mod}(ia - 1, mb\_a)$ 、  
 $icoffa = \text{mod}(ja - 1, nb\_a)$ 、  
 $iarow = \text{indxg2p}(ia, mb\_a, MYROW, rsrc\_a, NPROW)$ 、  
 $npa0 = \text{numroc}(n + iroffa, mb\_a, MYROW, iarow, NPROW)$ 、  
 $iroffc = \text{mod}(ic - 1, mb\_c)$ 、  
 $icoffc = \text{mod}(jc - 1, nb\_c)$ 、  
 $icrow = \text{indxg2p}(ic, mb\_c, MYROW, rsrc\_c, NPROW)$ 、  
 $iccol = \text{indxg2p}(jc, nb\_c, MYCOL, csrc\_c, NPCOL)$ 、  
 $mpc0 = \text{numroc}(m + iroffc, mb\_c, MYROW, icrow, NPROW)$ 、  
 $nqc0 = \text{numroc}(n + icoffc, nb\_c, MYCOL, iccol, NPCOL)$

*ilcm*、*indxg2p* および *numroc* は、ScaLAPACK ツール関数である。  
*MYROW*、*MYCOL*、*NPROW* および *NPCOL* は、サブルーチン *blacs\_gridinfo* を呼び出して決定できる。

$lwork = -1$  の場合、 $lwork$  はグローバル入力であり、ワークスペースのクエリとみなされ、ルーチンはすべての  $work$  配列の最小かつ最適なサイズだけを計算する。各値は該当する  $work$  配列の最初のエントリとして返され、[pxerbla](#) はエラー・メッセージを生成しない。

### 出力パラメータ

$c$	積 $Q * \text{sub}(C)$ 、 $Q^H \text{sub}(C)$ 、 $\text{sub}(C) * Q^H$ 、または $\text{sub}(C) * Q$ のいずれかによって上書きされる。
$work(1)$	終了時に、 $work(1)$ には、最適なパフォーマンスを得るために必要な $lwork$ の最小値が格納される。
$info$	(グローバル) INTEGER。 $info = 0$ の場合、実行は正常に終了したことを示す。 $info < 0$ の場合： $i$ 番目の引数が配列で、 $j$ 番目の値が不正だった場合、 $info = -(i * 100 + j)$ 。 $i$ 番目の引数がスカラで値が不正だった場合、 $info = -i$ 。

---

## p?gelqf

一般矩形行列の  $LQ$  因子分解を行う。

---

### 構文

```
call psgelqf( m, n, a, ia, ja, desca, tau, work, lwork, info )
call pdgelqf( m, n, a, ia, ja, desca, tau, work, lwork, info )
call pcgelqf( m, n, a, ia, ja, desca, tau, work, lwork, info )
call pzgelqf( m, n, a, ia, ja, desca, tau, work, lwork, info )
```

### 説明

このルーチンは、次の式に従って、 $m \times n$  の実 / 複素分散行列  $\text{sub}(A) = A(ia:ia+m-1, ia:ia+n-1) = L * Q$  の  $LQ$  因子分解を行う。

### 入力パラメータ

$m$	(グローバル) INTEGER。 部分行列 $\text{sub}(Q)$ の行数 ( $m \geq 0$ )。
-----	--

<i>n</i>	(グローバル) INTEGER。 部分行列 $\text{sub}(Q)$ の列数 ( $n \geq 0$ )。
<i>k</i>	(グローバル) INTEGER。 その積が行列 $Q$ となる基本リフレクタの個数 ( $n \geq k \geq 0$ )。
<i>a</i>	(ローカル)  REAL (psgelqf の場合) DOUBLE PRECISION (pdgelqf の場合) COMPLEX (pcgelqf の場合) DOUBLE COMPLEX (pzgelqf の場合) ローカルメモリにある、ローカル次元 ( $lld\_a$ , $LOCc(ja+n-1)$ ) の配列へのポインタ。因子分解する分散行列 $\text{sub}(A)$ のローカル部分を含む。
<i>ia, ja</i>	(グローバル) INTEGER。 それぞれ、部分行列 $A(ia:ia+m-1, ia:ia+n-1)$ の最初の行と最初の列を示すグローバル配列 <i>a</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。 分散行列 $A$ の配列ディスクリプタ。
<i>work</i>	(ローカル)  REAL (psgelqf の場合) DOUBLE PRECISION (pdgelqf の場合) COMPLEX (pcgelqf の場合) DOUBLE COMPLEX (pzgelqf の場合) 次元 <i>lwork</i> のワークスペース配列。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 <i>work</i> の次元は $lwork \geq mb\_a * (mp0 + nq0 + mb\_a)$ でなければならない。 ここで、  $iroff = \text{mod}(ia-1, mb\_a)$ 、 $icoff = \text{mod}(ja-1, nb\_a)$ 、 $iarow = \text{indxg2p}(ia, mb\_a, \text{MYROW}, rsrc\_a, \text{NPROW})$ 、 $iacol = \text{indxg2p}(ja, nb\_a, \text{MYCOL}, csrc\_a, \text{NPCOL})$ 、 $mp0 = \text{numroc}(m+iroff, mb\_a, \text{MYROW}, iarow, \text{NPROW})$ 、 $nq0 = \text{numroc}(n+icoff, nb\_a, \text{MYCOL}, iacol, \text{NPCOL})$

indxg2p および numroc は、ScaLAPACK ツール関数である。MYROW、MYCOL、NPROW、および NPCOL は、サブルーチン blacs\_gridinfo を呼び出して決定できる。

$lwork = -1$  の場合、 $lwork$  はグローバル入力であり、ワークスペースのクエリとみなされ、ルーチンはすべての  $work$  配列の最小かつ最適なサイズだけを計算する。各値は該当する  $work$  配列の最初のエントリとして返され、[pxerbla](#) はエラー・メッセージを生成しない。

## 出力パラメータ

$a$	sub(A) の対角成分とその下の成分は、 $m \times \min(m,n)$ の下台形行列 $L$ ( $m \leq n$ の場合、 $L$ は下三角行列) によって上書きされる。対角より上の成分は、配列 $\tau$ とともに、基本リフレクタの積として直交/ユニタリの行列 $Q$ を表す。(次の「アプリケーション・ノート」を参照)。
$\tau$	(ローカル) REAL (psgelqf の場合) DOUBLE PRECISION (pdgelqf の場合) COMPLEX (pcgelqf の場合) DOUBLE COMPLEX (pzgelqf の場合) 配列、次元は $LOCr(ia+\min(m,n)-1)$ 。 基本リフレクタのスカラ係数が格納される。 $\tau$ は、分散行列 $A$ に関連付けられる。
$work(1)$	終了時に、 $work(1)$ には、最適なパフォーマンスを得るために必要な $lwork$ の最小値が格納される。
$info$	(グローバル) INTEGER。 $info = 0$ の場合、実行は正常に終了したことを示す。 $info < 0$ の場合: $i$ 番目の引数が配列で、 $j$ 番目の値が不正だった場合、 $info = -(i*100+j)$ 。 $i$ 番目の引数がスカラで値が不正だった場合、 $info = -i$ 。

## アプリケーション・ノート

行列  $Q$  は、以下の基本リフレクタの積として表現される。

$$Q = H(ia+k-1) H(ia+k-2) \dots H(ia)$$

ここで、 $k = \min(m,n)$  である。

それぞれの  $H(i)$  は次のような形式を持つ。



$$H(i) = I - \tau v v'$$

ここで、 $\tau$  は実 / 複素スカラー、 $v$  は実 / 複素ベクトルで、 $v(1:i-1) = 0$  および  $v(i) = 1$ 。  
終了時に、 $v(i+1:n)$  は  $A(ia+i-1:ia+i-1, ja+n-1)$  に、 $\tau$  は  $\tau(ia+i-1)$  に格納される。

## p?orglq

p?gelqf で求めた  $LQ$  因子分解の実直交行列  $Q$  を生成する。

### 構文

```
call psorglq( m, n, k, a, ia, ja, desca, tau, work, lwork, info )
call pdorglq( m, n, k, a, ia, ja, desca, tau, work, lwork, info )
```

### 説明

このルーチンは、直交列を持つ  $m \times n$  の実分散行列  $Q$  の全体または一部を生成する。これは、以下の次数  $m$  の  $k$  個の基本リフレクタの積による最初の  $n$  列として定義される。ここで、 $Q$  は  $A(ia:ia+m-1, ja:ja+n-1)$  である。

$$Q = H(k) \dots H(2) H(1)$$

[p?gelqf](#) によって返される。

### 入力パラメータ

$m$	(グローバル) INTEGER。 部分行列 $\text{sub}(Q)$ の行数 ( $m \geq 0$ )。
$n$	(グローバル) INTEGER。 部分行列 $\text{sub}(Q)$ の列数 ( $n \geq m \geq 0$ )。
$k$	(グローバル) INTEGER。 その積が行列 $Q$ となる基本リフレクタの個数 ( $m \geq k \geq 0$ )。
$a$	(ローカル) REAL (psorglq の場合) DOUBLE PRECISION (pdorglq の場合) ローカルメモリにある、ローカル次元 ( $lld\_a$ , $LOCc(ja+n-1)$ ) の配

列へのポインタ。呼び出し時に、 $i$  番目の行は、`p?gelqf` によって分散行列引数  $A(ia:ia+k-1, ja:*)$  の  $k$  行に返される、基本リフレクタ  $H(i)$  ( $ia \leq i \leq ia+k-1$ ) を定義するベクトルを含んでいなければならない。

*ia, ja* (グローバル) INTEGER。  
それぞれ、部分行列  $A(ia:ia+m-1, ja:ja+n-1)$  の最初の行と最初の列を示すグローバル配列 *a* の行インデックスと列インデックス。

*desca* (グローバルおよびローカル) INTEGER。配列、次元は (*dlen\_*)。分散行列  $A$  の配列ディスクリプタ。

*work* (ローカル)

REAL (`psorglq` の場合)  
DOUBLE PRECISION (`pdorglq` の場合)  
次元 *lwork* のワークスペース配列。

*lwork* (ローカルまたはグローバル) INTEGER。  
*work* の次元は  $lwork \geq mb\_a * (mpa0 + nqa0 + mb\_a)$  でなければならない。ここで、

$iroffa = \text{mod}(ia-1, mb\_a)$ 、

$icoffa = \text{mod}(ja-1, nb\_a)$ 、

$iarow = \text{indxg2p}(ia, mb\_a, \text{MYROW}, rsrc\_a, \text{NPROW})$ 、

$iacol = \text{indxg2p}(ja, nb\_a, \text{MYCOL}, csrc\_a, \text{NPCOL})$ 、

$mpa0 = \text{numroc}(m + iroffa, mb\_a, \text{MYROW}, iarow, \text{NPROW})$ 、

$nqa0 = \text{numroc}(n + icoffa, nb\_a, \text{MYCOL}, iacol, \text{NPCOL})$

`indxg2p` および `numroc` は、ScaLAPACK ツール関数である。MYROW、MYCOL、NPROW、および NPCOL は、サブルーチン `blacs_gridinfo` を呼び出して決定できる。

*lwork* = -1 の場合、*lwork* はグローバル入力であり、ワークスペースのクエリとみなされ、ルーチンはすべての *work* 配列の最小かつ最適なサイズだけを計算する。各値は該当する *work* 配列の最初のエントリとして返され、[pxerbla](#) はエラー・メッセージを生成しない。

## 出力パラメータ

*a* 因子分解の対象となる  $m \times n$  の分散行列  $Q$  のローカル部分が格納される。

<code>tau</code>	(ローカル) REAL (psorglq の場合) DOUBLE PRECISION (pdorglq の場合) 配列、次元は $LOCr(ia+k-1)$ 。 基本リフレクタ $H(i)$ のスカラー係数 <code>tau</code> が格納される。 <code>tau</code> は、分散行列 $A$ に関連付けられる。
<code>work(1)</code>	終了時に、 <code>work(1)</code> には、最適なパフォーマンスを得るために必要な <code>lwork</code> の最小値が格納される。
<code>info</code>	(グローバル) INTEGER。 <code>info = 0</code> の場合、実行は正常に終了したことを示す。 <code>info &lt; 0</code> の場合： $i$ 番目の引数が配列で、 $j$ 番目の値が不正だった場合、 $info = -(i*100+j)$ 。 $i$ 番目の引数がスカラーで値が不正だった場合、 $info = -i$ 。

## p?unglq

p?gelqf で求めた  $LQ$  因子分解のユニタリ行列  $Q$  を生成する。

### 構文

```
call pcunglq( m, n, k, a, ia, ja, desca, tau, work, lwork, info )
call pzunglq( m, n, k, a, ia, ja, desca, tau, work, lwork, info )
```

### 説明

このルーチンは、直交列を持つ  $m \times n$  の複素分散行列  $Q$  の全体または一部を生成する。これは、以下の次数  $m$  の  $k$  個の基本リフレクタの積による最初の  $n$  列として定義される。ここで、 $Q$  は  $A(ia:ia+m-1, ja:ja+n-1)$  である。

$$Q = H(k) \dots H(2)' H(1)'$$

[p?gelqf](#) によって返される。

### 入力パラメータ

`m` (グローバル) INTEGER。  
部分行列  $\text{sub}(Q)$  の行数 ( $m \geq 0$ )。

<i>n</i>	(グローバル) INTEGER。 部分行列 $\text{sub}(Q)$ の列数 ( $n \geq m \geq 0$ )。
<i>k</i>	(グローバル) INTEGER。 その積が行列 $Q$ となる基本リフレクタの個数 ( $m \geq k \geq 0$ )。
<i>a</i>	(ローカル) COMPLEX (pcunglq の場合) DOUBLE COMPLEX (pzunglq の場合) ローカルメモリにある、ローカル次元 ( $lld\_a$ , $LOCc(ja+n-1)$ ) の配列へのポインタ。呼び出し時に、 $i$ 番目の行は、p?gelqf によって分散行列引数 $A(ia:ia+k-1, ja:*)$ の $k$ 行に返される、基本リフレクタ $H(i)$ ( $ia \leq i \leq ia+k-1$ ) を定義するベクトルを含んでいなければならない。
<i>ia, ja</i>	(グローバル) INTEGER。 それぞれ、部分行列 $A(ia:ia+m-1, ja:ja+n-1)$ の最初の行と最初の列を示すグローバル配列 <i>a</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。 分散行列 $A$ の配列ディスクリプタ。
<i>tau</i>	(ローカル) COMPLEX (pcunglq の場合) DOUBLE COMPLEX (pzunglq の場合) 配列、次元は $LOCr(ia+k-1)$ 。 基本リフレクタ $H(i)$ のスカラ係数 <i>tau</i> が格納される。 <i>tau</i> は、分散行列 $A$ に関連付けられる。
<i>work</i>	(ローカル) COMPLEX (pcunglq の場合) DOUBLE COMPLEX (pzunglq の場合) 次元 <i>lwork</i> のワークスペース配列。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 <i>work</i> の次元は $lwork \geq mb\_a * (mpa0 + nqa0 + mb\_a)$ でなければならない。ここで、  $iroffa = \text{mod}(ia-1, mb\_a),$ $icoffa = \text{mod}(ja-1, nb\_a),$ $iarow = \text{indxg2p}(ia, mb\_a, MYROW, rsrc\_a, NPROW),$ $iacol = \text{indxg2p}(ja, nb\_a, MYCOL, csrc\_a, NPCOL),$ $mpa0 = \text{numroc}(m+iroffa, mb\_a, MYROW, iarow, NPROW),$

```
nqa0 = numroc(n+icoffa, nb_a, MYCOL, iacol, NPCOL)
```

indxg2p および numroc は、ScaLAPACK ツール関数である。MYROW、MYCOL、NPROW、および NPCOL は、サブルーチン blacs\_gridinfo を呼び出して決定できる。

lwork = -1 の場合、lwork はグローバル入力であり、ワークスペースのクエリとみなされ、ルーチンはすべての work 配列の最小かつ最適なサイズだけを計算する。各値は該当する work 配列の最初のエントリとして返され、[pxerbla](#) はエラー・メッセージを生成しない。

### 出力パラメータ

a	因子分解の対象となる $m \times n$ の分散行列 $Q$ のローカル部分が格納される。
work(1)	終了時に、work(1) には、最適なパフォーマンスを得るために必要な lwork の最小値が格納される。
info	(グローバル) INTEGER。 info = 0 の場合、実行は正常に終了したことを示す。 info < 0 の場合： i 番目の引数が配列で、j 番目の値が不正だった場合、 info = -(i*100+j)。i 番目の引数がスカラで値が不正だった場合、 info = -i。

## p?ormlq

一般行列に p?gelqf によって求めた  $LQ$  因子分解のユニタリ行列  $Q$  を掛ける。

### 構文

```
call psormlq( side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc,
              work, lwork, info )
call pdormlq( side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc,
              work, lwork, info )
```

### 説明

このルーチンは、 $m \times n$  の一般実分散行列  $\text{sub}(C) = C(ic:ic+m-1, jc:jc+n-1)$  を以下により上書きする。

	$side = 'L'$	$side = 'R'$
$trans = 'N':$	$Q \text{ sub}(C)$	$\text{sub}(C) Q$
$trans = 'T':$	$Q^T \text{ sub}(C)$	$\text{sub}(C) Q^T$

ここで、 $Q$  は、 $k$  個の以下の基本リフレクタの積として定義される実直交分散行列である。

$$Q = H(k) \dots H(2) H(1)$$

[p?ge1qf](#) によって返される。 $Q$  の次数は、 $side = 'L'$  の場合は  $m$ 、 $side = 'R'$  の場合は  $n$  である。

## 入力パラメータ

$side$	(グローバル) CHARACTER = $'L'$ の場合、 $Q$ または $Q^T$ は左側から適用される。 = $'R'$ の場合、 $Q$ または $Q^T$ は右側から適用される。
$trans$	(グローバル) CHARACTER = $'N'$ の場合、 $Q$ が適用される (転置なし)。 = $'T'$ の場合、 $Q^T$ が適用される (転置)。
$m$	(グローバル) INTEGER。 分散行列 $\text{sub}(C)$ の行数 ( $m \geq 0$ )。
$n$	(グローバル) INTEGER。 分散行列 $\text{sub}(C)$ の列数 ( $n \geq 0$ )。
$k$	(グローバル) INTEGER。 その積が行列 $Q$ を定義する基本リフレクタの個数。次の制約がある。 $side = 'L'$ の場合、 $m \geq k \geq 0$ 、 $side = 'R'$ の場合、 $n \geq k \geq 0$ 。
$a$	(ローカル) REAL (psorm1q の場合) DOUBLE PRECISION (pdorm1q の場合)。 ローカルメモリにある、次元 ( $lld\_a, LOCc(ja+m-1)$ ) ( $side = 'L'$ の場合) または ( $lld\_a, LOCc(ja+n-1)$ ) ( $side = 'R'$ の場合) の配列へのポインタ。 $i$ 番目の行は、 <a href="#">p?ge1qf</a> によって分散行列引数 $a(ia:ia+k-1, ja:*)$ の $k$ 行に返される、基本リフレクタ $H(i)$ ( $ia \leq i \leq ia+k-1$ ) を定義するベクトルを含んでいなければならない。 $a(ia:ia+k-1, ja:*)$ はルーチンにより変更されるが終了時に復元される。

<i>ia, ja</i>	(グローバル) INTEGER。 それぞれ、部分行列 <i>A</i> の最初の行と最初の列を示す、グローバル配列 <i>a</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。 分散行列 <i>A</i> の配列ディスクリプタ。
<i>tau</i>	(ローカル) REAL ( <i>psormlq</i> の場合) DOUBLE PRECISION ( <i>pdormlq</i> の場合) 配列、次元は <i>LOCc(ja+k-1)</i> 。 <a href="#">p?gelqf</a> によって返される、基本リフレクタ $H(i)$ のスカラ係数 $\tau(i)$ を含む。 <i>tau</i> は、分散行列 <i>A</i> に関連付けられる。
<i>c</i>	(ローカル) REAL ( <i>psormlq</i> の場合) DOUBLE PRECISION ( <i>pdormlq</i> の場合) ローカルメモリにある、ローカル次元 ( <i>lld_c</i> , <i>LOCc(jc+n-1)</i> ) の配列へのポインタ。 因子分解する分散行列 <i>sub(C)</i> のローカル部分を含む。
<i>ic, jc</i>	(グローバル) INTEGER。それぞれ、部分行列 <i>C</i> の最初の行と最初の列を示す、グローバル配列 <i>c</i> の行インデックスと列インデックス。
<i>descC</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。 分散行列 <i>C</i> の配列ディスクリプタ。
<i>work</i>	(ローカル) REAL ( <i>psormlq</i> の場合) DOUBLE PRECISION ( <i>pdormlq</i> の場合)。次元 <i>lwork</i> のワークスペース配列。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 <i>work</i> の次元は  <i>side</i> = 'L' の場合、 $lwork \geq \max ((mb\_a*(mb\_a-1))/2, (mpc0 + \max mqa0) + \text{numroc}(\text{numroc}(m + iroffC, mb\_a, 0, 0, NPROW), mb\_a, 0, 0, lcmp), nqc0)) * mb\_a + mb\_a*mb\_a$  <i>side</i> = 'R' の場合、 $lwork \geq \max ((mb\_a*(mb\_a-1))/2, (mpc0 + nqc0) * mb\_a + mb\_a*mb\_a$  でなければならない。 ここで、

```

lcmp = lcm / NPROW で lcm = ilcm (NPROW, NPCOL)、
iroffa = mod(ia-1, mb_a)、
icoffa = mod(ja-1, nb_a)、
iacol = indxg2p (ja, nb_a, MYCOL, csrc_a, NPCOL)、
mqa0 = numroc(m+icoffa, nb_a, MYCOL, iacol, NPCOL)、
iroffc = mod(ic-1, mb_c)、
icoffc = mod(jc-1, nb_c)、
icrow = indxg2p(ic, mb_c, MYROW, rsrc_c, NPROW)、
iccol = indxg2p(jc, nb_c, MYCOL, csrc_c, NPCOL)、
mpc0 = numroc(m+iroffc, mb_c, MYROW, icrow, NPROW)、
nqc0 = numroc(n+icoffc, nb_c, MYCOL, iccol, NPCOL)

```

ilcm、indxg2p および numroc は、ScaLAPACK ツール関数である。  
MYROW、MYCOL、NPROW および NPCOL は、サブルーチン  
blacs\_gridinfo を呼び出して決定できる。

lwork = -1 の場合、lwork はグローバル入力であり、ワークスペース  
のクエリとみなされ、ルーチンはすべての work 配列の最小かつ最適  
なサイズだけを計算する。各値は該当する work 配列の最初のエン  
トリとして返され、[pxerbla](#) はエラー・メッセージを生成しない。

### 出力パラメータ

<i>c</i>	積 $Q * \text{sub}(C)$ 、 $Q' \text{sub}(C)$ 、 $\text{sub}(C) * Q'$ 、または $\text{sub}(C) * Q$ のいずれかに よって上書きされる。
<i>work(1)</i>	終了時に、 <i>work(1)</i> には、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が格納される。
<i>info</i>	(グローバル) INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> < 0 の場合： <i>i</i> 番目の引数が配列で、 <i>j</i> 番目の値が不正だった場合、 <i>info</i> = -( <i>i</i> *100+ <i>j</i> )。 <i>i</i> 番目の引数がスカラで値が不正だった場合、 <i>info</i> = - <i>i</i> 。



## p?unmlq

一般行列に `p?gelqf` によって求めた  $LQ$  因子分解のユニタリ行列  $Q$  を掛ける。

### 構文

```
call pcunmlq( side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc,
              descc, work, lwork, info )
call pzunmlq( side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc,
              descc, work, lwork, info )
```

### 説明

このルーチンは、 $m \times n$  の一般複素分散行列  $\text{sub}(C) = C(\text{ic}:\text{ic}+m-1, \text{jc}:\text{jc}+n-1)$  を以下により上書きする。

	$\text{side} = 'L'$	$\text{side} = 'R'$
$\text{trans} = 'N':$	$Q \text{sub}(C)$	$\text{sub}(C) Q$
$\text{trans} = 'T':$	$Q^H \text{sub}(C)$	$\text{sub}(C) Q^H$

ここで、 $Q$  は、 $k$  個の以下の基本リフレクタの積として定義される複素ユニタリ分散行列である。

$$Q = H(k)' \dots H(2)' H(1)'$$

[p?gelqf](#) によって返される。 $Q$  の次数は、 $\text{side} = 'L'$  の場合は  $m$ 、 $\text{side} = 'R'$  の場合は  $n$  である。

### 入力パラメータ

**side** (グローバル) CHARACTER  
 $= 'L'$  の場合、 $Q$  または  $Q^H$  は左側から適用される。  
 $= 'R'$  の場合、 $Q$  または  $Q^H$  は右側から適用される。

**trans** (グローバル) CHARACTER  
 $= 'N'$  の場合、 $Q$  が適用される (転置なし)。  
 $= 'C'$  の場合、 $Q^H$  が適用される (共役転置)。

**m** (グローバル) INTEGER。  
分散行列  $\text{sub}(C)$  の行数 ( $m \geq 0$ )。

$n$	(グローバル) INTEGER。 分散行列 $\text{sub}(C)$ の列数 ( $n \geq 0$ )。
$k$	(グローバル) INTEGER。 その積が行列 $Q$ を定義する基本リフレクタの個数。次の制約がある。 $\text{side} = 'L'$ の場合、 $m \geq k \geq 0$ 、 $\text{side} = 'R'$ の場合、 $n \geq k \geq 0$ 。
$a$	(ローカル) COMPLEX (pcunmlq の場合) DOUBLE COMPLEX (pzunmlq の場合)。 ローカルメモリにある、次元 ( $\text{lld\_a}$ , $\text{LOCc}(ja+m-1)$ ) ( $\text{side} = 'L'$ の場合) または ( $\text{lld\_a}$ , $\text{LOCc}(ja+n-1)$ ) ( $\text{side} = 'R'$ の場合) の配列へのポインタ。ここで、 $\text{lld\_a} \geq \max(1, \text{LOCr}(ia+k-1))$ である。 $i$ 番目の行は、 <a href="#">p?gelqf</a> によって分散行列引数 $a(ia:ia+k-1, ja:*)$ の $k$ 行に返される、基本リフレクタ $H(i)$ ( $ia \leq i \leq ia+k-1$ ) を定義するベクトルを含んでいなければならない。 $a(ia:ia+k-1, ja:*)$ はルーチンにより変更されるが終了時に復元される。
$ia, ja$	(グローバル) INTEGER。 それぞれ、部分行列 $A$ の最初の行と最初の列を示す、グローバル配列 $a$ の行インデックスと列インデックス。
$\text{desca}$	(グローバルおよびローカル) INTEGER。配列、次元は ( $\text{dlen\_}$ )。 分散行列 $A$ の配列ディスクリプタ。
$\text{tau}$	(ローカル) COMPLEX (pcunmlq の場合) DOUBLE COMPLEX (pzunmlq の場合) 配列、次元は $\text{LOCc}(ia+k-1)$ 。 <a href="#">p?gelqf</a> によって返される、基本リフレクタ $H(i)$ のスカラー係数 $\text{tau}(i)$ を含む。 $\text{tau}$ は、分散行列 $A$ に関連付けられる。
$c$	(ローカル) COMPLEX (pcunmlq の場合) DOUBLE COMPLEX (pzunmlq の場合)。 ローカルメモリにある、ローカル次元 ( $\text{lld\_c}$ , $\text{LOCc}(jc+n-1)$ ) の配列へのポインタ。  因子分解する分散行列 $\text{sub}(C)$ のローカル部分を含む。
$ic, jc$	(グローバル) INTEGER。 それぞれ、部分行列 $C$ の最初の行と最初の列を示す、グローバル配列 $c$ の行インデックスと列インデックス。

*descc* (グローバルおよびローカル) INTEGER。配列、次元は (*dlen\_*)。分散行列 *C* の配列ディスクリプタ。

*work* (ローカル)  
COMPLEX (*pcunmlq* の場合)  
DOUBLE COMPLEX (*pzunmlq* の場合)。次元 *lwork* のワークスペース配列。

*lwork* (ローカルまたはグローバル) INTEGER。  
*work* の次元は

*side* = 'L' の場合、  
 $lwork \geq \max ((mb\_a * (mb\_a - 1)) / 2, (mpc0 + \max mqa0) + \text{numroc}(\text{numroc}(m + iroffa, mb\_a, 0, 0, NPROW), mb\_a, 0, 0, lcmp), nqc0)) * mb\_a + mb\_a * mb\_a$

*side* = 'R' の場合、  
 $lwork \geq \max ((mb\_a * (mb\_a - 1)) / 2, (mpc0 + nqc0) * mb\_a + mb\_a * mb\_a)$

でなければならない。

ここで、

$lcmp = lcm / NPROW$  で  $lcm = ilcm(NPROW, NPCOL)$ 、  
 $iroffa = \text{mod}(ia - 1, mb\_a)$ 、  
 $icoffa = \text{mod}(ja - 1, nb\_a)$ 、  
 $iacol = \text{indxg2p}(ja, nb\_a, MYCOL, csrc\_a, NPCOL)$ 、  
 $mqa0 = \text{numroc}(m + icoffa, nb\_a, MYCOL, iacol, NPCOL)$ 、  
 $iroffc = \text{mod}(ic - 1, mb\_c)$ 、  
 $icoffc = \text{mod}(jc - 1, nb\_c)$ 、  
 $icrow = \text{indxg2p}(ic, mb\_c, MYROW, rsrc\_c, NPROW)$ 、  
 $iccol = \text{indxg2p}(jc, nb\_c, MYCOL, csrc\_c, NPCOL)$ 、  
 $mpc0 = \text{numroc}(m + iroffc, mb\_c, MYROW, icrow, NPROW)$ 、  
 $nqc0 = \text{numroc}(n + icoffc, nb\_c, MYCOL, iccol, NPCOL)$

*ilcm*、*indxg2p* および *numroc* は、ScaLAPACK ツール関数である。  
*MYROW*、*MYCOL*、*NPROW* および *NPCOL* は、サブルーチン *blacs\_gridinfo* を呼び出して決定できる。

$lwork = -1$  の場合、 $lwork$  はグローバル入力であり、ワークスペースのクエリとみなされ、ルーチンはすべての  $work$  配列の最小かつ最適なサイズだけを計算する。各値は該当する  $work$  配列の最初のエントリとして返され、[pxerbla](#) はエラー・メッセージを生成しない。

### 出力パラメータ

$c$	積 $Q * \text{sub}(C)$ 、 $Q' \text{sub}(C)$ 、 $\text{sub}(C) * Q'$ 、または $\text{sub}(C) * Q$ のいずれかによって上書きされる。
$work(1)$	終了時に、 $work(1)$ には、最適なパフォーマンスを得るために必要な $lwork$ の最小値が格納される。
$info$	(グローバル) INTEGER。 $info = 0$ の場合、実行は正常に終了したことを示す。 $info < 0$ の場合： $i$ 番目の引数が配列で、 $j$ 番目の値が不正だった場合、 $info = -(i * 100 + j)$ 。 $i$ 番目の引数がスカラで値が不正だった場合、 $info = -i$ 。

---

## p?geqlf

一般行列の  $QL$  因子分解を行う。

---

### 構文

```
call psgeqlf( m, n, a, ia, ja, desca, tau, work, lwork, info )
call pdgeqlf( m, n, a, ia, ja, desca, tau, work, lwork, info )
call pcgeqlf( m, n, a, ia, ja, desca, tau, work, lwork, info )
call pzgeqlf( m, n, a, ia, ja, desca, tau, work, lwork, info )
```

### 説明

このルーチンは、次の式に従って、 $m \times n$  の実 / 複素分散行列  $\text{sub}(A) = A(ia:ia+m-1, ja:ja+n-1) = Q * L$  の  $QL$  因子分解を行う。

### 入力パラメータ

$m$	(グローバル) INTEGER。 部分行列 $\text{sub}(Q)$ の行数 ( $m \geq 0$ )。
-----	--

<i>n</i>	(グローバル) INTEGER。 部分行列 $\text{sub}(Q)$ の列数 ( $n \geq 0$ )。
<i>a</i>	(ローカル) REAL (psgeqlf の場合) DOUBLE PRECISION (pdgeqlf の場合) COMPLEX (pcgeqlf の場合) DOUBLE COMPLEX (pzgeqlf の場合) ローカルメモリにある、ローカル次元 ( <i>lld_a</i> , <i>LOCc</i> ( <i>ja+n-1</i> )) の配列へのポインタ。因子分解する分散行列 $\text{sub}(A)$ のローカル部分を含む。
<i>ia, ja</i>	(グローバル) INTEGER。 それぞれ、部分行列 $A(\text{ia}:\text{ia}+\text{m}-1, \text{ia}:\text{ia}+\text{n}-1)$ の最初の行と最初の列を示すグローバル配列 <i>a</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。 分散行列 <i>A</i> の配列ディスクリプタ。
<i>work</i>	(ローカル) REAL (psgeqlf の場合) DOUBLE PRECISION (pdgeqlf の場合) COMPLEX (pcgeqlf の場合) DOUBLE COMPLEX (pzgeqlf の場合) 次元 <i>lwork</i> のワークスペース配列。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 <i>work</i> の次元は $\text{lwork} \geq \text{nb\_a} * (\text{mp0} + \text{nq0} + \text{nb\_a})$ でなければならない。 ここで、 $\text{iroff} = \text{mod}(\text{ia}-1, \text{mb\_a}),$ $\text{icoff} = \text{mod}(\text{ja}-1, \text{nb\_a}),$ $\text{iarow} = \text{indxg2p}(\text{ia}, \text{mb\_a}, \text{MYROW}, \text{rsrc\_a}, \text{NPROW}),$ $\text{iacol} = \text{indxg2p}(\text{ja}, \text{nb\_a}, \text{MYCOL}, \text{csrc\_a}, \text{NPCOL}),$ $\text{mp0} = \text{numroc}(\text{m} + \text{iroff}, \text{mb\_a}, \text{MYROW}, \text{iarow}, \text{NPROW}),$ $\text{nq0} = \text{numroc}(\text{n} + \text{icoff}, \text{nb\_a}, \text{MYCOL}, \text{iacol}, \text{NPCOL})$ numroc および indxg2p は、ScaLAPACK ツール関数である。MYROW、MYCOL、NPROW、および NPCOL は、サブルーチン blacs_gridinfo を呼び出して決定できる。

$lwork = -1$  の場合、 $lwork$  はグローバル入力であり、ワークスペースのクエリとみなされ、ルーチンはすべての  $work$  配列の最小かつ最適なサイズだけを計算する。各値は該当する  $work$  配列の最初のエントリとして返され、[pxerbla](#) はエラー・メッセージを生成しない。

## 出力パラメータ

$a$	終了時に、 $m \geq n$ の場合、分散部分行列 $A(ia+m-n:ia+m-1, ja:ja+n-1)$ の下三角部分に、 $n \times n$ の下三角行列 $L$ が格納される。 $m \leq n$ の場合、 $(n-m)$ 番目の優対角成分とその下の成分に、 $m \times n$ 下台形行列 $L$ が格納される。残りの成分は、配列 $\tau$ とともに、基本リフレクタの積として直交/ユニタリ行列 $Q$ を表す。(次の「アプリケーション・ノート」を参照)。
$\tau$	(ローカル) REAL (psgeqlf の場合) DOUBLE PRECISION (pdgeqlf の場合) COMPLEX (pcgeqlf の場合) DOUBLE COMPLEX (pzgeqlf の場合) 配列、次元は $LOCc(ja+n-1)$ 。 基本リフレクタのスカラー係数が格納される。 $\tau$ は、分散行列 $A$ に関連付けられる。
$work(1)$	終了時に、 $work(1)$ には、最適なパフォーマンスを得るために必要な $lwork$ の最小値が格納される。
$info$	(グローバル) INTEGER。 $info = 0$ の場合、実行は正常に終了したことを示す。 $info < 0$ の場合: $i$ 番目の引数が配列で、 $j$ 番目の値が不正だった場合、 $info = -(i*100+j)$ 。 $i$ 番目の引数がスカラーで値が不正だった場合、 $info = -i$ 。

## アプリケーション・ノート

行列  $Q$  は、以下の基本リフレクタの積として表現される。

$$Q = H(ja+k-1) \dots H(ja+1) H(ja)$$

ここで、 $k = \min(m, n)$  である。

それぞれの  $H(i)$  は次のような形式を持つ。

$$H(i) = I - \tau * v * v'$$

ここで、 $\tau$  は実 / 複素スカラ、 $v$  は実 / 複素ベクトルで、 $v(m-k+i+1:m) = 0$  および  $v(m-k+i) = 1$ 。終了時に、 $v(m-k+i-1)$  は  $A(ia:ia+m-k+i-2, ja+n-k+i-1)$  に、 $\tau$  は  $\tau(ja+n-k+i-1)$  に格納される。

## p?orgql

p?geqlf で求めた  $QL$  因子分解の実直交行列  $Q$  を生成する。

### 構文

```
call psorgql( m, n, k, a, ia, ja, desca, tau, work, lwork, info )
call pdorgql( m, n, k, a, ia, ja, desca, tau, work, lwork, info )
```

### 説明

このルーチンは、直交列を持つ  $m \times n$  の実分散行列  $Q$  の全体または一部を生成する。これは、以下の次数  $m$  の  $k$  個の基本リフレクタの積による最初の  $n$  列として定義される。ここで、 $Q$  は  $A(ia:ia+m-1, ja:ja+N-1)$  である。

$$Q = H(k) \dots H(2) H(1)$$

[p?geqlf](#) によって返される。

### 入力パラメータ

$m$  (グローバル) INTEGER。  
部分行列  $\text{sub}(Q)$  の行数 ( $m \geq 0$ )。

$n$  (グローバル) INTEGER。  
部分行列  $\text{sub}(Q)$  の列数 ( $m \geq n \geq 0$ )。

$k$  (グローバル) INTEGER。  
その積が行列  $Q$  となる基本リフレクタの個数 ( $n \geq k \geq 0$ )。

$a$  (ローカル)  
REAL (psorgql の場合)  
DOUBLE PRECISION (pdorgql の場合)  
ローカルメモリにある、ローカル次元 ( $lld\_a$ ,  $LOCc(ja+n-1)$ ) の配列へのポインタ。呼び出し時に、 $j$  番目の列は、[p?geqlf](#) によって分散

行列引数  $A(ia:*ja+n-k:ja+n-1)$  の  $k$  列に返される、基本リフレクタ  $H(j)$  ( $ja+n-k \leq j \leq ja+n-1$ ) を定義するベクトルを含んでいなければならない。

<i>ia, ja</i>	(グローバル) INTEGER。 それぞれ、部分行列 $A(ia:ia+m-1, ja:ja+n-1)$ の最初の行と最初の列を示すグローバル配列 <i>a</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。分散行列 <i>A</i> の配列ディスクリプタ。
<i>tau</i>	(ローカル) REAL ( <i>psorgql</i> の場合) DOUBLE PRECISION ( <i>pdorgql</i> の場合) 配列、次元は $LOCc(ja+n-1)$ 。 基本リフレクタ $H(j)$ のスカラ係数 $tau(j)$ を含む。 <i>tau</i> は、分散行列 <i>A</i> に関連付けられる。
<i>work</i>	(ローカル) REAL ( <i>psorgql</i> の場合) DOUBLE PRECISION ( <i>pdorgql</i> の場合) 次元 <i>lwork</i> のワークスペース配列。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 <i>work</i> の次元は $lwork \geq nb\_a * (nqa0 + mpa0 + nb\_a)$ でなければならない。ここで、

```

iroffa = mod(ia-1, mb_a)、
icoffa = mod(ja-1, nb_a)、
iarow = indxc2p(ia, mb_a, MYROW, rsrc_a, NPROW)、
iacol = indxc2p(ja, nb_a, MYCOL, csrc_a, NPCOL)、
mpa0 = numroc(m+iroffa, mb_a, MYROW, iarow, NPROW)、
nqa0 = numroc(n+icoffa, nb_a, MYCOL, iacol, NPCOL)

```

*indxc2p* および *numroc* は、ScaLAPACK ツール関数である。MYROW、MYCOL、NPROW、および NPCOL は、サブルーチン *blacs\_gridinfo* を呼び出して決定できる。

*lwork* = -1 の場合、*lwork* はグローバル入力であり、ワークスペースのクエリとみなされ、ルーチンはすべての *work* 配列の最小かつ最適なサイズだけを計算する。各値は該当する *work* 配列の最初のエン트리として返され、[pxerbla](#) はエラー・メッセージを生成しない。



### 出力パラメータ

<i>a</i>	因子分解の対象となる $m \times n$ の分散行列 $Q$ のローカル部分が格納される。
<i>work</i> (1)	終了時に、 <i>work</i> (1) には、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が格納される。
<i>info</i>	(グローバル) INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> < 0 の場合： <i>i</i> 番目の引数が配列で、 <i>j</i> 番目の値が不正だった場合、 <i>info</i> = -( <i>i</i> *100+ <i>j</i> )。 <i>i</i> 番目の引数がスカラーで値が不正だった場合、 <i>info</i> = - <i>i</i> 。

## p?ungql

p?geqlf で求めた  $QL$  因子分解のユニタリ行列  $Q$  を生成する。

### 構文

```
call pcungql( m, n, k, a, ia, ja, desca, tau, work, lwork, info )
call pzungql( m, n, k, a, ia, ja, desca, tau, work, lwork, info )
```

### 説明

このルーチンは、直交列を持つ  $m \times n$  の複素分散行列  $Q$  の全体または一部を生成する。これは、以下の次数  $m$  の  $k$  個の基本リフレクタの積による最初の  $n$  列として定義される。ここで、 $Q$  は  $A(ia:ia+m-1,ja:ja+n-1)$ 。

$$Q = H(k) \dots H(2) H(1)$$

[p?geqlf](#) によって返される。

### 入力パラメータ

<i>m</i>	(グローバル) INTEGER。 部分行列 $\text{sub}(Q)$ の行数 ( $m \geq 0$ )。
<i>n</i>	(グローバル) INTEGER。 部分行列 $\text{sub}(Q)$ の列数 ( $m \geq n \geq 0$ )。

<i>k</i>	(グローバル) INTEGER。 その積が行列 <i>Q</i> となる基本リフレクタの個数 ( $n \geq k \geq 0$ )。
<i>a</i>	(ローカル) COMPLEX (pcungql の場合) DOUBLE COMPLEX (pzungql の場合) ローカルメモリにある、ローカル次元 ( <i>lld_a</i> , <i>LOCc(ja+n-1)</i> ) の配列へのポインタ。呼び出し時に、 <i>j</i> 番目の列は、 <a href="#">p?geqlf</a> によって分散行列引数 <i>A(ia:*ja+n-k:ja+n-1)</i> の <i>k</i> 列に返される、基本リフレクタ <i>H(j) (ja+n-k ≤ j ≤ ja+n-1)</i> を定義するベクトルを含んでいなければならない。
<i>ia, ja</i>	(グローバル) INTEGER。 それぞれ、部分行列 <i>A(ia:ia+m-1, ja:ja+n-1)</i> の最初の行と最初の列を示すグローバル配列 <i>a</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。 分散行列 <i>A</i> の配列ディスクリプタ。
<i>tau</i>	(ローカル) COMPLEX (pcungql の場合) DOUBLE COMPLEX (pzungql の場合) 配列、次元は <i>LOCr(ia+n-1)</i> 。 基本リフレクタ <i>H(j)</i> のスカラ係数 <i>tau(j)</i> を含む。 <i>tau</i> は、分散行列 <i>A</i> に関連付けられる。
<i>work</i>	(ローカル) COMPLEX (pcungql の場合) DOUBLE COMPLEX (pzungql の場合) 次元 <i>lwork</i> のワークスペース配列。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 <i>work</i> の次元は $lwork \geq nb\_a * (nqa0 + mpa0 + nb\_a)$ でなければならない。ここで、  $iroffa = \text{mod}(ia-1, mb\_a)$ 、 $icoffa = \text{mod}(ja-1, nb\_a)$ 、 $iarow = \text{indxg2p}(ia, mb\_a, \text{MYROW}, rsrc\_a, \text{NPROW})$ 、 $iacol = \text{indxg2p}(ja, nb\_a, \text{MYCOL}, csrc\_a, \text{NPCOL})$ 、 $mpa0 = \text{numroc}(m+iroffa, mb\_a, \text{MYROW}, iarow, \text{NPROW})$ 、 $nqa0 = \text{numroc}(n+icoffa, nb\_a, \text{MYCOL}, iacol, \text{NPCOL})$

indxg2p および numroc は、ScaLAPACK ツール関数である。MYROW、MYCOL、NPROW、および NPCOL は、サブルーチン blacs\_gridinfo を呼び出して決定できる。

$lwork = -1$  の場合、 $lwork$  はグローバル入力であり、ワークスペースのクエリとみなされ、ルーチンはすべての  $work$  配列の最小かつ最適なサイズだけを計算する。各値は該当する  $work$  配列の最初のエン트리として返され、[pxerbla](#) はエラー・メッセージを生成しない。

### 出力パラメータ

$a$	因子分解の対象となる $m \times n$ の分散行列 $Q$ のローカル部分が格納される。
$work(1)$	終了時に、 $work(1)$ には、最適なパフォーマンスを得るために必要な $lwork$ の最小値が格納される。
$info$	(グローバル) INTEGER。 $info = 0$ の場合、実行は正常に終了したことを示す。 $info < 0$ の場合： $i$ 番目の引数が配列で、 $j$ 番目の値が不正だった場合、 $info = -(i*100+j)$ 。 $i$ 番目の引数がスカラで値が不正だった場合、 $info = -i$ 。

## p?ormql

一般行列に p?geqlf によって求めた  $QL$  因子分解のユニタリ行列  $Q$  を掛ける。

### 構文

```
call psormql( side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc,
              descc, work, lwork, info )
call pdormql( side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc,
              descc, work, lwork, info )
```

### 説明

このルーチンは、 $m \times n$  の一般実分散行列  $sub(C) = C(ic:ic+m-1, jc:jc+n-1)$  を以下により上書きする。

	$side = 'L'$	$side = 'R'$
$trans = 'N':$	$Q \text{ sub}(C)$	$\text{sub}(C) Q$
$trans = 'T':$	$Q^T \text{ sub}(C)$	$\text{sub}(C) Q^T$

ここで、 $Q$  は、 $k$  個の以下の基本リフレクタの積として定義される実直交分散行列である。

$$Q = H(k)' \dots H(2)' H(1)'$$

[p?geqlf](#) によって返される。 $Q$  の次数は、 $side = 'L'$  の場合は  $m$ 、 $side = 'R'$  の場合は  $n$  である。

## 入力パラメータ

$side$	(グローバル) CHARACTER = $'L'$ の場合、 $Q$ または $Q^T$ は左側から適用される。 = $'R'$ の場合、 $Q$ または $Q^T$ は右側から適用される。
$trans$	(グローバル) CHARACTER = $'N'$ の場合、 $Q$ が適用される (転置なし)。 = $'T'$ の場合、 $Q^T$ が適用される (転置)。
$m$	(グローバル) INTEGER。 分散行列 $\text{sub}(C)$ の行数 ( $m \geq 0$ )。
$n$	(グローバル) INTEGER。 分散行列 $\text{sub}(C)$ の列数 ( $n \geq 0$ )。
$k$	(グローバル) INTEGER。 その積が行列 $Q$ を定義する基本リフレクタの個数。次の制約がある。 $side = 'L'$ の場合、 $m \geq k \geq 0$ 、 $side = 'R'$ の場合、 $n \geq k \geq 0$ 。
$a$	(ローカル) REAL (psormql の場合) DOUBLE PRECISION (pdormql の場合)。 ローカルメモリにある、次元 ( $lld\_a$ , $LOCc(ja+k-1)$ ) の配列へのポインタ。 $j$ 番目の列は、 <a href="#">p?geqlf</a> によって分散行列引数 $a(ia:*,ja:ja+k-1)$ の $k$ 列に返される、基本リフレクタ $H(j)$ ( $ja \leq j \leq ja+k-1$ ) を定義するベクトルを含んでいなければならない。 $a(ia:*,ja:ja+k-1)$ はルーチンにより変更されるが終了時に復元される。 $side = 'L'$ の場合、 $lld\_a \geq \max(1, LOCr(ia+m-1))$

	$side = 'R'$ の場合、 $lld\_a \geq \max(1, LOCr(ia+n-1))$
$ia, ja$	(グローバル) INTEGER。 それぞれ、部分行列 $A$ の最初の行と最初の列を示す、グローバル配列 $a$ の行インデックスと列インデックス。
$desca$	(グローバルおよびローカル) INTEGER。配列、次元は $(dlen\_)$ 。 分散行列 $A$ の配列ディスクリプタ。
$tau$	(ローカル) REAL (psormql の場合) DOUBLE PRECISION (pdormql の場合)。配列、次元は $LOCc(ja+n-1)$ 。 <a href="#">p?geqlf</a> によって返される、基本リフレクタ $H(j)$ のスカラー係数 $tau(j)$ を含む。 $tau$ は、分散行列 $A$ に関連付けられる。
$c$	(ローカル) REAL (psormql の場合) DOUBLE PRECISION (pdormql の場合)。 ローカルメモリにある、ローカル次元 $(lld\_c, LOCc(jc+n-1))$ の配列へのポインタ。  因子分解する分散行列 $sub(C)$ のローカル部分を含む。
$ic, jc$	(グローバル) INTEGER。 それぞれ、部分行列 $C$ の最初の行と最初の列を示す、グローバル配列 $c$ の行インデックスと列インデックス。
$descc$	(グローバルおよびローカル) INTEGER。配列、次元は $(dlen\_)$ 。 分散行列 $C$ の配列ディスクリプタ。
$work$	(ローカル) REAL (psormql の場合) DOUBLE PRECISION (pdormql の場合) 次元 $lwork$ のワークスペース配列。
$lwork$	(ローカルまたはグローバル) INTEGER。 $work$ の次元は  $side = 'L'$ の場合、 $lwork \geq \max((nb\_a*(nb\_a-1))/2, (nqc0 + mpc0)*nb\_a + nb\_a*nb\_a$  $side = 'R'$ の場合、 $lwork \geq \max((nb\_a*(nb\_a-1))/2, (nqc0 + \max npa0) + numroc(numroc(n + icoffc, nb\_a, 0, 0, NPCOL), nb\_a, 0, 0, lcmq), mpc0)) * nb\_a + nb\_a*nb\_a$

でなければならない。

ここで、

```
lcmp = lcm / NPCOL で lcm = ilcm (NPROW, NPCOL)、
iroffa = mod(ia-1, mb_a)、
icoffa = mod(ja-1, nb_a)、
iarow = indxg2p (ia, mb_a, MYROW, rsrc_a, NPROW)、
npa0 = numroc(n + iroffa, mb_a, MYROW, iarow, NPROW)、
iroffc = mod(ic-1, mb_c)、
icoffc = mod(jc-1, nb_c)、
icrow = indxg2p(ic, mb_c, MYROW, rsrc_c, NPROW)、
iccol = indxg2p(jc, nb_c, MYCOL, csrc_c, NPCOL)、
mpc0 = numroc(m+iroffc, mb_c, MYROW, icrow, NPROW)、
nqc0 = numroc(n+icoffc, nb_c, MYCOL, iccol, NPCOL)
```

ilcm、indxg2p および numroc は、ScaLAPACK ツール関数である。  
MYROW、MYCOL、NPROW および NPCOL は、サブルーチン  
blacs\_gridinfo を呼び出して決定できる。

lwork = -1 の場合、lwork はグローバル入力であり、ワークスペース  
のクエリとみなされ、ルーチンはすべての work 配列の最小かつ最適  
なサイズだけを計算する。各値は該当する work 配列の最初のエン  
トリとして返され、[pxerbla](#) はエラー・メッセージを生成しない。

### 出力パラメータ

<i>c</i>	積 $Q * \text{sub}(C)$ 、 $Q' \text{sub}(C)$ 、 $\text{sub}(C) * Q'$ 、または $\text{sub}(C) * Q$ のいずれかに よって上書きされる。
<i>work(1)</i>	終了時に、 <i>work(1)</i> には、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が格納される。
<i>info</i>	(グローバル) INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> < 0 の場合： <i>i</i> 番目の引数が配列で、 <i>j</i> 番目の値が不正だった場合、 <i>info</i> = -( <i>i</i> *100+ <i>j</i> )。 <i>i</i> 番目の引数がスカラで値が不正だった場合、 <i>info</i> = - <i>i</i> 。

## p?unmql

一般行列に `p?geqlf` によって求めた  $QL$  因子分解のユニタリ行列  $Q$  を掛ける。

### 構文

```
call pcunmql( side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc,
              descc, work, lwork, info )
call pzunmql( side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc,
              descc, work, lwork, info )
```

### 説明

このルーチンは、 $m \times n$  の一般複素分散行列  $\text{sub}(C) = C(\text{ic}:\text{ic}+m-1, \text{jc}:\text{jc}+n-1)$  を以下により上書きする。

	$\text{side} = 'L'$	$\text{side} = 'R'$
$\text{trans} = 'N':$	$Q \text{sub}(C)$	$\text{sub}(C) Q$
$\text{trans} = 'C':$	$Q^H \text{sub}(C)$	$\text{sub}(C) Q^H$

ここで、 $Q$  は、 $k$  個の以下の基本リフレクタの積として定義される複素ユニタリ分散行列である。

$$Q = H(k)' \dots H(2)' H(1)'$$

[p?geqlf](#) によって返される。 $Q$  の次数は、 $\text{side} = 'L'$  の場合は  $m$ 、 $\text{side} = 'R'$  の場合は  $n$  である。

### 入力パラメータ

**side** (グローバル) CHARACTER  
 $= 'L'$  の場合、 $Q$  または  $Q^H$  は左側から適用される。  
 $= 'R'$  の場合、 $Q$  または  $Q^H$  は右側から適用される。

**trans** (グローバル) CHARACTER  
 $= 'N'$  の場合、 $Q$  が適用される ( 転置なし )。  
 $= 'C'$  の場合、 $Q^H$  が適用される ( 共役転置 )。

**m** (グローバル) INTEGER。  
 分散行列  $\text{sub}(C)$  の行数 ( $m \geq 0$ )。

<i>n</i>	(グローバル) INTEGER。 分散行列 <i>sub(C)</i> の列数 ( $n \geq 0$ )。
<i>k</i>	(グローバル) INTEGER。 その積が行列 <i>Q</i> を定義する基本リフレクタの個数。次の制約がある。 <i>side</i> = 'L' の場合、 $m \geq k \geq 0$ 、 <i>side</i> = 'R' の場合、 $n \geq k \geq 0$ 。
<i>a</i>	(ローカル) COMPLEX (pcunmq1 の場合) DOUBLE COMPLEX (pzunmq1 の場合)。 ローカルメモリにある、次元 ( <i>lld_a</i> , <i>LOCc(ja+k-1)</i> ) の配列へのポインタ。 <i>j</i> 番目の列は、 <a href="#">p?geqlf</a> によって分散行列引数 <i>a(ia:*,ja:ja+k-1)</i> の <i>k</i> 列に返される、基本リフレクタ <i>H(j)</i> ( $ja \leq j \leq ja+k-1$ ) を定義するベクトルを含んでいなければならない。 <i>a(ia:*,ja:ja+k-1)</i> はルーチンにより変更されるが終了時に復元される。  <i>side</i> = 'L' の場合、 $lld\_a \geq \max(1, LOCr(ia+m-1))$ <i>side</i> = 'R' の場合、 $lld\_a \geq \max(1, LOCr(ia+n-1))$
<i>ia, ja</i>	(グローバル) INTEGER。 それぞれ、部分行列 <i>A</i> の最初の行と最初の列を示す、グローバル配列 <i>a</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。 分散行列 <i>A</i> の配列ディスクリプタ。
<i>tau</i>	(ローカル) COMPLEX (pcunmq1 の場合) DOUBLE COMPLEX (pzunmq1 の場合) 配列、次元は <i>LOCr(ia+n-1)</i> 。 <a href="#">p?geqlf</a> によって返される、基本リフレクタ <i>H(j)</i> のスカラ係数 <i>tau(j)</i> を含む。 <i>tau</i> は、分散行列 <i>A</i> に関連付けられる。
<i>c</i>	(ローカル) COMPLEX (pcunmq1 の場合) DOUBLE COMPLEX (pzunmq1 の場合)。 ローカルメモリにある、ローカル次元 ( <i>lld_c</i> , <i>LOCc(jc+n-1)</i> ) の配列へのポインタ。  因子分解する分散行列 <i>sub(C)</i> のローカル部分を含む。



<i>ic, jc</i>	(グローバル) INTEGER。 それぞれ、部分行列 <i>C</i> の最初の行と最初の列を示す、グローバル配列 <i>c</i> の行インデックスと列インデックス。
<i>desc</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen</i> )。 分散行列 <i>C</i> の配列ディスクリプタ。
<i>work</i>	(ローカル) COMPLEX ( <i>pcunmq1</i> の場合) DOUBLE COMPLEX ( <i>pzunmq1</i> の場合)。次元 <i>lwork</i> のワークスペース配列。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 <i>work</i> の次元は  $side = 'L'$ の場合、 $lwork \geq \max ((nb\_a * (nb\_a - 1)) / 2, (nqc0 + mpc0) * nb\_a + nb\_a * nb\_a)$  $side = 'R'$ の場合、 $lwork \geq \max ((nb\_a * (nb\_a - 1)) / 2, (nqc0 + \max npa0) + \text{numroc}(\text{numroc}(n + icoffc, nb\_a, 0, 0, NPCOL), nb\_a, 0, 0, lcmq, mpc0)) * nb\_a + nb\_a * nb\_a)$  でなければならない。 ここで、  $lcmp = lcm / NPCOL$ で $lcm = ilcm(NPROW, NPCOL)$ 、 $iroffa = \text{mod}(ia - 1, mb\_a)$ 、 $icoffa = \text{mod}(ja - 1, nb\_a)$ 、 $iarow = \text{indxg2p}(ia, mb\_a, MYROW, rsrc\_a, NPROW)$ 、 $npa0 = \text{numroc}(n + iroffa, mb\_a, MYROW, iarow, NPROW)$ 、 $iroffc = \text{mod}(ic - 1, mb\_c)$ 、 $icoffc = \text{mod}(jc - 1, nb\_c)$ 、 $icrow = \text{indxg2p}(ic, mb\_c, MYROW, rsrc\_c, NPROW)$ 、 $iccol = \text{indxg2p}(jc, nb\_c, MYCOL, csrc\_c, NPCOL)$ 、 $mpc0 = \text{numroc}(m + iroffc, mb\_c, MYROW, icrow, NPROW)$ 、 $nqc0 = \text{numroc}(n + icoffc, nb\_c, MYCOL, iccol, NPCOL)$

ilcm、indxg2p および numroc は、ScaLAPACK ツール関数である。  
MYROW、MYCOL、NPROW、および NPCOL は、サブルーチン  
blacs\_gridinfo を呼び出して決定できる。

lwork=-1 の場合、lwork はグローバル入力であり、ワークスペース  
のクエリとみなされ、ルーチンはすべての work 配列の最小かつ最適  
なサイズだけを計算する。各値は該当する work 配列の最初のエン  
トリとして返され、[pxerbla](#) はエラー・メッセージを生成しない。

### 出力パラメータ

<i>c</i>	積 $Q^* \text{sub}(C)$ 、 $Q' \text{sub}(C)$ 、 $\text{sub}(C)^* Q'$ 、または $\text{sub}(C)^* Q$ のいずれかに よって上書きされる。
<i>work(1)</i>	終了時に、 <i>work(1)</i> には、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が格納される。
<i>info</i>	(グローバル) INTEGER。 <i>info</i> =0 の場合、実行は正常に終了したことを示す。 <i>info</i> <0 の場合： <i>i</i> 番目の引数が配列で、 <i>j</i> 番目の値が不正だった場合、 <i>info</i> = -( <i>i</i> *100+ <i>j</i> )。 <i>i</i> 番目の引数がスカラで値が不正だった場合、 <i>info</i> = - <i>i</i> 。

---

## p?gerqf

一般矩形行列の *RQ* 因子分解を行う。

---

### 構文

```
call psggerqf( m, n, a, ia, ja, desca, tau, work, lwork, info )
call pdggerqf( m, n, a, ia, ja, desca, tau, work, lwork, info )
call pcggerqf( m, n, a, ia, ja, desca, tau, work, lwork, info )
call pzggerqf( m, n, a, ia, ja, desca, tau, work, lwork, info )
```

### 説明

このルーチンは、次の式に従って、 $m \times n$  の一般分散行列  
 $\text{sub}(A) = A(ia:ia+m-1, ja:ja+n-1)$  の *QR* 因子分解を行う。

$$A = R Q$$

## 入力パラメータ

<i>m</i>	(グローバル) INTEGER。 分散部分行列 <i>sub(A)</i> の行数 ( $m \geq 0$ )。
<i>n</i>	(グローバル) INTEGER。 分散部分行列 <i>sub(A)</i> の列数 ( $n \geq 0$ )。
<i>a</i>	(ローカル) REAL (psgeqrf の場合) DOUBLE PRECISION (pdgeqrf の場合) COMPLEX (pcgeqrf の場合) DOUBLE COMPLEX (pzgeqrf の場合)。 ローカルメモリにある、ローカル次元 ( <i>lld_a</i> , <i>LOCc(ja+n-1)</i> ) の配列へのポインタ。 因子分解する分散行列 <i>sub(A)</i> のローカル部分を含む。
<i>ia, ja</i>	(グローバル) INTEGER。 それぞれ、部分行列 <i>A(ia:ia+m-1, ja:ja+n-1)</i> の最初の行と最初の列を示すグローバル配列 <i>a</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。 分散行列 <i>A</i> の配列ディスクリプタ。
<i>work</i>	(ローカル)。 REAL (psgeqrf の場合) DOUBLE PRECISION (pdgeqrf の場合) COMPLEX (pcgeqrf の場合) DOUBLE COMPLEX (pzgeqrf の場合) 次元 <i>lwork</i> のワークスペース配列。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 <i>work</i> の次元は $lwork \geq mb\_a * (mp0 + nq0 + mb\_a)$ でなければならない。 ここで、 $iroff = \text{mod}(ia-1, mb\_a)$ 、 $icoff = \text{mod}(ja-1, nb\_a)$ 、 $iarow = \text{indxg2p}(ia, mb\_a, MYROW, rsrc\_a, NPROW)$ 、 $iacol = \text{indxg2p}(ja, nb\_a, MYCOL, csrc\_a, NPCOL)$ 、 $mp0 = \text{numroc}(m+iroff, mb\_a, MYROW, iarow, NPROW)$ 、

`ng0 = numroc (n+icoff, nb_a, MYCOL, iacol, NPCOL)`

`numroc` および `indxg2p` は ScaLAPACK ツール関数である。MYROW、MYCOL、NPROW および NPCOL は、サブルーチン `blacs_gridinfo` を呼び出して決定できる。

`lwork = -1` の場合、`lwork` はグローバル入力であり、ワークスペースのクエリとみなされ、ルーチンはすべての `work` 配列の最小かつ最適なサイズだけを計算する。各値は該当する `work` 配列の最初のエン트리として返され、[pxerbla](#) はエラー・メッセージを生成しない。

### 出力パラメータ

<code>a</code>	終了時に、 $m \leq n$ の場合、分散部分行列 $A(ia:ia+m-1, ja:ja+n-1)$ の上三角部分に、 $m \times m$ の上三角行列 $R$ が格納される。 $m \geq n$ の場合、 $(m-n)$ 番目の劣対角成分とその上の成分に、 $m \times n$ 上台形行列 $R$ が格納される。残りの成分は、配列 <code>tau</code> とともに、基本リフレクタの積として直交/ユニタリ行列 $Q$ を表す。(次の「アプリケーション・ノート」を参照)。
<code>tau</code>	(ローカル) REAL (psgeqrf の場合) DOUBLE PRECISION (pdgeqrf の場合) COMPLEX (pcgeqrf の場合) DOUBLE COMPLEX (pzgeqrf の場合)。 配列、次元は $LOC_r(ia+m-1)$ 。 基本リフレクタのスカラー係数 <code>tau</code> が格納される。 <code>tau</code> は、分散行列 $A$ に関連付けられる。
<code>work(1)</code>	終了時に、 <code>work(1)</code> には、最適なパフォーマンスを得るために必要な <code>lwork</code> の最小値が格納される。
<code>info</code>	(グローバル) INTEGER。 <code>info = 0</code> の場合、実行は正常に終了したことを示す。 <code>info &lt; 0</code> の場合: $i$ 番目の引数が配列で、 $j$ 番目の値が不正だった場合、 <code>info = -(i*100+j)</code> 。 $i$ 番目の引数がスカラーで値が不正だった場合、 <code>info = -i</code> 。

### アプリケーション・ノート

行列  $Q$  は、以下の基本リフレクタの積として表現される。

$$Q = H(ia) H(ia+1) \dots H(ia+k-1)$$

ここで、 $k = \min(m, n)$  である。

それぞれの  $H(i)$  は次のような形式を持つ。

$$H(i) = I - \tau v v^*$$

ここで、 $\tau$  は実 / 複素スカラ、 $v$  は実 / 複素ベクトルで、 $v(n-k+i+1:n) = 0$  および  $v(n-k+i) = 1$ 。終了時に、 $v(1:n-k+i-1)/\text{conjg}(v(1:n-k+i-1))$  は  $A(ia+m-k+i-1, ja:ja+n-k+i-2)$  に、 $\tau$  は  $\tau(ia+m-k+i-1)$  に格納される。

## p?orgrq

p?gerqf で求めた  $RQ$  因子分解の実直交行列  $Q$  を生成する。

### 構文

```
call psorgrq( m, n, k, a, ia, ja, desca, tau, work, lwork, info )
call pdorgrq( m, n, k, a, ia, ja, desca, tau, work, lwork, info )
```

### 説明

このルーチンは、直交列を持つ  $m \times n$  の実分散行列  $Q$  の全体または一部を生成する。これは、以下の次数  $m$  の  $k$  個の基本リフレクタの積の最後の  $m$  列として定義される。ここで、 $Q$  は  $A(ia:ia+m-1, ja:ja+n-1)$ 。

$$Q = H(1) H(2) \dots H(k)$$

[p?gerqf](#) によって返される。

### 入力パラメータ

$m$	(グローバル) INTEGER。 部分行列 $\text{sub}(Q)$ の行数 ( $m \geq 0$ )。
$n$	(グローバル) INTEGER。 部分行列 $\text{sub}(Q)$ の列数 ( $n \geq m \geq 0$ )。
$k$	(グローバル) INTEGER。 その積が行列 $Q$ となる基本リフレクタの個数 ( $m \geq k \geq 0$ )。
$a$	(ローカル) REAL (psorgrq の場合) DOUBLE PRECISION (pdorgrq の場合)

ローカルメモリにある、ローカル次元 ( $lld\_a$ ,  $LOCc(ja+n-1)$ ) の配列へのポインタ。  $i$  番目の列は、 $p?geqrf$  によって分散行列引数  $a(ia:*,ja:ja+k-1)$  の  $k$  列に返される、基本リフレクタ  $H(j)$  ( $ja \leq j \leq ja+k-1$ ) を定義するベクトルを含んでいなければならない。

$ia, ja$  (グローバル) INTEGER。  
それぞれ、部分行列  $A(ia:ia+m-1, ja:ja+n-1)$  の最初の行と最初の列を示すグローバル配列  $a$  の行インデックスと列インデックス。

$desca$  (グローバルおよびローカル) INTEGER。配列、次元は ( $dlen\_$ )。分散行列  $A$  の配列ディスクリプタ。

$tau$  (ローカル)  
REAL ( $psorgqrq$  の場合)  
DOUBLE PRECISION ( $pdorgqrq$  の場合)  
配列、次元は  $LOCc(ja+k-1)$ 。  
[p?gerqf](#) によって返される、基本リフレクタ  $H(i)$  のスカラー係数  $tau(i)$  を含む。  $tau$  は、分散行列  $A$  に関連付けられる。

$work$  (ローカル)  
REAL ( $psorgqrq$  の場合)  
DOUBLE PRECISION ( $pdorgqrq$  の場合)  
次元  $lwork$  のワークスペース配列。

$lwork$  (ローカルまたはグローバル) INTEGER。  
 $work$  の次元は  $lwork \geq mb\_a * (mpa0 + nqa0 + mb\_a)$  でなければならない。ここで、

```
iroffa = mod(ia-1, mb_a),
icoffa = mod(ja-1, nb_a),
iarow = indxc2p(ia, mb_a, MYROW, rsrc_a, NPROW),
iacol = indxc2p(ja, nb_a, MYCOL, csrc_a, NPCOL),
mpa0 = numroc(m+iroffa, mb_a, MYROW, iarow, NPROW),
nqa0 = numroc(n+icoffa, nb_a, MYCOL, iacol, NPCOL)
```

$indxc2p$  および  $numroc$  は、ScaLAPACK ツール関数である。MYROW、MYCOL、NPROW、および NPCOL は、サブルーチン `blacs_gridinfo` を呼び出して決定できる。

$lwork = -1$  の場合、 $lwork$  はグローバル入力であり、ワークスペースのクエリとみなされ、ルーチンはすべての  $work$  配列の最小かつ最適なサイズだけを計算する。各値は該当する  $work$  配列の最初のエントリとして返され、[pxerbla](#) はエラー・メッセージを生成しない。

### 出力パラメータ

$a$	$m \times n$ の分散行列 $Q$ のローカル部分が格納される。
$work(1)$	終了時に、 $work(1)$ には、最適なパフォーマンスを得るために必要な $lwork$ の最小値が格納される。
$info$	(グローバル) INTEGER。 $info = 0$ の場合、実行は正常に終了したことを示す。 $info < 0$ の場合： $i$ 番目の引数が配列で、 $j$ 番目の値が不正だった場合、 $info = -(i*100+j)$ 。 $i$ 番目の引数がスカラで値が不正だった場合、 $info = -i$ 。

## p?ungrq

p?gerqf で求めた  $RQ$  因子分解のユニタリ行列  $Q$  を生成する。

### 構文

```
call pcungrq( m, n, k, a, ia, ja, desca, tau, work, lwork, info )
call pzungrq( m, n, k, a, ia, ja, desca, tau, work, lwork, info )
```

### 説明

このルーチンは、直交列を持つ  $m \times n$  の複素分散行列  $Q$  の全体または一部を生成する。これは、以下の次数  $n$  の  $k$  個の基本リフレクタの積の最後の  $m$  列として定義される。ここで、 $Q$  は  $A(ia:ia+m-1, ja:ja+n-1)$ 。

$$Q = H(1)' H(2)' \dots H(k)'$$

[p?gerqf](#) によって返される。

## 入力パラメータ

<i>m</i>	(グローバル) INTEGER。 部分行列 $\text{sub}(Q)$ の行数 ( $m \geq 0$ )。
<i>n</i>	(グローバル) INTEGER。 部分行列 $\text{sub}(Q)$ の列数 ( $n \geq m \geq 0$ )。
<i>k</i>	(グローバル) INTEGER。 その積が行列 $Q$ となる基本リフレクタの個数 ( $m \geq k \geq 0$ )。
<i>a</i>	(ローカル) COMPLEX (pcungrq の場合) DOUBLE COMPLEX (pzungrq の場合) ローカルメモリにある、次元 ( $lld\_a$ , $LOCc(ja+n-1)$ ) の配列へのポインタ。 <i>i</i> 番目の行は、p?gerqf によって分散行列引数 $a(ia+m-k:ia+m-1, ja:*)$ の <i>k</i> 行に返される、基本リフレクタ $H(i)(ia+m-k \leq i \leq ia+m-1)$ を定義するベクトルを含んでいなければならない。
<i>ia, ja</i>	(グローバル) INTEGER。 それぞれ、部分行列 $A$ の最初の行と最初の列を示す、グローバル配列 $a$ の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( $dlen\_$ )。 分散行列 $A$ の配列ディスクリプタ。
<i>tau</i>	(ローカル) COMPLEX (pcungrq の場合) DOUBLE COMPLEX (pzungrq の場合) 配列、次元は $LOCr(ia+m-1)$ 。 <a href="#">p?gerqf</a> によって返される、基本リフレクタ $H(i)$ のスカラ係数 $\tau(i)$ を含む。 $\tau$ は、分散行列 $A$ に関連付けられる。
<i>work</i>	(ローカル) COMPLEX (pcungrq の場合) DOUBLE COMPLEX (pzungrq の場合) 次元 <i>lwork</i> のワークスペース配列。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 <i>work</i> の次元は $lwork \geq mb\_a * (mpa0 + nqa0 + mb\_a)$ でなければならない。ここで、 $iroffa = \text{mod}(ia-1, mb\_a)$ 、 $icoffa = \text{mod}(ja-1, nb\_a)$ 、



```

iarow = indxg2p(ia, mb_a, MYROW, rsrc_a, NPROW),
iacol = indxg2p(ja, nb_a, MYCOL, csrc_a, NPCOL),
mpa0 = numroc(m+iroffa, mb_a, MYROW, iarow, NPROW),
nqa0 = numroc(n+icoffa, nb_a, MYCOL, iacol, NPCOL)

```

indxg2p および numroc は、ScaLAPACK ツール関数である。MYROW、MYCOL、NPROW、および NPCOL は、サブルーチン blacs\_gridinfo を呼び出して決定できる。

$lwork = -1$  の場合、 $lwork$  はグローバル入力であり、ワークスペースのクエリとみなされ、ルーチンはすべての  $work$  配列の最小かつ最適なサイズだけを計算する。各値は該当する  $work$  配列の最初のエン트리として返され、[pxerbla](#) はエラー・メッセージを生成しない。

### 出力パラメータ

$a$	$m \times n$ の分散行列 $Q$ のローカル部分が格納される。
$work(1)$	終了時に、 $work(1)$ には、最適なパフォーマンスを得るために必要な $lwork$ の最小値が格納される。
$info$	(グローバル) INTEGER。 $info = 0$ の場合、実行は正常に終了したことを示す。 $info < 0$ の場合： $i$ 番目の引数が配列で、 $j$ 番目の値が不正だった場合、 $info = -(i*100+j)$ 。 $i$ 番目の引数がスカラで値が不正だった場合、 $info = -i$ 。

## p?ormrq

一般行列に p?gerqf によって求めた  $RQ$  因子分解の直交行列  $Q$  を掛ける。

### 構文

```

call psormrq( side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc,
              descc, work, lwork, info )
call pdormrq( side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc,
              descc, work, lwork, info )

```

## 説明

このルーチンは、 $m \times n$  の一般実分散行列  $\text{sub}(C) = C(ic:ic+m-1, jc:jc+n-1)$  を以下により上書きする。

	$side = 'L'$	$side = 'R'$
$trans = 'N':$	$Q \text{ sub}(C)$	$\text{sub}(C) Q$
$trans = 'T':$	$Q^T \text{ sub}(C)$	$\text{sub}(C) Q^T$

ここで、 $Q$  は、 $k$  個の以下の基本リフレクタの積として定義される実直交分散行列である。

$$Q = H(1) H(2) \dots H(k)$$

[p?gerqf](#) によって返される。 $Q$  の次数は、 $side = 'L'$  の場合は  $m$ 、 $side = 'R'$  の場合は  $n$  である。

## 入力パラメータ

**side** (グローバル) CHARACTER  
 $= 'L'$  の場合、 $Q$  または  $Q^T$  は左側から適用される。  
 $= 'R'$  の場合、 $Q$  または  $Q^T$  は右側から適用される。

**trans** (グローバル) CHARACTER  
 $= 'N'$  の場合、 $Q$  が適用される (転置なし)。  
 $= 'T'$  の場合、 $Q^T$  が適用される (転置)。

**m** (グローバル) INTEGER。  
分散行列  $\text{sub}(C)$  の行数 ( $m \geq 0$ )。

**n** (グローバル) INTEGER。  
分散行列  $\text{sub}(C)$  の列数 ( $n \geq 0$ )。

**k** (グローバル) INTEGER。  
その積が行列  $Q$  を定義する基本リフレクタの個数。次の制約がある。  
 $side = 'L'$  の場合、 $m \geq k \geq 0$ 、  
 $side = 'R'$  の場合、 $n \geq k \geq 0$ 。

**a** (ローカル)  
REAL (psormqr の場合)  
DOUBLE PRECISION (pdormqr の場合)。  
ローカルメモリにある、次元 ( $lld\_a$ ,  $LOCc(ja+m-1)$ ) ( $side = 'L'$  の場合) または ( $lld\_a$ ,  $LOCc(ja+n-1)$ ) ( $side = 'R'$  の場合) の配列へのポインタ。 $i$  番目の行は、[p?gelqf](#) によって分散行列引数  $a(ia:ia+k-1, ja:*)$  の  $k$  行に返される、基本リフレクタ

	<p><math>H(i)</math> (<math>ia &lt; i &lt; ia+k-1</math>) を定義するベクトルを含んでいなければならない。</p> <p><math>a(ia:ia+k-1, ja:*)</math> はルーチンにより変更されるが終了時に復元される。</p>
<i>ia, ja</i>	<p>(グローバル) INTEGER。</p> <p>それぞれ、部分行列 <math>A</math> の最初の行と最初の列を示す、グローバル配列 <math>a</math> の行インデックスと列インデックス。</p>
<i>desca</i>	<p>(グローバルおよびローカル) INTEGER。配列、次元は (<math>dlen\_</math>)。</p> <p>分散行列 <math>A</math> の配列ディスクリプタ。</p>
<i>tau</i>	<p>(ローカル)</p> <p>REAL (psormqr の場合)</p> <p>DOUBLE PRECISION (pdormqr の場合)</p> <p>配列、次元は <math>LOC_c(ja+k-1)</math>。</p> <p><a href="#">p?gerqf</a> によって返される、基本リフレクタ <math>H(i)</math> のスカラ係数 <math>\tau(i)</math> を含む。tau は、分散行列 <math>A</math> に関連付けられる。</p>
<i>c</i>	<p>(ローカル)</p> <p>REAL (psormrq の場合)</p> <p>DOUBLE PRECISION (pdormrq の場合)</p> <p>ローカルメモリにある、ローカル次元 (<math>lld\_c</math>, <math>LOC_c(jc+n-1)</math>) の配列へのポインタ。</p> <p>因子分解する分散行列 <math>\text{sub}(C)</math> のローカル部分を含む。</p>
<i>ic, jc</i>	<p>(グローバル) INTEGER。</p> <p>それぞれ、部分行列 <math>C</math> の最初の行と最初の列を示す、グローバル配列 <math>c</math> の行インデックスと列インデックス。</p>
<i>descc</i>	<p>(グローバルおよびローカル) INTEGER。配列、次元は (<math>dlen\_</math>)。</p> <p>分散行列 <math>C</math> の配列ディスクリプタ。</p>
<i>work</i>	<p>(ローカル)</p> <p>REAL (psormrq の場合)</p> <p>DOUBLE PRECISION (pdormrq の場合)。次元 <math>lwork</math> のワークスペース配列。</p>
<i>lwork</i>	<p>(ローカルまたはグローバル) INTEGER。</p> <p><math>work</math> の次元は</p> <p><math>side = 'L'</math> の場合、</p> <p><math>lwork \geq \max ((mb\_a * (mb\_a - 1)) / 2, (mpc0 + \max (mqa0 + \text{numroc}(\text{numroc}(n + iroffc, mb\_a, 0, 0, NPROW), mb\_a, 0, 0, lcmp), nqc0)) * mb\_a) + mb\_a * mb\_a</math></p>

*side* = 'R' の場合、  
 $lwork \geq \max ((mb\_a * (mb\_a - 1)) / 2, (mpc0 + nqc0) * mb\_a) + mb\_a * mb\_a$

でなければならない。

ここで、

```
lcmp = lcm / NPROW で lcm = ilcm (NPROW, NPCOL)、
iroffa = mod(ia-1, mb_a)、
icoffa = mod(ja-1, nb_a)、
iacol = indxg2p (ja, nb_a, MYCOL, csrc_a, NPCOL)、
mqa0 = numroc(n+icoffa, nb_a, MYCOL, iacol, NPCOL)、
iroffc = mod(ic-1, mb_c)、
icoffc = mod(jc-1, nb_c)、
icrow = indxg2p(ic, mb_c, MYROW, rsrc_c, NPROW)、
iccol = indxg2p(jc, nb_c, MYCOL, csrc_c, NPCOL)、
mpc0 = numroc(m+iroffc, mb_c, MYROW, icrow, NPROW)、
nqc0 = numroc(n+icoffc, nb_c, MYCOL, iccol, NPCOL)
```

ilcm、indxg2p および numroc は、ScaLAPACK ツール関数である。  
 MYROW、MYCOL、NPROW および NPCOL は、サブルーチン  
 blacs\_gridinfo を呼び出して決定できる。

*lwork* = -1 の場合、*lwork* はグローバル入力であり、ワークスペース  
 のクエリとみなされ、ルーチンはすべての *work* 配列の最小かつ最適  
 なサイズだけを計算する。各値は該当する *work* 配列の最初のエン  
 トリとして返され、[pxerbla](#) はエラー・メッセージを生成しない。

### 出力パラメータ

<i>c</i>	積 $Q * \text{sub}(C)$ 、 $Q' \text{sub}(C)$ 、 $\text{sub}(C) * Q'$ 、または $\text{sub}(C) * Q$ のいずれかに よって上書きされる。
<i>work</i> (1)	終了時に、 <i>work</i> (1) には、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が格納される。
<i>info</i>	(グローバル) INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> < 0 の場合：

$i$  番目の引数が配列で、 $j$  番目の値が不正だった場合、  
 $info = -(i*100+j)$ 。 $i$  番目の引数がスカラーで値が不正だった場合、  
 $info = -i$ 。

## p?unmrq

一般行列に `p?gerqf` によって求めた  $RQ$  因子  
 分解のユニタリ行列  $Q$  を掛ける。

### 構文

```
call pcunmrq( side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc,
              descc, work, lwork, info )
call pzunmrq( side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc,
              descc, work, lwork, info )
```

### 説明

このルーチンは、 $m \times n$  の一般複素分散行列  $\text{sub}(C) = C(ic:ic+m-1, jc:jc+n-1)$  を以下により上書きする。

	$side = 'L'$	$side = 'R'$
$trans = 'N':$	$Q \text{ sub}(C)$	$\text{sub}(C) Q$
$trans = 'C':$	$Q^H \text{ sub}(C)$	$\text{sub}(C) Q^H$

ここで、 $Q$  は、 $k$  個の以下の基本リフレクタの積として定義される複素ユニタリ分散行列である。

$$Q = H(1)' H(2)' \dots H(k)'$$

[p?gerqf](#) によって返される。 $Q$  の次数は、 $side = 'L'$  の場合は  $m$ 、 $side = 'R'$  の場合は  $n$  である。

### 入力パラメータ

$side$  (グローバル) CHARACTER  
 $= 'L'$  の場合、 $Q$  または  $Q^H$  は左側から適用される。  
 $= 'R'$  の場合、 $Q$  または  $Q^H$  は右側から適用される。

<i>trans</i>	(グローバル) CHARACTER ='N' の場合、 $Q$ が適用される (転置なし)。 ='C' の場合、 $Q^H$ が適用される (共役転置)。
<i>m</i>	(グローバル) INTEGER。 分散行列 $\text{sub}(C)$ の行数 ( $m \geq 0$ )。
<i>n</i>	(グローバル) INTEGER。 分散行列 $\text{sub}(C)$ の列数 ( $n \geq 0$ )。
<i>k</i>	(グローバル) INTEGER。 その積が行列 $Q$ を定義する基本リフレクタの個数。次の制約がある。 <i>side</i> = 'L' の場合、 $m \geq k \geq 0$ 、 <i>side</i> = 'R' の場合、 $n \geq k \geq 0$ 。
<i>a</i>	(ローカル) COMPLEX (pcunmrq の場合) DOUBLE COMPLEX (pzunmrq の場合)。 ローカルメモリにある、次元 ( $\text{lld\_a}$ , $\text{LOCc}(ja+m-1)$ ) ( <i>side</i> = 'L' の場合) または ( $\text{lld\_a}$ , $\text{LOCc}(ja+n-1)$ ) ( <i>side</i> = 'R' の場合) の配列へのポインタ。 <i>i</i> 番目の行は、 <a href="#">p?gerqf</a> によって分散行列引数 $a(ia:ia+k-1, ja:*)$ の <i>k</i> 行に返される、基本リフレクタ $H(i)$ ( $ia \leq i \leq ia+k-1$ ) を定義するベクトルを含んでいなければならない。 $a(ia:ia+k-1, ja:*)$ はルーチンにより変更されるが終了時に復元される。
<i>ia, ja</i>	(グローバル) INTEGER。 それぞれ、部分行列 $A$ の最初の行と最初の列を示す、グローバル配列 $a$ の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( $dlen\_$ )。 分散行列 $A$ の配列ディスクリプタ。
<i>tau</i>	(ローカル) COMPLEX (pcunmrq の場合) DOUBLE COMPLEX (pzunmrq の場合) 配列、次元は $\text{LOCc}(ja+k-1)$ 。 <a href="#">p?gerqf</a> によって返される、基本リフレクタ $H(i)$ のスカラー係数 $\tau(i)$ を含む。 <i>tau</i> は、分散行列 $A$ に関連付けられる。

<i>c</i>	<p>(ローカル)  COMPLEX (pcunmrq の場合)  DOUBLE COMPLEX (pzunmrq の場合)。  ローカルメモリにある、ローカル次元 (<math>lld\_c</math>, <math>LOC(jc+n-1)</math>) の配列へのポインタ。</p> <p>因子分解する分散行列 <math>\text{sub}(C)</math> のローカル部分を含む。</p>
<i>ic, jc</i>	<p>(グローバル) INTEGER。  それぞれ、部分行列 <math>C</math> の最初の行と最初の列を示す、グローバル配列 <math>c</math> の行インデックスと列インデックス。</p>
<i>desc</i>	<p>(グローバルおよびローカル) INTEGER。配列、次元は (<math>dlen\_</math>)。  分散行列 <math>C</math> の配列ディスクリプタ。</p>
<i>work</i>	<p>(ローカル)  COMPLEX (pcunmrq の場合)  DOUBLE COMPLEX (pzunmrq の場合)。次元 <math>lwork</math> のワークスペース配列。</p>
<i>lwork</i>	<p>(ローカルまたはグローバル) INTEGER。  <math>work</math> の次元は</p> <p><math>side = 'L'</math> の場合、  <math>lwork \geq \max ((mb\_a * (mb\_a - 1)) / 2, (mpc0 + \max (mqa0 + \text{numroc}(\text{numroc}(n + iroffa, mb\_a, 0, 0, NPROW), mb\_a, 0, 0, lcmp), nqc0)) * mb\_a) + mb\_a * mb\_a)</math></p> <p><math>side = 'R'</math> の場合、  <math>lwork \geq \max ((mB\_A * (mB\_A - 1)) / 2, (mpC0 + nqC0) * mB\_A) + mB\_A * mB\_A)</math></p> <p>でなければならない。  ここで、</p> <p><math>lcmp = lcm / NPROW</math> で <math>lcm = ilcm(NPROW, NPCOL)</math>、  <math>iroffa = \text{mod}(ia - 1, mb\_a)</math>、  <math>icoffa = \text{mod}(ja - 1, nb\_a)</math>、  <math>iacol = \text{indxg2p}(ja, nb\_a, MYCOL, csrc\_a, NPCOL)</math>、  <math>mqa0 = \text{numroc}(m + icoffa, nb\_a, MYCOL, iacol, NPCOL)</math>、  <math>iroffc = \text{mod}(ic - 1, mb\_c)</math>、  <math>icoffc = \text{mod}(jc - 1, nb\_c)</math>、  <math>icrow = \text{indxg2p}(ic, mb\_c, MYROW, rsrc\_c, NPROW)</math>、</p>

```
iccol = indxcg2p(jc, nb_c, MYCOL, csrc_c, NPCOL),  
mpc0 = numroc(m+iroffc, mb_c, MYROW, icrow, NPROW),  
nqc0 = numroc(n+icoffc, nb_c, MYCOL, iccol, NPCOL)
```

ilcm、indxcg2p および numroc は、ScaLAPACK ツール関数である。  
MYROW、MYCOL、NPROW および NPCOL は、サブルーチン  
blacs\_gridinfo を呼び出して決定できる。

lwork = -1 の場合、lwork はグローバル入力であり、ワークスペース  
のクエリとみなされ、ルーチンはすべての work 配列の最小かつ最適  
なサイズだけを計算する。各値は該当する work 配列の最初のエン  
トリとして返され、[pxerbla](#) はエラー・メッセージを生成しない。

### 出力パラメータ

<i>c</i>	積 $Q * \text{sub}(C)$ 、 $Q' \text{sub}(C)$ 、 $\text{sub}(C) * Q'$ 、または $\text{sub}(C) * Q$ のいずれかに よって上書きされる。
<i>work(1)</i>	終了時に、 <i>work(1)</i> には、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が格納される。
<i>info</i>	(グローバル) INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> < 0 の場合： <i>i</i> 番目の引数が配列で、 <i>j</i> 番目の値が不正だった場合、 <i>info</i> = -( <i>i</i> *100+ <i>j</i> )。 <i>i</i> 番目の引数がスカラーで値が不正だった場合、 <i>info</i> = - <i>i</i> 。

---

## p?tzzrf

上台形行列 *A* を上三角形式に縮退させる。

---

### 構文

```
call pstzzrf( m, n, a, ia, ja, desca, tau, work, lwork, info )  
call pdtzzrf( m, n, a, ia, ja, desca, tau, work, lwork, info )  
call pztzzrf( m, n, a, ia, ja, desca, tau, work, lwork, info )  
call pztzzrf( m, n, a, ia, ja, desca, tau, work, lwork, info )
```



## 説明

このルーチンは、直交 / ユニタリ変換を用いて、 $m \times n$  ( $m \leq n$ ) の実 / 複素上台形行列  $\text{sub}(A) = (ia:ia+m-1, ja:ja+n-1)$  を上三角形式に縮退させる。上台形行列  $A$  は、次のように因子分解される。

$$A = (R \ 0) * Z。$$

ここで、 $Z$  は  $n \times n$  の直交 / ユニタリ行列、 $R$  は  $m \times m$  の上三角行列。

## 入力パラメータ

$m$	(グローバル) INTEGER。 部分行列 $\text{sub}(A)$ の行数 ( $m \geq 0$ )。
$n$	(グローバル) INTEGER。 部分行列 $\text{sub}(A)$ の列数 ( $n \geq 0$ )。
$a$	(ローカル) REAL (pztzrzf の場合) DOUBLE PRECISION (pdtzrzf の場合)。 COMPLEX (pctzrzf の場合)。 DOUBLE COMPLEX (pzdtzrzf の場合)。 ローカルメモリにある、次元 ( $lld\_a$ , $LOCc(ja+n-1)$ ) の配列へのポインタ。因子分解する $m \times n$ の分散行列 $\text{sub}(A)$ のローカル部分を含む。
$ia, ja$	(グローバル) INTEGER。 それぞれ、部分行列 $A$ の最初の行と最初の列を示す、グローバル配列 $a$ の行インデックスと列インデックス。
$desca$	(グローバルおよびローカル) INTEGER。 配列、次元は ( $dlen$ )。分散行列 $A$ の配列ディスクリプタ。
$work$	(ローカル) REAL (pztzrzf の場合) DOUBLE PRECISION (pdtzrzf の場合)。 COMPLEX (pctzrzf の場合)。 DOUBLE COMPLEX (pzdtzrzf の場合)。次元 $lwork$ のワークスペース配列。
$lwork$	(ローカルまたはグローバル) INTEGER。 $work$ の次元は $lwork \geq mb\_a * (mp0 + nq0 + mb\_a)$ でなければならない。 ここで、 $iroff = \text{mod}(ia-1, mb\_a)$ 、

```
icoff = mod(ja-1, nb_a)、
iarow = indxg2p(ia, mb_a, MYROW, rsrc_a, NPROW)、
iacol = indxg2p(ja, nb_a, MYCOL, csrc_a, NPCOL)、
mp0 = numroc (m+iroff, mb_a, MYROW, iarow, NPROW)、
nq0 = numroc (n+icoff, nb_a, MYCOL, iacol, NPCOL)
```

indxg2p および numroc は、ScaLAPACK ツール関数である。MYROW、MYCOL、NPROW、および NPCOL は、サブルーチン blacs\_gridinfo を呼び出して決定できる。

lwork = -1 の場合、lwork はグローバル入力であり、ワークスペースのクエリとみなされ、ルーチンはすべての work 配列の最小かつ最適なサイズだけを計算する。各値は該当する work 配列の最初のエン트리として返され、[pxerbla](#) はエラー・メッセージを生成しない。

## 出力パラメータ

<i>a</i>	終了時に、sub( <i>A</i> ) の先頭の $m \times m$ 上三角部分には上三角行列 <i>R</i> が格納される。sub( <i>A</i> ) の最初の <i>m</i> 行の <i>m</i> +1 から <i>n</i> までの成分は、配列 <i>tau</i> とともに、基本リフレクタ <i>m</i> の積として直交/ユニタリ行列 <i>Z</i> を表す。
<i>work</i> (1)	終了時に、 <i>work</i> (1) には、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が格納される。
<i>tau</i>	(ローカル) REAL (pstzrzf の場合) DOUBLE PRECISION (pdtzrzf の場合)。 COMPLEX (pctzrzf の場合)。 DOUBLE COMPLEX (pztzrzf の場合)。 配列、次元は <i>LOCr</i> ( <i>ia</i> + <i>m</i> -1)。 基本リフレクタのスカラー係数が格納される。 <i>tau</i> は、分散行列 <i>A</i> に関連付けられる。
<i>info</i>	(グローバル) INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> < 0 の場合： <i>i</i> 番目の引数が配列で、 <i>j</i> 番目の値が不正だった場合、 <i>info</i> = -( <i>i</i> *100+ <i>j</i> )。 <i>i</i> 番目の引数がスカラーで値が不正だった場合、 <i>info</i> = - <i>i</i> 。

## アプリケーション・ノート

因子分解は Householder 法を用いて得る。sub(A) の  $(m - k + 1)$  番目の行にゼロを導入するために共役置換が使用される  $k$  番目の変換行列  $Z(k)$  は、次の形式になる。

$$Z(k) = \begin{bmatrix} i & 0 \\ 0 & T(k) \end{bmatrix}$$

ここで、

$$T(k) = i - \tau u(k) u(k)',$$

$$u(k) = \begin{bmatrix} 1 \\ 0 \\ z(k) \end{bmatrix}$$

$\tau$  はスカラー、 $Z(k)$  は  $(n - m)$  成分で構成されるベクトルである。 $\tau$  および  $Z(k)$  は、sub(A) の  $k$  番目の行の成分をゼロにするために選択される。スカラー  $\tau$  は、 $Z(k)$  の成分が  $a(k, m + 1), \dots, a(k, n)$  にあるように、 $\tau$  の  $k$  番目の成分および sub(A) の  $k$  番目の行のベクトル  $u(k)$  で返される。 $R$  の成分は sub(A) の上三角部分で返される。 $Z$  は次の式で与えられる。

$$Z = Z(1) * Z(2) * \dots * Z(m)$$

## p?ormrz

一般行列に p?tzrzf によって求めた上三角形式に縮退した直交行列を掛ける。

### 構文

```
call psormrz( side, trans, m, n, k, l, a, ia, ja, desca, tau, c, ic, jc,
              descc, work, lwork, info )
call pdormrz( side, trans, m, n, k, l, a, ia, ja, desca, tau, c, ic, jc,
              descc, work, lwork, info )
```

### 説明

このルーチンは、 $m \times n$  の一般実分散行列 sub(C) = C(ic:ic+m-1,jc:jc+n-1) を以下により上書きする。

	$side = 'L'$	$side = 'R'$
$trans = 'N':$	$Q \text{ sub}(C)$	$\text{sub}(C) Q$
$trans = 'T':$	$Q^T \text{ sub}(C)$	$\text{sub}(C) Q^T$

ここで、 $Q$  は、 $k$  個の以下の基本リフレクタの積として定義される実直交分散行列である。

$$Q = H(1) H(2) \dots H(k)$$

[p?tzrrzf](#) によって返される。 $Q$  の次数は、 $side = 'L'$  の場合は  $m$ 、 $side = 'R'$  の場合は  $n$  である。

## 入力パラメータ

$side$	(グローバル) CHARACTER = $'L'$ の場合、 $Q$ または $Q^T$ は左側から適用される。 = $'R'$ の場合、 $Q$ または $Q^T$ は右側から適用される。
$trans$	(グローバル) CHARACTER = $'N'$ の場合、 $Q$ が適用される (転置なし)。 = $'T'$ の場合、 $Q^T$ が適用される (転置)。
$m$	(グローバル) INTEGER。 分散行列 $\text{sub}(C)$ の行数 ( $m \geq 0$ )。
$n$	(グローバル) INTEGER。 分散行列 $\text{sub}(C)$ の列数 ( $n \geq 0$ )。
$k$	(グローバル) INTEGER。 その積が行列 $Q$ を定義する基本リフレクタの個数。次の制約がある。 $side = 'L'$ の場合、 $m \geq k \geq 0$ 、 $side = 'R'$ の場合、 $n \geq k \geq 0$ 。
$l$	(グローバル)。 Householder リフレクタとして意味を持つ部分が格納されている分散部分行列 $\text{sub}(A)$ の列数。 $side = 'L'$ の場合、 $m \geq l \geq 0$ 、 $side = 'R'$ の場合、 $n \geq l \geq 0$ 。
$a$	(ローカル) REAL (psormrz の場合) DOUBLE PRECISION (pdormrz の場合)。 ローカルメモリにある、次元 ( $lld\_a$ , $LOCc(ja+m-1)$ ) ( $side = 'L'$ の場合) または ( $lld\_a$ , $LOCc(ja+n-1)$ ) ( $side = 'R'$ の場合) の配列へ

	<p>のポインタ。ここで、<math>lld\_a \geq \max(1, LOCr(ia+k-1))</math> である。<math>i</math> 番目の行は、<code>p?tzrzf</code> によって分散行列引数 <math>a(ia:ia+k-1, ja:*)</math> の <math>k</math> 行に返される、基本リフレクタ <math>H(i)</math> (<math>ia \leq i \leq ia+k-1</math>) を定義するベクトルを含んでいなければならない。<math>a(ia:ia+k-1, ja:*)</math> はルーチンにより変更されるが終了時に復元される。</p>
<i>ia, ja</i>	<p>(グローバル) INTEGER。 それぞれ、部分行列 <math>A</math> の最初の行と最初の列を示す、グローバル配列 <math>a</math> の行インデックスと列インデックス。</p>
<i>desca</i>	<p>(グローバルおよびローカル) INTEGER。配列、次元は (<math>dlen\_</math>)。 分散行列 <math>A</math> の配列ディスクリプタ。</p>
<i>tau</i>	<p>(ローカル) REAL (psormrz の場合) DOUBLE PRECISION (pdormrz の場合) 配列、次元は <math>LOCc(ia+k-1)</math>。 <a href="#">p?tzrzf</a> によって返される、基本リフレクタ <math>H(i)</math> のスカラー係数 <math>\tau(i)</math> を含む。<math>\tau</math> は、分散行列 <math>A</math> に関連付けられる。</p>
<i>c</i>	<p>(ローカル) REAL (psormrz の場合) DOUBLE PRECISION (pdormrz の場合) ローカルメモリにある、ローカル次元 (<math>lld\_c, LOCc(jc+n-1)</math>) の配列へのポインタ。 因子分解する分散行列 <math>\text{sub}(C)</math> のローカル部分を含む。</p>
<i>ic, jc</i>	<p>(グローバル) INTEGER。 それぞれ、部分行列 <math>C</math> の最初の行と最初の列を示す、グローバル配列 <math>c</math> の行インデックスと列インデックス。</p>
<i>descc</i>	<p>(グローバルおよびローカル) INTEGER。配列、次元は (<math>dlen\_</math>)。 分散行列 <math>C</math> の配列ディスクリプタ。</p>
<i>work</i>	<p>(ローカル) REAL (psormrz の場合) DOUBLE PRECISION (pdormrz の場合)。次元 <math>lwork</math> のワークスペース配列。</p>

*lwork* (ローカルまたはグローバル) INTEGER。  
*work* の次元は

*side* = 'L' の場合、  
 $lwork \geq \max((mb\_a * (mb\_a - 1)) / 2, (mpc0 + \max(mqa0 + \text{numroc}(\text{numroc}(n + iroffc, mb\_a, 0, 0, NPROW), mb\_a, 0, 0, lcmp), nqc0)) * mb\_a) + mb\_a * mb\_a)$

*side* = 'R' の場合、  
 $lwork \geq \max((mb\_a * (mb\_a - 1)) / 2, (mpc0 + nqc0) * mb\_a) + mb\_a * mb\_a$

でなければならない。  
 ここで、

$lcmp = lcm / NPROW$  で  $lcm = ilcm(NPROW, NPCOL)$ 、  
 $iroffa = \text{mod}(ia - 1, mb\_a)$ ,  $icoffa = \text{mod}(ja - 1, nb\_a)$ 、  
 $iacol = \text{indxg2p}(ja, nb\_a, MYCOL, csrc\_a, NPCOL)$ 、  
 $mqa0 = \text{numroc}(n + icoffa, nb\_a, MYCOL, iacol, NPCOL)$ 、  
 $iroffc = \text{mod}(ic - 1, mb\_c)$ 、  
 $icoffc = \text{mod}(jc - 1, nb\_c)$ 、  
 $icrow = \text{indxg2p}(ic, mb\_c, MYROW, rsrc\_c, NPROW)$ 、  
 $iccol = \text{indxg2p}(jc, nb\_c, MYCOL, csrc\_c, NPCOL)$ 、  
 $mpc0 = \text{numroc}(m + iroffc, mb\_c, MYROW, icrow, NPROW)$ 、  
 $nqc0 = \text{numroc}(n + icoffc, nb\_c, MYCOL, iccol, NPCOL)$

$ilcm$ 、 $\text{indxg2p}$  および  $\text{numroc}$  は、ScaLAPACK ツール関数である。  
 $MYROW$ 、 $MYCOL$ 、 $NPROW$  および  $NPCOL$  は、サブルーチン `blacs_gridinfo` を呼び出して決定できる。

*lwork* = -1 の場合、*lwork* はグローバル入力であり、ワークスペースのクエリとみなされ、ルーチンはすべての *work* 配列の最小かつ最適なサイズだけを計算する。各値は該当する *work* 配列の最初のエン트리として返され、[pxerbla](#) はエラー・メッセージを生成しない。

## 出力パラメータ

*c* 積  $Q * \text{sub}(C)$ 、 $Q' \text{sub}(C)$ 、 $\text{sub}(C) * Q'$ 、または  $\text{sub}(C) * Q$  のいずれかによって上書きされる。

<code>work(1)</code>	終了時に、 <code>work(1)</code> には、最適なパフォーマンスを得るために必要な <code>lwork</code> の最小値が格納される。
<code>info</code>	(グローバル) INTEGER。 <code>info = 0</code> の場合、実行は正常に終了したことを示す。 <code>info &lt; 0</code> の場合： $i$ 番目の引数が配列で、 $j$ 番目の値が不正だった場合、 <code>info = -(i*100+j)</code> 。 $i$ 番目の引数がスカラで値が不正だった場合、 <code>info = -i</code> 。

## p?unmrz

一般行列に `p?tzrzf` によって求めた上三角形式に縮退したユニタリ変換行列を掛ける。

### 構文

```
call pcunmrz( side, trans, m, n, k, l, a, ia, ja, desca, tau, c, ic, jc,
              descc, work, lwork, info )
call pzunmrz( side, trans, m, n, k, l, a, ia, ja, desca, tau, c, ic, jc,
              descc, work, lwork, info )
```

### 説明

このルーチンは、 $m \times n$  の一般複素分散行列  $\text{sub}(C) = C(ic:ic+m-1, jc:jc+n-1)$  を以下により上書きする。

	<code>side = 'L'</code>	<code>side = 'R'</code>
<code>trans = 'N':</code>	$Q \text{ sub}(C)$	$\text{sub}(C) Q$
<code>trans = 'C':</code>	$Q^H \text{ sub}(C)$	$\text{sub}(C) Q^H$

ここで、 $Q$  は、 $k$  個の以下の基本リフレクタの積として定義される複素ユニタリ分散行列である。

$$Q = H(1)' H(2)' \dots H(k)'$$

[pctzrzf](#)/[pztzrzf](#) によって返される。 $Q$  の次数は、`side = 'L'` の場合は  $m$ 、`side = 'R'` の場合は  $n$  である。

## 入力パラメータ

<i>side</i>	(グローバル) CHARACTER ='L' の場合、 $Q$ または $Q^H$ は左側から適用される。 ='R' の場合、 $Q$ または $Q^H$ は右側から適用される。
<i>trans</i>	(グローバル) CHARACTER ='N' の場合、 $Q$ が適用される (転置なし)。 ='C' の場合、 $Q^H$ が適用される (共役転置)。
<i>m</i>	(グローバル) INTEGER。 分散行列 $\text{sub}(C)$ の行数 ( $m \geq 0$ )。
<i>n</i>	(グローバル) INTEGER。 分散行列 $\text{sub}(C)$ の列数 ( $n \geq 0$ )。
<i>k</i>	(グローバル) INTEGER。 その積が行列 $Q$ を定義する基本リフレクタの個数。次の制約がある。 <i>side</i> = 'L' の場合、 $m \geq k \geq 0$ 、 <i>side</i> = 'R' の場合、 $n \geq k \geq 0$ 。
<i>a</i>	(ローカル) COMPLEX (pcunmrz の場合) DOUBLE COMPLEX (pzunmrz の場合)。 ローカルメモリにある、次元 ( $lld\_a, LOCc(ja+m-1)$ ) ( <i>side</i> = 'L' の場合) または ( $lld\_a, LOCc(ja+n-1)$ ) ( <i>side</i> = 'R' の場合) の配列へのポインタ。ここで、 $lld\_a \geq \max(1, LOCr(ja+k-1))$ である。 $i$ 番目の行は、 <a href="#">p?gerqf</a> によって分散行列引数 $a(ia:ia+k-1, ja:*)$ の $k$ 行に返される、基本リフレクタ $H(i)$ ( $ia < i < ia+k-1$ ) を定義するベクトルを含んでいなければならない。 $a(ia:ia+k-1, ja*)$ はルーチンにより変更されるが終了時に復元される。
<i>ia, ja</i>	(グローバル) INTEGER。 それぞれ、部分行列 $A$ の最初の行と最初の列を示す、グローバル配列 $a$ の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( $dlen\_$ )。 分散行列 $A$ の配列ディスクリプタ。
<i>tau</i>	(ローカル) COMPLEX (pcunmrz の場合) DOUBLE COMPLEX (pzunmrz の場合)



	<p>配列、次元は <math>LOCc(ia+k-1)</math>。  <a href="#">p?gerqf</a> によって返される、基本リフレクタ <math>H(i)</math> のスカラー係数 <math>\tau(i)</math> を含む。<math>\tau</math> は、分散行列 <math>A</math> に関連付けられる。</p>
<i>c</i>	<p>(ローカル)          COMPLEX (pcunmrz の場合)          DOUBLE COMPLEX (pzunmrz の場合)。          ローカルメモリにある、ローカル次元 (<math>lld\_c</math>, <math>LOCc(jc+n-1)</math>) の配列へのポインタ。          因子分解する分散行列 <math>\text{sub}(C)</math> のローカル部分を含む。</p>
<i>ic, jc</i>	<p>(グローバル) INTEGER。          それぞれ、部分行列 <math>C</math> の最初の行と最初の列を示す、グローバル配列 <math>c</math> の行インデックスと列インデックス。</p>
<i>descc</i>	<p>(グローバルおよびローカル) INTEGER。配列、次元は (<math>dlen\_</math>)。          分散行列 <math>C</math> の配列ディスクリプタ。</p>
<i>work</i>	<p>(ローカル)          COMPLEX (pcunmrz の場合)          DOUBLE COMPLEX (pzunmrz の場合)。次元 <math>lwork</math> のワークスペース配列。</p>
<i>lwork</i>	<p>(ローカルまたはグローバル) INTEGER。  <math>work</math> の次元は</p> <p><math>side = 'L'</math> の場合、  <math>lwork \geq \max ((mb\_a * (mb\_a - 1)) / 2, (mpc0 + \max (mqa0 + \text{numroc}(\text{numroc}(n + iroffc, mb\_a, 0, 0, NPROW), mb\_a, 0, 0, lcm), nqc0)) * mb\_a) + mb\_a * mb\_a</math></p> <p><math>side = 'R'</math> の場合、  <math>lwork \geq \max ((mb\_a * (mb\_a - 1)) / 2, (mpc0 + nqc0) * mb\_a) + mb\_a * mb\_a</math></p> <p>でなければならない。          ここで、</p> <p><math>lcmp = lcm / NPROW</math> で <math>lcm = ilcm(NPROW, NPCOL)</math>、  <math>iroffa = \text{mod}(ia - 1, mb\_a)</math>、  <math>icoffa = \text{mod}(ja - 1, nb\_a)</math>、  <math>iacol = \text{indxg2p}(ja, nb\_a, MYCOL, csrc\_a, NPCOL)</math>、  <math>mqa0 = \text{numroc}(m + icoffa, nb\_a, MYCOL, iacol, NPCOL)</math>、  <math>iroffc = \text{mod}(ic - 1, mb\_c)</math>、</p>

```

icoffc = mod(jc-1, nb_c)、
icrow = indxg2p(ic, mb_c, MYROW, rsrc_c, NPROW)、
iccol = indxg2p(jc, nb_c, MYCOL, csrc_c, NPCOL)、
mpc0 = numroc(m+iroffc, mb_c, MYROW, icrow, NPROW)、
nqc0 = numroc(n+icoffc, nb_c, MYCOL, iccol, NPCOL)

```

ilcm、indxg2p および numroc は、ScaLAPACK ツール関数である。  
 MYROW、MYCOL、NPROW および NPCOL は、サブルーチン  
 blacs\_gridinfo を呼び出して決定できる。

lwork = -1 の場合、lwork はグローバル入力であり、ワークスペース  
 のクエリとみなされ、ルーチンはすべての work 配列の最小かつ最適  
 なサイズだけを計算する。各値は該当する work 配列の最初のエン  
 トリとして返され、[pxerbla](#) はエラー・メッセージを生成しない。

### 出力パラメータ

<i>c</i>	積 $Q * \text{sub}(C)$ 、 $Q' \text{sub}(C)$ 、 $\text{sub}(C) * Q'$ 、または $\text{sub}(C) * Q$ のいずれかに よって上書きされる。
<i>work(1)</i>	終了時に、 <i>work(1)</i> には、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が格納される。
<i>info</i>	(グローバル) INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> < 0 の場合： <i>i</i> 番目の引数が配列で、 <i>j</i> 番目の値が不正だった場合、 <i>info</i> = -( <i>i</i> *100+ <i>j</i> )。 <i>i</i> 番目の引数がスカラで値が不正だった場合、 <i>info</i> = - <i>i</i> 。

## p?ggqrf

汎用  $QR$  因子分解を行う。

### 構文

```
call psggqrf(n, m, p, a, ia, ja, desca, taua, b, ib, jb, descb, taub,
             work, lwork, info)
call pdggqrf(n, m, p, a, ia, ja, desca, taua, b, ib, jb, descb, taub,
             work, lwork, info)
call pcggqrf(n, m, p, a, ia, ja, desca, taua, b, ib, jb, descb, taub,
             work, lwork, info)
call pzggqrf(n, m, p, a, ia, ja, desca, taua, b, ib, jb, descb, taub,
             work, lwork, info)
```

### 説明

このルーチンは、 $n \times m$  の行列

$$\text{sub}(A) = A(ia:ia+n-1, ja:ja+m-1)$$

および  $n \times p$  の行列

$$\text{sub}(B) = B(ib:ib+n-1, jb:jb+p-1)$$

の汎用  $QR$  因子分解を、

$$\text{sub}(A) = Q R, \quad \text{sub}(B) = Q T Z$$

として行う。

ここで、 $Q$  は  $n \times n$  の直交 / ユニタリ行列、 $Z$  は  $p \times p$  の直交 / ユニタリ行列、 $R$  および  $T$  は、次のいずれかの形式である。

$n \geq m$  の場合、

$$R = \begin{pmatrix} R_{11} & \\ & 0 \end{pmatrix} \begin{matrix} m \\ n-m \end{matrix}$$

$m$

または  $n < m$  の場合、

$$R = \begin{pmatrix} R_{11} & R_{12} \\ & \end{pmatrix} \begin{matrix} n \\ m-n \end{matrix}$$

ここで、 $R_{11}$  は上三角行列である。

$$T = \begin{pmatrix} 0 & T_{12} \\ p-n & n \end{pmatrix} \quad (n \leq p \text{ の場合}),$$

$$\text{または } T = \begin{pmatrix} T_{11} \\ T_{21} \end{pmatrix} \begin{pmatrix} n-p \\ p \end{pmatrix}, \quad (n > p \text{ の場合})$$

$p$

ここで、 $T_{12}$  または  $T_{21}$  は上三角行列である。

特に、 $\text{sub}(B)$  が正方で非特異の場合、 $\text{sub}(A)$  と  $\text{sub}(B)$  の  $GQR$  因子分解によって、 $\text{inv}(\text{sub}(B)) * \text{sub}(A)$  の  $QR$  因子分解が次のように暗黙的に得られる。

$$\text{inv}(\text{sub}(B)) * \text{sub}(A) = Z^H (T^{-1} R)$$

## 入力パラメータ

$n$	(グローバル) INTEGER。 分散行列 $\text{sub}(A)$ および $\text{sub}(B)$ の行数 ( $n \geq 0$ )。
$m$	(グローバル) INTEGER。 分散行列 $\text{sub}(A)$ の列数 ( $m \geq 0$ )。
$p$	INTEGER。 分散行列 $\text{sub}(B)$ の列数 ( $p \geq 0$ )。
$a$	(ローカル) REAL (psggqrf の場合) DOUBLE PRECISION (pdggqrf の場合) COMPLEX (pcggqrf の場合) DOUBLE COMPLEX (pzggqrf の場合)。 ローカルメモリにある、次元 ( $11d\_a, LOCc(ja+m-1)$ ) の配列へのポインタ。因子分解する $m \times n$ の行列 $\text{sub}(A)$ のローカル部分を含む。
$ia, ja$	(グローバル) INTEGER。 それぞれ、部分行列 $A$ の最初の行と最初の列を示す、グローバル配列 $a$ の行インデックスと列インデックス。
$desca$	(グローバルおよびローカル) INTEGER。配列、次元は ( $dlen\_$ )。 分散行列 $A$ の配列ディスクリプタ。

<i>b</i>	<p>(ローカル)</p> <p>REAL (psggqrf の場合)</p> <p>DOUBLE PRECISION (pdggqrf の場合)</p> <p>COMPLEX (pcggqrf の場合)</p> <p>DOUBLE COMPLEX (pzggqrf の場合)。</p> <p>ローカルメモリにある、次元 (<i>lld_b</i>, <i>LOCc(ja+p-1)</i>) の配列へのポインタ。因子分解する <math>n \times p</math> の行列 <i>sub(B)</i> のローカル部分を含む。</p>
<i>ib, jb</i>	<p>(グローバル) INTEGER。</p> <p>それぞれ、部分行列 <i>B</i> の最初の行と最初の列を示す、グローバル配列 <i>b</i> の行インデックスと列インデックス。</p>
<i>descb</i>	<p>(グローバルおよびローカル) INTEGER。配列、次元は (<i>dlen_</i>)。</p> <p>分散行列 <i>B</i> の配列ディスクリプタ。</p>
<i>work</i>	<p>(ローカル)</p> <p>REAL (psggqrf の場合)</p> <p>DOUBLE PRECISION (pdggqrf の場合)</p> <p>COMPLEX (pcggqrf の場合)</p> <p>DOUBLE COMPLEX (pzggqrf の場合)。次元 <i>lwork</i> のワークスペース配列。</p>
<i>lwork</i>	<p>(ローカルまたはグローバル) INTEGER。</p> <p><i>work</i> の次元は</p> $lwork \geq \max (nb\_a * (npa0 + mqa0 + nb\_a), \max((nb\_a * (nb\_a - 1)) / 2, (pqb0 + npb0) * nb\_a) + nb\_a * nb\_a, mb\_b * (npb0 + pqb0 + mb\_b))$ <p>でなければならない。ここで、</p> $iroffa = \text{mod}(ia - 1, mb\_a),$ $icoffa = \text{mod}(ja - 1, nb\_a),$ $iarow = \text{indxg2p}(ia, mb\_a, MYROW, rsrc\_a, NPROW),$ $iacol = \text{indxg2p}(ja, nb\_a, MYCOL, csrc\_a, NPCOL),$ $npa0 = \text{numroc}(n + iroffa, mb\_a, MYROW, iarow, NPROW),$ $mqa0 = \text{numroc}(m + icoffa, nb\_a, MYCOL, iacol, NPCOL)$ $iroffb = \text{mod}(ib - 1, mb\_b),$ $icoffb = \text{mod}(jb - 1, nb\_b),$ $ibrow = \text{indxg2p}(ib, mb\_b, MYROW, rsrc\_b, NPROW),$ $ibcol = \text{indxg2p}(jb, nb\_b, MYCOL, csrc\_b, NPCOL),$

```
npb0 = numroc (n+iroffa, mb_b, MYROW, ibrow, NPROW)、
pqb0 = numroc (m+icoffb, nb_b, MYCOL, ibcol, NPCOL)
```

numroc および indxcg2p は、ScaLAPACK ツール関数である。MYROW、MYCOL、NPROW および NPCOL は、サブルーチン blacs\_gridinfo を呼び出して決定できる。

lwork=-1 の場合、lwork はグローバル入力であり、ワークスペースのクエリとみなされ、ルーチンはすべての work 配列の最小かつ最適なサイズだけを計算する。各値は該当する work 配列の最初のエン트리として返され、[pxerbla](#) はエラー・メッセージを生成しない。

### 出力パラメータ

a	終了時に、sub(A) の対角成分とその上の成分は、 $\min(n, m) \times m$ の上台形行列 $R$ ( $n \geq m$ の場合、 $R$ は上三角行列) によって上書きされる。対角より下の成分は、配列 taua とともに、 $\min(n, m)$ 個の基本リフレクタの積として直交/ユニタリ行列 $Q$ を表す。(次の「アプリケーション・ノート」を参照)。
taua, taub	(ローカル) REAL (psggqrf の場合) DOUBLE PRECISION (pdggqrf の場合) COMPLEX (pcggqrf の場合) DOUBLE COMPLEX (pzggqrf の場合)。 配列、次元は $LOCc(ja+\min(n,m)-1)$ (taua の場合)、 $LOCr(ib+n-1)$ (taub の場合)。 配列 taua には、直交/ユニタリ行列 $Q$ を表す基本リフレクタのスカラ係数が格納される。taua は、分散行列 $A$ に関連付けられる。(次の「アプリケーション・ノート」を参照)。  配列 taub には、直交/ユニタリ行列 $Z$ を表す基本リフレクタのスカラ係数が格納される。taub は、分散行列 $B$ に関連付けられる。(次の「アプリケーション・ノート」を参照)。
work(1)	終了時に、work(1) には、最適なパフォーマンスを得るために必要な lwork の最小値が格納される。
info	(グローバル) INTEGER。 info=0 の場合、実行は正常に終了したことを示す。 info<0 の場合: i 番目の引数が配列で、j 番目の値が不正だった場合、 info = -(i*100+j)。i 番目の引数がスカラで値が不正だった場合、 info = -i。

## アプリケーション・ノート

行列  $Q$  は、以下の基本リフレクタの積として表現される。

$$Q = H(ja) H(ja+1) \dots H(ja+k-1)$$

ここで、 $k = \min(n, m)$  である。

それぞれの  $H(i)$  は次のような形式を持つ。

$$H(i) = I - \tau_{ua} * v * v'$$

ここで、 $\tau_{ua}$  は実 / 複素スカラ、 $v$  は実 / 複素ベクトルで、 $v(1:i-1) = 0$  および  $v(i) = 1$ 。終了時に、 $v(i+1:n)$  は  $A(ia+i:ia+n-1, ja+i-1)$  に、 $\tau_{ua}$  は  $\tau_{ua}(ja+i-1)$  に格納される。 $Q$  を明示的に生成するには、ScaLAPACK サブルーチン [p?orgqr](#)/[p?ungqr](#) を使用する。 $Q$  を使用して他の行列を更新するには、ScaLAPACK サブルーチン [p?ormqr](#)/[p?unmqr](#) を使用する。

行列  $Z$  は、次の基本リフレクタの積として表現される。

$$Z = H(ib) H(ib+1) \dots H(ib+k-1)$$

ここで、 $k = \min(n, p)$  である。

それぞれの  $H(i)$  は次のような形式を持つ。

$$H(i) = I - \tau_{ub} * v * v'$$

ここで、 $\tau_{ub}$  は実 / 複素スカラ、 $v$  は実 / 複素ベクトルで、 $v(p-k+i+1:p) = 0$  および  $v(p-k+i) = 1$ 。終了時に、 $v(1:p-k+i-1)$  は  $B(ib+n-k+i-1, jb:jb+p-k+i-2)$  に、 $\tau_{ub}$  は  $\tau_{ub}(ib+n-k+i-1)$  に格納される。 $Z$  を明示的に生成するには、ScaLAPACK サブルーチン [p?orgqr](#)/[p?ungqr](#) を使用する。 $Z$  を使用して他の行列を更新するには、ScaLAPACK サブルーチン [p?ormqr](#)/[p?unmqr](#) を使用する。

## p?ggrqf

汎用 RQ 因子分解を行う。

### 構文

```
call psggrqf(m, p, n, a, ia, ja, desca, taua, b, ib, jb, descb, taub,
             work, lwork, info)
call pdggrqf(m, p, n, a, ia, ja, desca, taua, b, ib, jb, descb, taub,
             work, lwork, info)
```

```
call pcggrqf(m, p, n, a, ia, ja, desca, taua, b, ib, jb, descb, taub,
            work, lwork, info)
call pzggrqf(m, p, n, a, ia, ja, desca, taua, b, ib, jb, descb, taub,
            work, lwork, info)
```

## 説明

このルーチンは、 $m \times n$  の行列

$\text{sub}(A) = (ia:ia+m-1, ja:ja+n-1)$

および  $p \times n$  の行列

$\text{sub}(B) = (ib:ib+p-1, ja:ja+n-1)$

の汎用  $RQ$  因子分解を、

$$\text{sub}(A) = R Q, \quad \text{sub}(B) = Z T Q$$

として行う。

ここで、 $Q$  は  $n \times n$  の直交 / ユニタリ行列、 $Z$  は  $p \times p$  の直交 / ユニタリ行列、 $R$  および  $T$  は、次のいずれかの形式である。

$$R = \begin{pmatrix} m & 0 & R_{12} \\ n-m & m \end{pmatrix}, \quad (m \leq n \text{ の場合}),$$

または

$$R = \begin{pmatrix} R_{11} & m-n \\ R_{12} & n \end{pmatrix}, \quad (m > n \text{ の場合})$$

ここで、 $R_{11}$  または  $R_{21}$  は上三角行列である。

$$T = \begin{pmatrix} T_{11} & n \\ 0 & p-n \end{pmatrix}, \quad (p \geq n \text{ の場合}),$$

または

$$T = \begin{pmatrix} p & T_{11} & T_{12} \\ p & n-p \end{pmatrix}, \quad (p < n \text{ の場合})$$

ここで、 $T_{11}$  は上三角行列である。



特に、 $\text{sub}(B)$  が正方で非特異の場合、 $\text{sub}(A)$  と  $\text{sub}(B)$  の  $GRQ$  因子分解によって、 $\text{sub}(A)*\text{inv}(\text{sub}(B))$  の  $RQ$  因子分解が次のように暗黙的に得られる。

$$\text{sub}(A)*\text{inv}(\text{sub}(B)) = (R*\text{inv}(T))*Z'$$

ここで、 $\text{inv}(\text{sub}(B))$  は行列  $\text{sub}(B)$  の逆行列、 $Z'$  は行列  $Z$  の転置行列である。

### 入力パラメータ

$m$	(グローバル) INTEGER。 分散行列 $\text{sub}(A)$ の行数 ( $m \geq 0$ )。
$p$	INTEGER。 分散行列 $\text{sub}(B)$ の行数 ( $p \geq 0$ )。
$n$	(グローバル) INTEGER。 分散行列 $\text{sub}(A)$ および $\text{sub}(B)$ の列数 ( $n \geq 0$ )。
$a$	(ローカル) REAL (psggrqf の場合) DOUBLE PRECISION (pdggrqf の場合) COMPLEX (pcggrqf の場合) DOUBLE COMPLEX (pzggrqf の場合)。 ローカルメモリにある、次元 ( $lld\_a$ , $LOCc(ja+n-1)$ ) の配列へのポインタ。因子分解する $m \times n$ の分散行列 $\text{sub}(A)$ のローカル部分を含む。
$ia, ja$	(グローバル) INTEGER。 それぞれ、部分行列 $A$ の最初の行と最初の列を示す、グローバル配列 $a$ の行インデックスと列インデックス。
$desca$	(グローバルおよびローカル) INTEGER。配列、次元は ( $dlen\_$ )。 分散行列 $A$ の配列ディスクリプタ。
$b$	(ローカル) REAL (psggrqf の場合) DOUBLE PRECISION (pdggrqf の場合) COMPLEX (pcggrqf の場合) DOUBLE COMPLEX (pzggrqf の場合)。  ローカルメモリにある、次元 ( $lld\_b$ , $LOCc(jb+n-1)$ ) の配列へのポインタ。因子分解する $n \times p$ の行列 $\text{sub}(B)$ のローカル部分を含む。
$ib, jb$	(グローバル) INTEGER。 それぞれ、部分行列 $B$ の最初の行と最初の列を示す、グローバル配列 $b$ の行インデックスと列インデックス。

<i>descb</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。分散行列 <i>B</i> の配列ディスクリプタ。
<i>work</i>	(ローカル) REAL (psggrqf の場合) DOUBLE PRECISION (pdggrqf の場合) COMPLEX (pcggrqf の場合) DOUBLE COMPLEX (pzggrqf の場合)。次元 <i>lwork</i> のワークスペース配列。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 <i>work</i> の次元は $lwork \geq \max (mb\_a * (mpa0 + nqa0 + mb\_a), \max((mb\_a * (mb\_a - 1)) / 2, (ppb0 + nqb0) * mb\_a) + mb\_a * mb\_a, nb\_b * (ppb0 + nqb0 + nb\_b))$ でなければならない。ここで、 $iroffa = \text{mod}(ia - 1, mb\_a),$ $icoffa = \text{mod}(ja - 1, nb\_a),$ $iarow = \text{indxg2p}(ia, mb\_a, MYROW, rsrc\_a, NPROW),$ $iacol = \text{indxg2p}(ja, nb\_a, MYCOL, csrc\_a, NPCOL),$ $mpa0 = \text{numroc}(m + iroffa, mb\_a, MYROW, iarow, NPROW),$ $nqa0 = \text{numroc}(m + icoffa, nb\_a, MYCOL, iacol, NPCOL)$ $iroffb = \text{mod}(ib - 1, mb\_b),$ $icoffb = \text{mod}(jb - 1, nb\_b),$ $ibrow = \text{indxg2p}(ib, mb\_b, MYROW, rsrc\_b, NPROW),$ $ibcol = \text{indxg2p}(jb, nb\_b, MYCOL, csrc\_b, NPCOL),$ $ppb0 = \text{numroc}(p + iroffb, mb\_b, MYROW, ibrow, NPROW),$ $nqb0 = \text{numroc}(n + icoffb, nb\_b, MYCOL, ibcol, NPCOL)$

numroc および indxg2p は、ScaLAPACK ツール関数である。MYROW、MYCOL、NPROW および NPCOL は、サブルーチン blacs\_gridinfo を呼び出して決定できる。

*lwork* = -1 の場合、*lwork* はグローバル入力であり、ワークスペースのクエリとみなされ、ルーチンはすべての *work* 配列の最小かつ最適なサイズだけを計算する。各値は該当する *work* 配列の最初のエン트리として返され、[pxerbla](#) はエラー・メッセージを生成しない。

## 出力パラメータ

<i>a</i>	終了時に、 $m \leq n$ の場合、 $A(ia:ia+m-1, ja+n-m:ja+n-1)$ の上三角部分に、 $m \times m$ の上三角行列 $R$ が格納される。 $m \geq n$ の場合、 $(m-n)$ 番目の劣対角成分とその上の成分に、 $m \times n$ の上台形行列 $R$ が格納される。残りの成分は、配列 <i>taua</i> とともに、 $\min(n, m)$ 個の基本リフレクタの積として直交/ユニタリ行列 $Q$ を表す。(次の「アプリケーション・ノート」を参照)。
<i>taua</i> , <i>taub</i>	(ローカル) REAL (psggqrf の場合) DOUBLE PRECISION (pdggqrf の場合) COMPLEX (pcggqrf の場合) DOUBLE COMPLEX (pzggqrf の場合)。 配列、次元は $LOCr(ia+m-1)$ ( <i>taua</i> の場合)、 $LOCc(jb+\min(p,n)-1)$ ( <i>taub</i> の場合)。 配列 <i>taua</i> には、直交/ユニタリ行列 $Q$ を表す基本リフレクタのスカラー係数が格納される。 <i>taua</i> は、分散行列 $A$ に関連付けられる。(次の「アプリケーション・ノート」を参照)。  配列 <i>taub</i> には、直交/ユニタリ行列 $Z$ を表す基本リフレクタのスカラー係数が格納される。 <i>taub</i> は、分散行列 $B$ に関連付けられる。(次の「アプリケーション・ノート」を参照)。
<i>work</i> (1)	終了時に、 <i>work</i> (1) には、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が格納される。
<i>info</i>	(グローバル) INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> < 0 の場合: <i>i</i> 番目の引数が配列で、 <i>j</i> 番目の値が不正だった場合、 <i>info</i> = $-(i*100+j)$ 。 <i>i</i> 番目の引数がスカラーで値が不正だった場合、 <i>info</i> = - <i>i</i> 。

## アプリケーション・ノート

行列  $Q$  は、以下の基本リフレクタの積として表現される。

$$Q = H(ia) H(ia+1) \dots H(ia+k-1)$$

ここで、 $k = \min(m, n)$  である。

それぞれの  $H(i)$  は次のような形式を持つ。

$$H(i) = I - \tau v v'$$

ここで、 $\tau_{aua}$  は実 / 複素スカラ、 $v$  は実 / 複素ベクトルで、 $v(n-k+i+1:n)=0$  および  $v(n-k+i)=1$ 。終了時に、 $v(1:n-k+i-1)$  は  $A(ia+m-k+i-1, ja:ja+n-k+i-2)$  に、 $\tau_{aua}$  は  $\tau_{aua}(ia+m-k+i-1)$  に格納される。 $Q$  を明示的に生成するには、ScaLAPACK サブルーチン [p?orgqr/p?ungrq](#) を使用する。 $Q$  を使用して他の行列を更新するには、ScaLAPACK サブルーチン [p?ormrq/p?unmrq](#) を使用する。

行列  $Z$  は、次の基本リフレクタの積として表現される。

$$Z = H(jb) H(jb+1) \dots H(jb+k-1)$$

ここで  $k = \min(p, n)$  である。

それぞれの  $H(i)$  は次のような形式を持つ。

$$H(i) = I - \tau_{aub} * v * v'$$

ここで、 $\tau_{aub}$  は実 / 複素スカラ、 $v$  は実 / 複素ベクトルで、 $v(1:i-1)=0$  および  $v(i)=1$ 。終了時に、 $v(i+1:p)$  は  $B(ib+i:ib+p-1, jb+i-1)$  に、 $\tau_{aub}$  は  $\tau_{aub}(jb+i-1)$  に格納される。 $Z$  を明示的に生成するには、ScaLAPACK サブルーチン [p?orgqr/p?ungqr](#) を使用する。 $Z$  を使用して他の行列を更新するには、ScaLAPACK サブルーチン [p?ormqr/p?unmqr](#) を使用する。

## 対称固有値問題

ScaLAPACK を使用して対称固有値問題を解くには、通常、行列を実三重対角形式  $T$  に縮退させてから三重対角行列  $T$  の固定値と固定ベクトルを見つける必要がある。

ScaLAPACK には、三重対角対称固定値問題を解くルーチンだけでなく、直交 (またはユニタリ) 相似変換  $A = QTQ^H$  を用いて行列を三重対角形式に縮退させるルーチンが含まれている。これらのルーチンのリストを、[表 6-4](#) に示す。

必要な項目 (固有値のみ、または固有値と固有ベクトル) と使用するアルゴリズム ( $QR$  アルゴリズム、または二分法と逆反復法) に応じて、さまざまな対称固有値問題用のルーチンがある。

**表 6-4 対称固有値問題を解くための計算ルーチン**

機能	密対称 / エルミート行列	直交 / ユニタリ行列	対称三重対角行列
三重対角形式 $A = QTQ^H$ に縮退させる	<a href="#">p?sytrd</a> / <a href="#">p?hetrd</a>		
縮退後に行列を掛ける		<a href="#">p?ormtr</a> / <a href="#">p?unmtr</a>	
$QR$ 法を用いて、三重対角行列 $T$ の固有値と固有ベクトルをすべて求める			<a href="#">?steqr2</a> <sup>*)</sup>
二分法を用いて、三重対角行列 $T$ の固有値を選択的に求める			<a href="#">p?stebz</a>
逆反復法を用いて、三重対角 行列 $T$ の固有ベクトルを選択的に求める			<a href="#">p?stein</a>

\*) このルーチンは、補助 ScaLAPACK ルーチンの一部として説明される。

## p?sytrd

直交相似変換を用いて、対称行列を実数対称三重対角形式に縮退させる。

### 構文

```
call pssytrd( uplo, n, a, ia, ja, desca, d, e, tau, work, lwork, info )
call pdsytrd( uplo, n, a, ia, ja, desca, d, e, tau, work, lwork, info )
```

### 説明

このルーチンは、直交相似変換を用いて、実対称行列  $\text{sub}(A)$  を対称三重対角形式  $T$  に縮退させる。

$$Q' \text{sub}(A) * Q = T$$

ここで、 $\text{sub}(A) = A(ia:ia+n-1, ja:ja+n-1)$ 。

### 入力パラメータ

**uplo** (グローバル) CHARACTER。  
対称行列  $\text{sub}(A)$  の上三角部分と下三角部分のどちらが格納されるかを指定する。  
  
 $uplo = 'U'$  の場合、上三角部分。  
 $uplo = 'L'$  の場合、下三角部分。

**n** (グローバル) INTEGER。  
分散行列  $\text{sub}(A)$  の次数 ( $n \geq 0$ )。

**a** (ローカル)  
REAL (pssytrd の場合)  
DOUBLE PRECISION (pdsytrd の場合)。  
ローカルメモリにある、次元 ( $lld\_a, LOCc(ja+n-1)$ ) の配列へのポインタ。呼び出し時に、この配列は、対称分散行列  $\text{sub}(A)$  のローカル部分を含む。  
 $uplo = 'U'$  の場合、 $\text{sub}(A)$  の先頭の  $n \times n$  上三角部分は行列の上三角部分を含む。厳密な下三角部分は参照されない。  
 $uplo = 'L'$  の場合、 $\text{sub}(A)$  の先頭の  $n \times n$  下三角部分は行列の下三角部分を含む。厳密な上三角部分は参照されない。(次の「アプリケーション・ノート」を参照)。

<i>ia, ja</i>	(グローバル) INTEGER。 それぞれ、部分行列 <i>A</i> の最初の行と最初の列を示す、グローバル配列 <i>a</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。 分散行列 <i>A</i> の配列ディスクリプタ。
<i>work</i>	(ローカル) REAL ( <i>pssytrd</i> の場合) DOUBLE PRECISION ( <i>pdsytrd</i> の場合)。 次元 <i>lwork</i> のワークスペース配列。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 <i>work</i> の次元は $lwork \geq \max(NB * (np + 1), 3 * NB)$ でなければならない。ここで、 $NB = mb\_a = nb\_a$ 、 $np = \text{numroc}(n, NB, MYROW, iarow, NPROW)$ 、 $iarow = \text{indxg2p}(ia, NB, MYROW, rsrc\_a, NPROW)$  $\text{indxg2p}$ および $\text{numroc}$ は、ScaLAPACK ツール関数である。MYROW、MYCOL、NPROW、および NPCOL は、サブルーチン <i>blacs_gridinfo</i> を呼び出して決定できる。  <i>lwork</i> = -1 の場合、 <i>lwork</i> はグローバル入力であり、ワークスペースのクエリとみなされ、ルーチンはすべての <i>work</i> 配列の最小かつ最適なサイズだけを計算する。各値は該当する <i>work</i> 配列の最初のエントリとして返され、 <a href="#">pxerbla</a> はエラー・メッセージを生成しない。

## 出力パラメータ

<i>a</i>	終了時に、 <i>uplo</i> = 'U' の場合、 <i>sub(A)</i> の対角成分と最初の優対角成分は、三重対角行列 <i>T</i> の対応する成分によって上書きされる。最初の優対角より上の成分は、配列 <i>tau</i> とともに、基本リフレクタの積として直交行列 <i>Q</i> を表す。 <i>uplo</i> = 'L' の場合、 <i>sub(A)</i> の対角成分と最初の劣対角成分は、三重対角行列 <i>T</i> の対応する成分によって上書きされる。最初の劣対角より下の成分は、配列 <i>tau</i> とともに、基本リフレクタの積として直交行列 <i>Q</i> を表す。(次の「アプリケーション・ノート」を参照)。
<i>d</i>	(ローカル) REAL ( <i>pssytrd</i> の場合) DOUBLE PRECISION ( <i>pdsytrd</i> の場合)。 配列、次元は <i>LOCc(ja+n-1)</i> 。三重対角行列 <i>T</i> の対角成分。

	$d(i) = A(i, i)$ <p><math>d</math> は、分散行列 <math>A</math> に関連付けられる。</p>
$e$	<p>(ローカル)</p> <p>REAL (pssytrd の場合)</p> <p>DOUBLE PRECISION (pdsytrd の場合)。</p> <p>配列、次元は <math>LOCc(ja+n-1)</math> (<math>uplo = 'U'</math> の場合)、または <math>LOCc(ja+n-2)</math> (それ以外の場合)。三重対角行列 <math>T</math> の非対角成分。</p> <p><math>e(i) = A(i, i+1)</math> (<math>uplo = 'U'</math> の場合)</p> <p><math>e(i) = A(i+1, i)</math> (<math>uplo = 'L'</math> の場合)。</p> <p><math>e</math> は分散行列 <math>A</math> に関連付けられる。</p>
$\tau$	<p>(ローカル)</p> <p>REAL (pssytrd の場合)</p> <p>DOUBLE PRECISION (pdsytrd の場合)。</p> <p>配列、次元は <math>LOCc(ja+n-1)</math>。この配列には、基本リフレクタのスカラ係数 <math>\tau</math> が格納される。<math>\tau</math> は、分散行列 <math>A</math> に関連付けられる。</p>
$work(1)$	<p>終了時に、<math>work(1)</math> には、最適なパフォーマンスを得るために必要な <math>lwork</math> の最小値が格納される。</p>
$info$	<p>(グローバル) INTEGER。</p> <p><math>info = 0</math> の場合、実行は正常に終了したことを示す。</p> <p><math>info &lt; 0</math> の場合：</p> <p><math>i</math> 番目の引数が配列で、<math>j</math> 番目の値が不正だった場合、<math>info = -(i*100+j)</math>。<math>i</math> 番目の引数がスカラで値が不正だった場合、<math>info = -i</math>。</p>

## アプリケーション・ノート

$uplo = 'U'$  の場合、行列  $Q$  は、以下の基本リフレクタの積として表現される。

$$Q = H(n-1) \dots H(2) H(1)$$

それぞれの  $H(i)$  は次のような形式を持つ。

$$H(i) = I - \tau * v * v'$$

ここで、 $\tau$  は実スカラ、 $v$  は実ベクトルで  $v(i+1:n) = 0$  および  $v(i) = 1$  である。終了時に、 $v(1:i-1)$  は  $A(ia:ia+i-2, ja+i)$  に、 $\tau$  は  $\tau(ja+i-1)$  に格納される。

$uplo = 'L'$  の場合、行列  $Q$  は、以下の基本リフレクタの積として表現される。

$$Q = H(1) H(2) \dots H(n-1)$$



それぞれの  $H(i)$  は次のような形式を持つ。

$$H(i) = i - \tau v v'$$

ここで、 $\tau$  は実スカラ、 $v$  は実ベクトルで  $v(1:i) = 0$  および  $v(i+1) = 1$  である。出力時、 $v(i+2:n)$  は  $A(ia+i+1:ia+n-1, ja+i-1)$  に、 $\tau$  は  $\tau(ja+i-1)$  に格納される。

終了時の  $\text{sub}(A)$  の内容 ( $n = 5$  の場合) を次に示す。

$\text{uplo} = 'U'$  の場合、

$$\begin{bmatrix} d & e & v2 & v3 & v4 \\ & d & e & v3 & v4 \\ & & d & e & v4 \\ & & & d & e \\ & & & & d \end{bmatrix}$$

$\text{uplo} = 'L'$  の場合、

$$\begin{bmatrix} d & & & & \\ e & d & & & \\ v1 & e & d & & \\ v1 & v2 & e & d & \\ v1 & v2 & v3 & e & d \end{bmatrix}$$

ここで、 $d$  と  $e$  は  $T$  の対角成分と非対角成分である。 $v_i$  は  $H(i)$  を定義するベクトルの成分を表す。

## p?ormtr

一般行列に p?sytrd によって求めた上三角形式に縮退した直交変換行列を掛ける。

### 構文

```
call psormtr( side, uplo, trans, m, n, a, ia, ja, desca, tau, c, ic, jc,
             descc, work, lwork, info )
call pdormtr( side, uplo, trans, m, n, a, ia, ja, desca, tau, c, ic, jc,
             descc, work, lwork, info )
```

### 説明

このルーチンは、 $m \times n$  の一般実分散行列  $\text{sub}(C) = C(ic:ic+m-1, jc:jc+n-1)$  を以下により上書きする。

	$side = 'L'$	$side = 'R'$
$trans = 'N':$	$Q \text{ sub}(C)$	$\text{sub}(C) Q$
$trans = 'T':$	$Q^T \text{ sub}(C)$	$\text{sub}(C) Q^T$

ここで、 $Q$  は次数  $nq$  の実直交分散行列で、 $nq = m$  ( $side = 'L'$  の場合)、 $nq = n$  ( $side = 'R'$  の場合) である。 $Q$  は、[p?sytrd](#) によって返される、 $nq$  個の基本リフレクタの積として定義される。

$uplo = 'U'$  の場合、 $Q = H(nq-1) \dots H(2) H(1)$ 。

$uplo = 'L'$  の場合、 $Q = H(1) H(2) \dots H(nq-1)$ 。

### 入力パラメータ

$side$	(グローバル) CHARACTER $= 'L'$ の場合、 $Q$ または $Q^T$ は左側から適用される。 $= 'R'$ の場合、 $Q$ または $Q^T$ は右側から適用される。
$trans$	(グローバル) CHARACTER $= 'N'$ の場合、 $Q$ が適用される (転置なし)。 $= 'T'$ の場合、 $Q^T$ が適用される (転置)。
$uplo$	(グローバル) CHARACTER。 $= 'U'$ の場合、 $A(ia:*, ja:*)$ の上三角は <a href="#">p?sytrd</a> からの基本リフレクタを格納する。

	<p>= 'L' の場合、<math>A(ia:*,ja:*)</math> の下三角は <math>p?sytrd</math> からの基本リフレクタを格納する。</p>
<i>m</i>	<p>(グローバル) INTEGER。 分散行列 <math>\text{sub}(C)</math> の行数 (<math>m \geq 0</math>)。</p>
<i>n</i>	<p>(グローバル) INTEGER。 分散行列 <math>\text{sub}(C)</math> の列数 (<math>n \geq 0</math>)。</p>
<i>a</i>	<p>(ローカル) REAL (psormtr の場合) DOUBLE PRECISION (pdormtr の場合)。 ローカルメモリにある、次元 <math>(lld\_a, LOCc(ja+m-1))</math> (<math>side='L'</math> の場合) または <math>(lld\_a, LOCc(ja+n-1))</math> (<math>side='R'</math> の場合) の配列へのポインタ。 <a href="#">p?sytrd</a> によって返される、基本リフレクタを定義する一連のベクトルを含む。 <math>side='L'</math> の場合、<math>lld\_a \geq \max(1, LOCr(ia+m-1))</math>。 <math>side='R'</math> の場合、<math>lld\_a \geq \max(1, LOCr(ia+n-1))</math>。</p>
<i>ia, ja</i>	<p>(グローバル) INTEGER。 それぞれ、部分行列 <math>A</math> の最初の行と最初の列を示す、グローバル配列 <math>a</math> の行インデックスと列インデックス。</p>
<i>desca</i>	<p>(グローバルおよびローカル) INTEGER。配列、次元は <math>(dlen\_)</math>。 分散行列 <math>A</math> の配列ディスクリプタ。</p>
<i>tau</i>	<p>(ローカル) REAL (psormtr の場合) DOUBLE PRECISION (pdormtr の場合)。 配列、<math>ltau</math> の次元。ここで、 <math>side='L'</math> および <math>uplo='U'</math> の場合、<math>ltau = LOCc(m\_a)</math>、 <math>side='L'</math> および <math>uplo='L'</math> の場合、<math>ltau = LOCc(ja+m-2)</math>、 <math>side='R'</math> および <math>uplo='U'</math> の場合、<math>ltau = LOCc(n\_a)</math>、 <math>side='R'</math> および <math>uplo='L'</math> の場合、<math>ltau = LOCc(ja+n-2)</math>。<math>\tau(i)</math> は、 <a href="#">p?sytrd</a> によって返される、基本リフレクタ <math>H(i)</math> のスカラー係数を含んでいなければならない。<math>\tau</math> は、分散行列 <math>A</math> に関連付けられる。</p>
<i>c</i>	<p>(ローカル) REAL (psormtr の場合) DOUBLE PRECISION (pdormtr の場合)。 ローカルメモリにある、次元 <math>(lld\_a, LOCc(ja+n-1))</math> の配列へのポインタ。分散行列 <math>\text{sub}(C)</math> のローカル部分を含む。</p>

*work* (ローカル)  
 REAL (psormtr の場合)  
 DOUBLE PRECISION (pdormtr の場合)。  
 次元 *lwork* のワークスペース配列。

*lwork* (ローカルまたはグローバル) INTEGER。 *work* の次元は  
*uplo* = 'U' の場合、  
*iaa*=*ia*。 *jaa*=*ja*+1、 *icc*=*ic*。 *jcc*=*jc*。  
*uplo* = 'L' の場合、  
*iaa*=*ia*+1、 *jaa*=*ja*。  
*side* = 'L' の場合、  
*icc*=*ic*+1。 *jcc*=*jc*。  
 または *icc*=*ic*。 *jcc*=*jc*+1。  
 でなければならない。

*side* = 'L' の場合、  
*mi*=*m*-1。 *ni*=*n*。  
 $lwork \geq \max((nb\_a*(nb\_a-1))/2, (nqc0 + mpc0)*nb\_a) + nb\_a * nb\_a$   
*side* = 'R' の場合、  
*mi*=*m*。 *ni* = *n*-1。  
 $lwork \geq \max((nb\_a*(nb\_a-1))/2, (nqc0 + \max(npa0 +$   
 $\text{numroc}(\text{numroc}(ni+icoffc, nb\_a, 0, 0, NPCOL), * nb\_a, 0, 0, lcmq),$   
 $mpc0))*nb\_a) + nb\_a * nb\_a$   
 でなければならない。

ここで、  $lcmq = lcm / NPCOL$  で  $lcm = ilcm(NPROW, NPCOL)$ 、  
*iroffa* = mod(*iaa*-1, *mb\_a*)、  
*icoffa* = mod(*jaa*-1, *nb\_a*)、  
*iarow* = indxcg2p (*iaa*, *mb\_a*, MYROW, *rsrc\_a*, NPROW)、  
*npa0* = numroc(*ni*+*iroffa*, *mb\_a*, MYROW, *iarow*, NPROW)、  
*iroffc* = mod(*icc*-1, *mb\_c*)、

```

icoffc = mod(jcc-1, nb_c)、
icrow = indxg2p (icc, mb_c, MYROW, rsrc_c, NPROW)、
iccol = indxg2p (jcc, nb_c, MYCOL, csrc_c, NPCOL)、
mpc0 = numroc(mi+iroffc, mb_c, MYROW, icrow, NPROW)、
nqc0 = numroc(ni+icoffc, nb_c, MYCOL, iccol, NPCOL)

```

ilcm、indxg2p および numroc は、ScaLAPACK ツール関数である。  
 MYROW、MYCOL、NPROW、および NPCOL は、サブルーチン  
 blacs\_gridinfo を呼び出して決定できる。lwork = -1 の場合、lwork  
 はグローバル入力であり、ワークスペースのクエリとみなされ、ルー  
 チンはすべての work 配列の最小かつ最適なサイズだけを計算する。  
 各値は該当する work 配列の最初のエントリとして返され、[pxerbla](#)  
 はエラー・メッセージを生成しない。

### 出力パラメータ

<i>c</i>	積 $Q \text{ sub}(C)$ 、 $Q' \text{ sub}(C)$ 、 $\text{sub}(C) Q'$ 、または $\text{sub}(C) Q$ のいずれかによっ て上書きされる。
<i>work(1)</i>	終了時に、 <i>work(1)</i> には、最適なパフォーマンスを得るために必要な lwork の最小値が格納される。
<i>info</i>	(グローバル) INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> < 0 の場合： <i>i</i> 番目の引数が配列で、 <i>j</i> 番目の値が不正だった場合、 <i>info</i> = -( <i>i</i> *100+ <i>j</i> )。 <i>i</i> 番目の引数がスカラで値が不正だった場合、 <i>info</i> = - <i>i</i> 。

## p?hetrd

ユニタリ相似変換を用いて、エルミート行列を  
エルミート三重対角形式に縮退させる。

### 構文

```

call pchetrd( uplo, n, a, ia, ja, desca, d, e, tau, work, lwork, info )
call pzhetrd( uplo, n, a, ia, ja, desca, d, e, tau, work, lwork, info )

```

## 説明

このルーチンは、ユニタリ相似変換を用いて、複素エルミート行列 `sub(A)` をエルミート三重対角形式  $T$  に縮退させる。

$$Q' \text{sub}(A) Q = T$$

ここで、 $\text{sub}(A) = A(\text{ia}:\text{ia}+n-1, \text{ja}:\text{ja}+n-1)$ 。

## 入力パラメータ

<code>uplo</code>	(グローバル) CHARACTER。 エルミート行列 <code>sub(A)</code> の上三角部分と下三角部分のどちらが格納されるかを指定する。 <code>uplo = 'U'</code> の場合、上三角部分。 <code>uplo = 'L'</code> の場合、下三角部分。
<code>n</code>	(グローバル) INTEGER。 分散行列 <code>sub(A)</code> の次数 ( $n \geq 0$ )。
<code>a</code>	(ローカル) COMPLEX ( <code>pchetrdr</code> の場合) DOUBLE COMPLEX ( <code>pzhetrdr</code> の場合)。 ローカルメモリにある、次元 ( <code>lld_a, LOCc(ja+n-1)</code> ) の配列へのポインタ。呼び出し時に、この配列は、エルミート分散行列 <code>sub(A)</code> のローカル部分を含む。 <code>uplo = 'U'</code> の場合、 <code>sub(A)</code> の先頭の $n \times n$ 上三角部分は行列の上三角部分を含む。厳密な下三角部分は参照されない。 <code>uplo = 'L'</code> の場合、 <code>sub(A)</code> の先頭の $n \times n$ 下三角部分は行列の下三角部分を含む。厳密な上三角部分は参照されない。(次の「アプリケーション・ノート」を参照)。
<code>ia, ja</code>	(グローバル) INTEGER。 それぞれ、部分行列 $A$ の最初の行と最初の列を示す、グローバル配列 <code>a</code> の行インデックスと列インデックス。
<code>desca</code>	(グローバルおよびローカル) INTEGER。配列、次元は ( <code>dlen_</code> )。 分散行列 $A$ の配列ディスクリプタ。
<code>work</code>	(ローカル) COMPLEX ( <code>pchetrdr</code> の場合)。 DOUBLE COMPLEX ( <code>pzhetrdr</code> の場合)。 次元 <code>lwork</code> のワークスペース配列。
<code>lwork</code>	(ローカルまたはグローバル) INTEGER。 <code>work</code> の次元は

$lwork \geq \max(NB * (np + 1), 3 * NB)$

でなければならない。ここで、 $NB = mb\_a = nb\_a$ 、

$np = \text{numroc}(n, NB, \text{MYROW}, iarow, \text{NPROW})$ 、

$iarow = \text{indxg2p}(ia, NB, \text{MYROW}, rsrc\_a, \text{NPROW})$

$\text{indxg2p}$  および  $\text{numroc}$  は、ScaLAPACK ツール関数である。MYROW、MYCOL、NPROW、および NPCOL は、サブルーチン `blacs_gridinfo` を呼び出して決定できる。

$lwork = -1$  の場合、 $lwork$  はグローバル入力であり、ワークスペースのクエリとみなされ、ルーチンはすべての `work` 配列の最小かつ最適なサイズだけを計算する。各値は該当する `work` 配列の最初のエントリとして返され、[pxerbla](#) はエラー・メッセージを生成しない。

## 出力パラメータ

- a 終了時に、 $uplo = 'U'$  の場合、 $\text{sub}(A)$  の対角成分と最初の優対角成分は、三重対角行列  $T$  の対応する成分によって上書きされる。最初の優対角より上の成分は、配列 `tau` とともに、基本リフレクタの積として直交行列  $Q$  を表す。 $uplo = 'L'$  の場合、 $\text{sub}(A)$  の対角成分と最初の劣対角成分は、三重対角行列  $T$  の対応する成分によって上書きされる。最初の劣対角より下の成分は、配列 `tau` とともに、基本リフレクタの積として直交行列  $Q$  を表す。(次の「アプリケーション・ノート」を参照)。
- d (ローカル)  
REAL (pchetrd の場合)  
DOUBLE PRECISION (pzhetrd の場合)。  
配列、次元は  $LOCc(ja+n-1)$ 。三重対角行列  $T$  の対角成分。  
 $d(i) = A(i, i)$   
 $d$  は、分散行列  $A$  に関連付けられる。
- e (ローカル)  
REAL (pchetrd の場合)  
DOUBLE PRECISION (pzhetrd の場合)。  
配列、次元は  $LOCc(ja+n-1)$  ( $uplo = 'U'$  の場合)、または  $LOCc(ja+n-2)$  (それ以外の場合)。三重対角行列  $T$  の非対角成分。  
 $e(i) = A(i, i+1)$  ( $uplo = 'U'$  の場合)  
 $e(i) = A(i+1, i)$  ( $uplo = 'L'$  の場合)。  
 $e$  は分散行列  $A$  に関連付けられる。

<i>tau</i>	(ローカル) COMPLEX (pchetrd の場合) DOUBLE COMPLEX (pzhetrd の場合)。 配列、次元は $LOCc(ja+n-1)$ 。この配列は、基本リフレクタのスカラー係数 <i>tau</i> を含む。 <i>tau</i> は、分散行列 <i>A</i> に関連付けられる。
<i>work(1)</i>	終了時に、 <i>work(1)</i> には、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が格納される。
<i>info</i>	(グローバル) INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> < 0 の場合： <i>i</i> 番目の引数が配列で、 <i>j</i> 番目の値が不正だった場合、 <i>info</i> = -( <i>i</i> *100+ <i>j</i> )。 <i>i</i> 番目の引数がスカラーで値が不正だった場合、 <i>info</i> = - <i>i</i> 。

## アプリケーション・ノート

*uplo* = 'U' の場合、行列 *Q* は、以下の基本リフレクタの積として表現される。

$$Q = H(n-1) \dots H(2) H(1)$$

それぞれの *H(i)* は次のような形式を持つ。

$$H(i) = I - \tau v v'$$

ここで、*tau* は複素スカラー、*v* は複素ベクトルで  $v(i+1:n) = 0$  および  $v(i) = 1$  である。終了時に、 $v(1:i-1)$  は  $A(ia:ia+i-2, ja+i)$  に、*tau* は  $\tau(ja+i-1)$  に格納される。

*uplo* = 'L' の場合、行列 *Q* は、以下の基本リフレクタの積として表現される。

$$Q = H(1) H(2) \dots H(n-1)$$

それぞれの *H(i)* は次のような形式を持つ。

$$H(i) = I - \tau v v'$$

ここで、*tau* は複素スカラー、*v* は複素ベクトルで  $v(1:i) = 0$  および  $v(i+1) = 1$  である。終了時に、 $v(i+2:n)$  は  $A(ia+i+1:ia+n-1, ja+i-1)$  に、*tau* は  $\tau(ja+i-1)$  に格納される。



終了時の  $\text{sub}(A)$  の内容 ( $n = 5$  の場合) を次に示す。

$\text{uplo} = 'U'$  の場合、

$$\begin{bmatrix} d & e & v2 & v3 & v4 \\ & d & e & v3 & v4 \\ & & d & e & v4 \\ & & & d & e \\ & & & & d \end{bmatrix}$$

$\text{uplo} = 'L'$  の場合、

$$\begin{bmatrix} d & & & & \\ e & d & & & \\ v1 & e & d & & \\ v1 & v2 & e & d & \\ v1 & v2 & v3 & e & d \end{bmatrix}$$

ここで、 $d$  と  $e$  は  $T$  の対角成分と非対角成分である。 $v_i$  は  $H(i)$  を定義するベクトルの成分を表す。

---

## p?unmtr

一般行列に p?hetrd によって求めた三角形式に縮退したユニタリ変換行列を掛ける。

---

### 構文

```
call pcunmtr( side, uplo, trans, m, n, a, ia, ja, desca, tau, c, ic, jc,
              descc, work, lwork, info )
call pzunmtr( side, uplo, trans, m, n, a, ia, ja, desca, tau, c, ic, jc,
              descc, work, lwork, info )
```

## 説明

このルーチンは、 $m \times n$  の一般複素分散行列  $\text{sub}(C) = C(ic:ic+m-1, jc:jc+n-1)$  を以下により上書きする。

	$side = 'L'$	$side = 'R'$
$trans = 'N':$	$Q \text{ sub}(C)$	$\text{sub}(C) Q$
$trans = 'C':$	$Q^H \text{ sub}(C)$	$\text{sub}(C) Q^H$

ここで、 $Q$  は、次数  $nq$  の複素ユニタリ分散行列で、 $nq = m$  ( $side = 'L'$  の場合)、 $nq = n$  ( $side = 'R'$  の場合) である。 $Q$  は、[p?hetrd](#) によって返される、 $nq-1$  個の基本リフレクタの積として定義される。

$uplo = 'U'$  の場合、 $Q = H(nq-1) \dots H(2) H(1)$ 。

$uplo = 'L'$  の場合、 $Q = H(1) H(2) \dots H(nq-1)$ 。

## 入力パラメータ

$side$	(グローバル) CHARACTER $= 'L'$ の場合、 $Q$ または $Q^H$ は左側から適用される。 $= 'R'$ の場合、 $Q$ または $Q^H$ は右側から適用される。
$trans$	(グローバル) CHARACTER $= 'N'$ の場合、 $Q$ が適用される (転置なし)。 $= 'C'$ の場合、 $Q^H$ が適用される (共役転置)。
$uplo$	(グローバル) CHARACTER。 $= 'U'$ の場合、 $A(ia:*, ja:*)$ の上三角は <a href="#">p?hetrd</a> からの基本リフレクタを格納する。 $= 'L'$ の場合、 $A(ia:*, ja:*)$ の下三角は <a href="#">p?hetrd</a> からの基本リフレクタを格納する。
$m$	(グローバル) INTEGER。 分散行列 $\text{sub}(C)$ の行数 ( $m \geq 0$ )。
$n$	(グローバル) INTEGER。 分散行列 $\text{sub}(C)$ の列数 ( $n \geq 0$ )。
$a$	(ローカル) REAL (pcunmtr の場合) DOUBLE PRECISION (pzunmtr の場合)。 ローカルメモリにある、次元 $(lld\_a, LOCc(ja+m-1))$ ( $side = 'L'$ の場合) または $(lld\_a, LOCc(ja+n-1))$ ( $side = 'R'$ の場合) の配列へのポインタ。 <a href="#">p?hetrd</a> によって返される、基本リフレクタを定義する一連のベクト

	<p>ルを含む。</p> <p><math>side = 'L'</math> の場合、<math>lld\_a \geq \max(1, LOCr(ia+m-1))</math>。</p> <p><math>side = 'R'</math> の場合、<math>lld\_a \geq \max(1, LOCr(ia+n-1))</math>。</p>
<i>ia, ja</i>	<p>(グローバル) INTEGER。</p> <p>それぞれ、部分行列 <i>A</i> の最初の行と最初の列を示す、グローバル配列 <i>a</i> の行インデックスと列インデックス。</p>
<i>desca</i>	<p>(グローバルおよびローカル) INTEGER。配列、次元は (<i>dlen</i><sub>0</sub>)。</p> <p>分散行列 <i>A</i> の配列ディスクリプタ。</p>
<i>tau</i>	<p>(ローカル)</p> <p>COMPLEX (<i>pcunmtr</i> の場合)</p> <p>DOUBLE COMPLEX (<i>pzunmtr</i> の場合)。</p> <p>配列、<i>ltau</i> の次元。ここで、</p> <p><math>side = 'L'</math> および <math>uplo = 'U'</math> の場合、<math>ltau = LOCc(m\_a)</math>、</p> <p><math>side = 'L'</math> および <math>uplo = 'L'</math> の場合、<math>ltau = LOCc(ja+m-2)</math>、</p> <p><math>side = 'R'</math> および <math>uplo = 'U'</math> の場合、<math>ltau = LOCc(n\_a)</math>、</p> <p><math>side = 'R'</math> および <math>uplo = 'L'</math> の場合、<math>ltau = LOCc(ja+n-2)</math>。<i>tau</i>(<i>i</i>) は、<a href="#">p?hetrd</a> によって返される、基本リフレクタ <i>H</i>(<i>i</i>) のスカラー係数を含んでいなければならない。<i>tau</i> は、分散行列 <i>A</i> に関連付けられる。</p>
<i>c</i>	<p>(ローカル)</p> <p>COMPLEX (<i>pcunmtr</i> の場合)</p> <p>DOUBLE COMPLEX (<i>pzunmtr</i> の場合)。</p> <p>ローカルメモリにある、次元 (<math>lld\_a, LOCc(ja+n-1)</math>) の配列へのポインタ。分散行列 <i>sub</i>(<i>C</i>) のローカル部分を含む。</p>
<i>work</i>	<p>(ローカル)</p> <p>COMPLEX (<i>pcunmtr</i> の場合)</p> <p>DOUBLE COMPLEX (<i>pzunmtr</i> の場合)。</p> <p>次元 <i>lwork</i> のワークスペース配列。</p>
<i>lwork</i>	<p>(ローカルまたはグローバル) INTEGER。<i>work</i> の次元は</p> <p><math>uplo = 'U'</math> の場合、</p> <p><math>iaa=ia</math>。 <math>jaa=ja+1</math>、<math>icc=ic</math>。 <math>jcc=jc</math>。</p> <p><math>uplo = 'L'</math> の場合、</p> <p><math>iaa=ia+1</math>、<math>jaa=ja</math>。</p> <p><math>side = 'L'</math> の場合、</p> <p><math>icc=ic+1</math>。 <math>jcc=jc</math>。</p>

または  $icc=ic$ 。  $jcc=jc+1$ 。

でなければならない。

$side = 'L'$  の場合、

$mi=m-1$ 。  $ni=n$ 。

$lwork \geq \max((nb\_a*(nb\_a-1))/2, (nqc0 + mpc0)*nb\_a) + nb\_a * nb\_a$

$side = 'R'$  の場合、

$mi=m$ 。  $mi = n-1$ 。

$lwork \geq \max((nb\_a*(nb\_a-1))/2, (nqc0 + \max(npa0 + \text{numroc}(\text{numroc}(ni+icoffc, nb\_a, 0, 0, NPCOL), * nb\_a, 0, 0, lcmq), mpc0))*nb\_a) + nb\_a * nb\_a$

でなければならない。

ここで、  $lcmq = lcm / NPCOL$  で  $lcm = ilcm(NPROW, NPCOL)$ 、

$iroffa = \text{mod}(iaa-1, mb\_a)$ 、

$icoffa = \text{mod}(jaa-1, nb\_a)$ 、

$iarow = \text{indxg2p}(iaa, mb\_a, MYROW, rsrc\_a, NPROW)$ 、

$npa0 = \text{numroc}(ni+iroffa, mb\_a, MYROW, iarow, NPROW)$ 、

$iroffc = \text{mod}(icc-1, mb\_c)$ 、

$icoffc = \text{mod}(jcc-1, nb\_c)$ 、

$icrow = \text{indxg2p}(icc, mb\_c, MYROW, rsrc\_c, NPROW)$ 、

$iccol = \text{indxg2p}(jcc, nb\_c, MYCOL, csrc\_c, NPCOL)$ 、

$mpc0 = \text{numroc}(mi+iroffc, mb\_c, MYROW, icrow, NPROW)$ 、

$nqc0 = \text{numroc}(ni+icoffc, nb\_c, MYCOL, iccol, NPCOL)$

$ilcm$ 、 $\text{indxg2p}$  および  $\text{numroc}$  は、ScaLAPACK ツール関数である。  
 $MYROW$ 、 $MYCOL$ 、 $NPROW$ 、および  $NPCOL$  は、サブルーチン `blacs_gridinfo` を呼び出して決定できる。 $lwork = -1$  の場合、 $lwork$  はグローバル入力であり、ワークスペースのクエリとみなされ、ルーチンはすべての `work` 配列の最小かつ最適なサイズだけを計算する。各値は該当する `work` 配列の最初のエントリとして返され、[pxerbla](#) はエラー・メッセージを生成しない。

### 出力パラメータ

<i>c</i>	積 $Q \text{ sub}(C)$ 、 $Q' \text{ sub}(C)$ 、 $\text{sub}(C) Q'$ 、または $\text{sub}(C) Q$ のいずれかによって上書きされる。
<i>work</i> (1)	終了時に、 <i>work</i> (1) には、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が格納される。
<i>info</i>	(グローバル) INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> < 0 の場合： <i>i</i> 番目の引数が配列で、 <i>j</i> 番目の値が不正だった場合、 <i>info</i> = - <i>i</i> 。

## p?stebz

二分法を用いて、対称三重対角行列の固有値を計算する。

### 構文

```
call psstebz( ictxt, range, order, n, vl, vu, il, iu, abstol, d, e, m,
              nsplit, w, iblock, isplit, work, iwork, liwork, info)
call pdstebz( ictxt, range, order, n, vl, vu, il, iu, abstol, d, e, m,
              nsplit, w, iblock, isplit, work, iwork, liwork, info)
```

### 説明

このルーチンは、二分法を用いて、対称三重対角行列の固有値を計算する。計算される固定値は、すべての固定値、区間

[*vl vu*] 内のすべての固定値、またはインデックス *il* ~ *iu* の固定値である。*work* の静的分割は [p?stebz](#) のはじめに完了する。これにより、各プロセスは、ほぼ同数の固有値を見つける。

### 入力パラメータ

<i>ictxt</i>	(グローバル) INTEGER。BLACS コンテキスト・ハンドル。
<i>range</i>	(グローバル) CHARACTER。 'A'、'V'、または 'I' のいずれかでなければならない。 <i>range</i> = 'A' の場合、固有値をすべて計算する。 <i>range</i> = 'V' の場合、次の区間内の固有値を計算する。

	<p>区間 <math>[v_l \ v_u]</math></p> <p><math>range = 'I'</math> の場合、インデックス <math>i_l</math> から <math>i_u</math> の固有値を計算する。 (グローバル) CHARACTER。'B' または 'E' でなければならない。 <math>order = 'B'</math> の場合、一連の固有値は、分解された各ブロック内で最小のものから昇順に配置される。 <math>order = 'E'</math> の場合、固有値が行列全体で最小のものから昇順に配置される。</p>
<i>order</i>	
<i>n</i>	(グローバル) INTEGER。三重対角行列 $T$ の次数 ( $n \geq 0$ )。
<i>v_l, v_u</i>	<p>(グローバル)</p> <p>REAL (psstebz の場合)</p> <p>DOUBLE PRECISION (pdstebz の場合)。</p> <p><math>range = 'V'</math> の場合、区間 <math>[v_l \ v_u]</math> の固有値の下限値と上限値を計算する。</p> <p><math>range = 'A'</math> または 'I' の場合、<math>v_l</math> と <math>v_u</math> は参照されない。</p>
<i>i_l, i_u</i>	<p>(グローバル)</p> <p>INTEGER。次の制約がある。<math>1 \leq i_l \leq i_u \leq n</math>。</p> <p><math>range = 'I'</math> の場合、(固有値が昇順であると仮定して) 最も小さな固有値のインデックスは <math>i_l</math> に、最も大きな固有値のインデックスは <math>i_u</math> に返されなければならない。 <math>i_l</math> は 1 以上でなければならない。<math>i_u</math> は <math>i_l</math> 以上で <math>n</math> 以下でなければならない。</p> <p><math>range = 'A'</math> または 'V' の場合、<math>i_l</math> と <math>i_u</math> は参照されない。</p>
<i>abstol</i>	<p>(グローバル)</p> <p>REAL (psstebz の場合)</p> <p>DOUBLE PRECISION (pdstebz の場合)。</p> <p>各固有値に対する絶対許容値は、必ず指定しなければならない。固有値 (または集積値) は、幅 <math>abstol</math> の区間内に存在している場合に収束しているものとみなされる。<math>abstol \leq 0</math> の場合、許容値は <math>ulp \ T\ </math> (<math>ulp</math> はマシン精度で <math>\ T\ </math> は <math>T</math> の 1- ノルム) になる。</p> <p>固有値が最も正確に計算されるのは、<math>abstol</math> がゼロではなく、アンダーフローのしきい値 <math>slamch('U')</math> に設定されたときである。 固有ベクトルを後から逆反復法 (<a href="#">p?stein</a>) で使用する場合、<math>abstol</math> を <math>2 * p * slamch('S')</math> に設定する必要がある。</p>
<i>d</i>	<p>(グローバル)</p> <p>REAL (psstebz の場合)</p> <p>DOUBLE PRECISION (pdstebz の場合)。</p> <p>配列、次元は (<math>n</math>)。</p>

三重対角行列  $T$  の  $n$  個の対角成分を格納する。アンダーフローを防ぐために、行列を、その最大エントリが  $\text{overflow}^{(1/2)} * \text{underflow}^{(1/4)}$  よりも絶対値において超えないようにスケールしなければならない。最も精度を上げるには、行列を上記条件よりもはるかに小さくスケールしてはならない。

$e$	<p>(グローバル)</p> <p>REAL (psstebz の場合)</p> <p>DOUBLE PRECISION (pdstebz の場合)。</p> <p>配列、次元は <math>(n-1)</math>。</p> <p>三重対角行列 <math>T</math> の <math>n-1</math> 個の非対角成分を格納する。アンダーフローを防ぐために、行列を、その最大エントリが <math>\text{overflow}^{(1/2)} * \text{underflow}^{(1/4)}</math> よりも絶対値において超えないようにスケールしなければならない。最も精度を上げるには、行列を上記条件よりもはるかに小さくスケールしてはならない。</p>
$work$	<p>(グローバル)</p> <p>REAL (psstebz の場合)</p> <p>DOUBLE PRECISION (pdstebz の場合)。</p> <p>配列、次元は <math>\max(5n, 7)</math>。ワークスペース配列。</p>
$lwork$	<p>(ローカル) INTEGER。</p> <p><math>work</math> 配列のサイズは <math>\max(5n, 7)</math> 以上でなければならない。</p> <p><math>lwork = -1</math> の場合、<math>lwork</math> はグローバル入力であり、ワークスペースのクエリとみなされ、ルーチンはすべての <math>work</math> 配列の最小かつ最適なサイズだけを計算する。各値は該当する <math>work</math> 配列の最初のエントリとして返され、<a href="#">pxerbla</a> はエラー・メッセージを生成しない。</p>
$iwork$	<p>(ローカル) INTEGER。</p> <p>配列、次元は <math>\max(4n, 14)</math>。ワークスペース配列。</p>
$liwork$	<p>(ローカル) INTEGER。</p> <p><math>iwork</math> 配列のサイズは <math>\max(4n, 14, \text{NPROCS})</math> 以上でなければならない。</p> <p><math>liwork = -1</math> の場合、<math>liwork</math> はグローバル入力であり、ワークスペースのクエリとみなされ、ルーチンはすべての配列の最小かつ最適なサイズだけを計算する。各値は該当する <math>work</math> 配列の最初のエントリとして返され、<a href="#">pxerbla</a> はエラー・メッセージを生成しない。</p>

## 出力パラメータ

$m$	<p>(グローバル) INTEGER。</p> <p>見つかった固有値の実際の個数。 <math>0 \leq m \leq n</math></p>
-----	---

<i>nsplit</i>	(グローバル) INTEGER。 <i>T</i> で見つかった対角ブロックの個数。 $1 \leq nsplit \leq n$
<i>w</i>	(グローバル) REAL (psstebz の場合) DOUBLE PRECISION (pdstebz の場合)。 配列、次元は ( <i>n</i> )。 終了時に、 <i>w</i> の最初の <i>m</i> 個の成分に、すべてのプロセスの固有値が格納される。
<i>iblock</i>	(グローバル) INTEGER。 配列、次元は ( <i>n</i> )。

*e(j)* がゼロまたは小さい行 / 列 *j* では、行列 *T* はブロック対角行列に分割するとみなされる。終了時に、*iblock(i)* は (1 からブロック数まで) どのブロックに固有値 *w(i)* が属するかを示す。



**注:** 二分法で一部またはすべての固有値が収束しない (理論上不可能な) イベントでは、*info* は 1 に設定される。収束しなかったイベントは、負のブロック番号で識別できる。

<i>isplit</i>	(グローバル) INTEGER。配列、次元は ( <i>n</i> )。 <i>T</i> を部分行列に分割した分割点が格納される。最初の部分行列は 1 から <i>isplit</i> (1) の行 / 列で構成され、2 番目の部分行列は <i>isplit</i> (1)+1 ~ <i>isplit</i> (2) の行 / 列で構成され、以下同様である。 <i>nsplit</i> 番目は <i>isplit</i> ( <i>nsplit</i> -1)+1 ~ <i>isplit</i> ( <i>nsplit</i> )= <i>n</i> の行 / 列で構成される (最初の <i>nsplit</i> 個の成分のみが使用される。しかし、 <i>nsplit</i> の値はわからないため、 <i>n</i> ワードを <i>isplit</i> 用に予約しておかなければならない)。
<i>info</i>	(グローバル) INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> < 0 の場合、 <i>info</i> = - <i>i</i> は <i>i</i> 番目の引数の値が不正だったことを示す。 <i>info</i> > 0 の場合、一部またはすべての固有値が収束に失敗したか計算できなかったことを示す。 <i>info</i> = 1 の場合、二分法で一部の固有値が収束に失敗したことを示す。これらの固有値には、負のブロック番号が付けられる。固有値は、絶対的および相対的な許容値ほど正確ではない。



$info=2$  の場合、出力された固有値の番号と要求した番号が一致していないことを示す。

$info=3$  の場合、 $range='i'$  で最初に使用された Gershgorin 区間が正しくないことを示す。固有値は計算されない。マシンの浮動小数点演算に問題がある可能性がある。 $fudge$  パラメータの値を大きくして再コンパイルし、再度実行する。

## p?stein

逆反復法を用いて、三重対角行列の固有ベクトル計算する。

### 構文

```
call psstein( n, d, e, m, w, iblock, isplit, orfac, z, iz, jz, descz,
              work, lwork, iwork, liwork, ifail, iclustr, gap, info)
call pdstein( n, d, e, m, w, iblock, isplit, orfac, z, iz, jz, descz,
              work, lwork, iwork, liwork, ifail, iclustr, gap, info)
call pcstein( n, d, e, m, w, iblock, isplit, orfac, z, iz, jz, descz,
              work, lwork, iwork, liwork, ifail, iclustr, gap, info)
call pzstein( n, d, e, m, w, iblock, isplit, orfac, z, iz, jz, descz,
              work, lwork, iwork, liwork, ifail, iclustr, gap, info)
```

### 説明

このルーチンは、逆反復法を用いて、対称三重対角行列  $T$  の指定された固有値に対応する固有ベクトルを計算する。 [p?stein](#) は、異なるプロセス上にあるベクトルを直交しない。直交化の範囲は、入力パラメータ  $lwork$  によって制御される。直交する固有ベクトルは、同じプロセスによって計算される。 [p?stein](#) は、プロセス間の  $work$  の割り当てを決定してから、個々のプロセスで [sstein2](#) (LAPACK ルーチンの修正版) を呼び出す。ワークスペースの割り当てが十分でない場合、要求した直交化は行われない。



**注:** 得られた固有ベクトルが直交でない場合、 $lwork$  の値を増やしてコードを再度実行する。

$p = \text{NPROW} * \text{NPCOL}$  はプロセスの総数。

## 入力パラメータ

<i>n</i>	(グローバル) INTEGER。 行列 <i>T</i> の次数 ( $n \geq 0$ )。
<i>m</i>	(グローバル) INTEGER。 返すべき固有ベクトルの個数。
<i>d</i> , <i>e</i> , <i>w</i>	(グローバル) REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 配列: <i>d</i> (*) には、 <i>T</i> の対角成分を格納する。 次元は ( <i>n</i> )。  <i>e</i> (*) には、 <i>T</i> の非対角成分を格納する。 次元は ( <i>n</i> -1)。  <i>w</i> (*) には、分割ブロックによってグループ化されたすべての固有値を格納する。固有値は、ブロック内で最も小さなものから最も大きなものまで提供される (order = 'B' の場合の <a href="#">p?stebz</a> の出力配列 <i>w</i> がここで要求される)。配列は、すべてのプロセスで複製する必要がある。 次元は ( <i>m</i> )。
<i>iblock</i>	(グローバル) INTEGER。 配列、次元は ( <i>n</i> )。 <i>w</i> の対応する固有値に関連する部分行列インデックス。一番上から最初の部分行列に属する固有値は 1、2 番目の部分行列に属する固有値は 2、以下同様 ( <a href="#">p?stebz</a> の出力配列 <i>iblock</i> がここで要求される)。
<i>isplit</i>	(グローバル) INTEGER。 配列、次元は ( <i>n</i> )。 <i>T</i> を部分行列に分割した分割点。最初の部分行列は 1 から <i>isplit</i> (1) の行 / 列で構成され、2 番目の部分行列は <i>isplit</i> (1)+1 ~ <i>isplit</i> (2) の行 / 列で構成され、以下同様である。 <i>nsplit</i> 番目は <i>isplit</i> ( <i>nsplit</i> -1)+1 ~ <i>isplit</i> ( <i>nsplit</i> )= <i>n</i> の行 / 列で構成される ( <a href="#">p?stebz</a> の出力配列 <i>isplit</i> がここで要求される)。
<i>orfac</i>	(グローバル) REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 <i>orfac</i> は、直交化される固有ベクトルを指定する。互いの <i>orfac</i> *   <i>T</i>    内にある固有値に対応する固有ベクトルは直交化される。しかし、ワークスペースが十分でない場合 ( <i>lwork</i> を参照)、この許容値は、すべての固有ベクトルが 1 つのプロ

セスで格納できるようになるまで下げられる。 *orfac* がゼロの場合、直交化は行われない。 *orfac* が負の場合、デフォルト値の  $10^3$  が使用される。 *orfac* はすべての処理で同一でなければならない。

<i>iz, jz</i>	(グローバル) INTEGER。 それぞれ、部分行列 <i>Z</i> の最初の行と最初の列を示す、グローバル配列 <i>z</i> の行インデックスと列インデックス。
<i>descz</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。分散行列 <i>Z</i> の配列ディスクリプタ。
<i>work</i>	(ローカル)。 REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。ワークスペース配列、次元は ( <i>lwork</i> )。
<i>lwork</i>	(ローカル) INTEGER。 <i>lwork</i> は、直交化の範囲を制御する。各プロセスで割り当てられる格納形式に対する固有ベクトルの数は次のとおりである。  $nvec = \text{floor}((lwork - \max(5*n, np00*mq00))/n)$ サイズ $nvec - \text{ceil}(m/p) + 1$ の固有値クラスタに対応する固有ベクトルは、直交することが保証される (直交性は、?stein2 で得られたものに似ている)。



**注:** *lwork* は、 $\max(5*n, np00*mq00) + \text{ceil}(m/p)*n$  以上でなければならない。また、すべてのプロセスで入力値が同じでなければならない。

重要なのは、異なるプロセスにおける *lwork* 入力の最小値である。

*lwork* = -1 の場合、*lwork* はグローバル入力であり、ワークスペースのクエリとみなされ、ルーチンはすべての *work* 配列の最小かつ最適なサイズだけを計算する。各値は該当する *work* 配列の最初のエントリとして返され、[pxerbla](#) はエラー・メッセージを生成しない。

<i>iwork</i>	(ローカル) INTEGER。 ワークスペース配列、次元は ( $3n+p+1$ )。
<i>liwork</i>	(ローカル) INTEGER。配列 <i>iwork</i> のサイズ。 $3*n + p + 1$ 以上でなければならない。

*liwork* = -1 の場合、*liwork* はグローバル入力であり、ワークスペースのクエリとみなされ、ルーチンはすべての配列の最小かつ最適なサイズだけを計算する。各値は該当する *work* 配列の最初のエントリとして返され、[pxerbla](#) はエラー・メッセージを生成しない。

## 出力パラメータ

<i>z</i>	(ローカル) REAL (psstein の場合) DOUBLE PRECISION (pdstein の場合) COMPLEX (pcstein の場合) DOUBLE COMPLEX (pzstein の場合)。 配列、次元は (descz(dlen_), n/NPCOL + NB)。z には、指定された固有値と関連する計算された固有ベクトルが格納される。収束に失敗したすべてのベクトルは、MAXIT 回反復した後、現在の反復に設定される (?stein2 を参照)。出力時に、z は、ブロック・サイクリック形式で <i>p</i> 個のプロセスに分散される。
<i>work(1)</i>	終了時に、 <i>work(1)</i> には、ユーザが要求する直交化を保証するワークスペース ( <i>lwork</i> ) の下限が格納される ( <i>orfac</i> を参照)。必要な最小ワークスペースを過大評価する場合がある点に注意する。
<i>iwork</i>	終了時に、 <i>iwork(1)</i> には、要求されたワークスペースの量が整数で格納される。 終了時に、 <i>iwork(2)</i> ~ <i>iwork(p+2)</i> には、各プロセスによって計算された固有ベクトルが格納される。プロセス <i>i</i> は、インデックス <i>iwork(i+2)+1</i> ~ <i>iwork(i+3)</i> の固有ベクトルを計算する。
<i>ifail</i>	(グローバル) INTEGER。配列、次元は ( <i>m</i> )。 通常終了時は、 <i>ifail</i> のすべての成分はゼロである。MAXIT 回反復した後、1 つ以上の固有ベクトルが収束に失敗した場合 (?stein を参照)、 <i>info</i> > 0 が返される。mod( <i>info</i> , <i>m</i> +1) > 0 で、 <i>i</i> = 1 から mod( <i>info</i> , <i>m</i> +1) の場合、固有値 <i>w</i> ( <i>ifail(i)</i> ) に対応する固有ベクトルは収束に失敗した ( <i>w</i> は出力時に固有値の配列を参照する)。
<i>iclustr</i>	(グローバル) INTEGER。配列、次元は (2* <i>p</i> )。 この出力配列には、ワークスペースが十分でなかったために直交できなかった固有値のクラスタに対応する固有ベクトルのインデックスが格納される ( <i>lwork</i> , <i>orfac</i> および <i>info</i> を参照)。インデックス <i>iclustr(2*I-1)</i> から <i>iclustr(2*I)</i> 、 <i>i</i> = 1 から <i>info/(m+1)</i> の固有値のクラスタに対応する固有ベクトルは、ワークスペースの不足により直交できなかった。したがって、これらのクラスタに対応する固有ベクトルは直交しない。 <i>iclustr</i> は、ゼロで終了する配列である。 ( <i>iclustr(2*k).ne.0.and. iclustr(2*k+1).eq.0</i> ) ( <i>k</i> がクラスタ数の場合)。

*gap* (グローバル)  
 REAL (単精度の場合)  
 DOUBLE PRECISION (倍精度の場合)。  
 この出力配列には、固有ベクトルが直交できなかった固有値間のギャップが格納される。この配列の *info/m* 出力値は、配列 *iclustr* によって示された *info/(m+1)* クラスタに対応する。その結果、*i* 番目のクラスタに対応する固有ベクトルの内積は、 $(O(n)*macheps)/gap(i)$  と同程度になる。

*info* (グローバル) INTEGER。  
*info* = 0 の場合、実行は正常に終了したことを示す。

*info* < 0 の場合：  
*i* 番目の引数が配列で、*j* 番目の値が不正だった場合、  
*info* =  $-(i*100+j)$ 。*i* 番目の引数がスカラーで値が不正だった場合、  
*info* = -*i*。

*info* < 0 の場合：  
*info* = -*i* の場合、*i* 番目の引数の値が不正だったことを示す。

*info* > 0 の場合：  
 $\text{mod}(\text{info}, m+1) = i$  の場合、*i* 固有ベクトルは MAXIT 回の反復で収束に失敗した。インデックスは、配列 *ifail* に格納される。  
 $\text{info}/(m+1) = i$  の場合、固有値の *i* クラスタに対応する固有ベクトルが、ワークスペースが十分でなかったために直交できなかった。クラスタのインデックスは、配列 *iclustr* に格納される。

## 非対称固有値問題

このセクションでは、非対称固有値問題の解決、一般行列の Schur 因子分解の計算、およびそれらに関連する各種の計算タスクを実行するための ScaLAPACK ルーチンについて説明する。

ScaLAPACK を使って非対称固有値問題を解くには、通常、行列を上 Hessenberg 形式に縮退させた後、得られた Hessenberg 行列を使って固有値問題を解く必要がある。

表 6-5 に示されている ScaLAPACK ルーチンを利用すると、直交（またはユニタリ）の相似変換  $A = QHQ^H$  を使って、該当行列を上 Hessenberg 形式に縮退させることができる。また、Hessenberg 行列を使って固有値問題を解き、縮退後に行列を乗算することもできる。

表 6-5 非対称固有値問題を解くための計算ルーチン

機能	一般行列	直交 / ユニタリ行列	Hessenberg 行列
Hessenberg 形式 $A = QHQ^H$ に縮退させる	<a href="#">p?gehrd</a>		
縮退後に行列を掛ける		<a href="#">p?ormhr</a> / <a href="#">p?unmhr</a>	
固有値と Schur 因子分解を求める			<a href="#">p?lahqr</a>

## p?gehrd

一般行列を上 Hessenberg 形式に縮退させる。

### 構文

```
call psgehrd( n, ilo, ihi, a, ia, ja, desca, tau, work, lwork,  
             info )  
call pdgehrd( n, ilo, ihi, a, ia, ja, desca, tau, work, lwork,  
             info )  
call pcgehrd( n, ilo, ihi, a, ia, ja, desca, tau, work, lwork,  
             info )  
call pzgehrd( n, ilo, ihi, a, ia, ja, desca, tau, work, lwork,  
             info )
```

### 説明

このルーチンは、直交またはユニタリの相似変換を用いて、実 / 複素一般分散行列  $\text{sub}(A)$  を上 Hessenberg 形式  $H$  に縮退させる。

$$Q' \text{sub}(A) Q = H$$

ここで、 $\text{sub}(A) = A(\text{ia}+n-1:\text{ia}+n-1, \text{ja}+n-1:\text{ja}+n-1)$ 。

## 入力パラメータ

<i>n</i>	(グローバル) INTEGER。 分散行列 <i>sub(A)</i> の次数 ( $n \geq 0$ )。
<i>ilo, ihi</i>	(グローバル) INTEGER。 <i>sub(A)</i> はすでに、行 <i>ia:ia+ilo-2</i> と <i>ia+ihi:ia+n-1</i> および列 <i>ja:ja+ilo-2</i> と <i>ja+ihi:ja+n-1</i> で上三角になっていると仮定する。 (次の「アプリケーション・ノート」を参照)。 $n > 0$ の場合、 $1 \leq ilo \leq ihi \leq n$ 。それ以外の場合、 $ilo = 1$ 、 $ihi = n$ を 設定する。
<i>a</i>	(ローカル) REAL (psgehrd の場合) DOUBLE PRECISION (pdgehrd の場合) COMPLEX (pcgehrd の場合) DOUBLE COMPLEX (pzgehrd の場合)。 ローカルメモリにある、次元 ( <i>lld_a</i> , <i>LOCc(ja+n-1)</i> ) の配列へのポイン タ。呼び出し時に、この配列は、因子分解する $n \times n$ の一般分散行列 <i>sub(A)</i> のローカル部分を含む。
<i>ia, ja</i>	(グローバル) INTEGER。 それぞれ、部分行列 <i>A</i> の最初の行と最初の列を示す、グローバル配列 <i>a</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。 分散行列 <i>A</i> の配列ディスクリプタ。
<i>work</i>	(ローカル) REAL (psgehrd の場合) DOUBLE PRECISION (pdgehrd の場合) COMPLEX (pcgehrd の場合) DOUBLE COMPLEX (pzgehrd の場合)。 次元 <i>lwork</i> のワークスペース配列。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。配列 <i>work</i> の次元。 ローカル入力で、  $lwork \geq NB * NB + NB * \max(ihip+1, ihlp+inlq)$ でなければならない。 ここで、 $NB = mb\_a = nb\_a$ 、 $iroffa = \text{mod}(ia-1, NB)$ 、 $icoffa = \text{mod}(ja-1, NB)$ 、 $ioff = \text{mod}(ia+ilo-2, NB)$ 、 $iarow = \text{indxg2p}(ia, NB, MYROW, rsrc\_a, NPROW)$ 、

```

ihip = numroc(ihi+ioffa, NB, MYROW, iarow, NPROW)、
ilrow = indxg2p(ia+ilo-1, NB, MYROW, rsrc_a, NPROW)、
ihlp = numroc(ihi-ilo+ioffa+1, NB, MYROW, ilrow, NPROW)、
ilcol = indxg2p(ja+ilo-1, NB, MYCOL, csrc_a, NPCOL)、
inlq = numroc(n-ilo+ioffa+1, NB, MYCOL, ilcol, NPCOL)

```

indxg2p および numroc は、ScaLAPACK ツール関数である。MYROW、MYCOL、NPROW、および NPCOL は、サブルーチン blacs\_gridinfo を呼び出して決定できる。

lwork = -1 の場合、lwork はグローバル入力であり、ワークスペースのクエリとみなされ、ルーチンはすべての work 配列の最小かつ最適なサイズだけを計算する。各値は該当する work 配列の最初のエン트리として返され、[pxerbla](#) はエラー・メッセージを生成しない。

## 出力パラメータ

a	終了時に、sub(A) の上三角と第 1 の劣対角は、上 Hessenberg 行列 <i>H</i> および第 1 の劣対角よりも下の成分によって上書きされ、配列 tau とともに、基本リフレクタの積として直交/ユニタリの行列 <i>Q</i> を表す。 (次の「アプリケーション・ノート」を参照)。
tau	(ローカル)。 REAL (psgehrd の場合) DOUBLE PRECISION (pdgehrd の場合) COMPLEX (pcgehrd の場合) DOUBLE COMPLEX (pzgehrd の場合)。 配列、次元は max(ja+n-2) 以上。 基本リフレクタのスカラー係数。(次の「アプリケーション・ノート」を参照)。tau の成分 ja:ja+ilo-2 と ja+ihi:ja+n-2 はゼロに設定される。tau は、分散行列 <i>A</i> に関連付けられる。
work(1)	終了時に、work(1) には、最適なパフォーマンスを得るために必要な lwork の最小値が格納される。
info	(グローバル) INTEGER。 info = 0 の場合、実行は正常に終了したことを示す。 info < 0 の場合: <i>i</i> 番目の引数が配列で、 <i>j</i> 番目の値が不正だった場合、 info = -(i*100+j)。i 番目の引数がスカラーで値が不正だった場合、 info = -i。



## アプリケーション・ノート

行列  $Q$  は、 $(ihi-ilo)$  個の基本リフレクタの積として表現される。

$$Q = H(ilo) H(ilo+1) \dots H(ihi-1)$$

それぞれの  $H(i)$  は次のような形式を持つ。

$$H(i) = I - \tau * v * v'$$

ここで、 $\tau$  は実 / 複素スカラ、 $v$  は実 / 複素ベクトルで  $v(1:i) = 0$ 、 $v(i+1) = 1$ 、および  $v(ihi+1:n) = 0$ 。終了時に、 $v(i+2:ihi)$  は  $a(ia+ilo+i:ia+ihi-1, ja+ilo+i-2)$  に、 $\tau$  は  $\tau(ja+ilo+i-2)$  に格納される。 $a(ia:ia+n-1, ja:ja+n-1)$  の内容を次に示す ( $n=7$ 、 $ilo=2$  および  $ihi=6$  の場合)。

呼び出し時

$$\begin{bmatrix} a & a & a & a & a & a & a \\ & a & a & a & a & a & a \\ & & a & a & a & a & a \\ & & & a & a & a & a \\ & & & & a & a & a \\ & & & & & a & a \\ & & & & & & a \end{bmatrix}$$

終了時

$$\begin{bmatrix} a & a & h & h & h & h & a \\ & a & h & h & h & h & a \\ & h & h & h & h & h & h \\ v2 & h & h & h & h & h & h \\ v2 & v3 & h & h & h & h & h \\ v2 & v3 & v4 & h & h & h & h \\ & & & & & & a \end{bmatrix}$$

$a$  は元の行列  $\text{sub}(A)$  の成分、 $H$  は上 Hessenberg 行列  $H$  の変更された成分、 $v_i$  は  $H(ja+ilo+i-2)$  を定義するベクトルの成分を表す。

## p?ormhr

一般行列に p?gehrd で求めた *Hessenberg* 形式に縮退した直交変換行列を掛ける。

### 構文

```
call psormhr( side, trans, m, n, ilo, ihi, a, ia, ja, desca, tau, c, ic,
              jc, descc, work, lwork, info )
call pdormhr( side, trans, m, n, ilo, ihi, a, ia, ja, desca, tau, c, ic,
              jc, descc, work, lwork, info )
```

### 説明

このルーチンは、 $m \times n$  の一般実分散行列  $\text{sub}(C) = C(ic:ic+m-1, jc:jc+n-1)$  を以下により上書きする。

	$side = 'L'$	$side = 'R'$
$trans = 'N':$	$Q \text{ sub}(C)$	$\text{sub}(C) Q$
$trans = 'T':$	$Q^T \text{ sub}(C)$	$\text{sub}(C) Q^T$

ここで、 $Q$  は次数  $nq$  の実直交分散行列で、 $nq = m$  ( $side = 'L'$  の場合)、 $nq = n$  ( $side = 'R'$  の場合) である。 $Q$  は、[p?gehrd](#) によって返される、 $ihi-ilo$  個の基本リフレクタの積として定義される。

$Q = H(ilo) H(ilo+1) \dots H(ihi-1)$

### 入力パラメータ

<b>side</b>	(グローバル) CHARACTER ='L' の場合、 $Q$ または $Q^T$ は左側から適用される。 ='R' の場合、 $Q$ または $Q^T$ は右側から適用される。
<b>trans</b>	(グローバル) CHARACTER ='N' の場合、 $Q$ が適用される (転置なし)。 ='T' の場合、 $Q^T$ が適用される (転置)。
<b>m</b>	(グローバル) INTEGER。 分散行列 $\text{sub}(C)$ の行数 ( $m \geq 0$ )。
<b>n</b>	(グローバル) INTEGER。 分散行列 $\text{sub}(C)$ の列数 ( $n \geq 0$ )。

<i>ilo, ihi</i>	<p>(グローバル) INTEGER。  <i>ilo</i> と <i>ihi</i> は <code>p?gehrd</code> の以前の呼び出しで指定された値と同じ値でなければならない。<i>Q</i> は、分散部分行列 <math>Q(ia+ilo:ia+ihi-1, ia+ilo:ja+ihi-1)</math> を除いてユニタリ行列と等しい。</p> <p><i>side</i> = 'L' の場合、<math>1 \leq ilo \leq ihi \leq \max(1, m)</math>。  <i>side</i> = 'R' の場合、<math>1 \leq ilo \leq ihi \leq \max(1, n)</math>。  <i>ilo</i> と <i>ihi</i> は相対インデックス。</p>
<i>a</i>	<p>(ローカル)  REAL (psormhr の場合)  DOUBLE PRECISION (pdormhr の場合)  ローカルメモリにある、次元 (<i>lld_a</i>, <math>LOCc(ja+m-1)</math>) (<i>side</i>='L' の場合) または (<i>lld_a</i>, <math>LOCc(ja+n-1)</math>) (<i>side</i>='R' の場合) の配列へのポインタ。  <a href="#">p?gehrd</a> によって返される、基本リフレクタを定義する一連のベクトルを含む。</p>
<i>ia, ja</i>	<p>(グローバル) INTEGER。  それぞれ、部分行列 <i>A</i> の最初の行と最初の列を示す、グローバル配列 <i>a</i> の行インデックスと列インデックス。</p>
<i>desca</i>	<p>(グローバルおよびローカル) INTEGER。配列、次元は (<i>dlen_</i>)。  分散行列 <i>A</i> の配列ディスクリプタ。</p>
<i>tau</i>	<p>(ローカル)  REAL (psormhr の場合)  DOUBLE PRECISION (pdormhr の場合)  配列、次元は <math>LOCc(ja+m-2)</math> (<i>side</i>='L' の場合) または <math>LOCc(ja+n-2)</math> (<i>side</i>='R' の場合)。  <a href="#">p?gehrd</a> によって返される、基本リフレクタ <math>H(j)</math> のスカラ係数 <math>\tau(j)</math> を含む。<i>tau</i> は、分散行列 <i>A</i> に関連付けられる。</p>
<i>c</i>	<p>(ローカル)  REAL (psormhr の場合)  DOUBLE PRECISION (pdormhr の場合)  ローカルメモリにある、次元 (<i>lld_c</i>, <math>LOCc(jc+n-1)</math>) の配列へのポインタ。分散行列 <i>sub(C)</i> のローカル部分を含む。</p>
<i>ic, jc</i>	<p>(グローバル) INTEGER。  それぞれ、部分行列 <i>C</i> の最初の行と最初の列を示す、グローバル配列 <i>c</i> の行インデックスと列インデックス。</p>
<i>descc</i>	<p>(グローバルおよびローカル) INTEGER。配列、次元は (<i>dlen_</i>)。  分散行列 <i>C</i> の配列ディスクリプタ。</p>

*work* (ローカル)  
 REAL (psormhr の場合)  
 DOUBLE PRECISION (pdormhr の場合)  
 ワークスペース配列、次元は *lwork*。

*lwork* (ローカルまたはグローバル) INTEGER。  
 配列 *work* の次元。  
*lwork* は  
 $iaa = ia + ilo; jaa = ja + ilo - 1$  以上でなければならない。  
*side* = 'L' の場合、  
 $mi = ihi - ilo; ni = n; icc = ic + ilo; jcc = jc; lwork \geq \max((nb\_a * (nb\_a - 1)) / 2, (nqc0 + mpc0) * nb\_a) + nb\_a * nb\_a$   
*side* = 'R' の場合、  
 $mi = m; ni = ihi - ilo; icc = ic; jcc = jc + ilo; lwork \geq \max((nb\_a * (nb\_a - 1)) / 2, (nqc0 + \max(npa0 + \text{numroc}(\text{numroc}(ni + icoffc, nb\_a, 0, 0, NPCOL), nb\_a, 0, 0, lcmq), mpc0)) * nb\_a) + nb\_a * nb\_a$   
 でなければならない。  
 ここで、 $lcmq = lcm / NPCOL$  で  $lcm = ilcm(NPROW, NPCOL)$ ,  
 $iroffa = \text{mod}(iaa - 1, mb\_a)$ ,  
 $icoffa = \text{mod}(jaa - 1, nb\_a)$ ,  
 $iarow = \text{indxg2p}(iaa, mb\_a, MYROW, rsrc\_a, NPROW)$ ,  
 $npa0 = \text{numroc}(ni + iroffa, mb\_a, MYROW, iarow, NPROW)$ ,  
 $iroffc = \text{mod}(icc - 1, mb\_c)$ ,  
 $icoffc = \text{mod}(jcc - 1, nb\_c)$ ,  
 $icrow = \text{indxg2p}(icc, mb\_c, MYROW, rsrc\_c, NPROW)$ ,  
 $iccol = \text{indxg2p}(jcc, nb\_c, MYCOL, csrc\_c, NPCOL)$ ,  
 $mpc0 = \text{numroc}(mi + iroffc, mb\_c, MYROW, icrow, NPROW)$ ,  
 $nqc0 = \text{numroc}(ni + icoffc, nb\_c, MYCOL, iccol, NPCOL)$   
 $ilcm$ 、 $\text{indxg2p}$  および  $\text{numroc}$  は、ScaLAPACK ツール関数である。  
 MYROW、MYCOL、NPROW、および NPCOL は、サブルーチン `blacs_gridinfo` を呼び出して決定できる。  
*lwork* = -1 の場合、*lwork* はグローバル入力であり、ワークスペースのクエリとみなされ、ルーチンはすべての *work* 配列の最小かつ最適なサイズだけを計算する。各値は該当する *work* 配列の最初のエン트리として返され、[pxerbla](#) はエラー・メッセージを生成しない。

## 出力パラメータ

<code>c</code>	$\text{sub}(C)$ は、 $Q \text{sub}(C)$ 、 $Q' \text{sub}(C)$ 、 $\text{sub}(C)Q'$ 、または $\text{sub}(C)Q$ のいずれかによって上書きされる。
<code>work(1)</code>	終了時に、 <code>work(1)</code> には、最適なパフォーマンスを得るために必要な <code>lwork</code> の最小値が格納される。
<code>info</code>	(グローバル) INTEGER。 <code>info = 0</code> の場合、実行は正常に終了したことを示す。 <code>info &lt; 0</code> の場合： $i$ 番目の引数が配列で、 $j$ 番目の値が不正だった場合、 <code>info</code> = $-(i*100+j)$ 。 $i$ 番目の引数がスカラーで値が不正だった場合、 <code>info</code> = $-i$ 。

## p?unmhr

一般行列に p?gehrd で求めた *Hessenberg* 形式に縮退したユニタリ変換行列を掛ける。

## 構文

```
call pcunmhr( side, trans, m, n, ilo, ihi, a, ia, ja, desca, tau, c, ic,
              jc, descc, work, lwork, info )
call pzunmhr( side, trans, m, n, ilo, ihi, a, ia, ja, desca, tau, c, ic,
              jc, descc, work, lwork, info )
```

## 説明

このルーチンは、 $m \times n$  の一般複素分散行列  $\text{sub}(C) = C(ic:ic+m-1, jc:jc+n-1)$  を以下により上書きする。

`side = 'L'`                      `side = 'R'`

`trans = 'N':`     $Q \text{sub}(C)$                        $\text{sub}(C)Q$

`trans = 'C':`     $Q^H \text{sub}(C)$                        $\text{sub}(C)Q^H$

ここで、 $Q$  は、次数  $nq$  の複素ユニタリ分散行列で、 $nq = m$  (`side = 'L'` の場合)、 $nq = n$  (`side = 'R'` の場合) である。 $Q$  は、[p?gehrd](#) によって返される、 $ihi-ilo$  個の基本リフレクタの積として定義される。

$Q = H(ilo)H(ilo+1)\dots H(ihi-1)$

## 入力パラメータ

<i>side</i>	(グローバル) CHARACTER ='L' の場合、 $Q$ または $Q^H$ は左側から適用される。 ='R' の場合、 $Q$ または $Q^H$ は右側から適用される。
<i>trans</i>	(グローバル) CHARACTER ='N' の場合、 $Q$ が適用される (転置なし)。 ='C' の場合、 $Q^H$ が適用される (共役転置)。
<i>m</i>	(グローバル) INTEGER。 分散部分行列 $\text{sub}(C)$ の行数 ( $m \geq 0$ )。
<i>n</i>	(グローバル) INTEGER。 分散部分行列 $\text{sub}(C)$ の列数 ( $n \geq 0$ )。
<i>ilo, ihi</i>	(グローバル) INTEGER。 これらの値は、それぞれ <a href="#">p?gehrd</a> に指定した <i>ilo</i> および <i>ihi</i> と同じでなければならない。 $Q$ は、分散部分行列 $Q(ia+ilo:ia+ihi-1, ia+ilo:ja+ihi-1)$ を除いてユニタリ行列と等しい。 <i>side</i> ='L' の場合、 $1 \leq ilo \leq ihi \leq \max(1, m)$ 。 <i>side</i> ='R' の場合、 $1 \leq ilo \leq ihi \leq \max(1, n)$ 。 <i>ilo</i> と <i>ihi</i> は相対インデックス。
<i>a</i>	(ローカル) COMPLEX (pcunmhr の場合) DOUBLE COMPLEX (pzunmhr の場合)。 ローカルメモリにある、次元 ( $lld\_a, LOCc(ja+m-1)$ ) ( <i>side</i> ='L' の場合) または ( $lld\_a, LOCc(ja+n-1)$ ) ( <i>side</i> ='R' の場合) の配列へのポインタ。 <a href="#">p?gehrd</a> によって返される、基本リフレクタを定義する一連のベクトルを含む。
<i>ia, ja</i>	(グローバル) INTEGER。 それぞれ、部分行列 $A$ の最初の行と最初の列を示す、グローバル配列 $a$ の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( $dlen\_$ )。 分散行列 $A$ の配列ディスクリプタ。
<i>tau</i>	(ローカル) COMPLEX (pcunmhr の場合) DOUBLE COMPLEX (pzunmhr の場合)。 配列、次元は $LOCc(ja+m-2)$ ( <i>side</i> ='L' の場合)

または  $LOCc(ja+n-2)$  ( $side = 'R'$  の場合)。

[p?gehrd](#) によって返される、基本リフレクタ  $H(j)$  のスカラー係数  $\tau(j)$  を含む。  $\tau$  は、分散行列  $A$  に関連付けられる。

$c$	(ローカル) COMPLEX (pcunmhr の場合) DOUBLE COMPLEX (pzunmhr の場合)。 ローカルメモリにある、次元 ( $lld\_c, LOCc(jc+n-1)$ ) の配列へのポインタ。分散行列 $sub(C)$ のローカル部分を含む。
$ic, jc$	(グローバル) INTEGER。 それぞれ、部分行列 $C$ の最初の行と最初の列を示す、グローバル配列 $c$ の行インデックスと列インデックス。
$descc$	(グローバルおよびローカル) INTEGER。配列、次元は ( $dlen\_$ )。 分散行列 $C$ の配列ディスクリプタ。
$work$	(ローカル) COMPLEX (pcunmhr の場合) DOUBLE COMPLEX (pzunmhr の場合)。 次元 $lwork$ のワークスペース配列。
$lwork$	(ローカルまたはグローバル) 配列 $work$ の次元。 $lwork$ は $iaa = ia + ilo; jaa = ja + ilo - 1$ 以上でなければならない。 $side = 'L'$ の場合、 $mi = ihi - ilo; ni = n; icc = ic + ilo; jcc = jc; lwork \geq \max((nb\_a * (nb\_a - 1)) / 2, (nqc0 + mpc0) * nb\_a) + nb\_a * nb\_a$ $side = 'R'$ の場合、 $mi = m; ni = ihi - ilo; icc = ic; jcc = jc + ilo; lwork \geq \max((nb\_a * (nb\_a - 1)) / 2, (nqc0 + \max(npa0 + \text{numroc}(\text{numroc}(ni + icoffc, nb\_a, 0, 0, NPCOL), nb\_a, 0, 0, lcmq), mpc0)) * nb\_a) + nb\_a * nb\_a$ でなければならない。 ここで、 $lcmq = lcm / NPCOL$ で $lcm = ilcm(NPROW, NPCOL)$ , $iroffa = \text{mod}(iaa - 1, mb\_a)$ , $icoffa = \text{mod}(jaa - 1, nb\_a)$ , $iarow = \text{indxg2p}(iaa, mb\_a, MYROW, rsrc\_a, NPROW)$ , $npa0 = \text{numroc}(ni + iroffa, mb\_a, MYROW, iarow, NPROW)$ , $iroffc = \text{mod}(icc - 1, mb\_c)$ , $icoffc = \text{mod}(jcc - 1, nb\_c)$ ,

```
icrow = indxg2p (icc, mb_c, MYROW, rsrc_c, NPROW)、
iccol = indxg2p (jcc, nb_c, MYCOL, csrc_c, NPCOL)、
mpc0 = numroc(mi+iroffc, mb_c, MYROW, icrow, NPROW)、
nqc0 = numroc(ni+icoffc, nb_c, MYCOL, iccol, NPCOL)
```

ilcm、indxg2p および numroc は、ScaLAPACK ツール関数である。  
MYROW、MYCOL、NPROW、および NPCOL は、サブルーチン  
blacs\_gridinfo を呼び出して決定できる。

lwork = -1 の場合、lwork はグローバル入力であり、ワークスペース  
のクエリとみなされ、ルーチンはすべての work 配列の最小かつ最適  
なサイズだけを計算する。各値は該当する work 配列の最初のエント  
リとして返され、[pxerbla](#) はエラー・メッセージを生成しない。

### 出力パラメータ

c	C は $Q^* \text{sub}(C)$ 、 $Q^* \text{sub}(C)$ 、 $\text{sub}(C)^* Q'$ 、または $\text{sub}(C)^* Q$ のいずれかに よって上書きされる。
work(1)	終了時に、work(1) には、最適なパフォーマンスを得るために必要な lwork の最小値が格納される。
info	(グローバル) INTEGER。 info = 0 の場合、実行は正常に終了したことを示す。 info < 0 の場合： i 番目の引数が配列で、j 番目の値が不正だった場合、 info = -(i*100+j)。i 番目の引数がスカラで値が不正だった場合、 info = -i。

---

## p?lahqr

すでに *Hessenberg* 形式の行列の *Schur* 分解および  
固有値を計算する。

---

### 構文

```
call pslahqr(wantt, wantz, n, ilo, ihi, a, desca, wr, wi, iloz, ihiz, z,
             descz, work, lwork, iwork, ilwork, info)
call pdlahqr(wantt, wantz, n, ilo, ihi, a, desca, wr, wi, iloz, ihiz, z,
             descz, work, lwork, iwork, ilwork, info)
```



## 説明

このルーチンは、既に **Hessenberg** 形式の行列の列  $ilo$  から  $ihi$  までの **Schur** 分解および固有値を見つけるために使用される補助ルーチンである。

## 入力パラメータ

<code>wantt</code>	(グローバル) LOGICAL。 <code>wantt = .TRUE.</code> の場合、すべての <b>Schur</b> 形式 $T$ が必要である。 <code>wantt = .FALSE.</code> の場合、固有値のみが必要である。
<code>wantz</code>	(グローバル) LOGICAL。 <code>wantz = .TRUE.</code> の場合、 <b>Schur</b> ベクトル $z$ が必要である。 <code>wantz = .FALSE.</code> の場合、 <b>Schur</b> ベクトルは必要ない。
<code>n</code>	(グローバル) INTEGER。 <b>Hessenberg</b> 行列 $A$ の次数 ( <code>wantz</code> の場合は $z$ ) ( $n \geq 0$ )。
<code>ilo, ihi</code>	(グローバル) INTEGER。 $A$ は、行と列が $ihi+1:n$ の上準三角になっているとする。また、 $A(ilo, ilo-1) = 0$ ( $ilo = 1$ でない限り) であるとする。 <code>?lahqr</code> は基本的に、行と列が $ilo$ から $ihi$ の <b>Hessenberg</b> 部分行列を対象とするが、 <code>wantt</code> が <code>.TRUE.</code> 、 $1 \leq ilo \leq \max(1, ihi); ihi \leq n$ の場合は $h$ のすべてに変換が適用される。
<code>a</code>	(グローバル) REAL ( <code>pslahqr</code> の場合) DOUBLE PRECISION ( <code>pdlahqr</code> の場合) 配列、次元は ( <code>desca(lld_)</code> ,*)。呼び出し時は、上 <b>Hessenberg</b> 行列 $A$ 。
<code>desca</code>	(グローバルおよびローカル) INTEGER。配列、次元は ( <code>dlen_</code> )。 分散行列 $A$ の配列ディスクリプタ。
<code>iloz, ihiz</code>	(グローバル) INTEGER。 <code>wantz</code> が <code>.TRUE.</code> の場合、変換が適用される $z$ の行を指定する。 $1 \leq iloz \leq ilo, ihi \leq ihiz \leq n$ 。
<code>z</code>	(グローバル) REAL ( <code>pslahqr</code> の場合) DOUBLE PRECISION ( <code>pdlahqr</code> の場合) 配列。 <code>wantz</code> が <code>.TRUE.</code> の場合、呼び出し時に、 $z$ は <code>pdhseqr</code> で蓄積された変換の現在の行列 $z$ を含んでいなければならない。 <code>wantz</code> が <code>.FALSE.</code> の場合、 $z$ は参照されない。
<code>descz</code>	(グローバルおよびローカル) INTEGER。配列、次元は ( <code>dlen_</code> )。 分散行列 $Z$ の配列ディスクリプタ。

<i>work</i>	(ローカル) REAL (pslahqr の場合) DOUBLE PRECISION (pdlahqr の場合) ワークスペース配列、次元は <i>lwork</i> 。
<i>lwork</i>	(ローカル) INTEGER。 <i>work</i> の次元。 <i>lwork</i> は十分大きいと仮定される。 すなわち、 $lwork \geq 3*n + \max(2*\max(descz(lld\_), desca(lld\_)) + 2*LOCq(n), 7*\text{ceil}(n/hbl)/lcm(NPROW, NPCOL))$ 。 <i>lwork</i> = -1 の場合、 <i>work</i> (1) は上記の数に設定され、コードは直ちに戻る。
<i>iwork</i>	(グローバルおよびローカル) INTEGER。 配列 <i>ilwork</i> のサイズ。
<i>ilwork</i>	(ローカル) INTEGER。 <i>iblk</i> 整数配列の一部を保持する。

## 出力パラメータ

<i>a</i>	終了時に、 <i>wantt</i> が .TRUE. の場合、 <i>A</i> は行と列が <i>ilo:ihi</i> にある上準三角で、標準形式ではない $2 \times 2$ 以上の対角ブロックを持つ。 <i>wantt</i> が .FALSE. の場合、 <i>A</i> の内容は指定されない。
<i>work</i> (1)	終了時に、 <i>work</i> (1) には、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が格納される。
<i>wr, wi</i>	(グローバル複製出力) REAL (pslahqr の場合) DOUBLE PRECISION (pdlahqr の場合) 配列、次元はそれぞれ ( <i>n</i> )。 計算された固有値 <i>ilo</i> から <i>ihi</i> の実数部分と虚数部分は、それぞれ <i>wr</i> と <i>wi</i> の対応する成分に格納される。2 個の固有値が複素共役ペアとして計算された場合、それらは <i>wr</i> と <i>wi</i> の連続する成分に格納される。 すなわち <i>i</i> 番目と ( <i>i</i> +1) 番目で $wi(i) > 0$ かつ $wi(i+1) < 0$ である。 <i>wantt</i> が .TRUE. の場合、固有値は <i>A</i> に返される Schur 形式の対角と同じ順序で格納される。 <i>A</i> は、次のリリースまで、より大きな対角ブロックで返されることがある。
<i>z</i>	終了時に、 <i>z</i> は更新される。変換は部分行列 <i>z</i> ( <i>iloz:ihiz, ilo:ihi</i> ) のみに適用される。
<i>info</i>	(グローバル) INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> < 0 の場合、パラメータ数 - <i>info</i> が正しくないか一貫していないことを示す。

$info > 0$  の場合、`p?lahqr` は合計  $30*(ihi-ilo+1)$  回の反復で  $ilo$  から  $ihi$  のすべての固有値を計算できなかったことを示す。

$info = i$  の場合、 $wr$  および  $wi$  の成分  $i+1:ihi$  に、正常に計算された固有値が含まれていることを示す。

## 特異値分解

このセクションでは、 $m \times n$  の一般行列  $A$  の特異値分解 (SVD) を計算する ScaLAPACK ルーチンについて説明する (LAPACK の[特異値分解](#)を参照)。

一般行列  $A$  の SVD を見つけるために、この行列は、最初にユニタリ (直交) 変換を用いて二重対角行列  $B$  に縮退され、次に二重対角行列の SVD が計算される。 $B$  の SVD は LAPACK ルーチン [?bdsqr](#) を使用して計算されることに注意する。

[表 6-6](#) に、この分解を行う ScaLAPACK 計算ルーチンを示す。

**表 6-6**      **特異値分解 (SVD) 用のルーチン**

機能	一般行列	直交 / ユニタリ行列
$A$ を二重対角行列に縮退させる	<a href="#">p?gebrd</a>	
縮退後に行列を掛ける		<a href="#">p?ormbr</a> / <a href="#">p?unmbr</a>

## p?gebrd

一般行列を二重対角形式に縮退させる。

### 構文

```
call psgebrd(m, n, a, ia, ja, desca, d, e, tauq, taup, work, lwork, info)
call pdgebrd(m, n, a, ia, ja, desca, d, e, tauq, taup, work, lwork, info)
call pcgebrd(m, n, a, ia, ja, desca, d, e, tauq, taup, work, lwork, info)
call pzgebrd(m, n, a, ia, ja, desca, d, e, tauq, taup, work, lwork, info)
```

### 説明

このルーチンは、直交 / ユニタリ変換を用いて、 $m \times n$  の一般実 / 複素分布行列  $\text{sub}(A) = A(ia:ia+m-1, ja:ja+n-1)$  を上または下二重対角形式  $B$  に縮退させる。

$$Q' * \text{sub}(A) * P = B$$

$m \geq n$  の場合、 $B$  は上二重対角である。 $m < n$  の場合、 $B$  は下二重対角である。

## 入力パラメータ

$m$	(グローバル) INTEGER。 分散行列 $\text{sub}(A)$ の行数 ( $m \geq 0$ )。
$n$	(グローバル) INTEGER。 分散行列 $\text{sub}(A)$ の列数 ( $n \geq 0$ )。
$a$	(ローカル) REAL (psgebrd の場合) DOUBLE PRECISION (pdgebrd の場合) COMPLEX (pcgebrd の場合) DOUBLE COMPLEX (pzgebrd の場合)。  ローカルメモリにある、次元 ( $lld\_a, LOCc(ja+n-1)$ ) の配列へのポインタ。呼び出し時に、この配列は、分散行列 $\text{sub}(A)$ を含む。
$ia, ja$	(グローバル) INTEGER。 それぞれ、部分行列 $A$ の最初の行と最初の列を示す、グローバル配列 $a$ の行インデックスと列インデックス。
$desca$	(グローバルおよびローカル) INTEGER。配列、次元は ( $dlen\_$ )。 分散行列 $A$ の配列ディスクリプタ。
$work$	(ローカル) REAL (sgebrd の場合) DOUBLE PRECISION (pdgebrd の場合) COMPLEX (pcgebrd の場合) DOUBLE COMPLEX (pzgebrd の場合)。次元 $lwork$ のワークスペース配列。
$lwork$	(ローカルまたはグローバル) INTEGER。 $work$ の次元は $lwork \geq nb * (mpa0 + nqa0 + 1) + nqa0$ でなければならない。ここで、 $NB = mb\_a = nb\_a$ 、 $iroffa = \text{mod}(ia-1, nb)$ 、 $icoffa = \text{mod}(ja-1, NB)$ 、 $iarow = \text{indxg2p}(ia, nb, MYROW, rsrc\_a, NPROW)$ 、 $iacol = \text{indxg2p}(ja, NB, MYCOL, csrc\_a, NPCOL)$ 、 $mpa0 = \text{numroc}(m + iroffa, NB, MYROW, iarow, NPROW)$ 、

`nqa0 = numroc(n + icoffa, NB, MYCOL, iacol, NPCOL)`

`indxg2p` および `numroc` は、ScaLAPACK ツール関数である。MYROW、MYCOL、NPROW、および NPCOL は、サブルーチン `blacs_gridinfo` を呼び出して決定できる。

`lwork = -1` の場合、`lwork` はグローバル入力であり、ワークスペースのクエリとみなされ、ルーチンはすべての `work` 配列の最小かつ最適なサイズだけを計算する。各値は該当する `work` 配列の最初のエン트리として返され、[pxerbla](#) はエラー・メッセージを生成しない。

## 出力パラメータ

- a** 終了時に、 $m \geq n$  の場合、 $\text{sub}(A)$  の対角成分と最初の優対角成分は、上二重対角行列  $B$  によって上書きされる。対角成分より下の成分は、配列 `tauq` とともに、基本リフレクタの積として直交/ユニタリの行列  $Q$  を表現する。最初の優対角成分より上の成分は、配列 `taup` とともに、基本リフレクタの積として直交行列  $P$  を表現する。 $m < n$  の場合、対角成分と最初の劣対角成分は、下二重対角行列  $B$  によって上書きされる。最初の劣対角より下の成分は、配列 `tauq` とともに、基本リフレクタの積として直交/ユニタリの行列  $Q$  を表現する。対角より上の成分は、配列 `taup` とともに、基本リフレクタの積として直交行列  $P$  を表す。(次の「アプリケーション・ノート」を参照)。
- d** (ローカル)  
REAL (単精度の場合)  
DOUBLE PRECISION (倍精度の場合)。配列、次元は  $LOCc(ja + \min(m, n) - 1)$  ( $m \geq n$  の場合) または  $LOCr(ia + \min(m, n) - 1)$  (それ以外の場合)。二重対角行列  $B$  の分散対角成分。  $d(i) = a(i, i)$ 。  
 $d$  は分散行列  $A$  に関連付けられる。
- e** (ローカル)  
REAL (単精度の場合)  
DOUBLE PRECISION (倍精度の場合)。配列、次元は  $LOCr(ia + \min(m, n) - 1)$  ( $m \geq n$  の場合) または  $LOCc(ja + \min(m, n) - 2)$  (それ以外の場合)。二重対角分散行列  $B$  の非対角成分。  
 $m \geq n$  の場合、  
 $e(i) = a(i, i+1)$  ( $i = 1, 2, \dots, n-1$ )  
 $m < n$  の場合、  
 $e(i) = a(i+1, i)$  ( $i = 1, 2, \dots, m-1$ )  
 $e$  は、分散行列  $A$  に関連付けられる。

<i>tauq, tau</i>	(ローカル) REAL (psgebrd の場合) DOUBLE PRECISION (pdgebrd の場合) COMPLEX (pcgebrd の場合) DOUBLE COMPLEX (pzgebrd の場合)。 配列、次元は $LOCc(ja+\min(m,n)-1)$ ( <i>tauq</i> の場合) または $LOCr(ia+\min(m,n)-1)$ ( <i>tau</i> の場合)。 それぞれ、直交/ユニタリ行列 $Q$ と $P$ を表す基本リフレクタのスカラー 係数が格納される。 <i>tauq</i> と <i>tau</i> は、分散行列 $A$ に関連付けられる。 (次の「アプリケーション・ノート」を参照)。
<i>work(1)</i>	終了時に、 <i>work(1)</i> には、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が格納される。
<i>info</i>	(グローバル) INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> < 0 の場合: <i>i</i> 番目の引数が配列で、 <i>j</i> 番目の値が不正だった場合、 <i>info</i> = -( <i>i</i> *100+ <i>j</i> )。 <i>i</i> 番目の引数がスカラーで値が不正だった場合、 <i>info</i> = - <i>i</i> 。

## アプリケーション・ノート

行列  $Q$  と  $P$  は基本リフレクタの積として表現される。

$m \geq n$  の場合、

$$Q = H(1) H(2) \dots H(n) \text{ および } P = G(1) G(2) \dots G(n-1)。$$

それぞれの  $H(i)$  と  $G(i)$  は次のような形式を持つ。

$$H(i) = I - \tau q * v * v' \text{ および } G(i) = I - \tau p * u * u'$$

ここで、*tauq* と *taup* は実/複素スカラー、 $v$  と  $u$  は実/複素ベクトルである。

$v(1:i-1) = 0$ 、 $v(i) = 1$  および  $v(i+1:m)$  は、終了時に、 $A(ia+i:ia+m-1, ja+i-1)$  に格納され、

$u(1:i) = 0$ 、 $u(i+1) = 1$  および  $u(i+2:n)$  は、終了時に、 $A(ia+i-1, ja+i+1:ja+n-1)$  に格納され、

*tauq* は *tauq(ja+i-1)* に格納され、*taup* は *taup(ia+i-1)* に格納される。

$m < n$  の場合、

$$Q = H(1) H(2) \dots H(m-1) \text{ および } P = G(1) G(2) \dots G(m)$$

それぞれの  $H(i)$  と  $G(i)$  は次のような形式を持つ。

$$H(i) = I - \text{tauq} * v * v' \text{ および } G(i) = I - \text{taup} * u * u'$$

ここで、 $\text{tauq}$  と  $\text{taup}$  は実 / 複素スカラ、 $v$  と  $u$  は実 / 複素ベクトルである。

$v(1:i) = 0$ 、 $v(i+1) = 1$  および  $v(i+2:m)$  は、終了時に、 $A(ia+i:ia+m-1, ja+i-1)$  に格納され、 $u(1:i-1) = 0$ 、 $u(i) = 1$  および  $u(i+1:n)$  は、終了時に、 $A(ia+i-1, ja+i+1:ja+n-1)$  に格納される。

$\text{tauq}$  は  $\text{tauq}(ja+i-1)$  に格納され、 $\text{taup}$  は  $\text{taup}(ia+i-1)$  に格納される。

ルーチン終了後の  $\text{sub}(A)$  の内容は、次の例に示される。

$m = 6$  および  $n = 5$  ( $m > n$ )。

$$\begin{bmatrix} d & e & u1 & u1 & u1 \\ v1 & d & e & u2 & u2 \\ v1 & v2 & d & e & u3 \\ v1 & v2 & v3 & d & e \\ v1 & v2 & v3 & v4 & d \\ v1 & v2 & v3 & v4 & v5 \end{bmatrix}$$

$m = 5$  および  $n = 6$  ( $m < n$ )。

$$\begin{bmatrix} d & u1 & u1 & u1 & u1 & u1 \\ e & d & u2 & u2 & u2 & u2 \\ v1 & e & d & u3 & u3 & u3 \\ v1 & v2 & e & d & u4 & u4 \\ v1 & v2 & v3 & e & d & u5 \end{bmatrix}$$

ここで、 $d$  と  $e$  は  $B$  の対角成分と非対角成分である。 $v_i$  は  $H(i)$  を定義するベクトルの成分を表し、 $u_i$  は  $G(i)$  を定義するベクトルの成分を表す。

### p?ormbr

一般行列に p?gebrd で求めた 二重対角形式に縮退した直交行列を掛ける。

#### 構文

```
call psormbr(vect, side, trans, m, n, k, a, ia, ja, desca, tau, c, ic,
             jc, descc, work, lwork, info)
call pdormbr(vect, side, trans, m, n, k, a, ia, ja, desca, tau, c, ic,
             jc, descc, work, lwork, info)
```

#### 説明

$vect = 'Q'$  の場合、このルーチンは、 $m \times n$  の一般実分散行列  $sub(C) = C(c:ic+m-1, jc:jc+n-1)$  を以下により上書きする。

	$side = 'L'$	$side = 'R'$
$trans = 'N':$	$Q \ sub(C)$	$sub(C) \ Q$
$trans = 'T':$	$Q^T \ sub(C)$	$sub(C) \ Q^T$

$vect = 'P'$  の場合、このルーチンは、 $sub(C)$  を以下により上書きする。

	$side = 'L'$	$side = 'R'$
$trans = 'N':$	$P \ sub(C)$	$sub(C) \ P$
$trans = 'T':$	$P^T \ sub(C)$	$sub(C) \ P^T$

ここで、 $Q$  と  $P^T$  は、実分散行列  $A(ia:*, ja:*)$  を二重対角形式  $A(ia:*, ja:*) = Q B P^T$  に縮退させるときに [p?gebrd](#) で求めた直交分散行列である。 $Q$  と  $P^T$  は、それぞれ、基本リフレクタ  $H(i)$  と  $G(i)$  の積として定義される。

$nq = m$  ( $side = 'L'$  の場合) または  $nq = n$  ( $side = 'R'$  の場合)。したがって、 $nq$  は適用される直交行列  $Q$  または  $P^T$  の次数である。

$vect = 'Q'$  の場合、 $A(ia:*, ja:*)$  は  $nq \times k$  行列であると仮定する。



$nq \geq k$  の場合、 $Q = H(1) H(2) \dots H(k)$

$nq < k$  の場合、 $Q = H(1) H(2) \dots H(nq-1)$

$vect = 'P'$  の場合、 $A(ia:*,ja:*)$  は、 $k \times nq$  行列であると仮定する。

$k < nq$  の場合、 $P = G(1) G(2) \dots G(k)$

$k \geq nq$  の場合、 $P = G(1) G(2) \dots G(nq-1)$

## 入力パラメータ

<b>vect</b>	(グローバル) CHARACTER。 $vect = 'Q'$ の場合、 $Q$ または $Q^T$ が適用される。 $vect = 'P'$ の場合、 $P$ または $P^T$ が適用される。
<b>side</b>	(グローバル) CHARACTER。 $side = 'L'$ の場合、 $Q$ または $Q^T$ 、 $P$ または $P^T$ は左側から適用される。 $side = 'R'$ の場合、 $Q$ または $Q^T$ 、 $P$ または $P^T$ は右側から適用される。
<b>trans</b>	(グローバル) CHARACTER。 $trans = 'N'$ の場合、 $Q$ または $P$ が適用される (転置なし)。 $trans = 'T'$ の場合、 $Q^T$ または $P^T$ が適用される。
<b>m</b>	(グローバル) INTEGER。分散行列 $\text{sub}(C)$ の行数。
<b>n</b>	(グローバル) INTEGER。 分散行列 $\text{sub}(C)$ の列数。
<b>k</b>	(グローバル) INTEGER。 $vect = 'Q'$ の場合、 <a href="#">p?gebrd</a> により縮退された元の分散行列の列数。 $vect = 'P'$ の場合、 <a href="#">p?gebrd</a> により縮退された元の分散行列の行数。 次の制約がある。 $k \geq 0$ 。
<b>a</b>	(ローカル) REAL (psormbr の場合) DOUBLE PRECISION (pdormbr の場合)。 ローカルメモリにある、次元 $(lld\_a, LOCC(ja+\min(nq,k)-1))$ ( $vect = 'Q'$ の場合) または $(lld\_a, LOCC(ja+nq-1))$ ( $vect = 'P'$ の場合) の配列へのポインタ。 $nq = m$ ( $side = 'L'$ の場合) または $nq = n$ (それ以外の場合)。 基本リフレクタ $H(i)$ と $G(i)$ を定義するベクトル。 <a href="#">p?gebrd</a> から返さ

	<p>れたとおりに、行列 <math>Q</math> と <math>P</math> を決定する積である。</p> <p><math>vect = 'Q'</math> の場合、<math>lld\_a \geq \max(1, LOCr(ia+nq-1))</math>。</p> <p><math>vect = 'P'</math> の場合、<math>lld\_a \geq \max(1, LOCr(ia+\min(nq,k)-1))</math>。</p>
$ia, ja$	<p>(グローバル) INTEGER。</p> <p>それぞれ、部分行列 <math>A</math> の最初の行と最初の列を示す、グローバル配列 <math>a</math> の行インデックスと列インデックス。</p>
$desca$	<p>(グローバルおよびローカル) INTEGER。配列、次元は <math>(dlen\_)</math>。</p> <p>分散行列 <math>A</math> の配列ディスクリプタ。</p>
$tau$	<p>(ローカル)</p> <p>REAL (psormbr の場合)</p> <p>DOUBLE PRECISION (pdormbr の場合)。</p> <p>配列、次元は <math>LOCc(ja+\min(nq,k)-1)</math> (<math>vect = 'Q'</math> の場合) または <math>LOCr(ia+\min(nq,k)-1)</math> (<math>vect = 'P'</math> の場合)。</p>
	<p><math>tau(i)</math> は、配列引数 <math>tauq</math> または <math>taup</math> を使用して pdgebrd から返されたとおりに、<math>Q</math> あるいは <math>P</math> を決定する、基本リフレクタ <math>H(i)</math> または <math>G(i)</math> のスカラ係数を含んでいなければならない。<math>tau</math> は、分散行列 <math>A</math> に関連付けられる。</p>
$c$	<p>(ローカル)</p> <p>REAL (psormbr の場合)</p> <p>DOUBLE PRECISION (pdormbr の場合)。</p> <p>ローカルメモリにある、次元 <math>(lld\_a, LOCc(jc+n-1))</math> の配列へのポインタ。分散行列 <math>sub(C)</math> のローカル部分を含む。</p>
$ic, jc$	<p>(グローバル) INTEGER。</p> <p>それぞれ、部分行列 <math>C</math> の最初の行と最初の列を示す、グローバル配列 <math>c</math> の行インデックスと列インデックス。</p>
$descc$	<p>(グローバルおよびローカル) INTEGER。配列、次元は <math>(dlen\_)</math>。</p> <p>分散行列 <math>C</math> の配列ディスクリプタ。</p>
$work$	<p>(ローカル)</p> <p>REAL (psormbr の場合)</p> <p>DOUBLE PRECISION (pdormbr の場合)。</p> <p>次元 <math>lwork</math> のワークスペース配列。</p>
$lwork$	<p>(ローカルまたはグローバル) INTEGER。 <math>work</math> の次元は</p> <p><math>side = 'L'</math> の場合、</p> <p><math>nq = m</math>。</p>

((*vect* = 'Q' かつ  $nq \geq k$ ) または (*vect* が 'Q' と等しくなく、かつ  $nq > k$ ))  
の場合、*iaa*=*ia*、*jaa*=*ja*、*mi*=*m*、*ni*=*n*、*icc*=*ic*、*jcc*=*jc*。

そうでない場合、

*iaa*=*ia*+1、*jaa*=*ja*、*mi*=*m*-1、*ni*=*n*、*icc*=*ic*+1、*jcc*=*jc*。

でなければならない。

*side* = 'R' の場合、 $nq = n$ 。

((*vect* = 'Q' かつ  $nq \geq k$ ) または (*vect* が 'Q' と等しくなく、かつ  $nq > k$ ))  
の場合、

*iaa*=*ia*、*jaa*=*ja*、*mi*=*m*、*ni*=*n*、*icc*=*ic*、*jcc*=*jc*。

そうでない場合、

*iaa*=*ia*、*jaa*=*ja*+1、*mi*=*m*、*ni*=*n*-1、*icc*=*ic*、*jcc*=*jc*+1。

でなければならない。

*vect* = 'Q' の場合、

*side* = 'L' の場合、 $lwork \geq \max((nb\_a*(nb\_a-1))/2, (nqc0 + mpc0)*nb\_a + nb\_a * nb\_a)$

*side* = 'R' の場合、

$lwork \geq \max((nb\_a*(nb\_a-1))/2, (nqc0 + \max(npa0 + \text{numroc}(\text{numroc}(ni+icoffc, nb\_a, 0, 0, NPCOL), nb\_a, 0, 0, lcmq), mpc0))*nb\_a) + nb\_a * nb\_a * \text{ でなければならない。}$

*vect* が 'Q' と等しくない場合で、*side* = 'L' の場合、

$lwork \geq \max((mb\_a*(mb\_a-1))/2, (mpc0 + \max(mqa0 + \text{numroc}(\text{numroc}(mi+iroffc, mb\_a, 0, 0, NPROW), mb\_a, 0, 0, lcmp), nqc0))*mb\_a) + mb\_a * mb\_a$

*side* = 'R' の場合、

$lwork \geq \max((mb\_a*(mb\_a-1))/2, (mpc0 + nqc0)*mb\_a) + mb\_a * mb\_a$

でなければならない。

ここで、 $lcmp = lcm / NPROW$ 、 $lcmq = lcm / NPCOL$ 、

$lcm = ilcm(NPROW, NPCOL)$ 、

$iroffa = \text{mod}(iaa-1, mb\_a)$ 、

```

icoffa = mod(jaa-1, nb_a)、
iarow = indxc2p (iaa, mb_a, MYROW, rsrc_a, NPROW)、
iacol = indxc2p (jaa, nb_a, MYCOL, csrc_a, NPCOL)、
mqa0 = numroc(mi+icoffa, nb_a, MYCOL, iacol, NPCOL)、
npa0 = numroc(ni+iroffa, mb_a, MYROW, iarow, NPROW)、
iroffc = mod(icc-1, mb_c)、
icoffc = mod(jcc-1, nb_c)、
icrow = indxc2p (icc, mb_c, MYROW, rsrc_c, NPROW)、
iccol = indxc2p (jcc, nb_c, MYCOL, csrc_c, NPCOL)、
mpc0 = numroc(mi+iroffc, mb_c, MYROW, icrow, NPROW)、
nqc0 = numroc(ni+icoffc, nb_c, MYCOL, iccol, NPCOL)

```

indxc2p および numroc は、ScaLAPACK ツール関数である。MYROW、MYCOL、NPROW、および NPCOL は、サブルーチン blacs\_gridinfo を呼び出して決定できる。

lwork = -1 の場合、lwork はグローバル入力であり、ワークスペースのクエリとみなされ、ルーチンはすべての work 配列の最小かつ最適なサイズだけを計算する。各値は該当する work 配列の最初のエン트리として返され、[pxerbla](#) はエラー・メッセージを生成しない。

## 出力パラメータ

<i>c</i>	終了時に、 <i>vect</i> ='Q' の場合、sub( <i>C</i> ) は $Q \cdot \text{sub}(C)$ 、 $Q' \cdot \text{sub}(C)$ 、 $\text{sub}(C) \cdot Q'$ 、または $\text{sub}(C) \cdot Q$ のいずれかによって上書きされる。 <i>vect</i> ='P' の場合、sub( <i>C</i> ) は $P \cdot \text{sub}(C)$ 、 $P' \cdot \text{sub}(C)$ 、 $\text{sub}(C) \cdot P$ 、または $\text{sub}(C) \cdot P'$ のいずれかによって上書きされる。
<i>work</i> (1)	終了時に、 <i>work</i> (1) には、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が格納される。
<i>info</i>	(グローバル) INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> < 0 の場合： <i>i</i> 番目の引数が配列で、 <i>j</i> 番目の値が不正だった場合、 <i>info</i> = -( <i>i</i> *100+ <i>j</i> )。 <i>i</i> 番目の引数がスカラで値が不正だった場合、 <i>info</i> = - <i>i</i> 。

## p?unmbr

p?gebrd で求めた二重対角形式に縮退したユニタリ変換行列を一般行列に掛ける。

### 構文

```
call cunmbr(vect, side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc,
            descc, work, lwork, info)
call zunmbr(vect, side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc,
            descc, work, lwork, info)
```

### 説明

$\text{vect} = 'Q'$  の場合、このルーチンは、 $m \times n$  の一般複素分散行列  $\text{sub}(C) = C(\text{ic}:\text{ic}+m-1, \text{jc}:\text{jc}+n-1)$  を以下により上書きする。

	$\text{side} = 'L'$	$\text{side} = 'R'$
$\text{trans} = 'N':$	$Q \text{sub}(C)$	$\text{sub}(C) Q$
$\text{trans} = 'C':$	$Q^H \text{sub}(C)$	$\text{sub}(C) Q^H$

$\text{vect} = 'P'$  の場合、このルーチンは、 $\text{sub}(C)$  を以下により上書きする。

	$\text{side} = 'L'$	$\text{side} = 'R'$
$\text{trans} = 'N':$	$P \text{sub}(C)$	$\text{sub}(C) P$
$\text{trans} = 'C':$	$P^H \text{sub}(C)$	$\text{sub}(C) P^H$

ここで、 $Q$  と  $P^H$  は、複素分散行列  $A(\text{ia}:\text{ia}^*, \text{ja}:\text{ja}^*)$  を二重対角形式  $A(\text{ia}:\text{ia}^*, \text{ja}:\text{ja}^*) = Q B P^H$  に縮退させるときに [p?gebrd](#) で求めたユニタリ分散行列である。 $Q$  と  $P^H$  は、それぞれ、基本リフレクタ  $H(i)$  と  $G(i)$  の積として定義される。

$nq = m$  ( $\text{side} = 'L'$  の場合) または  $nq = n$  ( $\text{side} = 'R'$  の場合)。したがって、 $nq$  は適用されるユニタリ行列  $Q$  または  $P^H$  の次数である。

$\text{vect} = 'Q'$  の場合、 $A(\text{ia}:\text{ia}^*, \text{ja}:\text{ja}^*)$  は  $nq \times k$  行列であると仮定する。

$nq \geq k$  の場合、 $Q = H(1) H(2) \dots H(k)$

$nq < k$  の場合、 $Q = H(1) H(2) \dots H(nq-1)$

$\text{vect} = 'P'$  の場合、 $A(\text{ia}:\text{ia}^*, \text{ja}:\text{ja}^*)$  は、 $k \times nq$  行列であると仮定する。

$k < nq$  の場合、 $P = G(1) G(2) \dots G(k)$

$k \geq nq$  の場合、 $P = G(1) G(2) \dots G(nq-1)$

## 入力パラメータ

<i>vect</i>	(グローバル) CHARACTER。 $vect = 'Q'$ の場合、 $Q$ または $Q^H$ が適用される。 $vect = 'P'$ の場合、 $P$ または $P^H$ が適用される。
<i>side</i>	(グローバル) CHARACTER。 $side = 'L'$ の場合、 $Q$ または $Q^H$ 、 $P$ または $P^H$ は左側から適用される。 $side = 'R'$ の場合、 $Q$ または $Q^H$ 、 $P$ または $P^H$ は右側から適用される。
<i>trans</i>	(グローバル) CHARACTER。 $trans = 'N'$ の場合、 $Q$ または $P$ が適用される (転置なし)。 $trans = 'C'$ の場合、 $Q^H$ または $P^H$ が適用される (共役転置)。
<i>m</i>	(グローバル) INTEGER。分散行列 $\text{sub}(C)$ の行数 ( $m \geq 0$ )。
<i>n</i>	(グローバル) INTEGER。 分散行列 $\text{sub}(C)$ の列数 ( $n \geq 0$ )。
<i>k</i>	(グローバル) INTEGER。 $vect = 'Q'$ の場合、 <a href="#">p?gebrd</a> により縮退された元の分散行列の列数。 $vect = 'P'$ の場合、 <a href="#">p?gebrd</a> により縮退された元の分散行列の行数。 次の制約がある。 $k \geq 0$ 。
<i>a</i>	(ローカル) COMPLEX (psormbr の場合) DOUBLE COMPLEX (pdormbr の場合)。 ローカルメモリにある、次元 ( $11d\_a, LOCc(ja+\min(nq,k)-1)$ ) ( $vect = 'Q'$ の場合) または ( $11d\_a, LOCc(ja+nq-1)$ ) ( $vect = 'P'$ の場合) の配列へのポインタ。 $nq = m$ ( $side = 'L'$ の場合) または $nq = n$ (それ以外の場合)。 基本リフレクタ $H(i)$ と $G(i)$ を定義するベクトル。 <a href="#">p?gebrd</a> から返されたとおりに、行列 $Q$ と $P$ を決定する積である。 $vect = 'Q'$ の場合、 $11d\_a \geq \max(1, LOCr(ia+nq-1))$ 。 $vect = 'P'$ の場合、 $11d\_a \geq \max(1, LOCr(ia+\min(nq,k)-1))$ 。
<i>ia, ja</i>	(グローバル) INTEGER。 それぞれ、部分行列 $A$ の最初の行と最初の列を示す、グローバル配列 $a$ の行インデックスと列インデックス。

<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。 分散行列 <i>A</i> の配列ディスクリプタ。
<i>tau</i>	(ローカル) COMPLEX ( <i>pcunmbr</i> の場合) DOUBLE COMPLEX ( <i>pzunmbr</i> の場合)。 配列、次元は $LOCc(ja+\min(nq,k)-1)$ ( <i>vect</i> = 'Q' の場合) または $LOCr(ia+\min(nq,k)-1)$ ( <i>vect</i> = 'P' の場合)。 <i>tau(i)</i> は、配列引数 <i>tauq</i> または <i>taup</i> を使用して <a href="#">p?gebrd</a> から返されたとおりに、 <i>Q</i> または <i>P</i> を決定する、基本リフレクタ <i>H(i)</i> あるいは <i>G(i)</i> のスカラー係数を含んでいなければならない。 <i>tau</i> は、分散行列 <i>A</i> に関連付けられる。
<i>c</i>	(ローカル) COMPLEX ( <i>pcunmbr</i> の場合) DOUBLE COMPLEX ( <i>pzunmbr</i> の場合)。 ローカルメモリにある、次元 ( <i>lld_a</i> , $LOCc(jc+n-1)$ ) の配列へのポインタ。分散行列 <i>sub(C)</i> のローカル部分を含む。
<i>ic, jc</i>	(グローバル) INTEGER。 それぞれ、部分行列 <i>C</i> の最初の行と最初の列を示す、グローバル配列 <i>c</i> の行インデックスと列インデックス。
<i>descc</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。 分散行列 <i>C</i> の配列ディスクリプタ。
<i>work</i>	(ローカル) COMPLEX ( <i>pcunmbr</i> の場合) DOUBLE COMPLEX ( <i>pzunmbr</i> の場合)。 次元 <i>lwork</i> のワークスペース配列。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 <i>work</i> の次元は <i>side</i> = 'L' の場合、 $nq = m_0$ $((vect = 'Q' \text{ かつ } nq \geq k) \text{ または } (vect \text{ が 'Q' と等しくなく、かつ } nq > k))$ の場合、 <i>iaa</i> = <i>ia</i> 、 <i>jaa</i> = <i>ja</i> 、 <i>mi</i> = <i>m</i> 、 <i>ni</i> = <i>n</i> 、 <i>icc</i> = <i>ic</i> 、 <i>jcc</i> = <i>jc</i> 。 そうでない場合、 <i>iaa</i> = <i>ia</i> +1、 <i>jaa</i> = <i>ja</i> 、 <i>mi</i> = <i>m</i> -1、 <i>ni</i> = <i>n</i> 、 <i>icc</i> = <i>ic</i> +1、 <i>jcc</i> = <i>jc</i> 。 でなければならない。 <i>side</i> = 'R' の場合、 $nq = n_0$ 。

((*vect* = 'Q' かつ  $nq \geq k$ ) または (*vect* が 'Q' と等しくなく、かつ  $nq > k$ )) の場合、

*iaa*=*ia*、*jaa*=*ja*、*mi*=*m*、*ni*=*n*、*icc*=*ic*、*jcc*=*jc*。

そうでない場合、

*iaa*=*ia*、*jaa*=*ja*+1、*mi*=*m*、*ni*=*n*-1、*icc*=*ic*、*jcc*=*jc*+1。

でなければならない。

*vect* = 'Q' の場合、

*side* = 'L' の場合、 $lwork \geq \max((nb\_a*(nb\_a-1))/2, (nqc0 + mpc0)*nb\_a + nb\_a * nb\_a)$

*side* = 'R' の場合、

$lwork \geq \max((nb\_a*(nb\_a-1))/2, (nqc0 + \max(npa0 + \text{numroc}(\text{numroc}(ni+icoffc, nb\_a, 0, 0, NPCOL), nb\_a, 0, 0, lcmq), mpc0))*nb\_a + nb\_a * nb\_a * \text{でなければならない。})$

*vect* が 'Q' と等しくない場合で、*side* = 'L' の場合、

$lwork \geq \max((mb\_a*(mb\_a-1))/2, (mpc0 + \max(mqa0 + \text{numroc}(\text{numroc}(mi+iroffc, mb\_a, 0, 0, NPROW), mb\_a, 0, 0, lcmq), nqc0))*mb\_a + mb\_a * mb\_a$

*side* = 'R' の場合、

$lwork \geq \max((mb\_a*(mb\_a-1))/2, (mpc0 + nqc0)*mb\_a + mb\_a * mb\_a$

でなければならない。

ここで、 $lcmp = lcm / NPROW$ 、 $lcmq = lcm / NPCOL$ 、 $lcm = ilcm(NPROW, NPCOL)$ 、

$iroffa = \text{mod}(iaa-1, mb\_a)$ 、

$icoffa = \text{mod}(jaa-1, nb\_a)$ 、

$iarow = \text{indxg2p}(iaa, mb\_a, MYROW, rsrc\_a, NPROW)$ 、

$iacol = \text{indxg2p}(jaa, nb\_a, MYCOL, csrc\_a, NPCOL)$ 、

$mqa0 = \text{numroc}(mi+icoffa, nb\_a, MYCOL, iacol, NPCOL)$ 、

$npa0 = \text{numroc}(ni+iroffa, mb\_a, MYROW, iarow, NPROW)$ 、



```

irow = mod(icc-1, mb_c),
icrow = mod(jcc-1, nb_c),
icrow = indxg2p (icc, mb_c, MYROW, rsrc_c, NPROW),
iccol = indxg2p (jcc, nb_c, MYCOL, csrc_c, NPCOL),
mpc0 = numroc(mi+irow, mb_c, MYROW, icrow, NPROW),
nqc0 = numroc(ni+icrow, nb_c, MYCOL, iccol, NPCOL)

```

indxg2p および numroc は、ScaLAPACK ツール関数である。MYROW、MYCOL、NPROW、および NPCOL は、サブルーチン `blacs_gridinfo` を呼び出して決定できる。

`lwork = -1` の場合、`lwork` はグローバル入力であり、ワークスペースのクエリとみなされ、ルーチンはすべての `work` 配列の最小かつ最適なサイズだけを計算する。各値は該当する `work` 配列の最初のエン트리として返され、[pxerbla](#) はエラー・メッセージを生成しない。

### 出力パラメータ

<code>c</code>	終了時に、 <code>vect='Q'</code> の場合、 <code>sub(C)</code> は $Q \cdot \text{sub}(C)$ 、 $Q' \cdot \text{sub}(C)$ 、 $\text{sub}(C) \cdot Q'$ 、または $\text{sub}(C) \cdot Q$ のいずれかによって上書きされる。 <code>vect='P'</code> の場合、 <code>sub(C)</code> は $P \cdot \text{sub}(C)$ 、 $P' \cdot \text{sub}(C)$ 、 $\text{sub}(C) \cdot P$ 、または $\text{sub}(C) \cdot P'$ のいずれかによって上書きされる。
<code>work(1)</code>	終了時に、 <code>work(1)</code> には、最適なパフォーマンスを得るために必要な <code>lwork</code> の最小値が格納される。
<code>info</code>	(グローバル) INTEGER。 <code>info = 0</code> の場合、実行は正常に終了したことを示す。 <code>info &lt; 0</code> の場合： $i$ 番目の引数が配列で、 $j$ 番目の値が不正だった場合、 <code>info = -(i*100+j)</code> 。 $i$ 番目の引数がスカラーで値が不正だった場合、 <code>info = -i</code> 。

## 汎用対称固有値問題

このセクションで説明する ScaLAPACK ルーチンを利用すると、一般対称固有値問題 (LAPACK の [汎用対称固有値問題](#) を参照) を標準の対称固有値問題  $Cy = \lambda y$  に縮退させることができる。縮退させた後の標準の対称固有値問題は、この章ですでに説明した一連の ScaLAPACK ルーチン ([対称固有値問題](#) を参照) を呼び出して解くことができる。

[表 6-7](#) に、これらのルーチンの一覧を示す。

**表 6-7 汎用固有値問題から標準固有値問題への縮退用の計算ルーチン**

機能	実対称行列	複素エルミート行列
標準問題への縮退	<a href="#">p?sygst</a>	<a href="#">p?hegst</a>

## p?sygst

実対称の汎用固有値問題を標準形式に縮退させる。

### 構文

```
call pssygst( ibtype, uplo, n, a, ia, ja, desca, b, ib, jb, descb, scale,
             info )
call pdsygst( ibtype, uplo, n, a, ia, ja, desca, b, ib, jb, descb, scale,
             info )
```

### 説明

このルーチンは、実対称の汎用固有値問題を標準形式に縮退させる。

以下において、 $\text{sub}(A)$  は  $A(ia:ia+n-1, ja:ja+n-1)$  を表し、 $\text{sub}(B)$  は  $B(ib:ib+n-1, jb:jb+n-1)$  を表す。

$ibtype = 1$  の場合、問題は

$$\text{sub}(A)x = \lambda \text{sub}(B)x$$

$\text{sub}(A)$  は、 $\text{inv}(U^T) \text{sub}(A) \text{inv}(U)$  または  $\text{inv}(L) \text{sub}(A) \text{inv}(L^T)$  によって上書きされる。

$ibtype = 2$  または  $3$  の場合、問題は

$$\text{sub}(A) \text{sub}(B)x = \lambda x \text{ または } \text{sub}(B) \text{sub}(A)x = \lambda x$$

$\text{sub}(A)$  は、 $U \text{sub}(A) U^T$  または  $L^T \text{sub}(A) L$  によって上書きされる。

$\text{sub}(B)$  は、 $U^T U$  または  $LL^T$  として、`?potrf` によって事前に因子分解されていなければならない。

## 入力パラメータ

<i>ibtype</i>	(グローバル) INTEGER。1、2、または3のいずれかでなければならない。 $ibtype = 1$ の場合、 $\text{inv}(U^T) \text{sub}(A) \text{inv}(U)$ または $\text{inv}(L) \text{sub}(A) \text{inv}(L^T)$ を計算する。 $ibtype = 2$ または 3 の場合、 $U \text{sub}(A) U^T$ あるいは $L^T \text{sub}(A) L$ を計算する。
<i>uplo</i>	(グローバル) CHARACTER。'U' または 'L' でなければならない。 $uplo = 'U'$ の場合、 $\text{sub}(A)$ の上三角部分が格納される。 $\text{sub}(B)$ は $U^T U$ として因子分解される。 $uplo = 'L'$ の場合、 $\text{sub}(A)$ の下三角部分が格納される。 $\text{sub}(B)$ は $LL^T$ として因子分解される。
<i>n</i>	(グローバル) INTEGER。 行列 $\text{sub}(A)$ と $\text{sub}(B)$ の次数 ( $n \geq 0$ )。
<i>a</i>	(ローカル) REAL ( <code>pssygst</code> の場合) DOUBLE PRECISION ( <code>pdsygst</code> の場合)。 ローカルメモリにある、次元 ( <code>lld_a</code> , <code>LOCc(ja+n-1)</code> ) の配列へのポインタ。呼び出し時に、配列は $n \times n$ の対称分散行列 $\text{sub}(A)$ のローカル部分を含む。 $uplo = 'U'$ の場合、 $\text{sub}(A)$ の先頭の $n \times n$ 上三角部分は行列の上三角部分を含む。厳密な下三角部分は参照されない。 $uplo = 'L'$ の場合、 $\text{sub}(A)$ の先頭の $n \times n$ 下三角部分は行列の下三角部分を含む。厳密な上三角部分は参照されない。
<i>ia, ja</i>	(グローバル) INTEGER。 それぞれ、部分行列 $A$ の最初の行と最初の列を示す、グローバル配列 $a$ の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <code>dlen_</code> )。 分散行列 $A$ の配列ディスクリプタ。
<i>b</i>	(ローカル) REAL ( <code>pssygst</code> の場合) DOUBLE PRECISION ( <code>pdsygst</code> の場合)。

ローカルメモリにある、次元 ( $lld\_b, LOCc(jb+n-1)$ ) の配列へのポインタ。呼び出し時に、配列は、`p?potrf` によって返される  $\text{sub}(B)$  のコレスキー因子分解で得られた三角係数を含む。

*ib, jb* (グローバル) INTEGER。  
それぞれ、部分行列  $B$  の最初の行と最初の列を示す、グローバル配列  $b$  の行インデックスと列インデックス。

*descb* (グローバルおよびローカル) INTEGER。配列、次元は ( $dlen\_$ )。分散行列  $B$  の配列ディスクリプタ。

### 出力パラメータ

*a* 終了時に、 $info = 0$  の場合、変換後の行列が  $\text{sub}(A)$  と同じ形式で格納される。

*scale* (グローバル)  
REAL (`pssygst` の場合)  
DOUBLE PRECISION (`pdsygst` の場合)。  
  
このルーチンで実行されたスケーリングを補正するための固有値。  
 $scale$  は将来の拡張のために用意されており、現在は常に 1.0 として返される。

*info* (グローバル) INTEGER。  
 $info = 0$  の場合、実行は正常に終了したことを示す。  
 $info < 0$  の場合、 $i$  番目の引数が配列で、 $j$  番目の値が不正だった場合、 $info = -(i100+j)$ 。 $i$  番目の引数がスカラで値が不正だった場合、 $info = -i$ 。

---

## p?hegst

エルミートの汎用固有値問題を標準形式に縮退させる。

---

### 構文

```
call pchegst( ibtype, uplo, n, a, ia, ja, desca, b, ib, jb, descb, scale,
             info )
call pzhegst( ibtype, uplo, n, a, ia, ja, desca, b, ib, jb, descb, scale,
             info )
```

## 説明

このルーチンは、複素エルミートの汎用固有値問題を標準形式に縮退させる。

以下において、 $\text{sub}(A)$  は  $A(\text{ia}:\text{ia}+n-1, \text{ja}:\text{ja}+n-1)$  を表し、 $\text{sub}(B)$  は  $B(\text{ib}:\text{ib}+n-1, \text{jb}:\text{jb}+n-1)$  を表す。

$\text{ibtype} = 1$  の場合、問題は

$$\text{sub}(A)x = \lambda \text{sub}(B)x$$

$\text{sub}(A)$  は、 $\text{inv}(U^H) \text{sub}(A) \text{inv}(U)$  または  $\text{inv}(L) \text{sub}(A) \text{inv}(L^H)$  によって上書きされる。

$\text{ibtype} = 2$  または  $3$  の場合、問題は

$$\text{sub}(A)\text{sub}(B)x = \lambda x \text{ または } \text{sub}(B)\text{sub}(A)x = \lambda x$$

$\text{sub}(A)$  は、 $U \text{sub}(A) U^H$  または  $L^H \text{sub}(A) L$  によって上書きされる。

$\text{sub}(B)$  は、 $U^H U$  または  $LL^H$  として、`?potrf` によって事前に因子分解されていなければならない。

## 入力パラメータ

<i>ibtype</i>	(グローバル) INTEGER。 1、2、または3のいずれかでなければならない。 <i>itype</i> = 1 の場合、 $\text{inv}(U^H) \text{sub}(A) \text{inv}(U)$ または $\text{inv}(L) \text{sub}(A) \text{inv}(L^H)$ を計算する。 <i>itype</i> = 2 または 3 の場合、 $U \text{sub}(A) U^H$ または $L^H \text{sub}(A) L$ を計算する。
<i>uplo</i>	(グローバル) CHARACTER。 'U' または 'L' でなければならない。 <i>uplo</i> = 'U' の場合、 $\text{sub}(A)$ の上三角部分が格納される。 $\text{sub}(B)$ は $U^H U$ として因子分解される。 <i>uplo</i> = 'L' の場合、 $\text{sub}(A)$ の下三角部分が格納される。 $\text{sub}(B)$ は $LL^H$ として因子分解される。
<i>n</i>	(グローバル) INTEGER。 行列 $\text{sub}(A)$ と $\text{sub}(B)$ の次数 ( $n \geq 0$ )。
<i>a</i>	(ローカル) COMPLEX (pchegst の場合) DOUBLE COMPLEX (pzhegst の場合)。 ローカルメモリにある、次元 ( <i>lld_a</i> , <i>LOCc(ja+n-1)</i> ) の配列へのポインタ。呼び出し時に、配列は $n \times n$ のエルミート分散行列 $\text{sub}(A)$ のローカル部分を含む。 <i>uplo</i> = 'U' の場合、 $\text{sub}(A)$ の先頭の $n \times n$ 上三角部分

は行列の上三角部分を含む。厳密な下三角部分は参照されない。  
`uplo = 'L'` の場合、`sub(A)` の先頭の  $n \times n$  下三角部分は行列の下三角部分を含む。厳密な上三角部分は参照されない。

<code>ia, ja</code>	(グローバル) INTEGER。 それぞれ、部分行列 <i>A</i> の最初の行と最初の列を示す、グローバル配列 <i>a</i> の行インデックスと列インデックス。
<code>desca</code>	(グローバルおよびローカル) INTEGER。配列、次元は ( <code>dlen_</code> )。 分散行列 <i>A</i> の配列ディスクリプタ。
<code>b</code>	(ローカル) COMPLEX ( <code>pchegst</code> の場合) DOUBLE COMPLEX ( <code>pzhegst</code> の場合)。 ローカルメモリにある、次元 ( <code>lld_b</code> , <code>LOCc(jb+n-1)</code> ) の配列へのポインタ。呼び出し時に、配列は、 <code>p?potrf</code> によって返される <code>sub(B)</code> のコレスキー因子分解で得られた三角係数を含む。
<code>ib, jb</code>	(グローバル) INTEGER。 それぞれ、部分行列 <i>B</i> の最初の行と最初の列を示す、グローバル配列 <i>b</i> の行インデックスと列インデックス。
<code>descb</code>	(グローバルおよびローカル) INTEGER。配列、次元は ( <code>dlen_</code> )。 分散行列 <i>B</i> の配列ディスクリプタ。

## 出力パラメータ

<code>a</code>	終了時に、 <code>info = 0</code> の場合、変換後の行列が <code>sub(A)</code> と同じ形式で格納される。
<code>scale</code>	(グローバル) REAL ( <code>pchegst</code> の場合) DOUBLE PRECISION ( <code>pzhegst</code> の場合)。  このルーチンで実行されたスケーリングを補正するための固有値。 <i>scale</i> は将来の拡張のために用意されており、現在は常に 1.0 として返される。
<code>info</code>	(グローバル) INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> < 0 の場合、 <i>i</i> 番目の引数が配列で、 <i>j</i> 番目の値が不正だった場合、 <i>info</i> = -( <i>i</i> 100+ <i>j</i> )。 <i>i</i> 番目の引数がスカラで値が不正だった場合、 <i>info</i> = - <i>i</i> 。

## ドライバルーチン

表 6-8 に、連立 1 次方程式、最小二乗問題、標準固有値と特異値問題、および汎用対称固有値問題を解くための ScaLAPACK ドライバルーチンを示す。

表 6-8 ScaLAPACK ドライバルーチン

問題のタイプ	行列のタイプ、 格納形式	ドライバ
1 次方程式	一般 (部分ピボット演算)	<a href="#">p?gesv</a> (簡易ドライバ) <a href="#">p?gesvx</a> (高度ドライバ)
	一般帯 (部分ピボット演算)	<a href="#">p?gbsv</a> (簡易ドライバ)
	一般帯 (ピボット演算なし)	<a href="#">p?dbsv</a> (簡易ドライバ)
	一般三重対角 (ピボット演算なし)	<a href="#">p?dtsv</a> (簡易ドライバ)
	対称 / エルミート正定値	<a href="#">p?posv</a> (簡易ドライバ) <a href="#">p?posvx</a> (高度ドライバ)
	対称 / エルミート正定値 (帯形式)	<a href="#">p?pbsv</a> (簡易ドライバ)
	対称 / エルミート正定値 (三重対角形式)	<a href="#">p?ptsv</a> (簡易ドライバ)
	線形最小二乗問題	<a href="#">p?gels</a>
	対称固有値問題	<a href="#">p?syev</a> (簡易ドライバ) <a href="#">p?syevx</a> / <a href="#">p?heevx</a> (高度ドライバ)
特異値分解	一般的な $m \times n$	<a href="#">p?gesvd</a>
汎用対称固有値問題	対称 / エルミート、単一行列 正定値	<a href="#">p?sygvx</a> / <a href="#">p?hegvx</a> (高度ドライバ)

## p?gesv

正方分散行列を係数行列とする、複数の右辺を持つ連立 1 次方程式の解を計算する。

### 構文

```
call psgesv(n, nrhs, a, ia, ja, desca, ipiv, b, ib, jb, descb, info)
call pdgesv(n, nrhs, a, ia, ja, desca, ipiv, b, ib, jb, descb, info)
call pcgesv(n, nrhs, a, ia, ja, desca, ipiv, b, ib, jb, descb, info)
call pzgesv(n, nrhs, a, ia, ja, desca, ipiv, b, ib, jb, descb, info)
```

### 説明

p?gesv は、実 / 複素連立 1 次方程式  $\text{sub}(A) * X = \text{sub}(B)$  の解を計算する。ここで、 $\text{sub}(A) = A(\text{ia}:\text{ia}+n-1, \text{ja}:\text{ja}+n-1)$  は  $n \times n$  の分散行列、 $X$  と  $\text{sub}(B) = B(\text{ib}:\text{ib}+n-1, \text{jb}:\text{jb}+nrhs-1)$  は  $n \times nrhs$  の分散行列である。

部分的なピボット演算と行の交換による LU 分解を使用して、 $\text{sub}(A)$  を  $\text{sub}(A) = P L U$  として因子分解する。ここで、 $P$  は置換行列、 $L$  は単位下三角行列、 $U$  は上三角行列である。 $L$  と  $U$  は  $\text{sub}(A)$  に格納される。次に、 $\text{sub}(A)$  の因子分解された形式を使用して、連立方程式  $\text{sub}(A) * X = \text{sub}(B)$  を解く。

### 入カパラメータ

**n** (グローバル) INTEGER。処理される行数と列数。  
すなわち、分散部分行列  $\text{sub}(A)$  の次数 ( $n \geq 0$ )。

**nrhs** (グローバル) INTEGER。右辺の数。  
分散部分行列  $B$  と  $X$  の列の数 ( $nrhs \geq 0$ )。

**a, b** (ローカル)  
REAL (psgesv の場合)  
DOUBLE PRECISION (pdgesv の場合)  
COMPLEX (pcgesv の場合)  
DOUBLE COMPLEX (pzgesv の場合)。  
それぞれ、ローカルメモリにある、ローカル次元の配列  
 $a(\text{lld\_a}, \text{LOC}_c(\text{ja}+n-1))$  と  $b(\text{lld\_b}, \text{LOC}_c(\text{jb}+nrhs-1))$  へのポインタ。

呼び出し時に、配列  $a$  は因子分解の対象となる  $n \times n$  の分散行列  $\text{sub}(A)$  のローカル部分を含む。



呼び出し時に、配列  $b$  は右辺の分散行列  $\text{sub}(B)$  のローカル部分を含む。

$ia, ja$	(グローバル) INTEGER。 それぞれ、 $\text{sub}(A)$ の最初の行と最初の列を示す、グローバル配列 $A$ の行インデックスと列インデックス。
$desca$	(グローバルおよびローカル) INTEGER。配列、次元は $(dlen\_)$ 。 分散行列 $A$ の配列ディスクリプタ。
$ib, jb$	(グローバル) INTEGER。 それぞれ、 $\text{sub}(B)$ の最初の行と最初の列を示す、グローバル配列 $B$ の行インデックスと列インデックス。
$descb$	(グローバルおよびローカル) INTEGER。配列、次元は $(dlen\_)$ 。 分散行列 $B$ の配列ディスクリプタ。

### 出力パラメータ

$a$	$\text{sub}(A) = PLU$ の因子分解で得られた係数 $L$ と $U$ によって上書きされる。 $L$ の単位対角成分は格納されない。
$b$	解の分散行列 $X$ によって上書きされる。
$ipiv$	(ローカル) INTEGER。配列。 $ipiv$ の次元は $(LOC_r(m_a) + mb_a)$ 。 この配列には、ピボット情報が格納される。 行列の (ローカル) 行 $i$ は、(グローバル) 行 $ipiv(i)$ と交換される。 この配列は、分散行列 $A$ に関連付けられる。
$info$	(グローバル) INTEGER。 $info = 0$ の場合、実行は正常に終了したことを示す。  $info < 0$ の場合： $i$ 番目の引数が配列で、 $j$ 番目の値が不正だった場合、 $info = -(i*100+j)$ 。 $i$ 番目の引数がスカラで値が不正だった場合、 $info = -i$ 。  $info > 0$ の場合： $info = k$ の場合、 $U(ia+k-1, ja+k-1)$ は完全にゼロである。因子分解は完了したが、係数 $U$ が完全に特異であるため、解を計算できなかった。

## p?gesvx

LU 因子分解を使用して、正方行列  $A$  を係数行列とする、複数の右辺を持つ連立 1 次方程式の解を計算し、解の誤差範囲を示す。

### 構文

```
call psgesvx(fact, trans, n, nrhs, a, ia, ja, desca, af, iaf, jaf,
             descf, ipiv, equed, r, c, b, ib, jb, descb, x, ix, jx, descx, rcond,
             ferr, berr, work, lwork, iwork, liwork, info)
call pdgesvx(fact, trans, n, nrhs, a, ia, ja, desca, af, iaf, jaf,
             descf, ipiv, equed, r, c, b, ib, jb, descb, x, ix, jx, descx, rcond,
             ferr, berr, work, lwork, iwork, liwork, info)
call pcgesvx(fact, trans, n, nrhs, a, ia, ja, desca, af, iaf, jaf,
             descf, ipiv, equed, r, c, b, ib, jb, descb, x, ix, jx, descx, rcond,
             ferr, berr, work, lwork, rwork, lrwork, info)
call pzgesvx(fact, trans, n, nrhs, a, ia, ja, desca, af, iaf, jaf,
             descf, ipiv, equed, r, c, b, ib, jb, descb, x, ix, jx, descx, rcond,
             ferr, berr, work, lwork, rwork, lrwork, info)
```

### 説明

このルーチンは、LU 因子分解を使用して、実 / 複素連立 1 次方程式  $AX=B$  の解を計算する。ここで、 $A$  は  $n \times n$  の部分行列  $A(ia:ia+n-1, ja:ja+n-1)$ 、 $B$  は  $n \times nrhs$  の部分行列  $B(ib:ib+n-1, jb:jb+nrhs-1)$ 、および  $X$  は  $n \times nrhs$  の部分行列  $X(ix:ix+n-1, jx:jx+nrhs-1)$  を表す。

このルーチンは、解の誤差範囲と条件数の推定値も示す。

以下の説明で、 $af$  は部分配列  $af(iaf:iaf+n-1, jaf:jaf+n-1)$  を表す。

ルーチン ?gesvx は、以下の手順を実行する。

1.  $fact = 'E'$  の場合、連立方程式を平衡化するための実スケール係数  $R$  と  $C$  を計算する。

$trans = 'N'$ :  $diag(R) * A * diag(C) * diag(C)^{-1} * X = diag(R) * B$

$trans = 'T'$ :  $(diag(R) * A * diag(C))^T * diag(R)^{-1} * X = diag(C) * B$

$trans = 'C'$ :  $(diag(R) * A * diag(C))^H * diag(R)^{-1} * X = diag(C) * B$

連立方程式が平衡化されるかどうかは、行列  $A$  のスケーリングによって決まる。平衡化が行われる場合、 $A$  は  $\text{diag}(R) * A * \text{diag}(C)$  によって上書きされ、 $B$  は  $\text{diag}(R) * B$  ( $\text{trans} = 'N'$  の場合) または  $\text{diag}(C) * B$  ( $\text{trans} = 'T'$  または  $'C'$  の場合) によって上書きされる。

2.  $\text{fact} = 'N'$  または  $'E'$  の場合、 $LU$  分解を使用して、( $\text{fact} = 'E'$  の場合は平衡化の後に) 行列  $A$  を  $A = P L U$  として因子分解する。ここで、 $P$  は置換行列、 $L$  は単位下三角行列、 $U$  は上三角行列である。

3.  $A$  の因子分解された形式を使用して、行列  $A$  の条件数を推定する。条件数の逆数がマシンの精度より小さい場合、ステップ 4 から 6 はスキップされる。

4.  $A$  の因子分解された形式を使用して、連立方程式を  $X$  について解く。

5. 計算された解の行列の精度を繰り返し改善し、解の誤差範囲と後退誤差の推定値を計算する。

6. 平衡化が行われている場合、平衡化が行われる前の元の連立方程式を解くために、 $\text{diag}(C)$  ( $\text{trans} = 'N'$  の場合) または  $\text{diag}(R)$  ( $\text{trans} = 'T'$  または  $'C'$  の場合) によって、行列  $X$  を事前乗算する。

## 入力パラメータ

*fact* (グローバル) CHARACTER\*1。

'F'、'N'、または 'E' でなければならない。

呼び出し時に、行列  $A$  の因子分解された形式が与えられるかどうか、与えられない場合は、行列  $A$  を因子分解する前に平衡化するかどうかを指定する。

$\text{fact} = 'F'$  の場合、呼び出し時に、 $af$  と  $ipiv$  は、 $A$  の因子分解された形式を含む。 $equed$  が 'N' でない場合、行列  $A$  は、 $r$  と  $c$  で指定されたスケール係数によって平衡化されている。配列  $a$ 、 $af$ 、および  $ipiv$  は変更されない。

$\text{fact} = 'N'$  の場合、行列  $A$  を  $af$  にコピーし、因子分解を実行する。 $\text{fact} = 'E'$  の場合、必要に応じて行列  $A$  を平衡化してから、 $af$  にコピーし、因子分解を実行する。

*trans* (グローバル) CHARACTER\*1。

'N'、'T'、または 'C' でなければならない。

連立方程式の形式を指定する。

$\text{trans} = 'N'$  の場合、連立方程式の形式は  $A X = B$  (転置なし) である。

$\text{trans} = 'T'$  の場合、連立方程式の形式は  $A^T X = B$  (転置) である。

$\text{trans} = 'C'$  の場合、連立方程式の形式は  $A^H X = B$  (共役転置) である。

<i>n</i>	(グローバル) INTEGER。1 次方程式の数。 部分行列 <i>A</i> の次数 ( $n \geq 0$ )。
<i>nrhs</i>	(グローバル) INTEGER。右辺の数。 分散部分行列 <i>B</i> と <i>X</i> の列の数 ( $nrhs \geq 0$ )。
<i>a, af, b, work</i>	(ローカル) REAL (psgesvx の場合) DOUBLE PRECISION (pdgesvx の場合) COMPLEX (pcgesvx の場合) DOUBLE COMPLEX (pzgesvx の場合)。 それぞれ、ローカルメモリにある、ローカル次元の配列 <i>a</i> ( <i>lld_a</i> , <i>LOC<sub>c</sub></i> ( <i>ja+n-1</i> )), <i>af</i> ( <i>lld_af</i> , <i>LOC<sub>c</sub></i> ( <i>ja+n-1</i> )), <i>b</i> ( <i>lld_b</i> , <i>LOC<sub>c</sub></i> ( <i>jb+nrhs-1</i> )), および <i>work</i> ( <i>lwork</i> ) へのポインタ。  配列 <i>a</i> には、行列 <i>A</i> が格納される。 <i>fact</i> = 'F' かつ <i>equed</i> が 'N' でない 場合、 <i>A</i> は、 <i>r</i> または <i>c</i> 、あるいはその両方のスケール係数によって平 衡化されている。  配列 <i>af</i> は、 <i>fact</i> = 'F' の場合は入力引数になる。この場合、配列は、 呼び出し時に、行列 <i>A</i> の因子分解された形式、すなわち、 <a href="#">p?getrf</a> に よる因子分解 $A = PLU$ で得られた係数 <i>L</i> と <i>U</i> を含む。 <i>equed</i> が 'N' で ない場合、 <i>af</i> は、平衡化された行列 <i>A</i> の因子分解された形式になる。  呼び出し時に、配列 <i>b</i> は行列 <i>B</i> を含む。この行列の列は連立方程式の 右辺である。  <i>work</i> (*) は、ワークスペース配列。 <i>work</i> の次元は、( <i>lwork</i> )。
<i>ia, ja</i>	(グローバル) INTEGER。 それぞれ、部分行列 <i>A</i> ( <i>ia:ia+n-1, ja:ja+n-1</i> ) の最初の行と最初の列 を示すグローバル配列 <i>A</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。 分散行列 <i>A</i> の配列ディスクリプタ。
<i>iaf, jaf</i>	(グローバル) INTEGER。 それぞれ、部分配列 <i>af</i> ( <i>iaf:iaf+n-1, jaf:jaf+n-1</i> ) の最初の行と最 初の列を示すグローバル配列 <i>af</i> の行インデックスと列インデックス。
<i>descaf</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。 分散行列 <i>AF</i> の配列ディスクリプタ。

<i>ib, jb</i>	(グローバル) INTEGER。 それぞれ、部分行列 $B(ib:ib+n-1, jb:jb+nrhs-1)$ の最初の行と最初の列を示すグローバル配列 $B$ の行インデックスと列インデックス。
<i>descb</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。分散行列 $B$ の配列ディスクリプタ。
<i>ipiv</i>	(ローカル) INTEGER。配列。 <i>ipiv</i> の次元は $(LOC_r(m_a) + mb_a)$ 。 配列 <i>ipiv</i> は、 <i>fact</i> = 'F' の場合は入力引数になる。 呼び出し時に、この配列は、 <a href="#">p?getrf</a> による因子分解 $A = PLU$ で得られたピボットのインデックスを含む。行列の (ローカル) 行 <i>i</i> は、(グローバル) 行 <i>ipiv</i> ( <i>i</i> ) と交換される。 この配列は、 $A(ia:ia+n-1, *)$ とアライメントされなければならない。
<i>equed</i>	(グローバル) CHARACTER*1。 'N'、'R'、'C'、または 'B' でなければならない。 <i>equed</i> は、 <i>fact</i> = 'F' の場合は入力引数になる。この引数は、実行された平衡化の形式を指定する。 <i>equed</i> = 'N' の場合、平衡化は行われていない ( <i>fact</i> = 'N' の場合、常に真)。 <i>equed</i> = 'R' の場合、行の平衡化が行われた。すなわち、 $A$ が $diag(r)$ によって事前乗算された。 <i>equed</i> = 'C' の場合、列の平衡化が行われた。すなわち、 $A$ が $diag(c)$ によって事前乗算された。 <i>equed</i> = 'B' の場合、行と列の平衡化が行われた。すなわち、 $A$ が $diag(r)*A*diag(c)$ によって置き換えられた。
<i>r, c</i>	(ローカル) REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 配列、次元はそれぞれ $LOC_r(m_a)$ および $LOC_c(n_a)$ 。 配列 <i>r</i> には $A$ の行のスケール係数が格納され、配列 <i>c</i> は $A$ の列のスケール係数が格納される。これらの配列は、 <i>fact</i> = 'F' の場合にのみ入力引数になり、それ以外の場合は出力引数になる。 <i>equed</i> = 'R' または 'B' の場合、 $A$ は $diag(r)$ によって左辺で乗算される。 <i>equed</i> = 'N' または 'C' の場合、 <i>r</i> は使用されない。 <i>fact</i> = 'F' かつ <i>equed</i> = 'R' または 'B' の場合、 <i>r</i> の各成分は正の値でなければならない。  <i>equed</i> = 'C' または 'B' の場合、 $A$ は $diag(c)$ によって右辺で乗算される。 <i>equed</i> = 'N' または 'R' の場合、 <i>c</i> は使用されない。 <i>fact</i> = 'F' かつ <i>equed</i> = 'C' または 'B' の場合、 <i>c</i> の各成分は正の値でなければならない。

配列  $r$  はすべてのプロセス列に複製され、分散行列  $A$  とアライメントされる。

配列  $c$  はすべてのプロセス行に複製され、分散行列  $A$  とアライメントされる。

$ix, jx$	(グローバル) INTEGER。 それぞれ、部分行列 $X(ix:ix+n-1, jx:jx+nrhs-1)$ の最初の行と最初の列を示すグローバル配列 $X$ の行インデックスと列インデックス。
$descx$	(グローバルおよびローカル) INTEGER。配列、次元は ( $dlen\_$ )。分散行列 $X$ の配列ディスクリプタ。
$lwork$	(ローカルまたはグローバル) INTEGER。配列 $work$ の次元。 $\max(p?gecon(lwork), p?gerfs(lwork)) + LOC_r(n\_a)$ 以上でなければならない。
$iwork$	(ローカル、psgesvx/pdgesvx のみ) INTEGER。ワークスペース配列。 $iwork$ の次元は ( $liwork$ )。
$liwork$	(ローカル、psgesvx/pdgesvx のみ) INTEGER。配列 $iwork$ の次元。 $LOC_r(n\_a)$ 以上でなければならない。
$rwork$	(ローカル) REAL (pcgesvx の場合) DOUBLE PRECISION (pzgesvx の場合)。 複素数型の場合にのみ使用されるワークスペース配列。 $rwork$ の次元は ( $lrwork$ )。
$lrwork$	(ローカルまたはグローバル、pcgesvx/pzgesvx のみ) INTEGER。 配列 $rwork$ の次元。 $2*LOC_c(n\_a)$ 以上でなければならない。

## 出力パラメータ

$x$	(ローカル) REAL (psgesvx の場合) DOUBLE PRECISION (pdgesvx の場合) COMPLEX (pcgesvx の場合) DOUBLE COMPLEX (pzgesvx の場合)。 ローカルメモリにある、ローカル次元 $x(1:d_x, LOC_c(jx+nrhs-1))$ の配列へのポインタ。  $info=0$ の場合、配列 $x$ には、元の連立方程式の解の行列 $X$ が格納される。ただし、 $equed \neq 'N'$ の場合、 $A$ と $B$ は終了時に変更され、平衡化された連立方程式の解は次のようになる。
-----	---

	$\text{diag}(C)^{-1} * X$ ( $\text{trans} = 'N'$ かつ $\text{equed} = 'C'$ または $'B'$ の場合) $\text{diag}(R)^{-1} * X$ ( $\text{trans} = 'T'$ または $'C'$ かつ $\text{equed} = 'R'$ または $'B'$ の場合)。
<i>a</i>	配列 <i>a</i> は、 $\text{fact} = 'F'$ または $'N'$ の場合、あるいは $\text{fact} = 'E'$ で $\text{equed} = 'N'$ の場合、終了時に変更されない。 $\text{equed} \neq 'N'$ の場合、 <i>A</i> は終了時に次のようにスケーリングされる。 $\text{equed} = 'R'$ : $A = \text{diag}(R) * A$ $\text{equed} = 'C'$ : $A = A * \text{diag}(c)$ $\text{equed} = 'B'$ : $A = \text{diag}(R) * A * \text{diag}(c)$
<i>af</i>	$\text{fact} = 'N'$ または $'E'$ の場合、 <i>af</i> は出力引数になる。この引数は、終了時に、元の行列 <i>A</i> ( $\text{fact} = 'N'$ の場合) または平衡化された行列 <i>A</i> ( $\text{fact} = 'E'$ の場合) の因子分解 $A = P L U$ で得られた係数 <i>L</i> と <i>U</i> を返す。平衡化された行列の形式については、 <i>a</i> の説明を参照のこと。
<i>b</i>	$\text{trans} = 'N'$ かつ $\text{equed} = 'R'$ または $'B'$ の場合、 $\text{diag}(R) * B$ によって上書きされる。 $\text{trans} = 'T'$ かつ $\text{equed} = 'C'$ または $'B'$ の場合、 $\text{diag}(c) * B$ によって上書きされる。 $\text{equed} = 'N'$ の場合は変更されない。
<i>r, c</i>	これらの配列は、 $\text{fact} \neq 'F'$ の場合は出力引数になる。 「入力引数」セクションの <i>r, c</i> の説明を参照。
<i>rcond</i>	(グローバル) REAL (単精度の場合)。 DOUBLE PRECISION (倍精度の場合)。 平衡化後の行列 <i>A</i> の条件数の逆数の推定値 (平衡化が行われる場合)。推定値のアンダーフローが発生した場合、 $\text{rcond} = 0$ に設定される。この場合、行列は有効な精度で特異になる。ただし、 <i>rcond</i> が (有効な精度で) 1.0 より小さい場合、条件の良くない行列または特異な行列である。
<i>ferr, berr</i>	(ローカル) REAL (単精度の場合)。 DOUBLE PRECISION (倍精度の場合)。 配列、次元はそれぞれ $LOC_c(n_b)$ 。各解ベクトルの成分ごとの前進誤差と相対後退誤差を格納する。 配列 <i>ferr</i> と <i>berr</i> はすべてのプロセス行に複製され、行列 <i>B</i> および <i>X</i> とアライメントされる。
<i>ipiv</i>	$\text{fact} = 'N'$ または $'E'$ の場合、 <i>ipiv</i> は出力引数になる。この引数には、元の行列 <i>A</i> ( $\text{fact} = 'N'$ の場合) または平衡化された行列 <i>A</i> ( $\text{fact} = 'E'$ の場合) の因子分解 $A = P L U$ で得られたピボットのインデックスが格納される。

<i>eques</i>	<i>fact</i> ≠ 'F' の場合、 <i>eques</i> は出力引数になる。この引数は、実行された平衡化の形式を指定する (「入力引数」セクションの <i>eques</i> の説明を参照)。
<i>work(1)</i>	<i>info</i> = 0 の場合、終了時に <i>work(1)</i> は、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値を返す。
<i>iwork(1)</i>	<i>info</i> = 0 の場合、終了時に <i>iwork(1)</i> は、最適なパフォーマンスを得るために必要な <i>liwork</i> の最小値を返す。
<i>rwork(1)</i>	<i>info</i> = 0 の場合、終了時に <i>rwork(1)</i> は、最適なパフォーマンスを得るために必要な <i>lrwork</i> の最小値を返す。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。  <i>info</i> < 0 の場合： <i>i</i> 番目の引数が配列で、 <i>j</i> 番目の値が不正だった場合、 <i>info</i> = -( <i>i</i> *100+ <i>j</i> )。 <i>i</i> 番目の引数がスカラで値が不正だった場合、 <i>info</i> = - <i>i</i> 。 <i>info</i> = <i>i</i> で、 <i>i</i> ≤ <i>n</i> の場合、 <i>U</i> ( <i>i</i> , <i>i</i> ) は完全にゼロである。因子分解は完了したが、係数 <i>U</i> が完全に特異であるため、解と誤差範囲を計算できなかった。 <i>info</i> = <i>i</i> で、 <i>i</i> = <i>n</i> + 1 の場合、 <i>U</i> は特異でないが、 <i>rcond</i> はマシンの精度より小さい。因子分解は完了したが、行列は有効な精度で完全に特異であるため、解と誤差範囲を計算できなかった。

---

### p?gbsv

一般帯分散行列を係数行列とする、複数の右辺を持つ連立 1 次方程式の解を計算する。

---

#### 構文

```
call psgbsv(n, bwl, bwu, nrhs, a, ja, desca, ipiv, b, ib, descb, work,
           lwork, info)
call pdgbsv(n, bwl, bwu, nrhs, a, ja, desca, ipiv, b, ib, descb, work,
           lwork, info)
call pcgbsv(n, bwl, bwu, nrhs, a, ja, desca, ipiv, b, ib, descb, work,
           lwork, info)
call pzgbsv(n, bwl, bwu, nrhs, a, ja, desca, ipiv, b, ib, descb, work,
           lwork, info)
```



## 説明

p?gbsv は、実 / 複素連立 1 次方程式

$$\text{sub}(A) * X = \text{sub}(B)$$

の解を計算する。ここで、 $\text{sub}(A) = A(1:n, ja:ja+n-1)$  は  $bw1$  劣対角と  $bwu$  優対角を持つ  $n \times n$  の実 / 複素一般帯分散行列、 $X$  および  $\text{sub}(B) = B(ib:ib+n-1, 1:nrhs)$  は  $n \times nrhs$  の分散行列である。

部分的なピボット演算と行の交換による  $LU$  分解を使用して、 $\text{sub}(A)$  を  $\text{sub}(A) = P L U Q$  として因子分解する。ここで、 $P$  と  $Q$  は置換行列、 $L$  と  $U$  はそれぞれ帯形式の下三角行列、上三角行列である。行列  $Q$  は並列化のために列の順序を変更することを表し、 $P$  は数の安定のために部分的なピボット演算を使用して行の順序を変更することを表す。

## 入力パラメータ

- $n$  (グローバル) INTEGER。処理される行数と列数。  
すなわち、分散部分行列  $\text{sub}(A)$  の次数 ( $n \geq 0$ )。
- $bw1$  (グローバル) INTEGER。  
 $A$  の帯内の劣対角成分の数 ( $0 \leq bw1 \leq n-1$ )。
- $bwu$  (グローバル) INTEGER。  
 $A$  の帯内の優対角成分の数 ( $0 \leq bwu \leq n-1$ )。
- $nrhs$  (グローバル) INTEGER。右辺の数。  
分散部分行列  $\text{sub}(B)$  の列数 ( $nrhs \geq 0$ )。
- $a, b$  (ローカル)  
REAL (psgbsv の場合)  
DOUBLE PRECISION (pdgbsv の場合)  
COMPLEX (pcgbsv の場合)  
DOUBLE COMPLEX (pzgbsv の場合)。  
それぞれ、ローカルメモリにある、ローカル次元の配列  
 $a(lld\_a, LOC_c(ja+n-1))$  と  $b(lld\_b, LOC_c(nrhs))$  へのポインタ。  
呼び出し時に、配列  $a$  はグローバル配列  $A$  のローカル部分を含む。  
呼び出し時に、配列  $b$  は右辺の分散行列  $\text{sub}(B)$  のローカル部分を含む。
- $ja$  (グローバル) INTEGER。  
( $A$  のすべて、または  $A$  の部分行列で) 処理される行列の先頭を指すグローバル配列  $A$  のインデックス。

<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。 分散行列 <i>A</i> の配列ディスクリプタ。 <i>desca</i> ( <i>dtype_</i> ) = 501 の場合、 <i>dlen_</i> ≥ 7。 <i>desca</i> ( <i>dtype_</i> ) = 1 の場合、 <i>dlen_</i> ≥ 9。
<i>ib</i>	(グローバル) INTEGER。 ( <i>B</i> のすべて、または <i>B</i> の部分行列で) 処理される行列の最初の行を指すグローバル配列 <i>B</i> の行インデックス。
<i>descb</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。 分散行列 <i>B</i> の配列ディスクリプタ。 <i>descb</i> ( <i>dtype_</i> ) = 502 の場合、 <i>dlen_</i> ≥ 7。 <i>descb</i> ( <i>dtype_</i> ) = 1 の場合、 <i>dlen_</i> ≥ 9。
<i>work</i>	(ローカル) REAL (psgbsv の場合) DOUBLE PRECISION (pdgbsv の場合) COMPLEX (pcgbsv の場合) DOUBLE COMPLEX (pzgbsv の場合)。 次元 ( <i>lwork</i> ) のワークスペース配列。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 配列 <i>work</i> のサイズは、 $lwork \geq (NB+bwu)*(bwl+bwu)+6*(bwl+bwu)*(bwl+2*bwu) + \max(nrhs*(NB+2*bwl+4*bwu), 1)$ でなければならない。

### 出力パラメータ

<i>a</i>	終了時に、因子分解の詳細が格納される。 因子分解の結果は、LAPACK から返される因子分解とは異なる点に注意する。並列用の行列では、追加の置換が実行される。
<i>b</i>	終了時に、この配列には解の分散行列 <i>X</i> のローカル部分が格納される。
<i>ipiv</i>	(ローカル) INTEGER。配列。 <i>ipiv</i> の次元は <i>desca</i> ( <i>NB</i> ) 以上でなければならない。 ローカル因子分解用のピボットのインデックスを含む。因子分解および解の算出を行う間、この配列の内容は変更すべきでない点に注意する。
<i>work</i> (1)	終了時に、 <i>work</i> (1) には、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が格納される。

*info* INTEGER。 *info* = 0 の場合、実行は正常に終了したことを示す。  
*info* < 0 の場合：  
*i* 番目の引数が配列で、*j* 番目の値が不正だった場合、  
*info* = -(*i*\*100+*j*)。 *i* 番目の引数がスカラーで値が不正だった場合、  
*info* = -*i*。  
*info* > 0:  
*info* = *k* ≤ NPROCS の場合、プロセッサ *info* に格納され、ローカルに  
 因子分解された部分行列が非特異でないため、因子分解を完了できな  
 かったことを示す。  
*info* = *k* > NPROCS の場合、他のプロセッサとの対話を表すプロセッサ  
*info*-NPROCS に格納された部分行列が非特異でないため、因子分解を  
 完了できなかったことを示す。

## p?dbsv

一般帯形式の連立 1 次方程式を解く。

### 構文

```
call psdbsv(n, bwl, bwu, nrhs, a, ja, desca, b, ib, descb, work, lwork,
            info)
call pddbsv(n, bwl, bwu, nrhs, a, ja, desca, b, ib, descb, work, lwork,
            info)
call pcdbsv(n, bwl, bwu, nrhs, a, ja, desca, b, ib, descb, work, lwork,
            info)
call pzdbsv(n, bwl, bwu, nrhs, a, ja, desca, b, ib, descb, work, lwork,
            info)
```

### 説明

このルーチンは、次の連立 1 次方程式を解く。

$$A(1:n, ja:ja+n-1) * X = B(ib:ib+n-1, 1:nrhs)$$

ここで、 $A(1:n, ja:ja+n-1)$  はバンド幅 *bwl*、*bwu* の  $n \times n$  の実 / 複素帯形式の対角優位分散行列である。

行列の順序を変更して  $LU$  に因子分解するために、ピボット演算を用いないガウス消去が使用される。

## 入力パラメータ

<i>n</i>	(グローバル) INTEGER。 分散部分行列 <i>A</i> の次数 ( $n \geq 0$ )。
<i>bw1</i>	(グローバル) INTEGER。 劣対角成分の数。 $0 \leq bw1 \leq n-1$ 。
<i>bwu</i>	(グローバル) INTEGER。 劣対角成分の数。 $0 \leq bwu \leq n-1$ 。
<i>nrhs</i>	(グローバル) INTEGER。 右辺の数。分散部分行列 <i>B</i> の列数 ( $nrhs \geq 0$ )。
<i>a</i>	(ローカル)。 REAL (psdbsv の場合) DOUBLE PRECISION (pddbsv の場合) COMPLEX (pcdbsv の場合) DOUBLE COMPLEX (pzdbsv の場合)。 ローカルメモリにある、第 1 次元が $lld\_a \geq (bw1+bwu+1)$ の配列へのポインタ ( <i>desca</i> に格納)。呼び出し時に、この配列は、分散行列のローカル部分を含む。
<i>ja</i>	(グローバル) INTEGER。 ( <i>A</i> のすべて、または <i>A</i> の部分行列で) 処理される行列の先頭を指すグローバル配列 <i>a</i> のインデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen</i> )。 1d type ( <i>dtype_a</i> = 501 または 502) の場合、 $dlen \geq 7$ 。 2d type ( <i>dtype_a</i> = 1) の場合、 $dlen \geq 9$ 。 分散行列 <i>A</i> の配列ディスクリプタ。 <i>A</i> のマッピング情報をメモリに格納する。
<i>b</i>	(ローカル) REAL (psdbsv の場合) DOUBLE PRECISION (pddbsv の場合) COMPLEX (pcdbsv の場合) DOUBLE COMPLEX (pzdbsv の場合)。 ローカルメモリにある、ローカル・リーディング・ディメンジョン $lld\_b \geq NB$ の配列へのポインタ。呼び出し時に、この配列は、右辺 <i>B</i> ( <i>ib:ib+n-1, 1:nrhs</i> ) のローカル部分を含む。
<i>ib</i>	(グローバル) INTEGER。 ( <i>b</i> のすべて、または <i>B</i> の部分行列で) 処理される行列の最初の行を指すグローバル配列 <i>b</i> の行インデックス。

<i>desb</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen</i> )。 1d type ( <i>dtype_b</i> = 502) の場合、 <i>dlen</i> ≥ 7。 2d type ( <i>dtype_b</i> = 1) の場合、 <i>dlen</i> ≥ 9。 分散行列 <i>B</i> の配列ディスクリプタ。 <i>B</i> のマッピング情報をメモリに格納する。
<i>work</i>	(ローカル)。 REAL ( <i>psdbsv</i> の場合) DOUBLE PRECISION ( <i>pddbsv</i> の場合) COMPLEX ( <i>pcdbsv</i> の場合) DOUBLE COMPLEX ( <i>pzdbsv</i> の場合)。 テンポラリ・ワークスペース。このスペースは、ルーチンの呼び出しの間に上書きされる。 <i>work</i> のサイズは <i>lwork</i> で与えられる。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 ユーザ入力ワークスペース <i>work</i> のサイズ。 <i>lwork</i> が小さすぎる場合、エラーコードが返され、エラーにならない最小のサイズが <i>work</i> (1) に返される。 $lwork \geq NB(bw1+bwu)+6 \max(bw1,bwu)*\max(bw1,bwu) + \max((\max(bw1,bwu)nrhs), \max(bw1,bwu)\max(bw1,bwu))$

### 出力パラメータ

<i>a</i>	終了時に、この配列には因子分解の詳細が格納される。 行列で置換が行われるため、返される係数は LAPACK から返される係数とは異なる点に注意する。
<i>b</i>	終了時に、解の分散行列 <i>X</i> のローカル部分が格納される。
<i>work</i>	終了時に、 <i>work</i> (1) には、 <i>lwork</i> の最小値が格納される。
<i>info</i>	(ローカル) INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> < 0 の場合： <i>i</i> 番目の引数が配列で、 <i>j</i> 番目の値が不正だった場合、 <i>info</i> = -( <i>i</i> *100+ <i>j</i> )。 <i>i</i> 番目の引数がスカラで値が不正だった場合、 <i>info</i> = - <i>i</i> 。 <i>info</i> > 0 の場合： <i>info</i> = <i>k</i> ≤ NPROCS の場合、プロセッサ <i>info</i> に格納され、ローカルに因子分解された部分行列が正定値でないため、因子分解を完了できなかったことを示す。 <i>info</i> = <i>k</i> > NPROCS の場合、他のプロセッサとの対話を表すプロセッサ <i>info</i> -NPROCS に格納された部分行列が正定値ではないため、因子分解を完了できなかったことを示す。

## p?dtsv

三重対角形式の連立1次方程式を解く。

### 構文

```
call psdtsv(n, nrhs, dl, d, du, ja, desca, b, ib, descb, work, lwork, info)
call pddtsv(n, nrhs, dl, d, du, ja, desca, b, ib, descb, work, lwork, info)
call pcdtsv(n, nrhs, dl, d, du, ja, desca, b, ib, descb, work, lwork, info)
call pzdtsv(n, nrhs, dl, d, du, ja, desca, b, ib, descb, work, lwork, info)
```

### 説明

このルーチンは、次の連立1次方程式を解く。

$$A(1:n, ja:ja+n-1) * X = B(ib:ib+n-1, 1:nrhs)$$

ここで、 $A(1:n, ja:ja+n-1)$  は  $n \times n$  の複素対角優位三重対角行列である。

行列の順序を変更して  $LU$  に因子分解するために、ピボット演算を用いないガウス消去が使用される。

### 入力パラメータ

<i>n</i>	(グローバル) INTEGER。 分散部分行列 <i>A</i> の次数 ( $n \geq 0$ )。
<i>nrhs</i>	INTEGER。右辺の数。分散行列 <i>B</i> の列数 ( $nrhs \geq 0$ )。
<i>dl</i>	(ローカル)。 REAL (psdtsv の場合) DOUBLE PRECISION (pddtsv の場合) COMPLEX (pcdtsv の場合) DOUBLE COMPLEX (pzdtsv の場合)。  行列の下対角を格納するグローバル・ベクトルのローカル部分へのポインタ。全体的に、 <i>dl</i> (1) は参照されず、 <i>dl</i> は <i>d</i> とアライメントされなければならない。サイズは <i>desca</i> ( <i>nb_</i> ) 以上でなければならない。
<i>d</i>	(ローカル)。 REAL (psdtsv の場合) DOUBLE PRECISION (pddtsv の場合) COMPLEX (pcdtsv の場合)

	DOUBLE COMPLEX (pzdtsv の場合 )。 行列の主対角を格納するグローバル・ベクトルのローカル部分へのポインタ。
<i>du</i>	( ローカル )。 REAL (psdtsv の場合 ) DOUBLE PRECISION (pddtsv の場合 ) COMPLEX (pcdtsv の場合 ) DOUBLE COMPLEX (pzdtsv の場合 )。 行列の上対角を格納するグローバル・ベクトルのローカル部分へのポインタ。全体的に、 <i>du</i> ( <i>n</i> ) は参照されず、 <i>du</i> は <i>d</i> とアライメントされなければならない。
<i>ja</i>	( グローバル ) INTEGER。 ( <i>A</i> のすべて、または <i>A</i> の部分行列で ) 処理される行列の先頭を指すグローバル配列 <i>a</i> のインデックス。
<i>desca</i>	( グローバルおよびローカル ) INTEGER。配列、次元は ( <i>dlen</i> )。 1d type ( <i>dtype_a</i> = 501 または 502) の場合、 <i>dlen</i> ≥ 7。 2d type ( <i>dtype_a</i> = 1) の場合、 <i>dlen</i> ≥ 9。 分散行列 <i>A</i> の配列ディスクリプタ。 <i>A</i> のマッピング情報をメモリに格納する。
<i>b</i>	( ローカル ) REAL (psdtsv の場合 ) DOUBLE PRECISION (pddtsv の場合 ) COMPLEX (pcdtsv の場合 ) DOUBLE COMPLEX (pzdtsv の場合 )。 ローカルメモリにある、ローカル・リーディング・ディメンジョン $lld\_b \geq NB$ の配列へのポインタ。呼び出し時に、この配列は、右辺 $B(ib:ib+n-1, 1:nrhs)$ のローカル部分を含む。
<i>ib</i>	( グローバル ) INTEGER。 ( <i>b</i> のすべて、または <i>B</i> の部分行列で ) 処理される行列の最初の行を指すグローバル配列 <i>b</i> の行インデックス。
<i>desb</i>	( グローバルおよびローカル ) INTEGER。配列、次元は ( <i>dlen</i> )。 1d type ( <i>dtype_b</i> = 502) の場合、 <i>dlen</i> ≥ 7。 2d type ( <i>dtype_b</i> = 1) の場合、 <i>dlen</i> ≥ 9。 分散行列 <i>B</i> の配列ディスクリプタ。 <i>B</i> のマッピング情報をメモリに格納する。

<i>work</i>	(ローカル)。 REAL (psdtsv の場合) DOUBLE PRECISION (pddtsv の場合) COMPLEX (pcdtsv の場合) DOUBLE COMPLEX (pzdtsv の場合)。 テンポラリ・ワークスペース。このスペースは、ルーチンの呼び出しの間に上書きされる。 <i>work</i> のサイズは <i>lwork</i> で与えられる。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 ユーザ入力ワークスペース <i>work</i> のサイズ。 <i>lwork</i> が小さすぎる場合、エラーコードが返され、エラーにならない最小のサイズが <i>work(1)</i> に返される。 $lwork \geq (12 * NPCOL + 3 * NB) + \max((10 + 2 * \min(100, nrhs)) * NPCOL + 4 * nrhs, 8 * NPCOL)$

## 出力パラメータ

<i>d1</i>	終了時に、行列の係数を含む情報が格納される。
<i>d</i>	終了時に、行列の係数を含む情報が格納される。 サイズは <i>desca(nb_)</i> 以上でなければならない。
<i>du</i>	終了時に、行列の係数を含む情報が格納される。 サイズは <i>desca(nb_)</i> 以上でなければならない。
<i>b</i>	終了時に、解の分散行列 <i>X</i> のローカル部分が格納される。
<i>work</i>	終了時に、 <i>work(1)</i> には、 <i>lwork</i> の最小値が格納される。
<i>info</i>	(ローカル) INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> < 0 の場合： <i>i</i> 番目の引数が配列で、 <i>j</i> 番目の値が不正だった場合、 $info = -(i * 100 + j)$ 。 <i>i</i> 番目の引数がスカラで値が不正だった場合、 $info = -i$ 。 <i>info</i> > 0 の場合： $info = k \leq NPROCS$ の場合、プロセッサ <i>info</i> に格納され、ローカルに因子分解された部分行列が正定値でないため、因子分解を完了できなかったことを示す。 $info = k > NPROCS$ の場合、他のプロセッサとの対話を表すプロセッサ <i>info</i> - <i>NPROCS</i> に格納された部分行列が正定値ではないため、因子分解を完了できなかったことを示す。



## p?posv

対称正定値連立1次方程式を解く。

### 構文

```
call psposv(uplo, n, nrhs, a, ia, ja, desca, b, ib, jb, descb, info)
call pdposv(uplo, n, nrhs, a, ia, ja, desca, b, ib, jb, descb, info)
call pcposv(uplo, n, nrhs, a, ia, ja, desca, b, ib, jb, descb, info)
call pzposv(uplo, n, nrhs, a, ia, ja, desca, b, ib, jb, descb, info)
```

### 説明

このルーチンは、実/複素連立1次方程式を計算する。

$$\text{sub}(A) * X = \text{sub}(B)$$

ここで、 $\text{sub}(A)$  は  $A(ia:ia+n-1, ja:ja+n-1)$  を表し、 $n \times n$  の対称/エルミート分散正定値行列である。 $B(ib:ib+n-1, jb:jb+nrhs-1)$  を表す  $X$  と  $\text{sub}(B)$  は、 $n \times nrhs$  の分散行列である。コレスキー因子分解を使用して  $\text{sub}(A)$  を以下のように因子分解する。

$$\text{sub}(A) = U^T * U \quad (\text{uplo} = 'U' \text{ の場合})$$

$$\text{sub}(A) = L * L^T \quad (\text{uplo} = 'L' \text{ の場合})$$

ここで、 $U$  は上三角行列、 $L$  は下三角行列である。次に、 $\text{sub}(A)$  の因子分解された形式を使用して、連立方程式を解く。

### 入力パラメータ

<code>uplo</code>	(グローバル) CHARACTER。 'U' または 'L' でなければならない。 $\text{sub}(A)$ の上三角部分と下三角部分のどちらが格納されるかを指定する。
<code>n</code>	(グローバル) INTEGER。分散部分行列 $\text{sub}(A)$ の次数 ( $n \geq 0$ )。
<code>nrhs</code>	INTEGER。右辺の数。 分散部分行列 $\text{sub}(B)$ の列数 ( $nrhs \geq 0$ )。
<code>a</code>	(ローカル) REAL (psposv の場合) DOUBLE PRECISION (pdposv の場合) COMPLEX (pcposv の場合) COMPLEX*16 (pzposv の場合)。

ローカルメモリにある、次元 ( $lld\_a, LOCc(ja+n-1)$ ) の配列へのポインタ。呼び出し時に、この配列は、因子分解する  $n \times n$  の対称分散行列  $sub(A)$  のローカル部分を含む。  $uplo = 'U'$  の場合、  $sub(A)$  の先頭の  $n \times n$  上三角部分は行列の上三角部分を含む。厳密な下三角部分は参照されない。  $uplo = 'L'$  の場合、  $sub(A)$  の先頭の  $n \times n$  下三角部分は行列の下三角部分を含む。厳密な上三角部分は参照されない。

<i>ia, ja</i>	(グローバル) INTEGER。 それぞれ、部分行列 <i>A</i> の最初の行と最初の列を示す、グローバル配列 <i>a</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。 分散行列 <i>A</i> の配列ディスクリプタ。
<i>b</i>	(ローカル) REAL (psposv の場合) DOUBLE PRECISION (pdposv の場合) COMPLEX (pcposv の場合) COMPLEX*16 (pzposv の場合)。 ローカルメモリにある、次元 ( $lld\_b, LOC(jb+nrhs-1)$ ) の配列へのポインタ。呼び出し時は、分散行列 $sub(B)$ の右辺のローカル部分。
<i>ib, jb</i>	(グローバル) INTEGER。 それぞれ、部分行列 <i>B</i> の最初の行と最初の列を示す、グローバル配列 <i>b</i> の行インデックスと列インデックス。
<i>descb</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。 分散行列 <i>B</i> の配列ディスクリプタ。

## 出力パラメータ

<i>a</i>	終了時に、 $info = 0$ の場合、この配列には、コレスキー因子分解 $sub(A) = U^H U$ または $LL^H$ で得られた係数 <i>U</i> あるいは <i>L</i> が格納される。
<i>b</i>	終了時に、 $info = 0$ の場合、 $sub(B)$ は解の行列 <i>X</i> によって上書きされる。
<i>info</i>	(グローバル) INTEGER。 $info = 0$ の場合、実行は正常に終了したことを示す。 $info < 0$ の場合： <i>i</i> 番目の引数が配列で、 <i>j</i> 番目の値が不正だった場合、 $info = -(i*100+j)$ 。 <i>i</i> 番目の引数がスカラで値が不正だった場合、 $info = -i$ 。

$info > 0$  の場合 :  
 $info = k$  の場合、次数  $k$  の先頭の小行列式  $A(ia:ia+k-1, ja:ja+k-1)$  が  
 正定値でないため、因子分解を完了できず、解が計算できなかったこ  
 とを示す。

## p?posvx

対称 / エルミート 正定値連立 1 次方程式を解く。

### 構文

```
call psposvx(fact, uplo, n, nrhs, a, ia, ja, desca, af, iaf, jaf, descaf,
             equed, sr, sc, b, ib, jb, descb, x, ix, jx, descx, rcond, ferr, berr,
             work, lwork, iwork, liwork, info)

call pdposvx(fact, uplo, n, nrhs, a, ia, ja, desca, af, iaf, jaf, descaf,
             equed, sr, sc, b, ib, jb, descb, x, ix, jx, descx, rcond, ferr, berr,
             work, lwork, iwork, liwork, info)

call pcposvx(fact, uplo, n, nrhs, a, ia, ja, desca, af, iaf, jaf, descaf,
             equed, sr, sc, b, ib, jb, descb, x, ix, jx, descx, rcond, ferr, berr,
             work, lwork, iwork, liwork, info)

call pzposvx(fact, uplo, n, nrhs, a, ia, ja, desca, af, iaf, jaf, descaf,
             equed, sr, sc, b, ib, jb, descb, x, ix, jx, descx, rcond, ferr, berr,
             work, lwork, iwork, liwork, info)
```

### 説明

このルーチンは、コレスキー因子分解  $A=U^T U$  または  $A=LL^T$  を使用して実 / 複素連立 1 次方程式の解を計算する。

$$A(ia:ia+n-1, ja:ja+n-1) * X = B(ib:ib+n-1, jb:jb+nrhs-1)$$

ここで、 $A(ia:ia+n-1, ja:ja+n-1)$  は  $n \times n$  の行列、 $X$  と  $B(ib:ib+n-1, jb:jb+nrhs-1)$  は  $n \times nrhs$  の行列である。

このルーチンは、解の誤差範囲と条件数の推定値も示す。

次の入力 / 出力パラメータの説明では、 $y$  は  $Y(iy:iy+m-1, jy:jy+k-1)$  が  $m \times k$  の行列 ( $y$  は  $a$ 、 $af$ 、 $b$  および  $x$ ) であることを示す。

ルーチン p?posvx は、以下の手順を実行する。

1.  $fact = 'E'$  の場合、連立方程式を平衡化するための実スケール係数  $s$  を計算する。

$$\text{diag}(sr) * A * \text{diag}(sc) * \text{inv}(\text{diag}(sc)) * X = \text{diag}(sr) * B$$

連立方程式が平衡化されるかどうかは、行列  $A$  のスケーリングによって決まる。平衡化が行われる場合、 $A$  は  $\text{diag}(sr) * A * \text{diag}(sc)$  によって上書きされ、 $B$  は  $\text{diag}(sr) * B$  によって上書きされる。

2.  $fact = 'N'$  または  $'E'$  の場合、コレスキー分解を使用して、( $fact = 'E'$  の場合は平衡化の後に) 行列  $A$  を次のように因子分解する。

$$A = U^T U \text{ (uplo = 'U' の場合)}$$

$$A = L L^T \text{ (uplo = 'L' の場合)}$$

ここで、 $U$  は上三角行列、 $L$  は下三角行列である。

3.  $A$  の因子分解された形式を使用して、行列  $A$  の条件数を推定する。条件数の逆数がマシンの精度より小さい場合、ステップ 4 から 6 はスキップされる。

4.  $A$  の因子分解された形式を使用して、連立方程式を  $X$  について解く。

5. 計算された解の行列の精度を繰り返し改善し、解の誤差範囲と後退誤差の推定値を計算する。

6. 平衡化が行われている場合、平衡化が行われる前の元の連立方程式を解くために、 $\text{diag}(sr)$  によって行列  $X$  を事前乗算する。

### 入力パラメータ

**fact** (グローバル) CHARACTER。  
'F'、'N'、または 'E' でなければならない。

呼び出し時に、行列  $A$  の因子分解された形式が与えられるかどうか、与えられない場合は、行列  $A$  を因子分解する前に平衡化するかどうかを指定する。

$fact = 'F'$  の場合、呼び出し時に、 $af$  は  $A$  の因子分解された形式を含む。 $equed = 'Y'$  の場合、行列  $A$  は、 $s$  で指定されたスケール係数によって平衡化されている。  
 $a$  と  $af$  は変更されない。

$fact = 'N'$  の場合、行列  $A$  を  $af$  にコピーし、因子分解を実行する。  
 $fact = 'E'$  の場合、必要に応じて行列  $A$  を平衡化してから、 $af$  にコピーし、因子分解を実行する。

**uplo** (グローバル) CHARACTER。  
'U' または 'L' でなければならない。

<i>n</i>	<p><i>A</i> の上三角部分と下三角部分のどちらが格納されるかを指定する。 (グローバル) INTEGER。 分散部分行列 <i>sub(A)</i> の次数 (<math>n \geq 0</math>)。</p>
<i>nrhs</i>	<p>(グローバル) INTEGER。 右辺の数。分散部分行列 <i>B</i> と <i>X</i> の列の数 (<math>nrhs \geq 0</math>)。</p>
<i>a</i>	<p>(ローカル) REAL (psposvx の場合) DOUBLE PRECISION (pdposvx の場合) COMPLEX (pcposvx の場合) DOUBLE COMPLEX (pzposvx の場合)。 ローカルメモリにある、ローカル次元 (<i>lld_a</i>, <i>LOCc(ja+n-1)</i>) の配列へのポインタ。呼び出し時は、対称 / エルミート行列 <i>A</i>。 ただし、<i>fact</i> = 'F' で <i>equed</i> = 'Y' の場合、<i>A</i> は平衡化された行列 <i>diag(sr)*A*diag(sc)</i> を含む。<i>uplo</i> = 'U' の場合、<i>A</i> の先頭の <math>n \times n</math> 上三角部分に行列 <i>A</i> の上三角部分を含む。<i>A</i> の厳密な下三角部分は参照されない。<i>uplo</i> = 'L' の場合、<i>A</i> の先頭の <math>n \times n</math> 下三角部分は行列 <i>A</i> の下三角部分を含む。<i>A</i> の厳密な上三角部分は参照されない。 <i>A</i> は、<i>fact</i> = 'F' または 'N' の場合、または <i>fact</i> = 'E' で <i>equed</i> = 'N' の場合、終了時に変更されない。</p>
<i>ia,ja</i>	<p>(グローバル) INTEGER。 それぞれ、部分行列 <i>A</i> の最初の行と最初の列を示す、グローバル配列 <i>a</i> の行インデックスと列インデックス。</p>
<i>desca</i>	<p>(グローバルおよびローカル) INTEGER。配列、次元は (<i>dlen_</i>)。 分散行列 <i>A</i> の配列ディスクリプタ。</p>
<i>af</i>	<p>(ローカル) REAL (psposvx の場合) DOUBLE PRECISION (pdposvx の場合) COMPLEX (pcposvx の場合) DOUBLE COMPLEX (pzposvx の場合)。 ローカルメモリにある、ローカル次元 (<i>lld_af</i>, <i>LOCc(ja+n-1)</i>) の配列へのポインタ。 <i>fact</i> = 'F' の場合、<i>af</i> は入力引数になる。呼び出し時に、この配列は、コレスキー因子分解 <math>A = U^T * U</math> または <math>A = L * L^T</math> で得られた三角係数 <i>U</i> あるいは <i>L</i> を、<i>A</i> と同じ格納形式で含む。<i>equed</i> が 'N' でない場合、<i>af</i> は、平衡化された行列 <i>diag(sr)*A*diag(sc)</i> の因子分解された形式になる。</p>

<i>iaf, jaf</i>	(グローバル) INTEGER。 それぞれ、部分行列 <i>sub(AF)</i> の最初の行と最初の列を示す、グローバル配列 <i>af</i> の行インデックスと列インデックス。
<i>descaf</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。 分散行列 <i>AF</i> の配列ディスクリプタ。
<i>equed</i>	(グローバル) CHARACTER。'N' または 'Y' でなければならない。 <i>equed</i> は、 <i>fact</i> = 'F' の場合は入力引数になる。この引数は、実行された平衡化の形式を指定する。 <i>equed</i> = 'N' の場合、平衡化は行われていない ( <i>fact</i> = 'N' の場合は常に真)。 <i>equed</i> = 'Y' の場合、平衡化が行われ、 <i>A</i> は <i>diag(sr)*A*diag(sc)</i> で置き換えられた。
<i>sr</i>	(ローカル) REAL ( <i>psposvx</i> の場合) DOUBLE PRECISION ( <i>pdposvx</i> の場合) COMPLEX ( <i>pcposvx</i> の場合) DOUBLE COMPLEX ( <i>pzposvx</i> の場合)。 配列、次元は ( <i>lld_a</i> )。 配列 <i>s</i> には、 <i>A</i> のスケール係数が格納される。この配列は、 <i>fact</i> = 'F' の場合にのみ入力引数になり、それ以外の場合は出力引数になる。 <i>equed</i> = 'N' の場合、 <i>s</i> は使用されない。 <i>fact</i> = 'F' かつ <i>equed</i> = 'Y' の場合、 <i>s</i> の各成分は正の値でなければならない。
<i>b</i>	(ローカル) REAL ( <i>psposvx</i> の場合) DOUBLE PRECISION ( <i>pdposvx</i> の場合) COMPLEX ( <i>pcposvx</i> の場合) DOUBLE COMPLEX ( <i>pzposvx</i> の場合)。 ローカルメモリにある、ローカル次元 ( <i>lld_b</i> , <i>LOCc(jb+nrhs-1)</i> ) の配列へのポインタ。呼び出し時は、 <i>n</i> × <i>nrhs</i> の行列 <i>B</i> の右辺。
<i>ib, jb</i>	(グローバル) INTEGER。 それぞれ、部分行列 <i>B</i> の最初の行と最初の列を示す、グローバル配列 <i>b</i> の行インデックスと列インデックス。
<i>descb</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。 分散行列 <i>B</i> の配列ディスクリプタ。

<i>x</i>	<p>(ローカル)</p> <p>REAL (psposvx の場合)</p> <p>DOUBLE PRECISION (pdposvx の場合)</p> <p>COMPLEX (pcposvx の場合)</p> <p>DOUBLE COMPLEX (pzposvx の場合)。</p> <p>ローカルメモリにある、ローカル次元 (<i>lld_x</i>, <i>LOCc(jx+nrhs-1)</i>) の配列へのポインタ。</p>
<i>ix, jx</i>	<p>(グローバル) INTEGER。</p> <p>それぞれ、部分行列 <i>X</i> の最初の行と最初の列を示す、グローバル配列 <i>x</i> の行インデックスと列インデックス。</p>
<i>descx</i>	<p>(グローバルおよびローカル) INTEGER。配列、次元は (<i>dlen_</i>)。</p> <p>分散行列 <i>X</i> の配列ディスクリプタ。</p>
<i>work</i>	<p>(ローカル)</p> <p>REAL (psposvx の場合)</p> <p>DOUBLE PRECISION (pdposvx の場合)</p> <p>COMPLEX (pcposvx の場合)</p> <p>DOUBLE COMPLEX (pzposvx の場合)。</p> <p>ワークスペース配列、次元は (<i>lwork</i>)。</p>
<i>lwork</i>	<p>(ローカルまたはグローバル)</p> <p>INTEGER。配列 <i>work</i> の次元。 <i>lwork</i> はローカル入力で、  <i>lwork</i> = max( <i>p?pocon</i>( <i>lwork</i> ), <i>p?porfs</i>( <i>lwork</i> ) ) + <i>LOCr</i>( <i>n_a</i> ) 以上でなければならない。  <i>lwork</i> = 3*<i>desca</i>( <i>lld_</i> )</p> <p><i>lwork</i> = -1 の場合、 <i>lwork</i> はグローバル入力であり、ワークスペースのクエリとみなされ、ルーチンはすべての <i>work</i> 配列の最小かつ最適なサイズだけを計算する。各値は該当する <i>work</i> 配列の最初のエントリとして返され、<a href="#">pxerbla</a> はエラー・メッセージを生成しない。</p>
<i>liwork</i>	<p>(ローカルまたはグローバル)</p> <p>INTEGER。</p> <p>配列 <i>iwork</i> の次元。 <i>liwork</i> はローカル入力で、  <i>liwork</i> = <i>desca</i>( <i>lld_</i> ) <i>liwork</i> = <i>LOCr</i>( <i>n_a</i> ) 以上でなければならない。  <i>liwork</i> = -1 の場合、 <i>liwork</i> はグローバル入力であり、ワークスペースのクエリとみなされ、ルーチンはすべての配列の最小かつ最適なサイズだけを計算する。各値は該当する <i>work</i> 配列の最初のエントリとして返され、<a href="#">pxerbla</a> はエラー・メッセージを生成しない。</p>

## 出力パラメータ

<i>a</i>	終了時に、 <i>fact</i> = 'E' で <i>equed</i> = 'Y' の場合、 <i>a</i> は $\text{diag}(sr) * a * \text{diag}(sc)$ によって上書きされる。
<i>af</i>	<i>fact</i> = 'N' の場合、 <i>af</i> は出力引数になる。この引数は、元の行列 <i>A</i> のコレスキー因子分解 $A = U^T * U$ または $A = L * L^T$ で得られた三角係数 <i>U</i> あるいは <i>L</i> を返す。 <i>fact</i> = 'E' の場合、 <i>af</i> は出力引数になる。この引数は、平衡化された行列 <i>A</i> のコレスキー因子分解 $A = U^T * U$ または $A = L * L^T$ で得られた三角係数 <i>U</i> あるいは <i>L</i> を返す (平衡化された行列の形式は、 <i>A</i> の説明を参照)。
<i>equed</i>	<i>fact</i> ≠ 'F' の場合、 <i>equed</i> は出力引数になる。この引数は、実行された平衡化の形式を指定する (「入力引数」セクションの <i>equed</i> の説明を参照)。
<i>sr</i>	この配列は、 <i>fact</i> ≠ 'F' の場合は出力引数になる。 「入力引数」セクションの <i>sr</i> の説明を参照。
<i>sc</i>	この配列は、 <i>fact</i> ≠ 'F' の場合は出力引数になる。 「入力引数」セクションの <i>sc</i> の説明を参照。
<i>b</i>	終了時に、 <i>equed</i> = 'N' の場合、 <i>b</i> は変更されない。 <i>trans</i> = 'N' で <i>equed</i> = 'R' または 'B' の場合、 <i>b</i> は $\text{diag}(r) * b$ によって上書きされる。 <i>trans</i> = 'T' または 'C' で <i>equed</i> = 'C' または 'B' の場合、 <i>b</i> は $\text{diag}(c) * b$ によって上書きされる。
<i>x</i>	(ローカル) REAL (psposvx の場合) DOUBLE PRECISION (pdposvx の場合) COMPLEX (pcposvx の場合) DOUBLE COMPLEX (pzposvx の場合)。  <i>info</i> = 0 の場合、元の連立方程式の $n \times nrhs$ の解の行列 <i>X</i> が格納される。ただし、 <i>equed</i> .ne. 'N' の場合、 <i>A</i> と <i>B</i> は終了時に修正され、平衡化された連立方程式の解は次のようになる。 $\text{inv}(\text{diag}(sc)) * X$ ( <i>trans</i> = 'N' かつ <i>equed</i> = 'C' または 'B' の場合) $\text{inv}(\text{diag}(sr)) * X$ ( <i>trans</i> = 'T' または 'C' かつ <i>equed</i> = 'R' または 'B' の場合)。
<i>rcond</i>	(グローバル) REAL (単精度の場合)。 DOUBLE PRECISION (倍精度の場合)。 平衡化後の行列 <i>A</i> の条件数の逆数の推定値 (平衡化が行われる場合)。



	<p><math>rcond</math> がマシンの精度より小さい場合 (特に、<math>rcond = 0</math> の場合)、行列は有効な精度で特異になる。この条件は、<math>info &gt; 0</math> のリターンコードで示される。</p>
<i>ferr</i>	<p>REAL (単精度の場合)。 DOUBLE PRECISION (倍精度の場合)。</p> <p>配列、次元は <math>\max(LOC, n\_b)</math> 以上。各解ベクトル用に推定された前進誤差範囲 <math>X(j)</math> (すなわち解の行列 <math>X</math> の <math>j</math> 番目の列)。<math>xtrue</math> が真の解である場合、<math>ferr(j)</math> は、<math>(X(j) - xtrue)</math> の最大エントリの大きさを <math>X(j)</math> の最大エントリの大きさを割って範囲を決める。誤差範囲の質は、コードで計算された <math>\text{norm}(\text{inv}(A))</math> の推定の質に依存する。<math>\text{norm}(\text{inv}(A))</math> の推定が正確である場合、誤差範囲は保証される。</p>
<i>berr</i>	<p>(ローカル)</p> <p>REAL (単精度の場合)。 DOUBLE PRECISION (倍精度の場合)。</p> <p>配列、次元は <math>\max(LOC, n\_b)</math> 以上。</p> <p>各解ベクトルの成分ごとの相対後退誤差 <math>X(j)</math> (すなわち <math>X(j)</math> が正確な解となる <math>A</math> または <math>B</math> の任意のエントリにおける最小相対変化)。</p>
<i>info</i>	<p>(グローバル) INTEGER。</p> <p><math>info = 0</math> の場合、実行は正常に終了したことを示す。</p> <p><math>info &lt; 0</math> の場合：</p> <p><math>info = -i</math> の場合、<math>i</math> 番目の引数の値が不正であることを示す。</p> <p><math>info &gt; 0</math> の場合：</p> <p><math>info = i</math> で、<math>i \leq n</math> の場合：<math>info = i</math> の場合、<math>a</math> の次数 <math>i</math> の先頭の小行列式が正定値ではないため因子分解を完了できず、解と誤差範囲を計算できなかったことを示す。</p> <p><math>i = n+1</math> の場合：<math>rcond</math> はマシンの精度より小さいことを示す。因子分解は完了したが、行列は有効な精度で完全に特異であるため、解と誤差範囲を計算できなかった。</p>

### p?pbsv

対称/エルミート正定値帯形式の連立1次方程式を解く。

---

#### 構文

```
call pspbsv(uplo, n, bw, nrhs, a, ja, desca, b, ib, descb, work, lwork, info)
call pdpbsv(uplo, n, bw, nrhs, a, ja, desca, b, ib, descb, work, lwork, info)
call pcpbsv(uplo, n, bw, nrhs, a, ja, desca, b, ib, descb, work, lwork, info)
call pzpbsv(uplo, n, bw, nrhs, a, ja, desca, b, ib, descb, work, lwork, info)
```

#### 説明

このルーチンは、次の連立1次方程式を解く。

$$A(1:n, ja:ja+n-1) * X = B(ib:ib+n-1, 1:nrhs)$$

ここで、 $A(1:n, ja:ja+n-1)$  は帯域が  $bw$  の  $n \times n$  の実/複素対称正定値帯分散行列である。

行列の順序を変更して  $LL'$  に因子分解するために、コレスキー分解が使用される。

#### 入力パラメータ

<code>uplo</code>	(グローバル) CHARACTER。 'U' または 'L' でなければならない。 $A$ の上三角部分と下三角部分のどちらが格納されるかを指定する。 <code>uplo = 'U'</code> の場合、 $A$ の上三角部分が格納される。 <code>uplo = 'L'</code> の場合、 $A$ の下三角部分が格納される。
<code>n</code>	(グローバル) INTEGER。 分散行列 $A$ の次数 ( $n \geq 0$ )。
<code>bw</code>	(グローバル) INTEGER。 $L$ または $U$ の劣対角の数。 $0 \leq bw \leq n-1$ 。
<code>nrhs</code>	(グローバル) INTEGER。 右辺の数。 $B$ の列の数 ( $nrhs \geq 0$ )。

<i>a</i>	<p>(ローカル)。  REAL (pspbsv の場合)  DOUBLE PRECISION (pdpbsv の場合)  COMPLEX (pcpbsv の場合)  DOUBLE COMPLEX (pzpbsv の場合)。  ローカルメモリにある、第 1 次元が <math>lld\_a \geq (bw+1)</math> の配列へのポインタ (<i>desca</i> に格納される)。  呼び出し時に、この配列は、因子分解する対称分散行列 <math>\text{sub}(A)</math> のローカル部分を含む。</p>
<i>ja</i>	<p>(グローバル) INTEGER。  (<i>A</i> のすべて、または <i>A</i> の部分行列で) 処理される行列の先頭を指すグローバル配列 <i>a</i> のインデックス。</p>
<i>desca</i>	<p>(グローバルおよびローカル) INTEGER。配列、次元は (<i>dlen</i>)。  分散行列 <i>A</i> の配列ディスクリプタ。</p>
<i>b</i>	<p>(ローカル)  REAL (pspbsv の場合)  DOUBLE PRECISION (pdpbsv の場合)  COMPLEX (pcpbsv の場合)  DOUBLE COMPLEX (pzpbsv の場合)。  ローカルメモリにある、ローカル・リーディング・ディメンジョン <math>lld\_b \geq NB</math> の配列へのポインタ。呼び出し時に、この配列は、右辺 <math>B(ib:ib+n-1, 1:nrhs)</math> のローカル部分を含む。</p>
<i>ib</i>	<p>(グローバル) INTEGER。  (<i>b</i> のすべて、または <i>B</i> の部分行列で) 処理される行列の最初の行を指すグローバル配列 <i>b</i> の行インデックス。</p>
<i>desb</i>	<p>(グローバルおよびローカル) INTEGER。配列、次元は (<i>dlen</i>)。  1D type (<i>dtype_b</i> = 502) の場合、<math>dlen \geq 7</math>。  2D type (<i>dtype_b</i> = 1) の場合、<math>dlen \geq 9</math>。  分散行列 <i>B</i> の配列ディスクリプタ。<i>B</i> のマッピング情報をメモリに格納する。</p>
<i>work</i>	<p>(ローカル)。  REAL (pspbsv の場合)  DOUBLE PRECISION (pdpbsv の場合)  COMPLEX (pcpbsv の場合)  DOUBLE COMPLEX (pzpbsv の場合)。  テンポラリ・ワークスペース。このスペースは、ルーチンの呼び出しの間に上書きされる。<i>work</i> のサイズは <i>lwork</i> で与えられる。</p>

*lwork* (ローカルまたはグローバル) INTEGER。  
ユーザ入力ワークスペース *work* のサイズ。 *lwork* が小さすぎる場合、エラーコードが返され、エラーにならない最小のサイズが *work(1)* に返される。  $lwork \geq (NB+2*bw)*bw + \max((bw*nrhs), bw*bw)$

### 出力パラメータ

*a* 終了時に、この配列には因子分解の詳細が格納される。行列で置換が行われるため、返される係数は LAPACK から返される係数とは異なる点に注意する。

*b* 終了時に、解の分散行列 *X* のローカル部分が格納される。

*work* 終了時に、*work(1)* には、*lwork* の最小値が格納される。

*info* (グローバル) INTEGER。 *info* = 0 の場合、実行は正常に終了したことを示す。  
*info* < 0 の場合：  
*i* 番目の引数が配列で、*j* 番目の値が不正だった場合、  
*info* = -(*i*\*100+*j*)。 *i* 番目の引数がスカラで値が不正だった場合、  
*info* = -*i*。  
*info* > 0 の場合：  
*info* = *k* ≤ NPROCS の場合、プロセッサ *info* に格納され、ローカルに因子分解された部分行列が正定値でないため、因子分解を完了できなかったことを示す。  
*info* = *k* > NPROCS の場合、他のプロセッサとの対話を表すプロセッサ *info*-NPROCS に格納された部分行列が正定値ではないため、因子分解を完了できなかったことを示す。

---

## p?ptsv

対称/エルミート正定値三重対角連立1次方程式を解く。

---

### 構文

```
call psptsv(n, nrhs, d, e, ja, desca, b, ib, descb, work, lwork, info)
call pdptsv(n, nrhs, d, e, ja, desca, b, ib, descb, work, lwork, info)
call pcptsv(n, nrhs, d, e, ja, desca, b, ib, descb, work, lwork, info)
call pzptsv(n, nrhs, d, e, ja, desca, b, ib, descb, work, lwork, info)
```

## 説明

このルーチンは、次の連立 1 次方程式を解く。

$$A(1:n, ja:ja+n-1) * X = B(ib:ib+n-1, 1:nrhs)$$

ここで、 $A(1:n, ja:ja+n-1)$  は  $n \times n$  の実対称正定値三重対角分散行列である。

行列の順序を変更して  $LL'$  に因子分解するために、コレスキー分解が使用される。

## 入力パラメータ

$n$	(グローバル) INTEGER。 行列 $A$ の次数 ( $n \geq 0$ )。
$nrhs$	(グローバル) INTEGER。 右辺の数。分散部分行列 $B$ の列数 ( $nrhs \geq 0$ )。
$d$	(ローカル) REAL (psptsv の場合) DOUBLE PRECISION (pdptsv の場合) COMPLEX (pcptsv の場合) DOUBLE COMPLEX (pzptsv の場合)。 行列の主対角を格納するグローバル・ベクトルのローカル部分へのポインタ。
$e$	(ローカル) REAL (psptsv の場合) DOUBLE PRECISION (pdptsv の場合) COMPLEX (pcptsv の場合) DOUBLE COMPLEX (pzptsv の場合)。 行列の上対角を格納するグローバル・ベクトルのローカル部分へのポインタ。全体的に、 $du(n)$ は参照されず、 $du$ は $d$ とアライメントされなければならない。
$ja$	(グローバル) INTEGER。 ( $A$ のすべて、または $A$ の部分行列で) 処理される行列の先頭を指すグローバル配列 $A$ のインデックス。
$desca$	(グローバルおよびローカル) INTEGER。配列、次元は ( $dlen$ )。 1d type ( $dtype\_a = 501$ または $502$ ) の場合、 $dlen \geq 7$ 。 2d type ( $dtype\_a = 1$ ) の場合、 $dlen \geq 9$ 。 分散行列 $A$ の配列ディスクリプタ。 $A$ のマッピング情報をメモリに格納する。
$b$	(ローカル) REAL (psptsv の場合) DOUBLE PRECISION (pdptsv の場合)

	COMPLEX (pcptsv の場合 ) DOUBLE COMPLEX (pzptsv の場合 )。 ローカルメモリにある、ローカル・リーディング・ディメンジョン $lld\_b \geq NB$ の配列へのポインタ。呼び出し時に、この配列は、右辺 $B(ib:ib+n-1, 1:nrhs)$ のローカル部分を含む。
<i>ib</i>	( グローバル ) INTEGER。 ( <i>b</i> のすべて、または $B$ の部分行列で ) 処理される行列の最初の行を指すグローバル配列 <i>b</i> の行インデックス。
<i>desb</i>	( グローバルおよびローカル ) INTEGER。配列、次元は ( <i>dlen</i> )。 1d type ( <i>dtype_b</i> = 502) の場合、 $dlen \geq 7$ 。 2d type ( <i>dtype_b</i> = 1) の場合、 $dlen \geq 9$ 。 分散行列 $B$ の配列ディスクリプタ。 $B$ のマッピング情報をメモリに格納する。
<i>work</i>	( ローカル )。 REAL (psptsv の場合 ) DOUBLE PRECISION (pdptsv の場合 ) COMPLEX (pcptsv の場合 ) DOUBLE COMPLEX (pzptsv の場合 )。 テンポラリ・ワークスペース。このスペースは、ルーチンの呼び出しの間に上書きされる。 <i>work</i> のサイズは <i>lwork</i> で与えられる。
<i>lwork</i>	( ローカルまたはグローバル ) INTEGER。 ユーザ入力ワークスペース <i>work</i> のサイズ。 <i>lwork</i> が小さすぎる場合、エラーコードが返され、エラーにならない最小のサイズが <i>work</i> (1) に返される。 $lwork \geq (12 * NPCOL + 3 * NB) + \max((10 + 2 * \min(100, nrhs)) * NPCOL + 4 * nrhs, 8 * NPCOL)$

## 出力パラメータ

<i>d</i>	終了時に、行列の係数を含む情報が格納される。サイズは <i>desca</i> ( <i>nb_</i> ) 以上でなければならない。
<i>e</i>	行列の係数を含む情報が格納される。サイズは <i>desca</i> ( <i>nb_</i> ) 以上でなければならない。
<i>b</i>	終了時に、解の分散行列 $X$ のローカル部分が格納される。
<i>work</i>	終了時に、 <i>work</i> (1) には、 <i>lwork</i> の最小値が格納される。

*info* (ローカル) INTEGER。 *info* = 0 の場合、実行は正常に終了したことを示す。  
*info* < 0 の場合：  
*i* 番目の引数が配列で、*j* 番目の値が不正だった場合、  
*info* = -(*i*\*100+*j*)。 *i* 番目の引数がスカラで値が不正だった場合、  
*info* = -*i*。  
*info* > 0 の場合：  
*info* = *k* ≤ NPROCS の場合、プロセッサ *info* に格納され、ローカルに因子分解された部分行列が正定値でないため、因子分解を完了できなかったことを示す。  
*info* = *k* > NPROCS の場合、他のプロセッサとの対話を表すプロセッサ *info*-NPROCS に格納された部分行列が正定値ではないため、因子分解を完了できなかったことを示す。

## p?gels

最大階数の行列で表される、優決定または劣決定の線形問題を解く。

### 構文

```
call psgels( trans, m, n, nrhs, a, ia, ja, desca, b, ib, jb, descb,
             work, lwork, info )
call pdgels( trans, m, n, nrhs, a, ia, ja, desca, b, ib, jb, descb,
             work, lwork, info )
call pcgels( trans, m, n, nrhs, a, ia, ja, desca, b, ib, jb, descb,
             work, lwork, info )
call pzgels( trans, m, n, nrhs, a, ia, ja, desca, b, ib, jb, descb,
             work, lwork, info )
```

### 説明

このルーチンは、 $\text{sub}(A)$  の  $QR$  または  $LQ$  因子分解を使用して、 $m \times n$  の行列  $\text{sub}(A) = A(\text{ia}:\text{ia}+m-1, \text{ja}:\text{ja}+n-1)$  (またはその転置 / 共役転置行列) で表される、実 / 複素の優決定 / 劣決定の線形問題を解く。ただし、 $\text{sub}(A)$  は最大階数の行列とする。

次のオプションが提供されている。

1.  $trans = 'N'$  かつ  $m \geq n$  の場合: 優決定の最小二乗解を求める。すなわち、次の最小二乗問題を解く。

$$\text{minimize } \| \text{sub}(B) - \text{sub}(A) X \|$$

2.  $trans = 'N'$  かつ  $m < n$  の場合: 劣決定の問題  $\text{sub}(A) X = \text{sub}(B)$  の最小ノルム解を求める。

3.  $trans = 'T'$  かつ  $m \geq n$  の場合: 劣決定の問題  $\text{sub}(A)^T X = \text{sub}(B)$  の最小ノルム解を求める。

4.  $trans = 'T'$  かつ  $m < n$  の場合: 優決定の最小二乗解を求める。すなわち、次の最小二乗問題を解く。

$$\text{minimize } \| \text{sub}(B) - \text{sub}(A)^T X \|$$

ここで、 $\text{sub}(B)$  は  $B(ib:ib+m-1, jb:jb+nrhs-1)$  ( $trans = 'N'$  の場合)、 $B(ib:ib+n-1, jb:jb+nrhs-1)$  (それ以外の場合) である。複数の右辺のベクトル  $b$  と解ベクトル  $x$  を、一度の呼び出しで処理できる。

$trans = 'N'$  の場合、解ベクトルは  $n \times nrhs$  の右辺の行列  $\text{sub}(B)$  に格納され、それ以外の場合、 $m \times nrhs$  の右辺の行列  $\text{sub}(B)$  に格納される。

## 入力パラメータ

<i>trans</i>	(グローバル) CHARACTER。'N' または 'T' でなければならない。 <i>trans</i> = 'N' の場合、行列 $\text{sub}(A)$ で表される線形問題。 <i>trans</i> = 'T' の場合、転置行列 $A^T$ で表される線形問題 (実数型の場合のみ)。
<i>m</i>	(グローバル) INTEGER。 分散部分行列 $\text{sub}(A)$ の行数 ( $m \geq 0$ )。
<i>n</i>	(グローバル) INTEGER。 分散部分行列 $\text{sub}(A)$ の列数 ( $n \geq 0$ )。
<i>nrhs</i>	(グローバル) INTEGER。 右辺の数。分散部分行列 $\text{sub}(B)$ と $X$ の列数 ( $nrhs \geq 0$ )。
<i>a</i>	(ローカル) REAL (psgels の場合) DOUBLE PRECISION (pdgels の場合) COMPLEX (pcgels の場合) DOUBLE 1COMPLEX (pzgels の場合)。 ローカルメモリにある、次元 ( $lld\_a, LOCc(ja+n-1)$ ) の配列へのポインタ。呼び出し時に、 $m \times n$ の行列 $A$ を含む。



<i>ia, ja</i>	(グローバル) INTEGER。 それぞれ、部分行列 <i>A</i> の最初の行と最初の列を示す、グローバル配列 <i>a</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。 分散行列 <i>A</i> の配列ディスクリプタ。
<i>b</i>	(ローカル) REAL ( <i>psgels</i> の場合) DOUBLE PRECISION ( <i>pdgels</i> の場合) COMPLEX ( <i>pcgels</i> の場合) DOUBLE COMPLEX ( <i>pzgel</i> s の場合)。 ローカルメモリにある、ローカル次元 ( <i>lld_b</i> , <i>LOCc(jb+nrhs-1)</i> ) の配列へのポインタ。呼び出し時に、この配列は、右辺の分散行列 <i>B</i> のローカル部分を列方向に含む。 <i>sub(B)</i> は $m \times nrhs$ ( <i>trans</i> = 'N' の場合) または $n \times nrhs$ (それ以外の場合)。
<i>ib, jb</i>	(グローバル) INTEGER。 それぞれ、部分行列 <i>B</i> の最初の行と最初の列を示す、グローバル配列 <i>b</i> の行インデックスと列インデックス。
<i>descb</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。 分散行列 <i>B</i> の配列ディスクリプタ。
<i>work</i>	(ローカル) REAL ( <i>psgels</i> の場合) DOUBLE PRECISION ( <i>pdgels</i> の場合) COMPLEX ( <i>pcgels</i> の場合) DOUBLE COMPLEX ( <i>pzgel</i> s の場合)。 次元 <i>lwork</i> のワークスペース配列。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。配列 <i>work</i> の次元。 <i>lwork</i> はローカル入力で、 $lwork \geq ltau + \max(lwf, lws)$ でなければならない。 ここで、 $m \geq n$ の場合、 $ltau = \text{numroc}(ja + \min(m, n) - 1, nb\_a, MYCOL, csrc\_a, NPCOL)$ 、 $lwf = nb\_a * (mpa0 + nqa0 + nb\_a)$ $lws = \max((nb\_a * (nb\_a - 1)) / 2, (nrhsqb0 + mpb0) * nb\_a) + nb\_a * nb\_a$ そうでない場合、 $ltau = \text{numroc}(ia + \min(m, n) - 1, mb\_a, MYROW, rsrc\_a, NPROW)$ 、 $lwf = mb\_a * (mpa0 + nqa0 + mb\_a)$

```
lws = max((mb_a*(mb_a-1))/2, (npb0 + max(nqa0 +
numroc(numroc(n+irofffb, mb_a, 0, 0, NPROW), mb_a, 0, 0, lcmp),
nrhsqb0))*mb_a) + mb_a * mb_a
```

である。

```
ここで、lcmp = lcm / NPROW で lcm = ilcm(NPROW, NPCOL、
irofffa = mod(ia-1, mb_a)、
icofffa = mod(ja-1, nb_a)、
iarow = indxg2p(ia, mb_a, MYROW, rsrc_a, NPROW)、
iacol = indxg2p(ja, nb_a, MYROW, rsrc_a, NPROW)、
mpa0 = numroc(m+irofffa, mb_a, MYROW, iarow, NPROW)、
nqa0 = numroc(n+icofffa, nb_a, MYCOL, iacol, NPCOL)、
irofffb = mod(ib-1, mb_b)、
icofffb = mod(jb-1, nb_b)、
ibrow = indxg2p(ib, mb_b, MYROW, rsrc_b, NPROW)、
ibcol = indxg2p(jb, nb_b, MYCOL, csrc_b, NPCOL)、
mpb0 = numroc(m+irofffb, mb_b, MYROW, icrow, NPROW)、
nqb0 = numroc(n+icofffb, nb_b, MYCOL, ibcol, NPCOL)
```

ilcm、indxg2p および numroc は、ScaLAPACK ツール関数である。  
MYROW、MYCOL、NPROW、および NPCOL は、サブルーチン  
blacs\_gridinfo を呼び出して決定できる。

lwork = -1 の場合、lwork はグローバル入力であり、ワークスペース  
のクエリとみなされ、ルーチンはすべての work 配列の最小かつ最適  
なサイズだけを計算する。各値は該当する work 配列の最初のエン  
トリとして返され、[pxerbla](#) はエラー・メッセージを生成しない。

### 出力パラメータ

- a 終了時に、 $m \geq n$  の場合、 $\text{sub}(A)$  は  $QR$  因子分解の詳細 ([p?qgeqrf](#) から返される値) によって上書きされる。 $m < n$  の場合、 $\text{sub}(A)$  は  $LQ$  因子分解の詳細 ([p?qelqf](#) から返される値) によって上書きされる。
- b 終了時に、 $\text{sub}(B)$  は解ベクトルによって上書きされ、列方向に格納される。 $\text{trans} = 'N'$  かつ  $m \geq n$  の場合、 $\text{sub}(B)$  の 1 から  $n$  行に、最小二乗解のベクトルが格納される。各列の解の二乗和の誤差は、それぞれの列の  $n+1$  から  $m$  の成分の二乗和で与えられる。 $\text{trans} = 'N'$  かつ  $m < n$  の場合、 $\text{sub}(B)$  の 1 から  $n$  行に、最小ノルム解のベクトルが格納される。 $\text{trans} = 'T'$  かつ  $m \geq n$  の場合、 $\text{sub}(B)$  の 1 から  $m$  行に、最小ノルム解のベクトルが格納される。

	$trans = 'T'$ かつ $m < n$ の場合、 $sub(B)$ の 1 から $m$ 行に、最小ノルム解のベクトルが格納される。各列の解の二乗和の誤差は、それぞれの列の $m+1$ から $n$ の成分の二乗和で与えられる。
<code>work(1)</code>	終了時に、 <code>work(1)</code> には、最適なパフォーマンスを得るために必要な <code>lwork</code> の最小値が格納される。
<code>info</code>	(グローバル) INTEGER。 <code>info = 0</code> の場合、実行は正常に終了したことを示す。 <code>info &lt; 0</code> の場合： $i$ 番目の引数が配列で、 $j$ 番目の値が不正だった場合、 <code>info = -(i*100+j)</code> 。 $i$ 番目の引数がスカラーで値が不正だった場合、 <code>info = -i</code> 。

## p?syev

対称対角行列の固有値と固有ベクトルを選択的に計算する。

### 構文

```
call pssyev( jobz, uplo, n, a, ia, ja, desca, w, z, iz, jz, descz, work,
             lwork, info )
call pdsyev( jobz, uplo, n, a, ia, ja, desca, w, z, iz, jz, descz, work,
             lwork, info )
```

### 説明

このルーチンは、ScaLAPACK ルーチンの推奨される組み合わせの呼び出しによって、実対称行列  $A$  のすべての固有値と (オプションで) 固有ベクトルを計算する。現在の形式では、ルーチンは、均一な方程式を仮定しており、異なるプロセスにおける固有値や固有ベクトルの整合性をチェックしない。このため、均一でない方程式は、エラー・メッセージを出力せずに、正しくない値を返す可能性がある。

### 入力パラメータ

$np$  = 与えられたプロセスに対するローカルの行数。

$nq$  = 与えられたプロセスに対するローカルの列数。

<i>jobz</i>	(グローバル) CHARACTER。 'N' または 'V' でなければならない。 固有ベクトルを計算する場合に指定する。 <i>jobz</i> = 'N' の場合、固有値のみ計算する。 <i>jobz</i> = 'V' の場合、固有値と固有ベクトルを計算する。
<i>uplo</i>	(グローバル) CHARACTER。 'U' または 'L' でなければならない。 対称行列 <i>A</i> の上三角部分または下三角部分のどちらが格納されるかを指定する。 <i>uplo</i> = 'U' の場合、 <i>a</i> には <i>A</i> の上三角部分が格納される。 <i>uplo</i> = 'L' の場合、 <i>a</i> には <i>A</i> の下三角部分が格納される。
<i>n</i>	(グローバル) INTEGER。行列 <i>A</i> ( $n \geq 0$ ) の行数と列数。
<i>a</i>	(ローカル) REAL ( <i>pssyev</i> の場合) DOUBLE PRECISION ( <i>pdsyev</i> の場合)。 グローバル次元 ( <i>n,n</i> ) とローカル次元 ( <i>lld_a</i> , <i>LOCc(ja+n-1)</i> ) のブロック・サイクリック配列。呼び出し時は、対称行列 <i>A</i> 。 <i>uplo</i> = 'U' の場合、行列 <i>A</i> の上三角部分だけが対称行列の成分を定義するために使用される。 <i>uplo</i> = 'L' の場合、行列 <i>A</i> の下三角部分だけが対称行列の成分を定義するために使用される。
<i>ia,ja</i>	(グローバル) INTEGER。 それぞれ、部分行列 <i>A</i> の最初の行と最初の列を示す、グローバル配列 <i>a</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。 分散行列 <i>A</i> の配列ディスクリプタ。
<i>iz,jz</i>	(グローバル) INTEGER。 それぞれ、部分行列 <i>Z</i> の最初の行と最初の列を示す、グローバル配列 <i>z</i> の行インデックスと列インデックス。
<i>descz</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。 分散行列 <i>Z</i> の配列ディスクリプタ。
<i>work</i>	(ローカル) REAL ( <i>pssyev</i> の場合) DOUBLE PRECISION ( <i>pdsyev</i> の場合)。配列、次元は ( <i>lwork</i> )。
<i>lwork</i>	(ローカル) INTEGER。 <i>lwork</i> の定義に使用する変数の定義は以下を参照。 固有ベクトルが要求されなかった場合 ( <i>jobz</i> = 'N')、

$lwork \geq 5*n + sizesytrd + 1$

ここで、*sizesytrd* は、[p?sytrd](#) のワークスペース要件で

$\max(NB * (np + 1), 3 * NB)$ 。

固有ベクトルが要求された場合 (*jobz* = 'V')、すべての固有ベクトルが計算されることを保証するために必要なワークスペースの量は次のとおりである。

$qrmem = 2*n-2$

$lwmin = 5*n + n*ldc + \max(sizemqrleft, qrmem) + 1$

変数定義:

$NB = desca(mb\_)= desca(nb\_)= * descz(mb\_)= descz(nb\_)$

$nn = \max(n, NB, 2)$

$desca(rsrc\_)= desca(rsrc\_)= descz(rsrc\_)= * descz(csrc\_)= 0$

$np = \text{numroc}(nn, NB, 0, 0, NPROW)$

$nq = \text{numroc}(\max(n, NB, 2), NB, 0, 0, NPCOL)$

$nrc = \text{numroc}(n, NB, myprowc, 0, NPROCS)$

$ldc = \max(1, nrc)$

*sizemqrleft* は、その *side* 引数が 'L' のときの *p?ormtr* のワークスペース要件。

定義済みの *myprowc* を使用して、新しいコンテキストは次のようにして作成される。

`call blacs_get(desca(ctxt_), 0, contextc) call`

`blacs_gridinit(contextc, 'R', NPROCS, 1) call`

`blacs_gridinfo(contextc, nprowc, npcolc, myprowc, mypcolc)`

*lwork* = -1 の場合、*lwork* はグローバル入力であり、ワークスペースのクエリとみなされ、ルーチンはすべての *work* 配列の最小かつ最適なサイズだけを計算する。各値は該当する *work* 配列の最初のエン트리として返され、[pxerbla](#) はエラー・メッセージを生成しない。

## 出力パラメータ

*a* 終了時に、行列 *A* の下三角部分 (*uplo* = 'L' の場合) または上三角部分 (*uplo* = 'U' の場合) が、対角成分を含めて削除される。

*w* (グローバル)。

REAL (*pssyev* の場合)

DOUBLE PRECISION (*pdsyev* の場合)

配列、次元は *n*。

正常終了時に、先頭の *m* 個の成分には、指定された固有値が昇順で格納される。

<i>z</i>	<p>(ローカル)。</p> <p>REAL (pssyev の場合)</p> <p>DOUBLE PRECISION (pdsyev の場合)</p> <p>配列、グローバル次元 (<math>n, n</math>)、ローカル次元 (<math>lld\_z, LOCc(jz+n-1)</math>)。</p> <p><i>jobz</i> = 'V' の場合、正常終了時に、<i>z</i> の先頭の <i>m</i> 列には、選択された固有値に対応する正規直交固有ベクトルが格納される。<i>jobz</i> = 'N' の場合、<i>z</i> は参照されない。</p>
<i>work(1)</i>	<p>終了時に、<i>work(1)</i> は計算の完了を保証するために必要なワークスペースを返す。入力パラメータが正しくない場合、<i>work(1)</i> も正しくない。</p> <p><i>jobz</i> = 'N' の場合、<i>work(1)</i> はワークスペースの最小の (最適な) 値。</p> <p><i>jobz</i> = 'V' の場合、<i>work(1)</i> はすべての固有ベクトルを生成するために必要な最小ワークスペース。</p>
<i>info</i>	<p>(グローバル)</p> <p>INTEGER。</p> <p><i>info</i> = 0 の場合、実行は正常に終了したことを示す。</p> <p><i>info</i> &lt; 0 の場合:</p> <p><i>i</i> 番目の引数が配列で、<i>j</i> 番目の値が不正だった場合、  <i>info</i> = -(<i>i</i>*100+<i>j</i>)。 <i>i</i> 番目の引数がスカラで値が不正だった場合、  <i>info</i> = -<i>i</i>。</p> <p><i>info</i> &gt; 0:</p> <p><i>info</i> = 1 ~ <i>n</i> の場合、<i>i</i> 番目の固有値は合計 30<i>n</i> 回の反復で ?steqr2 に収束しなかった。</p> <p><i>info</i> = <i>n</i>+1 の場合、p?syeve は固有値がプロセスグリッドで同一でなかったことから不均一性を検出した。この場合、p?syeve の結果は保証されない。</p>

---

### p?syeve

対称行列の固有値と (オプションで)  
固有ベクトルを選択的に計算する。

---

#### 構文

```
call pssyevx(jobz, range, uplo, n, a, ia, ja, desca, vl, vu, il, iu,
             abstol, m, nz, w, orfac, z, iz, jz, descz, work, lwork, iwork, liwork,
             ifail, iclustr, gap, info)
```

```
call pdsyevx(jobz, range, uplo, n, a, ia, ja, desca, vl, vu, il, iu,
             abstol, m, nz, w, orfac, z, iz, jz, descz, work, lwork, iwork, liwork,
             ifail, iclustr, gap, info)
```

## 説明

このルーチンは、ScaLAPACK ルーチンの推奨される組み合わせの呼び出しによって、実対称行列  $A$  の固有値と (オプションで) 固有ベクトルを選択的に計算する。固有値と固有ベクトルを選択するには、希望する固有値の値の範囲かインデックスの範囲を指定する。

## 入力パラメータ

$np$  = 与えられたプロセスに対するローカルの行数。

$nq$  = 与えられたプロセスに対するローカルの列数。

**jobz** (グローバル) CHARACTER\*1。  
 'N' または 'V' でなければならない。  
 固有ベクトルを計算する場合に指定する。  
 jobz='N' の場合、固有値のみ計算する。  
 jobz='V' の場合、固有値と固有ベクトルを計算する。

**range** (グローバル) CHARACTER\*1。  
 'A'、'V'、または 'I' でなければならない。  
 range='A' の場合、固有値をすべて計算する。  
 range='V' の場合、半開区間  $[vl, vu]$  の固有値をすべて計算する。  
 range='I' の場合、インデックスが  $il \sim iu$  の固有値を計算する。

**uplo** (グローバル) CHARACTER\*1。  
 'U' または 'L' でなければならない。  
 対称行列  $A$  の上三角部分または下三角部分のどちらが格納されるかを指定する。  
 uplo='U' の場合、 $a$  には  $A$  の上三角部分が格納される。  
 uplo='L' の場合、 $a$  には  $A$  の下三角部分が格納される。

**n** (グローバル) INTEGER。行列  $A$  ( $n \geq 0$ ) の行数と列数。

**a** (ローカル)。  
 REAL (pssyevx の場合)  
 DOUBLE PRECISION (pdsyevx の場合)。  
 グローバル次元 ( $n, n$ ) とローカル次元 ( $lld\_a, LOCC(ja+n-1)$ ) のブロック・サイクリック配列。呼び出し時は、対称行列  $A$ 。uplo='U' の場

合、行列  $A$  の上三角部分だけが対称行列の成分を定義するために使用される。 $uplo = 'L'$  の場合、行列  $A$  の下三角部分だけが対称行列の成分を定義するために使用される。

$ia, ja$	(グローバル) INTEGER。 それぞれ、部分行列 $A$ の最初の行と最初の列を示す、グローバル配列 $a$ の行インデックスと列インデックス。
$desca$	(グローバルおよびローカル) INTEGER。配列、次元は ( $dlen\_$ )。分散行列 $A$ の配列ディスクリプタ。
$vl, vu$	(グローバル) REAL (pssyevx の場合) DOUBLE PRECISION (pdsyevx の場合)。 $range = 'V'$ の場合、固有値を探す区間の下限値と上限値。 $vl \leq vu$ 。 $range = 'A'$ または $'I'$ の場合は参照されない。
$il, iu$	(グローバル) INTEGER。 $range = 'I'$ の場合、求める固有値の最小インデックスと最大インデックス。次の制約がある。 $il \geq 1$ $\min(il, n) \leq iu \leq n$ $range = 'A'$ または $'V'$ の場合は参照されない。
$abstol$	(グローバル)。 REAL (pssyevx の場合) DOUBLE PRECISION (pdsyevx の場合) $jobz = 'V'$ の場合、 $abstol$ を $p?lamch(context, 'U')$ に設定すると、ほとんどの直交固有ベクトルが生成される。

固有値に対する絶対エラー許容値。 $abstol + eps * \max(|a|, |b|)$  以下の幅の区間  $[a, b]$  内に存在していると判別された場合 ( $eps$  はマシン精度)、近似固有値は、収束したものとして受け入れられる。 $abstol$  がゼロまたは負の場合、代わりに  $eps * \text{norm}(T)$  を使用する。ここで、 $\text{norm}(T)$  は、 $A$  を三重対角形式に縮退して得られた三重対角行列の 1- ノルムである。

固有値が最も正確に計算されるのは、 $abstol$  がゼロでなく、アンダーフローのしきい値の 2 倍、 $2 * p?lamch('S')$  に設定されたときである。このルーチンで  $((\text{mod}(\text{info}, 2).ne.0).or. * (\text{mod}(\text{info}/8, 2).ne.0))$  が返された場合、固有値または固有ベクトルのいくつか収束に失敗したことを意味するので、 $abstol$  を  $2 * p?lamch('S')$  に設定して、やり直してみる。



<i>orfac</i>	<p>(グローバル)。  REAL (pssyevx の場合)  DOUBLE PRECISION (pdsyevx の場合)。  再直交化される固有ベクトルを指定する。互いの <math>tol=orfac*\text{norm}(A)</math> 内にある固有値に対応する固有ベクトルは再直交される。しかし、ワークスペースが十分でない場合 (<i>lwork</i> を参照)、<i>tol</i> は、すべての固有ベクトルが 1 つのプロセスで再直交化されるようになるまで下げられる。<i>orfac</i> がゼロの場合、再直交化は行われない。<i>orfac</i> が負の場合、デフォルト値の <math>10^3</math> が使用される。<i>orfac</i> はすべての処理で同一でなければならない。</p>
<i>iz, jz</i>	<p>(グローバル) INTEGER。  それぞれ、部分行列 <i>Z</i> の最初の行と最初の列を示す、グローバル配列 <i>z</i> の行インデックスと列インデックス。</p>
<i>descz</i>	<p>(グローバルおよびローカル) INTEGER。配列、次元は (<i>dlen_</i>)。  分散配列 <i>Z</i> のディスクリプタ。<i>descz</i>(<i>ctxt_</i>) は <i>desca</i>(<i>ctxt_</i>) と等しくなければならない。</p>
<i>work</i>	<p>(ローカル)  REAL (pssyevx の場合)  DOUBLE PRECISION (pdsyevx の場合)。配列、次元は (<i>lwork</i>)。</p>
<i>lwork</i>	<p>(ローカル) INTEGER。配列 <i>work</i> の次元。  <i>lwork</i> の定義に使用する変数の定義は以下を参照。  固有ベクトルが要求されなかった場合 (<i>jobz</i> = 'N')、  <math>lwork \geq 5 * n + \max(5 * nn, NB * (np0 + 1))</math>。  固有ベクトルが要求された場合 (<i>jobz</i> = 'V')、すべての固有ベクトルが計算されることを保証するために必要なワークスペースの量は次のとおりである。  <math>lwork \geq 5 * n + \max(5 * nn, np0 * mq0 + 2 * NB * NB) + \text{iceil}(\text{neig}, \text{NPROW} * \text{NPCOL}) * nn</math>  最小のワークスペースが与えられ、<i>orfac</i> が小さすぎる場合、計算されるベクトル固有値は直交ではない。(パフォーマンスが低下しても) 直交性を保証するには、次の値を <i>lwork</i> に追加する。  <math>(\text{clustersize}-1)*n</math>  ここで、<i>clustersize</i> は、クラスタが近い固有値のセットとして定義された、最大クラスタの固有値の数。  <math>\{w(k), \dots, w(k+\text{clustersize}-1) \mid w(j+1) \leq w(j) + orfac*2*\text{norm}(A)\}</math></p>

変数定義:

```
neig = 要求された固有ベクトルの数
NB = desca(mb_) = desca(nb_) = descz(mb_) = descz(nb_)
nn = max(n, NB, 2)
desca(rsrc_) = desca(nb_) = descz(rsrc_) = descz(csrc_) = 0
np0 = numroc(nn, NB, 0, 0, NPROW)
mq0 = numroc(max(neig, NB, 2), NB, 0, 0, NPCOL) iceil(x, y) は
ceiling(x/y) を返す ScaLAPACK 関数。
```

*lwork* が小さすぎる場合:

*lwork* が直交性を保証するには小さすぎる場合、[p?syevx](#) は、固有値間の間隔が最も小さいクラスタで直交性を維持するように試みる。*lwork* が小さすぎて要求された固有値をすべて計算できない場合、計算は行われず、*info*=-23 が返される。*range*='V' の場合、[p?syevx](#) は固有値が計算されるまで要求されている固有ベクトルがわからない点に注意する。したがって、*range*='V' で *lwork* が十分大きく [p?syevx](#) で固有値が計算可能な場合、[p?syevx](#) は固有値とできるだけ多くの固有ベクトルを計算する。

ワークスペース、直交性とパフォーマンスの関係:

適切なワークスペースが提供されれば、より優れたパフォーマンスを達成できる。逆に、状況によっては、以下に示すワークスペースよりも多くのワークスペースが提供されると、パフォーマンスが低下することがある。

最適なパフォーマンスを得るには、より大きなワークスペースが必要である。すなわち、

$$lwork \geq \max(lwork, 5 * n + nsytrd\_lwopt)$$

ここで、*lwork* は、上で定義され、要求された固有ベクトルの数に依存する。また、

$$nsytrd\_lwopt = n + 2 * (anb + 1) * (4 * nps + 2) + (nps + 3) * nps$$

$$anb = pjlaenv(desca(ctxt_), 3, 'p?sytttrd', 'L', 0, 0, 0, 0)$$

$$sqnpc = \text{int}(\text{sqrt}(\text{dble}(\text{NPROW} * \text{NPCOL})))$$

$$nps = \max(\text{numroc}(n, 1, 0, 0, sqnpc), 2 * anb)$$

*numroc* は ScaLAPACK ツール関数である。

*pjlaenv* は ScaLAPACK 環境問い合わせ関数である。

*MYROW*、*MYCOL*、*NPROW*、および *NPCOL* は、サブルーチン

*blacs\_gridinfo* を呼び出して決定できる。

*n* が大きな場合でも追加のワークスペースは必要ないが、*n* が小さな場合にパフォーマンスが最も上昇するため、追加のワークスペース (プロセスあたり 1MB 未満) を提供するほうがよい。

$clustersize \geq n/\sqrt{NPROW*NPCOL}$  の場合、すべての固有ベクトルを直交して計算するために十分なスペースを提供すると、パフォーマンスが大幅に低下する。限界 (すなわち、 $clustersize = n-1$ ) では、[p?stein](#) は 1 プロセッサ上の ?stein よりもパフォーマンスが高くなることはない。

$clustersize = n/\sqrt{NPROW*NPCOL}$  の場合、すべての固有ベクトルの再直交化は、2 以上の係数では合計の実行時間が長くなる。

$clustersize > n/\sqrt{NPROW*NPCOL}$  の場合、実行時間はクラスタサイズの 2 乗で増加し、すべての他の係数は同等なままで十分なワークスペースが提供されていると仮定する。ワークスペースが少なくなると再直交化も少なくなるが、実行は速くなる。

$lwork = -1$  の場合、 $lwork$  はグローバル入力であり、ワークスペースのクエリとみなされ、ルーチンはすべての `work` 配列で最適なパフォーマンスのために必要なサイズだけを計算する。各値は該当する `work` 配列の最初のエントリとして返され、[pxerbla](#) はエラー・メッセージを生成しない。

*iwork* (ローカル) INTEGER。ワークスペース配列。

*liwork* (ローカル) INTEGER。 *iwork* の次元。  
 $liwork \geq 6 * nnp$

ここで、 $nnp = \max(n, NPROW*NPCOL + 1, 4)$

$liwork = -1$  の場合、 $liwork$  はグローバル入力であり、ワークスペースのクエリとみなされ、ルーチンはすべての `work` 配列の最小かつ最適なサイズだけを計算する。各値は該当する `work` 配列の最初のエントリとして返され、[pxerbla](#) はエラー・メッセージを生成しない。

## 出力パラメータ

*a* 終了時に、行列 *A* の下三角部分 ( $uplo = 'L'$  の場合) または上三角部分 ( $uplo = 'U'$  の場合) が、対角成分を含めて上書きされる。

*m* (グローバル) INTEGER。検出された固有値の総数 ( $0 \leq m \leq n$ )。

*w* (グローバル)。

REAL (pssyevx の場合)

DOUBLE PRECISION (pdsyevx の場合)

配列、次元は  $n$ 。

*w* の先頭から  $m$  個の成分には、指定された固有値が昇順で格納される。

*z* (ローカル)。

REAL (pssyevx の場合)

DOUBLE PRECISION (pdsyevx の場合)

配列、グローバル次元 ( $n, n$ )、ローカル次元 ( $lld\_z, LOCc(jz+n-1)$ )。  
 $jobz = 'V'$  の場合、正常終了時に、 $z$  の先頭の  $m$  列には、選択された固有値に対応する正規直交固有ベクトルが格納される。固有ベクトルが収束に失敗した場合、 $z$  のその列には、その固有ベクトルに対する最新の近似値が格納され、固有ベクトルのインデックスは  $ifail$  に戻される。

$jobz = 'N'$  の場合、 $z$  は参照されない。

**work(1)** 終了時に、最適なパフォーマンスを得るために適切なワークスペースを返す。

**iwork(1)** 終了時に、 $iwork(1)$  には、要求された整数ワークスペースの量が格納される。

**ifail** (グローバル) INTEGER。配列、次元は ( $n$ )。  
 $jobz = 'V'$  の場合、正常終了時に、 $ifail$  の先頭の  $m$  個の成分はゼロである。終了時に、 $(mod(info,2).ne.0)$  の場合、 $ifail$  には収束に失敗した固有ベクトルのインデックスが格納される。  
 $jobz = 'N'$  の場合、 $ifail$  は参照されない。

**iclustr** (グローバル) INTEGER。  
 配列、次元は ( $2 * NPROW * NPCOL$ )。  
 この配列には、ワークスペースが十分でなかったために再直交できなかった、固有値のクラスタに対応する固有ベクトルのインデックスが格納される ( $lwork$ 、 $orfac$  および  $info$  を参照)。インデックス  $iclustr(2*i-1)$  から  $iclustr(2*i)$  の固有値のクラスタに対応する固有ベクトルは、ワークスペースの不足により再直交できなかった。したがって、これらのクラスタに対応する固有ベクトルは直交しない。  
 $iclustr()$  は、ゼロで終了する配列である。( $iclustr(2*k).ne.0.and. iclustr(2*k+1).eq.0$ ) ( $k$  がクラスタ数の場合)。  
 $jobz = 'N'$  の場合、 $iclustr$  は参照されない。

**gap** (グローバル)  
 REAL ( $pssyevx$  の場合)  
 DOUBLE PRECISION ( $pdsyevx$  の場合)  
 配列、次元は ( $NPROW * NPCOL$ )。  
 この配列には、固有ベクトルが再直交できなかった固有値間のギャップが格納される。この配列の出力値は、配列  $iclustr$  によって示されたクラスタに対応する。その結果、 $i$  番目のクラスタに対応する固有ベクトルの内積は、 $(C * n) / gap(i)$  と同程度になる。ここで、 $C$  は小さな定数である。

*info* (グローバル) INTEGER。  
*info* = 0 の場合、実行は正常に終了したことを示す。  
*info* < 0 の場合：  
*i* 番目の引数が配列で、*j* 番目の値が不正だった場合、  
*info* = -(*i*\*100+*j*)。 *i* 番目の引数がスカラで値が不正だった場合、  
*info* = -*i*。  
*info* > 0 の場合：  
(mod(*info*,2).ne.0) の場合、1 つ以上の固有ベクトルは収束に失敗したことを示す。それらのインデックスは *ifail* に格納される。  
*abstol* = 2.0\*p?lamch('U') でなければならない。  
(mod(*info*/2,2).ne.0) の場合、1 つ以上の固有値のクラスタに対する固有ベクトルが、ワークスペースが十分でなかったために再直交化できなかった。クラスタのインデックスは、配列 *iclustr* に格納される。  
(mod(*info*/4,2).ne.0) の場合、空間の制限により、[p?syevx](#) で *vl* と *vu* 間のすべての固有ベクトルを計算できなかった。計算された固有ベクトルの個数は、*nz* に返される。  
(mod(*info*/8,2).ne.0) の場合、[p?stebz](#) は、固有値の計算に失敗したことを示す。*abstol* = 2.0\*p?lamch('U') でなければならない。

## p?heevx

エルミート行列の固有値と ( オプションで )  
固有ベクトルを選択的に計算する。

### 構文

```
call pcheevx(jobz, range, uplo, n, a, ia, ja, desca, vl, vu, il, iu,
             abstol, m, nz, w, orfac, z, iz, jz, descz, work, lwork, rwork, lrwork,
             iwork, liwork, ifail, iclustr, gap, info)

call pzheevx(jobz, range, uplo, n, a, ia, ja, desca, vl, vu, il, iu,
             abstol, m, nz, w, orfac, z, iz, jz, descz, work, lwork, rwork, lrwork,
             iwork, liwork, ifail, iclustr, gap, info)
```

### 説明

このルーチンは、ScaLAPACK ルーチンの推奨される組み合わせの呼び出しによって、複素エルミート行列 *A* の固有値と ( オプションで ) 固有ベクトルを選択的に計算する。固有値と固有ベクトルを選択するには、希望する固有値の値の範囲かインデックスの範囲を指定する。

## 入力パラメータ

*np* = 与えられたプロセスに対するローカルの行数。

*nq* = 与えられたプロセスに対するローカルの列数。

<i>jobz</i>	(グローバル) CHARACTER*1。 'N' または 'V' でなければならない。 固有ベクトルを計算する場合に指定する。 <i>jobz</i> = 'N' の場合、固有値のみ計算する。 <i>jobz</i> = 'V' の場合、固有値と固有ベクトルを計算する。
<i>range</i>	(グローバル) CHARACTER*1。 'A'、'V'、または 'I' でなければならない。 <i>range</i> = 'A' の場合、固有値をすべて計算する。 <i>range</i> = 'V' の場合、半開区間 [ <i>vl</i> , <i>vu</i> ] の固有値をすべて計算する。 <i>range</i> = 'I' の場合、インデックスが <i>il</i> ~ <i>iu</i> の固有値を計算する。
<i>uplo</i>	(グローバル) CHARACTER*1。 'U' または 'L' でなければならない。 エルミート行列 <i>A</i> の上三角部分と下三角部分のどちらが格納されるかを指定する。 <i>uplo</i> = 'U' の場合、 <i>a</i> には <i>A</i> の上三角部分が格納される。 <i>uplo</i> = 'L' の場合、 <i>a</i> には <i>A</i> の下三角部分が格納される。
<i>n</i>	(グローバル) INTEGER。行列 <i>A</i> ( $n \geq 0$ ) の行数と列数。
<i>a</i>	(ローカル)。 COMPLEX ( <i>pcheevx</i> の場合) DOUBLE COMPLEX ( <i>pzheevx</i> の場合)。 グローバル次元 ( <i>n</i> , <i>n</i> ) とローカル次元 ( <i>lld_a</i> , <i>LOCc(ja+n-1)</i> ) のブロック・サイクリック配列。呼び出し時は、エルミート行列 <i>A</i> 。 <i>uplo</i> = 'U' の場合、行列 <i>A</i> の上三角部分だけがエルミート行列の成分を定義するために使用される。 <i>uplo</i> = 'L' の場合、行列 <i>A</i> の下三角部分だけがエルミート行列の成分を定義するために使用される。
<i>ia, ja</i>	(グローバル) INTEGER。 それぞれ、部分行列 <i>A</i> の最初の行と最初の列を示す、グローバル配列 <i>a</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。 分散行列 <i>A</i> の配列ディスクリプタ。 <i>desca(ctxt_)</i> が正しくない場合、 <a href="#">p?heevx</a> での正しいエラー報告は保証されない。

<i>vl, vu</i>	<p>(グローバル)</p> <p>REAL (pcheevx の場合)</p> <p>DOUBLE PRECISION (pzheevx の場合)。</p> <p><i>range</i>='V' の場合、固有値を探す区間の下限値と上限値。<i>range</i>='A' または 'I' の場合は参照されない。</p>
<i>il, iu</i>	<p>(グローバル)</p> <p>INTEGER。 <i>range</i>='I' の場合、求める固有値の最小インデックスと最大インデックス。次の制約がある。</p> <p><math>il \geq 1</math></p> <p><math>\min(il, n) \leq iu \leq n</math></p> <p><i>range</i>='A' または 'V' の場合は参照されない。</p>
<i>abstol</i>	<p>(グローバル)。</p> <p>REAL (pcheevx の場合)</p> <p>DOUBLE PRECISION (pzheevx の場合)。</p> <p><i>jobz</i>='V' の場合、<i>abstol</i> を <code>p?lamch(context, 'U')</code> に設定すると、ほとんどの直交固有ベクトルが生成される。</p> <p>固有値に対する絶対エラー許容値。<math>abstol + eps * \max( a ,  b )</math> 以下の幅の区間 <math>[a, b]</math> 内に存在していると判別された場合 (<i>eps</i> はマシン精度)、近似固有値は、収束したものとして受け入れられる。<i>abstol</i> がゼロまたは負の場合、代わりに <math>eps * \text{norm}(T)</math> を使用する。ここで、<math>\text{norm}(T)</math> は、<i>A</i> を三重対角形式に縮退して得られた三重対角行列の 1- ノルムである。</p> <p>固有値が最も正確に計算されるのは、<i>abstol</i> がゼロでなく、アンダーフローのしきい値の 2 倍、<math>2 * p?lamch('S')</math> に設定されたときである。このルーチンで <math>((\text{mod}(\text{info}, 2).ne.0).or. (\text{mod}(\text{info}/8, 2).ne.0))</math> が返された場合、固有値または固有ベクトルのいくつかは収束に失敗したことを意味するので、<i>abstol</i> を <math>2 * p?lamch('S')</math> に設定して、やり直してみる。</p>
<i>orfac</i>	<p>(グローバル)。</p> <p>REAL (pcheevx の場合)</p> <p>DOUBLE PRECISION (pzheevx の場合)。</p> <p>再直交化される固有ベクトルを指定する。互いの <math>tol = orfac * \text{norm}(A)</math> 内にある固有値に対応する固有ベクトルは再直交化される。しかし、ワークスペースが十分でない場合 (<i>lwork</i> を参照)、<i>tol</i> は、すべての固有ベクトルが 1 つのプロセスで再直交化されるようになるまで下げられる。<i>orfac</i> がゼロの場合、再直交化は行われない。<i>orfac</i> が負の場合、デフォルト値の <math>10^3</math> が使用される。<i>orfac</i> はすべての処理で同一でなければならない。</p>

<i>iz, jz</i>	(グローバル) INTEGER。 それぞれ、部分行列 <i>Z</i> の最初の行と最初の列を示す、グローバル配列 <i>z</i> の行インデックスと列インデックス。
<i>descz</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。 分散配列 <i>Z</i> のディスクリプタ。 <i>descz(ctxt_)</i> は <i>desca(ctxt_)</i> と等しくなければならない。
<i>work</i>	(ローカル) COMPLEX ( <i>pcheevx</i> の場合) DOUBLE COMPLEX ( <i>pzheevx</i> の場合)。 配列、次元は ( <i>lwork</i> )。
<i>lwork</i>	(ローカル) INTEGER。配列 <i>work</i> の次元。 固有値のみが要求された場合： $lwork \geq n + \max(NB * (np0 + 1), 3)$ 固有ベクトルが要求された場合： $lwork \geq n + (np0 + mq0 + NB) * NB$ $mq0 = \text{numroc}(nn, NB, 0, 0, NPCOL)$ $lwork \geq 5 * n + \max(5 * nn, np0 * mq0 + 2 * NB * NB) + \text{iceil}(neig, NPROW * NPCOL) * nn$ <p>最適なパフォーマンスを得るには、より大きなワークスペースが必要である。すなわち、  <math display="block">lwork \geq \max(lwork, nhetr\_lwork)</math> ここで、<i>lwork</i> は、上で定義され、  <math display="block">nhetr\_lwork = n + 2 * (anb + 1) * (4 * nps + 2) + (nps + 1) * nps</math> <math display="block">ictxt = \text{desca}(ctxt\_)</math> <math display="block">anb = \text{pjlaenv}(ictxt, 3, 'pchettrd', 'L', 0, 0, 0, 0)</math> <math display="block">sqnpc = \text{sqrt}(\text{dble}(NPROW * NPCOL))</math> <math display="block">nps = \max(\text{numroc}(n, 1, 0, 0, sqnpc), 2 * anb)</math>   <i>lwork</i> = -1 の場合、<i>lwork</i> はグローバル入力であり、ワークスペースのクエリとみなされ、ルーチンはすべての <i>work</i> 配列で最適なパフォーマンスのために必要なサイズだけを計算する B 各値は該当する <i>work</i> 配列の最初のエントリとして返され、<a href="#">pxerbla</a> はエラー・メッセージを生成しない。</p>
<i>rwork</i>	(ローカル) REAL ( <i>pcheevx</i> の場合) DOUBLE PRECISION ( <i>pzheevx</i> の場合)。 ワークスペース配列、次元は ( <i>lrwork</i> )。



*lwork*

(ローカル)

INTEGER。配列 *work* の次元。

*lwork* の定義に使用する変数の定義は以下を参照。固有ベクトルが要求されなかった場合 (*jobz* = 'N'),  $lwork \geq 5 * nn + 4 * n$ 。固有ベクトルが要求された場合 (*jobz* = 'V'), すべての固有ベクトルが計算されることを保証するために必要なワークスペースの量は次のとおりである。

$$lwork \geq 4 * n + \max(5 * nn, np0 * mq0 + 2 * NB * NB) + \text{iceil}(neig, \text{NPROW} * \text{NPCOL}) * nn$$

最小のワークスペースが与えられ、*orfac* が小さすぎる場合、計算されるベクトル固有値は直交ではない。(パフォーマンスが低下しても) 直交性を保証するには、次の値を *lwork* に追加する。

$$(clustersize-1) * n$$

ここで、*clustersize* は、クラスタが近い固有値のセットとして定義された、最大クラスタの固有値の数。

$$\{w(k), \dots, w(k+clustersize-1) \mid w(j+1) \leq w(j)\} + orfac * 2 * \text{norm}(A)$$

変数定義:

*neig* = 要求された固有ベクトルの数*NB* = *desca*(*mb\_*) = *desca*(*nb\_*) = *descz*(*mb\_*) = *descz*(*nb\_*)*nn* = max(*n*, *NB*, 2)*desca*(*rsrc\_*) = *desca*(*nb\_*) = *descz*(*rsrc\_*) = *descz*(*csrc\_*) = 0*np0* = numroc(*nn*, *NB*, 0, 0, *NPROW*)

*mq0* = numroc(max(*neig*, *NB*, 2), *NB*, 0, 0, *NPCOL*) *iceil*(*x*, *y*) は *ceiling*(*x/y*) を返す ScaLAPACK 関数。

*lwork* が小さすぎる場合:

*lwork* が直交性を保証するには小さすぎる場合、[p?heevx](#) は、固有値間の間隔が最も小さいクラスタで直交性を維持するように試みる。

*lwork* が小さすぎて要求された固有値をすべて計算できない場合、計算は行われず、*info* = -23 が返される。*range* = 'V' の場合、[p?heevx](#) は固有値が計算されるまで要求されている固有ベクトルがわからない点に注意する。したがって、*range* = 'V' で *lwork* が十分大きく [p?heevx](#) で固有値が計算可能な場合、[p?heevx](#) は固有値とできるだけ多くの固有ベクトルを計算する。

ワークスペース、直交性とパフォーマンスの関係:

$clustersize \geq n / \sqrt{\text{NPROW} * \text{NPCOL}}$  の場合、すべての固有ベクトルを直交して計算するために十分なスペースを提供すると、パフォーマンスが大幅に低下する。限界 (すなわち、*clustersize* = *n*-1) では、[p?stein](#) は 1 プロセッサ上の *?stein* よりもパフォーマンスが高くな

ることではない。

$clustersize = n/\sqrt{NPROW*NPCOL}$  の場合、すべての固有ベクトルの再直交化は、2 以上の係数では合計の実行時間が長くなる。

$clustersize > n/\sqrt{NPROW*NPCOL}$  の場合、実行時間はクラスタサイズの 2 乗で増加し、すべての他の係数は同等なままで十分なワークスペースが提供されていると仮定する。ワークスペースが少なくなると再直交化も少なくなるが、実行は速くなる。

$lwork = -1$  の場合、 $lwork$  はグローバル入力であり、ワークスペースのクエリとみなされ、ルーチンはすべての `work` 配列で最適なパフォーマンスのために必要なサイズだけを計算する。各値は該当する `work` 配列の最初のエントリとして返され、[pxerbla](#) はエラー・メッセージを生成しない。

`iwork` (ローカル) INTEGER。ワークスペース配列。

`liwork` (ローカル) INTEGER。 `iwork` の次元。  
 $liwork \geq 6 * nnp$

ここで、 $nnp = \max(n, NPROW*NPCOL + 1, 4)$

$liwork = -1$  の場合、 $liwork$  はグローバル入力であり、ワークスペースのクエリとみなされ、ルーチンはすべての `work` 配列の最小かつ最適なサイズだけを計算する。各値は該当する `work` 配列の最初のエントリとして返され、[pxerbla](#) はエラー・メッセージを生成しない。

## 出力パラメータ

`a` 終了時に、行列 `A` の下三角部分 ( $uplo = 'L'$  の場合) または上三角部分 ( $uplo = 'U'$  の場合) が、対角成分を含めて上書きされる。

`m` (グローバル) INTEGER。  
 検出された固有値の総数 ( $0 \leq m \leq n$ )。

`nz` (グローバル) INTEGER。  
 計算された固有ベクトルの総数 ( $0 \leq nz \leq m$ )。  
`z` の列数。  
`jobz.ne. 'V'` の場合、`nz` は参照されない。  
`jobz.eq. 'V'` の場合、 $nz = m$  (ユーザが十分な空間を提供しないで、[p?heevx](#) が計算を開始する前にこれを検出できない場合を除く)。要求されたすべての固有ベクトルを得るには、ユーザは `z(m.le. descz(n_))` の固有ベクトルを保持するための十分な空間とそれらを計算するための十分なワークスペースの両方を提供しなければならない。( $lwork$  を参照)。[p?heevx](#) は、`range.eq. 'V'` の場合を除いて、計算を開始する前に空間が十分でないことを常に検出できる。

<i>w</i>	<p>(グローバル)。  REAL (pcheevx の場合)  DOUBLE PRECISION (pzheevx の場合)  配列、次元は <math>n</math>。  <i>w</i> の先頭から <math>m</math> 個の成分には、指定された固有値が昇順で格納される。</p>
<i>z</i>	<p>(ローカル)。  COMPLEX (pcheevx の場合)  DOUBLE COMPLEX (pzheevx の場合)  配列、グローバル次元 (<math>n, n</math>)、ローカル次元 (<math>lld\_z, LOCc(jz+n-1)</math>)。  <math>jobz = 'V'</math> の場合、正常終了時に、<i>z</i> の先頭の <math>m</math> 列には、選択された固有値に対応する正規直交固有ベクトルが格納される。固有ベクトルが収束に失敗した場合、<i>z</i> のその列には、その固有ベクトルに対する最新の近似値が格納され、固有ベクトルのインデックスは <i>ifail</i> に戻される。  <math>jobz = 'N'</math> の場合、<i>z</i> は参照されない。</p>
<i>work(1)</i>	<p>終了時に、最適なパフォーマンスを得るために適切なワークスペースを返す。</p>
<i>rwork</i>	<p>(ローカル)。  REAL (pcheevx の場合)  DOUBLE PRECISION (pzheevx の場合)  配列、次元は <math>lrwork</math>。  終了時に、<i>rwork(1)</i> には、効率的に実行するために必要なワークスペースの最適な量が格納される。  <math>jobz = 'N'</math> の場合、<i>rwork(1)</i> には、固有値を効率的に計算するために必要なワークスペースの最適な量が格納される。  <math>jobz = 'V'</math> の場合、<i>rwork(1)</i> には、直交性を保証しないで固有値と固有ベクトルを効率的に計算するために必要なワークスペースの最適な量が格納される。  <math>range = 'V'</math> の場合、すべての固有ベクトルが必要であるとみなされる。</p>
<i>iwork(1)</i>	<p>(ローカル)  終了時に、<i>iwork(1)</i> には、要求された整数ワークスペースの量が格納される。</p>
<i>ifail</i>	<p>(グローバル) INTEGER。  配列、次元は (<math>n</math>)。  <math>jobz = 'V'</math> の場合、正常終了時に、<i>ifail</i> の先頭の <math>m</math> 個の成分はゼロである。終了時に、<math>(mod(info,2).ne.0)</math> の場合、<i>ifail</i> には収束に失敗した固有ベクトルのインデックスが格納される。  <math>jobz = 'N'</math> の場合、<i>ifail</i> は参照されない。</p>

<i>iclustr</i>	<p>(グローバル) INTEGER。</p> <p>配列、次元は (2*NPROW*NPCOL)。</p> <p>この配列には、ワークスペースが十分でなかったために再直交できなかった、固有値のクラスタに対応する固有ベクトルのインデックスが格納される (<i>lwork</i>, <i>orfac</i> および <i>info</i> を参照)。インデックス <i>iclustr</i>(2*i-1) から <i>iclustr</i>(2*i) の固有値のクラスタに対応する固有ベクトルは、ワークスペースの不足により再直交できなかった。したがって、これらのクラスタに対応する固有ベクトルは直交しない。<i>iclustr</i>() は、ゼロで終了する配列である。</p> <p>(<i>iclustr</i>(2*k).ne.0.and. <i>iclustr</i>(2*k+1).eq.0) (<i>k</i> がクラスタ数の場合)。</p> <p><i>jobz</i> = 'N' の場合、<i>iclustr</i> は参照されない。</p>
<i>gap</i>	<p>(グローバル)</p> <p>REAL (<i>pcheevx</i> の場合)</p> <p>DOUBLE PRECISION (<i>pzheevx</i> の場合)</p> <p>配列、次元は (NPROW*NPCOL)。</p> <p>この配列には、固有ベクトルが再直交できなかった固有値間のギャップが格納される。この配列の出力値は、配列 <i>iclustr</i> によって示されたクラスタに対応する。その結果、<i>i</i> 番目のクラスタに対応する固有ベクトルの内積は、(<i>C</i> * <i>n</i>) / <i>gap</i>(<i>i</i>) と同程度になる。ここで、<i>C</i> は小さな定数である。</p>
<i>info</i>	<p>(グローバル) INTEGER。</p> <p><i>info</i> = 0 の場合、実行は正常に終了したことを示す。</p> <p><i>info</i> &lt; 0 の場合：</p> <p><i>i</i> 番目の引数が配列で、<i>j</i> 番目の値が不正だった場合、  <i>info</i> = -(<i>i</i>*100+<i>j</i>)。 <i>i</i> 番目の引数がスカラで値が不正だった場合、  <i>info</i> = -<i>i</i>。</p> <p><i>info</i> &gt; 0 の場合：</p> <p>(mod(<i>info</i>,2).ne.0) の場合、1 つ以上の固有ベクトルは収束に失敗したことを示す。それらのインデックスは <i>ifail</i> に格納される。  <i>abstol</i>=2.0*p?lamch('U') でなければならない。</p> <p>(mod(<i>info</i>/2,2).ne.0) の場合、1 つ以上の固有値のクラスタに対する固有ベクトルが、ワークスペースが十分でなかったために再直交化できなかった。クラスタのインデックスは、配列 <i>iclustr</i> に格納される。</p> <p>(mod(<i>info</i>/4,2).ne.0) の場合、空間の制限により、<a href="#">p?syevx</a> で <i>vl</i> と <i>vu</i> 間のすべての固有ベクトルを計算できなかった。計算された固有ベクトルの個数は、<i>nz</i> に返される。</p> <p>(mod(<i>info</i>/8,2).ne.0) の場合、<a href="#">p?stebz</a> は、固有値の計算に失敗したことを示す。<i>abstol</i>=2.0*p?lamch('U') でなければならない。</p>

## p?gesvd

一般行列の特異値分解と ( オプションで ) 左特異ベクトルまたは右特異ベクトルを計算する。

### 構文

```
call psgesvd( jobu, jobvt, m, n, a, ia, ja, desca, s, u, iu, ju, descu,
              vt, ivt, jvt, descvt, work, lwork, info )
call pdgesvd( jobu, jobvt, m, n, a, ia, ja, desca, s, u, iu, ju, descu,
              vt, ivt, jvt, descvt, work, lwork, info )
```

### 説明

このルーチンは、行列  $A (m \times n)$  の特異値分解 (SVD) と ( オプションで ) 左特異ベクトルまたは右特異ベクトルを計算する。SVD は、次のように記述される。

$$A = U \Sigma V^T$$

ここで、 $\Sigma$  はその  $\min(m, n)$  対角成分を除いてゼロの  $m \times n$  の行列、 $U$  は  $m \times m$  の直交行列、 $V$  は  $n \times n$  の直交行列である。 $\Sigma$  の対角成分は  $A$  の特異値で、 $U$  と  $V$  の列は、それぞれ、右と左の特異ベクトルに対応する。特異値は  $s$  に降順で返され、 $U$  の先頭から  $\min(m, n)$  の列と  $vt = V^T$  の行のみが計算される。

### 入力パラメータ

$mp = A$  と  $U$  のローカル行数

$nq = A$  と  $VT$  のローカル列数

$size = \min(m, n)$

$sizeq = U$  のローカル列数

$sizep = VT$  のローカル行数

$jobu$  (グローバル) CHARACTER\*1。

行列  $U$  のすべて、または一部を計算するオプションを指定する。

$jobu = 'V'$  の場合、 $U$  の先頭から  $size$  列 ( 左特異ベクトル ) が配列  $u$  に返される。

$jobu = 'N'$  の場合、 $U$  の列 ( 左特異ベクトル ) は計算されない。

$jobvt$  (グローバル) CHARACTER\*1。

行列  $V^T$  のすべて、または一部を計算するオプションを指定する。

	$jobvt = 'V'$ の場合、 $V^T$ の先頭から $size$ 行 (右特異ベクトル) が配列 $vt$ に返される。 $jobvt = 'N'$ の場合、 $V^T$ の行 (右特異ベクトル) は計算されない。
$m$	(グローバル) INTEGER。行列 $A$ の行数 ( $m \geq 0$ )。
$n$	(グローバル) INTEGER。行列 $A$ の列数 ( $n \geq 0$ )。
$a$	(ローカル)。 DOUBLE PRECISION (psgesvd および pdgesvd の場合) ブロック・サイクリック配列、グローバル次元 ( $m, n$ )、ローカル次元 ( $mp, nq$ )。 $work(lwork)$ は、ワークスペース配列である。
$ia, ja$	(グローバル) INTEGER。 それぞれ、部分行列 $A$ の最初の行と最初の列を示す、グローバル配列 $a$ の行インデックスと列インデックス。
$desca$	(グローバルおよびローカル) INTEGER。配列、次元は ( $dlen\_$ )。 分散行列 $A$ の配列ディスクリプタ。
$iu, ju$	(グローバル) INTEGER。 それぞれ、部分行列 $U$ の最初の行と最初の列を示す、グローバル配列 $a$ の行インデックスと列インデックス。
$descu$	(グローバルおよびローカル) INTEGER。配列、次元は ( $dlen\_$ )。 分散行列 $U$ の配列ディスクリプタ。
$ivt, jvt$	(グローバル) INTEGER。 それぞれ、部分行列 $VT$ の最初の行と最初の列を示す、グローバル配列 $vt$ の行インデックスと列インデックス。
$descvt$	(グローバルおよびローカル) INTEGER。配列、次元は ( $dlen\_$ )。 分散行列 $VT$ の配列ディスクリプタ。
$work$	(ローカル) DOUBLE PRECISION (psgesvd および pdgesvd の場合) ワークスペース配列、次元は $lwork$ 。
$lwork$	(ローカル) INTEGER。配列 $work$ の次元。 $lwork > 2 + 6 * sizeb + \max(watobd, wbdto svd)$

ここで、 $sizeb = \max(m, n)$ 、 $watobd$  および  $wbdto svd$  は、行列  $A$  を二重対角形式にし、二重対角行列から特異値分解  $USVT$  に移動するため、それぞれ必要なワークスペースを参照する。

*watobd* には、次の内容が含まれている。

```
watobd = max(max(wpslange,wpsgebrd),
max(wpslared2d,wpslared1d))
```

ここで、*wpslange*、*wpslared1d*、*wpslared2d*、*wpsgebrd* は、それぞれ、サブプログラム *pslange*、*pslared1d*、*pslared2d*、*psgebrd* に必要なワークスペースである。標準の表記を使用すると、次のようになる。

```
mp = numroc(m, mb, MYROW, desca(ctxt_), NPROW),
nq = numroc(n, NB, MYCOL, desca(1ld_), NPCOL)
```

上記サブプログラムに必要なワークスペースは次のとおりである。

```
wpslange = mp,
wpslared1d = nq0,
wpslared2d = mp0,
wpsgebrd = NB*(mp + nq + 1) + nq
```

ここで、*nq0* と *mp0* は、それぞれ、*MYCOL* = 0 と *MYROW* = 0 で得られた値を参照する。一般的に、ワークスペースの上限は、プロセッサ (0,0) で必要なワークスペースによって与えられる。

$$watobd \leq NB*(mp0 + nq0 + 1) + nq0$$

均一なプロセスグリッドの場合、この上限は各プロセッサの最小ワークスペースの推定として使用できる。

*wbdtosvd* には、次の内容が含まれている。

```
wbdtosvd = size*(wantu*nru + wantvt*ncvt) + max(wsbdsqr,
max(wantu*wpsormbrqln, wantvt*wpsormbrprt))
```

ここで、

*wantu*(*wantvt*) = 1 ( 左 ( 右 ) 特異ベクトルが必要な場合 ) および  
*wantu*(*wantvt*) = 0 ( それ以外の場合 )

*wsbdsqr*、*wpsormbrqln* および *wpsormbrprt* は、それぞれ、サブプログラム [sbdsqr](#)、[p?ormbr](#)(*qln*)、および [p?ormbr](#)(*prt*) に必要なワークスペース。ここで、*qln* と *prt* は、[p?ormbr](#) への呼び出しで指定する、引数 *vect*、*side*、および *trans* の値。*nru* は、プロセスの 1 次元の「列」を分散したときの、行列 *U* のローカル行数に等しい。同様に、*ncvt* は、プロセスの 1 次元の「行」を分散したときの、行列 *VT* のローカル列数に等しい。LAPACK プロシージャ *sbdsqr* を呼び出すには、各プロセッサで、

$$wsbdsqr = \max(1, 2*size + (2*size - 4)* \max(wantu, wantvt))$$

が必要である。最終的には、次のとおり。

$$wpsormbrqln = \max((NB*(NB-1))/2, (sizeq+mp)*NB)+NB*NB,$$

$$wpsormbrprt = \max((mb*(mb-1))/2, (sizep+nq)*mb)+mb*mb$$

$lwork = -1$  の場合、 $lwork$  はグローバル入力であり、ワークスペースのクエリとみなされ、ルーチンは  $work$  配列の最小サイズだけを計算する。必要なワークスペースは  $work$  の最初の成分として返され、[pxerbla](#) はエラー・メッセージを生成しない。

## 出力パラメータ

$a$	終了時に、 $a$ の内容は削除される。
$s$	(グローバル)。 DOUBLE PRECISION (psgesvd と pdgesvd の場合)。 配列、次元は ( $size$ )。 $s(i) \geq s(i+1)$ となるようにソートされて $A$ の特異値が格納される。
$u$	(ローカル)。 DOUBLE PRECISION (psgesvd および pdgesvd の場合) ローカル次元 ( $mp, sizeq$ )、グローバル次元 ( $m, size$ ) $jobu = 'V'$ の場合、 $u$ は $U$ の先頭の $\min(m, n)$ 列を格納する。 $jobu = 'N'$ または $'O'$ の場合、 $u$ は参照されない。
$vt$	(ローカル) DOUBLE PRECISION (psgesvd および pdgesvd の場合) ローカル次元 ( $sizep, nq$ )、グローバル次元 ( $size, n$ ) $jobvt = 'V'$ の場合、 $VT$ は $V^T$ の先頭の $size$ 行を格納する。 $jobu = 'N'$ の場合、 $VT$ は参照されない。
$work$	終了時に、 $info = 0$ の場合、 $work(1)$ は、 $lwork$ の必要最小サイズを返す。
$rwork$	(複素数型の場合) 終了時に、 $info > 0$ の場合、 $rwork(1:\min(m, n)-1)$ には、対角が (必ずしもソートされていない) $s$ 内にある上二重対角行列 $B$ の非収束優対角成分が格納される。 $B$ は、 $A = u * B * vt$ を満たすため、 $A$ と同じ特異値を持ち、 $u$ と $vt$ で関連付けられた特異ベクトルを持つ。
$info$	(グローバル) INTEGER。 $info = 0$ の場合、実行は正常に終了したことを示す。 $info < 0$ の場合: $info = -i$ の場合、 $i$ 番目のパラメータの値が不正だったことを示す。



$info > 0$  の場合 :  
 $info = i$  の場合、`p?bdsqr` は収束しなかった。  
 $info = \min(m,n) + 1$  の場合、[p?gesvd](#) 固有値がプロセスグリッドで同一でなかったことから不均一性を検出した。この場合、[p?gesvd](#) の結果は保証されない。

## p?sygvx

実数の汎用対称固有値問題について、固有値と ( オプションで ) 固有ベクトルを選択的に計算する。

### 構文

```
call pssygvx(ibtype, jobz, range, uplo, n, a, ia, ja, desca, b, ib, jb,
             descb, vl, vu, il, iu, abstol, m, nz, w, orfac, z, iz, jz, descz,
             work, lwork, iwork, liwork, ifail, iclustr, gap, info)

call pdsygvx(ibtype, jobz, range, uplo, n, a, ia, ja, desca, b, ib, jb,
             descb, vl, vu, il, iu, abstol, m, nz, w, orfac, z, iz, jz, descz,
             work, lwork, iwork, liwork, ifail, iclustr, gap, info)
```

### 説明

このルーチンは、次の形式で示される実数の汎用対称固有値問題について、固有値と ( オプションで ) 固有ベクトルを選択的に計算する。

$$\text{sub}(A)x = \lambda \text{sub}(B)x, \text{sub}(A) \text{sub}(B)x = \lambda x, \text{または} \text{sub}(B) \text{sub}(A)x = \lambda x.$$

ここで、 $A(ia:ia+n-1, ja:ja+n-1)$  を表す  $\text{sub}(A)$  は対称とみなされ、 $B(ib:ib+n-1, jb:jb+n-1)$  を表す  $\text{sub}(B)$  もまた、正定値とみなされる。

### 入力パラメータ

*ibtype* ( グローバル ) INTEGER。  
 1、2、または 3 のいずれかでなければならない。  
 解決する問題のタイプを指定する。  
 $ibtype = 1$  の場合、問題のタイプは  $\text{sub}(A)x = \lambda \text{sub}(B)x$  である。  
 $ibtype = 2$  の場合、問題のタイプは

	$\text{sub}(A)\text{sub}(B)x = \lambda x$ である。 $\text{ibtype} = 3$ の場合、問題のタイプは $\text{sub}(B)\text{sub}(A)x = \lambda x$ である。
<i>jobz</i>	(グローバル) CHARACTER*1。 'N' または 'V' でなければならない。 $\text{jobz} = 'N'$ の場合、固有値のみを計算する。 $\text{jobz} = 'V'$ の場合、固有値と固有ベクトルを計算する。
<i>range</i>	(グローバル) CHARACTER*1。 'A'、'V'、または 'I' でなければならない。 $\text{range} = 'A'$ の場合、固有値をすべて計算する。 $\text{range} = 'V'$ の場合、ルーチンは区間 $[v1, vu]$ の固有値を計算する。 $\text{range} = 'I'$ の場合、ルーチンはインデックスが $i1 \sim iu$ の固有値を計算する。
<i>uplo</i>	(グローバル) CHARACTER*1。 'U' または 'L' でなければならない。 $\text{uplo} = 'U'$ の場合、配列 <i>a</i> と <i>b</i> は、 $\text{sub}(A)$ と $\text{sub}(B)$ の上三角を格納する。 $\text{uplo} = 'L'$ の場合、配列 <i>a</i> と <i>b</i> は、 $\text{sub}(A)$ と $\text{sub}(B)$ の下三角を格納する。
<i>n</i>	(グローバル) INTEGER。行列 $\text{sub}(A)$ と $\text{sub}(B)$ の次数 ( $n \geq 0$ )。
<i>a</i>	(ローカル) REAL (pssygvx の場合) DOUBLE PRECISION (pdsygvx の場合)。 ローカルメモリにある、次元 ( $lld\_a, LOCc(ja+n-1)$ ) の配列へのポインタ。呼び出し時に、この配列は、 $n \times n$ の対称分散行列 $\text{sub}(A)$ のローカル部分を含む。 $\text{uplo} = 'U'$ の場合、 $\text{sub}(A)$ の先頭の $n \times n$ 上三角部分は行列の上三角部分を含む。 $\text{uplo} = 'L'$ の場合、 $\text{sub}(A)$ の先頭の $n \times n$ 下三角部分は行列の下三角部分を含む。
<i>ia, ja</i>	(グローバル) INTEGER。 それぞれ、部分行列 <i>A</i> の最初の行と最初の列を示す、グローバル配列 <i>a</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。 分散行列 <i>A</i> の配列ディスクリプタ。 $\text{desca}(ctxt\_)$ が正しくない場合、 <a href="#">p?sygvx</a> での正しいエラー報告は保証されない。
<i>b</i>	(ローカル)。 REAL (pssygvx の場合) DOUBLE PRECISION (pdsygvx の場合)。

	ローカルメモリにある、次元 ( $lld\_b, LOCc(jb+n-1)$ ) の配列へのポインタ。呼び出し時に、この配列は、 $n \times n$ の対称分散行列 $\text{sub}(B)$ のローカル部分を含む。 $uplo = 'U'$ の場合、 $\text{sub}(B)$ の先頭の $n \times n$ 上三角部分は行列の上三角部分を含む。 $uplo = 'L'$ の場合、 $\text{sub}(B)$ の先頭の $n \times n$ 下三角部分は行列の下三角部分を含む。
<i>ib, jb</i>	(グローバル) INTEGER。 それぞれ、部分行列 $B$ の最初の行と最初の列を示す、グローバル配列 $b$ の行インデックスと列インデックス。
<i>descb</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( $dlen\_$ )。分散行列 $B$ の配列ディスクリプタ。 $descb(ctxt\_)$ は $desca(ctxt\_)$ と等しくなければならない。
<i>v1, vu</i>	(グローバル) REAL (pssygvx の場合) DOUBLE PRECISION (pdsygvx の場合)。 $range = 'V'$ の場合、固有値を探す区間の下限値と上限値。 $range = 'A'$ または $'I'$ の場合、 $v1$ と $vu$ は参照されない。
<i>il, iu</i>	(グローバル) INTEGER。 $range = 'I'$ の場合、返される最小固有値と最大固有値に対する昇順のインデックス。 次の制約がある。 $il \geq 1, \min(il, n) \leq iu \leq n$ $range = 'A'$ または $'V'$ の場合、 $il$ と $iu$ は参照されない。
<i>abstol</i>	(グローバル) REAL (pssygvx の場合) DOUBLE PRECISION (pdsygvx の場合)。 $jobz = 'V'$ の場合、 $abstol$ を $p?lamch(context, 'U')$ に設定すると、ほとんどの直交固有ベクトルが生成される。 固有値に対する絶対エラー許容値。次に示す値以下の幅の区間 $[a, b]$ 内に存在していると判別された場合 ( $eps$ はマシン精度)、近似固有値は、収束したものとして受け入れられる。 $abstol + eps * \max( a ,  b )$ ここで、 $eps$ はマシン精度である。 $abstol$ がゼロまたは負の場合、代わりに $eps * \text{norm}(T)$ を使用する。ここで、 $\text{norm}(T)$ は、 $A$ を三重対角形式に縮退して得られた三重対角行列の 1- ノルムである。

固有値が最も正確に計算されるのは、*abstol* がゼロでなく、アンダーフローのしきい値の 2 倍、 $2 * p?lamch('S')$  に設定されたときである。このルーチンで  $((mod(info,2).ne.0).or.* (mod(info/8,2).ne.0))$  が返された場合、固有値または固有ベクトルのいくつか収束に失敗したことを意味するので、*abstol* を  $2 * p?lamch('S')$  に設定して、やり直してみる。

<i>orfac</i>	( グローバル )。 REAL (pssygvx の場合 ) DOUBLE PRECISION (pdsygvx の場合 )。 再直交化される固有ベクトルを指定する。互いの $tol=orfac * norm(A)$ 内にある固有値に対応する固有ベクトルは再直交される。しかし、ワークスペースが十分でない場合 ( <i>lwork</i> を参照 )、 <i>tol</i> は、すべての固有ベクトルが 1 つのプロセスで再直交化されるようになるまで下げられる。 <i>orfac</i> がゼロの場合、再直交化は行われず。 <i>orfac</i> が負の場合、デフォルト値の $10^{-3}$ が使用される。 <i>orfac</i> はすべての処理で同一でなければならない。
<i>iz,jz</i>	( グローバル ) INTEGER。 それぞれ、部分行列 <i>Z</i> の最初の行と最初の列を示す、グローバル配列 <i>z</i> の行インデックスと列インデックス。
<i>descz</i>	( グローバルおよびローカル ) INTEGER。配列、次元は ( <i>dlen_</i> )。 分散配列 <i>Z</i> のディスクリプタ。 <i>descz(ctxt_)</i> は <i>desca(ctxt_)</i> と等しくなければならない。
<i>work</i>	( ローカル ) REAL (pssygvx の場合 ) DOUBLE PRECISION (pdsygvx の場合 )。 ワークスペース配列、次元は ( <i>lwork</i> )。
<i>lwork</i>	( ローカル ) INTEGER。 <i>lwork</i> の定義に使用する変数の定義は以下を参照。 固有ベクトルが要求されなかった場合 ( <i>jobz</i> = 'N')、 $lwork \geq 5 * n + \max(5 * nn, NB * (np0 + 1))$ 。 固有ベクトルが要求された場合 ( <i>jobz</i> = 'V')、すべての固有ベクトルが計算されることを保証するために必要なワークスペースの量は次のとおりである。  $lwork \geq 5 * n + \max(5 * nn, np0 * mq0 + 2 * NB * NB) + \text{iceil}(neig, NPROW * NPCOL) * nn$

最小のワークスペースが与えられ、*orfac* が小さすぎる場合、計算されるベクトル固有値は直交ではない。(パフォーマンスが低下しても) 直交性を保証するには、次の値を *lwork* に追加する。

$(clustersize-1)*n$

ここで、*clustersize* は、クラスタが近い固有値のセットとして定義された、最大クラスタの固有値の数。

$\{w(k), \dots, w(k+clustersize-1) \mid w(j+1) \leq w(j) + orfac*2*norm(A)\}$

変数定義:

*neig* = 要求された固有ベクトルの数

*NB* = *desca*(*mb\_*) = *desca*(*nb\_*) = *descz*(*mb\_*) = *descz*(*nb\_*)

*nn* = max(*n*, *NB*, 2)

*desca*(*rsrc\_*) = *desca*(*nb\_*) = *descz*(*rsrc\_*) = *descz*(*csrc\_*) = 0

*np0* = numroc(*nn*, *NB*, 0, 0, *NPROW*)

*nq0* = numroc(max(*neig*, *NB*, 2), *NB*, 0, 0, *NPCOL*) iceil(*x*, *y*) は ceiling(*x/y*) を返す ScaLAPACK 関数。

*lwork* が小さすぎる場合:

*lwork* が直交性を保証するには小さすぎる場合、[p?sygvx](#) は、固有値間の間隔が最も小さいクラスタで直交性を維持するように試みる。

*lwork* が小さすぎて要求された固有値をすべて計算できない場合、計算は行われず、*info*=-23 が返される。*range*='V' の場合、[p?sygvx](#) は固有値が計算されるまで要求されている固有ベクトルがわからない点に注意する。したがって、*range*='V' で *lwork* が十分大きく [p?sygvx](#) で固有値が計算可能な場合、[p?sygvx](#) は固有値とできるだけ多くの固有ベクトルを計算する。

ワークスペース、直交性とパフォーマンスの関係:

適切なワークスペースが提供されれば、より優れたパフォーマンスを達成できる。逆に、状況によっては、以下に示すワークスペースよりも多くのワークスペースが提供されると、パフォーマンスが低下することがある。

最適なパフォーマンスを得るには、より大きなワークスペースが必要である。すなわち、

$lwork \geq \max(lwork, 5*n + nsytrd\_lwopt, nsygst\_lwopt)$

ここで、*lwork* は、上で定義され、要求された固有ベクトルの数に依存する。また、

$nsytrd\_lwopt = n + 2*(anb+1)*(4*nps+2) + (nps + 3) * nps$

$nsygst\_lwopt = 2*np0*NB + nq0*NB + NB*NB$

*anb* = *pjlaenv*(*desca*(*ctxt\_*), 3, *p?sytttrd*!, 'L', 0, 0, 0, 0)

*sqnpc* = int(sqrt(dble(*NPROW* \* *NPCOL*)))

```
nps = max(numroc(n, 1, 0, 0, sqnpc), 2*anb)
NB = desca(mb_)
np0 = numroc(n, NB, 0, 0, NPROW)
nq0 = numroc(n, NB, 0, 0, NPCOL)
```

numroc は ScaLAPACK ツール関数である。  
 pjlaenv は ScaLAPACK 環境問い合わせ関数である。  
 MYROW、MYCOL、NPROW、および NPCOL は、サブルーチン  
 blacs\_gridinfo を呼び出して決定できる。

$n$  が大きな場合でも追加のワークスペースは必要ないが、 $n$  が小さな場合にパフォーマンスが最も上昇するため、追加のワークスペース (プロセスあたり 1MB 未満) を提供するほうがよい。

$clustersize \geq n/\sqrt{NPROW \cdot NPCOL}$  の場合、すべての固有ベクトルを直交して計算するために十分なスペースを提供すると、パフォーマンスが大幅に低下する。限界 (すなわち、 $clustersize = n-1$ ) では、[p?stein](#) は 1 プロセッサ上の ?stein よりもパフォーマンスが高くなることはない。

$clustersize = n/\sqrt{NPROW \cdot NPCOL}$  の場合、すべての固有ベクトルの再直交化は、2 以上の係数では合計の実行時間が長くなる。

$clustersize > n/\sqrt{NPROW \cdot NPCOL}$  の場合、実行時間はクラスタサイズの 2 乗で増加し、すべての他の係数は同等なままで十分なワークスペースが提供されていると仮定する。ワークスペースが少なくなると再直交化も少なくなるが、実行は速くなる。

$lwork = -1$  の場合、 $lwork$  はグローバル入力であり、ワークスペースのクエリとみなされ、ルーチンはすべての  $work$  配列で最適なパフォーマンスのために必要なサイズだけを計算する。各値は該当する  $work$  配列の最初のエン트리として返され、[pxerbla](#) はエラー・メッセージを生成しない。

$iwork$  (ローカル) INTEGER。ワークスペース配列。

$liwork$  (ローカル) INTEGER。  $iwork$  の次元。  
 $liwork \geq 6 * nnp$

ここで、 $nnp = \max(n, NPROW \cdot NPCOL + 1, 4)$

$liwork = -1$  の場合、 $liwork$  はグローバル入力であり、ワークスペースのクエリとみなされ、ルーチンはすべての  $work$  配列の最小かつ最適なサイズだけを計算する。各値は該当する  $work$  配列の最初のエン트리として返され、[pxerbla](#) はエラー・メッセージを生成しない。

## 出力パラメータ

- a** 終了時に、`jobz = 'V'` で `info = 0` の場合、`sub(A)` には、固有ベクトルの分散行列  $Z$  が格納される。固有ベクトルは、次のように正規化される。  
 $ibtype = 1$  または  $2$  の場合  
 $Z^T \text{sub}(B) Z = I$ ;  
 $ibtype = 3$  の場合、 $Z^T \text{inv}(\text{sub}(B)) Z = I$ 。  
`jobz = 'N'` の場合、終了時に、`sub(A)` の上三角 (`uplo = 'U'` の場合) または下三角 (`uplo = 'L'` の場合) が対角を含めて破棄される。
- b** 終了時に、`info ≤ n` の場合、行列を格納している `sub(B)` の一部が、コレスキー因子分解  $\text{sub}(B) = U^T U$  または  $\text{sub}(B) = L L^T$  からの三角係数  $U$  または  $L$  で上書きされる。
- m** (グローバル)  
 INTEGER。検出された固有値の総数 ( $0 \leq m \leq n$ )。
- nz** (グローバル)  
 INTEGER。  
 計算された固有ベクトルの総数 ( $0 \leq nz \leq m$ )。  $z$  の列数。  
`jobz.ne. 'V'` の場合、`nz` は参照されない。  
`jobz.eq. 'V'` の場合、`nz = m` (ユーザが十分な空間を提供しないで、[p?sygvx](#) が計算を開始する前にこれを検出できない場合を除く)。  
 要求されたすべての固有ベクトルを得るには、ユーザは  $z(m.le. \text{descz}(n\_))$  の固有ベクトルを保持するための十分な空間とそれらを計算するための十分なワークスペースの両方を提供しなければならない。( `lwork` を参照 )。 [p?sygvx](#) は、`range.eq. 'V'` の場合を除いて、計算を開始する前に空間が十分でないことを常に検出できる。
- w** (グローバル)  
 REAL (`pssygvx` の場合)  
 DOUBLE PRECISION (`pdsygvx` の場合)。  
 配列、次元は  $(n)$ 。  
 正常終了時に、先頭の  $m$  個の成分には、指定された固有値が昇順で格納される。
- z** (ローカル)。  
 REAL (`pssygvx` の場合)  
 DOUBLE PRECISION (`pdsygvx` の場合)。  
 グローバル次元  $(n, n)$ 、ローカル次元  $(lld\_z, LOCc(jz+n-1))$ 。  
`jobz = 'V'` の場合、正常終了時に、 $z$  の先頭の  $m$  列には、選択された固有値に対応する正規直交固有ベクトルが格納される。固有ベクトルが収束に失敗した場合、 $z$  のその列には、その固有ベクトルに対する最

新の近似値が格納され、固有ベクトルのインデックスは *ifail* に戻される。

*jobz* = 'N' の場合、*z* は参照されない。

*work* *jobz* = 'N' の場合、*work*(1) には、固有値を効率的に計算するために必要な、ワークスペースの最適な量が格納される。  
*jobz* = 'V' の場合、*work*(1) には、直交性を保証しないで固有値と固有ベクトルを効率的に計算するために必要な、ワークスペースの最適な量が格納される。  
*range* = 'V' の場合、すべての固有ベクトルが必要であるとみなされる。

*ifail* (グローバル)  
 INTEGER。  
 配列、次元は (*n*)。  
*info.ne.0* の場合、*ifail* は追加情報を提供する。  
 (*mod(info/16,2).ne.0*) の場合、*ifail*(1) は、正定値でない最小の小行列式の次数を示す。終了時に、(*mod(info,2).ne.0*) の場合、*ifail* には収束に失敗した固有ベクトルのインデックスが格納される。

上記のエラー条件がどちらもセットされてなく、かつ *jobz* = 'V' の場合、*ifail* の先頭から *m* 個の成分はゼロに設定される。

*iclustr* (グローバル)  
 INTEGER。  
 配列、次元は (2\*NPROW\*NPCOL)。この配列には、ワークスペースが十分でなかったために再直交できなかった、固有値のクラスタに対する固有ベクトルのインデックスが格納される (*lwork*、*orfac* および *info* を参照)。インデックス *iclustr*(2\*i-1) から *iclustr*(2\*i) の固有値のクラスタに対応する固有ベクトルは、ワークスペースの不足により再直交できなかった。したがって、これらのクラスタに対応する固有ベクトルは直交しない。*iclustr*() は、ゼロで終了する配列である。  
 (*iclustr*(2\*k).ne.0.and. *iclustr*(2\*k+1).eq.0) (*k* がクラスタ数の場合)。*jobz* = 'N' の場合、*iclustr* は参照されない。

*gap* (グローバル)  
 REAL (*pssygvx* の場合)  
 DOUBLE PRECISION (*pdsygvx* の場合)。  
 配列、次元は (NPROW\*NPCOL)。  
 この配列には、固有ベクトルが再直交できなかった固有値間のギャップが格納される。この配列の出力値は、配列 *iclustr* によって示されたクラスタに対応する。その結果、*i* 番目のクラスタに対応する固有ベクトルの内積は、(*C* \* *n*) / *gap*(*i*) と同程度になる。ここで、*C* は小さな定数である。



*info* (グローバル)  
 INTEGER。  
*info* = 0 の場合、実行は正常に終了したことを示す。  
*info* < 0 の場合：  
*i* 番目の引数が配列で、*j* 番目の値が不正だった場合、  
*info* = -(*i*\*100+*j*)。 *i* 番目の引数がスカラで値が不正だった場合、  
*info* = -*i*。  
*info* > 0 の場合：  
 (mod(*info*,2).ne.0) の場合、1 つ以上の固有ベクトルは収束に失敗したことを示す。それらのインデックスは *ifail* に格納される。  
 (mod(*info*/2,2).ne.0) の場合、1 つ以上の固有値のクラスタに対する固有ベクトルが、ワークスペースが十分でなかったために再直交化できなかった。クラスタのインデックスは、配列 *iclustr* に格納される。  
 (mod(*info*/4,2).ne.0) の場合、空間の制限により、*p?sygvx* で *vl* と *vu* 間のすべての固有ベクトルを計算できなかった。計算された固有ベクトルの個数は、*nz* に返される。  
 (mod(*info*/8,2).ne.0) の場合、[p?stebz](#) は、固有値の計算に失敗したことを示す。  
 (mod(*info*/16,2).ne.0) の場合、*B* は正定値ではなかったことを示す。  
*ifail*(1) は、正定値でない最小の小行列式の次数を示す。

## p?hegvx

複素数の汎用エルミート固有値問題について、  
 固有値と (オプションで) 固有ベクトルを選択的に計算する。

### 構文

```
call pchegvx(ibtype, jobz, range, uplo, n, a, ia, ja, desca, b, ib, jb,
             descb, vl, vu, il, iu, abstol, m, nz, w, orfac, z, iz, jz, descz,
             work, lwork, rwork, lrwork, iwork, liwork, ifail, iclustr, gap, info)
call pzhegvx(ibtype, jobz, range, uplo, n, a, ia, ja, desca, b, ib, jb,
             descb, vl, vu, il, iu, abstol, m, nz, w, orfac, z, iz, jz, descz,
             work, lwork, rwork, lrwork, iwork, liwork, ifail, iclustr, gap, info)
```

## 説明

このルーチンは、次の形式で示される複素数の汎用エルミート固有値問題について、固有値と (オプションで) 固有ベクトルを選択的に計算する。

$$\text{sub}(A)x = \lambda \text{sub}(B)x, \text{sub}(A) \text{sub}(B)x = \lambda x, \text{または} \text{sub}(B) \text{sub}(A)x = \lambda x.$$

ここで、 $A(\text{ia}:\text{ia}+n-1, \text{ja}:\text{ja}+n-1)$  を表す  $\text{sub}(A)$  と  $\text{sub}(B)$  はエルミートとみなされ、 $B(\text{ib}:\text{ib}+n-1, \text{jb}:\text{jb}+n-1)$  を表す  $\text{sub}(B)$  もまた、正定値とみなされる。

## 入力パラメータ

<i>ibtype</i>	(グローバル) INTEGER。 1、2、または3のいずれかでなければならない。 解決する問題のタイプを指定する。 <i>ibtype</i> = 1 の場合、問題のタイプは $\text{sub}(A)x = \lambda \text{sub}(B)x$ である。 <i>ibtype</i> = 2 の場合、問題のタイプは $\text{sub}(A)\text{sub}(B)x = \lambda x$ である。 <i>ibtype</i> = 3 の場合、問題のタイプは $\text{sub}(B) \text{sub}(A)x = \lambda x$ である。
<i>jobz</i>	(グローバル) CHARACTER*1。 'N' または 'V' でなければならない。 <i>jobz</i> = 'N' の場合、固有値のみを計算する。 <i>jobz</i> = 'V' の場合、固有値と固有ベクトルを計算する。
<i>range</i>	(グローバル)。CHARACTER*1。 'A'、'V'、または 'I' でなければならない。 <i>range</i> = 'A' の場合、固有値をすべて計算する。 <i>range</i> = 'V' の場合、ルーチンは区間 $[\text{vl}, \text{vu}]$ の固有値を計算する。 <i>range</i> = 'I' の場合、ルーチンはインデックスが $\text{il} \sim \text{i u}$ の固有値を計算する。
<i>uplo</i>	(グローバル)。CHARACTER*1。 'U' または 'L' でなければならない。 <i>uplo</i> = 'U' の場合、配列 <i>a</i> と <i>b</i> は、 $\text{sub}(A)$ と $\text{sub}(B)$ の上三角を格納する。 <i>uplo</i> = 'L' の場合、配列 <i>a</i> と <i>b</i> は、 $\text{sub}(A)$ と $\text{sub}(B)$ の下三角を格納する。
<i>n</i>	(グローバル)。INTEGER。 行列 $\text{sub}(A)$ と $\text{sub}(B)$ の次数 ( $n \geq 0$ )。

<i>a</i>	<p>(ローカル)</p> <p>COMPLEX (pchegvx の場合)</p> <p>DOUBLE COMPLEX (pzhegvx の場合)。</p> <p>ローカルメモリにある、次元 (<i>lld_a</i>, <i>LOCc(ja+n-1)</i>) の配列へのポインタ。呼び出し時に、この配列は、<math>n \times n</math> のエルミート分散行列 <i>sub(A)</i> のローカル部分を含む。<i>uplo</i> = 'U' の場合、<i>sub(A)</i> の先頭の <math>n \times n</math> 上三角部分は行列の上三角部分を含む。<i>uplo</i> = 'L' の場合、<i>sub(A)</i> の先頭の <math>n \times n</math> 下三角部分は行列の下三角部分を含む。</p>
<i>ia, ja</i>	<p>(グローバル) INTEGER。</p> <p>それぞれ、部分行列 <i>A</i> の最初の行と最初の列を示す、グローバル配列 <i>a</i> の行インデックスと列インデックス。</p>
<i>desca</i>	<p>(グローバルおよびローカル) INTEGER。配列、次元は (<i>dlen_</i>)。</p> <p>分散行列 <i>A</i> の配列ディスクリプタ。<i>desca</i>(<i>ctxt_</i>) が正しくない場合、<a href="#">pzhegvx</a> での正しいエラー報告は保証されない。</p>
<i>b</i>	<p>(ローカル)。</p> <p>COMPLEX (pchegvx の場合)</p> <p>DOUBLE COMPLEX (pzhegvx の場合)。</p> <p>ローカルメモリにある、次元 (<i>lld_b</i>, <i>LOCc(jb+n-1)</i>) の配列へのポインタ。呼び出し時に、この配列は、<math>n \times n</math> のエルミート分散行列 <i>sub(B)</i> のローカル部分を含む。<i>uplo</i> = 'U' の場合、<i>sub(B)</i> の先頭の <math>n \times n</math> 上三角部分は行列の上三角部分を含む。<i>uplo</i> = 'L' の場合、<i>sub(B)</i> の先頭の <math>n \times n</math> 下三角部分は行列の下三角部分を含む。</p>
<i>ib, jb</i>	<p>(グローバル) INTEGER。</p> <p>それぞれ、部分行列 <i>B</i> の最初の行と最初の列を示す、グローバル配列 <i>b</i> の行インデックスと列インデックス。</p>
<i>descb</i>	<p>(グローバルおよびローカル) INTEGER。配列、次元は (<i>dlen_</i>)。</p> <p>分散行列 <i>B</i> の配列ディスクリプタ。<i>descb</i>(<i>ctxt_</i>) は <i>desca</i>(<i>ctxt_</i>) と等しくなければならない。</p>
<i>v1, vu</i>	<p>(グローバル)</p> <p>REAL (pchegvx の場合)</p> <p>DOUBLE PRECISION (pzhegvx の場合)。</p> <p><i>range</i> = 'V' の場合、固有値を探す区間の下限値と上限値。</p> <p><i>range</i> = 'A' または 'I' の場合、<i>v1</i> と <i>vu</i> は参照されない。</p>

<i>il, iu</i>	<p>(グローバル)</p> <p>INTEGER。</p> <p><i>range</i>='I' の場合、返される最小固有値と最大固有値に対する昇順のインデックス。</p> <p>次の制約がある。<math>il \geq 1, \min(il, n) \leq iu \leq n</math></p> <p><i>range</i>='A' または 'V' の場合、<i>il</i> と <i>iu</i> は参照されない。</p>
<i>abstol</i>	<p>(グローバル)</p> <p>REAL (pchegvx の場合)</p> <p>DOUBLE PRECISION (pzhegvx の場合)。</p> <p><i>jobz</i>='V' の場合、<i>abstol</i> を <code>p?lamch(context, 'U')</code> に設定すると、ほとんどの直交固有ベクトルが生成される。</p> <p>固有値に対する絶対エラー許容値。次に示す値以下の幅の区間 <math>[a, b]</math> 内に存在していると判別された場合 (<i>eps</i> はマシン精度)、近似固有値は、収束したものとして受け入れられる。</p> <p><math>abstol + eps * \max( a ,  b )</math></p> <p>ここで、<i>eps</i> はマシン精度である。<i>abstol</i> がゼロまたは負の場合、代わりに <math>eps * \text{norm}(T)</math> を使用する。ここで、<math>\text{norm}(T)</math> は、<i>A</i> を三重対角形式に縮退して得られた三重対角行列の 1- ノルムである。</p> <p>固有値が最も正確に計算されるのは、<i>abstol</i> がゼロでなく、アンダーフローのしきい値の 2 倍、<math>2 * p?lamch('S')</math> に設定されたときである。</p> <p>このルーチンで <math>((\text{mod}(\text{info}, 2).ne.0).or. * (\text{mod}(\text{info}/8, 2).ne.0))</math> が返された場合、固有値または固有ベクトルのいくつか収束に失敗したことを意味するので、<i>abstol</i> を <math>2 * p?lamch('S')</math> に設定して、やり直してみる。</p>
<i>orfac</i>	<p>(グローバル)。</p> <p>REAL (pchegvx の場合)</p> <p>DOUBLE PRECISION (pzhegvx の場合)。</p> <p>再直交化される固有ベクトルを指定する。互いの <math>\text{tol} = \text{orfac} * \text{norm}(A)</math> 内にある固有値に対応する固有ベクトルは再直交化される。しかし、ワークスペースが十分でない場合 (<i>lwork</i> を参照)、<i>tol</i> は、すべての固有ベクトルが 1 つのプロセスで再直交化されるようになるまで下げられる。<i>orfac</i> がゼロの場合、再直交化は行われない。<i>orfac</i> が負の場合、デフォルト値の <math>10^{-3}</math> が使用される。<i>orfac</i> はすべての処理で同一でなければならない。</p>
<i>iz, jz</i>	<p>(グローバル) INTEGER。</p> <p>それぞれ、部分行列 <i>Z</i> の最初の行と最初の列を示す、グローバル配列 <i>z</i> の行インデックスと列インデックス。</p>

*descz* (グローバルおよびローカル) INTEGER。配列、次元は (*dlen\_*)。  
分散配列 *Z* のディスクリプタ。*descz*(*ctxt\_*) は *desca*(*ctxt\_*) と等しくなければならない。

*work* (ローカル)  
COMPLEX (*pchegvx* の場合)  
DOUBLE COMPLEX (*pzhegvx* の場合)。  
ワークスペース配列、次元は (*lwork*)。

*lwork* (ローカル)。  
INTEGER。配列 *work* の次元。  
固有値のみが要求された場合:  
 $lwork \geq n + \max(NB * (np0 + 1), 3)$   
固有ベクトルが要求された場合:  
 $lwork \geq n + (np0 + nq0 + NB) * NB$   
 $nq0 = \text{numroc}(nn, NB, 0, 0, NPCOL)$

最適なパフォーマンスを得るには、より大きなワークスペースが必要である。すなわち、

$lwork \geq \max(lwork, n, nhetr\_lwork, nhegst\_lwork)$

ここで、*lwork* は上で定義され、

$nhetr\_lwork = 2 * (anb + 1) * (4 * nps + 2) + (nps + 1) * nps$

$nhegst\_lwork = 2 * np0 * NB + nq0 * NB + NB * NB$

$NB = \text{desca}(mb\_)$

$np0 = \text{numroc}(n, NB, 0, 0, NPROW)$

$nq0 = \text{numroc}(n, NB, 0, 0, NPCOL)$

$ictxt = \text{desca}(ctxt\_)$

$anb = \text{pjlaenv}(ictxt, 3, 'p?hettrd', 'L', 0, 0, 0, 0)$

$sqnpc = \text{sqrt}(\text{dble}(NPROW * NPCOL))$

$nps = \max(\text{numroc}(n, 1, 0, 0, sqnpc), 2 * anb)$

*numroc* は ScaLAPACK ツール関数である。

*pjlaenv* は ScaLAPACK 環境問い合わせ関数である。

*MYROW*、*MYCOL*、*NPROW*、および *NPCOL* は、サブルーチン

*blacs\_gridinfo* を呼び出して決定できる。

*lwork* = -1 の場合、*lwork* はグローバル入力であり、ワークスペースのクエリとみなされ、ルーチンはすべての *work* 配列で最適なパフォーマンスのために必要なサイズだけを計算する。各値は該当する *work* 配列の最初のエントリとして返され、[pxerbla](#) はエラー・メッセージを生成しない。

*rwork* (ローカル)  
 REAL (pchegvx の場合)  
 DOUBLE PRECISION (pzhegvx の場合)。  
 ワークスペース配列、次元は (*lrwork*)。

*lrwork* (ローカル)  
 INTEGER。配列 *rwork* の次元。  
*lrwork* の定義に使用する変数の定義は以下を参照。  
 固有ベクトルが要求されなかった場合 (*jobz* = 'N')、  
 $lrwork \geq 5 * nn + 4 * n$ 。固有ベクトルが要求された場合 (*jobz* = 'V')、  
 すべての固有ベクトルが計算されることを保証するために必要なワークスペースの量は次のとおりである。

$$lrwork \geq 4 * n + \max(5 * nn, np0 * mq0) + \text{iceil}(neig, NPROW * NPCOL) * nn$$

最小のワークスペースが与えられ、*orfac* が小さすぎる場合、計算されるベクトル固有値は直交ではない。(パフォーマンスが低下しても) 直交性を保証するには、次の値を *lrwork* に追加する。  
 $(clustersize - 1) * n$   
 ここで、*clustersize* は、クラスタが近い固有値のセットとして定義された、最大クラスタの固有値の数。  
 $\{w(k), \dots, w(k + clustersize - 1)\}$   
 $w(j + 1) \leq w(j) + orfac * 2 * \text{norm}(A)$

変数定義:  
*neig* = 要求された固有ベクトルの数  
 $NB = \text{desca}(mb\_)=\text{desca}(nb\_)=\text{descz}(mb\_)=\text{descz}(nb\_)$   
 $nn = \max(n, NB, 2)$   
 $\text{desca}(rsrc\_)=\text{desca}(nb\_)=\text{descz}(rsrc\_)=\text{descz}(csrc\_)=0$   
 $np0 = \text{numroc}(nn, NB, 0, 0, NPROW)$   
 $mq0 = \text{numroc}(\max(neig, NB, 2), NB, 0, 0, NPCOL)$  *iceil*(*x*, *y*) は *ceiling*(*x*/*y*) を返す ScaLAPACK 関数。

*lrwork* が小さすぎる場合:  
*lwork* が直交性を保証するには小さすぎる場合、p?hegvx は、固有値間の間隔が最も小さいクラスタで直交性を維持するように試みる。  
*lwork* が小さすぎて要求された固有値をすべて計算できない場合、計算は行われず、*info*=-25 が返される。*range*='V' の場合、p?hegvx は固有値が計算されるまで要求されている固有ベクトルがわからない点に注意する。したがって、*range*='V' で *lwork* が十分大きく p?hegvx で固有値が計算可能な場合、p?hegvx は固有値とできるだけ多くの固有ベクトルを計算する。

ワークスペース、直交性とパフォーマンスの関係:

$clustersize \geq n/\sqrt{NPROW*NPCOL}$  の場合、すべての固有ベクトルを直交して計算するために十分なスペースを提供すると、パフォーマンスが大幅に低下する。限界 (すなわち、 $clustersize = n-1$ ) では、[p?stein](#) は 1 プロセッサ上の ?stein よりもパフォーマンスが高くなることはない。

$clustersize = n/\sqrt{NPROW*NPCOL}$  の場合、すべての固有ベクトルの再直交化は、2 以上の係数では合計の実行時間が長くなる。

$clustersize > n/\sqrt{NPROW*NPCOL}$  の場合、実行時間はクラスタサイズの 2 乗で増加し、すべての他の係数は同等なままで十分なワークスペースが提供されていると仮定する。ワークスペースが少なくなると再直交化も少なくなるが、実行は速くなる。

$lwork = -1$  の場合、 $lwork$  はグローバル入力であり、ワークスペースのクエリとみなされ、ルーチンはすべての  $work$  配列で最適なパフォーマンスのために必要なサイズだけを計算する。各値は該当する  $work$  配列の最初のエントリとして返され、[pxerbla](#) はエラー・メッセージを生成しない。

$iwork$  (ローカル) INTEGER。ワークスペース配列。

$liwork$  (ローカル) INTEGER。 $iwork$  の次元。

$liwork \geq 6 * nnp$

ここで、 $nnp = \max(n, NPROW*NPCOL + 1, 4)$

$liwork = -1$  の場合、 $liwork$  はグローバル入力であり、ワークスペースのクエリとみなされ、ルーチンはすべての  $work$  配列の最小かつ最適なサイズだけを計算する。各値は該当する  $work$  配列の最初のエントリとして返され、[pxerbla](#) はエラー・メッセージを生成しない。

## 出力パラメータ

- a 終了時に、 $jobz = 'v'$  で  $info = 0$  の場合、 $sub(A)$  には、固有ベクトルの分散行列  $Z$  が格納される。  
固有ベクトルは、次のように正規化される。  
 $ibtype = 1$  または  $2$  の場合  
 $Z^H * sub(B) * Z = I$ ;  
 $ibtype = 3$  の場合、 $Z^H * inv(sub(B)) * Z = I$ 。  
 $jobz = 'N'$  の場合、終了時に、 $sub(A)$  の上三角 ( $uplo = 'U'$  の場合) または下三角 ( $uplo = 'L'$  の場合) が対角を含めて破棄される。
- b 終了時に、 $info \leq n$  の場合、行列を格納している  $sub(B)$  の一部が、コレスキー因子分解  $sub(B) = U^H U$  または  $sub(B) = L L^H$  からの三角係数  $U$  あるいは  $L$  で上書きされる。

<i>m</i>	(グローバル) INTEGER。検出された固有値の総数 ( $0 \leq m \leq n$ )。
<i>nz</i>	(グローバル) INTEGER。 計算された固有ベクトルの総数 ( $0 \leq nz \leq m$ )。 <i>z</i> の列数。 <i>jobz</i> .ne. 'V' の場合、 <i>nz</i> は参照されない。 <i>jobz</i> .eq. 'V' の場合、 <i>nz</i> = <i>m</i> ( ユーザが十分な空間を提供しないで、 <a href="#">p?hegvx</a> が計算を開始する前にこれを検出できない場合を除く )。 要求されたすべての固有ベクトルを得るには、ユーザは <i>z</i> ( <i>m</i> .le. <i>descz</i> ( <i>n</i> )) の固有ベクトルを保持するための十分な空間とそれらを計算するための十分なワークスペースの両方を提供しなければならない。( <i>lwork</i> を参照 )。 <i>p?hegvx</i> は、 <i>range</i> .eq. 'V' の場合を除いて、計算を開始する前に空間が十分でないことを常に検出できる。
<i>w</i>	(グローバル) REAL ( <i>pchegvx</i> の場合 ) DOUBLE PRECISION ( <i>pzhegvx</i> の場合 )。 配列、次元は ( <i>n</i> )。 正常終了時に、先頭の <i>m</i> 個の成分には、指定された固有値が昇順で格納される。
<i>z</i>	(ローカル)。 COMPLEX ( <i>pchegvx</i> の場合 ) DOUBLE COMPLEX ( <i>pzhegvx</i> の場合 )。 グローバル次元 ( <i>n</i> , <i>n</i> )、ローカル次元 ( <i>lld_z</i> , <i>LOCc</i> ( <i>jz</i> + <i>n</i> -1))。 <i>jobz</i> = 'V' の場合、正常終了時に、 <i>z</i> の先頭の <i>m</i> 列には、選択された固有値に対応する正規直交固有ベクトルが格納される。固有ベクトルが収束に失敗した場合、 <i>z</i> のその列には、その固有ベクトルに対する最新の近似値が格納され、固有ベクトルのインデックスは <i>ifail</i> に戻される。 <i>jobz</i> = 'N' の場合、 <i>z</i> は参照されない。
<i>work</i>	終了時に、 <i>work</i> (1) は、ワークスペースの最適値を返す。
<i>rwork</i>	終了時に、 <i>rwork</i> (1) には、効率的に実行するために必要なワークスペースの最適な量が格納される。 <i>jobz</i> ='N' の場合、 <i>rwork</i> (1) には、固有値を効率的に計算するために必要な、ワークスペースの最適な量が格納される。 <i>jobz</i> ='V' の場合、 <i>rwork</i> (1) には、直交性を保証しないで固有値と固有ベクトルを効率的に計算するために必要な、ワークスペースの最適な



量が格納される。

*range='V'* の場合、最適なワークスペースを計算するときに、すべての固有ベクトルが必要であるとみなされる。

*ifail*

(グローバル)

INTEGER。

配列、次元は (n)。

*info.ne.0* の場合、*ifail* は追加情報を提供する。

(*mod(info/16,2).ne.0*) の場合、*ifail(1)* は、正定値でない最小の小行列式の次数を示す。終了時に、(*mod(info,2).ne.0*) の場合、*ifail(1)* には収束に失敗した固有ベクトルのインデックスが格納される。

上記のエラー条件がどちらもセットされてなく、かつ *jobz='V'* の場合、*ifail* の先頭から *m* 個の成分はゼロに設定される。

*iclustr*

(グローバル)

INTEGER。

配列、次元は (2\*NPROW\*NPCOL)。この配列には、ワークスペースが十分でなかったために再直交できなかった、固有値のクラスタに対する固有ベクトルのインデックスが格納される (*lwork*、*orfac*、および *info* を参照)。インデックス *iclustr(2\*i-1)* から *iclustr(2\*i)* の固有値のクラスタに対応する固有ベクトルは、ワークスペースの不足により再直交できなかった。したがって、これらのクラスタに対応する固有ベクトルは直交しない。*iclustr()* は、ゼロで終了する配列である。( *iclustr(2\*k).ne.0.and. iclustr(2\*k+1).eq.0* ) ( *k* がクラスタ数の場合 )。 *jobz='N'* の場合、*iclustr* は参照されない。

*gap*

(グローバル)

REAL (*pchegvx* の場合 )

DOUBLE PRECISION (*pzhgevx* の場合 )。

配列、次元は (NPROW\*NPCOL)。

この配列には、固有ベクトルが再直交できなかった固有値間のギャップが格納される。この配列の出力値は、配列 *iclustr* によって示されたクラスタに対応する。その結果、*i* 番目のクラスタに対応する固有ベクトルの内積は、(*C \* n*) / *gap(i)* と同程度になる。ここで、*C* は小さな定数である。

*info*

(グローバル)

INTEGER。

*info=0* の場合、実行は正常に終了したことを示す。

*info<0* の場合：

$i$  番目の引数が配列で、 $j$  番目の値が不正だった場合、  
 $info = -(i*100+j)$ 。 $i$  番目の引数がスカラーで値が不正だった場合、  
 $info = -i$ 。

$info > 0$  の場合：

$(\text{mod}(info,2), \text{ne.}0)$  の場合、1 つ以上の固有ベクトルは収束に失敗したことを示す。それらのインデックスは *ifail* に格納される。

$(\text{mod}(info/2,2), \text{ne.}0)$  の場合、1 つ以上の固有値のクラスタに対する固有ベクトルが、ワークスペースが十分でなかったために再直交化できなかった。クラスタのインデックスは、配列 *iclustr* に格納される。

$(\text{mod}(info/4,2), \text{ne.}0)$  の場合、空間の制限により、*p?sygvx* で *v1* と *vu* 間のすべての固有ベクトルを計算できなかった。計算された固有ベクトルの個数は、*nz* に返される。

$(\text{mod}(info/8,2), \text{ne.}0)$  の場合、[p?stebz](#) は、固有値の計算に失敗したことを示す。

$(\text{mod}(info/16,2), \text{ne.}0)$  の場合、*B* は正定値ではなかったことを示す。  
*ifail(1)* は、正定値でない最小の小行列式の次数を示す。

# ScaLAPACK 補助ルーチンとユーティリティ・ルーチン

## 7

この章では ScaLAPACK [補助ルーチン](#)および[ユーティリティ関数とルーチン](#)に関するインテル® マス・カーネル・ライブラリの実装について説明する。ライブラリは実数と複素数の両方のデータに対応したルーチンで構成される。



**注** :ScaLAPACK ルーチンは、インテル MKL のスーパーセットであるインテル® クラスタ MKL で提供されている。

ScaLAPACK 補助ルーチンとユーティリティ・ルーチンで使用されているルーチン名の規則、数学表記、行列の格納形式は、これまでの章に記載されているものと同一である。一部のルーチンや関数では、sc や dz などの組み合わされたキャラクタ・コードを持つことができる。例えば、ルーチン pscsum1 は、入力として複素数配列を使用し、出力として実数値を返す。

## 補助ルーチン

表 7-1 ScaLAPACK 補助ルーチン

ルーチン名	データ型	説明
<a href="#">p?lacgv</a>	C, Z	複素ベクトルを共役する。
<a href="#">p?max1</a>	C, Z	実数部分が最大絶対値を持つ成分のインデックスを検出する (レベル 1 PBLAS の p?amax に似ているが、実数部分の絶対値を使用している)。
<a href="#">?combamax1</a>	C, Z	実数部分が最大絶対値を持つ成分とそのグローバル・インデックスを検出する。

表 7-1 ScaLAPACK 補助ルーチン ( 続き )

ルーチン名	データ型	説明
<a href="#">p?sum1</a>	s,c,d,z	レベル 1 PBLAS の p?asum と同様に複素ベクトルの 1- ノルムを実行するが、真の絶対値を用いる。
<a href="#">p?dbtrsv</a>	s,d,c,z	一般三重対角行列の LU 因子分解を、ピボット演算を用いないで計算する。p?dbtrs によって呼び出される。
<a href="#">p?dttrsv</a>	s,d,c,z	一般帯行列の LU 因子分解を、行交換を伴う部分ピボット演算を用いて計算する。p?dttrs によって呼び出される。
<a href="#">p?gebd2</a>	s,d,c,z	直交 / ユニタリ変換を用いて、一般矩形行列を実数二重対角形式に縮退させる ( 非ブロック化アルゴリズム )。
<a href="#">p?gehd2</a>	s,d,c,z	直交 / ユニタリ相似変換を用いて、一般行列を上 Hessenberg 形式に縮退させる ( 非ブロック化アルゴリズム )。
<a href="#">p?gelq2</a>	s,d,c,z	一般矩形行列の LQ 因子分解を計算する ( 非ブロック化アルゴリズム )。
<a href="#">p?geql2</a>	s,d,c,z	一般矩形行列の QL 因子分解を計算する ( 非ブロック化アルゴリズム )。
<a href="#">p?geqr2</a>	s,d,c,z	一般矩形行列の QR 因子分解を計算する ( 非ブロック化アルゴリズム )。
<a href="#">p?gerq2</a>	s,d,c,z	一般矩形行列の RQ 因子分解を計算する ( 非ブロック化アルゴリズム )。
<a href="#">p?getf2</a>	s,d,c,z	一般行列の LU 因子分解を、行交換を伴う部分ピボット演算を用いて計算する ( ローカルブロック化アルゴリズム )。
<a href="#">p?labrd</a>	s,d,c,z	直交 / ユニタリ変換を用いて、一般矩形行列 A の最初の nb 行と列を実数二重対角形式に縮退させ、A の縮退されていない部分に変換を適用するために必要となる補助行列を返す。
<a href="#">p?lacon</a>	s,d,c,z	行列 - ベクトル積の評価にリバース・コミュニケーションを使用して正方行列の 1- ノルムを推定する。
<a href="#">p?laconsb</a>	s,d	2 つの連続する小さい対角成分を検出する。
<a href="#">p?lacp2</a>	s,d,c,z	分散行列の一部または全部を他の分散行列にコピーする。
<a href="#">p?lacp3</a>	s,d	グローバル並列配列からローカル複製配列にコピーする。逆の場合も同じ。
<a href="#">p?lacpy</a>	s,d,c,z	1 つの 2 次元配列の一部または全部を他にコピーする。
<a href="#">p?laevswp</a>	s,d,c,z	計算された固有ベクトルを ScaLAPACK 標準ブロックサイクル配列に移動する。
<a href="#">p?lahrd</a>	s,d,c,z	直交 / ユニタリ相似変換を用いて、k 番目の劣対角よりも下の成分がゼロになるように一般矩形行列の最初の nb 列を縮退させ、A の縮退されていない部分に変換を適用するために必要となる補助行列を返す。

表 7-1 ScaLAPACK 補助ルーチン ( 続き )

ルーチン名	データ型	説明
<a href="#">p?lalect</a>	s, d, c, z	IEEE 演算を利用して固有値の計算を加速させる (C インターフェイス関数)。
<a href="#">p?lange</a>	s, d, c, z	一般矩形行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。
<a href="#">p?lanhs</a>	s, d, c, z	上 Hessenberg 行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。
<a href="#">p?lansy,</a> <a href="#">p?lanhe</a>	s, d, c, z / c, z	実対称行列または複素エルミート行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。
<a href="#">p?lantr</a>	s, d, c, z	三角行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。
<a href="#">p?lapiv</a>	s, d, c, z	一般分散行列に置換行列を適用し、行または列のピボット演算を行う。
<a href="#">p?lagge</a>	s, d, c, z	p?geequ で計算された行と列のスケール係数を使って一般矩形行列をスケールリングする。
<a href="#">p?lagsy</a>	s, d, c, z	p?poequ で計算されたスケール係数を用いて対称/エルミート行列をスケールリングする。
<a href="#">p?lared1d</a>	s, d	入力配列 <i>bycol</i> は複数の行に分配され、すべてのプロセス列に同じ <i>bycol</i> のコピーが格納されることを前提に、配列を再分配する。
<a href="#">p?lared2d</a>	s, d	入力配列 <i>byrow</i> は複数の列に分配され、すべてのプロセス行に同じ <i>byrow</i> のコピーが格納されることを前提に、配列を再分配する。
<a href="#">p?larf</a>	s, d, c, z	一般矩形行列に基本リフレクタを適用する。
<a href="#">p?larfb</a>	s, d, c, z	ブロック・リフレクタまたはその転置 / 共役転置を一般矩形行列に適用する。
<a href="#">p?larfc</a>	c, z	一般行列に基本リフレクタの共役転置を適用する。
<a href="#">p?larfq</a>	s, d, c, z	基本リフレクタ (Householder 行列) を生成する。
<a href="#">p?larft</a>	s, d, c, z	ブロック・リフレクタ $H = I - VTV^H$ の三角ベクトル $T$ を生成する。
<a href="#">p?larz</a>	s, d, c, z	p?tzrzf が返したとおりの基本リフレクタを一般行列に適用する。
<a href="#">p?larzb</a>	s, d, c, z	p?tzrzf が返したとおりのブロック・リフレクタまたはその転置 / 共役転置を、一般行列に適用する。
<a href="#">p?larzc</a>	c, z	p?tzrzf が返したとおりの基本リフレクタの共役転置を、一般行列に適用 (乗算) する。

表 7-1 ScaLAPACK 補助ルーチン ( 続き )

ルーチン名	データ型	説明
<a href="#">p?larzt</a>	s,d,c,z	p?tzzrzf が返したとおりのブロック・リフレクタ $H=I VTV^H$ の三角係数 $T$ を生成する。
<a href="#">p?lascl</a>	s,d,c,z	一般矩形行列に $C_{to}/C_{from}$ として定義される実スカラを掛ける。
<a href="#">p?laset</a>	s,d,c,z	行列の非対角成分を $\alpha$ に、対角成分を $\beta$ に初期化する。
<a href="#">p?lasmsub</a>	s,d	安全に 0 に設定できる小さい劣対角成分を行列の一番下から検出する。
<a href="#">p?lassq</a>	s,d,c,z	スケーリング形式で表現された二乗和を更新する。
<a href="#">p?laswp</a>	s,d,c,z	一般矩形行列に対して一連の行交換を実行する。
<a href="#">p?latra</a>	s,d,c,z	一般正分散行列の対角和を計算する。
<a href="#">p?latrd</a>	s,d,c,z	直交 / ユニタリ相似変換を用いて、対称 / エルミート行列 $A$ の最初の $nb$ 行 / 列を実数三重対角形式に縮退する。
<a href="#">p?latrz</a>	s,d,c,z	直交 / ユニタリ相似変換を用いて、実 / 複素上台形行列を上三角形式に縮退させる。
<a href="#">p?lauu2</a>	s,d,c,z	積 $UU^H$ または $L^H L$ を計算する。ここで、 $U$ と $L$ は上三角または下三角行列 ( ローカル非ブロック化アルゴリズム )。
<a href="#">p?lauum</a>	s,d,c,z	積 $UU^H$ または $L^H L$ を計算する。ここで、 $U$ と $L$ は上三角または下三角行列。
<a href="#">p?lawil</a>	s,d	Wilkinson 変換を実行する。
<a href="#">p?org2l/p?ung2l</a>	s,d,c,z	p?geqlf で求めた $QL$ 因子分解から、全部または一部の直交 / ユニタリ行列 $Q$ を生成する ( 非ブロック化アルゴリズム )。
<a href="#">p?org2r/p?ung2r</a>	s,d,c,z	p?geqrf で求めた $QR$ 因子分解から、全部または一部の直交 / ユニタリ行列 $Q$ を生成する ( 非ブロック化アルゴリズム )。
<a href="#">p?orgl2/p?ungl2</a>	s,d,c,z	p?gelqf で求めた $LQ$ 因子分解から、全部または一部の直交 / ユニタリ行列 $Q$ を生成する ( 非ブロック化アルゴリズム )。
<a href="#">p?org2/p?ungr2</a>	s,d,c,z	p?gerqf で求めた $RQ$ 因子分解から、全部または一部の直交 / ユニタリ行列 $Q$ を生成する ( 非ブロック化アルゴリズム )。
<a href="#">p?orm2l/p?unm2l</a>	s,d,c,z	一般行列に p?geqlf で求めた $QL$ 因子分解の直交 / ユニタリ行列を掛ける ( 非ブロック化アルゴリズム )。
<a href="#">p?orm2r/p?unm2r</a>	s,d,c,z	一般行列に p?geqrf で求めた $QR$ 因子分解の直交 / ユニタリ行列を掛ける ( 非ブロック化アルゴリズム )。
<a href="#">p?orml2/p?unml2</a>	s,d,c,z	一般行列に p?gelqf で求めた $LQ$ 因子分解の直交 / ユニタリ行列を掛ける ( 非ブロック化アルゴリズム )。
<a href="#">p?ormr2/p?unmr2</a>	s,d,c,z	一般行列に p?gerqf で求めた $RQ$ 因子分解の直交 / ユニタリ行列を掛ける ( 非ブロック化アルゴリズム )。

表 7-1 ScaLAPACK 補助ルーチン ( 続き )

ルーチン名	データ型	説明
<a href="#">p?pbtrsv</a>	s,d,c,z	前進解法または後退解法によって、単一の三角行列線形問題を解く。ここで、三角行列は p?pbtrf で計算された帯行列の係数である。
<a href="#">p?pttrsv</a>	s,d,c,z	前進解法または後退解法によって、単一の三角行列線形問題を解く。ここで、三角行列は p?pttrf で計算された三重対角行列の係数である。
<a href="#">p?potf2</a>	s,d,c,z	対称 / エルミート正定値行列のコレスキー因子分解を行う ( ローカル非ブロック化アルゴリズム ) 。
<a href="#">p?rscl</a>	s,d,cs, zd	ベクトルに実スカラの逆数を掛ける。
<a href="#">p?syqs2/p?heqs2</a>	s,d,c,z	p?potrf で得られた因子分解の結果を用いて、対称 / エルミート汎用固有値問題を標準形式に縮退させる ( ローカル非ブロック化アルゴリズム ) 。
<a href="#">p?sytd2/p?hetd2</a>	s,d,c,z	直交 / ユニタリ相似変換を用いて、対称 / エルミート行列を実数対称三重対角形式に縮退させる ( ローカル非ブロック化アルゴリズム ) 。
<a href="#">p?trti2</a>	s,d,c,z	三角行列の逆行列を計算する ( ローカル非ブロック化アルゴリズム ) 。
<a href="#">?lamsh</a>	s,d	送られるバルジの数を最大にするために、小さな ( 単一成分 ) 行列を介して複数のシフトを送る。
<a href="#">?laref</a>	s,d	Householder リフレクタを行列の行または列に適用する。
<a href="#">?lasorte</a>	s,d	固有ペアを実数および複素数データ型でソートする。
<a href="#">?lasrt2</a>	s,d	数を昇順または降順でソートする。
<a href="#">?stein2</a>	s,d	逆反復法を用いて、実対称の三重対角行列の指定された固有値に対応する固有ベクトルを計算する。
<a href="#">?dbtf2</a>	s,d,c,z	一般帯行列の $LU$ 因子分解を、ピボット演算を用いなくて計算する ( ローカル非ブロック化アルゴリズム ) 。
<a href="#">?dbtrf</a>	s,d,c,z	一般帯行列の $LU$ 因子分解を、ピボット演算を用いなくて計算する ( ローカルブロック化アルゴリズム ) 。
<a href="#">?dttrf</a>	s,d,c,z	一般三重対角行列の $LU$ 因子分解を、ピボット演算を用いなくて計算する ( ローカルブロック化アルゴリズム ) 。
<a href="#">?dttrsv</a>	s,d,c,z	?dttrf によって行われた $LU$ 因子分解を使用して、一般三重対角行列を係数行列とする連立 1 次方程式を解く。
<a href="#">?pttrsv</a>	s,d,c,z	?pttrf によって行われた $LDL^H$ 因子分解を使用して、対称 ( エルミート ) 正定値三重対角行列を係数行列とする連立 1 次方程式を解く。

表 7-1 ScaLAPACK 補助ルーチン ( 続き )

ルーチン名	データ型	説明
<a href="#">?stegr2</a>	s, d	暗黙的 QL 法または QR 法を使用して、対称三重行列の固有値をすべて計算し、さらにオプションで、固有ベクトルも計算する。

## p?lacgv

複素ベクトルを共役する。

### 構文

```
call pclacgv(n, x, ix, jx, descx, incx)
call pzlacgv(n, x, ix, jx, descx, incx)
```

### 説明

このルーチンは、長さ  $n$ 、 $\text{sub}(x)$  の複素ベクトルを共役する。ここで、 $\text{sub}(x)$  は  $X(ix, jx:jx+n-1)$  ( $incx = \text{descx}(m\_)$  の場合)、または  $X(ix:ix+n-1, jx)$  ( $incx = 1$  の場合) である。

### 入力パラメータ

$n$  (グローバル) INTEGER。乱数ベクトル  $\text{sub}(x)$  の長さ。

$x$  (ローカル)  
 COMPLEX (pclacgv の場合)  
 COMPLEX\*16 (pzlacgv の場合)。  
 ローカルメモリにある、次元 ( $lld\_x, *$ ) の配列へのポインタ。共役するベクトルを格納する。 $x(i) = X(ix+(jx-1)*m\_x + (i-1)*incx)$ 、 $1 \leq i \leq n$ 。

$ix$  (グローバル) INTEGER。 $\text{sub}(x)$  の最初の行を示す、グローバル配列  $x$  の行インデックス。

$jx$  (グローバル) INTEGER。 $\text{sub}(x)$  の最初の列を示す、グローバル配列  $x$  の列インデックス。

$descx$  (グローバルおよびローカル) INTEGER。  
 配列、次元は ( $dlen\_$ )。分散行列  $X$  の配列ディスクリプタ。



*incx* (グローバル) INTEGER。X の成分のグローバルな増分。このバージョンでは、1 と *m\_x* の 2 つの *incx* の値のみサポートされる。*incx* の値は、ゼロであってはならない。

### 出力パラメータ

*x* (ローカル) 終了時に、共役ベクトルが格納される。

## p?max1

実数部分が最大絶対値を持つ成分のインデックスを検出する ( レベル 1 PBLAS の p?amax に似ているが、実数部分の絶対値を使用している )。

### 構文

```
call pmax1(n, amax, indx, x, ix, jx, descx, incx)
call pzmax1(n, amax, indx, x, ix, jx, descx, incx)
```

### 説明

このルーチンは、乱数ベクトル *sub(x)* の最大絶対値を持つ成分のグローバル・インデックスを計算する。グローバル・インデックスは *indx* に戻され、値は *amax* に戻される。ここで、*sub(x)* は次のとおりである。

$X(ix:ix+n-1, jx)$  (*incx* = 1 の場合)  
 $X(ix, jx:jx+n-1)$  (*incx* = *m\_x* の場合)

### 入力パラメータ

*n* (グローバル) INTEGER へのポインタ。  
 乱数ベクトル *sub(x)* の成分の個数。  $n \geq 0$ 。

*x* (ローカル)  
 COMPLEX (pmax1 の場合)  
 COMPLEX\*16 (pzmax1 の場合)。  
 次元が  $(jx-1)*m_x + ix + (n-1)*abs(incx)$  以上の分散行列のローカル部分を格納する配列。乱数ベクトル *sub(x)* の成分を格納する。

*ix* (グローバル) INTEGER。 *sub(x)* の最初の行を示す、グローバル配列 X の行インデックス。

<i>jx</i>	(グローバル) INTEGER。sub( <i>x</i> ) の最初の列を示す、グローバル配列 <i>X</i> の列インデックス。
<i>descx</i>	(グローバルおよびローカル) INTEGER。 配列、次元は( <i>dlen_</i> )。分散行列 <i>X</i> の配列ディスクリプタ。
<i>incx</i>	(グローバル) INTEGER。 <i>X</i> の成分のグローバルな増分。このバージョンでは、1 と <i>m_x</i> の 2 つの <i>incx</i> の値のみサポートされる。 <i>incx</i> の値は、ゼロであってはならない。

### 出力パラメータ

<i>amax</i>	(グローバル出力) REAL へのポインタ。sub( <i>x</i> ) の範囲における、乱数ベクトル sub( <i>x</i> ) の最大成分の絶対値。
<i>indx</i>	(グローバル出力) INTEGER へのポインタ。実数部分が最大絶対値を持つ、乱数ベクトル sub( <i>x</i> ) の成分のグローバル・インデックス。

---

## ?combamax1

実数部分が最大絶対値を持つ成分とそのグローバル・インデックスを検出する。

---

### 構文

```
call ccombamax1(v1, v2)
call zcombamax1(v1, v2)
```

### 説明

このルーチンは、実数部分が最大絶対値を持つ成分とそのグローバル・インデックスを検出する。

### 入力パラメータ

*v1* (ローカル)  
COMPLEX (ccombamax1 の場合)  
COMPLEX\*16 (zcombamax1 の場合)。  
配列、次元は 2。  
最初の絶対最大成分とそのグローバル・インデックス。  
*v1*(1) = *amax*、*v1*(2) = *indx*。

`v2` (ローカル)  
 COMPLEX (ccombamax1 の場合)  
 COMPLEX\*16 (zcombamax1 の場合)。  
 配列、次元は 2。  
 2 番目の絶対最大成分とそのグローバル・インデックス。  
 $v2(1) = \text{amax}$ 、 $v2(2) = \text{indx}$ 。

### 出力パラメータ

`v1` (ローカル) 最初の絶対最大成分とそのグローバル・インデックス。  
 $v1(1) = \text{amax}$ 、 $v1(2) = \text{indx}$ 。

## p?sum1

レベル 1 PBLAS の `p?asum` と同様に複素ベクトルの 1- ノルムを実行するが、真の絶対値を用いる。

### 構文

```
call pscsum1(n, asum, x, ix, jx, descx, incx)
call pdzsum1(n, asum, x, ix, jx, descx, incx)
```

### 説明

このルーチンは、複素乱数ベクトル `sub(x)` の絶対値の合計を `asum` に格納する。

ここで、`sub(x)` は次のとおりである。

$X(ix:ix+n-1, jx:jx) (incx = 1 \text{ の場合})$   
 $X(ix:ix, jx:jx+n-1) (incx = m_x \text{ の場合})$

レベル 1 BLAS の `p?asum` を基にしている。違いは、'真の'絶対値を使用していることである。

### 入力パラメータ

`n` (グローバル) INTEGER へのポインタ。  
 乱数ベクトル `sub(x)` の成分の個数。  $n \geq 0$ 。

`x` (ローカル)  
 COMPLEX (pscsum1 の場合)  
 COMPLEX\*16 (pdzsum1 の場合)。

次元が  $(jx-1)*m_x + ix + (n-1)*abs(incx)$  以上の分散行列のローカル部分を格納する配列。乱数ベクトル  $sub(x)$  の成分を格納する。

<i>ix</i>	(グローバル) INTEGER。 $sub(x)$ の最初の行を示す、グローバル配列 $X$ の行インデックス。
<i>jx</i>	(グローバル) INTEGER。 $sub(x)$ の最初の列を示す、グローバル配列 $X$ の列インデックス。
<i>descx</i>	(グローバルおよびローカル) INTEGER。 配列、次元は 8。分散行列 $X$ の配列ディスクリプタ。
<i>incx</i>	(グローバル) INTEGER。 $X$ の成分のグローバルな増分。このバージョンでは、1 と $m_x$ の 2 つの <i>incx</i> の値のみサポートされる。

### 出力パラメータ

<i>asum</i>	(ローカル) REAL へのポインタ。 $sub(x)$ の範囲における、乱数ベクトル $sub(x)$ の最大成分の絶対値。
-------------	--

---

## p?dbtrsv

一般三角行列の LU 因子分解を、ピボット演算を用いずに計算する。p?dbtrs によって呼び出される。

---

### 構文

```
call psdbtrsv(uplo, trans, n, bwl, bwu, nrhs, a, ja, desca, b, ib, descb, af,
  laf, work, lwork, info)
call pddbtrsv(uplo, trans, n, bwl, bwu, nrhs, a, ja, desca, b, ib, descb, af,
  laf, work, lwork, info)
call pcdbtrsv(uplo, trans, n, bwl, bwu, nrhs, a, ja, desca, b, ib, descb, af,
  laf, work, lwork, info)
call pzdbtrsv(uplo, trans, n, bwl, bwu, nrhs, a, ja, desca, b, ib, descb, af,
  laf, work, lwork, info)
```

### 説明

このルーチンは、次の三角帯行列を係数行列とする連立 1 次方程式を解く。

$A(1:n, ja:ja+n-1) * X = B(ib:ib+n-1, 1:nrhs)$  または

$A(1:n, ja:ja+n-1)^T * X = B(ib:ib+n-1, 1:nrhs)$  (実数型の場合)

$A(1:n, ja:ja+n-1)^H * X = B(ib:ib+n-1, 1:nrhs)$  (複素数型の場合)

ここで、 $A(1:n, ja:ja+n-1)$  は、ガウス消去コード PD@(dom\_pre)BTRF によって生成される三角帯行列の係数であり、 $A(1:n, ja:ja+n-1)$  と  $af$  に格納される。

$A(1:n, ja:ja+n-1)$  には、 $uplo$  の値に従って、上三角行列または下三角行列が格納される。 $(A(1:n, ja:ja+n-1)$  と  $A(1:n, ja:ja+n-1)^T)$  のどちらの式を解くのかは、パラメータ  $trans$  を介して、ユーザによって決定される。

ルーチン [p?dbtrf](#) は、最初に呼び出さなければならない。

## 入力パラメータ

$uplo$	(グローバル) CHARACTER。 $uplo = 'U'$ の場合、 $A(1:n, ja:ja+n-1)$ の上三角行列が格納される。 $uplo = 'L'$ の場合、 $A(1:n, ja:ja+n-1)$ の下三角行列が格納される。
$trans$	(グローバル) CHARACTER。  $trans = 'N'$ の場合、 $A(1:n, ja:ja+n-1)$ を解く。 $trans = 'C'$ の場合、共役転置式 $A(1:n, ja:ja+n-1)$ を解く。
$n$	(グローバル) INTEGER。分散部分行列 $A$ の次数。 $(n \geq 0)$ 。
$bwl$	(グローバル) INTEGER。 劣対角成分の数。 $0 \leq bwl \leq n-1$ 。
$bwu$	(グローバル) INTEGER。 劣対角成分の数。 $0 \leq bwu \leq n-1$ 。
$nrhs$	(グローバル) INTEGER。右辺の数。分散部分行列 $B$ の列数 ( $nrhs \geq 0$ )。
$a$	(ローカル) REAL (psdbtrsv の場合) DOUBLE PRECISION (pddbtrsv の場合) COMPLEX (pcdbtrsv の場合) COMPLEX*16 (pzdbtrsv の場合)。 ローカルメモリにある、第 1 次元が $lld\_a \geq (bwl+bwu+1)$ の配列へのポインタ ( $desca$ に格納される)。 $n \times n$ の非対称帯形式の分割コレスキー因子 $L$ または $L^T A(1:n, ja:ja+n-1)$ のローカル部分を格納する。 このローカル部分は、LAPACK で使用される圧縮帯形式で格納される。分散行列の形式の詳細は、次の「アプリケーション・ノート」および ScaLAPACK マニュアルを参照。

<i>ja</i>	(グローバル) INTEGER。(A のすべてまたは A の部分行列で) 処理される行列の先頭を指すグローバル配列 <i>a</i> のインデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen</i> <sub>1</sub> )。 1d type ( <i>dtype_a</i> = 501 または 502) の場合、 <i>dlen</i> ≥ 7。 2d type ( <i>dtype_a</i> = 1) の場合、 <i>dlen</i> ≥ 9。 分散行列 A の配列ディスクリプタ。A のマッピング情報をメモリに格納する。
<i>b</i>	(ローカル) REAL (psdbtrsv の場合) DOUBLE PRECISION (pddbtrsv の場合) COMPLEX (pcdbtrsv の場合) COMPLEX*16 (pzdbtrsv の場合)。 ローカルメモリにある、ローカル・リーディング・ディメンジョン <i>lld_b</i> ≥ <i>nb</i> の配列へのポインタ。右辺 <i>B(ib:ib+n-1, 1:nrhs)</i> のローカル部分を格納する。
<i>ib</i>	(グローバル) INTEGER。(b のすべてまたは B の部分行列で) 処理される行列の最初の行を指すグローバル配列 <i>b</i> の行インデックス。
<i>desb</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen</i> <sub>1</sub> )。 1d type ( <i>dtype_b</i> = 502) の場合、 <i>dlen</i> ≥ 7。 2d type ( <i>dtype_b</i> = 1) の場合、 <i>dlen</i> ≥ 9。 分散行列 B の配列ディスクリプタ。B のマッピング情報をメモリに格納する。
<i>laf</i>	(ローカル) INTEGER。ユーザ入力の補助非零要素空間 <i>af</i> のサイズ。 <i>laf</i> ≥ <i>nb</i> *( <i>bwl</i> + <i>bwu</i> )+6*max( <i>bwl</i> , <i>bwu</i> )*max( <i>bwl</i> , <i>bwu</i> ) でなければならない。 <i>laf</i> が小さすぎる場合、エラーコードが返され、エラーにならない最小のサイズが <i>af</i> (1) に返される。
<i>work</i>	(ローカル) REAL (psdbtrsv の場合) DOUBLE PRECISION (pddbtrsv の場合) COMPLEX (pcdbtrsv の場合) COMPLEX*16 (pzdbtrsv の場合)。 テンポラリ・ワークスペース。このスペースは、ルーチンの呼び出しの間に上書きされる。 <i>work</i> のサイズは <i>lwork</i> で与えられる。

*lwork* (ローカルまたはグローバル) INTEGER。  
 ユーザ入力ワークスペース *work* のサイズ。 *lwork* が小さすぎる場合、エラーコードが返され、エラーにならない最小のサイズが *work*(1) に返される。  
 $lwork \geq \max(bw1, bwu) * nrhs$ 。

## 出力パラメータ

*a* (ローカル)  
 このローカル部分は、LAPACK で使用される圧縮帯形式で格納される。分散行列の形式の詳細は、次の「アプリケーション・ノート」および ScaLAPACK マニュアルを参照。

*b*  
 終了時に、解の分散行列 *X* のローカル部分が格納される。

*af* (ローカル)  
 REAL (psdbtrsv の場合)  
 DOUBLE PRECISION (pddbtrsv の場合)  
 COMPLEX (pcdbtrsv の場合)  
 COMPLEX\*16 (pzdbtrsv の場合)。  
 補助非零要素空間。非零要素は、因子分解ルーチン p?dbtrf で作成され、*af* に格納される。因子分解ルーチンの後で、p?dbtrf を使って連立 1 次方程式を解く場合、因子分解後に *af* を変更してはならない。

*work*  
 終了時に、*lwork* の最小値が *work*(1) に格納される。

*info* (ローカル) INTEGER。 *info* = 0 の場合、正常に終了したことを示す。  
*info* < 0 の場合、*i* 番目の引数が配列で *j* 番目の成分の値が不正ならば *info* = - (*i*\*100+*j*)、*i* 番目の引数がスカラで値が不正ならば *info* = -*i*。

## p?dttrsv

一般帯行列の LU 因子分解を、行交換を伴う部分ピボット演算を用いて計算する。p?dttrs によって呼び出される。

### 構文

```
call psdttrsv(uplo, trans, n, nrhs, dl, d, du, ja, desca, b, ib, descb, af, laf,
              work, lwork, info)
```

```
call pddttrsv(uplo, trans, n, nrhs, dl, d, du, ja, desca, b, ib, descb, af, laf,
work, lwork, info)
call pcdttrsv(uplo, trans, n, nrhs, dl, d, du, ja, desca, b, ib, descb, af, laf,
work, lwork, info)
call pzdttrsv(uplo, trans, n, nrhs, dl, d, du, ja, desca, b, ib, descb, af, laf,
work, lwork, info)
```

## 説明

このルーチンは、次の三重対角三角行列を係数行列とする連立 1 次方程式を解く。

$A(1:n, ja:ja+n-1) * X = B(ib:ib+n-1, 1:nrhs)$  または

$A(1:n, ja:ja+n-1)^T * X = B(ib:ib+n-1, 1:nrhs)$  (実数型の場合)

$A(1:n, ja:ja+n-1)^H * X = B(ib:ib+n-1, 1:nrhs)$  (複素数型の場合)

ここで、 $A(1:n, ja:ja+n-1)$  は、ガウス消去コード PS@(dom\_pre)TTRF によって生成される三重対角行列の係数であり、 $A(1:n, ja:ja+n-1)$  と  $af$  に格納される。

$A(1:n, ja:ja+n-1)$  には、 $uplo$  の値に従って、上三角行列または下三角行列が格納される。 $A(1:n, ja:ja+n-1)$  と  $A(1:n, ja:ja+n-1)^T$  のどちらの式を解くのかは、パラメータ  $trans$  を介して、ユーザによって決定される。

ルーチン [p?dttrf](#) は、最初に呼び出さなければならない。

## 入力パラメータ

$uplo$	(グローバル) CHARACTER。 $uplo = 'U'$ の場合、 $A(1:n, ja:ja+n-1)$ の上三角行列が格納される。 $uplo = 'L'$ の場合、 $A(1:n, ja:ja+n-1)$ の下三角行列が格納される。
$trans$	(グローバル) CHARACTER。 $trans = 'N'$ の場合、 $A(1:n, ja:ja+n-1)$ を解く。 $trans = 'C'$ の場合、共役転置式 $A(1:n, ja:ja+n-1)$ を解く。
$n$	(グローバル) INTEGER。分散部分行列 $A$ の次数。 $(n \geq 0)$ 。
$nrhs$	(グローバル) INTEGER。右辺の数。 分散部分行列 $B(ib:ib+n-1, 1:nrhs)$ の列数。 $(nrhs \geq 0)$ 。
$dl$	(ローカル) REAL (psdttrsv の場合) DOUBLE PRECISION (pddttrsv の場合) COMPLEX (pcdttrsv の場合) COMPLEX*16 (pzdttrsv の場合)。



行列の下対角を格納するグローバル・ベクトルのローカル部分へのポインタ。全体的に、 $d1(1)$  は参照されず、 $d1$  は  $d$  とアライメントされなければならない。

サイズは  $desca(nb\_)$  以上でなければならない。

$d$	(ローカル) REAL (psdttrsv の場合) DOUBLE PRECISION (pddttrsv の場合) COMPLEX (pcdttrsv の場合) COMPLEX*16 (pzdttrsv の場合)。 行列の主対角を格納するグローバル・ベクトルのローカル部分へのポインタ。
$du$	(ローカル) REAL (psdttrsv の場合) DOUBLE PRECISION (pddttrsv の場合) COMPLEX (pcdttrsv の場合) COMPLEX*16 (pzdttrsv の場合)。 行列の上対角を格納するグローバル・ベクトルのローカル部分へのポインタ。全体的に、 $du(n)$ は参照されず、 $du$ は $d$ とアライメントされなければならない。
$ja$	(グローバル) INTEGER。 ( $A$ のすべてまたは $A$ の部分行列で) 処理される行列の先頭を指すグローバル配列 $a$ のインデックス。
$desca$	(グローバルおよびローカル) INTEGER。 配列、次元は ( $dlen\_$ )。 1d type ( $dtype\_a = 501$ または $502$ ) の場合、 $dlen \geq 7$ 。 2d type ( $dtype\_a = 1$ ) の場合、 $dlen \geq 9$ 。 分散行列 $A$ の配列ディスクリプタ。 $A$ のマッピング情報をメモリに格納する。
$b$	(ローカル) REAL (psdttrsv の場合) DOUBLE PRECISION (pddttrsv の場合) COMPLEX (pcdttrsv の場合) COMPLEX*16 (pzdttrsv の場合)。 ローカルメモリにある、ローカル・リーディング・ディメンジョン $lld\_b \geq nb$ の配列へのポインタ。右辺 $B(ib:ib+n-1, 1:nrhs)$ のローカル部分を格納する。
$ib$	(グローバル) INTEGER。 ( $b$ のすべてまたは $B$ の部分行列で) 処理される行列の最初の行を指すグローバル配列 $b$ の行インデックス。

<i>desb</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。 1d type ( <i>dtype_b</i> = 502) の場合、 <i>dlen</i> ≥ 7。 2d type ( <i>dtype_b</i> = 1) の場合、 <i>dlen</i> ≥ 9。 分散行列 <i>B</i> の配列ディスクリプタ。 <i>B</i> のマッピング情報をメモリに格納する。
<i>laf</i>	(ローカル) INTEGER。ユーザ入力の補助非零要素空間 <i>af</i> のサイズ。 $laf \geq 2 \cdot (nb + 2)$ でなければならない。 <i>laf</i> が小さすぎる場合、エラーコードが返され、エラーにならない最小のサイズが <i>af</i> (1) に返される。
<i>work</i>	(ローカル)  REAL ( <i>psdttrsv</i> の場合) DOUBLE PRECISION ( <i>pddttrsv</i> の場合) COMPLEX ( <i>pcdttrsv</i> の場合) COMPLEX*16 ( <i>pzdttrsv</i> の場合)。 テンポラリ・ワークスペース。このスペースは、ルーチンの呼び出しの間に上書きされる。 <i>work</i> のサイズは <i>lwork</i> で与えられる。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 ユーザ入力ワークスペース <i>work</i> のサイズ。 <i>lwork</i> が小さすぎる場合、エラーコードが返され、エラーにならない最小のサイズが <i>work</i> (1) に返される。 $lwork \geq 10 \cdot npcol + 4 \cdot nrhs$ 。

## 出力パラメータ

<i>d1</i>	(ローカル) 終了時に、行列の係数を含む情報が格納される。
<i>d</i>	終了時に、行列の係数を含む情報が格納される。 サイズは ≥ <i>desca</i> ( <i>nb_</i> ) でなければならない。
<i>b</i>	終了時に、解の分散行列 <i>X</i> のローカル部分が格納される。
<i>af</i>	(ローカル)  REAL ( <i>psdttrsv</i> の場合) DOUBLE PRECISION ( <i>pddttrsv</i> の場合) COMPLEX ( <i>pcdttrsv</i> の場合) COMPLEX*16 ( <i>pzdttrsv</i> の場合)。 補助非零要素空間。非零要素空間は、因子分解ルーチン <i>p?dttrf</i> で作成され、 <i>af</i> に格納される。因子分解ルーチンの後で、 <a href="#">p?dttrs</a> を使って連立 1 次方程式を解く場合、因子分解後に <i>af</i> を変更してはならない。

*work*                    終了時に、*lwork* の最小値が *work*(1) に格納される。

*info*                    (ローカル) INTEGER。  
*info* = 0 の場合、正常に終了したことを示す。  
*info* < 0 の場合、*i* 番目の引数が配列で *j* 番目の成分の値が不正ならば *info* = - (*i*\*100+*j*)、*i* 番目の引数がスカラで値が不正ならば *info* = -*i*。

## p?gebd2

直交/ユニタリ変換を用いて、一般矩形行列を実数二重対角形式に縮退させる (非ブロック化アルゴリズム)。

### 構文

```
call psgebd2(m, n, a, ia, ja, desca, d, e, tauq, taup, work, lwork, info)
call pdgebd2(m, n, a, ia, ja, desca, d, e, tauq, taup, work, lwork, info)
call pcgebd2(m, n, a, ia, ja, desca, d, e, tauq, taup, work, lwork, info)
call pzgebd2(m, n, a, ia, ja, desca, d, e, tauq, taup, work, lwork, info)
```

### 説明

このルーチンは、直交/ユニタリ変換を用いて、 $m \times n$  の一般実/複素分布行列  $\text{sub}(A) = A(ia:ia+m-1, ja:ja+n-1)$  を上または下二重対角形式  $B$  に縮退させる。

$$Q' * \text{sub}(A) * P = B$$

$m \geq n$  の場合、 $B$  は上二重対角である。 $m < n$  の場合、 $B$  は下二重対角である。

### 入力パラメータ

*m*                    (グローバル) INTEGER。  
分散部分行列  $\text{sub}(A)$  の行数。( $m \geq 0$ )。

*n*                    (グローバル) INTEGER。分散部分行列  $\text{sub}(A)$  の次数。( $n \geq 0$ )。

*a*                    (ローカル)  
REAL (psgebd2 の場合)  
DOUBLE PRECISION (pdgebd2 の場合)  
COMPLEX (pcgebd2 の場合)

	COMPLEX*16 (pzgebd2 の場合)。 ローカルメモリにある、次元 ( $lld\_a$ , $LOCc(ja+n-1)$ ) の配列へのポインタ。一般分散行列 $sub(A)$ のローカル部分を格納する。
<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 $A$ の最初の行と最初の列を示す、グローバル配列 $a$ の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( $dlen\_$ )。分散行列 $A$ の配列ディスクリプタ。
<i>work</i>	(ローカル) REAL (psgebd2 の場合) DOUBLE PRECISION (pdgebd2 の場合) COMPLEX (pcgebd2 の場合) COMPLEX*16 (pzgebd2 の場合)。 次元 ( $lwork$ ) のワークスペース配列。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 配列 $work$ の次元。 $lwork$ はローカル入力であり、 $lwork \geq \max(mpa0, nqa0)$ でなければならない。ここで、 $nb = mb\_a = nb\_a, iroffa = \text{mod}(ia-1, nb)$ $iarow = \text{indxg2p}(ia, nb, myrow, rsrc\_a, nprow),$ $iacol = \text{indxg2p}(ja, nb, mycol, csrc\_a, npc1),$ $mpa0 = \text{numroc}(m+iroffa, nb, myrow, iarow, nprow),$ $nqa0 = \text{numroc}(n+icoffa, nb, mycol, iacol, npc1)。$ $indxg2p$ および $numroc$ は、ScaLAPACK ツール関数である。 $myrow$ , $mycol$ , $nprow$ 、および $npcol$ は、サブルーチン $blacs\_gridinfo$ を呼び出して求められる。 $lwork = -1$ の場合、 $lwork$ はグローバル入力であり、ワークスペースのクエリとみなされ、ルーチンはすべての配列の最小かつ最適なサイズだけを計算する。各値は該当する $work$ 配列の最初のエントリとして返され、 <a href="#">pxerbla</a> はエラー・メッセージを生成しない。

## 出力パラメータ

<i>a</i>	(ローカル) 終了時に、 $m \geq n$ の場合、 $sub(A)$ の対角および第 1 の優対角成分は、上二重対角行列 $B$ によって上書きされる。対角よりも下の成分は、配列 $tauq$ とともに、基本リフレクタの積として直交/ユニタリの行列 $Q$ を表現する。また、第 1 の優対角よりも上の成分は、配列 $taup$ とともに、基本リフレクタの積として直交行列 $P$ を表現する。
----------	---

$m < n$  の場合、対角および第 1 の優対角成分は、下二重対角行列  $B$  によって上書きされる。第 1 の劣対角よりも下の成分は、配列  $\text{tauq}$  とともに、基本リフレクタの積として直交 / ユニタリ行列  $Q$  を表現する。対角よりも上の成分は、配列  $\text{taup}$  とともに、基本リフレクタの積として直交行列  $P$  を表現する。次の「アプリケーション・ノート」を参照。

- $d$  (ローカル)  
 REAL (psgebd2 の場合)  
 DOUBLE PRECISION (pdgebd2 の場合)  
 COMPLEX (pcgebd2 の場合)  
 COMPLEX\*16 (pzgebd2 の場合)。  
 配列、次元は  $LOCc(ja+\min(m,n)-1)$  ( $m \geq n$  の場合) または  $LOCr(ia+\min(m,n)-1)$  (それ以外の場合)。二重対角行列  $B$  の分散対角成分  $d(i) = a(i,i)$ 。  $d$  は、分散行列  $A$  に関連付けられる。
- $e$  (ローカル)  
 REAL (psgebd2 の場合)  
 DOUBLE PRECISION (pdgebd2 の場合)  
 COMPLEX (pcgebd2 の場合)  
 COMPLEX\*16 (pzgebd2 の場合)。  
 配列、次元は  $LOCc(ja+\min(m,n)-1)$  ( $m \geq n$  の場合) または  $LOCr(ia+\min(m,n)-2)$  (それ以外の場合)。二重対角行列  $B$  の分散対角成分が格納される。  
 $m \geq n$  の場合、 $e(i) = a(i, i+1)$  ( $i = 1, 2, \dots, n-1$ )。  
 $m < n$  の場合、 $e(i) = a(i+1, i)$  ( $i = 1, 2, \dots, m-1$ )。  $e$  は、分散行列  $A$  に関連付けられる。
- $\text{tauq}$  (ローカル)  
 REAL (psgebd2 の場合)  
 DOUBLE PRECISION (pdgebd2 の場合)  
 COMPLEX (pcgebd2 の場合)  
 COMPLEX\*16 (pzgebd2 の場合)。  
 配列、次元は  $LOCc(ja+\min(m,n)-1)$ 。直交 / ユニタリ行列  $Q$  を表す基本リフレクタのスカラー係数。  $\text{tauq}$  は、分散行列  $A$  に関連付けられる。
- $\text{taup}$  (ローカル)  
 REAL (psgebd2 の場合)  
 DOUBLE PRECISION (pdgebd2 の場合)  
 COMPLEX (pcgebd2 の場合)  
 COMPLEX\*16 (pzgebd2 の場合)。  
 配列、次元は  $LOCr(ia+\min(m,n)-1)$ 。直交 / ユニタリ行列  $P$  を表す基本リフレクタのスカラー係数。  $\text{taup}$  は、分散行列  $A$  に関連付けられる。

*work* 終了時に、*work(1)* は、最小かつ最適な *lwork* 値を返す。

*info* (ローカル) INTEGER。  
*info* = 0 の場合、正常に終了したことを示す。  
*info* < 0 の場合、*i* 番目の引数が配列で *j* 番目の成分の値が不正ならば *info* = - (*i*\*100+*j*)、*i* 番目の引数がスカラーで値が不正ならば *info* = -*i*。

## アプリケーション・ノート

行列 *Q* と *P* は基本リフレクタの積として表現される。

$m \geq n$  の場合、

$$Q = H(1) H(2) \dots H(n) \text{ および } P = G(1) G(2) \dots G(n-1)$$

それぞれの  $H(i)$  と  $G(i)$  は次のような形式を持つ。

$$H(i) = I - \tau_{\text{auq}} * v * v' \text{ および } G(i) = I - \tau_{\text{aup}} * u * u'$$

ここで、 $\tau_{\text{auq}}$  と  $\tau_{\text{aup}}$  は実 / 複素スカラー、 $v$  と  $u$  は実 / 複素ベクトルである。

終了時に、 $v(1:i-1) = 0$ 、 $v(i) = 1$ 、 $v(i+1:m)$  は

$A(ia+i-ia+m-1, ja+i-1)$  に格納される。

終了時に、 $u(1:i) = 0$ 、 $u(i+1) = 1$ 、 $u(i+2:n)$  は  $A(ia+i-1, ja+i+1:ja+n-1)$  に格納される。

$\tau_{\text{auq}}$  は  $TAUQ(ja+i-1)$  に、 $\tau_{\text{aup}}$  は  $TAUP(ia+i-1)$  に格納される。

$m < n$  の場合

終了時に、 $v(1:i) = 0$ 、 $v(i+1) = 1$ 、 $v(i+2:m)$  は  $A(ia+i+1:ia+m-1, ja+i-1)$  に格納される。

終了時に、 $u(1:i-1) = 0$ 、 $u(i) = 1$ 、 $u(i+1:n)$  は  $A(ia+i-1, ja+i:ja+n-1)$  に格納される。

$\tau_{\text{auq}}$  は  $TAUQ(ja+i-1)$  に、 $\tau_{\text{aup}}$  は  $TAUP(ia+i-1)$  に格納される。

終了時の  $\text{sub}(A)$  の内容を次に示す。

$m = 6$ 、 $n = 5$  ( $m > n$ ) の場合：

$$\begin{bmatrix} d & e & u1 & u1 & u1 \\ v1 & d & e & u2 & u2 \\ v1 & v2 & d & e & u3 \\ v1 & v2 & v3 & d & e \\ v1 & v2 & v3 & v4 & d \\ v1 & v2 & v3 & v4 & v5 \end{bmatrix}$$

$m = 5$ 、 $n = 6$  ( $m < n$ ) の場合：

$$\begin{bmatrix} d & u1 & u1 & u1 & u1 & u1 \\ e & d & u2 & u2 & u2 & u2 \\ v1 & e & d & u3 & u3 & u3 \\ v1 & v2 & e & d & u4 & u4 \\ v1 & v2 & v3 & e & d & u5 \end{bmatrix}$$

ここで、 $d$  と  $e$  は  $B$  の対角成分と非対角成分である。 $v_i$  は  $H(i)$  を定義するベクトルの成分を表し、 $u_i$  は  $G(i)$  を定義するベクトルの成分を表す。

## p?gehd2

直交 / ユニタリ相似変換を用いて、一般行列を上 Hessenberg 形式に縮退させる (非ブロック化アルゴリズム)。

### 構文

```
call psgehd2(n, ilo, ihi, a, ia, ja, desca, tau, work, lwork, info)
call pdgehd2(n, ilo, ihi, a, ia, ja, desca, tau, work, lwork, info)
call pcgehd2(n, ilo, ihi, a, ia, ja, desca, tau, work, lwork, info)
call pzgehd2(n, ilo, ihi, a, ia, ja, desca, tau, work, lwork, info)
```

### 説明

このルーチンは、直交 / ユニタリの相似変換  $Q' * \text{sub}(A) * Q = H$  によって、実 / 複素一般行列  $\text{sub}(A)$  を上 Hessenberg 形式  $H$  に縮退させる。ここで、 $\text{sub}(A) = A(ia+n-1:ia+n-1, ja+n-1:ja+n-1)$  である。

### 入力パラメータ

$n$  (グローバル) INTEGER。分散部分行列  $A$  の次数。 $(n \geq 0)$ 。

$ilo, ihi$  (グローバル) INTEGER。  $\text{sub}(A)$  はすでに行  $ia:ia+ilo-2$  と列  $ja:ja+jlo-2$ 、行  $ia+ihi:ia+n-1$  と  $ja+jhi:ja+n-1$  で上三角になっていると仮定する。詳細は、「アプリケーション・ノート」を参照。  
 $n > 0$  の場合は  $1 \leq ilo \leq ihi \leq n$ 、それ以外の場合は  $ilo = 1$  かつ  $ihi = n$ 。

$a$  (ローカル)

REAL (psgehd2 の場合)  
 DOUBLE PRECISION (pdgehd2 の場合)  
 COMPLEX (pcgehd2 の場合)  
 COMPLEX\*16 (pzgehd2 の場合)。

ローカルメモリにある、次元 ( $lld\_a$ ,  $LOCc(ja+n-1)$ ) の配列へのポインタ。呼び出し時に、この配列は、因子分解する  $n \times n$  の一般分散行列  $sub(A)$  のローカル部分を含む。

<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 $A$ の最初の行と最初の列を示す、グローバル配列 $A$ の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( $dlen\_$ )。分散行列 $A$ の配列ディスクリプタ。
<i>work</i>	(ローカル) REAL (psgehd2 の場合) DOUBLE PRECISION (pdgehd2 の場合) COMPLEX (pcgehd2 の場合) COMPLEX*16 (pzgehd2 の場合)。 次元 ( $lwork$ ) のワークスペース配列。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 配列 <i>work</i> の次元。 $lwork$ はローカル入力であり、 $lwork \geq nb + \max(npa0, nb)$ でなければならない。ここで、 $nb = mb\_a = nb\_a$ , $iroffa = \text{mod}(ia-1, nb)$ 、 $iarow = \text{indxg2p}(ia, nb, myrow, rsrc\_a, nprow)$ 、 $npa0 = \text{numroc}(ihi+iroffa, nb, myrow, iarow, nprow)$ 。  $\text{indxg2p}$ および $\text{numroc}$ は、ScaLAPACK ツール関数である。 $myrow$ , $mycol$ , $nprow$ , および $npcol$ は、サブルーチン $\text{blacs\_gridinfo}$ を呼び出して求められる。  $lwork = -1$ の場合、 $lwork$ はグローバル入力であり、ワークスペースのクエリとみなされ、ルーチンはすべての配列の最小かつ最適なサイズだけを計算する。各値は該当する <i>work</i> 配列の最初のエントリとして返され、 <a href="#">pxerbla</a> はエラー・メッセージを生成しない。

## 出力パラメータ

<i>a</i>	(ローカル) 終了時に、 $sub(A)$ の上三角と第 1 の劣対角は、上 Hessenberg 行列 $H$ および第 1 の劣対角よりも下の成分で上書きされ、配列 <i>tau</i> とともに、基本リフレクタの積として直交/ユニタリの行列 $Q$ を表現する。次の「アプリケーション・ノート」を参照。
<i>tau</i>	(ローカル) REAL (psgehd2 の場合) DOUBLE PRECISION (pdgehd2 の場合) COMPLEX (pcgehd2 の場合)



COMPLEX\*16 (pzgehd2 の場合)。

配列、次元は  $LOCc(ja+n-2)$ 。基本リフレクタのスカラ係数 (次の「アプリケーション・ノート」を参照)。 $\tau$  の成分  $ja:ja+ilo-2$  と  $ja+ihi:ja+n-2$  はゼロに設定される。 $\tau$  は、分散行列  $A$  に関連付けられる。

**work** 終了時に、 $work(1)$  は、最小かつ最適な  $lwork$  値を返す。

**info** (ローカル) INTEGER。  
 $info=0$  の場合、正常に終了したことを示す。  
 $info<0$  の場合、 $i$  番目の引数が配列で  $j$  番目の成分の値が不正ならば  $info = -(i*100+j)$ 、 $i$  番目の引数がスカラで値が不正ならば  $info = -i$ 。

### アプリケーション・ノート

行列  $Q$  を  $(ihi-ilo)$  個の基本リフレクタの積として表現する。

$$Q = H(ilo) H(ilo+1) \dots H(ihi-1)$$

それぞれの  $H(i)$  は次のような形式を持つ。

$$H(i) = I - \tau v v'$$

ここで、 $\tau$  は実 / 複素スカラ、 $v$  は実 / 複素ベクトルで  $v(1:i) = 0$ 、 $v(i+1) = 1$  および  $v(ihi+1:n) = 0$ 。 $v(i+2:ihi)$  は、終了時に、 $A(ia+ilo+i:ia+ihi-1, ia+ilo+i-2)$  に、 $\tau$  は  $\tau(ja+ilo+i-2)$  に格納される。

終了時の  $A(ia:ia+n-1, ja:ja+n-1)$  の内容を次に示す ( $n=7$ 、 $ilo=2$ 、および  $ihi=6$  の場合)。

入力

出力

$$\begin{array}{c} \begin{bmatrix} a & a & a & a & a & a & a \\ & a & a & a & a & a & a \\ & & a & a & a & a & a \\ & & & a & a & a & a \\ & & & & a & a & a \\ & & & & & a & a \\ & & & & & & a \end{bmatrix} \end{array} \qquad \begin{array}{c} \begin{bmatrix} a & a & h & h & h & h & a \\ & a & h & h & h & h & a \\ & & h & h & h & h & h \\ & & & v2 & h & h & h & h \\ & & & v2 & v3 & h & h & h & h \\ & & & v2 & v3 & v4 & h & h & h \\ & & & & & & & a \end{bmatrix} \end{array}$$

ここで、 $a$  は元の行列  $\text{sub}(A)$  の成分を表し、 $h$  は上 Hessenberg 行列  $H$  の更新された成分を表し、 $v_i$  は  $H(ja+ilo+i-2)$  を定義するベクトルの成分を表している。

## p?gelq2

一般矩形行列の  $LQ$  因子分解を計算する ( 非ブロック化アルゴリズム )。

### 構文

```
call psgelq2(m, n, a, ia, ja, desca, tau, work, lwork, info)
call psgelq2(m, n, a, ia, ja, desca, tau, work, lwork, info)
call psgelq2(m, n, a, ia, ja, desca, tau, work, lwork, info)
call psgelq2(m, n, a, ia, ja, desca, tau, work, lwork, info)
```

### 説明

このルーチンは、 $m \times n$  の実 / 複素行列  $\text{sub}(A) = A(ia:ia+m-1, ja:ja+n-1) = L * Q$  の  $LQ$  因子分解を計算する。

### 入力パラメータ

<i>m</i>	( グローバル ) INTEGER。 演算を行う行数、すなわち、分散部分行列 $\text{sub}(A)$ の行数。 ( $m \geq 0$ )。
<i>n</i>	( グローバル ) INTEGER。 演算を行う列数、すなわち、分散部分行列 $\text{sub}(A)$ の列数。 ( $n \geq 0$ )。
<i>a</i>	( ローカル ) REAL (psgelq2 の場合 ) DOUBLE PRECISION (pdgelq2 の場合 ) COMPLEX (pcgelq2 の場合 ) COMPLEX*16 (pzgelq2 の場合 )。 ローカルメモリにある、次元 ( $lld\_a$ , $LOCc(ja+n-1)$ ) の配列へのポインタ。因子分解する $m \times n$ の分散行列 $\text{sub}(A)$ のローカル部分を格納する。
<i>ia, ja</i>	( グローバル ) INTEGER。それぞれ、部分行列 $A$ の最初の行と最初の列を示す、グローバル配列 $a$ の行インデックスと列インデックス。
<i>desca</i>	( グローバルおよびローカル ) INTEGER。配列、次元は ( $dlen\_$ )。分散行列 $A$ の配列ディスクリプタ。

**work** (ローカル)  
 REAL (psgelq2 の場合)  
 DOUBLE PRECISION (pdgelq2 の場合)  
 COMPLEX (pcgelq2 の場合)  
 COMPLEX\*16 (pzgelq2 の場合)。  
 次元 (*lwork*) のワークスペース配列。

**lwork** (ローカルまたはグローバル) INTEGER。  
 配列 **work** の次元。  
*lwork* はローカル入力であり、 $lwork \geq nq0 + \max(1, mp0)$  でなければならない。ここで、

```

iroff = mod(ia-1, mb_a)、icoff = mod(ja-1, nb_a)、
iarow = indxg2p(ia, mb_a, myrow, rsrc_a, nprow)、
iacol = indxg2p(ja, nb_a, mycol, csrc_a, npcol)、
mp0 = numroc(m+iroff, mb_a, myrow, iarow, nprow)、
nq0 = numroc(n+icoff, nb_a, mycol, iacol, npcol)。

```

indxg2p および numroc は、ScaLAPACK ツール関数である。  
*myrow*, *mycol*, *nprow*、および *npcol* は、サブルーチン `blacs_gridinfo` を呼び出して求められる。

*lwork* = -1 の場合、*lwork* はグローバル入力であり、ワークスペースのクエリとみなされ、ルーチンはすべての配列の最小かつ最適なサイズだけを計算する。各値は該当する **work** 配列の最初のエントリとして返され、[pxerbla](#) はエラー・メッセージを生成しない。

## 出力パラメータ

**a** (ローカル)  
 終了時に、`sub(A)` の対角成分とその下の成分は、 $m \times \min(m, n)$  の下台形行列  $L$  ( $m \leq n$  の場合、 $L$  は下三角行列) によって上書きされる。対角よりも上の成分は、配列 **tau** とともに、基本リフレクタの積として直交/ユニタリの行列  $Q$  を表現する (次の「アプリケーション・ノート」を参照)。

**tau** (ローカル)  
 REAL (psgelq2 の場合)  
 DOUBLE PRECISION (pdgelq2 の場合)  
 COMPLEX (pcgelq2 の場合)  
 COMPLEX\*16 (pzgelq2 の場合)。  
 配列、次元は  $LOCr(ia+\min(m,n)-1)$ 。基本リフレクタのスカラー係数が格納される。**tau** は、分散行列  $A$  に関連付けられる。

*work*                    終了時に、*work*(1) は、最小かつ最適な *lwork* 値を返す。

*info*                    (ローカル)。INTEGER。  
                         *info* = 0 の場合、正常に終了したことを示す。  
                         *info* < 0 の場合、*i* 番目の引数が配列で *j* 番目の成分の値が不正ならば *info* = - (i\*100+j)、*i* 番目の引数がスカラーで値が不正ならば *info* = -*i*。

### アプリケーション・ノート

行列  $Q$  は基本リフレクタの積として表現される。

$Q = H(ia+k-1) H(ia+k-2) \dots H(ia)$  (実数型の場合)  
 $Q = H(ia+k-1)' H(ia+k-2)' \dots H(ia)'$  (複素数型の場合)

ここで、 $k = \min(m, n)$ 。

それぞれの  $H(i)$  は次のような形式を持つ。

$$H(i) = I - \tau * v * v'$$

ここで、 $\tau$  は実 / 複素スカラー、 $v$  は実 / 複素ベクトルで  $v(1:i-1) = 0$  および  $v(i) = 1$ 。終了時に、 $v(i+1:n)$  (実数型の場合) と  $\text{conjg}(v(i+1:n))$  (複素数型の場合) は  $A(ia+i-1, ja+i:ja+n-1)$  に、 $\tau$  は  $TAU(ia+i-1)$  に格納される。

---

## psgeql2

一般矩形行列の  $QL$  因子分解を計算する (非ブロック化アルゴリズム)。

---

### 構文

```
call psgeql2(m, n, a, ia, ja, desca, tau, work, lwork, info)
call psgeql2(m, n, a, ia, ja, desca, tau, work, lwork, info)
call psgeql2(m, n, a, ia, ja, desca, tau, work, lwork, info)
call psgeql2(m, n, a, ia, ja, desca, tau, work, lwork, info)
```

### 説明

このルーチンは、 $m \times n$  の実 / 複素行列  $\text{sub}(A) = A(ia:ia+m-1, ja:ja+n-1) = Q * L$  の  $QL$  因子分解を計算する。

## 入力パラメータ

<i>m</i>	(グローバル) INTEGER。 演算を行う行数、すなわち、分散部分行列 <i>sub(A)</i> の行数。( $m \geq 0$ )。
<i>n</i>	(グローバル) INTEGER。 演算を行う列数、すなわち、分散部分行列 <i>sub(A)</i> の列数。( $n \geq 0$ )。
<i>a</i>	(ローカル) REAL (psgeql2 の場合) DOUBLE PRECISION (pdgeql2 の場合) COMPLEX (pcgeql2 の場合) COMPLEX*16 (pzgeql2 の場合)。 ローカルメモリにある、次元 ( <i>lld_a</i> , <i>LOCc(ja+n-1)</i> ) の配列へのポインタ。因子分解する $m \times n$ の分散行列 <i>sub(A)</i> のローカル部分を格納する。
<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 <i>A</i> の最初の行と最初の列を示す、グローバル配列 <i>a</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。分散行列 <i>A</i> の配列ディスクリプタ。
<i>work</i>	(ローカル) REAL (psgeql2 の場合) DOUBLE PRECISION (pdgeql2 の場合) COMPLEX (pcgeql2 の場合) COMPLEX*16 (pzgeql2 の場合)。 次元 ( <i>lwork</i> ) のワークスペース配列。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 配列 <i>work</i> の次元。 <i>lwork</i> はローカル入力であり、 $lwork \geq mp0 + \max(1, nq0)$ でなければならない。ここで、 $iroff = \text{mod}(ia-1, mb\_a), \quad icoff = \text{mod}(ja-1, nb\_a),$ $iarow = \text{indxg2p}(ia, mb\_a, myrow, rsrc\_a, nprow),$ $iacol = \text{indxg2p}(ja, nb\_a, mycol, csrc\_a, npcot),$ $mp0 = \text{numroc}(m+iroff, mb\_a, myrow, iarow, nprow),$ $nq0 = \text{numroc}(n+icoff, nb\_a, mycol, iacol, npcot).$ <i>indxg2p</i> および <i>numroc</i> は、ScaLAPACK ツール関数である。 <i>myrow</i> , <i>mycol</i> , <i>nprow</i> , および <i>npcot</i> は、サブルーチン <i>blacs_gridinfo</i> を呼び出して求められる。

$lwork = -1$  の場合、 $lwork$  はグローバル入力であり、ワークスペースのクエリとみなされ、ルーチンはすべての配列の最小かつ最適なサイズだけを計算する。各値は該当する  $work$  配列の最初のエントリとして返され、[pxerbla](#) はエラー・メッセージを生成しない。

## 出力パラメータ

$a$	(ローカル) 終了時に、 $m \geq n$ の場合、分散部分行列 $A(ia+m-n:ia+m-1, ja:ja+n-1)$ の下三角部分は、 $n \times n$ の下三角行列 $L$ によって上書きされる。 $m \leq n$ の場合、 $(n-m)$ 番目の優対角成分とその下の各成分は、 $m \times n$ 下台形行列 $L$ によって上書きされる。残りの成分は、配列 $\tau$ とともに、基本リフレクタの積として直交/ユニタリ行列 $Q$ を表現する (次の「アプリケーション・ノート」を参照)。
$\tau$	(ローカル) REAL (psgeql2 の場合) DOUBLE PRECISION (pdgeql2 の場合) COMPLEX (pcgeql2 の場合) COMPLEX*16 (pzgeql2 の場合)。 配列、次元は $LOC(ja+n-1)$ 。基本リフレクタのスカラー係数が格納される。 $\tau$ は、分散行列 $A$ に関連付けられる。
$work$	終了時に、 $work(1)$ は、最小かつ最適な $lwork$ 値を返す。
$info$	(ローカル)。INTEGER。 $info = 0$ の場合、正常に終了したことを示す。 $info < 0$ の場合、 $i$ 番目の引数が配列で $j$ 番目の成分の値が不正ならば $info = -(i*100+j)$ 、 $i$ 番目の引数がスカラーで値が不正ならば $info = -i$ 。

## アプリケーション・ノート

行列  $Q$  は基本リフレクタの積として表現される。

$$Q = H(ja+k-1) \dots H(ja+1) H(ja)。$$

ここで、 $k = \min(m, n)$ 。

それぞれの  $H(i)$  は次のような形式を持つ。

$$H(i) = I - \tau * v * v'$$

$\tau$  は実/複素スカラー、 $v$  は実/複素ベクトルで  $v(m-k+i+1:m) = 0$  および  $v(m-k+i) = 1$ 。終了時に、 $v(1:m-k+i-1)$  は  $A(ia:ia+m-k+i-2, ja+n-k+i-1)$  に、 $\tau$  は  $TAU(ja+n-k+i-1)$  に格納される  $B$  に格納される。

## p?geqr2

一般矩形行列の  $QR$  因子分解を計算する ( 非ブ  
ロック化アルゴリズム )。

### 構文

```
call psgeqr2(m, n, a, ia, ja, desca, tau, work, lwork, info)
call psgeqr2(m, n, a, ia, ja, desca, tau, work, lwork, info)
call psgeqr2(m, n, a, ia, ja, desca, tau, work, lwork, info)
call psgeqr2(m, n, a, ia, ja, desca, tau, work, lwork, info)
```

### 説明

このルーチンは、 $m \times n$  の実 / 複素行列  $\text{sub}(A) = A(ia:ia+m-1, ja:ja+n-1) = Q * R$  の  $QR$  因子分解を計算する。

### 入力パラメータ

$m$	( グローバル ) INTEGER。 演算を行う行数、すなわち、分散部分行列 $\text{sub}(A)$ の行数。 ( $m \geq 0$ )。
$n$	( グローバル ) INTEGER。 演算を行う列数、すなわち、分散部分行列 $\text{sub}(A)$ の列数。 ( $n \geq 0$ )。
$a$	( ローカル ) REAL (psgeqr2 の場合 ) DOUBLE PRECISION (pdgeqr2 の場合 ) COMPLEX (pcgeqr2 の場合 ) COMPLEX*16 (pzgeqr2 の場合 )。 ローカルメモリにある、次元 ( $lld\_a$ , $LOCc(ja+n-1)$ ) の配列へのポインタ。因子分解する $m \times n$ の分散行列 $\text{sub}(A)$ のローカル部分を格納する。
$ia, ja$	( グローバル ) INTEGER。それぞれ、部分行列 $A$ の最初の行と最初の列を示す、グローバル配列 $a$ の行インデックスと列インデックス。
$desca$	( グローバルおよびローカル ) INTEGER。配列、次元は ( $dlen\_$ )。分散行列 $A$ の配列ディスクリプタ。

<code>work</code>	(ローカル) REAL (psgeqr2 の場合) DOUBLE PRECISION (pdgeqr2 の場合) COMPLEX (pcgeqr2 の場合) COMPLEX*16 (pzgeqr2 の場合)。 次元 ( <code>lwork</code> ) のワークスペース配列。
<code>lwork</code>	(ローカルまたはグローバル) INTEGER。 配列 <code>work</code> の次元。 <code>lwork</code> はローカル入力であり、 $lwork \geq mp0 + \max(1, nq0)$ でなければならない。ここで、 $iroff = \text{mod}(ia-1, mb\_a)$ 、 $icoff = \text{mod}(ja-1, nb\_a)$ 、 $iarow = \text{indxg2p}(ia, mb\_a, myrow, rsrc\_a, nprow)$ 、 $iacol = \text{indxg2p}(ja, nb\_a, mycol, csrc\_a, npcot)$ 、 $mp0 = \text{numroc}(m+iroff, mb\_a, myrow, iarow, nprow)$ 、 $nq0 = \text{numroc}(n+icoff, nb\_a, mycol, iacol, npcot)$ 。  <code>indxg2p</code> および <code>numroc</code> は、ScaLAPACK ツール関数である。 <code>myrow</code> 、 <code>mycol</code> 、 <code>nprow</code> 、および <code>npcot</code> は、サブルーチン <code>blacs_gridinfo</code> を呼び出して求められる。  <code>lwork = -1</code> の場合、 <code>lwork</code> はグローバル入力であり、ワークスペース のクエリとみなされ、ルーチンはすべての配列の最小かつ最適なサイ ズだけを計算する。各値は該当する <code>work</code> 配列の最初のエントリとし て返され、 <a href="#">pxerbla</a> はエラー・メッセージを生成しない。

## 出力パラメータ

<code>a</code>	(ローカル) 終了時に、 <code>sub(A)</code> の対角成分とその上の成分は、 $\min(m, n) \times n$ の上台 形行列 $R$ ( $m \geq n$ の場合、 $R$ は上三角行列) によって上書きされる。対 角よりも下の成分は、配列 <code>tau</code> とともに基本リフレクタの積として直 交/ユニタリの行列 $Q$ を表現する (次の「アプリケーション・ノート」 を参照)。
<code>tau</code>	(ローカル) REAL (psgeqr2 の場合) DOUBLE PRECISION (pdgeqr2 の場合) COMPLEX (pcgeqr2 の場合) COMPLEX*16 (pzgeqr2 の場合)。 配列、次元は $LOC(ja+\min(m,n)-1)$ 。基本リフレクタのスカラ係数が 格納される。 <code>tau</code> は、分散行列 $A$ に関連付けられる。
<code>work</code>	終了時に、 <code>work(1)</code> は、最小かつ最適な <code>lwork</code> 値を返す。



*info* (ローカル) INTEGER。  
*info* = 0 の場合、正常に終了したことを示す。  
*info* < 0 の場合、*i* 番目の引数が配列で *j* 番目の成分の値が不正ならば *info* = - (*i*\*100+*j*)、*i* 番目の引数がスカラーで値が不正ならば *info* = -*i*。

### アプリケーション・ノート

行列  $Q$  は基本リフレクタの積として表現される。

$Q = H(ja) H(ja+1) \dots H(ja+k-1)$ 。ここで、 $k = \min(m, n)$ 。

それぞれの  $H(i)$  は次のような形式を持つ。

$$H(j) = I - \tau * v * v'$$

ここで、 $\tau$  は実 / 複素スカラー、 $v$  は実 / 複素ベクトルで  $v(1:i-1) = 0$  および  $v(i) = 1$ 。終了時に、 $v(i+1:m)$  は  $A(ia+i:ia+m-1, ja+i-1)$  に、 $\tau$  は  $TAU(ja+i-1)$  に格納される。

## p?gerq2

一般矩形行列の  $RQ$  因子分解を計算する ( 非ブ  
 ロック化アルゴリズム )。

### 構文

```
call psggerq2(m, n, a, ia, ja, desca, tau, work, lwork, info)
call psrgerq2(m, n, a, ia, ja, desca, tau, work, lwork, info)
call psdgerq2(m, n, a, ia, ja, desca, tau, work, lwork, info)
call psqgerq2(m, n, a, ia, ja, desca, tau, work, lwork, info)
```

### 説明

このルーチンは、 $m \times n$  の実 / 複素分布行列  $\text{sub}(A) = A(ia:ia+m-1, ja:ja+n-1) = R * Q$  の  $RQ$  因子分解を計算する。

### 入力パラメータ

*m* (グローバル) INTEGER。  
 演算を行う行数、すなわち、分散部分行列  $\text{sub}(A)$  の行数。( $m \geq 0$ )。

<i>n</i>	(グローバル) INTEGER。 演算を行う列数、すなわち、分散部分行列 <i>sub(A)</i> の列数。( $n \geq 0$ )。
<i>a</i>	(ローカル) REAL (psgerq2 の場合) DOUBLE PRECISION (pdgerq2 の場合) COMPLEX (pcgerq2 の場合) COMPLEX*16 (pzgerq2 の場合)。 ローカルメモリにある、次元 ( <i>lld_a</i> , <i>LOCc(ja+n-1)</i> ) の配列へのポインタ。因子分解する $m \times n$ の分散行列 <i>sub(A)</i> のローカル部分を格納する。
<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 <i>A</i> の最初の行と最初の列を示す、グローバル配列 <i>a</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。分散行列 <i>A</i> の配列ディスクリプタ。
<i>work</i>	(ローカル) REAL (psgerq2 の場合) DOUBLE PRECISION (pdgerq2 の場合) COMPLEX (pcgerq2 の場合) COMPLEX*16 (pzgerq2 の場合)。 次元 ( <i>lwork</i> ) のワークスペース配列。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 配列 <i>work</i> の次元。 <i>lwork</i> はローカル入力であり、 $lwork \geq nq0 + \max(1, mp0)$ でなければならない。ここで、  $iroff = \text{mod}(ia-1, mb\_a)$ 、 $icoff = \text{mod}(ja-1, nb\_a)$ 、 $iarow = \text{indxg2p}(ia, mb\_a, myrow, rsrc\_a, nprow)$ 、 $iacol = \text{indxg2p}(ja, nb\_a, mycol, csrc\_a, npc1)$ 、 $mp0 = \text{numroc}(m+iroff, mb\_a, myrow, iarow, nprow)$ 、 $nq0 = \text{numroc}(n+icoff, nb\_a, mycol, iacol, npc1)$ 。  <i>indxg2p</i> および <i>numroc</i> は、ScaLAPACK ツール関数である。 <i>myrow</i> 、 <i>mycol</i> 、 <i>nprow</i> 、および <i>npc1</i> は、サブルーチン <i>blacs_gridinfo</i> を呼び出して求められる。  <i>lwork</i> = -1 の場合、 <i>lwork</i> はグローバル入力であり、ワークスペース のクエリとみなされ、ルーチンはすべての配列の最小かつ最適なサイ ズだけを計算する。各値は該当する <i>work</i> 配列の最初のエントリとし て返され、 <a href="#">pxerbla</a> はエラー・メッセージを生成しない。

## 出力パラメータ

<i>a</i>	(ローカル) 終了時に、 $m \leq n$ の場合、 $A(ia+m-n:ia+m-1, ja:ja+n-1)$ の上三角部分は、 $m \times m$ の上三角行列 $R$ によって上書きされる。 $m \geq n$ の場合、 $(m-n)$ 番目の劣対角成分とその上の成分は、 $m \times n$ 上台形行列 $R$ によって上書きされる。残りの成分は、配列 $\tau$ とともに、基本リフレクタの積として直交/ユニタリ行列 $Q$ を表現する (次の「アプリケーション・ノート」を参照)。
<i>tau</i>	(ローカル) REAL (psgeqr2 の場合) DOUBLE PRECISION (pdgeqr2 の場合) COMPLEX (pcgeqr2 の場合) COMPLEX*16 (pzgeqr2 の場合)。 配列、次元は $LOCr(ia+m-1)$ 。基本リフレクタのスカラー係数が格納される。 $\tau$ は、分散行列 $A$ に関連付けられる。
<i>work</i>	終了時に、 $work(1)$ は、最小かつ最適な $lwork$ 値を返す。
<i>info</i>	(ローカル)。INTEGER。 $info = 0$ の場合、正常に終了したことを示す。 $info < 0$ の場合、 $i$ 番目の引数が配列で $j$ 番目の成分の値が不正ならば $info = -(i*100+j)$ 、 $i$ 番目の引数がスカラーで値が不正ならば $info = -i$ 。

## アプリケーション・ノート

行列  $Q$  は基本リフレクタの積として表現される。

$$Q = H(ia) H(ia+1) \dots H(ia+k-1) \text{ (実数型の場合)}$$

$$Q = H(ia)' H(ia+1)' \dots H(ia+k-1)' \text{ (複素数型の場合)}$$

ここで、 $k = \min(m, n)$ 。

それぞれの  $H(i)$  は次のような形式を持つ。

$$H(i) = I - \tau * v * v'$$

ここで、 $\tau$  は実/複素スカラー、 $v$  は実/複素ベクトルで  $v(n-k+i+1:n) = 0$  および  $v(n-k+i) = 1$ 。終了時に、 $v(1:n-k+i-1)$  (実数型の場合) または  $\text{conjg}(v(1:n-k+i-1))$  (複素数型の場合) は  $A(ia+m-k+i-1, ja:ja+n-k+i-2)$  に、 $\tau$  は  $TAU(ia+m-k+i-1)$  に格納される。

## p?getf2

一般行列の  $LU$  因子分解を、行交換を伴う部分ピボット演算を用いて計算する (ローカルブロック化アルゴリズム)。

### 構文

```
call psgetf2(m, n, a, ia, ja, desca, ipiv, info)
call pdgetf2(m, n, a, ia, ja, desca, ipiv, info)
call pcgetf2(m, n, a, ia, ja, desca, ipiv, info)
call pzgetf2(m, n, a, ia, ja, desca, ipiv, info)
```

### 説明

このルーチンは、行交換を伴う部分ピボット演算を用いて、 $m \times n$  の一般分散行列  $\text{sub}(A) = A(ia:ia+m-1, ja:ja+n-1)$  の  $LU$  因子分解を計算する。

因子分解は形式  $\text{sub}(A) = P * L * U$  を持ち、 $P$  は置換行列、 $L$  は単位対角成分を持つ下三角 ( $m > n$  の場合、下台形)、 $U$  は上三角 ( $m < n$  の場合、上台形) である。このルーチンは、アルゴリズムの right-looking 法により並列化されたレベル 2 BLAS バージョンである。

### 入カパラメータ

$m$	(グローバル) INTEGER。 演算を行う行数、すなわち、分散部分行列 $\text{sub}(A)$ の行数。( $m \geq 0$ )。
$n$	(グローバル) INTEGER。 演算を行う列数、すなわち、分散部分行列 $\text{sub}(A)$ の列数。 ( $nb\_a - \text{mod}(ja-1, nb\_a) \geq n \geq 0$ )。
$a$	(ローカル) REAL (psgetf2 の場合) DOUBLE PRECISION (pdgetf2 の場合) COMPLEX (pcgetf2 の場合) COMPLEX*16 (pzgetf2 の場合)。 ローカルメモリにある、次元 ( $lld\_a, LOCc(ja+n-1)$ ) の配列へのポインタ。 $m \times n$ の分散行列 $\text{sub}(A)$ のローカル部分を格納する。
$ia, ja$	(グローバル) INTEGER。それぞれ、部分行列 $\text{sub}(A)$ の最初の行と最初の列を示す、グローバル配列 $a$ の行インデックスと列インデックス。

*desca* (グローバルおよびローカル) INTEGER。配列、次元は (*dlen\_*)。分散行列 *A* の配列ディスクリプタ。

### 出力パラメータ

*ipiv* (ローカル) INTEGER。  
配列、次元は (*LOCr(m\_a) + mb\_a*)。この配列には、ピボット情報が格納される。*ipiv(i)* -> ローカル行 *i* と交換されたグローバル行。分散行列 *A* に関連付けられる。

*info* (ローカル) INTEGER。  
*info* = 0 の場合、正常に終了したことを示す。  
*info* < 0 の場合、

- *i* 番目の引数が配列で *j* 番目の成分の値が不正ならば  
*info* = - (*i*\*100+*j*)、
- *i* 番目の引数がスカラーで値が不正でならば *info* = -*i* であることを示す。

*info* > 0 の場合、*info* = *k* ならば *u(ia+k-1, ja+k-1)* は完全に 0 であることを示す。因子分解は完了したが、係数 *u* は完全に特異で、連立方程式の解の算出に *D* を使用すると 0 での除算が発生する。

## p?labrd

直交/ユニタリ変換を用いて、一般矩形行列 *A* の最初の *nb* 行と列を実数二重対角形式に縮退させ、*A* の縮退されていない部分に変換を適用するために必要となる補助行列を返す。

```
call pslabrd(m, n, nb, a, ia, ja, desca, d, e, tauq, taup, x, ix, jx, descx, y,
            iy, jy, descy, work)
call pdlabrd(m, n, nb, a, ia, ja, desca, d, e, tauq, taup, x, ix, jx, descx, y,
            iy, jy, descy, work)
call pclabrd(m, n, nb, a, ia, ja, desca, d, e, tauq, taup, x, ix, jx, descx, y,
            iy, jy, descy, work)
call pzlabrd(m, n, nb, a, ia, ja, desca, d, e, tauq, taup, x, ix, jx, descx, y,
            iy, jy, descy, work)
```

## 説明

このルーチンは、直交 / ユニタリ変換  $Q' * A * P$  を用いて  $m \times n$  の一般実 / 複素分布行列  $\text{sub}(A) = A(ia:ia+m-1, ja:ja+n-1)$  の最初の  $nb$  行と列を上または下二重対角形式に縮退させ、 $\text{sub}(A)$  の縮退されていない部分に変換を適用するために必要な行列  $X$  と  $Y$  を返す。

$m \geq n$  の場合、 $\text{sub}(A)$  は上二重対角形式に縮退される。

$m < n$  の場合、 $\text{sub}(A)$  は下二重対角形式に縮退される。

このルーチンは、[p?gebrd](#) から呼び出される補助ルーチンである。

## 入力パラメータ

$m$	(グローバル) INTEGER。 演算を行う行数、すなわち、分散部分行列 $\text{sub}(A)$ の行数。( $m \geq 0$ )。
$n$	(グローバル) INTEGER。 演算を行う列数、すなわち、分散部分行列 $\text{sub}(A)$ の列数。( $n \geq 0$ )。
$nb$	(グローバル) INTEGER。縮退の対象となる $\text{sub}(A)$ の先頭の行数と列数。
$a$	(ローカル) REAL (pslabrd の場合) DOUBLE PRECISION (pdlabrd の場合) COMPLEX (pclabrd の場合) COMPLEX*16 (pzlabrd の場合)。 ローカルメモリにある、次元 ( $lld\_a, LOCc(ja+n-1)$ ) の配列へのポインタ。一般分散行列 $\text{sub}(A)$ のローカル部分を格納する。
$ia, ja$	(グローバル) INTEGER。それぞれ、部分行列 $\text{sub}(A)$ の最初の行と最初の列を示す、グローバル配列 $a$ の行インデックスと列インデックス。
$desca$	(グローバルおよびローカル) INTEGER。配列、次元は ( $dlen\_$ )。分散行列 $A$ の配列ディスクリプタ。
$ix, jx$	(グローバル) INTEGER。それぞれ、部分行列 $\text{sub}(X)$ の最初の行と最初の列を示す、グローバル配列 $x$ の行インデックスと列インデックス。
$descx$	(グローバルおよびローカル) INTEGER。配列、次元は ( $dlen\_$ )。分散行列 $X$ の配列ディスクリプタ。
$iy, jy$	(グローバル) INTEGER。それぞれ、部分行列 $\text{sub}(Y)$ の最初の行と最初の列を示す、グローバル配列 $y$ の行インデックスと列インデックス。
$descy$	(グローバルおよびローカル) INTEGER。配列、次元は ( $dlen\_$ )。分散行列 $Y$ の配列ディスクリプタ。

*work* (ローカル)  
 REAL (pslabrd の場合)  
 DOUBLE PRECISION (pdlabrd の場合)  
 COMPLEX (pclabrd の場合)  
 COMPLEX\*16 (pzlabrd の場合)。  
 ワークスペース配列、次元は (*lwork*)。  
 $lwork \geq nb\_a + nq$ 、  
 $nq = \text{numroc}(n + \text{mod}(ia-1, nb\_y), nb\_y, mycol, iacol, npcol)$   
 $iacol = \text{indxg2p}(ja, nb\_a, mycol, csrc\_a, npcol)$ 。  
*indxg2p* および *numroc* は、ScaLAPACK ツール関数である。  
*myrow*, *mycol*, *nprow*, および *npcol* は、サブルーチン  
*blacs\_gridinfo* を呼び出して求められる。

## 出力パラメータ

*a* (ローカル)  
 終了時に、行列の最初の *nb* 行と列は上書きされる。分散行列 *sub(A)* の残りの部分は変更されない。  
 $m \geq n$  の場合、最初の *nb* 列の対角線上とその下の成分は、配列 *tauq* とともに、基本リフレクタの積として直交/ユニタリ行列 *Q* を表現する。最初の *nb* 行の対角よりも上の成分は、配列 *taup* とともに、基本リフレクタの積として直交/ユニタリ行列 *P* を表現する。  
 $m < n$  の場合、最初の *nb* 列の対角よりも下の成分は、配列 *tauq* とともに、基本リフレクタの積として直交/ユニタリ行列 *Q* を表現する。最初の *nb* 行の対角線上とその上の成分は、配列 *taup* とともに、基本リフレクタの積として直交/ユニタリ行列 *P* を表現する。次の「アプリケーション・ノート」を参照。

*e* (ローカル)  
 REAL (pslabrd の場合)  
 DOUBLE PRECISION (pdlabrd の場合)  
 COMPLEX (pclabrd の場合)  
 COMPLEX\*16 (pzlabrd の場合)。  
 配列、次元は  $LOCr(ia+\min(m,n)-1)$  ( $m \geq n$  の場合) または  $LOCc(ja+\min(m,n)-2)$  (それ以外の場合)。二重対角分散行列 *B* の分散非対角成分が格納される。  
 $m \geq n$  の場合、 $E(i) = A(ia+i-1, ja+i)$  ( $i = 1, 2, \dots, n-1$ )。  
 $m < n$  の場合、 $E(i) = A(ia+i, ja+i-1)$  ( $i = 1, 2, \dots, m-1$ )。  
*E* は、分散行列 *A* に関連付けられる。

<i>tauq,tau</i> <i>p</i>	(ローカル) REAL (pslabrd の場合) DOUBLE PRECISION (pdlabrd の場合) COMPLEX (pclabrd の場合) COMPLEX*16 (pzlabrd の場合)。 配列、 <i>tauq</i> の次元は $LOCc(ja+\min(m,n)-1)$ 、 <i>tau</i> <i>p</i> の次元は $LOCr(ia+\min(m,n)-1)$ 。直交 / ユニタリ行列 $Q$ ( <i>tauq</i> の場合)、 $P$ ( <i>tau</i> <i>p</i> の場合) を表す基本リフレクタのスカラー係数が格納される。 <i>tauq</i> と <i>tau</i> <i>p</i> は、分散行列 $A$ に関連付けられる。次の「アプリケーション・ノート」を参照。
<i>x</i>	(ローカル) REAL (pslabrd の場合) DOUBLE PRECISION (pdlabrd の場合) COMPLEX (pclabrd の場合) COMPLEX*16 (pzlabrd の場合)。 ローカルメモリにある、次元 ( <i>lld_x</i> , <i>nb</i> ) の配列へのポインタ。終了時に、 $\text{sub}(A)$ の縮退されていない部分の更新に必要な $m \times nb$ の分散行列 $X(ix:ix+m-1, jx:jx+nb-1)$ のローカル部分が格納される。
<i>y</i>	(ローカル)。 REAL (pslabrd の場合) DOUBLE PRECISION (pdlabrd の場合) COMPLEX (pclabrd の場合) COMPLEX*16 (pzlabrd の場合)。 ローカルメモリにある、次元 ( <i>lld_a</i> , <i>nb</i> ) の配列へのポインタ。終了時に、 $\text{sub}(A)$ の縮退されていない部分の更新に必要な $n \times nb$ の分散行列 $Y(iy:iy+n-1, jy:jy+nb-1)$ のローカル部分が格納される。

## アプリケーション・ノート

行列  $Q$  と  $P$  は基本リフレクタの積として表現される。

$$Q = H(1) H(2) \dots H(nb) \quad \text{および} \quad P = G(1) G(2) \dots G(nb)$$

それぞれの  $H(i)$  と  $G(i)$  は次のような形式を持つ。

$$H(i) = I - \tau_{auq} * v * v' \quad \text{および} \quad G(i) = I - \tau_{aup} * u * u'$$

ここで、*tauq* と *tau**p* は実 / 複素スカラ、*v* と *u* は実 / 複素ベクトルである。



$m \geq n$  の場合、終了時に、 $v(1:i-1) = 0$ 、 $v(i) = 1$ 、 $v(i:m)$  は  $A(ia+i-1:ia+m-1, ja+i-1)$  に格納される。終了時に、 $u(1:i) = 0$ 、 $u(i+1) = 1$ 、 $u(i+1:n)$  は  $A(ia+i-1, ja+i:ja+n-1)$  に格納される。 $\tau_{auq}$  は  $TAUQ(ja+i-1)$  に、 $\tau_{aup}$  は  $TAUP(ia+i-1)$  に格納される。

$m < n$  の場合、終了時に、 $v(1:i) = 0$ 、 $v(i+1) = 1$ 、 $v(i+1:m)$  は  $A(ia+i+1:ia+m-1, ja+i-1)$  に格納される。終了時に、 $u(1:i-1) = 0$ 、 $u(i) = 1$ 、 $u(i:n)$  は  $A(ia+i-1, ja+i:ja+n-1)$  に格納される。 $\tau_{auq}$  は  $TAUQ(ja+i-1)$  に、 $\tau_{aup}$  は  $TAUP(ia+i-1)$  に格納される。ベクトル  $v$  と  $u$  の成分は  $m \times nb$  の行列  $V$  と  $nb \times n$  の行列  $U'$  を形成する。行列  $\text{sub}(A)$  の縮退されていない部分に次の形式のブロック更新を使って変換を適用するために、これらの行列と行列  $X$  および  $Y$  が必要となる。 $\text{sub}(A) := \text{sub}(A) - V*Y' - X*U'$ 。終了時の  $\text{sub}(A)$  の内容を次に示す ( $nb=2$  の場合)。

$m = 6$ 、 $n = 5$  ( $m > n$ ) の場合：

$$\begin{bmatrix} 1 & 1 & u1 & u1 & u1 \\ v1 & 1 & 1 & u2 & u2 \\ v1 & v2 & a & a & a \\ v1 & v2 & a & a & a \\ v1 & v2 & a & a & a \\ v1 & v2 & a & a & a \end{bmatrix}$$

$m = 5$ 、 $n = 6$  ( $m < n$ ) の場合：

$$\begin{bmatrix} 1 & u1 & u1 & u1 & u1 & u1 \\ 1 & 1 & u2 & u2 & u2 & u2 \\ v1 & 1 & a & a & a & a \\ v1 & v2 & a & a & a & a \\ v1 & v2 & a & a & a & a \end{bmatrix}$$

$a$  は元の行列の変更されていない成分を意味し、 $v_i$  は  $H(i)$  を定義するベクトルの成分を意味し、 $u_i$  は  $G(i)$  を定義するベクトルの成分を意味する。

## p?lacon

行列-ベクトル積の評価にリバーズ・コミュニケーションを使用して正方向列の1-ノルムを推定する。

### 構文

```
call pslacon(n, v, iv, jv, descv, x, ix, jx, descx, isgn, est, kase)
```

```
call pdlacon(n, v, iv, jv, descv, x, ix, jx, descx, isgn, est, kase)
call pclacon(n, v, iv, jv, descv, x, ix, jx, descx, isgn, est, kase)
call pzlacon(n, v, iv, jv, descv, x, ix, jx, descx, isgn, est, kase)
```

## 説明

このルーチンは、実/ユニタリ分散行列  $A$  の 1- ノルムを推定する。行列 - ベクトル積の評価にはリバース・コミュニケーションが使用される。 $x$  と  $v$  は、分散行列  $A$  でアライメントされている。この情報は、 $iv$ 、 $ix$ 、 $descv$ 、および  $descx$  内に暗黙的に含まれる。

## 入カパラメータ

$n$	(グローバル) INTEGER。 乱数ベクトル $v$ と $x$ の長さ ( $n \geq 0$ )。
$v$	(ローカル) REAL (pslacon の場合) DOUBLE PRECISION (pdlacon の場合) COMPLEX (pclacon の場合) COMPLEX*16 (pzlacon の場合)。 ローカルメモリにある、次元 $LOCr(n+\text{mod}(iv-1, mb_v))$ の配列へのポインタ。最終戻りでは、 $v = a * w$ 。ここで、 $est = \text{norm}(v) / \text{norm}(w)$ ( $w$ は返されない)。
$iv, jv$	(グローバル) INTEGER。それぞれ、部分行列 $\text{sub}(V)$ の最初の行と最初の列を示す、グローバル配列 $v$ の行インデックスと列インデックス。
$descv$	(グローバルおよびローカル) INTEGER。配列、次元は ( $dlen\_$ )。分散行列 $V$ の配列ディスクリプタ。
$x$	(ローカル) REAL (pslacon の場合) DOUBLE PRECISION (pdlacon の場合) COMPLEX (pclacon の場合) COMPLEX*16 (pzlacon の場合)。 ローカルメモリにある、次元 $LOCr(n+\text{mod}(ix-1, mb_x))$ の配列へのポインタ。
$ix, jx$	(グローバル) INTEGER。それぞれ、部分行列 $X$ の最初の行と最初の列を示す、グローバル配列 $x$ の行インデックスと列インデックス。
$descx$	(グローバルおよびローカル) INTEGER。配列、次元は ( $dlen\_$ )。分散行列 $X$ の配列ディスクリプタ。

*isgn* (ローカル) INTEGER。  
配列、次元は  $LOCr(n+\text{mod}(ix-1, mb\_x))$ 。*isgn* は、*x* および *v* でアライメントされている。

*kase* (ローカル) INTEGER。  
*p?lacon* を初めて呼び出すときは *kase* はゼロでなければならない。

### 出力パラメータ

*x* (ローカル)  
中間戻りでは、*X* は次の値で上書きされる。  
 $A * X$  (*kase*=1 の場合)  
 $A' * X$  (*kase*=2 の場合)  
また、他のパラメータは変更しない状態で *p?lacon* を再度呼び出さなければならない。

*est* (グローバル)  
REAL (単精度の場合)  
DOUBLE PRECISION (倍精度の場合)。

*kase* (ローカル) INTEGER。  
中間戻りでは、*kase* は 1 か 2 となり、*X* が  $A * X$  または  $A' * X$  によって上書きされるかどうかを示す。*p?lacon* からの最終戻りでは、*kase* は再びゼロになる。

---

## p?laconsb

2 つの連続する小さい対角成分を検出する。

---

```
call pslaconsb(a, desca, i, l, m, h44, h33, h43h34, buf, lwork)
call pdlaconsb(a, desca, i, l, m, h44, h33, h43h34, buf, lwork)
```

### 説明

このルーチンは、*h44*、*h33*、および *h43h34* で与えられる倍精度シフト *QR* の反復法で開始する影響を分析し、このプロセスによって無視できる程度の劣対角成分が作成されるかどうか調べ、2 つの連続する小さい対角成分を検出する。

## 入力パラメータ

<i>a</i>	(グローバル) REAL (pslaconsb の場合) DOUBLE PRECISION (pdlaconsb の場合)。 配列、次元は ( <i>desca</i> ( <i>lld_</i> ),*)。三重対角部分をスキャンされた Hessenberg 行列を格納する。終了時に変更されない。
<i>descx</i>	(グローバルおよびローカル) INTEGER。 次元 ( <i>dlen_</i> ) の配列。分散行列 <i>A</i> の配列ディスクリプタ。
<i>i</i>	(グローバル) INTEGER。 <i>A</i> の縮退されていない部分行列の一番下のグローバル位置。終了時に 変更されない。
<i>l</i>	(グローバル) INTEGER。 <i>A</i> の縮退されていない部分行列の一番上のグローバル位置。終了時に 変更されない。
<i>h44</i> , <i>h33</i>	
<i>h43h34</i>	(グローバル) REAL (pslaconsb の場合) DOUBLE PRECISION (pdlaconsb の場合)。 これらの 3 種類の値は、倍精度シフト <i>QR</i> の反復法で使用される。
<i>lwork</i>	(グローバル) INTEGER。 この値は、 $7 * \text{ceil}(\text{ceil}((i-1)/hb1) / \text{lcm}(\text{nrow}, \text{npcol}))$ 以上でなければなら ない。lcm は公倍数以上で、 <i>nrow</i> <i>xnpcol</i> は論理グリッドサ イズである。

## 出力パラメータ

<i>m</i>	(グローバル) 終了時に、倍精度シフト <i>QR</i> の開始位置を格納する。次の条件を満た す。 $1 \leq m \leq i-2$ 。
<i>buf</i>	(ローカル) REAL (pslaconsb の場合) DOUBLE PRECISION (pdlaconsb の場合)。 サイズ <i>lwork</i> の配列。
<i>lwork</i>	(グローバル) 終了時に、 <i>lwork</i> は、ワークバッファのサイズである。

## p?lapc2

分散行列の一部または全部を他の分散行列にコピーする。

### 構文

```
call pslacp2(uplo, m, n, a, ia, ja, desca, b, ib, jb, descb)
call pdlacp2(uplo, m, n, a, ia, ja, desca, b, ib, jb, descb)
call pclacp2(uplo, m, n, a, ia, ja, desca, b, ib, jb, descb)
call pzlapc2(uplo, m, n, a, ia, ja, desca, b, ib, jb, descb)
```

### 説明

このルーチンは、分散行列  $A$  の一部または全部を他の分散行列  $B$  にコピーする。コミュニケーションが実行されない場合、p?lapc2 はローカルコピー  $\text{sub}(A) := \text{sub}(B)$  を実行する。ここで、 $\text{sub}(A)$  は  $A(ia:ia+m-1, ja:ja+n-1)$ 、 $\text{sub}(B)$  は  $B(ib:ib+m-1, jb:jb+n-1)$  である。

p?lapc2 は、行列オペランドの次元のみ再分配する。

### 入力パラメータ

uplo	(グローバル) CHARACTER。 分散行列 $\text{sub}(A)$ のコピーする部分を指定する。 'u' の場合、上三角部分をコピーする。 $\text{sub}(A)$ の厳密な下三角部分は参照されない。 'l' の場合、下三角部分をコピーする。 $\text{sub}(A)$ の厳密な上三角部は参照されない。 それ以外の場合、行列 $\text{sub}(A)$ の全部をコピーする。
m	(グローバル) INTEGER。 演算を行う行数、すなわち、分散部分行列 $\text{sub}(A)$ の行数。( $m \geq 0$ )。
n	(グローバル) INTEGER。 演算を行う列数、すなわち、分散部分行列 $\text{sub}(A)$ の列数。( $n \geq 0$ )。
a	(ローカル) REAL (pslapc2 の場合) DOUBLE PRECISION (pdlapc2 の場合) COMPLEX (pclacp2 の場合)

	COMPLEX*16 (pzlapc2 の場合)。 ローカルメモリにある、次元 ( $lld\_a$ , $LOCc(ja+n-1)$ ) の配列へのポインタ。 $m \times n$ の分散行列 $sub(A)$ のローカル部分を格納する。
$ia, ja$	(グローバル) INTEGER。それぞれ、 $sub(A)$ の最初の行と最初の列を示す、グローバル配列 $A$ の行インデックスと列インデックス。
$desca$	(グローバルおよびローカル) INTEGER。配列、次元は ( $dlen\_$ )。分散行列 $A$ の配列ディスクリプタ。
$ib, jb$	(グローバル) INTEGER。それぞれ、 $sub(B)$ の最初の行と最初の列を示す、グローバル配列 $B$ の行インデックスと列インデックス。
$descb$	(グローバルおよびローカル) INTEGER。配列、次元は ( $dlen\_$ )。分散行列 $B$ の配列ディスクリプタ。

## 出力パラメータ

$b$	(ローカル) REAL (pslapc2 の場合) DOUBLE PRECISION (pdlapc2 の場合) COMPLEX (pclapc2 の場合) COMPLEX*16 (pzlapc2 の場合)。 ローカルメモリにある、次元 ( $lld\_b$ , $LOCc(jb+n-1)$ ) の配列へのポインタ。終了時に、次のように設定された、分散行列 $sub(B)$ のローカル部分が格納される。  $uplo = 'U'$ の場合、 $B(ib+i-1, jb+j-1) = A(ia+i-1, ja+j-1)$ $(1 \leq i \leq j, 1 \leq j \leq n)$ 。 $uplo = 'L'$ の場合、 $B(ib+i-1, jb+j-1) = A(ia+i-1, ja+j-1)$ $(j < i \leq m, 1 \leq j \leq n)$ 。 それ以外の場合、 $B(ib+i-1, jb+j-1) = A(ia+i-1, ja+j-1)$ $(1 \leq i \leq m, 1 \leq j \leq n)$ 。
-----	---

---

## p?lapc3

グローバル並列配列からローカル複製配列にコピーする。逆の場合も同じ。

---

### 構文

```
call pslapc3(m, i, a, desca, b, ldb, ii, jj, rev)
```

```
call pdlaccp3(m, i, a, desca, b, ldb, ii, jj, rev)
```

## 説明

このルーチンは、グローバル並列配列からローカル複製配列にコピーしたり、逆にローカル複製配列からグローバル並列配列にコピーする補助ルーチンである。コピーされた部分行列全体は 1 つ以上の成分に配置される点に注意する。配置先成分は厳密に指定することができる。すべての成分、あるいは成分の 1 行または 1 列を指定することもできる。

## 入力パラメータ

- m* (グローバル) INTEGER。 *m* はコピーされた正方部分行列の次数。  
 $m \geq 0$ 。終了時に変更されない。
- i* (グローバル) INTEGER。  
 $A(i, i)$  は、コピーを開始するグローバル位置。終了時に変更されない。
- a* (グローバル)  
 REAL (pslaccp3 の場合)  
 DOUBLE PRECISION (pdlaccp3 の場合)。  
 配列、次元は (*desca*(*lld*),\*)。コピー先またはコピー元の並列行列。
- desca* (グローバルおよびローカル) INTEGER。配列、次元は (*dlen*)。分散行列 *A* の配列ディスクリプタ。
- b* (ローカル)  
 REAL (pslaccp3 の場合)  
 DOUBLE PRECISION (pdlaccp3 の場合)。  
 配列、次元は (*ldb*, *m*)。  
 $rev = 0$  の場合、配列  $A(i:i+m-1, i:i+m-1)$  のグローバル成分。  
 $rev = 1$  の場合、終了時に変更されない。
- ldb* (ローカル) INTEGER。  
*B* のリーディング・ディメンジョン。
- ii* (グローバル) INTEGER。  
 $rev = 0$  と 1 を使用すると、データを再度送信したり、返したりすることができる。 $rev = 0$  の場合、*ii* は複製 *B* を受け取る成分の行インデックスである。  
 $ii \geq 0, jj \geq 0$  の場合、成分 (*ii*, *jj*) がデータを受け取る。  
 $ii = -1, jj \geq 0$  の場合、列 *jj* のすべての行がデータを受け取る。

$ii \geq 0$ 、 $jj = -1$  の場合、行  $ii$  のすべての列がデータを受け取る。  
 $ii = -1$ 、 $jj = -1$  の場合、すべての成分がデータを受け取る。  
 $rev \neq 0$  の場合、 $ii$  は複製  $B$  を送信する成分の行インデックスである。

$jj$  (グローバル) INTEGER。上述の  $ii$  と同じ。

$rev$  (グローバル) INTEGER。  
 $rev = 0$  を使用して、グローバル  $A$  をローカル複製  $B$  (成分  $(ii, jj)$ ) に送る。  
 $rev \neq 0$  を使用して、成分  $(ii, jj)$  からローカル複製  $B$  をグローバル  $A$  内の所有成分に送る (所有成分は、 $A$  における成分の位置によって変更する)。

### 出力パラメータ

$a$  (グローバル)  $rev = 1$  の場合、終了時に、コピーされたデータを格納する。 $rev = 0$  の場合、終了時に変更されない。

$b$  (ローカル)  $rev = 1$  の場合、終了時に変更されない。

---

## p?lacpy

1 つの 2 次元配列の一部または全部を他にコピーする。

---

### 構文

```
call pslacpy(uplo, m, n, a, ia, ja, desca, b, ib, jb, descb)
call pdlacpy(uplo, m, n, a, ia, ja, desca, b, ib, jb, descb)
call pclacpy(uplo, m, n, a, ia, ja, desca, b, ib, jb, descb)
call pzlacpy(uplo, m, n, a, ia, ja, desca, b, ib, jb, descb)
```

### 説明

このルーチンは、分散行列  $A$  の一部または全部を他の分散行列  $B$  にコピーする。コミュニケーションが実行されない場合、p?lacpy はローカルコピー  $\text{sub}(A) := \text{sub}(B)$  を実行する。ここで、 $\text{sub}(A)$  は  $A(ia:ia+m-1, ja:ja+n-1)$ 、 $\text{sub}(B)$  は  $B(ib:ib+m-1, jb:jb+n-1)$  である。



## 入力パラメータ

<code>uplo</code>	(グローバル) CHARACTER。 分散行列 <code>sub(A)</code> のコピーする部分を指定する。 'u' の場合、上三角部分をコピーする。 <code>sub(A)</code> の厳密な下三角部分は参照されない。 'l' の場合、下三角部分をコピーする。 <code>sub(A)</code> の厳密な上三角部は参照されない。 それ以外の場合、行列 <code>sub(A)</code> の全部をコピーする。
<code>m</code>	(グローバル) INTEGER。 演算を行う行数、すなわち、分散部分行列 <code>sub(A)</code> の行数。( $m \geq 0$ )。
<code>n</code>	(グローバル) INTEGER。 演算を行う列数、すなわち、分散部分行列 <code>sub(A)</code> の列数。( $n \geq 0$ )。
<code>a</code>	(ローカル) REAL (pslapy の場合) DOUBLE PRECISION (pdlapy の場合) COMPLEX (pclapy の場合) COMPLEX*16 (pzlapy の場合)。 ローカルメモリにある、次元 ( <code>lld_a</code> , <code>LOCc(ja+n-1)</code> ) の配列へのポインタ。分散行列 <code>sub(A)</code> のローカル部分を格納する。
<code>ia,ja</code>	(グローバル) INTEGER。それぞれ、部分行列 <code>sub(A)</code> の最初の行と最初の列を示す、グローバル配列 <code>a</code> の行インデックスと列インデックス。
<code>desca</code>	(グローバルおよびローカル) INTEGER。配列、次元は ( <code>dlen_</code> )。分散行列 <code>A</code> の配列ディスクリプタ。
<code>ib,jb</code>	(グローバル) INTEGER。それぞれ、 <code>sub(B)</code> の最初の行と最初の列を示す、グローバル配列 <code>B</code> の行インデックスと列インデックス。
<code>descb</code>	(グローバルおよびローカル) INTEGER。配列、次元は ( <code>dlen_</code> )。分散行列 <code>A</code> の配列ディスクリプタ。

## 出力パラメータ

<code>b</code>	(ローカル) REAL (pslapy の場合) DOUBLE PRECISION (pdlapy の場合) COMPLEX (pclapy の場合) COMPLEX*16 (pzlapy の場合)。
----------------	--

ローカルメモリにある、次元 ( $lld\_b$ ,  $LOCc(jb+n-1)$ ) の配列へのポインタ。終了時に、次のように設定された、分散行列  $\text{sub}(\mathbf{B})$  のローカル部分が格納される。

$\text{uplo} = 'U'$  の場合、 $B(ib+i-1, jb+j-1) = A(ia+i-1, ja+j-1)$   
( $1 \leq i \leq j$ ,  $1 \leq j \leq n$ )。

$\text{uplo} = 'L'$  の場合、 $B(ib+i-1, jb+j-1) = A(ia+i-1, ja+j-1)$   
( $j \leq i \leq m$ ,  $1 \leq j \leq n$ )。

それ以外の場合、 $B(ib+i-1, jb+j-1) = A(ia+i-1, ja+j-1)$   
( $1 \leq i \leq m$ ,  $1 \leq j \leq n$ )。

---

### p?laevswp

計算された固有ベクトルを ScaLAPACK 標準  
ブロックサイクル配列に移動する。

---

#### 構文

```
call pslaevswp(n, zin, ldzi, z, iz, jz, descz, nvs, key, rwork,  
              lrwork)  
call pdlaevswp(n, zin, ldzi, z, iz, jz, descz, nvs, key, rwork,  
              lrwork)  
call pclaevswp(n, zin, ldzi, z, iz, jz, descz, nvs, key, rwork,  
              lrwork)  
call pzlaevswp(n, zin, ldzi, z, iz, jz, descz, nvs, key, rwork,  
              lrwork)
```

#### 説明

このルーチンは、( ソートされていない可能性のある ) 計算された固有ベクトルを ScaLAPACK 標準ブロックサイクル配列に移動する (ScaLAPACK はソートされているため、対応する固有ベクトルもソートされる)。

#### 入力パラメータ

$np$  = 与えられたプロセスに対するローカルの行数。

$nq$  = 与えられたプロセスに対するローカルの列数。

<i>n</i>	(グローバル) INTEGER。 行列 <i>A</i> の次数。 $n \geq 0$ 。
<i>zin</i>	(ローカル) REAL (pslaevswp の場合) DOUBLE PRECISION (pdlaevswp の場合) COMPLEX (pclaevswp の場合) COMPLEX*16 (pzlaevswp の場合)。 配列、次元は ( <i>ldzi</i> , <i>nvs(iam)</i> )。固有ベクトル。各固有ベクトルは、1 つのプロセスに完全に属する。各プロセスは、隣接した <i>nvs(iam)</i> 固有 ベクトルのセットを格納する。プロセスが格納する最初の固有ベクト ルは、 <i>nvs(i)</i> の $i = [0, iam-1)$ の合計。
<i>ldzi</i>	(ローカル) INTEGER。配列 <i>zin</i> のリーディング・ディメンジョン。
<i>iz, jz</i>	(グローバル) INTEGER。それぞれ、部分行列 <i>Z</i> の最初の行と最初の列 を示す、グローバル配列 <i>Z</i> の行インデックスと列インデックス。
<i>descz</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。分散行 列 <i>Z</i> の配列ディスクリプタ。
<i>nvs</i>	(グローバル) INTEGER。 配列、次元は ( <i>nprocs+1</i> )。 <i>nvs(i)</i> = プロセス $[0, i-1)$ の固有ベクトル数。 <i>nvs(1)</i> = プロセス $[0, 1-1)$ の固有ベクトル数 (= 0)。 <i>nvs(nprocs+1)</i> = プロセス $[0, nprocs)$ のベクトル数 (= 固有ベクトルの 総数)。
<i>key</i>	(グローバル) INTEGER。 配列、次元は ( <i>n</i> )。各固有ベクトルの (ソート後の) 実際のインデック スを指定する。
<i>rwork</i>	(ローカル) REAL (pslaevswp の場合) DOUBLE PRECISION (pdlaevswp の場合) COMPLEX (pclaevswp の場合) COMPLEX*16 (pzlaevswp の場合)。 配列、次元は ( <i>lrwork</i> )。
<i>lrwork</i>	(ローカル) INTEGER。 <i>work</i> の次元。

## 出力パラメータ

*z* (ローカル)  
REAL (pslaevswp の場合)  
DOUBLE PRECISION (pdlaevswp の場合)  
COMPLEX (pclaevswp の場合)  
COMPLEX\*16 (pzlaevswp の場合)。  
配列、グローバル次元は  $(n, n)$ 、ローカル次元は  $(descz(dlen\_), nq)$ 。  
固有ベクトル。固有ベクトルは、どちらの次元でもサイズ *nb* のブロックサイクル形式で配分される。

## p?lahrd

直交/ユニタリ相似変換を用いて、*k* 番目の劣対角よりも下の成分がゼロになるように一般矩形行列の最初の *nb* 列を縮退させ、*A* の縮退されていない部分に変換を適用するために必要となる補助行列を返す。

### 構文

```
call pslahrd(n, k, nb, a, ia, ja, desca, tau, t, y, iy, jy, descy, work)
call pdlahrd(n, k, nb, a, ia, ja, desca, tau, t, y, iy, jy, descy, work)
call pclahrd(n, k, nb, a, ia, ja, desca, tau, t, y, iy, jy, descy, work)
call pzlahrd(n, k, nb, a, ia, ja, desca, tau, t, y, iy, jy, descy, work)
```

### 説明

このルーチンは、*k* 番目の劣対角よりも下の成分がゼロになるように、 $n \times (n-k+1)$  の一般実数分布行列  $A(ia:ia+n-1, ja:ja+n-k)$  の最初の *nb* 列を縮退させる。この縮退は直交/ユニタリ相似変換  $Q' * A * Q$  によって実行される。このルーチンは、ブロック・リフレクタ  $I - V * T * V'$  として  $Q$  を定義する行列  $V$  と  $T$ 、および行列  $Y = A * V * T$  を返す。

このルーチンは、[p?gehrd](#) から呼び出される補助ルーチンである。次の入力/出力パラメータの説明では、 $\text{sub}(A)$  は  $A(ia:ia+n-1, ja:ja+n-1)$  を表す。

### 入力パラメータ

*n* (グローバル) INTEGER。分散部分行列  $\text{sub}(A)$  の次数。( $n \geq 0$ )。

<i>k</i>	(グローバル) INTEGER。縮退のオフセット。最初の <i>nb</i> 列にある <i>k</i> 番目の劣対角よりも下の成分がゼロに縮退される。
<i>nb</i>	(グローバル) INTEGER。縮退させる列数。
<i>a</i>	(ローカル) REAL (pslahrd の場合) DOUBLE PRECISION (pdlahrd の場合) COMPLEX (pclahrd の場合) COMPLEX*16 (pzlahrd の場合)。ローカルメモリにある、次元 ( <i>lld_a</i> , <i>LOCc</i> ( <i>ja+n-k</i> )) の配列へのポインタ。 $n \times (n-k+1)$ の一般分散行列 $A(\text{ia:ia}+n-1, \text{ja:ja}+n-k)$ のローカル部分を格納する。
<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 $\text{sub}(A)$ の最初の行と最初の列を示す、グローバル配列 $A$ の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。分散行列 $A$ の配列ディスクリプタ。
<i>iy, jy</i>	(グローバル) INTEGER。それぞれ、部分行列 $\text{sub}(Y)$ の最初の行と最初の列を示す、グローバル配列 $Y$ の行インデックスと列インデックス。
<i>descy</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。分散行列 $Y$ の配列ディスクリプタ。
<i>work</i>	(ローカル) REAL (pslahrd の場合) DOUBLE PRECISION (pdlahrd の場合) COMPLEX (pclahrd の場合) COMPLEX*16 (pzlahrd の場合)。 配列、次元は ( <i>nb</i> )。

## 出力パラメータ

<i>a</i>	(ローカル) 終了時に、最初の <i>nb</i> 列にある <i>k</i> 番目の劣対角成分とその上の成分は、縮退された分散行列の対応する成分で上書きされる。 <i>k</i> 番目の劣対角よりも下の成分は、配列 <i>tau</i> とともに、基本リフレクタの積として行列 $Q$ を表現する。 $A(\text{ia:ia}+n-1, \text{ja:ja}+n-k)$ のそのほかの列は変更されない。次の「アプリケーション・ノート」を参照。
<i>tau</i>	(ローカル) REAL (pslahrd の場合) DOUBLE PRECISION (pdlahrd の場合) COMPLEX (pclahrd の場合)

COMPLEX\*16 (pzlahrd の場合)。  
 配列、次元は  $LOCc(ja+n-2)$ 。  
 基本リフレクタのスカラ係数 (次の「アプリケーション・ノート」を参照)。 $\tau$  は、分散行列  $A$  に関連付けられる。

$t$  (ローカル)  
 REAL (pslahrd の場合)  
 DOUBLE PRECISION (pdlahrd の場合)  
 COMPLEX (pclahrd の場合)  
 COMPLEX\*16 (pzlahrd の場合)。  
 配列、次元は  $(nb\_a, nb\_a)$ 。  
 上三角行列  $T$ 。

$y$  (ローカル)  
 REAL (pslahrd の場合)  
 DOUBLE PRECISION (pdlahrd の場合)  
 COMPLEX (pclahrd の場合)  
 COMPLEX\*16 (pzlahrd の場合)。  
 ローカルメモリにある、次元  $(lld\_y, nb\_a)$  の配列へのポインタ。終了時に、 $n \times nb$  の分散行列  $Y$  のローカル部分が格納される。  
 $lld\_y \geq LOCr(ia+n-1)$ 。

## アプリケーション・ノート

行列  $Q$  は基本リフレクタ  $nb$  の積として表現される。

$$Q = H(1) H(2) \dots H(nb)$$

それぞれの  $H(i)$  は次のような形式を持つ。

$$H(i) = I - \tau v v'$$

ここで、 $\tau$  は実 / 複素スカラ、 $v$  は実 / 複素ベクトルで  $v(1:i+k-1) = 0$  および  $v(i+k) = 1$ 。  
 終了時に、 $v(i+k+1:n)$  は  $A(ia+i+k:ia+n-1, ja+i-1)$  に、 $\tau$  は  $TAU(ja+i-1)$  に格納される。

ベクトル  $v$  の成分は、行列の縮退されていない部分に変換を適用するために、次の形式の更新を用いて、 $T$  と  $Y$  とともに必要な  $(n-k+1) \times nb$  の行列  $V$  を形成する。

$A(ia:ia+n-1, ja:ja+n-k) := (I - V * T * V') * (A(ia:ia+n-1, ja:ja+n-k) - Y * V')$ 。終了時の  $A(ia:ia+n-1, ja:ja+n-k)$  の内容を次に示す ( $n=7$ 、 $k=3$ 、および  $nb=2$  の場合)。

$$\begin{bmatrix} a & h & a & a & a \\ a & h & a & a & a \\ a & h & a & a & a \\ h & h & a & a & a \\ v1 & h & a & a & a \\ v1 & v2 & a & a & a \\ v1 & v2 & a & a & a \end{bmatrix}$$

$a$  は元の行列  $A(ia:ia+n-1, ja:ja+n-k)$  の成分、 $h$  は上 Hessenberg 行列  $H$  の変更された成分、 $vi$  は  $H(i)$  を定義するベクトルの成分を示す。

## p?laiect

IEEE 演算を利用して固有値の計算を加速させる  
(C インターフェイス関数)。

### 構文

```
void pslaiect(float *sigma, int *n, float *d, int *count);
void pdlaiectb(float *sigma, int *n, float *d, int *count);
void pdlaiectl(float *sigma, int *n, float *d, int *count);
```

### 説明

このルーチンは、 $(A - \sigma I)$  の負の固有値の個数を計算する。Sturm シーケンス・ループは、IEEE 演算を利用し、最内ループに条件を持たない。ビット 32 を実数ルーチン pslaiect の符号ビットとみなす。倍精度ルーチン pdlaiectb と pdlaiectl では、倍精度 WORD 型の格納順序が異なる。そのため、符号ビットの位置も異なる。pdlaiectb では、倍精度 WORD 型をビッグ・エンディアンで格納し、符号ビットはビット 32 である。pdlaiectl では、倍精度 WORD 型をリトル・エンディアンで格納し、符号ビットはビット 64 である。

すべての引数は参照による呼び出しであるため、このルーチンは Fortran コードから直接呼び出すことができる。

これは ScaLAPACK の内部サブルーチンであり、引数の妥当性は確認されない。

## 入力パラメータ

<i>sigma</i>	REAL (pslaiect の場合 ) DOUBLE PRECISION (pdlaiectb/pdlaiectl の場合 )。 シフト。p?laiect は、 <i>sigma</i> よりも小さいか等しい固有値を検出する。
<i>n</i>	INTEGER。 三重対角行列 <i>T</i> の次数。 $n \geq 1$ 。
<i>d</i>	REAL (pslaiect の場合 ) DOUBLE PRECISION (pdlaiectb/pdlaiectl の場合 )。 次元 $(2n - 1)$ の配列。 三重対角行列 <i>T</i> の非対角成分の平方および非対角成分を格納する。高いキャッシュ・パフォーマンスを得るために、これらの成分はメモリでインターリーブされるものみなす。 <i>T</i> の対角成分は $d(1), d(3), \dots, d(2n-1)$ 、非対角成分の平方は $d(2), d(4), \dots, d(2n-2)$ とする。オーバーフローを避けるためには、行列を、その最大の成分が $\text{overflow}^{(1/2)} * \text{underflow}^{(1/4)}$ よりも絶対値において超えないようにスケールしなければならない。また、最高の精度を得るためには、行列を上記条件よりもはるかに小さくスケールしてはならない。

## 出力パラメータ

<i>n</i>	INTEGER。 <i>sigma</i> よりも小さいか等しい <i>T</i> の固有値の総数。
----------	---

---

## p?lange

一般矩形行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。

---

### 構文

```
val = pslange(norm, m, n, a, ia, ja, desca, work)
val = pdlange(norm, m, n, a, ia, ja, desca, work)
val = pclange(norm, m, n, a, ia, ja, desca, work)
val = pzlange(norm, m, n, a, ia, ja, desca, work)
```



## 説明

この関数は、分散行列  $\text{sub}(A) = A(\text{ia}:\text{ia}+m-1, \text{ja}:\text{ja}+n-1)$  の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。

`p?lange` は次の値を返す。

(  $\max(\text{abs}(A(i,j)))$  ), `norm` = 'M' または 'm',  $\text{ia} \leq i \leq \text{ia}+m-1$  および  $\text{ja} \leq j \leq \text{ja}+n-1$

(

(  $\text{norm1}(\text{sub}(A))$  ), `norm` = '1', 'o' または 'o'

(

(  $\text{normI}(\text{sub}(A))$  ), `norm` = 'I' または 'i'

(

(  $\text{normF}(\text{sub}(A))$  ), `norm` = 'F', 'f', 'E' または 'e'

$\text{norm1}$  は行列の 1- ノルム ( 最大列合計 ),  $\text{normI}$  は行列の無限ノルム ( 最大行合計 ),  $\text{normF}$  は行列の Frobenius ノルム ( 二乗和の平方根 ) を示す。 $\max(\text{abs}(A(i,j)))$  は行列ノルムではない点に注意する。

## 入力パラメータ

- `norm` ( グローバル ) CHARACTER。  
ルーチン `p?lange` が返す値を上述のように指定する。
- `m` ( グローバル ) INTEGER。  
演算を行う行数、すなわち、分散部分行列  $\text{sub}(A)$  の行数。  $m = 0$  と指定した場合、`p?lange` はゼロに設定される。  $m \geq 0$ 。
- `n` ( グローバル ) INTEGER。  
演算を行う列数、すなわち、分散部分行列  $\text{sub}(A)$  の列数。  $n = 0$  と指定した場合、`p?lange` はゼロに設定される。  $n \geq 0$ 。
- `a` ( ローカル )  
REAL (`pslange` の場合 )  
DOUBLE PRECISION (`pdlange` の場合 )  
COMPLEX (`pclange` の場合 )  
COMPLEX\*16 (`pzlange` の場合 )。  
ローカルメモリにある、次元 ( $\text{ld\_a}$ ,  $\text{LOCc}(\text{ja}+n-1)$ ) の配列。  
分散行列  $\text{sub}(A)$  のローカル部分を格納する。

*ia, ja* (グローバル) INTEGER。それぞれ、部分行列 *sub(A)* の最初の行と最初の列を示す、グローバル配列 *A* の行インデックスと列インデックス。

*desca* (グローバルおよびローカル) INTEGER。配列、次元は (*dlen\_*)。分散行列 *A* の配列ディスクリプタ。

*work* (ローカル)  
REAL (*pslange* の場合)  
DOUBLE PRECISION (*pdlange* の場合)  
COMPLEX (*pclange* の場合)  
COMPLEX\*16 (*pzlange* の場合)。

配列、次元は (*lwork*)。

*norm* = 'M' または 'm' の場合 (参照されない場合)、*lwork* ≥ 0。

*norm* = 'l'、'o' または 'o' の場合、*nq*0。

*norm* = 't' または 'i' の場合、*mp*0。

*norm* = 'F'、'f'、'E' または 'e' の場合 (参照されない場合)、0。

ここで、

*iroffa* = mod(*ia*-1, *mb\_a*)、*icoffa* = mod(*ja*-1, *nb\_a*)、

*iarow* = indxg2p(*ia*, *mb\_a*, *myrow*, *rsrc\_a*, *nprow*)、

*iacol* = indxg2p(*ja*, *nb\_a*, *mycol*, *csrc\_a*, *npcol*)、

*mp*0 = numroc(*m*+*iroffa*, *mb\_a*, *myrow*, *iarow*, *nprow*)、

*nq*0 = numroc(*n*+*icoffa*, *nb\_a*, *mycol*, *iacol*, *npcol*)。

indxg2p および numroc は、ScaLAPACK ツール関数である。*myrow*、*mycol*、*nprow*、および *npcol* は、サブルーチン *lacs\_gridinfo* を呼び出して求められる。

### 出力パラメータ

*val* 関数の戻り値。

---

## p?lanhs

上 Hessenberg 行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。

---

### 構文

*val* = pslanhs(*norm*, *n*, *a*, *ia*, *ja*, *desca*, *work*)

```

val = pdlanhs(norm, n, a, ia, ja, desca, work)
val = pclanhs(norm, n, a, ia, ja, desca, work)
val = pzlanhs(norm, n, a, ia, ja, desca, work)

```

## 説明

この関数は、分散行列  $\text{sub}(A) = A(ia:ia+m-1, ja:ja+n-1)$  の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。

p?lange は次の値を返す。

```

(max(abs(A(i,j))), norm='M' または 'm', ia ≤ i ≤ ia+m-1 および ja ≤ j ≤ ja+n-1
(
( norm1(sub(A)), norm='l', 'o' または 'o'
(
( normI(sub(A)), norm='I' または 'i'
(
( normF(sub(A)), norm='F', 'f', 'E' または 'e'

```

$\text{norm}_1$  は行列の 1- ノルム ( 最大列合計 ),  $\text{norm}_I$  は行列の無限ノルム ( 最大行合計 ),  $\text{norm}_F$  は行列の Frobenius ノルム ( 二乗和の平方根 ) を示す。 $\max(\text{abs}(A(i,j)))$  は行列ノルムではない点に注意する。

## 入力パラメータ

**norm** (グローバル) CHARACTER。  
ルーチン p?lange が返す値を上述のように指定する。

**n** (グローバル) INTEGER。  
演算を行う列数、すなわち、分散部分行列  $\text{sub}(A)$  の列数。 $n=0$  と指定した場合、p?lanhs はゼロに設定される。 $n \geq 0$ 。

**a** (ローカル)  
REAL (pslanhs の場合)  
DOUBLE PRECISION (pdlanhs の場合)  
COMPLEX (pclanhs の場合)  
COMPLEX\*16 (pzlanhs の場合)。  
ローカルメモリにある、次元 ( $lld\_a, LOCc(ja+n-1)$ ) の配列。  
分散行列  $\text{sub}(A)$  ローカル部分を格納する。

*ia, ja* (グローバル) INTEGER。それぞれ、部分行列 *sub(A)* の最初の行と最初の列を示す、グローバル配列 *A* の行インデックスと列インデックス。

*desca* (グローバルおよびローカル) INTEGER。配列、次元は (*dlen\_*)。分散行列 *A* の配列ディスクリプタ。

*work* (ローカル)  
 REAL (*pslanhs* の場合)  
 DOUBLE PRECISION (*pdlanhs* の場合)  
 COMPLEX (*pclanhs* の場合)  
 COMPLEX\*16 (*pzlanh* の場合)。  
 配列、次元は (*lwork*)。  
*norm* = 'M' または 'm' の場合 (参照されない場合)、*lwork* ≥ 0。  
*norm* = 'l', 'o' または 'o' の場合、*nq0*。  
*norm* = 'I' または 'i' の場合、*mp0*。  
*norm* = 'F', 'f', 'E' または 'e' の場合 (参照されない場合)、0。  
 ここで、  
*iroffa* = mod(*ia*-1, *mb\_a*)、*icoffa* = mod(*ja*-1, *nb\_a*)、  
*iarow* = indxcg2p(*ia*, *mb\_a*, *myrow*, *rsrc\_a*, *nprow*)、  
*iacol* = indxcg2p(*ja*, *nb\_a*, *mycol*, *csrc\_a*, *npcol*)、  
*mp0* = numroc(*m*+*iroffa*, *mb\_a*, *myrow*, *iarow*, *nprow*)、  
*nq0* = numroc(*n*+*icoffa*, *nb\_a*, *mycol*, *iacol*, *npcol*)。  
*indxcg2p* および *numroc* は、ScaLAPACK ツール関数である。*myrow*、  
*mycol*、*nprow*、および *npcol* は、サブルーチン *lacs\_gridinfo* を呼び出して求められる。

## 出力パラメータ

*val* 関数の戻り値。

---

## p?lansy, p?lanhe

実対称行列または複素エルミート行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。

---

## 構文

```
val = pslansy(norm, uplo, n, a, ia, ja, desca, work)
val = pdlansy(norm, uplo, n, a, ia, ja, desca, work)
```

```

val = pclansy(norm, uplo, n, a, ia, ja, desca, work)
val = pzlansy(norm, uplo, n, a, ia, ja, desca, work)
val = pclarhe(norm, uplo, n, a, ia, ja, desca, work)
val = pzlarhe(norm, uplo, n, a, ia, ja, desca, work)

```

## 説明

この関数は、分散行列  $\text{sub}(A) = A(ia:ia+m-1, ja:ja+n-1)$  の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。

p?lansy、p?larhe は次の値を返す。

( $\max(\text{abs}(A(i,j)))$ )、 $\text{norm} = 'M'$  または  $'m'$ 、 $ia \leq i \leq ia+m-1$  および  $ja \leq j \leq ja+n-1$

(

( $\text{norm}_1(\text{sub}(A))$ )、 $\text{norm} = '1'$ 、 $'o'$  または  $'o'$

(

( $\text{norm}_I(\text{sub}(A))$ )、 $\text{norm} = 'I'$  または  $'i'$

(

( $\text{norm}_F(\text{sub}(A))$ )、 $\text{norm} = 'F'$ 、 $'f'$ 、 $'E'$  または  $'e'$

$\text{norm}_1$  は行列の 1- ノルム ( 最大列合計 )、 $\text{norm}_I$  は行列の無限ノルム ( 最大行合計 )、 $\text{norm}_F$  は行列の Frobenius ノルム ( 二乗和の平方根 ) を示す。 $\max(\text{abs}(A(i,j)))$  は行列ノルムではない点に注意する。

## 入力パラメータ

*norm* (グローバル) CHARACTER。  
ルーチン [p?lange](#) が返す値を上述のように指定する。

*uplo* (グローバル) CHARACTER。  
対称行列  $\text{sub}(A)$  の上三角部分と下三角部分のどちらを参照させるかを指定する。  
'U' の場合、 $\text{sub}(A)$  の上三角部分を参照させる。  
'L' の場合、 $\text{sub}(A)$  の下三角部分を参照させる。

*n* (グローバル) INTEGER。  
演算を行う列数、すなわち、分散部分行列  $\text{sub}(A)$  の列数。 $n = 0$  と指定した場合、p?lansy はゼロに設定される。 $n \geq 0$ 。

<i>a</i>	<p>(ローカル)</p> <p>REAL (pslansy の場合)</p> <p>DOUBLE PRECISION (pdlansy の場合)</p> <p>COMPLEX (pclansy、pclanhe の場合)</p> <p>COMPLEX*16 (pzlansy、pzlanhe の場合)。</p> <p>ローカルメモリにある、次元 (<i>lld_a</i>, <i>LOCc</i>(<i>ja+n-1</i>)) の配列。</p> <p>分散行列 <i>sub(A)</i> のローカル部分を格納する。</p> <p><i>uplo</i> = 'U' の場合、<i>sub(A)</i> の先頭の <math>n \times n</math> の上三角部分に、ノルムを計算する上三角行列を格納する。この行列の厳密な下三角部分は参照されない。</p> <p><i>uplo</i> = 'L' の場合、<i>sub(A)</i> の先頭の <math>n \times n</math> の下三角部分に、ノルムを計算する下三角行列を格納する。<i>sub(A)</i> の厳密な上三角部分は参照されない。</p>
<i>ia, ja</i>	<p>(グローバル) INTEGER。それぞれ、部分行列 <i>sub(A)</i> の最初の行と最初の列を示す、グローバル配列 <i>A</i> の行インデックスと列インデックス。</p>
<i>desca</i>	<p>(グローバルおよびローカル) INTEGER。配列、次元は (<i>dlen_</i>)。分散行列 <i>A</i> の配列ディスクリプタ。</p>
<i>work</i>	<p>(ローカル)</p> <p>REAL (pslansy の場合)</p> <p>DOUBLE PRECISION (pdlansy の場合)</p> <p>COMPLEX (pclansy、pclanhe の場合)</p> <p>COMPLEX*16 (pzlansy、pzlanhe の場合)。</p> <p>配列、次元は (<i>lwork</i>)。</p> <p><i>norm</i> = 'M' または 'm' の場合 (参照されない場合)、<i>lwork</i> <math>\geq 0</math>。</p> <p><i>norm</i> = 'L'、'O' または 'o'、'I' または 'i' の場合、<math>2 * nq0 + np0 + ldw</math>。</p> <p><i>ldw</i> は次のように与えられる。</p> <pre>         if( nprow.ne.npcol ) then             ldw = mb_a*ceil(ceil(np0/mb_a)/(lcm/nprow))         else             ldw = 0         end if     </pre> <p><i>norm</i> = 'F'、'f'、'E' または 'e' の場合 (参照されない場合)、0。</p> <p><i>lcm</i> は <i>nprow</i> と <i>npcol</i> の最小公倍数である。</p> <p><i>lcm</i> = <i>ilcm</i>(<i>nprow</i>, <i>npcol</i>)、<i>ceil</i> は <i>ceiling</i> 演算 (<i>iceil</i>)。</p> <p><i>iroffa</i> = <i>mod</i>(<i>ia-1</i>, <i>mb_a</i>)、<i>icoffa</i> = <i>mod</i>(<i>ja-1</i>, <i>nb_a</i>)、</p> <p><i>iarow</i> = <i>indxg2p</i>(<i>ia</i>, <i>mb_a</i>, <i>myrow</i>, <i>rsrc_a</i>, <i>nprow</i>)、</p> <p><i>iacol</i> = <i>indxg2p</i>(<i>ja</i>, <i>nb_a</i>, <i>mycol</i>, <i>csrc_a</i>, <i>npcol</i>)、</p> <p><i>mp0</i> = <i>numroc</i>(<i>m+iroffa</i>, <i>mb_a</i>, <i>myrow</i>, <i>iarow</i>, <i>nprow</i>)、</p>

`ng0 = numroc(n+icoffa, nb_a, mycol, iacol, npcol)`。  
`indxg2p` および `numroc` は、ScaLAPACK ツール関数である。`myrow`、  
`mycol`、`nprow`、および `npcol` は、サブルーチン `blacs_gridinfo` を  
呼び出して求められる。

### 出力パラメータ

`val`                      関数の戻り値。

## p?lantr

三角行列の 1- ノルムの値、Frobenius ノルムの値、  
無限ノルムの値、あるいは最大絶対値の成分を返  
す。

### 構文

```
val = pslantr(norm, uplo, diag, m, n, a, ia, ja, desca, work)
val = pdlantr(norm, uplo, diag, m, n, a, ia, ja, desca, work)
val = pclantr(norm, uplo, diag, m, n, a, ia, ja, desca, work)
val = pzlantr(norm, uplo, diag, m, n, a, ia, ja, desca, work)
```

### 説明

この関数は、台形または三角分散行列  $\text{sub}(A) = A(ia:ia+m-1, ja:ja+n-1)$  の 1- ノルムの  
値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。

`p?lantr` は次の値を返す。

```
( max(abs(A(i, j))), norm = 'M' または 'm', ia ≤ i ≤ ia+m-1 および ja ≤ j ≤ ja+n-1,
(
( norm1(sub(A)), norm = '1', 'o' または 'o'
(
( normI(sub(A)), norm = 'I' または 'i'
(
( normF(sub(A)), norm = 'F', 'f', 'E' または 'e'
```

*norm1* は行列の 1- ノルム ( 最大列合計 )、*normI* は行列の無限ノルム ( 最大行合計 )、*normF* は行列の Frobenius ノルム ( 二乗和の平方根 ) を示す。 $\max(\text{abs}(A(i,j)))$  は行列ノルムではない点に注意する。

## 入力パラメータ

<i>norm</i>	( グローバル ) CHARACTER。 ルーチン <i>p?lantr</i> が返す値を上述のように指定する。
<i>uplo</i>	( グローバル ) CHARACTER。 対称行列 <i>sub(A)</i> の上三角部分と下三角部分のどちらを参照させるかを指定する。  'U' の場合、上台形。 'L' の場合、下台形。  $m = n$ の場合、 <i>sub(A)</i> は台形ではなく三角であることに注意する。
<i>diag</i>	( グローバル ) CHARACTER。 分散行列 <i>sub(A)</i> が単位対角を持っているかどうかを指定する。 'N' の場合、単位対角ではない。 'U' の場合、単位対角。
<i>m</i>	( グローバル ) INTEGER。 演算を行う行数、すなわち、分散部分行列 <i>sub(A)</i> の行数。 $m = 0$ と指定した場合、 <i>p?lantr</i> はゼロに設定される。 $m \geq 0$ 。
<i>n</i>	( グローバル ) INTEGER。 演算を行う列数、すなわち、分散部分行列 <i>sub(A)</i> の列数。 $n = 0$ と指定した場合、 <i>p?lantr</i> はゼロに設定される。 $n \geq 0$ 。
<i>a</i>	( ローカル ) REAL ( <i>pslantr</i> の場合 ) DOUBLE PRECISION ( <i>pdlantr</i> の場合 ) COMPLEX ( <i>pclantr</i> の場合 ) COMPLEX*16 ( <i>pzlantr</i> の場合 )。 ローカルメモリにある、次元 ( <i>lld_a</i> , <i>LOCc(ja+n-1)</i> ) の配列。 分散行列 <i>sub(A)</i> のローカル部分を格納する。
<i>ia, ja</i>	( グローバル ) INTEGER。それぞれ、部分行列 <i>sub(A)</i> の最初の行と最初の列を示す、グローバル配列 <i>a</i> の行インデックスと列インデックス。
<i>desca</i>	( グローバルおよびローカル ) INTEGER。配列、次元は ( <i>dlen_</i> )。分散行列 <i>A</i> の配列ディスクリプタ。



*work* (ローカル)  
 REAL (pslantr の場合)  
 DOUBLE PRECISION (pdlantr の場合)  
 COMPLEX (pclantr の場合)  
 COMPLEX\*16 (pzlantr の場合)。  
 配列、次元は (*lwork*)。  
*norm* = 'M' または 'm' の場合 (参照されない場合)、*lwork* ≥ 0。  
*norm* = 'l', 'o' または 'o' の場合、*nq0*。  
*norm* = 't' または 'i' の場合、*mp0*  
*norm* = 'F', 'f', 'E' または 'e' の場合 (参照されない場合)、0。  
*lcm* は *nprow* と *npcol* の最小公倍数である。  
*lcm* = *ilcm*(*nprow*, *npcol*)、*ceil* は ceiling 演算 (*iceil*)。  
*iroffa* = *mod*(*ia*-1, *mb\_a*)、*icoffa* = *mod*(*ja*-1, *nb\_a*)、  
*iarow* = *indxg2p*(*ia*, *mb\_a*, *myrow*, *rsrc\_a*, *nprow*)、  
*iacol* = *indxg2p*(*ja*, *nb\_a*, *mycol*, *csrc\_a*, *npcol*)、  
*mp0* = *numroc*(*m*+*iroffa*, *mb\_a*, *myrow*, *iarow*, *nprow*)、  
*nq0* = *numroc*(*n*+*icoffa*, *nb\_a*, *mycol*, *iacol*, *npcol*)。  
*indxg2p* および *numroc* は、ScaLAPACK ツール関数である。*myrow*、  
*mycol*、*nprow*、および *npcol* は、サブルーチン *blacs\_gridinfo* を  
 呼び出して求められる。

## 出力パラメータ

*val* 関数の戻り値。

## p?lapiv

一般分散行列に置換行列を適用し、行または列の  
 ピボット演算を行う。

### 構文

```
call pslapiv(direc, rowcol, pivroc, m, n, a, ia, ja, desca, ipiv, ip, jp,
            descip, iwork)
call pdlapiv(direc, rowcol, pivroc, m, n, a, ia, ja, desca, ipiv, ip, jp,
            descip, iwork)
call pclapiv(direc, rowcol, pivroc, m, n, a, ia, ja, desca, ipiv, ip, jp,
            descip, iwork)
```

```
call pzlapiv(direc, rowcol, pivroc, m, n, a, ia, ja, desca, ipiv, ip, jp,  
            descip, iwork)
```

## 説明

このルーチンは、 $m \times n$  の一般分散行列  $\text{sub}(A) = A(ia:ia+m-1, ja:ja+n-1)$  に  $P(ipiv)$  で示される置換行列) または  $\text{inv}(P)$  を適用し、行または列のピボット演算を行う。ピボットのベクトルは、プロセス行 / 列に分配される。ピボットのベクトルは、分散行列  $A$  とアライメントされなければならない。このルーチンは、必要に応じてピボットのベクトルを転置する。

例えば、行のピボット演算が  $\text{sub}(A)$  の列に適用される場合、`rowcol = 'c'` と `pivroc = 'c'` を渡す。

## 入力パラメータ

<code>direc</code>	(グローバル) CHARACTER*1。 置換順序を指定する。 'F' (順方向) の場合、行列の一番上から順方向にピボットを適用する。 $P * \text{sub}(A)$ を計算する。 'B' (逆方向) の場合、行列の一番下から逆方向にピボットを適用する。 $\text{inv}(P) * \text{sub}(A)$ を計算する。
<code>rowcol</code>	(グローバル) CHARACTER*1。 行、列どちらを置換するのかを指定する。  'R' の場合、行を置換する。 'C' の場合、列を置換する。
<code>pivroc</code>	(グローバル) CHARACTER*1。 <code>ipiv</code> をプロセス行 / 列に分配するかどうかを指定する。 'R' の場合、 <code>ipiv</code> をプロセス行に分配する。 'C' の場合、 <code>ipiv</code> をプロセス列に分配する。
<code>m</code>	(グローバル) INTEGER。 演算を行う行数、すなわち、分散部分行列 $\text{sub}(A)$ の行数。 $m = 0$ と指定した場合、 <code>pzlapiv</code> はゼロに設定される。 $m \geq 0$ 。
<code>n</code>	(グローバル) INTEGER。 演算を行う列数、すなわち、分散部分行列 $\text{sub}(A)$ の列数。 $n = 0$ と指定した場合、 <code>pzlapiv</code> はゼロに設定される。 $n \geq 0$ 。
<code>a</code>	(ローカル) REAL ( <code>pslapiv</code> の場合) DOUBLE PRECISION ( <code>pdlapiv</code> の場合)

COMPLEX (pclapiv の場合)  
 COMPLEX\*16 (pzlapiv の場合)。  
 ローカルメモリにある、次元 ( $lld\_a, LOCC(ja+n-1)$ ) の配列。  
 分散行列  $\text{sub}(A)$  のローカル部分を格納する。

*ia, ja* (グローバル) INTEGER。それぞれ、部分行列  $\text{sub}(A)$  の最初の行と最初の列を示す、グローバル配列  $A$  の行インデックスと列インデックス。

*desca* (グローバルおよびローカル) INTEGER。配列、次元は ( $dlen\_$ )。分散行列  $A$  の配列ディスクリプタ。

*ipiv* (ローカル) INTEGER。  
 配列、次元は ( $lipiv$ )。  $lipiv$  の値は次の通り。  
 $rowcol = 'R'$  または  $'r'$  の場合、  
 $pivroc = 'C'$  または  $'c'$  ならば  $lipiv \geq LOCr(ia+m-1) + mb\_a$ 。  
 $pivroc = 'R'$  または  $'r'$  ならば  $lipiv \geq LOCC(m + \text{mod}(jp-1, nb\_p))$ 。  
 $rowcol = 'C'$  または  $'c'$  の場合、  
 $pivroc = 'C'$  または  $'c'$  ならば  $lipiv \geq LOCr(n + \text{mod}(ip-1, mb\_p))$ 。  
 $pivroc = 'R'$  または  $'r'$  ならば  $lipiv \geq LOCC(ja+n-1) + nb\_a$ 。  
 この配列には、ピボット情報が格納される。  $ipiv(i)$  はローカル行 / 列  $i$  と交換されたグローバル行 / 列。  $rowcol = 'R'$ 、 $'r'$  かつ  $pivroc = 'C'$ 、 $'c'$  のとき、あるいは  $rowcol = 'C'$ 、 $'c'$  かつ  $pivroc = 'R'$ 、 $'r'$  のとき、サイズ  $mb\_a(nb\_a)$  の配列の一番最後の成分は、ワークスペースとして使用される。このような場合、分散行列  $A$  に関連付けられる。

*ip, jp* (グローバル) INTEGER。それぞれ、部分行列  $\text{sub}(P)$  の最初の行と最初の列を示す、グローバル配列  $P$  の行インデックスと列インデックス。

*descip* (グローバルおよびローカル) INTEGER。配列、次元は ( $dlen\_$ )。乱数ベクトル  $ipiv$  の配列ディスクリプタ。

*iwork* (ローカル) INTEGER。  
 配列、次元は ( $ldw$ )。  $ldw$  は、転置に必要なワークスペースのサイズ、および転置された  $ipiv$  の格納と同じ大きさである。  
 $lcm$  は  $nprow$  と  $npcol$  の最小公倍数である。  
 If(  $rowcol.eq.'r'$  .and.  $pivroc.eq.'r'$  ) then  
   If(  $nprow.eq.npcol$  ) then  
      $ldw = LOCr( n\_p + \text{mod}(jp-1, nb\_p) ) + nb\_p$   
   else  
      $ldw = LOCr( n\_p + \text{mod}(jp-1, nb\_p) ) +$   
        $nb\_p * \text{ceil}( \text{ceil}(LOCC(n\_p)/nb\_p) / (lcm/npcol) )$

```
end if
else if( rowcol.eq.'c' .and. pivroc.eq.'c' ) then
  if( nprow.eq.npcol ) then
    ldw=LOCc( m_p + mod(ip-1, mb_p) ) + mb_p
  else
    ldw = LOCc( m_p + mod(ip-1, mb_p) ) +
          mb_p * ceil(ceil(LOCr(m_p)/mb_p) /
(lcm/nprow) )
  end if
else
  iwork is not referenced.
end if.
```

### 出力パラメータ

**a** (ローカル)  
終了時に、置換された分散部分行列のローカル部分。

---

## p?laqge

p?geegu で計算された行と列のスケール係数を使って一般矩形行列をスケールリングする。

---

### 構文

```
call pslaqge(m, n, a, ia, ja, desca, r, c, rowcnd, colcnd, amax, equed)
call pdlaqge(m, n, a, ia, ja, desca, r, c, rowcnd, colcnd, amax, equed)
call pclaqge(m, n, a, ia, ja, desca, r, c, rowcnd, colcnd, amax, equed)
call pzlaqge(m, n, a, ia, ja, desca, r, c, rowcnd, colcnd, amax, equed)
```

### 説明

このルーチンは、 $m \times n$  の一般分散行列  $\text{sub}(A) = A(ia:ia+m-1, ja:ja+n-1)$  を、[p?geegu](#) で計算されたベクトル  $r$  と  $c$  に格納されている行と列のスケール係数を用いて平衡化する。

## 入力パラメータ

<i>m</i>	(グローバル) INTEGER。 演算を行う行数、すなわち、分散部分行列 <i>sub(A)</i> の行数。( $m \geq 0$ )。
<i>n</i>	(グローバル) INTEGER。 演算を行う列数、すなわち、分散部分行列 <i>sub(A)</i> の列数。( $n \geq 0$ )。
<i>a</i>	(ローカル) REAL (pslaqge の場合) DOUBLE PRECISION (pdlaqge の場合) COMPLEX (pclaqge の場合) COMPLEX*16 (pzlaqge の場合)。 ローカルメモリにある、次元 ( <i>lld_a</i> , <i>LOCc(ja+n-1)</i> ) の配列へのポインタ。分散行列 <i>sub(A)</i> を格納する。
<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 <i>sub(A)</i> の最初の行と最初の列を示す、グローバル配列 <i>A</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。分散行列 <i>A</i> の配列ディスクリプタ。
<i>r</i>	(ローカル) REAL (pslaqge の場合) DOUBLE PRECISION (pdlaqge の場合) COMPLEX (pclaqge の場合) COMPLEX*16 (pzlaqge の場合)。 配列、次元は <i>LOCr(m_a)</i> 。 <i>sub(A)</i> の行スケール係数。 <i>r</i> は分散行列 <i>A</i> でアライメントされ、すべてのプロセス列に複製される。 <i>r</i> は、分散行列 <i>A</i> に関連付けられる。
<i>c</i>	(ローカル) REAL (pslaqge の場合) DOUBLE PRECISION (pdlaqge の場合) COMPLEX (pclaqge の場合) COMPLEX*16 (pzlaqge の場合)。 配列、次元は <i>LOCc(n_a)</i> 。 <i>sub(A)</i> の行スケール係数。 <i>c</i> は分散行列 <i>A</i> でアライメントされ、すべてのプロセス列に複製される。 <i>c</i> は、分散行列 <i>A</i> に関連付けられる。
<i>rowcnd</i>	(ローカル) REAL (pslaqge の場合) DOUBLE PRECISION (pdlaqge の場合)

	COMPLEX (pclaqqe の場合 ) COMPLEX*16 (pzlaqqe の場合 )。 最小の $r(i)$ を最大の $r(i)$ , で割ったグローバル値 ( $ia \leq i \leq ia+m-1$ )。
<i>colcnd</i>	( ローカル ) REAL (pslaqqe の場合 ) DOUBLE PRECISION (pdlaqqe の場合 ) COMPLEX (pclaqqe の場合 ) COMPLEX*16 (pzlaqqe の場合 )。 最小の $c(i)$ を最大の $r(i)$ , で割ったグローバル値 ( $ia \leq i \leq ia+n-1$ )。
<i>amax</i>	( グローバル ) REAL (pslaqqe の場合 ) DOUBLE PRECISION (pdlaqqe の場合 ) COMPLEX (pclaqqe の場合 ) COMPLEX*16 (pzlaqqe の場合 )。 最大分散部分行列成分の絶対値。

## 出力パラメータ

<i>a</i>	( ローカル ) 終了時に、平衡化された行列で上書きされる。平衡化された分散行列の形式は <i>equed</i> を参照のこと。
<i>equed</i>	( グローバル ) CHARACTER。 実行された平衡化の形式を表わす。  'N' の場合、平衡化は行われていない。 'R' の場合、行の平衡化、すなわち、 $\text{sub}(A)$ は $\text{diag}(r(ia:ia+m-1))$ で事前乗算された。 'C' の場合、列の平衡化、すなわち、 $\text{sub}(A)$ は $\text{diag}(c(ja:ja+n-1))$ で事後乗算された。 'B' の場合、行と列の平衡化、すなわち、 $\text{sub}(A)$ は $\text{diag}(r(ia:ia+m-1)) * \text{sub}(A) * \text{diag}(c(ja:ja+n-1))$ で置き換えられた。

## p?laqsy

p?poequ で計算されたスケール係数を用いて  
対称/エルミート行列をスケールリングする。

### 構文

```
call pslaqsy(uplo, n, a, ia, ja, desca, sr, sc, scond, amax, equed)
call pdlaqsy(uplo, n, a, ia, ja, desca, sr, sc, scond, amax, equed)
call pclaqsy(uplo, n, a, ia, ja, desca, sr, sc, scond, amax, equed)
call pzlaqsy(uplo, n, a, ia, ja, desca, sr, sc, scond, amax, equed)
```

### 説明

このルーチンは、対称分散行列  $\text{sub}(A) = A(ia:ia+n-1, ja:ja+n-1)$  を、ベクトル  $sr$  と  $sc$  に格納されているスケール係数を用いて平衡化する。スケール係数は [p?poequ](#) で計算される。

### 入力パラメータ

**uplo** (グローバル) CHARACTER。  
sub(A) の上三角部分と下三角部分のどちらを参照させるかを指定する。  
'U' の場合、上三角部分を参照させる。  
'L' の場合、下三角部分を参照させる。

**n** (グローバル) INTEGER。分散部分行列 sub(A) の次数。( $n \geq 0$ )。

**a** (ローカル)  
REAL (pslaqsy の場合)  
DOUBLE PRECISION (pdlaqsy の場合)  
COMPLEX (pclaqsy の場合)  
COMPLEX\*16 (pzlaqsy の場合)。  
ローカルメモリにある、次元 ( $lld\_a, LOCc(ja+n-1)$ ) の配列へのポインタ。分散行列 sub(A) のローカル部分を格納する。分散対称行列 sub(A) のローカル部分を格納する。

uplo = 'U' の場合、sub(A) の先頭の  $n \times n$  の上三角部分に行列の上三角部分を格納する。sub(A) の厳密な下三角部分は参照されない。

uplo = 'L' の場合、sub(A) の先頭の  $n \times n$  の下三角部分に行列の下三角部分を格納する。sub(A) の厳密な上三角部分は参照されない。

<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 <i>sub(A)</i> の最初の行と最初の列を示す、グローバル配列 <i>A</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。分散行列 <i>A</i> の配列ディスクリプタ。
<i>sr</i>	(ローカル) REAL ( <i>pslaqsy</i> の場合) DOUBLE PRECISION ( <i>pdlaqsy</i> の場合) COMPLEX ( <i>pclaqsy</i> の場合) COMPLEX*16 ( <i>pzlaqsy</i> の場合)。 配列、次元は <i>LOCr(m_a)</i> 。 <i>A(ia:ia+m-1, ja:ja+n-1)</i> のスケール係数。 <i>sr</i> は分散行列 <i>A</i> でアライメントされ、すべてのプロセス列に複製される。 <i>sr</i> は、分散行列 <i>A</i> に関連付けられる。
<i>sc</i>	(ローカル) REAL ( <i>pslaqsy</i> の場合) DOUBLE PRECISION ( <i>pdlaqsy</i> の場合) COMPLEX ( <i>pclaqsy</i> の場合) COMPLEX*16 ( <i>pzlaqsy</i> の場合)。 配列、次元は <i>LOCr(m_a)</i> 。 <i>A(ia:ia+m-1, ja:ja+n-1)</i> のスケール係数。 <i>sr</i> は分散行列 <i>A</i> でアライメントされ、すべてのプロセス列に複製される。 <i>sr</i> は、分散行列 <i>A</i> に関連付けられる。
<i>scond</i>	(グローバル) REAL ( <i>pslaqsy</i> の場合) DOUBLE PRECISION ( <i>pdlaqsy</i> の場合) COMPLEX ( <i>pclaqsy</i> の場合) COMPLEX*16 ( <i>pzlaqsy</i> の場合)。 最小の <i>sr(i) (sc(j))</i> を最大の <i>sr(i) (sc(j))</i> で割った値 ( $ia \leq i \leq ia+n-1$ および $ja \leq j \leq ja+n-1$ )。
<i>amax</i>	(グローバル) REAL ( <i>pslaqsy</i> の場合) DOUBLE PRECISION ( <i>pdlaqsy</i> の場合) COMPLEX ( <i>pclaqsy</i> の場合) COMPLEX*16 ( <i>pzlaqsy</i> の場合)。 最大分散部分行列成分の絶対値。

## 出力パラメータ

<i>a</i>	終了時に、 <i>equed</i> = 'Y' の場合、平均化された行列は $\text{diag}(sr(ia:ia+n-1)) * \text{sub}(A) * \text{diag}(sc(ja:ja+n-1))$ 。
----------	---



*equed* (グローバル) CHARACTER\*1。  
 平衡化が行われたかどうかを示す。  
 'N' の場合、平衡化は行われていない。  
 'Y' の場合、平衡化が行われ、 $\text{sub}(A)$  は  
 $\text{diag}(\text{sr}(\text{ia}:\text{ia}+\text{n}-1)) * \text{sub}(A) * \text{diag}(\text{sc}(\text{ja}:\text{ja}+\text{n}-1))$  で置き換えられて  
 いる。

## p?lared1d

入力配列 *bycol* は複数の行に分配され、すべての  
 プロセス列に同じ *bycol* のコピーが格納されるこ  
 とを前提に、配列を再分配する。

### 構文

```
call pslared1d(n, ia, ja, desc, bycol, byall, work, lwork)
call pdlared1d(n, ia, ja, desc, bycol, byall, work, lwork)
```

### 説明

このルーチンは、1次元配列を再分配する。入力配列 *bycol* は複数の行に分配され、す  
 べてのプロセス列に同じ *bycol* のコピーが格納されると仮定する。出力配列 *byall* は  
 すべてのプロセスで同じであり、配列全体を格納する。

### 入力パラメータ

*np = bycol()* のローカル行数。

*n* (グローバル) INTEGER。  
 再分配する成分の数。  $n \geq 0$ 。

*ia, ja* (グローバル) INTEGER。 *ia*, *ja* は 1 に等しくなければならない。

*desc* (グローバルおよびローカル) INTEGER。配列、次元は 8。 *bycol* の 2 次  
 元配列ディスクリプタ。

*bycol* (ローカル)  
 REAL (pslared1d の場合)  
 DOUBLE PRECISION (pdlared1d の場合)  
 COMPLEX (pclared1d の場合)  
 COMPLEX\*16 (pzlared1d の場合)。

分散ブロックサイクル配列、グローバル次元は ( $n$ )、ローカル次元は  $np$ 。  $bycol$  はプロセス行に分配される。すべてのプロセス列には同じ値が格納されるものと仮定する。

**work** (ローカル)  
REAL (pslared1d の場合)  
DOUBLE PRECISION (pdlared1d の場合)  
COMPLEX (pclared1d の場合)  
COMPLEX\*16 (pzlared1d の場合)。  
次元は ( $lwork$ )。1つのプロセスから他のプロセスへ送られたバッファを格納する。

**lwork** (ローカル) INTEGER。  
配列 **work** のサイズ。  $lwork \geq \text{numroc}(n, \text{desc}(\text{nb\_}), 0, 0, npcol)$ 。

## 出力パラメータ

**byall** (グローバル)  
REAL (pslared1d の場合)  
DOUBLE PRECISION (pdlared1d の場合)  
COMPLEX (pclared1d の場合)  
COMPLEX\*16 (pzlared1d の場合)。  
グローバル次元は ( $n$ )、ローカル次元は ( $n$ )。 **byall** は、すべてのプロセスで完全に複製される。  $bycol$  と同じ値を格納する。すべてのプロセスに、分配ではなく、複製される。

---

## p?lared2d

入力配列 **byrow** は複数の列に分配され、すべてのプロセス行に同じ **byrow** のコピーが格納されることを前提に、配列を再分配する。

---

### 構文

```
call pslared2d( $n$ ,  $ia$ ,  $ja$ ,  $desc$ ,  $byrow$ ,  $byall$ ,  $work$ ,  $lwork$ )  
call pdlared2d( $n$ ,  $ia$ ,  $ja$ ,  $desc$ ,  $byrow$ ,  $byall$ ,  $work$ ,  $lwork$ )
```

## 説明

このルーチンは、1次元配列を再分配する。入力配列 *byrow* は複数の列に分配され、すべてのプロセス行に同じ *byrow* のコピーが格納されると仮定する。出力配列 *byall* はすべてのプロセスで同じであり、配列全体を格納する。

## 入力パラメータ

*np* = *byrow*() のローカル行数

*n* (グローバル) INTEGER。  
再分配する成分の数。  $n \geq 0$ 。

*ia, ja* (グローバル) INTEGER。 *ia*、*ja* は 1 に等しくなければならない。

*desc* (グローバルおよびローカル) INTEGER。配列、次元は (*dlen\_*)。 *byrow* の 2 次元配列ディスクリプタ。

*byrow* (ローカル)  
REAL (*pslared2d* の場合)  
DOUBLE PRECISION (*pdlared2d* の場合)  
COMPLEX (*pclared2d* の場合)  
COMPLEX\*16 (*pzlared2d* の場合)。  
分散ブロックサイクル配列、グローバル次元は (*n*)、ローカル次元は *np*。 *bycol* はプロセス列に分配される。すべてのプロセス行には同じ値が格納されるものと仮定する。

*work* (ローカル)  
REAL (*pslared2d* の場合)  
DOUBLE PRECISION (*pdlared2d* の場合)  
COMPLEX (*pclared2d* の場合)  
COMPLEX\*16 (*pzlared2d* の場合)。  
次元は (*lwork*)。1つのプロセスから他のプロセスへ送られたバッファを格納する。

*lwork* (ローカル) INTEGER。  
配列 *work* のサイズ。  $lwork \geq \text{numroc}(n, \text{desc}(\text{nb\_}), 0, 0, \text{npcol})$ 。

## 出力パラメータ

*byall* (グローバル)  
REAL (*pslared2d* の場合)  
DOUBLE PRECISION (*pdlared2d* の場合)  
COMPLEX (*pclared2d* の場合)  
COMPLEX\*16 (*pzlared2d* の場合)。

グローバル次元は (n)、ローカル次元は (n)。byall は、すべてのプロセスで完全に複製される。bycol と同じ値を格納する。すべてのプロセスに、分配ではなく、複製される。

---

### p?larf

一般矩形行列に基本リフレクタを適用する。

---

#### 構文

```
call pslarf(side, m, n, v, iv, jv, descv, incv, tau, c, ic, jc, descc, work)
call pdlarf(side, m, n, v, iv, jv, descv, incv, tau, c, ic, jc, descc, work)
call pclarf(side, m, n, v, iv, jv, descv, incv, tau, c, ic, jc, descc, work)
call pzlarf(side, m, n, v, iv, jv, descv, incv, tau, c, ic, jc, descc, work)
```

#### 説明

このルーチンは、実 / 複素基本リフレクタ  $Q$  (または  $Q^T$ ) を、 $m \times n$  の実 / 複素分布行列  $\text{sub}(C) = C(ic:ic+m-1, jc:jc+n-1)$  に左または右のいずれかから適用する。 $Q$  は次の形式で表現される。

$$Q = I - \tau * v * v'$$

$\tau$  は実 / 複素スカラ、 $v$  は実 / 複素ベクトルである。

$\tau = 0$  の場合、 $Q$  は単位行列をとる。

#### 入力パラメータ

side	(グローバル) CHARACTER。 'L' の場合、形式 $Q * \text{sub}(C)$ 。 'R' の場合、形式 $\text{sub}(C) * Q$ 、 $Q = Q^T$ 。
m	(グローバル) INTEGER。 演算を行う行数、すなわち、分散部分行列 $\text{sub}(A)$ の行数。( $m \geq 0$ )。
n	(グローバル) INTEGER。 演算を行う列数、すなわち、分散部分行列 $\text{sub}(A)$ の列数。( $n \geq 0$ )。
v	(ローカル) REAL (pslarf の場合) DOUBLE PRECISION (pdlarf の場合)

	<p>COMPLEX (pclarf の場合 )  COMPLEX*16 (pzlarf の場合 )。  ローカルメモリにある、Householder 変換 <math>Q</math> を表す分散行列 <math>V</math> のローカル部分を格納する、次元 (<math>lld\_v, *</math>) の配列へのポインタ。</p> <p><math>side = 'L'</math>、<math>incv = 1</math> の場合、<math>v(iv:iv+m-1, jv)</math>。  <math>side = 'L'</math>、<math>incv = m\_v</math> の場合、<math>v(iv, jv:jv+m-1)</math>。  <math>side = 'R'</math>、<math>incv = 1</math> の場合、<math>v(iv:iv+n-1, jv)</math>。  <math>side = 'R'</math>、<math>incv = m\_v</math> の場合、<math>v(iv, jv:jv+n-1)</math>。</p> <p><math>Q</math> の表現におけるベクトル <math>v</math>。 <math>tau = 0</math> の場合、<math>v</math> は使用されない。</p>
<i>iv, jv</i>	(グローバル) INTEGER。それぞれ、部分行列 $sub(V)$ の最初の行と最初の列を示す、グローバル配列 $V$ の行インデックスと列インデックス。
<i>descv</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( $dlen\_$ )。分散行列 $V$ の配列ディスクリプタ。
<i>incv</i>	(グローバル) INTEGER。 $v$ の成分のグローバルな増分。このバージョンでは、1 と $m\_v$ の 2 つの $incv$ の値のみサポートされる。 $incv$ の値は、ゼロであってはならない。
<i>tau</i>	<p>(ローカル)  REAL (pslarf の場合 )  DOUBLE PRECISION (pdlarf の場合 )  COMPLEX (pclarf の場合 )  COMPLEX*16 (pzlarf の場合 )。  配列、次元は <math>LOCc(jv)</math> (<math>incv = 1</math> の場合 ) または <math>LOCr(iv)</math> (それ以外の場合 )。Householder ベクトルに関する Householder スカラを格納する。  <math>tau</math> は、分散行列 <math>v</math> に関連付けられる。</p>
<i>c</i>	<p>(ローカル)  REAL (pslarf の場合 )  DOUBLE PRECISION (pdlarf の場合 )  COMPLEX (pclarf の場合 )  COMPLEX*16 (pzlarf の場合 )。  ローカルメモリにある、<math>sub(C)</math> のローカル部分を格納する、次元 (<math>lld\_c, LOCc(jc+n-1)</math>) の配列へのポインタ。</p>
<i>ic, jc</i>	(グローバル) INTEGER。それぞれ、部分行列 $sub(C)$ の最初の行と最初の列を示す、グローバル配列 $C$ の行インデックスと列インデックス。
<i>desc</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( $dlen\_$ )。分散行列 $C$ の配列ディスクリプタ。

work

(ローカル)

```

REAL (pslarf の場合)
DOUBLE PRECISION (pdlarf の場合)
COMPLEX (pcalarf の場合)
COMPLEX*16 (pzlarf の場合)。
配列、次元は (lwork)。

If incv = 1,
  if side = 'L',
    if ivcol = iccol,
      lwork ≥ nqc0
    else
      lwork ≥ mpc0 + max( 1, nqc0 )
    end if
  else if side = 'R',
    lwork > nqc0 + max( max( 1, mpc0 ), numroc( numroc( n
      + icoffc, nb_v, 0, 0, npc0l ), nb_v, 0, 0, lcmq ) )
    end if
else if incv = m_v,
  if side = 'L',
    lwork ≥ mpc0 + max( max( 1, nqc0 ), numroc(
numroc(m+iroffc, mb_v, 0, 0, nprow ), mb_v, 0, 0, lcmq ) )
  else if side = 'R',
    if ivrow = icrow,
      lwork ≥ mpc0
    else
      lwork ≥ nqc0 + max( 1, mpc0 )
    end if
  end if
end if,

lcm は nprow と npc0l の最小公倍数である。lcm = ilcm(nprow,
npc0l)、lcmp = lcm / nprow、lcmq = lcm / npc0l、

iroffc = mod(ic-1, mb_c)、icoffc = mod(jc-1, nb_c)、
icrow = indxg2p(ic, mb_c, myrow, rsrc_c, nprow)、
iccol = indxg2p(jc, nb_c, mycol, csrc_c, npc0l)、
mpc0 = numroc(m+iroffc, mb_c, myrow, icrow, nprow)、
nqc0 = numroc(n+icoffc, nb_c, mycol, iccol, npc0l)。

```

ilcm、indxg2p および numroc は、ScaLAPACK ツール関数である。  
 myrow、mycol、nprow、および npc1 は、サブルーチン  
 blacs\_gridinfo を呼び出して求められる。

### 出力パラメータ

*c* (ローカル)  
 終了時に、sub(*C*) は  $Q * \text{sub}(C)$  (*side* = 'L' の場合) または  $\text{sub}(C) * Q$  (*side* = 'R' の場合) で上書きされる。

## p?larfb

ブロック・リフレクタまたはその転置/共役転置  
 を一般矩形行列に適用する。

### 構文

```
call pslarfb(side, trans, direct, storev, m, n, k, v, iv, jv, descv, t,c, ic,
             jc, descc, work)
call pdlarfb(side, trans, direct, storev, m, n, k, v, iv, jv, descv, t,c, ic,
             jc, descc, work)
call pclarfb(side, trans, direct, storev, m, n, k, v, iv, jv, descv, t,c, ic,
             jc, descc, work)
call pzlarfb(side, trans, direct, storev, m, n, k, v, iv, jv, descv, t,c, ic,
             jc, descc, work)
```

### 説明

このルーチンは、実/複素ブロック・リフレクタ  $Q$  またはその転置  $Q^T$  (または共役置換  $Q^H$ ) を、 $m \times n$  の実/複素分布行列  $\text{sub}(C) = C(ic:ic+m-1, jc:jc+n-1)$  に左または右のいずれかから適用する。

### 入力パラメータ

*side* (グローバル) CHARACTER。  
*side* = 'L' の場合、 $Q$ 、 $Q^T$  (実数型の場合) または  $Q^H$  (複素数型の場合) を左から適用する。  
*side* = 'R' の場合、 $Q$ 、 $Q^T$  (実数型の場合) または  $Q^H$  (複素数型の場合) を右から適用する。

<i>trans</i>	<p>(グローバル) CHARACTER。</p> <p><i>trans</i> = 'N' の場合、<math>Q</math> を適用する (転置なし)。</p> <p><i>trans</i> = 'T' の場合、<math>Q^T</math> を適用する (転置、実数型の場合)。</p> <p><i>trans</i> = 'C' の場合、<math>Q^H</math> を適用する (共役転置、複素数型の場合)。</p>
<i>direct</i>	<p>(グローバル) CHARACTER。</p> <p>基本リフレクタの積からどのように <math>Q</math> が表現されているかを示す。</p> <p><i>direct</i> = 'F' の場合、<math>Q = H(1) H(2) \dots H(k)</math> (順方向)。</p> <p><i>direct</i> = 'B' の場合、<math>Q = H(k) \dots H(2) H(1)</math> (逆方向)。</p>
<i>storev</i>	<p>(グローバル) CHARACTER。</p> <p>基本リフレクタを定義するベクトルがどのように格納されているかを示す。</p> <p><i>storev</i> = 'C' の場合、列方向。</p> <p><i>storev</i> = 'R' の場合、行方向。</p>
<i>m</i>	<p>(グローバル) INTEGER。</p> <p>演算を行う行数、すなわち、分散部分行列 <math>\text{sub}(C)</math> の行数。 (<math>m \geq 0</math>)。</p>
<i>n</i>	<p>(グローバル) INTEGER。</p> <p>演算を行う列数、すなわち、分散部分行列 <math>\text{sub}(C)</math> の列数。 (<math>n \geq 0</math>)。</p>
<i>k</i>	<p>(グローバル) INTEGER。</p> <p>行列 <math>T</math> の次数。</p>
<i>v</i>	<p>(ローカル)</p> <p>REAL (pslarfb の場合)</p> <p>DOUBLE PRECISION (pdlarfb の場合)</p> <p>COMPLEX (pclarfb の場合)</p> <p>COMPLEX*16 (pzlarfb の場合)。</p> <p>ローカルメモリにある、次元</p> <p><math>(11d\_v, LOCc(jv+k-1))</math> (<i>storev</i> = 'C' の場合)、</p> <p><math>(11d\_v, LOCc(jv+m-1))</math> (<i>storev</i> = 'R' および <i>side</i> = 'L' の場合)、</p> <p>または <math>(11d\_v, LOCc(jv+n-1))</math> (<i>storev</i> = 'R' および <i>side</i> = 'R' の場合)</p> <p>の配列へのポインタ。Householder 変換を表す乱数ベクトル <math>V</math> のローカル部分を格納する。</p> <p><i>storev</i> = 'C' および <i>side</i> = 'L' の場合、<math>11d\_v \geq \max(1, LOCr(iv+m-1))</math>。</p> <p><i>storev</i> = 'C' および <i>side</i> = 'R' の場合、<math>11d\_v \geq \max(1, LOCr(iv+n-1))</math>。</p> <p><i>storev</i> = 'R' の場合、<math>11d\_v \geq LOCr(jv+k-1)</math>。</p>



<i>iv, jv</i>	(グローバル) INTEGER。それぞれ、部分行列 <i>sub(V)</i> の最初の行と最初の列を示す、グローバル配列 <i>V</i> の行インデックスと列インデックス。
<i>descv</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。分散行列 <i>V</i> の配列ディスクリプタ。
<i>c</i>	(ローカル) REAL ( <i>pslarfb</i> の場合) DOUBLE PRECISION ( <i>pdlarfb</i> の場合) COMPLEX ( <i>pclarfb</i> の場合) COMPLEX*16 ( <i>pzlarfb</i> の場合)。 ローカルメモリにある、 <i>sub(C)</i> のローカル部分を格納する、次元 ( <i>lld_c</i> , <i>LOCc(jc+n-1)</i> ) の配列へのポインタ。
<i>ic, jc</i>	(グローバル) INTEGER。それぞれ、部分行列 <i>sub(C)</i> の最初の行と最初の列を示す、グローバル配列 <i>C</i> の行インデックスと列インデックス。
<i>descC</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。分散行列 <i>C</i> の配列ディスクリプタ。
<i>work</i>	(ローカル) REAL ( <i>pslarfb</i> の場合) DOUBLE PRECISION ( <i>pdlarfb</i> の場合) COMPLEX ( <i>pclarfb</i> の場合) COMPLEX*16 ( <i>pzlarfb</i> の場合)。 ワークスペース配列、次元は ( <i>lwork</i> )。  If <i>storev</i> = 'C', if <i>side</i> = 'L', <i>lwork</i> ≥ ( <i>nqc0</i> + <i>mpc0</i> ) * <i>k</i> else if <i>side</i> = 'R', <i>lwork</i> ≥ ( <i>nqc0</i> + max( <i>npv0</i> + numroc( numroc( <i>n</i> + <i>icoffc</i> , <i>nb_v</i> , 0, 0, <i>npcol</i> ), <i>nb_v</i> , 0, 0, <i>lcmq</i> ), <i>mpc0</i> ) ) * <i>k</i> end if else if <i>storev</i> = 'R', if <i>side</i> = 'L', <i>lwork</i> ≥ ( <i>mpc0</i> + max( <i>mqv0</i> + numroc( numroc( <i>m</i> + <i>iroffc</i> , <i>mb_v</i> , 0, 0, <i>nprow</i> ), <i>mb_v</i> , 0, 0, <i>lcmp</i> ), <i>nqc0</i> ) ) * <i>k</i>

```

        else if side='R',
            lwork ≥ ( mpc0 + nqc0 ) * k
        end if
    end if,
    ここで、lcmq = lcm / npcol、lcm = iclm( nprow, npcol )、
    iroffv = mod( iv-1, mb_v ), icoffv = mod( jv-1, nb_v ),
    ivrow = indxg2p( iv, mb_v, myrow, rsrc_v, nprow ),
    ivcol = indxg2p( jv, nb_v, mycol, csrc_v, npcol ),
    MqV0 = numroc( m+icoffv, nb_v, mycol, ivcol, npcol ),
    NpV0 = numroc( n+iroffv, mb_v, myrow, ivrow, nprow ),

    iroffc = mod( ic-1, mb_c ), icoffc = mod( jc-1, nb_c ),
    icrow = indxg2p( ic, mb_c, myrow, rsrc_c, nprow ),
    iccol = indxg2p( jc, nb_c, mycol, csrc_c, npcol ),
    MpC0 = numroc( m+iroffc, mb_c, myrow, icrow, nprow ),
    NpC0 = numroc( n+icoffc, mb_c, myrow, icrow, nprow ),
    NqC0 = numroc( n+icoffc, nb_c, mycol, iccol, npcol )。

    ilcm、indxg2p および numroc は、ScaLAPACK ツール関数である。
    myrow、mycol、nrow、および npcol は、サブルーチン
    blacs_gridinfo を呼び出して求められる。

```

## 出力パラメータ

*c* (ローカル)  
 終了時に、*sub(C)* は、 $Q * \text{sub}(C)$ 、 $Q' * \text{sub}(C)$ 、 $\text{sub}(C) * Q$ 、または  $\text{sub}(C) * Q'$  のいずれかで上書きされる。

---

## p?larfc

一般行列に基本リフレクタの共役転置を適用する。

---

### 構文

```

call pclarfc(side, m, n, v, iv, jv, descv, incv, tau, c, ic, jc, descc, work)
call pzlarfc(side, m, n, v, iv, jv, descv, incv, tau, c, ic, jc, descc, work)

```

## 説明

このルーチンは、複素基本リフレクタ  $Q^H$  を、 $m \times n$  の複素分布行列  $\text{sub}(C) = C(ic:ic+m-1, jc:jc+n-1)$  に左または右のいずれかから適用する。 $Q$  は次の形式で表現される。

$$Q = I - \tau * v * v'$$

$\tau$  は複素スカラ、 $v$  は複素ベクトルである。

$\tau = 0$  の場合、 $Q$  は単位行列をとる。

## 入力パラメータ

<i>side</i>	(グローバル) CHARACTER。 $side = 'L'$ の場合、形式 $Q^H * \text{sub}(C)$ 。 $side = 'R'$ の場合、形式 $\text{sub}(C) * Q^H$ 。
<i>m</i>	(グローバル) INTEGER。 演算を行う行数、すなわち、分散部分行列 $\text{sub}(C)$ の行数。( $m \geq 0$ )。
<i>n</i>	(グローバル) INTEGER。 演算を行う列数、すなわち、分散部分行列 $\text{sub}(C)$ の列数。( $n \geq 0$ )。
<i>v</i>	(ローカル) COMPLEX (pclarfc の場合) COMPLEX*16 (pzlarfc の場合)。 ローカルメモリにある、Householder 変換 $Q$ を表す分散行列 $v$ のローカル部分を格納する、次元 ( $lld_v, *$ ) の配列へのポインタ。  $side = 'L'$ 、 $incv = 1$ の場合、 $v(iv:iv+m-1, jv)$ 。 $side = 'L'$ 、 $incv = m_v$ の場合、 $v(iv, jv:jv+m-1)$ 。 $side = 'R'$ 、 $incv = 1$ の場合、 $v(iv:iv+n-1, jv)$ 。 $side = 'R'$ 、 $incv = m_v$ の場合、 $v(iv, jv:jv+n-1)$ 。  $Q$ の表現におけるベクトル $v$ 。 $\tau = 0$ の場合、 $v$ は使用されない。
<i>iv, jv</i>	(グローバル) INTEGER。それぞれ、部分行列 $\text{sub}(V)$ の最初の行と最初の列を示す、グローバル配列 $V$ の行インデックスと列インデックス。
<i>descv</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( $dlen_v$ )。分散行列 $V$ の配列ディスクリプタ。
<i>incv</i>	(グローバル) INTEGER。 $v$ の成分のグローバルな増分。このバージョンでは、1 と $m_v$ の2つの $incv$ の値のみサポートされる。 $incv$ の値は、ゼロであってはならない。

<code>tau</code>	<p>(ローカル)</p> <p>COMPLEX (pclarfc の場合)</p> <p>COMPLEX*16 (pzlarfc の場合)。</p> <p>配列、次元は <math>LOCc(jv)</math> (<math>incv = 1</math> の場合) または <math>LOCr(iv)</math> (それ以外の場合)。Householder ベクトルに関する Householder スカラを格納する。</p> <p><code>tau</code> は、分散行列 <math>V</math> に関連付けられる。</p>
<code>c</code>	<p>(ローカル)</p> <p>COMPLEX (pclarfc の場合)</p> <p>COMPLEX*16 (pzlarfc の場合)。</p> <p>ローカルメモリにある、<code>sub(C)</code> のローカル部分を格納する、次元 (<math>lld\_c, LOCc(jc+n-1)</math>) の配列へのポインタ。</p>
<code>ic, jc</code>	<p>(グローバル) INTEGER。それぞれ、部分行列 <code>sub(C)</code> の最初の行と最初の列を示す、グローバル配列 <math>C</math> の行インデックスと列インデックス。</p>
<code>descc</code>	<p>(グローバルおよびローカル) INTEGER。配列、次元は (<math>dlen\_</math>)。分散行列 <math>C</math> の配列ディスクリプタ。</p>
<code>work</code>	<p>(ローカル)</p> <p>COMPLEX (pclarfc の場合)</p> <p>COMPLEX*16 (pzlarfc の場合)。</p> <p>ワークスペース配列、次元は (<math>lwork</math>)。</p> <pre> If incv = 1,   if side = 'L',     if ivcol = iccol,       lwork ≥ nqc0     else       lwork ≥ mpc0 + max( 1, nqc0 )     end if   else if side = 'R',     lwork ≥ nqc0 + max( max( 1, mpc0 ), numroc( numroc(       n+icoffc,nb_v,0,0,npcol ),nb_v,0,0,lcmq ) )     end if   else if incv = m_v,     if side = 'L',       lwork ≥ mpc0 + max( max( 1, nqc0 ), numroc( numroc(         m+iroffc,mb_v,0,0,nprow ),mb_v,0,0,lcmp ) )     else if side = 'R',       if ivrow = icrow, </pre>

```

        lwork ≥ mpc0
    else
        lwork ≥ nqc0 + max( 1, mpc0 )
    end if
end if
end if,

```

$lcm$  は  $nprow$  と  $npcol$  の最小公倍数である。

$lcm = ilcm(nprow, npcol)$ 、 $lcmp = lcm / nprow$ 、 $lcmq = lcm / npcol$ 、

```

iroffc = mod( ic-1, mb_c ), icoffc = mod( jc-1, nb_c ),
icrow = indxg2p( ic, mb_c, myrow, rsrc_c, nprow ),
iccol = indxg2p( jc, nb_c, mycol, csrc_c, npcol ),
mpc0 = numroc( m+iroffc, mb_c, myrow, icrow, nprow ),
nqc0 = numroc( n+icoffc, nb_c, mycol, iccol, npcol ).

```

$ilcm$ 、 $indxg2p$  および  $numroc$  は、ScaLAPACK ツール関数である。  
 $myrow$ 、 $mycol$ 、 $nprow$ 、および  $npcol$  は、サブルーチン  $blacs\_gridinfo$  を呼び出して求められる。

### 出力パラメータ

$c$  (ローカル) 終了時に、 $\text{sub}(C)$  は、 $Q^H * \text{sub}(C)$  ( $side = 'L'$  の場合) または  $\text{sub}(C) * Q^H$  ( $side = 'R'$  の場合) で上書きされる。

## p?larfg

基本リフレクタ (Householder 行列) を生成する

### 構文

```

call pslarfg(n, alpha, iax, jax, x, ix, jx, descx, incx, tau)
call pdlarfg(n, alpha, iax, jax, x, ix, jx, descx, incx, tau)
call pclarfg(n, alpha, iax, jax, x, ix, jx, descx, incx, tau)
call pzlarfg(n, alpha, iax, jax, x, ix, jx, descx, incx, tau)

```

### 説明

このルーチンは、次を満たすような次数  $n$  の実 / 複素基本リフレクタ  $H$  を生成する。

$$H * \text{sub}(X) = H * (x(\text{iax}, \text{jax})) = (\alpha), H' * H = I, \\ \begin{pmatrix} & & x & & \end{pmatrix} \begin{pmatrix} & & 0 & & \end{pmatrix}$$

ここで、 $\alpha$  はスカラ (複素数型の場合、実スカラ)、 $\text{sub}(X)$  は  $(n-1)$  成分で構成される実 / 複素乱数ベクトル  $X(\text{ix}:\text{ix}+n-2, \text{jx})$  ( $\text{incx}=1$  の場合) または  $X(\text{ix}, \text{jx}:\text{jx}+n-2)$  ( $\text{incx}=\text{descx}(m_)$  の場合) である。 $H$  は次の形式で表現される。

$$H = I - \tau * (1) * (1 \ v') \\ \begin{pmatrix} & & & & v \end{pmatrix}$$

$\tau$  は実 / 複素スカラ、 $v$  は実 / 複素の  $(n-1)$  成分で構成されるベクトルである。 $H$  はエルミートではない点に注意する。

$\text{sub}(X)$  の成分がすべてゼロの場合 (かつ、複素数型の場合  $X(\text{iax}, \text{jax})$  が実数ならば)、 $\tau=0$  となり、 $H$  は単位行列をとる。

それ以外の場合、 $1 \leq \text{real}(\tau) \leq 2$  かつ  $\text{abs}(\tau-1) \leq 1$ 。

## 入力パラメータ

$n$	(グローバル) INTEGER。 基本リフレクタのグローバル次数。 $n \geq 0$ 。
$\text{iax}, \text{jax}$	(グローバル) INTEGER。 $X(\text{iax}, \text{jax})$ の $x$ のグローバル行インデックスとグローバル列インデックス。
$x$	(ローカル)  REAL (pslarfg の場合) DOUBLE PRECISION (pdlarfg の場合) COMPLEX (pcclarfg の場合) COMPLEX*16 (pzlarfg の場合)。  ローカルメモリにある、次元 $(\text{lld}_x, *)$ の配列へのポインタ。乱数ベクトル $\text{sub}(X)$ のローカル部分を格納する。このルーチンに入る前に、増分された配列 $\text{sub}(X)$ にベクトル $x$ を格納しなければならない。
$\text{ix}, \text{jx}$	(グローバル) INTEGER。それぞれ、 $\text{sub}(X)$ の最初の行と最初の列を示す、グローバル配列 $X$ の行インデックスと列インデックス。
$\text{descx}$	(グローバルおよびローカル) INTEGER。 次元 $(\text{dlen}_)$ の配列。分散行列 $X$ の配列ディスクリプタ。
$\text{incx}$	(グローバル) INTEGER。

$x$  の成分のグローバルな増分。このバージョンでは、1 と  $m_x$  の 2 つの  $incx$  の値のみサポートされる。  
 $incx$  の値は、ゼロであってはならない。

### 出力パラメータ

$\alpha$  (ローカル)

REAL (pslafg の場合)  
 DOUBLE PRECISION (pdlafg の場合)  
 COMPLEX (pclafg の場合)  
 COMPLEX\*16 (pzlafg の場合)。

終了時に、 $\alpha$  は、ベクトル  $\text{sub}(X)$  のプロセス範囲において計算される。

$x$  (ローカル)

終了時に、ベクトル  $v$  で上書きされる。

$\tau$  (ローカル)

REAL (pslarfg の場合)  
 DOUBLE PRECISION (pdlarfg の場合)  
 COMPLEX (pclarfg の場合)  
 COMPLEX\*16 (pzlarfg の場合)。

配列、次元は  $LOCc(jx)$  ( $incx = 1$  の場合) または  $LOCr(ix)$  (それ以外の場合)。Householder ベクトルに関する Householder スカラを格納する。

$\tau$  は、分散行列  $X$  に関連付けられる。

## p?larft

ブロック・リフレクタ  $H = I - VTV^H$  の三角ベクトル  $T$  を生成する。

### 構文

```
call pslarft(direct, storev, n, k, v, iv, jv, descv, tau, t, work)
call pdlarft(direct, storev, n, k, v, iv, jv, descv, tau, t, work)
call pclarft(direct, storev, n, k, v, iv, jv, descv, tau, t, work)
```

```
call pzlarft(direct, storev, n, k, v, iv, jv, descv, tau, t, work)
```

## 説明

このルーチンは、 $k$  個の基本リフレクタの積として定義されている次数  $n$  の実 / 複素ブロック・リフレクタ  $H$  の三角係数  $T$  を生成する。

$direct = 'F'$  の場合、 $H = H(1) H(2) \dots H(k)$ 、 $T$  は上三角。

$direct = 'B'$  の場合、 $H = H(k) \dots H(2) H(1)$ 、 $T$  は下三角。

$storev = 'C'$  の場合、基本リフレクタ  $H(i)$  を定義するベクトルは、分散行列  $V$  の  $i$  番目の列に次のように格納される。

$$H = I - V * T * V'$$

$storev = 'C'$  の場合、基本リフレクタ  $H(i)$  を定義するベクトルは、分散行列  $V$  の  $i$  番目の行に次のように格納される。

$$H = I - V' * T * V$$

## 入力パラメータ

<i>direct</i>	(グローバル) CHARACTER*1。 ブロック・リフレクタを生成するために基本リフレクタを乗算する次数を指定する。  $direct = 'F'$ の場合、 $H = H(1) H(2) \dots H(k)$ (順方向) $direct = 'B'$ の場合、 $H = H(k) \dots H(2) H(1)$ (逆方向)
<i>storev</i>	(グローバル) CHARACTER*1。 基本リフレクタを定義するベクトルがどのように格納されているかを指定する (次の「アプリケーション・ノート」を参照)。  $storev = 'C'$ の場合、列方向。 $storev = 'R'$ の場合、行方向。
<i>n</i>	(グローバル) INTEGER。 ブロック・リフレクタ $H$ の次数。 $n \geq 0$ 。
<i>k</i>	(グローバル) INTEGER。 三角係数 $T$ の次数 (基本リフレクタの個数に等しい)。 $1 \leq k \leq mb\_v (= nb\_v)$ 。
<i>v</i>	REAL (pzlarft の場合) DOUBLE PRECISION (pdlarft の場合) COMPLEX (pcalarft の場合)



	COMPLEX*16 (pzlarft の場合)。 ローカルメモリにある、ローカル次元 ( $LOCr(iv+n-1), LOCc(jv+k-1)$ ) (storev = 'C' の場合) または ( $LOCr(iv+k-1), LOCc(jv+n-1)$ ) (storev = 'R' の場合) の配列へのポインタ。分散行列 $V$ は Householder ベクトルを格納する (次の「アプリケーション・ノート」を参照)。
<i>iv, jv</i>	(グローバル) INTEGER。それぞれ、部分行列 sub( $V$ ) の最初の行と最初 の列を示す、グローバル配列 $v$ の行インデックスと列インデックス。
<i>descv</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( $dlen\_$ )。分散行 列 $V$ の配列ディスクリプタ。
<i>tau</i>	(ローカル) REAL (pslarft の場合) DOUBLE PRECISION (pdlarft の場合) COMPLEX (pclarft の場合) COMPLEX*16 (pzlarft の場合)。 配列、次元は $LOCr(iv+k-1)$ ( $incv = m\_v$ の場合) または $LOCc(jv+k-1)$ (それ以外の場合)。Householder ベクトルに関する Householder スカ ラを格納する。  $tau$ は、分散行列 $V$ に関連付けられる。
<i>work</i>	(ローカル)  REAL (pslarft の場合) DOUBLE PRECISION (pdlarft の場合) COMPLEX (pclarft の場合) COMPLEX*16 (pzlarft の場合)。 ワークスペース配列、次元は $(k*(k-1)/2)$ 。

## 出力パラメータ

<i>v</i>	REAL (pslarft の場合) DOUBLE PRECISION (pdlarft の場合) COMPLEX (pclarft の場合) COMPLEX*16 (pzlarft の場合)。
<i>t</i>	(ローカル)  REAL (pslarft の場合) DOUBLE PRECISION (pdlarft の場合) COMPLEX (pclarft の場合) COMPLEX*16 (pzlarft の場合)。 配列、次元は ( $nb\_v, nb\_v$ ) (storev = 'Col' の場合) または ( $mb\_v, mb\_v$ )

(それ以外の場合)。vに関連するブロック・リフレクタの  $k \times k$  の三角係数が格納される。*direct* = 'F' の場合、*t* は上三角。  
*direct* = 'B' の場合、*t* は下三角。

## アプリケーション・ノート

行列 *V* の形状と *H(i)* を定義するベクトルの格納形式の例を、 $n=5$ 、 $k=3$  において次の図に示す。1 に等しい成分は格納されない。対応する配列成分は変更されるが、終了時に復元される。配列の残りの部分は使用されない。

*direct* = 'F' および *storev* = 'C':

$$V(iv:iv+n-1, jv:jv+k-1) = \begin{bmatrix} 1 & & & & \\ v1 & 1 & & & \\ v1 & v2 & 1 & & \\ v1 & v2 & v3 & & \\ v1 & v2 & v3 & & \end{bmatrix}$$

*direct* = 'F' および *storev* = 'R':

$$V(iv:iv+k-1, jv:jv+n-1) = \begin{bmatrix} 1 & v1 & v1 & v1 & v1 \\ & 1 & v2 & v2 & v2 \\ & & 1 & v3 & v3 \end{bmatrix}$$

*direct* = 'B' および *storev* = 'C':

$$V(iv:iv+n-1, jv:jv+k-1) = \begin{bmatrix} v1 & v2 & v3 \\ v1 & v2 & v3 \\ 1 & v2 & v3 \\ & 1 & v3 \\ & & 1 \end{bmatrix}$$

*direct* = 'B' および *storev* = 'R':

$$V(iv:iv+k-1, jv:jv+n-1) = \begin{bmatrix} v1 & v1 & 1 \\ v2 & v2 & v2 & 1 \\ v3 & v3 & v3 & v3 & 1 \end{bmatrix}$$

## p?larz

p?tzrzf が返したとおりの基本リフレクタを一般行列に適用する。

### 構文

```
call pslarz(side, m, n, l, v, iv, jv, descv, incv, tau, c, ic, jc, desc, work)
```

```
call pdlarz(side, m, n, l, v, iv, jv, descv, incv, tau, c, ic, jc, descc,
            work)
call pclarz(side, m, n, l, v, iv, jv, descv, incv, tau, c, ic, jc, descc,
            work)
call pzlarz(side, m, n, l, v, iv, jv, descv, incv, tau, c, ic, jc, descc,
            work)
```

## 説明

このルーチンは、実 / 複素基本リフレクタ  $Q$  (または  $Q^T$ ) を、 $m \times n$  の実 / 複素分布行列  $\text{sub}(C) = C(ic:ic+m-1, jc:jc+n-1)$  に左または右のいずれかから適用する。 $Q$  は次の形式で表現される。

$$Q = I - \tau * v * v'$$

$\tau$  は実 / 複素スカラ、 $v$  は実 / 複素ベクトルである。

$\tau = 0$  の場合、 $Q$  は単位行列をとる。

$Q$  は [p?tzrzf](#) から返されたとおり、 $k$  個の基本リフレクタの積である。

## 入力パラメータ

*side* (グローバル) CHARACTER。  
*side* = 'L' の場合、形式  $Q * \text{sub}(C)$ 。  
*side* = 'R' の場合、形式  $\text{sub}(C) * Q$ ,  $Q = Q^T$  (実数型の場合)。

*m* (グローバル) INTEGER。  
 演算を行う行数、すなわち、分散部分行列  $\text{sub}(C)$  の行数。( $m \geq 0$ )。

*n* (グローバル) INTEGER。  
 演算を行う列数、すなわち、分散部分行列  $\text{sub}(C)$  の列数。( $n \geq 0$ )。

*l* (グローバル) INTEGER。  
 Householder リフレクタとして意味を持った部分が格納されている分散部分行列  $\text{sub}(A)$  の列数。*side* = 'L' の場合、 $m \geq l \geq 0$ 。  
*side* = 'R' の場合、 $n \geq l \geq 0$ 。

*v* (ローカル)  
 REAL (pslarz の場合)  
 DOUBLE PRECISION (pdlarz の場合)  
 COMPLEX (pclarz の場合)

COMPLEX\*16 (pzlarz の場合)。ローカルメモリにある、Householder 変換  $Q$  を表す分散行列  $v$  のローカル部分を格納する、次元 ( $lld_v, *$ ) の配列へのポインタ。

$side = 'L'$ 、 $incv = 1$  の場合、 $v(iv:iv+l-1, jv)$ 。  
 $side = 'L'$ 、 $incv = m_v$  の場合、 $v(iv, jv:jv+l-1)$ 。  
 $side = 'R'$ 、 $incv = 1$  の場合、 $v(iv:iv+l-1, jv)$ 。  
 $side = 'R'$ 、 $incv = m_v$  の場合、 $v(iv, jv:jv+l-1)$ 。

$Q$  の表現におけるベクトル  $v$ 。  $tau = 0$  の場合、 $v$  は使用されない。

$iv, jv$	(グローバル) INTEGER。それぞれ、部分行列 $sub(V)$ の最初の行と最初の列を示す、グローバル配列 $V$ の行インデックスと列インデックス。
$descv$	(グローバルおよびローカル) INTEGER。配列、次元は ( $dlen_v$ )。分散行列 $V$ の配列ディスクリプタ。
$incv$	(グローバル) INTEGER。 $v$ の成分のグローバルな増分。このバージョンでは、1 と $m_v$ の 2 つの $incv$ の値のみサポートされる。 $incv$ の値は、ゼロであってはならない。
$tau$	(ローカル) REAL (pslarz の場合) DOUBLE PRECISION (pdlarz の場合) COMPLEX (pcclarz の場合) COMPLEX*16 (pzlarz の場合)。 配列、次元は $LOCc(jv)$ ( $incv = 1$ の場合) または $LOCr(iv)$ (それ以外の場合)。Householder ベクトルに関する Householder スカラを格納する。 $tau$ は、分散行列 $V$ に関連付けられる。
$c$	(ローカル) REAL (pslarz の場合) DOUBLE PRECISION (pdlarz の場合) COMPLEX (pcclarz の場合) COMPLEX*16 (pzlarz の場合)。 ローカルメモリにある、 $sub(C)$ のローカル部分を格納する、次元 ( $lld_c, LOCc(jc+n-1)$ ) の配列へのポインタ。
$ic, jc$	(グローバル) INTEGER。それぞれ、部分行列 $sub(C)$ の最初の行と最初の列を示す、グローバル配列 $C$ の行インデックスと列インデックス。

*descc* (グローバルおよびローカル) INTEGER。配列、次元は (*dlen\_*)。分散行列 *C* の配列ディスクリプタ。

*work* (ローカル)

REAL (pslarz の場合)  
 DOUBLE PRECISION (pdlarz の場合)  
 COMPLEX (pcclarz の場合)  
 COMPLEX\*16 (pzlarz の場合)。  
 配列、次元は (*lwork*)。

```

If incv = 1,
  if side = 'L',
    if ivcol = iccol,
      lwork ≥ NqC0
    else
      lwork ≥ MpC0 + max( 1, NqC0 )
    end if
else if side = 'R',
  lwork ≥ NqC0 + max( max( 1, MpC0 ), numroc( numroc(
    n+icoffc,nb_v,0,0,npcol ),nb_v,0,0,lcmq ) )
end if
else if incv = m_v,
  if side = 'L',
    lwork ≥ MpC0 + max( max( 1, NqC0 ), numroc( numroc(
m+iroffc,mb_v,0,0,nprow ),mb_v,0,0,lcmp ) )
  else if side = 'R',
    if ivrow = icrow,
      lwork ≥ MpC0
    else
      lwork ≥ NqC0 + max( 1, MpC0 )
    end if
  end if
end if,
lcm は nprow と npcol の最小公倍数である。
lcm = ilcm( nprow, npcol ), lcmp = lcm / nprow,
lcmq = lcm / npcol,

iroffc = mod( ic-1, mb_c ), icoffc = mod( jc-1, nb_c ),
icrow = indxg2p( ic, mb_c, myrow, rsrc_c, nprow ),
iccol = indxg2p( jc, nb_c, mycol, csrc_c, npcol ),
mpc0 = numroc( m+iroffc, mb_c, myrow, icrow, nprow ),
nqc0 = numroc( n+icoffc, nb_c, mycol, iccol, npcol )。

```

`ilcm`、`indxg2p` および `numroc` は、ScaLAPACK ツール関数である。  
`myrow`、`mycol`、`nprow`、および `npcol` は、サブルーチン  
`blacs_gridinfo` を呼び出して求められる。

### 出力パラメータ

`c` (ローカル) 終了時に、`sub(C)` は  $Q * \text{sub}(C)$  (`side = 'L'` の場合) または  
 $\text{sub}(C) * Q$  (`side = 'R'` の場合) で上書きされる。

---

## p?larzb

`p?tzrzb` が返したとおりのブロック・リフレクタ  
またはその転置/共役転置を、一般行列に適用す  
る。

---

### 構文

```
call pslarzb(side, trans, direct, storev, m, n, k, l, v, iv, jv, descv, t, c,  
            ic, jc, descc, work)  
call pdlarzb(side, trans, direct, storev, m, n, k, l, v, iv, jv, descv, t, c,  
            ic, jc, descc, work)  
call pclarzb(side, trans, direct, storev, m, n, k, l, v, iv, jv, descv, t, c,  
            ic, jc, descc, work)  
call pzlarzb(side, trans, direct, storev, m, n, k, l, v, iv, jv, descv, t, c,  
            ic, jc, descc, work)
```

### 説明

このルーチンは、実/複素ブロック・リフレクタ  $Q$  またはその転置  $Q^T$  (複素数型の場  
合、共役転置  $Q^H$ ) を、 $m \times n$  の実/複素分布行列  $\text{sub}(C) = C(\text{ic}:\text{ic}+m-1, \text{jc}:\text{jc}+n-1)$  に左  
または右のいずれかから適用する。

$Q$  は `p?tzrzb` から返されたとおり、 $k$  個の基本リフレクタの積である。

現時点で、`storev = 'R'` と `direct = 'B'` のみがサポートされている。

### 入力パラメータ

`side` (グローバル) CHARACTER。

	$side = 'L'$ の場合、 $Q$ または $Q^T$ (複素数型の場合 $Q^H$ ) を左から適用する。 $side = 'R'$ の場合、 $Q$ または $Q^T$ (複素数型の場合 $Q^H$ ) を右から適用する。
<i>trans</i>	(グローバル) CHARACTER。 $trans = 'N'$ の場合、転置なし。 $Q$ を適用する。 $trans = 'T'$ の場合、転置。 $Q^T$ (実数型) を適用する。 $trans = 'C'$ の場合、共役転置。 $Q^H$ (複素数型) を適用する。
<i>direct</i>	(グローバル) CHARACTER。 基本リフレクタの積からどのように $H$ が表現されているかを示す。 $direct = 'F'$ の場合、 $H = H(1) H(2) \dots H(k)$ (順方向、現時点でこの機能はサポートされていない)。 $direct = 'B'$ の場合、 $H = H(k) \dots H(2) H(1)$ (逆方向)。
<i>storev</i>	(グローバル) CHARACTER。 基本リフレクタを定義するベクトルがどのように格納されているかを示す。 $storev = 'C'$ の場合、列方向 (現時点でこの機能はサポートされていない)。 $storev = 'R'$ の場合、行方向。
<i>m</i>	(グローバル) INTEGER。 演算を行う行数、すなわち、分散部分行列 $\text{sub}(C)$ の行数。( $m \geq 0$ )。
<i>n</i>	(グローバル) INTEGER。 演算を行う列数、すなわち、分散部分行列 $\text{sub}(C)$ の列数。( $n \geq 0$ )。
<i>k</i>	(グローバル) INTEGER。 行列 $T$ の次数 (積がブロック・リフレクタを定義する基本リフレクタの個数に等しい)。
<i>l</i>	(グローバル) INTEGER。 Householder リフレクタとして意味を持った部分が格納されている分散部分行列 $\text{sub}(A)$ の列数。 $side = 'L'$ の場合、 $m \geq l \geq 0$ 。 $side = 'R'$ の場合、 $n \geq l \geq 0$ 。
<i>v</i>	(ローカル)

	<p>REAL (pslarzb の場合 )  DOUBLE PRECISION (pdlarzb の場合 )  COMPLEX (pclarzb の場合 )  COMPLEX*16 (pzlarzb の場合 )。  ローカルメモリにある、次元 ( <math>lld\_v</math>, <math>LOCc(jv+m-1)</math> ) ( <math>side = 'L'</math> の場合 ) または ( <math>lld\_v</math>, <math>LOCc(jv+m-1)</math> ) ( <math>side = 'R'</math> の場合 ) の配列へのポインタ。  <math>p?tzrzb</math> が返したとおりの Householder 変換を表す乱数ベクトル <math>V</math> のローカル部分を格納する。</p> <p><math>lld\_v \geq LOCc(iv+k-1)</math>。</p>
$iv, jv$	( グローバル ) INTEGER。それぞれ、部分行列 $sub(V)$ の最初の行と最初の列を示す、グローバル配列 $V$ の行インデックスと列インデックス。
$descv$	( グローバルおよびローカル ) INTEGER。配列、次元は ( $dlen\_v$ )。分散行列 $V$ の配列ディスクリプタ。
$t$	( ローカル )  REAL (pslarzb の場合 ) DOUBLE PRECISION (pdlarzb の場合 ) COMPLEX (pclarzb の場合 ) COMPLEX*16 (pzlarzb の場合 )。 配列、次元は ( $mb\_v$ , $mb\_v$ )。  ブロック・リフレクタの表現中にある下三角行列 $T$ 。
$c$	( ローカル ) REAL (pslarfb の場合 ) DOUBLE PRECISION (pdlarfb の場合 ) COMPLEX (pclarfb の場合 ) COMPLEX*16 (pzlarfb の場合 )。 ローカルメモリにある、次元 ( $lld\_c$ , $LOCc(jc+n-1)$ ) の配列へのポインタ。 $m \times n$ の分散行列 $sub(C)$ を格納する。
$ic, jc$	( グローバル ) INTEGER。それぞれ、部分行列 $sub(C)$ の最初の行と最初の列を示す、グローバル配列 $c$ の行インデックスと列インデックス。
$descC$	( グローバルおよびローカル ) INTEGER。配列、次元は ( $dlen\_c$ )。分散行列 $C$ の配列ディスクリプタ。
$work$	( ローカル )



REAL (pslarzb の場合)  
 DOUBLE PRECISION (pdlarzb の場合)  
 COMPLEX (pclarzb の場合)  
 COMPLEX\*16 (pzlarzb の場合)。  
 配列、次元は (*lwork*)。

```

If storev='C',
  if side='L',
    lwork ≥ ( NqC0 + MpC0 ) * k
  else if side='R',
    lwork ≥ ( NqC0 + max( NpV0 + numroc( numroc(
      n+icoffc, nb_v, 0, 0, npcol ), nb_v, 0, 0
      ,lcmq ), mpc0 ) ) * k
  end if
else if storev='R',
  if side='L',
    lwork ≥ ( mpc0 + max( mqv0 + numroc( numroc( m+iroffc,
      mb_v, 0, 0, nprow ), mb_v, 0, 0, lcmq ),
      nqc0 ) ) * k
  else if side='R',
    lwork ≥ ( MpC0 + NqC0 ) * k
  end if
end if,

```

ここで、*lcmq* = *lcm* / *npcol*、*lcm* = *iclm*( *nprow*, *npcol* )、  
*iroffv* = *mod*( *iv*-1, *mb\_v* )、*icoffv* = *mod*( *jv*-1, *nb\_v* )、  
*ivrow* = *indxg2p*( *iv*, *mb\_v*, *myrow*, *rsrc\_v*, *nprow* )、  
*ivcol* = *indxg2p*( *jv*, *nb\_v*, *mycol*, *csrc\_v*, *npcol* )、  
*MqV0* = *numroc*( *m+icoffv*, *nb\_v*, *mycol*, *ivcol*, *npcol* )、  
*NpV0* = *numroc*( *n+iroffv*, *mb\_v*, *myrow*, *ivrow*, *nprow* )、  
*iroffc* = *mod*( *ic*-1, *mb\_c* )、*icoffc* = *mod*( *jc*-1, *nb\_c* )、  
*icrow* = *indxg2p*( *ic*, *mb\_c*, *myrow*, *rsrc\_c*, *nprow* )、  
*iccol* = *indxg2p*( *jc*, *nb\_c*, *mycol*, *csrc\_c*, *npcol* )、  
*MpC0* = *numroc*( *m+iroffc*, *mb\_c*, *myrow*, *icrow*, *nprow* )、  
*NpC0* = *numroc*( *n+icoffc*, *mb\_c*, *myrow*, *icrow*, *nprow* )、  
*NqC0* = *numroc*( *n+icoffc*, *nb\_c*, *mycol*, *iccol*, *npcol* )。

*ilcm*、*indxg2p* および *numroc* は、ScaLAPACK ツール関数である。  
*myrow*、*mycol*、*nprow*、および *npcol* は、サブルーチン  
*blacs\_gridinfo* を呼び出して求められる。

## p?larzc

p?tzzrf が返したとおりの基本リフレクタの共役転置を、一般行列に適用(乗算)する。

### 構文

```
call pclarzc(side, m, n, l, v, iv, jv, descv, incv, tau, c, ic, jc,
             descc, work)
call pzlarzc(side, m, n, l, v, iv, jv, descv, incv, tau, c, ic, jc,
             descc, work)
```

### 説明

このルーチンは、複素基本リフレクタ  $Q^H$  を、 $m \times n$  の複素分布行列  $\text{sub}(C) = C(ic:ic+m-1, jc:jc+n-1)$  に左または右のいずれかから適用する。 $Q$  は次の形式で表現される。

$$Q = I - \tau \cdot v \cdot v'$$

$\tau$  は複素スカラ、 $v$  は複素ベクトルである。

$\tau = 0$  の場合、 $Q$  は単位行列をとる。

$Q$  は [p?tzzrf](#) から返されたとおり、 $k$  個の基本リフレクタの積である。

### 入力パラメータ

<i>side</i>	(グローバル) CHARACTER。  <i>side</i> = 'L' の場合、形式 $Q^H * \text{sub}(C)$ 。 <i>side</i> = 'R' の場合、形式 $\text{sub}(C) * Q^H$ 。
<i>m</i>	(グローバル) INTEGER。 演算を行う行数、すなわち、分散部分行列 $\text{sub}(C)$ の行数。( $m \geq 0$ )。
<i>n</i>	(グローバル) INTEGER。 演算を行う列数、すなわち、分散部分行列 $\text{sub}(C)$ の列数。( $n \geq 0$ )。
<i>l</i>	(グローバル) INTEGER。  Householder リフレクタとして意味を持った部分が格納されている分散部分行列 $\text{sub}(A)$ の列数。

	$side = 'L'$ の場合、 $m \geq l \geq 0$ 。 $side = 'R'$ の場合、 $n \geq l \geq 0$ 。
$v$	(ローカル)  COMPLEX (pclarzc の場合) COMPLEX*16 (pzlarzc の場合)。 ローカルメモリにある、Householder 変換 $Q$ を表す分散行列 $v$ のローカル部分を格納する、次元 ( $lld_v, *$ ) の配列へのポインタ。  $side = 'L'$ 、 $incv = 1$ の場合、 $v(iv:iv+l-1, jv)$ 。 $side = 'L'$ 、 $incv = m_v$ の場合、 $v(iv, jv:jv+l-1)$ 。 $side = 'R'$ 、 $incv = 1$ の場合、 $v(iv:iv+l-1, jv)$ 。 $side = 'R'$ 、 $incv = m_v$ の場合、 $v(iv, jv:jv+l-1)$ 。  $Q$ の表現におけるベクトル $v$ 。 $tau = 0$ の場合、 $v$ は使用されない。
$iv, jv$	(グローバル) INTEGER。それぞれ、部分行列 $sub(V)$ の最初の行と最初の列を示す、グローバル配列 $V$ の行インデックスと列インデックス。
$descv$	(グローバルおよびローカル) INTEGER。配列、次元は ( $dlen_v$ )。分散行列 $V$ の配列ディスクリプタ。
$incv$	(グローバル) INTEGER。 $v$ の成分のグローバルな増分。このバージョンでは、1 と $m_v$ の 2 つの $incv$ の値のみサポートされる。  $incv$ の値は、ゼロであってはならない。
$tau$	(ローカル) COMPLEX (pclarzc の場合) COMPLEX*16 (pzlarzc の場合)。 配列、次元は $LOCc(jv)$ ( $incv = 1$ の場合) または $LOCr(iv)$ (それ以外の場合)。Householder ベクトルに関する Householder スカラを格納する。  $tau$ は、分散行列 $V$ に関連付けられる。
$c$	(ローカル) COMPLEX (pclarzc の場合) COMPLEX*16 (pzlarzc の場合)。 ローカルメモリにある、 $sub(C)$ のローカル部分を格納する、次元 ( $lld_c, LOCc(jc+n-1)$ ) の配列へのポインタ。
$ic, jc$	(グローバル) INTEGER。それぞれ、部分行列 $sub(C)$ の最初の行と最初の列を示す、グローバル配列 $C$ の行インデックスと列インデックス。

*descc* (グローバルおよびローカル) INTEGER。配列、次元は (*dlen\_*)。分散行列 *C* の配列ディスクリプタ。

*work* (ローカル)

```
If incv=1,
  if side='L',
    if ivcol=iccol,
      lwork ≥ NqC0
    else
      lwork ≥ MpC0 + max( 1, NqC0 )
    end if
  else if side='R',
    lwork ≥ nqc0 + max( max( 1, mpc0 ), numroc( numroc(
      n+icoffc,nb_v,0,0,npcol ),nb_v,0,0,lcmq ) )
  end if
else if incv=m_v,
  if side='L',
    lwork ≥ mpc0 + max( max( 1, nqc0 ), numroc( numroc(
      m+iroffc,mb_v,0,0,nprow ),mb_v,0,0,lcmp ) )
  else if side='R',
    if ivrow=icrow,
      lwork ≥ mpc0
    else
      lwork ≥ nqc0 + max( 1, mpc0 )
    end if
  end if
end if,
```

*lcm* は *nprow* と *npcol* の最小公倍数である。

*lcm* = *ilcm*(*nprow*, *npcol*)、*lcmp* = *lcm* / *nprow*、  
*lcmq* = *lcm* / *npcol*、

*iroffc* = mod( *ic*-1, *mb\_c* )、*icoffc* = mod( *jc*-1, *nb\_c* )、  
*icrow* = *indxg2p*( *ic*, *mb\_c*, *myrow*, *rsrc\_c*, *nprow* )、  
*iccol* = *indxg2p*( *jc*, *nb\_c*, *mycol*, *csrc\_c*, *npcol* )、  
*MpC0* = *numroc*( *m+iroffc*, *mb\_c*, *myrow*, *icrow*, *nprow* )、  
*NqC0* = *numroc*( *n+icoffc*, *nb\_c*, *mycol*, *iccol*, *npcol* )。

*ilcm*, *indxg2p*、および *numroc* は、ScaLAPACK ツール関数である。

*myrow*, *mycol*, *nprow*、および *npcol* は、サブルーチン *blacs\_gridinfo* を呼び出して求められる。

## p?larzt

p?tzzrf が返したとおりのブロック・リフレクタ  
 $H=I-VTV^H$  の三角係数  $T$  を生成する。

### 構文

```
call pslarzt(direct, storev, n, k, v, iv, jv, descv, tau, t, work)
call pdlarzt(direct, storev, n, k, v, iv, jv, descv, tau, t, work)
call pclarzt(direct, storev, n, k, v, iv, jv, descv, tau, t, work)
call pzlarzt(direct, storev, n, k, v, iv, jv, descv, tau, t, work)
```

### 説明

このルーチンは、[p?tzzrf](#) によって返される  $k$  個の基本リフレクタの積として定義されている次数  $> n$  の実/複素ブロック・リフレクタ  $H$  の三角係数  $T$  を生成する。

$direct = 'F'$  の場合、 $H = H(1)H(2) \dots H(k)$ 、 $T$  は上三角。

$direct = 'B'$  の場合、 $H = H(k) \dots H(2)H(1)$ 、 $T$  は下三角。

$storev = 'C'$  の場合、基本リフレクタ  $H(i)$  を定義するベクトルは、配列  $v$  の  $i$  番目の列に次のように格納される。

$$H = i - v * t * v'$$

$storev = 'C'$  の場合、基本リフレクタ  $H(i)$  を定義するベクトルは、配列  $v$  の  $i$  番目の行に次のように格納される。

$$H = i - v' * t * v$$

現時点で、 $storev = 'R'$  と  $direct = 'B'$  のみがサポートされている。

### 入力パラメータ

**direct** (グローバル) CHARACTER。  
 ブロック・リフレクタを生成するために基本リフレクタを乗算する次数を指定する。  
 $direct = 'F'$  の場合、 $H = H(1)H(2) \dots H(k)$  (順方向、現時点でこの機能はサポートされていない)。  
 $direct = 'B'$  の場合、 $H = H(k) \dots H(2)H(1)$  (逆方向)。

<i>storev</i>	(グローバル) CHARACTER。 基本リフレクタを定義するベクトルがどのように格納されているかを指定する。  <i>storev</i> = 'C' の場合、列方向 (現時点でこの機能はサポートされていない)。 <i>storev</i> = 'R' の場合、行方向。
<i>n</i>	(グローバル) INTEGER。 ブロック・リフレクタ <i>H</i> の次数。 $n \geq 0$ 。
<i>k</i>	(グローバル) INTEGER。 三角係数 <i>T</i> の次数 (基本リフレクタの個数に等しい)。 $1 \leq k \leq mb\_v (= nb\_v)$ 。
<i>v</i>	REAL (pslarzt の場合) DOUBLE PRECISION (pdlarzt の場合) COMPLEX (pclarzt の場合) COMPLEX*16 (pzlarzt の場合) ローカルメモリにある、ローカル次元 ( <i>LOCr</i> ( <i>iv</i> + <i>k</i> -1), <i>LOCc</i> ( <i>jv</i> + <i>n</i> -1)) の配列へのポインタ。  分散行列 <i>V</i> は Householder ベクトルを格納する。次の「アプリケーション・ノート」を参照。
<i>iv, jv</i>	(グローバル) INTEGER。それぞれ、部分行列 sub( <i>V</i> ) の最初の行と最初の列を示す、グローバル配列 <i>V</i> の行インデックスと列インデックス。
<i>descv</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen</i> )。分散行列 <i>V</i> の配列ディスクリプタ。
<i>tau</i>	(ローカル) REAL (pslarzt の場合) DOUBLE PRECISION (pdlarzt の場合) COMPLEX (pclarzt の場合) COMPLEX*16 (pzlarzt の場合)。 配列、次元は <i>LOCr</i> ( <i>iv</i> + <i>k</i> -1) ( <i>incv</i> = <i>m_v</i> の場合) または <i>LOCc</i> ( <i>jv</i> + <i>k</i> -1) (それ以外の場合)。Householder ベクトルに関する Householder スカラを格納する。  <i>tau</i> は、分散行列 <i>V</i> に関連付けられる。
<i>work</i>	(ローカル)

REAL (pslarzt の場合 )  
 DOUBLE PRECISION (pdlarzt の場合 )  
 COMPLEX (pclarzt の場合 )  
 COMPLEX\*16 (pzlarzt の場合 )。  
 ワークスペース配列、次元は  $(k*(k-1)/2)$ 。

## 出力パラメータ

$v$  REAL (pslarzt の場合 )  
 DOUBLE PRECISION (pdlarzt の場合 )  
 COMPLEX (pclarzt の場合 )  
 COMPLEX\*16 (pzlarzt の場合 )。  
 $t$  (ローカル)  
 REAL (pslarzt の場合 )  
 DOUBLE PRECISION (pdlarzt の場合 )  
 COMPLEX (pclarzt の場合 )  
 COMPLEX\*16 (pzlarzt の場合 )。  
 配列、次元は  $(mb_v, mb_v)$ 。 $v$  に関連するブロック・リフレクタの  $k \times k$  の三角係数が格納される。 $t$  は下三角。

## アプリケーション・ノート

行列  $V$  の形状と  $H(i)$  を定義するベクトルの格納形式の例を次に示す ( $n=5$ 、 $k=3$  の場合)。1 に等しい成分は格納されない。対応する配列成分は変更されるが、終了時に復元される。配列の残りの部分は使用されない。

$direct = 'F'$  および  $storev = 'C'$ :

$$\begin{bmatrix} v1 & v2 & v3 \\ v1 & v2 & v3 \\ v1 & v2 & v3 \\ v1 & v2 & v3 \\ v1 & v2 & v3 \end{bmatrix}$$

$$V = \begin{bmatrix} . & . & . \\ . & . & . \\ 1 & . & . \\ . & 1 & . \\ . & . & 1 \end{bmatrix}$$

$direct = 'F'$  および  $storev = 'R'$ :

$$V \begin{bmatrix} \overbrace{v1 \ v1 \ v1 \ v1 \ v1} & . & . & . & . & 1 \\ v2 \ v2 \ v2 \ v2 \ v2 & . & . & . & 1 \\ v3 \ v3 \ v3 \ v3 \ v3 & . & . & 1 \end{bmatrix}$$

$direct = 'B'$  および  $storev = 'C'$ :

$$V = \begin{matrix} 1 \\ . \ 1 \\ . \ . \ 1 \\ . \ . \ . \\ . \ . \ . \end{matrix} \begin{bmatrix} v1 \ v2 \ v3 \\ v1 \ v2 \ v3 \\ v1 \ v2 \ v3 \\ v1 \ v2 \ v3 \\ v1 \ v2 \ v3 \end{bmatrix}$$

$direct = 'B'$  および  $storev = 'R'$ :

$$V \begin{matrix} 1 \ . \ . \ . \ . \\ . \ 1 \ . \ . \ . \\ . \ . \ 1 \ . \ . \end{matrix} \begin{bmatrix} \overbrace{v1 \ v1 \ v1 \ v1 \ v1} \\ v2 \ v2 \ v2 \ v2 \ v2 \\ v3 \ v3 \ v3 \ v3 \ v3 \end{bmatrix}$$



## p?lascl

一般矩形行列に  $c_{to}/c_{from}$  として定義される実スカラを掛ける。

### 構文

```
call pslascl(type, cfrom, cto, m, n, a, ia, ja, desca, info)
call pdlascl(type, cfrom, cto, m, n, a, ia, ja, desca, info)
call pclascl(type, cfrom, cto, m, n, a, ia, ja, desca, info)
call pzlascl(type, cfrom, cto, m, n, a, ia, ja, desca, info)
```

### 説明

このルーチンは、 $m \times n$  の実 / 複素行列  $\text{sub}(A)$  に実 / 複素スカラ  $cto/cfrom$  を掛ける。ここで、 $\text{sub}(A)$  は  $A(ia:ia+m-1, ja:ja+n-1)$  である。最終結果  $cto * A(i, j) / cfrom$  がオーバーフロー / アンダーフローしない限り、演算はオーバーフロー / アンダーフローなしで実行される。

$\text{sub}(A)$  の形式は  $type$  によって、フル、上三角、下三角、または上 Hessenberg から指定される。

### 入力パラメータ

$type$  (グローバル) CHARACTER。

入力分散行列の格納形式の  $type$ 。

$type = 'G'$  の場合、 $\text{sub}(A)$  はフル行列。

$type = 'L'$  の場合、 $\text{sub}(A)$  は下三角行列。

$type = 'U'$  の場合、 $\text{sub}(A)$  は上三角行列。

$type = 'H'$  の場合、 $\text{sub}(A)$  は上 Hessenberg 行列。

$cfrom, cto$  (グローバル)

REAL (pslascl/pclascl の場合)

DOUBLE PRECISION (pdlascl/pzlascl の場合)。

分散行列  $\text{sub}(A)$  に  $cto/cfrom$  を掛ける。最終結果  $cto * A(i, j) / cfrom$  がオーバーフロー / アンダーフローなしで表現される場合、 $A(i, j)$  はオーバーフロー / アンダーフローなしで計算される。 $cfrom$  は非ゼロでなければならない。

<i>m</i>	(グローバル) INTEGER。 演算を行う行数、すなわち、分散部分行列 <i>sub(A)</i> の行数。( $m \geq 0$ )。
<i>n</i>	(グローバル) INTEGER。 演算を行う列数、すなわち、分散部分行列 <i>sub(A)</i> の列数。( $n \geq 0$ )。
<i>a</i>	(ローカル入力/ローカル出力)  REAL (pslascl の場合) DOUBLE PRECISION (pdlascl の場合) COMPLEX (pcblascl の場合) COMPLEX*16 (pzlascl の場合)。  ローカルメモリにある、次元 ( <i>lld_a</i> , <i>LOCc(ja+n-1)</i> ) の配列へのポインタ。  分散行列 <i>sub(A)</i> のローカル部分を格納する。
<i>ia</i> , <i>ja</i>	(グローバル) INTEGER。 それぞれ、部分行列 <i>sub(A)</i> の最初の行と最初の列を示す、グローバル配列 <i>A</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。  次元 ( <i>dlen_</i> ) の配列。分散行列 <i>A</i> の配列ディスクリプタ。

## 出力パラメータ

<i>a</i>	(ローカル) 終了時に、 <i>cto/cfrom</i> が掛けられる分散行列のローカル部分が格納される。
<i>info</i>	(ローカル) INTEGER。  <i>info</i> = 0 の場合、正常に終了したことを示す。  <i>info</i> < 0 の場合、 <i>i</i> 番目の引数が配列で <i>j</i> 番目の成分の値が不正ならば <i>info</i> = - ( <i>i</i> *100+ <i>j</i> )、 <i>i</i> 番目の引数がスカラで値が不正ならば <i>info</i> = - <i>i</i> 。

## p?laset

行列の非対角成分を  $\alpha$  に、対角成分を  $\beta$  に初期化する。

### 構文

```
call pslaset(uplo, m, n, alpha, beta, a, ia, ja, desca)
call pdlaset(uplo, m, n, alpha, beta, a, ia, ja, desca)
call pclaset(uplo, m, n, alpha, beta, a, ia, ja, desca)
call pzlaset(uplo, m, n, alpha, beta, a, ia, ja, desca)
```

### 説明

このルーチンは、 $m \times n$  の分散行列  $\text{sub}(A)$  で対角を  $\text{beta}$  に、対角外を  $\text{alpha}$  に初期化する。ここで、 $\text{sub}(A)$  は  $A(\text{ia}:\text{ia}+m-1, \text{ja}:\text{ja}+n-1)$  である。

### 入力パラメータ

**uplo** (グローバル) CHARACTER。  
設定対象の分散行列  $\text{sub}(A)$  の部分を指定する。  
 $\text{uplo} = 'U'$  の場合、上三角部分が設定される。 $\text{sub}(A)$  の厳密な下三角部分は変更されない。  
 $\text{uplo} = 'L'$  の場合、下三角部分が設定される。 $\text{sub}(A)$  の厳密な上三角部分は変更されない。  
それ以外の場合、行列  $\text{sub}(A)$  のすべての部分が設定される。

**m** (グローバル) INTEGER。  
演算を行う行数、すなわち、分散部分行列  $\text{sub}(A)$  の行数。( $m \geq 0$ )。

**n** (グローバル) INTEGER。  
演算を行う列数、すなわち、分散部分行列  $\text{sub}(A)$  の列数。( $n \geq 0$ )。

**alpha** (グローバル)  
REAL (pslaset の場合)  
DOUBLE PRECISION (pdlaset の場合)  
COMPLEX (pclaset の場合)  
COMPLEX\*16 (pzlaset の場合)。

非対角成分に設定する定数。

*beta* (グローバル)

REAL (pslaset の場合)  
 DOUBLE PRECISION (pdlaset の場合)  
 COMPLEX (pclaset の場合)  
 COMPLEX\*16 (pzlaset の場合)。

対角成分に設定する定数。

## 出力パラメータ

*a* (ローカル)

REAL (pslaset の場合)  
 DOUBLE PRECISION (pdlaset の場合)  
 COMPLEX (pclaset の場合)  
 COMPLEX\*16 (pzlaset の場合)。

ローカルメモリにある、次元 (*lld\_a*, *LOCc(ja+n-1)*) の配列へのポインタ。設定する分散行列 *sub(A)* のローカル部分が格納される。終了時に、 $m \times n$  の部分行列 *sub(A)* の先頭は、次のように設定される。

*uplo* = 'U' の場合、 $A(ia+i-1, ja+j-1) = \alpha$  ( $1 < i \leq j-1$ ,  $1 \leq j \leq n$ )、  
*uplo* = 'L' の場合、 $A(ia+i-1, ja+j-1) = \alpha$  ( $j+1 \leq i \leq m$ ,  $1 \leq j \leq n$ )、  
 それ以外の場合、 $A(ia+i-1, ja+j-1) = \alpha$   
 ( $1 \leq i \leq m$ ,  $1 \leq j \leq n$ ,  $ia+i.ne.ja+j$ )、すべての *uplo* で、  
 $A(ia+i-1, ja+i-1) = \beta$  ( $1 \leq i \leq \min(m, n)$ )

*ia*, *ja* (グローバル) INTEGER。  
 それぞれ、部分行列 *sub(A)* の最初の行と最初の列を示す、グローバル配列 *A* の行インデックスと列インデックス。

*desca* (グローバルおよびローカル) INTEGER。  
 次元 (*dlen\_*) の配列。分散行列 *A* の配列ディスクリプタ。

## p?lasmsub

安全に 0 に設定できる小さい劣対角成分を行列の一番下から検出する。

### 構文

```
call pslasmsub(a, desca, i, l, k, smlnum, buf, lwork)
call pdlasmsub(a, desca, i, l, k, smlnum, buf, lwork)
```

### 説明

このルーチンは、安全に 0 に設定できる小さい劣対角成分を行列の一番下から検出する。このルーチンは、グローバルな最大値を検出するため、すべてのプロセスで呼び出されなければならない。

### 入力パラメータ

<i>a</i>	(グローバル)  REAL (pslasmsub の場合) DOUBLE PRECISION (pdlasmsub の場合)。 配列、次元は ( <i>desca</i> ( <i>lld</i> ),*)。 三重対角部分をスキャンされた Hessenberg 行列を格納する。終了時に変更されない。
<i>desca</i>	(グローバルおよびローカル) INTEGER。 次元 ( <i>dlen</i> ) の配列。 分散行列 <i>A</i> の配列ディスクリプタ。
<i>i</i>	(グローバル) INTEGER。 <i>A</i> の縮退されていない部分行列の一番下のグローバル位置。終了時に変更されない。
<i>l</i>	(グローバル) INTEGER。 <i>A</i> の縮退されていない部分行列の一番上のグローバル位置。終了時に変更されない。
<i>smlnum</i>	(グローバル)  REAL (pslasmsub の場合) DOUBLE PRECISION (pdlasmsub の場合)。 与えられた行列に対する「小さい数」を格納する。終了時に変更されない。

*lwork* (グローバル) INTEGER。  
終了時に、*lwork* は、*work* バッファのサイズである。  
この値は、 $2 * \text{ceil}(\text{ceil}((i-1)/hbl)/lcm(nprow, npcol))$  以上でなければならない。*lcm* は公倍数以上で、*nprow* x *npcol* は論理グリッドサイズである。

## 出力パラメータ

*k* (グローバル) INTEGER。  
終了時に、*A* の縮退されていない部分行列の一番下が格納される。次の条件を満たす。 $1 \leq m \leq i-1$

*buf* (ローカル)  
REAL (*pslasmsub* の場合)  
DOUBLE PRECISION (*pdlasmsub* の場合)。  
サイズ *lwork* の配列。

---

## p?lassq

スケーリング形式で表現された二乗和を更新する。

---

### 構文

```
call pslasq(n, x, ix, jx, descx, incx, scale, sumsq)
call pdlasq(n, x, ix, jx, descx, incx, scale, sumsq)
call pclasq(n, x, ix, jx, descx, incx, scale, sumsq)
call pzlasq(n, x, ix, jx, descx, incx, scale, sumsq)
```

### 説明

このルーチンは、次のように値 *scl* と *sumsq* を返す。

$$scl^2 * sumsq = x(1)^2 + \dots + x(n)^2 + scale^2 * sumsq$$

ここで、 $x(i) = \text{sub}(x) = x(ix + (jx-1)*descx(m_) + (i-1)*incx)$  (*pslasq*/*pdlasq* の場合)、 $x(i) = \text{sub}(x) = \text{abs}(x(ix + (jx-1)*descx(m_) + (i-1)*incx))$  (*pclasq*/*pzlasq* の場合)。

実数ルーチン `pslassq`/`pdlassq` では、`sumsq` の値は非負であるとみなされ、`scl` は次の値を返す。

$$scl = \max( scale, \text{abs}(x(i)) )$$

複素数ルーチン `pclassq`/`pzlassq` では、`sumsq` の値は少なくともユニティであるとみなされ、`ssq` の値は、次のように満たされる。

$$1.0 \leq ssq \leq \text{sumsq} + 2n$$

`scale` の値は非負であるとみなされ、`scl` は次の値を返す。

$$scl = \max_i( scale, \text{abs}(\text{real}(x(i))), \text{abs}(\text{aimag}(x(i))) )$$

`p?lassq` の全ルーチンで、`scale` と `sumsq` は、それぞれ `scale` と `sumsq` によって与えられなければならない。また `scale` と `sumsq` は `scl` と `ssq` にそれぞれ上書きされる。

`p?lassq` の全ルーチンは唯一、ベクトル `sub(x)` を出力に渡す。

## 入力パラメータ

<code>n</code>	(グローバル) INTEGER。乱数ベクトル <code>sub(x)</code> の長さ。
<code>x</code>	REAL ( <code>pslassq</code> の場合) DOUBLE PRECISION ( <code>pdlassq</code> の場合) COMPLEX ( <code>pclassq</code> の場合) COMPLEX*16 ( <code>pzlassq</code> の場合)。 スケーリングされた二乗和を計算するベクトル。 $x(ix + (jx-1)*m\_x + (i-1)*incx), 1 \leq i \leq n$
<code>ix</code>	(グローバル) INTEGER。 <code>sub(X)</code> の最初の行を示す、グローバル配列 <code>X</code> の行インデックス。
<code>jx</code>	(グローバル) INTEGER。 <code>sub(X)</code> の最初の列を示す、グローバル配列 <code>X</code> の列インデックス。
<code>descx</code>	(グローバルおよびローカル) INTEGER。配列、次元は ( <code>dlen_</code> )。 分散行列 <code>X</code> の配列ディスクリプタ。
<code>incx</code>	(グローバル) INTEGER。 <code>X</code> の成分のグローバルな増分。このバージョンでは、1 と <code>m_x</code> の2つの <code>incx</code> の値のみサポートされる。 <code>incx</code> の値は、ゼロであってはならない。
<code>scale</code>	(ローカル) REAL ( <code>pslassq</code> / <code>pclassq</code> の場合) DOUBLE PRECISION ( <code>pdlassq</code> / <code>pzlassq</code> の場合)。 上の式で示した <code>scale</code> の値を格納する。

*sumsq* (ローカル) REAL (pslassq/pclassq の場合)  
DOUBLE PRECISION (pdlassq/pzlassq の場合)。  
上の式で示した *sumsq* の値を格納する。

### 出力パラメータ

*scale* (ローカル) 終了時に、*scale* は、二乗和に対するスケール係数 *scl* で上書きされる。

*sumsq* (ローカル) 終了時に、*sumsq* は、*scl* が因子分解された二乗和 *smsq* の値で上書きされる。

---

## p?laswp

一般矩形行列に対して一連の行交換を実行する。

---

### 構文

```
call pslaswp(direc, rowcol, n, a, ia, ja, desca, k1, k2, ipiv)
call pdlaswp(direc, rowcol, n, a, ia, ja, desca, k1, k2, ipiv)
call pclaswp(direc, rowcol, n, a, ia, ja, desca, k1, k2, ipiv)
call pzlaswp(direc, rowcol, n, a, ia, ja, desca, k1, k2, ipiv)
```

### 説明

このルーチンは、分散行列  $\text{sub}(A)=A(ia:ia+n-1, ja:ja+n-1)$  に一連の行 / 列交換を実行する。交換は、 $\text{sub}(A)$  の行 / 列  $k1$  から  $k2$  のそれぞれに対して開始される。ピボット演算情報はすでにプロセス行 / 列に送信済みであると仮定する。また、このルーチンは、同じ  $mb$  (または  $nb$ ) ブロック内の  $k1-k2$  に対してのみ動作する点に注意する。フル行列のピボット演算を行う場合は、[p?lapiv](#) を使用する。

### 入力パラメータ

*direc* (グローバル) CHARACTER。  
置換順序を指定する。  
'F' (順方向)  
'B' (逆方向)



<i>rowcol</i>	(グローバル) CHARACTER。 行、列どちらを置換するのかを指定する。 'R' (行) 'C' (列)
<i>n</i>	(グローバル) INTEGER。  <i>rowcol</i> = 'R' の場合、置換する分散行列 $A(*, ja:ja+n-1)$ の行数。 <i>rowcol</i> = 'C' の場合、置換する分散行列 $A(ia:ia+n-1, *)$ の列数。
<i>a</i>	(ローカル) REAL (pslaswp の場合) DOUBLE PRECISION (pdlaswp の場合) COMPLEX (pclaswp の場合) COMPLEX*16 (pzlaswp の場合)。 ローカルメモリにある、次元 ( <i>lld_a</i> , *) の配列へのポインタ。 行 / 列交換を適用する分散行列のローカル部分を格納する。
<i>ix</i>	(グローバル) INTEGER。 sub( <i>A</i> ) の最初の行を示す、グローバル配列 <i>A</i> の行インデックス。
<i>jx</i>	(グローバル) INTEGER。 sub( <i>A</i> ) の最初の列を示す、グローバル配列 <i>A</i> の列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。 分散行列 <i>A</i> の配列ディスクリプタ。
<i>k1</i>	(グローバル) INTEGER。行 / 列交換を適用する <i>ipiv</i> の最初の成分。
<i>k2</i>	(グローバル) INTEGER。行 / 列交換を適用する <i>ipiv</i> の最後の成分。
<i>ipiv</i>	(ローカル) INTEGER。 配列、次元は $LOCr(m\_a)+mb\_a$ (行ピボット演算の場合) または $LOCr(n\_a)+nb\_a$ (列ピボット演算の場合)。行列 <i>A</i> に関連付けられて おり、 $ipiv(k)=1$ は行 (または列) <i>k</i> と 1 の交換を意味する。

## 出力パラメータ

<i>a</i>	(ローカル) REAL (pslaswp の場合) DOUBLE PRECISION (pdlaswp の場合) COMPLEX (pclaswp の場合) COMPLEX*16 (pzlaswp の場合)。 終了時に、置換された分散行列で上書きされる。
----------	--

## p?latra

一般正方分散行列の対角和を計算する。

### 構文

```
val = pslatra(n, a, ia, ja, desca)
val = pdlatra(n, a, ia, ja, desca)
val = pclatra(n, a, ia, ja, desca)
val = pzlatra(n, a, ia, ja, desca)
```

### 説明

この関数は、 $n \times n$  の分散行列  $\text{sub}(A)$  の対角和を計算する。ここで、 $\text{sub}(A)$  は  $A(\text{ia}:\text{ia}+n-1, \text{ja}:\text{ja}+n-1)$  である。結果はグリッドのすべてのプロセスで残される。

### 入力パラメータ

<i>n</i>	(グローバル) INTEGER。 演算を行う行数と列数、すなわち、分散部分行列 $\text{sub}(A)$ の次数。 $n \geq 0$ 。
<i>a</i>	(ローカル) REAL (pslatra の場合) DOUBLE PRECISION (pdlatra の場合) COMPLEX (pclatra の場合) COMPLEX*16 (pzlatra の場合)。 ローカルメモリにある、対角和を計算する分散行列のローカル部分格納する、次元 ( $11d\_a, LOCc(\text{ja}+n-1)$ ) の配列へのポインタ。
<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 $\text{sub}(A)$ の最初の行と最初の列を示す、グローバル配列 $A$ の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( $dlen\_$ )。分散行列 $A$ の配列ディスクリプタ。

### 出力パラメータ

<i>val</i>	関数の戻り値。
------------	---------

## p?latrd

直交/ユニタリ相似変換を用いて、対称/エルミート行列  $A$  の最初の  $nb$  行/列を実数三重対角形式に縮退する。

### 構文

```
call pslatrd(uplo, n, nb, a, ia, ja, desca, d, e, tau, w, iw, jw, descw,
             work)
call pdlatrd(uplo, n, nb, a, ia, ja, desca, d, e, tau, w, iw, jw, descw,
             work)
call pclatrd(uplo, n, nb, a, ia, ja, desca, d, e, tau, w, iw, jw, descw,
             work)
call pzlatrd(uplo, n, nb, a, ia, ja, desca, d, e, tau, w, iw, jw, descw,
             work)
```

### 説明

このルーチンは、直交/ユニタリ相似変換  $Q' * \text{sub}(A) * Q$  を用いて、実対称/複素エルミート行列  $\text{sub}(A) = A(ia:ia+n-1, ja:ja+n-1)$  の  $nb$  行を対称/複素三重対角形式に縮退し、 $\text{sub}(A)$  の縮退されていない部分に変換を適用するために必要となる  $V$  と  $W$  を返す。

$uplo = 'U'$  の場合、p?latrd は、上三角が与えられる行列に対して最後の  $nb$  行/列の縮退を実行する。

$uplo = 'L'$  の場合、p?latrd は、下三角が与えられる行列に対して最初の  $nb$  行/列の縮退を実行する。

このルーチンは、[p?sytrd](#)/[p?hetrd](#) から呼び出される補助ルーチンである。

### 入力パラメータ

$uplo$	(グローバル) CHARACTER。 対称/エルミート行列 $\text{sub}(A)$ の上三角または下三角部分のどちらを格納するか指定する。 'U' の場合、上三角。 'L' の場合、下三角。
$n$	(グローバル) INTEGER。 演算を行う行数と列数、すなわち、分散部分行列 $\text{sub}(A)$ の次数。 $n \geq 0$ 。

<i>nb</i>	(グローバル) INTEGER。 縮退する行数と列数。
<i>a</i>	REAL (pslatrd の場合 ) DOUBLE PRECISION (pdlatrd の場合 ) COMPLEX (pclatrd の場合 ) COMPLEX*16 (pzlatrd の場合 )。 ローカルメモリにある、次元 ( <i>lld_a</i> , <i>LOCc(ja+n-1)</i> ) の配列へのポインタ。 対称 / エルミート行列 <i>sub(A)</i> のローカル部分を格納する。 <i>uplo</i> = 'U' の場合、 <i>sub(A)</i> の先頭の $n \times n$ の上三角部分に行列の上三角部分を格納する。厳密な下三角部分は参照されない。 <i>uplo</i> = 'L' の場合、 <i>sub(A)</i> の先頭の $n \times n$ の下三角部分に行列の下三角部分を格納する。厳密な上三角部分は参照されない。
<i>ia</i>	(グローバル) INTEGER。 <i>sub(A)</i> の最初の行を示す、グローバル配列 <i>A</i> の行インデックス。
<i>ja</i>	(グローバル) INTEGER。 <i>sub(A)</i> の最初の列を示す、グローバル配列 <i>A</i> の列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。分散行列 <i>A</i> の配列ディスクリプタ。
<i>iw</i>	(グローバル) INTEGER。 <i>sub(W)</i> の最初の行を示す、グローバル配列 <i>W</i> の行インデックス。
<i>jw</i>	(グローバル) INTEGER。 <i>sub(W)</i> の最初の列を示す、グローバル配列 <i>W</i> の列インデックス。
<i>descw</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。分散行列 <i>W</i> の配列ディスクリプタ。
<i>work</i>	(ローカル) REAL (pslatrd の場合 ) DOUBLE PRECISION (pdlatrd の場合 ) COMPLEX (pclatrd の場合 ) COMPLEX*16 (pzlatrd の場合 )。 次元 ( <i>nb_a</i> ) のワークスペース配列。

## 出力パラメータ

<i>a</i>	(ローカル) 終了時に、 <i>uplo</i> = 'U' の場合、最後の <i>nb</i> 列が三重対角形式に縮退され、 <i>sub(A)</i> の対角成分は、三重対角形式の対角成分で上書きされる。対角よりも上の成分は、配列 <i>tau</i> とともに、基本リフレクタ
----------	---

の積として直交/ユニタリ行列  $Q$  を表現する。

$uplo = 'L'$  の場合、最初の  $nb$  列が三重対角形式に縮退され、 $sub(A)$  の対角成分は、三重対角形式の対角成分で上書きされる。対角よりも下の成分は、配列  $tau$  とともに、基本リフレクタの積として直交/ユニタリ行列  $Q$  を表現する。

$d$	<p>(ローカル)</p> <p>REAL (pslatrd/pclatrd の場合)</p> <p>DOUBLE PRECISION (pdlatrd/pzlatrd の場合)。</p> <p>配列、次元は <math>LOCc(ja+n-1)</math>。</p> <p>三重対角行列 <math>T</math> の対角成分。 <math>d(i) = a(i,i)</math> <math>d</math> は、分散行列 <math>A</math> に関連付けられる。</p>
$e$	<p>(ローカル)</p> <p>REAL (pslatrd/pclatrd の場合)</p> <p>DOUBLE PRECISION (pdlatrd/pzlatrd の場合)。</p> <p>配列、次元は <math>LOCc(ja+n-1)</math> (<math>uplo = 'U'</math> の場合) または <math>LOCc(ja+n-2)</math> (それ以外の場合)。</p> <p>三重対角行列 <math>T</math> の非対角成分。</p> <p><math>uplo = 'U'</math> の場合、 <math>e(i) = a(i, i+1)</math>。</p> <p><math>uplo = 'L'</math> の場合、 <math>e(i) = a(i+1, i)</math>。</p> <p><math>e</math> は分散行列 <math>A</math> に関連付けられる。</p>
$tau$	<p>(ローカル)</p> <p>REAL (pslatrd の場合)</p> <p>DOUBLE PRECISION (pdlatrd の場合)</p> <p>COMPLEX (pclatrd の場合)</p> <p>COMPLEX*16 (pzlatrd の場合)。</p> <p>配列、次元は <math>LOCc(ja+n-1)</math>。</p> <p>基本リフレクタのスカラー係数 <math>tau</math> が格納される。<math>tau</math> は、分散行列 <math>A</math> に関連付けられる。</p>
$w$	<p>(ローカル)</p> <p>REAL (pslatrd の場合)</p> <p>DOUBLE PRECISION (pdlatrd の場合)</p> <p>COMPLEX (pclatrd の場合)</p> <p>COMPLEX*16 (pzlatrd の場合)。</p> <p>ローカルメモリにある、次元 <math>(lld\_w, nb\_w)</math> の配列へのポインタ。</p> <p><math>sub(A)</math> の縮退されていない部分の更新に必要な <math>n \times nb\_w</math> の行列 <math>W</math> のローカル部分が格納される。</p>

## アプリケーション・ノート

`uplo = 'U'` の場合、行列  $Q$  は基本リフレクタの積として表現される。

$$Q = H(n) H(n-1) \dots H(n-nb+1)$$

それぞれの  $H(i)$  は次のような形式を持つ。

$$H(i) = I - \tau v v'$$

ここで、 $\tau$  は実 / 複素スカラー、 $v$  は実 / 複素ベクトルで  $v(i:n) = 0$  および  $v(i-1) = 1$ 。終了時に、 $v(1:i-1)$  は  $A(ia:ia+i-1, ja+i)$  に、 $\tau$  は  $\tau(ja+i-1)$  に格納される。

`uplo = 'L'` の場合、行列  $Q$  は基本リフレクタの積として表現される。

$$Q = H(1) H(2) \dots H(nb)$$

それぞれの  $H(i)$  は次のような形式を持つ。

$$H(i) = I - \tau v v'$$

$\tau$  は実 / 複素スカラー、 $v$  は実 / 複素ベクトルで  $v(1:i) = 0$  および  $v(i+1) = 1$ 。終了時に、 $v(i+2:n)$  は  $A(ia+i+1:ia+n-1, ja+i-1)$  に、 $\tau$  は  $\tau(ja+i-1)$  に格納される。

ベクトル  $v$  の成分は、行列の縮退されていない部分に変換を適用するために、行列  $W$  とともに必要となる  $n \times nb$  の行列  $V$  を、形式の対称 / エルミート階数  $-2k$  更新を使用し形成する。

$$\text{sub}(A) := \text{sub}(A) - v w' - w v'$$

終了時の  $a$  の内容を次に示す ( $n = 5$  および  $nb = 2$  の場合)。

`uplo = 'U'` の場合

$$\begin{bmatrix} a & a & a & v_4 & v_5 \\ & a & a & v_4 & v_5 \\ & & a & 1 & v_5 \\ & & & d & 1 \\ & & & & d \end{bmatrix}$$

`uplo = 'L'` の場合

$$\begin{bmatrix} d \\ 1 & d \\ v_1 & 1 & a \\ v_1 & v_2 & a & a \\ v_1 & v_2 & a & a & a \end{bmatrix}$$

$d$  は縮退された行列の対角成分、 $a$  は変更されていない元の行列の成分、 $v_i$  は  $H(i)$  を定義するベクトルの成分を表わす。

## p?latrs

オーバーフローを防ぐために設定されたスケール係数を持つ三角連立方程式を解く。

### 構文

```
call pslatrs(uplo, trans, diag, normin, n, a, ia, ja, desca, x, ix, jx, descx,
             scale, cnorm, work)
call pdlatrs(uplo, trans, diag, normin, n, a, ia, ja, desca, x, ix, jx, descx,
             scale, cnorm, work)
call pclatrs(uplo, trans, diag, normin, n, a, ia, ja, desca, x, ix, jx, descx,
             scale, cnorm, work)
call pzlatrs(uplo, trans, diag, normin, n, a, ia, ja, desca, x, ix, jx, descx,
             scale, cnorm, work)
```

### 説明

このルーチンは、三角連立方程式  $Ax = \sigma b$ 、 $A^T x = \sigma b$ 、または  $A^H x = \sigma b$  を解く。ここで、 $\sigma$  はオーバーフローを防ぐために設定されたスケール係数である。このルーチンの説明は、将来のリリースで拡張される。

### 入力パラメータ

<i>uplo</i>	CHARACTER*1。 行列 $A$ が上三角か下三角かを指定する。 'U' の場合、上三角である。 'L' の場合、下三角である。
<i>trans</i>	CHARACTER*1。 $A$ に適用する演算を指定する。 'N' の場合、 $Ax = \sigma b$ を解く (転置なし) 'T' の場合、 $A^T x = \sigma b$ を解く (転置) 'C' の場合、 $A^H x = \sigma b$ を解く (共役転置)。
<i>diag</i>	CHARACTER*1。 行列 $A$ が単位三角かどうかを指定する。 'N' の場合、単位三角ではない。 'U' の場合、単位三角である。

<i>normin</i>	<p>CHARACTER*1。  <i>cnorm</i> を設定したかどうかを指定する。          'Y' の場合、<i>cnorm</i> には列ノルムを格納した。          'N' の場合、<i>cnorm</i> は設定していない。終了時に、ノルムが計算され、<i>cnorm</i> に格納される。</p>
<i>n</i>	<p>INTEGER。          行列 <i>A</i> の次数。 <math>n \geq 0</math></p>
<i>a</i>	<p>REAL (pslatrs/pclatrs の場合 )          DOUBLE PRECISION (pdlatrs/pzlatrs の場合 )。          配列、次元は (<i>lda</i>, <i>n</i>)。三角行列 <i>A</i> を格納する。 <i>uplo</i> = 'U' の場合、配列 <i>a</i> の先頭の <math>n \times n</math> の上三角部分に上三角行列を格納する。<i>a</i> の厳密な下三角部分は参照されない。 <i>uplo</i> = 'L' の場合、配列 <i>a</i> の先頭の <math>n \times n</math> の下三角部分に下三角行列を格納する。<i>a</i> の厳密な上三角部分は参照されない。 <i>diag</i> = 'U' の場合も <i>a</i> の対角成分は参照されず 1 とみなされる。</p>
<i>ia, ja</i>	<p>(グローバル) INTEGER。それぞれ、部分行列 <i>A</i> の最初の行と最初の列を示す、グローバル配列 <i>a</i> の行インデックスと列インデックス。</p>
<i>desca</i>	<p>(グローバルおよびローカル) INTEGER。配列、次元は (<i>dlen_</i>)。分散行列 <i>A</i> の配列ディスクリプタ。</p>
<i>x</i>	<p>REAL (pslatrs/pclatrs の場合 )          DOUBLE PRECISION (pdlatrs/pzlatrs の場合 )。          配列、次元は (<i>n</i>)。三角法の右辺 <i>b</i> を格納する。</p>
<i>ix</i>	<p>(グローバル) INTEGER。 <i>sub(x)</i> の最初の行を示す、グローバル配列 <i>x</i> の行インデックス。</p>
<i>jx</i>	<p>(グローバル) INTEGER。 <i>sub(x)</i> の最初の列を示す、グローバル配列 <i>x</i> の列インデックス。</p>
<i>descx</i>	<p>(グローバルおよびローカル) INTEGER。          配列、次元は (<i>dlen_</i>)。分散行列 <i>X</i> の配列ディスクリプタ。</p>
<i>cnorm</i>	<p>REAL (pslatrs/pclatrs の場合 )          DOUBLE PRECISION (pdlatrs/pzlatrs の場合 )。          配列、次元は (<i>n</i>)。 <i>normin</i> = 'Y' の場合、<i>cnorm</i> は入力引数となり <i>cnorm(j)</i> には <i>A</i> の <i>j</i> 番目列の非対角部分を格納する。 <i>trans</i> = 'N' の場合、<i>cnorm(j)</i> は無限ノルムに等しいか大きくなければならない。  <i>trans</i> = 'T' または 'C' の場合、<i>cnorm(j)</i> は 1- ノルムに等しいか大きくなければならない。</p>
<i>work</i>	<p>(ローカル)</p>



REAL (pslatrs の場合 )  
 DOUBLE PRECISION (pdlatrs の場合 )  
 COMPLEX (pclatrs の場合 )  
 COMPLEX\*16 (pzlatrs の場合 )。  
 テンポラリ・ワークスペース。

### 出力パラメータ

*x*                    終了時に、*x* は解のベクトル *x* によって上書きされる。

*scale*                REAL (pslatrs/pclatrs の場合 )  
                       DOUBLE PRECISION (pdlatrs/pzlatrs の場合 )。  
                       配列、次元は (lda, n)。上述のとおり三角法に対するスケール係数 *s*。  
                       *scale* = 0 は行列 *A* が特異であるか不適切にスケールされたことを示し、ベクトル *x* は  $Ax = 0$  に対する完全または近似解となる。

*cnorm*                *normin* = 'N' の場合、*cnorm* は出力引数となり *cnorm*(*j*) は *A* の *j* 番目  
                       列の非対角部分の 1- ノルムを返す。

## p?latrz

直交 / ユニタリ相似変換を用いて、実 / 複素上台形行列を上三角形式に縮退させる。

### 構文

```
call pslatz(m, n, l, a, ia, ja, desca, tau, work)
call pdlatrz(m, n, l, a, ia, ja, desca, tau, work)
call pclatz(m, n, l, a, ia, ja, desca, tau, work)
call pzlatrz(m, n, l, a, ia, ja, desca, tau, work)
```

### 説明

このルーチンは、直交 / ユニタリ変換を用いて、 $m \times n$  ( $m \leq n$ ) の実 / 複素上台形行列  $\text{sub}(A) = [A(ia:ia+m-1, ja:ja+m-1) \ A(ia:ia+m-1, ja+n-l:ja+n-1)]$  を、上三角形式に縮退させる。

上台形行列  $\text{sub}(A)$  は、次のように因子分解される。

$$\text{sub}(A) = \begin{pmatrix} R & 0 \end{pmatrix} * Z$$

$Z$  は  $n \times n$  の直交 / ユニタリ行列、 $R$  は  $m \times m$  の上三角行列である。

## 入力パラメータ

$m$	(グローバル) INTEGER。 演算を行う行数、すなわち、分散部分行列 $\text{sub}(A)$ の行数。 $m \geq 0$ 。
$n$	(グローバル) INTEGER。 演算を行う列数、すなわち、分散部分行列 $\text{sub}(A)$ の列数。 $n \geq 0$ 。
$l$	(グローバル) INTEGER。 Householder リフレクタとして意味を持った部分が格納されている分散部分行列 $\text{sub}(A)$ の列数。 $l > 0$ 。
$a$	(ローカル)  REAL (pslatrz の場合) DOUBLE PRECISION (pdlatrz の場合) COMPLEX (pclatz の場合) COMPLEX*16 (pzlatrz の場合)。 ローカルメモリにある、次元 ( $lld\_a$ , $LOCc(ja+n-1)$ ) の配列へのポインタ。 因子分解を実行する、 $m \times n$ の分散行列 $\text{sub}(A)$ のローカル部分を格納する。
$ia$	(グローバル) INTEGER。 $\text{sub}(A)$ の最初の行を示す、グローバル配列 $A$ の行インデックス。
$ja$	(グローバル) INTEGER。 $\text{sub}(A)$ の最初の列を示す、グローバル配列 $A$ の列インデックス。
$desca$	(グローバルおよびローカル) INTEGER。配列、次元は ( $dlen\_$ )。 分散行列 $A$ の配列ディスクリプタ。
$work$	(ローカル)  REAL (pslatrz の場合) DOUBLE PRECISION (pdlatrz の場合) COMPLEX (pclatz の場合) COMPLEX*16 (pzlatrz の場合)。 ワークスペース配列、次元は ( $lwork$ )。 $lwork \geq nq0 + \max(1, mp0)$ ここで、

```

irow = mod(ia-1, mb_a), icoff = mod(ja-1, nb_a),
iarow = indxg2p(ia, mb_a, myrow, rsrc_a, nprow),
iacol = indxg2p(ja, nb_a, mycol, csrc_a, npcol),
mp0 = numroc(m+irow, mb_a, myrow, iarow, nprow),
nq0 = numroc(n+icoff, nb_a, mycol, iacol, npcol).

```

numroc、indxg2p および numroc は、ScaLAPACK ツール関数である。  
myrow、mycol、nprow、および npcol は、サブルーチン  
blacs\_gridinfo を呼び出して求められる。

## 出力パラメータ

- a** 終了時に、sub(A) の先頭の  $m \times m$  の上三角部分には上三角行列  $R$  が格納される。sub(A) の最初の  $m$  行の  $n-L+1$  から  $N$  までの成分は、配列  $\tau$  とともに、基本リフレクタ  $m$  の積として直交 / ユニタリ行列  $Z$  を表現する。
- tau** (ローカル) REAL (pslatrz の場合)  
DOUBLE PRECISION (pdlatrz の場合)  
COMPLEX (pclatz の場合)  
COMPLEX\*16 (pzlatrz の場合)  
配列、次元は ( $LOCr(ja+m-1)$ )。基本リフレクタのスカラー係数が格納される。 $\tau$  は、分散行列  $A$  に関連付けられる。

## アプリケーション・ノート

因子分解は Householder 法を用いて得る。sub(A) の  $(m-k+1)$  番目の行にゼロを導入するため (複素数ルーチンの場合は、共役置換) に使用される  $k$  番目の変換行列  $Z(k)$  は次の形式で与えられる。

$$Z(k) = \begin{bmatrix} I & 0 \\ 0 & T(k) \end{bmatrix}$$

$$T(k) = I - \tau u(k)^* u(k)' \quad u(k) = \begin{bmatrix} 1 \\ 0 \\ z(k) \end{bmatrix}$$

$\tau$  はスカラー、 $z(k)$  は  $(n-m)$  成分のベクトルである。 $\tau$  と  $z(k)$  は、 $\text{sub}(A)$  の  $k$  番目の行の成分をゼロにするために選択される。

スカラー  $\tau$  は  $\tau$  の  $k$  番目の成分で返され、ベクトル  $u(k)$  は  $z(k)$  の成分が  $a(k, m+1), \dots, a(k, n)$  に格納されるような  $\text{sub}(A)$  の  $k$  番目の行で返される。 $R$  の成分は  $\text{sub}(A)$  の上三角部分で返される。

$Z$  は次の式で与えられる。

$$Z = Z(1) Z(2) \dots Z(m)$$

---

### p?lauu2

積  $UU^H$  または  $L^H L$  を計算する。ここで、 $U$  と  $L$  は上三角または下三角行列(ローカル非ブロック化アルゴリズム)。

---

#### 構文

```
call pslauu2(uplo, n, a, ia, ja, desca)
call pdlauu2(uplo, n, a, ia, ja, desca)
call pclauu2(uplo, n, a, ia, ja, desca)
call pzlauu2(uplo, n, a, ia, ja, desca)
```

#### 説明

このルーチンは、積  $UU^H$  または  $L^H L$  を計算する。三角係数  $U$  または  $L$  は、分散行列  $\text{sub}(A) = A(ia:ia+n-1, ja:ja+n-1)$  の上三角または下三角部分に格納される。

$\text{uplo} = 'U'$  または  $'u'$  の場合、 $\text{sub}(A)$  の係数  $U$  は結果の上三角で上書きされる。

$\text{uplo} = 'L'$  または  $'l'$  の場合、 $\text{sub}(A)$  の係数  $L$  は結果の下三角で上書きされる。

このルーチンは、アルゴリズムの非ブロック化形式で、[BLAS レベル 2 のルーチン](#) を呼び出す。このルーチンはコミュニケーションを実行しないため、演算を行う行列は 1 つのプロセスに対してローカルでなければならない。

## 入力パラメータ

<code>uplo</code>	(グローバル) CHARACTER*1。 行列 <code>sub(A)</code> に格納されている三角係数が上三角か下三角かを指定する。 'U' の場合、上三角。 'L' の場合、下三角。
<code>n</code>	(グローバル) INTEGER。 演算を行う行数と列数、すなわち、三角係数 $U$ または $L$ の次数。 $n \geq 0$ 。
<code>a</code>	(ローカル) REAL (pslauu2 の場合) DOUBLE PRECISION (pdlauu2 の場合) COMPLEX (pclauu2 の場合) COMPLEX*16 (pzlauu2 の場合)。 ローカルメモリにある、次元 ( <code>lld_a</code> , <code>LOCc(ja+n-1)</code> ) の配列へのポインタ。三角係数 $U$ または $L$ のローカル部分を格納する。
<code>ia</code>	(グローバル) INTEGER。 <code>sub(A)</code> の最初の行を示す、グローバル配列 $A$ の行インデックス。
<code>ja</code>	(グローバル) INTEGER。 <code>sub(A)</code> の最初の列を示す、グローバル配列 $A$ の列インデックス。
<code>desca</code>	(グローバルおよびローカル) INTEGER。配列、次元は ( <code>dlen_</code> )。 分散行列 $A$ の配列ディスクリプタ。

## 出力パラメータ

<code>a</code>	(ローカル) 終了時に、 <code>uplo</code> = 'U' の場合、分散行列 <code>sub(A)</code> の上三角は積 $UU'$ の上三角で上書きされる。 <code>uplo</code> = 'L' の場合、 <code>sub(A)</code> の下三角は積 $LL'$ の下三角で上書きされる。
----------------	---

## p?lauum

積  $UU^H$  または  $L^H L$  を計算する。ここで、 $U$  と  $L$  は上三角または下三角行列。

### 構文

```
call pslauum(uplo, n, a, ia, ja, desca)
call pdlauum(uplo, n, a, ia, ja, desca)
call pclauum(uplo, n, a, ia, ja, desca)
call pzlauum(uplo, n, a, ia, ja, desca)
```

### 説明

このルーチンは、積  $UU^H$  または  $L^H L$  を計算する。三角係数  $U$  または  $L$  は、行列  $\text{sub}(A) = A(ia:ia+n-1, ja:ja+n-1)$  の上三角または下三角部分に格納される。

$\text{uplo} = 'U'$  または  $'u'$  の場合、 $\text{sub}(A)$  の係数  $U$  は結果の上三角で上書きされる。

$\text{uplo} = 'L'$  または  $'l'$  の場合、 $\text{sub}(A)$  の係数  $L$  は結果の下三角で上書きされる。

このルーチンは、アルゴリズムのブロック化形式で、レベル 3 PBLAS を呼び出す。

### 入力パラメータ

$\text{uplo}$	(グローバル) CHARACTER*1。 行列 $\text{sub}(A)$ に格納されている三角係数が上三角か下三角かを指定する。 'U' の場合、上三角。 'L' の場合、下三角。
$n$	(グローバル) INTEGER。 演算を行う行数と列数、すなわち、三角係数 $U$ または $L$ の次数。 $n \geq 0$ 。
$a$	(ローカル) REAL (pslauum の場合) DOUBLE PRECISION (pdlauum の場合) COMPLEX (pclauum の場合) COMPLEX*16 (pzlauum の場合)。ローカルメモリにある、次元 ( $\text{lld\_a}$ , $\text{LOCc}(ja+n-1)$ ) の配列へのポインタ。三角係数 $U$ または $L$ のローカル部分を格納する。

<i>ia</i>	(グローバル) INTEGER。 sub(A) の最初の行を示す、グローバル配列 A の行インデックス。
<i>ja</i>	(グローバル) INTEGER。 sub(A) の最初の列を示す、グローバル配列 A の列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。 分散行列 A の配列ディスクリプタ。

### 出力パラメータ

<i>a</i>	(ローカル) 終了時に、 <i>uplo</i> = 'U' の場合、分散行列 sub(A) の上三角は積 $UU'$ の上三角で上書きされる。 <i>uplo</i> = 'L' の場合、sub(A) の下三角は積 $LL$ の下三角で上書きされる。
----------	---

## p?lawil

Wilkinson 変換を実行する。

### 構文

```
call pslawil(ii, jj, m, a, desca, h44, h33, h43h34, v)
call pdlawil(ii, jj, m, a, desca, h44, h33, h43h34, v)
```

### 説明

このルーチンは、*h44*、*h33*、および *h43h34* で与えられる変換を、行 *m* から始まるように *v* に配置する。

### 入力パラメータ

<i>ii</i>	(グローバル) INTEGER。 $h(m+2, m+2)$ の行所有成分。
<i>jj</i>	(グローバル) INTEGER。 $h(m+2, m+2)$ の列所有成分。
<i>m</i>	(グローバル) INTEGER。 変換を開始する位置 (行 <i>m</i> )。終了時に変更されない。

`a` (グローバル)  
REAL(`pslawil` の場合)  
DOUBLE PRECISION(`pdlawil` の場合)。  
配列、次元は(`desca(11d_)`,\*)。Hessenberg 行列。終了時に変更されない。

`desca` (グローバルおよびローカル) INTEGER。配列、次元は(`dlen_`)。分散行列  $A$  の配列ディスクリプタ。終了時に変更されない。

`h44,`  
`h33,`  
`h43h34` (グローバル)  
REAL(`pslawil` の場合)  
DOUBLE PRECISION(`pdlawil` の場合)。  
これらの 3 種類の値は、倍精度シフト  $QR$  の反復法で使用される。終了時に変更されない。

### 出力パラメータ

`v` (グローバル)  
REAL(`pslawil` の場合)  
DOUBLE PRECISION(`pdlawil` の場合)  
変換を格納する、サイズ 3 の配列。

---

## p?org2l/p?ung2l

p?geqlf で求めた  $QL$  因  $q$  分解から、全部または一部の直交/ユニタリ行列  $Q$  を生成する(非ブロッキング化アルゴリズム)。

---

### 構文

```
call psorg2l(m, n, k, a, ia, ja, desca, tau, work, lwork, info)
call pdorg2l(m, n, k, a, ia, ja, desca, tau, work, lwork, info)
call pcung2l(m, n, k, a, ia, ja, desca, tau, work, lwork, info)
call pzung2l(m, n, k, a, ia, ja, desca, tau, work, lwork, info)
```



## 説明

ルーチン `p?org21/p?ung21` は、直交列を持つ  $m \times n$  の実 / 複素行列  $Q$  を生成する。これは、次数  $m$  の  $k$  個の基本リフレクタの積の最後の  $n$  列として次のように定義される。ここで、 $Q$  は  $A(ia:ia+m-1, ja:ja+n-1)$  である。

$Q = H(k) \dots H(2) H(1)$  (`p?geqlf` によって返される)。

## 入力パラメータ

$m$	(グローバル) INTEGER。 演算を行う行数、すなわち、分散部分行列 $Q$ の行数。 $m \geq 0$ 。
$n$	(グローバル) INTEGER。 演算を行う列数、すなわち、分散部分行列 $Q$ の列数。 $m \geq n \geq 0$ 。
$k$	(グローバル) INTEGER。 その積が行列 $Q$ を定義する基本リフレクタの個数。 $n \geq k \geq 0$ 。
$a$	REAL ( <code>psorg21</code> の場合) DOUBLE PRECISION ( <code>pdorg21</code> の場合) COMPLEX ( <code>pcung21</code> の場合) COMPLEX*16 ( <code>pzung21</code> の場合)。 ローカルメモリにある、次元 ( <code>lld_a</code> , $LOCc(ja+n-1)$ ) の配列へのポインタ。 $j$ 番目の列には、 <code>p?geqlf</code> によって分散行列引数 $A(ia:*, ja+n-k:ja+n-1)$ の $k$ 列に返されたとおりに、基本リフレクタ $H(j)$ を定義するベクトルを格納しておかなければならない。ここで、 $H(j)$ は $ja+n-k \leq j \leq ja+n-1$ である。
$ia$	(グローバル) INTEGER。 $sub(A)$ の最初の行を示す、グローバル配列 $A$ の行インデックス。
$ja$	(グローバル) INTEGER。 $sub(A)$ の最初の列を示す、グローバル配列 $A$ の列インデックス。
$desca$	(グローバルおよびローカル) INTEGER。配列、次元は ( <code>dlen_</code> )。 分散行列 $A$ の配列ディスクリプタ。
$\tau$	(ローカル) REAL ( <code>psorg21</code> の場合) DOUBLE PRECISION ( <code>pdorg21</code> の場合) COMPLEX ( <code>pcung21</code> の場合) COMPLEX*16 ( <code>pzung21</code> の場合)。

配列、次元は  $LOCc(ja+n-1)$ 。

[p?geqlf](#) から返されたとおりに、基本リフレクタ  $H(j)$  のスカラ係数  $\tau(j)$  を格納する。

**work** (ローカル)  
 REAL (psorg21 の場合)  
 DOUBLE PRECISION (pdorg21 の場合)  
 COMPLEX (pcung21 の場合)  
 COMPLEX\*16 (pzung21 の場合)。  
 ワークスペース配列、次元は (*lwork*)。

**lwork** (ローカルまたはグローバル) INTEGER。  
 配列 **work** の次元。  
*lwork* はローカル入力であり、 $lwork \geq mpa0 + \max(1, nqa0)$  でなければならない。  
 ここで、 $iroffa = \text{mod}(ia-1, mb\_a)$ 、 $icoffa = \text{mod}(ja-1, nb\_a)$ 、  
 $iarow = \text{indxg2p}(ia, mb\_a, myrow, rsrc\_a, nprow)$ 、  
 $iacol = \text{indxg2p}(ja, nb\_a, mycol, csrc\_a, npcot)$ 、  
 $mpa0 = \text{numroc}(m+iroffa, mb\_a, myrow, iarow, nprow)$ 、  
 $nqa0 = \text{numroc}(n+icoffa, nb\_a, mycol, iacol, npcot)$ 。  
 indxg2p および numroc は、ScaLAPACK ツール関数である。  
 myrow、mycol、npro、および npcot は、サブルーチン  
 blacs\_gridinfo を呼び出して求められる。  
*lwork* = -1 の場合、*lwork* はグローバル入力であり、ワークスペース  
 のクエリとみなされ、ルーチンはすべての配列の最小かつ最適なサイ  
 ズだけを計算する。各値は該当する **work** 配列の最初のエントリとし  
 て返され、[pxerbla](#) はエラー・メッセージを生成しない。

## 出力パラメータ

**a** 終了時に、 $m \times n$  の分散行列  $Q$  のローカル部分が格納される。

**work** 終了時に、**work**(1) は、最小かつ最適な *lwork* 値を返す。

**info** (ローカル) INTEGER。  
*info* = 0 の場合、正常に終了したことを示す。  
*info* < 0 の場合、*i* 番目の引数が配列で *j* 番目の成分の値が不正なら  
 ば  $info = -(i*100+j)$ 、*i* 番目の引数がスカラで値が不正ならば  
 $info = -i$ 。

## p?org2r/p?ung2r

p?geqrf で求めた  $QR$  因子分解から、全部または一部の直交/ユニタリ行列  $Q$  を生成する (非ブロック化アルゴリズム)。

### 構文

```
call psorg2r(m, n, k, a, ia, ja, desca, tau, work, lwork, info)
call pdorg2r(m, n, k, a, ia, ja, desca, tau, work, lwork, info)
call pcung2r(m, n, k, a, ia, ja, desca, tau, work, lwork, info)
call pzung2r(m, n, k, a, ia, ja, desca, tau, work, lwork, info)
```

### 説明

ルーチン p?org2r/p?ung2r は、直交列を持つ  $m \times n$  の実 / 複素行列  $Q$  を生成する。これは、次数  $m$  の  $k$  個の基本リフレクタの積の最初の  $n$  列として次のように定義される。ここで、 $Q$  は  $A(ia:ia+m-1, ja:ja+n-1)$  である。

$$Q = H(1)H(2) \dots H(k)$$

[p?geqrf](#) によって返される。

### 入力パラメータ

- |     |   |
|-----|---|
| $m$ | (グローバル) INTEGER。<br>演算を行う行数、すなわち、分散部分行列 $Q$ の行数。 $m \geq 0$ 。   |
| $n$ | (グローバル) INTEGER。<br>演算を行う列数、すなわち、分散部分行列 $Q$ の列数。 $m \geq n \geq 0$ 。  |
| $k$ | (グローバル) INTEGER。<br>その積が行列 $Q$ を定義する基本リフレクタの個数。<br>$n \geq k \geq 0$ 。  |
| $a$ | REAL (psorg2r の場合)<br>DOUBLE PRECISION (pdorg2r の場合)<br>COMPLEX (pcung2r の場合)<br>COMPLEX*16 (pzung2r の場合)。<br>ローカルメモリにある、次元 ( $lld\_a$ , $LOCc(ja+n-1)$ ) の配列へのポインタ。<br>$j$ 番目の列には、 <a href="#">p?geqrf</a> によって分散行列引数 $A(ia:*, ja:ja+k-1)$ |

の  $k$  列に返されたとおりに、基本リフレクタ  $H(j)$  を定義するベクトルを格納しておかなければならない。ここで、 $H(j)$  は  $ja \leq j \leq ja+k-1$  である。

<i>ia</i>	(グローバル) INTEGER。 sub(A) の最初の行を示す、グローバル配列 A の行インデックス。
<i>ja</i>	(グローバル) INTEGER。 sub(A) の最初の列を示す、グローバル配列 A の列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は (dlen_)。 分散行列 A の配列ディスクリプタ。
<i>tau</i>	(ローカル) REAL (psorg2r の場合) DOUBLE PRECISION (pdorg2r の場合) COMPLEX (pcung2r の場合) COMPLEX*16 (pzung2r の場合)。 配列、次元は $LOCc(ja+k-1)$ 。 <a href="#">p?gegrf</a> から返されたとおりに、基本リフレクタ $H(j)$ のスカラ係数 $\tau(j)$ を格納する。分散行列 A に関連付けられる。
<i>work</i>	(ローカル) REAL (psorg2r の場合) DOUBLE PRECISION (pdorg2r の場合) COMPLEX (pcung2r の場合) COMPLEX*16 (pzung2r の場合) ワークスペース配列、次元は (lwork)。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 配列 work の次元。lwork はローカル入力であり、 $lwork \geq mpa0 + \max(1, nqa0)$ でなければならない。 ここで、 $iroffa = \text{mod}(ia-1, mb\_a)$ 、 $icoffa = \text{mod}(ja-1, nb\_a)$ 、 $iarow = \text{indxg2p}(ia, mb\_a, myrow, rsrc\_a, nprow)$ 、 $iacol = \text{indxg2p}(ja, nb\_a, mycol, csrc\_a, npcot)$ 、 $mpa0 = \text{numroc}(m+iroffa, mb\_a, myrow, iarow, nprow)$ 、 $nqa0 = \text{numroc}(n+icoffa, nb\_a, mycol, iacol, npcot)$ 。  $\text{indxg2p}$ および $\text{numroc}$ は、ScaLAPACK ツール関数である。 $myrow$ 、 $mycol$ 、 $nprow$ 、および $npcot$ は、サブルーチン $\text{blacs\_gridinfo}$ を呼び出して求められる。

$lwork = -1$  の場合、 $lwork$  はグローバル入力であり、ワークスペースのクエリとみなされ、ルーチンはすべての配列の最小かつ最適なサイズだけを計算する。各値は該当する  $work$  配列の最初のエントリとして返され、[pxerbla](#) はエラー・メッセージを生成しない。

### 出力パラメータ

$a$	終了時に、 $m \times n$ の分散行列 $Q$ のローカル部分が格納される。
$work$	終了時に、 $work(1)$ は、最小かつ最適な $lwork$ 値を返す。
$info$	(ローカル) INTEGER。 $info = 0$ の場合、正常に終了したことを示す。 $info < 0$ の場合、 $i$ 番目の引数が配列で $j$ 番目の成分の値が不正ならば $info = -(i * 100 + j)$ 、 $i$ 番目の引数がスカラで値が不正ならば $info = -i$ 。

## p?orgl2/p?ungl2

$p?gelqf$  で求めた  $LQ$  因子分解から、全部または一部の直交/ユニタリ行列  $Q$  を生成する (非ブロック化アルゴリズム)。

### 構文

```
call psorgl2(m, n, k, a, ia, ja, desca, tau, work, lwork, info)
call pdorgl2(m, n, k, a, ia, ja, desca, tau, work, lwork, info)
call pcungl2(m, n, k, a, ia, ja, desca, tau, work, lwork, info)
call pzungl2(m, n, k, a, ia, ja, desca, tau, work, lwork, info)
```

### 説明

ルーチン  $p?orgl2/p?ungl2$  は、直交列を持つ  $m \times n$  の実/複素行列  $Q$  を生成する。これは、次数  $n$  の  $k$  個の基本リフレクタの積の最初の  $m$  行として次のように定義される。ここで、 $Q$  は  $A(ia:ia+m-1, ja:ja+n-1)$  である。

$$Q = H(k) \dots H(2) H(1) \text{ (実数型の場合)}$$

$$Q = H(k)' \dots H(2)' H(1)' \text{ (複素数型の場合)}$$

[p?gelqf](#) によって返される。

## 入力パラメータ

<i>m</i>	(グローバル) INTEGER。 演算を行う行数、すなわち、分散部分行列 <i>Q</i> の行数。 $m \geq 0$ 。
<i>n</i>	(グローバル) INTEGER。 演算を行う列数、すなわち、分散部分行列 <i>Q</i> の列数。 $n \geq m \geq 0$ 。
<i>k</i>	(グローバル) INTEGER。 その積が行列 <i>Q</i> を定義する基本リフレクタの個数。 $m \geq k \geq 0$ 。
<i>a</i>	REAL (psorgl2 の場合) DOUBLE PRECISION (pdorgl2 の場合) COMPLEX (pcungl2 の場合) COMPLEX*16 (pzungl2 の場合) ローカルメモリにある、次元 ( <i>lld_a</i> , <i>LOCc(ja+n-1)</i> ) の配列へのポインタ。 <i>i</i> 番目の行には、 <a href="#">p?gelqf</a> によって分散行列引数 <i>A</i> ( <i>ia:ia+k-1</i> , <i>ja:*</i> ) の <i>k</i> 行に返されたとおりに、基本リフレクタ <i>H</i> ( <i>i</i> ) を定義するベクトルを格納しておかなければならない。 ここで、 <i>H</i> ( <i>i</i> ) は $ia \leq i \leq ia+k-1$ 。
<i>ia</i>	(グローバル) INTEGER。 sub( <i>A</i> ) の最初の行を示す、グローバル配列 <i>A</i> の行インデックス。
<i>ja</i>	(グローバル) INTEGER。 sub( <i>A</i> ) の最初の列を示す、グローバル配列 <i>A</i> の列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。 分散行列 <i>A</i> の配列ディスクリプタ。
<i>tau</i>	(ローカル) REAL (psorgl2 の場合) DOUBLE PRECISION (pdorgl2 の場合) COMPLEX (pcungl2 の場合) COMPLEX*16 (pzungl2 の場合) 配列、次元は <i>LOCr(ja+k-1)</i> 。 <a href="#">p?gelqf</a> から返されたとおりに、基本リフレクタ <i>H</i> ( <i>i</i> ) のスカラー係数 <i>tau</i> ( <i>i</i> ) を格納する。分散行列 <i>A</i> に関連付けられる。
<i>work</i>	(ローカル) REAL (psorgl2 の場合) DOUBLE PRECISION (pdorgl2 の場合)

COMPLEX (pcungl2 の場合)  
 COMPLEX\*16 (pzungl2 の場合)。  
 ワークスペース配列、次元は (*lwork*)。

*lwork* (ローカルまたはグローバル) INTEGER。  
 配列 *work* の次元。  
*lwork* はローカル入力であり、 $lwork \geq nqa0 + \max(1, mpa0)$  でなければならない。ここで、  
 $iroffa = \text{mod}(ia-1, mb\_a)$ 、 $icoffa = \text{mod}(ja-1, nb\_a)$ 、  
 $iarow = \text{indxg2p}(ia, mb\_a, myrow, rsrc\_a, nprow)$ 、  
 $iacol = \text{indxg2p}(ja, nb\_a, mycol, csrc\_a, npcot)$ 、  
 $mpa0 = \text{numroc}(m+iroffa, mb\_a, myrow, iarow, nprow)$ 、  
 $nqa0 = \text{numroc}(n+icoffa, nb\_a, mycol, iacol, npcot)$ 。

*indxg2p* および *numroc* は、ScaLAPACK ツール関数である。  
*myrow*、*mycol*、*nprow*、および *npcot* は、サブルーチン *blacs\_gridinfo* を呼び出して求められる。

*lwork* = -1 の場合、*lwork* はグローバル入力であり、ワークスペースのクエリとみなされ、ルーチンはすべての配列の最小かつ最適なサイズだけを計算する。各値は該当する *work* 配列の最初のエントリとして返され、[pxerbla](#) はエラー・メッセージを生成しない。

## 出力パラメータ

*a* 終了時に、 $m \times n$  の分散行列 *Q* のローカル部分が格納される。

*work* 終了時に、*work*(1) は、最小かつ最適な *lwork* 値を返す。

*info* (ローカル) INTEGER。  
*info* = 0 の場合、正常に終了したことを示す。  
*info* < 0 の場合、*i* 番目の引数が配列で *j* 番目の成分の値が不正ならば  $info = -(i*100+j)$ 、  
*i* 番目の引数がスカラで値が不正ならば  $info = -i$ 。

## p?orgr2/p?ungr2

p?gerqf で求めた  $RQ$  因子分解から、全部または一部の直交/ユニタリ行列  $Q$  を生成する (非ブロック化アルゴリズム)。

---

### 構文

```
call psorgr2(m, n, k, a, ia, ja, desca, tau, work, lwork, info)
call pdorgr2(m, n, k, a, ia, ja, desca, tau, work, lwork, info)
call pcungr2(m, n, k, a, ia, ja, desca, tau, work, lwork, info)
call pzungr2(m, n, k, a, ia, ja, desca, tau, work, lwork, info)
```

### 説明

ルーチン p?orgr2/p?ungr2 は、直交列を持つ  $m \times n$  の実 / 複素行列  $Q$  を生成する。これは、次数  $n$  の  $k$  個の基本リフレクタの積の最後の  $m$  行として次のように定義される。ここで、 $Q$  は  $A(ia:ia+m-1, ja:ja+n-1)$  である。

$Q = H(1) H(2) \dots H(k)$  (実数型の場合)  
 $Q = H(1)' H(2)' \dots H(k)'$  (複素数型の場合)

[p?gerqf](#) によって返される。

### 入力パラメータ

$m$	(グローバル) INTEGER。 演算を行う行数、すなわち、分散部分行列 $Q$ の行数。 $m \geq 0$ 。
$n$	(グローバル) INTEGER。 演算を行う列数、すなわち、分散部分行列 $Q$ の列数。 $n \geq m \geq 0$ 。
$k$	(グローバル) INTEGER。 その積が行列 $Q$ を定義する基本リフレクタの個数。 $m \geq k \geq 0$ 。
$a$	REAL (psorgr2 の場合) DOUBLE PRECISION (pdorgr2 の場合) COMPLEX (pcungr2 の場合) COMPLEX*16 (pzungr2 の場合)。ローカルメモリにある、次元 ( $lld\_a$ , $LOCc(ja+n-1)$ ) の配列へのポインタ。 $i$ 番目の行には、 <a href="#">p?gerqf</a> によって分散行列引数



$A(ia+m-k:ia+m-1, ja:*)$  の  $k$  行に返されたとおりに、基本リフレクタ  $H(i)$  を定義するベクトルを格納しておかなければならない。ここで、 $H(i)$  は  $ia+m-k \leq i \leq ia+m-1$ 。

<i>ia</i>	(グローバル) INTEGER。 sub(A) の最初の行を示す、グローバル配列 A の行インデックス。
<i>ja</i>	(グローバル) INTEGER。 sub(A) の最初の列を示す、グローバル配列 A の列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は (dlen_)。 分散行列 A の配列ディスクリプタ。
<i>tau</i>	(ローカル) REAL (psorgl2 の場合) DOUBLE PRECISION (pdorgl2 の場合) COMPLEX (pcungl2 の場合) COMPLEX*16 (pzungl2 の場合)。 配列、次元は $LOCr(ja+m-1)$ 。 <a href="#">p?qerqf</a> から返されたとおりに、基本リフレクタ $H(i)$ のスカラ係数 $tau(i)$ を格納する。分散行列 A に関連付けられる。
<i>work</i>	(ローカル) REAL (psorgr2 の場合) DOUBLE PRECISION (pdorgr2 の場合) COMPLEX (pcungr2 の場合) COMPLEX*16 (pzungr2 の場合)。 ワークスペース配列、次元は (lwork)。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 配列 work の次元。lwork はローカル入力であり、 $lwork \geq nqa0 + \max(1, mpa0)$ でなければならない。 ここで、 $iroffa = \text{mod}(ia-1, mb\_a)$ 、 $icoffa = \text{mod}(ja-1, nb\_a)$ 、 $iarow = \text{indxg2p}(ia, mb\_a, myrow, rsrc\_a, nprow)$ 、 $iacol = \text{indxg2p}(ja, nb\_a, mycol, csrc\_a, npcot)$ 、 $mpa0 = \text{numroc}(m+iroffa, mb\_a, myrow, iarow, nprow)$ 、 $nqa0 = \text{numroc}(n+icoffa, nb\_a, mycol, iacol, npcot)$ 。  $\text{indxg2p}$ および $\text{numroc}$ は、ScaLAPACK ツール関数である。 $myrow$ 、 $mycol$ 、 $nprow$ 、および $npcot$ は、サブルーチン $\text{blacs\_gridinfo}$ を呼び出して求められる。

$lwork = -1$  の場合、 $lwork$  はグローバル入力であり、ワークスペースのクエリとみなされ、ルーチンはすべての配列の最小かつ最適なサイズだけを計算する。各値は該当する  $work$  配列の最初のエントリとして返され、[pxerbla](#) はエラー・メッセージを生成しない。

### 出力パラメータ

$a$	終了時に、 $m \times n$ の分散行列 $Q$ のローカル部分が格納される。
$work$	終了時に、 $work(1)$ は、最小かつ最適な $lwork$ 値を返す。
$info$	(ローカル) INTEGER。 $info = 0$ の場合、正常に終了したことを示す。 $info < 0$ の場合、 $i$ 番目の引数が配列で $j$ 番目の成分の値が不正ならば $info = -(i*100+j)$ 、 $i$ 番目の引数がスカラで値が不正ならば $info = -i$ 。

---

## p?orm2l/p?unm2l

一般行列に `p?geqlf` で求めた  $QL$  因子分解の直交/ユニタリ行列を掛ける (非ブロック化アルゴリズム)。

---

### 構文

```
call psorm2l(side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc, descc,  
            work, lwork, info)  
call pdorm2l(side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc, descc,  
            work, lwork, info)  
call pcunm2l(side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc, descc,  
            work, lwork, info)  
call pzunm2l(side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc, descc,  
            work, lwork, info)
```

### 説明

ルーチン `p?orm2l/p?unm2l` は、 $m \times n$  の一般実/複素分布行列  $sub(C)=C(ic:ic+m-1,jc:jc+n-1)$  を以下で上書きする。

	$side = 'L'$	$side = 'R'$
$trans = 'N'$	$Q * sub(C)$	$sub(C) * Q$
$trans = 'T'$ (実数型の場合)	$Q^T * sub(C)$	$sub(C) * Q^T$
$trans = 'C'$ (複素数型の場合)	$Q^H * sub(C)$	$sub(C) * Q^H$

$Q$  は、 $k$  個の基本リフレクタの積として定義される実直交 / 複素ユニタリ行列である。

$$Q = H(k) \dots H(2) H(1)$$

[p?geqlf](#) によって返される。 $Q$  の次数は、 $side = 'L'$  の場合は  $m$ 、 $side = 'R'$  の場合は  $n$  である。

## 入力パラメータ

$side$	(グローバル) CHARACTER。 'L' の場合、 $Q$ 、 $Q^T$ (実数型の場合) / $Q^H$ (複素数型の場合) を左から適用する。 'R' の場合、 $Q$ 、 $Q^T$ (実数型の場合) / $Q^H$ (複素数型の場合) を右から適用する。
$trans$	(グローバル) CHARACTER。 'N' の場合、 $Q$ を適用する (転置なし) 'T' の場合、 $Q^T$ を適用する (転置、実数型の場合) 'C' の場合、 $Q^H$ を適用する (共役転置、複素数型の場合)。
$m$	(グローバル) INTEGER。 演算を行う行数、すなわち、分散部分行列 $sub(C)$ の行数。 $m \geq 0$ 。
$n$	(グローバル) INTEGER。 演算を行う列数、すなわち、分散部分行列 $sub(C)$ の列数。 $n \geq 0$ 。
$k$	(グローバル) INTEGER。 その積が行列 $Q$ を定義する基本リフレクタの個数。 $side = 'L'$ の場合、 $m \geq k \geq 0$ 。 $side = 'R'$ の場合、 $n \geq k \geq 0$ 。
$a$	(ローカル) REAL (psorm21 の場合) DOUBLE PRECISION (pdorm21 の場合) COMPLEX (pcunm21 の場合) COMPLEX*16 (pzunm21 の場合)。 ローカルメモリにある、次元 ( $lld\_a$ , $LOCc(ja+k-1)$ ) の配列へのポイ

ンタ。

$j$  番目の行には、[p?geqlf](#) によって分散行列引数  $A(ia:*, ja:ja+k-1)$  の  $k$  列に返されたとおりに、基本リフレクタ  $H(j)$  を定義するベクトルを格納しておかなければならない。ここで、 $H(j)$  は  $ja \leq j \leq ja+k-1$  である。引数  $A(ia:*, ja:ja+k-1)$  はルーチンにより変更されるが終了時に復元される。

$side = 'L'$  の場合、 $lld\_a \geq \max(1, LOCr(ia+m-1))$ 。

$side = 'R'$  の場合、 $lld\_a \geq \max(1, LOCr(ia+n-1))$ 。

<i>ia</i>	(グローバル) INTEGER。 sub(A) の最初の行を示す、グローバル配列 A の行インデックス。
<i>ja</i>	(グローバル) INTEGER。 sub(A) の最初の列を示す、グローバル配列 A の列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は (dlen_)。 分散行列 A の配列ディスクリプタ。
<i>tau</i>	(ローカル) REAL (psorm21 の場合) DOUBLE PRECISION (pdorm21 の場合) COMPLEX (pcunm21 の場合) COMPLEX*16 (pzunm21 の場合) 配列、次元は $LOCc(ja+n-1)$ 。 <a href="#">p?geqlf</a> から返されたとおりに、基本リフレクタ $H(j)$ のスカラ係数 $\tau(j)$ を格納する。分散行列 A に関連付けられる。
<i>c</i>	(ローカル) REAL (psorm21 の場合) DOUBLE PRECISION (pdorm21 の場合) COMPLEX (pcunm21 の場合) COMPLEX*16 (pzunm21 の場合)。 ローカルメモリにある、次元 $(lld\_c, LOCc(jc+n-1))$ の配列。 分散行列 sub(C) のローカル部分を格納する。
<i>ic</i>	(グローバル) INTEGER。 sub(C) の最初の行を示す、グローバル配列 C の行インデックス。
<i>jc</i>	(グローバル) INTEGER。 sub(C) の最初の列を示す、グローバル配列 C の列インデックス。
<i>descc</i>	(グローバルおよびローカル) INTEGER。配列、次元は (dlen_)。 分散行列 C の配列ディスクリプタ。

<i>work</i>	<p>(ローカル)</p> <p>REAL (psorm21 の場合)</p> <p>DOUBLE PRECISION (pdorm21 の場合)</p> <p>COMPLEX (pcunm21 の場合)</p> <p>COMPLEX*16 (pzunm21 の場合)。</p> <p>ワークスペース配列、次元は (<i>lwork</i>)。</p> <p>終了時に、<i>work</i>(1) は、最小かつ最適な <i>lwork</i> 値を返す。</p>
<i>lwork</i>	<p>(ローカルまたはグローバル) INTEGER。</p> <p>配列 <i>work</i> の次元。</p> <p><i>lwork</i> はローカル入力であり、次の値でなければならない。</p> <p><i>side</i> = 'L' の場合、<math>lwork \geq mpc0 + \max(1, nqc0)</math>。</p> <p><i>side</i> = 'R' の場合、<math>lwork \geq nqc0 + \max(\max(1, mpc0), \text{numroc}(\text{numroc}(n+icoffc, nb\_a, 0, 0, npc0), nb\_a, 0, 0, lcmq))</math>。</p> <p>ここで、<math>lcmq = lcm / npc0</math>、<math>lcm = iclm(nprow, npc0)</math>、</p> <p><math>iroffc = \text{mod}(ic-1, mb\_c)</math>、<math>icoffc = \text{mod}(jc-1, nb\_c)</math>、</p> <p><math>icrow = \text{indxg2p}(ic, mb\_c, myrow, rsrc\_c, nprow)</math>、</p> <p><math>iccol = \text{indxg2p}(jc, nb\_c, mycol, csrc\_c, npc0)</math>、</p> <p><math>Mqc0 = \text{numroc}(m+icoffc, nb\_c, mycol, icrow, nprow)</math>、</p> <p><math>Npc0 = \text{numroc}(n+iroffc, mb\_c, myrow, iccol, npc0)</math>。</p> <p><math>iclm</math>、<math>\text{indxg2p}</math> および <math>\text{numroc}</math> は、ScaLAPACK ツール関数である。</p> <p><math>myrow</math>、<math>mycol</math>、<math>nprow</math>、および <math>npc0</math> は、サブルーチン <code>blacs_gridinfo</code> を呼び出して求められる。</p> <p><i>lwork</i> = -1 の場合、<i>lwork</i> はグローバル入力であり、ワークスペースのクエリとみなされ、ルーチンはすべての配列の最小かつ最適なサイズだけを計算する。各値は該当する <i>work</i> 配列の最初のエン트리として返され、<a href="#">pxerbla</a> はエラー・メッセージを生成しない。</p>

## 出力パラメータ

<i>c</i>	<p>終了時に、<i>sub</i>(<i>C</i>) は、<math>Q \cdot \text{sub}(C)</math>、<math>Q' \cdot \text{sub}(C)</math>、<math>\text{sub}(C) \cdot Q'</math>、または <math>\text{sub}(C) \cdot Q</math> のいずれかで上書きされる。</p>
<i>work</i>	<p>終了時に、<i>work</i>(1) は、最小かつ最適な <i>lwork</i> 値を返す。</p>
<i>info</i>	<p>(ローカル) INTEGER。</p> <p><i>info</i> = 0 の場合、正常に終了したことを示す。</p> <p><i>info</i> &lt; 0 の場合、<i>i</i> 番目の引数が配列で <i>j</i> 番目の成分の値が不正ならば <math>info = -(i \cdot 100 + j)</math>、<i>i</i> 番目の引数がスカラで値が不正ならば <math>info = -i</math>。</p>



**注:** 分散部分行列  $A(ia:*, ja:*)$  と  $C(ic:ic+m-1, jc:jc+n-1)$  は、次の式を満たすようにアライメントされていなければならない。  
 $side = 'L'$  の場合、 $(mb\_a.eq.mb\_c \ .AND. \ iroffa.eq.iroffc \ .AND. \ iarow.eq.icrow)$ 。  
 $side = 'R'$  の場合、 $(mb\_a.eq.nb\_c \ .AND. \ iroffa.eq.iroffc)$ 。

## p?orm2r/p?unm2r

一般行列に p?geqrf で求めた **QR** 因子分解の直交 / ユニタリ行列を掛ける ( 非ブロック化アルゴリズム )。

### 構文

```
call psorm2r(side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc, descc,
work, lwork, info)
call pdorm2r(side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc, descc,
work, lwork, info)
call pcunm2r(side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc, descc,
work, lwork, info)
call pzunm2r(side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc, descc,
work, lwork, info)
```

### 説明

ルーチン p?orm2r/p?unm2r は、 $m \times n$  の一般実 / 複素分布行列  $sub(C)=C(ic:ic+m-1, jc:jc+n-1)$  を以下で上書きする。

	$side = 'L'$	$side = 'R'$
$trans = 'N'$	$Q * sub(C)$	$sub(C) * Q$
$trans = 'T'$ ( 実数型の場合 )	$Q^T * sub(C)$	$sub(C) * Q^T$
$trans = 'C'$ ( 複素数型の場合 )	$Q^H * sub(C)$	$sub(C) * Q^H$

$Q$  は、 $k$  個の基本リフレクタの積として定義される実直交 / 複素ユニタリ行列である。

$$Q = H(k) \dots H(2) H(1)$$

[p?gqgrf](#) によって返される。 $Q$  の次数は、 $side = 'L'$  の場合は  $m$ 、 $side = 'R'$  の場合は  $n$  である。

## 入力パラメータ

<i>side</i>	(グローバル) CHARACTER。 'L' の場合、 $Q$ 、 $Q^T$ (実数型の場合)/ $Q^H$ (複素数型の場合) を左から適用する。 'R' の場合、 $Q$ 、 $Q^T$ (実数型の場合)/ $Q^H$ (複素数型の場合) を右から適用する。
<i>trans</i>	(グローバル) CHARACTER。 'N' の場合、 $Q$ を適用する (転置なし) 'T' の場合、 $Q^T$ を適用する (転置、実数型の場合) 'C' の場合、 $Q^H$ を適用する (共役転置、複素数型の場合)。
<i>m</i>	(グローバル) INTEGER。 演算を行う行数、すなわち、分散部分行列 $sub(C)$ の行数。 $m \geq 0$ 。
<i>n</i>	(グローバル) INTEGER。 演算を行う列数、すなわち、分散部分行列 $sub(C)$ の列数。 $n \geq 0$ 。
<i>k</i>	(グローバル) INTEGER。 その積が行列 $Q$ を定義する基本リフレクタの個数。 $side = 'L'$ の場合、 $m \geq k \geq 0$ 。 $side = 'R'$ の場合、 $n \geq k \geq 0$ 。
<i>a</i>	(ローカル) REAL (psorm2r の場合) DOUBLE PRECISION (pdorm2r の場合) COMPLEX (pcunm2r の場合) COMPLEX*16 (pzunm2r の場合)。 ローカルメモリにある、次元 ( $lld\_a$ , $LOCc(ja+k-1)$ ) の配列へのポインタ。 $j$ 番目の列には、 <a href="#">p?gqgrf</a> によって分散行列引数 $A(ia:*, ja:ja+k-1)$ の $k$ 列に返されたとおりに、基本リフレクタ $H(j)$ を定義するベクトルを格納しておかなければならない。ここで、 $H(j)$ は $ja \leq j \leq ja+k-1$ である。引数 $A(ia:*, ja:ja+k-1)$ はルーチンにより変更されるが終了時に復元される。 $side = 'L'$ の場合、 $lld\_a \geq \max(1, LOCr(ia+m-1))$ 。 $side = 'R'$ の場合、 $lld\_a \geq \max(1, LOCr(ia+n-1))$ 。

<i>ia</i>	(グローバル) INTEGER。 sub(A) の最初の行を示す、グローバル配列 A の行インデックス。
<i>ja</i>	(グローバル) INTEGER。 sub(A) の最初の列を示す、グローバル配列 A の列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。 分散行列 A の配列ディスクリプタ。
<i>tau</i>	(ローカル) REAL (psorm2r の場合) DOUBLE PRECISION (pdorm2r の場合) COMPLEX (pcunm2r の場合) COMPLEX*16 (pzunm2r の場合)。 配列、次元は <i>LOCc(ja+k-1)</i> 。 <a href="#">p?geqrf</a> から返されたとおりに、基本リフレクタ <i>H(j)</i> のスカラ係数 <i>tau(j)</i> を格納する。分散行列 A に関連付けられる。
<i>c</i>	(ローカル) REAL (psorm2r の場合) DOUBLE PRECISION (pdorm2r の場合) COMPLEX (pcunm2r の場合) COMPLEX*16 (pzunm2r の場合)。 ローカルメモリにある、次元 ( <i>lld_c, LOCc(jc+n-1)</i> ) の配列へのポインタ。分散行列 sub (C) のローカル部分を格納する。
<i>ic</i>	(グローバル) INTEGER。 sub(C) の最初の行を示す、グローバル配列 C の行インデックス。
<i>jc</i>	(グローバル) INTEGER。 sub(C) の最初の列を示す、グローバル配列 C の列インデックス。
<i>descc</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。 分散行列 C の配列ディスクリプタ。
<i>work</i>	(ローカル) REAL (psorm2r の場合) DOUBLE PRECISION (pdorm2r の場合) COMPLEX (pcunm2r の場合) COMPLEX*16 (pzunm2r の場合)。 ワークスペース配列、次元は ( <i>lwork</i> )。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 配列 work の次元。 <i>lwork</i> はローカル入力であり、次の値でなければならない。



$side = 'L'$  の場合、 $lwork \geq mpc0 + \max(1, nqc0)$ 。  
 $side = 'R'$  の場合、 $lwork \geq nqc0 + \max(\max(1, mpc0), \text{numroc}(\text{numroc}(n+icoffc, nb\_a, 0, 0, npc0), nb\_a, 0, 0, lcmq))$ 。

ここで、 $lcmq = lcm / npc0$ 、 $lcm = iclm(nprow, npc0)$ 、

$iroffc = \text{mod}(ic-1, mb\_c)$ 、 $icoffc = \text{mod}(jc-1, nb\_c)$ 、  
 $icrow = \text{indxg2p}(ic, mb\_c, myrow, rsrc\_c, nprow)$ 、  
 $iccol = \text{indxg2p}(jc, nb\_c, mycol, csrc\_c, npc0)$ 、  
 $Mqc0 = \text{numroc}(m+icoffc, nb\_c, mycol, icrow, nprow)$ 、  
 $Npc0 = \text{numroc}(n+iroffc, mb\_c, myrow, iccol, npc0)$ 。

$iclm$ 、 $\text{indxg2p}$  および  $\text{numroc}$  は、ScaLAPACK ツール関数である。  
 $myrow$ 、 $mycol$ 、 $nprow$ 、および  $npc0$  は、サブルーチン  $\text{blacs\_gridinfo}$  を呼び出して求められる。

$lwork = -1$  の場合、 $lwork$  はグローバル入力であり、ワークスペースのクエリとみなされ、ルーチンはすべての配列の最小かつ最適なサイズだけを計算する。各値は該当する  $work$  配列の最初のエン트리として返され、[pxerbla](#) はエラー・メッセージを生成しない。

## 出力パラメータ

$c$	終了時に、 $\text{sub}(C)$ は、 $Q * \text{sub}(C)$ 、 $Q' * \text{sub}(C)$ 、 $\text{sub}(C) * Q'$ 、または $\text{sub}(C) * Q$ のいずれかで上書きされる。
$work$	終了時に、 $work(1)$ は、最小かつ最適な $lwork$ 値を返す。
$info$	(ローカル) INTEGER。 $info = 0$ の場合、正常に終了したことを示す。 $info < 0$ の場合、 $i$ 番目の引数が配列で $j$ 番目の成分の値が不正ならば $info = -(i * 100 + j)$ 、 $i$ 番目の引数がスカラで値が不正ならば $info = -i$ 。



**注:** 分散部分行列  $A(ia:*, ja:*)$  と  $C(ic:ic+m-1, jc:jc+n-1)$  は、次の式を満たすようにアライメントされていなければならない。

$side = 'L'$  の場合、 $(mb\_a.eq.mb\_c .AND. iroffa.eq.iroffc .AND. iarow.eq.icrow)$ 。

$side = 'R'$  の場合、 $(mb\_a.eq.nb\_c .AND. iroffa.eq.iroffc)$ 。

## p?orml2/p?unml2

一般行列に p?gelqf で求めた  $LQ$  因子分解の直交 / ユニタリ行列を掛ける (非ブロック化アルゴリズム)。

### 構文

```
call psorml2(side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc, descc,
             work, lwork, info)
call pdorml2(side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc, descc,
             work, lwork, info)
call pcunml2(side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc, descc,
             work, lwork, info)
call pzunml2(side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc, descc,
             work, lwork, info)
```

### 説明

ルーチン p?orml2/p?unml2 は、 $m \times n$  の一般実 / 複素分布行列  $\text{sub}(C) = C(ic:ic+m-1, jc:jc+n-1)$  を以下で上書きする。

	$side = 'L'$	$side = 'R'$
$trans = 'N'$	$Q * \text{sub}(C)$	$\text{sub}(C) * Q$
$trans = 'T'$ (実数型の場合)	$Q^T * \text{sub}(C)$	$\text{sub}(C) * Q^T$
$trans = 'C'$ (複素数型の場合)	$Q^H * \text{sub}(C)$	$\text{sub}(C) * Q^H$

$Q$  は、 $k$  個の基本リフレクタの積として定義される実直交 / 複素ユニタリ分散行列である。

$$Q = H(k) \dots H(2) H(1) \text{ (実数型の場合)}$$

$$Q = H(k)^H \dots H(2)^H H(1)^H \text{ (複素数型の場合)}$$

[p?gelqf](#) によって返される。  $Q$  の次数は、 $side = 'L'$  の場合は  $m$ 、 $side = 'R'$  の場合は  $n$  である。

## 入力パラメータ

<i>side</i>	(グローバル) CHARACTER。 'L' の場合、 $Q$ 、 $Q^T$ (実数型の場合) / $Q^H$ (複素数型の場合) を左から適用する。 'R' の場合、 $Q$ 、 $Q^T$ (実数型の場合) / $Q^H$ (複素数型の場合) を右から適用する。
<i>trans</i>	(グローバル) CHARACTER。 'N' の場合、 $Q$ を適用する (転置なし) 'T' の場合、 $Q^T$ を適用する (転置、実数型の場合) 'C' の場合、 $Q^H$ を適用する (共役転置、複素数型の場合)
<i>m</i>	(グローバル) INTEGER。 演算を行う行数、すなわち、分散部分行列 $\text{sub}(C)$ の行数。 $m \geq 0$ 。
<i>n</i>	(グローバル) INTEGER。 演算を行う列数、すなわち、分散部分行列 $\text{sub}(C)$ の列数。 $n \geq 0$ 。
<i>k</i>	(グローバル) INTEGER。 その積が行列 $Q$ を定義する基本リフレクタの個数。 <i>side</i> = 'L' の場合、 $m \geq k \geq 0$ 。 <i>side</i> = 'R' の場合、 $n \geq k \geq 0$ 。
<i>a</i>	(ローカル) REAL (psorml2 の場合) DOUBLE PRECISION (pdorml2 の場合) COMPLEX (pcunml2 の場合) COMPLEX*16 (pzunml2 の場合)。 ローカルメモリにある、次元 ( $lld\_a$ , $LOCc(ja+m-1)$ ) ( <i>side</i> ='L' の場合) または ( $lld\_a$ , $LOCc(ja+n-1)$ ) ( <i>side</i> ='R' の場合) の配列へのポインタ。ここで、 $lld\_a \geq \max(1, LOCr(ia+k-1))$ 。 $i$ 番目の行には、 <a href="#">p?gelqf</a> によって分散行列引数 $A(ia:ia+k-1, ja:*)$ の $k$ 行に返されたとおりに、基本リフレクタ $H(i)$ を定義するベクトルを格納しておかなければならない。ここで、 $H(i)$ は $ia \leq i \leq ia+k-1$ 。引数 $A(ia:ia+k-1, ja:*)$ はルーチンにより変更されるが終了時に復元される。
<i>ia</i>	(グローバル) INTEGER。 $\text{sub}(A)$ の最初の行を示す、グローバル配列 $A$ の行インデックス。
<i>ja</i>	(グローバル) INTEGER。 $\text{sub}(A)$ の最初の列を示す、グローバル配列 $A$ の列インデックス。

<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。分散行列 <i>A</i> の配列ディスクリプタ。
<i>tau</i>	(ローカル) REAL ( <i>psorml2</i> の場合) DOUBLE PRECISION ( <i>pdorml2</i> の場合) COMPLEX ( <i>pcunml2</i> の場合) COMPLEX*16 ( <i>pzunml2</i> の場合)。 配列、次元は <i>LOCc(ia+k-1)</i> 。 <a href="#">p?qelqf</a> から返されたとおりに、基本リフレクタ <i>H(i)</i> のスカラ係数 <i>tau(i)</i> を格納する。分散行列 <i>A</i> に関連付けられる。
<i>c</i>	(ローカル) REAL ( <i>psorml2</i> の場合) DOUBLE PRECISION ( <i>pdorml2</i> の場合) COMPLEX ( <i>pcunml2</i> の場合) COMPLEX*16 ( <i>pzunml2</i> の場合)。 ローカルメモリにある、次元 ( <i>lld_c, LOCc(jc+n-1)</i> ) の配列へのポインタ。分散行列 <i>sub(C)</i> のローカル部分を格納する。
<i>ic</i>	(グローバル) INTEGER。 <i>sub(C)</i> の最初の行を示す、グローバル配列 <i>C</i> の行インデックス。
<i>jc</i>	(グローバル) INTEGER。 <i>sub(C)</i> の最初の列を示す、グローバル配列 <i>C</i> の列インデックス。
<i>descc</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。分散行列 <i>C</i> の配列ディスクリプタ。
<i>work</i>	(ローカル) REAL ( <i>psorml2</i> の場合) DOUBLE PRECISION ( <i>pdorml2</i> の場合) COMPLEX ( <i>pcunml2</i> の場合) COMPLEX*16 ( <i>pzunml2</i> の場合)。 ワークスペース配列、次元は ( <i>lwork</i> )。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 配列 <i>work</i> の次元。 <i>lwork</i> はローカル入力であり、次の値でなければならない。 <i>side</i> = 'L' の場合、 $lwork \geq mqc0 + \max(\max(1, npc0), \text{numroc}(\text{numroc}(m+icoffc, mb\_a, 0, 0, nprow), mb\_a, 0, 0, lcmp))$ 。 <i>side</i> = 'R' の場合、 $lwork \geq npc0 + \max(1, mqc0)$ 。 ここで、 $lcmp = lcm / nprow$ 、 $lcm = iclm(nprow, npc0)$ 、

```

iroffc = mod( ic-1, mb_c ), icoffc = mod( jc-1, nb_c ),
icrow = indxg2p( ic, mb_c, myrow, rsrc_c, nprow ),
iccol = indxg2p( jc, nb_c, mycol, csrc_c, npc0l ),
Mpc0 = numroc( m+icoffc, mb_c, mycol, icrow, nprow ),
Nqc0 = numroc( n+iroffc, nb_c, myrow, iccol, npc0l ).

```

ilcm、indxg2p および numroc は、ScaLAPACK ツール関数である。  
myrow、mycol、nprow、および npc0l は、サブルーチン  
blacs\_gridinfo を呼び出して求められる。

lwork = -1 の場合、lwork はグローバル入力であり、ワークスペース  
のクエリとみなされ、ルーチンはすべての配列の最小かつ最適なサイ  
ズだけを計算する。各値は該当する work 配列の最初のエントリとし  
て返され、[pxerbla](#) はエラー・メッセージを生成しない。

## 出力パラメータ

<i>c</i>	終了時に、sub( <i>C</i> ) は、 $Q \cdot \text{sub}(C)$ 、 $Q' \cdot \text{sub}(C)$ 、 $\text{sub}(C) \cdot Q'$ 、または $\text{sub}(C) \cdot Q$ のいずれかで上書きされる。
<i>work</i>	終了時に、work(1) は、最小かつ最適な lwork 値を返す。
<i>info</i>	(ローカル) INTEGER。 info = 0 の場合、正常に終了したことを示す。 info < 0 の場合、 <i>i</i> 番目の引数が配列で <i>j</i> 番目の成分の値が不正なら ば info = -( <i>i</i> *100+ <i>j</i> )、 <i>i</i> 番目の引数がスカラーで値が不正ならば info = - <i>i</i> 。



**注:** 分散部分行列  $A(ia:*, ja:*)$  と  $C(ic:ic+m-1, jc:jc+n-1)$  は、  
次の式を満たすようにアライメントされていなければならない。

side = 'L' の場合、(nb\_a.eq.mb\_c .AND.

icoffa.eq.iroffc)。

side = 'R' の場合、(nb\_a.eq.nb\_c .AND. icoffa.eq.icoffc  
.AND. iacol.eq.iccol)。

## p?ormr2/p?unmr2

一般行列に p?gerqf で求めた  $RQ$  因子分解の直交/ユニタリ行列を掛ける (非ブロック化アルゴリズム)。

### 構文

```
call psormr2(side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc, descc,
             work, lwork, info)
call pdormr2(side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc, descc,
             work, lwork, info)
call pcunmr2(side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc, descc,
             work, lwork, info)
call pzunmr2(side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc, descc,
             work, lwork, info)
```

### 説明

ルーチン p?ormr2/p?unmr2 は、 $m \times n$  の一般実 / 複素分布行列  $\text{sub}(C) = C(ic:ic+m-1, jc:jc+n-1)$  を以下で上書きする。

	$side = 'L'$	$side = 'R'$
$trans = 'N'$	$Q * \text{sub}(C)$	$\text{sub}(C) * Q$
$trans = 'T'$ (実数型の場合)	$Q^T * \text{sub}(C)$	$\text{sub}(C) * Q^T$
$trans = 'C'$ (複素数型の場合)	$Q^H * \text{sub}(C)$	$\text{sub}(C) * Q^H$

$Q$  は、 $k$  個の基本リフレクタの積として定義される実直交 / 複素ユニタリ分散行列である。

$Q = H(1) H(2) \dots H(k)$  (実数型の場合)  
 $Q = H(1)^H H(2)^H \dots H(k)^H$  (複素数型の場合)

[p?gerqf](#) によって返される。 $Q$  の次数は、 $side = 'L'$  の場合は  $m$ 、 $side = 'R'$  の場合は  $n$  である。

## 入力パラメータ

<i>side</i>	(グローバル) CHARACTER。 'L' の場合、 $Q$ 、 $Q^T$ (実数型の場合) / $Q^H$ (複素数型の場合) を左から適用する。 'R' の場合、 $Q$ 、 $Q^T$ (実数型の場合) / $Q^H$ (複素数型の場合) を右から適用する。
<i>trans</i>	(グローバル) CHARACTER。 'N' の場合、 $Q$ を適用する (転置なし) 'T' の場合、 $Q^T$ を適用する (転置、実数型の場合) 'C' の場合、 $Q^H$ を適用する (共役転置、複素数型の場合)
<i>m</i>	(グローバル) INTEGER。 演算を行う行数、すなわち、分散部分行列 $\text{sub}(C)$ の行数。 $m \geq 0$ 。
<i>n</i>	(グローバル) INTEGER。 演算を行う列数、すなわち、分散部分行列 $\text{sub}(C)$ の列数。 $n \geq 0$ 。
<i>k</i>	(グローバル) INTEGER。 その積が行列 $Q$ を定義する基本リフレクタの個数。 <i>side</i> = 'L' の場合、 $m \geq k \geq 0$ 。 <i>side</i> = 'R' の場合、 $n \geq k \geq 0$ 。
<i>a</i>	(ローカル) REAL (psormr2 の場合) DOUBLE PRECISION (pdormr2 の場合) COMPLEX (pcunmr2 の場合) COMPLEX*16 (pzunmr2 の場合)。 ローカルメモリにある、次元 ( $lld\_a$ , $LOCc(ja+m-1)$ ) ( <i>side</i> ='L' の場合) または ( $lld\_a$ , $LOCc(ja+n-1)$ ) ( <i>side</i> ='R' の場合) の配列へのポインタ。ここで、 $lld\_a \geq \max(1, LOCr(ia+k-1))$ 。 <i>i</i> 番目の行には、 <a href="#">p?gerqf</a> によって分散行列引数 $A(ia:ia+k-1, ja:*)$ の <i>k</i> 行に返されたとおりに、基本リフレクタ $H(i)$ を定義するベクトルを格納しておかなければならない。 ここで、 $H(i)$ は $ia \leq i \leq ia+k-1$ 。引数 $A(ia:ia+k-1, ja:*)$ はルーチンにより変更されるが終了時に復元される。
<i>ia</i>	(グローバル) INTEGER。 $\text{sub}(A)$ の最初の行を示す、グローバル配列 <i>A</i> の行インデックス。
<i>ja</i>	(グローバル) INTEGER。 $\text{sub}(A)$ の最初の列を示す、グローバル配列 <i>A</i> の列インデックス。

<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。分散行列 <i>A</i> の配列ディスクリプタ。
<i>tau</i>	(ローカル) REAL ( <i>psormr2</i> の場合) DOUBLE PRECISION ( <i>pdormr2</i> の場合) COMPLEX ( <i>pcunmr2</i> の場合) COMPLEX*16 ( <i>pzunmr2</i> の場合)。 配列、次元は <i>LOCc(ia+k-1)</i> 。 <a href="#">p?qerqf</a> から返されたとおりに、基本リフレクタ <i>H(i)</i> のスカラ係数 <i>tau(i)</i> を格納する。分散行列 <i>A</i> に関連付けられる。
<i>c</i>	(ローカル) REAL ( <i>psormr2</i> の場合) DOUBLE PRECISION ( <i>pdormr2</i> の場合) COMPLEX ( <i>pcunmr2</i> の場合) COMPLEX*16 ( <i>pzunmr2</i> の場合) ローカルメモリにある、次元 ( <i>lld_c, LOCc(jc+n-1)</i> ) の配列へのポインタ。分散行列 <i>sub(C)</i> のローカル部分を格納する。
<i>ic</i>	(グローバル) INTEGER。 <i>sub(C)</i> の最初の行を示す、グローバル配列 <i>C</i> の行インデックス。
<i>jc</i>	(グローバル) INTEGER。 <i>sub(C)</i> の最初の列を示す、グローバル配列 <i>C</i> の列インデックス。
<i>descc</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。分散行列 <i>C</i> の配列ディスクリプタ。
<i>work</i>	(ローカル) REAL ( <i>psormr2</i> の場合) DOUBLE PRECISION ( <i>pdormr2</i> の場合) COMPLEX ( <i>pcunmr2</i> の場合) COMPLEX*16 ( <i>pzunmr2</i> の場合) ワークスペース配列、次元は ( <i>lwork</i> )。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 配列 <i>work</i> の次元。 <i>lwork</i> はローカル入力であり、次の値でなければならない。 <i>side</i> = 'L' の場合、 $lwork \geq mpc0 + \max(\max(1, nqc0), \text{numroc}(\text{numroc}(m+iroffc, mb\_a, 0, 0, nprow), mb\_a, 0, 0, lcmp))$ 。 <i>side</i> = 'R' の場合、 $lwork \geq nqc0 + \max(1, mpc0)$ 。 ここで、 $lcmp = lcm / nprow$ 、 $lcm = iclm(nprow, npc0l)$ 、



```

iroffc = mod( ic-1, mb_c ), icoffc = mod( jc-1, nb_c ),
icrow = indxg2p( ic, mb_c, myrow, rsrc_c, nprow ),
iccol = indxg2p( jc, nb_c, mycol, csrc_c, npc1 ),
Mpc0 = numroc( m+iroffc, mb_c, myrow, icrow, nprow ),
Nqc0 = numroc( n+icoffc, nb_c, mycol, iccol, npc1 ).

```

ilcm、indxg2p および numroc は、ScaLAPACK ツール関数である。  
myrow、mycol、nprow、および npc1 は、サブルーチン  
blacs\_gridinfo を呼び出して求められる。

lwork = -1 の場合、lwork はグローバル入力であり、ワークスペース  
のクエリとみなされ、ルーチンはすべての配列の最小かつ最適なサイ  
ズだけを計算する。各値は該当する work 配列の最初のエントリとし  
て返され、[pxerbla](#) はエラー・メッセージを生成しない。

### 出力パラメータ

<i>c</i>	終了時に、sub( <i>C</i> ) は、 $Q \cdot \text{sub}(C)$ 、 $Q' \cdot \text{sub}(C)$ 、 $\text{sub}(C) \cdot Q'$ 、または $\text{sub}(C) \cdot Q$ のいずれかで上書きされる。
<i>work</i>	終了時に、work(1) は、最小かつ最適な lwork 値を返す。
<i>info</i>	(ローカル) INTEGER。 info = 0 の場合、正常に終了したことを示す。 info < 0 の場合、 <i>i</i> 番目の引数が配列で <i>j</i> 番目の成分の値が不正なら ば info = -( <i>i</i> *100+ <i>j</i> )、 <i>i</i> 番目の引数がスカラーで値が不正ならば info = - <i>i</i> 。



**注:** 分散部分行列  $A(ia:*, ja:*)$  と  $C(ic:ic+m-1, jc:jc+n-1)$  は、  
次の式を満たすようにアライメントされていなければならない。  
side = 'L' の場合、(nb\_a.eq.mb\_c .AND. icoffa.eq.iroffc)。  
side = 'R' の場合、(nb\_a.eq.nb\_c .AND. icoffa.eq.icoffc  
.AND. iacol.eq.iccol)。

## p?pbtrsv

前進解法または後退解法によって、単一の三角行列線形問題を解く。ここで、三角行列は p?pbtrf で計算された帯行列の係数である。

### 構文

```
call pspbtrsv(uplo, trans, n, bw, nrhs, a, ja, desca, b, ib, descb, af,
             laf, work, lwork, info)
call pdpbtrsv(uplo, trans, n, bw, nrhs, a, ja, desca, b, ib, descb, af,
             laf, work, lwork, info)
call pcpbtrsv(uplo, trans, n, bw, nrhs, a, ja, desca, b, ib, descb, af,
             laf, work, lwork, info)
call pzpbttrsv(uplo, trans, n, bw, nrhs, a, ja, desca, b, ib, descb, af,
             laf, work, lwork, info)
```

### 説明

ルーチン p?pbtrsv は、次の三角帯行列を係数行列とする連立 1 次方程式を解く。

$$A(1:n, ja:ja+n-1) * X = B(jb:jb+n-1, 1:nrhs)$$

または

$$A(1:n, ja:ja+n-1)^T * X = B(jb:jb+n-1, 1:nrhs) \text{ (実数型の場合)}$$

$$A(1:n, ja:ja+n-1)^H * X = B(jb:jb+n-1, 1:nrhs) \text{ (複素数型の場合)}$$

ここで、 $A(1:n, ja:ja+n-1)$  はコレスキー因子分解コード p?pbtrf によって生成される三角帯行列であり、 $A(1:n, ja:ja+n-1)$  と  $af$  に格納される。 $A(1:n, ja:ja+n-1)$  には、 $uplo$  の値に従って、上三角行列または下三角行列が格納される。 $A(1:n, ja:ja+n-1)$  または  $A(1:n, ja:ja+n-1)^T$  (実数型の場合) と  $A(1:n, ja:ja+n-1)^H$  (複素数型の場合) のどの式を解くのかは、パラメータ  $trans$  を介して、ユーザによって決定される。

ルーチン [p?pbtrf](#) は、最初に呼び出さなければならない。

### 入力パラメータ

$uplo$  (グローバル) CHARACTER。'U' または 'L' でなければならない。

$uplo = 'U'$  の場合、 $A(1:n, ja:ja+n-1)$  の上三角行列が格納される。

$uplo = 'L'$  の場合、 $A(1:n, ja:ja+n-1)$  の下三角行列が格納される。

<i>trans</i>	(グローバル) CHARACTER。 'N'、'T'、または 'C' のいずれかでなければならない。  <i>trans</i> = 'N' の場合、 $A(1:n, ja:ja+n-1)$ を解く。  <i>trans</i> = 'T' または 'C' (実数型) の場合、 $A(1:n, ja:ja+n-1)^T$ を解く。  <i>trans</i> = 'C' (複素数型) の場合、 $\text{conjugate\_transpose}(A(1:n, ja:ja+n-1))$ を解く。
<i>n</i>	(グローバル) INTEGER。演算を行う行数と列数、すなわち、分散部分行列 $A(1:n, ja:ja+n-1)$ の次数。 $Bn \geq 0$ 。
<i>bw</i>	(グローバル) INTEGER。'L' または 'U' の劣対角成分の数。 $0 \leq bw \leq n-1$ 。
<i>nrhs</i>	(グローバル) INTEGER。右辺の数。分散部分行列 $B(jb:jb+n-1, 1:nrhs)$ の列数。 ( $nrhs \geq 0$ )。
<i>a</i>	(ローカル)  REAL (pspbtrsv の場合) DOUBLE PRECISION (pdpbtrsv の場合) COMPLEX (pcpbtrsv の場合) COMPLEX*16 (pzpbtrsv の場合)。 ローカルメモリにある、第 1 次元が $lld\_a \geq (bw+1)$ の配列へのポインタ ( <i>desca</i> に格納される)。  $n \times n$ の対称帯形式の分割コレスキー因子 $L$ または $L^T A(1:n, ja:ja+n-1)$ のローカル部分を格納する。  このローカル部分は、LAPACK で使用される圧縮帯形式で格納される。分散行列の形式の詳細は、次の「アプリケーション・ノート」および ScaLAPACK マニュアルを参照。
<i>ja</i>	(グローバル) INTEGER。(A のすべてまたは A の部分行列で) 処理される行列の先頭を指すグローバル配列 A のインデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen</i> )。分散行列 A の配列ディスクリプタ。 1D type ( <i>dtype_a</i> = 501) の場合、 $dlen \geq 7$ 。 2D type ( <i>dtype_a</i> = 1) の場合、 $dlen \geq 9$ 。 A のマッピング情報をメモリに格納する。詳細は、ScaLAPACK マニュアルを参照。
<i>b</i>	(ローカル)

	<p>REAL (pspbtrsv の場合 )</p> <p>DOUBLE PRECISION (pdpbtrsv の場合 )</p> <p>COMPLEX (pcpbtrsv の場合 )</p> <p>COMPLEX*16 (pzpbtrsv の場合 )。</p> <p>ローカルメモリにある、ローカル・リーディング・ディメンジョン <math>lld\_b \geq nb</math> の配列へのポインタ。</p> <p>右辺 <math>B(jb:jb+n-1, 1:nrhs)</math> のローカル部分を格納する。</p>
<i>ib</i>	(グローバル) INTEGER。( $B$ のすべてまたは $B$ の部分行列で ) 処理される行列の最初の行を指すグローバル配列 $B$ の行インデックス。
<i>descb</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen</i> )。分散行列 $B$ の配列ディスクリプタ。
	<p>1D type (<i>dtype_a</i> = 502) の場合、<i>dlen</i> <math>\geq</math> 7。</p> <p>2D type (<i>dtype_b</i> = 1) の場合、<i>dlen</i> <math>\geq</math> 9。</p> <p><math>B</math> のマッピング情報をメモリに格納する。詳細は、ScaLAPACK マニュアルを参照。</p>
<i>laf</i>	(ローカル) INTEGER。ユーザ入力の補助非零要素空間 <i>af</i> のサイズ。 $laf \geq (nb+2*bw)*bw$ でなければならない。 $laf$ が小さすぎる場合、エラーコードが返され、エラーにならない最小のサイズが <i>af</i> (1) に返される。
<i>work</i>	(ローカル) <p>REAL (pspbtrsv の場合 )</p> <p>DOUBLE PRECISION (pdpbtrsv の場合 )</p> <p>COMPLEX (pcpbtrsv の場合 )</p> <p>COMPLEX*16 (pzpbtrsv の場合 )。</p> <p>配列 <i>work</i> は、次元 <i>lwork</i> の一次ワークスペース配列。このスペースは、ルーチンの呼び出しの間に上書きされる。</p>
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。ユーザー入力の <i>work</i> のサイズは、 $lwork \geq bw*nrhs$ 以上でなければならない。 <i>lwork</i> が小さすぎる場合、エラーコードが戻され、 <i>work</i> (1) に最小許容サイズが格納される。

## 出力パラメータ

<i>af</i>	<p>(ローカル)</p> <p>REAL (pspbtrsv の場合 )</p> <p>DOUBLE PRECISION (pdpbtrsv の場合 )</p> <p>COMPLEX (pcpbtrsv の場合 )</p> <p>COMPLEX*16 (pzpbtrsv の場合 )。</p>
-----------	---

配列  $af$  の次元は  $la_f$ 。補助非零要素が格納される。非零要素は、因子分解ルーチン `p?dttrf` で作成され、 $af$  に格納される。因子分解ルーチンの後で、`p?pbtrs` を使って連立 1 次方程式を解く場合、因子分解後に  $af$  を変更してはならない。

$b$  終了時に、解の分散行列  $X$  のローカル部分が格納される。

$work(1)$  終了時に、 $lwork$  の最小値が  $work(1)$  に格納される。

$info$  (ローカル) INTEGER。  
 $info = 0$  の場合、正常に終了したことを示す。  
 $info < 0$  の場合、 $i$  番目の引数が配列で  $j$  番目の成分の値が不正ならば  $info = -(i*100+j)$ 、 $i$  番目の引数がスカラで値が不正ならば  $info = -i$ 。

### アプリケーション・ノート

同じ係数行列を使用して右辺のさまざまなセットを解くために、因子分解ルーチンと算出ルーチンを別々に呼び出す場合、補助空間  $af$  は因子分解ルーチンと算出ルーチンの呼び出しの間に変更してはならない。

帯行列と三重対角行列を係数行列とする連立 1 次方程式を解くための最良のアルゴリズムは、さまざまなパラメータ (特に、帯幅) に依存する。現時点では、 $N/P \gg bw$  用のアルゴリズムのみ実装されている。これらのアルゴリズムには、分割統治、分割、領域分解などがある。

#### アルゴリズムの説明：分割統治 \*

分割統治アルゴリズムは、方程式の数と比較して、行列が綿密に帯格納されているのものと仮定する。このような場合、アトミックな列とプロセス間で分割された行と共に、入力行列  $A$  を 1 次元で分配するのが最良である。基本アルゴリズムは、帯行列を 1 つのブロックが 1 つのプロセッサに格納されるよう  $P$  個に分割し、フェーズ 2 の因子分解またはフェーズ 3 の連立 1 次方程式の算出に進む。

5. **ローカルフェーズ**: 各ブロックは、並行して個別に因子分解される。これらの係数は、補助空間  $af$  に検査不可能な方法で格納された、非零要素を生成する行列に適用される。数学上、これは行列  $A$  を  $PAP^T$  に順序変更し、各プロセッサで因子分解された行列のサイズの合計と同じサイズの主要な先頭部分行列を因子分解することに等しい。これらの部分行列の係数は、メモリ中の  $A$  に対応する部分を上書きする。

6. **縮退された連立方程式フェーズ**: より大きなブロックの相互作用を表す小さな ( $bw * (P-1)$ ) 連立方程式が形成され、(係数と同様に) 補助空間 *af* に格納される。並列ブロック・サイクリック分割が使用される。連立 1 次方程式に対して、並列前進解法とそれに続く後進解法が実行される。それぞれの解法は係数行列の構造を使用する。
7. **後退代入フェーズ**: 連立 1 次方程式に対して、各プロセッサでローカルの後退代入が並列に実行される。

## p?pttrsv

前進解法または後退解法によって、単一の三角行列線形問題を解く。ここで、三角行列は *p?pttrf* で計算された三重対角行列の係数である。

### 構文

```
call pspttrsv(uplo, n, nrhs, d, e, ja, desca, b, ib, descb, af, laf,
              work, lwork, info)
call pdpttrsv(uplo, n, nrhs, d, e, ja, desca, b, ib, descb, af, laf,
              work, lwork, info)
call pcpttrsv(uplo, trans, n, nrhs, d, e, ja, desca, b, ib, descb, af,
              laf, work, lwork, info)
call pzpttrsv(uplo, trans, n, nrhs, d, e, ja, desca, b, ib, descb, af,
              laf, work, lwork, info)
```

### 説明

このルーチンは、次の三重対角三角行列を係数行列とする連立 1 次方程式を解く。

$$A(1:n, ja:ja+n-1) * X = B(jb:jb+n-1, 1:nrhs)$$

または

$$A(1:n, ja:ja+n-1)^T * X = B(jb:jb+n-1, 1:nrhs) \text{ (実数型の場合)}$$

$$A(1:n, ja:ja+n-1)^H * X = B(jb:jb+n-1, 1:nrhs) \text{ (複素数型の場合)}$$

ここで、 $A(1:n, ja:ja+n-1)$  はコレスキー因子分解コード [p?pttrf](#) によって生成される三角帯行列であり、 $A(1:n, ja:ja+n-1)$  と *af* に格納される。 $A(1:n, ja:ja+n-1)$  には、*uplo* の値に従って、上三角行列または下三角行列が格納される。 $A(1:n, ja:ja+n-1)$  または  $A(1:n, ja:ja+n-1)^T$  (実数型の場合) と  $A(1:n, ja:ja+n-1)^H$  (複素数型の場合) のどの式を解くのかは、パラメータ *trans* を介して、ユーザによって決定される。

ルーチン [p?pttrf](#) は、最初に呼び出さなければならない。

## 入力パラメータ

<i>uplo</i>	(グローバル) CHARACTER。'U' または 'L' でなければならない。  <i>uplo</i> = 'U' の場合、 $A(1:n, ja:ja+n-1)$ の上三角行列が格納される。 <i>uplo</i> = 'L' の場合、 $A(1:n, ja:ja+n-1)$ の下三角行列が格納される。
<i>trans</i>	(グローバル) CHARACTER。'N' または 'C' でなければならない。  <i>trans</i> = 'N' の場合、 $A(1:n, ja:ja+n-1)$ を解く。 <i>trans</i> = 'C' (複素数型) の場合、 $\text{conjugate\_transpose}(A(1:n, ja:ja+n-1))$ を解く。
<i>n</i>	(グローバル) INTEGER。演算を行う行数と列数、すなわち、分散部分 行列 $A(1:n, ja:ja+n-1)$ の次数。 $n \geq 0$ 。
<i>nrhs</i>	(グローバル) INTEGER。右辺の数。分散部分行列 $B(jb:jb+n-1, 1:nrhs)$ の列数 ( $nrhs \geq 0$ )。
<i>d</i>	(ローカル)  REAL (pspttrsv の場合) DOUBLE PRECISION (pdpttrsv の場合) COMPLEX (pcpttrsv の場合) COMPLEX*16 (pzpttrsv の場合)。  行列の主対角を格納するグローバル・ベクトルのローカル部分へのポ インタ。サイズは、 $\geq \text{desca}(nb\_)$ でなければならない。
<i>e</i>	(ローカル)  REAL (pspttrsv の場合) DOUBLE PRECISION (pdpttrsv の場合) COMPLEX (pcpttrsv の場合) COMPLEX*16 (pzpttrsv の場合)。  行列の上対角を格納するグローバル・ベクトルのローカル部分へのポ インタ。サイズは、 $\geq \text{desca}(nb\_)$ でなければならない。全体的に、 $du(n)$ は参照されず、 $du$ は $d$ とアライメントされなければならない。
<i>ja</i>	(グローバル) INTEGER。(A のすべてまたは A の部分行列で) 処理され る行列の先頭を指すグローバル配列 A のインデックス。

<i>desca</i>	<p>(グローバルおよびローカル) INTEGER。配列、次元は (<i>dlen</i><sub>1</sub>)。分散行列 <i>A</i> の配列ディスクリプタ。  1D type (<i>dtype_a</i> = 501 または 502) の場合、<i>dlen</i> ≥ 7。  2D type (<i>dtype_a</i> = 1) の場合、<i>dlen</i> ≥ 9。  <i>A</i> のマッピング情報をメモリに格納する。詳細は、ScaLAPACK マニュアルを参照。</p>
<i>b</i>	<p>(ローカル)</p> <p>REAL (<i>pspttrsv</i> の場合)  DOUBLE PRECISION (<i>pdpttrsv</i> の場合)  COMPLEX (<i>pcpttrsv</i> の場合)  COMPLEX*16 (<i>pzpttrsv</i> の場合)。  ローカルメモリにある、ローカル・リーディング・ディメンジョン <i>lld_b</i> ≥ <i>nb</i> の配列へのポインタ。右辺 <i>B</i>(<i>jb</i>:<i>jb</i>+<i>n</i>-1, 1:<i>nrhs</i>) のローカル部分を格納する。</p>
<i>ib</i>	<p>(グローバル) INTEGER。(<i>B</i> のすべてまたは <i>B</i> の部分行列で) 処理される行列の最初の行を指すグローバル配列 <i>B</i> の行インデックス。</p>
<i>descb</i>	<p>(グローバルおよびローカル) INTEGER。配列、次元は (<i>dlen</i><sub>1</sub>)。分散行列 <i>B</i> の配列ディスクリプタ。  1D type (<i>dtype_a</i> = 502) の場合、<i>dlen</i> ≥ 7。  2D type (<i>dtype_b</i> = 1) の場合、<i>dlen</i> ≥ 9。  <i>B</i> のマッピング情報をメモリに格納する。詳細は、ScaLAPACK マニュアルを参照。</p>
<i>laf</i>	<p>(ローカル) INTEGER。ユーザ入力の補助非零要素空間 <i>af</i> のサイズ。  <i>laf</i> ≥ (<i>nb</i>+2*<i>bw</i>)*<i>bw</i> でなければならない。  <i>laf</i> が小さすぎる場合、エラーコードが返され、エラーにならない最小のサイズが <i>af</i>(1) に返される。</p>
<i>work</i>	<p>(ローカル)</p> <p>REAL (<i>pspttrsv</i> の場合)  DOUBLE PRECISION (<i>pdpttrsv</i> の場合)  COMPLEX (<i>pcpttrsv</i> の場合)  COMPLEX*16 (<i>pzpttrsv</i> の場合)。  配列 <i>work</i> は、次元 <i>lwork</i> の一次ワークスペース配列。このスペースは、ルーチンの呼び出しの間に上書きされる。</p>
<i>lwork</i>	<p>(ローカルまたはグローバル) INTEGER。ユーザー入力の <i>work</i> のサイズは、<i>lwork</i> ≥ (10+2*min(100, <i>nrhs</i>))*<i>npcol</i>+4*<i>nrhs</i> 以上でなければならない。<i>lwork</i> が小さすぎる場合、エラーコードが返され、エラーにならない最小のサイズが <i>work</i>(1) に返される。</p>



## 出力パラメータ

<i>d, e</i>	(ローカル) REAL (pspttrsv の場合) DOUBLE PRECISION (pdpttrsv の場合) COMPLEX (pcpttrsv の場合) COMPLEX*16 (pzpttrsv の場合)。 終了時に、行列の係数を含む情報が格納される。
<i>af</i>	(ローカル) REAL (pspttrsv の場合) DOUBLE PRECISION (pdpttrsv の場合) COMPLEX (pcpttrsv の場合) COMPLEX*16 (pzpttrsv の場合) 配列 <i>af</i> の次元は <i>laf</i> 。補助非零要素が格納される。非零要素は、因子分解ルーチン <a href="#">p?pbtrf</a> で作成され、 <i>af</i> に格納される。因子分解ルーチンの後で、 <a href="#">p?pttrs</a> を使って連立 1 次方程式を解く場合、因子分解後に <i>af</i> を変更してはならない。
<i>b</i>	終了時に、解の分散行列 <i>X</i> のローカル部分が格納される。
<i>work(1)</i>	終了時に、 <i>lwork</i> の最小値が <i>work(1)</i> に格納される。
<i>info</i>	(ローカル) INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> < 0 の場合、 <i>i</i> 番目の引数が配列で <i>j</i> 番目の成分の値が不正ならば <i>info</i> = -( <i>i</i> *100+ <i>j</i> )、 <i>i</i> 番目の引数がスカラで値が不正ならば <i>info</i> = - <i>i</i> 。

## p?potf2

対称/エルミート正定値行列のコレスキー因子分解を行う (ローカル非ブロック化アルゴリズム)。

### 構文

```
call pspotf2(uplo, n, a, ia, ja, desca, info)
call pdpotf2(uplo, n, a, ia, ja, desca, info)
call pcpotf2(uplo, n, a, ia, ja, desca, info)
call pzpotf2(uplo, n, a, ia, ja, desca, info)
```

## 説明

このルーチンは、実対称 / 複素エルミート正定値分散行列  
 $\text{sub}(A)=A(\text{ia}:\text{ia}+n-1, \text{ja}:\text{ja}+n-1)$  に対してコレスキー因子分解を行う。

因子分解は以下の形式を持つ。

$\text{uplo} = 'U'$  の場合、 $\text{sub}(A) = U' U$ 。

$\text{uplo} = 'L'$  の場合、 $\text{sub}(A) = L L'$ 。

ここで、 $U$  は上三角行列、 $L$  は下三角行列である。

## 入力パラメータ

$\text{uplo}$	(グローバル) CHARACTER。 対称 / エルミート行列 $A$ の上三角または下三角部分のどちらを格納するか指定する。 'U' の場合、 $\text{sub}(A)$ の上三角が格納される。 'L' の場合、 $\text{sub}(A)$ の下三角が格納される。
$n$	(グローバル) INTEGER。演算を行う行数と列数、すなわち、分散部分行列 $\text{sub}(A)$ の次数。 $n \geq 0$ 。
$a$	(ローカル) REAL (pspotf2 の場合) DOUBLE PRECISION (pdspotf2 の場合) COMPLEX (pcspotf2 の場合) COMPLEX*16 (pzspotf2 の場合)。 ローカルメモリにある、因子分解する $n \times n$ の対称分散行列 $\text{sub}(A)$ を格納する次元 ( $\text{lld\_a}$ , $\text{LOCc}(\text{ja}+n-1)$ ) の配列へのポインタ。 $\text{uplo} = 'U'$ の場合、 $\text{sub}(A)$ の先頭の $n \times n$ の上三角部分に、上三角行列を格納する。この行列の厳密な下三角部分は参照されない。 $\text{uplo} = 'L'$ の場合、 $\text{sub}(A)$ の先頭の $n \times n$ の下三角部分に、下三角行列を格納する。 $\text{sub}(A)$ の厳密な上三角部分は参照されない。
$\text{ia}, \text{ja}$	(グローバル) INTEGER。それぞれ、部分行列 $\text{sub}(A)$ の最初の行と最初の列を示す、グローバル配列 $A$ の行インデックスと列インデックス。
$\text{desca}$	(グローバルおよびローカル) INTEGER。配列、次元は ( $\text{dlen\_}$ )。分散行列 $A$ の配列ディスクリプタ。

## 出力パラメータ

<i>a</i>	(ローカル) 終了時に、 <i>uplo</i> = 'U' の場合、分散行列の上三角部分にコレスキー因子 <i>U</i> を格納する。 <i>uplo</i> = 'L' の場合、分散行列の下三角部分にコレスキー因子 <i>L</i> を格納する。
<i>info</i>	(ローカル) INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> < 0 の場合、 <i>i</i> 番目の引数が配列で <i>j</i> 番目の成分の値が不正ならば <i>info</i> = -( <i>i</i> *100+ <i>j</i> )、 <i>i</i> 番目の引数がスカラで値が不正ならば <i>info</i> = - <i>i</i> 。 <i>info</i> > 0 の場合、 <i>info</i> = <i>k</i> は次数 <i>k</i> の先頭の小行列式が正定値でないため因子分解を完了できなかったことを示す。

## p?rscl

ベクトルに実スカラの逆数を掛ける。

### 構文

```
call psrscl(n, sa, sx, ix, jx, descx, incx)
call pdrscl(n, sa, sx, ix, jx, descx, incx)
call pcsrscl(n, sa, sx, ix, jx, descx, incx)
call pzdrscl(n, sa, sx, ix, jx, descx, incx)
```

### 説明

このルーチンは、*n* 成分の実 / 複素ベクトル *sub(x)* に実スカラ  $1/a$  を掛ける。最終結果 *sub(x)/a* がオーバーフローまたはアンダアフローを起こさない限り、演算はオーバーフローまたはアンダアフローを起こすことなく実行される。

*sub(x)* は *x*(*ix*:*ix*+*n*-1, *jx*:*jx*) (*incx* = 1 の場合)、または  
*x*(*ix*:*ix*, *jx*:*jx*+*n*-1) (*incx* = *m\_x* の場合) である。

## 入力パラメータ

<i>n</i>	(グローバル) INTEGER。 乱数ベクトル <i>sub(x)</i> の成分の個数。( <i>n</i> ≥ 0)。
----------	--

<b>sa</b>	REAL (psrsc1/pcsrsc1 の場合 ) DOUBLE PRECISION (pdrsc1/pzdrsc1 の場合 )。 ベクトル $x$ の各成分の除算に使用されるスカラー $a$ 。この引数は $\geq 0$ でなければならない。そうでないと、サブルーチンでゼロ除算が発生する。
<b>sx</b>	REAL (psrsc1 の場合 ) DOUBLE PRECISION (pdrsc1 の場合 ) COMPLEX (pcsrsc1 の場合 ) COMPLEX*16 (pzdrsc1 の場合 )。 次元が $((jx-1)*m_x + ix + (n-1)*abs(incx))$ 以上のローカル部分を格納する配列。 乱数ベクトル sub ( $x$ ) の成分を格納する。
<b>ix</b>	( グローバル ) INTEGER。演算を行う分散行列 $X$ の部分行列の行インデックス。
<b>jx</b>	( グローバル ) INTEGER。演算を行う分散行列 $X$ の部分行列の列インデックス。
<b>descx</b>	( グローバルおよびローカル )。INTEGER。 次元 8 の配列。分散行列 $X$ の配列ディスクリプタ。
<b>incx</b>	( グローバル ) INTEGER。 $X$ の成分に対する増分。このバージョンでは、1 と $m_x$ の 2 つの $incx$ の値のみサポートされる。

### 出力パラメータ

<b>sx</b>	終了時に、結果 $x/a$ で上書きされる。
-----------	------------------------

---

## p?sygs2/p?hegs2

p?potrf で得られた因子分解の結果を用いて、対称/エルミート汎用固有値問題を標準形式に縮退させる ( ローカル非ブロック化アルゴリズム )。

---

### 構文

```
call pssygs2(ibtype, uplo, n, a, ia, ja, desca, b, ib, jb, descb, info)
call pdsygs2(ibtype, uplo, n, a, ia, ja, desca, b, ib, jb, descb, info)
```

```
call pchegs2(ibtype, uplo, n, a, ia, ja, desca, b, ib, jb, descb, info)
call pzhegs2(ibtype, uplo, n, a, ia, ja, desca, b, ib, jb, descb, info)
```

## 説明

ルーチン `p?sygs2/p?hegs2` は、実対称 / 複素エルミートの汎用固有値問題を標準化形式に縮退させる。

$\text{sub}(A)$  は  $A(ia:ia+n-1, ja:ja+n-1)$ 、 $\text{sub}(B)$  は  $B(ib:ib+n-1, jb:jb+n-1)$  である。

$ibtype = 1$  の場合、問題は次のとおりである。

$$\text{sub}(A)x = \lambda \text{sub}(B)x$$

$\text{sub}(A)$  は次のように上書きされる。

$\text{inv}(U^T) * \text{sub}(A) * \text{inv}(U)$  または  $\text{inv}(L) * \text{sub}(A) * \text{inv}(L^T)$  (実数型の場合)  
 $\text{inv}(U^H) * \text{sub}(A) * \text{inv}(U)$  または  $\text{inv}(L) * \text{sub}(A) * \text{inv}(L^H)$  (複素数型の場合)

$ibtype = 2$  または  $3$  の場合、問題は次のとおりである。

$$\text{sub}(A)\text{sub}(B)x = \lambda x \quad \text{または} \quad \text{sub}(B)\text{sub}(A)x = \lambda x$$

$\text{sub}(A)$  は次のように上書きされる。

$U * \text{sub}(A) * U^T$  または  $L * T * \text{sub}(A) * L$  (実数型の場合)  
 $U * \text{sub}(A) * U^H$  または  $L * H * \text{sub}(A) * L$  (複素数型の場合)

$\text{sub}(B)$  は  $U^T U$ 、 $LL^T$  (実数型の場合) または  $U^H U$ 、 $LL^H$  (複素数型の場合) として、[p?potrf](#) によって事前に因子分解されていないといけない。

## 入力パラメータ

**ibtype** (グローバル) INTEGER。  
**lbTYPE** = 1 の場合、 $\text{inv}(U^T) * \text{sub}(A) * \text{inv}(U)$  または  $\text{inv}(L) * \text{sub}(A) * \text{inv}(L^T)$  (実数サブルーチンの場合)、 $\text{inv}(U^H) * \text{sub}(A) * \text{inv}(U)$  または  $\text{inv}(L) * \text{sub}(A) * \text{inv}(L^H)$  (複素数サブルーチンの場合) を計算する。  
**lbTYPE** = 2 または 3 の場合、 $U * \text{sub}(A) * U^T$  または  $L^T * \text{sub}(A) * L$  (実数サブルーチンの場合)  $U * \text{sub}(A) * U^H$  または  $L^H * \text{sub}(A) * L$  (複素数サブルーチンの場合) を計算する。

**uplo** (グローバル) CHARACTER。  
 対称 / エルミート行列  $\text{sub}(A)$  の上三角部分または下三角部分のどちらが格納されているか、および  $\text{sub}(B)$  がどのように因子分解されるかを指定する。

	<p>'U' の場合、<math>\text{sub}(A)</math> の上三角を格納し、<math>\text{sub}(B)</math> を <math>U^T U</math> (実数サブルーチンの場合) または <math>U^H U</math> (複素数サブルーチン) として因数分解する。          'L' の場合、<math>\text{sub}(A)</math> の下三角を格納し、<math>\text{sub}(B)</math> を <math>L L^T</math> (実数サブルーチンの場合) または <math>L L^H</math> (複素数サブルーチンの場合)</p>
<i>n</i>	<p>(グローバル) INTEGER。          行列 <math>\text{sub}(A)</math> と <math>\text{sub}(B)</math> の次数。 <math>n \geq 0</math>。</p>
<i>a</i>	<p>(ローカル)          REAL (pssygs2 の場合)          DOUBLE PRECISION (pdsygs2 の場合)          COMPLEX (pchegs2 の場合)          COMPLEX*16 (pzhegs2 の場合)。          ローカルメモリにある、次元 (<math>lld\_a</math>, <math>LOCc(ja+n-1)</math>) の配列へのポインタ。  <math>n \times n</math> の対称 / エルミート行列 <math>\text{sub}(A)</math> のローカル部分を格納する。  <math>uplo = 'U'</math> の場合、<math>\text{sub}(A)</math> の先頭の <math>n \times n</math> の上三角部分に行列の上三角部分を格納する。<math>\text{sub}(A)</math> の厳密な下三角部分は参照されない。  <math>uplo = 'L'</math> の場合、<math>\text{sub}(A)</math> の先頭の <math>n \times n</math> の下三角部分に行列の下三角部分を格納する。<math>\text{sub}(A)</math> の厳密な上三角部分は参照されない。</p>
<i>ia, ja</i>	<p>(グローバル) INTEGER。それぞれ、部分行列 <math>\text{sub}(A)</math> の最初の行と最初の列を示す、グローバル配列 <i>A</i> の行インデックスと列インデックス。</p>
<i>desca</i>	<p>(グローバルおよびローカル) INTEGER。配列、次元は (<math>dlen\_</math>)。分散行列 <i>A</i> の配列ディスクリプタ。</p>
<i>b</i>	<p>(ローカル)          REAL (pssygs2 の場合)          DOUBLE PRECISION (pdsygs2 の場合)          COMPLEX (pchegs2 の場合)          COMPLEX*16 (pzhegs2 の場合)。          ローカルメモリにある、次元 (<math>lld\_b</math>, <math>LOCc(jb+n-1)</math>) の配列へのポインタ。  <a href="#">p?potrf</a> によって返された、<math>\text{sub}(B)</math> のコレスキー因子分解で得られた三角係数を格納する。</p>
<i>ib, jb</i>	<p>(グローバル) INTEGER。それぞれ、<math>\text{sub}(B)</math> の最初の行と最初の列を示す、グローバル配列 <i>B</i> の行インデックスと列インデックス。</p>
<i>descb</i>	<p>(グローバルおよびローカル) INTEGER。配列、次元は (<math>dlen\_</math>)。分散行列 <i>B</i> の配列ディスクリプタ。</p>

### 出力パラメータ

<i>a</i>	(ローカル)終了時に、 <i>info</i> = 0 の場合、変換後の行列が <i>sub</i> ( <i>A</i> ) と同じ形式で格納される。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> < 0 の場合、 <i>i</i> 番目の引数が配列で <i>j</i> 番目の成分の値が不正ならば <i>info</i> = -( <i>i</i> *100)、 <i>i</i> 番目の引数がスカラーで値が不正ならば <i>info</i> = - <i>i</i> 。

## p?sytd2/p?hetd2

直交/ユニタリ相似変換を用いて、対称/エルミート行列を実数対称三重対角形式に縮退させる (ローカル非ブロック化アルゴリズム)。

### 構文

```
call pssytd2(uplo, n, a, ia, ja, desca, d, e, tau, work, lwork, info)
call pdsytd2(uplo, n, a, ia, ja, desca, d, e, tau, work, lwork, info)
call pchetd2(uplo, n, a, ia, ja, desca, d, e, tau, work, lwork, info)
call pzhetd2(uplo, n, a, ia, ja, desca, d, e, tau, work, lwork, info)
```

### 説明

ルーチン p?sytd2/p?hetd2 は、直交/ユニタリ相似変換  $Q' \text{sub}(A) Q = T$  を用いて、実対称/複素エルミート行列 *sub*(*A*) を対称/エルミート三重対角形式 *T* に縮退させる。ここで、*sub*(*A*) = *A*(*ia*:*ia*+*n*-1, *ja*:*ja*+*n*-1) である。

### 入力パラメータ

<i>uplo</i>	(グローバル) CHARACTER。 対称/エルミート行列 <i>sub</i> ( <i>A</i> ) の上三角または下三角部分のどちらを格納するか指定する。 'U' の場合、上三角部分。 'L' の場合、下三角部分。
<i>n</i>	(グローバル) INTEGER。 演算を行う行数と列数、すなわち、分散部分行列 <i>sub</i> ( <i>A</i> ) の次数。 $n \geq 0$ 。

<i>a</i>	(ローカル) REAL (pssytd2 の場合) DOUBLE PRECISION (pdsytd2 の場合) COMPLEX (pchetd2 の場合) COMPLEX*16 (pzhetd2 の場合)。 ローカルメモリにある、次元 ( <i>lld_a</i> , <i>LOCc(ja+n-1)</i> ) の配列へのポインタ。 <i>n</i> × <i>n</i> の対称 / エルミート行列 <i>sub(A)</i> のローカル部分を格納する。 <i>uplo</i> = 'U' の場合、 <i>sub(A)</i> の先頭の <i>n</i> × <i>n</i> の上三角部分に行列の上三角部分を格納する。 <i>sub(A)</i> の厳密な下三角部分は参照されない。 <i>uplo</i> = 'L' の場合、 <i>sub(A)</i> の先頭の <i>n</i> × <i>n</i> の下三角部分に行列の下三角部分を格納する。 <i>sub(A)</i> の厳密な上三角部分は参照されない。
<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 <i>sub(A)</i> の最初の行と最初の列を示す、グローバル配列 <i>A</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。分散行列 <i>A</i> の配列ディスクリプタ。
<i>work</i>	(ローカル) REAL (pssytd2 の場合) DOUBLE PRECISION (pdsytd2 の場合) COMPLEX (pchetd2 の場合) COMPLEX*16 (pzhetd2 の場合)。 配列 <i>work</i> は、次元 <i>lwork</i> の一次ワークスペース配列。

## 出力パラメータ

<i>a</i>	終了時に、 <i>uplo</i> = 'U' の場合、 <i>sub(A)</i> の対角成分と最初の優対角成分は、三重対角行列 <i>T</i> の対応する成分で上書きされる。最初の優対角成分より上の成分は、配列 <i>tau</i> とともに、基本リフレクタの積として直交 / ユニタリ行列 <i>Q</i> を表現する。 <i>uplo</i> = 'L' の場合、 <i>a</i> の対角成分と最初の劣対角成分は、三重対角行列 <i>T</i> の対応する成分で上書きされる。最初の劣対角成分より下の成分は、配列 <i>tau</i> とともに、基本リフレクタの積として直交 / ユニタリ行列 <i>Q</i> を表現する。次の「アプリケーション・ノート」を参照。
<i>d</i>	(ローカル) REAL (pssytd2/pchetd2 の場合) DOUBLE PRECISION (pdsytd2/pzhetd2 の場合) 配列、次元は ( <i>LOCc(ja+n-1)</i> )。



	<p>三重対角行列 <math>T</math> の対角成分。  <math>d(i) = a(i,i)</math>  <math>d</math> は、分散行列 <math>A</math> に関連付けられる。</p>
$e$	<p>(ローカル)  REAL (pssytd2/pchetd2 の場合)  DOUBLE PRECISION (pdsytd2/pzhetd2 の場合)。  配列、次元は <math>(LOCc(ja+n-1))</math> (<math>uplo = 'U'</math> の場合) または  <math>LOCc(ja+n-2)</math> (それ以外の場合)。  三重対角行列 <math>T</math> の非対角成分。  <math>uplo = 'U'</math> の場合、<math>e(i) = a(i,i+1)</math>。  <math>uplo = 'L'</math> の場合、<math>e(i) = a(i+1,i)</math>。  <math>e</math> は分散行列 <math>A</math> に関連付けられる。</p>
$\tau$	<p>(ローカル)  REAL (pssytd2 の場合)  DOUBLE PRECISION (pdsytd2 の場合)  COMPLEX (pchetd2 の場合)  COMPLEX*16 (pzhetd2 の場合)。  配列、次元は <math>(LOCc(ja+n-1))</math>。  基本リフレクタのスカラ係数。  <math>\tau</math> は、分散行列 <math>A</math> に関連付けられる。</p>
$work(1)$	<p>終了時に、<math>work(1)</math> は、最小かつ最適な <math>lwork</math> 値を返す。</p>
$lwork$	<p>(ローカルまたはグローバル) INTEGER。  ワークスペース配列 <math>work</math> の次元。  <math>lwork</math> はローカル入力であり、<math>lwork \geq 3n</math> でなければならない。   <math>lwork = -1</math> の場合、<math>lwork</math> はグローバル入力であり、ワークスペースのクエリとみなされ、ルーチンはすべての配列の最小かつ最適なサイズだけを計算する。各値は該当する <math>work</math> 配列の最初のエントリとして返され、<a href="#">pxerbla</a> はエラー・メッセージを生成しない。</p>
$info$	<p>(ローカル) INTEGER。  <math>info = 0</math> の場合、正常に終了したことを示す。  <math>info &lt; 0</math> の場合、<math>i</math> 番目の引数が配列で <math>j</math> 番目の成分の値が不正ならば <math>info = -(i*100)</math>、<math>i</math> 番目の引数がスカラで値が不正ならば <math>info = -i</math>。</p>

## アプリケーション・ノート

$uplo = 'U'$  の場合、行列  $Q$  は基本リフレクタの積として表現される。

$$Q = H(n-1) \dots H(2) H(1)$$

それぞれの  $H(i)$  は次のような形式を持つ。

$$H(i) = I - \tau v v'$$

ここで、 $\tau$  は実 / 複素スカラ、 $v$  は実 / 複素ベクトルで  $v(i+1:n) = 0$  および  $v(i) = 1$ 。終了時に、 $v(1:i-1)$  は  $A(ia:ia+i-2, ja+i)$  に、 $\tau$  は  $TAU(ja+i-1)$  に格納される。

$uplo = 'U'$  の場合、行列  $Q$  は基本リフレクタの積として表現される。

$$Q = H(1) H(2) \dots H(n-1)$$

それぞれの  $H(i)$  は次のような形式を持つ。

$$H(i) = I - \tau v v'$$

$\tau$  は実 / 複素スカラ、 $v$  は実 / 複素ベクトルで  $v(1:i) = 0$  および  $v(i+1) = 1$ 。終了時に、 $v(i+2:n)$  は  $A(ia+i+1:ia+n-1, ja+i-1)$  に、 $\tau$  は  $TAU(ja+i-1)$  に格納される。

終了時の  $sub(A)$  の内容を次に示す ( $n=5$  の場合)。

$uplo = 'U'$  の場合

$uplo = 'L'$  の場合

$$\begin{bmatrix} d & e & v_2 & v_3 & v_4 \\ & d & e & v_3 & v_4 \\ & & d & e & v_4 \\ & & & d & e \\ & & & & d \end{bmatrix} \qquad \begin{bmatrix} d & & & & \\ e & d & & & \\ v_1 & e & d & & \\ v_1 & v_2 & e & d & \\ v_1 & v_2 & v_3 & e & d \end{bmatrix}$$

ここで、 $d$  と  $e$  は  $T$  の対角成分と非対角成分である。 $v_i$  は  $H(i)$  を定義するベクトルの成分を表す。



**注:** 分散部分行列  $\text{sub}(A)$  次の式を満たすようにアライメントされていなければならない。

$(mb\_a.eq.nb\_a \ . \ \text{AND} \ . \ iroffa.eq.icoffa)$

ここで、 $iroffa = \text{mod}(ia - 1, mb\_a)$ 、 $coffa = \text{mod}(ja - 1, nb\_a)$  である。

## p?trti2

三角行列の逆行列を計算する ( ローカル非ブロック化アルゴリズム )。

### 構文

```
call pstrti2(uplo, diag, n, a, ia, ja, desca, info)
call pdtrti2(uplo, diag, n, a, ia, ja, desca, info)
call pctrti2(uplo, diag, n, a, ia, ja, desca, info)
call pztrti2(uplo, diag, n, a, ia, ja, desca, info)
```

### 説明

このルーチンは、実 / 複素上三角 / 下三角ブロック行列  $\text{sub}(A) = A(ia:ia+n-1, ja:ja+n-1)$  の逆行列を計算する。

この行列は、1 つのプロセスメモリにのみ格納されなければならない ( ローカル演算 )。

### 入力パラメータ

**uplo** ( グローバル ) CHARACTER\*1。  
行列  $\text{sub}(A)$  が上三角か下三角かを指定する。  
'U' の場合、 $\text{sub}(A)$  は上三角である。  
'L' の場合、 $\text{sub}(A)$  は下三角である。

**diag** ( グローバル ) CHARACTER\*1。  
行列  $A$  が単位三角かどうかを指定する。  
'N' の場合、 $\text{sub}(A)$  は単位三角ではない。  
'U' の場合、 $\text{sub}(A)$  は単位三角である。

<i>n</i>	(グローバル) INTEGER。 演算を行う行数と列数、すなわち、分散部分行列 <i>sub(A)</i> の次数。 $n \geq 0$ 。
<i>a</i>	(ローカル) REAL ( <i>pstrti2</i> の場合) DOUBLE PRECISION ( <i>pdttrti2</i> の場合) COMPLEX ( <i>pctrti2</i> の場合) COMPLEX*16 ( <i>pztrti2</i> の場合)。 ローカルメモリにある、次元 ( <i>lld_a</i> , <i>LOCc(ja+n-1)</i> ) の配列へのポインタ。 三角行列 <i>sub(A)</i> のローカル部分を格納する。 <i>uplo</i> = 'U' の場合、行列 <i>sub(A)</i> の先頭の $n \times n$ の上三角部分に行列の上三角部分を格納する。 <i>sub(A)</i> の厳密な下三角部分は参照されない。 <i>uplo</i> = 'L' の場合、行列 <i>sub(A)</i> の先頭の $n \times n$ の下三角部分に行列の下三角部分を格納する。 <i>sub(A)</i> の厳密な上三角部分は参照されない。 <i>diag</i> = 'U' の場合、 <i>sub(A)</i> の対角成分は参照されず 1 とみなされる。
<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 <i>sub(A)</i> の最初の行と最初の列を示す、グローバル配列 <i>A</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。配列、次元は ( <i>dlen_</i> )。分散行列 <i>A</i> の配列ディスクリプタ。

## 出力パラメータ

<i>a</i>	終了時に、元の行列の (三角) 逆行列で、同じ格納形式で上書きされる。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> < 0 の場合、 <i>i</i> 番目の引数が配列で <i>j</i> 番目の成分の値が不正ならば <i>info</i> = -( <i>i</i> *100)、 <i>i</i> 番目の引数がスカラで値が不正ならば <i>info</i> = - <i>i</i> 。

## ?lamsh

送られるバルジの数を最大にするために、小さな  
(単一成分) 行列を介して複数のシフトを送る。

### 構文

```
call slamsh(s, lds, nbulge, jblk, h, ldh, n, ulp)
call dlamsh(s, lds, nbulge, jblk, h, ldh, n, ulp)
```

### 説明

このルーチンは、小さな (単一成分) 行列を介して複数のシフトを送り、送られるバルジの数が最大となるよう、小さな連続した対角成分が、続くシフトによってどのように変形されるかを検証する。複数のシフト / バルジ ( $nbulge > 1$ ) があり、2 つ以上の小さな連続した対角成分のために、最初のシフトが縮退されていない Hessenberg 行列の途中から開始される場合のみ、サブルーチン呼び出すべきである。

### 入力パラメータ

<i>s</i>	(ローカル) INTEGER。 REAL (slamsh の場合) DOUBLE PRECISION (dlamsh の場合)。  配列、次元は ( $lds, *$ )。 シフト行列。 <i>s</i> の $2 \times 2$ 対角のみ参照される。 <i>s</i> は <i>jblk</i> 倍精度シフト (サイズ 2) とする。
<i>lds</i>	(ローカル) INTEGER。 <i>S</i> のリーディング・ディメンジョン。終了時に変更されない。 $1 < nbulge \leq jblk \leq lds/2$ 。
<i>nbulge</i>	(ローカル) INTEGER。 <i>h</i> ( $> 1$ ) を介して送るバルジの数。 <i>nbulge</i> は、決定された ( <i>jblk</i> ) の最大値以下でなければならない。 $1 < nbulge \leq jblk \leq lds/2$ 。
<i>jblk</i>	(ローカル) INTEGER。 <i>S</i> のリーディング・ディメンジョン。終了時に変更されない。
<i>h</i>	(ローカル) INTEGER。 REAL (slamsh の場合) DOUBLE PRECISION (dlamsh の場合)。

配列、次元は  $(lds, n)$ 。  
シフトを適用するローカル行列。  
 $h$  は、開始行が 2 となるようにアライメントされていなければならない。

$ldh$  (ローカル) INTEGER。  
 $H$  のリーディング・ディメンジョン。終了時に変更されない。

$n$  (ローカル) INTEGER。  
 $H$  のサイズ。すべてのバルジが送られると見込まれる場合、 $n$  は  $4nbulge+2$  以上でなければならない。それ以外の場合、 $nbulge$  はこのルーチンによって縮小されることがある。

$ulp$  (ローカル)  
REAL (slamsh の場合)  
DOUBLE PRECISION (dlamsh の場合)。  
マシン精度。終了時に変更されない。

## 出力パラメータ

$s$  終了時に、データは、適用するのに最良な順序に再配置される。

$nbulge$  終了時に、送信可能なバルジの最大値。

$h$  終了時に、データは破棄される。

---

## ?laref

*Householder* リフレクタを行列の行または列に適用する。

---

## 構文

```
call slaref(type, a, lda, wantz, z, ldz, block, irow1, icol1, istart, istop,  
           itmp1, itm2, lilo, lihiz, vecs, v2, v3, t1, t2, t3)  
call dlaref(type, a, lda, wantz, z, ldz, block, irow1, icol1, istart, istop,  
           itmp1, itm2, lilo, lihiz, vecs, v2, v3, t1, t2, t3)
```

## 説明

このルーチンは、1 つまたは複数のサイズ 3 の *Householder* リフレクタ を 1 つまたは 2 つの行列 (列が指定された場合) の行または列に適用する。

## 入力パラメータ

<i>type</i>	(グローバル) CHARACTER*1。 <i>type</i> = 'R' の場合、リフレクタを行列の行に左から適用する。 それ以外の場合、リフレクタを行列の列に適用する。終了時に変更されない。
<i>a</i>	(グローバル) REAL( <i>slaref</i> の場合) DOUBLE PRECISION( <i>dlaref</i> の場合)。 配列、次元は ( <i>lda</i> , *)。リフレクタを受け取る行列。
<i>lda</i>	(ローカル) INTEGER。 <i>A</i> のリーディング・ディメンジョン。終了時に変更されない。
<i>wantz</i>	(グローバル) LOGICAL。 <i>wantz</i> = .TRUE. の場合、任意の列リフレクタを <i>Z</i> にも適用する。 <i>wantz</i> = .FALSE. の場合、 <i>Z</i> はそのまま。
<i>z</i>	(グローバル) REAL( <i>slaref</i> の場合) DOUBLE PRECISION( <i>dlaref</i> の場合)。 配列、次元は ( <i>ldz</i> , *)。列リフレクタを受け取る 2 番目の行列。
<i>ldz</i>	(ローカル) INTEGER。 <i>Z</i> のリーディング・ディメンジョン。終了時に変更されない。
<i>block</i>	(グローバル) LOGICAL。 .TRUE. の場合、直ちに複数のリフレクタを適用し、データを配列 <i>vecs</i> から読み込む。 .FALSE. の場合、 <i>v2</i> , <i>v3</i> , <i>t1</i> , <i>t2</i> , および <i>t3</i> で与えられる 1 つのリフレクタを適用する。
<i>ipow1</i>	(ローカル) INTEGER。 行列 <i>A</i> のローカル行成分。
<i>icol1</i>	(ローカル) INTEGER。 行列 <i>A</i> のローカル列成分。
<i>istart</i>	(グローバル) INTEGER。 最初のリフレクタの " 番号 " を指定する。 <i>block</i> が設定されている場合、 <i>istart</i> は <i>vecs</i> へのインデックスとして使用される。 <i>block</i> が .FALSE. の場合、 <i>istart</i> は無視される。

<i>istop</i>	(グローバル) INTEGER。 最後のリフレクタの " 番号 " を指定する。 <i>block</i> が設定されている場合、 <i>istop</i> は <i>vecs</i> へのインデックスとして使用される。 <i>block</i> が .FALSE. の場合、 <i>istop</i> は無視される。
<i>itmp1</i>	(ローカル) INTEGER。 <i>A</i> への開始範囲。行に対する最初のローカル列。列に対する最初のローカル行。
<i>itmp2</i>	(ローカル) INTEGER。 <i>A</i> への終了範囲。行に対する最後のローカル列。列に対する最後のローカル行。
<i>liloz,lihiz</i>	(ローカル) INTEGER。 <i>itmp1</i> 、 <i>itmp2</i> と同じ働きをする。ただし、 <i>Z</i> に対しては <i>wantz</i> が設定されている場合のみ。
<i>vecs</i>	(グローバル) REAL ( <i>slaref</i> の場合) DOUBLE PRECISION ( <i>dlaref</i> の場合)。 サイズ 3*n (行列のサイズ) の配列。サイズ 3 のリフレクタを次々と格納する。 <i>block</i> が .TRUE. の場合のみアクセスされる。
<i>v2,v3,t1,t2,t3</i>	(グローバル) INTEGER。 REAL ( <i>slaref</i> の場合) DOUBLE PRECISION ( <i>dlaref</i> の場合)。 これらのパラメータは、1 つのサイズ 3 の <b>Householder</b> リフレクタの情報を格納する。 <i>block</i> が .FALSE. の場合には読み込まれ、 <i>block</i> が .TRUE. の場合には上書きされる。

### 出力パラメータ

<i>a</i>	終了時に、更新された行列で上書きされる。
<i>z</i>	<i>wantz</i> が設定されている場合のみ変更される。 <i>wantz</i> が .FALSE. の場合、 <i>z</i> は参照されない。
<i>ipow1</i>	定義されていない。
<i>icol1</i>	定義されていない。
<i>v2,v3,t1,t2,t3</i>	これらのパラメータは、 <i>block</i> が .FALSE. の場合には読み込まれ、 <i>block</i> が .TRUE. の場合には上書きされる。



## ?lasorte

固有ペアを実数および複素数データ型でソートする。

### 構文

```
call slasorte(s, lds, j, out, info)
call dlasorte(s, lds, j, out, info)
```

### 説明

このルーチンは、固有ペアが同じデータ型 (実数または複素数) でまとまるようにソートする。これにより、第2劣対角成分が0であることが保証され、 $2 \times 2$  シフトを容易に使用することができるようになる。このルーチンは、並列では動作しない。また、呼び出しも行わない。

### 入力パラメータ

<i>s</i>	(ローカル) INTEGER。 REAL (slasorte の場合) DOUBLE PRECISION (dlasorte の場合)。  配列、次元は ( <i>lds</i> )。 行列はすでに <b>Schur</b> 形式である。
<i>lds</i>	(ローカル) INTEGER。 配列 <i>s</i> のリーディング・ディメンジョン。終了時に変更されない。
<i>j</i>	(ローカル) INTEGER。 行列 <i>S</i> の次数。終了時に変更されない。
<i>out</i>	(ローカル) INTEGER。 REAL (slasorte の場合) DOUBLE PRECISION (dlasorte の場合)。  配列、次元は ( <i>j</i> ×2)。 ルーチンに必要なワークバッファ。
<i>info</i>	(ローカル) INTEGER。 入力行列の実数固有値の数が奇数でペアにできない場合、または入力行列 <i>S</i> がもともと <b>Schur</b> 形式でない場合に設定する。 0 は正常に終了したことを示す。

## 出力パラメータ

<i>s</i>	終了時に、 <i>S</i> の対角ブロックは、固有値をペアにするために書き直される。結果として得られる行列は、入力とは似ていない。
<i>out</i>	ワークバッファ。

---

## ?lasrt2

数を昇順または降順でソートする。

---

### 構文

```
call slasrt2(id, n, d, key, info)
call dlasrt2(id, n, d, key, info)
```

### 説明

このルーチンは、*d* に格納されている数を昇順 (*id* = 'I' の場合) または降順 (*ID* = 'D' の場合) でソートする、LAPACK ルーチン **?lasrt** の修正版である。このルーチンはクイックソートを使用するが、*size* ≤ 20 の配列では挿入ソートが使われる。スタックの次元によって *n* はおよそ  $2^{32}$  に制限される。

## 入力パラメータ

<i>id</i>	CHARACTER*1。 'I' の場合、 <i>d</i> を昇順でソートする。 'D' の場合、 <i>d</i> を降順でソートする。
<i>n</i>	INTEGER。配列 <i>d</i> の長さ。
<i>d</i>	REAL (slasrt2 の場合) DOUBLE PRECISION (dlasrt2 の場合)。 配列、次元は ( <i>n</i> )。 ソートする配列を格納する。
<i>key</i>	INTEGER。 配列、次元は ( <i>n</i> )。 <i>key</i> には <i>d</i> () の各成分のキーを格納する。 通常は、すべての <i>i</i> に対して <i>key</i> ( <i>i</i> ) = <i>i</i> 。

### 出力パラメータ

<i>d</i>	終了時に、 <i>id</i> の設定によって、 <i>d</i> は昇順 ( $d(1) \leq \dots \leq d(n)$ ) または降順 ( $d(1) \geq \dots \geq d(n)$ ) となる。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> < 0 の場合、 <i>info</i> = - <i>i</i> は <i>i</i> 番目の引数の値が不正だったことを示す。
<i>key</i>	終了時に、 <i>d</i> () が入力から出力に置換されたのと同様に、 <i>key</i> は置換される。したがって、入力時にすべての <i>i</i> に対して <i>key</i> ( <i>i</i> ) = <i>i</i> の場合、 <i>*d_out</i> ( <i>i</i> ) = <i>d_in</i> ( <i>key</i> ( <i>i</i> )) である。

## ?stein2

逆反復法を用いて、実対称の三重対角行列の指定された固有値に対応する固有ベクトルを計算する。

### 構文

```
call sstein2(n, d, e, m, w, iblock, isplit, orfac, z, ldz,
            work, iwork, ifail, info)
call dstein2(n, d, e, m, w, iblock, isplit, orfac, z, ldz,
            work, iwork, ifail, info)
```

### 説明

このルーチンは、LAPACK ルーチン [?stein](#) の修正版である。このルーチンでは、実対称の三重対角行列 *T* の指定された固有値に対応する固有ベクトルを、反転 (逆行列の計算) を繰り返して求める。

固有ベクトルごとの最大反復回数は、内部パラメータ *maxits* で指定する (現時点では、5 に設定されている)。

### 入力パラメータ

<i>n</i>	INTEGER。行列 <i>T</i> の次数 ( $n \geq 0$ )。
<i>m</i>	INTEGER。見つける固有値の数 ( $0 \leq m \leq n$ )。

<i>d, e, w</i>	<p>REAL ( 単精度の場合 )  DOUBLE PRECISION ( 倍精度の場合 )。  配列：  <i>d</i>(*)、次元は (<i>n</i>)。  三重対角行列 <i>T</i> の <i>n</i> 個の対角成分。    <i>e</i>(*)、次元は (<i>n</i>)。  三重対角行列 <i>T</i> の (<i>n</i>-1) 個の劣対角成分を成分 1 から <i>n</i>-1 に格納する。  <i>e</i>(<i>n</i>) を設定する必要はない。    <i>w</i>(*)、次元は (<i>n</i>)。  <i>w</i> の最初の <i>m</i> 個の成分には固有ベクトルを計算する固有値を格納する。  固有値は分割されたブロックごとにグループ分けし、ブロック内で最小から最大に並べられていなければならない (ORDER = 'B' である <a href="#">?stebz</a> の出力配列 <i>w</i> がここで要求される)。    <i>w</i> の次元は、max(1, <i>n</i>) 以上でなければならない。</p>
<i>iblock</i>	<p>INTEGER。  配列、次元は (<i>n</i>)。  <i>w</i> 内の対応する固有値に関連した部分行列インデックス。固有値 <i>w</i>(<i>i</i>) が先頭から 1 番目の部分行列の属する場合、<i>iblock</i>(<i>i</i>) = 1。  固有値 <i>w</i>(<i>i</i>) が 2 番目の部分行列に属する場合、<i>iblock</i>(<i>i</i>) = 2。以下同様 (?<i>stebz</i> の出力配列 <i>iblock</i> がここで要求される)。</p>
<i>isplit</i>	<p>INTEGER。  配列、次元は (<i>n</i>)。  <i>T</i> を部分行列に分割した分割点。第 1 の部分行列は 1 から <i>isplit</i>(1) の行 / 列で構成され、第 2 の部分行列は <i>isplit</i>(1)+1 から <i>isplit</i>(2) の行 / 列で構成され、以下同様 (?<i>stebz</i> の出力配列 <i>isplit</i> がここで要求される)。</p>
<i>orfac</i>	<p>REAL ( 単精度の場合 )  DOUBLE PRECISION ( 倍精度の場合 )  <i>orfac</i> は、直交化される固有ベクトルを指定する。互いの <i>orfac</i>*   <i>T</i>    内にある固有値に対応する固有ベクトルは直交される。</p>
<i>ldz</i>	<p>INTEGER。出力配列 <i>z</i> のリーディング・ディメンジョン。  <math>ldz \geq \max(1, n)</math>。</p>
<i>work</i>	<p>REAL ( 単精度の場合 )  DOUBLE PRECISION ( 倍精度の場合 )。  ワークスペース配列、次元は (5<i>n</i>)。</p>

*iwork* INTEGER。  
ワークスペース配列、次元は  $(n)$ 。

### 出力パラメータ

*z* REAL (sstein2 の場合)  
DOUBLE PRECISION (dstein2 の場合)  
配列、次元は  $(ldz, m)$ 。  
計算された固有ベクトル。固有値  $w(i)$  に対応する固有ベクトルが *z* の *i* 番目の列に格納される。収束しなかったすべてのベクトルは、*maxits* 回反復した後、現在の反復に設定される。

*ifail* INTEGER。配列、次元は  $(m)$ 。  
通常終了時は、*ifail* のすべての成分はゼロになる。1 つまたは複数の固有ベクトルが *maxits* 回反復した後で収束に失敗した場合、その固有ベクトルのインデックスが配列 *ifail* に格納される。

*info* INTEGER。  
*info* = 0 の場合、正常に終了したことを示す。  
*info* < 0 の場合、*info* = -*i* は *i* 番目の値が不正だったことを示す。  
*info* > 0 の場合、*info* = *i* は *i* 番目の固有ベクトルが *maxits* 回目の反復で収束に失敗したことを示す。その固有ベクトルのインデックスが配列 *ifail* に格納される。

---

## ?dbtf2

一般帯行列の LU 因子分解を、ピボット演算を用いないで計算する (ローカル非ブロック化アルゴリズム)。

---

### 構文

```
call sdbtf2(m, n, kl, ku, ab, ldab, info)
call ddbtf2(m, n, kl, ku, ab, ldab, info)
call cdbtf2(m, n, kl, ku, ab, ldab, info)
call zdbtf2(m, n, kl, ku, ab, ldab, info)
```

## 説明

このルーチンは、 $m \times n$  の一般実 / 複素帯行列  $A$  の  $LU$  因子分解を、行交換を伴う部分ピポット演算を用いないで計算する。

このルーチンは、アルゴリズムの非ブロック化形式で、[BLAS のルーチンと関数](#) を呼び出す。

## 入力パラメータ

<i>m</i>	INTEGER。行列 $A$ の行数 ( $m \geq 0$ )。
<i>n</i>	INTEGER。 $A$ の列数 ( $n \geq 0$ )。
<i>kl</i>	INTEGER。 $A$ の帯内の劣対角成分の数 ( $kl \geq 0$ )。
<i>ku</i>	INTEGER。 $A$ の帯内の優対角成分の数 ( $ku \geq 0$ )。
<i>ab</i>	REAL (sdbtf2 の場合 ) DOUBLE PRECISION (ddbtf2 の場合 ) COMPLEX (cdbtf2 の場合 ) COMPLEX*16 (zdbtf2 の場合 )。 配列、次元は ( <i>ldab</i> , <i>n</i> )。 行列 $A$ を帯格納形式で行 $kl+1$ から $2kl+ku+1$ に格納する。配列の行 1 から $kl$ を設定する必要はない。 $A$ の $j$ 番目の列は配列 <i>ab</i> の $j$ 番目の列に次のように格納する。 $\max(1, j-ku) \leq i \leq \min(m, j+kl)$ に対して $ab(kl+ku+1+i-j, j) = A(i, j)$ 。
<i>ldab</i>	INTEGER。 <i>ab</i> のリーディング・ディメンジョン。 ( $ldab \geq 2kl + ku + 1$ )

## 出力パラメータ

<i>ab</i>	終了時に、因子分解の詳細で上書きされる。 $U$ は、 $kl+ku$ 個の優対角成分を持つ上三角帯行列として、行 1 から $kl+ku+1$ に格納される。因子分解に使用した乗数は、行 $kl+ku+2$ から $2*kl+ku+1$ に格納される。詳細は、次の「アプリケーション・ノート」を参照。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> < 0 の場合、 <i>info</i> = - <i>i</i> は <i>i</i> 番目の引数の値が不正だったことを示す。 <i>info</i> > 0 の場合、 <i>info</i> = + <i>i</i> は $u(i, i)$ が 0 であることを示す。因子分解は完了したが、係数 $U$ は完全に特異である。連立 1 次方程式の解の算出に係数 $U$ を使用すると、ゼロ除算が発生する。

## アプリケーション・ノート

帯格納方式の例を次に示す ( $m = n = 6$ 、 $kl = 2$ 、 $ku = 1$  の場合)。

入力

$$\begin{bmatrix} * & a_{12} & a_{23} & a_{34} & a_{45} & a_{56} \\ a_{11} & a_{22} & a_{33} & a_{44} & a_{55} & a_{66} \\ a_{21} & a_{32} & a_{43} & a_{54} & a_{65} & * \\ a_{31} & a_{42} & a_{53} & a_{64} & * & * \end{bmatrix}$$

出力

$$\begin{bmatrix} * & u_{12} & u_{23} & u_{34} & u_{45} & u_{56} \\ u_{11} & u_{22} & u_{33} & u_{44} & u_{55} & u_{66} \\ m_{21} & m_{32} & m_{43} & m_{54} & m_{65} & * \\ m_{31} & m_{42} & m_{53} & m_{64} & * & * \end{bmatrix}$$

\* でマークされた配列の成分はルーチンで使用されない。+ でマークされた成分は入力時に設定する必要はないが、行交換の結果として生じる非零要素のために、 $U$  の成分を格納するのにルーチンが必要となる。

## ?dbtrf

一般帯行列の  $LU$  因子分解を、ピボット演算を用いないで計算する (ローカルブロック化アルゴリズム)。

### 構文

```
call sdbtrf(m, n, kl, ku, ab, ldab, info)
call ddbtrf(m, n, kl, ku, ab, ldab, info)
call cdbtrf(m, n, kl, ku, ab, ldab, info)
call zdbtrf(m, n, kl, ku, ab, ldab, info)
```

### 説明

このルーチンは、 $m \times n$  の一般実 / 複素帯行列  $A$  の  $LU$  因子分解を、行交換または部分ピボット演算を用いないで計算する。

このルーチンは、アルゴリズムのブロック化形式で、[BLAS のルーチンと関数](#) を呼び出す。

## 入力パラメータ

<i>m</i>	INTEGER。行列 <i>A</i> の行数 ( $m \geq 0$ )。
<i>n</i>	INTEGER。 <i>A</i> の列数 ( $n \geq 0$ )。
<i>kl</i>	INTEGER。 <i>A</i> の帯内の劣対角成分の数 ( $kl \geq 0$ )。
<i>ku</i>	INTEGER。 <i>A</i> の帯内の優対角成分の数 ( $ku \geq 0$ )。
<i>ab</i>	REAL (sdbtrf の場合 ) DOUBLE PRECISION (ddbtrf の場合 ) COMPLEX (cdbtrf の場合 ) COMPLEX*16 (zdbtrf の場合 )。 配列、次元は ( <i>ldab</i> , <i>n</i> )。 行列 <i>A</i> を帯格納形式で行 <i>kl</i> +1 から <i>kl</i> + <i>ku</i> +1 に格納する。配列の行 1 から <i>kl</i> を設定する必要はない。 <i>A</i> の <i>j</i> 番目の列は配列 <i>ab</i> の <i>j</i> 番目の列に次のように格納する。 $\max(1, j - ku) \leq i \leq \min(m, j + kl)$ に対して $ab(kl + ku + 1 + i - j, j) = A(i, j)$ 。
<i>ldab</i>	INTEGER。 <i>ab</i> のリーディング・ディメンジョン。 ( $ldab \geq 2kl + ku + 1$ )

## 出力パラメータ

<i>ab</i>	終了時に、因子分解の詳細で上書きされる。 <i>U</i> は、 <i>kl</i> + <i>ku</i> 個の優対角成分を持つ上三角帯行列として、行 1 から <i>kl</i> + <i>ku</i> +1 に格納される。因子分解に使用した乗数は、行 <i>kl</i> + <i>ku</i> +2 から <i>kl</i> + <i>ku</i> +1 に格納される。詳細は、次の「アプリケーション・ノート」を参照。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> < 0 の場合、 <i>info</i> = - <i>i</i> は <i>i</i> 番目の引数の値が不正だったことを示す。 <i>info</i> > 0 の場合、 <i>info</i> = + <i>i</i> は <i>u</i> ( <i>i</i> , <i>i</i> ) が 0 であることを示す。因子分解は完了したが、係数 <i>U</i> は完全に特異である。連立 1 次方程式の解の算出に係数 <i>U</i> を使用すると、ゼロ除算が発生する。



## アプリケーション・ノート

帯格納方式の例を次に示す ( $m = n = 6$ 、 $kl = 2$ 、 $ku = 1$  の場合)。

入力

$$\begin{bmatrix} * & a_{12} & a_{23} & a_{34} & a_{45} & a_{56} \\ a_{11} & a_{22} & a_{33} & a_{44} & a_{55} & a_{66} \\ a_{21} & a_{32} & a_{43} & a_{54} & a_{65} & * \\ a_{31} & a_{42} & a_{53} & a_{64} & * & * \end{bmatrix}$$

出力

$$\begin{bmatrix} * & u_{12} & u_{23} & u_{34} & u_{45} & u_{56} \\ u_{11} & u_{22} & u_{33} & u_{44} & u_{55} & u_{66} \\ m_{21} & m_{32} & m_{43} & m_{54} & m_{65} & * \\ m_{31} & m_{42} & m_{53} & m_{64} & * & * \end{bmatrix}$$

\* でマークされた配列の成分はルーチンで使用されない。

## ?dttrf

一般三重対角行列の  $LU$  因子分解を、ピボット演算を用いないで計算する (ローカルブロック化アルゴリズム)。

### 構文

```
call sdttrf(n, dl, d, du, info)
call ddttrf(n, dl, d, du, info)
call cdttrf(n, dl, d, du, info)
call zdttrf(n, dl, d, du, info)
```

### 説明

このルーチンは、実 / 複素帯行列  $A$  の  $LU$  因子分解を、部分ピボット演算を用いない消去を使用して計算する。

因子分解は次の形式を持つ。

$$A = LU$$

$L$  単位下二重対角行列の積で、 $U$  は主対角成分と最初の優対角成分にのみ 0 でない値を持つ上三角行列である。

## 入力パラメータ

<i>n</i>	INTEGER。行列 <i>A</i> の次数 ( $n \geq 0$ )。
<i>dl</i> , <i>d</i> , <i>du</i>	REAL (sdttrf の場合 ) DOUBLE PRECISION (ddttrf の場合 ) COMPLEX (cdttrf の場合 ) COMPLEX*16 (zdttrf の場合 )。 <i>A</i> の成分を格納する配列。 次元 ( $n - 1$ ) の配列 <i>dl</i> は、 <i>A</i> の劣対角成分を格納する。 次元 <i>n</i> の配列 <i>d</i> は、 <i>A</i> の対角成分を格納する。 次元 ( $n - 1$ ) の配列 <i>du</i> は、 <i>A</i> の優対角成分を格納する。

## 出力パラメータ

<i>dl</i>	<i>A</i> の <i>LU</i> 因子分解で得られた行列 <i>L</i> を定義する、( $n-1$ ) 個の乗数によって上書きされる。
<i>d</i>	<i>A</i> の <i>LU</i> 因子分解で得られた上三角行列 <i>U</i> の <i>n</i> 個の対角成分によって上書きされる。
<i>du</i>	<i>U</i> の最初の優対角成分の ( $n-1$ ) 個の成分によって上書きされる。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> < 0 の場合、 <i>info</i> = - <i>i</i> は <i>i</i> 番目の引数の値が不正だったことを示す。 <i>info</i> > 0 の場合、 <i>info</i> = + <i>i</i> は <i>u</i> ( <i>i</i> , <i>i</i> ) が 0 であることを示す。因子分解は完了したが、係数 <i>U</i> は完全に特異である。連立 1 次方程式の解の算出に係数 <i>U</i> を使用すると、ゼロ除算が発生する。

---

## ?dttrsv

?dttrf によって行われた *LU* 因子分解を使用して、一般三重対角行列を係数行列とする連立 1 次方程式を解く。

---

## 構文

```
call sdttrsv(uplo, trans, n, nrhs, dl, d, du, b, ldb, info)
call ddttrsv(uplo, trans, n, nrhs, dl, d, du, b, ldb, info)
```

```
call cdttrsv(uplo, trans, n, nrhs, dl, d, du, b, ldb, info)
call zdttrsv(uplo, trans, n, nrhs, dl, d, du, b, ldb, info)
```

## 説明

このルーチンは、次の連立 1 次方程式のいずれかを解く。

$$LX = B, \quad L^T X = B, \quad \text{または} \quad L^H X = B$$

$$UX = B, \quad U^T X = B, \quad \text{または} \quad U^H X = B$$

[?dttrf](#) による  $LU$  因子分解で得られた三重対角行列  $A$  の係数

## 入力パラメータ

**uplo** CHARACTER\*1。  
 $L$  と  $U$  のどちらを解くのか指定する。

**trans** CHARACTER。'N'、'T'、または 'C' のいずれかでなければならない。  
方程式の形式を指定する。  
 $trans = 'N'$  の場合、 $AX = B$  を  $X$  について解く (転置なし)。  
 $trans = 'T'$  の場合、 $A^T X = B$  を  $X$  について解く (転置)。  
 $trans = 'C'$  の場合、 $A^H X = B$  を  $X$  について解く (共役転置)。

**n** INTEGER。行列  $A$  の次数 ( $n \geq 0$ )。

**nrhs** INTEGER。右辺の数、すなわち、行列  $B$  の列数 ( $nrhs \geq 0$ )。

**dl, d, du, b** REAL (sdttrsv の場合)  
DOUBLE PRECISION (ddttrsv の場合)  
COMPLEX (cdttrsv の場合)  
COMPLEX\*16 (zdttrsv の場合)。  
配列、次元は  $dl(n-1)$ 、 $d(n)$ 、 $du(n-1)$ 、 $b(ldb, nrhs)$ 。配列  $dl$  には、 $A$  の  $LU$  因子分解で得られた行列  $L$  を定義する  $(n-1)$  個の乗数が格納される。  
配列  $d$  には、 $A$  の  $LU$  因子分解で得られた上三角行列  $U$  の  $n$  個の対角成分が格納される。  
配列  $du$  には、 $U$  の最初の優対角成分の  $(n-1)$  個の成分が格納される。  
配列  $b$  には右辺の行列  $B$  を格納する。

**ldb** INTEGER。配列  $b$  のリーディング・ディメンジョン。  $ldb \geq \max(1, n)$ 。

## 出力パラメータ

**b** 解の行列  $X$  によって上書きされる。

*info* INTEGER。 *info* = 0 の場合、正常に終了したことを示す。  
*info* = -*i* の場合、*i* 番目のパラメータの値が不正であったことを示す。

---

### ?pttrsv

?pttrf によって行われた  $LDL^H$  因子分解を使用して、対称(エルミート)正定値三重対角行列を係数行列とする連立1次方程式を解く。

---

#### 構文

```
call sptrsv(trans, n, nrhs, d, e, b, ldb, info)
call dptrsv(trans, n, nrhs, d, e, b, ldb, info)
call cptrsv(uplo, trans, n, nrhs, d, e, b, ldb, info)
call zptrsv(uplo, trans, n, nrhs, d, e, b, ldb, info)
```

#### 説明

このルーチンは、次の三角法のいずれかを解く。

$L^T X = B$  または  $LX = B$  (実数型の場合)  
 $LX = B$  または  $L^H X = B$ 、  
 $UX = B$  または  $U^H X = B$  (複素数型の場合)

$L$  (複素数型の場合は  $U$ ) は、以下の条件を満たすエルミート正定値三重対角行列  $A$  のコレスキー因子である。

$A = LDL^H$  ([spttrf/dpttrf](#) で計算)

または

$A = U^H D U$  または  $A = LDL^H$  ([cpttrf/zpttrf](#) で計算)

#### 入力パラメータ

*uplo* CHARACTER\*1。 'U' または 'L' でなければならない。  
三重対角行列  $A$  の優対角成分と劣対角成分のどちらを格納するかと、因子分解の形式を指定する。

$uplo = 'U'$  の場合、 $e$  は  $U$  の優対角成分で、 $A = U'DU$ 。

$uplo = 'L'$  の場合、 $e$  は  $L$  の劣対角成分で、 $A = LDL'$ 。

$A$  が実数型の場合、2つの形式は等価である。

*trans*

CHARACTER。

連立方程式の形式を指定する。

実数型の場合：

$trans = 'N'$  の場合、 $LX = B$  (転置なし)

$trans = 'T'$  の場合、 $L^T X = B$  (転置)

複素数型の場合：

$trans = 'N'$  の場合、 $LX = B$  (転置なし)

$trans = 'N'$  の場合、 $LX = B$  (転置なし)

$trans = 'C'$  の場合、 $U^H X = B$  (共役転置)

$trans = 'C'$  の場合、 $L^H X = B$  (共役転置)

*n*

INTEGER。三重対角行列  $A$  の次数。 $n \geq 0$ 。

*nrhs*

INTEGER。右辺の数、すなわち、行列  $B$  の列数。 $nrhs \geq 0$ 。

*d*

REAL 配列、次元は  $(n)$ 。[?pttrf](#) による因子分解で得られた対角行列  $D$  の対角成分  $n$  個を格納する。

*e*

COMPLEX 配列、次元は  $(n-1)$ 。[?pttrf](#) による因子分解で得られた単位二重対角係数  $U$  または  $L$  の  $(n-1)$  個の非対角成分を格納する。 $uplo$  を参照。

*b*

COMPLEX。配列、次元は  $(ldb, nrhs)$ 。

右辺の行列  $B$  を格納する。

*ldb*

INTEGER。

配列  $b$  のリーディング・ディメンジョン。

$ldb \geq \max(1, n)$ 。

## 出力パラメータ

*b*

終了時に、解ベクトル  $X$  で上書きされる。

*info*

INTEGER。

$info = 0$  の場合、正常に終了したことを示す。

$info < 0$  の場合、 $info = -i$  は  $i$  番目の引数の値が不正だったことを示す。

## ?steqr2

暗黙的 QL 法または QR 法を使用して、対称三重行列の固有値をすべて計算し、さらにオプションで、固有ベクトルも計算する。

### 構文

```
call ssteqr2(compz, n, d, e, z, ldz, nr, work, info)
```

```
call dsteqr2(compz, n, d, e, z, ldz, nr, work, info)
```

### 説明

このルーチンは、LAPACK ルーチン [?steqr](#) の修正版である。ルーチン ?steqr2 は、暗黙的 QL 法または QR 法を使用して、対称三重行列の固有値をすべて計算し、さらにオプションで、固有ベクトルも計算する。?steqr2 は、?steqr2 を実行する各 ScaLAPACK プロセスが分散行列 Q を更新できるように、?steqr を修正したものである。?steqr2 の適切な使用方法については、ScaLAPACK ルーチン [p?syev](#) を参照。

### 入力パラメータ

**compz** CHARACTER\*1。'N' または 'I' でなければならない。  
compz = 'N' の場合、固有値のみ計算する。  
compz = 'I' の場合、三重対角行列  $T$  の固有値と固有ベクトルを計算する。  
このサブルーチンを実行する前に、[p?laset](#) または [?laset](#) で  $z$  を単位行列に初期化しなければならない。

**n** INTEGER。行列  $T$  の次数 ( $n \geq 0$ )。

**d, e, work** REAL (単精度の場合)  
DOUBLE PRECISION (倍精度の場合)。  
配列:  
 $d$  には、 $T$  の対角成分を格納する。  
 $d$  の次元は、 $\max(1, n)$  以上でなければならない。  
 $e$  には、 $T$  の  $(n-1)$  個の劣対角成分を格納する。  
 $e$  のサイズは、 $\max(1, n-1)$  以上でなければならない。  
 $work$  は、ワークスペース配列。  
 $work$  の次元は  $(1, 2*n-2)$ 。  
compz = 'N' の場合、work は参照されない。

<i>z</i>	(ローカル) REAL (ssteqr2 の場合) DOUBLE PRECISION (dsteqr2 の場合) 配列、グローバル次元は $(n, n)$ 、ローカル次元は $(ldz, nr)$ 。 <i>compz</i> = 'V' の場合、 <i>z</i> は三重対角形式への縮退に使用された直交行列を格納する。
<i>ldz</i>	INTEGER。配列 <i>z</i> のリーディング・ディメンジョン。 制約条件: $ldz \geq 1$ 。 固有ベクトルを計算する場合は、 $ldz \geq \max(1, n)$ 。
<i>nr</i>	INTEGER。 $nr = \max(1, \text{numroc}(n, nb, \text{myproc}, 0, nprocs))$ <i>compz</i> = 'N' の場合、 <i>nr</i> は参照されない。

### 出力パラメータ

<i>d</i>	REAL。配列、次元は $(n)$ (ssteqr2 の場合) DOUBLE PRECISION。配列、次元は $(n)$ (dsteqr2 の場合) 終了時に、 <i>info</i> = 0 の場合、固有値が昇順で格納される。 <i>info</i> も参照のこと。
<i>e</i>	REAL。配列、次元は $(n-1)$ (ssteqr2 の場合) DOUBLE PRECISION。配列、次元は $(n-1)$ (dsteqr2 の場合) 終了時に、 <i>e</i> は削除される。
<i>z</i>	(ローカル) REAL (ssteqr2 の場合) DOUBLE PRECISION (dsteqr2 の場合)。 配列、グローバル次元は $(n, n)$ 、ローカル次元は $(ldz, nr)$ 。 終了時に、 <i>info</i> = 0 の場合、 <i>compz</i> = 'V' ならば、元の対称行列の正規直交固有ベクトルが <i>z</i> に格納され、 <i>compz</i> = 'I' ならば、対称三重対角行列の正規直交固有ベクトルが <i>z</i> に格納される。 <i>compz</i> = 'N' の場合、 <i>z</i> は参照されない。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> < 0 の場合、 <i>info</i> = - <i>i</i> は <i>i</i> 番目の値が不正だったことを示す。 <i>info</i> > 0 の場合、このアルゴリズムで $30n$ 回の処理を繰り返した結果、すべての固有値を見つけられなかった。 <i>info</i> = <i>i</i> の場合、 <i>e</i> の <i>i</i> 個の成分が、ゼロに収束しなかった。終了時に、 <i>d</i> と <i>e</i> に、直交性が元の行列に近似している対称三重行列の成分を格納する。

## ユーティリティ関数とルーチン

このセクションでは、ScaLAPACK のユーティリティ関数とルーチンについて説明する。これらのルーチンに関するサマリを次の表に示す。

表 7-2 ScaLAPACK ユーティリティ関数とルーチン

ルーチン名	データ型	説明
<a href="#">p?labad</a>	s, d	指数範囲がきわめて大きい場合にアンダーフローおよびオーバーフローしきい値の平方根を返す。
<a href="#">p?lachkieee</a>	s, d	IEEE 標準機能に対して簡単なチェックを実行する (C インターフェイス関数)。
<a href="#">p?lamch</a>	s, d	浮動小数点演算に対するマシン・パラメータを決定する。
<a href="#">p?lasnbt</a>	s, d	浮動小数点演算の符号ビットの位置を計算する (C インターフェイス関数)。
<a href="#">pxerbla</a>		ScaLAPACK ルーチンから呼び出されるエラー処理ルーチン。

### p?labad

指数範囲がきわめて大きい場合にアンダーフローおよびオーバーフローしきい値の平方根を返す。

#### 構文

```
call pslabad(ictxt, small, large)
call pdlabad(ictxt, small, large)
```

#### 説明

このルーチンは、アンダーフローとオーバーフローに対して [p?lamch](#) によって計算された値を入力として受け取り、*large* のログが十分に大きい場合にそれら値のそれぞれの平方根を返す。このサブルーチンは、Cray のような大きな指数範囲を持つマシンを判別するために使用し、アンダーフローとオーバーフローのリミット値を [p?lamch](#) で計算された値の平方根に再定義する。Cray に見られるような上半分の指数範囲における計算能力が弱い場合でも [p?lamch](#) は補正を行わないため、このサブルーチンが必要となる。

さらに、このルーチンは異種混合のコンピューティング・ネットワークをサポートするために、これらの値の最小化および最大化をグローバルに実行する。



### 入力パラメータ

<i>ictxt</i>	(グローバル) INTEGER。 計算を行う BLACS コンテキスト・ハンドル。
<i>small</i>	(ローカル) REAL PRECISION (pslabad の場合) DOUBLE PRECISION (pdlabad の場合)。 p?lamch によって計算されたアンダーフローしきい値。
<i>large</i>	(ローカル) REAL PRECISION (pslabad の場合) DOUBLE PRECISION (pdlabad の場合)。 p?lamch によって計算されたオーバーフローしきい値。

### 出力パラメータ

<i>small</i>	(ローカル) 終了時に、 $\log_{10}(\textit{large})$ が十分に大きい場合、 <i>small</i> の平方根で上書きされ、そうでない場合は変更されない。
<i>large</i>	(ローカル) 終了時に、 $\log_{10}(\textit{large})$ が十分に大きい場合、 <i>large</i> の平方根で上書きされ、そうでない場合は変更されない。

---

## p?lachkieee

IEEE 標準機能に対して簡単なチェックを実行する (C インターフェイス関数)。

---

### 構文

```
void pslachkieee(int *isieee, float *rmax, float *rmin);  
void pdlachkieee(int *isieee, float *rmax, float *rmin);
```

### 説明

このルーチンは、簡単なチェックを実行し、IEEE 標準機能が実装されていることを確認する。一部の実装では、p?lachkieee が返されない場合がある。

すべての引数は参照による呼び出しであるため、このルーチンは Fortran コードから直接呼び出すことができる。

これは ScaLAPACK の内部サブルーチンであり、引数の妥当性は確認されない。

### 入力パラメータ

*rmax* (ローカル)  
REAL (pslachkieee の場合)  
DOUBLE PRECISION (pdlachkieee の場合)。  
オーバーフローしきい値 (= ?lamch('O'))。

*rmin* (ローカル)  
REAL (pslachkieee の場合)  
DOUBLE PRECISION (pdlachkieee の場合)  
アンダーフローしきい値 (= ?lamch('U'))。

### 出力パラメータ

*isieee* (ローカル) INTEGER。  
終了時に、*isieee* = 1 の場合、依存するすべての IEEE 標準機能が実装されていることを意味する。  
終了時に、*isieee* = 0 の場合、依存する IEEE 標準機能のいくつかが実装されていないことを意味する。

---

## p?lamch

浮動小数点演算に対するマシン・パラメータを決定する。

---

### 構文

```
val = pslamch(ictxt, cmach)  
val = pdlamch(ictxt, cmach)
```

### 説明

この関数は、単精度のマシン・パラメータを決定する。

### 入力パラメータ

*ictxt* (グローバル) INTEGER。計算を行う BLACS コンテキスト・ハンドル。

*cmach*

(グローバル) CHARACTER\*1。  
 p?lamch が返す値を指定する。  
 'E' または 'e' の場合、p?lamch := eps  
 'S' または 's' の場合、p?lamch := sfmin  
 'B' または 'b' の場合、p?lamch := base  
 'P' または 'p' の場合、p?lamch := eps\*base  
 'N' または 'n' の場合、p?lamch := t  
 'R' または 'r' の場合、p?lamch := rnd  
 'M' または 'm' の場合、p?lamch := emin  
 'U' または 'u' の場合、p?lamch := rmin  
 'L' または 'l' の場合、p?lamch := emax  
 'O' または 'o' の場合、p?lamch := rmax

eps = 相対マシン精度。  
 sfmin = 1/sfmin がオーバーフローしないような安全な最小値。  
 base = マシンの基数。  
 prec = eps\*base  
 t = 仮数における (基数) 桁数。  
 rnd = 丸めが追加で発生した場合 1.0、それ以外では 0.0。  
 emin = (緩やかに) アンダーフローを起こす前の最小指数。  
 rmin = アンダーフローしきい値 - base<sup>(emin-1)</sup>  
 emax = オーバーフローを起こす前の最大指数  
 rmax = オーバーフローしきい値 - (base<sup>emax</sup>)\*(1-eps)

## 出力パラメータ

*val*

関数の戻り値。

## p?lasnbt

浮動小数演算の符号ビットの位置を計算する。  
 (C インターフェイス関数)。

### 構文

```
void pslasnbt(int *ieflag);
void pdlasnbt(int *ieflag);
```

### 説明

このルーチンは、単精度 / 倍精度の浮動小数点値の符号ビットの位置を探す。このルーチンは IEEE 演算を仮定する。したがって、符号ビットの可能性のあるビット 32 (単精度の場合) またはビット 32 とビット 64 (倍精度の場合) のみテストする。sizeof(int) は 4 バイトとみなされる。

コンパイル・タイム・フラグ (NO\_IEEE) が、マシンに IEEE 演算がないことを示す場合、ieflag = 0 が返される。

### 出力パラメータ

**ieflag**                    INTEGER。  
このフラグは、任意の単精度 / 倍精度の浮動小数点値の符号ビットの位置を探す。  
コンパイル・タイム・フラグ (NO\_IEEE) が、マシンに IEEE 演算がないことを示す場合、または sizeof(int) が 4 バイトでない場合、ieflag = 0。ieflag = 1 は、単精度ルーチンの場合の符号ビットがビット 32 であることを示す。  
倍精度ルーチンの場合：  
ieflag = 1 は、符号ビットがビット 32 (ビッグ・エンディアン) であることを示す。  
ieflag = 2 は、符号ビットがビット 64 (リトル・エンディアン) であることを示す。

---

## pxerbla

ScaLAPACK ルーチンから呼び出されるエラー処理ルーチン。

---

### 構文

```
call pxerbla( ictxt, srname, info )
```

### 説明

このルーチンは、ScaLAPACK ルーチンのエラー処理ルーチンである。ScaLAPACK ルーチンで不正な入力パラメータの値があったときに呼び出される。

メッセージを出力する。プログラムの実行は停止しない。ScaLAPACK ドライバおよび計算ルーチンに対して、pxerbla の呼び出しに続き、RETURN 文が生成される。高レベ

ルの呼び出しルーチンに制御が返され、プログラムをどうするのかはユーザによって決められる。ただし、特殊なローレベル ScaLAPACK ルーチン (レベル 2 の計算ルーチンと等価な補助ルーチン) では、`pxerbla()` を呼び出した直後に、プログラムの実行を停止する `BLACS_ABORT()` が呼び出される。これは、このレベルでエラーから計算を復元することは不可能なためである。

ScaLAPACK ルーチンから戻った際に、`info` の非ゼロ値を確認するとよい。  
LAPACK を導入する場合に、システム固有の例外処理機能呼び出すのであれば、このルーチンの変更を考慮するとよい。

### 入力パラメータ

<code>ictxt</code>	(グローバル) INTEGER。 演算のグローバル・コンテキストを示す BLACS コンテキスト・ハンドル。コンテキストそのものがグローバル。
<code>sname</code>	(グローバル) CHARACTER*6。 <code>pxerbla</code> を呼び出したルーチンの名前。
<code>info</code>	(グローバル) INTEGER。 呼び出しルーチンのパラメータ・リストにおいて、不正なパラメータの位置。



# スパース・ソルバ・ルーチン

## 8

インテル® マス・カーネル・ライブラリ (インテル® MKL) には、ユーザが直接呼び出すことができる、実数、複素数係数の対称行列と対称構造行列を解く、線形スパースソルバが用意されている。スパース対称行列の場合、このソルバを使用して、正定値系と不定値系の両方を解くことができる。

インテル MKL のスパース・ソルバ・サブルーチンを使用するために必要な用語と概念は、「[線形ソルバの基礎](#)」に記述されている。線形スパースソルバおよびスパース行列 (疎行列) の格納形式について熟知している場合は、これらのセクションを省略し、直接、インターフェイスについて記述されているセクションに進んでよい。この後の節では、直接法スパースソルバ PARDISO\* について説明する。続いて、ステップ・バイ・ステップの解法を用いた、いくつかのインテル MKL ルーチンからなる 2 つの代替インターフェイス ([直接法スパースソルバ](#)と[反復法スパースソルバ](#)) について説明する。

## PARDISO - 並列化対応直接法スパース・ソルバ・インターフェイス

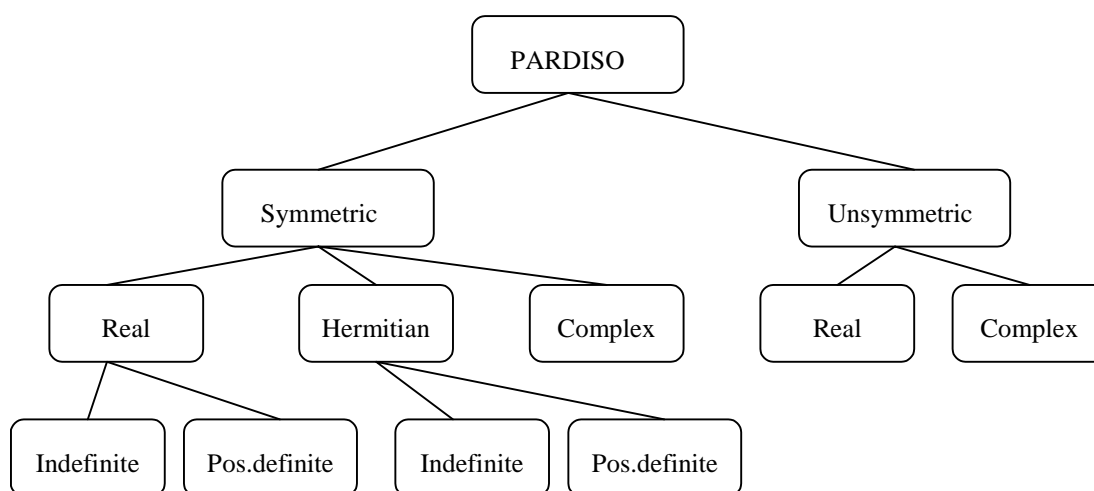
このセクションでは、PARDISO として知られる、メモリ共有型マルチプロセッシング並列化直接法スパースソルバについて説明する。インターフェイスは Fortran で記述されているが、サポート対象のコンパイラとオペレーティング・システムで使用される Fortran のパラメータの受け渡しと命名規則に従うことにより、C プログラムから呼び出すこともできる。PARDISO で使用されているアルゴリズムについての説明およびソルバについての詳細は、<http://www.computational.unibas.ch/cs/scicomp> (英語) を参照のこと。

PARDISO パッケージは、共有メモリ型マルチプロセッサ上で大規模なスパース対称連立 1 次方程式とスパース非対称連立 1 次方程式を解くことができる、高性能で、安定した、メモリ効率が良く、使いやすいソフトウェアである。ソルバは、レベル 3 BLAS の left-looking および right-looking スーパーノード手法 [Schenk00-2] を組み合わせて使用する。逐次および並列のスパース数値因子分解の性能を向上させるため、アルゴリズムはレベル 3 BLAS のアップデート版に基づき、パイプライン化並列処理は left-looking およ

び right-looking スーパーノード手法が組み合わされている [\[Schenk00\]](#)、[\[Schenk01\]](#)、[\[Schenk02\]](#)、[\[Schenk03\]](#)。並列ピボット演算法は、因子分解中の数値計算の安定性とスケーラビリティを両立させるために、完全なスーパーノード・ピボット演算を行うことができる。大規模な問題サイズの場合、並列アルゴリズムのスケーラビリティは共有メモリ型マルチプロセッシング・アーキテクチャにはほとんど依存せず、8 個のプロセッサを使用して 7 倍まで高速化されることが実証されている。

図 1 に示されるように、PARDISO は広範囲のスパース行列をサポートしており、共有メモリ型マルチプロセッシング・アーキテクチャで、実数または複素数、対称、構造対称または非対称、正定値、不定値またはエルミートのスパース連立 1 次方程式を解くことができる。

図 8-1 PARDISO で解くことができるスパース行列



PARDISO インターフェイス・ルーチンを使用して連立 1 次方程式を解くコードの例は、付録 C の「[PARDISO コードの例](#)」のセクションを参照のこと。



## pardiso

複数の右辺を持つスパース連立1次方程式の解を計算する。

### 構文

#### Fortran:

```
call pardiso(pt, maxfct, mnum, mtype, phase, n, a, ia, ja,
             perm, nrhs, iparm, msglvl, b, x, error)
```

#### C:

```
pardiso(pt, &maxfct, &mnum, &mtype, &phase, &n, a, ia, ja, perm, &nrhs,
        iparm, &msglvl, b, x, &error);
```

(“pardiso“ の後の下線は、OS とその OS 用のコンパイラの規則によって、必須である場合と必須でない場合がある)。

#### インターフェイス:

```
SUBROUTINE pardiso(pt, maxfct, mnum, mtype, phase, n, a, ia, ja,
                   perm, nrhs, iparm, msglvl, b, x, error)

INTEGER*4 pt(64)
INTEGER*4 maxfct, mnum, mtype, phase, n, nrhs, error,
          ia(*), ja(*), perm(*), iparm(*)
REAL*8 a(*), b(n,nrhs), x(n,nrhs)
```

注) 上記のインターフェイスは、32ビット・アーキテクチャ用である。64ビット・アーキテクチャの場合は、変数 pt(64) を INTEGER\*8 型で定義する。

### 説明

PARDISO は、複数の右辺を持つスパース連立1次方程式の解を計算する。

$$AX = B$$

右辺は複数列あり、*LU*、*LDL* または  $LL^T$  並列因子分解を用いる。*A* は  $n \times n$  の行列、*X* と *B* は  $n \times nrhs$  の行列である。PARDISO は、入力行列 *A* の構造に応じて、次の解析手順を実行する。

**対称行列：**最初に METIS パッケージ [Karypis98] (インテル MKL に含まれている) から最小次数アルゴリズム [Liu85] または Nested Dissection アルゴリズムに基づいて対称非零要素低減置換  $P$  を計算し、次に、対称正定値行列には  $PAP^T = LL^T$ 、対称不定値行列には  $PAP^T = LDL^T$  の並列 left-looking および right-looking コレスキー因子分解 [Schenk00-2] を計算する。この手順ではピボット演算は行なわない。 $X$  の近似値は前方 / 後方代入と反復改善法によって得られる。

**構造対称行列：**最初に対称非零要素低減置換  $P$  を計算し、次に、 $PAP^T = QLU^T$  の並列数値因子分解を計算する。スーパーノードで部分ピボット演算が使用され、 $X$  の近似値は前方 / 後方代入と反復改善法によって得られる。

**非対称行列：**最初に、大きな値を持つ要素を対角に移すために非対称置換  $P_{MPS}$  とスケールリング行列  $D_r$  と  $D_c$  を計算する。これは数値因子分解の信頼性を大きく向上させる [Duff99]。次に、行列  $P_{MPS}A + (P_{MPS}A)^T$  に基づいて非零要素低減置換  $P$  を計算し、スーパーノード・ピボット行列  $Q$  および  $R$  を使用して並列数値因子分解

$$QLUR = PP_{MPS}D_rAD_cP$$

を計算する。因子分解アルゴリズムを実行していき、このピボット演算法ではスーパーノードを因子分解できなくなったら、[Li99] のような摂動によるピボット演算法を使用する。ピボットの大きさは  $\varepsilon = \alpha \cdot \|A_2\|_\infty$  の定数しきい値でテストされる。ここで、 $\varepsilon$  は計算機の精度、 $A_2 = PP_{MPS}D_rAD_c$  と  $\|A_2\|_\infty$  はスケールリングと置換が行われた行列  $A$  の無限ノルムである。したがって、消去中に検出された小ピボットはすべて

$\text{sign}(l_{ij}) \cdot \varepsilon \cdot \|A_2\|_\infty$  と設定される。ピボットが小さくなりすぎることを防止することで、数値計算が安定する。失敗が多いと本質的に役に立たない因子分解が返されるが、大規模な行列で対角成分が変更されることは稀である。このため、一般に、このピボット演算で得られる因子分解は不正確で、反復改善法が必要となる。

**非対称連立 1 次方程式を解くための直接法と反復法の前処理：**過渡現象シミュレーション用の 1 次方程式を解くプロセスを加速するために、直接法と反復法を組み合わせで使用することもできる [Sonn89]。スパースソルバの多くのアプリケーションでは、非零係数行列の値が徐々に変化するが、スパースパターンが同一である方程式を解く必要がある。これらのアプリケーションでは、ソルバの解析フェーズは一度しか実行されないため、数値因子分解がシミュレーションで最も時間を費やす重要なステップとなる。

PARDISO は、最初の系に数値因子分解  $A = LU$  を使用して、得られた  $L$  と  $U$  を次の前処理付きクリロフ部分空間反復法のステップで適用する。反復計算が収束しない場合、ソルバは自動的に数値因子分解に戻る。この手法は、PARDISO における非対称行列に適用できる。ユーザは、1 つの入力パラメータのみを使用して、この手法を選択できる。詳細は、パラメータの説明 (`iparm(4)`, `iparm(20)`) を参照のこと。

**PARDISO のスパースデータ格納形式** 「[スパース行列の格納形式](#)」で記述されているように、*ja* が *columns*、*ia* が *rowIndex*、*a* が *values* を表す。

対称行列および構造対称行列の場合は、行ごとに列番号 *ja* を増加して順序付けする必要がある。また、行ごとに対角成分が必要である。非対称行列の場合は、対角成分は必要ない。

PARDISO では、解析とシンボリック因子分解、数値因子分解、反復改善法を含む前方 / 後方代入、すべての内部ソルバメモリを解放する終了化処理の 4 つの処理を実行できる。入力データ構造が呼び出しでアクセスされない場合、NULL ポインタまたは有効なアドレスを引数のプレースホルダとして渡すことができる。

## 入力パラメータ

*pt*            INTEGER\*4 (32 ビットのオペレーティング・システムの場合)  
                  INTEGER\*8 (64 ビットのオペレーティング・システムの場合)。  
                  配列、DIMENSION は (64)。  
                  ソルバの内部データのアドレスポインタ。アドレスはソルバに渡され、関連する内部メモリの管理はすべて、このポインタにより行われる。



**注** *pt* は、64 の成分を持つ整数配列である。PARDISO を最初に呼び出すとポインタがゼロに初期化されるが、これは非常に重要である。深刻なメモリリークが発生する可能性があるため、呼び出した後は決してポインタを変更するべきではない。整数の長さは 32 ビットのオペレーティング・システムでは 4 バイト、64 ビットのオペレーティング・システムでは 8 バイトでなければならない。

*maxfct*        INTEGER。  
                  ユーザがメモリ中に同時に保持する同じ非ゼロのスパース構造を持つ係数の最大数。同じ非ゼロの構造を持つ複数の異なる因子分解を同時にソルバの内部データ管理に格納することもできる。多くのアプリケーションで、この値は 1 である。

PARDISO は、行列のスパースパターンが同じである複数の行列を処理することができ、これらの行列の係数を同時に格納することが可能であることに注意。スパース構造が異なる行列は、それぞれ異なるメモリ・アドレス・ポインタ (*pt*) でメモリに保持できる。

*mnum*          INTEGER。  
                  解フェーズの実際の行列。このスカラで、因子分解する行列を定義できる。値は、 $1 \leq mnum \leq maxfct$  でなければならない。  
                  多くのアプリケーションで、この値は 1 である。

**mtype** INTEGER。  
このスカラ値は行列のタイプを定義する。PARDISO ソルバは次の行列をサポートしている。

<b>mtype</b> = 1	実構造対称行列
= 2	実対称正定値行列
= -2	実対称不定値行列
= 3	複素構造対称行列
= 4	複素エルミート正定値行列
= -4	複素エルミート不定値行列
= 6	複素対称行列
= 11	実非対称行列
= 13	複素非対称行列

このパラメータはピボット法に影響を与えることに注意。

**phase** INTEGER。  
ソルバの実行を制御する。2 桁の整数  $ij$  ( $10i+j$ ,  $1 \leq i \leq 3$ ,  $i < j \leq 3$ 。通常の実行モードの場合)。  $i$  桁は実行の開始フェーズ、  $j$  桁は終了フェーズを示す。PARDISO では次の実行フェーズがある。

- フェーズ 1: 非零要素低減解析とシンボリック因子分解
- フェーズ 2: 数値因子分解
- フェーズ 3: 順方向、逆方向算出 (反復改善法を含む)
- 終了 / メモリ解放フェーズ ( $phase \leq 0$ )

前のフェーズからの計算情報が前のルーチンへの呼び出しにある場合、任意のフェーズで実行を開始できる。 **phase** パラメータには次の値を指定できる。

<b>phase</b>	<b>ソルバの実行ステップ</b>
11	解析、シンボリック因子分解
12	解析、シンボリック因子分解、数値因子分解
13	解析、シンボリック因子分解、数値因子分解、解の算出
22	数値因子分解
23	数値因子分解、解の算出
33	解の算出
0	$L$ と $U$ 行列の番号 $mnum$ の内部メモリの解放
-1	すべての行列のメモリ解放

- n* INTEGER。  
方程式の数。これはスパース連立 1 次方程式  $AX = B$  における方程式の数である。次の制約がある。 $n > 0$ 。
- a* REAL/COMPLEX。  
配列。*ja* 内のインデックスに対応する係数行列 *A* の非ゼロの値を格納する。*a* のサイズは、*ja* のサイズと同じで、係数行列は実数または複素数のいずれかである。行列は、行ごとに *ja* の値が増加する行圧縮形式で格納されなければならない。詳細は、「[スパース行列の格納形式](#)」の *values* 配列の説明を参照のこと。



**注：**行列 *A* の各行の非ゼロは、昇順で格納されなければならない。対称行列または構造対称行列の場合、対角成分を利用でき、行列に格納されることも重要である。行列が対称の場合、配列 *a* は、因子分解のフェーズ、三角分解および反復改善フェーズでのみアクセスされる。非対称行列は、解のプロセスにおけるすべてのフェーズでアクセスされる。

- ia* INTEGER。  
配列、次元は  $(n+1)$ 。 $i \leq n$  の場合、*ia*(*i*) は行圧縮形式で格納された配列 *ja* 内にある *i* 行の最初の列インデックスを指す。つまり、*ia*(*i*) は、*A* の *i* 行からの最初の非ゼロの成分を格納する配列 *a* にある成分のインデックスを提供する。最後の成分 *ia*( $n+1$ ) は、*A* 内の非ゼロの数に 1 を加えたものに等しいとみなされる。詳細は、「[スパース行列の格納形式](#)」の *rowIndex* 配列の説明を参照のこと。  
配列 *ia* は、解のプロセスのフェーズでアクセスされる。行番号と列番号は、1 から始まることに注意。
- ja* INTEGER。  
配列。*ja*(\*) は、行圧縮形式で格納されているスパース行列 *A* の列インデックスを含む。各行のインデックスは、昇順で格納する。  
配列 *ia* は、解のプロセスのすべてのフェーズでアクセスされる。対称行列および構造対称行列の場合は、ゼロ対角成分は *a* と *ja* の非ゼロのリストにも格納される。対称行列の場合、ソルバは、「[スパース行列の格納形式](#)」の *columns* 配列のように、方程式の上三角部分だけを必要とする。
- perm* INTEGER。  
配列、次元は  $(n)$ 。サイズ *n* の置換ベクトルを格納する。  
配列 *perm* は、次のように定義される。*A* を元の行列とし、 $B = PAP^T$  を置換行列とする。*A* の *i* 行 (列) は、*B* の *perm*(*i*) 行 (列) である。配列の番号付けは 1 を起点とし、置換を説明する。

独自の非零要素低減順序付けをソルバに適用することができる。置換ベクトル *perm* は、*iparm*(5) = 1 の場合にのみアクセスされる。

*nrhs*      INTEGER。  
解を求める右辺の数。

*iparm*      INTEGER。  
配列、次元は (64)。この配列はさまざまなパラメータを PARDISO に渡すために使用され、ソルバの実行後に有益な情報を返すためにも使用される。*iparm*(1) = 0 の場合、PARDISO は *iparm*(2)、および *iparm*(4) から *iparm*(64) までをデフォルト値で埋め、それらの値を使用する。*iparm*(3) のデフォルト値はなく、*iparm*(1) が 0 か 1 かをユーザが指定しなければならないことに注意。

*iparm* 配列の個々の成分を以下に説明する (一部の成分は「出力パラメータ」のセクションで説明)。

## ***iparm*(1)**

*iparm*(1) = 0 の場合、*iparm*(2) および *iparm*(4) から *iparm*(64) まではデフォルト値で埋められる。それ以外では *iparm*(2) から *iparm*(64) までの *iparm* のすべての値をユーザが指定しなければならない。

## ***iparm*(2)**

*iparm*(2) は、入力行列に対する非零要素低減順序付けを制御する。*iparm*(2) が 0 の場合、最小次数アルゴリズムが適用され [5]、*iparm*(2) が 2 の場合は、ソルバは METIS パッケージからの Nested Dissection アルゴリズムを使用する [2]。

*iparm*(2) のデフォルト値は 2 である。

## ***iparm*(3)**

*iparm*(3) には、並列実行の際に利用できるプロセッサ数を格納する。プロセッサ数は OpenMP 環境変数 OMP\_NUM\_THREADS と等しくなければならない。



**注意：**ユーザが明示的に OMP\_NUM\_THREADS を設定していない場合、この値はオペレーティング・システムによりシステムの最大プロセッサ数に設定される。そのため、OMP\_NUM\_THREADS を明示的に設定して、ソルバの並列実行を制御することを推奨する。指定されたプロセッサ数よりも少ない数のプロセッサしか利用できない場合、実行速度は向上せずに、遅くなることもある。

$iparm(3)$  のデフォルト値はない。

#### $iparm(4)$

このパラメータは、非対称行列または構造対称行列の前処理付き CGS [11] および対称行列の共役勾配を制御する。

$iparm(4)$  は次の形式を持つ。

$$iparm(4) = 10 * L + K$$

値  $K$  と  $L$  の意味は次のとおりである。

値  $K$ :

$K$ の値	説明
0	$phase$ で必要なため、因子分解が常に行われる。
1	CGS 反復は $LU$ の計算を置換する。前処理は、同一のスパースパターンに必要な一連の方法における、前のステップ (最初のステップまたは失敗を含む最後のステップ) で計算された $LU$ である。
2	対称行列の CG 反復は $LU$ の計算を置換する。前処理は、同一のスパースパターンに必要な一連の方法における、前のステップ (最初のステップまたは失敗を含む最後のステップ) で計算された $LU$ である。

値  $L$ :

クリロフ部分空間反復法の停止条件を制御する。

$\epsilon_{CGS} = 10^{-L}$  は次の停止条件で使用される。

$$\|dx_i\| / \|dx_1\| < \epsilon_{CGS}$$

$\|dx_i\| = \|(LU)^{-1}r_i\|$  で、 $r_i$  は前処理付きクリロフ部分空間反復法における反復  $i$  の際の誤差である。

方針: 因子分解時間の半分を費やす前に反復計算は収束するという予想から、反復最大回数は 150 に固定される。中間収束率および誤差の偏位が確認され、反復プロセスを終了することができる。

$phase = 23$  の場合、クリロフ部分空間反復法が失敗し、対応する直接解が返されるようなケースでは、 $A$  の因子分解は自動で再計算される。それ以外では前処理付きクリロフ部分空間反復法からの解が返される。 $phase = 33$  を使用した場合、クリロフ部分空間反復法の停止条件を満たさないとエラー・メッセージ (error=4) が表示される。このエラーについての詳細は、 $iparm(20)$  から得ることができる。

デフォルトは  $iparm(4)=0$  で、上級ユーザでない限り、その他の値を使用しないことを推奨する。 $iparm(4)$  は 0 以上でなければならない。

例：

<b><i>iparm(4)</i></b>	<b>説明</b>
31	非対称行列の場合は停止条件が $10^{-3}$ の LU- 前処理付き CGS 反復法
61	非対称行列の場合は停止条件が $10^{-6}$ の LU- 前処理付き CGS 反復法
62	対称行列の場合は停止条件が $10^{-6}$ の LU- 前処理付き CGS 反復法

## ***iparm(5)***

*iparm(5)* では、統合 MMD (Multiple-Minimum Degree) アルゴリズムや Nested Dissection アルゴリズムの代わりにユーザ独自の非零要素低減置換を適用することができる。

このオプションは、順序付けアルゴリズムのテストや特殊なアプリケーション問題（例えば、ゼロの対角成分を  $\text{end PAP}^T$  に移動するなど）にコードを適応させる際に役立つ。置換の定義については、*perm* パラメータの説明を参照のこと。

*iparm(5)* のデフォルト値は 0 である。

## ***iparm(6)***

*iparm(6)* が 0 (デフォルト値) の場合、配列 *x* は解を格納し、*b* の値は変わらない。*iparm(6)* が 1 の場合、ソルバは右辺 *b* の解を格納する。

配列 *x* が常に使用されることに注意。*iparm(6)* のデフォルト値は 0 である。

## ***iparm(7)***

この値は参照されない。この値は将来用いることができるように予約されている。

## ***iparm(8)***

反復改善ステップの入力では、*iparm(8)* はソルバが実行する反復改善ステップの最大数に設定されなければならない。反復改善は後退誤差の観点から満足できるレベルの解の精度に達した場合に停止する。ソルバは、反復改善の *iparm(8)* ステップの絶対値を超えて実行されることはなく、後退誤差の点から満足できるレベルの解の精度に達した場合に停止する。

*iparm(8)* のデフォルト値は 0 である。

*iparm(8)* < 0 の場合、誤差の蓄積には拡張精度の実数データ型と複素データ型を使用する。摂動ピボットにより、反復改善法 (*iparm(8)*=0 とは関係なく) が行われ、実行された反復番号が *iparm(20)* に報告される。



#### ***iparm(9)***

この値は将来用いることができるように予約されている。値は、0 に設定しなければならない。

#### ***iparm(10)***

*iparm(10)* は PARDISO に対し、非対称行列 (*mtype* =11 または *mtype* =13) の小さなピボットやゼロピボットの処理方法を指示する。これらの行列の場合、ソルバは完全なスーパーノード・ピボット演算法を使用する。因子分解アルゴリズムを実行していき、このピボット演算法ではスーパーノードを因子分解できなくなったら、[\[Li99\]](#) のような摂動によるピボット演算法を使用する。ピボットの大きさは、次の定数しきい値でテストされる。

$$\varepsilon = \alpha \cdot \|A_2\|_{\infty}$$

ここで  $\varepsilon = 10^{-iparm(10)}$ 、 $\|PP_{MPS}D_rAD_cP\|_{\infty}$  はスケーリングと置換が行われた行列  $A$  の無限ノルムである。したがって、消去中に検出された小ピボットはすべて  $\text{sign}(l_{ij}) \cdot \varepsilon \cdot \|A_2\|_{\infty}$  と設定される。ピボットが小さくなりすぎることを防止することで、数値計算が安定する。それゆえ、小さなピボットは  $\varepsilon = 10^{-iparm(10)}$  に置換される。

*iparm(10)* のデフォルト値は 13 であるため、 $\varepsilon = 10^{-13}$  となる。

#### ***iparm(11)***

PARDISO は、最大重みマッチング・アルゴリズムを用いて対角上の大きな成分を置換し、行列をスケーリングして、対角成分が 1 に等しく、非対角成分の絶対値が 1 以下になるようにする。この手法は、非対称行列 (*mtype* =11 または *mtype* =13) にのみ適用され、デフォルトでは *iparm(11)*=1 で、このオプションは常にオンである。それ以外では、スケーリングは省略される。*iparm(11)* のデフォルト値は 1 である。

#### ***iparm(12)***

この値は将来用いることができるように予約されている。値は、0 に設定しなければならない。

#### ***iparm(13)***

この値は将来用いることができるように予約されている。値は、0 に設定しなければならない。

#### ***iparm(18)***

*iparm(18)* < 0 の場合、ソルバは係数の非ゼロ数を報告する。

*iparm(18)* のデフォルト値は 0 である。

## ***iparm(19)***

*iparm(19)* < 0 の場合、ソルバは行列 *A* の因子分解に必要な MFlop ( $10^6$ ) を報告する。このため、順序付け時間が増加する。

*iparm(19)* のデフォルト値は 0 である。

- msglvl* INTEGER。  
メッセージ・レベルの情報。*msglvl* = 0 の場合、PARDISO は出力を生成しない。  
*msglvl* = 1 の場合、ソルバは *pardiso.stat.nproc* ファイルに統計情報を出力する。
- b* REAL/COMPLEX  
配列、次元は (*n*, *nrhs*)。右辺のベクトル / 行列 *B* を格納する。  
*b* は、解フェーズでのみアクセスされる。

## 出力パラメータ

- pt* 内部アドレスポインタを格納する。
- iparm* 一部の *iparm* 値は有益な情報を格納する (例えば、係数にある非ゼロの数など)。

## ***iparm(14)***

因子分解の後、*iparm(14)* は、*mtype* = 11 または *mtype* = 13 に対する消去プロセス中に置換されたピボット数を格納する。

## ***iparm(15)***

*iparm(15)* は、解析およびシンボリック因子分解フェーズにおいてソルバが必要としたピークメモリの総量を KB でユーザに知らせる。この値はフェーズ 1 でのみ計算される。

## ***iparm(16)***

*iparm(16)* は、因子分解フェーズと解フェーズにおける解析およびシンボリック因子分解フェーズでソルバが必要とした永久メモリの総量を KB でユーザに知らせる。この値はフェーズ 1 でのみ計算される。

## ***iparm(17)***

*iparm(17)* は、因子分解フェーズおよび解フェーズでソルバが消費した倍精度のメモリ総量 (KB) をユーザに知らせる。この値はフェーズ 2 でのみ計算される。

ソルバが消費するピークメモリの総量は  $\max(iparm(15), iparm(16)+iparm(17))$  であることに注意。

**iparm(18)**

係数上の非ゼロ数がユーザに返される。

**iparm(19)**

行列  $A$  の因子分解に必要な MFlop ( $10^6$  回) における演算回数をユーザに返す。

**iparm(20)**

この値は、CG/CGS 診断 (例えば、反復回数や失敗の原因など) に用いられる。

$iparm(20) > 0$  の場合、CGS が行われ、実行された反復回数が  $iparm(20)$  にされる。

$iparm(20) < 0$  の場合、反復は実行されるが、CG/CGS は失敗する。

$iparm(20)$  のエラーレポートの詳細は、次の形式をとる。

$iparm(20) = -it\_cgs * 10 - cgs\_error$

$phase$  が 23 だった場合、行列  $A$  に対して係数  $L$ 、 $U$  が再計算されており、因子分解が正常に行われた場合はエラーフラグの  $error$  はゼロである。

$phase$  が 33 だった場合、 $error = -4$  はエラーを示す。

次の表は  $cgs\_error$  の説明である。

<b>cgs_error</b>	<b>説明</b>
1	誤差の変動が大きすぎる
2	$\ dx_{\max\_it\_cgs/2}\ $ が大きすぎる (収束が遅い)
3	停止条件が $\max\_it\_cgs$ に達しなかった
4	摂動ピボットにより反復改善法が行われた
5	因子分解がこの行列には速すぎる。 $iparm(4) = 0$ で因子分解法を用いるほうが良い。

**iparm(21) ~ iparm(64)**

これらの値は将来用いることができるように予約されている。値は、0 に設定しなければならない。

$b$   $iparm(6) = 1$  の場合、配列は解に置換される。

$x$  REAL/COMPLEX。  
配列、次元は  $(n, nrhs)$ 。 $iparm(6) = 0$  の場合、解を格納する。  
 $x$  は、解フェーズでのみアクセスされる。

`error` INTEGER。  
エラー・インジケータ：

<code>error</code>	情報
0	エラーなし
-1	入力が不整合
-2	十分なメモリがない
-3	順序付けの問題
-4	ゼロピボット、数値因子分解の問題
-5	分類されない (内部) エラー
-6	事前順序付けに失敗 (行列タイプ 11、13 のみ)
-7	対角行列の問題

## 直接法スパースソルバ (DSS) インターフェイス・ルーチン

インテル MKL は、直接法スパースソルバ用の PARDISO インターフェイスに代わるインターフェイス (ここでは DSS インターフェイスと呼ぶ) をサポートしている。DSS インターフェイスは、ユーザが呼び出すことのできるルーチングループを実装している。ステップ・バイ・ステップの解の算出プロセスで使用され、「[線形ソルバの基礎](#)」で説明されている一般的なスキームを利用してスパース連立 1 次方程式を解く。このインターフェイスには、解プロセスに関連した統計を収集するルーチンや Fortran ルーチンから C ルーチンに文字列を渡す補助ルーチンも含まれている。

解のプロセスは、概念的に 6 つのフェーズに分けられる。[表 8-1](#) は、ルーチン名、フェーズごとのグループ、一般的な使用法の説明をまとめたものである。

**表 8-1 DSS インターフェイス・ルーチン**

ルーチン	説明
<a href="#">dss_create</a>	ソルバを初期化し、ソルバに必要な基本的なデータ構造を作成する。このルーチンは、他の DSS ルーチンが呼び出される前に呼び出される必要がある。
<a href="#">dss_define_structure</a>	ソルバに、配列内の非ゼロ成分の場所を知らせるために使用される。
<a href="#">dss_reorder</a>	このルーチンは、配列の非ゼロ構造に基づいて、置換ベクトルを計算し、因子分解処理中の非零要素を低減させる。
<a href="#">dss_factor_real</a> 、 <a href="#">dss_factor_complex</a>	実行列または複素行列の $LU$ 、 $LDT^t$ または $LL^T$ 因子分解を計算する。

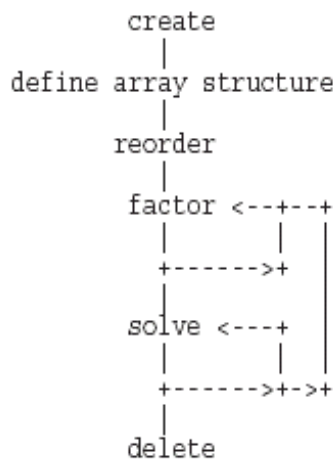
表 8-1 DSS インターフェイス・ルーチン

ルーチン	説明
<a href="#">dss_solve_real、 dss_solve_complex</a>	前のフェーズで計算された因子分解に基づき、連立方程式の解ベクトルを計算する。
<a href="#">dss_delete</a>	解のプロセスで作成されたすべてのデータ構造を削除する。
<a href="#">dss_statistics</a>	解のプロセスにおけるさまざまなフェーズについての統計を返す。順序付けに要した時間、因子分解に要した時間、求解時間、行列式、行列の慣性、因子分解中に行われた浮動小数点演算の回数などの統計を収集するために使用される。「順序付け」フェーズ後でかつ「削除」フェーズ前の解のプロセスにおける任意のフェーズで起動できる。このルーチンが起動されるフェーズに対応した適切な引数をルーチンに与える必要があることに注意。
<a href="#">mkl_cvt_to_null_terminated_str</a>	Fortran ルーチンから C ルーチンに文字列を渡すために使用される。

右辺が 1 つの連立方程式の解ベクトルを求めるため、[表 8-1](#) のリスト順 ([dss\\_statistics](#) を除く。表の説明を参照のこと) でインテル MKL DSS インターフェイス・ルーチンが起動される。

ただし、一部のアプリケーションでは、複数の右辺に対して解ベクトルを作成したり、同じ非ゼロ構造で複数の行列を因子分解する必要がある。そのため、インテル MKL スパースルーチンを表とは別の順序で起動できるようにすることも必要である。[図 8-2](#) のダイアグラムは、DSS インターフェイス・ルーチンの一般的な起動順を示す。

図 8-2 DSS インターフェイス・ルーチンを起動する一般的な順序



DSS インターフェイス・ルーチンを使用して連立 1 次方程式を解くコードの例は、付録 C の「[直接法スパース・ソルバのコード例](#)」のセクションを参照のこと。

## インターフェイスの説明

「[メモリ割り当てとハンドル](#)」のセクションで注記したように、各 DSS ルーチンは、ハンドルと呼ばれるデータ・オブジェクトを記述する。ハンドルの宣言は言語ごとに異なるため、本書では MKL\_DSS\_HANDLE 型として宣言されることとする。ハンドル引数の正しい宣言方法については、「[メモリ割り当てとハンドル](#)」を参照のこと。

本書のその他すべての型については、標準 Fortran 型の INTEGER、REAL、COMPLEX、DOUBLE PRECISION、DOUBLE COMPLEX を参照のこと。

C プログラマおよび C++ プログラマは、Fortran 型から C/C++ 型へのマッピングについての情報が記述された「[C/C++ からのスパース・ソルバ・ルーチンの呼び出し](#)」を参照のこと。

## ルーチン・オプション

すべての DSS ルーチンは、整数引数 (以下、*opt* として参照する) を持ち、さまざまなオプションをルーチンに渡す。*opt* に許容される値は言語固有のヘッダファイルで定義される (「[実装の詳細](#)」を参照)。すべてのルーチンは、[表 8-2](#) の説明のように、メッセージレベルや終了レベルの設定に使用されるオプションを受け入れる。さらに、すべてのルーチンは各 DSS ルーチンのデフォルト・オプションを確立する MKL\_DSS\_DEFAULTS オプションを受け入れる。

**表 8-2      メッセージ・レベルと終了レベルの記号名**

メッセージ・レベル	終了レベル
MKL_DSS_MSG_LVL_SUCCESS	MKL_DSS_TERM_LVL_SUCCESS
MKL_DSS_MSG_LVL_INFO	MKL_DSS_TERM_LVL_INFO
MKL_DSS_MSG_LVL_WARNING	MKL_DSS_TERM_LVL_WARNING
MKL_DSS_MSG_LVL_ERROR	MKL_DSS_TERM_LVL_ERROR
MKL_DSS_MSG_LVL_FATAL	MKL_DSS_TERM_LVL_FATAL

メッセージ・レベルおよび終了レベルは、任意の DSS ルーチンの呼び出しで設定できる。ただし、一旦、特定のレベルに設定されると、他の DSS ルーチンへの呼び出しで変更されるまで、そのレベルのままである。

オプションを同時に追加することによって、DSS ルーチンに複数のオプションを指定することができる。例えば、メッセージ・レベルをデバッグに、終了レベルをすべての DSS ルーチンのエラーに設定するには、次のようにする。

```
CALL dss_create( handle, MKL_DSS_MSG_LVL_INFO + MKL_DSS_TERM_LVL_ERROR)
```

## ユーザデータ配列

多くの DSS ルーチンは、ユーザデータ配列を入力として受け取る。例えば、ユーザ配列は `dss_define_structure` ルーチンに渡され、行列内にある非ゼロの成分の位置について説明する。格納要件を最小限に抑えて、全体のランタイム効率を向上させるため、インテル MKL DSS ルーチンはユーザ入力配列のコピーを作成しない。



**警告：** ソルバルーチンに渡された後、ユーザは、これらの配列の内容を編集することはできない。

### DSS ルーチン

---

## dss\_create

ソルバを初期化する。

---

### 構文

```
dss_create(handle, opt)
```

### 入力パラメータ

*opt*                                      INTEGER。パラメータを渡すオプション。デフォルト値は MKL\_DSS\_MSG\_LVL\_WARNING + MKL\_DSS\_TERM\_LVL\_ERROR。

### 出力パラメータ

*handle*                                      MKL\_DSS\_HANDLE 型のデータ・オブジェクト (「[インターフェイスの説明](#)」を参照)。

### 説明

dss\_create ルーチンは、ソルバを初期化するために呼び出される。dss\_create の呼び出し後に起動されるインテル MKL DSS ルーチンは dss\_create によって返されたハンドルの値を使用する。



---

**警告:** ハンドルの値を直接書き込まないこと。

---

### 戻り値

MKL\_DSS\_SUCCESS

MKL\_DSS\_INVALID\_OPTION

MKL\_DSS\_OUT\_OF\_MEMORY



## dss\_define\_structure

行列内の非ゼロ成分の位置をソルバに知らせる。

### 構文

```
dss_define_structure(handle, opt, rowIndex, nRows, nCols, columns,
                    nNonZeros);
```

### 入力パラメータ

<i>opt</i>	INTEGER。パラメータを渡すオプション。行列構造のデフォルト・オプションは、MKL_DSS_SYMMETRIC。
<i>rowIndex</i>	INTEGER。サイズ $\min(nRows, nCols)+1$ の配列。行列内の非ゼロ成分の場所を定義する。
<i>nRows</i>	INTEGER。行列の行数。
<i>nCols</i>	INTEGER。行列の列数。
<i>columns</i>	INTEGER。サイズ <i>nNonZeros</i> の配列。行列内の非ゼロ成分の場所を定義する。
<i>nNonZeros</i>	INTEGER。行列内の非ゼロ成分の数。

### 出力パラメータ

<i>handle</i>	MKL_DSS_HANDLE 型のデータ・オブジェクト (「 <a href="#">インターフェイスの説明</a> 」を参照)。
---------------	---

### 説明

dss\_define\_structure ルーチンは、サイズが  $nRows \times nCols$  の行列内の非ゼロ成分の数 *nNonZeros* をソルバに知らせる。

現在のところ、インテル MKL DSS ソフトウェアは正方行列に対してのみ動作するので、*nRows* と *nCols* は等しくなければならない。

行列内の非ゼロ成分の位置を知らせるには、次の操作を行う。

1. オプション引数 *opt* に次の値のいずれかを指定して、行列の一般的な非ゼロの構造を定義する。  
MKL\_DSS\_SYMMETRIC\_STRUCTURE  
MKL\_DSS\_SYMMETRIC  
MKL\_DSS\_NON\_SYMMETRIC
2. 配列 *rowIndex* と *columns* を用いて非ゼロの実際の位置を示す (「[スパース行列の格納形式](#)」を参照)。



**注：**現在、インテル MKL DSS ソフトウェアは、非対称行列を直接サポートしていない。その代わりに、MKL\_DSS\_NON\_SYMMETRIC オプションが指定されると、ソルバは適宜ゼロを追加して非対称行列を対称構造行列に変換する。

### 戻り値

MKL\_DSS\_SUCCESS  
MKL\_DSS\_STATE\_ERR  
MKL\_DSS\_INVALID\_OPTION  
MKL\_DSS\_COL\_ERR  
MKL\_DSS\_NOT\_SQUARE  
MKL\_DSS\_TOO\_FEW\_VALUES  
MKL\_DSS\_TOO\_MANY\_VALUES

---

## dss\_reorder

因子分解フェーズ中の非零要素を最小限に抑える  
置換ベクトルを計算する。

---

### 構文

`dss_reorder(handle, opt, perm)`

### 入力パラメータ

*opt*                      INTEGER。パラメータを渡すオプション。置換型のデフォルト・オプションは MKL\_DSS\_AUTO\_ORDER。

*perm* INTEGER。長さ *nRows* の配列。ユーザ定義の置換ベクトル (*opt* が MKL\_DSS\_MY\_ORDER を格納する場合にのみアクセスされる) を格納する。

### 出力パラメータ

*handle* MKL\_DSS\_HANDLE 型のデータ・オブジェクト (「[インターフェイスの説明](#)」を参照)。

### 説明

*opt* に MKL\_DSS\_AUTO\_ORDER オプションが含まれている場合、*dss\_reorder* は因子分解フェーズ中の非零要素を最小限に抑える置換ベクトルを計算する。このオプションでは、*perm* 配列はアクセスされない。

*opt* に MKL\_DSS\_MY\_ORDER オプションが含まれている場合、配列 *perm* はユーザが提供した置換ベクトルとみなされる。この場合、配列 *perm* は *nRows* 長である。ここで、*nRows* は前の [dss\\_define\\_structure](#) への呼び出しで定義された行列内の行数である。

### 戻り値

MKL\_DSS\_SUCCESS  
MKL\_DSS\_STATE\_ERR  
MKL\_DSS\_INVALID\_OPTION  
MKL\_DSS\_OUT\_OF\_MEMORY

---

## dss\_factor\_real、 dss\_factor\_complex

前に指定された位置にある行列の因子分解を計算する。

---

### 構文

```
dss_factor_real(handle, opt, rValues)
dss_factor_complex(handle, opt, cValues)
```

### 入力パラメータ

<i>handle</i>	MKL_DSS_HANDLE 型のデータ・オブジェクト (「 <a href="#">インターフェイスの説明</a> 」を参照)。
<i>opt</i>	INTEGER。パラメータを渡すオプション。行列のデフォルト・オプションは MKL_DSS_POSITIVE_DEFINITE。
<i>rValues</i>	DOUBLE PRECISION。サイズ <i>nNonZeros</i> の配列。行列内の実数の非ゼロ成分。
<i>cValues</i>	DOUBLE COMPLEX。サイズ <i>nNonZeros</i> の配列。行列内の複素数の非ゼロ成分。

### 説明

これらのルーチンは、前に [dss\\_define\\_structure](#) への呼び出しによって非ゼロの部分が指定された行列で、非ゼロの値が配列 *rValues* または *cValues* で与えられている行列の因子分解を行う。配列 *rValues* と *cValues* は前の [dss\\_define\\_structure](#) への呼び出しで定義されているように、*nNonZeros* 長とみなされる。

*opt* 引数は、次のいずれかのオプションを含む。

```
MKL_DSS_POSITIVE_DEFINITE、  
MKL_DSS_INDEFINITE、  
MKL_DSS_HERMITIAN_POSITIVE_DEFINITE、  
MKL_DSS_HERMITIAN_INDEFINITE
```

*rValues* と *cValues* の非ゼロの値が、正定値行列、不定値行列、またはエルミート行列かによりオプションは異なる。

### 戻り値

```
MKL_DSS_SUCCESS  
MKL_DSS_STATE_ERR  
MKL_DSS_INVALID_OPTION  
MKL_DSS_OPTION_CONFLICT  
MKL_DSS_OUT_OF_MEMORY  
MKL_DSS_ZERO_PIVOT
```

## dss\_solve\_real、 dss\_solve\_complex

対応する解ベクトルを計算して、出力配列に配置する。

### 構文

```
dss_solve_real(handle, opt, rRhsValues, nRhs, rSolValues)
dss_solve_complex(handle, opt, cRhsValues, nRhs, cSolValues)
```

### 入力パラメータ

<i>handle</i>	MKL_DSS_HANDLE 型のデータ・オブジェクト (「 <a href="#">インターフェースの説明</a> 」を参照)。
<i>opt</i>	INTEGER。パラメータを渡すオプション。
<i>nRhs</i>	INTEGER。1 次方程式の右辺の数。
<i>rRhsValues</i>	DOUBLE PRECISION。サイズ <i>nRows</i> × <i>nRhs</i> の配列。実数の右辺ベクトルを格納する。
<i>cRhsValues</i>	DOUBLE COMPLEX。サイズ <i>nRows</i> × <i>nRhs</i> の配列。複素数の右辺ベクトルを格納する。

### 出力パラメータ

<i>rSolValues</i>	DOUBLE PRECISION。サイズ <i>nCols</i> × <i>nRhs</i> の配列。実数の解ベクトルを格納する。
<i>cSolValues</i>	DOUBLE COMPLEX。サイズ <i>nCols</i> × <i>nRhs</i> の配列。複素数の解ベクトルを格納する。

### 説明

?RhsValues (? は *r* または *c*) で定義される各右辺の列では、これらのルーチンに対応する解ベクトルを計算し、配列 ?SolValues に置く。

右辺の長さと解ベクトル、*nCols* と *nRows* は前の [dss\\_define\\_structure](#) への呼び出しで定義されているものとみなされる。

### 戻り値

MKL\_DSS\_SUCCESS  
MKL\_DSS\_STATE\_ERR  
MKL\_DSS\_INVALID\_OPTION  
MKL\_DSS\_OUT\_OF\_MEMORY

---

## dss\_delete

解のプロセスで作成されたすべてのデータ構造を削除する。

---

### 構文

```
dss_delete(handle, opt)
```

### 入力パラメータ

*opt*                                      INTEGER。パラメータを渡すオプション。デフォルト値は MKL\_DSS\_MSG\_LVL\_WARNING + MKL\_DSS\_TERM\_LVL\_ERROR。

### 出力パラメータ

*handle*                                      MKL\_DSS\_HANDLE 型のデータ・オブジェクト (「[インターフェイスの説明](#)」を参照)。

### 説明

dss\_delete ルーチンは、解プロセス中に作成されたすべてのデータ構造を削除するために呼び出される。

### 戻り値

MKL\_DSS\_SUCCESS  
MKL\_DSS\_INVALID\_OPTION  
MKL\_DSS\_OUT\_OF\_MEMORY

## dss\_statistics

解のプロセスにおけるさまざまなフェーズについての統計を返す。

### 構文

```
dss_statistics(handle, opt, statArr, retValues)
```

### 入力パラメータ

<code>handle</code>	MKL_DSS_HANDLE 型のデータ・オブジェクト (「 <a href="#">インターフェースの説明</a> 」を参照)。
<code>opt</code>	INTEGER。パラメータを渡すオプション。
<code>statArr</code>	STRING。返す統計のタイプを定義する入力文字列。次の文字列定数のうち、1 つまたは複数を含むことができる (入力文字列の大文字・小文字は区別しない)。
	ReorderTime 順序付けに要した時間。
	FactorTime 因子分解に要した時間。
	SolveTime 因子分解後の求解に要した時間。
Determinant	行列 $A$ の行列式。実行列の場合、2 つの連続した戻り配列の位置で行列式は <code>det_pow</code> 、 <code>det_base</code> として返される。ここで、 $1.0 \leq \text{abs}(\text{det\_base}) < 10.0$ および $\text{determinant} = \text{det\_base} \cdot 10^{\text{det\_pow}}$ 複素数行列の場合、3 の連続した戻り配列の位置で行列式は <code>det_pow</code> 、 <code>det_re</code> 、 <code>det_im</code> が返される。ここで、 $1.0 \leq \text{abs}(\text{det\_re}) + \text{abs}(\text{det\_im}) < 10.0$ および $\text{determinant} = (\text{det\_re}, \text{det\_im}) \cdot 10^{\text{det\_pow}}$
Inertia	実対称行列の慣性は、3 つの非負整数 ( $p$ 、 $n$ 、 $z$ ) として定義される。ここで、 $p$ は正の固有値の数、 $n$ は負の固有値の数、 $z$ はゼロの固有値の数。  Inertia は、 $p$ 、 $n$ 、 $z$ と 3 つの連続した戻り配列の位置として返される。

Inertia の計算は安定した行列に対してのみ行うことを推奨する。不安定な行列に対して行くと、誤った結果をもたらすことがある。

実対称正定値行列  $k \times k$  の Inertia は常に  $(k,0,0)$ 。そのため、Inertia は実対称不定値行列の場合のみ返される。その他の行列タイプでは、エラー・メッセージが返される。

Flops 因子分解中の浮動小数点演算の数。



**注**: Fortran から C に文字列を渡す際の問題を回避するため、Fortran では `dss_statistics` を呼び出す前に `mk1_cvt_to_null_terminated_str` ルーチン呼び出さなければならない。詳細は、「[mk1\\_cvt\\_to\\_null\\_terminated\\_str](#)」を参照のこと。

## 出力パラメータ

`retValues` DOUBLE PRECISION。統計値が返される。

## 説明

`dss_statistics` ルーチンは、解のプロセスのさまざまなフェーズについての統計を返す。  
このルーチンを使用して次の分野の統計を収集できる。

- 順序付けに要した時間
- 因子分解に要した時間
- 求解時間
- 行列式
- 行列の慣性
- 因子分解中の浮動小数点演算の数

指定された入力文字列に対応する統計が返される。ユーザにより割り当てられた戻り配列で統計の値が倍精度で返される。

複数の統計の場合、カンマで区切られた文字列定数を入力として使用できる。戻り値は、入力文字列で指定されたものと同じ順序で戻り配列に置かれる。



統計は、解のプロセス中の適切な段階でのみ要求されなければならない。例えば、行列が因子分解される前に `FactorTime` が要求されるとエラーとなる。

次の表は、各統計の呼び出しポイントを示す。

**表 8-3 統計呼び出しシーケンス**

統計のタイプ	呼び出しのタイミング
ReorderTime	<code>dss_reorder</code> が正常に完了した後。
FactorTime	<code>dss_factor_real</code> または <code>dss_factor_complex</code> が正常に完了した後。
SolveTime	<code>dss_solve_real</code> または <code>dss_solve_complex</code> が正常に完了した後。
Determinant	<code>dss_factor_real</code> または <code>dss_factor_complex</code> が正常に完了した後。
Inertia	<code>dss_factor_real</code> が正常に完了した後。行列が実数、対称、不定値の場合。
Flops	<code>dss_factor_real</code> または <code>dss_factor_complex</code> が正常に完了した後。

次は、`dss_statistics` ルーチンの使用例である。

**例 8-1 「順序付けに要した時間」と行列の「慣性」の求め方**

これらの値を求めるには、次のルーチンを呼び出す。  
`dss_statistics(handle, opt, statArr, retValues)`、  
 ここで `statArr` は "ReorderTime,Inertia"

この例では、`retValues` には次の値が含まれる。

```
retValue[0]    順序付け時間。
retValue[1]    正の固有値。
retValue[2]    負の固有値。
retValue[3]    ゼロの固有値。
```

## 戻り値

```
MKL_DSS_SUCCESS
MKL_DSS_STATISTICS_INVALID_MATRIX
MKL_DSS_STATISTICS_INVALID_STATE
MKL_DSS_STATISTICS_INVALID_STRING
```

### **mkl\_cvt\_to\_null\_terminated\_str**

*Fortran* ルーチンから *C* ルーチンに文字列を渡す。

---

#### **構文**

```
mkl_cvt_to_null_terminated_str(destStr, destLen, srcStr)
```

#### **入力パラメータ**

*destLen*    INTEGER。出力配列 *destStr* の長さ。

*srcStr*     STRING。入力文字列。

#### **出力パラメータ**

*destStr*    INTEGER。1 次元の整数配列。

#### **説明**

`mkl_cvt_to_null_terminated_str` ルーチンは、**Fortran** ルーチンから **C** ルーチンに文字列を渡すために使用される。文字列は **C** に渡される前に整数配列に変換される。このルーチンを使用すると、**Fortran** から **C** に文字列を渡す際に一部のプラットフォームで発生する問題を回避できる。このルーチンの使用を強く推奨する。

### **実装の詳細**

インテル **MKL DSS** インターフェイスの一部は、プラットフォーム固有でかつ言語固有である。プラットフォーム間の移植性の確保と、異なる言語でも簡単に使用できるように、インテル **MKL DSS** の言語固有のヘッダファイルを使用することを推奨する。現在、次の 3 つの言語固有のヘッダファイルが用意されている。

- `mkl_dss.f77` - **F77** プログラム
- `mkl_dss.f90` - **F90** プログラム
- `mkl_dss.h` - **C** プログラム

これらの言語固有のヘッダファイルは、エラーの戻り値の記号定数、関数オプション、定義されたデータ型、関数プロトタイプを定義する。



**注：**ヘッダファイルで定義されている記号名でのみ、オプション、エラーの戻り値、メッセージ重要度の定数を参照することを推奨する。上記のヘッダファイルのいずれかをインクルードしないでインテル MKL DSS ソフトウェアを使用することはできない。

## メモリ割り当てとハンドル

インテル MKL DSS ルーチンをできるだけ簡単に使用できるように、ルーチンは、ユーザが一時作業領域を割り当てる必要がないように設計されている。ソルバに必要な領域（ユーザ入力ではない）は、ソルバ自身が割り当てる。複数のユーザが同時にソルバにアクセスできるように、ソルバは**ハンドル**と呼ばれるデータ・オブジェクトを使用して特定のアプリケーションに割り当てられた領域の軌跡を記録する。

各インテル MKL DSS ルーチンは、ハンドルを作成、使用、または削除する。そのため、ユーザ・プログラムは、ハンドルに領域を割り当てることができなければならない。ハンドルに割り当てる領域の構文は言語間で異なる。ハンドル宣言の標準化のため、言語固有のヘッダはユーザコードにハンドル・オブジェクトを宣言する際に使用するべき定数とデータ型を宣言する。

Fortran 90 プログラムでは、次のようにハンドルを宣言する。

```
INCLUDE "mkl_dss.f90"
TYPE(MKL_DSS_HANDLE) handle
```

C プログラムおよび C++ プログラムでは、次のようにハンドルを宣言する。

```
#include "mkl_dss.h"
_MKL_DSS_HANDLE_t handle;
```

8 バイト整数がサポートされたコンパイラを使用する Fortran 77 プログラムでは、次のようにハンドルを宣言する。

```
INCLUDE "mkl_dss.f77"
INTEGER*8 handle
```

それ以外では、INTEGER\*8 を DOUBLE PRECISION に置き換える。

インクルード・ファイルは、ハンドルの宣言を正しく定義することに加えて、次の項目も定義する。

- プロトタイプをサポートする言語の関数プロトタイプ

- エラーの戻り値に使用される記号定数
- ソルバルーチンのユーザ・オプション
- メッセージの重要度

## リバース・コミュニケーション・インターフェイス (RCI ISS) に基づく反復法スパースソルバ

### 共役勾配ソルバ (RCI CG)

インテル MKL は、PARDISO インターフェイスに加えて、RCI ベースの CG インターフェイス (ここでは RCI CG インターフェイスと呼ぶ) をサポートしている。RCI CG インターフェイスは、ユーザが呼び出すことが可能なルーチンで、対称正定値連立 1 次代数方程式をステップ・バイ・ステップの解の算出プロセスで解くために使用され、[\[Dong95\]](#) に説明されている一般的な RCI 体系を利用する。インテル MKL RCI CG サブルーチンを使用するために必要な用語と概念は、「[線形ソルバの基礎](#)」に記述されている。RCI とはユーザ自身が特定の演算をソルバ用に行うことを意味する (例えば、行列 - ベクトル演算など)。そのような演算結果をソルバが必要とするとき、ユーザは演算結果をソルバに渡さなければならない。行列 - ベクトル乗算などの特定の演算実装に依存しないため、ソルバに普遍性を持たせることができる。ただし、この方法ではユーザ側で若干の作業が必要となる。この作業を簡素化するため、ユーザは、組み込みのスパース行列 - ベクトル乗算と三角ソルバルーチン (第 2 章の「[スパース BLAS レベル 2 およびレベル 3](#)」を参照) を使用することができる。



**注：**この手法は、式の行列が対称、正定値でない場合に求解に失敗したり、誤った解を導くことがある。

解のプロセスは、概念的に 4 ステップに分けられる。[表 8-4](#) は、ルーチン名と一般的な使用法の説明をまとめたものである。

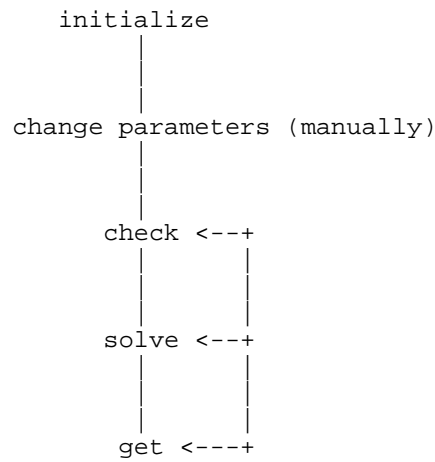
**表 8-4 RCI CG インターフェイス ルーチン**

ルーチン	説明
<a href="#">dcg_init</a>	ソルバを初期化する。
<a href="#">dcg_check</a>	ユーザ定義のデータの一貫性や正確性を検証する。
<a href="#">dcg</a>	近似解ベクトルを計算する。
<a href="#">dcg_get</a>	現在の反復数を検索する。

右辺が 1 つの連立方程式の解ベクトルを求めるため、通常表 8-4 のリスト順 ( コードの任意の場所で起動可能な dcg\_get ルーチンを除く ) でインテル MKL RCI CG インターフェイス・ルーチンが起動される。上級ユーザは、必要に応じてこの順序を変更することができる。一般ユーザは、この呼び出し順に従うことを推奨する。

図 8-3 のダイアグラムは、RCI CG インターフェイス・ルーチンの一般的な起動順を示す。

図 8-3 一般的な RCI CG インターフェイス・ルーチンの起動順



[図 8-4](#) は、RCICG ルーチンの一般的な使用体系を示す。

**図 8-4 RCI CG ルーチンの一般的な使用体系**

---

```
...
generate matrix A
generate preconditioner C (optional)
    call dcg_init(N, x, b, RCI_request, ipar, dpar, tmp)
    change parameters in ipar, dpar
    call dcg_check(N, x, b, RCI_request, ipar, dpar, tmp)
1 call dcg(N, x, b, RCI_request, ipar, dpar, tmp)
    if (RCI_request.eq.1)
        multiply the matrix A by tmp(1:N,1) and put the result in tmp(1:N,2)

この演算で、MKL スペース BLAS レベル 2 のサブルーチンを使用することもできる。

c      proceed with CG iterations
        go to 1
    end
    if (RCI_request.eq.2)
        do the stopping test
        if (test not passed) then
c      proceed with CG iterations
            go to 1
        else
c      stop CG iterations
            go to 2
        end if
    end
    if (RCI_request.eq.3) (optional)
        multiply the preconditioner C by tmp(1:N,3) and put the result in
tmp(1:N,2)
c      proceed with CG iterations
        go to 1
    end

2 call dcg_get(N, x, b, RCI_request, ipar, dpar, tmp, itercount)
```

現在の反復数は itercount にある。

計算された近似値は配列 x にある。

---

図 8-4 RCI CG ルーチンの一般的な使用体系

RCI CG インターフェイス・ルーチンを使用して連立 1 次方程式を解くコードの例は、付録 C の「[反復法スパース・ソルバのコード例](#)」のセクションを参照のこと。

## インターフェイスの説明

本書のすべての型については、標準 Fortran 型の INTEGER、DOUBLE PRECISION を参照のこと。

C プログラマおよび C++ プログラマは、Fortran 型から C/C++ 型へのマッピングについての情報が記述された「[C/C++ からのスパース・ソルバ・ルーチンの呼び出し](#)」を参照のこと。

## ルーチン・オプション

すべての RCI CG ルーチンはパラメータを持ち、さまざまなオプションをルーチンに渡す。これらのパラメータ値は、注意深く指定する必要がある（「[共通パラメータ](#)」を参照）、必要に応じて計算中に変更できるものでなければならない。



**注：**ユーザは、計算を失敗したり、誤った結果を取得しないように、正確で一貫性のあるパラメータをサブルーチンに渡さなければならない。

## ユーザデータ配列

多くの RCI CG ルーチンはユーザデータを入力として受け取る。例えば、ユーザ配列は dcg ルーチンに渡されて、連立 1 次代数方程式の解を計算する。格納要件を最小限に抑えて、全体のランタイム効率を向上させるため、インテル MKL RCI CG ルーチンはユーザ入力配列のコピーを作成しない。

## 共通パラメータ



**注:** 次にリストされたデフォルト値と初期値は、`dcg_init` ルーチン呼び出すことによってパラメータに割り当てられる。

`n` - INTEGER。このパラメータは、`dcg_init` ルーチンの問題サイズを設定する。他のルーチンは、代わりに `ipar(1)` を使用する。

`x` - DOUBLE PRECISION。配列。このパラメータは、解ベクトルの近似値を格納する。`dcg` ルーチンを初めて呼び出す前に、解ベクトルの初期近似値を格納する。

`gb` - DOUBLE PRECISION。配列。このパラメータは、右辺ベクトルを格納する。

`RCI_request` - INTEGER。このパラメータは、`RCICG` ルーチンの作業結果についての情報を知らせるために使用される。パラメータの負の値は、ルーチンがエラーまたは警告により終了したことを示す。0 値は、作業が正常に完了したことを示す。正の値は、ユーザが特定の動作を実行したことを示す。具体的には次のとおりである。

`RCI_request=1` - 行列に `tmp(1:N,1)` を掛け、結果を `tmp(1:N,2)` に置き、`dcg` ルーチンに制御を戻す。

`RCI_request=2` - 停止テストを実行する。失敗した場合、制御が `dcg` ルーチンに戻る。成功した場合、解が得られ、ベクトル `x` に格納される。

`RCI_request=3` - 前処理を `tmp(:,3)` に適用し、結果を `tmp(:,4)` に置き、`dcg` ルーチンに制御を戻す。

`dcg_get` ルーチンは `RCI_request` パラメータを変更しないことに注意。これにより、このルーチンをリバース・コミュニケーション計算内で使用することができる。

`ipar(128)` - INTEGER。配列。このパラメータは、`RCICG` 計算における整数データセットを指定するために用いられる。

`ipar(1)` - 問題のサイズを指定する。`dcg_init` ルーチンは `ipar(1)=n` を割り当てる。このパラメータにはデフォルト値はない。

`ipar(2)` - `RCICG` ルーチンにより生成されるエラー・メッセージと警告メッセージの出力タイプを指定する。デフォルト値の 6 は、すべてのメッセージが画面上に表示されることを意味する。それ以外の値の場合、エラー・メッセージと警告メッセージは新しく作成された `dcg_errors.txt` ファイルおよび



dcg\_check\_warnings.txt ファイルにそれぞれ書き込まれる。ipar(6) パラメータと ipar(7) パラメータが 0 に設定された場合、エラー・メッセージと警告メッセージは生成されないことに注意。

ipar(3) - RCI CG 計算の現在の過程を格納する。初期値は 1。



**注：**計算中は、この変数を変更しないことを強く推奨する。

ipar(4) - 現在の反復番号を格納する。初期値は 0。

ipar(5) - 反復の最大数を指定する。デフォルト値は  $\min \{150, n\}$ 。

ipar(6) - 値が 0 でない場合、ルーチンはエラー・メッセージを ipar(2) パラメータに沿って出力する。それ以外の場合、ルーチンはエラー・メッセージを出力せず、RCI\_request パラメータの負の値を返す。デフォルト値は 1。

ipar(7) - 値が 0 でない場合、ルーチンは警告メッセージを ipar(2) パラメータに沿って出力する。それ以外の場合、ルーチンは警告メッセージを出力せず、RCI\_request パラメータの負の値を返す。デフォルト値は 1。

ipar(8) - 値が 0 でない場合、dcg ルーチンは、反復の最大数を算出するために停止テストを行う。すなわち、 $\text{ipar}(4) \leq \text{ipar}(5)$ 。それ以外の場合、停止され、対応する値が RCI\_request に割り当てられる。値が 0 の場合、dcg ルーチンはこの停止テストを行わない。デフォルト値は 1。

ipar(9) - 値が 0 でない場合、dcg ルーチンは、誤差停止テストを行う。すなわち、 $\text{dpar}(5) \leq \text{dpar}(4) = \text{dpar}(1) \cdot \text{dpar}(3) + \text{dpar}(2)$ 。それ以外の場合、停止され、対応する値が RCI\_request に割り当てられる。値が 0 の場合、dcg ルーチンはこの停止テストを行わない。デフォルト値は 0。

ipar(10) - 値が 0 でない場合、dcg ルーチンは、RCI\_request=2 を設定してユーザ定義の停止テストを要求する。値が 0 の場合、dcg ルーチンはこのユーザ定義の停止テストを行わない。デフォルト値は 1。



**注：**ipar(8)-ipar(10) のパラメータのうち少なくとも 1 つは 1 に設定しなければならない。

ipar(11) - 値が 0 の場合、dcg ルーチンは共役勾配法を実行する。それ以外の場合、ルーチンは前処理付き共役勾配法を実行し、RCI\_request=3 を設定してユーザに前処理のステップを実行するように要求する。デフォルト値は 0。

*ipar*(12:128) は予約されており、現在実行されている RCI CG ルーチンでは使用されない。



---

**注：**上級ユーザは、コード内の配列を次のように定義することができる。  
INTEGER *ipar*(11)

---

*dpar*(128) - DOUBLE PRECISION。配列。このパラメータは、RCI CG 計算に倍精度のデータセットを指定するために用いられる。具体的には次のとおりである。

*dpar*(1) - 相対許容値を指定する。デフォルト値は 1.0D-6。

*dpar*(2) - 絶対許容値を指定する。デフォルト値は 0.0D-0。

*dpar*(3) - 初期誤差 (dcg ルーチン計算された場合) の二乗ノルムを指定する。初期値は 0。

*dpar*(4) - サービスルーチン。 *dpar*(1)\**dpar*(3)+*dpar*(2) と同じ (dcg ルーチンで計算された場合)。初期値は 0。

*dpar*(5) - 現在の誤差の二乗ノルムを指定する。初期値は 0。

*dpar*(6) - 前の反復ステップ (該当する場合) からの誤差の二乗ノルムを指定する。初期値は 0。

*dpar*(7) - CG 法の "alpha" パラメータを格納する。初期値は 0。

*dpar*(8) - CG 法の "beta" パラメータを格納する。 *dpar*(5)/*dpar*(6) と同じで、初期値は 0。

*dpar*(9:128) は予約されており、現在実行されている RCI CG ルーチンでは使用されない。



---

**注：**上級ユーザは、コード内の配列を次のように定義することができる。  
DOUBLE PRECISION *dpar*(8)

---

*tmp*(N,4) - DOUBLE PRECISION。配列。このパラメータは、RCI CG 計算に倍精度の一時スペースを与えるために用いられる。具体的には次のとおりである。

*tmp*(:,1) - 現在の検索方向を指定する。初期値は 0。

*tmp*(:,2) - 現在の検索方向で乗算された行列を格納する。初期値は 0。

*tmp*(:,3) - 現在の誤差を格納する。初期値は 0。

$tmp(:, 4)$  - 現在の誤差に割り当てられた前処理の逆数を格納する。このパラメータには初期値はない。



**注:** 上級ユーザは、反復のみを行う場合は、CG コード内の配列を次のように定義することができる。

```
DOUBLE PRECISION tmp(N, 3)
```

## RCI CG ルーチン

### dcg\_init

ソルバを初期化する。

#### 構文

```
dcg_init(n, x, b, RCI_request, ipar, dpar, tmp)
```

#### 入力パラメータ

$n$	INTEGER。問題のサイズおよび配列 $x$ と $b$ のサイズを格納する。
$x$	DOUBLE PRECISION。サイズ $n$ の配列。解ベクトルの初期近似値を格納する。通常、0 または $b$ 。
$b$	DOUBLE PRECISION。サイズ $n$ の配列。右辺ベクトルを格納する。

#### 出力パラメータ

$RCI\_request$	INTEGER。作業完了を知らせる。
$ipar$	INTEGER。サイズ 128 の配列。「 <a href="#">共通パラメータ</a> 」を参照のこと。
$dpar$	DOUBLE PRECISION。サイズ 128 の配列。「 <a href="#">共通パラメータ</a> 」を参照のこと。
$tmp$	DOUBLE PRECISION。サイズ $(n, 4)$ の配列。「 <a href="#">共通パラメータ</a> 」を参照のこと。

### 説明

dcg\_init ルーチンは、ソルバを初期化するために呼び出される。初期化後は、インテル MKL RCI CG ルーチンは dcg\_init によって返されるすべてのパラメータの値を使用することができる。上級ユーザは、このステップを省略して対応するルーチンのパラメータに直接値を設定することができる。



**警告:** 値が正しく、一貫性があることが確実にあれば、ソルバルーチンに配列を渡した後で、これらの配列の内容を編集できる。dcg\_check ルーチンを呼び出すことにより、正確性と一貫性の基本的な検証を行うことができる。ただし、この基本的な検証は、方法が正しいことを保証するものではないことに注意。

### 戻り値

`RCI_request = 0`      ルーチンは、作業を通常通り終了した。  
`RCI_request = -1`     ルーチンは、作業に失敗した。

---

## dcg\_check

ユーザ定義のデータの一貫性や正確性を検証する。

---

### 構文

```
dcg_check(n, x, b, RCI_request, ipar, dpar, tmp)
```

### 入力パラメータ

<code>n</code>	INTEGER。問題のサイズおよび配列 <code>x</code> と <code>b</code> のサイズを格納する。
<code>x</code>	DOUBLE PRECISION。サイズ <code>n</code> の配列。解ベクトルの初期近似値を格納する。通常、0 または <code>b</code> 。
<code>b</code>	DOUBLE PRECISION。サイズ <code>n</code> の配列。右辺ベクトルを格納する。

## 出力パラメータ

<i>RCI_request</i>	INTEGER。作業完了を知らせる。
<i>ipar</i>	INTEGER。サイズ 128 の配列。「 <a href="#">共通パラメータ</a> 」を参照のこと。
<i>dpar</i>	DOUBLE PRECISION。サイズ 128 の配列。「 <a href="#">共通パラメータ</a> 」を参照のこと。
<i>tmp</i>	DOUBLE PRECISION。サイズ $(n, 4)$ の配列。「 <a href="#">共通パラメータ</a> 」を参照のこと。

## 説明

dcg\_check ルーチンは、dcg ソルバルーチンに渡されるパラメータの一貫性と正確性を検証する。ただし、この演算はこの方法が正確な結果をもたらすことを保証するものではない。用いられるパラメータで誤った結果が導かれる率を減少させるだけである。上級ユーザは、ソルバ・パラメータに正しいデータが指定されていることがわかっていれば、このステップを省略することができる。  
すべてのベクトルの長さが前の dcg\_init サブルーチンへの呼び出しで定義されているとみなされる。

## 戻り値

<i>RCI_request</i> = 0	ルーチンは、作業を通常通り終了した。
<i>RCI_request</i> = -100	ルーチンは、一時停止された。エラーが発生した。
<i>RCI_request</i> = -1	ルーチンは、警告メッセージを返す。
<i>RCI_request</i> = -10	ルーチンは、一貫性または正確性が保持されるように一部のパラメータを変更した。
<i>RCI_request</i> = -11	ルーチンは、警告メッセージを返す。一部のパラメータを変更した。

---

## dcg

近似解ベクトルを計算する。

---

## 構文

```
dcg(n, x, b, RCI_request, ipar, dpar, tmp)
```

### 入力パラメータ

<i>n</i>	INTEGER。問題のサイズおよび配列 <i>x</i> と <i>b</i> のサイズを格納する。
<i>x</i>	DOUBLE PRECISION。サイズ <i>n</i> の配列。解ベクトルの初期近似値を格納する。
<i>b</i>	DOUBLE PRECISION。サイズ <i>n</i> の配列。右辺ベクトルを格納する。
<i>tmp</i>	DOUBLE PRECISION。サイズ ( <i>n</i> ,4) の配列。「 <a href="#">共通パラメータ</a> 」を参照のこと。

### 出力パラメータ

<i>RCI_request</i>	INTEGER。作業完了を知らせる。
<i>x</i>	DOUBLE PRECISION。サイズ <i>n</i> の配列。解ベクトルの更新された近似値を格納する。
<i>ipar</i>	INTEGER。サイズ 128 の配列。「 <a href="#">共通パラメータ</a> 」を参照のこと。
<i>dpar</i>	DOUBLE PRECISION。サイズ 128 の配列。「 <a href="#">共通パラメータ</a> 」を参照のこと。
<i>tmp</i>	DOUBLE PRECISION。サイズ ( <i>n</i> ,4) の配列。「 <a href="#">共通パラメータ</a> 」を参照のこと。

### 説明

dcg ルーチンは、CG 法 [\[Young71\]](#) を使用して、解ベクトルの近似を計算する。dcg ルーチンは、最初の呼び出しの前のベクトル *x* にある値を解の初期近似値として使用する。*RCI\_request* パラメータは、作業終了状況をユーザに知らせ、ソルバに必要な演算結果を要求する。

すべてのベクトルの長さが *dcg\_init* ルーチンへの前の呼び出しで定義されているとみなされる。

### 戻り値

<i>RCI_request</i> = 0	ルーチンは、作業を正常に完了し、求められた解をベクトル <i>x</i> に格納した。これは停止テストが完全に自動的に行われた場合にのみ行われる。ユーザが停止テストを定義した場合は、 <i>RCI_request</i> = 2 についての注釈を参照のこと。
<i>RCI_request</i> = -1	ルーチンは、反復の最大数に達したが、停止条件が満たされなかったために一時停止された (これは、ユーザが両方のテストを要求した場合にのみ発生する)。

<code>RCI_request= -2</code>	ルーチンは、ゼロの除算が起こったために一時停止された。(これは、行列のほとんどが 0 未満の正定値の場合に発生する)。
<code>RCI_request= -100</code>	ルーチンは、誤差ノルムが妥当でないために一時停止された。(おそらく、 <code>dpar(6)</code> にあるデータはルーチンの外に変更されるか、 <code>dcg_check</code> ルーチンは呼び出されない)。
<code>RCI_request= -101</code>	ルーチンは、無限サイクルに入ってしまったために一時停止された。(おそらく、 <code>lpar(8)</code> 、 <code>lpar(9)</code> 、 <code>lpar(10)</code> にあるデータはルーチンの外に変更されるか、 <code>dcg_check</code> ルーチンは呼び出されない)。
<code>RCI_request= 1</code>	ユーザに <code>tmp(1:N,1)</code> で行列を乗算し、その結果を <code>tmp(1:N,2)</code> に置いて、 <code>dcg</code> ルーチンに制御を戻すように要求する。
<code>RCI_request= 2</code>	ユーザに停止テストを行うように要求する。失敗した場合、ユーザは <code>dcg</code> ルーチンに制御を戻さなければならない。成功した場合、解が得られ、ベクトル <code>x</code> に格納される。
<code>RCI_request= 3</code>	前処理を <code>tmp(:,3)</code> に適用し、結果を <code>tmp(:,4)</code> に置き、 <code>dcg</code> ルーチンに制御を戻す。

## dcg\_get

現在の反復数を検索する。

### 構文

```
dcg_get(n, x, b, RCI_request, ipar, dpar, tmp, itercount)
```

### 入力パラメータ

<code>n</code>	INTEGER。問題のサイズおよび配列 <code>x</code> と <code>b</code> のサイズを格納する。
<code>x</code>	DOUBLE PRECISION。サイズ <code>n</code> の配列。解の初期近似ベクトルを格納する。
<code>b</code>	DOUBLE PRECISION。サイズ <code>n</code> の配列。右辺ベクトルを格納する。
<code>RCI_request</code>	INTEGER。作業完了についての情報を格納する。
<code>ipar</code>	INTEGER。サイズ 128 の配列。「 <a href="#">共通パラメータ</a> 」を参照のこと。

<i>dpar</i>	DOUBLE PRECISION。サイズ 128 の配列。「 <a href="#">共通パラメータ</a> 」を参照のこと。
<i>tmp</i>	DOUBLE PRECISION。サイズ ( <i>n</i> , 4) の配列。「 <a href="#">共通パラメータ</a> 」を参照のこと。

### 出力パラメータ

<i>itercount</i>	INTEGER。引数。現在の反復番号の値を格納する。
------------------	----------------------------

### 説明

`dcg_get` ルーチンは、解プロセス中の現在の反復番号を取得するために呼び出される。

### 戻り値

`dcg_get` ルーチンは、値を返さない。

## 実装の詳細

インテル MKL RCI CG インターフェイスの一部は、プラットフォーム固有でかつ言語固有である。プラットフォーム間の移植性の確保と、異なる言語でも簡単に使用できるように、インテル MKL RCI CG の言語固有のヘッダファイルを使用することを推奨する。現在、C プログラム用の言語固有のヘッダファイルが用意されている。

言語固有のヘッダファイルは、関数のプロトタイプおよび次のような項目を定義する。

```
void dcg_init(int *n, double *x, double *b, int *rci_request, int *ipar,
             double *dpar, double *tmp);
void dcg_check(int *n, double *x, double *b, int *rci_request, int *ipar,
              double *dpar, double *tmp);
void dcg(int *n, double *x, double *b, int *rci_request, int *ipar,
         double *dpar, double *tmp);
void dcg_get(int *n, double *x, double *b, int *rci_request, int *ipar,
            double *dpar, double *tmp, int *itercount);
```



---

**注:** 言語固有のヘッダファイルをインクルードしないでインテル MKL RCI CG ソフトウェアを使用することはできない。

---



## C/C++ からのスパース・ソルバ・ルーチンの呼び出し

インテル MKL スパース・ソルバ・ルーチンの呼び出しインターフェイスは、Fortran 77 または Fortran 90 から簡単に使用できるように設計されている。ただし、使用しているプラットフォームの異言語呼び出し規則について知識がある場合、これらのルーチンは C または C++ から直接起動することもできる。規則には、その言語での引数の渡し方、Fortran から C/C++ へのデータ型マッピング、およびプラットフォームにおける Fortran 外部名の修飾方法などが含まれる。

移植性を向上させ、ユーザの負担を軽減させるため、C ヘッダファイルでは、異言語呼び出し規則を隠蔽するためのマクロセットと型定義が提供され、C/C++ にとって自然なインターフェイスをインテル MKL スパース・ソルバ・ルーチンに提供する。

例えば、foo というライブラリ・ルーチンがあるとする。このルーチンは、長さが  $n$  の実ベクトルをとり、整数ステータスを返す。Fortran の場合、このような関数には次のようにアクセスする。

```
INTEGER n, status, foo
REAL x(*)
status = foo(x, n)
```

C 言語では、foo を起動する場合に、Fortran の INTEGER 型と REAL 型に対応する C のデータ型、Fortran コンパイラが使用する引数の渡し方、そして (該当する場合は) 外部シンボル foo の生成時に Fortran コンパイラにより実行される名前修飾などを知っておく必要がある。

しかし、例えば mkl\_solver.h などの C 固有のヘッダファイルを使用すると、foo の起動は C プログラムでは次のようになる。

```
#include "mkl_solver.h"
_INTEGER_t i, status;
_REAL_t x[];
status = foo( x, i );
```

上記の例でわかるように、ヘッダファイルの mkl\_solver.h は、Fortran の INTEGER 型と REAL 型に対応する、\_INTEGER\_t 型と \_REAL\_t 型を提供する。

インテル MKL スパース・ソルバ・ルーチンを C および C++ から簡単に使用できるように、Fortran 型の C 定義がライブラリ全体で使用される。特に、スパースソルバからの引数または結果が Fortran 言語固有の型である xxx の場合は、C ヘッダファイルおよび C++ ヘッダファイルは \_xxx\_t に適切な C 言語型の定義を提供する。

### C ユーザが気をつけるべきこと

C/C++ と Fortran の主な違いは、引数の渡し方が異なるということである。Fortran プログラムは参照渡しセマンティクスを使用するのに対し、C/C++ プログラムは値渡しセマンティクスを使用する。前のセクションの例では、ヘッダファイルの `mkl_solver.h` は、適切な引数のアドレスをとるマクロ `foo` を定義することにより、この相違を隠蔽しようとした。例えば、Tru64 UNIX の場合、`mkl_solver.h` はマクロを次のように定義する。

```
#define foo(a,b) foo_((a), &(b))
```

注意すべき点は、`foo` というマクロ形式を使用する際、定数をどのように処理するかである。`foo( x, 10 )` と記述すると、`foo_( x, &10 )` と翻訳される。ANSI に厳密に準拠する C コンパイラでは、定数のアドレスは許容されないため、厳密に準拠したプログラムは次のようになる。

```
_INTEGER_t iTen = 10;
_REAL_t * x;
status = foo( x, iTen );
```

ただし、ANSI 規格に準拠していない一部の C コンパイラでは、Fortran プログラムでも簡単に使用できるように定数のアドレスをとることができる。このように、`foo( x, 10 )` はこのようなコンパイラでは許容される。

# ベクトル数学関数

## 9

本章では、ベクトル引数に基づいて基本関数を計算するベクトル数学関数ライブラリ (VML) について説明する。VML は、インテル<sup>®</sup> マス・カーネル・ライブラリの必須部分である。ここでは、VML に含まれる関数の説明を簡単にするために、VML 固有の用語を使用する。

VML では、ベクトルを演算するためのコアとなる数学関数の中でも非常に計算時間を要する一群の関数 ( 累乗、三角方程式、指数、双曲線など ) について、高度な最適化をし実装している。

VML により、非線形プログラミング・ソフトウェアや積分計算などの多くのプログラムの性能が大幅に向上すると期待できる。VML は FORTRAN と C 言語のどちらのインターフェイスも備えている。

VML 関数は、実行する演算に応じて、以下のグループに分けられる。

- [VML 数学関数](#) 単位増分のインデックスを使用するベクトルに基づいて、基本関数 ( 正弦、余弦、指数、対数など ) の値を計算する。
- [VML Pack/Unpack 関数](#) 正増分インデックス、ベクトル・インデックス、マスク・インデックスを使用するベクトルへの変換、およびその逆の変換を実行する ( ベクトル・インデックス方式の詳細は、[付録 B](#) を参照のこと ) 。
- [VML サービス関数](#) 精度モードの設定 / 取得、エラーコードの設定 / 取得を行う。

VML 数学関数は、入力ベクトルを引数とし、それぞれの基本関数の値を成分ごとに計算して、結果を出力ベクトルに入れて戻す。

## データ型と精度モード

VML の数学ベクトル関数と pack/unpack ベクトル関数は、単精度実数データと倍精度実数データのベクトル引数に対応するように実装されている。ライブラリには、VML サービス関数を含むすべての関数に対する Fortran インターフェイスと C インターフェイスが用意されている。Fortran インターフェイスと C インターフェイスにおける関数の命名方法と呼び出し方法の違いは、この後の[関数命名規則](#)のセクションで説明する。

VML の各ベクトル関数は、( 各データ形式ごとに ) 高精度 (HA) と低精度 (LA) の 2 つのモードで機能できる。多くの関数の場合、LA 版を使用すると、精度が低下するが処理速度が向上する。

しかし、精度を落としても処理速度がほとんど向上しない場合には、両方のモードに対して同じ関数を使用される。エラーの動作は、HA モードと LA モードのどちらを選択しているかだけではなく、ソフトウェアを実行するプロセッサによっても変わってくる。

また、特殊な値に対する動作は、関数で HA モードと LA モードのどちらを使用しているかにより異なる場合がある。精度に関係した事項は、インテル MKL リリースノートを参照のこと。

HA モードと LA モードを切り替えるには、vmlSetMode(mode) を使用する ( [表 9-11](#) を参照 )。関数 vmlGetMode() は、現在使用しているモードを戻す。デフォルトでは、高精度 (HA) モードが使用される。

## 関数命名規則

Fortran インターフェイスの場合、VML 関数のフルネームには小文字しか含まれない。C インターフェイスの場合は、小文字と大文字が混在して含まれる。

VML 数学関数と pack/unpack 関数のフルネームは、以下の形式で指定する。

v <p> <name> <mod>

先頭文字の v は、関数が VML に属していることを示すプリフィックスである。

<p> フィールドは、以下のデータ型を指定する精度プリフィックスである。

s	REAL (Fortran インターフェイスの場合)、float (C インターフェイスの場合)
d	DOUBLE PRECISION (Fortran インターフェイスの場合)、double (C インターフェイスの場合)

<name> フィールドは関数の短縮名を示し、C インターフェイスの場合は一部大文字が含まれる ( [表 9-2](#) あるいは [表 9-10](#) の例を参照 )。

<mod> フィールド (C インターフェイスの場合は大文字で表記) は、pack/unpack 関数でのみ指定する。次に示すインデックス方式を指定する。

i	正増分を使用するインデックス
v	インデックス・ベクトルを使用するインデックス
m	マスクベクトルを使用するインデックス

VML サービス関数のフルネームは、次の形式で指定する。

vml <name>

ここで vml は関数が VML に属しているのを示すプリフィックスであり、<name> は関数の短縮名である。C インターフェイスの場合は一部大文字が含まれる。(表 9-10 を参照のこと)。

アプリケーション・プログラムから VML 関数を呼び出すには、従来の関数呼び出しを使用する。例えば、単精度データの VML 指数関数は、次のように呼び出せる。

```
call vsexp ( n, a, y );    (Fortran インターフェイスの場合)、
vsExp ( n, a, y );        (C インターフェイスの場合)。
```

## 関数インターフェイス

VML 関数のインターフェイスには、関数のフルネームと一連の引数が含まれる。それぞれの VML 関数グループにおける Fortran インターフェイスと C インターフェイスの説明を以下に示す。関数によっては、引数として a と b の 2 つの入力ベクトルを持つものがある (Div、Pow、Atan2)。一方、SinCos 関数は、y と z の 2 つの出力ベクトルを持つ。

### VML 数学関数

Fortran:

```
call v<p><name>( n, a, y )
call v<p><name>( n, a, b, y )
call v<p><name>( n, a, y, z )
```

C:

```
v<p><name>( n, a, y );
v<p><name>( n, a, b, y );
v<p><name>( n, a, y, z );
```

### Pack 関数

Fortran:

```
call v<p>packi( n, a, inca, y )
call v<p>packv( n, a, ia, y )
call v<p>packm( n, a, ma, y )
```

C:

```
v<p>PackI( n, a, inca, y );
v<p>PackV( n, a, ia, y );
v<p>PackM( n, a, ma, y );
```

### Unpack 関数

Fortran:

```
call v<p>unpacki( n, a, y, incy )
call v<p>unpackv( n, a, y, iy )
call v<p>unpackm( n, a, y, my )
```

C:

```
v<p>UnpackI( n, a, y, incy );
v<p>UnpackV( n, a, y, iy );
v<p>UnpackM( n, a, y, my );
```

### サービス関数

Fortran:

```
oldmode = vmlsetmode( mode )
mode     = vmlgetmode( )
olderr   = vmlseterrstatus ( err )
err      = vmlgeterrstatus( )
olderr   = vmlclearerrstatus( )
oldcallback = vmlseterrorcallback( callback )
callback  = vmlgeterrorcallback( )
oldcallback = vmlclearerrorcallback( )
```

C:

```
oldmode = vmlSetMode( mode );  
mode     = vmlGetMode( void );  
olderr   = vmlSetErrStatus( err );  
err      = vmlGetErrStatus( void );  
olderr   = vmlClearErrStatus( void );  
oldcallback = vmlSetErrorCallBack( callback );  
callback  = vmlGetErrorCallBack( void );  
oldcallback = vmlClearErrorCallBack( void );
```

### 入力パラメータ

<i>n</i>	計算する成分の数
<i>a</i>	1 番目の入力ベクトル
<i>a</i>	1 番目の入力ベクトル
<i>inca</i>	入力ベクトル <i>a</i> のベクトル増分
<i>ia</i>	入力ベクトル <i>a</i> のインデックス・ベクトル
<i>ma</i>	入力ベクトル <i>a</i> のマスクベクトル
<i>incy</i>	出力ベクトル <i>y</i> のベクトル増分
<i>iy</i>	出力ベクトル <i>y</i> のインデックス・ベクトル
<i>my</i>	出力ベクトル <i>y</i> のマスクベクトル
<i>err</i>	エラーコード
<i>mode</i>	VML モード
<i>callback</i>	コールバック関数のアドレス

### 出力パラメータ

<i>y</i>	1 番目の出力ベクトル
<i>z</i>	2 番目の出力ベクトル
<i>err</i>	エラーコード
<i>mode</i>	VML モード
<i>olderr</i>	前のエラーコード

`oldmode` 前の VML モード

`oldcallback` 前のコールバック関数のアドレス

各関数で使用するパラメータのデータ型は、それぞれの関数の説明のセクションで記述する。VML 数学関数はすべて、インプレース演算を実行できる。つまり、同一のベクトルを入力パラメータとしても出力パラメータとしても使用可能である。これは、2つの入力ベクトルを持つ関数にも当てはまる。この場合、2つの入力パラメータの一方が、出力ベクトルで上書きできる。2つの出力ベクトルを持つ関数の場合、出力ベクトルの一方は、入力ベクトルと一致させてもよい。

## ベクトル・インデックス方式

現在の VML 数学関数は、単位増分インデックスでのみ機能する。他の増分を使用する配列、またはより複雑なインデックスを使用する配列を関数に適合させるには、成分を収集して連続する単位増分インデックスのベクトルを生成し、計算の完了後、それらを分散すればよい。

VML では、ベクトル成分の収集 / 分散に以下のインデックス方式を使用する。

- 正増分
- インデックス・ベクトル
- マスクベクトル

関数で使用するインデックス方式は、インデックス修飾子で指定する ([関数命名規則](#)の `<mod>` フィールドの説明を参照)。インデックス方式の詳細は、付録 B の [VML のベクトル引数](#)を参照のこと。

## エラー診断

VML ライブラリは、独自のエラーハンドラを備えている。C インターフェイスと Fortran インターフェイスでは、異なる点が 1 つだけある。それは、Fortran インターフェイスでは、VML 関数がエラーを検出した後で、インテル MKL エラー報告ルーチン [xerbla](#) の呼び出しが可能なことである。このルーチンは、`VML_STATUS_BADSIZE` と `VML_STATUS_BADMEM` の入力エラーに関する情報を取得する ([表 9-13](#)を参照)。

VML エラーハンドラは、以下のような機能を持つ。

1. エラー・ステータス (`vmlErrStatus`) グローバル変数が、VML 関数をそれぞれ呼び出した後で設定される。この変数が取り得る値については、[表 9-13](#)に示している。
2. VML モードに応じて、エラーハンドラ関数は以下を呼び出す。



- `errno` 変数設定。`errno` が取り得る値をに示す。[表 9-1](#)
- `stderr` ストリームへのエラーテキスト情報の書き込み。
- エラー時の適切な例外処理の起動 ( 必要な場合 )
- 追加のエラー・ハンドラ・コールバック関数の呼び出し

表 9-1      `errno` 変数の値の設定

errno の値	説明
0	検出されたエラーはない。
EINVAL	配列の次元が正でない。
EACCES	NULL ポインタが渡された。
EDOM	配列の値の少なくとも 1 つが、定義の範囲外。
ERANGE	配列の値の少なくとも 1 つが、特異点、オーバーフロー、またはアンダーフローを引き起こした。

VML 数学関数

このセクションでは、単位増分インデックスを持つ実数のベクトル引数に基づいて基本数学関数の値を計算する VML 関数について説明する。

それぞれの関数グループの説明では、短縮名と機能の簡単な説明を記載するとともに、Fortran インターフェイスと C インターフェイスの両方に対応した各データ型の呼び出しシーケンスを紹介し、入力 / 出力引数の説明も記載している。

すべての VML 数学関数において、パラメータの入力範囲は、それぞれのデータ型に定められた一連の値を用いて定義した数学的な範囲に等しい。VML 関数の一部、具体的には、`Div`、`Exp`、`Sinh`、`Cosh`、`Pow` では、結果がオーバーフローする場合がある。これらの関数のそれぞれについて、オーバーフローとなるかどうかを振り分ける入力しきい値は、各関数の説明のセクションで指定する。その指定の際、`FLT_MAX` は単精度データ型で表現可能な最大数を表し、`DBL_MAX` は倍精度データ型で表現可能な最大数を表す。

[表 9-2](#) は、使用可能な数学関数とそれに関連するデータタイプをまとめたものである。

表 9-2      VML 数学関数

関数の種類	データ型	説明
累乗関数と累乗根関数		
<a href="#">Inv</a>	s, d	ベクトル成分の逆数
<a href="#">Div</a>	s, d	1 つのベクトルの成分を 2 番目のベクトルの成分で割る

**表 9-2 VML 数学関数 ( 続き )**

関数の種類	データ型	説明
<a href="#">Sqrt</a>	s, d	ベクトル成分の平方根
<a href="#">InvSqrt</a>	s, d	ベクトル成分の平方根の逆数
<a href="#">Cbrt</a>	s, d	ベクトル成分の立方根
<a href="#">InvCbrt</a>	s, d	ベクトル成分の立方根の逆数
<a href="#">Pow</a>	s, d	各ベクトル成分を指定された値で累乗する
<a href="#">Powx</a>	s, d	各ベクトル成分を定数で累乗する
<b>指数関数と対数関数</b>		
<a href="#">Exp</a>	s, d	ベクトル成分の指数
<a href="#">Ln</a>	s, d	ベクトル成分の自然対数
<a href="#">Log10</a>	s, d	ベクトル成分の 10 進対数
<b>三角関数</b>		
<a href="#">Cos</a>	s, d	ベクトル成分の余弦
<a href="#">Sin</a>	s, d	ベクトル成分の正弦
<a href="#">SinCos</a>	s, d	ベクトル成分の正弦および余弦
<a href="#">Tan</a>	s, d	ベクトル成分の正接
<a href="#">Acos</a>	s, d	ベクトル成分の逆余弦
<a href="#">Asin</a>	s, d	ベクトル成分の逆正弦
<a href="#">Atan</a>	s, d	ベクトル成分の逆正接
<a href="#">Atan2</a>	s, d	2 つのベクトルの成分の 4 象限逆正接
<b>双曲線関数</b>		
<a href="#">Cosh</a>	s, d	ベクトル成分の双曲余弦
<a href="#">Sinh</a>	s, d	ベクトル成分の双曲正弦
<a href="#">Tanh</a>	s, d	ベクトル成分の双曲正接
<a href="#">Acosh</a>	s, d	ベクトル成分の逆双曲余弦 ( 非負 )
<a href="#">Asinh</a>	s, d	ベクトル成分の逆双曲正弦
<a href="#">Atanh</a>	s, d	ベクトル成分の逆双曲正接を計算する。
<b>特殊関数</b>		
<a href="#">Erf</a>	s, d	ベクトル成分の誤差関数値
<a href="#">Erfc</a>	s, d	ベクトル成分の相補誤差関数

## Inv

ベクトルの成分単位で逆数を計算する。

### 構文

#### Fortran:

```
call vsinv( n, a, y )
call vdiv( n, a, y )
```

#### C:

```
vsInv( n, a, y );
vdInv( n, a, y );
```

### 入力パラメータ

名称	型		説明
	FORTRAN	C	
<i>n</i>	INTEGER, INTENT(IN)	int	計算する成分の数
<i>a</i>	REAL, INTENT(IN) (vsinv の場合 ) DOUBLE PRECISION, INTENT(IN) (vdiv の場合 )	const float* (vsInv の場合 ) const double* (vdInv の場合 )	Fortran: 入力ベクトル <i>a</i> を指定する配列 C: 入力ベクトル <i>a</i> を含む配列へのポインタ

### 出力パラメータ

名称	型		説明
	FORTRAN	C	
<i>y</i>	REAL (vsinv の場合 ) DOUBLE PRECISION (vdiv の場合 )	float* (vsInv の場合 ) double* (vdInv の場合 )	Fortran: 出力ベクトル <i>y</i> を指定する配列 C: 出力ベクトル <i>y</i> を含む配列へのポインタ

Div

ベクトル *a* をベクトル *b* で対応成分ごとに割る。

構文

**Fortran:**

```
call vsdiv( n, a, b, y )
call vddiv( n, a, b, y )
```

**C:**

```
vsDiv( n, a, b, y );
vdDiv( n, a, b, y );
```

入力パラメータ

名称	型		説明
	FORTTRAN	C	
<i>n</i>	INTEGER, INTENT(IN)	int	計算する成分の数
<i>a</i> , <i>b</i>	REAL, INTENT(IN) (vsdiv の場合 ) DOUBLE PRECISION, INTENT(IN) (vddiv の場合 )	const float* (vsDiv の場合 ) const double* (vdDiv の場合 )	<i>Fortran</i> : 入力ベクトル <i>a</i> と <i>b</i> を指定する配列 <i>C</i> : 入力ベクトル <i>a</i> と <i>b</i> を含む配列へのポインタ

表 9-3 Div 関数における各精度でのオーバーフローしきい値

データ型	入力パラメータに対するしきい値制限
単精度	$\text{abs}(a[i]) < \text{abs}(b[i]) * \text{FLT\_MAX}$
倍精度	$\text{abs}(a[i]) < \text{abs}(b[i]) * \text{DBL\_MAX}$

## 出力パラメータ

名称	型		説明
	FORTRAN	C	
$y$	REAL (vsdiv の場合 ) DOUBLE PRECISION (vddiv の場合 )	float* (vsDiv の場合 ) double* (vdDiv の場合 )	Fortran: 出力ベクトル $y$ を指定する配列 C: 出力ベクトル $y$ を含む配列へのポインタ

## Sqrt

ベクトル成分の平方根を計算する。

## 構文

## Fortran:

```
call vssqrt( n, a, y )  
call vdsqrt( n, a, y )
```

## C:

```
vsSqrt( n, a, y );  
vdSqrt( n, a, y );
```

## 入力パラメータ

名称	型		説明
	FORTRAN	C	
$n$	INTEGER, INTENT(IN)	int	計算する成分の数
$a$	REAL, INTENT(IN) (vssqrt の場合 ) DOUBLE PRECISION, INTENT(IN) (vdsqrt の場合 )	const float* (vsSqrt の場合 ) const double* (vdSqrt の場合 )	Fortran: 入力ベクトル $a$ を指定する配列 C: 入力ベクトル $a$ を含む配列へのポインタ

## 出力パラメータ

名称	型		説明
	FORTRAN	C	
<i>y</i>	REAL ( <i>vssqrt</i> の場合 ) DOUBLE PRECISION ( <i>vdsqrt</i> の場合 )	float* ( <i>vsSqrt</i> の場合 ) double* ( <i>vdSqrt</i> の場合 )	<i>Fortran</i> : 出力ベクトル <i>y</i> を指定する配列 C: 出力ベクトル <i>y</i> を含む配列へのポインタ

## InvSqrt

ベクトル成分の平方根の逆数を計算する。

### 構文

#### Fortran:

```
call vsinvsqrt( n, a, y )
call vdinvsqrt( n, a, y )
```

#### C:

```
vsInvSqrt( n, a, y );
vdInvSqrt( n, a, y );
```

## 入力パラメータ

名称	型		説明
	FORTRAN	C	
<i>n</i>	INTEGER, INTENT(IN)	int	計算する成分の数
<i>a</i>	REAL, INTENT(IN) ( <i>vsinvsqrt</i> の場合 ) DOUBLE PRECISION, INTENT(IN) ( <i>vdinvsqrt</i> の場合 )	const float* ( <i>vsInvSqrt</i> の場合 ) const double* ( <i>vdInvSqrt</i> の場合 )	<i>Fortran</i> : 入力ベクトル <i>a</i> を指定する配列 C: 入力ベクトル <i>a</i> を含む配列へのポインタ

## 出力パラメータ

名称	型		説明
	FORTTRAN	C	
$y$	REAL (vsinvsqrt の場合) DOUBLE PRECISION (vdinvsqrt の場合)	float* (vsInvSqrt の場合) double* (vdInvSqrt の場合)	Fortran: 出力ベクトル $y$ を指定する配列 C: 出力ベクトル $y$ を含む配列へのポインタ

## Cbirt

ベクトル成分の立方根を計算する。

## 構文

## Fortran:

```
call vscbirt( n, a, y )
call vdcbirt( n, a, y )
```

## C:

```
vsCbirt( n, a, y );
vdCbirt( n, a, y );
```

## 入力パラメータ

名称	型		説明
	FORTTRAN	C	
$n$	INTEGER, INTENT(IN)	int	計算する成分の数
$a$	REAL, INTENT(IN) (vscbirt の場合) DOUBLE PRECISION, INTENT(IN) (vdcbirt の場合)	const float* (vsCbirt の場合) const double* (vdCbirt の場合)	Fortran: 入力ベクトル $a$ を指定する配列 C: 入力ベクトル $a$ を含む配列へのポインタ

## 出力パラメータ

名称	型		説明
	FORTRAN	C	
<i>y</i>	REAL ( <i>vsqrt</i> の場合) DOUBLE PRECISION ( <i>vdsqrt</i> の場合)	float* ( <i>vsqrt</i> の場合) double* ( <i>vdsqrt</i> の場合)	<i>Fortran</i> : 出力ベクトル <i>y</i> を指定する配列 C: 出力ベクトル <i>y</i> を含む配列へのポインタ

## InvCbrt

ベクトル成分の立方根の逆数を計算する。

### 構文

#### Fortran:

```
call vsinvcbirt( n, a, y )
call vdivinvcbirt( n, a, y )
```

#### C:

```
vsInvCbirt( n, a, y );
vdInvCbirt( n, a, y );
```

## 入力パラメータ

名称	型		説明
	FORTRAN	C	
<i>n</i>	INTEGER, INTENT(IN)	int	計算する成分の数
<i>a</i>	REAL, INTENT(IN) ( <i>vsinvcbirt</i> の場合) DOUBLE PRECISION, INTENT(IN) ( <i>vdivinvcbirt</i> の場合)	const float* ( <i>vsInvCbirt</i> の場合) const double* ( <i>vdInvCbirt</i> の場合)	<i>Fortran</i> : 入力ベクトル <i>a</i> を指定する配列 C: 入力ベクトル <i>a</i> を含む配列へのポインタ



## 出力パラメータ

名称	型		説明
	FORTTRAN	C	
$y$	REAL (vsinvcbrt の場合) DOUBLE PRECISION (vdinvcbrt の場合)	float* (vsInvCbrt の場合) double* (vdInvCbrt の場合)	Fortran: 出力ベクトル $y$ を指定する配列 C: 出力ベクトル $y$ を含む配列へのポインタ

## Pow

2 つのベクトルの成分について、 $a$  の  $b$  乗を計算する。

## 構文

## Fortran:

```
call vspow( n, a, b, y )
call vdpow( n, a, b, y )
```

## C:

```
vsPow( n, a, b, y );
vdPow( n, a, b, y );
```

## 入力パラメータ

名称	型		説明
	FORTTRAN	C	
$n$	INTEGER, INTENT(IN)	int	計算する成分の数
$a, b$	REAL, INTENT(IN) (vspow の場合) DOUBLE PRECISION, INTENT(IN) (vdpow の場合)	const float* (vsPow の場合) const double* (vdPow の場合)	Fortran: 入力ベクトル $a$ と $b$ を指定する配列 C: 入力ベクトル $a$ と $b$ を含む配列へのポインタ

表 9-4 Pow 関数における各精度でのオーバーフローしきい値

データ型	入力パラメータに対するしきい値制限
単精度	$\text{abs}(a[i]) < (\text{FLT\_MAX})^{1/b[i]}$
倍精度	$\text{abs}(a[i]) < (\text{DBL\_MAX})^{1/b[i]}$

出力パラメータ

名称	型		説明
	FORTRAN	C	
<i>y</i>	REAL (vspow の場合) DOUBLE PRECISION (vdpow の場合)	float* (vsPow の場合) double* (vdPow の場合)	Fortran: 出力ベクトル <i>y</i> を指定する配列 C: 出力ベクトル <i>y</i> を含む配列へのポインタ

説明

関数 Pow には、パラメータ *a* と *b* の入力範囲について、一定の制限がある。具体的には、*a*[*i*] が正の場合、*b*[*i*] には任意の値を指定できる。しかし、*a*[*i*] が負またはゼロの場合、*b*[*i*] の値は整数 ( 正または負 ) でなければならない。

Powx

各ベクトル成分を定数で累乗する。

構文

Fortran:

```
call vspowx( n, a, b, y )
call vdpowx( n, a, b, y )
```

C:

```
vsPowx( n, a, b, y );
vdPowx( n, a, b, y );
```

入力パラメータ

名称	型		説明
	FORTTRAN	C	
<i>n</i>	INTEGER, INTENT(IN)	int	計算する成分の数
<i>a</i>	REAL, INTENT(IN) (vspowx の場合) DOUBLE PRECISION, INTENT(IN) (vdpowx の場合)	const float* (vsPowx の場合) const double* (vdPowx の場合)	Fortran: 入力ベクトル <i>a</i> を指定する配列 C: 入力ベクトル <i>a</i> を含む配列へのポインタ
<i>b</i>	REAL, INTENT(IN) (vspowx の場合) DOUBLE PRECISION, INTENT(IN) (vdpowx の場合)	const float* (vsPowx の場合) const double* (vdPowx の場合)	Fortran: 定数の冪数として用いるスカラー値 <i>b</i> C: 冪数として用いる定数 <i>b</i>

表 9-5 Powx 関数における各精度でのオーバーフローしきい値

データ型	入力パラメータに対するしきい値制限
単精度	$\text{abs}(a[i]) < (\text{FLT\_MAX})^{1/b}$
倍精度	$\text{abs}(a[i]) < (\text{DBL\_MAX})^{1/b}$

出力パラメータ

名称	型		説明
	FORTTRAN	C	
<i>y</i>	REAL (vspowx の場合) DOUBLE PRECISION (vdpowx の場合)	float* (vsPowx の場合) double* (vdPowx の場合)	Fortran: 出力ベクトル <i>y</i> を指定する配列 C: 出力ベクトル <i>y</i> を含む配列へのポインタ

説明

関数 Powx には、パラメータ *a* および *b* の入力範囲について、一定の制限がある。具体的には、*a*[*i*] が正の場合、*b*[*i*] には任意の値を指定できる。しかし、*a*[*i*] が負またはゼロの場合、*b*[*i*] の値は整数 (正または負) でなければならない。

Exp

ベクトル成分の指数を計算する。

構文

**Fortran:**

```
call vsexp( n, a, y )
call vdexp( n, a, y )
```

**C:**

```
vsExp( n, a, y );
vdExp( n, a, y );
```

入力パラメータ

名称	型		説明
	FORTRAN	C	
<i>n</i>	INTEGER, INTENT(IN)	int	計算する成分の数
<i>a</i>	REAL, INTENT(IN) (vsexp の場合 ) DOUBLE PRECISION, INTENT(IN) (vdexp の場合 )	const float* (vsExp の場合 ) const double* (vdExp の場合 )	Fortran: 入力ベクトル <i>a</i> を指定する配列 C: 入力ベクトル <i>a</i> を含む配列へのポインタ

表 9-6 Exp 関数における各精度でのオーバーフローしきい値

データ型	入力パラメータに対するしきい値制限
単精度	$a[i] < \ln(FLT\_MAX)$
倍精度	$a[i] < \ln(DBL\_MAX)$

## 出力パラメータ

名称	型		説明
	FORTRAN	C	
$y$	REAL (vsexp の場合) DOUBLE PRECISION (vdexp の場合)	float* (vsExp の場合) double* (vdExp の場合)	Fortran: 出力ベクトル $y$ を指定する配列 C: 出力ベクトル $y$ を含む配列へのポインタ

## Ln

ベクトル成分の自然対数を計算する。

## 構文

## Fortran:

```
call vsln( n, a, y )
call vdln( n, a, y )
```

## C:

```
vsLn( n, a, y );
vdLn( n, a, y );
```

## 入力パラメータ

名称	型		説明
	FORTRAN	C	
$n$	INTEGER, INTENT(IN)	int	計算する成分の数
$a$	REAL, INTENT(IN) (vsln の場合) DOUBLE PRECISION, INTENT(IN) (vdln の場合)	const float* (vsLn の場合) const double* (vdLn の場合)	Fortran: 入力ベクトル $a$ を指定する配列 C: 入力ベクトル $a$ を含む配列へのポインタ

## 出力パラメータ

名称	型		説明
	FORTRAN	C	
$y$	REAL (vsln の場合) DOUBLE PRECISION (vdlN の場合)	float* (vsLn の場合) double* (vdLn の場合)	Fortran: 出力ベクトル $y$ を指定する配列 C: 出力ベクトル $y$ を含む配列へのポインタ

## Log10

ベクトル成分の 10 進対数を計算する。

### 構文

#### Fortran:

```
call vslog10( n, a, y )
call vdlog10( n, a, y )
```

#### C:

```
vsLog10( n, a, y );
vdLog10( n, a, y );
```

## 入力パラメータ

名称	型		説明
	FORTRAN	C	
$n$	INTEGER, INTENT(IN)	int	計算する成分の数
$a$	REAL, INTENT(IN) (vslog10 の場合) DOUBLE PRECISION, INTENT(IN) (vdlog10 の場合)	const float* (vsLog10 の場合) const double* (vdLog10 の場合)	Fortran: 入力ベクトル $a$ を指定する配列 C: 入力ベクトル $a$ を含む配列へのポインタ

## 出力パラメータ

名称	型		説明
	FORTRAN	C	
$y$	REAL (vslog10 の場合) DOUBLE PRECISION (vdlog10 の場合)	float* (vsLog10 の場合) double* (vdLog10 の場合)	Fortran: 出力ベクトル $y$ を指定する配列 C: 出力ベクトル $y$ を含む配列へのポインタ

## Cos

ベクトル成分の余弦を計算する。

## 構文

## Fortran:

```
call vscol ( n, a, y )
call vdcos ( n, a, y )
```

## C:

```
vsCos ( n, a, y );
vdCos ( n, a, y );
```

## 入力パラメータ

名称	型		説明
	FORTRAN	C	
$n$	INTEGER, INTENT(IN)	int	計算する成分の数
$a$	REAL, INTENT(IN) (vscos の場合) DOUBLE PRECISION, INTENT(IN) (vdcos の場合)	const float* (vsCos の場合) const double* (vdCos の場合)	Fortran: 入力ベクトル $a$ を指定する配列 C: 入力ベクトル $a$ を含む配列へのポインタ

## 出力パラメータ

名称	型		説明
	FORTRAN	C	
$y$	REAL (vscos の場合) DOUBLE PRECISION (vdcos の場合)	float* (vsCos の場合) double* (vdCos の場合)	Fortran: 出力ベクトル $y$ を指定する配列 C: 出力ベクトル $y$ を含む配列へのポインタ

## Sin

ベクトル成分の正弦を計算する。

### 構文

#### Fortran:

```
call vssin( n, a, y )
call vdsin( n, a, y )
```

#### C:

```
vsSin( n, a, y );
vdSin( n, a, y );
```

## 入力パラメータ

名称	型		説明
	FORTRAN	C	
$n$	INTEGER, INTENT(IN)	int	計算する成分の数
$a$	REAL, INTENT(IN) (vssin の場合) DOUBLE PRECISION, INTENT(IN) (vdsin の場合)	const float* (vsSin の場合) const double* (vdSin の場合)	Fortran: 入力ベクトル $a$ を指定する配列 C: 入力ベクトル $a$ を含む配列へのポインタ



## 出力パラメータ

名称	型		説明
	FORTRAN	C	
$y$	REAL (vssin の場合) DOUBLE PRECISION (vdsin の場合)	float* (vsSin の場合) double* (vdSin の場合)	Fortran: 出力ベクトル $y$ を指定する配列 C: 出力ベクトル $y$ を含む配列へのポインタ

## SinCos

ベクトル成分の正弦および余弦を計算する。

## 構文

## Fortran:

```
call vssincos( n, a, y, z )
call vdsincos( n, a, y, z )
```

## C:

```
vsSinCos( n, a, y, z );
vdSinCos( n, a, y, z );
```

## 入力パラメータ

名称	型		説明
	FORTRAN	C	
$n$	INTEGER, INTENT(IN)	int	計算する成分の数
$a$	REAL, INTENT(IN) (vssincos の場合) DOUBLE PRECISION, INTENT(IN) (vdsincos の場合)	const float* (vsSinCos の場合) const double* (vdSinCos の場合)	Fortran: 入力ベクトル $a$ を指定する配列 C: 入力ベクトル $a$ を含む配列へのポインタ

## 出力パラメータ

名称	型		説明
	FORTTRAN	C	
$y, z$	REAL (vssincos の場合) DOUBLE PRECISION (vdsincos の場合)	float* (vsSinCos の場合) double* (vdSinCos の場合)	<i>Fortran</i> : 正弦値出力ベクトル $y$ と余弦値出力ベクトル $z$ を指定する配列 <i>C</i> : 出力ベクトル $y$ (正弦値) と出力ベクトル $z$ (余弦値) を格納する配列へのポインタ。

## Tan

ベクトル成分の正接を計算する。

### 構文

#### Fortran:

```
call vstan( n, a, y )
call vdtan( n, a, y )
```

#### C:

```
vsTan( n, a, y );
vdTan( n, a, y );
```

## 入力パラメータ

名称	型		説明
	FORTTRAN	C	
$n$	INTEGER, INTENT(IN)	int	計算する成分の数
$a$	REAL, INTENT(IN) (vstan の場合) DOUBLE PRECISION, INTENT(IN) (vdtan の場合)	const float* (vsTan の場合) const double* (vdTan の場合)	<i>Fortran</i> : 入力ベクトル $a$ を指定する配列 <i>C</i> : 入力ベクトル $a$ を含む配列へのポインタ

## 出力パラメータ

名称	型		説明
	FORTRAN	C	
$y$	REAL (vstan の場合) DOUBLE PRECISION (vdtan の場合)	float* (vsTan の場合) double* (vdTan の場合)	Fortran: 出力ベクトル $y$ を指定する配列 C: 出力ベクトル $y$ を含む配列へのポインタ

## Acos

ベクトル成分の逆余弦を計算する。

## 構文

## Fortran:

```
call vsacos( n, a, y )
call vdacos( n, a, y )
```

## C:

```
vsAcos( n, a, y );
vdAcos( n, a, y );
```

## 入力パラメータ

名称	型		説明
	FORTRAN	C	
$n$	INTEGER, INTENT(IN)	int	計算する成分の数
$a$	REAL, INTENT(IN) (vsacos の場合) DOUBLE PRECISION, INTENT(IN) (vdacos の場合)	const float* (vsAcos の場合) const double* (vdAcos の場合)	Fortran: 入力ベクトル $a$ を指定する配列 C: 入力ベクトル $a$ を含む配列へのポインタ

## 出力パラメータ

名称	型		説明
	FORTTRAN	C	
$y$	REAL (vsacos の場合) DOUBLE PRECISION (vdacos の場合)	float* (vsAcos の場合) double* (vdAcos の場合)	Fortran: 出力ベクトル $y$ を指定する配列 C: 出力ベクトル $y$ を含む配列へのポインタ

## Asin

ベクトル成分の逆正弦を計算する。

### 構文

#### Fortran:

```
call vsasin( n, a, y )
call vdasin( n, a, y )
```

#### C:

```
vsAsin( n, a, y );
vdAsin( n, a, y );
```

## 入力パラメータ

名称	型		説明
	FORTTRAN	C	
$n$	INTEGER, INTENT(IN)	int	計算する成分の数
$a$	REAL, INTENT(IN) (vsasin の場合) DOUBLE PRECISION, INTENT(IN) (vdasin の場合)	const float* (vsAsin の場合) const double* (vdAsin の場合)	Fortran: 入力ベクトル $a$ を指定する配列 C: 入力ベクトル $a$ を含む配列へのポインタ

## 出力パラメータ

名称	型		説明
	FORTRAN	C	
$y$	REAL (vsasin の場合) DOUBLE PRECISION (vdasin の場合)	float* (vsAsin の場合) double* (vdAsin の場合)	Fortran: 出力ベクトル $y$ を指定する配列 C: 出力ベクトル $y$ を含む配列へのポインタ

## Atan

ベクトル成分の逆正接を計算する。

## 構文

## Fortran:

```
call vsatan( n, a, y )
call vdatan( n, a, y )
```

## C:

```
vsAtan( n, a, y );
vdAtan( n, a, y );
```

## 入力パラメータ

名称	型		説明
	FORTRAN	C	
$n$	INTEGER, INTENT(IN)	int	計算する成分の数
$a$	REAL, INTENT(IN) (vsatan の場合) DOUBLE PRECISION, INTENT(IN) (vdatan の場合)	const float* (vsAtan の場合) const double* (vdAtan の場合)	Fortran: 入力ベクトル $a$ を指定する配列 C: 入力ベクトル $a$ を含む配列へのポインタ

## 出力パラメータ

名称	型		説明
	FORTTRAN	C	
$y$	REAL (vsatan の場合) DOUBLE PRECISION (vdatan の場合)	float* (vsAtan の場合) double* (vdAtan の場合)	Fortran: 出力ベクトル $y$ を指定する配列 C: 出力ベクトル $y$ を含む配列へのポインタ

## Atan2

2 つのベクトルの成分の 4 象限逆正接を計算する。

### 構文

**Fortran:**

```
call vsatan2( n, a, b, y )
call vdatan2( n, a, b, y )
```

**C:**

```
vsAtan2( n, a, b, y );
vdAtan2( n, a, b, y );
```

### 入力パラメータ

名称	型		説明
	FORTTRAN	C	
$n$	INTEGER, INTENT(IN)	int	計算する成分の数
$a, b$	REAL, INTENT(IN) (vsatan2 の場合) DOUBLE PRECISION, INTENT(IN) (vdatan2 の場合)	const float* (vsAtan2 の場合) const double* (vdAtan2 の場合)	Fortran: 入力ベクトル $a$ と $b$ を指定する配列 C: 入力ベクトル $a$ と $b$ を含む配列へのポインタ

出力パラメータ

名称	型		説明
	FORTRAN	C	
$y$	REAL (vsatan2 の場合 ) DOUBLE PRECISION (vdatan2 の場合 )	float* (vsAtan2 の場合 ) double* (vdAtan2 の場合 )	Fortran: 出力ベクトル $y$ を指定する配列 C: 出力ベクトル $y$ を含む配列へのポインタ

説明

出力ベクトル  $y$  の各成分は、 $a[i] / b[i]$  の逆正接として計算される。入力の数値の組み合わせにより出力は 4 象限にわたる。

Cosh

ベクトル成分の双曲余弦を計算する。

構文

Fortran:

```
call vscosh( n, a, y )  
call vdcosh( n, a, y )
```

C:

```
vsCosh( n, a, y );  
vdCosh( n, a, y );
```

入力パラメータ

名称	型		説明
	FORTRAN	C	
$n$	INTEGER, INTENT(IN)	int	計算する成分の数

a	REAL, INTENT(IN) (vscosh の場合 ) DOUBLE PRECISION, INTENT(IN) (vdcosh の場合 )	const float* (vsCosh の場合 ) const double* (vdCosh の場合 )	Fortran: 入力ベクトル a を指定する配 列 C: 入力ベクトル a を含む配列へのポイ ンタ
---	---	---	--

表 9-7 Cosh 関数における各精度でのオーバーフローしきい値

データ型	入力パラメータに対するしきい値制限
単精度	$-\ln(\text{FLT\_MAX}) - \ln 2 < a[i] < \ln(\text{FLT\_MAX}) + \ln 2$
倍精度	$-\ln(\text{DBL\_MAX}) - \ln 2 < a[i] < \ln(\text{DBL\_MAX}) + \ln 2$

出力パラメータ

名称	型		説明
	FORTRAN	C	
y	REAL (vscosh の場合 ) DOUBLE PRECISION (vdcosh の場合 )	float* (vsCosh の場合 ) double* (vdCosh の場合 )	Fortran: 出力ベクトル y を指定する配 列 C: 出力ベクトル y を含む配列へのポイ ンタ

Sinh

ベクトル成分の双曲正弦を計算する。

構文

Fortran:

```
call vssinh( n, a, y )  
call vdsinh( n, a, y )
```

C:

```
vsSinh( n, a, y );  
vdSinh( n, a, y );
```



入力パラメータ

名称	型		説明
	FORTTRAN	C	
<i>n</i>	INTEGER, INTENT(IN)	int	計算する成分の数
<i>a</i>	REAL, INTENT(IN) (vssinh の場合 ) DOUBLE PRECISION, INTENT(IN) (vdsinh の場合 )	const float* (vsSinh の場合 ) const double* (vdSinh の場合 )	Fortran: 入力ベクトル <i>a</i> を指定する配列 C: 入力ベクトル <i>a</i> を含む配列へのポインタ

表 9-8      sinh 関数における各精度でのオーバーフローしきい値

データ型	入力パラメータに対するしきい値制限
単精度	$-\text{Ln}(\text{FLT\_MAX}) - \text{Ln}2 < a[i] < \text{Ln}(\text{FLT\_MAX}) + \text{Ln}2$
倍精度	$-\text{Ln}(\text{DBL\_MAX}) - \text{Ln}2 < a[i] < \text{Ln}(\text{DBL\_MAX}) + \text{Ln}2$

出力パラメータ

名称	型		説明
	FORTTRAN	C	
<i>y</i>	REAL (vssinh の場合 ) DOUBLE PRECISION (vdsinh の場合 )	float* (vsSinh の場合 ) double* (vdSinh の場合 )	Fortran: 出力ベクトル <i>y</i> を指定する配列 C: 出力ベクトル <i>y</i> を含む配列へのポインタ

Tanh

ベクトル成分の双曲正接を計算する。

構文

Fortran:

```
call vstanh( n, a, y )  
call vdtanh( n, a, y )
```

**C:**

```
vsTanh( n, a, y );
```

```
vdTanh( n, a, y );
```

## 入力パラメータ

名称	型		説明
	FORTRAN	C	
<i>n</i>	INTEGER, INTENT(IN)	int	計算する成分の数
<i>a</i>	REAL, INTENT(IN) (vstanh の場合 ) DOUBLE PRECISION, INTENT(IN) (vdtanh の場合 )	const float* (vsTanh の場合 ) const double* (vdTanh の場合 )	<i>Fortran</i> : 入力ベクトル <i>a</i> を指定する配列 <i>C</i> : 入力ベクトル <i>a</i> を含む配列へのポインタ

## 出力パラメータ

名称	型		説明
	FORTRAN	C	
<i>y</i>	REAL (vstanh の場合 ) DOUBLE PRECISION (vdtanh の場合 )	float* (vsTanh の場合 ) double* (vdTanh の場合 )	<i>Fortran</i> : 出力ベクトル <i>y</i> を指定する配列 <i>C</i> : 出力ベクトル <i>y</i> を含む配列へのポインタ

## Acosh

ベクトル成分の逆双曲余弦( 非負 ) を計算する。

### 構文

**Fortran:**

```
call vsacosh( n, a, y )
```

```
call vdacosh( n, a, y )
```

**C:**

```
vsAcosh( n, a, y );
```

```
vdAcosh( n, a, y );
```

## 入力パラメータ

名称	型		説明
	FORTTRAN	C	
$n$	INTEGER, INTENT(IN)	int	計算する成分の数
$a$	REAL, INTENT(IN) (vsacosh の場合 ) DOUBLE PRECISION, INTENT(IN) (vdacosh の場合 )	const float* (vsAcosh の場合 ) const double* (vdAcosh の場合 )	Fortran: 入力ベクトル $a$ を指定する配列 C: 入力ベクトル $a$ を含む配列へのポインタ

## 出力パラメータ

名称	型		説明
	FORTTRAN	C	
$y$	REAL (vsacosh の場合 ) DOUBLE PRECISION (vdacosh の場合 )	float* (vsAcosh の場合 ) double* (vdAcosh の場合 )	Fortran: 出力ベクトル $y$ を指定する配列 C: 出力ベクトル $y$ を含む配列へのポインタ

## Asinh

ベクトル成分の逆双曲正弦を計算する。

### 構文

#### Fortran:

```
call vsasinh( n, a, y )
call vdasinh( n, a, y )
```

#### C:

```
vsAsinh( n, a, y );
vdAsinh( n, a, y );
```

## 入力パラメータ

名称	型		説明
	FORTRAN	C	
<i>n</i>	INTEGER, INTENT(IN)	int	計算する成分の数
<i>a</i>	REAL, INTENT(IN) (vsasinh の場合 ) DOUBLE PRECISION, INTENT(IN) (vdasinh の場合 )	const float* (vsAsinh の場合 ) const double* (vdAsinh の場合 )	<i>Fortran</i> : 入力ベクトル <i>a</i> を指定する配列 <i>C</i> : 入力ベクトル <i>a</i> を含む配列へのポインタ

## 出力パラメータ

名称	型		説明
	FORTRAN	C	
<i>y</i>	REAL (vsasinh の場合 ) DOUBLE PRECISION (vdasinh の場合 )	float* (vsAsinh の場合 ) double* (vdAsinh の場合 )	<i>Fortran</i> : 出力ベクトル <i>y</i> を指定する配列 <i>C</i> : 出力ベクトル <i>y</i> を含む配列へのポインタ

## Atanh

---

ベクトル成分の逆双曲正接を計算する。

---

### 構文

#### Fortran:

```
call vsatanh( n, a, y )
call vdatanh( n, a, y )
```

#### C:

```
vsAtanh( n, a, y );
vdAtanh( n, a, y );
```

## 入力パラメータ

名称	型		説明
	FORTRAN	C	
<i>n</i>	INTEGER, INTENT(IN)	int	計算する成分の数
<i>a</i>	REAL, INTENT(IN) (vsatanh の場合 ) DOUBLE PRECISION, INTENT(IN) (vdatanh の場合 )	const float* (vsAtanh の場合 ) const double* (vdAtanh の場合 )	Fortran: 入力ベクトル <i>a</i> を指定する配列 C: 入力ベクトル <i>a</i> を含む配列へのポインタ

## 出力パラメータ

名称	型		説明
	FORTRAN	C	
<i>y</i>	REAL (vsatanh の場合 ) DOUBLE PRECISION (vdatanh の場合 )	float* (vsAtanh の場合 ) double* (vdAtanh の場合 )	Fortran: 出力ベクトル <i>y</i> を指定する配列 C: 出力ベクトル <i>y</i> を含む配列へのポインタ

## Erf

ベクトル成分の誤差関数を計算する。

## 構文

## Fortran:

```
call vserf( n, a, y )
call vderf( n, a, y )
```

## C:

```
vsErf( n, a, y );
vdErf( n, a, y );
```

## 入力パラメータ

名称	型		説明
	FORTRAN	C	
$n$	INTEGER, INTENT(IN)	int	計算する成分の数
$a$	REAL, INTENT(IN) (vserf の場合 ) DOUBLE PRECISION, INTENT(IN) (vderf の場合 )	const float* (vsErf の場合 ) const double* (vdErf の場合 )	Fortran: 入力ベクトル $a$ を指定する配列 C: 入力ベクトル $a$ を含む配列へのポインタ

## 出力パラメータ

名称	型		説明
	FORTRAN	C	
$y$	REAL (vserf の場合 ) DOUBLE PRECISION (vderf の場合 )	float* (vsErf の場合 ) double* (vdErf の場合 )	Fortran: 出力ベクトル $y$ を指定する配列 C: 出力ベクトル $y$ を含む配列へのポインタ

## 説明

関数 Erf は、入力ベクトル  $a$  の成分に対して誤差関数を計算し、結果を出力ベクトル  $y$  に書き出す。

誤差関数は次のように定義される。

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

## Erfc

ベクトル成分の相補誤差関数を計算する。

### 構文

#### Fortran:

```
call vserfc( n, a, y )  
call vderfc( n, a, y )
```

#### C:

```
vsErfc( n, a, y );  
vdErfc( n, a, y );
```

### 入力パラメータ

名称	型		説明
	FORTRAN	C	
<i>n</i>	INTEGER, INTENT(IN)	int	計算する成分の数
<i>a</i>	REAL, INTENT(IN) (vserfc の場合 ) DOUBLE PRECISION, INTENT(IN) (vderfc の場合 )	const float* (vsErfc の場合 ) const double* (vdErfc の場合 )	Fortran: 入力ベクトル <i>a</i> を指定する配列 C: 入力ベクトル <i>a</i> を含む配列へのポインタ

### 出力パラメータ

名称	型		説明
	FORTRAN	C	
<i>y</i>	REAL (vserfc の場合 ) DOUBLE PRECISION (vdeefc の場合 )	float* (vsErfc の場合 ) double* (vdErfc の場合 )	Fortran: 出力ベクトル <i>y</i> を指定する配列 C: 出力ベクトル <i>y</i> を含む配列へのポインタ

## 説明

関数 `Erfc` は、入力ベクトル `a` の成分に対して相補誤差関数を計算し、結果を出力ベクトル `y` として書き出す。

誤差関数は次のように定義される。

$$\text{erfc}(x) = 1 - \text{erf}(x)$$

または書き換えると、

$$\text{erfc}(x) = \frac{2}{\sqrt{\pi}} \int_x^{\infty} e^{-t^2} dt$$

## VML Pack/Unpack 関数

このセクションでは、単位増分を使用するベクトルから正増分インデックス、ベクトル・インデックス、マスク・インデックスを使用するベクトルへの変換、およびその逆の変換を実行する VML 関数について説明する (ベクトル・インデックス方式の詳細は、[付録 B](#) を参照のこと)。

[表 9-9](#) は、使用可能な VML Pack/Unpack 関数とそれに関連するデータ型とインデックス方式をまとめたものである。

**表 9-9** VML Pack/Unpack 関数

関数の短縮名	データ型	インデックス方式	説明
<a href="#">Pack</a>	s, d	I, V, M	各種の方式でインデックス化されている配列の成分を収集する。
<a href="#">Unpack</a>	s, d	I, V, M	各種のインデックスによって配列にベクトル成分を分散する。



---

## Pack

指定されたインデックスを使用している配列の成分を、単位増分を使用するベクトルにコピーする。

---

### 構文

#### Fortran:

```
call vspacki( n, a, inca, y )  
call vspackv( n, a, ia, y )  
call vspackm( n, a, ma, y )  
call vdpacki( n, a, inca, y )  
call vdpackv( n, a, ia, y )  
call vdpackm( n, a, ma, y )
```

#### C:

```
vsPackI( n, a, inca, y );  
vsPackV( n, a, ia, y );  
vsPackM( n, a, ma, y );  
vdPackI( n, a, inca, y );  
vdPackV( n, a, ia, y );  
vdPackM( n, a, ma, y );
```

## 入力パラメータ

名称	型		説明
	FORTTRAN	C	
<i>n</i>	INTEGER, INTENT(IN)	int	計算する成分の数
<i>a</i>	REAL, INTENT(IN) (vspacki, vspackv, vspackm の場合) DOUBLE PRECISION, INTENT(IN) (vdpacki, vdpackv, vdpackm の場合)	const float* (vsPackI, vsPackV, vsPackM の場合) const double* (vdPackI, vdPackV, vdPackM の場合)	<i>Fortran</i> : 配列、DIMENSION (1 + (n-1)*inca) 以上 (vspacki/vdpacki の場合)、 max( n,max(ia[j]) ), j=0, ..., n-1 以上 (vspackv/vdpackv の場合)、 n 以上 (vspackm/vdpackm の場合)。 入力ベクトル <i>a</i> を指定する配列 C: 入力ベクトル <i>a</i> を含む配列へのポイン タ。配列のサイズは、以下でなければなら ない。 (1 + (n-1)*inca) 以上 (vsPackI/vdPackI の場合)、 max(n,max(ia[j])), j=0, ..., n-1 以 上 (vsPackV/vdPackV の場合)、 n 以上 (vsPackM/vdPackM の場合)。
<i>inca</i>	INTEGER, INTENT(IN) (vspacki, vdpacki の 場合)	int (vsPackI, vdPackI の場合)	<i>inca</i> 入力ベクトル <i>a</i> のインデックス増分定 数
<i>ia</i>	INTEGER, INTENT(IN) (vspackv, vdpackv の 場合)	const int* (vsPackV, vdPackV の場合)	<i>Fortran</i> : 配列、DIMENSION <i>n</i> 以上。 <i>a</i> の成分のためのインデックス・ベクトル を指定する。 C: <i>a</i> の成分のインデックス・ベクトルが格 納されたサイズが <i>n</i> 以上の配列へのポイン タ。
<i>ma</i>	INTEGER, INTENT(IN) (vspackm, vdpackm の 場合)	const int* (vsPackM, vdPackM の場合)	<i>Fortran</i> : 配列、DIMENSION <i>n</i> 以上。 <i>a</i> の成分のためのマスク・ベクトルを指定す る。 C: <i>a</i> の成分のマスク・ベクトルが格納され たサイズが <i>n</i> 以上の配列へのポインタ。

## 出力パラメータ

名称	型		説明
	FORTTRAN	C	
$y$	REAL (vspacki, vspackv, vspackm の場合) DOUBLE PRECISION (vdpacki, vdpackv, vdpackm の場合)	float* (vsPackI, vsPackV, vsPackM の場合) double* (vdPackI, vdPackV, vdPackM の場合)	Fortran: 配列、DIMENSION $n$ 以上。出力ベクトル $y$ を指定する。 C: 出力ベクトル $y$ を含むサイズが $n$ 以上の配列へのポインタ

## Unpack

単位増分を使用するベクトルの成分を、指定されたインデックスを使用した配列にコピーする。

## 構文

## Fortran:

```
call vsunpacki( n, a, y, incy )
call vsunpackv( n, a, y, iy )
call vsunpackm( n, a, y, my )
call vdunpacki( n, a, y, incy )
call vdunpackv( n, a, y, iy )
call vdunpackm( n, a, y, my )
```

## C:

```
vsUnpackI( n, a, y, incy );
vsUnpackV( n, a, y, iy );
vsUnpackM( n, a, y, my );
vdUnpackI( n, a, y, incy );
vdUnpackV( n, a, y, iy );
vdUnpackM( n, a, y, my );
```

## 入力パラメータ

名称	型		説明
	FORTRAN	C	
<i>n</i>	INTEGER, INTENT(IN)	int	計算する成分の数
<i>a</i>	REAL, INTENT(IN) (vsunpacki, vsunpackv, vsunpackm の場合) DOUBLE PRECISION, INTENT(IN) (vdunpacki, vdunpackv, vdunpackm の場合)	const float* (vsUnpackI, vsUnpackV, vsUnpack M の場合) const double* (vdUnpackI, vdUnpackV, vdUnpackM の場合)	<i>Fortran</i> : 配列、DIMENSION <i>n</i> 以上。 入力ベクトル <i>a</i> を指定する。 <i>C</i> : 入力ベクトル <i>a</i> を格納する配列へのポイン タ。
<i>incy</i>	INTEGER, INTENT(IN) (vsunpacki, vdunpacki の場合)	int (vsUnpackI, vdUnpackI の場合)	<i>y</i> の成分の増分を指定する。
<i>iy</i>	INTEGER, INTENT(IN) (vsunpackv, vdsunpackv の場合)	const int* (vsUnpackV, vdUnpackV の場合)	<i>Fortran</i> : 配列、DIMENSION <i>n</i> 以上。 <i>y</i> の成分のインデックス・ベクトル を指定する。 <i>C</i> : <i>y</i> の成分のインデックス・ベクトルを格 納するサイズが <i>n</i> 以上の配列へのポイン タ。
<i>my</i>	INTEGER, INTENT(IN) (vpackm, vdpackm の場 合)	const int* (vsUnpackM, vdUnpackM の場合)	<i>Fortran</i> : 配列、DIMENSION <i>n</i> 以上。 <i>y</i> の成分のためのマスク・ベクトル を指定する。 <i>C</i> : <i>y</i> の成分のマスク・ベクトルを格納する サイズが <i>n</i> 以上の配列へのポインタ。

出力パラメータ

名称	型		説明
	FORTRAN	C	
$y$	REAL (vsunpacki, vsunpackv, vsunpackm の場合 ) DOUBLE PRECISION (vdunpacki, vdunpackv, vdunpackm の場合 )	float* (vsUnpackI, vsUnpackV, vsUnpackM の場合 ) double* (vdUnpackI, vdUnpackV, vdUnpackM の場合 )	<i>Fortran</i> : 配列、DIMENSION (1 + (n-1)*incy) 以上 (vsunpacki/vdunpacki の場合 )、 max( n,max(iy[j])), j=0, ..., n-1 以上 (vsunpackv/vdunpackv の場合 )、 n 以上 (vsunpackm/vdunpackm の場合 )。 C: 出力ベクトル $y$ を格納する配列へのポインタ。配列のサイズは、以下でなければならない。 (1 + (n-1)*incy) 以上 (vsUnpackI/vdUnpackI の場合 )、 max( n,max(iy[j])), j=0, ..., n-1 以上 (vsUnpackV/vdUnpackV の場合 )、 n 以上 (vsUnpackM/vdUnpackM の場合 )。

VML サービス関数

このセクションでは精度モードの設定 / 取得、エラーコードの設定 / 取得を行う VML 関数について説明する。これらの関数はすべて、Fortran インターフェイスと C インターフェイスの両方で使用できる。

表 9-10 は、使用可能な VML サービス関数とそれぞれについての簡単な説明をまとめたものである。

表 9-10 VML サービス関数

関数の短縮名	説明
<a href="#">SetMode</a>	VML モードを設定する
<a href="#">GetMode</a>	VML モードを取得する
<a href="#">SetErrStatus</a>	VML エラー・ステータスを設定する
<a href="#">GetErrStatus</a>	VML エラー・ステータスを取得する
<a href="#">ClearErrStatus</a>	VML エラー・ステータスをクリアする
<a href="#">SetErrorCallBack</a>	追加のエラー・ハンドラ・コールバック関数を設定する
<a href="#">GetErrorCallBack</a>	追加のエラー・ハンドラ・コールバック関数を取得する
<a href="#">ClearErrorCallBack</a>	追加のエラー・ハンドラ・コールバック関数を削除する

## SetMode

*mode* パラメータに従って VML 関数の新しいモードを設定し、前の VML モードを *oldmode* に保存する。

### 構文

**Fortran:**

```
oldmode = vmlsetmode( mode )
```

**C:**

```
oldmode = vmlSetMode( mode );
```

### 入力パラメータ

名称	型		説明
	FORTRAN	C	
<i>mode</i>	INTEGER, INTENT(IN)	int	設定する VML モード

### 出力パラメータ

名称	型		説明
	FORTRAN	C	
<i>oldmode</i>	INTEGER	int	前の VML モード

### 説明

*mode* パラメータは、精度、FPU、エラー処理オプションを制御する。[表 9-11](#) は、*mode* パラメータの値をまとめたものである。*mode* パラメータに指定できるその他の値はすべて、これらの値から得られる。その方法は、ビット単位 OR (|) 演算を使用して、精度用の値、FPU 用の値、エラー制御オプション用の値を各 1 つずつ組み合わせるものである。*mode* パラメータのデフォルト値は、VML\_HA | VML\_ERRMODE\_DEFAULT である。したがって、デフォルトでは、現在の FPU 制御ワード (FPU 精度および丸め方式) が使用される。

異なる FPU 精度または丸め方式が必要な場合、VML 数学関数は、これらのオプションを自動的に変更し、前の値を復元する。*mode* パラメータを使用すると、同じ精度の設定で機能する各 VML 数学関数内の内部 FPU モードの切り替えを最小限に抑えられる。そのためには、*mode* パラメータに VML\_FLOAT\_CONSISTENT（単精度関数の場合）、または VML\_DOUBLE\_CONSISTENT（倍精度関数の場合）を設定する。それぞれの関数グループで、これらの *mode* パラメータの値は最適な選択となる。なぜなら、これらの値は、ほとんどの VML 数学関数で必要だからである。前の FPU モードを復元する必要がある場合は、実行の完了後、*mode* に VML\_RESTORE を設定する。

表 9-11 *mode* パラメータの値

<i>mode</i> の値	説明
精度の制御	
VML_HA	VML 関数の高精度モードを使用する
VML_LA	VML 関数の低精度モードを使用する
追加 FPU モードの制御	
VML_FLOAT_CONSISTENT	単精度関数に最適な FPU モード（制御ワード）を設定し、前の FPU モードを保存する
VML_DOUBLE_CONSISTENT	倍精度関数に最適な FPU モード（制御ワード）を設定し、前の FPU モードを保存する
VML_RESTORE	前に保存した FPU モードを復元する
エラーモードの制御	
VML_ERRMODE_IGNORE	計算エラーに対するアクションを設定しない
VML_ERRMODE_ERRNO	エラー時に <code>errno</code> 変数を設定する
VML_ERRMODE_STDERR	エラー時にエラーテキスト情報を <code>stderr</code> に書き込む
VML_ERRMODE_EXCEPT	エラー時に例外処理を起動する
VML_ERRMODE_CALLBACK	エラー時に追加のエラーハンドラ関数を呼び出す
VML_ERRMODE_DEFAULT	エラー時に <code>errno</code> 変数を設定し、例外処理を起動して、追加のエラーハンドラ関数を呼び出す

例

*mode* パラメータにさまざまな値を指定して関数 `vmSetMode()` を呼び出す例をいくつか以下に示す。

Fortran:

```
oldmode = vmlsetmode( VML_LA )
```

```
call vmlsetmode( IOR(VML_LA, IOR(VML_FLOAT_CONSISTENT,  
    VML_ERRMODE_IGNORE )))  
call vmlsetmode( VML_RESTORE)
```

**C:**

```
vmlSetMode( VML_LA );  
vmlSetMode( VML_LA | VML_FLOAT_CONSISTENT | VML_ERRMODE_IGNORE );  
vmlSetMode( VML_RESTORE);
```

---

## GetMode

VML モードを取得する。

---

### 構文

**Fortran:**

```
mod = vmlgetmode()
```

**C:**

```
mod = vmlGetMode( void );
```

### 出力パラメータ

名称	型		説明
	FORTTRAN	C	
<i>mod</i>	INTEGER	int	合成された <i>mode</i> パラメータ

### 説明

関数 `vmlGetMode()` は、精度、FPU、エラー処理オプションを制御する VML *mode* パラメータを戻す。*mod* 変数の値は、[表 9-11](#) で示した値の組み合わせで示される。[表 9-12](#) のそれぞれのマスクを使用すると、これらの値の一部を得られる。例を以下に示す。

**Fortran:**

```
mod = vmlgetmode()  
accm = IAND(mod, VML_ACCURACY_MASK)
```



```
fpum = IAND(mod, VML_FPUMODE_MASK)
errm = IAND(mod, VML_ERRMODE_MASK)

C:
accm = vmlGetMode(void )& VML_ACCURACY_MASK;
fpum = vmlGetMode(void )& VML_FPUMODE _MASK;
errm = vmlGetMode(void )& VML_ERRMODE _MASK;
```

表 9-12 *mode* パラメータに対するマスクの値

マスクの値	説明
VML_ACCURACY_MASK	精度 <i>mode</i> を選択する場合のマスクを指定する。
VML_FPUMODE_MASK	FPU <i>mode</i> を選択する場合のマスクを指定する。
VML_ERRMODE_MASK	エラー <i>mode</i> を選択する場合のマスクを指定する。

SetErrStatus

*err* に従ってVML エラー・ステータスを設定し、  
前のVML エラー・ステータスを*olderr* に保存する。

構文

Fortran:

```
olderr = vmlseterrstatus( err )
```

C:

```
olderr = vmlSetErrStatus( err );
```

入力パラメータ

名称	型		説明
	FORTRAN	C	
<i>err</i>	INTEGER, INTENT(IN)	int	設定する VML エラー・ステータス

## 出力パラメータ

名称	型		説明
	FORTRAN	C	
<i>olderr</i>	INTEGER	int	前の VML エラー・ステータス

表 9-13 は、*err* パラメータに指定可能な値をまとめたものである。

**表 9-13 VML エラー・ステータスの値**

エラー・ステータス	説明
VML_STATUS_OK	実行が正常に完了した。
VML_STATUS_BADSIZE	配列の次元が正でない。
VML_STATUS_BADMEM	NULL ポインタが渡された。
VML_STATUS_ERRDOM	配列の値の少なくとも 1 つが、定義の範囲外。
VML_STATUS_SING	配列の値の少なくとも 1 つが、特異点を引き起こした。
VML_STATUS_OVERFLOW	計算プロセスでオーバーフローが発生した。
VML_STATUS_UNDERFLOW	計算プロセスでアンダーフローが発生した。

### 例：

```

vmlSetErrStatus( VML_STATUS_OK );
vmlSetErrStatus( VML_STATUS_ERRDOM );
vmlSetErrStatus( VML_STATUS_UNDERFLOW );

```

## GetErrStatus

VML エラー・ステータスを取得する。

### 構文

#### Fortran:

```
err = vmlgeterrstatus( )
```

#### C:

```
err = vmlGetErrStatus( void );
```

## 出力パラメータ

名称	型		説明
	FORTRAN	C	
<i>err</i>	INTEGER	int	VML エラー・ステータス

---

## ClearErrStatus

VML エラー・ステータスに VML\_STATUS\_OK を設定し、前の VML エラー・ステータスを *olderr* に保存する。

---

## 構文

**Fortran:**

```
olderr = vmlclearerrstatus( )
```

**C:**

```
olderr = vmlClearErrStatus( void );
```

## 出力パラメータ

名称	型		説明
	FORTRAN	C	
<i>olderr</i>	INTEGER	int	前の VML エラー・ステータス

---

## SetErrorCallback

追加のエラー・ハンドラ・コールバック関数を設定し、古いコールバック関数を取得する。

---

## 構文

**Fortran:**

```
oldcallback = vmlseterrorcallback( callback )
```

**C:**

```
oldcallback = vmlSetErrorCallBack( callback );
```

## 入力パラメータ

**Fortran:**

*callback* コールバック関数のアドレス。  
 コールバック関数は、以下の形式で示される。

```
INTEGER FUNCTION ERRFUNC(par)
  TYPE (ERROR_STRUCTURE) par
  !...
!user error processing
  !...
ERRFUNC = 0
!if ERRFUNC = 0 - standard VML error handler
!is called after the callback
!if ERRFUNC != 0 - standard VML error handler
!is not called
END
```

渡されるエラー構造は、次のように定義される。

```
TYPE ERROR_STRUCTURE
  SEQUENCE
  INTEGER*4 ICODE
  INTEGER*4 IINDEX
  REAL*8 DBA1
  REAL*8 DBA2
  REAL*8 DBR1
  REAL*8 DBR2
  CHARACTER(64) CFUNCNAME
  INTEGER*4 IFUNCNAMELEN
END TYPE ERROR_STRUCTURE
```

**C:**

*callback* コールバック関数へのポインタ。  
 コールバック関数は、以下の形式で示される。

```
static int __stdcall MyHandler(DefVmlErrorContext*
pContext)
```

```

{
/* Handler body */
};

```

渡されるエラー構造は、次のように定義される。

```

typedef struct _DefVmlErrorContext
{
    int iCode; /* Error status
               value */
    int iIndex; /* Index for bad array
               element, or bad
               array dimension, or
               bad array pointer */
    double dbA1; /* Error argument 1 */
    double dbA2; /* Error argument 2 */
    double dbR1; /* Error result 1 */
    double dbR2; /* Error result 2 */
    char cFuncName[64]; /* Function name */
    int iFuncNameLen; /* Length of function
                     name */
} DefVmlErrorContext;

```

## 出力パラメータ

名称	型		説明
	FORTRAN	C	
<i>oldcallback</i>	INTEGER	int	<i>Fortran</i> : 前のコールバック関数のアドレス、 <i>C</i> : 前のコールバック関数へのポインタ。

## 説明

VML\_ERRMODE\_CALLBACK エラーモードが設定されている場合、VML 数学関数のエラーが発生すると、そのたびにコールバック関数が呼び出される ([表 9-11](#) を参照)。

デフォルトの空のコールバック関数ではなく、独自のコールバック関数を定義する必要がある場合は、`vmlSetErrorCallBack()` 関数を使用する。

コールバック関数の入力構造には、検出されたエラーに関して次の情報が含まれる。

- エラーを引き起こした入力値
- その値のロケーション (配列インデックス)

- 計算された結果値
- エラーコード
- エラーが発生した関数の名前

コールバック関数には、独自のエラー処理を組み込める。例えば、渡された結果値を訂正し、それを戻して計算を再開するような処理を組み込むことも可能である。コールバック関数の後に標準エラーハンドラが呼び出されるのは、0 が戻された場合だけである。

---

## GetErrorCallback

追加のエラー・ハンドラ・コールバック関数を取得する。

---

### 構文

#### Fortran:

```
fun = vmlgeterrorcallback( )
```

#### C:

```
fun = vmlGetErrorCallback( void );
```

### 出力パラメータ

#### Fortran:

*fun*            コールバック関数のアドレス。

#### C:

*fun*            コールバック関数へのポインタ。

---

## ClearErrorCallback

追加のエラー・ハンドラ・コールバック関数を削除し、前のコールバック関数を回復する。

---

### 構文

#### Fortran:

```
oldcallback = vmlclearerrorcallback( )
```

#### C:

```
oldcallback = vmlClearErrorCallBack( void );
```

### 出力パラメータ

名称	型		説明
	FORTRAN	C	
<i>oldcallback</i>	INTEGER	int	<i>Fortran</i> : 前のコールバック関数のアドレス、 <i>C</i> : 前のコールバック関数へのポインタ。





本章では、ベクトル統計ライブラリ (VSL) として知られる、擬似乱数ベクトルの生成を目的として設計されたインテル<sup>®</sup> マス・カーネル・ライブラリの機能について説明する。

- 擬似乱数ベクトルと準乱数ベクトルの生成
- 畳み込み / 相関演算の実行

機能性については、「[乱数生成器](#)」のセクションおよび「[畳み込みと相関](#)」のセクションで説明する。

## 乱数生成器

VSL は基本的な連続分布または離散分布に対応した擬似 / 準乱数生成サブルーチン群で構成される。VSL ルーチンは高度に最適化された *基本乱数生成器* とベクトル数学関数ライブラリ VML を呼び出して最高レベルの性能を実現している (VML については、第 9 章の「[ベクトル数学関数](#)」を参照)。

VSL では Fortran インターフェイスと C インターフェイスの両方が用意されている。



**注 :** FORTRAN インターフェイスでは、サブルーチン形式と関数形式のインターフェイスが提供される。デフォルトは関数形式である。サブルーチン形式は下位互換性のためにのみ提供されている。サブルーチン形式のインターフェイスを使用するには、手動で `include \mkl.fi` ファイル内の `include 'mkl_vsl.fi'` という行を `include 'mkl_vsl_subroutine.fi'` に変更し、`mkl_vsl.fi` ファイルの代わりに `mkl_vsl_subroutine.fi` ファイルを組み込む。

関数形式のインターフェイスでは、サブルーチン形式とは異なり、ユーザは各ルーチンのエラー・ステータスを取得することができる。

VSL ルーチンは 3 種類のカテゴリに分類できる。

- 一様分布、正規 ( ガウス ) 分布、二項分布など、さまざまなタイプの統計的分布に対応した変換ルーチン。これらのルーチンは、擬似乱数生成器または準乱数生成器を間接的に呼び出す。生成器の詳細は、「[分布生成器](#)」のセクションで述べる。
- 乱数ストリームを取り扱うサービスルーチン。生成、初期化、削除、コピー、バイナリファイルへの保存、バイナリファイルからのロード、基本生成器のインデックス取得がある。ルーチンの詳細は、「[サービスルーチン](#)」のセクションで述べる。
- 基本擬似乱数生成器を登録するルーチン、および登録された生成器のプロパティを取得するルーチン ( 詳細は、「[アドバンスト・サービス・ルーチン](#)」のセクションで述べる )。

カテゴリの後半 2 つをサービス・ルーチンと呼ぶ。

## 規則

本章では、文脈上必要でない限り、真の乱数、擬似乱数、準乱数の間にも、真の乱数生成器、擬似乱数生成器、準乱数生成器の間にも、特定の相違はないものとする。詳細は、製品に付属の [VSL Notes](#) の「*Random Numbers*」のセクションを参照のこと。

一様分布以外の各生成器は、離散分布と連続分布の両方とも、基本乱数生成器 (BRNG) と呼ばれる一様分布生成器を基に構築されている。一様分布以外の擬似乱数は、一様分布の擬似乱数にしかるべき変換を行って取得する。このような変換を生成方法と呼ぶ。一部の分布では複数の生成法を選択できる。各生成器で利用可能な生成法については、[VSL Notes](#) を参照のこと。

ストリーム・ディスクリプタは、与えられた変換方法で使用する BRNG を指定する。[VSL Notes](#) の「*Random Streams and RNGs in Parallel Computation*」のセクションを参照のこと。

処理ノードは、データを並列処理する能力を持った論理的または物理的ユニットを意味する。

## 数学的表記

本文では次の表記を使用する。

$N$	自然数の集合 = $\{1, 2, 3, \dots\}$
$Z$	整数の集合 $\dots -3, -2, -1, 0, 1, 2, 3$
$R$	実数の集合
$\lfloor a \rfloor$	$a$ の切捨て値 ( $a$ よりも小さいか等しい最大の整数)
$\oplus$ または <b>xor</b>	ビット単位の排他的 OR
$C_{\alpha}^k$ または $\binom{\alpha}{k}$	二項係数または組み合わせ ( $\alpha \in R, \alpha \geq 0; k \in N \cup \{0\}$ )。 $C_{\alpha}^0 = 1$ 。 $\alpha \geq k$ では二項係数は次のように定義される。 $C_{\alpha}^k = \frac{\alpha(\alpha-1) \dots (\alpha-k+1)}{k!}$ $\alpha < k$ では $C_{\alpha}^k = 0$ となる。
$\Phi(x)$	累積ガウス分布関数 $\Phi(x) = \int_{-\infty}^x \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{y^2}{2}\right) dy$ $-\infty < x < +\infty$ $\Phi(-\infty) = 0, \Phi(+\infty) = 1$
$\Gamma(\alpha)$	完全なガンマ関数 $\Gamma(\alpha) = \int_0^{\infty} t^{\alpha-1} e^{-t} dt, \alpha > 0$
$B(p, q)$	完全なベータ関数 $B(p, q) = \int_0^1 t^{p-1} (1-t)^{q-1} dt, p > 0 \text{ および } q > 0$
$LCG(a, c, m)$	線形合同生成器 $x_{n+1} = (ax_n + c) \bmod m$ ここで $a$ は乗数、 $c$ は加数、 $m$ は生成器の係数である。

MCG( $a, m$ )      乗算合同生成器  $x_{n+1} = (ax_n) \bmod m$  は、線形合同生成器で  $c$  が 0 の場合である。

GFSR( $p, q$ )      GFSR 乱数生成器  $x_n = x_{n-p} \oplus x_{n-q}$

## 命名規則

FORTRAN 用の VSL 関数名はすべて小文字である。C 用の関数名は小文字と大文字で構成される。

生成器ルーチンの名前は、次の構造になっている。

`v<type of result>rng<distribution>` (FORTRAN インターフェイスの場合)

`v<type of result>Rng<distribution>` (C インターフェイスの場合)

ここで `v` は VSL ベクトル関数のプリフィックスである。`<type of result>` フィールドは `s`、`d`、`i` のいずれかで、次のタイプから 1 つを指定する。

`s`                      REAL (FORTRAN インターフェイスの場合)

float (C インターフェイスの場合)

`d`                      DOUBLE PRECISION (FORTRAN インターフェイスの場合)

double (C インターフェイスの場合)

`i`                      INTEGER (FORTRAN インターフェイスの場合)

int (C インターフェイスの場合)

プリフィックス `s` と `d` は連続分布にのみ適用され、プリフィックス `i` は離散型にのみ適用される。プリフィックス `rng` はルーチンが乱数生成器であることを示し、`<distribution>` フィールドは統計的分布のタイプを示す。

サービス・サブルーチン名は次の規則に従う。

`vsl<name>`

ここで `vsl` は VSL サービス関数のプリフィックスである。`<name>` フィールドには関数の短縮名が入る。サービスルーチンの詳細は、「[サービスルーチン](#)」のセクションおよび「[アドバンスト・サービス・ルーチン](#)」のセクションを参照のこと。

指定された確率分布に対応する各生成期ルーチンのプロトタイプは次の構造に適合する。

`<function name>( method, stream, n, r, [<distribution parameters>] )`

- `method` は生成法を指定する数値である。このパラメータの詳細は「[分布生成器](#)」のセクションに記載されている。`method` の名前の構造体定義については、次のページを参照のこと。
- `stream` はランダム・ストリーム・ディスクリプタを定義する。非ゼロでなければならない。ランダム・ストリームとその使い方は、後述する「[ランダム・ストリーム](#)」と「[サービスルーチン](#)」に記載されている。
- `n` は生成させる乱数の個数を指定する。`n` がゼロかゼロ以下の場合、乱数は生成されない。また `n` が負の場合はエラー条件がセットされる。
- `r` は生成した乱数の出力先配列を指定する。配列の次元は、少なくとも `n` 個の乱数を格納できるだけの大きさがなくてはならない。

<*distribution parameters*> フィールドの追加パラメータは生成器ルーチンごとに異なり、詳細は「[分布生成器](#)」のセクションに記載されている。

乱数生成器を呼び出すにはそれぞれの VSL ルーチンをコールする。例えば、平均値  $a$ 、標準偏差  $\sigma$  を持つ正規 ( ガウス ) 分布に従った  $n$  個の独立かつ乱数としてみなされるベクトル  $r$  を取得するには次のように記述する。

(FORTRAN インターフェイスの場合)

```
status = vsrnggaussian( method, stream, n, r, a, sigma )
```

(C インターフェイスの場合)

```
status = vsRngGaussian( method, stream, n, r, a, sigma )
```

`method` パラメータの名前は、次の構造になっている。

```
VSL_METHOD_<precision><distribution>_<method>
```

<precision>	S 単精度連続分布
	D 倍精度連続分布
	I 離散分布
<distribution>	確率分布
<method>	生成法の名前

VSL\_METHOD\_<precision><distribution>\_<method> は、  
vsl<precision>Rng<distribution> 関数でのみ使用しなければならない。

<precision>                    s   単精度連続分布  
                                  d   倍精度連続分布  
                                  i   離散分布  
  
<distribution>                確率分布

[表 10-1](#) は、定義済みの *method* の名前について説明する。3 番目の列は、その *method* を使用する関数の名前である。

**表 10-1**                *method* パラメータの <method> の値

method	説明	関数
STD	標準生成法。現時点で、これらの関数に対する method は 1 つのみ。	Uniform ( <a href="#">continuous</a> )、 Uniform ( <a href="#">discrete</a> )、 <a href="#">UniformBits</a>
BOXMULLER	BOXMULLER は、次の式に従い、一様分布の乱数 $u_1$ と $u_2$ のペアを介して正規分布に従った乱数 $x$ を生成する。 $x = \sqrt{-2\ln u_1} \sin 2\pi u_2$	<a href="#">Gaussian</a> 、 <a href="#">GaussianMV</a>
BOXMULLER2	BOXMULLER2 は、次の式に従い、一様分布の乱数 $u_1$ と $u_2$ のペアを介して正規分布に従った乱数 $x_1$ と $x_2$ を生成する。 $x_1 = \sqrt{-2\ln u_1} \sin 2\pi u_2$ $x_2 = \sqrt{-2\ln u_1} \cos 2\pi u_2$	<a href="#">Gaussian</a> 、 <a href="#">GaussianMV</a>
ICDF	逆累積分布関数法	<a href="#">Exponential</a> 、 <a href="#">Laplace</a> 、 <a href="#">Weibull</a> 、 <a href="#">Cauchy</a> 、 <a href="#">Rayleigh</a> 、 <a href="#">Lognormal</a> 、 <a href="#">Gumbel</a> 、 <a href="#">Bernoulli</a> 、 <a href="#">Geometric</a>
GNORM	ガンマ分布に従った乱数は、次のように生成される。 $\alpha > 1$ の場合、適切にスケールされた正規乱数の 3 乗として生成される。 $0.6 \leq \alpha < 1$ の場合、ワイブル分布からの棄却を使用して生成される。 $\alpha < 0.6$ の場合、指数分布の変換を使用して取得する。 $\alpha = 1$ の場合、指数分布になる。	<a href="#">Gamma</a>

表 10-1 *method* パラメータの *<method>* の値 ( 続き )

method	説明	関数
CJA	$\min(p, q) > 1$ の場合、Cheng 法が使用される。 $\min(p, q) < 1$ の場合、 $q + K \cdot p^2 + C \leq 0$ ( $K = 0.852\dots$ , $C = -0.956\dots$ ) ならば Jöhnk 法が、それ以外は Atkinson 切り替えアルゴリズムが使用される。 $\max(p, q) < 1$ の場合、Jöhnk 法が使用される。 $\min(p, q) < 1$ 、 $\max(p, q) > 1$ の場合、Atkinson 切り替えアルゴリズムが使用される (CJA は Cheng、Jöhnk、Atkinson の頭文字を示す)。 $p = 1$ または $q = 1$ の場合、逆累積分布関数法を使用する。 $p = 1$ および $q = 1$ の場合、ベータ分布は一樣分布になる。	<a href="#">Beta</a>
BTPE	$ntrial \cdot \min(p, 1-p) \geq 30$ の受容 / 棄却法。 次の 4 つの領域に分解する。 - 2 つの平行四辺形 - 三角形 - 指数左裾 - 指数右裾	<a href="#">Binomial</a>
H2PE	大規模な分布の受容 / 棄却法。次の 3 つの領域に分解する。 - 矩形 - 指数左裾 - 指数右裾	<a href="#">Hypergeometric</a>
PTPE	$\lambda \geq 27$ の受容 / 棄却法。次の 4 つの領域に分解する。 - 2 つの平行四辺形 - 三角形 - 指数左裾 - 指数右裾 それ以外の場合、テーブル・ルックアップ法が使用される。	<a href="#">Poisson</a>
POISNORM	$\lambda \geq 1$ の場合、ガウス分布の逆累積分布関数によるポアソン分布の逆累積分布関数近似に基づく方法が使用される。 $\lambda < 1$ の場合、テーブル・ルックアップ法が使用される。	<a href="#">Poisson</a> 、 <a href="#">PoissonV</a>

表 10-1 *method* パラメータの *<method>* の値 ( 続き )

method	説明	関数
NBAR	<p>以下の場合の受容 / 棄却法。</p> $\frac{(a-1) \cdot (1-p)}{p} \geq 100$ <p>次の 5 つの領域に分解する。</p> <ul style="list-style-type: none"> <li>- 矩形</li> <li>- 2 つの台形</li> <li>- 指数左裾</li> <li>- 指数右裾</li> </ul>	<a href="#">NegBinomial</a>

## 基本生成器

VSL は、処理速度と特性の異なる次の基本乱数生成器 (BRNG) を提供する。

- 32 ビット乗算合同擬似乱数生成器 *MCG (1132489760, 2<sup>31</sup> - 1)* [[L'Ecuyer99](#)]
- 32 ビット GFSR 擬似乱数生成器 *GFSR (250, 103)* [[Kirkpatrick81](#)]
- 複合再帰擬似乱数生成器 *MRG-32k3a* [[L'Ecuyer99a](#)]
- NAG 数値ライブラリ [[NAG](#)] から 59 ビット乗算合同擬似乱数生成器 *MCG (13<sup>13</sup>, 2<sup>59</sup>)*
- NAG 数値ライブラリ [[NAG](#)] から Wichmann-Hill 擬似乱数生成器 ( 実際は 273 個の基本生成器の集合 )
- 生成されるシーケンスの周期の長さが 2<sup>19937</sup>-1 である Mersenne Twister 擬似乱数生成器 MT19937 [[Matsumoto98](#)]
- 1024 個の Mersenne Twister 擬似乱数生成器の集合 MT2203 [[Matsumoto98](#)], [[Matsumoto2000](#)]。各生成器は、周期が 2<sup>2203</sup>-1 のシーケンスを生成する。生成器のパラメータは、対応するシーケンスの相互独立を提供する。

これらの擬似乱数生成器に加え、VSL では基本準乱数生成器も 2 つ用意されている。

- 1 から 40 までの次元で機能する Sobol 準乱数生成器 [[Sobol76](#)]、[\[Bratley88\]](#)
- 1 から 318 までの次元で機能する Niederreiter 準乱数生成器 [[Bratley92](#)]

生成器の性能比較解析と検定結果は [VSL Notes](#) に記載されている。

VSL では、「[アドバンスド・サービス・ルーチン](#)」のセクションに記載のとおり、ユーザ定義の生成器を登録する手段を設けている。



一部の基本生成器では、マルチプロセッサ処理に対応して、複数の独立したランダム・ストリームを生成するリープフロッグ法とブロック分割法の2種類が提供される。これらのシーケンス分割法は、逐次モンテカルロ法にも役立つ。

また、MT2203 擬似乱数生成器は、1024 までの独立した乱数シーケンスを生成する、1024 個の生成器の集合である。この生成器は、並列モンテカルロ・シミュレーションで使用される。Wichmann-Hill 生成器も同様の機能を持つ。この生成器は、273 までの独立した乱数シーケンスを生成する。並列処理向けに設計された生成器の特性は [\[Coddington94\]](#) に詳細が述べられている。

ユーザは独自の基本生成器も作成できる。VSL では、「[アドバンスト・サービス・ルーチン](#)」のセクションに記載のとおり、ユーザ定義の生成器を登録する手段を設けている。

また、外部で生成された乱数を VSL 分布生成器ルーチンで利用するためのオプションもある。この目的のために、VSL ではさらに3つの基本乱数生成器を提供している。

- 32 ビットの整数配列に格納された外部乱数データ用
- 倍精度の浮動小数点配列に格納された外部乱数データ用。データは、区間 (a,b) を持つ一様分布であるものとする。
- 単精度の浮動小数点配列に格納された外部乱数データ用。データは、区間 (a,b) を持つ一様分布であるものとする。

このような基本生成器を抽象基本生成器と呼ぶ。

生成器の特性に関する詳細は、[VSL Notes](#) を参照のこと。

BRNG パラメータの定義

brng 入力パラメータの定義済み値を以下に示す。

表 10-2 brng パラメータの値

値	説明
VSL_BRNG_MCG31	31 ビット乗算合同生成器
VSL_BRNG_R250	GFSR 乱数生成器。
VSL_BRNG_MRG32K3A	次数 3 のコンポーネントを 2 つ持つ複合再帰生成器
VSL_BRNG_MCG59	59 ビット乗算合同生成器
VSL_BRNG_WH	273 個の Wichmann-Hill 乗算合同生成器の集合
VSL_BRNG_MT19937	Mersenne Twister 擬似乱数生成器
VSL_BRNG_MT2203	1024 個の Mersenne Twister 擬似乱数生成器の集合

表 10-2 *brng* パラメータの値 ( 続き )

値	説明
VSL_BRNG_SOBOL	次元 $1 \leq s \leq 40$ に対して、不一致の少ないシーケンスを生成する 32 ビット Gray コードベース生成器
VSL_BRNG_NIEDERR	次元 $1 \leq s \leq 318$ に対して、不一致の少ないシーケンスを生成する 32 ビット Gray コードベース生成器
VSL_BRNG_IABSTRACT	整数配列用の抽象乱数生成器
VSL_BRNG_DABSTRACT	倍精度の浮動小数点配列用の抽象乱数生成器
VSL_BRNG_SABSTRACT	単精度の浮動小数点配列用の抽象乱数生成器

詳細は、[VSL Notes](#) を参照のこと。

## ランダム・ストリーム

ランダム・ストリーム ( または ストリーム ) は、一様分布の擬似乱数シーケンスおよび準乱数シーケンスの抽象的なソースである。これらのシーケンスに直接アクセスすることはできない。ストリーム・ステート・ディスクリプタを使用して操作する。ストリーム・ステート・ディスクリプタは、特定の BRNG のステート・ディスクリプティブ情報を格納し、分布生成器の各ルーチンにおいて必須パラメータである。分布生成器ルーチンのみ、ランダム・ストリームで直接動作する。詳細は [VSL Notes](#) を参照のこと。



**注：** 抽象乱数生成器に関連するランダム・ストリームを抽象ランダム・ストリームと呼ぶ。抽象ストリームとその使用に関する詳細は、[VSL Notes](#) を参照のこと。

[NewStream](#) のように、VSL [サービスルーチン](#) によって作成できるランダム・ストリームの数に制限はない。また、指定された確率分布の乱数ストリームを取得するために、任意の分布生成器で作成したランダム・ストリームを利用することができる。ストリームが必要でなくなった場合は、サービスルーチン [DeleteStream](#) を呼び出して、ストリームを削除しなければならない。

VSL では、ランダム・ストリーム・ディスクリプティブ・データをバイナリファイルに保存したり、バイナリファイルから読み取るために、サービスルーチン [SaveStreamF](#) と [LoadStreamF](#) を提供している。詳細は、[VSL Notes](#) を参照のこと。

データ型

FORTRAN:

```
TYPE VSL_STREAM_STATE
    INTEGER*4 descriptor1
    INTEGER*4 descriptor2
END TYPE VSL_STREAM_STATE
```

C:

```
typedef (void*) VSLStreamStatePtr;
```

ユーザ定義生成器のストリーム・ステート構造体のフォーマットは、「[アドバンスト・サービス・ルーチン](#)」のセクションを参照のこと。

エラー報告

VSL ルーチンは、呼び出し元のプログラムにエラーおよび警告を報告するために、実行した演算のステータス・コードを返す。このため、エラーに関連するアクションやエラーからの復元を実行するかどうかはアプリケーションによって決定される。ステータス・コードは整数型で以下の形式で示される。

VSL\_ERROR\_<ERROR\_NAME> - VSL エラーを示す。

VSL\_WARNING\_<WARNING\_NAME> - VSL 警告を示す。

VSL エラーは負の値で、警告は正の値である。ゼロの場合は、演算が正常に終了したことを示す (VSL\_ERROR\_OK または VSL\_STATUS\_OK)。

表 10-3      ステータス・コードとメッセージ

ステータス・コード	メッセージ
VSL_ERROR_OK, VSL_STATUS_OK	正常に終了したことを示す。
VSL_ERROR_BAD_ARG	入力引数の値が不正である。
VSL_ERROR_NULL_PTR	入力ポインタ引数が NULL である。
VSL_ERROR_MEM_FAILURE	システムがメモリを割り当てることができない。
VSL_ERROR_INVALID_BRNG_INDEX	BRNG のインデックスが不正である。
VSL_ERROR_BRNGS_INCOMPATIBLE	2 つの BRNG は、この演算では互換性がない。
VSL_ERROR_LEAPFROG_UNSUPPORTED	BRNG がリープフロッグ法をサポートしていない。

**表 10-3 ステータス・コードとメッセージ ( 続き )**

VSL_ERROR_SKIPAHEAD_UNSUPPORTED	BRNG が Skip-Ahead 法をサポートしていない。
VSL_ERROR_BAD_STREAM	ランダム・ストリームが無効である。
VSL_ERROR_FILE_OPEN	ファイルを開く際にエラーが発生したことを示す。
VSL_ERROR_FILE_READ	ファイルを読み取る際にエラーが発生したことを示す。
VSL_ERROR_FILE_WRITE	ファイルへ書き込む際にエラーが発生したことを示す。
VSL_ERROR_FILE_CLOSE	ファイルを閉じる際にエラーが発生したことを示す。
VSL_ERROR_BAD_FILE_FORMAT	ファイル形式が不明である。
VSL_ERROR_UNSUPPORTED_FILE_VER	サポートされていないファイル形式のバージョンである。
VSL_ERROR_BRNG_TABLE_FULL	登録されている BRNG の表にフリー成分が不足しているため、登録を完了することができない。
VSL_ERROR_BAD_STREAM_STATE_SIZE	StreamStateSize フィールドの値が不正である。
VSL_ERROR_BAD_WORD_SIZE	WordSize フィールドの値が不正である。
VSL_ERROR_BAD_NSEEDS	NSeeds フィールドの値が不正である。
VSL_ERROR_BAD_NBITS	NBits フィールドの値が不正である。
VSL_ERROR_BAD_UPDATE	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数として無効な値 (<0 または > nmax) を返した。
VSL_ERROR_NO_NUMBERS	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数としてゼロを返した。
VSL_ERROR_INVALID_ABSTRACT_STREAM	抽象ランダム・ストリームが無効である。

## サビスルーチン

ストリームを取り扱うルーチン群であり、ストリームの作成、削除、コピー、基本生成器のインデックス取得がある。ランダム・ストリームは、バイナリファイルへ保存、またはバイナリファイルから読み出すこともできる。[表 10-4](#) は、利用可能なサビス・サブルーチンの全リストを示す。

**表 10-4 サビスルーチン**

ルーチン	説明
<a href="#">NewStream</a>	ランダム・ストリームを作成し初期化する。
<a href="#">NewStreamEx</a>	複数の初期化条件を必要とする生成器に対し、ランダム・ストリームを作成し初期化する。
<a href="#">iNewAbstractStream</a>	整数配列の抽象ランダム・ストリームを作成し、初期化する。

表 10-4 サービスルーチン ( 続き )

ルーチン	説明
<a href="#">dNewAbstractStream</a>	倍精度の浮動小数点配列の抽象ランダム・ストリームを作成し、初期化する。
<a href="#">sNewAbstractStream</a>	単精度の浮動小数点配列の抽象ランダム・ストリームを作成し、初期化する。
<a href="#">DeleteStream</a>	以前に生成したストリームを削除する
<a href="#">CopyStream</a>	別のストリームにコピーする
<a href="#">CopyStreamState</a>	ランダム・ストリーム・ステートのコピーを作成する
<a href="#">SaveStreamF</a>	ストリームをバイナリファイルに書き込む。
<a href="#">LoadStreamF</a>	ストリームをバイナリファイルから読み取る。
<a href="#">LeapfrogStream</a>	元のシーケンスのサブシーケンスを生成するため、ストリームをリープフロッグ法で初期化する。
<a href="#">SkipAheadStream</a>	ストリームを Skip-Ahead 法で初期化する。
<a href="#">GetStreamStateBrng</a>	指定のランダム・ストリームを生成した基本生成器のインデックスを取得する
<a href="#">GetNumRegBrngs</a>	現在登録されている基本生成器の数を取得する。



**注：**上記の表では関数名の `vs1` プリフィックスを省略した。このプリフィックスは、関数のリファレンスでは関数プロトタイプとコード例とともに常に使用される。

生成器を用いた処理は、基本的な 3 つのステップで構成する。

1. ストリームを作成し、初期化する ([NewStream](#)、[NewStreamEx](#)、[CopyStream](#)、[CopyStreamState](#)、[LeapfrogStream](#)、[SkipAheadStream](#))。
2. 指定された分布を用いて乱数を生成する。「[分布生成器](#)」を参照。
3. ストリームを削除する ([DeleteStream](#))。

ストリームは同時に複数を作成可能で、1 つまたは複数の生成器からストリーム・ステートを用いて乱数データを取得できる。最後に [DeleteStream](#) 関数を用いてすべてのストリームを削除しなければならない。

## NewStream

ランダム・ストリームを作成し初期化する。

### 構文

#### Fortran:

```
status = vslnewstream( stream, brng, seed )
```

#### C:

```
status = vslNewStream( &stream, brng, seed );
```

### 説明

この関数は、*brng* 値を持つ基本生成器のストリームを新規に作成し、32 ビットの種 (シード) を使用して初期化を行う。種とは、基本生成器 *brng* によって生成された特定のシーケンスを選択するのに使用する初期値である。また、この関数は、複数の初期化条件を伴う生成器にも適用可能である。基本生成器ごとのストリーム初期化の詳細は [VSL Notes](#) を参照のこと。



**注：**この関数は、抽象基本乱数生成器には適用できない。整数、単精度、倍精度の外部乱数データを利用するには、それぞれ

[vslNewAbstractStream](#)、[vslsNewAbstractStream](#)、または [vslNewAbstractStream](#) を使用する。

### 入力パラメータ

名前	データ型		説明
	FORTRAN	C	
<i>brng</i>	INTEGER, INTENT( IN )	int	ストリームを初期化する基本生成器のインデックス値。特定の値については、 <a href="#">表 10-2</a> を参照のこと。
<i>seed</i>	INTEGER, INTENT( IN )	unsigned int	ストリームの初期条件。準乱数生成器では、 <i>seed</i> パラメータは次元を設定するのに使用される。次元が <i>brng</i> のサポートする次元よりも大きい場合、または 1 よりも小さい場合、次元は 1 とみなされる。

出力パラメータ

名前	データ型		説明
	FORTRAN	C	
<i>stream</i>	TYPE(VSL_STREAM_STATE), INTENT(OUT)	VSLStreamStatePtr*	ストリーム・ステート・ディスクリプタ

戻り値

VSL_ERROR_OK, VSL_STATUS_OK	正常に終了したことを示す。
VSL_ERROR_INVALID_BRNG_INDEX	BRNG のインデックスが無効である。
VSL_ERROR_MEM_FAILURE	システムが、 <i>stream</i> に対してメモリを割り当てる ことができない。

NewStreamEx

複数の初期化条件を使用する生成器でランダム・ストリームを作成し初期化する。

構文

Fortran:

```
status = vslnewstreamex( stream, brng, n, params )
```

C:

```
status = vslNewStreamEx( &stream, brng, n, params );
```

説明

この関数は、入力引数が複数の初期化パラメータで構成される基本生成器に複数の初期条件を設定する。初期値は、基本生成器 *brng* によって生成された特定のシーケンスを選択するのに使用される。可能であれば、32 ビットの単一初期条件のみを扱う

[NewStream](#) を、[vslNewStreamEx](#) に相当する関数として使用する。ただし、GFSR のステートテーブルの初期化には [vslNewStreamEx](#) を使用する。詳細は [VSL Notes](#) を参照のこと。



**注：**この関数は、抽象基本乱数生成器には適用できない。整数、単精度、倍精度の外部乱数データを利用するには、それぞれ [vslNewAbstractStream](#)、[vslsNewAbstractStream](#)、または [vslNewAbstractStream](#) を使用する。

## 入力パラメータ

名前	データ型		説明
	FORTRAN	C	
<i>brng</i>	INTEGER, INTENT ( IN )	int	ストリームを初期化する基本生成器のインデックス値。特定の値については、 <a href="#">表 10-2</a> を参照のこと。
<i>n</i>	INTEGER, INTENT ( IN )	int	<i>params</i> に含まれる初期条件の数。
<i>params</i>	INTEGER, INTENT ( IN )	const unsigned int	基本生成器 <i>brng</i> がストリームの初期化に必要とする初期条件の配列。準乱数生成器では、次元を設定するのに <i>params</i> パラメータの最初の成分が使用される。次元が <i>brng</i> のサポートする次元よりも大きい場合、または 1 よりも小さい場合、次元は 1 とみなされる。

## 出力パラメータ

名前	データ型		説明
	FORTRAN	C	
<i>stream</i>	TYPE ( VSL_STREAM_STATE ), INTENT ( OUT )	VSLStreamStatePtr*	ストリーム・ステート・ディスクリプタ



## 戻り値

VSL_ERROR_OK, VSL_STATUS_OK	正常に終了したことを示す。
VSL_ERROR_INVALID_BRNG_INDEX	BRNG のインデックスが無効である。
VSL_ERROR_MEM_FAILURE	システムが、 <i>stream</i> に対してメモリを割り当てる ことができない。

---

## iNewAbstractStream

整数配列の抽象ランダム・ストリームを作成し、  
初期化する。

---

## 構文

### Fortran:

```
status = vslnewabstractstream( stream, n, ibuf, icallback )
```

### C:

```
status = vsliNewAbstractStream( &stream, n, ibuf, icallback );
```

## 説明

この関数は、新規の抽象ストリームを作成し、整数配列 *ibuf* とユーザのコールバック関数 *icallback* に関連付ける。*icallback* は *ibuf* の内容を更新する。

## 入力パラメータ

名前	データ型		説明
	FORTRAN	C	
<i>n</i>	INTEGER, INTENT( IN )	int	配列 <i>ibuf</i> のサイズ。
<i>ibuf</i>	INTEGER, INTENT( IN )	unsigned int*	<i>n</i> 個の 32 ビット整数の配列。
<i>icallback</i>	次のノート を参照	次のノート を参照	Fortran の場合: <i>ibuf</i> の更新に使用するコールバック関数のアドレス。 C の場合: <i>ibuf</i> の更新に使用するコールバック関数へのポインタ。

## 出力パラメータ

名前	データ型		説明
	FORTRAN	C	
<i>stream</i>	TYPE(VSL_STREAM_STATE), INTENT( OUT )	VSLStreamStatePtr*	ストリーム・ステート構造体のディスクリプタ。

ノート:

Fortran におけるコールバック関数の形式:

```
INTEGER FUNCTION IUPDATEFUNC[C]( stream, n, ibuf, nmin, nmax, idx )
    TYPE(VSL_STREAM_STATE), POINTER :: stream[reference]
    INTEGER(KIND=4), INTENT( IN)      :: n[reference]
    INTEGER(KIND=4), INTENT( OUT)     :: ibuf[reference](0:n-1)
    INTEGER(KIND=4), INTENT( IN)      :: nmin[reference]
    INTEGER(KIND=4), INTENT( IN)      :: nmax[reference]
    INTEGER(KIND=4), INTENT( IN)      :: idx[reference]
```

C におけるコールバック関数の形式:

```
int iUpdateFunc( VSLStreamStatePtr stream, int* n, unsigned int ibuf[],
int* nmin, int* nmax, int* idx );
```

コールバック関数は、この関数によって実際に更新される配列の成分の数を返す。[表 10-5](#) は、コールバック関数のパラメータの説明である。

表 10-5 `icallback` コールバック関数のパラメータ

パラメータ	説明
<code>stream</code>	抽象ストリーム・ディスクリプタ
<code>n</code>	<code>ibuf</code> のサイズ
<code>ibuf</code>	ストリーム <code>stream</code> に関連付けられた乱数の配列
<code>nmin</code>	更新する乱数の最小個数
<code>nmax</code>	更新できる乱数の最大個数
<code>idx</code>	更新を開始するサイクリック・バッファ <code>ibuf</code> の位置 $0 \leq idx < n$

### 戻り値

<code>VSL_ERROR_OK</code> , <code>VSL_STATUS_OK</code>	正常に終了したことを示す。
<code>VSL_ERROR_BAD_ARG</code>	パラメータ <code>n</code> が正ではない。
<code>VSL_ERROR_MEM_FAILURE</code>	システムが、 <code>stream</code> に対してメモリを割り当てる ことができない。
<code>VSL_ERROR_NULL_PTR</code>	バッファまたはコールバック関数のパラメータが NULL ポインタである。

## dNewAbstractStream

倍精度の浮動小数点配列の抽象ランダム・スト  
リームを作成し、初期化する。

### 構文

#### Fortran:

```
status = vsldnewabstractstream( stream, n, dbuf, a, b, dcallback )
```

#### C:

```
status = vsldNewAbstractStream( &stream, n, dbuf, a, b, dcallback );
```

## 説明

この関数は、区間 (a,b) を持つ一様分布に従った乱数を格納する、倍精度の浮動小数点配列に対する新規の抽象ストリームを作成する。この関数は、ストリームを倍精度の配列 *dbuf* とユーザのコールバック関数 *dcallback* に関連付ける。*dcallback* は *dbuf* の内容を更新する。

## 入力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>n</i>	INTEGER, INTENT ( IN )	int	配列 <i>dbuf</i> のサイズ。
<i>dbuf</i>	DOUBLE PRECISION, INTENT ( IN )	double*	区間 (a,b) を持つ一様分布に従った、 <i>n</i> 個の倍精度の浮動小数点乱数の配列。
<i>a</i>	DOUBLE PRECISION, INTENT ( IN )	double	区間の開始 <i>a</i> 。
<i>b</i>	DOUBLE PRECISION, INTENT ( IN )	double	区間の終了 <i>b</i> 。
<i>dcallback</i>	次のノート を参照	次のノート を参照	Fortran の場合：配列 <i>dbuf</i> の更新に使用したコールバック関数のアドレス。 C の場合：配列 <i>dbuf</i> の更新に使用したコールバック関数へのポインタ。

## 出力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>stream</i>	TYPE (VSL_STREAM_STATE), INTENT ( OUT )	VSLStreamStatePtr*	ストリーム・ステート構造体のディスクリプタ。

ノート：

Fortran におけるコールバック関数の形式：

```

INTEGER FUNCTION DUUPDATEFUNC[C]( stream, n, dbuf, nmin, nmax, idx )
    TYPE(VSL_STREAM_STATE), POINTER :: stream[reference]
    INTEGER(KIND=4), INTENT( IN)      :: n[reference]
    REAL(KIND=8), INTENT(OUT)         :: dbuf[reference](0:n-1)

```

```
INTEGER(KIND=4), INTENT(IN)      :: nmin[reference]
INTEGER(KIND=4), INTENT(IN)      :: nmax[reference]
INTEGER(KIND=4), INTENT(IN)      :: idx[reference]
```

C におけるコールバック関数の形式：

```
int dUpdateFunc( VSLStreamStatePtr stream, int* n, double dbuf[], int*
nmin, int* nmax, int* idx );
```

コールバック関数は、この関数によって実際に更新される配列の成分の数を返す。[表 10-6](#) は、コールバック関数のパラメータの説明である。

表 10-6 dcallback コールバック関数のパラメータ

パラメータ	説明
<i>stream</i>	抽象ストリーム・ディスクリプタ
<i>n</i>	<i>dbuf</i> のサイズ
<i>dbuf</i>	ストリーム <i>stream</i> に関連付けられた乱数の配列
<i>nmin</i>	更新する乱数の最小個数
<i>nmax</i>	更新できる乱数の最大個数
<i>idx</i>	更新を開始するサイクリック・バッファ <i>dbuf</i> の位置 $0 \leq idx < n$

戻り値

VSL_ERROR_OK, VSL_STATUS_OK	正常に終了したことを示す。
VSL_ERROR_BAD_ARG	パラメータ <i>n</i> が正ではない。
VSL_ERROR_MEM_FAILURE	システムが、 <i>stream</i> に対してメモリを割り当てる ことができない。
VSL_ERROR_NULL_PTR	バッファまたはコールバック関数のパラメータが NULL ポインタである。

## sNewAbstractStream

単精度の浮動小数点配列の抽象ランダム・ストリームを作成し、初期化する。

### 構文

**Fortran:**

```
status = vslsnewabstractstream( stream, n, sbuf, a, b, scallback )
```

**C:**

```
status = vslsNewAbstractStream( &stream, n, sbuf, a, b, scallback );
```

### 説明

この関数は、区間 (a,b) を持つ一様分布に従った乱数を格納する、単精度の浮動小数点配列に対する新規の抽象ストリームを作成する。この関数は、ストリームを単精度の配列 *sbuf* とユーザのコールバック関数 *scallback* に関連付ける。*dcallback* は *sbuf* の内容を更新する。

### 入力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>n</i>	INTEGER, INTENT ( IN )	int	配列 <i>sbuf</i> のサイズ。
<i>sbuf</i>	REAL, INTENT ( IN )	float*	区間 (a,b) を持つ一様分布に従った、 <i>n</i> 個の単精度の浮動小数点乱数の配列。
<i>a</i>	REAL, INTENT ( IN )	float	区間の開始 <i>a</i> 。
<i>b</i>	REAL, INTENT ( IN )	float	区間の終了 <i>b</i> 。
<i>scallback</i>	次のノート を参照	次のノート を参照	<i>Fortran</i> の場合 : 配列 <i>sbuf</i> の更新に使用したコールバック関数のアドレス。 <i>C</i> の場合 : 配列 <i>sbuf</i> の更新に使用したコールバック関数へのポインタ。

## 出力パラメータ

名前	データ型		説明
	FORTRAN	C	
<i>stream</i>	TYPE(VSL_STREAM_STATE), INTENT(OUT)	VSLStreamStatePtr*	ストリーム・ステート構造体の ディスクリプタ。

ノート:

Fortran におけるコールバック関数の形式:

```
INTEGER FUNCTION SUPDATEFUNC(C)( stream, n, sbuf, nmin, nmax, idx )
    TYPE(VSL_STREAM_STATE), POINTER :: stream[reference]
    INTEGER(KIND=4), INTENT(IN)      :: n[reference]
    REAL(KIND=4), INTENT(OUT)        :: sbuf[reference](0:n-1)
    INTEGER(KIND=4), INTENT(IN)      :: nmin[reference]
    INTEGER(KIND=4), INTENT(IN)      :: nmax[reference]
    INTEGER(KIND=4), INTENT(IN)      :: idx[reference]
```

C におけるコールバック関数の形式:

```
int sUpdateFunc( VSLStreamStatePtr stream, int* n, float sbuf[], int*
nmin, int* nmax, int* idx );
```

コールバック関数は、この関数によって実際に更新される配列の成分の数を返す。[表 10-7](#) は、コールバック関数のパラメータの説明である。

**表 10-7**      **scallback コールバック関数のパラメータ**

パラメータ	説明
<i>stream</i>	抽象ストリーム・ディスクリプタ
<i>n</i>	<i>sbuf</i> のサイズ
<i>sbuf</i>	ストリーム <i>stream</i> に関連付けられた 乱数の配列
<i>nmin</i>	更新する乱数の最小個数
<i>nmax</i>	更新できる乱数の最大個数
<i>idx</i>	更新を開始するサイクリック・バッファ <i>sbuf</i> の位置 $0 \leq idx < n$

## 戻り値

VSL_ERROR_OK, VSL_STATUS_OK	正常に終了したことを示す。
VSL_ERROR_BAD_ARG	パラメータ <i>n</i> が正ではない。
VSL_ERROR_MEM_FAILURE	システムが、 <i>stream</i> に対してメモリを割り当てる ことができない。
VSL_ERROR_NULL_PTR	バッファまたはコールバック関数のパラメータが NULL ポインタである。

---

## DeleteStream

ランダム・ストリームを削除する。

---

## 構文

### Fortran:

```
status = vsldeletestream( stream )
```

### C:

```
status = vslDeleteStream( &stream );
```

## 説明

この関数は、いずれかの初期化関数で作成されたランダム・ストリームを削除する。



入力 / 出力パラメータ

名前	データ型		説明
	FORTRAN	C	
<i>stream</i>	TYPE(VSL_STREAM_STATE), INTENT(OUT)	VSLStreamStatePtr*	<i>Fortran</i> の場合: ストリーム・ステート・ディスクリプタ。非ゼロであること。ストリームの削除に成功すると、ディスクリプタは無効になる。 <i>C</i> の場合: ストリーム・ステート・ディスクリプタ。非ゼロであること。ストリームの削除に成功するとポインタには <code>NULL</code> がセットされる。

戻り値

VSL_ERROR_OK, VSL_STATUS_OK	正常に終了したことを示す。
VSL_ERROR_NULL_PTR	<i>stream</i> パラメータが <code>NULL</code> ポインタである。
VSL_ERROR_BAD_STREAM	<i>stream</i> が有効なランダム・ストリームでない。

CopyStream

ランダム・ストリームをコピーする。

構文

**Fortran:**

```
status = vslcopystream( newstream, srcstream )
```

**C:**

```
status = vslCopyStream( &newstream, srcstream );
```

説明

この関数は *srcstream* の完全なコピーを作成し *newstream* のディスクリプタに格納する。

## 入力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>srcstream</i>	TYPE(VSL_STREAM_STATE), INTENT(IN)	VSLStreamStatePtr	Fortran の場合: コピーするストリームのディスクリプタ。 C の場合: コピーするストリーム・ステート構造体へのポインタ。

## 出力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>newstream</i>	TYPE(VSL_STREAM_STATE), INTENT(OUT)	VSLStreamStatePtr*	コピーされたストリーム・ディスクリプタ

## 戻り値

VSL_ERROR_OK, VSL_STATUS_OK	正常に終了したことを示す。
VSL_ERROR_NULL_PTR	<i>srcstream</i> パラメータが NULL ポインタである。
VSL_ERROR_BAD_STREAM	<i>srcstream</i> が有効なランダム・ストリームでない。
VSL_ERROR_MEM_FAILURE	システムが、 <i>newstream</i> に対してメモリを割り当てることができない。

## CopyStreamState

ランダム・ストリーム・ステートのコピーを作成する

### 構文

#### Fortran:

```
status = vslcopystreamstate( deststream, srcstream )
```

#### C:

```
status = vslCopyStreamState( deststream, srcstream );
```

説明

この関数は *srcstream* から既存の *deststream* ストリームへストリーム・ステートをコピーする。両方のストリームは同一の基本生成器を使って作成されていなければならない。*deststream* ストリームを作成した **BRNG** インデックスが *srcstream* ストリームを作成した **BRNG** インデックスと異なっている場合はエラー・メッセージが出力される。

[CopyStream](#) 関数は新規のストリームを作成して *srcstream* からストリーム・ステートと他のデータをコピーするが、[CopyStreamState](#) 関数は *srcstream* ストリーム・ステート・データのみ作成済みの *deststream* ストリームにコピーする。

入力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>srcstream</i>	TYPE(VSL_STREAM_STATE), INTENT(IN)	VSLStreamStatePtr	<i>Fortran</i> の場合: <i>srcstream</i> ストリームのステートがコピーされるコピー先ストリームのディスクリプタ。 <i>C</i> の場合: ステート構造体がコピーされるストリーム・ステート構造体へのポインタ。

出力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>deststream</i>	TYPE(VSL_STREAM_STATE), INTENT(OUT)	VSLStreamStatePtr	<i>Fortran</i> の場合: コピーされたステートを持つストリームのディスクリプタ。 <i>C</i> の場合: ストリーム・ステートがコピーされたストリーム・ステート構造体へのポインタ。

戻り値

VSL_ERROR_OK, VSL_STATUS_OK	正常に終了したことを示す。
VSL_ERROR_NULL_PTR	<i>srcstream</i> または <i>deststream</i> が NULL ポインタである。
VSL_ERROR_BAD_STREAM	<i>srcstream</i> または <i>deststream</i> が有効なランダム・ストリームではない。

VSL\_ERROR\_BRNGS\_INCOMPATIBLE

*srcstream* に関連する BRNG と *deststream* に関連する BRNG との間に互換性がない。

## SaveStreamF

ランダム・ストリーム・ディスクリプティブ・データをバイナリファイルに書き込む。

### 構文

#### Fortran:

```
errstatus = vslsavestreamf( stream, fname )
```

#### C:

```
errstatus = vslSaveStreamF( stream, fname );
```

### 説明

この関数は、ランダム・ストリーム・ディスクリプティブ・データをバイナリファイルに書き込む。ランダム・ストリーム・ディスクリプティブ・データは、*fname* という名前でバイナリファイルに保存される。ランダム・ストリーム *stream* は、[NewStream](#) や [CopyStream](#) などのサービスルーチンによって作成された、有効なストリームでなければならない。ストリームをファイルに保存できない場合、*errstatus* は非ゼロである。[LoadStreamF](#) 関数を使用して、バイナリファイルからランダム・ストリームを読み取ることができる。

### 入力パラメータ

名前	データ型		説明
	FORTRAN	C	
<i>stream</i>	TYPE(VSL_STREAM_STATE), INTENT(IN)	VSLStreamStatePtr	ファイルへ書き込むランダム・ストリーム。
<i>fname</i>	CHARACTER(*), INTENT(IN)	char*	<i>Fortran</i> の場合 : C 形式の NULL で終了する文字列として指定されたファイル名。 C の場合 : Fortran 形式の文字列として指定されたファイル名。

## 出力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>errstatus</i>	INTEGER	int	演算のエラー・ステータス。

## 戻り値

VSL_ERROR_OK, VSL_STATUS_OK	正常に終了したことを示す。
VSL_ERROR_NULL_PTR	<i>fname</i> または <i>stream</i> が NULL ポインタである。
VSL_ERROR_BAD_STREAM	<i>stream</i> が有効なランダム・ストリームでない。
VSL_ERROR_FILE_OPEN	ファイルを開く際にエラーが発生したことを示す。
VSL_ERROR_FILE_WRITE	ファイルへ書き込む際にエラーが発生したことを示す。
VSL_ERROR_FILE_CLOSE	ファイルを閉じる際にエラーが発生したことを示す。
VSL_ERROR_MEM_FAILURE	システムが、内部ニーズに対してメモリを割り当てることができない。

## LoadStreamF

新規のストリームを作成して、ストリーム・ディ  
スクリプティブ・データをバイナリファイルから  
読み取る。

### 構文

#### Fortran:

```
errstatus = vslloadstreamf( stream, fname )
```

#### C:

```
errstatus = vslLoadStreamF( &stream, fname );
```

## 説明

この関数は、新規のストリームを作成して、ストリーム・ディスクリプティブ・データをバイナリファイルから読み取る。*fname* という名前のバイナリファイルから、ストリーム・ディスクリプティブ・データを使用して、新規のランダム・ストリームが作成される。(I/O エラーや無効ファイル形式などが原因で) ストリームが読み取れない場合、*errstatus* は非ゼロである。ランダム・ストリームをファイルに保存するには、[SaveStreamF](#) 関数を使用する。

## 入力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>fname</i>	CHARACTER(*), INTENT(IN)	char*	<i>Fortran</i> の場合 : C 形式の NULL で終了する文字列として指定されたファイル名。 C の場合 : Fortran 形式の文字列として指定されたファイル名。

## 出力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>stream</i>	TYPE(VSL_STREAM_STATE), INTENT(OUT)	VSLStreamStatePtr*	<i>Fortran</i> の場合 : 新規ランダム・ストリームのディスクリプタ C の場合 : 新規ランダム・ストリームへのポインタ
<i>errstatus</i>	INTEGER	int	演算のエラー・ステータス。

## 戻り値

VSL\_ERROR\_OK, VSL\_STATUS\_OK

正常に終了したことを示す。

VSL\_ERROR\_NULL\_PTR

*fname* が NULL ポインタである。

VSL\_ERROR\_FILE\_OPEN

ファイルを開く際にエラーが発生したことを示す。

VSL\_ERROR\_FILE\_WRITE

ファイルへ書き込む際にエラーが発生したことを示す。

VSL\_ERROR\_FILE\_CLOSE

ファイルを閉じる際にエラーが発生したことを示す。

VSL\_ERROR\_MEM\_FAILURE

システムが、内部ニーズに対してメモリを割り当てることできない。

---

VSL_ERROR_BAD_FILE_FORMAT	ファイル形式が不明である。
VSL_ERROR_UNSUPPORTED_FILE_VER	サポートされていないファイル形式のバージョンである。

---

## LeapfrogStream

リープフロッグ法を用いてストリームを初期化する。

---

### 構文

#### Fortran:

```
status = vslleapfrogstream( stream, k, nstreams )
```

#### C:

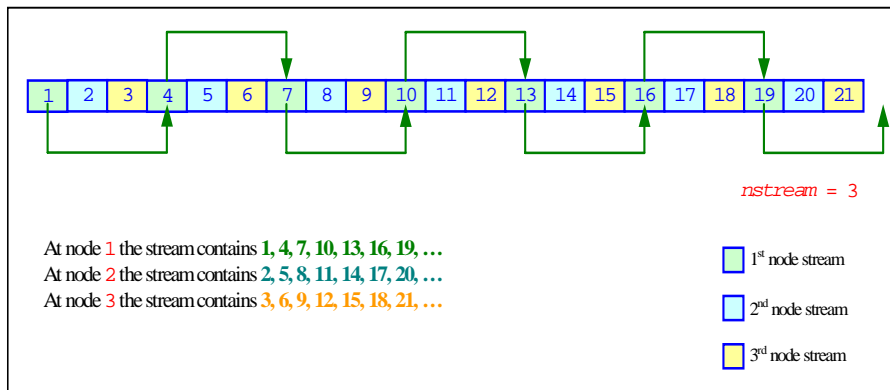
```
status = vslLeapfrogStream( stream, k, nstreams );
```

### 説明

この関数は、単位ストライドではないランダム・ストリームで乱数を生成する。この機能は、元のランダム・シーケンスの生成とそれに続くマニュアル分配をせずに、元のストリームから *nstreams* バッファに乱数を分配するのに特に役立つ。リープフロッグ法の重要な適用方法の1つは、元のシーケンスを *nstreams* 処理ノードにまたがるオー

オーバーラップしないサブシーケンスに分割することである。この関数は、元のランダム・ストリーム (図 10-1 を参照) を初期化し、処理ノード  $k$  に対して乱数を生成する。 $0 \leq k < nstreams$ 、 $nstreams$  は対象の処理ノードの最大値である。

図 10-1 リープフロッグ法



リープフロッグ法は、リープフロッグ法で成分の分割が可能な基本生成器に対してのみサポートされている。リープフロッグ法を使用すると、処理ノードへのマニュアル分配が伴う、生成器による生成よりも効率が良い。詳細は [VSL Notes](#) を参照のこと。

準乱数基本生成器では、リープフロッグ法は準乱数ベクトル全体ではなく、個別のコンポーネントを生成することができる。この場合、 $nstreams$  パラメータは準乱数ベクトルの次元に等しく、 $k$  パラメータは生成されるコンポーネントのインデックスでなければならない ( $0 \leq k < nstreams$ )。他のパラメータ値は認められない。



以下に、リープフロッグ法を用いて3つの独立したストリームを初期化するコード例を示す。

---

**例 10-1      リープフロッグ法の FORTRAN コード**

---

```
...
TYPE(VSL_STREAM_STATE)      ::stream1
TYPE(VSL_STREAM_STATE)      ::stream2
TYPE(VSL_STREAM_STATE)      ::stream3

!Creating 3 identical streams
status = vslnewstream(stream1, VSL_BRNG_MCG31, 174)
status = vslcopystream(stream2, stream1)
status = vslcopystream(stream3, stream1)

!Leapfrogging the streams
status = vslleapfrogstream(stream1, 0, 3)
status = vslleapfrogstream(stream2, 1, 3)
status = vslleapfrogstream(stream3, 2, 3)

!Generating random numbers
...
!Deleting the streams
status = vsldeletestream(stream1)
status = vsldeletestream(stream2)
status = vsldeletestream(stream3)
...
```

---

---

**例 10-2      リープフロッグ法の C コード**

---

```
...
VSLStreamStatePtr stream1;
VSLStreamStatePtr stream2;
VSLStreamStatePtr stream3;

/* Creating 3 identical streams */
status = vslNewStream(&stream1, VSL_BRNG_MCG31, 174);
status = vslCopyStream(&stream2, stream1);
status = vslCopyStream(&stream3, stream1);

/* Leapfrogging the streams */
status = vslLeapfrogStream(stream1, 0, 3);
status = vslLeapfrogStream(stream2, 1, 3);
status = vslLeapfrogStream(stream3, 2, 3);

/* Generating random numbers */
...
/* Deleting the streams */
```

---

## 例 10-2 リープフロッグ法の C コード ( 続き )

```
status = vslDeleteStream(&stream1);
status = vslDeleteStream(&stream2);
status = vslDeleteStream(&stream3);
```

...

### 入力パラメータ

名前	データ型		説明
	FORTRAN	C	
<i>stream</i>	TYPE(VSL_STREAM_STATE), INTENT( IN)	VSLStreamStatePtr	<i>Fortran</i> の場合 : リープフロッグ法を適用するストリームのディスクリプタ <i>C</i> の場合 : リープフロッグ法を適用するストリーム・ステート構造体へのポインタ
<i>k</i>	INTEGER, INTENT( IN)	int	処理ノードのインデックス、またはストリーム番号
<i>nstreams</i>	INTEGER, INTENT( IN)	int	処理ノードまたはストライドの最大値

### 戻り値

VSL_ERROR_OK, VSL_STATUS_OK	正常に終了したことを示す。
VSL_ERROR_NULL_PTR	<i>stream</i> が NULL ポインタである。
VSL_ERROR_BAD_STREAM	<i>stream</i> が有効なランダム・ストリームでない。
VSL_ERROR_LEAPFROG_UNSUPPORTED	BRNG がリープフロッグ法をサポートしていない。

## SkipAheadStream

ブロック分割法を用いてストリームを初期化する。

### 構文

#### Fortran:

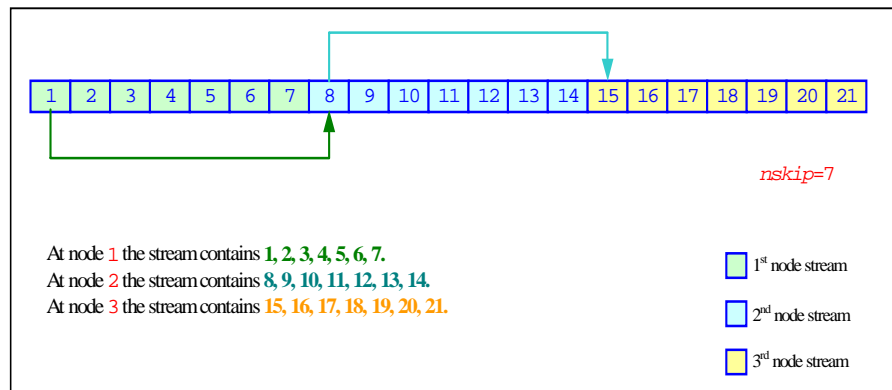
```
status = vslskipaheadstream( stream, nskip )
```

**C:**

```
status = vslSkipAheadStream( stream, nskip );
```

**説明**

この関数は、ランダム・ストリームで指定された数の成分をスキップする。この機能は、元のランダム・ストリームから異なる処理ノードに乱数を分配するのに特に役立つ。処理ノードで使用する乱数の最大値が *nskip* の場合、元のランダム・シーケンスは、各ブロックがそれぞれの処理ノードに対応するように、SkipAheadStream によってサイズ *nskip* のオーバーラップしないブロックに分割される。処理ノードの値に制限はない。これは、ブロック分割法または Skip-Ahead 法と呼ばれている（[図 10-2](#) を参照）。

**図 10-2**      **ブロック分割法**

Skip-Ahead 法は、Skip-Ahead 法で成分をスキップすることができる基本生成器に対してのみサポートされている。Skip-Ahead 法を使用すると、処理ノードへのマニュアル・スキップが伴う、生成器による生成よりも効率が良い。詳細は [VSL Notes](#) を参照のこと。

準乱数基本生成器では、Skip-Ahead 法は準乱数ベクトル全体ではなく、コンポーネントと共に動作する点に注意する。このため、NS 準乱数ベクトルをスキップするには、*nskip* パラメータを  $NS \times DIMEN$  と等しくなるように設定する ( $DIMEN$  は準乱数ベクトルの次元)。

以下に、SkipAheadStream 関数を用いて 3 つの独立したストリームを初期化するコード例を示す。

## 例 10-3      ブロック分割法の FORTRAN コード

---

```
...
TYPE(VSL_STREAM_STATE)    ::stream1
TYPE(VSL_STREAM_STATE)    ::stream2
TYPE(VSL_STREAM_STATE)    ::stream3

!Creating the 1st stream
status = vslnewstream(stream1, VSL_BRNG_MCG31, 174)

!Skipping ahead by 7 elements the 2nd stream
status = vslcopystream(stream2, stream1);
status = vslskipaheadstream(stream2, 7);

!Skipping ahead by 7 elements the 3rd stream
status = vslcopystream(stream3, stream2);
status = vslskipaheadstream(stream3, 7);

!Generating random numbers
...
!Deleting the streams
status = vsldeletestream(stream1)
status = vsldeletestream(stream2)
status = vsldeletestream(stream3)
...
```

---

## 例 10-4      ブロック分割法の C コード

---

```
VSLStreamStatePtr stream1;
VSLStreamStatePtr stream2;
VSLStreamStatePtr stream3;

/* Creating the 1st stream */
status = vslNewStream(&stream1, VSL_BRNG_MCG31, 174);

/* Skipping ahead by 7 elements the 2nd stream */
status = vslCopyStream(&stream2, stream1);
status = vslSkipAheadStream(stream2, 7);

/* Skipping ahead by 7 elements the 3rd stream */
status = vslCopyStream(&stream3, stream2);
status = vslSkipAheadStream(stream3, 7);

/* Generating random numbers */
...
/* Deleting the streams */
status = vslDeleteStream(&stream1);
```

---

例 10-4      ブロック分割法の C コード ( 続き )

```
status = vslDeleteStream(&stream2);
status = vslDeleteStream(&stream3);
...
```

入力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>stream</i>	TYPE(VSL_STREAM_STATE), INTENT(IN)	VSLStreamStatePtr	Fortran の場合 : ブロック分割法を適用するストリームのディスクリプタ C の場合 : ブロック分割法を適用するストリーム・ステート構造体へのポインタ
<i>nskip</i>	INTEGER, INTENT(IN)	int	スキップする成分数。

戻り値

VSL_ERROR_OK, VSL_STATUS_OK	正常に終了したことを示す。
VSL_ERROR_NULL_PTR	<i>stream</i> が NULL ポインタである。
VSL_ERROR_BAD_STREAM	<i>stream</i> が有効なランダム・ストリームでない。
VSL_ERROR_SKIPAHEAD_UNSUPPORTED	BRNG が Skip-Ahead 法をサポートしていない。

GetStreamStateBrng

指定されたランダム・ストリームを生成した基本生成器のインデックス値を返す。

構文

Fortran:

```
brng = vslgetstreamstatebrng( stream )
```

C:

```
brng = vslGetStreamStateBrng( stream );
```

## 説明

この関数は、指定されたランダム・ストリームの生成に使用された基本生成器のインデックスを返す。

## 入力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>stream</i>	TYPE(VSL_STREAM_STATE), INTENT(IN)	VSLStreamStatePtr	<i>Fortran</i> の場合：ストリーム・ステートのディスクリプタ <i>C</i> の場合：ストリーム・ステート構造体へのポインタ

## 出力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>brng</i>	INTEGER	int	<i>stream</i> の生成に使用された基本生成器のインデックス値。値が負の場合はエラーを表す。

## 戻り値

VSL\_ERROR\_NULL\_PTR

*stream* が NULL ポインタである。

VSL\_ERROR\_BAD\_STREAM

*stream* が有効なランダム・ストリームでない。

---

## GetNumRegBrngs

現在登録されている基本生成器の数を取得する。

---

## 構文

**Fortran:**

```
nregbrngs = vslgetnumregbrngs( )
```

**C:**

```
nregbrngs = vslGetNumRegBrngs( void );
```

## 説明

この関数は現在登録されている基本生成器の数を取得する。ユーザがユーザ定義生成器を登録すると、登録されている基本生成器の数はインクリメントされる。登録できる基本生成器の上限は `VSL_MAX_REG_BRNGS` によって決まる。

## 出力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<code>nregbrngs</code>	INTEGER	int	関数呼び出しを発した時点で登録されている基本生成器の数

## 分布生成器

ここでは複数の分布タイプに対応した乱数生成 `VSL` ルーチンについて説明する。それぞれの関数は基本となる分布タイプによってグループに分けられ、機能概要に加えて `FORTTRAN` と `C` インターフェイスの呼び出し手順仕様および入力と出力パラメータの説明が記載される。

乱数生成器ルーチンを、データ型、出力分布、生成器ルーチンのデータ型と呼び出された基本乱数生成器の対応のセットと併せて、[表 10-8](#) と [表 10-9](#) に一覧表示した。

**表 10-8 連続分布生成器**

分布のタイプ	データ型	BRNG データ型	説明
<a href="#">Uniform</a>	s, d	s, d	区間 [a,b] に対する連続一様分布
<a href="#">Gaussian</a>	s, d	s, d	正規 ( ガウス ) 分布
<a href="#">GaussianMV</a>	s, d	s, d	他変量正規 ( ガウス ) 分布
<a href="#">Exponential</a>	s, d	s, d	指数分布
<a href="#">Laplace</a>	s, d	s, d	ラプラス分布 ( 両側指数分布 )
<a href="#">Weibull</a>	s, d	s, d	ワイブル分布
<a href="#">Cauchy</a>	s, d	s, d	コーシー分布
<a href="#">Rayleigh</a>	s, d	s, d	レイリー分布
<a href="#">Lognormal</a>	s, d	s, d	対数正規分布
<a href="#">Gumbel</a>	s, d	s, d	ガンベル ( 極値 ) 分布
<a href="#">Gamma</a>	s, d	s, d	ガンマ分布
<a href="#">Beta</a>	s, d	s, d	ベータ分布

**表 10-9 離散分布生成器**

分布のタイプ	データ型	BRNG データ型	説明
<a href="#">Uniform</a>	i	d	区間 [a、b) に対する離散一様分布
<a href="#">UniformBits</a>	i	i	一様ビット分布を持つ整数乱数生成器
<a href="#">Bernoulli</a>	i	s	ベルヌーイ分布
<a href="#">Geometric</a>	i	s	幾何分布
<a href="#">Binomial</a>	i	d	二項分布
<a href="#">Hypergeometric</a>	i	d	超幾何分布
<a href="#">Poisson</a>	i	s (VSL_METHOD_IPOISSON_POISNORM の場合 ) s ( 分布パラメータが $\lambda \geq 27$ の場合 )、 d ( $\lambda < 27$ の場合 ) (VSL_METHOD_IPOISSON_PTPE の場合 )	ポワソン分布
<a href="#">PoissonV</a>	i	s	多様な平均値を持つポアソン分布
<a href="#">NegBinomial</a>	i	d	負の二項分布、またはパスカル分布

## 連続分布

連続分布の乱数を生成するルーチンについて説明する。

## Uniform

一様分布に従う乱数を生成する。

### 構文

#### Fortran:

```
status = vsrnguniform( method, stream, n, r, a, b )
status = vdrnguniform( method, stream, n, r, a, b )
```



C:

```
status = vsRngUniform( method, stream, n, r, a, b );
status = vdRngUniform( method, stream, n, r, a, b );
```

### 説明

この関数は、区間  $[a, b]$  を持つ一様分布の乱数を生成する。  
 $a, b$  は、それぞれ区間の始まりと終わりであり、 $a, b \in R; a < b$ 。

確率密度関数は次のように示される。

$$f_{a,b}(x) = \begin{cases} \frac{1}{b-a}, & x \in [a, b] \\ 0, & x \notin [a, b] \end{cases}, -\infty < x < +\infty$$

累積分布関数は次のように示される。

$$F_{a,b}(x) = \begin{cases} 0, & x < a \\ \frac{x-a}{b-a}, & a \leq x < b \\ 1, & x \geq b \end{cases}, -\infty < x < +\infty$$

### 入力パラメータ

名前	データ型		説明
	FORTRAN	C	
<i>method</i>	INTEGER, INTENT(IN)	int	生成方法。このパラメータはダミーで一様分布の場合は 0 を設定する。特定の値は次のとおりである。 VSL_METHOD_SUNIFORM_STD VSL_METHOD_DUNIFORM_STD 標準生成法。現時点で、この分布生成器に対する method は 1 つのみ。
<i>stream</i>	TYPE (VSL_STREAM_STATE), INTENT(IN)	VSLStreamStatePtr	Fortran の場合：ストリーム・ステート構造体のディスクリプタ。 C の場合：ストリーム・ステート構造体へのポインタ。
<i>n</i>	INTEGER, INTENT(IN)	int	生成させる乱数の個数。

名前	データ型		説明
	FORTTRAN	C	
<i>a</i>	REAL, INTENT(IN) (vsrnguniform の場合)。 DOUBLE PRECISION, INTENT(IN) (vdrnguniform の 場合)。	float (vsRngUniform の場合)。 double (vdRngUniform の場 合)。	区間の開始 <i>a</i> 。
<i>b</i>	REAL, INTENT(IN) (vsrnguniform の場合)。 DOUBLE PRECISION, INTENT(IN) (vdrnguniform の 場合)。	float (vsRngUniform の場合)。 double (vdRngUniform の場 合)。	区間の終了 <i>b</i> 。

## 出力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>r</i>	REAL, INTENT(OUT) (vsrnguniform の場合)。 DOUBLE PRECISION, INTENT(OUT) (vdrnguniform の 場合)	float* (vsRngUniform の場 合)。 double* (vdRngUniform の場 合)	区間 [ <i>a</i> , <i>b</i> ] を持つ一様分布に従っ た <i>n</i> 個の乱数のベクトル。

## 戻り値

VSL_ERROR_OK, VSL_STATUS_OK	正常に終了したことを示す。
VSL_ERROR_NULL_PTR	<i>stream</i> が NULL ポインタである。
VSL_ERROR_BAD_STREAM	<i>stream</i> が有効なランダム・ストリームでない。
VSL_ERROR_BAD_UPDATE	抽象 BRNG のコールバック関数が、バッファ内の 更新された成分の数として無効な値 (< 0 または > <i>nmax</i> ) を返した。
VSL_ERROR_NO_NUMBERS	抽象 BRNG のコールバック関数が、バッファ内の 更新された成分の数としてゼロを返した。

## Gaussian

正規分布に従う乱数を生成する。

### 構文

#### Fortran:

```
status = vsrnggaussian( method, stream, n, r, a, sigma )
status = vdrnggaussian( method, stream, n, r, a, sigma )
```

#### C:

```
status = vsRngGaussian( method, stream, n, r, a, sigma );
status = vdRngGaussian( method, stream, n, r, a, sigma );
```

### 説明

この関数は、平均値  $a$ 、標準偏差  $\sigma$  の正規 ( ガウス ) 分布の乱数を生成する。ここで、 $a, \sigma \in R; \sigma > 0$ 。

確率密度関数は次のように示される。

$$f_{a, \sigma}(x) = \frac{1}{\sigma \sqrt{2\pi}} \exp\left(-\frac{(x-a)^2}{2\sigma^2}\right), -\infty < x < +\infty$$

累積分布関数は次のように示される。

$$F_{a, \sigma}(x) = \int_{-\infty}^x \frac{1}{\sigma \sqrt{2\pi}} \exp\left(-\frac{(y-a)^2}{2\sigma^2}\right) dy, -\infty < x < +\infty$$

累積分布関数  $F_{a, \sigma}(x)$  は、標準的な正規分布  $\Phi(x)$  を用いて次のようにも表せる

$$F_{a, \sigma}(x) = \Phi((x-a)/\sigma)$$

## 入力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>method</i>	INTEGER, INTENT(IN)	int	生成方法。特定の値は次のとおりである。 VSL_METHOD_SGAUSSIAN_BOXMULLER VSL_METHOD_SGAUSSIAN_BOXMULLER2 VSL_METHOD_DGAUSSIAN_BOXMULLER VSL_METHOD_DGAUSSIAN_BOXMULLER2 <a href="#">表 10-1</a> の BOXMULLER と BOXMULLER2 の説明を参照。
<i>stream</i>	TYPE (VSL_STREAM_STATE), INTENT(IN)	VSLStreamStatePtr	<i>Fortran</i> の場合: ストリーム・ステート構造体のディスクリプタ。 <i>C</i> の場合: ストリーム・ステート構造体へのポインタ。
<i>n</i>	INTEGER, INTENT(IN)	int	生成させる乱数の個数。
<i>a</i>	REAL, INTENT(IN) (vsrnggaussian の場合) DOUBLE PRECISION, INTENT(IN) (vdrnggaussian の場合)	float (vsRngGaussian の場合) double (vdRngGaussian の場合)	平均値 $a$ 。
<i>sigma</i>	REAL, INTENT(IN) (vsrnggaussian の場合) DOUBLE PRECISION, INTENT(IN) (vdrnggaussian の場合)	float (vsRngGaussian の場合) double (vdRngGaussian の場合)	標準偏差 $\sigma$

## 出力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>r</i>	REAL, INTENT(OUT) (vsrnggaussian の場合) DOUBLE PRECISION, INTENT(OUT) (vdrnggaussian の場合)	float* (vsRngGaussian の場合) double* (vdRngGaussian の場合)	正規分布に従った $n$ 個の乱数のベクトル。

## 戻り値

VSL_ERROR_OK, VSL_STATUS_OK	正常に終了したことを示す。
VSL_ERROR_NULL_PTR	<i>stream</i> が NULL ポインタである。
VSL_ERROR_BAD_STREAM	<i>stream</i> が有効なランダム・ストリームでない。
VSL_ERROR_BAD_UPDATE	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数として無効な値 (< 0 または > nmax) を返した。
VSL_ERROR_NO_NUMBERS	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数としてゼロを返した。

## GaussianMV

多変量正規分布から乱数を生成する。

## 構文

## Fortran:

```
status = vsrnggaussianmv( method, stream, n, r, dimen, mstorage, a, t )
status = vdrnggaussianmv( method, stream, n, r, dimen, mstorage, a, t )
```

## C:

```
status = vsRngGaussianMV( method, stream, n, r, dimen, mstorage, a, t );
status = vdRngGaussianMV( method, stream, n, r, dimen, mstorage, a, t );
```

## 説明

この関数は、平均値  $\mathbf{a}$ 、分散共分散行列  $\mathbf{C}$  の  $d$ -変量正規 ( ガウス ) 分布の乱数を生成する。ここで、

$\mathbf{a} \in R^d$ 、 $\mathbf{C}$  は  $d \times d$  対象正定値行列。

確率密度関数は次のように示される。

$$f_{\mathbf{a}, \mathbf{C}}(\mathbf{x}) = \frac{1}{\sqrt{\det(2\pi\mathbf{C})}} \exp(-1/2(\mathbf{x} - \mathbf{a})^T \mathbf{C}^{-1}(\mathbf{x} - \mathbf{a})), \mathbf{x} \in R^d$$

行列  $C$  は  $C = TT^T$  として表すことができる。ここで、 $T$  は  $C$  のコレスキー因子である下三角行列。

分散共分散行列  $C$  の代わりに、生成ルーチンは入力  $C$  のコレスキー因子を要求する。行列  $C$  のコレスキー因子を計算するには、行列の因子分解用の MKL LAPACK ルーチン呼び出すとよい。v?RngGaussianMV/v?rnggaussianmv ルーチンの場合、[?potrf](#) または [?pptrf](#) (? は、s (単精度) または d (倍精度) を意味する)。詳細は、[アプリケーション・ノート](#) を参照。

## 入カパラメータ

名前	データ型		説明
	FORTRAN	C	
<i>method</i>	INTEGER, INTENT(IN)	int	生成方法。特定の値は次のとおりである。 VSL_METHOD_SGAUSSIANMV_BOXMULLER VSL_METHOD_SGAUSSIANMV_BOXMULLER2 VSL_METHOD_DGAUSSIANMV_BOXMULLER VSL_METHOD_DGAUSSIANMV_BOXMULLER2 <a href="#">表 10-1</a> の BOXMULLER と BOXMULLER2 の説明を参照。
<i>stream</i>	TYPE (VSL_STREAM_STATE), INTENT(IN)	VSLStreamStatePtr	Fortran の場合: ストリーム・ステート構造体のディスクリプタ。 C の場合: ストリーム・ステート構造体へのポインタ。
<i>n</i>	INTEGER, INTENT(IN)	int	生成させる乱数の個数。
<i>dimen</i>	INTEGER, INTENT(IN)	int	出力乱数ベクトルの次元 $d$ ( $d \geq 1$ )

<i>mstorage</i>	INTEGER, INTENT(IN)	int	<p><i>Fortran</i> の場合：上三角行列 <math>T^T</math> の行列格納形式。このルーチンは 3 つの行列格納形式をサポートしている。</p> <ul style="list-style-type: none"> <li>VSL_MATRIX_STORAGE_FULL – 行列 <math>T^T</math> のすべての <math>d \times d</math> 成分が渡されるが、実際にルーチンで使用されるのは上三角部分のみ。</li> <li>VSL_MATRIX_STORAGE_PACKED – <math>T^T</math> の上三角成分が、行ごとに 1 次元配列に圧縮される。</li> <li>VSL_MATRIX_STORAGE_DIAGONAL – <math>T^T</math> の対角成分のみ渡される。</li> </ul> <p><i>C</i> の場合：下三角行列 <math>T</math> の行列格納形式。このルーチンは 3 つの行列格納形式をサポートしている。</p> <ul style="list-style-type: none"> <li>VSL_MATRIX_STORAGE_FULL – 行列 <math>T</math> のすべての <math>d \times d</math> 成分が渡されるが、実際にルーチンで使用されるのは下三角部分のみ。</li> <li>VSL_MATRIX_STORAGE_PACKED – <math>T</math> の下三角成分が、行ごとに 1 次元配列に圧縮される。</li> <li>VSL_MATRIX_STORAGE_DIAGONAL – <math>T</math> の対角成分のみ渡される。</li> </ul>
<i>a</i>	REAL, INTENT(IN) (vsrnggaussianmv の場合) DOUBLE PRECISION, INTENT(IN) (vdrnggaussianmv の場合)	float* (vsRngGaussianMV の場合) double* for vdRngGaussianMV の場合)	次元 $d$ の平均値ベクトル <i>a</i>
<i>t</i>	REAL, INTENT(IN) (vsrnggaussianmv の場合) DOUBLE PRECISION, INTENT(IN) (vdrnggaussianmv の場合)	float* (vsRngGaussianMV の場合) double* for vdRngGaussianMV の場合)	<p><i>Fortran</i> の場合：上三角行列の成分は、行列 <math>T^T</math> の格納形式 <i>mstorage</i> に従って渡される。</p> <p><i>C</i> の場合：下三角行列の成分は、行列 <math>T</math> の格納形式 <i>mstorage</i> に従って渡される。</p>

## 出力パラメータ

名前	データ型		説明
	FORTRAN	C	
r	REAL, INTENT(OUT) (vsrnggaussianmv の場合) DOUBLE PRECISION, INTENT(OUT) (vdrnggaussianmv の場合)	float* (vsRngGaussianMV の 場合) double* for vdRngGaussianMV の 場合)	次元 <i>dimen</i> の <i>n</i> 個の乱数ベクトルの 配列

## アプリケーション・ノート

行列は Fortran では列ごとに、C では行ごとに格納されるため、MKL 因子分解ルーチン (Fortran 行列格納形式) を多変量正規 RNG (C 行列格納形式) と一緒に使用する場合、C と Fortran では使用方法がやや異なる。次の表は、C と Fortran でこれらのルーチンを使用する際に役立つ。詳細は、適切な VSL example ファイルを参照のこと。

表 10-10 Fortran におけるコレスキー因子分解ルーチンの使用

行列の格納形式	分散共分散行列引数	因子分解ルーチン	因子分解 ルーチンに おける UPLO パラ メータ	RNG の入力引数 としての因子分 解の結果
VSL_MATRIX_STORAGE_FULLL	Fortran 2 次元配列で の C	spotrf (vsrnggaussianmv の場合) dpotrf (vdrnggaussianmv の場合)	'U'	$T^T$ の上三角。 下三角は使用され ない。
VSL_MATRIX_STORAGE_PACKED	列ごとに 1 次元配列に 圧縮された C の下三 角部分。	spptrf (vsrnggaussianmv の場合) dpptrf (vdrnggaussianmv の場合)	'L'	列ごとに 1 次元 配列に圧縮され た $T^T$ の上三角 部分。



表 10-11 C におけるコレスキー因子分解ルーチンの使用

行列の格納形式	分散共分散行列引数	因子分解ルーチン	因子分解ルーチンにおける UPLO パラメータ	RNG の入力引数としての因子分解の結果
VSL_MATRIX_STORAGE_FULLL	C の 2 次元配列での C	spotrf (vsRngGaussianM v の場合 ) dpotrf (vdRngGaussianM v の場合 )	'U'	$T^T$ の上三角。 下三角は使用 されない。
VSL_MATRIX_STORAGE_PACKED	列ごとに 1 次元配列に 圧縮された C の下三角 部分。	sppturf (vsRngGaussianM v の場合 ) dppturf (vdRngGaussianM v の場合 )	'L'	列ごとに 1 次 元配列に圧縮 された $T^T$ の 上三角部分。

戻り値

VSL_ERROR_OK, VSL_STATUS_OK	正常に終了したことを示す。
VSL_ERROR_NULL_PTR	<i>stream</i> が NULL ポインタである。
VSL_ERROR_BAD_STREAM	<i>stream</i> が有効なランダム・ストリームでない。
VSL_ERROR_BAD_UPDATE	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数として無効な値 (< 0 または > nmax) を返した。
VSL_ERROR_NO_NUMBERS	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数としてゼロを返した。

## Exponential

指数分布に従う乱数を生成する。

---

### 構文

#### Fortran:

```
status = vsrngexponential( method, stream, n, r, a, beta )  
status = vdrngexponential( method, stream, n, r, a, beta )
```

#### C:

```
status = vsRngExponential( method, stream, n, r, a, beta );  
status = vdRngExponential( method, stream, n, r, a, beta );
```

### 説明

この関数は、位置母数  $a$ 、尺度母数  $\beta$  を持つ指数分布の乱数を生成する。ここで、 $a, \beta \in R; \beta > 0$ 。

確率密度関数は次のように示される。

$$f_{a,\beta}(x) = \begin{cases} \frac{1}{\beta} \exp(-(x-a)/\beta), & x \geq a \\ 0, & x < a \end{cases}, -\infty < x < +\infty$$

累積分布関数は次のように示される。

$$F_{a,\beta}(x) = \begin{cases} 1 - \exp(-(x-a)/\beta), & x \geq a \\ 0, & x < a \end{cases}, -\infty < x < +\infty$$

## 入力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>method</i>	INTEGER, INTENT( IN)	int	生成方法。特定の値は次のとおりである。 VSL_METHOD_SEXPONENTIAL_ICDF VSL_METHOD_DEXPONENTIAL_ICDF 逆累積分布関数法
<i>stream</i>	TYPE (VSL_STREAM_STATE), INTENT( IN)	VSLStreamStatePtr	Fortran の場合 : ストリーム・ステート構造体のディスクリプタ。 C の場合 : ストリーム・ステート構造体へのポインタ。
<i>n</i>	INTEGER, INTENT( IN)	int	生成させる乱数の個数。
<i>a</i>	REAL, INTENT( IN) (vsrngexponential の場合 ) DOUBLE PRECISION, INTENT( IN) (vdrngexponential の場合 )	float (vsRngExponential の場合 ) double (vdRngExponential の場合 )	位置母数 $a$ 。
<i>beta</i>	REAL, INTENT( IN) (vsrngexponential の場合 ) DOUBLE PRECISION, INTENT( IN) (vdrngexponential の場合 )	float (vsRngExponential の場合 ) double (vdRngExponential の場合 )	尺度母数 $\beta$ 。

## 出力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>r</i>	REAL, INTENT( OUT) (vsrngexponential の場合 ) DOUBLE PRECISION, INTENT( OUT) (vdrngexponential の場合 )	float* (vsRngExponential の場合 ) double* (vdRngExponential の場合 )	指数分布に従った $n$ 個の乱数のベクトル。

## 戻り値

VSL_ERROR_OK, VSL_STATUS_OK	正常に終了したことを示す。
VSL_ERROR_NULL_PTR	<i>stream</i> が NULL ポインタである。
VSL_ERROR_BAD_STREAM	<i>stream</i> が有効なランダム・ストリームでない。
VSL_ERROR_BAD_UPDATE	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数として無効な値 (< 0 または > nmax) を返した。
VSL_ERROR_NO_NUMBERS	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数としてゼロを返した。

---

## Laplace

ラプラス分布に従う乱数を生成する。

---

### 構文

#### Fortran:

```
status = vsrnglaplace( method, stream, n, r, a, beta )
status = vdrnglaplace( method, stream, n, r, a, beta )
```

#### C:

```
status = vsRngLaplace( method, stream, n, r, a, beta );
status = vdRngLaplace( method, stream, n, r, a, beta );
```

### 説明

この関数は、平均値  $a$ 、尺度母数  $\beta$  を持つラプラス分布の乱数を生成する。ここで、 $a, \beta \in \mathbb{R}; \beta > 0$ 。尺度母数から標準偏差が次のように決まる

$$\sigma = \beta\sqrt{2}$$

確率密度関数は次のように示される。

$$f_{a, \beta}(x) = \frac{1}{\sqrt{2}\beta} \exp\left(-\frac{|x-a|}{\beta}\right), -\infty < x < +\infty$$

累積分布関数は次のように示される。

$$F_{a,\beta}(x) = \begin{cases} \frac{1}{2} \exp\left(-\frac{|x-a|}{\beta}\right), & x < a \\ 1 - \frac{1}{2} \exp\left(-\frac{|x-a|}{\beta}\right), & x \geq a \end{cases}, -\infty < x < +\infty$$

### 入力パラメータ

名前	データ型		説明
	FORTRAN	C	
<i>method</i>	INTEGER, INTENT(IN)	int	生成方法。特定の値は次のとおりである。 VSL_METHOD_SLAPLACE_ICDF VSL_METHOD_DLAPLACE_ICDF 逆累積分布関数法
<i>stream</i>	TYPE (VSL_STREAM_STATE), INTENT(IN)	VSLStreamStatePtr	<i>Fortran</i> の場合：ストリーム・ステート構造体のディスクリプタ。 <i>C</i> の場合：ストリーム・ステート構造体へのポインタ。
<i>n</i>	INTEGER, INTENT(IN)	int	生成させる乱数の個数。
<i>a</i>	REAL, INTENT(IN) (vsrnglaplace の場合) DOUBLE PRECISION, INTENT(IN) (vdrnglaplace の場合)	float (vsRngLaplace の場合) double (vdRngLaplace の場 合)	平均値 $a$ 。
<i>beta</i>	REAL, INTENT(IN) (vsrnglaplace の場合) DOUBLE PRECISION, INTENT(IN) (vdrnglaplace の場合)	float (vsRngLaplace の場合) double (vdRngLaplace の場 合)	尺度母数 $\beta$ 。

## 出力パラメータ

名前	データ型		説明
	FORTRAN	C	
<i>r</i>	REAL, INTENT(OUT) (vsrnglaplace の場合) DOUBLE PRECISION, INTENT(OUT) (vdrnglaplace の場合)	float* (vsRngLaplace の場合) double* (vdRngLaplace の場合)	ラプラス分布に従った <i>n</i> 個の乱数のベクトル。

## 戻り値

VSL_ERROR_OK, VSL_STATUS_OK	正常に終了したことを示す。
VSL_ERROR_NULL_PTR	<i>stream</i> が NULL ポインタである。
VSL_ERROR_BAD_STREAM	<i>stream</i> が有効なランダム・ストリームでない。
VSL_ERROR_BAD_UPDATE	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数として無効な値 (< 0 または > nmax) を返した。
VSL_ERROR_NO_NUMBERS	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数としてゼロを返した。

## Weibull

ワイブル分布に従う乱数を生成する。

## 構文

## Fortran:

```
status = vsrngweibull( method, stream, n, r, alpha, a, beta )
status = vdrngweibull( method, stream, n, r, alpha, a, beta )
```

## C:

```
status = vsRngWeibull( method, stream, n, r, alpha, a, beta );
status = vdRngWeibull( method, stream, n, r, alpha, a, beta );
```

## 説明

この関数は、位置母数  $a$ 、尺度母数  $\beta$ 、形状母数  $\alpha$  を持つワイブル分布の乱数を生成する。ここで、 $\alpha, \beta, a \in \mathbb{R}; \alpha > 0; \beta > 0$ 。

確率密度関数は次のように示される。

$$f_{a, \alpha, \beta}(x) = \begin{cases} \frac{\alpha}{\beta^\alpha} (x-a)^{\alpha-1} \exp\left(-\left(\frac{x-a}{\beta}\right)^\alpha\right), & x \geq a \\ 0, & x < a \end{cases}$$

累積分布関数は次のように示される。

$$F_{a, \alpha, \beta}(x) = \begin{cases} 1 - \exp\left(-\left(\frac{x-a}{\beta}\right)^\alpha\right), & x \geq a \\ 0, & x < a \end{cases}, -\infty < x < +\infty$$

## 入力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>method</i>	INTEGER, INTENT(IN)	int	生成方法。特定の値は次のとおりである。 VSL_METHOD_SWEIBULL_ICDF VSL_METHOD_DWEIBULL_ICDF 逆累積分布関数法
<i>stream</i>	TYPE (VSL_STREAM_STATE), INTENT(IN)	VSLStreamStatePtr	<i>Fortran</i> の場合：ストリーム・ステート構造体のディスクリプタ。 <i>C</i> の場合：ストリーム・ステート構造体へのポインタ。
<i>n</i>	INTEGER, INTENT(IN)	int	生成させる乱数の個数。
<i>alpha</i>	REAL, INTENT(IN) (vsrngweibull の場合) DOUBLE PRECISION, INTENT(IN) (vdrngweibull の場合)	float (vsRngWeibull の場合) double (vdRngWeibull の場合)	形状母数 $\alpha$ 。

名前	データ型		説明
	FORTTRAN	C	
<i>a</i>	REAL, INTENT(IN) (vsrngweibull の場合) DOUBLE PRECISION, INTENT(IN) (vdrngweibull の場合)	float (vsRngWeibull の場合) double (vdRngWeibull の場 合)	位置母数 $a$ 。
<i>beta</i>	REAL, INTENT(IN) (vsrngweibull の場合) DOUBLE PRECISION, INTENT(IN) (vdrngweibull の場合)	float (vsRngWeibull の場合) double (vdRngWeibull の場 合)	尺度母数 $\beta$ 。

### 出力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>r</i>	REAL, INTENT(OUT) (vsrngweibull の場合) DOUBLE PRECISION, INTENT(OUT) (vdrngweibull の場合)	float* (vsRngWeibull の 場合) double* (vdRngWeibull の 場合)	ワイブル分布に従った $n$ 個の乱数のベクトル。

### 戻り値

VSL\_ERROR\_OK, VSL\_STATUS\_OK

正常に終了したことを示す。

VSL\_ERROR\_NULL\_PTR

*stream* が NULL ポインタである。

VSL\_ERROR\_BAD\_STREAM

*stream* が有効なランダム・ストリームでない。

VSL\_ERROR\_BAD\_UPDATE

抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数として無効な値 ( $< 0$  または  $> nmax$ ) を返した。

VSL\_ERROR\_NO\_NUMBERS

抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数としてゼロを返した。



---

## Cauchy

コーシー分布に従う乱数を生成する。

---

### 構文

#### Fortran:

```
status = vsrngcauchy( method, stream, n, r, a, beta )  
status = vdrngcauchy( method, stream, n, r, a, beta )
```

#### C:

```
status = vsRngCauchy( method, stream, n, r, a, beta );  
status = vdRngCauchy( method, stream, n, r, a, beta );
```

### 説明

この関数は、位置母数  $a$ 、尺度母数  $\beta$  を持つコーシー分布の乱数を生成する。ここで、 $a, \beta \in R; \beta > 0$ 。

確率密度関数は次のように示される。

$$f_{a, \beta}(x) = \frac{1}{\pi \beta \left( 1 + \left( \frac{x-a}{\beta} \right)^2 \right)}, -\infty < x < +\infty$$

累積分布関数は次のように示される。

$$F_{a, \beta}(x) = \frac{1}{2} + \frac{1}{\pi} \arctan\left(\frac{x-a}{\beta}\right), -\infty < x < +\infty$$

## 入力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>method</i>	INTEGER, INTENT(IN)	int	生成方法。特定の値は次のとおりである。 VSL_METHOD_SCAUCHY_ICDF VSL_METHOD_DCAUCHY_ICDF 逆累積分布関数法
<i>stream</i>	TYPE (VSL_STREAM_STATE), INTENT(IN)	VSLStreamStatePtr	Fortran の場合 : ストリーム・ステート構造体のディスクリプタ。 C の場合 : ストリーム・ステート構造体へのポインタ。
<i>n</i>	INTEGER, INTENT(IN)	int	生成させる乱数の個数。
<i>a</i>	REAL, INTENT(IN) (vsrngcauchy の場合) DOUBLE PRECISION, INTENT(IN) (vdrngcauchy の場合)	float (vsRngCauchy の場合) double (vdRngCauchy の場合)	位置母数 $a$ 。
<i>beta</i>	REAL, INTENT(IN) (vsrngcauchy の場合) DOUBLE PRECISION, INTENT(IN) (vdrngcauchy の場合)	float (vsRngCauchy の場合) double (vdRngCauchy の場合)	尺度母数 $\beta$ 。

## 出力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>r</i>	REAL, INTENT(OUT) (vsrngcauchy の場合) DOUBLE PRECISION, INTENT(OUT) (vdrngcauchy の場合)	float* (vsRngCauchy の 場合) double* (vdRngCauchy の場 合)	コーシー分布に従った $n$ 個の乱数のベクトル。

## 戻り値

VSL_ERROR_OK, VSL_STATUS_OK	正常に終了したことを示す。
VSL_ERROR_NULL_PTR	<i>stream</i> が NULL ポインタである。
VSL_ERROR_BAD_STREAM	<i>stream</i> が有効なランダム・ストリームでない。
VSL_ERROR_BAD_UPDATE	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数として無効な値 ( $< 0$ または $> n_{\max}$ ) を返した。
VSL_ERROR_NO_NUMBERS	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数としてゼロを返した。

---

## Rayleigh

レイリー分布に従う乱数を生成する。

---

### 構文

#### Fortran:

```
status = vsrnggrayleigh( method, stream, n, r, a, beta )
status = vdrnggrayleigh( method, stream, n, r, a, beta )
```

#### C:

```
status = vsRngRayleigh( method, stream, n, r, a, beta );
status = vdRngRayleigh( method, stream, n, r, a, beta );
```

### 説明

この関数は、位置母数  $a$ 、尺度母数  $\beta$  を持つレイリー分布の乱数を生成する。ここで、 $a, \beta \in \mathbb{R}; \beta > 0$ 。

レイリー分布は [Weibull](#) 分布の一種であり、形状母数  $\alpha = 2$  のときである。

確率密度関数は次のように示される。

$$f_{a,\beta}(x) = \begin{cases} \frac{2(x-a)}{\beta^2} \exp\left(-\frac{(x-a)^2}{\beta^2}\right), & x \geq a \\ 0, & x < a \end{cases}, -\infty < x < +\infty$$

累積分布関数は次のように示される。

$$F_{a,\beta}(x) = \begin{cases} 1 - \exp\left(-\frac{(x-a)^2}{\beta^2}\right), & x \geq a \\ 0, & x < a \end{cases}, -\infty < x < +\infty$$

## 入カパラメータ

名前	データ型		説明
	FORTRAN	C	
<i>method</i>	INTEGER, INTENT(IN)	int	生成方法。特定の値は次のとおりである。 VSL_METHOD_SRAYLEIGH_ICDF VSL_METHOD_DRAYLEIGH_ICDF 逆累積分布関数法
<i>stream</i>	TYPE (VSL_STREAM_STATE), INTENT(IN)	VSLStreamStatePtr	<i>Fortran</i> の場合：ストリーム・ステート構造体のディスクリプタ。 <i>C</i> の場合：ストリーム・ステート構造体へのポインタ。
<i>n</i>	INTEGER, INTENT(IN)	int	生成させる乱数の個数。
<i>a</i>	REAL, INTENT(IN) (vsrnggrayleigh の場合) DOUBLE PRECISION, INTENT(IN) (vdrnggrayleigh の場合)	float (vsRngRayleigh の場合) double (vdRngRayleigh の場合)	位置母数 $a$ 。
<i>beta</i>	REAL, INTENT(IN) (vsrnggrayleigh の場合) DOUBLE PRECISION, INTENT(IN) (vdrnggrayleigh の場合)	float (vsRngRayleigh の場合) double (vdRngRayleigh の場合)	尺度母数 $\beta$ 。

## 出力パラメータ

名前	データ型		説明
	FORTRAN	C	
<i>r</i>	REAL, INTENT(OUT) (vsrnggrayleigh の場合) DOUBLE PRECISION, INTENT(OUT) (vdrnggrayleigh の場合)	float* (vsRngRayleigh の場合) double* (vdRngRayleigh の場合)	レイリー分布に従った <i>n</i> 個の乱数のベクトル。

## 戻り値

VSL_ERROR_OK, VSL_STATUS_OK	正常に終了したことを示す。
VSL_ERROR_NULL_PTR	<i>stream</i> が NULL ポインタである。
VSL_ERROR_BAD_STREAM	<i>stream</i> が有効なランダム・ストリームでない。
VSL_ERROR_BAD_UPDATE	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数として無効な値 (< 0 または > nmax) を返した。
VSL_ERROR_NO_NUMBERS	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数としてゼロを返した。

## Lognormal

対数正規分布に従う乱数を生成する。

## 構文

## Fortran:

```
status = vsrnglognormal( method, stream, n, r, a, sigma, b, beta )
status = vdrnglognormal( method, stream, n, r, a, sigma, b, beta )
```

## C:

```
status = vsRngLognormal( method, stream, n, r, a, sigma, b, beta );
status = vdRngLognormal( method, stream, n, r, a, sigma, b, beta );
```

## 説明

この関数は、分布の平均  $a$ 、正規分布の標準偏差  $\sigma$ 、位置母数  $b$ 、尺度母数  $\beta$  を持つ対数正規分布の乱数を生成する。ここで、 $a, \sigma, b, \beta \in \mathbb{R}$ ;  $\sigma > 0$ ;  $\beta > 0$ 。

確率密度関数は次のように示される。

$$f_{a, \sigma, b, \beta}(x) = \begin{cases} \frac{1}{\sigma(x-b)\sqrt{2\pi}} \exp\left(-\frac{[\ln((x-b)/\beta) - a]^2}{2\sigma^2}\right), & x > b \\ 0, & x \leq b \end{cases}$$

累積分布関数は次のように示される。

$$F_{a, \sigma, b, \beta}(x) = \begin{cases} \Phi((\ln((x-b)/\beta) - a)/\sigma), & x > b \\ 0, & x \leq b \end{cases}$$

## 入力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>method</i>	INTEGER, INTENT(IN)	int	生成方法。特定の値は次のとおりである。 VSL_METHOD_SLOGNORMAL_ICDF VSL_METHOD_DLOGNORMAL_ICDF 逆累積分布関数法
<i>stream</i>	TYPE (VSL_STREAM_STATE), INTENT(IN)	VSLStreamStatePtr	Fortran の場合: ストリーム・ステート構造体のディスクリプタ。 C の場合: ストリーム・ステート構造体へのポインタ。
<i>n</i>	INTEGER, INTENT(IN)	int	生成させる乱数の個数。
<i>a</i>	REAL, INTENT(IN) (vsrnglognormal の場合) DOUBLE PRECISION, INTENT(IN) (vdrnglognormal の場合)	float (vsRngLognormal の場合) double (vdRngLognormal の場合)	正規分布の平均 $a$ 。

名前	データ型		説明
	FORTTRAN	C	
<i>sigma</i>	REAL, INTENT(IN) (vsrnglognormal の場合) DOUBLE PRECISION, INTENT(IN) (vdrnglognormal の場合)	float (vsRngLognormal の場合) double (vdRngLognormal の場合)	正規分布 $\sigma$ の標準偏差。
<i>b</i>	REAL, INTENT(IN) (vsrnglognormal の場合) DOUBLE PRECISION, INTENT(IN) (vdrnglognormal の場合)	float (vsRngLognormal の場合) double (vdRngLognormal の場合)	位置母数 $b$ 。
<i>beta</i>	REAL, INTENT(IN) (vsrnglognormal の場合) DOUBLE PRECISION, INTENT(IN) (vdrnglognormal の場合)	float (vsRngLognormal の場合) double (vdRngLognormal の場合)	尺度母数 $\beta$ 。

### 出力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>r</i>	REAL, INTENT(OUT) (vsrnglognormal の場合) DOUBLE PRECISION, INTENT(OUT) (vdrnglognormal の場合)	float* (vsRngLognormal の場合) double* (vdRngLognormal の場合)	対数正規分布に従った $n$ 個の乱数のベクトル。

## 戻り値

VSL_ERROR_OK, VSL_STATUS_OK	正常に終了したことを示す。
VSL_ERROR_NULL_PTR	<i>stream</i> が NULL ポインタである。
VSL_ERROR_BAD_STREAM	<i>stream</i> が有効なランダム・ストリームでない。
VSL_ERROR_BAD_UPDATE	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数として無効な値 (< 0 または > nmax) を返した。
VSL_ERROR_NO_NUMBERS	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数としてゼロを返した。

---

## Gumbel

ガンベル分布に従う乱数を生成する。

---

## 構文

### Fortran:

```
status = vsrnggumbel( method, stream, n, r, a, beta )  
status = vdrnggumbel( method, stream, n, r, a, beta )
```

### C:

```
status = vsRngGumbel( method, stream, n, r, a, beta );  
status = vdRngGumbel( method, stream, n, r, a, beta );
```

## 説明

この関数は、位置母数  $a$ 、尺度母数  $\beta$  を持つガンベル分布の乱数を生成する。ここで、 $a, \beta \in R; \beta > 0$ 。

確率密度関数は次のように示される。

$$f_{a, \beta}(x) = \frac{1}{\beta} \exp\left(\frac{x-a}{\beta}\right) \exp(-\exp((x-a)/\beta)), \quad -\infty < x < +\infty$$



累積分布関数は次のように示される。

$$F_{a,\beta}(x) = 1 - \exp(-\exp((x-a)/\beta)) , \quad -\infty < x < +\infty$$

### 入力パラメータ

名前	データ型		説明
	FORTRAN	C	
<i>method</i>	INTEGER, INTENT(IN)	int	生成方法。特定の値は次のとおりである。 VSL_METHOD_SGUMBEL_ICDF VSL_METHOD_DGUMBEL_ICDF 逆累積分布関数法
<i>stream</i>	TYPE (VSL_STREAM_STATE), INTENT(IN)	VSLStreamStatePtr	Fortran の場合: ストリーム・ステート構造体のディスクリプタ C の場合: ストリーム・ステート構造体へのポインタ
<i>n</i>	INTEGER, INTENT(IN)	int	生成させる乱数の個数。
<i>a</i>	REAL, INTENT(IN) (vsrnggumbel の場合) DOUBLE PRECISION, INTENT(IN) (vdrnggumbel の場合)	float (vsRngGumbel の場合) double (vdRngGumbel の場合)	位置母数 $a$ 。
<i>beta</i>	REAL, INTENT(IN) (vsrnggumbel の場合) DOUBLE PRECISION, INTENT(IN) (vdrnggumbel の場合)	float (vsRngGumbel の場合) double (vdRngGumbel の場合)	尺度母数 $\beta$ 。

### 出力パラメータ

名前	データ型		説明
	FORTRAN	C	
<i>r</i>	REAL, INTENT(OUT) (vsrnggumbel の場合) DOUBLE PRECISION, INTENT(OUT) (vdrnggumbel の場合)	float* (vsRngGumbel の 場合) double* (vdRngGumbel の場 合)	ガンベル分布に従った $n$ 個の乱数のベクトル。

## 戻り値

VSL_ERROR_OK, VSL_STATUS_OK	正常に終了したことを示す。
VSL_ERROR_NULL_PTR	<i>stream</i> が NULL ポインタである。
VSL_ERROR_BAD_STREAM	<i>stream</i> が有効なランダム・ストリームでない。
VSL_ERROR_BAD_UPDATE	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数として無効な値 (< 0 または > nmax) を返した。
VSL_ERROR_NO_NUMBERS	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数としてゼロを返した。

## Gamma

ガンマ分布に従う乱数を生成する。

---

### 構文

#### Fortran:

```
status = vsrnggamma( method, stream, n, r, alpha, a, beta )
status = vdrnggamma( method, stream, n, r, alpha, a, beta )
```

#### C:

```
status = vsRngGamma( method, stream, n, r, alpha, a, beta );
status = vdRngGamma( method, stream, n, r, alpha, a, beta );
```

### 説明

この関数は、形状母数  $\alpha$ 、位置母数  $a$ 、尺度母数  $\beta$  を持つガンマ分布の乱数を生成する。 $\alpha, \beta, a \in \mathbb{R}$ 。 $\alpha > 0, \beta > 0$ 。

確率密度関数は次のように示される。

$$f_{\alpha, a, \beta}(x) = \begin{cases} \frac{1}{\Gamma(\alpha)\beta^\alpha} (x-a)^{\alpha-1} e^{-(x-a)/\beta}, & x \geq a \\ 0, & x < a \end{cases}, \quad -\infty < x < \infty,$$

$\Gamma(\alpha)$  は完全ガンマ関数である。

累積分布関数は次のように示される。

$$F_{\alpha, a, \beta}(x) = \begin{cases} \int_a^x \frac{1}{\Gamma(\alpha)\beta^\alpha} (y-a)^{\alpha-1} e^{-(y-a)/\beta} dy, & x \geq a \\ 0, & x < a \end{cases}, -\infty < x < \infty$$

## 入力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>method</i>	INTEGER, INTENT(IN)	int	生成方法。特定の値は次のとおりである。 VSL_METHOD_SGAMMA_GNORM VSL_METHOD_DGAMMA_GNORM ガウス分布に従った乱数を使用する受容 / 棄却法。 <a href="#">表 10-1</a> の GNORM の説明を参照。
<i>stream</i>	TYPE (VSL_STREAM_STATE), INTENT(IN)	VSLStreamStatePtr	<i>Fortran</i> の場合 : ストリーム・ステート構造体のディスクリプタ C の場合 : ストリーム・ステート構造体へのポインタ
<i>n</i>	INTEGER, INTENT(IN)	int	生成させる乱数の個数。
<i>alpha</i>	REAL, INTENT(IN) (vsrnggamma の場合) DOUBLE PRECISION, INTENT(IN) (vdrnggumbel の場合)	float (vsRngGamma の場合) double (vdRngGamma の場合)	形状母数 $\alpha$ 。
<i>a</i>	REAL, INTENT(IN) (vsrnggamma の場合) DOUBLE PRECISION, INTENT(IN) (vdrnggumbel の場合)	float (vsRngGamma の場合) double (vdRngGamma の場合)	位置母数 $a$ 。
<i>beta</i>	REAL, INTENT(IN) (vsrnggamma の場合) DOUBLE PRECISION, INTENT(IN) (vdrnggumbel の場合)	float (vsRngGamma の場合) double (vdRngGamma の場合)	尺度母数 $\beta$ 。

## 出力パラメータ

名前	データ型		説明
	FORTRAN	C	
<i>r</i>	REAL, INTENT(OUT) (vsrnggamma の場合) DOUBLE PRECISION, INTENT(OUT) (vdrnggamma の場合)	float* (vsRngGamma の場合) double* (vdRngGamma の場合)	ガンマ分布に従った <i>n</i> 個の乱数のベクトル。

## 戻り値

VSL_ERROR_OK, VSL_STATUS_OK	正常に終了したことを示す。
VSL_ERROR_NULL_PTR	<i>stream</i> が NULL ポインタである。
VSL_ERROR_BAD_STREAM	<i>stream</i> が有効なランダム・ストリームでない。
VSL_ERROR_BAD_UPDATE	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数として無効な値 (< 0 または > nmax) を返した。
VSL_ERROR_NO_NUMBERS	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数としてゼロを返した。

## Beta

ベータ分布に従う乱数を生成する。

## 構文

## Fortran:

```
status = vsrngbeta( method, stream, n, r, p, q, a, beta )
status = vdrngbeta( method, stream, n, r, p, q, a, beta )
```

## C:

```
status = vsRngBeta( method, stream, n, r, p, q, a, beta );
status = vdRngBeta( method, stream, n, r, p, q, a, beta );
```

## 説明

この関数は、形状母数  $p$  と  $q$ 、位置母数  $a$ 、尺度母数  $\beta$  を持つベータ分布の乱数を生成する。 $p, q, a, \beta \in \mathbb{R}$ 。 $p > 0, q > 0, \beta > 0$ 。

確率密度関数は次のように示される。

$$f_{p,q,a,\beta}(x) = \begin{cases} \frac{1}{B(p,q)\beta^{p+q-1}}(x-a)^{p-1}(\beta+a-x)^{q-1}, & a \leq x < a+\beta \\ 0, & x < a, x \geq a+\beta \end{cases}, -\infty < x < \infty,$$

$B(p,q)$  は完全ベータ関数である。

累積分布関数は次のように示される。

$$F_{p,q,a,\beta}(x) = \begin{cases} 0, & x < a \\ \int_a^x \frac{1}{B(p,q)\beta^{p+q-1}}(y-a)^{p-1}(\beta+a-y)^{q-1} dy, & a \leq x < a+\beta \\ 1, & x \geq a+\beta \end{cases}, -\infty < x < \infty.$$

## 入力パラメータ

名前	データ型		説明
	FORTRAN	C	
<i>method</i>	INTEGER, INTENT(IN)	int	生成方法。特定の値は次のとおりである。 VSL_METHOD_SBETA_CJA VSL_METHOD_DBETA_CJA <a href="#">表 10-1</a> の CJA の説明を参照。
<i>stream</i>	TYPE (VSL_STREAM_STATE), INTENT(IN)	VSLStreamStatePtr	<i>Fortran</i> の場合：ストリーム・ステート構造体のディスクリプタ C の場合：ストリーム・ステート構造体へのポインタ
<i>n</i>	INTEGER, INTENT(IN)	int	生成させる乱数の個数。
<i>p</i>	REAL, INTENT(IN) (vsrngbeta の場合) DOUBLE PRECISION, INTENT(IN) (vdrngbeta の場合)	float (vsrngbeta の場合) double (vdrngbeta の場合)	形状母数 $p$ 。

名前	データ型		説明
	FORTRAN	C	
$q$	REAL, INTENT(IN) (vsrngbeta の場合) DOUBLE PRECISION, INTENT(IN) (vdrngbeta の場合)	float (vsRngBeta の場合) double (vdRngBeta の場合)	形状母数 $q$ 。
$a$	REAL, INTENT(IN) (vsrngbeta の場合) DOUBLE PRECISION, INTENT(IN) (vdrngbeta の場合)	float (vsRngBeta の場合) double (vdRngBeta の場合)	位置母数 $a$ 。
$\beta$	REAL, INTENT(IN) (vsrngbeta の場合) DOUBLE PRECISION, INTENT(IN) (vdrngbeta の場合)	float (vsRngBeta の場合) double (vdRngBeta の場合)	尺度母数 $\beta$ 。

### 出力パラメータ

名前	データ型		説明
	FORTRAN	C	
$r$	REAL, INTENT(OUT) (vsrngbeta の場合) DOUBLE PRECISION, INTENT(OUT) (vdrngbeta の場合)	float* (vsRngBeta の場合) double* (vdRngBeta の場合)	ベータ分布に従った $n$ 個の乱数のベクトル。

### 戻り値

VSL_ERROR_OK, VSL_STATUS_OK	正常に終了したことを示す。
VSL_ERROR_NULL_PTR	<i>stream</i> が NULL ポインタである。
VSL_ERROR_BAD_STREAM	<i>stream</i> が有効なランダム・ストリームでない。
VSL_ERROR_BAD_UPDATE	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数として無効な値 ( $< 0$ または $> nmax$ ) を返した。
VSL_ERROR_NO_NUMBERS	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数としてゼロを返した。

## 離散分布

離散分布の乱数を生成するルーチンについて説明する。

---

## Uniform

区間  $[a, b)$  を持つ一様離散分布に従った乱数を生成する。

---

### 構文

**Fortran:**

```
status = virnguniform( method, stream, n, r, a, b )
```

**C:**

```
status = viRngUniform( method, stream, n, r, a, b );
```

### 説明

この関数は区間  $[a, b)$  を持つ一様離散分布に従った乱数を生成する。 $a, b$  は、それぞれ区間の始まりと終わりであり、 $a, b \in \mathbb{Z}; a < b$ 。

確率分布は次のように示される。

$$P(X = k) = \frac{1}{b - a}, \quad k \in \{a, a + 1, \dots, b - 1\}$$

累積分布関数は次のように示される。

$$F_{a,b}(x) = \begin{cases} 0, & x < a \\ \frac{\lfloor x - a + 1 \rfloor}{b - a}, & a \leq x < b, \quad x \in \mathbb{R} \\ 1, & x \geq b \end{cases}$$

## 入力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>method</i>	INTEGER, INTENT(IN)	int	生成方法。このパラメータはダミーで一様分布の場合は 0 を設定する。特定の値は次のとおりである。 VSL_METHOD_IUNIFORM_STD 標準生成法。現時点で、この分布生成器に対する <i>method</i> は 1 つのみ。
<i>stream</i>	TYPE (VSL_STREAM_STATE), INTENT(IN)	VSLStreamStatePtr	<i>Fortran</i> : ストリーム・ステート構造体のディスクリプタ。 <i>C</i> : ストリーム・ステート構造体へのポインタ。
<i>n</i>	INTEGER, INTENT(IN)	int	生成させる乱数の個数。
<i>a</i>	INTEGER, INTENT(IN)	int	区間の開始 <i>a</i> 。
<i>b</i>	INTEGER, INTENT(IN)	int	区間の終了 <i>b</i> 。

## 出力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>r</i>	INTEGER, INTENT(OUT)	int*	区間 $[a,b)$ を持つ一様分布に従った <i>n</i> 個の乱数のベクトル。

## 戻り値

VSL_ERROR_OK, VSL_STATUS_OK	正常に終了したことを示す。
VSL_ERROR_NULL_PTR	<i>stream</i> が NULL ポインタである。
VSL_ERROR_BAD_STREAM	<i>stream</i> が有効なランダム・ストリームでない。
VSL_ERROR_BAD_UPDATE	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数として無効な値 ( $< 0$ または $> nmax$ ) を返した。
VSL_ERROR_NO_NUMBERS	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数としてゼロを返した。



## UniformBits

一様ビット分布に従う整数乱数を生成する。

### 構文

**Fortran:**

```
status = virnguniformbits( method, stream, n, r )
```

**C:**

```
status = viRngUniformBits( method, stream, n, r );
```

### 説明

この関数は一様ビット分布に従う整数の乱数を生成する。一様分布の乱数生成器は、整数値に対する剰余算の漸化式として表せる。各整数は、それぞれ複数のビットを持つベクトルとして扱えることは明らかである。真の乱数を発する生成器ではこれらビットはランダムであるが、擬似乱数生成器ではランダムさは失われる。例えば線形合同生成器には、上位ビットに比べて下位ビットのランダムさが少ないというよく知られた欠点がある（例えば [\[Knuth81\]](#) を参照）。この理由から、この関数を使用する場合は注意が必要である。一般に 32 ビット *LCG* では、整数値の上位 24 ビットのみがランダムであるとみなされる。詳細は [VSL Notes](#) を参照のこと。

### 入力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>method</i>	INTEGER, INTENT( IN )	int	生成方法。ダミーで 0 を設定する。特定の値は次のとおりである。 VSL_METHOD_IUNIFORMBITS_STD 標準生成法。現時点で、この分布生成器に対する <i>method</i> は 1 つのみ。
<i>stream</i>	TYPE( VSL_STREAM_STATE ), INTENT( IN )	VSLStreamStatePtr	<i>Fortran</i> の場合：ストリーム・ステート構造体のディスクリプタ。 <i>C</i> の場合：ストリーム・ステート構造体へのポインタ。
<i>n</i>	INTEGER, INTENT( IN )	int	生成させる乱数の個数。

## 出力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>r</i>	INTEGER, INTENT(OUT)	unsigned int*	<p><i>Fortran</i> の場合: <i>n</i> 個の乱数のベクトル。<i>stream</i> が 64 ビットまたは 128 ビットの生成器で生成された場合、各整数値は 2 個または 4 個の <i>r</i> の成分でそれぞれ表される。各整数が占有するバイト数は VSL_BRNG_PROPERTIES 構造体の <i>wordsize</i> フィールドで指定される。値を格納するために実際に使用される合計ビット数は同 VSL_BRNG_PROPERTIES 構造体の <i>nbits</i> フィールドで指定される。VSLBrngProperties の詳細は、「<a href="#">アドバンスト・サービス・ルーチン</a>」を参照のこと。</p> <p><i>C</i> の場合: <i>n</i> 個の乱数のベクトル。<i>stream</i> が 64 ビットまたは 128 ビットの生成器で生成された場合、各整数値は 2 個または 4 個の <i>r</i> の成分でそれぞれ表される。各整数が占有するバイト数は VSLBrngProperties 構造体の <i>WordSize</i> フィールドで指定される。値を格納するために実際に使用される合計ビット数は同 VSLBrngProperties 構造体の <i>NBits</i> フィールドで指定される。VSLBrngProperties の詳細は、「<a href="#">アドバンスト・サービス・ルーチン</a>」を参照のこと。</p>

## 戻り値

VSL\_ERROR\_OK, VSL\_STATUS\_OK

正常に終了したことを示す。

VSL\_ERROR\_NULL\_PTR

*stream* が NULL ポインタである。

VSL\_ERROR\_BAD\_STREAM

*stream* が有効なランダム・ストリームでない。

VSL\_ERROR\_BAD\_UPDATE

抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数として無効な値 (< 0 または > *nmax*) を返した。

VSL\_ERROR\_NO\_NUMBERS

抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数としてゼロを返した。

---

## Bernoulli

ベルヌーイ分布に従う乱数を生成する。

---

### 構文

#### Fortran:

```
status = virngbernoulli( method, stream, n, r, p )
```

#### C:

```
status = viRngBernoulli( method, stream, n, r, p );
```

### 説明

この関数は、単一の試行成功確率を  $p$  とするベルヌーイ分布に従った乱数を生成する。

$$p \in R; 0 \leq p \leq 1$$

変数はベルヌーイ分布と呼ばれ、試行の結果 1 となる成功確率は  $p$ 、0 となる確率は  $1-p$  である。

確率分布は次のように示される。

$$P(X = 1) = p,$$

$$P(X = 0) = 1 - p$$

累積分布関数は次のように示される。

$$F_p(x) = \begin{cases} 0, & x < 0 \\ 1 - p, & 0 \leq x < 1, x \in R \\ 1, & x \geq 1 \end{cases}$$

## 入力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>method</i>	INTEGER, INTENT(IN)	int	生成方法。特定の値は次のとおりである。 VSL_METHOD_IBERNOULLI_ICDF 逆累積分布関数法。
<i>stream</i>	TYPE (VSL_STREAM_STATE), INTENT(IN)	VSLStreamStatePtr	<i>Fortran</i> の場合 : ストリーム・ステート構造体のディスクリプタ。 <i>C</i> の場合 : ストリーム・ステート構造体へのポインタ。
<i>n</i>	INTEGER, INTENT(IN)	int	生成させる乱数の個数。
<i>p</i>	DOUBLE PRECISION, INTENT(IN)	double	試行の成功確率 $p$ 。

## 出力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>r</i>	INTEGER, INTENT(OUT)	int*	ベルヌーイ分布に従った $n$ 個の乱数のベクトル。

## 戻り値

VSL_ERROR_OK, VSL_STATUS_OK	正常に終了したことを示す。
VSL_ERROR_NULL_PTR	<i>stream</i> が NULL ポインタである。
VSL_ERROR_BAD_STREAM	<i>stream</i> が有効なランダム・ストリームでない。
VSL_ERROR_BAD_UPDATE	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数として無効な値 ( $< 0$ または $> nmax$ ) を返した。
VSL_ERROR_NO_NUMBERS	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数としてゼロを返した。

# Geometric

幾何分布に従う乱数を生成する。

## 構文

**Fortran:**

```
status = virnggeometric( method, stream, n, r, p )
```

**C:**

```
status = viRngGeometric( method, stream, n, r, p );
```

## 説明

この関数は、試行の成功確率を  $p$  とする幾何分布に従った乱数を生成する。ここで、 $p \in R; 0 < p < 1$ 。

幾何分布の変数は最初の成功が得られるまでの独立したベルヌーイ試行の回数である。ベルヌーイ試行の成功確率は  $p$  である。

確率分布は次のように示される。

$$P(X = k) = p \cdot (1 - p)^k, k \in \{0, 1, 2, \dots\}$$

累積分布関数は次のように示される。

$$\begin{aligned} &0, && x < 0 \\ &1 - (1 - p)^{\lfloor x + 1 \rfloor}, && x \geq 0 \end{aligned}, x \in R$$

## 入力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>method</i>	INTEGER, INTENT( IN)	int	生成方法。特定の値は次のとおりである。 VSL_METHOD_IGEOMETRIC_ICDF 逆累積分布関数法。

名前	データ型		説明
	FORTTRAN	C	
<i>stream</i>	TYPE (VSL_STREAM_STATE), INTENT (IN)	VSLStreamStatePtr	<i>Fortran</i> の場合 : ストリーム・ステート構造体のディスクリプタ。 <i>C</i> の場合 : ストリーム・ステート構造体へのポインタ。
<i>n</i>	INTEGER, INTENT (IN)	int	生成させる乱数の個数。
<i>p</i>	DOUBLE PRECISION, INTENT (IN)	double	試行の成功確率 <i>p</i> 。

## 出力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>r</i>	INTEGER, INTENT (OUT)	int*	幾何分布に従った <i>n</i> 個の乱数のベクトル。

## 戻り値

VSL_ERROR_OK, VSL_STATUS_OK	正常に終了したことを示す。
VSL_ERROR_NULL_PTR	<i>stream</i> が NULL ポインタである。
VSL_ERROR_BAD_STREAM	<i>stream</i> が有効なランダム・ストリームでない。
VSL_ERROR_BAD_UPDATE	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数として無効な値 (< 0 または > nmax) を返した。
VSL_ERROR_NO_NUMBERS	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数としてゼロを返した。

## Binomial

二項分布に従う乱数を生成する。

### 構文

**Fortran:**

```
status = virngbinomial( method, stream, n, r, ntrial, p )
```

**C:**

```
status = viRngBinomial( method, stream, n, r, ntrial, p );
```

### 説明

この関数は、独立したベルヌーイ試行回数  $m$  と単一の試行成功確率  $p$  を持つ二項分布に従った乱数を生成する。ここで、 $p \in R; 0 \leq p \leq 1, m \in N$ 。

二項分布の変数は、単一の試行成功確率  $p$  を持つベルヌーイ試行を、独立に  $m$  回行ったときの成功回数として表される。

確率分布は次のように示される。

$$P(X = k) = C_m^k p^k (1-p)^{m-k}, k \in \{0, 1, \dots, m\}$$

累積分布関数は次のように示される。

$$F_{m,p}(x) = \begin{cases} 0, & x < 0 \\ \sum_{k=0}^{\lfloor x \rfloor} C_m^k p^k (1-p)^{m-k}, & 0 \leq x < m, x \in R \\ 1, & x \geq m \end{cases}$$

## 入力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>method</i>	INTEGER, INTENT(IN)	int	生成方法。特定の値は次のとおりである。 VSL_METHOD_IBINOMIAL_BTPE <a href="#">表 10-1</a> の BTPE の説明を参照。
<i>stream</i>	TYPE (VSL_STREAM_STATE), INTENT(IN)	VSLStreamStatePtr	<i>Fortran</i> の場合：ストリーム・ステート構造体のディスクリプタ。 <i>C</i> の場合：ストリーム・ステート構造体へのポインタ。
<i>n</i>	INTEGER, INTENT(IN)	int	生成させる乱数の個数。
<i>ntrials</i>	INTEGER, INTENT(IN)	int	独立した試行回数 <i>m</i>
<i>p</i>	DOUBLE PRECISION, INTENT(IN)	double	単一試行の成功確率 <i>p</i>

## 出力パラメータ

s

名前	データ型		説明
	FORTTRAN	C	
<i>r</i>	INTEGER, INTENT(OUT)	int*	二項分布に従った <i>n</i> 個の乱数のベクトル。

## 戻り値

VSL_ERROR_OK, VSL_STATUS_OK	正常に終了したことを示す。
VSL_ERROR_NULL_PTR	<i>stream</i> が NULL ポインタである。
VSL_ERROR_BAD_STREAM	<i>stream</i> が有効なランダム・ストリームでない。
VSL_ERROR_BAD_UPDATE	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数として無効な値 (< 0 または > nmax) を返した。
VSL_ERROR_NO_NUMBERS	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数としてゼロを返した。



## Hypergeometric

超幾何分布に従う乱数を生成する。

### 構文

**Fortran:**

```
status = virnghypergeometric( method, stream, n, r, l, s, m )
```

**C:**

```
status = viRngHypergeometric( method, stream, n, r, l, s, m );
```

### 説明

この関数は、母集団の大きさ  $l$ 、抽出の大きさ  $s$ 、母集団内の「あたり」の成分数  $m$  を持つ超幾何分布に従った乱数を生成する。

ここで、 $l, m, s \in \mathbb{N} \cup \{0\}$ ;  $l \geq \max(s, m)$ 。

母集団  $l$  の成分は、 $m$  個の「あたり」成分と  $l-m$  個の「はずれ」成分で構成されているとする。この母集団から復元なしで  $s$  個の成分を試行抽出すると、 $s$  個の成分の中に  $k$  個の「あたり」を含んでいる確率を示す超幾何分布が得られる。

確率分布は次のように示される。

$$P(X = k) = \frac{C_m^k C_{l-m}^{s-k}}{C_l^s}, \quad k \in \{\max(0, s+m-l), \dots, \min(s, m)\}$$

累積分布関数は次のように示される。

$$F_{l,s,m}(x) = \begin{cases} 0, & x < \max(0, s+m-l) \\ \sum_{k=\max(0, s+m-l)}^{\lfloor x \rfloor} \frac{C_m^k C_{l-m}^{s-k}}{C_l^s}, & \max(0, s+m-l) \leq x \leq \min(s, m) \\ 1, & x > \min(s, m) \end{cases}$$

## 入力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>method</i>	INTEGER, INTENT(IN)	int	生成方法。特定の値は次のとおりである。 VSL_METHOD_IHYPERGEOMETRIC_H2PE <a href="#">表 10-1</a> の H2PE の説明を参照。
<i>stream</i>	TYPE (VSL_STREAM_STATE), INTENT(IN)	VSLStreamStatePtr	<i>Fortran</i> の場合: ストリーム・ステート構造体のディスクリプタ。 <i>C</i> の場合: ストリーム・ステート構造体へのポインタ。
<i>n</i>	INTEGER, INTENT(IN)	int	生成させる乱数の個数。
<i>l</i>	INTEGER, INTENT(IN)	int	母集団の大きさ <i>l</i> 。
<i>s</i>	INTEGER, INTENT(IN)	int	復元なしに抽出する個数 <i>s</i> 。
<i>m</i>	INTEGER, INTENT(IN)	int	「あたり」成分の個数 <i>m</i> 。

## 出力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>r</i>	INTEGER, INTENT(OUT)	int*	超幾何分布に従った <i>n</i> 個の乱数のベクトル。

## 戻り値

VSL_ERROR_OK, VSL_STATUS_OK	正常に終了したことを示す。
VSL_ERROR_NULL_PTR	<i>stream</i> が NULL ポインタである。
VSL_ERROR_BAD_STREAM	<i>stream</i> が有効なランダム・ストリームでない。
VSL_ERROR_BAD_UPDATE	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数として無効な値 (< 0 または > nmax) を返した。
VSL_ERROR_NO_NUMBERS	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数としてゼロを返した。

---

## Poisson

ポアソン分布に従う乱数を生成する。

---

### 構文

#### Fortran:

```
status = virngpoisson( method, stream, n, r, lambda )
```

#### C:

```
status = viRngPoisson( method, stream, n, r, lambda );
```

### 説明

この関数は、分布パラメータを  $\lambda$  とするポアソン分布に従った乱数を生成する。ここで、 $\lambda \in R; \lambda > 0$ 。

確率分布は次のように示される。

$$P(X = k) = \frac{\lambda^k e^{-\lambda}}{k!}, k \in \{0, 1, 2, \dots\}$$

累積分布関数は次のように示される。

$$F_{\lambda}(x) = \begin{cases} \sum_{k=0}^{\lfloor x \rfloor} \frac{\lambda^k e^{-\lambda}}{k!}, & x \geq 0 \\ 0, & x < 0 \end{cases}, x \in R$$

## 入力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>method</i>	INTEGER, INTENT(IN)	int	生成方法。特定の値は次のとおりである。 VSL_METHOD_IPOISSON_PTPE VSL_METHOD_IPOISSON_POISNORM <a href="#">表 10-1</a> の PTPE と POISNORM の説明を参照。
<i>stream</i>	TYPE (VSL_STREAM_STATE), INTENT(IN)	VSLStreamStatePtr	<i>Fortran</i> の場合: ストリーム・ステート構造体のディスクリプタ。 <i>C</i> の場合: ストリーム・ステート構造体へのポインタ。
<i>n</i>	INTEGER, INTENT(IN)	int	生成させる乱数の個数。
<i>lambda</i>	DOUBLE PRECISION, INTENT(IN)	double	分布パラメータ $\lambda$

## 出力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>r</i>	INTEGER, INTENT(OUT)	int*	ポアソン分布に従った <i>n</i> 個の乱数のベクトル。

## 戻り値

VSL\_ERROR\_OK, VSL\_STATUS\_OK

正常に終了したことを示す。

VSL\_ERROR\_NULL\_PTR

*stream* が NULL ポインタである。

VSL\_ERROR\_BAD\_STREAM

*stream* が有効なランダム・ストリームでない。

VSL\_ERROR\_BAD\_UPDATE

抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数として無効な値 (&lt; 0 または &gt; nmax) を返した。

VSL\_ERROR\_NO\_NUMBERS

抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数としてゼロを返した。

## PoissonV

多様な平均値を持つポアソン分布に従った乱数を生成する。

### 構文

**Fortran:**

```
status = virngpoissonv( method, stream, n, r, lambda )
```

**C:**

```
status = viRngPoissonV( method, stream, n, r, lambda );
```

### 説明

この関数は、分布パラメータを  $\lambda_i$  とするポアソン分布に従った乱数  $x_i (i = 1, \dots, n)$  を  $n$  個生成する。ここで  $\lambda_i \in R; \lambda_i > 0$ 。

確率分布は次のように示される。

$$P(X_i = k) = \frac{\lambda_i^k \exp(-\lambda_i)}{k!}, k \in \{0, 1, 2, \dots\}$$

累積分布関数は次のように示される。

$$F_{\lambda_i}(x) = \begin{cases} \sum_{k=0}^{\lfloor x \rfloor} \frac{\lambda_i^k e^{-\lambda_i}}{k!}, & x \geq 0 \\ 0, & x < 0 \end{cases}, x \in R$$

## 入力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>method</i>	INTEGER, INTENT(IN)	int	生成方法。特定の値は次のとおりである。 VSL_METHOD_IPOISSONV_POISNORM <a href="#">表 10-1</a> の POISNORM の説明を参照。
<i>stream</i>	TYPE (VSL_STREAM_STATE), INTENT(IN)	VSLStreamStatePtr	Fortran の場合：ストリーム・ステート構造体のディスクリプタ。 C の場合：ストリーム・ステート構造体へのポインタ。
<i>n</i>	INTEGER, INTENT(IN)	int	生成させる乱数の個数。
<i>lambda</i>	DOUBLE PRECISION, INTENT(IN)	double*	$n$ 個の分布パラメータの配列 $\lambda_i$

## 出力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>r</i>	INTEGER, INTENT(OUT)	int*	ポアソン分布に従った $n$ 個の乱数のベクトル。

## 戻り値

VSL\_ERROR\_OK, VSL\_STATUS\_OK

正常に終了したことを示す。

VSL\_ERROR\_NULL\_PTR

*stream* が NULL ポインタである。

VSL\_ERROR\_BAD\_STREAM

*stream* が有効なランダム・ストリームでない。

VSL\_ERROR\_BAD\_UPDATE

抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数として無効な値 (&lt; 0 または &gt; nmax) を返した。

VSL\_ERROR\_NO\_NUMBERS

抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数としてゼロを返した。

## NegBinomial

負の二項分布に従う乱数を生成する。

### 構文

**Fortran:**

```
status = virngnegbinomial( method, stream, n, r, a, p )
```

**C:**

```
status = viRngNegBinomial( method, stream, n, r, a, p );
```

### 説明

この関数は、分布パラメータが  $a$  と  $p$  を持つ負の二項分布に従った乱数を生成する。ここで、 $p, a \in R; 0 < p < 1; a > 0$ 。

第 1 の分布パラメータ  $a \in N$  がなら、この分布はパスカル分布と等しくなる。この分布は  $a \in N$  では、成功確率  $p$  のベルヌーイ試行のときに、 $a$  番目の成功を得るまでの見込まれる試行回数として解釈できる。

確率分布は次のように示される。

$$P(X = k) = C_{a+k-1}^k p^a (1-p)^k, k \in \{0, 1, 2, \dots\}$$

累積分布関数は次のように示される。

$$F_{a,p}(x) = \begin{cases} \sum_{k=0}^{\lfloor x \rfloor} C_{a+k-1}^k p^a (1-p)^k, & x \geq 0 \\ 0, & x < 0 \end{cases}, x \in R.$$

## 入力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>method</i>	INTEGER, INTENT(IN)	int	生成方法。特定の値は次のとおりである。 VSL_METHOD_INEGBINOMIAL_NBAR <a href="#">表 10-1</a> の NBAR の説明を参照。
<i>stream</i>	TYPE (VSL_STREAM_STATE), INTENT(IN)	VSLStreamStatePtr	Fortran の場合：ストリーム・ステート構造体のディスクリプタ。 C の場合：ストリーム・ステート構造体へのポインタ。
<i>n</i>	INTEGER, INTENT(IN)	int	生成させる乱数の個数。
<i>a</i>	DOUBLE PRECISION, INTENT(IN)	double	第 1 の分布パラメータ <i>a</i>
<i>p</i>	DOUBLE PRECISION, INTENT(IN)	double	第 2 の分布パラメータ <i>p</i>

## 出力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>r</i>	INTEGER, INTENT(OUT)	int *	ポアソン分布に従った <i>n</i> 個の乱数のベクトル。

## 戻り値

VSL_ERROR_OK, VSL_STATUS_OK	正常に終了したことを示す。
VSL_ERROR_NULL_PTR	<i>stream</i> が NULL ポインタである。
VSL_ERROR_BAD_STREAM	<i>stream</i> が有効なランダム・ストリームでない。
VSL_ERROR_BAD_UPDATE	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数として無効な値 (< 0 または > nmax) を返した。
VSL_ERROR_NO_NUMBERS	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数としてゼロを返した。



## アドバンスト・サービス・ルーチン

このセクションでは、ユーザ定義基本生成器の登録を行うルーチン ([RegisterBrng](#)) と、登録済み基本生成器のプロパティを取得するルーチン ([GetBrngProperties](#)) について説明する。ユーザ定義の BRNG を含む基本生成器の必要性については、[VSL Notes](#) (VSL Structure の章の「Basic Generators」セクション) を参照。

## データ型

ここで記載のルーチン群は、基本生成器のプロパティを定義する構造体を参照する。この構造体は、Fortran では次のようになっている。

```
TYPE VSL_BRNG_PROPERTIES
    INTEGER streamstatesize
    INTEGER nseeds
    INTEGER includeszero
    INTEGER wordsize
    INTEGER nbits
    INTEGER initstream
    INTEGER sbrng
    INTEGER dbrng
    INTEGER ibrng
END TYPE VSL_BRNG_PROPERTIES
```

C では次のようになっている。

```
typedef struct _VSLBRngProperties {
    int          StreamStateSize;
    int          NSeeds;
    int          IncludesZero;
    int          WordSize;
    int          NBits;
    InitStreamPtr InitStream;
    sBRngPtr     sBRng;
    dBRngPtr     dBRng;
    iBRngPtr     iBRng;
} VSLBRngProperties;
```

上記の構造体に関与しているフィールドの説明を次の表に示す。

**表 10-12      フィールドの説明**

フィールド	説明
FORTRAN: streamstatesize C: StreamStateSize	バイトを単位とする、指定された基本生成器に対するストリーム・ステート構造体のサイズ。
FORTRAN: nseeds C: NSeeds	指定された基本生成器に対するストリーム・ステート構造体の初期化に必要な、32 ビット初期条件 (種) の個数。
FORTRAN: includeszero C: IncludesZero	生成器が乱数 0 を生成可能であることを示すフラグ値。 <sup>1</sup>
FORTRAN: wordsize C: WordSize	バイトを単位とする、整数値計算で使用するマシン・ワード・サイズ。取り得る値は、32 ビット、64 ビット、128 ビット生成器で、それぞれ 4、8、16。
FORTRAN: nbits C: NBits	整数算で乱数を表すために必要となるビット数。例えば、48 ビット乱数は 64 ビット (8 バイト) メモリ・ロケーションに格納される点に注意する。この場合を例に取ると、wordsize/WordSize (その乱数の格納に使用されるバイト数) は 8 となるが、nbits/NBits はその値によって占有される実際のビット数であるから 48 となる。
FORTRAN: initstream C: InitStream	指定された基本生成器の初期化ルーチンに対するポインタ。
FORTRAN: sbrng C: sBRng	区間 (a, B) を持つ一様分布を単精度実数で処理する基本生成器に対するポインタ (FORTRAN では REAL、C では float)。
FORTRAN: dbrng C: dBRng	区間 (a, b) を持つ一様分布を倍精度実数で処理する基本生成器に対するポインタ (FORTRAN では DOUBLE PRECISION、C では double)。
FORTRAN: ibrng C: iBRng	一様ビット分布を整数で処理する基本生成器に対するポインタ (FORTRAN では INTEGER、C では unsigned int)。 <sup>2</sup>

1. 例えば GFSR のような一部の生成器はゼロの乱数を生成する可能性を持っている。一方、乗算合同生成器などはゼロを生成することはありません。ゼロが生成される可能性は低いため、一般にこのような情報にはそれほど意味はない。しかし、一部の非一様分布生成器では、基本生成器がゼロを生成する可能性があるのは、無限に近い大きな値を生成する可能性もあることを意味する (オーバーフロー)。たとえソフトウェアがオーバーフローを  $+\infty$  や  $-\infty$  として正しく取り扱ったとしても、このような性質には注意が必要であり、結果の確認が求められる。無限に近い大きな値が計算に影響を与える場合、生成されたベクトルからそのような値を取り除くか、あるいはゼロを生成しない安全な生成器を使用する必要がある。
2. 乱数の単一ビットとビットグループとの演算を許可している特有の生成器である。

## RegisterBrng

ユーザ定義の基本生成器を登録する。

### 構文

**Fortran:**

```
brng = vslregisterbrng( properties )
```

**C:**

```
brng = vslRegisterBrng( &properties );
```

### 説明

登録手順の例は各システムの VSL examples ディレクトリにある。

### 入力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>properties</i>	TYPE(VSL_BRNG_PROPERTIES), INTENT(IN)	VSLBrngProperties*	登録する基本生成器のプロパティを含む構造体へのポインタ。

### 出力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>brng</i>	INTEGER, INTENT(OUT)	int	登録された基本生成器の番号 (インデックス値) で、識別に使用する。負の値が返ってきた場合は登録時にエラーが発生したことを示す。

## 戻り値

VSL_ERROR_OK, VSL_STATUS_OK	正常に終了したことを示す。
VSL_ERROR_BRNG_TABLE_FULL	登録されている BRNG の表にフリー成分が不足しているため、登録を完了することができない。
VSL_ERROR_BAD_STREAM_STATE_SIZE	StreamStateSize フィールドの値が不正である。
VSL_ERROR_BAD_WORD_SIZE	WordSize フィールドの値が不正である。
VSL_ERROR_BAD_NSEEDS	NSeeds フィールドの値が不正である。
VSL_ERROR_BAD_NBITS	NBits フィールドの値が不正である。
VSL_ERROR_NULL_PTR	iBrng、dBrng、sBrng、または InitStream の少なくとも 1 つが NULL ポインタである。

## GetBrngProperties

指定した基本生成器のプロパティを構造体で返す。

### 構文

#### Fortran:

```
status = vslgetbrngproperties( brng, properties )
```

#### C:

```
status = vslGetBrngProperties( brng, &properties );
```

### 入力パラメータ

名前	データ型		説明
	FORTRAN	C	
<i>brng</i>	INTEGER, INTENT(IN)	int	登録された基本生成器の番号 (インデックス値) で、識別に使用する。 <a href="#">表 10-2</a> の特定の値を参照。負の値が返ってきた場合は登録時にエラーが発生したことを示す。

出力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>properties</i>	TYPE(VSL_BRNG_PROPERTIES), INTENT(OUT)	VSLBrngProperties*	番号 <i>brng</i> を持つ生成器のプロパティを含む構造体へのポインタ

戻り値

VSL_ERROR_OK, VSL_STATUS_OK	正常に終了したことを示す。
VSL_ERROR_INVALID_BRNG_INDEX	BRNG のインデックスが無効である。

ユーザ定義生成器のフォーマット

[RegisterBrng](#) 関数を用いてユーザ定義基本生成器を登録するには、整数値で実装されている生成器のポインタを *iBrng* に、単精度値および倍精度値で実装されている生成器のポインタをそれぞれ *sBrng* と *dBrng* に、ストリーム初期化ルーチンのポインタを *InitStream* に、それぞれ渡さなければならない。以下のセクションに、入力引数と出力引数の使い方と合わせ各関数定義の推奨例を記載している。ユーザ定義生成器の登録手順例は各システムの **VSL examples** ディレクトリにある。

それぞれのポインタは次のように定義される。

```
typedef    int (*InitStreamPtr)( int method, void * stream, int n,  
                                const unsigned int params[] );  
  
typedef    int (*sBRngPtr)( void * stream, int n, float r[],  
                            float a, float b );  
  
typedef    int (*dBRngPtr)( void * stream, int n, double r[],  
                            double a, double b );  
  
typedef    int (*iBRngPtr)( void * stream, int n,  
                            unsigned int r[] );
```

## InitStream

C:

```
int MyBrngInitStream( int method, VSLStreamStatePtr stream,
    int n, const unsigned int params[] );
{
    /* Initialize the stream */
    ...
} /* MyBrngInitStream */
```

## 説明

ユーザ定義生成器の初期化ルーチンは、指定された初期化 *method*、初期条件 *params*、引数 *n* に従って *stream* を初期化しなければならない。*method* の値によって初期化方法が指定される。

- *method* が 0 の場合、すべての基本生成器がサポートしなければならない標準的な生成方法を用いて初期化を行う。ここで関数は、*stream* 構造体は過去に初期化されていないと仮定する。*n* は、初期化条件として *params* を介して渡される 32 ビット値の個数を表す。関数に渡される初期条件の個数が生成器の初期化に足りない場合でもエラーではないことに注意が必要である。このような状況が起こった場合、基本生成器は、不足している初期条件に代わってデフォルト値を用いて初期化を行わなければならない。
- *method* が 1 の場合はリープフロッグ法による生成となり、*n* は処理ノード（独立ストリーム）数を指定する。*stream* は標準生成法で初期化されていると仮定する。*params* は処理ノードを指定する単一成分のみで構成される。生成器がリープフロッグ法をサポートしない場合、関数はエラーコード `VSL_ERROR_LEAPFROG_UNSUPPORTED` を返さなければならない。
- *method* が 2 の場合はブロック分割法による生成となる。上記と同様に、*stream* は標準生成法で初期化されていると仮定する。*params* は使用されない。*n* はスキップする成分数を指定する。生成器がブロック分割法をサポートしていない場合、関数はエラーコード `VSL_ERROR_SKIPAHEAD_UNSUPPORTED` を返さなければならない。

リープフロッグ法とブロック分割法の詳細は、[LeapfrogStream](#) および [SkipAheadStream](#) をそれぞれ参照のこと。

ストリーム・ステート構造体はそれぞれの生成器ごとに固有である。ただし、すべての生成器で共通なフィールドはすべての構造体を持っている。

**C:**

```
typedef struct
{
    unsigned int Reserved1[2];
    unsigned int Reserved2[2];
    [fields specific for the given generator]
} MyStreamState;
```

フィールド *Reserved1* と *Reserved2* は非公開目的のために予約されており、ユーザは変更してはならない。構造体に専用のフィールドを追加する場合は次の規則に従うこと。

- フィールドは生成器の最新ステートを完全に記述しなくてはならない。例えば、線形合同生成器のステートは単一の初期条件しか持たない。
- 生成器がリープフロッグ法とブロック分割法の両方に対応するのであれば、独立したストリームを識別するために追加フィールドを導入すべきである。例えば  $LCG(a, c, m)$  では、初期条件とは別に、乗数  $a^k$  と、加数  $(a^k - 1)c / (a - 1)$  の 2 つの追加フィールドを規定する必要がある。

詳細は [\[Knuth81\]](#) と [\[Gentle98\]](#) を参照のこと。登録手順の例は各システムの VSL examples ディレクトリにある。

## iBRng

**C:**

```
void iMyBrng( VSLStreamStatePtr stream, int n,
             unsigned int r[] )
{
    int i;    /* Loop variable */
    /* Generating integer random numbers */
    /* Pay attention to word size needed to
       store only random number */
    for( i = 0; i < n; i++ )
    {
        r[i] = ...;
    }
}
```

```
/* Update stream state */
...
return errcode;
} /* iMyBrng */
```



---

**注** :64 ビットと 128 ビットの生成器を使用する場合、乱数ベクトル  $r$  に値を正しく格納するために桁容量を考慮すること。例えば、1 個の 64 ビット値は  $r$  の 2 個の成分を必要とし、1 番目の成分に下位 32 ビット、2 番目の成分に上位 32 ビットが格納される。同様に 128 ビット値では 4 個の  $r$  成分が使われる。

---

## sBRng

**C:**

```
void sMyBrng( VSLStreamStatePtr stream, int n, float r[],
             float a, float b )
{
    int      i;      /* Loop variable */
    /* Generating float (a,b) random numbers */
    for ( i = 0; i < n; i++ )
    {
        r[i] = ...;
    }
    /* Update stream state */
    ...
    return errcode;
} /* sMyBrng */
```



## dBRng

**C:**

```
void dMyBrng( VSLStreamStatePtr stream, int n, double r[],
             double a, double b )
{
    int i;      /* Loop variable */
    /* Generating double (a,b) random numbers */
    for ( i = 0; i < n; i++ )
    {
        r[i] = ...;
    }
    /* Update stream state */
    ...
    return errcode;
} /* dMyBrng */
```

## 畳み込みと相関

### 概要

VSL では、単精度および倍精度データの線形畳み込み / 相関変換を実行するためのルーチン群が用意されている。

実装されている演算の正しい定義については、「[数学表記と定義](#)」のセクションを参照のこと。

現時点では、次の実装が提供されている。

- 1次元の単精度データ用フーリエ・アルゴリズム
- 1次元の単精度および倍精度データ用直接アルゴリズム
- 多次元の単精度および倍精度データ用直接アルゴリズム

1次元用のアルゴリズムは、IBM ESSL ライブラリの次の関数を備えている。

SCONF, SCORF

SCOND, SCORD

SDCON, SDCOR  
DDCON, DDCOR  
SDDCON, SDDCOR.

これらの ESSL 関数をシミュレートするために特殊なラッパーが用意されている。ラッパーは、**FORTRAN** および **C** のサンプルソースとして提供されている。それらを利用するには、次のディレクトリを使用する。

```
${MKL}/examples/vslc/essl/vsl_wrappers  
${MKL}/examples/vslf/essl/vsl_wrappers
```

また、次のディレクトリでは、ラッパーによる ESSL 関数の計算例を参照することもできる。

```
${MKL}/examples/vslc/essl  
${MKL}/examples/vslf/essl
```

畳み込み / 相関 API は、**FORTRAN-90** インターフェイスと **C/89** インターフェイスを提供する。

**C/89** インターフェイスは、**C/C++** のそれ以降のバージョンで 사용할 こともできる。また、**FORTRAN-90** は **FORTRAN-95** のプログラムと使用することができる。**FORTRAN-77** はサポートされない。

畳み込み / 相関 API は、タスク・オブジェクトやタスクによって実装される。タスク・オブジェクトは、特定の畳み込み / 相関演算を決定するパラメータを格納するデータ構造またはディスクリプタである。これらのパラメータは、精度、データ型、ユーザーデータの次元、使用する計算アルゴリズムの識別子、データ配列の形状などである。

すべてのインテル・マス・カーネル・ライブラリの畳み込み / 相関ルーチンは、タスク・ディスクリプタを新規に作成してパラメータの設定を変更し、格納されているパラメータを用いて畳み込みや相関の数値演算結果を計算するか、またはそのほかの演算を実行してタスク・オブジェクトを処理する。すべてのルーチンは以下のグループに分類される。

[タスク・コンストラクタ](#) - タスク・オブジェクト・ディスクリプタを作成して共通のパラメータを設定するルーチン。

[タスクエディタ](#) - 既存のタスク・ディスクリプタのパラメータ設定を設定または変更するルーチン。

[タスク実行ルーチン](#) - タスク・ディスクリプタに格納されている演算パラメータを用いて、実際の入力データに対する畳み込み / 相関演算の結果を計算する。

[タスクコピー](#) - タスク・ディスクリプタのコピーを作成するのに使用するルーチン。

[タスク・ディストラクタ](#) - タスク・オブジェクトを削除してメモリを開放するルーチン。

タスクを最初に実行またはコピーする場合、タスクの遂行と呼ばれる特殊処理を実行する。この処理では、タスク・パラメータの一貫性がチェックされ、必要な作業データが準備される。パラメータが一貫している場合、タスクが正常に遂行されたことを示すタグが付けられる。タスクは、パラメータが編集されるまで、遂行されたままである。そのため、1回の遂行処理でタスクを複数回実行することが可能である。タスクの遂行処理には、入力データのフーリエ変換の準備などコストのかかる中間計算が含まれるため、処理を一度だけ開始することにより全体の性能を向上させることができる。

## 命名規則

畳み込み / 相関 API では、FORTRAN ルーチン名は小文字で記述され、FORTRAN データ型と定数は大文字で記述される。これらの名前は大文字と小文字を区別しない。

C では、ルーチン、データ型、および定数の名前は大文字と小文字を区別し、大文字と小文字のどちらも使用することができる。

ルーチンの名前は、次の構造になっている。

```
vs1[datatype]{Conv|Corr}<base name>      (C インターフェイスの場合)
vs1[datatype]{conv|corr}<base name>        (FORTRAN インターフェイスの場合)
```

vs1 は、ルーチンがインテル・マス・カーネル・ライブラリのベクトル統計ライブラリに属していることを示すプリフィックスである。

[datatype] はオプションである。存在する場合、記号は入力データおよび出力データのデータ型を指定する。s (単精度実数型の場合) または d (倍精度実数型の場合) のいずれか。

プリフィックス Conv と Corr は、ルーチンが畳み込みタスクを参照するのか、または相関タスクを参照するのかを指定する。

<base name> フィールドは、NewTask や DeleteTask など、ルーチンの特性を示す。



**注:** この章では、曖昧にならない場合は、ルーチンの基本名を使用する。ルーチンのリファレンスでは、プロトタイプやコード例で常にフルネームを使用する。

## データ型

すべての畳み込み / 相関ルーチンは、データ・オブジェクトの指定に次のデータ型を使用する。

データ型		データ・オブジェクト
FORTTRAN	C	
TYPE (VSL_CONV_TASK)	VSLConvTaskPtr	畳み込みのタスク・ディスクリプタへのポインタ
TYPE (VSL_CORR_TASK)	VSLCorrTaskPtr	相関のタスク・ディスクリプタへのポインタ
REAL*4	float	単精度の入力 / 出力ユーザデータ
REAL*8	double	倍精度の入力 / 出力ユーザデータ
INTEGER	int	他のすべてのデータ

すべての整数データには一般整数型 ( バイトサイズの指定なし ) を使用する。



**注：**一般整数型の実際のサイズはプラットフォームに依存する。整数型の適切なバイトサイズは、ソフトウェアをコンパイルする段階で選択しなければならない。

## パラメータ

タスク・ディスクリプタに格納される基本パラメータは、タスク・オブジェクトがタスクエディタによって作成、コピー、または変更された際に割り当てられた値である。畳み込み / 相関タスクのパラメータは、タスク・オブジェクトが作成される際に、タスク・コンストラクタによって最初に設定される。パラメータの変更や追加設定はタスクエディタによって行われる。タスク実行ルーチンを呼び出す前に、畳み込みを行うデータの位置を定義するパラメータを指定する必要がある。

引き渡し方法や割り当てられた値に従って、すべてのパラメータは明示的 ( タスク・オブジェクトが作成時または実行時に、ルーチン・パラメータとして直接渡される ) あるいはオプション ( タスク構成時にデフォルトまたは暗黙的な値が割り当てられる ) のいずれかのカテゴリに分類される。

インテル・マス・カーネル・ライブラリの畳み込み / 相関 API で使用される、すべての適用可能なパラメータを次の表に示す。

**表 10-13 畳み込み / 相関タスクのパラメータ**

名前	カテゴリ	データ型	デフォルト値ラベル	説明
<i>job</i>	明示的	整数	コンストラクタの名前によって示される	タスクを畳み込みまたは相関のどちらに関連付けるかを指定する。
<i>type</i>	明示的	整数	コンストラクタの名前によって示される	入力 / 出力データのデータ型 ( 実数または複素数 ) を指定する。現在のバージョンでは実数に設定する。
<i>precision</i>	明示的	整数	コンストラクタの名前によって示される	配列 <i>x</i> 、 <i>y</i> 、 <i>z</i> で与えられる入力 / 出力データの精度 ( 単精度または倍精度 ) を指定する。
<i>kind</i>	オプション	整数	“ 線形 ”	タスクを線形または巡回のどちらの畳み込み / 相関演算に関連付けるかを指定する。
<i>mode</i>	明示的	整数	なし	畳み込み / 相関演算の実行方法 ( フーリエ変換、直接法、あるいは 2 つの間で自動的に選択 ) を指定する。このパラメータで使用する名前付き定数のリストは、 <a href="#">SetMode</a> を参照のこと。
<i>method</i>	オプション	整数	“ 自動 ”	与えられた <i>mode</i> に対して複数の方法が利用可能な場合、特定の計算方法を示す。このパラメータを “ 自動 ” に設定すると、ソフトウェアは利用可能な最良の方法を選択する。
<i>internal_precision</i>	オプション	整数	<i>precision</i> の値と等しくなるように設定する。	内部計算の精度を指定する。入力 / 出力データが単精度の場合でも倍精度計算を強制することができる。このパラメータで 사용되는名前付き定数のリストは、 <a href="#">SetInternalPrecision</a> を参照のこと。
<i>dims</i>	明示的	整数	なし	配列 <i>x</i> 、 <i>y</i> 、 <i>z</i> で与えられるユーザデータの階数 ( 次元 ) を指定する。範囲は 1 ~ 7。
<i>x, y</i>	明示的	実数配列	なし	入力データ配列を指定する。詳細は、 <a href="#">データの割り当て</a> を参照。
<i>z</i>	明示的	実数配列	なし	出力データ配列を指定する。詳細は、 <a href="#">データの割り当て</a> を参照。
<i>xshape, yshape, zshape</i>	明示的	整数配列	なし	配列 <i>x</i> 、 <i>y</i> 、 <i>z</i> の形状を定義する。詳細は、 <a href="#">データの割り当て</a> を参照。

表 10-13 畳み込み / 関連タスクのパラメータ ( 続き )

名前	カテゴリ	データ型	デフォルト値ラベル	説明
<i>xstride</i> , <i>ystride</i> , <i>zstride</i>	明示的	整数配列	なし	配列 <i>x</i> 、 <i>y</i> 、 <i>z</i> 内のストライドを定義する。ストライドは、それぞれの配列における入力 / 出力データの物理的な位置を指定する。詳細は、 <a href="#">データの割り当て</a> を参照。
<i>start</i>	オプション	整数配列	未定義	数値演算の結果の最初の成分を定義する。これは、出力配列 <i>z</i> に格納される。詳細は、 <a href="#">SetStart</a> と <a href="#">データの割り当て</a> を参照。
<i>decimation</i>	オプション	整数配列	未定義	数値演算の結果を減少させる方法を定義する。これは、出力配列 <i>z</i> に格納される。詳細は、 <a href="#">SetDecimation</a> と <a href="#">データの割り当て</a> を参照。

## タスク・ステータス

タスク・ステータスは整数値である。タスク処理中にエラーが検出されなかった場合はゼロに、それ以外の場合は特定の非ゼロのエラーコードになる。負の値はエラーに使用され、正の値は警告に使用される。

エラーは、不正なパラメータ値、メモリ割り当ての失敗などのシステム障害によって引き起こされる。また、ソフトウェアによって検出される内部エラーのこともある。

タスク・ディスクリプタはそれぞれ、タスクの現状ステータスを含んでいる。タスク・オブジェクトを作成する際、コンストラクタはタスクに `VSL_STATUS_OK` ステータスを割り当てる。後でタスクを処理する際にエラーが発生した場合、実行エディタなど他のルーチンは、タスク・ステータスを変更し、タスク・ステータス・フィールドに対応するエラーコードを書き込むことができる。

タスクの作成またはそのパラメータの編集段階では、パラメータのセットが一貫しないことがある。パラメータの一貫性のチェックは、タスクの実行またはコピー前に暗黙的に呼び出され、タスク遂行処理中にのみ実行される。この段階でエラーが検出されると、タスクの実行またはコピーは終了され、タスク・ディスクリプタは対応するエラーコードを保存する。一旦エラーが発生すると、そのタスク・ディスクリプタを処理するそれ以上の試みは終了され、タスクは同じエラーコードを保持する。

通常、それぞれの畳み込み / 関連関数 (DeleteTask を除く) は、関数を実行中にタスクに割り当てられたステータスを返す。

ステータス・コードは、それぞれのヘッダファイルで定義された記号名を与えられる。C/C++ ではこれらの名前は `#define` 構文によってマクロとして定義され、FORTRAN では `PARAMETER` 演算子によって整数定数として定義される。次に例を示す。

```
C/C++:      #define VSL_STATUS_OK 0
F90/F95:    INTEGER,PARAMETER:: VSL_STATUS_OK = 0
```

タスク・コンストラクタ

タスク・コンストラクタは、タスク・ディスクリプタを新規に作成して、基本パラメータを設定するためのルーチンである。これは、追加でパラメータを調整する必要がなく、他のルーチンがタスク・オブジェクトを使用できることを意味する。

インテル・マス・カーネル・ライブラリの畳み込み / 相関 API では、一般形式と X 形式の 2 つの異なる形式のコンストラクタを用意している。X 形式のコンストラクタは一般形式のコンストラクタと同じ働きをするが、畳み込み / 相関演算で使われる最初の演算子ベクトル ( 配列 *x* に格納される ) に特定のデータも割り当てる。

配列 *y* に格納された異なるベクトルに対して、配列 *x* に格納された同じデータベクトルを使用して複数の畳み込み / 相関演算を行う必要がある場合、X 形式のコンストラクタの使用が推奨される。これは、演算に必要な中間データの繰り返し計算で発生する不必要なオーバーヘッドを排除し、性能を向上させるためである。

各コンストラクタ・ルーチンに対して、関連する 1 次元のバージョンが存在する。このバージョンは、1 次元のデータ構造の単純性によってもたらされる、アルゴリズムおよび計算上の利点を活用する。



注：コンストラクタはタスク・ディスクリプタの作成に失敗すると、NULL タスクポインタを返す。

表 10-14 は、利用可能なタスク・コンストラクタをまとめたものである。

表 10-14 タスク・コンストラクタ

ルーチン	説明
<a href="#">NewTask</a>	多次元の畳み込み / 相関タスク・ディスクリプタを新規に作成する。
<a href="#">NewTask1D</a>	1 次元の畳み込み / 相関タスク・ディスクリプタを新規に作成する。

表 10-14 タスク・コンストラクタ ( 続き )

ルーチン	説明
<a href="#">NewTaskX</a>	多次元の畳み込み / 関連タスク・ディスクリプタを X 形式で新規に作成する。
<a href="#">NewTaskX1D</a>	1 次元の畳み込み / 関連タスク・ディスクリプタを X 形式で新規に作成する。

## NewTask

多次元の畳み込み / 関連タスク・ディスクリプタを新規に作成する。

### 構文

#### FORTRAN:

```
status = vslsconvnewtask(task, mode, dims, xshape, yshape, zshape)
status = vsldconvnewtask(task, mode, dims, xshape, yshape, zshape)
status = vslscorrnewtask(task, mode, dims, xshape, yshape, zshape)
status = vsldcorrnewtask(task, mode, dims, xshape, yshape, zshape)
```

#### C:

```
status = vslsConvNewTask(task, mode, dims, xshape, yshape, zshape);
status = vsldConvNewTask(task, mode, dims, xshape, yshape, zshape);
status = vslsCorrNewTask(task, mode, dims, xshape, yshape, zshape);
status = vsldCorrNewTask(task, mode, dims, xshape, yshape, zshape);
```

### 説明

各 NewTask コンストラクタは、明示的パラメータにユーザ指定の値を使用した新規の畳み込み / 関連タスク・ディスクリプタを作成する。オプション・パラメータはデフォルト値に設定される ( [表 10-13](#) を参照 )。

パラメータ *xshape*、*yshape*、*zshape* はそれぞれ、配列 *x*、*y*、*z* で与えられる入力 / 出力データの形状を定義する。各形状パラメータは、*dims* の値と同じ長さの整数配列である。

コンストラクタを呼び出す際に形状パラメータを明示的に割り当てる。パラメータ



*dims* の値が 1 の場合、*xshape*、*yshape*、*zshape* は配列 *x* と *y* から読み取られる成分の個数、または配列 *z* に格納される成分の個数に等しい。ストライドが割り当てられた場合、形状パラメータの値は配列 *x*、*y*、*z* の物理的形状とは異なることがある。

コンストラクタはタスク・ディスクリプタの作成に失敗すると、NULL タスクポインタを返す。

### 入力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>mode</i>	INTEGER	int	畳み込み / 相関演算を直接法を使用して実行すべきか、または入力データのフーリエ変換によって実行すべきかを指定する。 <a href="#">表 10-16</a> は指定可能な値をまとめたものである。
<i>dims</i>	INTEGER	int	ユーザデータの階数 実行段階で使用される入力 / 出力配列 <i>x</i> 、 <i>y</i> 、 <i>z</i> の次元を指定する。範囲は 1 ～ 7 でなければならない。値はコンストラクタによって明示的に割り当てられる。
<i>xshape</i>	INTEGER, DIMENSION( *)	int[ ]	ソース配列 <i>x</i> の入力データの形状を定義する。詳細は、 <a href="#">データの割り当て</a> を参照。
<i>yshape</i>	INTEGER, DIMENSION( *)	int[ ]	ソース配列 <i>y</i> の入力データの形状を定義する。詳細は、 <a href="#">データの割り当て</a> を参照。
<i>zshape</i>	INTEGER, DIMENSION( *)	int[ ]	配列 <i>z</i> に格納される出力データの形状を定義する。詳細は、 <a href="#">データの割り当て</a> を参照。

## 出力パラメータ

名前	データ型		説明
	FORTRAN	C	
<i>task</i>	TYPE(VSL_CONV_TASK) (vslsconvnewtask、 vsldconvnewtask の場 合 ) TYPE(VSL_CORR_TASK) (vslscorrnewtask、 vsldcorrnewtask の場 合 )	VSLConvTaskPtr* (vslsConvNewTask、 vsldConvNewTask の 場合 ) VSLCorrTaskPtr* (vslsCorrNewTask、 vsldCorrNewTask の 場合 )	正常に作成された場合は、タ スク・ディスクリプタへのポ インタ。それ以外の場合は、 NULL ポインタ。
<i>status</i>	INTEGER	int	タスクが正常に作成された場 合は VSL_STATUS_OK に設 定し、それ以外の場合は非ゼロ のエラーコードに設定する。

## NewTask1D

1 次元の畳み込み / 関連タスク・ディスクリプタ  
を新規に作成する。

## 構文

## FORTRAN:

```
status = vslsconvnewtask1d(task, mode, xshape, yshape, zshape)
status = vsldconvnewtask1d(task, mode, xshape, yshape, zshape)
status = vslscorrnewtask1d(task, mode, xshape, yshape, zshape)
status = vsldcorrnewtask1d(task, mode, xshape, yshape, zshape)
```

## C:

```
status = vslsConvNewTask1D(task, mode, xshape, yshape, zshape);
status = vsldConvNewTask1D(task, mode, xshape, yshape, zshape);
status = vslsCorrNewTask1D(task, mode, xshape, yshape, zshape);
status = vsldCorrNewTask1D(task, mode, xshape, yshape, zshape);
```

説明

各 NewTask1D コンストラクタは、明示的パラメータにユーザ指定の値を使用した新規の畳み込み / 相関タスク・ディスクリプタを作成する。オプション・パラメータはデフォルト値に設定される ([表 10-13](#) を参照)。  
[NewTask](#) と異なり、これらのルーチンは、パラメータ [dims](#) の値が 1 であると仮定するコンストラクタの特殊な 1 次元バージョンを表す。  
パラメータ *xshape*、*yshape*、*zshape* は、配列 *x* と *y* から読み出される成分の個数、または配列 *z* に格納される成分の個数に等しい。コンストラクタを呼び出す際に形状パラメータを明示的に割り当てる。

入力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>mode</i>	INTEGER	int	畳み込み / 相関演算を直接法を使用して実行すべきか、または入力データのフーリエ変換によって実行すべきかを指定する。 <a href="#">表 10-16</a> は指定可能な値をまとめたものである。
<i>xshape</i>	INTEGER	int	ソース配列 <i>x</i> の入力データ・シーケンスの長さを定義する。詳細は、 <a href="#">データの割り当て</a> を参照。
<i>yshape</i>	INTEGER	int	ソース配列 <i>y</i> の入力データ・シーケンスの長さを定義する。詳細は、 <a href="#">データの割り当て</a> を参照。
<i>zshape</i>	INTEGER	int	配列 <i>z</i> に格納される出力データ・シーケンスの長さを定義する。詳細は、 <a href="#">データの割り当て</a> を参照。

## 出力パラメータ

名前	データ型		説明
	FORTRAN	C	
<i>task</i>	TYPE(VSL_CONV_TASK) ( <i>vslsconvnewtask1d</i> 、 <i>vsldconvnewtask1d</i> の場合) TYPE(VSL_CORR_TASK) ( <i>vsldcorrnewtask1d</i> 、 <i>vsldcorrnewtask1d</i> の場合)	VSLConvTaskPtr* ( <i>vslsConvNewTask1D</i> 、 <i>vsldConvNewTask1D</i> の場合) VSLCorrTaskPtr* ( <i>vsldCorrNewTask1D</i> 、 <i>vsldCorrNewTask1D</i> の場合)	正常に作成された場合は、タスク・ディスクリプタへのポインタ。それ以外の場合は、NULL ポインタ。
<i>status</i>	INTEGER	int	タスクが正常に作成された場合は VSL_STATUS_OK に設定し、それ以外の場合は非ゼロのエラーコードに設定する。

## NewTaskX

多次元の畳み込み / 相関タスク・ディスクリプタを新規に作成して、最初の演算子ベクトルにソースデータを割り当てる。

## 構文

## FORTRAN:

```
status = vslsconvnewtaskx(task, mode, dims, xshape, yshape, zshape, x, xstride)
status = vsldconvnewtaskx(task, mode, dims, xshape, yshape, zshape, x, xstride)
status = vsldcorrnewtaskx(task, mode, dims, xshape, yshape, zshape, x, xstride)
status = vsldcorrnewtaskx(task, mode, dims, xshape, yshape, zshape, x, xstride)
```

## C:

```
status = vslsConvNewTaskX(task, mode, dims, xshape, yshape, zshape, x,
                           xstride);
```

```
status = vsldConvNewTaskX(task, mode, dims, xshape, yshape, zshape, x,
                           xstride);
status = vslsCorrNewTaskX(task, mode, dims, xshape, yshape, zshape, x,
                           xstride);
status = vsldCorrNewTaskX(task, mode, dims, xshape, yshape, zshape, x,
                           xstride);
```

## 説明

各 NewTaskX コンストラクタは、明示的パラメータにユーザ指定の値を使用した新規の畳み込み / 相関タスク・ディスクリプタを作成する。オプション・パラメータはデフォルト値に設定される ([表 10-13](#) を参照)。

[NewTask](#) と異なり、これらのルーチンは、コンストラクタの X 形式バージョンを表す。X 形式バージョンは、タスク・ディスクリプタの作成に加え、畳み込み / 相関演算で使用する配列 *x* の最初の演算子ベクトルに特定のデータを割り当てる。NewTaskX コンストラクタによって作成されたタスク・ディスクリプタは、デストラクタ・ルーチンによってタスク・オブジェクトが削除されるまで常に配列 *x* へのポインタを保持する ([DeleteTask](#) を参照)。

配列 *y* に格納された異なるベクトルに対して、配列 *x* に格納された同じデータベクトルを使用して複数の畳み込み / 相関演算を行う必要がある場合、この形式のコンストラクタの使用が推奨される。これは、演算に必要な中間データの繰り返し計算で発生する不必要なオーバーヘッドを排除し、性能を向上させるためである。

パラメータ *xshape*、*yshape*、*zshape* はそれぞれ、配列 *x*、*y*、*z* で与えられる入力 / 出力データの形状を定義する。各形状パラメータは、*dims* の値と同じ長さの整数配列である。

コンストラクタを呼び出す際に形状パラメータを明示的に割り当てる。パラメータ *dims* の値が 1 の場合、*xshape*、*yshape*、*zshape* は配列 *x* と *y* から読み取られる成分の個数、または配列 *z* に格納される成分の個数に等しい。ストライドが割り当てられた場合、形状パラメータの値は配列 *x*、*y*、*z* の物理的形状とは異なることがある。

ストライド・パラメータ *xstride* は、配列 *x* における入力データの物理的な位置を指定する。

1 次元の場合、ストライドは配列の続く成分間の区間である。例えば、パラメータ *xstride* の値が *s* の場合、配列 *x* の *s* 番目ごとの成分のみ、入力シーケンスを形成するのに使用される。ストライド値は、正または負でなければならない。ゼロであってはならない。

## 入力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>mode</i>	INTEGER	int	畳み込み / 相関演算を直接法を使用して実行すべきか、または入力データのフーリエ変換によって実行すべきかを指定する。 <a href="#">表 10-16</a> は指定可能な値をまとめたものである。
<i>dims</i>	INTEGER	int	ユーザデータの階数 実行段階で使用する入力 / 出力配列 <i>x</i> 、 <i>y</i> 、 <i>z</i> の次元を指定する。範囲は 1 ～ 7 でなければならない。値はコンストラクタによって明示的に割り当てられる。
<i>xshape</i>	INTEGER, DIMENSION(*)	int[]	ソース配列 <i>x</i> の入力データの形状を定義する。詳細は、 <a href="#">データの割り当て</a> を参照。
<i>yshape</i>	INTEGER, DIMENSION(*)	int[]	ソース配列 <i>y</i> の入力データの形状を定義する。詳細は、 <a href="#">データの割り当て</a> を参照。
<i>zshape</i>	INTEGER, DIMENSION(*)	int[]	配列 <i>z</i> に格納される出力データの形状を定義する。詳細は、 <a href="#">データの割り当て</a> を参照。
<i>x</i>	REAL*4, DIMENSION(*) ( 単精度の場合 ) REAL*8, DIMENSION(*) ( 倍精度の場合 )	float[] ( 単精度の場合 ) double[] ( 倍精度の場合 )	最初の演算子ベクトルの入力データを含む配列へのポインタ。詳細は、 <a href="#">データの割り当て</a> を参照。
<i>xstride</i>	INTEGER, DIMENSION(*)	int[]	配列 <i>x</i> の入力データのストライド

## 出力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>task</i>	TYPE(VSL_CONV_TASK) ( <i>vslsconvnewtaskx</i> 、 <i>vsldconvnewtaskx</i> の場合) TYPE(VSL_CORR_TASK) ( <i>vsldcorrnewtaskx</i> 、 <i>vsldcorrnewtaskx</i> の場合)	VSLConvTaskPtr* ( <i>vslsConvNewTaskX</i> 、 <i>vsldConvNewTaskX</i> の場合) VSLCorrTaskPtr* ( <i>vsldCorrNewTaskX</i> 、 <i>vsldCorrNewTaskX</i> の場合)	正常に作成された場合は、タスク・ディスクリプタへのポインタ。それ以外の場合は、NULL ポインタ。
<i>status</i>	INTEGER	int	タスクが正常に作成された場合は VSL_STATUS_OK に設定し、それ以外の場合は非ゼロのエラーコードに設定する。

## NewTaskX1D

1 次元の畳み込み / 相関タスク・ディスクリプタを新規に作成して、最初の演算子ベクトルにソースデータを割り当てる。

## 構文

## FORTRAN:

```
status = vslsconvnewtaskx1d(task, mode, xshape, yshape, zshape, x, xstride)
status = vsldconvnewtaskx1d(task, mode, xshape, yshape, zshape, x, xstride)
status = vsldcorrnewtaskx1d(task, mode, xshape, yshape, zshape, x, xstride)
status = vsldcorrnewtaskx1d(task, mode, xshape, yshape, zshape, x, xstride)
```

## C:

```
status = vslsConvNewTaskX1D(task, mode, xshape, yshape, zshape, x, xstride);
status = vsldConvNewTaskX1D(task, mode, xshape, yshape, zshape, x, xstride);
status = vsldCorrNewTaskX1D(task, mode, xshape, yshape, zshape, x, xstride);
status = vsldCorrNewTaskX1D(task, mode, xshape, yshape, zshape, x, xstride);
```

## 説明

各 `NewTaskX1D` コンストラクタは、明示的パラメータにユーザ指定の値を使用した新規の畳み込み / 相関タスク・ディスクリプタを作成する。オプション・パラメータはデフォルト値に設定される ([表 10-13](#) を参照)。

これらのルーチンは、コンストラクタの `X` 形式の特殊な 1 次元バージョンを表す。パラメータ `dims` の値は 1 であると仮定し、タスク・ディスクリプタの作成に加え、コンストラクタ・ルーチンは畳み込み / 相関演算で使用される配列 `x` の最初の演算子ベクトルに特定のデータを割り当てる。`NewTaskX1D` コンストラクタによって作成されたタスク・ディスクリプタは、デストラクタ・ルーチンによってタスク・オブジェクトが削除されるまで常に配列 `x` へのポインタを保持する ([DeleteTask](#) を参照)。

配列 `y` に格納された異なるベクトルに対して、配列 `x` に格納された同じデータベクトルを使用して複数の畳み込み / 相関演算を行う必要がある場合、この形式のコンストラクタの使用が推奨される。これは、演算に必要な中間データの繰り返し計算で発生する不必要なオーバーヘッドを排除し、性能を向上させるためである。

パラメータ `xshape`、`yshape`、`zshape` は、配列 `x` と `y` から読み出される成分の個数、または配列 `z` に格納される成分の個数に等しい。コンストラクタを呼び出す際に形状パラメータを明示的に割り当てる。

ストライド・パラメータ `xstride` は、配列の続く成分間の区間であり、配列 `x` における入力データの物理的な位置を指定する。例えば、パラメータ `xstride` の値が `s` の場合、配列 `x` の `s` 番目ごとの成分のみ、入力シーケンスを形成するのに使用される。ストライド値は、正または負でなければならない。ゼロであってはならない。



## 入力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>mode</i>	INTEGER	int	畳み込み / 相関演算を直接法を使用して実行すべきか、または入力データのフーリエ変換によって実行すべきかを指定する。 <a href="#">表 10-16</a> は指定可能な値をまとめたものである。
<i>xshape</i>	INTEGER	int	ソース配列 <i>x</i> の入力データ・シーケンスの長さを定義する。詳細は、 <a href="#">データの割り当て</a> を参照。
<i>yshape</i>	INTEGER	int	ソース配列 <i>y</i> の入力データ・シーケンスの長さを定義する。詳細は、 <a href="#">データの割り当て</a> を参照。
<i>zshape</i>	INTEGER	int	配列 <i>z</i> に格納される出力データ・シーケンスの長さを定義する。詳細は、 <a href="#">データの割り当て</a> を参照。
<i>x</i>	REAL*4, DIMENSION (*) (単精度の場合) REAL*8, DIMENSION (*) (倍精度の場合)	float[] (単精度の場合) double[] (倍精度の場合)	最初の演算子ベクトルの入力データを含む配列へのポインタ。詳細は、 <a href="#">データの割り当て</a> を参照。
<i>xstride</i>	INTEGER	int	配列 <i>x</i> の入力データ・シーケンスのストライド。

## 出力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>task</i>	TYPE(VSL_CONV_TASK) ( <i>vslsconvnewtaskx1d</i> 、 <i>vsldconvnewtaskx1d</i> の場合) TYPE(VSL_CORR_TASK) ( <i>vsldcorrnewtaskx1d</i> 、 <i>vsldcorrnewtaskx1d</i> の場合)	VSLConvTaskPtr* ( <i>vslsConvNewTaskX1D</i> 、 <i>vsldConvNewTaskX1D</i> の場合) VSLCorrTaskPtr* ( <i>vsldCorrNewTaskX1D</i> 、 <i>vsldCorrNewTaskX1D</i> の場合)	正常に作成された場合は、タスク・ディスクリプタへのポインタ。それ以外の場合は、NULL ポインタ。
<i>status</i>	INTEGER	int	タスクが正常に作成された場合は VSL_STATUS_OK に設定し、それ以外の場合は非ゼロのエラーコードに設定する。

## タスクエディタ

インテル・マス・カーネル・ライブラリ の畳み込み / 相関 API のタスクエディタは、次のタスク・パラメータの設定または変更を行うためのルーチンである (表 10-13 を参照)。

- `mode`
- `internal_precision`
- `start`
- `decimation`

各パラメータの設定または変更には、別々のルーチンが存在する。



**注:** タスク・ディスクリプタ構造のフィールドは、ソフトウェアに備えられたタスク・エディタ・ルーチンのセットによってのみアクセス可能である。

タスク・ディスクリプタの設定を変更するためにエディタルーチンを適用すると、タスクの遂行ステータスが失われ、次回の実行またはコピー時に再度、完全な遂行処理が行われる。これは、前回の遂行処理中に格納された現在の作業データは、新しいパラメータ設定に対して無効となることがあるためである。タスク遂行に関する詳細は、[概要](#)を参照のこと。

表 10-15 は利用可能なタスクエディタをまとめたものである。

**表 10-15      タスクエディタ**

ルーチン	説明
<a href="#">SetMode</a>	畳み込み / 相関演算のためにパラメータ <code>mode</code> の値を変更する。
<a href="#">SetInternalPrecision</a>	畳み込み / 相関演算のためにパラメータ <code>internal_precision</code> の値を変更する。
<a href="#">SetStart</a>	畳み込み / 相関演算のためにパラメータ <code>start</code> の値を設定する。
<a href="#">SetDecimation</a>	畳み込み / 相関演算のためにパラメータ <code>decimation</code> の値を設定する。



注：エディタルーチンの呼び出しに `NULL` タスクポインタを使用することができる。この場合、ルーチンは終了し、システム障害は発生しない。

## SetMode

畳み込み / 相関演算タスク・ディスクリプタの  
パラメータ `mode` の値を変更する。

### 構文

**FORTRAN:**

```
status = vslconvsetmode(task, newmode)
status = vslcorrsetmode(task, newmode)
```

**C:**

```
status = vslConvSetMode(task, newmode);
status = vslCorrSetMode(task, newmode);
```

### 説明

このルーチンは、畳み込み / 相関演算のためにパラメータ `mode` の値を変更する。このパラメータは、入力 / 出力データのフーリエ変換によって計算を行うか、または直接法を使用して計算を行うかを定義する。`mode` の初期値はタスク・コンストラクタによって割り当てられる。

`mode` パラメータの定義済み値を以下に示す。

表 10-16 `mode` パラメータの値

値	目的
VSL_CONV_MODE_FFT	高速フーリエ変換により畳み込み演算を行う。
VSL_CORR_MODE_FFT	高速フーリエ変換により相関演算を行う。
VSL_CONV_MODE_DIRECT	直接畳み込み演算を行う。
VSL_CORR_MODE_DIRECT	直接相関演算を行う。

表 10-16 *mode* パラメータの値 ( 続き )

値	目的
VSL_CONV_MODE_AUTO	畳み込み演算方法 ( 直接またはフーリエ ) を自動的に選択する。
VSL_CORR_MODE_AUTO	相関演算方法 ( 直接またはフーリエ ) を自動的に選択する。

## 入力パラメータ

名前	データ型		説明
	FORTRAN	C	
<i>task</i>	TYPE(VSL_CONV_TASK) (vslconvsetmode の場合) TYPE(VSL_CORR_TASK) (vslcorrsetmode の場合)	VSLConvTaskPtr (vslConvSetMode の場合) VSLCorrTaskPtr (vslCorrSetMode の場合)	タスク・ディスクリプタへのポインタ。
<i>newmode</i>	INTEGER	int	パラメータ <i>mode</i> の新しい値。

## 出力パラメータ

名前	データ型		説明
	FORTRAN	C	
<i>status</i>	INTEGER	int	タスクの現在のステータス。

## SetInternalPrecision

畳み込み / 相関演算タスク・ディスクリプタのパラメータ *internal\_precision* の値を変更する。

### 構文

#### FORTRAN:

```
status = vslconvsetinternalprecision(task, precision)
status = vslcorrsetinternalprecision(task, precision)
```

#### C:

```
status = vslConvSetInternalPrecision(task, precision);
status = vslCorrSetInternalPrecision(task, precision);
```

説明

このルーチンは、畳み込み / 相関演算のためにパラメータ `internal_precision` の値を変更する。このパラメータは、畳み込み / 相関結果の中間計算を単精度で行うべきか、または倍精度で行うべきか定義する。`internal_precision` の初期値はタスク・コンストラクタによって割り当てられ、使用されるコンストラクタのデータ型に従って“単精度”または“倍精度”のいずれかに設定される。

`internal_precision` を変更すると、デフォルト設定が“単精度”で、入力 / 出力データが単精度の場合でも倍精度の計算結果を求めることができる。

`internal_precision` 入力パラメータの定義済み値を以下に示す。

表 10-17 `internal_precision` パラメータの値

値	目的
VSL_CONV_PRECISION_SINGLE	単精度の畳み込み演算を行う。
VSL_CORR_PRECISION_SINGLE	単精度の相関演算を行う。
VSL_CONV_PRECISION_DOUBLE	倍精度の畳み込み演算を行う。
VSL_CORR_PRECISION_DOUBLE	倍精度の相関演算を行う。

入力パラメータ

名前	データ型		説明
	FORTRAN	C	
<code>task</code>	TYPE(VSL_CONV_TASK) ( <code>vslconvsetinternalprecision</code> の場合 ) TYPE(VSL_CORR_TASK) ( <code>vslcorrsetinternalprecision</code> の 場合 )	VSLConvTaskPtr ( <code>vslConvSetInternalPrecision</code> の場合 ) VSLCorrTaskPtr ( <code>vslCorrSetInternalPrecision</code> の場合 )	タスク・ディスクリ プタへのポインタ。
<code>precision</code>	INTEGER	int	パラメータ <code>internal_precision</code> の新しい値。

### 出力パラメータ

名前	データ型		説明
	FORTRAN	C	
<i>status</i>	INTEGER	int	タスクの現在のステータス。

## SetStart

畳み込み / 相関演算タスク・ディスクリプタの  
パラメータ *start* の値を変更する。

### 構文

#### FORTRAN:

```
status = vslconvsetstart(task, start)
status = vslcorrsetstart(task, start)
```

#### C:

```
status = vslConvSetStart(task, start);
status = vslCorrSetStart(task, start);
```

### 説明

このルーチンは、畳み込み / 相関演算のためにパラメータ *start* の値を設定する。1次元の場合、このパラメータは出力配列に格納される数値演算結果の最初の成分を指す。多次元の場合、*start* はインデックスの配列であり、その長さはパラメータ *dims* で指定された次元に等しい。このパラメータの定義と効果に関する詳細は、[データの割り当て](#)を参照のこと。

タスク・ディスクリプタの初期構成時、*start* のデフォルト値は未定義で、このパラメータは使用されていない。そのため、*start* パラメータを設定して使用する唯一の方法は、SetStart ルーチンを使用して値を割り当てることである。

## 入力パラメータ

名前	データ型		説明
	FORTRAN	C	
<i>task</i>	TYPE(VSL_CONV_TASK) ( <i>vslconvsetstart</i> の場合) TYPE(VSL_CORR_TASK) ( <i>vslcorrsetstart</i> の場合)	VSLConvTaskPtr ( <i>vslConvSetStart</i> の場合) VSLCorrTaskPtr ( <i>vslCorrSetStart</i> の場合)	タスク・ディスクリプタへのポインタ。
<i>start</i>	INTEGER, DIMENSION (*)	int[]	パラメータ <i>start</i> の新しい値。

## 出力パラメータ

名前	データ型		説明
	FORTRAN	C	
<i>status</i>	INTEGER	int	タスクの現在のステータス。

---

SetDecimation

畳み込み / 相関演算タスク・ディスクリプタのパラメータ *decimation* の値を変更する。

---

## 構文

## FORTRAN:

```
status = vslconvsetdecimation(task, decimation)
status = vslcorrsetdecimation(task, decimation)
```

## C:

```
status = vslConvSetDecimation(task, decimation);
status = vslCorrSetDecimation(task, decimation);
```

## 説明

このルーチンは、畳み込み / 相関演算のためにパラメータ *decimation* の値を設定する。

このパラメータは、畳み込み / 相関の数値演算結果を、出力データ配列に書き込む前に減少させる方法を定義する。例えば 1 次元の場合、*decimation* = *d* > 1 ならば数値演算結果の *d* 番目ごとの成分のみ出力配列に書き込まれる。

多次元の場合、*decimation* はインデックスの配列であり、その長さはパラメータ *dims* で指定された次元に等しい。このパラメータの定義と効果に関する詳細は、[データの割り当て](#)を参照のこと。

タスク・ディスクリプタの初期構成時、*decimation* のデフォルト値は未定義で、このパラメータは使用されていない。そのため、*decimation* パラメータを設定して使用する唯一の方法は、SetDecimation ルーチンを使用して値を割り当てることである。

### 入力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>task</i>	TYPE(VSL_CONV_TASK) (vslconvsetdecimation の場合) TYPE(VSL_CORR_TASK) (vslcorrsetdecimation の場合)	VSLConvTaskPtr (vslConvSetDecimation の場合) VSLCorrTaskPtr (vslCorrSetDecimation の場合)	タスク・ディスクリプタへのポインタ。
<i>start</i>	INTEGER, DIMENSION (*)	int[]	パラメータ <i>decimation</i> の新しい値。

### 出力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>status</i>	INTEGER	int	タスクの現在のステータス。



## タスク実行ルーチン

タスク実行ルーチンは、タスク・ディスクリプタに格納されたパラメータと入力ベクトルに対して与えられたユーザデータを基に畳み込み / 相関結果を計算する。

一旦作成および調整されると、タスクは同じデータ型、精度、および形状の異なる入力 / 出力データに適用することによって複数回実行することができる。

インテル・マス・カーネル・ライブラリの畳み込み / 相関 API では、一般形式と X 形式の 2 つの異なる形式の実行ルーチンを用意している。一般形式の実行ルーチンは、一般形式のコンストラクタによって作成されたタスク・ディスクリプタを使用し、入力時に 2 つのソースデータ配列  $x$  と  $y$  を取得する。一方、X 形式の実行ルーチンは、X 形式のコンストラクタによって作成されたタスク・ディスクリプタを使用する。最初の配列  $x$  は構成時にすでに指定されているため、入力時には 1 つのソースデータ配列  $y$  のみを取得する。

タスクを最初に実行する際、実行ルーチンはタスク遂行処理を行う。この処理は、パラメータの一貫性のチェックと補助データの準備（例えば、入力データのフーリエ変換の計算など）の基本的な 2 つのステップで構成される。

各実行ルーチンに対して、関連する 1 次元のバージョンが存在する。このバージョンは、1 次元のデータ構造の単純性によってもたらされる、アルゴリズムおよび計算上の利点を活用する。



**注：**実行ルーチンの呼び出しに NULL タスクポインタを使用することができる。この場合、ルーチンは終了し、システム障害は発生しない。

タスクが正常に実行されると、実行ルーチンはゼロ・ステータス・コードを返す。エラーが検出されると、実行ルーチンは特定のエラーの発生を知らせるエラーコードを返す。次の場合、エラー・ステータス・コードが返される。

- タスクポインタが NULL の場合
- タスク・ディスクリプタが壊れている場合
- その他の理由で計算に失敗した場合

エラーが発生した場合、タスク・ディスクリプタはエラーコードを格納する。

すべてのタスク実行ルーチンを次の表に示す。

**表 10-18      タスク実行ルーチン**

ルーチン	説明
<a href="#">Exec</a>	多次元の畳み込み / 相関演算を行う。
<a href="#">Exec1D</a>	1 次元の畳み込み / 相関演算を行う。
<a href="#">ExecX</a>	多次元の畳み込み / 相関演算を X 形式として行う。
<a href="#">ExecX1D</a>	1 次元の畳み込み / 相関演算を X 形式として行う。

## Exec

多次元の畳み込み / 相関演算を行う。

### 構文

#### FORTRAN:

```
status = vslsconvexec(task, x, xstride, y, ystride, z, zstride)
status = vsldconvexec(task, x, xstride, y, ystride, z, zstride)
status = vslscorrexec(task, x, xstride, y, ystride, z, zstride)
status = vsldcorrexec(task, x, xstride, y, ystride, z, zstride)
```

#### C:

```
status = vslsConvExec(task, x, xstride, y, ystride, z, zstride);
status = vsldConvExec(task, x, xstride, y, ystride, z, zstride);
status = vslsCorrExec(task, x, xstride, y, ystride, z, zstride);
status = vsldCorrExec(task, x, xstride, y, ystride, z, zstride);
```

### 説明

各 Exec ルーチンは、配列 *x* と *y* で与えられたデータの畳み込み / 相関を計算し、その結果を配列 *z* に格納する。処理のパラメータは、対応する [NewTask](#) コンストラクタによって以前に作成され、*task* によってポイントされたタスク・ディスクリプタから読み出される。

*task* が NULL の場合、処理は行われない。

ストライド・パラメータ *xstride*、*ystride*、*zstride* はそれぞれ、配列 *x*、*y*、*z* の入力 / 出力データの物理的な位置を指定する。1 次元の場合、ストライドは配列の続く成分間の区間である。例えば、パラメータ *zstride* の値が *s* の場合、配列 *z* の *s* 番目ごとの成分のみ、出力データを格納するのに使用される。ストライド値は、正または負でなければならない。ゼロであってはならない。

### 入力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>task</i>	TYPE(VSL_CONV_TASK) ( <i>vslsconvexec</i> 、 <i>vsldconvexec</i> の場合) TYPE(VSL_CORR_TASK) for( <i>vslscorrexec</i> 、 <i>vsldcorrexec</i> の場合)	VSLConvTaskPtr ( <i>vslsConvExec</i> 、 <i>vsldConvExec</i> の場合) VSLCorrTaskPtr ( <i>vslsCorrExec</i> 、 <i>vsldCorrExec</i> の場合)	タスク・ディスクリプタへのポインタ。
<i>x</i> , <i>y</i>	REAL*4, DIMENSION(*) ( <i>vslsconvexec</i> 、 <i>vslscorrexec</i> の場合) REAL*8, DIMENSION(*) ( <i>vsldconvexec</i> 、 <i>vsldcorrexec</i> の場合)	float[] ( <i>vslsConvExec</i> 、 <i>vslsCorrExec</i> の場合) double[] ( <i>vsldConvExec</i> 、 <i>vsldCorrExec</i> の場合)	入力データを含む配列へのポインタ。詳細は、 <a href="#">データの割り当て</a> を参照。
<i>xstride</i> 、 <i>ystride</i> 、 <i>zstride</i>	INTEGER, DIMENSION(*)	int[]	入力 / 出力データのストライド。詳細は、 <a href="#">データの割り当て</a> を参照。

### 出力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>z</i>	REAL*4, DIMENSION(*) ( <i>vslsconvexec</i> 、 <i>vslscorrexec</i> の場合) REAL*8, DIMENSION(*) ( <i>vsldconvexec</i> 、 <i>vsldcorrexec</i> の場合)	float[] ( <i>vslsConvExec</i> 、 <i>vslsCorrExec</i> の場合) double[] ( <i>vsldConvExec</i> 、 <i>vsldCorrExec</i> の場合)	出力データを格納する配列へのポインタ。詳細は、 <a href="#">データの割り当て</a> を参照。
<i>status</i>	INTEGER	int	タスクが正常に実行された場合は VSL_STATUS_OK に設定し、それ以外の場合は非ゼロのエラーコードに設定する。

## Exec1D

1 次元の畳み込み / 相関演算を行う。

---

### 構文

#### FORTTRAN:

```
status = vslsconvexec1d(task, x, xstride, y, ystride, z, zstride)
status = vsldconvexec1d(task, x, xstride, y, ystride, z, zstride)
status = vslscorrexec1d(task, x, xstride, y, ystride, z, zstride)
status = vsldcorrexec1d(task, x, xstride, y, ystride, z, zstride)
```

#### C:

```
status = vslsConvExec1D(task, x, xstride, y, ystride, z, zstride);
status = vsldConvExec1D(task, x, xstride, y, ystride, z, zstride);
status = vslsCorrExec1D(task, x, xstride, y, ystride, z, zstride);
status = vsldCorrExec1D(task, x, xstride, y, ystride, z, zstride);
```

### 説明

各 Exec1D ルーチンは、配列 *x* と *y* で与えられたデータの畳み込み / 相関を計算し、その結果を配列 *z* に格納する。これらのルーチンは、パラメータ [dims](#) の値が 1 であると仮定し、演算の特殊な 1 次元のバージョンを表す。実行ルーチンのこのバージョンを使用すると、1 次元データにおける性能を向上させることができる。

処理のパラメータは、対応する [NewTask1D](#) コンストラクタによって以前に作成され、*task* によってポイントされたタスク・ディスクリプタから読み出される。*task* が NULL の場合、処理は行われない。

## 入力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>task</i>	TYPE(VSL_CONV_TASK) (vslsconvexec1d、 vsldconvexec1d の場合) TYPE(VSL_CORR_TASK) (vslscorrexec1d、 vsldcorrexec1d の場合)	VSLConvTaskPtr (vslsConvExec1D、 vsldConvExec1D の場 合) VSLCorrTaskPtr (vslsCorrExec1D、 vsldCorrExec1D の場 合)	タスク・ディスクリプタへの ポインタ。
<i>x , y</i>	REAL*4, DIMENSION(*) (vslsconvexec1d、 vslscorrexec1d の場合) REAL*8, DIMENSION(*) (vsldconvexec1d、 vsldcorrexec1d の場合)	float[] (vslsConvExec1D、 vslsCorrExec1D の場 合) double[] (vsldConvExec1D、 vsldCorrExec1D の場 合)	入力データを含む配列へのポ インタ。詳細は、 <a href="#">データの割 り当て</a> を参照。
<i>xstride, ystride, zstride</i>	INTEGER	int	入力/出力データのストライ ド。詳細は <a href="#">ストライド・パラ メータ</a> を参照のこと。

## 出力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>z</i>	REAL*4, DIMENSION(*) (vslsconvexec1d、 vslscorrexec1d の場合) REAL*8, DIMENSION(*) (vsldconvexec1d、 vsldcorrexec1d の場合)	float[] (vslsConvExec1D、 vslsCorrExec1D の 場合) double[] (vsldConvExec1D、 vsldCorrExec1D の 場合)	出力データを格納する配列への ポインタ。詳細は、 <a href="#">データの割 り当て</a> を参照。
<i>status</i>	INTEGER	int	タスクが正常に実行された場合 は VSL_STATUS_OK に設定 し、それ以外の場合は非ゼロの エラーコードに設定する。

---

## ExecX

最初の演算子ベクトルが固定である多次元の  
畳み込み / 相関演算を行う。

---

### 構文

#### FORTTRAN:

```
status = vslsconvexec(task, y, ystride, z, zstride)
status = vsldconvexec(task, y, ystride, z, zstride)
status = vslscorrexec(task, y, ystride, z, zstride)
status = vsldcorrexec(task, y, ystride, z, zstride)
```

#### C:

```
status = vslsConvExecX(task, y, ystride, z, zstride);
status = vsldConvExecX(task, y, ystride, z, zstride);
status = vslsCorrExecX(task, y, ystride, z, zstride);
status = vsldCorrExecX(task, y, ystride, z, zstride);
```

### 説明

各 ExecX ルーチンは、配列 *x* と *y* で与えられたデータの畳み込み / 相関を計算し、その結果を配列 *z* に格納する。これらのルーチンは、最初の演算子ベクトルはタスク構造時に設定され、タスクポインタは配列 *x* へのポインタを保持すると仮定する、演算の特殊なバージョンを表す。

処理のパラメータは、対応する [NewTaskX](#) コンストラクタによって以前に作成され、*task* によってポイントされたタスク・ディスクリプタから読み出される。*task* が NULL の場合、処理は行われない。

配列 *y* に格納された異なるベクトルに対して、配列 *x* に格納された同じデータベクトルを使用して複数の畳み込み / 相関演算を行う必要がある場合、この形式の実行ルーチンの使用が推奨される。これは、演算に必要な中間データの繰り返し計算で発生する不要なオーバーヘッドを排除し、性能を向上させるためである。

## 入力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>task</i>	TYPE(VSL_CONV_TASK) (vslsconvexecx、 vsldconvexecx の場合) TYPE(VSL_CORR_TASK) (vslscorrexecx、 vsldcorrexecx の場合)	VSLConvTaskPtr (vslsConvExecX、 vsldConvExecX の場 合) VSLCorrTaskPtr (vslsCorrExecX、 vsldCorrExecX の場 合)	タスク・ディスクリプタへの ポインタ。
<i>y</i>	REAL*4, DIMENSION(*) (vslsconvexecx、 vslscorrexecx の場合) REAL*8, DIMENSION(*) (vsldconvexecx、 vsldcorrexecx の場合)	float[] (vslsConvExecX、 vslsCorrExecX の場 合) double[] (vsldConvExecX、 vsldCorrExecX の場 合)	2 番目の演算子ベクトルの入 力データを含む配列へのポイ ンタ。詳細は、 <a href="#">データの割り 当て</a> を参照。
<i>ystride, zstride</i>	INTEGER, DIMENSION(*)	int[]	入力/出力データのストライ ド。詳細は <a href="#">ストライド・パラ メータ</a> を参照のこと。

## 出力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>z</i>	REAL*4, DIMENSION(*) (vslsconvexecx、 vslscorrexecx の場合) REAL*8, DIMENSION(*) (vsldconvexecx、 vsldcorrexecx の場合)	float[] (vslsConvExecX、 vslsCorrExecX の場 合) double[] (vsldConvExecX、 vsldCorrExecX の場 合)	出力データを格納する配列への ポインタ。詳細は、 <a href="#">データの割 り当て</a> を参照。
<i>status</i>	INTEGER	int	タスクが正常に実行された場合 は VSL_STATUS_OK に設定 し、それ以外の場合は非ゼロの エラーコードに設定する。

---

## ExecX1D

最初の演算子ベクトルが固定である 1 次元の  
畳み込み / 相関演算を行う。

---

### 構文

#### FORTTRAN:

```
status = vslsconvexecx1d(task, y, ystride, z, zstride)
status = vsldconvexecx1d(task, y, ystride, z, zstride)
status = vslscorrexx1d(task, y, ystride, z, zstride)
status = vsldcorrexx1d(task, y, ystride, z, zstride)
```

#### C:

```
status = vslsConvExecX1D(task, y, ystride, z, zstride);
status = vsldConvExecX1D(task, y, ystride, z, zstride);
status = vslsCorrExecX1D(task, y, ystride, z, zstride);
status = vsldCorrExecX1D(task, y, ystride, z, zstride);
```

### 説明

各 ExecX1D ルーチンは、配列  $x$  と  $y$  で与えられたデータの畳み込み / 相関を ( $\text{dims} = 1$  と仮定して) 計算し、その結果を配列  $z$  に格納する。これらのルーチンは、最初の演算子ベクトルがタスク構造時に設定されることを前提とする、演算の特殊なバージョンを表す。

処理のパラメータは、対応する [NewTaskX1D](#) コンストラクタによって以前に作成され、 $task$  によってポイントされたタスク・ディスクリプタから読み出される。 $task$  が NULL の場合、処理は行われない。

配列  $y$  に格納された異なるベクトルに対して、配列  $x$  に格納された同じデータベクトルを使用して複数の 1 次元畳み込み / 相関演算を行う必要がある場合、この形式の実行ルーチンの使用が推奨される。これは、演算に必要な中間データの繰り返し計算で発生する不必要なオーバーヘッドを排除し、性能を向上させるためである。



## 入力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>task</i>	TYPE(VSL_CONV_TASK) (vslsconvexecx1d、 vsldconvexecx1d の場合) TYPE(VSL_CORR_TASK) (vsldcorrexecx1d、 vsldcorrexecx1d の場合)	VSLConvTaskPtr (vslsConvExecX1D、 vsldConvExecX1D の場合) VSLCorrTaskPtr (vsldCorrExecX1D、 vsldCorrExecX1D の場合)	タスク・ディスクリプタへのポインタ。
<i>y</i>	REAL*4, DIMENSION(*) (vslsconvexecx1d、 vsldconvexecx1d の場合) REAL*8, DIMENSION(*) (vsldconvexecx1d、 vsldcorrexecx1d の場合)	float[] (vsldConvExecX1D、 vsldCorrExecX1D の場合) double[] (vsldConvExecX1D、 vsldCorrExecX1D の場合)	2 番目の演算子ベクトルの入力データを含む配列へのポインタ。詳細は、 <a href="#">データの割り当て</a> を参照。
<i>ystride, zstride</i>	INTEGER	int	入力 / 出力データのストライド。詳細は <a href="#">ストライド・パラメータ</a> を参照のこと。

## 出力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>z</i>	REAL*4, DIMENSION(*) (vsldconvexecx1d、 vsldcorrexecx1d の場合) REAL*8, DIMENSION(*) (vsldconvexecx1d、 vsldcorrexecx1d の場合)	float[] (vsldConvExecX1D、 vsldCorrExecX1D の場合) double[] (vsldConvExecX1D、 vsldCorrExecX1D の場合)	出力データを格納する z 列へのポインタ。詳細は、 <a href="#">データの割り当て</a> を参照。
<i>status</i>	INTEGER	int	タスクが正常に実行された場合は VSL_STATUS_OK に設定し、それ以外の場合は非ゼロのエラーコードに設定する。

## タスク・ディストラクタ

タスク・ディストラクタは、タスク・オブジェクトを削除し、メモリを開放するためのルーチンである。

---

### DeleteTask

タスク・オブジェクトを削除し、メモリを開放する。

---

#### 構文

##### FORTRAN:

```
errcode = vslconvdeletetask(task)
errcode = vslcorrdeletetask(task)
```

##### C:

```
errcode = vslConvDeleteTask(task);
errcode = vslCorrDeleteTask(task);
```

#### 説明

タスク・ディスクリプタへのポインタを与えられると、このルーチンはタスク・ディスクリプタ・オブジェクトを削除し、データ構造に割り当てられていたメモリを開放する。タスクに作業メモリがある場合、作業メモリも開放される。タスクポインタは NULL に設定される。

何らかの理由でタスクが正常に削除されない場合、ルーチンはエラーコードを返す。このエラーコードは、タスク・ステータスとは関係がなく、タスク・ステータスを変更しない。



---

**注:** デストラクタ・ルーチンの呼び出しに NULL タスクポインタを使用することができる。この場合、ルーチンは終了し、システム障害は発生しない。

---

### 入力パラメータ

名前	データ型		説明
	FORTRAN	C	
<i>task</i>	TYPE(VSL_CONV_TASK) (vslconvdeletetask の場合) TYPE(VSL_CORR_TASK) (vslcorrdeletetask の場合)	VSLConvTaskPtr* (vslConvDeleteTask の場合) VSLCorrTaskPtr* (vslCorrDeleteTask の場合)	タスク・ディスクリプタへのポインタ。

### 出力パラメータ

名前	データ型		説明
	FORTRAN	C	
<i>errcode</i>	INTEGER	int	タスク・オブジェクトが正常に削除された場合は 0。エラーが発生した場合は、エラーコードを含む。

## タスクコピー

このルーチンは、畳み込み / 関連タスク・ディスクリプタをコピーする。

## CopyTask

畳み込み / 関連タスクのディスクリプタをコピーする。

### 構文

#### FORTRAN:

```
status = vslconvcopytask(newtask, srctask)
status = vslcorrcopytask(newtask, srctask)
```

#### C:

```
status = vslConvTaskCopy(newtask, srctask);
status = vslCorrTaskCopy(newtask, srctask);
```

## 説明

タスク・オブジェクト *srctask* がすでに存在する場合、適切な CopyTask ルーチンを使用して *newtask* にコピーを作成することができる。コピー後、元のタスク・オブジェクトと新しいタスク・オブジェクトは遂行される（タスク遂行については、[概要](#)を参照）。元のタスクが過去に遂行されていない場合、コピーを開始する前に遂行処理が暗黙的に呼び出される。元のタスクを遂行中にエラーが発生すると、ステータス・フィールドにエラーコードが格納される。コピー中にエラーが発生すると、このルーチンは新しいタスク・オブジェクトへの参照の代わりに NULL ポインタを返す。

## 入力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>srctask</i>	TYPE(VSL_CONV_TASK) (vslconvcopytask の場合) TYPE(VSL_CORR_TASK) (vslcorrcopytask の場合)	VSLConvTaskPtr (vslConvCopyTask の場合) VSLCorrTaskPtr (vslCorrCopyTask の場合)	元のタスク・ディスクリプタへのポインタ。

## 出力パラメータ

名前	データ型		説明
	FORTTRAN	C	
<i>newtask</i>	TYPE(VSL_CONV_TASK) (vslconvcopytask の場合) TYPE(VSL_CORR_TASK) (vslcorrcopytask の場合)	VSLConvTaskPtr* (vslConvCopyTask の場合) VSLCorrTaskPtr* (vslCorrCopyTask の場合)	新しいタスク・ディスクリプタへのポインタ。
<i>status</i>	INTEGER	int	元のタスクの現在のステータス。

## 使用法の例

このセクションでは、インテル・マス・カーネル・ライブラリを使用した一般的な畳み込み/相関演算(シングルスレッド化計算およびマルチスレッド化計算)の実行方法を示す。次の2つの関数例 `scond1` と `sconf1` は、IBM ESSL\* ライブラリの畳み込み/相関関数 `SCOND` と `SCONF` をシミュレートする。これらの関数はシングルスレッド化計算を前提としており、C/C++ コンパイラで 사용할 ことができる。

### 例 10-5 シングルスレッド化計算用関数 `scond1`

```
#include "mkl_vsl.h"

int acond1(
    float h[], int inch,
    float x[], int incx,
    float y[], int incy,
    int nh, int nx, int iy0, int ny)
{
    int status;
    VSLConvTaskPtr task;
    vslsConvNewTask1D(&task, VSL_CONV_MODE_DIRECT, nh, nx, ny);
    vslConvSetStart(task, &iy0);
    status = vslsConvExec1D(task, h, inch, x, incx, y, incy);
    vslConvDeleteTask(&task);
    return status;
}
```

**例 10-6 シングルスレッド化計算用関数 sconfl**

---

```
#include "mkl_vsl.h"

int sconfl(
    int init,
    float h[], int inclh,
    float x[], int inclx, int inc2x,
    float y[], int incly, int inc2y,
    int nh, int nx, int m, int iy0, int ny,
    void* aux1, int naux1, void* aux2, int naux2)
{
    int status;
    /* assume that aux1!=0 and naux1 is big enough */
    VSLConvTaskPtr* task = (VSLConvTaskPtr*)aux1;
    if (init != 0)
        /* initialization: */
        status = vslsConvNewTaskX1D(task, VSL_CONV_MODE_FFT,
                                     nh, nx, ny, h, inclh);
    if (init == 0) {
        /* calculations: */
        int i;
        vslConvSetStart(*task, &iy0);
        for (i=0; i<m; i++) {
            float* xi = &x[inc2x * i];
            float* yi = &y[inc2y * i];
            /* task is implicitly committed at i==0 */
            status = vslsConvExecX1D(*task, xi, inclx, yi, incly);
        };
    };
    vslConvDeleteTask(task);
    return status;
}
```

---

## マルチスレッドの使用

前述の `sconf1` のような関数では、循環よりも並列計算のほうが適している。 $m > 1$  の場合、マルチスレッドを使用して異なるデータ・シーケンスに対してタスクの実行を呼び出すことができる。この場合、計算する前にタスク・コピー・ルーチンを使用してタスク・オブジェクトのコピーを  $m$  個作成し、これらのコピーを異なるスレッドで実行する。コピーする前に、([タスクエディタ](#)を使用して) タスクに必要なパラメータ調整がすべて行われていることを確認する。

コード例を次に示す。

```
if (init == 0) {
    int i, status, ss[M];
    VSLConvTaskPtr tasks[M];
    /* assume that M is big enough */
    ...
    vslConvSetStart(*task, &iy0);
    ...
    for (i=0; i<m; i++)
        /* implicit commitment at i==0 */
        vslConvCopyTask(&tasks[i], *task);
    ...
}
```

ここで、タスクの異なるコピーを実行するために、 $m$  個のスレッドが開始される。

```
...
    float* xi = &x[inc2x * i];
    float* yi = &y[inc2y * i];
    ss[i]=vslsConvExecX1D(tasks[i], xi, inclx, yi, incly);
    ...
}
```

最後に (すべてのスレッドが計算を完了した後で)、全体のステータスがすべてのタスク・オブジェクトから回収される。次のコードは最初に検出されたエラー (存在する場合) を知らせる。

```
...
for (i=0; i<m; i++) {
    status = ss[i];
    if (status != 0) /* 0 means "OK" */
        break;
}
```

```
};
return status;
}; /* end if init==0 */
```

実行ルーチンはタスク内部ステート（タスク構造のフィールド）を変更する。異なるスレッドが同じタスク・オブジェクトを処理する場合、そのような変更は互いに矛盾を引き起こすことがある。このため、異なるスレッドはタスクの異なるコピーを使用しなければならない。

## 数学表記と定義

次の表記は、本文で使用されている基本となる数学定義を説明するのに必要である。

$\mathbf{R} = (-\infty, +\infty)$  実数の集合

$\mathbf{Z} = \{0, \pm 1, \pm 2, \dots\}$  整数の集合

$\mathbf{Z}^N = \mathbf{Z} \times \dots \times \mathbf{Z}$   $N$ 次元の整数の級数のセット。

$p = (p_1, \dots, p_N) \in \mathbf{Z}^N$   $N$ 次元の整数の級数。

$u: \mathbf{Z}^N \rightarrow \mathbf{R}$   $\mathbf{Z}^N$ からの引数と  $\mathbf{R}$ からの値を持つ関数  $u$ 。

$u(p) = u(p_1, \dots, p_N)$  引数  $p = (p_1, \dots, p_N)$  に対する関数  $u$  の値。

$w = u * v$  関数  $w$  は、関数  $u$ 、 $v$  の畳み込み。

$w = u \bullet v$  関数  $w$  は、関数  $u$ 、 $v$  の相関。

級数  $p, q \in \mathbf{Z}^N$  が与えられたとき：

- 級数  $r = p + q$  は各  $n = 1, \dots, N$  に対して  $r_n = p_n + q_n$  と定義される。
- 級数  $r = p - q$  は各  $n = 1, \dots, N$  に対して  $r_n = p_n - q_n$  と定義される。
- 級数  $r = \sup\{p, q\}$  は各  $n = 1, \dots, N$  に対して  $r_n = \max\{p_n, q_n\}$  と定義される。
- 級数  $r = \inf\{p, q\}$  は各  $n = 1, \dots, N$  に対して  $r_n = \min\{p_n, q_n\}$  と定義される。
- 不等式  $p \leq q$  は各  $n = 1, \dots, N$  に対して  $p_n \leq q_n$  を示す。

次の条件を満たす級数  $p^{\min}, p^{\max} \in \mathbf{Z}^N$  が存在する場合、関数  $u(p)$  は有限関数と呼ばれる。

$u(p) \neq 0$  は  $p^{\min} \leq p \leq p^{\max}$  を意味する。



畳み込み / 相関演算は有限関数に対してのみ定義される。

次の条件を満たす関数  $u$ 、 $v$  と級数  $P^{\min}, P^{\max}, Q^{\min}, Q^{\max} \in \mathbb{Z}^N$  について考える。

$$u(p) \neq 0 \text{ は } P^{\min} \leq p \leq P^{\max}$$

$$v(q) \neq 0 \text{ は } Q^{\min} \leq q \leq Q^{\max}$$

関数  $u$  と  $v$  の線形畳み込み / 線形相関の定義は以下のとおりである。

### 線形畳み込み

関数  $w = u * v$  が  $u$  と  $v$  の畳み込みの場合：

$$w(r) \neq 0 \text{ は } R^{\min} \leq r \leq R^{\max}$$

$$R^{\min} = P^{\min} + Q^{\min} \text{ および } R^{\max} = P^{\max} + Q^{\max}$$

$R^{\min} \leq r \leq R^{\max}$  の場合：

$w(r) = \sum u(t) \cdot v(r-t)$  は、次の条件を満たすすべての  $t \in \mathbb{Z}^N$  の合計。

$$T^{\min} \leq t \leq T^{\max}$$

$$T^{\min} = \sup\{P^{\min}, r - Q^{\max}\} \text{ および } T^{\max} = \inf\{P^{\max}, r - Q^{\min}\}$$

### 線形相関

関数  $w = u \bullet v$  が  $u$  と  $v$  の相関の場合：

$$w(r) \neq 0 \text{ は } R^{\min} \leq r \leq R^{\max}$$

$$R^{\min} = Q^{\min} - P^{\max} \text{ および } R^{\max} = Q^{\max} - P^{\min}$$

$R^{\min} \leq r \leq R^{\max}$  の場合：

$w(r) = \sum u(t) \cdot v(r+t)$  は、次の条件を満たすすべての  $t \in \mathbb{Z}^N$  の合計。

$$T^{\min} \leq t \leq T^{\max}$$

$$T^{\min} = \sup\{P^{\min}, Q^{\min} - r\} \text{ および } T^{\max} = \inf\{P^{\max}, Q^{\max} - r\}$$

インテル・マス・カーネル・ライブラリの畳み込み / 相関関数の入力 / 出力データとしての関数  $u$ 、 $v$ 、 $w$  の表現は、次の[データの割り当て](#)のセクションで説明する。

## データの割り当て

このセクションでは次の関係について説明する。

- [数学表記と定義](#) のセクションの数学有限関数  $u$ 、 $v$ 、 $w$ 。

- 関数  $u$ 、 $v$ 、 $w$  を表す多次元入力 / 出力データベクトル。
- コンピュータ・メモリに入力 / 出力データベクトルを格納するのに使用する配列  $x$ 、 $y$ 、 $z$ 。

入力 / 出力データの割り当てを決定する量み込み / 相関ルーチン・パラメータは次のとおりである。

- データ配列  $x$ 、 $y$ 、 $z$
- 形状配列  $xshape$ 、 $yshape$ 、 $zshape$
- 配列内のストライド  $xstride$ 、 $ystride$ 、 $zstride$
- パラメータ  $start$ 、 $decimation$

## 有限関数とデータベクトル

前述の有限関数  $u(p)$ 、 $v(q)$ 、 $w(r)$  は、入力 / 出力データの多次元ベクトルとして表される。

$inputu(i_1, \dots, i_{dims})$  ( $u(p_1, \dots, p_N)$  の場合)

$inputv(j_1, \dots, j_{dims})$  ( $v(q_1, \dots, q_N)$  の場合)

$output(k_1, \dots, k_{dims})$  ( $w(r_1, \dots, r_N)$  の場合)

パラメータ  $dims$  は次元を表し、 $N$  に等しい。

パラメータ  $xshape$ 、 $yshape$ 、 $zshape$  は入力 / 出力ベクトルの形状を定義する。

各  $n = 1, \dots, dims$  に対して  $1 \leq i_n \leq xshape(n)$  の場合、 $inputu(i_1, \dots, i_{dims})$  は定義される。

各  $n = 1, \dots, dims$  に対して  $1 \leq j_n \leq yshape(n)$  の場合、 $inputv(j_1, \dots, j_{dims})$  は定義される。

各  $n = 1, \dots, dims$  に対して  $1 \leq k_n \leq zshape(n)$  の場合、 $output(k_1, \dots, k_{dims})$  は定義される。

入力ベクトルと関数  $u$ 、 $v$  の関係は次の式によって定義される。

$inputu(i_1, \dots, i_{dims}) = u(p_1, \dots, p_N)$ 、各  $n$  に対して  $p_n = p_n^{\min} + (i_n - 1)$ 。

$inputv(j_1, \dots, j_{dims}) = v(q_1, \dots, q_N)$ 、各  $n$  に対して  $q_n = q_n^{\min} + (j_n - 1)$ 。

出力ベクトルと関数  $w(r)$  の関係も似ている (ただし、パラメータ  $start$  と  $decimation$  が定義されていない場合のみ)。

$\text{output}(k_1, \dots, k_{\text{dims}}) = w(r_1, \dots, r_N)$ 、各  $n$  に対して  $r_n = R_n^{\min} + (k_n - 1)$ 。

パラメータ  $\text{start}$  が定義されている場合は、次の区間内でなければならない。

$$R_n^{\min} \leq \text{start}(n) \leq R_n^{\max}$$

定義されている場合、 $\text{start}$  パラメータは次の式で  $R^{\min}$  を置き換える。

$$\text{output}(k_1, \dots, k_{\text{dims}}) = w(r_1, \dots, r_N), \text{ ここで } r_n = \text{start}(n) + (k_n - 1)$$

パラメータ  $\text{decimation}$  が定義されている場合は、次の式に従って関係を変更する。

$$\text{output}(k_1, \dots, k_{\text{dims}}) = w(r_1, \dots, r_N), \text{ ここで } r_n = R_n^{\min} + (k_n - 1) \cdot \text{decimation}(n)$$

パラメータ  $\text{start}$  と  $\text{decimation}$  の両方が定義されている場合の式は次のとおりである。

$$\text{output}(k_1, \dots, k_{\text{dims}}) = w(r_1, \dots, r_N), \text{ ここで } r_n = \text{start}(n) + (k_n - 1) \cdot \text{decimation}(n)$$

畳み込み / 相関ソフトウェアは、タスク遂行時に  $\text{zshape}$ 、 $\text{start}$ 、 $\text{decimation}$  の値をチェックする。 $k_n, n=1, \dots, \text{dims}$ 、に対して  $r_n$  が  $R_n^{\max}$  を超えると、エラーが発生する。

### データベクトルの割り当て

パラメータ配列  $x$  と  $y$  はメモリに入力データベクトルを含み、配列  $z$  は出力データベクトルを格納する。メモリにアクセスするのに、畳み込み / 相関ソフトウェアはこれらの配列への唯一のポインタを使用し、配列の形状は無視する。

パラメータ  $x$ 、 $y$ 、 $z$  に対して、これらの配列の実際の長さがデータベクトルを格納するのに十分であることを条件とする 1 次元の配列を与えることができる。

配列  $x$ 、 $y$ 、 $z$  内に入力 / 出力データの割り当てについて以下に示す。ここで配列は 1 次元であると仮定する。多次元インデックス  $i, j, k \in \mathbf{Z}^N$  が与えられると、1 次元インデックス  $e, f, g \in \mathbf{Z}$  は次のように定義される。

$\text{inputu}(i_1, \dots, i_{\text{dims}})$  は  $x(e)$  で割り当てられる。

$\text{inputv}(j_1, \dots, j_{\text{dims}})$  は  $y(f)$  で割り当てられる。

$\text{output}(k_1, \dots, k_{\text{dims}})$  は  $z(g)$  で割り当てられる。

インデックス  $e$ 、 $f$ 、 $g$  は次のように定義される。

$$e = 1 + \sum x_{\text{stride}}(n) \cdot dx(n) \quad (\text{すべての } n = 1, \dots, \text{dims} \text{ に対する合計})$$

$$f = 1 + \sum y_{\text{stride}}(n) \cdot dy(n) \quad (\text{すべての } n = 1, \dots, \text{dims} \text{ に対する合計})$$

$$g = 1 + \sum z_{\text{stride}}(n) \cdot dz(n) \quad (\text{すべての } n = 1, \dots, \text{dims} \text{ に対する合計})$$

距離  $dx(n)$ 、 $dy(n)$ 、 $dz(n)$  はストライドの符号に依存する。

$xstride(n) > 0$  の場合は  $dx(n) = i_n - 1$ 、 $xstride(n) < 0$  の場合は  $dx(n) = i_n - xshape(n)$ 。

$ystride(n) > 0$  の場合は  $dy(n) = j_n - 1$ 、 $ystride(n) < 0$  の場合は  $dy(n) = j_n - yshape(n)$ 。

$zstride(n) > 0$  の場合は  $dz(n) = k_n - 1$ 、 $zstride(n) < 0$  の場合は  $dz(n) = k_n - zshape(n)$ 。

インデックス  $e$ 、 $f$ 、 $g$  の定義は、配列  $x$ 、 $y$ 、 $z$  のインデックスが 1 から開始されることを前提としている。

$x(e)$  は  $e = 1, \dots, \text{length}(x)$  に対して定義される。

$y(f)$  は  $f = 1, \dots, \text{length}(y)$  に対して定義される。

$z(g)$  は  $g = 1, \dots, \text{length}(z)$  に対して定義される。

1 次元または 2 次元の場合、多次元出力ベクトルの成分が配列  $z$  にどのように格納されるのかを以下に説明する。

**1 次元の場合：**  $dims = 1$  の場合、 $zshape$  は配列  $z$  に格納される出力値の個数。配列  $z$  の実際の長さは  $zshape$  個の成分よりも大きいことがある。

$zstride > 1$  の場合、出力値はストライドと共に格納される。 $output(1)$  は  $z(1)$  に、 $output(2)$  は  $z(1+zstride)$  に格納される。そのため、 $z$  の実際の長さは  $1+zstride*(zshape-1)$  個の成分以上でなければならない。

$zstride < 0$  の場合も配列  $z$  の成分間のストライドを定義する。ただし、使用される成分の次数は逆。 $k$  番目の出力値では、 $output(k)$  は  $z(1+|zstride|*(zshape-k))$  に格納される。 $|zstride|$  は  $zstride$  の絶対値。配列  $z$  の実際の長さは、 $1+|zstride|*(zshape - 1)$  個の成分以上でなければならない。

**2 次元の場合：**  $dims = 2$  の場合、出力データは 2 次元の行列である。 $zstride(1)$  の値は、行列の列内のストライド、つまりインデックス  $k_1$ 、 $k_2$  それぞれの組み合わせに対する  $output(k_1, k_2)$  と  $output(k_1+1, k_2)$  の間のストライドを定義する。一方、 $zstride(2)$  は列間のストライド、つまり  $output(k_1, k_2)$  と  $output(k_1, k_2+1)$  の間のストライドを定義する。

$zstride(2)$  が  $zshape(1)$  よりも大きい場合、列のスパース割り当てを引き起こす。 $zstride(2)$  の値が  $zshape(1)$  よりも小さい場合、出力行列が転置されることがある。例えば  $zshape = (2, 3)$  の場合、形状が  $3 \times 2$  の転置行列のような出力値を割り当てるために  $zstride = (3, 1)$  を定義することができる。

`zstride` がこのような転置を前提としているかどうかに関わらず、異なる成分 `output(k1, ..., kdims)` が異なる場所  $z(g)$  に格納されることを確認する必要がある。



# フーリエ変換関数

# 11

本章では、インテル<sup>®</sup> マス・カーネル・ライブラリ (インテル<sup>®</sup> MKL) で利用可能な、次の離散フーリエ変換関数について説明する。

- シングル・プロセッサまたは共有メモリ・システムにおける離散フーリエ変換 (DFT) 関数 (以下の「[DFT 関数](#)」を参照)。
- 分散メモリ・アーキテクチャにおける「[クラスタ DFT 関数](#)」 (Intel<sup>®</sup> クラスタ MKL 製品でのみ利用可能)。

これらの DFT 関数は、使い勝手のよいアプリケーション・プログラム・インターフェイスとして提供され、高速フーリエ変換 (FFT) アルゴリズムによって高速に DFT 計算を実行する。



**注：**FFT を必要とするすべてのアプリケーションは、これらの DFT 関数のセットを使用すべきである。これらのルーチンは、高性能で広範の機能性を提供する。

以前のバージョンのライブラリとの互換性のため、インテル MKL は本章で述べているように (「[高速フーリエ変換 \(非推奨\)](#)」を参照)、従来の FFT インターフェイスを引き続きサポートしているが、従来のユーザ・アプリケーションは、性能と機能の両方の面から DFT 新関数に移行するのが望ましい。

## DFT 関数

インテル MKL の離散フーリエ変換関数ライブラリは、1 次元、2 次元、多次元 (最大 7 次元) のルーチン、およびすべての変換関数で Fortran インターフェイスと C インターフェイスの両方を提供する。

DFT 関数は、従来の FFT ルーチンと異なり、2 の累乗以外の変換長での混合基数変換をサポートする。

インテル MKL に実装されているすべての DFT 関数を次の表に示す。

**表 11-1      インテル MKL の DFT 関数**

関数名	演算
<b>ディスクリプタ操作関数</b>	
<a href="#">DftiCreateDescriptor</a>	ディスクリプタ・データ構造にメモリを割り当て、デフォルト構成設定で具体化する。
<a href="#">DftiCommitDescriptor</a>	実際の DFT 演算を実行可能にするすべての初期化を実行する。
<a href="#">DftiCopyDescriptor</a>	既存のディスクリプタをコピーする。
<a href="#">DftiFreeDescriptor</a>	ディスクリプタに割り当てられていたメモリを解放する。
<b>DFT 計算関数</b>	
<a href="#">DftiComputeForward</a>	順方向 DFT を計算する。
<a href="#">DftiComputeBackward</a>	逆方向 DFT を計算する。
<b>ディスクリプタ構成関数</b>	
<a href="#">DftiSetValue</a>	特定の 1 個の構成パラメータを指定された構成値で設定する。
<a href="#">DftiGetValue</a>	特定の 1 個の構成パラメータの構成値を取得する。
<b>ステータス確認関数</b>	
<a href="#">DftiErrorClass</a>	ステータスが定義済みクラスのエラーを反映しているか確認する。
<a href="#">DftiErrorMessage</a>	エラー・メッセージを生成する。

DFT 関数の説明に続いて、構成設定 (「[構成設定](#)」を参照) および使用されるさまざまな構成パラメータを説明する。



## DFT の計算

本章で後述する DFT 関数は Fortran インターフェイスと C インターフェイスで実装される。Fortran とは Fortran 95 を意味する。DFT インターフェイスは、Fortran 77 にはない Fortran 95 のさまざまな新機能に強く依存している。



**注：**Fortran では明示的な関数インターフェイスに続くデータ配列は、どの変換タイプに対しても 1 次元として定義されなければならない。

本章で示される資料では、C のネイティブ複素タイプの有効性は C9X で規定されているものと仮定している。

DFT インターフェイスを使用して変換結果を計算するコードの例については、付録の「[DFT コードの例](#)」のセクションを参照のこと。

一般的な状況では 1 回の DFT 計算は 4 個の関数呼び出しで成立すると考えられる。インテル MKL では、1 つの単一データ構造、すなわちディスクリプタを使用して、パラメータを個別に変更できるようにした自由度の高い手法を DFT 計算に適用している。これによって機能の拡張性と使いやすさを実現した。

生成された DFTI\_DESCRIPTOR のレコードタイプは計算される DFT の長さや領域の情報だけでなく、構成パラメータの多くの設定も含んでいる。これらすべてのパラメータに対してデフォルト設定を使用しており、例えば次のような項目である。

- 計算される DFT は倍率因子を持っていない
- 変換されるデータは 1 セットのみ
- データはメモリに連続して格納
- 計算結果は入力データに上書き (インプレース) される、ほか

これらのデフォルト設定が適当でない場合は、[例 C-18](#) と [例 C-19](#) で示すように、[DftiSetValue](#) 関数を用いて 1 つずつ変更できる。

## DFT インターフェイス

DFT 関数を使用するには、Fortran では "use" 構文を使用して MKL\_DFTI モジュールにアクセスする必要がある。C では "include" 構文を使用してヘッダファイル mkl\_dfti.h にアクセスする必要がある。

Fortran インターフェイスでは、派生型 `DFTI_DESCRIPTOR`、さまざまな構成パラメータの名称とそれらの取り得る値を表す名前付き定数、Fortran 95 の汎用機能性を介したさまざまなオーバーロード関数が利用できる。

C インターフェイスでは、構造型 `DFTI_DESCRIPTOR`、マクロ定義

```
#define DFTI_DESCRIPTOR_HANDLE DFTI_DESCRIPTOR、
```

`DFTI_CONFIG_PARAM` と `DFTI_CONFIG_VALUE` の 2 個の列挙型の名前付き定数、関数によっては異なる個数の入力引数を受け付ける種々の関数が与えられる。



**注：**現時点のライブラリ実装では、本章の続くセクションで記載されている関数あるいは機能の一部がサポートされていない場合がある。実装に依存して除外される項目については、使用しているライブラリ・バージョンのリリースノートを参照のこと。

インテル MKL では DFT 関数を 4 種類の大きなカテゴリに分けている。

1. **ディスクリプタ操作。** このカテゴリは 4 つの関数で構成される。1 つ目の [DftiCreateDescriptor](#) は DFT ディスクリプタを生成しストレージを動的に割り当てる関数である。この関数はユーザが与える入力値によってディスクリプタをデフォルト設定に構成する。  
2 つ目の [DftiCommitDescriptor](#) はディスクリプタを設定に基づいて「遂行」させる関数である。実際には、すべての必要な事前計算が実行されることを意味する。この事前計算には、入力長さの因子分解と必要なすべての回転因子の計算が含まれる。3 つ目の [DftiCopyDescriptor](#) はディスクリプタのコピーを作成する関数、4 つ目の [DftiFreeDescriptor](#) はディスクリプタ情報に割り当てられていたすべてのメモリを解放する関数である。
2. **DFT 計算。** このカテゴリは 2 つの関数で構成される。1 つ目の [DftiComputeForward](#) は順方向 DFT 計算を実行する関数で、2 つ目の [DftiComputeBackward](#) は逆方向 DFT 計算を実行する関数である。
3. **ディスクリプタ構成。** このカテゴリは 2 つの関数で構成される。1 つ目の [DftiSetValue](#) 関数は特定の 1 個の値を、多くの構成パラメータのうち変更可能な（わずかにあるか、まったくない）パラメータの 1 つに設定する。2 つ目の

[DftiGetValue](#) 関数は、これら構成パラメータの任意の 1 つの現在値を取得する (すべてが読み出し可能)。読み出し可能なパラメータ数が多いが、扱えるのは 1 回の関数呼び出しにつき 1 つである。

4. **ステータス確認。** 上記の 3 種類のカテゴリに記述されている各関数は演算のステータスを表す整数値を返す。  
戻り値が非ゼロの場合にはある種の問題が起きたことを意味する。インテル MKL では将来のリリースにおける拡張を想定して、現在の DFT インターフェイスは単一の論理ステータス・クラス関数 [DftiErrorClass](#) および単純なステータス・メッセージ生成関数 [DftiErrorMessage](#) のみを提供している。

## ステータス確認関数

ディスクリプタ操作、DFT 計算、ディスクリプタ構成のカテゴリに含まれる各関数は、演算のステータスを示す整数値を返す。ここで述べる 2 つの関数は、このステータスを確認する関数である。1 つ目の関数は論理関数で定義済みクラスのエラーをステータスが反映しているか確認し、2 つ目の関数はエラー・メッセージ関数で文字列を返す。

---

## ErrorClass

ステータスが定義済みクラスのエラーを反映しているか確認する。

---

### 構文

```
!Fortran
Predicate = DftiErrorClass( Status, Error_Class )
/* C */
predicate = DftiErrorClass( status, error_class );
```

## 説明

インテル MKL の DFT インターフェイスには[表 11-2](#) に示す定義済みエラークラスが与えられている。これらは名前付き定数で、Fortran では INTEGER 型、C では long 型である。

**表 11-2 定義済みエラークラス**

名前付き定数	意味
DFTI_NO_ERROR	エラーなし
DFTI_MEMORY_ERROR	通常、メモリ割り当てに関係する
DFTI_INVALID_CONFIGURATION	構成パラメータの 1 個以上が無効な設定
DFTI_INCONSISTENT_CONFIGURATION	構成パラメータまたは入力パラメータが不整合
DFTI_NUMBER_OF_THREADS_ERROR	計算関数の OMP スレッド数は、( 遂行関数の ) 初期化段階の OMP スレッド数と等しくない。
DFTI_MULTITHREADED_ERROR	通常、OMP ルーチンのエラー戻り値に関係する
DFTI_BAD_DESCRIPTOR	ディスクリプタを計算に使用できない
DFTI_UNIMPLEMENTED	正当な設定が未実装、実装依存
DFTI_MKL_INTERNAL_ERROR	ライブラリ内部エラー

定義済みエラークラスの確認では、指定したエラークラスに対して DftiErrorClass を適用して、その結果ステータス戻り値が .TRUE. または .FALSE. かを確認することが正しい使用方法である。ステータスと定義済みクラスの直接比較は使用方法としては適切ではない。ステータス確認関数の正しい使用方法是[例 C-20](#) を参照。

## インターフェイスとプロトタイプ

```
//Fortran interface
INTERFACE DftiErrorClass
//Note that the body provided here is to illustrate the different
//argument list and types of dummy arguments.The interface
//does not guarantee what the actual function names are.
//Users can only rely on the function name following the
//keyword INTERFACE
FUNCTION some_actual_function_8( Status, Error_Class )
    LOGICAL some_actual_function_8
    INTEGER, INTENT(IN) :: Status, Error_Class
END FUNCTION some_actual_function_8
```

```
END INTERFACE DftiErrorClass

/* C prototype */
long DftiErrorClass( long , long );
```

---

## ErrorMessage

エラー・メッセージを生成する。

---

### 構文

```
!Fortran
ERROR_MESSAGE = DftiErrorMessage( Status )
/* C */
error_message = DftiErrorMessage( status );
```

### 説明

このエラー・メッセージ関数はエラー・メッセージ文字列を生成する。文字列の最大長は Fortran では名前付き定数 DFTI\_MAX\_MESSAGE\_LENGTH で与えられる。実際のエラー・メッセージは実装依存である。Fortran の場合、ユーザは DFTI\_MAX\_MESSAGE\_LENGTH を最大長とした文字列を使用する。C ではこの関数は文字列へのポインタを返す。すなわちデリミタ '0' を持つ文字配列である。

[例 C-20](#) に、この関数の実装例を示す。

### インターフェイスとプロトタイプ

```
//Fortran interface
INTERFACE DftiErrorMessage
//Note that the body provided here is to illustrate the different
//argument list and types of dummy arguments.The interface
//does not guarantee what the actual function names are.
//Users can only rely on the function name following the
//keyword INTERFACE
FUNCTION some_actual_function_9( Status, Error_Class )
CHARACTER(LEN=DFTI_MAX_MESSAGE_LENGTH) some_actual_function_9( Status )
```

```
INTEGER, INTENT(IN) :: Status
END FUNCTION some_actual_function_9
END INTERFACE DftiErrorMessage

/* C prototype */
char *DftiErrorMessage( long );
```

## ディスクリプタ操作

このカテゴリは、ディスクリプタの生成、ディスクリプタの遂行、ディスクリプタのコピー、ディスクリプタの解放、の4つの関数で構成される。

---

## CreateDescriptor

ディスクリプタ・データ構造にメモリを割り当て、デフォルト構成設定で具体化する。

---

### 構文

```
!Fortran
      Status = DftiCreateDescriptor( Desc_Handle,      &
                                     Precision,         &
                                     Forward_Domain,    &
                                     Dimension,         &
                                     Length )

/* C */
status = DftiCreateDescriptor( &desc_handle,
                               precision,
                               forward_domain,
                               dimension,
                               length );
```

## 説明

この関数はディスクリプタ・データ構造にメモリを割り当て、変換対象の精度、領域、次元、長さをそれぞれのデフォルト構成設定を使って具体化する。領域は順方向変換での領域として解釈される。メモリは動的に割り当てられるため、出力結果として生成されたディスクリプタのポインタが得られる。この関数は、DFT を計算する従来型の多くのソフトウェア・パッケージやライブラリに見られる「初期化」ルーチンとはやや異なっている。ユーザの要求によって値設定関数 [DftiSetValue](#) を通してデフォルト構成設定を変更できることから、この関数は回転因子計算のような計算量の多い計算はおそらく実行しない。

精度と(順方向)領域は DFT インターフェイスで構成値に与えられる名前付き定数によって指定される。精度の選択肢は DFTI\_SINGLE か DFTI\_DOUBLE である。(順方向)領域の選択肢は DFTI\_COMPLEX または DFTI\_REAL である。[表 11-5](#) に構成値に対する名前付き定数の全リストを示す。

次元は変換の次元を表す単純な正の整数である。長さは、1 次元変換では単純な正の整数、多次元変換では長さ次元に対応した正の整数を含む整数配列 (C ではポインタ) である。

この関数は正常に終了すると DFTI\_NO\_ERROR を返す。戻り値ステータスの詳細は「[ステータス確認関数](#)」を参照のこと。

## インターフェイスとプロトタイプ

!Fortran interface.

INTERFACE DftiCreateDescriptor

!Note that the body provided here is to illustrate the different

!argument list and types of dummy arguments.The interface

!does not guarantee what the actual function names are.

!Users can only rely on the function name following the keyword INTERFACE

FUNCTION some\_actual\_function\_1D( Desc\_Handle, Prec, Dom, Dim, Length )

INTEGER :: some\_actual\_function\_1D

TYPE(DFTI\_DESCRIPTOR), POINTER :: Desc\_Handle

INTEGER, INTENT(IN) :: Prec, Dom

INTEGER, INTENT(IN) :: Dim, Length

END FUNCTION some\_actual\_function\_1D

FUNCTION some\_actual\_function\_HIGHD( Desc\_Handle, Prec, Dom, Dim, Length )

INTEGER :: some\_actual\_function\_HIGHD

```
TYPE(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
INTEGER, INTENT(IN) :: Prec, Dom
INTEGER, INTENT(IN) :: Dim, Length(*)
END FUNCTION some_actual_function_HIGHD
END INTERFACE DftiCreateDescriptor
```

関数はオーバーロードであるとともに、長さの実際の引数はスカラまたは階数 1 配列を取り得る点に注意する。

```
/* C prototype */
long DftiCreateDescriptor( DFTI_DESCRIPTOR_HANDLE *,
                           DFTI_CONFIG_PARAM ,
                           DFTI_CONFIG_PARAM ,
                           long ,
                           ... );
```

可変引数機能は、スカラ (long) または配列 (long \*) で表現される長さの引数に対処するために使用される。

---

## CommitDescriptor

実際の DFT 演算を実行可能にするすべての初期化を実行する。

---

### 構文

```
!Fortran
Status = DftiCommitDescriptor( Desc_Handle )
/* C */
status = DftiCommitDescriptor( desc_handle );
```

### 説明

DFT インターフェイスでは、ディスクリプタを使って DFT 計算を実行する前に、生成されたディスクリプタを遂行する関数を呼び出す必要がある。通常この遂行処理では、実際の DFT 計算を実行可能にするすべての初期化が実行される。近代的な実装では、高度に効率化された計算方法を検索するために、さまざまな長さの入力を用いた因子分解を行う調査が含まれる場合がある。



遂行されたディスクリプタの構成パラメータを値設定関数 (「[ディスクリプタの構成](#)」を参照) を介して変更した場合は、計算関数を呼び出す前にディスクリプタの再遂行を必要とする。通常は、この遂行関数コールのすぐあとに計算関数コールを続ける (「[DFT計算](#)」を参照)。

この関数は正常に終了すると DFTI\_NO\_ERROR を返す。戻り値ステータスの詳細は「[ステータス確認関数](#)」を参照のこと。

## インターフェイスとプロトタイプ

```
!Fortran interface
INTERFACE DftiCommitDescriptor
!Note that the body provided here is to illustrate the different
!argument list and types of dummy arguments.The interface
!does not guarantee what the actual function names are.
!Users can only rely on the function name following the
!keyword INTERFACE
    FUNCTION some_actual function_1 ( Desc_Handle )
        INTEGER :: some_actual function_1
        TYPE(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
    END FUNCTION some_actual function_1
END INTERFACE DftiCommitDescriptor

/* C prototype */
long DftiCommitDescriptor( DFTI_DESCRIPTOR_HANDLE );
```

---

## CopyDescriptor

既存のディスクリプタをコピーする。

---

### 構文

```
!Fortran
Status = DftiCopyDescriptor( Desc_Handle_Original,
                             Desc_Handle_Copy )

/* C */
```

```
status = DftiCopyDescriptor( desc_handle_original,  
                             &desc_handle_copy );
```

## 説明

この関数は既存ディスクリプタのコピーを作成し、それに対するポインタを与える。この関数の目的は、ディスクリプタ解放関数 `DftiFreeDescriptor` によって元のディスクリプタが破壊された場合でも、元のディスクリプタの全情報を維持することである。

この関数は正常に終了すると `DFTI_NO_ERROR` を返す。戻り値ステータスの詳細は「[ステータス確認関数](#)」を参照のこと。

## インターフェイスとプロトタイプ

```
!Fortran interface  
INTERFACE DftiCopyDescriptor  
!Note that the body provided here is to illustrate the different  
!argument list and types of dummy arguments.The interface  
!does not guarantee what the actual function names are.  
!Users can only rely on the function name following the  
!keyword INTERFACE  
    FUNCTION some_actual_function_2( Desc_Handle_Original,  
                                     Desc_Handle_Copy )  
  
        INTEGER :: some_actual_function_2  
        TYPE(DFTI_DESCRIPTOR), POINTER :: Desc_Handle_Original, Desc_Handle_Copy  
    END FUNCTION some_actual_function_2  
END INTERFACE DftiCopyDescriptor  
  
/* C prototype */  
long DftiCopyDescriptor( DFTI_DESCRIPTOR_HANDLE, DFTI_DESCRIPTOR_HANDLE * );
```

---

## FreeDescriptor

ディスクリプタに割り当てられていたメモリを解放する。

---

### 構文

```
!Fortran
Status = DftiFreeDescriptor( Desc_Handle )
/* C */
status = DftiFreeDescriptor( &desc_handle );
```

### 説明

この関数はディスクリプタに割り当てられているすべてのメモリを解放する。

この関数は正常に終了すると DFTI\_NO\_ERROR を返す。戻り値ステータスの詳細は「[ステータス確認関数](#)」を参照のこと。

### インターフェイスとプロトタイプ

```
!Fortran interface
INTERFACE DftiFreeDescriptor
//Note that the body provided here is to illustrate the different
//argument list and types of dummy arguments.The interface
//does not guarantee what the actual function names are.
//Users can only rely on the function name following the
//keyword INTERFACE
    FUNCTION some_actual_function_3( Desc_Handle )
        INTEGER :: some_actual_function_3
        TYPE(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
    END FUNCTION some_actual_function_3
END INTERFACE DftiFreeDescriptor

/* C prototype */
long DftiFreeDescriptor( DFTI_DESCRIPTOR_HANDLE * );
```

## DFT 計算

このカテゴリは、順方向変換の計算、逆方向変換の計算の 2 つの関数で構成される。

---

## ComputeForward

順方向 DFT を計算する。

---

### 構文

```
!Fortran
Status = DftiComputeForward( Desc_Handle, X_inout )
Status = DftiComputeForward( Desc_Handle, X_in, X_out )
/* C */
status = DftiComputeForward( desc_handle, x_inout );
status = DftiComputeForward( desc_handle, x_in, x_out );
```

### 説明

実際の DFT 計算はディスクリプタの生成と遂行を完了した後に実行する。  
DftiComputeForward 関数は順方向 DFT を計算する。これは、因子  $e^{-i2\pi/n}$  を使用した変換である。構成の自由度が高いため、入力データを複数の方法で表すことができ、また出力結果を複数の方法で配置できる。そのため、入力パラメータの個数とタイプは多様である。このような多様性は Fortran 95 の一般関数機能によって実現される。データと結果パラメータは、すべて擬寸法階数 1 の配列 DIMENSION(0:\*) として宣言される。

この関数は正常に終了すると DFTI\_NO\_ERROR を返す。戻り値ステータスの詳細は

「[ステータス確認関数](#)」を参照のこと。

### インターフェイスとプロトタイプ

```
//Fortran interface.
INTERFACE DftiComputeFoward
//Note that the body provided here is to illustrate the different
//argument list and types of dummy arguments.The interface
//does not guarantee what the actual function names are.
```

```
//Users can only rely on the function name following the
//keyword INTERFACE
// One argument single precision complex
FUNCTION some_actual_function_4_C( Desc_Handle, X )
  INTEGER :: some_actual_function_4_C
  TYPE(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
  COMPLEX, INTENT(INOUT) :: X(*)
END FUNCTION some_actual_function_4_C
// One argument double precision complex
FUNCTION some_actual_function_4_Z( Desc_Handle, X )
  INTEGER :: some_actual_function_4_Z
  TYPE(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
  COMPLEX (Kind((0D0,0D0))), INTENT(INOUT) :: X(*)
END FUNCTION some_actual_function_4_Z
// One argument single precision real
FUNCTION some_actual_function_4_R( Desc_Handle, X )
  INTEGER :: some_actual_function_4_R
  TYPE(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
  REAL, INTENT(INOUT) :: X(*)
END FUNCTION some_actual_function_4_R
// One argument double precision real
...
// Two argument single precision complex
...
...
FUNCTION some_actual_function_4_CC( Desc_Handle, X_In, Y_Out )
  INTEGER :: some_actual_function_4_CC
  TYPE(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
  COMPLEX, INTENT(IN) :: X_In
  COMPLEX, INTENT(OUT) :: Y_Out(*)
END FUNCTION some_actual_function_4_CC
END INTERFACE DftiComputeFoward
```

```
/* C prototype */
long DftiComputeForward( DFTI_DESCRIPTOR_HANDLE,
                        void *,
                        ... );
```

DFT インターフェイスはその実装で、規則的な「ストライド」パターンで線形にメモリに格納されるものとしてデータが扱われることを前提としている（詳細は「[ストライド](#)」[\[3\]](#)を参照のこと）。関数は最初の成分の開始アドレスを期待する。そのため、Fortran では擬寸法宣言を使用する。

ディスクリプタには、引数の個数と存在すべきタイプを正確に決めるために必要な十分な情報が含まれている。実際の実装では、この情報を使って入力の変数を確認する場合がある。

---

## ComputeBackward

逆方向 DFT を計算する。

---

### 構文

```
!Fortran
Status = DftiComputeBackward( Desc_Handle, X_inout )
Status = DftiComputeBackward( Desc_Handle, X_in, X_out )
/* C */
status = DftiComputeBackward( desc_handle, x_inout );
status = DftiComputeBackward( desc_handle, x_in, x_out );
```

### 説明

実際の DFT 計算はディスクリプタの生成と遂行を完了した後に実行する。

DftiComputeBackward 関数は逆方向 DFT を計算する。

これは、因子  $e^{i2\pi/n}$  を使用した変換である。構成の自由度が高いため、入力データを複数の方法で表すことができ、また出力結果を複数の方法で配置できる。そのため、入力パラメータの個数とタイプは多様である。このような多様性は Fortran 95 の一般関数機能によって実現される。データと結果パラメータは、すべて擬寸法階数 1 の配列 DIMENSION(0:\*) として宣言される。

この関数は正常に終了すると DFTI\_NO\_ERROR を返す。戻り値ステータスの詳細は「[ステータス確認関数](#)」を参照のこと。

## インターフェイスとプロトタイプ

```
//Fortran interface.
INTERFACE DftiComputeBackward
//Note that the body provided here is to illustrate the different
//argument list and types of dummy arguments.The interface
//does not guarantee what the actual function names are.
//Users can only rely on the function name following the
//keyword INTERFACE
// One argument single precision complex
FUNCTION some_actual_function_5_C( Desc_Handle, X )
    INTEGER :: some_actual_function_5_C
    TYPE(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
    COMPLEX, INTENT(INOUT) :: X(*)
END FUNCTION some_actual_function_5_C
// One argument double precision complex
FUNCTION some_actual_function_5_Z( Desc_Handle, X )
    INTEGER :: some_actual_function_5_Z
    TYPE(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
    COMPLEX (Kind((0D0,0D0))), INTENT(INOUT) :: X(*)
END FUNCTION some_actual_function_5_Z
// One argument single precision real
FUNCTION some_actual_function_5_R( Desc_Handle, X )
    INTEGER :: some_actual_function_5_R
    TYPE(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
    REAL, INTENT(INOUT) :: X(*)
END FUNCTION some_actual_function_5_R
// One argument double precision real
...
// Two argument single precision complex
...
...
FUNCTION some_actual_function_5_CC( Desc_Handle, X_In, Y_Out )
    INTEGER :: some_actual_function_5_CC
```

```
TYPE(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
COMPLEX, INTENT(IN) :: X_In(*)
COMPLEX, INTENT(OUT) :: Y_Out(*)
END FUNCTION some_actual_function_5_CCEND INTERFACE DftiComputeBackward
END INTERFACE DftiComputeBackward

/* C prototype */
long DftiComputeBackward( DFTI_DESCRIPTOR_HANDLE,
                          void *,
                          ... );
```

DFT インターフェイスはその実装で、規則的な「ストライド」パターンで線形にメモリに格納されるものとしてデータが扱われることを前提としている（詳細は「[ストライド](#)」、[3] を参照のこと）。関数は最初の成分の開始アドレスを期待する。そのため、Fortran では擬寸法宣言を使用する。

ディスクリプタには、引数の個数と存在すべきタイプを正確に決めるために必要な十分な情報が含まれている。実際の実装では、この情報を使って入力の変数を確認する場合がある。



## ディスクリプタの構成

このカテゴリは、特定の 1 個の構成パラメータに適切な値を設定する値設定関数 `DftiSetValue` と、特定の 1 個の構成パラメータ値を読み出す値取得関数 `DftiGetValue` の 2 つで構成される。すべての構成パラメータは読み出し可能であるが、一部はユーザが設定できない。構成パラメータには、バージョン番号、動的な情報など、実装に固有な固定情報が含まれているが、固定情報は関数の実行時の実装によって求められる。詳細は「[構成設定](#)」を参照のこと。

---

## SetValue

特定の 1 個の構成パラメータを指定された構成値で設定する。

---

### 構文

```
!Fortran
Status = DftiSetValue( Desc_Handle, &
                      Config_Param, &
                      Config_Val )

/* C */
status = DftiSetValue( desc_handle,
                      config_param,
                      config_val );
```

### 説明

この関数は特定の 1 個の構成パラメータを指定された構成値で設定する。構成パラメータは[表 11-3](#)に示される名前付き定数のうちの 1 つであり、構成値は対応する適切なタイプ (名前付き定数またはネイティブ・タイプ) である。設定の詳細は「[構成設定](#)」を参照のこと。

この関数は正常に終了すると `DFTI_NO_ERROR` を返す。戻り値ステータスの詳細は「[ステータス確認関数](#)」を参照のこと。

## インターフェイスとプロトタイプ

```
!Fortran interface
INTERFACE DftiSetValue
//Note that the body provided here is to illustrate the different
//argument list and types of dummy arguments.The interface
//does not guarantee what the actual function names are.
//Users can only rely on the function name following the
//keyword INTERFACE
FUNCTION some_actual_function_6_INTVAL( Desc_Handle, Config_Param, INTVAL )
    INTEGER :: some_actual_function_6_INTVAL
    Type(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
    INTEGER, INTENT(IN) :: Config_Param
    INTEGER, INTENT(IN) :: INTVAL
END FUNCTION some_actual_function_6_INTVAL

FUNCTION some_actual_function_6_SGLVAL( Desc_Handle, Config_Param, SGLVAL )
    INTEGER :: some_actual_function_6_SGLVAL
    Type(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
    INTEGER, INTENT(IN) :: Config_Param
    REAL, INTENT(IN) :: SGLVAL
END FUNCTION some_actual_function_6_SGLVAL

FUNCTION some_actual_function_6_DBLVAL( Desc_Handle, Config_Param, DBLVAL )
    INTEGER :: some_actual_function_6_DBLVAL
    Type(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
    INTEGER, INTENT(IN) :: Config_Param
    REAL (KIND(0D0)), INTENT(IN) :: DBLVAL
END FUNCTION some_actual_function_6_DBLVAL

FUNCTION some_actual_function_6_INTVEC( Desc_Handle, Config_Param, INTVEC )
    INTEGER :: some_actual_function_6_INTVEC
    Type(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
    INTEGER, INTENT(IN) :: Config_Param
```

```
INTEGER, INTENT(IN) :: INTVEC(*)
END FUNCTION some_actual_function_6_INTVEC

FUNCTION some_actual_function_6_CHARS( Desc_Handle, Config_Param, CHARS )
  INTEGER :: some_actual_function_6_CHARS
  Type(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
  INTEGER, INTENT(IN) :: Config_Param
  CHARACTER(*), INTENT(IN) :: CHARS
END FUNCTION some_actual_function_6_CHARS
END INTERFACE DftiSetValue

/* C prototype */
long DftiSetValue( DFTI_DESCRIPTOR_HANDLE,
                  DFTI_CONFIG_PARAM ,
                  ... );
```

---

## GetValue

特定の1個の構成パラメータの構成値を取得する。

---

### 構文

```
!Fortran
    Status = DftiGetValue( Desc_Handle,      &
                          Config_Param,      &
                          Config_Val )

/* C */
    status = DftiGetValue( desc_handle,
                          config_param,
                          &config_val );
```

## 説明

この関数は特定の 1 個の構成パラメータを取得する。構成パラメータは[表 11-3](#) または [表 11-4](#) に示される名前付き定数の中のものであり、構成値は対応する適切なタイプ (名前付き定数またはネイティブ・タイプ) である。

この関数は正常に終了すると DFTI\_NO\_ERROR を返す。戻り値ステータスの詳細は「[ステータス確認関数](#)」を参照のこと。

## インターフェイスとプロトタイプ

!Fortran interface

INTERFACE DftiGetValue

//Note that the body provided here is to illustrate the different

//argument list and types of dummy arguments.The interface

//does not guarantee what the actual function names are.

//Users can only rely on the function name following the

//keyword INTERFACE

FUNCTION some\_actual\_function\_7\_INTVAL( Desc\_Handle, Config\_Param, INTVAL )

INTEGER :: some\_actual\_function\_7\_INTVAL

Type(DFTI\_DESCRIPTOR), POINTER :: Desc\_Handle

INTEGER, INTENT(IN) :: Config\_Param

INTEGER, INTENT(OUT) :: INTVAL

END FUNCTION DFTI\_GET\_VALUE\_INTVAL

FUNCTION some\_actual\_function\_7\_SGLVAL( Desc\_Handle, Config\_Param, SGLVAL )

INTEGER :: some\_actual\_function\_7\_SGLVAL

Type(DFTI\_DESCRIPTOR), POINTER :: Desc\_Handle

INTEGER, INTENT(IN) :: Config\_Param

REAL, INTENT(OUT) :: SGLVAL

END FUNCTION some\_actual\_function\_7\_SGLVAL

FUNCTION some\_actual\_function\_7\_DBLVAL( Desc\_Handle, Config\_Param, DBLVAL )

INTEGER :: some\_actual\_function\_7\_DBLVAL

Type(DFTI\_DESCRIPTOR), POINTER :: Desc\_Handle

INTEGER, INTENT(IN) :: Config\_Param

REAL (KIND(0D0)), INTENT(OUT) :: DBLVAL

```
END FUNCTION some_actual_function_7_DBLVAL

FUNCTION some_actual_function_7_INTVEC( Desc_Handle, Config_Param, INTVEC )
    INTEGER :: some_actual_function_7_INTVEC
    Type(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
    INTEGER, INTENT(IN) :: Config_Param
    INTEGER, INTENT(OUT) :: INTVEC(*)
END FUNCTION some_actual_function_7_INTVEC

FUNCTION some_actual_function_7_INTPNT( Desc_Handle, Config_Param, INTPNT )
    INTEGER :: some_actual_function_7_INTPNT
    Type(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
    INTEGER, INTENT(IN) :: Config_Param
    INTEGER, DIMENSION(*), POINTER :: INTPNT
END FUNCTION some_actual_function_7_INTPNT

FUNCTION some_actual_function_7_CHARS( Desc_Handle, Config_Param, CHARS )
    INTEGER :: some_actual_function_7_CHARS
    Type(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
    INTEGER, INTENT(IN) :: Config_Param
    CHARACTER(*), INTENT(OUT):: CHARS
END FUNCTION some_actual_function_7_CHARS
END INTERFACE DftiGetValue

/* C prototype */
long DftiGetValue( DFTI_DESCRIPTOR_HANDLE,
                  DFTI_CONFIG_PARAM ,
                  ... );
```

## 構成設定

各構成パラメータは MKL\_DFTI モジュール内で名前付き定数によって区別される。C では、これら名前付き定数は列挙型 DFTI\_CONFIG\_PARAM を持つ。ユーザによって設定可能な構成パラメータのリストを表 11-3 に示す。読み出し専用構成パラメータのリストを表 11-4 に示す。なお、すべてのパラメータは読み出し可能である。これらのパラメータのほとんどは自明であるが、残りについては関連した関数に詳細を記載している。

表 11-3 設定可能な構成パラメータ

名前付き定数	値のタイプ	意味
<i>最も共通的な構成で、デフォルトはなく、明示的な設定が必要</i>		
DFTI_PRECISION	名前付き定数	計算精度
DFTI_FORWARD_DOMAIN	名前付き定数	順方向変換の領域
DFTI_DIMENSION	整数スカラ	変換の次元
DFTI_LENGTHS	整数スカラ / 配列	各次元の長さ
<i>複数の変換とデータ表現を含む共通的な構成</i>		
DFTI_NUMBER_OF_TRANSFORMS	整数スカラ	変換の回数
DFTI_FORWARD_SCALE	浮動小数点スカラ	順方向変換の倍率因子
DFTI_BACKWARD_SCALE	浮動小数点スカラ	逆方向変換の倍率因子
DFTI_PLACEMENT	名前付き定数	計算結果の配置
DFTI_COMPLEX_STORAGE	名前付き定数	格納方法、複素領域データ
DFTI_REAL_STORAGE	名前付き定数	格納方法、実数領域データ
DFTI_CONJUGATE_EVEN_STORAGE	名前付き定数	格納方法、共役偶領域データ
DFTI_DESCRIPTOR_NAME	文字列	DFTI_MAX_NAME_LENGTH より短いこと
DFTI_PACKED_FORMAT	名前付き定数	圧縮形式、実数領域データ
DFTI_NUMBER_OF_USER_THREADS	整数スカラ	DFT 計算で同じディスクリプタを使用するユーザスレッド数
<i>データのストライドに関する構成</i>		
DFTI_INPUT_DISTANCE	整数スカラ	複数変換、最初の成分の距離
DFTI_OUTPUT_DISTANCE	整数スカラ	複数変換、最初の成分の距離
DFTI_INPUT_STRIDES	整数配列	入力データのストライド情報

表 11-3 設定可能な構成パラメータ ( 続き )

名前付き定数	値のタイプ	意味
DFTI_OUTPUT_STRIDES	整数配列	出力データのストライド情報
<i>その他の構成</i>		
DFTI_ORDERING	名前付き定数	データ順のスクランブル
DFTI_TRANSPOSE	名前付き定数	次元のスクランブル

表 11-4 読み出し専用構成パラメータ

名前付き定数	値のタイプ	意味
DFTI_COMMIT_STATUS	名前付き定数	ディスクリプタが遂行されているかを示す
DFTI_VERSION	文字列	インテル MKL のバージョン番号

構成パラメータにはさまざまな値に設定される。これらの値の一部は、整数値 ( 例えば、同時変換要求数 ) や単精度値 ( 例えば、順方向変換に適用したい倍率因子 ) のようなネイティブ・データ型で規定される。

その他の構成値は性質において個別であり ( 例えば、順方向変換の領域 )、またそのため、DFTI モジュールでは名前付き定数として与えられる。C では、これらの名前付き定数は列挙型 DFTI\_CONFIG\_VALUE を持つ。この種の構成値で使用される名前付き定数の全リストを [表 11-5](#) に示す。

表 11-5 名前付き定数構成値

名前付き定数	意味
DFTI_SINGLE	単精度
DFTI_DOUBLE	倍精度
DFTI_COMPLEX	複素数領域
DFTI_REAL	実数領域
DFTI_INPLACE	出力は入力を上書き
DFTI_NOT_INPLACE	出力は入力を上書きしない
DFTI_COMPLEX_COMPLEX	格納方法 ( 「 <a href="#">格納体系</a> 」 を参照 )
DFTI_REAL_REAL	格納方法 ( 「 <a href="#">格納体系</a> 」 を参照 )
DFTI_COMPLEX_REAL	格納方法 ( 「 <a href="#">格納体系</a> 」 を参照 )
DFTI_REAL_COMPLEX	格納方法 ( 「 <a href="#">格納体系</a> 」 を参照 )
DFTI_COMMITTED	ディスクリプタの遂行ステータス
DFTI_UNCOMMITTED	ディスクリプタの遂行ステータス

**表 11-5 名前付き定数構成値 ( 続き )**

名前付き定数	意味
DFTI_ORDERED	順方向領域および逆方向領域で順序どおりのデータ
DFTI_BACKWARD_SCRAMBLED	逆方向領域 ( 順方向変換による ) でスクランブルされたデータ
DFTI_NONE	転置なしを指定
DFTI_CCS_FORMAT	圧縮形式、実数データ ( 「 <a href="#">圧縮形式</a> 」を参照 )
DFTI_PACK_FORMAT	圧縮形式、実数データ ( 「 <a href="#">圧縮形式</a> 」を参照 )
DFTI_PERM_FORMAT	圧縮形式、実数データ ( 「 <a href="#">圧縮形式</a> 」を参照 )
DFTI_CCE_FORMAT	圧縮形式、実数データ ( 「 <a href="#">圧縮形式</a> 」を参照 )
DFTI_VERSION_LENGTH	ライブラリ・バージョン長の文字数
DFTI_MAX_NAME_LENGTH	ディスクリプタ名の最大長
DFTI_MAX_MESSAGE_LENGTH	ステータス・メッセージの最大長

性質において個別なこれら構成パラメータで設定可能な値を[表 11-6](#)に示す。

**表 11-6 個別の構成パラメータの設定**

名前付き定数	設定可能な値
DFTI_PRECISION	DFTI_SINGLE、または DFTI_DOUBLE ( デフォルト値なし )
DFTI_FORWARD_DOMAIN	DFTI_COMPLEX、または DFTI_REAL
DFTI_PLACEMENT	DFTI_INPLACE ( デフォルト ) または DFTI_NOT_INPLACE
DFTI_COMPLEX_STORAGE	DFTI_COMPLEX_COMPLEX ( デフォルト )
DFTI_REAL_STORAGE	DFTI_REAL_REAL ( デフォルト )、または DFTI_REAL_COMPLEX
DFTI_CONJUGATE_EVEN_STORAGE	DFTI_COMPLEX_COMPLEX、または DFTI_COMPLEX_REAL ( デフォルト )
DFTI_PACKED_FORMAT	DFTI_CCS_FORMAT ( デフォルト )、または、 DFTI_PACK_FORMAT、または DFTI_PERM_FORMAT、または DFTI_CCE_FORMAT



[表 11-7](#) に設定可能な構成パラメータのデフォルト値のリストを示す。

**表 11-7      設定可能パラメータのデフォルト構成値**

名前付き定数	デフォルト値
DFTI_NUMBER_OF_TRANSFORMS	1
DFTI_NUMBER_OF_USER_THREADS	1
DFTI_FORWARD_SCALE	1.0
DFTI_BACKWARD_SCALE	1.0
DFTI_PLACEMENT	DFTI_INPLACE
DFTI_COMPLEX_STORAGE	DFTI_COMPLEX_COMPLEX
DFTI_REAL_STORAGE	DFTI_REAL_REAL
DFTI_CONJUGATE_EVEN_STORAGE	DFTI_COMPLEX_REAL
DFTI_PACKED_FORMAT	DFTI_CCS_FORMAT
DFTI_DESCRIPTOR_NAME	名前なし、長さゼロの文字列
DFTI_INPUT_DISTANCE	0
DFTI_OUTPUT_DISTANCE	0
DFTI_INPUT_STRIDES	次元、長さ、格納に従って厳密に圧縮されている
DFTI_OUTPUT_STRIDES	同上。詳細は「 <a href="#">ストライド</a> 」を参照のこと。
DFTI_ORDERING	DFTI_ORDERED
DFTI_TRANSPOSE	DFTI_NONE

### 変換精度

構成パラメータ `DFTI_PRECISION` は変換の実行精度が浮動小数点であることを示す。設定が `DFTI_SINGLE` の場合は単精度が有効で、設定が `DFTI_DOUBLE` の場合は倍精度が有効となる。データがこの精度で示されることを意味する。すなわち計算はこの精度で実行され、結果はこの精度で与えられる。これはデフォルト値を持たない設定可能な 4 個の構成パラメータのうちの 1 個である。ユーザの明示的な設定が必要で、ディスクリプタ生成関数 [DftiCreateDescriptor](#) の呼び出しが最も便利である。

## 順方向変換

離散フーリエ変換の一般の形は次のとおりである。

$$Z_{k_1, k_2, \dots, k_d} = \sigma \times \sum_{j_d=0}^{n_d-1} \dots \sum_{j_2=0}^{n_2-1} \sum_{j_1=0}^{n_1-1} w_{j_1, j_2, \dots, j_d} \exp \left( \delta i 2\pi \sum_{l=1}^d j_l k_l / n_l \right) \quad (7.1)$$

$k_l = 0, \pm 1, \pm 2, \dots$ 、ここで  $\sigma$  は任意実数値の倍率因子、また  $\delta = \pm 1$  である。順方向変換は  $\sigma = 1$  および  $\delta = -1$  として定義される。最も一般的な状況では、順方向変換の領域、すなわち入力 (周期的) シーケンス  $\{w_{j_1, j_2, \dots, j_d}\}$  が属するセットは、複素シーケンス、実数シーケンス、複素共役偶シーケンスのいずれかのセットである。構成パラメータ `DFTI_FORWARD_DOMAIN` は順方向変換の領域を示す。この黙示性によって、DFT の数学的な特性から逆方向変換の領域が指定される点に注意する。詳細は表 11-8 を参照のこと。

表 11-8 順方向領域と逆方向領域の対応

順方向領域		黙示的な逆方向領域
複素数	( <code>DFTI_COMPLEX</code> )	複素数
実数	( <code>DFTI_REAL</code> )	共役偶

実領域での変換において、一部のソフトウェア・パッケージは単一の「実数から複素数」変換のみを提供している。これは順方向変換での共役偶領域が本質的に省略されている。順方向領域構成パラメータ `DFTI_FORWARD_DOMAIN` は、デフォルトを持たない 4 個の構成パラメータのうちの 2 個目である。

## 変換の次元と長さ

変換の次元は、Fortran では Integer データ型、C では long データ型の整数スカラで表される正の整数値である。1 次元変換において、変換の長さは、Fortran では Integer データ型、C では long データ型の整数スカラで表される正の整数値によって指定される。多次元 ( $\geq 2$ ) 変換では、各次元の長さは整数配列 (Fortran では Integer データ型、C では long データ型) で与えられる。`DFTI_DIMENSION` と `DFTI_LENGTHS` は、デフォルトを持たない 4 個の構成パラメータのうちの残りの 2 個である。

これまでに述べたように、これら 4 個のパラメータはデフォルト値を持っていない。これらはディスクリプタ生成関数で設定するのがもっとも妥当である。これらの値は、`DftiSetValue` 関数では設定できず、ディスクリプタ生成関数でしか設定できない。

## 変換回数

状況によっては、ユーザは同じ次元、同じ長さの DFT 変換を複数回実行したい場合がある。最も一般的な状況として、同じ長さを持つ複数の 1 次元データの変換が考えられる。このパラメータのデフォルト値は 1 で、Fortran では Integer データ型、C では long データ型によって、正の整数として設定する。データセットは共通成分を持たない。回数値が 1 より大きい場合は距離パラメータを必須とする。

## 倍率

順方向変換と逆方向変換は、それぞれの倍率因子  $\sigma$  のデフォルト値が 1 という点で互いに関連している。ユーザは 2 個の構成パラメータ DFTI\_FORWARD\_SCALE と DFTI\_BACKWARD\_SCALE によって、どちらか、または両方の設定が可能である。例えば、長さ  $n$  の 1 次元の変換で、順方向変換の倍率はデフォルトの 1 を使用し、一方、逆方向変換の倍率を  $1/n$  に設定すれば、逆方向変換を順方向変換の逆にできる。

倍率因子構成パラメータは、DFTI\_PRECISION と同じ精度の実数浮動小数点データ型によって設定しなければならない。

## 結果の配置

デフォルトでは、計算関数は入力データを出力結果で上書きする。すなわち、構成パラメータ DFTI\_PLACEMENT のデフォルト設定は DFTI\_INPLACE である。ユーザは、これを DFTI\_NOT\_INPLACE に変更可能である。データセットは共通成分を持たない。

## 圧縮形式

実数の順方向変換 (例えば、周波数領域) の結果は、**Pack**、**Perm**、**CCS**、または **CCE** の圧縮形式によって表現される。実数データの DFT 変換の対称性によってデータは圧縮可能になっている。

**CCE** 形式は順方向 DFT から得られる出力複素共役偶信号の前半の値を格納する。CCE 形式に格納される 1 次元信号は 1 個の複素成分だけ長いことに注意する。多次元実数変換  $n_1 * n_2 * n_3 * \dots * n_k$  に対する CCE 形式の複素行列のサイズは、 $(n_1/2+1) * n_2 * n_3 * \dots * n_k$  (Fortran の場合) または  $n_1 * n_2 * \dots * (n_k/2+1)$  (C の場合) である。

**CCS** 形式は CCE 形式に似ている。1 次元変換では CCE と同じ形式で、多次元実数変換ではやや異なる。CCS 形式では、DFT の出力標本は、1 次元 DFT の [表 11-9](#) および 2 次元の [表 11-10](#) に示されるように整列配置される。

**Pack** 形式は複素共役対称シーケンスをコンパクトに表現したものである。この形式の欠点は、real DFT アルゴリズムで使用されている自然な形式とは異なることである（この「自然」とは配列が complex DFT に対して自然であることを意味する）。Pack 形式では、DFT の出力標本は、1 次元 DFT の表 11-9 および 2 次元の表 11-11 に示されるように整列配置される。

**Perm** 形式は、偶数長の Pack 形式に対しては任意の置換であり、奇数長の Pack 形式に対しては同一である。Perm 形式では、DFT の出力標本は、1 次元 DFT の表 11-9 および 2 次元の表 11-12 に示されるように整列配置される。

**表 11-9 Pack 形式出力標本**

(n = s\*2) の場合

DFT Real	0	1	2	3	...	n-2	n-1	n	n+1
CCS	R <sub>0</sub>	0	R <sub>1</sub>	I <sub>1</sub>	...	R <sub>n/2-1</sub>	I <sub>n/2-1</sub>	R <sub>n/2</sub>	0
Pack	R <sub>0</sub>	R <sub>1</sub>	I <sub>1</sub>	R <sub>2</sub>	...	I <sub>n/2-1</sub>	R <sub>n/2</sub>		
Perm	R <sub>0</sub>	R <sub>n/2</sub>	R <sub>1</sub>	I <sub>1</sub>	...	R <sub>n/2-1</sub>	I <sub>n/2-1</sub>		

(n = s\*2 + 1) の場合

DFT Real	0	1	2	3	...	n-4	n-3	n-2	n-1	n	n+1
CCS	R <sub>0</sub>	0	R <sub>1</sub>	I <sub>1</sub>	...	I <sub>s-2</sub>	R <sub>s-1</sub>	I <sub>s-1</sub>	R <sub>s</sub>	I <sub>s</sub>	
Pack	R <sub>0</sub>	R <sub>1</sub>	I <sub>1</sub>	R <sub>2</sub>	...	R <sub>s-1</sub>	I <sub>s-1</sub>	R <sub>s-1</sub>	I <sub>s</sub>		
Perm	R <sub>0</sub>	R <sub>1</sub>	I <sub>1</sub>	R <sub>2</sub>	...	R <sub>s-1</sub>	I <sub>s-1</sub>	R <sub>s-1</sub>	I <sub>s</sub>		

表 11-9 は複素数データ成分に対して次の表記を使用する点に注意する。

$$R_j = \text{Re } z_j$$

$$I_j = \text{Im } z_j$$

表 11-13 と表 11-14 も参照のこと。

表 11-10 CCS 形式出力標本 (2 次元行列  $(m+2) \times (n+2)$ )

(m = s*2) の場合								
z(1,1)	0	REz(1,2)	IMz(1,2)	...	REz(1,k)	IMz(1,k)	z(1,k+1)	0
0	0	0	0	...	0	0	0	0
REz(2,1)	REz(2,2)	REz(2,3)	REz(2,4)	...	REz(2,n-1)	REz(2,n)	n/u	n/u
IMz(2,1)	IMz(2,2)	IMz(2,3)	IMz(2,4)	...	IMz(2,n-1)	IMz(2,n)	n/u	n/u
...	...	...	...	...	...	...	n/u	n/u
REz(m/2,1)	REz(m/2,2)	REz(m/2,3)	REz(m/2,4)	...	REz(m/2,n-1)	REz(m/2,n)	n/u	n/u
IMz(m/2,1)	IMz(m/2,2)	IMz(m/2,3)	IMz(m/2,4)	...	IMz(m/2,n-1)	IMz(m/2,n)	n/u	n/u
z(m/2+1,1)	0	REz(m/2+1,2)	IMz(m/2+1,2)	...	REz(m/2+1,k)	IMz(m/2+1,k)	z(m/2+1,k+1)	0
0	0	0	0	...	0	0	n/u	n/u
(m = s*2+1) の場合								
z(1,1)	0	REz(1,2)	IMz(1,2)	...	REz(1,k)	IMz(1,k)	z(1,k+1)	0
0	0	0	0	...	0	0	0	0
REz(2,1)	REz(2,2)	REz(2,3)	REz(2,4)	...	REz(2,n-1)	REz(2,n)	n/u	n/u
IMz(2,1)	IMz(2,2)	IMz(2,3)	IMz(2,4)	...	IMz(2,n-1)	IMz(2,n)	n/u	n/u
...	...	...	...	...	...	...	n/u	n/u
REz(s,1)	REz(s,2)	REz(s,3)	REz(s,4)	...	REz(s,n-1)	REz(s,n)	n/u	n/u
IMz(s,1)	IMz(s,2)	IMz(s,3)	IMz(s,4)	...	IMz(s,n-1)	IMz(s,n)	n/u	n/u

\* n/u - 使用しない

表 11-10 の  $(n+2)$  列は偶数  $n = k*2$  に使用され、 $n$  列は奇数  $n = k*2+1$  に使用される。  
後者の場合、最初の行は次のとおりである。

z(1,1)      0      REz(1,2)    IMz(1,2)    ...    REz(1,k)      IMz(1,k)

$m$  が偶数の場合、 $(m+1)$  番目の行は次のとおりである。

z(m/2+1,1)    0    REz(m/2+1,2)    IMz(m/2+1,2)    ...    REz(m/2+1,k)    IMz(m/2+1,k)

**表 11-11** *Pack* 形式出力標本 (2 次元行列 ( $m$ )  $\times$  ( $n$ ))

(m = s*2) の場合						
$z(1,1)$	$REz(1,2)$	$IMz(1,2)$	$REz(1,3)$	...	$IMz(1,k)$	$z(1,k+1)$
$REz(2,1)$	$REz(2,2)$	$REz(2,3)$	$REz(2,4)$	...	$REz(2,n-1)$	$REz(2,n)$
$IMz(2,1)$	$IMz(2,2)$	$IMz(2,3)$	$IMz(2,4)$	...	$IMz(2,n-1)$	$IMz(2,n)$
...	...	...	...	...	...	...
$REz(m/2,1)$	$REz(m/2,2)$	$REz(m/2,3)$	$REz(m/2,4)$	...	$REz(m/2,n-1)$	$REz(m/2,n)$
$IMz(m/2,1)$	$IMz(m/2,2)$	$IMz(m/2,3)$	$IMz(m/2,4)$	...	$IMz(m/2,n-1)$	$IMz(m/2,n)$
$z(m/2+1,1)$	$REz(m/2+1,2)$	$IMz(m/2+1,2)$	$REz(m/2+1,3)$	...	$IMz(m/2+1,k)$	$z(m/2+1,k+1)$
(m = s*2+1) の場合						
$z(1,1)$	$REz(1,2)$	$IMz(1,2)$	$REz(1,3)$	...	$IMz(1,k)$	$z(1,n/2+1)$
$REz(2,1)$	$REz(2,2)$	$REz(2,3)$	$REz(2,4)$	...	$REz(2,n-1)$	$REz(2,n)$
$IMz(2,1)$	$IMz(2,2)$	$IMz(2,3)$	$IMz(2,4)$	...	$IMz(2,n-1)$	$IMz(2,n)$
...	...	...	...	...	...	...
$REz(s,1)$	$REz(s,2)$	$REz(s,3)$	$REz(s,4)$	...	$REz(s,n-1)$	$REz(s,n)$
$IMz(s,1)$	$IMz(s,2)$	$IMz(s,3)$	$IMz(s,4)$	...	$IMz(s,n-1)$	$IMz(s,n)$

**表 11-12** *Perm* 形式出力標本 (2 次元行列 ( $m$ )  $\times$  ( $n$ ))

(m = s*2) の場合						
$z(1,1)$	$z(1,k+1)$	$REz(1,2)$	$IMz(1,2)$	...	$REz(1,k)$	$IMz(1,k)$
$z(m/2+1,1)$	$z(m/2+1,k+1)$	$REz(m/2+1,2)$	$IMz(m/2+1,2)$	...	$REz(m/2+1,k)$	$IMz(m/2+1,k)$
$REz(2,1)$	$REz(2,2)$	$REz(2,3)$	$REz(2,4)$	...	$REz(2,n-1)$	$REz(2,n)$
$IMz(2,1)$	$IMz(2,2)$	$IMz(2,3)$	$IMz(2,4)$	...	$IMz(2,n-1)$	$IMz(2,n)$
...	...	...	...	...	...	...
$REz(m/2,1)$	$REz(m/2,2)$	$REz(m/2,3)$	$REz(m/2,4)$	...	$REz(m/2,n-1)$	$REz(m/2,n)$
$IMz(m/2,1)$	$IMz(m/2,2)$	$IMz(m/2,3)$	$IMz(m/2,4)$	...	$IMz(m/2,n-1)$	$IMz(m/2,n)$
(m = s*2+1) の場合						
$z(1,1)$	$z(1,k+1)$	$REz(1,2)$	$IMz(1,2)$	...	$REz(1,k)$	$IMz(1,k)$
$REz(2,1)$	$REz(2,2)$	$REz(2,3)$	$REz(2,4)$	...	$REz(2,n-1)$	$REz(2,n)$
$IMz(2,1)$	$IMz(2,2)$	$IMz(2,3)$	$IMz(2,4)$	...	$IMz(2,n-1)$	$IMz(2,n)$
...	...	...	...	...	...	...
$REz(s,1)$	$REz(s,2)$	$REz(s,3)$	$REz(s,4)$	...	$REz(s,n-1)$	$REz(s,n)$
$IMz(s,1)$	$IMz(s,2)$	$IMz(s,3)$	$IMz(s,4)$	...	$IMz(s,n-1)$	$IMz(s,n)$

表 11-11 および表 11-12 では、列数が偶数の場合は  $n = k*2$ 、奇数の場合は  $n = k*2+1$  で、最初の行は次のとおりである。

$z(1,1)$   $REz(1,2)$   $IMz(1,2)$  ...  $REz(1,k)$   $IMz(1,k)$

$m$  が偶数の場合、Pack 形式の最後の行と Perm 形式の 2 番目の行は次のとおりである。

$z(m/2+1,1)$   $REz(m/2+1,2)$   $IMz(m/2+1,2)$  ...  $REz(m/2+1,k)$   $IMz(m/2+1,k)$

2 次元 DFT の表は Fortran インターフェイス規則を使用する。圧縮データの格納における C インターフェイスの詳細は、次の「格納体系」のセクションを参照のこと。

また、Fortran インターフェイス形式と C インターフェイス形式の例は、表 11-15 と表 11-16 を参照のこと。

## 格納体系

DFTI\_COMPLEX、DFTI\_REAL、DFTI\_CONJUGATE\_EVEN (これは順方向の場合だが逆方向演算も同様) の 3 種類の領域の計算に対して、4 つの格納体系のサブセット

DFTI\_COMPLEX\_COMPLEX、DFTI\_COMPLEX\_REAL、DFTI\_REAL\_COMPLEX、および DFTI\_REAL\_REAL が提供される。ここでは格納体系を説明するための例を示す。この格納体系の定義の背景にある理論的根拠は文献 [3] を参照のこと。



**注：**データは Fortran 形式でのみ格納される。すなわち、実数部と虚数部は隣接して格納される。

**複素領域の格納体系：**この設定は構成パラメータ DFTI\_COMPLEX\_STORAGE に登録される。設定可能な 3 種類の値は、DFTI\_COMPLEX\_COMPLEX、DFTI\_COMPLEX\_REAL、または DFTI\_REAL\_REAL である。1 次元、長さ  $n$  の形式の変換を考える。

$$z_k = \sum_{j=0}^{n-1} w_j e^{-i2\pi jk/n}, \quad w_j, z_k \in \mathbb{C}$$

ストライドはデフォルト値 (単位ストライド) を持ち、DFTI\_PLACEMENT 設定はデフォルトのインプレース (入力上書き) であると仮定する。

**DFTI\_COMPLEX\_COMPLEX** 格納体系 (デフォルト)。典型的な使用法は次のとおりである。

```
COMPLEX :: X(0:n-1)
```

```
...some other code...
```

```
Status = DftiComputeForward( Desc_Handle, X )
```

入力において、

$$x(j) = w_j, j = 0, 1, \dots, n-1$$

出力において、

$$x(k) = z_k, k = 0, 1, \dots, n-1$$

**実数領域と共役偶領域の格納体系：**これら領域に対する格納体系の設定は構成パラメータ DFTI\_REAL\_STORAGE と DFTI\_CONJUGATE\_EVEN\_STORAGE に登録される。順方向の実数領域は逆方向の共役偶領域に一致することから、これらは一緒に扱われる。[1次元](#)、[2次元](#)、および [3次元](#) 実数を共役偶に変換する例を使用した。可能であればインプレース計算を仮定している（すなわち、入力データ型が出力データ型に一致する場合）。

## 1 次元変換

1 次元、長さ  $n$  の形式の変換を考える。

$$z_k = \sum_{j=0}^{n-1} w_j e^{-i2\pi jk/n}, \quad w_j \in \mathbb{R}, z_k \in \mathbb{C}$$

これは対称である。

$n$  が偶数の場合： $z(n/2+i) = \text{conjg}(z(n/2-i))$ 、 $1 \leq i \leq n/2-1$ 、またさらに、 $z(0)$  と  $z(n/2)$  は実数値。

$n$  が奇数の場合： $z(m+i) = \text{conjg}(z(m-i+1))$ 、 $1 \leq i \leq m$ 、またさらに、 $z(0)$  は実数値。

$$m = \text{floor}(n/2)$$

**表 11-13 順方向変換での複素 - 複素 DFT および実数 - 複素 DFT の格納効果の比較**

N=8								
入力ベクトル			出力ベクトル					
Complex DFT		Real DFT	complex DFT		real DFT			
複素数データ		実数データ	複素数データ		実数データ			
実数部	虚数部		実数部	虚数部	CCS	Pack	Perm	
w0	0.000000	w0	z0	0.000000	z0	z0	z0	
w1	0.000000	w1	Re(z1)	Im(z1)	0.000000	Re(z1)	z4	
w2	0.000000	w2	Re(z2)	Im(z2)	Re(z1)	Im(z1)	Re(z1)	
w3	0.000000	w3	Re(z3)	Im(z3)	Im(z1)	Re(z2)	Im(z1)	



表 11-13 順方向変換での複素 - 複素 DFT および実数 - 複素 DFT の格納効果の比較

N=8								
入力ベクトル			出力ベクトル					
Complex DFT		Real DFT	complex DFT			real DFT		
複素数データ		実数データ	複素数データ			実数データ		
実数部	虚数部		実数部	虚数部	CCS	Pack	Perm	
w4	0.000000	w4	z4	0.000000	Re(z2)	Im(z2)	Re(z2)	
w5	0.000000	w5	Re(z3)	-Im(z3)	Im(z2)	Re(z3)	Im(z2)	
w6	0.000000	w6	Re(z2)	-Im(z2)	Re(z3)	Im(z3)	Re(z3)	
w7	0.000000	w7	Re(z1)	-Im(z1)	Im(z3)	z4	Im(z3)	
					z4			
					0.000000			

N=7								
入力ベクトル			出力ベクトル					
Complex DFT		Real DFT	complex DFT			real DFT		
複素数データ		実数データ	複素数データ			実数データ		
実数部	虚数部		実数部	虚数部	CCS	Pack	Perm	
w0	0.000000	w0	z0	0.000000	z0	z0	z0	
w1	0.000000	w1	Re(z1)	Im(z1)	0.000000	Re(z1)	Re(z1)	
w2	0.000000	w2	Re(z2)	Im(z2)	Re(z1)	Im(z1)	Im(z1)	
w3	0.000000	w3	Re(z3)	Im(z3)	Im(z1)	Re(z2)	Re(z2)	
w4	0.000000	w4	Re(z3)	-Im(z3)	Re(z2)	Im(z2)	Im(z2)	
w5	0.000000	w5	Re(z2)	-Im(z2)	Im(z2)	Re(z3)	Re(z3)	
w6	0.000000	w6	Re(z1)	-Im(z1)	Re(z3)	Im(z3)	Im(z3)	
					Im(z3)			

表 11-14 逆方向変換での複素 - 複素 DFT および複素 - 実数 DFT の格納効果の比較

N=8								
出力ベクトル			入力ベクトル					
Complex DFT		Real DFT	complex DFT					
複素数データ		実数データ	複素数データ					
実数部	虚数部		実数部	虚数部	CCS	Pack	Perm	
w0	0.000000	w0	z0	0.000000	z0	z0	z0	
w1	0.000000	w1	Re(z1)	Im(z1)	0.000000	Re(z1)	z4	
w2	0.000000	w2	Re(z2)	Im(z2)	Re(z1)	Im(z1)	Re(z1)	
w3	0.000000	w3	Re(z3)	Im(z3)	Im(z1)	Re(z2)	Im(z1)	
w4	0.000000	w4	z4		Re(z2)	Im(z2)	Re(z2)	
w5	0.000000	w5	Re(z3)	-Im(z3)	Im(z2)	Re(z3)	Im(z2)	
w6	0.000000	w6	Re(z2)	-Im(z2)	Re(z3)	Im(z3)	Re(z3)	
w7	0.000000	w7	Re(z1)	-Im(z1)	Im(z3)	z4	Im(z3)	
					z4			
					0.000000			

N=7								
出力ベクトル			入力ベクトル					
Complex DFT		Real DFT	complex DFT			real DFT		
複素数データ		実数データ	複素数データ			実数データ		
実数部	虚数部		実数部	虚数部	CCS	Pack	Perm	
w0	0.000000	w0	z0	0.000000	z0	z0	z0	
w1	0.000000	w1	Re(z1)	Im(z1)	0.000000	Re(z1)	Re(z1)	
w2	0.000000	w2	Re(z2)	Im(z2)	Re(z1)	Im(z1)	Im(z1)	
w3	0.000000	w3	Re(z3)	Im(z3)	Im(z1)	Re(z2)	Re(z2)	
w4	0.000000	w4	Re(z3)	-Im(z3)	Re(z2)	Im(z2)	Im(z2)	
w5	0.000000	w5	Re(z2)	-Im(z2)	Im(z2)	Re(z3)	Re(z3)	
w6	0.000000	w6	Re(z1)	-Im(z1)	Re(z3)	Im(z3)	Im(z3)	
					Im(z3)			

ストライドはデフォルト値 ( 単位ストライド ) であると仮定する。

この複素共役対称ベクトルは、圧縮形式に依存して、大きさ  $m+1$  の複素配列、または大きさ  $2m+2$  か  $2m$  の実数配列に格納できる。

## 2 次元変換

実数から複素数へのルーチンは、いずれも次の式に従って、2 次元実数行列の順方向の DFT を計算する。

$$z_{i,j} = \sum_{k=0}^{m-1} \sum_{l=0}^{n-1} t_{k,l} w_m^{-i*k} w_n^{-j*l}, \quad 0 \leq i \leq m-1, \quad 0 \leq j \leq n-1$$

$t_{k,l} = \text{cmplx}(r_{k,l}, 0)$ 、ここで  $r_{k,l}$  は実数入力行列で、 $0 \leq k \leq m-1, 0 \leq l \leq n-1$  である。数値演算の結果  $z_{i,j}, 0 \leq i \leq m-1, 0 \leq j \leq n-1$  は、サイズ  $(m,n)$  の複素行列である。各列は、次のような複素共役対称ベクトルになる。

$m$  が偶数の場合：

$0 \leq j \leq n-1$  の場合は、

$$z(m/2+i, j) = \text{conjg}(z(m/2-i, j)), \quad 1 \leq i \leq m/2-1$$

また、 $z(0, j)$  と  $z(m/2, j)$  の場合は、 $j = 0$  と  $j = n/2$  は実数値になる。

$m$  が奇数の場合：

$0 \leq j \leq n-1$  の場合は、

$$z(s+i, j) = \text{conjg}(z(s-i, j)), \quad 1 \leq i \leq s-1$$

ここで  $s = \text{floor}(m/2)$

また、 $z(0, j)$  の場合は  $j = 0$  と  $j = n/2$  は実数値になる。

この数値演算の結果は、実数 2 次元配列に次のように格納される。

$(m+2, n+2)$  (CCS 形式の場合)、

$(m, n)$  (Pack または Perm 形式の場合)、

$(2*(m/2+1), n)$  (CCE 形式、Fortran インターフェイスの場合) または

$(m, 2*(n/2+1))$  (CCE 形式、C インターフェイスの場合)

または、複素数 2 次元配列に次のように格納される。

$(m/2+1, n)$  (CCE 形式、Fortran インターフェイス) または

$(m, n/2+1)$  (CCE 形式、C インターフェイス)

多次元配列データの配置は Fortran と C では異なるため (「[ストライド](#)」を参照)、計算結果を格納する出力配列は複素共役対称列 (Fortran の場合) または 複素共役対称行 (C の場合) を含む。

次の表は、 $6 \times 4$  の実数行列に対して実数から複素数への順方向 2 次元 DFT を行った場合の、Pack 形式の出力データのレイアウト例を示す。同じレイアウトが、対応する複素数から実数への逆方向 DFT の入力データに使用される点に注意する。

**表 11-15  $6 \times 4$  の行列の Fortran インターフェイス・データ・レイアウト**

z(1,1)	Re z(1,2)	Im z(1,2)	z(1,3)
Re z(2,1)	Re z(2,2)	Re z(2,3)	Re z(2,4)
Im z(2,1)	Im z(2,2)	Im z(2,3)	Im z(2,4)
Re z(3,1)	Re z(3,2)	Re z(3,3)	Re z(3,4)
Im z(3,1)	Im z(3,2)	Im z(3,3)	Im z(3,4)
z(4,1)	Re z(4,2)	Im z(4,2)	z(4,3)

上の例で、ストライド配列は (0, 1, 6) をとる。

**表 11-16  $6 \times 4$  の行列の C インターフェイス・データ・レイアウト**

z(1,1)	Re z(1,2)	Im z(1,2)	z(1,3)
Re z(2,1)	Re z(2,2)	Im z(2,2)	Re z(2,3)
Im z(2,1)	Re z(3,2)	Im z(3,2)	Im z(2,3)
Re z(3,1)	Re z(4,2)	Im z(4,2)	Re z(3,3)
Im z(3,1)	Re z(5,2)	Im z(5,2)	Im z(3,3)
z(4,1)	Re z(6,2)	Im z(6,2)	z(4,3)

2 つ目の例で、ストライド配列は /0, 4, 1/ をとる。

「[圧縮形式](#)」も参照のこと。

### 3 次元変換

実数から複素数へのルーチンは、いずれも次の式に従って、3 次元実数行列の順方向の DFT を計算する。

$$z_{i,j,q} = \sum_{k=0}^{m-1} \sum_{l=0}^{n-1} \sum_{s=0}^{k-1} t_{p,l,s} w_m^{-i*p} w_n^{-j*l} w_k^{-q*s}, \quad 0 \leq i \leq m-1, \quad 0 \leq j \leq n-1,$$

$$0 \leq s \leq k-1$$

$t_{p,l,s} = \text{cmplx}(r_{p,l,s}, 0)$ 、ここで  $r_{p,l,s}$  は実数入力行列で、 $0 \leq k \leq m-1$ ,  $0 \leq l \leq n-1$ ,  $0 \leq s \leq k-1$  である。数値演算の結果  $z_{i,j,q}$ 、 $0 \leq i \leq m-1$ ,  $0 \leq j \leq n-1$ ,  $0 \leq q \leq k-1$  は、次のようなサイズ  $(m, n, k)$  の複素共役対称行列または複素共役偶行列である。

$z_{m1,n1,k1} = \text{conjg}(z_{m-m1,n-n1,k-k1})$  では、各次元は周期的である。

この数値演算の結果は、実数 3 次元配列に次のように格納される。

$(m/2+1, n, k)$  (CCE 形式、Fortran インターフェイスの場合)、  
 $(m, n, k/2+1)$  (CCE 形式、C インターフェイスの場合)

多次元配列データの配置は Fortran と C では異なるため (「[ストライド](#)」を参照)、計算結果を格納する出力配列は複素共役対称列 (Fortran の場合) または複素共役対称行 (C の場合) を含む。

現バージョンでは、3 次元 REAL DFT はアウトオブプレース変換として実装される。

1. 実数領域では **DFTI\_REAL\_REAL**、共役偶領域では **DFTI\_COMPLEX\_REAL** (デフォルト)。1 次元および 2 次元 REAL DFT に使用される。典型的な使用法は次のとおりである。

```
// m = floor( n/2 )
REAL :: X(0:2*m+1)
...some other code...
...assuming inplace...
Status = DftiComputeForward( Desc_Handle, X )
```

入力において、

$x(j) = w_j, j = 0, 1, \dots, n-1$

出力において、

出力データは、Pack、Perm、または CCS のどれか 1 つの形式で格納される (「[圧縮形式](#)」を参照)。

CCS 形式:  $x(2*k) = \text{Re}(z_k)$ ,  $x(2*k+1) = \text{Im}(z_k)$ ,  $k = 0, 1, \dots, m$

Pack 形式: 偶数  $n$ :  $x(0) = \text{Re}(z_0)$ ,  $x(2*k-1) = \text{Re}(z_k)$ ,  $x(2*k) = \text{Im}(z_k)$ ,  
 $k = 1, \dots, m-1$ ,  $x(n-1) = \text{Re}(z_m)$

奇数  $n$ :  $x(0) = \text{Re}(z_0)$ ,  $x(2*k-1) = \text{Re}(z_k)$ ,  $x(2*k) = \text{Im}(z_k)$ ,  $k = 1, \dots, m$

Perm 形式 : 偶数  $n$ :  $x(0) = \text{Re}(z_0)$ ,  $x(1) = \text{Re}(z_m)$ ,  $x(2*k) = \text{Re}(z_k)$ ,  
 $x(2*k+1) = \text{Im}(z_k)$ ,  $k = 1, \dots, m-1$

奇数  $n$ :  $x(0) = \text{Re}(z_0)$ ,  $x(2*k-1) = \text{Re}(z_k)$ ,  $x(2*k) = \text{Im}(z_k)$ ,  $k = 1, \dots, m$

2. 実数領域では **DFTI\_REAL\_REAL**、共役偶領域では **DFTI\_COMPLEX\_REAL** (デフォルト)。1次元および2次元 REAL DFT に使用される。典型的な使用法は次のとおりである。

```
// m = floor( n/2 )
REAL :: X(0:n-1)
REAL :: Y(0:2*m+1)
...some other code...
...assuming out-of-place...
Status = DftiComputeForward( Desc_Handle, X, Y )
```

入力において、

$x(j) = w^j$ ,  $j = 0, 1, \dots, n-1$

出力において、

出力データは、Pack、Perm、または CCS のどれか 1 つの形式で格納される (「[圧縮形式](#)」を参照)。

CCS 形式 :  $Y(2*k) = \text{Re}(z_k)$ ,  $Y(2*k+1) = \text{Im}(z_k)$ ,  $k = 0, 1, \dots, m$

Pack 形式 : 偶数  $n$ :  $y(0) = \text{Re}(z_0)$ ,  $y(2*k-1) = \text{Re}(z_k)$ ,  $y(2*k) = \text{Im}(z_k)$ ,  
 $k = 1, \dots, m-1$ ,  $y(n-1) = \text{Re}(z_m)$

奇数  $n$ :  $y(0) = \text{Re}(z_0)$ ,  $y(2*k-1) = \text{Re}(z_k)$ ,  $y(2*k) = \text{Im}(z_k)$ ,  $k = 1, \dots, m$

Perm 形式 : 偶数  $n$ :  $y(0) = \text{Re}(z_0)$ ,  $y(1) = \text{Re}(z_m)$ ,  $y(2*k) = \text{Re}(z_k)$ ,  
 $y(2*k+1) = \text{Im}(z_k)$ ,  $k = 1, \dots, m-1$

奇数  $n$ :  $y(0) = \text{Re}(z_0)$ ,  $y(2*k-1) = \text{Re}(z_k)$ ,  $y(2*k) = \text{Im}(z_k)$ ,  $k = 1, \dots, m$

出力配列のストライドがデフォルト値 (単位ストライド) に設定されていない場合、1 個の複素成分の実数部と虚数部はこのストライドで配置される。

例えば、

CCS 形式 :  $Y(2*k*s) = \text{Re}(z_k)$ ,  $Y(2*k+1*s) = \text{Im}(z_k)$ ,  $k = 0, 1, \dots, m$ ,  $s = \text{stride}$

3. 実数領域では **DFTI\_REAL\_REAL**、共役偶領域では **DFTI\_COMPLEX\_COMPLEX**。1 次元、2 次元、および 3 次元 REAL DFT に使用される。CCE 形式はデフォルトで設定される。典型的な使用法は次のとおりである。

```
// m = floor( n/2 )
REAL :: X(0:n-1)
COMPLEX :: Y(0:m)
...some other code...
...out of place transform...
Status = DftiComputeForward( Desc_Handle, X, Y )
```

入力において、

$$X(j) = w_j, j = 0, 1, \dots, n-1$$

出力において、

$$Y(k) = z_k, k = 0, 1, \dots, m$$

4. 実数領域では **DFTI\_REAL\_COMPLEX**、共役偶領域では **DFTI\_COMPLEX\_COMPLEX**。現在のバージョンでは使用しない。11-3 ページを参照のこと。典型的な使用法は次のとおりである。

```
// m = floor( n/2 )
COMPLEX :: X(0:m)
...some other code...
...inplace transform...
Status = DftiComputeForward( Desc_Handle, X )
```

入力において、

$$X(j) = w_j, j = 0, 1, \dots, n-1$$

すなわち、 $X(j)$  の虚数部はゼロである。

出力において、

$$Y(k) = z_k, k = 0, 1, \dots, m$$

ここで  $m$  は  $\lfloor n/2 \rfloor$  である。

## ユーザスレッド数

次に示す技法によりカスタム・アプリケーションを並列化することができる。

1. アプリケーションでスレッドを作成しないが、インテル MKL の DFT モジュール内で並列モードを指定する。詳細は「インテル MKL テクニカル・ユーザ・ノート」を参照のこと。
2. アプリケーションでスレッドを作成して、ディスクリプタの初期化、DFT 計算、ディスクリプタの割り当て解除を含む、DFT 実装のすべての段階を各スレッドに実行させる。この場合、それぞれのディスクリプタは対応するスレッド内でのみ使用される。
3. DFT ディスクリプタを初期化した後でスレッドを作成する。これは、スレッディングが並列 DFT 計算でのみ使用され、並列領域から戻った後でディスクリプタが解放されることを意味する。この場合、各スレッドは同じディスクリプタを使用する。

上記の 1 と 2 の技法では、特定のディスクリプタはそれぞれシングルスレッドでのみ使用されるため、パラメータ `DFTI_NUMBER_OF_USER_THREADS` を 1 (デフォルト値) に設定する。

3 の技法では、複数のスレッドが同じディスクリプタを使用するため、`DftiSetValue()` 関数を使用して `DFTI_NUMBER_OF_USER_THREADS` を実際の DFT 計算スレッド数に設定しなければならない。ディスクリプタには各スレッドの個々のデータが含まれるため、この設定を行わないとプログラムが正しく実行されない。



### 警告：

1. プログラムの並列化とインテル MKL の内部スレッディングの使用を同時に行うことは、性能に悪影響を与えるため推奨されない。上記の 3 の技法では、DFT 計算は自動的にシングル・スレッディング・モードで開始される点に注意する。
2. スレッド数は、`DftiCommitDescriptor()` 関数による DFT の初期化が完了した後で変更してはならない。例えば、OMP 関数 `omp_set_max_threads()` を使用することはできない。

付録 C の [例 C-22](#)、[例 C-23](#)、および [例 C-24](#) を参照のこと。

## 入力と出力との距離

インテル MKL の DFT インターフェイスは複数回の変換計算を許可している。そのため、複数セットのデータのデータ分布をユーザが指定できる必要がある。これは、連続するデータセットの最初のデータ成分同士の距離で実現される。このパラメータは変換



回数が 1 より大きい場合は必須である。パラメータは、Fortran では Integer データ型、C では long データ型の値である。データセットは何ら共通成分は持っていない。次の例で仕様を説明する。 $x(0:31, 1)$ 、 $x(0:31, 2)$ 、 $x(0:31, 3)$  に格納された 3 つの長さ 32 の複素シーケンスの順方向 DFT を計算する場合を考える。結果は配列  $y(0:63, 3)$  の位置  $y(0:31, k)$ 、 $k=1, 2, 3$  に格納されるとする。入力距離は 32 となり、出力距離は 64 となる。計算関数のデータと結果パラメータは、すべて擬寸法階数 1 の配列 DIMENSION(0:\*) として宣言されている点に注意する。そのため、2 次元配列は EQUIVALENCE 文か Fortran のその他の機能を使用して 1 次元配列に変換しなければならない。以下にコードの一部を示す。

```
Complex :: X_2D(0:31,3), Y_2D(0:63, 3)
Complex :: X(96), Y(192)
Equivalence (X_2D, X)
Equivalence (Y_2D, Y)
.....
Status = DftiCreateDescriptor(Desc_Handle, DFTI_SINGLE,
                             DFTI_COMPLEX, 1, 32)
Status = DftiSetValue(Desc_Handle, DFTI_NUMBER_OF_TRANSFORMS, 3)
Status = DftiSetValue(Desc_Handle, DFTI_INPUT_DISTANCE, 32)
Status = DftiSetValue(Desc_Handle, DFTI_OUTPUT_DISTANCE, 64)
Status = DftiSetValue(Desc_Handle, DFTI_PLACEMENT, DFTI_NOT_INPLACE)
Status = DftiCommitDescriptor(Desc_Handle)
Status = DftiComputeForward(Desc_Handle, X, Y)
Status = DftiFreeDescriptor(Desc_Handle)
```

## ストライド

DFT インターフェイスは、データセットの複数回変換のサポートに加えて、各データセット内のデータに対して単位ストライド以外のストライド分布をサポートしている。パラメータは、Fortran では Integer データ型、C では long データ型の値の配列である。シーケンス  $x_j$ 、 $0 \leq j < 32$  において長さ 32 の DFT が計算される次のような状況を考える。これら値の実際の位置は、配列  $x(1:68)$  の  $x(5)$ 、 $x(7)$ 、...、 $x(67)$  である。DFT インターフェイスによって実現されるストライドは、データ配列の最初の成分からの変位  $L0$  (この場合 4) と、続く成分の一定の距離  $L1$  (この場合 2) で構成される。よって Fortran 配列  $x$  は次のようになる。

$$x_j = x(1 + L0 + L1 * j) = x(5 + L1 * j)$$

ストライド・ベクトル (4,2) は長さ 2、階数 1 の整数配列によって与えられる。

```

COMPLEX :: X(68)
INTEGER :: Stride(2)
.....
Status = DftiCreateDescriptor(Desc_Handle, DFTI_SINGLE,
                             DFTI_COMPLEX, 1, 32)

Stride = (/ 4, 2 /)
Status = DftiSetValue(Desc_Handle, DFTI_INPUT_STRIDES, Stride)
Status = DftiSetValue(Desc_Handle, DFTI_OUTPUT_STRIDES, Stride)
Status = DftiCommitDescriptor(Desc_Handle)
Status = DftiComputeForward(Desc_Handle, X)
Status = DftiFreeDescriptor(Desc_Handle)

```

一般に  $d$  次元変換ではストライドは長さ  $d+1$  の整数ベクトル  $(L0, L1, L2, \dots, Ld)$  で与えられ、それぞれの意味は次のとおりである。

$L0$  = 最初の配列成分からの変位

$L1$  = 最初の次元の連続するデータ成分間の距離

$L2$  = 2 番目の次元の連続するデータ成分間の距離

... = ...

$Ld$  =  $d$  番目の次元の連続するデータ成分間の距離

$d$  次元のデータ・シーケンス、

$$x_{j_1, j_2, \dots, j_d}, \quad 0 \leq j_i < J_i, \quad 1 \leq i \leq d$$

は、階数 1 の配列  $x$  に以下のマッピングによって格納される。

$$x_{j_1, j_2, \dots, j_d} = x(\text{first index} + L0 + j_1 L1 + j_2 L2 + \dots + j_d Ld)$$

複数回の変換では、値  $L0$  は最初のデータ・シーケンスに適用され、 $L_j$ 、 $j = 1, 2, \dots, d$  はすべてのデータ・シーケンスに適用される。

単一の 1 次元シーケンスの場合、 $L1$  は単純に通常のストライドとなる。一般的な多次元の状態でのストライドのデフォルト設定は、シーケンスが配列内に厳密に分布している場合に相当する。

$$L1 = 1, L2 = J1, L3 = J1J2, \dots, Ld = \prod_{i=1}^{d-1} J_i$$

入力データと出力データのそれぞれは関連するストライドを有する。デフォルトは、データが言語に自然な方法でメモリに隣接して格納されるように設定されている。上記の構成パラメータの使用例については[例 C-21](#) を参照のこと。

## 順序指定

数々の FFT アルゴリズムには、処理時間を要する明示的な置換段階が適用されていることはよく知られている [\[4\]](#)。ただし、スクランブルされた入力データに DFT を適用するか、あるいはスクランブルした DFT 結果出力を許せば、上記段階を回避できる。畳み込みとパワー・スペクトラム計算のようなアプリケーションでは、結果またはデータの順序は重要ではないため、性能向上が得られるのであればスクランブル順の許可は魅力的である。インテル MKL では次の 3 つのオプションが利用可能である。

1. DFTI\_ORDERED: 順方向変換データは順序どおり、逆方向変換データは順序どおり (デフォルト・オプション)。
2. DFTI\_BACKWARD\_SCRAMBLED: 順方向変換データは順序どおり、逆方向変換データはスクランブル。

[表 11-17](#) に、この構成設定の効果を示す。

**表 11-17      スクランブル順の変換**

	DftiComputeForward	DftiComputeBackward
DFTI_ORDERING	入力 → 出力	入力 → 出力
DFTI_ORDERED	順序どおり → 順序どおり	順序どおり → 順序どおり
DFTI_BACKWARD_SCRAMBLED	順序どおり → スクランブル	スクランブル → 順序どおり

後半の 2 つのオプションは「可能ならスクランブル順を許す」意味である点に注意する。データのスクランブルを許可しても性能向上が得られない状況もあり、そのため実装によっては示唆を無視する場合がある。厳密には、些細であるが、普通の順序もスクランブルされた順序の 1 つである。

## 転置

高次変換の結果を転置の形式で表現可能にするオプションである。デフォルト設定は DFTI\_NONE だが、DFTI\_ALLOW に設定可能である。スクランブル順と同様に、高次変換では、結果を転置形式で与えるようにすると性能が得られる場合がある。DFT インターフェイスは、性能向上が可能な場合に対応して、転置形式で結果を出力するオプションを備えている。一般的なストライド仕様は転置表現に自然に適合することから、このオ

プッシュンを利用すると、出力のストライドがユーザによって元々指定されたストライドと異なることがある。2次元結果

$$Y_{j_1, j_2}, 0 \leq j_i < n_i$$

が見込まれる例を考えてみる。はじめにユーザは、汎用ストライド  $L_1 = 1$  および  $L_2 = n_1$  を用いて結果を (フラットな) 配列  $y$  に割り当てるように指定した。転置オプションを使用すると実際の計算結果はストライド  $L_1 = n_2$  と  $L_2 = 1$  で  $y$  に返される場合がある。これらストライドは対応する問い合わせ関数で取得できる。3次元以上では、転置は次元の任意な置換を意味する点に注意する。

## クラスタ DFT 関数

このセクションでは、インテル MKL に実装されているクラスタ離散フーリエ変換 (DFT) 関数 (インテル MKL Linux 版でのみ利用可能) について説明する。

クラスタ DFT 関数ライブラリは、クラスタ (すなわち、ネットワークにより相互に連結されたコンピュータのグループ) 上で離散フーリエ変換を実行するための関数である。クラスタ上の各コンピュータ (ノード) は個別のメモリとプロセッサを持ち、ノード間のデータ交換はネットワークにより提供される。

クラスタ DFT 関数ライブラリは、MPI (Message Passing Interface) を使用して、クラスタ上のノード間のコミュニケーションを行う。利用可能な MPI 実装 (例 MPICH、インテル® MPI ほか) の数が与えられると、特定の MPI 実装における依存を回避するために、クラスタ DFT は BLACS を介して、MPI と動作する。

インテル MKL のクラスタ離散フーリエ変換関数ライブラリは、1 次元、2 次元、多次元 (最大 7 次元) のルーチン、およびすべての変換関数で Fortran インターフェイスと C インターフェイスの両方を提供する。

インテル・クラスタ MKL の DFT 関数のインターフェイスは、この章ですでに説明した通常の MKL の「[DFT 関数](#)」に対応するインターフェイスとよく似ている。詳細は該当するセクションを参照のこと。

インテル MKL に実装されているすべてのクラスタ DFT 関数を次の表に示す。

**表 11-18      インテル MKL のクラスタ DFT 関数**

関数名	演算
<b>ディスクリプタ操作関数</b>	
<a href="#">DftiCreateDescriptorDM</a>	ディスクリプタ・データ構造にメモリを割り当て、デフォルト構成設定で具体化する。
<a href="#">DftiCommitDescriptorDM</a>	実際の DFT 演算を実行可能にするすべての初期化を実行する。
<a href="#">DftiFreeDescriptorDM</a>	ディスクリプタに割り当てられていたメモリを解放する。
<b>DFT 計算関数</b>	
<a href="#">DftiComputeForwardDM</a>	順方向 DFT を計算する。
<a href="#">DftiComputeBackwardDM</a>	逆方向 DFT を計算する。

**表 11-18      インテル MKL のクラスタ DFT 関数 ( 続き )**

関数名	演算
<b>ディスクリプタ構成関数</b>	
<a href="#">DftiSetValueDM</a>	特定の 1 個の構成パラメータを指定された構成値で設定する。
<a href="#">DftiGetValueDM</a>	特定の 1 個の構成パラメータの構成値を取得する。
<b>ステータス確認関数</b>	
<a href="#">DftiErrorClass</a>	ステータスが定義済みクラスのエラーを反映しているか確認する。
<a href="#">DftiErrorMessage</a>	エラー・メッセージを生成する。

## クラスタ DFT の計算

このセクションで後述するクラスタ DFT 関数は Fortran インターフェイスと C インターフェイスで実装される。Fortran とは Fortran 95 を意味する。クラスタ DFT インターフェイス関数を使用するいくつかのコード例を以下に示す。

### 例 11-1      2次元 クラスタ DFT (Fortran インターフェイス)

```
!Fortran example.
!Two-dimensional complex to complex
use MKL_DFT_DM_TYPE
use MKL_DFTI_DM

INTEGER :: nProc, nRank, IERROR

COMPLEX(8) :: p_2D(16, 32)
COMPLEX(8) :: p(512)
COMPLEX(8), ALLOCATABLE :: pp(:)
Equivalence (p_2D, p)

!sizes(0) = 16, sizes(1) = 32
INTEGER :: sizes(0:1)

TYPE(DFTI_DESCRIPTOR_DM), POINTER :: hdm

!Fill the array p_2D here...

sizes(0)=16
sizes(1)=32

!Initialization of MPI
CALL MPI_INIT( IERROR )
```

---

**例 11-1      2 次元 クラスタ DFT (Fortran インターフェイス) ( 続き )**

---

```
!Obtain number of processors
CALL MPI_Comm_size(MPI_COMM_WORLD, nProc, IERROR)

!Obtain current rank
CALL MPI_Comm_rank(MPI_COMM_WORLD, nRank, IERROR)

pp_size = ((nRank+1)*sizes(0)/nProc-nRank*sizes(0)/nProc)*sizes(1);
ALLOCATE(pp(0:pp_size-1))
pp(0:pp_size-1) =
p(nRank*sizes(0)/nProc*sizes(1):(nRank+1)*sizes(0)/nProc*sizes(1)-1)

!Create
DftiCreateDescriptorDM(MPI_COMM_WORLD, phdm, DFTI_DOUBLE, DFTI_COMPLEX,
dim, sizes)

!Commit
DftiCommitDescriptorDM(phdm)

!Compute
DftiComputeForwardDM(phdm, pp, pp)

!Free
DftiFreeDescriptorDM(phdm)

DEALLOCATE(pp)

!Finalization of MPI
CALL MPI_Finalize(IERROR)
```

---



---

**例 11-2      2次元 クラスタ DFT (C インターフェイス)**

---

```
/*
C example
Two-dimensional complex to complex
*/
#include <stdlib.h>
#include "mpi.h"

#include "mkl_dfti_dm.h"

int nProc, nRank, i;
complex *p;
DFTI_DESCRIPTOR_DM_HANDLE hdm;
long sizes[2];

//Initialize the array "sizes" here...

//Initialization of MPI
MPI_Init(&argc, &argv);

//Obtain number of processors
MPI_Comm_size(MPI_COMM_WORLD, &nProc);

//Obtain current rank
MPI_Comm_rank(MPI_COMM_WORLD, &nRank);

//Allocating memory
p = (complex*)malloc(sizes[0]*sizes[1]*sizeof(complex));

//Fill the array "p" here...
```

---

## 例 11-2 2 次元 クラスタ DFT (C インターフェイス) (続き)

```
DftiCreateDescriptorDM(MPI_COMM_WORLD, &hdm, DFTI_TYPE, DFTI_COMPLEX,
dim, sizes);

DftiCommitDescriptorDM(hdm);

DftiComputeForwardDM(hdm, &p[nRank*sizes[0]/nProc*sizes[1]],
&p[nRank*sizes[0]/nProc*sizes[1]]);

DftiFreeDescriptorDM(&hdm);

MPI_Finalize();
```

## クラスタ DFT インターフェイス

クラスタ DFT 関数を使用するには、Fortran では "use" 構文を使用して MKL\_DFTI\_DM モジュールにアクセスする必要がある。C では "include" 構文を使用してヘッダファイル `dfti_dm.h` にアクセスする必要がある。

Fortran インターフェイスでは、派生型 `DFTI_DESCRIPTOR_DM`、さまざまな構成パラメータの名称とそれらの取り得る値を表す名前付き定数、Fortran 95 の汎用機能性を介したさまざまなオーバーロード関数が利用できる。

C インターフェイスでは、構造型 `DFTI_DESCRIPTOR_DM_HANDLE` や、関数によっては異なる個数の入力引数を受け付ける種々の関数が利用できる。

インテル MKL では クラスタ DFT 関数を 4 種類の大きなカテゴリに分けている。

1. **ディスクリプタ操作。** このカテゴリは 3 つの関数で構成される。1 つ目の [DftiCreateDescriptorDM](#) は DFT ディスクリプタを生成しストレージを動的に割り当てる関数である。2 つ目の [DftiCommitDescriptorDM](#) はディスクリプタを設定にもとづいて「遂行」させる関数である。3 つ目の [DftiFreeDescriptorDM](#) はディスクリプタ情報に割り当てられていたすべてのメモリを解放する関数である。
2. **DFT 計算。** このカテゴリは 2 つの関数で構成される。1 つ目の [DftiComputeForwardDM](#) は順方向 DFT 計算を実行する関数で、2 つ目の [DftiComputeBackwardDM](#) は逆方向 DFT 計算を実行する関数である。
3. **ディスクリプタ構成。** このカテゴリは 2 つの関数で構成される。1 つ目の関数 [DftiSetValueDM](#) は、特定の 1 個の構成値を、多くの構成パラメータの 1 つに設定する。2 つ目の関数 [DftiGetValueDM](#) は、これら構成パラメータの任意の 1 つの現在値を取得する (すべてが読み出し可能)。読み出し可能なパラメータ数が多いが、扱えるのは 1 回の関数呼び出しにつき 1 つである。

4. **ステータス確認。**上記の3種類のカテゴリに記述されている各関数は演算のステータスを表す整数値を返す。  
 戻り値が非ゼロの場合はある種の問題が起きたことを意味する。インテル MKL では将来のリリースにおける拡張を想定して、現在の DFT インターフェイスは単一の論理ステータス・クラス関数 [DftiErrorClass](#) および単純なステータス・メッセージ生成関数 [DftiErrorMessage](#) のみを提供している。

## ディスクリプタ操作

このカテゴリは、ディスクリプタの生成、ディスクリプタの遂行、ディスクリプタの解放、の3つの関数で構成される。

---

## CreateDescriptorDM

ディスクリプタ・データ構造にメモリを割り当て、デフォルト構成設定で具体化する。

---

### 構文

```
!Fortran
Status = DftiCreateDescriptorDM(comm, handle, v1, v2, dim, size)
Status = DftiCreateDescriptorDM(comm, handle, v1, v2, dim, sizes)
/* C/C++ */
status = DftiCreateDescriptorDM(comm, phandle, v1, v2, dim, sizes);
```

### 入力パラメータ

<i>comm</i>	MPI コミュニケータ。MPI_COMM_WORLD など。
<i>v1</i>	精度。
<i>v2</i>	順方向領域のタイプ。現在のバージョンでは DFTI_COMPLEX でなければならない。
<i>dim</i>	変換の次元。
<i>size</i>	1 次元の場合の変換の長さ。
<i>sizes</i>	多次元の場合の変換の長さ。

## 出力パラメータ

*handle*      変換ハンドルのポインタ。この関数は正常に終了すると、作成されたハンドルへのポインタが変数に格納される。

## 説明

この関数はディスクリプタ・データ構造にメモリを割り当て、変換対象の精度、領域、次元、長さをそれぞれのデフォルト構成設定を使って具体化する。領域は順方向変換での領域として解釈される。メモリは動的に割り当てられるため、出力結果として生成されたディスクリプタのポインタが得られる。この関数は、DFT を計算する従来型の多くのソフトウェア・パッケージやライブラリに見られる「初期化」ルーチンとはやや異なっている。ユーザの要求によって値設定関数 [DftiSetValueDM](#) を通してデフォルト構成設定を変更できることから、この関数は回転因子計算のような計算量の多い計算はおそらく実行しない。

精度は、インターフェイスで構成値に与えられる名前付き定数によって指定される。精度の選択肢は DFTI\_SINGLE か DFTI\_DOUBLE である。入力データ、出力データ、および計算の精度に対応する。設定が DFTI\_SINGLE の場合は単精度浮動小数点データ型が有効で、設定が DFTI\_DOUBLE の場合は倍精度浮動小数点データ型が有効となる。

次元は変換の次元を表す単純な正の整数である。C/C++ では、長さは変換の長さへのポインタ (long 型の整数値または long 型の整数配列) である。Fortran では、長さは、1 次元変換では正の整数、多次元変換では長さの配列である。

## 戻り値

この関数は正常に終了すると DFTI\_NO\_ERROR を返す。この場合、作成されたハンドルへのポインタは *handle* に格納される。関数が異常終了すると、戻り値にはエラーコードが格納される。エラーの拡張情報を取得するには、[DftiErrorClass](#) 関数と [DftiErrorMessage](#) 関数を呼び出す。

## インターフェイスとプロトタイプ

!Fortran Interface

INTERFACE DftiComputeForwardDM

INTEGER(4) FUNCTION DftiComputeForwardDM(h, in\_X, out\_X)

!DEC\$ATTRIBUTES C, DECORATE, ALIAS : "DftiComputeForwardDM\_fortran"  
:: DftiComputeForwardDM

!DEC\$ATTRIBUTES VALUE :: h

USE MKL\_DFT\_DM\_TYPE

```

        TYPE(DFTI_DESCRIPTOR_DM), POINTER :: h
        COMPLEX(8), DIMENSION(*) :: in_x, out_X
    END FUNCTION DftiComputeForwardDM
    INTEGER(4) FUNCTION DftiComputeForwardDMs(h, in_X, out_X)
        !DEC$ATTRIBUTES C, DECORATE, ALIAS : "DftiComputeForwardDM_fortran"
        :: DftiComputeForwardDMs
        !DEC$ATTRIBUTES VALUE :: h
        USE MKL_DFT_DM_TYPE
        TYPE(DFTI_DESCRIPTOR_DM), POINTER :: h
        COMPLEX(4), DIMENSION(*) :: in_x, out_X
    END FUNCTION DftiComputeForwardDMs
    END INTERFACE
    /* C/C++ prototype */
    long DftiCreateDescriptorDM(MPI_Comm comm, DFTI_DESCRIPTOR_DM_HANDLE
    *phandle, enum DFTI_CONFIG_VALUE v1, enum DFTI_CONFIG_VALUE v2, long dim,
    long *sizes);

```

---

## CommitDescriptorDM

実際のDFT演算を実行可能にするすべての初期化を実行する。

---

### 構文

```

!Fortran
Status = DftiCommitDescriptorDM(handle)
/* C/C++ */
status = DftiCommitDescriptorDM(handle);

```

### 入力パラメータ

*handle* DftiCreateDescriptorDM から取得される有効なハンドル。

## 説明

DFT インターフェイスでは、ディスクリプタを使って DFT 計算を実行する前に、生成されたディスクリプタを遂行する関数を呼び出す必要がある。

DftiCommitDescriptorDM 関数は実際の DFT 演算を実行可能にするすべての初期化を実行する。近代的な実装では、高度に効率化された計算方法を検索するために、さまざまな長さの入力を用いた因子分解を行う調査が含まれる場合がある。

遂行されたディスクリプタの構成パラメータを値設定関数(「[ディスクリプタの構成](#)」を参照)を介して変更した場合は、計算関数を呼び出す前にディスクリプタの再遂行を必要とする。通常は、この遂行関数コールのすぐあとに計算関数コールを続ける(「[DFT 計算](#)」を参照)。

## 戻り値

この関数は正常に終了すると DFTI\_NO\_ERROR を返す。関数が異常終了すると、戻り値にはエラーコードが格納される。エラーの拡張情報を取得するには、[DftiErrorClass](#) 関数および [DftiErrorMessage](#) 関数を呼び出す。

## インターフェイスとプロトタイプ

!Fortran Interface

INTERFACE DftiCommitDescriptorDM

INTEGER(4) FUNCTION DftiCommitDescriptorDM(handle);

!DEC\$ATTRIBUTES C, DECORATE, ALIAS : "DftiCommitDescriptorDM\_fortran"  
:: DftiCommitDescriptorDM

!DEC\$ATTRIBUTES VALUE :: handle

USE MKL\_DFT\_DM\_TYPE

TYPE(DFTI\_DESCRIPTOR\_DM), POINTER :: handle

END FUNCTION

END INTERFACE

/\* C/C++ prototype \*/

long DftiCommitDescriptorDM(DFTI\_DESCRIPTOR\_DM\_HANDLE handle);

---

## FreeDescriptorDM

ディスクリプタに割り当てられていたメモリを解放する。

---

### 構文

```
!Fortran
Status = DftiFreeDescriptorDM(handle)
/* C/C++ */
status = DftiFreeDescriptorDM(handle);
```

### 入力パラメータ

*handle* DftiCreateDescriptorDM から取得される有効なハンドル。

### 説明

この関数はディスクリプタに割り当てられているすべてのメモリを解放する。  
DftiFreeDescriptorDM 関数を呼び出してディスクリプタ・ハンドルを削除する。  
DftiFreeDescriptorDM の使用後に、ディスクリプタ・ハンドルは無効になる。

### 戻り値

この関数は正常に終了すると DFTI\_NO\_ERROR を返す。関数が異常終了すると、戻り値にはエラーコードが格納される。エラーの拡張情報を取得するには、[DftiErrorClass](#) 関数および [DftiErrorMessage](#) 関数を呼び出す。

### インターフェイスとプロトタイプ

```
!Fortran Interface
INTERFACE DftiFreeDescriptorDM
  INTEGER(4) FUNCTION DftiFreeDescriptorDM(handle)
    !DEC$ATTRIBUTES C, DECORATE, ALIAS : "DftiFreeDescriptorDM_fortran"
    :: DftiFreeDescriptorDM
    !DEC$ATTRIBUTES VALUE :: handle
    USE MKL_DFT_DM_TYPE
    TYPE(DFTI_DESCRIPTOR_DM), POINTER :: handle
  END FUNCTION
```

```
END INTERFACE
```

```
/* C/C++ prototype */  
long DftiFreeDescriptorDM(DFTI_DESCRIPTOR_DM_HANDLE handle);
```

## DFT 計算

このカテゴリは、順方向変換の計算、逆方向変換の計算の 2 つの関数で構成される。

---

## ComputeForwardDM

順方向離散フーリエ変換を計算する。

---

### 構文

```
!Fortran  
Status = DftiComputeForwardDM(handle, in_X, out_X)  
Status = DftiComputeForwardDM(handle, in_Xs, out_Xs)  
/* C/C++ */  
status = DftiComputeForwardDM(handle, in_X, out_X);
```

### 入力パラメータ

*handle*     有効なディスクリプタ・ハンドル。  
*in\_X*        入力データのローカル部分。  
*in\_Xs*       入力データのローカル部分 ( 単精度 )。

### 出力パラメータ

*out\_X*       出力データのローカル部分。  
*out\_Xs*      出力データのローカル部分 ( 単精度 )。

### 説明

実際の DFT 計算はディスクリプタの生成と遂行を完了した後に実行する。  
DftiComputeForwardDM 関数は順方向 DFT を計算する。



これは、因子  $e^{-i2\pi/n}$  を使用した変換である。構成の自由度が高いため、入力データを複数の方法で表すことができ、また出力結果を複数の方法で配置できる。入力パラメータの個数とタイプは多様である。このような多様性は Fortran 95 の一般関数機能によって実現される。データと結果パラメータは、すべて擬寸法階数 1 の配列 `DIMENSION(0:*)` として宣言される。

有効なディスクリプタ・ハンドルは [DftiCreateDescriptorDM](#) によって作成され、[DftiCommitDescriptorDM](#) によって遂行される。ディスクリプタには、変換精度を正確に決めるために必要な充分な情報が含まれている。実際の実装では、この情報を使って入力との矛盾を確認する場合がある。

入出力データは、プロセス間で分配される複素値のシーケンス (それぞれの複素値は、実数部と虚数部の 2 つの実数で構成される) である。例えば、配列 `sizes[] = {length_1, length_2}` を [DftiComputeForwardDM](#) のパラメータとして使用した場合、グローバル配列 `p` の入力のローカル部分 `local_p` は次の式によって定義される。

$$local\_p[i] = \{p[i], \\ nRank * sizes[0] / nProc * sizes[1] < i < (nRank + 1) * sizes[0] / nProc * sizes[1]\}$$

ここで、`nRank` はプロセスのランクで、`nProc` はプロセスの数である。

入力データと出力データの精度の選択は、変換精度と同じである。設定が `DFTI_SINGLE` の場合は単精度浮動小数点データ型が有効で、設定が `DFTI_DOUBLE` の場合は倍精度浮動小数点データ型が有効となる。

## 戻り値

この関数は正常に終了すると `DFTI_NO_ERROR` を返す。関数が異常終了すると、戻り値にはエラーコードが格納される。エラーの拡張情報を取得するには、[DftiErrorClass](#) 関数および [DftiErrorMessage](#) 関数を呼び出す。

## インターフェイスとプロトタイプ

```
!Fortran Interface
INTERFACE DftiComputeForwardDM
INTEGER(4) FUNCTION DftiComputeForwardDM(h, in_X, out_X)
    !DEC$ATTRIBUTES C, DECORATE, ALIAS : "DftiComputeForwardDM_fortran"
    :: DftiComputeForwardDM
    !DEC$ATTRIBUTES VALUE :: h
    USE MKL_DFT_DM_TYPE
    TYPE(DFTI_DESCRIPTOR_DM), POINTER :: h
```

```
COMPLEX(8), DIMENSION(*) :: in_x, out_X
END FUNCTION DftiComputeForwardDM
INTEGER(4) FUNCTION DftiComputeForwardDMs(h, in_X, out_X)
    !DEC$ATTRIBUTES C, DECORATE, ALIAS : "DftiComputeForwardDM_fortran"
    :: DftiComputeForwardDMs
    !DEC$ATTRIBUTES VALUE :: h
    USE MKL_DFT_DM_TYPE
    TYPE(DFTI_DESCRIPTOR_DM), POINTER :: h
    COMPLEX(4), DIMENSION(*) :: in_x, out_X
END FUNCTION DftiComputeForwardDMs
END INTERFACE

/* C/C++ prototype */
long DftiComputeForwardDM(DFTI_DESCRIPTOR_DM_HANDLE handle, void *in_X,
void *out_X);
```

---

## ComputeBackwardDM

逆方向離散フーリエ変換を計算する。

---

### 構文

```
!Fortran
Status = DftiComputeBackwardDM(handle, in_X, out_X)
Status = DftiComputeBackwardDM(handle, in_Xs, out_Xs)
/* C/C++ */
status = DftiComputeBackwardDM(handle, in_X, out_X);
```

### 入力パラメータ

*handle*     有効なディスクリプタ・ハンドル。

*in\_X*        入力データのローカル部分。

*in\_Xs*       入力データのローカル部分 ( 単精度 )。

## 出力パラメータ

`out_x` 出力データのローカル部分。

`out_xs` 出力データのローカル部分 (単精度)。

## 説明

実際の DFT 計算はディスクリプタの生成と遂行を完了した後に実行する。

`DftiComputeBackwardDM` 関数は逆方向 DFT を計算する。

これは、因子  $e^{i2\pi/n}$  を使用した変換である。構成の自由度が高いため、入力データを複数の方法で表すことができ、また出力結果を複数の方法で配置できる。そのため、入力パラメータの個数とタイプは多様である。このような多様性は Fortran 95 の一般関数機能によって実現される。データと結果パラメータは、すべて擬寸法階数 1 の配列 `DIMENSION(0:*)` として宣言される。

有効なディスクリプタ・ハンドルは `DftiCreateDescriptorDM` によって作成され、`DftiCommitDescriptorDM` によって遂行される。ディスクリプタには、変換精度を正確に決めるために必要な充分な情報が含まれている。実際の実装では、この情報を使って入力の矛盾を確認する場合がある。

入出力データは、プロセス間で分配される複素値のシーケンス (それぞれの複素値は、実数部と虚数部の 2 つの実数で構成される) である。例えば、配列 `sizes[] = {length_1, length_2}` を `DftiComputeBackwardDM` のパラメータとして使用した場合、グローバル配列 `p` の入力のローカル部分 `local_p` は次の式によって定義される。

```
local_p[] = {p[i],
nRank*sizes[0]/nProc*sizes[1]<i<(nRank+1)*sizes[0]/nProc*sizes[1]}
```

ここで、`nRank` はプロセスのランクで、`nProc` はプロセスの数である。

入力データと出力データの精度の選択は、変換精度と同じである。設定が `DFTI_SINGLE` の場合は単精度浮動小数点データ型が有効で、設定が `DFTI_DOUBLE` の場合は倍精度浮動小数点データ型が有効となる。

## 戻り値

この関数は正常に終了すると `DFTI_NO_ERROR` を返す。関数が異常終了すると、戻り値にはエラーコードが格納される。エラーの拡張情報を取得するには、`DftiErrorClass` 関数および `DftiErrorMessage` 関数を呼び出す。

## インターフェイスとプロトタイプ

```
!Fortran Interface
INTERFACE DftiComputeBackwardDM
INTEGER(4) FUNCTION DftiComputeBackwardDM(h, in_X, out_X)
    !DEC$ATTRIBUTES C, DECORATE, ALIAS : "DftiComputeBackwardDM_fortran"
    :: DftiComputeBackwardDM
    !DEC$ATTRIBUTES VALUE :: h
    USE MKL_DFT_DM_TYPE
    TYPE(DFTI_DESCRIPTOR_DM), POINTER :: h
    COMPLEX(8), DIMENSION(*) :: in_x, out_X
END FUNCTION DftiComputeBackwardDM
INTEGER(4) FUNCTION DftiComputeBackwardDMs(h, in_X, out_X)
    !DEC$ATTRIBUTES C, DECORATE, ALIAS : "DftiComputeBackwardDM_fortran"
    :: DftiComputeBackwardDMs
    !DEC$ATTRIBUTES VALUE :: h
    USE MKL_DFT_DM_TYPE
    TYPE(DFTI_DESCRIPTOR_DM), POINTER :: h
    COMPLEX(4), DIMENSION(*) :: in_x, out_X
END FUNCTION DftiComputeBackwardDMs
END INTERFACE

/* C/C++ prototype */
long DftiComputeBackwardDM(DFTI_DESCRIPTOR_DM_HANDLE handle, void *in_X,
void *out_X);
```

## ディスクリプタの構成

このカテゴリは、特定の 1 個の構成パラメータに適切な値を設定する値設定関数 `DftiSetValue` と、特定の 1 個の構成パラメータ値を読み出す値取得関数 `DftiGetValue` の 2 つで構成される。すべての構成パラメータは読み出し可能であるが、一部はユーザが設定できない。構成パラメータには、バージョン番号、動的な情報など、実装に固有な固定情報が含まれているが、固定情報は関数の実行時の実装によって求められる。詳細は「[構成設定](#)」を参照のこと。

## SetValueDM

特定の 1 個の構成パラメータを指定された構成値で設定する。

### 構文

```
!Fortran
Status = DftiSetValueDM(handle, param, ivalue)
Status = DftiSetValueDM(handle, param, rvalue)
/* C/C++ */
status = DftiSetValueDM(handle, param, ivalue);
status = DftiSetValueDM(handle, param, rvalue);
```

### 入力パラメータ

**handle** 有効なディスクリプタ・ハンドル。

**param** ディスクリプタ・ハンドルに設定されるパラメータ名。利用できる名前の一覧は[表 11-19](#)を参照のこと。

**ivalue** パラメータの Integer 値。

**rvalue** パラメータの Double 値。

### 説明

この関数は特定の 1 個の構成パラメータを指定された構成値で設定する。構成パラメータは以下の表に示される名前付き定数のうちの 1 つであり、構成値は、適切なタイプによって設定される。設定の詳細は「[構成設定](#)」を参照のこと。

表 11-19 設定可能な構成パラメータ

名前付き定数	パラメータのタイプ	説明
DFTI_FORWARD_SCALE	double	順方向変換の倍率因子
DFTI_BACKWARD_SCALE	double	逆方向変換の倍率因子
DFTI_WORKSPACE	long	計算で補助バッファを使用するかどうかを示す。

## 戻り値

この関数は正常に終了すると `DFTI_NO_ERROR` を返す。関数が異常終了すると、戻り値にはエラーコードが格納される。エラーの拡張情報を取得するには、[DftiErrorClass](#) 関数および [DftiErrorMessage](#) 関数を呼び出す。

## インターフェイスとプロトタイプ

```
!Fortran Interface
INTERFACE DftiSetValueDM
INTEGER(4) FUNCTION DftiSetValueDM(h, p, v)
    !DEC$ATTRIBUTES C, DECORATE, ALIAS : "DftiSetValueDM_fortran" ::
DftiSetValueDM
    !DEC$ATTRIBUTES VALUE :: h, p, v
    USE MKL_DFT_DM_TYPE
    TYPE(DFTI_DESCRIPTOR_DM), POINTER :: h
    INTEGER(4) :: p, v
END FUNCTION
INTEGER(4) FUNCTION DftiSetValueDMd(h, p, v)
    !DEC$ATTRIBUTES C, DECORATE, ALIAS : "DftiSetValueDM_fortran" ::
DftiSetValueDMd
    !DEC$ATTRIBUTES VALUE :: h, p, v
    USE MKL_DFT_DM_TYPE
    TYPE(DFTI_DESCRIPTOR_DM), POINTER :: h
    INTEGER(4) :: p
    REAL(8) :: v
END FUNCTION
END INTERFACE

/* C/C++ prototype */
long DftiSetValueDM(DFTI_DESCRIPTOR_DM_HANDLE handle, enum
DFTI_CONFIG_PARAM param, long ivalue);
long DftiSetValueDM(DFTI_DESCRIPTOR_DM_HANDLE handle, enum
DFTI_CONFIG_PARAM param, double rvalue);
```

---

## GetValueDM

特定の 1 個の構成パラメータの構成値を取得する。

---

### 構文

```
!Fortran
Status = DftiGetValueDM(handle, param, ivalue)
Status = DftiGetValueDM(handle, param, rvalue)
/* C/C++ */
status = DftiGetValueDM(handle, param, pivalue);
status = DftiGetValueDM(handle, param, prvalue);
```

### 入力パラメータ

*handle* 有効なディスクリプタ・ハンドル。

*param* ディスクリプタ・ハンドルから取得されるパラメータ名。利用できる名前の一覧は[表 11-20](#)を参照のこと。

*ivalue* パラメータの Integer 値。

*rvalue* パラメータの Double 値。

### 出力パラメータ

*pivalue* パラメータの Integer 値が格納されるバッファへのポインタ。

*prvalue* パラメータの Double 値が格納されるバッファへのポインタ。

### 説明

この関数は特定の 1 個の構成パラメータを取得する。構成パラメータは以下の表に示される名前付き定数の中の 1 つであり、構成値は対応する適切なタイプ (名前付き定数またはネイティブ・タイプ) である。

表 11-20 設定可能な構成パラメータ

名前付き定数	パラメータのタイプ	説明
DFTI_PRECISION	long	計算精度、入力データおよび出力データ。
DFTI_DIMENSION	long	変換の次元
DFTI_LENGTHS	long[n]	変換の長さの配列 変換の次元に対応する長さ
DFTI_FORWARD_SCALE	double	順方向変換の倍率因子
DFTI_BACKWARD_SCALE	double	逆方向変換の倍率因子
DFTI_WORKSPACE	long	計算で補助バッファを使用するかどうかを示す。

## 戻り値

この関数は正常に終了すると DFTI\_NO\_ERROR を返す。関数が異常終了すると、戻り値にはエラーコードが格納される。エラーの拡張情報を取得するには、[DftiErrorClass](#) 関数および [DftiErrorMessage](#) 関数を呼び出す。

## インターフェイスとプロトタイプ

```
!Fortran Interface
INTERFACE DftiGetValueDM
INTEGER(4) FUNCTION DftiGetValueDM(h, p, v)
    !DEC$ATTRIBUTES C, DECORATE, ALIAS : "DftiGetValueDM_fortran" ::
    DftiGetValueDM
    !DEC$ATTRIBUTES VALUE :: h, p
    !DEC$ATTRIBUTES REFERENCE :: v
    USE MKL_DFT_DM_TYPE
    TYPE(DFTI_DESCRIPTOR_DM), POINTER :: h
    INTEGER(4) :: p, v
END FUNCTION
INTEGER(4) FUNCTION DftiGetValueDMd(h, p, v)
    !DEC$ATTRIBUTES C, DECORATE, ALIAS : "DftiGetValueDM_fortran" ::
    DftiGetValueDMd
    !DEC$ATTRIBUTES VALUE :: h, p
    !DEC$ATTRIBUTES REFERENCE :: v
    USE MKL_DFT_DM_TYPE
    TYPE(DFTI_DESCRIPTOR_DM), POINTER :: h
```



```
INTEGER(4) :: p
REAL(8) :: v
END FUNCTION
END INTERFACE

/* C/C++ prototype */
long DftiGetValueDM(DFTI_DESCRIPTOR_DM_HANDLE handle, enum
DFTI_CONFIG_PARAM param, long * pival);
```

## ステータス確認関数

ディスクリプタ操作、クラスタ DFT 計算、ディスクリプタ構成のカテゴリに含まれる各関数は、演算のステータスを示す整数値を返す。ここで述べる 2 つの関数は、このステータスを確認する関数である。1 つ目の関数は論理関数で定義済みクラスのエラーをステータスが反映しているか確認し、2 つ目の関数はエラー・メッセージ関数で文字列を返す。これらは、[DftiErrorClass](#) と [DftiErrorMessage](#) である。

## 高速フーリエ変換 ( 非推奨 )

FFT ルーチンは 2 の累乗長さの変換に対応し、以前のバージョンのライブラリとの互換性をサポートする。



**注：**このセクションで説明する FFT 関数はすでに置き換えられており、以前のバージョンをサポートすることだけを目的としてライブラリに残されている。これらの FFT 関数は、この章ですでに説明した DFT 関数と同じレベルの性能または機能を提供するものではなく、新規の開発では使用すべきでない。「[DFT 関数](#)」を代わりに使用すること。

このセクションは、以下のトピックで構成される。

- 「[1 次元 FFT](#)」
- 「[2 次元 FFT](#)」

各トピックでは、FFT を 3 つのグループに分けて説明する。

### 1 次元 FFT

1 次元 FFT は、以下のグループに分けられる。

- [複素数から複素数への変換](#)
- [実数から複素数への変換](#)
- [複素数から実数への変換](#)

すべての 1 次元 FFT は、インプレースである。変換の長さは 2 の累乗でなければならない。複素数から複素数への変換ルーチンは、複素ベクトルの順方向と逆方向の変換を実行する。実数から複素数への変換ルーチンは、実ベクトルの順方向の変換を実行する。複素数から実数への変換ルーチンは、実数配列で圧縮された複素共役対称ベクトルの逆方向の変換を実行する。

### データ格納の型

FFT の各グループは、同じような機能を持つ 2 つのセットの FFT で構成される。1 つのセットは Fortran インターフェイスで使用し、もう 1 つのセットは C インターフェイスで使用する。Fortran インターフェイス向けの FFT は、複素数データを FORTRAN の複素数データ型として格納する。C インターフェイス向けの FFT は、複素数データを実数

部と虚数部の浮動小数点配列として別々に格納する。これらのセットは、各セット内の FFT の名前で区別される。C インターフェイスで使用する FFT の名前には、FFT の Fortran 名の最後に文字“c” が付けられている。例えば、cfft1d/zfft1d FFT に対応する C インターフェイス・ルーチンの名前は、cfft1dc/zfft1dc となる。C タイプのデータ項目の名前はすべて、小文字である。

[表 11-21](#) に、1 次元 FFT ルーチンのグループと、それらに関連するデータ型を示す。

**表 11-21 1 次元 FFT: 名前およびデータ型**

グループ	FORTTRAN の複素数 データとし て格納	C の実数 データとし て格納	データ型	説明
複素数から 複素数	<a href="#">cfft1d/ zfft1d</a>	<a href="#">cfft1dc/ zfft1dc</a>	c, z	複素数データから複素数データへの変換を実行する。
実数から 複素数へ	<a href="#">scfft1d/ dzfft1d</a>	<a href="#">scfft1dc/ dzfft1dc</a>	sc, dz	実数データから複素数データへの順方向の変換を実行する。 scfft1d/dzfft1d と scfft1dc/dzfft1dc の FFT を補足する。
複素数から 実数へ	<a href="#">csfft1d/ zdfft1d</a>	<a href="#">csfft1dc/ zdfft1dc</a>	cs, zd	複素数データから実数データへの逆方向の変換を実行する。 csfft1d/zdfft1d と csfft1dc/zdfft1dc の FFT を補足する。

## データ構造の要件

C インターフェイスでは、複素数から複素数への変換ルーチンでは、実数部と虚数部それぞれに独立した float 配列が必要である。実数から複素数、複素数から実数への変換ルーチンでは、単一の float 入力 / 出力配列が必要である。

C インターフェイスでは、値で渡されるスカラ値が必要である。

すべての変換は、変換係数を格納するためのメモリが余分に必要になる。同次元の複数の FFT を実行する場合は、係数の表を一度だけ作成し、それ以後はすべての FFT でこの表を使用するとよい。FFT ごとに表を繰り返し作成するのではなく同じ表を使用すれば、性能が大幅に向上する。

## 複素数から複素数への 1 次元 FFT

複素数から複素数への各ルーチンでは、複素ベクトルの順方向または逆方向の FFT を計算する。

順方向の FFT は、次の式に従って計算される。

$$z_j = \sum_{k=0}^{n-1} r_k \cdot w^{-j \cdot k}, \quad 0 \leq j \leq n-1$$

逆方向の FFT は、次の式に従って計算される。

$$r_j = \frac{1}{n} \sum_{k=0}^{n-1} z_k \cdot w^{j \cdot k}, \quad 0 \leq j \leq n-1$$

$w = \exp\left[\frac{2\pi i}{n}\right]$  で、 $i$  は虚数単位である。

複素数から複素数へのルーチンが実行する演算は、各ルーチンが使用する *isign* パラメータの値によって決まる。

*isign* = -1 の場合、順方向の FFT を実行する。入力と出力は通常の順序である。

*isign* = +1 の場合、逆方向の FFT を実行する。入力と出力は通常の順序である。

*isign* = -2 の場合、順方向の FFT を実行する。入力は通常の順序、出力はビット反転した順序である。

*isign* = +2 の場合、逆方向の FFT を実行する。入力はビット反転した順序、出力は通常の順序である。

*isign* = 0 の場合、順方向の FFT と逆方向の FFT のいずれも、FFT 係数を初期化する。

上記の式は、[表 11-21](#) に示したすべてのデータ型を使用するすべての FFT に適用される。

所定の長さを持つ順方向または逆方向の FFT を計算するには、まず、*isign* = 0 に設定して関数を呼び出し、係数を初期化しなければならない。係数の初期化後は、*isign* = +1, -1, +2, -2 に設定してこの関数を呼び出し、同じ長さの変換を何度でも計算できる。

---

## cfft1d/zfft1d (非推奨)

Fortran インターフェイス・ルーチン。複素ベクトルの順方向または逆方向の FFT を計算する (インプレース)。

---

### 構文

```
call cfft1d( r, n, isign, wsave )  
call zfft1d( r, n, isign, wsave )
```

### 説明

cfft1d/zfft1d ルーチンが実行する演算は、*isign* の値によって決まる。上記の「[複素数から複素数への 1 次元 FFT](#)」の演算式を参照。

### 入力パラメータ

<i>r</i>	COMPLEX (cfft1d の場合) DOUBLE COMPLEX (zfft1d の場合) 配列、次元は ( <i>n</i> ) 以上。変換の対象となる複素ベクトルを格納する。 <i>isign</i> = 0 の場合は、参照されない。
<i>n</i>	INTEGER。変換の長さ。 <i>n</i> は、2 の累乗でなければならない。
<i>isign</i>	INTEGER。実行する演算のタイプを指示するフラグ。 <i>isign</i> = 0 の場合は、係数 <i>wsave</i> が初期化される。 <i>isign</i> = -1 の場合は、順方向の FFT を実行する。入力と出力は通常の順序である。 <i>isign</i> = +1 の場合は、逆方向の FFT を実行する。入力と出力は通常の順序である。 <i>isign</i> = -2 の場合は、順方向の FFT を実行する。入力は通常の順序、出力はビット反転した順序である。 <i>isign</i> = +2 の場合は、逆方向の FFT を実行する。入力はビット反転した順序、出力は通常の順序である。
<i>wsave</i>	COMPLEX (cfft1d の場合) DOUBLE COMPLEX (zfft1d の場合) 配列、次元は $((3*n)/2)$ 以上。 <i>isign</i> = 0 の場合、 <i>wsave</i> は出力パラメータになる。それ以外の場合、 <i>wsave</i> には、 <i>isign</i> = 0 と指定してこのルーチン呼び出したときに初期化された FFT 係数が格納される。

## 出力パラメータ

- r*            *isign* に従って、変換の結果得られた複素数データが格納される。  
*isign* = 0 の場合は、変わらない。
- wsave*       *isign* = 0 の場合、*wsave* には、初期化された FFT の係数が格納される。それ以外の場合、*wsave* は変わらない。

---

## cfft1dc/zfft1dc    ( 非推奨 )

C インターフェイス・ルーチン。複素ベクトルの順方向または逆方向の FFT を計算する ( インプレース )。

---

### 構文

```
void cfft1dc(float* r, float* i, int n, int isign, float* wsave)
void zfft1dc(double* r, double* i, int n, int isign, double* wsave)
```

### 説明

cfft1dc/zfft1dc ルーチンが実行する演算は、*isign* の値によって決まる。上記の「[複素数から複素数への 1 次元 FFT](#)」の演算式を参照。

## 入力パラメータ

- r*            float\* (cfft1dc の場合 )  
             double\* (zfft1dc の場合 ) (*n*) 以上のサイズの配列へのポインタ。変換の対象となる複素ベクトルの実数部を格納する。*isign* = 0 の場合は、参照されない。
- i*            float\* (cfft1dc の場合 )  
             double\* (zfft1dc の場合 )  
  
             (*n*) 以上のサイズの配列へのポインタ。変換の対象となる複素ベクトルの虚数部を格納する。  
  
             *isign* = 0 の場合は、参照されない。
- n*            int。変換の長さ。*n* は、2 の累乗でなければならない。

*isign* int。実行する演算のタイプを指示するフラグ。  
*isign* = 0 の場合は、係数 *wsave* が初期化される。  
*isign* = -1 の場合は、順方向の FFT を実行する。入力と出力は通常の順序である。  
*isign* = +1 の場合は、逆方向の FFT を実行する。入力と出力は通常の順序である。  
*isign* = -2 の場合は、順方向の FFT を実行する。入力は通常の順序、出力はビット反転した順序である。  
*isign* = +2 の場合は、逆方向の FFT を実行する。入力はビット反転した順序、出力は通常の順序である。

*wsave* float\* (*cfft1dc* の場合)  
double\* (*zfft1dc* の場合) (3\*n) 以上のサイズの配列へのポインタ。  
*isign* = 0 の場合、*wsave* は出力パラメータになる。それ以外の場合、*wsave* には、*isign* = 0 と指定してこのルーチン呼び出したときに初期化された FFT 係数が格納される。

### 出力パラメータ

*r* *isign* に従って、変換後の実数部を格納する。*isign* = 0 の場合は、変わらない。

*i* *isign* に従って、変換後の虚数部を格納する。*isign* = 0 の場合は、変わらない。

*wsave* *isign* = 0 の場合、*wsave* には、初期化された FFT の係数が格納される。それ以外の場合、*wsave* は変わらない。

### 実数から複素数への 1 次元 FFT

実数から複素数へのルーチンは、いずれも次の式に従って、実数の入力ベクトルの順方向の FFT を計算する。

$$z_j = \sum_{k=0}^{n-1} t_k w^{-j*k}, \quad 0 \leq j \leq n-1$$

$t_k = \text{cmplx}(r_k, 0)$  の場合。 $r_k$  は実数の入力ベクトルで、 $0 \leq k \leq n-1$  である。  
数値演算の結果  $z_j, 0 \leq j \leq n-1$  は、複素共役対称ベクトルになる。  
 $z(n/2+i) = \text{conjg}(z(n/2-i)), 1 \leq i \leq n/2-1$ 、また  $z(0)$  と  $z(n/2)$  は実数値である。

この複素共役対称 (CCS) ベクトルは、サイズ  $(n/2+1)$  の複素数の配列またはサイズ  $(n+2)$  の実数の配列として格納される。CCS 形式のデータ格納は、Fortran インターフェイス向けルーチンと C インターフェイス向けルーチンについて別々に後で定義する。

表 11-13 は、すべての虚数成分が 0 である長さ  $n = 8$  のベクトルに対して複素数から複素数への `cfft1d/zfft1d` の FFT を実行した場合と、同じベクトルに対して実数から複素数への `scfft1d/zdfft1d` の FFT を実行した場合について、両方の効果を比較したものである。後者の方法のメリットは、必要なデータ格納量が半分で済み、虚数部を 0 にする必要がないことである。最後の 2 つの列は、複素共役対称ベクトルを CCS 形式で格納する、サイズ  $(n+2)$  の実数の配列に格納される。

所定の長さを持つ順方向の FFT を計算するには、まず、`isign = 0` に設定して該当するルーチンを呼び出し、係数を初期化しなければならない。係数の初期化後は、`isign` に 0 以外の値を指定してルーチンを呼び出し、同じ長さの実数から複素数への変換と複素数から実数への変換を何度でも計算できる。

**表 11-22 複素数から複素数への FFT と実数から複素数への FFT の格納効果の比較**

入力ベクトル			出力ベクトル			
<code>cfft1d</code>		<code>scfft1d</code>	<code>cfft1d</code>		<code>scfft1d</code>	
複素数データ		実数データ	複素数データ		実数データ	
実数	虚数部		実数	虚数部	( 実数部 )	( 虚数部 )
0.841471	0.000000	0.841471	1.543091	0.000000	1.543091	0.000000
0.909297	0.000000	0.909297	3.875664	0.910042	3.875664	0.910042
0.141120	0.000000	0.141120	-0.915560	-0.397326	-0.915560	-0.397326
-0.756802	0.000000	-0.756802	-0.274874	-0.121691	-0.274874	-0.121691
-0.958924	0.000000	-0.958924	-0.181784	0.000000	-0.181784	0.000000
-0.279415	0.000000	-0.279415	-0.274874	0.121691		
0.656987	0.000000	0.656987	-0.915560	0.397326		
0.989358	0.000000	0.989358	3.875664	-0.910042		



## scfft1d/dzfft1d (非推奨)

Fortran インターフェイス・ルーチン。実数ベクトルの順方向の FFT を計算し、結果の複素共役対称ベクトルを CCS 形式で表現する (インプレース)。

### 構文

```
call scfft1d( r, n, isign, wsave )
call dzfft1d( r, n, isign, wsave )
```

### 説明

scfft1d/dzfft1d ルーチンが実行する演算は、*isign* の値によって決まる。上記の「[実数から複素数への 1 次元 FFT](#)」の演算式を参照。これらのルーチンは、複素数から実数への変換ルーチンである [csfft1d/zdfft1d](#) を補足する。

### 入力パラメータ

<i>r</i>	REAL (scfft1d の場合) DOUBLE PRECISION (dzfft1d の場合)  配列、次元は $(n+2)$ 以上。最初の $n$ 個の成分には、変換される入力ベクトルを格納する。出力では、成分 $r(n+1)$ と $r(n+2)$ が使用される。 <i>isign</i> = 0 の場合は、配列 <i>r</i> は参照されない。
<i>n</i>	INTEGER。変換の長さ。 <i>n</i> は、2 の累乗でなければならない。
<i>isign</i>	INTEGER。実行する演算のタイプを指示するフラグ。 <i>isign</i> が 0 の場合は、係数 <i>wsave</i> が初期化される。 <i>isign</i> が 0 でない場合は、順方向の FFT を実行する。
<i>wsave</i>	REAL (scfft1d の場合) DOUBLE PRECISION (dzfft1d の場合)  配列、次元は $(2*n+4)$ 以上。 <i>isign</i> = 0 の場合、 <i>wsave</i> には出力データが格納される。それ以外の場合、 <i>wsave</i> には、FFT の実行に必要な係数 (このルーチンか、このルーチンを補足する複素数から実数への FFT ルーチンを直前に呼び出したときに初期化されたもの) が格納される。

## 出力パラメータ

*r*            *isign* = 0 の場合、*r* は変わらない。*isign* が 0 でない場合、出力される実数値の配列 *r*(1:*n*+2) には、Fortran インターフェイス向け CCS 形式で圧縮された複素共役対称ベクトル *z*(1:*n*) が格納される。  
次の表は、出力される配列と格納されるベクトルの関係を示している。

<i>r</i> (1)	<i>r</i> (2)	<i>r</i> (3)	<i>r</i> (4)	...	<i>r</i> ( <i>n</i> -1)	<i>r</i> ( <i>n</i> )	<i>r</i> ( <i>n</i> +1)	<i>r</i> ( <i>n</i> +2)
<i>z</i> (1)	0	RE <i>z</i> (2)	IM <i>z</i> (2)	...	RE <i>z</i> ( <i>n</i> /2)	IM <i>z</i> ( <i>n</i> /2)	<i>z</i> ( <i>n</i> /2+1)	0

完全複素ベクトル *z*(1:*n*) が次のように定義される場合、

*z*(*i*) = `cmplx(r(2*i-1), r(2*i))`,  
 $1 \leq i \leq n/2+1$ ,

*z*(*n*/2+*i*) = `conjg(z(n/2+2-i))`,  
 $2 \leq i \leq n/2$

*z*(1:*n*) は、実数の入力ベクトル *r*(1:*n*) の順方向の FFT になる。

*wsave*        *isign* = 0 の場合、*wsave* には、呼び出されたルーチンに必要な係数が格納される。それ以外の場合、*wsave* は変わらない。

## scfft1dc/dzfft1dc (非推奨)

C インターフェイス・ルーチン。実数ベクトルの順方向の FFT を計算し、結果の複素共役対称ベクトルを CCS 形式で表現する (インプレース)。

### 構文

```
void scfft1dc( float* r, int n, int isign, float* wsave );
void dzfft1dc( double* r, int n, int isign, double* wsave );
```

### 説明

scfft1dc/dzfft1dc ルーチンが実行する演算は、*isign* の値によって決まる。上記の「[実数から複素数への 1 次元 FFT](#)」の演算式を参照。  
これらのルーチンは、複素数から実数への変換ルーチンである [csfft1dc/zdfft1dc](#) を補足する。

## 入力パラメータ

**r** float\* (scffft1dc の場合)  
double\* (dzffft1dc の場合)

( $n+2$ ) 以上のサイズの配列へのポインタ。最初の  $n$  個の成分には、変換される入力ベクトルを格納する。 $isign=0$  の場合は、配列  $r$  は参照されない。

**n** int。変換の長さ。 $n$  は、2 の累乗でなければならない。

**isign** int。実行する演算のタイプを指示するフラグ。

$isign$  が 0 の場合は、係数  $wsave$  が初期化される。

$isign$  が 0 でない場合は、順方向の FFT を実行する。

**wsave** float\* (scffft1dc の場合)  
double\* (dzffft1dc の場合)

( $2*n+4$ ) 以上のサイズの配列へのポインタ。

$isign=0$  の場合、 $wsave$  には出力データが格納される。それ以外の場合、 $wsave$  には、FFT の実行に必要な係数 (このルーチンか、このルーチンを補足する複素数から実数への FFT ルーチンを直前に呼び出したときに初期化されたもの) が格納される。

## 出力パラメータ

**r**  $isign=0$  の場合、 $r$  は変わらない。 $isign$  が 0 でない場合、出力される実数値の配列  $r(0:n+1)$  には、C インターフェイス向け CCS 形式で圧縮された複素共役対称ベクトル  $z(0:n-1)$  が格納される。

次の表は、出力される配列と格納されるベクトルの関係を示している。

$r(0)$	$r(1)$	$r(2)$	...	$r(n/2)$	$r(n/2+1)$	$r(n/2+2)$	...	$r(n)$	$r(n+1)$
$z(0)$	RE $z(1)$	RE $z(2)$	...	$z(n/2)$	0	IM $z(1)$	...	IM $z(n/2-1)$	0

完全複素ベクトル  $z(0:n-1)$  が次のように定義される場合、

$$z(i) = \text{cmplx}(r(i), r(n/2+1+i)), 0 \leq i \leq n/2,$$

$$z(n/2+i) = \text{conjg}(z(n/2-i)), 1 \leq i \leq n/2-1$$

$z(0:n-1)$  は、長さ  $n$  の実数の入力ベクトルの順方向の FFT になる。

**wsave**  $isign=0$  の場合、 $wsave$  には、呼び出されたルーチンに必要な係数が格納される。それ以外の場合、 $wsave$  は変わらない。

## 複素数から実数への 1 次元 FFT

複素数から実数へのルーチンは、いずれも次の式に従って、1 次元逆 FFT を計算する。

$$t_j = \frac{1}{n} \sum_{k=0}^{n-1} z_k * w^{j*k}, \quad 0 \leq j \leq n-1$$

数値演算の入力は、複素共役対称ベクトル  $z_j$ 、 $0 \leq j \leq n-1$  である。  
 $z(n/2+i) = \text{conjg}(z(n/2-i))$ 、 $1 \leq i \leq n/2-1$ 、また  $z(0)$  と  $z(n/2)$  は実数値である。

数値演算の結果は、 $t_j = \text{cmlpx}(r_j, 0)$  である。 $r_j$  は実数ベクトルで、 $0 \leq j \leq n-1$  である。

複素数から実数への変換ルーチンの入力は、サイズ  $(n+2)$  の実数の配列である。この配列には、複素共役対称ベクトル  $z(0:n-1)$  が CCS 形式で格納される（上記の「[実数から複素数への 1 次元 FFT](#)」を参照）。

複素数から実数への変換ルーチンの出力は、サイズ  $n$  の実数ベクトルである。

[表 11-23](#) は、入力ベクトルと出力ベクトルを逆にした点を除いて、[表 11-13](#) と同じ内容である。複素数から実数へのルーチンでは、最後の 2 つの列は、複素共役対称ベクトルを CCS 形式で格納する、サイズ  $(n+2)$  の入力実数配列に格納される。

所定の長さを持つ逆方向の FFT を計算するには、まず、 $isign = 0$  に設定して使用するルーチンと呼び出し、係数を初期化しなければならない。係数の初期化後は、 $isign$  に 0 以外の値を設定して適切なルーチンと呼び出し、同じ長さの実数から複素数への変換と複素数から実数への変換を何度でも計算できる。

**表 11-23 複素数から実数への FFT と複素数から複素数への FFT の格納効果の比較**

出力ベクトル			入力ベクトル			
cfft1d		csfft1d	cfft1d		csfft1d	
複素数データ		実数データ	複素数データ		実数データ	
実数	虚数部		実数	虚数部	(実数部)	(虚数部)
0.841471	0.000000	0.841471	1.543091	0.000000	1.543091	0.000000
0.909297	0.000000	0.909297	3.875664	0.910042	3.875664	0.910042
0.141120	0.000000	0.141120	-0.915560	-0.397326	-0.915560	-0.397326
-0.756802	0.000000	-0.756802	-0.274874	-0.121691	-0.274874	-0.121691
-0.958924	0.000000	-0.958924	-0.181784	0.000000	-0.181784	0.000000
-0.279415	0.000000	-0.279415	-0.274874	0.121691		

表 11-23 複素数から実数への FFT と複素数から複素数への FFT の格納効果の比較

出力ベクトル			入力ベクトル			
cfft1d		csfft1d	cfft1d		csfft1d	
複素数データ		実数データ	複素数データ		実数データ	
実数	虚数部		実数	虚数部	( 実数部 )	( 虚数部 )
0.656987	0.000000	0.656987	-0.915560	0.397326		
0.989358	0.000000	0.989358	3.875664	-0.910042		

## csfft1d/zdfft1d ( 非推奨 )

Fortran インターフェイス・ルーチン。CCS 形式で圧縮された複素共役対称ベクトルの逆 FFT を計算する ( インプレース )。

### 構文

```
call csfft1d( r, n, isign, wsave )
call zdfft1d( r, n, isign, wsave )
```

### 説明

csfft1d/zdfft1d ルーチンが実行する演算は、*isign* の値によって決まる。上記の「[複素数から実数への 1 次元 FFT](#)」の演算式を参照。

これらのルーチンは、実数から複素数への変換ルーチンである [scfft1d/zdfft1d](#) を補足する。

### 入力パラメータ

*r*            REAL (csfft1d の場合 )  
               DOUBLE PRECISION (zdfft1d の場合 )  
 配列、次元は ( *n*+2 ) 以上。  
*isign* = 0 の場合は、参照されない。

*isign* が 0 でない場合は、 $r(1:n+2)$  には、Fortran インターフェイス向け CCS 形式で圧縮された複素共役対称ベクトルが格納される。  
次の表は、出力される配列と格納されるベクトルの関係を示している。

$r(1)$	$r(2)$	$r(3)$	$r(4)$	...	$r(n-1)$	$r(n)$	$r(n+1)$	$r(n+2)$
$z(1)$	0	REz(2)	IMz(2)	...	REz(n/2)	IMz(n/2)	$z(n/2+1)$	0

完全複素ベクトル  $z(1:n)$  が次のように定義される場合、

$z(i) = \text{cmplx}(r(2*i-1), r(2*i)),$   
 $1 \leq i \leq n/2+1,$

$z(n/2+i) = \text{conjg}(z(n/2+2-i)),$   
 $2 \leq i \leq n/2$

変換の実行後、 $r(1:n)$  には、複素共役対称ベクトル  $z(1:n)$  の逆 FFT が格納される。

- n*            INTEGER。変換の長さ。 $n$  は、2 の累乗でなければならない。
- isign*        INTEGER。実行する演算のタイプを指示するフラグ。  
*isign* が 0 の場合は、係数 *wsave* が初期化される。  
*isign* が 0 でない場合は、逆方向の FFT を実行する。
- wsave*        REAL (csfftld の場合)  
              DOUBLE PRECISION (zdfftld の場合) 配列、次元は  $(2*n+4)$  以上。  
*isign* = 0 の場合、*wsave* には出力データが格納される。それ以外の場合  
は、*wsave* には、FFT の実行に必要な係数 (このルーチンか、このルーチン  
を補足する実数から複素数への FFT ルーチンを直前に呼び出したときに初  
期化されたもの) が格納される。

## 出力パラメータ

- r*            *isign* が 0 でない場合は、 $r(1:n)$  は、複素共役対称ベクトル  $z(1:n)$  の逆 FFT の実数の結果になる。*isign* = 0 の場合は、変わらない。
- wsave*        *isign* = 0 の場合、*wsave* には、呼び出されたルーチンに必要な係数が格納される。それ以外の場合、*wsave* は変わらない。

## csfft1dc/zdfft1dc (非推奨)

C インターフェイス・ルーチン。CCS 形式で圧縮された複素共役対称ベクトルの逆FFTを計算する (インプレース)。

### 構文

```
void csfft1dc( float* r, int n, int isign, float* wsave )
void zdfft1dc( double* r, int n, int isign, double* wsave )
```

### 説明

csfft1dc/zdfft1dc ルーチンが実行する演算は、*isign* の値によって決まる。上記の「[複素数から実数への1次元FFT](#)」の演算式を参照。

これらのルーチンは、実数から複素数への変換ルーチンである [scfft1dc/zdfft1dc](#) を補足する。

### 入力パラメータ

*r*            float\* (csfft1dc の場合)  
               double\* (zdfft1dc の場合)

(*n*+2) 以上のサイズの配列へのポインタ。*isign* = 0 の場合は、参照されない。

*isign* が 0 でない場合は、*r*(0:*n*+1) には、C インターフェイス向け CCS 形式で圧縮された複素共役対称ベクトルが格納される。

次の表は、出力される配列と格納されるベクトルの関係を示している。

<i>r</i> (0)	<i>r</i> (1)	<i>r</i> (2)	...	<i>r</i> ( <i>n</i> /2)	<i>r</i> ( <i>n</i> /2+1)	<i>r</i> ( <i>n</i> /2+2)	...	<i>r</i> ( <i>n</i> )	<i>r</i> ( <i>n</i> +1)
<i>z</i> (0)	RE <i>z</i> (1)	RE <i>z</i> (2)	...	<i>z</i> ( <i>n</i> /2)	0	IM <i>z</i> (1)	...	IM <i>z</i> ( <i>n</i> /2-1)	0

完全複素ベクトル *z*(0:*n*-1) が次のように定義される場合、  
 $z(i) = \text{cmplx}(r(i), r(n/2+1+i)), 0 \leq i \leq n/2,$

$z(n/2+i) = \text{conjg}(z(n/2-i)), 1 \leq i \leq n/2-1$

変換の実行後、*r*(0:*n*-1) は、複素共役対称ベクトル *z*(0:*n*-1) の逆FFTになる。

*n*            int. 変換の長さ。*n* は、2 の累乗でなければならない。

*isign*      *int*。実行する演算のタイプを指示するフラグ。  
*isign* = 0 の場合は、係数 *wsave* が初期化される。  
*isign* が 0 でない場合は、逆方向の FFT が実行される。

*wsave*      *float\** (*csfft1dc* の場合 )  
*double\** (*zdfft1dc* の場合 ) ( $2 \cdot n + 4$ ) 以上のサイズの配列へのポインタ。  
*isign* = 0 の場合、*wsave* には出力データが格納される。それ以外の場合  
は、*wsave* には、FFT の実行に必要な係数 ( このルーチンか、このルーチン  
を補足する実数から複素数への FFT ルーチンを直前に呼び出したときに初  
期化されたもの ) が格納される。

## 出力パラメータ

*r*            *isign* が 0 でない場合は、*r*(0:*n*-1) は、複素共役対称ベクトル *z*(0:*n*-1)  
の逆 FFT の実数の結果になる。  
*isign* = 0 の場合は、変わらない。

*wsave*      *isign* = 0 の場合、*wsave* には、呼び出されたルーチンに必要な係数が格納  
される。それ以外の場合、*wsave* は変わらない。

## 2 次元 FFT

2 次元 FFT は、機能的には 1 次元 FFT と同じである。2 次元 FFT は、以下のグループに分けられる。

- [複素数から複素数への変換](#)
- [実数から複素数への変換](#)
- [複素数から実数への変換](#)



---

**注:** これらの関数はすでに置き換えられており、新規の開発では使用すべきでない。「[DFT 関数](#)」を代わりに使用すること。

---

すべての 2 次元 FFT は、インプレースである。変換の長さは 2 の累乗でなければならない。複素数から複素数への変換ルーチンは、複素行列の順方向と逆方向の変換を実行する。実数から複素数への変換ルーチンは、実行列の順方向の変換を実行する。複素数から実数への変換ルーチンは、実数配列で圧縮された複素共役対称行列の逆方向の変換を実行する。



2次元FFTに対する命名規則は、すべてのケースで"1d"が"2d"に置き換わることを除けば、1次元FFTに対する規則と同じである。[表 11-24](#)に、2次元FFTルーチンのグループと、それらに関連するデータ型を示す。

**表 11-24 2次元FFT: 名前およびデータ型**

グループ	FORTRAN の 複素数データ として格納	C の実数 データとして 格納	データ型	説明
複素数から 複素数へ	<a href="#">cfft2d/</a> <a href="#">zfft2d</a>	<a href="#">cfft2dc/</a> <a href="#">zfft2dc</a>	c, z	複素数データから複素数データへの変換を実行する。
実数から 複素数へ	<a href="#">scfft2d/dz</a> <a href="#">fft2d</a>	<a href="#">scfft2dc/d</a> <a href="#">zfft2dc</a>	sc, dz	実数データから複素数データへの順方向の変換を実行する。 scfft2d/zdfft2d と scfft2dc/zdfft2dc のFFTを補足する。
複素数から 実数へ	<a href="#">csfft2d/zd</a> <a href="#">fft2d</a>	<a href="#">csfft2dc/z</a> <a href="#">dfft2dc</a>	cs, zd	複素数データから実数データへの逆方向の変換を実行する。 scfft2d/dzfft2d と scfft2dc/dzfft2dc のFFTを補足する。

C インターフェイスでは、値で渡されるスカラ値が必要である。2次元FFTが1次元FFTと大きく異なるのは、アプリケーションで変換係数用の格納域を用意する必要がないことである。

データ格納の型とデータ構造の要件は、1次元FFTに対するものと同じである。詳細は、本章の最初にある「[データ格納の型](#)」と「[データ構造の要件](#)」のセクションを参照。

## 複素数から複素数への2次元FFT

複素数から複素数への各ルーチンでは、複素行列の順方向または逆方向のFFTをインプレースで計算する。

順方向のFFTは、次の式に従って計算される。

$$z_{i,j} = \sum_{k=0}^{m-1} \sum_{l=0}^{n-1} r_{k,l} * w_m^{-i*k} * w_n^{-j*l}, \quad 0 \leq i \leq m-1, \quad 0 \leq j \leq n-1$$

逆方向のFFTは、次の式に従って計算される。

$$r_{i,j} = \frac{1}{m \cdot n} \sum_{k=0}^{m-1} \sum_{l=0}^{n-1} z_{k,l} \cdot w_m^{i \cdot k} \cdot w_n^{j \cdot l}, \quad 0 \leq i \leq m-1, 0 \leq j \leq n-1$$

$w_m = \exp\left[\frac{2\pi i}{m}\right]$ 、 $w_n = \exp\left[\frac{2\pi i}{n}\right]$ 、で、 $i$  は虚数単位である。

複素数から複素数へのルーチンが実行する演算は、*isign* パラメータの値によって決まる。

*isign* = -1 の場合は、順方向の FFT を実行する。入力と出力は通常順序である。

*isign* = +1 の場合は、逆方向の FFT を実行する。入力と出力は通常順序である。

*isign* = -2 の場合は、順方向の FFT を実行する。入力は通常順序、出力はビット反転した順序である。

*isign* = +2 の場合は、逆方向の FFT を実行する。入力はビット反転した順序、出力は通常順序である。

上記の式は、[表 11-24](#) に示したすべてのデータ型を使用するすべての FFT に適用される。

---

## cfft2d/zfft2d (非推奨)

*Fortran* インターフェイス・ルーチン。複素行列の順方向または逆方向の FFT を計算する (インプレーズ)。

---

### 構文

```
call cfft2d( r, m, n, isign )
call zfft2d( r, m, n, isign )
```

### 説明

cfft2d/zfft2d ルーチンが実行する演算は、*isign* の値によって決まる。上記の「[複素数から複素数への 2 次元 FFT](#)」の演算式を参照。

## 入力パラメータ

<i>r</i>	COMPLEX (cfft2d の場合) DOUBLE COMPLEX (zfft2d の場合) 配列、次元は ( <i>m</i> , <i>n</i> ) 以上。リーディング・ディメンジョンは <i>m</i> に等しくなる。この配列には、変換される複素行列を格納する。
<i>m</i>	INTEGER。列変換の長さ (行数)。 <i>m</i> は、2 の累乗でなければならない。
<i>n</i>	INTEGER。行変換の長さ (列数)。 <i>n</i> は、2 の累乗でなければならない。
<i>isign</i>	INTEGER。実行する演算のタイプを指示するフラグ。 <i>isign</i> = -1 の場合は、順方向の FFT が実行される。入力と出力は通常の順序である。 <i>isign</i> = +1 の場合は、逆方向の FFT が実行される。入力と出力は通常の順序である。 <i>isign</i> = -2 の場合は、順方向の FFT を実行する。入力は通常の順序、出力はビット反転した順序である。 <i>isign</i> = +2 の場合は、逆方向の FFT を実行する。入力はビット反転した順序、出力は通常の順序である。

## 出力パラメータ

<i>r</i>	<i>isign</i> に従って、変換の結果得られた複素数データが格納される。
----------	--

---

## cfft2dc/zfft2dc (非推奨)

C インターフェイス・ルーチン。複素行列の順方向または逆方向の FFT を計算する (インプレース)。

---

### 構文

```
void cfft2dc( float* r, float* i, int m, int n, int isign )  
void zfft2dc( double* r, double* i, int m, int n, int isign )
```

### 説明

cfft2dc/zfft2dc ルーチンが実行する演算は、*isign* の値によって決まる。上記の「[複素数から複素数への 2 次元 FFT](#)」の演算式を参照。

## 入力パラメータ

- r* float\* (cfft2dc の場合)  
double\* (zfft2dc の場合)
- (*m*, *n*) 以上のサイズの 2 次元配列へのポインタ。リーディング・ディメンジョンは *n* に等しくなる。この配列には、変換される複素行列の実数部を格納する。
- i* float\* (cfft2dc の場合)  
double\* (zfft2dc の場合)
- (*m*, *n*) 以上のサイズの 2 次元配列へのポインタ。リーディング・ディメンジョンは *n* に等しくなる。この配列には、変換される複素行列の虚数部を格納する。
- m* int。列変換の長さ (行数)。 *m* は、2 の累乗でなければならない。
- n* int。行変換の長さ (列数)。 *n* は、2 の累乗でなければならない。
- isign* int。実行する演算のタイプを指示するフラグ。
- isign* = -1 の場合は、順方向の FFT が実行される。入力と出力は通常の順序である。
- isign* = +1 の場合は、逆方向の FFT が実行される。入力と出力は通常の順序である。
- isign* = -2 の場合は、順方向の FFT を実行する。入力は通常の順序、出力はビット反転した順序である。
- isign* = +2 の場合は、逆方向の FFT を実行する。入力はビット反転した順序、出力は通常の順序である。

## 出力パラメータ

- r* *isign* に従って、変換後の実数部が格納される。
- i* *isign* に従って、変換後の虚数部が格納される。

## 実数から複素数への 2 次元 FFT

実数から複素数へのルーチンは、いずれも次の式に従って、実数行列の順方向の FFT を計算する。

$$z_{i,j} = \sum_{k=0}^{m-1} \sum_{l=0}^{n-1} t_{k,l} w_m^{-i*k} w_n^{-j*l}, \quad 0 \leq i \leq m-1, \quad 0 \leq j \leq n-1$$

$t_{k,l} = \text{cmplx}(r_{k,l}, 0)$ 、ここで  $r_{k,l}$  は実数入力行列で、 $0 \leq k \leq m-1$ ,  $0 \leq l \leq n-1$  である。

数値演算の結果  $z_{i,j}$ ,  $0 \leq i \leq m-1$ ,  $0 \leq j \leq n-1$  は、サイズ  $(m,n)$  の複素行列である。各列は、次のような複素共役対称ベクトルになる。

$0 \leq j \leq n-1$  の場合は、

$z(m/2+i, j) = \text{conjg}(z(m/2-i, j))$ ,  $1 \leq i \leq m/2-1$ 、  
また  $j=0$  と  $j=n/2$  の場合は、 $z(0, j)$  と  $z(m/2, j)$  は実数値になる。

この数値演算の結果はサイズが  $(m+2, n+2)$  の実数 2 次元配列、サイズ  $(m/2+1, n+1)$  (Fortran インターフェイスの場合) または  $(m+1, n/2+1)$  (C インターフェイスの場合) の複素数 2 次元配列に格納される。CCS 形式のデータ格納は、Fortran インターフェイス向けルーチンと C インターフェイス向けルーチンについて別々に後で定義する。

## scfft2d/dzfft2d (非推奨)

Fortran インターフェイス・ルーチン。実数行列の順方向の FFT を計算し、結果の複素共役対称行列を CCS 形式で表現する (インプレース)。

### 構文

```
call scfft2d( r, m, n )
call dzfft2d( r, m, n )
```

### 説明

上記の「[実数から複素数への 2 次元 FFT](#)」の演算式を参照。

これらのルーチンは、複素数から実数への変換ルーチンである [csfft2d/zdfft2d](#) を補足する。

### 入力パラメータ

$r$  REAL (scfft2d の場合)  
DOUBLE PRECISION (dzfft2d の場合) 配列、次元は  $(m+2, n+2)$  以上。リーディング・ディメンジョンは  $(m+2)$  に等しくなる。この配列の最初の  $m$  行  $n$  列には、変換される実行列を格納する。[表 11-25](#) に、入力データのレイアウトを示す。

$m$             INTEGER。列変換の長さ (行数)。  $m$  は、2 の累乗でなければならない。

$n$             INTEGER。行変換の長さ (列数)。  $n$  は、2 の累乗でなければならない。

表 11-25 実数から複素数への 2 次元 FFT と複素数から実数への 2 次元 FFT の  
実数データ格納形式 (Fortran インターフェイス)

$r(1, 1)$	$r(1, 2)$	...	$r(1, n-1)$	$r(1, n)$	n/u	n/u
$r(2, 1)$	$r(2, 2)$	...	$r(2, n-1)$	$r(2, n)$	n/u	n/u
$r(3, 1)$	$r(3, 2)$	...	$r(3, n-1)$	$r(3, n)$	n/u	n/u
$r(4, 1)$	$r(4, 2)$	...	$r(4, n-1)$	$r(4, n)$	n/u	n/u
...	...	...	...	...	...	...
$r(m-1, 1)$	$r(m-1, 2)$	...	$r(m-1, n-1)$	$r(m-1, n)$	n/u	n/u
$r(m, 1)$	$r(m, 2)$	...	$r(m, n-1)$	$r(m, n)$	n/u	n/u
n/u	n/u	...	n/u	n/u	n/u	n/u
n/u	n/u	...	n/u	n/u	n/u	n/u

\* n/u - 使用しない

### 出力パラメータ

$r$  出力される実数の配列  $r(1:m+2, 1:n+2)$  には、Fortran インターフェイス向け CCS 形式で以下のように圧縮された複素共役対称行列  $z(1:m, 1:n)$  が格納される。

- 行 1 と  $m+1$  には、 $n+2$  の位置に、CCS 形式で圧縮された複素共役対称ベクトル  $z(1, j)$  と  $z(m/2+1, j)$  が格納される (上記の「[実数から複素数への 1 次元 FFT](#)」を参照)。

完全複素ベクトル  $z(1, j)$  は、次のように定義される。

$$z(1, j) = \text{cmplx}(r(1, 2*j-1), r(1, 2*j)), 1 \leq j \leq n/2+1,$$

$$z(1, n/2+1+j) = \text{conjg}(z(1, n/2+1-j)), 1 \leq j \leq n/2-1$$

完全複素ベクトル  $z(m/2+1, j)$  は次のように定義される。

$$z(m/2+1, j) = \text{cmplx}(r(m+1, 2*j-1), r(m+1, 2*j)),$$

$$1 \leq j \leq n/2+1,$$

$$z(m/2+1, n/2+1+j) = \text{conjg}(z(m/2+1, n/2+1-j)),$$

$$1 \leq j \leq n/2-1;$$

- 行 3 ～  $m$  には、 $n$  の位置に、次のように表現される複素ベクトルを格納する。

$$z(i+1, j) = \text{cmplx}(r(2*i+1, j), r(2*i+2, j)),$$

$$1 \leq i \leq m/2-1, 1 \leq j \leq n$$

- その他の行列成分は、次のように求められる。  

$$z(m/2+1+i, j) = \text{conjg}(z(m/2+1-i, j)),$$

$$1 \leq i \leq m/2-1, 1 \leq j \leq n$$

Fortran インターフェイスの場合の複素共役対称行列  $z$  の格納形式を、[表 11-26](#) に示す。

**表 11-26 実数から複素数への 2 次元 FFT と複素数から実数への 2 次元 FFT の CCS 形式のデータ格納形式 (Fortran インターフェイス)**

$z(1,1)$	0	$\text{RE}z(1,2)$	$\text{IM}z(1,2)$	...	$\text{RE}z(1,n/2)$	$\text{IM}z(1,n/2)$	$z(1, n/2+1)$	0
0	0	0	0	...	0	0	0	0
$\text{RE}z(2,1)$	$\text{RE}z(2,2)$	$\text{RE}z(2,3)$	$\text{RE}z(2,4)$	...	$\text{RE}z(2,n-1)$	$\text{RE}z(2,n)$	n/u	n/u
$\text{IM}z(2,1)$	$\text{IM}z(2,2)$	$\text{IM}z(2,3)$	$\text{IM}z(2,4)$	...	$\text{IM}z(2,n-1)$	$\text{IM}z(2,n)$	n/u	n/u
...	...	...	...	...	...	...	n/u	n/u
$\text{RE}z(m/2,1)$	$\text{RE}z(m/2,2)$	$\text{RE}z(m/2,3)$	$\text{RE}z(m/2,4)$	...	$\text{RE}z(m/2, n-1)$	$\text{RE}z(m/2, n)$	n/u	n/u
$\text{IM}z(m/2,1)$	$\text{IM}z(m/2,2)$	$\text{IM}z(m/2,3)$	$\text{IM}z(m/2,4)$	...	$\text{IM}z(m/2, n-1)$	$\text{IM}z(m/2, n)$	n/u	n/u
$z(m/2+1,1)$	0	$\text{RE}z(m/2+1,2)$	$\text{IM}z(m/2+1,2)$	...	$\text{RE}z(m/2+1, n/2)$	$\text{IM}z(m/2+1, n/2)$	$z(m/2+1, n/2+1)$	0
0	0	0	0	...	0	0	n/u	n/u

\* n/u - 使用しない

## scfft2dc/dzfft2dc (非推奨)

C インターフェイス・ルーチン。実数行列の順方向の FFT を計算し、結果の複素共役対称行列を CCS 形式で表現する (インプレース)。

### 構文

```
void scfft2dc( float* r, int m, int n )
void dzfft2dc( double* r, int m, int n )
```

### 説明

上記の「[実数から複素数への 2 次元 FFT](#)」の演算式を参照。



これらのルーチンは、複素数から実数への変換ルーチンである [csfft2dc/zdfft2dc](#) を補足する。

## 入力パラメータ

$r$  float\* (scfft2dc の場合)  
double\* (dzfft2dc の場合)

( $m+2, n+2$ ) 以上のサイズの配列へのポインタ。リーディング・ディメンジョンは ( $n+2$ ) に等しくなる。この配列の最初の  $m$  行  $n$  列には、変換される実行列を格納する。

[表 11-27](#) に、入力データのレイアウトを示す。

$m$  int。列変換の長さ。  
 $m$  は、2 の累乗でなければならない。

$n$  int。行変換の長さ。  
 $n$  は、2 の累乗でなければならない。

**表 11-27 実数から複素数への 2 次元 FFT と複素数から実数への 2 次元 FFT の実数データ格納形式 (C インターフェイス)**

$r(0, 0)$	$r(0, 1)$	...	$r(0, n-2)$	$r(0, n-1)$	n/u	n/u
$r(1, 0)$	$r(1, 1)$	...	$r(1, n-2)$	$r(1, n-1)$	n/u	n/u
$r(2, 0)$	$r(2, 1)$	...	$r(2, n-2)$	$r(2, n-1)$	n/u	n/u
$r(3, 0)$	$r(3, 1)$	...	$r(3, n-2)$	$r(3, n-1)$	n/u	n/u
...	...	...	...	...	...	...
$r(m-2, 0)$	$r(m-2, 1)$	...	$r(m-2, n-2)$	$r(m-2, n-1)$	n/u	n/u
$r(m-1, 0)$	$r(m-1, 1)$	...	$r(m-1, n-2)$	$r(m-1, n-1)$	n/u	n/u
n/u	n/u	...	n/u	n/u	n/u	n/u
n/u	n/u	...	n/u	n/u	n/u	n/u

## 出力パラメータ

$r$  出力実数配列  $r(0:m+1, 0:n+1)$  には、C インターフェイス向け CCS 形式で以下のように圧縮された複素共役対称行列  $z(0:m-1, 0:n-1)$  が格納される。

- 列 0 と  $n/2$  には、 $m+2$  の位置に、CCS 形式で複素共役対称ベクトル  $z(i, 0)$  と  $z(i, n/2)$  が格納される (上記の「[実数から複素数への 1 次元 FFT](#)」を参照)。  
完全複素ベクトル  $z(i, 0)$  は、次のように定義される。  

$$z(i, 0) = \text{cmplx}(r(i, 0), r(m/2+i+1, 0)), 0 \leq i \leq m/2,$$

$$z(m/2+i, 0) = \text{conjg}(z(m/2-i, 0)), 1 \leq i \leq m/2-1$$
  
完全複素ベクトル  $z(i, n/2)$  は、次のように定義される。  

$$z(i, n/2) = \text{cmplx}(r(i, n/2), r(m/2+i+1, n/2)), 0 \leq i \leq m/2,$$

$$z(m/2+i, n/2) = \text{conjg}(z(m/2-i, n/2)), 1 \leq i \leq m/2-1$$
- 列  $1 \sim n/2-1$  には複素ベクトルの実数部を格納し、列  $n/2+2 \sim n$  には虚数部を格納する。各ベクトルのこれらの値は  $m$  の位置に格納され、次のように表現される。  

$$z(i, j) = \text{cmplx}(r(i, j), r(i, n/2+1+j)),$$

$$0 \leq i \leq m-1, 1 \leq j \leq n/2-1$$
- その他の行列成分は、次のように求められる。  

$$z(i, n/2+j) = \text{conjg}(z(i, n/2-j)),$$

$$0 \leq i \leq m-1, 1 \leq j \leq n/2-1$$

[表 11-28](#) は、C インターフェイスの場合の複素共役対称行列  $z$  の格納形式を示す。

表 11-28 実数から複素数への 2 次元 FFT と複素数から実数への 2 次元 FFT の CCS 形式のデータ格納形式 (C インターフェイス)

z(0,0)	REz(0,1)	...	REz(0, n/2-1)	z(0,n/2)	0	IMz(0,1)	...	IMz(0, n/2-1)	0
REz(1,0)	REz(1,1)	...	REz(1, n/2-1)	REz(1,n/2)	0	IMz(1,1)	...	IMz(1, n/2-1)	0
...	...	...	...	...	0	...	...	...	0
REz(m/2-1, 0)	REz(m/2-1, 1)	...	REz(m/2-1, n/2-1)	REz(m/2-1, n/2)	0	IMz(m/2-1, 1)	...	IMz(m/2-1, n/2-1)	0
z(m/2,0)	REz(m/2,1)	...	REz(m/2, n/2-1)	z(m/2,n/2)	0	IMz(m/2,1)	...	IMz(m/2, n/2-1)	0
0	REz(m/2+1, 1)	...	REz(m/2+1, n/2-1)	0	0	IMz(m/2+1, 1)	...	IMz(m/2+1, n/2-1)	0
IMz(1,0)	REz(m/2+2, 1)	...	REz(m/2+2, n/2-1)	IMz(1,n/2)	0	IMz(m/2+2, 1)	...	IMz(m/2+2, n/2-1)	0
...	...	...	...	...	0	...	...	...	0
IMz(m/2-2, 0)	REz(m-1,1)	...	REz(m-1, n/2-1)	IMz(m/2-2, n/2)	0	IMz(m-1,1)	...	IMz(m-1, n/2-1)	0
IMz(m/2-1, 0)	n/u	...	n/u	IMz(m/2-1, n/2)	n/u	n/u	...	n/u	n/u
0	n/u	...	n/u	0	n/u	n/u	...	n/u	n/u

## 複素数から実数への 2 次元 FFT

複素数から実数への各ルーチンは、算術式に対して 2 次元の逆方向 FFT を計算する。

$$t_{i,j} = \frac{1}{m \cdot n} \sum_{k=0}^{m-1} \sum_{l=0}^{n-1} z_{k,l} \cdot w_m^{i \cdot k} \cdot w_n^{j \cdot l}, \quad 0 \leq i \leq m-1, \quad 0 \leq j \leq n-1$$

数値演算の入力  $z_{i,j}$ 、 $0 \leq i \leq m-1$ 、 $0 \leq j \leq n-1$  は、サイズ  $(m, n)$  の複素行列である。各列は、次のような複素共役対称ベクトルになる。

$0 \leq j \leq n-1$  の場合は、 $z(m/2+i, j) = \text{conjg}(z(m/2-i, j))$ 、 $1 \leq i \leq m/2-1$ 、  
また  $j=0$  と  $j=n/2$  の場合は、 $z(0, j)$  と  $z(m/2, j)$  は実数値になる。

この数値演算の結果はサイズが  $(m+2, n+2)$  の実数 2 次元配列、サイズ  $(m/2+1, n+1)$  (Fortran インターフェイスの場合) または  $(m+1, n/2+1)$  (C インターフェイスの場合) の複素数 2 次元配列に格納される。CCS 形式のデータ格納は、Fortran インターフェイス向けルーチンと C インターフェイス向けルーチンについて別々に後で定義する。

この変換の数値演算の結果は、 $t_{k,l} = \text{cmplx}(r_{k,l}, 0)$  になる。 $r_{k,l}$  は実数行列で  $0 \leq k \leq m-1, 0 \leq l \leq n-1$  である。

---

## csfft2d/zdfft2d (非推奨)

Fortran インターフェイス・ルーチン。CCS 形式に圧縮された複素共役対称行列の逆 FFT を計算する (インプレース)。

---

### 構文

```
call csfft2d( r, m, n )
call zdfft2d( r, m, n )
```

### 説明

上記の「[複素数から実数への 2 次元 FFT](#)」の演算式を参照。これらのルーチンは、実数から複素数への変換ルーチンである [scfft2d/dzfft2d](#) を補足する。

### 入力パラメータ

*r*            SINGLE PRECISION REAL\*4 (csfft2d の場合)  
              DOUBLE PRECISION REAL\*8 (zdfft2d の場合)

配列、次元は  $(m+2, n+2)$  以上。リーディング・ディメンションは  $(m+2)$  に等しくなる。この配列には、変換される複素共役対称行列を CCS 形式で格納する。[表 11-26](#) に、入力データのレイアウトを示す。

*m*            INTEGER。列変換の長さ (行数)。 *m* は、2 の累乗でなければならない。

*n*            INTEGER。行変換の長さ (列数)。 *n* は、2 の累乗でなければならない。

### 出力パラメータ

*r*            変換の結果の実数部が格納される。出力データのレイアウトは、[表 11-25](#) を参照。

---

## csfft2dc/zdfft2dc (非推奨)

C インターフェイス・ルーチン。CCS 形式に圧縮された複素共役対称行列の逆 FFT を計算する (インプレース)。

---

### 構文

```
void csfft2dc( float* r, int m, int n );  
void zdfft2dc( double* r, int m, int n );
```

### 説明

上記の「[複素数から実数への 2 次元 FFT](#)」の演算式を参照。これらのルーチンは、実数から複素数への変換ルーチンである [scfft2dc/dzfft2dc](#) を補足する。

### 入力パラメータ

*r*            float\* (csfft2dc の場合 )  
             double\* (zdfft2dc の場合 )  
  
             (m+2, n+2) 以上のサイズの配列へのポインタ。リーディング・ディメンジョンは (n+2) に等しくなる。この配列には、変換される複素共役対称行列を CCS 形式で格納する。[表 11-28](#) に、入力データのレイアウトを示す。  
  
*m*            int。列変換の長さ。m は、2 の累乗でなければならない。  
  
*n*            int。行変換の長さ。n は、2 の累乗でなければならない。

### 出力パラメータ

*r*            変換の結果の実数部が格納される。出力データのレイアウトは、scfft2dc/dzfft2dc の入力データと同じである。詳細は、[表 11-27](#) を参照。



# 区間線形ソルバ

12

この章では以下の目的に使用することのできるインテル® MKL ルーチンについて説明する。

- 区間線形連立方程式  $Ax = b$  を解く。 $A = (a_{ij})$  は区間行列、 $b = (b_i)$  は右辺の区間ベクトルである。
- 区間行列の特性を調べる。

区間線形方程式の主要な概念についての詳細は付録 A の「[線形ソルバの基礎](#)」を参照のこと。

以下のルーチンの説明は、課題に応じて次のグループに分けて行う。

[区間方程式を高速に解くためのルーチン](#)

[区間方程式の精密な解を求めるためのルーチン](#)

[区間行列逆転用のルーチン](#)

[区間行列の特性確認用のルーチン](#)

[補助およびユーティリティ・ルーチン](#)

表 12-1 は、区間線形方程式を解くためのインテル MKL ルーチンの全リストである。

表 12-1      インテル MKL 区間線形ソルバルーチン

ルーチン名	説明
<a href="#">?trtrs</a>	三角区間線形連立方程式を後方置換プロシージャによって解く。
<a href="#">?gegas</a>	区間線形連立方程式を区間ガウス法によって解く。
<a href="#">?gehss</a>	区間線形連立方程式を区間 Housholder 法によって解く。
<a href="#">?gekws</a>	区間線形連立方程式を Krawczyk 反復法によって解く。
<a href="#">?gegss</a>	区間線形連立方程式を区間 Gauss-Seidel 反復法によって解く。

表 12-1 インテル MKL 区間線形ソルバルーチン

ルーチン名	説明
<a href="#">?gehbs</a>	区間線形連立方程式を Hansen-Bliek-Rohn プロシージャによって解く。
<a href="#">?gepps</a>	区間線形連立方程式をパラメータ分割法によって解く。
<a href="#">?trtri</a>	三角区間行列の逆区間行列を計算する。
<a href="#">?geszi</a>	逆区間行列を Schulz 区間反復プロシージャによって計算する。
<a href="#">?gerbr</a>	区間行列の正則性を Ris-Beeck と Rex-Rohn 判定基準によってテストする。
<a href="#">?gesvr</a>	区間行列の正則性 / 特異性を Rump と Rex-Rohn 特異値判定基準によってテストする。
<a href="#">?gemip</a>	区間線形方程式の midpoint 逆転プリコンディショニングを実行する。

## ルーチン命名規則

以下に紹介するルーチンでは、LAPACK に類似する命名規則が使用されている。具体的には、すべてのルーチン名は `xxyyzzz` の構造である。冒頭の文字 `xx` は、データ型を示す。

`si` 単精度実数区間  
`di` 倍精度実数区間

3 番目と 4 番目の文字 `yy` は、行列のタイプを示す。

`ge` 一般行列  
`tr` 三角行列

最後の 3 文字 `zzz` は、実行される処理 ( 下記を参照 ) を示す。

`trs` 三角区間線形方程式の後方置換ソルバ  
`gas` 区間線形方程式の区間 Gauss ソルバ  
`has` 区間線形方程式の区間 Householder 法ソルバ  
`kws` 区間線形方程式の Krawczyk 反復法ソルバ  
`gss` 区間線形方程式の区間 Gauss-Seidel 反復法ソルバ  
`hbs` 区間線形方程式の Hansen-Bliek-Rohn ソルバ  
`pps` 区間線形方程式のパラメータ分割法に基づくソルバ  
`tri` 後方置換に基づく三角区間行列の逆行列計算  
`szi` Schulz 反復法による一般区間行列の逆行列計算  
`rbr` Ris-Beeck 判定基準による区間行列の正則性 / 特異性のテスト



svr Rump と Rex-Rohn 特異値判定基準による区間行列の正則性 / 特異性のテスト  
 mip 区間線形方程式の中点逆転プリコンディショニング

グループ名にある疑問符は、データ型を示す各種のキャラクタ・コード (si または di) に対応している。例えば、?trtri は、ルーチン sitrtri または ditrtri のグループ名を示す。

## 区間方程式を高速に解くためのルーチン

### ?trtrs

三角区間線形連立方程式を後方置換プロシージャによって解く。

#### 構文

```
call sitrtrs( uplo, trans, diag, n, nrhs, a, lda, b, ldb, info)
call ditrtrs( uplo, trans, diag, n, nrhs, a, lda, b, ldb, info)
```

#### 説明

?trtrs ルーチンは、行列 **B** に格納された複数の右辺を使用して、三角行列 **A** を持つ以下の区間線形連立方程式を **X** について解く。

$\mathbf{AX} = \mathbf{B}$  ( $trans = 'N'$  の場合)

$\mathbf{A}^T \mathbf{X} = \mathbf{B}$  ( $trans = 'T'$  または  $'C'$  の場合)

このルーチンは、後方置換アルゴリズムを実装し、行列 **A** の単純な構造により、区間線形方程式に対する解集合の最適なエンクロージャを生成する。

#### 入力パラメータ

uplo CHARACTER(1)。'U'、'L'、'u'、または 'l' のいずれかでなければならない。  
**A** が上三角行列か、下三角行列かを指定する。  
 uplo = 'U' または 'u' の場合、**A** は上三角行列である。  
 uplo = 'L' または 'l' の場合、**A** は下三角行列である。

<i>trans</i>	CHARACTER(1)。'N'、'T'、'C'、'n'、't'、または 'c' のいずれかでなければならない。 <i>trans</i> = 'N' または 'n' の場合、 $\mathbf{AX} = \mathbf{B}$ を $\mathbf{X}$ について解く。 <i>trans</i> = 'T'、'C'、't'、'c' のいずれかの場合、 $\mathbf{A}^T \mathbf{X} = \mathbf{B}$ を $\mathbf{X}$ について解く。
<i>diag</i>	CHARACTER(1)。'N'、'U'、'n'、または 'u' のいずれかでなければならない。 <i>diag</i> = 'N' または 'n' の場合、 $\mathbf{A}$ は単位三角行列ではない。 <i>diag</i> = 'U' または 'u' の場合、 $\mathbf{A}$ は単位三角行列である。 $\mathbf{A}$ の対角成分は 1 とみなされ、配列 <i>a</i> 内で参照されない。
<i>n</i>	INTEGER。 $\mathbf{A}$ の次数。 $\mathbf{B}$ の行の数 ( $n \geq 0$ )。
<i>nrhs</i>	INTEGER。 右辺の数 ( $nrhs \geq 0$ )。
<i>a, b</i>	REAL (sitrttrs の場合)。 DOUBLE PRECISION (ditrttrs の場合)。 配列: <i>a</i> ( <i>lda</i> ,*), <i>b</i> ( <i>ldb</i> ,*)。 配列 <i>a</i> には、行列 $\mathbf{A}$ を格納する。 配列 <i>b</i> には、行列 $\mathbf{B}$ を格納する。この行列の列は、連立方程式の右辺である。 <i>a</i> の第 2 次元は $\max(1, n)$ 以上でなければならない。 <i>b</i> の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $lda \geq \max(1, n)$ 。
<i>ldb</i>	INTEGER。 <i>b</i> の第 1 次元。 $ldb \geq \max(1, n)$ 。

## 出力パラメータ

<i>b</i>	解の行列 $\mathbf{X}$ によって上書きされる。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> > 0 の場合、実行は正常に終了しなかったことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正だったことを示す。

## ?gegas

区間線形連立方程式を区間ガウス法によって解く。

### 構文

```
call sigegas( trans, n, nrhs, a, lda, b, ldb, info)
call digegas( trans, n, nrhs, a, lda, b, ldb, info)
```

### 説明

?gegas ルーチンは、区間ガウス法を使用して、以下の区間線形連立方程式に対する解集合のエンクロージャを計算する。

$\mathbf{A}\mathbf{x} = \mathbf{B}$  ( $trans = 'N'$  の場合)

$\mathbf{A}^T \mathbf{x} = \mathbf{B}$  ( $trans = 'T'$  または  $'C'$  の場合)

### 入力パラメータ

<i>trans</i>	CHARACTER(1)。'N'、'T'、'C'、'n'、't'、または 'c' のいずれかでなければならない。 連立方程式の形式を指定する。 $trans = 'N'$ または $'n'$ の場合、 $\mathbf{A}\mathbf{x} = \mathbf{B}$ を $\mathbf{x}$ について解く。 $trans = 'T'$ 、'C'、't'、'c' のいずれかの場合、 $\mathbf{A}^T \mathbf{x} = \mathbf{B}$ を $\mathbf{x}$ について解く。
<i>n</i>	INTEGER。A の次数。B の行の数 ( $n \geq 0$ )。
<i>nrhs</i>	INTEGER。右辺の数 ( $nrhs \geq 0$ )。
<i>a, b</i>	REAL (sigegas の場合)。 DOUBLE PRECISION (digegas の場合)。 配列 $a(lda, *)$ 、 $b(ldb, *)$ 。 配列 $a$ には、行列 A を格納する。 配列 $b$ には、行列 B を格納する。この行列の列は、連立方程式の右辺である。 $a$ の第 2 次元は $\max(1, n)$ 以上でなければならない。 $b$ の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。
<i>lda</i>	INTEGER。a の第 1 次元。 $lda \geq \max(1, n)$ 。

*ldb*                                      INTEGER。 *b* の第 1 次元。  $ldb \geq \max(1, n)$ 。

## 出力パラメータ

*b*    解の行列 *X* によって上書きされる。

*info*                                      INTEGER。  
*info* = 0 の場合、実行は正常に終了したことを示す。  
*info* > 0 の場合、実行は正常に終了しなかったことを示す。  
*info* = -*i* の場合、*i* 番目のパラメータの値が不正だったことを示す。

### 例 12-1                      区間ガウス法の Fortran 90 コード

---

次の Fortran コードの部分は、digegas ルーチンを使用して、区間ガウス法で、区間線形連立方程式に対する解集合のエンクロージャを計算する。

$$\begin{pmatrix} [2, 3] & [0, 1] \\ [1, 2] & [2, 3] \end{pmatrix} \mathbf{x} = \begin{pmatrix} [0, 120] \\ [60, 240] \end{pmatrix}$$

```
-----
.....
USE INTERVAL_ARITHMETIC
.....
TYPE(D_INTERVAL)      :: A(2,2), B(2)
INTEGER                :: N, INFO
CHARACTER(1)           :: TRANS = 'n'
.....
N = 2
A(1,1) = DINTERVAL(2.,3.); A(1,2) = DINTERVAL(0.,1.)
A(2,1) = DINTERVAL(1.,2.); A(2,2) = DINTERVAL(2.,3.)
B(1,1) = DINTERVAL(0.,120.); B(2,1) = DINTERVAL(60.,240.)
.....
CALL DIGEGAS( TRANS, N, 1, A, 2, B, 2, INFO )
-----
```

行列  $A$  と右辺ベクトル  $B$  の成分への倍精度区間の割り当ては、INTERVAL\_ARITHMETIC モジュールで供給される DINTERVAL 関数によって実行される点に注意すること。

## ?gehss

区間線形連立方程式を区間 *Housholder* 法によって解く。

### 構文

```
call sigehss( trans, n, nrhs, a, lda, b, ldb, info)
call digehss( trans, n, nrhs, a, lda, b, ldb, info)
```

### 説明

?gegas ルーチンは、区間 *Housholder* 法を使用して、以下の区間線形連立方程式に対する解集合のエンクロージャを計算する。

$AX = B$  ( $trans = 'N'$  の場合)

$A^T X = B$  ( $trans = 'T'$  または  $'C'$  の場合)

### 入力パラメータ

<i>trans</i>	CHARACTER(1)。'N'、'T'、'C'、'n'、't'、または 'c' のいずれかでなければならない。 連立方程式の形式を指定する。 $trans = 'N'$ または $'n'$ の場合、 $AX = B$ を $X$ について解く。 $trans = 'T'$ 、'C'、't'、'c' のいずれかの場合、 $A^T X = B$ を $X$ について解く。
<i>n</i>	INTEGER。A の次数。B の行の数 ( $n \geq 0$ )。
<i>nrhs</i>	INTEGER。右辺の数 ( $nrhs \geq 0$ )。
<i>a, b</i>	REAL (sigehsss の場合)。 DOUBLE PRECISION (digehss の場合)。 配列 $a$ ( $lda, *$ )、 $b$ ( $ldb, *$ )。 配列 $a$ には、行列 $A$ を格納する。 配列 $b$ には、行列 $B$ を格納する。この行列の列は、連立方程式

の右辺である。

$a$  の第 2 次元は  $\max(1, n)$  以上でなければならない。 $b$  の第 2 次元は  $\max(1, nrhs)$  以上でなければならない。

*lda*                    INTEGER。  $a$  の第 1 次元。  $lda \geq \max(1, n)$ 。

*ldb*                    INTEGER。  $b$  の第 1 次元。  $ldb \geq \max(1, n)$ 。

## 出力パラメータ

*b*                      解の行列  $X$  のエンクロージャによって上書きされる。

*info*                    INTEGER。  
*info* = 0 の場合、実行は正常に終了したことを示す。  
*info* > 0 の場合、実行は正常に終了しなかったことを示す。  
*info* = -*i* の場合、*i* 番目のパラメータの値が不正だったことを示す。

---

## ?gekws

区間線形連立方程式を Krawczyk 反復法によって解く。

---

### 構文

```
call sigekws( trans, n, nrhs, a, lda, b, ldb, epsilon, info)
call digekws( trans, n, nrhs, a, lda, b, ldb, epsilon, info)
```

### 説明

?gekws ルーチンは、Krawczyk 区間反復法を使用して、以下の区間線形連立方程式に対する解集合のエンクロージャを計算する。

$AX = B$  (*trans* = 'N' の場合)

$A^T X = B$  (*trans* = 'T' または 'C' の場合)

### 入力パラメータ

*trans*                    CHARACTER(1)。  
'N'、'T'、'C'、'n'、't'、または 'c' のいずれかでなければならない。  
連立方程式の形式を指定する。

$trans = 'N'$  または  $'n'$  の場合、 $\mathbf{A}\mathbf{x} = \mathbf{B}$  を  $\mathbf{x}$  について解く。  
 $trans = 'T'$ 、 $'C'$ 、 $'t'$ 、 $'c'$  のいずれかの場合、 $\mathbf{A}^T \mathbf{x} = \mathbf{B}$  を  $\mathbf{x}$  について解く。

$n$  INTEGER。  $\mathbf{A}$  の次数。  $\mathbf{B}$  の行の数 ( $n \geq 0$ )。  
 $nrhs$  INTEGER。 右辺の数 ( $nrhs \geq 0$ )。  
 $a, b$  REAL (sigekws の場合 )。  
DOUBLE PRECISION (digekws の場合 )。  
配列 :  $a(lda, *)$ 、 $b ldb, *)$ 。  
配列  $a$  には、行列  $\mathbf{A}$  を格納する。  
配列  $b$  には、行列  $\mathbf{B}$  を格納する。この行列の列は、連立方程式の右辺である。  
 $lda$  INTEGER。  $a$  の第 1 次元。  $lda \geq \max(1, n)$ 。  
 $ldb$  INTEGER。  $b$  の第 1 次元。  $ldb \geq \max(1, n)$ 。  
 $epsilon$  REAL (sigekws の場合 )  
DOUBLE PRECISION (digekws の場合 )。  
推定の指示精度。

## 出力パラメータ

$b$  解の行列  $\mathbf{X}$  のエンクロージャによって上書きされる。  
 $info$  INTEGER。  
 $info = 0$  の場合、実行は正常に終了したことを示す。  
 $info > 0$  の場合、実行は正常に終了しなかったことを示す。  
 $info = -i$  の場合、 $i$  番目のパラメータの値が不正だったことを示す。

## アプリケーション・ノート

Krawczyk 区間反復法では、すでに中点逆転プリコンディショニングが組み込まれているため、さらに [?gemip](#) ルーチンを適用する必要はなく、全体の効率向上にもつながらない。

## ?gegss

区間線形連立方程式を区間 Gauss-Seidel 反復法によって解く。

### 構文

```
call sigegss( trans, n, nrhs, a, lda, b, ldb, encl, epsilon, nits, info)
call digegss( trans, n, nrhs, a, lda, b, ldb, encl, epsilon, nits, info)
```

### 説明

?gegss ルーチンは、区間 Gauss-Seidel 反復法を使用して、以下の区間線形連立方程式に対する解集合の部分のエンクロージャを計算する。

$\mathbf{A}\mathbf{X} = \mathbf{B}$  ( $trans = 'N'$  の場合)

$\mathbf{A}^T\mathbf{X} = \mathbf{B}$  ( $trans = 'T'$  または  $'C'$  の場合)

このルーチンを使用したコードの例は、本書付録 C の [区間連立 1 次方程式ソルバのコード例](#) を参照のこと。

### 入力パラメータ

<i>trans</i>	CHARACTER(1)。'N'、'T'、'C'、'n'、't'、または 'c' のいずれかでなければならない。 連立方程式の形式を指定する。 $trans = 'N'$ または $'n'$ の場合、 $\mathbf{A}\mathbf{X} = \mathbf{B}$ を $\mathbf{X}$ について解く。 $trans = 'T'$ 、'C'、't'、'c' のいずれかの場合、 $\mathbf{A}^T\mathbf{X} = \mathbf{B}$ を $\mathbf{X}$ について解く。
<i>n</i>	INTEGER。A の次数。B の行の数 ( $n \geq 0$ )。
<i>nrhs</i>	INTEGER。右辺の数 ( $nrhs \geq 0$ )。
<i>a, b</i>	REAL (sigegss の場合)。 DOUBLE PRECISION (digegss の場合)。 配列: $a(lda,*)$ 、 $b(ldb,*)$ 。 配列 $a$ には、行列 $\mathbf{A}$ を格納する。 配列 $b$ には、行列 $\mathbf{B}$ を格納する。この行列の列は、連立方程式の右辺である。
<i>lda</i>	INTEGER。a の第 1 次元。 $lda \geq \max(1, n)$ 。



<code>ldb</code>	INTEGER。 $b$ の第 1 次元。 $ldb \geq \max(1, n)$ 。
<code>encl</code>	REAL (sigegss の場合 )。 DOUBLE PRECISION (digegss の場合 )。 配列 : <code>encl(ldb,*)</code> 。 配列 <code>encl</code> は、このルーチンが推定する解集合の部分境界を境界付ける区間箱を定義する。
<code>epsilon</code>	REAL (sigegss の場合 )。 DOUBLE PRECISION (digegss の場合 )。 推定の指示精度。
<code>nits</code>	INTEGER。 割り当てた Gauss-Seidel 反復法の回数 ( $nits \geq 0$ )。

## 出力パラメータ

<code>b</code>	解の行列 $X$ のエンクロージャによって上書きされる。
<code>info</code>	INTEGER。 <code>info = 0</code> の場合、実行は正常に終了したことを示す。 <code>info = i &gt; 0</code> の場合、行列の対角成分 $a(i, i)$ はゼロを含む。ルーチンの実行は失敗ではないが、行列の行や列を交換し、その主対角からゼロを含む成分を除外することが推奨される。 <code>info = -i</code> の場合、 $i$ 番目のパラメータ値が不正だったことを示す。

## アプリケーション・ノート

区間 Gauss-Seidel 反復法は、区間線形方程式の局所ソルバである。すなわち、この処理は  $\mathbf{R}^n$  空間において与えられた区間箱で境界付けられた解集合の部分のエンクロージャを計算することを目的とするものである。

?gegss によって区間線形方程式に対する全体の解の集合のエンクロージャを計算する場合は、その最初の ( 自然の ) エンクロージャを `encl` 引数で指定しなければならない。

## ?gehbs

区間線形連立方程式を *Hansen-Bliek-Rohn* プロシージャによって解く。

---

### 構文

```
call sigehbs( trans, n, a, lda, b, ldb, info)
call digehbs( trans, n, a, lda, b, ldb, info)
```

### 説明

?gehbs ルーチンは、区間 *Hansen-Bliek-Rohn* 法を使用して、以下の区間線形連立方程式に対する解集合のエンクロージャを計算する。

$\mathbf{A}\mathbf{X} = \mathbf{B}$  ( $trans = 'N'$  の場合)

$\mathbf{A}^T \mathbf{X} = \mathbf{B}$  ( $trans = 'T'$  または  $'C'$  の場合)

このルーチンを使用したコードの例は、本書付録 C の [区間連立 1 次方程式ソルバのコード例](#) を参照のこと。

### 入力パラメータ

<i>trans</i>	CHARACTER(1)。'N'、'T'、'C'、'n'、't'、または 'c' のいずれかでなければならない。 連立方程式の形式を指定する。 $trans = 'N'$ または 'n' の場合、 $\mathbf{A}\mathbf{X} = \mathbf{B}$ を $\mathbf{X}$ について解く。 $trans = 'T'$ 、'C'、't'、'c' のいずれかの場合、 $\mathbf{A}^T \mathbf{X} = \mathbf{B}$ を $\mathbf{X}$ について解く。
<i>n</i>	INTEGER。A の次数。B の行の数 ( $n \geq 0$ )。
<i>a, b</i>	REAL (sigehbs の場合)。 DOUBLE PRECISION (digehbs の場合)。 配列 <i>a</i> ( <i>lda</i> , *)、 <i>b</i> ( <i>ldb</i> , *)。 配列 <i>a</i> には、行列 <b>A</b> を格納する。 配列 <i>b</i> には、行列 <b>B</b> を格納する。この行列の列は、連立方程式の右辺である。
<i>lda</i>	INTEGER。a の第 1 次元。 $lda \geq \max(1, n)$ 。
<i>ldb</i>	INTEGER。b の第 1 次元。 $ldb \geq \max(1, n)$ 。

## 出力パラメータ

*b* 解の行列  $X$  によって上書きされる。

*info* INTEGER。  
*info* = 0 の場合、実行は正常に終了したことを示す。  
*info* > 0 の場合、実行は正常に終了しなかったことを示す。  
*info* = -*i* の場合、*i* 番目のパラメータの値が不正だったことを示す。

## アプリケーション・ノート

$A$  の中点行列が対角行列に近くない場合には、区間線形方程式を修正し、より良い結果を得るために、[?qemip](#) ルーチンによる中点逆転プリコンディショニングが必要と考えられる。

## 区間方程式の精密な解を求めるためのルーチン

### ?gepps

区間線形連立方程式をパラメータ分割法によって解く。

#### 構文

```
call sigepps( trans, n, a, lda, b, ldb, cmpt, mode, estm, epsilon, nits,
             info)
call digepps( trans, n, a, lda, b, ldb, cmpt, mode, estm, epsilon, nits,
             info)
```

#### 説明

?gepps ルーチンは、パラメータ分割法 (PPS) を使用して、以下の区間線形連立方程式に対する解集合のいくつかの (またはすべての) 精密な外部の成分に関する推定を計算する。

$AX = B$  (*trans* = 'N' の場合)

$A^T X = B$  (*trans* = 'T' または 'C' の場合)

## 入力パラメータ

<i>trans</i>	<p>CHARACTER(1)。'N'、'T'、'C'、'n'、't'、または 'c' のいずれかでなければならない。</p> <p>連立方程式の形式を指定する。</p> <p><i>trans</i> = 'N' または 'n' の場合、<math>\mathbf{AX} = \mathbf{B}</math> を <math>\mathbf{X}</math> について解く。</p> <p><i>trans</i> = 'T'、'C'、't'、'c' のいずれかの場合、<math>\mathbf{A}^T \mathbf{X} = \mathbf{B}</math> を <math>\mathbf{X}</math> について解く。</p>
<i>n</i>	INTEGER。 $\mathbf{A}$ の次数。 $\mathbf{B}$ の行の数 ( $n \geq 0$ )。
<i>a, b</i>	<p>REAL (sigepps の場合)。</p> <p>DOUBLE PRECISION (digepps の場合)。</p> <p>配列: <i>a</i> (<i>lda</i>, *), <i>b</i> (<i>ldb</i>)。</p> <p>配列 <i>a</i> には、行列 <math>\mathbf{A}</math> を格納する。</p> <p>配列 <i>b</i> には、行列 <math>\mathbf{B}</math> を格納する。この行列の列は、連立方程式の右辺である。</p>
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $lda \geq \max(1, n)$ 。
<i>ldb</i>	INTEGER。 <i>b</i> の第 1 次元。 $ldb \geq \max(1, n)$ 。
<i>cmpt</i>	INTEGER。推定する解集合の成分の数。
<i>mode</i>	<p>CHARACTER(1)。「L」または「U」(または対応する小文字)でなければならない。</p> <p>パラメータ <i>cmpt</i> で指定された座標に沿う解集合の推定方法を示す。</p> <p><i>mode</i> = 'L' または 'l' の場合、ルーチンは <i>cmpt</i> 番目の座標での解集合の推定下端を計算する。</p> <p><i>mode</i> = 'U' または 'u' の場合、ルーチンは <i>cmpt</i> 番目の座標での解集合の推定上端を計算する。</p>
<i>epsilon</i>	<p>REAL (sigepps の場合)。</p> <p>DOUBLE PRECISION (digepps の場合)。</p> <p>推定の指示精度。</p>
<i>nits</i>	INTEGER。割り当てた PPS アルゴリズムの反復回数 ( $nits \geq 0$ )。

## 出力パラメータ

<i>epsilon</i>	<p>REAL (sigepps の場合)。</p> <p>DOUBLE PRECISION (digepps の場合)。</p> <p><i>cmpt</i> 番目の座標軸に沿う解集合の推定値。<i>mode</i> = 'L' の場合、<i>estm</i> は解集合の推定下端を表す。<i>mode</i> = 'U' の場合、<i>estm</i> は解集合の推定上端を表す。</p>
----------------	--

<i>epsilon</i>	推定の実際の精度。
<i>nits</i>	INTEGER。実際に実行した PPS アルゴリズムの反復回数。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = <i>i</i> > 0 の場合、実行は正常に終了しなかったことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正だったことを示す。

## アプリケーション・ノート

指示した精度内に収まっていることが保証されるエンクロージャを求めることと同様に、区間線形連立方程式に対する解集合の最適な (あるいは精密な) エンクロージャを計算することは、NP 困難な問題として知られている。したがって、?gepps ルーチンを効率良く使用して目的の結果を得るには、パラメータ *epsilon* と *nits* を適切に選択することがきわめて重要である。

この事情を念頭に置き、全体のソリューション・プロセスを対話式に構成することが推奨される。すなわち、?gepps ルーチンの呼び出しを、適度な *nits* と粗い *epsilon* で始め、?gepps が実行を完了するまで、*nits* を増加、*epsilon* を減少して繰り返す。

### 例 12-2 パラメータ分割法 (PPS) の Fortran 90 コード

第 1 座標軸に沿って、(例えば、1.E-4 以内の精度で) 区間線形連立方程式に対する解集合の精密な推定下限を計算するサンプル問題を考える。

$$\begin{pmatrix} 3.5 & [0,2] & [0,2] \\ [0,2] & 3.5 & [0,2] \\ [0,2] & [0,2] & 3.5 \end{pmatrix} \mathbf{x} = \begin{pmatrix} [-1,1] \\ [-1,1] \\ [-1,1] \end{pmatrix}$$

この問題は、パラメータ分割法 (PPS 法) を実装し、sigepps ルーチンを使用する以下の Fortran コードで解くことができる。

```

.....
USE INTERVAL_ARITHMETIC
.....
INTEGER, PARAMETER :: LDA = 3, LDB = 3
INTEGER              :: NITS, CMPT, INFO, I, J
CHARACTER(1)         :: MODE = 'L'

```

```
REAL(4)                :: EPS, ESTM
TYPE(S_INTERVAL)       :: A(3,3), B(3)
.....
DO I = 1, 3
    DO J = 1, 3
        IF( I/=J ) THEN
            A(I,J) = SINTERVAL(0.,2.)
        ELSE
            A(I,J) = SINTERVAL(3.5)
        END IF
        B(I) = SINTERVAL(-1.,1.)
    END DO
END DO
CMPT = 1
NITS = 100
Eps= 1.E-4
.....
CALL SIGEPPS( 'n', 3, A, LDA, B, LDB, CMPT, MODE, ESTM, EPS, NITS, INFO )
```

-----

アルゴリズムが確実に完了するように、パラメータ **NITS** の値は 100 ( 回 ) とした。上記の例では、この値で十分である。行列 **A** と右辺ベクトル **B** の成分への単精度区間の割り当ては、INTERVAL\_ARITHMETIC モジュールで供給される SINTERVAL 関数によって実行される点に注意すること。

## 区間行列逆転用のルーチン

---

### ?trtri

三角区間行列の逆区間行列を計算する。

---

#### 構文

```
call sitrtri( uplo, diag, n, a, lda, info )
```

```
call ditrtri( uplo, diag, n, a, lda, info )
```

## 説明

?trtri ルーチンは、区間三角行列  $\mathbf{A}$  の逆行列  $\mathbf{A}^{-1}$  の区間エンクロージャを計算する。

このルーチンは、後方置換アルゴリズムを実装し、元の行列の単純な構造により、逆区間行列の最適なエンクロージャを生成する。

## 入力パラメータ

**uplo** CHARACTER(1)。'U'、'L'、'u'、または 'l' のいずれかでなければならない。  
 $\mathbf{A}$  が上三角行列か、下三角行列かを指定する。  
 uplo = 'U' または 'u' の場合、 $\mathbf{A}$  は上三角行列である。  
 uplo = 'L' または 'l' の場合、 $\mathbf{A}$  は下三角行列である。

**diag** CHARACTER(1)。  
 'N'、'U'、'n'、または 'u' のいずれかでなければならない。  
 diag = 'N' または 'n' の場合、 $\mathbf{A}$  は単位三角行列ではない。  
 diag = 'U' または 'u' の場合、 $\mathbf{A}$  は単位三角行列である。 $\mathbf{A}$  の対角成分は 1 とみなされ、配列 a 内で参照されない。

**n** INTEGER。行列  $\mathbf{A}$  の次数 ( $n \geq 0$ )。

**a** REAL (sitrttri の場合)。  
 DOUBLE PRECISION (ditrttri の場合)。  
 配列: 次元は (lda, \*)。  
 行列  $\mathbf{A}$  を格納する。  
 a の第 2 次元は、 $\max(1, n)$  以上でなければならない。

**lda** INTEGER。a の第 1 次元。lda  $\geq \max(1, n)$ 。

## 出力パラメータ

**a** 逆行列  $\mathbf{A}^{-1}$  のエンクロージャとなる  $n \times n$  の区間行列によって上書きされる。

**info** INTEGER。  
 info = 0 の場合、実行は正常に終了したことを示す。  
 info = -i の場合、i 番目のパラメータの値が不正だったことを示す。  
 info = i の場合、 $\mathbf{A}$  の i 番目の対角成分がゼロで、 $\mathbf{A}$  が特異になるため、行列の反転を完了できなかったことを示す。

## ?geszi

*Schulz* 反復法によって逆区間行列を計算する。

---

### 構文

```
call sigeszi( n, a, lda, info )  
call digeszi( n, a, lda, info )
```

### 説明

一般の区間正方行列  $A$  に対して、?geszi ルーチンは、*Schulz* 反復法により逆区間行列  $A^{-1}$  のエンクロージャを計算する。このルーチンを使用したコードの例は、本書付録 C の [区間連立 1 次方程式ソルバのコード例](#) を参照のこと。

### 入力パラメータ

<i>n</i>	INTEGER。行列 $A$ の次数 ( $n \geq 0$ )。
<i>a</i>	REAL (sigeszi の場合)。 DOUBLE PRECISION (digeszi の場合)。 配列 : 次元は ( <i>lda</i> , *)。 行列 $A$ を格納する。 <i>a</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $lda \geq \max(1, n)$ 。

### 出力パラメータ

<i>a</i>	逆区間行列 $A^{-1}$ のエンクロージャによって上書きされる。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = <i>i</i> > 0 の場合、実行は正常に終了しなかったことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正だったことを示す。

### アプリケーション・ノート

?geszi ルーチンで使用されている *Schulz* 反復は区間行列  $A$  が「広すぎる」ことがない条件でのみ収束する。逆に、*Schulz* 反復が発散するとき、結果は [ -*infy*, +*infy* ] 要素の区間行列に等しいとされる。ここで、*infy* は、コンピュータ上で対応する数値表現での最大値である。



## 区間行列の特性確認用のルーチン

### ?gerbr

区間行列の正則性を *Ris-Beeck* と *Rex-Rohn* 判定基準によってテストする。

#### 構文

```
call sigerbr( n, a, lda, sr, reg, info )
call digerbr( n, a, lda, sr, reg, info )
```

#### 説明

?gerbr ルーチンは、*Ris-Beeck* のスペクトル判定基準と *Rex-Rohn* テストを組み合わせ使用して、一般区間正方行列  $\mathbf{A}$  が正則か特異かを確認する。

#### 入力パラメータ

$n$                     INTEGER。行列  $\mathbf{A}$  の次数 ( $n \geq 0$ )。

$a$                     REAL (sigerbr の場合)。  
                     DOUBLE PRECISION (digerbr の場合)。  
                     配列: 次元は ( $lda, *$ )。  
                     行列  $\mathbf{A}$  を格納する。  
                      $a$  の第 2 次元は、 $\max(1, n)$  以上でなければならない。

$lda$                    INTEGER。  $a$  の第 1 次元。  $lda \geq \max(1, n)$ 。

#### 出力パラメータ

$sr$                    REAL (sigerbr の場合)。  
                     DOUBLE PRECISION (digerbr の場合)。  
                     行列  $(|\operatorname{mid} \mathbf{A}|^{-1} \cdot \operatorname{rad} \mathbf{A})$  のスペクトル半径の推定上限。  
                     これは行列  $\mathbf{A}$  についての付加情報であるが、 $\mathbf{A}$  のいわゆる強固な正則性にとって非常に重要なものである。

*reg*                    INTEGER。特異性テストの結果を示す。  
                       $reg > 0$  の場合、**A** は正則である。  
                       $reg < 0$  の場合、**A** は特異である。  
                       $reg = 0$  の場合、結果は不定である。すなわち、テストは行列 **A**  
                      が正則か特異かを検出できるほど感度が高くなかった。

*info*                    INTEGER。  $info = 0$  の場合、実行は正常に終了したことを示す。  
                       $info = i > 0$  の場合、実行は正常に終了しなかったことを示す。  
                       $info = -i$  の場合、 $i$  番目のパラメータの値が不正だったことを  
                      示す。

## アプリケーション・ノート

?gerbr ルーチンに実装されているテストは大まかなものであり、境界上の場合には行列 **A** をさらに調査することが推奨される。しかし、このルーチンは ( $sr$  の値を 1 と比較することで) その区間行列が強固に正則かどうかを判定するのに役立つ。

---

## ?gesvr

区間行列の正則性 / 特異性を *Rump* と *Rex-Rohn*  
特異値判定基準によってテストする。

---

### 構文

```
call sigesvr( n, a, lda, msr, rsr, reg, info )  
call digesvr( n, a, lda, msr, rsr, reg, info )
```

### 説明

?gesvr ルーチンは、*Rump* と *Rex-Rohn* 特異値判定基準を使用して、一般区間正方行列 **A** が正則か特異かを確認する。

### 入力パラメータ

*n*                    INTEGER。行列 **A** の次数 ( $n \geq 0$ )。

*a*                    REAL (sigesvr の場合)。  
                      DOUBLE PRECISION (digesvr の場合)。  
                      配列 : 次元は ( $lda, *$ )。  
                      行列 **A** を格納する。  
                      *a* の第 2 次元は、 $\max(1, n)$  以上でなければならない。

*lda* INTEGER。 *a* の第 1 次元。  $lda \geq \max(1, n)$ 。

## 出力パラメータ

*msr, rsr* S\_INTERVAL (*sigesvr* の場合 )。  
D\_INTERVAL (*digesvr* の場合 )。  
行列 **A** に関するその他の情報。  
これらの区間はそれぞれ中点行列と半径行列の特異スペクトルの範囲を示す。

*reg* INTEGER。特異性テストの結果を示す。  
*reg* > 0 の場合、**A** は正則である。  
*reg* < 0 の場合、**A** は特異である。  
*reg* = 0 の場合、結果は不定である。すなわち、テストは行列 **A** が正則か特異かを検出できるほど感度が高くなかった。

*info* INTEGER。 *info* = 0 の場合、実行は正常に終了したことを示す。  
*info* = *i* > 0 の場合、実行は正常に終了しなかったことを示す。  
*info* = -*i* の場合、*i* 番目のパラメータの値が不正だったことを示す。

## アプリケーション・ノート

?gesvr ルーチンは、行列が正則か特異かに関する十分条件のみをテストするものである。これは、いくつかの境界上では、行列が正則か特異かを検出できるほどテストの感度が高くないことを意味する。この場合、ルーチンは *reg* = 0 を返す。

## 例 12-3 特異値判定基準による区間行列の正則性をテストする Fortran 90 のコード

---

区間行列

$$\begin{pmatrix} [2,4] & [-1,2] \\ [-2,1] & [2,4] \end{pmatrix}$$

の正則性を特異値判定基準によってテストするには、以下の Fortran 90 コードの断片が参考になる。

```
-----  
.....  
USE INTERVAL_ARITHMETIC  
.....  
INTEGER, PARAMETER :: LDA = 2, N = 2  
TYPE(D_INTERVAL)    :: A(LDA,N), MSR, RSR  
INTEGER              :: REG, INFO  
.....  
A(1,1) = DINTERVAL(2.,4.); A(1,2) = DINTERVAL(-2.,1.)  
A(2,1) = DINTERVAL(-1.,2.); A(2,2) = DINTERVAL(2.,4.)  
.....  
CALL DIGESVR( N, A, LDA, MSR, RSR, REG, INFO )  
-----
```

実軸上の区間 MSR と RSR の相互関係は、正則性にどの程度の余裕があるか (MSR > RSR の場合)、あるいは行列が正則なものからどの程度かけ離れているか (MSR < RSR の場合) の基準として、ある程度まで使用できる。

## 補助およびユーティリティ・ルーチン

### ?gemip

区間線形方程式の midpoint 逆転プリコンディショニングを実行する。

#### 構文

```
call sigemip( n, nrhs, a, lda, b, ldb, info )
call digemip( n, nrhs, a, lda, b, ldb, info )
```

#### 説明

?gemip ルーチンは、区間線形方程式  $\mathbf{Ax} = \mathbf{B}$  の midpoint 逆転プリコンディショニングを実行する。この処理は、 $\mathbf{A}$  と  $\mathbf{B}$  の両行列に、( 丸め付き ) 区間演算の midpoint 逆行列  $(\text{mid } \mathbf{A})^{-1}$  を乗算して行われる。

#### 入力パラメータ

<i>n</i>	INTEGER。行列 $\mathbf{A}$ の次数。
<i>nrhs</i>	INTEGER。右辺の数 ( $nrhs \geq 0$ )。
<i>a, b</i>	REAL (sigemip の場合 ) DOUBLE PRECISION (digemip の場合 ) 配列 : <i>a</i> ( <i>lda</i> , *), <i>b</i> ( <i>ldb</i> , *)。 配列 <i>a</i> には、行列 $\mathbf{A}$ を格納する。 配列 <i>b</i> には、行列 $\mathbf{B}$ を格納する。この行列の列は、連立方程式の右辺である。 <i>a</i> の第 2 次元は $\max(1, n)$ 以上でなければならない。 <i>b</i> の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $lda \geq \max(1, n)$ 。
<i>ldb</i>	INTEGER。 <i>b</i> の第 1 次元。 $ldb \geq \max(1, n)$ 。

#### 出力パラメータ

<i>a</i>	プリコンディショニング済の行列 $\mathbf{A}$ によって上書きされる。
<i>b</i>	プリコンディショニング済の行列 $\mathbf{B}$ によって上書きされる。

INTEGER<sub>0</sub>

*info* = 0 の場合、実行は正常に終了したことを示す。

$info > 0$  の場合、実行は正常に終了しなかったことを示す。

`info = -i` の場合、`i` 番目のパラメータの値が不正だったことを示す。

## アプリケーション・ノート

プリコンディショニングによって、区間線形方程式を解くアルゴリズムの適用範囲が拡張したり、そのアルゴリズムで生成される結果の質が向上する可能性がある。

特に、区間ガウス法、区間 Householder 法、および区間 Gauss-Seidel 反復法を、対角の比重が大きい「広い」行列を持つ区間線形方程式に適用する場合は、より良い結果を得るため、その前にプリコンディショニングをしておくべきである。

Hansen-Bliek-Rohn プロシージャについては、方程式の中点行列が対角からかけ離れている場合、中点逆転プリコンディショニングを実施することが推奨される。

### 例 12-4 区間線形方程式のプリコンディショニング用の Fortran 90 コード

以下の Fortran コードは、区間線形方程式

$$\begin{pmatrix} [2, 4] & [-2, 1] \\ [-1, 2] & [2, 4] \end{pmatrix}_X = \begin{pmatrix} [0, 2] & [-2, 2] \\ [0, 2] & [-2, 2] \end{pmatrix}$$

に対してプリコンディショニングを実施した上で区間ガウス法で解く方法を示している。

```

.....
USE INTERVAL_ARITHMETIC
.....
TYPE(D_INTERVAL)      :: A(2,2), B(2,2)
INTEGER                :: N = 2, NRHS = 2, LDA = 2, LDB = 2, INFO
CHARACTER(1)           :: TRANS = 'n'
.....
A(1,1) = DINTERVAL(2.,4.);  A(1,2) = DINTERVAL(-2.,1.)
A(2,1) = DINTERVAL(-1.,2.); A(2,2) = DINTERVAL(2.,4.)
B(1,1) = DINTERVAL(0.,2.);  B(2,1) = DINTERVAL(0.,2.)

```

```
B(1,2) = DINTERVAL(-2.,2.); B(2,2) = DINTERVAL(-2.,2.)
```

```
.....
```

```
CALL DIGEMIP( N, NRHS, A, LDA, B, LDB, INFO )
```

```
CALL DIGEGAS( TRANS, N, NRHS, A, LDA, B, LDB, INFO )
```

```
-----
```

このルーチンを使用したコードの他の例は、本書付録 C の [区間連立 1 次方程式ソルバのコード例](#)を参照のこと。





# 線形ソルバの基礎

A

科学工学分野の多くのアプリケーションでは、連立 1 次方程式を解く必要がある。この問題は通常、行列とベクトルによる式  $Ax = b$  で数学的に表現される。ここで、 $A$  は  $m \times n$  の行列、 $x$  は  $n$  成分の列ベクトル、 $b$  は  $m$  成分の列ベクトルである。行列  $A$  は通常、係数行列と呼ばれる。ベクトル  $x$  と  $b$  は、それぞれ、解ベクトルと右辺と呼ばれる。

スパース行列を使用した連立 1 次方程式の解法に関する基本的な概念は、この後の「[スパース連立 1 次方程式](#)」セクションで説明する。

行列  $A$  の係数とベクトル  $b$  の右辺が確定できず、既知の区間に属している場合、方程式は区間線形方程式と呼ばれる。区間線形方程式の解法に使用される基本的な定義と概念は、この後の「[区間線形方程式](#)」セクションで説明する。

## スパース連立 1 次方程式

現実の多くのアプリケーションでは、 $A$  の多くの成分はゼロである。このような行列は、スパース行列（疎行列）と呼ばれる。逆に、ゼロの成分が少ない行列は、密行列と呼ばれる。スパース行列では、行列が疎であることを活用できれば、方程式  $Ax = b$  の解を計算することで、容量と計算時間の両方の面で効率を上げることができる。アルゴリズムが正確さを犠牲にすることなく、疎であることを活用できれば、アルゴリズムはより良くなる。

一般に、連立 1 次方程式の解を求めるソフトウェアは、ソルバと呼ばれる。特に、スパース線形方程式を解くために設計されたソルバは、スパースソルバと呼ばれる。ソルバは通常、反復ソルバと直接ソルバの 2 つのグループに分類される。

**反復ソルバ：** 解の初期近似で開始し、近似と真の解との差を推定する。その差をもとに、反復ソルバは初期近似より真の解により近い新しい近似を計算する。この処理過程は、近似と真の解との差が十分小さくなるまで繰り返される。反復ソルバの主な欠点は、収束率が行列  $A$  の値に大きく依存することである。したがって、解を得るまでの所要時間

を予測することができない。事実、悪条件の行列では、反復処理は全く解に収束しない。しかし、好条件の行列では、反復ソルバは非常に速く解に収束する可能性がある。したがって、適切なアプリケーションでは、反復ソルバは非常に有効である。

**直接ソルバ:** 行列  $A$  を 2 つの三角行列の積に分解し、その上で順方向と逆方向から三角行列を解く。

この方法では、行列の大きさをもとに、連立 1 次方程式を解くために要する時間を予測できる。実際、スパース行列では、配列  $a$  の非ゼロ成分の数をもとに計算時間を予測できる。

## 行列の基礎事項

行列は、実数または複素数の長方形の配列である。行列は大文字で示され、その成分は同じ文字の小文字で行と列の添字を付けて示される。したがって、行列  $A$  の  $i$  行  $j$  列の成分の値は  $a(i, j)$  と示される。

例えば、3 行 4 列の行列  $A$  は、次のように記述される。

$$A = \begin{bmatrix} a(1, 1) & a(1, 2) & a(1, 3) & a(1, 4) \\ a(2, 1) & a(2, 2) & a(2, 3) & a(2, 4) \\ a(3, 1) & a(3, 2) & a(3, 3) & a(3, 4) \end{bmatrix}$$

上の表記では、配列のインデックスは、0 から始まる C プログラム言語表記ではなく、1 から始まる標準 Fortran プログラム言語表記であると仮定している。

すべての成分が実数の行列は、実行列と呼ばれる。1 つでも複素数を含む行列は、複素行列と呼ばれる。

実 / 複素行列  $A$  でプロパティが  $a(i, j) = a(j, i)$  のものは、対称行列と呼ばれる。実 / 複素行列  $A$  でプロパティが  $a(i, j) = \text{conj}(a(j, i))$  のものは、エルミート行列と呼ばれる。対称 / エルミート行列を扱うプログラムでは、格納されていない成分の値は格納された値から素早く再構成できるため、行列の半分の値を格納するだけでよい。

行数と列数が同じ行列は、正方行列と呼ばれる。正方行列で行インデックスと列インデックスが同じ成分は、行列の対角成分、または単に行列の対角と呼ばれる。

行列  $A$  の転置とは、配列の成分を対角で「反転」させて得られる行列である。すなわち、成分  $a(i, j)$  と  $a(j, i)$  を交換する。複素行列で、対角で成分を反転してから成分の複素共役をとった場合、結果の行列は元の行列のエルミート転置または共役転置と呼ばれる。行列  $A$  の転置およびエルミート転置は、それぞれ、 $A^T$  および  $A^H$  で示される。

列ベクトル ( または単にベクトル ) は  $n \times 1$  の行列であり、行ベクトルは  $1 \times n$  の行列である。

すべて非ゼロのベクトル  $x$  でベクトル - 行列の積  $x^T A x$  がゼロより大きい場合、実 / 複素行列  $A$  は正定値である。正定値ではない行列は不定値と呼ばれる。

上 ( 下 ) 三角行列は、対角の下 ( 上 ) のすべての成分がゼロである正方行列である。単位三角行列は、対角がすべて 1 である上あるいは下三角行列である。

任意の行列  $A$  で、行列積の結果  $PA$  が  $A$  の行の交換を除いて  $A$  と同一である場合、行列  $P$  は、置換行列と呼ばれる。正方行列の場合、 $PA$  が  $A$  の行の置換行列であれば、 $AP^T$  は  $A$  の同じ列の置換行列である。さらに  $P$  の逆行列は  $P^T$  となる。

スペースを節約するため、置換行列は通常、行列としてではなく、置換ベクトルと呼ばれる線形配列として格納される。特に、置換行列が行列の  $i$  番目の行を  $j$  番目の行にマップする場合、置換ベクトルの  $i$  番目の成分は  $j$  となる。

対角上にもみ非ゼロ成分を持つ行列は、対角行列と呼ばれる。置換行列の場合と同様に、それは行列としてではなく、ベクトルとして格納される。

## 直接法

直接法を用いるソルバでは、方程式  $Ax = b$  の解を求める基本的な技法は、最初に  $A$  を三角行列に因子分解することである。すなわち、 $A = LU$  となるような下三角行列  $L$  と上三角行列  $U$  を求めることである。そのような因子分解 ( 通常  $LU$  分解あるいは  $LU$  因子分解と呼ばれる ) を用いることで、元の問題の解は次のように記述できる。

$$\begin{aligned} Ax &= b \\ \Rightarrow LUx &= b \\ \Rightarrow (Ux) &= b \end{aligned}$$

この結果、次に示す 2 ステップのプロセスで元の方程式の解を求めることになる。

1. 方程式  $Ly = b$  を解く。
2. 方程式  $Ux = y$  を解く。

方程式  $Ly = b$  と  $Ux = y$  を解くことはそれぞれ、順方向で解く、逆方向で解く、と呼ばれる。

対称行列  $A$  が正定値の場合、 $A$  は  $LL^T$  として因子分解できる。ここで  $L$  は下三角行列である。同様に、エルミート行列  $A$  が正定値の場合、 $A = LL^H$  として因子分解できる。対称 / エルミート行列では、この形式の因子分解は、コレスキー因子分解と呼ばれる。

コレスキー因子分解で、 $LU$  分解の行列  $U$  は、 $L^T$  または  $L^H$  である。したがって、 $L$  と  $A$  の半分のみを格納すればよく、 $U$  の計算も省けるため、ソルバの効率を上げることができる。そのため、正定値方程式の解としてアプリケーションを表現できるユーザは、一般的な場合に比べて、大幅な効率向上が可能である。

対称 (またはエルミート) であるが正定値ではない行列についても、まだかなりの効率向上の余地がある。 $A$  が対称であるが正定値ではない場合、 $A$  は  $A = LDL^T$  として因子分解できる。ここで、 $D$  は対角行列、 $L$  は下単位三角行列である。同様に、 $A$  がエルミートの場合、 $A = LDL^H$  として因子分解できる。どちらの場合も、 $L$ 、 $D$ 、と  $A$  の半分のみを格納すればよく、 $U$  を計算する必要はない。ただし、逆方向に解く場合は、 $L^T x = y$  ではなく、 $L^T x = D^{-1} y$  を解く必要がある。

## スパース行列のフィルインとリオーダーリング

スパース方程式を解くことに関連した 2 つの重要な概念は、フィルインとリオーダーリングである。次の例で、これらの概念を説明する。

連立 1 次方程式  $Ax = b$  を考える。ここで、 $A$  は対称正定値のスパース行列であり、次のように定義される。

$$A = \begin{bmatrix} 9 & \frac{3}{2} & 6 & \frac{3}{4} & 3 \\ \frac{3}{2} & \frac{1}{2} & * & * & * \\ 6 & * & 12 & * & * \\ \frac{3}{4} & * & * & \frac{5}{8} & * \\ 3 & * & * & * & 16 \end{bmatrix} \quad b = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{bmatrix}$$

星印(\*)はゼロを表し、 $A$  が疎であることを強調している。 $A$  のコレスキー因子分解は  $A = LL^T$  とする。ここで、 $L$  は次のようになる。

$$L = \begin{bmatrix} 3 & * & * & * & * \\ \frac{1}{2} & \frac{1}{2} & * & * & * \\ 2 & -2 & 2 & * & * \\ \frac{1}{4} & \frac{1}{-4} & \frac{1}{-2} & \frac{1}{2} & * \\ 1 & -1 & -2 & -3 & 1 \end{bmatrix}$$

行列  $A$  が相対的に疎であるにもかかわらず、下三角行列  $L$  は対角より下にゼロがないことに注意する。 $L$  を計算してから、順方向および逆方向に解く段階で使用すると、 $A$  が密である場合と同じ計算量となる。

$A$  がゼロの場所に  $L$  が非ゼロの成分を持つことは、フィルインと呼ばれる。計算量の面からは、ソルバが  $A$  の非ゼロ構造を活用して  $L$  の計算でフィルインを減らせれば、より効率が向上する。この結果、ソルバは  $L$  の非ゼロの箇所のみ計算すればよいことになる。この目的に沿って、 $A$  の行と列の置換を考える。「[行列の基礎事項](#)」セクションで説明したように、 $A$  の行の置換は、置換行列  $P$  で表される。行の置換の結果は、 $P$  と  $A$  の積となる。上記の例で、 $A$  の最初の行と 5 番目行を交換し、 $A$  の最初の列と 5 番目の列を交換し、結果が行列  $B$  になるとする。数学的には、 $A$  の行と列を置換して  $B$  を得る処理を  $B = PAP^T$  として表現できる。 $A$  の行と列を置換すると、 $B$  は次のようになる。

$$B = \begin{bmatrix} 16 & * & * & * & 3 \\ * & \frac{1}{2} & * & * & \frac{3}{2} \\ * & * & 12 & * & 6 \\ * & * & * & \frac{5}{8} & \frac{3}{4} \\ 3 & \frac{3}{2} & 6 & \frac{3}{4} & 9 \end{bmatrix}$$

$B$  は  $A$  の行と列を入れ替えただけなので、 $A$  と  $B$  の非ゼロ成分の数は同じである。しかし、コレスキー因子分解  $B = LL^T$  を求めると、次のようになる。

$$L = \begin{bmatrix} 4 & * & * & * & * \\ * & \frac{1}{\sqrt{2}} & * & * & * \\ * & * & 2(\sqrt{3}) & * & * \\ * & * & * & \frac{\sqrt{10}}{4} & * \\ \frac{3}{4} & \frac{3}{\sqrt{2}} & \sqrt{3} & \frac{3}{\sqrt{10}} & \frac{\sqrt{3}}{4} \end{bmatrix}$$

$B$  を元にしたフィルインは、 $A$  を元にしたフィルインよりも少ない。したがって、 $B$  の因子分解に必要な容量と計算時間は  $A$  の因子分解よりも少なくなる。このことから、効率のよいスパースソルバは、行列  $A$  の置換  $P$  を求めて、 $B = PAP^T$  の因子分解でのフィルインを最小にする必要があることがわかる。そして  $B$  の因子分解を元の方程式を解くのに使用する。

上記の例は対称正定値行列とコレスキー分解をもとにしたが、同様の手法は一般の  $LU$  分解にも有効である。具体的には、 $P$  を置換行列として、 $B = PAP^T$  であり、 $B$  は  $B = LU$  と因子分解できる。すると、

$$Ax = b$$

$$\Rightarrow PA(P^{-1}P)x = Pb$$

$$\Rightarrow PA(P^T P)x = Pb$$

$$\Rightarrow (PAP^T)(Px) = Pb$$

$$\Rightarrow B(Px) = Pb$$

$$\Rightarrow LU(Px) = Pb$$

したがって、 $B$  に対する  $LU$  分解が得られれば、元の方程式を次の 3 ステップのプロセスで解くことができる。

1.  $Ly = Pb$  を解く。
2.  $Uz = y$  を解く。
3.  $x = P^T z$  とする。

この 3 ステップのプロセスを上例に適用する場合、最初に、連立 1 次方程式  $LY = Pb$  を順方向に解く必要がある。

$$\begin{bmatrix} 4 & * & * & * & * \\ * & \frac{1}{\sqrt{2}} & * & * & * \\ * & * & 2(\sqrt{3}) & * & * \\ * & * & * & \frac{\sqrt{10}}{4} & * \\ \frac{3}{4} & \frac{3}{\sqrt{2}} & \sqrt{3} & \frac{3}{\sqrt{10}} & \frac{\sqrt{3}}{4} \end{bmatrix} * \begin{bmatrix} y1 \\ y2 \\ y3 \\ y4 \\ y5 \end{bmatrix} = \begin{bmatrix} 5 \\ 2 \\ 3 \\ 4 \\ 1 \end{bmatrix}$$

結果は、 $y^T = \frac{5}{4}, 2\sqrt{2}, \frac{\sqrt{3}}{2}, \frac{16}{\sqrt{10}}, \frac{-979\sqrt{3}}{12\sqrt{5}}$  になる。

2 番目のステップでは、 $Uz = y$  を逆方向から解く。または、コレスキー因子分解を用いて、 $L^T z = y$  を解く。

$$\begin{bmatrix} 4 & * & * & * & \frac{3}{4} \\ * & \frac{1}{\sqrt{2}} & * & * & \frac{3}{\sqrt{2}} \\ * & * & 2(\sqrt{3}) & * & \sqrt{3} \\ * & * & * & \frac{\sqrt{10}}{4} & \frac{3}{\sqrt{10}} \\ * & * & * & * & \frac{\sqrt{3}}{4} \end{bmatrix} * \begin{bmatrix} z1 \\ z2 \\ z3 \\ z4 \\ z5 \end{bmatrix} = \begin{bmatrix} \frac{5}{4} \\ 2(\sqrt{2}) \\ \frac{\sqrt{3}}{2} \\ \frac{16}{\sqrt{10}} \\ \frac{-979\sqrt{3}}{12\sqrt{5}} \end{bmatrix}$$

結果は、 $z = \frac{123}{2}, 983, \frac{1961}{12}, 398, \frac{-979}{3}$  になる。

最後のステップでは、 $x = P^T z$  を計算する。結果は、 $x^T = \frac{-979}{3}, 983, \frac{1961}{12}, 398, \frac{123}{2}$  になる。

## スパース行列の格納形式

これまで説明したように、スパース行列の非ゼロ成分のみを格納すると効率が向上する。これは、疎の程度が大きい、すなわち、非ゼロ成分が非常に少ないと仮定しているからである。逆に、ゼロ成分が非常に少ない場合、スパース行列用の処理は、単純に行列を密として処理する、すなわちゼロも非ゼロもすべての値を計算に用いる場合よりも遅くなってしまう。

スパース行列用を使用する一般的な格納方式はいくつかあるが、ほとんどの方式が同じ基本的な技法を用いている。つまり、行列のすべての非ゼロ成分を線形配列に圧縮し、補助配列を使用して、元の行列における非ゼロの位置を格納する。

### PARDISO ソルバの格納形式

スパース行列  $A$  の非ゼロ成分の線形配列への圧縮は、各列 (列主体形式) または各行 (行主体形式) を順番に調べて、見つかった非ゼロ成分を順番に線形配列に書き込むことで行われる。

対称行列を格納するときは、行列の上半分の三角のみ (上三角形式) または行列の下半分の三角のみ (下三角形式) を格納すればよい。

インテル MKL の直接法スパースソルバは、行主体の上三角格納形式を採用している。すなわち、行列は行ごとに圧縮され、対称行列は行列の上三角のみが格納される。

スパース行列用の PARDISO ソフトウェアで利用できるインテル MKL の格納形式は、`values`、`columns`、および `rowIndex` 配列と呼ばれる 3 つの配列からなる。次の表は、スパース行列  $A$  の非ゼロ成分の値、行、および列の位置の点から配列を説明している。

`values`      $A$  の非ゼロ成分を含む実数または複素数配列。 $A$  の非ゼロの値は、上で説明した行主体の上三角格納マッピングを使用して `values` 配列にマップされる。

`columns`     整数配列 `columns` の  $i$  番目の成分は、`values(i)` の値を含む  $A$  の列の番号を格納する。

`rowIndex`   整数配列 `rowIndex` の  $j$  番目の成分は、 $A$  の行  $j$  の最初の非ゼロ成分を含む `values` 配列へのインデックスを格納する。

`values` と `columns` 配列の長さは  $A$  の非ゼロの個数に等しい。

`rowIndex` 配列は行で最初の非ゼロの位置を示し、さらに非ゼロが続けて格納されるため、 $i$  番目の行の非ゼロの数を `rowIndex(i)` と `rowIndex(i+1)` の差として計算できる。



この関係を  $A$  の最後の行まで保持するため、 $rowIndex$  の最後に 1 つの成分 (ダミー成分) を追加し、 $A$  の非ゼロの数に 1 を加えた値を設定する。これにより、 $rowIndex$  配列の合計の長さは、 $A$  の行数より 1 大きくなる。



**注：**インテル MKL のスパース格納方式では、配列のインデックスは、0 から始まる C プログラム言語表記ではなく、1 から始まる標準 Fortran プログラム言語表記を使用している。

上記を念頭において、以前のセクションの例で使った、次の対称行列を格納することを考える。

$$A = \begin{bmatrix} 9 & \frac{3}{2} & 6 & \frac{3}{4} & 3 \\ * & \frac{1}{2} & * & * & * \\ * & * & \frac{1}{2} & * & * \\ * & * & * & \frac{5}{8} & * \\ * & * & * & * & 16 \end{bmatrix}$$

この場合、 $A$  には 9 個の非ゼロ成分があるので、 $values$  と  $columns$  配列の長さは 9 である。また、行列  $A$  には 5 つの行があるので  $rowIndex$  配列の長さは 6 である。この例では、各配列の実際の値は次のようになる。

**表 A-1**      対称行列の例に対する格納配列

$values$	=	(9   3/2   6   3/4   3   1/2   1/2   5/8   16)
$columns$	=	(1   2   3   4   5   2   3   4   5)
$rowIndex$	=	(1   6   7   8   9   10)

非対称または非エルミート配列では、すべての非ゼロを格納する必要がある。次のように定義される非対称行列  $B$  を考える。

$$B = \begin{bmatrix} 1 & -1 & * & -3 & * \\ -2 & 5 & * & * & * \\ * & * & 4 & 6 & 4 \\ -4 & * & 2 & 7 & * \\ * & 8 & * & * & -5 \end{bmatrix}$$

$B$  には 13 個の非ゼロがあり、次のように  $B$  を格納する。

**表 A-2 非対称行列の例に対する格納配列**

<code>values</code>	=	(1 -1 -3 -2 5 4 6 4 -4 2 7 8 -5)
<code>columns</code>	=	(1 2 4 1 2 3 4 5 1 3 4 2 5)
<code>rowIndex</code>	=	(1 4 6 9 12 14)

インテル MKL の現在のバージョンでは、直接法スパースソルバで非対称の連立方程式を解くことはできない。しかし、対称構造の連立方程式は解くことができる。対称構造の連立方程式とは、非ゼロのパターンが対称のものである。すなわち、 $a(i,j)$  が非ゼロの場合に  $a(j,i)$  も非ゼロの場合にのみ、行列は対称構造である。ソルバ・ソフトウェアの観点からすると、行列の非ゼロ成分は、`values` 配列に格納されるものすべてである。これはつまり、`values` 配列に注意深くゼロを追加することにより、任意の非対称行列を対称構造の行列に変換できることを意味する。例えば、行列  $B$  に次のような非ゼロ成分が含まれているとする。

$$B = \begin{bmatrix} 1 & -1 & * & -3 & * \\ -2 & 5 & * & * & 0 \\ * & * & 4 & 6 & 4 \\ -4 & * & 2 & 7 & * \\ * & 8 & 0 & * & -5 \end{bmatrix}$$

ここで、 $B$  は 15 の非ゼロ成分を含む対称構造であると考えられる。

**表 A-3 対称構造の行列の例に対する格納配列**

<code>values</code>	=	(1 -1 -3 -2 5 0 4 6 4 -4 2 7 8 0 -5)
<code>columns</code>	=	(1 2 4 1 2 5 3 4 5 1 3 4 2 3 5)
<code>rowIndex</code>	=	(1 4 7 10 13 16)

**格納形式の制約：**スパースソルバ用の格納形式は、2つの重要な制約に従わなければならない。

1) 行の非ゼロの値は、行で非ゼロが存在する順に (左から右に) *values* 配列に配置しなければならない。2) 対称または対称構造の行列で、対角成分を *values* 配列から省略してはならない。

2の制約は、対角上にゼロを含む対称または対称構造の行列を扱う場合、ゼロ対角成分を *values* 配列で (省略しないで) 明示的に表現しなければならないことを意味する。

### スパース BLAS レベル 2-3 のスパース格納形式

このセクションでは、インテル MKL の現在のバージョンでサポートしているスパース BLAS レベル 2 および 3 のスパースデータ構造の詳細について説明する。

#### CSR 形式

スパース行列用のインテル MKL の圧縮スパース行 (CSR) 形式は、*values*、*columns*、*pointerB*、および *pointerE* 配列と呼ばれる 4 つの配列からなる。次の表は、スパース行列 *A* の非ゼロ成分の値、行、および列の位置の点から配列を説明している。

*values*     *A* の非ゼロ成分を含む実数または複素数配列。*A* の非ゼロの値は、上で説明した行主体格納マッピングを使用して *values* 配列にマップされる。

*columns*     整数配列 *columns* の *i* 番目の成分は、*values*(*i*) の値を含む *A* の列の番号を格納する。

*pointerB*     この整数配列の *j* 番目の成分は、*A* の行 *j* の最初の非ゼロ成分を含む *values* 配列へのインデックスを格納する。このインデックスは *pointerB*(*j*) - *pointerB*(1)+1 に等しいことに注意。

*pointerE*     この整数配列は、*A* の行 *j* の最後の非ゼロ成分を含む *values* 配列へのインデックス *pointerE*(*j*)-*pointerB*(1) のような行インデックスを格納する。

*values* および *columns* 配列の長さは *A* の非ゼロの個数に等しい。*pointerB* と *pointerE* の配列の長さは *A* の行数に等しい。

前に定義した行列  $B$

$$B = \begin{bmatrix} 1 & -1 & * & -3 & * \\ -2 & 5 & * & * & * \\ * & * & 4 & 6 & 4 \\ -4 & * & 2 & 7 & * \\ * & 8 & * & * & -5 \end{bmatrix}$$

は、CSR 形式では次のように表される。

**表 A-4 例の行列に対する格納配列 (CSR 形式)**

<code>values</code>	=	(1 -1 -3 -2 5 4 6 4 -4 2 7 8 -5)
<code>columns</code>	=	(1 2 4 1 2 3 4 5 1 3 4 2 5)
<code>pointerB</code>	=	(1 4 6 9 12)
<code>pointerE</code>	=	(4 6 9 12 14)

この格納形式は、NIST スペース BLAS ライブラリ [\[Rem05\]](#) で使用されている。

PARDISO ソフトウェアで利用できる、上で記述した格納形式 ([PARDISO ソルバの格納形式](#)を参照) は、CSR 形式を変形したものである。PARDISO 形式では、すべての非ゼロ成分が連続して格納されなければならないという制約があった。すなわち、 $J$  行の非ゼロ成分の集合は、 $J-1$  行の非ゼロ成分の集合の直後になければならない。

CSR 形式にはそのような制約はない。これは、例えば、同時に異なる部分行列を処理する必要があるときに有用である。この場合、必要なそれぞれの部分行列に対して `pointerB` と `pointerE` の配列を定義して、それらの配列を 1 つの `values` 配列へのポインタとすればよい。

[表 A-2](#) の `rowIndex` 配列と [表 A-4](#) の `pointerB` および `pointerE` 配列を比較すると、  
 $pointerB(i) = rowIndex(i) \ (i=1, \dots, 5)$   
 $pointerE(i) = rowIndex(i+1) \ (i=1, \dots, 5)$

となっていることが解る。

このことから、`values`、`columns`、`pointerB` および `pointerE` を入力パラメータとして持つルーチンは PARDISO に採用された形式で格納されたスパース行列に対しても適用できる可能性がある。例えば、次のインタフェースを持つルーチン

```
Subroutine name_routine(... , values, columns, pointerB, pointerE, ...)
は、引数 values、columns、rowIndex を使用して次のように呼び出すことができる。
call name_routine(... , values, columns, rowIndex, rowindex(2), ...)
```



**注：**インテル MKL のスパース BLAS レベル 2 ライブラリでは、CSR 形式の両方の変形に対するルーチンを用意している。

## CSC 形式

圧縮スパース列形式 (CSC、*Harwell-Boeing* スパース行列形式と呼ばれることもある) は CSR に似た形式であるが、行の代わりに列が使用される。つまり、CSC 形式は転置行列に対する CSR 形式に等しい。

CSR 形式の場合と同様に、インテル MKL のスパース BLAS レベル 2 ライブラリでは、CSC 形式の両方の型に対するルーチンを用意している。

PARDISO ソフトウェアで利用できる、この形式の変形は、*values*、*row*、および *colIndex* 配列と呼ばれる 3 つの配列からなる。次の表で、これらの配列を説明する。

*values*     *A* の非ゼロ成分を含む実数または複素数配列。*A* の非ゼロの値は、上で説明した列主体格納マッピングを使用して *values* 配列にマップされる。

*rows*       整数配列 *rows* の *i* 番目の成分は、*values*(*i*) の値を含む *A* の行の番号を格納する。

*colIndex*   整数配列 *colIndex* の *j* 番目の成分は、*A* の列 *j* の最初の非ゼロ成分を含む *values* 配列へのインデックスを格納する。

*values* と *rows* 配列の長さは *A* の非ゼロの個数に等しい。

例えば、次のスパース行列 *B*

$$B = \begin{bmatrix} 1 & -1 & * & -3 & * \\ -2 & 5 & * & * & * \\ * & * & 4 & 6 & 4 \\ -4 & * & 2 & 7 & * \\ * & 8 & * & * & -5 \end{bmatrix}$$

は、PARDISO 用の CSC 形式では次のように表される。

**表 A-5 例の行列に対する格納配列 (Harwell-Boeing 形式)**

```
values = (1 -2 -4 -1 5 8 4 2 -3 6 7 4 -5)
rows   = (1 2 4 1 2 5 3 4 1 3 4 3 5)
colIndex = (1 4 7 9 12 14)
```

## 座標形式

座標形式は、スパース行列の表現として最も柔軟で簡潔な形式である。非ゼロ成分のみが対象とされ、各非ゼロ成分の座標が明示的に示される。多くの商用ライブラリは、座標形式でスパース行列用の行列 - ベクトル乗算をサポートしている。

インテル MKL の座標形式は、*values*、*rows*、および *column* 配列と呼ばれる 3 つの配列と、*A* の非ゼロ成分の数を示すパラメータ *nnz* からなる。3 つの配列の次元はすべて *nnz* である。次の表は、スパース行列 *A* の非ゼロ成分の値、行、および列の位置の点から配列を説明している。

*values* 任意の順序で *A* の非ゼロ成分を含む実数または複素数配列。

*rows* 整数配列 *rows* の *i* 番目の成分は、*values*(*i*) の値を含む *A* の行の番号を格納する。

*columns* 整数配列 *columns* の *i* 番目の成分は、*values*(*i*) の値を含む *A* の列の番号を格納する。

例えば、次のスパース行列 *C*

$$C = \begin{bmatrix} 1 & -1 & -3 & 0 & 0 \\ -2 & 5 & 0 & 0 & 0 \\ 0 & 0 & 4 & 6 & 4 \\ -4 & 0 & 2 & 7 & 0 \\ 0 & 8 & 0 & 0 & -5 \end{bmatrix}$$

は、座標形式では次のように表される。

**表 A-6 例の行列に対する格納配列 (座標形式)**

```
values = (1 -1 -3 -2 5 4 6 4 -4 2 7 8 -5)
rows   = (1 1 1 2 2 3 3 3 4 4 4 5 5)
columns = (1 2 3 1 2 3 4 5 1 3 4 2 5)
```

**対角格納方式**

行列  $A$  に対角が含まれている場合、その構造を使用して非ゼロ成分の位置に必要な情報を削減できる。この格納方式は、行列が有限成分あるいは差分離散化から発生するようなアプリケーションにおいて特に有用である。インテル MKL の対角格納方式は、`values` と `distance` 配列と呼ばれる 2 つの配列、空でない対角の数を示すパラメータ `ndiag`、および呼び出す (サブ) プログラムで宣言されるリーディング・ディメンジョン `lval` からなる。次の表で、配列 `values` と `distance` を説明する。

- values**     実数または複素数の 2 次元配列。次元は `lval` と `ndiag`。  $A$  の非ゼロ対角を含む。この格納形式の重要な点は、`values` の各成分が元の行列の行に対応する行を保持していることである。このために、 $A$  の下三角部分にある対角は上から、上三角にある対角は下からパディングされる。`distance(i)` の値は、対角  $i$  でパディングされる成分の数である。
- distance**   整数配列。次元は `ndiag`。整数配列 `distance` の  $i$  番目の成分は、 $i$  番目の対角と主対角との間の距離を格納する。この距離は、対角が主対角の上にあるときは正、主対角の下にあるときは負である。主対角の距離はゼロである。

上で説明したスパース行列  $C$  は、対角格納方式では次のように格納される。

`distance = (-3 -1 0 1 2)`

$$values = \begin{bmatrix} * & * & 1 & -1 & -3 \\ * & -2 & 5 & 0 & 0 \\ * & 0 & 4 & 6 & 4 \\ -4 & 2 & 7 & 0 & * \\ 8 & 0 & -5 & * & * \end{bmatrix}$$

ここでアスタリスクはパディングされた成分を示す。

行列が対称、エルミート、またはスキュー対称の場合、上三角または下三角が格納されることは明らかである。

スパース対角表現をインテル MKL のスパース行列 - 行列 (または行列 - ベクトル) 乗算ルーチンで使用する場合、対角は任意の順序で格納できる。しかし、スパース対角表現をインテル MKL のスパース三角ソルバルーチンで使用する場合、`distance` 配列の全成分は昇順に格納されていなければならない。

## スカイライン格納方式

スカイライン格納方式は、直接法スパースソルバにおいて重要である。また、ピボット演算を必要としないコレスキー分解や LU 分解にも適している。

インテル MKL で利用できるスカイライン格納方式は、三角行列または行列の三角部分のみを格納できる。この変形は、*values* と *pointers* 配列と呼ばれる 2 つの配列からなる。次の表で、これらの配列を説明する。

**values** スカラ配列。行列が下三角の場合、最初の非ゼロ成分から対角成分までの (対角成分を含む)  $A$  の各行の成分を格納する。行列が上三角の場合、最初の非ゼロ成分から対角成分までの (対角成分を含む)  $A$  の各列の成分を格納する。途中のゼロ成分も含まれる。

**pointers** 整数配列。次元は  $m+1$ 。ここで、 $m$  は下三角では行 (上三角では列) の数である。 $pointers(i)-pointers(1)+1$  は、行 (列)  $i$  の最初の非ゼロ成分の *values* における位置を指す。 $pointers(m+1)$  の値は、値  $nnz+pointers(1)$  に設定される。ここで、 $nnz$  は配列 *values* の成分の数である。

インテル MKL のスパース BLAS は、スカイライン格納形式で処理するルーチンで一般行列をサポートしていない点に注意。

例えば、上で説明した行列  $C$  の上三角は次のように格納される。

```
values = (1 -2 5 4 -4 0 2 7 8 0 0 -5)
pointers = (1 2 4 5 9 13)
```

また、行列  $C$  の下三角は次のように格納される。

```
values = (1 -1 5 -3 0 4 6 7 4 0 -5)
pointers = (1 2 4 7 9 12)
```

この格納形式は、NIST スパース BLAS ライブラリ [\[Rem05\]](#) で使用されている。

## 区間線形方程式

### 区間

区間とは、実数軸  $\mathbf{R}$  の密で連続した部分集合である。したがって、それは 2 つの数値すなわちその下端と上端 (それぞれ、左端と右端と呼ばれる場合もある) で完全に定義でき、 $[a, b]$  で区間  $\{x \in \mathbf{R} | a \leq x \leq b\}$  を表記できる。すべての実数区間の集合は  $\mathbf{IR}$  で表記する。数学的表記法では、区間の上端および下端を取り出すには次のように表記する。

$\inf[a, b] = a$  、  $\sup[a, b] = b$  。



以下の説明で、区間と区間オブジェクトは太字で表記し、下線と上線で区間  $\mathbf{x} = [\underline{x}, \bar{x}]$  の下端と上端を示す。

各区間は次の中点、

$$\text{mid } \mathbf{a} = \frac{1}{2}(\bar{\mathbf{a}} + \underline{\mathbf{a}})$$

および半径、

$$\text{rad } \mathbf{a} = \frac{1}{2}(\bar{\mathbf{a}} - \underline{\mathbf{a}})$$

で一意的に決定できる。後者は幅  $\text{wid } \mathbf{a} = (\bar{\mathbf{a}} - \underline{\mathbf{a}})$  と等価である。 $[a, a]$  の形で下端と上端が等しいもの、すなわち、幅 0 の区間は、退化した区間、点の区間、または薄い区間と呼ばれ、通常の実数と一致する。したがって  $\mathbf{R} \subset \mathbf{IR}$  と言える。逆に、幅が 0 でない区間は、厚い区間と呼ばれる。

区間は集合なので、例えば、包含、共通集合などの集合論での相互の関係、操作が適用できる。特に点  $t \in \mathbf{R}$  は、 $\mathbf{a} \leq t \leq \bar{\mathbf{a}}$  の場合、区間  $\mathbf{a}$  の成分である ( $t \in \mathbf{a}$  と書く)。さらに、包含は次のように定義できる。 $\underline{\mathbf{a}} \geq \underline{\mathbf{b}}$  かつ  $\bar{\mathbf{a}} \leq \bar{\mathbf{b}}$  の場合に限り  $\mathbf{a} \subseteq \mathbf{b}$  である。

区間と区間オブジェクト (ベクトル、行列、その他) は、ある変数の変動可能な下端と上端のみが既知であるとき、いわゆる境界のある不確定性とあいまい度を表現するのに便利なツールである。この意味で、区間は、定量的不確定性を記述する確率論やファジー論のアプローチと並んでその代換案を提供するものである。

加算、減算、乗算、および除算といった算術的な操作も、次の基本原理に従って区間に拡張できる。

$$\mathbf{a} * \mathbf{b} := \{a * b \mid a \in \mathbf{a}, b \in \mathbf{b}\}, \quad * \in \{+, -, \cdot, /\}, \quad (1)$$

これはいわゆる古典区間算術を定義することを可能にする。空の区間  $[\emptyset]$  がしばしばコンピュータ区間算術構造に導入されることに注意。

## 区間ベクトルと行列

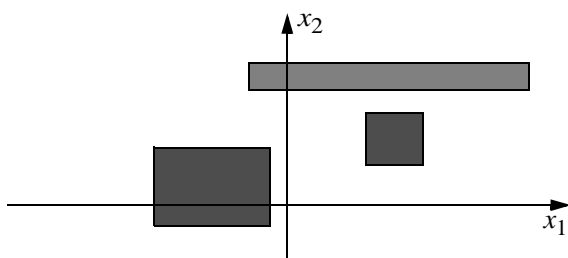
区間ベクトルとは垂直 (列ベクトル) あるいは水平 (行ベクトル) に並べた順序付きの区間の組である。そこで  $a_1$ 、 $a_2$ 、...、 $a_n$  が区間とすると、

$$a = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} \text{ は列ベクトルであり、}$$

そして、

$a = (a_1, a_2, \dots, a_n)$  は行ベクトルである。

すべての区間  $n$  次ベクトルの集合は、 $\mathbf{IR}^n$  と表記される。



区間ベクトルは、側面が座標軸と並行であるという幾何学的なイメージ (すなわち、区間  $\mathbf{R}^n$  の長方形の箱) で連想できる。この理由から、区間ベクトルはしばしばボックスと呼ばれる。

区間行列は、区間からなる長方形の表である。

$$A := \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix},$$

あるいは  $A = (a_{ij})$ 。区間ベクトルは、大きさが  $n \times 1$  (列ベクトル) または  $1 \times n$  (行ベクトル) のいずれかの区間行列と同一である。区間  $m \times n$  行列の集合は  $\mathbf{IR}^{m \times n}$  と表記される。区間ベクトルと行列の間の算術演算は、(1) 式を一般化する関係に基づいて導入できる ([Alefeld83](#)、[Neumaier90](#) を参照)。

区間正方行列  $A \in \mathbf{IR}^{n \times n}$  は、すべての点行列  $A \in A$  が正則 (非特異) すなわち、非ゼロの決定要素を持つ場合に限り、**正則** (非特異) と呼ばれる。そうでない場合、区間行列  $A \in \mathbf{IR}^{n \times n}$  は**特異**と呼ばれ、少なくとも 1 つの特異な点行列を含むことを意味する。

一般に、区間行列が正則か特異かを見分けることは、この問題を妥当な時間内に解く比較的単純 (多項式の複雑度) なアルゴリズムがないため、NP 困難な問題とされている。

実用的な必要性からは、広い範囲の区間行列に対して正則性を調べるための、いくつかの実行可能で十分な基準を持つことが重要である。インテル MKL には、区間行列の正則性 / 特異性をテストするために **Ris-Beeck** スペクトル判定、**Rump** 特異値判定、および **Rohn-Rex** 特異値判定を実装したルーチンが用意されている。

ときどき、区間行列について関連するプロパティ (**強固な正則性**) を調べる必要が生じる。強固な正則性には、区間行列とその中点逆行列との積が正則であることが求められる。`?gerbr` ルーチンは、その出力パラメータ `sr` の値を判定して強固な正則性を調べることができる。

## 区間線形方程式

次の形式の線形代数方程式

$$\left\{ \begin{array}{l} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1, \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2, \\ \quad \quad \quad \cdot \quad \quad \quad \cdot \quad \quad \quad \cdot \\ \quad \quad \quad \cdot \quad \quad \quad \cdot \quad \quad \quad \cdot \\ \quad \quad \quad \cdot \quad \quad \quad \cdot \quad \quad \quad \cdot \\ a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n = b_m, \end{array} \right. \quad (2)$$

あるいは、簡潔に、

$$Ax = b$$

(ここで  $A$  は  $m \times n$  次元の行列、 $b$  は  $m$  次のベクトルである) を解くことは、科学工学分野における主要問題の 1 つである。 $a_{ij}$  と  $b_i$  が確定できず、それぞれ、区間  $a_{ij}$  と  $b_i$  に属している場合、方程式は**区間線形方程式**と呼ばれ、次のように記述される。

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1, \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2, \\ \quad \quad \quad \cdot \quad \quad \quad \cdot \quad \quad \quad \cdot \\ \quad \quad \quad \cdot \quad \quad \quad \cdot \quad \quad \quad \cdot \\ \quad \quad \quad \cdot \quad \quad \quad \cdot \quad \quad \quad \cdot \\ a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nn}x_n = b_n, \end{cases} \quad (3)$$

ここで、 $a_{ij}$  と  $b_i$  は区間である。簡潔な形で記述すると次のようになる。

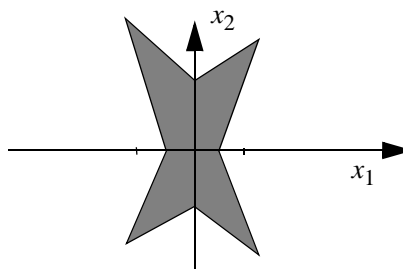
$$Ax = b \quad (4)$$

ここで、区間行列  $A = (a_{ij})$  および右辺ベクトル  $b = (b_i)$  である。  
区間線形方程式 (3)–(4) は、同じ形の点線形方程式  $Ax = b$  の集合とも考えられる。ここで、パラメータ  $a_{ij}$  と  $b_i$  は  $a_{ij} \in a_{ij}$  と  $b_i \in b_i$  を満たす。

$a_{ij}$  と  $b_i$  が区間  $a_{ij}$  と  $b_i$  内で変化するとき、 $A = (a_{ij})$  および  $b = (b_i)$  の対応する点方程式  $Ax = b$  の解は  $\mathbf{R}^n$  空間での集合を形成する。すなわち、

$$\Xi(A, b) := \{x \in \mathbf{R}^n \mid (\exists A \in A)(\exists b \in b)(Ax = b)\}. \quad (5)$$

(5) の集合は、 $A \in A$  および  $b \in b$  のすべての点方程式  $Ax = b$  の解から形成されており、区間線形方程式 (3)–(4) の解集合と呼ばれる。通常、個々の  $a_{ij}$  と  $b_i$  ( $1 \leq i, j \leq n$ ) の解集合は  $\mathbf{R}^n$  における多面体であり、ときには下の図のように星形をしている。





**注：**区間線形方程式には、様々な他の解集合が存在するため、上に記述された集合  $\Xi(A, b)$  は、しばしば、**結合解集合**と呼ばれる ([Shary02](#)を参照)。

解集合の複雑さは次元  $n$  のべき乗で増加するため、その正確な記述は次元  $n$  が大きくなると現実的には不可能である。また、ほとんどの場合、そのような正確な記述は実際には必要ない。通常は、単に解集合の**推定**を計算するだけでよい。実際に最も一般的に使用されるのは、次に述べる外からの (超集合による) 区間推定問題である。

区間線形方程式  $Ax = b$  について、  
解集合  $\Xi(A, b)$  の区間エンクロージャを求めよ。 (6)

しばしば、次のように問題 (6) の成分ごとの形が考えられる。

区間線形方程式  $Ax = b$  について、  
下から  $\min\{x_v \mid x \in \Xi(A, b)\}$  の推定を求めよ。 (7)  
上から  $\max\{x_v \mid x \in \Xi(A, b)\}$  の推定を求めよ ( $v = 1, 2, \dots, n$ )。

特に、インテル MKL の `?gepps` ルーチンは、この問題設定で処理される。

問題 (6)–(7) は、現代の区間解析において最もポピュラーな問題の 1 つである。この問題に関する多数の文献がある (例えば、[Alefeld83](#)、[Kearfott96](#)、[Neumaier90](#))。

つまり、区間線形方程式を解くことは、ここでは区間線形方程式 (3)–(4) の解集合に対する外からの区間推定を計算することと理解されている。方程式の行列  $A$  は通常、正方で非特異であると仮定される。

古典計算線形代数とは異なり、区間線形方程式は一般に、計算が非常に困難であることがわかっている。(6) の解の最適 (最小) 区間エンクロージャを計算すること、または等価的に (7) の解集合の正確な推定を計算することは、方程式の区間の幅や行列  $A$  の非ゼロ成分の構造に制限がない場合は、NP 困難な問題である ([Kreinovich97](#) を参照)。さらに、解への要求を緩和し、精度が前もって定義した絶対的 / 相対的な精度内になるように解集合の推定を計算したとしても、問題はやはり NP 困難なままである。

実際的な見地から NP 困難が意味するところは、一般的な問題では、高い確率で、問題のサイズに関して多項式時間で解くことができないことである。

この理由により、インテル MKL では、区間線形方程式用の数値計算アルゴリズムは、結果の精度が保証されているかどうかによって 2 つのクラスに分かれる。「高速」アルゴリズムは、高速に妥当な時間で解集合のエンクロージャを計算するが、精度の保証はない。「最適」あるいは「正確」アルゴリズムは、実行の完了に時間がかかるが、得られた結果は精度の要求を満たすものである。

インテル MKL には、両方の種類のアルゴリズムを実装した区間ソルバルーチンが用意されている。例えば、区間 Gauss 法、区間 Householder 法、Hansen-Blik-Rohn 法、および Krawczyk 反復のような高速手法は、それぞれ、?gegas、?gehss、?gehbs、および ?gekws ルーチンとして実装されている。パラメータ分割法 (PPS 法) は、正確な手法の例として ?gepps ルーチンで実装されている。?trtrs ルーチンは、非常に特殊な行列構造のため、両方のカテゴリに含まれる。

## 前処理

区間線形方程式 (4) の前処理は、システムの特性を改善するために行列  $A$  と右辺ベクトル  $b$  とを点行列で掛け合わせる。方程式  $Ax = b$  は次の方程式で置き換えられる。

$$(CA)x = Cb$$

ここで  $C$  はある正方点行列である。前処理は、古典計算線形代数において広く使用されており、多くの区間ソルバ・アルゴリズム (例えば、区間ガウス法、区間 Gauss-Seidel 法など) も使用の前に適切な前処理を必要とする。

区間線形方程式用に広く使用されている前処理の 1 つは、中点行列の逆行列による前処理で、しばしば *中点逆転前処理* と呼ばれる。インテル MKL では、中点逆転前処理は ?gemip ルーチンで実装されている。

## 区間行列の逆転

区間正方向行列  $A$  に対して、 $A$  に含まれる全点行列の逆行列の集合のエンクロージャは、*逆区間行列*  $A^{-1}$  と呼ばれる。すなわち、

$$A^{-1} \supseteq \{A^{-1} \mid A \in A\}。$$

古典線形代数では、正方非特異行列  $A$  を持つ線形代数方程式  $Ax = b$  の解は逆行列  $A^{-1}$  と右辺ベクトルの積として表現される。すなわち、 $x = A^{-1}b$ 。

区間解析でも、同様の積  $A^{-1}b$  で区間線形方程式  $\Xi(A, b)$  の解集合  $Ax = b$  へのエンクロージャが求められる。しかし、この方法は通常かなりの過大推定を引き起こすため推奨されない。解集合の外からの推定に特化したプロシージャの使用が好ましい。

それでもなお、逆区間行列の密なエンクロージャを計算することは、方程式の感度解析などには欠かせないものである。

逆区間行列の計算は、次の区間行列方程式の解集合のエンクロージャを見つけることとして実行される。

$AY = I$ 、ここで  $I$  は単位行列で、

区間線形方程式のいずれかの手法を  $n$  回 ( 行列  $Y$  の各列ごとに ) 適用して実行される。

Schulz 法 ([Herzberger94](#) を参照) のような、逆区間行列を見出すための直接反復プロシージャもある。これは、インテル MKL に `geszi` ルーチンとして含まれている。





# ルーチンおよび関数の 引数



BLAS ルーチンの主要な引数は、ベクトルと行列である。一方、VML 関数はベクトル引数のみで動作する。  
この後の各セクションでは、これらの引数それぞれについて、例を挙げながら説明する。

## BLAS のベクトル引数

ベクトル引数は、1 次元の配列で渡される。配列の次元 (長さ) とベクトルの増分は、整数の変数として渡される。長さによって、ベクトル内の成分の数が決まる。増分 (ストライドとも呼ばれる) によって、ベクトルの成分間の間隔と、ベクトルが渡される配列内の成分の順序が決まる。

長さが  $n$  で増分が  $incx$  のベクトルは、その値が次のように定義される 1 次元の配列  $x$  で渡される。

$x(1), x(1+|incx|), \dots, x(1+(n-1)*|incx|)$

$incx$  が正の場合は、配列  $x$  内の成分は昇順に格納される。 $incx$  が負の場合は、配列  $x$  内の成分は降順に格納され、また最初の成分は  $x(1+(n-1)*|incx|)$  として定義される。 $incx$  がゼロの場合は、ベクトルの全成分は同じ値  $x(1)$  をとる。ベクトルを格納する 1 次元配列の次元は、常に、

$idimx = 1 + (n-1)*|incx|$  以上でなければならない。

## 例 B-1 1 次元の実数配列

---

`x(1:7)` が、次の 1 次元の実数配列とする。  
`x = (1.0, 3.0, 5.0, 7.0, 9.0, 11.0, 13.0)`  
`incx = 2` かつ `n = 3` の場合は、このベクトル引数の成分を始めから終わりまで並べると、`(1.0, 5.0, 9.0)` になる。  
`incx = -2` かつ `n = 4` の場合は、このベクトル成分を始めから終わりまで並べると、`(13.0, 9.0, 5.0, 1.0)` になる。  
`incx = 0` かつ `n = 4` の場合は、このベクトル成分を始めから終わりまで並べると、`(1.0, 1.0, 1.0, 1.0)` になる。

行列の 1 次元下部構造 ( 行、列、対角成分など ) は、開始アドレスと増分を指定してベクトル引数として渡せる。Fortran では、 $m$  行  $n$  列の行列は、列主体の順序に基づいて格納される。つまり、同じ列内の成分間の増分は 1、同じ行内の成分間の増分は  $m$ 、同じ対角成分上の成分間の増分は  $m + 1$  になる。

## 例 B-2 2 次元の実行列

---

`a` が、REAL `A(5, 4)` として宣言された  $5 \times 4$  の実行列とする。  
`a` の 3 番目の列を 2.0 でスケーリングするには、BLAS ルーチン `sscal` を次の呼び出しシーケンスで使用する。  
`call sscal (5, 2.0, a(1,3), 1).`  
2 番目の行をスケーリングするには、次の構文を使用する。  
`call sscal (4, 2.0, a(2,1), 5).`  
`A` の主対角成分を 2.0 でスケーリングするには、次の構文を使用する。  
`call sscal (5, 2.0, a(1,1), 6).`



---

**注:** デフォルトのベクトル引数は 1 とみなされる。

---

## VML のベクトル引数

VML 数学関数のベクトル引数は、単位ベクトル増分を使用する 1 次元配列に渡される。つまり、長さ  $n$  のベクトルは、 $a[0]$ ,  $a[1]$ , ...,  $a[n-1]$  として値が定義される配列  $a$  に連続して渡される (C インターフェイスの場合)。  
他の増分を使用する配列、またはより複雑なインデックスを使用する配列を収容するため、VML では、`pack/unpack` 関数を補助的に用意している。`pack/unpack` 関数は、配列の成分を収集して連続するベクトルを生成し、計算の完了後、それらを分散するものである。

一般に、ベクトル成分が 1 次元配列  $a$  に次のように格納されており、

$a[m0]$ ,  $a[m1]$ , ...,  $a[mn-1]$

配列  $y$  に次のように再グループ化する場合、

$y[k0]$ ,  $y[k1]$ , ...,  $y[kn-1]$ ,

VML `pack/unpack` 関数では、以下のいずれかのインデックス方式を使用できる。

### 正増分インデックス

$$kj = \text{incy} * j, \text{mj} = \text{inca} * j, \quad j = 0, \dots, n-1$$

制約:  $\text{incy} > 0, \text{inca} > 0$

例えば、 $\text{incy} = 1$  を設定すると、配列の成分が収集されて連続するベクトルが生成される。

この方式は、負とゼロの増分が許可されない点を除けば、BLAS で使用される方式と同じである。

### インデックス・ベクトル・インデックス

$$kj = iy[j], \text{mj} = ia[j], \quad j = 0, \dots, n-1,$$

$ia$  と  $iy$  は長さ  $n$  の配列であり、それぞれ入力配列  $a$  用のインデックス・ベクトルと出力配列  $y$  用のインデックス・ベクトルを格納する。

### マスク・ベクトル・インデックス

インデックス  $kj$ ,  $mj$  は、以下の条件を満たす。

$my[kj] \neq 0, \text{ma}[mj] \neq 0, \quad j = 0, \dots, n-1,$

ma と my は、それぞれ入力配列 a 用のマスクベクトル、出力配列 y 用のマスクベクトルを格納する配列である。

## 行列引数

インテル<sup>®</sup> マス・カーネル・ライブラリ・ルーチンの行列引数は、以下に挙げる格納形式を使用して、1 次元または 2 次元の配列に格納できる。

- 通常のフル格納 (2 次元配列に格納)
- エルミート行列、対称行列、または三角行列用の圧縮格納 (1 次元配列に格納)
- 帯行列用の帯格納 (2 次元配列に格納)

**フル格納** は、次のような明確な形式である。行列 A は、2 次元配列 a に格納される。このとき、行列の成分  $a_{ij}$  は、配列の成分  $a(i, j)$  に格納される。

行列が、**三角行列** (引数 `uplo` に指定した値に従って、上三角行列または下三角行列) の場合は、関連する三角成分だけが格納される。配列の残りの成分は、設定する必要はない。

対称行列またはエルミート行列を処理するルーチンを使用すると、行列の上三角または下三角のいずれかを、配列の対応する成分に格納できる。

`uplo = 'U'` の場合は、 $i \leq j$  に対して、 $a_{ij}$  が  $a(i, j)$  に格納される。  
a の他の成分は、設定する必要はない。

`uplo = 'L'` の場合は、 $j \leq i$  に対して、 $a_{ij}$  が  $a(i, j)$  に格納される。  
a の他の成分は、設定する必要はない。

**圧縮格納** を使用すると、対称行列、エルミート行列、または三角行列をよりコンパクトに格納できる。関連する三角部分 (この場合も、引数 `uplo` で指定) は、列ごとに 1 次元配列 `ap` にパックされる。

`uplo = 'U'` の場合は、 $i \leq j$  に対して、 $a_{ij}$  が  $ap(i+j(j-1)/2)$  に格納される。

`uplo = 'L'` の場合は、 $j \leq i$  に対して、 $a_{ij}$  が  $ap(i+(2*n-j)*(j-1)/2)$  に格納される。

LAPACK ルーチンの説明では、圧縮形式の行列を保持する配列は、名前の末尾に `p` が付いている。

**帯格納** では、 $k_l$  個の非ゼロの劣対角成分と  $k_u$  個の優対角成分を持つ  $m$  行  $n$  列の帯行列は、 $(k_l+k_u+1)$  行  $n$  列の 2 次元配列 `ab` にコンパクトに格納される。行列の各列は配列の対応する列に、行列の各対角成分は配列の各行にそれぞれ格納される。したがって、

$a_{ij}$  は、 $\max(1, j-k_u) \leq i \leq \min(n, j+k_l)$  に対して、 $ab(k_u+1+i-j, j)$  に格納される。

帯格納形式は、 $k_l$  と  $k_u$  が行列のサイズ  $n$  よりはるかに小さい場合にのみ使用する。  
ルーチンは、 $k_l$  と  $k_u$  がどのような値であっても正常に動作するが、行列が実際に帯になっていない場合は、帯格納を使用すると効率が低下する。

帯格納方式の例を以下に示す ( $m = n = 6$ ,  $k_l = 2$ ,  $k_u = 1$  の場合)。

\* でマークされた配列の成分はルーチンで使用されない。

帯行列 A	A の帯格納
$\begin{bmatrix} a_{11} & a_{12} & 0 & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 & 0 & 0 \\ a_{31} & a_{32} & a_{33} & a_{34} & 0 & 0 \\ 0 & a_{42} & a_{43} & a_{44} & a_{45} & 0 \\ 0 & 0 & a_{53} & a_{54} & a_{55} & a_{56} \\ 0 & 0 & 0 & a_{64} & a_{65} & a_{66} \end{bmatrix}$	$\begin{array}{cccccc} * & a_{12} & a_{23} & a_{34} & a_{45} & a_{56} \\ a_{11} & a_{22} & a_{33} & a_{44} & a_{55} & a_{66} \\ a_{21} & a_{32} & a_{43} & a_{54} & a_{65} & * \\ a_{31} & a_{42} & a_{53} & a_{64} & * & * \end{array}$

一般帯行列を  $LU$  因子分解用に入力として与える場合は、行の交換の結果として埋め込みによって生成される優対角成分をさらに  $k_l$  個だけ格納するためのスペースがなくてはならない。つまり、行列は上記の形式で格納されるが、 $k_l + k_u$  個の優対角成分を持つことにならない。したがって、

$a_{ij}$  は、 $\max(1, j - k_u) \leq i \leq \min(n, j + k_l)$  に対して、 $ab(k_l + k_u + 1 + i - j, j)$  に格納される。

$LU$  因子分解の帯格納方式の例を以下に示す ( $m = n = 6$ ,  $k_l = 2$ ,  $k_u = 1$  の場合)。

帯行列 A	A の帯格納
$\begin{bmatrix} a_{11} & a_{12} & 0 & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 & 0 & 0 \\ a_{31} & a_{32} & a_{33} & a_{34} & 0 & 0 \\ 0 & a_{42} & a_{43} & a_{44} & a_{45} & 0 \\ 0 & 0 & a_{53} & a_{54} & a_{55} & a_{56} \\ 0 & 0 & 0 & a_{64} & a_{65} & a_{66} \end{bmatrix}$	$\begin{array}{cccccc} * & * & * & + & + & + \\ * & * & + & + & + & + \\ * & a_{12} & a_{23} & a_{34} & a_{45} & a_{56} \\ a_{11} & a_{22} & a_{33} & a_{44} & a_{55} & a_{66} \\ a_{21} & a_{32} & a_{43} & a_{54} & a_{65} & * \\ a_{31} & a_{42} & a_{53} & a_{64} & * & * \end{array}$

\* でマークされた配列の成分はルーチンで使用されない。+ でマークされた成分は入力時に設定する必要はないが、結果を格納するために LU 因子分解ルーチンで必要となる。入力配列は、終了時に LU 因子分解の各成分で次のように上書きされる。

$$\begin{array}{cccccc}
 * & * & * & u_{14} & u_{25} & u_{36} \\
 * & * & u_{13} & u_{24} & u_{35} & u_{46} \\
 * & u_{12} & u_{23} & u_{34} & u_{45} & u_{56} \\
 u_{11} & u_{22} & u_{33} & u_{44} & u_{55} & u_{66} \\
 m_{21} & m_{32} & m_{43} & m_{54} & m_{65} & * \\
 m_{31} & m_{42} & m_{53} & m_{64} & * & *
 \end{array}$$

ここで  $u_{ij}$  は上三角行列  $U$  の各成分であり、 $m_{ij}$  は因子分解で使用される乗数である。

三角帯行列は、上三角行列の場合は  $k_l = 0$ 、下三角行列の場合は  $k_u = 0$  として同じ形式で格納される。 $k$  個の劣対角成分または優対角成分を持つ対称帯行列またはエルミート帯行列に対しては、引数 `uplo` で指定したとおりに、上三角行列または下三角行列だけを格納しなければならない。

`uplo = 'U'` の場合は、 $\max(1, j-k) \leq i \leq j$  に対して、 $a_{ij}$  は  $ab(k+1+i-j, j)$  に格納される。  
`uplo = 'L'` の場合は、 $j \leq i \leq \min(n, j+k)$  に対して、 $a_{ij}$  は  $ab(1+i-j, j)$  に格納される。

LAPACK ルーチンの説明では、帯格納で行列を保持する配列は、名前の末尾に  $b$  が付いている。

Fortran では、列を主体とする順序での格納方法が想定されている。つまり、同じ列の成分が続けて格納されることになる。

2 次元配列の引数には、リーディング・ディメンジョン (同じ行内の成分間の格納位置の数を指定する)、行数、列数の 3 つの特性が関連付けられるのが普通である。フル格納の行列に対しては、配列のリーディング・ディメンジョンは、行列内の行数と同じ大きさ以上でなければならない。

文字転置パラメータは、行列引数を通常形式と転置形式のどちらで使用するか、あるいは、複素行列の場合は行列の共役転置を使用するかどうかを指示するために頻繁に使用される。

これら 3 つのケースに対する転置パラメータの値は、次のようになる。

'N' または 'n' 通常形式 ( 共役や転置は使用しない )

'T' または 't' 転置

'C' または 'c' 共役転置

### 例 B-3      2 次元複素配列

A ( 1:5, 1:4 ) は、次の行列で表される 2 次元の複素配列とする。

$$\begin{bmatrix} (1.1, 0.11) & (1.2, 0.12) & (1.3, 0.13) & (1.4, 0.14) \\ (2.1, 0.21) & (2.2, 0.22) & (2.3, 0.23) & (2.4, 0.24) \\ (3.1, 0.31) & (3.2, 0.32) & (3.3, 0.33) & (3.4, 0.34) \\ (4.1, 0.41) & (4.2, 0.42) & (4.3, 0.43) & (4.4, 0.44) \\ (5.1, 0.51) & (5.2, 0.52) & (5.3, 0.53) & (5.4, 0.54) \end{bmatrix}$$

*transa* が転置パラメータ、*m* が行数、*n* が列数、*lda* がリーディング・ディメンジョンとする。このとき、

*transa* = 'N'、*m* = 4、*n* = 2、*lda* = 5 の場合は、行列引数は次のようになる。

$$\begin{bmatrix} (1.1, 0.11) & (1.2, 0.12) \\ (2.1, 0.21) & (2.2, 0.22) \\ (3.1, 0.31) & (3.2, 0.32) \\ (4.1, 0.41) & (4.2, 0.42) \end{bmatrix}$$

*transa* = 'T'、*m* = 4、*n* = 2、*lda* = 5 の場合は、行列引数は次のようになる。

$$\begin{bmatrix} (1.1, 0.11) & (2.1, 0.21) & (3.1, 0.31) & (4.1, 0.41) \\ (1.2, 0.12) & (2.2, 0.22) & (3.2, 0.32) & (4.2, 0.42) \end{bmatrix}$$

*transa* = 'C'、*m* = 4、*n* = 2、*lda* = 5 の場合は、行列引数は次のようになる。

$$\begin{bmatrix} (1.1, -0.11) & (2.1, -0.21) & (3.1, -0.31) & (4.1, -0.41) \\ (1.2, -0.12) & (2.2, -0.22) & (3.2, -0.32) & (4.2, -0.42) \end{bmatrix}$$

リーディング・ディメンジョンの値として、2次元配列の宣言で指定した行数とは異なる値を使用する場合は、特に注意が必要である。例えば、上記の配列 *A* を COMPLEX A (5, 4) として宣言するものとする。

このとき、*transa* = 'N'、*m* = 3、*n* = 4、*lda* = 4 の場合は、行列引数は次のようになる。

$$\begin{bmatrix} (1.1, 0.11) & (5.1, 0.51) & (4.2, 0.42) & (3.3, 0.33) \\ (2.1, 0.21) & (1.2, 0.12) & (5.2, 0.52) & (4.3, 0.43) \\ (3.1, 0.31) & (2.2, 0.22) & (1.3, 0.13) & (5.3, 0.53) \end{bmatrix}$$

---



# コード例



本付録では、インテル<sup>®</sup> マス・カーネル・ライブラリのルーチンおよび関数を使用するコード例を示す。このコード例には、Fortran と C の両方の例が用意されている。

以下のセクションが含まれる。

- [BLAS コードの例](#)
- [PARDISO コードの例](#)
- [直接法スパース・ソルバのコード例](#)
- [反復法スパース・ソルバのコード例](#)
- [DFT コードの例](#)
- [区間連立 1 次方程式ソルバのコード例](#)

関数パラメータと演算に関する詳細は、本マニュアルのそれぞれの章を参照のこと。

## BLAS コードの例

### 例 C-1 BLAS レベル 1 の関数の使用法

---

次の例は、BLAS レベル 1 の関数 `sdot` の呼び出し方法を表す。この関数は、2 つの単精度実ベクトル  $x$  と  $y$  のスカラー積を計算するベクトル-ベクトル演算を実行する。

#### パラメータ

$n$	ベクトル $x$ と $y$ の次数を指定する。
$incx$	$x$ の成分に対する増分を指定する。
$incy$	$y$ の成分に対する増分を指定する。

```
program dot_main
real x(10), y(10), sdot, res
integer n, incx, incy, i
external sdot

n = 5
incx = 2
incy = 1

do i = 1, 10
    x(i) = 2.0e0
    y(i) = 1.0e0
end do

res = sdot (n, x, incx, y, incy)
print*, 'SDOT = ', res
end
```

このプログラムを実行すると、次の行が出力される。

SDOT = 10.000

---

## 例 C-2      BLAS レベル 1 のルーチンの使用法

---

次の例は、BLAS レベル 1 のルーチン `scopy` の呼び出し方法を表す。このルーチンは、2つの単精度実ベクトル  $x$  を  $y$  にコピーするベクトル - ベクトル演算を実行する。

### パラメータ

$n$             ベクトル  $x$  と  $y$  の次数を指定する。  
 $incx$          $x$  の成分に対する増分を指定する。  
 $incy$          $y$  の成分に対する増分を指定する。

```
program copy_main
real x(10), y(10)
integer n, incx, incy, i

n = 3
incx = 3
incy = 1
do i = 1, 10
```

```
      x(i) = i
    end do
    call scopy (n, x, incx, y, incy)
    print*, 'Y = ', (y(i), i = 1, n)
  end
```

このプログラムを実行すると、次の行が出力される。

```
Y = 1.00000 4.00000 7.00000
```

---

### 例 C-3      BLAS レベル 2 のルーチンの使用法

---

次の例は、BLAS レベル 2 のルーチン `sger` の呼び出し方法を表す。このルーチンは、次に示す行列 - ベクトル演算を実行する。

```
a := alpha*x*y' + a
```

#### パラメータ

*alpha*      スカラ *alpha* を指定する。  
*x*            *m* 個の成分を持つベクトル。  
*y*            *n* 個の成分を持つベクトル。  
*a*             $m \times n$  の行列。

```
program ger_main
  real a(5,3), x(10), y(10), alpha
  integer m, n, incx, incy, i, j, lda
  m = 2
  n = 3
  lda = 5
  incx = 2
  incy = 1
  alpha = 0.5
  do i = 1, 10
    x(i) = 1.0
    y(i) = 1.0
  end do
```

```
do i = 1, m
  do j = 1, n
    a(i,j) = j
  end do
end do

call sger (m, n, alpha, x, incx, y, incy, a, lda)
print*, 'Matrix A: '
do i = 1, m
  print*, (a(i,j), j = 1, n)
end do
end
```

このプログラムを実行すると、次に示す行列 *a* が出力される。

行列 A:

1.50000 2.50000 3.50000

1.50000 2.50000 3.50000

---

## 例 C-4      BLAS レベル 3 のルーチンの使用法

---

次の例は、BLAS レベル 3 のルーチン `ssymm` の呼び出し方法を表す。このルーチンは、次に示す行列 - 行列演算を実行する。

*c* := *alpha*\**a*\**b'* + *beta*\**c*

### パラメータ

*alpha*      スカラ *alpha* を指定する。

*beta*      スカラ *beta* を指定する。

*a*      対称行列。

*b*       $m \times n$  の行列。

*c*       $m \times n$  の行列。

```
program symm_main
real a(3,3), b(3,2), c(3,3), alpha, beta
integer m, n, lda, ldb, ldc, i, j
```

```
character uplo, side
uplo = 'u'
side = 'l'
m = 3
n = 2
lda = 3
ldb = 3
ldc = 3
alpha = 0.5
beta = 2.0
do i = 1, m
  do j = 1, m
    a(i,j) = 1.0
  end do
end do
do i = 1, m
  do j = 1, n
    c(i,j) = 1.0
    b(i,j) = 2.0
  end do
end do
call ssymm (side, uplo, m, n, alpha, a, lda, b, ldb, beta, c, ldc)
print*, 'Matrix C: '
do i = 1, m
  print*, (c(i,j), j = 1, n)
end do
end
```

このプログラムを実行すると、次に示す行列  $c$  が出力される。

行列 C:

5.00000 5.00000

5.00000 5.00000

5.00000 5.00000

## 例 C-5 BLAS レベル 1 の複素関数を C から呼び出す

---

次の例は、C プログラムから BLAS レベル 1 の複素関数 `zdotc()` を呼び出す方法を表す。この関数は、2 つの倍精度複素ベクトルの内積を計算する。

この例では、複素数型の内積が構造体 `c` に返される。

```
#define N 5
void main()
{
    int n, inca = 1, incb = 1, i;
    typedef struct{ double re; double im; } complex16;
    complex16 a[N], b[N], c;
    void zdotc();
    n = N;
    for( i = 0; i < n; i++ ){
        a[i].re = (double)i; a[i].im = (double)i * 2.0;
        b[i].re = (double)(n - i); b[i].im = (double)i * 2.0;
    }
    zdotc( &c, &n, a, &inca, b, &incb );
    printf( "The complex dot product is: ( %6.2f, %6.2f )\n", c.re, c.im );
}
```

---



---

**注** :C プログラムから BLAS を直接呼び出す代わりに、CBLAS インターフェイスの使用も可能である。これは、C から BLAS を呼び出す方法としてサポートされている。CBLAS の詳細については、[付録 D](#) を参照。

CBLAS は、インテル MKL に導入された BLAS (Basic Linear Algebra Subprograms) に対する C インターフェイスである。

---

## PARDISO コードの例

このセクションでは、PARDISO 直説法ソルバを使用してスパース行列の連立 1 次方程式を計算するコード例を示す。このソルバの詳細は、本マニュアルの[第 8 章](#)を参照のこと。

### スパース対称連立 1 次方程式の例

このセクションでは、PARDISO を使用して対称連立 1 次方程式の解を算出する 2 つの例 (Fortran および C) を示す。連立方程式  $Ax = b$  を解く。

$$A = \begin{bmatrix} 7.0 & 0.0 & 1.0 & 0.0 & 0.0 & 2.0 & 7.0 & 0.0 \\ 0.0 & -4.0 & 8.0 & 0.0 & 2.0 & 0.0 & 0.0 & 0.0 \\ 1.0 & 8.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 5.0 \\ 0.0 & 0.0 & 0.0 & 7.0 & 0.0 & 0.0 & 9.0 & 0.0 \\ 0.0 & 2.0 & 0.0 & 0.0 & 5.0 & 1.0 & 5.0 & 0.0 \\ 2.0 & 0.0 & 0.0 & 0.0 & 1.0 & -1.0 & 0.0 & 5.0 \\ 7.0 & 0.0 & 0.0 & 9.0 & 5.0 & 0.0 & 11.0 & 0.0 \\ 0.0 & 0.0 & 5.0 & 0.0 & 0.0 & 5.0 & 0.0 & 5.0 \end{bmatrix} \quad \text{および} \quad B = \begin{bmatrix} 1.0 \\ 1.0 \\ 1.0 \\ 1.0 \\ 1.0 \\ 1.0 \\ 1.0 \\ 1.0 \end{bmatrix}$$

### 対称連立 1 次方程式の計算結果の例

ソルバが正常に実行されると、 $X$  の解は次のようになる。

```
Reordering completed ...
Number of nonzeros in factors = 30
Number of factorization MFLOPS = 0
Factorization completed ...
Solve completed ...
The solution of the system is
x(1) = -0.0418602013
x(2) = -0.00341312416
x(3) = 0.117250377
x(4) = -0.11263958
x(5) = 0.0241722445
```

x(6) = -0.10763334

x(7) = 0.198719673

x(8) = 0.190382964

## 例 C-6 対称連立 1 次方程式の例 (pardiso\_sym.f)

---

```
C-----
C Example program to show the use of the "PARDISO" routine
C for symmetric linear systems
C-----
C This program can be downloaded from the following site:
C http://www.computational.unibas.ch/cs/scicomp
C
C (C) Olaf Schenk, Department of Computer Science,
C University of Basel, Switzerland.
C Email: olaf.schenk@unibas.ch
C
C-----

      PROGRAM pardiso_sym
      IMPLICIT NONE

C..Internal solver memory pointer for 64-bit architectures
C..INTEGER*8 pt(64)
C..Internal solver memory pointer for 32-bit architectures
C..INTEGER*4 pt(64)
C..This is OK in both cases
      INTEGER*8 pt(64)
C..All other variables
      INTEGER maxfct, mnum, mtype, phase, n, nrhs, error, msglvl
      INTEGER iparm(64)
      INTEGER ia(9)
      INTEGER ja(18)
      REAL*8 a(18)
      REAL*8 b(8)
      REAL*8 x(8)
```



```
INTEGER i, idum
REAL*8 waltimel, waltime2, ddum
C..Fill all arrays containing matrix data.
DATA n /8/, nrhs /1/, maxfct /1/, mnum /1/
DATA ia /1,5,8,10,12,15,17,18,19/
DATA ja
1 /1, 3, 6,7,
2 2,3, 5,
3 3, 8,
4 4, 7,
5 5,6,7,
6 6, 8,
7 7,
8 8/
DATA a
1 /7.d0, 1.d0, 2.d0,7.d0,
2 -4.d0,8.d0, 2.d0,
3 1.d0, 5.d0,
4 7.d0, 9.d0,
5 5.d0,1.d0,5.d0,
6 -1.d0, 5.d0,
7 11.d0,
8 5.d0/
integer omp_get_max_threads
external omp_get_max_threads
C..
C..Set up PARDISO control parameter
C..
do i = 1, 64
iparm(i) = 0
end do
iparm(1) = 1 !no solver default
iparm(2) = 2 !fill-in reordering from METIS
```

```
iparm(3) = omp_get_max_threads() !numbers of processors, value of OMP_NUM_THREADS
iparm(4) = 0 !no iterative-direct algorithm
iparm(5) = 0 !no user fill-in reducing permutation
iparm(6) = 0 !=0 solution on the first n compoments of x
iparm(7) = 16 !default logical fortran unit number for output
iparm(8) = 9 !numbers of iterative refinement steps
iparm(9) = 0 !not in use
iparm(10) = 13 !perturbe the pivot elements with 1E-13
iparm(11) = 1 !use nonsymmetric permutation and scaling MPS
iparm(12) = 0 !not in use
iparm(13) = 0 !not in use
iparm(14) = 0 !Output: number of perturbed pivots
iparm(15) = 0 !not in use
iparm(16) = 0 !not in use
iparm(17) = 0 !not in use
iparm(18) = -1 !Output: number of nonzeros in the factor LU
iparm(19) = -1 !Output: Mflops for LU factorization
iparm(20) = 0 !Output: Numbers of CG Iterations
error = 0 !initialize error flag
msglvl = 0 !don't print statistical information
mtype = -2 !unsymmetric matrix symmetric, indefinite, no pivoting
C..Initiliaze the internal solver memory pointer.This is only
C necessary for the FIRST call of the PARDISO solver.
  do i = 1, 64
    pt(i) = 0
  end do
C..Reordering and Symbolic Factorization, This step also allocates
C all memory that is necessary for the factorization
  phase = 11 !only reordering and symbolic factorization
  CALL pardiso (pt, maxfct, mnum, mtype, phase, n, a, ia, ja,
1 idum, nrhs, iparm, msglvl, ddum, ddum, error)
  WRITE(*,*) 'Reordering completed ... '
  IF (error .NE.0) THEN
```

```
        WRITE(*,*) 'The following ERROR was detected: ', error
        STOP
    END IF
    WRITE(*,*) 'Number of nonzeros in factors = ',iparm(18)
    WRITE(*,*) 'Number of factorization MFLOPS = ',iparm(19)
C..Factorization.
    phase = 22 !only factorization
    CALL pardiso (pt, maxfct, mnum, mtype, phase, n, a, ia, ja,
1 idum, nrhs, iparm, msglvl, ddum, ddum, error)
    WRITE(*,*) 'Factorization completed ... '
    IF (error .NE.0) THEN
        WRITE(*,*) 'The following ERROR was detected: ', error
        STOP
    ENDIF
C..Back substitution and iterative refinement
    iparm(8) = 2 !max numbers of iterative refinement steps
    phase = 33 !only factorization
    do i = 1, n
        b(i) = 1.d0
    end do
    CALL pardiso (pt, maxfct, mnum, mtype, phase, n, a, ia, ja,
1 idum, nrhs, iparm, msglvl, b, x, error)
    WRITE(*,*) 'Solve completed ... '
    WRITE(*,*) 'The solution of the system is '
    DO i = 1, n
        WRITE(*,*) ' x(',i,') = ', x(i)
    END DO
C..Termination and release of memory
    phase = -1 !release internal memory
    CALL pardiso (pt, maxfct, mnum, mtype, phase, n, ddum, idum, idum,
1 idum, nrhs, iparm, msglvl, ddum, ddum, error)
    END
```

## 例 C-7 対称連立 1 次方程式の例 (pardiso\_sym.c)

---

```
/* ----- */
/* Example program to show the use of the "PARDISO" routine */
/* on symmetric linear systems */
/* ----- */
/* This program can be downloaded from the following site: */
/* http://www.computational.unibas.ch/cs/scicomp */
/* */
/* (C) Olaf Schenk, Department of Computer Science, */
/* University of Basel, Switzerland. */
/* Email: olaf.schenk@unibas.ch */
/* ----- */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
extern int omp_get_max_threads();
/* PARDISO prototype.*/
extern int PARDISO
    (void *, int *, int *, int *, int *, int *,
     double *, int *, int *, int *, int *, int *,
     int *, double *, double *, int *);

int main( void ) {
    /* Matrix data.*/
    int n = 8;
    int ia[ 9] = { 1, 5, 8, 10, 12, 15, 17, 18, 19 };
    int ja[18] = { 1, 3, 6, 7,
                  2, 3, 5,
                  3, 8,
                  4, 7,
                  5, 6, 7,
                  6, 8,
```

```
        7,
        8 };
double a[18] = { 7.0, 1.0, 2.0, 7.0,
                -4.0, 8.0, 2.0,
                1.0, 5.0,
                7.0, 9.0,
                5.0, 1.0, 5.0,
                -1.0, 5.0,
                11.0,
                5.0 };
int mtype = -2; /* Real symmetric matrix */
/* RHS and solution vectors.*/
double b[8], x[8];
int nrhs = 1; /* Number of right hand sides.*/
/* Internal solver memory pointer pt, */
/* 32-bit: int pt[64]; 64-bit: long int pt[64] */
/* or void *pt[64] should be OK on both architectures */
void *pt[64];
/* Pardiso control parameters.*/
int iparm[64];
int maxfct, mnum, phase, error, msglvl;
/* Auxiliary variables.*/
int i;
double ddum; /* Double dummy */
int idum; /* Integer dummy.*/
/* ----- */
/* ..Setup Pardiso control parameters. */
/* ----- */
    for (i = 0; i < 64; i++) {
        iparm[i] = 0;
    }
    iparm[0] = 1; /* No solver default */
    iparm[1] = 2; /* Fill-in reordering from METIS */

```

```

/* Numbers of processors, value of OMP_NUM_THREADS */
iparm[2] = omp_get_max_threads();
iparm[3] = 0; /* No iterative-direct algorithm */
iparm[4] = 0; /* No user fill-in reducing permutation */
iparm[5] = 0; /* Write solution into x */
iparm[6] = 16; /* Default logical fortran unit number for output */
iparm[7] = 2; /* Max numbers of iterative refinement steps */
iparm[8] = 0; /* Not in use */
iparm[9] = 13; /* Perturb the pivot elements with 1E-13 */
iparm[10] = 1; /* Use nonsymmetric permutation and scaling MPS */
iparm[11] = 0; /* Not in use */
iparm[12] = 0; /* Not in use */
iparm[13] = 0; /* Output: Number of perturbed pivots */
iparm[14] = 0; /* Not in use */
iparm[15] = 0; /* Not in use */
iparm[16] = 0; /* Not in use */
iparm[17] = -1; /* Output: Number of nonzeros in the factor LU */
iparm[18] = -1; /* Output: Mflops for LU factorization */
iparm[19] = 0; /* Output: Numbers of CG Iterations */
maxfct = 1; /* Maximum number of numerical factorizations.*/
mnum = 1; /* Which factorization to use.*/
msglvl = 0; /* Don't print statistical information in file */
error = 0; /* Initialize error flag */

/* ----- */
/* ..Initialize the internal solver memory pointer.This is only */
/* necessary for the FIRST call of the PARDISO solver. */
/* ----- */

    for (i = 0; i < 64; i++) {
        pt[i] = 0;
    }

/* ----- */
/* ..Reordering and Symbolic Factorization.This step also allocates */
/* all memory that is necessary for the factorization. */

```

```
/* ----- */
    phase = 11;
    PARDISO (pt, &maxfct, &mnum, &mtype, &phase,
             &n, a, ia, ja, &idum, &nrhs,
             iparm, &msglvl, &ddum, &ddum, &error);
    if (error != 0) {
        printf("\nERROR during symbolic factorization: %d", error);
        exit(1);
    }
    printf("\nReordering completed ... ");
    printf("\nNumber of nonzeros in factors = %d", iparm[17]);
    printf("\nNumber of factorization MFLOPS = %d", iparm[18]);
/* ----- */
/* ..Numerical factorization. */
/* ----- */

    phase = 22;
    PARDISO (pt, &maxfct, &mnum, &mtype, &phase,
             &n, a, ia, ja, &idum, &nrhs,
             iparm, &msglvl, &ddum, &ddum, &error);
    if (error != 0) {
        printf("\nERROR during numerical factorization: %d", error);
        exit(2);
    }
    printf("\nFactorization completed ... ");
/* ----- */
/* ..Back substitution and iterative refinement. */
/* ----- */

    phase = 33;
    iparm[7] = 2; /* Max numbers of iterative refinement steps.*/
    /* Set right hand side to one.*/
    for (i = 0; i < n; i++) {
        b[i] = 1;
    }
```

```

    PARDISO (pt, &maxfct, &mnum, &mtype, &phase,
              &n, a, ia, ja, &idum, &nrhs,
              iparm, &msglvl, b, x, &error);
    if (error != 0) {
        printf("\nERROR during solution: %d", error);
        exit(3);
    }
    printf("\nSolve completed ... ");
    printf("\nThe solution of the system is: ");
    for (i = 0; i < n; i++) {
        printf("\n x [%d] = % f", i, x[i] );
    }
    printf ("\n");

/* ----- */
/* ..Termination and release of memory.                */
/* ----- */

    phase = -1; /* Release internal memory.*/
    PARDISO (pt, &maxfct, &mnum, &mtype, &phase,
              &n, &ddum, ia, ja, &idum, &nrhs,
              iparm, &msglvl, &ddum, &ddum, &error);
    return 0;
}

```



**スパース非対称連立 1 次方程式の例**

このセクションでは、**PARDISO** を使用して非対称連立 1 次方程式の解を算出する 2 つの例 (Fortran および C) を示す。連立方程式  $Ax = b$  を解く。

$$A = \begin{bmatrix} 1.0 & -1.0 & 0.0 & -3.0 & 0.0 \\ -2.0 & 5.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 4.0 & 6.0 & 4.0 \\ -4.0 & 0.0 & 2.0 & 7.0 & 0.0 \\ 0.0 & 8.0 & 0.0 & 0.0 & -5.0 \end{bmatrix} \quad \text{および} \quad B = \begin{bmatrix} 1.0 \\ 1.0 \\ 1.0 \\ 1.0 \\ 1.0 \end{bmatrix}$$

**非対称連立 1 次方程式の計算結果の例**

ソルバが正常に実行されると、 $X$  の解は次のようになる。

```
Reordering completed ...
Number of nonzeros in factors = 21
Number of factorization MFLOPS = 0
Factorization completed ...
Solve completed ...
The solution of the system is
x( 1) = -0.522321429
x( 2) = -0.00892857143
x( 3) = 1.22098214
x( 4) = -0.504464286
x( 5) = -0.214285714
```

**例 C-8 非対称連立 1 次方程式の例 (pardiso\_unsym.f)**

```
*****
* Copyright(C) 2004 Intel Corporation.All Rights Reserved.
* The source code contained or described herein and all documents related to
* the source code ("Material") are owned by Intel Corporation or its suppliers
* or licensors.Title to the Material remains with Intel Corporation or its
* suppliers and licensors.The Material contains trade secrets and proprietary
* and confidential information of Intel or its suppliers and licensors.The
* Material is protected by worldwide copyright and trade secret laws and
* treaty provisions.No part of the Material may be used, copied, reproduced,
* modified, published, uploaded, posted, transmitted, distributed or disclosed
```

\* in any way without Intel's prior express written permission.  
\* No license under any patent, copyright, trade secret or other intellectual  
\* property right is granted to or conferred upon you by disclosure or delivery  
\* of the Materials, either expressly, by implication, inducement, estoppel or  
\* otherwise. Any license under such intellectual property rights must be  
\* express and approved by Intel in writing.

\*  
\*\*\*\*\*  
\*

\* Content : MKL DSS Fortran-77 example

\*  
\*\*\*\*\*  
\*

C-----

C Example program to show the use of the "PARDISO" routine  
C for symmetric linear systems

C-----

C This program can be downloaded from the following site:

C <http://www.computational.unibas.ch/cs/scicomp>

C

C (C) Olaf Schenk, Department of Computer Science,

C University of Basel, Switzerland.

C Email: [olaf.schenk@unibas.ch](mailto:olaf.schenk@unibas.ch)

C

C-----

PROGRAM pardiso\_unsym

IMPLICIT NONE

C..Internal solver memory pointer for 64-bit architectures

C..INTEGER\*8 pt(64)

C..Internal solver memory pointer for 32-bit architectures

C..INTEGER\*4 pt(64)

C..This is OK in both cases

INTEGER\*8 pt(64)

C..All other variables

INTEGER maxfct, mnum, mtype, phase, n, nrhs, error, msglvl

```
INTEGER iparm(64)
INTEGER ia(6)
INTEGER ja(13)
REAL*8 a(13)
REAL*8 b(5)
REAL*8 x(5)
INTEGER i, idum
REAL*8 waltimel, waltime2, ddum
C..Fill all arrays containing matrix data.
DATA n /5/, nrhs /1/, maxfct /1/, mnum /1/
DATA ia /1,4,6,9,12,14/
DATA ja
1 / 1, 2, 4,
2 1, 2,
3 3, 4, 5,
4 1, 3, 4,
5 2, 5/
DATA a
1 /1.d0,-1.d0, -3.d0,
2 -2.d0, 5.d0,
3 4.d0, 6.d0, 4.d0,
4 -4.d0, 2.d0, 7.d0,
5 8.d0, -5.d0/
integer omp_get_max_threads
external omp_get_max_threads
C..
C..Set up PARDISO control parameter
C..
do i = 1, 64
iparm(i) = 0
end do
iparm(1) = 1 !no solver default
iparm(2) = 2 !fill-in reordering from METIS
```

```
      iparm(3) = omp_get_max_threads() !numbers of processors, value of
OMP_NUM_THREADS
      iparm(4) = 0 !no iterative-direct algorithm
      iparm(5) = 0 !no user fill-in reducing permutation
      iparm(6) = 0 !=0 solution on the first n compoments of x
      iparm(7) = 0 !not in use
      iparm(8) = 9 !numbers of iterative refinement steps
      iparm(9) = 0 !not in use
      iparm(10) = 13 !perturbe the pivot elements with 1E-13
      iparm(11) = 1 !use nonsymmetric permutation and scaling MPS
      iparm(12) = 0 !not in use
      iparm(13) = 0 !not in use
      iparm(14) = 0 !Output: number of perturbed pivots
      iparm(15) = 0 !not in use
      iparm(16) = 0 !not in use
      iparm(17) = 0 !not in use
      iparm(18) = -1 !Output: number of nonzeros in the factor LU
      iparm(19) = -1 !Output: Mflops for LU factorization
      iparm(20) = 0 !Output: Numbers of CG Iterations
      error = 0 !initialize error flag
      msglvl = 1 !print statistical information
      mtype = 11 !real unsymmetric
C..Initiliaze the internal solver memory pointer.This is only
C necessary for the FIRST call of the PARDISO solver.
      do i = 1, 64
        pt(i) = 0
      end do
C..Reordering and Symbolic Factorization, This step also allocates
C all memory that is necessary for the factorization
      phase = 11 !only reordering and symbolic factorization
      CALL pardiso (pt, maxfct, mnum, mtype, phase, n, a, ia, ja,
1 idum, nrhs, iparm, msglvl, ddum, ddum, error)
      WRITE(*,*) 'Reordering completed ... '
```

```
IF (error .NE.0) THEN
  WRITE(*,*) 'The following ERROR was detected: ', error
  STOP
END IF
WRITE(*,*) 'Number of nonzeros in factors = ',iparm(18)
WRITE(*,*) 'Number of factorization MFLOPS = ',iparm(19)
C..Factorization.
  phase = 22 !only factorization
  CALL pardiso (pt, maxfct, mnum, mtype, phase, n, a, ia, ja,
1 idum, nrhs, iparm, msglvl, ddum, ddum, error)
  WRITE(*,*) 'Factorization completed ... '
  IF (error .NE.0) THEN
    WRITE(*,*) 'The following ERROR was detected: ', error
    STOP
  ENDIF
C..Back substitution and iterative refinement
  iparm(8) = 2 !max numbers of iterative refinement steps
  phase = 33 !only factorization
  do i = 1, n
    b(i) = 1.d0
  end do
  CALL pardiso (pt, maxfct, mnum, mtype, phase, n, a, ia, ja,
1 idum, nrhs, iparm, msglvl, b, x, error)
  WRITE(*,*) 'Solve completed ... '
  WRITE(*,*) 'The solution of the system is '
  DO i = 1, n
    WRITE(*,*) ' x('i,') = ', x(i)
  END DO
C..Termination and release of memory
  phase = -1 !release internal memory
  CALL pardiso (pt, maxfct, mnum, mtype, phase, n, ddum, idum, idum,
1 idum, nrhs, iparm, msglvl, ddum, ddum, error)
  END
```

## 例 C-9 非対称連立 1 次方程式の例 (pardiso\_unsym.c)

---

```
/*
*****
*
* Copyright(C) 2004 Intel Corporation.All Rights Reserved.
* The source code contained or described herein and all documents related to
* the source code ("Material") are owned by Intel Corporation or its suppliers
* or licensors.Title to the Material remains with Intel Corporation or its
* suppliers and licensors.The Material contains trade secrets and proprietary
* and confidential information of Intel or its suppliers and licensors.The
* Material is protected by worldwide copyright and trade secret laws and
* treaty provisions.No part of the Material may be used, copied, reproduced,
* modified, published, uploaded, posted, transmitted, distributed or disclosed
* in any way without Intel's prior express written permission.
* No license under any patent, copyright, trade secret or other intellectual
* property right is granted to or conferred upon you by disclosure or delivery
* of the Materials, either expressly, by implication, inducement, estoppel or
* otherwise.Any license under such intellectual property rights must be
* express and approved by Intel in writing.
*
*****
*
* Content : MKL DSS C example
*
*****
*/
/* ----- */
/* Example program to show the use of the "PARDISO" routine */
/* on symmetric linear systems */
/* ----- */
/* This program can be downloaded from the following site: */
/* http://www.computational.unibas.ch/cs/scicomp */
/* */
/* (C) Olaf Schenk, Department of Computer Science, */
/* University of Basel, Switzerland. */
/* Email: olaf.schenk@unibas.ch */
/* ----- */
#include <stdio.h>
```

```
#include <stdlib.h>
#include <math.h>
extern int omp_get_max_threads();
/* PARDISO prototype.*/
#if defined(_WIN32) || defined(_WIN64)
#define pardiso_ PARDISO
#else
#define PARDISO pardiso_
#endif
extern int PARDISO
    (void *, int *, int *, int *, int *, int *,
     double *, int *, int *, int *, int *, int *,
     int *, double *, double *, int *);

int main( void ) {
    /* Matrix data.*/
    int n = 5;
    int ia[ 6] = { 1, 4, 6, 9, 12, 14 };
    int ja[13] = { 1, 2, 4,
                  1, 2,
                  3, 4, 5,
                  1, 3, 4,
                  2, 5 };
    double a[18] = { 1.0, -1.0, -3.0,
                    -2.0, 5.0,
                    4.0, 6.0, 4.0,
                    -4.0, 2.0, 7.0,
                    8.0, -5.0 };
    int mtype = 11; /* Real unsymmetric matrix */
    /* RHS and solution vectors.*/
    double b[5], x[5];
    int nrhs = 1; /* Number of right hand sides.*/
    /* Internal solver memory pointer pt, */
```

```

/* 32-bit: int pt[64]; 64-bit: long int pt[64] */
/* or void *pt[64] should be OK on both architectures */
void *pt[64];
/* Pardiso control parameters.*/
int iparm[64];
int maxfct, mnum, phase, error, msglvl;
/* Auxiliary variables.*/
int i;
double ddum; /* Double dummy */
int idum; /* Integer dummy.*/

/* ----- */
/* ..Setup Pardiso control parameters. */
/* ----- */

    for (i = 0; i < 64; i++) {
        iparm[i] = 0;
    }
    iparm[0] = 1; /* No solver default */
    iparm[1] = 2; /* Fill-in reordering from METIS */
    /* Numbers of processors, value of OMP_NUM_THREADS */
    iparm[2] = omp_get_max_threads();
    iparm[3] = 0; /* No iterative-direct algorithm */
    iparm[4] = 0; /* No user fill-in reducing permutation */
    iparm[5] = 0; /* Write solution into x */
    iparm[6] = 0; /* Not in use */
    iparm[7] = 2; /* Max numbers of iterative refinement steps */
    iparm[8] = 0; /* Not in use */
    iparm[9] = 13; /* Perturb the pivot elements with 1E-13 */
    iparm[10] = 1; /* Use nonsymmetric permutation and scaling MPS */
    iparm[11] = 0; /* Not in use */
    iparm[12] = 0; /* Not in use */
    iparm[13] = 0; /* Output: Number of perturbed pivots */
    iparm[14] = 0; /* Not in use */
    iparm[15] = 0; /* Not in use */

```



```

    iparm[16] = 0; /* Not in use */
    iparm[17] = -1; /* Output: Number of nonzeros in the factor LU */
    iparm[18] = -1; /* Output: Mflops for LU factorization */
    iparm[19] = 0; /* Output: Numbers of CG Iterations */
    maxfct = 1; /* Maximum number of numerical factorizations.*/
    mnum = 1; /* Which factorization to use.*/
    msglvl = 1; /* Print statistical information in file */
    error = 0; /* Initialize error flag */

/* ----- */
/* ..Initialize the internal solver memory pointer.This is only */
/* necessary for the FIRST call of the PARDISO solver. */
/* ----- */
    for (i = 0; i < 64; i++) {
        pt[i] = 0;
    }

/* ----- */
/* ..Reordering and Symbolic Factorization.This step also allocates */
/* all memory that is necessary for the factorization. */
/* ----- */
    phase = 11;
    PARDISO (pt, &maxfct, &mnum, &mtype, &phase,
             &n, a, ia, ja, &idum, &nrhs,
             iparm, &msglvl, &ddum, &ddum, &error);
    if (error != 0) {
        printf("\nERROR during symbolic factorization: %d", error);
        exit(1);
    }
    printf("\nReordering completed ... ");
    printf("\nNumber of nonzeros in factors = %d", iparm[17]);
    printf("\nNumber of factorization MFLOPS = %d", iparm[18]);

/* ----- */
/* ..Numerical factorization. */
/* ----- */

```

```

    phase = 22;
    PARDISO (pt, &maxfct, &mnum, &mtype, &phase,
             &n, a, ia, ja, &idum, &nrhs,
             iparm, &msglvl, &ddum, &ddum, &error);
    if (error != 0) {
        printf("\nERROR during numerical factorization: %d", error);
        exit(2);
    }
    printf("\nFactorization completed ... ");

/* ----- */
/* ..Back substitution and iterative refinement. */
/* ----- */

    phase = 33;
    iparm[7] = 2; /* Max numbers of iterative refinement steps.*/
    /* Set right hand side to one.*/
    for (i = 0; i < n; i++) {
        b[i] = 1;
    }
    PARDISO (pt, &maxfct, &mnum, &mtype, &phase,
             &n, a, ia, ja, &idum, &nrhs,
             iparm, &msglvl, b, x, &error);
    if (error != 0) {
        printf("\nERROR during solution: %d", error);
        exit(3);
    }
    printf("\nSolve completed ... ");
    printf("\nThe solution of the system is: ");
    for (i = 0; i < n; i++) {
        printf("\n x [%d] = % f", i, x[i] );
    }
    printf ("\n");

/* ----- */
/* ..Termination and release of memory. */

```

```

/* ----- */
    phase = -1; /* Release internal memory.*/
    PARDISO (pt, &maxfct, &mnum, &mtype, &phase,
             &n, &ddum, ia, ja, &idum, &nrhs,
             iparm, &msglvl, &ddum, &ddum, &error);
    return 0;
}

```

## 直接法スパース・ソルバのコード例

このセクションでは、Fortran 77、Fortran 90、および C のコード例を示す。コードで使われるスパース・ソルバ・ルーチンの詳細は、本マニュアル [第 8 章](#) の「[直接法スパースソルバ \(DSS\) インターフェイス・ルーチン](#)」を参照のこと。

このコード例は、付録 A の [直接法](#) のセクションにある方程式 (スパース行列による対称正定値連立 1 次方程式  $Ax = b$ ) の解を算出する。

$$A = \begin{bmatrix} 9 & 1.5 & 6 & 0.75 & 3 \\ 1.5 & 0.5 & 0 & 0 & 0 \\ 6 & 0 & 12 & 0 & 0 \\ 0.75 & 0 & 0 & 0.625 & 0 \\ 3 & 0 & 0 & 0 & 16 \end{bmatrix} \quad \text{および} \quad B = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{bmatrix}$$

### 対称連立 1 次方程式の計算結果の例

ソルバが正常に実行されると、行列式および解の配列は次のようになる。

```

pow of determinant is      0.000
base of determinant is      2.250
Determinant is             2.250
Solution Array:   -326.333   983.000   163.417   398.000   61.500

```

### 例 C-10 対称正定値連立方程式の例 (Fortran 77)

```

*****
*
*   Copyright(C) 2001-2004 Intel Corporation.All Rights Reserved.
*   The source code contained or described herein and all documents related to
*   the source code ("Material") are owned by Intel Corporation or its suppliers
*   or licensors.Title to the Material remains with Intel Corporation or its
*   suppliers and licensors.The Material contains trade secrets and proprietary
*   and confidential information of Intel or its suppliers and licensors.The

```

```
* Material is protected by worldwide copyright and trade secret laws and
* treaty provisions.No part of the Material may be used, copied, reproduced,
* modified, published, uploaded, posted, transmitted, distributed or disclosed
* in any way without Intel's prior express written permission.
* No license under any patent, copyright, trade secret or other intellectual
* property right is granted to or conferred upon you by disclosure or delivery
* of the Materials, either expressly, by implication, inducement, estoppel or
* otherwise.Any license under such intellectual property rights must be
* express and approved by Intel in writing.
```

```
*
*****
*
```

```
* Content : Intel MKL DSS Fortran-77 example
```

```
*
*****
*
```

```
C-----
C Example program for solving symmetric positive definite system of
C equations.
```

```
C-----
PROGRAM solver_f77_test
IMPLICIT NONE
INCLUDE 'mkl_dss.f77'
```

```
C-----
C Define the array and rhs vectors
```

```
C-----
INTEGER nRows, nCols, nNonZeros, i, nRhs
PARAMETER (nRows = 5,
1 nCols = 5,
2 nNonZeros = 9,
3 nRhs = 1)
INTEGER rowIndex(nRows + 1), columns(nNonZeros)
DOUBLE PRECISION values(nNonZeros), rhs(nRows)
DATA rowIndex / 1, 6, 7, 8, 9, 10 /
DATA columns / 1, 2, 3, 4, 5, 2, 3, 4, 5 /
DATA values / 9, 1.5, 6, .75, 3, 0.5, 12, .625, 16 /
DATA rhs / 1, 2, 3, 4, 5 /
```

```
C-----
C Allocate storage for the solver handle and the solution vector
```

```
C-----
DOUBLE PRECISION solution(nRows)
INTEGER*8 handle
INTEGER error
```

```

    CHARACTER*15 statIn
    DOUBLE PRECISION statOut(5)
    INTEGER bufLen
    PARAMETER(bufLen = 20)
    INTEGER buff(bufLen)
C-----
C Initialize the solver
C-----
    error = dss_create(handle, MKL_DSS_DEFAULTS)
    IF (error .NE.MKL_DSS_SUCCESS ) GOTO 999
C-----
C Define the non-zero structure of the matrix
C-----
    error = dss_define_structure( handle, MKL_DSS_SYMMETRIC,
    & rowIndex, nRows, nCols, columns, nNonZeros )
    IF (error .NE.MKL_DSS_SUCCESS ) GOTO 999
C-----
C Reorder the matrix
C-----
    error = dss_reorder( handle, MKL_DSS_DEFAULTS, 0)
    IF (error .NE.MKL_DSS_SUCCESS ) GOTO 999
C-----
C Factor the matrix
C-----
    error = dss_factor_real( handle,
    & MKL_DSS_DEFAULTS, VALUES)
    IF (error .NE.MKL_DSS_SUCCESS ) GOTO 999
C-----
C Get the solution vector
C-----
    error = dss_solve_real( handle, MKL_DSS_DEFAULTS,
    & rhs, nRhs, solution)
    IF (error .NE.MKL_DSS_SUCCESS ) GOTO 999
C-----
C Print Determinant of the matrix
C-----
    statIn = 'determinant'
    call mkl_cvt_to_null_terminated_str(buff,bufLen,statIn)
    error = dss_statistics(handle, MKL_DSS_DEFAULTS,
    & buff,statOut)
    WRITE(*, "(' pow of determinant is ', 5(F10.3))") statOut(1)
    WRITE(*, "(' base of determinant is ', 5(F10.3))") statOut(2)
    WRITE(*, "(' Determinant is ', 5(F10.3))(10**statOut(1))*
    & statOut(2)

```

```

C-----
C Deallocate solver storage
C-----
      error = dss_delete( handle, MKL_DSS_DEFAULTS )
      IF (error .NE.MKL_DSS_SUCCESS ) GOTO 999
C-----
C Print solution vector
C-----
      WRITE(*,900) (solution(i), i = 1, nCols)
900 FORMAT(' Solution Array: ',5(F10.3))
      GOTO 1000
999 WRITE(*,*) "Solver returned error code ", error
1000 END

```

## 例 C-11 対称正定値連立方程式の例 (C)

---

```

/*
*****
* Copyright(C) 2001-2004 Intel Corporation.All Rights Reserved.
* The source code contained or described herein and all documents related to
* the source code ("Material") are owned by Intel Corporation or its suppliers
* or licensors.Title to the Material remains with Intel Corporation or its
* suppliers and licensors.The Material contains trade secrets and proprietary
* and confidential information of Intel or its suppliers and licensors.The
* Material is protected by worldwide copyright and trade secret laws and
* treaty provisions.No part of the Material may be used, copied, reproduced,
* modified, published, uploaded, posted, transmitted, distributed or disclosed
* in any way without Intel's prior express written permission.
* No license under any patent, copyright, trade secret or other intellectual
* property right is granted to or conferred upon you by disclosure or delivery
* of the Materials, either expressly, by implication, inducement, estoppel or
* otherwise.Any license under such intellectual property rights must be
* express and approved by Intel in writing.
*
*****
*
* Content : Intel MKL DSS C example
*
*****
*/
/*

** Example program to solve symmetric positive definite system of equations.

```

```
*/
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include "mkl_dss.h"
/*
** Define the array and rhs vectors
*/
#define NROWS 5
#define NCOLS 5
#define NNONZEROS 9
#define NRHS 1
static const int nRows =      NROWS ;
static const int nCols =      NCOLS ;
static const int nNonZeros = NNONZEROS ;
static const int nRhs =       NRHS ;
static _INTEGER_t rowIndex[NROWS+1] = { 1, 6, 7, 8, 9, 10 };
static _INTEGER_t columns[NNONZEROS] = { 1, 2, 3, 4, 5, 2, 3, 4, 5 };
static _DOUBLE_PRECISION_t values[NNONZEROS] = { 9, 1.5, 6, .75, 3, 0.5, 12, .625, 16 };
static _DOUBLE_PRECISION_t rhs[NCOLS] = { 1, 2, 3, 4, 5 };

void main() {
    int i;
    /* Allocate storage for the solver handle and the right-hand side.*/
    _DOUBLE_PRECISION_t solValues[NROWS];
    _MKL_DSS_HANDLE_t handle;
    _INTEGER_t error;
    _CHARACTER_STR_t statIn[] = "determinant";
    _DOUBLE_PRECISION_t statOut[5];
    int opt = MKL_DSS_DEFAULTS;
    int sym = MKL_DSS_SYMMETRIC;
    int type = MKL_DSS_POSITIVE_DEFINITE;
/* ----- */
/* Initialize the solver */
/* ----- */
    error = dss_create(handle, opt );
    if ( error != MKL_DSS_SUCCESS ) goto printError;
/* ----- */
/* Define the non-zero structure of the matrix */
/* ----- */
    error = dss_define_structure(
        handle, sym, rowIndex, nRows, nCols,
        columns, nNonZeros );
    if ( error != MKL_DSS_SUCCESS ) goto printError;
```

```

/* ----- */
/* Reorder the matrix */
/* ----- */
    error = dss_reorder( handle, opt, 0);
    if ( error != MKL_DSS_SUCCESS ) goto printError;
/* ----- */
/* Factor the matrix */
/* ----- */
    error = dss_factor_real( handle, type, values );
    if ( error != MKL_DSS_SUCCESS ) goto printError;
/* ----- */
/* Get the solution vector */
/* ----- */
    error = dss_solve_real( handle, opt, rhs, nRhs, solValues );
    if ( error != MKL_DSS_SUCCESS ) goto printError;
/* ----- */
/* Get the determinant */
/*-----*/
    error = dss_statistics(handle, opt, statIn, statOut);
    if ( error != MKL_DSS_SUCCESS ) goto printError;
/*-----*/
/* print determinant */
/*-----*/
    printf(" determinant power is %g \n", statOut[0]);
    printf(" determinant base is %g \n", statOut[1]);
    printf(" Determinant is %g \n", (pow(10.0,statOut[0]))*statOut[1]);
    free((void *) statIn);
/* ----- */
/* Deallocate solver storage */
/* ----- */
    error = dss_delete( handle, opt );
    if ( error != MKL_DSS_SUCCESS ) goto printError;
/* ----- */
/* Print solution vector */
/* ----- */
    printf(" Solution array: ");
    for(i = 0; i< nCols; i++)
        printf(" %g", solValues[i] );
    printf("\n");
    exit(0);
printError:
    printf("Solver returned error code %d\n", error);
    exit(1);
}

```



## 例 C-12 対称正定値連立方程式の例 (Fortran 90)

```

!*****
*
!   Copyright(C) 2001-2004 Intel Corporation.All Rights Reserved.
!   The source code contained or described herein and all documents related to
!   the source code ("Material") are owned by Intel Corporation or its suppliers
!   or licensors.Title to the Material remains with Intel Corporation or its
!   suppliers and licensors.The Material contains trade secrets and proprietary
!   and confidential information of Intel or its suppliers and licensors.The
!   Material is protected by worldwide copyright and trade secret laws and
!   treaty provisions.No part of the Material may be used, copied, reproduced,
!   modified, published, uploaded, posted, transmitted, distributed or disclosed
!   in any way without Intel's prior express written permission.
!   No license under any patent, copyright, trade secret or other intellectual
!   property right is granted to or conferred upon you by disclosure or delivery
!   of the Materials, either expressly, by implication, inducement, estoppel or
!   otherwise.Any license under such intellectual property rights must be
!   express and approved by Intel in writing.
!
!*****
*
!   Content : Intel MKL DSS Fortran-90 example
!
!*****
*
!-----
!
!Example program for solving a symmetric positive definite system of
!equations.
!
!-----
INCLUDE 'mkl_dss.f90' !Include the standard DSS "header file."
PROGRAM solver_f90_test
use mkl_dss
IMPLICIT NONE
INTEGER, PARAMETER :: dp = KIND(1.0D0)
INTEGER :: error
INTEGER :: i
INTEGER, PARAMETER :: bufLen = 20
!Define the data arrays and the solution and rhs vectors.
INTEGER, ALLOCATABLE :: columns( : )
INTEGER :: nCols
INTEGER :: nNonZeros

```

```

INTEGER :: nRhs
INTEGER :: nRows
REAL(KIND=DP), ALLOCATABLE :: rhs( : )
INTEGER, ALLOCATABLE :: rowIndex( : )
REAL(KIND=DP), ALLOCATABLE :: solution( : )
REAL(KIND=DP), ALLOCATABLE :: values( : )
TYPE(MKL_DSS_HANDLE) :: handle !Allocate storage for the solver handle.
REAL(KIND=DP),ALLOCATABLE::statOut( : )
CHARACTER*15 statIn
INTEGER perm(1)
INTEGER buff(bufLen)
EXTERNAL MKL_CVT_TO_NULL_TERMINATED_STR
!Set the problem to be solved.
nRows = 5
nCols = 5
nNonZeros = 9
nRhs = 1
perm(1) = 0
ALLOCATE( rowIndex( nRows + 1 ) )
rowIndex = (/ 1, 6, 7, 8, 9, 10 /)
ALLOCATE( columns( nNonZeros ) )
columns = (/ 1, 2, 3, 4, 5, 2, 3, 4, 5 /)
ALLOCATE( values( nNonZeros ) )
values = (/ 9.0_DP, 1.5_DP, 6.0_DP, 0.75_DP, 3.0_DP, 0.5_DP, 12.0_DP, &
& 0.625_DP, 16.0_DP /)
ALLOCATE( rhs( nRows ) )
rhs = (/ 1.0_DP, 2.0_DP, 3.0_DP, 4.0_DP, 5.0_DP /)
!Initialize the solver.
error = dss_create( handle, MKL_DSS_DEFAULTS )
IF (error /= MKL_DSS_SUCCESS) GOTO 999
!Define the non-zero structure of the matrix.
error = dss_define_structure( handle, MKL_DSS_SYMMETRIC, rowIndex, nRows, &
& nCols, columns, nNonZeros )
IF (error /= MKL_DSS_SUCCESS) GOTO 999
!Reorder the matrix.
error = dss_reorder( handle, MKL_DSS_DEFAULTS, perm )
IF (error /= MKL_DSS_SUCCESS) GOTO 999
!Factor the matrix.
error = dss_factor_real( handle, MKL_DSS_DEFAULTS, values )
IF (error /= MKL_DSS_SUCCESS) GOTO 999
!Allocate the solution vector and solve the problem.
ALLOCATE( solution( nRows ) )
error = dss_solve_real(handle, MKL_DSS_DEFAULTS, rhs, nRhs, solution )
IF (error /= MKL_DSS_SUCCESS) GOTO 999

```

```

!Print Out the determinant of the matrix
ALLOCATE(statOut( 5 ) )
statIn = 'determinant'
call mkl_cvt_to_null_terminated_str(buff,bufLen,statIn);
error = dss_statistics(handle, MKL_DSS_DEFAULTS, buff, statOut )
IF (error /= MKL_DSS_SUCCESS) GOTO 999
WRITE(*, "('pow of determinant is '(5F10.3))") ( statOut(1) )
WRITE(*, "('base of determinant is '(5F10.3))") ( statOut(2) )
WRITE(*, "('Determinant is '(5F10.3))") ( (10**statOut(1))*statOut(2) )
!Deallocate solver storage and various local arrays.
error = dss_delete( handle, MKL_DSS_DEFAULTS )
IF (error /= MKL_DSS_SUCCESS ) GOTO 999
IF ( ALLOCATED( rowIndex ) ) DEALLOCATE( rowIndex )
IF ( ALLOCATED( columns ) ) DEALLOCATE( columns )
IF ( ALLOCATED( values ) ) DEALLOCATE( values )
IF ( ALLOCATED( rhs ) ) DEALLOCATE( rhs )
IF ( ALLOCATED( statOut ) ) DEALLOCATE( statOut )
!Print the solution vector, deallocate it and exit
WRITE(*, "('Solution Array: '(5F10.3))") ( solution(i), i = 1, nCols )
IF ( ALLOCATED( solution ) ) DEALLOCATE( solution )
GOTO 1000
!Print an error message and exit
999 WRITE(*,*) "Solver returned error code ", error
1000 CONTINUE
END PROGRAM solver_f90_test

```

## 反復法スパース・ソルバのコード例

このセクションでは Fortran 77 のコード例を示す。コードで使用されるリバース・コミュニケーション・インターフェイス (RCI ISS) に基づく反復法スパース・ソルバ・ルーチンの詳細は、本マニュアル第 8 章の「[リバース・コミュニケーション・インターフェイス \(RCI ISS\) に基づく反復法スパースソルバ](#)」を参照のこと。

### RCI (プリコンディショニング) 共役勾配ソルバの使用例

対称正定値連立 1 次代数方程式の計算結果の例を示す。ソルバが正常に実行されると、解の配列は次のようになる。

```

The system is successfully solved
  The following solution obtained
    1.000    0.000    1.000    0.000
    1.000    0.000    1.000    0.000
  Expected solution

```

```

1.000      0.000      1.000      0.000
1.000      0.000      1.000      0.000
Number of iterations:  8

```

## 例 C-13 対称正定値連立方程式の例 (Fortran 77)

---

```

*****
*
* Copyright(C) 2001-2005 Intel Corporation.All Rights Reserved.
* The source code contained or described herein and all documents related to
* the source code ("Material") are owned by Intel Corporation or its suppliers
* or licensors.Title to the Material remains with Intel Corporation or its
* suppliers and licensors.The Material contains trade secrets and proprietary
* and confidential information of Intel or its suppliers and licensors.The
* Material is protected by worldwide copyright and trade secret laws and
* treaty provisions.No part of the Material may be used, copied, reproduced,
* modified, published, uploaded, posted, transmitted, distributed or disclosed
* in any way without Intel's prior express written permission.
* No license under any patent, copyright, trade secret or other intellectual
* property right is granted to or conferred upon you by disclosure or delivery
* of the Materials, either expressly, by implication, inducement, estoppel or
* otherwise.Any license under such intellectual property rights must be
* express and approved by Intel in writing.
*
*****
*
* Content : Intel MKL RCI (P)CG Fortran-77 example
*
*****
*
C-----
C Example program for solving symmetric positive definite system of
C equations.
C-----
      PROGRAM rci_pcg_f77_test
      IMPLICIT NONE

C-----
C Define arrays for the upper triangle of the coefficient matrix and rhs vector
C Compressed sparse row storage is used for sparse representation
C-----
      INTEGER N, RCI_request, itercount, i
      PARAMETER (N=8)
      DOUBLE PRECISION  rhs(N), solution(N)

```

**例 C-13** 対称正定値連立方程式の例 (Fortran 77)

```

      INTEGER ia(9)
      INTEGER ja(18)
      DOUBLE PRECISION a(18)
C..Fill all arrays containing matrix data.
      DATA ia /1,5,8,10,12,15,17,18,19/
      DATA ja
1 /1,  3,  6,7,
2      2,  3,  5,
3      3,      8,
4      4,      7,
5      5,6,7,
6      6,  8,
7      7,
8      8/
      DATA a
1 /7.D0,      1.D0,      2.D0, 7.D0,
2      -4.D0,8.D0,      2.D0,
3      1.D0,      5.D0,
4      7.D0,      9.D0,
5      5.D0, 1.D0, 5.D0,
6      -1.D0,      5.D0,
7      11.D0,
8      5.D0/
C-----
C Allocate storage for the solver ?par and the initial solution vector
C-----
      INTEGER length
      PARAMETER (length=128)
      DOUBLE PRECISION expected(N)
      DATA expected/1.D0, 0.D0, 1.D0, 0.D0, 1.D0, 0.D0, 1.D0, 0.D0/
      INTEGER ipar(length)
      DOUBLE PRECISION dpar(length),tmp(N,4)
C-----
C Initialize the right hand side through matrix-vector product
C-----
      CALL DCSRMV_SY('U', N, A, IA, JA, expected, rhs)

```

## 例 C-13 対称正定値連立方程式の例 (Fortran 77)

---

```

C-----
C Initialize the initial guess
C-----
      DO I=1, N
         solution(I)=1.D0
      ENDDO

C-----
C Initialize the solver
C-----
      CALL dcg_init(N,solution,rhs,RCI_request,ipar,dpar,tmp)
      IF (RCI_request .NE.0 ) GOTO 999

C-----
C Set the desired parameters:
C LOGICAL parameters:
C do residual stopping test
C do not request for the user defined stopping test
C do Preconditioned Conjugate Gradient iterations
C DOUBLE PRECISION parameters
C set the relative tolerance to 1.0D-5 instead of default value 1.0D-6
C-----
      ipar(9)=1
      ipar(10)=0
      ipar(11)=1
      dpar(1)=1.D-5

C-----
C Check the correctness and consistency of the newly set parameters
C-----
      CALL dcg_check(N,solution,rhs,RCI_request,ipar,dpar,tmp)
      IF (RCI_request .NE.0 ) GOTO 999

C-----
C Compute the solution by RCI PCG solver
C Reverse Communications starts here
C-----
1      CALL dcg(N,solution,rhs,RCI_request,ipar,dpar,tmp)
C-----
C If RCI_request=0, then the solution was found with the required precision
C-----
      IF (RCI_request .EQ.0) THEN
         GOTO 700

C-----
C If RCI_request=1, then compute the vector A*tmp(:,1)
C and put the result in vector tmp(:,2)
C-----

```

**例 C-13 対称正定値連立方程式の例 (Fortran 77)**

```

      ELSE IF (RCI_request .EQ.1) THEN
        CALL DCSRMV_SY('U', N, A, IA, JA, TMP, TMP(1, 2))
        GOTO 1
C-----
C If RCI_request=3, then compute vector preconditioner matrix on tmp(:,3)
C and put the result in vector tmp(:,4)
C-----
      ELSE IF (RCI_request .EQ.3) THEN
        CALL DCOPY(N, TMP(1,3),1, TMP(1, 4), 1)
        GOTO 1
      ELSE
C-----
C If RCI_request=anything else, then dcg subroutine failed
C to compute the solution vector: solution(N)
C-----
        GOTO 999
      ENDIF
C-----
C Reverse Communication ends here
C Get the current iteration number
C-----
700  CALL dcg_get(N,solution,rhs,RCI_request,ipar,dpar,tmp,
      &            itercount)
C-----
C Print solution vector: solution(N) and number of iterations: itercount
C-----
      WRITE(*, *) ' The system is successfully solved '
      WRITE(*, *) ' The following solution obtained '
      WRITE(*,800)(solution(i),i =1,N)
      WRITE(*, *) ' Expected solution '
      WRITE(*,800)(expected(i),i =1,N)
800  FORMAT(4(F10.3))
      WRITE(*,900)(itercount)
900  FORMAT(' Number of iterations: ',1(I2))
      GOTO 1000
999  WRITE(*,*) 'Solver returned error code ', RCI_request
      STOP
1000 CONTINUE
      read *
      END

```

## DFT コードの例

このセクションでは、「[フーリエ変換関数](#)」の章で説明された DFT インターフェイス関数を使用するコード例を示す。

この例では 1 次元の計算を 2 回行っている。これらの例では、「[構成設定](#)」で指定されるすべての構成パラメータに対してのデフォルト設定を使用する。

### 例 C-14 1 次元 DFT (Fortran インターフェイス)

---

```
!Fortran example.
!1D complex to complex, and real to conjugate even
Use MKL_DFTI
Complex :: X(32)
Real :: Y(34)
type(DFTI_DESCRIPTOR), POINTER :: My_Desc1_Handle, My_Desc2_Handle
Integer :: Status
...put input data into X(1),...,X(32); Y(1),...,Y(32)

!Perform a complex to complex transform
Status = DftiCreateDescriptor( My_Desc1_Handle, DFTI_SINGLE,
                               DFTI_COMPLEX, 1, 32 )
Status = DftiCommitDescriptor( My_Desc1_Handle )
Status = DftiComputeForward( My_Desc1_Handle, X )
Status = DftiFreeDescriptor(My_Desc1_Handle)
!result is given by {X(1),X(2),...,X(32)}

!Perform a real to complex conjugate even transform
Status = DftiCreateDescriptor(My_Desc2_Handle, DFTI_SINGLE,
                               DFTI_REAL, 1, 32)
Status = DftiCommitDescriptor(My_Desc2_Handle)
Status = DftiComputeForward(My_Desc2_Handle, Y)
Status = DftiFreeDescriptor(My_Desc2_Handle)
!result is given in CCS format.
```

---



**例 C-15      1 次元 DFT (C インターフェイス)**

```
/* C example, float _Complex is defined in C9X */
#include "mkl_dfti.h"
float _Complex x[32];
float y[34];
DFTI_DESCRIPTOR *my_desc1_handle, *my_desc2_handle;
/* ....or alternatively
DFTI_DESCRIPTOR_HANDLE my_desc1_handle, my_desc2_handle; */

long status;
...put input data into x[0],...,x[31]; y[0],...,y[31]
status = DftiCreateDescriptor( &my_desc1_handle, DFTI_SINGLE,
                              DFTI_COMPLEX, 1, 32);
status = DftiCommitDescriptor( my_desc1_handle );
status = DftiComputeForward( my_desc1_handle, x);
status = DftiFreeDescriptor(&my_desc1_handle);
/* result is x[0], ..., x[31] */
status = DftiCreateDescriptor( &my_desc2_handle, DFTI_SINGLE,
                              DFTI_REAL, 1, 32);
status = DftiCommitDescriptor( my_desc2_handle);
status = DftiComputeForward( my_desc2_handle, y);
status = DftiFreeDescriptor(&my_desc2_handle);
/* result is given in CCS format */
```

次に示すのは単純な 2 次元変換を 2 回行った例である。計算関数のデータと結果パラメータは、すべて擬寸法階数 1 の配列 DIMENSION(0:\*) として宣言されている点に注意する。そのため、2 次元配列は EQUIVALENCE 文か Fortran のその他の機能を使用して 1 次元配列に変換しなければならない。

## 例 C-16      2 次元 DFT (Fortran インターフェイス)

---

```
!Fortran example.
!2D complex to complex, and real to conjugate even
Use MKL_DFTI
Complex :: X_2D(32,100)
Real :: Y_2D(34, 102)
Complex :: X(3200)
Real :: Y(3468)
Equivalence (X_2D, X)
Equivalence (Y_2D, Y)
type(DFTI_DESCRIPTOR), POINTER :: My_Desc1_Handle, My_Desc2_Handle
Integer :: Status, L(2)
...put input data into X_2D(j,k), Y_2D(j,k), 1<=j<=32,1<=k<=100
...set L(1) = 32, L(2) = 100
...the transform is a 32-by-100

!Perform a complex to complex transform
Status = DftiCreateDescriptor( My_Desc1_Handle, DFTI_SINGLE,
                               DFTI_COMPLEX, 2, L)
Status = DftiCommitDescriptor( My_Desc1_Handle)
Status = DftiComputeForward( My_Desc1_Handle, X)
Status = DftiFreeDescriptor(My_Desc1_Handle)
!result is given by X_2D(j,k), 1<=j<=32, 1<=k<=100

!Perform a real to complex conjugate even transform
Status = DftiCreateDescriptor( My_Desc2_Handle, DFTI_SINGLE,
                               DFTI_REAL, 2, L)
Status = DftiCommitDescriptor( My_Desc2_Handle)
Status = DftiComputeForward( My_Desc2_Handle, Y)
Status = DftiFreeDescriptor(My_Desc2_Handle)
!result is given by the complex value z(j,k) 1<=j<=32; 1<=k<=100
!and is stored in CCS format
```

---

**例 C-17      2次元 DFT (C インターフェイス)**

---

```
/* C example */
#include "mkl_dfti.h"
float _Complex x[32][100];
float y[34][102];
DFTI_DESCRIPTOR_HANDLE my_desc1_handle, my_desc2_handle;
/* or alternatively
DFTI_DESCRIPTOR *my_desc1_handle, *my_desc2_handle; */
long status, l[2];
...put input data into x[j][k] 0<=j<=31, 0<=k<=99
...put input data into y[j][k] 0<=j<=31, 0<=k<=99
l[0] = 32; l[1] = 100;
status = DftiCreateDescriptor( &my_desc1_handle, DFTI_SINGLE,
                             DFTI_COMPLEX, 2, l);
status = DftiCommitDescriptor( my_desc1_handle);
status = DftiComputeForward( my_desc1_handle, x);
status = DftiFreeDescriptor(&my_desc1_handle);
/* result is the complex value x[j][k], 0<=j<=31, 0<=k<=99 */
status = DftiCreateDescriptor( &my_desc2_handle, DFTI_SINGLE,
                             DFTI_REAL, 2, l);
status = DftiCommitDescriptor( my_desc2_handle);
status = DftiComputeForward( my_desc2_handle, y);
status = DftiFreeDescriptor(&my_desc2_handle);
/* result is the complex value z(j,k) 0<=j<=31; 0<=k<=99
/* and is stored in CCS format */
```

---

次の例は、DftiSetValue 関数を使用してデフォルト構成設定を変更する方法を示す。

例えば、入力データを DFT 計算後にも保全しておくには、DFTI\_PLACEMENT の設定をデフォルトの「インプレース」から「ノット・イン・プレース」に変更する必要がある。

以下にコードの例を示す。

### 例 C-18 デフォルト設定の変更 (Fortran)

---

```
!Fortran example
!1D complex to complex, not in place
Use MKL_DFTI
Complex :: X_in(32), X_out(32)
type(DFTI_DESCRIPTOR), POINTER :: My_Desc_Handle
Integer :: Status
...put input data into X_in(j), 1<=j<=32
Status = DftiCreateDescriptor( My_Desc_Handle, DFTI_SINGLE,
                               DFTI_COMPLEX, 1, 32)
Status = DftiSetValue( My_Desc_Handle, DFTI_PLACEMENT, DFTI_NOT_INPLACE)
Status = DftiCommitDescriptor( My_Desc_Handle)
Status = DftiComputeForward( My_Desc_Handle, X_in, X_out)
Status = DftiFreeDescriptor (My_Desc_Handle)
!result is X_out(1),X_out(2),...,X_out(32)
```

---

**例 C-19      デフォルト設定の変更 (C)**

---

```
/* C example */
#include "mkl_dfti.h"
float _Complex x_in[32], x_out[32];
DFTI_DESCRIPTOR_HANDLE my_desc_handle;
/* or alternatively
DFTI_DESCRIPTOR *my_desc_handle; */
long status;
...put input data into x_in[j], 0 <= j < 32
status = DftiCreateDescriptor( &my_desc_handle, DFTI_SINGLE,
DFTI_COMPLEX, 1, 32);
status = DftiSetValue( my_desc_handle, DFTI_PLACEMENT,
DFTI_NOT_INPLACE);
status = DftiCommitDescriptor( my_desc_handle);
status = DftiComputeForward( my_desc_handle, x_in, x_out);
status = DftiFreeDescriptor(&my_desc_handle);
/* result is x_out[0], x_out[1], ..., x_out[31] */
```

---

次の例 C-20 は、第 11 章のステータス確認関数の使用方法を説明する。

## 例 C-20      ステータス確認関数の使用

---

```
from C language:

DFTI_DESCRIPTOR_HANDLE desc;
long status, class_error, value;
char* error_message;
...descriptor creation and other code
status = DftiGetValue( desc, DFTI_PRECISION, &value); //
//or any DFTI function

class_error = DftiErrorClass(status, DFTI_NO_ERROR);
if (!class_error) {
    printf ("DftiGetValue() fixes the wrong situation and
            returns the corresponding value n");
    error_message = DftiErrorMessage(status);
    printf("error_message = %s \n", error_message);
}
...
from Fortran:

type(DFTI_DESCRIPTOR), POINTER :: desc
integer value, status
character(DFTI_MAX_MESSAGE_LENGTH) error_message
logical class_error
...descriptor creation and other code
status = DftiGetValue( desc, DFTI_PRECISION, value)
class_error = DftiErrorClass(status, DFTI_NO_ERROR)
if (.not.class_error) then
    print *, ' DftiGetValue() fixes the wrong situation and
            returns the corresponding value '
    error_message = DftiErrorMessage(status)
    print *, 'error_message = ', error_message
endif
```

---

以下は、 $20 \times 40$  の 2 次元 DFT を 1 次元変換を明示的に用いて計算した例である。計算関数のデータと結果パラメータは、すべて擬寸法階数 1 の配列 `DIMENSION(0:*)` として宣言されている点に注意する。そのため、2 次元配列は `EQUIVALENCE` 文か Fortran のその他の機能を使用して 1 次元配列に変換しなければならない。

**例 C-21      1 次元変換による 2 次元 DFT の計算**

```
!Fortran
Complex :: X_2D(20,40),
Complex :: X(800)
Equivalence (X_2D, X)
INTEGER :: STRIDE(2)
type(DFTI_DESCRIPTOR), POINTER :: Desc_Handle_Dim1
type(DFTI_DESCRIPTOR), POINTER :: Desc_Handle_Dim2
...
Status = DftiCreateDescriptor( Desc_Handle_Dim1, DFTI_SINGLE,
                               DFTI_COMPLEX, 1, 20 )
Status = DftiCreateDescriptor( Desc_Handle_Dim2, DFTI_SINGLE,
                               DFTI_COMPLEX, 1, 40 )

!perform 40 one-dimensional transforms along 1st dimension
Status = DftiSetValue( Desc_Handle_Dim1, DFTI_NUMBER_OF_TRANSFORMS, 40 )
Status = DftiSetValue( Desc_Handle_Dim1, DFTI_INPUT_DISTANCE, 20 )
Status = DftiSetValue( Desc_Handle_Dim1, DFTI_OUTPUT_DISTANCE, 20 )
Status = DftiCommitDescriptor( Desc_Handle_Dim1 )
Status = DftiComputeForward( Desc_Handle_Dim1, X )

!perform 20 one-dimensional transforms along 2nd dimension
Stride(1) = 0; Stride(2) = 20
Status = DftiSetValue( Desc_Handle_Dim2, DFTI_NUMBER_OF_TRANSFORMS, 20 )
Status = DftiSetValue( Desc_Handle_Dim2, DFTI_INPUT_DISTANCE, 1 )
Status = DftiSetValue( Desc_Handle_Dim2, DFTI_OUTPUT_DISTANCE, 1 )
Status = DftiSetValue( Desc_Handle_Dim2, DFTI_INPUT_STRIDES, Stride )
Status = DftiSetValue( Desc_Handle_Dim2, DFTI_OUTPUT_STRIDES, Stride )
Status = DftiCommitDescriptor( Desc_Handle_Dim2 )
```

```
Status = DftiComputeForward( Desc_Handle_Dim2, X )
Status = DftiFreeDescriptor( Desc_Handle_Dim1 )
Status = DftiFreeDescriptor( Desc_Handle_Dim2 )

/* C */
float _Complex x[20][40];
long stride[2];
long status;
DFTI_DESCRIPTOR_HANDLE desc_handle_dim1;
DFTI_DESCRIPTOR_HANDLE desc_handle_dim2;
...
status = DftiCreateDescriptor( &desc_handle_dim1, DFTI_SINGLE,
                              DFTI_COMPLEX, 1, 20 );
status = DftiCreateDescriptor( &desc_handle_dim2, DFTI_SINGLE,
                              DFTI_COMPLEX, 1, 40 );

/* perform 40 one-dimensional transforms along 1st dimension */
/* note that the 1st dimension data are not unit-stride */
stride[0] = 0; stride[1] = 40;
status = DftiSetValue( desc_handle_dim1, DFTI_NUMBER_OF_TRANSFORMS, 40 );
status = DftiSetValue( desc_handle_dim1, DFTI_INPUT_DISTANCE, 1 );
status = DftiSetValue( desc_handle_dim1, DFTI_OUTPUT_DISTANCE, 1 );
status = DftiSetValue( desc_handle_dim1, DFTI_INPUT_STRIDES, stride );
status = DftiSetValue( desc_handle_dim1, DFTI_OUTPUT_STRIDES, stride );
status = DftiCommitDescriptor( desc_handle_dim1 );
status = DftiComputeForward( desc_handle_dim1, x );

/* perform 20 one-dimensional transforms along 2nd dimension */
/* note that the 2nd dimension is unit stride */
status = DftiSetValue( desc_handle_dim2, DFTI_NUMBER_OF_TRANSFORMS, 20 );
status = DftiSetValue( desc_handle_dim2, DFTI_INPUT_DISTANCE, 40 );
status = DftiSetValue( desc_handle_dim2, DFTI_OUTPUT_DISTANCE, 40 );
status = DftiCommitDescriptor( desc_handle_dim2 );
status = DftiComputeForward( desc_handle_dim2, x );
```



```
status = DftiFreeDescriptor( &Desc_Handle_Dim1 );
status = DftiFreeDescriptor( &Desc_Handle_Dim2 );
```

## DFT 計算でのマルチスレッディングの使用例

次のプログラムは、インテル・マス・カーネル・ライブラリの DFT 計算で内部スレッディングを使用する方法を示す(「[ユーザスレッド数](#)」のケース 1 を参照)。

インテル・マス・カーネル・ライブラリ内のスレッド数を指定するには、次の設定を使用する。

set OMP\_NUM\_THREADS = 1 (シングルスレッド・モードの場合)

set OMP\_NUM\_THREADS = 4 (マルチスレッド・モードの場合)

構成パラメータ DFTI\_NUMBER\_OF\_USER\_THREADS は、デフォルト値 1 でなければならない。

### 例 C-22 インテル・マス・カーネル・ライブラリ内部スレッディング・モードの使用

```
#include "mkl_dfti.h"

void main () {

float x[200][100];
DFTI_DESCRIPTOR_HANDLE my_desc1_handle;
long status, len[2];
//...put input data into x[j][k] 0<=j<=199, 0<=k<=99
len[0] = 200; len[1] = 100;
status = DftiCreateDescriptor( &my_desc1_handle, DFTI_SINGLE,DFTI_REAL, 2,
len);
status = DftiCommitDescriptor( my_desc1_handle);
status = DftiComputeForward( my_desc1_handle, x);
status = DftiFreeDescriptor(&my_desc1_handle);
}
```

次の例 C-23 では、各ディスクリプタ・インスタンスがシングルスレッドでのみ使用される並列カスタム・プログラムについて説明する（「[ユーザスレッド数](#)」のケース 3 を参照）。

スレッド数を指定するには次の設定を使用する。

set MKL\_SERIAL = yes ( または YES ) ( インテル・マス・カーネル・ライブラリでシングルスレッド・モードの場合。推奨 )

set OMP\_NUM\_THREADS = 4 ( カスタム・プログラムでマルチスレッド・モードの場合 )

構成パラメータ DFTI\_NUMBER\_OF\_USER\_THREADS は、デフォルト値 1 でなければならない。

この例では、プログラムはカスタムレベルではシングルスレッドに変換されるが、インテル・マス・カーネル・ライブラリ内では並列モードで使用される。これには、パラメータ DFTI\_NUMBER\_OF\_TRANSFORMS = 4 および対応するパラメータ DFTI\_INPUT\_DISTANCE = 5000 を設定する必要がある。

### 例 C-23 複数のディスクリプタを伴う並列モードの使用

---

```
#include "mkl_dfti.h"
#include "omp.h"
void main ()
{
    float _Complex x[200][100];
    long len[2];
    //...put input data into x[j][k] 0<=j<=199, 0<=k<=99
    len[0] = 50; len[1] = 100;

    // each thread calculates real DFT for matrix (50*100)
    #pragma omp parallel
    {
        DFTI_DESCRIPTOR_HANDLE my_desc_handle;
        long myStatus;
        int myID = omp_get_thread_num ();
```

```
myStatus=DftiCreateDescriptor(my_desc_handle,DFTI_SINGLE,DFTI_COMPLEX,2,len);
    myStatus = DftiCommitDescriptor (my_desc_handle);
    myStatus = DftiComputeForward (my_desc_handle, &x[myID * len[0] * len[1]]);
    myStatus = DftiFreeDescriptor (&my_desc_handle);
} /* End OpenMP parallel region */
}
```

次の例 C-24 では、共通のディスクリプタが複数のスレッドで使用する並列カスタム・プログラムについて説明する(「[ユーザスレッド数](#)」のケース 3 を参照)。

この場合、DftiCommitDescriptor() 関数によって DFT の初期化が実行された後に、スレッド数と他の構成パラメータを変更してはならない。

#### 例 C-24 共通のディスクリプタを伴う並列モードの使用

```
// set number of threads inside Intel MKL:
//rem set MKL_SERIAL = YES      -   is not required since one-threaded mode for
Intel MKL is forced automatically
// set OMP_NUM_THREADS = 4      -   multi-threaded mode for customer

#include "mkl_dfti.h"
#include "omp.h"
void main ()
{
    float _Complex x[200][100];
    long status;
    DFTI_DESCRIPTOR_HANDLE desc_handle;
    int nThread = omp_get_max_threads ();
    long len[2];
    //...put input data into x[j][k] 0<=j<=199, 0<=k<=99
    len[0] = 50; len[1] = 100;

    status = DftiCreateDescriptor (desc_handle, DFTI_SINGLE, DFTI_COMPLEX, 2, len);
```

```
status = DftiSetValue (desc_handle, DFTI_NUMBER_OF_USER_THREADS, nThread);
status = DftiCommitDescriptor (desc_handle);

// each thread calculates real DFT for matrix (50*100)
#pragma omp parallel num_threads(nThread)
{
    long myStatus;
    int myID = omp_get_thread_num ();

    myStatus = DftiComputeForward (desc_handle, &x[myID * len[0] * len[1]]);
} /* End OpenMP parallel region */

status = DftiFreeDescriptor (&desc_handle);
}
```

多次元実数から CCE 格納形式の共役偶複素行列へのアウトオブプレース変換の例を次に示す。[例 C-25](#) は、Fortran インターフェイスでの 2 次元変換である。[例 C-26](#) は、C インターフェイスでの 3 次元変換である。計算関数のデータと結果パラメータは、すべて擬寸法階数 1 の配列 DIMENSION(0:\*) として宣言されている点に注意する。そのため、2 次元配列は EQUIVALENCE 文か Fortran のその他の機能を使用して 1 次元配列に変換しなければならない。

---

## 例 C-25      2 次元 REAL DFT (Fortran インターフェイス)

---

```
!Fortran example.
!2D and real to conjugate even
Use MKL_DFTI
Real :: X_2D(32,100)
Complex :: Y_2D(17, 100) !17 = 32/2 + 1
Real :: X(3200)
Complex :: Y(1700)
Equivalence (X_2D, X)
Equivalence (Y_2D, Y)
type(DFTI_DESCRIPTOR), POINTER :: My_Desc_Handle
```

```
Integer :: Status, L(2)
Integer :: strides_out(3)

...put input data into X_2D(j,k), 1<=j=32,1<=k=100
...set L(1) = 32, L(2) = 100
...set strides_out(1) = 0, strides_out(2) = 1, strides_out(3) = 17

...the transform is a 32-by-100
!Perform a real to complex conjugate even transform
Status = DftiCreateDescriptor( My_Desc_Handle, DFTI_SINGLE,
DFTI_REAL, 2, L)
Status = DftiSetValue(My_Desc_Handle, DFTI_CONJUGATE_EVEN_STORAGE,
DFTI_COMPLEX_COMPLEX)
Status = DftiSetValue(My_Desc_Handle, DFTI_OUTPUT_STRIDES, strides_out)

Status = DftiCommitDescriptor( My_Desc_Handle)
Status = DftiComputeForward( My_Desc_Handle, X, Y)
Status = DftiFreeDescriptor(My_Desc_Handle)
!result is given by the complex value z(j,k) 1<=j<=32; 1<=k<=100 and
!is stored in complex matrix Y_2D in CCE format.
```

---

**例 C-26      3次元 REAL DFT (C インターフェイス)**

---

```
/* C example */
#include "mkl_dfti.h"
float x[32][100][19];
float _Complex y[32][100][10]; /* 10 = 19/2 + 1 */
DFTI_DESCRIPTOR_HANDLE my_desc_handle
/* or alternatively
DFTI_DESCRIPTOR *my_desc_handle */
long status, l[3];
```

```
long strides_out[4];

...put input data into x[j][k][s] 0<=j<=31, 0<=k<=99, 0<=s<=18
l[0] = 32; l[1] = 100; l[2] = 19;
strides_out[0] = 0; strides_out[1] = 1000;
strides_out[2] = 10; strides_out[3] = 1;

status = DftiCreateDescriptor( &my_desc_handle, DFTI_SINGLE,
DFTI_REAL, 3, 1);
Status = DftiSetValue(my_desc_handle, DFTI_CONJUGATE_EVEN_STORAGE,
DFTI_COMPLEX_COMPLEX);
Status = DftiSetValue(My_Desc_Handle, DFTI_OUTPUT_STRIDES, strides_out);

status = DftiCommitDescriptor( my_desc_handle);
status = DftiComputeForward( my_desc_handle, x, y);
status = DftiFreeDescriptor(&my_desc_handle);
/* result is the complex value z(j,k,s) 0<=j<=31; 0<=k<=99, 0<=s<=18
and is stored in complex matrix y in CCE format.*/
```

## 区間連立 1 次方程式ソルバのコード例

このセクションでは、第 12 章の「[区間線形ソルバ](#)」で説明されたルーチンを使用するコード例を示す。これらのルーチンは、区間連立 1 次方程式のエンクロージャと解の集合の推定を算出し、区間行列とその逆転のプロパティをチェックする。

### 例 C-27 区間 Gauss-Seidel 法

区間連立 1 次代数方程式が与えられると、区間 Gauss-Seidel 法 ([?gegss](#) ルーチンとして実装される) は、指定された区間箱によって限定される解の集合の目的部分のエンクロージャに適用される。

E. Hansen によって最初に提案された (Hansen92 を参照)、次の区間連立 1 次方程式について考える。

$$\begin{pmatrix} [2, 3] & [0, 1] \\ [1, 2] & [2, 3] \end{pmatrix} x = \begin{pmatrix} [0, 120] \\ [60, 240] \end{pmatrix}$$

この解の集合は次の区間箱と交差するか？

$$\begin{pmatrix} [0, 200] \\ [0, 200] \end{pmatrix}$$

上記の質問への答えとなるプログラムを以下に示す。

---

```
PROGRAM DIGEGSS_EXAMPLE
!
!Example program enclosing the solution set to a square interval
!linear system by interval Gauss-Seidel iterative method
!
!-----!
USE INTERVAL_ARITHMETIC
IMPLICIT NONE
!-----!
```

```

INTEGER, PARAMETER :: DIM = 2
INTEGER :: NRHS, LDA, LDB, NITS, INFO, I, J
REAL(8) :: EPSILON
TYPE(D_INTERVAL) :: A(DIM,DIM), B(DIM,1), ENCL(DIM,1)
CHARACTER(1) :: TRANS

!-----!
PRINT 300
!-----!
!!
!Initializing the input data - !
!!
TRANS = 'N'
NRHS = 2
A(1,1) = DINTERVAL(2.,3.); A(1,2) = DINTERVAL(0.,1.);
A(2,1) = DINTERVAL(0.,1.); A(2,2) = DINTERVAL(2.,3.);
LDA = 2
B(1,1) = DINTERVAL(0.,120.); B(2,1) = DINTERVAL(60.,240.);
LDB = 2
EPSILON = 1.D-6
NITS = 20
!-----!
!
!Assigning the bounding box for the solution set -
DO I = 1, DIM
  ENCL(I,1) = DINTERVAL(0.,200.)
END DO
!-----!
CALL DIGEGSS(TRANS, DIM, NRHS, A, LDA, B, LDB, ENCL, EPSILON, NITS, INFO )
!-----!
!
!Outputting the solution
IF( INFO /= 0 ) THEN
PRINT 400

```



```

ELSE
PRINT 600
DO I = 1, DIM
PRINT *, '[', B(I,1), ']'
END DO
END IF

!-----!
300 FORMAT (/, ' **** SOLVING INTERVAL LINEAR SYSTEM **** ', /, &
' by interval Gauss-Seidel method ')
400 FORMAT (/, ' The interval Gauss-Seidel method fails. ')
600 FORMAT (/, ' Outer interval estimate of the solution set:', /)
!-----!
END PROGRAM DIGEGSS_EXAMPLE

```

2つの実数からそれぞれを終点とする区間を作成する DINTERVAL 関数によって、行列 **A** と右辺ベクトル **B** の成分に倍精度区間が割り当てられる。上記のコードを実行すると解答が得られる。

```

**** SOLVING INTERVAL LINEAR SYSTEM ****
by interval Gauss-Seidel method
Outer interval estimate of the solution set:
[ 0.0000000000000000E+000 60.00000000000000 ]
[ 0.0000000000000000E+000 120.0000000000000 ]

```

[Hansen92](#)にある対応するグラフを参照すると、作成された区間箱が解の集合の必要な部分のエンクロージャとなることを確認できる。さらに、最も厳密なエンクロージャであることも確認できる。

## 例 C-28 Hansen-Bliek-Rohn プロシージャ

次の Fortran-90 のプログラムは、区間連立 1 次方程式の解の集合の外側区間を推定する、“半区間” Hansen-Bliek-Rohn プロシージャを実装する digehbs ルーチンの使用方法を示す。

```
PROGRAM DIGEHBS_EXAMPLE
!
!Example program for enclosing the solution set to square interval
!interval system of equations by Hansen-Bliek-Rohn procedure
!
!-----!
USE INTERVAL_ARITHMETIC
IMPLICIT NONE
!-----!
INTEGER, PARAMETER :: DIM = 2
INTEGER :: LDA, LDB, INFO, I, J
TYPE(D_INTERVAL), ALLOCATABLE :: A(:, :), B(:)
CHARACTER(1) :: TRANS
!-----!
PRINT 300
!-----!
!
!Initializing the input data -
!
TRANS = 'N'
ALLOCATE( A(DIM,DIM), B(DIM) )
A(1,1) = DINTERVAL(2.,4.); A(1,2) = DINTERVAL(-2.,1.)
A(2,1) = DINTERVAL(-1.,2.); A(2,2) = DINTERVAL(2.,2.)
LDA = 2
B(1) = DINTERVAL(0.,2.); B(2) = DINTERVAL(0.,2.)
LDB = 2
```

```

!-----!
CALL DIGEHBS( TRANS, DIM, A, LDA, B, LDB, INFO )
!-----!
IF( INFO /= 0 ) THEN
PRINT 400
ELSE
PRINT 600
DO I = 1, DIM
PRINT *, I, ' ) [ ', B(I), ' ]'
END DO
END IF
!-----!
DEALLOCATE( A, B )
!-----!
300 FORMAT (/, ' **** SOLVING INTERVAL LINEAR SYSTEM ****', /, &
' by Hansen-Bliek-Rohn procedure ', /)
400 FORMAT (/, ' The matrix of the system is not an H-matrix,', /, &
' Hansen-Bliek-Rohn procedure fails.', /)
600 FORMAT (/, ' Enclosure of the solution set: ', /)
!-----!
END PROGRAM DIGEHBS_EXAMPLE

```

ただし、プログラムの出力は次のようになる。

```

**** SOLVING INTERVAL LINEAR SYSTEM ****
by Hansen-Bliek-Rohn procedure
The matrix of the system is not an H-matrix,
Hansen-Bliek-Rohn procedure fails.

```

これは、プログラムが次の区間連立 1 次方程式に適用されたためである。

$$\begin{pmatrix} [2, 4] & [-2, 1] \\ [-1, 2] & [2, 4] \end{pmatrix}_X = \begin{pmatrix} [0, 2] \\ [0, 2] \end{pmatrix}$$

ここで、区間行列は H- 行列 ( 優対角性を持たない ) ではない。

ただし、digemip ルーチンによるプリコンディショニングは問題の解決に役立つ。結果に区間連立 1 次方程式の予備のプリコンディショニングを組み込む次の変更されたプログラムは、行列に優対角性を持たせ、問題に対して許容範囲の結果を算出する。

---

```
PROGRAM DIGEMIP_DIGEHS_EXAMPLE
!
!Example program for enclosing the solution set to square interval
!interval system of equations by Hansen-Bliek-Rohn procedure
!
!-----!
USE INTERVAL_ARITHMETIC
IMPLICIT NONE
!-----!
INTEGER, PARAMETER :: DIM = 2, NRHS = 1
INTEGER :: LDA, LDB, INFO, I, J
TYPE(D_INTERVAL), ALLOCATABLE :: A(:, :), B(:)
CHARACTER(1) :: TRANS
!-----!
PRINT 300
!-----!
!
!Initializing the input data -
!
TRANS = 'N'
ALLOCATE( A(DIM,DIM), B(DIM) )
A(1,1) = DINTERVAL(2.,4.); A(1,2) = DINTERVAL(-2.,1.)
A(2,1) = DINTERVAL(-1.,2.); A(2,2) = DINTERVAL(2.,2.)
LDA = 2
B(1) = DINTERVAL(0.,2.); B(2) = DINTERVAL(0.,2.)
LDB = 2
!-----!
CALL DIGEMIP( DIM, NRHS, A, LDA, B, LDB, INFO )
```

```
CALL DIGEHBS( TRANS, DIM, A, LDA, B, LDB, INFO )
!-----!
IF( INFO /= 0 ) THEN
PRINT 400
ELSE
PRINT 600
DO I = 1, DIM
PRINT *, I, ' ) [ ', B(I), ' ] '
END DO
END IF
DEALLOCATE( A, B )
!-----!
300 FORMAT (/, ' **** SOLVING INTERVAL LINEAR SYSTEM ****', /, &
' by Hansen-Bliek-Rohn procedure ', /)
400 FORMAT (/, ' The matrix of the system is not an H-matrix,', /, &
' Hansen-Bliek-Rohn procedure fails.', /)
600 FORMAT (/, ' Enclosure of the solution set: ', /)
!-----!
END PROGRAM DIGEMIP_DIGEHBS_EXAMPLE
```

---

この場合、プログラムの出力は次のようになる。

```
**** SOLVING INTERVAL LINEAR SYSTEM ****
by Hansen-Bliek-Rohn procedure
Enclosure of the solution set:
1 ) [ -4.23529411764708 10.7058823529412 ]
2 ) [ -6.70588235294119 10.8235294117647 ]
```

( コンピュータ・アーキテクチャによって最後の桁は異なる。 )

**例 C-29 逆区間行列のエンクロージャの計算**

---

次のような  $2 \times 2$  区間行列が与えられた場合、

$$\begin{pmatrix} 3 & [0, 1] \\ [1, 2] & [2, 3] \end{pmatrix}$$

次の Fortran-90 のコードは、逆区間行列のエンクロージャを計算する。

---

```

PROGRAM SIGESZI_EXAMPLE
!
!Example program inverting an interval matrix by Sczulz iterative procedure
!
!-----!
USE INTERVAL_ARITHMETIC
IMPLICIT NONE
!-----!
INTEGER, PARAMETER :: DIM = 2, LDA = 2
INTEGER :: INFO, I, J
TYPE(S_INTERVAL), ALLOCATABLE :: A(:, :)
!-----!
PRINT 300
!-----!
!
!Initislizing the input data -
!
ALLOCATE( A(LDA,DIM) )
A(1,1) = SINTERVAL(3.,3.); A(1,2) = SINTERVAL(0.,1.)
A(2,1) = SINTERVAL(1.,2.); A(2,2) = SINTERVAL(2.,3.)
!-----!
CALL SIGESZI ( DIM, A, LDA, INFO )
!-----!
PRINT 600

```

```

DO I = 1, DIM
PRINT *, ( '[', A(I,J), ']', J = 1, DIM )
END DO
DEALLOCATE( A )
!-----!
300 FORMAT (/, ' **** INVERTING INTERVAL MATRIX ****', /, &
' by interval Schulz method ' )
400 FORMAT (/, ' Schulz inversion procedure failed.', /)
600 FORMAT (/, ' Enclosure of the inverse matrix ', /)
!-----!
END PROGRAM SIGESZI_EXAMPLE

```

出力は次のようになる (アーキテクチャにより若干の違いがある)。

```

**** INVERTING INTERVAL MATRIX ****
by interval Schulz method
Enclosure of the inverse matrix
[ 0.2407409 0.5000001 ][ -0.2500000 0.1018518 ]
[ -0.5000000 5.5555239E-02 ][ 0.1388889 0.7500001 ]

```

同時に、行列の (1,1) 成分を区間 [2, 3] に変更すると、sigeszi プロシージャは新しい区間行列に対する逆の有限エンクロージャの計算に失敗する。

$$\begin{pmatrix} [2, 3] & [0, 1] \\ [1, 2] & [2, 3] \end{pmatrix}$$

しかし、このような行列を持つ区間連立 1 次方程式は、区間ガウス法や区間 Gauss-Seidel 法など、特別なルーチンを用いて解くことができる ([例 C-27](#) を参照)。





# BLAS に対する CBLAS インターフェイス



本付録では、CBLAS について説明する。CBLAS は、インテル<sup>®</sup> MKL に導入された BLAS (Basic Linear Algebra Subprograms) に対する C インターフェイスである。

BLAS の場合と同じように、CBLAS インターフェイスには、以下に示すレベルの関数が含まれる。

- 「[レベル 1 の CBLAS](#)」(ベクトル - ベクトル演算)
- 「[レベル 2 の CBLAS](#)」(行列 - ベクトル演算)
- 「[レベル 3 の CBLAS](#)」(行列 - 行列演算)
- 「[スパース CBLAS](#)」(スパースベクトルでの演算)

C インターフェイスを得るには、Fortran ルーチン名の先頭に `cblas_` を付ける (例えば、`dasum` は `cblas_dasum` となる)。すべての CBLAS 関数は、小文字で表す。

複素関数 `?dotc` と `?dotu` は、CBLAS のサブルーチン (void 関数) になる。これらの関数は、最後のパラメータとして追加される void ポインタを介して複素数型の結果を返す。これらの関数の CBLAS 名には、末尾に `_sub` を付ける。例えば、BLAS 関数 `cdotc` は `cblas_cdotc_sub` に対応する。

## CBLAS の引数

CBLAS 関数の引数は、以下の規則に従る。

- 入力引数は、`const` 修飾子で宣言する。
- 非複素数のスカラ入力引数は、値で渡される。
- 複素数のスカラ入力引数は、void ポインタとして渡される。
- 配列の引数は、アドレスで渡される。

- 出力のスカラー引数は、アドレスで渡される。
- BLAS のキャラクタ引数は、該当する列挙型で置き換えられる。
- レベル 2 とレベル 3 のルーチンは、最初の引数として型 CBLAS\_ORDER の追加パラメータをとる。このパラメータは、2 次元配列が行主体 (CblasRowMajor) か列主体 (CblasColMajor) のどちらであるかを指定する。

## 列挙型

CBLAS インターフェイスは、以下に示す列挙型を使用する。

```
enum CBLAS_ORDER {
    CblasRowMajor=101,          /* row-major arrays */
    CblasColMajor=102};        /* column-major arrays */

enum CBLAS_TRANSPOSE {
    CblasNoTrans=111,          /* trans='N' */
    CblasTrans=112,            /* trans='T' */
    CblasConjTrans=113};       /* trans='C' */

enum CBLAS_UPLO {
    CblasUpper=121,            /* uplo ='U' */
    CblasLower=122};           /* uplo ='L' */

enum CBLAS_DIAG {
    CblasNonUnit=131,          /* diag ='N' */
    CblasUnit=132};            /* diag ='U' */

enum CBLAS_SIDE {
    CblasLeft=141,              /* side ='L' */
    CblasRight=142};           /* side ='R' */
```

## レベル 1 の CBLAS

基本的なベクトル - ベクトル演算を実行する「[BLAS レベル 1 のルーチンと関数](#)」へのインターフェイスである。

### [ipps?asum](#)

```
float cblas_sasum(const int N, const float *X, const int incX);
double cblas_dasum(const int N, const double *X, const int incX);
float cblas_scasum(const int N, const void *X, const int incX);
double cblas_dzasum(const int N, const void *X, const int incX);
```

### [ipps?axpy](#)

```
void cblas_saxpy(const int N, const float alpha, const float *X, const int incX,
float *Y, const int incY);
void cblas_daxpy(const int N, const double alpha, const double *X, const int
incX, double *Y, const int incY);
void cblas_caxpy(const int N, const void *alpha, const void *X, const int incX,
void *Y, const int incY);
void cblas_zaxpy(const int N, const void *alpha, const void *X, const int incX,
void *Y, const int incY);
```

### [ipps?copy](#)

```
void cblas_scopy(const int N, const float *X, const int incX, float *Y, const int
incY);
void cblas_dcopy(const int N, const double *X, const int incX, double *Y, const
int incY);
void cblas_ccopy(const int N, const void *X, const int incX, void *Y, const int
incY);
void cblas_zcopy(const int N, const void *X, const int incX, void *Y, const int
incY);
```

### [ipps?dot](#)

```
float cblas_sdot(const int N, const float *X, const int incX,
const float *Y, const int incY);
double cblas_ddot(const int N, const double *X, const int incX,
const double *Y, const int incY);
```

### [ipps?sdot](#)

```
float cblas_sdsdot(const int N, const float *SB, const float *SX, const int incX,
const float *SY, const int incY);
double cblas_dsdot(const int N, const float *SX, const int incX, const float *SY,
const int incY);
```

## [ipps?dotc](#)

```
void cblas_cdotc_sub(const int N, const void *X, const int incX, const void *Y,
const int incY, void *dotc);
```

```
void cblas_zdotc_sub(const int N, const void *X, const int incX, const void *Y,
const int incY, void *dotc);
```

## [ipps?dotu](#)

```
void cblas_cdotu_sub(const int N, const void *X, const int incX, const void *Y,
const int incY, void *dotu);
```

```
void cblas_zdotu_sub(const int N, const void *X, const int incX, const void *Y,
const int incY, void *dotu);
```

## [ipps?nrm2](#)

```
float cblas_snrm2(const int N, const float *X, const int incX);
```

```
double cblas_dnrm2(const int N, const double *X, const int incX);
```

```
float cblas_scnrm2(const int N, const void *X, const int incX);
```

```
double cblas_dznrm2(const int N, const void *X, const int incX);
```

## [ipps?rot](#)

```
void cblas_srot(const int N, float *X, const int incX, float *Y, const int incY,
const float c, const float s);
```

```
void cblas_drot(const int N, double *X, const int incX, double *Y, const int incY,
const double c, const double s);
```

## [ipps?rotg](#)

```
void cblas_srotg(float *a, float *b, float *c, float *s);
```

```
void cblas_drotg(double *a, double *b, double *c, double *s);
```

## [ipps?rotm](#)

```
void cblas_srotm(const int N, float *X, const int incX, float *Y, const int incY,
const float *P);
```

```
void cblas_drotm(const int N, double *X, const int incX, double *Y, const int
incY, const double *P);
```

## [ipps?rotmg](#)

```
void cblas_srotmg(float *d1, float *d2, float *b1, const float b2, float *P);
```

```
void cblas_drotmg(double *d1, double *d2, double *b1, const double b2, double
*P);
```

**ipps?scal**

```
void cblas_sscal(const int N, const float alpha, float *X, const int incX);
void cblas_dscal(const int N, const double alpha, double *X, const int incX);
void cblas_cscal(const int N, const void *alpha, void *X, const int incX);
void cblas_zscal(const int N, const void *alpha, void *X, const int incX);
void cblas_csscal(const int N, const float alpha, void *X, const int incX);
void cblas_zdscal(const int N, const double alpha, void *X, const int incX);
```

**ipps?swap**

```
void cblas_sswap(const int N, float *X, const int incX, float *Y, const int incY);
void cblas_dswap(const int N, double *X, const int incX, double *Y, const int incY);
void cblas_cswap(const int N, void *X, const int incX, void *Y, const int incY);
void cblas_zswap(const int N, void *X, const int incX, void *Y, const int incY);
```

**ippsi?amax**

```
CBLAS_INDEX cblas_isamax(const int N, const float *X, const int incX);
CBLAS_INDEX cblas_idamax(const int N, const double *X, const int incX);
CBLAS_INDEX cblas_icamax(const int N, const void *X, const int incX);
CBLAS_INDEX cblas_izamax(const int N, const void *X, const int incX);
```

**ippsi?amin**

```
CBLAS_INDEX cblas_isamin(const int N, const float *X, const int incX);
CBLAS_INDEX cblas_idamin(const int N, const double *X, const int incX);
CBLAS_INDEX cblas_icamin(const int N, const void *X, const int incX);
CBLAS_INDEX cblas_izamin(const int N, const void *X, const int incX);
```

## レベル 2 の CBLAS

基本的な行列 - ベクトル演算を実行する「[BLAS レベル 2 のルーチン](#)」へのインターフェイスである。このグループに属する各 C ルーチンは、型 CBLAS\_ORDER の追加パラメータ (最初の引数) をとる。このパラメータは、2 次元の配列が、列主体か行主体のどちらの格納方式を使用するかを決定する。

**ipps?gbmv**

```
void cblas_sgbmv(const enum CBLAS_ORDER order, const enum CBLAS_TRANSPOSE TransA,
const int M, const int N, const int KL, const int KU, const float alpha, const
float *A, const int lda, const float *X, const int incX, const float beta, float
*Y, const int incY);
```

```
void cblas_dgbmv(const enum CBLAS_ORDER order, const enum CBLAS_TRANSPOSE TransA,
const int M, const int N, const int KL, const int KU, const double alpha, const
double *A, const int lda, const double *X, const int incX, const double beta,
double *Y, const int incY);
```

```
void cblas_cgbmv(const enum CBLAS_ORDER order, const enum CBLAS_TRANSPOSE TransA,
const int M, const int N, const int KL, const int KU, const void *alpha, const
void *A, const int lda, const void *X, const int incX, const void *beta, void *Y,
const int incY);
```

```
void cblas_zgbmv(const enum CBLAS_ORDER order, const enum CBLAS_TRANSPOSE TransA,
const int M, const int N, const int KL, const int KU, const void *alpha, const
void *A, const int lda, const void *X, const int incX, const void *beta, void *Y,
const int incY);
```

## [ipps?gemv](#)

```
void cblas_sgemv(const enum CBLAS_ORDER order, const enum CBLAS_TRANSPOSE TransA,
const int M, const int N, const float alpha, const float *A, const int lda, const
float *X, const int incX, const float beta, float *Y, const int incY);
```

```
void cblas_dgemv(const enum CBLAS_ORDER order, const enum CBLAS_TRANSPOSE TransA,
const int M, const int N, const double alpha, const double *A, const int lda,
const double *X, const int incX, const double beta, double *Y, const int incY);
```

```
void cblas_cgemv(const enum CBLAS_ORDER order, const enum CBLAS_TRANSPOSE TransA,
const int M, const int N, const void *alpha, const void *A, const int lda, const
void *X, const int incX, const void *beta, void *Y, const int incY);
```

```
void cblas_zgemv(const enum CBLAS_ORDER order, const enum CBLAS_TRANSPOSE TransA,
const int M, const int N, const void *alpha, const void *A, const int lda, const
void *X, const int incX, const void *beta, void *Y, const int incY);
```

## [ipps?ger](#)

```
void cblas_sger(const enum CBLAS_ORDER order, const int M, const int N, const
float alpha, const float *X, const int incX, const float *Y, const int incY, float
*A, const int lda);
```

```
void cblas_dger(const enum CBLAS_ORDER order, const int M, const int N, const
double alpha, const double *X, const int incX, const double *Y, const int incY,
double *A, const int lda);
```

## [ipps?gerc](#)

```
void cblas_cgerc(const enum CBLAS_ORDER order, const int M, const int N, const
void *alpha, const void *X, const int incX, const void *Y, const int incY, void
*A, const int lda);
```

```
void cblas_zgerc(const enum CBLAS_ORDER order, const int M, const int N, const
void *alpha, const void *X, const int incX, const void *Y, const int incY, void
*A, const int lda);
```

[ipps?geru](#)

```
void cblas_cgeru(const enum CBLAS_ORDER order, const int M, const int N, const
void *alpha, const void *X, const int incX, const void *Y, const int incY, void
*A, const int lda);
```

```
void cblas_zgeru(const enum CBLAS_ORDER order, const int M, const int N, const
void *alpha, const void *X, const int incX, const void *Y, const int incY, void
*A, const int lda);
```

[ipps?hbmrv](#)

```
void cblas_chbmrv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
int N, const int K, const void *alpha, const void *A, const int lda, const void
*X, const int incX, const void *beta, void *Y, const int incY);
```

```
void cblas_zhbmrv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
int N, const int K, const void *alpha, const void *A, const int lda, const void
*X, const int incX, const void *beta, void *Y, const int incY);
```

[ipps?hemv](#)

```
void cblas_chemv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
int N, const void *alpha, const void *A, const int lda, const void *X, const int
incX, const void *beta, void *Y, const int incY);
```

```
void cblas_zhemv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
int N, const void *alpha, const void *A, const int lda, const void *X, const int
incX, const void *beta, void *Y, const int incY);
```

[ipps?her](#)

```
void cblas_cher(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
int N, const float alpha, const void *X, const int incX, void *A, const int lda);
```

```
void cblas_zher(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
int N, const double alpha, const void *X, const int incX, void *A, const int lda);
```

[ipps?her2](#)

```
void cblas_cher2(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
int N, const void *alpha, const void *X, const int incX, const void *Y, const int
incY, void *A, const int lda);
```

```
void cblas_zher2(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
int N, const void *alpha, const void *X, const int incX, const void *Y, const int
incY, void *A, const int lda);
```

[ipps?hpmv](#)

```
void cblas_chpmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
int N, const void *alpha, const void *Ap, const void *X, const int incX, const
void *beta, void *Y, const int incY);
```

```
void cblas_zhpmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
int N, const void *alpha, const void *Ap, const void *X, const int incX, const
void *beta, void *Y, const int incY);
```

## [ipps?hpr](#)

```
void cblas_chpr(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
int N, const float alpha, const void *X, const int incX, void *A);
void cblas_zhpr(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
int N, const double alpha, const void *X, const int incX, void *A);
```

## [ipps?hpr2](#)

```
void cblas_chpr2(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
int N, const void *alpha, const void *X, const int incX, const void *Y, const int
incY, void *Ap);
void cblas_zhpr2(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
int N, const void *alpha, const void *X, const int incX, const void *Y, const int
incY, void *Ap);
```

## [ipps?sbmv](#)

```
void cblas_ssbmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
int N, const int K, const float alpha, const float *A, const int lda, const float
*X, const int incX, const float beta, float *Y, const int incY);
void cblas_dsbmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
int N, const int K, const double alpha, const double *A, const int lda, const
double *X, const int incX, const double beta, double *Y, const int incY);
```

## [ipps?spmv](#)

```
void cblas_sspmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
int N, const float alpha, const float *Ap, const float *X, const int incX, const
float beta, float *Y, const int incY);
void cblas_dspmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
int N, const double alpha, const double *Ap, const double *X, const int incX,
const double beta, double *Y, const int incY);
```

## [ipps?spr](#)

```
void cblas_sspr(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
int N, const float alpha, const float *X, const int incX, float *Ap);
void cblas_dspr(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
int N, const double alpha, const double *X, const int incX, double *Ap);
```

## [ipps?spr2](#)

```
void cblas_sspr2(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
int N, const float alpha, const float *X, const int incX, const float *Y, const
int incY, float *A);
```



```
void cblas_dspr2(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
int N, const double alpha, const double *X, const int incX, const double *Y, const
int incY, double *A);
```

### [ipps?symv](#)

```
void cblas_ssymv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
int N, const float alpha, const float *A, const int lda, const float *X, const int
incX, const float beta, float *Y, const int incY);
```

```
void cblas_dsymv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
int N, const double alpha, const double *A, const int lda, const double *X, const
int incX, const double beta, double *Y, const int incY);
```

### [ipps?syr](#)

```
void cblas_ssyr(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
int N, const float alpha, const float *X, const int incX, float *A, const int
lda);
```

```
void cblas_dsyr(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
int N, const double alpha, const double *X, const int incX, double *A, const int
lda);
```

### [ipps?syr2](#)

```
void cblas_ssyr2(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
int N, const float alpha, const float *X, const int incX, const float *Y, const
int incY, float *A, const int lda);
```

```
void cblas_dsyr2(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
int N, const double alpha, const double *X, const int incX, const double *Y, const
int incY, double *A, const int lda);
```

### [ipps?tbmv](#)

```
void cblas_stbmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag, const int N, const int
K, const float *A, const int lda, float *X, const int incX);
```

```
void cblas_dtbmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag, const int N, const int
K, const double *A, const int lda, double *X, const int incX);
```

```
void cblas_ctbmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag, const int N, const int
K, const void *A, const int lda, void *X, const int incX);
```

```
void cblas_ztbmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag, const int N, const int
K, const void *A, const int lda, void *X, const int incX);
```

## [ipps?tbsv](#)

```
void cblas_stbsv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag, const int N, const int
K, const float *A, const int lda, float *X, const int incX);
```

```
void cblas_dtbsv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag, const int N, const int
K, const double *A, const int lda, double *X, const int incX);
```

```
void cblas_ctbsv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag, const int N, const int
K, const void *A, const int lda, void *X, const int incX);
```

```
void cblas_ztbsv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag, const int N, const int
K, const void *A, const int lda, void *X, const int incX);
```

## [ipps?tpmv](#)

```
void cblas_stpmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag, const int N, const float
*Ap, float *X, const int incX);
```

```
void cblas_dtpmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag, const int N, const
double *Ap, double *X, const int incX);
```

```
void cblas_ctpmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag, const int N, const void
*Ap, void *X, const int incX);
```

```
void cblas_ztpmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag, const int N, const void
*Ap, void *X, const int incX);
```

## [ipps?tpsv](#)

```
void cblas_stpsv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag, const int N, const float
*Ap, float *X, const int incX);
```

```
void cblas_dtpsv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag, const int N, const double
*Ap, double *X, const int incX);
```

```
void cblas_ctpsv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag, const int N, const void
*Ap, void *X, const int incX);
```

```
void cblas_ztpsv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag, const int N, const void
*Ap, void *X, const int incX);
```

[ipps?trmv](#)

```
void cblas_strmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag, const int N,const float
*A, const int lda, float *X, const int incX);
```

```
void cblas_dtrmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag, const int N,const double
*A, const int lda, double *X, const int incX);
```

```
void cblas_ctrmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag, const int N,const void
*A, const int lda, void *X, const int incX);
```

```
void cblas_ztrmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag, const int N,const void
*A, const int lda, void *X, const int incX);
```

[ipps?trsv](#)

```
void cblas_strsv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag, const int N,const float
*A, const int lda, float *X, const int incX);
```

```
void cblas_dtrsv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag, const int N,const double
*A, const int lda, double *X, const int incX);
```

```
void cblas_ctrsv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag, const int N,const void
*A, const int lda, void *X, const int incX);
```

```
void cblas_ztrsv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag, const int N,const void
*A, const int lda, void *X, const int incX);
```

## レベル 3 の CBLAS

基本的な行列 - 行列演算を実行する「[BLAS レベル 3 のルーチン](#)」へのインターフェイスである。このグループに属する各 C ルーチンは、型 CBLAS\_ORDER の追加パラメータ (最初の引数) をとる。このパラメータは、2 次元の配列が、列主体か行主体のどちらの格納方式を使用するかを決定する。

### [ipps?gemm](#)

```
void cblas_sgemm(const enum CBLAS_ORDER Order, const enum CBLAS_TRANSPOSE TransA,
const enum CBLAS_TRANSPOSE TransB, const int M, const int N, const int K, const
float alpha, const float *A, const int lda, const float *B, const int ldb, const
float beta, float *C, const int ldc);

void cblas_dgemm(const enum CBLAS_ORDER Order, const enum CBLAS_TRANSPOSE TransA,
const enum CBLAS_TRANSPOSE TransB, const int M, const int N, const int K, const
double alpha, const double *A, const int lda, const double *B, const int ldb,
const double beta, double *C, const int ldc);

void cblas_cgemm(const enum CBLAS_ORDER Order, const enum CBLAS_TRANSPOSE TransA,
const enum CBLAS_TRANSPOSE TransB, const int M, const int N, const int K, const
void *alpha, const void *A, const int lda, const void *B, const int ldb, const
void *beta, void *C, const int ldc);

void cblas_zgemm(const enum CBLAS_ORDER Order, const enum CBLAS_TRANSPOSE TransA,
const enum CBLAS_TRANSPOSE TransB, const int M, const int N, const int K, const
void *alpha, const void *A, const int lda, const void *B, const int ldb, const
void *beta, void *C, const int ldc);
```

### [ipps?hemm](#)

```
void cblas_chemm(const enum CBLAS_ORDER Order, const enum CBLAS_SIDE Side, const
enum CBLAS_UPLO Uplo, const int M, const int N, const void *alpha, const void *A,
const int lda, const void *B, const int ldb, const void *beta, void *C, const int
ldc);

void cblas_zhemm(const enum CBLAS_ORDER Order, const enum CBLAS_SIDE Side, const
enum CBLAS_UPLO Uplo, const int M, const int N, const void *alpha, const void *A,
const int lda, const void *B, const int ldb, const void *beta, void *C, const int
ldc);
```

### [ipps?herk](#)

```
void cblas_cherk(const enum CBLAS_ORDER Order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE Trans, const int N, const int K, const float alpha, const
void *A, const int lda, const float beta, void *C, const int ldc);

void cblas_zherk(const enum CBLAS_ORDER Order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE Trans, const int N, const int K, const double alpha, const
void *A, const int lda, const double beta, void *C, const int ldc);
```

[ipps?her2k](#)

```
void cblas_cher2k(const enum CBLAS_ORDER Order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE Trans, const int N, const int K, const void *alpha, const
void *A, const int lda, const void *B, const int ldb, const float beta, void *C,
const int ldc);
```

```
void cblas_zher2k(const enum CBLAS_ORDER Order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE Trans, const int N, const int K, const void *alpha, const
void *A, const int lda, const void *B, const int ldb, const double beta, void *C,
const int ldc);
```

[ipps?symm](#)

```
void cblas_ssymm(const enum CBLAS_ORDER Order, const enum CBLAS_SIDE Side, const
enum CBLAS_UPLO Uplo, const int M, const int N, const float alpha, const float *A,
const int lda, const float *B, const int ldb, const float beta, float *C, const
int ldc);
```

```
void cblas_dsymm(const enum CBLAS_ORDER Order, const enum CBLAS_SIDE Side, const
enum CBLAS_UPLO Uplo, const int M, const int N, const double alpha, const double
*A, const int lda, const double *B, const int ldb, const double beta, double *C,
const int ldc);
```

```
void cblas_csymm(const enum CBLAS_ORDER Order, const enum CBLAS_SIDE Side, const
enum CBLAS_UPLO Uplo, const int M, const int N, const void *alpha, const void *A,
const int lda, const void *B, const int ldb, const void *beta, void *C, const int
ldc);
```

```
void cblas_zsymm(const enum CBLAS_ORDER Order, const enum CBLAS_SIDE Side, const
enum CBLAS_UPLO Uplo, const int M, const int N, const void *alpha, const void *A,
const int lda, const void *B, const int ldb, const void *beta, void *C, const int
ldc);
```

[ipps?syrk](#)

```
void cblas_ssyrk(const enum CBLAS_ORDER Order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE Trans, const int N, const int K, const float alpha, const
float *A, const int lda, const float beta, float *C, const int ldc);
```

```
void cblas_dsyrk(const enum CBLAS_ORDER Order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE Trans, const int N, const int K, const double alpha, const
double *A, const int lda, const double beta, double *C, const int ldc);
```

```
void cblas_csyrk(const enum CBLAS_ORDER Order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE Trans, const int N, const int K, const void *alpha, const
void *A, const int lda, const void *beta, void *C, const int ldc);
```

```
void cblas_zsyrk(const enum CBLAS_ORDER Order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE Trans, const int N, const int K, const void *alpha, const
void *A, const int lda, const void *beta, void *C, const int ldc);
```

## [ipps?syr2k](#)

```
void cblas_ssyr2k(const enum CBLAS_ORDER Order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE Trans, const int N, const int K, const float alpha, const
float *A, const int lda, const float *B, const int ldb, const float beta, float
*C, const int ldc);
```

```
void cblas_dsyr2k(const enum CBLAS_ORDER Order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE Trans, const int N, const int K, const double alpha, const
double *A, const int lda, const double *B, const int ldb, const double beta,
double *C, const int ldc);
```

```
void cblas_csyr2k(const enum CBLAS_ORDER Order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE Trans, const int N, const int K, const void *alpha, const
void *A, const int lda, const void *B, const int ldb, const void *beta, void *C, const
int ldc);
```

```
void cblas_zsyr2k(const enum CBLAS_ORDER Order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE Trans, const int N, const int K, const void *alpha, const
void *A, const int lda, const void *B, const int ldb, const void *beta, void *C,
const int ldc);
```

## [ipps?trmm](#)

```
void cblas_strmm(const enum CBLAS_ORDER Order, const enum CBLAS_SIDE Side, const
enum CBLAS_UPLO Uplo, const enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG
Diag, const int M, const int N, const float alpha, const float *A, const int lda,
float *B, const int ldb);
```

```
void cblas_dtrmm(const enum CBLAS_ORDER Order, const enum CBLAS_SIDE Side, const
enum CBLAS_UPLO Uplo, const enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG
Diag, const int M, const int N, const double alpha, const double *A, const int
lda, double *B, const int ldb);
```

```
void cblas_ctrmm(const enum CBLAS_ORDER Order, const enum CBLAS_SIDE Side, const
enum CBLAS_UPLO Uplo, const enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG
Diag, const int M, const int N, const void *alpha, const void *A, const int lda,
void *B, const int ldb);
```

```
void cblas_ztrmm(const enum CBLAS_ORDER Order, const enum CBLAS_SIDE Side, const
enum CBLAS_UPLO Uplo, const enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG
Diag, const int M, const int N, const void *alpha, const void *A, const int lda,
void *B, const int ldb);
```

## [ipps?trsm](#)

```
void cblas_strsm(const enum CBLAS_ORDER Order, const enum CBLAS_SIDE Side, const
enum CBLAS_UPLO Uplo, const enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG
Diag, const int M, const int N, const float alpha, const float *A, const int lda,
float *B, const int ldb);
```

```
void cblas_dtrsm(const enum CBLAS_ORDER Order, const enum CBLAS_SIDE Side, const
enum CBLAS_UPLO Uplo, const enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG
Diag, const int M, const int N, const double alpha, const double *A, const int
lda, double *B, const int ldb);
```

```
void cblas_ctrsm(const enum CBLAS_ORDER Order, const enum CBLAS_SIDE Side, const
enum CBLAS_UPLO Uplo, const enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG
Diag, const int M, const int N, const void *alpha, const void *A, const int lda,
void *B, const int ldb);
```

```
void cblas_ztrsm(const enum CBLAS_ORDER Order, const enum CBLAS_SIDE Side, const
enum CBLAS_UPLO Uplo, const enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG
Diag, const int M, const int N, const void *alpha, const void *A, const int lda,
void *B, const int ldb);
```

## スパース CBLAS

「[スパース BLAS レベル 1 のルーチンと関数](#)」に対するインターフェイスであり、圧縮形式で格納されたスパースベクトルに共通的なベクトル演算を実行する。

すべてのインデックス・パラメータ *indx* は C タイプで表記され  $[0..N-1]$  の範囲を取る。

### [ipps?axpyi](#)

```
void cblas_saxpyi(const int N, const float alpha,
const float *X, const int *indx, float *Y);
void cblas_daxpyi(const int N, const double alpha,
const double *X, const int *indx, double *Y);
void cblas_caxpyi(const int N, const void *alpha,
const void *X, const int *indx, void *Y);
void cblas_zaxpyi(const int N, const void *alpha,
const void *X, const int *indx, void *Y);
```

### [ipps?doti](#)

```
float cblas_sdoti(const int N, const float *X,
const int *indx, const float *Y);
double cblas_ddoti(const int N, const double *X,
const int *indx, const double *Y);
```

### [ipps?dotci](#)

```
void cblas_cdotci_sub(const int N, const void *X, const int *indx, const void
*Y, void *dotui);
void cblas_zdotci_sub(const int N, const void *X, const int *indx, const void
*Y, void *dotui);
```

### [ipps?dotui](#)

```
void cblas_cdotui_sub(const int N, const void *X, const int *indx, const void
*Y, void *dotui);
void cblas_zdotui_sub(const int N, const void *X, const int *indx, const void
*Y, void *dotui);
```

### [ipps?gthr](#)

```
void cblas_sgthr(const int N, const float *Y, float *X,
const int *indx);
void cblas_dgthr(const int N, const double *Y, double *X,
const int *indx);
void cblas_cgthr(const int N, const void *Y, void *X,
const int *indx);
```



```
void cblas_zgthr(const int N, const void *Y, void *X,  
const int *indx);
```

#### [ipps?gthrz](#)

```
void cblas_sgthrz(const int N, float *Y, float *X,  
const int *indx);  
void cblas_dgthrz(const int N, double *Y, double *X,  
const int *indx);  
void cblas_cgthrz(const int N, void *Y, void *X,  
const int *indx);  
void cblas_zgthrz(const int N, void *Y, void *X,  
const int *indx);
```

#### [ipps?roti](#)

```
void cblas_sroti(const int N, float *X, const int *indx,  
float *Y, const float c, const float s);  
void cblas_droti(const int N, double *X, const int *indx,  
double *Y, const double c, const double s);
```

#### [ipps?sctr](#)

```
void cblas_ssctr(const int N, const float *X, const int *indx, float *Y);  
void cblas_dsctr(const int N, const double *X, const int *indx, double *Y);  
void cblas_csctr(const int N, const void *X, const int *indx, void *Y);  
void cblas_zsctr(const int N, const void *X, const int *indx, void *Y);
```



# 用語集

---

$A^H$	一般行列 $A$ の共役を表す。 共役行列も参照。
$A^T$	一般行列 $A$ の転置を表す。 転置も参照。
帯行列	$ i - j  > l$ に対して $a_{ij} = 0$ となる $m \times n$ の一般行列 $A$ 。 ここで、 $1 < l < \min(m, n)$ 。例えば、三角行列は、いずれも帯行列である。
帯格納	帯行列に対する特殊な格納形式。 行列は、2次元配列に格納される。このとき、行列の各列は配列の対応する列に、行列の各対角成分は配列の各行に格納される。
BLAS	Basic Linear Algebra (基本線形代数) の略。これらのサブプログラムには、ベクトル演算、行列 - ベクトル演算、行列 - 行列演算が導入されている。
BRNG	Basic Random Number Generator (基本乱数生成器) の略。基本乱数生成器は、一様分布の i.i.d. 乱数列を模倣した擬似乱数生成器である。一様分布以外の分布は、一様分布の乱数列に異なる変換手法を用いて生成される。
BRNG 登録	ユーザ定義の BRNG を VSL に含め、定義済み VSL 基本生成器と共に使用することのできる標準化されたメカニズム。
Bunch-Kaufman 因子分解	実数称行列あるいは複素エルミート行列 $A$ を形式 $A = PUDU^H P^T$ (または $A = PLDL^H P^T$ ) で表したものの。 $P$ は置換行列、 $U$ と $L$ は単位対角成分を持つ上および下三角行列、 $D$ は $1 \times 1$ と $2 \times 2$ の対角ブロックを持つエルミート・ブロック対角行列である。 $U$ と $L$ は、 $D$ の $2 \times 2$ のブロックに対応する $2 \times 2$ の単位対角ブロックを持つ。

c	ルーチン名の最初の文字に使われている場合、c は単精度の複素数データ型の使用を示す。
CBLAS	BLAS に対する C インターフェイス。BLAS を参照。
CDF	累積分布関数。一変量または多変量のランダム変数 $X$ における確率分布を決定する関数。一変量分布において累積分布関数は、実引数 $x$ の関数であり、それぞれの $x$ は事象 $A: X \leq x$ の確率と等しい値をとる。多変量分布において累積分布関数は、実ベクトル $\mathbf{x} = (x_1, x_2, \dots, x_n)$ の関数であり、それぞれの $\mathbf{x}$ は事象 $A = (X_1 \leq x_1 \& X_2 \leq x_2, \& \dots, \& X_n \leq x_n)$ の確率と等しい値をとる。
コレスキー因子分解	対称 (複素データの場合はエルミート) 正定値行列 $A$ を、形式 $A = U^H U$ か $A = LL^H$ で表したものの。 $L$ は下三角行列、 $U$ は上三角行列である。
条件数	所定の正方行列 $A$ に対して、次のように定義される数 $\kappa(A): \kappa(A) = \ A\  \ A^{-1}\ $ 。
共役行列	所定の一般行列 $A$ に対して、次のように定義される行列 $A^H: (A^H)_{ij} = (a_{ji})^*$ 。
共役数	複素数 $z = a + bi$ の共役は、 $z^* = a - bi$ になる。
d	ルーチン名の最初の文字に使われている場合、d は倍精度の実数データ型を使用することを示す。
内積	$x \cdot y$ で表され、特定のベクトル $x$ と $y$ に対して次のように定義される数: $x \cdot y = \sum_i x_i y_i$ 。 $x_i$ と $y_i$ はそれぞれ、 $x$ と $y$ の $i$ 番目の成分を表す。
倍精度	浮動小数点データ型の 1 つ。インテル® プロセッサでは、このデータ型は $2.23 \cdot 10^{-308} <  x  < 1.79 \cdot 10^{308}$ の範囲にある実数 $x$ を格納できる。 このデータ型に対するマシン精度 $\epsilon$ は、およそ $10^{-15}$ になる。 つまり、倍精度の数は、通常は、15 以下の有効 10 進桁を格納できる。 詳細は、『Pentium® Processor Family Developer's Manual, Volume 3: Architecture and Programming Manual』(英語) を参照。
固有値	固有値問題を参照。
固有値問題	所定の正方行列 $A$ に対して、 $Ax = \lambda x$ となるような非ゼロのベクトル $x$ と値 $\lambda$ を求める問題。値 $\lambda$ は、行列 $A$ の固有値と呼ばれ、ベクトル $x$ は行列 $A$ の固有ベクトルと呼ばれる。
固有ベクトル	固有値問題を参照。

基本リフレクタ (Householder 行列)	一般形式 $H = I - \tau v v^T$ の行列。 $v$ は列ベクトル、 $\tau$ はスカラーである。 <b>LAPACK</b> では、基本リフレクタは、例えば行列 $Q$ を $QR$ 因子分解で表すために使用する (行列 $Q$ は、基本リフレクタの積として表す)。
因子分解	行列を行列同士の積として表すこと。 <b>Bunch-Kaufman</b> 因子分解、 <b>コレスキー</b> 因子分解、 <b>LU</b> 因子分解、 <b>LQ</b> 因子分解、 <b>QR</b> 因子分解、 <b>Schur</b> 因子分解も参照。
FFT	<b>Fast Fourier Transforms</b> (高速フーリエ変換) の略。本書の第 3 章を参照。
フル格納	あらゆる種類の行列の格納が可能な格納形式。行列 $A$ は、2 次元の配列 $a$ に格納される。このとき、行列の成分 $a_{ij}$ は、配列の成分 $a(i, j)$ に格納される。
エルミート行列	共役行列 $A^H$ に等しい正方行列 $A$ 。共役 $A^H$ は、次のように定義される: $(A^H)_{ij} = (a_{ji})^*$ 。
<b>I</b>	単位行列を参照。
単位行列	対角成分が 1、非対角成分が 0 である正方行列 $I$ 。任意の行列 $A$ に対し、 $AI = A$ と $IA = A$ 。
i.i.d.	<b>Independent Identically Distributed</b> (独立同一分布) の略。
インプレース	演算の修飾子。演算をインプレースで実行する関数では、その入力を配列から読み取り、その出力を同じ配列に返す。
インテル MKL	<b>Intel<sup>®</sup> Math Kernel Library</b> (インテル <sup>®</sup> マス・カーネル・ライブラリ) の略。
逆行列	$A^{-1}$ のように表し、所定の正方行列 $A$ に対して次のように定義される行列: $AA^{-1} = A^{-1}A = I$ 。 特異行列 $A$ に対しては、 $A^{-1}$ は存在しない。
<b>LQ</b> 因子分解	$m \times n$ の行列 $A$ を、 $A = LQ$ あるいは $A = (L \ 0) Q$ として表したものの。 $Q$ は、 $n \times n$ の直交 (ユニタリ) 行列である。 $m \leq n$ に対しては、 $L$ は実数の対角成分を持つ $m \times m$ の下三角行列である。 $m > n$ に対しては、次のようになる。 $L = \begin{bmatrix} L_1 \\ L_2 \end{bmatrix}$ $L_1$ は、 $n \times n$ の下三角行列、 $L_2$ は矩形行列である。

LU 因子分解	$m \times n$ の一般行列 $A$ を、 $A = PLU$ として表したもの。 $P$ は置換行列、 $L$ は単位対角成分を持つ下三角行列 ( $m > n$ の場合は、下台形行列)、 $U$ は上三角行列 ( $m < n$ の場合は上台形行列) である。
マシン精度	マシンによる実数表現の精度を決定する数 $\epsilon$ 。インテル® アーキテクチャでは、マシン精度は単精度のデータに対しては約 $10^{-7}$ に、倍精度のデータに対しては約 $10^{-15}$ になる。この精度は、マシンによる実数の表現における有効 10 進桁数も決定する。倍精度と単精度も参照。
MPI	Message Passing Interface (メッセージ・パッシング・インターフェイス) の略。並列計算処理でのさまざまなメッセージ通信機能におけるユーザ・インターフェイスと機能を標準化した規格。
MPICH	移植可能な無償の MPI の実装ライブラリ。
直交行列	転置とその逆が等しくなる、つまり $A^T = A^{-1}$ であるような実数の正方行列 $A$ 。したがって、 $AA^T = A^TA = I$ 。直交行列では、すべての固有値は絶対値 1 を持つ。
圧縮格納	対称行列、エルミート行列、三角行列をよりコンパクトに格納できる格納形式。行列の上三角または下三角が、列ごとに 1 次元配列に圧縮される。
PDF	<p>Probability Density Function (確率密度関数) の略。一変量または多変量の連続ランダム変数 <math>X</math> における確率分布を決定する関数。確率密度関数 <math>f(x)</math> は、累積分布関数 <math>F(x)</math> と密接な関係を持つ。一変量分布における関係を次に示す。</p> $F(x) = \int_{-\infty}^x f(t) dt$ <p>多変量分布における関係を次に示す。</p> $f(x_1, x_2, \dots, x_n) = \int_{-\infty}^{x_1} \int_{-\infty}^{x_2} \dots \int_{-\infty}^{x_n} f(t_1, t_2, \dots, t_n) dt_1 dt_2 \dots$
正定値行列	非ゼロの任意のベクトル $x$ に対して、 $Ax \cdot x > 0$ であるような正方行列 $A$ 。は、内積を表す。
擬似乱数生成器	真にランダムな数列を模倣する完全な決定論的アルゴリズム。

QR 因子分解	$m \times n$ の行列 $A$ を、 $A = QR$ として表したもの。 $Q$ は $m \times m$ の直交 (ユニタリ) 行列、 $R$ は実数の対角成分を持つ $n \times n$ の上三角行列 ( $m \geq n$ の場合) あるいは上台形行列 ( $m < n$ の場合) である。
ランダム・ストリーム	一様分布の独立同一分布乱数の抽象的なソース。本書におけるランダム・ストリームとは、指定されたランダム・ストリームに関連付けられた基本生成器によって生成される乱数列を、一意に定義する構造を指す。
RNG	Random Number Generator (乱数生成器) の略。本書における乱数生成器とは、真にランダムな数列を模倣する、完全な決定論的アルゴリズムに基づいた擬似乱数生成器を意味する。
s	ルーチン名の最初の文字に使われている場合、s は、単精度の実数データ型を使用することを示す。
ScaLAPACK	Scalable Linear Algebra PACKage の略。
Schur 因子分解	正方行列 $A$ を $A = ZTZ^H$ の形式で表したもの。 $T$ は、Schur 形式と呼ばれる $A$ の上疑似三角行列 (複素数 $A$ に対しては、三角行列) である。行列 $Z$ は、直交行列 (複素数 $A$ に対してはユニタリ行列) である。 $Z$ の列は、Schur ベクトルと呼ばれる。
単精度	浮動小数点データ型の 1 つ。インテル <sup>®</sup> プロセッサでは、このデータ型は $1.18 \times 10^{-38} <  x  < 3.40 \times 10^{38}$ の範囲にある実数 $x$ を格納できる。 このデータ型に対するマシン精度 $\varepsilon$ は、およそ $10^{-7}$ になる。つまり、単精度の数は、通常は、7 以下の有効 10 進桁を格納できる。詳細は、『 <i>Pentium<sup>®</sup> Processor Family Developer's Manual, Volume 3: Architecture and Programming Manual</i> 』(英語) を参照。
特異行列	行列式がゼロであるような行列。 $A$ が特異行列とすると、逆 $A^{-1}$ は存在せず、また連立方程式 $Ax = b$ は独自の解を持たない (つまり、解が存在しないか無限数の解が存在する)。
特異値	所定の一般行列 $A$ に対し、行列 $AA^H$ の固有値として定義される数。SVD も参照。
SMP	Symmetric MultiProcessing (対称型マルチプロセッシング) の略。MKL では、SMP によって提供される並列化によって性能を改善している。
スパース BLAS	スパースベクトルに対して基本的なベクトル演算を実行するルーチン。スパース BLAS ルーチンは、ベクトルが粗であるのを利用し、ベクトルの非ゼロの成分だけを格納できる。BLAS を参照。

スパースベクトル	成分の大半がゼロであるベクトル。
格納形式	行列を格納する方法。フル格納、圧縮格納、帯格納を参照。
SVD	Singular Value Decomposition (特異値分解) の略。第 5 章の特異値分解も参照。
対称行列	$a_{ij} = a_{ji}$ である正方行列 $A$ 。
転置	所定の行列 $A$ の転置は、 $(A^T)_{ij} = a_{ji}$ となる行列 $A^T$ になる ( $A$ の行が $A^T$ の列に、 $A$ の列が $A^T$ の行になる)。
台形行列	$A = (A_1 A_2)$ である行列 $A$ 。 $A_1$ は上三角行列、 $A_2$ は矩形行列である。
三角行列	劣対角成分 (優対角成分) すべてがゼロである場合、行列 $A$ は上 (下) 三角行列と呼ばれる。したがって、上三角行列に対しては $i > j$ の場合に $a_{ji} = 0$ に、下三角行列に対しては $i < j$ の場合に $a_{ji} = 0$ になる。
三重対角行列	非ゼロの成分が、3 つの対角成分 (主対角成分、最初の劣対角成分、最初の優対角成分) だけに存在する行列。
ユニタリ行列	共役とその逆が等しくなる複素数型の正方行列 $A$ 。つまり、 $A^H = A^{-1}$ となり、したがって $AA^H = A^H A = I$ となる。ユニタリ行列のすべての固有値は、絶対値 1 を持つ。
VML	Vector Mathematical Library (ベクトル数学ライブラリ) の略。本書の第 9 章を参照。
VSL	Vector Statistical Library (ベクトル統計ライブラリ) の略。本書の第 10 章を参照。
z	ルーチン名の最初の文字に使われている場合、z は倍精度の複素数データ型を使用することを示す。



# 参考文献

---

BLAS、スパース BLAS、LAPACK、ScaLAPACK、スパースソルバ、区間ソルバ、VML、VSL、および DFT の機能のさらに詳しい説明は、以下の資料を参照。

- **BLAS レベル 1**

C. Lawson、R. Hanson、D. Kincaid、F. Krough 著。『*Basic Linear Algebra Subprograms for Fortran Usage*』。ACM Transactions on Mathematical Software, Vol.5, No.3 (1979 年 9 月) 308-325。

- **BLAS レベル 2**

J. Dongarra、J. Du Croz、S. Hammarling、R. Hanson 著。『*An Extended Set of Fortran Basic Linear Algebra Subprograms*』。ACM Transactions on Mathematical Software, Vol.14, No.1 (1988 年 3 月) 1-32。

- **BLAS レベル 3**

J. Dongarra、J. DuCroz、I. Duff、S. Hammarling 著。『*A Set of Level 3 Basic Linear Algebra Subprograms*』。ACM Transactions on Mathematical Software (1989 年 12 月)。

- **スパース BLAS**

D. Dodson、R. Grimes、J. Lewis 著。『*Sparse Extensions to the FORTRAN Basic Linear Algebra Subprograms*』。ACM Transactions on Math Software, Vol.17, No.2 (1991 年 6 月)。

D. Dodson、R. Grimes、J. Lewis 著。『*Algorithm 692: Model Implementation and Test Package for the Sparse Basic Linear Algebra Subprograms*』。ACM Transactions on Mathematical Software, Vol.17, No.2 (1991 年 6 月)。

[Duff86] I.S.Duff、A.M.Erisman、J.K.Reid 著。『*Direct Methods for Sparse Matrices*』。Clarendon Press, Oxford, UK、1986 年。

[CXML01] 『*Compaq Extended Math Library Reference Guide*』。2001 年 10 月。

[Rem05] K.Remington 著。『*A NIST FORTRAN Sparse Blas User's Guide*』。  
<http://math.nist.gov/~KRemington/fspblas/> を参照のこと。

[Saad94] Y.Saad 著。『SPARSKIT: A Basic Tool-kit for Sparse Matrix Computation』Version 2、1994 年。 <http://www.cs.umn.edu/~saad> を参照のこと。

[Saad96] Y.Saad 著。『Iterative Methods for Linear Systems』。 PWS Publishing, Boston、1996 年。

• **LAPACK**

[LUG] E. Anderson、Z. Bai、C. Bischof、S. Blackford、J. Demmel、J. Dongarra、J. Du Croz、A. Greenbaum、S. Hammarling、A. McKenney、D. Sorensen 著。『LAPACK Users' Guide』Third Edition。 Society for Industrial and Applied Mathematics (SIAM)、1999 年。

[Golub96] G. Golub、C. Van Loan 著。『Matrix Computations』。 Johns Hopkins University Press, Baltimore, third edition、1996 年。

• **ScaLAPACK**

[SLUG] L. Blackford、J. Choi、A. Cleary、E. D'Azevedo、J. Demmel、I. Dhillon、J. Dongarra、S. Hammarling、G. Henry、A. Petitet、K. Stanley、D. Walker、R. Whaley 著。『ScaLAPACK Users' Guide』。 Society for Industrial and Applied Mathematics (SIAM)、1997 年。

• **スパースソルバ**

[Duff99] I. S. Duff、J. Koster 著。『The design and use of algorithms for permuting large entries to the diagonal of sparse matrices』。 SIAM J. Matrix Analysis and Applications, 20(4):889–901、1999 年。

[Dong95] J. Dongarra、V. Eijkhout、A. Kalhan 著。『Reverse Communication Interface for Linear Algebra Templates for Iterative Methods』。 UT-CS-95-291、1995 年 5 月。  
<http://www.netlib.org/lapack/lawnspdf/lawn99.pdf> を参照のこと。

[Karypis98] G. Karypis、V. Kumar 著。『A fast and high quality multilevel scheme for partitioning irregular graphs』。 SIAM Journal on Scientific Computing, 20(1):359–392、1998 年。

[Li99] X.S. Li、J.W. Demmel 著。『A scalable sparse direct solver using static pivoting』。 9th SIAM conference on Parallel Processing for Scientific Computing 発表論文。 San Antonio, Texas、1999 年 3 月 22-24 日。

[Liu85] J.W.H. Liu 著。『Modification of the Minimum-Degree algorithm by multiple elimination』。 ACM Transactions on Mathematical Software, 11(2):141–153、1985 年。

- [Menon98] R. Menon L. Dagnum 著。『*OpenMP: An Industry-Standard API for Shared-Memory Programming*』。IEEE Computational Science & Engineering, 1:46–55、1998 年。 <http://www.openmp.org> を参照のこと。
- [Schenk00] O. Schenk 著。『*Scalable Parallel Sparse LU Factorization Methods on Shared Memory Multiprocessors*』。博士論文。ETH Zurich、2000 年。
- [Schenk00-2] O. Schenk、K. Gartner、W. Fichtner 著。『*Efficient sparse LU factorization with left-right looking strategy on shared memory multiprocessors*』。BIT, 40(1): 158–176、2000 年。
- [Schenk01] O. Schenk、K. Gartner 著。『*Sparse factorization with two-level scheduling in PARDISO*』。10th SIAM conference on Parallel Processing for Scientific Computing 発表論文。Portsmouth, Virginia、2001 年 3 月 12-14 日。
- [Schenk02] O. Schenk、K. Gartner 著。『*Two-level scheduling in PARDISO: Improved scalability on shared memory multiprocessing systems*』。Parallel Computing, 28:187–197、2002 年。
- [Schenk03] O. Schenk、K. Gartner 著。『*Solving unsymmetric sparse systems of linear equations with PARDISO*』。Future Generation Computer Systems、2003 年。
- [Sonn89] P. Sonneveld 著。『*CGS, a fast Lanczos-type solver for nonsymmetric linear systems*』。SIAM Journal on Scientific and Statistical Computing, 10: 36–52、1989 年。
- [Young71] D.M.Young 著。『*Iterative Solution of Large Linear Systems*』。New York, Academic Press, Inc.、1971 年。

- VSL

- [VSL Notes] インテル® MKL 製品に同梱されるドキュメント (ファイル名: `vslnotes.pdf`)。
- [Bratley87] Bratley P.、Fox B.L.、Schrage L.E. 著。『*A Guide to Simulation*』 2nd edition。Springer-Verlag, New York、1987 年。
- [Bratley88] Bratley P.、Fox B.L. 著。『*Implementing Sobol's Quasirandom Sequence Generator*』。ACM Transactions on Mathematical Software, Vol. 14, No. 1, Pages 88–100、1988 年 3 月。

- [Bratley92] Bratley P., Fox B.L., Niederreiter H. 著。『*Implementation and Tests of Low-Discrepancy Sequences*』。ACM Transactions on Modeling and Computer Simulation, Vol. 2, No. 3, Pages 195-213、1992 年 7 月。
- [Coddington94] Coddington, P. D. 著。『*Analysis of Random Number Generators Using Monte Carlo Simulation*』。Int. J. Mod. Phys. C-5, 547、1994 年。
- [Gentle98] Gentle, James E. 著。『*Random Number Generation and Monte Carlo Methods*』。Springer-Verlag New York, Inc., 1998 年。
- [L'Ecuyer94] L'Ecuyer, Pierre 著。『*Uniform Random Number Generation*』。Annals of Operations Research, 53, 77-120、1994 年。
- [L'Ecuyer99] L'Ecuyer, Pierre 著。『*Tables of Linear Congruential Generators of Different Sizes and Good Lattice Structure*』。Mathematics of Computation, 68, 225, 249-260、1999 年。
- [L'Ecuyer99a] L'Ecuyer, Pierre 著。『*Good Parameter Sets for Combined Multiple Recursive Random Number Generators*』。Operations Research, 47, 1, 159-164、1999 年。
- [L'Ecuyer01] L'Ecuyer, Pierre 著。『*Software for Uniform Random Number Generation: Distinguishing the Good and the Bad*』。2001 Winter Simulation Conference 発表論文。IEEE Press, 95-105、2001 年 12 月。
- [Kirkpatrick81] Kirkpatrick, S., Stoll, E. 著。『*A Very Fast Shift-Register Sequence Random Number Generator*』。Journal of Computational Physics, V. 40.517-526、1981 年。
- [Knuth81] Knuth, Donald E. 著。『*The Art of Computer Programming, Volume 2, Seminumerical Algorithms*』 2nd edition。Addison-Wesley Publishing Company, Reading, Massachusetts、1981 年。
- [Matsumoto98] Matsumoto, M., Nishimura, T. 著。『*Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator*』。ACM Transactions on Modeling and Computer Simulation, Vol. 8, No. 1。Pages 3-30、1998 年 1 月。
- [Matsumoto2000] Matsumoto, M., Nishimura, T. 著。『*Dynamic Creation of Pseudorandom Number Generators*』。56-69: Monte Carlo and Quasi-Monte Carlo Methods 1998 に収録。Ed. Niederreiter, H., Spanier, J. 著。Springer 2000。 <http://www.math.sci.hiroshima-u.ac.jp/%7Emat/MT/DC/dc.html> を参照のこと。
- [NAG] NAG Numerical Libraries。  
[http://www.nag.co.uk/numeric/numerical\\_libraries.asp](http://www.nag.co.uk/numeric/numerical_libraries.asp) を参照のこと。

- [Sobol76] Sobol, I.M., Levitan, Yu.L. 著。『*The production of points uniformly distributed in a multidimensional cube*』。Preprint 40, Institute of Applied Mathematics, USSR Academy of Sciences、1976 年 ( ロシア )。

- DFT

- [1] E. Oran Brigham 著。『*The Fast Fourier Transform and Its Applications*』。Prentice Hall, New Jersey、1988 年。
- [2] Athanasios Papoulis 著。『*The Fourier Integral and its Applications*』2nd edition。McGraw-Hill, New York、1984 年。
- [3] Ping Tak Peter Tang 著。『*DFTI, a New API for DFT: Motivation, Design, and Rationale*』。2002 年 7 月。
- [4] Charles Van Loan 著。『*Computational Frameworks for the Fast Fourier Transform*』。SIAM, Philadelphia、1992 年。

- VML

J.M.Muller 著。『*Elementary functions: algorithms and implementation*』。Birkhauser Boston、1997 年。

『IEEE Standard for Binary Floating-Point Arithmetic. ANSI/IEEE Std 754-1985』。

- 区間ソルバ

- [Alefeld83] G. Alefeld, J. Herzberger 著。『*Introduction to Interval Computations*』。Academic Press, New York、1983 年。
- [Bentbib02] A.H.Bentbib 著。『*Solving the full rank interval least squares problem*』// Applied Numerical Mathematics。2002 年。Vol. 41.– P. 283–294。
- [Blik92] Ch. Blik 著。『*Computer methods for design automation*』。博士論文。Dept. of Ocean Engineering, Massachusetts Institute of Technology、1992 年。
- [Hammer95] R. Hammer, M. Hocks, U. Kulisch, D. Ratz 著。『*C++ Toolbox for Verified Computing I. Basic Numerical Problems*』。Berlin-Heidelberg: Springer、1995 年。
- [Hansen92] E. Hansen 著。『*Bounding the solution of interval linear equations*』// SIAM Journal on Numerical Analysis。1992 年。Vol. 29, No. 5.– P. 1493–1503。

- [Herzberger94] J. Herzberger 著。『Iterative methods for the inclusion of the inverse of a matrix』// *Topics in Validated Computations*. J. Herzberger 編集 – Amsterdam: Elsevier、1994 年。
- [Jansson91] Ch.Jansson 著。『Interval linear systems with symmetric matrices, skew-symmetric matrices, and dependencies in the right hand side』// *Computing*. 1991 年。Vol. 46.– P. 265 – 274。
- [Kearfott96] R.B. Kearfott 著。『*Rigorous Global Search: Continuous Problems*』。Dordrecht, Kluwer、1996 年。
- [Kearfott] R.B. Kearfott、M.T. Nakao、A. Neumaier、S.M. Rump、S.P. Shary、P. van Hentenryck 著。『Standardized notation in interval analysis』。  
<http://www.mat.univie.ac.at/~neum/software/int/> を参照のこと。
- [Kreinovich97] V. Kreinovich、A. Lakeyev、J. Rohn、P. Kahl 著。『*Computational Complexity and Feasibility of Data Processing and Interval Computations*』。Kluwer, Dordrecht、1997 年。
- [Neumaier90] A. Neumaier 著。『*Interval Methods for Systems of Equations*』。Cambridge, Cambridge University Press、1990 年。
- [Neumaier99] A.Neumaier 著。『A simple derivation of Hansen-Bliek-Rohn-Ning-Kearfott enclosure for linear interval equations』// *Reliable Computing*. 1999 年。Vol. 5, No. 2.– P. 131–136。
- [Ning97] S. Ning、R.B. Kearfott 著。『A comparison of some methods for solving linear interval equations』// *SIAM Journal on Numerical Analysis*. 1997 年。Vol. 34, No. 4.– P. 1289–1305。
- [Rex99] G.Rex、J.Rohn 著。『Sufficient conditions for regularity and singularity of interval matrices』// *SIAM Journal on Numerical Analysis*. 1999 年。Vol. 20.– P. 437–445。
- [Rohn93] J. Rohn 著。『Cheap and tight bounds: the recent result by E. Hansen can be made more efficient』// *Interval Computations*. 1993 年。No. 4.– P. 13–21。
- [Rump83] S. M.Rump 著。『Solving algebraic problems with high accuracy』// *A New Approach to Scientific Computation*; Kulisch U. W. and Miranker W. L., eds. – New York: Academic Press. 1983 年。P. 51–120。
- [Rump84] S. M.Rump 著。『Solution of linear and nonlinear algebraic problems with sharp guaranteed bounds』// *Computing Supplement*. 1984 年。Vol. 5.– P. 147–168。

- [Rump80] S. M.Rump、Kaucher E. 著。『Small bounds for the solution of systems of linear equations』// *Computing Supplement*。1980 年。Vol. 2.– P. 157–164。
- [Rump] S. Rump 著。『INTLAB — INTerval LABoratory』。21 p。  
<http://www.ti3.tu-harburg.de/~rump/intlab/> を参照のこと。
- [Shary92] S.P. Shary 著。『A new class of algorithms for optimal solution of interval linear systems』// *Interval Computations*。1992 年。No. 2(4). – P. 18–29。
- [Shary95] S.P. Shary 著。『On optimal solution of interval linear equations』// *SIAM Journal on Numerical Analysis*。1995 年。Vol. 32, No. 2.– P. 610–630。
- [Shary02] S. P. Shary 著。『A new technique in systems analysis under interval uncertainty and ambiguity』// *Reliable Computing*。2002 年。Vol. 8.– 321–419。
- [Shary] S.P. Shary 著。『A new class of methods for optimal enclosing solution sets to interval linear systems』// *Journal of Computational Mathematics*。出版予定。

BLAS、スパース BLAS、LAPACK、および ScaLAPACK パッケージ (プラットフォーム固有の最適化なし) を参考用に導入するには、[www.netlib.org](http://www.netlib.org) を参照のこと。





# 索引

---

## シンボル

?asum 2-6  
?axpy 2-7  
?axpyi 2-128  
?bdsdc 4-104  
?bdsqr 4-100  
?combamax1 7-8  
?ConvExec 10-122  
?ConvExec1D 10-124  
?ConvExecX 10-126  
?ConvExecX1D 10-128  
?ConvNewTask 10-104  
?ConvNewTask1D 10-106  
?ConvNewTaskX 10-108  
?ConvNewTaskX1D 10-111  
?copy 2-9  
?CorrExec 10-122  
?CorrExec1D 10-124  
?CorrExecX 10-126  
?CorrExecX1D 10-128  
?CorrNewTask 10-104  
?CorrNewTask1D 10-106  
?CorrNewTaskX 10-108  
?CorrNewTaskX1D 10-111  
?dbtf2 7-179  
?dbtrf 7-181  
?disna 4-169  
?dot 2-10  
?dotc 2-13  
?dotci 2-131  
?doti 2-130  
?dotu 2-15

?dotui 2-133  
?dttrf 7-183  
?dttrsv 7-184  
?fft1d 11-71, 11-75, 11-79  
?fft1dc 11-72, 11-76, 11-81  
?fft2d 11-84, 11-87, 11-94  
?fft2dc 11-85, 11-90, 11-95  
?gbbrd 4-83  
?gbcon 3-73  
?gbequ 3-168  
?gbmv 2-32  
?gbrfs 3-107  
?gbsv 3-186  
?gbsvx 3-189  
?gbtf2 5-24  
?gbtrf 3-11  
?gbtrs 3-37  
?gebak 4-210  
?gebal 4-207  
?gebd2 5-25  
?gebrd 4-80  
?gecon 3-71  
?geequ 3-165  
?gees 4-404  
?geesx 4-408  
?geev 4-414  
?geevx 4-419  
?gegas 12-5  
?gegss 12-10  
?gehbs 12-12  
?gehd2 5-28  
?gehrd 4-193  
?gehss 12-7

?gekws 12-8	?ggrqf 4-74
?gelq2 5-30	?ggsvd 4-434
?gelqf 4-26	?ggsvp 4-283
?gels 4-295	?gtcon 3-76
?gelsd 4-305	?gthr 2-134
?gelss 4-302	?gthrz 2-136
?gelsy 4-298	?gtrfs 3-110
?gemip 12-23	?gtsv 3-196
?gemm 2-95	?gtsvx 3-198
?gemv 2-35	?gttrf 3-13
?gepps 12-13	?gttrs 3-40
?geql2 5-32	?gtts2 5-41
?geqlf 4-39	?hbev 4-370
?geqp3 4-13	?hbevd 4-376
?geqpf 4-10	?hbevz 4-384
?geqr2 5-33	?hbgst 4-184
?geqrf 4-7	?hbgv 4-491
?ger 2-38	?hbgvd 4-497
?gerbr 12-19	?hbgvx 4-506
?gerc 2-40	?hbtrd 4-141
?gerfs 3-104	?hecon 3-90
?gerq2 5-35	?heev 4-319
?gerqf 4-51	?heevd 4-324
?geru 2-42	?heevr 4-342
?gesc2 5-37	?heevx 4-332
?gesdd 4-430	?heft2 5-306
?gesv 3-178	?hegst 4-175
?gesvd 4-426	?hegv 4-444
?gesvr 12-20	?hegvd 4-451
?gesvx 3-180	?hegvx 4-460
?geszi 12-18	?hemm 2-99
?getc2 5-38	?hemv 2-47
?getf2 5-40	?her 2-49
?getrf 3-9	?her2 2-51
?getri 3-146	?her2k 2-105
?getrs 3-35	?herfs 3-128
?ggbak 4-250	?herk 2-102
?ggbal 4-247	?hesv 3-238
?gges 4-511	?hesvx 3-241
?ggesx 4-518	?hetrd 4-119
?ggev 4-525	?hetrf 3-26
?ggevx 4-530	?hetri 3-155
?ggglm 4-313	?hetrs 3-55
?gghrd 4-244	?hgeqz 4-252
?gglse 4-310	?hpcon 3-94
?ggqrf 4-71	?hpev 4-350

---

?hpevd 4-355	?lagv2 5-94
?hpevx 4-363	?lahef 5-255
?hpgst 4-179	?lahqr 5-96
?hpgv 4-469	?lahrd 5-99
?hpgvd 4-475	?laic1 5-101
?hpgvx 4-484	?laln2 5-104
?hpmv 2-54	?lals0 5-107
?hpr 2-56	?lalsa 5-111
?hpr2 2-58	?lalsd 5-114
?hprfs 3-134	?lamc1 5-323
?hpsvx 3-255	?lamc2 5-324
?hptrd 4-133	?lamc3 5-325
?hptrf 3-32	?lamc4 5-325
?hptri 3-159	?lamc5 5-326
?hptrs 3-60	?lamch 5-322
?hsein 4-217	?lamrg 5-117
?hseqr 4-212	?lamsh 7-171
?labrd 5-42	?langb 5-118
?lacgv 5-10	?lange 5-119
?lacon 5-45	?langt 5-121
?lacy 5-47	?lanhb 5-125
?lacrm 5-10	?lanhe 5-133
?lacrt 5-12	?lanhp 5-129
?ladiv 5-48	?lanhs 5-122
?lae2 5-49	?lansb 5-124
?laebz 5-50	?lansp 5-127
?laed0 5-55	?lanst/?lanht 5-130
?laed1 5-57	?lansy 5-132
?laed2 5-59	?lantb 5-135
?laed3 5-62	?lantp 5-137
?laed4 5-64	?lantr 5-138
?laed5 5-66	?lanv2 5-140
?laed6 5-67	?lapll 5-141
?laed7 5-68	?lapmt 5-142
?laed8 5-72	?lapy2 5-144
?laed9 5-75	?lapy3 5-144
?laeda 5-77	?laqgb 5-145
?laein 5-79	?laqge 5-147
?laesy 5-13	?laqp2 5-149
?laev2 5-82	?laqps 5-151
?laexc 5-84	?laqsb 5-153
?lag2 5-85	?laqsp 5-155
?lags2 5-87	?laqsy 5-156
?lagtf 5-89	?laqtr 5-158
?lagtm 5-91	?lar1v 5-161
?lagts 5-92	?lar2v 5-163

?laref 7-172	?laswp 5-249
?larf 5-164	?lasy2 5-250
?larfb 5-166	?lasyf 5-252
?larfg 5-168	?latbs 5-257
?larft 5-170	?latdf 5-260
?larfx 5-173	?latps 5-262
?largv 5-174	?latrd 5-264
?larnv 5-176	?latrs 5-267
?larrb 5-177	?latrz 5-271
?larre 5-179	?lauu2 5-273
?larrrf 5-181	?lauum 5-275
?larrrv 5-183	?nrm2 2-16
?lartg 5-185	?opgtr 4-128
?lartv 5-187	?opmtr 4-130
?laruv 5-188	?org2l/?ung2l 5-276
?larz 5-189	?org2r/?ung2r 5-278
?larzb 5-191	?orgbr 4-87
?larzt 5-193	?orghr 4-195
?las2 5-197	?orgl2/?ungl2 5-279
?lascl 5-198	?orglq 4-29
?lasd0 5-200	?orgql 4-41
?lasd1 5-201	?orgqr 4-16
?lasd2 5-204	?org2/?ungr2 5-281
?lasd3 5-208	?orgrq 4-53
?lasd4 5-211	?orgtr 4-114
?lasd5 5-212	?orm2l/?unm2l 5-282
?lasd6 5-214	?orm2r/?unm2r 5-285
?lasd7 5-218	?ormbr 4-90
?lasd8 5-222	?ormhr 4-198
?lasd9 5-224	?orml2/?unml2 5-287
?lasda 5-226	?ormlq 4-31
?lasdq 5-229	?ormql 4-45
?lasdt 5-232	?ormqr 4-18
?laset 5-233	?ormr2/?unmr2 5-290
?lasorte 7-175	?ormr3/?unmr3 5-292
?lasq1 5-234	?ormrq 4-57
?lasq2 5-235	?ormrz 4-65
?lasq3 5-237	?ormtr 4-116
?lasq4 5-239	?pbcon 3-83
?lasq5 5-240	?pbequ 3-175
?lasq6 5-242	?pbrfs 3-119
?lasr 5-243	?pbstf 4-187
?lasrt 5-245	?pbsv 3-217
?lasrt2 7-176	?pbsvx 3-220
?lassq 5-246	?ptbf2 5-295
?lasv2 5-247	?pbtrf 3-20

?pbtrs 3-48  
?pocon 3-78  
?poequ 3-170  
?porfs 3-113  
?posv 3-203  
?posvx 3-205  
?potf2 5-296  
?potrf 3-15  
?potri 3-149  
?potrs 3-43  
?ppecon 3-81  
?ppequ 3-172  
?pprfs 3-116  
?ppsv 3-210  
?ppsvx 3-212  
?pptrf 3-18  
?pptri 3-151  
?pptrs 3-45  
?ptcon 3-85  
?pteqr 4-159  
?ptrfs 3-122  
?ptsv 3-225  
?ptsvx 3-227  
?pttrf 3-22  
?pttrs 3-51  
?pttrsv 7-186  
?ptts2 5-298  
?rot 2-17  
?rot ( 複素数 ) 5-14  
?rotg 2-19  
?roti 2-137  
?rotm 2-21  
?rotmg 2-23  
?rscl 5-299  
?sbev 4-367  
?sbevd 4-372  
?sbevx 4-380  
?sbgst 4-182  
?sbgv 4-489  
?sbgvd 4-494  
?sbgvx 4-501  
?sbmv 2-61  
?sbtrd 4-139  
?scal 2-25  
?sctr 2-139  
?sdot 2-12  
?spcon 3-92  
?spev 4-348  
?spevd 4-352  
?spevx 4-359  
?spgst 4-177  
?spgv 4-466  
?spgvd 4-472  
?spgvx 4-479  
?spm 2-64, 5-15  
?spr 2-66, 5-17  
?spr2 2-68  
?sprfs 3-131  
?spsv 3-245  
?spsvx 3-248  
?sptrd 4-126  
?sptrf 3-29  
?sptri 3-157  
?sptrs 3-58  
?stebz 4-163  
?stedc 4-150  
?stegr 4-154  
?stein 4-167  
?stein2 7-177  
?steqr 4-146  
?steqr2 7-188  
?sterf 4-144  
?stev 4-389  
?stevd 4-391  
?stevr 4-398  
?stevx 4-394  
?sum1 5-23  
?swap 2-26  
?sycon 3-87  
?syev 4-316  
?syevd 4-321  
?syevr 4-337  
?syevx 4-328  
?sygs2/?hegs2 5-300  
?sygst 4-173  
?sygv 4-441  
?sygvd 4-447  
?sygvx 4-455  
?symm 2-108  
?symv 2-70  
?symv ( 複素数 ) 5-19  
?syr 2-73

?syr ( 複素数 ) 5-21  
 ?syr2 2-75  
 ?syr2k 2-115  
 ?syrrs 3-125  
 ?syrrk 2-111  
 ?sysv 3-230  
 ?sysvx 3-233  
 ?sytd2/?hetd2 5-302  
 ?sytf2 5-304  
 ?sytrd 4-112  
 ?sytrf 3-23  
 ?sytri 3-153  
 ?sytrs 3-53  
 ?tbcon 3-101  
 ?tbmv 2-77  
 ?tbsv 2-80  
 ?tbtrs 3-68  
 ?tgevc 4-259  
 ?tgex2 5-308  
 ?tgexc 4-264  
 ?tgsen 4-267  
 ?tgsja 4-287  
 ?tgsna 4-278  
 ?tgsy2 5-310  
 ?tgsyl 4-273  
 ?tpcon 3-99  
 ?tpmv 2-83  
 ?tprfs 3-140  
 ?tpsv 2-86  
 ?tptri 3-163  
 ?tprts 3-65  
 ?trcon 3-96  
 ?trevc 4-222  
 ?trexc 4-232  
 ?trmm 2-119  
 ?trmv 2-88  
 ?trrfs 3-137  
 ?trsen 4-235  
 ?trsm 2-122  
 ?trsna 4-227  
 ?trsv 2-91  
 ?trsyl 4-240  
 ?trti2 5-314  
 ?trtri 12-16  
 ?trtri (LAPACK) 3-161  
 ?trtrs 12-3

?trtrs (LAPACK) 3-63  
 ?tzrnf 4-62  
 ?ungbr 4-93  
 ?unghr 4-201  
 ?unglq 4-34  
 ?ungql 4-43  
 ?ungqr 4-21  
 ?ungrq 4-55  
 ?ungtr 4-122  
 ?unmbr 4-97  
 ?unmhr 4-204  
 ?unmlq 4-36  
 ?unmql 4-48  
 ?unmqr 4-23  
 ?unmrq 4-60  
 ?unmrz 4-68  
 ?unmtr 4-124  
 ?upgtr 4-135  
 ?upmtr 4-137

## 数字

1 次元 FFT 11-68  
     格納効果 11-34, 11-36, 11-74, 11-78  
     グループ 11-69  
     複素シーケンス 11-76, 11-77, 11-80, 11-81  
     複素数から複素数 11-70  
     複素ベクトルの順方向または逆方向 FFT の計  
         算 11-71, 11-72  
 1 次方程式の解の誤差  
     一般行列 3-104, 6-52  
         帯格納 3-107  
     エルミート行列 3-128  
         圧縮格納 3-134  
     エルミート正定値行列 3-113, 3-122, 6-56  
         圧縮格納 3-116  
         帯格納 3-119  
     三角行列 3-137  
         圧縮格納 3-140  
         帯格納 3-143  
     三重対角行列 3-110  
     対称行列 3-125  
         圧縮格納 3-131  
     対称正定値行列 3-113, 3-122, 6-56  
         圧縮格納 3-116  
         帯格納 3-119

- 分散三重対角係数行列 6-60
- 1 次方程式の解の算出。1 次方程式を参照
- 1 次方程式の解の精度の改善
  - 一般行列 3-104, 6-52
  - エルミート行列 3-128
    - 圧縮格納 3-134
  - エルミート正定値行列 3-113, 3-122
    - 圧縮格納 3-116
    - 帯格納 3-119
  - 帯行列 3-107
  - 三重対角行列 3-110
  - 対称 / エルミート正定値分散行列 6-56
  - 対称行列 3-125
    - 圧縮格納 3-131
  - 対称正定値行列 3-113, 3-122
    - 圧縮格納 3-116
    - 帯格納 3-119
- 1 次方程式の誤差推定
  - 分散三重対角係数行列 6-60
- 1 次方程式、解の算出
  - エルミート正定値三重対角 1 次方程式 7-186
  - 三角行列
    - 帯格納 7-152
  - 三重対角行列
    - ScaLAPACK 補助 7-184
  - 対称正定値三重対角 1 次方程式 7-186
- 1 次方程式、算出 5-104, 5-158
  - 一般帯行列
    - ScaLAPACK 6-225
  - 一般行列 3-35, 6-22
    - 帯格納 3-37, 6-24
  - 一般三重対角行列
    - ScaLAPACK 6-228
  - エルミート行列 3-55
    - 圧縮格納 3-60, 3-252, 3-255
    - 誤差範囲 3-241, 3-255
  - エルミート正定値行列 3-43, 6-27
    - LAPACK 3-203, 3-205
    - ScaLAPACK 6-233
    - 圧縮格納 3-45, 3-210, 3-212
    - 帯格納 3-48, 3-220, 6-29
      - LAPACK 3-217
      - ScaLAPACK 6-240
    - 誤差範囲 3-212, 3-220
      - LAPACK 3-205
      - ScaLAPACK 6-233
- エルミート正定値三重対角行列 3-227, 6-31
  - LAPACK 3-225
  - ScaLAPACK 6-242
  - 誤差範囲 3-227
- 帯行列
  - LAPACK 3-186, 3-189
  - ScaLAPACK 6-222
- 過大評価問題または過小評価問題 6-245
- コレスキー因子分解された行列
  - LAPACK 3-48
  - ScaLAPACK 6-29
- 三角行列 3-63, 6-40
  - 圧縮格納 3-65
  - 帯格納 3-68
- 三重対角行列
  - LAPACK 3-40, 3-51, 3-196, 3-198
  - LAPACK 補助 5-161
  - 誤差範囲 3-198
- 正方行列
  - LAPACK 3-178, 3-180
  - ScaLAPACK 6-214, 6-216
  - 誤差範囲
    - LAPACK 3-180, 3-189
    - ScaLAPACK 6-216
- 対角優位行列
  - 帯行列 6-37
  - 三重対角行列 6-34
- 対称行列 3-53
  - 圧縮格納 3-58, 3-245, 3-248
  - 誤差範囲 3-233, 3-248
- 対称正定値行列 3-43, 6-27, 6-31
  - LAPACK 3-203, 3-205
  - ScaLAPACK 6-231, 6-233
  - 圧縮格納 3-45, 3-210, 3-212
  - 帯格納 3-48, 3-220, 6-29
    - LAPACK 3-217
    - ScaLAPACK 6-240
  - 誤差範囲 3-212, 3-220
    - LAPACK 3-205
    - ScaLAPACK 6-233
- 対称正定値三重対角行列 3-227
  - LAPACK 3-225
  - ScaLAPACK 6-242
  - 誤差範囲 3-227
- 複数の右辺
  - エルミート行列 3-238, 3-252

エルミート正定値行列 3-203, 3-210  
 帯格納 3-217  
 帯行列  
     LAPACK 3-186, 3-189  
     ScaLAPACK 6-222  
 三重対角行列 3-196, 3-198  
 正方行列  
     LAPACK 3-178, 3-180  
     ScaLAPACK 6-214, 6-216  
 対称行列 3-230, 3-245  
 対称正定値行列 3-203, 3-210  
     帯格納 3-217

1 次元 FFT  
     逆方向 FFT の実行、複素数入力データ 11-79, 11-81  
     順方向 FFT の計算、実数入力データ 11-75, 11-76

1- ノルムの値  
     一般矩形行列 7-54  
     一般三重対角行列 5-121  
     一般正方行列 5-119  
     上 Hessenberg 行列 5-122, 7-56  
     エルミート帯行列 5-125  
     三角帯行列 5-135  
     三角行列 5-138, 7-61  
         圧縮格納 5-137  
     実対称行列 5-132, 7-58  
     実対称三重対角行列 5-130  
     台形行列 5-138  
     対称帯行列 5-124  
     対称行列  
         圧縮格納 5-127  
     複素エルミート行列 5-133, 7-58  
         圧縮格納 5-129  
     複素エルミート三重対角行列 5-130  
     複素対称行列 5-132

2 次元 FFT 11-82  
     グループ 11-83  
     順方向または逆方向 FFT の計算 11-84, 11-85  
     データ構造の要件 11-83  
     方程式 11-84  
     データ格納の型 11-83

2 つの行列  
     QR 因子分解  
         LAPACK 4-71  
         ScaLAPACK 6-145

2 次元 FFT  
     逆方向 FFT の計算、複素数入力データ 11-94, 11-95  
     順方向 FFT の計算、実数入力データ 11-87, 11-90

## B

Bernoulli 10-75  
 Beta 10-68  
 Binomial 10-79  
 BLACS 6-1  
 BLAS ルーチン  
     行列の引数 B-4  
     ベクトルの引数 B-1  
     ルーチングループ 1-7, 2-1

BLAS レベル 1 の関数  
     ?asum 2-5, 2-6  
     ?dot 2-5, 2-10  
     ?dotc 2-5, 2-13  
     ?dotu 2-5, 2-15  
     ?nrm2 2-5, 2-16  
     ?sdot 2-5, 2-12  
     dcabs1 2-5, 2-30  
     i?amax 2-5, 2-28  
     i?amin 2-5, 2-29  
     コード例 C-1

BLAS レベル 1 のルーチン  
     ?axpy 2-5, 2-7  
     ?copy 2-5, 2-9  
     ?rot 2-5, 2-17  
     ?rotg 2-5, 2-19  
     ?rotm 2-5, 2-21  
     ?rotmg 2-23  
     ?rotmq 2-5  
     ?scal 2-5, 2-25  
     ?swap 2-5, 2-26  
     コード例 C-2

BLAS レベル 2 のルーチン  
     ?gbmv 2-31, 2-32  
     ?gemv 2-31, 2-35  
     ?ger 2-31, 2-38  
     ?gerc 2-31, 2-40  
     ?geru 2-31, 2-42  
     ?hbmV 2-31, 2-44  
     ?hemv 2-31, 2-47



- ?her 2-31, 2-49
- ?her2 2-31, 2-51
- ?hpmv 2-31, 2-54
- ?hpr 2-31, 2-56
- ?hpr2 2-31, 2-58
- ?sbmv 2-31, 2-61
- ?spmv 2-31, 2-64
- ?spr 2-31, 2-66
- ?spr2 2-31, 2-68
- ?symv 2-31, 2-70
- ?syr 2-31, 2-73
- ?syr2 2-32, 2-75
- ?tbmv 2-32, 2-77
- ?tbsv 2-32, 2-80
- ?tpmv 2-32, 2-83
- ?tpsv 2-32, 2-86
- ?trmv 2-32, 2-88
- ?trsv 2-32, 2-91
- コード例 C-3
- BLAS レベル 3 のルーチン
  - ?gemm 2-94, 2-95
  - ?hemm 2-94, 2-99
  - ?her2k 2-94, 2-105
  - ?herk 2-94, 2-102
  - ?symm 2-94, 2-108
  - ?syr2k 2-94, 2-115
  - ?syrk 2-94, 2-111
  - ?trmm 2-94, 2-119
  - ?trsm 2-94, 2-122
- コード例 C-4
- BRNG 10-1, 10-8
- Bunch-Kaufman 因子分解 3-8, 6-6
  - エルミート行列 3-26
    - 圧縮格納 3-32
  - 対称行列 3-23
    - 圧縮格納 3-29
- C**
  - C インターフェイス 11-69
  - Cauchy 10-57
  - CBLAS D-1
    - スパース BLAS D-16
    - 引数 D-1
    - レベル 1 (ベクトル演算) D-3
    - レベル 2 (行列 - ベクトル演算) D-5
    - レベル 3 (行列 - 行列演算) D-12
  - CommitDescriptor 11-10
  - CommitDescriptorDM 11-55
  - ComputeBackward 11-16
  - ComputeBackwardDM 11-60
  - ComputeForward 11-14
  - ComputeForwardDM 11-58
  - ConvCopyTask 10-131
  - ConvDeleteTask 10-130
  - ConvInternalPrecision 10-116
  - ConvSetDecimation 10-119
  - ConvSetMode 10-115
  - ConvSetStart 10-118
  - CopyDescriptor 11-11
  - CopyStream 10-25
  - CopyStreamState 10-26
  - CorrCopyTask 10-131
  - CorrDeleteTask 10-130
  - CorrSetInternalDecimation 10-119
  - CorrSetInternalPrecision 10-116
  - CorrSetMode 10-115
  - CorrSetStart 10-118
  - Cray 7-190
  - CreateDescriptor 11-8
  - CreateDescriptorDM 11-53
- D**
  - dcabs1 2-30
  - DeleteStream 10-24
  - DFT インターフェイス 11-3
  - DFT 計算 11-4
    - クラスタ DFTI 11-52
  - DFT ルーチン
    - DFT 計算
      - ComputeBackward 11-16
      - ComputeBackwardDM 11-60
      - ComputeForward 11-14
      - ComputeForwardDM 11-58
    - ステータス確認 11-5
      - ErrorClass 11-5
      - ErrorMessage 11-7
    - ステータス確認、クラスタ DFTI 11-53
    - ディスクリプタ構成
      - GetValue 11-21
      - GetValueDM 11-65

SetValue 11-19  
ディスクリプタ操作  
CommitDescriptor 11-10  
CopyDescriptor 11-11  
CreateDescriptor 11-8  
CreateDescriptorDM 11-53  
FreeDescriptor 11-13  
FreeDescriptorDM 11-57  
CommitDescriptorDM 11-55  
dNewAbstractStream 10-19  
DSS インターフェイス、スパースソルバ 8-14

## E

ErrorClass 11-5  
ErrorMessage 11-7  
ESSL ライブラリ 10-97  
Exponential 10-50

## F

FFT のデータ構造の要件 11-69  
FFTs  
1 次元  
実数から複素数 11-73  
複素数から実数 11-78  
複素数から複素数 11-70  
2 次元  
実数から複素数 11-86  
複素数から実数 11-93  
複素数から複素数 11-83  
実数から複素数  
1 次元 11-73  
2 次元 11-86  
複素数から実数  
1 次元 11-78  
複素数から複素数  
1 次元 11-70  
2 次元 11-83  
複素数から実数  
2 次元 11-93  
FFT。高速フーリエ変換を参照  
Fortran-95 インターフェイス規則  
BLAS、スパース BLAS 2-3  
LAPACK 3-3  
FreeDescriptor 11-13

FreeDescriptorDM 11-57  
Frobenius の値  
一般矩形行列 7-54  
Frobenius ノルムの値  
一般三重対角行列 5-121  
一般正方行列 5-119  
上 Hessenberg 行列 5-122, 7-56  
エルミート帯行列 5-125  
三角帯行列 5-135  
三角行列 5-138, 7-61  
圧縮格納 5-137  
実対称行列 5-132, 7-58  
実対称三重対角行列 5-130  
台形行列 5-138  
対称帯行列 5-124  
対称行列  
圧縮格納 5-127  
複素エルミート行列 5-133, 7-58  
圧縮格納 5-129  
複素エルミート三重対角行列 5-130  
複素対称行列 5-132

## G

Gamma 10-66  
Gaussian 10-43  
GaussianMV 10-45  
Gauss-Seidel の反復法、区間方程式 12-10, 12-24  
Geometric 10-77  
GetBrngProperties 10-92  
GetNumRegBrngs 10-38  
GetStreamStateBrng 10-37  
GetValue 11-21  
GetValueDM 11-65  
GFSR 10-4  
Givens 回転  
スパースベクトル 2-137  
パラメータ 2-19  
変形 Givens 変換パラメータ 2-23  
Gumbel 10-64

## H

Hansen-Bliek-Rohn プロシージャ、区間方程式 12-12  
Householder 行列  
LAPACK 5-168

ScaLAPACK 7-83  
 Householder 法、区間方程式 12-7, 12-24  
 Householder リフレクタ 7-172  
 Hypergeometric 10-81

## I

i?amax 2-28  
 i?amin 2-29  
 i?max1 5-22  
 IEEE 演算 7-53  
 IEEE スタンダード  
   実装 7-191  
   符号ビットの位置 7-193  
 ilaenv 5-316  
 iNewAbstractStream 10-17

## K

Krawczyk 反復法、区間方程式 12-8

## L

LAPACK ルーチン  
   1 次方程式の解の算出  
     ?gbtrs 3-37  
     ?getrs 3-35  
     ?gttrs 3-40  
     ?hetrs 3-55  
     ?hptrs 3-60  
     ?laln2 5-104  
     ?laqtr 5-158  
     ?pbtrs 3-48  
     ?potrs 3-43  
     ?pptrs 3-45  
     ?pttrs 3-51  
     ?sptrs 3-58  
     ?sytrs 3-53  
     ?tbtrs 3-68  
     ?tptrs 3-65  
     ?trtrs 3-63  
    $2 \times 2$  一般固有値問題 5-85  
    $2 \times 2$  エルミート行列  
     面回転 5-163  
    $2 \times 2$  三角行列  
     SVD 5-247

特異値 5-197  
 $2 \times 2$  実数非対称行列  
   Schur 因子分解 5-140  
 $2 \times 2$  直交行列 5-87  
 $2 \times 2$  対称行列  
   面回転 5-163  
 dqd 変換 5-240, 5-242  
 Householder 行列  
   基本リフレクタ 5-168  
 LQ 因子分解  
   ?gelq2 5-30  
   ?gelqf 4-26  
   ?orglq 4-29  
   ?ormlq 4-31  
   ?unglq 4-34  
   ?unmlq 4-36  
 LU 因子分解  
   一般帯行列 5-24  
 QL 因子分解  
   ?geql2 5-32  
   ?geqlf 4-39  
   ?orgql 4-41  
   ?ormql 4-45  
   ?ungql 4-43  
   ?unmql 4-48  
 QR 因子分解  
   ?geqpf 4-10  
   ?geqr2 5-33  
   ?geqp3 4-13  
   ?geqrf 4-7  
   ?ggqrf 4-71  
   ?ggrqf 4-74  
   ?laqp2 5-149  
   ?laqps 5-151  
   ?orgqr 4-16  
   ?ormqr 4-18  
   ?ungqr 4-21  
   ?unmqr 4-23  
   p?geqrf 6-75  
 RQ 因子分解  
   ?geqr2 5-35  
   ?gerqf 4-51  
   ?unmrq 4-60  
   ?orgrq 4-53  
   ?ormrq 4-57  
   ?ungrq 4-55

- RZ 因子分解
  - ?tzrf 4-62
  - ?ormrz 4-65
  - ?unmrz 4-68
- 圧縮形式のエルミート帯行列
  - 平衡化 5-155
- 圧縮形式の対称帯行列
  - 平衡化 5-155
- 一様分布 5-188
- 一般帯行列 5-118
  - 平衡化 5-145
- 一般行列
  - 基本リフレクタ 5-189
  - 二重対角形式への縮退 5-25, 5-42
  - ブロック・リフレクタ 5-191
- 一般矩形行列
  - LQ 因子分解 5-30
- 一般三重対角行列 5-89, 5-91, 5-92, 5-121, 5-179, 5-183
- 一般正方行列 5-99, 5-119, 5-198
  - QL 因子分解 5-32
  - QR 因子分解 5-33
  - RQ 因子分解 5-35
  - 上 Hessenberg 形式への縮退 5-28
  - 基本リフレクタ 5-164, 5-173
  - 行変換 5-249
  - ブロック・リフレクタ 5-166
  - 平衡化 5-147
  - 面回転 5-243
- 上 Hessenberg 行列 5-122
  - Schur 因子分解 5-96
  - 固有値 5-96
  - 指定された固有ベクトルの計算 5-79
- エルミート帯行列 5-125
  - 平衡化 5-153, 5-156
- エルミート行列
  - 固有値および固有ベクトルの計算 5-82
- 解の精度の改善と誤差の推定
  - ?ptrfs 3-122
  - ?gbrfs 3-107
  - ?porfs 3-113
  - ?gerfs 3-104
  - ?gtrfs 3-110
  - ?herfs 3-128
  - ?hprfs 3-134
  - ?pbrfs 3-119
  - ?pprfs 3-116
  - ?sprfs 3-131
  - ?syrf 3-125
  - ?tbrfs 3-143
  - ?tptrfs 3-140
  - ?trrfs 3-137
- 数のソート 5-245
- 環境問い合わせ 5-316
- 行列 - 行列の積
  - ?lagtm 5-91
- 行列の反転
  - ?potri 3-149
  - ?getri 3-146
  - ?hetri 3-155
  - ?hptri 3-159
  - ?pptri 3-151
  - ?sptri 3-157
  - ?sytri 3-153
  - ?tptri 3-163
  - ?trtri 3-161
- 行列の平衡化
  - ?gbequ 3-168
  - ?geequ 3-165
  - ?laqgb 5-145
  - ?laqge 5-147
  - ?laqsb 5-153
  - ?laqsp 5-155
  - ?laqsy 5-156
  - ?pbequ 3-175
  - ?poequ 3-170
  - ?ppequ 3-172
- 行列の列の置換 5-142
- 更新された上二重対角行列
  - SVD 5-214
- 最小固有値に対する近似 5-239
- 三角行列 5-138
  - 圧縮格納 5-137
- 三角分解
  - ?gbtrf 3-11
  - ?getrf 3-9
  - ?gttrf 3-13
  - ?hetrf 3-26
  - ?hptrf 3-32
  - ?pbrf 3-20
  - ?potrf 3-15
  - ?pptrf 3-18

- ?pttrf 3-22
- ?spttrf 3-29
- ?sytrf 3-23
- p?dbtrf 6-10
- 三角連立方程式 5-262, 5-267
- 三重帯行列 5-135
- 実数対称三重対角行列 5-50, 5-130
- 実数の  $2 \times 2$  行
  - 汎用 Schur 因子分解 5-94
- 実数の上準三角行列
  - 直交相似変換 5-84
- 実数の上二重対角行列
  - SVD 5-201, 5-226, 5-229
  - 特異値 5-200
- 実数の下二重対角行列
  - SVD 5-229
- 実数平方二重対角行列
  - 特異値 5-234
- 実対称行列 5-132
- 縮退されていない対称三重対角行列 5-55
- 条件数推定
  - ?pbcon 3-83
  - ?ppcon 3-81
  - ?disna 4-169
  - ?gbcon 3-73
  - ?gecon 3-71
  - ?gtcon 3-76
  - ?hecon 3-90
  - ?hpcon 3-94
  - ?pocon 3-78
  - ?ptcon 3-85
  - ?spcon 3-92
  - ?sycon 3-87
  - ?tbcon 3-101
  - ?tpcon 3-99
  - ?trcon 3-96
- シルベスター式
  - ?lasy2 5-250
  - ?tgsy2 5-310
  - ?trsyl 4-240
- 増加条件推定 5-101
- 台形行列 5-138, 5-271
- 対称帯行列 5-124
  - 平衡化 5-153, 5-156
- 対称行列
  - 圧縮格納 5-127
  - 固有値および固有ベクトルの計算 5-82
- 対称固有値問題
  - ?disna 4-169
  - ?hbtrd 4-141
  - ?hetrd 4-119
  - ?hptrd 4-133
  - ?opgtr 4-128
  - ?opmtr 4-130
  - ?upmtr 4-137
  - ?orgtr 4-114
  - ?ormtr 4-116
  - ?pteqr 4-159
  - ?sbtrd 4-139
  - ?sptrd 4-126
  - ?stebz 4-163
  - ?stedc 4-150
  - ?stegr 4-154
  - ?stein 4-167
  - ?steqr 4-146
  - ?sterf 4-144
  - ?sytrd 4-112
  - ?ungtr 4-122
  - ?unmtr 4-124
  - ?upgtr 4-135
- 補助
  - ?lae2 5-49
  - ?laebz 5-50
  - ?laed0 5-55
  - ?laed1 5-57
  - ?laed2 5-59
  - ?laed3 5-62
  - ?laed4 5-64
  - ?laed5 5-66
  - ?laed6 5-67
  - ?laed7 5-68
  - ?laed8 5-72
  - ?laed9 5-75
  - ?laeda 5-77
- 対称正定値三重対角行列
  - 固有値 5-235
- 置換リストの作成 5-117
- 特異値のセットの併合 5-204, 5-218
- 特異値分解 5-107, 5-111, 5-114
  - ?bdsdc 4-104
  - ?bdsqr 4-100
  - ?gbbrd 4-83

?gebrd 4-80	?hpevd 4-355
?orgbr 4-87	?hpevx 4-363
?ormbr 4-90	?sbev 4-367
?ungbr 4-93	?sbevd 4-372
?unmbr 4-97	?sbevz 4-380
ドライバルーチン	?spev 4-348
1 次方程式の解の算出	?spevd 4-352
?pbsvx 3-220	?spevx 4-359
?gbsv 3-186	?stev 4-389
?gbsvx 3-189	?stevd 4-391
?gesv 3-178	?stevr 4-398
?gesvx 3-180	?stevx 4-394
?gtsv 3-196	?syev 4-316
?gtsvx 3-198	?syevd 4-321
?hesv 3-238	?syevr 4-337
?hesvx 3-241	?syevx 4-328
?hpsv 3-252	特異値分解
?hpsvx 3-255	?gesdd 4-430
?pbsv 3-217	?gesvd 4-426
?posv 3-203	?ggsvd 4-434
?posvx 3-205	汎用 LLS 問題
?ppsv 3-210	?ggglm 4-313
?ppsvx 3-212	?gglse 4-310
?ptsv 3-227	汎用対称固有値問題
?ptsv 3-225	?hbgv 4-491
?spsv 3-245	?hbgvd 4-497
?spsvx 3-248	?hbgvx 4-506
?sysv 3-230	?hegv 4-444
?sysvx 3-233	?hegvd 4-451
線形最小二乗問題	?hegvx 4-460
?gels 4-295	?hpgv 4-469
?gelsd 4-305	?hpgvd 4-475
?gelss 4-302	?hpgvx 4-484
?gelsy 4-298	?sbgv 4-489
?lalsd ( 補助 ) 5-114	?sbgvd 4-494
?lals0 ( 補助 ) 5-107	?sbgvx 4-501
?lalsa ( 補助 ) 5-111	?spgv 4-466
対称固有値問題	?spgvd 4-472
?hbev 4-370	?spgvx 4-479
?hbevd 4-376	?sygv 4-441
?hbevz 4-384	?sygvd 4-447
?heev 4-319	?sygvx 4-455
?heevd 4-324	汎用非対称固有値問題
?heevr 4-342	?gges 4-511
?heevx 4-332	?ggesx 4-518
?hpev 4-350	?ggevd 4-525

- ?ggevxx 4-530
- 非対称固有値問題
  - ?gees 4-404
  - ?geesx 4-408
  - ?geev 4-414
  - ?geevx 4-419
- 二重対角分割統治 5-232
- 二乗和の更新 5-246
- 汎用固有値問題
  - ?hbgst 4-184
  - ?hegst 4-175
  - ?hpgst 4-179
  - ?pbstf 4-187
  - ?sbgst 4-182
  - ?spgst 4-177
  - ?sygst 4-173
- 比較的分離された固有値の検索 5-181
- 非対角成分と対角成分 5-233
- 非対称固有値問題
  - ?gebak 4-210
  - ?gebal 4-207
  - ?gehrd 4-193
  - ?hsein 4-217
  - ?hseqr 4-212
  - ?orghr 4-195
  - ?ormhr 4-198
  - ?trevc 4-222
  - ?trsen 4-235
  - ?trsna 4-227
  - ?trexc 4-232
  - ?unghr 4-201
  - ?unmhr 4-204
- 複素
  - 線形変換 5-12
- 複素エルミート行列 5-133
  - 圧縮格納 5-129
- 複素エルミート三重対角行列 5-130
- 複素行列の乗算 5-10
- 複素対称圧縮行列
  - 対称階数 1 の更新 5-17
- 複素対称行列 5-132
  - 行列 - ベクトルの積 5-19
  - 固有値および固有ベクトルの計算 5-13
  - 対称階数 1 の更新 5-21
- 複素ベクトル
  - 行列 - ベクトルの積 5-15
- 最大絶対値を持つ成分のインデックス 5-22
- 真の絶対値を使用した 1- ノルム 5-23
- 面回転 5-14
- ブロック・リフレクタ
  - 三角係数 5-170, 5-193
- 平方根 5-144, 5-208, 5-211, 5-212, 5-222, 5-224, 5-321
- ベクトルの線形従属性 5-141
- 補助ルーチン
  - ?gbtf2 5-24
  - ?gebd2 5-25
  - ?gehd2 5-28
  - ?gelq2 5-30
  - ?geql2 5-32
  - ?geqr2 5-33
  - ?gerq2 5-35
  - ?gesc2 5-37
  - ?getc2 5-38
  - ?getf2 5-40
  - ?gtts2 5-41
  - ?hetf2 5-306
  - ?labrd 5-42
  - ?lacgv 5-10
  - ?lacon 5-45
  - ?lacpy 5-47
  - ?lacrm 5-10
  - ?lactr 5-12
  - ?ladiv 5-48
  - ?lae2 5-49
  - ?laebz 5-50
  - ?laed0 5-55
  - ?laed1 5-57
  - ?laed2 5-59
  - ?laed3 5-62
  - ?laed4 5-64
  - ?laed5 5-66
  - ?laed6 5-67
  - ?laed7 5-68
  - ?laed8 5-72
  - ?laed9 5-75
  - ?laeda 5-77
  - ?laein 5-79
  - ?laesy 5-13
  - ?laev2 5-82
  - ?laexc 5-84

?lag2 5-85	?larfg 5-168
?lags2 5-87	?larft 5-170
?lagtf 5-89	?larfx 5-173
?lagtm 5-91	?largv 5-174
?lagts 5-92	?larnv 5-176
?lagv2 5-94	?larrb 5-177
?lahef 5-255	?larre 5-179
?lahqr 5-96	?larrf 5-181
?lahrd 5-99	?larrv 5-183
?laic1 5-101	?lartg 5-185
?laln2 5-104	?lartv 5-187
?lals0 5-107	?laruv 5-188
?lalsa 5-111	?larz 5-189
?lalsd 5-114	?larzb 5-191
?lamrg 5-117	?larzt 5-193
?langb 5-118	?las2 5-197
?lange 5-119	?lascl 5-198
?langt 5-121	?lasd0 5-200
?lanhb 5-125	?lasd1 5-201
?lanhe 5-133	?lasd2 5-204
?lanhp 5-129	?lasd3 5-208
?lanhs 5-122	?lasd4 5-211
?lansb 5-124	?lasd5 5-212
?lansp 5-127	?lasd6 5-214
?lanst/?lanht 5-130	?lasd7 5-218
?lansy 5-132	?lasd8 5-222
?lantb 5-135	?lasd9 5-224
?lantp 5-137	?lasda 5-226
?lantr 5-138	?lasdq 5-229
?lanv2 5-140	?lasdt 5-232
?lapll 5-141	?laset 5-233
?lapmt 5-142	?lasq1 5-234
?lapy2 5-144	?lasq2 5-235
?lapy3 5-144	?lasq3 5-237
?laqgb 5-145	?lasq4 5-239
?laqge 5-147	?lasq5 5-240
?laqp2 5-149	?lasq6 5-242
?laqps 5-151	?lasr 5-243
?laqsb 5-153	?lasrt 5-245
?laqsp 5-155	?lassq 5-246
?laqsy 5-156	?lasv2 5-247
?laqtr 5-158	?laswp 5-249
?lar1v 5-161	?lasy2 5-250
?lar2v 5-163	?lasyf 5-252
?larf 5-164	?latbs 5-257
?larfb 5-166	?latdf 5-260



- ?latps 5-262
  - ?latrd 5-264
  - ?latrs 5-267
  - ?latrz 5-271
  - ?lauu2 5-273
  - ?lauum 5-275
  - ?org2l/?ung2l 5-276
  - ?org2r/?ung2r 5-278
  - ?orgl2l/?ungl2 5-279
  - ?orgr2/?ungr2 5-281
  - ?orm2l/?unm2l 5-282
  - ?orm2r/?unm2r 5-285
  - ?orml2/?unml2 5-287
  - ?ormr2/?unmr2 5-290
  - ?ormr3/?unmr3 5-292
  - ?pbt2 5-295
  - ?potf2 5-296
  - ?ptts2 5-298
  - ?rot 5-14
  - ?rscl 5-299
  - ?spmv 5-15
  - ?spr 5-17
  - ?sum1 5-23
  - ?sygs2/?hegs2 5-300
  - ?symv 5-19
  - ?syr 5-21
  - ?sytd2/?hetd2 5-302
  - ?sytf2 5-304
  - ?tgex2 5-308
  - ?tgsy2 5-310
  - ?trti2 5-314
  - i?max1 5-22
  - マシン・パラメータの決定 5-323, 5-324
  - 面回転 5-185, 5-187, 5-243
  - 面回転ベクトル 5-174
  - 文字の同一性の確認 5-320
  - 文字列の同一性の確認 5-320
  - ユーティリティ関数とルーチン
    - ?labad 5-321
    - ?lamc1 5-323
    - ?lamc2 5-324
    - ?lamc3 5-325
    - ?lamc4 5-325
    - ?lamc5 5-326
    - ?lamch 5-322
    - ieeck 5-319
    - ilaenv 5-316
    - lsame 5-320
    - lsamen 5-320
    - second/dsecnd 5-327
    - xerbla 5-327
  - 乱数ベクトル 5-176
  - 複素ベクトルの共役 5-10
  - 無限大演算の安全性の確認 5-319
  - Laplace 10-52
  - LeapfrogStream 10-31
  - LoadStreamF 10-29
  - Lognormal 10-61
  - LQ 因子分解 4-5
    - 一般矩形行列 7-24
    - 一般正方行列 5-30
    - 成分の計算
      - 実直交行列 Q 6-95
      - 直交行列 Q 4-29
      - ユニタリ行列 Q 4-34, 6-97
  - lsame 5-320
  - lsamen 5-320
  - LU 因子分解 3-9, 6-6
    - 1 次方程式の解の算出
      - 一般行列 5-37
      - 三重対角行列 5-41, 5-92
      - 正方行列 6-216
    - 一般帯行列 5-24
    - 一般行列 5-40, 7-34
    - 帯行列 3-11, 6-8, 6-10
      - 非ブロック化アルゴリズム 7-179
      - ブロック化アルゴリズム 7-181
    - 完全ピボット演算 5-38, 5-260
    - 三角帯行列 7-10
    - 三重対角帯行列 7-13
    - 三重対角行列 3-13, 5-89, 7-183
    - 対角優位三重対角行列 6-19
    - 部分的なピボット演算 5-40, 7-34
- ## M
- mkl\_cvt\_to\_null\_terminated\_str 8-28
  - mkl\_dcoogemv 2-161
  - mkl\_dcoomm 2-196
  - mkl\_dcoomv 2-158
  - mkl\_dcoosm 2-210
  - mkl\_dcoosv 2-180

mkl\_dcoosymv 2-163  
 mkl\_dcootrsv 2-182  
 mkl\_dcscmm 2-194  
 mkl\_dcscmv 2-156  
 mkl\_dcscsm 2-207  
 mkl\_dcscsv 2-178  
 mkl\_dcsgemv 2-152  
 mkl\_dcsmmm 2-191  
 mkl\_dcsmv 2-149  
 mkl\_dcsm 2-204  
 mkl\_dcsmv 2-173  
 mkl\_dcsmymv 2-154  
 mkl\_dcsmtrsv 2-175  
 mkl\_ddiagemv 2-167  
 mkl\_ddiamm 2-199  
 mkl\_ddiamv 2-165  
 mkl\_ddiasm 2-212  
 mkl\_ddiasv 2-184  
 mkl\_ddiasymv 2-169  
 mkl\_ddiatsv 2-187  
 mkl\_dskymm 2-202  
 mkl\_dskymv 2-171  
 mkl\_dskysm 2-215  
 mkl\_dskysv 2-189  
 MPI 6-1

## N

NegBinomial 10-87  
 NewStream 10-14  
 NewStreamEx 10-15

## P

p?dbsv 6-225  
 p?dbtrf 6-10  
 p?dbtrs 6-37  
 p?dbtrsv 7-10  
 p?dtsv 6-228  
 p?dttrf 6-19  
 p?dttrs 6-34  
 p?dttrsv 7-13  
 p?gbsv 6-222  
 p?gbtrf 6-8  
 p?gbtrs 6-24  
 p?gebd2 7-17

p?gebrd 6-193  
 p?gecon 6-42  
 p?geequ 6-70  
 p?gehd2 7-21  
 p?gehrd 6-180  
 p?gelq2 7-24  
 p?gelqf 6-92  
 p?gels 6-245  
 p?geql2 7-26  
 p?geqlf 6-106  
 p?geqpf 6-78  
 p?geqr2 7-29  
 p?geqrf 6-75  
 p?gerfs 6-52  
 p?gerq2 7-31  
 p?gerqf 6-120  
 p?gesv 6-214  
 p?gesvd 6-267  
 p?gesvx 6-216  
 p?getf2 7-34  
 p?getrf 6-6  
 p?getri 6-64  
 p?getrs 6-22  
 p?ggqrf 6-145  
 p?ggrqf 6-149  
 p?heevx 6-259  
 p?hegst 6-210  
 p?hegvx 6-279  
 p?hetrd 6-163  
 p?labad 7-190  
 p?labrd 7-35  
 p?lacgv 7-6  
 p?lachkiee 7-191  
 p?lacon 7-39  
 p?laconsb 7-41  
 p?lacp2 7-43  
 p?lacp3 7-44  
 p?lacpy 7-46  
 p?laevswp 7-48  
 p?lahqr 6-190  
 p?lahrd 7-50  
 p?laiect 7-53  
 p?lamch 7-192  
 p?lange 7-54  
 p?lanhs 7-56  
 p?lansy, p?lanhe 7-58

- p?lantr 7-61  
p?lapiv 7-63  
p?laqge 7-66  
p?laqsy 7-69  
p?lared1d 7-71  
p?lared2d 7-72  
p?larf 7-74  
p?larfb 7-77  
p?larfc 7-80  
p?larfg 7-83  
p?larft 7-85  
p?larz 7-88  
p?larzb 7-92  
p?larzc 7-96  
p?larzt 7-99  
p?lascl 7-103  
p?laset 7-105  
p?lasmsub 7-107  
p?lasnbt 7-193  
p?lassq 7-108  
p?laswp 7-110  
p?latra 7-112  
p?latrd 7-113  
p?latrs 7-117  
p?latrz 7-119  
p?lauu2 7-122  
p?lauum 7-124  
p?lawil 7-125  
p?max1 7-7  
p?org2l/p?ung2l 7-126  
p?org2r/p?ung2r 7-129  
p?orgl2/p?ungl2 7-131  
p?orglq 6-95  
p?orgql 6-109  
p?orgqr 6-81  
p?orgr2/p?ungr2 7-134  
p?orgrq 6-123  
p?orm2l/p?unm2l 7-136  
p?orm2r/p?unm2r 7-140  
p?ormbr 6-198  
p?ormhr 6-184  
p?orml2/p?unml2 7-144  
p?ormlq 6-99  
p?ormql 6-113  
p?ormqr 6-85  
p?ormr2/p?unmr2 7-148  
p?ormrq 6-127  
p?ormrz 6-137  
p?ormtr 6-160  
p?pbsv 6-240  
p?pbtrf 6-15  
p?pbtrs 6-29  
p?pbtrsv 7-152  
p?pocon 6-45  
p?poequ 6-72  
p?porfs 6-56  
p?posv 6-231  
p?posvx 6-233  
p?potf2 7-159  
p?potrf 6-13  
p?potri 6-66  
p?potrs 6-27  
p?ptsv 6-242  
p?pttrf 6-17  
p?pttrs 6-31  
p?pttrsv 7-156  
p?rscl 7-161  
p?stebz 6-171  
p?stein 6-175  
p?sum1 7-9  
p?syev 6-249  
p?syevx 6-252  
p?sygs2/p?hegs2 7-162  
p?sygst 6-208  
p?sygvx 6-271  
p?sytd2/p?hetd2 7-165  
p?sytrd 6-156  
p?trcon 6-48  
p?trrfs 6-60  
p?trti2 7-169  
p?trtri 6-68  
p?trtrs 6-40  
p?tzrzf 6-134  
p?unglq 6-97  
p?ungql 6-111  
p?ungqr 6-83  
p?ungrq 6-125  
p?unmbr 6-203  
p?unmhr 6-187  
p?unmlq 6-103  
p?unmql 6-117  
p?unmqr 6-89

p?unmrq 6-131  
p?unmrz 6-141  
p?unmtr 6-167  
PARDISO 8-1  
pardiso 関数 8-3  
pdlaiectb 7-53  
pdlaiectl 7-53  
Poisson 10-83  
PoissonV 10-85  
pslaiect 7-53  
pxerbla 7-194

## Q

QL 因子分解  
一般行列との掛け合わせ  
直交行列 Q 6-113  
ユニタリ行列 Q 6-117  
一般矩形行列  
ScaLAPACK 7-26  
一般正方行列  
LAPACK 5-32  
成分の計算  
実行列 Q 4-41  
直交行列 Q 6-109  
複素行列 Q 4-43  
ユニタリ行列 Q 6-111  
QR 因子分解 4-5  
一般矩形行列  
ScaLAPACK 7-29, 7-31  
一般正方行列  
LAPACK 5-33, 5-35  
成分の計算  
直交行列 Q 4-16, 6-81  
ユニタリ行列 Q 4-21, 6-83  
ピボット演算 4-10, 4-13, 5-149, 5-151  
ScaLAPACK 6-78

## R

Rayleigh 10-59  
RCI (P)CG インターフェイス 8-30  
RCI (P)CG スパース・ソルバ・ルーチン  
dcg 8-39  
dcg\_check 8-38  
dcg\_get 8-41

dcg\_init 8-37  
RegisterBrng 10-91  
Rex-Rohn テスト 12-19, 12-20  
Ris-Beeck スペクトル条件 12-19  
RQ 因子分解  
成分の計算  
実行列 Q 4-53  
直交行列 Q 6-123  
複素行列 Q 4-55  
ユニタリ行列 Q 6-125  
Rump 判断 12-20

## S

SaveStreamF 10-28  
ScaLAPACK 6-1  
ScaLAPACK ルーチン  
1 次元配列の再分配 7-71, 7-72  
1 次方程式の解の算出  
?dtrsv 7-184  
?pttrsv 7-186  
p?dbtrs 6-37  
p?dtrrs 6-34  
p?gbtrs 6-24  
p?getrs 6-22  
p?potrs 6-27  
p?pttrs 6-31  
p?trtrs 6-40  
Householder 行列  
基本リフレクタ 7-83  
LQ 因子分解  
p?gelq2 7-24  
p?gelqf 6-92  
p?orglq 6-95  
p?ormlq 6-99  
p?unglq 6-97  
p?unmlq 6-103  
LU 因子分解  
p?getf2 7-34  
p?dbtrsv 7-10  
p?dttrf 6-19  
p?dttrsv 7-13  
QL 因子分解  
?geqlf 6-106  
?ungql 6-111  
p?geql2 7-26

- p?orgql 6-109
- p?ormql 6-113
- p?unmql 6-117
- QR 因子分解
  - p?geqpf 6-78
  - p?geqr2 7-29
  - p?ggqrf 6-145
  - p?orgqr 6-81
  - p?ormqr 6-85
  - p?ungqr 6-83
  - p?unmqr 6-89
- RQ 因子分解
  - p?gerq2 7-31
  - p?gerqf 6-120
  - p?ggrqf 6-149
  - p?orgrq 6-123
  - p?ormrq 6-127
  - p?ungrq 6-125
  - p?unmrq 6-131
- RZ 因子分解
  - p?ormrz 6-137
  - p?trzf 6-134
  - p?unmrz 6-141
- 一般行列
  - LU 因子分解 7-34
  - 上 Hessenberg 形式への縮退 7-21
  - 基本リフレクタ 7-88
  - ブロック・リフレクタ 7-92
- 一般矩形行列 7-103
  - LQ 因子分解 7-24
  - QL 因子分解 7-26
  - QR 因子分解 7-29
  - RQ 因子分解 7-31
  - 基本リフレクタ 7-74
  - 行交換 7-110
  - 実二重対角形式への縮退 7-17
  - 二重対角形式への縮退 7-35
- エラー制御
  - pxerbla 7-194
- 解の精度の改善と誤差の推定
  - p?gerfs 6-52
  - p?porfs 6-56
- 行列の反転
  - p?getri 6-64
  - p?potri 6-66
  - p?trtri 6-68
- 行列の平衡化
  - p?geequ 6-70
  - p?poequ 6-72
- 誤差推定
  - p?trrfs 6-60
- コレスキー因子分解 6-17
- 三角分解
  - ?dtrf 7-183
  - ?dbtrf 7-181
  - p?dbtrsv 7-10
  - p?dttrsv 7-13
  - p?gbtrf 6-8
  - p?getrf 6-6
  - p?pbtrf 6-15
  - p?potrf 6-13
  - p?pttrf 6-17
- 三角連立方程式 7-117
- 条件数推定
  - p?gecon 6-42
  - p?pocon 6-45
  - p?trcon 6-48
- 台形行列 7-119
- 対称固有値問題
  - ?stein2 7-177
  - ?steqr2 7-188
  - p?hetrd 6-163
  - p?ormtr 6-160
  - p?stebz 6-171
  - p?stein 6-175
  - p?sytrd 6-156
  - p?unmtr 6-167
- 特異値分解
  - p?gebrd 6-193
  - p?ormbr 6-198
  - p?unmbr 6-203
- ドライバルーチン
  - p?dbsv 6-225
  - p?dtsv 6-228
  - p?gbsv 6-222
  - p?gels 6-245
  - p?gesv 6-214
  - p?gesvd 6-267
  - p?gesvx 6-216
  - p?heevx 6-259
  - p?hegvx 6-279
  - p?pbsv 6-240

p?posv 6-231	p?laconsb 7-41
p?posvx 6-233	p?lcp2 7-43
p?ptsv 6-242	p?lcp3 7-44
p?syev 6-249	p?lcpy 7-46
p?syevx 6-252	p?laevswp 7-48
p?sygvx 6-271	p?lahrd 7-50
二乗和の更新 7-108	p?laiect 7-53
汎用固有値問題	p?lange 7-54
p?hegst 6-210	p?lanhs 7-56
p?sygst 6-208	p?lansy、p?lanhe 7-58
非対称固有値問題	p?lantr 7-61
p?gehrd 6-180	p?lapiv 7-63
p?lahqr 6-190	p?laqge 7-66
p?ormhr 6-184	p?laqsy 7-69
p?unmhr 6-187	p?lared1d 7-71
複素行列	p?lared2d 7-72
複素基本リフレクタ 7-96	p?larf 7-74
複素ベクトル	p?larfb 7-77
真の絶対値を使用した 1- ノルム 7-9	p?larfc 7-80
複素ベクトルの共役 7-6	p?larfg 7-83
ブロック・リフレクタ	p?larft 7-85
三角係数 7-85, 7-99	p?larz 7-88
補助ルーチン	p?larzb 7-92
?combamax1 7-8	p?larzc 7-96
?dbtrf 7-181	p?larzt 7-99
?dbtf2 7-179	p?lascl 7-103
?dttrf 7-183	p?laset 7-105
?dttrsv 7-184	p?lasmsub 7-107
?lamsh 7-171	p?lassq 7-108
?laref 7-172	p?laswp 7-110
?lasorte 7-175	p?latra 7-112
?lasrt2 7-176	p?latrd 7-113
?pttrsv 7-186	p?latrs 7-117
?stein2 7-177	p?latrz 7-119
?steqr2 7-188	p?lauu2 7-122
p?dbtrsv 7-10	p?lauum 7-124
p?gebd2 7-17	p?lawil 7-125
p?gehd2 7-21	p?max1 7-7
p?gelq2 7-24	p?org2l/p?ung2l 7-126
p?geql2 7-26	p?org2r/p?ung2r 7-129
p?geqr2 7-29	p?orgl2/p?ungl2 7-131
p?gerq2 7-31	p?orgr2/p?ungr2 7-134
p?getf2 7-34	p?orm2l/p?unm2l 7-136
p?labrd 7-35	p?orm2r/p?unm2r 7-140
p?lacgv 7-6	p?orml2/p?unml2 7-144
p?lacon 7-39	p?ormr2/p?unmr2 7-148

p?pbtrsv 7-152  
 p?potf2 7-159  
 p?pttrsv 7-156  
 p?rscl 7-161  
 p?sum1 7-9  
 p?sygs2/p?hegs2 7-162  
 p?sytd2/p?hetd2 7-165  
 p?trti2 7-169  
 pdlaiectb 7-53  
 pdlaiectl 7-53  
 pslaiect 7-53  
 ユーティリティ関数とルーチン  
 p?labad 7-190  
 p?lachkieee 7-191  
 p?lamch 7-192  
 p?lasnbt 7-193  
 pxerbla 7-194  
 Schulz 区間プロシージャ 12-18  
 Schur 因子分解 5-94, 5-96, 5-140  
 SetValue 11-19  
 SkipAheadStream 10-34  
 sNewAbstractStream 10-22  
 SVD (特異値分解)  
   LAPACK 4-78  
   ScaLAPACK 6-193

## U

Uniform (離散) 10-71  
 Uniform (連続) 10-40  
 UniformBits 10-73

## V

VML 9-1  
 VML 関数  
   pack/unpack 関数  
     Pack 9-39  
     Unpack 9-41  
   サービス関数  
     ClearErrorCallBack 9-53  
     ClearErrStatus 9-49  
     GetErrorCallBack 9-52  
     GetErrStatus 9-48  
     GetMode 9-46  
     SetErrorCallBack 9-49

SetErrStatus 9-47  
 SetMode 9-44

### 数学関数

Acos 9-25  
 Acosh 9-32  
 Asin 9-26  
 Asinh 9-33  
 Atan 9-27  
 Atan2 9-28  
 Atanh 9-34  
 Cbrt 9-13  
 Cos 9-21  
 Cosh 9-29  
 Div 9-10  
 Erf 9-35  
 Erfc 9-37  
 Exp 9-18  
 Inv 9-9  
 InvCbrt 9-14  
 InvSqrt 9-12  
 Ln 9-19  
 Log10 9-20  
 Pow 9-15  
 Powx 9-16  
 Sin 9-22  
 SinCos 9-23  
 Sinh 9-30  
 Sqrt 9-11  
 Tan 9-24  
 Tanh 9-31

### VSL ルーチン

アドバンスド・サービス・サブルーチン  
   GetBrngProperties 10-92  
   RegisterBrng 10-91  
 サービス・サブルーチン  
   CopyStream 10-25  
   CopyStreamState 10-26  
   DeleteStream 10-24  
   dNewAbstractStream 10-19  
   GetNumRegBrngs 10-38  
   GetStreamStateBrng 10-37  
   iNewAbstractStream 10-17  
   LeapfrogStream 10-31  
   LoadStreamF 10-29  
   NewStream 10-14  
   NewStreamEx 10-15

SaveStreamF 10-28  
SkipAheadStream 10-34  
sNewAbstractStream 10-22

#### 生成器

Bernoulli 10-75  
Beta 10-68  
Binomial 10-79  
Cauchy 10-57  
Exponential 10-50  
Gaussian 10-43  
GaussianMV 10-45  
Geometric 10-77  
Gumbel 10-64  
Hypergeometric 10-81  
Laplace 10-52  
Lognormal 10-61  
NegBinomial 10-87  
Poisson 10-83  
PoissonV 10-85  
Rayleigh 10-59  
Uniform ( 離散 ) 10-71  
Uniform ( 連続 ) 10-40  
UniformBits 10-73  
Weibull 10-54  
Gamma 10-66

#### 畳み込み / 相関

CopyTask 10-131  
DeleteTask 10-130  
Exec 10-122  
Exec1D 10-124  
ExecX 10-126  
ExecX1D 10-128  
NewTask 10-104  
NewTask1D 10-106  
NewTaskX 10-108  
NewTaskX1D 10-111  
SetInternalPrecision 10-116  
SetInternalDecimation 10-119  
SetMode 10-115  
SetStart 10-118

## W

Weibull 10-54  
Wilkinson 変換 7-125

## X

xerbla、エラー報告ルーチン 2-1, 5-327, 9-6

## あ

圧縮格納体系 B-4

圧縮形式 11-29

圧縮形式の三角行列

1- ノルムの値 5-137

Frobenius ノルムの値 5-137

最大絶対値の成分 5-137

無限ノルムの値 5-137

圧縮形式のスパースベクトル 2-126

圧縮形式のスパースベクトルの成分のフル圧縮形式  
への分散 2-139

圧縮形式のスパースベクトルのフル格納形式への変  
換 2-139

圧縮形式の対称行列

1- ノルムの値 5-127

Frobenius ノルムの値 5-127

最大絶対値の成分 5-127

無限ノルムの値 5-127

圧縮形式の複素エルミート行列

1- ノルムの値 5-129

Frobenius ノルムの値 5-129

最大絶対値の成分 5-129

無限ノルムの値 5-129

## い

一般行列

LQ 因子分解 4-26, 6-92

LU 因子分解 3-9, 5-40, 6-6, 7-34

帯格納 3-11, 5-24, 6-8, 6-10, 7-179, 7-181

QL 因子分解

LAPACK 4-39

ScaLAPACK 6-106

QR 因子分解 4-7, 4-74, 6-75

ピボット演算 4-10, 4-13, 6-78

RQ 因子分解

LAPACK 4-51

ScaLAPACK 6-149

上 Hessenberg 形式への縮退 7-21

階数 1 の更新 2-38

階数 1 の更新、共役 2-40



- 階数 1 の更新、非共役 2-42
- 基本リフレクタ 5-189, 7-88
- 行列の逆転
  - LAPACK 3-146
  - ScaLAPACK 6-64
- 行列 - ベクトルの積 2-35
  - 帯格納 2-32
- 固有値問題 4-189, 4-243, 6-179
- 条件数の推定 3-71, 6-42, 6-45, 6-48
  - 帯格納 3-73
- スカラ - 行列 - 行列の積 2-95
- 直交行列による乗算
  - LQ 因子分解から 5-287, 7-144
  - QR 因子分解から 5-285, 7-140
  - RQ 因子分解から 5-290, 7-148
  - RZ 因子分解から 5-292
- 二重対角形式への縮退 5-25, 5-42, 7-17
- ブロック・リフレクタ 5-191, 7-92
- ユニタリ行列による乗算
  - LQ 因子分解から 5-287, 7-144
  - QR 因子分解から 5-285, 7-140
  - RQ 因子分解から 5-290, 7-148
  - RZ 因子分解から 5-292
- 連立 1 次方程式の解の算出 3-35, 6-22
  - 帯格納
    - LAPACK 3-37
    - ScaLAPACK 6-24
- 一般矩形行列
  - 1- ノルムの値
    - ScaLAPACK 7-54
  - Frobenius ノルムの値
    - ScaLAPACK 7-54
  - LQ 因子分解
    - LAPACK 5-30
    - ScaLAPACK 7-24
  - QL 因子分解
    - ScaLAPACK 7-26
  - QR 因子分解
    - ScaLAPACK 7-29
  - RQ 因子分解
    - ScaLAPACK 6-120, 7-31
  - 基本リフレクタ
    - LAPACK 7-80
    - ScaLAPACK 7-74
  - 行交換
    - ScaLAPACK 7-110
- 最初の列への縮退
  - ScaLAPACK 7-50
- 最大絶対値の成分
  - ScaLAPACK 7-54
- 乗算
  - ScaLAPACK 7-103
- スケーリング 7-66
- 二重対角形式への縮退 7-35
- ブロック・リフレクタ
  - ScaLAPACK 7-77
- 無限ノルムの値
  - ScaLAPACK 7-54
- 一般矩形分散行列
  - スケール係数の計算 6-70
  - 平衡化 6-70
- 一般三角行列
  - LU 因子分解
    - 帯格納 7-10
- 一般三重対角行列
  - 1- ノルムの値 5-121
  - Frobenius ノルムの値 5-121
  - LU 因子分解
    - 帯格納 7-13
  - 最大絶対値の成分 5-121
  - 無限ノルムの値 5-121
- 一般正方向列
  - 1- ノルムの値
    - LAPACK 5-119
  - Frobenius ノルムの値
    - LAPACK 5-119
  - QL 因子分解
    - LAPACK 5-32
  - QR 因子分解
    - LAPACK 5-33
  - RQ 因子分解
    - LAPACK 5-35
  - 上 Hessenberg 形式への縮退 5-28
  - 基本リフレクタ 5-173
    - LAPACK 5-164
  - 行変換
    - LAPACK 5-249
  - 最大絶対値の成分
    - LAPACK 5-119
  - 乗算
    - LAPACK 5-198
  - 先頭列の縮退

- LAPACK 5-99
- 対角和 7-112
- ブロック・リフレクタ
  - LAPACK 5-166
- 無限ノルムの値
  - LAPACK 5-119
- 因子分解
  - Bunch-Kaufman
    - LAPACK 3-8
    - ScaLAPACK 6-6
  - LU
    - LAPACK 3-8
    - ScaLAPACK 6-6
  - 上台形行列 5-271
  - コレスキー 6-6
    - LAPACK 3-8, 5-295, 5-296
    - ScaLAPACK 7-159
  - 三角分解を参照
  - 対角ピボット
    - エルミート行列 3-241
      - 圧縮 3-255
    - 対称行列 3-233
      - 不定値 5-304
    - エルミート行列
      - 複素数 5-306
  - 直交
    - LAPACK 4-6
    - ScaLAPACK 6-75
  - 部分
    - 実数 / 複素対称行列 5-252
    - 複素エルミート無限行列 5-255
  - 対角ピボット
    - 対称行列
      - 圧縮 3-248

## う

- 上 Hessenberg 行列 4-189, 4-243
  - 1- ノルムの値
    - LAPACK 5-122
    - ScaLAPACK 7-56
  - Frobenius ノルムの値
    - LAPACK 5-122
    - ScaLAPACK 7-56
  - ScaLAPACK 6-179
  - 最大絶対値の成分

- LAPACK 5-122
- ScaLAPACK 7-56
- 無限ノルムの値
  - LAPACK 5-122
  - ScaLAPACK 7-56
- 上 Hessenberg 形式への縮退
  - 一般行列 7-21
  - 一般正方行列 5-28

## え

- エラー診断
  - VML 9-6
- エラー制御
  - pxerbla 7-194
  - xerbla 2-1, 5-327
- エルミート帯行列
  - 1- ノルムの値 5-125
  - Frobenius ノルムの値 5-125
  - 最大絶対値の成分 5-125
  - 無限ノルムの値 5-125
- エルミート行列 4-107, 4-172
  - Bunch-Kaufman 因子分解 3-26
    - 圧縮格納 3-32
  - 階数 1 の更新 2-49
    - 圧縮格納 2-56
  - 階数 2 の更新 2-51
    - 圧縮格納 2-58
  - 階数 2k の更新 2-105
  - 階数 n の更新 2-102
  - 行列の逆転 3-155
    - 圧縮格納 3-159
  - 行列 - ベクトルの積 2-47
    - 圧縮格納 2-54
    - 帯格納 2-44
  - 固有値と固有ベクトル 6-259
  - 三重対角形式への縮退
    - LAPACK 5-264, 5-302
    - ScaLAPACK 7-113, 7-165
  - 条件数の推定 3-90
    - 圧縮格納 3-94
  - スカラ - 行列 - 行列の積 2-99
  - スケーリング 7-69
  - 標準形式への縮退
    - LAPACK 5-300
    - ScaLAPACK 7-162

- 連立 1 次方程式の解の算出 3-55
  - 圧縮格納 3-60
- 汎用固有値問題 4-172
- エルミート正定値帯行列
  - コレスキー因子分解 5-295
- エルミート正定値行列
  - 行列の逆転 3-149
    - 圧縮格納 3-151
  - コレスキー因子分解 3-15, 5-296, 6-13, 7-159
    - 圧縮格納 3-18
    - 帯格納 3-20, 6-15
  - 条件数の推定 3-78
    - 圧縮格納 3-81
    - 帯格納 3-83
  - 連立 1 次方程式の解の算出 3-43, 6-27
    - 圧縮格納 3-45
    - 帯格納 3-48, 6-29
- エルミート正定値三重対角行列
  - 連立 1 次方程式の解の算出 6-31
- エルミート正定値分散行列
  - 行列の逆転 6-66
  - スケール係数の計算 6-72
  - 平衡化 6-72

## お

- 帯格納体系 B-4

## か

- 階数 1 の更新
  - 一般行列 2-38
  - エルミート行列 2-49
    - 圧縮格納 2-56
  - 共役、一般行列 2-40
  - 実対称行列 2-73
    - 圧縮格納 2-66
  - 非共役、一般行列 2-42
  - 複素対称行列 5-21
    - 圧縮格納 5-17
- 階数 2 の更新
  - エルミート行列 2-51
    - 圧縮格納 2-58
  - 対称行列 2-75
    - 圧縮格納 2-68
- 階数 2k の更新

- エルミート行列 2-105
- 対称行列 2-115
- 階数  $n$  の更新
  - エルミート行列 2-102
  - 対称行列 2-111
- 回転
  - Givens 回転に対するパラメータ 2-19
  - スパースベクトル 2-137
  - 変形 Givens 変換のパラメータ 2-23
  - 変形面における点 2-21
  - 面における点 2-17
- ガウス法、区間方程式 12-5, 12-24
- 格納、スパース行列 A-8
- 簡易ドライバ 6-4
- 関数命名規則
  - VML 9-2

## き

- 擬似乱数 10-1
- 基本準乱数生成器
  - Niederreiter 10-8
  - Sobol 10-8
- 基本生成器の登録 10-89
- 基本乱数生成器 10-1, 10-8
  - GFSR 10-8
  - MCG、32 ビット 10-8
  - MCG、59 ビット 10-8
  - Mersenne Twister
    - MT19937 10-8
    - MT2203 10-8
  - MRG 10-8
  - Wichmann-Hill 10-8
- 基本リフレクタ
  - LAPACK 生成 5-168
  - ScaLAPACK 生成 7-83
  - 一般行列 5-189, 7-88
  - 一般矩形行列
    - ScaLAPACK 7-74, 7-80
  - 一般正方行列
    - LAPACK 5-164, 5-173
  - 複素基本リフレクタ
    - ScaLAPACK 7-96
  - 複素行列 7-96
- 逆行列。行列の逆転を参照
- 共役勾配ソルバ 8-30

共役ベクトル 7-6  
 行列 - 行列演算  
   階数  $2k$  の更新  
     エルミート行列 2-105  
     対称行列 2-115  
   階数  $n$  の更新  
     エルミート行列 2-102  
     対称行列 2-111  
   スカラ - 行列 - 行列の積  
     エルミート行列 2-99  
     三角行列 2-119  
     対称行列 2-108  
   積  
     一般行列 2-95  
 行列の 1 次元下部構造 B-2  
 行列の逆転  
   一般行列  
     LAPACK 3-146  
     ScaLAPACK 6-64  
   エルミート行列 3-155  
     圧縮格納 3-159  
   エルミート正定値行列  
     LAPACK 3-149  
     ScaLAPACK 6-66  
     圧縮格納 3-151  
   三角行列 3-161  
     圧縮格納 3-163  
   三角分散行列 6-68  
   対称行列 3-153  
     圧縮格納 3-157  
   対称正定値行列  
     LAPACK 3-149  
     ScaLAPACK 6-66  
     圧縮格納 3-151  
 行列の行または列のピボット演算 7-63  
 行列の順序変更 A-4  
 行列の引数 B-4  
   行数 B-6  
   転置パラメータ B-7  
   リーディング・ディメンジョン B-6  
   例 B-7  
   列主体の順序 B-2, B-6  
   列数 B-6  
 行列の平衡化 4-207  
 行列の方程式  
    $AX = B$  2-122, 3-7, 3-35, 6-5, 6-22

行列ブロック  
 QR 因子分解  
   ピボット演算 5-149  
 行列 - ベクトル演算  
   階数 1 の更新 2-38, 2-40, 2-42  
     エルミート行列 2-49  
     圧縮格納 2-56  
   実対称行列 2-73  
     圧縮格納 2-66  
   複素対称行列 5-21  
     圧縮格納 5-17  
   階数 2 の更新  
     エルミート行列 2-51  
     圧縮格納 2-58  
   対称行列 2-75  
     圧縮格納 2-68  
   積 2-32, 2-35  
     エルミート行列 2-47  
     圧縮格納 2-54  
     帯格納 2-44  
   三角行列 2-88  
     圧縮格納 2-83  
     帯格納 2-77  
   実対称行列 2-70  
     圧縮格納 2-64  
   対称行列  
     帯格納 2-61  
   複素対称行列 5-19  
     圧縮格納 5-15



区間ソルバルーチン  
   ?gegas 12-5  
   ?gegss 12-10  
   ?gehbs 12-12  
   ?gehss 12-7  
   ?gekws 12-8  
   ?gemip 12-23  
   ?gepps 12-13  
   ?gerbr 12-19  
   ?gesvr 12-20  
   ?geszi 12-18  
   ?trtri 12-16  
   ?trtrs 12-3  
 グローバル配列 6-2

## け

計算ルーチン 4-5

検索

最小絶対値を持つベクトルの成分 2-29

最大絶対値を持つベクトルの成分 2-28

実数部分が最大絶対値を持つ成分とそのグローバル・インデックス 7-8

実数部分が最大絶対値を持つベクトルの成分のインデックス 7-7

## こ

合計

スパースベクトルとフル格納ベクトル 2-128

ベクトル 2-7

ベクトル成分の大きさ 2-6

更新

階数 1

一般行列 2-38

エルミート行列 2-49

圧縮格納 2-56

実対称行列 2-73

圧縮格納 2-66

複素対称行列 5-21

圧縮格納 5-17

階数 1、共役

一般行列 2-40

階数 1、非共役

一般行列 2-42

階数 2

エルミート行列 2-51

圧縮格納 2-58

対称行列 2-75

圧縮格納 2-68

階数 2k

エルミート行列 2-105

対称行列 2-115

階数 n

エルミート行列 2-102

対称行列 2-111

構成パラメータ、DFTI 11-3

高速フーリエ変換

C インターフェイス 11-69

データ格納の型 11-68

データ構造の要件 11-69

ルーチン

?fft1d 11-71, 11-75, 11-79

?fft1dc 11-72, 11-76, 11-81

?fft2d 11-84, 11-87, 11-94

?fft2dc 11-85, 11-90, 11-95

高度ドライバ 6-4

コード例

BLAS レベル 1 の関数 C-1

BLAS レベル 1 のルーチン C-2

BLAS レベル 2 のルーチン C-3

BLAS レベル 3 のルーチン C-4

コピー

行列

2 次元

LAPACK 5-47

ScaLAPACK 7-46

グローバル並列 7-44

分散 7-43

ローカル複製 7-44

ベクトル 2-9

コミュニケーション・サブプログラム 6-1

固有値問題 4-1

一般行列 4-189, 4-243, 6-179

エルミート行列 4-107

対称行列 4-107

対称三重対角行列 7-177, 7-188

汎用形式 4-172

固有値。固有値問題を参照

固有ペア、ソート 7-175

固有ベクトル。固有値問題を参照

コレスキー因子分解

エルミート正定値行列 3-15, 3-205, 6-13, 6-233

圧縮格納 3-18, 3-212

帯格納 3-20, 3-48, 3-220, 6-15, 6-29

対称正定値行列 3-15, 3-205, 6-13, 6-233

圧縮格納 3-18, 3-212

帯格納 3-20, 3-48, 3-220, 6-15, 6-29

分割 4-187

## さ

最小二乗問題 4-1

最大絶対値の成分

一般矩形行列 7-54

一般三重対角行列 5-121

一般正方行列 5-119

- 上 Hessenberg 行列 5-122, 7-56
- エルミート帯行列 5-125
- 三角帯行列 5-135
- 三角行列 5-138, 7-61
  - 圧縮格納 5-137
- 実対称行列 5-132, 7-58
- 実対称三重対角行列 5-130
- 台形行列 5-138
- 対称帯行列 5-124
- 対称行列
  - 圧縮格納 5-127
- 複素エルミート行列 5-133, 7-58
  - 圧縮格納 5-129
- 複素エルミート三重対角行列 5-130
- 複素対称行列 5-132
- 三角帯行列
  - 1- ノルムの値 5-135
  - Frobenius ノルムの値 5-135
  - 最大絶対値の成分 5-135
  - 無限ノルムの値 5-135
- 三角帯方程式
  - ScaLAPACK 7-152
- 三角帯連立方程式
  - LAPACK 5-257
- 三角行列 4-189, 4-243
  - 1- ノルムの値
    - LAPACK 5-138
    - ScaLAPACK 7-61
  - Frobenius ノルムの値
    - LAPACK 5-138
    - ScaLAPACK 7-61
  - ScaLAPACK 6-179
- 行列の逆転 3-161
  - LAPACK 5-314
  - ScaLAPACK 7-169
  - 圧縮格納 3-163
- 行列 - ベクトルの積 2-88
  - 圧縮格納 2-83
  - 帯格納 2-77
- 最大絶対値の成分
  - LAPACK 5-138
  - ScaLAPACK 7-61
- 条件数の推定 3-96
  - 圧縮格納 3-99
  - 帯格納 3-101
- スカラ - 行列 - 行列の積 2-119
- 積
  - LAPACK 5-273, 5-275
  - ScaLAPACK 7-122, 7-124
  - 非ブロック化アルゴリズム 5-273
  - ブロック化アルゴリズム 5-275, 7-124
- 無限ノルムの値
  - LAPACK 5-138
  - ScaLAPACK 7-61
- 隣接対角ブロックの交換 5-308
- 連立 1 次方程式の解の算出 2-91, 3-63
  - ScaLAPACK 6-40
  - 圧縮格納 2-86, 3-65
  - 帯格納 2-80, 3-68
- 三角分解
  - 一般行列 3-9, 6-6
  - エルミート行列 3-26
    - 圧縮格納 3-32
  - エルミート正定値行列 3-15, 6-13
    - 圧縮格納 3-18
    - 帯格納 3-20, 6-15
    - 三重対角行列 3-22, 6-17
  - 帯行列 3-11, 6-8, 6-10, 7-10, 7-181
  - 三重対角行列
    - LAPACK 3-13
    - ScaLAPACK 7-183
  - 対称行列 3-23
    - 圧縮格納 3-29
  - 対称正定値行列 3-15, 6-13
    - 圧縮格納 3-18
    - 帯格納 3-20, 6-15
    - 三重対角行列 3-22, 6-17
  - 帯行列 7-13
- 三角分散行列
  - 行列の逆転 6-68
- 三角連立方程式 7-156
  - スケール係数で解く
    - LAPACK 5-267
    - ScaLAPACK 7-117
- 三重対角行列 4-107
  - 条件数の推定 3-76
  - 連立 1 次方程式の解の算出 3-40, 3-51
    - ScaLAPACK 7-184
- 三重連立方程式 5-298

## し

次元 B-1  
字体規則 1-9  
実行列  
    QR 因子分解  
        ピボット演算 5-151  
実数演算での複素除算 5-48  
実数準三角連立方程式 5-158  
実対称行列  
    1- ノルムの値 5-132  
    Frobenius ノルムの値 5-132  
    最大絶対値の成分 5-132  
    無限ノルムの値 5-132  
実対称三重対角行列  
    1- ノルムの値 5-130  
    Frobenius ノルムの値 5-130  
    最大絶対値の成分 5-130  
    無限ノルムの値 5-130  
準三角行列  
    LAPACK 4-189, 4-243  
    ScaLAPACK 6-179  
順方向または逆方向 FFT 11-71, 11-72, 11-84, 11-85  
準乱数 10-1  
条件数  
    エルミート行列 3-90  
        圧縮格納 3-94  
    エルミート正定値行列 3-78  
        圧縮格納 3-81  
        帯格納 3-83  
        三重対角 3-85  
    帯行列 3-73  
    三角行列 3-96  
        圧縮格納 3-99  
        帯格納 3-101  
    三重対角行列 3-76  
    対称行列 3-87, 4-169  
        圧縮格納 3-92  
    対称正定値行列 3-78  
        圧縮格納 3-81  
        帯格納 3-83  
        三重対角 3-85  
条件数推定  
    一般行列  
        LAPACK 3-71  
        ScaLAPACK 6-42, 6-45, 6-48  
乗算合同生成器 10-4

処理ノード 10-2  
シルベスター方程式 4-240

## す

スカラ - 行列 - 行列の積 2-99  
    一般行列 2-95  
    三角行列 2-119  
    対称行列 2-108  
スカラ - 行列の積 2-95, 2-99, 2-108  
スケーリング  
    一般矩形行列 7-66  
    対称 / エルミート行列 7-69  
スケール係数  
    一般矩形分散行列 6-70  
    エルミート正定値分散行列 6-72  
    対称正定値分散行列 6-72  
ステータス確認  
    DFTI 11-5  
        クラスタ DFTI 11-53  
ストライド 増分 を参照  
ストリーム 10-10  
ストリーム・ディスクリプタ 10-2  
スパース BLAS レベル 1 2-126  
    データ型 2-127  
    命名規則 2-127  
スパース BLAS レベル 1 のルーチンと関数 2-127  
    ?axpyi 2-128  
    ?dotci 2-131  
    ?doti 2-130  
    ?dotui 2-133  
    ?gthr 2-134  
    ?gthrz 2-136  
    ?roti 2-137  
    ?sctr 2-139  
スパース BLAS レベル 2 2-141  
    命名規則 2-141  
スパース BLAS レベル 2 のルーチン  
    mkl\_dcoogemv 2-161  
    mkl\_dcoomv 2-158  
    mkl\_dcoosv 2-180  
    mkl\_dcoosymv 2-163  
    mkl\_dcootrsv 2-182  
    mkl\_dcscmv 2-156  
    mkl\_dcscsv 2-178  
    mkl\_dcsrgemv 2-152

mkl\_dcsrsv 2-149  
 mkl\_dcscrsv 2-173  
 mkl\_dcscrsvmv 2-154  
 mkl\_dcscrsv 2-175  
 mkl\_ddiagemv 2-167  
 mkl\_ddiamv 2-165  
 mkl\_ddiasv 2-184  
 mkl\_ddiasymv 2-169  
 mkl\_ddiatrsv 2-187  
 mkl\_dskymv 2-171  
 mkl\_dskysv 2-189  
 スパース BLAS レベル 3 2-141  
   命名規則 2-141  
 スパース BLAS レベル 3 のルーチン  
   mkl\_dcoomm 2-196  
   mkl\_dcoosm 2-210  
   mkl\_dcscmm 2-194  
   mkl\_dcscsm 2-207  
   mkl\_dcscrmm 2-191  
   mkl\_dcscrsm 2-204  
   mkl\_ddiamm 2-199  
   mkl\_ddiasm 2-212  
   mkl\_dskymm 2-202  
   mkl\_dskysm 2-215  
 スパース行列 2-141, 8-1  
 スパースソルバ  
   直接法スパース・ソルバ・インターフェイス  
     dss\_create 8-18  
     dss\_define\_structure 8-19  
     dss\_delete 8-24  
     dss\_factor\_real、dss\_factor\_complex 8-21  
     dss\_reorder 8-20  
     dss\_solve\_real、dss\_solve\_complex 8-23  
     dss\_statistics 8-25  
     mkl\_cvt\_to\_null\_terminated\_str 8-28  
   反復法スパース・ソルバ・インターフェイス  
     dcg 8-39  
     dcg\_check 8-38  
     dcg\_get 8-41  
     dcg\_init 8-37  
 スパースベクトル 2-126  
   BLAS レベル 1 のルーチンへの引渡し 2-127  
   Givens 回転 2-137  
   norm 2-127  
   scaling 2-127  
   圧縮形式 2-126

圧縮形式への変換 2-134, 2-136  
 実ドット積 2-130  
 追加とスケーリング 2-128  
 複素ドット積、共役 2-131  
 複素ドット積、非共役 2-133  
 フル格納形式 2-126  
 フル格納形式への変換 2-139  
 スパースベクトルの圧縮格納形式への変換 2-134  
   元のベクトルへのゼロの書き込み 2-136  
 スパースベクトルの成分の圧縮形式への集積 2-134  
   これらの成分へのゼロの書き込み 2-136

## せ

生成方法 10-2  
 正方行列  
   1- ノルムの推定  
     LAPACK 5-45  
     ScaLAPACK 7-39  
 精密モード  
   VML 9-2  
 積  
   行列 - ベクトル  
     一般行列 2-35  
     帯格納 2-32  
   エルミート行列 2-47  
     圧縮格納 2-54  
     帯格納 2-44  
   三角行列 2-88  
     圧縮格納 2-83  
     帯格納 2-77  
   実対称行列 2-70  
     圧縮格納 2-64  
   対称行列  
     帯格納 2-61  
     複素対称行列 5-19  
     圧縮格納 5-15  
   スカラ - 行列  
     一般行列 2-95  
     エルミート行列 2-99  
   スカラ - 行列 - 行列  
     一般行列 2-95  
     エルミート行列 2-99  
     三角行列 2-119  
     対称行列 2-108  
   ベクトル - スカラ 2-25



積。ドット積を参照  
線形合同生成器 10-3

## そ

相関関数

- ?CorrExec 10-122
- ?CorrExec1D 10-124
- ?CorrExecX 10-126
- ?CorrExecX1Ds 10-128
- ?CorrNewTask 10-104
- ?CorrNewTask1D 10-106
- ?CorrNewTaskX 10-108
- ?CorrNewTaskX1D 10-111
- ?CorrNewTaskX1D 10-111
- CorrSetDecimation 10-119
- CorrSetInternalPrecision 10-116
- CorrSetMode 10-115
- CorrSetStart 10-118

増分 B-1

ソート

- 数の昇順 / 降順
  - LAPACK 5-245
  - ScaLAPACK 7-176

- 固有ペア 7-175

ソルバ

- スパース 8-1
- 直接法 A-2
- 反復法 A-1

## た

対応プラットフォーム 1-7

対称行列

- 三重対角形式への縮退 7-165

対角成分

- LAPACK 5-233
- ScaLAPACK 7-105

対角優位帯行列

- 連立 1 次方程式の解の算出 6-37

対角優位三重対角行列

- 連立 1 次方程式の解の算出 6-34

台形行列

- 1- ノルムの値 5-138
- Frobenius ノルムの値 5-138
- RZ 因子分解

LAPACK 4-62

ScaLAPACK 6-134

最大絶対値の成分 5-138

三角形式への縮退 7-119

無限ノルムの値 5-138

対称帯行列

- 1- ノルムの値 5-124

Frobenius ノルムの値 5-124

最大絶対値の成分 5-124

無限ノルムの値 5-124

対称行列 4-107, 4-172

Bunch-Kaufman 因子分解 3-23

- 圧縮格納 3-29

階数 1 の更新 2-73, 5-21

- 圧縮格納 2-66, 5-17

階数 2 の更新 2-75

- 圧縮格納 2-68

階数 2k の更新 2-115

階数 n の更新 2-111

行列の逆転 3-153

- 圧縮格納 3-157

行列 - ベクトルの積 2-70, 5-19

- 圧縮格納 2-64, 5-15

- 帯格納 2-61

固有値と固有ベクトル 6-249, 6-252

三重対角形式への縮退 5-302

- LAPACK 5-264

- ScaLAPACK 7-113

条件数の推定 3-87, 4-169

- 圧縮格納 3-92

スカラ - 行列 - 行列の積 2-108

スケーリング 7-69

汎用固有値問題 4-172

標準形式への縮退

- LAPACK 5-300

- ScaLAPACK 7-162

連立 1 次方程式の解の算出 3-53

- 圧縮格納 3-58

対称構造式 A-10

対称正定値帯行列

- コレスキー因子分解 5-295

対称正定値行列

- 条件数の推定

- 圧縮格納 3-81

行列の逆転 3-149

- 圧縮格納 3-151

コレスキー因子分解  
   LAPACK 3-15, 5-296  
   ScaLAPACK 6-13, 7-159  
   圧縮格納 3-18  
   帯格納 3-20, 6-15  
 条件数の推定 3-78  
   帯格納 3-83  
   三重対角行列 3-85  
 連立 1 次方程式の解の算出  
   LAPACK 3-43  
   ScaLAPACK 6-27  
   圧縮格納 3-45  
   帯格納 3-48, 6-29  
 対称正定値三重対角行列  
   連立 1 次方程式の解の算出 6-31  
 対称正定値分散行列  
   行列の逆転 6-66  
   スケール係数の計算 6-72  
   平衡化 6-72  
 対称不定値行列  
   対角ピボット演算法での因子分解 5-304  
 畳み込み関数  
   ?ConvExec 10-122  
   ?ConvExec1D 10-124  
   ?ConvExecX 10-126  
   ?ConvExecX1D 10-128  
   ?ConvNewTask 10-104  
   ?ConvNewTask1D 10-106  
   ?ConvNewTaskX 10-108  
   ?ConvNewTaskX1D 10-111  
   ConvCopyTask 10-131  
   ConvDeleteTask 10-130  
   ConvSetDecimation 10-119  
   ConvSetInternalPrecision 10-116  
   ConvSetMode 10-115  
   ConvSetStart 10-118  
   CorrCopyTask 10-131  
   CorrDeleteTask 10-130  
 単一成分行列 7-171

## ち

小さい劣対角成分 7-107  
 置換行列 A-3  
 直接法スパースソルバ (DSS) インターフェイス・ルーチン 8-14

直交行列 4-78, 4-107, 4-189, 4-243, 6-179, 6-193  
 LQ 因子分解から  
   LAPACK 5-279  
   ScaLAPACK 7-131  
 QL 因子分解から  
   LAPACK 5-276, 5-282  
   ScaLAPACK 7-126, 7-136  
 QR 因子分解から  
   LAPACK 5-278  
   ScaLAPACK 7-129  
 RQ 因子分解から  
   LAPACK 5-281  
   ScaLAPACK 7-134

## て

ディスクリプタ構成  
   DFTI 11-4  
   クラスタ DFTI 11-52  
 ディスクリプタ操作  
   DFTI 11-4  
   クラスタ DFTI 11-52  
 データ型  
   VML 9-2  
   省略表記 1-9  
 転置パラメータ B-7  
 点の回転  
   変形面 2-21  
   面 2-17

## と

特異値分解  
   LAPACK 4-78, 4-426  
   LAPACK ルーチン、特異値分解を参照  
   ScaLAPACK 6-193, 6-267  
 ドット積  
   実ベクトル 2-10  
   実ベクトル ( 拡張精度 ) 2-12  
   スパース実ベクトル 2-130  
   スパース複素ベクトル 2-133  
   スパース複素ベクトル、共役 2-131  
   複素ベクトル、共役 2-13  
   複素ベクトル、非共役 2-15  
 ドライバ  
   簡易 6-4

高度 6-4  
ドライバルーチン 4-294, 3-177

## な

長さ。次元を参照

## に

二重対角行列  
LAPACK 4-78  
ScaLAPACK 6-193  
二乗和  
更新  
LAPACK 5-246  
ScaLAPACK 7-108  
二分法 5-177

## は

配列ディスクリプタ 6-2  
パラメータ  
Givens 回転 2-19  
変形 Givens 変換 2-23  
パラメータの分割、区間方程式 12-13  
汎用 Schur 因子分解 5-94, 5-163, 5-174, 5-176  
汎用固有値問題 4-172  
LAPACK ルーチン、汎用固有値問題を参照  
実対称問題 4-173, 5-300, 5-302, 6-208, 7-162, 7-165  
圧縮格納 4-177  
帯格納 4-182  
複素エルミート問題 4-175, 5-300, 5-302, 6-210, 7-162, 7-165  
圧縮格納 4-179  
帯格納 4-184  
汎用固有値問題の縮退  
LAPACK 4-173  
ScaLAPACK 6-208

## ひ

引数  
行列 B-4  
スパースベクトル 2-126

ベクトル B-1  
非対角成分  
LAPACK 5-233  
ScaLAPACK 7-105  
初期化 7-105

## ふ

フィルイン、スパース行列 A-4  
複素エルミート行列  
1- ノルムの値  
LAPACK 5-133  
ScaLAPACK 7-58  
Frobenius ノルムの値  
LAPACK 5-133  
ScaLAPACK 7-58  
最大絶対値の成分  
LAPACK 5-133  
ScaLAPACK 7-58  
対角ピボット演算法での因子分解 5-306  
無限ノルムの値  
LAPACK 5-133  
ScaLAPACK 7-58  
複素エルミート三重対角行列  
1- ノルムの値 5-130  
Frobenius ノルムの値 5-130  
最大絶対値の成分 5-130  
無限ノルムの値 5-130  
複素対称行列  
1- ノルムの値 5-132  
Frobenius ノルムの値 5-132  
最大絶対値の成分 5-132  
無限ノルムの値 5-132  
複素ベクトル  
共役  
LAPACK 5-10  
ScaLAPACK 7-6  
真の絶対値を使用した 1- ノルム  
LAPACK 5-23  
ScaLAPACK 7-9  
複素ベクトルの共役  
LAPACK 5-10  
ScaLAPACK 7-6  
負の固有値 7-53  
プリコンディショニング、区間方程式 12-24  
フル格納体系 B-4

フル格納ベクトル 2-126  
付録テンプレート A-16  
プロセスグリッド 6-2  
ブロック分割法 10-9  
ブロック・サイクリック分割 6-2  
ブロック・リフレクタ  
  一般行列  
    LAPACK 5-191  
    ScaLAPACK 7-92  
  一般矩形行列  
    ScaLAPACK 7-77  
  一般正方行列  
    LAPACK 5-166  
  三角係数  
    LAPACK 5-170, 5-193  
    ScaLAPACK 7-85, 7-99  
分割コレスキー因子分解 (帯行列) 4-187  
分散メモリ演算 6-1



並列化対応直接法ソルバ (Pardiso) 8-1

ベクトル  
  Givens 回転 2-19  
  交換 2-26  
  コピー 2-9  
  最小絶対値を持つ成分 2-29  
  最大絶対値を持つ成分 2-28  
  実数部分が最大絶対値を持つ成分とそのイン  
    デックス 7-8  
  実数部分が最大絶対値を持つ成分のインデッ  
    クス 7-7  
  スパースベクトル 2-127  
  点の回転 2-17  
  ドット積  
    実ベクトル 2-10  
    複素ベクトル 2-15  
    複素ベクトル、共役 2-13  
  ベクトル - スカラの積 2-25  
  ベクトル成分の大きさの追加 2-6  
  ベクトルの合計 2-7  
  ベクトルの線形組み合わせ 2-7  
  変形 Givens 変換パラメータ 2-23  
  変形面における点の回転 2-21  
  ユークリッド・ノルム 2-16  
ベクトル pack 関数 9-39

ベクトル unpack 関数 9-41  
ベクトル数学関数 9-7  
  10 進対数 9-20  
  power (定数) 9-16  
  逆正接 9-27  
  逆双曲正弦 9-33  
  逆双曲正接 9-34  
  逆双曲余弦 9-32  
  逆転 9-9  
  逆平方根 9-12  
  逆余弦 9-25  
  逆立方根 9-14  
  誤差関数値 9-35  
  指数 9-18  
  自然対数 9-19  
  四分円逆正接 9-28  
  除算 9-10  
  正弦と余弦 9-23  
  正接 9-24  
  双曲正弦 9-30  
  双曲正接 9-31  
  双曲余弦 9-29  
  相補誤差関数値 9-37  
  平方根 9-11  
  立方根 9-13  
  累乗 9-15  
  正弦 9-22  
  逆正弦 9-26  
  余弦 9-21  
ベクトル - スカラの積 2-25  
  スパースベクトル 2-128  
ベクトル成分の大きさの追加 2-6  
ベクトル成分の最小絶対値 2-29  
ベクトル成分の絶対値  
  最小値 2-29  
  最大値 2-28  
ベクトル統計関数  
  Bernoulli 10-75  
  Beta 10-68  
  Binomial 10-79  
  Cauchy 10-57  
  CopyStream 10-25  
  CopyStreamState 10-26  
  DeleteStream 10-24  
  dNewAbstractStream 10-19  
  Exponential 10-50

Gamma 10-66  
Gaussian 10-43  
GaussianMV 10-45  
Geometric 10-77  
GetBrngProperties 10-92  
GetNumRegBrngs 10-38  
GetStreamStateBrng 10-37  
Gumbel 10-64  
Hypergeometric 10-81  
iNewAbstractStream 10-17  
Laplace 10-52  
LeapfrogStream 10-31  
LoadStreamF 10-29  
Lognormal 10-61  
NegBinomial 10-87  
NewStream 10-14  
NewStreamEx 10-15  
Poisson 10-83  
PoissonV 10-85  
Rayleigh 10-59  
RegisterBrng 10-91  
SaveStreamF 10-28  
SkipAheadStream 10-34  
sNewAbstractStream 10-22  
Uniform ( 離散 ) 10-71  
Uniform ( 連続 ) 10-40  
UniformBits 10-73  
Weibull 10-54  
ベクトルの共役 5-10  
ベクトルの交換 2-26  
ベクトルの乗算  
    LAPACK 5-299  
    ScaLAPACK 7-161  
ベクトルの線形組み合わせ 2-7  
ベクトルの引数 B-1  
    行列の 1 次元下部構造 B-2  
    スパースベクトル 2-126  
    デフォルト B-2  
    長さ B-1  
    配列の次元 B-1  
    例 B-2  
ベクトル・インデックス 9-6

## ほ

補助ルーチン

LAPACK 5-1  
ScaLAPACK 7-1

## ま

マシン・パラメータ  
    ScaLAPACK 7-192  
    LAPACK 5-322

## む

無限ノルムの値  
    一般矩形行列 7-54  
    一般三重対角行列 5-121  
    一般正方行列 5-119  
    上 Hessenberg 行列 5-122, 7-56  
    エルミート帯行列 5-125  
    三角帯行列 5-135  
    三角行列 5-138, 7-61  
        圧縮格納 5-137  
    実対称行列 5-132, 7-58  
    実対称三重対角行列 5-130  
    台形行列 5-138  
    対称帯行列 5-124  
    対称行列  
        圧縮格納 5-127  
    複素エルミート行列 5-133, 7-58  
        圧縮格納 5-129  
    複素エルミート三重対角行列 5-130  
    複素対称行列 5-132

## め

命名規則 1-9  
    BLAS 2-2  
    LAPACK 3-2, 4-3, 6-3  
    VML 9-2  
    スパース BLAS レベル 1 2-127  
    スパース BLAS レベル 2 2-141  
    スパース BLAS レベル 3 2-141

## ゆ

ユークリッド・ノルム  
    ベクトルの 2-16

ユーザータイム 5-327  
ユニタリ行列 4-78, 4-107, 4-189, 4-243  
  LQ 因子分解から  
    LAPACK 5-279  
    ScaLAPACK 7-131  
  QL 因子分解から  
    LAPACK 5-276, 5-282  
    ScaLAPACK 7-126, 7-136  
  QR 因子分解から  
    LAPACK 5-278  
    ScaLAPACK 7-129  
  RQ 因子分解から  
    LAPACK 5-281  
    ScaLAPACK 7-134  
  ScaLAPACK 6-179, 6-193

## ら

乱数生成器 10-1  
ランダム・ストリーム 10-10

## り

リーディング・ディメンション B-6  
リープフロッグ法 10-9  
離散フーリエ変換  
  CommitDescriptor 11-10  
  CommitDescriptorDM 11-55  
  ComputeBackward 11-16  
  ComputeBackwardDM 11-60  
  ComputeForward 11-14  
  ComputeForwardDM 11-58  
  CopyDescriptor 11-11  
  CreateDescriptor 11-8  
  CreateDescriptorDM 11-53  
  ErrorClass 11-5  
  ErrorMessage 11-7  
  FreeDescriptor 11-13  
  FreeDescriptorDM 11-57  
  GetValue 11-21  
  GetValueDM 11-65  
  SetValue 11-19  
  SetValueDM 11-63  
離散分布生成器 10-40  
リバース・コミュニケーション・インターフェイス  
8-30

隣接対角ブロックの交換 5-84, 5-308

## る

ルーチン命名規則  
  BLAS 2-2  
    スパース BLAS レベル 1 2-127  
    スパース BLAS レベル 2 2-141  
    スパース BLAS レベル 3 2-141

## れ

連続分布生成器 9-7, 10-39  
連立 1 次方程式  
  三角行列 2-91  
    圧縮格納 2-86  
    帯格納 2-80  
連立 1 次方程式。1 次方程式を参照

