インテル® マス・カーネル・ライブラリ

リファレンス・マニュアル

Copyright © 1994-2004, Intel Corporation.
無断での引用、転載を禁じます。
資料番号 : 630813-6004J

Web サイト : http://intel.co.jp/jp/developer/ (日本語)
http://developer.intel.com (英語)

目次

第 1 章　概要
本ソフトウェアについて 1-1
技術サポート 1-2
BLAS ルーチン 1-2
スパース BLAS ルーチン 1-3
高速フーリエ変換 1-3
LAPACK ルーチン 1-3
VML 関数 1-3
VSL 関数 1-4
DFT 関数 1-4
性能の改善 1-4
並列化 1-5
対応プラットフォーム 1-5
本書について 1-6
本書の対象読者 1-6
本書の構成 1-6
表記の規則 1-8
ルーチン名の省略表記 1-8
字体の規則 1-8
参考文献 1-9

第 2 章　BLAS ルーチンとスパース BLAS ルーチン
ルーチン名の規則 2-2
マトリックスの格納形式 2-3
BLAS レベル 1 のルーチンと関数 2-4
?asum 2-5
?axpy 2-6
?copy 2-7
?dot 2-8
?sdot 2-9
?dotc 2-10
?dotu 2-11
?nrm2 2-12
?rot 2-13
?rotg 2-15

?rotm 2-16
?rotmg 2-18
?scal 2-20
?swap 2-21
i?amax 2-22
i?amin 2-23
BLAS レベル 2 のルーチン 2-25
?gbmv 2-26
?gemv 2-29
?ger 2-32
?gerc 2-33
?geru 2-35
?hbmv 2-37
?hemv 2-40
?her 2-42
?her2 2-44
?hpmv 2-46
?hpr 2-49
?hpr2 2-51
?sbmv 2-53
?spmv 2-56
?spr 2-58
?spr2 2-60
?symv 2-62
?syr 2-64
?syr2 2-66
?tbmv 2-68
?tbsv 2-71
?tpmv 2-74
?tpsv 2-77
?trmv 2-79
?trsv 2-81
BLAS レベル 3 のルーチン 2-84
インテル® MKL のシンメトリック・マルチプロセシング・バージョン 2-84
?gemm 2-85
?hemm 2-88
?herk 2-91
?her2k 2-94

?symm 2-98
?syrk 2-102
?syr2k 2-105
?trmm 2-109
?trsm 2-112
スパース BLAS のルーチンと関数 2-116
スパース BLAS におけるベクトル引数 2-116
スパース BLAS における命名規則 2-117
スパース BLAS におけるルーチンとデータ・タイプ 2-117
スパース・ベクトルで使用できる BLAS ルーチン 2-118
?axpyi 2-118
?doti 2-120
?dotci 2-121
?dotui 2-122
?gthr 2-123
?gthrz 2-124
?roti 2-125
?sctr 2-126

第 3 章　高速フーリエ変換

1 次元 FFT 3-1
データ格納の型 3-2
データ構造の要件 3-3
複素数から複素数への 1 次元 FFT 3-3
cfft1d/zfft1d 3-4
cfft1dc/zfft1dc 3-5
実数から複素数への 1 次元 FFT 3-7
scfft1d/dzfft1d 3-8
scfft1dc/dzfft1dc 3-10
複素数から実数への 1 次元 FFT 3-12
csfft1d/zdfft1d 3-13
csfft1dc/zdfft1dc 3-15
2 次元 FFT 3-17
複素数から複素数への 2 次元 FFT 3-18
cfft2d/zfft2d 3-19
cfft2dc/zfft2dc 3-20
実数から複素数への 2 次元 FFT 3-21
scfft2d/dzfft2d 3-22
scfft2dc/dzfft2dc 3-24

複素数から実数への 2 次元 FFT 3-27
csfft2d/zdfft2d 3-28
csfft2dc/zdfft2dc 3-29

第 4 章 LAPACK ルーチン : 1 次方程式
ルーチンや規則 4-2
マトリックスの格納形式 4-3
数学的表記 4-3
誤差の分析 4-4
計算ルーチン 4-5
マトリックスの因子分解用のルーチン 4-7
?getrf 4-7
?gbtrf 4-10
?gttrf 4-12
?potrf 4-14
?pptrf 4-16
?pbtrf 4-18
?pttrf 4-20
?sytrf 4-22
?hetrf 4-25
?sptrf 4-28
?hptrf 4-31
連立 1 次方程式を解くためのルーチン 4-33
?getrs 4-34
?gbtrs 4-36
?gttrs 4-38
?potrs 4-41
?pptrs 4-43
?pbtrs 4-45
?pttrs 4-47
?sytrs 4-49
?hetrs 4-51
?sptrs 4-53
?hptrs 4-55
?trtrs 4-57
?tptrs 4-59
?tbtrs 4-61
条件数を推定するためのルーチン 4-63
?gecon 4-63

?gbcon 4-65
?gtcon 4-67
?pocon 4-70
?ppcon 4-72
?pbcon 4-74
?ptcon 4-76
?sycon 4-78
?hecon 4-80
?spcon 4-82
?hpcon 4-84
?trcon 4-86
?tpcon 4-88
?tbcon 4-90
解の精度の改善と誤差の推定 4-92
?gerfs 4-92
?gbrfs 4-95
?gtrfs 4-98
?porfs 4-101
?pprfs 4-104
?pbrfs 4-107
?ptrfs 4-110
?syrfs 4-113
?herfs 4-116
?sprfs 4-119
?hprfs 4-122
?trrfs 4-124
?tprfs 4-127
?tbrfs 4-130
マトリックスの反転用のルーチン 4-133
?getri 4-133
?potri 4-135
?pptri 4-137
?sytri 4-139
?hetri 4-141
?sptri 4-143
?hptri 4-145
?trtri 4-147
?tptri 4-148

マトリックスの平衡化 4-150
?geequ 4-150
?gbequ 4-153
?poequ 4-155
?ppequ 4-157
?pbequ 4-159
ドライバ・ルーチン 4-161
?gesv 4-162
?gesvx 4-163
?gbsv 4-170
?gbsvx 4-172
?gtsv 4-179
?gtsvx 4-181
?posv 4-186
?posvx 4-188
?ppsv 4-193
?ppsvx 4-195
?pbsv 4-200
?pbsvx 4-202
?ptsv 4-207
?ptsvx 4-209
?sysv 4-212
?sysvx 4-215
?hesvx 4-220
?hesv 4-224
?spsv 4-227
?spsvx 4-230
?hpsvx 4-235
?hpsv 4-239

第 5 章 LAPACK ルーチン : 最小二乗問題および固有値問題
ルーチン名の規則 5-4
マトリックスの格納形式 5-5
数学表記 5-5
計算ルーチン 5-6
直交因子分解 5-6
?geqrf 5-8
?geqpf 5-11
?geqp3 5-14

?orgqr 5-17
?ormqr 5-19
?ungqr 5-21
?unmqr 5-23
?gelqf 5-25
?orglq 5-28
?ormlq 5-30
?unglq 5-32
?unmlq 5-34
?geqlf 5-36
?orgql 5-38
?ungql 5-40
?ormql 5-42
?unmql 5-45
?gerqf 5-48
?orgrq 5-50
?ungrq 5-52
?ormrq 5-54
?unmrq 5-57
?tzrzf 5-60
?ormrz 5-62
?unmrz 5-65
?ggqrf 5-68
?ggrqf 5-71
特異値分解 5-74
?gebrd 5-76
?gbbrd 5-79
?orgbr 5-82
?ormbr 5-85
?ungbr 5-88
?unmbr 5-91
?bdsqr 5-94
?bdsdc 5-98
対称固有値問題 5-101
?sytrd 5-105
?orgtr 5-107
?ormtr 5-109
?hetrd 5-111

?ungtr 5-113
?unmtr 5-115
?sptrd 5-117
?opgtr 5-119
?opmtr 5-120
?hptrd 5-122
?upgtr 5-124
?upmtr 5-125
?sbtrd 5-128
?hbtrd 5-130
?sterf 5-132
?steqr 5-134
?stedc 5-137
?stegr 5-141
?pteqr 5-146
?stebz 5-149
?stein 5-152
?disna 5-154
汎用対称固有値問題 5-157
?sygst 5-158
?hegst 5-160
?spgst 5-162
?hpgst 5-164
?sbgst 5-166
?hbgst 5-169
?pbstf 5-172
非対称固有値問題 5-174
?gehrd 5-178
?orghr 5-180
?ormhr 5-182
?unghr 5-185
?unmhr 5-187
?gebal 5-190
?gebak 5-193
?hseqr 5-195
?hsein 5-199
?trevc 5-205
?trsna 5-210

?trexc 5-215
?trsen 5-217
?trsyl 5-222
汎用非対称固有値問題 5-225
?gghrd 5-226
?ggbal 5-230
?ggbak 5-233
?hgeqz 5-235
?tgevc 5-242
?tgexc 5-247
?tgsen 5-250
?tgsyl 5-256
?tgsna 5-261
汎用特異値分解 5-266
?ggsvp 5-267
?tgsja 5-271
ドライバ・ルーチン 5-278
線形最小二乗 (LLS) 問題 5-278
?gels 5-279
?gelsy 5-282
?gelss 5-286
?gelsd 5-289
汎用 LLS 問題 5-293
?gglse 5-293
?ggglm 5-296
対称固有値問題 5-298
?syev 5-299
?heev 5-301
?syevd 5-303
?heevd 5-306
?syevx 5-309
?heevx 5-313
?syevr 5-317
?heevr 5-322
?spev 5-327
?hpev 5-329
?spevd 5-331
?hpevd 5-334

?spevx 5-338
?hpevx 5-342
?sbev 5-346
?hbev 5-348
?sbevd 5-350
?hbevd 5-353
?sbevx 5-357
?hbevx 5-361
?stev 5-365
?stevd 5-367
?stevx 5-370
?stevr 5-374
非対称固有値問題 5-379
?gees 5-379
?geesx 5-384
?geev 5-390
?geevx 5-394
特異値分解 5-400
?gesvd 5-400
?gesdd 5-405
?ggsvd 5-409
汎用対称固有値問題 5-415
?sygv 5-416
?hegv 5-419
?sygvd 5-422
?hegvd 5-425
?sygvx 5-429
?hegvx 5-434
?spgv 5-439
?hpgv 5-442
?spgvd 5-445
?hpgvd 5-448
?spgvx 5-452
?hpgvx 5-456
?sbgv 5-460
?hbgv 5-463
?sbgvd 5-466
?hbgvd 5-469

?sbgvx 5-473
?hbgvx 5-477
汎用非対称固有値問題 5-482
?gges 5-482
?ggesx 5-489
?ggev 5-497
?ggevx 5-502
参考文献 5-509

第 6 章 LAPACK 補助ルーチン
?lacgv 6-1
?lacrm 6-2
?lacrt 6-3
?laesy 6-4
?rot 6-6
?spmv 6-7
?spr 6-9
?symv 6-11
?syr 6-13
i?max1 6-15
ilaenv 6-16
lsame 6-18
lsamen 6-19
?sum1 6-20
?gbtf2 6-21
?gebd2 6-23
?gehd2 6-26
?gelq2 6-29
?geql2 6-31
?geqr2 6-33
?gerq2 6-35
?gesc2 6-37
?getc2 6-39
?getf2 6-40
?gtts2 6-42
?labad 6-43
?labrd 6-45
?lacon 6-48
?lacpy 6-50

?ladiv 6-51
?lae2 6-52
?laebz 6-54
?laed0 6-60
?laed1 6-64
?laed2 6-67
?laed3 6-70
?laed4 6-73
?laed5 6-74
?laed6 6-75
?laed7 6-78
?laed8 6-83
?laed9 6-87
?laeda 6-89
?laein 6-92
?laev2 6-95
?laexc 6-98
?lag2 6-100
?lags2 6-103
?lagtf 6-105
?lagtm 6-108
?lagts 6-110
?lagv2 6-113
?lahqr 6-115
?lahrd 6-118
?laic1 6-121
?laln2 6-124
?lals0 6-127
?lalsa 6-132
?lalsd 6-136
?lamch 6-139
?lamc1 6-140
?lamc2 6-141
?lamc3 6-142
?lamc4 6-143
?lamc5 6-144
?lamrg 6-145
?langb 6-146

?lange 6-148
?langt 6-149
?lanhs 6-151
?lansb 6-152
?lanhb 6-154
?lansp 6-156
?lanhp 6-158
?lanst/?lanht..... 6-159
?lansy..... 6-161
?lanhe 6-163
?lantb 6-164
?lantp 6-167
?lantr 6-169
?lanv2 6-171
?lapll..... 6-172
?lapmt 6-174
?lapy2..... 6-175
?lapy3..... 6-176
?laqgb 6-176
?laqge 6-179
?laqp2 6-181
?laqps 6-183
?laqsb 6-185
?laqsp 6-187
?laqsy..... 6-189
?laqtr 6-191
?lar1v 6-194
?lar2v 6-196
?larf 6-198
?larfb 6-200
?larfg 6-202
?larft 6-204
?larfx 6-207
?largv 6-208
?larnv 6-210
?larrb..... 6-212
?larre..... 6-214
?larrf..... 6-216

?larrv 6-218
?lartg 6-221
?lartv 6-223
?laruv 6-224
?larz 6-225
?larzb 6-227
?larzt 6-230
?las2 6-233
?lascl 6-234
?lasd0 6-236
?lasd1 6-238
?lasd2 6-241
?lasd3 6-246
?lasd4 6-249
?lasd5 6-251
?lasd6 6-252
?lasd7 6-257
?lasd8 6-262
?lasd9 6-264
?lasda 6-267
?lasdq 6-271
?lasdt 6-274
?laset 6-275
?lasq1 6-277
?lasq2 6-278
?lasq3 6-280
?lasq4 6-281
?lasq5 6-283
?lasq6 6-284
?lasr 6-286
?lasrt 6-288
?lassq 6-289
?lasv2 6-291
?laswp 6-292
?lasy2 6-294
?lasyf 6-296
?lahef 6-299
?latbs 6-301

?latdf 6-304
?latps 6-307
?latrd 6-309
?latrs 6-313
?latrz 6-317
?lauu2 6-320
?lauum 6-321
?org2l/?ung2l 6-323
?org2r/?ung2r 6-324
?orgl2/?ungl2 6-326
?orgr2/?ungr2 6-328
?orm2l/?unm2l 6-330
?orm2r/?unm2r 6-332
?orml2/?unml2 6-335
?ormr2/?unmr2 6-337
?ormr3/?unmr3 6-340
?pbtf2 6-343
?potf2 6-345
?ptts2 6-346
?rscl 6-348
?sygs2/?hegs2 6-349
?sytd2/?hetd2 6-351
?sytf2 6-354
?hetf2 6-356
?tgex2 6-358
?tgsy2 6-361
?trti2 6-366
xerbla 6-367

第 7 章　ベクトル数学関数

データ・タイプと精密モード 7-2
関数規則 7-2
　関数インターフェイス 7-3
ベクトル・インデックス方式 7-6
エラー診断 7-6
VML 数学関数 7-8
　Inv 7-10
　Div 7-11
　Sqrt 7-12

InvSqrt 7-13
Cbrt 7-15
InvCbrt 7-16
Pow 7-17
Powx 7-18
Exp 7-20
Ln 7-21
Log10 7-22
Cos 7-23
Sin 7-24
SinCos 7-25
Tan 7-27
Acos 7-28
Asin 7-29
Atan 7-30
Atan2 7-31
Cosh 7-32
Sinh 7-34
Tanh 7-35
Acosh 7-36
Asinh 7-37
Atanh 7-38
Erf 7-39
Erfc 7-41
VML Pack/Unpack 関数 7-43
Pack 7-43
Unpack 7-45
VML サービス関数 7-48
SetMode 7-48
GetMode 7-51
SetErrStatus 7-52
GetErrStatus 7-54
ClearErrStatus 7-54
SetErrorCallBack 7-55
GetErrorCallBack 7-57
ClearErrorCallBack 7-58

第 8 章　統計的分布のベクトル生成器
規則 8-2

数学的表記 8-3
命名規則 8-4
基本擬似乱数生成器 8-6
ランダム・ストリーム 8-6
データ・タイプ 8-7
サービス・サブルーチン 8-7
NewStream 8-9
NewStreamEx 8-10
DeleteStream 8-12
CopyStream 8-12
CopyStreamState 8-13
LeapfrogStream 8-15
SkipAheadStream 8-18
GetStreamStateBrng 8-21
GetNumRegBrng 8-22
擬似乱数生成器 8-23
連続分布 8-24
Uniform 8-24
Gaussian 8-26
Exponential 8-29
Laplace 8-31
Weibull 8-34
Cauchy 8-36
Rayleigh 8-39
Lognormal 8-41
Gumbel 8-44
離散分布 8-46
Uniform 8-47
UniformBits 8-49
Bernoulli 8-51
Geometric 8-53
Binomial 8-55
Hypergeometric 8-57
Poisson 8-59
NegBinomial 8-61
アドバンスト・サービス・サブルーチン 8-63
データ・タイプ 8-63
RegisterBrng 8-66
GetBrngProperties 8-67

ユーザ定義生成器のフォーマット 8-68
iBRng 8-71
sBRng 8-72
dBRng 8-73

第 9 章 アドバンスト DFT 関数
DFT の計算 9-2
DFT インターフェイス 9-9
ステータス確認関数 9-10
ErrorClass 9-11
ErrorMessage 9-13
ディスクリプタ操作 9-15
CreateDescriptor 9-15
CommitDescriptor 9-17
CopyDescriptor 9-18
FreeDescriptor 9-20
DFT ComputationDFT の計算 9-21
ComputeForward 9-21
ComputeBackward 9-23
ディスクリプタの構成 9-26
SetValue 9-31
GetValue 9-33
構成設定 9-35
変換精度 9-35
順方向変換 9-36
変換の次元と長さ 9-36
変換回数 9-37
符号と倍率 9-37
結果の配置 9-38
パックド形式 9-38
格納体系 9-39
入力と出力との距離 9-48
ストライド 9-49
初期化努力 9-52
順序指定 9-52
ワークスペース 9-54
転置 9-54

付録 A ルーチンおよび関数の引数
BLAS のベクトル引数 A-1
VML のベクトル引数 A-3

マトリックス引数 A-4

付録 B　コード例

付録 C　BLAS に対する CBLAS インターフェイス

CBLAS の引数 C-1

列挙型 C-2

レベル 1 の CBLAS C-3

レベル 2 の CBLAS C-6

レベル 3 の CBLAS C-14

スパース CBLAS C-18

用語集

索引

インテル® マス・カーネル・ライブラリ リファレンス・マニュアル

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

【輸出規制に関する告知と注意事項】

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

【資料内容に関する注意事項】

・本ドキュメントの内容を予告なしに変更することがあります。

・インテルでは、この資料に掲載された内容について、市販製品に使用した場合の保証あるいは特別な目的に合うことの保証等は、いかなる場合についてもいたしかねます。また、このドキュメント内の誤りについても責任を負いかねる場合があります。

・インテルは、インテル製品の内部回路以外の使用では責任を負いません。また、外部回路の特許についても関知いたしません。

・本書の情報はインテル製品を使用できるようにする目的でのみ記載されています。

インテルは、製品について「取引条件」で提示されている場合を除き、インテル製品の販売や使用に関して、いかなる特許または著作権の侵害をも含み、あらゆる責任を負わないものとします。

・いかなる形および方法によっても、インテルの文書による許可なく、この資料の一部またはすべてを複写することは禁じられています。

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

* その他の社名、商品名などは、一般に各社の商標または登録商標です。

© 1994-2003, Intel Corporation. 無断での引用、転載を禁じます。

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

概要

1

インテル ® マス・カーネル・ライブラリ (インテル ® MKL) では、ベクトルやマトリックスに対して広範囲にわたる演算を実行する Fortran のルーチンおよび関数を提供する。また、Fortran および C のインターフェイスを使用するベクトル数値演算関数とベクトル統計関数だけでなく、高速フーリエ変換関数と新しい離散フーリエ変換関数も含んでいる。インテル MKL はインテル ® プロセッサ用に最適化されているため、MKL を使用することにより、プログラムの性能が向上する。

本章では、マス・カーネル・ライブラリを紹介するとともに、本マニュアルの構成について説明する。

本ソフトウェアについて

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

- BLAS (Basic Linear Algebra Subprograms)
  – ベクトル演算
  – マトリックス - ベクトル演算
  – マトリックス - マトリックス演算
- スパース BLAS (スパース・ベクトルに対する基本ベクトル演算)
- 高速フーリエ変換ルーチン (FortranおよびCのインターフェイスを使用)
- 連立一次方程式を解くための LAPACK ルーチン
- 最小二乗問題、固有値ならびに特異値問題、シルベスター式を解くための LAPACK ルーチン
- 補助 LAPACK ルーチン
- ベクトル引数に対する基本的な数値演算関数を計算するための VML (Vector Mathematical Library) 関数 (Fortran および C のインターフェイスを使用)
- さまざまな種類の統計的分布に従った擬似乱数ベクトルを発生させるベクトル統計ライブラリ (VSL) 関数
- 先進の離散フーリエ変換関数 (DFT)

ライブラリの使用に関する個々の疑問は、MKL リリース・ノートを参照のこと。

技術サポート

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

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

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

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

ユーザ登録、インテルへの問い合わせ、製品サポートについては、以下のサイトを参照のこと。
http://www.intel.com/software/products/support (英語)

BLAS ルーチン

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

- 「BLAS レベル 1 のルーチンと関数」は、ベクトル・データに対して加算と減算を実行する。典型的な演算として、スケーリングやドット積などがある。
- 「BLAS レベル 2 のルーチン」は、マトリックス - ベクトルの乗算、階数 1 と階数 2 のマトリックスの更新、三角法の計算などのマトリックス - ベクトル演算を実行する。
- 「BLAS レベル 3 のルーチン」は、マトリックス - マトリックスの乗算、階数 k の更新、三角法の計算などのマトリックス - マトリックス演算を実行する。

スパース BLAS ルーチン

「スパース BLAS のルーチンと関数」は、スパース・ベクトル (成分の大半がゼロであるようなベクトル) に対する演算を行う。これらのルーチンは、BLAS レベル 1 ルーチンと同じようなベクトル演算を実行する。ただし、スパース BLAS ルーチンではベクトルの成分数が少ないのを利用するため、ベクトルのゼロ以外の成分だけを格納できる。

高速フーリエ変換

「高速フーリエ変換」(FFT) は、デジタル信号処理や画像処理で使用する。また、偏微分方程式 (PDE) の計算にも使用される。FFT は、BLAS ルーチンと組み合わせることで、プログラムを移植しやすいものにするばかりでなく、プログラムと利用可能なライブラリとの間の簡易なインターフェイスを提供する。より多くの機能を使用して使いやすくするには、第 9 章で説明する新しい DFT 関数を使用することも検討するとよい。

LAPACK ルーチン

インテル・マス・カーネル・ライブラリは、LAPACK の計算およびドライバ・ルーチンをすべて含んでいる。これらのルーチンは、実行する演算によって以下のグループに分類できる。

- 連立一次方程式を解くためのルーチンや、マトリックスの因子分解ならびに逆マトリックスの計算、条件数の推定などを実行するためのルーチン (第 4 章を参照)。
- 最小二乗問題、固有値ならびに特異値問題、シルベスター式を解くためのルーチン (第 5 章を参照)。
- 特定のサブタスク、または共通的なローレベル計算に使用される補助ルーチン (第 6 章を参照)。

VML 関数

VML 関数 (第 7 章を参照) には、実数ベクトルの引数を操作する、大量の計算を必要とする一連の基本的な数値演算関数 (べき乗、三角関数、指数関数、双曲線関数など) の高度に最適化されたコードを含む。

VSL 関数

ベクトル統計ライブラリ (VSL) 関数 (第 8 章参照) は基本的な連続分布または離散分布に対応した擬似乱数生成サブルーチン群で構成されている。VSL サブルーチンは高度に最適化された基本乱数生成器とベクトル数学関数ライブラリ VML を呼び出すことで最高レベルの性能を実現している。

DFT 関数

新規に開発された離散フーリエ変換関数 (第 9 章参照) は、DFT 計算機能の統一性を提供するとともに、多機能性と使いやすさの両立を実現する。Fortran と C のインターフェイス仕様が与えられる。従来の FFT 関数を利用したアプリケーション・プログラムは、新しい DFT インターフェイスに移行することが望ましい。

性能の改善

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

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

ライブラリで使用される主な最適化技法には、次のものがある。

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

これらの技法により、演算コードにおいてメリットを最大限に活用できる。

並列化

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

- 1 つは、プログラム内のユーザ管理スレッドを基本とし、さらにデータ分解、領域分解、制御分解、あるいはその他の並列技法に基づいて複数のスレッドに処理を分配するものである。ライブラリはスレッド・セーフとなるよう設計されているため、それぞれのスレッドで任意のインテル MKL 関数を使用できる。
- もう一つは、FFT ルーチンや BLAS レベル 3 ルーチンを使用するものである。これらのルーチンは並列化されているため、アプリケーションを変更しなくても多重処理による性能改善が得られる。BLAS レベル 3 で複数のプロセッサを使用した場合の性能は、プロセッサ数に比例して改善される。スレッドはライブラリ内で呼び出されて管理されるため、アプリケーションをスレッド・セーフに再コンパイルする必要がない (第 2 章の「BLAS レベル 2 のルーチン」を参照)。
- 性能改善に役立つもう 1 つの方法は、**機能調整した LAPACK ルーチン**の使用である。現時点では、これらのルーチンには、一般マトリックスの *QR* 因子分解、一般マトリックスと正値対称マトリックスの三角因子分解、このようなマトリックスを使用しての連立方程式の解、対称固有値問題の解を得るための単精度と倍精度のルーチンが含まれている。

BLAS レベル 3 ルーチンや LAPACK ルーチンに対して使用可能なプロセッサ数を設定する方法については、**リリース・ノート**を参照のこと。

対応プラットフォーム

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

本書について

本書では、インテル・マス・カーネル・ライブラリのルーチンについて説明する。リファレンスの各項では、4 つの基本的なデータ・タイプ (単精度実数、倍精度実数、単精度複素数、倍精度複素数) で使用するルーチンを 1 つのルーチン・グループにまとめて説明する。

それぞれのルーチン・グループの説明では、名前と機能の簡単な説明とともに、グループ内の各ルーチンで使用するデータ・タイプそれぞれについての呼び出しシーケンスを紹介する。

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

本書の対象読者

本書では、線形代数の計算に精通し、線形代数やフーリエ変換の原理や用語についての知識があるプログラマを対象にしている。

本書の構成

本書は、以下の各章と付録で構成している。

第 1 章	「概要」。インテル・マス・カーネル・ライブラリ・ソフトウェアを紹介するとともに、マニュアルの構成や表記上の規則について説明する。
第 2 章	「BLAS ルーチンと スパース BLAS ルーチン」。BLAS ならびにスパース BLAS の関数とルーチンを説明する。

第 3 章	「高速フーリエ変換」。高速フーリエ変換 (FFT) について説明する。
第 4 章	「LAPACK ルーチン : 1 次方程式」。連立一次方程式の解の算出と、これに関連する多数の計算タスク (三角因子分解、逆マトリックスの計算、マトリックスの条件数の推定など) を実行するための LAPACK ルーチンについて説明する。
第 5 章	「LAPACK ルーチン : 最小二乗問題および 固有値問題」。最小二乗問題、標準ならびに一般化された固有値問題、特異値問題、シルベスター式を解くための LAPACK ルーチンを説明する。
第 6 章	「LAPACK 補助ルーチン」。特定のサブタスク、共通のローレベル計算を実行する補助 LAPACK ルーチンについて説明する。
第 7 章	「ベクトル数学関数」。ベクトル引数に対する基本的な数値演算関数を計算するための VML 関数について説明する。
第 8 章	「統計的分布のベクトル生成器」。擬似乱数ベクトルを生成する VSL 関数について説明する。
第 9 章	「アドバンスト DFT 関数」。離散フーリエ変換を計算する新しい関数について説明する。
付録 A	「ルーチンおよび関数の引数」。BLAS ルーチンと VML 関数の主要な引数 (ベクトル引数とマトリックス引数) について説明する。
付録 B	「コード例」。BLAS の関数やルーチンを呼び出す場合のコード例を掲載する。
付録 C	「BLAS に対する CBLAS インターフェイス」。BLAS に対する C インターフェイスを掲載する。

本書には、「用語集」と「索引」も含まれる。

表記の規則

本書では、以下の表記上の規則を使用する。

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

ルーチン名の省略表記

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

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

字体の規則

次の字体の表記規則を使用する。

UPPERCASE COURIER	Fortran インターフェイスの入力パラメータと出力パラメータの説明で使用されるデータ・タイプ。たとえば、CHARACTER*1
lowercase courier	コード例 a(k+i, j) = matrix(i, j) および C インターフェイスのデータ・タイプ。たとえば、const float*。
lowercase courier mixed with UpperCase courier	C インターフェイスの関数名。たとえば、vmlSetMode。
lowercase courier italic	引数やパラメータの説明における変数。たとえば、*incx*。
*	コード例や式で乗算記号として使用。また、Fortran シンタックスでの必要箇所で使用。

参考文献

BLAS ルーチン、スパース BLAS ルーチン、LAPACK ルーチン、VML ルーチン、VSL ルーチン、DFT ルーチンのさらに詳しい説明は、以下の資料を参照。

- BLAS レベル 1

  C. Lawson, R. Hanson, D. Kincaid, and F. Krough,『*Basic Linear Algebra Subprograms for Fortran Usage*』, ACM Transactions on Mathematical Software, Vol.5, No.3 (September 1979) 308-325.

- BLAS レベル 2

  J. Dongarra, J. Du Croz, S. Hammarling, and R. Hanson, 『*An Extended Set of Fortran Basic Linear Algebra Subprograms*』, ACM Transactions on Mathematical Software, Vol.14, No.1 (March 1988) 1-32.

- BLAS レベル 3

  J. Dongarra, J. DuCroz, I. Duff, and S. Hammarling, 『*A Set of Level 3 Basic Linear Algebra Subprograms*』, ACM Transactions on Mathematical Software (December 1989).

- スパース BLAS

  D. Dodson, R. Grimes, and J. Lewis.『*Sparse Extensions to the FORTRAN Basic Linear Algebra Subprograms*』, ACM Transactions on Mathematical Software. Vol.17, No.2 (June 1991).

  D. Dodson, R. Grimes, and J. Lewis. 『*Algorithm 692: Model Implementation and Test Package for the Sparse Basic Linear Algebra Subprograms*』, ACM Transactions on Mathematical Software. Vol.17, No.2 (June 1991).

- LAPACK

  E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, 『*LAPACK Users's Guide*』, Third Edition, Society for Industrial and Applied Mathematics (SIAM), 1999.

  G. Golub and C. Van Loan, *Matrix Computations*, Johns Hopkins University Press, 1989.

- VML

  J.M. Muller, 『*Elementary functions: algorithms and implementation*』, Birkhauser Boston, 1997.

  IEEE Standard for Binary Floating-Point Arithmetic. ANSI/IEEE Std 754-1985.

- VSL

[Bratley87] Bratley P., Fox B.L., and Schrage L.E. 『A Guide to Simulation』, 2nd edition. Springer-Verlag, New York, 1987.

[Coddington94] Coddington, P. D. 『Analysis of Random Number Generators Using Monte Carlo Simulation』, Int. J. Mod. Phys. C-5, 547, 1994.

[Gentle98] Gentle, James E. 『Random Number Generation and Monte Carlo Methods』, Springer-Verlag New York, Inc., 1998.

[L'Ecuyer94] L'Ecuyer, Pierre. 『Uniform Random Number Generation』, Annals of Operations Research, 53, 77-120, 1994.

[L'Ecuyer99] L'Ecuyer, Pierre. 『Tables of Linear Congruential Generators of Different Sizes and Good Lattice Structure』, Mathematics of Computation, 68, 225, 249-260, 1999.

[L'Ecuyer99a] L'Ecuyer, Pierre. 『Good Parameter Sets for Combined Multiple Recursive Random Number Generators』, Operations Research, 47, 1, 159-164, 1999.

[L'Ecuyer01] L'Ecuyer, Pierre. 『Software for Uniform Random Number Generation: Distinguishing the Good and the Bad』, Proceedings of the 2001Winter Simulation Conference, IEEE Press, 95-105, Dec. 2001.

[Kirkpatrick81] Kirkpatrick, S., and Stoll, E. 『A Very Fast Shift-Register Sequence Random Number Generatory』, Journal of Computational Physics, V. 40. 517-526, 1981.

[Knuth81] Knuth, Donald E. 『The Art of Computer Programming, Volume 2, Seminumerical Algorithms』, 2nd edition, Addison-Wesley Publishing Company, Reading, Massachusetts, 1981.

[NAG] NAG Numerical Libraries. http://www.nag.co.uk/numeric/numerical_libraries.asp

- DFT

[1] E. Oran Brigham,『The Fast Fourier Transform and Its Applications』, Prentice Hall, New Jersey, 1988.

[2] Athanasios Papoulis,『The Fourier Integral and its Applications』, 2nd edition, McGraw-Hill, New York, 1984.

[3] Ping Tak Peter Tang,『DFTI, a New API for DFT: Motivation, Design, and Rationale』, July 2002.

[4] Charles Van Loan,『Computational Frameworks for the Fast Fourier Transform』, SIAM, Philadelphia, 1992

BLAS パッケージ、スパース BLAS パッケージ、LAPACK パッケージ (プラットフォーム固有の最適化なし) を参考用に導入するには、www.netlib.org を参照のこと。

BLAS ルーチンと スパース BLAS ルーチン

2

本章では、インテル ® マス・カーネル・ライブラリの BLAS ルーチンとスパース BLAS ルーチンについて説明する。ルーチンの説明は、BLAS の演算レベルによって以下の 4 つの節に分かれている。

- 「BLAS レベル 1 のルーチンと関数」(ベクトル・ベクトル演算)
- 「BLAS レベル 2 のルーチン」(マトリックス・ベクトル演算)
- 「BLAS レベル 3 のルーチン」(マトリックス・マトリックス演算)
- 「スパース BLAS のルーチンと関数」

それぞれの節では、ルーチンと関数のグループを、たとえば、?asum グループや ?axpy グループなどのように、アルファベット順に説明する。グループ名にある疑問符は、データ・タイプを示す各種のキャラクタ・コード (s、d、c、z、あるいはそれらの組み合わせ) に対応している。次のページの「ルーチン名の規則」を参照のこと。

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

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

ルーチン名の規則

BLAS ルーチンの名前は、次の構造になっている。

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

<character code> は、データ・タイプを示すキャラクタ・コードである。

s	実数、単精度	c	複素数、単精度
d	実数、倍精度	z	複素数、倍精度

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

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

BLAS レベル 2 と 3 では、*<name>* はマトリックスの引数のタイプを表す。

ge	一般マトリックス
gb	一般バンド・マトリックス
sy	対称マトリックス
sp	対称マトリックス (パック格納形式)
sb	対称バンド・マトリックス
he	エルミート・マトリックス
hp	エルミート・マトリックス (パック格納形式)
hb	エルミート・バンド・マトリックス
tr	三角マトリックス
tp	三角マトリックス (パック格納形式)
tb	三角バンド・マトリックス

<mod> フィールド (存在する場合) では、演算をさらに詳しく指示する。BLAS レベル 1 のルーチン名の *<mod>* フィールドには、次のキャラクタが入れられる。

c	共役ベクトル
u	非共役ベクトル
g	Givens 回転

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

mv	マトリックス・ベクトルの積
sv	マトリックス・ベクトル演算による連立 1 次方程式の解
r	階数 1 のマトリックスの更新
r2	階数 2 のマトリックスの更新

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

mm　マトリックス・マトリックスの積
sm　マトリックス・マトリックス演算による連立 1 次方程式の解
rk　階数 k のマトリックスの更新
r2k　階数 $2k$ のマトリックスの更新

以下に、BLAS ルーチン名を解釈する方法の例を示す。

<d> <dot>　ddot：倍精度実数型のベクトル・ベクトルのドット積
<c> <dot> <c>　cdotc：単精度複素共役のベクトル・ベクトルのドット積
<sc> <asum>　scasum：単精度複素数型を入力とし、単精度実数型を出力とするベクトルの成分の大きさの合計
<c> <dot> <u>　cdotu：単精度複素非共役のベクトル・ベクトルのドット積
<s> <ge> <mv>　sgemv：単精度一般マトリックスのマトリックス・ベクトルの積
<z> <tr> <mm>　ztrmm：倍精度複素数型三角マトリックスのマトリックス・マトリックスの積

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

マトリックスの格納形式

BLAS ルーチンのマトリックス引数では、次の格納形式を使用できる。

- **フル格納**では、マトリックス A は 2 次元配列 a に格納され、この場合、マトリックス成分 a_{ij} は配列成分 a(i, j) に格納される。
- **パックド格納**では、対称マトリックス、エルミート・マトリックス、あるいは三角マトリックスをコンパクトに格納できる。マトリックスの上三角または下三角が 1 次元配列の各列にパックされる。
- **バンド格納**では、バンド・マトリックスが 2 次元配列にコンパクトに格納される。マトリックスの各列は配列の対応する列に格納され、マトリックスの各**対角成分**は配列の各行に格納する。

マトリックスの格納形式の詳細は、付録 A の「マトリックス引数」を参照。

BLAS レベル 1 のルーチンと関数

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

表 2-1 BLAS レベル 1 のルーチン・グループおよびそのデータ・タイプ

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

?asum

ベクトル成分の大きさの合計を計算する。

```
res = sasum ( n, x, incx )
res = scasum ( n, x, incx )
res = dasum ( n, x, incx )
res = dzasum ( n, x, incx )
```

説明

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

$res = |\mathrm{Re}x(1)| + |\mathrm{Im}x(1)| + |\mathrm{Re}x(2)| + |\mathrm{Im}x(2)| + \ldots + |\mathrm{Re}x(n)| + |\mathrm{Im}x(n)|$

x は次数 *n* のベクトルである。

入力パラメータ

n	INTEGER。ベクトル *x* の次数を指定する。
x	REAL (sasum の場合) DOUBLE PRECISION (dasum の場合) COMPLEX (scasum の場合) DOUBLE COMPLEX (dzasum の場合) 配列 DIMENSION は少なくとも (1+ (*n*-1)*abs(*incx*))
incx	INTEGER。*x* の成分に対する増分を指定する。

出力パラメータ

res	REAL (sasum の場合) DOUBLE PRECISION (dasum の場合) REAL (scasum の場合) DOUBLE PRECISION (dzasum の場合) すべての成分の実数部の大きさに虚数部の大きさを加算した合計が格納される。

?axpy

ベクトル・スカラー積を計算し、
その結果をベクトルに加える。

```
call saxpy ( n, a, x, incx, y, incy )
call daxpy ( n, a, x, incx, y, incy )
call caxpy ( n, a, x, incx, y, incy )
call zaxpy ( n, a, x, incx, y, incy )
```

説明

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

*y := a*x + Y*

a はスカラー、

x と *y* は次数 *n* のベクトルである。

入力パラメータ

n	INTEGER。ベクトル *x* と *y* の次数を指定する。
a	REAL (saxpy の場合) DOUBLE PRECISION (daxpy の場合) COMPLEX (caxpy の場合) DOUBLE COMPLEX (zaxpy の場合) スカラー *a* を指定する。
x	REAL (saxpy の場合) DOUBLE PRECISION (daxpy の場合) COMPLEX (caxpy の場合) DOUBLE COMPLEX (zaxpy の場合) 配列 DIMENSION は少なくとも (1+ (*n*-1)*abs(*incx*))
incx	INTEGER。*x* の成分に対する増分を指定する。

y	REAL (saxpy の場合) DOUBLE PRECISION (daxpy の場合) COMPLEX (caxpy の場合) DOUBLE COMPLEX (zaxpy の場合) 配列 DIMENSION は少なくとも (1+ (*n*-1)*abs(*incy*))
incy	INTEGER。*y* の成分に対する増分を指定する。

出力パラメータ

y	更新されたベクトル *y* が格納される。

?copy

ベクトルを別のベクトルにコピーする。

```
call scopy ( n, x, incx, y, incy )
call dcopy ( n, x, incx, y, incy )
call ccopy ( n, x, incx, y, incy )
call zcopy ( n, x, incx, y, incy )
```

説明

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

y = *x*

x と *y* はベクトルである。

入力パラメータ

n	INTEGER。ベクトル *x* と *y* の次数を指定する。
x	REAL (scopy の場合) DOUBLE PRECISION (dcopy の場合) COMPLEX (ccopy の場合) DOUBLE COMPLEX (zcopy の場合) 配列 DIMENSION は少なくとも (1+ (*n*-1)*abs(*incx*))
incx	INTEGER。*x* の成分に対する増分を指定する。

y	REAL (scopy の場合) DOUBLE PRECISION (dcopy の場合) COMPLEX (ccopy の場合) DOUBLE COMPLEX (zcopy の場合) 配列 DIMENSION は少なくとも (1+ (*n*-1)*abs(*incy*))
incy	INTEGER。*y* の成分に対する増分を指定する。

出力パラメータ

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

?dot

ベクトル・ベクトルのドット積を計算する。

```
res = sdot ( n, x, incx, y, incy )
res = ddot ( n, x, incx, y, incy )
```

説明

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

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

x と *y* はベクトルである。

入力パラメータ

n	INTEGER。ベクトル *x* と *y* の次数を指定する。
x	REAL (sdot の場合) DOUBLE PRECISION (ddot の場合) 配列 DIMENSION は少なくとも (1+(*n*-1)*abs(*incx*))
incx	INTEGER。*x* の成分に対する増分を指定する。

y REAL (sdot の場合)
DOUBLE PRECISION (ddot の場合)
配列 DIMENSION は少なくとも
(1+(*n*-1)*abs(*incy*))

incy INTEGER。*y* の成分に対する増分を指定する。

出力パラメータ

res REAL (sdot の場合)
DOUBLE PRECISION (ddot の場合)
n が正の場合は、*x* と *y* のドット積の結果が格納される。そうでない場合には、*res* には 0 が格納される。

?sdot

ベクトル・ベクトルのドット積を
拡張精度で計算する。

```
res = sdsdot ( n, sb, sx, incx, sy, incy )
res = dsdot ( n, sx, incx, sy, incy )
```

説明

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

入力パラメータ

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

sb REAL。内積に加算する単精度スカラー (sdsdot 関数のみ)。

sx, sy REAL。
少なくとも、それぞれ (1+(*n*-1)*abs(*incx*))、(1+(*n*-1)*abs(*incy*)) の DIMENSION を持つ配列。入力単精度ベクトルが格納される。

incx INTEGER。*sx* の成分に対する増分を指定する。

incy NTEGER。*sy* の成分に対する増分を指定する。

出力パラメータ

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

?dotc

共役ベクトルと別のベクトルのドット積を計算する。

```
res = cdotc ( n, x, incx, y, incy )
res = zdotc ( n, x, incx, y, incy )
```

説明

?dotc 関数は、次のように定義されるベクトル・ベクトル演算を実行する。

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

x と *y* は *n* 個の成分を持つベクトルである。

入力パラメータ

n INTEGER。ベクトル *x* と *y* の次数を指定する。

x	COMPLEX (cdotc の場合) DOUBLE COMPLEX (zdotc の場合) 配列 DIMENSION は少なくとも (1 + (*n*-1)*abs(*incx*))
incx	INTEGER。*x* の成分に対する増分を指定する。
y	COMPLEX (cdotc の場合) DOUBLE COMPLEX (zdotc の場合) 配列 DIMENSION は少なくとも (1 + (*n*-1)*abs(*incy*))
incy	INTEGER。*y* の成分に対する増分を指定する。

出力パラメータ

res	COMPLEX (cdotc の場合) DOUBLE COMPLEX (zdotc の場合) *n* が正の場合は、共役の *x* と非共役の *y* のドット積の結果が格納される。そうでない場合には、*res* には 0 が格納される。

?dotu

ベクトル・ベクトルのドット積を
計算する。

```
res = cdotu ( n, x, incx, y, incy )
res = zdotu ( n, x, incx, y, incy )
```

説明

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

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

x と *y* は *n* 個の成分を持つベクトルである。

入力パラメータ

n	INTEGER。ベクトル *x* と *y* の次数を指定する。

x	COMPLEX (cdotu の場合) DOUBLE COMPLEX (zdotu の場合) 配列 DIMENSION は少なくとも (1 + (*n*-1)*abs(*incx*))
incx	INTEGER。*x* の成分に対する増分を指定する。
y	COMPLEX (cdotu の場合) DOUBLE COMPLEX (zdotu の場合) 配列 DIMENSION は少なくとも (1 + (*n*-1)*abs(*incy*))
incy	INTEGER。*y* の成分に対する増分を指定する。

出力パラメータ

res	COMPLEX (cdotu の場合) DOUBLE COMPLEX (zdotu の場合) *n* が正の場合は、*x* と *y* のドット積の結果が格納される。そうでない場合には、*res* には 0 が格納される。

?nrm2

ベクトルのユークリッド・ノルムを
計算する。

```
res = snrm2 ( n, x, incx )
res = dnrm2 ( n, x, incx )
res = scnrm2 ( n, x, incx )
res = dznrm2 ( n, x, incx )
```

説明

?nrm2 関数は、次のように定義されるベクトルの縮退演算を実行する。

res = ||*x*||

x はベクトルである。

res は、*x* の成分のユークリッド・ノルムが格納される値である。

入力パラメータ

n INTEGER。ベクトル x の次数を指定する。

x REAL (snrm2 の場合)
DOUBLE PRECISION (dnrm2 の場合)
COMPLEX (scnrm2 の場合)
DOUBLE COMPLEX (dznrm2 の場合)
配列 DIMENSION は少なくとも
(1 + (*n*-1)*abs (*incx*))

incx INTEGER。*x* の成分に対する増分を指定する。

出力パラメータ

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

?rot

面における点の回転を実行する。

```
call srot ( n, x, incx, y, incy, c, s )
call drot ( n, x, incx, y, incy, c, s )
call csrot ( n, x, incx, y, incy, c, s )
call zdrot ( n, x, incx, y, incy, c, s )
```

説明

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

```
x(i) = c*x(i) + s*y(i)
y(i) = c*y(i) - s*x(i)
```

入力パラメータ

n	INTEGER。ベクトル *x* と *y* の次数を指定する。
x	REAL (srot の場合) DOUBLE PRECISION (drot の場合) COMPLEX (csrot の場合) DOUBLE COMPLEX (zdrot の場合) 配列 DIMENSION は少なくとも (1 + (*n*-1)*abs(*incx*))
incx	INTEGER。*x* の成分に対する増分を指定する。
y	REAL (srot の場合) DOUBLE PRECISION (drot の場合) COMPLEX (csrot の場合) DOUBLE COMPLEX (zdrot の場合) 配列 DIMENSION は少なくとも (1 + (*n*-1)*abs(*incy*))
incy	INTEGER。*y* の成分に対する増分を指定する。
c	REAL (srot の場合) DOUBLE PRECISION (drot の場合) REAL (csrot の場合) DOUBLE PRECISION (zdrot の場合) スカラー。
s	REAL (srot の場合) DOUBLE PRECISION (drot の場合) REAL (csrot の場合) DOUBLE PRECISION (zdrot の場合) スカラー。

出力パラメータ

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

?rotg

Givens 回転に対するパラメータを
計算する。

```
call srotg ( a, b, c, s )
call drotg ( a, b, c, s )
call crotg ( a, b, c, s )
call zrotg ( a, b, c, s )
```

説明

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

入力パラメータ

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

出力パラメータ

a	Givens回転に関連付けられたパラメータ r が格納される。
b	Givens回転に関連付けられたパラメータ z が格納される。

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

?rotm

変形面における点の回転を実行する。

```
call srotm ( n, x, incx, y, incy, param )
call drotm ( n, x, incx, y, incy, param )
```

説明

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

```
x(i) = H*x(i) + H*y(i)
y(i) = H*y(i) - H*x(i)
```

H は変形 Givens 変換マトリックスで、その値は *param(2)* から *param(5)* の配列に格納される。*param* 引数の説明を参照のこと。

入力パラメータ

n INTEGER。ベクトル *x* と *y* の次数を指定する。

x REAL (srotm の場合)
DOUBLE PRECISION (drotm の場合)
配列 DIMENSION は少なくとも
(1 + (*n*-1)*abs(*incx*))

incx INTEGER。*x* の成分に対する増分を指定する。

y REAL (srotm の場合)
DOUBLE PRECISION (drotm の場合)
配列 DIMENSION は少なくとも
(1 + (*n*-1)*abs(*incy*))

incy INTEGER。*y* の成分に対する増分を指定する。

param REAL (srotm の場合)
DOUBLE PRECISION (drotm の場合)
配列 DIMENSION は 5

param 配列の成分は次のようになる。

param(1) にはスイッチ、*flag* を格納する。

param(2-5) にはそれぞれ、配列 *H* の成分である *h11*、*h21*、*h12*、*h22* を格納する。

flag の値により、*H* の成分は次のとおりに設定される。

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

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

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

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

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

出力パラメータ

x	それぞれの成分が *h11*x + h12*y* で置き換えられる。
y	それぞれの成分が *h21*x + h22*y* で置き換えられる。
H	更新された Givens 変換マトリックス。

?rotmg

Givens 回転に対する変形パラメータを計算する。

```
call srotmg ( d1, d2, x1, y1, param )
call drotmg ( d1, d2, x1, y1, param )
```

説明

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

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

入力パラメータ

d1	REAL (srotmg の場合) DOUBLE PRECISION (drotmg の場合) 入力ベクトル (sqrt *(d1)x1*) の *x* 座標に対するスケール係数を与える。
d2	REAL (srotmg の場合) DOUBLE PRECISION (drotmg の場合) 入力ベクトル (sqrt *(d2)y1*) の *y* 座標に対するスケール係数を与える。

x1 REAL (srotmg の場合)
DOUBLE PRECISION (drotmg の場合)
入力ベクトルの *x* 座標を与える。

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

出力パラメータ

param REAL (srotmg の場合)
DOUBLE PRECISION (drotmg の場合)
配列 DIMENSION は 5

param 配列の成分は次のようになる。

param(1) にはスイッチ、*flag* が格納される。
param(2-5) には、配列 *H* の成分である *h11*、*h21*、*h12*、*h22* がそれぞれ格納される。

flag の値により、*H* の成分は次のように設定される。

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

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

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

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

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

?scal

ベクトルとスカラーの積を計算する。

```
call sscal ( n, a, x, incx )
call dscal ( n, a, x, incx )
call cscal ( n, a, x, incx )
call zscal ( n, a, x, incx )
call csscal ( n, a, x, incx )
call zdscal ( n, a, x, incx )
```

説明

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

x = *a***x*

a はスカラー、*x* は *n* 個の成分を持つベクトルである。

入力パラメータ

n	INTEGER。ベクトル *x* の次数を指定する。
a	REAL (sscal、csscal の場合) DOUBLE PRECISION (dscal、zdscal の場合) REAL (cscal の場合) DOUBLE PRECISION (zscal の場合) スカラー *a* を指定する。
x	REAL (sscal の場合) DOUBLE PRECISION (dscal の場合) COMPLEX (cscal、csscal の場合) DOUBLE COMPLEX (zscal、csscal の場合) 配列 DIMENSION は少なくとも (1 + (*n*-1)*abs(*incx*))
incx	INTEGER。*x* の成分に対する増分を指定する。

出力パラメータ

x	更新されたベクトル x によって上書きされる。

?swap

ベクトルを別のベクトルと交換する。

```
call sswap ( n, x, incx, y, incy )
call dswap ( n, x, incx, y, incy )
call cswap ( n, x, incx, y, incy )
call zswap ( n, x, incx, y, incy )
```

説明

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

入力パラメータ

n	INTEGER。ベクトル *x* と *y* の次数を指定する。
x	REAL (sswap の場合) DOUBLE PRECISION (dswap の場合) COMPLEX (cswap の場合) DOUBLE COMPLEX (zswap の場合) 配列 DIMENSION は少なくとも (1 + (*n*-1)*abs(*incx*))
incx	INTEGER。x の成分に対する増分を指定する。
y	REAL (sswap の場合) DOUBLE PRECISION (dswap の場合) COMPLEX (cswap の場合) DOUBLE COMPLEX (zswap の場合) 配列 DIMENSION は少なくとも (1 + (*n*-1)*abs(*incy*))
incy	INTEGER。*y* の成分に対する増分を指定する。

出力パラメータ

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

i?amax

最大絶対値を持つベクトルの成分を検出する。

```
index = isamax ( n, x, incx )
index = idamax ( n, x, incx )
index = icamax ( n, x, incx )
index = izamax ( n, x, incx )
```

説明

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

n が正でない場合は、0 が返される。

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

入力パラメータ

n	INTEGER。ベクトル *x* の次数を指定する。
x	REAL (isamax の場合) DOUBLE PRECISION (idamax の場合) COMPLEX (icamax の場合) DOUBLE COMPLEX (izamax の場合) 配列 DIMENSION は少なくとも (1+(*n*-1)*abs(*incx*))
incx	INTEGER。*x* の成分に対する増分を指定する。

出力パラメータ

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

i?amin

最小絶対値を持つベクトル成分を検出する。

```
index = isamin ( n, x, incx )
index = idamin ( n, x, incx )
index = icamin ( n, x, incx )
index = izamin ( n, x, incx )
```

説明

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

n が正でない場合は , 0 が返される。

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

入力パラメータ

n	INTEGER。*n* にはベクトル *x* の次数を指定する。
x	REAL (isamin の場合) DOUBLE PRECISION (idamin の場合) COMPLEX (icamin の場合) DOUBLE COMPLEX (izamin の場合) 配列 DIMENSION は少なくとも (1+(*n*-1)*abs(*incx*))
incx	INTEGER。*x* の成分に対する増分を指定する。

出力パラメータ

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

BLAS レベル 2 のルーチン

この節では、マトリックス・ベクトル演算を実行する BLAS レベル 2 のルーチンについて説明する。表 2-2 に、BLAS レベル 2 のルーチンのグループと、それらに関連するデータ・タイプを示す。

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

ルーチン・グループ	データ・タイプ	説明
?gbmv	s, d, c, z	一般バンド・マトリックスでのマトリックス・ベクトルの積
?gemv	s, d, c, z	一般マトリックスでのマトリックス・ベクトルの積
?ger	s, d	一般マトリックスの階数 1 の更新
?gerc	c, z	共役一般マトリックスの階数 1 の更新
?geru	c, z	非共役一般マトリックスの階数 1 の更新
?hbmv	c, z	エルミート・バンド・マトリックスでのマトリックス・ベクトルの積
?hemv	c, z	エルミート・マトリックスでのマトリックス・ベクトルの積
?her	c, z	エルミート・マトリックスの階数 1 の更新
?her2	c, z	エルミート・マトリックスの階数 2 の更新
?hpmv	c, z	パックド形式のエルミート・マトリックスでのマトリックス・ベクトルの積
?hpr	c, z	パックド形式のエルミート・マトリックスの階数 1 の更新
?hpr2	c, z	パックド形式のエルミート・マトリックスの階数 2 の更新
?sbmv	s, d	対称バンド・マトリックスでのマトリックス・ベクトルの積
?spmv	s, d	パックド形式の対称マトリックスでのマトリックス・ベクトルの積
?spr	s, d	パックド形式の対称マトリックスの階数 1 の更新
?spr2	s, d	パックド形式の対称マトリックスの階数 2 の更新
?symv	s, d	対称マトリックスでのマトリックス・ベクトルの積
?syr	s, d	対称マトリックスの階数 1 の更新
?syr2	s, d	対称マトリックスの階数 2 の更新

（続く）

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

ルーチン・グループ	データ・タイプ	説明
?tbmv	s, d, c, z	三角バンド・マトリックスでのマトリックス・ベクトルの積
?tbsv	s, d, c, z	三角バンド・マトリックスの 1 次方程式の解
?tpmv	s, d, c, z	パックド形式の三角マトリックスでのマトリックス・ベクトルの積
?tpsv	s, d, c, z	パックド形式の三角マトリックスの 1 次方程式の解
?trmv	s, d, c, z	三角マトリックスでのマトリックス・ベクトルの積
?trsv	s, d, c, z	三角マトリックスの 1 次方程式の解

?gbmv

一般バンド・マトリックスを使用して
マトリックス・ベクトルの積を計算する。

```
call sgbmv ( trans, m, n, kl, ku, alpha, a, lda, x, inxc, beta, y,
             incy )
call dgbmv ( trans, m, n, kl, ku, alpha, a, lda, x, incx, beta, y,
             incy )
call cgbmv ( trans, m, n, kl, ku, alpha, a, lda, x, incx, beta, y,
             incy )
call zgbmv ( trans, m, n, kl, ku, alpha, a, lda, x, incx,  beta, y,
             incy )
```

説明

?gbmv ルーチンは、次のように定義されるマトリックス・ベクトル演算を実行する。

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

または

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

または

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

alpha と *beta* は、スカラーである。

x と *y* は、ベクトルである。

a は、*kl* 個の劣対角成分と *ku* 個の優対角成分を持つ *m* × *n* のバンド・マトリックスである。

入力パラメータ

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

trans 値	実行する演算
N または n	*y*:= *alpha***a***x* + *beta***y*
T または t	*y*:= *alpha***a*'**x* + *beta***y*
C または c	*y*:= *alpha**conjg(*a*')**x* +*beta***y*

m INTEGER。マトリックス *a* の行数を指定する。*m* の値は、ゼロ以上でなければならない。

n INTEGER。マトリックス *a* の列数を指定する。*n* の値は、ゼロ以上でなければならない。

kl INTEGER。マトリックス *a* の劣対角成分の数を指定する。*kl* の値は、0 ≤ *kl* を満たしていなければならない。

ku INTEGER。マトリックス *a* の優対角成分の数を指定する。*ku* の値は、0 ≤ *ku* を満たしていなければならない。

alpha REAL (sgbmv の場合)
DOUBLE PRECISION (dgbmv の場合)
COMPLEX (cgbmv 場合)
DOUBLE COMPLEX (zgbmv の場合)
スカラー *alpha* を指定する。

a REAL (sgbmv の場合)
DOUBLE PRECISION (dgbmv の場合)
OMPLEX (cgbmv の場合)
DOUBLE COMPLEX (zgbmv の場合)

配列 DIMENSION は (*lda*, *n*)。このルーチンに入る前に、配列 *a* の先頭の (*kl* + *ku* + 1) × *n* の部分に係数のマトリックスを格納しなければならない。このマトリックスは、マトリックスの主対角成分が配列の行 (*ku* + 1) にくるように、また、最初の優対角成分が行 (*ku* + 1) の位置 2 から始まり、最初の劣対角成分が行 (*ku* + 2) の位置 1 から始まるように、列ごとに与えなければならない。バンド・マトリックスの成分に対応しない配列 *a* の成分 (左上の *ku* × *ku* の三角など) は参照されない。

次に示すプログラムの一部は、バンド・マトリックスを通常のフル格納形式からバンド格納形式に転送するためのものである。

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

lda

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

x

REAL (sgbmv の場合)
DOUBLE PRECISION (dgbmv の場合)
COMPLEX (cgbmv の場合)
DOUBLE COMPLEX (zgbmv の場合)

配列 DIMENSION は *trans* = 'N' か 'n' の場合は少なくとも (1 + (*n* - 1)*abs(*incx*))、それ以外の場合は少なくとも (1 + (*m* - 1)*abs(*incx*)) でなければならない。このルーチンに入る前に、増分された配列 *x* にベクトル *x* を格納しなければならない。

incx

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

beta

REAL (sgbmv の場合)
DOUBLE PRECISION (dgbmv の場合)
COMPLEX (cgbmv の場合)
DOUBLE COMPLEX (zgbmv の場合)

スカラー beta を指定する。*beta* をゼロに設定した場合は、*y* を設定する必要はない。

y REAL (sgbmv の場合)
DOUBLE PRECISION (dgbmv の場合)
COMPLEX (cgbmv の場合)
DOUBLE COMPLEX (zgbmv の場合)
配列 DIMENSION は、*trans* ='N' または 'n' の場合は少なくとも (1 + (*m* - 1)*abs(*incy*))、それ以外の場合は少なくとも (1 + (*n* - 1)*abs(*incy*)) でなければならない。このルーチンに入る前に、増分された配列 *y* にベクトル *y* を格納しなければならない。

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

出力パラメータ

y 更新されたベクトル *y* によって上書きされる。

?gemv

一般マトリックスを使用して
マトリックス・ベクトルの積を計算する。

```
call sgemv ( trans, m, n, alpha, a, lda, x, incx,beta, y, incy )
call dgemv ( trans, m, n, alpha, a, lda, x, incx,beta, y, incy )
call cgemv ( trans, m, n, alpha, a, lda, x, incx,beta, y, incy )
call zgemv ( trans, m, n, alpha, a, lda, x, incx,beta, y, incy )
```

説明

?gemv ルーチンは、次のように定義されるマトリックス・ベクトル演算を実行する。

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

または

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

または

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

alpha と *beta* はスカラーである。

x と *y* はベクトルである。

a は、*m* × *n* のマトリックスである。

入力パラメータ

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

trans 値	実行する演算
N または n	*y:= alpha*a*x + beta*y*
T または t	*y:= alpha*a'*x + beta*y*
C または c	*y:= alpha*conjg(a')*x +beta*y*

m INTEGER。マトリックス *a* の行数を指定する。*m* の値は、ゼロ以上でなければならない。

n INTEGER。マトリックス *a* の列数を指定する。*n* の値は、ゼロ以上でなければならない。

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

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

配列 DIMENSION は (*lda*, *n*)。このルーチンに入る前に、配列 *a* の先頭の *m* × *n* の部分に係数のマトリックスを格納しなければならない。

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

x REAL (sgemv の場合)
DOUBLE PRECISION (dgemv の場合)
OMPLEX (cgemv の場合)
DOUBLE COMPLEX (zgemv の場合)
配列 DIMENSION は、*trans* = 'N' または 'n' の場合は少なくとも (1+(*n*-1)*abs(*incx*))、それ以外の場合は少なくとも (1+(*m* - 1)*abs(*incx*)) でなければならない。このルーチンに入る前に、増分された配列 *x* にベクトル *x* を格納しなければならない。

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

beta REAL (sgemv の場合)
DOUBLE PRECISION (dgemv の場合)
COMPLEX (cgemv の場合)
DOUBLE COMPLEX (zgemv の場合)
スカラー *beta* を指定する。*beta* をゼロに設定した場合は、*y* を設定する必要はない。

y REAL (sgemv の場合)
DOUBLE PRECISION (dgemv の場合)
COMPLEX (cgemv の場合)
DOUBLE COMPLEX (zgemv の場合)
配列 DIMENSION は、*trans* = 'N' または 'n' の場合は少なくとも (1+(*m*-1)*abs(*incy*))、それ以外の場合は少なくとも (1+(*n*-1)*abs(*incy*)) でなければならない。*beta* が非ゼロでこのルーチンに入る前に、増分された配列 *y* にベクトル *y* を格納しなければならない。

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

出力パラメータ

y	更新されたベクトル *y* によって上書きされる。

?ger

一般マトリックスの階数 1 の更新を
実行する。

```
call sger ( m, n, alpha, x, incx, y, incy, a, lda )
call dger ( m, n, alpha, x, incx, y, incy, a, lda )
```

説明

?ger ルーチンは、次のように定義されるマトリックス・ベクトル演算を実行する。

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

alpha は、スカラーである。

x は、*m* 個の成分を持つベクトルである。

y は、*n* 個の成分を持つベクトルである。

a は、*m* × *n* のマトリックスである。

入力パラメータ

m	INTEGER。マトリックス *a* の行数を指定する。*m* の値は、ゼロ以上でなければならない。
n	INTEGER。マトリックス *a* の列数を指定する。*n* の値は、ゼロ以上でなければならない。
alpha	REAL (sger の場合) DOUBLE PRECISION (dger の場合) スカラー *alpha* を指定する。
x	REAL (sger の場合) DOUBLE PRECISION (dger の場合)

	配列 DIMENSION は少なくとも (1 + (*m* - 1)*abs(*incx*)) でなければならない。このルーチンに入る前に、増分された配列 *x* に *m* 個の成分を持つベクトル *x* を格納しなければならない。
incx	INTEGER。*x* の成分に対する増分を指定する。*incx* の値は、ゼロであってはならない。
y	REAL (sger の場合) DOUBLE PRECISION (dger の場合) 配列 DIMENSION は少なくとも (1 + (*n* - 1)*abs(*incy*))。このルーチンに入る前に、増分された配列 *y* に *n* 個の成分を持つベクトル *y* を格納しなければならない。
incy	INTEGER。*y* の成分に対する増分を指定する。*incy* の値は、ゼロであってはならない。
a	REAL (sger の場合) DOUBLE PRECISION (dger の場合) 配列 DIMENSION は (*lda*, *n*)。このルーチンに入る前に、配列 *a* の先頭の *m* × *n* の部分に係数のマトリックスを格納しなければならない。
lda	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、*a* の第 1 次元を指定する。*lda* の値は、少なくとも max(1, *m*) でなければならない。

出力パラメータ

a	更新されたマトリックスによって上書きされる。

?gerc

一般マトリックスの階数 1 の更新
(共役) を実行する。

```
call cgerc ( m, n, alpha, x, incx, y, incy, a, lda )
call zgerc ( m, n, alpha, x, incx, y, incy, a, lda )
```

説明

?gerc ルーチンは、次のように定義されるマトリックス・ベクトル演算を実行する。

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

alpha は、スカラーである。

x は、*m* 個の成分を持つベクトルである。

y は、*n* 個の成分を持つベクトルである。

a は、*m* × *n* のマトリックスである。

入力パラメータ

m	INTEGER。マトリックス *a* の行数を指定する。*m* の値は、ゼロ以上でなければならない。
n	INTEGER。マトリックス *a* の列数を指定する。*n* の値は、ゼロ以上でなければならない。
alpha	SINGLE PRECISION COMPLEX (cgerc の場合) DOUBLE PRECISION COMPLEX (zgerc の場合) スカラー *alpha* を指定する。
x	SINGLE PRECISION COMPLEX (cgerc の場合) DOUBLE PRECISION COMPLEX (zgerc の場合) 配列 DIMENSION は少なくとも (1 + (*m* - 1)*abs(*incx*))。このルーチンに入る前に、増分された配列 *x* に *m* 個の成分を持つベクトル *x* を格納しなければならない。
incx	INTEGER。*x* の成分に対する増分を指定する。*incx* の値は、ゼロであってはならない。
y	COMPLEX (cgerc の場合) DOUBLE COMPLEX (zgerc の場合) 配列 DIMENSION は少なくとも (1 + (*n* - 1)*abs(*incy*)) このルーチンに入る前に、増分された配列 *y* に *n* 個の成分を持つベクトル *y* を格納しなければならない。
incy	INTEGER。*y* の成分に対する増分を指定する。*incy* の値は、ゼロであってはならない。

a	COMPLEX (cgerc の場合) DOUBLE COMPLEX (zgerc の場合) 配列 DIMENSION は (*lda*, *n*)。このルーチンに入る前に、配列 *a* の先頭の *m* × *n* の部分に係数のマトリックスを格納しなければならない。
lda	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、*a* の第 1 次元を指定する。*lda* の値は、少なくとも max(1, *m*) でなければならない。

出力パラメータ

a	更新されたマトリックスによって上書きされる。

?geru

一般マトリックスの階数 1 の更新
(非共役) を実行する。

```
call cgeru ( m, n, alpha, x, incx, y, incy, a, lda )
call zgeru ( m, n, alpha, x, incx, y, incy, a, lda )
```

説明

?geru ルーチンは、次のように定義されるマトリックス・ベクトル演算を実行する。

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

alpha は、スカラーである。

x は、*m* 個の成分を持つベクトルである。

y は、*n* 個の成分を持つベクトルである。

a は、*m* × *n* のマトリックスである。

入力パラメータ

m
INTEGER。マトリックス *a* の行数を指定する。*m* の値は、ゼロ以上でなければならない。

n
INTEGER。マトリックス *a* の列数を指定する。*n* の値は、ゼロ以上でなければならない。

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

x
COMPLEX (cgeru の場合)
DOUBLE COMPLEX (zgeru の場合)
配列 DIMENSION は少なくとも (1 + (*m* - 1)*abs(*incx*))。このルーチンに入る前に、増分された配列 *x* に *m* 個の成分を持つベクトル *x* を格納しなければならない。

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

y
COMPLEX (cgeru の場合)
DOUBLE COMPLEX (zgeru の場合)
配列 DIMENSION は少なくとも (1 + (*n* - 1)*abs(*incy*))。このルーチンに入る前に、増分された配列 *y* に *n* 個の成分を持つベクトル *y* を格納しなければならない。

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

a
COMPLEX (cgeru の場合)
DOUBLE COMPLEX (zgeru の場合)
配列 DIMENSION は (*lda*, *n*)。このルーチンに入る前に、配列 *a* の先頭の *m* × *n* の部分に係数のマトリックスを格納しなければならない。

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

出力パラメータ

a
更新されたマトリックスによって上書きされる。

?hbmv

エルミート・バンド・マトリックスを
使用してマトリックス・ベクトルの積を
計算する。

```
call chbmv ( uplo, n, k, alpha, a, lda, x, incx, beta, y, incy )
call zhbmv ( uplo, n, k, alpha, a, lda, x, incx, beta, y, incy )
```

説明

?hbmv ルーチンは、次のように定義されるマトリックス・ベクトル演算を実行する。

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

alpha と *beta* は、スカラーである。

x と *y* は、*n* 個の成分を持つベクトルである。

a は、*k* 個の優対角成分を持つ *n* × *n* のエルミート・バンド・マトリックスである。

入力パラメータ

uplo CHARACTER*1。次に示すように、バンド・マトリックス *a* の上三角部分と下三角部分のどちらを与えるかを指定する。

uplo の値	与えるマトリックス *a* の部分
U または u	マトリックス *a* の上三角部分を与える。
L または l	マトリックス *a* の下三角部分を与える。

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

k INTEGER。マトリックス *a* の優対角成分の数を指定する。*k* の値は、$0 \le k$ を満たしていなければならない。

alpha

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

a

COMPLEX (chbmv の場合)
DOUBLE COMPLEX (zhbmv の場合)
配列 DIMENSION は (*lda*, *n*)。*uplo* = 'U' または 'u' と指定した場合は、このルーチンに入る前に、配列 *a* の先頭の (*k* + 1) × *n* の部分に、エルミート・マトリックスの上三角バンド部分を格納しなければならない。このマトリックスには、マトリックスの主対角成分が配列の行 (*k* + 1) にくるように、また最初の優対角成分が行 *k* の位置 2 から始まるように、列ごとに与えなければならない。配列 *a* の左上の *k* × *k* の三角は、参照されない。

次に示すプログラムの一部は、エルミート・バンド・マトリックスの上三角部分を通常のフル格納形式のマトリックスからバンド格納形式に転送するためのものである。

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

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

次に示すプログラムの一部は、エルミート・バンド・マトリックスの下三角部分を通常のフル格納形式のマトリックスからバンド格納形式に転送するためのものである。

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

```
  a( m + i, j ) = matrix( i, j )
 10 continue
20 continue
```

対角成分の虚数部は設定する必要はなく、ゼロとみなされる。

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

x COMPLEX (chbmv の場合)
DOUBLE COMPLEX (zhbmv の場合)
配列 DIMENSION は少なくとも (1 + (*n* - 1)*abs(*incx*))。このルーチンに入る前に、増分された配列 *x* にベクトル *x* を格納しなければならない。

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

beta COMPLEX (chbmv の場合)
DOUBLE COMPLEX (zhbmv の場合)
スカラー *beta* を指定する。

y COMPLEX (chbmv の場合)
DOUBLE COMPLEX (zhbmv の場合)
配列 DIMENSION は少なくとも (1 + (*n* - 1)*abs(*incy*))。このルーチンに入る前に、増分された配列 *y* にベクトル *y* を格納しなければならない。

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

出力パラメータ

y 更新されたベクトル *y* によって上書きされる。

?hemv

エルミート・マトリックスを使用して
マトリックス・ベクトルの積を計算する。

```
call chemv ( uplo, n, alpha, a, lda, x, incx, beta, y, incy )
call zhemv ( uplo, n, alpha, a, lda, x, incx, beta, y, incy )
```

説明

?hemv ルーチンは、次のように定義されるマトリックス・ベクトル演算を実行する。

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

alpha と *beta* は、スカラーである。

x と *y* は、*n* 個の成分を持つベクトルである。

a は、*n* × *n* のエルミート・マトリックスである。

入力パラメータ

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

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

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

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

a
COMPLEX (chemv の場合)
DOUBLE COMPLEX (zhemv の場合)
配列 DIMENSION は (*lda*, *n*)。*uplo* = 'U' または 'u' と指定した場合は、このルーチンに入る前に、配列 *a* の先頭の *n* × *n* の上三角部分に、エルミート・マトリックスの上三角部分を格納しなければならず、また *a* の厳密な意味での下三角部分は参照されない。また、*uplo* = 'L' または 'l' と指定した場合は、このルーチンに入る前に、配列 *a* の先頭の *n* × *n* の下三角部分に、エルミート・マトリックスの下三角部分を格納しなければならず、また *a* の厳密な意味での上三角部分は参照されない。
対角成分の虚数部は設定する必要はなく、ゼロとみなされる。

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

x
COMPLEX (chemv の場合)
DOUBLE COMPLEX (zhemv の場合)
配列 DIMENSION は少なくとも (1 + (*n* - 1)*abs(*incx*))。このルーチンに入る前に、増分された配列 *x* に *n* 個の成分を持つベクトル *x* を格納しなければならない。

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

beta
COMPLEX (chemv の場合)
DOUBLE COMPLEX (zhemv の場合)
スカラー *beta* を指定する。*beta* をゼロに設定した場合は、*y* を設定する必要はない。

y	COMPLEX (chemv の場合) DOUBLE COMPLEX (zhemv の場合) 配列 DIMENSION は少なくとも (1 + (*n* - 1)*abs(*incy*))。このルーチンに入る前に、増分された配列 *y* に *n* 個の成分を持つベクトル *y* を格納しなければならない。
incy	INTEGER。*y* の成分に対する増分を指定する。*incy* の値は、ゼロであってはならない。

出力パラメータ

y	更新されたベクトル *y* によって上書きされる。

?her

エルミート・マトリックスの階数 1 の更新を実行する。

```
call cher ( uplo, n, alpha, x, incx, a, lda )
call zher ( uplo, n, alpha, x, incx, a, lda )
```

説明

?her ルーチンは、次のように定義されるマトリックス・ベクトル演算を実行する。

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

alpha は、実数のスカラーである。

x は、*n* 個の成分を持つベクトルである。

a は、*n* × *n* のエルミート・マトリックスである。

入力パラメータ

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

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

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

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

x COMPLEX (cher の場合)
DOUBLE COMPLEX (zher の場合)
配列、次元は少なくとも
(1 + (*n* - 1)*abs(*incx*))。このルーチンに入る前に、増分された配列 *x* に *n* 個の成分を持つベクトル *x* を格納しなければならない。

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

a COMPLEX (cher の場合)
DOUBLE COMPLEX (zher の場合)
配列 DIMENSION は (*lda*, *n*)。*uplo* = 'U' または 'u' と指定した場合は、このルーチンに入る前に、配列 *a* の先頭の *n* × *n* の上三角部分に、エルミート・マトリックスの上三角部分を格納しなければならず、また *a* の厳密な意味での下三角部分は参照されない。
uplo = 'L' または 'l' と指定した場合は、このルーチンに入る前に、配列 *a* の先頭の *n* × *n* の下三角部分に、エルミート・マトリックスの下三角部分を格納しなければならず、また *a* の厳密な意味での上三角部分は参照されない。

対角成分の虚数部は設定する必要はなく、ゼロとみなされる。

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

出力パラメータ

a *uplo* = 'U' または 'u' と指定した場合は、配列 *a* の上三角部分が、更新されたマトリックスの上三角部分によって上書きされる。

uplo = 'L' または 'l' と指定した場合は、配列 *a* の下三角部分が、更新されたマトリックスの下三角部分によって上書きされる。

対角成分の虚数部は、ゼロに設定される。

?her2

エルミート・マトリックスの階数 2 の更新を実行する。

```
call cher2 ( uplo, n, alpha, x, incx, y, incy, a, lda )
call zher2 ( uplo, n, alpha, x, incx, y, incy, a, lda )
```

説明

?her2 ルーチンは、次のように定義されるマトリックス・ベクトル演算を実行する。

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

alpha は、スカラーである。

x と *y* は、*n* 個の成分を持つベクトルである。

a は、*n* × *n* のエルミート・マトリックスである。

入力パラメータ

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

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

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

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

x COMPLEX (cher2 の場合)
DOUBLE COMPLEX (zher2 の場合)
配列 DIMENSION は少なくとも (1 + (*n* - 1)*abs(*incx*))。このルーチンに入る前に、増分された配列 *x* に *n* 個の成分を持つベクトル *x* を格納しなければならない。

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

y COMPLEX (cher2 の場合)
DOUBLE COMPLEX (zher2 の場合)
配列 DIMENSION は少なくとも (1 + (*n* - 1)*abs(*incy*))。このルーチンに入る前に、増分された配列 *y* に *n* 個の成分を持つベクトル *y* を格納しなければならない。

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

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

配列 DIMENSION (*lda*, *n*)。*uplo* = 'U' または 'u' と指定した場合は、このルーチンに入る前に、配列 *a* の先頭の *n* × *n* の上三角部分に、エルミート・マトリックスの上三角部分を格納しなければならず、また *a* の厳密な意味での下三角部分は参照されない。

uplo = 'L' または 'l' と指定した場合は、このルーチンに入る前に、配列 *a* の先頭の *n* × *n* の下三角部分に、エルミート・マトリックスの下三角部分を格納しなければならず、また *a* の厳密な意味での上三角部分は参照されない。

対角成分の虚数部は設定する必要はなく、ゼロとみなされる。

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

出力パラメータ

a *uplo* = 'U' または 'u' と指定した場合は、配列 *a* の上三角部分が、更新されたマトリックスの上三角部分によって上書きされる。

uplo = 'L' または 'l' と指定した場合は、配列 *a* の下三角部分が、更新されたマトリックスの下三角部分によって上書きされる。

対角成分の虚数部はゼロに設定される。

?hpmv

パックド形式のエルミート・マトリックスを使用してマトリックス・ベクトルの積を計算する。

```
call chpmv ( uplo, n, alpha, ap, x, incx, beta, y, incy )
call zhpmv ( uplo, n, alpha, ap, x, incx, beta, y, incy )
```

説明

?hpmv ルーチンは、次のように定義されるマトリックス・ベクトル演算を実行する。

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

alpha と *beta* は、スカラーである。

x と *y* は、*n* 個の成分を持つベクトルである。

a は、パックド形式で与えられる *n* × *n* のエルミート・マトリックスである。

入力パラメータ

uplo CHARACTER*1。次に示すように、マトリックス *a* の上三角部分と下三角部分のどちらをパックド形式の配列 *ap* に与えるかを指定する。

uplo の値	与えるマトリックス *a* の部分
U または u	マトリックス *a* の上三角部分を *ap* に与える。
L または l	マトリックス *a* の下三角部分を *ap* に与える。

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

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

ap COMPLEX (chpmv の場合)
DOUBLE COMPLEX (zhpmv の場合)
配列 DIMENSION は少なくとも ((*n**(*n* + 1))/2)。*uplo* = 'U' または 'u' と指定した場合は、このルーチンに入る前に、*ap*(1) に *a*(1, 1) が、*ap*(2) に *a*(1, 2) が、*ap*(3) に *a*(2, 2) がそれぞれ格納されるように、配列 *ap* には、順番にパックされたエルミート・マトリックスの上三角部分を列ごとに格納しなければならない。また、*uplo* = 'L' または 'l' と指定した場合は、このルーチンに入る前に、*ap*(1) に *a*(1, 1) が、*ap*(2) に *a*(2, 1) が、*ap*(3) に *a*(3, 1) がそれぞれ格納されるように、配列 *ap* には、順番にパックされたエルミート・マトリックスの下三角部分を列ごとに格納しなければならない。

対角成分の虚数部は設定する必要はなく、ゼロとみなされる。

x
COMPLEX (chpmv の場合)
DOUBLE PRECISION COMPLEX (zhpmv の場合)
配列 DIMENSION は少なくとも (1 + (*n* - 1)*abs(*incx*))。このルーチンに入る前に、増分された配列 *x* に *n* 個の成分を持つベクトル *x* を格納しなければならない。

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

beta
COMPLEX (chpmv の場合)
DOUBLE COMPLEX (zhpmv の場合)
スカラー *beta* を指定する。*beta* をゼロに設定した場合は、*y* を設定する必要はない。

y
COMPLEX (chpmv の場合)
DOUBLE COMPLEX (zhpmv の場合)
配列 DIMENSION は少なくとも (1 + (*n* - 1)*abs(*incy*))。このルーチンに入る前に、増分された配列 *y* に *n* 個の成分を持つベクトル *y* を格納しなければならない。

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

出力パラメータ

y
更新されたベクトル *y* によって上書きされる。

?hpr

パックド形式のエルミート・
マトリックスの階数 1 の更新を実行する。

```
call chpr ( uplo, n, alpha, x, incx, ap )
call zhpr ( uplo, n, alpha, x, incx, ap )
```

説明

?hpr ルーチンは、次のように定義されるマトリックス・ベクトル演算を実行する。

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

alpha は、実数のスカラーである。

x は、*n* 個の成分を持つベクトルである。

a は、パックド形式で与える *n* × *n* のエルミート・マトリックスである。

入力パラメータ

uplo CHARACTER*1。次に示すように、マトリックス *a* の上三角部分と下三角部分のどちらをパックド形式の配列 *ap* に与えるかを指定する。

uplo の値	与えるマトリックス *a* の部分
U または u	マトリックス *a* の上三角部分を *ap* に与える。
L または l	マトリックス *a* の下三角部分を *ap* に与える。

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

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

x
COMPLEX (chpr の場合)
DOUBLE COMPLEX (zhpr の場合)
配列 DIMENSION は少なくとも (1 + (*n* - 1)*abs(*incx*))。このルーチンに入る前に、増分された配列 *x* に *n* 個の成分を持つベクトル *x* を格納しなければならない。

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

ap
COMPLEX (chpr の場合)
DOUBLE COMPLEX (zhpr の場合)
配列 DIMENSION は少なくとも ((*n*(*n* + 1))/2)。
uplo = 'U' または 'u' と指定した場合は、このルーチンに入る前に、*ap*(1) に *a*(1, 1) が、*ap*(2) に *a*(1, 2) が、*ap*(3) に *a*(2, 2) がそれぞれ格納されるように、配列 *ap* には、順番にパックされたエルミート・マトリックスの上三角部分を列ごとに格納しなければならない。
uplo = 'L' または 'l' と指定した場合は、このルーチンに入る前に、*ap*(1) に *a*(1, 1) が、*ap*(2) に *a*(2, 1) が、*ap*(3) に *a*(3, 1) がそれぞれ格納されるように、配列 *ap* には、順番にパックされたエルミート・マトリックスの下三角部分を列ごとに格納しなければならない。
対角成分の虚数部は設定する必要はなく、ゼロとみなされる。

出力パラメータ

ap
uplo = 'U' または 'u' と指定した場合は、更新されたマトリックスの上三角部分によって上書きされる。
uplo = 'L' または 'l' と指定した場合は、更新されたマトリックスの下三角部分によって上書きされる。
対角成分の虚数部は、ゼロに設定される。

?hpr2

パックド形式のエルミート・
マトリックスの階数 2 の更新を実行する。

```
call chpr2 ( uplo, n, alpha, x, incx, y, incy, ap )
call zhpr2 ( uplo, n, alpha, x, incx, y, incy, ap )
```

説明

?hpr2 ルーチンは、次のように定義されるマトリックス・ベクトル演算を実行する。

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

alpha は、スカラーである。

x と *y* は、*n* 個の成分を持つベクトルである。

a は、パックド形式で与える *n* × *n* のエルミート・マトリックスである。

入力パラメータ

uplo CHARACTER*1。次に示すように、マトリックス *a* の上三角部分と下三角部分のどちらをパックド形式の配列 *ap* に与えるかを指定する。

uplo の値	与えるマトリックス *a* の部分
U または u	マトリックス *a* の上三角部分を *ap* に与える。
L または l	マトリックス *a* の下三角部分を *ap* に与える。

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

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

x	COMPLEX (chpr2 の場合) DOUBLE COMPLEX (zhpr2 の場合) 配列、次元は少なくとも (1 + (*n* - 1)*abs(*incx*))。このルーチンに入る前に、増分された配列 *x* に *n* 個の成分を持つベクトル *x* を格納しなければならない。
incx	INTEGER。*x* の成分に対する増分を指定する。*incx* の値は、ゼロであってはならない。
y	COMPLEX (chpr2 の場合) DOUBLE COMPLEX (zhpr2 の場合) 配列 DIMENSION は少なくとも (1 + (*n* - 1)*abs(*incy*))。このルーチンに入る前に、増分された配列 *y* に *n* 個の成分を持つベクトル *y* を格納しなければならない。
incy	INTEGER。*y* の成分に対する増分を指定する。*incy* の値は、ゼロであってはならない。
ap	COMPLEX (chpr2 の場合) DOUBLE COMPLEX (zhpr2 の場合) 配列 DIMENSION は少なくとも ((*n**(*n* + 1))/2)。 *uplo* = 'U' または 'u' と指定した場合は、このルーチンに入る前に、*ap*(1) に *a*(1, 1) が、*ap*(2) に *a*(1, 2) が、*ap*(3) に *a*(2, 2) がそれぞれ格納されるように、配列 *ap* には、順番にパックされたエルミート・マトリックスの上三角部分を列ごとに格納しなければならない。 *uplo* = 'L' または 'l' と指定した場合は、このルーチンに入る前に、*ap*(1) に *a*(1, 1) が、*ap*(2) に *a*(2, 1) が、*ap*(3) に *a*(3, 1) がそれぞれ格納されるように、配列 *ap* には、順番にパックされたエルミート・マトリックスの下三角部分を列ごとに格納しなければならない。 対角成分の虚数部は設定する必要はなく、ゼロとみなされる。

出力パラメータ

ap

uplo = 'U' または 'u' と指定した場合は、更新されたマトリックスの上三角部分によって上書きされる。

uplo = 'L' または 'l' と指定した場合は、更新されたマトリックスの下三角部分によって上書きされる。

対角成分の虚数部はゼロに設定される。

?sbmv

対称バンド・マトリックスを使用して
マトリックス・ベクトルの積を計算する。

```
call ssbmv ( uplo, n, k, alpha, a, lda, x, incx, beta, y, incy )
call dsbmv ( uplo, n, k, alpha, a, lda, x, incx, beta, y, incy )
```

説明

?sbmv ルーチンは、次のように定義されるマトリックス・ベクトル演算を実行する。

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

alpha と *beta* は、スカラーである。

x と *y* は、*n* 個の成分を持つベクトルである。

a は、*k* 個の優対角成分を持つ *n* × *n* の対称バンド・マトリックスである。

入力パラメータ

uplo CHARACTER*1。次に示すように、バンド・マトリックス a の上三角部分と下三角部分のどちらを与えるかを指定する。

uplo の値	与えるマトリックス *a* の部分
U または u	マトリックス *a* の上三角部分を与える。
L または l	マトリックス *a* の下三角部分を与える。

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

k INTEGER。マトリックス *a* の優対角成分の数を指定する。*k* の値は、$0 \le k$ を満たしていなければならない。

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

a REAL (ssbmv の場合)
DOUBLE PRECISION (dsbmv の場合)
配列 DIMENSION は (*lda*, *n*)。*uplo* = 'U' または 'u' と指定した場合は、このルーチンに入る前に、配列 *a* の先頭の $(k + 1) \times n$ の部分には、対称マトリックスの上三角バンド部分を格納しなければならない。このマトリックスには、マトリックスの主対角成分が配列の行 $(k + 1)$ にくるように、また最初の優対角成分が行 *k* の位置 2 から始まるように、列ごとに与えなければならない。配列 *a* の左上の $k \times k$ の三角は、参照されない。
次に示すプログラムの一部は、対称バンド・マトリックスの上三角部分を通常のフル格納形式のマトリックスからバンド格納形式に転送するためのものである。

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

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

次に示すプログラムの一部は、対称バンド・マトリックスの下三角部分を通常のフル格納形式のマトリックスからバンド格納形式に転送するためのものである。

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

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

x REAL (ssbmv の場合)
DOUBLE PRECISION (dsbmv の場合)
配列 DIMENSION は少なくとも (1 + (*n* - 1)*abs(*incx*))。このルーチンに入る前に、増分された配列 *x* にベクトル *x* を格納しなければならない。

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

beta REAL (ssbmv の場合)
DOUBLE PRECISION (dsbmv の場合)
スカラー *beta* を指定する。

y REAL (ssbmv の場合)
DOUBLE PRECISION (dsbmv の場合)
配列 DIMENSION は少なくとも (1 + (*n* - 1)*abs(*incy*))。このルーチンに入る前に、増分された配列 *y* にベクトル *y* を格納しなければならない。

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

出力パラメータ

y 更新されたベクトル *y* によって上書きされる。

?spmv

パックド形式の対称マトリックスを使用してマトリックス・ベクトルの積を計算する。

```
call sspmv ( uplo, n, alpha, ap, x, incx, beta, y, incy )
call dspmv ( uplo, n, alpha, ap, x, incx, beta, y, incy )
```

説明

?spmv ルーチンは、次のように定義されるマトリックス・ベクトル演算を実行する。

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

alpha と *beta* は、スカラーである。

x と *y* は、*n* 個の成分を持つベクトルである。

a は、パックド形式で与える *n* × *n* の対称マトリックスである。

入力パラメータ

uplo CHARACTER*1。次に示すように、マトリックス *a* の上三角部分と下三角部分のどちらをパックド形式の配列 *ap* に与えるかを指定する。

uplo の値	与えるマトリックス *a* の部分
U または u	マトリックス *a* の上三角部分を *ap* に与える。
L または l	マトリックス *a* の下三角部分を *ap* に与える。

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

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

ap
REAL (sspmv の場合)
DOUBLE PRECISION (dspmv の場合)
配列 DIMENSION は少なくとも ((n*(n + 1))/2)。*uplo* = 'U' または 'u' と指定した場合は、このルーチンに入る前に、*ap*(1) に *a*(1, 1) が、*ap*(2) に *a*(1, 2) が、*ap*(3) に *a*(2, 2) がそれぞれ格納されるように、配列 *ap* には、順番にパックされた対称マトリックスの上三角部分を列ごとに格納しなければならない。また、*uplo* = 'L' または 'l' と指定した場合は、このルーチンに入る前に、*ap*(1) に *a*(1, 1) が、*ap*(2) に *a*(2, 1) が、*ap*(3) に *a*(3, 1) がそれぞれ格納されるように、配列 *ap* には、順番にパックされた対称マトリックスの下三角部分を列ごとに格納しなければならない。

x
REAL (sspmv の場合)
DOUBLE PRECISION (dspmv の場合)
配列 DIMENSION は少なくとも (1 + (*n* - 1)*abs(*incx*))。このルーチンに入る前に、増分された配列 x に *n* 個の成分を持つベクトル *x* を格納しなければならない。

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

beta
REAL (sspmv の場合)
DOUBLE PRECISION (dspmv の場合)
スカラー *beta* を指定する。*beta* をゼロに設定した場合は、*y* を設定する必要はない。

y
REAL (sspmv の場合)
DOUBLE PRECISION (dspmv の場合)
配列 DIMENSION は少なくとも (1 + (*n* - 1)*abs(*incy*))。このルーチンに入る前に、増分された配列 *y* に *n* 個の成分を持つベクトル *y* を格納しなければならない。

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

出力パラメータ

y 更新されたベクトル *y* によって上書きされる。

?spr

パックド形式の対称マトリックスの
階数 1 の更新を実行する。

```
call sspr( uplo, n, alpha, x, incx, ap )
call dspr( uplo, n, alpha, x, incx, ap )
```

説明

?spr ルーチンは、次のように定義されるマトリックス・ベクトル演算を実行する。

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

alpha は、実数のスカラーである。

x は、*n* 個の成分を持つベクトルである。

a は、パックド形式で与える *n* × *n* の対称マトリックスである。

入力パラメータ

uplo CHARACTER*1。次に示すように、マトリックス *a* の上三角部分と下三角部分のどちらをパックド形式の配列 *ap* に与えるかを指定する。

uplo の値	与えるマトリックス *a* の部分
U または u	マトリックス *a* の上三角部分を *ap* に与える。
L または l	マトリックス *a* の下三角部分を *ap* に与える。

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

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

x REAL (sspr の場合)
DOUBLE PRECISION (dspr の場合)
配列 DIMENSION は少なくとも (1 + (*n* - 1)*abs(*incx*))。このルーチンに入る前に、増分された配列 *x* に *n* 個の成分を持つベクトル *x* を格納しなければならない。

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

ap REAL (sspr の場合)
DOUBLE PRECISION (dspr の場合)
配列 DIMENSION は少なくとも ((*n*(*n* + 1))/2)。
uplo = 'U' または 'u' と指定した場合は、このルーチンに入る前に、*ap*(1) に *a*(1,1) が、*ap*(2) に *a*(1,2) が、*ap*(3) に *a*(2,2) がそれぞれ格納されるように、配列 *ap* には、順番にパックされた対称マトリックスの上三角部分を列ごとに格納しなければならない。
uplo = 'L' または 'l' と指定した場合は、このルーチンに入る前に、*ap*(1) に *a*(1,1) が、*ap*(2) に *a*(2,1) が、*ap*(3) に *a*(3,1) がそれぞれ格納されるように、配列 *ap* には、順番にパックされた対称マトリックスの下三角部分を列ごとに格納しなければならない。

出力パラメータ

ap *uplo* = 'U' または 'u' と指定した場合は、更新されたマトリックスの上三角部分によって上書きされる。
uplo = 'L' または 'l' と指定した場合は、更新されたマトリックスの下三角部分によって上書きされる。

?spr2

パックド形式の対称マトリックスの
階数 2 の更新を実行する。

```
call sspr2( uplo, n, alpha, x, incx, y, incy, ap )
call dspr2( uplo, n, alpha, x, incx, y, incy, ap )
```

説明

?spr2 ルーチンは、次のように定義されるマトリックス・ベクトル演算を実行する。

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

alpha は、スカラーである。

x と *y* は、*n* 個の成分を持つベクトルである。

a は、パックド形式で与える *n* × *n* の対称マトリックスである。

入力パラメータ

uplo CHARACTER*1。次に示すように、マトリックス *a* の上三角部分と下三角部分のどちらをパックド形式の配列 *ap* に与えるかを指定する。

uplo の値	与えるマトリックス *a* の部分
U または u	マトリックス *a* の上三角部分を *ap* に与える。
L または l	マトリックス *a* の下三角部分を *ap* に与える。

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

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

x REAL (sspr2 の場合)
DOUBLE PRECISION (dspr2 の場合)
配列 DIMENSION は少なくとも (1 + (*n* - 1)*abs(*incx*))。このルーチンに入る前に、増分された配列 *x* に *n* 個の成分を持つベクトル *x* を格納しなければならない。

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

y REAL (sspr2 の場合)
DOUBLE PRECISION (dspr2 の場合)
配列 DIMENSION は少なくとも (1 + (*n* - 1)*abs(*incy*))。このルーチンに入る前に、増分された配列 *y* に *n* 個の成分を持つベクトル *y* を格納しなければならない。

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

ap REAL (sspr2 の場合)
DOUBLE PRECISION (dspr2 の場合)
配列 DIMENSION は少なくとも ((*n*(*n* + 1))/2)。
uplo = 'U' または 'u' と指定した場合は、このルーチンに入る前に、*ap*(1) に *a*(1,1) が、*ap*(2) に *a*(1,2) が、*ap*(3) に *a*(2,2) がそれぞれ格納されるように、配列 *ap* には、順番にパックされた対称マトリックスの上三角部分を列ごとに格納しなければならない。
uplo = 'L' または 'l' と指定した場合は、このルーチンに入る前に、*ap*(1) に *a*(1,1) が、*ap*(2) に *a*(2,1) が、*ap*(3) に *a*(3,1) がそれぞれ格納されるように、配列 *ap* には、順番にパックされたエルミート・マトリックスの下三角部分を列ごとに格納しなければならない。

出力パラメータ

ap *uplo* = 'U' または 'u' と指定した場合は、更新されたマトリックスの上三角部分によって上書きされる。
uplo = 'L' または 'l' と指定した場合は、更新されたマトリックスの下三角部分によって上書きされる。

?symv

対称マトリックスに対してマトリックス・ベクトルの積を計算する。

```
call ssymv ( uplo, n, alpha, a, lda, x, incx, beta, y, incy )
call dsymv ( uplo, n, alpha, a, lda, x, incx, beta, y, incy )
```

説明

?symv ルーチンは、次のように定義されるマトリックス・ベクトル演算を実行する。

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

alpha と *beta* は、スカラーである。

x と *y* は、*n* 個の成分を持つベクトルである。

a は、*n* × *n* の対称マトリックスである。

入力パラメータ

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

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

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

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

a
REAL (ssymv の場合)
DOUBLE PRECISION (dsymv の場合)
配列 DIMENSION (*lda*, *n*)。*uplo* = 'U' または 'u' と指定した場合は、このルーチンに入る前に、配列 *a* の先頭の *n* × *n* の上三角部分には、対称マトリックスの上三角部分を格納しなければならず、*a* の厳密な意味での下三角部分は参照されない。また、*uplo* = 'L' または 'l' と指定した場合は、このルーチンに入る前に、配列 *a* の先頭の *n* × *n* の下三角部分には、対称マトリックスの下三角部分を格納しなければならず、*a* の厳密な意味での上三角部分は参照されない。

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

x
REAL (ssymv の場合)
DOUBLE PRECISION (dsymv の場合)
配列 DIMENSION は少なくとも (1 + (*n* - 1)*abs(*incx*))。このルーチンに入る前に、増分された配列 *x* に *n* 個の成分を持つベクトル *x* を格納しなければならない。

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

beta
REAL (ssymv の場合)
DOUBLE PRECISION (dsymv の場合)
スカラー *beta* を指定する。*beta* をゼロに設定した場合は、*y* を設定する必要はない。

y
REAL (ssymv の場合)
DOUBLE PRECISION (dsymv の場合)
配列 DIMENSION は少なくとも (1 + (*n* - 1)*abs(*incy*))。このルーチンに入る前に、増分された配列 *y* に *n* 個の成分を持つベクトル *y* を格納しなければならない。

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

出力パラメータ

y 更新されたベクトル *y* によって上書きされる。

?syr

対称マトリックスの階数 1 の更新を実行する。

```
call ssyr( uplo, n, alpha, x, incx, a, lda )
call dsyr( uplo, n, alpha, x, incx, a, lda )
```

説明

?syr ルーチンは、次のように定義されるマトリックス・ベクトル演算を実行する。

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

alpha は、実数のスカラーである。

x は、*n* 個の成分を持つベクトルである。

a は、*n* × *n* の対称マトリックスである。

入力パラメータ

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

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

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

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

x REAL (ssyr の場合)
DOUBLE PRECISION (dsyr の場合)
配列 DIMENSION は少なくとも (1 + (*n* - 1)*abs(*incx*))。このルーチンに入る前に、増分された配列 *x* に *n* 個の成分を持つベクトル *x* を格納しなければならない。

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

a REAL (ssyr の場合)
DOUBLE PRECISION (dsyr の場合)
配列 DIMENSION (*lda*, *n*)。*uplo* = 'U' または 'u' と指定した場合は、このルーチンに入る前に、配列 *a* の先頭の *n* × *n* の上三角部分には対称マトリックスの上三角部分を格納しなければならず、*a* の厳密な意味での下三角部分は参照されない。
uplo = 'L' または 'l' と指定した場合は、このルーチンに入る前に、配列 *a* の先頭の *n* × *n* の下三角部分には対称マトリックスの下三角部分を格納しなければならず、*a* の厳密な意味での上三角部分は参照されない。

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

出力パラメータ

a *uplo* = 'U' または 'u' と指定した場合は、配列 *a* の上三角部分が、更新されたマトリックスの上三角部分によって上書きされる。
uplo = 'L' または 'l' と指定した場合は、配列 *a* の下三角部分が、更新されたマトリックスの下三角部分によって上書きされる。

?syr2

対称マトリックスの階数 2 の更新を
実行する。

```
call ssyr2( uplo, n, alpha, x, incx, y, incy, a, lda )
call dsyr2( uplo, n, alpha, x, incx, y, incy, a, lda )
```

説明

?syr2 ルーチンは、次のように定義されるマトリックス・ベクトル演算を実行する。

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

alpha は、スカラーである。

x と *y* は、*n* 個の成分を持つベクトルである。

a は、*n* × *n* の対称マトリックスである。

入力パラメータ

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

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

n INTEGER。マトリックス *a* の次数を指定する。*n* の値はゼロ以上でなければならない。

alpha REAL (ssyr2 の場合)
DOUBLE PRECISION (dsyr2 の場合)
スカラー *alpha* を指定する。

x
REAL (ssyr2 の場合)
DOUBLE PRECISION (dsyr2 の場合)
配列 DIMENSION は少なくとも (1 + (*n* - 1)*abs(*incx*))。このルーチンに入る前に、増分された配列 *x* に *n* 個の成分を持つベクトル *x* を格納しなければならない。

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

y
REAL (ssyr2 の場合)
DOUBLE PRECISION (dsyr2 の場合)
配列 DIMENSION は少なくとも (1 + (*n* - 1)*abs(*incy*))。このルーチンに入る前に、増分された配列 *y* に *n* 個の成分を持つベクトル *y* を格納しなければならない。

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

a
REAL (ssyr2 のの場合)
DOUBLE PRECISION (dsyr2 の場合)
配列 DIMENSION (*lda*, *n*)。*uplo* = 'U' または 'u' と指定した場合は、このルーチンに入る前に、配列 *a* の先頭の *n* × *n* の上三角部分には対称マトリックスの上三角部分を格納しなければならず、*a* の厳密な意味での下三角部分は参照されない。
uplo = 'L' または 'l' と指定した場合は、このルーチンに入る前に、配列 *a* の先頭の *n* × *n* の下三角部分には対称マトリックスの下三角部分を格納しなければならず、*a* の厳密な意味での上三角部分は参照されない。

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

出力パラメータ

a

uplo = 'U' または 'u' と指定した場合は、配列 *a* の上三角部分が、更新されたマトリックスの上三角部分によって上書きされる。

uplo = 'L' または 'l' と指定した場合は、配列 *a* の下三角部分が、更新されたマトリックスの下三角部分によって上書きされる。

?tbmv

三角バンド・マトリックスを使用して
マトリックス・ベクトルの積を計算する。

```
call stbmv ( uplo, trans, diag, n, k, a, lda, x, incx )
call dtbmv ( uplo, trans, diag, n, k, a, lda, x, incx )
call ctbmv ( uplo, trans, diag, n, k, a, lda, x, incx )
call ztbmv ( uplo, trans, diag, n, k, a, lda, x, incx )
```

説明

?tbmv ルーチンは、次のように定義されるマトリックス・ベクトル演算を実行する。

x := *a***x*、または *x* := *a*'**x*、または *x* := conjg(*a*')**x*

x は、*n* 個の成分を持つベクトルである。

a は、*n* × *n* 単位あるいは非単位の、(*k* + 1) 個の対角成分を持つ、上位または下位の三角バンド・マトリックスである。

入力パラメータ

uplo

CHARACTER*1。次に示すように、マトリックスが上三角マトリックスと下三角マトリックスのどちらであるかを指定する。

uplo の値	マトリックス *a*
U または u	上三角マトリックス。
L または l	下三角マトリックス。

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

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

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

diag の値	マトリックス *a*
U か u	マトリックス *a* は、単位三角マトリックスとみなされる。
N か n	マトリックス *a* は、単位三角マトリックスとはみなされない。

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

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

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

配列 DIMENSION (*lda*, *n*)。*uplo* = 'U' または 'u' と指定した場合は、このルーチンに入る前に、配列 *a* の先頭の $(k+1) \times n$ の部分には、係数マトリックスの上三角バンド部分を格納しなければならない。このマトリックスには、マトリックスの主対角成分が配列の行 $(k+1)$ にくるように、また最初の優対角成分が行 *k* の位置 2 から始まるように、列ごとに与

えなければならない。配列 *a* の左上の *k* × *k* の三角は、参照されない。次に示すプログラムの一部は、上三角バンド・マトリックスを通常のフル格納形式のマトリックスからバンド格納形式に転送するためのものである。

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

uplo = 'L' または 'l' と指定した場合は、このルーチンに入る前に、配列 a の先頭の (*k* + 1) × *n* の部分には、係数マトリックスの下三角バンド部分を格納しなければならない。このマトリックスには、マトリックスの主対角成分が配列の行 1 にくるように、また最初の劣対角成分が行 2 の位置 1 から始まるように、列ごとに与えなければならない。配列 *a* の右下の *k* × *k* の三角は、参照されない。次に示すプログラムの一部は、下三角バンド・マトリックスを通常のフル格納形式のマトリックスからバンド格納形式に転送するためのものである。

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

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

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

x	REAL (stbmv の場合) DOUBLE PRECISION (dtbmv の場合) COMPLEX (ctbmv の場合) DOUBLE COMPLEX (ztbmv の場合) 配列 DIMENSION は少なくとも (1 + (*n* - 1)*abs(*incx*))。このルーチンに入る前に、増分された配列 *x* に *n* 個の成分を持つベクトル *x* を格納しなければならない。
incx	INTEGER。*x* の成分に対する増分を指定する。*incx* の値は、ゼロであってはならない。

出力パラメータ

x	変換されたベクトル *x* によって上書きされる。

?tbsv

係数が三角バンド・マトリックスに
格納されている連立 1 次方程式を解く。

```
call stbsv ( uplo, trans, diag, n, k, a, lda, x, incx )
call dtbsv ( uplo, trans, diag, n, k, a, lda, x, incx )
call ctbsv ( uplo, trans, diag, n, k, a, lda, x, incx )
call ztbsv ( uplo, trans, diag, n, k, a, lda, x, incx )
```

説明

?tbsv ルーチンは、次に示す連立方程式のいずれかを解く。

*a*x = b*、または *a'*x = b*、または conjg(*a'*)*x = b*

b と *x* は、*n* 個の成分を持つベクトルである。

a は、*n* × *n* 単位あるいは非単位の、(*k* + 1) 個の対角成分を持つ、上位または下位の三角バンド・マトリックスである。

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

入力パラメータ

uplo CHARACTER*1。次に示すように、マトリックスが上三角マトリックスと下三角マトリックスのどちらであるかを指定する。

uplo の値	マトリックス *a*
U または u	上三角マトリックス。
L または l	下三角マトリックス。

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

trans の値	実行する演算
N または n	*a*x = b*
T または t	*a'*x = b*
C または c	conjg(*a*')*x = b*

diag CHARACTER*1。次に示すように、*a* が単位三角マトリックスであるかどうかを指定する。

diag の値	マトリックス *a*
U または u	マトリックス *a* は、単位三角マトリックスとみなされる。
N または n	マトリックス *a* は、単位三角マトリックスとはみなされない。

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

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

a

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

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

次のプログラム・セグメントは、上三角バンド・マトリックスを通常のフル格納形式のマトリックスからバンド格納形式に転送する。

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

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

次に示すプログラムの一部は、下三角バンド・マトリックスをフル格納形式のマトリックスからバンド格納形式に転送するためのものである。

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

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

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

x REAL (stbsv の場合)
DOUBLE PRECISION (dtbsv の場合)
COMPLEX (ctbsv の場合)
DOUBLE COMPLEX (ztbsv の場合)
配列 DIMENSION は少なくとも (1 + (*n* - 1)*abs(*incx*))。このルーチンに入る前に、増分された配列 *x* に *n* 個の成分からなる右辺のベクトル *b* を格納しなければならない。

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

出力パラメータ

x 解のベクトル *x* によって上書きされる。

?tpmv

パックド形式の三角マトリックスを使用してマトリックス・ベクトルの積を計算する。

```
call stpmv ( uplo, trans, diag, n, ap, x, incx )
call dtpmv ( uplo, trans, diag, n, ap, x, incx )
call ctpmv ( uplo, trans, diag, n, ap, x, incx )
call ztpmv ( uplo, trans, diag, n, ap, x, incx )
```

説明

?tpmv ルーチンは、次のように定義されるマトリックス・ベクトル演算を実行する。

x := *a*x*、または *x* := *a*'*x*、または *x* := conjg(*a*')*x*

x は、*n* 個の成分を持つベクトルである。

a は、*n* × *n* 単位あるいは非単位の、パックド格納形式の上三角マトリックスまたは下三角マトリックスである。

入力パラメータ

uplo CHARACTER*1。次に示すように、マトリックス *a* が上三角マトリックスと下三角マトリックスのどちらであるかを指定する。

uplo の値	マトリックス *a*
U または u	上三角マトリックス。
L または l	下三角マトリックス。

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

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

diag CHARACTER*1。次に示すように、*a* が単位三角マトリックスであるかどうかを指定する。

diag の値	マトリックス *a*
U または u	マトリックス *a* は、単位三角マトリックスとみなされる。
N または n	マトリックス *a* は、単位三角マトリックスとはみなされない。

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

ap REAL (stpmv の場合)
DOUBLE PRECISION (dtpmv の場合)
COMPLEX (ctpmv の場合)
DOUBLE COMPLEX (ztpmv の場合)

配列 DIMENSION は少なくとも ((*n**(*n* + 1))/2)。*uplo* = 'U' または 'u' と指定した場合は、このルーチンに入る前に、*ap*(1) に *a*(1,1) が、*ap*(2) に *a*(1,2) が、*ap*(3) に *a*(2,2) がそれぞれ格納されるように、配列 *ap* には、順番にパックされた上三角マトリックスを列ごとに格納しなければならない。また、*uplo* = 'L' または 'l' と指定した場合は、このルーチンに入る前に、*ap*(1) に *a*(1,1) が、*ap*(2) に *a*(2,1) が、*ap*(3) に *a*(3,1) がそれぞれ格納されるように、配列 *ap* には、順番にパックされた下三角マトリックスを列ごとに格納しなければならない。*diag* = 'U' または 'u' と指定した場合、*a* の対角成分は参照されないが、1 とみなされる。

x

REAL (stpmv の場合)
DOUBLE PRECISION (dtpmv の場合)
COMPLEX (ctpmv の場合)
DOUBLE COMPLEX (ztpmv の場合)

配列 DIMENSION は少なくとも (1 + (*n* - 1)*abs(*incx*))。このルーチンに入る前に、増分された配列 *x* に *n* 個の成分を持つベクトル *x* を格納しなければならない。

incx

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

出力パラメータ

x

変換されたベクトル *x* によって上書きされる。

?tpsv

係数がパックド形式の三角マトリックスに
格納されている連立 1 次方程式を解く。

```
call stpsv ( uplo, trans, diag, n, ap, x, incx )
call dtpsv ( uplo, trans, diag, n, ap, x, incx )
call ctpsv ( uplo, trans, diag, n, ap, x, incx )
call ztpsv ( uplo, trans, diag, n, ap, x, incx )
```

説明

?tpsv ルーチンは、次に示す連立方程式のいずれかを解く。

*a*x = b*、または a'*x = b、または conjg(a')*x = b

b と *x* は、*n* 個の成分を持つベクトルである。

a は、*n* × *n* 単位あるいは非単位の、パックド格納形式の上三角マトリックスまたは下三角マトリックスである。

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

入力パラメータ

uplo CHARACTER*1。次に示すように、マトリックス *a* が上三角マトリックスと下三角マトリックスのどちらであるかを指定する。

uplo の値	マトリックス *a*
U または u	上三角マトリックス。
L または l	下三角マトリックス。

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

trans の値	実行する演算
N または n	*a*x = b*
T または t	*a'*x = b*
C または c	conjg(*a'*)*x = b*

diag CHARACTER*1。次に示すように、*a* が単位三角マトリックスであるかどうかを指定する。

diag の値	マトリックス *a*
U または u	マトリックス *a* は、単位三角マトリックスとみなされる。
N または n	マトリックス *a* は、単位三角マトリックスとはみなされない。

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

ap REAL (stpsv の場合)
DOUBLE PRECISION (dtpsv の場合)
COMPLEX (ctpsv の場合)
DOUBLE COMPLEX (ztpsv の場合)
配列 DIMENSION は少なくとも ((*n*(*n* + 1))/2)。
uplo = 'U' または 'u' と指定した場合は、このルーチンに入る前に、*ap*(1) に *a*(1, 1) が、*ap*(2) に *a*(1, 2) が、*ap*(3) に *a*(2, 2) がそれぞれ格納されるように、配列 *ap* には、順番にパックされた上三角マトリックスを列ごとに格納しなければならない。また、*uplo* = 'L' または 'l' と指定した場合は、このルーチンに入る前に、*ap*(1) に *a*(1, 1) が、*ap*(2) に *a*(2, 1) が、*ap*(3) に *a*(3, 1) がそれぞれ格納されるように、配列 *ap* には、順番にパックされた下三角マトリックスを列ごとに格納しなければならない。*diag* = 'U' または 'u' と指定した場合、*a* の対角成分は参照されないが、1 とみなされる。

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

出力パラメータ

x	解として得られたベクトル *x* によって上書きされる。

?trmv

三角マトリックスを使用して
マトリックス・ベクトルの積を計算する。

```
call strmv ( uplo, trans, diag, n, a, lda, x, incx )
call dtrmv ( uplo, trans, diag, n, a, lda, x, incx )
call ctrmv ( uplo, trans, diag, n, a, lda, x, incx )
call ztrmv ( uplo, trans, diag, n, a, lda, x, incx )
```

説明

?trmv ルーチンは、次のように定義されるマトリックス・ベクトル演算のいずれかを実行する。

x := *a***x* または *x* := *a*'**x* または *x* := conjg(*a*')**x*

x は、*n* 個の成分を持つベクトルである。

a は、*n* × *n* 単位あるいは非単位の上三角マトリックスまたは下三角マトリックスである。

入力パラメータ

uplo CHARACTER*1。次に示すように、マトリックス *a* が上三角マトリックスと下三角マトリックスのどちらであるかを指定する。

uplo の値	マトリックス *a*
U または u	上三角マトリックス。
L または l	下三角マトリックス。

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

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

diag CHARACTER*1。次に示すように、*a* が単位三角マトリックスであるかどうかを指定する。

diag の値	マトリックス *a*
U または u	マトリックス *a* は、単位三角マトリックスとみなされる。
N または n	マトリックス *a* は、単位三角マトリックスとはみなされない。

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

a REAL (strmv の場合)
DOUBLE PRECISION (dtrmv の場合)
COMPLEX (ctrmv の場合)
DOUBLE COMPLEX (ztrmv の場合)
配列 DIMENSION は (*lda*, *n*)。*uplo* = 'U' または 'u' と指定した場合は、このルーチンに入る前に、配列 *a* の先頭の *n* × *n* の上三角部分に上三角マトリックスを格納しなければならず、*a* の厳密な意味での下三角部分は参照されない。また、*uplo* = 'L' または 'l' と指定した場合は、このルーチンに入る前に、配列 *a* の先頭の *n* × *n* の下三角部分に下三角マトリックスを格納しなければならず、*a* の厳密な

意味での上三角部分は参照されない。*diag* = 'U' または 'u' と指定した場合は、*a* の対角成分は参照されないが、1 とみなされる。

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

x REAL (strmv の場合)
DOUBLE PRECISION (dtrmv の場合)
COMPLEX (ctrmv の場合)
DOUBLE COMPLEX (ztrmv の場合)
配列 DIMENSION は少なくとも (1 + (*n* - 1)*abs(*incx*))。このルーチンに入る前に、増分された配列 *x* に *n* 個の成分を持つベクトル *x* を格納しなければならない。

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

出力パラメータ

x 変換されたベクトル *x* によって上書きされる。

?trsv

係数が三角マトリックスに格納されている
連立 1 次方程式を解く。

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

説明

?trsv ルーチンは、次に示す連立方程式のいずれかを解く。

*a*x = b* または *a'*x = b*、または conjg(*a*')*x = b*

b と *x* は、*n* 個の成分を持つベクトルである。

a は、*n* × *n* 単位あるいは非単位の上三角マトリックスまたは下三角マトリックスである。

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

入力パラメータ

uplo CHARACTER*1。次に示すように、マトリックスが上三角マトリックスと下三角マトリックスのどちらであるかを指定する。

uplo の値	マトリックス *a*
U または u	上三角マトリックス。
L または l	下三角マトリックス。

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

trans の値	実行する演算
N または n	*a*x = b*
T または t	*a'*x = b*
C または c	conjg(*a*')*x = b*

diag CHARACTER*1。次に示すように、*a* が単位三角マトリックスであるかどうかを指定する。

diag の値	マトリックス *a*
U または u	マトリックス *a* は、単位三角マトリックスとみなされる。
N または n	マトリックス *a* は、単位三角マトリックスとはみなされない。

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

a
REAL (strsv の場合)
DOUBLE PRECISION (dtrsv の場合)
COMPLEX (ctrsv の場合)
DOUBLE COMPLEX (ztrsv の場合)
配列 DIMENSION (*lda*, *n*)。*UPLO* = 'U' または 'u' と指定した場合は、このルーチンに入る前に、配列 *a* の先頭の *n* × *n* の上三角部分に上三角マトリックスを格納しなければならず、*a* の厳密な意味での下三角部分は参照されない。また、*uplo*= 'L' または 'l' と指定した場合は、このルーチンに入る前に、配列 *a* の先頭の *n* × *n* の下三角部分に下三角マトリックスを格納しなければならず、*a* の厳密な意味での上三角部分は参照されない。
diag = 'U' または 'u' と指定した場合は、*a* の対角成分は参照されないが、1 とみなされる。

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

x
REAL (strsv の場合)
DOUBLE PRECISION (dtrsv の場合)
COMPLEX (ctrsv の場合)
DOUBLE COMPLEX (ztrsv の場合)
配列 DIMENSION は少なくとも (1 + (*n* - 1)*abs(*incx*))。このルーチンに入る前に、増分された配列 *x* に *n* 個の成分からなる右辺のベクトル *b* を格納しなければならない。

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

出力パラメータ

x
解として得られたベクトル *x* によって上書きされる。

BLAS レベル 3 のルーチン

BLAS レベル 3 のルーチンは、マトリックス・マトリックス演算を実行する。表 2-3 に、BLAS レベル 3 のルーチンのグループと、それらに関連するデータ・タイプを示す。

表 2-3 BLAS レベル 3 のルーチン・グループおよびそのデータ・タイプ

ルーチン・グループ	データ・タイプ	説明
?gemm	s, d, c, z	一般マトリックスでのマトリックス・マトリックスの積
?hemm	c, z	エルミート・マトリックスでのマトリックス・マトリックスの積
?herk	c, z	エルミート・マトリックスの階数 k の更新
?her2k	c, z	エルミート・マトリックスの階数 2k の更新
?symm	s, d, c, z	対称マトリックスでのマトリックス・マトリックスの積
?syrk	s, d, c, z	対称マトリックスの階数 k の更新
?syr2k	s, d, c, z	対称マトリックスの階数 2k の更新
?trmm	s, d, c, z	三角マトリックスでのマトリックス・マトリックスの積
?trsm	s, d, c, z	三角マトリックスに対するマトリックス・マトリックスによる 1 次方程式の解

インテル® MKL のシンメトリック・マルチプロセシング・バージョン

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

MKL は、次の手法を使用して、性能の向上を実現している。

- BLAS レベル 3 のマトリックス・マトリックス関数による演算により、データの参照をローカルで実行し、キャッシュ・メモリの利用度を改善し、メモリ・バスへの依存度を低減する方法でコードを再構築できる。

- 上に説明した方法でコードが効率よくブロック化された後、マトリックスの 1 つが複数のプロセッサに分配され、2 番目のマトリックスで乗算される。このような分配により、キャッシュ管理の効率が改善され、メモリ・バスへの依存度が低減され、プロセッサ数に応じた優れた結果が得られる。

?gemm

スカラー・マトリックス・マトリックスの積を計算し、その結果をスカラー・マトリックスの積に加える。

```
call sgemm ( transa, transb, m, n, k, alpha, a, lda, b, ldb, beta, c,
             ldc )
call dgemm ( transa, transb, m, n, k, alpha, a, lda, b, ldb, beta, c,
             ldc )
call cgemm ( transa, transb, m, n, k, alpha, a, lda, b, ldb, beta, c,
             ldc )
call zgemm ( transa, transb, m, n, k, alpha, a, lda, b, ldb, beta, c,
             ldc )
```

説明

?gemm ルーチンは、一般マトリックスを使用してマトリックス・マトリックス演算を実行する。この演算は、次のように定義される。

c := *alpha**op(*a*)*op(*b*) + *beta***c*

op(*x*) は、op(*x*) = *x* または op(*x*) = *x*' または op(*x*) = conjg(*x*') のいずれかである。

alpha と *beta* は、スカラーである。

a、*b*、*c* は、マトリックスである。

op(*a*) は、*m* × *k* のマトリックスである。

op(*b*) は、*k* × *n* のマトリックスである。

c は、*m* × *n* のマトリックスである。

入力パラメータ

transa CHARACTER*1。次に示すように、マトリックスの乗算で使用する op(*a*) の形式を指定する。

transa の値	op(*a*) の形式
N または n	op(*a*) = *a*
T または t	op(*a*) = *a*'
C または c	op(*a*) = conjg(*a*')

transb CHARACTER*1。次に示すように、マトリックスの乗算で使用する op(*b*) の形式を指定する。

transb の値	op(*b*) の形式
N または n	op(*b*) = *b*
T または t	op(*b*) = *b*'
C または c	op(*b*) = conjg(*b*')

m INTEGER。マトリックス op(*a*) の行数とマトリックス *c* の行数を指定する。*m* の値はゼロ以上でなければならない。

n INTEGER。マトリックス op(*b*) の列数とマトリックス *c* の列数を指定する。*n* の値はゼロ以上でなければならない。

k INTEGER。マトリックス op(*a*) の列数とマトリックス op(*b*) の行数を指定する。*k* の値はゼロ以上でなければならない。

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

a

REAL (sgemm の場合)
DOUBLE PRECISION (dgemm の場合)
COMPLEX (cgemm の場合)
DOUBLE COMPLEX (zgemm の場合)

配列 DIMENSION は (*lda*, *ka*)。*ka* は *transa* = 'N' または 'n' の場合は *k* に、そうでない場合は *m* になる。*transa* = 'N' または 'n' と指定した場合は、このルーチンに入る前に、配列 *a* の先頭の *m* × *k* の部分にマトリックス *a* を格納しなければならない。そうでない場合は、配列 *a* の先頭の *k* × *m* の部分にマトリックス *a* を格納しなければならない。

lda

INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、*a* の第 1 次元を指定する。
transa = 'N' または 'n' の場合は、*lda* は少なくとも max(1, *m*) でなければならない。そうでない場合は、*lda* は少なくとも max(1, *k*) でなければならない。

b

REAL (sgemm の場合)
DOUBLE PRECISION (dgemm の場合)
COMPLEX (cgemm の場合)
DOUBLE COMPLEX (zgemm の場合)

配列 DIMENSION は (*ldb*, *kb*)。*kb* は *transb* = 'N' または 'n' の場合は *n* に、そうでない場合は *k* になる。*transb* = 'N' または 'n' と指定した場合は、このルーチンに入る前に、配列 *b* の先頭の *k* × *n* の部分にマトリックス *b* を格納しなければならない。そうでない場合は、配列 *b* の先頭の *n* × *k* の部分にマトリックス *b* を格納しなければならない。

ldb

INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、*b* の第 1 次元を指定する。
transb = 'N' または 'n' の場合は、*ldb* は少なくとも max(1, *k*) でなければならない。そうでない場合は、*ldb* は少なくとも max(1, *n*) でなければならない。

beta

REAL (sgemm の場合)
DOUBLE PRECISION (dgemm の場合)
COMPLEX (cgemm の場合)
DOUBLE COMPLEX (zgemm の場合)

スカラー *beta* を指定する。*beta* にゼロを設定した場合には、*c* を設定する必要はない。

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

出力パラメータ

c	*m*×*n*のマトリックス(*alpha**op(*a*)*op(*b*) + *beta***c*) によって上書きされる。

?hemm

スカラー・マトリックス・マトリックスの積を計算し (マトリックス・オペランドのいずれかはエルミート・マトリックス)、その結果をスカラー・マトリックスの積に加える。

```
call chemm ( side, uplo, m, n, alpha, a, lda, b, ldb, beta, c, ldc )
call zhemm ( side, uplo, m, n, alpha, a, lda, b, ldb, beta, c, ldc )
```

説明

?hemm ルーチンは、エルミート・マトリックスを使用してマトリックス・マトリックス演算を実行する。この演算は、次のように定義される。

```
c := alpha*a*b + beta*c
```

または、

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

alpha と *beta* は、スカラーである。

a は、エルミート・マトリックスである。

b と *c* は、*m* × *n* のマトリックスである。

入力パラメータ

side CHARACTER*1。次に示すように、演算でエルミート・マトリックス *a* が左と右のどちらにくるかを指定する。

side の値	実行する演算
L または l	*c := alpha*a*b + beta*c*
R または r	*c := alpha*b*a + beta*c*

uplo CHARACTER*1。次に示すように、エルミート・マトリックス *a* の上三角部分と下三角部分のどちらが参照されるかを指定する。

uplo の値	参照されるマトリックス *a* の部分
U または u	エルミート・マトリックスの上三角部分だけが参照される。
L または l	エルミート・マトリックスの下三角部分だけが参照される。

m INTEGER。マトリックス *c* の行数を指定する。*m* の値はゼロ以上でなければならない。

n INTEGER。マトリックス *c* の列数を指定する。*n* の値はゼロ以上でなければならない。

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

a

COMPLEX (chemm の場合)
DOUBLE COMPLEX (zhemm の場合)

配列 DIMENSION は (*lda*, *ka*)。*ka* は *side* = 'L' または 'l' の場合に *m* に、そうでない場合は *n* になる。*side* = 'L' または 'l' と指定した場合は、このルーチンに入る前に、次のように、配列 *a* の *m* × *m* の部分にエルミート・マトリックスを格納しなければならない。すなわち、*uplo* = 'U' または 'u' の場合は、配列 *a* の先頭の *m* × *m* の上三角部分にエルミート・マトリックスの上三角部分を格納しなければならず、また *a* の厳密な意味での下三角部分は参照されない。*uplo* = 'L' または 'l' の場合は、配列 *a* の先頭の *m* × *m* の下三角部分にエルミート・マトリックスの下三角部分を格納しなければならず、また *a* の厳密な意味での上三角部分は参照されない。*side* = 'R' または 'r' と指定した場合は、このルーチンに入る前に、次のように、配列 *a* の *n* × *n* の部分にエルミート・マトリックスを格納しなければならない。すなわち、*uplo* = 'U' または 'u' の場合は、配列 *a* の先頭の *n* × *n* の上三角部分にエルミート・マトリックスの上三角部分を格納しなければならず、また *a* の厳密な意味での下三角部分は参照されない。*uplo* = 'L' または 'l' の場合は、配列 *a* の先頭の *n* × *n* の下三角部分にエルミート・マトリックスの下三角部分を格納しなければならず、また *a* の厳密な意味での上三角部分は参照されない。対角成分の虚数部は設定する必要はない。これらは、ゼロであるとみなされる。

lda

INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、*a* の第 1 次元を指定する。
side = 'L' または 'l' の場合は、*lda* は少なくとも max(1, *m*) でなければならない。そうでない場合は、*lda* は少なくとも max(1, *n*) でなければならない。

b

COMPLEX (chemm の場合)
DOUBLE COMPLEX (zhemm の場合)

配列 DIMENSION は (*ldb*, *n*)。このルーチンに入る前に、配列 *b* の先頭の *m* × *m* の部分にマトリックス *b* を格納しなければならない。

ldb

INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、*b* の第 1 次元を指定する。*ldc* の値は、少なくとも max(1, *m*) でなければならない。

beta COMPLEX (chemm の場合)
DOUBLE COMPLEX (zhemm の場合)
スカラー *beta* を指定する。*beta* にゼロを設定した場合には、*c* を設定する必要はない。

c COMPLEX (chemm の場合)
DOUBLE COMPLEX (zhemm の場合)
配列 DIMENSION は (*c*, *n*)。このルーチンに入る前に、配列 *c* の先頭の *m* × *n* の部分にマトリックス c を格納しなければならない。ただし、*beta* がゼロの場合は、*c* を設定する必要はない。

ldc INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、*c* の第 1 次元を指定する。*ldc* の値は、少なくとも max(1, *m*) でなければならない。

出力パラメータ

c 更新された *m* × *n* のマトリックスによって上書きされる。

?herk

エルミート・マトリックスの階数 *n* の
更新を実行する。

```
call cherk ( uplo, trans, n, k, alpha, a, lda, beta, c, ldc )
call zherk ( uplo, trans, n, k, alpha, a, lda, beta, c, ldc )
```

説明

?herk ルーチンは、エルミート・マトリックスを使用してマトリックス・マトリックス演算を実行する。この演算は、次のように定義される。

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

または、

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

alpha と *beta* は、スカラーである。

c は、*n* × *n* のエルミート・マトリックスである。

a は、1 番目の定義では *n* × *k* のマトリックス、2 番目の定義では *k* × *n* のマトリックスである。

入力パラメータ

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

uplo の値	参照されるマトリックス *c* の部分
U または u	*C* の上三角部分だけが参照される。
L または l	*C* の下三角部分だけが参照される。

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

trans の値	実行する演算
N または n	*c*:= *alpha**a**conjg(*a*')+*beta**c*
C または c	*c*:= *alpha**conjg(*a*')**a*+*beta**c*

n INTEGER。マトリックス *c* の次数を指定する。*n* の値はゼロ以上でなければならない。

k INTEGER。*trans* = 'N' または 'n' と指定した場合は、*k* にはマトリックス *a* の列数を指定する。また、*trans* = 'C' または 'c' と指定した場合は、*k* にはマトリックス *a* の行数を指定する。*k* の値はゼロ以上でなければならない。

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

a

COMPLEX (cherk の場合)
DOUBLE COMPLEX (zherk の場合)

配列 DIMENSION は (*lda*, *ka*)。*ka* は *trans* = 'N' または 'n' と指定した場合は *k* に、そうでない場合は *n* になる。*trans* = 'N' または 'n' と指定した場合は、このルーチンに入る前に、配列 *a* の先頭の *n* × *k* の部分にマトリックス *a* を格納しなければならない。そうでない場合は、配列 *a* の先頭の *k* × *n* の部分にマトリックス *a* を格納しなければならない。

lda

INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、*a* の第 1 次元を指定する。
trans = 'N' か 'n' の場合は、*lda* は少なくとも max(1, *n*) でなければならない。そうでない場合は、*lda* は少なくとも max(1, *k*) でなければならない。

beta

REAL (cherk の場合)
DOUBLE PRECISION (zherk の場合)

スカラー *beta* を指定する。

c

COMPLEX (cherk の場合)
DOUBLE COMPLEX (zherk の場合)

配列 DIMENSION は (*ldc*, *n*)。*uplo* = 'U' または 'u' と指定した場合は、このルーチンに入る前に、配列 *c* の先頭の *n* × *n* の上三角部分にエルミート・マトリックスの上三角部分を格納しなければならず、また *c* の厳密な意味での下三角部分は参照されない。

uplo = 'L' または 'l' と指定した場合は、このルーチンに入る前に、配列 *c* の先頭の *n* × *n* の下三角部分にエルミート・マトリックスの下三角部分を格納しなければならず、また *c* の厳密な意味での上三角部分は参照されない。

対角成分の虚数部は設定する必要はない。これらは、ゼロであるとみなされる。

ldc

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

出力パラメータ

c

uplo = 'U' または 'u' と指定した場合は、配列 *c* の上三角部分が、更新されたマトリックスの上三角部分によって上書きされる。

uplo = 'L' または 'l' と指定した場合は、配列 *c* の下三角部分が、更新されたマトリックスの下三角部分によって上書きされる。

対角成分の虚数部は、ゼロに設定される。

?her2k

エルミート・マトリックスの階数 2k の更新を実行する。

```
call cher2k ( uplo, trans, n, k, alpha, a, lda, b, ldb, beta, c, ldc )
call zher2k ( uplo, trans, n, k, alpha, a, lda, b, ldb, beta, c, ldc )
```

説明

?her2k ルーチンは、エルミート・マトリックスを使用して階数 2k のマトリックス・マトリックス演算を実行する。この演算は、次のように定義される。

c := *alpha***a**conjg(*b*') + conjg(*alpha*)**b**conjg(*a*') + *beta***c*

または、

c := *alpha**conjg(*b*')**a* + conjg(*alpha*)*conjg(*a*')**b* + *beta***c*

alpha はスカラー、*beta* は実数のスカラーである。

c は、*n* × *n* のエルミート・マトリックスである。

a と *b* は、最初の定義では *n* × *k* のマトリックス、2 番目の定義では *k* × *n* のマトリックスである。

入力パラメータ

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

uplo の値	参照されるマトリックス *c* の部分
U または u	*C* の上三角部分だけが参照される。
L または l	*C* の下三角部分だけが参照される。

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

trans の値	実行する演算
N または n	*c*:=*alpha***a**conjg(*b*') +*alpha***b**conjg(*a*') +*beta***c*
C または c	*c*:=*alpha**conjg(*a*')**b* +*alpha**conjg(*b*')**a*+*beta***c*

n INTEGER。マトリックス *c* の次数を指定する。*n* の値はゼロ以上でなければならない。

k INTEGER。*trans* = 'N' または 'n' と指定した場合は、*k* にはマトリックス *a* の列数を指定する。また、*trans* = 'C' または 'c' と指定した場合は、*k* にはマトリックス *a* の行数を指定する。*k* の値はゼロ以上でなければならない。

alpha COMPLEX (cher2k の場合)
DOUBLE COMPLEX (zher2k の場合)
スカラー *alpha* を指定する。

a

COMPLEX (cher2k の場合)
DOUBLE COMPLEX (zher2k の場合)

配列 DIMENSION は (*lda*, *ka*)。*ka* は *trans* = 'N' または 'n' の場合は *k* に、そうでない場合は *n* になる。*trans* = 'N' または 'n' と指定した場合は、このルーチンに入る前に、配列 *a* の先頭の *n* × *k* の部分にマトリックス *a* を格納しなければならない。そうでない場合は、配列 *a* の先頭の *k* × *n* の部分にマトリックス *a* を格納しなければならない。

lda

INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、*a* の第 1 次元を指定する。
trans = 'N' か 'n' の場合は、*lda* は少なくとも max(1, *n*) でなければならない。そうでない場合は、*lda* は少なくとも max(1, *k*) でなければならない。

beta

REAL (cher2k の場合)
DOUBLE PRECISION (zher2k の場合)

スカラー *beta* を指定する。

b

COMPLEX (cher2k の場合)
DOUBLE COMPLEX (zher2k の場合)

配列 DIMENSION は (*ldb*, *kb*)。*kb* は *trans* = 'N' または 'n' の場合は *k* に、そうでない場合は *n* になる。*trans* = 'N' または 'n' と指定した場合は、このルーチンに入る前に、配列 *b* の先頭の *n* × *k* の部分にマトリックス *b* を格納しなければならない。そうでない場合は、配列 *b* の先頭の *k* × *n* の部分にマトリックス *b* を格納しなければならない。

ldb

INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、*b* の第 1 次元を指定する。
trans = 'N' か 'n' の場合は、*ldb* は少なくとも max(1, *n*) でなければならない。そうでない場合は、*ldb* は少なくとも max(1, *k*) でなければならない。

c

COMPLEX (cher2k の場合)
DOUBLE COMPLEX (zher2k の場合)

配列 DIMENSION は (*ldc*, *n*)。*uplo* = 'U' または 'u' と指定した場合は、このルーチンに入る前に、配列 *c* の先頭の *n* × *n* の上三角部分にエルミート・マトリックスの上三角部分を格納しなければならず、また *c* の厳密な意味での下三角部分は参照されない。

uplo = 'L' または 'l' と指定した場合は、このルーチンに入る前に、配列 *c* の先頭の *n* × *n* の下三角部分にエルミート・マトリックスの下三角部分を格納しなければならず、また *c* の厳密な意味での上三角部分は参照されない。

対角成分の虚数部は設定する必要はない。これらは、ゼロであるとみなされる。

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

出力パラメータ

c *uplo* = 'U' または 'u' と指定した場合は、配列 *c* の上三角部分が、更新されたマトリックスの上三角部分によって上書きされる。

uplo = 'L' または 'l' と指定した場合は、配列 *c* の下三角部分が、更新されたマトリックスの下三角部分によって上書きされる。

対角成分の虚数部は、ゼロに設定される。

?symm

スカラー・マトリックス・マトリックスの積 (マトリックス・オペランドのいずれかは対称マトリックス) を計算し、その結果をスカラー・マトリックスの積に加える。

```
call ssymm ( side, uplo, m, n, alpha, a, lda, b, ldb, beta, c, ldc )
call dsymm ( side, uplo, m, n, alpha, a, lda, b, ldb, beta, c, ldc )
call csymm ( side, uplo, m, n, alpha, a, lda, b, ldb, beta, c, ldc )
call zsymm ( side, uplo, m, n, alpha, a, lda, b, ldb, beta, c, ldc )
```

説明

?symm ルーチンは、対称マトリックスを使用してマトリックス・マトリックス演算を実行する。この演算は、次のように定義される。

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

または、

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

alpha と *beta* は、スカラーである。

a は、対称マトリックスである。

b と *c* は、*m* × *n* のマトリックスである。

入力パラメータ

side CHARACTER*1。次に示すように、演算で対称マトリックス *a* が左と右のどちらにくるかを指定する。

side の値	実行する演算
L または l	*c := alpha*a*b + beta*c*
R または r	*c := alpha*b*a + beta*c*

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

uplo の値	参照されるマトリックス *a* の部分
U または u	対称マトリックスの上三角部分だけが参照される。
L または l	対称マトリックスの下三角部分だけが参照される。

m INTEGER。マトリックス *c* の行数を指定する。*m* の値はゼロ以上でなければならない。

n INTEGER。マトリックス *c* の列数を指定する。*n* の値はゼロ以上でなければならない。

alpha REAL (ssymm の場合)
DOUBLE PRECISION (dsymm の場合)
COMPLEX (csymm の場合)
DOUBLE COMPLEX (zsymm の場合)
スカラー *alpha* を指定する。

a REAL (ssymm の場合)
DOUBLE PRECISION (dsymm の場合)
COMPLEX (csymm の場合)
DOUBLE COMPLEX (zsymm の場合)
配列 DIMENSION は *(lda, ka)*。*ka* は *side* = 'L' または 'l' の場合は *m* に、そうでない場合は n になる。*side* = 'L' または 'l' と指定した場合は、このルーチンに入る前に、次のように、配列 *a* の *m* × *m* の部分に対称マトリックスを格納しなければならない。すなわち、*uplo* = 'U' または 'u' の場合は、配列 *a* の先頭の *m* × *m* の上三角部分に対称マトリックスの上三角部分を格納しなければならず、また *a* の厳密な意味での下三角部分は参照されない。また、*uplo* = 'L' または 'l' の場合は、配列 *a* の先頭の *m* × *m*

の下三角部分に対称マトリックスの下三角部分を格納しなければならず、また *a* の厳密な意味での上三角部分は参照されない。

side = 'R' または 'r' と指定した場合は、このルーチンに入る前に、次のように、配列 *a* の *n* × *n* の部分に対称マトリックスを格納しなければならない。すなわち、*uplo* = 'U' または 'u' の場合は、配列 *a* の先頭の *n* × *n* の上三角部分に対称マトリックスの上三角部分を格納しなければならず、また *a* の厳密な意味での下三角部分は参照されない。また、*uplo* = 'L' または 'l' の場合は、配列 *a* の先頭の *n* × *n* の下三角部分に対称マトリックスの下三角部分を格納しなければならず、また *a* の厳密な意味での上三角部分は参照されない。

lda INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、*a* の第 1 次元を指定する。*side* = 'L' または 'l' の場合は、*lda* は少なくとも max(1, *m*) でなければならない。そうでない場合は、*lda* は少なくとも max(1, *n*) でなければならない。

b REAL (ssymm の場合)
DOUBLE PRECISION (dsymm の場合)
COMPLEX (csymm の場合)
DOUBLE COMPLEX (zsymm の場合)

配列 DIMENSION は (*ldb*, *n*)。このルーチンに入る前に、配列 b の先頭の *m* × *n* の部分にマトリックス *b* を格納しなければならない。

ldb INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、b の第 1 次元を指定する。*ldb* の値は、少なくとも max(1, *m*) でなければならない。

beta
REAL (ssymm の場合)
DOUBLE PRECISION (dsymm の場合)
COMPLEX (csymm の場合)
DOUBLE COMPLEX (zsymm の場合)
スカラー *beta* を指定する。*beta* にゼロを設定した場合には、*c* を設定する必要はない。

c
REAL (ssymm の場合)
DOUBLE PRECISION (dsymm の場合)
COMPLEX (csymm の場合)
DOUBLE COMPLEX (zsymm の場合)
配列 DIMENSION(*ldc*, *n*)。このルーチンに入る前に、配列 *c* の先頭の *m* × *n* の部分にマトリックス *c* を格納しなければならない。ただし、beta がゼロの場合は、*c* を設定する必要はない。

ldc
INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、*c* の第 1 次元を指定する。*ldc* の値は、少なくとも max(1, *m*) でなければならない。

出力パラメータ

c
更新された *m* × *n* のマトリックスによって上書きされる。

?syrk

対称マトリックスの階数 *n* の更新を
実行する。

```
call ssyrk ( uplo, trans, n, k, alpha, a, lda, beta, c, ldc )
call dsyrk ( uplo, trans, n, k, alpha, a, lda, beta, c, ldc )
call csyrk ( uplo, trans, n, k, alpha, a, lda, beta, c, ldc )
call zsyrk ( uplo, trans, n, k, alpha, a, lda, beta, c, ldc )
```

説明

`?syrk` ルーチンは、対称マトリックスを使用してマトリックス・マトリックス演算を実行する。この演算は、次のように定義される。

```
c := alpha*a*a' + beta*c
```

または、

```
c := alpha*a'*a + beta*c
```

alpha と *beta* は、スカラーである。

c は、*n* × *n* の対称マトリックスである。

a は、1 番目の定義では *n* × *k* のマトリックス、2 番目の定義では *k* × *n* のマトリックスである。

入力パラメータ

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

uplo の値	参照されるマトリックス *c* の部分
U または u	*c* の上三角部分だけが参照される。
L または l	*c* の下三角部分だけが参照される。

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

trans の値	実行する演算
N または n	*c*:= *alpha*a*a'* + *beta*c*
T または t	*c*:= *alpha*a'*a* + *beta*c*
C または c	*c*:= *alpha*a'*a* + *beta*c*

n INTEGER。マトリックス *c* の次数を指定する。*n* の値はゼロ以上でなければならない。

k INTEGER。*trans* = 'N' または 'n' と指定した場合は、*k* にはマトリックス *a* の列数を指定する。また、*trans* = 'T' または 't'、あるいは 'C' または 'c' と指定した場合は、*k* にはマトリックス *a* の行数を指定する。*k* の値はゼロ以上でなければならない。

alpha REAL (ssyrk の場合)
DOUBLE PRECISION (dsyrk の場合)
COMPLEX (csyrk の場合)
DOUBLE COMPLEX (zsyrk の場合)
スカラー *alpha* を指定する。

a

REAL (ssyrk の場合)
DOUBLE PRECISION (dsyrk の場合)
COMPLEX (csyrk の場合)
DOUBLE COMPLEX (zsyrk の場合)

配列 DIMENSION (*lda*, *ka*)。*ka* は *trans* = 'N' または 'n' の場合は *k* に、そうでない場合は *n* になる。*trans* = 'N' または 'n' と指定した場合は、このルーチンに入る前に、配列 *a* の先頭の *n* × *k* の部分にマトリックス *a* を格納しなければならない。そうでない場合は、配列 *a* の先頭の *k* × *n* の部分にマトリックス *a* を格納しなければならない。

lda

INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、*a* の第 1 次元を指定する。*trans* = 'N' または 'n' の場合は、*lda* は少なくとも max(1, *n*) でなければならない。そうでない場合は、*lda* は少なくとも max(1, *k*) でなければならない。

beta

REAL (ssyrk の場合)
DOUBLE PRECISION (dsyrk の場合)
COMPLEX (csyrk の場合)
DOUBLE COMPLEX (zsyrk の場合)

スカラー *beta* を指定する。

c

REAL (ssyrk の場合)
DOUBLE PRECISION (dsyrk の場合)
COMPLEX (csyrk の場合)
DOUBLE COMPLEX (zsyrk の場合)

配列 DIMENSION は (*ldc*, *n*)。*uplo* = 'U' または 'u' と指定した場合は、このルーチンに入る前に、配列 *c* の先頭の *n* × *n* の上三角部分に対称マトリックスの上三角部分を格納しなければならず、また *c* の厳密な意味での下三角部分は参照されない。

uplo = 'L' または 'l' と指定した場合は、このルーチンに入る前に、配列 *c* の先頭の *n* × *n* の下三角部分に対称マトリックスの下三角部分を格納しなければならず、また *c* の厳密な意味での上三角部分は参照されない。

ldc

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

出力パラメータ

c

uplo = 'U' または 'u' と指定した場合は、配列 *c* の上三角部分が、更新されたマトリックスの上三角部分によって上書きされる。

uplo = 'L' または 'l' と指定した場合は、配列 *c* の下三角部分が、更新されたマトリックスの下三角部分によって上書きされる。

?syr2k

対称マトリックスの階数 2 の更新を実行する。

```
call ssyr2k ( uplo, trans, n, k, alpha, a, lda, b, ldb, beta, c, ldc )
call dsyr2k ( uplo, trans, n, k, alpha, a, lda, b, ldb, beta, c, ldc )
call csyr2k ( uplo, trans, n, k, alpha, a, lda, b, ldb, beta, c, ldc )
call zsyr2k ( uplo, trans, n, k, alpha, a, lda, b, ldb, beta, c, ldc )
```

説明

?syr2k ルーチンは、対称マトリックスを使用して階数 2k のマトリックス・マトリックス演算を実行する。この演算は、次のように定義される。

```
c := alpha*a*b' + alpha*b*a' + beta*c
```

または、

```
c := alpha*a'*b + alpha*b'*a + beta*c
```

alpha と *beta* は、スカラーである。

c は、*n* × *n* の対称マトリックスである。

a と *b* は、1 番目の定義では *n* × *k* のマトリックス、2 番目の定義では *k* × *n* のマトリックスである。

入力パラメータ

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

uplo の値	参照されるマトリックス *c* の部分
U または u	*c* の上三角部分だけが参照される。
L または l	*c* の下三角部分だけが参照される。

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

trans の値	実行する演算
N または n	*c*:= *alpha*a*b'+alpha*b*a'+beta*c*
T または t	*c*:= *alpha*a'*b+alpha*b'*a+beta*c*
C または c	*c*:= *alpha*a'*b+alpha*b'*a+beta*c*

n INTEGER。マトリックス *c* の次数を指定する。*n* の値はゼロ以上でなければならない。

k INTEGER。*trans* = 'N' または 'n' と指定した場合は、*k* にはマトリックス *a* と *b* の列数を指定する。また、*trans* = 'T' または 't'、あるいは 'C' または 'c' と指定した場合は、*k* にはマトリックス *a* と *b* の行数を指定する。*k* の値はゼロ以上でなければならない。

alpha REAL (ssyr2k の場合)
DOUBLE PRECISION (dsyr2k の場合)
COMPLEX (csyr2k の場合)
DOUBLE COMPLEX (zsyr2k の場合)
スカラー *alpha* を指定する。

a

REAL (ssyr2k の場合)
DOUBLE PRECISION (dsyr2k の場合)
COMPLEX (csyr2k の場合)
DOUBLE COMPLEX (zsyr2k の場合)

配列 DIMENSION は *(lda, ka)*。*ka* は *trans* = 'N' または 'n' の場合は *k* に、そうでない場合は *n* になる。*trans* = 'N' または 'n' と指定した場合は、このルーチンに入る前に、配列 *a* の先頭の *n* × *k* の部分にマトリックス *a* を格納しなければならない。そうでない場合は、配列 *a* の先頭の *k* × *n* の部分にマトリックス *a* を格納しなければならない。

lda

INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、*a* の第 1 次元を指定する。
trans = 'N' または 'n' の場合は、*lda* は少なくとも max(1, *n*) でなければならない。そうでない場合は、*lda* は少なくとも max(1, *k*) でなければならない。

b

REAL (ssyr2k の場合)
DOUBLE PRECISION (dsyr2k の場合)
COMPLEX (csyr2k の場合)
DOUBLE COMPLEX (zsyr2k の場合)

配列 DIMENSION は *(ldb, kb)*。*kb* は *trans* = 'N' または 'n' の場合は *k* に、そうでない場合は n になる。*trans* = 'N' または 'n' と指定した場合は、このルーチンに入る前に、配列 *b* の先頭の *n* × *k* の部分にマトリックス *b* を格納しなければならない。そうでない場合は、配列 *b* の先頭の *k* × *n* の部分にマトリックス *b* を格納しなければならない。

ldb

INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、*a* の第 1 次元を指定する。
trans = 'N' または 'n' の場合は、*ldb* は少なくとも max(1, *n*) でなければならない。そうでない場合は、*ldb* は少なくとも max(1, *k*) でなければならない。

beta

REAL (ssyr2k の場合)
DOUBLE PRECISION (dsyr2k の場合)
COMPLEX (csyr2k の場合)
DOUBLE COMPLEX (zsyr2k の場合)

スカラー *beta* を指定する。

c

REAL (ssyr2k の場合)
DOUBLE PRECISION (dsyr2k の場合)
COMPLEX (csyr2k の場合)
DOUBLE COMPLEX (zsyr2k の場合)

配列 DIMENSION は (*ldc*, *n*)。*uplo* = 'U' または 'u' と指定した場合は、このルーチンに入る前に、配列 *c* の先頭の *n* × *n* の上三角部分に対称マトリックスの上三角部分を格納しなければならず、また *c* の厳密な意味での下三角部分は参照されない。

uplo = 'L' または 'l' と指定した場合は、このルーチンに入る前に、配列 *c* の先頭の *n* × *n* の下三角部分にエルミート・マトリックスの下三角部分を格納しなければならず、また *c* の厳密な意味での上三角部分は参照されない。

ldc

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

出力パラメータ

c

uplo = 'U' または 'u' と指定した場合は、配列 *c* の上三角部分が、更新されたマトリックスの上三角部分によって上書きされる。

uplo = 'L' または 'l' と指定した場合は、配列 *c* の下三角部分が、更新されたマトリックスの下三角部分によって上書きされる。

?trmm

スカラー・マトリックス・マトリックスの積 (マトリックス・オペランドのいずれかは三角マトリックス) を計算する。

```
call strmm ( side, uplo, trans, diag, m, n, alpha, a, lda, b, ldb )
call dtrmm ( side, uplo, trans, diag, m, n, alpha, a, lda, b, ldb )
call ctrmm ( side, uplo, trans, diag, m, n, alpha, a, lda, b, ldb )
call ztrmm ( side, uplo, trans, diag, m, n, alpha, a, lda, b, ldb )
```

説明

?trmm ルーチンは、三角マトリックスを使用してマトリックス・マトリックス演算を実行する。この演算は、次のように定義される。

b := *alpha**op(*a*)**b*

または、

b := *alpha***b**op(*a*)

alpha は、スカラーである。

b は、*m* × *n* のマトリックスである。

a は、単位または非単位の、上三角マトリックスまたは下三角マトリックスである。

op(*a*) は、op(*a*) = *a* 、op(*a*) = *a*'、または op(*a*) = conjg(*a*') のいずれかである。

入力パラメータ

side CHARACTER*1。次に示すように、演算で op(*a*) が *b* を左側と右側のどちらから掛け合わせるかを指定する。

side の値	実行する演算
L または l	*b* := *alpha*op(*a*)*b*
R または r	*b* := *alpha*b*op(*a*)

uplo CHARACTER*1。次に示すように、マトリックス *a* が上三角マトリックスと下三角マトリックスのどちらであるかを指定する。

uplo の値	マトリックス *a*
U または u	マトリックス *a* は、上三角マトリックス。
L または l	マトリックス *a* は、下三角マトリックス。

transa CHARACTER*1。次に示すように、マトリックスの乗算で使用する op(*a*) の形式を指定する。

transa の値	op(*a*) の形式
N または n	op(*a*) = *a*
T または t	op(*a*) = *a*'
C または c	op(*a*) = conjg(*a*')

diag CHARACTER*1。次に示すように、*a* が単位三角マトリックスであるかどうかを指定する。

diag の値	マトリックス *a*
U または u	マトリックス *a* は、単位三角マトリックスとみなされる。
N または n	マトリックス *a* は、単位三角マトリックスとはみなされない。

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

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

alpha

REAL (strmm の場合)
DOUBLE PRECISION (dtrmm の場合)
COMPLEX (ctrmm の場合)
DOUBLE COMPLEX (ztrmm の場合)

スカラー *alpha* を指定する。*alpha* がゼロの場合は *a* は参照されず、またこのルーチンに入る前に *b* を設定する必要はない。

a

REAL (strmm 場合)
DOUBLE PRECISION (dtrmm の場合)
COMPLEX (ctrmm の場合)
DOUBLE COMPLEX (ztrmm の場合)

配列 DIMENSION (*lda*, *k*)。*k* は *side* = 'L' または 'l' の場合は *m* に、*side* = 'R' または 'r' の場合は *n* になる。*uplo* = 'U' または 'u' と指定した場合は、このルーチンに入る前に、配列 *a* の先頭の *k* × *k* の上三角部分に上三角マトリックスを格納しなければならず、また *a* の厳密な意味での下三角部分は参照されない。

uplo = 'L' または 'l' と指定した場合は、このルーチンに入る前に、配列 *a* の先頭の *k* × *k* の下三角部分に下三角マトリックスを格納しなければならず、また *a* の厳密な意味での上三角部分は参照されない。*diag* = 'U' または 'u' と指定した場合は、*a* の対角成分も参照されず、1 とみなされる。

lda

INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、*a* の第 1 次元を指定する。
side = 'L' または 'l' の場合は、*lda* は少なくとも max(1, *m*) でなければならない。*side* = 'R' または 'r' の場合は、*lda* は少なくとも max(1, *n*) でなければならない。

b

REAL (strmm の場合)
DOUBLE PRECISION (dtrmm の場合)
COMPLEX (ctrmm の場合)
DOUBLE COMPLEX (ztrmm の場合)

配列 DIMENSION は (*ldb*, *n*)。このルーチンに入る前に、配列 *b* の先頭の *m* × *n* の部分にマトリックス *b* を格納しなければならない。

ldb

INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、*b* の第 1 次元を指定する。*ldb* の値は、少なくとも max(1, *m*) でなければならない。

出力パラメータ

b 変換されたマトリックスによって上書きされる。

?trsm

マトリックス式 (マトリックス・オペランドのいずれかは三角マトリックス) を解く。

```
call strsm ( side, uplo, transa, diag, m, n, alpha, a, lda, b, ldb )
call dtrsm ( side, uplo, transa, diag, m, n, alpha, a, lda, b, ldb )
call ctrsm ( side, uplo, transa, diag, m, n, alpha, a, lda, b, ldb )
call ztrsm ( side, uplo, transa, diag, m, n, alpha, a, lda, b, ldb )
```

説明

?trsm ルーチンは、次のマトリックス式のいずれかを解く。

op(*a*)*x = alpha*b,

または、

*x*op(*a*) = *alpha*b*

alpha は、スカラーである。

x と *b* は、*m* × *n* のマトリックスである。

a は、単位または非単位の、三角マトリックスまたは下三角マトリックスである。

op(*a*) は、op(*a*) = a 、op(*a*) = a'、または op(*a*) = conjg(*a*') のいずれかである。

マトリックス *x* は、*b* に上書きされる。

入力パラメータ

side CHARACTER*1。次に示すように、実行する演算で op(*a*) が *x* の左と右のどちらにくるかを指定する。

side の値	実行する演算
L または l	op(*a*)*x = *alpha*b
R または r	*x*op(*a*) = *alpha*b

uplo CHARACTER*1。次に示すように、マトリックス *a* が上三角マトリックスと下三角マトリックスのどちらであるかを指定する。

uplo の値	マトリックス *a*
U または u	マトリックス *a* は、上三角マトリックス。
L または l	マトリックス *a* は、下三角マトリックス。

transa CHARACTER*1。次に示すように、マトリックスの乗算で使用する op(*A*) の形式を指定する。

transa の値	op(*a*) の形式
N または n	op(*a*) = *a*
T または t	op(*a*) = *a*'
C または c	op(*a*) = conjg(*a*')

diag CHARACTER*1。次に示すように、*a* が単位三角マトリックスであるかどうかを指定する。

diag の値	マトリックス *a*
U または u	マトリックス *a* は、単位三角マトリックスとみなされる。
N または n	マトリックス *a* は、単位三角マトリックスとはみなされない。

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

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

alpha

REAL (strsm の場合)
DOUBLE PRECISION (dtrsm の場合)
COMPLEX (ctrsm の場合)
DOUBLE COMPLEX (ztrsm の場合)

スカラー *alpha* を指定する。*alpha* がゼロの場合は *a* は参照されず、またこのルーチンに入る前に *b* を設定する必要はない。

a

REAL (strsm の場合)
DOUBLE PRECISION (dtrsm の場合)
COMPLEX (ctrsm の場合)
DOUBLE COMPLEX (ztrsm の場合)

配列 DIMENSION (*lda*, *k*)。*k* は *side* = 'L' または 'l' の場合は *m* に、*side* = 'R' または 'r' の場合は *n* になる。*uplo* = 'U' または 'u' と指定した場合は、このルーチンに入る前に、配列 *a* の先頭の *k* × *k* の上三角部分に上三角マトリックスを格納しなければならず、また *a* の厳密な意味での下三角部分は参照されない。

uplo = 'L' または 'l' と指定した場合は、このルーチンに入る前に、配列 *a* の先頭の *k* × *k* の下三角部分に下三角マトリックスを格納しなければならず、また *a* の厳密な意味での上三角部分は参照されない。*diag* = 'U' または 'u' と指定した場合は、*a* の対角成分も参照されず、1 とみなされる。

lda

INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、*a* の第 1 次元を指定する。
side = 'L' または 'l' の場合は、*lda* は少なくとも max(1, *m*) でなければならない。*side* = 'R' または 'r' の場合は、*lda* は少なくとも max(1, *n*) でなければならない。

b

REAL (strsm の場合)
DOUBLE PRECISION (dtrsm の場合)
COMPLEX (ctrsm の場合)
DOUBLE COMPLEX (ztrsm の場合)

配列 DIMENSION は (*ldb*, *n*)。このルーチンに入る前に、配列 *b* の先頭の *m* × *n* の部分に右辺のマトリックス *b* を格納しなければならない。

ldb INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、*b* の第 1 次元を指定する。*ldb* の値は、少なくとも max(1, *m*) でなければならない。

出力パラメータ

b 解のマトリックス *x* によって上書きされる。

スパース BLAS のルーチンと関数

この節では、BLAS レベル 1 の拡張であるスパース BLAS について説明する。スパース BLAS は、インテル・マス・カーネル・ライブラリのリリース 2.1 からライブラリに組み込まれたもので、圧縮形式で格納されたスパース・ベクトルに対して多数の一般的なベクトル演算を実行するためのルーチンと関数のグループで構成している。

スパース・ベクトルは、その成分の大半がゼロのベクトルである。スパース BLAS のルーチンと関数は、ベクトルが粗 (スパース) であるのを利用するために特に導入された。この特徴を利用すると、計算時間とメモリを大幅に節約が可能である。ベクトルの非ゼロの成分数が *nz* である場合、スパース BLAS の演算に必要な計算時間は O (*nz*) になる。

スパース BLAS におけるベクトル引数

圧縮形式のスパース・ベクトル。 *a* を配列に格納されているベクトルとし、また *a* の非ゼロの成分を次のように仮定する。

$$a(k_1), a(k_2), a(k_3) \ldots a(k_{nz})$$

nz は、*a* 内の非ゼロの成分の合計数である。

スパース BLAS では、このベクトルは 2 つの FORTRAN 配列、*x* (値) と *indx* (インデックス) を使って圧縮形式で表すことができる。それぞれの配列は、*nz* 個の成分を持つ。

$$x(1)=a(k_1), x(2)=a(k_2), \ldots x(nz)=a(k_{nz}),$$

$$indx(1)=k_1, indx(2)=k_2, \ldots indx(nz)=k_{nz}$$

したがって、スパース・ベクトルは 3 つの組 (*nz*、*x*、*indx*) で完全に表せる。*nz* の値として負の数やゼロをスパース BLAS ルーチンに渡した場合は、サブルーチンはいずれの配列や変数も変更しない。

フル格納形式のベクトル。 スパース BLAS のルーチンでは、1 つの FORTRAN 配列に完全に格納されているベクトル引数 (フル格納形式のベクトル) の使用もできる。*y* がフル格納形式のベクトルとすると、この成分は連続して格納しなければならない。すなわち、最初の成分を *y*(1) に、2 番目の成分を *y*(2) に格納する。これは、BLAS レベル 1 の増分 *incy* = 1 に対応する。フル格納形式のベクトルの増分値がスパース BLAS のルーチンや関数に引数で渡されることはない。

スパース BLAS における命名規則

BLAS ルーチンの場合と同じように、スパース BLAS のサブプログラムの名前の先頭には、関連するデータ・タイプを決定する文字 (単精度と倍精度の実数に対しては s と d、単精度と倍精度の複素数に対しては c と z) が付く。

スパース BLAS のルーチンが「密」なルーチンの拡張である場合は、サブプログラムの名前は、対応する「密」なサブプログラムの名前の末尾に i (*indexed* を表す) が付く。たとえば、スパース BLAS のルーチン saxpyi は、BLAS のルーチン saxpy に、またスパース BLAS の関数 cdotci は BLAS の関数 cdotc に対応する。

スパース BLAS におけるルーチンとデータ・タイプ

表 2-4 に、インテル MKL に導入されているスパース BLAS のルーチンとデータ・タイプを示す。

表 2-4 スパース BLAS のルーチンおよびそのデータ・タイプ

ルーチン / 関数	データ・タイプ	説明
?axpyi	s, d, c, z	スカラー・ベクトルの積にベクトルを加算 (ルーチン)。
?doti	s, d	ドット積 (関数)。
?dotci	c, z	複素共役のドット積 (関数)。
?dotui	c, z	複素非共役のドット積 (関数)。
?gthr	s, d, c, z	フル格納形式のスパース・ベクトルを圧縮形式 : *nz*、*x*、*indx* に集積する。
?gthrz	s, d, c, z	フル格納形式のスパース・ベクトルを圧縮形式に集積し、フル格納形式のベクトルに集積した成分にゼロを割り当てる (ルーチン)。
?roti	s, d	Givens 回転 (ルーチン)。
?sctr	s, d, c, z	ベクトルを圧縮形式からフル格納形式に分散させる (ルーチン)。

スパース・ベクトルで使用できる BLAS ルーチン

以下に示す BLAS レベル 1 のルーチンでは、圧縮形式の配列 *x* を渡した場合にも正しい結果が得られる (増分は、*incx* = 1)。

- ?asum　ベクトル成分の絶対値の合計。
- ?copy　ベクトルのコピー。
- ?nrm2　ベクトルのユークリッド・ノルム。
- ?scal　ベクトルのスケーリング。
- i?amax　最大絶対値を持つ成分のインデックス。あるいは、複素数の場合は、最大の合計 |Re*x(i)*| + |Im*x(i)*| を持つ成分のインデックス。
- i?amin　最小絶対値を持つ成分のインデックス。あるいは、複素数の場合は、最小の合計 |Re*x(i)*| + |Im*x(i)*| を持つ成分のインデックス。

i?amax と i?amin によって返される結果 *i* は、圧縮形式の配列のインデックスとして解釈しなければならない。つまり、最大 (最小) 値が *x(i)* で、これに対応するフル格納形式の配列のインデックスが *indx(i)* になる。

また、?rotg を呼び出して Givens 回転パラメータを計算し、その後でこれらのパラメータをスパース BLAS のルーチン ?roti に渡すのも可能である。

?axpyi

圧縮形式のスパース・ベクトルのスカラー倍を、フル格納形式のベクトルに加える。

```
call saxpyi ( nz, a, x, indx, y )
call daxpyi ( nz, a, x, indx, y )
call caxpyi ( nz, a, x, indx, y )
call zaxpyi ( nz, a, x, indx, y )
```

説明

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

```
y := a*x + Y
```

a は、スカラーである。

(*nz*, *x*, *indx*) は、圧縮形式で格納されたスパース・ベクトルである。

y は、フル格納形式のベクトルである。

?axpyi ルーチンは、インデックスが配列 *indx* に格納されている *y* の成分に対してのみ、参照あるいは変更を行う。*indx* の値は、明確に指定する。

入力パラメータ

nz	INTEGER。*x* と *indx* の成分の数。
a	REAL (saxpyi の場合) DOUBLE PRECISION (daxpyi の場合) COMPLEX (caxpyi の場合) DOUBLE COMPLEX (zaxpyi の場合) スカラー *a* を指定する。
x	REAL (saxpyi の場合) DOUBLE PRECISION (daxpyi の場合) COMPLEX (caxpyi の場合) DOUBLE COMPLEX (zaxpyi の場合) 配列 DIMENSION は *nz* 以上。
indx	INTEGER。*x* の成分に対するインデックスを指定する。 配列 DIMENSION は *nz* 以上。
y	REAL (saxpyi の場合) DOUBLE PRECISION (daxpyi の場合) COMPLEX (caxpyi の場合) DOUBLE COMPLEX (zaxpy の場合) 配列 DIMENSION は少なくとも $\max_i$ (*indx*(*i*))

出力パラメータ

y	更新されたベクトル *y* が格納される。

?doti

圧縮形式の実数型スパース・ベクトルとフル格納形式の実ベクトルのドット積を計算する。

```
res = sdoti ( nz, x, indx, y )
res = ddoti ( nz, x, indx, y )
```

説明

?doti 関数は、次のように定義される *x* と *y* のドット積を返す。

```
x(1)*y(indx(1)) + x(2)*y(indx(2)) +...+ x(nz)*y(indx(nz))
```

3 つの組 (*nz*, *x*, *indx*) には、圧縮形式で格納された実数型スパース・ベクトルを、*y* にはフル格納形式の実ベクトルを定義する。関数は、インデックスが配列 *indx* に格納されている *y* の成分だけを参照する。*indx* の値は、明確に指定する。

入力パラメータ

nz	INTEGER。*x* と *indx* の成分の数。
x	REAL (sdoti の場合) DOUBLE PRECISION (ddoti の場合) 配列 DIMENSION は *nz* 以上。
indx	INTEGER。*x* の成分に対するインデックスを指定する。 配列 DIMENSION は *nz* 以上。
y	REAL (sdoti 場合) DOUBLE PRECISION (ddoti の場合) 配列 DIMENSION は少なくとも $\max_i$ (*indx*(*i*))

出力パラメータ

res	REAL (sdoti の場合) DOUBLE PRECISION (ddoti の場合) *nz* が正の場合は、*x* と *y* のドット積が格納される。そうでない場合は、*res* には 0 が格納される。

?dotci

圧縮形式の複素数型スパース・ベクトルと
フル格納形式の複素ベクトルの共役
ドット積を計算する。

```
res = cdotci ( nz, x, indx, y )
res = zdotci ( nz, x, indx, y )
```

説明

?dotci 関数は、次のように定義される *x* と *y* のドット積を返す。

```
conjg(x(1))*y(indx(1)) + ... + conjg(x(nz))*y(indx(nz))
```

3 つの組 (*nz*, *x*, *indx*) には圧縮形式で格納された複素数型スパース・ベクトルを、*y* にはフル格納形式の複素ベクトルを定義する。関数は、インデックスが配列 *indx* に格納されている *y* の成分だけを参照する。*indx* の値は、明確に指定する。

入力パラメータ

nz	INTEGER。*x* と *indx* の成分の数。
x	COMPLEX (cdotci の場合) DOUBLE COMPLEX (zdotci の場合) 配列 DIMENSION は *nz* 以上。
indx	INTEGER。*x* の成分に対するインデックスを指定する。 配列 DIMENSION は *nz* 以上。
y	COMPLEX (cdotci の場合) DOUBLE COMPLEX (zdotci の場合) 配列 DIMENSION は少なくとも $\max_i$ (*indx*(*i*))

出力パラメータ

res	COMPLEX (cdotci の場合) DOUBLE COMPLEX (zdotci の場合) *nz* が正の場合は、*x* と *y* の共役ドット積が格納される。そうでない場合は、*res* には 0 が格納される。

?dotui

圧縮形式の複素数型スパース・ベクトルと
フル格納形式の複素ベクトルとの
ドット積を計算する。

```
res = cdoTui ( nz, x, indx, y )
res = zdotui ( nz, x, indx, y )
```

説明

?dotui 関数は、次のように定義される *x* と *y* のドット積を返す。

$$x(1)*y(indx(1)) + x(2)*y(indx(2)) + ... + x(nz)*y(indx(nz))$$

3 つの組 (*nz*, *x*, *indx*) には圧縮形式で格納された複素数型スパース・ベクトルを、*y* にはフル格納形式の複素ベクトルを定義する。関数は、インデックスが配列 *indx* に格納されている *y* の成分だけを参照する。*indx* の値は、明確に指定する。

入力パラメータ

nz	INTEGER。*x* と *indx* の成分の数。
x	COMPLEX (cdotui の場合) DOUBLE COMPLEX (zdotui の場合) 配列 DIMENSION は *nz* 以上。
indx	INTEGER。*x* の成分に対するインデックスを指定する。 配列 DIMENSION は *nz* 以上。
y	COMPLEX (cdotui の場合) DOUBLE COMPLEX (zdotui の場合) 配列 DIMENSION は少なくとも $\max_i(indx(i))$

出力パラメータ

res	COMPLEX (cdotui の場合) DOUBLE COMPLEX (zdotui の場合) *nz* が正の場合は、*x* と *y* のドット積が格納される。そうでない場合は、*res* には 0 が格納される。

?gthr

フル格納形式のスパース・ベクトルの
成分を圧縮形式に集積する。

```
call sgthr ( nz, y, x, indx )
call dgthr ( nz, y, x, indx )
call cgthr ( nz, y, x, indx )
call zgthr ( nz, y, x, indx )
```

説明

?gthr ルーチンは、フル格納形式のスパース・ベクトル *y* の指定の成分を、圧縮形式 (*nz*, *x*, *indx*) に集積する。ルーチンは、インデックスが配列 *indx* に格納されている *y* の成分だけを参照する。

```
x(i) = y(indx(i)),(i=1,2,...nz)
```

入力パラメータ

nz INTEGER。集積する *y* の成分の数。

indx INTEGER。集積する成分のインデックスを指定する。
配列 DIMENSION は *nz* 以上。

y REAL (sgthr の場合)
DOUBLE PRECISION (dgthr の場合)
COMPLEX (cgthr の場合)
DOUBLE COMPLEX (zgthr の場合)
配列 DIMENSION は少なくとも $\max_i$ (*indx*(*i*))

出力パラメータ

x REAL (sgthr の場合)
DOUBLE PRECISION (dgthr の場合)
COMPLEX (cgthr の場合)
DOUBLE COMPLEX (zgthr の場合)
配列 DIMENSION は *nz* 以上。
圧縮形式に変換されたベクトルが格納される。

?gthrz

スパース・ベクトルの成分を圧縮形式に
集積し、これらの成分をゼロで
置き換える。

```
call sgthrz ( nz, y, x, indx )
call dgthrz ( nz, y, x, indx )
call cgthrz ( nz, y, x, indx )
call zgthrz ( nz, y, x, indx )
```

説明

?gthrz ルーチンは、配列 *indx* で指定したインデックスを持つ成分に対して、フル格納形式のベクトル *y* から圧縮形式 (*nz*, *x*, *indx*) に集積し、その集積した *y* の成分をゼロで上書きする。
y の指定されていない成分に対しては、参照も変更も行わない (?gthr も参照)。

入力パラメータ

nz	INTEGER。*y* の成分の数。
indx	INTEGER。集積する成分のインデックスを指定する。 配列 DIMENSION は *nz* 以上。
y	REAL (sgthrz の場合) DOUBLE PRECISION (dgthrz の場合) COMPLEX (cgthrz の場合) DOUBLE COMPLEX (zgthrz の場合) 配列 DIMENSION は少なくとも $\max_i$ (*indx*(*i*))

出力パラメータ

x	REAL (sgthrz の場合) DOUBLE PRECISION (dgthrz の場合) COMPLEX (cgthrz の場合) DOUBLE COMPLEX (zgthrz の場合) 配列 DIMENSION は *nz* 以上。 圧縮形式に変換されたベクトルが格納される。
y	更新されたベクトル *y*

?roti

一方が圧縮形式であるスパース・
ベクトルに Givens 回転を適用する。

```
call sroti ( nz, x, indx, y, c, s )
call droti ( nz, x, indx, y, c, s )
```

説明

?roti ルーチンは、2 つの実数ベクトル、*x* (圧縮形式 *nz*, *x*, *indx*) と *y* (フル格納形式) の成分に Givens 回転を適用する。

```
x(i) = c*x(i) + s*y(indx(i))
y(indx(i)) = c*y(indx(i)) - s*x(i)
```

ルーチンは、インデックスが配列 *indx* に格納されている *y* の成分だけを参照する。*indx* の値は、明確に指定する。

入力パラメータ

nz	INTEGER。*x* と *indx* の成分の数。
x	REAL (sroti の場合) DOUBLE PRECISION (droti の場合) 配列 DIMENSION は *nz* 以上。
indx	INTEGER。*x* の成分に対するインデックスを指定する。 配列 DIMENSION は *nz* 以上。
y	REAL (sroti の場合) DOUBLE PRECISION (droti の場合) 配列 DIMENSION は少なくとも $\max_i$ (*indx*(*i*))
c	スカラー。REAL (sroti の場合) DOUBLE PRECISION (droti の場合)
s	スカラー。REAL (sroti の場合) DOUBLE PRECISION (droti の場合)

出力パラメータ

x および *y*	更新された配列。

?sctr

圧縮形式のスパース・ベクトルを
フル格納形式に変換する。

```
call ssctr ( nz, x, indx, y )
call dsctr ( nz, x, indx, y )
call csctr ( nz, x, indx, y )
call zsctr ( nz, x, indx, y )
```

説明

?sctr ルーチンは、圧縮形式のスパース・ベクトル (*nz*, *x*, *indx*) の成分を、フル格納形式のベクトル *y* に分散する。ルーチンは、インデックスが配列 *indx* に格納されている成分だけを変更する。

y(*indx*(*i*)) = x(*i*),(*i*=1,2,...*nz*)

入力パラメータ

nz	INTEGER。分散する *x* の成分の数。
indx	INTEGER。分散する成分のインデックスを指定する。 配列 DIMENSION は *nz* 以上。
x	REAL (ssctr の場合) DOUBLE PRECISION (dsctr の場合) COMPLEX (csctr の場合) DOUBLE COMPLEX (zsctr の場合) 配列 DIMENSION は *nz* 以上。 フル格納形式に変換するベクトルを格納する。

出力パラメータ

y	REAL (ssctr の場合) DOUBLE PRECISION (dsctr の場合) COMPLEX (csctr の場合) DOUBLE COMPLEX (zsctr の場合) 配列 DIMENSION は少なくとも $\max_i$ (*indx*(*i*)) 更新された成分を持つベクトル *y* が格納される。

高速フーリエ変換

3

本章では、インテル® MKL に導入された高速フーリエ変換 (FFT) ルーチンについて説明する。インテル® マス・カーネル・ライブラリに含まれる FFT ルーチンは、1 次元と 2 次元の 2 つのクラスで構成される。1 次元と 2 次元のルーチンはいずれも、キャッシュ・メモリを効果的に使用するように最適化している。すべてのルーチンは 2 の累乗長さの変換に対応する。

インテル・マス・カーネル・ライブラリの離散フーリエ変換関数の詳細は、本マニュアルの「アドバンスト DFT 関数」を参照のこと。

インテル・マス・カーネル・ライブラリは本章で述べるように FFT インターフェイスを引き続きサポートしているが、従来のユーザ・アプリケーションは新しい DFT 関数に移行することが望ましい。FFT ルーチンと異なり DFT ルーチンは、最大 7 次元の変換と、2 のべき乗混合基数以外の変換長をサポートする。

この章は、以下の節で構成する。

- 1 次元 FFT
- 2 次元 FFT

それぞれの節では、FFT を 3 つのグループに分けて説明する。

1 次元 FFT

1 次元 FFT は、以下のグループに分けられる。

- 複素数から複素数への変換
- 実数から複素数への変換
- 複素数から実数への変換

すべての 1 次元 FFT は、インプレースである。変換の長さは 2 の累乗でなければならない。複素数から複素数への変換ルーチンは、複素ベクトルの順方向と逆方向の変換を実行する。実数から複素数への変換ルーチンは、実ベクトルの順方向の変換を実行する。複素数から実数への変換ルーチンは、実数配列にパックされた複素共役対称ベクトルの逆方向の変換を実行する。

データ格納の型

FFT の各グループは、同じような機能を持つ 2 つのセットの FFT で構成される。1 つのセットは Fortran インターフェイスで使用し、もう 1 つのセットは C インターフェイスで使用する。Fortran インターフェイス向けの FFT は、複素数データを FORTRAN の複素数データ・タイプとして格納する。C インターフェイス向けの FFT は、複素数データを実数部と虚数部の浮動小数点配列として別々に格納する。これらのセットは、各セット内の FFT の名前で区別される。C インターフェイスで使用する FFT の名前には、FFT の Fortran 名の最後に文字 "c" が付けられている。たとえば、対応する C インターフェイス・ルーチン用の `cfft1d/zfft1d` FFT の名前は、`cfft1dc/zfft1dc` となる。C タイプのデータ項目の名前はすべて、小文字である。

表 3-1 に、1 次元 FFT ルーチンのグループと、それらに関連するデータ・タイプを示す。

表 3-1　1 次元 FET：名前およびデータ・タイプ

グループ	FORTRAN の複素数データとして格納	C の実数データとして格納	データ・タイプ	説明
複素数から複素数へ	cfft1d/ zfft1d	cfft1dc/ zfft1dc	c, z	複素数データから複素数データへの変換を実行する。
実数から複素数へ	scfft1d/ dzfft1d	scfft1dc/ dzfft1dc	sc, dz	実数データから複素数データへの順方向の変換を実行する。`scfft1d/dzfft1d` と `scfft1dc/dzfft1dc` の FFT を補足する。
複素数から実数へ	csfft1d/ zdfft1d	csfft1dc/ zdfft1dc	cs, zd	複素数データから実数データへの逆方向の変換を実行する。`csfft1d/zdfft1d` と `csfft1dc/zdfft1dc` の FFT を補足する。

データ構造の要件

C インターフェイスでは、複素数から複素数への変換ルーチンでは、実数部と虚数部それぞれに 独立した float 配列が必要である。実数から複素数、複素数から実数への変換ルーチンでは、単一の float 入力 / 出力配列が必要である。

C インターフェイスでは、値で渡されるスカラー値が必要である。

すべての変換は、変換係数を格納するためのメモリが余分に必要になる。同次元の複数の FFT を実行する場合は、係数の表を一度だけ作成し、それ以後はすべての FFT でこの表を使用しなければならない。FFT ごとに表を繰り返し作成するのではなく同じ表を使用れば、性能が大幅に向上する。

複素数から複素数への 1 次元 FFT

複素数から複素数への各ルーチンでは、複素ベクトルの順方向または逆方向の FFT を計算する。
順方向の FFT は、次の式に従って計算される。

$$z_j = \sum_{k=0}^{n-1} r_k * w^{-j*k}, \quad 0 \le j \le n-1$$

逆方向の FFT は、次の式に従って計算される。

$$r_j = \frac{1}{n} \sum_{k=0}^{n-1} z_k * w^{j*k}, \quad 0 \le j \le n-1$$

$w = \exp\left[\frac{2\pi i}{n}\right]$ で、i は虚数単位である。

複素数から複素数へのルーチンが実行する演算は、各ルーチンが使用する *isign* パラメータの値によって決まる。

isign = -1 の場合は、順方向の FFT を実行する。入力と出力は通常の順序である。

isign = +1 の場合は、逆方向の FFT を実行する。入力と出力は通常の順序である。

isign = -2 の場合は、順方向の FFT を実行する。入力は通常の順序、出力はビット反転した順序である。

isign = +2 の場合は、逆方向の FFT を実行する。入力はビット反転した順序、出力は通常の順序である。

isign = *0* の場合は、順方向の FFT と逆方向の FFT のいずれも、FFT 係数を初期化する。

上記の式は、表 3-1 に示されているすべてのデータ・タイプで、すべての FFT に適用される。

所定の長さを持つ順方向または逆方向の FFT を計算するには、まず、*isign* = 0 に設定して関数を呼び出し、係数を初期化しなければならない。係数の初期化後は、*isign* = +1, -1, +2, -2 に設定してこの関数を呼び出し、同じ長さの変換を何度でも計算できる。

cfft1d/zfft1d

Fortran インターフェイス・ルーチン。
複素ベクトルの順方向または逆方向の
FFT を計算する (インプレース)。

```
call cfft1d ( r, n, isign, wsave )
call zfft1d ( r, n, isign, wsave )
```

説明

cfft1d/zfft1d ルーチンが実行する演算は、*isign* の値によって決まる。上記の「複素数から複素数への 1 次元 FFT」の演算式を参照。

入力パラメータ

r	COMPLEX (cfft1d の場合) DOUBLE COMPLEX (zfft1d の場合) 配列 DIMENSION は少なくとも (*n*)。変換の対象となる複素ベクトルを格納する。*isign* = 0 の場合は、参照されない。
n	INTEGER。変換の長さ。*n* は、2 の累乗でなければならない。
isign	INTEGER。実行する演算のタイプを指示するフラグ。*isign* = 0 の場合は、係数 *wsave* が初期化される。

isign = -1 の場合は、順方向の FFT を実行する。入力と出力は通常の順序である。
isign = +1 の場合は、逆方向の FFT を実行する。入力と出力は通常の順序である。
isign = -2 の場合は、順方向の FFT を実行する。入力は通常の順序、出力はビット反転した順序である。
isign = +2 の場合は、逆方向の FFT を実行する。入力はビット反転した順序、出力は通常の順序である。

wsave COMPLEX (cfft1d の場合)
DOUBLE COMPLEX (zfft1d の場合)
配列 DIMENSION は少なくとも ((3*n)/2)
isign = 0 の場合は、*wsave* は出力パラメータになる。それ以外の場合は、*wsave* には、*isign* = 0 と指定してこのルーチンを呼び出したときに初期化された FFT 係数を格納する。

出力パラメータ

r *isign* に従って、変換の結果得られた複素数データが格納される。*isign* = 0 の場合は、変わらない。

wsave *isign* = 0 に従って、*wsave* には初期化された FFT の係数が格納される。そうでない場合は、*wsave* は変わらない。

cfft1dc/zfft1dc

C インターフェイス・ルーチン。複素ベクトルの順方向または逆方向の FFT を計算する (インプレース)。

```
void cfft1dc (float* r, float* i, int n, int isign, float* wsave)
void zfft1dc (double* r, double* i, int n, int isign, double* wsave)
```

説明

cfft1dc/zfft1dc ルーチンが実行する演算は、*isign* の値によって決まる。上記の「複素数から複素数への 1 次元 FFT」の演算式を参照。

入力パラメータ

r
float* (cfft1dc の場合)
double* (zfft1dc の場合)
少なくとも (*n*) のサイズを持つ配列へのポインタ。変換の対象となる複素ベクトルの実数部を格納する。
isign = 0 の場合は、参照されない。

i
float* (cfft1dc の場合)
double* (zfft1dc の場合)
少なくとも (*n*) のサイズを持つ配列へのポインタ。変換の対象となる複素ベクトルの虚数部を格納する。
isign = 0 の場合は、参照されない。

n
int。変換の長さ。*n* は、2 の累乗でなければならない。

isign
int。実行する演算のタイプを指示するフラグ。
isign = 0 の場合は、係数 *wsave* が初期化される。
isign = -1 の場合は、順方向の FFT を実行する。入力と出力は通常の順序である。
isign = +1 の場合は、逆方向の FFT を実行する。入力と出力は通常の順序である。
isign = -2 の場合は、順方向の FFT を実行する。入力は通常の順序、出力はビット反転した順序である。
isign = +2 の場合は、逆方向の FFT を実行する。入力はビット反転した順序、出力は通常の順序である。

wsave
float* (cfft1dc の場合)
double* (zfft1dc の場合)
少なくとも (3*n) のサイズを持つ配列へのポインタ。
isign = 0 の場合は、*wsave* は出力パラメータになる。それ以外の場合は、*wsave* には、*isign* = 0 と指定してこのルーチンを呼び出したときに初期化された FFT 係数を格納する。

出力パラメータ

r
isign に従って、変換後の実数部が格納される。
isign = 0 の場合は、変わらない。

i	*isign* に従って、変換後の虚数部が格納される。*isign* = 0 の場合は、変わらない。
wsave	*isign* = 0 の場合は、*wsave* には初期化された FFT の係数が格納される。そうでない場合は、*wsave* は変わらない。

実数から複素数への 1 次元 FFT

実数から複素数へのルーチンは、いずれも次の式に従って、実数の入力ベクトルの順方向の FFT を計算する。

$$z_j = \sum_{k=0}^{n-1} t_k * w^{-j*k}, \quad 0 \le j \le n-1$$

tk = cmplx(r_k, 0) の場合。r_k は実数の入力ベクトルで、$0 \le k \le n-1$ である。
数値演算の結果 z_j, $0 \le j \le n-1$ は、複素共役対称ベクトルになる。z(n/2+i) = conjg(z(n/2-i))、$1 \le i \le n/2-1$ 、また z(0) と z(n/2) は実数値である。

この複素共役対称 (CCS) ベクトルは、サイズ (n/2+1) の複素数の配列またはサイズ (n+2) の実数の配列として格納される。CCS 形式のデータ格納は、Fortran インターフェイス向けルーチンと C インターフェイス向けルーチンについて別々に後で定義する。

表 3-2 は、すべての虚数成分が 0 である長さ n=8 のベクトルに対して複素数から複素数への cfft1d/zfft1d の FFT を実行した場合と、同じベクトルに対して実数から複素数への scfft1d/zdfft1d の FFT を実行した場合について、両方の効果を比較したものである。後者の方法のメリットは、必要なデータ格納量が半分で済み、虚数部を 0 にする必要がないことである。最後の 2 つの列は、複素共役対称ベクトルを CCS 形式で格納する、サイズ (n+2) の実数の配列に格納される。

所定の長さを持つ順方向の FFT を計算するには、まず、*isign* = 0 に設定して該当するルーチンを呼び出し、係数を初期化しなければならない。係数の初期化後は、*isign* に 0 以外の値を指定してルーチンを呼び出し、同じ長さの実数から複素数への変換と複素数から実数への変換を何度でも計算できる。

表 3-2 複素数から複素数への FFT と実数から複素数への FFT の格納効果の比較

入力ベクトル			出力ベクトル			
cfft1d		scfft1d	cfft1d		scfft1d	
複素数データ		実数データ	複素数データ		実数データ	
実数部	虚数部		実数部	虚数部	(実数部)	(虚数部)
0.841471	0.000000	0.841471	1.543091	0.000000	1.543091	0.000000
0.909297	0.000000	0.909297	3.875664	0.910042	3.875664	0.910042
0.141120	0.000000	0.141120	-0.915560	-0.397326	-0.915560	-0.397326
-0.756802	0.000000	-0.756802	-0.274874	-0.121691	-0.274874	-0.121691
-0.958924	0.000000	-0.958924	-0.181784	0.000000	-0.181784	0.000000
-0.279415	0.000000	-0.279415	-0.274874	0.121691		
0.656987	0.000000	0.656987	-0.915560	0.397326		
0.989358	0.000000	0.989358	3.875664	-0.910042		

scfft1d/dzfft1d

Fortran インターフェイス・ルーチン。
実数ベクトルの順方向の FFT を計算し、
結果の複素共役対称ベクトルを CCS 形式で
表現する (インプレース)。

```
call scfft1d ( r, n, isign, wsave )
call dzfft1d ( r, n, isign, wsave )
```

説明

scfft1d/dzfft1d ルーチンが実行する演算は、*isign* の値によって決まる。上記の「実数から複素数への 1 次元 FFT」の演算式を参照。これらのルーチンは、複素数から実数への変換ルーチンである csfft1d/zdfft1d を補足する。

入力パラメータ

r
REAL (scfft1d の場合)
DOUBLE PRECISION (dzfft1d の場合)
配列 DIMENSION は少なくとも (*n*+2)。最初の *n* 個の要素には、変換される入力ベクトルを格納する。出力では、要素 *r*(*n*+1) と *r*(*n*+2) が使用される。
isign = 0 の場合は、配列 *r* は参照されない。

n
INTEGER。変換の長さ。*n* は、2 の累乗でなければならない。

isign
INTEGER。実行する演算のタイプを指示するフラグ。
isign が 0 の場合は、係数 *wsave* が初期化される。
isign が 0 でない場合は、順方向の FFT を実行する。

wsave
REAL (scfft1d の場合)
DOUBLE PRECISION (dzfft1d の場合)
配列 DIMENSION は少なくとも (2*n+4)
isign = 0 の場合は、*wsave* には出力データが格納される。そうでない場合は、*wsave* には、FFT の実行に必要な係数 (このルーチンか、このルーチンを補足する複素数から実数への FFT ルーチンを直前に呼び出したときに初期化されたもの) を格納する。

出力パラメータ

r
isign = 0 の場合は、*r* は変わらない。*isign* が 0 でない場合は、出力される実数値の配列 *r*(1:*n*+2) には、Fortran インターフェイス向け CCS 形式にパックされた複素共役対称ベクトル *z*(1:*n*) を格納する。
次の表は、出力される配列と格納されるベクトルの関係を示している。

r(1)	r(2)	r(3)	r(4)	...	r(n-1)	r(n)	r(n+1)	r(n+2)
z(1)	0	REz(2)	IMz(2)	...	REz(n/2)	IMz(n/2)	z(n/2+1)	0

完全複素ベクトル z(1:n) が次のように定義される場合、

z(i) = cmplx(r(2*i-1), r(2*i)), 1 ≤ i ≤ n/2+1,

z(n/2+i) = conjg(z(n/2+2-i)), 2 ≤ i ≤ n/2

z(1:n) は、実数の入力ベクトル r(1:n) の順方向の FFT になる。

wsave isign = 0 の場合は、wsave には、呼び出されたルーチンに必要な係数が格納される。そうでない場合は、wsave は変わらない。

scfft1dc/dzfft1dc

C インターフェイス・ルーチン。実数ベクトルの順方向の FFT を計算し、結果の複素共役対称ベクトルを CCS 形式で表現する (インプレース)。

```
void scfft1dc ( float* r, int n, int isign, float* wsave );
void dzfft1dc ( double* r, int n, int isign, double* wsave );
```

説明

scfft1dc/dzfft1dc ルーチンが実行する演算は、isign の値によって決まる。上記の「実数から複素数への 1 次元 FFT」の演算式を参照。これらのルーチンは、複素数から実数への変換ルーチンである csfft1dc/zdfft1dc を補足する。

入力パラメータ

r float* (scfft1dc の場合)
double* (dzfft1dc の場合)
少なくとも (*n*+2) のサイズを持つ配列へのポインタ。最初の *n* 個の要素には、変換される入力ベクトルを格納する。*isign* = 0 の場合は、配列 *r* は参照されない。

n int。変換の長さ。*n* は、2 の累乗でなければならない。

isign int。実行する演算のタイプを指示するフラグ。
isign が 0 の場合は、係数 *wsave* が初期化される。
isign が 0 でない場合は、順方向の FFT を実行する。

wsave float* (scfft1dc の場合)
double* (dzfft1dc の場合)
少なくとも (2**n*+4) のサイズを持つ配列へのポインタ。
isign = 0 の場合は、*wsave* には出力データを格納する。それ以外の場合は、*wsave* には、このルーチンまたはこのルーチンと対になる複素数から実数への FFT ルーチンを呼び出したときに初期化された、FFT の実行に必要な係数を格納する。

出力パラメータ

r *isign* = 0 の場合は、*r* は変わらない。*isign* が 0 でない場合は、出力される実数値の配列 *r*(0:*n*+1) には、C インターフェイス向け CCS 形式にパックされた複素共役対称ベクトル *z*(0:*n*-1) を格納する。
次の表は、出力される配列と格納されるベクトルの関係を示してる。

r(0)	*r*(1)	*r*(2)	...	*r*(*n*/2)	*r*(*n*/2+1)	*r*(*n*/2+2)	...	*r*(*n*)	*r*(*n*+1)
z(0)	REz(1)	REz(2)	...	*z*(*n*/2)	0	IMz(1)	...	IMz(*n*/2-1)	0

完全複素ベクトル *z*(0:*n*-1) が次のように定義される場合、
z(i) = cmplx(*r*(i), *r*(n/2+1+i)), 0 ≤ i ≤ *n*/2,

	z(n/2+i) = conjg(z(n/2-i)), 1 ≤ i ≤ n/2-1 z(0:n-1) は、長さ n の実数の入力ベクトルの順方向の FFT になる。
wsave	isign = 0 の場合は、wsave には呼び出されたルーチンに必要な係数が格納される。そうでない場合は、wsave は変わらない。

複素数から実数への 1 次元 FFT

複素数から実数へのルーチンは、いずれも次の式に従って、1 次元逆 FFT を計算する。

$$t_j = \frac{1}{n}\sum_{k=0}^{n-1} z_k * w^{j*k}, \quad 0 \le j \le n-1$$

数値演算の入力は、複素共役対称ベクトル z_j、$0 \le j \le n-1$ である。z(n/2+i) = conjg(z(n/2-i))、$1 \le i \le n/2-1$、また z(0) と z(n/2) は実数値である。

数値演算の結果は、t_j = cmplx(rj, 0) である。r_j は実数ベクトルで、$0 \le j \le n-1$ である。

複素数から実数への変換ルーチンの入力は、サイズ (n+2) の実数の配列である。この配列には、複素共役対称ベクトル z(0:n-1) が CCS 形式で格納される (上記の「実数から複素数への 1 次元 FFT」を参照)。

複素数から実数への変換ルーチンの出力は、サイズ n の実数ベクトルである。

表 3-3 は、入力ベクトルと出力ベクトルを逆にした点を除いて、表 3-2 と同じ内容であるす。複素数から実数へのルーチンでは、最後の 2 つの列は、複素共役対称ベクトルを CCS 形式で格納する、サイズ (n+2) の入力実数配列に格納される。

所定の長さを持つ逆方向の FFT を計算するには、まず、isign = 0 に設定して使用するルーチンを呼び出し、係数を初期化しなければならない。係数の初期化後は、isign に 0 以外の値を設定して適切なルーチンを呼び出し、同じ長さの実数から複素数への変換と複素数から実数への変換を何度でも計算できる。

表 3-3　複素数から実数への FFT と複素数から複素数への FFT の格納効果の比較

出力ベクトル			入力ベクトル			
cfft1d		csfft1d	cfft1d		csfft1d	
複素数データ		実数データ	複素数データ		実数データ	
実数部	虚数部		実数部	虚数部	(実数部)	(虚数部)
0.841471	0.000000	0.841471	1.543091	0.000000	1.543091	0.000000
0.909297	0.000000	0.909297	3.875664	0.910042	3.875664	0.910042
0.141120	0.000000	0.141120	-0.915560	-0.397326	-0.915560	-0.397326
-0.756802	0.000000	-0.756802	-0.274874	-0.121691	-0.274874	-0.121691
-0.958924	0.000000	-0.958924	-0.181784	0.000000	-0.181784	0.000000
-0.279415	0.000000	-0.279415	-0.274874	0.121691		
0.656987	0.000000	0.656987	-0.915560	0.397326		
0.989358	0.000000	0.989358	3.875664	-0.910042		

csfft1d/zdfft1d

Fortran インターフェイス・ルーチン。
CCS 形式にパックされた複素共役対称ベクトルの逆 FFT を計算する (インプレース)。

```
call csfft1d ( r, n, isign, wsave )
call zdfft1d ( r, n, isign, wsave )
```

説明

csfft1d/zdfft1d ルーチンが実行する演算は、*isign* の値によって決まる。上記の「複素数から実数への 1 次元 FFT」の演算式を参照。

これらのルーチンは、実数から複素数への変換ルーチンである scfft1d/zdfft1d を補足する。

入力パラメータ

r

REAL (csfft1d の場合)
DOUBLE PRECISION (zdfft1d の場合)
配列 DIMENSION は少なくとも (*n*+2)
isign = 0 の場合は、参照されない。
isign が 0 でない場合は、*r*(1:*n*+2) には、Fortran インターフェイス向け CCS 形式にパックされた複素共役対称ベクトルを格納する。
次の表は、出力される配列と格納されるベクトルの関係を示す。

r(1)	*r*(2)	*r*(3)	*r*(4)	...	*r*(*n*-1)	*r*(*n*)	*r*(*n*+1)	*r*(*n*+2)
z(1)	0	RE*z*(2)	IM*z*(2)	...	RE*z*(*n*/2)	IM*z*(*n*/2)	*z*(*n*/2+1)	0

完全複素ベクトル *z*(1:*n*) が次のように定義される場合、
z(i) = cmplx(*r*(2*i-1), *r*(2*i)), 1 ≤ i ≤ *n*/2+1,
z(*n*/2+i) = conjg(*z*(*n*/2+2-i)), 2 ≤ i ≤ *n*/2
変換の実行後、*r*(1:*n*) には、複素共役対称ベクトル *z*(1:*n*) の逆 FFT が格納される。

n

INTEGER。変換の長さ。*n* は、2 の累乗でなければならない。

isign

INTEGER。実行する演算のタイプを指示するフラグ。
isign が 0 の場合は、係数 wsave が初期化される。
isign が 0 でない場合は、逆方向の FFT を実行する。

wsave

REAL (csfft1d の場合)
DOUBLE PRECISION (zdfft1d の場合)
配列 DIMENSION は少なくとも (2*n+4)
isign = 0 の場合は、*wsave* には出力データが格納される。そうでない場合は、*wsave* には、FFT の実行に必要な係数 (このルーチンか、このルーチンを補足する実数から複素数への FFT ルーチンを直前に呼び出したときに初期化されたもの) を格納する。

出力パラメータ

r	*isign* が 0 でない場合は、*r*(1:*n*) は、複素共役対称ベクトル *z*(1:*n*) の逆 FFT の実数の結果になる。 *isign* = 0 の場合は、変わらない。
wsave	*isign* = 0 の場合は、*wsave* には、呼び出されたルーチンに必要な係数が格納される。そうでない場合は、*wsave* は変わらない。

csfft1dc/zdfft1dc

C インターフェイス・ルーチン。CCS 形式にパックされた複素共役対称ベクトルの逆 FFT を計算する (インプレース)。

```
void csfft1dc ( float* r, int n, int isign, float* wsave )
void zdfft1dc ( double* r, int n, int isign, double* wsave )
```

説明

csfft1dc/zdfft1dc ルーチンが実行する演算は、*isign* の値によって決まる。上記の「複素数から実数への 1 次元 FFT」の演算式を参照。

これらのルーチンは、実数から複素数への変換ルーチンである scfft1dc/zdfft1dc を補足する。

入力パラメータ

r	float* (csfft1dc の場合) double* (zdfft1dc の場合)

少なくとも、(*n*+2) のサイズを持つ配列へのポインタ。*isign* = 0 の場合は、参照されない。

isign が 0 でない場合は、*r*(0:*n*+1) には、C インターフェイス向け CCS 形式にパックされた複素共役対称ベクトルを格納する。
次の表は、出力される配列と格納されるベクトルの関係を示す。

r(0)	*r*(1)	*r*(2)	...	*r*(*n*/2)	*r*(*n*/2+1)	*r*(*n*/2+2)	...	*r*(*n*)	*r*(*n*+1)
z(0)	REz(1)	REz(2)	...	*z*(*n*/2)	0	IMz(1)	...	IMz(*n*/2-1)	0

完全複素ベクトル *z*(0:*n*-1) が次のように定義される場合、
z(i) = cmplx(*r*(i), *r*(n/2+1+i)), 0 ≤ i ≤ *n*/2,
z(*n*/2+i) = conjg(*z*(*n*/2-i)), 1 ≤ i ≤ *n*/2-1
変換の実行後、*r*(0:*n*-1) は、複素共役対称ベクトル *z*(0:*n*-1) の逆 FFT になる。

n int。変換の長さ。*n* は、2 の累乗でなければならない。

isign int。実行する演算のタイプを指示するフラグ。
isign = 0 の場合は、係数 *wsave* が初期化される。
isign が 0 でない場合は、逆方向の FFT が実行される。

wsave float* (csfft1dc の場合)
double* (zdfft1dc の場合)
少なくとも (2*n+4) のサイズを持つ配列へのポインタ。
isign = 0 の場合は、*wsave* には出力データを格納する。それ以外の場合は、*wsave* には、このルーチンまたはこのルーチンと対になる実数から複素数への FFT ルーチンを呼び出したときに初期化された、FFT の実行に必要な係数を格納する。

出力パラメータ

r *isign* が 0 でない場合は、*r*(0:*n*-1) は、複素共役対称ベクトル *z*(0:*n*-1) の逆 FFT の実数の結果になる。
isign = 0 の場合は、変わらない。

wsave *isign* = 0 の場合は、*wsave* は呼び出されたルーチンに必要な係数が格納される。そうでない場合は、*wsave* は変わらない。

2 次元 FFT

2 次元 FFT は、機能的には 1 次元 FFT と同じである。2 次元 FFT は、以下のグループに分けられる。

- 複素数から複素数への変換
- 実数から複素数への変換
- 複素数から実数への変換

すべての 2 次元 FFT は、インプレースである。変換の長さは 2 の累乗でなければならない。複素数から複素数への変換ルーチンは、複素マトリックスの順方向と逆方向の変換を実行する。実数から複素数への変換ルーチンは、実マトリックスの順方向の変換を実行する。複素数から実数への変換ルーチンは、実数配列にパックされた複素共役対称マトリックスの逆方向の変換を実行する。

2 次元 FFT に対する命名規則は、すべてのケースで "1d" が "2d" に置き換わることを除けば、1 次元 FFT に対する規則と同じである。表 3-4 に、2 次元 FFT ルーチンのグループと、それらに関連するデータ・タイプを示す。

表 3-4　2 次元 FET：名前およびデータ・タイプ

グループ	FORTRAN の複素数データとして格納	C の実数データとして格納	データ・タイプ	説明
複素数から複素数へ	cfft2d/ zfft2d	cfft2dc/ zfft2dc	c, z	複素数データから複素数データへの変換を実行する。
実数から複素数へ	scfft2d/ dzfft2d	scfft2dc/ dzfft2dc	sc, dz	実数データから複素数データへの順方向の変換を実行する。csfft2d/zdfft2d と csfft2dc/zdfft2dcのFFTを補足する。
複素数から実数へ	csfft2d/ zdfft2d	csfft2dc/ zdfft2dc	cs, zd	複素数データから実数データへの逆方向の変換を実行する。scfft2d/dzfft2d と scfft2dc/dzfft2dcのFFTを補足する。

C インターフェイスでは、値で渡されるスカラー値が必要である。2 次元 FFT が 1 次元 FFT と大きく異なるのは、アプリケーションで変換係数用の格納域を用意する必要がないことである。

データ格納の型とデータ構造の要件は、1 次元 FFT に対するものと同じである。詳細は、本章の最初にある「データ格納の型」と「データ構造の要件」の項を参照。

複素数から複素数への 2 次元 FFT

複素数から複素数への各ルーチンでは、複素マトリックスの順方向または逆方向の FFT をインプレースで計算する。

順方向の FFT は、次の式に従って計算される。

$$z_{i,j} = \sum_{k=0}^{m-1} \sum_{l=0}^{n-1} r_{k,l} * w_m^{-i*k} * w_n^{-j*l}, \quad 0 \le i \le m-1, \quad 0 \le j \le n-1$$

逆方向の FFT は、次の式に従って計算される。

$$r_{i,j} = \frac{1}{m*n} \sum_{k=0}^{m-1} \sum_{l=0}^{n-1} z_{k,l} * w_m^{i*k} * w_n^{j*l}, \quad 0 \le i \le m-1, \quad 0 \le j \le n-1$$

ここで、$w_m = \exp\left[\frac{2\pi i}{m}\right]$、$w_n = \exp\left[\frac{2\pi i}{n}\right]$ で、i は虚数単位である。

複素数から複素数へのルーチンが実行する演算は、*isign* パラメータの値によって決まる。

isign = -1 の場合は、順方向の FFT を実行する。入力と出力は通常の順序である。
isign = +1 の場合は、逆方向の FFT を実行する。入力と出力は通常の順序である。
isign = -2 の場合は、順方向の FFT を実行する。入力は通常の順序、出力はビット反転した順序である。
isign = +2 の場合は、逆方向の FFT を実行する。入力はビット反転した順序、出力は通常の順序である。

上記の式は、表 3-4 に示したすべてのデータ・タイプを使用するすべての FFT に適用される。

cfft2d/zfft2d

Fortran インターフェイス・ルーチン。
複素マトリックスの順方向または逆方向の
FFT を計算する (インプレース)。

```
call cfft2d ( r, m, n, isign )
call zfft2d ( r, m, n, isign )
```

説明

cfft2d/zfft2d ルーチンが実行する演算は、*isign* の値によって決まる。上記の「複素数から複素数への 2 次元 FFT」の演算式を参照。

入力パラメータ

r	COMPLEX (cfft2d の場合) DOUBLE COMPLEX (zfft2d の場合) 配列 DIMENSION は少なくとも (*m*, *n*)。リーディング・ディメンジョンは *m* に等しくなる。この配列には、変換される複素マトリックスを格納する。
m	INTEGER。列変換の長さ (行数)。*m* は、2 の累乗でなければならない。
n	INTEGER。行変換の長さ (列数)。*n* は、2 の累乗でなければならない。
isign	INTEGER。実行する演算のタイプを指示するフラグ。 *isign* = -1 の場合は、順方向の FFT が実行される。入力と出力は通常の順序である。 *isign* = +1 の場合は、逆方向の FFT が実行される。入力と出力は通常の順序である。 *isign* = -2 の場合は、順方向の FFT を実行する。入力は通常の順序、出力はビット反転した順序である。 *isign* = +2 の場合は、逆方向の FFT を実行する。入力はビット反転した順序、出力は通常の順序である。

出力パラメータ

r	*isign* に従って、変換の結果得られた複素数データが格納される。

cfft2dc/zfft2dc

C インターフェイス・ルーチン。複素マトリックスの順方向または逆方向の FFT を計算する (インプレース)。

```
void cfft2dc ( float* r, float* i, int m, int n, int isign )
void zfft1dc ( double* r, double* i, int m, int n, int isign )
```

説明

cfft2dc/zfft2dc ルーチンが実行する演算は、*isign* の値によって決まる。上記の「複素数から複素数への 2 次元 FFT」の演算式を参照。

入力パラメータ

r	float* (cfft2dc の場合) double* (zfft2dc の場合) 少なくとも (*m*, *n*) のサイズを持つ 2 次元配列へのポインタ。リーディング・ディメンジョンは *n* に等しくなる。この配列には、変換される複素マトリックスの実数部を格納する。
i	float* (cfft2dc の場合) double* (zfft2dc の場合) 少なくとも (*m*, *n*) のサイズを持つ 2 次元配列へのポインタ。リーディング・ディメンジョンは *n* に等しくなる。この配列には、変換される複素マトリックスの虚数部を格納する。
m	int。列変換の長さ (行数)。*m* は、2 の累乗でなければならない。

n	int。行変換の長さ (列数)。*n* は、2 の累乗でなければならない。
isign	int。実行する演算のタイプを指示するフラグ。 *isign* = -1 の場合は、順方向の FFT が実行される。入力と出力は通常の順序である。 *isign* = +1 の場合は、逆方向の FFT が実行される。入力と出力は通常の順序である。 *isign* = -2 の場合は、順方向の FFT を実行する。入力は通常の順序、出力はビット反転した順序である。 *isign* = +2 の場合は、逆方向の FFT を実行する。入力はビット反転した順序、出力は通常の順序である。

出力パラメータ

r	*isign* に従って、変換後の実数部が格納される。
i	*isign* に従って、変換後の虚数部が格納される。

実数から複素数への 2 次元 FFT

実数から複素数へのルーチンは、いずれも次の式に従って、実数マトリックスの順方向の FFT を計算する。

$$z_{i,j} = \sum_{k=0}^{m-1} \sum_{l=0}^{n-1} t_{k,l} * w_m^{-i*k} * w_n^{-j*l}, \quad 0 \le i \le m-1, \quad 0 \le j \le n-1$$

$t_{k,l}$ = cmplx($r_{k,l}$, 0)、ここで $r_{k,l}$ は実数入力マトリックスで、$0 \le k \le m-1$, $0 \le l \le n-1$ である。数値演算の結果 $z_{i,j}$、$0 \le i \le m-1$, $0 \le j \le n-1$ は、サイズ (*m*, *n*) の複素マトリックスである。各列は、次のような複素共役対称ベクトルになる。

$0 \le j \le n-1$ の場合は、

z(*m*/2+*i*, *j*) = conjg(*z*(*m*/2-*i*, *j*))、1 ≤ *i* ≤ *m*/2-1、また *j*=0 と *j*=*n*/2 の場合は、*z*(0, *j*) と *z*(*m*/2, *j*) は実数値になる。

この数値演算の結果は、サイズ (*m*/2+1, *n*/2+1) の複素数の 2 次元配列またはサイズ (*m*+2, *n*+2) の実数の 2 次元配列に格納される。CCS 形式のデータ格納は、Fortran インターフェイス向けルーチンと C インターフェイス向けルーチンについて別々に後で定義する。

scfft2d/dzfft2d

Fortran インターフェイス・ルーチン。
実数マトリックスの順方向の FFT を
計算し、結果の複素共役対称
マトリックスを CCS 形式で表現する
(インプレース)。

```
call scfft2d ( r, m, n )
call dzfft2d ( r, m, n )
```

説明

上記の「実数から複素数への 2 次元 FFT」の演算式を参照。

これらのルーチンは、複素数から実数への変換ルーチンである csfft2d/zdfft2d を補足する。

入力パラメータ

r	REAL (scfft2d の場合) DOUBLE PRECISION (dzfft2d の場合) 配列 DIMENSION は少なくとも (*m*+2, *n*+2)。リーディング・ディメンジョンは (*m*+2) に等しくなる。この配列の最初の *m* 行 *n* 列には、変換される実マトリックスを格納する。表 3-5 に、入力データのレイアウトを示す。
m	INTEGER。列変換の長さ (行数)。*m* は、2 の累乗でなければならない。
n	INTEGER。行変換の長さ (列数)。*n* は、2 の累乗でなければならない。

表 3-5　実数から複素数への 2 次元 FFT と複素数から実数への 2 次元 FFT の実数データ格納形式 (Fortran インターフェイス)

r(1, 1)	r(1, 2)	...	r(1, n-1)	r(1, n)	n/u	n/u
r(2, 1)	r(2, 2)	...	r(2, n-1)	r(2, n)	n/u	n/u
r(3, 1)	r(3, 2)	...	r(3, n-1)	r(3, n)	n/u	n/u
r(4, 1)	r(4, 2)	...	r(4, n-1)	r(4, n)	n/u	n/u
...	...	...	...	...	...	...
r(m-1, 1)	r(m-1, 2)	...	r(m-1, n-1)	r(m-1, n)	n/u	n/u
r(m, 1)	r(m, 2)	...	r(m, n-1)	r(m, n)	n/u	n/u
n/u	n/u	...	n/u	n/u	n/u	n/u
n/u	n/u	...	n/u	n/u	n/u	n/u

* n/u - 使用しない

出力パラメータ

r　出力される実数の配列 r(1:m+2, 1:n+2) には、Fortran インターフェイス向け CCS 形式にパックされた複素共役対称マトリックス z(1:m, 1:n) を、次のように格納する。

- 行 1 と m+1 には、n+2 の位置に、CCS 形式にパックされた複素共役対称ベクトル z(1, j) と z(m/2+1, j) を格納する (上記の「実数から複素数への 1 次元 FFT」を参照)。
  完全複素ベクトル z(1, j) は、次のように定義される。
  z(1, j) = cmplx(r(1, 2*j-1), r(1, 2*j)), 1 ≤ j ≤ n/2+1,
  z(1, n/2+1+j) = conjg(z(1, n/2+1-j)), 1 ≤ j ≤ n/2-1
  完全複素ベクトル z(m/2+1, j) は、次のように定義される。
  z(m/2+1, j) = cmplx(r(m+1, 2*j-1), r(m+1, 2*j)), 1 ≤ j ≤ n/2+1,
  z(m/2+1, n/2+1+j) = conjg(z(m/2+1, n/2+1-j)), 1 ≤ j ≤ n/2-1;
- 行 3 ～ m には、n の位置に、次のように表現される複素ベクトルを格納する。
  z(i+1, j) = cmplx(r(2*i+1, j),r(2*i+2, j)), 1 ≤ i ≤ m/2-1, 1 ≤ j ≤ n

- その他のマトリックス成分は、次のように求められる。
  z(m/2+1+i, j) = conjg(z(m/2+1-i, j)),
  1 ≤ i ≤ m/2-1, 1 ≤ j ≤ n

Fortran インターフェイスの場合の複素共役対称マトリックス z の格納形式を、表 3-6 に示す。

表 3-6 実数から複素数への 2 次元 FFT と複素数から実数への 2 次元 FFT の CCS 形式のデータ格納形式 (Fortran インターフェイス)

z(1,1)	0	REz(1,2)	IMz(1,2)	...	REz(1, n/2)	IMz(1, n/2)	z(1, n/2+1)	0
0	0	0	0	...	0	0	0	0
REz(2,1)	REz(2,2)	REz(2,3)	REz(2,4)	...	REz(2, n-1)	REz(2, n)	n/u	n/u
IMz(2,1)	IMz(2,2)	IMz(2,3)	IMz(2,4)	...	IMz(2, n-1)	IMz(2, n)	n/u	n/u
...	...	...	...	...	...	...	n/u	n/u
REz(m/2,1)	REz(m/2,2)	REz(m/2,3)	REz(m/2,4)	...	REz(m/2, n-1)	REz(m/2, n)	n/u	n/u
IMz(m/2,1)	IMz(m/2,2)	IMz(m/2,3)	IMz(m/2,4)	...	IMz(m/2, n-1)	IMz(m/2, n)	n/u	n/u
z(m/2+1,1)	0	REz(m/2+1,2)	IMz(m/2+1,2)	...	REz(m/2+1, n/2)	IMz(m/2+1, n/2)	z(m/2+1, n/2+1)	0
0	0	0	0	...	0	0	n/u	n/u

* n/u - 使用しない

scfft2dc/dzfft2dc

C インターフェイス・ルーチン。実数マトリックスの順方向の FFT を計算し、結果の複素共役対称マトリックスを CCS 形式で表現する (インプレース)。

```
void scfft2dc ( float* r, int m, int n )
void dzfft2dc ( double* r, int m, int n )
```

説明

上記の「実数から複素数への 2 次元 FFT」の演算式を参照。

これらのルーチンは、複素数から実数への変換ルーチンである csfft2dc/zdfft2dc を補足する。

入力パラメータ

r float* (scfft2dc の場合)
double* (dzfft2dc の場合)
少なくとも (*m*+2, *n*+2) のサイズを持つ配列へのポインタ。リーディング・ディメンジョンは (*n*+2) に等しくなる。この配列の最初の *m* 行 *n* 列には、変換される実マトリックスを格納する。
表 3-7 に、入力データのレイアウトを示す。

m int。列変換の長さ。*m* は、2 の累乗でなければならない。

n int。行変換の長さ。*n* は、2 の累乗でなければならない。

表 3-7 実数から複素数への 2 次元 FFT と複素数から実数への 2 次元 FFT の実数データ格納形式 (C インターフェイス)

r(0,0)	r(0,1)	...	r(0, n-2)	r(0, n-1)	n/u	n/u
r(1,0)	r(1,1)	...	r(1, n-2)	r(1, n-1)	n/u	n/u
r(2,0)	r(2,1)	...	r(2, n-2)	r(2, n-1)	n/u	n/u
r(3,0)	r(3,1)	...	r(3, n-2)	r(3, n-1)	n/u	n/u
...	...	...	...	...	...	...
r(m-2,0)	r(m-2,1)	...	r(m-2, n-2)	r(m-2, n-1)	n/u	n/u
r(m-1,0)	r(m-1,1)	...	r(m-1, n-2)	r(m-1, n-1)	n/u	n/u
n/u	n/u	...	n/u	n/u	n/u	n/u
n/u	n/u	...	n/u	n/u	n/u	n/u

出力パラメータ

r 出力される実数の配列 *r*(0:*m*+1, 0:*n*+1) には、C インターフェイス向け CCS 形式にパックされた複素共役対称マトリックス *z*(0:*m*-1, 0:*n*-1) を、次のように格納する。

- 列 0 と *n*/2 には、*m*+2 の位置に、複素共役対称ベクトル *z*(i, 0) と *z*(i, *n*/2) を CCS 形式で格納する (上記の「実数から複素数への 1 次元 FFT」を参照)。

  完全複素ベクトル *z*(i, 0) は、次のように定義される。

  z(i, 0) = cmplx(*r*(i, 0), *r*(*m*/2+i+1, 0)),
  0 ≤ i ≤ *m*/2, *z*(*m*/2+i, 0) = conj(*z*(*m*/2-i, 0)),
  1 ≤ i ≤ *m*/2-1

  完全複素ベクトル *z*(i, *n*/2) は、次のように定義される。

  z(i, *n*/2) = cmplx(*r*(i, *n*/2), *r*(*m*/2+i+1, *n*/2)),
  0 ≤ i ≤ *m*/2,
  z(*m*/2+i, *n*/2) = conj(*z*(*m*/2-i, *n*/2)),
  1 ≤ i ≤ *m*/2-1

- 列 1 ～ *n*/2-1 には複素ベクトルの実数部を格納し、列 *n*/2+2 ～ *n* には虚数部を格納する。各ベクトルのこれらの値は *m* の位置に格納され、次のように表現される。

  z(i, j) = cmplx(*r*(i, j), *r*(i, *n*/2+1+j)),
  0 ≤ i ≤ *m*-1, 1 ≤ j ≤ *n*/2-1

- その他のマトリックス成分は、次のように求められる。

  z(i, *n*/2+j) = conj(*z*(i, *n*/2-j)),
  0 ≤ i ≤ *m*-1, 1 ≤ j ≤ *n*/2-1

C インターフェイスの場合の複素共役対称マトリックス *z* の格納形式を、表 3-8 に示す。

表 3-8　実数から複素数への 2 次元 FFT と複素数から実数への 2 次元 FFT の CCS 形式のデータ格納形式 (C インターフェイス)

z(0, 0)	REz(0, 1)	...	REz(0, n/2-1)	z(0, n/2)	0	IMz(0, 1)	...	IMz(0, n/2-1)	0
REz(1, 0)	REz(1, 1)	...	REz(1, n/2-1)	REz(1, n/2)	0	IMz(1, 1)	...	IMz(1, n/2-1)	0
...	...	...	...	...	0	...	...	...	0
REz(m/2-1, 0)	REz(m/2-1, 1)	...	REz(m/2-1, n/2-1)	REz(m/2-1, n/2)	0	IMz(m/2-1, 1)	...	IMz(m/2-1, n/2-1)	0
z(m/2, 0)	REz(m/2, 1)	...	REz(m/2, n/2-1)	z(m/2, n/2)	0	IMz(m/2, 1)	...	IMz(m/2, n/2-1)	0
0	REz(m/2+1, 1)	...	REz(m/2+1, n/2-1)	0	0	IMz(m/2+1, 1)	...	IMz(m/2+1, n/2-1)	0
IMz(1, 0)	REz(m/2+2, 1)	...	REz(m/2+2, n/2-1)	IMz(1, n/2)	0	IMz(m/2+2, 1)	...	IMz(m/2+2, n/2-1)	0
...	...	...	...	...	0	...	...	...	0
IMz(m/2-2, 0)	REz(m-1, 1)	...	REz(m-1, n/2-1)	IMz(m/2-2, n/2)	0	IMz(m-1, 1)	...	IMz(m-1, n/2-1)	0
IMz(m/2-1, 0)	n/u	...	n/u	IMz(m/2-1, n/2)	n/u	n/u	...	n/u	n/u
0	n/u	...	n/u	0	n/u	n/u	...	n/u	n/u

複素数から実数への 2 次元 FFT

複素数から実数への各ルーチンは、算術式に対して 2 次元の逆方向 FFT を計算する。

$$t_{i,j} = \frac{1}{m*n}\sum_{k=0}^{m-1}\sum_{l=0}^{n-1} z_{k,l}*w_m^{i*k}*w_n^{j*l}, \quad 0 \le i \le m-1, \quad 0 \le j \le n-1$$

数値演算の入力 $z_{i,j}$、 $0 \le i \le m-1$, $0 \le j \le n-1$ は、サイズ (m, n) の複素マトリックスである。各列は、次のような複素共役対称ベクトルになる。

$0 \le j \le n-1$ の場合は、
$z(m/2+i,\ j) = \mathrm{conjg}(z(m/2-i,\ j))$、$1 \le i \le m/2-1$ $(z(m/2-i,\ j))$、$1 \le i \le m/2-1$、また $j=0$ と $j=n/2$ の場合は、$z(0,\ j)$ と $z(m/2,\ j)$ は実数値になる。

この数値演算の入力は、サイズ $(m/2+1,\ n/2+1)$ の複素数 2 次元配列またはサイズ $(m+2,\ n+2)$ の実数 2 次元配列に格納される。CCS 形式のデータ格納の詳細は、上記の「実数から複素数への 1 次元 FFT」を参照。

この変換の数値演算の結果は、$t_{k,l} = \mathrm{cmplx}(r_{k,l},\ 0)$ になる。$r_{k,l}$ は実数マトリックスで $0 \le k \le m-1$, $0 \le l \le n-1$ である。

csfft2d/zdfft2d

Fortran インターフェイス・ルーチン。
CCS 形式にパックされた複素共役対称マトリックスの逆 FFT を計算する (インプレース)。

```
call csfft2d ( r, m, n )
call zdfft2d ( r, m, n )
```

説明

上記の「複素数から実数への 2 次元 FFT」の演算式を参照。これらのルーチンは、実数から複素数への変換ルーチンである scfft2d/dzfft2d を補足する。

入力パラメータ

r

SINGLE PRECISION REAL*4 (csfft2d の場合)
DOUBLE PRECISION REAL*8 (zdfft2d の場合)

DIMENSION が少なくとも $(m+2,\ n+2)$ で、リーディング・ディメンジョンが $(m+2)$ に等しい配列。この配列には、変換される複素共役対称マトリックスを CCS 形式で格納する。表 3-6 に、入力データのレイアウトを示す。

m INTEGER。列変換の長さ (行数)。*m* は、2 の累乗でなければならない。

n INTEGER。行変換の長さ (列数)。*n* は、2 の累乗でなければならない。

出力パラメータ

r 変換の結果の実数部が格納される。出力データのレイアウトは、表 3-5 を参照。

csfft2dc/zdfft2dc

C インターフェイス・ルーチン。CCS 形式にパックされた複素共役対称マトリックスの逆 FFT を計算する (インプレース)。

```
void csfft2dc ( float* r, int m, int n );
void zdfft2dc ( double* r, int m, int n );
```

説明

上記の「複素数から実数への 2 次元 FFT」の演算式を参照。これらのルーチンは、実数から複素数への変換ルーチンである scfft2dc/dzfft2dc を補足する。

入力パラメータ

r float* (csfft2dc の場合)
double* (zdfft2dc の場合)
少なくとも (*m*+2, *n*+2) のサイズを持つ配列へのポインタ。リーディング・ディメンジョンは (*n*+2) に等しくなる。これらの配列には、変換される複素共役対称マトリックスを CCS 形式で格納する。表 3-8 に、入力データのレイアウトを示す。

m int。列変換の長さ。*m* は、2 の累乗でなければならない。

n int。行変換の長さ。*n* は、2 の累乗でなければならない。

出力パラメータ

r 変換の結果の実数データが格納される。出力データのレイアウトは、scfft2dc/dzfft2dc の入力データと同じである。詳細は、表 3-7 を参照。

LAPACK ルーチン：1 次方程式

4

本章では、連立 1 次方程式の解の算出と、それに関連する一連の計算タスクを実行するための、LAPACK パッケージからインテル ® マス・カーネル・ライブラリに組み込まれているルーチンについて説明する。このライブラリには、実データと複素データ用の LAPACK ルーチンが含まれる。

このライブラリは、以下のタイプのマトリックスを持つ連立方程式用のルーチンをサポートしている。

- 一般マトリックス
- バンド・マトリックス
- 正値の対称またはエルミート・マトリックス (フル格納およびパックド格納形式)
- 正値の対称またはエルミートバンド・マトリックス
- 不定値の対称またはエルミートマトリックス (フル格納およびパックド格納形式)
- 不定値の対称またはエルミートバンド・マトリックス
- 三角マトリックス (フル格納およびパックド格納形式)
- 三角バンド・マトリックス
- 三角対角マトリックス

上記のマトリックス・タイプのそれぞれについて、ライブラリにはマトリックスの**因子分解** (三角マトリックスを除く)、マトリックスの**平衡化**、連立 1 次方程式の解の**算出**、マトリックスの**条件数の推定**、1 次方程式の解の精度の**改善**と誤差範囲の計算、**逆**マトリックスの計算を実行するためのルーチンを用意している。

特定の問題を解くために、2 つ以上の計算ルーチンを呼び出すか、1 回の呼び出しで複数のタスクを組み合わせるドライバ・ルーチン (例えば、因子分解と解の算出のための `?gesv`) を呼び出せる。したがって、一般マトリックスの連立 1 次方程式を解く場合には、最初に `?getrf` (*LU* 因子分解) を呼び出し、次に `?getrs` (解の算出) を呼び出せる。さらに、`?gerfs` を呼び出して、解の精度を改善し、誤差範囲を計算できる。これらの代わりにドライバ・ルーチン `?gesvx` を使用することもできる。このルーチンでは、これらのタスクを 1 回の呼び出して実行する。

ルーチンや規則

本章の各ルーチンについて、LAPACK 名を使用できる。

表 4-1 と表 4-2 に、LAPACK 名の一覧を示す。LAPACK 名は、以下に説明するように、xyyzzz または xyyzz の構造になっている。

最初の文字 x は、次のデータ・タイプを示す。

s	実数、単精度	c	複素数、単精度
d	実数、倍精度	z	複素数、倍精度

2 番目と 3 番目の文字 yy は、マトリックスのタイプと格納形式を示す。

ge	一般
gb	一般バンド
gt	一般三重対角マトリックス
po	正値対称またはエルミート
pp	正値対称またはエルミート (パックド格納形式)
pb	正値対称またはエルミート
pt	正値対称またはエルミート三重対角
sy	不定値対称
sp	不定値対称 (パックド格納形式)
he	不定値エルミート
hp	不定値エルミート (パックド格納形式)
tr	三角
tp	三角 (パックド格納形式)
tb	三角バンド

計算ルーチンでは、最後の 3 つの文字 zzz は、実行される処理を示す。

trf	三角マトリックスの因子分解を実行する。
trs	因子分解されたマトリックスを使って連立 1 次方程式を解く。
con	マトリックスの条件数を推定する。
rfs	解の精度を改善し、誤差範囲を計算する。
tri	因子分解を使用して逆マトリックスを計算する。
equ	マトリックスを平衡化する。

例えば、ルーチン sgetrf は、単精度実数型の一般マトリックスの三角因子分解を実行する。これに対応する複素マトリックス用のルーチンは、cgetrf である。

ドライバ・ルーチンには、名前の最後に -sv (**簡易**ドライバ) または -svx (**高度**ドライバ) が付く。

マトリックスの格納形式

LAPACK ルーチンでは、以下のマトリックス格納形式を使用している。

- **フル格納**では、マトリックス A は 2 次元配列 `a` に格納され、この場合、マトリックス成分 a_{ij} は配列成分 `a(i, j)` に格納される。
- **パックド格納**では、対称マトリックス、エルミート・マトリックス、または三角マトリックスをコンパクトに格納できる。マトリックスの上三角または下三角が 1 次元配列の各列にパックされる。
- **バンド格納**では、`kl` 個の劣対角成分と `ku` 個の優対角成分を持つ `m` × `n` のバンド・マトリックスは、(`kl`+`ku`+1) 行 `n` 列の 2 次元配列 `ab` にコンパクトに格納される。マトリックスの各列は配列の対応する列に格納され、マトリックスの**各対角成分**は配列の各行に格納される。

第 4 章と第 5 章では、パックド形式でマトリックスを格納する配列は名前の最後の文字を `p` で、バンド形式でマトリックスを格納する配列には名前の最後の文字を `b` で示している。

マトリックスの格納形式の詳細は、付録 A の「マトリックス引数」を参照。

数学的表記

LAPACK ルーチンの説明では、以下の表記を使用している。

$Ax = b$	$n \times n$ のマトリックス $A = \{a_{ij}\}$、右辺のベクトル $b = \{b_i\}$、未知のベクトル $x = \{x_i\}$ を持つ連立 1 次方程式。
$AX = B$	共通のマトリックス A と複数の右辺を持つ連立方程式の集合。B の各列は右辺のそれぞれに相当する。X の各列は対応する解である。
$\|x\|$	成分 $\|x_i\|$ (x_i の絶対値) を持つベクトル。
$\|A\|$	成分 $\|a_{ij}\|$ (a_{ij} の絶対値) を持つマトリックス。
$\|\|x\|\|_\infty = \max_i \|x_i\|$	ベクトル x の**無限ノルム**。
$\|\|A\|\|_\infty = \max_i \sum_j \|a_{ij}\|$	マトリックス A の**無限ノルム**。
$\|\|A\|\|_1 = \max_j \sum_i \|a_{ij}\|$	マトリックス A の **1- ノルム**。$\|\|A\|\|_1 = \|\|A^T\|\|_\infty = \|\|A^H\|\|_\infty$
$\kappa(A) = \|\|A\|\| \, \|\|A^{-1}\|\|$	マトリックス A の**条件数**

誤差の分析

実際には、ほとんどの計算で実行時に丸め誤差が発生する。また、連立方程式 $Ax = b$ で、データ (A と b の成分) が正確にわかっていない場合もある。したがって、データの誤差と丸め誤差が解 x に与える影響を理解する必要がある。

データの摂動。 x が $Ax = b$ の正確な解であり、$x + \delta x$ が摂動問題 $(A + \delta A)x = (b + \delta b)$ の正確な解である場合は、次の式が成り立つ。

$$\frac{\|\delta x\|}{\|x\|} \le \kappa(A)\left(\frac{\|\delta A\|}{\|A\|} + \frac{\|\delta b\|}{\|b\|}\right), \text{ 係数 } \kappa(A) = \|A\|\,\|A^{-1}\|$$

つまり、A または b の相対誤差は、解のベクトル x 内で係数 $\kappa(A) = \|A\|\,\|A^{-1}\|$ によって増幅される場合がある。この係数を、A の**条件数**と呼ぶ。

丸め誤差は、元のデータの相対摂動 $c(n)\varepsilon$ と同じ効果を持つ。ε は**マシンの精度**、$c(n)$ はマトリックスの次数 n の適度な関数である。対応する解の誤差は、$\|\delta x\|/\|x\| \le c(n)\kappa(A)\varepsilon$ になる ($c(n)$ の値が $10n$ より大きくなることはほとんどない)。

したがって、マトリックス A が**悪条件**である場合 (つまり、条件数 $\kappa(A)$ が非常に大きい場合) は、解 x の誤差も大きくなる。ときには、精度が全く失われてしまう場合もある。LAPACK には、$\kappa(A)$ を推定するためのルーチンが含まれている (「計算ルーチン」を参照)。また、実際の解の誤差をより正確に推定する方法もある (「計算ルーチン」を参照)。

計算ルーチン

表 4-1 に、**実数**マトリックスの因子分解、平衡化、逆マトリックスの計算、実数マトリックスの条件数の推定、実数マトリックスを持つ連立方程式の解の算出、解の精度の改善、解の誤差の推定を行うための LAPACK 計算ルーチンを示す。

表 4-2 に、これに対応する**複素**マトリックス用のルーチンを示す。

表 4-1　実数マトリックスを持つ連立方程式用の計算ルーチン

マトリックスのタイプ、格納形式	マトリックスの因子分解	マトリックスの平衡化	連立方程式の解の算出	条件数	誤差の推定	逆マトリックスの計算
一般	?getrf	?geequ	?getrs	?gecon	?gerfs	?getri
一般バンド	?gbtrf	?gbequ	?gbtrs	?gbcon	?gbrfs	
一般三重対角	?gttrf		?gttrs	?gtcon	?gtrfs	
正値対称	?potrf	?poequ	?potrs	?pocon	?porfs	?potri
正値対称パックド格納	?pptrf	?ppequ	?pptrs	?ppcon	?pprfs	?pptri
正値対称バンド	?pbtrf	?pbequ	?pbtrs	?pbcon	?pbrfs	
正値対称三重対角	?pttrf		?pttrs	?ptcon	?ptrfs	
対称不定値	?sytrf		?sytrs	?sycon	?syrfs	?sytri
対称不定値パックド格納	?sptrf		?sptrs	?spcon	?sprfs	?sptri
三角			?trtrs	?trcon	?trrfs	?trtri
三角パックド格納			?tptrs	?tpcon	?tprfs	?tptri
三角バンド			?tbtrs	?tbcon	?tbrfs	

表中の **?** は、**s** (単精度) または **d** (倍精度) を示す。

表 4-2　複素マトリックスを持つ連立方程式用の計算ルーチン

マトリックスのタイプ、格納形式	マトリックスの因子分解	マトリックスの平衡化	連立方程式の解の算出	条件数	誤差の推定	逆マトリックスの計算
一般	?getrf	?geequ	?getrs	?gecon	?gerfs	?getri
一般バンド	?gbtrf	?gbequ	?gbtrs	?gbcon	?gbrfs	
一般三重対角	?gttrf		?gttrs	?gtcon	?gtrfs	
正値エルミート	?potrf	?poequ	?potrs	?pocon	?porfs	?potri
正値エルミートパックド格納	?pptrf	?ppequ	?pptrs	?ppcon	?pprfs	?pptri
正値エルミートバンド	?pbtrf	?pbequ	?pbtrs	?pbcon	?pbrfs	
正値エルミート三重対角	?pttrf		?pttrs	?ptcon	?ptrfs	
エルミート不定値	?hetrf		?hetrs	?hecon	?herfs	?hetri
対称不定値	?sytrf		?sytrs	?sycon	?syrfs	?sytri
エルミート不定値パック格納	?hptrf		?hptrs	?hpcon	?hprfs	?hptri
対称不定値パックド格納	?sptrf		?sptrs	?spcon	?sprfs	?sptri
三角			?trtrs	?trcon	?trrfs	?trtri
三角パックド格納			?tptrs	?tpcon	?tprfs	?tptri
三角バンド			?tbtrs	?tbcon	?tbrfs	

表中の **?** は、**c** (単精度複素数) または **z** (倍精度複素数) を示す。

マトリックスの因子分解用のルーチン

この節では、マトリックスの因子分解用の LAPACK ルーチンについて説明する。以下の因子分解をサポートしている。

- *LU* 因子分解
- 正値実対称マトリックスのコルスキー因子分解
- 正値エルミート・マトリックスのコルスキー因子分解
- 実対称マトリックスと複素対称マトリックスの Bunch-Kaufman 因子分解
- エルミート・マトリックスの Bunch-Kaufman 因子分解

LU 因子分解の計算には、フル格納とバンド格納形式のマトリックスを使用できる。コルスキー因子分解では、フル格納、パックド格納、バンド格納形式を使用できる。Bunch-Kaufman 因子分解では、フル格納とパックド格納形式を使用できる。

?getrf

$m \times n$ の一般マトリックスの *LU* 因子分解を行う。

```
call sgetrf ( m, n, a, lda, ipiv, info )
call dgetrf ( m, n, a, lda, ipiv, info )
call cgetrf ( m, n, a, lda, ipiv, info )
call zgetrf ( m, n, a, lda, ipiv, info )
```

説明

このルーチンは、次の式に従って、$m \times n$ の一般マトリックス A の *LU* 因子分解を実行する。

$$A = PLU$$

P は置換マトリックス、L は単位対角成分を含む下三角 ($m > n$ の場合は下台形)、U は上三角 ($m < n$ の場合は上台形) である。通常は、A は正方マトリックス ($m = n$) で、L と U はいずれも三角マトリックスである。このルーチンでは、行を交換し、部分的にピボット (要) 演算を行う。

入力パラメータ

m	INTEGER。マトリックス *A* の行の数 ($m \geq 0$)。
n	INTEGER。*A* の列の数 ($n \geq 0$)。
a	REAL (sgetrf の場合) DOUBLE PRECISION (dgetrf の場合) COMPLEX (cgetrf の場合) DOUBLE COMPLEX (zgetrf の場合) 配列 DIMENSION (*lda*, *)。マトリックス *A* を格納する。 *a* の第 2 次元は、少なくとも max(1, *n*) でなければならない。
lda	INTEGER。*a* の第 1 次元。

出力パラメータ

a	*L* と *U* によって上書きされる。*L* の単位対角成分は格納されない。
ipiv	INTEGER 配列 DIMENSION は少なくとも max(1, min(*m*, *n*)) ピボット (要) のインデックスの行 *i* は *ipiv(i)* と交換される。
info	INTEGER *info* = 0 の場合は、正常に終了する。 *info* = -*i* の場合は、*i* 番目のパラメータの値が不正である。 *info* = *i* の場合は、u_{ii} は 0 である。因子分解は完了したが、*U* は完全に特異で、ゼロである。連立 1 次方程式の解の算出に係数 *U* を使用すると、0 による除算が発生する。

アプリケーション・ノート

算出された *L* と *U* は、摂動マトリックス *A* + *E* の正確な係数になる。

$$|E| \leq c(\min(m, n))\varepsilon P|L||U|$$

$c(n)$ は、*n* の適度な 1 次関数で、ε はマシンの精度である。

実数型の浮動小数点演算のおおよその回数は次のとおりである。

$(2/3)n^3$	($m = n$ の場合)
$(1/3)n^2(3m-n)$	($m > n$ の場合)

$(1/3)m^2(3n-m)$　　($m < n$ の場合)

複素数型の演算回数は、この 4 倍になる。

$m = n$ でこのルーチンを呼び出した後、以下のルーチンを呼び出せる。

?getrs	$AX = B$ または $A^TX = B$ または $A^HX = B$ を解く。
?gecon	A の条件数を推定する。
?getri	A の逆マトリックスを計算する。

?gbtrf

m × *n* の一般バンド・マトリックスの
LU 因子分解を行う。

```
call sgbtrf ( m, n, kl, ku, ab, ldab, ipiv, info )
call dgbtrf ( m, n, kl, ku, ab, ldab, ipiv, info )
call cgbtrf ( m, n, kl, ku, ab, ldab, ipiv, info )
call zgbtrf ( m, n, kl, ku, ab, ldab, ipiv, info )
```

説明

このルーチンは、*kl* 個のゼロ以外の劣対角成分と *ku* 個のゼロ以外の優対角成分を含む *m* × *n* のバンド・マトリックス *A* の *LU* 因子分解を実行する。通常は、*A* は正方マトリックス (*m* = *n*) で、次の式が成り立つ。

$$A = PLU$$

P は置換マトリックスである。*L* は、単位対角成分を含む、各列のゼロ以外の成分の数が *kl* 個以下の下三角マトリックスである。*U* は、*kl* + *ku* 個の優対角成分を含む上三角バンド・マトリックスである。このルーチンは、行を交換して、部分的にピボット (要) 演算を行う (行の交換によって、*U* に *kl* 個の優対角成分が追加される)。

入力パラメータ

m	INTEGER。マトリックス *A* の行の数 (*m* ≥ 0)。
n	INTEGER。*A* の列の数 (*n* ≥ 0)。
kl	INTEGER。*A* のバンドの中の劣対角成分の数 (*kl* ≥ 0)。
ku	INTEGER。*A* のバンドの中の優対角成分の数 (*ku* ≥ 0)。
ab	REAL (sgbtrf の場合) DOUBLE PRECISION (dgbtrf の場合) COMPLEX (cgbtrf の場合) DOUBLE COMPLEX (zgbtrf の場合) 配列、DIMENSION (*ldab*, *)

配列 *ab* には、マトリックス *A* をバンド格納形式で格納する (「マトリックスの格納形式」を参照)。
ab の第 2 次元は、少なくとも max(1, *n*) でなければならない。

ldab INTEGER。配列 *ab* の第 1 次元。
($ldab \geq 2kl + ku + 1$)

出力パラメータ

ab *L* と *U* によって上書きされる。*U* の対角成分と *kl* + *ku* 個の優対角成分は、*ab* の最初の 1 + *kl* + *ku* 行に格納される。*L* の計算に使用される乗数は、次の *kl* 行に格納される。

ipiv INTEGER
配列 DIMENSION は少なくとも max(1, min (*m*, *n*))
ピボット (要) のインデックスの行 *i* は *ipiv(i)* と交換されている。

info INTEGER。*info* = 0 の場合は、正常に終了する。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。
info = *i* の場合は、u_{ii} は 0 である。因子分解は完了したが、*U* は完全に特異で、ゼロである。連立 1 次方程式の解の算出に係数 *U* を使用すると、0 による除算が発生する。

アプリケーション・ノート

算出された *L* と *U* は、摂動マトリックス *A* + *E* の正確な係数になる。

$$|E| \leq c(kl + ku + 1)\varepsilon P|L||U|$$

$c(k)$ は、*k* の適度な 1 次関数で、ε はマシンの精度である。

実数型の浮動小数点演算の合計回数は、約 $2n(ku+1)kl$ から $2n(kl+ku+1)kl$ までの範囲である。複素数型の演算回数は、この 4 倍になる。これらの推定では、*kl* と *ku* は min(*m*, *n*) よりはるかに小さいものとする。

m = *n* にしてこのルーチンを呼び出した後、以下のルーチンを呼び出せる。

?gbtrs $AX = B$ または $A^TX = B$ または $A^HX = B$ を解く。

?gbcon *A* の条件数を推定する。

?gttrf

三重対角マトリックスの *LU* 因子分解を行う。

```
call sgttrf ( N, dl, d, du, du2, Ipiv, info )
call dgttrf ( N, dl, d, du, du2, Ipiv, info )
call cgttrf ( N, dl, d, du, du2, Ipiv, info )
call zgttrf ( N, dl, d, du, du2, Ipiv, info )
```

説明

このルーチンは、次の形式で、実数または複素三重対角マトリックス *A* の *LU* 因子分解を実行する。

$$A = PLU$$

P は置換マトリックス、*L* は単位対角成分を持つ下二重対角マトリックス、*U* は主対角成分と最初の 2 つの優対角成分にのみ 0 でない値を持つ上三角マトリックスである。このルーチンは、部分的なピボット (要) 演算で行を交換し、消去を実行する。

入力パラメータ

n	INTEGER。マトリックス *A* の次数 ($n \geq 0$)。
dl, d, du	REAL (sgttrf の場合) DOUBLE PRECISION (dgttrf の場合) COMPLEX (cgttrf の場合) DOUBLE COMPLEX (zgttrf の場合) A の成分を格納する配列。 次元 (n - 1) の配列 *dl* は、*A* の劣対角成分を格納する。 次元 *n* の配列 *d* は、*A* の対角成分を格納する。 次元 (n - 1) の配列 *du* は、*A* の優対角成分を格納する。

出力パラメータ

dl
A の *LU* 因子分解で得られたマトリックス *L* を定義する、(*n*-1) 個の乗数によって上書きされる。

d
A の *LU* 因子分解で得られた上三角マトリックス *U* の *n* 個の対角成分によって上書きされる。

du
U の最初の優対角成分の (*n*-1) 個の成分によって上書きされる。

du2
REAL (sgttrf の場合)
DOUBLE PRECISION (dgttrf の場合)
COMPLEX (cgttrf の場合)
DOUBLE COMPLEX (zgttrf の場合)
配列、次元は (*n*-2)。終了時に、*du2* には *U* の 2 番目の優対角成分の (*n*-2) 個の成分が格納される。

ipiv
INTEGER
配列、次元は (*n*)
ピボット (要) のインデックス : 行 *i* は行 *ipiv*(*i*) と交換されている。

info
INTEGER。*info* = 0 の場合は、ルーチンの実行は正常に終了した。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。
info = *i* の場合は、u_{ii} が 0 である。因子分解は完了したが、*U* は完全に特異である。連立 1 次方程式の解の算出に係数 *U* を使用すると、0 による除算が発生する。

アプリケーション・ノート

?gbtrs
$AX = B$ または $A^TX = B$ または $A^HX = B$ を解く。

?gbcon
A の条件数を推定する。

?potrf

正値対称 (エルミート) マトリックスの
コルスキー因子分解を行う。

```
call spotrf ( uplo, n, a, lda, info )
call dpotrf ( uplo, n, a, lda, info )
call cpotrf ( uplo, n, a, lda, info )
call zpotrf ( uplo, n, a, lda, info )
```

説明

このルーチンは、正値対称 (複素データの場合は正値エルミート) マトリックス A のコルスキー因子分解を実行する。

$A = U^H U$ (*uplo*='U' の場合)

$A = LL^H$ (*uplo*='L' の場合)

L は下三角マトリックス、U は上三角マトリックスである。

入力パラメータ

uplo CHARACTER*1。'U' または 'L' でなければならない。
A の上三角部分と下三角部分のどちらを格納するか、また A をどのように因子分解するかを指定する。
uplo = 'U' の場合は、マトリックス A の上三角部分を配列 *a* に格納し、A を $U^H U$ として因子分解する。
uplo = 'L の場合は、マトリックス A の下三角部分を配列 *a* に格納し、A を LL^H として因子分解する。

n INTEGER。マトリックス A の次数 ($n \geq 0$)

a REAL (spotrf の場合)
DOUBLE PRECISION (dpotrf の場合)
COMPLEX (cpotrf の場合)
DOUBLE COMPLEX (zpotrf の場合)
配列、DIMENSION (*lda*, *)

配列 *a* には、マトリックス *A* の上三角部分または下三角部分を格納する (*uplo* を参照)。
a の第 2 次元は、少なくとも max(1, *n*) でなければならない。

lda INTEGER。*a* の第 1 次元。

出力パラメータ

a *auplo* の指定に従って、*a* の上三角部分または下三角部分が、コルスキー係数 *U* または *L* によって上書きされる。

info INTEGER。*info* = 0 の場合は、正常に終了する。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。
info = *i* の場合は、次数 *i* の先頭の小行列式 (およびマトリックス *A* そのもの) が正値でないため、因子分解を完了できない。これは、マトリックス *A* の構成に誤りがあるのを示す。

アプリケーション・ノート

uplo = 'U' の場合、算出された係数 *U* は、摂動マトリックス *A* + *E* の正確な係数になる。

$$|E| \le c(n)\varepsilon\,|U^H||U|, \quad |e_{ij}| \le c(n)\varepsilon\sqrt{a_{ii}a_{jj}}$$

c(*n*) は *n* の適度な 1 次関数で、ε はマシンの精度である。

uplo = 'L' の場合も、同様の推定が成り立つ。

浮動小数点演算の合計回数は、実数型の場合は約 (1/3)n^3、複素数型の場合は約 (4/3)n^3 になる。

このルーチンを呼び出した後、以下のルーチンを呼び出せる。

?potrs	*AX* = *B* を解く。
?pocon	*A* の条件数を推定する。
?potri	*A* の逆マトリックスを計算する。

?pptrf

パックド格納形式による正値対称
(エルミート) マトリックスのコルスキー
因子分解を行う。

```
call spptrf ( uplo, n, ap, info )
call dpptrf ( uplo, n, ap, info )
call cpptrf ( uplo, n, ap, info )
call zpptrf ( uplo, n, ap, info )
```

説明

このルーチンは、正値対称 (複素データの場合は正値エルミート) のパックド形式のマトリックス A のコルスキー因子分解を実行する。

$A = U^H U$ (*uplo*='U' の場合)

$A = LL^H$ (*uplo*='L' の場合)

L は下三角マトリックス、U は上三角マトリックスである。

入力パラメータ

uplo CHARACTER*1。'U' または 'L' でなければならない。
A の上三角部分と下三角部分のどちらを配列 *ap* にパックするか、また A をどのように因子分解するかを指定する。
uplo = 'U' の場合は、マトリックス A の上三角部分を配列 *ap* に格納し、A を $U^H U$ として因子分解する。
uplo = 'L' の場合は、マトリックス A の下三角部分を配列 *ap* に格納し、A を LL^H として因子分解する。

n INTEGER。マトリックス A の次数 ($n \geq 0$)。

ap REAL (spptrf の場合)
DOUBLE PRECISION (dpptrf の場合)
COMPLEX (cpptrf の場合)
DOUBLE COMPLEX (zpptrf の場合)
配列 DIMENSION は少なくとも max(1, $n(n+1)/2$)

配列 *ap* には、(*uplo* の指定に従って) マトリックス *A* の上三角部分または下三角部分を**パックド格納形式**で格納する (「マトリックスの格納形式」を参照)。

出力パラメータ

ap *uplo* の指定に従って、パックド格納形式の *A* の上三角部分または下三角部分が、コルスキー係数 *U* または *L* によって上書きされる。

info INTEGER。*info* = 0 の場合は、正常に終了する。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。
info = *i* の場合は、次数 *i* の先頭の小行列式 (およびマトリックス *A* そのもの) が正値でないため、因子分解を完了できない。これは、マトリックス *A* の構成に誤りがあるのを示す。

アプリケーション・ノート

uplo = 'U' の場合、算出された係数 *U* は、摂動マトリックス *A* + *E* の正確な係数になる。

$$|E| \le c(n)\varepsilon \left|U^H\right||U|, \quad |e_{ij}| \le c(n)\varepsilon \sqrt{a_{ii}a_{jj}}$$

$c(n)$ は n の適度な 1 次関数で、ε はマシンの精度である。

uplo = 'L' の場合も、同様の推定が成り立つ。

浮動小数点演算の合計回数は、実数型の場合は約 $(1/3)n^3$, 複素数型の場合は約 $(4/3)n^3$ になる。

このルーチンを呼び出した後、以下のルーチンを呼び出せる。

?pptrs	*AX* = *B* を解く。
?ppcon	*A* の条件数を推定する。
?pptri	*A* の逆マトリックスを計算する。

?pbtrf

正値対称 (エルミート) バンド・
マトリックスのコルスキー因子分解を
行う。

```
call spbtrf ( uplo, N, kd, ab, ldab, info )
call dpbtrf ( uplo, N, kd, ab, ldab, info )
call cpbtrf ( uplo, N, kd, ab, ldab, info )
call zpbtrf ( uplo, N, kd, ab, ldab, info )
```

説明

このルーチンは、正値対称 (複素データの場合は正値エルミート) バンド・マトリックス A のコルスキー因子分解を実行する。

$A = U^H U$ (*uplo*='U' の場合)

$A = LL^H$ (*uplo*='L' の場合)

L は下三角マトリックス、U は上三角マトリックスである。

入力パラメータ

uplo CHARACTER*1。'U' または 'L' でなければならない。
A の上三角部分と下三角部分のどちらを配列 *ab* に格納するか、また A をどのように因子分解するかを指定する。
uplo = 'U' の場合は、マトリックス A の上三角部分を配列 *ab* に格納し、A を $U^H U$ として因子分解する。
uplo = 'L' の場合は、マトリックス A の下三角部分を配列 *ab* に格納し、A を LL^H として因子分解する。

n INTEGER。マトリックス A の次数 ($n \geq 0$)

kd INTEGER。配列 A の優対角成分または劣対角成分の数 ($kd \geq 0$)

ab REAL (*spbtrf* の場合)
DOUBLE PRECISION (*dpbtrf* の場合)
COMPLEX (*cpbtrf* の場合)
DOUBLE COMPLEX (*zpbtrf* の場合)
配列、DIMENSION (*ldab*, *)

配列 *ab* には、(*uplo* の指定に従って) マトリックス *A* の上三角部分または下三角部分を**バンド格納形式**で格納する (「マトリックスの格納形式」を参照)。
ab の第 2 次元は、少なくとも max(1, *n*) でなければならない。

ldab INTEGER。配列 *ab* の第 1 次元 (*ldab* ≥ *kd* +1)

出力パラメータ

ap *uplo* の指定に従って、バンド格納形式の *A* の上三角部分または下三角部分が、コルスキー係数 *U* または *L* によって上書きされる。

info INTEGER。*info* = 0 の場合は、正常に終了する。
info = *-i* の場合は、*i* 番目のパラメータの値が不正である。
info = *i* の場合は、次数 *i* の先頭の小行列式 (およびマトリックス *A* そのもの) が正値マトリックスでないため、因子分解を完了できない。これは、マトリックス *A* の構成に誤りがあるのを示す。

アプリケーション・ノート

uplo = 'U' の場合、算出された係数 U は、摂動マトリックス *A* + *E* の正確な係数になる。

$$|E| \le c(kd+1)\varepsilon\,|U^H||U|, \quad |e_{ij}| \le c(kd+1)\varepsilon\sqrt{a_{ii}a_{jj}}$$

c(*n*) は *n* の適度な 1 次関数で、ε はマシンの精度である。

uplo = 'L' の場合も、同様の推定が成り立つ。

実数型の場合はの浮動小数点演算の合計回数は、約 $n(kd+1)^2$ になる。複素数型の場合はの演算回数は、この 4 倍になる。これらの推定では、*kd* は *n* よりはるかに小さいものとする。

このルーチンを呼び出した後、以下のルーチンを呼び出せる。

?pbtrs	*AX* = *B* を解く。
?pbcon	*A* の条件数を推定する。

?pttrf

正値の対称 (エルミート) 三重対角
マトリックスの因子分解を行う。

```
call spttrf ( n, d, e, info )
call dpttrf ( n, d, e, info )
call cpttrf ( n, d, e, info )
call zpttrf ( n, d, e, info )
```

説明

このルーチンは、正値対称 (複素データの場合は、正値エルミート) 三重対角マトリックス A の因子分解を実行する。

$A = LDL^H$ で、D は対角マトリックス、L は単位下二重対角マトリックスである。この因子分解の形式は、$A = U^HDU$ とも見なせる。D は単位上二重対角マトリックスである。

入力パラメータ

n INTEGER。マトリックス A の次数 ($n \geq 0$)。

d REAL (sptrf, cpttrf の場合)
DOUBLE PRECISION (dpttrf, zpttrf の場合)
配列、次元は (n)。A の対角成分を格納する。

e REAL (sptrf の場合)
DOUBLE PRECISION (dpttrf の場合)
COMPLEX (cpttrf の場合)
DOUBLE COMPLEX (zpttrf の場合)
配列、次元は (n - 1)。A の劣対角成分を格納する。

出力パラメータ

d A の LDL^H 因子分解で得られた対角マトリックス D の n 個の対角成分によって上書きされる。

e	A の因子分解で得られた単位二重対角係数 L または U の (n - 1) 個の対角線外の成分によって上書きされる。
info	INTEGER。*info* = 0 の場合は、ルーチンの実行は正常に終了した。 *info* = -*i* の場合は、*i* 番目のパラメータの値が不正である。 *info* = *i* の場合は、次数 *i* の先頭の小行列式 (したがって、マトリックス A そのもの) が正値になっていない。*i* < *n* の場合は、因子分解を完了できない。*i* = *n* の場合は、因子分解は完了したが、*d* (*n*) = 0 である。

?sytrf

対称マトリックスの Bunch-Kaufman
因子分解を行う。

```
call ssytrf ( uplo, n, a, lda, ipiv, work, lwork, info )
call dsytrf ( uplo, n, a, lda, ipiv, work, lwork, info )
call csytrf ( uplo, n, a, lda, ipiv, work, lwork, info )
call zsytrf ( uplo, n, a, lda, ipiv, work, lwork, info )
```

説明

このルーチンは、対称マトリックスの Bunch-Kaufman 因子分解を実行する。

`uplo='U'` の場合、$A = PUDU^TP^T$

`uplo='L'` の場合、$A = PLDL^TP^T$

A は入力マトリックス、P は置換マトリックス、U と L は単位対角成分を含む上三角マトリックスと下三角マトリックス、D は 1×1 と 2×2 の対角ブロックを持つ対称ブロック対角マトリックスである。U と L は、D の 2×2 のブロックに対応する 2×2 の単位対角ブロックを持っている。

入力パラメータ

uplo
: CHARACTER*1。'U' または 'L' でなければならない。
A の上三角部分と下三角部分のどちらを格納するか、また A をどのように因子分解するかを指定する。
uplo = 'U' の場合は、マトリックス A の上三角部分を配列 *a* に格納し、A を $PUDU^TP^T$ として因子分解する。
uplo = 'L' の場合は、マトリックス A の下三角部分を配列 *a* に格納し、A を $PLDL^TP^T$ として因子分解する。

n
: INTEGER。マトリックス A の次数 ($n \geq 0$)。

a
: REAL (ssytrf の場合)
DOUBLE PRECISION (dsytrf の場合)
COMPLEX (csytrf の場合)
DOUBLE COMPLEX (zsytrf の場合)
配列 DIMENSION (*lda*, *)

配列 *a* には、マトリックス *A* の上三角部分または下三角部分を格納する (*uplo* を参照)。
a の第 2 次元は、少なくとも max(1, *n*) でなければならない。

lda INTEGER。*a* の第 1 次元。少なくとも max(1, *n*) でなければならない。

work *a* と同じタイプ。次元 *lwork* のワークスペース配列。

lwork INTEGER。*work* 配列のサイズ (*lwork* ≥ *n*)。
lwork の推奨値は、「アプリケーション・ノート」を参照。

出力パラメータ

a *a* の上三角部分または下三角部分は、ブロック対角マトリックス *D* の各成分と、係数 *U* (または *L*) の計算に使用された乗数によって上書きされる。

work(1) *info* = 0 の場合は、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の *lwork* の値が *work(1)* に出力される。これ以降の実行には、*lwork* の値を使用する。

ipiv INTEGER
配列 DIMENSION は少なくとも max(1, *n*)
交換の結果と *D* のブロック構造の各成分が格納される。
ipiv(i) = *k* > 0 の場合は、d_{ii} は 1 × 1 のブロックである。*A* の *i* 番目の行と列は、*k* 番目の行と列に交換する。
uplo = 'U' と *ipiv(i)* =*ipiv(i-1)* = -*m* < 0 の場合は、*D* の *i* 行 (*i*-1) 列に 2 × 2 のブロックが作成される。*A* の (*i*-1) 番目の行と列は、*m* 番目の行と列に交換する。
uplo = 'L' と *ipiv(i)* =*ipiv(i+1)* = -*m* < 0 の場合は、*D* の *i* 行 (*i*+1) 列に 2 × 2 のブロックが作成される。*A* の (*i*+1) 番目の行と列は、*m* 番目の行と列に交換する。

info INTEGER。*info* = 0 の場合は、正常に終了する。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。
info = *i* の場合は、d_{ii} は 0 である。因子分解は完了したが、*D* は完全に特異で、ゼロである。連立 1 次方程式の解の算出に *D* を使用すると、0 による除算が発生する。

アプリケーション・ノート

パフォーマンスを最適化するには、*lwork* =*n***blocksize* に設定する。*blocksize* は、**ブロック・アルゴリズム**の最適なパフォーマンスに必要な、マシンによって異なる値 (通常は 16 ～ 64) である。

必要なワークスペースの大きさがわからない場合は、最初の実行では *lwork* を大きめの値に設定する。実行の終了時に *work*(1) を調べて、これ以降の実行にはその値を使用する。

U と L の 2×2 の単位対角ブロックと単位対角成分は格納されない。U と L のそれ以外の成分は、配列 *a* の対応する列に格納される。ただし、U または L を明示的に復元するには、さらに行の交換が必要である (その必要はほとんどない)。

すべての *i* =1 . . . *n* について *ipiv*(i) = *i* が成り立つ場合は、U (L) のすべての対角線外の成分は、配列 *a* の対応する成分に明示的に格納される。

uplo = 'U' の場合、算出された係数 U と D は、摂動マトリックス $A + E$ の正確な係数になる。

$$|E| \leq c(n)\varepsilon P|U||D||U^T|P^T$$

$c(n)$ は n の適度な 1 次関数で、ε はマシンの精度である。
uplo = 'L の場合も、算出された L と D について同様の推定が成り立つ。

浮動小数点演算の合計回数は、実数型の場合は約 $(1/3)n^3$, 複素数型の場合は約 $(4/3)n^3$ になる。

このルーチンを呼び出した後、以下のルーチンを呼び出せる。

?sytrs	$AX = B$ を解く。
?sycon	A の条件数を推定する。
?sytri	A の逆マトリックスを計算する。

?hetrf

複素数型エルミート・マトリックスの
Bunch-Kaufman 因子分解を行う。

```
call chetrf ( uplo, n, a, lda, ipiv, work, lwork, info )
call zhetrf ( uplo, n, a, lda, ipiv, work, lwork, info )
```

説明

このルーチンは、エルミート・マトリックスの Bunch-Kaufman 因子分解を実行する。

`uplo='U'` の場合、$A = PUDU^HP^T$

`uplo='L'` の場合、$A = PLDL^HP^T$

A は入力マトリックス、P は置換マトリックス、U と L は単位対角成分を含む上三角マトリックスと下三角マトリックス、D は 1×1 と 2×2 の対角ブロックを持つエルミート・ブロック対角マトリックスである。U と L は、D の 2×2 のブロックに対応する 2×2 の単位対角ブロックを持っている。

入力パラメータ

uplo `CHARACTER*1`。`'U'` または `'L'` でなければならない。
A の上三角部分と下三角部分のどちらを格納するか、また A をどのように因子分解するかを指定する。
uplo = `'U'` の場合は、マトリックス A の上三角部分を配列 *a* に格納し、A を $PUDU^HP^T$ として因子分解する。
uplo = `'L'` の場合は、マトリックス A の下三角部分を配列 *a* に格納し、A を $PLDL^HP^T$ として因子分解する。

n `INTEGER`。マトリックス A の次数 ($n \geq 0$)。

a `COMPLEX` (`chetrf` の場合)
`DOUBLE COMPLEX` (`zhetrf` の場合)
配列 `DIMENSION` (*lda*, *)
配列 *a* には、マトリックス A の上三角部分または下三角部分を格納する (*uplo* を参照)。
a の第 2 次元は、少なくとも max(1, *n*) でなければならない。

lda INTEGER。*a* の第 1 次元。少なくとも max(1, *n*) でなければならない。

work *a* と同じタイプ。次元 *lwork* のワークスペース配列。

lwork INTEGER。*work* 配列のサイズ (*lwork* ≥ *n*)。
lwork の推奨値は、「アプリケーション・ノート」を参照。

出力パラメータ

a *a* の上三角部分または下三角部分は、ブロック対角マトリックス *D* の各成分と、係数 *U* (または *L*) の計算に使用された乗数によって上書きされる。

work(1) *info* = 0 の場合は、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の *lwork* の値が *work(1)* に出力される。これ以降の実行には、この *lwork* の値を使用する。

ipiv INTEGER
配列 DIMENSION は少なくとも max(1, *n*)
交換の結果と *D* のブロック構造の各成分が格納される。
ipiv(i) = *k* > 0 の場合は、d_{ii} は 1 × 1 のブロックである。*A* の *i* 番目の行と列は、*k* 番目の行と列に交換する。
uplo = 'U' と *ipiv(i)* =*ipiv(i-1)* = -*m* < 0 の場合は、*D* の *i* 行 (*i*-1) 列に 2 × 2 のブロックが作成される。*A* の (*i*-1) 番目の行と列は、*m* 番目の行と列に交換する。
uplo = 'L' と *ipiv(i)* =*ipiv(i+1)* = -*m* < 0 の場合は、*D* の *i* 行 (*i*+1) 列に 2 × 2 のブロックが作成される。*A* の (*i*+1) 番目の行と列は、*m* 番目の行と列に交換する。

info INTEGER。*info* = 0 の場合は、正常に終了する。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。
info = *i* の場合は、d_{ii} は 0 である。因子分解は完了したが、*D* は完全に特異で、ゼロである。連立 1 次方程式の解の算出に *D* を使用すると、0 による除算が発生する。

アプリケーション・ノート

このルーチンは、正値マトリックスかどうかわからないエルミート・マトリックスに最適である。*A* が実際に正値マトリックスの場合は、このルーチンは交換を実行せず、*D* 内に 2 × 2 の対角ブロックは作成されない。

パフォーマンスを最適化するには、`lwork` =`n`**blocksize* に設定する。*blocksize* は、**ブロック・アルゴリズム**の最適なパフォーマンスに必要な、マシンによって異なる値 (通常は 16 ～ 64) である。

必要なワークスペースの大きさがわからない場合は、最初の実行では `lwork` を大きめの値に設定する。実行の終了時に `work(1)` を調べて、これ以降の実行にはその値を使用する。

U と *L* の 2 × 2 の単位対角ブロックと単位対角成分は格納されない。*U* と *L* のそれ以外の成分は、配列 `a` の対応する列に格納される。ただし、*U* または *L* を明示的に復元するには、さらに行の交換が必要である (その必要はほとんどない)。

すべての `i` =1 . . . `n` で `ipiv(i)` = `i` が成り立つ場合は、*U* (*L*) のすべての対角線外の成分は、配列 `a` の対応する成分に明示的に格納される。

`uplo` = `'U'` の場合、算出された係数 *U* と *D* は、摂動マトリックス *A* + *E* の正確な係数になる。

$$|E| \le c(n)\varepsilon P|U||D||U^T|P^T$$

c(`n`) は `n` の適度な 1 次関数で、ε はマシンの精度である。

`uplo` = `'L'` の場合も、算出された *L* と *D* について同様の推定が成り立つ。

浮動小数点演算の合計回数は、約 $(4/3)n^3$ になる。

このルーチンを呼び出した後、以下のルーチンを呼び出せる。

?hetrs	*AX* = *B* を解く。
?hecon	*A* の条件数を推定する。
?hetri	*A* の逆マトリックスを計算する。

?sptrf

パックド格納形式による対称マトリックスの Bunch-Kaufman 因子分解を行う。

```
call ssptrf ( uplo, n, ap, ipiv, info )
call dsptrf ( uplo, n, ap, ipiv, info )
call csptrf ( uplo, n, ap, ipiv, info )
call zsptrf ( uplo, n, ap, ipiv, info )
```

説明

このルーチンは、パックド格納形式による対称マトリックス A の Bunch-Kaufman 因子分解を実行する。

`uplo='U'` の場合、$A = PUDU^TP^T$

`uplo='L'` の場合、$A = PLDL^TP^T$

P は置換マトリックス、U と L は単位対角成分を含む上三角マトリックスと下三角マトリックス、D は 1×1 と 2×2 の対角ブロックを持つ対称ブロック対角マトリックスである。U と L は、D の 2×2 のブロックに対応する 2×2 の単位対角ブロックを持っている。

入力パラメータ

uplo	`CHARACTER*1`。`'U'` または `'L'` でなければならない。 A の上三角部分と下三角部分のどちらを配列 *ap* にパックするか、また A をどのように因子分解するかを指定する。 *uplo* = `'U'` の場合は、マトリックス A の上三角部分を配列 *ap* に格納し、A を $PUDU^TP^T$ として因子分解する。 *uplo* = `'L'` の場合は、マトリックス A の下三角部分を配列 *ap* に格納し、A を $PLDL^TP^T$ として因子分解する。
n	`INTEGER`。マトリックス A の次数 ($n \geq 0$)。

ap

REAL (ssptrf の場合)
DOUBLE PRECISION (dsptrf の場合)
COMPLEX (csptrf の場合)
DOUBLE COMPLEX (zsptrf の場合)
配列 DIMENSION は少なくとも max(1, *n*(*n*+1)/2)
配列 *ap* には、(*uplo* の指定に従って) マトリックス *A* の上三角部分または下三角部分を**パックド格納形式**で格納する (「マトリックスの格納形式」を参照)。

出力パラメータ

ap

(*uplo* の指定に従って) *A* の上三角部分または下三角部分は、ブロック対角マトリックス *D* の各成分と、係数 *U* (または *L*) の計算に使用された乗数によって上書きされる。

ipiv

INTEGER
配列 DIMENSION は少なくとも max(1, *n*)
交換の結果と *D* のブロック構造の各成分が格納される。

ipiv(i) = *k* > 0 の場合は、d_{ii} は 1 × 1 のブロックである。*A* の *i* 番目の行と列は、*k* 番目の行と列に交換する。

uplo = 'U' と *ipiv(i)* =*ipiv(i-1)* = -*m* < 0 の場合は、*D* の *i* 行 (*i*-1) 列に 2 × 2 のブロックが作成される。*A* の (*i*-1) 番目の行と列は、*m* 番目の行と列に交換する。

uplo = 'L' と *ipiv(i)* =*ipiv(i+1)* = -*m* < 0 の場合は、*D* の *i* 行 (*i*+1) 列に 2 × 2 のブロックが作成される。A の (*i*+1) 番目の行と列は、*m* 番目の行と列に交換する。

info

INTEGER。*info* = 0 の場合は、正常に終了する。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。
info = *i* の場合は、d_{ii} は 0 である。因子分解は完了したが、*D* は完全に特異で、ゼロである。連立 1 次方程式の解の算出に *D* を使用すると、0 による除算が発生する。

アプリケーション・ノート

U と L の 2×2 の単位対角ブロックと単位対角成分は格納されない。U と L のそれ以外の成分は、マトリックス A の対応する列に格納される。ただし、U または L を明示的に復元するには、さらに行の交換が必要である (その必要はほとんどない)。

すべての `i` =1...`n` で `ipiv(i)` = `i` が成り立つ場合は、U (L) のすべての対角線外の成分は、パックド形式で明示的に格納される。

`uplo` = `'U'` の場合、算出された係数 U と D は、摂動マトリックス $A + E$ の正確な係数になる。

$$|E| \leq c(n)\varepsilon P|U||D||U^T|P^T$$

$c(n)$ は `n` の適度な 1 次関数で、ε はマシンの精度である。
`uplo` = `'L'` の場合も、算出された L と D について同様の推定が成り立つ。

浮動小数点演算の合計回数は、実数型の場合は約 $(1/3)n^3$, 複素数型の場合は約 $(4/3)n^3$ になる。

このルーチンを呼び出した後、以下のルーチンを呼び出せる。

?sptrs	$AX = B$ を解く。
?spcon	A の条件数を推定する。
?sptri	A の逆マトリックスを計算する。

?hptrf

パックド格納形式による複素数型
エルミート・マトリックスの
Bunch-Kaufman 因子分解を行う。

```
call chptrf ( uplo, n, ap, ipiv, info )
call zhptrf ( uplo, n, ap, ipiv, info )
```

説明

このルーチンは、パックド格納形式によるエルミート・マトリックスの Bunch-Kaufman 因子分解を実行する。

`uplo='U'` の場合、$A = PUDU^HP^T$

`uplo='L'` の場合、$A = PLDL^HP^T$

A は入力マトリックス、P は置換マトリックス、U と L は単位対角成分を含む上三角マトリックスと下三角マトリックス、D は 1×1 と 2×2 の対角ブロックを持つエルミート・ブロック対角マトリックスである。U と L は、D の 2×2 のブロックに対応する 2×2 の単位対角ブロックを持っている。

入力パラメータ

uplo	`CHARACTER*1`。`'U'` または `'L'` でなければならない。 A の上三角部分と下三角部分のどちらをパックするか、また A をどのように因子分解するかを指定する。 *uplo* = `'U'` の場合は、マトリックス A の上三角部分を配列 *ap* に格納し、A を $PUDU^HP^T$ として因子分解する。 *uplo* = `'L'` の場合は、マトリックス A の下三角部分を配列 *ap* に格納し、A を $PLDL^HP^T$ として因子分解する。
n	`INTEGER`。マトリックス A の次数 ($n \geq 0$)。
ap	`COMPLEX` (`chptrf` の場合) `DOUBLE COMPLEX` (`zhptrf` の場合) 配列 `DIMENSION` は少なくとも max(1, *n*(*n*+1)/2)

配列 *ap* には、(*uplo* の指定に従って) マトリックス A の上三角部分または下三角部分を**パックド格納形式**で格納する (「マトリックスの格納形式」を参照)。

出力パラメータ

ap

(*uplo* の指定に従って) A の上三角部分または下三角部分は、ブロック対角マトリックス D の各成分と、係数 U (または L) の計算に使用された乗数によって上書きされる。

ipiv

INTEGER

配列 DIMENSION は少なくとも max(1, *n*)

交換の結果と D のブロック構造の各成分が格納される。

ipiv(i) = *k* > 0 の場合は、d_{ii} は 1×1 のブロックである。A の *i* 番目の行と列は、*k* 番目の行と列に交換する。

uplo = 'U' と *ipiv(i)* =*ipiv(i-1)* = -*m* < 0 の場合は、D の *i* 行 (*i*-1) 列に 2×2 のブロックが作成される。A の (*i*-1) 番目の行と列は、*m* 番目の行と列に交換する。

uplo = 'L' と *ipiv(i)* =*ipiv(i+1)* = -*m* < 0 の場合は、D の *i* 行 (*i*+1) 列に 2×2 のブロックが作成される。A の (*i*+1) 番目の行と列は、*m* 番目の行と列に交換する。

info

INTEGER。*info* = 0 の場合は、正常に終了する。

info = -*i* の場合は、*i* 番目のパラメータの値が不正である。

info = *i* の場合は、d_{ii} は 0 である。因子分解は完了したが、D は完全に特異で、ゼロである。連立 1 次方程式の解の算出に D を使用すると、0 による除算が発生する。

アプリケーション・ノート

U と L の 2×2 の単位対角ブロックと単位対角成分は格納されない。U と L のそれ以外の成分は、配列 *a* の対応する列に格納される。ただし、U または L を明示的に復元するには、さらに行の交換が必要である (その必要はほとんどない)。

すべての *i* =1 . . . *n* について *ipiv*(i) = *i* が成り立つ場合は、U (L) のすべての対角線外の成分は、配列 *a* の対応する成分に明示的に格納される。

uplo = 'U' の場合、算出された係数 U と D は、摂動マトリックス $A + E$ の正確な係数になる。

$$|E| \leq c(n)\varepsilon P|U||D||U^T|P^T$$

$c(n)$ は *n* の適度な 1 次関数で、ε はマシンの精度である。
uplo = 'L' の場合も、算出された L と D について同様の推定が成り立つ。

浮動小数点演算の合計回数は、約 $(4/3)n^3$ になる。

このルーチンを呼び出した後、以下のルーチンを呼び出せる。

?hptrs	$AX = B$ を解く。
?hpcon	A の条件数を推定する。
?hptri	A の逆マトリックスを計算する。

連立 1 次方程式を解くためのルーチン

この節では、連立 1 次方程式を解くための LAPACK ルーチンについて説明する。通常は、これらのルーチンを呼び出す前に、連立方程式のマトリックスを因子分解する必要がある (本章の「計算ルーチン」を参照)。ただし、解を求める連立方程式が三角マトリックスを持つ場合は、因子分解の必要はない。

?getrs

LU 因子分解された正方マトリックスを使って、複数の右辺を持つ連立 1 次方程式を解く。

```
call sgetrs (trans, n, nrhs, a, lda, ipiv, b, ldb, info)
call dgetrs (trans, n, nrhs, a, lda, ipiv, b, ldb, info)
call cgetrs (trans, n, nrhs, a, lda, ipiv, b, ldb, info)
call zgetrs (trans, n, nrhs, a, lda, ipiv, b, ldb, info)
```

説明

このルーチンは、以下の連立 1 次方程式を X について解く。

$AX = B$	(*trans*='N' の場合)
$A^TX = B$	(*trans*='T' の場合)
$A^HX = B$	(*trans*='C' の場合。ただし、複素マトリックスのみ)

このルーチンを呼び出す前に、?getrf を呼び出して、A の LU 因子分解を行う必要がある。

入力パラメータ

trans	CHARACTER*1。'N' または 'T' または 'C' でなければならない。 方程式の形式を指定する。 *trans* = 'N' の場合は、$AX = B$ を X について解く。 *trans* = 'T' の場合は、$A^TX = B$ を X について解く。 *trans* = 'C' の場合は、$A^HX = B$ を X について解く。
n	INTEGER。A の次数。B の行の数 ($n \geq 0$)。
nrhs	INTEGER。右辺の数 ($nrhs \geq 0$) 。
a, *b*	REAL (sgetrs の場合) DOUBLE PRECISION (dgetrs の場合) COMPLEX (cgetrs の場合) DOUBLE COMPLEX (zgetrs の場合) 配列 : *a*(*lda*, *), *b*(*ldb*, *)

配列 *a* には、マトリックス *A* を格納する。
配列 *b* には、マトリックス *B* を格納する。このマトリックスの列は、連立方程式の右辺である。
a の第 2 次元は、少なくとも max(1, *n*) でなければならない。*b* の第 2 次元は、少なくとも max(1, *nrhs*) でなければならない。

lda INTEGER。*a* の第 1 次元。*lda* ≥ max(1, *n*)

ldb INTEGER。*b* の第 1 次元。*ldb* ≥ max(1, *n*)

ipiv INTEGER
配列 DIMENSION は少なくとも max(1, *n*)
?getrf によって返される *ipiv* 配列。

出力パラメータ

b 解のマトリックス *X* によって上書きされる。

info INTEGER。*info* = 0 の場合は、正常に終了する。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

各右辺 *b* について算出された解は、摂動連立方程式 $(A + E)x = b$ の正確な解になる。

$$|E| \le c(n)\varepsilon P|L||U|$$

$c(n)$ は n の適度な 1 次関数で、ε はマシンの精度である。

x_0 が真の解である場合、算出された解 x は、次の誤差範囲を満たす。

$$\frac{\|x - x_0\|_\infty}{\|x\|_\infty} \le c(n)\,\mathrm{cond}(A, x)\varepsilon$$

ただし、$\mathrm{cond}(A, x) = \left\| |A^{-1}|\,|A|\,|x| \right\|_\infty / \|x\|_\infty \le \|A^{-1}\|_\infty \|A\|_\infty = \kappa_\infty(A)$

$\mathrm{cond}(A, x)$ は、$\kappa_\infty(A)$ よりはるかに小さくなる場合がある。A^T と A^H の条件数は、$\kappa_\infty(A)$ に等しいことも、等しくないこともある。

1 つの右辺ベクトル *b* の浮動小数点演算のおおよその回数は、実数型の場合は $2n^2$, 複素数型の場合は $8n^2$ になる。

条件数 $\kappa_\infty(A)$ を推定するには、?gecon を呼び出す。
解の精度を改善して誤差を推定するには、?gerfs を呼び出す。

?gbtrs

LU 因子分解されたバンド・マトリックスを使って、複数の右辺を持つ連立 1 次方程式を解く。

```
call sgbtrs (trans, n, kl, ku, nrhs, ab, ldab, ipiv, b, ldb, info)
call dgbtrs (trans, n, kl, ku, nrhs, ab, ldab, ipiv, b, ldb, info)
call cgbtrs (trans, n, kl, ku, nrhs, ab, ldab, ipiv, b, ldb, info)
call zgbtrs (trans, n, kl, ku, nrhs, ab, ldab, ipiv, b, ldb, info)
```

説明

このルーチンは、以下の連立 1 次方程式を X について解く。

$AX = B$	`trans='N'` の場合
$A^TX = B$	`trans='T'` の場合
$A^HX = B$	`trans='C'` の場合 (ただし、複素マトリックスのみ)

A は、`kl` 個のゼロ以外の劣対角成分と `ku` 個のゼロ以外の優対角成分を持つ、*LU* 因子分解された `n` 次の一般バンド・マトリックスである。このルーチンを呼び出す前に、?gbtrf を呼び出して、A の *LU* 因子分解を行う必要がある。

入力パラメータ

`trans`	`CHARACTER*1`。`'N'` または `'T'` または `'C'` でなければならない。
`n`	`INTEGER`。A の次数。B の行の数 ($n \geq 0$)。
`kl`	`INTEGER`。A のバンドの中の劣対角成分の数 ($kl \geq 0$)。
`ku`	`INTEGER`。A のバンドの中の優対角成分の数 ($ku \geq 0$)。
`nrhs`	`INTEGER`。右辺の数 ($nrhs \geq 0$)。
`ab, b`	`REAL` (`sgbtrs` の場合) `DOUBLE PRECISION` (`dgbtrs` の場合) `COMPLEX` (`cgbtrs` の場合) `DOUBLE COMPLEX` (`zgbtrs` の場合) 配列 : `ab(ldab, *)`, `b(ldb, *)`

配列 *ab* には、マトリックス *A* を**バンド格納形式**で格納する (「マトリックスの格納形式」を参照)。
配列 *b* には、マトリックス *B* を格納する。このマトリックスの列は、連立方程式の右辺である。
ab の第 2 次元は、少なくとも max(1, *n*) でなければならない。
b の第 2 次元は、少なくとも max(1, *nrhs*) でなければならない。

ldab INTEGER。配列 *ab* の第 1 次元。
($ldab \geq 2kl + ku + 1$)

ldb INTEGER。*b* の第 1 次元。$ldb \geq \max(1, n)$

ipiv INTEGER。配列 DIMENSION は少なくとも max(1, *n*)
?gbtrf によって返される *ipiv* 配列。

出力パラメータ

b 解のマトリックス *X* によって上書きされる。

info INTEGER。*info* = 0 の場合は、正常に終了する。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

各右辺 *b* について算出された解は、摂動連立方程式 $(A + E)x = b$ の正確な解になる。

$$|E| \leq c(kl + ku + 1)\varepsilon P|L||U|$$

$c(k)$ は、k の適度な 1 次関数で、ε はマシンの精度である。

x_0 が真の解である場合、算出された解 *x* は、次の誤差範囲を満たす。

$$\frac{\|x - x_0\|_\infty}{\|x\|_\infty} \leq c(kl + ku + 1)\ \mathrm{cond}(A, x)\varepsilon$$

ただし、$\mathrm{cond}(A, x) = \left\| |A^{-1}|\,|A|\,|x| \right\|_\infty / \|x\|_\infty \leq \|A^{-1}\|_\infty \|A\|_\infty = \kappa_\infty(A)$

$\mathrm{cond}(A, x)$ は、$\kappa_\infty(A)$ よりはるかに小さくなる場合がある。A^T と A^H の条件数は、$\kappa_\infty(A)$ に等しいことも、等しくないこともある。

1 つの右辺ベクトルの浮動小数点演算のおおよその回数は、実数型の場合は $2n(ku + 2kl)$ になる。複素数型の場合は、この 4 倍になる。これらの推定では、*kl* と *ku* は $\min(m, n)$ よりはるかに小さいものとする。

条件数 $\kappa_\infty(A)$ を推定するには、?gbcon を呼び出す。
解の精度を改善して誤差を推定するには、?gbrfs を呼び出す。

?gttrs

?gttrf によって行われた *LU* 因子分解を
使用して、三重対角マトリックスを
係数行列とする連立 1 次方程式を解く。

```
call sgttrs (trans, n, nrhs, dl, d, du, du2, ipiv, b, ldb, info)
call dgttrs (trans, n, nrhs, dl, d, du, du2, ipiv, b, ldb, info)
call cgttrs (trans, n, nrhs, dl, d, du, du2, ipiv, b, ldb, info)
call zgttrs (trans, n, nrhs, dl, d, du, du2, ipiv, b, ldb, info)
```

説明

このルーチンは、複数の右辺を持つ以下の連立 1 次方程式を、*X* について解く。

$AX = B$	(*trans*='N' の場合)
$A^TX = B$	(*trans*='T' の場合)
$A^HX = B$	(*trans*='C' の場合。複素マトリックスのみ)

このルーチンを呼び出す前に、?gttrf を呼び出して、*A* の *LU* 因子分解を行う必要がある。

入力パラメータ

trans
CHARACTER*1。'N' または 'T' または 'C' でなければならない。
方程式の形式を指定する。
trans = 'N' の場合は、$AX = B$ を *X* について解く。
trans = 'T' の場合は、$A^TX = B$ を *X* について解く。
trans = 'C' の場合は、$A^HX = B$ を *X* について解く。

n
INTEGER。*A* の次数 ($n \geq 0$)。

nrhs
INTEGER。右辺の数、すなわち、*B* の列の数 ($nrhs \geq 0$)。

dl, d, du, du2, b
REAL (sgttrs の場合)
DOUBLE PRECISION (dgttrs の場合)
COMPLEX (cgttrs の場合)
DOUBLE COMPLEX (zgttrf の場合)

配列 : dl(n - 1), d(n), du(n - 1), du2(n - 2), b(ldb, nrhs)
配列 dl には、A の LU 因子分解で得られたマトリックス L を定義する (n - 1) 個の乗数が格納される。
配列 d には、A の LU 因子分解で得られた上三角マトリックス U の n 個の対角成分が格納される。
配列 du には、U の最初の優対角成分の (n - 1) 個の成分が格納される。
配列 du2 には、U の 2 番目の優対角成分の (n - 2) 個の成分が格納される。
配列 b には、マトリックス B が格納される。このマトリックスの列は、連立方程式の右辺である。

ldb　INTEGER。b のリーディング・ディメンジョン。
ldb ≥ max(1, n)

ipiv　INTEGER
配列 DIMENSION は (n)
?gttrf によって返される ipiv 配列。

出力パラメータ

b　解のマトリックス X によって上書きされる。

info　INTEGER。info = 0 の場合は、ルーチンの実行は正常に終了した。
info = -i の場合は、i 番目のパラメータの値が不正である。

アプリケーション・ノート

各右辺 b について算出された解は、摂動連立方程式 $(A + E)x = b$ の正確な解になる。

$$|E| \le c(n)\varepsilon P|L||U|$$

$c(n)$ は n の適度な 1 次関数で、ε はマシンの精度である。

x_0 が真の解である場合、算出された解 x は、次の誤差範囲を満たす。

$$\frac{\|x - x_0\|_\infty}{\|x\|_\infty} \le c(n)\,\mathrm{cond}(A, x)\varepsilon$$

ここで、$\mathrm{cond}(A, x) = \left\| |A^{-1}|\,|A|\,|x| \right\|_\infty / \|x\|_\infty \le \|A^{-1}\|_\infty \|A\|_\infty = \kappa_\infty(A)$

cond(A, x) は、$\kappa_\infty(A)$ よりはるかに小さくなる場合がある。A^T と A^H の条件数は、$\kappa_\infty(A)$ に等しいことも、等しくないこともある。

1 つの右辺ベクトル b の浮動小数点演算のおおよその回数は、実数型の場合は $2n^2$、複素数型の場合は $8n^2$ になる。

条件数 $\kappa_\infty(A)$ を推定するには、?gecon を呼び出す。
解の精度を改善して誤差を推定するには、?gerfs を呼び出す。

?potrs

コルスキー因子分解された正値対称
(エルミート) マトリックスを使って、
連立 1 次方程式を解く。

```
call spotrs ( uplo, n, nrhs, a, lda, b, ldb, info )
call dpotrs ( uplo, n, nrhs, a, lda, b, ldb, info )
call cpotrs ( uplo, n, nrhs, a, lda, b, ldb, info )
call zpotrs ( uplo, n, nrhs, a, lda, b, ldb, info )
```

説明

このルーチンは、正値対称マトリックス (複素データの場合は正値エルミート) マトリックス A を持つ連立 1 次方程式 $AX = B$ を、A のコルスキー因子分解に基づいて、X について解く。

$A = U^H U$ (*uplo* = 'U' の場合)

$A = LL^H$ (*uplo* = 'L' の場合)

L は下三角マトリックス、U は上三角マトリックスである。マトリックス B の各列に格納されている複数の右辺を使用して、この連立方程式を解く。

このルーチンを呼び出す前に、?potrf を呼び出して、A のコルスキー因子分解を行う必要がある。

入力パラメータ

uplo CHARACTER*1。'U' または 'L' でなければならない。
入力マトリックス A の因子分解の方法を指定する。
uplo = 'U' の場合は、配列 *a* には、コルスキー因子分解 $A = U^H U$ の係数 U を格納する。
uplo = 'L' の場合は、配列 *a* には、コルスキー因子分解 $A = LL^H$ の係数 L を格納する。

n INTEGER。マトリックス A の次数 ($n \geq 0$)。

nrhs INTEGER。右辺の数 ($nrhs \geq 0$)。

a, b	REAL (spotrs の場合) DOUBLE PRECISION (dpotrs の場合) COMPLEX (cpotrs の場合) DOUBLE COMPLEX (zpotrs の場合) 配列 : *a(lda, *)*, *b(ldb, *)* 配列 *a* には、係数 *U* または *L* を格納する (*uplo* を参照)。 配列 *b* には、マトリックス *B* を格納する。このマトリックスの列は、連立方程式の右辺である。 *a* の第 2 次元は、少なくとも max(1, *n*) でなければならない。*b* の第 2 次元は、少なくとも max(1, *nrhs*) でなければならない。
lda	INTEGER。*a* の第 1 次元。*lda* ≥ max(1, *n*)
ldb	INTEGER。*b* の第 1 次元。*ldb* ≥ max(1, *n*)

出力パラメータ

b	解のマトリックス *X* によって上書きされる。
info	INTEGER。*info* = 0 の場合は、正常に終了する。 *info* = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

uplo = 'U' の場合、各右辺 *b* について算出された解は、摂動連立方程式 $(A + E)x = b$ の正確な解になる。

$$|E| \le c(n)\varepsilon \left|U^H\right||U|$$

$c(n)$ は *n* の適度な 1 次関数で、ε はマシンの精度である。
uplo = 'L' の場合も、同様の推定が成り立つ。

x_0 が真の解である場合、算出された解 x は、次の誤差範囲を満たす。

$$\frac{\|x - x_0\|_\infty}{\|x\|_\infty} \le c(n)\,\mathrm{cond}(A, x)\varepsilon$$

ただし、$\mathrm{cond}(A, x) = \left\| |A^{-1}|\,|A|\,|x| \right\|_\infty / \|x\|_\infty \le \|A^{-1}\|_\infty \|A\|_\infty = \kappa_\infty(A)$

$\mathrm{cond}(A, x)$ は、$\kappa_\infty(A)$ よりはるかに小さくなる場合がある。
1 つの右辺ベクトル *b* の浮動小数点演算のおおよその回数は、実数型の場合は $2n^2$、複素数型の場合は $8n^2$ になる。

条件数 $\kappa_\infty(A)$ を推定するには、?pocon を呼び出す。
解の精度を改善して誤差を推定するには、?porfs を呼び出す。

?pptrs

コルスキー因子分解されたパックド形式の
正値対称 (エルミート) マトリックスを
使って、連立 1 次方程式を解く。

```
call spptrs ( uplo, n, nrhs, ap, b, ldb, info )
call dpptrs ( uplo, n, nrhs, ap, b, ldb, info )
call cpptrs ( uplo, n, nrhs, ap, b, ldb, info )
call zpptrs ( uplo, n, nrhs, ap, b, ldb, info )
```

説明

このルーチンは、パックド形式の正値対称 (複素データの場合は正値エルミート) マトリックス A を持つ連立 1 次方程式 $AX = B$ を、A のコルスキー因子分解に基づいて、X について解く。

$A = U^H U$ (*uplo* = 'U' の場合)

$A = LL^H$ (*uplo* = 'L' の場合)

L は下三角マトリックス、U は上三角マトリックスである。マトリックス B の各列に格納されている複数の右辺を使用して、この連立方程式を解く。

このルーチンを呼び出す前に、?pptrf を呼び出して、A のコルスキー因子分解を行う必要がある。

入力パラメータ

uplo CHARACTER*1。'U' または 'L' でなければならない。入力マトリックス A の因子分解の方法を指定する。
uplo = 'U' の場合は、配列 *a* には、コルスキー因子分解 $A = U^H U$ の係数 U をパックド形式で格納する。
uplo = 'L' の場合は、配列 *a* には、コルスキー因子分解 $A = LL^H$ の係数 L をパックド形式で格納する。

n INTEGER。マトリックス A の次数 ($n \geq 0$)。

nrhs INTEGER。右辺の数 ($nrhs \geq 0$)。

ap, b	REAL (spptrs の場合) DOUBLE PRECISION (dpptrs の場合) COMPLEX (cpptrs の場合) DOUBLE COMPLEX (zpptrs の場合) 配列、*ap(*)*, *b(ldb, *)* *ap* の次元は、少なくとも max(1, *n*(*n*+1)/2) でなければならない。 配列 *ap* には、*uplo* の指定に従って、係数 *U* または *L* を**パックド格納形式**で格納する (「マトリックスの格納形式」を参照)。 配列 *b* には、マトリックス *B* を格納する。このマトリックスの列は、連立方程式の右辺である。*b* の第 2 次元は、少なくとも max(1, *nrhs*) でなければならない。
ldb	INTEGER。*b* の第 1 次元。*ldb* ≥ max(1, *n*)

出力パラメータ

b	解のマトリックス *X* によって上書きされる。
info	INTEGER。*info* = 0 の場合は、正常に終了する。 *info* = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

uplo = 'U' の場合、各右辺 *b* について算出された解は、摂動連立方程式 $(A + E)x = b$ の正確な解になる。

$$|E| \le c(n)\varepsilon \left|U^H\right||U|$$

$c(n)$ は *n* の適度な 1 次関数で、ε はマシンの精度である。

uplo = 'L' の場合も、同様の推定が成り立つ。

x_0 が真の解である場合、算出された解 x は、次の誤差範囲を満たす。

$$\frac{\|x - x_0\|_\infty}{\|x\|_\infty} \le c(n)\ \mathrm{cond}(A, x)\varepsilon$$

ただし、$\mathrm{cond}(A, x) = \left\| |A^{-1}|\,|A|\,|x| \right\|_\infty / \|x\|_\infty \le \|A^{-1}\|_\infty \|A\|_\infty = \kappa_\infty(A)$

$\mathrm{cond}(A, x)$ は、$\kappa_\infty(A)$ よりはるかに小さくなる場合がある。

1 つの右辺ベクトル *b* の浮動小数点演算のおおよその回数は、実数型の場合は $2n^2$、複素数型の場合は $8n^2$ になる。

条件数 $\kappa_\infty(A)$ を推定するには、?ppcon を呼び出す。
解の精度を改善して誤差を推定するには、?pprfs を呼び出す。

?pbtrs

コルスキー因子分解された正値対称
(エルミート) バンド・マトリックスを
使って、連立 1 次方程式を解く。

```
call spbtrs (uplo, n, kd, nrhs, ab, ldab, b, ldb, info)
call dpbtrs (uplo, n, kd, nrhs, ab, ldab, b, ldb, info)
call cpbtrs (uplo, n, kd, nrhs, ab, ldab, b, ldb, info)
call zpbtrs (uplo, n, kd, nrhs, ab, ldab, b, ldb, info)
```

説明

このルーチンは、正値対称 (複素データの場合は正値エルミート) バンド・マトリックス A を持つ連立 1 次方程式 $AX = B$ を、A のコルスキー因子分解に基づいて、X について解く。

$A = U^H U$ (`uplo='U'` の場合)

$A = LL^H$ (`uplo='L'` の場合)

L は下三角マトリックス、U は上三角マトリックスである。マトリックス B の各列に格納されている複数の右辺を使用して、この連立方程式を解く。

このルーチンを呼び出す前に、?pbtrf を呼び出して、A のコルスキー因子分解をバンド格納形式で計算する必要がある。

入力パラメータ

uplo `CHARACTER*1`。`'U'` または `'L'` でなければならない。入力マトリックス A の因子分解の方法を指定する。
uplo = `'U'` の場合は、*a* には、因子分解 $A = U^H U$ の係数 U をバンド格納形式で格納する。
uplo = `'L'` の場合は、配列 *a* には、因子分解 $A = LL^H$ の係数 L をバンド格納形式で格納する。

n `INTEGER`。マトリックス A の次数 ($n \geq 0$)。

kd `INTEGER`。A の優対角成分または劣対角成分の数 ($kd \geq 0$)。

nrhs `INTEGER`。右辺の数 ($nrhs \geq 0$)。

ab, b	REAL (spbtrs の場合) DOUBLE PRECISION (dpbtrs の場合) COMPLEX (cpbtrs の場合) DOUBLE COMPLEX (zpbtrs の場合) 配列 : *ab(ldab, *)*, *b(ldb, *)* 配列 *ab* には、因子分解ルーチンによって返されるコルスキー係数を**バンド格納形式**で格納する。 配列 *b* には、マトリックス B を格納する。このマトリックスの列は、連立方程式の右辺である。 *ab* の第 2 次元は、少なくとも max(1, *n*) でなければならない。 *b* の第 2 次元は、少なくとも max(1, *nrhs*) でなければならない。
ldab	INTEGER。配列 *ab* の第 1 次元。(*ldab* ≥ *kd* +1)
ldb	INTEGER。*b* の第 1 次元。*ldb* ≥ max(1, *n*)

出力パラメータ

b	解のマトリックス X によって上書きされる。
info	INTEGER。*info* = 0 の場合は、正常に終了する。 *info* = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

各右辺 b について算出された解は、摂動連立方程式 $(A + E)x = b$ の正確な解になる。

$$|E| \le c(kd+1)\varepsilon P|U^H||U| \quad \text{または} \quad |E| \le c(kd+1)\varepsilon P|L^H||L|$$

$c(k)$ は k の適度な関数で、ε はマシンの精度である。

x_0 が真の解である場合、算出された解 x は、次の誤差範囲を満たす。

$$\frac{\|x - x_0\|_\infty}{\|x\|_\infty} \le c(kd+1)\ \mathrm{cond}(A, x)\varepsilon$$

ただし、$\mathrm{cond}(A, x) = \left\| |A^{-1}|\,|A|\,|x| \right\|_\infty / \|x\|_\infty \le \|A^{-1}\|_\infty \|A\|_\infty = \kappa_\infty(A)$

$\mathrm{cond}(A, x)$ は、$\kappa_\infty(A)$ よりはるかに小さくなる場合がある。

1 つの右辺ベクトルの浮動小数点演算のおおよその回数は、実数型の場合は 4*n***kd*, 複素数型の場合は 16*n***kd* になる。

条件数 $\kappa_\infty(A)$ を推定するには、?pbcon を呼び出す。
解の精度を改善して誤差を推定するには、?pbrfs を呼び出す。

?pttrs

?pttrf によって行われた因子分解を
使用して、正値の対称 (エルミート)
三重対角マトリックスを係数行列とする
連立 1 次方程式を解く。

```
call spttrs (n, nrhs, d, e, b, ldb, info)
call dpttrs (n, nrhs, d, e, b, ldb, info)
call cpttrs (uplo, n, nrhs, d, e, b, ldb, info)
call zpttrs (uplo, n, nrhs, d, e, b, ldb, info)
```

説明

このルーチンは、正値の対称 (エルミート) 三重対角マトリックス A を係数行列とする連立 1 次方程式 $AX = B$ を、X について解く。
このルーチンを呼び出す前に、?pttrf を呼び出して、A の LDL^H または U^HDU 因子分解を行う必要がある。

入力パラメータ

uplo CHARACTER*1。cpttrs/zpttrs でのみ使用される。
'U' または 'L' でなければならない。
三重対角マトリックス A の優対角成分と劣対角成分のどちらを格納するかと、A の因子分解の方法を指定する。
uplo = 'U' の場合は、配列 *e* には A の優対角成分が格納され、A は U^HDU として因子分解される。
uplo = 'L' の場合は、配列 *e* には A の劣対角成分が格納され、A は LDL^H として因子分解される。

n INTEGER。A の次数 ($n \geq 0$)。

nrhs INTEGER。右辺の数、すなわち、マトリックス B の列の数 ($nrhs \geq 0$)。

d
REAL (spttrs, cpttrs の場合)
DOUBLE PRECISION (dpttrs, zpttrs の場合)
配列、次元は (*n*)。?pttrf による因子分解で得られた対角マトリックス *D* の対角成分を格納する。

e, b
REAL (spttrs, cpttrs の場合)
DOUBLE PRECISION (dpttrs の場合)
COMPLEX (cpttrs の場合)
DOUBLE COMPLEX (zpttrs の場合)
配列 : *e*(*n* - 1), *b*(*ldb, nrhs*)
配列 *e* には、?pttrf による因子分解で得られた単位二重対角係数 *U* または *L* の (*n* - 1) 個の対角線外の成分が格納される (*uplo* を参照)。
配列 *b* には、マトリックス *B* が格納される。このマトリックスの列は、連立方程式の右辺である。

ldb
INTEGER。*b* のリーディング・ディメンジョン。
ldb ≥ max(1, *n*)

出力パラメータ

b
解のマトリックス *X* によって上書きされる。

info
INTEGER。*info* = 0 の場合は、ルーチンの実行は正常に終了した。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。

?sytrs

UDU または *LDL* 因子分解された対称マトリックスを使って、連立 1 次方程式を解く。

```
call ssytrs (uplo, n, nrhs, a, lda, ipiv, b, ldb, info)
call dsytrs (uplo, n, nrhs, a, lda, ipiv, b, ldb, info)
call csytrs (uplo, n, nrhs, a, lda, ipiv, b, ldb, info)
call zsytrs (uplo, n, nrhs, a, lda, ipiv, b, ldb, info)
```

説明

このルーチンは、対称マトリックス A を持つ連立 1 次方程式 $AX = B$ を、A の Bunch-Kaufman 因子分解に基づいて、X について解く。

`uplo='U'` の場合、$A = PUDU^TP^T$

`uplo='L'` の場合、$A = PLDL^TP^T$

P は置換マトリックス、U と L は単位対角成分を含む上三角マトリックスと下三角マトリックス、D は対称ブロック対角マトリックスである。マトリックス B の各列に格納されている複数の右辺を使用して、この連立方程式を解く。このルーチンには、因子分解ルーチン ?sytrf によって返される係数 U (または L) と配列 *ipiv* を与える必要がある。

入力パラメータ

uplo `CHARACTER*1`。`'U'` または `'L'` でなければならない。入力マトリックス A の因子分解の方法を指定する。

uplo = `'U'` の場合は、配列 *a* には、因子分解 $A = PUDU^TP^T$ の上三角係数 U を格納する。

uplo = `'L'` の場合は、配列 *a* には、因子分解 $A = PLDL^TP^T$ の上三角係数 L を格納する。

n `INTEGER`。マトリックス A の次数 ($n \geq 0$)。

nrhs `INTEGER`。右辺の数 ($nrhs \geq 0$)。

ipiv `INTEGER`。配列 `DIMENSION` は少なくとも max(1, *n*) ?sytrf によって返される *ipiv* 配列。

a, *b*	REAL (ssytrs の場合) DOUBLE PRECISION (dsytrs の場合) COMPLEX (csytrs の場合) DOUBLE COMPLEX (zsytrs の場合) 配列 : *a*(*lda*, *), *b*(*ldb*, *) 配列 *a* には、係数 U または L を格納する (*uplo* を参照)。 配列 *b* には、マトリックス B を格納する。このマトリックスの列は、連立方程式の右辺である。 *a* の第 2 次元は、少なくとも max(1, *n*) でなければならない。*b* の第 2 次元は、少なくとも max(1, *nrhs*) でなければならない。
lda	INTEGER。*a* の第 1 次元。*lda* ≥ max(1, *n*)
ldb	INTEGER。*b* の第 1 次元。*ldb* ≥ max(1, *n*)

出力パラメータ

b	解のマトリックス X によって上書きされる。
info	INTEGER。*info* = 0 の場合は、正常に終了する。 *info* = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

各右辺 b について算出された解は、摂動連立方程式 $(A + E)x = b$ の正確な解になる。

$$|E| \le c(n)\varepsilon\, P|U||D|\left|U^T\right|P^T \quad \text{または} \quad |E| \le c(n)\varepsilon\, P|L||D|\left|L^T\right|P^T$$

$c(n)$ は n の適度な 1 次関数で、ε はマシンの精度である。

x_0 が真の解である場合、算出された解 x は、次の誤差範囲を満たす。

$$\frac{\|x - x_0\|_\infty}{\|x\|_\infty} \le c(n)\ \mathrm{cond}(A, x)\,\varepsilon$$

ただし、$\mathrm{cond}(A, x) = \left\| |A^{-1}|\,|A|\,|x| \right\|_\infty / \|x\|_\infty \le \|A^{-1}\|_\infty \|A\|_\infty = \kappa_\infty(A)$

$\mathrm{cond}(A, x)$ は、$\kappa_\infty(A)$ よりはるかに小さくなる場合がある。

1 つの右辺ベクトルの浮動小数点演算の合計回数は、実数型の場合は約 $2n^2$、複素数型の場合は約 $8n^2$ になる。

条件数 $\kappa_\infty(A)$ を推定するには、?sycon を呼び出す。
解の精度を改善して誤差を推定するには、?syrfs を呼び出す。

?hetrs

UDU または *LDL* 因子分解された
エルミート・マトリックスを使って、
連立 1 次方程式を解く。

```
call chetrs (uplo, n, nrhs, a, lda, ipiv, b, ldb, info)
call zhetrs (uplo, n, nrhs, a, lda, ipiv, b, ldb, info)
```

説明

このルーチンは、エルミート・マトリックス A を持つ連立 1 次方程式 $AX = B$ を、A の Bunch-Kaufman 因子分解に基づいて、X について解く。

`uplo='U'` の場合、$A = PUDU^HP^T$

`uplo='L'` の場合、$A = PLDL^HP^T$

P は置換マトリックス、U と L は単位対角成分を含む上三角マトリックスと下三角マトリックス、D は対称ブロック対角マトリックスである。マトリックス B の各列に格納されている複数の右辺を使用して、この連立方程式を解く。このルーチンには、因子分解ルーチン ?hetrf によって返される係数 U (または L) と配列 `ipiv` を与える必要がある。

入力パラメータ

`uplo`	`CHARACTER*1`。`'U'` または `'L'` でなければならない。入力マトリックス A の因子分解の方法を指定する。 `uplo = 'U'` の場合は、配列 `a` には、因子分解 $A = PUDU^HP^T$ の上三角係数 U を格納する。 `uplo = 'L'` の場合は、配列 `a` には、因子分解 $A = PLDL^HP^T$ の上三角係数 L を格納する。
`n`	`INTEGER`。マトリックス A の次数 ($n \geq 0$)。
`nrhs`	`INTEGER`。右辺の数 ($nrhs \geq 0$)。
`ipiv`	`INTEGER`。配列 `DIMENSION` は少なくとも max(1, n) ?hetrf によって返される `ipiv` 配列。

a, b	COMPLEX (chetrs の場合) DOUBLE COMPLEX (zhetrs の場合) 配列 : *a(lda, *)*, *b(ldb, *)* 配列 *a* には、係数 U または L を格納する (*uplo* を参照)。 配列 *b* には、マトリックス B を格納する。このマトリックスの列は、連立方程式の右辺である。 *a* の第 2 次元は、少なくとも max(1, *n*) でなければならない。*b* の第 2 次元は、少なくとも max(1, *nrhs*) でなければならない。
lda	INTEGER。*a* の第 1 次元。*lda* ≥ max(1, *n*)
ldb	INTEGER。*b* の第 1 次元。*ldb* ≥ max(1, *n*)

出力パラメータ

b	解のマトリックス X によって上書きされる。
info	INTEGER。*info* = 0 の場合は、正常に終了する。 *info* = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

各右辺 b について算出された解は、摂動連立方程式 $(A + E)x = b$ の正確な解になる。

$$|E| \le c(n)\varepsilon\, P|U||D|\left|U^H\right|P^T \quad \text{または} \quad |E| \le c(n)\varepsilon\, P|L||D|\left|L^H\right|P^T$$

$c(n)$ は n の適度な 1 次関数で、ε はマシンの精度である。

x_0 が真の解である場合、算出された解 x は、次の誤差範囲を満たす。

$$\frac{\|x - x_0\|_\infty}{\|x\|_\infty} \le c(n)\ \mathrm{cond}(A, x)\varepsilon$$

ただし、$\mathrm{cond}(A, x) = \left\| |A^{-1}|\,|A|\,|x| \right\|_\infty / \|x\|_\infty \le \|A^{-1}\|_\infty \|A\|_\infty = \kappa_\infty(A)$

$\mathrm{cond}(A, x)$ は、$\kappa_\infty(A)$ よりはるかに小さくなる場合がある。

1 つの右辺ベクトルの浮動小数点演算の合計回数は、約 $8n^2$ になる。

条件数 $\kappa_\infty(A)$ を推定するには、?hecon を呼び出す。
解の精度を改善して誤差を推定するには、?herfs を呼び出す。

?sptrs

UDU または *LDL* 因子分解されたパックド格納形式による対称マトリックスを使って、連立 1 次方程式を解く。

```
call ssptrs ( uplo, n, nrhs, ap, ipiv, b, ldb, info )
call dsptrs ( uplo, n, nrhs, ap, ipiv, b, ldb, info )
call csptrs ( uplo, n, nrhs, ap, ipiv, b, ldb, info )
call zsptrs ( uplo, n, nrhs, ap, ipiv, b, ldb, info )
```

説明

このルーチンは、対称マトリックス A を持つ連立 1 次方程式 $AX = B$ を、A の Bunch-Kaufman 因子分解に基づいて、X について解く。

`uplo='U'` の場合、$A = PUDU^TP^T$

`uplo='L'` の場合、$A = PLDL^TP^T$

P は置換マトリックス、U と L は単位対角成分を含む**パックド形式**の上三角マトリックスと下三角マトリックス、D は対称ブロック対角マトリックスである。マトリックス B の各列に格納されている複数の右辺を使用して、この連立方程式を解く。このルーチンには、因子分解ルーチン ?sptrf によって返される係数 U (または L) と配列 `ipiv` を与える必要がある。

入力パラメータ

`uplo`	`CHARACTER*1`。`'U'` または `'L'` でなければならない。入力マトリックス A の因子分解の方法を指定する。 `uplo` = `'U'` の場合は、配列 `ap` には、因子分解 $A = PUDU^TP^T$ のパックド形式の係数 U を格納する。 `uplo` = `'L'` の場合は、配列 `ap` には、因子分解 $A = PLDL^TP^T$ のパックド形式の係数 L を格納する。
`n`	`INTEGER`。マトリックス A の次数 ($n \geq 0$)。
`nrhs`	`INTEGER`。右辺の数 ($nrhs \geq 0$)。
`ipiv`	`INTEGER`。配列 `DIMENSION` は少なくとも max(1, `n`) ?sptrf によって返される `ipiv` 配列。
`ap, b`	`REAL` (`ssptrs` の場合) `DOUBLE PRECISION` (`dsptrs` の場合) `COMPLEX` (`csptrs` の場合) `DOUBLE COMPLEX` (`zsptrs` の場合) 配列、`ap(*)`, `b(ldb, *)`

	ap の次元は、少なくとも max(1, *n*(*n*+1)/2) でなければならない。 配列 *ap* には、*uplo* の指定に従って、係数 *U* または *L* を**パックド格納形式**で格納する (「マトリックスの格納形式」を参照)。 配列 *b* には、マトリックス *B* を格納する。このマトリックスの列は、連立方程式の右辺である。*b* の第 2 次元は、少なくとも max(1, *nrhs*) でなければならない。
ldb	INTEGER。*b* の第 1 次元。*ldb* ≥ max(1, *n*)

出力パラメータ

b	解のマトリックス *X* によって上書きされる。
info	INTEGER。*info* = 0 の場合は、正常に終了する。 *info* = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

各右辺 *b* について算出された解は、摂動連立方程式 $(A + E)x = b$ の正確な解になる。

$$|E| \le c(n)\varepsilon\, P|U||D|\left|U^T\right|P^T \quad \text{または} \quad |E| \le c(n)\varepsilon\, P|L||D|\left|L^T\right|P^T$$

$c(n)$ は n の適度な 1 次関数で、ε はマシンの精度である。

x_0 が真の解である場合、算出された解 x は、次の誤差範囲を満たす。

$$\frac{\|x - x_0\|_\infty}{\|x\|_\infty} \le c(n)\ \mathrm{cond}(A, x)\varepsilon$$

ただし、$\mathrm{cond}(A, x) = \left\| |A^{-1}|\,|A|\,|x| \right\|_\infty / \|x\|_\infty \le \|A^{-1}\|_\infty \|A\|_\infty = \kappa_\infty(A)$

$\mathrm{cond}(A, x)$ は、$\kappa_\infty(A)$ よりはるかに小さくなる場合がある。

1 つの右辺ベクトルの浮動小数点演算の合計回数は、実数型の場合は約 $2n^2$、複素数型の場合は約 $8n^2$ になる。

条件数 $\kappa_\infty(A)$ を推定するには、?spcon を呼び出す。
解の精度を改善して誤差を推定するには、?sprfs を呼び出す。

?hptrs

UDU または *LDL* 因子分解されたパックド格納形式によるエルミート・マトリックスを使って、連立 1 次方程式を解く。

```
call chptrs ( uplo, n, nrhs, ap, ipiv, b, ldb, info )
call zhptrs ( uplo, n, nrhs, ap, ipiv, b, ldb, info )
```

説明

このルーチンは、エルミート・マトリックス A を持つ連立 1 次方程式 $AX = B$ を、A の Bunch-Kaufman 因子分解に基づいて、X について解く。

`uplo='U'` の場合、 $A = PUDU^HP^T$

`uplo='L'` の場合、 $A = PLDL^HP^T$

P は置換マトリックス、U と L は単位対角成分を含む**パックド形式**の上三角マトリックスと下三角マトリックス、D は対称ブロック対角マトリックスである。マトリックス B の各列に格納されている複数の右辺を使用して、この連立方程式を解く。

このルーチンには、(U または L を格納する) 配列 *ap* と、因子分解ルーチン ?hptrf によって返される形式の配列 *ipiv* を与える必要がある。

入力パラメータ

uplo `CHARACTER*1`。`'U'` または `'L'` でなければならない。入力マトリックス A の因子分解の方法を指定する。

uplo = `'U'` の場合は、配列 *ap* には、因子分解 $A = PUDU^HP^T$ の上三角係数 U を格納する。

uplo = `'L'` の場合は、配列 *ap* には、因子分解 $A = PLDL^HP^T$ の上三角係数 L を格納する。

n `INTEGER`。マトリックス A の次数 ($n \geq 0$)。

nrhs `INTEGER`。右辺の数 ($nrhs \geq 0$)。

ipiv `INTEGER`。配列 `DIMENSION` は少なくとも max(1, *n*) ?hptrf によって返される *ipiv* 配列。

ap, b	COMPLEX (chptrs の場合) DOUBLE COMPLEX (zhptrs の場合) 配列、*ap(*)*, *b(ldb, *)* *ap* の次元は、少なくとも max(1, *n*(*n*+1)/2) でなければならない。 配列 *ap* には、*uplo* の指定に従って、係数 *U* または *L* を**パックド格納形式**で格納する (「マトリックスの格納形式」を参照)。 配列 *b* には、マトリックス *B* を格納する。このマトリックスの列は、連立方程式の右辺である。*b* の第 2 次元は、少なくとも max(1, *nrhs*) でなければならない。
ldb	INTEGER。*b* の第 1 次元。*ldb* ≥ max(1, *n*)

出力パラメータ

b	解のマトリックス *X* によって上書きされる。
info	INTEGER。*info* = 0 の場合は、正常に終了する。 *info* = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

各右辺 *b* について算出された解は、摂動連立方程式 $(A + E)x = b$ の正確な解になる。

$$|E| \le c(n)\varepsilon\, P|U||D|\left|U^H\right|P^T \quad \text{または} \quad |E| \le c(n)\varepsilon\, P|L||D|\left|L^H\right|P^T$$

$c(n)$ は n の適度な 1 次関数で、ε はマシンの精度である。

x_0 が真の解である場合、算出された解 x は、次の誤差範囲を満たす。

$$\frac{\|x - x_0\|_\infty}{\|x\|_\infty} \le c(n)\ \mathrm{cond}(A, x)\varepsilon$$

ただし、$\mathrm{cond}(A, x) = \left\| |A^{-1}|\,|A|\,|x| \right\|_\infty / \|x\|_\infty \le \|A^{-1}\|_\infty \|A\|_\infty = \kappa_\infty(A)$

$\mathrm{cond}(A, x)$ は、$\kappa_\infty(A)$ よりはるかに小さくなる場合がある。

1 つの右辺ベクトルの浮動小数点演算の合計回数は、複素数型の場合は約 $8n^2$ になる。

条件数 $\kappa_\infty(A)$ を推定するには、?hpcon を呼び出す。
解の精度を改善して誤差を推定するには、?hprfs を呼び出す。

?trtrs

三角マトリックスを使って、複数の
右辺を持つ連立 1 次方程式を解く。

```
call strtrs (uplo, trans, diag, n, nrhs, a, lda, b, ldb, info)
call dtrtrs (uplo, trans, diag, n, nrhs, a, lda, b, ldb, info)
call ctrtrs (uplo, trans, diag, n, nrhs, a, lda, b, ldb, info)
call ztrtrs (uplo, trans, diag, n, nrhs, a, lda, b, ldb, info)
```

説明

このルーチンは、マトリックス B に格納された複数の右辺を使用して、三角マトリックス A を持つ以下の連立 1 次方程式を X について解く。

$AX = B$	*trans*='N' の場合
$A^TX = B$	*trans*='T' の場合
$A^HX = B$	*trans*='C' の場合 (ただし、複素マトリックスのみ)

入力パラメータ

uplo CHARACTER*1。'U' または 'L' でなければならない。
A が上三角マトリックスか、下三角マトリックスかを指定する。
uplo = 'U' の場合は、A は上三角マトリックスである。
uplo = 'L' の場合は、A は下三角マトリックスである。

trans CHARACTER*1。'N' または 'T' または 'C' でなければならない。
trans = 'N' の場合は、$AX = B$ を X について解く。
trans = 'T' の場合は、$A^TX = B$ を X について解く。
trans = 'C' の場合は、$A^HX = B$ を X について解く。

diag CHARACTER*1。'N' または 'U' でなければならない。
diag = 'N' の場合は、A は単位三角マトリックスではない。
diag = 'U' の場合は、A は単位三角マトリックスである。A の対角成分は 1 と見なされ、配列 *a* 内で参照されない。

n INTEGER。A の次数。B の行の数 ($n \geq 0$)。

nrhs INTEGER。右辺の数 ($nrhs \geq 0$)。

a, *b*	REAL (strtrs の場合) DOUBLE PRECISION (dtrtrs の場合) COMPLEX (ctrtrs の場合) DOUBLE COMPLEX (ztrtrs の場合) 配列 : *a*(*lda*, *), *b*(*ldb*, *) 配列 *a* には、マトリックス A を格納する。 配列 *b* には、マトリックス B を格納する。このマトリックスの列は、連立方程式の右辺である。 *a* の第 2 次元は、少なくとも max(1, *n*) でなければならない。*b* の第 2 次元は、少なくとも max(1, *nrhs*) でなければならない。
lda	INTEGER。*a* の第 1 次元。*lda* ≥ max(1, *n*)
ldb	INTEGER。*b* の第 1 次元。*ldb* ≥ max(1, *n*)

出力パラメータ

b	解のマトリックス X によって上書きされる。
info	INTEGER。*info* = 0 の場合は、正常に終了する。 *info* = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

各右辺 b について算出された解は、摂動連立方程式 $(A + E)x = b$ の正確な解になる。

$$|E| \le c(n)\varepsilon\,|A|$$

$c(n)$ は n の適度な 1 次関数で、ε はマシンの精度である。

x_0 が真の解である場合、算出された解 x は、次の誤差範囲を満たす。

$$\frac{\|x - x_0\|_\infty}{\|x\|_\infty} \le c(n)\,\mathrm{cond}(A, x)\varepsilon, \quad \text{provided } c(n)\,\mathrm{cond}(A, x)\varepsilon < 1$$

ただし、$\mathrm{cond}(A, x) = \left\| |A^{-1}|\,|A|\,|x| \right\|_\infty / \|x\|_\infty \le \|A^{-1}\|_\infty \|A\|_\infty = \kappa_\infty(A)$

$\mathrm{cond}(A, x)$ は、$\kappa_\infty(A)$ よりはるかに小さくなる場合がある。A^T と A^H の条件数は、$\kappa_\infty(A)$ に等しいことも、等しくないこともある。

1 つの右辺ベクトル b の浮動小数点演算のおおよその回数は、実数型の場合は n^2、複素数型の場合は $4n^2$ になる。

条件数 $\kappa_\infty(A)$ を推定するには、?trcon を呼び出す。
解の誤差を推定するには、?trrfs を呼び出す。

?tptrs

パックド形式の三角マトリックスを使って、複数の右辺を持つ連立 1 次方程式を解く。

```
call stptrs (uplo, trans, diag, n, nrhs, ap, b, ldb, info)
call dtptrs (uplo, trans, diag, n, nrhs, ap, b, ldb, info)
call ctptrs (uplo, trans, diag, n, nrhs, ap, b, ldb, info)
call ztptrs (uplo, trans, diag, n, nrhs, ap, b, ldb, info)
```

説明

このルーチンは、マトリックス B に格納された複数の右辺を使用して、パックド形式の三角マトリックス A を持つ以下の連立 1 次方程式を X について解く。

$AX = B$	*trans*='N' の場合
$A^TX = B$	*trans*='T' の場合
$A^HX = B$	*trans*='C' の場合 (ただし、複素マトリックスのみ)

入力パラメータ

uplo CHARACTER*1。'U' または 'L' でなければならない。

A が上三角マトリックスか、下三角マトリックスかを指定する。

uplo = 'U' の場合は、A は上三角マトリックスである。

uplo = 'L' の場合は、A は下三角マトリックスである。

trans CHARACTER*1。'N' または 'T' または 'C' でなければならない。

trans = 'N' の場合は、$AX = B$ を X について解く。

trans = 'T' の場合は、$A^TX = B$ を X について解く。

trans = 'C' の場合は、$A^HX = B$ を X について解く。

diag CHARACTER*1。'N' または 'U' でなければならない。

diag = 'N' の場合は、A は単位三角マトリックスではない。

diag = 'U' の場合は、A は単位三角マトリックスである。対角成分は 1 と見なされ、配列 *ap* 内で参照されない。

n INTEGER。A の次数。B の行の数 ($n \geq 0$)。

nrhs INTEGER。右辺の数 ($nrhs \geq 0$)。

ap, b	REAL (stptrs の場合) DOUBLE PRECISION (dtptrs の場合) COMPLEX (ctptrs の場合) DOUBLE COMPLEX (ztptrs の場合) 配列、*ap(*)*, *b(ldb, *)* *ap* の次元は、少なくとも max(1, *n* (*n*+1) / 2) でなければならない。 配列 *ap* にはマトリックス *A* を**パックド格納形式**で格納する。(「マトリックスの格納形式」を参照)。 配列 *b* には、マトリックス *B* を格納する。このマトリックスの列は、連立方程式の右辺である。*b* の第 2 次元は、少なくとも max(1, *nrhs*) でなければならない。
ldb	INTEGER。*b* の第 1 次元。*ldb* ≥ max(1, *n*)

出力パラメータ

b	解のマトリックス *X* によって上書きされる。
info	INTEGER。*info* = 0 の場合は、正常に終了する。 *info* = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

各右辺 *b* について算出された解は、摂動連立方程式 $(A + E)x = b$ の正確な解になる。

$$|E| \le c(n)\varepsilon\,|A|$$

$c(n)$ は *n* の適度な 1 次関数で、ε はマシンの精度である。

x_0 が真の解である場合、算出された解 *x* は、次の誤差範囲を満たす。

$$\frac{\|x - x_0\|_\infty}{\|x\|_\infty} \le c(n)\ \mathrm{cond}(A, x)\varepsilon, \quad \text{provided } c(n)\ \mathrm{cond}(A, x)\varepsilon < 1$$

ただし、$\mathrm{cond}(A, x) = \left\| |A^{-1}|\,|A|\,|x| \right\|_\infty / \|x\|_\infty \le \|A^{-1}\|_\infty \|A\|_\infty = \kappa_\infty(A)$

$\mathrm{cond}(A, x)$ は、$\kappa_\infty(A)$ よりはるかに小さくなる場合がある。A^T と A^H の条件数は、$\kappa_\infty(A)$ に等しいことも、等しくないこともある。

1 つの右辺ベクトル b の浮動小数点演算のおおよその回数は、実数型の場合は n^2、複素数型の場合は $4n^2$ になる。

条件数 $\kappa_\infty(A)$ を推定するには、?tpcon を呼び出す。
解の誤差を推定するには、?tprfs を呼び出す。

?tbtrs

バンド三角マトリックスを使って、複数の右辺を持つ連立 1 次方程式を解く。

```
call stbtrs (uplo, trans, diag, n, kd, nrhs, ab, ldab, b, ldb, info)
call dtbtrs (uplo, trans, diag, n, kd, nrhs, ab, ldab, b, ldb, info)
call ctbtrs (uplo, trans, diag, n, kd, nrhs, ab, ldab, b, ldb, info)
call ztbtrs (uplo, trans, diag, n, kd, nrhs, ab, ldab, b, ldb, info)
```

説明

このルーチンは、マトリックス B に格納された複数の右辺を使用して、バンド三角マトリックス A を持つ以下の連立 1 次方程式を X について解く。

$AX = B$	*trans*='N' の場合
$A^TX = B$	*trans*='T' の場合
$A^HX = B$	*trans*='C' の場合 (ただし、複素マトリックスのみ)

入力パラメータ

uplo CHARACTER*1。'U' または 'L' でなければならない。
A が上三角マトリックスか、下三角マトリックスかを指定する。
uplo = 'U' の場合は、A は上三角マトリックスである。
uplo = 'L' の場合は、A は下三角マトリックスである。

trans CHARACTER*1。'N' または 'T' または 'C' でなければならない。
trans = 'N' の場合は、$AX = B$ を X について解く。
trans = 'T' の場合は、$A^TX = B$ を X について解く。
trans = 'C' の場合は、$A^HX = B$ を X について解く。

diag CHARACTER*1。'N' または 'U' でなければならない。
diag = 'N' の場合は、A は単位三角マトリックスではない。
diag = 'U' の場合は、A は単位三角マトリックスである。対角成分は 1 と見なされ、配列 *ab* 内で参照されない。

n INTEGER。A の次数。B の行の数 ($n \geq 0$)。

kd INTEGER。A の優対角成分または劣対角成分の数 ($kd \geq 0$)。

nrhs INTEGER。右辺の数 ($nrhs \geq 0$)。

ab, b	REAL (stbtrs の場合) DOUBLE PRECISION (dtbtrs の場合) COMPLEX (ctbtrs 場合) DOUBLE COMPLEX (ztbtrs の場合) 配列 : *ab(ldab, *)*, *b(ldb, *)* 配列 *ab* には、マトリックス *A* を**バンド格納形式**で格納する 。 配列 *b* には、マトリックス *B* を格納する。このマトリックスの列は、連立方程式の右辺である。 *ab* の第 2 次元は、少なくとも max(1, *n*) でなければならない。 *b* の第 2 次元は、少なくとも max(1, *nrhs*) でなければならない。
ldab	INTEGER。*ab* の第 1 次元。*ldab* ≥ *kd* + 1
ldb	INTEGER。*b* の第 1 次元。*ldb* ≥ max(1, *n*)

出力パラメータ

b	解のマトリックス *X* によって上書きされる。
info	INTEGER。*info* = 0 の場合は、正常に終了する。 *info* = -*i*の場合は、*i*番目のパラメータの値が不正である。

アプリケーション・ノート

各右辺 *b* について算出された解は、摂動連立方程式 $(A + E)x = b$ の正確な解になる。

$$|E| \le c(n)\varepsilon\,|A|$$

$c(n)$ は *n* の適度な 1 次関数で、ε はマシンの精度である。
x_0 が真の解である場合、算出された解 *x* は、次の誤差範囲を満たす。

$$\frac{\|x - x_0\|_\infty}{\|x\|_\infty} \le c(n)\,\mathrm{cond}(A, x)\varepsilon, \quad \text{provided } c(n)\,\mathrm{cond}(A, x)\varepsilon < 1$$

ただし、$\mathrm{cond}(A, x) = \left\| |A^{-1}|\,|A|\,|x| \right\|_\infty / \|x\|_\infty \le \|A^{-1}\|_\infty \|A\|_\infty = \kappa_\infty(A)$

$\mathrm{cond}(A, x)$ は、$\kappa_\infty(A)$ よりはるかに小さくなる場合がある。A^T と A^H の条件数は、$\kappa_\infty(A)$ に等しいことも、等しくないこともある。

1 つの右辺ベクトル *b* の浮動小数点演算のおおよその回数は、実数型の場合は 2*n***kd*, 複素数型の場合は 8*n***kd* になる。

条件数 $\kappa_\infty(A)$ を推定するには、?tbcon を呼び出す。
解の誤差を推定するには、?tbrfs を呼び出す。

条件数を推定するためのルーチン

この節では、マトリックスの**条件数**を推定するための LAPACK ルーチンについて説明する。条件数は、連立 1 次方程式の解の誤差の分析に使用される (「誤差の分析」を参照)。マトリックスがほとんど特異である (ゼロに近い) 場合は、条件数が非常に大きくなる場合がある。このため、これらのルーチンでは、実際には条件数の**逆数**を計算する。

?gecon

一般マトリックスの条件数の逆数を、
1- ノルムまたは無限ノルムによって
推定する。

```
call sgecon ( norm, n, a, lda, anorm, rcond, work, iwork, info )
call dgecon ( norm, n, a, lda, anorm, rcond, work, iwork, info )
call cgecon ( norm, n, a, lda, anorm, rcond, work, rwork, info )
call zgecon ( norm, n, a, lda, anorm, rcond, work, rwork, info )
```

説明

このルーチンは、次のように、一般マトリックス A の条件数の逆数を、1- ノルムまたは無限ノルムによって推定する。

$$\kappa_1(A) = \|A\|_1 \, \|A^{-1}\|_1 = \kappa_\infty(A^T) = \kappa_\infty(A^H)$$
$$\kappa_\infty(A) = \|A\|_\infty \, \|A^{-1}\|_\infty = \kappa_1(A^T) = \kappa_1(A^H)$$

このルーチンを呼び出す前に、以下の手順を実行する必要がある。

- *anorm* を計算する (either $\|A\|_1 = \max_j \Sigma_i |a_{ij}|$ または $\|A\|_\infty = \max_i \Sigma_j |a_{ij}|$)。
- ?getrf を呼び出して、A の LU 因子分解を計算する。

入力パラメータ

norm	CHARACTER*1。'1' または 'O' または 'I' でなければならない。 *norm* = '1' または 'O' の場合は、ルーチンは $\kappa_1(A)$ を推定する。 *norm* = 'I' の場合は、ルーチンは $\kappa_\infty(A)$ を推定する。
n	INTEGER。マトリックス A の次数 ($n \geq 0$)
a, work	REAL (sgecon の場合) DOUBLE PRECISION (dgecon の場合) COMPLEX (cgecon の場合) DOUBLE COMPLEX (zgecon の場合)

配列 :*a*(*lda*, *), *work*(*)
配列 *a* には、?getrf によって返される、*LU* 因子分解されたマトリックス *A* を格納する。
a の第 2 次元は、少なくとも max(1, *n*) でなければならない。
配列 *work* は、このルーチン用のワークスペースである。
work の次元は、実数型の場合は少なくとも max(1, 4**n*) で、複素数型の場合は max(1, 2**n*) でなければならない。

anorm REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
元のマトリックス *A* のノルム (説明を参照)。

lda INTEGER。*a* の第 1 次元。*lda* ≥ max(1, *n*)

iwork INTEGER
ワークスペース配列 DIMENSION は少なくとも max(1, *n*)

rwork REAL (cgecon の場合)
DOUBLE PRECISION (zgecon の場合)
ワークスペース配列 DIMENSION は少なくとも max(1, 2*n)

出力パラメータ

rcond REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
条件数の逆数の推定値。推定値のアンダーフローが発生した場合は、*rcond* =0 に設定される。この場合は、マトリックスは有効な精度で特異になる。ただし、*rcond* が (有効な精度で) 1.0 より小さい場合は、条件の良くないマトリックスまたは特異なマトリックスである。

info INTEGER。*info* = 0 の場合は、正常に終了する。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

算出された *rcond* は、ρ (真の条件数の逆数) より小さくはならない。また、この値は、実際にはほぼ常に、10ρ より小さくなる。このルーチンを呼び出す場合は、連立 1 次方程式 $Ax = b$ または $A^H x = b$ を繰り返し解く必要がある。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約 $2n^2$ で、複素数型の場合は約 $8n^2$ である。

?gbcon

バンド・マトリックスの条件数の逆数を、
1- ノルムまたは無限ノルムによって
推定する。

```
call sgbcon (norm, n, kl, ku, ab, ldab, ipiv, anorm, rcond, work,
             iwork, info)
call dgbcon (norm, n, kl, ku, ab, ldab, ipiv, anorm, rcond, work,
             iwork, info)
call cgbcon (norm, n, kl, ku, ab, ldab, ipiv, anorm, rcond, work,
             rwork, info)
call zgbcon (norm, n, kl, ku, ab, ldab, ipiv, anorm, rcond, work,
             rwork, info)
```

説明

このルーチンは、次のように、一般バンド・マトリックス A の条件数の逆数を、1- ノルムまたは無限ノルムによって推定する。

$$\kappa_1(A) = \|A\|_1 \, \|A^{-1}\|_1 = \kappa_\infty(A^T) = \kappa_\infty(A^H)$$
$$\kappa_\infty(A) = \|A\|_\infty \, \|A^{-1}\|_\infty = \kappa_1(A^T) = \kappa_1(A^H)$$

このルーチンを呼び出す前に、以下の手順を実行する必要がある。

- *anorm* を計算する (either $\|A\|_1 = \max_j \Sigma_i |a_{ij}|$ または $\|A\|_\infty = \max_i \Sigma_j |a_{ij}|$)。
- ?gbtrf を呼び出して、A の LU 因子分解を計算する。

入力パラメータ

norm	CHARACTER*1。'1' または 'O' または 'I' でなければならない。 *norm* = '1' または 'O' の場合は、このルーチンは $\kappa_1(A)$ を推定する。 *norm* = 'I' の場合は、このルーチンは $\kappa_\infty(A)$ を推定する。
n	INTEGER。マトリックス A の次数 ($n \geq 0$)
kl	INTEGER。A のバンドの中の劣対角成分の数 ($kl \geq 0$)。
ku	INTEGER。A のバンドの中の優対角成分の数 ($ku \geq 0$)。
ldab	INTEGER。配列 *ab* の第 1 次元。($ldab \geq 2kl + ku + 1$)
ipiv	INTEGER。配列 DIMENSION は少なくとも max(1, *n*) ?gbtrf によって返される *ipiv* 配列。

ab, work
REAL (sgbcon の場合)
DOUBLE PRECISION (dgbcon の場合)
COMPLEX (cgbcon の場合)
DOUBLE COMPLEX (zgbcon の場合)
配列 : *ab(ldab, *)*, *work(*)*
配列 *ab* には、?gbtrf によって返される、因子分解後のバンド・マトリックス *A* を格納する。
ab の第 2 次元は、少なくとも max(1, *n*) でなければならない。
配列 *work* は、このルーチン用のワークスペースである。
work の次元は、実数型の場合は少なくとも max(1, 3**n*) で、複素数型の場合は max(1, 2**n*) でなければならない。

anorm
REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
元のマトリックス *A* のノルム (説明を参照)。

iwork
INTEGER
ワークスペース配列 DIMENSION は少なくとも max(1, *n*)

rwork
REAL (cgbcon/f07bue の場合)
DOUBLE PRECISION (zgbcon/f07buf の場合)
ワークスペース配列 DIMENSION は少なくとも max(1, 2**n*)

出力パラメータ

rcond
REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
条件数の逆数の推定値。推定値のアンダーフローが発生した場合は、*rcond* =0 に設定される。この場合は、マトリックスは有効な精度で特異になる。ただし、*rcond* が (有効な精度で) 1.0 より小さい場合は、条件の良くないマトリックスまたは特異なマトリックスである。

info
INTEGER。*info* = 0 の場合は、正常に終了する。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

算出された *rcond* は、ρ (真の条件数の逆数) より小さくはならない。また、この値は、実際にはほぼ常に、10ρ より小さくなる。このルーチンを呼び出す場合は、連立 1 次方程式 $Ax = b$ または $A^H x = b$ を繰り返し解く必要がある。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約 $2n(ku + 2kl)$ で、複素数型の場合は約 $8n(ku + 2kl)$ である。

?gtcon

?gttrf によって行われた因子分解を使用して、三重対角マトリックスの条件数の逆数を推定する。

```
call sgtcon ( norm, n, dl, d, du, du2, ipiv, anorm, rcond, work, iwork,
              info )
call dgtcon ( norm, n, dl, d, du, du2, ipiv, anorm, rcond, work, iwork,
              info )
call cgtcon ( norm, n, dl, d, du, du2, ipiv, anorm, rcond, work, info )
call zgtcon ( norm, n, dl, d, du, du2, ipiv, anorm, rcond, work, info )
```

説明

このルーチンは、実数または複素三重対角マトリックス A の (1- ノルムまたは無限ノルムの) 条件数の逆数を、次のように推定する。

$$\kappa_1(A) = \|A\|_1 \, \|A^{-1}\|_1$$
$$\kappa_\infty(A) = \|A\|_\infty \, \|A^{-1}\|_\infty$$

推定値は $\|A^{-1}\|$ について得られる。条件数の逆数は、*rcond* $= 1 / (\|A\| \, \|A^{-1}\|)$ として計算される。

このルーチンを呼び出す前に、以下の手順を実行する必要がある。

- *anorm* を計算する ($\|A\|_1 = \max_j \Sigma_i |a_{ij}|$ または $\|A\|_\infty = \max_i \Sigma_j |a_{ij}|$)。
- ?gttrf を呼び出して、A の LU 因子分解を実行する。

入力パラメータ

norm
CHARACTER*1。'1' または 'O' または 'I' でなければならない。
norm = '1' または 'O' の場合は、このルーチンは $\kappa_1(A)$ を推定する。
norm = 'I' の場合は、このルーチンは $\kappa_\infty(A)$ を推定する。

n
INTEGER。マトリックス *A* の次数 ($n \geq 0$)。

dl, d, du, du2
REAL (sgtcon の場合)
DOUBLE PRECISION (dgtcon の場合)
COMPLEX (cgtcon の場合)
DOUBLE COMPLEX (zgtcon の場合)
配列 : *dl*(*n* - 1), *d*(*n*), *du*(*n* - 1), *du2*(*n* - 2)
配列 *dl* には、?gttrf による *A* の *LU* 因子分解で得られたマトリックス *L* を定義する、(*n* - 1) 個の乗数が格納される。
配列 *d* には、*A* の *LU* 因子分解で得られた上三角マトリックス *U* の *n* 個の対角成分が格納される。
配列 *du* には、*U* の最初の優対角成分の (*n* - 1) 個の成分が格納される。
配列 *du2* には、*U* の 2 番目の優対角成分の (*n* - 2) 個の成分が格納される。

ipiv
INTEGER
配列 DIMENSION は (*n*)
?gttrf によって返される、ピボット (要) のインデックスの配列。

anorm
REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
元のマトリックス *A* のノルム (説明を参照)。

work
REAL (sgtcon の場合)
DOUBLE PRECISION (dgtcon の場合)
COMPLEX (cgtcon の場合)
DOUBLE COMPLEX (zgtcon の場合)
ワークスペース配列 DIMENSION は (2*n)

iwork
INTEGER
ワークスペース配列 DIMENSION は (*n*)
実数型でのみ使用される。

出力パラメータ

rcond REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
条件数の逆数の推定値。
推定値のアンダーフローが発生した場合は、*rcond* =0 に設定される。この場合は、マトリックスは有効な精度で特異になる。ただし、*rcond* が (有効な精度で) 1.0 より小さい場合は、条件の良くないマトリックスまたは特異なマトリックスである。

info INTEGER
info = 0 の場合は、ルーチンの実行は正常に終了した。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

計算された *rcond* は、ρ (真の条件数の逆数) より小さくはならない。この値は、実際にはほぼ常に、10ρ より小さくなる。このルーチンを呼び出すと、複数の連立 1 次方程式 $Ax = b$ の解が計算される。その数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は $2n^2$ 回、複素数型の場合は $8n^2$ 回である。

?pocon

正値対称 (エルミート) マトリックスの
条件数の逆数を推定する。

```
call spocon ( uplo, n, a, lda, anorm, rcond, work, iwork, info )
call dpocon ( uplo, n, a, lda, anorm, rcond, work, iwork, info )
call cpocon ( uplo, n, a, lda, anorm, rcond, work, rwork, info )
call zpocon ( uplo, n, a, lda, anorm, rcond, work, rwork, info )
```

説明

このルーチンは、次のように、正値対称 (エルミート) マトリックス A の条件数の逆数を推定する。

$\kappa_1(A) = \|A\|_1 \|A^{-1}\|_1$ ((A は対称またはエルミート・マトリックスであるため、$\kappa_\infty(A) = \kappa_1(A)$)

このルーチンを呼び出す前に、以下の手順を実行する必要がある。

- *anorm* を計算する (either $\|A\|_1 = \max_j \Sigma_i |a_{ij}|$ または $\|A\|_\infty = \max_i \Sigma_j |a_{ij}|$)。
- ?potrf を呼び出して、A のコルスキー因子分解を行う必要がある。

入力パラメータ

uplo CHARACTER*1。'U' または 'L' でなければならない。入力マトリックス A の因子分解の方法を指定する。

uplo = 'U' の場合は、配列 *a* には、因子分解 $A = U^H U$ の上三角係数 U を格納する。

uplo = 'L' の場合は、配列 *a* には、因子分解 $A = LL^H$ の下三角係数 L を格納する。

n INTEGER。マトリックス A の次数 ($n \geq 0$)

a, work REAL (spocon の場合)
DOUBLE PRECISION (dpocon の場合)
COMPLEX (cpocon の場合)
DOUBLE COMPLEX (zpocon の場合)
配列 :*a(lda, *)*, *work(*)*

配列 *a* には、?potrf によって返される、因子分解後のマトリックス *A* を格納する。
a の第 2 次元は、少なくとも max(1, *n*) でなければならない。
配列 *work* は、このルーチン用のワークスペースである。
work の次元は、実数型の場合は少なくとも max(1, 3**n*) で、複素数型の場合は max(1, 2**n*) でなければならない。

lda INTEGER。*a* の第 1 次元。*lda* ≥ max(1, *n*)

anorm REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
元のマトリックス *A* のノルム (**説明**を参照)。

iwork INTEGER
ワークスペース配列 DIMENSION は少なくとも max(1, *n*)

rwork REAL (cpocon の場合)
DOUBLE PRECISION (zpocon の場合)
ワークスペース配列 DIMENSION は少なくとも max(1, *n*)

出力パラメータ

rcond REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
条件数の逆数の推定値。推定値のアンダーフローが発生した場合は、*rcond* =0 に設定される。この場合は、マトリックスは有効な精度で特異になる。ただし、*rcond* が (有効な精度で) 1.0 より小さい場合は、条件の良くないマトリックスまたは特異なマトリックスである。

info INTEGER
info = 0 の場合は、正常に終了する。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

算出された *rcond* は、ρ (真の条件数の逆数) より小さくはならない。また、この値は、実際にはほぼ常に、10ρ より小さくなる。このルーチンを呼び出す場合は、連立 1 次方程式 $Ax = b$ を繰り返し解く必要がある。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約 $2n^2$ で、複素数型の場合は約 $8n^2$ である。

?ppcon

パックド格納形式の正値対称
(エルミート) マトリックスの条件数の
逆数を推定する。

```
call sppcon ( uplo, n, ap, anorm, rcond, work, iwork, info )
call dppcon ( uplo, n, ap, anorm, rcond, work, iwork, info )
call cppcon ( uplo, n, ap, anorm, rcond, work, rwork, info )
call zppcon ( uplo, n, ap, anorm, rcond, work, rwork, info )
```

説明

このルーチンは、次のように、パックド格納形式の正値対称 (エルミート) マトリックス A の条件数の逆数を推定する。

$\kappa_1(A) = \|A\|_1 \|A^{-1}\|_1$ (A は対称またはエルミート・マトリックスであるため、$\kappa_\infty(A) = \kappa_1(A)$)

このルーチンを呼び出す前に、以下の手順を実行する必要がある。

- *anorm* を計算する (either $\|A\|_1 = \max_j \Sigma_i |a_{ij}|$ または $\|A\|_\infty = \max_i \Sigma_j |a_{ij}|$)。
- ?pptrf を呼び出して、A のコルスキー因子分解を行う必要がある。

入力パラメータ

uplo	CHARACTER*1。'U' または 'L' でなければならない。入力マトリックス A の因子分解の方法を指定する。 *uplo* = 'U' の場合は、配列 *ap* には、因子分解 $A = U^H U$ の上三角係数 U を格納する。 *uplo* = 'L' の場合は、配列 *ap* には、因子分解 $A = LL^H$ の下三角係数 L を格納する。
n	INTEGER。マトリックス A の次数 ($n \geq 0$)
ap, work	REAL (sppcon の場合) DOUBLE PRECISION (dppcon の場合) COMPLEX (cppcon の場合) DOUBLE COMPLEX (zppcon の場合) 配列 : *ap(*)*, *work(*)*

配列 *ap* には、?pptrf によって返される、因子分解後のパックド形式のマトリックス A を格納する。
ap の次元は、少なくとも max(1, *n*(*n*+1)/2) でなければならない。
配列 *work* は、このルーチン用のワークスペースである。
work の次元は、実数型の場合は少なくとも max(1, 3**n*) で、複素数型の場合は max(1, 2**n*) でなければならない。

anorm REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
元のマトリックス *A* のノルム (**説明**を参照)。

iwork INTEGER
ワークスペース配列 DIMENSION は少なくとも max(1, *n*)

rwork REAL (cppcon の場合)
DOUBLE PRECISION (zppcon の場合)
ワークスペース配列 DIMENSION は少なくとも max(1, *n*)

出力パラメータ

rcond REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
条件数の逆数の推定値。推定値のアンダーフローが発生した場合は、*rcond* =0 に設定される。この場合は、マトリックスは有効な精度で特異になる。ただし、*rcond* が (有効な精度で) 1.0 より小さい場合は、条件の良くないマトリックスまたは特異なマトリックスである。

info INTEGER
info = 0 の場合は、正常に終了する。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

算出された *rcond* は、ρ (真の条件数の逆数) より小さくはならない。また、この値は、実際にはほぼ常に、10ρ より小さくなる。このルーチンを呼び出す場合は、連立 1 次方程式 $Ax = b$ を繰り返し解く必要がある。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約 $2n^2$ で、複素数型の場合は約 $8n^2$ である。

?pbcon

正値対称 (エルミート) バンド・
マトリックスの条件数の逆数を推定する。

```
call spbcon (uplo, n, kd, ab, ldab, anorm, rcond, work, iwork, info)
call dpbcon (uplo, n, kd, ab, ldab, anorm, rcond, work, iwork, info)
call cpbcon (uplo, n, kd, ab, ldab, anorm, rcond, work, rwork, info)
call zpbcon (uplo, n, kd, ab, ldab, anorm, rcond, work, rwork, info)
```

説明

このルーチンは、次のように、正値対称 (エルミート) バンド・マトリックス A の条件数の逆数を推定する。

$\kappa_1(A) = \|A\|_1 \|A^{-1}\|_1$ (A は対称またはエルミート・マトリックスであるため、$\kappa_\infty(A) = \kappa_1(A)$)

このルーチンを呼び出す前に、以下の手順を実行する必要がある。

- *anorm* を計算する (either $\|A\|_1 = \max_j \Sigma_i |a_{ij}|$ または $\|A\|_\infty = \max_i \Sigma_j |a_{ij}|$)。
- ?pbtrf を呼び出して、A のコルスキー因子分解を行う必要がある。

入力パラメータ

uplo	CHARACTER*1。'U' または 'L' でなければならない。入力マトリックス A の因子分解の方法を指定する。 *uplo* = 'U' の場合は、配列 *ab* には、コルスキー因子分解 $A = U^H U$ の上三角係数 U を格納する。 *uplo* = 'L' の場合は、配列 *ab* には、因子分解 $A = LL^H$ の下三角係数 L を格納する。
n	INTEGER。マトリックス A の次数 ($n \geq 0$)
kd	INTEGER。A の優対角成分または劣対角成分の数 ($kd \geq 0$)
ldab	INTEGER。配列 *ab* の第 1 次元。($ldab \geq kd + 1$)
ab, work	REAL (spbcon の場合) DOUBLE PRECISION (dpbcon の場合) COMPLEX (cpbcon の場合) DOUBLE COMPLEX (zpbcon の場合)

配列 : *ab*(*ldab*, **)*, *work*(**)*

配列 *ab* には、?pbtrf によって返される、因子分解後のバンド形式のマトリックス *A* を格納する。
ab の第 2 次元は、少なくとも max(1, *n*) でなければならない。
配列 *work* は、このルーチン用のワークスペースである。
work の次元は、実数型の場合は少なくとも max(1, 3**n*) で、複素数型の場合は max(1, 2**n*) でなければならない。

anorm REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
元のマトリックス *A* のノルム (**説明**を参照)。

iwork INTEGER
ワークスペース配列 DIMENSION は少なくとも max(1, *n*)

rwork REAL (cpbcon の場合)
DOUBLE PRECISION (zpbcon の場合)
ワークスペース配列 DIMENSION は少なくとも max(1, *n*)

出力パラメータ

rcond REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
条件数の逆数の推定値。推定値のアンダーフローが発生した場合は、*rcond* =0 に設定される。この場合は、マトリックスは有効な精度で特異になる。ただし、*rcond* が (有効な精度で) 1.0 より小さい場合は、条件の良くないマトリックスまたは特異なマトリックスである。

info INTEGER。*info* = 0 の場合は、正常に終了する。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

算出された *rcond* は、ρ (真の条件数の逆数) より小さくはならない。また、この値は、実際にはほぼ常に、10ρ より小さくなる。このルーチンを呼び出す場合は、連立 1 次方程式 $Ax = b$ を繰り返し解く必要がある。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約 $4n(kd + 1)$ で、複素数型の場合は約 $16n(kd + 1)$ である。

?ptcon

正値の対称 (エルミート) 三重対角
マトリックスの条件数の逆数を推定する。

```
call sptcon (n, d, e, anorm, rcond, work, info)
call dptcon (n, d, e, anorm, rcond, work, info)
call cptcon (n, d, e, anorm, rcond, work, info)
call zptcon (n, d, e, anorm, rcond, work, info)
```

説明

このルーチンは、?pttrf による因子分解 $A = LDL^H$ または $A = U^HDU$ を使用して、正値実数対称または複素エルミート三重対角マトリックスの (1- ノルムの) 条件数の逆数を、次のように計算する。

$\kappa_1(A) = \|A\|_1 \|A^{-1}\|_1$ (A は対称またはエルミート・マトリックスであるため、$\kappa_\infty(A) = \kappa_1(A)$)

ノルム $\|A^{-1}\|$ は直接法で計算される。条件数の逆数は、
rcond $= 1 / (\|A\| \|A^{-1}\|)$ として計算される。

このルーチンを呼び出す前に、以下の手順を実行する必要がある。

- *anorm* を $\|A\|_1 = \max_j \Sigma_i |a_{ij}|$ として計算する。
- ?pttrf を呼び出して、A の因子分解を実行する。

入力パラメータ

n	INTEGER。マトリックス A の次数 ($n \geq 0$)。
d, work	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列、次元は (*n*) 配列 *d* には、?pttrf による A の因子分解で得られた対角マトリックス D の *n* 個の対角成分が格納される。 *work* はワークスペース配列である。

e
REAL (sptcon の場合)
DOUBLE PRECISION (dptcon の場合)
COMPLEX (cptcon の場合)
DOUBLE COMPLEX (zptcon の場合)
配列 DIMENSION は (*n* - 1)
?pttrf による因子分解で得られた単位二重対角係数 *U* または *L* の対角線外の成分を格納する。

anorm
REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
元のマトリックス *A* の 1- ノルム (**説明**を参照)。

出力パラメータ

rcond
REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
条件数の逆数の推定値。推定値のアンダーフローが発生した場合は、*rcond* =0 に設定される。この場合は、マトリックスは有効な精度で特異になる。
ただし、*rcond* が (有効な精度で) 1.0 より小さい場合は、条件の良くないマトリックスまたは特異なマトリックスである。

info
INTEGER
info = 0 の場合は、ルーチンの実行は正常に終了した。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

計算された *rcond* は、ρ (真の条件数の逆数) より小さくはならない。この値は、実際にはほぼ常に、10ρ より小さくなる。このルーチンを呼び出すと、複数の連立 1 次方程式 $Ax = b$ の解が計算される。その数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は $4n(kd + 1)$ 回、複素数型の場合は $16n(kd + 1)$ 回である。

?sycon

対称マトリックスの条件数の逆数を
推定する。

```
call ssycon (uplo, n, a, lda, ipiv, anorm, rcond, work, iwork, info)
call dsycon (uplo, n, a, lda, ipiv, anorm, rcond, work, iwork, info)
call csycon (uplo, n, a, lda, ipiv, anorm, rcond, work, rwork, info)
call zsycon (uplo, n, a, lda, ipiv, anorm, rcond, work, rwork, info)
```

説明

このルーチンは、次のように、対称マトリックス A の条件数の逆数を推定する。

$\kappa_1(A) = \|A\|_1 \|A^{-1}\|_1$ (A は対称マトリックスであるため、$\kappa_\infty(A) = \kappa_1(A)$)

このルーチンを呼び出す前に、以下の手順を実行する必要がある。

- *anorm* を計算する (either $\|A\|_1 = \max_j \Sigma_i |a_{ij}|$ または $\|A\|_\infty = \max_i \Sigma_j |a_{ij}|$)。
- ?sytrf を呼び出して、A の因子分解を計算する。

入力パラメータ

uplo
: CHARACTER*1。'U' または 'L' でなければならない。入力マトリックス A の因子分解の方法を指定する。

  uplo = 'U' の場合は、配列 *a* には、因子分解 $A = PUDU^TP^T$ の上三角係数 U を格納する。

  uplo = 'L' の場合は、配列 *a* には、因子分解 $A = PLDL^TP^T$ の下三角係数 L を格納する。

n
: INTEGER。マトリックス A の次数 ($n \geq 0$)。

a, work
: REAL (ssycon の場合)
  DOUBLE PRECISION (dsycon の場合)
  COMPLEX (csycon の場合)
  DOUBLE COMPLEX (zsycon の場合)
  配列 :*a(lda, *)*, *work(*)*

  配列 *a* には、?sytrf によって返される、因子分解後のマトリックス A を格納する。
  a の第 2 次元は、少なくとも max(1, *n*) でなければならない。

配列 *work* は、このルーチン用のワークスペースである。
work の次元は、少なくとも max(1, 2*n) でなければならない。

lda INTEGER。*a* の第 1 次元。*lda* ≥ max(1, n)

ipiv INTEGER。配列 DIMENSION は少なくとも max(1, n)
?sytrf によって返される配列 *ipiv*

anorm REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
元のマトリックス A のノルム (**説明**を参照)。

iwork INTEGER
ワークスペース配列 DIMENSION は少なくとも max(1, n)

rwork REAL (csycon の場合)
DOUBLE PRECISION (zsycon の場合)
ワークスペース配列 DIMENSION は少なくとも max(1, n)

出力パラメータ

rcond REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
条件数の逆数の推定値。推定値のアンダーフローが発生した場合は、*rcond* =0 に設定される。この場合は、マトリックスは有効な精度で特異になる。ただし、*rcond* が (有効な精度で) 1.0 より小さい場合は、条件の良くないマトリックスまたは特異なマトリックスである。

info INTEGER
info = 0 の場合は、正常に終了する。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

算出された *rcond* は、ρ (真の条件数の逆数) より小さくはならない。また、この値は、実際にはほぼ常に、10ρ より小さくなる。このルーチンを呼び出す場合は、連立 1 次方程式 $Ax = b$ を繰り返し解く必要がある。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約 $2n^2$ で、複素数型の場合は約 $8n^2$ である。

?hecon

エルミート・マトリックスの条件数の
逆数を計算する。

```
call checon (uplo, n, a, lda, ipiv, anorm, rcond, work, rwork, info)
call zhecon (uplo, n, a, lda, ipiv, anorm, rcond, work, rwork, info)
```

説明

このルーチンは、次のように、エルミート・マトリックス A の条件数の逆数を計算する。

$\kappa_1(A) = \|A\|_1 \|A^{-1}\|_1$ (A はエルミート・マトリックスのため、$\kappa_\infty(A) = \kappa_1(A)$)

このルーチンを呼び出す前に、以下の手順を実行する必要がある。

- *anorm* を計算する (either $\|A\|_1 = \max_j \Sigma_i |a_{ij}|$ または $\|A\|_\infty = \max_i \Sigma_j |a_{ij}|$)。
- ?hetrf を呼び出して、A の因子分解を計算する。

入力パラメータ

uplo
: CHARACTER*1。'U' または 'L' でなければならない。入力マトリックス A の因子分解の方法を指定する。

  uplo = 'U' の場合は、配列 *a* には、因子分解 $A = PUDU^HP^T$ の上三角係数 U を格納する。

  uplo = 'L' の場合は、配列 *a* には、因子分解 $A = PLDL^HP^T$ の下三角係数 L を格納する。

n
: INTEGER。マトリックス A の次数 ($n \geq 0$)。

a, work
: COMPLEX (checon の場合)
  DOUBLE COMPLEX (zhecon の場合)
  配列 :*a(lda, *)*, *work(*)*

  配列 *a* には、?hetrf によって返される、因子分解後のマトリックス A を格納する。
  a の第 2 次元は、少なくとも max(1, *n*) でなければならない。

  配列 *work* は、このルーチン用のワークスペースである。

  work の次元は、少なくとも max(1, 2*n) でなければならない。

lda INTEGER。*a* の第 1 次元。*lda* ≥ max(1, *n*)

ipiv INTEGER。配列 DIMENSION は少なくとも max(1, *n*)
?hetrf によって返される配列 *ipiv*

anorm REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
元のマトリックス *A* のノルム (**説明**を参照)。

rwork REAL (checon の場合)
DOUBLE PRECISION (zhecon の場合)
ワークスペース配列 DIMENSION は少なくとも max(1, *n*)

出力パラメータ

rcond REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
条件数の逆数の推定値。推定値のアンダーフローが発生した場合は、*rcond* =0 に設定される。この場合は、マトリックスは有効な精度で特異になる。ただし、*rcond* が (有効な精度で) 1.0 より小さい場合は、条件の良くないマトリックスまたは特異なマトリックスである。

info INTEGER
info = 0 の場合は、正常に終了する。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

算出された *rcond* は、ρ (真の条件数の逆数) より小さくはならない。また、この値は、実際にはほぼ常に、10ρ より小さくなる。このルーチンを呼び出す場合は、連立 1 次方程式 $Ax = b$ を繰り返し解く必要がある。その回数は、通常は 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、約 $8n^2$ である。

?spcon

パックド格納形式の対称マトリックスの
条件数の逆数を推定する。

```
call sspcon ( uplo, n, ap, ipiv, anorm, rcond, work, iwork, info )
call dspcon ( uplo, n, ap, ipiv, anorm, rcond, work, iwork, info )
call cspcon ( uplo, n, ap, ipiv, anorm, rcond, work, rwork, info )
call zspcon ( uplo, n, ap, ipiv, anorm, rcond, work, rwork, info )
```

説明

このルーチンは、次のように、パックド格納形式の対称マトリックス A の条件数の逆数を推定する。

$\kappa_1(A) = \|A\|_1 \|A^{-1}\|_1$ (A は対称マトリックスであるため、$\kappa_\infty(A) = \kappa_1(A)$)

このルーチンを呼び出す前に、以下の手順を実行する必要がある。

- *anorm* を計算する (either $\|A\|_1 = \max_j \Sigma_i |a_{ij}|$ または $\|A\|_\infty = \max_i \Sigma_j |a_{ij}|$)。
- ?sptrf を呼び出して、A の因子分解を計算する。

入力パラメータ

uplo
: CHARACTER*1。'U' または 'L' でなければならない。入力マトリックス A の因子分解の方法を指定する。
uplo = 'U' の場合は、配列 *ap* には、因子分解 $A = PUDU^TP^T$ のパックド形式の上三角係数 U を格納する。
uplo = 'L' の場合は、配列 *ap* には、因子分解 $A = PLDL^TP^T$ のパックド形式の下三角係数 L を格納する。

n
: INTEGER。マトリックス A の次数 ($n \geq 0$)。

ap, work
: REAL (sspcon の場合)
DOUBLE PRECISION (dspcon の場合)
COMPLEX (cspcon の場合)
DOUBLE COMPLEX (zspcon の場合)
配列 : *ap*(*), *work*(*)
配列 *ap* には、?sptrf によって返される、因子分解後のパックド形式のマトリックス A を格納する。
ap の次元は、少なくとも max(1, $n(n+1)/2$) でなければならない。

配列 *work* は、このルーチン用のワークスペースである。
work の次元は、少なくとも max(1, 2*n) でなければならない。

ipiv INTEGER。配列 DIMENSION は少なくとも max(1, n)
?sptrf によって返される *ipiv* 配列。

anorm REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
元のマトリックス A のノルム (**説明**を参照)。

iwork INTEGER
ワークスペース配列 DIMENSION は少なくとも max(1, n)

rwork REAL (cspcon の場合)
DOUBLE PRECISION (zspcon の場合)
ワークスペース配列 DIMENSION は少なくとも max(1, n)

出力パラメータ

rcond REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
条件数の逆数の推定値。推定値のアンダーフローが発生した場合は、*rcond* =0 に設定される。この場合は、マトリックスは有効な精度で特異になる。ただし、*rcond* が (有効な精度で) 1.0 より小さい場合は、条件の良くないマトリックスまたは特異なマトリックスである。

info INTEGER
info = 0 の場合は、正常に終了する。
info = -*i*の場合は、*i*番目のパラメータの値が不正である。

アプリケーション・ノート

算出された *rcond* は、ρ (真の条件数の逆数) より小さくはならない。また、この値は、実際にはほぼ常に、10ρ より小さくなる。このルーチンを呼び出す場合は、連立 1 次方程式 $Ax = b$ を繰り返し解く必要がある。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約 $2n^2$ で、複素数型の場合は約 $8n^2$ である。

?hpcon

パックド格納形式のエルミート・
マトリックスの条件数の逆数を推定する。

```
call chpcon ( uplo, n, ap, ipiv, anorm, rcond, work, rwork, info )
call zhpcon ( uplo, n, ap, ipiv, anorm, rcond, work, rwork, info )
```

説明

このルーチンは、次のように、エルミート・マトリックス A の条件数の逆数を推定する。

$\kappa_1(A) = \|A\|_1 \|A^{-1}\|_1$ (A はエルミート・マトリックスのため、$\kappa_\infty(A) = \kappa_1(A)$)

このルーチンを呼び出す前に、以下の手順を実行する必要がある。

- *anorm* を計算する (either $\|A\|_1 = \max_j \Sigma_i |a_{ij}|$ または $\|A\|_\infty = \max_i \Sigma_j |a_{ij}|$)。
- ?hptrf を呼び出して、A の因子分解を計算する。

入力パラメータ

uplo
: CHARACTER*1。'U' または 'L' でなければならない。入力マトリックス A の因子分解の方法を指定する。

  uplo = 'U' の場合は、配列 *ap* には、因子分解 $A = PUDU^TP^T$ のパックド形式の上三角係数 U を格納する。

  uplo = 'L' の場合は、配列 *ap* には、因子分解 $A = PLDL^TP^T$ のパックド形式の下三角係数 L を格納する。

n
: INTEGER。マトリックス A の次数 ($n \geq 0$)。

ap, work
: COMPLEX (chpcon の場合)
  DOUBLE COMPLEX (zhpcon の場合)
  配列 : *ap*(*), *work*(*)

  配列 *ap* には、?hptrf によって返される、因子分解後のパックド形式のマトリックス A を格納する。
  ap の次元は、少なくとも max(1, *n* (*n*+1) / 2) でなければならない。

  配列 *work* は、このルーチン用のワークスペースである。
  work の次元は、少なくとも max(1, 2*n) でなければならない。

ipiv INTEGER。配列 DIMENSION は少なくとも max(1, *n*)
?hptrf によって返される *ipiv* 配列。

anorm REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
元のマトリックス *A* のノルム (**説明**を参照)。

rwork REAL (chpcon の場合)
DOUBLE PRECISION (zhpcon の場合)
ワークスペース配列 DIMENSION は少なくとも max(1, *n*)

出力パラメータ

rcond REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
条件数の逆数の推定値。推定値のアンダーフローが発生した場合は、*rcond* =0 に設定される。この場合は、マトリックスは有効な精度で特異になる。ただし、*rcond* が (有効な精度で) 1.0 より小さい場合は、条件の良くないマトリックスまたは特異なマトリックスである。

info INTEGER
info = 0 の場合は、正常に終了する。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

算出された *rcond* は、ρ (真の条件数の逆数) より小さくはならない。また、この値は、実際にはほぼ常に、10ρ より小さくなる。このルーチンを呼び出す場合は、連立 1 次方程式 $Ax = b$ を繰り返し解く必要がある。その回数は、通常は 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、約 $8n^2$ である。

?trcon

三角マトリックスの条件数の逆数を
推定する。

```
call strcon (norm, uplo, diag, n, a, lda, rcond, work, iwork, info)
call dtrcon (norm, uplo, diag, n, a, lda, rcond, work, iwork, info)
call ctrcon (norm, uplo, diag, n, a, lda, rcond, work, rwork, info)
call ztrcon (norm, uplo, diag, n, a, lda, rcond, work, rwork, info)
```

説明

このルーチンは、次のように、三角マトリックス A の条件数の逆数を、1- ノルムまたは無限ノルムによって推定する。

$$\kappa_1(A) = \|A\|_1 \, \|A^{-1}\|_1 = \kappa_\infty(A^T) = \kappa_\infty(A^H)$$
$$\kappa_\infty(A) = \|A\|_\infty \, \|A^{-1}\|_\infty = \kappa_1(A^T) = \kappa_1(A^H)$$

入力パラメータ

norm CHARACTER*1。'1' または 'O' または 'I' でなければならない。

norm = '1' または 'O' の場合は、このルーチンは $\kappa_1(A)$ を推定する。

norm = 'I' の場合は、このルーチンは $\kappa_\infty(A)$ を推定する。

uplo CHARACTER*1。'U' または 'L' でなければならない。A が上三角マトリックスか、下三角マトリックスかを指定する。

uplo = 'U' の場合は、配列 *a* には、A の上三角部分を格納する。その他の配列成分は参照されない。

uplo = 'L' の場合は、配列 *a* には、A の下三角部分を格納する。その他の配列成分は参照されない。

diag CHARACTER*1。'N' または 'U' でなければならない。

diag = 'N' の場合は、A は単位三角マトリックスではない。

diag = 'U' の場合は、A は単位三角マトリックスである。対角成分は 1 と見なされ、配列 *a* 内で参照されない。

n INTEGER。マトリックス A の次数 ($n \geq 0$)

a, work
REAL (strcon の場合)
DOUBLE PRECISION (dtrcon の場合)
COMPLEX (ctrcon の場合)
DOUBLE COMPLEX (ztrcon の場合)
配列 :*a(lda, *)*, *work(*)*
配列 *a* には、マトリックス *A* を格納する。
a の第 2 次元は、少なくとも max(1, *n*) でなければならない。
配列 *work* は、このルーチン用のワークスペースである。
work の次元は、実数型の場合は少なくとも max(1, 3**n*) で、複素数型の場合は max(1, 2**n*) でなければならない。

lda
INTEGER。*a* の第 1 次元。*lda* ≥ max(1, *n*)

iwork
INTEGER
ワークスペース配列 DIMENSION は少なくとも max(1, *n*)

rwork
REAL (ctrcon の場合)
DOUBLE PRECISION (ztrcon の場合)
ワークスペース配列 DIMENSION は少なくとも max(1, *n*)

出力パラメータ

rcond
REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
条件数の逆数の推定値。推定値のアンダーフローが発生した場合は、*rcond* =0 に設定される。この場合は、マトリックスは有効な精度で特異になる。ただし、*rcond* が (有効な精度で) 1.0 より小さい場合は、条件の良くないマトリックスまたは特異なマトリックスである。

info
INTEGER
info = 0 の場合は、正常に終了する。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

算出された *rcond* は、ρ (真の条件数の逆数) より小さくはならない。また、この値は、実際にはほぼ常に、10ρ より小さくなる。このルーチンを呼び出す場合は、連立 1 次方程式 $Ax = b$ を繰り返し解く必要がある。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約 n^2 で、複素数型の場合は約 $4n^2$ である。

?tpcon

パックド格納形式の三角マトリックスの
条件数の逆数を推定する。

```
call stpcon (norm, uplo, diag, n, ap, rcond, work, iwork, info)
call dtpcon (norm, uplo, diag, n, ap, rcond, work, iwork, info)
call ctpcon (norm, uplo, diag, n, ap, rcond, work, rwork, info)
call ztpcon (norm, uplo, diag, n, ap, rcond, work, rwork, info)
```

説明

このルーチンは、次のように、パックド格納形式の三角マトリックス A の条件数の逆数を、1- ノルムまたは無限ノルムによって推定する。

$$\kappa_1(A) = \|A\|_1 \, \|A^{-1}\|_1 = \kappa_\infty(A^T) = \kappa_\infty(A^H)$$
$$\kappa_\infty(A) = \|A\|_\infty \, \|A^{-1}\|_\infty = \kappa_1(A^T) = \kappa_1(A^H)$$

入力パラメータ

norm CHARACTER*1。'1' または 'O' または 'I' でなければならない。

norm = '1' または 'O' の場合は、このルーチンは $\kappa_1(A)$ を推定する。

norm = 'I' の場合は、このルーチンは $\kappa_\infty(A)$ を推定する。

uplo CHARACTER*1。'U' または 'L' でなければならない。*A* が上三角マトリックスか、下三角マトリックスかを指定する。

uplo = 'U' の場合は、配列 *ap* には、*A* の上三角部分をパックド形式で格納する。

uplo = 'L' の場合は、配列 *ap* には、*A* の下三角部分をパックド形式で格納する。

diag CHARACTER*1。'N' または 'U' でなければならない。

diag = 'N' の場合は、*A* は単位三角マトリックスではない。

diag = 'U' の場合は、*A* は単位三角マトリックスである。対角成分は 1 と見なされ、配列 *ap* 内で参照されない。

n INTEGER。マトリックス *A* の次数 ($n \geq 0$)

ap, work
REAL (stpcon の場合)
DOUBLE PRECISION (dtpcon の場合)
COMPLEX (ctpcon の場合)
DOUBLE COMPLEX (ztpcon の場合)
配列 : *ap*(*), *work*(*)
配列 *ap* には、パックド形式のマトリックス *A* を格納する。
ap の次元は、少なくとも max(1, *n*(*n*+1)/2) でなければならない。
配列 *work* は、このルーチン用のワークスペースである。
work の次元は、実数型の場合は少なくとも max(1, 3*n*) で、複素数型の場合は max(1, 2*n*) でなければならない。

iwork
INTEGER
ワークスペース配列 DIMENSION は少なくとも max(1, *n*)

rwork
REAL (ctpcon の場合)
DOUBLE PRECISION (ztpcon の場合)
ワークスペース配列 DIMENSION は少なくとも max(1, *n*)

出力パラメータ

rcond
REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
条件数の逆数の推定値。推定値のアンダーフローが発生した場合は、*rcond* =0 に設定される。この場合は、マトリックスは有効な精度で特異になる。ただし、*rcond* が (有効な精度で) 1.0 より小さい場合は、条件の良くないマトリックスまたは特異なマトリックスである。

info
INTEGER
info = 0 の場合は、正常に終了する。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

算出された *rcond* は、ρ (真の条件数の逆数) より小さくはならない。また、この値は、実際にはほぼ常に、10ρ より小さくなる。このルーチンを呼び出す場合は、連立 1 次方程式 $Ax = b$ を繰り返し解く必要がある。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約 n^2 で、複素数型の場合は約 $4n^2$ である。

?tbcon

三角バンド・マトリックスの条件数の
逆数を推定する。

```
call stbcon ( norm, uplo, diag, n, kd, ab, ldab, rcond, work, iwork, info )
call dtbcon ( norm, uplo, diag, n, kd, ab, ldab, rcond, work, iwork, info )
call ctbcon ( norm, uplo, diag, n, kd, ab, ldab, rcond, work, rwork, info )
call ztbcon ( norm, uplo, diag, n, kd, ab, ldab, rcond, work, rwork, info )
```

説明

このルーチンは、次のように、三角バンド・マトリックス A の条件数の逆数を、1- ノルムまたは無限ノルムによって推定する。

$$\kappa_1(A) = \|A\|_1 \, \|A^{-1}\|_1 = \kappa_\infty(A^T) = \kappa_\infty(A^H)$$
$$\kappa_\infty(A) = \|A\|_\infty \, \|A^{-1}\|_\infty = \kappa_1(A^T) = \kappa_1(A^H)$$

入力パラメータ

norm CHARACTER*1。'1' または 'O' または 'I' でなければならない。

norm = '1' または 'O' の場合は、このルーチンは $\kappa_1(A)$ を推定する。

norm = 'I' の場合は、このルーチンは $\kappa_\infty(A)$ を推定する。

uplo CHARACTER*1。'U' または 'L' でなければならない。A が上三角マトリックスか、下三角マトリックスかを指定する。

uplo = 'U' の場合は、配列 *ap* には、A の上三角部分をパックド形式で格納する。

uplo = 'L' の場合は、配列 *ab* には、A の下三角部分をパックド形式で格納する。

diag CHARACTER*1。'N' または 'U' でなければならない。

diag = 'N' の場合は、A は単位三角マトリックスではない。

diag = 'U' の場合は、A は単位三角マトリックスである。対角成分は 1 と見なされ、配列 *ab* 内で参照されない。

n INTEGER。マトリックス A の次数 ($n \geq 0$)

kd INTEGER。A の優対角成分または劣対角成分の数 ($kd \geq 0$)

ab, work
REAL (stbcon の場合)
DOUBLE PRECISION (dtbcon の場合)
COMPLEX (ctbcon の場合)
DOUBLE COMPLEX (ztbcon の場合)
配列 : *ab(ldab, *)*, *work(*)*
配列 *ab* には、バンド・マトリックス *A* を格納する。*ab* の第 2 次元は、少なくとも max(1, *n*) でなければならない。
配列 *work* は、このルーチン用のワークスペースである。
work の次元は、実数型の場合は少なくとも max(1, 3**n*) で、複素数型の場合は max(1, 2**n*) でなければならない。

ldab
INTEGER。配列 *ab* の第 1 次元。(*ldab* ≥ *kd* +1)

iwork
INTEGER
ワークスペース配列 DIMENSION は少なくとも max(1, *n*)

rwork
REAL (ctbcon の場合)
DOUBLE PRECISION (ztbcon の場合)
ワークスペース配列 DIMENSION は少なくとも max(1, *n*)

出力パラメータ

rcond
REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
条件数の逆数の推定値。推定値のアンダーフローが発生した場合は、*rcond* =0 に設定される。この場合は、マトリックスは有効な精度で特異になる。ただし、*rcond* が (有効な精度で) 1.0 より小さい場合は、条件の良くないマトリックスまたは特異なマトリックスである。

info
INTEGER。*info* = 0 の場合は、正常に終了する。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

算出された *rcond* は、ρ (真の条件数の逆数) より小さくはならない。また、この値は、実際にはほぼ常に、10ρ より小さくなる。このルーチンを呼び出す場合は、連立 1 次方程式 $Ax = b$ を繰り返し解く必要がある。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約 $2n(kd + 1)$ で、複素数型の場合は約 $8n(kd + 1)$ である。

解の精度の改善と誤差の推定

この節では、算出された連立 1 次方程式の解の精度を改善し、解の誤差を推定するための LAPACK ルーチンについて説明する。これらのルーチンを呼び出す前に、連立方程式のマトリックスを因子分解し、解を計算する必要がある (「計算ルーチン」および「計算ルーチン」を参照)。

?gerfs

一般マトリックスによる連立 1 次方程式の
解の精度を改善し、解の誤差を推定する。

```
call sgerfs (trans, n, nrhs, a, lda, af, ldaf, ipiv, b, ldb, x, ldx,
             ferr, berr, work, iwork, info)
call dgerfs (trans, n, nrhs, a, lda, af, ldaf, ipiv, b, ldb, x, ldx,
             ferr, berr, work, iwork, info)

call cgerfs (trans, n, nrhs, a, lda, af, ldaf, ipiv, b, ldb, x, ldx,
             ferr, berr, work, rwork, info)
call zgerfs (trans, n, nrhs, a, lda, af, ldaf, ipiv, b, ldb, x, ldx,
             ferr, berr, work, rwork, info)
```

説明

このルーチンは、一般マトリックス A による、複数の右辺を持つ連立 1 次方程式 $AX = B$ または $A^TX = B$ または $A^HX = B$ の解の精度の改善を繰り返し実行する。このルーチンは、算出されたそれぞれの解ベクトル x について、**成分ごとに後退誤差** β を計算する。次のように、x が摂動連立方程式の正確な解である場合、この誤差は、A と b の成分の最小相対摂動になる。

$|\delta a_{ij}|/|a_{ij}| \le \beta\,|a_{ij}|,\ |\delta b_i|/|b_i| \le \beta\,|b_i|$ ただし $(A + \delta A)x = (b + \delta b)$

最後に、このルーチンは、算出された解 $\|x - x_e\|_\infty/\|x\|_\infty$ の**成分ごとに前進誤差**を推定する (x_e は正確な解である)。

このルーチンを呼び出す前に、以下の手順を実行する必要がある。

- ?getrf 因子分解ルーチンを呼び出す。
- ?getrs 解の算出ルーチンを呼び出す。

入力パラメータ

trans CHARACTER*1。'N' または 'T' または 'C' でなければならない。

方程式の形式を指定する。

trans = 'N' の場合は、連立方程式の形式は $AX = B$ である。

trans = 'T' の場合は、連立方程式の形式は $A^TX = B$ である。

trans = 'C' の場合は、連立方程式の形式は $A^HX = B$ である。

n INTEGER。マトリックス A の次数 ($n \geq 0$)

nrhs INTEGER。右辺の数 ($nrhs \geq 0$)

a, af, b, x, work REAL (sgerfs の場合)
DOUBLE PRECISION (dgerfs の場合)
COMPLEX (cgerfs の場合)
DOUBLE COMPLEX (zgerfs の場合)

配列 :

a(*lda*, *) には、?getrf に入力として与えた元のマトリックス A を格納する。

af(*ldaf*, *) には、?getrf によって返された、因子分解後のマトリックス A を格納する。

b(*ldb*, *) には、右辺のマトリックス B を格納する。

x(*ldx*, *) には、解のマトリックス X を格納する。

work (*) は、ワークスペース配列である。

a と *af* の第 2 次元は、少なくとも max(1, *n*) でなければならない。*b* と *x* の第 2 次元は、少なくとも max (1, *nrhs*) でなければならない。*work* の次元は、実数型の場合は少なくとも max(1, 3*n*) で、複素数型の場合は max(1, 2*n*) でなければならない。

lda INTEGER。*a* の第 1 次元。*lda* ≥ max(1, *n*)

ldaf INTEGER。*af* の第 1 次元。*ldaf* ≥ max(1, *n*)

ldb INTEGER。*b* の第 1 次元。*ldb* ≥ max(1, *n*)

ldx INTEGER。*x* の第 1 次元。*ldx* ≥ max(1, *n*)

ipiv INTEGER
配列 DIMENSION は少なくとも max(1, *n*)
?getrf によって返される *ipiv* 配列。

iwork INTEGER
ワークスペース配列 DIMENSION は少なくとも max(1, *n*)

rwork REAL (cgerfs の場合)
DOUBLE PRECISION (zgerfs の場合)
ワークスペース配列 DIMENSION は少なくとも max(1, *n*)

出力パラメータ

x 精度が改善された解のマトリックス *X*。

ferr, berr REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
配列 DIMENSION は少なくとも max(1, *nrhs*)。各解ベクトルの成分ごとの前進誤差と後退誤差を格納する。

info INTEGER
info = 0 の場合は、正常に終了する。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

ferr に返される誤差範囲は厳密ではないが、実際の誤差はほぼ常にこれより小さくなる。

各右辺について、後退誤差の計算に必要な浮動小数点演算の回数は、実数型の場合は少なくとも $4n^2$ で、複素数型の場合は $16n^2$ である。さらに、繰り返し行われる解の精度の改善の各段階で必要な演算の回数は、実数型の場合は $6n^2$ で、複素数型の場合は $24n^2$ である。繰り返される回数の範囲は 1 ～ 5 になる。前進誤差を推定するには、連立 1 次方程式 $Ax = b$ を繰り返し解く必要がある。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約 $2n^2$ で、複素数型の場合は約 $8n^2$ である。

?gbrfs

一般バンド・マトリックスによる連立 1 次方程式の解の精度を改善し、解の誤差を推定する。

```
call sgbrfs (trans, n, kl, ku, nrhs, ab, ldab, afb, ldafb, ipiv, b,
             ldb, x, ldx, ferr, berr, work, iwork, info)

call dgbrfs (trans, n, kl, ku, nrhs, ab, ldab, afb, ldafb, ipiv, b,
             ldb, x, ldx, ferr, berr, work, iwork, info)

call cgbrfs (trans, n, kl, ku, nrhs, ab, ldab, afb, ldafb, ipiv, b,
             ldb, x, ldx, ferr, berr, work, rwork, info)

call zgbrfs (trans, n, kl, ku, nrhs, ab, ldab, afb, ldafb, ipiv, b,
             ldb, x, ldx, ferr, berr, work, rwork, info)
```

説明

このルーチンは、バンド・マトリックス A による、複数の右辺を持つ連立 1 次方程式 $AX = B$ または $A^TX = B$ または $A^HX = B$ の解の精度の改善を繰り返し実行する。このルーチンは、算出されたそれぞれの解ベクトル x について、**成分ごとに後退誤差** β を計算する。次のように、x が摂動連立方程式の正確な解である場合、この誤差は、A と b の成分の最小相対摂動になる。

$|\delta a_{ij}|/|a_{ij}| \le \beta\,|a_{ij}|,\ |\delta b_i|/|b_i| \le \beta\,|b_i|$ ただし、$(A + \delta A)x = (b + \delta b)$

最後に、このルーチンは、算出された解 $\|x - x_e\|_\infty/\|x\|_\infty$ の**成分ごとに前進誤差**を推定する (x_e は正確な解である)。

このルーチンを呼び出す前に、以下の手順を実行する必要がある。

- ?gbtrf 因子分解ルーチンを呼び出す。
- ?gbtrs 解の算出ルーチンを呼び出す。

入力パラメータ

trans CHARACTER*1。'N' または 'T' または 'C' でなければならない。

方程式の形式を指定する。

trans = 'N' の場合は、連立方程式の形式は $AX = B$ である。

trans = 'T' の場合は、連立方程式の形式は $A^TX = B$ である。

trans = 'C' の場合は、連立方程式の形式は $A^HX = B$ である。

n INTEGER。マトリックス A の次数 ($n \geq 0$)。

kl INTEGER。A のバンドの中の劣対角成分の数 ($kl \geq 0$)。

ku INTEGER。A のバンドの中の優対角成分の数 ($ku \geq 0$)。

nrhs INTEGER。右辺の数 ($nrhs \geq 0$)。

ab, afb, b, x, work REAL (sgbrfs の場合)
DOUBLE PRECISION (dgbrfs の場合)
COMPLEX (cgbrfs の場合)
DOUBLE COMPLEX (zgbrfs の場合)

配列 :

ab(*ldab*, *) には、?gbtrf に入力として与えた元のバンド・マトリックス A を格納する。このマトリックスは、行 1 ～ ($kl + ku + 1$) に格納される。

afb(*ldafb*, *) には、?gbtrf によって返された、因子分解後のバンド・マトリックス A を格納する。

b(*ldb*, *) には、右辺のマトリックス B を格納する。

x(*ldx*, *) には、解のマトリックス X を格納する。

work (*) は、ワークスペース配列である。

ab と *afb* の第 2 次元は、少なくとも max(1, *n*) でなければならない。*b* と *x* の第 2 次元は、少なくとも max(1, *nrhs*) でなければならない。*work* の次元は、実数型の場合は少なくとも max(1, 3*n) で、複素数型の場合は max(1, 2*n) でなければならない。

ldab INTEGER。*ab* の第 1 次元。

ldafb INTEGER。*afb* の第 1 次元。

ldb INTEGER。*b* の第 1 次元。*ldb* ≥ max(1, *n*)

ldx INTEGER。*x* の第 1 次元。*ldx* ≥ max(1, *n*)

ipiv INTEGER
配列 DIMENSION は少なくとも max(1, *n*)
?gbtrf によって返される *ipiv* 配列。

iwork INTEGER
ワークスペース配列 DIMENSION は少なくとも max(1, *n*)

rwork REAL (cgbrfs の場合)
DOUBLE PRECISION (zgbrfs の場合)
ワークスペース配列 DIMENSION は少なくとも max(1, *n*)

出力パラメータ

x 精度が改善された解のマトリックス *X*。

ferr, berr REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
配列 DIMENSION は少なくとも max(1, *nrhs*)。各解ベクトルの成分ごとの前進誤差と後退誤差を格納する。

info INTEGER
info =0 の場合は、正常に終了する。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

ferr に返される誤差範囲は厳密ではないが、実際の誤差はほぼ常にこれより小さくなる。

各右辺について、後退誤差の計算に必要な浮動小数点演算の回数は、実数型の場合は少なくとも $4n(kl + ku)$ で、複素数型の場合は $16n(kl + ku)$ である。さらに、繰り返し行われる解の精度の改善の各段階で必要な演算の回数は、実数型の場合は $2n(4kl + 3ku)$ で、複素数型の場合は $8n(4kl + 3ku)$ である。繰り返される回数の範囲は 1 ～ 5 になる。前進誤差を推定するには、連立 1 次方程式 $Ax = b$ を繰り返し解く必要がある。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約 $2n^2$ で、複素数型の場合は約 $8n^2$ である。

?gtrfs

三重対角マトリックスを係数行列とする
連立 1 次方程式の解の精度を改善し、
解の誤差を推定する。

```
call sgtrfs (trans, n, nrhs, dl, d, du, dlf, df, duf, du2, ipiv, b,
             ldb, x, ldx, ferr, berr, work, iwork, info)
call dgtrfs (trans, n, nrhs, dl, d, du, dlf, df, duf, du2, ipiv, b,
             ldb, x, ldx, ferr, berr, work, iwork, info)
call cgtrfs (trans, n, nrhs, dl, d, du, dlf, df, duf, du2, ipiv, b,
             ldb, x, ldx, ferr, berr, work, rwork, info)
call zgtrfs (trans, n, nrhs, dl, d, du, dlf, df, duf, du2, ipiv, b,
             ldb, x, ldx, ferr, berr, work, rwork, info)
```

説明

このルーチンは、三重対角マトリックス A を係数行列とする、複数の右辺を持つ連立 1 次方程式 $AX = B$ または $A^TX = B$ または $A^HX = B$ の解の精度の改善を繰り返し実行する。このルーチンは、算出されたそれぞれの解ベクトル x について、**成分ごとの後退誤差** β を計算する。次のように、x が摂動連立方程式の正確な解だとすると、この誤差は、A と b の成分の最小相対摂動になる。

$|\delta a_{ij}|/|a_{ij}| \le \beta\,|a_{ij}|,\ |\delta b_i|/|b_i| \le \beta\,|b_i|$ ただし、$(A + \delta A)x = (b + \delta b)$

最後に、このルーチンは、算出された解 $\|x - x_e\|_\infty/\|x\|_\infty$ の**成分ごとの前進誤差**を推定する (x_e は正確な解である)。

このルーチンを呼び出す前に、以下の手順を実行する必要がある。

- 因子分解ルーチン ?gttrf を呼び出す。
- 解の算出ルーチン ?gttrs を呼び出す。

入力パラメータ

trans　　CHARACTER*1。'N' または 'T' または 'C' でなければならない。
方程式の形式を指定する。
trans = 'N' の場合は、連立方程式の形式は $AX = B$ である。
trans = 'T' の場合は、連立方程式の形式は $A^TX = B$ である。
trans = 'C' の場合は、連立方程式の形式は $A^HX = B$ である。

n INTEGER。マトリックス *A* の次数 ($n \geq 0$)。

nrhs INTEGER。右辺の数、すなわち、マトリックス B の列の数 ($nrhs \geq 0$)。

dl,d, du, dlf, df, duf,
du2, b, x, work

REAL (sgtrfs の場合)
DOUBLE PRECISION (dgtrfs の場合)
COMPLEX (cgtrfs の場合)
DOUBLE COMPLEX (zgtrfs の場合)

配列 :

dl は、次元は (*n* - 1) で、*A* の劣対角成分を格納する。

d は、次元は (*n*) で、*A* の対角成分を格納する。

du は、次元は (*n* - 1) で、*A* の優対角成分を格納する。

dlf は、次元は (*n* - 1) で、?gttrf による *A* の *LU* 因子分解で得られたマトリックス *L* を定義する (*n* - 1) 個の乗数を格納する。

df は、次元は (*n*) で、*A* の *LU* 因子分解で得られた上三角マトリックス *U* の *n* 個の対角成分を格納する。

duf は、次元は (*n* - 1) で、*U* の最初の優対角成分の (*n* - 1) 個の成分を格納する。

du2 は、次元は (*n* - 2) で、*U* の 2 番目の優対角成分の (*n* - 2) 個のの成分を格納する。

b(ldb, nrhs) は、右辺のマトリックス *B* を格納する。

x(ldx, nrhs) は、?gttrs によって計算された解のマトリックス *X* を格納する。

work (*) はワークスペース配列である。
work の次元は、実数型の場合は少なくとも max(1, 3*n)、複素数型の場合は少なくとも max(1, 2*n) でなければならない。

ldb INTEGER。*b* の第 1 次元。*ldb* $\geq$ max(1, n)

ldx INTEGER。*x* の第 1 次元。*ldx* $\geq$ max(1, n)

ipiv INTEGER

配列 DIMENSION は少なくとも max(1, n)
?gttrf によって返される *ipiv* 配列。

iwork INTEGER
ワークスペース配列 DIMENSION は (*n*)。実数型でのみ使用される。

rwork REAL (cgtrfs の場合)
DOUBLE PRECISION (zgtrfs の場合)
ワークスペース配列 DIMENSION は (*n*)。複素数型でのみ使用される。

出力パラメータ

x 精度が改善された解のマトリックス *X*。

ferr, berr REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
配列 DIMENSION は少なくとも max(1, *nrhs*)。各解ベクトルの成分ごとの前進誤差と後退誤差を格納する。

info INTEGER
info = 0 の場合は、ルーチンの実行は正常に終了した。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。

?porfs

正値対称 (エルミート) マトリックス
による連立 1 次方程式の解の精度を
改善し、解の誤差を推定する。

```
call sporfs (uplo, n, nrhs, a, lda, af, ldaf, b, ldb, x, ldx, ferr,
             berr, work, iwork, info)

call dporfs (uplo, n, nrhs, a, lda, af, ldaf, b, ldb, x, ldx, ferr,
             berr, work, iwork, info)

call cporfs (uplo, n, nrhs, a, lda, af, ldaf, b, ldb, x, ldx, ferr,
             berr, work, rwork, info)

call zporfs (uplo, n, nrhs, a, lda, af, ldaf, b, ldb, x, ldx, ferr,
             berr, work, rwork, info)
```

説明

このルーチンは、正値対称 (エルミート) マトリックス A による、複数の右辺を持つ連立 1 次方程式 $AX = B$ の解の精度の改善を繰り返し実行する。このルーチンは、算出されたそれぞれの解ベクトル x について、**成分ごとに後退誤差** β を計算する。次のように、x が摂動連立方程式の正確な解である場合、この誤差は、A と b の成分の最小相対摂動になる。

$|\delta a_{ij}|/|a_{ij}| \le \beta\,|a_{ij}|,\ |\delta b_i|/|b_i| \le \beta\,|b_i|$ ただし、$(A + \delta A)x = (b + \delta b)$

最後に、このルーチンは、算出された解 $\|x - x_e\|_\infty/\|x\|_\infty$ の**成分ごとに前進誤差**を推定する (x_e は正確な解である)。

このルーチンを呼び出す前に、以下の手順を実行する必要がある。

- ?potrf 因子分解ルーチンを呼び出す。
- ?potrs 解の算出ルーチンを呼び出す。

入力パラメータ

uplo CHARACTER*1。'U' または 'L' でなければならない。
入力マトリックス A の因子分解の方法を指定する。
uplo = 'U' の場合は、配列 *af* には、コルスキー因子分解 $A = U^H U$ の係数 U を格納する。
uplo = 'L' の場合は、配列 *af* には、コルスキー因子分解 $A = LL^H$ の係数 L を格納する。

n INTEGER。マトリックス A の次数 ($n \geq 0$)

nrhs INTEGER。右辺の数 ($nrhs \geq 0$)

a, af, b, x, work REAL (sporfs の場合)
DOUBLE PRECISION (dporfs の場合)
COMPLEX (cporfs の場合)
DOUBLE COMPLEX (zporfs の場合)
配列 :
*a(lda, *)* には、?potrf に入力として与えた元のマトリックス A を格納する。
*af(ldaf, *)* には、?potrf によって返された、因子分解後のマトリックス A を格納する。
*b(ldb, *)* には、右辺のマトリックス B を格納する。
*x(ldx, *)* には、解のマトリックス X を格納する。
work (*) は、ワークスペース配列である。
a と *af* の第 2 次元は、少なくとも max(1, *n*) でなければならない。*b* と *x* の第 2 次元は、少なくとも max (1, *nrhs*) でなければならない。*work* の次元は、実数型の場合は少なくとも max(1, 3*n*) で、複素数型の場合は max(1, 2*n*) でなければならない。

lda INTEGER。*a* の第 1 次元。$lda \geq \max(1, n)$

ldaf INTEGER。*af* の第 1 次元。$ldaf \geq \max(1, n)$

ldb INTEGER。*b* の第 1 次元。$ldb \geq \max(1, n)$

ldx INTEGER。*x* の第 1 次元。$ldx \geq \max(1, n)$

iwork INTEGER
ワークスペース配列 DIMENSION は少なくとも max(1, *n*)

rwork REAL (cporfs の場合)
DOUBLE PRECISION (zporfs の場合)
ワークスペース配列 DIMENSION は少なくとも max(1, *n*)

出力パラメータ

x	精度が改善された解のマトリックス X。
ferr, berr	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列 DIMENSION は少なくとも max(1, *nrhs*)。各解ベクトルの成分ごとの前進誤差と後退誤差を格納する。
info	INTEGER *info* = 0 の場合は、正常に終了する。 *info* = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

ferr に返される誤差範囲は厳密ではないが、実際の誤差はほぼ常にこれより小さくなる。

各右辺について、後退誤差の計算に必要な浮動小数点演算の回数は、実数型の場合は少なくとも $4n^2$ で、複素数型の場合は $16n^2$ である。さらに、繰り返し行われる解の精度の改善の各段階で必要な演算の回数は、実数型の場合は $6n^2$ で、複素数型の場合は $24n^2$ である。繰り返される回数の範囲は 1 ～ 5 になる。前進誤差を推定するには、連立 1 次方程式 $Ax = b$ を繰り返し解く必要がある。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約 $2n^2$ で、複素数型の場合は約 $8n^2$ である。

?pprfs

パックド格納形式の正値対称
(エルミート) マトリックスによる
連立 1 次方程式の解の精度を改善し、
解の誤差を推定する。

```
call spprfs (uplo, n, nrhs, ap, afp, b, ldb, x, ldx, ferr, berr, work,
             iwork, info)
call dpprfs (uplo, n, nrhs, ap, afp, b, ldb, x, ldx, ferr, berr, work,
             iwork, info)

call cpprfs (uplo, n, nrhs, ap, afp, b, ldb, x, ldx, ferr, berr, work,
             rwork, info)
call zpprfs (uplo, n, nrhs, ap, afp, b, ldb, x, ldx, ferr, berr, work,
             rwork, info)
```

説明

このルーチンは、パックド格納形式の正値対称 (エルミート) マトリックス A による、複数の右辺を持つ連立 1 次方程式 $AX = B$ の解の精度の改善を繰り返し実行する。このルーチンは、算出されたそれぞれの解ベクトル x について、**成分ごとに後退誤差** β を計算する。次のように、x が摂動連立方程式の正確な解である場合、この誤差は、A と b の成分の最小相対摂動になる。ただし、

$|\delta a_{ij}|/|a_{ij}| \le \beta\,|a_{ij}|,\ |\delta b_i|/|b_i| \le \beta\,|b_i|$ ただし、$(A + \delta A)x = (b + \delta b)$

最後に、このルーチンは、算出された解 $\|x - x_e\|_\infty/\|x\|_\infty$ の**成分ごとに前進誤差**を推定する (x_e は正確な解である)。

このルーチンを呼び出す前に、以下の手順を実行する必要がある。

- ?pptrf 因子分解ルーチンを呼び出す。
- ?pptrs 解の算出ルーチンを呼び出す。

入力パラメータ

uplo CHARACTER*1。'U' または 'L' でなければならない。
入力マトリックス A の因子分解の方法を指定する。

uplo = 'U' の場合は、配列 afp には、コルスキー因子分解 $A = U^H U$ のパックド形式の係数 U を格納する。
uplo = 'L' の場合は、配列 afp には、コルスキー因子分解 $A = LL^H$ のパックド形式の係数 L を格納する。

n　INTEGER。マトリックス A の次数 ($n \geq 0$)

nrhs　INTEGER。右辺の数 ($nrhs \geq 0$)

ap, afp, b, x, work　REAL (spprfs の場合)
DOUBLE PRECISION (dpprfs の場合)
COMPLEX (cpprfs の場合)
DOUBLE COMPLEX (zpprfs の場合)

配列 :

ap(*) には、?pptrf に入力として与えた元のパックド形式のマトリックス A を格納する。

afp(*) には、?pptrf によって返された、因子分解後のパックド形式のマトリックス A を格納する。

b(ldb, *) には、右辺のマトリックス B を格納する。

x(ldx, *) には、解のマトリックス X を格納する。

work (*) は、ワークスペース配列である。

配列 ap と afp の次元は、少なくとも max(1, n(n+1)/2) でなければならない。b と x の第 2 次元は、少なくとも max(1, nrhs) でなければならない。work の次元は、実数型の場合は少なくとも max(1, 3*n) で、複素数型の場合は max(1, 2*n) でなければならない。

ldb　INTEGER。b の第 1 次元。ldb ≥ max(1, n)

ldx　INTEGER。x の第 1 次元。ldx ≥ max(1, n)

iwork　INTEGER
ワークスペース配列 DIMENSION は少なくとも max(1, n)

rwork　REAL (cpprfs の場合)
DOUBLE PRECISION (zpprfs の場合)
ワークスペース配列 DIMENSION は少なくとも max(1, n)

出力パラメータ

x　精度が改善された解のマトリックス X。

ferr, berr	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列 DIMENSION は少なくとも max(1, *nrhs*)。各解ベクトルの成分ごとの前進誤差と後退誤差を格納する。
info	INTEGER。*info* = 0 の場合は、正常に終了する。 *info* = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

ferr に返される誤差範囲は厳密ではないが、実際の誤差はほぼ常にこれより小さくなる。

各右辺について、後退誤差の計算に必要な浮動小数点演算の回数は、実数型の場合は少なくとも $4n^2$ で、複素数型の場合は $16n^2$ である。さらに、繰り返し行われる解の精度の改善の各段階で必要な演算の回数は、実数型の場合は $6n^2$ で、複素数型の場合は $24n^2$ である。繰り返される回数の範囲は 1 ～ 5 になる。

前進誤差を推定するには、連立 1 次方程式 $Ax = b$ を繰り返し解く必要がある。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約 $2n^2$ で、複素数型の場合は約 $8n^2$ である。

?pbrfs

正値対称 (エルミート) バンド・
マトリックスによる連立 1 次方程式の
解の精度を改善し、解の誤差を推定する。

```
call spbrfs (uplo, n, kd, nrhs, ab, ldab, afb, ldafb, b, ldb, x, ldx,
             ferr, berr, work, iwork, info)
call dpbrfs (uplo, n, kd, nrhs, ab, ldab, afb, ldafb, b, ldb, x, ldx,
             ferr, berr, work, iwork, info)

call cpbrfs (uplo, n, kd, nrhs, ab, ldab, afb, ldafb, b, ldb, x, ldx,
             ferr, berr, work, rwork, info)
call zpbrfs (uplo, n, kd, nrhs, ab, ldab, afb, ldafb, b, ldb, x, ldx,
             ferr, berr, work, rwork, info)
```

説明

このルーチンは、正値対称 (エルミート) バンド・マトリックス A による、複数の右辺を持つ連立 1 次方程式 $AX = B$ の解の精度の改善を繰り返し実行する。このルーチンは、算出されたそれぞれの解ベクトル x について、**成分ごとに後退誤差** β を計算する。次のように、x が摂動連立方程式の正確な解である場合、この誤差は、A と b の成分の最小相対摂動になる。

$|\delta a_{ij}|/|a_{ij}| \le \beta\,|a_{ij}|,\ |\delta b_i|/|b_i| \le \beta\,|b_i|$ ただし、$(A + \delta A)x = (b + \delta b)$

最後に、このルーチンは、算出された解 $\|x - x_e\|_\infty/\|x\|_\infty$ の**成分ごとに前進誤差**を推定する (x_e は正確な解である)。

このルーチンを呼び出す前に、以下の手順を実行する必要がある。

- ?pbtrf 因子分解ルーチンを呼び出す。
- ?pbtrs 解の算出ルーチンを呼び出す。

入力パラメータ

uplo CHARACTER*1。'U' または 'L' でなければならない。
入力マトリックス A の因子分解の方法を指定する。

uplo = 'U' の場合は、配列 afb には、コルスキー因子分解 $A = U^H U$ の係数 U を格納する。
uplo = 'L' 場合は、配列 afb には、コルスキー因子分解 $A = LL^H$ の係数 L を格納する。

n INTEGER。マトリックス A の次数 ($n \geq 0$)

kd INTEGER。A の優対角成分または劣対角成分の数 ($kd \geq 0$)

nrhs INTEGER。右辺の数 ($nrhs \geq 0$)

ab, afb, b, x, work REAL (spbrfs の場合)
DOUBLE PRECISION (dpbrfs の場合)
COMPLEX (cpbrfs の場合)
DOUBLE COMPLEX (zpbrfs の場合)

配列 :

ab(ldab, *) には、?pbtrf に入力として与えた元のバンド・マトリックス A を格納する。

afb(ldafb, *) には、?pbtrf によって返された、因子分解後のバンド・マトリックス A を格納する。

b(ldb, *) には、右辺のマトリックス B を格納する。

x(ldx, *) には、解のマトリックス X を格納する。

work (*) は、ワークスペース配列である。

ab と afb の第 2 次元は、少なくとも max(1, n) でなければならない。b と x の第 2 次元は、少なくとも max (1, nrhs) でなければならない。work の次元は、実数型の場合は少なくとも max(1, 3*n) で、複素数型の場合は max(1, 2*n) でなければならない。

ldab INTEGER。ab の第 1 次元。ldab ≥ kd + 1

ldafb INTEGER。fb の第 1 次元。ldafb ≥ kd + 1

ldb INTEGER。b の第 1 次元。ldb ≥ max(1, n)

ldx INTEGER。x の第 1 次元。ldx ≥ max(1, n)

iwork INTEGER
ワークスペース配列 DIMENSION は少なくとも max(1, n)

rwork REAL (cpbrfs の場合)
DOUBLE PRECISION (zpbrfs の場合)
ワークスペース配列 DIMENSION は少なくとも max(1, n)

出力パラメータ

x	精度が改善された解のマトリックス *X*。
ferr, berr	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列 DIMENSION は少なくとも max(1, *nrhs*)。各解ベクトルの成分ごとの前進誤差と後退誤差を格納する。
info	INTEGER *info* =0 の場合は、正常に終了する。 *info* = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

ferr に返される誤差範囲は厳密ではないが、実際の誤差はほぼ常にこれより小さくなる。

各右辺について、後退誤差の計算に必要な浮動小数点演算の回数は、実数型の場合は少なくとも 8*n***kd* で、複素数型の場合は 32*n***kd* である。さらに、繰り返し行われる解の精度の改善の各段階で必要な演算の回数は、実数型の場合は 12*n***kd* で、複素数型の場合は 48*n***kd* である。繰り返される回数の範囲は 1 ～ 5 になる。

前進誤差を推定するには、連立 1 次方程式 $Ax = b$ を繰り返し解く必要がある。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約 4*n***kd* で、複素数型の場合は約 16*n***kd* である。

?ptrfs

正値の対称 (エルミート) 三重対角
マトリックスを係数行列とする
連立 1 次方程式の解の精度を改善し、
解の誤差を推定する。

```
call sptrfs (n, nrhs, d, e, df, ef, b, ldb, x, ldx, ferr, berr, work,
             info)
call dptrfs (n, nrhs, d, e, df, ef, b, ldb, x, ldx, ferr, berr, work,
             info)
call cptrfs (uplo, n, nrhs, d, e, df, ef, b, ldb, x, ldx, ferr, berr,
             work, rwork, info)
call cptrfs (uplo, n, nrhs, d, e, df, ef, b, ldb, x, ldx, ferr, berr,
             work, rwork, info)
```

説明

このルーチンは、正値の対称 (エルミート) 三重対角マトリックス A を係数行列とし複数の右辺を持つ連立 1 次方程式 $AX = B$ の解の精度の改善を繰り返し実行する。このルーチンは、算出されたそれぞれの解ベクトル x について、**成分ごとの後退誤差** β を計算する。次のように、x が摂動連立方程式の正確な解だとすると、この誤差は、A と b の成分の最小相対摂動になる。

$|\delta a_{ij}|/|a_{ij}| \le \beta\,|a_{ij}|,\ |\delta b_i|/|b_i| \le \beta\,|b_i|$ ただし、$(A + \delta A)x = (b + \delta b)$

最後に、このルーチンは、算出された解 $\|x - x_e\|_\infty/\|x\|_\infty$ の**成分ごとの前進誤差**を推定する (x_e は正確な解である)。

このルーチンを呼び出す前に、以下の手順を実行する必要がある。

- 因子分解ルーチン ?pttrf を呼び出す。
- 解の算出ルーチン ?pttrs を呼び出す。

入力パラメータ

uplo `CHARACTER*1`。複素数型でのみ使用される。
`'U'` または `'L'` でなければならない。
三重対角マトリックス A の優対角成分と劣対角成分のどちらを格納するかと、A の因子分解の方法を指定する。

uplo = 'U' の場合は、配列 e には A の優対角成分が格納され、A は U^HDU として因子分解される。
uplo = 'L' の場合は、配列 e には A の劣対角成分が格納され、A は LDL^H として因子分解される。

n INTEGER。マトリックス A の次数 ($n \geq 0$)。

nrhs INTEGER。右辺の数 ($nrhs \geq 0$)。

d, df, rwork REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
配列 : d(n), df(n), rwork(n)
配列 d には、三重対角マトリックス A の n 個の対角成分が格納される。
配列 df には、?pttrf による A の因子分解で得られた対角マトリックス D の n 個の対角成分が格納される。
配列 rwork は、複素数型でのみ使用されるワークスペース配列である。

e, ef, b, x, work REAL (sptrfs の場合)
DOUBLE PRECISION (dptrfs の場合)
COMPLEX (cptrfs の場合)
DOUBLE COMPLEX (zptrfs の場合)
配列 : e(n - 1), ef(n - 1), b(ldb, nrhs), x(ldx, nrhs), work(*)。
配列 e には、三重対角マトリックス A の (n - 1) 個の対角線外の成分が格納される (uplo を参照)。
配列 ef には、?pttrf による因子分解で得られた単位二重対角係数 U または L の (n - 1) 個の対角線外の成分が格納される (uplo を参照)。
配列 b には、マトリックス B が格納される。このマトリックスの列は、連立方程式の右辺である。
配列 x には、?pttrs によって計算された解のマトリックス X が格納される。
配列 work はワークスペース配列である。work の次元は、実数型の場合は少なくとも 2*n、複素数型の場合は少なくとも n でなければならない。

ldb INTEGER。b のリーディング・ディメンジョン。
$ldb \geq \max(1, n)$

ldx INTEGER。x のリーディング・ディメンジョン。
$ldx \geq \max(1, n)$

出力パラメータ

x	精度が改善された解のマトリックス *X*。
ferr, berr	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列 DIMENSION は少なくとも max(1, *nrhs*)。各解ベクトルの成分ごとの前進誤差と後退誤差を格納する。
info	INTEGER *info* = 0 の場合は、ルーチンの実行は正常に終了した。 *info* = -*i* の場合は、*i* 番目のパラメータの値が不正である。

?syrfs

対称マトリックスによる連立 1 次方程式の
解の精度を改善し、解の誤差を推定する。

```
call ssyrfs (uplo, n, nrhs, a, lda, af, ldaf, ipiv, b, ldb, x, ldx,
             ferr, berr, work, iwork, info)
call dsyrfs (uplo, n, nrhs, a, lda, af, ldaf, ipiv, b, ldb, x, ldx,
             ferr, berr, work, iwork, info)

call csyrfs (uplo, n, nrhs, a, lda, af, ldaf, ipiv, b, ldb, x, ldx,
             ferr, berr, work, rwork, info)
call zsyrfs (uplo, n, nrhs, a, lda, af, ldaf, ipiv, b, ldb, x, ldx,
             ferr, berr, work, rwork, info)
```

説明

このルーチンは、フル格納形式の対称マトリックス A による、複数の右辺を持つ連立 1 次方程式 $AX = B$ の解の精度の改善を繰り返し実行する。このルーチンは、算出されたそれぞれの解ベクトル x について、**成分ごとに後退誤差** β を計算する。次のように、x が摂動連立方程式の正確な解である場合、この誤差は、A と b の成分の最小相対摂動になる。

$|\delta a_{ij}|/|a_{ij}| \le \beta\,|a_{ij}|,\ |\delta b_i|/|b_i| \le \beta\,|b_i|$ ただし、$(A + \delta A)x = (b + \delta b)$

最後に、このルーチンは、算出された解 $\|x - x_e\|_\infty/\|x\|_\infty$ の**成分ごとに前進誤差**を推定する (x_e は正確な解である)。

このルーチンを呼び出す前に、以下の手順を実行する必要がある。

- ?sytrf 因子分解ルーチンを呼び出す。
- ?sytrs 解の算出ルーチンを呼び出す。

入力パラメータ

uplo CHARACTER*1。'U' または 'L' でなければならない。
入力マトリックス A の因子分解の方法を指定する。

uplo = 'U' の場合は、配列 af には、Bunch-Kaufman 因子分解 $A = PUDU^TP^T$ を格納する。
uplo = 'L' の場合は、配列 af には、Bunch-Kaufman 因子分解 $A = PLDL^TP^T$ を格納する。

n　INTEGER。マトリックス A の次数 ($n \geq 0$)

nrhs　INTEGER。右辺の数 ($nrhs \geq 0$)

a, af, b, x, work　REAL (ssyrfs の場合)
DOUBLE PRECISION (dsyrfs の場合)
COMPLEX (csyrfs の場合)
DOUBLE COMPLEX (zsyrfs の場合)

配列 :

a(lda, *) には、?sytrf に入力として与えた元のマトリックス A を格納する。

af(ldaf, *) には、?sytrf によって返された、因子分解後のマトリックス A を格納する。

b(ldb, *) には、右辺のマトリックス B を格納する。

x(ldx, *) には、解のマトリックス X を格納する。

work (*) は、ワークスペース配列である。

a と af の第 2 次元は、少なくとも max(1, n) でなければならない。b と x の第 2 次元は、少なくとも max (1, nrhs) でなければならない。work の次元は、実数型の場合は少なくとも max(1, 3*n) で、複素数型の場合は max(1, 2*n) でなければならない。

lda　INTEGER。a の第 1 次元。lda ≥ max(1, n)

ldaf　INTEGER。af の第 1 次元。ldaf ≥ max(1, n)

ldb　INTEGER。b の第 1 次元。ldb ≥ max(1, n)

ldx　INTEGER。x の第 1 次元。ldx ≥ max(1, n)

ipiv　INTEGER
配列 DIMENSION は少なくとも max(1, n)
?sytrf によって返される ipiv 配列。

iwork　INTEGER
ワークスペース配列 DIMENSION は少なくとも max(1, n)

rwork　REAL (csyrfs の場合)
DOUBLE PRECISION (zsyrfs の場合)
ワークスペース配列 DIMENSION は少なくとも max(1, n)

出力パラメータ

x	精度が改善された解のマトリックス X。
ferr, berr	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列 DIMENSION は少なくとも max(1, *nrhs*)。各解ベクトルの成分ごとの前進誤差と後退誤差を格納する。
info	INTEGER *info* = 0 の場合は、正常に終了する。 *info* = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

ferr に返される誤差範囲は厳密ではないが、実際の誤差はほぼ常にこれより小さくなる。

各右辺について、後退誤差の計算に必要な浮動小数点演算の回数は、実数型の場合は少なくとも $4n^2$ で、複素数型の場合は $16n^2$ である。さらに、繰り返し行われる解の精度の改善の各段階で必要な演算の回数は、実数型の場合は $6n^2$ で、複素数型の場合は $24n^2$ である。繰り返される回数の範囲は 1 ～ 5 になる。前進誤差を推定するには、連立 1 次方程式 $Ax = b$ を繰り返し解く必要がある。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約 $2n^2$ で、複素数型の場合は約 $8n^2$ である。

?herfs

複素エルミート・マトリックスによる
連立 1 次方程式の解の精度を改善し、
解の誤差を推定する。

```
call cherfs (uplo, n, nrhs, a, lda, af, ldaf, ipiv, b, ldb, x, ldx,
             ferr, berr, work, rwork, info)
call zherfs (uplo, n, nrhs, a, lda, af, ldaf, ipiv, b, ldb, x, ldx,
             err, berr, work, rwork, info)
```

説明

このルーチンは、フル格納形式の複素エルミート・マトリックス A による、複数の右辺を持つ連立 1 次方程式 $AX = B$ の解の精度の改善を繰り返し実行する。このルーチンは、算出されたそれぞれの解ベクトル x について、**成分ごとに後退誤差** β を計算する。次のように、x が摂動連立方程式の正確な解である場合、この誤差は、A と b の成分の最小相対摂動になる。

$|\delta a_{ij}|/|a_{ij}| \le \beta\,|a_{ij}|,\ |\delta b_i|/|b_i| \le \beta\,|b_i|$ ただし、$(A + \delta A)x = (b + \delta b)$

最後に、このルーチンは、算出された解 $\|x - x_e\|_\infty/\|x\|_\infty$ の**成分ごとに前進誤差**を推定する (x_e は正確な解である)。

このルーチンを呼び出す前に、以下の手順を実行する必要がある。

- ?hetrf 因子分解ルーチンを呼び出す。
- ?hetrs 解の算出ルーチンを呼び出す。

入力パラメータ

uplo CHARACTER*1。'U' または 'L' でなければならない。
入力マトリックス A の因子分解の方法を指定する。
uplo = 'U' の場合は、配列 *af* には、Bunch-Kaufman 因子分解 $A = PUDU^HP^T$ を格納する。
uplo = 'L' の場合は、配列 *af* には、Bunch-Kaufman 因子分解 $A = PLDL^HP^T$ を格納する。

n INTEGER。マトリックス A の次数 ($n \ge 0$)

nrhs INTEGER。右辺の数 ($nrhs \geq 0$)

a, af, b, x, work COMPLEX (cherfs の場合)
DOUBLE COMPLEX (zherfs の場合)

配列 :

a(*lda*, *) には、?hetrf に入力として与えた元のマトリックス A を格納する。

af(*ldaf*, *) には、?hetrf によって返された、因子分解後のマトリックス A を格納する。

b(*ldb*, *) には、右辺のマトリックス B を格納する。

x(*ldx*, *) には、解のマトリックス X を格納する。

work (*) は、ワークスペース配列である。

a と *af* の第 2 次元は、少なくとも max(1, *n*) でなければならない。*b* と *x* の第 2 次元は、少なくとも max (1, *nrhs*) でなければならない。*work* の次元は、少なくとも max(1, 2*n*) でなければならない。

lda INTEGER。*a* の第 1 次元。$lda \geq \max(1, n)$

ldaf INTEGER。*af* の第 1 次元。$ldaf \geq \max(1, n)$

ldb INTEGER。*b* の第 1 次元。$ldb \geq \max(1, n)$

ldx INTEGER。*x* の第 1 次元。$ldx \geq \max(1, n)$

ipiv INTEGER
配列 DIMENSION は少なくとも max(1, *n*)
?hetrf によって返される *ipiv* 配列。

rwork REAL (cherfs の場合)
DOUBLE PRECISION (zherfs の場合)
ワークスペース配列 DIMENSION は少なくとも max(1, *n*)

出力パラメータ

x	精度が改善された解のマトリックス *X*。
ferr, berr	REAL (cherfs の場合) DOUBLE PRECISION (zherfs の場合) 配列 DIMENSION は少なくとも max(1, *nrhs*)。各解ベクトルの成分ごとの前進誤差と後退誤差を格納する。
info	INTEGER *info* = 0 の場合は、正常に終了する。 *info* = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

ferr に返される誤差範囲は厳密ではないが、実際の誤差はほぼ常にこれより小さくなる。

各右辺について、後退誤差の計算に必要な浮動小数点演算の回数は、少なくとも $16n^2$ である。さらに、繰り返し行われる解の精度の改善の各段階で必要な演算の回数は、$24n^2$ である。繰り返される回数の範囲は 1 ～ 5 になる。

前進誤差を推定するには、連立 1 次方程式 $Ax = b$ を繰り返し解く必要がある。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、約 $8n^2$ である。

このルーチンに対応する実数型は、ssyrfs / dsyrfs である。

?sprfs

パックド格納形式の対称マトリックス
による連立 1 次方程式の解の精度を
改善し、解の誤差を推定する。

```
call ssprfs (uplo, n, nrhs, ap, afp, ipiv, b, ldb, x, ldx, ferr, berr,
             work, iwork, info)
call dsprfs (uplo, n, nrhs, ap, afp, ipiv, b, ldb, x, ldx, ferr, berr,
             work, iwork, info)

call csprfs (uplo, n, nrhs, ap, afp, ipiv, b, ldb, x, ldx, ferr, berr,
             work, rwork, info)
call zsprfs (uplo, n, nrhs, ap, afp, ipiv, b, ldb, x, ldx, ferr, berr,
             work, rwork, info)
```

説明

このルーチンは、パックド格納形式の対称マトリックス A による、複数の右辺を持つ連立 1 次方程式 $AX = B$ の解の精度の改善を繰り返し実行する。このルーチンは、算出されたそれぞれの解ベクトル x について、**成分ごとに後退誤差** β を計算する。次のように、x が摂動連立方程式の正確な解である場合、この誤差は、A と b の成分の最小相対摂動になる。

$|\delta a_{ij}|/|a_{ij}| \le \beta\,|a_{ij}|,\ |\delta b_i|/|b_i| \le \beta\,|b_i|$ ただし、$(A + \delta A)x = (b + \delta b)$

最後に、このルーチンは、算出された解 $\|x - x_e\|_\infty/\|x\|_\infty$ の**成分ごとに前進誤差**を推定する (x_e は正確な解である)。

このルーチンを呼び出す前に、以下の手順を実行する必要がある。

- ?sptrf 因子分解ルーチンを呼び出す。
- ?sptrs 解の算出ルーチンを呼び出す。

入力パラメータ

uplo `CHARACTER*1`。`'U'` または `'L'` でなければならない。
入力マトリックス A の因子分解の方法を指定する。

uplo = 'U' の場合は、配列 afp には、パックド形式の Bunch-Kaufman 因子分解 $A = PUDU^TP^T$ を格納する。
uplo = 'L' の場合は、配列 afp には、パックド形式の Bunch-Kaufman 因子分解 $A = PLDL^TP$ を格納する。

n　INTEGER。マトリックス A の次数 ($n \geq 0$)

nrhs　INTEGER。右辺の数 ($nrhs \geq 0$)

ap, afp, b, x, work　REAL (ssprfs の場合)
DOUBLE PRECISION (dsprfs の場合)
COMPLEX (csprfs の場合)
DOUBLE COMPLEX (zsprfs の場合)

配列 :

ap(*) には、?sptrf に入力として与えた元のパックド形式のマトリックス A を格納する。

afp(*) には、?sptrf によって返された、因子分解後のパックド形式のマトリックス A を格納する。

b(ldb, *) には、右辺のマトリックス B を格納する。

x(ldx, *) には、解のマトリックス X を格納する。

work (*) は、ワークスペース配列である。

配列 ap と afp の次元は、少なくとも max(1, n(n+1)/2) でなければならない。b と x の第 2 次元は、少なくとも max(1, nrhs) でなければならない。work の次元は、実数型の場合は少なくとも max(1, 3*n) で、複素数型の場合は max(1, 2*n) でなければならない。

ldb　INTEGER。b の第 1 次元。ldb ≥ max(1, n)

ldx　INTEGER。x の第 1 次元。ldx ≥ max(1, n)

ipiv　INTEGER
配列 DIMENSION は少なくとも max(1, n)
?sptrf によって返される ipiv 配列。

iwork　INTEGER
ワークスペース配列 DIMENSION は少なくとも max(1, n)

rwork　REAL (csprfs の場合)
DOUBLE PRECISION (zsprfs の場合)
ワークスペース配列 DIMENSION は少なくとも max(1, n)

出力パラメータ

x	精度が改善された解のマトリックス X。
ferr, berr	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列 DIMENSION は少なくとも max(1, *nrhs*)。各解ベクトルの成分ごとの前進誤差と後退誤差を格納する。
info	INTEGER *info* = 0 の場合は、正常に終了する。 *info* = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

ferr に返される誤差範囲は厳密ではないが、実際の誤差はほぼ常にこれより小さくなる。

各右辺について、後退誤差の計算に必要な浮動小数点演算の回数は、実数型の場合は少なくとも $4n^2$ で、複素数型の場合は $16n^2$ である。さらに、繰り返し行われる解の精度の改善の各段階で必要な演算の回数は、実数型の場合は $6n^2$ で、複素数型の場合は $24n^2$ である。繰り返される回数の範囲は 1 ～ 5 になる。

前進誤差を推定するには、連立 1 次方程式 $Ax = b$ を繰り返し解く必要がある。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約 $2n^2$ で、複素数型の場合は約 $8n^2$ である。

?hprfs

パックド格納形式の複素エルミート・
マトリックスによる連立 1 次方程式の
解の精度を改善し、解の誤差を推定する。

```
call chprfs (uplo, n, nrhs, ap, afp, ipiv, b, ldb, x, ldx, ferr, berr,
             work, rwork, info)
call zhprfs (uplo, n, nrhs, ap, afp, ipiv, b, ldb, x, ldx, ferr, berr,
             work, rwork, info)
```

説明

このルーチンは、パックド格納形式の複素エルミート・マトリックス A による、複数の右辺を持つ連立 1 次方程式 $AX = B$ の解の精度の改善を繰り返し実行する。このルーチンは、算出されたそれぞれの解ベクトル x について、**成分ごとに後退誤差** β を計算する。次のように、x が摂動連立方程式の正確な解である場合、この誤差は、A と b の成分の最小相対摂動になる。

$|\delta a_{ij}|/|a_{ij}| \le \beta\,|a_{ij}|,\ |\delta b_i|/|b_i| \le \beta\,|b_i|$ ただし、$(A + \delta A)x = (b + \delta b)$

最後に、このルーチンは、算出された解 $\|x - x_e\|_\infty/\|x\|_\infty$ の**成分ごとに前進誤差**を推定する (x_e は正確な解である)。

このルーチンを呼び出す前に、以下の手順を実行する必要がある。

- ?hptrf 因子分解ルーチンを呼び出す。
- ?hptrs 解の算出ルーチンを呼び出す。

入力パラメータ

uplo	CHARACTER*1。'U' または 'L' でなければならない。入力マトリックス A の因子分解の方法を指定する。 *uplo* = 'U' の場合は、配列 *afp* には、パックド形式の Bunch-Kaufman 因子分解 $A = PUDU^HP^T$ を格納する。 *uplo* = 'L' の場合には、配列 *afp* は、パックド形式の Bunch-Kaufman 因子分解 $A = PLDL^HP^T$ を格納する。
n	INTEGER。マトリックス A の次数 ($n \ge 0$)
nrhs	INTEGER。右辺の数 ($nrhs \ge 0$)

ap, afp, b, x, work　　COMPLEX (chprfs の場合)
DOUBLE COMPLEX (zhprfs の場合)

配列 :

ap(*) には、?hptrf に入力として与えた元のパックド形式のマトリックス *A* を格納する。

afp(*) には、?hptrf によって返された、因子分解後のパックド形式のマトリックス *A* を格納する。

b(*ldb*, *) には、右辺のマトリックス *B* を格納する。

x(*ldx*, *) には、解のマトリックス *X* を格納する。

work (*) は、ワークスペース配列である。

配列 *ap* と *afp* の次元は、少なくとも max(1, *n*(*n*+1)/2) でなければならない。*b* と *x* の第 2 次元は、少なくとも max(1, *nrhs*) でなければならない。*work* の次元は、少なくとも max(1, 2**n*) でなければならない。

ldb　　INTEGER。*b* の第 1 次元。*ldb* ≥ max(1, *n*)

ldx　　INTEGER。*x* の第 1 次元。*ldx* ≥ max(1, *n*)

ipiv　　INTEGER
配列 DIMENSION は少なくとも max(1, *n*)
?hptrf によって返される *ipiv* 配列。

rwork　　REAL (chprfs の場合)
DOUBLE PRECISION (zhprfs の場合)
ワークスペース配列 DIMENSION は少なくとも max(1, *n*)

出力パラメータ

x　　精度が改善された解のマトリックス *X*。

ferr, berr　　REAL (chprfs の場合)
DOUBLE PRECISION (zhprfs の場合)
配列 DIMENSION は少なくとも max(1, *nrhs*)。各解ベクトルの成分ごとの前進誤差と後退誤差を格納する。

info　　INTEGER
info = 0 の場合は、正常に終了する。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

ferr に返される誤差範囲は厳密ではないが、実際の誤差はほぼ常にこれより小さくなる。

各右辺について、後退誤差の計算に必要な浮動小数点演算の回数は、少なくとも $16n^2$ である。さらに、繰り返し行われる解の精度の改善の各段階で必要な演算の回数は、$24n^2$ である。繰り返される回数の範囲は 1 ～ 5 になる。

前進誤差を推定するには、連立 1 次方程式 $Ax = b$ を繰り返し解く必要がある。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、約 $8n^2$ である。

このルーチンに対応する実数型は、ssprfs / dsprfs である。

?trrfs

三角マトリックスによる連立 1 次方程式の解の誤差を推定する。

```
call strrfs (uplo, trans, diag, n, nrhs, a, lda, b, ldb, x, ldx, ferr,
             berr, work, iwork, info)
call dtrrfs (uplo, trans, diag, n, nrhs, a, lda, b, ldb, x, ldx, ferr,
             berr, work, iwork, info)

call ctrrfs (uplo, trans, diag, n, nrhs, a, lda, b, ldb, x, ldx, ferr,
             berr, work, rwork, info)
call ztrrfs (uplo, trans, diag, n, nrhs, a, lda, b, ldb, x, ldx, ferr,
             berr, work, rwork, info)
```

説明

このルーチンは、三角マトリックス A による、複数の右辺を持つ連立 1 次方程式 $AX = B$ または $A^TX = B$ または $A^HX = B$ の解の誤差を推定する。このルーチンは、算出されたそれぞれの解ベクトル x について、**成分ごとに後退誤差** β を計算する。次のように、x が摂動連立方程式の正確な解である場合、この誤差は、A と b の成分の最小相対摂動になる。

$|\delta a_{ij}|/|a_{ij}| \le \beta\,|a_{ij}|,\ |\delta b_i|/|b_i| \le \beta\,|b_i|$ ただし、$(A + \delta A)x = (b + \delta b)$

また、このルーチンは、算出された解 $\|x - x_e\|_\infty / \|x\|_\infty$ の**成分ごとに前進誤差**を推定する (x_e は正確な解である)。

このルーチンを呼び出す前に、?trtrs 解の算出ルーチンを呼び出す。

入力パラメータ

uplo CHARACTER*1。'U' または 'L' でなければならない。

A が上三角マトリックスか、下三角マトリックスかを指定する。

uplo = 'U' の場合は、A は上三角マトリックスである。
uplo = 'L' の場合は、A は下三角マトリックスである。

trans CHARACTER*1。'N' または 'T' または 'C' でなければならない。

方程式の形式を指定する。

trans = 'N' の場合は、連立方程式の形式は $AX = B$ である。

trans = 'T' の場合は、連立方程式の形式は $A^TX = B$ である。

trans = 'C' の場合は、連立方程式の形式は $A^HX = B$ である。

diag CHARACTER*1。'N' または 'U' でなければならない。

diag = 'N' の場合は、A は単位三角マトリックスではない。

diag = 'U' の場合は、A は単位三角マトリックスである。A の対角成分は 1 と見なされ、配列 *a* 内で参照されない。

n INTEGER。マトリックス A の次数 ($n \geq 0$)

nrhs INTEGER。右辺の数 ($nrhs \geq 0$)

a, b, x, work
REAL (strrfs の場合)
DOUBLE PRECISION (dtrrfs の場合)
COMPLEX (ctrrfs の場合)
DOUBLE COMPLEX (ztrrfs の場合)

配列 :

*a(lda, *)* には、*uplo* によって指定される上三角または下三角マトリックス A を格納する。

*b(ldb, *)* には、右辺のマトリックス B を格納する。

*x(ldx, *)* には、解のマトリックス X を格納する。

work (*) は、ワークスペース配列である。
a の第 2 次元は、少なくとも max(1, n) でなければならない。b と x の第 2 次元は、少なくとも max(1, nrhs) でなければならない。work の次元は、実数型の場合は少なくとも max(1, 3*n) で、複素数型の場合は max(1, 2*n) でなければならない。

lda	INTEGER。a の第 1 次元。lda ≥ max(1, n)
ldb	INTEGER。b の第 1 次元。ldb ≥ max(1, n)
ldx	INTEGER。x の第 1 次元。ldx ≥ max(1, n)
iwork	INTEGER ワークスペース配列 DIMENSION は少なくとも max(1, n)
rwork	REAL (ctrrfs の場合) DOUBLE PRECISION (ztrrfs の場合) ワークスペース配列 DIMENSION は少なくとも max(1, n)

出力パラメータ

ferr, berr	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列 DIMENSION は少なくとも max(1, nrhs)。各解ベクトルの成分ごとの前進誤差と後退誤差を格納する。
info	INTEGER info = 0 の場合は、正常に終了する。 info = -i の場合は、i 番目のパラメータの値が不正である。

アプリケーション・ノート

ferr に返される誤差範囲は厳密ではないが、実際の誤差はほぼ常にこれより小さくなる。

このルーチンを呼び出す場合は、各右辺について、連立 1 次方程式 $Ax = b$ を繰り返し解く必要がある。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約 n^2 で、複素数型の場合は約 $4n^2$ である。

?tprfs

パックド格納形式の三角マトリックスによる連立 1 次方程式の解の誤差を推定する。

```
call stprfs (uplo, trans, diag, n, nrhs, ap, b, ldb, x, ldx, ferr,
             berr, work, iwork, info)

call dtprfs (uplo, trans, diag, n, nrhs, ap, b, ldb, x, ldx, ferr,
             berr, work, iwork, info)

call ctprfs (uplo, trans, diag, n, nrhs, ap, b, ldb, x, ldx, ferr,
             berr, work, rwork, info)

call ztprfs (uplo, trans, diag, n, nrhs, ap, b, ldb, x, ldx, ferr,
             berr, work, rwork, info)
```

説明

このルーチンは、パックド格納形式の三角マトリックス A による、複数の右辺を持つ連立 1 次方程式 $AX = B$ または $A^TX = B$ または $A^HX = B$ の解の誤差を推定する。このルーチンは、算出されたそれぞれの解ベクトル x について、**成分ごとに後退誤差** β を計算する。次のように、x が摂動連立方程式の正確な解である場合、この誤差は、A と b の成分の最小相対摂動になる。

$|\delta a_{ij}|/|a_{ij}| \le \beta\,|a_{ij}|,\ |\delta b_i|/|b_i| \le \beta\,|b_i|$ ただし、$(A + \delta A)x = (b + \delta b)$

また、このルーチンは、算出された解 $\|x - x_e\|_\infty/\|x\|_\infty$ の**成分ごとに前進誤差**を推定する (x_e は正確な解である)。

このルーチンを呼び出す前に、?tptrs 解の算出ルーチンを呼び出す。

入力パラメータ

uplo
CHARACTER*1。'U' または 'L' でなければならない。
A が上三角マトリックスか、下三角マトリックスかを指定する。
uplo = 'U' の場合は、A は上三角マトリックスである。
uplo = 'L' の場合は、A は下三角マトリックスである。

trans
CHARACTER*1。'N' または 'T' または 'C' でなければならない。
方程式の形式を指定する。
trans = 'N' の場合は、連立方程式の形式は $AX = B$ である。
trans = 'T' の場合は、連立方程式の形式は $A^TX = B$ である。
trans = 'C' の場合は、連立方程式の形式は $A^HX = B$ である。

diag
CHARACTER*1。'N' または 'U' でなければならない。
diag = 'N' の場合は、A は単位三角マトリックスではない。
diag = 'U' の場合は、A は単位三角マトリックスである。A の対角成分は 1 と見なされ、配列 *ap* 内で参照されない。

n
INTEGER。マトリックス A の次数 ($n \geq 0$)

nrhs
INTEGER。右辺の数 ($nrhs \geq 0$)

ap, b, x, work
REAL (strrfs の場合)
DOUBLE PRECISION (dtrrfs の場合)
COMPLEX (ctrrfs の場合)
DOUBLE COMPLEX (ztrrfs の場合)
配列 :
ap(*) には、*uplo* によって指定される上三角または下三角マトリックス A を格納する。
b(*ldb*, *) には、右辺のマトリックス B を格納する。
x(*ldx*, *) には、解のマトリックス X を格納する。
work (*) は、ワークスペース配列である。
ap の次元は、少なくとも max(1, *n*(*n*+1)/2) でなければならない。*b* と *x* の第 2 次元は、少なくとも max(1, *nrhs*) でなければならない。*work* の次元は、実数型の場合は少なくとも max(1, 3*n*) で、複素数型の場合は max(1, 2*n*) でなければならない。

ldb
INTEGER。*b* の第 1 次元。*ldb* ≥ max(1, *n*)

ldx INTEGER。*x* の第 1 次元。*ldx* ≥ max(1, *n*)

iwork INTEGER
ワークスペース配列 DIMENSION は少なくとも max(1, *n*)

rwork REAL (ctrrfs の場合)
DOUBLE PRECISION (ztrrfs の場合)
ワークスペース配列 DIMENSION は少なくとも max(1, *n*)

出力パラメータ

ferr, berr REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
配列 DIMENSION は少なくとも max(1, *nrhs*)。各解ベクトルの成分ごとの前進誤差と後退誤差を格納する。

info INTEGER
info = 0 の場合は、正常に終了する。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

ferr に返される誤差範囲は厳密ではないが、実際の誤差はほぼ常にこれより小さくなる。

このルーチンを呼び出す場合は、各右辺について、連立 1 次方程式 $Ax = b$ を繰り返し解く必要がある。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約 n^2 で、複素数型の場合は約 $4n^2$ である。

?tbrfs

三角バンド・マトリックスによる
連立 1 次方程式の解の誤差を推定する。

```
call stbrfs (uplo, trans, diag, n, kd, nrhs, ab, ldab, b, ldb, x, ldx,
             ferr, berr, work, iwork, info)

call dtbrfs (uplo, trans, diag, n, kd, nrhs, ab, ldab, b, ldb, x, ldx,
             ferr, berr, work, iwork, info)

call ctbrfs (uplo, trans, diag, n, kd, nrhs, ab, ldab, b, ldb, x, ldx,
             ferr, berr, work, rwork, info)

call ztbrfs (uplo, trans, diag, n, kd, nrhs, ab, ldab, b, ldb, x, ldx,
             ferr, berr, work, rwork, info)
```

説明

このルーチンは、三角バンド・マトリックス A による、複数の右辺を持つ連立 1 次方程式 $AX = B$ または $A^TX = B$ または $A^HX = B$ の解の誤差を推定する。このルーチンは、算出されたそれぞれの解ベクトル x について、**成分ごとに後退誤差** β を計算する。次のように、x が摂動連立方程式の正確な解である場合、この誤差は、A と b の成分の最小相対摂動になる。

$|\delta a_{ij}|/|a_{ij}| \le \beta\,|a_{ij}|,\ |\delta b_i|/|b_i| \le \beta\,|b_i|$ ただし、$(A + \delta A)x = (b + \delta b)$

また、このルーチンは、算出された解 $\|x - x_e\|_\infty/\|x\|_\infty$ の**成分ごとに前進誤差**を推定する (x_e は正確な解である)。

このルーチンを呼び出す前に、?tbtrs 解の算出ルーチンを呼び出す。

入力パラメータ

uplo CHARACTER*1。'U' または 'L' でなければならない。

A が上三角マトリックスか、下三角マトリックスかを指定する。

uplo = 'U' の場合は、A は上三角マトリックスである。
uplo = 'L' の場合は、A は下三角マトリックスである。

trans CHARACTER*1。'N' または 'T' または 'C' でなければならない。

方程式の形式を指定する。

trans = 'N' の場合は、連立方程式の形式は $AX = B$ である。

trans = 'T' の場合は、連立方程式の形式は $A^TX = B$ である。

trans = 'C' の場合は、連立方程式の形式は $A^HX = B$ である。

diag CHARACTER*1。'N' または 'U' でなければならない。

diag = 'N' の場合は、A は単位三角マトリックスではない。

diag = 'U' の場合は、A は単位三角マトリックスである。A の対角成分は 1 と見なされ、配列 *ab* 内で参照されない。

n INTEGER。マトリックス A の次数 ($n \geq 0$)

kd INTEGER。A の優対角成分または劣対角成分の数 ($kd \geq 0$)

nrhs INTEGER。右辺の数 ($nrhs \geq 0$)

ab, b, x, work

REAL (stbrfs の場合)
DOUBLE PRECISION (dtbrfs の場合)
COMPLEX (ctbrfs の場合)
DOUBLE COMPLEX (ztbrfs の場合)

配列 :

*ab(ldab, *)* には、*uplo* によって指定される上三角または下三角マトリックス A をバンド格納形式で格納する。

*b(ldb, *)* には、右辺のマトリックス B を格納する。

*x(ldx, *)* には、解のマトリックス X を格納する。

work (*) は、ワークスペース配列である。

	a の第 2 次元は、少なくとも max(1, *n*) でなければならない。*b* と *x* の第 2 次元は、少なくとも max(1, *nrhs*) でなければならない。*work* の次元は、実数型の場合は少なくとも max(1, 3**n*) で、複素数型の場合は max(1, 2**n*) でなければならない。
ldab	INTEGER。配列 *ab* の第 1 次元。(*ldab* ≥ *kd* +1)
ldb	INTEGER。*b* の第 1 次元。*ldb* ≥ max(1, *n*)
ldx	INTEGER。*x* の第 1 次元。*ldx* ≥ max(1, *n*)
iwork	INTEGER ワークスペース配列 DIMENSION は少なくとも max(1, *n*)
rwork	REAL (ctbrfs の場合) DOUBLE PRECISION (ztbrfs の場合) ワークスペース配列 DIMENSION は少なくとも max(1, *n*)

出力パラメータ

ferr, berr	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列 DIMENSION は少なくとも max(1, *nrhs*)。各解ベクトルの成分ごとの前進誤差と後退誤差を格納する。
info	INTEGER *info* = 0 の場合は、正常に終了する。 *info* = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

ferr に返される誤差範囲は厳密ではないが、実際の誤差はほぼ常にこれより小さくなる。

このルーチンを呼び出す場合は、各右辺について、連立 1 次方程式 $Ax = b$ を繰り返し解く必要がある。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約 2*n***kd* で、複素数型の場合は約 8*n***kd* である。

マトリックスの反転用のルーチン

マトリックスの逆行列を明示的に計算することは、実際にはほとんどない。
特に、連立方程式 $Ax = b$ を解く場合に、最初に A^{-1} を計算してから、マトリックスとベクトルの積 $x = A^{-1}b$ を計算してはならない。
連立方程式を解く場合には、解の算出ルーチンを呼び出す (「計算ルーチン」を参照)。この方が効率が良く、高い精度の結果が得られる。

ただし、逆行列を求める必要が生じた場合に備えて、マトリックス反転用のルーチンが用意されている。

?getri

LU 因子分解された一般マトリックスの
逆行列を計算する。

```
call sgetri (n, a, lda, ipiv, work, lwork, info)
call dgetri (n, a, lda, ipiv, work, lwork, info)
call cgetri (n, a, lda, ipiv, work, lwork, info)
call zgetri (n, a, lda, ipiv, work, lwork, info)
```

説明

このルーチンは、一般マトリックス A の逆行列 (A^{-1}) を計算する。
このルーチンを呼び出す前に、?getrf を呼び出して、A を因子分解する。

入力パラメータ

n	INTEGER。マトリックス A の次数 ($n \geq 0$)
a, work	REAL (sgetri の場合) DOUBLE PRECISION (dgetri の場合) COMPLEX (cgetri の場合) DOUBLE COMPLEX (zgetri の場合) 配列 : *a(lda, *)*, *work(lwork)* *a(lda, *)* には、?getrf によって返される、マトリックス A の因子分解 $A = PLU$ を格納する。 *a* の第 2 次元は、少なくとも max(1, *n*) でなければならない。 *work(lwork)* は、ワークスペース配列である。

lda	INTEGER。*a* の第 1 次元。*lda* ≥ max(1, *n*)
ipiv	INTEGER 配列 DIMENSION は少なくとも max(1, *n*) ?getrf によって返される *ipiv* 配列。
lwork	INTEGER。*work* 配列のサイズ (*lwork* ≥ *n*) *lwork* の推奨値については、「アプリケーション・ノート」を参照。

出力パラメータ

a	*n* × *n* のマトリックス A^{-1} によって上書きされる。
work(1)	*info* = 0 の場合は、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の *lwork* の値が *work*(1) に出力される。これ以降の実行には、この *lwork* の値を使用する。
info	INTEGER。*info* = 0 の場合は、正常に終了する。 *info* = -*i* の場合は、*i* 番目のパラメータの値が不正である。 *info* = *i* の場合は、係数 *U* の *i* 番目の対角成分がゼロで、*U* が特異になるため、マトリックスの反転を完了できない。

アプリケーション・ノート

パフォーマンスを最適化するには、*lwork* = *n***blocksize* に設定する。*blocksize* は、**ブロック・アルゴリズム**の最適なパフォーマンスに必要なマシンによって異なる値 (通常は 16 ～ 64) である。

必要なワークスペースの大きさがわからない場合は、最初の実行では *lwork* を大きめの値に設定する。実行の終了時に *work*(1) を調べて、これ以降の実行にはその値を使用する。

算出された逆行列 *X* は、以下の誤差範囲を満たす。

$$|XA - I| \leq c(n)\varepsilon\, |X|P|L||U|$$

c(*n*) は *n* の適度な 1 次関数で、ε はマシンの精度である。*I* は元のマトリックスを示す。*P*、*L*、*U* は、マトリックスの因子分解 *A* = *PLU* の係数である。

浮動小数点演算の合計回数は、実数型の場合は約 $(4/3)n^3$ で、複素数型の場合は約 $(16/3)n^3$ になる。

?potri

正値対称 (エルミート) マトリックスの
逆行列を計算する。

```
call spotri (uplo, n, a, lda, info)
call dpotri (uplo, n, a, lda, info)
call cpotri (uplo, n, a, lda, info)
call zpotri (uplo, n, a, lda, info)
```

説明

このルーチンは、正値対称または (複素マトリックスの場合は) 正値エルミート・マトリックス A の逆行列 (A^{-1}) を計算する。
このルーチンを呼び出す前に、?potrf を呼び出して、A を因子分解する。

入力パラメータ

uplo CHARACTER*1。'U' または 'L' でなければならない。
入力マトリックス A の因子分解の方法を指定する。
uplo = 'U' の場合は、配列 *a* には、コルスキー因子分解 $A = U^H U$ の係数 U を格納する。
uplo = 'L' の場合は、配列 *a* には、コルスキー因子分解 $A = LL^H$ の係数 L を格納する。

n INTEGER。マトリックス A の次数 ($n \geq 0$)

a REAL (spotri の場合)
DOUBLE PRECISION (dpotri の場合)
COMPLEX (cpotri の場合)
DOUBLE COMPLEX (zpotri の場合)
配列 : *a(lda, *)*
?potrf によって返される、マトリックス A の因子分解を格納する。
a の第 2 次元は、少なくとも max(1, *n*) でなければならない。

lda INTEGER。*a* の第 1 次元。*lda* ≥ max(1, *n*)

出力パラメータ

a	*n* × *n* のマトリックス A^{-1} によって上書きされる。
info	INTEGER *info* = 0 の場合は、正常に終了する。 *info* = -*i* の場合は、*i* 番目のパラメータの値が不正である。 *info* = *i* の場合は、コルスキー係数の *i* 番目の対角成分 (したがって、係数そのもの) がゼロのため、反転を完了できない。

アプリケーション・ノート

算出された逆行列 X は、以下の誤差範囲を満たす。

$$\|XA - I\|_2 \le c(n)\varepsilon\kappa_2(A), \quad \|AX - I\|_2 \le c(n)\varepsilon\kappa_2(A)$$

$c(n)$ は n の適度な 1 次関数で、ε はマシンの精度である。I は元のマトリックスを示す。

マトリックス A の 2- ノルム $\|A\|_2$ は、$\|A\|_2 = \max_{x \cdot x=1}(Ax \cdot Ax)^{1/2}$ によって定義される。条件数 $\kappa_2(A)$ は、$\kappa_2(A) = \|A\|_2 \|A^{-1}\|_2$ によって定義される。

浮動小数点演算の合計回数は、実数型の場合は約 $(2/3)n^3$ で、複素数型の場合は約 $(8/3)n^3$ になる。

?pptri

パックド格納形式の正値対称
(エルミート) マトリックスの逆行列を
計算する。

```
call spptri (uplo, n, ap, info)
call dpptri (uplo, n, ap, info)
call cpptri (uplo, n, ap, info)
call zpptri (uplo, n, ap, info)
```

説明

このルーチンは、**パックド**形式の正値対称または (複素マトリックスの場合は) 正値エルミート・マトリックス A の逆行列 (A^{-1}) を計算する。このルーチンを呼び出す前に、?pptrf を呼び出して、A を因子分解する。

入力パラメータ

uplo CHARACTER*1。'U' または 'L' でなければならない。
入力マトリックス A の因子分解の方法を指定する。
uplo = 'U' の場合は、配列 *ap* には、コルスキー因子分解 $A = U^H U$ のパックド形式の係数 U を格納する。
uplo = 'L' の場合は、配列 *ap* には、コルスキー因子分解 $A = LL^H$ のパックド形式の係数 L を格納する。

n INTEGER。マトリックス A の次数 ($n \geq 0$)

ap REAL (spptri の場合)
DOUBLE PRECISION (dpptri の場合)
COMPLEX (cpptri の場合)
DOUBLE COMPLEX (zpptri の場合)
配列 DIMENSION は少なくとも max(1, $n(n+1)/2$)
?pptrf によって返される、パックド形式のマトリックス A の因子分解を格納する。
ap の次元は、少なくとも max(1, $n(n+1)/2$) でなければならない。

出力パラメータ

ap	パックド形式の $n \times n$ のマトリックス A^{-1} によって上書きされる。
info	INTEGER *info* = 0 の場合は、正常に終了する。 *info* = -*i* の場合は、*i* 番目のパラメータの値が不正である。 *info* = *i* の場合は、コルスキー係数の *i* 番目の対角成分 (したがって、係数そのもの) がゼロのため、反転を完了できない。

アプリケーション・ノート

算出された逆行列 X は、以下の誤差範囲を満たす。

$$\|XA - I\|_2 \le c(n)\varepsilon\kappa_2(A), \quad \|AX - I\|_2 \le c(n)\varepsilon\kappa_2(A)$$

$c(n)$ は n の適度な 1 次関数で、ε はマシンの精度である。I は元のマトリックスを示す。

マトリックス A の 2- ノルム $\|A\|_2$ は、$\|A\|_2 = \max_{x \cdot x=1}(Ax \cdot Ax)^{1/2}$ によって定義される。条件数 $\kappa_2(A)$ は、$\kappa_2(A) = \|A\|_2 \|A^{-1}\|_2$ によって定義される。

浮動小数点演算の合計回数は、実数型の場合は約 $(2/3)n^3$ で、複素数型の場合は約 $(8/3)n^3$ になる。

?sytri

対称マトリックスの逆行列を計算する。

```
call ssytri (uplo, n, a, lda, ipiv, work, info)
call dsytri (uplo, n, a, lda, ipiv, work, info)
call csytri (uplo, n, a, lda, ipiv, work, info)
call zsytri (uplo, n, a, lda, ipiv, work, info)
```

説明

このルーチンは、対称マトリックス A の逆行列 (A^{-1}) を計算する。
このルーチンを呼び出す前に、?sytrf を呼び出して、A を因子分解する。

入力パラメータ

uplo CHARACTER*1。'U' または 'L' でなければならない。
入力マトリックス A の因子分解の方法を指定する。
uplo = 'U' の場合は、配列 *a* には、Bunch-Kaufman 因子分解 $A = PUDU^TP^T$ を格納する。
uplo = 'L' の場合は、配列 *a* には、Bunch-Kaufman 因子分解 $A = PLDL^TP^T$ を格納する。

n INTEGER。マトリックス A の次数 ($n \geq 0$)

a, work REAL (ssytri の場合)
DOUBLE PRECISION (dsytri の場合)
COMPLEX (csytri の場合)
DOUBLE COMPLEX (zsytri の場合)
配列 :
*a(lda, *)* には、?sytrf によって返される、マトリックス A の因子分解を格納する。
a の第 2 次元は、少なくとも max(1, *n*) でなければならない。
work(*) は、ワークスペース配列である。
work の次元は、少なくとも max(1, 2*n) でなければならない。

lda INTEGER。*a* の第 1 次元。*lda* ≥ max(1, *n*)

ipiv INTEGER
配列 DIMENSION は少なくとも max(1, *n*)
?sytrf によって返される *ipiv* 配列。

出力パラメータ

a $n \times n$ のマトリックス A^{-1} によって上書きされる。

info INTEGER
info = 0 の場合は、正常に終了する。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。
info = *i* の場合は、*D* の *i* 番目の対角成分がゼロで、*D* が特異になるため、マトリックスの反転を完了できない。

アプリケーション・ノート

算出された逆行列 *X* は、以下の誤差範囲を満たす。

uplo = 'U' の場合

$$|DU^TP^TXPU - I| \le c(n)\varepsilon\,(|D||U^T||P^T||X||P||U| + |D||D^{-1}|)$$

uplo = 'L' の場合

$$|DL^TP^TXPL - I| \le c(n)\varepsilon\,(|D||L^T||P^T||X||P||L| + |D||D^{-1}|)$$

$c(n)$ は n の適度な 1 次関数で、ε はマシンの精度である。*I* は元のマトリックスを示す。

浮動小数点演算の合計回数は、実数型の場合は約 $(2/3)n^3$ で、複素数型の場合は約 $(8/3)n^3$ になる。

?hetri

複素エルミート・マトリックスの
逆行列を計算する。

```
call chetri (uplo, n, a, lda, ipiv, work, info)
call zhetri (uplo, n, a, lda, ipiv, work, info)
```

説明

このルーチンは、複素エルミート・マトリックス A の逆行列 (A^{-1}) を計算する。
このルーチンを呼び出す前に、?hetrf を呼び出して、A を因子分解する。

入力パラメータ

uplo CHARACTER*1。'U' または 'L' でなければならない。
入力マトリックス A の因子分解の方法を指定する。
uplo = 'U' の場合は、配列 *a* には、Bunch-Kaufman 因子分解 $A = PUDU^HP^T$ を格納する。
uplo = 'L' の場合は、配列 *a* には、Bunch-Kaufman 因子分解 $A = PLDL^HP^T$ を格納する。

n INTEGER。マトリックス A の次数 ($n \geq 0$)

a, work COMPLEX (chetri の場合)
DOUBLE COMPLEX (zhetri の場合)
配列 :
*a(lda, *)* には、?hetrf によって返される、マトリックス A の因子分解を格納する。
a の第 2 次元は、少なくとも max(1, *n*) でなければならない。
work(*) は、ワークスペース配列である。
work の次元は、少なくとも max(1, *n*) でなければならない。

lda INTEGER。*a* の第 1 次元。*lda* ≥ max(1, *n*)

ipiv INTEGER
配列 DIMENSION は少なくとも max(1, *n*)
?hetrf によって返される *ipiv* 配列。

出力パラメータ

a	*n* × *n* のマトリックス A^{-1} によって上書きされる。
info	INTEGER *info* = 0 の場合は、正常に終了する。 *info* = -*i* の場合は、*i* 番目のパラメータの値が不正である。 *info* = *i* の場合は、*D* の *i* 番目の対角成分がゼロで、*D* が特異になるため、マトリックスの反転を完了できない。

アプリケーション・ノート

算出された逆行列 *X* は、以下の誤差範囲を満たす。

uplo = 'U' の場合

$$|DU^H P^T XPU - I| \le c(n)\varepsilon\,(|D||U^H||P^T||X||P||U| + |D||D^{-1}|)$$

uplo = 'L' の場合

$$|DL^H P^T XPL - I| \le c(n)\varepsilon\,(|D||L^H||P^T||X||P||L| + |D||D^{-1}|)$$

$c(n)$ は *n* の適度な 1 次関数で、ε はマシンの精度である。*I* は元のマトリックスを示す。

浮動小数点演算の合計回数は、実数型の場合は約 $(2/3)n^3$ で、複素数型の場合は約 $(8/3)n^3$ になる。

このルーチンに対応する実数型は、?sytri である。

?sptri

パックド格納形式の対称マトリックスの
逆行列を計算する。

```
call ssptri (uplo, n, ap, ipiv, work, info)
call dsptri (uplo, n, ap, ipiv, work, info)
call csptri (uplo, n, ap, ipiv, work, info)
call zsptri (uplo, n, ap, ipiv, work, info)
```

説明

このルーチンは、パックド格納形式の対称マトリックス A の逆行列 (A^{-1}) を計算する。
このルーチンを呼び出す前に、?sptrf を呼び出して、A を因子分解する。

入力パラメータ

uplo CHARACTER*1。'U' または 'L' でなければならない。
入力マトリックス A の因子分解の方法を指定する。
uplo = 'U' の場合は、配列 *ap* には、Bunch-Kaufman 因子分解 $A = PUDU^TP^T$ を格納する。
uplo = 'L' の場合は、配列 *ap* には、Bunch-Kaufman 因子分解 $A = PLDL^TP^T$ を格納する。

n INTEGER。マトリックス A の次数 ($n \geq 0$)

ap, work REAL (ssptri の場合)
DOUBLE PRECISION (dsptri の場合)
COMPLEX (csptri の場合)
DOUBLE COMPLEX (zsptri の場合)
配列 :
ap(*) には、?sptrf によって返される、マトリックス A の因子分解を格納する。
ap の次元は、少なくとも max(1, *n* (*n*+1) / 2) でなければならない。
work(*) は、ワークスペース配列である。
work の次元は、少なくとも max(1, *n*) でなければならない。

ipiv INTEGER
配列 DIMENSION は少なくとも max(1, *n*)
?sptrf によって返される *ipiv* 配列。

出力パラメータ

ap *ap* パックド形式の $n \times n$ のマトリックス A^{-1} によって上書きされる。

info INTEGER
info = 0 の場合は、正常に終了する。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。
info = *i* の場合は、*D* の *i* 番目の対角成分がゼロで、*D* が特異になるため、マトリックスの反転を完了できない。

アプリケーション・ノート

算出された逆行列 *X* は、以下の誤差範囲を満たす。

uplo = 'U' の場合

$$|DU^TP^TXPU - I| \le c(n)\varepsilon\,(|D||U^T||P^T||X||P||U| + |D||D^{-1}|)$$

uplo = 'L' の場合

$$|DL^TP^TXPL - I| \le c(n)\varepsilon\,(|D||L^T||P^T||X||P||L| + |D||D^{-1}|)$$

c(*n*) は *n* の適度な 1 次関数で、ε はマシンの精度である。*I* は元のマトリックスを示す。

浮動小数点演算の合計回数は、実数型の場合は約 $(2/3)n^3$ で、複素数型の場合は約 $(8/3)n^3$ になる。

?hptri

パックド格納形式の複素エルミート・
マトリックスの逆行列を計算する。

```
call chptri (uplo, n, ap, ipiv, work, info)
call zhptri (uplo, n, ap, ipiv, work, info)
```

説明

このルーチンは、パックド格納形式の複素エルミート・マトリックス A の逆行列 (A^{-1}) を計算する。
このルーチンを呼び出す前に、?hptrf を呼び出して、A を因子分解する。

入力パラメータ

uplo CHARACTER*1。'U' または 'L' でなければならない。
入力マトリックス A の因子分解の方法を指定する。
uplo = 'U' の場合は、配列 *ap* には、パックド形式の Bunch-Kaufman 因子分解 $A = PUDU^HP^T$ を格納する。
uplo = 'L' の場合は、配列 *ap* には、パックド形式の Bunch-Kaufman 因子分解 $A = PLDL^HP^T$ を格納する。

n INTEGER。マトリックス A の次数 ($n \geq 0$)

ap COMPLEX (chptri の場合)
DOUBLE COMPLEX (zhptri の場合)
配列 :
ap(*) には、?hptrf によって返される、マトリックス A の因子分解を格納する。
ap の次元は、少なくとも max(1, $n(n+1)/2$) でなければならない。
work(*) は、ワークスペース配列である。
work の次元は、少なくとも max(1, n) でなければならない。

ipiv INTEGER
配列 DIMENSION は少なくとも max(1, n)
?hptrf によって返される *ipiv* 配列。

出力パラメータ

ap	*n* × *n* のマトリックス A^{-1} によって上書きされる。
info	INTEGER *info* = 0 の場合は、正常に終了する。 *info* = -*i* の場合は、*i* 番目のパラメータの値が不正である。 *info* = *i* の場合は、*D* の *i* 番目の対角成分がゼロで、*D* が特異になるため、マトリックスの反転を完了できない。

アプリケーション・ノート

算出された逆行列 *X* は、以下の誤差範囲を満たす。

uplo = 'U' の場合

$$|DU^H P^T XPU - I| \le c(n)\varepsilon\,(|D||U^H||P^T||X||P||U| + |D||D^{-1}|)$$

uplo = 'L' の場合

$$|DL^H P^T XPL - I| \le c(n)\varepsilon\,(|D||L^H||P^T||X||P||L| + |D||D^{-1}|)$$

$c(n)$ は *n* の適度な 1 次関数で、ε はマシンの精度である。*I* は元のマトリックスを示す。

浮動小数点演算の合計回数は、実数型の場合は約 $(2/3)n^3$ で、複素数型の場合は約 $(8/3)n^3$ になる。

このルーチンに対応する実数型は、?sptri である。

?trtri

三角マトリックスの逆行列を計算する。

```
call strtri (uplo, diag, n, a, lda, info)
call dtrtri (uplo, diag, n, a, lda, info)
call ctrtri (uplo, diag, n, a, lda, info)
call ztrtri (uplo, diag, n, a, lda, info)
```

説明

このルーチンは、三角マトリックス A の逆行列 (A^{-1}) を計算する。

入力パラメータ

uplo CHARACTER*1。'U' または 'L' でなければならない。
A が上三角マトリックスか、下三角マトリックスかを指定する。
uplo = 'U' の場合は、A は上三角マトリックスである。
uplo = 'L' の場合は、A は下三角マトリックスである。

diag CHARACTER*1。'N' または 'U' でなければならない。
diag = 'N' の場合は、A は単位三角マトリックスではない。
diag = 'U' の場合は、A は単位三角マトリックスである。A の対角成分は 1 と見なされ、配列 *a* 内で参照されない。

n INTEGER。マトリックス A の次数 ($n \geq 0$)

a REAL (strtri の場合)
DOUBLE PRECISION (dtrtri の場合)
COMPLEX (ctrtri の場合)
DOUBLE COMPLEX (ztrtri の場合)
配列 : DIMENSION (*lda*, *)
マトリックス A を格納する。
a の第 2 次元は、少なくとも max(1, *n*) でなければならない。

lda INTEGER。*a* の第 1 次元。*lda* ≥ max(1, *n*)

出力パラメータ

a	*n* × *n* のマトリックス A^{-1} によって上書きされる。
info	INTEGER *info* = 0 の場合は、正常に終了する。 *info* = -*i* の場合は、*i* 番目のパラメータの値が不正である。 *info* = *i* の場合は、*A* の *i* 番目の対角成分がゼロで、*A* が特異になるため、反転を完了できない。

アプリケーション・ノート

算出された逆行列 *X* は、以下の誤差範囲を満たす。

$$|XA - I| \le c(n)\varepsilon\,|X||A|$$

$$\left|X - A^{-1}\right| \le c(n)\varepsilon\,\left|A^{-1}\right||A||X|$$

$c(n)$ は *n* の適度な 1 次関数で、ε はマシンの精度である。
I は元のマトリックスを示す。

浮動小数点演算の合計回数は、実数型の場合は約 $(1/3)n^3$ で、複素数型の場合は約 $(4/3)n^3$ になる。

?tptri

パックド格納形式の三角マトリックスの逆行列を計算する。

```
call stptri (uplo, diag, n, ap, info)
call dtptri (uplo, diag, n, ap, info)
call ctptri (uplo, diag, n, ap, info)
call ztptri (uplo, diag, n, ap, info)
```

説明

このルーチンは、パックド格納形式の三角マトリックス *A* の逆行列 (A^{-1}) を計算する。

入力パラメータ

uplo CHARACTER*1。'U' または 'L' でなければならない。
A が上三角マトリックスか、下三角マトリックスかを指定する。
uplo = 'U' の場合は、A は上三角マトリックスである。
uplo = 'L' の場合は、A は下三角マトリックスである。

diag CHARACTER*1。'N' または 'U' でなければならない。
diag = 'N' の場合は、A は単位三角マトリックスではない。
diag = 'U' の場合は、A は単位三角マトリックスである。A の対角成分は 1 と見なされ、配列 *ap* 内で参照されない。

n INTEGER。マトリックス A の次数 ($n \geq 0$)

ap REAL (stptri の場合)
DOUBLE PRECISION (dtptri の場合)
COMPLEX (ctptri の場合)
DOUBLE COMPLEX (ztptri の場合)
配列 : DIMENSION は少なくとも max(1, $n(n+1)/2$)
パックド形式の三角マトリックス A を格納する。

出力パラメータ

ap パックド形式の $n \times n$ のマトリックス A^{-1} によって上書きされる。

info INTEGER
info = 0 の場合は、正常に終了する。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。
info = *i* の場合は、A の *i* 番目の対角成分がゼロで、A が特異になるため、反転を完了できない。

アプリケーション・ノート

算出された逆行列 X は、以下の誤差範囲を満たす。

$$|XA - I| \leq c(n)\varepsilon\,|X||A|$$

$$|X - A^{-1}| \leq c(n)\varepsilon\,|A^{-1}||A||X|$$

$c(n)$ は n の適度な 1 次関数で、ε はマシンの精度である。I は元のマトリックスを示す。

浮動小数点演算の合計回数は、実数型の場合は約 $(1/3)n^3$ で、複素数型の場合は約 $(4/3)n^3$ になる。

マトリックスの平衡化

この節では、マトリックスの平衡化に必要なスケール係数の計算に使用されるルーチンについて説明する。ただし、これらのルーチンがマトリックスを実際にスケーリングするわけではない。

?geequ

マトリックスを平衡化して、条件数を小さくするための行と列のスケール係数を計算する。

```
call sgeequ (m, n, a, lda, r, c, rowcnd, colcnd, amax, info)
call dgeequ (m, n, a, lda, r, c, rowcnd, colcnd, amax, info)
call cgeequ (m, n, a, lda, r, c, rowcnd, colcnd, amax, info)
call zgeequ (m, n, a, lda, r, c, rowcnd, colcnd, amax, info)
```

説明

このルーチンは、*m* × *n* マトリックス *A* を平衡化して条件数を小さくするための、行と列のスケール係数を計算する。出力配列 *r* に行のスケール係数を返し、配列 *c* に列のスケール係数を返す。これらの係数は、成分 $b_{ij}=r(i)*a_{ij}*c(j)$ を持つマトリックス B の各行と各列の最大の成分の絶対値が 1 になるように選択される。

入力パラメータ

m	INTEGER 。マトリックス A の行数。*m* ≥0
n	INTEGER。マトリックス A の列数。*n* ≥0
a	REAL (sgeequ の場合) DOUBLE PRECISION (dgeequ の場合) COMPLEX (cgeequ の場合) DOUBLE COMPLEX (zgeequ の場合)

配列 : DIMENSION は (lda, *)
平衡化係数を計算する $m \times n$ マトリックス A を格納する。
a の第 2 次元は、少なくとも max(1, n) でなければならない。

lda INTEGER。a のリーディング・ディメンジョン。
$lda \geq \max(1, m)$

出力パラメータ

r, c REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
配列 : $r(m)$, $c(n)$
$info = 0$ または $info > m$ の場合は、配列 r にマトリックス A の行のスケール係数が格納される。
$info = 0$ の場合は、配列 c にマトリックス A の列のスケール係数が格納される。

rowcnd REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
$info = 0$ または $info > m$ の場合は、rowcnd に、最小の r(i) を最大の r(i) で割った値が格納される。

colcnd REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
$info = 0$ の場合は、colcnd に、最小の c(i) を最大の c(i) で割った値が格納される。

amax REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
マトリックス A の最大の成分の絶対値。

info INTEGER
$info = 0$ の場合は、ルーチンの実行は正常に終了した。
$info = -i$ の場合は、i 番目のパラメータの値が不正である。
$info = i$、かつ
$i \leq m$ の場合は、A の i 番目の行が完全に 0 である。
$i > m$ の場合は、A の $(i-m)$ 番目の列が完全に 0 である。

アプリケーション・ノート

r と *c* のすべての成分は、SMLNUM = 最小の安全な数値と BIGNUM = 最大の安全な数値の範囲内に制限される。これらのスケール係数を使用しても、*A* の条件数が小さくなることは保証されないが、実際には有効に機能する。

rowcnd ≥ 0.1 であり、*amax* が大きすぎる値でも小さすぎる値でもない場合は、*r* によるスケーリングは不要である。*colcnd* ≥ 0.1 の場合は、*c* によるスケーリングは不要である。

amax の値がオーバーフローまたはアンダーフローに非常に近い場合は、マトリックス *A* をスケーリングする必要がある。

?gbequ

バンド・マトリックスを平衡化して
条件数を小さくするための、行と列の
スケール係数を計算する。

```
call sgbequ (m, n, kl, ku, ab, ldab, r, c, rowcnd, colcnd,amax,info)
call dgbequ (m, n, kl, ku, ab, ldab, r, c, rowcnd, colcnd,amax,info)
call cgbequ (m, n, kl, ku, ab, ldab, r, c, rowcnd, colcnd,amax,info)
call zgbequ (m, n, kl, ku, ab, ldab, r, c, rowcnd, colcnd,amax,info)
```

説明

このルーチンは、$m \times n$ のバンド・マトリックス A を平衡化して、条件数を小さくするための、行と列のスケール係数を計算する。出力配列 *r* に行のスケール係数を返し、配列 *c* に列のスケール係数を返す。これらの係数は、成分 $b_{ij}=r(i)*a_{ij}*c(j)$ を持つマトリックス B の各行と各列の最大の成分の絶対値が 1 になるように選択される。

入力パラメータ

m	INTEGER。マトリックス A の行数。$m \geq 0$
n	INTEGER。マトリックス A の列数。$n \geq 0$
kl	INTEGER。A のバンド内の劣対角成分の数。($kl \geq 0$)
ku	INTEGER。A のバンド内の優対角成分の数。($ku \geq 0$)
ab	REAL (sgbequ の場合) DOUBLE PRECISION (dgbequ の場合) COMPLEX (cgbequ の場合) DOUBLE COMPLEX (zgbequ の場合) 配列 DIMENSION は DIMENSION 行 1 ～ $kl + ku + 1$ に格納される、元のバンド・マトリックス A を格納する。

ab の第 2 次元は、少なくとも max(1, n) でなければならない。

ldab　INTEGER。ab のリーディング・ディメンジョン、ldab ≥ kl+ku+1

出力パラメータ

r, c　REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
配列 : r(m), c(n)
info = 0 または info > m の場合は、配列 r にマトリックス A の行のスケール係数が格納される。
info = 0 の場合は、配列 c にマトリックス A の列のスケール係数が格納される。

rowcnd　REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
info = 0 または info > m の場合は、rowcnd に、最小の r(i) を最大の r(i) で割った値が格納される。

colcnd　REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
info = 0 の場合は、colcnd に、最小の c(i) を最大の c(i) で割った値が格納される。

amax　REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
マトリックス A の最大の成分の絶対値。

info　INTEGER
info = 0 の場合は、ルーチンの実行は正常に終了した。
info = -i の場合は、i 番目のパラメータの値が不正である。
info = i、かつ
i ≤ m の場合は、A の i 番目の行が完全に 0 である。
i > m の場合は、A の (i-m) 番目の列が完全に 0 である。

アプリケーション・ノート

r と c のすべての成分は、SMLNUM = 最小の安全な数値と BIGNUM = 最大の安全な数値の範囲内に制限される。これらのスケール係数を使用しても、A の条件数が小さくなることは保証されないが、実際には有効に機能する。

rowcnd ≥ 0.1 であり、amax が大きすぎる値でも小さすぎる値でもない場合は、r によるスケーリングは不要である。colcnd ≥ 0.1 の場合は、c によるスケーリングは不要である。

amax の値がオーバーフローまたはアンダーフローに非常に近い場合は、マトリックス *A* をスケーリングする必要がある。

?poequ

正値の対称 (エルミート) マトリックスを
平衡化して条件数を小さくするための、
行と列のスケール係数を計算する。

```
call spoequ (n, a, lda, s, scond, amax, info)
call dpoequ (n, a, lda, s, scond, amax, info)
call cpoequ (n, a, lda, s, scond, amax, info)
call zpoequ (n, a, lda, s, scond, amax, info)
```

説明

このルーチンは、正値の対称 (エルミート) マトリックス *A* を平衡化して (2- ノルムに関して) 条件数を小さくするための、行と列のスケール係数を計算する。出力配列 s に、次のように計算されたスケール係数を返す。

$$s(i) = \frac{1}{\sqrt{a_{i,i}}}$$

これらの係数は、成分 b_{ij}=s(i)*a_{ij}*s(j) を持つスケーリング後のマトリックス B の対角成分が 1 に等しくなるように選択される。

s をこのように選択すると、*B* の条件数は、すべての可能な対角スケーリングで、可能な最小の条件数に係数 n を掛けた値の範囲内に収まる。

入力パラメータ

n　　INTEGER。マトリックス A の次数。n ≥0

a
REAL (spoequ の場合)
DOUBLE PRECISION (dpoequ の場合)
COMPLEX (cpoequ の場合)
DOUBLE COMPLEX (zpoequ の場合)
配列 : DIMENSION (*lda*, *)
スケール係数を計算する、*n* × *n* の正値対称またはエルミート・マトリックス *A* を格納する。*A* の対角成分だけが参照される。
a の第 2 次元は、少なくとも max(1, *n*) でなければならない。

lda
INTEGER。*a* のリーディング・ディメンジョン。
lda ≥ max(1, *m*)

出力パラメータ

s
REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
配列 DIMENSION は (*n*)。
info = 0 の場合は、配列 *s* に *A* のスケール係数が格納される。

scond
REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
info = 0 の場合は、*scond* に、最小の *s*(i) を最大の *s*(i) で割った値が格納される。

amax
REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
マトリックス *A* の最大の成分の絶対値。

info
INTEGER
info = 0 の場合は、ルーチンの実行は正常に終了した。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。
info = *i* の場合は、*A* の *i* 番目の対角成分が正の値でない。

アプリケーション・ノート

scond ≥ 0.1 であり、*amax* が大きすぎる値でも小さすぎる値でもない場合は、*s* によるスケーリングは不要である。

amax の値がオーバーフローまたはアンダーフローに非常に近い場合は、マトリックス *A* をスケーリングする必要がある。

?ppequ

パックド格納形式の正値の対称
(エルミート) マトリックスを平衡化して、
条件数を小さくするための行と列の
スケール係数を計算する。

```
call sppequ (uplo, n, ap, s, scond, amax, info)
call dppequ (uplo, n, ap, s, scond, amax, info)
call cppequ (uplo, n, ap, s, scond, amax, info)
call zppequ (uplo, n, ap, s, scond, amax, info)
```

説明

このルーチンは、パックド格納形式の正値の対称 (エルミート) マトリックス A を平衡化して (2- ノルムに関して) 条件数を小さくするための、行と列のスケール係数を計算する。出力配列 *s* に、次のように計算されたスケール係数を返す。

$$s(i) = \frac{1}{\sqrt{a_{i,i}}}$$

これらの係数は、成分 $b_{ij}=s(i)*a_{ij}*s(j)$ を持つスケーリング後のマトリックス B の対角成分が 1 に等しくなるように選択される。

s をこのように選択すると、B の条件数は、すべての可能な対角スケーリングで、可能な最小の条件数に係数 *n* を掛けた値の範囲内に収まる。

入力パラメータ

uplo CHARACTER*1。'U' または 'L' でなければならない。
A の上三角部分と下三角部分のどちらを配列 *ap* にパックするかを指定する。
uplo = 'U' の場合は、配列 *ap* にはマトリックス A の上三角部分が格納される。
uplo = 'L' の場合は、配列 *ap* にはマトリックス A の下三角部分が格納される。

n INTEGER。マトリックス *A* の次数 ($n \geq 0$)。

ap REAL (sppequ の場合)
DOUBLE PRECISION (dppequ の場合)
COMPLEX (cppequ の場合)
DOUBLE COMPLEX (zppequ の場合)
配列 DIMENSION は少なくとも max(1, *n*(*n*+1)/2)
配列 *ap* には、(*uplo* で指定された) マトリックス *A* の上三角部分または下三角部分が**パックド形式**で*格納*される (「「マトリックスの格納形式」」を参照)。

出力パラメータ

s REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
配列 DIMENSION は (*n*)
info = 0 の場合は、配列 *s* に *A* のスケール係数が格納される。

scond REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
info = 0 の場合は、*scond* に、最小の *s*(i) を最大の *s*(i) で割った値が格納される。

amax REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
マトリックス *A* の最大の成分の絶対値。

info INTEGER
info = 0 の場合は、ルーチンの実行は正常に終了した。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。
info = *i* の場合は、*A* の *i* 番目の対角成分が正の値でない。

アプリケーション・ノート

scond ≥ 0.1 であり、*amax* が大きすぎる値でも小さすぎる値でもない場合は、*s* によるスケーリングは不要である。

amax の値がオーバーフローまたはアンダーフローに非常に近い場合は、マトリックス *A* をスケーリングする必要がある。

?pbequ

正値の対称 (エルミート) バンド・
マトリックスを平衡化して、条件数を
小さくするための行と列のスケール
係数を計算する。

```
call spbequ (uplo, n, kd, ab, ldab, s, scond, amax, info)
call dpbequ (uplo, n, kd, ab, ldab, s, scond, amax, info)
call cpbequ (uplo, n, kd, ab, ldab, s, scond, amax, info)
call zpbequ (uplo, n, kd, ab, ldab, s, scond, amax, info)
```

説明

このルーチンは、バンド格納形式の正値の対称 (エルミート) マトリックス A を平衡化して (2- ノルムに関して) 条件数を小さくするための、行と列のスケール係数を計算する。出力配列 *s* に、次のように計算されたスケール係数を返す。

$$s(i) = \frac{1}{\sqrt{a_{i,i}}}$$

これらの係数は、成分 $b_{ij}=s(i)*a_{ij}*s(j)$ を持つスケーリング後のマトリックス B の対角成分が 1 に等しくなるように選択される。
s をこのように選択すると、B の条件数は、すべての可能な対角スケーリングで、可能な最小の条件数に係数 *n* を掛けた値の範囲内に収まる。

入力パラメータ

uplo CHARACTER*1。'U' または 'L' でなければならない。
A の上三角部分と下三角部分のどちらを配列 *ab* にパックするかを指定する。
uplo = 'U' の場合は、配列 *ab* にはマトリックス A の上三角部分が格納される。
uplo = 'L' の場合は、配列 *ab* にはマトリックス A の下三角部分が格納される。

n INTEGER。マトリックス A の次数 ($n \geq 0$)。

kd INTEGER。マトリックス A の優対角成分または劣対角成分の数 ($kd \geq 0$)。

ab
REAL (spbequ の場合)
DOUBLE PRECISION (dpbequ の場合)
COMPLEX (cpbequ の場合)
DOUBLE COMPLEX (zpbequ の場合)
配列 DIMENSION は (*ldab*, *)。
配列 *ap* には、(*uplo* で指定された) マトリックス *A* の上三角部分または下三角部分が**バンド形式**で*格納*される (「「マトリックスの格納形式」」 を参照)。
ab の第 2 次元は、は少なくとも max(1, *n*) でなければならない。

ldab
INTEGER。*ab* のリーディング・ディメンジョン。
(*ldab* ≥ *kd* +1)

出力パラメータ

s
REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
配列 DIMENSION は (*n*)。
info = 0 の場合は、配列 *s* に *A* のスケール係数が格納される。

scond
REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
info = 0 の場合は、*scond* に、最小の *s*(i) を最大の *s*(i) で割った値が格納される。

amax
REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
マトリックス *A* の最大の成分の絶対値。

info
INTEGER
info = 0 の場合は、ルーチンの実行は正常に終了した。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。
info = *i* の場合は、*A* の *i* 番目の対角成分が正の値でない。

アプリケーション・ノート

scond ≥ 0.1 であり、*amax* が大きすぎる値でも小さすぎる値でもない場合は、*s* によるスケーリングは不要である。

amax の値がオーバーフローまたはアンダーフローに非常に近い場合は、マトリックス *A* をスケーリングする必要がある。

ドライバ・ルーチン

表 4-3 に、実数または複素マトリックスを係数行列とする連立 1 次方程式を解くための LAPACK ドライバ・ルーチンを示す。

表 4-3　連立 1 次方程式の解の算出用のドライバ・ルーチン

マトリックスのタイプ (格納形式)	簡易ドライバ	高度ドライバ
一般	?gesv	?gesvx
一般バンド	?gbsv	?gbsvx
一般三重対角	?gtsv	?gtsvx
正値対称 / エルミート	?posv	?posvx
正値対称 / エルミート (パックド格納形式)	?ppsv	?ppsvx
正値対称 / エルミート (バンド形式)	?pbsv	?pbsvx
正値対称 / エルミート (三重対角形式)	?ptsv	?ptsvx
不定値対称 / エルミート	?sysv/?hesv	?sysvx/?hesvx
不定値対称 / エルミート (パックド格納形式)	?spsv/?hpsv	?spsvx/?hpsvx
複素対称	?sysv	?sysvx
複素対称 (パックド格納形式)	?spsv	?spsvx

表中の **?** は、**s** (単精度実数)、**d** (倍精度実数)、**c** (単精度複素数)、または **z** (倍精度複素数) を示す。

?gesv

正方マトリックス A を係数行列とする、
複数の右辺を持つ連立 1 次方程式の解を
計算する。

```
call sgesv (n, nrhs, a, lda, ipiv, b, ldb, info)
call dgesv (n, nrhs, a, lda, ipiv, b, ldb, info)
call cgesv (n, nrhs, a, lda, ipiv, b, ldb, info)
call zgesv (n, nrhs, a, lda, ipiv, b, ldb, info)
```

説明

このルーチンは、連立 1 次方程式 $AX = B$ を X について解く。A は $n \times n$ マトリックス、マトリックス B の列は個々の右辺、X の列はそれに対応する解である。

部分的なピボット (要) 演算と行の交換による LU 分解を使用して、A を $A = P\,L\,U$ として因子分解する。P は置換マトリックス、L は単位下三角マトリックス、U は上三角マトリックスである。次に、A の因子分解された形式を使用して、連立方程式 $AX = B$ を解く。

入力パラメータ

n	INTEGER。A の次数。B の行の数 ($n \geq 0$)。
nrhs	INTEGER。右辺の数。B の列の数 ($nrhs \geq 0$)。
a, b	REAL (sgesv の場合) DOUBLE PRECISION (dgesv の場合) COMPLEX (cgesv の場合) DOUBLE COMPLEX (zgesv の場合) 配列 : *a(lda, *)*, *b(ldb, *)* 配列 *a* には、マトリックス A が格納される。 配列 *b* には、マトリックス B が格納される。このマトリックスの列は、連立方程式の右辺である。 *a* の第 2 次元は、少なくとも max(1, *n*) でなければならない。*b* の第 2 次元は、少なくとも max(1, *nrhs*) でなければならない。

lda	INTEGER。*a* の第 1 次元。*lda* ≥ max(1, *n*)
ldb	INTEGER。*b* の第 1 次元。*ldb* ≥ max(1, *n*)

出力パラメータ

a	$A = P L U$ の因子分解で得られた係数 L と U によって上書きされる。L の単位対角成分は格納されない。
b	解のマトリックス X によって上書きされる。
ipiv	INTEGER 配列 DIMENSION は少なくとも max(1, *n*) 置換マトリックス P を定義するピボット (要) のインデックス。マトリックスの行 i は、行 *ipiv*(i) と交換されている。
info	INTEGER。*info* = 0 の場合は、ルーチンの実行は正常に終了した。 *info* = -*i* の場合は、*i* 番目のパラメータの値が不正である。 *info* = *i* の場合は、U (*i*, *i*) が完全に 0 である。因子分解は完了したが、係数 U が完全に特異であるため、解を計算できなかった。

?gesvx

正方マトリックス A を係数行列とする、
複数の右辺を持つ連立 1 次方程式の解を
計算し、解の誤差範囲を示す。

```
call sgesvx (fact, trans, n, nrhs, a, lda, af, ldaf, ipiv, equed, r,
             c, b, ldb, x, ldx, rcond, ferr, berr, work, iwork, info)
call dgesvx (fact, trans, n, nrhs, a, lda, af, ldaf, ipiv, equed, r,
             c, b, ldb, x, ldx, rcond, ferr, berr, work, iwork, info)
call cgesvx (fact, trans, n, nrhs, a, lda, af, ldaf, ipiv, equed, r,
             c, b, ldb, x, ldx, rcond, ferr, berr, work, rwork, info)
call zgesvx (fact, trans, n, nrhs, a, lda, af, ldaf, ipiv, equed, r,
             c, b, ldb, x, ldx, rcond, ferr, berr, work, rwork, info)
```

説明

このルーチンは、*LU* 因子分解を使用して、実数または複素連立 1 次方程式 $AX = B$ の解を計算する。A は $n \times n$ マトリックス、マトリックス B の列は個々の右辺、X の列はそれに対応する解である。

このルーチンは、解の誤差範囲と条件数の推定値も示す。

ルーチン ?gesvx は、以下の手順を実行する。

1. *fact* = 'E' の場合は、連立方程式を平衡化するための実スケール係数 *r* と *c* を計算する。

trans = 'N': $\mathrm{diag}(r)*A*\mathrm{diag}(c)*\mathrm{diag}(c)^{-1}*X = \mathrm{diag}(r)*B$

trans = 'T': $(\mathrm{diag}(r)*A*\mathrm{diag}(c))^{T}*\mathrm{diag}(r)^{-1}*X = \mathrm{diag}(c)*B$

trans = 'C': $(\mathrm{diag}(r)*A*\mathrm{diag}(c))^{H}*\mathrm{diag}(r)^{-1}*X = \mathrm{diag}(c)*B$

連立方程式が平衡化されるかどうかは、マトリックス A のスケーリングによって決まる。平衡化が行われる場合は、A は $\mathrm{diag}(r)*A*\mathrm{diag}(c)$ によって上書きされ、B は $\mathrm{diag}(r)*B$ (*trans*='N' の場合) または $\mathrm{diag}(c)*B$ (*trans* = 'T' または 'C' の場合) によって上書きされる。

2. *fact* = 'N' または 'E' の場合は、*LU* 分解を使用して、(*fact* = 'E' の場合は平衡化の後に) マトリックス A を $A = P\,L\,U$ として因子分解する。P は置換マトリックス、L は単位下三角マトリックス、U は上三角マトリックスである。

3. $U_{i,i} = 0$ の場合、つまり U が完全に特異である場合は、ルーチンは *info* = i を返す。それ以外の場合は、A の因子分解された形式を使用して、マトリックス A の条件数を推定する。条件数の逆数がマシンの精度より小さい場合は、警告として *info* = n + 1 を返すが、後で説明するように、ルーチンは続行され、X について解を算出し、誤差範囲を計算する。

4. A の因子分解された形式を使用して、連立方程式を X について解く。

5. 計算された解のマトリックスの精度を繰り返し改善し、解の誤差範囲と後退誤差の推定値を計算する。

6. 平衡化が行われている場合は、平衡化が行われる前の元の連立方程式を解くために、diag(*c*) (*trans* = 'N' の場合) または diag(*r*) (*trans* = 'T' または 'C' の場合) によって、マトリックス X を事前に乗算する。

入力パラメータ

fact CHARACTER*1。'F'、'N'、または 'E' でなければならない。

ルーチンの開始時にマトリックス A の因子分解された形式が与えられるかどうか、与えられない場合は、マトリックス A を因子分解する前に A を平衡化するかどうかを指定する。

fact = 'F' の場合は、開始時に、*af* と *ipiv* に A の因子分解された形式が格納される。*equed* が 'N' でない場合は、マトリックス A は、*r* と *c* で指定されたスケール係数によって平衡化されている。
a、*af*、*ipiv* は変更されない。

fact = 'N' の場合は、マトリックス A を *af* にコピーし、因子分解を実行する。
fact = 'E' の場合は、必要に応じてマトリックス A を平衡化してから、*af* にコピーし、因子分解を実行する。

trans CHARACTER*1。'N'、'T'、または 'C' でなければならない。

連立方程式の形式を指定する。

trans = 'N' の場合は、連立方程式の形式は $A\,X = B$ (転置なし) である。
trans = 'T' の場合は、連立方程式の形式は $A^{\mathrm{T}}\,X = B$ (転置) である。
trans = 'C' の場合は、連立方程式の形式は $A^{\mathrm{H}}\,X = B$ (共役転置) である。

n INTEGER。1 次方程式の数。マトリックス A の次数 ($n \geq 0$)。

nrhs INTEGER。右辺の数。マトリックス B と X の列の数 ($nrhs \geq 0$)。

a, af, b, work REAL (sgesvx の場合)
DOUBLE PRECISION (dgesvx の場合)
COMPLEX (cgesvx の場合)
DOUBLE COMPLEX (zgesvx の場合)
配列 : *a(lda, *)*, *af(ldaf, *)*, *b(ldb, *)*, *work(*)*
配列 *a* には、マトリックス A が格納される。
fact = 'F' で、*equed* が 'N' でない場合は、A は、*r* または *c*、あるいはその両方のスケール係数によって平衡化されている。*a* の第 2 次元は、少なくとも max(1, *n*) でなければならない。

配列 *af* は、*fact* = 'F' の場合は入力引数になる。この配列には、マトリックス *A* の因子分解された形式、すなわち、?getrf による因子分解 *A* = *P L U* で得られた係数 *L* と *U* が格納される。*equed* が 'N' でない場合は、*af* は、平衡化されたマトリックス *A* の因子分解された形式になる。*af* の第 2 次元は、少なくとも max(1, *n*) でなければならない。
配列 *b* には、マトリックス *B* が格納される。このマトリックスの列は、連立方程式の右辺である。*b* の第 2 次元は、少なくとも max(1, *nrhs*) でなければならない。

work(*) はワークスペース配列である。
work の次元は、実数型の場合は少なくとも max(1, 4**n*)、複素数型の場合は少なくとも max(1, 2**n*) でなければならない。

lda INTEGER。*a* の第 1 次元。*lda* ≥ max(1, *n*)

ldaf INTEGER。*af* の第 1 次元。*ldaf* ≥ max(1, *n*)

ldb INTEGER。*b* の第 1 次元。*ldb* ≥ max(1, *n*)

ipiv INTEGER
配列 DIMENSION は少なくとも max(1, *n*)。
配列 *ipiv* は、*fact* = 'F' の場合は入力引数になる。この配列には、?getrf による因子分解 *A* = *P L U* で得られたピボット (要) のインデックスが格納される。マトリックスの行 i は、行 *ipiv*(i) と交換されている。

equed CHARACTER*1。'N'、'R'、'C'、または 'B' でなければならない。
equed は、*fact* = 'F' の場合は入力引数になる。この引数は、実行された平衡化の形式を指定する。
equed = 'N' の場合は、平衡化は行われていない (*fact* = 'N' の場合は常に真)。
equed = 'R' の場合は、行の平衡化が行われ、*A* は diag(*r*) によって事前に乗算されている。
equed = 'C' の場合は、列の平衡化が行われ、*A* は diag(*c*) によって事後に乗算されている。
equed = 'B' の場合は、行と列の平衡化が行われ、*A* は diag(*r*)**A**diag(*c*) で置き換えられている。

r, c
REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
配列 : *r*(*n*), *c*(*n*)
配列 *r* には *A* の行のスケール係数が格納され、配列 *c* には *A* の列のスケール係数が格納される。これらの配列は、*fact* = 'F' の場合にのみ入力引数になり、それ以外の場合は出力引数になる。
equed = 'R' または 'B' の場合は、*A* は diag(*r*) によって左辺で乗算される。*equed* = 'N' または 'C' の場合は、*r* は使用されない。
fact = 'F' で、*equed* = 'R' または 'B' の場合は、*r* の各成分は正の値でなければならない。
equed = 'C' または 'B' の場合は、*A* は diag(*c*) によって右辺で乗算される。*equed* = 'N' または 'R' の場合は、*c* は使用されない。
fact = 'F' で、*equed* = 'C' または 'B' の場合は、*c* の各成分は正の値でなければならない。

ldx
INTEGER。出力配列 *x* の第 1 次元。*ldx* ≥ max(1, *n*)

iwork
INTEGER
ワークスペース配列 DIMENSION は少なくとも max(1, *n*)。実数型でのみ使用される。

rwork
REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
ワークスペース配列 DIMENSION は少なくとも max(1, 2**n*)。複素数型でのみ使用される。

出力パラメータ

x
REAL (sgesvx の場合)
DOUBLE PRECISION (dgesvx の場合)
COMPLEX (cgesvx の場合)
DOUBLE COMPLEX (zgesvx の場合)
配列 DIMENSION は (*ldx*, *)。
info = 0 または *info* = *n*+1 の場合は、配列 *x* には、**元の**連立方程式の解のマトリックス *X* が格納される。ただし、*equed* ≠ 'N' の場合は、*A* と *B* は終了時に修正され、**平衡化された**連立方程式の解は次のようになる。
diag(*c*)$^{-1}$**X* (*trans* = 'N' で、*equed* 'C' または 'B' の場合)
diag(*r*)$^{-1}$**X* (*trans* = 'T' または 'C' で、*equed* = 'R'

	または 'B' の場合) x の第 2 次元は、少なくとも max(1, nrhs) でなければならない。
a	配列 a は、fact = 'F' または 'N' の場合、または fact = 'E' で equed = 'N' の場合は、終了時に変更されない。 equed ≠ 'N' の場合は、A は終了時に次のようにスケーリングされる。 equed = 'R': A = diag(r)*A equed = 'C': A = A*diag(c) equed = 'B': A = diag(r)*A*diag(c)
af	fact = 'N' または 'E' の場合は、af は出力引数になる。この引数は、終了時に、元のマトリックス A (fact = 'N' の場合) または平衡化されたマトリックス A (fact = 'E' の場合) の因子分解 A = P L U で得られた係数 L と U を返す。平衡化されたマトリックスの形式については、a の説明を参照のこと。
b	trans = 'N' で、equed = 'R' または 'B' の場合は、diag(r)*B によって上書きされる。 trans = 'T' で、equed = 'C' または 'B' の場合は、diag(c)*B によって上書きされる。 equed = 'N' の場合は変更されない。
r, c	これらの配列は、fact ≠ 'F' の場合は出力引数になる。 「入力引数」の項の r, c の説明を参照のこと。
rcond	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 平衡化後のマトリックス A の条件数の逆数の推定値 (平衡化が行われる場合)。推定値のアンダーフローが発生した場合は、rcond =0 に設定される。この場合は、マトリックスは有効な精度で特異になる。ただし、rcond が (有効な精度で) 1.0 より小さい場合は、条件の良くないマトリックスまたは特異なマトリックスである。
ferr, berr	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列 DIMENSION は少なくとも max(1, nrhs)。各解ベクトルの成分ごとの前進誤差と相対後退誤差を格納する。

ipiv

fact = 'N' または 'E' の場合は、*ipiv* は出力引数になる。この引数には、終了時に、元のマトリックス *A* (*fact* = 'N' の場合) または平衡化されたマトリックス *A* (*fact* = 'E' の場合) の因子分解 $A = P\,L\,U$ で得られたピボット (要) のインデックスが格納される。

equed

fact ≠ 'F' の場合は、*equed* は出力引数になる。この引数は、実行された平衡化の形式を指定する (「入力引数」の項の *equed* の説明を参照)。

work, rwork

終了時に、*work*(1) (実数型の場合) または *rwork*(1) (複素数型の場合) には、ピボット (要) 成長係数の逆数 norm(*A*)/norm(*U*) が格納される。「最大絶対成分」ノルムが使用される。*work*(1) (実数型の場合) または *rwork*(1) (複素数型の場合) が 1 よりはるかに小さい場合は、(平衡化された) マトリックス *A* の *LU* 因子分解の安定性は低くなる。このため、解 *x*、条件推定子 *rcond*、前進誤差範囲 *ferr* の信頼性も低くなる。$0 < info \le n$ で因子分解に失敗した場合は、*work*(1) (実数型の場合) または *rwork*(1) (複素数型の場合) には、*A* の先頭の *info* 列のピボット (要) 成長係数の逆数が格納される。

info

INTEGER。*info* = 0 の場合は、ルーチンの実行は正常に終了した。

info = -*i* の場合は、*i* 番目のパラメータの値が不正である。

info = *i* で、*i* ≤ *n* の場合は、*U* (*i*, *i*) が完全に 0 である。因子分解は完了したが、係数 *U* が完全に特異であるため、解と誤差範囲は計算できなかった。*rcond* = 0 が返される。

info = *i* で、*i* = *n* +1 の場合は、*U* は特異でないが、*rcond* がマシンの精度より小さいため、マトリックスは有効な精度で特異になる。しかし、この場合は、解と誤差範囲が計算される。これは、計算された解の精度が、*rcond* の値から想定される精度より高くなる場合があるためである。

?gbsv

バンド・マトリックス *A* を係数行列とする、複数の右辺を持つ連立 1 次方程式の解を計算する。

```
call sgbsv (n, kl, ku, nrhs, ab, ldab, ipiv, b, ldb, info)
call dgbsv (n, kl, ku, nrhs, ab, ldab, ipiv, b, ldb, info)
call cgbsv (n, kl, ku, nrhs, ab, ldab, ipiv, b, ldb, info)
call zgbsv (n, kl, ku, nrhs, ab, ldab, ipiv, b, ldb, info)
```

説明

このルーチンは、実数または複素連立 1 次方程式 $AX = B$ を、X について解く。A は *kl* 個の劣対角成分と *ku* 個の優対角成分を持つ $n \times n$ のバンド・マトリックス、マトリックス B の列は個々の右辺、X の列はそれに対応する解である。

部分的なピボット (要) 演算と行の交換による LU 分解を使用して、A を $A = L\,U$, として因子分解する。L は置換マトリックスと *kl* 個の劣対角成分を持つ単位下三角マトリックスの積、U は *kl*+*ku* 個の優対角成分を持つ上三角マトリックスである。次に、A の因子分解された形式を使用して、連立方程式 $AX = B$ を解く。

入力パラメータ

n	INTEGER。A の次数。B の行の数 ($n \geq 0$)。
kl	INTEGER。A のバンド内の劣対角成分の数 ($kl \geq 0$)。
ku	INTEGER。A のバンド内の優対角成分の数 ($ku \geq 0$)。
nrhs	INTEGER。右辺の数。B の列の数 ($nrhs \geq 0$)。
ab, b	REAL (sgbsv の場合) DOUBLE PRECISION (dgbsv の場合) COMPLEX (cgbsv の場合)

DOUBLE COMPLEX (zgbsv の場合)
配列 : *ab*(*ldab*, *)、*b*(*ldb*, *)
配列 *ab* には、マトリックス *A* がバンド形式で格納される (「「マトリックスの格納形式」」を参照)。
ab の第 2 次元は、少なくとも max(1, *n*) でなければならない。
配列 *b* には、マトリックス *B* が格納される。このマトリックスの列は、連立方程式の右辺である。
b の第 2 次元は、少なくとも max(1, *nrhs*) でなければならない。

ldab INTEGER。配列 *ab* の第 1 次元。
(*ldab* ≥ 2*kl* + *ku* +1)

ldb INTEGER。*b* の第 1 次元。*ldb* ≥ max(1, *n*)

出力パラメータ

ab *L* と *U* によって上書きされる。*U* の対角成分と *kl* + *ku* 個の優対角成分は、*ab* の最初の 1 + *kl* + *ku* 行に格納される。*L* の計算に使用した乗数は、次の *kl* 行に格納される。

b 解のマトリックス *X* によって上書きされる。

ipiv INTEGER
配列 DIMENSION は少なくとも max(1, *n*)
ピボット (要) のインデックス : 行 *i* は行 *ipiv*(*i*) と交換されている。

info INTEGER。*info* = 0 の場合は、ルーチンの実行は正常に終了した。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。
info = *i* の場合は、*U* (*i*, *i*) が完全に 0 である。因子分解は完了したが、係数 *U* が完全に特異であるため、解を計算できなかった。

?gbsvx

バンド・マトリックス A を係数行列とする、複数の右辺を持つ実数または複素連立 1 次方程式の解を計算し、解の誤差範囲を示す。

```
call sgbsvx (fact, trans, n, kl, ku, nrhs, ab, ldab, afb, ldafb,
             ipiv, equed, r, c, b, ldb, x, ldx, rcond, ferr, berr,
             work, iwork, info)
call dgbsvx (fact, trans, n, kl, ku, nrhs, ab, ldab, afb, ldafb,
             ipiv, equed, r, c, b, ldb, x, ldx, rcond, ferr, berr,
             work, iwork, info)
call cgbsvx (fact, trans, n, kl, ku, nrhs, ab, ldab, afb, ldafb,
             ipiv, equed, r, c, b, ldb, x, ldx, rcond, ferr, berr,
             work, rwork, info)
call zgbsvx (fact, trans, n, kl, ku, nrhs, ab, ldab, afb, ldafb,
             ipiv, equed, r, c, b, ldb, x, ldx, rcond, ferr, berr,
             work, rwork, info)
```

説明

このルーチンは、LU 因子分解を使用して、実数または複素連立 1 次方程式 $AX = B$、$A^TX = B$、または $A^HX = B$ の解を計算する。A は *kl* 個の劣対角成分と *ku* 個の優対角成分を持つ次数 *n* のバンド・マトリックス、マトリックス B の列は個々の右辺、X の列はそれに対応する解である。

このルーチンは、解の誤差範囲と条件数の推定値も示す。

ルーチン ?gbsvx は、以下の手順を実行する。

1. *fact* = 'E' の場合は、連立方程式を平衡化するための実スケール係数 *r* と *c* を計算する。

trans = 'N': $\mathrm{diag}(r)*A*\mathrm{diag}(c)*\mathrm{diag}(c)^{-1}*X = \mathrm{diag}(r)*B$

trans = 'T': $(\mathrm{diag}(r)*A*\mathrm{diag}(c))^{T}*\mathrm{diag}(r)^{-1}*X = \mathrm{diag}(c)*B$

trans = 'C': $(\mathrm{diag}(r)*A*\mathrm{diag}(c))^{H}*\mathrm{diag}(r)^{-1}*X = \mathrm{diag}(c)*B$

連立方程式が平衡化されるかどうかは、マトリックス A のスケーリングによって決まる。平衡化が行われる場合は、A は diag(*r*)**A**diag(*c*) によって上書きされ、B は diag(*r*)**B* (*trans*='N' の場合) または diag(*c*)**B* (*trans* = 'T' または 'C' の場合) によって上書きされる。

2. *fact* = 'N' または 'E' の場合は、*LU* 分解を使用して、(*fact* = 'E' の場合は平衡化の後に) マトリックス A を $A = L\,U$ として因子分解する。L は置換マトリックスと *kl* 個の劣対角成分を持つ単位下三角マトリックスの積、U は *kl*+*ku* 個の優対角成分を持つ上三角マトリックスである。

3. $U_{i,i} = 0$ の場合、つまり U が完全に特異である場合は、ルーチンは *info* = i を返す。それ以外の場合は、A の因子分解された形式を使用して、マトリックス A の条件数を推定する。条件数の逆数がマシンの精度より小さい場合は、警告として *info* = *n* + 1 を返すが、後で説明するように、ルーチンは続行され、X について解を算出し、誤差範囲を計算する。

4. A の因子分解された形式を使用して、連立方程式を X について解く。

5. 計算された解のマトリックスの精度を繰り返し改善し、解の誤差範囲と後退誤差の推定値を計算する。

6. 平衡化が行われている場合は、平衡化が行われる前の元の連立方程式を解くために、diag(*c*) (*trans* = 'N' の場合) または diag(*r*) (*trans* = 'T' または 'C' の場合) によって、マトリックス X を事前に乗算する。

入力パラメータ

fact

CHARACTER*1。'F'、'N'、または 'E' でなければならない。

ルーチンの開始時にマトリックス A の因子分解された形式が与えられるかどうか、与えられない場合は、マトリックス A を因子分解する前に A を平衡化するかどうかを指定する。

fact = 'F' の場合は、開始時に、*afb* と *ipiv* に A の因子分解された形式が格納される。*equed* が 'N' でない場合は、マトリックス A は、*r* と *c* で指定されたスケール係数によって平衡化されている。*ab*、*afb*、*ipiv* は変更されない。

fact = 'N' の場合は、マトリックス A を *afb* にコピーし、因子分解を実行する。

fact = 'E' の場合は、必要に応じてマトリックス A を平衡化してから、*afb* にコピーし、因子分解を実行する。

trans

CHARACTER*1。'N'、'T'、または 'C' でなければならない。

連立方程式の形式を指定する。

trans = 'N' の場合は、連立方程式の形式は $A X = B$ (転置なし) である。

trans = 'T' の場合は、連立方程式の形式は $A^T X = B$ (転置) である。

trans = 'C' の場合は、連立方程式の形式は $A^H X = B$ (共役転置) である。

n INTEGER。1 次方程式の数。マトリックス A の次数 ($n \geq 0$)。

kl INTEGER。A のバンド内の劣対角成分の数 ($kl \geq 0$)。

ku INTEGER。A のバンド内の優対角成分の数 ($ku \geq 0$)。

nrhs INTEGER。右辺の数。マトリックス B と X の列の数 ($nrhs \geq 0$)。

ab, afb, b, work

REAL (sgesvx の場合)

DOUBLE PRECISION (dgesvx の場合)

COMPLEX (cgesvx の場合)

DOUBLE COMPLEX (zgesvx の場合)

配列 : *a(lda, *)*, *af(ldaf, *)*, *b(ldb, *)*, *work(*)*

配列 *ab* には、マトリックス A がバンド形式で格納される (「「マトリックスの格納形式」」を参照)。

ab の第 2 次元は、少なくとも max(1, *n*) でなければならない。

fact = 'F' で、*equed* が 'N' でない場合は、A は、*r* または *c*、あるいはその両方のスケール係数によって平衡化されている。

配列 *afb* は、*fact* = 'F' の場合は入力引数になる。

afb の第 2 次元は、少なくとも max(1, *n*) でなければならない。

この配列には、マトリックス A の因子分解された形式、すなわち、?gbtrf による因子分解 $A = L U$ で得られた係数 L と U が格納される。U は、*kl* + *ku* 個の優対角成分を持つ上三角バンド・マトリックスとして、*afb* の最初の 1 + *kl* + *ku* 行に格納される。因子分解に使用した乗数は、次の *kl* 行に格納される。

equed が 'N' でない場合は、afb は、平衡化されたマトリックス A の因子分解された形式になる。

配列 b には、マトリックス B が格納される。このマトリックスの列は、連立方程式の右辺である。b の第 2 次元は、少なくとも max(1, nrhs) でなければならない。

work(*) はワークスペース配列である。
work の次元は、実数型の場合は少なくとも max(1, 3*n)、複素数型の場合は少なくとも max(1, 2*n) でなければならない。

ldab　INTEGER。ab の第 1 次元。ldab ≥ kl+ku+1

ldafb　INTEGER。afb の第 1 次元。
ldafb ≥ 2*kl+ku+1

ldb　INTEGER。b の第 1 次元。ldb ≥ max(1, n)

ipiv　INTEGER
配列 DIMENSION は少なくとも max(1, n)
配列 ipiv は、fact = 'F' の場合は入力引数になる。この配列には、?gbtrf による因子分解 $A = L\,U$ で得られたピボット (要) のインデックスが格納される。マトリックスの行 i は、行 ipiv(i) と交換されている。

equed　CHARACTER*1。'N'、'R'、'C'、または 'B' でなければならない。
equed は、fact = 'F' の場合は入力引数になる。この引数は、実行された平衡化の形式を指定する。
equed = 'N' の場合は、平衡化は行われていない (fact = 'N') の場合は常に真)。
equed = 'R' の場合は、行の平衡化が行われ、A は diag(r) によって事前に乗算されている。
equed = 'C' の場合は、列の平衡化が行われ、A は diag(c) によって事後に乗算されている。
equed = 'B' の場合は、行と列の平衡化が行われ、A は diag(r)*A*diag(c) で置き換えられている。

r, c　REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
配列 : r(n), c(n)
配列 r には A の行のスケール係数が格納され、配列 c には A の列のスケール係数が格納される。これらの配列は、fact = 'F' の場合にのみ入力引数になり、それ以外の場合は出力引数になる。

equed = 'R' または 'B' の場合は、A は diag(r) によって左辺で乗算される。equed = 'N' または 'C' の場合は、r は使用されない。
fact = 'F' で、equed = 'R' または 'B' の場合は、r の各成分は正の値でなければならない。
equed = 'C' または 'B' の場合は、A は diag(c) によって右辺で乗算される。equed = 'C' または 'B' の場合は、c は使用されない。
fact = 'F' で、equed = 'C' または 'B' の場合は、c の各成分は正の値でなければならない。

ldx INTEGER。出力配列 x の第 1 次元。ldx ≥ max(1, n)

iwork INTEGER
ワークスペース配列 DIMENSION は少なくとも max(1, n)。実数型でのみ使用される。

rwork REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
ワークスペース配列 DIMENSION は少なくとも max(1, n)。複素数型でのみ使用される。

出力パラメータ

x REAL (sgbsvx の場合)
DOUBLE PRECISION (dgbsvx の場合)
COMPLEX (cgbsvx の場合)
DOUBLE COMPLEX (zgbsvx の場合)
配列 DIMENSION は (ldx, *)。
info = 0 または info = n+1 の場合は、配列 x には、**元の**連立方程式の解のマトリックス X が格納される。ただし、equed ≠ 'N' の場合は、A と B は終了時に修正され、**平衡化された**連立方程式の解は次のようになる。
$\mathrm{diag}(c)^{-1}*X$ (trans = 'N' で、equed = 'C' または 'B' の場合)
$\mathrm{diag}(r)^{-1}*X$ (trans = 'T' または 'C' で、equed = 'R' または 'B' の場合)
x の第 2 次元は、少なくとも max(1, nrhs) でなければならない。

ab

配列 *ab* は、*fact* = 'F' または 'N' の場合、あるいは *fact* = 'E' で *equed* = 'N' の場合は、終了時に変更されない。

equed ≠ 'N' の場合は、*A* は終了時に次のようにスケーリングされる。

equed = 'R': $A = \mathrm{diag}(r)*A$

equed = 'C': $A = A*\mathrm{diag}(c)$

equed = 'B': $A = \mathrm{diag}(r)*A*\mathrm{diag}(c)$

afb

fact = 'N' または 'E' の場合は、*afb* は出力引数になる。この引数は、終了時に、元のマトリックス *A* (*fact* = 'N' の場合) または平衡化されたマトリックス *A* (*fact* = 'E' の場合) の *LU* 因子分解の詳細を返す。平衡化されたマトリックスの形式については、*ab* の説明を参照のこと。

b

trans = 'N' で、*equed* = 'R' または 'B' の場合は、diag(*c*)**b* によって上書きされる。

trans = 'T' で、*equed* = 'C' または 'B' の場合は、diag(*c*)**b* によって上書きされる。

r, c

これらの配列は、*fact* ≠ 'F' の場合は出力引数になる。

「入力引数」の項の *r, c* の説明を参照のこと。

rcond

REAL (単精度の場合)

DOUBLE PRECISION (倍精度の場合)

平衡化後のマトリックス *A* の条件数の逆数の推定値 (平衡化が行われる場合)。

rcond がマシンの精度より小さい場合 (特に、*rcond* = 0 の場合) は、マトリックスは有効な精度で特異になる。この条件は、*info* > 0 のリターン・コードで示される。

ferr, berr

REAL (単精度の場合)

DOUBLE PRECISION (倍精度の場合)

配列 DIMENSION は少なくとも max(1, *nrhs*)。各解ベクトルの成分ごとの前進誤差と相対後退誤差を格納する。

ipiv

fact = 'N' または 'E' の場合は、*ipiv* は出力引数になる。この引数には、終了時に、元のマトリックス *A* (*fact* = 'N' の場合) または平衡化されたマトリックス *A* (*fact* = 'E' の場合) の因子分解 $A = L\,U$ で得られたピボット (要) のインデックスが格納される。

equed	*fact* ≠ 'F' の場合は、*equed* は出力引数になる。この引数は、実行された平衡化の形式を指定する (「入力引数」の項の *equed* の説明を参照)。
work, *rwork*	終了時に、*work*(1) (実数型の場合) または *rwork*(1) (複素数型の場合) には、ピボット (要) 成長係数の逆数 norm(A)/norm(U) が格納される。「最大絶対成分」ノルムが使用される。*work*(1) (実数型の場合) または *rwork*(1) (複素数型の場合) が 1 よりはるかに小さい場合は、(平衡化された) マトリックス A の LU 因子分解の安定性は低くなる。このため、解 *x*、条件推定子 *rcond*、前進誤差範囲 *ferr* の信頼性も低くなる。0 < *info* ≤ *n* で因子分解に失敗した場合は、*work*(1) (実数型の場合) または *rwork*(1) (複素数型の場合) には、A の先頭の *info* 列のピボット (要) 成長係数の逆数が格納される。
info	INTEGER。*info* = 0 の場合は、ルーチンの実行は正常に終了した。 *info* = -*i* の場合は、*i* 番目のパラメータの値が不正である。 *info* = *i* で、*i* ≤ *n* の場合は、U (*i*, *i*) が完全に 0 である。因子分解は完了したが、係数 U が完全に特異であるため、解と誤差範囲は計算できなかった。 *rcond* = 0 が返される。 *info* = *i* で、*i* = *n* +1 の場合は、U は特異でないが、*rcond* がマシンの精度より小さいため、マトリックスは有効な精度で特異になる。しかし、この場合は、解と誤差範囲が計算される。これは、計算された解の精度が、*rcond* の値から想定される精度より高くなる場合があるためである。

?gtsv

三重対角マトリックス A を係数行列とする、複数の右辺を持つ連立 1 次方程式の解を計算する。

```
call sgtsv (n, nrhs, dl, d, du, b, ldb, info)
call dgtsv (n, nrhs, dl, d, du, b, ldb, info)
call cgtsv (n, nrhs, dl, d, du, b, ldb, info)
call zgtsv (n, nrhs, dl, d, du, b, ldb, info)
```

説明

このルーチンは、連立 1 次方程式 $AX = B$ を X について解く。A は $n \times n$ の三重対角マトリックス、マトリックス B の列は個々の右辺、X の列はそれに対応する解である。
このルーチンは、部分的なピボット (要) 演算によるガウス消去を使用する。

方程式 $A^{T}X = B$ の解は、引数 *du* と *dl* の順序を入れ替えて求められる。

入力パラメータ

n	INTEGER。A の次数。B の行の数 ($n \geq 0$)。
nrhs	INTEGER。右辺の数。B の列の数 ($nrhs \geq 0$)。
dl, d, du, b	REAL (sgtsv の場合) DOUBLE PRECISION (dgtsv の場合) COMPLEX (cgtsv の場合) DOUBLE COMPLEX (zgtsv の場合) 配列 : *dl*(n - 1), *d*(n), *du*(n - 1), *b*(*ldb*, *) 配列 *dl* には、A の (n - 1) 個の劣対角成分が格納される。 配列 *d* には、A の対角成分が格納される。 配列 *du* には、A の (n - 1) 個の優対角成分が格納される。

配列 *b* には、マトリックス *B* が格納される。このマトリックスの列は、連立方程式の右辺である。
b の第 2 次元は、少なくとも max(1, *nrhs*) でなければならない。

ldb INTEGER。*b* の第 1 次元。*ldb* ≥ max(1, *n*)

出力パラメータ

dl A の *LU* 因子分解で得られた上三角マトリックス *U* の 2 番目の優対角成分の (*n*-2) 個の成分によって上書きされる。これらの成分は、*dl*(1), . . . , *dl*(*n*-2) に格納される。

d *U* の *n* 個の対角成分によって上書きされる。

du *U* の最初の優対角成分の (*n*-1) 個の成分によって上書きされる。

b 解のマトリックス *X* によって上書きされる。

info INTEGER。*info* = 0 の場合は、ルーチンの実行は正常に終了した。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。
info = *i* の場合は、*U* (*i*, *i*) が完全に 0 であるため、解は計算されなかった。*i* = *n* でない限り、因子分解は完了していない。

?gtsvx

三重対角マトリックス A を係数行列とする、複数の右辺を持つ実数または複素連立 1 次方程式の解を計算し、解の誤差範囲を示す。

```
call sgtsvx (fact, trans, n, nrhs, dl, d, du, dlf, df, duf, du2, ipiv,
             b, ldb, x, ldx, rcond, ferr, berr, work, iwork, info)
call dgtsvx (fact, trans, n, nrhs, dl, d, du, dlf, df, duf, du2, ipiv,
             b, ldb, x, ldx, rcond, ferr, berr, work, iwork, info)
call cgtsvx (fact, trans, n, nrhs, dl, d, du, dlf, df, duf, du2, ipiv,
             b, ldb, x, ldx, rcond, ferr, berr, work, rwork, info)
call zgtsvx (fact, trans, n, nrhs, dl, d, du, dlf, df, duf, du2, ipiv,
             b, ldb, x, ldx, rcond, ferr, berr, work, rwork, info)
```

説明

このルーチンは、LU 因子分解を使用して、実数または複素連立 1 次方程式 $AX = B$、$A^TX = B$、または $A^HX = B$ の解を計算する。A は次数 `n` の三重対角マトリックス、マトリックス B の列は個々の右辺、X の列はそれに対応する解である。

このルーチンは、解の誤差範囲と条件数の推定値も示す。

ルーチン `?gtsvx` は、以下の手順を実行する。

1. `fact` = 'N' の場合は、LU 分解を使用して、マトリックス A を $A = LU$ として因子分解する。L は置換マトリックスと単位下二重対角マトリックスの積で、U は主対角成分と最初の 2 つの優対角成分にのみ 0 でない値を持つ上三角マトリックスである。

2. $U_{i,i} = 0$ の場合、つまり U が完全に特異である場合は、ルーチンは `info` = `i` を返す。それ以外の場合は、A の因子分解された形式を使用して、マトリックス A の条件数を推定する。条件数の逆数がマシンの精度より小さい場合は、警告として `info` = `n` + 1 を返すが、後で説明するようにルーチンは続行され、X について解を算出し、誤差範囲を計算する。

3. A の因子分解された形式を使用して、連立方程式を X について解く。

4. 計算された解のマトリックスの精度の改善を繰り返し実行し、解の誤差範囲と後退誤差の推定値を計算する。

入力パラメータ

fact CHARACTER*1。'F' または 'N' でなければならない。

ルーチンの開始時にマトリックス A の因子分解された形式が与えられるかどうかを指定する。

fact = 'F' の場合は、開始時に、*dlf*、*df*、*duf*、*du2*、*ipiv* に、A の因子分解された形式が格納される。配列 *dl*、*d*、*du*、*dlf*、*df*、*duf*、*du2*、*ipiv* は変更されない。

fact = 'N' の場合は、マトリックス A を *dlf*、*df*、*duf* にコピーし、因子分解を実行する。

trans CHARACTER*1。'N'、'T'、または 'C' でなければならない。

連立方程式の形式を指定する。

trans = 'N' の場合は、連立方程式の形式は $A\,X = B$ (転置なし) である。

trans = 'T' の場合は、連立方程式の形式は $A^{T}\,X = B$ (転置) である。

trans = 'C' の場合は、連立方程式の形式は $A^{H}\,X = B$ (共役転置) である。

n INTEGER。1 次方程式の数。マトリックス A の次数 ($n \geq 0$)

nrhs INTEGER。右辺の数。マトリックス B と X の列の数 ($nrhs \geq 0$)。

dl, d, du, dlf, df, duf, du2, b, x, work REAL (sgtsvx の場合)
DOUBLE PRECISION (dgtsvx の場合)
COMPLEX (cgtsvx の場合)
DOUBLE COMPLEX (zgtsvx の場合)

配列 :

dl、次元は (n - 1)。A の劣対角成分を格納する。

d、次元は (n)。A の対角成分を格納する。

du、次元は (n - 1)。A の優対角成分を格納する。

dlf、次元は (n - 1)。fact = 'F' の場合は、dlf は入力引数になる。この引数には、開始時に、?gttrf による A の LU 因子分解で得られたマトリックス L を定義する、(n - 1) 個の乗数が格納される。

df、次元は (n)。fact = 'F' の場合は、df は入力引数になる。この引数には、開始時に、A の LU 因子分解で得られた上三角マトリックス U の n 個の対角成分が格納される。

duf、次元は (n - 1)。fact = 'F' の場合は、duf は入力引数になる。この引数には、開始時に、U の最初の優対角成分の (n - 1) 個の成分が格納される。

du2、次元は (n - 2)。fact = 'F' の場合は、du2 は入力引数になる。この引数には、開始時に、U の 2 番目の優対角成分の (n - 2) 個の成分が格納される。

b(ldb, *) には、右辺のマトリックス B が格納される。b の第 2 次元は、少なくとも max(1, nrhs) でなければならない。

x(ldx, *) には、解のマトリックス X が格納される。x の第 2 次元は、少なくとも max(1, nrhs) でなければならない。

work (*) はワークスペース配列である。
work の次元は、実数型の場合は少なくとも max(1, 3*n)、複素数型の場合は少なくとも max(1, 2*n) でなければならない。

ldb　INTEGER。b の第 1 次元。ldb ≥ max(1, n)

ldx　INTEGER。x の第 1 次元。ldx ≥ max(1, n)

ipiv　INTEGER
配列 DIMENSION は少なくとも max(1, n)。fact = 'F' の場合は、ipiv は入力引数になる。この配列には、開始時に、?gttrf によって返されたピボット (要) のインデックスが格納される。

iwork　INTEGER
ワークスペース配列 DIMENSION (n)。実数型でのみ使用される。

rwork REAL (cgtsvx の場合)
DOUBLE PRECISION (zgtsvx の場合)
ワークスペース配列 DIMENSION (*n*)。複素数型でのみ使用される。

出力パラメータ

x REAL (sgtsvx の場合)
DOUBLE PRECISION (dgtsvx の場合)
COMPLEX (cgtsvx の場合)
DOUBLE COMPLEX (zgtsvx の場合)
配列 DIMENSION は (*ldx*, *)。

info = 0 または *info* = *n*+1 の場合は、配列 *x* には、解のマトリックス *X* が格納される。*x* の第 2 次元は、少なくとも max(1, *nrhs*) でなければならない。

dlf *fact* = 'N' の場合は、*dlf* は出力引数になる。この引数には、終了時に、*A* の *LU* 因子分解で得られたマトリックス *L* を定義する、(*n* - 1) 個の乗数が格納される。

df *fact* = 'N' の場合は、*df* は出力引数になる。この引数には、終了時に、*A* の *LU* 因子分解で得られた上三角マトリックス *U* の *n* 個の対角成分が格納される。

duf *fact* = 'N' の場合は、*duf* は出力引数になる。この引数には、終了時に、*U* の最初の優対角成分の (*n* - 1) 個の成分が格納される。

du2 *fact* = 'N' の場合は、*du2* は出力引数になる。この引数には、終了時に、*U* の 2 番目の優対角成分の (*n* - 2) 個の成分が格納される。

ipiv 配列 *ipiv* は、*fact* = 'N' の場合は出力引数になる。この配列は、終了時に、因子分解 *A* = *L U* で得られたピボット (要) のインデックスが格納される。マトリックスの行 *i* は、行 *ipiv*(*i*) と交換されている。*ipiv*(*i*) の値は、常に *i* または *i*+1 になる。*ipiv*(*i*)=*i* は、行の交換が不要であったことを示す。

rcond REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
マトリックス *A* の条件数の逆数の推定値。

rcond がマシンの精度より小さい場合 (特に、*rcond* = 0 の場合) は、マトリックスは有効な精度で特異になる。この条件は、*info* > 0 のリターン・コードで示される。

ferr, berr
: REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
配列 DIMENSION は少なくとも max(1, *nrhs*)。各解ベクトルの成分ごとの前進誤差と後退誤差を格納する。

info
: INTEGER。*info* = 0 の場合は、ルーチンの実行は正常に終了した。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。
info = *i* で、*i* ≤ *n* の場合は、U (*i*, *i*) が完全に 0 である。*i* = *n* でない限り、因子分解は完了していない。係数 U が完全に特異であるため、解と誤差範囲は計算できなかった。*rcond* = 0 が返される。
info = *i* で、*i* = *n* +1 の場合は、U は特異でないが、*rcond* がマシンの精度より小さいため、マトリックスは有効な精度で特異になる。しかし、この場合は、解と誤差範囲が計算される。これは計算された解の精度が、*rcond* の値から想定される精度より高くなる場合があるためである。

?posv

正値対称またはエルミート・
マトリックス A を係数行列とする、複数の
右辺を持つ連立 1 次方程式を解く。

```
call sposv (uplo, n, nrhs, a, lda, b, ldb, info)
call dposv (uplo, n, nrhs, a, lda, b, ldb, info)
call cposv (uplo, n, nrhs, a, lda, b, ldb, info)
call zposv (uplo, n, nrhs, a, lda, b, ldb, info)
```

説明

このルーチンは、実数または複素連立 1 次方程式 $AX = B$ を、X について解く。A は $n \times n$ の正値対称 / エルミート・マトリックス、マトリックス B の列は個々の右辺、X の列はそれに対応する解である。

コルスキー因子分解を使用して、A を $A = U^H U$ (*uplo* = 'U' の場合) または $A = LL^H$ (*uplo* = 'L' の場合) として因子分解する。U は上三角マトリックス、L は下三角マトリックスである。次に、A の因子分解された形式を使用して、連立方程式 $AX = B$ を解く。

入力パラメータ

uplo CHARACTER*1。'U' または 'U' でなければならない。
A の上三角部分と下三角部分のどちらを格納するか、また A をどのように因子分解するかを指定する。
uplo = 'U' の場合は、配列 *a* にはマトリックス A の上三角部分が格納され、A は $U^H U$ として因子分解される。
uplo = 'L' の場合は、配列 *a* にはマトリックス A の下三角部分が格納され、A は LL^H として因子分解される。

n INTEGER。A の次数 ($n \geq 0$)。

nrhs INTEGER。右辺の数。B の列の数 ($nrhs \geq 0$)。

a, b
REAL (sposv の場合)
DOUBLE PRECISION (dposv の場合)
COMPLEX (cposv の場合)
DOUBLE COMPLEX (zposv の場合)
配列 : *a*(*lda, **), *b*(*ldb, **)
配列 *a* には、マトリックス *A* の上三角部分または下三角部分が格納される (*uplo* を参照)。
a の第 2 次元は、少なくとも max(1, *n*) でなければならない。
配列 *b* には、マトリックス *B* が格納される。このマトリックスの列は、連立方程式の右辺である。
b の第 2 次元は、少なくとも max(1, *nrhs*) でなければならない。

lda
INTEGER。*a* の第 1 次元。*lda* ≥ max(1, *n*)

ldb
INTEGER。*b* の第 1 次元。*ldb* ≥ max(1, *n*)

出力パラメータ

a
info = 0 の場合は、*uplo* の指定に従って、*a* の上三角部分または下三角部分が、コルスキー係数 *U* または *L* によって上書きされる。

b
解のマトリックス *X* によって上書きされる。

info
INTEGER。*info* = 0 の場合は、ルーチンの実行は正常に終了した。
info = *-i* の場合は、*i* 番目のパラメータの値が不正である。
info = *i* の場合は、次数 *i* の先頭の小行列式 (したがって、マトリックス *A* そのもの) が正値になっていないため、因子分解を実行できず、解は計算されなかった。

?posvx

コルスキー因子分解を使用して、正値対称
またはエルミート・マトリックス *A* を
係数行列とする連立 1 次方程式の解を
計算し、解の誤差範囲を示す。

```
call sposvx (fact, uplo, n, nrhs, a, lda, af, ldaf, equed, s, b,
             ldb, x, ldx, rcond, ferr, berr, work, iwork, info)
call dposvx (fact, uplo, n, nrhs, a, lda, af, ldaf, equed, s, b,
             ldb, x, ldx, rcond, ferr, berr, work, iwork, info)
call cposvx (fact, uplo, n, nrhs, a, lda, af, ldaf, equed, s, b,
             ldb, x, ldx, rcond, ferr, berr, work, rwork, info)
call zposvx (fact, uplo, n, nrhs, a, lda, af, ldaf, equed, s, b,
             ldb, x, ldx, rcond, ferr, berr, work, rwork, info)
```

説明

このルーチンは、コルスキー因子分解 $A=U^HU$ または $A=LL^H$ を使用して、実数または複素連立 1 次方程式 $AX = B$ の解を計算する。A は $n \times n$ の正値実数対称 / エルミート・マトリックス、マトリックス *B* の列は個々の右辺、*X* の列はそれに対応する解である。

このルーチンは、解の誤差範囲と条件数の推定値も示す。

ルーチン ?posvx は、以下の手順を実行する。

1. *fact* = 'E' の場合は、連立方程式を平衡化するための実スケール係数 *s* を計算する。

$$\mathrm{diag}(s)*A*\mathrm{diag}(s)\ *\mathrm{diag}(s)^{-1}*X = \mathrm{diag}(s)*B$$

連立方程式が平衡化されるかどうかは、マトリックス *A* のスケーリングによって決まる。平衡化が行われる場合は、*A* は diag(*s*)**A**diag(*s*) によって上書きされ、*B* は diag(*s*)**B* によって上書きされる。

2. *fact* = 'N' または 'E' の場合は、コルスキー分解を使用して、(*fact* = 'E' の場合は平衡化の後に) マトリックス *A* を次のように因子分解する。

$A = U^H U$ (*uplo* = 'U' の場合)、または
$A = L L^H$ (*uplo* = 'L' の場合)
U は上三角マトリックスで、L は下三角マトリックスである。

3. 先頭の $i \times i$ の主小行列式が正値でない場合は、ルーチンは *info* = *i* を返す。それ以外の場合は、A の因子分解された形式を使用して、マトリックス A の条件数を推定する。条件数の逆数がマシンの精度より小さい場合は、警告として *info* = *n* + 1 を返すが、後で説明するようにルーチンは続行され、X について解を算出し、誤差範囲を計算する。

4. A の因子分解された形式を使用して、連立方程式を X について解く。

5. 計算された解のマトリックスの精度の改善を繰り返し実行し、解の誤差範囲と後退誤差の推定値を計算する。

6. 平衡化が行われている場合は、平衡化が行われる前の元の連立方程式を解くために、diag(*s*) によってマトリックス X を事前に乗算する。

入力パラメータ

fact

CHARACTER*1。'F'、'N'、または 'E' でなければならない。

ルーチンの開始時にマトリックス A の因子分解された形式が与えられるかどうか、与えられない場合は、マトリックス A を因子分解する前に平衡化を行うかどうかを指定する。

fact = 'F' の場合は、開始時に、*af* に A の因子分解された形式が格納される。*equed* = 'Y' の場合は、マトリックス A は、*s* で指定されたスケール係数によって平衡化されている。
a と *af* は変更されない。

fact = 'N' の場合は、マトリックス A を *af* にコピーし、因子分解を実行する。
fact = 'E' の場合は、必要に応じてマトリックス A を平衡化してから、*af* にコピーし、因子分解を実行する。

uplo

CHARACTER*1。'U' または 'L' でなければならない。

A の上三角部分と下三角部分のどちらを格納するか、また A をどのように因子分解するかを指定する。

uplo = 'U' の場合は、配列 *a* にはマトリックス A の上三角部分が格納され、A は $U^H U$ として因子分解される。

uplo = 'L' の場合は、配列 *a* にはマトリックス A の下三角部分が格納され、A は LL^H として因子分解される。

n	INTEGER。*A* の次数 ($n \geq 0$)。
nrhs	INTEGER。右辺の数。*B* の列の数 ($nrhs \geq 0$)。
a, af, b, work	REAL (sposvx の場合) DOUBLE PRECISION (dposvx の場合) COMPLEX (cposvx の場合) DOUBLE COMPLEX (zposvx の場合) 配列 : *a(lda, *)*, *af(ldaf, *)*, *b(ldb, *)*, *work(*)* 配列 *a* には、*uplo* の指定に従って、マトリックス *A* が格納される。 *fact* = 'F' で *equed* = 'Y' の場合は、*A* は *s* のスケール係数によって平衡化され、*a* には平衡化されたマトリックス diag(*s*)**A**diag(*s*) が格納されている。*a* の第 2 次元は、少なくとも max(1, *n*) でなければならない。 配列 *af* は、*fact* = 'F' の場合は入力引数になる。この配列には、*A* のコルスキー因子分解で得られた三角係数 *U* または *L* が、*A* と同じ格納形式で格納される。*equed* が 'N' でない場合は、*af* は、平衡化されたマトリックス diag(*s*)**A**diag(*s*) の因子分解された形式になる。*af* の第 2 次元は、少なくとも max(1, *n*) でなければならない。 配列 *b* には、マトリックス *B* が格納される。このマトリックスの列は、連立方程式の右辺である。*b* の第 2 次元は、少なくとも max(1, *nrhs*) でなければならない。 *work(*)* はワークスペース配列である。 *work* の次元は、実数型の場合は少なくとも max(1, 3**n*)、複素数型の場合は少なくとも max(1, 2**n*) でなければならない。
lda	INTEGER。*a* の第 1 次元。$lda \geq \max(1, n)$
ldaf	INTEGER。*af* の第 1 次元。$ldaf \geq \max(1, n)$
ldb	INTEGER。*b* の第 1 次元。$ldb \geq \max(1, n)$
equed	CHARACTER*1。'N' または 'Y' でなければならない。 *equed* は、*fact* = 'F' の場合は入力引数になる。この引数は、実行された平衡化の形式を指定する。 *equed* = 'N' の場合は、平衡化は行われていない (*fact* = 'N' の場合は常に真)。

equed = 'Y' の場合は、平衡化が行われ、A は diag(s)*A*diag(s) で置き換えられている。

s
REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
配列 DIMENSION は (n)
配列 s には、A のスケール係数が格納される。この配列は、fact = 'F' の場合にのみ入力引数になり、それ以外の場合は出力引数になる。
equed = 'N' の場合は、s は使用されない。
fact = 'F' で、equed = 'Y' の場合は、s の各成分は正の値でなければならない。

ldx
INTEGER。出力配列 x の第 1 次元。ldx ≥ max(1, n)

iwork
INTEGER
ワークスペース配列 DIMENSION は少なくとも max(1, n)。実数型でのみ使用される。

rwork
REAL (cposvx の場合)
DOUBLE PRECISION (zposvx の場合)
ワークスペース配列 DIMENSION は少なくとも max(1, n)。複素数型でのみ使用される。

出力パラメータ

x
REAL (sposvx の場合)
DOUBLE PRECISION (dposvx の場合)
COMPLEX (cposvx の場合)
DOUBLE COMPLEX (zposvx の場合)
配列 DIMENSION は (ldx, *)
info = 0 または info = n+1 の場合は、配列 x には、元の連立方程式の解のマトリックス X が格納される。ただし、equed = 'Y' の場合は、A と B は終了時に修正され、平衡化された連立方程式の解は $\mathrm{diag}(s)^{-1}*X$ になる。
x の第 2 次元は、少なくとも max(1, nrhs) でなければならない。

a	配列 *a* は、*fact* = 'F' または 'N' の場合、または *fact* = 'E' で *equed* = 'N' の場合は、終了時に変更されない。 *fact* = 'E' で *equed* = 'Y' の場合は、*A* は diag(*s*)*A*diag(*s*) によって上書きされる。
af	*fact* = 'N' または 'E' の場合は、*af* は出力引数になる。この引数は終了時に、元のマトリックス *A* (*fact* = 'N' の場合) または平衡化されたマトリックス *A* (*fact* = 'E' の場合) のコルスキー因子分解 $A=U^HU$ または $A=LL^H$ で得られた三角係数 *U* または *L* を返す。平衡化されたマトリックスの形式は、*a* の説明を参照のこと。
b	*equed* = 'Y' の場合は、diag(*s*)**B* によって上書きされる。 *equed* = 'N' の場合は変更されない。
s	この配列は、*fact* ≠ 'F' の場合は出力引数になる。「入力引数」の項の *s* の説明を参照。
rcond	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 平衡化後のマトリックス *A* の条件数の逆数の推定値 (平衡化が行われる場合)。*rcond* がマシンの精度より小さい場合 (特に、*rcond* = 0 の場合) は、マトリックスは有効な精度で特異になる。この条件は、*info* > 0 のリターン・コードで示される。
ferr, berr	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列 DIMENSION は少なくとも max(1, *nrhs*)。各解ベクトルの成分ごとの前進誤差と相対後退誤差を格納する。
equed	*fact* ≠ 'F' の場合は、*equed* は出力引数になる。この引数は、実行された平衡化の形式を指定する (「入力引数」の項の *equed* の説明を参照)。
info	INTEGER。*info* = 0 の場合は、ルーチンの実行は正常に終了した。 *info* = -*i* の場合は、*i* 番目のパラメータの値が不正である。 *info* = *i* で、*i* ≤ *n* の場合は、次数 *i* の先頭の小行列式 (したがって、マトリックス *A* そのもの) が正値になっていないため、因子分解を実行できず、解と誤差範囲は計算できなかった。 *rcond* = 0 が返される。

info = *i* で、*i* = *n* +1 の場合は、*U* は特異でないが、*rcond* がマシンの精度より小さいため、マトリックスは有効な精度で特異になる。しかし、この場合は、解と誤差範囲が計算される。これは計算された解の精度が、*rcond* の値から想定される精度より高くなる場合があるためである。

?ppsv

パックド形式の正値対称 (エルミート)
マトリックス *A* を係数行列とする、
複数の右辺を持つ連立 1 次方程式を解く。

```
call sppsv (uplo, n, nrhs, ap, b, ldb, info)
call dppsv (uplo, n, nrhs, ap, b, ldb, info)
call cppsv (uplo, n, nrhs, ap, b, ldb, info)
call zppsv (uplo, n, nrhs, ap, b, ldb, info)
```

説明

このルーチンは、実数または複素連立 1 次方程式 $AX = B$ を、X について解く。A はパックド形式で格納される $n \times n$ の実数正値対称 / エルミート・マトリックス、マトリックス B の列は個々の右辺、X の列はそれに対応する解である。

コルスキー因子分解を使用して、A を $A = U^H U$ (*uplo* = 'U' の場合) または $A = LL^H$ (*uplo* = 'L' の場合) として因子分解する。U は上三角マトリックス、L は下三角マトリックスである。次に、A の因子分解された形式を使用して、連立方程式 $AX = B$ を解く。

入力パラメータ

uplo CHARACTER*1。'U' または 'L' でなければならない。

	A の上三角部分と下三角部分のどちらを格納するか、また A をどのように因子分解するかを指定する。 *uplo* = 'U' の場合は、配列 *a* にはマトリックス A の上三角部分が格納され、A は $U^H U$ として因子分解される。 *uplo* = 'L' の場合は、配列 *a* にはマトリックス A の下三角部分が格納され、A は LL^H として因子分解される。
n	INTEGER。A の次数 ($n \geq 0$)。
nrhs	INTEGER。右辺の数。B の列の数 ($nrhs \geq 0$)。
ap, b	REAL (sppsv の場合) DOUBLE PRECISION (dppsv の場合) COMPLEX (cppsv の場合) DOUBLE COMPLEX (zppsv の場合) 配列 : *ap*(*), *b*(*ldb, *) 配列 *ap* には、*uplo* の指定に従って、マトリックス A の上三角部分または下三角部分が**パックド格納形式**で格納される (「「マトリックスの格納形式」」を参照)。 *ap* の次元は、少なくとも max(1, *n*(*n*+1)/2) でなければならない。 配列 *b* には、マトリックス B が格納される。このマトリックスの列は、連立方程式の右辺である。 *b* の第 2 次元は、少なくとも max(1, *nrhs*) でなければならない。
ldb	INTEGER。*b* の第 1 次元。$ldb \geq \max(1, n)$

出力パラメータ

ap	*info* = 0 の場合は、*uplo* の指定に従って、パックド格納形式の A の上三角部分または下三角部分が、コルスキー係数 U または L によって上書きされる。
b	解のマトリックス X によって上書きされる。
info	INTEGER。*info* = 0 の場合は、ルーチンの実行は正常に終了した。 *info* = -*i* の場合は、*i* 番目のパラメータの値が不正である。 *info* = *i* の場合は、次数 *i* の先頭の小行列式 (したがって、マトリックス A そのもの) が正値になっていないため、因子分解を実行できず、解は計算されなかった。

?ppsvx

コルスキー因子分解を使用して、パックド形式の正値対称 (エルミート) マトリックス *A* を係数行列とする連立 1 次方程式の解を計算し、解の誤差範囲を示す。

```
call sppsvx (fact, uplo, n, nrhs, ap, afp, equed, s, b, ldb, x, ldx,
             rcond, ferr, berr, work, iwork, info)
call dppsvx (fact, uplo, n, nrhs, ap, afp, equed, s, b, ldb, x, ldx,
             rcond, ferr, berr, work, iwork, info)
call cppsvx (fact, uplo, n, nrhs, ap, afp, equed, s, b, ldb, x, ldx,
             rcond, ferr, berr, work, rwork, info)
call zppsvx (fact, uplo, n, nrhs, ap, afp, equed, s, b, ldb, x, ldx,
             rcond, ferr, berr, work, rwork, info)
```

説明

このルーチンは、コルスキー因子分解 $A=U^HU$ または $A=LL^H$ を使用して、実数または複素連立 1 次方程式 $AX = B$ の解を計算する。A はパックド形式で格納される $n \times n$ の正値対称 / エルミート・マトリックス、マトリックス *B* の列は個々の右辺、*X* の列はそれに対応する解である。

このルーチンは、解の誤差範囲と条件数の推定値も示す。

ルーチン ?ppsvx は、以下の手順を実行する。

1. *fact* = 'E' の場合は、連立方程式を平衡化するための実スケール係数 *s* を計算する。

$$\mathrm{diag}(s)*A*\mathrm{diag}(s)\ *\mathrm{diag}(s)^{-1}*X = \mathrm{diag}(s)*B$$

連立方程式が平衡化されるかどうかは、マトリックス *A* のスケーリングによって決まる。平衡化が行われる場合は、*A* は diag(*s*)*A*diag(*s*) によって上書きされ、*B* は diag(*s*)**B* によって上書きされる。

2. *fact* = 'N' または 'E' の場合は、コルスキー分解を使用して、(*fact* = 'E' の場合は平衡化の後に) マトリックス *A* を次のように因子分解する。

$A = U^H U$ (*uplo* = 'U' の場合)、または
$A = L L^H$ (*uplo* = 'L' の場合)
U は上三角マトリックスで、L は下三角マトリックスである。

3. 先頭の $i \times i$ の主小行列式が正値でない場合は、ルーチンは *info* = *i* を返す。それ以外の場合は、A の因子分解された形式を使用して、マトリックス A の条件数を推定する。条件数の逆数がマシンの精度より小さい場合は、警告として *info* = *n* + 1 を返すが、後で説明するようにルーチンは続行され、X について解を算出し、誤差範囲を計算する。

4. A の因子分解された形式を使用して、連立方程式を X について解く。

5. 計算された解のマトリックスの精度の改善を繰り返し実行し、解の誤差範囲と後退誤差の推定値を計算する。

6. 平衡化が行われている場合は、平衡化が行われる前の元の連立方程式を解くために、diag(*s*) によってマトリックス X を事前に乗算する。

入力パラメータ

fact

CHARACTER*1。'F'、'N'、または 'E' でなければならない。

ルーチンの開始時にマトリックス A の因子分解された形式が与えられるかどうか、与えられない場合は、マトリックス A を因子分解する前に平衡化を行うかどうかを指定する。

fact = 'F' の場合は、開始時に、*afp* に A の因子分解された形式が格納される。*equed* = 'Y' の場合は、マトリックス A は、*s* で指定されたスケール係数によって平衡化されている。
ap と *afp* は変更されない。

fact = 'N' の場合は、マトリックス A を *afp* にコピーし、因子分解を実行する。
fact = 'E' の場合は、必要に応じてマトリックス A を平衡化してから、*afp* にコピーし、因子分解を実行する。

uplo

CHARACTER*1。'U' または 'L' でなければならない。

A の上三角部分と下三角部分のどちらを格納するか、また A をどのように因子分解するかを指定する。

uplo = 'U' の場合は、配列 *ap* にはマトリックス A の上三角部分が格納され、A は $U^H U$ として因子分解される。

uplo = 'L' の場合は、配列 *ap* にはマトリックス A の下三角部分が格納され、A は LL^H として因子分解される。

n INTEGER。*A* の次数 ($n \geq 0$)。

nrhs INTEGER。右辺の数。*B* の列の数 ($nrhs \geq 0$)。

ap, afp, b, work
REAL (sppsvx の場合)
DOUBLE PRECISION (dppsvx の場合)
COMPLEX (cppsvx の場合)
DOUBLE COMPLEX (zppsvx の場合)

配列 : *ap(*)*, *afp(*)*, *b(ldb, *)*, *work (*)*

配列 *ap* には、元の対称 / エルミート・マトリックス A の上三角部分または下三角部分が**パックド格納形式**で格納される (「「マトリックスの格納形式」」を参照)。*fact* = 'F' で *equed* = 'Y' の場合は、*ap* には平衡化されたマトリックス diag(*s*)**A**diag(*s*) が格納されている。

配列 *afp* は、*fact* = 'F' の場合は入力引数になる。この配列には、*A* のコルスキー因子分解で得られた三角係数 *U* または *L* が、*A* と同じ格納形式で格納される。*equed* が 'N' でない場合は、*afp* は、平衡化されたマトリックス *A* の因子分解された形式になる。

配列 *b* には、マトリックス *B* が格納される。このマトリックスの列は、連立方程式の右辺である。

work (*) はワークスペース配列である。

配列 *ap* と *afp* の次元は、少なくとも max(1, *n*(*n*+1)/2) でなければならない。*b* の第 2 次元は、少なくとも max(1, *nrhs*) でなければならない。*work* の次元は、実数型の場合は少なくとも max(1, 3**n*)、複素数型の場合は少なくとも max(1, 2**n*) でなければならない。

ldb INTEGER。*b* の第 1 次元。*ldb* ≥ max(1, *n*)

equed CHARACTER*1。'N' または 'Y' でなければならない。
equed は、*fact* = 'F' の場合は入力引数になる。この引数は、実行された平衡化の形式を指定する。
equed = 'N' の場合は、平衡化は行われていない (*fact* = 'N' の場合は常に真)。
equed = 'Y' の場合は、平衡化が行われ、*A* は diag(*s*)**A**diag(*s*) で置き換えられている。

s
REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
配列 DIMENSION は (*n*)
配列 *s* には、*A* のスケール係数が格納される。この配列は、*fact* = 'F' の場合にのみ入力引数になり、それ以外の場合は出力引数になる。
equed = 'N' の場合は、*s* は使用されない。
fact = 'F' で、*equed* = 'Y' の場合は、*s* の各成分は正の値でなければならない。

ldx
INTEGER。出力配列 *x* の第 1 次元。*ldx* ≥ max(1, *n*)

iwork
INTEGER
ワークスペース配列 DIMENSION は少なくとも max(1, *n*)。実数型でのみ使用される。

rwork
REAL (cppsvx の場合)
DOUBLE PRECISION (zppsvx の場合)
ワークスペース配列 DIMENSION は少なくとも max(1, *n*)。複素数型でのみ使用される。

出力パラメータ

x
REAL (sppsvx の場合)
DOUBLE PRECISION (dppsvx の場合)
COMPLEX (cppsvx の場合)
DOUBLE COMPLEX (zppsvx の場合)
配列 DIMENSION は (*ldx*, *)
info = 0 または *info* = *n*+1 の場合は、配列 *x* には、元の連立方程式の解のマトリックス *X* が格納される。ただし、*equed* = 'Y' の場合は、*A* と *B* は終了時に修正され、平衡化された連立方程式の解は diag(*s*)$^{-1}$*X* になる。
x の第 2 次元は、少なくとも max(1, *nrhs*) でなければならない。

ap
配列 *ap* は、*fact* = 'F' または 'N' の場合、または *fact* = 'E' で *equed* = 'N' の場合は、終了時に変更されない。
fact = 'E' で *equed* = 'Y' の場合は、*A* は diag(*s*)*A*diag(*s*) によって上書きされる。

afp

fact = 'N' または 'E' の場合は、*afp* は出力引数になる。この引数は終了時に、元のマトリックス A (*fact* = 'N' の場合) または平衡化されたマトリックス A (*fact* = 'E' の場合) のコルスキー因子分解 $A=U^HU$ または $A=LL^H$ で得られた三角係数 U または L を返す。平衡化されたマトリックスの形式は、*ap* の説明を参照のこと。

b

equed = 'Y' の場合は、diag(*s*)*B によって上書きされる。

equed = 'N' の場合は変更されない。

s

この配列は、*fact* ≠ 'F' の場合は出力引数になる。「入力引数」の項の *s* の説明を参照。

rcond

REAL (単精度の場合)

DOUBLE PRECISION (倍精度の場合)

平衡化後のマトリックス A の条件数の逆数の推定値 (平衡化が行われる場合)。*rcond* がマシンの精度より小さい場合 (特に、*rcond* = 0) の場合) は、マトリックスは有効な精度で特異になる。この条件は、*info* > 0 のリターン・コードで示される。

ferr, berr

REAL (単精度の場合)

DOUBLE PRECISION (倍精度の場合)

配列 DIMENSION は少なくとも max(1, *nrhs*)。各解ベクトルの成分ごとの前進誤差と相対後退誤差を格納する。

equed

fact ≠ 'F' の場合は、*equed* は出力引数になる。この引数は、実行された平衡化の形式を指定する (「入力引数」の項の *equed* の説明を参照)。

info

INTEGER。*info* = 0 の場合は、ルーチンの実行は正常に終了した。

info = -*i* の場合は、*i* 番目のパラメータの値が不正である。

info = *i* で、*i* ≤ *n* の場合は、次数 *i* の先頭の小行列式 (したがって、マトリックス A そのもの) が正値になっていないため、因子分解を実行できず、解と誤差範囲は計算できなかった。

rcond = 0 が返される。

info = *i* で、*i* = *n* +1 の場合は、U は特異でないが、*rcond* がマシンの精度より小さいため、マトリックスは有効な精度で特異になる。しかし、この場合は、解と誤差範囲が計算される。これは計算された解の精度が、*rcond* の値から想定される精度より高くなる場合があるためである。

?pbsv

正値対称またはエルミート・バンド・
マトリックス A を係数行列とする、
複数の右辺を持つ連立 1 次方程式を解く。

```
call spbsv (uplo, n, kd, nrhs, ab, ldab, b, ldb, info)
call dpbsv (uplo, n, kd, nrhs, ab, ldab, b, ldb, info)
call cpbsv (uplo, n, kd, nrhs, ab, ldab, b, ldb, info)
call zpbsv (uplo, n, kd, nrhs, ab, ldab, b, ldb, info)
```

説明

このルーチンは、実数または複素連立 1 次方程式 $AX = B$ を、X について解く。A は $n \times n$ の正値対称 / エルミート・バンド・マトリックス、マトリックス B の列は個々の右辺、X の列はそれに対応する解である。

コルスキー因子分解を使用して、A を $A = U^H U$ (*uplo* = 'U' の場合) または $A = LL^H$ (*uplo* = 'L' の場合) として因子分解する。U は A と同じ数の優対角成分を持つ上三角バンド・マトリックスで、L は A と同じ数の劣対角成分を持つ下三角バンド・マトリックスである。次に、A の因子分解された形式を使用して、連立方程式 $AX = B$ を解く。

入力パラメータ

uplo CHARACTER*1。'U' または 'L' でなければならない。

配列 *ab* に A の上三角部分と下三角部分のどちらを格納するか、また A をどのように因子分解するかを指定する。

uplo = 'U' の場合は、配列 *ab* にはマトリックス A の上三角部分が格納され、A は $U^H U$ として因子分解される。

uplo = 'L' の場合は、配列 *ab* にはマトリックス A の下三角部分が格納され、A は LL^H として因子分解される。

n INTEGER。A の次数 ($n \geq 0$)。

kd INTEGER。マトリックス *A* の優対角成分の数 (*uplo* = 'U' の場合) または劣対角成分の数 (*uplo* = 'L' の場合) (*kd* ≥ 0)

nrhs INTEGER。右辺の数。*B* の列の数 (*nrhs* ≥ 0)

ab, b REAL (spbsv の場合)
DOUBLE PRECISION (dpbsv の場合)
COMPLEX (cpbsv の場合)
DOUBLE COMPLEX (zpbsv の場合)
配列 : *ab*(*ldab, **), *b*(*ldb, **)
配列 *ab* には、*uplo* の指定に従って、マトリックス *A* の上三角部分または下三角部分が**バンド形式**で格納される (「「マトリックスの格納形式」」を参照)。
ab の第 2 次元は、少なくとも max(1, *n*) でなければならない。
配列 *b* には、マトリックス *B* が格納される。このマトリックスの列は、連立方程式の右辺である。
b の第 2 次元は、少なくとも max(1, *nrhs*) でなければならない。

ldab INTEGER。配列 *ab* の第 1 次元。(*ldab* ≥ *kd* +1)

ldb INTEGER。*b* の第 1 次元。*ldb* ≥ max(1, *n*)

出力パラメータ

ab *uplo* の指定に従って、(バンド形式の) *A* の上三角部分または下三角部分が、コルスキー係数 *U* または *L* によって、*A* と同じ格納形式で上書きされる。

b 解のマトリックス *X* によって上書きされる。

info INTEGER。*info* = 0 の場合は、ルーチンの実行は正常に終了した。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。
info = *i* の場合は、次数 *i* の先頭の小行列式 (したがって、マトリックス *A* そのもの) が正値になっていないため、因子分解を実行できず、解は計算されなかった。

?pbsvx

コルスキー因子分解を使用して、正値対称 (エルミート) バンド・マトリックス *A* を係数行列とする連立 1 次方程式の解を計算し、解の誤差範囲を示す。

```
call spbsvx (fact, uplo, n, kd, nrhs, ab, ldab, afb, ldafb, equed,
             s, b, ldb, x, ldx, rcond, ferr, berr, work, iwork, info)
call dpbsvx (fact, uplo, n, kd, nrhs, ab, ldab, afb, ldafb, equed,
             s, b, ldb, x, ldx, rcond, ferr, berr, work, iwork, info)
call cpbsvx (fact, uplo, n, kd, nrhs, ab, ldab, afb, ldafb, equed,
             s, b, ldb, x, ldx, rcond, ferr, berr, work, iwork, info)
call zpbsvx (fact, uplo, n, kd, nrhs, ab, ldab, afb, ldafb, equed,
             s, b, ldb, x, ldx, rcond, ferr, berr, work, iwork, info)
```

説明

このルーチンは、コルスキー因子分解 $A=U^HU$ または $A=LL^H$ を使用して、実数または複素連立 1 次方程式 $AX = B$ の解を計算する。A は $n \times n$ の正値対称またはエルミート・バンド・マトリックス、マトリックス *B* の列は個々の右辺、*X* の列はそれに対応する解である。

このルーチンは、解の誤差範囲と条件数の推定値も示す。

ルーチン ?pbsvx は、以下の手順を実行する。

1. *fact* = 'E' の場合は、連立方程式を平衡化するための実スケール係数 *s* を計算する。

$$\mathrm{diag}(s)*A*\mathrm{diag}(s)\ *\mathrm{diag}(s)^{-1}*X = \mathrm{diag}(s)*B$$

連立方程式が平衡化されるかどうかは、マトリックス *A* のスケーリングによって決まる。平衡化が行われる場合は、*A* は diag(*s*)*A*diag(*s*) によって上書きされ、*B* は diag(*s*)*B によって上書きされる。

2. *fact* = 'N' または 'E' の場合は、コルスキー分解を使用して、(*fact* = 'E' の場合は平衡化の後に) マトリックス *A* を次のように因子分解する。

$A = U^H U$ (*uplo* = 'U' の場合)、または
$A = L L^H$ (*uplo* = 'L' の場合)
U は上三角バンド・マトリックスで、L は下三角バンド・マトリックスである。

3. 先頭の $i \times i$ の主小行列式が正値でない場合は、ルーチンは *info* = *i* を返す。それ以外の場合は、A の因子分解された形式を使用して、マトリックス A の条件数を推定する。条件数の逆数がマシンの精度より小さい場合は、警告として *info* = *n* + 1 を返すが、後で説明するようにルーチンは続行され、X について解を算出し、誤差範囲を計算する。

4. A の因子分解された形式を使用して、連立方程式を X について解く。

5. 計算された解のマトリックスの精度の改善を繰り返し実行し、解の誤差範囲と後退誤差の推定値を計算する。

6. 平衡化が行われている場合は、平衡化が行われる前の元の連立方程式を解くために、diag(*s*) によってマトリックス X を事前に乗算する。

入力パラメータ

fact

CHARACTER*1。'F'、'N'、または 'E' でなければならない。

ルーチンの開始時にマトリックス A の因子分解された形式が与えられるかどうか、与えられない場合は、マトリックス A を因子分解する前に平衡化を行うかどうかを指定する。

fact = 'F' の場合は、開始時に、*afb* に A の因子分解された形式が格納される。*equed* = 'Y' の場合は、マトリックス A は、*s* で指定されたスケール係数によって平衡化されている。
ab と *afb* は変更されない。

fact = 'N' の場合は、マトリックス A を *afb* にコピーし、因子分解を実行する。
fact = 'E' の場合は、必要に応じてマトリックス A を平衡化してから、*afb* にコピーし、因子分解を実行する。

uplo

CHARACTER*1。'U' または 'L' でなければならない。

A の上三角部分と下三角部分のどちらを格納するか、また A をどのように因子分解するかを指定する。

uplo = 'U' の場合は、配列 ab にはマトリックス A の上三角部分が格納され、A は U^HU として因子分解される。
uplo = 'L' の場合は、配列 ab にはマトリックス A の下三角部分が格納され、A は LL^H として因子分解される。

n　INTEGER。A の次数 ($n \geq 0$)。

kd　INTEGER。マトリックス A の優対角成分または劣対角成分の数 ($kd \geq 0$)

nrhs　INTEGER。右辺の数。B の列の数 ($nrhs \geq 0$)。

ab, afb, b, work　REAL (spbsvx の場合)
DOUBLE PRECISION (dpbsvx の場合)
COMPLEX (cpbsvx の場合)
DOUBLE COMPLEX (zpbsvx の場合)

配列 : ab(ldab, *), afb(ldab, *), b(ldb, *), work(*)

配列 ab には、マトリックス A の上三角部分または下三角部分がバンド形式で格納される (「「マトリックスの格納形式」を参照)。
fact = 'F' で equed = 'Y' の場合は、ab には平衡化されたマトリックス diag(s)*A*diag(s) が格納されている。ab の第 2 次元は、少なくとも max(1, n) でなければならない。

配列 afb は、fact = 'F' の場合は入力引数になる。この配列には、バンド・マトリックス A のコルスキー因子分解で得られた三角係数 U または L が、A と同じ格納形式で格納される。equed = 'Y' の場合は、afb は、平衡化されたマトリックス A の因子分解された形式になる。
afb の第 2 次元は、少なくとも max(1, n) でなければならない。

配列 b には、マトリックス B が格納される。このマトリックスの列は、連立方程式の右辺である。b の第 2 次元は、少なくとも max(1, nrhs) でなければならない。

work(*) はワークスペース配列である。
work の次元は、実数型の場合は少なくとも max(1, 3*n)、複素数型の場合は少なくとも max(1, 2*n) でなければならない。

ldab　INTEGER。ab の第 1 次元。ldab ≥ kd+1

ldafb　INTEGER。afb の第 1 次元。ldafb ≥ kd+1

ldb INTEGER。*b* の第 1 次元。*ldb* ≥ max(1, *n*)

equed CHARACTER*1。'N' または 'Y' でなければならない。
equed は、*fact* = 'F' の場合は入力引数になる。この引数は、実行された平衡化の形式を指定する。
equed = 'N' の場合は、平衡化は行われていない (*fact* = 'N' の場合は常に真)。
equed = 'Y' の場合は、平衡化が行われ、*A* は diag(*s*)*A*diag(*s*) で置き換えられている。

s REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
配列 DIMENSION は (*n*)
配列 *s* には、*A* のスケール係数が格納される。この配列は、*fact* = 'F' の場合にのみ入力引数になり、それ以外の場合は出力引数になる。
equed = 'N' の場合は、*s* は使用されない。
fact = 'F' で、*equed* = 'Y' の場合は、*s* の各成分は正の値でなければならない。

ldx INTEGER。出力配列 *x* の第 1 次元。*ldx* ≥ max(1, *n*)

iwork INTEGER
ワークスペース配列 DIMENSION は少なくとも max(1, *n*)。実数型でのみ使用される。

rwork REAL (cpbsvx の場合)
DOUBLE PRECISION (zpbsvx の場合)
ワークスペース配列 DIMENSION は少なくとも max(1, *n*)。複素数型でのみ使用される。

出力パラメータ

x REAL (spbsvx の場合)
DOUBLE PRECISION (dpbsvx の場合)
COMPLEX (cpbsvx の場合)
DOUBLE COMPLEX (zpbsvx の場合)
配列 DIMENSION は (*ldx*, *)

	info = 0 または *info* = *n*+1 の場合は、配列 *x* には、元の連立方程式の解のマトリックス *X* が格納される。ただし、*equed* = 'Y' の場合は、*A* と *B* は終了時に修正され、平衡化された連立方程式の解は diag(*s*)$^{-1}$**X* になる。 *x* の第 2 次元は、少なくとも max(1, *nrhs*) でなければならない。
ab	終了時に、*fact* = 'E' で *equed* = 'Y' の場合は、*A* は diag(*s*)**A**diag(*s*) によって上書きされる。
afb	*fact* = 'N' または 'E' の場合は、*afb* は出力引数になる。この引数は終了時に、元のマトリックス *A* (*fact* = 'N' の場合) または平衡化されたマトリックス *A* (*fact* = 'E' の場合) のコルスキー因子分解 $A=U^HU$ または $A=LL^H$ で得られた三角係数 *U* または *L* を返す。平衡化されたマトリックスの形式は、*ab* の説明を参照のこと。
b	*equed* = 'Y' の場合は、diag(*s*)**B* によって上書きされる。 *equed* = 'N' の場合は変更されない。
s	この配列は、*fact* ≠ 'F' の場合は出力引数になる。「入力引数」の項の *s* の説明を参照。
rcond	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 平衡化後のマトリックス *A* の条件数の逆数の推定値 (平衡化が行われる場合)。*rcond* がマシンの精度より小さい場合 (特に、*rcond* = 0 の場合) は、マトリックスは有効な精度で特異になる。この条件は、*info* > 0 のリターン・コードで示される。
ferr, berr	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列 DIMENSION は少なくとも max(1, *nrhs*)。各解ベクトルの成分ごとの前進誤差と相対後退誤差を格納する。
equed	*fact* ≠ 'F' の場合は、*equed* は出力引数になる。この引数は、実行された平衡化の形式を指定する (「入力引数」の項の *equed* の説明を参照)。

info INTEGER。*info* = 0 の場合は、ルーチンの実行は正常に終了した。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。
info = *i* で、*i* ≤ *n* の場合は、次数 *i* の先頭の小行列式 (したがって、マトリックス *A* そのもの) が正値になっていないため、因子分解を実行できず、解と誤差範囲は計算できなかった。*rcond* = 0 が返される。
info = *i* で、*i* = *n* +1 の場合は、*U* は特異でないが、*rcond* がマシンの精度より小さいため、マトリックスは有効な精度で特異になる。しかし、この場合は、解と誤差範囲が計算される。これは計算された解の精度が、*rcond* の値から想定される精度より高くなる場合があるためである。

?ptsv

正値対称またはエルミート三重対角
マトリックス *A* を係数行列とする、
複数の右辺を持つ連立 1 次方程式を解く。

```
call sptsv (n, nrhs, d, e, b, ldb, info)
call dptsv (n, nrhs, d, e, b, ldb, info)
call cptsv (n, nrhs, d, e, b, ldb, info)
call zptsv (n, nrhs, d, e, b, ldb, info)
```

説明

このルーチンは、実数または複素連立 1 次方程式 $AX = B$ を、X について解く。A は $n \times n$ の正値対称 / エルミート三重対角マトリックス、マトリックス B の列は個々の右辺、X の列はそれに対応する解である。

A を $A = L\,D\,L^H$ として因子分解する。次に、A の因子分解された形式を使用して、連立方程式 $AX = B$ を解く。

入力パラメータ

n	INTEGER。*A* の次数 ($n \geq 0$)。
nrhs	INTEGER。右辺の数。*B* の列の数 ($nrhs \geq 0$)。
d	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列、次元は少なくとも max(1, *n*)。三重対角マトリックス *A* の対角成分を格納する。
e, b	REAL (sptsv の場合) DOUBLE PRECISION (dptsv の場合) COMPLEX (cptsv の場合) DOUBLE COMPLEX (zptsv の場合) 配列 : *e*(*n* - 1) , *b*(*ldb, **) 配列 *e* には、*A* の (*n* - 1) 個の劣対角成分が格納される。 配列 *b* には、マトリックス *B* が格納される。このマトリックスの列は、連立方程式の右辺である。 *b* の第 2 次元は、少なくとも max(1, *nrhs*) でなければならない。
ldb	INTEGER。*b* の第 1 次元。$ldb \geq \max(1, n)$

出力パラメータ

d	A の LDL^H 因子分解で得られた対角マトリックス *D* の *n* 個の対角成分によって上書きされる。
e	A の因子分解で得られた単位二重対角係数 *L* の (*n* - 1) 個の劣対角成分によって上書きされる。
b	解のマトリックス *X* によって上書きされる。
info	INTEGER。*info* = 0 の場合は、ルーチンの実行は正常に終了した。 *info* = -*i* の場合は、*i* 番目のパラメータの値が不正である。 *info* = *i* の場合は、次数 *i* の先頭の小行列式 (したがって、マトリックス *A* そのもの) が正値になっていないため、解は計算されなかった。*i* = *n* でない限り、因子分解は完了していない。

?ptsvx

因子分解 $A=LDL^H$ を使用して、正値対称 (エルミート) 三重対角マトリックス A を係数行列とする連立 1 次方程式の解を計算し、解の誤差範囲を示す。

```
call sptsvx (fact, n, nrhs, d, e, df, ef, b, ldb, x, ldx, rcond,
             ferr, berr, work, info)
call dptsvx (fact, n, nrhs, d, e, df, ef, b, ldb, x, ldx, rcond,
             ferr, berr, work, info)
call cptsvx (fact, n, nrhs, d, e, df, ef, b, ldb, x, ldx, rcond,
             ferr, berr, work, rwork, info)
call zptsvx (fact, n, nrhs, d, e, df, ef, b, ldb, x, ldx, rcond,
             ferr, berr, work, rwork, info)
```

説明

このルーチンは、コルスキー因子分解 $A=L\,D\,L^H$ を使用して、実数または複素連立 1 次方程式 $AX = B$ の解を計算する。A は $n \times n$ の正値対称またはエルミート三重対角マトリックス、マトリックス B の列は個々の右辺、X の列はそれに対応する解である。

このルーチンは、解の誤差範囲と条件数の推定値も示す。

ルーチン ?ptsvx は、以下の手順を実行する。

1. *fact* = 'N' の場合は、マトリックス A を $A = L\,D\,L^H$ として因子分解する。L は単位下二重対角マトリックスで、D は対角マトリックスである。この因子分解の形式は、$A = U^H\,D\,U$ とも見なせる。

2. 先頭の $i \times i$ の主小行列式が正値でない場合は、ルーチンは *info* = *i* を返す。それ以外の場合は、A の因子分解された形式を使用して、マトリックス A の条件数を推定する。条件数の逆数がマシンの精度より小さい場合は、警告として *info* = *n* + 1 を返すが、後で説明するようにルーチンは続行され、X について解を算出し、誤差範囲を計算する。

3. A の因子分解された形式を使用して、連立方程式を X について解く。

4. 計算された解のマトリックスの精度の改善を繰り返し実行し、解の誤差範囲と後退誤差の推定値を計算する。

入力パラメータ

fact
CHARACTER*1。'F' または 'N' でなければならない。
ルーチンの開始時にマトリックス A の因子分解された形式が与えられるかどうかを指定する。
fact = 'F' の場合は、開始時に、*df* と *ef* に A の因子分解された形式が格納される。配列 *d*、*e*、*df*、*ef* は変更されない。
fact = 'N' の場合は、マトリックス A を *df* と *ef* にコピーし、因子分解を実行する。

n
INTEGER。A の次数 ($n \geq 0$)。

nrhs
INTEGER。右辺の数。B の列の数 ($nrhs \geq 0$)。

d, df, rwork
REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
配列 : *d*(*n*), *df*(*n*), *rwork*(*n*)
配列 *d* には、三重対角マトリックス A の *n* 個の対角成分が格納される。
配列 *df* は、*fact* = 'F' の場合は入力引数になる。この配列には、開始時に、A の $L D L^H$ 因子分解で得られた対角マトリックス D の *n* 個の対角成分が格納される。
配列 *rwork* は、複素数型でのみ使用されるワークスペース配列である。

e, ef, b, work
REAL (sptsvx の場合)
DOUBLE PRECISION (dptsvx の場合)
COMPLEX (cptsvx の場合)
DOUBLE COMPLEX (zptsvx の場合)
配列 : *e*(*n* - 1), *ef*(*n* - 1), *b*(*ldb*, *), *work*(*)
配列 *e* には、三重対角マトリックス A の (*n* - 1 個の劣対角成分が格納される。

配列 *ef* は、*fact* = 'F' の場合は入力引数になる。この配列には、開始時に、A の $L D L^H$ 因子分解で得られた単位二重対角係数 L の (n - 1) 個の劣対角成分が格納される。

配列 *b* には、マトリックス B が格納される。このマトリックスの列は、連立方程式の右辺である。

配列 *work* はワークスペース配列である。*work* の次元は、実数型の場合は少なくとも 2**n*、複素数型の場合は少なくとも *n* でなければならない。

ldb INTEGER。*b* のリーディング・ディメンジョン。
ldb ≥ max(1, *n*)

ldx INTEGER。*x* のリーディング・ディメンジョン。
ldx ≥ max(1, *n*)

出力パラメータ

x REAL (sptsvx の場合)
DOUBLE PRECISION (dptsvx の場合)
COMPLEX (cptsvx の場合)
DOUBLE COMPLEX (zptsvx の場合)
配列 DIMENSION は (*ldx*, *)

info = 0 または *info* = *n*+1 の場合は、配列 *x* には、連立方程式の解のマトリックス X が格納される。*x* の第 2 次元は、少なくとも max(1, *nrhs*) でなければならない。

df, *ef* これらの配列は、*fact* = 'N' の場合は出力引数になる。
「入力引数」の項の *df*, *ef* の説明を参照。

rcond REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
平衡化後のマトリックス A の条件数の逆数の推定値 (平衡化が行われる場合)。*rcond* がマシンの精度より小さい場合 (特に、*rcond* = 0 の場合) は、マトリックスは有効な精度で特異になる。この条件は、*info* > 0 のリターン・コードで示される。

ferr, *berr* REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
配列 DIMENSION は少なくとも max(1, *nrhs*)。各解ベクトルの成分ごとの前進誤差と相対後退誤差を格納する。

info INTEGER。*info* = 0 の場合は、ルーチンの実行は正常に終了した。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。
info = *i* で、*i* ≤ *n* の場合は、次数 *i* の先頭の小行列式 (したがって、マトリックス *A* そのもの) が正値になっていないため、因子分解を実行できず、解と誤差範囲は計算できなかった。*rcond* = 0 が返される。
info = *i* で、*i* = *n* +1 の場合は、*U* は特異でないが、*rcond* がマシンの精度より小さいため、マトリックスは有効な精度で特異になる。しかし、この場合は、解と誤差範囲が計算される。これは計算された解の精度が、*rcond* の値から想定される精度より高くなる場合があるためである。

?sysv

実対称または複素対称マトリックス *A* を係数行列とする、複数の右辺を持つ連立 1 次方程式の解を計算する。

```
call ssysv (uplo, n, nrhs, a, lda, ipiv, b, ldb, work, lwork, info)
call dsysv (uplo, n, nrhs, a, lda, ipiv, b, ldb, work, lwork, info)
call csysv (uplo, n, nrhs, a, lda, ipiv, b, ldb, work, lwork, info)
call zsysv (uplo, n, nrhs, a, lda, ipiv, b, ldb, work, lwork, info)
```

説明

このルーチンは、実数または複素連立 1 次方程式 $AX = B$ を、X について解く。A は $n \times n$ の対称マトリックス、マトリックス B の列は個々の右辺、X の列はそれに対応する解である。

対角ピボット (要) 演算法を使用して、A を $A = U D U^T$ または $A = L D L^T$ として因子分解する。U (または L) は置換マトリックスと単位上 (下) 三角マトリックスの積で、D は 1×1 と 2×2 の対角ブロックを持つ対称ブロック対角マトリックスである。

次に、A の因子分解された形式を使用して、連立方程式 $AX = B$ を解く。

入力パラメータ

uplo
: CHARACTER*1。'U' または 'L' でなければならない。
A の上三角部分と下三角部分のどちらを格納するか、また A をどのように因子分解するかを指定する。
uplo = 'U' の場合は、配列 *a* にはマトリックス A の上三角部分が格納され、A は UDU^T として因子分解される。
uplo = 'L' の場合は、配列 *a* にはマトリックス A の下三角部分が格納され、A は LDL^T として因子分解される。

n
: INTEGER。A の次数 ($n \geq 0$)。

nrhs
: INTEGER。右辺の数。B の列の数 ($nrhs \geq 0$)。

a, b, work
: REAL (ssysv の場合)
DOUBLE PRECISION (dsysv の場合)
COMPLEX (csysv の場合)
DOUBLE COMPLEX (zsysv の場合)
配列 :*a*(*lda, **), *b*(*ldb, **), *work*(*lwork*)
配列 *a* には、対称マトリックス A の上三角部分または下三角部分が格納される (*uplo* を参照)。
a の第 2 次元は、少なくとも max(1, *n*) でなければならない。
配列 *b* には、マトリックス B が格納される。このマトリックスの列は、連立方程式の右辺である。
b の第 2 次元は、少なくとも max(1, *nrhs*) でなければならない。
work(*lwork*) はワークスペース配列である。

lda
: INTEGER。*a* の第 1 次元。$lda \geq \max(1, n)$

ldb
: INTEGER。*b* の第 1 次元。$ldb \geq \max(1, n)$

lwork
: INTEGER。*work* 配列のサイズ ($lwork \geq 1$)
lwork の推奨値は、「アプリケーション・ノート」を参照のこと。

出力パラメータ

a	*info* = 0 の場合は、*a* は ?sytrf による *A* の因子分解で得られたブロック対角マトリックス *D* と、係数 *U* (または *L*) の計算に使用された乗数によって上書きされる。
b	*info* = 0 の場合は、b は解のマトリックス *X* によって上書きされる。
ipiv	INTEGER 配列 DIMENSION は少なくとも max(1, *n*) ?sytrf によって決まる、交換操作と *D* のブロック構造の詳細を格納する。 *ipiv*(*i*) = *k* > 0 の場合は、d_{ii} は 1 × 1 の対角ブロックである。*A* の *i* 番目の行と列は、*k* 番目の行と列に交換されている。 *uplo* = 'U' で、*ipiv*(*i*) = *ipiv*(*i*-1) = -*m* < 0 の場合は、*D* の行 *i* と *i*-1 および列 *i* と *i*-1 に 2 × 2 のブロックが作成される。*A* の (*i*-1) 番目の行と列は、*m* 番目の行と列に交換されている。 *uplo* = 'L' で、*ipiv*(*i*) = *ipiv*(*i*+1) = -*m* < 0 の場合は、*D* の行 *i* と *i*+1 および列 *i* と *i*+1 に 2 × 2 のブロックが作成される。*A* の (*i*+1) 番目の行と列は、*m* 番目の行と列に交換されている。
work(1)	*info* = 0 の場合は、終了時に、*work*(1) には、最適なパフォーマンスを得るために必要な *lwork* の最小値が格納される。これ以降の実行には、この *lwork* の値を使用する。
info	INTEGER。*info* = 0 の場合は、ルーチンの実行は正常に終了した。 *info* = -*i* の場合は、*i* 番目のパラメータの値が不正である。 *info* = *i* の場合は、d_{ii} は 0 である。因子分解は完了したが、*D* が完全に特異であるため、解は計算されなかった。

アプリケーション・ノート

パフォーマンスを向上させるには、*lwork* = *n***blocksize* に設定する。*blocksize* は、**ブロック・アルゴリズム**の最適なパフォーマンスに必要な、マシンによって異なる値 (通常は 16 ～ 64) である。

必要なワークスペースの大きさがわからない場合は、最初の実行では *lwork* =-1 に設定する。この設定は、ワークスペースのクエリと見なされる。ルーチンは、*work* 配列の最適なサイズだけを計算し、この値を

work 配列の最初のエントリ work(1) として返す。XERBLA は、lwork に関するエラー・メッセージを生成しない。終了時に、work(1) の値を確認し、これ以降の実行にはその値を使用する。

?sysvx

対角ピボット (要) 演算による因子分解を使用して、実対称または複素対称マトリックス A を係数行列とする連立 1 次方程式の解を計算し、解の誤差範囲を示す。

```
call ssysvx (fact, uplo, n, nrhs, a, lda, af, ldaf, ipiv, b, ldb,
             x, ldx, rcond, ferr, berr, work, lwork, iwork, info)
call dsysvx (fact, uplo, n, nrhs, a, lda, af, ldaf, ipiv, b, ldb,
             x, ldx, rcond, ferr, berr, work, lwork, iwork, info)
call csysvx (fact, uplo, n, nrhs, a, lda, af, ldaf, ipiv, b, ldb,
             x, ldx, rcond, ferr, berr, work, lwork, rwork, info)
call zsysvx (fact, uplo, n, nrhs, a, lda, af, ldaf, ipiv, b, ldb,
             x, ldx, rcond, ferr, berr, work, lwork, rwork, info)
```

説明

このルーチンは、対角ピボット (要) 演算による因子分解を使用して、実数または複素連立 1 次方程式 $AX = B$ の解を計算する。A は $n \times n$ の対称マトリックス、マトリックス B の列は個々の右辺、X の列はそれに対応する解である。

このルーチンは、解の誤差範囲と条件数の推定値も示す。

ルーチン ?sysvx は、以下の手順を実行する。

1. fact = 'N' の場合は、対角ピボット (要) 演算法を使用して、マトリックス A を因子分解する。因子分解の形式は、$A = U D U^T$ または $A = L D L^T$ である。U (または L) は置換マトリックスと単位上 (下) 三角マトリックスの積で、D は 1×1 と 2×2 の対角ブロックを持つ対称ブロック対角マトリックスである。

2. $d_{i,i} = 0$ の場合、つまり D が完全に特異である場合は、ルーチンは *info* = i を返す。それ以外の場合は、A の因子分解された形式を使用して、マトリックス A の条件数を推定する。条件数の逆数がマシンの精度より小さい場合は、警告として *nfo* = *n* + 1 を返すが、後で説明するようにルーチンは続行され、X について解を算出し、誤差範囲を計算する。

3. A の因子分解された形式を使用して、連立方程式を X について解く。

4. 計算された解のマトリックスの精度の改善を繰り返し実行し、解の誤差範囲と後退誤差の推定値を計算する。

入力パラメータ

fact	CHARACTER*1。'F' または 'N' でなければならない。 ルーチンの開始時にマトリックス A の因子分解された形式が与えられるかどうかを指定する。 *fact* = 'F' の場合は、開始時に、*af* と *ipiv* に A の因子分解された形式が格納される。配列 *a*、*af*、*ipiv* は変更されない。 *fact* = 'N' の場合は、マトリックス A を *af* にコピーし、因子分解を実行する。
uplo	CHARACTER*1。'U' または 'L' でなければならない。 A の上三角部分と下三角部分のどちらを格納するか、また A をどのように因子分解するかを指定する。 *uplo* = 'U' の場合は、配列 *a* には対称マトリックス A の上三角部分が格納され、A は UDU^T として因子分解される。 *uplo* = 'L' の場合は、配列 *a* には対称マトリックス A の下三角部分が格納され、A は LDL^T として因子分解される。
n	INTEGER。A の次数 ($n \geq 0$)。
nrhs	INTEGER。右辺の数。B の列の数 ($nrhs \geq 0$)。
a, af, b, work	REAL (ssysvx の場合) DOUBLE PRECISION (dsysvx の場合) COMPLEX (csysvx の場合)

DOUBLE COMPLEX (zsysvx の場合)
配列 : a(lda, *), af(ldaf, *), b(ldb, *), work(*)
配列 a には、対称マトリックス A の上三角部分または下三角部分が格納される (uplo を参照)。
a の第 2 次元は、少なくとも max(1, n) でなければならない。

配列 af は、fact = 'F' の場合は入力引数になる。この配列には、?sytrf による因子分解 $A = U D U^T$ または $A = L D L^T$ で得られた、ブロック対角マトリックス D と、係数 U または L の計算に使用された乗数が格納される。
af の第 2 次元は、少なくとも max(1, n) でなければならない。

配列 b には、マトリックス B が格納される。このマトリックスの列は、連立方程式の右辺である。b の第 2 次元は、少なくとも max(1, nrhs) でなければならない。

work(*) は、次元 (lwork) のワークスペース配列である。

lda　INTEGER。a の第 1 次元。lda ≥ max(1, n)

ldaf　INTEGER。af の第 1 次元。ldaf ≥ max(1, n)

ldb　INTEGER。b の第 1 次元。ldb ≥ max(1, n)

ipiv　INTEGER
配列 DIMENSION は少なくとも max(1, n)
配列 ipiv は、fact = 'F' の場合は入力引数になる。この配列には、?sytrf によって決まる、交換操作と D のブロック構造の詳細が格納される。
ipiv(i) = k > 0 の場合は、d_{ii} は 1×1 の対角ブロックである。A の i 番目の行と列は、k 番目の行と列と交換されている。

uplo = 'U' で、ipiv(i) =ipiv(i-1) = -m < 0 の場合は、D の行 i と i-1 および列 i と i-1 に 2×2 のブロックが作成される。A の (i-1) 番目の行と列は、m 番目の行と列に交換されている。

uplo = 'L' で、ipiv(i) =ipiv(i+1) = -m < 0 の場合は、D の行 i と i+1 および列 i と i+1 に 2×2 のブロックが作成される。A の (i+1) 番目の行と列は、m 番目の行と列に交換されている。

ldx INTEGER。出力配列 *x* のリーディング・ディメンジョン。*ldx* ≥ max(1, *n*)

lwork INTEGER。*work* 配列のサイズ。
lwork の推奨値は、「アプリケーション・ノート」を参照のこと。

iwork INTEGER
ワークスペース配列 DIMENSION は少なくとも max(1, *n*)。実数型でのみ使用される。

rwork REAL (csysvx の場合)
DOUBLE PRECISION (zsysvx の場合)
ワークスペース配列 DIMENSION は少なくとも max(1, *n*)。複素数型でのみ使用される。

出力パラメータ

x REAL (ssysvx の場合)
DOUBLE PRECISION (dsysvx の場合)
COMPLEX (csysvx の場合)
DOUBLE COMPLEX (zsysvx の場合)
配列 DIMENSION は (*ldx*, *)
info = 0 または *info* = *n*+1 の場合は、配列 *x* には、連立方程式の解のマトリックス *X* が格納される。*x* の第 2 次元は、少なくとも max(1, *nrhs*) でなければならない。

af, *ipiv* これらの配列は、*fact* = 'N' の場合は出力引数になる。
「入力引数」の項の *af*, *ipiv* の説明を参照。

rcond REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
マトリックス *A* の条件数の逆数の推定値。*rcond* がマシンの精度より小さい場合 (特に、*rcond* = 0 の場合) は、マトリックスは有効な精度で特異になる。この条件は、*info* > 0 のリターン・コードで示される。

ferr, *berr* REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
配列 DIMENSION は少なくとも max(1, *nrhs*)。各解ベクトルの成分ごとの前進誤差と相対後退誤差を格納する。

work(1)　　*info* = 0 の場合は、終了時に、*work*(1) には、最適なパフォーマンスを得るために必要な *lwork* の最小値が格納される。これ以降の実行には、この *lwork* の値を使用する。

info　　INTEGER。*info* = 0 の場合は、ルーチンの実行は正常に終了した。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。
info = *i* で、*i* ≤ *n* の場合は、d_{ii} は完全に 0 である。因子分解は完了したが、ブロック対角マトリックス *D* が完全に特異であるため、解と誤差範囲は計算できなかった。*rcond* = 0 が返される。
info = *i* で、*i* = *n* +1 の場合は、*D* は特異でないが、*rcond* がマシンの精度より小さいため、マトリックスは有効な精度で特異になる。しかし、この場合は、解と誤差範囲が計算される。これは計算された解の精度が、*rcond* の値から想定される精度より高くなる場合があるためである。

アプリケーション・ノート

lwork は、実数型の場合は少なくとも 3**n*、複素数型の場合は少なくとも 2**n* でなければならない。パフォーマンスを向上させるには、*lwork* = *n***blocksize* に設定する。*blocksize* は、?sytrf に最適な *ブロック・サイズ*である。

必要なワークスペースの大きさがわからない場合は、最初の実行では *lwork* =-1 に設定する。この設定は、ワークスペースのクエリと見なされる。ルーチンは、*work* 配列の最適なサイズだけを計算し、この値を *work* 配列の最初のエントリ *work*(1) として返す。XERBLA は、*lwork* に関するエラー・メッセージを生成しない。終了時に、*work*(1) の値を確認し、これ以降の実行にはその値を使用する。

?hesvx

対角ピボット (要) 演算による因子分解を使用して、エルミート・マトリックス A を係数行列とする複素連立 1 次方程式の解を計算し、解の誤差範囲を示す。

```
call chesvx (fact, uplo, n, nrhs, a, lda, af, ldaf, ipiv, b, ldb,
             x, ldx, rcond, ferr, berr, work, lwork, rwork, info)
call zhesvx (fact, uplo, n, nrhs, a, lda, af, ldaf, ipiv, b, ldb,
             x, ldx, rcond, ferr, berr, work, lwork, rwork, info)
```

説明

このルーチンは、対角ピボット (要) 演算による因子分解を使用して、複素連立 1 次方程式 $AX = B$ の解を計算する。A は $n \times n$ のエルミート・マトリックス、マトリックス B の列は個々の右辺、X の列はそれに対応する解である。

このルーチンは、解の誤差範囲と条件数の推定値も示す。

ルーチン ?hesvx は、以下の手順を実行する。

1. *fact* = 'N' の場合は、対角ピボット (要) 演算法を使用して、マトリックス A を因子分解する。因子分解の形式は、$A = U D U^H$ または $A = L D L^H$ である。U (または L) は置換マトリックスと単位上 (下) 三角マトリックスの積で、D は 1×1 と 2×2 の対角ブロックを持つエルミート・ブロック対角マトリックスである。

2. $d_{i,i} = 0$ の場合、つまり D が完全に特異である場合は、ルーチンは *info* = i を返す。それ以外の場合は、A の因子分解された形式を使用して、マトリックス A の条件数を推定する。条件数の逆数がマシンの精度より小さい場合は、警告として *info* = $n + 1$ を返すが、後で説明するようにルーチンは続行され、X について解を算出し、誤差範囲を計算する。

3. A の因子分解された形式を使用して、連立方程式を X について解く。

4. 計算された解のマトリックスの精度の改善を繰り返し実行し、解の誤差範囲と後退誤差の推定値を計算する。

入力パラメータ

fact CHARACTER*1。'F' または 'N' でなければならない。

ルーチンの開始時にマトリックス A の因子分解された形式が与えられるかどうかを指定する。

fact = 'F' の場合は、開始時に、*af* と *ipiv* に A の因子分解された形式が格納される。配列 *a*、*af*、*ipiv* は変更されない。

fact = 'N' の場合は、マトリックス A を *af* にコピーし、因子分解を実行する。

uplo CHARACTER*1。'U' または 'L' でなければならない。

A の上三角部分と下三角部分のどちらを格納するか、また A をどのように因子分解するかを指定する。

uplo = 'U' の場合は、配列 *a* にはエルミート・マトリックス A の上三角部分が格納され、A は UDU^H として因子分解される。

uplo = 'L' の場合は、配列 *a* にはエルミート・マトリックス A の下三角部分が格納され、A は LDL^H として因子分解される。

n INTEGER。A の次数 ($n \geq 0$)。

nrhs INTEGER。右辺の数。B の列の数 ($nrhs \geq 0$)。

a, af, b, work COMPLEX (chesvx の場合)

DOUBLE COMPLEX (zhesvx の場合)

配列 : *a(lda, *)*, *af(ldaf, *)*, *b(ldb, *)*, *work(*)*

配列 *a* には、エルミート・マトリックス A の上三角部分または下三角部分が格納される (*uplo* を参照)。*a* の第 2 次元は、少なくとも max(1, *n*) でなければならない。

配列 *af* は、*fact* = 'F' の場合は入力引数になる。この配列には、?hetrf による因子分解 $A = U\,D\,U^H$ または $A = L\,D\,L^H$ で得られた、ブロック対角マトリックス D と、係数 U または L の計算に使用された乗数が格納される。

af の第 2 次元は、少なくとも max(1, *n*) でなければならない。

配列 *b* には、マトリックス *B* が格納される。このマトリックスの列は、連立方程式の右辺である。*b* の第 2 次元は、少なくとも max(1, *nrhs*) でなければならない。

work(*) は、次元 (*lwork*) のワークスペース配列である。

lda INTEGER。*a* の第 1 次元。*lda* ≥ max(1, *n*)

ldaf INTEGER。*af* の第 1 次元。*ldaf* ≥ max(1, *n*)

ldb INTEGER。*b* の第 1 次元。*ldb* ≥ max(1, *n*)

ipiv INTEGER

配列 DIMENSION は少なくとも max(1, *n*)
配列 *ipiv* は、*fact* = 'F' の場合は入力引数になる。この配列には、?hetrf によって決まる、交換操作と *D* のブロック構造の詳細が格納される。
ipiv(i) = *k* > 0 の場合は、d_{ii} は 1 × 1 の対角ブロックである。*A* の *i* 番目の行と列は、*k* 番目の行と列に交換されている。

uplo = 'U' で、*ipiv(i)* =*ipiv(i-1)* = -*m* < 0 の場合は、*D* の行 *i* と *i*-1 および列 *i* と *i*-1 に 2 × 2 のブロックが作成される。*A* の (*i*-1) 番目の行と列は、*m* 番目の行と列に交換されている。

uplo = 'L' で、*ipiv(i)* =*ipiv(i+1)* = -*m* < 0 の場合は、*D* の行 *i* と *i*+1 および列 *i* と *i*+1 に 2 × 2 のブロックが作成される。*A* の (*i*+1) 番目の行と列は、*m* 番目の行と列に交換されている。

ldx INTEGER。出力配列 *x* のリーディング・ディメンジョン。*ldx* ≥ max(1, *n*)

lwork INTEGER。*work* 配列のサイズ。
lwork の推奨値は、「アプリケーション・ノート」を参照のこと。

rwork REAL (chesvx の場合)
DOUBLE PRECISION (zhesvx の場合)
ワークスペース配列 DIMENSION は少なくとも max(1, *n*)。

出力パラメータ

x COMPLEX (chesvx の場合)
DOUBLE COMPLEX (zhesvx の場合)
配列 DIMENSION は (*ldx*, *)

info = 0 または *info* = *n*+1 の場合は、配列 *x* には、連立方程式の解のマトリックス *X* が格納される。*x* の第 2 次元は、少なくとも max(1, *nrhs*) でなければならない。

af, ipiv
これらの配列は、*fact* = 'N' の場合は出力引数になる。
「入力引数」の項の *af, ipiv* の説明を参照。

rcond
REAL (chesvx の場合)
DOUBLE PRECISION (zhesvx の場合)
マトリックス *A* の条件数の逆数の推定値。*rcond* がマシンの精度より小さい場合 (特に、*rcond* = 0 の場合) は、マトリックスは有効な精度で特異になる。この条件は、*info* > 0 のリターン・コードで示される。

ferr, berr
REAL (chesvx の場合)
DOUBLE PRECISION (zhesvx の場合)
配列 DIMENSION は少なくとも max(1, *nrhs*)。各解ベクトルの成分ごとの前進誤差と相対後退誤差を格納する。

work(1)
info = 0 の場合は、終了時に、*work*(1) には、最適なパフォーマンスを得るために必要な *lwork* の最小値が格納される。これ以降の実行には、この *lwork* の値を使用する。

info
INTEGER。*info* = 0 の場合は、ルーチンの実行は正常に終了した。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。
info = *i* で、*i* ≤ *n* の場合は、d_{ii} は完全に 0 である。因子分解は完了したが、ブロック対角マトリックス *D* が完全に特異であるため、解と誤差範囲は計算できなかった。*rcond* = 0 が返される。
info = *i* で、*i* = *n* +1 の場合は、*D* は特異でないが、*rcond* がマシンの精度より小さいため、マトリックスは有効な精度で特異になる。しかし、この場合は、解と誤差範囲が計算される。これは計算された解の精度が、*rcond* の値から想定される精度より高くなる場合があるためである。

アプリケーション・ノート

lwork の値は、少なくとも 2*n でなければならない。パフォーマンスを向上させるには、*lwork* = *n***blocksize* に設定する。*blocksize* は、?hetrf に最適なブロック・サイズである。

必要なワークスペースの大きさがわからない場合は、最初の実行では *lwork* =-1 に設定する。この設定は、ワークスペースのクエリと見なされる。ルーチンは、*work* 配列の最適なサイズだけを計算し、この値を *work* 配列の最初のエントリ *work*(1) として返す。XERBLA は、*lwork* に関するエラー・メッセージを生成しない。終了時に、*work*(1) の値を確認し、これ以降の実行にはその値を使用する。

?hesv

エルミート・マトリックス *A* を
係数行列とする、複数の右辺を持つ
連立 1 次方程式の解を計算する。

```
call chesv (uplo, n, nrhs, a, lda, ipiv, b, ldb, work, lwork, info)
call zhesv (uplo, n, nrhs, a, lda, ipiv, b, ldb, work, lwork, info)
```

説明

このルーチンは、実数または複素連立 1 次方程式 $AX = B$ を、X について解く。A は $n \times n$ の対称マトリックス、マトリックス B の列は個々の右辺、X の列はそれに対応する解である。

対角ピボット (要) 演算法を使用して、A を $A = U D U^H$ または $A = L D L^H$ として因子分解する。U (または L) は置換マトリックスと単位上 (下) 三角マトリックスの積で、D は 1×1 と 2×2 の対角ブロックを持つエルミート・ブロック対角マトリックスである。

次に、A の因子分解された形式を使用して、連立方程式 $AX = B$ を解く。

入力パラメータ

uplo CHARACTER*1。'U' または 'L' でなければならない。

A の上三角部分と下三角部分のどちらを格納するか、また A をどのように因子分解するかを指定する。

uplo = 'U' の場合は、配列 a にはマトリックス A の上三角部分が格納され、A は UDU^H として因子分解される。

uplo = 'L' の場合は、配列 a にはマトリックス A の下三角部分が格納され、A は LDL^H として因子分解される。

n　INTEGER。A の次数 ($n \geq 0$)。

nrhs　INTEGER。右辺の数。B の列の数 ($nrhs \geq 0$)。

a, b, work　COMPLEX (chesv の場合)
DOUBLE COMPLEX (zhesv の場合)
配列 : a(lda, *), b(ldb, *), work(lwork)
配列 a には、エルミート・マトリックス A の上三角部分または下三角部分が格納される (uplo を参照)。
a の第 2 次元は、少なくとも max(1, n) でなければならない。
配列 b には、マトリックス B が格納される。このマトリックスの列は、連立方程式の右辺である。
b の第 2 次元は、少なくとも max(1, nrhs) でなければならない。
work(lwork) はワークスペース配列である。

lda　INTEGER。a の第 1 次元。$lda \geq \max(1, n)$

ldb　INTEGER。b の第 1 次元。$ldb \geq \max(1, n)$

lwork　INTEGER。work 配列のサイズ ($lwork \geq 1$)
lwork の推奨値は、「アプリケーション・ノート」を参照のこと。

出力パラメータ

a　info = 0 の場合は、a は、?hetrf による A の因子分解で得られた、ブロック対角マトリックス D と、係数 U (または L) の計算に使用された乗数によって上書きされる。

b　info = 0 の場合は、b は解のマトリックス X によって上書きされる。

ipiv　INTEGER
配列 DIMENSION は少なくとも max(1, n)
?hetrf によって決まる、交換操作と D のブロック構造の詳細を格納する。

`ipiv(i)` = `k` > 0 の場合は、d_{ii} は 1 × 1 の対角ブロックである。*A* の `i` 番目の行と列は、`k` 番目の行と列に交換されている。

`uplo` = `'U'` で、`piv(i)` =`ipiv(i-1)` = `-m` < 0 の場合は、*D* の行 `i` と `i-1` および列 `i` と `i-1` に 2 × 2 のブロックが作成される。*A* の (`i-1`) 番目の行と列は、`m` 番目の行と列に交換されている。

`uplo` = `'L'` で、`ipiv(i)` =`ipiv(i+1)` = `-m` < 0 の場合は、*D* の行 `i` と `i+1` および列 `i` と `i+1` に 2 × 2 のブロックが作成される。*A* の (`i+1`) 番目の行と列は、`m` 番目の行と列に交換されている。

`work(1)` `info` = 0 の場合は、終了時に、`work(1)` には、最適なパフォーマンスを得るために必要な `lwork` の最小値が格納される。これ以降の実行には、この `lwork` の値を使用する。

`info` `INTEGER`。`info` = 0 の場合は、ルーチンの実行は正常に終了した。

`info` = `-i` の場合は、`i` 番目のパラメータの値が不正である。

`info` = `i` の場合は、d_{ii} は 0 である。因子分解は完了したが、*D* が完全に特異であるため、解は計算されなかった。

アプリケーション・ノート

パフォーマンスを向上させるには、`lwork` = `n`**blocksize* に設定する。*blocksize* は、**ブロック・アルゴリズム**の最適なパフォーマンスに必要な、マシンによって異なる値 (通常は 16 ～ 64) である。

必要なワークスペースの大きさがわからない場合は、最初の実行では `lwork` =-1 に設定する。この設定は、ワークスペースのクエリと見なされる。ルーチンは、`work` 配列の最適なサイズだけを計算し、この値を `work` 配列の最初のエントリ `work(1)` として返す。XERBLA は、`lwork` に関するエラー・メッセージを生成しない。終了時に、`work(1)` の値を確認し、これ以降の実行にはその値を使用する。

?spsv

パックド形式で格納される実対称または複素対称マトリックス A を係数行列とする、複数の右辺を持つ連立 1 次方程式の解を計算する。

```
call sspsv (uplo, n, nrhs, ap, ipiv, b, ldb, info)
call dspsv (uplo, n, nrhs, ap, ipiv, b, ldb, info)
call cspsv (uplo, n, nrhs, ap, ipiv, b, ldb, info)
call zspsv (uplo, n, nrhs, ap, ipiv, b, ldb, info)
```

説明

このルーチンは、実数または複素連立 1 次方程式 $AX = B$ を、X について解く。A はパックド形式で格納される $n \times n$ の対称マトリックス、マトリックス B の列は個々の右辺、X の列はそれに対応する解である。

対角ピボット (要) 演算法を使用して、A を $A = U D U^T$ または $A = L D L^T$ として因子分解する。U (または L) は置換マトリックスと単位上 (下) 三角マトリックスの積で、D は 1×1 と 2×2 の対角ブロックを持つ対称ブロック対角マトリックスである。

次に、A の因子分解された形式を使用して、連立方程式 $AX = B$ を解く。

入力パラメータ

uplo CHARACTER*1。'U' または 'L' でなければならない。

A の上三角部分と下三角部分のどちらを格納するか、また A をどのように因子分解するかを指定する。

uplo = 'U' の場合は、配列 *ap* にはマトリックス A の上三角部分が格納され、A は UDU^T として因子分解される。

uplo = 'L' の場合は、配列 *ap* にはマトリックス A の下三角部分が格納され、A は LDL^T として因子分解される。

n INTEGER。A の次数 ($n \geq 0$)。

nrhs INTEGER。右辺の数。B の列の数 ($nrhs \geq 0$)。

ap, b

REAL (sspsv の場合)
DOUBLE PRECISION (dspsv の場合)
COMPLEX (cspsv の場合)
DOUBLE COMPLEX (zspsv の場合)
配列 : *ap*(*), *b*(*ldb*, *)
ap の次元は、少なくとも max(1, *n*(*n*+1)/2) でなければならない。
配列 *ap* には、*uplo* の指定に従って、係数 *U* または *L* が**パックド格納形式**で格納される (「「マトリックスの格納形式」」を参照)。
配列 *b* には、マトリックス *B* が格納される。このマトリックスの列は、連立方程式の右辺である。
b の第 2 次元は、少なくとも max(1, *nrhs*) でなければならない。

ldb

INTEGER。*b* の第 1 次元。*ldb* ≥ max(1, *n*)

出力パラメータ

ap

?sptrf による *A* の因子分解で得られた、ブロック対角マトリックス *D* と、係数 *U* (または *L*) の計算に使用された乗数が、*A* と同じ格納形式で、パックド形式の三角マトリックスとして格納される 。

b

info = 0 の場合は、*b* は解のマトリックス *X* によって上書きされる。

ipiv

INTEGER
配列 DIMENSION は少なくとも max(1, *n*)
?sptrf によって決まる、交換操作と *D* のブロック構造の詳細を格納する。
ipiv(*i*) = *k* > 0 の場合は、d_{ii} は 1 × 1 のブロックである。*A* の *i* 番目の行と列は、*k* 番目の行と列に交換されている。
uplo = 'U' で、*ipiv*(*i*) =*ipiv*(*i*-1) = -*m* < 0 の場合は、*D* の行 *i* と *i*-1 および列 *i* と *i*-1 に 2 × 2 のブロックが作成される。*A* の (*i*-1) 番目の行と列は、*m* 番目の行と列に交換されている 。
uplo = 'L' で、*ipiv*(*i*) =*ipiv*(*i*+1) = -*m* < 0 の場合は、*D* の行 *i* と *i*+1 および列 *i* と *i*+1 に 2 × 2 のブロックが作成される。*A* の (*i*+1) 番目の行と列は、*m* 番目の行と列に交換されている。

info INTEGER。*info* = 0 の場合は、ルーチンの実行は正常に終了した。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。
info = *i* の場合は、d_{ii} は 0 である。因子分解は完了したが、D が完全に特異であるため、解は計算されなかった。

?spsvx

対角ピボット (要) 演算による因子分解を使用して、パックド形式で格納される実対称または複素対称マトリックス A を係数行列とする連立 1 次方程式の解を計算し、解の誤差範囲を示す。

```
call sspsvx (fact, uplo, n, nrhs, ap, afp, ipiv, b, ldb, x, ldx,
             rcond, ferr, berr, work, iwork, info)
call dspsvx (fact, uplo, n, nrhs, ap, afp, ipiv, b, ldb, x, ldx,
             rcond, ferr, berr, work, iwork, info)
call cspsvx (fact, uplo, n, nrhs, ap, afp, ipiv, b, ldb, x, ldx,
             rcond, ferr, berr, work, rwork, info)
call zspsvx (fact, uplo, n, nrhs, ap, afp, ipiv, b, ldb, x, ldx,
             rcond, ferr, berr, work, rwork, info)
```

説明

このルーチンは、対角ピボット (要) 演算による因子分解を使用して、実数または複素連立 1 次方程式 $AX = B$ の解を計算する。A はパックド形式で格納される $n \times n$ の対称マトリックス、マトリックス B の列は個々の右辺、X の列はそれに対応する解である。

このルーチンは、解の誤差範囲と条件数の推定値も示す。

ルーチン `?spsvx` は、以下の手順を実行する。

1. `fact` = 'N' の場合は、対角ピボット (要) 演算法を使用して、マトリックス A を因子分解する。因子分解の形式は、$A = U D U^T$ または $A = L D L^T$ である。U (または L) は置換マトリックスと単位上 (下) 三角マトリックスの積で、D は 1×1 と 2×2 の対角ブロックを持つ対称ブロック対角マトリックスである。

2. $d_{i,i} = 0$ の場合、つまり D が完全に特異である場合は、ルーチンは `info` = `i` を返す。それ以外の場合は、A の因子分解された形式を使用して、マトリックス A の条件数を推定する。条件数の逆数がマシンの精度より小さい場合は、警告として `info` = `n` + 1 を返すが、後で説明するようにルーチンは続行され、X について解を算出し、誤差範囲を計算する。

3. A の因子分解された形式を使用して、連立方程式を X について解く。

4. 計算された解のマトリックスの精度の改善を繰り返し実行し、解の誤差範囲と後退誤差の推定値を計算する。

入力パラメータ

fact CHARACTER*1。'F' または 'N' でなければならない。

ルーチンの開始時にマトリックス A の因子分解された形式が与えられるかどうかを指定する。

fact = 'F' の場合は、開始時に、*afp* と *ipiv* に A の因子分解された形式が格納される。配列 *ap*、*afp*、*ipiv* は変更されない。

fact = 'N' の場合は、マトリックス A を *afp* にコピーし、因子分解を実行する。

uplo CHARACTER*1。'U' または 'L' でなければならない。

A の上三角部分と下三角部分のどちらを格納するか、また A をどのように因子分解するかを指定する。

uplo = 'U' の場合は、配列 *ap* には対称マトリックス A の上三角部分が格納され、A は UDU^T として因子分解される。

uplo = 'L' の場合は、配列 *ap* には対称マトリックス A の下三角部分が格納され、A は LDL^T として因子分解される。

n INTEGER。A の次数 ($n \geq 0$)。

nrhs INTEGER。右辺の数。B の列の数 ($nrhs \geq 0$)。

ap, afp, b, work

REAL (sspsvx の場合)
DOUBLE PRECISION (dspsvx の場合)
COMPLEX (cspsvx の場合)
DOUBLE COMPLEX (zspsvx の場合)

配列 : *ap*(*), *afp*(*), *b*(*ldb*, *), *work* (*)

配列 *ap* には、対称マトリックス A の上三角部分または下三角部分が**パックド格納形式**で格納される (「「マトリックスの格納形式」」を参照)。

配列 *afp* は、*fact* = 'F' の場合は入力引数になる。この配列には、?sptrf による因子分解 $A = U D U^T$ または $A = L D L^T$ で得られた、ブロック対角マトリックス D と、係数 U または L の計算に使用された乗数が、A と同じ格納形式で格納される。
配列 *b* には、マトリックス B が格納される。このマトリックスの列は、連立方程式の右辺である。

work (*) はワークスペース配列である。

配列 *ap* と *afp* の次元は、少なくとも max(1, *n*(*n*+1)/2) でなければならない。*b* の第 2 次元は、少なくとも max(1, *nrhs*) でなければならない。*work* の次元は、実数型の場合は少なくとも max(1, 3**n*)、複素数型の場合は少なくとも max(1, 2**n*) でなければならない。

ldb INTEGER。*b* の第 1 次元。*ldb* ≥ max(1, *n*)

ipiv INTEGER

配列 DIMENSION は少なくとも max(1, *n*)
配列 *ipiv* は、*fact* = 'F' の場合は入力引数になる。この配列には、?sptrf によって決まる、交換操作と D のブロック構造の詳細が格納される。
ipiv(*i*) = *k* > 0 の場合は、d_{ii} は 1 × 1 の対角ブロックである。A の *i* 番目の行と列は、*k* 番目の行と列に交換されている。

uplo = 'U' で、*ipiv*(*i*) =*ipiv*(*i*-1) = -*m* < 0 の場合は、D の行 *i* と *i*-1 および列 *i* と *i*-1 に 2 × 2 のブロックが作成される。A の (*i*-1) 番目の行と列は、*m* 番目の行と列に交換されている。

uplo = 'L' で、*ipiv*(*i*) =*ipiv*(*i*+1) = -*m* < 0 の場合は、D の行 *i* と *i*+1 および列 *i* と *i*+1 に 2 × 2 のブロックが作成される。A の (*i*+1) 番目の行と列は、*m* 番目の行と列に交換されている。

ldx INTEGER。出力配列 *x* のリーディング・ディメンジョン。*ldx* ≥ max(1, *n*)

iwork INTEGER
ワークスペース配列 DIMENSION は少なくとも max(1, *n*)。実数型でのみ使用される。

rwork REAL (cspsvx の場合)
DOUBLE PRECISION (zspsvx の場合)
ワークスペース配列 DIMENSION は少なくとも max(1, *n*)。複素数型でのみ使用される。

出力パラメータ

x REAL (sspsvx の場合)
DOUBLE PRECISION (dspsvx の場合)
COMPLEX (cspsvx の場合)
DOUBLE COMPLEX (zspsvx の場合)
配列 DIMENSION は (*ldx*, *)
info = 0 または *info* = *n*+1 の場合は、配列 *x* には、連立方程式の解のマトリックス *X* が格納される。*x* の第 2 次元は、少なくとも max(1, *nrhs*) でなければならない。

afp, *ipiv* これらの配列は、*fact* = 'N' の場合は出力引数になる。
「入力引数」の項の *afp*, *ipiv* の説明を参照。

rcond REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
マトリックス *A* の条件数の逆数の推定値。*rcond* がマシンの精度より小さい場合 (特に、*rcond* = 0 の場合) は、マトリックスは有効な精度で特異になる。この条件は、*info* > 0 のリターン・コードで示される。

ferr, *berr* REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
配列 DIMENSION は少なくとも max(1, *nrhs*)。各解ベクトルの成分ごとの前進誤差と相対後退誤差を格納する。

info INTEGER。*info* = 0 の場合は、ルーチンの実行は正常に終了した。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。
info = *i* で、*i* ≤ *n* の場合は、d_{ii} は完全に 0 である。因子分解は完了したが、ブロック対角マトリックス *D* が完全に特異であるため、解と誤差範囲は計算できなかった。*rcond* = 0 が返される。

info = *i* で、*i* = *n* +1 の場合は、*D* は特異でないが、*rcond* がマシンの精度より小さいため、マトリックスは有効な精度で特異になる。しかし、この場合は、解と誤差範囲が計算される。これは計算された解の精度が、*rcond* の値から想定される精度より高くなる場合があるためである。

?hpsvx

対角ピボット (要) 演算による因子分解を
使用して、パックド形式で格納される
エルミート・マトリックス A を
係数行列とする連立 1 次方程式の解を
計算し、解の誤差範囲を示す。

```
call chpsvx (fact, uplo, n, nrhs, ap, afp, ipiv, b, ldb, x, ldx,
             rcond, ferr, berr, work, rwork, info)
call zhpsvx (fact, uplo, n, nrhs, ap, afp, ipiv, b, ldb, x, ldx,
             rcond, ferr, berr, work, rwork, info)
```

説明

このルーチンは、対角ピボット (要) 演算による因子分解を使用して、複素連立 1 次方程式 $AX = B$ の解を計算する。A はパックド形式で格納される $n \times n$ のエルミート・マトリックス、マトリックス B の列は個々の右辺、X の列はそれに対応する解である。

このルーチンは、解の誤差範囲と条件数の推定値も示す。

ルーチン ?hpsvx は、以下の手順を実行する。

1. *fact* = 'N' の場合は、対角ピボット (要) 演算法を使用して、マトリックス A を因子分解する。因子分解の形式は、$A = U D U^H$ または $A = L D L^H$ である。U (または L) は置換マトリックスと単位上 (下) 三角マトリックスの積で、D は 1×1 と 2×2 の対角ブロックを持つエルミート・ブロック対角マトリックスである。

2. $d_{i,i} = 0$ の場合、つまり D が完全に特異である場合は、ルーチンは *info* = i を返す。それ以外の場合は、A の因子分解された形式を使用して、マトリックス A の条件数を推定する。条件数の逆数がマシンの精度より小さい場合は、警告として *info* = $n + 1$ を返すが、後で説明するようにルーチンは続行され、X について解を算出し、誤差範囲を計算する。

3. A の因子分解された形式を使用して、連立方程式を X について解く。

4. 計算された解のマトリックスの精度の改善を繰り返し実行し、解の誤差範囲と後退誤差の推定値を計算する。

入力パラメータ

fact CHARACTER*1。'F' または 'N' でなければならない。

ルーチンの開始時にマトリックス *A* の因子分解された形式が与えられるかどうかを指定する。

fact = 'F' の場合は、開始時に、*afp* と *ipiv* に *A* の因子分解された形式が格納される。配列 *ap*、*afp*、*ipiv* は変更されない。

fact = 'N' の場合は、マトリックス *A* を *afp* にコピーし、因子分解を実行する。

uplo CHARACTER*1。'U' または 'L' でなければならない。

A の上三角部分と下三角部分のどちらを格納するか、また *A* をどのように因子分解するかを指定する。

uplo = 'U' の場合は、配列 *ap* にはエルミート・マトリックス *A* の上三角部分が格納され、*A* は UDU^H として因子分解される。

uplo = 'L' の場合は、配列 *ap* にはエルミート・マトリックス *A* の下三角部分が格納され、*A* は LDL^H として因子分解される。

n INTEGER。*A* の次数 ($n \geq 0$)。

nrhs INTEGER。右辺の数。*B*B の列の数 ($nrhs \geq 0$)。

ap, afp, b, work COMPLEX (chpsvx の場合)

DOUBLE COMPLEX (zhpsvx の場合)

配列 : *ap*(*), *afp*(*), *b*(*ldb*, *), *work* (*)

配列 *ap* には、エルミート・マトリックス A の上三角部分または下三角部分が**パックド格納形式**で格納される (「「マトリックスの格納形式」」を参照)。

配列 *afp* は、*fact* = 'F' の場合は入力引数になる。この配列には、?hptrf による因子分解 $A = U\,D\,U^H$ または $A = L\,D\,L^H$ で得られた、ブロック対角マトリックス *D* と、係数 *U* または *L* の計算に使用された乗数が、*A* と同じ格納形式で格納される。

配列 *b* には、マトリックス *B* が格納される。このマトリックスの列は、連立方程式の右辺である。

work (*) はワークスペース配列である。

配列 *ap* と *afp* の次元は、少なくとも max(1, *n*(*n*+1)/2) でなければならない。*b* の第 2 次元は、少なくとも max(1, *nrhs*) でなければならない。*work* の次元は、少なくとも max(1, 2**n*) でなければならない。

ldb INTEGER。*b* の第 1 次元。*ldb* ≥ max(1, *n*)

ipiv INTEGER

配列 DIMENSION は少なくとも max(1, *n*)
配列 *ipiv* は、*fact* = 'F' の場合は入力引数になる。この配列には、?hptrf によって決まる、交換操作と *D* のブロック構造の詳細が格納される。
ipiv(*i*) = *k* > 0 の場合は、d_{ii} は 1 × 1 の対角ブロックである。*A* の *i* 番目の行と列は、*k* 番目の行と列に交換されている。

uplo = 'U' で、*ipiv*(*i*) =*ipiv*(*i*-1) = -*m* < 0 の場合は、*D* の行 *i* と *i*-1 および列 *i* と *i*-1 に 2 × 2 のブロックが作成される。*A* の (*i*-1) 番目の行と列は、*m* 番目の行と列に交換されている。

uplo = 'L' で、*ipiv*(*i*) =*ipiv*(*i*+1) = -*m* < 0 の場合は、*D* の行 *i* と *i*+1 および列 *i* と *i*+1 に 2 × 2 のブロックが作成される。*A* の (*i*+1) 番目の行と列は、*m* 番目の行と列に交換されている。

ldx INTEGER。出力配列 *x* のリーディング・ディメンジョン。*ldx* ≥ max(1, *n*)

rwork REAL (chpsvx の場合)
DOUBLE PRECISION (zhpsvx の場合)
ワークスペース配列 DIMENSION は少なくとも max(1, *n*)

出力パラメータ

x COMPLEX (chpsvx の場合)
DOUBLE COMPLEX (zhpsvx の場合)
配列 DIMENSION は (*ldx*, *)

info = 0 または *info* = *n* +1 の場合は、配列 *x* には、連立方程式の解のマトリックス *X* が格納される。*x* の第 2 次元は、少なくとも max(1, *nrhs*) でなければならない。

afp, ipiv
これらの配列は、*fact* = 'N' の場合は出力引数になる。
「入力引数」の項の *afp, ipiv* の説明を参照。

rcond
REAL (chpsvx の場合)
DOUBLE PRECISION (zhpsvx の場合)
マトリックス A の条件数の逆数の推定値。*rcond* がマシンの精度より小さい場合 (特に、*rcond* = 0 の場合) は、マトリックスは有効な精度で特異になる。この条件は、*info* > 0 のリターン・コードで示される。

ferr, berr
REAL (chpsvx の場合)
DOUBLE PRECISION (zhpsvx の場合)
配列 DIMENSION は少なくとも max(1, *nrhs*)。各解ベクトルの成分ごとの前進誤差と相対後退誤差を格納する。

info
INTEGER。*info* = 0 の場合は、ルーチンの実行は正常に終了した。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。
info = *i* で、*i* ≤ *n* の場合は、d_{ii} は完全に 0 である。因子分解は完了したが、ブロック対角マトリックス D が完全に特異であるため、解と誤差範囲は計算できなかった。*rcond* = 0 が返される。
info = *i* で、*i* = *n* +1 の場合は、D は特異でないが、*rcond* がマシンの精度より小さいため、マトリックスは有効な精度で特異になる。しかし、この場合は、解と誤差範囲が計算される。これは計算された解の精度が、*rcond* の値から想定される精度より高くなる場合があるためである。

?hpsv

パックド形式で格納されるエルミート・マトリックス A を係数行列とする、複数の右辺を持つ連立 1 次方程式の解を計算する。

```
call chpsv (uplo, n, nrhs, ap, ipiv, b, ldb, info)
call zhpsv (uplo, n, nrhs, ap, ipiv, b, ldb, info)
```

説明

このルーチンは、連立 1 次方程式 $AX = B$ を X について解く。A はパックド形式で格納される $n \times n$ のエルミート・マトリックス、マトリックス B の列は個々の右辺、X の列はそれに対応する解である。

対角ピボット (要) 演算法を使用して、A を $A = U D U^H$ または $A = L D L^H$ として因子分解する。U (または L) は置換マトリックスと単位上 (下) 三角マトリックスの積で、D は 1×1 と 2×2 の対角ブロックを持つエルミート・ブロック対角マトリックスである。

次に、A の因子分解された形式を使用して、連立方程式 $AX = B$ を解く。

入力パラメータ

uplo CHARACTER*1。'U' または 'L' でなければならない。
A の上三角部分と下三角部分のどちらを格納するか、また A をどのように因子分解するかを指定する。
uplo = 'U' の場合は、配列 *ap* にはマトリックス A の上三角部分が格納され、A は UDU^H として因子分解される。
uplo = 'L' の場合は、配列 *ap* にはマトリックス A の下三角部分が格納され、A は LDL^H として因子分解される。

n INTEGER。A の次数 ($n \geq 0$)。

nrhs INTEGER。右辺の数。B の列の数 ($nrhs \geq 0$)。

ap, b
COMPLEX (chpsv の場合)
DOUBLE COMPLEX (zhpsv の場合)
配列 : *ap*(*), *b*(*ldb*, *)
ap の次元は、少なくとも max(1, *n*(*n*+1)/2) でなければならない。
配列 *ap* には、*uplo* の指定に従って、係数 *U* または *L* が**パックド格納形式**で格納される (「「マトリックスの格納形式」」を参照)。
配列 *b* には、マトリックス *B* が格納される。このマトリックスの列は、連立方程式の右辺である。
b の第 2 次元は、少なくとも max(1, *nrhs*) でなければならない。

ldb
INTEGER。*b* の第 1 次元。*ldb* ≥ max(1, *n*)

出力パラメータ

ap
?hptrf による *A* の因子分解で得られた、ブロック対角マトリックス *D* と、係数 *U* (または *L*) の計算に使用された乗数が、*A* と同じ格納形式で、パックド形式の三角マトリックスとして格納される。

b
info = 0 の場合は、*b* は解のマトリックス *X* によって上書きされる。

ipiv
INTEGER
配列 DIMENSION は少なくとも max(1, *n*)
?hptrf によって決まる、交換操作と *D* のブロック構造の詳細を格納する。
ipiv(*i*) = *k* > 0 の場合は、d_{ii} は 1 × 1 のブロックである。*A* の *i* 番目の行と列は、*k* 番目の行と列に交換されている。
uplo = 'U' で、*ipiv*(*i*) =*ipiv*(*i*-1) = -*m* < 0 の場合は、*D* の行 *i* と *i*-1 および列 *i* と *i*-1 に 2 × 2 のブロックが作成される。*A* の (*i*-1) 番目の行と列は、*m* 番目の行と列に交換されている。
uplo = 'L' で、*ipiv*(*i*) =*ipiv*(*i*+1) = -*m* < 0 の場合は、*D* の行 *i* と *i*+1 および列 *i* と *i*+1 に 2 × 2 のブロックが作成される。*A* の (*i*+1) 番目の行と列は、*m* 番目の行と列に交換されている。

info INTEGER。*info* = 0 の場合は、ルーチンの実行は正常に終了した。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。
info = *i* の場合は、d_{ii} は 0 である。因子分解は完了したが、D が完全に特異であるため、解は計算されなかった。

LAPACK ルーチン：最小二乗問題および固有値問題

5

本章では、線形の最小二乗問題、固有値および特異値問題の解決や、それらに関連する一連の計算タスクを実行するための、LAPACK パッケージからインテル ® マス・カーネル・ライブラリに組み込まれているルーチンについて説明する。

本章の各節で、LAPACK の計算ルーチンとドライバ・ルーチンについて説明する。

個々の LAPACK ルーチンの詳細は、[*LUG*] を参照。

最小二乗問題　典型的な**最小二乗問題**とは、マトリックス A とベクトル b が与えられたときに、二乗和 $\Sigma_i ((Ax)_i - b_i)^2$ を最小にするベクトル x を見つける、または 2- ノルム $\|Ax - b\|_2$ を最小にするベクトル x を見つけることである。

通常、A は $m \times n$ のマトリックス (ただし、$m \geq n$) で、$\mathrm{rank}(A) = n$ である。また、この問題は、**優決定**の連立 1 次方程式 (未知数より方程式の個数の方が多い) に対して**最小二乗解**を見つけることとも言える。この問題を解くには、マトリックス A の QR 因子分解 (5-6 ページの **QR 因子分解を参照**) を使用できる。

$m < n$ で $\mathrm{rank}(A) = m$ の場合は、$Ax = b$ を満たす (つまり、ノルム $\|Ax - b\|_2$ が最小になる) 解 x が無限に存在する。これには、$\|x\|_2$ を最小にする一意な解を見つける方が有用な場合がよくある。この問題は、**劣決定**の連立 1 次方程式 (方程式より未知数の個数の方が多い) に対して**最小ノルム解**を見つけることとも言える。こうした問題を解くには、マトリックス A の LQ 因子分解 (5-7 ページの **LQ 因子分解**を参照) を使用できる。

一般には、rank(A) min(m, n) となり、**階数不足の最小二乗問題**になるときがあるので、$\|x\|_2$ も $\|Ax-b\|_2$ も最小にする**最小ノルム最小二乗解**を見つける必要がある。その場合 (または A の階数が不明な場合) は、QR 因子分解と、ピボット (要) 演算または**特異値分解** (5-74 ページを参照) を併用する。

固有値 (eigenvalue) 問題 (*eigen* は、" 固有 " の意味のドイツ語) は、「マトリックス A が与えられたときに、下記のいずれかの方程式を満たす**固有値** λ と、それに対応する**固有ベクトル** z を見つけること」である。

$Az = \lambda z$ (右固有ベクトル z)

または

$z^H A = \lambda z^H$ (左固有ベクトル z)

A が実対称マトリックスまたは複素エルミート・マトリックスの場合は、上記の 2 つの方程式が等価になり、上記の問題は**対称**固有値問題と呼ばれる。このタイプの問題を解決するためのルーチンについては、この章の**対称固有値問題** (5-101 ページを参照) の節で説明する。

非対称マトリックスや非エルミート・マトリックスを使って固有値問題を解決するためのルーチンについては、この章の**非対称固有値問題** (5-174 ページを参照) の節で説明する。

本ライブラリには、**汎用対称固有値問題** (次のいずれかの方程式を満たす固有値 λ と、それに対応する固有ベクトル z を見つけること) を処理するためのルーチンも入っている。

$Az = \lambda Bz,\ ABz = \lambda z,$ または $BAz = \lambda z$

A は、対称マトリックスまたはエルミート・マトリックスである。B は、正値対称マトリックスまたは正値エルミート・マトリックスである。これらの問題を標準的な対称固有値問題に帰着させるためのルーチンについては、この章の**一般対称固有値問題** (5-157 ページを参照) の節で説明する。

* * *

個々の問題を解決するには、通常、いくつかの計算ルーチンを呼び出す。本章のルーチンを、第 4 章で説明している他の LAPACK ルーチンや BLAS ルーチン (第 2 章) と組み合わせて使用しなければならない場合もある。

たとえば、与えられたマトリックス B のすべての列 b に対して $\|Ax-b\|_2$ が最小になる一連の最小二乗問題(A も B も実マトリックス)を解くには、?geqrf を呼び出して $A = QR$ の因子分解を実行した後、?ormqr を呼び出して $C = Q^H B$ を求める。最後に、BLAS ルーチンの ?trsm を呼び出して、連立方程式 $RX = C$ を X について解く。

あるいは、適切なドライバ・ルーチンを使用すれば、複数のタスクを一度に実行できる。例えば、最小二乗問題を解く場合は、ドライバ・ルーチン ?gels を使用する。

ルーチン名の規則

本章の各ルーチンについて、LAPACK 名を使用できる。

LAPACK 名は、以下に示すように、xyyzzz の構造になっている。

最初の文字 x は、データ・タイプを示す。

s	実数、単精度	c	複素数、単精度
d	実数、倍精度	z	複素数、倍精度

2 番目と 3 番目の文字 yy は、マトリックスのタイプと格納形式を示す。

bd	二重対角マトリックス
ge	一般マトリックス
gb	一般バンド・マトリックス
hs	Hessenberg 上マトリックス
or	(実数) 直交マトリックス
op	(実数) 直交マトリックス (パックド格納形式)
un	(複素数) ユニタリ・マトリックス
up	(複素数) ユニタリ・マトリックス (パックド格納形式)
pt	対称またはエルミートの正値三重対角マトリックス
sy	対称マトリックス
sp	対称マトリックス (パックド格納形式)
sb	(実数) 対称バンド・マトリックス
st	(実数) 対称三重対角マトリックス
he	エルミート・マトリックス
hp	エルミート・マトリックス (パックド格納形式)
hb	(複素数) エルミート・バンド・マトリックス
tr	三角または準三角マトリックス

最後の 3 文字 zzz は、実行される処理 (下記を参照) を示す。

qrf	*QR* 因子分解を実行する。
lqf	*LQ* 因子分解を実行する。

たとえば、ルーチン sgeqrf は、単精度実数型の一般マトリックスの *QR* 因子分解を実行する。これに対応する複素マトリックス用のルーチンは、cgeqrf である。

マトリックスの格納形式

LAPACK ルーチンでは、以下のマトリックス格納形式を使用している。

- **フル格納**では、マトリックス A は 2 次元配列 `a` に格納され、この場合、マトリックス成分 a_{ij} は配列成分 `a(i, j)` に格納される。
- **パックド格納**では、対称マトリックス、エルミート・マトリックス、または三角マトリックスをコンパクトに格納できる。マトリックスの上三角または下三角が 1 次元配列の各列にパックされる。
- **バンド格納**では、`kl` 個の劣対角成分と `ku` 個の優対角成分を持つ `m`×`n` のバンド・マトリックスが、(`kl`+`ku`+1) 行 `n` 列の 2 次元配列 `ab` にコンパクトに格納される。マトリックスの各列は配列の対応する列に格納され、マトリックスの各**対角成分**は配列の各行に格納される。

第 4 章と第 5 章では、パックド形式でマトリックスを格納する配列は名前の最後の文字を `p` で、バンド形式でマトリックスを格納する配列は名前の最後の文字を `b` で示す。

マトリックスの格納形式の詳細は、付録 A の「マトリックス引数」を参照。

数学表記

これまでの章で使用した数学表記以外に、本章では以下に示す表記も使用している。

λ_i	マトリックス A の**固有値** (固有値の定義は、5-2 ページの**固有値問題**を参照)。
σ_i	マトリックス A の**特異値**。$A^H A$ の固有値の平方根に等しい (詳細は、特異値分解を参照)。
$\|x\|_2$	ベクトル x の 2- ノルム x: $\|x\|_2 = (\Sigma_i \|x_i\|^2)^{1/2} = \|x\|_E$
$\|A\|_2$	マトリックス A の 2- ノルム (またはスペクトル・ノルム)。 $\|A\|_2 = \max_i \sigma_i$, $\|A\|_2^2 = \max_{\|x\|=1}(Ax \cdot Ax)$
$\|A\|_E$	マトリックス A の**ユークリッド・ノルム**。 $\|A\|_E^2 = \Sigma_i \Sigma_j \|a_{ij}\|^2$ (ベクトルの場合は、ユークリッド・ノルムと 2- ノルムが等しくなるので、$\|x\|_E = \|x\|_2$)
$\theta(x, y)$	**ベクトル** x と y **の間の鋭角**。 $\cos\theta(x, y) = \|x \cdot y\| / (\|x\|_2 \|y\|_2)$

計算ルーチン

以下の各節では、LAPACK 計算ルーチンについて説明する。LAPACK 計算ルーチンは、次の目的の計算タスクを個別に実行する。

直交因子分解
特異値分解
対称固有値問題
汎用対称固有値問題
非対称固有値問題
汎用非対称固有値問題
汎用特異値分解

各ドライバ・ルーチンも参照のこと。

直交因子分解

本節では、マトリックスの *QR (RQ)* 因子分解と *LQ (QL)* 因子分解を実行するための LAPACK ルーチンについて説明する。一般化された *QR* 因子分解と *RQ* 因子分解と同様に *RZ* 因子分解用のルーチンも含まれている。

QR 因子分解　A を、因子分解の対象となる $m \times n$ のマトリックスとする。$m \geq n$ の場合、*QR* 因子分解は次の式で与えられる。

$$A = Q\begin{pmatrix} R \\ 0 \end{pmatrix} = (Q_1,\ Q_2)\begin{pmatrix} R \\ 0 \end{pmatrix}$$

R は、実数型の対角成分を持つ $n \times n$ の上三角マトリックスである。
Q は、$m \times m$ の直交 (またはユニタリ) マトリックスである。

QR 因子分解を使えば、「A が $m \times n$ の最大階数のマトリックス $(m \geq n)$ である場合に、$\|Ax - b\|_2$ を最小の値にする」という最小二乗問題が解ける。該当するマトリックスの因子分解を行ってから、$Rx = (Q_1)^T b$ を解いて、解 x を求める。

$m < n$ の場合、*QR* 因子分解は次の式で与えられる。

$$A = QR = Q(R_1 R_2)$$

R は台形マトリックスで、R_1 は上三角マトリックス、R_2 は矩形マトリックスである。

LAPACK ルーチンでは、マトリックス Q を明示的な形式では表現しない。Q は、$\min(m, n)$ の**基本リフレクタ**の積として表現される。一連のルーチンは、この形式で表現される Q を操作するためのものである。

LQ 因子分解　A を $m \times n$ のマトリックスとすると、$m \le n$ の場合は、次の式で与えられる。

$$A = (L, 0)Q = (L, 0)\begin{pmatrix} Q_1 \\ Q_2 \end{pmatrix} = LQ_1$$

L は、実数の対角成分を持つ $m \times m$ の下三角マトリックスで、Q は、$n \times n$ の直交 (またはユニタリ) マトリックスである。

$m > n$ の場合、LQ 因子分解は次の式で与えられる。

$$A = \begin{pmatrix} L_1 \\ L_2 \end{pmatrix} Q$$

L_1 は $n \times n$ の下三角マトリックス、L_2 は矩形マトリックス、Q は $n \times n$ の直交 (またはユニタリ) マトリックスである。

LQ 因子分解を使えば、劣決定の連立 1 次方程式 $Ax = b$ (A は、階数 $m(m < n)$ の $m \times n$ のマトリックス) の最小ノルム解を求められる。該当するマトリックスの因子分解を行ってから、$Ly = b$ を y について解いた後、$x = (Q_1)^H y$ を計算し、解ベクトル x を求める。

表 5-1 に、マトリックスの直交因子分解を実行するための LAPACK ルーチンを示す。

表 5-1　直交因子分解用の計算ルーチン

マトリックスのタイプ、因子分解	因子分解 (ピボット (要) 演算なし)	因子分解 (ピボット (要) 演算あり)	マトリックス Q の生成	マトリックス Q の適用
一般マトリックス、QR 因子分解	?geqrf	?geqpf ?geqp3	?orgqr ?ungqr	?ormqr ?unmqr
一般マトリックス、RQ 因子分解	?gerqf		?orgrq ?ungrq	?ormrq ?unmrq
一般マトリックス、LQ 因子分解	?gelqf		?orglq ?unglq	?ormlq ?unmlq
一般マトリックス、QL 因子分解	?geqlf		?orgql ?ungql	?ormql ?unmql
台形マトリックス、RZ 因子分解	?tzrzf			?ormrz ?unmrz
トリックスのペア、汎用 QR 因子分解	?ggqrf			
マトリックスのペア、汎用 RQ 因子分解	?ggrqf			

?geqrf

m × *n* の一般マトリックスの *QR* 因子分解を行う。

```
call sgeqrf ( m, n, a, lda, tau, work, lwork, info )
call dgeqrf ( m, n, a, lda, tau, work, lwork, info )
call cgeqrf ( m, n, a, lda, tau, work, lwork, info )
call zgeqrf ( m, n, a, lda, tau, work, lwork, info )
```

説明

このルーチンは、*m* × *n* の一般マトリックス *A* の *QR* 因子分解を実行する (5-6 ページの**直交因子分解**を参照)。ピボット (要) 演算は、行われない。

このルーチンでは、マトリックス *Q* を明示的な形式では表現しない。*Q* は、min(*m*, *n*) の一連の**基本リフレクタ**の積として表現される。一連のルーチンは、この形式で表現される *Q* を操作するためのものである。

入力パラメータ

m	INTEGER。マトリックス *A* の行数 (*m* ≥ 0)。
n	INTEGER。マトリックス *A* の列数 (*n* ≥ 0)。
a, *work*	REAL (sgeqrf の場合) DOUBLE PRECISION (dgeqrf の場合) COMPLEX (cgeqrf の場合) DOUBLE COMPLEX (zgeqrf の場合) 配列 : *a*(*lda*, *) には、マトリックス *A* を格納する。 *a* の第 2 次元は、max(1, *n*) 以上でなければならない。 *work*(*lwork*) は、ワークスペース配列である。
lda	INTEGER。*a* の第 1 次元。max(1, *m*) 以上であること。
lwork	INTEGER。配列 *work* のサイズ (*lwork* ≥ *n*) *lwork* の推奨値は、「アプリケーション・ノート」を参照。

出力パラメータ

a

因子分解のデータによって、次のように上書きされる。

m ≥ *n* の場合は、対角線より下の成分が、ユニタリ・マトリックス *Q* の各成分によって上書きされる。また、上三角マトリックスは、上三角マトリックス *R* の対応する成分によって上書きされる。

m < *n* の場合は、厳密な意味での下三角部分がユニタリ・マトリックス *Q* の各成分によって上書きされる。また、残りの成分は、*m* × *n* の上台形マトリックス *R* の対応する成分によって上書きされる。

tau

REAL (sgeqrf の場合)
DOUBLE PRECISION (dgeqrf の場合)
COMPLEX (cgeqrf の場合)
DOUBLE COMPLEX (zgeqrf の場合)
配列 DIMENSION は、max(1, min(*m*, *n*)) 以上であること。
マトリックス *Q* に関するその他の情報も格納される。

work(1)

info = 0 の場合は、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の *lwork* の値が *work*(1) に格納される。これ以降の実行には、この *lwork* の値を使用する。

info

INTEGER
info = 0 の場合は、正常に終了する。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

パフォーマンスを改善するには、*lwork* =*n***blocksize* に設定する。*blocksize* は、**ブロック化アルゴリズム**の最適パフォーマンスに必要な値で、マシンによって異なる (通常は 16 ～ 64)。

用意すべきワークスペースの量が不明な場合は、最初は *lwork* に大きな値を設定する。終了時に *work*(1) の値を調べ、これ以降はその値を使用する。

計算から得られる因子分解は、マトリックス $A + E$ ($\|E\|_2 = O(\varepsilon)\ \|A\|_2$) を因子分解したそのものである。

実数の場合の浮動小数演算のおおよその総数は、次のとおりである。

$(4/3)n^3$	($m = n$ の場合)
$(2/3)n^2(3m-n)$	($m > n$ の場合)
$(2/3)m^2(3n-m)$	($m < n$ の場合)

複素数の場合は、この 4 倍になる。

与えられたマトリックス B のすべての列 b に対して $\|Ax-b\|_2$ が最小にする一連の最小二乗問題を解くには、次の手順でルーチンを呼び出す。

?geqrf (このルーチン)	因子分解 $A = QR$ を行う。
?ormqr	$C = Q^TB$ を求める (実マトリックスの場合)。
?unmqr	$C = Q^HB$ を求める (複素マトリックスの場合)。
?trsm (BLAS ルーチン)	$RX = C$ を解く。

(得られた X の列は、最小二乗の解ベクトル x である。)

Q の成分を明示的に求めるには、以下のルーチンを呼び出す。

?orgqr	(実マトリックスの場合)
?ungqr	(複素マトリックスの場合)

?geqpf

m × *n* の一般マトリックスの *QR* 因子分解をピボット (要) 演算付きで行う。

```
call sgeqpf ( m, n, a, lda, jpvt, tau, work, info )
call dgeqpf ( m, n, a, lda, jpvt, tau, work, info )
call cgeqpf ( m, n, a, lda, jpvt, tau, work, rwork, info )
call zgeqpf ( m, n, a, lda, jpvt, tau, work, rwork, info )
```

説明

このルーチンは ?geqp3 に置き換えられている。

このルーチン ?geqpf は、*m* × *n* の一般マトリックス *A* の *QR* 因子分解である *AP* = *QR* を、列に対してピボット (要) 演算を行って実行する (5-6 ページの**直交因子分解**を参照)。*P* は、*n* × *n* の置換マトリックスである。

このルーチンでは、マトリックス *Q* を明示的な形式では表現しない。*Q* は、min(*m*, *n*) の一連の**基本リフレクタ**の積として表現される。一連のルーチンは、この形式で表現される *Q* を操作するためのものである。

入力パラメータ

m	INTEGER。マトリックス *A* の行数 (*m* ≥ 0)
n	INTEGER。マトリックス *A* の列数 (*n* ≥ 0)
a, *work*	REAL (sgeqpf の場合) DOUBLE PRECISION (dgeqpf の場合) COMPLEX (cgeqpf の場合) DOUBLE COMPLEX (zgeqpf の場合) 配列 : *a* (*lda*, *) には、マトリックス *A* を格納する。 *a* の第 2 次元は、max(1, *n*) 以上でなければならない。 *work* (*lwork*) は、ワークスペース配列である。
lda	INTEGER。*a* の第 1 次元。max(1, *m*) 以上であること。
lwork	INTEGER。配列 *work* のサイズ。max(1, 3*n) 以上であること。

jpvt
INTEGER。配列 DIMENSION は、max(1, *n*) 以上であること。
呼び出し時に *jpvt(i)* > 0 であると、計算が開始される前に、*A* の *i* 番目の列が *AP* の先頭に移され、計算の実行中はそこに保持される。
jpvt(i) = 0 の場合は、*A* の *i* 番目の列はフリー列なので、計算の実行時に他のフリー列と交換される場合がある。

rwork
REAL (cgeqpf の場合)
DOUBLE PRECISION (zgeqpf の場合)
ワークスペース配列 DIMENSION は、max(1, 2*n) 以上であること。

出力パラメータ

a
因子分解のデータによって、次のように上書きされる。
m ≥ *n* の場合は、対角線より下の成分が、ユニタリ (直交) マトリックス *Q* の各成分によって上書きされる。また、上三角マトリックスは、上三角マトリックス *R* の対応する成分によって上書きされる。
m < *n* の場合は、厳密な意味での下三角部分がマトリックス *Q* の各成分によって上書きされる。また、残りの成分は、*m* × *n* の上台形マトリックス *R* の対応する成分によって上書きされる。

tau
REAL (sgeqpf の場合)
DOUBLE PRECISION (dgeqpf の場合)
COMPLEX (cgeqpf の場合)
DOUBLE COMPLEX (zgeqpf の場合)
配列 DIMENSION は、max(1, min(*m*, *n*)) 以上であること。
マトリックス *Q* に関するその他の情報も格納される。

jpvt
因子分解 *AP* = *QR* における置換マトリックス *P* の各成分によって上書きされる。詳しく説明すると、*AP* の各列は、次の順序で *A* の各列になる。
jpvt(1), *jpvt*(2), ... *jpvt*(*n*)

info
INTEGER
info = 0 の場合は、正常に終了する。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

計算から得られる因子分解は、マトリックス $A + E$ ($\|E\|_2 = O(\varepsilon)\ \|A\|_2$) を因子分解したそのものである。

実数の場合の浮動小数演算のおおよその総数は、次のとおりである。

$(4/3)n^3$	if $m = n$
$(2/3)n^2(3m-n)$	($m > n$ の場合)。
$(2/3)m^2(3n-m)$	($m < n$ の場合)。

複素数の場合は、この 4 倍になる。

与えられたマトリックス B のすべての列 b に対して $\|Ax - b\|_2$ が最小にする一連の最小二乗問題を解くには、次の手順でルーチンを呼び出す。

`?geqpf` (このルーチン)	因子分解 $A = QR$ を実行する。
?ormqr	$C = Q^TB$ を計算する (実マトリックスの場合)。
?unmqr	$C = Q^HB$ を計算する (複素マトリックスの場合)。
?trsm (BLAS ルーチン)	$RX = C$ を解く。

(計算された X の列は、最小二乗解ベクトル x が置換されたものである。置換順序は、配列 `jpvt` に出力される。)

Q の成分を明示的に計算するには、次のルーチンを呼び出す。

?orgqr	(実マトリックスの場合)
?ungqr	(複素マトリックスの場合)

?geqp3

レベル 3 の BLAS を使用し、列ピボット演算を行って、*m* × *n* の一般マトリックスの *QR* 因子分解を実行する。

```
call sgeqp3 ( m, n, a, lda, jpvt, tau, work, lwork, info )
call dgeqp3 ( m, n, a, lda, jpvt, tau, work, lwork, info )
call cgeqp3 ( m, n, a, lda, jpvt, tau, work, lwork, rwork, info )
call zgeqp3 ( m, n, a, lda, jpvt, tau, work, lwork, rwork, info )
```

説明

このルーチンは、列ピボット演算を用いた *m* × *n* の一般マトリックス *A* の *QR* 因子分解 (*AP* = *QR*、5-6 ページの**直交因子分解**を参照) を、レベル 3 の BLAS を使用して実行する。*P* は *n* × *n* の置換マトリックスを表す。このルーチンを使用すると、?geqpf より高いパフォーマンスが得られる。

このルーチンは、マトリックス *Q* を明示的に生成するものではない。マトリックス *Q* は、min(*m*, *n*) 個の**基本リフレクタ**の積で表される。このように表されるマトリックス *Q* を操作するために、一連のルーチンが提供されている。

入力パラメータ

m	INTEGER。マトリックス *A* の行数 (*m* ≥ 0)。
n	INTEGER。マトリックス *A* の列数 (*n* ≥ 0)。
a, *work*	REAL (sgeqp3 の場合) DOUBLE PRECISION (dgeqp3 の場合) COMPLEX (cgeqp3 の場合) DOUBLE COMPLEX (zgeqp3 の場合) 配列 : *a* (*lda*, *) には、マトリックス *A* を格納する。 *a* の第 2 次元のサイズは、max(1, *n*) 以上でなければならない。 *work* (*lwork*) は、ワークスペース配列。
lda	INTEGER。*a* の第 1 次元のサイズ。max(1, *m*) 以上。

lwork INTEGER。配列 *work* のサイズ。実数の場合は max(1, 3**n*+1) 以上、複素数の場合は max(1, *n*+1) 以上でなければならない。

jpvt INTEGER。配列 DIMENSION は max(1, *n*) 以上。

呼び出し時に、*jpvt(i)* ≠ 0 ならば、計算を開始する前に *A* の *i* 番目の列が *AP* の先頭に移され、計算中はその位置に保持される。

jpvt(i) = 0 ならば、*A* の *i* 番目の列がフリー列になり、計算中に他のフリー列と交換される場合がある。

rwork REAL (cgeqp3 の場合)

DOUBLE PRECISION (zgeqp3 の場合)

ワークスペース配列 DIMENSION は max(1, 2**n*) 以上。複素数の場合にのみ使用する。

出力パラメータ

a 因子分解データによって、次のように上書きされる。

m ≥ *n* の場合、対角線より下の各成文は、ユニタリ (直交) マトリックス *Q* の情報で上書きされ、上三角部分は、上三角マトリックス *R* の対応する成分で上書きされる。

m < *n* の場合、厳密に下三角部分がマトリックス *Q* の情報で上書きされ、残りの各成分は、*m* × *n* の上台形マトリックス *R* の対応する成分で上書きされる。

tau REAL (sgeqp3 の場合)

DOUBLE PRECISION (dgeqp3 の場合)

COMPLEX (cgeqp3 の場合)

DOUBLE COMPLEX (zgeqp3 の場合)

配列 DIMENSION は max(1, min(*m*, *n*)) 以上。

マトリックス *Q* に対する基本リフレクタのスカラー係数が入る。

jpvt	因子分解$AP = QR$における置換マトリックスPの情報で上書きされる。すなわち、AP の列は、A の列を次の順に並べたものになる。 *jpvt*(1), *jpvt*(2), . . . , *jpvt*(*n*)
info	INTEGER *info* = 0 の場合は、正常終了している。 *info* = -*i* の場合は、*i* 番目のパラメータ値が不正である。

アプリケーション・ノート

与えられたマトリックス B のすべての列 b に対して $\|Ax - b\|_2$ を最小にする一連の最小二乗問題を解くには、次の順でルーチンを呼び出す。

?geqp3 (このルーチン)	因子分解 $AP = QR$ を実行する。
?ormqr	$C = Q^T B$ を計算する (実マトリックスの場合)。
?unmqr	$C = Q^H B$ を計算する (複素マトリックスの場合)。
?trsm (BLAS ルーチン)	$RX = C$ を解く。

(計算された X の列は、最小二乗解ベクトル x が置換されたものである。置換順序は、配列 *jpvt* に出力される。)

Q の成分を明示的に計算するには、次のルーチンを呼び出す。

?orgqr	(実マトリックスの場合)
?ungqr	(複素マトリックスの場合)

?orgqr

?geqrf で求めた *QR* 因子分解の実数型
直交マトリックス *Q* を生成する。

```
call sorgqr ( m, n, k, a, lda, tau, work, lwork, info )
call dorgqr ( m, n, k, a, lda, tau, work, lwork, info )
```

説明

このルーチンは、ルーチン sgeqrf/dgeqrf (5-8 ページを参照) または sgeqpf/dgeqpf (5-11ページを参照)によって求めた*QR*因子分解の直交マトリックス *Q* ($m \times m$) の全部または一部を生成する。このルーチンは、sgeqrf/dgeqrf または sgeqpf/dgeqpf を呼び出した後で使用する。

Q は、通常、$m \times p$ のマトリックス *A* ($m \geq p$) の *QR* 因子分解によって決定される。マトリックス *Q* の全体を計算するには、次のように呼び出す。

```
call ?orgqr ( m, m, p, a, lda, tau, work, lwork, info )
```

Q の先頭から p 個の列 (*A* の列で張られた空間内で直交基を形成) を計算するには、次のように呼び出す。

```
call ?orgqr ( m, p, p, a, lda, tau, work, lwork, info )
```

A の先頭から k 個の列に関して *QR* 因子分解のマトリックス Q^k を求めるには、次のように呼び出す。

```
call ?orgqr ( m, m, k, a, lda, tau, work, lwork, info )
```

Q^k の先頭から k 個の列 (*A* の先頭から k 個の列で張られた空間内で直交基を形成) を求めるには、次のように呼び出す。

```
call ?orgqr ( m, k, k, a, lda, tau, work, lwork, info )
```

入力パラメータ

m	INTEGER。直交マトリックス *Q* ($m \geq 0$) の次数
n	INTEGER。計算対象の *Q* の列数 ($0 \leq n \leq m$)
k	INTEGER。基本リフレクタ (その積でマトリックス *Q* を定義) の個数 ($0 \leq k \leq n$)

a, tau, work	REAL (sorgqr の場合) DOUBLE PRECISION (dorgqr の場合) 配列 : *a(lda, *)* と *tau(*)* は、sgeqrf / dgeqrf または sgeqpf / dgeqpf から返された配列である。 *a* の第 2 次元は、max(1, *n*) 以上でなければならない。 *tau* の次元は、max(1, *k*) 以上でなければならない。 *work* (*lwork*) は、ワークスペース配列である。
lda	INTEGER。*a* の第 1 次元。max(1, *m*) 以上であること。
lwork	INTEGER。配列 *work* のサイズ (*lwork* ≥ *n*) *lwork* の推奨値は、「アプリケーション・ノート」を参照。

出力パラメータ

a	*m* × *m* の直交マトリックス *Q* の先頭から *n* 個の列によって上書きされる。
work(1)	*info* = 0 の場合は、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の *lwork* の値が *work(1)* に格納される。これ以降の実行には、この *lwork* の値を使用する。
info	INTEGER *info* = 0 の場合は、正常に終了する。 *info* = *-i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

パフォーマンスを改善するには、*lwork* =*n***blocksize* に設定する。*blocksize* は、**ブロック化アルゴリズム**の最適パフォーマンスに必要な値で、マシンによって異なる (通常は 16 ～ 64)。用意すべきワークスペースの量が不明な場合は、最初は *lwork* に大きな値を設定する。終了時に *work(1)* の値を調べ、これ以降はその値を使用する。

計算で求めた *Q* は、正確な直交マトリックスとマトリックス E ($\|E\|_2 = O(\varepsilon)\,\|A\|_2$、ε はマシン精度) だけ異なる。

浮動小数演算のおおよその総数は、
$4*m*n*k - 2*(m+n)*k^2 + (4/3)*k^3$ である。
$n = k$ の場合、この値は約 $(2/3)*n^2*(3m - n)$ になる。

このルーチンの複素数版は、?ungqr である。

?ormqr

?geqrf または ?geqpf で求めた *QR* 因子分解の直交マトリックス *Q* と実マトリックスとを掛け合わせる。

```
call sormqr ( side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork,
              info )
call dormqr ( side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork,
              info )
```

説明

このルーチンは、実マトリックス *C* と *Q* または Q^T とを掛け合わせる。*Q* は、ルーチン sgeqrf/dgeqrf (5-8 ページを参照) または sgeqpf/dgeqpf (5-11 ページを参照) で求めた *QR* 因子分解の直交マトリックス *Q* である。

このルーチンを使用すれば、パラメータ *side* と *trans* に応じて、*QC*、Q^TC、*CQ*、または CQ^T のいずれかのマトリックス積を求められる (*C* の値は上書きされる)。

入力パラメータ

side	CHARACTER*1。'L' または 'R' でなければならない。 *side* ='L' の場合、*Q* または Q^T は、*C* に左側から適用される。 *side* ='R' の場合、*Q* または Q^T は、*C* に右側から適用される。
trans	CHARACTER*1。'N' または 'T' でなければならない。 *trans* ='N' の場合、*C* に *Q* が掛けられる。 *trans* ='T' の場合、*C* に Q^T が掛けられる。
m	INTEGER。マトリックス *C* の行数 ($m \geq 0$)。
n	INTEGER。マトリックス *C* の列数 ($n \geq 0$)。
k	INTEGER。基本リフレクタ (その積でマトリックス *Q* を定義) の個数。次の制約がある。 $0 \leq k \leq m$ (*side* ='L' の場合) $0 \leq k \leq n$ (*side* ='R' の場合)
a, work, tau, c	REAL (sgeqrf の場合) DOUBLE PRECISION (dgeqrf の場合) 配列 : *a(lda, *)* と *tau(*)* は、sgeqrf / dgeqrf または sgeqpf / dgeqpf から返された配列である。

	a の第 2 次元は、max(1, k) 以上でなければならない。 tau の次元は、max(1, k) 以上でなければならない。 c(ldc, *) には、マトリックス C を格納する。 c の第 2 次元は、max(1, n) 以上でなければならない。 work(lwork) は、ワークスペース配列である。
lda	INTEGER。a の第 1 次元。次の制約がある。 side='L' の場合は、lda ≥ max(1, m) side='R' の場合は、lda ≥ max(1, n)
ldc	INTEGER。c の第 1 次元。次の制約がある。 ldc ≥ max(1, m)。
lwork	INTEGER。配列 work のサイズ。次の制約がある。 side='L' の場合は、lwork ≥ max(1, n) side='R' の場合は、lwork ≥ max(1, m) lwork の推奨値は、「アプリケーション・ノート」を参照。

出力パラメータ

c	(side と trans の指定に従って) QC、Q^TC、CQ、または CQ^T のいずれかの積によって上書きされる。
work(1)	info = 0 の場合は、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の lwork の値が work(1) に格納される。これ以降の実行には、この lwork の値を使用する。
info	INTEGER info = 0 の場合は、正常に終了する。 info = -i の場合は、i 番目のパラメータの値が不正である。

アプリケーション・ノート

パフォーマンスを改善するには、lwork = n*blocksize (side='L' の場合) か lwork = m*blocksize (side='R' の場合) に設定する。*blocksize* は、**ブロック化アルゴリズム**の最適パフォーマンスに必要な値で、マシンによって異なる (通常は 16 ～ 64)。用意すべきワークスペースの量が不明な場合は、最初は lwork に大きな値を設定する。終了時に work(1) の値を調べ、これ以降はその値を使用する。

このルーチンの複素数版は、?unmqr である。

?ungqr

?geqrf で求めた *QR* 因子分解の複素
ユニタリ・マトリックス *Q* を生成する。

```
call cungqr ( m, n, k, a, lda, tau, work, lwork, info )
call zungqr ( m, n, k, a, lda, tau, work, lwork, info )
```

説明

このルーチンは、ルーチン cgeqrf/zgeqrf (5-8 ページを参照) または cgeqpf/zgeqpf (5-11ページを参照)によって求めた*QR*因子分解のユニタリ・マトリックス *Q* ($m \times m$) の全部または一部を生成する。このルーチンは、ルーチン cgeqrf/zgeqrf または cgeqpf/zgeqpf を呼び出した後で使用する。

Q は、通常、$m \times p$ のマトリックス *A* ($m \geq p$) の *QR* 因子分解によって決定される。マトリックス *Q* の全体を計算するには、次のように呼び出す。

```
call ?ungqr ( m, m, p, a, lda, tau, work, lwork, info )
```

Q の先頭から p 個の列 (*A* の列で張られた空間内で直交基を形成) を計算するには、次のように呼び出す。

```
call ?ungqr ( m, p, p, a, lda, tau, work, lwork, info )
```

A の先頭から k 個の列に関して *QR* 因子分解のマトリックス Q^k を求めるには、次のように呼び出す。

```
call ?ungqr ( m, m, k, a, lda, tau, work, lwork, info )
```

Q^k の先頭から k 個の列 (*A* の先頭から k 個の列で張られた空間内で直交基を形成) を求めるには、次のように呼び出す。

```
call ?ungqr ( m, k, k, a, lda, tau, work, lwork, info )
```

入力パラメータ

m	INTEGER。ユニタリ・マトリックス *Q* ($m \geq 0$) の次数
n	INTEGER。計算対象の *Q* の列数 ($0 \leq n \leq m$)
k	INTEGER。基本リフレクタ (その積でマトリックス *Q* を定義) の個数 ($0 \leq k \leq n$)

a, tau, work	COMPLEX (cungqr の場合) DOUBLE COMPLEX (zungqr の場合) 配列 : *a(lda, *)* と *tau(*)* は、cgeqrf/zgeqrf または cgeqpf/zgeqpf から返された配列である。 *a* の第 2 次元は、max(1, *n*) 以上でなければならない。 *tau* の次元は、max(1, *k*) 以上でなければならない。 *work* (*lwork*) は、ワークスペース配列である。
lda	INTEGER。*a* の第 1 次元。max(1, *m*) 以上であること。
lwork	INTEGER。配列 *work* のサイズ (*lwork* ≥ *n*) *lwork* の推奨値は、「アプリケーション・ノート」を参照。

出力パラメータ

a	*m* × *m* のユニタリ・マトリックス *Q* の先頭から *n* 個の列によって上書きされる。
work(1)	*info* = 0 の場合は、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の *lwork* の値が *work(1)* に格納される。これ以降の実行には、この *lwork* の値を使用する。
info	INTEGER *info* = 0 の場合は、正常に終了する。 *info* = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

パフォーマンスを改善するには、*lwork* =*n***blocksize* に設定する。*blocksize* は、**ブロック化アルゴリズム**の最適パフォーマンスに必要な値で、マシンによって異なる (通常は 16 ～ 64)。用意すべきワークスペースの量が不明な場合は、最初は *lwork* に大きな値を設定する。終了時に *work(1)* の値を調べ、これ以降はその値を使用する。

計算で求めた *Q* は、正確なユニタリ・マトリックスとマトリックス *E* ($\|E\|_2 = O(\varepsilon)\ \|A\|_2$、ε はマシン精度) だけ異なる。

浮動小数演算のおおよその総数は、
$16*m*n*k - 8*(m+n)*k^2 + (16/3)*k^3$ である。
$n = k$ の場合、この値は約 $(8/3)*n^2*(3m - n)$ になる。

このルーチンの実数版は、?orgqr である。

?unmqr

?geqrf で求めた *QR* 因子分解の
ユニタリ・マトリックス *Q* と複素
マトリックスとを掛け合わせる。

```
call cunmqr ( side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork,
              info )
call zunmqr ( side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork,
              info )
```

説明 1

このルーチンは、矩形の複素マトリックス *C* と *Q* または Q^H とを掛け合わせる。*Q* は、ルーチン cgeqrf/zgeqrf (5-8 ページを参照) または cgeqpf/zgeqpf (5-11 ページを参照) で求めた *QR* 因子分解のユニタリ・マトリックス *Q* である。

このルーチンを使用すれば、パラメータ *side* と *trans* に応じて、*QC*、Q^HC、*CQ*、または CQ^H のいずれかのマトリックス積を求められる (*C* の値は上書きされる)。

入力パラメータ

side CHARACTER*1。'L' または 'R' でなければならない。
side ='L' の場合、*Q* または Q^H は、*C* に左側から適用される。
side ='R' の場合、*Q* または Q^H は、*C* に右側から適用される。

trans CHARACTER*1。'N' または 'C' でなければならない。
trans ='N' の場合、*C* に *Q* が掛けられる。
trans ='C' の場合、*C* に Q^H が掛けられる。

m INTEGER。マトリックス *C* の行数 ($m \geq 0$)。

n INTEGER。マトリックス *C* の列数 ($n \geq 0$)。

k INTEGER。基本リフレクタ (その積でマトリックス *Q* を定義) の個数。次の制約がある。
$0 \leq k \leq m$ (*side* ='L' の場合)
$0 \leq k \leq n$ (*side* ='R' の場合)

a, work, tau, c COMPLEX (cgeqrf の場合)
DOUBLE COMPLEX (zgeqrf の場合)
配列 :
*a(lda, *)* と *tau(*)* は、cgeqrf / zgeqrf または cgeqpf / zgeqpf から返された配列である。

	a の第 2 次元は、max(1, k) 以上でなければならない。 tau の次元は、max(1, k) 以上でなければならない。 c(ldc, *) には、マトリックス C を格納する。 c の第 2 次元は、max(1, n) 以上でなければならない。 work(lwork) は、ワークスペース配列である。
lda	INTEGER。a の第 1 次元。次の制約がある。 side='L' の場合は、lda ≥ max(1, m)、 side='R' の場合は、lda ≥ max(1, n)
ldc	INTEGER。c の第 1 次元。次の制約がある。 ldc ≥ max(1, m)。
lwork	INTEGER。配列 work のサイズ。次の制約がある。 lwork ≥ max(1, n) (side='L' の場合) lwork ≥ max(1, m) (side='R' の場合) lwork の推奨値は、「アプリケーション・ノート」を参照。

出力パラメータ

c	(side と trans の指定に従って) QC、Q^HC、CQ、または CQ^H のいずれかの積によって上書きされる。
work(1)	info = 0 の場合は、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の lwork の値が work(1) に格納される。これ以降の実行には、この lwork の値を使用する。
info	INTEGER info = 0 の場合は、正常に終了する。 info = -i の場合は、i 番目のパラメータの値が不正である。

アプリケーション・ノート

パフォーマンスを改善するには、lwork = n*blocksize (side='L' の場合) か lwork = m*blocksize (side='R' の場合) に設定する。*blocksize* は、**ブロック化アルゴリズム**の最適パフォーマンスに必要な値で、マシンによって異なる (通常は 16 ～ 64)。用意すべきワークスペースの量が不明な場合は、最初は lwork に大きな値を設定する。終了時に work(1) の値を調べ、これ以降はその値を使用する。

このルーチンの実数版は、?ormqr である。

?gelqf

m × *n* の一般マトリックスの *LQ* 因子分解を行う。

```
call sgelqf ( m, n, a, lda, tau, work, lwork, info )
call dgelqf ( m, n, a, lda, tau, work, lwork, info )
call cgelqf ( m, n, a, lda, tau, work, lwork, info )
call zgelqf ( m, n, a, lda, tau, work, lwork, info )
```

説明

このルーチンは、*m* × *n* の一般マトリックス *A* の *LQ* 因子分解を実行する (5-6 ページの**直交因子分解**を参照)。ピボット (要) 演算は、行われない。

このルーチンでは、マトリックス *Q* を明示的な形式では表現しない。*Q* は、min(*m*, *n*) の一連の**基本リフレクタ**の積として表現される。一連のルーチンは、この形式で表現される *Q* を操作するためのものである。

入力パラメータ

m	INTEGER。マトリックス *A* の行数 (*m* ≥ 0)。
n	INTEGER。マトリックス *A* の列数 (*n* ≥ 0)。
a, *work*	REAL (sgelqf の場合) DOUBLE PRECISION (dgelqf の場合) COMPLEX (cgelqf の場合) DOUBLE COMPLEX (zgelqf の場合) 配列 : *a*(*lda*, *) には、マトリックス *A* を格納する。 *a* の第 2 次元は、max(1, *n*) 以上でなければならない。 *work*(*lwork*) は、ワークスペース配列である。
lda	INTEGER。*a* の第 1 次元。max(1, *m*) 以上であること。
lwork	INTEGER。配列 *work* のサイズ (max(1, *m*) 以上であること)。 *lwork* の推奨値は、「アプリケーション・ノート」を参照。

出力パラメータ

a
因子分解のデータによって、次のように上書きされる。
m ≤ *n* の場合は、対角線より上の成分が、ユニタリ (直交) マトリックス *Q* の各成分によって上書きされる。また、下三角マトリックスは、下三角マトリックス *L* の対応する成分によって上書きされる。
m > *n* の場合は、厳密な意味での上三角部分がマトリックス *Q* の各成分によって上書きされる。また、残りの成分は、*m* × *n* の下台形マトリックス *L* の対応する成分によって上書きされる。

tau
REAL (sgelqf の場合)
DOUBLE PRECISION (dgelqf の場合)
COMPLEX (cgelqf の場合)
DOUBLE COMPLEX (zgelqf の場合)
配列 DIMENSION は、max(1, min(*m*, *n*)) 以上。
マトリックス *Q* に関するその他の情報も格納される。

work(1)
info = 0 の場合は、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の *lwork* の値が *work*(1) に格納される。これ以降の実行には、この *lwork* の値を使用する。

info
INTEGER
info = 0 の場合は、正常に終了する。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

パフォーマンスを改善するには、*lwork* =*m***blocksize* に設定する。*blocksize* は、**ブロック化アルゴリズム**の最適パフォーマンスに必要な値で、マシンによって異なる (通常は 16 ～ 64)。

用意すべきワークスペースの量が不明な場合は、最初は *lwork* に大きな値を設定する。終了時に *work*(1) の値を調べ、これ以降はその値を使用する。

計算から得られる因子分解は、マトリックス $A + E$ ($\|E\|_2 = O(\varepsilon)\,\|A\|_2$) を因子分解したそのものである。

実数の場合の浮動小数演算のおおよその総数は、次のとおりである。

$(4/3)n^3$	($m = n$ の場合)
$(2/3)n^2(3m-n)$	($m > n$ の場合)
$(2/3)m^2(3n-m)$	($m < n$ の場合)

複素数の場合は、この 4 倍になる。

与えられたマトリックス B のすべての列 b に対して $\|Ax - b\|_2$ が最小になるような劣決定最小二乗問題の最小ノルム解を見つけるには、以下のルーチンを呼び出す。

`?gelqf` (このルーチン)	因子分解 $A = LQ$ を行う。
?trsm (BLAS ルーチン)	$LY = B$ を Y について解く。
?ormlq	$X = (Q_1)^T Y$ を求める (実マトリックスの場合)
?unmlq	$X = (Q_1)^H Y$ を求める (複素マトリックスの場合)

(計算から得られる X の列は、最小ノルムの解ベクトル x である。A は、$m \times n$ のマトリックス ($m < n$) である。Q_1 は、Q の先頭から m 個の列を表す。)

Q の成分を明示的に求めるには、以下のルーチンを呼び出す。

?orglq	(実マトリックスの場合)
?unglq	(複素マトリックスの場合)

?orglq

?gelqf で求めた *LQ* 因子分解の実数型
直交マトリックス *Q* を生成する。

```
call sorglq ( m, n, k, a, lda, tau, work, lwork, info )
call dorglq ( m, n, k, a, lda, tau, work, lwork, info )
```

説明

このルーチンは、ルーチン sgelqf/dgelqf (5-25 ページを参照) によって求めた *LQ* 因子分解の直交マトリックス *Q* ($n \times n$) の全部または一部を生成する。このルーチンは、ルーチン sgelqf/dgelqf を呼び出した後で使用する。

Q は、通常、$p \times n$ のマトリックス *A* ($n \geq p$) の *LQ* 因子分解によって決定される。マトリックス *Q* の全体を計算するには、次のように呼び出す。

```
call ?orglq ( n, n, p, a, lda, tau, work, lwork, info )
```

Q の先頭から *p* 個の行 (*A* の行で張られた空間内で直交基を形成) を計算するには、次のように呼び出す。

```
call ?orglq ( p, n, p, a, lda, tau, work, lwork, info )
```

A の先頭から *k* 個の行に関して *LQ* 因子分解のマトリックス Q^k を求めるには、次のように呼び出す。

```
call ?orglq ( n, n, k, a, lda, tau, work, lwork, info )
```

Q^k の先頭から *k* 個の行 (*A* の先頭からの *k* 個の行で張られた空間内で直交基を形成) を求めるには、次のように呼び出す。

```
call ?orgqr ( k, n, k, a, lda, tau, work, lwork, info )
```

入力パラメータ

m	INTEGER。計算対象の *Q* の行数 ($0 \leq m \leq n$)。
n	INTEGER。直交マトリックス *Q* ($n \geq m$) の次数。
k	INTEGER。基本リフレクタ (その積でマトリックス *Q* を定義) の個数 ($0 \leq k \leq m$)。

a, tau, work	REAL (sorglq の場合) DOUBLE PRECISION (dorglq の場合) 配列 : *a(lda, *)* と *tau(*)* は、sgelqf/dgelqf から返された配列である。 *a* の第 2 次元は、max(1, *n*) 以上でなければならない。 *tau* の次元は、max(1, *k*) 以上でなければならない。 *work(lwork)* は、ワークスペース配列である。
lda	INTEGER。*a* の第 1 次元。max(1, *m*) 以上であること。
lwork	INTEGER。配列 *work* のサイズ。(max(1, *m*) 以上であること)。 *lwork* の推奨値は、「アプリケーション・ノート」を参照。

出力パラメータ

a	直交マトリックス Q ($n \times n$)の先頭から m 個の行によって上書きされる。
work(1)	*info* = 0 の場合は、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の *lwork* の値が *work(1)* に格納される。これ以降の実行には、この *lwork* の値を使用する。
info	INTEGER *info* = 0 の場合は、正常に終了する。 *info* = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

パフォーマンスを改善するには、*lwork* =*m***blocksize* に設定する。*blocksize* は、**ブロック化アルゴリズム**の最適パフォーマンスに必要な値で、マシンによって異なる (通常は 16 ～ 64)。用意すべきワークスペースの量が不明な場合は、最初は *lwork* に大きな値を設定する。終了時に *work(1)* の値を調べ、これ以降はその値を使用する。

計算で求めた Q は、正確な直交マトリックスとマトリックス E ($\|E\|_2 = O(\varepsilon)\,\|A\|_2$、ε はマシン精度) だけ異なる。

浮動小数演算のおおよその総数は、
$4*m*n*k - 2*(m+n)*k^2 + (4/3)*k^3$ である。
$m = k$ の場合、この値は約 $(2/3)*m^2*(3n - m)$ になる。

このルーチンの複素数版は、?unglq である。

?ormlq

?gelqf で求めた *LQ* 因子分解の直交マトリックス *Q* と実マトリックスとを掛け合わせる。

```
call sormlq ( side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork,
              info )
call dormlq ( side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork,
              info )
```

説明

このルーチンは、*m* × *n* の実マトリックス *C* と *Q* または Q^T とを掛け合わせる。*Q* は、ルーチン sgelqf/dgelqf (5-25 ページを参照) で求めた *LQ* 因子分解の直交マトリックス *Q* である。

このルーチンを使用すれば、パラメータ *side* と *trans* に応じて、*QC*、Q^TC、*CQ*、または CQ^T のいずれかのマトリックス積を求められる (*C* の値は上書きされる)。

入力パラメータ

side CHARACTER*1。'L' または 'R' でなければならない。
side ='L' の場合、*Q* または Q^T は、*C* に左側から適用される。
side ='R' の場合、*Q* または Q^T は、*C* に右側から適用される。

trans CHARACTER*1。'N' または 'T' でなければならない。
trans ='N' の場合、*C* に *Q* が掛けられる。
trans ='T' の場合、*C* に Q^T が掛けられる。

m INTEGER。マトリックス *C* の行数 ($m \geq 0$)。

n INTEGER。マトリックス *C* の列数 ($n \geq 0$)。

k INTEGER。基本リフレクタ (その積でマトリックス *Q* を定義) の個数。次の制約がある。
$0 \leq k \leq m$ (*side* ='L' の場合)
$0 \leq k \leq n$ (*side* ='R' の場合)

a, work, tau, c REAL (sormlq の場合)
DOUBLE PRECISION (dormlq の場合)
配列 :
a(*lda*, *) と *tau*(*) は、?gelqf から返された配列である。

a の第 2 次元は、次の条件を満たさなければならない。
max(1, m) 以上であること (side='L' の場合)
max(1, n) 以上であること (side='R の場合)
tau の次元は、max(1, k) 以上でなければならない。
c(ldc, *) には、マトリックス C を格納する。
c の第 2 次元は、max(1, n) 以上でなければならない。
work(lwork) は、ワークスペース配列である。

lda　INTEGER。a の第 1 次元。lda ≥ max(1, k)

ldc　INTEGER。c の第 1 次元。ldc ≥ max(1, m)

lwork　INTEGER。配列 work のサイズ。次の制約がある。
lwork ≥ max(1, n) (side='L' の場合)
lwork ≥ max(1, m) (side='R' の場合)
lwork の推奨値は、「アプリケーション・ノート」を参照。

出力パラメータ

c　(side と trans の指定に従って) QC、Q^TC、CQ、または CQ^T のいずれかの積によって上書きされる。

work(1)　info = 0 の場合は、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の lwork の値が work(1) に格納される。これ以降の実行には、この lwork の値を使用する。

info　INTEGER
info = 0 の場合は、正常に終了する。
info = -i の場合は、i 番目のパラメータの値が不正である。

アプリケーション・ノート

パフォーマンスを改善するには、lwork = n*blocksize (side='L' の場合) または lwork = m*blocksize (side='R' の場合) に設定する。*blocksize* は、**ブロック化アルゴリズム**の最適パフォーマンスに必要な値で、マシンによって異なる (通常は 16 ～ 64)。用意すべきワークスペースの量が不明な場合は、最初は lwork に大きな値を設定する。終了時に work(1) の値を調べ、これ以降はその値を使用する。

このルーチンの複素数版は、?unmlq である。

?unglq

?gelqf で求めた *LQ* 因子分解の複素
ユニタリ・マトリックス *Q* を生成する。

```
call cunglq ( m, n, k, a, lda, tau, work, lwork, info )
call zunglq ( m, n, k, a, lda, tau, work, lwork, info )
```

説明

このルーチンは、ルーチン cgelqf/zgelqf (5-25 ページを参照) で求めた *LQ* 因子分解のユニタリ・マトリックス *Q* ($n \times n$) の全部または一部を生成する。このルーチンは、cgelqf/zgelqf を呼び出した後で使用する。

Q は通常、$p \times n$ のマトリックス *A* ($n \geq p$) の *LQ* 因子分解によって決定される。マトリックス *Q* の全体を計算するには、次のように呼び出す。

```
call ?unglq ( n, n, p, a, lda, tau, work, lwork, info )
```

Q の先頭から p 個の行 (*A* の行で張られた空間内で直交基を形成) を計算するには、次のように呼び出す。

```
call ?unglq ( p, n, p, a, lda, tau, work, lwork, info )
```

A の先頭から k 個の行に関して *LQ* 因子分解のマトリックス Q^k を求めるには、次のように呼び出す。

```
call ?unglq ( n, n, k, a, lda, tau, work, lwork, info )
```

Q^k の先頭から k 個の行 (*A* の先頭からの k 個の行で張られた空間内で直交基を形成) を求めるには、次のように呼び出す。

```
call ?ungqr ( k, n, k, a, lda, tau, work, lwork, info )
```

入力パラメータ

m	INTEGER。計算対象の *Q* の行数 ($0 \leq m \leq n$)。
n	INTEGER。ユニタリ・マトリックス *Q* ($n \geq m$) の次数。
k	INTEGER。基本リフレクタ (その積でマトリックス *Q* を定義) の個数 ($0 \leq k \leq m$)。

a, tau, work	COMPLEX (cunglq の場合) DOUBLE COMPLEX (zunglq の場合) 配列 : *a(lda, *)* と *tau(*)* は、sgelqf/dgelqf から返された配列である。 *a* の第 2 次元は、max(1, *n*) 以上でなければならない。 *tau* の次元は、max(1, *k*) 以上でなければならない。 *work* (*lwork*) は、ワークスペース配列である。
lda	INTEGER。*a* の第 1 次元。max(1, *m*) 以上であること。
lwork	INTEGER。配列 *work* のサイズ (max(1, *m*) 以上であること)。 *lwork* の推奨値は、「アプリケーション・ノート」を参照。

出力パラメータ

a	ユニタリ・マトリックス Q ($n \times n$) の先頭から *m* 個の行によって上書きされる。
work(1)	*info* = 0 の場合は、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の *lwork* の値が *work(1)* に格納される。これ以降の実行には、この *lwork* の値を使用する。
info	INTEGER *info* = 0 の場合は、正常に終了する。 *info* = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

パフォーマンスを改善するには、*lwork* = *m***blocksize* に設定する。*blocksize* は、**ブロック化アルゴリズム**の最適パフォーマンスに必要な値で、マシンによって異なる (通常は 16 ～ 64)。用意すべきワークスペースの量が不明な場合は、最初は *lwork* に大きな値を設定する。終了時に *work(1)* の値を調べ、これ以降はその値を使用する。

計算で求めた Q は、正確なユニタリ・マトリックスとマトリックス E ($\|E\|_2 = O(\varepsilon)\,\|A\|_2$、ε はマシン精度) だけ異なる。

浮動小数演算のおおよその総数は、
$16*m*n*k - 8*(m+n)*k^2 + (16/3)*k^3$ である。
$m = k$ の場合、この値は約 $(8/3)*m^2*(3n - m)$ になる。

このルーチンの実数版は、?orglq である。

?unmlq

?gelqf で求めた *LQ* 因子分解のユニタリ・マトリックス *Q* と複素マトリックスとを掛け合わせる。

```
call cunmlq ( side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork,
              info )
call zunmlq ( side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork,
              info )
```

説明

このルーチンは、*m* × *n* の複素マトリックス *C* と *Q* または Q^H とを掛け合わせる。*Q* は、ルーチン cgelqf/zgelqf (5-25 ページを参照) で求めた *LQ* 因子分解のユニタリ・マトリックス *Q* である。

このルーチンを使用すれば、パラメータ *side* と *trans* に応じて、*QC*、Q^HC、*CQ*、または CQ^H のいずれかのマトリックス積を求められる (*C* の値は上書きされる)。

入力パラメータ

side CHARACTER*1。'L' または 'R' でなければならない。
side ='L' の場合、*Q* または Q^H は、*C* に左側から適用される。
side ='R' の場合、*Q* または Q^H は、*C* に右側から適用される。

trans CHARACTER*1。'N' または 'C' でなければならない。
trans ='N' の場合、*C* に *Q* が掛けられる。
trans ='C' の場合は、*C* に Q^H が掛けられる。

m INTEGER。マトリックス *C* の行数 ($m \geq 0$)。

n INTEGER。マトリックス *C* の列数 ($n \geq 0$)。

k INTEGER。基本リフレクタ (その積でマトリックス *Q* を定義) の個数。次の制約がある。
$0 \leq k \leq m$ (*side* ='L' の場合)
$0 \leq k \leq n$ (*side* ='R' の場合)

a, work, tau, c COMPLEX (cunmlq の場合)
DOUBLE COMPLEX (zunmlq の場合)
配列 :
a(*lda*, *) と *tau*(*) は、?gelqf から返された配列である。

a の第 2 次元は、次の条件を満たさなければならない。
max(1, m) 以上であること (side='L' の場合)
max(1, n) 以上であること (side='R' の場合)
tau の次元は、max(1, k) 以上でなければならない。
c(ldc, *) には、マトリックス C を格納する。
c の第 2 次元は、max(1, n) 以上でなければならない。
work(lwork) は、ワークスペース配列である。

lda　INTEGER。a の第 1 次元。lda ≥ max(1, k)

ldc　INTEGER。c の第 1 次元。ldc ≥ max(1, m)

lwork　INTEGER。配列 work のサイズ。次の制約がある。
lwork ≥ max(1, n) (side='L' の場合)
lwork ≥ max(1, m) (side='R' の場合)
lwork の推奨値は、「アプリケーション・ノート」を参照。

出力パラメータ

c　(side と trans の指定に従って) QC、Q^HC、CQ、または CQ^H のいずれかの積によって上書きされる。

work(1)　info = 0 の場合は、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の lwork の値が work(1) に格納される。これ以降の実行には、この lwork の値を使用する。

info　INTEGER
info = 0 の場合は、正常に終了する。
info = -i の場合は、i 番目のパラメータの値が不正である。

アプリケーション・ノート

パフォーマンスを改善するには、lwork = n*blocksize (side='L' の場合) または lwork = m*blocksize (side='R' の場合) に設定する。*blocksize* は、**ブロック化アルゴリズム**の最適パフォーマンスに必要な値で、マシンによって異なる (通常は 16 ～ 64)。用意すべきワークスペースの量が不明な場合は、最初は lwork に大きな値を設定する。終了時に work(1) の値を調べ、これ以降はその値を使用する。

このルーチンの実数版は、?ormlq である。

?geqlf

m × *n* の一般マトリックスの *QL* 因子分解を実行する。

```
call sgeqlf ( m, n, a, lda, tau, work, lwork, info )
call dgeqlf ( m, n, a, lda, tau, work, lwork, info )
call cgeqlf ( m, n, a, lda, tau, work, lwork, info )
call zgeqlf ( m, n, a, lda, tau, work, lwork, info )
```

説明

このルーチンは、*m* × *n* の一般マトリックス *A* の *QL* 因子分解を実行する。ピボット演算は出力パラメータ行わない。

このルーチンは、マトリックス *Q* を明示的に生成するものではない。マトリックス *Q* は、min(*m*, *n*) の**基本リフレクタ**の積で表される。このように表されるマトリックス *Q* を操作するために、一連のルーチンが提供されている。

入力パラメータ

m	INTEGER。マトリックス *A* の行数 (*m* ≥ 0)。
n	INTEGER。マトリックス *A* の列数 (*n* ≥ 0)。
a, *work*	REAL (sgeqlf の場合) DOUBLE PRECISION (dgeqlf の場合) COMPLEX (cgeqlf の場合) DOUBLE COMPLEX (zgeqlf の場合) 配列 : *a*(*lda*, *) には、マトリックス *A* を格納する。 *a* の第 2 次元のサイズは、max(1, *n*) 以上でなければならない。 *work*(*lwork*) は、ワークスペース配列。
lda	INTEGER。*a* の第 1 次元のサイズ。max(1, *m*) 以上。
lwork	INTEGER。配列 *work* のサイズ。max(1, *n*) 以上。 *lwork* の推奨値は、「アプリケーション・ノート」を参照。

出力パラメータ

a

終了時に、因子分解データによって次のように上書きされる。

m ≥ *n* の場合は、部分配列 *a*(*m*-*n*+1:*m*, 1:*n*) の下三角部分に、*n* × *n* の下三角マトリックス *L* が格納される。
m ≤ *n* の場合は、(*n*-*m*) 番目の優対角成分とその下の各成分に、*m* × *n* の下台形マトリックス *L* が格納される。
どちらの場合も、残りの成分と配列 *tau* によって、直交 / ユニタリ・マトリックス *Q* が、基本リフレクタの積として表される。

tau

REAL (sgeqlf の場合)
DOUBLE PRECISION (dgeqlf の場合)
COMPLEX (cgeqlf の場合)
DOUBLE COMPLEX (zgeqlf の場合)
配列 DIMENSION は max(1, min(*m*, *n*)) 以上。
マトリックス *Q* に対する基本リフレクタのスカラー係数が入る。

work(1)

info = 0 の場合は、最適パフォーマンスに必要な *lwork* の最小値が、ルーチンの終了時に *work*(1) に格納される。

info

INTEGER
info = 0 の場合は、正常終了している。
info = -*i*, の場合は、*i* 番目のパラメータ値が不正である。

アプリケーション・ノート

パフォーマンスを改善するには、*lwork*=*n***blocksize* に設定する。
blockzise は、**ブロック化アルゴリズム**の最適パフォーマンスに必要な値で、マシンによって異なる (通常は 16 ～ 64)。必要なワークスペース・サイズが不明な場合は、*lwork* に大きな値を設定して実行する。ルーチン終了後に *work*(1) の値を調べ、以降はその値を使用する。
関連ルーチン :

?orgql	マトリックス Q を生成する (実マトリックスの場合)。
?ungql	マトリックス Q を生成する (複素マトリックスの場合)。
?ormql	マトリックス Q を適用する (実マトリックスの場合)。
?unmql	マトリックス Q を適用する (複素マトリックスの場合)。

?orgql

?geqlf で求めた *QL* 因子分解の
実マトリックス *Q* を生成する。

```
call sorgql ( m, n, k, a, lda, tau, work, lwork, info )
call dorgql ( m, n, k, a, lda, tau, work, lwork, info )
```

説明

このルーチンは、直交列を持つ $m \times n$ の実マトリックス Q を生成する。これは、ルーチン sgeqlf/dgeqlf から返された k 個の基本リフレクタ H_i (次数は m) の積の最終 n 列で定義される ($Q = H_k \cdots H_2 H_1$)。このルーチンは、sgeqlf/dgeqlf を呼び出してから使用する。

入力パラメータ

m	INTEGER。マトリックス Q の行数 ($m \geq 0$)。
n	INTEGER。マトリックス Q の列数 ($m \geq n \geq 0$)。
k	INTEGER。その積がマトリックス Q となる基本リフレクタの個数 ($n \geq k \geq 0$)。
a, tau, work	REAL (sorgql の場合) DOUBLE PRECISION (dorgql の場合) 配列 : *a(lda, *)*, *tau(*)*, *work(lwork)*。 *a* 呼び出し時は、sgeqlf/dgeqlf で配列引数 *a* の最終 *k* 列に返された通りに、*a* の (*n* - *k* + *i*) 番目の列に、基本リフレクタ H_i (ただし、i = 1, 2, ..., *k*) を定義するベクトルが入っていなければならない。 *tau*(i) には、sgeqlf/dgeqlf から返された通りに、基本リフレクタ *H*i のスカラー係数が入っていなければならない。 *a* の第 2 次元のサイズは、max(1, *n*) 以上でなければならない。 *tau* のサイズは、max(1, *k*) 以上でなければならない。 *work(lwork)* は、ワークスペース配列。

lda	INTEGER。*a* の第 1 次元のサイズ。max(1, *m*) 以上。
lwork	INTEGER。配列 *work* のサイズ。max(1, *n*) 以上。 *lwork* の推奨値は、「アプリケーション・ノート」を参照。

出力パラメータ

a	*m* × *n* のマトリックス *Q* で上書きされる。
work(1)	*info* = 0 の場合は、最適パフォーマンスに必要な *lwork* の最小値が、ルーチンの終了時に *work(1)* に格納される。以降の実行では、この *lwork* 値を使用する。
info	INTEGER *info* = 0 の場合は、正常終了している。 *info* = *-i* の場合は、*i* 番目のパラメータ値が不正である。

アプリケーション・ノート

パフォーマンスを改善するには、*lwork*=*n***blocksize* に設定する。*blocksize* は、**ブロック化アルゴリズム**の最適パフォーマンスに必要な値で、マシンによって異なる (通常は 16 ～ 64)。必要なワークスペース・サイズが不明な場合は、*lwork* に大きな値を設定して実行する。ルーチン終了後に *work(1)* の値を調べ、以降はその値を使用する。

このルーチンの複素数版は、?ungql である。

?ungql

?geqlf で求めた *QL* 因子分解の複素マトリックスを生成する。

```
call cungql ( m, n, k, a, lda, tau, work, lwork, info )
call zungql ( m, n, k, a, lda, tau, work, lwork, info )
```

説明

このルーチンは、直交列を持つ $m \times n$ の複素マトリックス Q を生成する。これは、ルーチン cgeqlf/zgeqlf から返された k 個の基本リフレクタ H_i (次数は m) の積の最終 n 列で定義される ($Q = H_k \cdots H_2 H_1$)。このルーチンは、cgeqlf/zgeqlf を呼び出してから使用する。

入力パラメータ

m	INTEGER。マトリックス Q の行数 ($m \geq 0$)。
n	INTEGER。マトリックス Q の列数 ($m \geq n \geq 0$)。
k	INTEGER。その積がマトリックス Q となる基本リフレクタの個数 ($n \geq k \geq 0$)。
a, tau, work	COMPLEX (cungql の場合) DOUBLE COMPLEX (zungql の場合) 配列 :*a(lda, *)*, *tau(*)*, *work(lwork)* 呼び出し時は、cgeqlf/zgeqlf で配列引数 *a* の最終 *k* 列に返された通りに、*a* の (*n* - *k* + *i*) 番目の列に、基本リフレクタ H_i (ただし、i = 1, 2, ..., *k*) を定義するベクトルが入っていなければならない。 *tau*(i) には、cgeqlf/zgeqlf から返された通りに、基本リフレクタ H_i のスカラー係数が入っていなければならない。 *a* の第 2 次元のサイズは、max(1, *n*) 以上でなければならない。 *tau* のサイズは、max(1, *k*) 以上でなければならない。 *work(lwork)* は、ワークスペース配列。

lda	INTEGER。*a* の第 1 次元のサイズ。max(1, *m*) 以上。
lwork	INTEGER。配列 *work* のサイズ。max(1, *n*) 以上。 *lwork* の推奨値は、「アプリケーション・ノート」を参照。

出力パラメータ

a	*m* × *n* のマトリックス *Q* で上書きされる。
work(1)	*info* = 0 の場合は、最適パフォーマンスに必要な *lwork* の最小値が、ルーチンの終了時に *work*(1) に格納される。以降の実行では、この *lwork* 値を使用する。
info	INTEGER *info* = 0 の場合は、正常終了している。 *info* = -*i* の場合は、*i* 番目のパラメータ値が不正である。

アプリケーション・ノート

パフォーマンスを改善するには、*lwork*=*n***blocksize* に設定する。*blocksize* は、**ブロック化アルゴリズム**の最適パフォーマンスに必要な値で、マシンによって異なる (通常は 16 ～ 64)。必要なワークスペース・サイズが不明な場合は、*lwork* に大きな値を設定して実行する。ルーチン終了後に *work*(1) の値を調べ、以降はその値を使用する。

このルーチンの実数版は、?orgql である。

?ormql

?geqlf で求めた QL 因子分解の直交マトリックス Q に、実マトリックスを掛ける。

```
call sormql ( side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork,
              info )
call dormql ( side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork,
              info )
```

説明

このルーチンは、$m \times n$ の実マトリックス C に、Q または Q^T を掛ける。Q は、ルーチン sgeqlf/dgeqlf で求めた QL 因子分解の直交マトリックス Q である。

ルーチン ?ormql は、パラメータ *side* と *trans* の値に従って、QC、Q^TC、CQ、CQ^T のどれかのマトリックス積を計算する (C は計算結果で上書きされる)。

入力パラメータ

side CHARACTER*1。'L' または 'R' でなければならない。
side ='L' の場合、Q または Q^T が C の左側から掛けられる。
side ='R' の場合、Q または Q^T が C の右側から掛けられる。

trans CHARACTER*1。'N' または 'T' でなければならない。
trans ='N' の場合、C に Q が掛けられる。
trans ='T' の場合、C に Q^T が掛けられる。

m INTEGER。マトリックス C の行数 ($m \geq 0$)。

n INTEGER。マトリックス C の列数 ($n \geq 0$)。

k INTEGER。その積がマトリックス Q となる基本リフレクタの個数。次の制約がある。
$0 \leq k \leq m$ (*side* ='L' の場合)
$0 \leq k \leq n$ (*side* ='R' の場合)

a, tau, c, work REAL (sormql の場合)
DOUBLE PRECISION (dormql の場合)
配列 : *a(lda, *)*, *tau(*)*, *c(ldc, *)*, *work(lwork)*
呼び出し時は、sgeqlf/dgeqlf で配列引数 *a* の最終 *k* 列に返された通りに、*a* の *i* 番目の列に、基本リフレクタ H_i (ただし、i = 1, 2, ..., *k*) を定義するベクトルが入っていなければならない。
a の第 2 次元のサイズは、max(1, *k*) 以上でなければならない。
tau(i) には、sgeqlf/dgeqlf から返された通りに、基本リフレクタ H_i のスカラー係数が入っていなければならない。
tau のサイズは、max(1, *k*) 以上でなければならない。
*c(ldc, *)* には、*m* × *n* のマトリックス *C* を格納する。
c の第 2 次元のサイズは、max(1, *n*) 以上でなければならない。
work(lwork) は、ワークスペース配列。

lda INTEGER。*a* の第 1 次元のサイズ。
side = 'L' の場合は、*lda* ≥ max(1, *m*)
side = 'R' の場合は、*lda* ≥ max(1, *n*)

ldc INTEGER。*c* の第 1 次元のサイズ。*ldc* ≥ max(1, *m*)

lwork INTEGER。配列 *work* のサイズ。次の制約がある。
lwork ≥ max(1, *n*) (*side* = 'L' の場合)
lwork ≥ max(1, *m*) (*side* = 'R' の場合)
lwork の推奨値は、「アプリケーション・ノート」を参照。

出力パラメータ

c *side* と *trans* の指定に従って、QC、Q^TC、CQ、CQ^T のどれかの積で上書きされる。

work(1) *info* = 0 の場合は、最適パフォーマンスに必要な *lwork* の最小値が、ルーチンの終了時に *work(1)* に格納される。以降の実行では、この *lwork* 値を使用する。

info INTEGER
info = 0 の場合は、正常終了している。
info = -*i* の場合は、*i* 番目のパラメータ値が不正である。

アプリケーション・ノート

パフォーマンスを改善するには、`lwork`=*n***blocksize* (`side='L'` の場合) または `lwork`=*m***blocksize* (`side='R'` の場合) に設定する。*blocksize* は、**ブロック化アルゴリズム**の最適パフォーマンスに必要な値で、マシンによって異なる (通常は 16 ～ 64)。必要なワークスペース・サイズが不明な場合は、`lwork` に大きな値を設定して実行する。ルーチン終了後に `work(1)` の値を調べ、以降はその値を使用する。

このルーチンの複素数版は、?unmql である。

?unmql

?geqlf で求めた QL 因子分解のユニタリ・マトリックス Q に、複素マトリックスを掛ける。

```
call cunmql ( side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork,
              info )
call zunmql ( side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork,
              info )
```

説明

このルーチンは、$m \times n$ の複素マトリックス C に、Q または Q^H を掛ける。Q は、ルーチン cgeqlf/zgeqlf で求めた QL 因子分解のユニタリ・マトリックス Q である。

ルーチン ?unmql は、パラメータ *side* と *trans* の値に従って、QC、Q^HC、CQ、CQ^H のどれかのマトリックス積を計算する (C は計算結果で上書きされる)。

入力パラメータ

side CHARACTER*1。'L' または 'R' でなければならない。
side ='L' の場合、Q または Q^H が C の左側から掛けられる。
side ='R' の場合、Q または Q^H が C の右側から掛けられる。

trans CHARACTER*1。'N' または 'C' でなければならない。
trans ='N' の場合、C に Q が掛けられる。
trans ='C' の場合、C に Q^H が掛けられる。

m INTEGER。マトリックス C の行数 ($m \geq 0$)。

n INTEGER。マトリックス C の列数 ($n \geq 0$)。

k INTEGER。その積がマトリックス Q となる基本リフレクタの個数。次の制約がある。
$0 \leq k \leq m$ (*side* ='L' の場合)
$0 \leq k \leq n$ (*side* ='R' の場合)

a, tau, c, work COMPLEX (cunmql の場合)
DOUBLE COMPLEX (zunmql の場合)
配列 :*a(lda, *)*, *tau(*)*, *c(ldc, *)*, *work(lwork)*
呼び出し時は、cgeqlf/zgeqlf で配列引数 *a* の最終 *k* 列に返された通りに、*a* の *i* 番目の列に、基本リフレクタ H_i (ただし、i = 1, 2, ..., *k*) を定義するベクトルが入っていなければならない。
a の第 2 次元のサイズは、max(1, *k*) 以上でなければならない。
tau(i) には、cgeqlf/zgeqlff から返された通りに、基本リフレクタ H_i のスカラー係数が入っていなければならない。
tau のサイズは、max(1, *k*) 以上でなければならない。
*c(ldc, *)* には、*m* × *n* のマトリックス *C* を格納する。
c の第 2 次元のサイズは、max(1, *n*) 以上でなければならない。
work(lwork) は、ワークスペース配列。

lda INTEGER。*a* の第 1 次元のサイズ。
side = 'L' の場合は、*lda* ≥ max(1, *m*)
side = 'R' の場合は、*lda* ≥ max(1, *n*)

ldc INTEGER。*c* の第 1 次元のサイズ。*ldc* ≥ max((1, *m*)

lwork INTEGER。配列 *work* のサイズ。次の制約がある。
lwork ≥ max(1, *n*) (*side* = 'L' の場合)
lwork ≥ max(1, *m*) (*side* = 'R' の場合)
lwork の推奨値は、「アプリケーション・ノート」を参照。

出力パラメータ

c *side* と *trans* の指定に従って、QC、Q^HC、CQ、CQ^H のどれかの積で上書きされる。

work(1) *info* = 0 の場合は、最適パフォーマンスに必要な *lwork* の最小値が、ルーチンの終了時に *work(1)* に格納される。以降の実行では、この *lwork* 値を使用する。

info INTEGER
info = 0 の場合は、正常終了している。
info = -*i* の場合は、*i* 番目のパラメータ値が不正である。

アプリケーション・ノート

パフォーマンスを改善するには、`lwork`=`n`**blocksize* (`side='L'` の場合) または `lwork`=`m`**blocksize* (`side='R'` の場合) に設定する。*blocksize* は、**ブロック化アルゴリズム**の最適パフォーマンスに必要な値で、マシンによって異なる (通常は 16 ～ 64)。必要なワークスペース・サイズが不明な場合は、`lwork` に大きな値を設定して実行する。ルーチン終了後に `work(1)` の値を調べ、以降はその値を使用する。

このルーチンの実数版は、?ormql である。

?gerqf

m × *n* の一般マトリックスの *RQ* 因子分解を実行する。

```
call sgerqf ( m, n, a, lda, tau, work, lwork, info )
call dgerqf ( m, n, a, lda, tau, work, lwork, info )
call cgerqf ( m, n, a, lda, tau, work, lwork, info )
call zgerqf ( m, n, a, lda, tau, work, lwork, info )
```

説明

このルーチンは、*m* × *n* の一般マトリックス *A* の *RQ* 因子分解を実行する。
ピボット演算は行わない。

このルーチンは、マトリックス *Q* を明示的に生成するものではない。マトリックス *Q* は、min(*m*, *n*) の**基本リフレクタ**の積で表される。このように表されるマトリックス *Q* を操作するために、一連のルーチンが提供されている。

入力パラメータ

m	INTEGER。マトリックス *A* の行数 (*m* ≥ 0)。
n	INTEGER。マトリックス *A* の列数 (*n* ≥ 0)。
a, *work*	REAL (sgerqf の場合) DOUBLE PRECISION (dgerqf の場合) COMPLEX (cgerqf の場合) DOUBLE COMPLEX (zgerqf の場合) 配列 : *a*(*lda*, *) には、*m* × *n* のマトリックス *A* を格納する。 *a* の第 2 次元のサイズは、max(1, *n*) 以上でなければならない。 *work*(*lwork*) は、ワークスペース配列。
lda	INTEGER。*a* の第 1 次元のサイズ。max(1, *m*) 以上。
lwork	INTEGER。配列 *work* のサイズ。 *lwork* ≥ max(1, *m*) *lwork* の推奨値は、「アプリケーション・ノート」を参照。

出力パラメータ

a	終了時に、因子分解データによって次のように上書きされる。 $m \le n$ の場合は、部分配列 *a*(1:*m*, *n*-*m*+1:*n*) の上三角部分に、$m \times m$ の上三角マトリックス *R* が格納される。 $m \ge n$ の場合は、(*m*-*n*) 番目の劣対角成分とその上の各成分に、$m \times n$ の上台形マトリックス *R* が格納される。 どちらの場合も、残りの成分と配列 *tau* によって、直交 / ユニタリ・マトリックス *Q* が、min(*m*, *n*) 個の基本リフレクタの積として表される。
tau	REAL (sgerqf の場合) DOUBLE PRECISION (dgerqf の場合) COMPLEX (cgerqf の場合) DOUBLE COMPLEX (zgerqf の場合) 配列 DIMENSION は max(1, min(*m*, *n*)) 以上。 マトリックス *Q* に対する基本リフレクタのスカラー係数が入る。
work(1)	*work*(1) *info* = 0 の場合は、最適パフォーマンスに必要な *lwork* の最小値が、ルーチンの終了時に *work*(1) に格納される。
info	INTEGER *info* = 0 の場合は、正常終了している。 *info* = -*i* の場合は、*i* 番目のパラメータ値が不正である。

アプリケーション・ノート

パフォーマンスを改善するには、*lwork*=*m***blocksize* に設定する。*blocksize* は、**ブロック化アルゴリズム**の最適パフォーマンスに必要な値で、マシンによって異なる (通常は 16 ～ 64)。必要なワークスペース・サイズが不明な場合は、*lwork* に大きな値を設定して実行する。ルーチン終了後に *work*(1) の値を調べ、以降はその値を使用する。

関連ルーチン :

?orgrq	マトリックス Q を生成する (実マトリックスの場合)。
?ungrq	マトリックス Q を生成する (複素マトリックスの場合)。
?ormrq	マトリックス Q を適用する (実マトリックスの場合)。
?unmrq	マトリックス Q を適用する (複素マトリックスの場合)。

?orgrq

?gerqf で求めた *RQ* 因子分解の
実マトリックス *Q* を生成する。

```
call sorgrq ( m, n, k, a, lda, tau, work, lwork, info )
call dorgrq ( m, n, k, a, lda, tau, work, lwork, info )
```

説明

このルーチンは、直交行を持つ $m \times n$ の実マトリックス Q を生成する。これは、ルーチン sgerqf/dgerqf から返された k 個の基本リフレクタ H_i (次数は n) の積の最終 m 行で定義される ($Q = H_1 H_2 \cdots H_k$)。このルーチンは、sgerqf/dgerqf を呼び出してから使用する。

入力パラメータ

m	INTEGER。マトリックス Q の行数 ($m \geq 0$)。
n	INTEGER。マトリックス Q の列数 ($n \geq m$)。
k	INTEGER。その積がマトリックス Q となる基本リフレクタの個数 ($m \geq k \geq 0$)。
a, tau, work	REAL (sorgrq の場合) DOUBLE PRECISION (dorgrq の場合) 配列 : *a(lda, *)*, *tau(*)*, *work(lwork)* 呼び出し時は、sgerqf/dgerqf で配列引数 *a* の最終 *k* 行に返された通りに、*a* の (*m* - *k* + *i*)) 番目の行に、基本リフレクタ H_i (ただし、i = 1, 2, ..., *k*) を定義するベクトルが入っていなければならない。 *tau*(i) には、sgerqf/dgerqf から返された通りに、基本リフレクタ H_i のスカラー係数が入っていなければならない。 *a* の第 2 次元のサイズは、max(1, *n*) 以上でなければならない。 *tau* のサイズは、max(1, *k*) 以上でなければならない。 *work(lwork)* は、ワークスペース配列。

lda	INTEGER。*a* の第 1 次元のサイズ。max(1, *m*) 以上。
lwork	INTEGER。配列 *work* のサイズ。max(1, *m*) 以上。 *lwork* の推奨値は、「アプリケーション・ノート」を参照。

出力パラメータ

a	*m* × *n* のマトリックス *Q* で上書きされる。
work(1)	*info* = 0 の場合は、最適パフォーマンスに必要な *lwork* の最小値が、ルーチンの終了時に *work*(1) に格納される。以降の実行では、この *lwork* 値を使用する。
info	INTEGER *info* = 0 の場合は、正常終了している。 *info* = -*i* の場合は、*i* 番目のパラメータ値が不正である。

アプリケーション・ノート

パフォーマンスを改善するには、*lwork*=*m***blocksize* に設定する。*blocksize* は、**ブロック化アルゴリズム**の最適パフォーマンスに必要な値で、マシンによって異なる (通常は 16 ～ 64)。必要なワークスペース・サイズが不明な場合は、*lwork* に大きな値を設定して実行する。ルーチン終了後に *work*(1) の値を調べ、以降はその値を使用する。

このルーチンの複素数版は、?ungrq である。

?ungrq

?gerqf で求めた *RQ* 因子分解の複素マトリックス *Q* を生成する。

```
call cungrq ( m, n, k, a, lda, tau, work, lwork, info )
call zungrq ( m, n, k, a, lda, tau, work, lwork, info )
```

説明

このルーチンは、直交行を持つ $m \times n$ の複素マトリックス Q を生成する。これは、ルーチン sgerqf/dgerqf から返された k 個の基本リフレクタ H_i (次数は n) の積の最終 m 行で定義される ($Q = H_1^H H_2^H \cdots H_k^H$)。このルーチンは、sgerqf/dgerqf を呼び出してから使用する。

入力パラメータ

m	INTEGER。マトリックス Q の行数 ($m \geq 0$)。
n	INTEGER。マトリックス Q の列数 ($n \geq m$)。
k	INTEGER。その積がマトリックス Q となる基本リフレクタの個数 ($m \geq k \geq 0$)。
a, tau, work	REAL (cungrq の場合) DOUBLE PRECISION (zungrq の場合) 配列 : *a(lda, *)*, *tau(*)*, *work(lwork)* 呼び出し時は、sgerqf/dgerqf で配列引数 *a* の最終 *k* 行に返された通りに、*a* の (*m* - *k* + *i*) 番目の行に、基本リフレクタ H_i (ただし、i = 1, 2, ..., *k*) を定義するベクトルが入っていなければならない。 *tau*(i) には、sgerqf/dgerqf から返された通りに、基本リフレクタ H_i のスカラー係数が入っていなければならない。 *a* の第 2 次元のサイズは、max(1, *n*) 以上でなければならない。 *tau* のサイズは、max(1, *k*) 以上でなければならない。 *work(lwork)* は、ワークスペース配列。

lda	INTEGER。*a* の第 1 次元のサイズ。max(1, *m*) 以上。
lwork	INTEGER。配列 *work* のサイズ。max(1, *m*) 以上。 *lwork* の推奨値は、「アプリケーション・ノート」を参照。

出力パラメータ

a	*m* × *n* のマトリックス *Q* で上書きされる。
work(1)	*info* = 0 の場合は、最適パフォーマンスに必要な *lwork* の最小値が、ルーチンの終了時に *work*(1) に格納される。以降の実行では、この *lwork* 値を使用する。
info	INTEGER *info* = 0 の場合は、正常終了している。 *info* = -*i* の場合は、*i* 番目のパラメータ値が不正である。

アプリケーション・ノート

パフォーマンスを改善するには、*lwork*=*m***blocksize* に設定する。*blocksize* は、**ブロック化アルゴリズム**の最適パフォーマンスに必要な値で、マシンによって異なる (通常は 16 ～ 64)。必要なワークスペース・サイズが不明な場合は、*lwork* に大きな値を設定して実行する。ルーチン終了後に *work*(1) の値を調べ、以降はその値を使用する。

このルーチンの実数版は、?orgrq である。

?ormrq

?gerqf で求めた RQ 因子分解の直交マトリックス Q に、実マトリックスを掛ける。

```
call sormrq ( side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork,
              info )
call dormrq ( side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork,
              info )
```

説明

このルーチンは、$m \times n$ の実マトリックス C に、Q または Q^T を掛ける。Q は、ルーチン sgerqf/dgerqf から返された k 個の基本リフレクタ H_i の積で定義される直交マトリックスである ($Q = H_1 H_2 \cdots H_k$)。

このルーチンは、パラメータ *side* と *trans* の値に従って、QC、Q^TC、CQ、CQ^T のどれかのマトリックス積を計算する (C は計算結果で上書きされる)。

入力パラメータ

side
: CHARACTER*1。'L' または 'R' でなければならない。
side ='L' の場合、Q または Q^T が C の左側から掛けられる。
side ='R' の場合、Q または Q^T が C の右側から掛けられる。

trans
: CHARACTER*1。'N' または 'T' でなければならない。
trans ='N' の場合、C に Q が掛けられる。
trans ='T' の場合、C に Q^T が掛けられる。

m
: INTEGER。マトリックス C の行数 ($m \geq 0$)。

n
: INTEGER。マトリックス C の列数 ($n \geq 0$)。

k
: INTEGER。その積がマトリックス Q となる基本リフレクタの個数。次の制約がある。
$0 \leq k \leq m$ (*side* ='L' の場合)
$0 \leq k \leq n$ (*side* ='R' の場合)

a, tau, c, work REAL (sormrq の場合)
DOUBLE PRECISION (dormrq の場合)
配列 : *a(lda, *)*, *tau(*)*, *c(ldc, *)*, *work(lwork)*。
呼び出し時は、sgerqf/dgerqf で配列引数 *a* の最終 *k* 行に返された通りに、*a* の *i* 番目の行に、基本リフレクタ H_i (ただし、i = 1, 2, ..., *k*) を定義するベクトルが入っていなければならない。
a の第 2 次元のサイズは、max(1, *m*) 以上 (*side* = 'L' の場合) または max(1, *n*) 以上 (*side* = 'R' の場合) でなければならない。
tau(i) には、sgerqf/dgerqf から返された通りに、基本リフレクタ H_i のスカラー係数が入っていなければならない。
tau のサイズは、max(1, *k*) 以上でなければならない。
*c(ldc, *)* には、*m* × *n* のマトリックス *C* を格納する。
c の第 2 次元のサイズは、max(1, *n*) 以上でなければならない。
work(lwork) は、ワークスペース配列。

lda INTEGER。*a* の第 1 次元のサイズ。*lda* ≥ max(1, *k*)

ldc INTEGER。*c* の第 1 次元のサイズ。*ldc* ≥ max(1, *m*)

lwork INTEGER。配列 *work* のサイズ。次の制約がある。
lwork ≥ max(1, *n*) (*side* = 'L' の場合)
lwork ≥ max(1, *m*) (*side* = 'R' の場合)
lwork の推奨値は、「アプリケーション・ノート」を参照。

出力パラメータ

c *side* と *trans* の指定に従って、QC、Q^TC、CQ、CQ^T のどれかの積で上書きされる。

work(1) *info* = 0 の場合は、最適パフォーマンスに必要な *lwork* の最小値が、ルーチンの終了時に *work(1)* に格納される。以降の実行では、この *lwork* 値を使用する。

info INTEGER
info = 0 の場合は、正常終了している。
info = -*i* の場合は、*i* 番目のパラメータ値が不正である。

アプリケーション・ノート

パフォーマンスを改善するには、`lwork`=`n`**blocksize* (`side` = `'L'` の場合) または `lwork`=`m`**blocksize* (`side` = `'R'` の場合) に設定する。*blocksize* は、**ブロック化アルゴリズム**の最適パフォーマンスに必要な値で、マシンによって異なる (通常は 16 ～ 64)。必要なワークスペース・サイズが不明な場合は、`lwork` に大きな値を設定して実行する。ルーチン終了後に `work(1)` の値を調べ、以降はその値を使用する。

このルーチンの複素数版は、?unmrq である。

?unmrq

?gerqf で求めた RQ 因子分解のユニタリ・マトリックス Q に、複素マトリックスを掛ける。

```
call cunmrq ( side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork,
              info )
call zunmrq ( side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork,
              info )
```

説明

このルーチンは、$m \times n$ の複素マトリックス C に、Q または Q^H を掛ける。Q は、ルーチン cgerqf/zgerqf から返された k 個の基本リフレクタの積で定義される複素ユニタリ・マトリックスである ($Q = H_1^H H_2^H \cdots H_k^H$)。

このルーチンは、パラメータ side と trans の値に従って、QC、Q^HC、CQ、CQ^H のどれかのマトリックス積を計算する (C は計算結果で上書きされる)。

入力パラメータ

side	CHARACTER*1。'L' または 'R' でなければならない。 side ='L' の場合、Q または Q^H が C の左側から掛けられる。 side ='R' の場合、Q または Q^H が C の右側から掛けられる。
trans	CHARACTER*1。'N' または 'C' でなければならない。 trans ='N' の場合、C に Q が掛けられる。 trans ='C' の場合、C に Q^H が掛けられる。
m	INTEGER。マトリックス C の行数 ($m \geq 0$)。
n	INTEGER。マトリックス C の列数 ($n \geq 0$)。
k	INTEGER。その積がマトリックス Q となる基本リフレクタの個数。次の制約がある。 $0 \leq k \leq m$ (side ='L' の場合) $0 \leq k \leq n$ (side ='R' の場合)

a, tau, c, work COMPLEX (cunmrq の場合)
DOUBLE COMPLEX (zunmrq の場合)
配列 : *a(lda, *)*, *tau(*)*, *c(ldc, *)*, *work(lwork)*
呼び出し時は、cgerqf/zgerqf で配列引数 *a* の最終 *k* 行に返された通りに、*a* の *i* 番目の行に、基本リフレクタ H_i (ただし、i = 1, 2, ..., *k*) を定義するベクトルが入っていなければならない。
a の第 2 次元のサイズは、max(1, *m*) 以上 (*side* ='L' の場合) または max(1, *n*) 以上 (*side* ='R' の場合) でなければならない。
tau(i) には、cgerqf/zgerqff から返された通りに、基本リフレクタ H_i のスカラー係数が入っていなければならない。
tau のサイズは、max(1, *k*) 以上でなければならない。
*c(ldc, *)* には、*m* × *n* のマトリックス *C* を格納する。
c の第 2 次元のサイズは、max(1, *n*) 以上でなければならない。
work(lwork) は、ワークスペース配列。

lda INTEGER。*a* の第 1 次元のサイズ。*lda* ≥ max(1, *k*)

ldc INTEGER。*c* の第 1 次元のサイズ。*ldc* ≥ max(1, *m*)

lwork INTEGER。配列 *work* のサイズ。次の制約がある。
lwork ≥ max(1, *n*) (*side* ='L' の場合)
lwork ≥ max(1, *m*) (*side* ='R' の場合)
lwork の推奨値は、「アプリケーション・ノート」を参照。

出力パラメータ

c *side* と *trans* の指定に従って、QC、Q^HC、CQ、CQ^H のどれかの積で上書きされる。

work(1) *info* = 0 の場合は、最適パフォーマンスに必要な *lwork* の最小値が、ルーチンの終了時に *work(1)* に格納される。以降の実行では、この *lwork* 値を使用する。

info INTEGER
info = 0 の場合は、正常終了している。
info = -*i* の場合は、*i* 番目のパラメータ値が不正である。

アプリケーション・ノート

パフォーマンスを改善するには、`lwork`=`n`**blocksize* (`side='L'` の場合) または `lwork`=`m`**blocksize* (`side='R'` の場合) に設定する。*blocksize* は、**ブロック化アルゴリズム**の最適パフォーマンスに必要な値で、マシンによって異なる (通常は 16 ～ 64)。必要なワークスペース・サイズが不明な場合は、`lwork` に大きな値を設定して実行する。ルーチン終了後に `work(1)` の値を調べ、以降はその値を使用する。

このルーチンの実数版は、?ormrq である。

?tzrzf

上台形マトリックス *A* を上三角形式に
縮退させる。

```
call stzrzf ( m, n, a, lda, tau, work, lwork, info )
call dtzrzf ( m, n, a, lda, tau, work, lwork, info )
call ctzrzf ( m, n, a, lda, tau, work, lwork, info )
call ztzrzf ( m, n, a, lda, tau, work, lwork, info )
```

説明

このルーチンは、$m \times n$ ($m \le n$) の実数 / 複素数の上台形マトリックス *A* を、直交 / ユニタリ変換によって上三角形式に縮退させる。上台形マトリックス *A* は、次のように因子分解される。

$$A = (R \;\; 0) * Z,$$

Z は $n \times n$ の直交 / ユニタリ・マトリックス、R は $m \times m$ の上三角マトリックスである。

入力パラメータ

m	INTEGER。マトリックス *A* の行数 ($m \ge 0$)。
n	INTEGER。マトリックス *A* の列数 ($n \ge m$)。
a, work	REAL (stzrzf の場合) DOUBLE PRECISION (dtzrzf の場合) COMPLEX (ctzrzf の場合) DOUBLE COMPLEX (ztzrzf の場合) 配列 : *a(lda, *), work(lwork)* 配列 *a* の $m \times n$ の主上台形部分に、因子分解するマトリックス *A* を格納する。 *a* の第 2 次元のサイズは、max(1, *n*) 以上でなければならない。 *work* は、ワークスペース配列。
lda	INTEGER。*a* の第 1 次元のサイズ。max(1, *m*) 以上。
lwork	INTEGER。配列 *work* のサイズ。

lwork ≥ max(1, *m*)
lwork の推奨値は、「アプリケーション・ノート」を参照。

出力パラメータ

a
終了時に、因子分解データによって次のように上書きされる。
a の *m* × *m* の主上三角部分に、上三角マトリックス *R* が格納される。また、*a* の最初の *m* 行における *m*+1 ～ *n* の成分と配列 *tau* によって、直交マトリックス *Z* が、*m* 個の基本リフレクタの積として表される。

tau
REAL (stzrzf の場合)
DOUBLE PRECISION (dtzrzf の場合)
COMPLEX (ctzrzf の場合)
DOUBLE COMPLEX (ztzrzf の場合)
配列 DIMENSION は max(1, *m*) 以上。
マトリックス *Z* に対する基本リフレクタのスカラー係数が入る。

work(1)
info = 0 の場合は、最適パフォーマンスに必要な *lwork* の最小値が、ルーチンの終了時に *work*(1) に格納される。以降の実行では、この *lwork* 値を使用する。

info
INTEGER
info = 0 の場合は、正常終了している。
info = -*i* の場合は、*i* 番目のパラメータ値が不正である。

アプリケーション・ノート

パフォーマンスを改善するには、*lwork*=*m***blocksize* に設定する。
blocksize は、**ブロック化アルゴリズム**の最適パフォーマンスに必要な値で、マシンによって異なる (通常は 16 ～ 64)。必要なワークスペース・サイズが不明な場合は、*lwork* に大きな値を設定して実行する。ルーチン終了後に *work*(1) の値を調べ、以降はその値を使用する。
関連ルーチン :

?ormrz　マトリックス Q を適用する (実マトリックスの場合)。
?unmrz　マトリックス Q を適用する (複素マトリックスの場合)。

?ormrz

?tzrzf による因子分解で求めた直交マトリックスに、実マトリックスを掛ける。

```
call sormrz ( side, trans, m, n, k, l, a, lda, tau, c, ldc, work,
              lwork, info )
call dormrz ( side, trans, m, n, k, l, a, lda, tau, c, ldc, work,
              lwork, info )
```

説明

このルーチンは、$m \times n$ の実マトリックス C に、Q または Q^T を掛ける。Q は、ルーチン stzrzf/dtzrzf から返された k 個の基本リフレクタ H_i の積で定義される直交実マトリックスである ($Q = H_1 H_2 \cdots H_k$)。

このルーチンは、パラメータ *side* と *trans* の値に従って、QC、Q^TC、CQ、CQ^T のどれかのマトリックス積を計算する (C は計算結果で上書きされる)。

マトリックス Q の次数は、*side* ='L' の場合は m、*side* ='R' の場合は n である。

入力パラメータ

side CHARACTER*1。'L' または 'R' でなければならない。
side ='L' の場合、Q または Q^T が C の左側から掛けられる。
side ='R' の場合、Q または Q^T が C の右側から掛けられる。

trans CHARACTER*1。'N' または 'T' でなければならない。
trans ='N' の場合、C に Q が掛けられる。
trans ='T' の場合、C に Q^T が掛けられる。

m INTEGER。マトリックス C の行数 ($m \geq 0$)。

n INTEGER。マトリックス C の列数 ($n \geq 0$)。

k INTEGER。その積がマトリックス Q となる基本リフレクタの個数。次の制約がある。
$0 \leq k \leq m$ (*side* ='L' の場合)
$0 \leq k \leq n$ (*side* ='R' の場合)

`l` INTEGER
マトリックス A において、Householder リフレクタとして意味を持つ列数。次の制約がある。
$0 \le l \le m$ (`side='L'` の場合)
$0 \le l \le n$ (`side='R'` の場合)

`a, tau, c, work` REAL (`sormrz` の場合)
DOUBLE PRECISION (`dormrz` の場合)
配列 : `a(lda, *)`, `tau(*)`, `c(ldc, *)`, `work(lwork)`
呼び出し時は、`stzrzf`/`dtzrzf` で配列引数 `a` の最終 `k` 行に返された通りに、`a` の `i` 番目の行に、基本リフレクタ H_i (ただし、i = 1, 2, ..., `k`) を定義するベクトルが入っていなければならない。
`a` の第 2 次元のサイズは、max(1, `m`) 以上 (`side='L'` の場合) または max(1, `n`) 以上 (`side='R'` の場合) でなければならない。
`tau`(i) には、`stzrzf`/`dtzrzff` から返された通りに、基本リフレクタ H_i のスカラー係数が入っていなければならない。
`tau` のサイズは、max(1, `k`) 以上でなければならない。
`c(ldc, *)` には、$m \times n$ のマトリックス C を格納する。
`c` の第 2 次元のサイズは、max(1, `n`) 以上でなければならない。
`work(lwork)` は、ワークスペース配列。

`lda` INTEGER。`a` の第 1 次元のサイズ。`lda` ≥ max(1, `k`)

`ldc` INTEGER。`c` の第 1 次元のサイズ。`ldc` ≥ max(1, `m`)

`lwork` INTEGER。配列 `work` のサイズ。次の制約がある。
`lwork` ≥ max(1, `n`) (`side='L'` の場合)
`lwork` ≥ max(1, `m`) (`side='R'` の場合)
`lwork` の推奨値は、「アプリケーション・ノート」を参照。

出力パラメータ

`c` `side` と `trans` の指定に従って、QC、Q^TC、CQ、CQ^T のどれかの積で上書きされる。

`work(1)` `info` = 0 の場合は、最適パフォーマンスに必要な `lwork` の最小値が、ルーチンの終了時に `work(1)` に格納される。以降の実行では、この `lwork` 値を使用する。

info INTEGER
info = 0 の場合は、正常終了している。
info = -*i* の場合は、*i* 番目のパラメータ値が不正である。

アプリケーション・ノート

パフォーマンスを改善するには、*lwork*=*n***blocksize* (*side* = 'L' の場合) または *lwork*=*m***blocksize* (*side* = 'R' の場合) に設定する。*blocksize* は、**ブロック化アルゴリズム**の最適パフォーマンスに必要な値で、マシンによって異なる (通常は 16 ～ 64)。必要なワークスペース・サイズが不明な場合は、*lwork* に大きな値を設定して実行する。ルーチン終了後に *work*(1) の値を調べ、以降はその値を使用する。

このルーチンの複素数版は、?unmrz である。

?unmrz

?tzrzf による因子分解で求めたユニタリ・マトリックス Q に、複素マトリックスを掛ける。

```
call cunmrz ( side, trans, m, n, k, l, a, lda, tau, c, ldc, work,
              lwork, info )
call zunmrz ( side, trans, m, n, k, l, a, lda, tau, c, ldc, work,
              lwork, info )
```

説明

このルーチンは、$m \times n$ の複素マトリックス C に、Q または Q^H を掛ける。Q は、ルーチン ctzrzf/ztzrzf から返された k 個の基本リフレクタ H_i の積で定義されるユニタリ・マトリックスである ($Q = H_1^H H_2^H \cdots H_k^H$)。

このルーチンは、パラメータ *side* と *trans* の値に従って、QC、Q^HC、CQ、CQ^H のどれかのマトリックス積を計算する (C は計算結果で上書きされる)。

マトリックス Q の次数は、*side* ='L' の場合は m、*side* ='R' の場合は n である。

入力パラメータ

side CHARACTER*1。'L' または 'R' でなければならない。
side ='L' の場合、Q または Q^H が C の左側から掛けられる。
side ='R' の場合、Q または Q^H が C の右側から掛けられる。

trans CHARACTER*1。'N' または 'C' でなければならない。
trans ='N' の場合、C に Q が掛けられる。
trans ='C' の場合、C に Q^H が掛けられる。

m INTEGER。マトリックス C の行数 ($m \geq 0$)。

n INTEGER。マトリックス C の列数 ($n \geq 0$)。

k INTEGER。その積がマトリックス Q となる基本リフレクタの個数。次の制約がある。
$0 \leq k \leq m$ (*side* ='L' の場合)
$0 \leq k \leq n$ (*side* ='R' の場合)

l INTEGER
マトリックス A において、Householder リフレクタとして意味を持つ列数。次の制約がある。
$0 \le l \le m$ (*side* = 'L' の場合)
$0 \le l \le n$ (*side* = 'R' の場合)

a, tau, c, work COMPLEX (cunmrz の場合)
DOUBLE COMPLEX (zunmrz の場合)
配列 : *a(lda, *)*, *tau(*)*, *c(ldc, *)*, *work(lwork)*
呼び出し時は、ctzrzf/ztzrzf で配列引数 *a* の最終 *k* 行に返された通りに、*a* の *i* 番目の行に、基本リフレクタ H_i (ただし、i = 1, 2, ..., *k*) を定義するベクトルが入っていなければならない。
a の第 2 次元のサイズは、max(1, *m*) 以上 (*side* = 'L' の場合) または max(1, *n*) 以上 (*side* = 'R' の場合) でなければならない。
tau(i) には、ctzrzf/ztzrzf から返された通りに、基本リフレクタ H_i のスカラー係数が入っていなければならない。
tau のサイズは、max(1, *k*) 以上でなければならない。
*c(ldc, *)* には、$m \times n$ のマトリックス C を格納する。
c の第 2 次元のサイズは、max(1, *n*) 以上でなければならない。
work(lwork) は、ワークスペース配列。

lda INTEGER。*a* の第 1 次元のサイズ。*lda* ≥ max(1, *k*)

ldc INTEGER。*c* の第 1 次元のサイズ。*ldc* ≥ max(1, *m*)

lwork INTEGER。配列 *work* のサイズ。次の制約がある。
lwork ≥ max(1, *n*) (*side* = 'L' の場合)
lwork ≥ max(1, *m*) (*side* = 'R' の場合)
lwork の推奨値は、「アプリケーション・ノート」を参照。

出力パラメータ

c *side* と *trans* の指定に従って、QC、Q^HC、CQ、CQ^H のどれかの積で上書きされる。

work(1) *info* = 0 の場合は、最適パフォーマンスに必要な *lwork* の最小値が、ルーチンの終了時に *work(1)* に格納される。以降の実行では、この *lwork* 値を使用する。

info INTEGER

info = 0 の場合は、正常終了している。

info = -*i* の場合は、*i* 番目のパラメータ値が不正である。

アプリケーション・ノート

パフォーマンスを改善するには、*lwork* = *n***blocksize* (*side* = 'L' の場合) または *lwork* = *m***blocksize* (*side* = 'R' の場合) に設定する。*blocksize* は、**ブロック化アルゴリズム**の最適パフォーマンスに必要な値で、マシンによって異なる (通常は 16 ～ 64)。必要なワークスペース・サイズが不明な場合は、*lwork* に大きな値を設定して実行する。ルーチン終了後に *work*(1) の値を調べ、以降はその値を使用する。

このルーチンの実数版は、?ormrz である。

?ggqrf

2 つのマトリックスに対して汎用 *QR* 因子分解を実行する。

```
call sggqrf (n, m, p, a, lda, taua, b, ldb, taub, work, lwork, info)
call dggqrf (n, m, p, a, lda, taua, b, ldb, taub, work, lwork, info)
call cggqrf (n, m, p, a, lda, taua, b, ldb, taub, work, lwork, info)
call zggqrf (n, m, p, a, lda, taua, b, ldb, taub, work, lwork, info)
```

説明

このルーチンは、$n \times m$ のマトリックス A と $n \times p$ のマトリックス B に対して、$A = Q\,R, B = Q\,T\,Z$ で表される汎用 *QR* 因子分解を実行する。Q は $n \times n$ の直交 / ユニタリ・マトリックス、Z は $p \times p$ の直交 / ユニタリ・マトリックスである。また、R と T は次のどちらかの形式である。

$$R = \begin{array}{cc} & m \\ m & \\ n-m & \end{array} \begin{pmatrix} R_{11} \\ 0 \end{pmatrix} \text{、}(n \geq m \text{ の場合})$$

または

$$R = \begin{array}{cc} & \begin{array}{cc} n & m-n \end{array} \\ n & \end{array} \begin{pmatrix} R_{11} & R_{12} \end{pmatrix} \text{、}(n < m \text{ の場合})$$

R_{11} は上三角マトリックスである。

$$T = \begin{array}{cc} & \begin{array}{cc} p-n & n \end{array} \\ n & \end{array} \begin{pmatrix} 0 & T_{12} \end{pmatrix} \text{、}(n \leq p \text{ の場合})\text{、または}$$

$$T = \begin{array}{cc} & p \\ n-p & \\ p & \end{array} \begin{pmatrix} T_{11} \\ T_{21} \end{pmatrix} \text{、}(n > p \text{ の場合})$$

T_{12} または T_{21} は $p \times p$ の上三角マトリックスである。

特に、マトリックス B が正方かつ非特異の場合は、A と B の GQR 因子分解によって、$B^{-1}A$ の QR 因子分解が次のように得られる。

$$B^{-1}A = Z^H(T^{-1}R)$$

入力パラメータ

n	INTEGER。マトリックス A と B の行数 ($n \geq 0$)。
m	INTEGER。マトリックス A の列数 ($m \geq 0$)。
p	INTEGER。マトリックス B の列数 ($p \geq 0$)。
a, b, work	REAL (sggqrf の場合) DOUBLE PRECISION (dggqrf の場合) COMPLEX (cggqrf の場合) DOUBLE COMPLEX (zggqrf の場合) 配列 : *a(lda, *)* には、マトリックス A を格納する。 *a* の第 2 次元のサイズは、max(1, *m*) 以上でなければならない。 *b(ldb, *)* には、マトリックス B を格納する。 *b* の第 2 次元のサイズは、max(1, *p*) 以上でなければならない。 *work(lwork)* は、ワークスペース配列。
lda	INTEGER。*a* の第 1 次元のサイズ。max(1, *n*) 以上。
ldb	INTEGER。*b* の第 1 次元のサイズ。max(1, *n*) 以上。
lwork	INTEGER。配列 *work* のサイズ。max(1, *n*, *m*, *p*) 以上でなければならない。 *lwork* の推奨値は、「アプリケーション・ノート」を参照。

出力パラメータ

a, b	因子分解データによって次のように上書きされる。 終了時に、配列 *a* の対角成分とその上の各成分に、min(*n*, *m*) × *m* の上台形マトリックス R が格納される ($n \geq m$ ならば、R は上三角マトリックスになる)。対角成分より下の各成分と配列 *taua* によって、直交 / ユニタリ・マトリックス Q が、min(*n*, *m*) 個の基本リフレクタの積として表される。

$n \le p$ の場合は、部分配列 $b(1:n, p-n+1:p)$ の上三角部分に、$n \times n$ の上三角マトリックス T が格納される。
$n>p$ の場合は、$(n-p)$ 番目の劣対角成分とその上の各成分に、$n \times p$ の上台形マトリックス T が格納される。残りの各成分と配列 *taub* によって、直交 / ユニタリ・マトリックス Z が、基本リフレクタの積として表される。

taua, taub

REAL (sggqrf の場合)
DOUBLE PRECISION (dggqrf の場合)
COMPLEX (cggqrf の場合)
DOUBLE COMPLEX (zggqrf の場合)
配列 DIMENSION は、*taua* は max(1, min(n, m)) 以上、*taub* は max(1, min(n, p)) 以上。
配列 *taua* には、直交 / ユニタリ・マトリックス Q を表す基本リフレクタのスカラー係数が格納される。
配列 *taub* には、直交 / ユニタリ・マトリックス Z を表す基本リフレクタのスカラー係数が格納される。

work(1)

info = 0 の場合は、最適パフォーマンスに必要な *lwork* の最小値が、ルーチンの終了時に *work(1)* に格納される。以降の実行では、この *lwork* 値を使用する。

info

INTEGER
info = 0 の場合は、正常終了している。
info = -*i* の場合は、*i* 番目のパラメータ値が不正である。

アプリケーション・ノート

パフォーマンスを改善するには、lwork を次のように設定する。
lwork ≥ max(n, m, p)*max(*nb1, nb2, nb3*)
nb1 は $n \times m$ のマトリックスの *QR* 因子分解における最適 blocksize、*nb2* は $n \times p$ のマトリックスの *RQ* 因子分解における最適 blocksize、*nb3* は ?ormqr/?unmqr を呼び出すときの最適 blocksize である。

?ggrqf

2 つのマトリックスに対して汎用 *RQ* 因子分解を実行する。

```
call sggrqf (m, p, n, a, lda, taua, b, ldb, taub, work, lwork, info)
call dggrqf (m, p, n, a, lda, taua, b, ldb, taub, work, lwork, info)
call cggrqf (m, p, n, a, lda, taua, b, ldb, taub, work, lwork, info)
call zggrqf (m, p, n, a, lda, taua, b, ldb, taub, work, lwork, info)
```

説明

このルーチンは、$m \times n$ のマトリックス A と $p \times n$ のマトリックス B に対して、$A = R\,Q, B = Z\,T\,Q$ で表される汎用 *RQ* 因子分解を実行する。Q は $n \times n$ の直交 / ユニタリ・マトリックス、Z は $p \times p$ の直交 / ユニタリ・マトリックスである。また、R と T は次のどちらかの形式である。

$$R = \begin{array}{c} \\ m \end{array} \begin{array}{c} \begin{array}{cc} n-m & m \end{array} \\ \begin{pmatrix} 0 & R_{12} \end{pmatrix} \end{array} \text{、}(m \le n \text{ の場合})$$

または

$$R = \begin{array}{c} \\ m-n \\ n \end{array} \begin{array}{c} n \\ \begin{pmatrix} R_{11} \\ R_{21} \end{pmatrix} \end{array} \text{、}(m > n \text{ の場合})$$

R_{11} と R_{21} は上三角マトリックスである。

$$T = \begin{array}{c} \\ n \\ p-n \end{array} \begin{array}{c} n \\ \begin{pmatrix} T_{11} \\ 0 \end{pmatrix} \end{array} \text{、}(p \ge n \text{ の場合})$$

または

$$T = \begin{array}{c} \\ p \end{array} \begin{array}{c} \begin{array}{cc} p & n-p \end{array} \\ \begin{pmatrix} T_{11} & T_{12} \end{pmatrix} \end{array} \text{、}(p < n \text{ の場合})$$

ここで、T_{11} は上三角マトリックスである。

特に、マトリックス B が正方かつ非特異の場合は、A と B の GRQ 因子分解によって、AB^{-1} の RQ 因子分解が次のように得られる。

$$AB^{-1} = (R\ \ T^{-1})\,Z^H$$

入力パラメータ

m	INTEGER。マトリックス A の行数 ($m \geq 0$)。
p	INTEGER。マトリックス B の行数 ($p \geq 0$)。
n	INTEGER。マトリックス A と B の列数 ($n \geq 0$)。
a, b, work	REAL (sggrqf の場合) DOUBLE PRECISION (dggrqf の場合) COMPLEX (cggrqf の場合) DOUBLE COMPLEX (zggrqf の場合) 配列 : *a(lda, *)* には、$m \times n$ のマトリックス A を格納する。*a* の第 2 次元のサイズは、max(1, *n*) 以上でなければならない。 *b(ldb, *)* には、$p \times n$ のマトリックス B を格納する。*b* の第 2 次元のサイズは、max(1, *n*) 以上でなければならない。 *work(lwork)* は、ワークスペース配列。
lda	INTEGER。*a* の第 1 次元のサイズ。max(1, *m*) 以上。
ldb	INTEGER。*b* の第 1 次元のサイズ。max(1, *p*) 以上。
lwork	INTEGER。配列 *work* のサイズ。max(1, *n*, *m*, *p*) 以上でなければならない。 *lwork* の推奨値は、「アプリケーション・ノート」を参照。

出力パラメータ

a, b	因子分解データによって次のように上書きされる。 終了時に、$m \leq n$ の場合は、部分配列 *a*(1:*m*, *n*-*m*+1:*n*) の上三角部分に、$m \times m$ の上三角マトリックス R が格納される。 $m > n$ の場合は、(*m*-*n*) 番目の劣対角成分とその上の各成分に、$m \times n$ の上台形マトリックス R が格納される。残りの各成分と配列 *taua* によって、直交 / ユニタリ・マトリックス Q が、基本リフレクタの積として表される。

配列 *b* の対角成分とその上の各成分に、min(*p*, *n*) × *n* の上台形マトリックス *T* が格納される (*p* ≥ *n* ならば、*T* は上三角マトリックスになる)。対角成分より下の各成分と配列 *taub* によって、直交 / ユニタリ・マトリックス *Z* が、基本リフレクタの積として表される。

taua, taub

REAL (sggrqf の場合)
DOUBLE PRECISION (dggrqf の場合)
COMPLEX (cggrqf の場合)
DOUBLE COMPLEX (zggrqf の場合)
配列 DIMENSION は、*taua* は max(1, min(*m*, *n*)) 以上、*taub* は max(1, min(*p*, *n*)) 以上。
配列 *taua* には、直交 / ユニタリ・マトリックス *Q* を表す基本リフレクタのスカラー係数が格納される。
配列 *taub* には、直交 / ユニタリ・マトリックス *Z* を表す基本リフレクタのスカラー係数が格納される。

work(1)

info = 0 の場合は、最適パフォーマンスに必要な *lwork* の最小値が、ルーチンの終了時に *work(1)* に格納される。以降の実行では、この *lwork* 値を使用する。

info

INTEGER
info = 0 の場合は、正常終了している。
info = -*i* の場合は、*i* 番目のパラメータ値が不正である。

アプリケーション・ノート

パフォーマンスを改善するには、*lwork* を次のように設定する。
lwork ≥ max(*n*, *m*, *p*)*max(*nb1, nb2, nb3*)

nb1 は *m* × *n* ののマトリックスの *RQ* 因子分解における最適 blocksize、*nb2* は *p* × *n* のマトリックスの *QR* 因子分解における最適 blocksize、*nb3* は ?ormrq/?unmrq を呼び出すときの最適 blocksize である。
必要なワークスペース・サイズが不明な場合は、*lwork* に大きな値を設定して実行する。ルーチン終了後に *work(1)* の値を調べ、以降はその値を使用する。

特異値分解

本節では、一般マトリックス A $(m \times n)$ を次のように**特異値分解** (SVD) するための LAPACK ルーチンについて説明する。

$$A = U\Sigma V^H$$

この分解処理では、U と V は、ユニタリ・マトリックス (A が複素マトリックスの場合) または直交マトリックス (A が実マトリックスの場合) である。Σ は、次に示す対角成分 σ_i を持つ $m \times n$ の対角マトリックスである。

$$\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_{\min(m,\,n)} \geq 0$$

対角成分 σ_i は、A の**特異値**である。マトリックス U と V の先頭から $\min(m, n)$ の列は、それぞれ、A の**左特異ベクトル**と**右特異ベクトル**である。対応する特異値と特異ベクトルは、次の条件を満たす。

$$Av_i = \sigma_i u_i \text{ かつ } A^H u_i = \sigma_i v_i$$

ui と vi は、それぞれ、U と V の i 番目の成分である。

一般マトリックス A の SVD を求めるには、まず、LAPACK ルーチンの `?gebrd` または `?gbbrd` を呼び出し、ユニタリ (直交) 変換 $A = QBP^H$ によって A を二重対角マトリックス B に縮退させる。次に `?bdsqr` を呼び出し、二重対角マトリックス B から $B = U_1 \Sigma V_1^H$ となるような SVD を求める。

そのため、求めるべき A の SVD は、$A = U\Sigma V^H = (QU_1)\,\Sigma\,(V_1^H P^H)$ のように表現できる。

表 5-2 特異値分解 (SVD) 用のルーチン

処理	実 マトリックス	複素 マトリックス
A を次の二重対角マトリックス B に縮退させる $A = QBP^H$ (フル格納)	`?gebrd`	`?gebrd`
A を次の二重対角マトリックス B に縮退させる $A = QBP^H$ (バンド格納)	`?gbbrd`	`?gbbrd`
直交 (ユニタリ) マトリックス Q または P を生成する	`?orgbr`	`?ungbr`
直交 (ユニタリ) マトリックス Q または P を適用する	`?ormbr`	`?unmbr`
二重対角マトリックス B から次の特異値分解を行う $B = U\Sigma V^H$	`?bdsqr` `?bdsdc`	`?bdsqr`

図 5-1　デシジョン・ツリー : 特異値分解

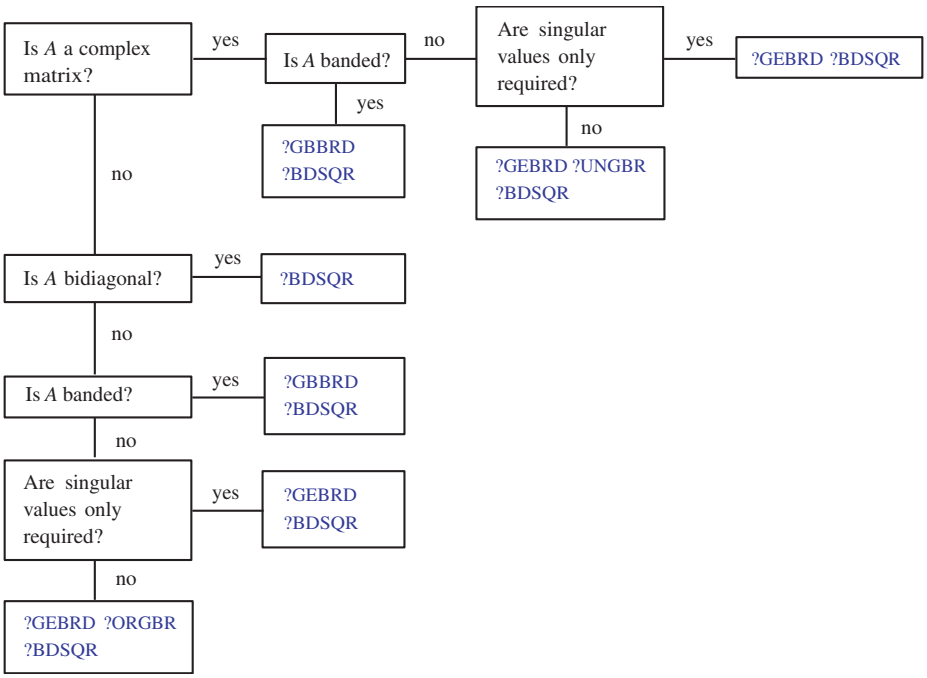


図 5-1 のデシジョン・ツリーを参考にすれば、特異値だけ必要な場合、特異値と特異ベクトルの両方が必要な場合、あるいは A が実数または複素数のいずれであるかなどといった条件に応じて、SVD を求めるための一連のルーチンを正しく選択できる。

SVD を使用すれば、$\|Ax - b\|_2$ が最小になるような階数不足の最小二乗問題に対する最小ノルム解を得られる (可能な場合)。マトリックス A の有効な階数 k は、適切なしきい値を超えるような特異値の個数によって決定できる。最小ノルム解は、次のとおりである。

$$x = V_k(\Sigma_k)^{-1}c$$

Σ_k は、Σ の先頭から $k \times k$ の部分マトリックスである。マトリックス V_k は $V = PV_1$ の先頭から k 個の列で構成され、ベクトル c は $U^H b = U_1{}^H Q^H b$ の先頭から k 個の成分で構成される。

?gebrd

一般マトリックスを二重対角形式に
縮退させる。

```
call sgebrd ( m, n, a, lda, d, e, tauq, taup, work, lwork, info )
call dgebrd ( m, n, a, lda, d, e, tauq, taup, work, lwork, info )
call cgebrd ( m, n, a, lda, d, e, tauq, taup, work, lwork, info )
call zgebrd ( m, n, a, lda, d, e, tauq, taup, work, lwork, info )
```

説明

このルーチンは、直交 (ユニタリ) 変換によって、$m \times n$ の一般マトリックス A を二重対角マトリックス B に縮退させる。

$m \geq n$ の場合、縮退は次の式で与えられる。

$$A = QBP^H = Q\begin{pmatrix} B_1 \\ 0 \end{pmatrix}P^H = Q_1 B_1 P^H,$$

B_1 は、$n \times n$ の上三角マトリックスである。Q と P は、直交マトリックスまたはユニタリ・マトリックス (A が複素マトリックスの場合) である。Q_1 は、Q の先頭から n 個の列によって構成される。

$m < n$ の場合、縮退は次の式で与えられる。

$$A = QBP^H = Q(B_1 0)P^H = Q_1 B_1 P_1^H,$$

B_1 は、$m \times m$ の下対角マトリックスである。Q と P は、直交マトリックスまたはユニタリ・マトリックス (A が複素マトリックスの場合) である。P_1 は、P の先頭から m 個の行によって構成される。

このルーチンでは、マトリックス Q と P を明示的な形式では表現せず、基本リフレクタの積として表現する。一連のルーチンは、この形式で表現されるマトリックス Q と P を操作するためのものである。

マトリックス A が実数型の場合、

- Q と P を明示的に求めるには、?orgbr を呼び出す。
- 一般マトリックスと Q または P とを掛け合わせるには、?ormbr を呼び出す。

マトリックス A が複素数型の場合、

- Q と P を明示的に求めるには、?ungbr を呼び出す。
- 一般マトリックスと Q または P とを掛け合わせるには、?unmbr を呼び出す。

入力パラメータ

m INTEGER。マトリックス *A* の行数 ($m \geq 0$)。

n INTEGER。マトリックス *A* の列数 ($n \geq 0$)。

a, work REAL (sgebrd の場合)
DOUBLE PRECISION (dgebrd の場合)
COMPLEX (cgebrd の場合)
DOUBLE COMPLEX (zgebrd の場合)
配列 :
*a(lda, *)* には、マトリックス *A* を格納する。
a の第 2 次元は、max(1, *n*) 以上でなければならない。
work(lwork) は、ワークスペース配列である。

lda INTEGER。*a* の第 1 次元。max(1, *m*) 以上であること。

lwork INTEGER。*work* の次元。max(1, *m*, *n*) 以上。
lwork の推奨値は、「アプリケーション・ノート」を参照。

出力パラメータ

a $m \geq n$ の場合、*a* の対角線と第 1 の優対角成分が、上二重対角マトリックス *B* によって上書きされる。対角線より下の成分は *Q* の各成分によって、残りの成分は *P* の各成分によって上書きされる。
$m < n$ の場合、*a* の対角線と第 1 の劣対角成分が、下二重対角マトリックス *B* によって上書きされる。対角線より上の成分は *P* の各成分によって、残りの成分は *Q* の各成分によって上書きされる。

d REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
配列 DIMENSION は、max(1, min(*m*, *n*)) 以上。
B の対角成分が格納される。

e REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
配列 DIMENSION は、max(1, min(*m*, *n*) – 1) 以上。
B の対角線外の成分が格納される。

tauq, taup	REAL (sgebrd の場合) DOUBLE PRECISION (dgebrd の場合) COMPLEX (cgebrd の場合) DOUBLE COMPLEX (zgebrd の場合) 配列 DIMENSION は、(1, min(m, n)) 以上。 マトリックス Q と P に関するさらなる各成分が格納される。
work(1)	*info* = 0 の場合は、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の *lwork* の値が *work(1)* に格納される。これ以降の実行には、この *lwork* の値を使用する。
info	INTEGER *info* = 0 の場合は、正常に終了する。 *info* = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

パフォーマンスを改善するには、*lwork* = (m + n)**blocksize* に設定する。*blocksize* は、**ブロック化アルゴリズム**の最適パフォーマンスに必要な値で、マシンによって異なる (通常は 16 ～ 64)。

用意すべきワークスペースの量が不明な場合は、最初は *lwork* に大きな値を設定する。終了時に *work(1)* の値を調べ、これ以降はその値を使用する。

計算で求めたマトリックス Q、B、P は、$QBP^H = A + E$ を満たす。$\|E\|_2 = c(n)\varepsilon \|A\|_2$ であり、$c(n)$ は漸増する n の関数で、ε はマシンによって異なる値である。

実数の場合の浮動小数演算のおおよその総数は、次のとおりである。
$(4/3)*n^2*(3*m - n)$ ($m \geq n$ の場合)
$(4/3)*m^2*(3*n - m)$ ($m < n$ の場合)
複素数の場合は、この 4 倍になる。

n が m に比べてはるかに小さい場合は、まず ?geqrf を呼び出して A の QR 因子分解を求めた後、因数 R を二重対角形式に縮退した方が効率が良くなる。その場合は、約 $2*n^2*(m + n)$ 回の浮動小数演算が必要になる。

一方、m が n に比べてはるかに小さい場合は、まず ?gelqf を呼び出して A の LQ 因子分解を求めた後、因数 L を二重対角形式に縮退した方が効率が良いかもしれない。その場合は、約 $2*m^2*(m + n)$ 回の浮動小数演算が必要になる。

?gbbrd

一般バンド・マトリックスを
二重対角形式に縮退させる。

```
call sgbbrd ( vect, m, n, ncc, kl, ku, ab, ldab, d, e, q, ldq, pt,
              ldpt, c, ldc, work, info )
call dgbbrd ( vect, m, n, ncc, kl, ku, ab, ldab, d, e, q, ldq, pt,
              ldpt, c, ldc, work, info )
call cgbbrd ( vect, m, n, ncc, kl, ku, ab, ldab, d, e, q, ldq, pt,
              ldpt, c, ldc, work, rwork, info )
call zgbbrd ( vect, m, n, ncc, kl, ku, ab, ldab, d, e, q, ldq, pt,
              ldpt, c, ldc, work, rwork, info )
```

説明

このルーチンは、$m \times n$ のバンド・マトリックス A を $A = QBP^H$ に縮退させ、上二重対角マトリックス B を求める。マトリックス Q と P は、直交マトリックス (A が実数型の場合) またはユニタリ・マトリックス (A が複素数型の場合) である。これらは、Givens 回転マトリックスの積として得られる。そのため、必要に応じて、このルーチンを使って、これを明示的に求められる。このルーチンでは、$C = Q^H C$ としてマトリックス C の更新もできる。

入力パラメータ

vect CHARACTER*1。'N'、'Q'、'P'、'B' のいずれかでなければならない。
vect = 'N' の場合は、Q も P^H も生成されない。
vect = 'Q' の場合は、マトリックス Q が生成される。
vect = 'P' の場合は、マトリックス P^H が生成される。
vect = 'B' の場合は、Q も P^H も生成される。

m INTEGER。マトリックス A の行数 ($m \geq 0$)。

n INTEGER。マトリックス A の列数 ($n \geq 0$)。

ncc INTEGER。マトリックス C の列数 ($ncc \geq 0$)。

kl INTEGER。A ($kl \geq 0$) のバンド内の劣対角成分の個数。

ku INTEGER。A ($ku \geq 0$) のバンド内の優対角成分の個数。

ab, c, work
REAL (sgbbrd の場合)
DOUBLE PRECISION (dgbbrd の場合)
COMPLEX (cgbbrd の場合)
DOUBLE COMPLEX (zgbbrd の場合)
配列 :
*ab(ldab, *)* には、バンド格納形式のマトリックス *A* (「マトリックスの格納形式」を参照) を格納する。
a の第 2 次元は、max(1, *n*) 以上でなければならない。
*c(ldc, *)* には、*m* × *ncc* のマトリックス *C* を格納する。
ncc = 0 の場合、配列 *c* は参照されない。*c* の第 2 次元は、max(1, *ncc*) 以上でなければならない。
work(*) は、ワークスペース配列である。
work の次元は、実数型の場合は 2*max(*m*, *n*) 以上、複素数型の場合は max(*m*, *n*) 以上でなければならない。

ldab
INTEGER。配列 *ab* の第 1 次元。(*ldab* ≥ *kl* + *ku* + 1)

ldq
INTEGER。出力配列 *q* の第 1 次元。
(*vect* = 'Q' または 'B' の場合は、*ldq* ≥ max(1, *m*)
そうでない場合は、*ldq* ≥ 1

ldpt
INTEGER。出力配列 *pt* の第 1 次元。
ldpt ≥ max(1, *n*) *vect* = 'P' または 'B' の場合は、
そうでない場合は、*ldpt* ≥ 1

ldc
INTEGER。配列 *c* の第 1 次元。
(*ncc* > 0 の場合は *ldc* ≥ max(1, *m*) で、*ncc* = 0 の場合は *ldc* ≥ 1)

rwork
REAL (cgbbrd の場合)
DOUBLE PRECISION (zgbbrd の場合)
ワークスペース配列 DIMENSION は、max(*m*, *n*) 以上。

出力パラメータ

ab
退演算時に生成された値によって上書きされる。

d
REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
配列 DIMENSION は、max(1, min(*m*, *n*)) 以上。
マトリックス *B* の対角成分が格納される。

e	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列 DIMENSION は、max(1, min(*m*, *n*) - 1) 以上。 *B* の対角線外の成分が格納される。
q, *pt*	REAL (sgebrd の場合) DOUBLE PRECISION (dgebrd の場合) COMPLEX (cgebrd の場合) DOUBLE COMPLEX (zgebrd の場合) 配列 : *q*(*ldq*, *) には、*m* × *m* の出力マトリックス *Q* が格納される。 *q* の第 2 次元は、max(1, *m*) 以上でなければならない。 *p*(*ldpt*, *) には、*n* × *n* の出力マトリックス P^H が格納される。 *pt* の第 2 次元は、max(1, *n*) 以上でなければならない。
info	INTEGER *info* = 0 の場合は、正常に終了する。 *info* = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

計算で求めたマトリックス *Q*、*B*、*P* は、$QBP^H = A + E$ を満たす。
$\|E\|_2 = c(n)\varepsilon \|A\|_2$ であり、$c(n)$ は漸増する n の関数で、ε はマシンによって異なる値である。

m = *n* の場合、実数の場合の浮動小数演算のおおよその総数は、次の合計になる。

$6*n^2*(kl + ku)$	(*vect* = 'N' かつ *ncc* = 0 の場合)
$3*n^2*ncc*(kl + ku - 1)/(kl + ku)$	(*C* を更新する場合)
$3*n^3*(kl + ku - 1)/(kl + ku)$	(*Q* または P^H を生成する場合) (両方とも生成する場合は、2 倍の値になる。)

複素数の場合の演算回数を見積もる場合は、同じ式で係数を (6 と 3 ではなく) 20 と 10 とする。

?orgbr

?gebrd で求めた実数型の直交
マトリックス Q または P^T を生成する。

```
call sorgbr ( vect, m, n, k, a, lda, tau, work, lwork, info )
call dorgbr ( vect, m, n, k, a, lda, tau, work, lwork, info )
```

説明

このルーチンは、ルーチン sgebrd/dgebrd (5-76 ページを参照) で求めた直交マトリックス Q と P^T の全部または一部を生成する。このルーチンは、ルーチン sgebrd/dgebrd を呼び出した後で使用する。引数の有効な組み合わせについては、**入力パラメータ**を参照。ほとんどの場合は、次のように呼び出す必要がある。

$m \times m$ のマトリックス Q の全体を求めるには、
```
call ?orgbr ( 'Q', m, m, n, a ... )
```
(ただし、配列 *a* には、列が *m* 個以上あること)

Q の先頭から *n* 列を求めるには ($m > n$ の場合)、
```
call ?orgbr ( 'Q', m, n, n, a ... )
```

$n \times n$ のマトリックス P^T の全体を求めるには、
```
call ?orgbr ( 'P', n, n, m, a ... )
```
(ただし、配列 *a* には、行が *m* 個以上あること)

P^T の先頭から m 行を求めるには ($m < n$ の場合)、
```
call ?orgbr ( 'P', m, n, m, a ... )
```

入力パラメータ

vect	CHARACTER*1。'Q' または 'P' でなければならない。 *vect* = 'Q' の場合は、マトリックス Q が生成される。 *vect* = 'P' の場合は、マトリックス P^T が生成される。
m	INTEGER。Q または P^T に必要な行数。
n	INTEGER。Q または P^T に必要な列数。
k	INTEGER。?gebrd で得られた A の次元のいずれか。 *vect* = 'Q' の場合は、A の列数。 *vect* = 'P' の場合は、A の行数。

次の制約がある。$m \geq 0$, $n \geq 0$, $k \geq 0$
vect = 'Q' の場合は、$k \leq n \leq m$ ($m > k$ のとき) または $m = n$ ($m \leq k$ のとき)
vect = 'P' の場合は、$k \leq m \leq n$ ($n > k$ のとき) または $m = n$ ($n \leq k$ のとき)

a, work REAL (sorgbr の場合)
DOUBLE PRECISION (dorgbr の場合)
配列 :
*a(lda, *)* は、?gebrd から返された配列 *a* である。
a の第 2 次元は、max(1, *n*) 以上でなければならない。
work(lwork) は、ワークスペース配列である。

lda INTEGER。*a* の第 1 次元。max(1, *m*) 以上であること。

tau REAL (sorgbr の場合)
DOUBLE PRECISION (dorgbr の場合)
vect = 'Q' の場合、*tauq* は ?gebrd から返された配列。
vect = 'P' の場合、*taup* は ?gebrd から返された配列。
tau の次元は、max(1, min(*m*, *k*)) 以上 (*vect* = 'Q' の場合)、または max(1, min(*m*, *k*)) 以上 (*vect* = 'P' の場合) でなければならない。

lwork INTEGER。配列 *work* のサイズ。
lwork の推奨値は、「アプリケーション・ノート」を参照。

出力パラメータ

a *vect*、*m*、*n* の指定に従って、直交マトリックス Q または P^T (あるいはそれに対応する先頭からの一連の行または列) によって上書きされる。

work(1) *info* = 0 の場合は、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の *lwork* の値が *work(1)* に格納される。これ以降の実行には、この *lwork* の値を使用する。

info INTEGER
info = 0 の場合は、正常に終了する。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

パフォーマンスを改善するには、*lwork* = min(*m*, *n*)**blocksize* に設定する。*blocksize* は、**ブロック化アルゴリズム**の最適パフォーマンスに必要な値で、マシンによって異なる (通常は 16 ～ 64)。

用意すべきワークスペースの量が不明な場合は、最初は `lwork` に大きな値を設定する。終了時に `work(1)` の値を調べ、これ以降はその値を使用する。

計算で求めたマトリックス Q は、正確な直交マトリックスとマトリックス E ($\|E\|_2 = O(\varepsilon)$) だけ異なる。

浮動小数演算のおおよその総数を、**説明**の個所で示した場合ごとに示すと、次のようになる。

Q の全体を求める場合 :

$(4/3)n(3m^2 - 3m*n + n^2)$ ($m > n$ の場合)

$(4/3)m^3$ ($m \le n$ の場合)

Q の先頭から n 列を求める場合 ($m > n$):

$(2/3)n^2(3m - n^2)$ ($m > n$ の場合)

P^T の全体を求める場合 :

$(4/3)n^3$ ($m \ge n$ の場合)

$(4/3)m(3n^2 - 3m*n + m^2)$ ($m < n$ の場合)

P^T の先頭から m 列を求める場合 ($m < n$):

$(2/3)n^2(3m - n^2)$ ($m > n$ の場合)

このルーチンの複素数版は、?ungbr である。

?ormbr

?gebrd で求めた実数型の直交
マトリックス Q または P^T と任意の
実マトリックスとを掛け合わせる。

```
call sormbr (vect, side, trans, m, n, k, a, lda, tau, c, ldc, work,
             lwork, info)
call dormbr (vect, side, trans, m, n, k, a, lda, tau, c, ldc, work,
             lwork, info)
```

説明

このルーチンは、任意の実マトリックス C に対して、QC、Q^TC、CQ、CQ^T、PC、P^TC、CP、または CP^T のいずれかのマトリックス積を生成する。Q と P は、ルーチン sgebrd/dgebrd (5-76 ページを参照) を呼び出して求めた直交マトリックスである。このルーチンを実行すると、C の値が上書きされる。

入力パラメータ

下記の説明で、*r* は Q または P^T の次数を表す。
side ='L' の場合は *r* = *m* で、*side* ='R' の場合は *r* = *n* である。

vect CHARACTER*1。'Q' または 'P' でなければならない。
vect ='Q' の場合は、Q または Q^T が C に適用される。
vect ='P' の場合は、P または P^T が C に適用される。

side CHARACTER*1。'L' または 'R' でなければならない。
side ='L' の場合は、乗数が C に左側から適用される。
side ='R' の場合は、乗数が C に右側から適用される。

trans CHARACTER*1。'N' または 'T' でなければならない。
trans ='N' の場合は、Q か P が C に適用される。
trans ='T' の場合は、Q^T か P^T が C に適用される。

m INTEGER。C の行数。

n INTEGER。C の列数。

k INTEGER。?gebrd で求めた A の次元のいずれか。
vect = 'Q' の場合は、A の列数。
vect = 'P' の場合は、A の行数。
次の制約がある。$m \geq 0$, $n \geq 0$, $k \geq 0$

a, c, work
REAL (sormbr の場合)
DOUBLE PRECISION (dormbr の場合)
配列 :
*a(lda, *)* は、?gebrd から返された配列 *a*。
その第 2 次元は、*vect* = 'Q' の場合は max(1, min(*r*, *k*)) 以上で、*vect* = 'P' の場合は max(1, *r*) 以上でなければならない。
*c(ldc, *)* には、マトリックス *C* を格納する。
その第 2 次元は、max(1, n) 以上でなければならない。
work(lwork) は、ワークスペース配列である。

lda
INTEGER。*a* の第 1 次元。次の制約がある。
lda ≥ max(1, *r*) (*vect* = 'Q' の場合)、
lda ≥ max(1, min(*r*, *k*)) (*vect* = 'P' の場合)。

ldc
INTEGER。*c* の第 1 次元。*ldc* ≥ max(1, *m*)

tau
REAL (sormbr の場合)
DOUBLE PRECISION (dormbr の場合)
配列 DIMENSION は、max(1, min(*r*, *k*)) 以上。
vect = 'Q' の場合、tauq は、?gebrd から返された配列。
vect = 'P' の場合、*taup* は、?gebrd から返された配列。

lwork
INTEGER。配列 *work* のサイズ。次の制約がある。
lwork ≥ max(1, *n*) (*side* = 'L' の場合)
lwork ≥ max(1, *m*) (*side* = 'R' の場合)
lwork の推奨値は、「アプリケーション・ノート」を参照。

出力パラメータ

c
vect、*side*、*trans* の値に従って、QC、Q^TC、CQ、CQ^T、PC、P^TC、CP、または CP^T いずれかの積によって上書きされる。

work(1)
info = 0 の場合は、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の *lwork* の値が *work(1)* に格納される。これ以降の実行には、この *lwork* の値を使用する。

info
INTEGER
info = 0 の場合は、正常に終了する。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

パフォーマンスを改善するには、次のいずれかに設定する。

`lwork` = `n`**blocksize* (`side` = `'L'` の場合)

`lwork` = `m`**blocksize* (`side` = `'R'` の場合)

blocksize は、**ブロック化アルゴリズム**の最適パフォーマンスに必要な値で、マシンによって異なる (通常は 16 ～ 64)。

用意すべきワークスペースの量が不明な場合は、最初は `lwork` に大きな値を設定する。終了時に `work(1)` の値を調べ、これ以降はその値を使用する。

計算で求めた積は、正確な積とマトリックス E ($\|E\|_2 = O(\varepsilon)\,\|C\|_2$) だけ異なる。

浮動小数演算のおおよその総数は、

2*`n`*`k`(2*`m` - `k`)	(`side` = `'L'` かつ `m` ≥ `k` の場合)
2*`m`*`k`(2*`n` - `k`)	(`side` = `'R'` かつ `n` ≥ `k` の場合)
2*m^2*`n`	(`side` = `'L'` かつ `m` < `k` の場合)
2*n^2*`m`	(`side` = `'R'` かつ `n` < `k` の場合)

このルーチンの複素数版は、?unmbr である。

?ungbr

?gebrd で求めた複素ユニタリ・
マトリックス Q または P^H を生成する。

```
call cungbr ( vect, m, n, k, a, lda, tau, work, lwork, info )
call zungbr ( vect, m, n, k, a, lda, tau, work, lwork, info )
```

説明

このルーチンは、ルーチン cgebrd/zgebrd (5-76 ページを参照) で求めたユニタリ・マトリックス Q と P^H の全部または一部を生成する。このルーチンは、cgebrd/zgebrd を呼び出した後で使用する。引数の有効な組み合わせについては、**入力パラメータ**を参照。ほとんどの場合は、次のように呼び出す必要がある。

$m \times m$ のマトリックス Q の全体を求めるには、
```
call ?ungbr ('Q', m, m, n, a ...)
```
(ただし、配列 a には、列が m 個以上あること)

Q の先頭から n 列を求めるには ($m > n$ の場合)、
```
call ?ungbr ('Q', m, n, n, a ...)
```

$n \times n$ のマトリックス P^H の全体を求めるには、
```
call ?ungbr ('P', n, n, m, a ...)
```
(ただし、配列 a には、行が n 個以上あること)

P^H の先頭から m 行を求めるには ($m < n$ の場合)、
```
call ?ungbr ('P', m, n, m, a ...)
```

入力パラメータ

vect	CHARACTER*1。'Q' または 'P' でなければならない。 *vect* ='Q' の場合は、マトリックス Q が生成される。 *vect* ='P' の場合は、マトリックス P^H が生成される。
m	INTEGER。Q または P^H に必要な行数。
n	INTEGER。Q または P^H に必要な列数。
k	INTEGER。?gebrd から返された A の次元のいずれか。 *vect* ='Q' の場合は、A の列数。 *vect* ='P' の場合は、A の行数。

次の制約がある。$m \geq 0$, $n \geq 0$, $k \geq 0$
vect = 'Q' の場合は、$k \leq n \leq m$ ($m > k$ のとき) または $m = n$ ($m \leq k$ のとき)
vect = 'P' の場合は、$k \leq m \leq n$ ($n > k$ のとき) または $m = n$ ($n \leq k$ のとき)

a, work — COMPLEX (cungbr の場合)
DOUBLE COMPLEX (zungbr の場合)
配列 :
*a(lda, *)* は、?gebrd から返された配列 *a* である。
a の第 2 次元は、max(1, *n*) 以上でなければならない。
work(lwork) は、ワークスペース配列である。

lda — INTEGER。*a* の第 1 次元。max(1, *m*) 以上であること。

tau — COMPLEX (cungbr の場合)
DOUBLE COMPLEX (zungbr の場合)
vect = 'Q' の場合、*tauq* は ?gebrd から返された配列。
vect = 'P' の場合、*taup* は ?gebrd から返された配列。
tau の次元は、max(1, min(*m*, *k*)) 以上 (*vect* = 'Q' の場合)、または max(1, min(*m*, *k*)) 以上 (*vect* = 'P' の場合) でなければならない。

lwork — INTEGER。配列 *work* のサイズ。
次の制約がある。*lwork* ≥ max(1, min(*m*, *n*))
lwork の推奨値は、「アプリケーション・ノート」を参照。

出力パラメータ

a — *vect*、*m*、*n* の値に従って、直交マトリックス Q または P^T (あるいはそれに対応する先頭からの一連の行または列) によって上書きされる。

work(1) — *info* = 0 の場合は、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の *lwork* の値が *work(1)* に格納される。これ以降の実行には、この *lwork* の値を使用する。

info — INTEGER
info = 0 の場合は、正常に終了する。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

パフォーマンスを改善するには、`lwork` = min(`m`, `n`)*blocksize* に設定する。*blocksize* は、**ブロック化アルゴリズム**の最適パフォーマンスに必要な値で、マシンによって異なる (通常は 16 ～ 64)。

用意すべきワークスペースの量が不明な場合は、最初は `lwork` に大きな値を設定する。終了時に `work(1)` の値を調べ、これ以降はその値を使用する。

計算で求めた Q は、正確な直交マトリックスとマトリックス E ($\|E\|_2 = O(\varepsilon)$) だけ異なる。

浮動小数演算のおおよその総数を、説明の個所で示した場合ごとに示すと、次のようになる。

Q の全体を求める場合 :

$(16/3)n(3m^2 - 3m*n + n^2)$ ($m > n$ の場合)

$(16/3)m^3$ ($m \le n$ の場合)

Q の先頭から n 列を求める場合 ($m > n$):

$(8/3)n^2(3m - n^2)$ ($m > n$ の場合)

P^T の全体を求める場合 :

$(16/3)n^3$ ($m \ge n$ の場合)

$(16/3)m(3n^2 - 3m*n + m^2)$ ($m < n$ の場合)

P^T の先頭から m 列を求める場合 ($m < n$):

$(8/3)n^2(3m - n^2)$ ($m > n$ の場合)

このルーチンの実数版は、?orgbr である。

?unmbr

?gebrd で求めたユニタリ・マトリックス Q または P と任意の複素マトリックスとを掛け合わせる。

```
call cunmbr (vect, side, trans, m, n, k, a, lda, tau, c, ldc, work,
             lwork, info)
call zunmbr (vect, side, trans, m, n, k, a, lda, tau, c, ldc, work,
             lwork, info)
```

説明

このルーチンは、任意の複素マトリックス C に対して、QC、Q^HC、CQ、CQ^H、PC、P^HC、CP、または CP^H のいずれかのマトリックス積を生成する。Q と P は、ルーチン cgebrd/zgebrd (5-76 ページを参照) を呼び出して求めた直交マトリックスである。このルーチンを実行すると、C の値が上書きされる。

入力パラメータ

下記の説明で、*r* は Q または P^H の次数を表す。
side = 'L' の場合は *r* = *m* で、*side* = 'R' の場合は *r* = *n* である。

vect CHARACTER*1。'Q' または 'P' でなければならない。
vect = 'Q' の場合は、Q か Q^H が C に適用される。
vect = 'P' の場合は、P か P^H が C に適用される。

side CHARACTER*1。'L' または 'R' でなければならない。
side = 'L' の場合は、乗数が C に左側から適用される。
side = 'R' の場合は、乗数が C に右側から適用される。

trans CHARACTER*1。'N' または 'C' でなければならない。
trans = 'N' の場合は、Q または P が C に適用される。
trans = 'C' の場合は、Q^H または P^H が C に適用される。

m INTEGER。C の行数。

n INTEGER。C の列数。

k INTEGER。?gebrd から返された A の次元のいずれか。
vect = 'Q' の場合は、A の列数。
vect = 'P' の場合は、A の行数。
次の制約がある。$m \geq 0$, $n \geq 0$, $k \geq 0$

a, c, work
COMPLEX (cunmbr の場合)
DOUBLE COMPLEX (zunmbr の場合)
配列 :
*a(lda, *)* は、?gebrd から返された配列 *a*。
その第 2 次元は、max(1, min(r, k)) 以上 (*vect* ='Q' の場合)、または max(1, r) 以上 (*vect* ='P' の場合) でなければならない。
*c(ldc, *)* には、マトリックス C を格納する。
その第 2 次元は、max(1, n) 以上でなければならない。
work(lwork) は、ワークスペース配列である。

lda
INTEGER。*a* の第 1 次元。次の制約がある。
lda ≥ max(1, r) (*vect* ='Q' の場合)
lda ≥ max(1, min(r, k)) (*vect* ='P' の場合)

ldc
INTEGER。*c* の第 1 次元。*ldc* ≥ max(1, m)

tau
COMPLEX (cunmbr の場合)
DOUBLE COMPLEX (zunmbr の場合)
配列 DIMENSION は、max(1, min(r, k)) 以上。
vect = 'Q' の場合、*tauq* は ?gebrd から返された配列。
vect = 'P' の場合、*taup* は ?gebrd から返された配列。

lwork
INTEGER。配列 *work* のサイズ。次の制約がある。
lwork ≥ max(1, n) (*side* ='L' の場合)
lwork ≥ max(1, m) (*side* ='R' の場合)
lwork の推奨値は、「アプリケーション・ノート」を参照。

出力パラメータ

c
vect、*side*、*trans* の値に従って、QC、Q^HC、CQ、CQ^H、PC、P^HC、CP、または CP^H のいずれかの積によって上書きされる。

work(1)
info = 0 の場合は、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の *lwork* の値が *work(1)* に格納される。これ以降の実行には、この *lwork* の値を使用する。

info
INTEGER
info = 0 の場合は、正常に終了する。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

パフォーマンスを改善するには、次のいずれかに設定する。

`lwork` = `n`**blocksize* (`side` = `'L'` の場合)

`lwork` = `m`**blocksize* (`side` = `'R'` の場合)

blocksize は、**ブロック化アルゴリズム**の最適パフォーマンスに必要な値で、マシンによって異なる (通常は 16 ～ 64)。

用意すべきワークスペースの量が不明な場合は、最初は `lwork` に大きな値を設定する。終了時に `work(1)` の値を調べ、これ以降はその値を使用する。

計算で求めた積は、正確な積とマトリックス E ($\|E\|_2 = O(\varepsilon)\,\|C\|_2$) だけ異なる。

浮動小数演算のおおよその総数は、

$8*n*k(2*m - k)$	(`side` = `'L'` かつ $m \geq k$ の場合)
$8*m*k(2*n - k)$	(`side` = `'R'` かつ $n \geq k$ の場合)
$8*m^2*n$	(`side` = `'L'` かつ $m < k$ の場合)
$8*n^2*m$	(`side` = `'R'` かつ $n < k$ の場合)

このルーチンの実数版は、?ormbr である。

?bdsqr

二重対角形式に縮退された一般
マトリックスの特異値分解を求める。

```
call sbdsqr ( uplo, n, ncvt, nru, ncc, d, e, vt, ldvt, u, ldu, c, ldc,
              work, info )
call dbdsqr ( uplo, n, ncvt, nru, ncc, d, e, vt, ldvt, u, ldu, c, ldc,
              work, info )
call cbdsqr ( uplo, n, ncvt, nru, ncc, d, e, vt, ldvt, u, ldu, c, ldc,
              work, info )
call zbdsqr ( uplo, n, ncvt, nru, ncc, d, e, vt, ldvt, u, ldu, c, ldc,
              work, info )
```

説明

このルーチンは、暗黙的ゼロシフト *QR* アルゴリズムを使用して、*n* × *n* の実数の (上または下) 二重対角マトリックス *B* に対して特異値分解 (SVD) を行い、その特異値を求める。さらにオプションで、右と左の特異ベクトルも求められる。マトリックス *B* の SVD は、$B = Q * S * P^H$ と表せられる。*S* は特異値を持つ対角マトリックス、*Q* は左特異ベクトルを持つ直交マトリックス、*P* は右特異ベクトルを持つ直交マトリックスである。このサブルーチンは、与えられた実数 / 複素数の入力マトリックス *U* と *VT* に関して、左特異ベクトルが要求されると *Q* ではなく *U* * *Q* を返し、右特異ベクトルが要求されると P^H ではなく $P^H * VT$ を返す。*U* と *VT* が、?gebrd で計算されるような、一般マトリックス *A* を対角形式に縮退させた直交 / ユニタリ・マトリックスの場合 ($A = U * B * VT$)、*A* の SVD は次のようになる。

$$A = (U * Q) * S * (P^H * VT)$$

このサブルーチンはオプションで、実数 / 複素数の入力マトリックス C に対する $Q^H * C$ の計算もできる。

入力パラメータ

uplo CHARACTER*1。'U' または 'L' でなければならない。
uplo = 'U' の場合、*B* は上二重対角マトリックスである。
uplo = 'L' の場合、*B* は下二重対角マトリックスである。

n	INTEGER。マトリックス B の次数 ($n \geq 0$)。
ncvt	INTEGER。マトリックス VT の列数、つまり、右特異ベクトルの個数 ($ncvt \geq 0$)。 右特異ベクトルが不要な場合は、$ncvt = 0$ に設定する。
nru	INTEGER。U の行数、つまり左特異ベクトルの個数 ($nru \geq 0$)。 左特異ベクトルが不要な場合は、$nru = 0$ と設定する。
ncc	INTEGER。積 $Q^H C$ を計算するためのマトリックス C の列数 ($ncc \geq 0$)。 マトリックス C を指定しない場合は、$ncc = 0$ に設定する。
d, e, work	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列 : *d(*)* B の対角成分が格納される。 *d* の次元は、max(1, *n*) 以上でなければならない。 *e(*)* B の (*n*-1) の対角線外の成分が格納される。 *e* の次元は、max(1, *n*) 以上でなければならない。 *e(n)* はワークスペース用に使用される。 *work(*)* は、ワークスペース配列である。 *work* の次元は、max(1, 2**n*) 以上 ($ncvt = nru = ncc = 0$ の場合)、または max(1, 4*(*n*-1)) 以上 (それ以外の場合) でなければならない。
vt, u, c	REAL (sbdsqr の場合) DOUBLE PRECISION (dbdsqr の場合) COMPLEX (cbdsqr の場合) DOUBLE COMPLEX (zbdsqr の場合) 配列 : *vt(ldvt, *)* には、$n \times ncvt$ のマトリックス VT を格納する。 *vt* の第 2 次元のサイズは、max(1, *ncvt*) 以上でなければならない。 $ncvt = 0$ の場合は、*vt* は参照されない。 *u(ldu, *)* には、$nru \times n$ の単位マトリックス U を格納する。 *u* の第 2 次元のサイズは、max(1, *n*) 以上でなければならない。 $nru = 0$ の場合は、*u* は参照されない。

`c(ldc, *)` には、積 $Q^H * C$ を計算するためのマトリックス C を格納する。`c` の第 2 次元は、max(1, `ncc`) 以上でなければならない。
`ncc` = 0 の場合、対応する配列は参照されない。

`ldvt` INTEGER。`vt` の第 1 次元。次の制約がある。
`ldvt` ≥ max(1, `n`) (`ncvt` > 0 の場合)
`ldvt` ≥ 1 (`ncvt` = 0 の場合)

`ldu` INTEGER。`u` の第 1 次元。次の制約がある。
`ldu` ≥ max(1, `nru`)

`ldc` INTEGER。`c` の第 1 次元。次の制約がある。
`ldc` ≥ max(1, `n`) (`ncc` > 0 の場合)
`ldc` ≥ 1 (その他の場合)

出力パラメータ

`d` 終了時に、`info` = 0 (`info` を参照) の場合は、特異値によって降順に上書きされる。

`e` 終了時に、`info` = 0 の場合は、`e` は壊される。後述の `info` も参照。

`c` 積 $Q^H * C$ によって上書きされる。

`vt` 終了時に、この配列は $P^H * VT$ によって上書きされる。

`u` 終了時に、この配列は $U * Q$ によって上書きされる。

`info` INTEGER
`info` = 0 の場合は、正常に終了する。
`info` = `-i` の場合は、`i` 番目のパラメータの値が不正である。
`info` = `i` の場合は、アルゴリズムが収束していない。`i` は、収束しなかった対角線外の成分の個数を示す。その場合は、終了時に `d` と `e` に、二重対角マトリックス (B の直交マトリックスと等価) の対角成分と非対角成分が格納される。

アプリケーション・ノート

それぞれの特異値や特異ベクトルは、比較的高い精度で計算される。ただし、(このルーチンが呼び出される前に実行される) 二重対角形式への縮退によって、特異値の絶対値が大きく変動するような場合には、元のマトリックスの特異値が小さいときに相対精度が低下する場合もある。

σ_i が B の正確な特異値であり、それに対する算出値が s_i とすると、次のようになる。

$$|s_i - \sigma_i| \le p(m, n)\varepsilon\sigma_i$$

$p(m, n)$ は、漸増する m と n の関数で、ε はマシン精度である。特異値だけを計算するほうが、特異ベクトルも一緒に計算する (つまり、関数 $p(m, n)$ が小さい) よりも結果は正確である。

u_i が B の正確な左特異ベクトルであり、それに対する算出された左特異ベクトルが wi とすると、それらの間の角 $\theta(u_i, w_i)$ は次のように制限される。

$$\theta(u_i, w_i) \le p(m, n)\varepsilon / \min_{i \ne j}(|\sigma_i - \sigma_j|/|\sigma_i + \sigma_j|)$$

$\mathrm{mini}_{i \ne j}(|\sigma_i - \sigma_j|/|\sigma_i + \sigma_j|)$ は、σ_i と他の特異値との間の相対誤差である。右特異ベクトルに関しても、これと同様に誤差が制限される。

特異値だけを計算する場合の、実数型の浮動小数演算のおおよその総数は、n^2 に比例する。左特異ベクトルを計算する場合は約 $6n^2$*`nru` (複素数型の場合は約 $12n^2$*`nru`) 回だけ、右特異ベクトルを計算する場合は約 $6n^2$*`ncvt` (複素数型の場合は約 $12n^2$*`ncvt`) 回だけ、余分な計算が必要になる。

?bdsdc

分割統治法を使用して、実数の二重対角マトリックスの特異値分解を実行する。

```
call sbdsdc ( uplo, compq, n, d, e, u, ldu, vt, ldvt, q, iq, work,
              iwork, info )
call dbdsdc ( uplo, compq, n, d, e, u, ldu, vt, ldvt, q, iq, work,
              iwork, info )
```

説明

このルーチンは、分割統治法を使用して、$n \times n$ の実数の (上または下) 二重対角マトリックス B に対する特異値分解 (SVD)、すなわち $B = U \Sigma V^T$ を計算する。Σ は非負の対角成分 (B の特異値) を持つ対角マトリックス、U と V は、それぞれ左と右の特異ベクトルを持つ直交マトリックスである。?bdsdc を使用して、すべての特異値を計算し、さらにオプションで特異ベクトル (またはコンパクト形式の特異ベクトル) を求められる。

入力パラメータ

uplo	CHARACTER*1。'U' または 'L' でなければならない。 *uplo* = 'U' の場合、B は上二重対角マトリックス。 *uplo* = 'L' の場合、B は下二重対角マトリックス。
compq	CHARACTER*1。'N'、'P'、'I' のどれかでなければならない。 *compq* = 'N' の場合、特異値のみ計算する。 *compq* = 'P' の場合、特異値と、コンパクト形式の特異ベクトルを計算する。 *compq* = 'I' の場合、特異値と特異ベクトルを計算する。
n	INTEGER。マトリックス B の次数 ($n \geq 0$)。
d, e, work	REAL (sbdsdc の場合) DOUBLE PRECISION (sbdsdc の場合) 配列 :

d(*) には、二重対角マトリックス B の n 個の対角成分を格納する。d のサイズは、max(1, n) 以上でなければならない。

e(*) 二重対角マトリックス B の非対角成分を格納する。e のサイズは、max(1, n) 以上でなければならない。

work(*) は、ワークスペース配列。
work のサイズは、次の値以上でなければならない。
max(1, 4*n) (compq='N' の場合)
max(1, 6*n) (compq='P' の場合)
max(1, 3*n^2+4*n) (compq='I' の場合)

ldu　INTEGER。出力配列 u の第 1 次元のサイズ。
ldu ≥ 1。特異ベクトルも要求する場合は、
ldu ≥ max(1, n)

ldvt　INTEGER。出力配列 vt の第 1 次元のサイズ。
ldvt ≥ 1。特異ベクトルも要求する場合は、
ldvt ≥ max(1, n)

iwork　INTEGER
ワークスペース配列。サイズは max(1, 8*n) 以上。

出力パラメータ

d　info = 0 の場合は、B の特異値で上書きされる。

e　終了時に、e は上書きされる。

u, vt, q　REAL (sbdsdc の場合)
DOUBLE PRECISION (sbdsdc の場合)
配列 :u(ldu, *), vt(ldvt, *), q(*)
compq = 'I' の場合、終了時に、二重対角マトリックス B の左特異ベクトルが u に格納される (ただし、info ≠ 0 でない場合(info を参照))。compq が他の値の場合は、u は参照されない。u の第 2 次元のサイズは、max(1, n) 以上でなければならない。

compq = 'I' の場合、終了時に、二重対角マトリックス B の右特異ベクトルが vt に格納される (ただし、info ≠ 0 でない場合 (info を参照))。compq が他の値の場合は、vt は参照されない。vt の第 2 次元のサイズは、max(1, n) 以上でなければならない。

compq = 'P' の場合、info = 0 ならば終了時に、q と iq に左と右の特異ベクトルがコンパクト形式で格納される。q には、特異ベクトルのすべての REAL データ (sbdsdc の場合) または DOUBLE PRECISION データ (dbdsdc の場合) が格納される。compq が他の値の場合は、q は参照されない。詳細は、「アプリケーション・ノート」を参照。

iq　INTEGER

配列 : iq(*)

compq = 'P' の場合、info = 0 ならば終了時に、q と iq に左と右の特異ベクトルがコンパクト形式で格納される。iq には、特異ベクトルのすべての INTEGER データが格納される。compq が他の値の場合は、iq は参照されない。詳細は、「アプリケーション・ノート」を参照。

info　INTEGER

info = 0 の場合は、正常終了している。

info = -i の場合は、i 番目のパラメータ値が不正である。

info = i, の場合は、このアルゴリズムによる特異値計算に失敗した。分割統治法の更新プロセスに失敗した。

対称固有値問題

対称固有値問題は、「$n \times n$ の実対称マトリックスまたは複素エルミート・マトリックス A が与えられたときに、次の方程式を満たす**固有値** λ と、それに対応する**固有ベクトル** z を見つけること」である。

$$Az = \lambda z. \ (\text{または } z^H A = \lambda z^H)$$

このような固有値問題の場合は、実対称マトリックスだけでなく複素エルミート・マトリックス A に関しても、n 個の固有値はすべて実数型になる。そして、n 個の固有ベクトルから構成される正規直交系が 1 つ存在する。A が正値の対称マトリックスまたはエルミート・マトリックスである場合は、どの固有値も正になる。

LAPACK を使って対称固有値問題を解く場合は、通常、対応するマトリックスをまず三重対角形式に縮退した後、得られた三重対角マトリックスを使って固有値問題を解く必要がある。LAPACK には、直交 (またはユニタリ) の相似変換 $A = QTQ^H$ によってマトリックスを三重対角形式に縮退させるためのルーチンだけでなく、三重対角の対称固有値問題を解くためのルーチンも入っている。それらのルーチンを、表 5-3 に示す。

対称固有値問題を解くためのルーチンには、固有ベクトルの全部あるいは一部が必要な場合、固有値のみが必要な場合、マトリックス A が正値であるかどうかなどに応じて、さまざまなものが用意されている。
これらのルーチンは、対称問題で固有値と固有ベクトルを計算する 3 つの主要アルゴリズムに基づいている。3 つの主要アルゴリズムとは、分割統治アルゴリズム、QR アルゴリズム、二分法と逆反復法の組み合わせである。一般に、分割統治アルゴリズムが最も効率が良いので、固有値と固有ベクトルをすべて計算するときには、このアルゴリズムが推奨される。
また、分割統治アルゴリズムを使用して固有値問題を解く場合は、ルーチンを 1 つだけ呼び出せばよい。一般に、QR アルゴリズム、または二分法と逆反復法の組み合わせを使用すると、複数のルーチンを呼び出す必要がある。

図 5-2 のデシジョン・ツリーを参考にすれば、実対称マトリックスの固有値問題を解くための正しいルーチン (1 つまたは複数個) を簡単に選べる。図 5-3 は、複素エルミート・マトリックスの場合のデシジョン・ツリーである。

図 5-2 デシジョン・ツリー : 実対称マトリックスの固有値問題

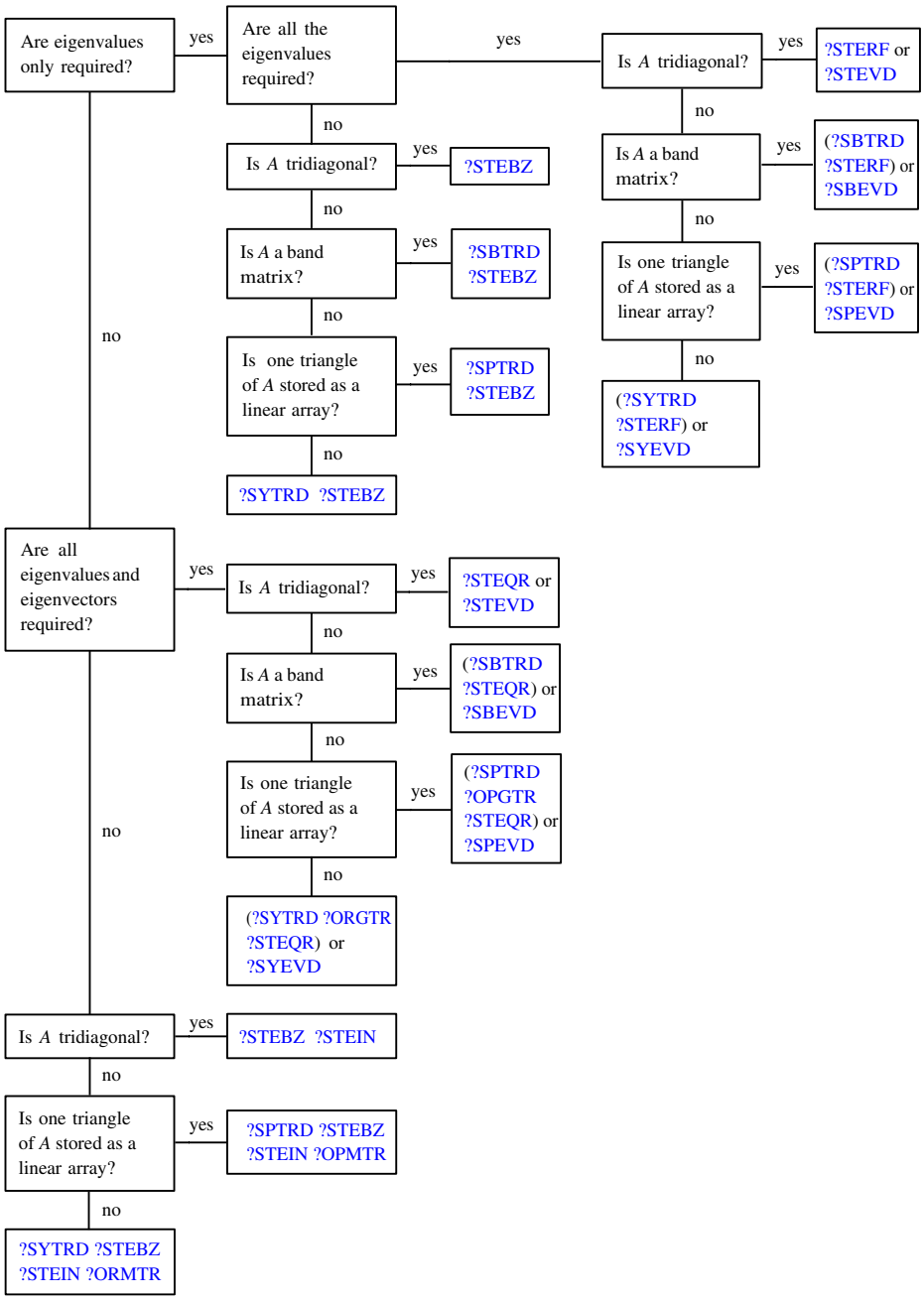


図 5-3　デシジョン・ツリー：複素エルミート・マトリックスの固有値問題

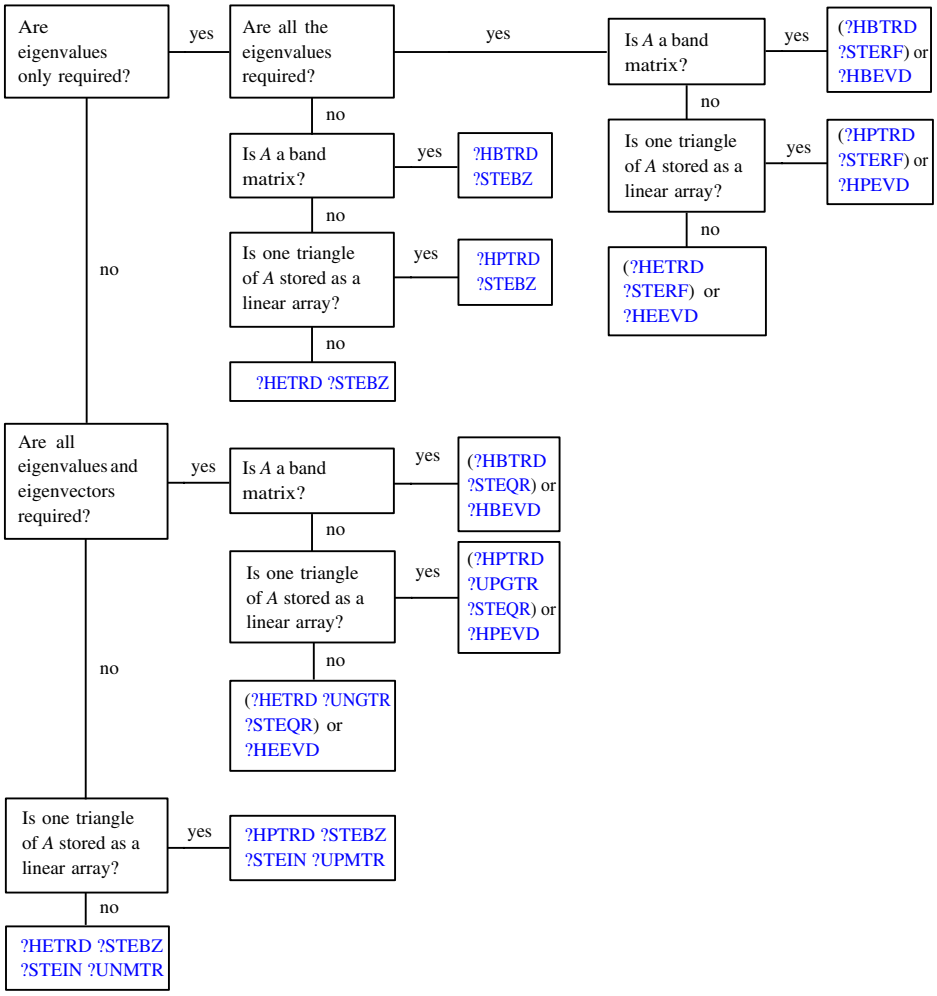


表 5-3 対称固有値問題用の計算ルーチン

処理	対称実 マトリックス	エルミート複素 マトリックス
次の三重対角形式に縮退させる $A = QTQ^H$ (フル格納)	?sytrd	?hetrd
次の三重対角形式に縮退させる $A = QTQ^H$ (パックド格納)	?sptrd	?hptrd
次の三重対角形式に縮退させる $A = QTQ^H$ (バンド格納)	?sbtrd	?hbtrd
マトリックス Q を生成する (フル格納)	?orgtr	?ungtr
マトリックス Q を生成する (パックド格納)	?opgtr	?upgtr
マトリックス Q を適用する (フル格納)	?ormtr	?unmtr
マトリックス Q を適用する (パックド格納)	?opmtr	?upmtr
三重対角マトリックス T のすべての固有値を見つける	?sterf	
三重対角マトリックス T のすべての固有値と固有ベクトルを見つける	?steqr ?stedc	?steqr ?stedc
正値三重対角マトリックス T のすべての固有値と固有ベクトルを見つける	?pteqr	?pteqr
三重対角マトリックス T の指定された固有値を求める	?stebz ?stegr	?stegr
三重対角マトリックス T の指定された固有ベクトルを求める	?stein ?stegr	?stein ?stegr
固有ベクトルの相互条件数を計算する	?disna	?disna

?sytrd

実対称マトリックスを三重対角形式に
縮退させる。

```
call ssytrd ( uplo, n, a, lda, d, e, tau, work, lwork, info )
call dsytrd ( uplo, n, a, lda, d, e, tau, work, lwork, info )
```

説明

このルーチンでは、直交相似変換 $A = QTQ^T$ によって、実対称マトリックス A を対称な三重対角形式に縮退させる。直交マトリックス Q は、明示的な形式では表現されず、(n-1) 個の基本リフレクタの積として表現される。一連のルーチンは、この形式で表現される Q を操作するためのものである (本節で後述)。

入力パラメータ

uplo	CHARACTER*1。'U' または 'L' でなければならない。 *uplo* = 'U' の場合、*a* には A の上三角部分を格納する。 *uplo* = 'L' の場合、*a* には A の下三角部分を格納する。
n	INTEGER。マトリックス A の次数 ($n \geq 0$)。
a, work	REAL (ssytrd の場合) DOUBLE PRECISION (dsytrd の場合) *a(lda, *)* の配列には、*uplo* の値に従って、マトリックス A の上三角部分または下三角部分を格納する。 *a* の第 2 次元は、max(1, *n*) 以上でなければならない。 *work(lwork)* は、ワークスペース配列である。
lda	INTEGER。*a* の第 1 次元 max(1, *n*) 以上。
lwork	INTEGER。配列 *work* のサイズ (*lwork* ≥ *n*) *lwork* の推奨値は、「アプリケーション・ノート」を参照。

出力パラメータ

a	*uplo* の値に従って、三重対角マトリックス T と直交マトリックス Q の各成分によって上書きされる。

d, e, tau	REAL (ssytrd の場合) DOUBLE PRECISION (dsytrd の場合) 配列 : *d*(*) には、マトリックス *T* の対角成分が格納される。*d* の次元は、max(1, *n*) 以上でなければならない。 *e*(*) には、*T* の対角線外の成分が格納される。*e* の次元は、max(1, *n*-1) 以上でなければならない。 *tau*(*) には、直交マトリックス *Q* の各成分が格納される。*tau* の次元は、max(1, *n*-1) 以上でなければならない。
work(1)	*info*=0 の場合は、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の *lwork* の値が *work*(1) に格納される。これ以降の実行には、この *lwork* の値を使用する。
info	INTEGER *info* = 0 の場合は、正常に終了する。 *info* = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

パフォーマンスを改善するには、*lwork* =*n***blocksize* に設定する。*blocksize* は、**ブロック化アルゴリズム**の最適パフォーマンスに必要な値で、マシンによって異なる (通常は 16 ～ 64)。用意すべきワークスペースの量が不明な場合は、最初は *lwork* に大きな値を設定する。終了時に *work*(1) の値を調べ、これ以降はその値を使用する。

計算で求めたマトリックス *T* は、マトリックス *A* + *E* の近似値である。$\|E\|_2 = c(n)\varepsilon \|A\|_2$ であり、*c*(*n*) は漸増する *n* の関数で、ε はマシンによって異なる値である。

浮動小数演算のおおよその総数は、$(4/3)n^3$ である。

このルーチンを呼び出した後は、以下のルーチンを呼び出せる。

?orgtr	計算で求めたマトリックス *Q* を明示的に生成する場合
?ormtr	実マトリックスと *Q* とを掛け合わせる場合

このルーチンの複素数版は、?hetrd である。

?orgtr

?sytrd で求めた実数型の直交
マトリックス Q を生成する。

```
call sorgtr ( uplo, n, a, lda, tau, work, lwork, info )
call dorgtr ( uplo, n, a, lda, tau, work, lwork, info )
```

説明

このルーチンでは、実対称マトリックス A を三重対角マトリックスの形式 $A = QTQ^T$ に縮退させるために ?sytrd (5-105 ページを参照) によって求めた $n \times n$ の直交マトリックス Q を明示的に生成する。このルーチンは、?sytrd を呼び出した後で使用する。

入力パラメータ

uplo	CHARACTER*1。'U' または 'L' でなければならない。 *uplo* は、?sytrd で指定したのと同じ値を使う。
n	INTEGER。マトリックス Q の次数 ($n \geq 0$)。
a, tau, work	REAL (sorgtr の場合) DOUBLE PRECISION (dorgtr の場合) 配列 : *a*(*lda*, *) は、?sytrd から返された配列 *a* である。 *a* の第 2 次元は、max(1, *n*) 以上でなければならない。 *tau*(*) は、?sytrd から返された配列 *tau* である。 *tau* の次元は、max(1, *n*-1) 以上でなければならない。 *work* (*lwork*) は、ワークスペース配列である。
lda	INTEGER。*a* の第 1 次元。max(1, *n*) 以上。
lwork	INTEGER。配列 *work* のサイズ (*lwork* ≥ *n*) *lwork* の推奨値は、「アプリケーション・ノート」を参照。

出力パラメータ

a	直交マトリックス Q によって上書きされる。

work(1)	*info* = 0 の場合は、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の *lwork* の値が *work*(1) に格納される。これ以降の実行には、この *lwork* の値を使用する。
info	INTEGER *info* = 0 の場合は、正常に終了する。 *info* = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

パフォーマンスを改善するには、*lwork* = (*n*-1)**blocksize* に設定する。*blocksize* は、**ブロック化アルゴリズム**の最適パフォーマンスに必要な値で、マシンによって異なる (通常は 16 ～ 64)。

用意すべきワークスペースの量が不明な場合は、最初は *lwork* に大きな値を設定する。終了時に *work*(1) の値を調べ、これ以降はその値を使用する。

計算で求めた Q は、正確な直交マトリックスとマトリックス E ($\|E\|_2 = O(\varepsilon)$、ε はマシン精度) だけ異なる。

浮動小数演算のおおよその総数は、$(4/3)n^3$ である。

このルーチンの複素数版は、?ungtr である。

?ormtr

?sytrd で求めた実数型の直交
マトリックス Q と実マトリックスとを
掛け合わせる。

```
call sormtr ( side, uplo, trans, m, n, a, lda, tau, c, ldc, work,
              lwork, info )
call dormtr ( side, uplo, trans, m, n, a, lda, tau, c, ldc, work,
              lwork, info )
```

説明

このルーチンでは、実マトリックス C と Q または Q^T とを掛け合わせる。Q は、実対称マトリックス A を三重対角形式 $A = QTQ^T$ に縮退させるために ?sytrd (5-105 ページを参照) によって求めた直交マトリックスである。このルーチンは、?sytrd を呼び出した後で使用する。

このルーチンを使用すれば、パラメータ *side* と *trans* に応じて、QC、Q^TC、CQ、または CQ^T のいずれかのマトリックス積を求められる (C の値は上書きされる)。

入力パラメータ

下記の説明で、*r* は Q の次数を表す。
side ='L' の場合は *r* = *m* で、*side* ='R' の場合は *r* = *n* である。

side CHARACTER*1。'L' または 'R' でなければならない。
side ='L' の場合、Q または Q^T は、C に左側から適用される。
side ='R' の場合、Q または Q^T は、C に右側から適用される。

uplo CHARACTER*1。'U' または 'L' でなければならない。
uplo は、?sytrd で指定したのと同じ値を使う。

trans CHARACTER*1。'N' または 'T' でなければならない。
trans ='N' の場合、C に Q が掛けられる。
trans ='T' の場合、C に Q^T が掛けられる。

m INTEGER。マトリックス C の行数 ($m \geq 0$)。

n INTEGER。マトリックス C の列数 ($n \geq 0$)。

a, work, tau, c REAL (sormtr の場合)
DOUBLE PRECISION (dormtr の場合)
*a(lda, *)* と *tau* は、?sytrd から返された配列である。

a の第 2 次元は、max(1, r) 以上でなければならない。
tau の次元は、max(1, r-1) 以上でなければならない。
c(ldc, *) には、マトリックス C を格納する。
c の第 2 次元は、max(1, n) 以上でなければならない。
work(lwork) は、ワークスペース配列である。

lda　INTEGER。a の第 1 次元 ; lda ≥ max(1, r)

ldc　INTEGER。c の第 1 次元。ldc ≥ max(1, n)

lwork　INTEGER。配列 work のサイズ。次の制約がある。
lwork ≥ max(1, n) (side='L' の場合)
lwork ≥ max(1, m) (side='R' の場合)
lwork の推奨値は、「アプリケーション・ノート」を参照。

出力パラメータ

c　(side と trans の指定に従って) QC、Q^TC、CQ、または CQ^T のいずれかの積によって上書きされる。

work(1)　info = 0 の場合は、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の lwork の値が work(1) に格納される。これ以降の実行には、この lwork の値を使用する。

info　INTEGER
info = 0 の場合は、正常に終了する。
info = -i の場合は、i 番目のパラメータの値が不正である。

アプリケーション・ノート

パフォーマンスを改善するには、lwork = n*blocksize (side='L' の場合) か、lwork = m*blocksize (side='R' の場合) に設定する。blocksize は、**ブロック化アルゴリズム**の最適パフォーマンスに必要な値で、マシンによって異なる (通常は 16 ～ 64)。用意すべきワークスペースの量が不明な場合は、最初は lwork に大きな値を設定する。終了時に work(1) の値を調べ、これ以降はその値を使用する。

計算で求めた積は、正確な積とマトリックス E ($\|E\|_2 = O(\varepsilon)\|C\|_2$) だけ異なる。

浮動小数演算のおおよその総数は、$2*m^2*n$ (side='L' の場合) か、$2*n^2*m$ (side='R' の場合) である。

このルーチンの複素数版は、?unmtr である。

?hetrd

複素エルミート・マトリックスを
三重対角形式に縮退させる。

```
call chetrd ( uplo, n, a, lda, d, e, tau, work, lwork, info )
call zhetrd ( uplo, n, a, lda, d, e, tau, work, lwork, info )
```

説明

このルーチンでは、ユニタリ相似変換 $A = QTQ^H$ によって、複素エルミート・マトリックス A を対称な三重対角形式 T に縮退させる。ユニタリ・マトリックス Q は、明示的な形式では表現されず、(n-1) 個の基本リフレクタの積として表現される。一連のルーチンは、この形式で表現される Q を操作するためのものである (本節で後述)。

入力パラメータ

uplo CHARACTER*1。'U' または 'L' でなければならない。
uplo = 'U' の場合、*a* には A の上三角部分を格納する。
uplo = 'L' の場合、*a* には A の下三角部分を格納する。

n INTEGER。マトリックス A の次数 ($n \geq 0$)。

a, work COMPLEX (chetrd の場合)
DOUBLE COMPLEX (zhetrd の場合)
配列 *a(lda, *)* には、*uplo* の値に従って、マトリックス A の上三角部分または下三角部分を格納する。
a の第 2 次元は、max(1, *n*) 以上でなければならない。
work(lwork) は、ワークスペース配列である。

lda INTEGER。*a* の第 1 次元。max(1, *n*) 以上。

lwork INTEGER。配列 *work* のサイズ (*lwork* ≥ *n*)。
lwork の推奨値は、「アプリケーション・ノート」を参照。

出力パラメータ

a *uplo* の値に従って、三重対角マトリックス T とユニタリ・マトリックス Q の各成分によって上書きされる。

d, e	REAL (chetrd の場合) DOUBLE PRECISION (zhetrd の場合) 配列 : *d*(*) には、マトリックス *T* の対角成分が格納される。 *d* の次元は、max(1, *n*) 以上でなければならない。 *e*(*) には、*T* の対角線外の成分が格納される。 *e* の次元は、max(1, *n*-1) 以上でなければならない。
tau	COMPLEX (chetrd の場合) DOUBLE COMPLEX (zhetrd の場合) 配列 DIMENSION は、max(1, *n*-1) 以上。 ユニタリ・マトリックス *Q* の各成分が格納される。
work(1)	*info* = 0 の場合は、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の *lwork* の値が *work*(1) に格納される。これ以降の実行には、この *lwork* の値を使用する。
info	INTEGER *info* = 0 の場合は、正常に終了する。 *info* = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

パフォーマンスを改善するには、*lwork* =*n***blocksize* に設定する。*blocksize* は、**ブロック化アルゴリズム**の最適パフォーマンスに必要な値で、マシンによって異なる (通常は 16 ～ 64)。用意すべきワークスペースの量が不明な場合は、最初は *lwork* に大きな値を設定する。終了時に *work*(1) の値を調べ、これ以降はその値を使用する。

計算で求めたマトリックス *T* は、マトリックス *A* + *E* の近似値である。$\|E\|_2 = c(n)\varepsilon \|A\|_2$ であり、*c*(*n*) は漸増する *n* の関数で、ε はマシンによって異なる値である。

浮動小数演算のおおよその総数は、$(16/3)n^3$ である。

このルーチンを呼び出した後は、以下のルーチンを呼び出せる。

?ungtr	計算で求めたマトリックス *Q* を明示的に生成する場合
?unmtr	複素マトリックスを *Q* に掛け合わせる場合

このルーチンの実数版は、?sytrd である。

?ungtr

?hetrd で求めた複素ユニタリ・
マトリックス Q を生成する。

```
call cungtr ( uplo, n, a, lda, tau, work, lwork, info )
call zungtr ( uplo, n, a, lda, tau, work, lwork, info )
```

説明

このルーチンは、複素エルミート・マトリックス A を三重対角形式 $A = QTQ^H$ に縮退するために ?hetrd (5-111 ページを参照) によって求めた $n \times n$ のユニタリ・マトリックス Q を明示的に生成する。このルーチンは、?hetrd を呼び出した後で使用する。

入力パラメータ

uplo	CHARACTER*1。'U' または 'L' でなければならない。 *uplo* は、?hetrd で指定したのと同じものを使う。
n	INTEGER。マトリックス Q の次数 ($n \geq 0$)。
a, tau, work	COMPLEX (cungtr の場合) DOUBLE COMPLEX (zungtr の場合) 配列 : *a*(*lda*, *) は、?hetrd から返された配列 *a* である。 *a* の第 2 次元は、max(1, *n*) 以上でなければならない。 *tau*(*) は、?hetrd から返された配列 *tau* である。 *tau* の次元は、max(1, *n*-1) 以上でなければならない。 *work* (*lwork*) は、ワークスペース配列である。
lda	INTEGER。*a* の第 1 次元。max(1, *n*) 以上。
lwork	INTEGER。配列 *work* のサイズ (*lwork* ≥ *n*)。 *lwork* の推奨値は、「アプリケーション・ノート」を参照。

出力パラメータ

a	ユニタリ・マトリックス Q によって上書きされる。

work(1)	info = 0 の場合は、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の lwork の値が work(1) に格納される。これ以降の実行には、この lwork の値を使用する。
info	INTEGER info = 0 の場合は、正常に終了する。 info = -i の場合は、i 番目のパラメータの値が不正である。

アプリケーション・ノート

パフォーマンスを改善するには、lwork = (n-1)*blocksize に設定する。blocksize は、**ブロック化アルゴリズム**の最適パフォーマンスに必要な値で、マシンによって異なる (通常は 16 ～ 64)。

用意すべきワークスペースの量が不明な場合は、最初は lwork に大きな値を設定する。終了時に work(1) の値を調べ、これ以降はその値を使用する。

計算で求めた Q は、正確なユニタリ・マトリックスとマトリックス E ($\|E\|_2 = O(\varepsilon)$、ε はマシン精度) だけ異なる。

浮動小数演算のおおよその総数は、$(16/3)n^3$ である。

このルーチンの実数版は、?orgtr である。

?unmtr

?hetrd で求めた複素ユニタリ・
マトリックス Q に複素マトリックスを
掛け合わせる。

```
call cunmtr ( side, uplo, trans, m, n, a, lda, tau, c, ldc, work,
              lwork, info )
call zunmtr ( side, uplo, trans, m, n, a, lda, tau, c, ldc, work,
              lwork, info )
```

説明

このルーチンは、複素マトリックス C と Q または Q^H (Q は、複素エルミート・マトリックス A を三重対角形式 $A = QTQ^H$ に縮退するために ?hetrd (5-111 ページを参照) によって求めたユニタリ・マトリックス) とを掛け合わせる。このルーチンは、?hetrd を呼び出した後で使用する。

このルーチンを使用すれば、パラメータ *side* と *trans* に応じて、QC、Q^HC、CQ、または CQ^H のいずれかのマトリックス積を求められる (C の値は上書きされる)。

入力パラメータ

下記の説明で、*r* は Q の次数を表す。
side ='L' の場合は *r* = *m* で、*side* ='R' の場合は *r* = *n* である。

side	CHARACTER*1。'L' または 'R' でなければならない。 *side* ='L' の場合、Q または Q^H は、C に左側から適用される。 *side* ='R' の場合、Q または Q^H は、C に右側から適用される。
uplo	CHARACTER*1。'U' または 'L' でなければならない。 *uplo* は、?hetrd で指定したのと同じものを使う。
trans	CHARACTER*1。'N' または 'T' でなければならない。 *trans* ='N' の場合、C に Q が掛けられる。 *trans* ='T' の場合、C に Q^H が掛けられる。
m	INTEGER。マトリックス C の行数 ($m \geq 0$)。
n	INTEGER。マトリックス C の列数 ($n \geq 0$)。
a, work, tau, c	COMPLEX (cunmtr の場合) DOUBLE COMPLEX (zunmtr の場合) *a(lda, *)* と *tau* は、?hetrd から返された配列である。

a の第 2 次元は、max(1, r) 以上でなければならない。
tau の次元は、max(1, r-1) 以上でなければならない。
c(ldc, *) には、マトリックス C を格納する。
c の第 2 次元は、max(1, n) 以上でなければならない。
work(lwork) は、ワークスペース配列である。

lda　INTEGER。a の第 1 次元。lda ≥ max(1, r)

ldc　INTEGER。c の第 1 次元。ldc ≥ max(1, n)

lwork　INTEGER。配列 work のサイズ。次の制約がある。
lwork ≥ max(1, n) (side='L' の場合)
lwork ≥ max(1, m) (side='R' の場合)
lwork の推奨値は、「アプリケーション・ノート」を参照。

出力パラメータ

c　(side と trans の指定に従って) QC、Q^HC、CQ、または CQ^H のいずれかの積によって上書きされる。

work(1)　info = 0 の場合は、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の lwork の値が work(1) に格納される。これ以降の実行には、この lwork の値を使用する。

info　INTEGER
info = 0 の場合は、正常に終了する。
info = -i の場合は、i 番目のパラメータの値が不正である。

アプリケーション・ノート

パフォーマンスを改善するには、lwork = n*blocksize (side='L' の場合) か、lwork = m*blocksize (side='R' の場合) に設定する。*blocksize* は、**ブロック化アルゴリズム**の最適パフォーマンスに必要な値で、マシンによって異なる (通常は 16 ～ 64)。用意すべきワークスペースの量が不明な場合は、最初は lwork に大きな値を設定する。終了時に work(1) の値を調べ、これ以降はその値を使用する。

計算で求めた積は、正確な積とマトリックス E ($\|E\|_2 = O(\varepsilon)\|C\|_2$、ε はマシン精度) だけ異なる。

浮動小数演算のおおよその総数は、$8*m^2*n$ (side='L' の場合) か、$8*n^2*m$ (side='R' の場合) である。

このルーチンの実数版は、?ormtr である。

?sptrd

パックド格納形式の実対称マトリックスを
三重対角形式に縮退させる。

```
call ssptrd ( uplo, n, ap, d, e, tau, info )
call dsptrd ( uplo, n, ap, d, e, tau, info )
```

説明

このルーチンでは、直交相似変換 $A = QTQ^T$ によって、パックド格納形式の実対称マトリックス A を対称の三重対角形式 T に縮退させる。直交マトリックス Q は、明示的な形式では表現されず、(n-1) 個の基本リフレクタの積として表現される。一連のルーチンは、この形式で表現される Q を操作するためのものである (本節で後述)。

入力パラメータ

uplo CHARACTER*1。'U' または 'L' でなければならない。
uplo ='U' の場合は、*ap* に A の上三角マトリックス (パックド形式) を格納する。
uplo ='L' の場合は、*ap* に A の下三角マトリックス (パックド形式) を格納する。

n INTEGER。マトリックス A の次数 ($n \geq 0$)。

ap REAL (ssptrd の場合)
DOUBLE PRECISION (dsptrd の場合)
配列 DIMENSION は、max(1, n(n+1)/2) 以上。
uplo の値に従って、パックド形式の A の上三角マトリックスまたは下三角マトリックスを格納する。

出力パラメータ

ap *uplo* の値に従って、三重対角マトリックス T と直交マトリックス Q の各成分によって上書きされる。

d, e, tau REAL (ssptrd の場合)
DOUBLE PRECISION (dsptrd の場合)
配列 :
d(*) には、マトリックス T の対角成分が格納される。
d の次元は、max(1, n) 以上でなければならない。

`e(*)` には、T の対角線外の成分が格納される。
`e` の次元は、max(1, n-1) 以上でなければならない。
`tau(*)` には、マトリックス Q の各成分が格納される。
`tau` の次元は、max(1, n-1) 以上でなければならない。

`info` INTEGER
`info` = 0 の場合は、正常に終了する。
`info` = `-i` の場合は、`i` 番目のパラメータの値が不正である。

アプリケーション・ノート

計算で求めたマトリックス T は、マトリックス $A + E$ の近似値である。$\|E\|_2 = c(n)\varepsilon \|A\|_2$ であり、$c(n)$ は漸増する n の関数で、ε はマシンによって異なる値である。

浮動小数演算のおおよその総数は、$(4/3)n^3$ である。

このルーチンを呼び出した後は、以下のルーチンを呼び出せる。

?opgtr	計算で求めたマトリックス Q を明示的に生成する場合
?opmtr	実マトリックスと Q とを掛け合わせる場合

このルーチンの複素数版は、?hptrd である。

?opgtr

?sptrd で求めた実数型直交マトリックス
Q を生成する。

```
call sopgtr ( uplo, n, ap, tau, q, ldq, work, info )
call dopgtr ( uplo, n, ap, tau, q, ldq, work, info )
```

説明

このルーチンでは、パックド形式の実対称マトリックス A を三重対角形式 $A = QTQ^T$ に縮退させるのに ?sptrd (5-117 ページを参照) で求めた $n \times n$ の直交マトリックス Q を明示的に生成する。このルーチンは、?sptrd を呼び出した後で使用する。

入力パラメータ

uplo CHARACTER*1。'U' または 'L' でなければならない。
?sptrd で指定したのと同じ値を使う。

n INTEGER。マトリックス Q の次数 ($n \geq 0$)。

ap, tau REAL (sopgtr の場合)
DOUBLE PRECISION (dopgtr の場合)
配列 *ap* と *tau* は、?sptrd から返された配列である。
ap の次元は、max(1, $n(n+1)/2$) 以上でなければならない。
tau の次元は、max(1, n-1) 以上でなければならない。

ldq INTEGER。出力配列 *q* の第 1 次元は、max(1, n) 以上でなければならない。

work REAL (sopgtr の場合)
DOUBLE PRECISION (dopgtr の場合)
ワークスペース配列 DIMENSION は、max(1, n-1) 以上。

出力パラメータ

q REAL (sopgtr の場合)
DOUBLE PRECISION (dopgtr の場合)
配列 DIMENSION は、(*ldq*, *)
計算で求めたマトリックス Q が格納される。
q の第 2 次元は、max(1, n) 以上でなければならない。

info INTEGER
info = 0 の場合は、正常に終了する。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

計算で求めた Q は、正確な直交マトリックスとマトリックス E ($\|E\|_2 = O(\varepsilon)$、ε はマシン精度) だけ異なる。

浮動小数演算のおおよその総数は、$(4/3)n^3$ である。

このルーチンの複素数版は、?upgtr である。

?opmtr

?sptrd で求めた実数型直交マトリックス Q に実マトリックスを掛け合わせる。

```
call sopmtr (side, uplo, trans, m, n, ap, tau, c, ldc, work, info)
call dopmtr (side, uplo, trans, m, n, ap, tau, c, ldc, work, info)
```

説明

このルーチンでは、実マトリックス C と Q または Q^T を掛け合わせる。Q は、パックド形式の実対称マトリックス A を三重対角形式 $A = QTQ^T$ に縮退させるのに ?sptrd (5-117 ページを参照) で求めた直交マトリックスである。このルーチンは、?sptrd を呼び出した後で使用する。

このルーチンを使用すれば、パラメータ *side* と *trans* に応じて、QC、Q^TC、CQ、または CQ^T のいずれかのマトリックス積を求められる (C の値は上書きされる)。

入力パラメータ

下記の説明で、*r* は Q の次数を表す。
side = 'L' の場合は *r* = *m*、*side* = 'R' の場合は *r* = *n* である。

side CHARACTER*1。'L' または 'R' でなければならない。
side = 'L' の場合、Q または Q^T は、C に左側から適用される。
side = 'R' の場合、Q または Q^T は、C に右側から適用される。

uplo CHARACTER*1。'U' または 'L' でなければならない。
?sptrd で指定したのと同じ値を使う。

trans CHARACTER*1。'N' または 'T' でなければならない。
trans ='N' の場合、C に Q が掛けられる。
trans ='T' の場合、C に Q^T が掛けられる。

m INTEGER。マトリックス C の行数 ($m \geq 0$)。

n INTEGER。マトリックス C の列数 ($n \geq 0$)。

ap, work, tau, c
REAL (sopmtr の場合)
DOUBLE PRECISION (dopmtr の場合)
ap と *tau* は、?sptrd から返された配列である。
ap の次元は、max(1, *r*(*r*+1)/2) 以上でなければならない。
tau の次元は、max(1, *r*-1) 以上でなければならない。
*c(ldc, *)* には、マトリックス C を格納する。
c の第 2 次元は、max(1, *n*) 以上でなければならない。
work(*) は、ワークスペース配列である。
work の次元は、次の値以上でなければならない。
max(1, *n*) (*side* ='L' の場合)
max(1, *m*) (*side* ='R' の場合)

ldc INTEGER。*c* の第 1 次元。*ldc* ≥ max(1, *n*)

出力パラメータ

c (*side* と *trans* の指定に従って) QC、Q^TC、CQ、または CQ^T のいずれかの積によって上書きされる。

info INTEGER
info = 0 の場合は、正常に終了する。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

計算で求めた積は、正確な積とマトリックス E ($\|E\|_2 = O(\varepsilon)\|C\|_2$、ε はマシン精度) だけ異なる。

浮動小数演算のおおよその総数は、2*m^2*n (*side* ='L' の場合) あるいは 2*n^2*m (*side* ='R' の場合) である。

このルーチンの複素数版は、?upmtr である。

?hptrd

パックド格納形式の複素エルミート・
マトリックスを三重対角形式に
縮退させる。

```
call chptrd ( uplo, n, ap, d, e, tau, info )
call zhptrd ( uplo, n, ap, d, e, tau, info )
```

説明

このルーチンでは、ユニタリ相似変換 $A = QTQ^H$ によって、パックド形式の複素エルミート・マトリックス A を対称の三重対角形式 T に縮退させる。ユニタリ・マトリックス Q は、明示的な形式では表現されず、(n-1) 個の基本リフレクタの積として表現される。一連のルーチンは、この形式で表現される Q を操作するためのものである (本節で後述)。

入力パラメータ

uplo CHARACTER*1。'U' または 'L' でなければならない。
uplo ='U' の場合は、*ap* に A の上三角マトリックス (パックド形式) を格納する。
uplo ='L' の場合は、*ap* に A の下三角マトリックス (パックド形式) を格納する。

n INTEGER。マトリックス A の次数 ($n \geq 0$)。

ap COMPLEX (chptrd の場合)
DOUBLE COMPLEX (zhptrd の場合)
配列 DIMENSION は、max(1, *n* (*n*+1)/ 2) 以上。
uplo の値に従って、パックド形式の A の上三角マトリックスまたは下三角マトリックスを格納する。

出力パラメータ

ap *uplo* の値に従って、三重対角マトリックス T と直交マトリックス Q の各成分によって上書きされる。

d, e REAL (chptrd の場合)
DOUBLE PRECISION (zhptrd の場合)
配列 :
d(*) には、マトリックス T の対角成分が格納される。
d の次元は、max(1, *n*) 以上でなければならない。

	`e(*)` には、T の対角線外の成分が格納される。 `e` の次元は、max(1, n-1) 以上でなければならない。
tau	`COMPLEX` (`chptrd` の場合) `DOUBLE COMPLEX` (`zhptrd` の場合) 配列 `DIMENSION` は、max(1, n-1) 以上。 直交マトリックス Q の各成分が格納される。
info	`INTEGER` *info* = 0 の場合は、正常に終了する。 *info* = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

計算で求めたマトリックス T は、マトリックス $A + E$ の近似値である。$\|E\|_2 = c(n)\varepsilon \|A\|_2$ であり、$c(n)$ は漸増する n の関数で、ε はマシンによって異なる値である。

浮動小数演算のおおよその総数は、$(16/3)n^3$ である。

このルーチンを呼び出した後は、以下のルーチンを呼び出せる。

?upgtr	計算で求めたマトリックス Q を明示的に生成する場合
?upmtr	複素マトリックスを Q に掛け合わせる場合

このルーチンの実数版は、?sptrd である。

?upgtr

?hptrd で求めた複素ユニタリ・
マトリックス Q を生成する。

```
call cupgtr ( uplo, n, ap, tau, q, ldq, work, info )
call zupgtr ( uplo, n, ap, tau, q, ldq, work, info )
```

説明

このルーチンでは、パックド形式の複素エルミート・マトリックス A を三重対角形式 $A = QTQ^H$ に縮退させるのに ?hptrd (5-122 ページを参照) で求めた $n \times n$ のユニタリ・マトリックス Q を明示的に生成する。このルーチンは、?hptrd を呼び出した後で使用する。

入力パラメータ

uplo	CHARACTER*1。'U' または 'L' でなければならない。?hptrd で指定したのと同じ値を使う。
n	INTEGER。マトリックス Q の次数 ($n \geq 0$)。
ap, *tau*	COMPLEX (cupgtr の場合) DOUBLE COMPLEX (zupgtr の場合) 配列 *ap* と *tau* は、?hptrd から返された配列である。 *ap* の次元は、max(1, *n*(*n*+1)/2) 以上でなければならない。 *tau* の次元は、max(1, *n*-1) 以上でなければならない。
ldq	INTEGER。出力配列 *q* の第 1 次元は、max(1, *n*) 以上でなければならない。
work	COMPLEX (cupgtr の場合) DOUBLE COMPLEX (zupgtr の場合) ワークスペース配列 DIMENSION は、max(1, *n*-1) 以上。

出力パラメータ

q COMPLEX (cupgtr の場合)
DOUBLE COMPLEX (zupgtr の場合)
配列 DIMENSION は (*ldq*, *)
計算で求めたマトリックス Q が格納される。
q の第 2 次元は、max(1, *n*) 以上でなければならない。

info INTEGER
info = 0 の場合は、正常に終了する。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

計算で求めた Q は、正確な直交マトリックスとマトリックス E ($\|E\|_2 = O(\varepsilon)$、ε はマシン精度) だけ異なる。

浮動小数演算のおおよその総数は、$(16/3)n^3$ である。

このルーチンの実数版は、?opgtr である。

?upmtr

?hptrd で求めたユニタリ・マトリックス Q に複素マトリックスを掛け合わせる。

```
call cupmtr (side, uplo, trans, m, n, ap, tau, c, ldc, work, info)
call zupmtr (side, uplo, trans, m, n, ap, tau, c, ldc, work, info)
```

説明

このルーチンでは、複素マトリックス C に Q または Q^H を掛け合わせる。Q は、パックド形式の複素エルミート・マトリックス A を三重対角形式 $A = QTQ^H$ に縮退させるのに ?hptrd (5-122 ページを参照) で求めたユニタリ・マトリックスである。このルーチンは、?hptrd を呼び出した後で使用する。

このルーチンを使用すれば、パラメータ *side* と *trans* に応じて、QC、Q^HC、CQ、または CQ^H のいずれかのマトリックス積を求められる (C の値は上書きされる)。

入力パラメータ

下記の説明で、*r* は *Q* の次数を表す。
side = 'L' の場合は *r* = *m* で、*side* = 'R' の場合は *r* = *n* である。

side
CHARACTER*1。'L' または 'R' でなければならない。
side = 'L' の場合は、*Q* または Q^H は、*C* に左側から適用される。
side = 'R' の場合は、*Q* または Q^H は、*C* に右側から適用される。

uplo
CHARACTER*1。'U' または 'L' でなければならない。
?hptrd で指定したのと同じ値を使う。

trans
CHARACTER*1。'N' または 'T' でなければならない。
trans = 'N' の場合は、*C* に *Q* が掛けられる。
trans = 'T' の場合は、*C* に Q^H が掛けられる。

m
INTEGER。マトリックス *C* の行数 ($m \geq 0$)。

n
INTEGER。マトリックス *C* の列数 ($n \geq 0$)。

ap, tau, c, work
COMPLEX (cupmtr の場合)
DOUBLE COMPLEX (zupmtr の場合)
ap と *tau* は、?hptrd から返された配列である。
ap の次元は、max(1, *r*(*r*+1)/2) 以上でなければならない。
tau の次元は、max(1, *r*-1) 以上でなければならない。
c(*ldc*, *) には、マトリックス *C* を格納する。
c の第 2 次元は、max(1, *n*) 以上でなければならない。
work(*) は、ワークスペース配列である。
work の次元は、次の値以上でなければならない。
max(1, *n*) (*side* = 'L' の場合)
max(1, *m*) (*side* = 'R' の場合)

ldc
INTEGER。*c* の第 1 次元。*ldc* ≥ max(1, *n*)

出力パラメータ

c
(*side* と *trans* の指定に従って) *QC*、Q^HC、*CQ*、または CQ^H のいずれかの積によって上書きされる。

info
INTEGER
info = 0 の場合は、正常に終了する。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

計算で求めた積は、正確な積とマトリックス E ($\|E\|_2 = O(\varepsilon)\|C\|_2$、$\varepsilon$ はマシン精度) だけ異なる。

浮動小数演算のおおよその総数は、$8*m^2*n$ (`side='L'` の場合)、または $8*n^2*m$ (`side='R'` の場合) である。

このルーチンの実数版は、?opmtr である。

?sbtrd

実対称バンド・マトリックスを
三重対角形式に縮退させる。

```
call ssbtrd (vect, uplo, n, kd, ab, ldab, d, e, q, ldq, work, info)
call dsbtrd (vect, uplo, n, kd, ab, ldab, d, e, q, ldq, work, info)
```

説明

このルーチンでは、直交相似変換 $A = QTQ^T$ によって、実対称バンド・マトリックス A を対称の三重対角形式 T に縮退させる。直交マトリックス Q は、Givens 回転マトリックスの積として得られる。そのため、必要に応じて、このルーチンを使って、これを明示的に求められる。

入力パラメータ

vect	CHARACTER*1。'V' または 'N' でなければならない。 *vect* = 'V' の場合は、マトリックス Q が明示的に返される。 *vect* = 'N' の場合は、Q は返されない。
uplo	CHARACTER*1。'U' または 'L' でなければならない。 *uplo* = 'U' の場合、*ab* に A の上三角部分を格納する。 *uplo* = 'L' の場合、*ab* に A の下三角部分を格納する。
n	INTEGER。マトリックス A の次数 ($n \geq 0$)。
kd	INTEGER。A の優対角成分と劣対角成分の個数 ($kd \geq 0$)。
ab, work	REAL (ssbtrd の場合) DOUBLE PRECISION (dsbtrd の場合) *ab* (*ldab*, *) は、(*uplo* での指定に従って) マトリックス A の上三角部分または下三角部分をバンド形式で格納する配列である。 *ab* の第 2 次元は、max(1, *n*) 以上でなければならない。 *work* (*) は、ワークスペース配列である。 *work* の次元は、次の値以上でなければならない。max(1, *n*)
ldab	INTEGER。*ab* の第 1 次元。*kd*+1 以上。

ldq INTEGER。*q* の第 1 次元。次の制約がある。
ldq ≥ max(1, *n*) (*vect* = 'V' の場合)
ldq ≥ 1 (*vect* = 'N' の場合)

出力パラメータ

ab 終了時に、配列 *ab* が上書きされる。

d, e, q REAL (ssbtrd の場合)
DOUBLE PRECISION (dsbtrd の場合)
配列 :
d(*) には、マトリックス *T* の対角成分が格納される。
d の次元は、max(1, *n*) 以上でなければならない。
e(*) には、*T* の対角線外の成分が格納される。
e の次元は、max(1, *n*-1) 以上でなければならない。
q(*ldq*, *) は、*vect* = 'N' の場合、参照されない。
vect = 'V' の場合、*q* には *n* × *n* のマトリックス *Q* が格納される。
q の第 2 次元は、次の条件を満たしていなければならない。
max(1, *n*) 以上 (*vect* = 'V' の場合)
1 以上 (*vect* = 'N' の場合)

info INTEGER
info = 0 の場合は、正常に終了する。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

計算で求めたマトリックス *T* は、マトリックス *A* + *E* の近似値である。$\|E\|_2 = c(n)\varepsilon \|A\|_2$ であり、$c(n)$ は漸増する (n の) 関数で、ε はマシン精度である。計算で求めたマトリックス *Q* は、正確な直交マトリックスとマトリックス *E* ($\|E\|_2 = O(\varepsilon)$) だけ異なる。

浮動小数演算のおおよその総数は、*vect* = 'N' の場合は、$6n^2$*kd d である。*vect* = 'V' の場合は、$3n^3$*(*kd*-1)/*kd* だけ余分な演算が必要になる。

このルーチンの複素数版は、?hbtrd である。

?hbtrd

複素エルミート・バンド・マトリックスを
三重対角形式に縮退させる。

```
call chbtrd (vect, uplo, n, kd, ab, ldab, d, e, q, ldq, work, info)
call zhbtrd (vect, uplo, n, kd, ab, ldab, d, e, q, ldq, work, info)
```

説明

このルーチンでは、ユニタリ相似変換 $A = QTQ^H$ によって、複素エルミート・バンド・マトリックス A を対称の三重対角形式 T に縮退させる。ユニタリ・マトリックス Q は、Givens 回転マトリックスの積として得られる。そのため、必要に応じて、このルーチンを使って、マトリックス Q を明示的に求められる。

入力パラメータ

vect	CHARACTER*1。'V' または 'N' でなければならない。 *vect* ='V' の場合は、マトリックス Q が明示的に返される。 *vect* ='N' の場合は、Q は返されない。
uplo	CHARACTER*1。'U' または 'L' でなければならない。 *uplo* ='U' の場合、*ab* に A の上三角部分を格納する。 *uplo* ='L' の場合、*ab* に A の下三角部分を格納する。
n	INTEGER。マトリックス A の次数 ($n \geq 0$)。
kd	INTEGER。A の優対角成分と劣対角成分の個数 ($kd \geq 0$)。
ab, work	COMPLEX (chbtrd の場合) DOUBLE COMPLEX (zhbtrd の場合) *ab* (*ldab*, *) は、(*uplo* での指定に従って) マトリックス A の上三角部分または下三角部分をバンド形式で格納する配列である。 *ab* の第 2 次元は、max(1, *n*) 以上でなければならない。 *work* (*) は、ワークスペース配列である。 *work* の次元は、max(1, *n*) 以上でなければならない。
ldab	INTEGER。*ab* の第 1 次元。*kd*+1 以上。

ldq INTEGER。*q* の第 1 次元。次の制約がある。
ldq ≥ max(1, *n*) (*vect* = 'V' の場合)
ldq ≥ 1 (*vect* = 'N' の場合)

出力パラメータ

ab 終了時に、配列 *ab* が上書きされる。

d, e REAL (chbtrd の場合)
DOUBLE PRECISION (zhbtrd の場合)
配列 :
d(*) には、マトリックス *T* の対角成分が格納される。
d の次元は、max(1, *n*) 以上でなければならない。
e(*) には、*T* の対角線外の成分が格納される。
e の次元は、max(1, *n*-1) 以上でなければならない。

q COMPLEX (chbtrd の場合)
DOUBLE COMPLEX (zhbtrd 場合)
配列 DIMENSION は、(*ldq*, *) である。
vect = 'N' の場合は、*q* は参照されない。
vect = 'V' の場合は、*q* には *n* × *n* のマトリックス *Q* が格納される。
q の第 2 次元は、次の条件を満たしていなければならない。
max(1, *n*) 以上 (*vect* = 'V' の場合)
1 以上 (*vect* = 'N' の場合)

info INTEGER
info = 0 の場合は、正常に終了する。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

計算で求めたマトリックス *T* は、マトリックス *A* + *E* の近似値である。$\|E\|_2 = c(n)\varepsilon \|A\|_2$ であり、$c(n)$ は漸増する (n の) の関数で、ε はマシン精度である。計算で求めたマトリックス *Q* は、正確なユニタリ・マトリックスとマトリックス *E* ($\|E\|_2 = O(\varepsilon)$) だけ異なる。

浮動小数演算のおおよその総数は、*vect* = 'N' の場合は $20n^2$*kd* である。*vect* = 'V' の場合は、$10n^3$*(*kd*-1)/*kd* だけ余分な演算が必要になる。

このルーチンの実数版は、?sbtrd である。

?sterf

QR アルゴリズムを使用して、実対称の三重対角マトリックスの固有値をすべて計算する。

```
call ssterf ( n, d, e, info )
call dsterf ( n, d, e, info )
```

説明

このルーチンは、実対称の三重対角マトリックス *T* (このマトリックスは、対称マトリックスまたはエルミート・マトリックスを対角形式に縮退させて得られる) の固有値をすべて計算する。このルーチンは、平方根なしの *QR* アルゴリズムを使用する。

固有値だけでなく固有ベクトルも必要な場合は、`?steqr` を使用する (5-134 ページを参照)。

入力パラメータ

n	`INTEGER`。マトリックス *T* の次数 ($n \geq 0$)。
d, e	`REAL` (`ssterf` の場合) `DOUBLE PRECISION` (`dsterf` の場合) 配列 : `d(*)` には、*T* の対角成分を格納する。 `d` のサイズは、max(1, *n*) 以上でなければならない。 `e(*)` には、*T* の非対角成分を格納する。 `e` のサイズは、max(1, *n*-1) 以上でなければならない。

出力パラメータ

d	*n* 個の固有値が昇順に格納される (*info* > 0 でない場合)。 *info* も参照のこと。
e	終了時に、この配列が上書きされる。*info* を参照。

info INTEGER

info = 0 の場合は、正常終了している。

info = *i* の場合は、このアルゴリズムは 30*n* 回の反復後にすべての固有値を見つけられず、*i* 個の非対角成分をゼロに収束できなかった。終了時には、直交性が *T* に近似したマトリックスの対角成分と非対角成分が、それぞれ *d* と *e* に格納される。

info = -*i* の場合は、*i* 番目のパラメータ値が不正である。

アプリケーション・ノート

計算された固有値と固有ベクトルは、厳密にはマトリックス $T + E$(ただし、$\|E\|_2 = O(\varepsilon)\,\|T\|_2$、ε はマシン精度) に対するものである。

正確な固有値を λ_i とし、それに対する計算値を μ_i とすると、その関係は次のようになる。

$$|\mu_i - \lambda_i| \le c(n)\varepsilon\,\|T\|_2$$

$c(n)$ は、n の漸増関数である。

浮動小数点演算の総実行数は、アルゴリズムの収束速度によって異なるが、一般に約 $14n^2$ である。

?steqr

三重対角形式に縮退させた対称マトリックスまたはエルミート・マトリックスのすべての固有値と固有ベクトルを計算する (*QR* アルゴリズム)。

```
call ssteqr ( compz, n, d, e, z, ldz, work, info )
call dsteqr ( compz, n, d, e, z, ldz, work, info )
call csteqr ( compz, n, d, e, z, ldz, work, info )
call zsteqr ( compz, n, d, e, z, ldz, work, info )
```

説明

このルーチンでは、実対称の三重対角マトリックス T のすべての固有値と (オプションで) 固有ベクトルを計算する。すなわち、スペクトル因子分解 $T = Z\Lambda Z^T$ を計算できる。

Λ は、対角成分が固有値 λ_i である対角マトリックスで、Z は列が固有ベクトルである直交マトリックスである。つまり、次のように表現できる。

$$Tz_i = \lambda_i z_i \ (i = 1, 2, \ldots, n)$$

(このルーチンでは、$\|z_i\|_2 = 1$ となるように、固有ベクトルを正規化する。)

このルーチンを使えば、三重対角形式 T ($A = QTQ^H$) に縮退させた任意の実対称 (または複素エルミート) マトリックス A の固有値と固有ベクトルの計算もできる。その場合、スペクトル因子分解は、$A = QTQ^H = (QZ)\Lambda(QZ)^H$ になる。`?steqr` を呼び出す場合は、事前に A を三重対角形式に縮退させ、次のルーチンを呼び出すことによってマトリックス Q を明示的に生成しなければならない。

	実マトリックスの場合	複素マトリックスの場合 :
フル格納形式	`?sytrd, ?orgtr`	`?hetrd, ?ungtr`
パックド格納形式	`?sptrd, ?opgtr`	`?hptrd, ?upgtr`
バンド格納形式	`?sbtrd (vect='V')`	`?hbtrd (vect='V')`

固有値だけが必要な場合は、`?sterf` (5-132 ページを参照) を呼び出した方が効率が良くなる。T が正値の場合は、`?steqr` よりも `?pteqr` (5-146 ページを参照) の方が、小さな固有値を正確に計算できる。

1 つのルーチン呼び出しでこの問題を解くには、分割統治ルーチンを使用する。すなわち、実対称マトリックスの場合は、`?stevd`、`?syevd`、`?spevd`、`?sbevd` のどれか、複素エルミート・マトリックスの場合は、`?heevd`、`?hpevd`、`?hbevd` のどれかを使用する。

入力パラメータ

compz CHARACTER*1。'N' 、'I'、'V' のいずれかでなければならない。
compz ='N' の場合、固有値だけを計算する。
compz ='I' の場合、三重対角マトリックス *T* の固有値と固有ベクトルを計算する。
compz ='V' の場合、*A* の固有値と固有ベクトルを計算する (配列 *z* には、あらかじめマトリックス *Q* を格納しておかなければならない)。

n INTEGER。マトリックス *T* の次数 ($n \geq 0$)。

d, e, work REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
配列 :
d(*) には、*T* の対角成分を格納する。
d の次元は、max(1, *n*) 以上でなければならない。
e(*) には、*T* の対角線外の成分が格納される。
e の次元は、max(1, *n*-1) 以上でなければならない。
work(*) は、ワークスペース配列である。
work の次元は、次の条件を満たさなければならない。
1 以上 (*compz* ='N' の場合)
max(1, 2**n*–2) 以上 (*compz* ='V' または 'I' の場合)

z REAL (ssteqr の場合)
DOUBLE PRECISION (dsteqr の場合)
COMPLEX (csteqr の場合)
DOUBLE COMPLEX (zsteqr の場合)
配列 DIMENSION は、(*ldz*, *) である。
compz ='N' または 'I' の場合、*z* を設定する必要はない。
vect ='V' の場合、*z* には $n \times n$ のマトリックス *Q* を格納しなければならない。
z の第 2 次元は、次の条件を満たさなければならない。
1 以上 (*compz* ='N' の場合)
max(1, *n*) 以上 (*compz* ='V' または 'I' の場合)
work (*lwork*) は、ワークスペース配列である。

ldz INTEGER。*z* の第 1 次元。次の制約がある。
ldz ≥ 1 (*compz* ='N' の場合)
ldz ≥ max(1, *n*) (*compz* ='V' または 'I' の場合)

出力パラメータ

d	*n* 個の固有値が、昇順に格納される。(*info* > 0 でない場合)。 *info* を参照。
e	終了時に、上書きされる。*info* を参照。
z	*info* = 0 の場合は、*n* 個の正規直交固有ベクトルが、列ごとに格納される (*i* 番目の列は、*i* 番目の固有値に対応する)。
info	INTEGER *info* = 0 の場合は、正常に終了する。 *info* = *i* の場合、このアルゴリズムで 30*n* 回の処理を繰り返した結果、すべての固有値を見つけられなかった (*i* 個の対角線外の成分がゼロに収束していない)。終了時に、*d* と *e* にそれぞれ、三重対角マトリックス (直交性が *T* に近似している) の対角成分と対角線外の成分が格納される。 *info* = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

計算で求めた固有値と固有ベクトルは、マトリックス $T + E(\|E\|_2 = O(\varepsilon)\ \|T\|_2$ であり、ε はマシン精度) の近似値である。

λ_i が正確な固有値であり、μ_i がそれに対応する計算で求めた値だとすると、次のようになる。

$$|\mu_i - \lambda_i| \le c(n)\varepsilon\ \|T\|_2$$

$c(n)$ は、(n の) 漸増する関数である。

z_i が正確な固有ベクトルであり、w_i がそれに対応する計算で求めた値だとすると、それらの間の角 $\theta(z_i, w_i)$ は、次のように制限される。

$$\theta(z_i, w_i) \le c(n)\varepsilon\ \|T\|_2 / \min_{i \ne j}|\lambda_i - \lambda_j|$$

浮動小数演算の総数は、このアルゴリズムの収束の速さによって決まる。通常は、ほぼ次の値になる。

$24n^2$ (*compz* = 'N' の場合)

$7n^3$ (複素数の場合は $14n^3$) (*compz* = 'V' または 'I' の場合)

?stedc

分割統治法を使用して、対称三重
マトリックスの固有値と固有ベクトルを
すべて求める。

```
call sstedc(compz, n, d, e, z, ldz, work, lwork, iwork, liwork,info)
call dstedc(compz, n, d, e, z, ldz, work, lwork, iwork, liwork,info)
call cstedc(compz, n, d, e, z, ldz, work, lwork, rwork, lrwork,iwork,
            liwork,info)
call zstedc(compz, n, d, e, z, ldz, work, lwork, rwork, lrwork,iwork,
            liwork,info)
```

説明

このルーチンは、分割統治法を使用して、対称三重マトリックスのすべての固有値、さらにオプションですべての固有ベクトルを計算する。フル形式またはバンド形式の実対称マトリックスまたは複素エルミート・マトリックスの固有ベクトルは、?sytrd/?hetrd、?sptrd/?hptrd、?sbtrd/?hbtrd のどれかを使用して、このマトリックスを三重対角形式に縮退させても得られる。

入力パラメータ

compz CHARACTER*1。'N'、'I'、'V' のどれかでなければならない。

compz ='N' の場合、このルーチンは固有値のみ計算する。

compz ='I' の場合、このルーチンは三重対角マトリックスの固有値と固有ベクトルを計算する。

compz ='V' の場合、このルーチンは元の対称 / エルミート・マトリックスの固有値と固有ベクトルを計算する。呼び出し時には、元のマトリックスを三重対角形式に縮退させるのに使用した直交 / ユニタリ・マトリックスが、配列 *z* に格納されていなければならない。

n INTEGER。対称三重対角マトリックスの次数 ($n \geq 0$)。

d, e, rwork
REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
配列 :
d(*) には、三重対角マトリックスの対角成分を格納する。1, *n* のサイズは、max(1, *n*) 以上でなければならない。
e(*) には、三重対角マトリックスの劣対角成分を格納する。*e* のサイズは、max(1, *n*-1) 以上でなければならない。
rwork(lrwork) は、複素数の場合にのみ使用するワークスペース配列。

z, work
REAL (sstedc の場合)
DOUBLE PRECISION (dstedc の場合)
COMPLEX (cstedc の場合)
DOUBLE COMPLEX (zstedc の場合)
配列 : *z*(*ldz, **), *work(*)*
compz ='V' の場合、呼び出し時に、元のマトリックスを三重対角形式に縮退させるのに使用した直交 / ユニタリ・マトリックスが、配列 *z* に格納されていなければならない。
z の第 2 次元のサイズは、max(1, *n*) 以上でなければならない。
work (*lwork*) は、ワークスペース配列。

ldz
INTEGER。*z* の第 1 次元のサイズ。次の制約がある。
ldz ≥ 1 (*compz* ='N' の場合)
ldz ≥ max(1, *n*) (*compz* ='V' または 'I' の場合)

lwork
INTEGER。配列 *work* の次元。
lwork に必要な値は、「アプリケーション・ノート」を参照。

lrwork
INTEGER。配列 *rwork* のサイズ (複素数の場合にのみ使用)。
lrwork に必要な値は、「アプリケーション・ノート」を参照。

iwork
INTEGER。ワークスペース配列 DIMENSION は *liwork*

liwork
INTEGER。配列 *iwork* のサイズ。
liwork に必要な値は、「アプリケーション・ノート」を参照。

出力パラメータ

d	昇順に並べた *n* 個の固有値 (*info* ≠ 0 でない場合)。*info* も参照のこと。
e	この配列は終了時に上書きされる。*info* を参照。
z	*info* = 0 の場合、*compz* = 'V' ならば、元の対称 / エルミート・マトリックスの正規直交固有ベクトルが *z* に格納され、*compz* = 'I' ならば、対称三重対角マトリックスの正規直交固有ベクトルが *z* に格納される。*compz* = 'N' ならば、*z* は参照されない。
work(1)	終了時に *info* = 0 ならば、*work(1)* に最適な *lwork* 値が返される。
rwork(1)	終了時に *info* = 0 ならば、*rwork(1)* に最適な *lrwork* 値が返される (複素数の場合のみ)。
iwork(1)	終了時に *info* = 0 ならば、*iwork(1)* に最適な *liwork* 値が返される。
info	INTEGER *info* = 0 の場合は、正常終了している。 *info* = -*i* の場合は、*i* 番目のパラメータ値が不正である。*info* = *i* の場合は、このアルゴリズムが、*i*/(*n*+1) から mod(*i*, *n*+1) までの行と列からなる部分マトリックスの操作中に、固有値の計算に失敗した。

アプリケーション・ノート

ワークスペース配列に必要なサイズは次のようになる。

sstedc/dstedc の場合 :

compz = 'N' または $n \le 1$ の場合、*lwork* は 1 以上でなければならない。
compz = 'V' および $n > 1$ の場合、*lwork* は $(1 + 3n + 2n \cdot \lg n + 3n^2)$ 以上でなければならない。$\lg(n)$ は $2^k \ge n$ となるような最小の整数 k

compz = 'I' および $n > 1$ の場合、*lwork* は $(1 + 4n + n^2)$ 以上でなければならない。

compz = 'N' または $n \le 1$ の場合、*liwork* は 1 以上でなければならない。
compz = 'V' および $n > 1$ の場合、*liwork* は $(6 + 6n + 5n \cdot \lg n)$ 以上でなければならない。
compz = 'I' および $n > 1$ の場合、*liwork* は $(3 + 5n)$ 以上でなければならない。

cstedc/zstedc の場合 :

compz = 'N' または 'I' の場合、または $n \le 1$ の場合、*lwork* は 1 以上でなければならない。
compz = 'V' および $n > 1$ の場合、*lwork* は n^2 以上でなければならない。

compz = 'N' または $n \le 1$ の場合、*lwork* は 1 以上でなければならない。
compz = 'V' および $n > 1$ の場合、*lwork* は $(1 + 3n + 2n \cdot \lg n + 3n^2)$ 以上でなければならない。$\lg(n)$ は $2^k \ge n$ となるような最小の整数 k

compz = 'I' および $n > 1$ の場合、*lwork* は $(1 + 4n + 2n^2)$ 以上でなければならない。

複素数の場合に *liwork* に必要な値は、実数の場合と同じである。

?stegr

実対称の三重対角マトリックスの固有値と
固有ベクトルを選択的に計算する。

```
call sstegr (jobz, range, n, d, e, vl, vu, il, iu, abstol, m, w, z,
             ldz, isuppz, work, lwork, iwork, liwork, info)
call dstegr (jobz, range, n, d, e, vl, vu, il, iu, abstol, m, w, z,
             ldz, isuppz, work, lwork, iwork, liwork, info)
call cstegr (jobz, range, n, d, e, vl, vu, il, iu, abstol, m, w, z,
             ldz, isuppz, work, lwork, iwork, liwork, info)
call zstegr (jobz, range, n, d, e, vl, vu, il, iu, abstol, m, w, z,
             ldz, isuppz, work, lwork, iwork, liwork, info)
```

説明

このルーチンは、実対称の三重対角マトリックス T の固有値、さらにオプションで固有ベクトルを選択的に計算する。目的の固有値と固有ベクトルは、固有値の範囲で選択するか、そのインデックスの範囲で選択する。固有値は *dqds* アルゴリズムで計算するが、直交固有ベクトルは種々の「良好な」LDL^T 表現 (比較的安定した表現とも呼ばれる) から計算される。グラム - シュミットの直交化はできる限り避ける。アルゴリズムの各ステップを詳しく説明すると次のようになる。T のまだ縮退されていないブロックに対して、

(a) $L_i D_i L_i^T$ が比較的安定な表現になるように、$T - \sigma_i = L_i D_i L_i^T$ を計算する。

(b) *dqds* アルゴリズムによって、$L_i D_i L_i^T$ の固有値 λ_j を高い相対精度で計算する。

(c) 値が近い固有値からなるクラスタが存在する場合は、そのクラスタに近い σ_i を「選択」し、ステップ (a) に戻る。

(d) $L_i D_i L_i^T$ の近似的な固有値 λ_j に対する固有ベクトルを、階数表示のツイスト分解によって求める。

出力値に必要な精度は、入力パラメータ *abstol* で指定できる。

入力パラメータ

jobz
CHARACTER*1。'N' または 'V' でなければならない。
job = 'N' の場合、固有値のみ計算する。
job = 'V' の場合、固有値と固有ベクトルを計算する。

range
CHARACTER*1。'A'、'V'、'I' のどれかでなければならない。
range = 'A' の場合、このルーチンはすべての固有値を計算する。
range = 'V' の場合、このルーチンは半開区間内の固有値 λ_i ($vl < \lambda_i \le vu$) を計算する。
range = 'I' の場合、このルーチンはインデックス *il* ～ *iu* の固有値を計算する。

n
INTEGER。マトリックス T の次数 ($n \ge 0$)。

d, e, work
REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
配列 :
d(*) には、T の対角成分を格納する。
d のサイズは、max(1, *n*) 以上でなければならない。
e(*) の 1 ～ *n*-1 の成分に、T の劣対角成分を格納する。
e(*n*) には値を設定しなくてよい。
e のサイズは、max(1, *n*) 以上でなければならない。
work(*lwork*) は、ワークスペース配列。

vl, vu
REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
range = 'V' の場合、求める固有値区間の下限値と上限値。
制約 : $vl < vu$
range = 'A' または 'I' の場合、*vl* と *vu* は参照されない。

il, iu
INTEGER
range = 'I' の場合、固有値を昇順に並べたときのインデックスの最小値と最大値とで、求める固有値を指定する。
制約 : $n > 0$ の場合は $1 \le il \le iu \le n$、$n = 0$ の場合は $il = 1$ かつ $iu = 0$

range = 'A' または 'V' の場合、il と iu は参照されない。

abstol
REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
各固有値と固有ベクトルに許される絶対許容値。
jobz = 'V' の場合、出力される固有値と固有ベクトルの誤差ノルムが abstol 以内になる。また、異なる固有ベクトル間のドット積も abstol 以内になる。
abstol < $n\varepsilon\|T\|_1$ の場合は、$n\varepsilon\|T\|_1$ がかわりに使用される (ε はマシン精度)。固有値は、abstol とは無関係に、$\varepsilon\|T\|_1$ の精度で計算される。高い相対精度が必要な場合は、abstol を ?lamch (「安全な最小値」) に設定する。

ldz
INTEGER。出力配列 z の第 1 次元のサイズ。次の制約がある。
ldz ≥ 1 (jobz = 'N' の場合)
ldz ≥ max(1, n) (jobz = 'V' の場合)

lwork
INTEGER。配列 work のサイズ。
lwork ≥ max(1, 18n)

iwork
INTEGER
ワークスペース配列 DIMENSION は liwork

liwork
INTEGER。配列 iwork のサイズ。
lwork ≥ max(1, 10n)

出力パラメータ

d, e
出力時に d と e が上書きされる。

m
INTEGER。見つかった固有値の個数、0 ≤ m ≤ n。
range = 'A' の場合は m = n、range = 'I' の場合は m = iu-il+1

w
REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
配列 DIMENSION は max(1, n) 以上。
選択された固有値が昇順に w(1) ～ w(m) に格納される。

z
REAL (sstegr の場合)
DOUBLE PRECISION (dstegr の場合)
COMPLEX (cstegr の場合)
DOUBLE COMPLEX (zstegr の場合)
配列 *z*(*ldz*, *)、*z* の第 2 次元のサイズは max(1, *m*) 以上でなければならない。

jobz = 'V' の場合、*info* = 0 ならば、マトリックス *T* の正規直交固有ベクトルのうち、選択された固有値に対応するものが、*z* の最初の *m* 列に格納される。すなわち、*w*(*i*) に対応する固有ベクトルが、*z* の *i* 番目の列に入る。*jobz* = 'N' の場合、*z* は参照されない。

注 : 配列 *z* には、max(1, *m*) 以上の列が提供されなければならない。*range* = 'V' の場合は、*m* の正確な値が事前にわからないため、上限値を使用する必要がある。

isuppz
INTEGER
配列 DIMENSION は 2*max(1, *m*) 以上。

z に入っている固有ベクトルのサポート情報、すなわち *z* に入っている非ゼロ成分を示すインデックス。*i* 番目の固有ベクトルは、*isuppz*(2*i*-1) から *isuppz*(2*i*) までの成分のみ非ゼロである。

work(1)
終了時に、*info* = 0 ならば、*lwork* に必要な最小値が *work(1)* に返される。

iwork(1)
終了時に、*info* = 0 ならば、*liwork* に必要な最小値が *iwork(1)* に返される。

info
INTEGER
info = 0 の場合は、正常終了している。
info = -*i* の場合は、*i* 番目のパラメータ値が不正である。

info = 1 の場合、slarre で内部エラーが発生した。
info = 2 の場合、?larrv で内部エラーが発生した。

アプリケーション・ノート

現時点での ?stegr は、O (n^2) 時間内で *T* の**すべての**固有値と固有ベクトルを見つけ出すように設定されている。すなわち、*range* = 'A' のみサポートしている。

現在、上記のステップ (c) で適切な σ_i が見つからない場合、固有値同士が近い値ならば、?stein が変形グラム - シュミットを呼び出す。

?stegr は、無限大と NaN の処理方法が、IEEE-754 の浮動小数点標準に準拠しているマシンのみ正しく機能する。通常の ?stegr 実行で無限大と NaN が発生するため、IEEE-754 標準に準拠しない環境で実行すると、浮動小数点例外によって異常終了する可能性がある。

?pteqr

実対称正値三重対角マトリックスの
すべての固有値および (オプションで)
固有ベクトルを計算する。

```
call spteqr ( compz, n, d, e, z, ldz, work, info )
call dpteqr ( compz, n, d, e, z, ldz, work, info )
call cpteqr ( compz, n, d, e, z, ldz, work, info )
call zpteqr ( compz, n, d, e, z, ldz, work, info )
```

説明

このルーチンでは、実対称正値三重対角マトリックス T のすべての固有値と (オプションで) 固有ベクトルを計算する。つまり、このルーチンでは、スペクトル因子分解 $T = Z\Lambda Z^T$ を計算できる。
Λ は、対角成分が固有値 λ_i である対角マトリックスで、Z は列が固有ベクトルである直交マトリックスである。つまり、次のように表現できる。

$$Tz_i = \lambda_i z_i \ (i = 1, 2, \ldots, n)$$

(このルーチンでは、$\|z_i\|_2 = 1$ となるように、固有ベクトルを正規化する。)

このルーチンを使えば、三重対角形式 T ($A = QTQ^H$) に縮退させた正値の実対称 (または複素エルミート) マトリックス A の固有値と固有ベクトルの計算もできる。その場合、スペクトル因子分解は、
$A = QTQ^H = (QZ)\Lambda(QZ)^H$ になる。`?pteqr` を呼び出す場合は、事前に A を三重対角形式に縮退させ、次のルーチンを呼び出すことによってマトリックス Q を明示的に生成しなければならない。

	実マトリックスの場合	複素マトリックスの場合
フル格納形式	`?sytrd, ?orgtr`	`?hetrd, ?ungtr`
パックド格納形式	`?sptrd, ?opgtr`	`?hptrd, ?upgtr`
バンド格納形式	`?sbtrd (vect='V')`	`?hbtrd (vect='V')`

このルーチンではまず、T を LDL^H として因子分解する (L は下側の単位二重対角マトリックスで、D は対角マトリックスである)。次に二重対角マトリックス $B = LD^{1/2}$ を求めた後、`?bdsqr` を呼び出して B の特異値 (T の固有値と同じ) を求める。

入力パラメータ

compz CHARACTER*1。'N'、'I'、'V' のいずれかでなければならない。
compz ='N' の場合、固有値だけを計算する。
compz ='I' の場合、三重対角マトリックス T の固有値と固有ベクトルを計算する。
compz ='V' の場合、A の固有値と固有ベクトルを計算する (配列 *z* には、あらかじめマトリックス Q を格納しておかなければならない)。

n INTEGER。マトリックス T の次数 ($n \geq 0$)。

d, e, work REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
配列 :
d(*) には、T の対角成分を格納する。
d の次元は、max(1, *n*) 以上でなければならない。
e(*) には、T の対角線外の成分が格納される。
e の次元は、max(1, *n*-1) 以上でなければならない。
work(*) は、ワークスペース配列である。
work の次元は、次の条件を満たさなければならない。
1 以上 (*compz* ='N' の場合)
max(1, 4*n-4) 以上 (*compz* ='V' または 'I' の場合)

z REAL (spteqr の場合)
DOUBLE PRECISION (dpteqr の場合)
COMPLEX (cpteqr の場合)
DOUBLE COMPLEX (zpteqr の場合)
配列 DIMENSION は、(*ldz*, *) である。
compz ='N' または 'I' の場合、*z* を設定する必要はない。
vect ='V' の場合、*z* には $n \times n$ のマトリックス Q を格納しなければならない。
1 以上 (*compz* ='N' の場合)
max(1, *n*) 以上 (*compz* ='V' または 'I' の場合)

ldz INTEGER。*z* の第 1 次元。次の制約がある。
ldz ≥ 1 (*compz* ='N' の場合)
ldz $\geq$ max(1, *n*) (*compz* ='V' または 'I' の場合)

出力パラメータ

d	*n* 個の固有値が、降順に格納される。(*info* > 0 でない場合)。 *info* を参照。
e	終了時に、上書きされる。
z	*info* = 0 の場合は、*n* 個の正規直交固有ベクトルが、列ごとに格納される (*i* 番目の列は、*i* 番目の固有値に対応する)。
info	INTEGER *info* = 0 の場合は、正常に終了する。 *info* = *i* の場合、先頭から次数 *i* の小行列 (したがって *T* そのもの) は、正値ではない。 *info* = *n* + *i* の場合は、特異値を計算するためのアルゴリズムが収束していない。つまり、*i* 個の対角線外の成分がゼロに収束していない。 *info* = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

λ_i が正確な固有値であり、μ_i がそれに対応する計算で求めた値だとすると、次のようになる。

$$|\mu_i - \lambda_i| \le c(n)\varepsilon K\lambda_i$$

$c(n)$ は (n の) 漸増する関数、ε はマシン精度、$\|DTD\|_2 \|(DTD)^{-1}\|_2$ (D は、$d_{ii} = t_{ii}^{-1/2}$ の対角成分) である。

z_i が正確な固有ベクトルであり、w_i がそれに対応する計算で求めた値だとすると、それらの間の角度 $\theta(z_i, w_i)$ には、次の制約がある。

$$\theta(u_i, w_i) \le c(n)\varepsilon K / \min_{i\ne j}(|\lambda_i - \lambda_j|/|\lambda_i + \lambda_j|)$$

$\min_{i\ne j}(|\lambda_i - \lambda_j|/|\lambda_i + \lambda_j|)$ は、λ_i ともう一方の固有値との間の**相対誤差**である。

浮動小数演算の総数は、このアルゴリズムの収束の速さによって決まる。通常は、ほぼ次の値になる。

$30n^2$ (*compz* = 'N' の場合)

$6n^3$ (複素数の場合は $12n^3$) (*compz* = 'V' または 'I' の場合)

?stebz

実対称の三重対角マトリックスの
指定された固有値を二分法で計算する。

```
call sstebz (range, order, n, vl, vu, il, iu, abstol, d, e, m, nsplit,
             w, iblock, isplit, work, iwork, info)
call dstebz (range, order, n, vl, vu, il, iu, abstol, d, e, m, nsplit,
             w, iblock, isplit, work, iwork, info)
```

説明

このルーチンでは、実対称の三重対角マトリックス T の固有値の一部 (または全部) を、二分法によって計算する。このルーチンでは、ゼロまたは無視できる程度の対角線外の成分を検索し、T をブロック - 対角形式 $T = \mathrm{diag}(T_1, T_2, ...)$ に分解できるかどうかを調べる。それが終わったら、T_i の各ブロックに対して二分法を実行し、計算で求めた各固有値のブロックのインデックスを返す。そのため、?stein を呼び出した後に、続けてこのブロック構造を利用できる。

入力パラメータ

range CHARACTER*1。'A'、'V'、'I' のいずれかでなければならない。
range ='A' の場合、すべての固有値を計算する。
range ='V' の場合、半開区間 $vl < \lambda_i \le vu$ における固有値 λ_i を計算する。
range ='I' の場合、インデックスが *il* ～ *iu* である固有値を計算する。

order CHARACTER*1。'B' または 'E' でなければならない。
order ='B' の場合、一連の固有値は、分解された各ブロック内で最小のものから昇順に配置される。
order ='E' の場合、固有値がマトリックス全体で最小のものから昇順に配置される。

n INTEGER。マトリックス T の次数 ($n \ge 0$)。

vl, vu
REAL (sstebz の場合)
DOUBLE PRECISION (dstebz の場合)
range = 'V' の場合、半開区間 $vl < \lambda_i \le vu$ における固有値 λ_i を計算する。
range = 'A' または 'I' の場合、*vl* と *vu* は参照されない。

il, iu
INTEGER。次の制約がある。$1 \le il \le iu \le n$
range = 'I' の場合、、$il \le i \le iu$ (固有値 λ_i は昇順とする) となるような固有値 λ_i を計算する。
range = 'A' または 'V' の場合、*il* と *iu* は参照されない。

abstol
REAL (sstebz の場合)
DOUBLE PRECISION (dstebz の場合)
各固有値に対する絶対許容値は、必ず指定しなければならない。固有値 (または集積値) は、幅 *abstol* の区間内に存在している場合に、収束しているものと見なされる。*abstol* ≤ 0.0 の場合は、許容値は $\varepsilon\|T\|_1$ (ε はマシン精度) になる。

d, e
REAL (sstebz/f08jje の場合)
DOUBLE PRECISION (dstebz/f08jjf の場合)
配列 :
d(*) には、*T* の対角成分を格納する。
d の次元は、max(1, *n*) 以上でなければならない。
e(*) には、*T* の対角線外の成分が格納される。
e の次元は、max(1, *n*-1) 以上でなければならない。

iwork
INTEGER。ワークスペース。
配列 DIMENSION は、max(1, 3*n*) 以上。

出力パラメータ

m
INTEGER。見つかった固有値の実際の個数。

nsplit
INTEGER。*T* 内で検出された対角ブロックの個数。

w
REAL (sstebz の場合)
DOUBLE PRECISION (dstebz の場合)
配列 DIMENSION は、max(1, *n*) 以上。
計算で求めた固有値が、*w(1)* ～ *w(m)* に格納される。

iblock,isplit INTEGER

配列 DIMENSION は、max(1, *n*) 以上。
正値 *iblock(i)* は、*w(i)* に格納されている固有値のブロック番号である (*info* を参照)。
isplit の先頭から *nsplit* 個の成分には、ブロック T_1 に 1 ～ *isplit(1)* のマトリックスを格納し、ブロック T_2 に *isplit(1)*+1 ～ *isplit(2)* のマトリックスを格納するという具合に、*T* をブロック T_i に分割した点が格納される。

info INTEGER

info = 0 の場合は、正常に終了する。
info = 1 の場合、*range* = 'A' または 'V' のときは、所定の精度で計算できなかった固有値がある。
iblock(i) < 0 の場合は、*w(i)* に格納されている固有値が収束していない。
info = 2 の場合、*range* = 'I' のときは、計算できなかった固有値がある。*range* = 'A' と指定して、このルーチンを再度呼び出すようにする。
info = 3 の場合は、次のとおりである。

range = 'A' または 'V' のときは、*info* = 1 の場合と同じである。
range = 'I' のときは、*info* = 2 の場合と同じである。

info = 4 の場合は、固有値を計算することはできない。コンピュータ上で、浮動小数演算が期待通りに動作していない。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

T の固有値が比較的高い精度で計算される場合は、その絶対値の変動が大きくても、標準の *QR* 法を使う場合などに比べて、小さな固有値がすべて正確に計算される。ただし、(このルーチンが呼び出される前に実行される) 三重対角形式への縮退によって、固有値の絶対値が大きく変動するような場合には、元のマトリックスの固有値が小さいときに相対精度が低下する場合もある。

?stein

実対称の三重対角マトリックスの指定された固有値に対応する固有ベクトルを計算する。

```
call sstein ( n, d, e, m, w, iblock, isplit, z, ldz, work, iwork,
              ifailv, info )
call dstein ( n, d, e, m, w, iblock, isplit, z, ldz, work, iwork,
              ifailv, info )
call cstein ( n, d, e, m, w, iblock, isplit, z, ldz, work, iwork,
              ifailv, info )
call zstein ( n, d, e, m, w, iblock, isplit, z, ldz, work, iwork,
              ifailv, info )
```

説明

このルーチンでは、実対称の三重対角マトリックス T の指定された固有値に対応する固有ベクトルを、反転 (逆マトリックスの計算) を繰り返して求める。このルーチンは特に、*order* ='B' と指定して ?stebz を呼び出し、指定の固有値を計算した後で使うように設計されている。ただし、このルーチンは他のルーチンを使って固有値を計算した後に使用できる。?stebz の後でこのルーチンを使う場合は、各 T_i ブロックごとに反転を個々に繰り返して、そのブロック構造を利用でき、マトリックス T の全体を使うよりも、効率が良くなる。

対称マトリックスまたはエルミート・マトリックス A の全体を三重対角形式に縮退させて T を求めた場合は、ルーチン ?ormtr または ?opmtr (実数の場合)、あるいは ?unmtr または ?upmtr (複素数の場合) を呼び出し、T の固有ベクトルを A の固有ベクトルに変換できる。

入力パラメータ

n	INTEGER。マトリックス T の次数 ($n \geq 0$)。
m	INTEGER。返すべき固有ベクトルの個数。

d, e, w
REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
配列 :
d(*) には、*T* の対角成分を格納する。
d の次元は、max(1, *n*) 以上でなければならない。
e(*) には、*T* の対角線外の成分が格納される。
e の次元は、max(1, *n*-1) 以上でなければならない。
w(*) (*w*(1) ～ *w*(m)) に、?stebz (5-149 ページを参照) から返された *T* の固有値を格納する。T_1 の固有値、T_2 の固有値の順番 (降順ではない) で指定する。次の制約がある。
iblock(*i*) = *iblock*(*i*+1) の場合は、
w(*i*) ≤ *w*(*i*+1)
w の次元は、max(1, *n*) 以上でなければならない。

iblock, isplit
INTEGER
配列 DIMENSION は、max(1, *n*) 以上。
配列 *iblock* と *isplit* は、*order* ='B' と指定して ?stebz を呼び出し、その出力として返された配列である。
order ='B' と指定して *?stebz* を呼び出さなかった場合は、*iblock* の全成分に 1 を、*isplit*(1) に *n* を設定する。

ldz
INTEGER。出力配列 *z* の第 1 次元。*ldz* ≥ max(1, *n*)

work
REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合) ワークスペース配列 DIMENSION は、max(1, 5*n*) 以上。

iwork
INTEGER
ワークスペース配列 DIMENSION は、max(1, *n*) 以上。

出力パラメータ

z
REAL (sstein の場合)
DOUBLE PRECISION (dstein の場合)
COMPLEX (cstein の場合)
DOUBLE COMPLEX (zstein の場合)

配列 DIMENSION (*ldz*, *)
info = 0 の場合は、*m* 個の正規直交固有ベクトル格納が列ごとに格納される (*i* 番目の列は、*i* 番目に指定した固有値に対応する)。

ifailv INTEGER。配列 DIMENSION は max(1, *m*) 以上。
ifailv = *i* > 0 の場合は、収束しなかったすべての固有ベクトルのインデックスが、*ifailv* の先頭から *i* 個の成分に格納される。

info INTEGER
info = 0 の場合は、正常に終了する。
info = *i* の場合は、*i* 個の固有ベクトル (パラメータ *ifailv* によって指定) が、5 回の繰り返しによっても収束しない。現時点の繰り返し計算の結果は、配列 *z* の対応する列に格納される。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

計算で求めた各固有ベクトル z_i は、マトリックス $T + E_i$ の正確な固有ベクトルである ($\|E_i\|_2 = O(\varepsilon)\ \|T\|_2$)。ただし、このルーチンで求める一連の固有ベクトルは、?steqr によって求める固有ベクトルと比較すると、直交性の面で精度が低くなる場合もある。

?disna

対称 / エルミート・マトリックスの固有ベクトル、または一般マトリックスの左または右の特異ベクトルに対して、条件数の逆数を計算する。

```
call sdisna (job, m, n, d, sep, info)
call ddisna (job, m, n, d, sep, info)
```

説明

このルーチンは、実対称マトリックスまたは複素エルミート・マトリックスの固有ベクトル、または *m* × *n* の一般マトリックスの左または右の特異ベクトルに対して、条件数の逆数を計算する。

条件数の逆数とは、最も近い固有値 (または特異値) 間の「ギャップ」である。

計算された *i* 番目のベクトルの誤差範囲 (ラジアン単位の角度で測定) は、次のように表される。

slamch('E') * (*anorm* / *sep*(*i*))

ここで、*anorm* = $\|A\|_2$ = max(|*d*(j)|)。誤差範囲を限定するには、*sep*(*i*) が slamch('E')*anorm より小さくてはならない。

?disna は、汎用対称固有値問題における固有ベクトルの誤差範囲の計算にも使用できる。

入力パラメータ

job
CHARACTER*1。'E'、'L'、'R' のどれかでなければならない。
どの問題に対して条件数の逆数を計算するか指定する。
job ='E': 対称 / エルミート・マトリックスの固有ベクトル
job ='L': 一般マトリックスの左特異ベクトル
job ='R': 一般マトリックスの右特異ベクトル

m
INTEGER。マトリックスの行数 (*m* ≥ 0)。

n
INTEGER。*job* ='L' または 'R' の場合は、マトリックスの列数 (*n* ≥ 0)。*job* ='E' の場合は、無視される。

d
REAL (sdisna の場合)
DOUBLE PRECISION (ddisna の場合)
配列。サイズは、*job* ='E' の場合は max(1, *m*) 以上、*job* ='L' または 'R' の場合は max(1, min(*m*, *n*)) 以上。この配列には、マトリックスの固有値 (*job* ='E' の場合) または特異値 (*job* ='L' または 'R' の場合) が、昇順または降順に格納されていなければならない。特異値は、非負でなければならない。

出力パラメータ

sep
REAL (sdisna の場合)
DOUBLE PRECISION (ddisna の場合)
配列。サイズは、*job* = 'E' の場合は max(1, *m*) 以上、*job* = 'L' または 'R' の場合は max(1, min(*m*, *n*)) 以上。ベクトルの条件数の逆数。

info
INTEGER
info = 0 の場合は、正常終了している。
info = -*i* の場合は、*i* 番目のパラメータ値が不正である。

汎用対称固有値問題

汎用対称固有値問題は、「次のいずれかの方程式を満たすような固有値 λ とそれに対応する固有ベクトル z を求めること」である。

$$Az = \lambda Bz、ABz = \lambda z、または\ BAz = \lambda z$$

A は $n \times n$ の対称マトリックスまたはエルミート・マトリックス、B は $n \times n$ の正値対称または正値エルミートのマトリックスである。

このような固有値問題では、実数型の固有値に対応する実数型の固有ベクトルが n 個存在している (エルミート・マトリックス A と B が複素型の場合でも実数型になる)。

ここで説明する一連のルーチンを使えば、上記の一般化された問題を標準の対称固有値問題 $Cy = \lambda y$ に縮退させられる。縮退させた後の標準の対称固有値問題は、この章ですでに説明した一連の LAPACK ルーチン (5-101 ページを参照) を呼び出して解くのが可能である。

種々のルーチンで、マトリックスを従来の形式またはパックド形式で格納できる。縮退の前に、まず正値マトリックス B を ?potrf または ?pptrf を使用して因子分解する必要がある。
バンド・マトリックス A と B の縮退ルーチンは、分割コルスキー因子分解を使用するが、このための専用ルーチン ?pbstf が提供されている。このルーチンを使用することにより、マトリックス C の生成に必要な作業量が半分で済む。

表 5-4　汎用固有値問題から標準固有値問題への縮退用の計算ルーチン

マトリックスのタイプ	標準問題への縮退 (フル格納形式)	標準問題への縮退 (パックド格納形式)	標準問題への縮退 (バンド・マトリックス)	バンド・マトリックスの因子分解
実対称マトリックス	?sygst	?spgst	?sbgst	?pbstf
複素エルミートマトリックス	?hegst	?hpgst	?hbgst	?pbstf

?sygst

実対称の汎用固有値問題を標準形式に
縮退させる。

```
call ssygst ( itype, uplo, n, a, lda, b, ldb, info )
call dsygst ( itype, uplo, n, a, lda, b, ldb, info )
```

説明

このルーチンでは、次の実対称の汎用固有値問題を縮退させ、

$$Az = \lambda Bz、ABz = \lambda z、または\ BAz = \lambda z$$

標準形式 $Cy = \lambda y$ にする。A は実対称マトリックスで、B は正値実対称マトリックスである。このルーチンを呼び出す前に、?potrf を呼び出して、コルスキー因子分解 $B = U^TU$ または $B = LL^T$ (4-14 ページを参照) を計算する。

入力パラメータ

itype　INTEGER。1、2 または 3 でなければならない。
itype = 1 の場合は、汎用固有値問題 $Az = \lambda Bz$ で、
　uplo = 'U' の場合は $C = U^{-T}AU^{-1}$、$z = U^{-1}y$
　uplo = 'L' の場合は $C = L^{-1}AL^{-T}$、$z = L^{-T}y$
itype = 2 の場合は、の場合は、汎用固有値問題 $ABz = \lambda z$; で、
　uplo = 'U' の場合は $C = UAU^T$、$z = U^{-1}y$
　uplo = 'L' の場合は $C = L^TAL$、$z = L^{-T}y$
itype = 3 の場合は、汎用固有値問題 $BAz = \lambda z$ で、
　uplo = 'U' の場合は $C = UAU^T$、$z = U^Ty$
　uplo = 'L' の場合は $C = L^TAL$、$z = Ly$

uplo　CHARACTER*1。'U' または 'L' でなければならない。
uplo = 'U' の場合は、配列 *a* に A の上三角マトリックスを格納する。B は、$B = U^TU$ に因子分解した形式で指定しなければならない。
uplo = 'L' の場合は、配列 *a* に A の下三角マトリックスを格納する。B は、$B = LL^T$ に因子分解した形式で指定しなければならない。

n　INTEGER。マトリックス A と B の次数 ($n \geq 0$)

a, b	REAL (ssygst の場合) DOUBLE PRECISION (dsygst の場合) 配列 : *a*(*lda*, *) には、*A* の上三角マトリックスまたは下三角マトリックスを格納する。 *a* の第 2 次元は、max(1, *n*) 以上でなければならない。 *b*(*ldb*, *) には、次のコルスキー因子分解されたマトリックス *B* を格納する。 $B = U^TU$ または $B = LL^T$ (?potrf から返されたマトリックス) *b* の第 2 次元は、max(1, *n*) 以上でなければならない。
lda	INTEGER。*a* の第 1 次元。max(1, *n*) 以上。
ldb	INTEGER。*b* の第 1 次元。max(1, *n*) 以上。

出力パラメータ

a	引数 *itype* と *uplo* の指定に従って、*A* の上三角マトリックスまたは下三角マトリックスが、*C* の上三角マトリックスまたは下三角マトリックスによって上書きされる。
info	INTEGER *info* = 0 の場合は、正常に終了する。 *info* = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

縮退させたマトリックス *C* を得るのは、安定した処理である。ただし、B^{-1} (*itype* = 1 の場合) または *B* (*itype* = 2 または 3 の場合) が、暗黙的に乗算される。このルーチンを元の問題の固有値および固有ベクトルを計算する際の 1 ステップとして使う場合は、反転処理に関して *B* の条件が悪いと、精度が大きく低下するときがある。

浮動小数演算のおおよその総数は、n^3 である。

?hegst

複素エルミートの汎用固有値問題を
標準形式に縮退させる。

```
call chegst ( itype, uplo, n, a, lda, b, ldb, info )
call zhegst ( itype, uplo, n, a, lda, b, ldb, info )
```

説明

このルーチンでは、次の複素エルミートの汎用固有値問題を、

$$Az = \lambda Bz、ABz = \lambda z、または\ BAz = \lambda z$$

標準形式 $Cy = \lambda y$ に縮退させる。マトリックス A は複素エルミート・マトリックスで、B は正値複素エルミート・マトリックスである。このルーチンを呼び出す前に、?potrf を呼び出し、コルスキー因子分解 $B = U^H U$ または $B = LL^H$ (4-14 ページを参照) を計算する。

入力パラメータ

itype　INTEGER。1、2 または 3 でなければならない。
itype = 1 の場合は、汎用固有値問題 $Az = \lambda Bz$ で、
　uplo = 'U' の場合は $C = U^{-H}AU^{-1}$、$z = U^{-1}y$
　uplo = 'L' の場合は $C = L^{-1}AL^{-H}$、$z = L^{-H}y$
itype = 2 の場合は、汎用固有値問題 $ABz = \lambda z$ で、
　uplo = 'U' の場合は $C = UAU^H$、$z = U^{-1}y$
　uplo = 'L' の場合は $C = L^H AL$、$z = L^{-H}y$
itype = 3 の場合は、汎用固有値問題 $BAz = \lambda z$ で、
　uplo = 'U' の場合は $C = UAU^H$、$z = U^H y$
　uplo = 'L' の場合は $C = L^H AL$、$z = Ly$

uplo　CHARACTER*1。'U' または 'L' でなければならない。
uplo = 'U' の場合は、配列 *a* に A の上三角マトリックスを格納する。B は、$B = U^H U$ に因子分解した形式で指定しなければならない。
uplo = 'L' の場合は、配列 *a* に A の下三角マトリックスを格納する。B は、$B = LL^H$ に因子分解した形式で指定しなければならない。

n	INTEGER。マトリックス A と B の次数 ($n \geq 0$)。
a, b	COMPLEX (chegst の場合) DOUBLE COMPLEX (zhegst の場合) 配列 : *a(lda, *)* には、A の上三角マトリックスまたは下三角マトリックスを格納する。 *a* の第 2 次元は、max(1, *n*) 以上でなければならない。 *b(ldb, *)* には、次のコルスキー因子分解されたマトリックス B を格納する。 $B = U^H U$ または $B = LL^H$ (?potrf から返されたマトリックス) *b* の第 2 次元は、max(1, *n*) 以上でなければならない。
lda	INTEGER。*a* の第 1 次元。max(1, *n*) 以上。
ldb	INTEGER。*b* の第 1 次元。max(1, *n*) 以上。

出力パラメータ

a	引数 *itype* と *uplo* の指定に従って、A の上三角マトリックスまたは下三角マトリックスが、C の上三角マトリックスまたは下三角マトリックスによって上書きされる。
info	INTEGER *info* = 0 の場合は、正常に終了する。 *info* = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

縮退させたマトリックス C を得るのは、安定した処理である。ただし、B^{-1} (*itype* = 1 の場合) または B (*itype* = 2 または 3 の場合) が、暗黙的に乗算される。このルーチンを元の問題の固有値と固有ベクトルを計算する際の 1 ステップとして使う場合は、反転処理に関して B の条件が悪いと、精度が大きく低下するときがある。

浮動小数演算のおおよその総数は、n^3 である。

?spgst

パックド格納形式の実対称の汎用固有値問題を標準形式に縮退させる。

```
call sspgst ( itype, uplo, n, ap, bp, info )
call dspgst ( itype, uplo, n, ap, bp, info )
```

説明

このルーチンでは、パックド格納形式を使って、次の汎用対称固有値問題を、

$$Az = \lambda Bz、ABz = \lambda z、または BAz = \lambda z$$

標準形式 $Cy = \lambda y$ に縮退させる。A は実対称マトリックスで、B は正値実対称マトリックスである。このルーチンを呼び出す前に、?pptrf を呼び出し、コルスキー因子分解 $B = U^TU$ または $B = LL^T$ (4-16 ページを参照) を計算する。

入力パラメータ

itype INTEGER。1、2 または 3 でなければならない。

itype = 1 の場合は、汎用固有値問題 $Az = \lambda Bz$ で、
- *uplo* = 'U' の場合は $C = U^{-T}AU^{-1}$、$z = U^{-1}y$
- *uplo* = 'L' の場合は $C = L^{-1}AL^{-T}$、$z = L^{-T}y$

itype = 2 の場合は、汎用固有値問題 $ABz = \lambda z$ で、
- *uplo* = 'U' の場合は $C = UAU^T$、$z = U^{-1}y$
- *uplo* = 'L' の場合は $C = L^TAL$、$z = L^{-T}y$

itype = 3 の場合は、汎用固有値問題 $BAz = \lambda z$ で、
- *uplo* = 'U' の場合は $C = UAU^T$、$z = U^Ty$
- *uplo* = 'L' の場合は $C = L^TAL$、$z = Ly$

uplo CHARACTER*1。'U' または 'L' でなければならない。

uplo = 'U' の場合は、*ap* に A の上三角マトリックス (パックド形式) を格納する。
B は、$B = U^TU$ に因子分解した形式で指定しなければならない。

uplo = 'L' の場合は、*ap* に A の下三角マトリックス (パックド形式) を格納する。
B は、$B = LL^T$ に因子分解した形式で指定しなければならない。

n INTEGER。マトリックス A と B の次数 ($n \geq 0$)。

ap, bp
REAL (sspgst の場合)
DOUBLE PRECISION (dspgst 場合)
配列 :
ap(*) には、*A* の上三角マトリックスまたは下三角マトリックス (パックド形式) を格納する。
ap の次元は、max(1, *n**(*n*+1)/2) 以上でなければならない。
bp(*) には、*B* (同一の *uplo* 値を指定して ?pptrf から返されたもの) のコルスキー因数 (パックド形式) を格納する。
bp の次元は、max(1, *n**(*n*+1)/2) 以上でなければならない。

出力パラメータ

ap
引数 *itype* と *uplo* の指定に従って、*A* の上三角マトリックスまたは下三角マトリックスが、*C* の上三角マトリックスまたは下三角マトリックスによって上書きされる。

info
INTEGER
info = 0 の場合は、正常に終了する。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

縮退させたマトリックス *C* を得るのは、安定した処理である。ただし、B^{-1} (*itype* = 1 の場合) または *B* (*itype* = 2 または 3 の場合) が、暗黙的に乗算される。このルーチンを元の問題の固有値および固有ベクトルを計算する際の 1 ステップとして使う場合は、反転処理に関して *B* の条件が悪いと、精度が大きく低下するときがある。

浮動小数演算のおおよその総数は、n^3 である。

?hpgst

パックド格納形式の複素エルミート・
マトリックスの汎用固有値問題を
標準形式に縮退させる。

```
call chpgst ( itype, uplo, n, ap, bp, info )
call zhpgst ( itype, uplo, n, ap, bp, info )
```

説明

このルーチンでは、パックド格納形式を使って、次の実対称汎用固有値問題を、

$$Az = \lambda Bz、ABz = \lambda z、または\ BAz = \lambda z$$

標準形式 $Cy = \lambda y$ に縮退させる。A は実対称マトリックス、B は正値実対称マトリックスである。このルーチンを呼び出す前に、?pptrf を呼び出し、コルスキー因子分解 $B = U^H U$ または $B = LL^H$ (4-16 ページを参照) を計算する。

入力パラメータ

itype INTEGER。1、2 または 3 でなければならない。
itype = 1 の場合は、汎用固有値問題 $Az = \lambda Bz$ で、
uplo = 'U' の場合は $C = U^{-H}AU^{-1}$、$z = U^{-1}y$
uplo = 'L' の場合は $C = L^{-1}AL^{-H}$、$z = L^{-H}y$
itype = 2 の場合は、汎用固有値問題 $ABz = \lambda z$ で、
uplo = 'U' の場合は $C = UAU^H$、$z = U^{-1}y$
uplo = 'L' の場合は $C = L^H AL$、$z = L^{-H}y$
itype = 3 の場合は、汎用固有値問題 $BAz = \lambda z$ で、
uplo = 'U' の場合は $C = UAU^H$、$z = U^H y$
uplo = 'L' の場合は $C = L^H AL$、$z = Ly$

uplo CHARACTER*1。'U' または 'L' でなければならない。
uplo = 'U' の場合は、*ap* に A の上三角マトリックス (パックド形式) を格納する。B は、$B = U^H U$ に因子分解した形式で指定しなければならない。
uplo = 'L' の場合は、*ap* に A の下三角マトリックス (パックド形式) を格納する。B は、$B = LL^H$ に因子分解した形式で指定しなければならない。

n INTEGER。マトリックス A と B の次数 ($n \geq 0$)。

ap, bp

COMPLEX (chpgst の場合)
DOUBLE COMPLEX (zhpgst の場合)
配列 :
ap(*) には、*A* の上三角マトリックスまたは下三角マトリックス (パックド形式) を格納する。
a の次元は、max(1, *n**(*n*+1)/2) 以上でなければならない。

bp(*) には、*B* (同一の *uplo* 値を指定して ?pptrf から返されたもの) のコルスキー因数 (パックド形式) を格納する。
b の次元は、max(1, *n**(*n*+1)/2) 以上でなければならない。

出力パラメータ

ap

引数 *itype* と *uplo* の指定に従って、*A* の上三角マトリックスまたは下三角マトリックスが、*C* の上三角マトリックスまたは下三角マトリックスによって上書きされる。

info

INTEGER
info = 0 の場合は、正常に終了する。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

縮退させたマトリックス *C* を得るのは、安定した処理である。ただし、B^{-1} (*itype* = 1 の場合) または *B* (*itype* = 2 または 3 の場合) が、暗黙的に乗算される。このルーチンを元の問題の固有値と固有ベクトルを計算する際の 1 ステップとして使う場合は、反転処理に関して *B* の条件が悪いと、精度が大きく低下するときがある。

浮動小数演算のおおよその総数は、n^3 である。

?sbgst

?pbstf による因子分解を使用して、
バンド形式の実対称マトリックスの
汎用固有値問題を標準形式に縮退させる。

```
call ssbgst ( vect, uplo, n, ka, kb, ab, ldab, bb, ldbb, x, ldx, work,
              info )
call dsbgst ( vect, uplo, n, ka, kb, ab, ldab, bb, ldbb, x, ldx, work,
              info )
```

説明

実対称汎用固有値問題 ($Az = \lambda Bz$) を標準形式 ($Cy = \lambda y$) に縮退させる (A、B、C はバンド・マトリックス) には、このルーチンに先立って spbstf/dpbstf を呼び出す必要がある。spbstf/dpbstf は、正値マトリックス B の分割コルスキー因子分解を行う (B: $B = S^TS$)。分割コレスキー因子分解は、通常のコルスキー因子分解と比べて、作業量が半分で済む。

このルーチンは、A を $C = X^TAX$ で上書きする。ここで、$X = S^{-1}Q$ と Q は A のバンド幅を保持するために (暗黙的に) 選択された直交マトリックスである。
また、このルーチンには、X を蓄積するオプションがある。その場合、z が C の固有ベクトルならば、Xz は元の問題の固有ベクトルとなる。

入力パラメータ

vect	CHARACTER*1。'N' または 'V' でなければならない。 *vect* ='N' の場合、マトリックス X は返されない。 *vect* ='V' の場合、マトリックス X が返される。
uplo	CHARACTER*1。'U' または 'L' でなければならない。 *uplo* ='U' の場合、*ab* に A の上三角部分を格納する。 *uplo* ='L' の場合、*ab* に A の下三角部分を格納する。
n	INTEGER。マトリックス A と B の次数 ($n \geq 0$)。
ka	INTEGER。A の優対角成分または劣対角成分の個数 ($ka \geq 0$)。

kb INTEGER。*B* の優対角成分または劣対角成分の個数 ($ka \geq kb \geq 0$)。

ab, bb, work REAL (ssbgst の場合)
DOUBLE PRECISION (dsbgst の場合)
ab (*ldab, **) は、対称マトリックス *A* の上または下三角部分 (*uplo* の指定に従う) がバンド形式で格納された配列。配列 *ab* の第 2 次元のサイズは、max(1, *n*) 以上でなければならない。
bb (*ldbb, **) は、spbstf/dpbstf から返された分割コルスキー因子 *B* が、*uplo*、*n*、*kb* に従ってバンド形式で格納された配列。配列 *bb* の第 2 次元のサイズは、max(1, *n*) 以上でなければならない。
work(*) は、ワークスペース配列 DIMENSION は、max(1, 2**n*) 以上。

ldab INTEGER。配列 *ab* の第 1 次元のサイズ。*ka*+1 以上でなければならない。

ldbb INTEGER。配列 *bb* の第 1 次元のサイズ。*kb*+1 以上でなければならない。

ldx 出力配列 x の第 1 次元のサイズ。次の制約がある。
vect ='N' ならば、*ldx* ≥ 1
vect ='V' ならば、*ldx* ≥ max(1, *n*)

出力パラメータ

ab この配列は終了時に、*C* の上または下三角部分 (*uplo* の指定に従う) で上書きされる。

x REAL (ssbgst の場合)
DOUBLE PRECISION (dsbgst の場合)
配列。
vect ='V' の場合、(*ldx, **) に $n \times n$ のマトリックス $X = S^{-1}Q$ が格納される。
vect ='N' の場合、*x* は参照されない。
x の第 2 次元のサイズは、
max(1, *n*) 以上 (*vect* ='V' の場合)
1 以上 (*vect* ='N' の場合)

info INTEGER

info = 0 の場合は、正常終了している。

info = -*i* の場合は、*i* 番目のパラメータ値が不正である。

アプリケーション・ノート

縮退したマトリックス C を生成するには、暗黙的に B^{-1} が掛けられる。元の問題の固有値と固有ベクトルを計算するときの 1 ステップとしてこのルーチンを使用する場合は、B の逆マトリックス計算の条件が悪いと、精度が大きく低下する可能性がある。

浮動小数点演算の総数は、*vect* = 'N' の場合、約 $6n^2 * kb$ になる。*vect* = 'V' の場合は、さらに $(3/2)n^3 * (kb/ka)$ 回の演算が必要になる。ただしこれらの値は、*ka* と *kb* がどちらも *n* と比べて十分に小さい場合の推定値である。

?hbgst

?pbstf による因子分解を使用して、
バンド形式の複素エルミート・
マトリックスの汎用固有値問題を
標準形式に縮退させる。

```
call chbgst ( vect, uplo, n, ka, kb, ab, ldab, bb, ldbb, x, ldx, work,
              rwork, info )
call zhbgst ( vect, uplo, n, ka, kb, ab, ldab, bb, ldbb, x, ldx, work,
              rwork, info )
```

説明

複素エルミート・マトリックスの汎用固有値問題 ($Az = \lambda Bz$) を標準形式 ($Cy = \lambda y$) に縮退させる (A、B、C はバンド・マトリックス) には、このルーチンに先立って cpbstf/zpbstf を呼び出す必要がある。
cpbstf/zpbstf は、正値マトリックス B の分割コルスキー因子分解を行う ($B = S^H S$)。分割コルスキー因子分解は、通常のコルスキー因子分解と比べて、作業量が半分で済む。

このルーチンは、A を $C = X^H AX$ で上書きする。ここで、$X = S^{-1}Q$ と Q は A のバンド幅を保持するために (暗黙的に) 選択されたユニタリ・マトリックスである。
また、このルーチンには、X を蓄積するオプションがある。その場合、z が C の固有ベクトルならば、Xz は元の問題の固有ベクトルとなる。

入力パラメータ

vect	CHARACTER*1。'N' または 'V' でなければならない。 *vect* ='N' の場合、マトリックス X は返されない。 *vect* ='V' の場合、マトリックス X が返される。
uplo	CHARACTER*1。'U' または 'L' でなければならない。 *uplo* ='U' の場合、*ab* に A の上三角部分を格納する。 *uplo* ='L' の場合、*ab* に A の下三角部分を格納する。
n	INTEGER。マトリックス A と B の次数 $n \geq 0$
ka	INTEGER。A の優対角成分または劣対角成分の個数 ($ka \geq 0$)。

kb INTEGER。*B* の優対角成分または劣対角成分の個数 ($ka \geq kb \geq 0$)。

ab, bb, work COMPLEX (chbgst の場合)
DOUBLE COMPLEX (zhbgst の場合)
ab (*ldab, **) は、エルミート・マトリックス *A* の上または下三角部分 (*uplo* の指定に従う) がバンド形式で格納された配列。配列 *ab* の第 2 次元のサイズは、max(1, *n*) 以上でなければならない。
bb (*ldbb, **) は、cpbstf/zpbstf から返された分割コルスキー因子 *B* が、*uplo*、*n*、*kb* に従ってバンド形式で格納された配列。配列 *bb* の第 2 次元のサイズは、max(1, *n*) 以上でなければならない。
work(*) は、ワークスペース配列 DIMENSION は、max(1, *n*) 以上。

ldab INTEGER。配列 *ab* の第 1 次元のサイズ。*ka*+1 以上でなければならない。

ldbb INTEGER。配列 *bb* の第 1 次元のサイズ。*kb*+1 以上でなければならない。

ldx 出力配列 x の第 1 次元のサイズ。次の制約がある。
vect ='N' ならば、*ldx* ≥ 1
vect ='V' ならば、*ldx* ≥ max(1, *n*)

rwork REAL (chbgst の場合)
DOUBLE PRECISION (zhbgst の場合)
ワークスペース配列 DIMENSION は max(1, *n*) 以上。

出力パラメータ

ab この配列は終了時に、*C* の上または下三角部分 (*uplo* の指定に従う) で上書きされる。

x COMPLEX (chbgst の場合)
DOUBLE COMPLEX (zhbgst の場合)
配列。
vect ='V' の場合、*x* (*ldx, **) に $n \times n$ のマトリックス $X = S^{-1}Q$ が格納される。
vect ='N' の場合、*x* は参照されない。

x の第 2 次元のサイズは、
max(1, *n*) 以上 (*vect* ='V' の場合)
1 以上 (*vect* ='N' の場合)

info INTEGER

info = 0 の場合は、正常終了している。
info = -*i* の場合は、*i* 番目のパラメータ値が不正である。

アプリケーション・ノート

縮退したマトリックス C を生成するには、暗黙的に B^{-1} が掛けられる。元の問題の固有値と固有ベクトルを計算するときの 1 ステップとしてこのルーチンを使用する場合は、B の逆マトリックス計算の条件が悪いと、精度が大きく低下する可能性がある。
浮動小数点演算の総数は、*vect* ='N' の場合、約 $20n^2$*kb になる。*vect* ='V' の場合は、さらに $5n^3$*(kb/ka) 回の演算が必要になる。ただしこれらの値は、*ka* と *kb* がどちらも *n* と比べて十分に小さい場合の推定値である。

?pbstf

?sbgst/?hbgst で使用するために、
バンド形式の正値の実対称マトリックス
または複素エルミート・マトリックスの
分割コルスキー因子分解を計算する。

```
call spbstf ( uplo, n, kb, bb, ldbb, info )
call dpbstf ( uplo, n, kb, bb, ldbb, info )
call cpbstf ( uplo, n, kb, bb, ldbb, info )
call zpbstf ( uplo, n, kb, bb, ldbb, info )
```

説明

このルーチンは、バンド形式の正値の実対称マトリックス B または複素エルミート・マトリックスの分割コルスキー因子分解を計算する。このルーチンは、?sbgst/?hbgst と共に使用する。

分割コルスキー因子分解は、$B = S^T S$ (複素数の場合) と表される。ここで、S は、バンド幅が B と同じバンド・マトリックスで、最初の (n+kb)/2 行は上三角マトリックス、残りの行は下三角マトリックスである。

入力パラメータ

uplo	CHARACTER*1。'U' または 'L' でなければならない。 *uplo* ='U' の場合、*bb* に B の上三角部分を格納する。 *uplo* ='L' の場合、*bb* に B の下三角部分を格納する。
n	INTEGER。マトリックス B の次数 ($n \geq 0$)。
kb	INTEGER。B の優対角成分または劣対角成分の個数 ($kb \geq 0$))。
bb	REAL (spbstf の場合) DOUBLE PRECISION (dpbstf の場合) COMPLEX (cpbstf の場合) DOUBLE COMPLEX (zpbstf の場合) *bb* (*ldbb*, *) 対称マトリックス B の上または下三角部分 (*uplo* の指定に従う) がバンド形式で格納された配列。

配列 bb の第 2 次元のサイズは、max(1, n) 以上でなければならない。

ldbb　INTEGER。bb の第 1 次元のサイズ。kb+1 以上でなければならない。

出力パラメータ

bb　この配列は終了時に、分割コルスキー因子 S の各成分で上書きされる。

info　INTEGER

info = 0 の場合は、正常終了している。

info = i の場合は、更新使用とした成分 b_{ii} が負値の平方根になるため、因子分解を完了できなかった。すなわち、マトリックス B が正値でない。

info = -i の場合は、i 番目のパラメータ値が不正である。

アプリケーション・ノート

計算された因子 S は、正確には摂動マトリックス $B + E$ の因子である。ただし、

$$|E| \le c(kb+1)\varepsilon|S^H||S|, \quad |e_{ij}| \le c(kb+1)\varepsilon\sqrt{b_{ii}b_{jj}}$$

$c(n)$ は n の適度な線形関数、ε はマシン精度である。

浮動小数点演算の総数は、実数の場合は約 $n(kb+1)^2$ である。複素数の場合は、この 4 倍大きくなる。ただしこれらの値は、kb が n と比べて十分に小さい場合の推定値である。

このルーチンを呼び出してから、?sbgst/?hbgst を呼び出して汎用固有値問題 ($Az = \lambda Bz$、ただし、A と B はバンド形式、B は正値) を解ける。

非対称固有値問題

ここでは、非対称固有値問題の解決、一般マトリックスの Schur 因子分解の計算、およびそれらに関連する各種の計算タスクを実行するための LAPACK ルーチンについて説明する。

非対称固有値問題は、「非対称 (または非エルミート) のマトリックス A が与えられたときに、次のいずれかの方程式を満たす**固有値** λ と、それに対応する**固有ベクトル** z を求めること」である。

$$Az = \lambda z \text{ (右固有ベクトル } z\text{)}$$

または

$$z^H A = \lambda z^H \text{ (左固有ベクトル } z\text{)}$$

非対称固有値問題には、次の特性がある。

- 固有ベクトルの個数が、マトリックスの次数より少ない場合がある (ただし、A の**相異なる固有値**の個数以上である)。
- 実マトリックス A に対して、固有値が複素数の場合もある。
- 実数型の非対称マトリックスが、固有ベクトル z に対応する固有値として複素数 $a + bi$ を持っている場合は、$a - bi$ も固有値である。固有値 a-bi は、z の成分に対して複素共役になっている成分の固有ベクトルに対応する。

LAPACK を使って非対称固有値問題を解くには、通常、マトリックスを上 Hessenberg 形式に縮退させた後、得られた Hessenberg マトリックスを使って固有値問題を解く必要がある。図 5-5 に掲載されている LAPACK ルーチンを利用すれば、直交 (またはユニタリ) の相似変換 $A = QHQ^H$ を使って、該当マトリックスを上 Hessenberg 形式に縮退させられる。また、Hessenberg マトリックスを使って固有値問題を解き、マトリックスの Schur 因子分解を求め、対応する条件数を計算できる。

図 5-4 のデシジョン・ツリーを参考にすれば、実数型の非対称マトリックスを使って固有値問題を解くためのルーチン (1 つまたは複数個) を簡単に選べる。
複素数型の非エルミート・マトリックスを使って固有値問題を解く場合は、図 5-5 のデシジョン・ツリーを参照。

表 5-5　非対称固有値問題を解くための計算ルーチン

実行する演算	実マトリックス用のルーチン	複素マトリックス用のルーチン
Hessenberg 形式 $A = QHQ^H$ に縮退させる	?gehrd	?gehrd
マトリックス Q を生成する	?orghr	?unghr
マトリックス Q を適用する	?ormhr	?unmhr
マトリックスの平衡化	?gebal	?gebal
平衡化したマトリックスの固有ベクトルを元のマトリックスの固有ベクトルに変換する	?gebak	?gebak
固有値と Schur 因子分解を求める (QR アルゴリズム)	?hseqr	?hseqr
Hessenberg 形式から固有ベクトルを求める (反転繰り返し法)	?hsein	?hsein
Schur の因子分解から固有ベクトルを求める	?trevc	?trevc
固有値と固有ベクトルの条件数を見積もる	?trsna	?trsna
Schur の因子分解の順序を変更する	?trexc	?trexc
Schur の因子分解の順序変更、不変部分空間の検出、条件数の見積もり	?trsen	?trsen
シルベスター式を解く	?trsyl	?trsyl

図 5-4　デシジョン・ツリー : 実数型の非対称固有値問題

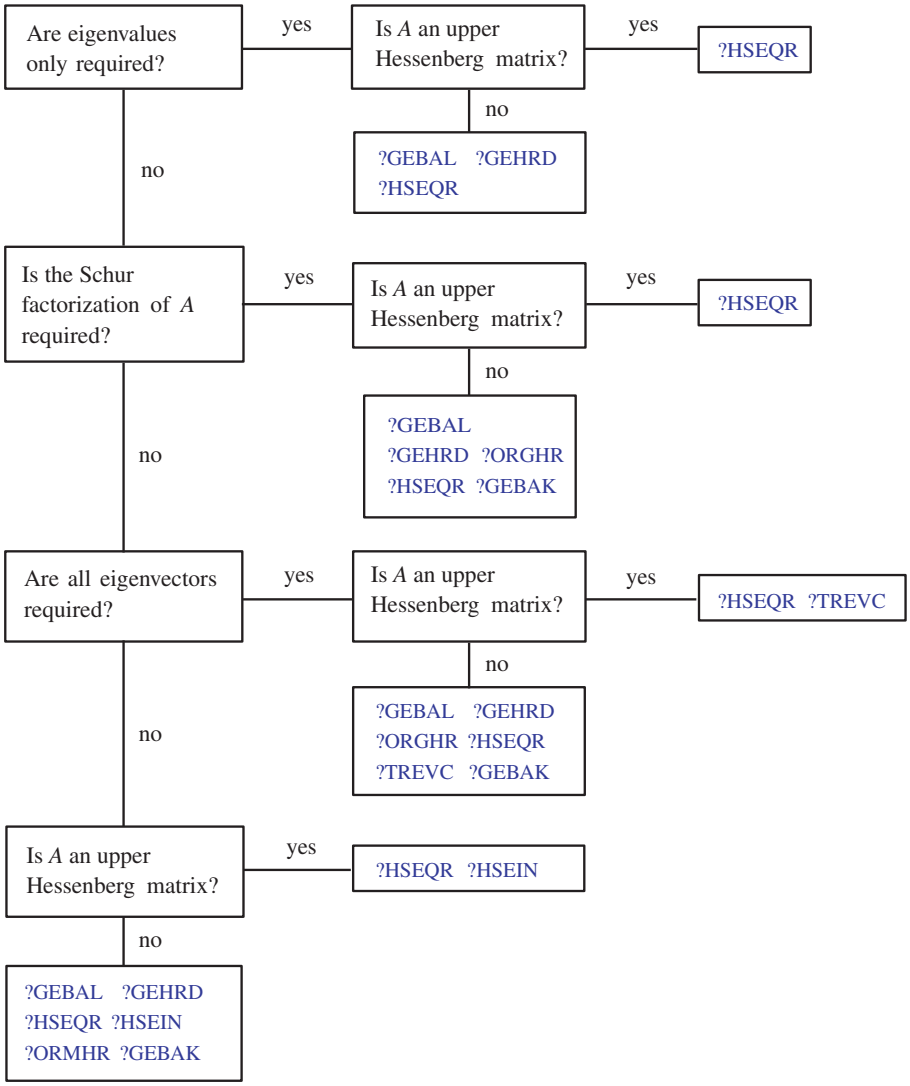
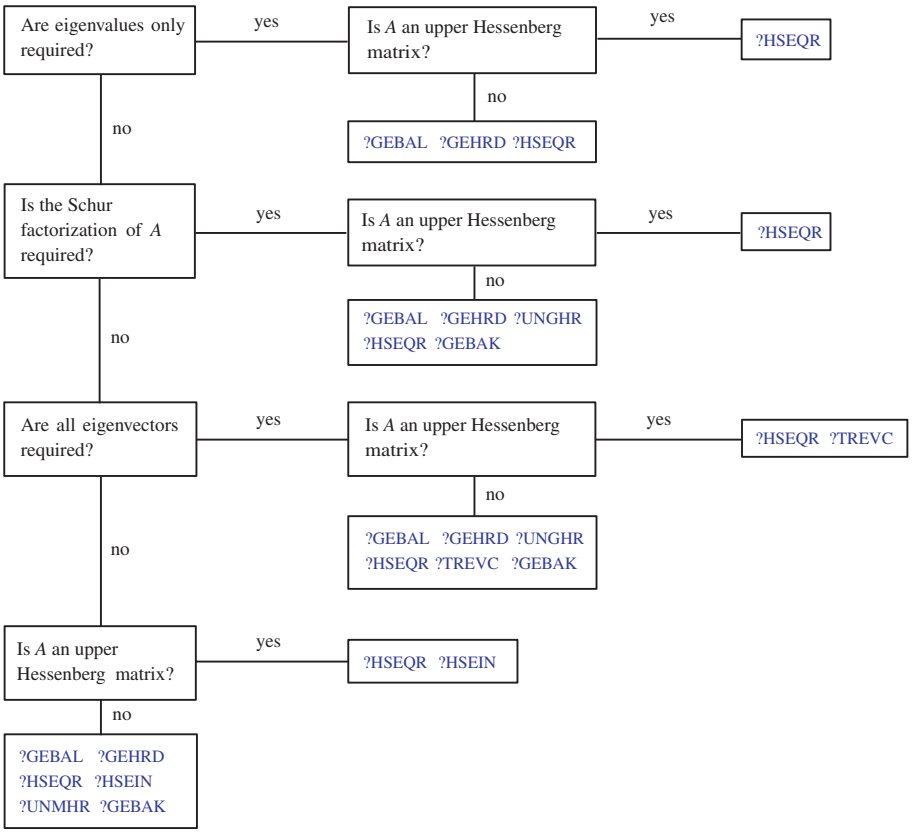


図 5-5　デシジョン・ツリー：複素数型の非エルミート固有値問題



?gehrd

一般マトリックスを上 Hessenberg 形式に
縮退させる。

```
call sgehrd ( n, ilo, ihi, a, lda, tau, work, lwork, info )
call dgehrd ( n, ilo, ihi, a, lda, tau, work, lwork, info )
call cgehrd ( n, ilo, ihi, a, lda, tau, work, lwork, info )
call zgehrd ( n, ilo, ihi, a, lda, tau, work, lwork, info )
```

説明

このルーチンでは、直交またはユニタリの相似変換 $A = QHQ^H$ によって、一般マトリックス A を上 Hessenberg 形式 H に縮退させる。H は、実数型の対角成分を持つ。

このルーチンでは、マトリックス Q を明示的に表現せず、一連の**基本リフレクタ**の積として表現する。一連のルーチンは、このように表現される Q を処理するためのものである。

入力パラメータ

n	INTEGER。マトリックス A の次数 ($n \geq 0$)。
ilo, ihi	INTEGER。A が ?gebal から出力された場合は、*ilo* と *ihi* に、そのルーチンから返された値を設定しなければならない。そうでない場合は、*ilo* = 1 かつ *ihi* = *n* とする ($n > 0$ の場合は $1 \leq ilo \leq ihi \leq n$ で、$n = 0$ の場合は *ilo* = 1 かつ *ihi* = 0 とする)。
a, work	REAL (sgehrd の場合) DOUBLE PRECISION (dgehrd の場合) COMPLEX (cgehrd の場合) DOUBLE COMPLEX (zgehrd の場合) 配列 : *a* (*lda*, *) には、マトリックス A を格納する。 *a* の第 2 次元は、max(1, *n*) 以上でなければならない。 *work* (*lwork*) は、ワークスペース配列である。
lda	INTEGER。*a* の第 1 次元。max(1, *m*) 以上であること。

lwork INTEGER。配列 *work* のサイズ (max(1, *n*) 以上)。
lwork の推奨値は、「アプリケーション・ノート」を参照。

出力パラメータ

a 上 Hessenberg マトリックス H とマトリックス Q の各成分によって上書きされる。H の劣対角成分は、実数である。

tau REAL (sgehrd の場合)
DOUBLE PRECISION (dgehrd の場合)
COMPLEX (cgehrd の場合)
DOUBLE COMPLEX (zgehrd の場合)
配列 DIMENSION は、max(1, *n*-1) 以上。
マトリックス Q に関するその他の情報も格納される。

work(1) *info* = 0 の場合は、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の *lwork* の値が *work(1)* に格納される。これ以降の実行には、この *lwork* の値を使用する。

info INTEGER
info = 0 の場合は、正常に終了する。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

パフォーマンスを改善するには、*lwork* = *n***blocksize* に設定する。*blocksize* は、**ブロック化アルゴリズム**の最適パフォーマンスに必要な値で、マシンによって異なる (通常は 16 ～ 64)。用意すべきワークスペースの量が不明な場合は、最初は *lwork* に大きな値を設定する。終了時に *work(1)* の値を調べ、これ以降はその値を使用する。

計算で求めた Hessenberg マトリックス H は、まさに近似マトリックスの $A + E$ に類似しており、$\|E\|_2 < c(n)\varepsilon\|A\|_2$ であるとともに、$c(n)$ は漸増する n の関数で、ε はマシンによって異なる値である。

実数の場合の浮動小数演算のおおよその総数は、次のとおりである。$(2/3)(ihi - ilo)^2(2ihi + 2ilo + 3n)$ である。複素数型の場合は、この 4 倍になる。

?orghr

?gehrd で求めた実数型直交マトリックス
Q を生成する。

```
call sorghr ( n, ilo, ihi, a, lda, tau, work, lwork, info )
call dorghr ( n, ilo, ihi, a, lda, tau, work, lwork, info )
```

説明

このルーチンでは、sgehrd/dgehrd を呼び出して求めた直交マトリックス Q を明示的に生成する (ルーチン ?gehrd は、直交相似変換 $A = QHQ^T$ によって実数型の一般マトリックス A を上 Hessenberg 形式 H に縮退させ、マトリックス Q を (*ihi*-*ilo*) 個の**基本リフレクタ**の積として表現する。*ilo* と *ihi* は、マトリックスを平衡化する際に sgebal/dgebal で求めた値である。マトリックスが平衡化されていない場合は、*ilo* = 1 かつ *ihi* = *n* になる)。

$$Q = \begin{bmatrix} I & 0 & 0 \\ 0 & Q_{22} & 0 \\ 0 & 0 & I \end{bmatrix}$$

Q_{22} は、*il* ～ *?ihi* の行と列を占める。

入力パラメータ

n	INTEGER。マトリックス Q の次数 ($n \geq 0$)。
ilo, ihi	INTEGER。これらの値はそれぞれ、?gehrd で指定したのと同じ値でなければならない ($n > 0$ の場合は $1 \leq ilo \leq ihi \leq n$ で、$n = 0$ の場合は $ilo = 1$ かつ $ihi = 0$)。
a, tau, work	REAL (sorghr の場合) DOUBLE PRECISION (dorghr の場合) 配列 :

	a(lda, *) には、?gehrd から返された一連のベクトル (基本リフレクタを定義) の各成分を格納する。 a の第 2 次元は、max(1, n) 以上でなければならない。 tau(*) には、?gehrd から返された基本リフレクタの各成分を格納する。 tau の次元は、max(1, n-1) 以上でなければならない。 work(lwork) は、ワークスペース配列である。
lda	INTEGER。a の第 1 次元。以上。
lwork	INTEGER。配列 work のサイズ。 lwork ≥ max(1,ihi−ilo) lwork の推奨値は、「アプリケーション・ノート」を参照。

出力パラメータ

a	n × n の直交マトリックス Q によって上書きされる。
work(1)	info = 0 の場合は、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の lwork の値が work(1) に格納される。これ以降の実行には、この lwork の値を使用する。
info	INTEGER info = 0 の場合は、正常に終了する。 info = -i の場合は、i 番目のパラメータの値が不正である。

アプリケーション・ノート

パフォーマンスを改善するには、lwork = (ihi-ilo)*blocksize に設定する。*blocksize* は、**ブロック化アルゴリズム**の最適パフォーマンスに必要な値で、マシンによって異なる (通常は 16 ～ 64)。用意すべきワークスペースの量が不明な場合は、最初は lwork に大きな値を設定する。終了時に work(1) の値を調べ、これ以降はその値を使用する。

計算で求めたマトリックス Q は、正確な値とマトリックス E ($\|E\|_2 = O(\varepsilon)$、ε はマシン精度) だけ異なる。

浮動小数演算のおおよその総数は、$(4/3)(ihi-ilo)^3$ である。

このルーチンの複素数版は、?unghr である。

?ormhr

?gehrd で求めた実数型直交マトリックス Q に、任意の実マトリックス C を掛け合わせる。

```
call sormhr ( side, trans, m, n, ilo, ihi, a, lda, tau, c, ldc, work,
              lwork, info )
call dormhr ( side, trans, m, n, ilo, ihi, a, lda, tau, c, ldc, work,
              lwork, info )
```

説明

このルーチンでは、sgehrd/dgehrd を呼び出して求めた直交マトリックス Q に、マトリックス C を掛け合わせる (ルーチン ?gehrd は、直交相似変換 $A = QHQ^T$ によって実数型の一般マトリックス A を上 Hessenberg 形式 H に縮退させ、マトリックス Q を (*ihi*-*ilo*) 個の**基本リフレクタ**の積として表現する。*ilo* と *ihi* は、マトリックスを平衡化する際に sgebal/dgebal で求めた値である。マトリックスが平衡化されていない場合は、*ilo* = 1 かつ *ihi* = *n* になる)。

?ormhr を使用すれば、QC、Q^TC、CQ、または CQ^T のいずれかのマトリックス積を求められる。その場合は、C (実数型矩形マトリックスの場合もある) の値が上書きされる。

?ormhr は、H の固有ベクトルで構成されるマトリックス V を、A の固有ベクトルで構成されるマトリックス QV に変換する際によく使用する。

入力パラメータ

side CHARACTER*1。'L' または 'R' でなければならない。
side ='L' の場合、QC または Q^TC を求める。
side ='R' の場合、CQ または CQ^T を求める。

trans CHARACTER*1。'N' または 'T' でなければならない。
trans = 'N' の場合は、Q が C に適用される。
trans = 'T' の場合は、Q^T が C に適用される。

m INTEGER。C の列数 ($m \geq 0$)。

n INTEGER。C の列数 ($n \geq 0$)。

ilo, ihi INTEGER。これらの値はそれぞれ、?gehrd で指定したのと同じ値でなければならない。
$m > 0$ かつ *side* = 'L' の場合は、$1 \le ilo \le ihi \le m$
$m = 0$ かつ *side* = 'L' の場合は、$ilo = 1$ かつ $ihi = 0$
$n > 0$ かつ *side* = 'R' の場合は、$1 \le ilo \le ihi \le n$
$n = 0$ かつ *side* = 'R' の場合は、$ilo = 1$ かつ $ihi = 0$

a, tau, c, work REAL (sormhr の場合)
DOUBLE PRECISION (dormhr の場合)
配列 :
*a(lda, *)* には、?gehrd から返された一連のベクトル (**基本リフレクタ**を定義) の各成分を格納する。
a の第 2 次元は、*side* = 'L' の場合は max(1, *m*) 以上で、*side* = 'R' の場合は max(1, *n*) 以上でなければならない。
tau(*) には、?gehrd から返された**基本リフレクタ**の各成分を格納する。
tau の次元は、max(1, *m*-1) 以上 (*side* = 'L' の場合)、あるいは max(1, *n*-1) 以上 (*side* = 'R' の場合) でなければならない。
*c(ldc, *)* には、$m \times n$ のマトリックス C を格納する。
c の第 2 次元は、max(1, *n*) 以上でなければならない。
work (lwork) は、ワークスペース配列である。

lda INTEGER。*a* の第 1 次元 *side* = 'L' の場合は、max(1, *m*) 以上で、*side* = 'R' の場合は、max(1, *n*) 以上。

ldc INTEGER。*c* の第 1 次元。max(1, *m*) 以上。

lwork INTEGER。配列 *work* のサイズ。
side = 'L' の場合は、*lwork* ≥ max(1, *n*)
side = 'R' の場合は、*lwork* ≥ max(1, *m*)
lwork の推奨値は、「アプリケーション・ノート」を参照。

出力パラメータ

c C は、*side* と *trans* の値に従って、QC、Q^TC、CQ^T、または CQ によって上書きされる。

work(1) *info* = 0 の場合は、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の *lwork* の値が *work(1)* に格納される。これ以降の実行には、この *lwork* の値を使用する。

info INTEGER

info = 0 の場合は、正常に終了する。

info = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

パフォーマンスを改善するには、*lwork* を、*side* = 'L' の場合は *n***blocksize* 以上に、*side* = 'R' の場合は *m***blocksize* 以上に設定する。*blocksize* は、**ブロック化アルゴリズム**の最適パフォーマンスに必要な値で、マシンによって異なる (通常は 16 ～ 64)。用意すべきワークスペースの量が不明な場合は、最初は *lwork* に大きな値を設定する。終了時に *work*(1) の値を調べ、これ以降はその値を使用する。

計算で求めたマトリックス Q は、正確な値とマトリックス E ($\|E\|_2 = O(\varepsilon)\|C\|_2$、ε はマシン精度) だけ異なる。

浮動小数演算のおおよその総数は、
$2n(ihi-ilo)^2$ (*side* = 'L' の場合)
$2m(ihi-ilo)^2$ (*side* = 'R' の場合)

このルーチンの複素数版は、?unmhr である。

?unghr

?gehrd で求めた複素ユニタリ・
マトリックス Q を生成する。

```
call cunghr ( n, ilo, ihi, a, lda, tau, work, lwork, info )
call zunghr ( n, ilo, ihi, a, lda, tau, work, lwork, info )
```

説明

このルーチンは、cgehrd/zgehrd (ユニタリ相似変換 $A = QHQ^H$ によって、複素マトリックス A を上 Hessenberg 形式 H に縮退させる) を呼び出した後で使用する。?gehrd は、マトリックス Q を (*ihi*-*ilo*) 個の**基本リフレクタ**の積として表現する。*ilo* と *ihi* は、マトリックスを平衡化する際に cgebal/zgebal で求めた値である。マトリックスが平衡化されていない場合は、*ilo* = 1 かつ *ihi* = *n* になる)。

Q を正方マトリックスとして明示的に生成するには、ルーチン ?unghr を使う。マトリックス Q は、次の構造になる。

$$Q = \begin{bmatrix} I & 0 & 0 \\ 0 & Q_{22} & 0 \\ 0 & 0 & I \end{bmatrix}$$

Q_{22} は、*ilo* ～ *ihi* の行と列を占める。

入力パラメータ

n	INTEGER。マトリックス Q の次数 ($n \geq 0$)。
ilo, ihi	INTEGER。これらの値はそれぞれ、?gehrd で指定したのと同じ値でなければならない ($n > 0$ の場合は $1 \leq ilo \leq ihi \leq n$ で、$n = 0$ の場合は $ilo = 1$ かつ $ihi = 0$)。
a, tau, work	COMPLEX (cunghr の場合) DOUBLE COMPLEX (zunghr の場合) 配列 :

	a(lda, *) には、?gehrd から返された一連のベクトル (**基本リフレクタ**を定義) の各成分を格納する。 a の第 2 次元は、max(1, n) 以上でなければならない。 tau(*) には、?gehrd から返された**基本リフレクタ**の各成分を格納する。 tau の次元は、max(1, n-1) 以上でなければならない。 work (lwork) は、ワークスペース配列である。
lda	INTEGER。a の第 1 次元。max(1, n) 以上。
lwork	INTEGER。配列 work のサイズ lwork ≥ max(1, ihi−ilo) lwork の推奨値は、「アプリケーション・ノート」を参照。

出力パラメータ

a	n × n のユニタリ・マトリックス Q によって上書きされる。
work(1)	info = 0 の場合は、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の lwork の値が work(1) に格納される。これ以降の実行には、この lwork の値を使用する。
info	INTEGER info = 0 の場合は、正常に終了する。 info = -i の場合は、i 番目のパラメータの値が不正である。

アプリケーション・ノート

パフォーマンスを改善するには、lwork = (ihi-ilo)*blocksize に設定する。*blocksize* は、**ブロック化アルゴリズム**の最適パフォーマンスに必要な値で、マシンによって異なる (通常は 16 ～ 64)。用意すべきワークスペースの量が不明な場合は、最初は lwork に大きな値を設定する。終了時に work(1) の値を調べ、これ以降はその値を使用する。

計算で求めたマトリックス Q は、正確な値とマトリックス E ($\|E\|_2 = O(\varepsilon)$、ε はマシン精度) だけ異なる。

実数の場合の浮動小数演算のおおよその総数は、(16/3) (ihi−ilo) である。

このルーチンの実数版は、?orghr である。

?unmhr

?gehrd で求めた複素ユニタリ・マトリックス Q に、任意の複素マトリックス C を掛け合わせる。

```
call cunmhr ( side, trans, m, n, ilo, ihi, a, lda, tau, c, ldc, work,
              lwork, info )
call zunmhr ( side, trans, m, n, ilo, ihi, a, lda, tau, c, ldc, work,
              lwork, info )
```

説明

このルーチンでは、cgehrd/zgehrd を呼び出して求めたユニタリ・マトリックス Q にマトリックス C を掛け合わせる (ルーチン ?gehrd は、直交相似変換 $A = QHQ^H$ によって実数型の一般マトリックス A を上 Hessenberg 形式 H に縮退させ、マトリックス Q を (*ihi*-*ilo*) 個の**基本リフレクタ**の積として表現する。*ilo* と *ihi* は、マトリックスを平衡化する際に cgebal/zgebal で求めた値である。マトリックスが平衡化されていない場合は、*ilo* = 1 かつ *ihi* = *n* になる)。

?unmhr を使用すれば、QC、Q^HC、CQ、または CQ^H のいずれかのマトリックス積を求められる。その場合は、C (複素矩形マトリックスの場合もある) の値が上書きされる。このルーチンは、H の固有ベクトルで構成されるマトリックス V を、A の固有ベクトルで構成されるマトリックス QV に変換する際によく使われる。

入力パラメータ

side CHARACTER*1。'L' または 'R' でなければならない。
side ='L' の場合、QC または Q^HC を求める。
side ='R' の場合、CQ または CQ^H を求める。

trans CHARACTER*1。'N' または 'C' でなければならない。
trans ='N' の場合は、Q が C に適用される。
trans ='T' の場合は、Q^H が C に適用される。

m INTEGER。C の行数 ($m \geq 0$)。

n INTEGER。C の列数 ($n \geq 0$)。

ilo, ihi INTEGER。これらの値はそれぞれ、?gehrd で指定したのと同じ値でなければならない。
$m > 0$ かつ *side* = 'L' の場合は、$1 \le ilo \le ihi \le m$
$m = 0$ かつ *side* = 'L' の場合は、$ilo = 1$ かつ $ihi = 0$
$n > 0$ かつ *side* = 'R' の場合は、$1 \le ilo \le ihi \le n$
$n = 0$ かつ *side* = 'R' の場合は、$ilo = 1$ かつ $ihi = 0$

a, tau, c, work COMPLEX (cunmhr の場合)
DOUBLE COMPLEX (zunmhr の場合)
配列 :
a (*lda*, *) には、?gehrd から返された一連のベクトル (**基本リフレクタ**を定義) の各成分を格納する。
a の第 2 次元は、*side* = 'L' の場合は max(1, *m*) 以上、*side* = 'R' の場合は max(1, *n*) 以上でなければならない。
tau(*) には、?gehrd から返された基本リフレクタの各成分を格納する。
tau の次元は、max(1, *m*-1) 以上 (*side* = 'L' の場合)、あるいは max(1, *n*-1) 以上 (*side* = 'R' の場合) でなければならない。
c (*ldc*, *) には、$m \times n$ のマトリックス C を格納する。
c の第 2 次元は、max(1, *n*) 以上でなければならない。
work (*lwork*) は、ワークスペース配列である。

lda INTEGER。*a* の第 1 次元。*side* = 'L' の場合は max(1, *m*) 以上で、*side* = 'R' の場合は max(1, *n*) 以上。

ldc INTEGER。*c* の第 1 次元。max(1, *m*) 以上。

lwork INTEGER。配列 *work* のサイズ。
side = 'L' の場合は、*lwork* ≥ max(1, *n*)
side = 'R' の場合は、*lwork* ≥ max(1, *m*)
lwork の推奨値は、「アプリケーション・ノート」を参照。

出力パラメータ

c C は、*side* と *trans* の値に従って、QC、Q^HC、CQ^H、または CQ のいずれかによって上書きされる。

work(1) *info* = 0 の場合は、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の *lwork* の値が *work*(1) に格納される。これ以降の実行には、この *lwork* の値を使用する。

info INTEGER

info = 0 の場合は、正常に終了する。

info = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

パフォーマンスを改善するには、*lwork* を、*n***blocksize* 以上 (*side* = 'L' の場合)、あるいは *m***blocksize* 以上 (*side* = 'R' の場合) に設定する。*blocksize* は、**ブロック化アルゴリズム**の最適パフォーマンスに必要な値で、マシンによって異なる (通常は 16 ～ 64)。用意すべきワークスペースの量が不明な場合は、最初は *lwork* に大きな値を設定する。終了時に *work*(1) の値を調べ、これ以降はその値を使用する。

計算で求めたマトリックス Q は、正確な値とマトリックス E ($\|E\|_2 = O(\varepsilon)\,\|C\|_2$、ε はマシン精度) だけ異なる。

浮動小数演算のおおよその総数は、次のとおりである。

$8n(ihi-ilo)^2$ (*side* = 'L' の場合)

$8m(ihi-ilo)^2$ (*side* = 'R' の場合)

このルーチンの実数版は、?ormhr である。

?gebal

一般マトリックスを平衡化し、固有値
および固有ベクトルの計算精度を
改善する。

```
call sgebal ( job, n, a, lda, ilo, ihi, scale, info )
call dgebal ( job, n, a, lda, ilo, ihi, scale, info )
call cgebal ( job, n, a, lda, ilo, ihi, scale, info )
call zgebal ( job, n, a, lda, ilo, ihi, scale, info )
```

説明

このルーチンでは、次の 2 つの相似変換の一方または両方を実行して、マトリックス A を*平衡化する*。

(1) まず、A をブロック上三角形式に置換してみる。

$$PAP^T = A' = \begin{bmatrix} A'_{11} & A'_{12} & A'_{13} \\ 0 & A'_{22} & A'_{23} \\ 0 & 0 & A'_{33} \end{bmatrix}$$

P は置換マトリックス、A'_{11} と A'_{33} は上三角マトリックスである。A'_{11} と A'_{33} の対角成分は、A の固有値である。A の残りの固有値は、中央の対角ブロック A'_{22} の固有値 (`ilo` ～ `ihi` の行と列) である。A の固有値 (またはその Schur 因子分解) を計算するためのこれ以降の演算は、この範囲の行と列に対して適用するだけで済む。そのため、`ilo` > 1 かつ `ihi` < `n` の場合は、作業を大幅に節約できる。適切な置換マトリックスが存在しない場合 (存在しない場合が多い) は、`ilo` = 1 かつ `ihi` = `n` と設定され、A'_{22} が A の全体になる。

(2) 対角相似変換を A' に適用し、A'_{22} の行と列をノルム内でできるだけ近づける。

$$A'' = DA'D^{-1} = \begin{bmatrix} I & 0 & 0 \\ 0 & D_{22} & 0 \\ 0 & 0 & I \end{bmatrix} \times \begin{bmatrix} A'_{11} & A'_{12} & A'_{13} \\ 0 & A'_{22} & A'_{23} \\ 0 & 0 & A'_{33} \end{bmatrix} \times \begin{bmatrix} I & 0 & 0 \\ 0 & D_{22}^{-1} & 0 \\ 0 & 0 & I \end{bmatrix}$$

このスケーリングによって、このマトリックスのノルムが縮退される (つまり、$\|A''_{22}\| < \|A'_{22}\|$)。その結果、固有値と固有ベクトルの計算精度に対する丸め誤差の影響が減る。

入力パラメータ

job CHARACTER*1。'N'、'P'、'S'、または 'B' のいずれかでなければならない。
job ='N' の場合は、*A* の置換およびスケーリングは実行されない (ただし、*ilo*、*ihi*、*scale* は、入力された値を受け取る)。
job ='P' の場合は、*A* の置換が実行され、スケーリングは実行されない。
job ='S' の場合は、*A* のスケーリングが実行され、置換は実行されない。
job ='B' の場合は、*A* のスケーリングと置換が両方とも実行される。

n INTEGER。マトリックス *A* の次数 ($n \geq 0$)。

a REAL (sgebal の場合)
DOUBLE PRECISION (dgebal の場合)
COMPLEX (cgebal の場合)
DOUBLE COMPLEX (zgebal の場合)
配列 :
a (*lda*, *) には、マトリックス *A* を格納する。
a の第 2 次元は、max(1, *n*) 以上でなければならない。
a は、*job* ='N' の場合は、参照されない。

lda INTEGER。*a* の第 1 次元。(1, *n*) 以上。

出力パラメータ

a 平衡化されたマトリックスによって上書きされる (*a* は、*job* ='N' の場合は、参照されない)。

ilo, ihi INTEGER。$i > j$ かつ $1 \leq j$ *ilo* または *ihi* $< i \leq n$ の場合は、終了時に、*a*(*i*, *j*) はゼロになる。
job ='N' または 'S' の場合は、*ilo* = 1 かつ *ihi* = *n* になる。

scale REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
配列 DIMENSION は、max(1, *n*) 以上。
置換とスケーリング係数の各成分が格納される。

もう少し詳しく説明すると、p_j が行と列 j と交換された行と列のインデックスであり、d_j が行と列 j を平衡化する際に使ったスケーリング係数である場合は、次のようになる。

$scale(j) = p_j$ (j = 1, 2, . . . , ilo-1, ihi+1, . . . , n);

$scale(j) = d_j$(j = ilo, ilo + 1, . . . , ihi)

交換の実行順序は、n ～ ihi+1、次に、1 ～ ilo-1 になる。

info INTEGER

info = 0 の場合は、正常に終了する。

info = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

誤差は、これ以降の計算における誤差に比べて無視できる程度である。

マトリックス A をこのルーチンによって平衡化した場合は、以降に計算する固有ベクトルはすべて、マトリックス A'' の固有ベクトルになる。そのため、?gebak (5-193 ページを参照) を呼び出して、それらを A の固有ベクトルに変換しなければならない。

A の Schur ベクトルが必要な場合は、*job* = 'S' または 'B' と指定してこのルーチンを呼び出してはならない。つまり、これらを指定すると、平衡化のための変換が直交 (複素数型の場合はユニタリ) にならないからである。*job* = 'P' と指定してこのルーチンを呼び出すと、以降に計算する Schur ベクトルはすべて、マトリックス A'' の Schur ベクトルになる。そのため、?gebak (*side* ='R' と指定) を呼び出して、それらを A の Schur ベクトルに変換し直す必要がある。

浮動小数演算の総数は、n^2 に比例する。

?gebak

平衡化後のマトリックスの固有ベクトルを、
元の非対称マトリックスの固有ベクトルに
変換する。

```
call sgebak ( job, side, n, ilo, ihi, scale, m, v, ldv, info )
call dgebak ( job, side, n, ilo, ihi, scale, m, v, ldv, info )
call cgebak ( job, side, n, ilo, ihi, scale, m, v, ldv, info )
call zgebak ( job, side, n, ilo, ihi, scale, m, v, ldv, info )
```

説明

このルーチンは、?gebal を呼び出してマトリックス A を平衡化した後、平衡化後のマトリックス A''_{22} の固有ベクトルを計算してから使用する。

平衡化については、?gebal (5-190 ページ) を参照。平衡化後のマトリックス A'' は、$A''= DPAP^TD^{-1}$ として得られる (P は置換マトリックス、D はスケーリング用の対角マトリックス)。このルーチンでは、固有ベクトルを次のように変換する。
x が A'' の右固有ベクトルである場合は、$P^TD^{-1}x$ が A の右固有ベクトルになる。
x が A'' の左固有ベクトルである場合は、P^TDy が A の左固有ベクトルになる。

入力パラメータ

job CHARACTER*1。'N' 、'P' 、'S'、または 'B' でなければならない。
job は、?gebal で指定したのと同じ値である。

side CHARACTER*1。'L' または 'R' でなければならない。
side ='L' の場合は、左固有ベクトルが変換される。
side ='R' の場合は、右固有ベクトルが変換される。

n INTEGER。固有ベクトルで構成されるマトリックスの行数 ($n \geq 0$)。

ilo, ihi INTEGER。*ilo* と *ihi* の値は、?gebal から返された値である ($n > 0$ の場合は $1 \leq ilo \leq ihi \leq n$ で、$n = 0$ の場合は $ilo = 1$ かつ $ihi = 0$)

scale REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
配列 DIMENSION は、max(1, *n*) 以上。
元の一般マトリックスを平衡化する際に使った置換やスケーリング係数の各成分 (?gebal から返されたもの) を格納する。

m INTEGER。固有ベクトルで構成されるマトリックスの列数 ($m \geq 0$)。

v REAL (sgebak の場合)
DOUBLE PRECISION (dgebak の場合)
COMPLEX (cgebak の場合)
DOUBLE COMPLEX (zgebak の場合)
配列 :
v(*ldv*, *) には、変換対象の左または右の固有ベクトルで構成されるマトリックスを格納する。
v の第 2 次元は、max(1, *m*) 以上でなければならない。

ldv INTEGER。*v* の第 1 次元。max(1, *n*) 以上。

出力パラメータ

v 変換後の固有ベクトルによって上書きされる。

info INTEGER
info = 0 の場合は、正常に終了する。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

このルーチンでの誤差は、無視できる程度である。

浮動小数演算のおおよその総数は、*m***n* に比例する。

?hseqr

Hessenberg 形式に縮退された
マトリックスのすべての固有値と
(オプションで) Schur 因子分解を
計算する。

```
call shseqr (job, compz, n, ilo, ihi, h, ldh, wr, wi, z, ldz, work,
             lwork, info)
call dhseqr (job, compz, n, ilo, ihi, h, ldh, wr, wi, z, ldz, work,
             lwork, info)
call chseqr (job, compz, n, ilo, ihi, h, ldh, w, z, ldz, work, lwork,
             info)
call zhseqr (job, compz, n, ilo, ihi, h, ldh, w, z, ldz, work, lwork,
             info)
```

説明

このルーチンでは、上 Hessenberg マトリックス H: $H = ZTZ^H$ のすべての固有値と (オプションで) Schur 因子分解を計算する。T は、上三角 (実数型の場合は準三角) マトリックス (H の Schur 形式) である。Z は、ユニタリ・マトリックスまたは直交マトリックスで、列は Schur ベクトル z_i である。

このルーチンを使えば、上 Hessenberg 形式 H:$A = QHQ^H$ (Q はユニタリ・マトリックスで、実数型の場合は直交マトリックス) に縮退されている一般マトリックス A の Schur 因子分解を求められる。
$A = (QZ)T(QZ)^H$ である。

この場合は、`?gehrd` (5-178 ページを参照) によって A を Hessenberg 形式に縮退させた後、`?orghr` を呼び出して Q を明示的に求め (5-180 ページを参照)、`compz` ='`V`' と指定して Q を `?hseqr` に渡す。

`?gebal` (5-190 ページを参照) を呼び出せば、`?hseqr` を使って Hessenberg 形式に縮退させる前に、元のマトリックスを平衡化できる。Hessenberg のマトリックス H は、次の構造を持つ。

$$\begin{bmatrix} H_{11} & H_{12} & H_{13} \\ 0 & H_{22} & H_{23} \\ 0 & 0 & H_{33} \end{bmatrix}$$

H_{11} と H_{33} は、上三角マトリックスである。

この場合は、中央の対角ブロック H_{22} (*ilo* ～ *ihi* の行と列) だけをさらに Schur 形式に縮退させる必要がある (H_{12} と H_{23} のブロックも影響を受ける)。したがって、*ilo* と *ihi* の値を、?hseqr に直接与えられる。また、このルーチンを呼び出した後は、?gebak (5-193 ページを参照) を呼び出して、平衡化後のマトリックスの Schur ベクトルを置換して、元のマトリックスの Schur ベクトルにしなければならない。

ただし、?gebal を呼び出さなかった場合は、*ilo* に 1 を、*ihi* に *n* を設定しなければならない。*A* の Schur 因子分解が必要な場合は、*job* ='S' または 'B' を指定して ?gebal を呼び出してはならない。つまり、これらを指定すると、平衡化のための変換がユニタリ (実数型の場合は直交) にならないからである。

?hseqr では、上 Hessenberg の *QR* アルゴリズムのマルチシフト形式を使う。Schur ベクトルは、$\|z_i\|_2 = 1$ になるように正規化される。ただし、絶対値が 1 となる複素数の係数 (実数の場合は係数 ± 1) の範囲に収まる。

入力パラメータ

job
CHARACTER*1。'E' または 'S' でなければならない。
job ='E の場合は、固有値だけを求める。
job ='S' の場合は、Schur 形式 *T* を求める。

compz
CHARACTER*1。'N'、'I'、または 'V' でなければならない。
compz ='N' の場合は、Schur ベクトルは計算されない (配列 *z* は参照されない)。
compz ='I' の場合は、*H* の Schur ベクトルが計算される (配列 *z* は、このルーチンによって初期化される)。
compz ='V' の場合は、*A* の Schur ベクトルが計算される (配列 *z* にマトリックス *Q* を設定しなければならない)。

n
INTEGER。マトリックス *H* の次数 ($n \geq 0$)。

ilo, ihi
INTEGER。*A* の平衡化を ?gebal によって実行した場合は、*ilo* と *ihi* に、?gebal から返された値を設定しなければならない。そうでない場合は、*ilo* に 1 を、*ihi* に *n* を設定しなければならない。

h, z, work
REAL (shseqr の場合)
DOUBLE PRECISION (dhseqr の場合)
COMPLEX (chseqr の場合)
DOUBLE COMPLEX (zhseqr の場合)

配列 :
h(*ldh*, *) *n* × *n* の上 Hessenberg マトリックス *H*。
h の第 2 次元は、max(1, *n*) 以上でなければならない。
z(*ldz*, *)
compz = 'V' の場合、*z* には、Hessenberg 形式に縮退させたマトリックス *Q* を設定しなければならない。
compz = 'I' の場合、*z* を設定する必要はない。
compz = 'N' の場合は、*z* は参照されない。
z の第 2 次元は、max(1, *n*) 以上 (*compz* = 'V' または 'I' の場合)、あるいは 1 以上 (*compz* = 'N' の場合) でなければならない。
work(*lwork*) は、ワークスペース配列である。
work の次元は、次の値以上でなければならない。
max(1, *n*)

ldh INTEGER。*h* の第 1 次元。max(1, *n*) 以上。

ldz INTEGER。*z* の第 1 次元。
compz = 'N' の場合は、*ldz* ≥ 1
compz = 'V' または 'I' の場合は、*ldz* ≥ max(1, *n*)

lwork INTEGER。配列 *work* の次数。*lwork* ≥ max(1, *n*)*lwork* = -1 の場合はワークスペースのクエリとみなされ、ルーチンは *work* 配列の最適サイズだけを計算し、その値を *work* 配列の最初のエントリとして返し、xerbla は *lwork* に関するエラー・メッセージを生成しない。

出力パラメータ

w COMPLEX (chseqr の場合)
DOUBLE COMPLEX (zhseqr の場合)
配列 DIMENSION は、max(1, *n*) 以上。
info>0 でない場合は、計算で求めた固有値が格納される。一連の固有値は、Schur 形式 *T* (求めるように指定した場合) の対角成分と同じ順序で格納される。

wr, *wi* REAL (shseqr の場合)
DOUBLE PRECISION (dhseqr の場合)
配列 DIMENSION はそれぞれ、max(1, *n*) 以上。
info > 0 でない場合は、計算で求めた固有値の実数部と虚数部がそれぞれ格納される。固有値の複素共役ペアは、虚数部が正である固有値に続いて格納される。一連の固有値は、Schur 形式 *T* (求めるように指定した場合) の対角成分と同じ順序で格納される。

z	*compz* = 'V' または 'I' の場合は、*info* > 0 でない限り、Schur ベクトルを求めるように指定したときには、Schur ベクトルのユニタリ (直交) マトリックスが *z* に格納される。 *compz* = 'N' の場合は、*z* は参照されない。
work(1)	終了時に *info* = 0 ならば、*work(1)* に最適な *lwork* 値が返される。
info	INTEGER *info* = 0 の場合は、正常に終了する。 *info* = -*i* の場合は、*i* 番目のパラメータの値が不正である。 *info* > 0 の場合は、このアルゴリズムで 30 (*ihi*−*ilo* + 1) 回の処理を繰り返した結果、すべての固有値を見つけられなかった。*info* = *i* の場合は、見つかった固有値の実数部と虚数部が、*wr* と *wi* の 1, 2, ..., *ilo*-1 および i+1, i+2, ..., *n* の成分に格納される。

アプリケーション・ノート

計算で求めた Schur 因子分解は、近似マトリックス $H + E$ の正確な因子分解である。
$\|E\|_2 < O(\varepsilon)\ \|H\|_2/s_i$ であり、ε はマシン精度である。
λ_i が正確な固有値であり、μ_i がそれに対応する計算で求めた値だとすると、$|\lambda_i - \mu_i| \le c(n)\varepsilon\ \|H\|_2/s_i$ になる。$c(n)$ は、(n の) 漸増する関数である。s_i は、λ_i の条件数の逆数である。条件数 s_i は、?trsna (5-210 ページを参照) を呼び出して、求められる。

浮動小数演算の総数は、このアルゴリズムの収束の速さによって決まり、通常は、次の数値になる。

固有値だけを計算する場合 :	$7n^3$ (実数の場合) $25n^3$ (複素数の場合)
Schur 形式を計算する場合 :	$10n^3$ (実数の場合) $35n^3$ (複素数の場合)
Schur の因子分解をすべて計算する場合 :	$20n^3$ (実数の場合) $70n^3$ (複素数の場合)

?hsein

指定された固有値に対応する上 Hessenberg マトリックスの固有ベクトルを選択して計算する。

```
call shsein ( job, eigsrc, initv, select, n, h, ldh, wr, wi, vl, ldvl,
              vr, ldvr, mm, m, work, ifaill, ifailr, info )
call dhsein ( job, eigsrc, initv, select, n, h, ldh, wr, wi, vl, ldvl,
              vr, ldvr, mm, m, work, ifaill, ifailr, info )
call chsein ( job, eigsrc, initv, select, n, h, ldh, w, vl, ldvl, vr,
              ldvr, mm, m, work, rwork, ifaill, ifailr, info )
call zhsein ( job, eigsrc, initv, select, n, h, ldh, w, vl, ldvl, vr,
              ldvr, mm, m, work, rwork, ifaill, ifailr, info )
```

説明

このルーチンでは、選択された固有値に対応する、上 Hessenberg マトリックス H の左または右の固有ベクトルを計算する。

右の固有ベクトル x と左の固有ベクトル y (固有値 λ に対応) は、$Hx = \lambda x$ と $y^H H = \lambda y^H$ (または $H^H y = \lambda^* y$) として定義される。λ* は、λ の共役を表す。

一連の固有ベクトルは、反転を繰り返すことによって計算される。そして、実数型の固有ベクトル x の場合は $\max|x_i| = 1$ で、複素数型の固有ベクトルの場合は $\max(|\mathrm{Re}x_i| + |\mathrm{Im}x_i|) = 1$ になるようにスケーリングされる。

一般マトリックス A を上 Hessenberg 形式に縮退させて H を求めた場合は、?ormhr (5-182 ページ) または ?unmhr (5-187 ページ) によって、H の固有ベクトルを A の固有ベクトルに変換できる。

入力パラメータ

job
: CHARACTER*1。'R' 、'L' 、または 'B' でなければならない。
job ='R' の場合は、右の固有ベクトルだけが計算される。
job ='L' の場合は、左の固有ベクトルだけが計算される。
job ='B' の場合は、すべての固有ベクトルが計算される。

eigsrc CHARACTER*1。'Q' または 'N' でなければならない。
eigsrc ='Q' の場合は、?hseqr (5-195 ページを参照) を使って、*H* の固有値が求められている。そのため、*H* の劣対角成分の中にゼロのものがある (つまり、ブロック三角である) 場合は、*j* 番目の固有値を *j* 番目の行と列が格納されているブロックの固有値とみなせる。このルーチンでは、この特性により、1 つの対角ブロックに関してだけ反転の繰り返し処理を実行できる。
eigsrc ='N' の場合は、そのような仮定を行わず、このルーチンではマトリックス全体を使って反転の繰り返し処理を実行する。

initv CHARACTER*1。'N' または 'U' でなければならない。
initv ='N' の場合は、選択した固有ベクトルに対して初期の見積もりを指定しない。
initv ='U' の場合は、選択した固有ベクトルに対して初期の見積もりを *vl* や *vr* で指定する。

select LOGICAL。
配列 DIMENSION は、max(1, *n*) 以上。
計算の対象となる固有ベクトルを指定する。
実数の場合 :
実数の固有値 *wr(j)* に対応する実数型の固有ベクトルを求めるには、*select(j)* に .TRUE. を設定する。複素数の固有値 (*wr(j),wi(j)*) (複素共役は (*wr(j+1),wi(j+1)*))に対応する複素数型の固有ベクトルを選択するには、*select*(*j*) や *select*(*j*+1) に .TRUE. を設定する。ペアの最初の固有値に対応する固有ベクトルが計算される。
複素数の場合 :
固有値 *w(j)* に対応する固有ベクトルを選択するには、*select(j)* に .TRUE. を設定する。

n INTEGER。マトリックス *H* の次数 (*n* ≥ 0)。

h, vl, vr, work REAL (shsein の場合)
DOUBLE PRECISION (dhsein の場合)
COMPLEX (chsein の場合)
DOUBLE COMPLEX (zhsein の場合)

配列 :
h(*ldh*, *)*n* × *n* の上 Hessenberg マトリックス *H*。
h の第 2 次元は、max(1, *n*) 以上でなければならない。

vl(*ldvl*, *)
initv = 'V' かつ *job* = 'L' または 'B' の場合は、左固有ベクトルの反転の繰り返し処理用の開始ベクトルを *vl* に格納しなければならない。それぞれの開始ベクトルは、対応する固有ベクトルを格納する際に使うのと同じ列 (1 つまたは複数個) に格納しなければならない。
initv = 'N' の場合、*vl* を設定する必要はない。
vl の第 2 次元は、max(1, *mm*) 以上 (*job* = 'L' または 'B' の場合)、あるいは 1 以上 (*job* = 'R' の場合) でなければならない。
配列 *vl* は、*job* = 'R' の場合は、参照されない。

vr(*ldvr*, *)
initv = 'V' かつ *job* = 'R' または 'B' の場合は、右固有ベクトルの反転の繰り返し処理用の開始ベクトルを *vr* に格納しなければならない。それぞれの開始ベクトルは、対応する固有ベクトルを格納する際に使うのと同じ列 (1 つまたは複数個) に格納しなければならない。
initv = 'N' の場合、*vr* を設定する必要はない。
vr の第 2 次元は、max(1, *mm*) 以上 (*job* = 'R' または 'B' の場合)、あるいは 1 以上 (*job* = 'L' の場合) でなければならない。
配列 *vr* は、*job* = 'L' の場合は、参照されない。

work(*) は、ワークスペース配列である。
DIMENSION は、max(1, *n**(*n*+2)) 以上 (実数の場合)、あるいは max(1, *n***n*) 以上 (複素数の場合) である。

ldh INTEGER。*h* の第 1 次元。max(1, *n*) 以上。

w COMPLEX (chsein の場合)
DOUBLE COMPLEX (zhsein の場合)
配列 DIMENSION は、max(1, *n*) 以上。
マトリックス *H* の固有値を格納する。
eigsrc = 'Q' の場合、この配列は ?hseqr から返されたものと同じでなければならない。

wr, wi
REAL (shsein の場合)
DOUBLE PRECISION (dhsein の場合)
配列 DIMENSION はそれぞれ、max(1, *n*) 以上。
マトリックス *H* の固有値の実数部と虚数部をそれぞれ格納する。複素共役ペアの値は、この配列の連続する成分に格納しなければならない。*eigsrc* ='Q' の場合、この配列は、?hseqr から返されたものと同じでなければならない。

ldvl
INTEGER。*vl* の第 1 次元。
job ='L' または 'B' の場合は、*ldvl* ≥ max(1, *n*)
job ='R' の場合は、*ldvl* ≥ 1

ldvr
INTEGER。*vr* の第 1 次元。
job ='R' または 'B' の場合は、*ldvr* ≥ max(1, *n*)
job ='L' の場合は、*ldvr* ≥ 1

mm
INTEGER。*vl* や *vr* の列数。
m (実際に必要な列数。下記の**出力パラメータ**を参照) 以上でなければならない。
実数の場合は、選択された実数の固有値ごとに 1 を、選択された複素数の固有値ごとに 2 をカウントすれば、*m* が得られる (*select* を参照)。
複素数の場合は、選択された固有ベクトルの個数が *m* になる (*select* を参照)。次の制約がある。
0 ≤ *mm* ≤ *n*

rwork
REAL (chsein の場合)
DOUBLE PRECISION (zhsein の場合)
配列 DIMENSION は、max(1, *n*) 以上。

出力パラメータ

select
select 実数の場合だけ、上書きされる。上記のように複素数の固有値が選択された場合は、*select*(*j*) に .TRUE. が、*select*(*j*+1) に .FALSE. が設定される。

w
w の一部の成分の実数部が変動する場合がある。これは、独立した固有ベクトルを検索する際に、近似した固有値がわずかに摂動するからである。

wr

wr の一部の成分が変動する場合がある。これは、独立した固有ベクトルを検索する際に、近似した固有値がわずかに摂動するからである。

vl, vr

job ='L' または 'B' の場合、*vl* には、(*select* の指定に従って) 計算された左の固有ベクトルが格納される。

job ='R' または 'B' の場合、*vr* には、(*select* の指定に従って) 計算された右の固有ベクトルが格納される。

一連の固有ベクトルは、対応する固有値と同じ順序で、この配列の列に連続して格納される。

実数の場合 : 選択された実数の固有値に対応する実数型の固有ベクトルによって列を 1 つ占める。選択された固有値に対応する複素数型の固有ベクトルによって列を 2 つ占める。最初の列には実数部が、2 番目の列には虚数部が格納される。

m

INTEGER。**実数の場合** : 選択された固有ベクトルの格納に必要な *vl* や *vr* の列数。

複素数の場合 : 選択された固有ベクトルの個数。

ifaill, ifailr

INTEGER

配列 DIMENSION はそれぞれ、max(1, *mm*) 以上。

ifaill(i) = 0 (*vl* の *i* 番目の列が収束した場合)

ifaill(i) = *j* > 0 (*vl* の *i* 番目の列に格納されている固有ベクトル (*j* 番目の固有値に対応) が収束しなかった場合)

ifailr(i) = 0 (*vr* の *i* 番目の列が収束した場合)

ifailr(i) = *j* > 0 (*vr* の *i* 番目の列に格納されている固有ベクトル (*j* 番目の固有値に対応) が収束しなかった場合)

実数の場合 : *vl* の *i* 番目と (*i*+1) 番目の列に選択した複素数型の固有ベクトルが格納されている場合は、*ifaill(i)* と *ifaill*(*i*+1) に同じ値が設定される。*vr* と *ifailr* にも、これと同様の規則が適用される。

job ='R' の場合は、配列 *ifaill* は参照されない。

job ='L' の場合は、配列 *ifailr* は参照されない。

info INTEGER

info = 0 の場合は、正常に終了する。

info = -*i* の場合は、*i* 番目のパラメータの値が不正である。

info > 0 の場合は、*i* 個の固有ベクトル (上記のパラメータ *ifaill* や *ifailr* で指定) が、収束していない。*vl* や *vr* の対応する列に、有用な情報が格納されていない。

アプリケーション・ノート

計算で求めた右固有ベクトル x_i はそれぞれ、近似マトリックス $A + E_i$ の正確な固有ベクトル ($\|E_i\| < O(\varepsilon)\|A\|$) である。そのため、誤差は小さいものになる ($\|Ax_i - \lambda_i x_i\| = O(\varepsilon)\|A\|$)。

ただし、近似したあるいは一致した固有値に対応する固有ベクトルが、関連する部分空間に正確に張られていない場合がある。

計算で求めた左固有ベクトルについても、これと同様の注意が適用される。

?trevc

?hseqr で求めた上(準)三角マトリックスの
固有ベクトルを選択して計算する。

```
call strevc ( side, howmny, select, n, t, ldt, vl, ldvl, vr, ldvr, mm,
              m, work, info )
call dtrevc ( side, howmny, select, n, t, ldt, vl, ldvl, vr, ldvr, mm,
              m, work, info )
call ctrevc ( side, howmny, select, n, t, ldt, vl, ldvl, vr, ldvr, mm,
              m, work, rwork, info )
call ztrevc ( side, howmny, select, n, t, ldt, vl, ldvl, vr, ldvr, mm,
              m, work, rwork, info )
```

説明

このルーチンでは、上三角マトリックス T (実数の場合は、上準三角マトリックス T) の左あるいは右の一部またはすべての固有ベクトルを計算する。このタイプのマトリックスは、一般マトリックスの Schur 因子分解 ($A = Q\,T\,Q^H$) で生成される (?hseqr (5-195 ページを参照) で計算される)。

T の固有値 w に対応する右固有ベクトル x と左固有ベクトル y は、次のように表される。

$$T\,x = w\,x\ , \qquad y^H T = w\,y^H$$

y^H は y の共役転置を表す。

固有値は、このルーチンに入力されず、T の対角ブロックから直接読み込まれる。

このルーチンは、T の右と左固有ベクトルが入ったマトリックス X と Y を返すか、積 $Q\,X$ と積 QY を返す (Q は入力マトリックス)。
Q が、マトリックス A を Schur 形式 T に縮退する直交 / ユニタリ・マトリックスの場合は、QX と QY は、A の右と左の固有ベクトルからなるマトリックスである。

入力パラメータ

side
CHARACTER*1。'R'、'L'、または 'B' でなければならない。
side ='R' の場合は、右の固有ベクトルだけが計算される。
side ='L' の場合は、左の固有ベクトルだけが計算される。
side ='B' の場合は、すべての固有ベクトルが計算される。

howmny
CHARACTER*1。'A'、'B'、または 'S' でなければならない。
howmny ='A' の場合は、(*side* の指定に従って) すべての固有ベクトルが計算される。
howmny ='B' の場合は、(*side* の指定に従って) すべての固有ベクトルが計算され、*vl* と *vr* に与えられるマトリックスによって変換が戻される。
howmny ='S' の場合は、(*side* と *select* の値に従って) 選択した固有ベクトルが計算される。

select
LOGICAL。
配列 DIMENSION は max(1, *n*) 以上。
howmny ='S' の場合、*select* は計算すべき固有ベクトルを指定する。
howmny ='A' または 'B' の場合、*select* は参照されない。
実数の場合 :
ω_j が実固有値の場合、*select*(*j*) が .TRUE. ならば、対応する実固有ベクトルが計算される。
ω_j と ω_{j+1} が複素固有値の実部と虚部の場合、*select*(*j*) または *select*(*j*+1) が .TRUE. ならば、対応する複素固有ベクトルが計算され、終了時に *select*(*j*) は .TRUE.、*select*(*j*+1) は .FALSE. に設定される。
複素数の場合 :
select(*j*) が .TRUE. ならば、*j* 番目の固有値に対応する固有ベクトルが計算される。

n
INTEGER。マトリックス *T* の次数 ($n \geq 0$)。

t, vl, vr, work
REAL (strevc の場合)
DOUBLE PRECISION (dtrevc の場合)
COMPLEX (ctrevc の場合)
DOUBLE COMPLEX (ztrevc の場合)
配列 :

t(ldt, *) には、n × n のマトリックス T を標準の Schur 形式で格納する。
t の第 2 次元は、max(1, n) 以上でなければならない。

vl(ldvl, *)
howmny='B' かつ side='L' または 'B' の場合、vl には、n × n のマトリックス Q (通常は、?hseqr から返された Schur ベクトルから成るマトリックス) を格納しなければならない。
howmny='A' または 'S' の場合、vl を設定する必要はない。
vl の第2次元は、max(1, mm)以上(side='L' または 'B' の場合)、あるいは 1 以上 (side='R' の場合) でなければならない。
配列 vl は、side='R' の場合は、参照されない。

vr (ldvr, *)
howmny='B' かつ side='R' または 'B' の場合、vr には、n × n のマトリックス Q (通常は、?hseqr から返された Schur ベクトルから成るマトリックス) を格納しなければならない。
howmny='A' または 'S' の場合、vr を設定する必要はない。
vr の第2次元は、max(1, mm)以上(side='R' または 'B' の場合)、あるいは 1 以上 (side='L' の場合) でなければならない。
配列 vr は、side='L' の場合は、参照されない。

work(*) は、ワークスペース配列である。
DIMENSION は、max(1, 3*n) 以上 (実数の場合)、あるいは max(1, 2*n) 以上 (複素数の場合) である。

ldt — INTEGER。t の第 1 次元。max(1, n) 以上。

ldvl — INTEGER。vl の第 1 次元。
side='L' または 'B' の場合は、ldvl ≥ max(1, n)
side='R' の場合は、ldvl ≥ 1

ldvr — INTEGER。vr の第 1 次元。
side='R' または 'B' の場合は、ldvr ≥ max(1, n)
side='L' の場合は、ldvr ≥ 1

mm — INTEGER。配列 vl や vr の列数。m (必要な列の正確な個数) 以上でなければならない。howmny='A' または 'B' の場合は、m = n である。
howmny='S' の場合 : **実数の場合は**、選択された実数型の固有ベクトルごとに 1 を、選択された複素数型の固有ベクトルごとに 2 をカウントすれば、m が得られる。

複素数の場合は、選択された固有ベクトルの個数 (*select* を参照) が *m* になる。次の制約がある。
$0 \le m \le n$

rwork REAL (ctrevc の場合)
DOUBLE PRECISION (ztrevc の場合)
ワークスペース配列 DIMENSION は、max(1, *n*) 以上。

出力パラメータ

select 上記のように実マトリックスの複素数型の固有ベクトルを選択した場合は、*select*(*j*) に .TRUE. が、*select*(*j*+1) に .FALSE. が設定される。

vl, vr *side* ='L' または 'B' の場合は、*vl* に、(*howmny* と *select* の指定に従って) 計算で求めた左固有ベクトルが格納される。
side ='R' または 'B' の場合は、*vr* に、(*howmny* と *select* の指定に従って) 計算で求めた右固有ベクトルが格納される。
一連の固有ベクトルは、対応する固有値と同じ順序で、この配列の列に連続して格納される。
実数の場合 : 実数の固有値のそれぞれに実数型の固有ベクトルが 1 つ対応し、列を 1 つ占める。固有値の複素共役ペアのそれぞれに複素数型の固有ベクトルが 1 つ対応し、列を 2 つ占める。最初の列には実数部、2 番目の列には虚数部が格納される。

m INTEGER
複素数の場合 : 選択した固有ベクトルの個数。
howmny ='A' または 'B' の場合、*m* には *n* が設定される。
実数の場合 : 選択した固有ベクトルの格納に実際に使用される *vl* や *vr* の列数。
howmny ='A' または 'B' の場合、*m* には *n* が設定される。

info INTEGER。*info* = 0 の場合は、正常に終了する。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

x_i が正確な固有ベクトルであり、y_i がそれに対応する計算で求めた固有ベクトルである場合、それらの間の角 $\theta(y_i, x_i)$ は、$\theta(y_i, x_i) \le (c(n)\varepsilon\|T\|_2)/\mathrm{sep}_i$ (sep_i は x_i の条件数の逆数) のように制限される。条件数 sep_i は、`?trsna` を呼び出すことによって求められる。

?trsna

上 (準) 三角マトリックスの指定された
固有値および右固有ベクトルに関して
条件数を見積もる。

```
call strsna ( job, howmny, select, n, t, ldt, vl, ldvl, vr, ldvr, s,
              sep, mm, m, work, ldwork, iwork, info )
call dtrsna ( job, howmny, select, n, t, ldt, vl, ldvl, vr, ldvr, s,
              sep, mm, m, work, ldwork, iwork, info )
call ctrsna ( job, howmny, select, n, t, ldt, vl, ldvl, vr, ldvr, s,
              sep, mm, m, work, ldwork, rwork, info )
call ztrsna ( job, howmny, select, n, t, ldt, vl, ldvl, vr, ldvr, s,
              sep, mm, m, work, ldwork, rwork, info )
```

説明

このルーチンでは、上三角マトリックス T (実数の場合は、標準形の Schur 形式の上準三角マトリックス T) の指定された固有値や右固有ベクトルに関して、条件数を見積もる。これらは、元のマトリックス $A = ZTZ^H$ (Z はユニタリ・マトリックス。ただし、実数の場合は直交マトリックス) の固有値と右固有ベクトルの条件数と同じである。ここから T を導出できる。

このルーチンでは、固有値 λ_i の条件数の逆数を $s_i = |v^H u|/(\|u\|_E \|v\|_E)$ として計算する。u と v はそれぞれ、λ_i に対応する T の右と左の固有ベクトルである。この条件数の逆数は、常に、ゼロ (悪い条件の場合) と 1 (良い条件の場合) の間になる。

計算で求めた固有値 λ_i のおおよその誤差は、$\varepsilon\|T\|/s_i$ (ε は**マシン精度**) と見積もれる。

λ_i に対応する右固有ベクトルの条件数の逆数を見積もるために、このルーチンではまず ?trexc (5-215 ページを参照) を呼び出して一連の固有値の順序を変更し、λ_i が先頭の位置にする。

$$T = Q\begin{bmatrix}\lambda_i & C^H \\ 0 & T_{22}\end{bmatrix}Q^H$$

次に、この固有ベクトルの条件数の逆数を、sep_i (マトリックス T_{22} の最小の特異値 $-\lambda_i I$) として見積もる。この値は、ゼロ (悪い条件の場合) から非常に大きな値 (良い条件の場合) の間になる。

λ_i に対応して計算で求めた右固有ベクトル u のおおよその誤差は、$\varepsilon \|T\|/\mathrm{sep}_i$ と見積もれる。

入力パラメータ

job
: CHARACTER*1。'E'、'V' 、または 'B' でなければならない。
job ='E' の場合は、固有値だけの条件数が計算される。
job ='V' の場合は、固有ベクトルだけの条件数が計算される。
job ='B' の場合は、固有値と固有ベクトルの両方の条件数が計算される。

howmny
: CHARACTER*1。'A' または 'S' でなければならない。
howmny ='A' の場合は、すべての固有ペアの条件数が計算される。
howmny ='S' の場合は、選択した固有ペア (*select* で指定) の条件数が計算される。

select
: LOGICAL。
配列 DIMENSION (*howmny* ='S' の場合は max(1, *n*) 以上で、そうでない場合は 1 以上)。
howmny = 'S' の場合は、条件数を計算する固有ペアを指定する。
実数の場合 :
実数の固有値 λ_j に対応する固有ペアの条件数を選択するには、*select*(j) に .TRUE. を設定しなければならない。固有値 λ_j と λ_{j+1} の複素共役ペアに対応する固有ペアの条件数を選択するには、*select*(j) や *select*(j+1) に .TRUE. を設定しなければならない。

複素数の場合 :
固有値 λ_j に対応する固有ペアの条件数を選択するには、*select*(*j*) に .TRUE. を設定しなければならない。
select は、*howmny* = 'A' の場合は、参照されない。

n INTEGER。マトリックス *T* の次数 ($n \geq 0$)。

t, vl, vr, work REAL (strsna の場合)
DOUBLE PRECISION (dtrsna の場合)
COMPLEX (ctrsna の場合)
DOUBLE COMPLEX (ztrsna の場合)
配列 :
*t(ldt, *)* には、$n \times n$ のマトリックス *T* を格納する。
t の第 2 次元は、max(1, *n*) 以上でなければならない。
*vl(ldvl, *)*
job = 'E' または 'B' の場合、*vl* には、*howmny* と *select* で指定した固有ペアに対応する *T* (または *Q* がユニタリ・マトリックスか直交マトリックスのような任意のマトリックス QTQ^H) の左固有ベクトルを格納しなければならない。一連の固有ベクトルは、?trevc または ?hsein から返されたとおりに、*vl* の連続した列に格納しなければならない。
vl の第 2 次元は、max(1, *mm*) 以上 (*job* = 'E' または 'B' の場合)、あるいは 1 以上 (*job* = 'V' の場合) でなければならない。
配列 *vl* は、*job* = 'V' の場合は、参照されない。
*vr(ldvr, *)*
job = 'E' または 'B' の場合、*vr* には、*howmny* と *select* で指定した固有ペアに対応する *T* (または *Q* がユニタリ・マトリックスか直交マトリックスのような任意のマトリックス QTQ^H) の右固有ベクトルを格納しなければならない。一連の固有ベクトルは、?trevc または ?hsein から返されたとおりに、*vr* の連続した列に格納しなければならない。
vr の第 2 次元は、max(1, *mm*) 以上 (*job* = 'E' または 'B' の場合)、あるいは 1 以上 (*job* = 'V' の場合) でなければならない。
配列 *vr* は、*job* = 'V' の場合は、参照されない。

work(ldwork, *) は、ワークスペース配列である。
work の第 2 次元は、job='V' または 'B' の場合、
max(1, n+1) 以上 (複素数の場合)、あるいは
max(1, n+6) 以上 (実数の場合) でなければならない。
job='E' の場合は、1 以上でなければならない。
配列 work は、job='E' の場合は、参照されない。

ldt　INTEGER。t の第 1 次元。max(1, n) 以上。

ldvl　INTEGER。vl の第 1 次元。
job='E' または 'B' の場合は、ldvl≥max(1, n)
job='V' の場合は、ldvl≥1

ldvr　INTEGER。vr の第 1 次元。
job='E' または 'B' の場合は、ldvr≥max(1, n)
job='R の場合は、ldvr≥1

mm　INTEGER。配列 s と sep の成分の個数、または vl と vr の列数 (使う場合)。m (必要とする正確な個数) 以上でなければならない。
howmny='A' の場合 m = n である。
howmny='S' の場合、**実数のときは**、選択された実数の固有値ごとに 1 を、固有値の選択された複素共役ペアごとに 2 をカウントすれば、m が得られる。
複素数のときは、選択された固有ペア (select を参照) の個数が m になる。次の制約がある。0 ≤ m ≤ n

ldwork　INTEGER。work の第 1 次元。
job='V' または 'B' の場合は、ldwork ≥ max(1, n)
job='E' の場合は、ldwork ≥ 1

rwork　REAL (ctrsna、ztrsna の場合)
配列 DIMENSION は、max(1, n) 以上。

iwork　INTEGER (strsna、dtrsna の場合)
配列 DIMENSION は、max(1, n) 以上。

出力パラメータ

s　REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
配列 DIMENSION (job='E' または 'B' の場合は max(1, mm) 以上、job='V' の場合は 1 以上)。

job='E' または 'B' の場合は、選択した固有値の条件数の逆数が、この配列の連続する成分に格納される。そのため、s(j)、sep(j)、および vl と vr の j 番目の列はすべて、同じ固有ペアに対応しています (ただし、選択しなかった固有ペアが存在する場合は、一般に、j 番目の固有ペアにはならない)。**実数の場合**: 固有値の複素共役ペアでは、S の 2 つの連続する成分に同じ値が設定される。
配列 s は、job='V' の場合は、参照されない。

sep

REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
配列 DIMENSION は、max(1, mm) 以上。
(job='V' または 'B' の場合)、あるいは 1 以上 (job='E' の場合) である。
job='V' または 'B' の場合は、選択した右固有ベクトルについての見積もりの条件数の逆数が、この配列の連続する成分に格納される。
実数の場合: 複素数型の固有ベクトルのときは、sep の 2 つの連続する成分に同じ値が設定される。sep(j) を計算するために一連の固有値の順序を変更できない場合は、sep(j) にゼロが設定される。これは、真の値が非常に小さい場合にだけ発生する可能性がある。
配列 sep は、job='E' の場合は、参照されない。

m

INTEGER
複素数の場合: 選択した固有ペアの個数。howmny='A の場合は、m に n が設定される。
実数の場合: 条件数の見積もり値の格納に実際に使用される s や sep の成分の個数。howmny='A' の場合は、m に n が設定される。

info

INTEGER
info = 0 の場合は、正常に終了する。
info = -i の場合は、i 番目のパラメータの値が不正である。

アプリケーション・ノート

計算で求めた値 sep_i は、真の値を過大見積もりする場合もあるが、3 を超える係数になるのはほとんどない。

?trexc

一般マトリックスの Schur 因子分解の
順序を変更する。

```
call strexc ( compq, n, t, ldt, q, ldq, ifst, ilst, work, info )
call dtrexc ( compq, n, t, ldt, q, ldq, ifst, ilst, work, info )
call ctrexc ( compq, n, t, ldt, q, ldq, ifst, ilst, info )
call ztrexc ( compq, n, t, ldt, q, ldq, ifst, ilst, info )
```

説明

このルーチンでは、一般マトリックス $A= QTQ^H$ の Schur 因子分解の順序を変更し、T の行インデックス *ifst* の対角成分または対角ブロックを行 *ilst* に移動する。

順序を変更した後の Schur 形式 S は、ユニタリ (実数の場合は直交) 相似変換 $S = Z^HTZ$ によって計算される。また、Schur ベクトルの更新後のマトリックス P を、$P = QZ(A=PSP^H)$ として計算できる。

入力パラメータ

compq
: CHARACTER*1。'V' または 'N' でなければならない。
compq ='V' の場合は、Schur ベクトル (Q) が更新される。
compq ='N' の場合は、Schur ベクトルは更新されない。

n
: INTEGER。マトリックス T の次数 ($n \geq 0$)。

t, q
: REAL (strexc の場合)
DOUBLE PRECISION (dtrexc の場合)
COMPLEX (ctrexc の場合)
DOUBLE COMPLEX (ztrexc の場合)
配列 :
*t(ldt, *)* には、$n \times n$ のマトリックス T を格納する。
t の第 2 次元は、max(1, *n*) 以上でなければならない。
*q(ldq, *)*
compq ='V' の場合は、*q* に Q (Schurベクトル)を格納しなければならない。
compq ='N' の場合は、*q* は参照されない。

q の第 2 次元は、max(1, n) 以上 (compq='V' の場合)、あるいは 1 以上 (compq='N' の場合) でなければならない。

ldt　INTEGER。t の第 1 次元。max(1, n) 以上。

ldq　INTEGER。q の第 1 次元。
compq='N' の場合は、ldq≥1
compq='V' の場合は、ldq≥max(1, n)

ifst, ilst　INTEGER。$1 \le ifst \le n$; $1 \le ilst \le n$
マトリックス T の対角成分 (実数の場合は対角ブロック) の順序の変更を指定する。行インデックスが ifst の成分 (またはブロック) は、隣接する成分 (またはブロック) 間での一連の交換処理により、行 ilst に移される。

work　REAL (strexc の場合)
DOUBLE PRECISION (dtrexc の場合)
配列 DIMENSION は、max(1, n) 以上。

出力パラメータ

t　更新後のマトリックス S によって上書きされる。

q　compq='V' の場合は、q に、Schur ベクトルで構成される更新後のマトリックスが格納される。

ifst, ilst　実数の場合だけ、上書きされる。
このルーチンの開始時に、ifst が 2×2 のブロックの 2 行目を指している場合は、先頭行を指すように変更される。ilst は常に、最終位置 (入力値から ±1 だけ変動している場合もある) のブロックの先頭行を指す。

info　INTEGER
info = 0 の場合は、正常に終了する。
info = -i の場合は、i 番目のパラメータの値が不正である。

アプリケーション・ノート

計算で求めたマトリックス S は、マトリックス $T + E$ の近似値である。$\|E\|_2 = O(\varepsilon)\,\|T\|_2$ であり、ε はマシン精度である。

2 × 2 の対角ブロックが順序変更に関与する場合は、一般に、その対角線外の成分が変更される。そのブロックの条件が非常に悪い場合を除き、そのブロックの対角成分と固有値は変更されない。条件が非常に悪い場合は、それらが明確に変更される場合もある。2 × 2 のブロックは、2 つの 1 × 1 のブロックに分割できる。つまり、複素数の固有値ペアを、純粋な実数にできる。

ただし、固有値の値が、順序の変更によって変更されたりはしない。

浮動小数演算のおおよその総数は、次のようになる。

実数の場合 :	$6n(ifst-ilst)$ (`compq='N'` の場合)
	$12n(ifst-ilst)$ (`compq='V'` の場合)
複素数の場合 :	$20n(ifst-ilst)$ (`compq='N'` の場合)
	$40n(ifst-ilst)$ (`compq='V'` の場合)

?trsen

マトリックスの Schur 因子分解の順序を変更し、(オプションで) 固有値の選択した束についての条件数の逆数と不変部分空間を計算する。

```
call strsen (job, compq, select, n, t, ldt, q, ldq, wr, wi, m, s, sep,
             work, lwork, iwork, liwork, info)
call dtrsen (job, compq, select, n, t, ldt, q, ldq, wr, wi, m, s, sep,
             work, lwork, iwork, liwork, info)
call ctrsen (job, compq, select, n, t, ldt, q, ldq, w, m, s, sep,
             work, lwork, info)
call ztrsen (job, compq, select, n, t, ldt, q, ldq, w, m, s, sep,
             work, lwork, info)
```

説明

このルーチンでは、一般マトリックス $A = QTQ^H$ の Schur 因子分解の順序を変更して、固有値の選択した束が Schur 形式の主対角成分 (実数の場合は対角ブロック) にくるようにする。

順序を変更した後の Schur 形式 R は、ユニタリ (直交) 相似変換 $R = Z^H TZ$ によって計算される。また、Schur ベクトルの更新後のマトリックス P を、$P = QZ(A = PRP^H)$ として計算できる。

次のように仮定し、

$$R = \begin{bmatrix} T_{11} & T_{12} \\ 0 & T_{13} \end{bmatrix}$$

選択した固有値が、厳密な意味で $m \times m$ の先頭の部分マトリックス T_{11} の固有値であるとする。P を $(Q_1\ Q_2)$ のように対応させて分割し、Q_1 が Q の先頭から m 列を構成させる。これで、$AQ_1 = Q_1T_{11}$ になり、Q_1 の m 列は固有値の選択した束に対応する不変部分空間の直交基になる。

また、このルーチンでは、固有値の束と不変部分空間の平均の条件数の逆数を見積もれる。

入力パラメータ

job
: CHARACTER*1。'N'、'E'、'V'、または 'B' でなければならない。
job='N' の場合、条件数は計算されない。
job='E' の場合は、固有値の束の条件数だけが計算される。
job='V' の場合は、不変部分空間の条件数だけが計算される。
job='B' の場合は、束と不変部分空間の条件数が計算される。

compq
: CHARACTER*1。'V' または 'N' でなければならない。
compq='V' の場合は、Schur ベクトルの Q が更新される。
compq='N' の場合は、Schur ベクトルは更新されない。

select
: LOGICAL。
配列 DIMENSION は、max(1, n) 以上。
選択するクラスタに属する固有値を指定する。
固有値 λ_j を選択するには、*select*(j) が .TRUE. でなければならない。**実数の場合** : 固有値 λ_j と λ_{j+1} (対応する 2×2 の対角ブロック) の複素共役ペアを選択するには、*select*(j) や *select*(j+1) が .TRUE. でなければならない。複素共役 λ_j と λ_{j+1} は、束に両方とも含まれているか、あるいは両方とも束から除外されていなければならない。

n	INTEGER。マトリックス T の次数 ($n \geq 0$)。
t, q, work	REAL (strsen の場合) DOUBLE PRECISION (dtrsen の場合) COMPLEX (ctrsen の場合) DOUBLE COMPLEX (ztrsen の場合) 配列 : *t* (*ldt*, *) $n \times n$ の T *t* の第 2 次元は、max(1, *n*) 以上でなければならない。 *q* (*ldq*, *) *compq* = 'V' の場合は、*q* に Schur ベクトルの Q を格納しなければならない。 *compq* = 'N' の場合は、*q* は参照されない。 *q* の第 2 次元は、max(1, *n*) 以上 (*compq* = 'V' の場合)、あるいは 1 以上 (*compq* = 'N' の場合) でなければならない。 *work* (*lwork*) は、ワークスペース配列である。 **複素数の場合** : 配列 *work* は、*job* = 'N' の場合は参照されない。 必要なワークスペースの実際の量は、$n^2/4$ (*job* = 'E' の場合)、あるいは $n^2/2$ (*job* = 'V' または 'B' の場合) を超えることはできない。
ldt	INTEGER。*t* の第 1 次元。max(1, *n*) 以上。
ldq	INTEGER。*q* の第 1 次元。 *compq* = 'N' の場合は、*ldq* ≥ 1 *compq* = 'V' の場合は、*ldq* ≥ max(1, *n*)
lwork	INTEGER。配列 *work* の次元。 *job* = 'V' または 'B' の場合は、 *lwork* ≥ max(1, 2*m* (*n*−*m*)) *job* = 'E' の場合は、*lwork* ≥ max(1, *m* (*n*−*m*)) *job* = 'N' の場合は、*lwork* ≥ 1 (複素数のとき)、あるいは *lwork* ≥ max(1, *n*) (実数のとき) である。
iwork	INTEGER *iwork*(*liwork*) は、ワークスペース配列である。 配列 *iwork* は、*job* = 'N' または 'E' の場合は、参照されない。 *job* = 'V' または 'B' の場合、必要なワークスペースの実際の量は $n^2/2$ を超えることはできない。

liwork INTEGER
配列 *iwork* の次元。
job = 'V' または 'B' の場合は、
lwork ≥ max(1, 2*m*(*n*−*m*))
job = 'E' または 'E' の場合は、*lwork* ≥ 1

出力パラメータ

t 更新後のマトリックス *R* によって上書きされる。

q *compq* = 'V' の場合、*q* には、Schur ベクトルで構成される更新後のマトリックスが格納される。*Q* の先頭から m 個の列は、指定した不変部分空間の直交基を形成する。

w COMPLEX (ctrsen の場合)
DOUBLE COMPLEX (ztrsen の場合)
配列 DIMENSION は、max(1, *n*) 以上。
格納される *R* の固有値。一連の固有値は、*R* の対角成分と同じ順序で格納される。

wr, wi REAL (strsen の場合)
DOUBLE PRECISION (dtrsen の場合)
配列 DIMENSION は、max(1, *n*) 以上。
R の順序変更後の固有値の実数部と虚数部が、それぞれ格納される。一連の固有値は、*R* の対角成分と同じ順序で格納される。ただし、複素数の固有値の条件が非常に悪い場合には、その値が順序変更前から大幅に変更される場合がある。

m INTEGER
複素数の場合 : 指定した不変部分空間の個数であり、選択した固有値の個数と同じである (*select* を参照)。
実数の場合 : 指定した不変部分空間の次元。選択した実数の固有値ごとに 1 を、固有値の選択した複素共役ペアごとに 2 をカウントすれば、*m* の値が得られる (*select* を参照)。
次の制約がある。0 ≤ *m* ≤ *n*

s REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
job = 'E' または 'B' の場合は、*s* は、固有値の選択した束の平均の条件数の逆数の下限になる。*m* = 0 または *n* の場合は、*s* = 1 である。
実数の場合 : *info* = 1 のとき、*s* にはゼロが設定される。
s は、*job* = 'N' または 'V' の場合は、参照されない。

sep REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
job = 'V' または 'B' の場合、*sep* は、指定した不変部分空間の条件数の逆数の見積もりである。
m = 0 または *n* の場合は、*sep* = $\|T\|$ である。
実数の場合 : *info* = 1 のときは、*sep* にゼロが設定される。
sep は、*job* = 'N' または 'E' の場合は、参照されない。

work(1) 終了時に、*info* = 0 ならば、*lwork* に必要な最小値が *work(1)* に返される。

iwork(1) 終了時に、*info* = 0 ならば、*liwork* に必要な最小値が *iwork(1)* に返される。

info INTEGER
info = 0 の場合は、正常に終了する。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。

アプリケーション・ノート

計算で求めたマトリックス R は、マトリックス $T + E$ の近似値である。$\|E\|_2 = O(\varepsilon)\|T\|_2$ であり、ε はマシン精度である。
計算で求めた *s* は、$(\min(m, n\text{-}m))^{1/2}$ を超える係数で条件数の真の逆数を過小評価できない。*sep* は、真の値から $(m*n\text{-}m^2)^{1/2}$ だけ異なっている場合がある。計算で求めた不変部分空間と真の部分空間の間の角は、$O(\varepsilon)\ \|A\|_2/sep$ である。
2 × 2 の対角ブロックが順序変更に関与する場合は、一般に、その対角線外の成分が変更される。そのブロックの条件が非常に悪い場合を除き、そのブロックの対角成分と固有値は変更されない。条件が非常に悪い場合は、それらが明確に変更される場合がある。2 × 2 のブロックは、2 つの 1 × 1 のブロックに分割できる。つまり、複素数の固有値ペアを、純粋な実数にできる。ただし、固有値の値が、順序の変更によって変更されたりはしない。

?trsyl

実数型準三角マトリックスまたは
複素数型三角マトリックスに関する
シルベスター式を解く。

```
call strsyl ( trana, tranb, isgn, m, n, a, lda, b, ldb, c, ldc, scale,
              info )
call dtrsyl ( trana, tranb, isgn, m, n, a, lda, b, ldb, c, ldc, scale,
              info )
call ctrsyl ( trana, tranb, isgn, m, n, a, lda, b, ldb, c, ldc, scale,
              info )
call ztrsyl ( trana, tranb, isgn, m, n, a, lda, b, ldb, c, ldc, scale,
              info )
```

説明

このルーチンは、シルベスターのマトリックス方程式 $\mathrm{op}(A)X \pm X\mathrm{op}(B) = \alpha C$ を解く。$\mathrm{op}(A) = A$ または A^H であり、マトリックス A と B は上三角マトリックス (実数の場合は標準形の Schur 形式の上準三角マトリックス) である。$\alpha \le 1$ は、X のオーバーフローを防止するためにこのルーチンで求めるスケール係数である。A は $m \times m$、B は $n \times n$、C と X はどちらも $m \times n$ である。マトリックス X は、後方置換によって簡単に求められる。

この方程式は、$\alpha_i \pm \beta_i \ne 0$ の場合にだけ、一意な解を持つ。$\{\alpha_i\}$ と $\{\beta_i\}$ はそれぞれ、A と B の固有値である。符合 (+ または –) は、解くべき方程式の中で使われているものと同じである。

入力パラメータ

trana CHARACTER*1。'N'、'T' 、または 'C' でなければならない。
trana ='N' の場合は、$\mathrm{op}(A) = A$
trana ='T' の場合は、$\mathrm{op}(A) = A^T$ (実数の場合のみ)
trana ='C' の場合は、$\mathrm{op}(A) = A^H$

tranb CHARACTER*1。'N'、'T' 、または 'C' でなければならない。
tranb ='N' の場合は、$\mathrm{op}(B) = B$
tranb ='T' の場合は、$\mathrm{op}(B) = B^T$ (実数の場合のみ)
tranb ='C' の場合は、$\mathrm{op}(B) = B^H$

isgn INTEGER。シルベスター式の形式を指定する。
isgn = +1 の場合は、$\mathrm{op}(A)X + X\mathrm{op}(B) = \alpha C$
isgn = −1 の場合は、$\mathrm{op}(A)X - X\mathrm{op}(B) = \alpha C$

m INTEGER。A の次数、および X と C ($m \geq 0$) の行数。

n INTEGER。B の次数、および X と C ($n \geq 0$) の行数。

a, b, c REAL (strsyl の場合)
DOUBLE PRECISION (dtrsyl の場合)
COMPLEX (ctrsyl の場合)
DOUBLE COMPLEX (ztrsyl の場合)
配列 :
*a(lda, *)* には、マトリックス A を格納する。
a の第 2 次元は、max(1, *m*) 以上でなければならない。
*b(ldb, *)* には、マトリックス B を格納する。
b の第 2 次元は、max(1, *n*) 以上でなければならない。
*c(ldc, *)* には、マトリックス C を格納する。
c の第 2 次元は、max(1, *n*) 以上でなければならない。

lda INTEGER。*a* の第 1 次元。max(1, *m*) 以上であること。

ldb INTEGER。*b* の第 1 次元。max(1, *n*) 以上であること。

ldc INTEGER。*c* の第 1 次元。max(1, *n*) 以上であること。

出力パラメータ

c 解マトリックス X によって上書きされる。

scale REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
スケール係数 α の値。

info INTEGER
info = 0 の場合は、正常に終了する。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。
info = 1 の場合、A と B は共通または近似の固有値を持ち、その摂動値を使ってこの方程式が解かれる。

アプリケーション・ノート

X が正確な解、Y がそれに対応する計算で求めた解、R が誤差マトリックスであり、$R = C - (AY \pm YB)$ とする。その場合の誤差は、常に小さくなる。

$$\|R\|_F = O(\varepsilon)\ (\|A\|_F + \|B\|_F)\ \|Y\|_F$$

ただし、Y は、わずかに摂動した方程式の場合は、正確な解とならない場合もある。つまり、この解は後方安定ではない。

前方誤差に関しては、次の制限が成立する。

$$\|Y - X\|_F \le \|R\|_F / \mathrm{sep}(A, B)$$

ただし、これは、大幅な過大評価である場合もある。sep(A, B) の定義については、[Golub96] を参照。

実数の場合の浮動小数演算のおおよその総数は、$m*n*(m + n)$ である。複素数の場合は、この 4 倍になる。

汎用非対称固有値問題

この節では、汎用非対称固有値問題を解いたり、マトリックス・ペアの Schur 因子分解を順序変更したり、関連する種々の計算タスクを実行するための LAPACK ルーチンについて説明する。

汎用非対称固有値問題は、次のように説明できる。n × n の非対称マトリックス (または非エルミート・マトリックス) のペア A と B があるとき、次の 2 つの式を満足する**汎用固有値** λ と、それに対応する**汎用固有ベクトル** x と y を見つけ出す。

$$Ax = \lambda Bx \quad (\text{右汎用固有ベクトル } x)$$

$$y^H A = \lambda y^H B \ (\text{左汎用固有ベクトル } y)$$

表 5-6 に、汎用非対称固有値問題と汎用シルベスター式を解くための LAPACK ルーチンを示す。

表 5-6　汎用非対称固有値問題を解く計算ルーチン

ルーチン名	機能
?gghrd	直交 / ユニタリ変換を使用して、マトリックス・ペアを汎用上 Hessenberg 形式に縮退する。
?ggbal	一般の実 / 複素マトリックス・ペアを平衡化する。
?ggbak	汎用固有値問題の右または左固有ベクトルを求める。
?hgeqz	QZ 法によりマトリックス・ペア (H, T) の汎用固有値を求める。
?tgevc	上三角マトリックス・ペアから右または左汎用固有ベクトルの一部または全部を求める。
?tgexc	マトリックス・ペア (A, B) において、ある対角ブロックが別の行インデックスに移動するように、(A, B) の汎用 Schur 分解の順序を変更する。
?tgsen	選択した固有値クラスタがマトリックス・ペア (A, B) の主対角ブロックに移動するように、(A, B) の汎用 Schur 分解の順序を変更する。
?tgsyl	汎用シルベスター式を解く。
?tgsna	汎用実 Schur 標準形のマトリックス・ペアに関して、指定した固有値と固有ベクトルの条件数の逆数を見積もる。

?gghrd

直交 / ユニタリ変換を使用して、
マトリックス・ペアを汎用上 Hessenberg
形式に縮退する。

```
call sgghrd ( compq, compz, n, ilo, ihi, a, lda, b, ldb, q, ldq, z,
              ldz, info )
call dgghrd ( compq, compz, n, ilo, ihi, a, lda, b, ldb, q, ldq, z,
              ldz, info )
call cgghrd ( compq, compz, n, ilo, ihi, a, lda, b, ldb, q, ldq, z,
              ldz, info )
call zgghrd ( compq, compz, n, ilo, ihi, a, lda, b, ldb, q, ldq, z,
              ldz, info )
```

説明

このルーチンは、直交 / ユニタリ変換を使用して、実 / 複素マトリックス・ペア (A, B) を、汎用上 Hessenberg 形式に縮退する。ここで、A は一般マトリックス、B は上三角マトリックスである。汎用固有値問題は、$Ax = \lambda Bx$ で表される。一般に、B は、QR 因子分解を計算し、直交マトリックス Q を式の左辺に移動すると、上三角マトリックスになる。このルーチンは、A を次のように Hessenberg マトリックス H に縮退し、

$$Q^H A \ Z = H$$

同時に B を次のように別の上三角マトリックス T に変換すると、

$$Q^H B \ Z = T$$

元の問題を標準形式 $Hy = \lambda Ty$ (ただし、$y = Z^H x$) に縮退する。

直交 / ユニタリ・マトリックス Q と Z は、Givens 回転の積で求められる。この 2 つのマトリックスは、明示的に求められ、次のように入力マトリックス Q_1 と Z_1 に後からも掛けられる。

$$Q_1 \ A \ Z_1^H = (Q_1 Q) \ H \ (Z_1 Z)^H$$

$$Q_1 \ B \ Z_1^H = (Q_1 Q) \ T \ (Z_1 Z)^H$$

Q_1 が、元の式 $Ax = \lambda Bx$ における B の QR 因子分解で得た直交マトリックスならば、?gghrd は元の問題を汎用 Hessenberg 形式に縮退する。

入力パラメータ

compq
CHARACTER*1。'N'、'I'、'V' のどれかでなければならない。
compq ='N' の場合、マトリックス Q は計算されない。
compq ='I' の場合、Q は単位マトリックスに初期化され、直交 / ユニタリ・マトリックス Q が返される。
compq ='V' の場合、Q には、呼び出し時に直交 / ユニタリ・マトリックス Q_1 が入っていなければならない。終了時には積 Q_1Q が返される。

compz
CHARACTER*1。'N'、'I'、'V' のどれかでなければならない。
compz ='N' の場合、マトリックス Z は計算されない。
compz ='I' の場合、Z は単位マトリックスに初期化され、直交 / ユニタリ・マトリックス Z が返される。
compz ='V' の場合、Z には、呼び出し時に直交 / ユニタリ・マトリックス Z_1 が入っていなければならない。終了時には積 Z_1Z が返される。

n
INTEGER。マトリックス A と B の次数 ($n \geq 0$)。

ilo, ihi
INTEGER。*ilo* と *ihi* は、A のどの行と列を縮退するかをマークする。A において、行と列が 1:*ilo*-1 と *ihi*+1:*n* の部分は、すでに上三角形式になっているものとする。一般に、*ilo* と *ihi* の値は、先に呼び出した ?ggbal によって設定されるが、そうでない場合はそれぞれ 1 と *n* に設定する。次の制約がある。
$n > 0$ の場合、$1 \leq ilo \leq ihi \leq n$
$n = 0$ の場合、$ilo = 1$ と $ihi = 0$

a, b, q, z
REAL (sgghrd の場合)
DOUBLE PRECISION (dgghrd の場合)
COMPLEX (cgghrd の場合)
DOUBLE COMPLEX (zgghrd の場合)
配列 :
a(*lda*, *) $n \times n$ の一般マトリックス A を格納する。*a* の第 2 次元のサイズは、max(1, *n*) 以上でなければならない。

`b(ldb, *)` $n \times n$ の上三角マトリックス B を格納する。
`b` の第 2 次元のサイズは、max(1, `n`) 以上でなければならない。

`q(ldq, *)`
`compq='N'` の場合、`q` は参照されない。
`compq='I'` の場合、呼び出し時に `q` を設定する必要はない。
`compq='V'` の場合、`q` に直交 / ユニタリ・マトリックス Q_1 (通常は B の QR 因子分解) が入っていなければならない。
`q` の第 2 次元のサイズは、max(1, `n`) 以上でなければならない。

`z(ldz, *)`
`compq='N'` の場合、`z` は参照されない。
`compq='I'` の場合、呼び出し時に `z` を設定する必要はない。
`compq='V'` の場合、`z` に直交 / ユニタリ・マトリックス Z_1 が入っていなければならない。
`z` の第 2 次元のサイズは、max(1, `n`) 以上でなければならない。

`lda` INTEGER。`a` の第 1 次元のサイズ。max(1, `n`) 以上。

`ldb` INTEGER。`b` の第 1 次元のサイズ。max(1, `n`) 以上。

`ldq` INTEGER。`q` の第 1 次元のサイズ。
`compq='N'` の場合、`ldq` ≥ 1
`compq='I'` または `'V'` の場合、`ldq` ≥ max(1, `n`)

`ldz` INTEGER。`z` の第 1 次元のサイズ。
`compq='N'` の場合、`ldz` ≥ 1
`compq='I'` または `'V'` の場合、`ldz` ≥ max(1, `n`)

出力パラメータ

`a` 終了時に、A の上三角部分と最初の劣対角成分が上 Hessenberg マトリックス H で上書きされ、残りの成分はゼロに設定される。

`b` 終了時に、上三角マトリックス $T = Q^H B\ Z$ で上書きされる。対角成分より下の各成分はゼロに設定される。

`q` `compq='I'` の場合、`q` に、直交 / ユニタリ・マトリックス Q (Q^H は、A と B の左側に適用される Givens 変換の積) が格納される。
`compq='V'` の場合、`q` は積 Q_1Q で上書きされる。

z	*compq* = 'I' の場合、*z* に、直交 / ユニタリ・マトリックス Z (A と B の右側に適用される Givens 変換の積) が格納される。 *compq* = 'V' の場合、*z* は積 Z_1Z で上書きされる。
info	INTEGER *info* = 0 の場合は、正常終了している。 *info* = -*i* の場合は、*i* 番目のパラメータ値が不正である。

?ggbal

一般の実 / 複素マトリックス・ペアを
平衡化する。

```
call sggbal ( job, n, a, lda, b, ldb, ilo, ihi, lscale, rscale, work,
              info )
call dggbal ( job, n, a, lda, b, ldb, ilo, ihi, lscale, rscale, work,
              info )
call cggbal ( job, n, a, lda, b, ldb, ilo, ihi, lscale, rscale, work,
              info )
call zggbal ( job, n, a, lda, b, ldb, ilo, ihi, lscale, rscale, ork,
              info )
```

説明

このルーチンは、一般の実 / 複素マトリックス・ペア (*A*, *B*) を平衡化する。すなわち、まず相似変換によって *A* と *B* を置換し、固有値を対角線上の最初の 1 ～ *ilo*-1 と最後の *ihi*+1 ～ *n* の成分に分離する。次に、*ilo* ～ *ihi* の行と列に対角相似変換を適用して、これらの行と列のノルムができる限り近くなるようにする。この 2 つのステップはオプションである。
平衡化により、マトリックスの 1- ノルムが縮退され、汎用固有値問題 $Ax = \lambda Bx$ における固有値と固有ベクトルの計算精度が向上する。

入力パラメータ

job CHARACTER*1。*A* と *B* に対して実行する演算を指定する。'N'、'P'、'S'、'B' のどれかでなければならない。
job ='N' の場合、演算は行わず、*ilo* = 1、*ihi* = *n*、*lscale*(i) = 1.0、*rscale*(i)=1.0 (ただし、i = 1, ..., *n*) に設定する。
job ='P' の場合、置換のみ。
job ='S' の場合、スケーリングのみ。
job ='B' の場合、置換とスケーリング。

n INTEGER。マトリックス *A* と *B* の次数 ($n \geq 0$)。

a, b
REAL (sggbal の場合)
DOUBLE PRECISION (dggbal の場合)
COMPLEX (cggbal の場合)
DOUBLE COMPLEX (zggbal の場合)
配列 :
*a(lda, *)* マトリックス *A* を格納する。
a の第 2 次元のサイズは、max(1, *n*) 以上でなければならない。
*b(ldb, *)* マトリックス *B* を格納する。
b の第 2 次元のサイズは、max(1, *n*) 以上でなければならない。

lda
INTEGER。*a* の第 1 次元のサイズ。max(1, *n*) 以上。

ldb
INTEGER。*b* の第 1 次元のサイズ。max(1, *n*) 以上。

work
REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
ワークスペース配列 DIMENSION は max(1, 6*n*) 以上。

出力パラメータ

a, b
平衡化したマトリックス*A*と*B*でそれぞれが上書きされる。*job*='N' の場合、*a* と *b* は参照されない。

ilo, ihi
INTEGER。終了時に *ilo* と *ihi* は、*i*>*j* で *j*=1, ..., *ilo*-1 または *i*=*ihi*+1, ..., *n* ならば、*a*(*i, j*)=0 と *b*(*i, j*)=0 が満たされるような整数に設定される。
job='N' または 'S' の場合は、*ilo* = 1、*ihi* = *n* に設定される。

lscale, rscale
REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
配列 DIMENSION は max(1, *n*) 以上。
lscale には、*A* と *B* の左側に適用される置換とスケール係数の詳細が格納される。
行 *j* と交換する行のインデックスを P_j をとし、行 *j* に適用されるスケール係数を D_j とすると、

lscale(*j*) = P_j (*j* = 1, ..., *ilo*-1 の場合)
= D_j (*j* = *ilo*, ..., *ihi* の場合)
= P_j (*j* = *ihi*+1, ..., *n* の場合)

rscale には、*A* と *B* の右側に適用される置換とスケール係数の詳細が格納される。

列 *j* と交換する列のインデックスを P_j をとし、列 *j* に適用されるスケール係数を D_j とすると、

rscale(*j*) = P_j (*j* = 1, ..., *ilo*-1 の場合)
= D_j (*j* = *ilo*, ..., *ihi* の場合)
= P_j (*j* = *ihi*+1, ..., *n* の場合)

交換は、まず *n* から *ihi*+1 の順、次に 1 から *ilo*-1 の順に行われる。

info INTEGER

info = 0 の場合は、正常終了している。

info = -*i* の場合は、*i* 番目のパラメータ値が不正である。

?ggbak

汎用固有値問題において右または左固有ベクトルを求める。

```
call sggbak(job, side, n, ilo, ihi, lscale, rscale, m, v, ldv, info)
call dggbak(job, side, n, ilo, ihi, lscale, rscale, m, v, ldv, info)
call cggbak(job, side, n, ilo, ihi, lscale, rscale, m, v, ldv, info)
call zggbak(job, side, n, ilo, ihi, lscale, rscale, m, v, ldv, info)
```

説明

このルーチンは、平衡化したマトリックス・ペア (?ggbal から返されたもの) で計算した固有ベクトルを逆変換して、実数 / 複素数の汎用固有値問題

$$Ax = \lambda Bx$$

における右または左固有ベクトルを求める。

入力パラメータ

job CHARACTER*1。必要な逆変換のタイプを指定する、'N'、'P'、'S'、'B' のどれかでなければならない。
job ='N' の場合、演算を行わずにリターンする。
job ='P' の場合、置換に対する逆変換のみ行う。
job ='S' の場合、スケーリングに対する逆変換のみ行う。
job ='B' の場合、置換とスケーリングに対する逆変換を行う。
この引数は、?ggbal に指定した引数 *job* と同じ値でなければならない。

side CHARACTER*1。'L' または 'R' でなければならない。
side ='L' の場合、*v* に左固有ベクトルが格納される。
side ='R' の場合、*v* に右固有ベクトルが格納される。

n INTEGER。マトリックス V の行数 ($n \geq 0$)。

ilo, ihi
INTEGER。?gebal で決定された整数 *ilo* と *ihi*。次の制約がある。
$n > 0$ の場合、$1 \le ilo \le ihi \le n$
$n = 0$ の場合、$ilo = 1$ と $ihi = 0$

lscale, rscale
REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
配列 DIMENSION は max(1, *n*) 以上。
配列 *lscale* には、*A* と *B* の左側に適用する置換およびスカラー係数の詳細 (?ggbal から返された値) を格納する。
配列 *rscale* には、*A* と *B* の右側に適用する置換およびスカラー係数の詳細 (?ggbal から返された値) を格納する。

m
INTEGER。マトリックス *V* の列数 ($m \ge 0$)。

v
REAL (sggbak の場合)
DOUBLE PRECISION (dggbak の場合)
COMPLEX (cggbak の場合)
DOUBLE COMPLEX (zggbak の場合)
配列 *v(ldv, *)*。変換すべき右または左固有ベクトルが入ったマトリックス (?tgevc から返された値) を格納する。
v の第 2 次元のサイズは、max(1, *m*) 以上でなければならない。

ldv
INTEGER。*v* の第 1 次元のサイズ。max(1, *n*) 以上。

出力パラメータ

v
変換後の固有ベクトルで上書きされる。

info
INTEGER
info = 0 の場合は、正常終了している。
info = -*i* の場合は、*i* 番目のパラメータ値が不正である。

?hgeqz

QZ 法を使用して、マトリックス・ペア
(*H*, *T*) の汎用固有値を見つける。

```
call shgeqz(job, compq, compz, n, ilo, ihi, h, ldh, t, ldt, alphar,
            alphai, beta, q, ldq, z, ldz, work, lwork, info )
call dhgeqz(job, compq, compz, n, ilo, ihi, h, ldh, t, ldt, alphar,
            alphai, beta, q, ldq, z, ldz, work, lwork, info )
call chgeqz(job, compq, compz, n, ilo, ihi, h, ldh, t, ldt, alpha,
            beta, q, ldq, z, ldz, work, lwork, rwork, info )
call zhgeqz(job, compq, compz, n, ilo, ihi, h, ldh, t, ldt, alpha,
            beta, q, ldq, z, ldz, work, lwork, rwork, info )
```

説明

このルーチンは、ダブルシフト版 (実数の場合) またはシングルシフト版 (複素数の場合) の *QZ* 法を使用して、実 / 複素マトリックス・ペア (*H*, *T*) の固有値を計算する。ただし、*H* は上 Hessenberg マトリックス、*T* は上三角マトリックスである。
このタイプのマトリックス・ペアは、`?gghrd` により、実 / 複素マトリックス・ペア (*A*, *B*) を汎用上 Hessenberg 形式に縮退すると得られる。

$$A = Q_1 \ H \ Z_1^H, \qquad B = Q_1 \ T \ Z_1^H$$

実数の場合 :
`job='S'` の場合、Hessenberg- 三角マトリックス・ペア (*H*, *T*) も、次のような汎用 Schur 形式に縮退される。

$$H = Q \ S \ Z^T, \qquad T = Q \ P \ Z^T$$

Q と *Z* は直交マトリックス、*P* は上三角マトリックス、*S* は、1 × 1 と 2 × 2 の対角ブロックを持つ準三角マトリックスである。
1 × 1 のブロックは、マトリックス・ペア (*H*, *T*) の実固有値に対応し、2 × 2 のブロックは固有値の複素共役ペアに対応する。
また、*S* の 2 × 2 のブロックに対応する *P* の 2 × 2 の上三角対角ブロックは、正値対角形式に縮退される。すなわち、*S*(j+1, j) がゼロでなければ、*P*(j+1, j) = *P*(j, j+1) = 0、*P*(j, j) > 0、*P*(j+1, j+1) > 0 となる。

複素数の場合 :
`job='S'` の場合、Hessenberg- 三角マトリックス・ペア (H, T) も、次のような汎用 Schur 形式に縮退される。

$$H = Q\ S\ Z^H, \qquad T = Q\ P\ Z^H$$

Q と Z はユニタリ・マトリックス、S と P は上三角マトリックスである。

すべての場合 :
オプションで、汎用 Schur 因子分解から得た直交 / ユニタリ・マトリックス Q を入力マトリックス Q_1 の後に掛け、直交 / ユニタリ・マトリックス Z を入力マトリックス Z_1 の後に掛けられる。
Q_1 と Z_1 が、`?gghrd` によりマトリックス・ペア (A, B) を汎用上 Hessenberg 形式に縮退した直交 / ユニタリ・マトリックスならば、出力マトリックス Q_1Q と Z_1Z は、(A, B) の汎用 Schur 因子分解で得た直交 / ユニタリ因子である。

$$A = (Q_1Q)\ S\ (Z_1Z)^H, \qquad B = (Q_1Q)\ P\ (Z_1Z)^H$$

オーバーフローを避けるために、マトリックス・ペア (H, T) の固有値 (すなわち、(A, B) の固有値) は、値のペア (*alpha, beta*) として計算される。`chgeqz`/`zhgeqz` の場合、*alpha* と *beta* は複素数である。
`shgeqz`/`dhgeqz` の場合、*alpha* は複素数、*beta* は実数である。*beta* がゼロでなければ、$\lambda = alpha / beta$ は、次の汎用非対称固有値問題 (GNEP) の固有値である。

$$Ax = \lambda Bx$$

また、*alpha* がゼロでなければ、$\mu = beta / alpha$ は、次のように別の形式で表された GNEP の固有値である。

$$\mu Ay = By$$

実固有値 (実数の場合)、または i 番目の固有値を表す *alpha* と *beta*(複素数の場合) は、次のように汎用 Schur 形式から直接読み込める。

$$alpha = S(\mathrm{i}, \mathrm{i}), \quad beta = P(\mathrm{i}, \mathrm{i})$$

入力パラメータ

job CHARACTER*1。実行する演算を指定する。'E' または 'S' でなければならない。
job ='E' の場合、固有値のみ計算する。
job ='S' の場合、固有値と Schur 形式を計算する。

compq CHARACTER*1。'N'、'I'、'V' のどれかでなければならない。
compq ='N' の場合、左 Schur ベクトル (*q*) は計算されない。
compq ='I' の場合、*q* は単位マトリックスに初期化され、(*H*, *T*) の左 Schur ベクトルからなるマトリックスが返される。
compq ='V' の場合、*q* は、呼び出し時に直交 / ユニタリ・マトリックス Q_1 が入っていなければならない。終了時には積 Q_1Q が返される。

compz CHARACTER*1。'N'、'I'、'V' のどれかでなければならない。
compz ='N' の場合、左 Schur ベクトル (*q*) は計算されない。
compz ='I' の場合、*z* は単位マトリックスに初期化され、(*H*, *T*) の右 Schur ベクトルからなるマトリックスが返される。
compz ='V' の場合、*z* は、呼び出し時に直交 / ユニタリ・マトリックス Z_1 が入っていなければならない。終了時には積 Z_1Z が返される。

n INTEGER。マトリックス *H*、*T*、*Q*、*Z* の次数 ($n \geq 0$)。

ilo, ihi INTEGER。*ilo* と *ihi* は、*H* のどの行と列が Hessenberg 形式かをマークする。*H* において、行と列が 1:*ilo*-1 と *ihi*+1:*n* の範囲は、すでに上三角形式になっているものとする。次の制約がある。
$n > 0$ の場合、$1 \leq ilo \leq ihi \leq n$
$n = 0$ の場合、$ilo = 1$ と $ihi = 0$

h, t, q, z, work REAL (shgeqz の場合)
DOUBLE PRECISION (dhgeqz の場合)
COMPLEX (chgeqz の場合)
DOUBLE COMPLEX (zhgeqz の場合)
配列 :
呼び出し時は、*h(ldh, *)* に $n \times n$ の上 Hessenberg マトリックス *H* を格納する。
h の第 2 次元のサイズは、max(1, *n*) 以上でなければならない。

呼び出し時は、*t*(*ldt*, *) に *n* × *n* の上三角マトリックス *T* を格納する。
t の第 2 次元のサイズは、max(1, *n*) 以上でなければならない。
q (*ldq*, *):
呼び出し時は、*compq* = 'V' ならば、(*A*, *B*) を汎用 Hessenberg 形式に縮退するのに使用した直交 / ユニタリ・マトリックス Q_1 を格納する。
compq = 'N' の場合、*q* は参照されない。
q の第 2 次元のサイズは、max(1, *n*) 以上でなければならない。
z (*ldz*, *):
呼び出し時は、*compz* = 'V' ならば、(*A*, *B*) を汎用 Hessenberg 形式に縮退するのに使用した直交 / ユニタリ・マトリックス Z_1 を格納する。
compz = 'N' の場合、*z* は参照されない。
z の第 2 次元のサイズは、max(1, *n*) 以上でなければならない。
work(*lwork*) はワークスペース配列。

ldh INTEGER。*h* の第 1 次元のサイズ。max(1, *n*) 以上。

ldt INTEGER。*t* の第 1 次元のサイズ。max(1, *n*) 以上。

ldq INTEGER。*q* の第 1 次元のサイズ。
compq = 'N' の場合、*ldq* ≥ 1
compq = 'I' または 'V' の場合、*ldq* ≥ max(1, *n*)

ldz INTEGER。*z* の第 1 次元のサイズ。
compq = 'N' の場合、*ldz* ≥ 1
compq = 'I' または 'V' の場合、*ldz* ≥ max(1, *n*)

lwork INTEGER。配列 *work* のサイズ。*lwork* ≥ max(1, *n*)

rwork REAL (chgeqz の場合)
DOUBLE PRECISION (zhgeqz の場合)
ワークスペース配列 DIMENSION は max(1, *n*) 以上。
複素数の場合にのみ使用する。

出力パラメータ

h

実数の場合 :

job = 'S' の場合、終了時に、汎用 Schur 因子分解で得た上準三角マトリックス *S* が *h* に格納される。
2 × 2 の対角ブロック (固有値の複素共役ペアに対応) が標準形で返され、*h*(i, i) = *h*(i+1, i+1) および *h*(i+1, i) * *h*(i, i+1) < 0 となる。
job = 'E' の場合、終了時に、*h* の対角ブロックは *S* と同じになるが、その他の部分は設定されない。

複素数の場合 :

job = 'S' の場合、終了時に、汎用 Schur 因子分解で得た上三角マトリックス *S* が *h* に格納される。
job = 'E' の場合、終了時に、*h* の対角成分は *S* と同じになるが、その他の部分は設定されない。

t

job = 'S' の場合、終了時に、汎用 Schur 因子分解で得た上三角マトリックス *P* が *t* に格納される。

実数の場合 :

S の 2 × 2 のブロックに対応する *P* の 2 × 2 の対角ブロックが、正値対角形式に縮退される。すなわち、*h*(j+1, j) がゼロでない場合、*t*(j+1, j)=*t*(j, j+1)=0、*t*(j, j) と *t*(j+1, j+1) が正になる。
job = 'E' の場合、終了時に、*t* の対角ブロックは *P* と同じになるが、その他の部分は設定されない。

複素数の場合 :

job = 'E' の場合、終了時に、*t* の対角成分は *P* と同じになるが、その他の部分は設定されない。

alphar, alphai

REAL (shgeqz の場合)
DOUBLE PRECISION (dhgeqz の場合)
配列 DIMENSION は max(1, *n*) 以上。
汎用非対称固有値問題において固有値を定義する各スカラー *alpha* の実部と虚部。

alphai(j) がゼロの場合、j 番目の固有値は実数である。正の場合、j 番目と (j+1) 番目の固有値は、alphai(j+1) = -alphai(j) であるような複素共役ペアである。

alpha
COMPLEX (chgeqz の場合)
DOUBLE COMPLEX (zhgeqz の場合)
配列 DIMENSION は max(1, n) 以上。
汎用非対称固有値問題で固有値を定義する複素スカラー *alpha*。汎用 Schur 因子分解で、alphai(i) = *S*(i, i)

beta
REAL (shgeqz の場合)
DOUBLE PRECISION (dhgeqz の場合)
COMPLEX (chgeqz の場合)
DOUBLE COMPLEX (zhgeqz の場合)
配列 DIMENSION は max(1, n) 以上。
実数の場合 :
汎用非対称固有値問題で固有値を定義するスカラー *beta*。
alpha = (alphar(j), alphai(j)) と *beta* = beta(j) とで、マトリックス・ペア (*A, B*) の j 番目の固有値が、λ = *alpha*/*beta*、μ = *beta*/*alpha* のどちらかの形式で表される。λ または μ がオーバーフローする可能性があるので、一般にその計算は行わない。
複素数の場合 :
汎用非対称固有値問題で固有値を定義する、非負の実スカラー *beta*。汎用 Schur 因子分解で、
beta(i) = *P*(i, i)
alpha = alpha(j) と *beta* = beta(j) とで、マトリックス・ペア (*A, B*) の j 番目の固有値が、λ = *alpha*/*beta*、μ = *beta*/*alpha* のどちらかの形式で表される。λ または μ がオーバーフローする可能性があるので、一般にその計算は行わない。

q
終了時に、compq='I' の場合、q は、マトリックス・ペア (*H, T*) の左 Schur ベクトルからなる直交 / ユニタリ・マトリックスで上書きされる。compq='V' の場合は、マトリックス・ペア (*A, B*) の左 Schur ベクトルからなる直交 / ユニタリ・マトリックスで上書きされる。

z	終了時に、*compz* = 'I' の場合、*z* は、マトリックス・ペア (*H, T*) の右 Schur ベクトルからなる直交 / ユニタリ・マトリックスで上書きされる。*compq* = 'V' の場合は、マトリックス・ペア (*A, B*) の右 Schur ベクトルからなる直交 / ユニタリ・マトリックスで上書きされる。
work(1)	*info* ≥ 0 の場合は、最適パフォーマンスに必要な *lwork* の最小値が、終了時に *work*(1) に格納される。以降の実行では、この *lwork* 値を使用する。
info	INTEGER *info* = 0 の場合は、正常終了している。 *info* = -*i* の場合は、*i* 番目のパラメータ値が不正である。 *info* = 1, ..., *n* の場合は、*QZ* 反復で収束しなかった。(*H, T*) は Schur 形式ではないが、*alphar*(i)、*alphai*(i) (実数の場合)、*alpha*(i) (複素数の場合)、*beta*(i) は正しい値である (ただし、i = *info*+1, ..., *n*)。 *info* = *n*+1, ..., 2*n* の場合は、シフト計算に失敗した。(*H, T*) は Schur 形式ではないが、*alphar*(i)、*alphai*(i) (実数の場合)、*alpha*(i) (複素数の場合)、*beta*(i) は正しい値である (ただし、i = *info*-*n*+1, ..., *n*)。

?tgevc

上三角マトリックス・ペアの右と左汎用
固有ベクトルの一部または全部を求める。

```
call stgevc ( side, howmny, select, n, s, lds, p, ldp, vl, ldvl, vr,
              ldvr, mm, m, work, info )
call stgevc ( side, howmny, select, n, s, lds, p, ldp, vl, ldvl, vr,
              ldvr, mm, m, work, info )
call stgevc ( side, howmny, select, n, s, lds, p, ldp, vl, ldvl, vr,
              ldvr, mm, m, work, rwork, info )
call stgevc ( side, howmny, select, n, s, lds, p, ldp, vl, ldvl, vr,
              ldvr, mm, m, work, rwork, info )
```

説明

このルーチンは、実 / 複素マトリックス・ペア (S, P) の右と左固有ベクトルの一部または全部を求める。*S* は準三角マトリックス (実数の場合) または上三角マトリックス (複素数の場合)、*P* は上三角マトリックスである。

このタイプのマトリックス・ペアは、`?gghrd` と `?hgeqz` による実 / 複素マトリックス・ペア (*A*, *B*) の汎用 Schur 因子分解で得られる。

$$A = Q\,S\,Z^H \quad , \qquad B = Q\,P\,Z^H$$

固有値 *w* に対応する、(*S*, *P*) の右固有ベクトル *x* と左固有ベクトル *y* は次のように定義される。

$$S\,x = w\,P\,x \quad , \qquad y^H S = w\,y^H P$$

固有値は、このルーチンに入力されず、*S* と *P* の対角ブロックまたは対角成分から直接計算される。

このルーチンは、(*S*, *P*) の右と左の固有ベクトルからなるマトリックス *X* と *Y*、または積 *ZX* と *QY* (*Z* と *Q* は入力マトリックス) を返す。*Q* と *Z* がマトリックス・ペア (*A*, *B*) の汎用 Schur 因子分解で得た直交 / ユニタリ因子ならば、*ZX* と *QY* は (*A*, *B*) の右と左の固有ベクトルからなるマトリックスである。

入力パラメータ

side CHARACTER*1。'R'、'L'、'B' のどれかでなければならない。
side ='R' の場合、右固有ベクトルのみ計算する。
side ='L' の場合、左固有ベクトルのみ計算する。
side ='B' の場合、右と左の固有ベクトルを両方計算する。

howmny CHARACTER*1。'A'、'B'、'S' のどれかでなければならない。
howmny ='A' の場合、右と左の固有ベクトルをすべて計算する。
howmny ='B' の場合、*vr* と *vl* 内のマトリックスを逆変換して、右と左の固有ベクトルをすべて計算する。
howmny ='S' の場合、論理配列 *select* で指定した右と左の固有ベクトルのみ計算する。

select LOGICAL。
配列 DIMENSION は max(1, *n*) 以上。
howmny ='S' の場合、*select* は、計算すべき固有値を指定する。
howmny ='A' または 'B' の場合、*select* は参照されない。
実数の場合 :
ω_j が実固有値の場合、*select*(*j*) が .TRUE. ならば、それに対応する実固有ベクトルが計算される。
ω_j と ω_{j+1} が複素固有値の実部と虚部の場合、*select*(j) または *select*(*j*+1) が .TRUE. ならば、それに対応する複素固有ベクトルが計算され、終了時に *select*(*j*) が .TRUE. に設定され、*select*(*j*+1) が .FALSE. に設定される。
複素数の場合 :
select(*j*) が .TRUE. ならば、*j* 番目の固有値に対応する固有ベクトルが計算される。

n INTEGER。マトリックス *A* と *B* の次数 (*n* ≥ 0)。

s, p, vl, vr, work REAL (stgevc の場合)
DOUBLE PRECISION (dtgevc の場合)
COMPLEX (ctgevc の場合)
DOUBLE COMPLEX (ztgevc の場合)
配列 :

`s(lds, *)` には、`?hgeqz` による汎用 Schur 因子分解で得たマトリックス *S* を格納する。このマトリックスは、上準三角マトリックス (実数の場合)、または三角マトリックス (複素数の場合) である。
`s` の第 2 次元のサイズは、max(1, `n`) 以上でなければならない。

`p(ldp, *)` には、`?hgeqz` による汎用 Schur 因子分解で得たマトリックス *P* を格納する。
実数の場合は、S の 2 × 2 のブロックに対応する *P* の 2 × 2 の対角ブロックが、正値対角形式でなければならない。
複素数の場合、*P* は実対角成分を持たなければならない。
`p` の第 2 次元のサイズは、max(1, `n`) 以上でなければならない。

`side='L'` または `'B'` で `howmny='B'` の場合、`vl(ldvl, *)` に `n` × `n` のマトリックス *Q* (通常は、`?hgeqz` から返された、左 Schur ベクトルからなる直交 / ユニタリ・マトリックス *Q*) が入っていなければならない。`vl` の第 2 次元のサイズは、max(1, `mm`) 以上でなければならない。`side='R'` の場合、`vl` は参照されない。

`side='R'` または `'B'` で `howmny='B'` の場合、`vr(ldvr, *)` に `n` × `n` のマトリックス *Z* (通常は、`?hgeqz` から返された、右 Schur ベクトルからなる直交 / ユニタリ・マトリックス *Z*) が入っていなければならない。`vr` の第 2 次元のサイズは、max(1, `mm`) 以上でなければならない。`side='L'` の場合、`vr` は参照されない。

`work(*)` は、ワークスペース配列。
`DIMENSION` は、実数の場合は max(1, 6*`n`) 以上、複素数の場合は max(1, 2*`n`) 以上。

`lda` `INTEGER`。`a` の第 1 次元のサイズ。max(1, `n`) 以上。

`ldb` `INTEGER`。`b` の第 1 次元のサイズ。max(1, `n`) 以上。

`ldvl` `INTEGER`。`vl` の第 1 次元のサイズ。
`side='L'` または `'B'` の場合、`ldvl` ≥ max(1, `n`)
`side='R'` の場合、`ldvl` ≥ 1

ldvr INTEGER。*vr* の第 1 次元のサイズ。
side ='R' または 'B' の場合、*ldvr* ≥ max(1, *n*)
side ='L' の場合、*ldvr* ≥ 1

mm INTEGER。配列 *vl* と *vr* の列数 (*mm* ≥ *m*)。

rwork REAL (ctgevc の場合)
DOUBLE PRECISION (ztgevc の場合)
ワークスペース配列 DIMENSION は max(1, 2*n) 以上。
複素数の場合にのみ使用する。

出力パラメータ

vl *side* ='L' または 'B' の場合、終了時に *vl* に次の値が格納される。
howmny ='A' の場合、(*S*, *P*) の左固有ベクトルからなるマトリックス *Y*。
howmny ='B' の場合、マトリックス *QY*。
howmny ='S' の場合、*select* で指定した (*S*, *P*) の左固有ベクトルが、固有値と同じ順に、*vl* の連続した列に格納される。
実数の場合 :
複素固有値に対応する複素固有ベクトルが、連続した 2 つの列に格納される (最初の列に実部、2 番目の列に虚部)。

vr *side* ='R' または 'B' の場合、*vr* に終了時に次の値が格納される。
howmny ='A' の場合、(*S*, *P*) の右固有ベクトルからなるマトリックス *X*。
howmny ='B' の場合、マトリックス *ZX*。
howmny ='S' の場合、*select* で指定した (*S*, *P*) の右固有ベクトルが、固有値と同じ順に、*vr* の連続した列に格納される。
実数の場合 :
複素固有値に対応する複素固有ベクトルが、連続した 2 つの列に格納される (最初の列に実部、2 番目の列に虚部)。

m

INTEGER。実際に固有値の格納に使用された配列 *vl* と *vr* の列数。

howmny = 'A' または 'B' の場合、*m* は *n* に設定される。

実数の場合:

選択された実固有ベクトル 1 つに対して 1 列、選択された複素固有ベクトル 1 つに対して 2 列が必要である。

複素数の場合:

選択された固有ベクトル 1 つに対して 1 列必要である。

info

INTEGER

info = 0 の場合は、正常終了している。

info = -*i* の場合は、*i* 番目のパラメータ値が不正である。

実数の場合:

info = *i* > 0 の場合、2 × 2 のブロック (*i*:*i*+1) に複素固有値が入っていない。

?tgexc

マトリックス・ペア (A, B) のある対角ブロックが別の行インデックスに移動するように、(A, B) の汎用 Schur 分解の順序を変更する。

```
call stgexc ( wantq, wantz, n, a, lda, b, ldb, q, ldq, z, ldz, fst,
              ilst, work, lwork, info )
call dtgexc ( wantq, wantz, n, a, lda, b, ldb, q, ldq, z, ldz, ifst,
              ilst, work, lwork, info )
call ctgexc ( wantq, wantz, n, a, lda, b, ldb, q, ldq, z, ldz, ifst,
              ilst, info )
call ztgexc ( wantq, wantz, n, a, lda, b, ldb, q, ldq, z, ldz, ifst,
              ilst, info )
```

説明

このルーチンは、直交 / ユニタリ等価変換を使用して、実 / 複素マトリックス・ペア (A, B) の汎用実 Schur/Schur 分解の順序を変更する。

$$(A, B) = Q\ (A, B)\ Z^H$$

このとき、(A, B) において、行インデックス *ifst* の対角ブロックが行 *ilst* に移動するように変更する。
マトリックス・ペア (A, B) は、汎用実 Schur/Schur 標準形 (?gges から返される形式)、すなわち、A は 1×1 と 2×2 の対角ブロックを持つブロック上三角マトリックス、B は上三角マトリックスでなければならない。
オプションで、汎用 Schur ベクトルからなるマトリックス Q と Z を更新できる。

$$Q(\text{in}) * A(\text{in}) * Z(\text{in})' = Q(\text{out}) * A(\text{out}) * Z(\text{out})'$$

$$Q(\text{in}) * B(\text{in}) * Z(\text{in})' = Q(\text{out}) * B(\text{out}) * Z(\text{out})'$$

入力パラメータ

wantq, wantz　LOGICAL。
wantq = .TRUE. の場合、左側の変換マトリックス Q を更新する。

	wantq =.FALSE. の場合、*Q* を更新しない。 *wantz* =.TRUE. の場合、右側の変換マトリックス *Z* を更新する。 *wantz* =.FALSE. の場合、*Z* を更新しない。
n	INTEGER。マトリックス *A* と *B* の次数 (*n* ≥ 0)。
a, b, q, z	REAL (stgexc の場合) DOUBLE PRECISION (dtgexc の場合) COMPLEX (ctgexc の場合) DOUBLE COMPLEX (ztgexc の場合) 配列 : *a(lda, *)* マトリックス *A* を格納する。 *a* の第 2 次元のサイズは、max(1, *n*) 以上でなければならない。 *b(ldb, *)* マトリックス *B* を格納する。 *b* の第 2 次元のサイズは、max(1, *n*) 以上でなければならない。 *q(ldq, *)* *wantq* =.FALSE. の場合、*q* は参照されない。 *wantq* =.TRUE. の場合、*q* に直交 / ユニタリ・マトリックス *Q* が入っていなければならない。 *q* の第 2 次元のサイズは、max(1, *n*) 以上でなければならない。 *z(ldz, *)* *wantz* =.FALSE. の場合、*z* は参照されない。 *wantz* =.TRUE. の場合、*z* に直交 / ユニタリ・マトリックス *Z* が入っていなければならない。 *z* の第 2 次元のサイズは、max(1, *n*) 以上でなければならない。
lda	INTEGER。*a* の第 1 次元のサイズ。max(1, *n*) 以上。
ldb	INTEGER。*b* の第 1 次元のサイズ。max(1, *n*) 以上。
ldq	INTEGER。*q* の第 1 次元のサイズ。 *wantq* =.FALSE. の場合、*ldq* ≥ 1 *wantq* =.TRUE. の場合、*ldq* ≥ max(1, n)
ldz	INTEGER。*z* の第 1 次元のサイズ。 *wantz* =.FALSE. の場合、*ldz* ≥ 1 *wantz* =.TRUE. の場合、*ldz* ≥ max(1, *n*)

ifst, ilst INTEGER。(*A*, *B*) の対角ブロックの順序変更を指定する。行インデックス *ifst* のブロックが、隣接ブロックの連続的交換により、行 *ilst* に移動される。制約は、$1 \le ifst, ilst \le n$

work REAL (stgexc の場合)
DOUBLE PRECISION (dtgexc の場合)
ワークスペース配列 DIMENSION は *lwork*。実数の場合にのみ使用する。

lwork INTEGER。*work* のサイズ、$4n+16$ 以上でなければならない

出力パラメータ

a, b 更新されたマトリックス *A* と *B* で上書きされる。

ifst, ilst 実数の場合にのみ上書きされる。
ifst が呼び出し時に 2×2 のブロックの第 2 行を指していた場合、第 1 行を指すように変更される。*ilst* は、常にブロックの最新位置の第 1 行 (入力値と ± 1 異なる可能性がある) を指す。

info INTEGER
info = 0 の場合は、正常終了している。
info = *-i* の場合は、*i* 番目のパラメータ値が不正である。
info = 1 の場合、変換後のマトリックス・ペア (*A*, *B*) が汎用 Schur 形式から遠すぎる。すなわち、この問題は悪条件である。(*A*, *B*) は部分的に順序変更されている可能性があり、*ilst* は移動中のブロックの現在位置の第 1 行を指している。

?tgsen

選択した固有値クラスタがマトリックス・ペア (*A*, *B*) の主対角ブロックに現れるように、(*A*, *B*) の汎用 Schur 分解の順序を変更する。

```
call stgsen ( ijob, wantq, wantz, select, n, a, lda, b, ldb, alphar,
              alphai, beta, q, ldq, z, ldz, m, pl, pr, dif, work,
              lwork, iwork, liwork, info )
call dtgsen ( ijob, wantq, wantz, select, n, a, lda, b, ldb, alphar,
              alphai, beta, q, ldq, z, ldz, m, pl, pr, dif, work,
              lwork, iwork, liwork, info )
call ctgsen ( ijob, wantq, wantz, select, n, a, lda, b, ldb, alpha,
              beta, q, ldq, z, ldz, m, pl, pr, dif, work, lwork,
              iwork, liwork, info )
call ztgsen ( ijob, wantq, wantz, select, n, a, lda, b, ldb, alpha,
              beta, q, ldq, z, ldz, m, pl, pr, dif, work, lwork,
              iwork, liwork, info )
```

説明

このルーチンは、実 / 複素マトリックス・ペア (*A*, *B*) の汎用実 Schur/Schur 分解の順序を変更し (直交 / ユニタリ等価変換 *Q*' * (*A*, *B*) * *Z* を使用)、選択した固有値クラスタが、マトリックス・ペア (*A*, *B*) の主対角ブロックに現れるようにする。
Q と *Z* の先頭列は、対応する左と右の固有空間 (収縮部分空間) の正規直交 / ユニタリ基底を形成する。
(*A*, *B*) は、汎用実 Schur/Schur 標準形 (?gges から返される形式)、すなわち *A* と *B* が共に上三角マトリックスでなければならない。
?tgsen は、順序変更後のマトリックス・ペア (*A*, *B*) の汎用固有値も計算する。

ω_j = (alphar(j) + alphai(j)*i)/beta(j)　　(実数の場合)
ω_j = alpha(j)/beta(j)　　(複素数の場合)

このルーチンは、オプションで、固有値と固有空間の条件数の逆数を見積もる。すなわち、Difu[(A_{11}, B_{11}), (A_{22}, B_{22})] と Difl[(A_{11}, B_{11}), (A_{22}, B_{22})] を計算する。これは、マトリックス・ペア (A_{11}, B_{11}) と (A_{22}, B_{22}) (選択したクラスタに対応するものとそれ以外の固有値に対応するもの) の差と、(1,1) ブロック内の選択したクラスタに対する左と右固有空間への「投影」のノルムの差である。

入力パラメータ

ijob

INTEGER。固有値クラスタに関する条件数 (*pl* と *pr*)、または収縮部分空間 Difu と Difl を求めるかどうか指定する。

ijob = 0 の場合、*select* に対する順序変更のみ。

ijob = 1 の場合、選択したクラスタに関する左と右の固有空間への「投影」のノルムの逆数 (*pl* と *pr*)。

ijob = 2 の場合、F- ノルム・ベースの推定値を使用して、Difu と Difl の上限値を計算する (*dif* (1:2))。

ijob = 3 の場合、1- ノルム・ベースの推定値を使用して、Difu と Difl の推定値を計算する (*dif* (1:2))。このオプションは、*ijob*=2 の 5 倍の負荷がかかる。

ijob = 4 の場合、*pl*、*pr*、*dif* を計算する (上記のオプション 0、1、2 の組み合わせ)。すべてを計算する効率のよいオプション。

ijob = 5 の場合、*pl*、*pr*、*dif* を計算する (上記のオプション 0、1、3 の組み合わせ)。

wantq, wantz

LOGICAL。

wantq = .TRUE. の場合、左側の変換マトリックス Q を更新する。

wantq = .FALSE. の場合、Q を更新しない。

wantz = .TRUE. の場合、右側の変換マトリックス Z を更新する。

wantz = .FALSE. の場合、Z を更新しない。

select

LOGICAL。

配列 DIMENSION は max(1, *n*) 以上。

選択するクラスタに属する固有値を指定する。

固有値 ω_j を選択するには、*select*(*j*) を .TRUE. に設定する。

実数の場合 : 複素共役固有値のペア ω_j と ω_{j+1} (2×2 の対角ブロックに対応) を選択するには、*select*(*j*) と *select*(*j*+1) のどちらかを .TRUE. に設定する。

複素共役固有値 ω_j と ω_{j+1} は、共にクラスタに含めるか、共にクラスタから除外する必要がある。

n INTEGER。マトリックス *A* と *B* の次数 ($n \geq 0$)。

a, b, q, z, work REAL (stgsen の場合)
DOUBLE PRECISION (dtgsen の場合)
COMPLEX (ctgsen の場合)
DOUBLE COMPLEX (ztgsen の場合)

配列 :

*a(lda, *)* マトリックス *A* を格納する。
実数の場合 : *A* は上準三角マトリックスで、(*A*, *B*) は汎用実 Schur 標準形。
複素数の場合 : *A* は上三角マトリックスで、汎用 Schur 標準形。
a の第 2 次元のサイズは、max(1, *n*) 以上でなければならない。

*b(ldb, *)* マトリックス *B* を格納する。
実数の場合 : *B* は上三角マトリックスで、(*A*, *B*) は汎用実 Schur 標準形。
複素数の場合 : *B* は上三角マトリックスで、汎用 Schur 標準形。
b の第 2 次元のサイズは、max(1, *n*) 以上でなければならない。

*q(ldq, *)*
wantq =.TRUE. の場合、*q* は $n \times n$ のマトリックス。
wantq =.FALSE. の場合、*q* は参照されない。
q の第 2 次元のサイズは、max(1, *n*) 以上でなければならない。

*z(ldz, *)*
wantz =.TRUE. の場合、*z* は $n \times n$ のマトリックス。
wantz =.FALSE. の場合、*z* は参照されない。
z の第 2 次元のサイズは、max(1, *n*) 以上でなければならない。

work(lwork) はワークスペース配列。*ijob* = 0 の場合、*work* は参照されない。

lda INTEGER。*a* の第 1 次元のサイズ。max(1, *n*) 以上。

ldb INTEGER。*b* の第 1 次元のサイズ。max(1, *n*) 以上。

ldq INTEGER。*q* の第 1 次元のサイズ。*ldq* ≥ 1
wantq = .TRUE. の場合、*ldq* ≥ max(1, *n*)

ldz INTEGER。*z* の第 1 次元のサイズ。*ldz* ≥ 1
wantz = .TRUE. の場合、*ldz* ≥ max(1, *n*)

lwork INTEGER。配列 *work* のサイズ。
実数の場合 :
ijob = 1、2、4 の場合、*lwork* ≥ max(4*n*+16, 2*m*(*n*-*m*))
ijob = 3 または 5 の場合、*lwork* ≥ max(4*n*+16, 4*m*(*n*-*m*))
複素数の場合 :
ijob = 1、2、4 の場合、*lwork* ≥ max(1, 2*m*(*n*-*m*))
ijob = 3 または 5 の場合、*lwork* ≥ max(1, 4*m*(*n*-*m*))

iwork INTEGER。ワークスペース配列 DIMENSION は *liwork*
ijob = 0 の場合、*iwork* は参照されない。

liwork INTEGER。配列 *iwork* のサイズ。
実数の場合 :
ijob = 1、2、4 の場合、*liwork* ≥ *n*+6
ijob = 3 または 5 の場合、*liwork* ≥ max(*n*+6, 2*m*(*n*-*m*))
複素数の場合 :
ijob = 1、2、4 の場合、*liwork* ≥ *n*+2
ijob = 3 または 5 の場合、*liwork* ≥ max(*n*+2, 2*m*(*n*-*m*))

出力パラメータ

a, b 順序変更されたマトリックス*A*と*B*でそれぞれ上書きされる。

alphar, alphai REAL (stgsen の場合)
DOUBLE PRECISION (dtgsen の場合)
配列 DIMENSION は max(1, *n*) 以上。汎用固有値を形成する値が格納される (実数の場合)。
beta を参照のこと。

alpha COMPLEX (ctgsen の場合)
DOUBLE COMPLEX (ztgsen の場合)
配列 DIMENSION は max(1, *n*) 以上。汎用固有値を形成する値が格納される (複素数の場合)。
beta を参照のこと。

beta

REAL (stgsen の場合)
DOUBLE PRECISION (dtgsen の場合)
COMPLEX (ctgsen の場合)
DOUBLE COMPLEX (ztgsen の場合)
配列 DIMENSION は max(1, *n*) 以上。

実数の場合 :

終了時に、(*alphar*(j) + *alphai*(j)*i)/*beta*(j) (ただし、j=1, ..., *n*) が、汎用固有値になる。

alphar(j) + *alphai*(j)*i と *beta*(j) (ただし、j=1, ..., *n*) は、複素ユニタリ変換を使用して、(*A*, *B*) の汎用実 Schur 形式の 2 × 2 の対角ブロックをさらに縮退して三角形式にした場合の、複素 Schur 形式 (*S*, *T*) の対角成分である。

alphai(j) がゼロの場合、j 番目の固有値は実数である。正の場合は、j 番目と (j+1) 番目の固有値は複素共役ペアで、*alphai*(j+1) は負である。

複素数の場合 :

(*A*, *B*) を汎用 Schur 形式に縮退させたときの *A* と *B* の対角成分。*alpha*(i)/*beta*(i), i=1, ..., *n* は、汎用固有値。

q

wantq =.TRUE. の場合、終了時に、(*A*, *B*) の順序を変更する左直交変換マトリックスが *Q* に掛けられる。*Q* の先頭 *m* 列は、指定された左固有空間 (収縮部分空間) のペアに対する正規直交基底を形成する。

z

wantz =.TRUE. の場合、終了時に、(*A*, *B*) の順序を変更する左直交変換マトリックスが *Z* に掛けられる。*Z* の先頭 *m* 列は、指定された左固有空間 (収縮部分空間) のペアに対する正規直交基底を形成する。

m

INTEGER。左と右固有空間 (収縮部分空間) のペアの次数、0 ≤ *m* ≤ *n*

pl, pr

REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
ijob = 1、4、5 の場合、*pl* と *pr* は、選択したクラスタに関する左と右固有空間への「投影」のノルムの逆数の下限値。

0 < *pl*, *pr* ≤ 1。*m* = 0 または *m* = *n* の場合は、
pl = *pr* = 1

ijob = 0、2、3 の場合、*pl* と *pr* は参照されない。

dif

REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
配列 DIMENSION は 2
ijob ≥ 2 の場合、*dif*(1:2) に Difu と Difl の推定値が格納される。
ijob = 2 または 4 の場合、*dif*(1:2) は、F- ノルム・ベースの Difu と Difl の上限値。
ijob = 3 または 5 の場合、*dif*(1:2) は、1- ノルム・ベースの Difu と Difl の推定値。*m* = 0 または *n* の場合、
dif(1:2) = F-norm([*A*, *B*])
ijob = 0 または 1 の場合、*dif* は参照されない。

work(1)

ijob ≠ 0 と *info* = 0 の場合、最適パフォーマンスに必要な *lwork* の最小値が、終了時に *work*(1) に格納される。以降の実行では、この *lwork* 値を使用する。

iwork(1)

ijob ≠ 0 と *info* = 0 の場合、最適パフォーマンスに必要な *liwork* の最小値が、終了時に *iwork*(1) に格納される。以降の実行では、この *liwork* 値を使用する。

info

INTEGER
info = 0 の場合は、正常終了している。
info = -*i* の場合は、*i* 番目のパラメータ値が不正である。
info = 1 の場合、変換後のマトリックス・ペア (*A*, *B*) が汎用 Schur 形式から遠すぎるため、(*A*, *B*) の順序変更に失敗した。すなわち、この問題は悪条件である。(*A*, *B*) は部分的に順序変更されている可能性がある。
dif(*)、*pl*、*pr* が要求されている場合は、0 が返される。

?tgsyl

汎用シルベスター式を解く。

```
call stgsyl ( trans, ijob, m, n, a, lda, b, ldb, c, ldc, d, ldd, e,
              lde, f, ldf, scale, dif, work, lwork, iwork, info )
call dtgsyl ( trans, ijob, m, n, a, lda, b, ldb, c, ldc, d, ldd, e,
              lde, f, ldf, scale, dif, work, lwork, iwork, info )
call ctgsyl ( trans, ijob, m, n, a, lda, b, ldb, c, ldc, d, ldd, e,
              lde, f, ldf, scale, dif, work, lwork, iwork, info )
call ztgsyl ( trans, ijob, m, n, a, lda, b, ldb, c, ldc, d, ldd, e,
              lde, f, ldf, scale, dif, work, lwork, iwork, info )
```

説明

このルーチンは、次の汎用シルベスター式を解く。

$$A\,R - L\,B = scale * C$$

$$D\,R - L\,E = scale * F$$

R と L は未知の $m \times n$ のマトリックス、(A, D)、(B, E)、(C, F) はそれぞれ $m \times m$、$n \times n$、$m \times n$ の与えられたマトリックスで、実 / 複素成分を持っている。(A, D) と (B, E) は、汎用実 Schur/Schur 標準形でなければならない。すなわち、A と B は上準三角マトリックス / 上三角マトリックス、D と E は上三角マトリックスでなければならない。

(C, F) は、解 (R, L) で上書きされる。$scale(0 \le scale \le 1)$ は、オーバーフローを避けるために選択された出力スケール係数である。

上の式をマトリックス表記すると次のようになる。
$Zx = scale * b$ を解く。ただし、Z は

$$Z = \begin{pmatrix} kron(I_n, A) & -kron(B', I_m) \\ kron(I_n, D) & -kron(E', I_m) \end{pmatrix}$$

I_k はサイズが k の単位マトリックス、X' は X の転置 / 共役転置マトリックスである。$kron(X, Y)$ は、マトリックス X と Y の Kronecker 積である。*trans* = 'T'(実数の場合のみ) または *trans* = 'C' (複素数の場合のみ) の場合、ルーチン ?tgsyl は、転置 / 共役転置式 $Z' y = scale * b$ を解く。これは、次の R と L を解くのと等価である。

$$A' R + D' L = scale * C$$

$$R B' + L E' = scale * (-F)$$

このケース (*stgsyl*/*dtgsyl* の場合は *trans* = 'T'、ctgsyl/ztgsyl の場合は *trans* = 'C') を使用して、Dif[(*A, D*), (*B, E*)] の 1- ノルム・ベースの推定値、すなわちマトリックス・ペア (*A, D*) と (*B, E*) の隔たりを slacon/clacon を使用して計算する。

ijob ≥ 1 の場合、?tgsyl は、Dif[(*A, D*), (*B, E*)] の Frobenius ノルム・ベースの推定値を計算する。すなわち、Z の最小特異値の逆数に対する下限値の逆数である。これはレベル 3 の BLAS アルゴリズムである。

入力パラメータ

trans
: CHARACTER*1。'N'、'T'、'C' のどれかでなければならない。
trans = 'N' の場合、汎用シルベスター式を解く。
trans = 'T' の場合、「転置」式を解く (実数の場合のみ)。
trans = 'C' の場合、「共役転置」式を解く (複素数の場合のみ)。

ijob
: INTEGER。実行する機能を指定する。
ijob = 0 の場合、汎用シルベスター式のみ解く。
ijob = 1 の場合、*ijob* = 0 と *ijob* = 3 を組み合わせた機能を実行する。
ijob = 2 の場合、*ijob* = 0 と *ijob* = 4 を組み合わせた機能を実行する。
ijob = 3 の場合、Dif[(*A, D*), (*B, E*)] の推定値のみ求める (先読み法を使用する)。

ijob = 4 の場合、Dif[(*A*, *D*), (*B*, *E*)] の推定値のみ求める (部分式に対して ?gecon を使用する)。
trans = 'T' または 'C' の場合、*ijob* は参照されない。

m INTEGER
マトリックス *A* と *D* の次数、マトリックス *C*、*F*、*R*、*L* の行のサイズ。

n INTEGER
マトリックス *B* と *E* の次数、マトリックス *C*、*F*、*R*、*L* の列のサイズ。

a, b, c, d, e, f, work REAL (stgsyl の場合)
DOUBLE PRECISION (dtgsyl の場合)
COMPLEX (ctgsyl の場合)
DOUBLE COMPLEX (ztgsyl の場合)

配列 :

*a(lda, *)* マトリックス *A* (実数の場合は上準三角マトリックス、複素数の場合は上三角マトリックス) を格納する。
a の第 2 次元のサイズは、max(1, *m*) 以上でなければならない。

*b(ldb, *)* マトリックス *B* (実数の場合は上準三角マトリックス、複素数の場合は上三角マトリックス) を格納する。
b の第 2 次元のサイズは、max(1, *n*) 以上でなければならない。

*c(ldc, *)* 汎用シルベスター式の最初のマトリックス式の右辺を格納する (*trans* で指定)。
c の第 2 次元のサイズは、max(1, *n*) 以上でなければならない。

*d(ldd, *)* 上三角マトリックス *D* を格納する。
d の第 2 次元のサイズは、max(1, *m*) 以上でなければならない。

*e(lde, *)* 上三角マトリックス *E* を格納する。
e の第 2 次元のサイズは、max(1, *n*) 以上でなければならない。

*f(ldf, *)* 汎用シルベスター式の 2 番目のマトリックス式の右辺を格納する (*trans* で指定)。
f の第 2 次元のサイズは、max(1, *n*) 以上でなければならない。

work(lwork) はワークスペース配列。ijob = 0 の場合、work は参照されない。

lda　INTEGER。a の第 1 次元のサイズ。max(1, m) 以上。

ldb　INTEGER。b の第 1 次元のサイズ。max(1, n) 以上。

ldc　INTEGER。c の第 1 次元のサイズ。max(1, m) 以上。

ldd　INTEGER。d の第 1 次元のサイズ。max(1, m) 以上。

lde　INTEGER。e の第 1 次元のサイズ。max(1, n) 以上。

ldf　INTEGER。f の第 1 次元のサイズ。max(1, m) 以上。

lwork　INTEGER。配列 work のサイズ、lwork ≥ 1
ijob = 1 または 2 で、trans = 'N' の場合は、
lwork ≥ 2mn

iwork　INTEGER。ワークスペース配列 DIMENSION は、実数の場合は (m+n+6) 以上、複素数の場合は (m+n+2) 以上。
ijob = 0 の場合、iwork は参照されない。

出力パラメータ

c　ijob = 0、1、2 の場合、解 R で上書きされる。
ijob = 3 または 4 で trans = 'N' の場合、c に R (Dif 推定値の計算中に求められた解) が格納される。

f　ijob = 0、1、2 の場合、解 L で上書きされる。
ijob = 3 または 4 および trans = 'N' の場合、f に L (Dif 推定値の計算中に求められた解) が格納される。

dif　REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
終了時に、dif は、Dif 関数の逆数の下限値の逆数が格納される。すなわち、dif は、
Dif[(A, D), (B, E)] = sigma_min(Z) の上限値である。
ijob = 0、または trans = 'T' (実数の場合) または trans = 'C' (複素数の場合) の場合、dif は設定されない。

scale	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 終了時に、*scale* に、汎用シルベスター式のスケール係数が入る。 $0 < scale < 1$ の場合、*c* と *f* にはそれぞれ、わずかに摂動する系の解 R と L が入るが、入力マトリックス A、B、D、E は変更されない。$scale = 0$ の場合、*c* と *f* にはそれぞれ、$C = F = 0$ の均一な系の解 R と L が入る。通常は、$scale = 1$
work(1)	$ijob \neq 0$ と $info = 0$ の場合、最適パフォーマンスに必要な *lwork* の最小値が、終了時に *work*(1) に格納される。以降の実行では、この *lwork* 値を使用する。
info	INTEGER $info = 0$ の場合は、正常終了している。 $info = -i$ の場合は、*i* 番目のパラメータ値が不正である。 $info > 0$ の場合、(A, D) と (B, E) は、同じか近い固有値を持つ。

?tgsna

汎用実 Schur 標準形のマトリックス・ペアにおいて、指定された固有値と固有ベクトルに対する条件数の逆数を見積もる。

```
call stgsna ( job, howmny, select, n, a, lda, b, ldb, vl, ldvl, vr,
              ldvr, s, dif, mm, m, work, lwork, iwork, info )
call dtgsna ( job, howmny, select, n, a, lda, b, ldb, vl, ldvl, vr,
              ldvr, s, dif, mm, m, work, lwork, iwork, info )
call ctgsna ( job, howmny, select, n, a, lda, b, ldb, vl, ldvl, vr,
              ldvr, s, dif, mm, m, work, lwork, iwork, info )
call ztgsna ( job, howmny, select, n, a, lda, b, ldb, vl, ldvl, vr,
              ldvr, s, dif, mm, m, work, lwork, iwork, info )
```

説明

実数の場合 (stgsna/dtgsna の場合) は、汎用実 Schur 標準形のマトリックス・ペア (A, B)、または直交マトリックス Q と Z を持つ任意のマトリックス・ペア $(Q A Z^T, Q B Z^T)$ において、指定された固有値と固有ベクトルの条件数の逆数を見積もる。
(A, B) は、汎用実 Schur 形式 (sgges/dgges から返される形式) でなければならない。すなわち、A は 1×1 と 2×2 の対角ブロックを持つブロック上三角マトリックス、B は上三角マトリックスでなければならない。

複素数の場合 (ctgsna/ztgsna の場合) は、マトリックス・ペア (A, B) において、指定された固有値と固有ベクトルの条件数の逆数を見積もる。(A, B) は汎用 Schur 標準形でなければならない。すなわち、A と B はどちらも上三角マトリックスでなければならない。

入力パラメータ

job CHARACTER*1。固有値と固有ベクトルに対して条件数の逆数を計算するかどうか指定する。
'E'、'V'、'B' のどれかでなければならない。
job ='E' の場合、固有値のみ (*s* を計算する)。

job='V' の場合、固有ベクトルのみ (dif を計算する)。
job='B' の場合、固有値と固有ベクトルの両方 (s と dif を計算する)。

howmny
CHARACTER*1。'A' または 'S' でなければならない。
howmny='A' の場合、すべての固有ペアに対して条件数を計算する。
howmny='S' の場合、論理配列 select で指定した固有ペアに対してのみ条件数を計算する。

select
LOGICAL。
配列 DIMENSION は max(1, n) 以上。
howmny='S' の場合、select は、条件数を計算すべき固有ペアを指定する。
howmny='A' の場合、select は参照されない。
実数の場合 :
実固有値 ω_j, に対応する固有ペアの条件数を選択するには、select(j) を .TRUE. に設定しなければならない。複素共役固有値ペア ω_j と ω_{j+1} の条件数を選択するには、select(j) と select(j+1) のどちらかを .TRUE. に設定しなければならない。
複素数の場合 :
j 番目の固有値と固有ベクトルの条件数を選択するには、select(j) を .TRUE. に設定しなければならない。

n
INTEGER。正方マトリックス・ペア (A, B) の次数 ($n \geq 0$)。

a, b, vl, vr, work
REAL (stgsna の場合)
DOUBLE PRECISION (dtgsna の場合)
COMPLEX (ctgsna の場合)
DOUBLE COMPLEX (ztgsna の場合)
配列 :
a(lda, *) マトリックス・ペア (A, B) における上準三角 (実数の場合) または上三角 (複素数の場合) マトリックス A を格納する。
a の第 2 次元のサイズは、max(1, n) 以上でなければならない。

`b(ldb, *)` マトリックス・ペア (*A*, *B*) における上三角マトリックス *B* を格納する。
`b` の第 2 次元のサイズは、max(1, `n`) 以上でなければならない。

`job`='`E`' または '`B`' の場合、`vl(ldvl, *)` には、`howmny` と `select` で指定した固有ペアに対応する、(*A*, *B*) の左固有ベクトルが入っていなければならない。この固有ベクトルは、`?tgevc` で返される通りに、`vl` の連続した列に入っていなければならない。
`job`='`V`' の場合、`vl` は参照されない。
`vl` の第 2 次元のサイズは、max(1, `m`) 以上でなければならない。

`job`='`E`' または '`B`' の場合、`vr(ldvr, *)` には、`howmny` と `select` で指定した固有ペアに対応する、(*A*, *B*) の右固有ベクトルが入っていなければならない。この固有ベクトルは、`?tgevc` で返される通りに、`vr` の連続した列に入っていなければならない。
`job`='`V`' の場合、`vr` は参照されない。
`vr` の第 2 次元のサイズは、max(1, `m`) 以上でなければならない。

`work(lwork)` はワークスペース配列。`job`='`E`' の場合、`work` は参照されない。

`lda` INTEGER。`a` の第 1 次元のサイズ。max(1, `n`) 以上。

`ldb` INTEGER。`b` の第 1 次元のサイズ。max(1, `n`) 以上。

`ldvl` INTEGER。`vl` の第 1 次元のサイズ。`ldvl` ≥ 1
`job`='`E`' または '`B`' の場合、`ldvl` ≥ max(1, `n`)

`ldvr` INTEGER。`vr` の第 1 次元のサイズ。`ldvr` ≥ 1
`job`='`E`' または '`B`' の場合、`ldvr` ≥ max(1, `n`)

`mm` INTEGER。配列 `s` と `dif` の成分の個数 (`mm` ≥ `m`)

`lwork` INTEGER。配列 `work` のサイズ。
実数の場合 :
`lwork` ≥ `n`
`job`='`V`' または '`B`' の場合、`lwork` ≥ 2`n`(`n`+2)+16
複素数の場合 :
`lwork` ≥ 1
`job`='`V`' または '`B`' の場合、`lwork` ≥ 2n^2

iwork INTEGER。ワークスペース配列 DIMENSION は、実数の場合は (*n*+6) 以上、複素数の場合は (*n*+2) 以上。
ijob = 'E' の場合、*iwork* は参照されない。

出力パラメータ

s REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
配列 DIMENSION は *mm*
job = 'E' または 'B' の場合、選択した固有値の条件数の逆数が、配列の連続成分に格納される。
job = 'V' の場合、*s* は参照されない。
実数の場合 :
固有値の複素共役ペアに対して、*s* の 2 つの連続成分が同じ値に設定される。
したがって、*s*(j)、*dif*(j)、*vl* と *vr* の j 番目の列が、同じ固有ペアに対応することになる (ただし、すべての固有ペアが選択される場合を除き、j 番目の固有ペアに対応することにはならない)。

dif REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
配列 DIMENSION は *mm*
job = 'V' または 'B' の場合、選択した固有ベクトルの条件数の逆数の推定値が、配列の連続成分に格納される。*dif*(j) を計算するための固有値の順序変更ができなければ、*dif*(j) が 0 に設定される。これは、True 値が非常に小さい場合にのみ起こる。
job = 'E' の場合、*dif* は参照されない。
実数の場合 :
1 つの複素固有ベクトルに対して、*dif* の 2 つの連続成分が同じ値に設定される。
複素数の場合 :
select で指定された 1 つの固有値/固有ベクトルに対して、Difl の Frobenius ノルム・ベースの推定値が *dif* に格納される。

m	INTEGER。配列 *s* と *dif* において、指定された条件数を格納するのに使用した成分の個数。選択された 1 つの固有値に対して、1 つの成分が使用される。 *howmny* = 'A' の場合、*m* は *n* に設定される。
work(1)	*job* が 'E' でなく *info* = 0 の場合、最適パフォーマンスに必要な *lwork* の最小値が、終了時に *work*(1) に格納される。以降の実行では、この *lwork* 値を使用する。
info	INTEGER *info* = 0 の場合は、正常終了している。 *info* = -*i* の場合は、*i* 番目のパラメータ値が不正である。

汎用特異値分解

この節では、2 つのマトリックス A と B に対して次のような汎用特異値分解 (GSVD) を行う LAPACK 計算ルーチンについて説明する。

$$U^H AQ = D_1 * (0 \;\; R)$$

$$V^H BQ = D_2 * (0 \;\; R)$$

U、V、Q は直交 / ユニタリ・マトリックス、R は非特異上三角マトリックス、D_1、D_2 は「対角」マトリックス (詳しい構造については、各ルーチンの説明を参照) である。

表 5-7 汎用特異値分解のための計算ルーチン

ルーチン名	機能
?ggsvp	汎用 SVD のための予備的な分解を行う。
?tgsja	2 つの上三角マトリックスまたは台形マトリックスの汎用 SVD を求める。

一般三角マトリックス・ペアの GSVD を得るには、上記のルーチンを使用することも、ドライバ・ルーチン ?ggsvd を使用することもできる。

?ggsvp

汎用 SVD のための予備的な分解を行う。

```
call sggsvp ( jobu, jobv, jobq, m, p, n, a, lda, b, ldb, tola, tolb,
              k, l, u, ldu, v, ldv, q, ldq, iwork, tau, work, info )
call dggsvp ( jobu, jobv, jobq, m, p, n, a, lda, b, ldb, tola, tolb,
              k, l, u, ldu, v, ldv, q, ldq, iwork, tau, work, info )
call cggsvp ( jobu, jobv, jobq, m, p, n, a, lda, b, ldb, tola, tolb,
              k, l, u, ldu, v, ldv, q, ldq, iwork, rwork, tau, work,
              info )
call zggsvp ( jobu, jobv, jobq, m, p, n, a, lda, b, ldb, tola, tolb,
              k, l, u, ldu, v, ldv, q, ldq, iwork, rwork, tau, work,
              info )
```

説明

このルーチンは、次のような直交マトリックス U、V、Q を求める。

$$U^H A\,Q = \begin{array}{cc} & \begin{array}{ccc} n-k-l & k & l \end{array} \\ \begin{array}{c} k \\ l \\ m-k-l \end{array} & \begin{pmatrix} 0 & A_{12} & A_{13} \\ 0 & 0 & A_{23} \\ 0 & 0 & 0 \end{pmatrix} \end{array}$$ 、m-k-l ≥ 0 の場合

$$= \begin{array}{cc} & \begin{array}{ccc} n-k-l & k & l \end{array} \\ \begin{array}{c} k \\ m-k \end{array} & \begin{pmatrix} 0 & A_{12} & A_{13} \\ 0 & 0 & A_{23} \end{pmatrix} \end{array}$$ 、m-k-l < 0 の場合

$$V^H B\,Q = \begin{array}{cc} & \begin{array}{ccc} n-k-l & k & l \end{array} \\ \begin{array}{c} l \\ p-l \end{array} & \begin{pmatrix} 0 & 0 & B_{13} \\ 0 & 0 & 0 \end{pmatrix} \end{array}$$

$k \times k$ のマトリックス A_{12} と $l \times l$ のマトリックス B_{13} は、非特異上三角マトリックスである。A_{23} は、$m-k-l \geq 0$ の場合は $l \times l$ の上三角マトリックス、それ以外の場合は $(m-k) \times l$ の上台形マトリックスである。$k+l$ の和は、$(m+p) \times n$ のマトリックス $(A^H, B^H))^H$ の有効階数に等しい。

この分解は、汎用特異値分解 (GSVD) の予備ステップとして行われる。サブルーチン ?ggsvd を参照。

入力パラメータ

jobu CHARACTER*1。'U' または 'N' でなければならない。
jobu ='U' の場合、直交 / ユニタリ・マトリックス U が計算される。
jobu ='N' の場合、U は計算されない。

jobv CHARACTER*1。'V' または 'N' でなければならない。
jobv ='V' の場合、直交 / ユニタリ・マトリックス V が計算される。
jobv ='N' の場合、V は計算されない。

jobq CHARACTER*1。'Q' または 'N' でなければならない。
jobq ='Q' の場合、直交 / ユニタリ・マトリックス Q が計算される。
jobq ='N' の場合、Q は計算されない。

m INTEGER。マトリックス A の行数 ($m \geq 0$)。

p INTEGER。マトリックス B の行数 ($p \geq 0$)。

n INTEGER。マトリックス A と B の列数 ($n \geq 0$)。

a, b, tau, work REAL (sggsvp の場合)
DOUBLE PRECISION (dggsvp の場合)
COMPLEX (cggsvp の場合)
DOUBLE COMPLEX (zggsvp の場合)
配列 :
a(*lda*, *) $m \times n$ のマトリックス A を格納する。
a の第 2 次元のサイズは、max(1, *n*) 以上でなければならない。
b(*ldb*, *) $p \times n$ のマトリックス B を格納する。
b の第 2 次元のサイズは、max(1, *n*) 以上でなければならない。
tau(*) は、ワークスペース配列。サイズは max(1, *n*) 以上でなければならない。
work(*) は、ワークスペース配列。サイズは、max(1, 3*n*, *m*, *p*) 以上でなければならない。

lda INTEGER。*a* の第 1 次元のサイズ。max(1, *m*) 以上。

ldb INTEGER。*b* の第 1 次元のサイズ。max(1, *p*) 以上。

tola, tolb REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
tola と *tolb* は、マトリックス *B* と部分ブロック *A* の有効階数を決定するためのしきい値である。一般に、次のように設定される。
tola = max(*m*, *n*)*‖*A*‖*MACHEPS
tolb = max(*p*, n)*‖*B*‖*MACHEPS
tola と *tolb* のサイズが、分解の後退誤差のサイズに影響する可能性がある。

ldu INTEGER。出力配列 *u* の第 1 次元のサイズ。
jobu = 'U' ならば *ldu* ≥ max(1, *m*)、それ以外ならば *ldu* ≥ 1

ldv INTEGER。出力配列 *v* の第 1 次元のサイズ。
jobv = 'V' ならば *ldv* ≥ max(1, *p*)、それ以外ならば *ldv* ≥ 1

ldq INTEGER。出力配列 *q* の第 1 次元のサイズ。
jobq = 'Q' ならば *ldq* ≥ max(1, *n*)、それ以外ならば *ldq* ≥ 1

iwork INTEGER。ワークスペース配列 DIMENSION は max(1, *n*) 以上。

rwork REAL (cggsvp の場合)
DOUBLE PRECISION (zggsvp の場合)
ワークスペース配列 DIMENSION は max(1, 2*n*) 以上。
複素数の場合にのみ使用する。

出力パラメータ

a **説明**で示した三角 (台形) マトリックスで上書きされる。

b **説明**で示した三角マトリックスで上書きされる。

k, l INTEGER
終了時に、*k* と *l* に部分ブロックのサイズが入る。
k+*l* は、$(A^H, B^H)^H$ の有効階数に一致する。

u, v, q

REAL (sggsvp の場合)
DOUBLE PRECISION (dggsvp の場合)
COMPLEX (cggsvp の場合)
DOUBLE COMPLEX (zggsvp の場合)
配列 :
jobu = 'U' の場合、*u(ldu, *)* に、直交 / ユニタリ・マトリックス *U* が格納される。
u の第 2 次元のサイズは、max(1, *m*) 以上でなければならない。
jobu = 'N' の場合、*u* は参照されない。
jobv = 'V' の場合、*v(ldv, *)* に、直交 / ユニタリ・マトリックス *V* が格納される。
v の第 2 次元のサイズは、max(1, *m*) 以上でなければならない。
jobv = 'N' の場合、*v* は参照されない。
jobq = 'Q' の場合、*q(ldq, *)* に、直交 / ユニタリ・マトリックス *Q* が格納される。
q の第 2 次元のサイズは、max(1, *n*) 以上でなければならない。
jobq = 'N' の場合、*q* は参照されない。

info

INTEGER
info = 0 の場合は、正常終了している。
info = -*i* の場合は、*i* 番目のパラメータ値が不正である。

?tgsja

2 つの上三角マトリックスまたは台形
マトリックスの汎用 SVD を計算する。

```
call stgsja ( jobu, jobv, jobq, m, p, n, k, l, a, lda, b, ldb, tola,
              tolb, alpha, beta, u, ldu, v, ldv, q, ldq, work, ncycle,
              info )
call dtgsja ( jobu, jobv, jobq, m, p, n, k, l, a, lda, b, ldb, tola,
              tolb, alpha, beta, u, ldu, v, ldv, q, ldq, work, ncycle,
              info )
call ctgsja ( jobu, jobv, jobq, m, p, n, k, l, a, lda, b, ldb, tola,
              tolb, alpha, beta, u, ldu, v, ldv, q, ldq, work, ncycle,
              info )
call ztgsja ( jobu, jobv, jobq, m, p, n, k, l, a, lda, b, ldb, tola,
              tolb, alpha, beta, u, ldu, v, ldv, q, ldq, work, ncycle,
              info )
```

説明

このルーチンは、2 つの実数 / 複素数の上三角マトリックス (または台形マトリックス) A と B に対して、汎用特異値分解 (GSVD) を計算する。呼び出し時に、マトリックス A と B は次の形式であるとみなされる。この形式は、$m \times n$ の一般マトリックス A と $p \times n$ の一般マトリックス B を、予備的サブルーチン ?ggsvp で前処理して得られる。

$$A = \begin{array}{c} \\ k \\ l \\ m-k-l \end{array} \begin{array}{c} \begin{array}{ccc} n-k-l & k & l \end{array} \\ \begin{pmatrix} 0 & A_{12} & A_{13} \\ 0 & 0 & A_{23} \\ 0 & 0 & 0 \end{pmatrix} \end{array}$$ 、$m-k-l \geq 0$ の場合

$$= \begin{array}{c} \\ k \\ m-k \end{array} \begin{array}{c} \begin{array}{ccc} n-k-l & k & l \end{array} \\ \begin{pmatrix} 0 & A_{12} & A_{13} \\ 0 & 0 & A_{23} \end{pmatrix} \end{array}$$ 、$m-k-l < 0$ の場合

$$B = \begin{array}{c} \\ l \\ p-l \end{array} \begin{array}{c} \begin{array}{ccc} n-k-l & k & l \end{array} \\ \begin{pmatrix} 0 & 0 & B_{13} \\ 0 & 0 & 0 \end{pmatrix} \end{array}$$

$k \times k$ のマトリックス A_{12} と $l \times l$ のマトリックス B_{13} は非特異上三角マトリックスである。A_{23} は、$m-k-l \geq 0$ の場合は $l \times l$ の上三角マトリックス、その他の場合は $(m-k) \times l$ の上台形マトリックスである。

終了時は、

$U^H A Q = D_1 * (0 \quad R)$, $V^H B Q = D_2 * (0 \quad R)$

U、V、Q は直交 / ユニタリ・マトリックス、R は非特異上三角マトリックス、D_1 と D_2 は次の構造を持つ「対角」マトリックスである。

$m-k-l \geq 0$ の場合、

$$D_1 = \begin{array}{c} \\ k \\ l \\ m-k-l \end{array} \begin{array}{c} \begin{array}{cc} k & l \end{array} \\ \begin{pmatrix} I & 0 \\ 0 & C \\ 0 & 0 \end{pmatrix} \end{array}$$

$$D_2 = \begin{array}{c} \\ l \\ p-l \end{array} \begin{array}{c} \begin{array}{cc} k & l \end{array} \\ \begin{pmatrix} 0 & S \\ 0 & 0 \end{pmatrix} \end{array}$$

$$(0 \quad R) = \begin{array}{c} \\ k \\ l \end{array} \begin{array}{c} \begin{array}{ccc} n-k-l & k & l \end{array} \\ \begin{pmatrix} 0 & R_{11} & R_{12} \\ 0 & 0 & R_{22} \end{pmatrix} \end{array}$$

$C = \mathrm{diag}\,(alpha(k+1), \ldots, alpha(k+l))$

$S = \mathrm{diag}\,(beta(k+1), \ldots, beta(k+l))$

$C^2 + S^2 = \mathrm{I}$

R は、終了時に $a(1{:}k+l, n-k-l+1{:}n)$ に格納される。

m-k-l < 0 の場合、

$$D_1 = \begin{array}{c} \\ k \\ m-k \end{array} \begin{array}{c} \begin{array}{ccc} k & m-k & k+l-m \end{array} \\ \begin{pmatrix} I & 0 & 0 \\ 0 & C & 0 \end{pmatrix} \end{array}$$

m-k-l < 0 の場合、

$$D_2 = \begin{array}{c} \\ m-k \\ k+l-m \\ p-l \end{array} \begin{array}{c} \begin{array}{ccc} k & m-k & k+l-m \end{array} \\ \begin{pmatrix} 0 & S & 0 \\ 0 & 0 & I \\ 0 & 0 & 0 \end{pmatrix} \end{array}$$

$$(0 \quad R) = \begin{array}{c} \\ k \\ m-k \\ k+l-m \end{array} \begin{array}{c} \begin{array}{cccc} n-k-l & k & m-k & k+l-m \end{array} \\ \begin{pmatrix} 0 & R_{11} & R_{12} & R_{13} \\ 0 & 0 & R_{22} & R_{23} \\ 0 & 0 & 0 & R_{33} \end{pmatrix} \end{array}$$

C = diag (alpha(k+1), ..., alpha(m))
S = diag (beta(k+1), ..., beta(m))
$C^2 + S^2 = \mathrm{I}$

終了時に、 $\begin{pmatrix} R_{11} R_{12} R_{13} \\ 0 \ R_{22} R_{23} \end{pmatrix}$ が a(1:m, n-k-l+1:n) に格納され、

R_{33} が b(m-k+1:l, n+m-k-l+1:n) に格納される。

オプションで、直交 / ユニタリ変換マトリックス U、V、Q を計算できる。これらのマトリックスは明示的に生成もできるし、入力マトリックス U_1、V_1、Q_1 に後からも掛けられる。

入力パラメータ

jobu CHARACTER*1。'U'、'I'、'N' のどれかでなければならない。

jobu ='U' の場合、呼び出し時に、*u* に直交 / ユニタリ・マトリックス U_1 が格納されていなければならない。

jobu ='I' の場合、*u* は単位マトリックスに初期化される。

jobu ='N' の場合、*u* は計算されない。

jobv CHARACTER*1。'V'、'I'、'N' のどれかでなければならない。

jobv ='V' の場合、呼び出し時に、*v* に直交 / ユニタリ・マトリックス V_1 が格納されていなければならない。

jobv ='I' の場合、*v* は単位マトリックスに初期化される。

jobv ='N' の場合、*v* は計算されない。

jobq CHARACTER*1。'Q'、'I'、'N' のどれかでなければならない。

jobq ='Q' の場合、呼び出し時に、*q* に直交 / ユニタリ・マトリックス Q_1 が格納されていなければならない。

jobq ='I' の場合、*q* は単位マトリックスに初期化される。

jobq ='N' の場合、*q* は計算されない。

m INTEGER。マトリックス *A* の行数 ($m \geq 0$)。

p INTEGER。マトリックス *B* の行数 ($p \geq 0$)。

n INTEGER。マトリックス *A* と *B* の列数 ($n \geq 0$)。

k, l INTEGER。入力マトリックス *A* と *B* の部分ブロックを指定する。?tgsja は、これから GSVD を計算する

a, b, u, v, q, work REAL (stgsja の場合)
DOUBLE PRECISION (dtgsja の場合)
COMPLEX (ctgsja の場合)
DOUBLE COMPLEX (ztgsja の場合)

配列 :

a(*lda*, *) $m \times n$ のマトリックス *A* を格納する。
a の第 2 次元のサイズは、max(1, *n*) 以上でなければならない。

b(*ldb*, *) $p \times n$ のマトリックス *B* を格納する。
b の第 2 次元のサイズは、max(1, *n*) 以上でなければならない。

jobu = 'U' の場合、u(ldu, *) には、マトリックス U_1 (通常は、?ggsvp から返された直交 / ユニタリ・マトリックス) が格納されていなければならない。
u の第 2 次元のサイズは、max(1, m) 以上でなければならない。

jobv = 'V' の場合、v(ldv, *) には、マトリックス V_1 (通常は、?ggsvp から返された直交 / ユニタリ・マトリックス) が格納されていなければならない。
v の第 2 次元のサイズは、max(1, p) 以上でなければならない。

jobq = 'Q' の場合、q(ldq, *) には、マトリックス Q_1 (通常は、?ggsvp から返された直交 / ユニタリ・マトリックス) が格納されていなければならない。
q の第 2 次元のサイズは、max(1, n) 以上でなければならない。

work(*) は、ワークスペース配列。サイズは、max(1, 2n) 以上でなければならない。

lda　INTEGER。a の第 1 次元のサイズ。max(1, m) 以上。

ldb　INTEGER。b の第 1 次元のサイズ。max(1, p) 以上。

ldu　INTEGER。配列 u の第 1 次元のサイズ。
jobu = 'U' ならば ldu ≥ max(1, m)、それ以外ならば ldu ≥ 1

ldv　INTEGER。配列 v の第 1 次元のサイズ。
jobv = 'V' ならば ldv ≥ max(1, p)、それ以外ならば ldv ≥ 1

ldq　INTEGER。配列 q の第 1 次元のサイズ。
jobq = 'Q' ならば ldq ≥ max(1, n)、それ以外ならば ldq ≥ 1

tola, tolb　REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
tola と tolb は、Jacobi-Kogbetliantz 反復法における収束基準である。一般に、?ggsvp で使用したのと同じ値にする。
tola = max(m, n)*||A||*MACHEPS
tolb = max(p, n)*||B||*MACHEPS

出力パラメータ

a　終了時に、a(n-k+1:n, 1:min(k+l, m)) に、三角マトリックス R、またはその一部が格納される。

b	終了時に、必要ならば、*b*(*m*-*k*+1: *l*, *n*+*m*-*k*-*l*+1: *n*) に *R* の一部が格納される。
alpha, beta	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列 DIMENSION は max(1, *n*) 以上。 *A* と *B* の汎用特異値ペアが、次のように格納される。 *alpha*(1:*k*) = 1 *beta*(1:*k*) = 0 *m*-*k*-*l* ≥ 0 の場合は、 *alpha*(*k*+1:*k*+*l*) = diag(*C*), *beta*(*k*+1:*k*+*l*) = diag(*S*), *m*-*k*-*l* < 0 の場合は、 *alpha*(*k*+1:*m*) = *C*, *alpha*(*m*+1:*k*+*l*) = 0 *beta*(*k*+1:*m*) = *S*, *beta*(*m*+1:*k*+*l*) = 1 *k*+*l* <*n* の場合は、さらに *alpha*(*k*+*l*+1:*n*) = 0 *beta*(*k*+*l*+1:*n*) = 0
u	*jobu* ='I' の場合、*u* に、直交 / ユニタリ・マトリックス *U* が格納される。 *jobu* ='U' の場合、*u* に積 U_1U が格納される。 *jobu* ='N' の場合、*u* は参照されない。
v	*jobv* ='I' の場合、*v* に、直交 / ユニタリ・マトリックス *U* が格納される。 *jobv* ='V' の場合、*v* に積 V_1V が格納される。 *jobv* ='N' の場合、*v* は参照されない。
q	*jobq* ='I' の場合、*q* に、直交 / ユニタリ・マトリックス *U* が格納される。 *jobq* ='Q' の場合、*q* に積 Q_1Q が格納される。 *jobq* ='N' の場合、*q* は参照されない。
ncycle	INTEGER。収束するのに要したサイクル数。

info INTEGER

info = 0 の場合は、正常終了している。

info = *-i* の場合は、*i* 番目のパラメータ値が不正である。

info = 1 の場合は、この方法では MAXIT サイクル以内に収束しなかった。

ドライバ・ルーチン

LAPACK ドライバ・ルーチンを使用すると、ある問題を完全に解くのに、1 つのルーチンを呼び出すだけで済む。各ドライバ・ルーチンは一般に、一連の計算ルーチンを適切に組み合わせて呼び出す。
以下の各節で、ドライバ・ルーチンについて説明する。

線形最小二乗 (LLS) 問題
汎用 LLS 問題
対称固有値問題
非対称固有値問題
特異値分解
汎用対称固有値問題
汎用非対称固有値問題

線形最小二乗 (LLS) 問題

この節では、線形最小二乗問題を解くための LAPACK ドライバ・ルーチンについて説明する。この節で詳しく説明するドライバ・ルーチン名を、表 5-8 に示す。

表 5-8 LLS 問題を解くためのドライバ・ルーチン

ルーチン名	機能
?gels	QR または LQ 因子分解を使用して、最大階数のマトリックスで表される、優決定または劣決定の線形問題を解く。
?gelsy	A の完全直交因子分解を使用して、線形最小二乗問題の最小ノルム解を求める。
?gelss	A の特異値分解を使用して、線形最小二乗問題の最小ノルム解を求める。
?gelsd	A の特異値分解と分割統治法を使用して、線形最小二乗問題の最小ノルム解を求める。

?gels

QR または *LQ* 因子分解を使用して、
最大階数のマトリックスで表される、
優決定または劣決定の線形問題を解く。

```
call sgels ( trans, m, n, nrhs, a, lda, b, ldb, work, lwork, info )
call dgels ( trans, m, n, nrhs, a, lda, b, ldb, work, lwork, info )
call cgels ( trans, m, n, nrhs, a, lda, b, ldb, work, lwork, info )
call zgels ( trans, m, n, nrhs, a, lda, b, ldb, work, lwork, info )
```

説明

このルーチンは、*A* の *QR* または *LQ* 因子分解を使用して、`m` × `n` のマトリックス *A* (またはその転置 / 共役転置マトリックス) で表される、実数 / 複素数の優決定 / 劣決定の線形問題を解く。ただし、*A* は最大階数のマトリックスとする。

次のオプションが提供されている。

1. `trans` = `'N'` で、`m` ≥ `n` の場合 : 優決定の最小二乗解を求める。すなわち次の最小二乗問題を解く。

$$\text{minimize} \quad \| b - A\,x \|_2$$

2. `trans` = `'N'` で、`m` < `n` の場合 : 劣決定の問題 $A\,X = B$ の最小ノルム解を求める。

3. `trans` = `'T'` または `'C'` で、`m` ≥ `n` の場合 : 劣決定の問題 $A^H X = B$ の最小ノルム解を求める。

4. `trans` = `'T'` または `'C'` で、`m` ≥ `n` の場合 : 優決定の最小二乗解を求める。すなわち次の最小二乗問題を解く。

$$\text{minimize} \quad \| b - A^H x \|_2$$

複数の右辺のベクトル *b* と解ベクトル *x* を、一度の呼び出しで処理できる。これらのベクトルは、`m` × `nrhs` の右辺のマトリックス *B* と `n` × `nrh` の解マトリックス *X* の各列に格納される。

入力パラメータ

trans	CCHARACTER*1。'N'、'T'、'C' のどれかでなければならない。 *trans* ='N' の場合、マトリックス A で表される線形問題。 *trans* ='T' の場合、転置マトリックス A^T で表される線形問題 (実数の場合のみ)。 *trans* ='C' の場合、共役転置マトリックス A^H で表される線形問題 (複素数の場合のみ)。
m	INTEGER。マトリックス A の行数 ($m \geq 0$)。
n	INTEGER。マトリックス A の列数 ($n \geq 0$)。
nrhs	INTEGER。右辺のベクトルの個数、すなわちマトリックス B の列数 ($nrhs \geq 0$)。
a, b, work	REAL (sgels の場合) DOUBLE PRECISION (dgels の場合) COMPLEX (cgels の場合) DOUBLE COMPLEX (zgels の場合) 配列 : *a(lda, *)* には、$m \times n$ のマトリックス A を格納する。 *a* の第 2 次元のサイズは max(1, *n*) 以上でなければならない。 *b(ldb, *)* には、右辺のベクトルが各列に入ったマトリックス B を格納する。B は、*trans* ='N' の場合は $m \times nrhs$、*trans* ='T' または 'C' の場合は $n \times nrhs$。 *b* の第 2 次元のサイズは max(1, *nrhs*) 以上でなければならない。 *work(lwork)* は、ワークスペース配列。
lda	INTEGER。*a* の第 1 次元のサイズ。max(1, *m*) 以上。
ldb	INTEGER。*b* の第 1 次元のサイズ。max(1, *m*, *n*) 以上でなければならない。
lwork	INTEGER。配列 *work* のサイズ。 min(*m*, *n*)+max(1, *m*, *n*, *nrhs*) 以上でなければならない。 *lwork* の推奨値は、「アプリケーション・ノート」を参照。

出力パラメータ

a

終了時に、因子分解データで次のように上書きされる。

m ≥ *n* の場合、マトリックス *A* の *QR* 因子分解の詳細 (?geqrf から返された値) が、配列 *a* に格納される。

m < *n* の場合、マトリックス *A* の *LQ* 因子分解の詳細 (?gelqf から返された値) が、配列 *a* に格納される。

b

各列が解ベクトルで上書きされる。

trans = 'N' で *m* ≥ *n* の場合、*b* の 1 ～ *n* 行に、最小二乗解のベクトルが格納される。各列の解の二乗和の誤差は、それぞれの列の *n*+1 ～ *m* の成分の二乗和で与えられる。

trans = 'N' で *m* < *n* の場合、*b* の 1 ～ *n* 行に、最小ノルム解のベクトルが格納される。

trans = 'T' または 'C' で *m* ≥ *n* の場合、*b* の 1 ～ *m* 行に、最小ノルム解のベクトルが格納される。

trans = 'T' または 'C' で *m* < *n* の場合、*b* の 1 ～ *m* 行に、最小ノルム解のベクトルが格納される。各列の解の二乗和の誤差は、それぞれの列の *m*+1 ～ *n* の成分の二乗和で与えられる。

work(1)

終了時に、*info* = 0 ならば、最適パフォーマンスに必要な *lwork* の最小値が、*work*(1) に格納される。以降の実行では、この *lwork* 値を使用する。

info

INTEGER

info = 0 の場合は、正常終了している。

info = -*i* の場合は、*i* 番目のパラメータ値が不正である。

アプリケーション・ノート

パフォーマンスを改善するには、*lwork*=min(*m*, *n*)+max(1, *m*, *n*, *nrhs*) **blocksize* に設定してみる。*blockzise* は、**ブロック化アルゴリズム**の最適パフォーマンスに必要な値で、マシンによって異なる (通常は 16 ～ 64)。

必要なワークスペース・サイズが不明な場合は、*lwork* に大きな値を設定して実行してみる。ルーチン終了後に *work*(1) の値を調べ、以降はその値を使用する。

?gelsy

A の完全直交因子分解を使用して、線形最小二乗問題の最小ノルム解を求める。

```
call sgelsy ( m, n, nrhs, a, lda, b, ldb, jpvt, rcond, rank, work,
              lwork, info )
call dgelsy ( m, n, nrhs, a, lda, b, ldb, jpvt, rcond, rank, work,
              lwork, info )
call cgelsy ( m, n, nrhs, a, lda, b, ldb, jpvt, rcond, rank, work,
              lwork, rwork, info )
call zgelsy ( m, n, nrhs, a, lda, b, ldb, jpvt, rcond, rank, work,
              lwork, rwork, info )
```

説明

このルーチンは、*A* の完全直交因子分解を使用して、実数 / 複素数の線形最小二乗問題、すなわち

$$\text{minimize} \quad \| b - A x \|_2$$

の最小ノルム解を求める。*A* は `m` × `n` のマトリックスで、階数不足の可能性がある。
複数の右辺のベクトル `b` と解ベクトル `x` を、一度の呼び出しで処理できる。これらのベクトルは、`m` × `nrhs` の右辺のマトリックス *B* と `n` × `nrh` の解マトリックス *X* の各列に格納される。

このルーチンは、まず列のピボット演算を用いた *QR* 因子分解を実行する。

$$AP = Q \begin{pmatrix} R_{11} R_{12} \\ 0 \quad R_{22} \end{pmatrix}$$

R_{11} は最大の先頭部分マトリックスで、条件数は 1/`rcond` 以下と見積もられる。R_{11} の次数 (`rank`) が、*A* の有効階数である。

次に、R_{22} は無視できるとみなし、R_{12} を右からの直交 / ユニタリ変換でゼロにして、次のような完全直交因子分解を得る。

$$AP = Q \begin{pmatrix} T_{11} & 0 \\ 0 & 0 \end{pmatrix} Z$$

最小ノルム解は次のようになる。

$$x = PZ^H \begin{pmatrix} T_{11}^{-1} Q_1^H b \\ 0 \end{pmatrix}$$

Q_1 は、Q の先頭 *rank* 列である。このルーチンは、元の ?gelsx と基本的に同じであるが、次の点が異なる。

- サブルーチン ?geqpf ではなく、サブルーチン ?geqp3 を呼び出している。このサブルーチンは、ピボット演算を用いた QR 因子分解の BLAS-3 版である。
- マトリックス B (右辺) が BLAS-3 で更新される。
- マトリックス B (右辺) の置換が、より高速かつ単純になっている。

入力パラメータ

m	INTEGER。マトリックス A の行数 ($m \geq 0$)。
n	INTEGER。マトリックス A の列数 ($n \geq 0$)。
nrhs	INTEGER。右辺の個数、すなわち B の列数 ($nrhs \geq 0$)。
a, b, work	REAL (sgelsy の場合) DOUBLE PRECISION (dgelsy の場合) COMPLEX (cgelsy の場合) DOUBLE COMPLEX (zgelsy の場合) 配列 : *a(lda, *)* には、$m \times n$ のマトリックス A を格納する。 *a* の第 2 次元のサイズは max(1, *n*)以上でなければならない。

	b(ldb, *) には、m × nrhs の右辺のマトリックス B を格納する。 b の第 2 次元のサイズは max(1, nrhs) 以上でなければならない。
	work(lwork) は、ワークスペース配列。
lda	INTEGER。a の第 1 次元のサイズ。max(1, m) 以上。
ldb	INTEGER。b の第 1 次元のサイズ。max(1, m, n) 以上でなければならない。
jpvt	INTEGER。配列 DIMENSION は max(1, n) 以上。 呼び出し時に、jpvt(i) ≠ 0 の場合、A の i 番目の列が AP の先頭に置換される。それ以外の場合は、A の i 番目の列がフリー列になる。
rcond	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) rcond は、A の有効階数を決定するのに使用する。有効階数は、A のピボット演算を用いた QR 因子分解で得られた、先頭の最大三角部分マトリックス R_{11} の次数で定義される。その条件数の推定値 < 1/rcond である。
lwork	INTEGER。配列 work のサイズ。lwork の推奨値は、「アプリケーション・ノート」を参照。
rwork	REAL (cgelsy の場合) DOUBLE PRECISION (zgelsy の場合) ワークスペース配列 DIMENSION は max(1, 2n) 以上。複素数の場合にのみ使用される。

出力パラメータ

a	終了時に、A の完全直交因子分解の詳細で上書きされる。
b	n × nrhs の解マトリックス X で上書きされる。
jpvt	終了時に、jpvt(i) = k ならば、AP の i 番目の列が A の k 番目の列である。

rank INTEGER

A の有効階数、すなわち部分マトリックス R_{11} の次数。これは、A の完全直交因子分解における部分マトリックス T_{11} の次数と同じである。

info INTEGER

info = 0 の場合は、正常終了している。

info = -*i* の場合は、*i* 番目のパラメータ値が不正である。

アプリケーション・ノート

実数の場合 :

ブロック化アルゴリズムを使用しない場合は、次のワークスペースが必要である。

$$lwork \geq \max(\ mn+3n+1,\ 2*mn\ +\ nrhs\ )$$

ただし、$mn = \min(\ m,\ n\ )$ である。

ブロック化アルゴリズムを使用する場合は、次のようになる。

$$lwork \geq \max(\ mn+2n+nb*(n+1),\ 2*mn+nb*nrhs\ ),$$

nb は `ilaenv` がルーチン `sgeqp3/dgeqp3`、`stzrzf/dtzrzf`、`stzrqf/dtzrqf`、`sormqr/dormqr`、`sormrz/dormrz` に対して返したブロックサイズの上限値である。

複素数の場合 :

ブロック化アルゴリズムを使用しない場合は、次のワークスペースが必要である。

$$lwork \geq mn + \max(\ 2*mn,\ n+1,\ \ mn + nrhs\ )$$

ただし、$mn\ = \min(\ m,\ n\ )$ である。

ブロック化アルゴリズムを使用する場合は、次のようになる。

$$lwork \geq mn + \max(2*mn,\ nb*(n+1),\ mn+mn*nb,\ mn+nb*nrhs\ ),$$

nb は `ilaenv` がルーチン `cgeqp3/zgeqp3`、`ctzrzf/ztzrzf`、`ctzrqf/ztzrqf`、`cunmqr/zunmqr`、`cunmrz/zunmrz` に対して返したブロックサイズの上限値である。

?gelss

A の特異値分解を使用して、線形最小二乗問題の最小ノルム解を求める。

```
call sgelss ( m, n, nrhs, a, lda, b, ldb, s, rcond, rank, work, lwork,
              info )
call dgelss ( m, n, nrhs, a, lda, b, ldb, s, rcond, rank, work, lwork,
              info )
call cgelss ( m, n, nrhs, a, lda, b, ldb, s, rcond, rank, work, lwork,
              rwork, info )
call zgelss ( m, n, nrhs, a, lda, b, ldb, s, rcond, rank, work, lwork,
              rwork, info )
```

説明

このルーチンは、*A* の特異値分解 (SVD) を使用して、実数の線形最小二乗問題、すなわち

$$\text{minimize} \quad \| b - A x \|_2$$

の最小ノルム解を求める。*A* は *m* × *n* のマトリックスで、階数不足の可能性がある。
複数の右辺のベクトル *b* と解ベクトル *x* を、一度の呼び出しで処理できる。これらのベクトルは、*m* × *nrhs* の右辺のマトリックス *B* と *n* × *nrhs* の解マトリックス *X* の各列に格納される。
A の有効階数は、最大の特異値の *rcond* 倍より小さい特異値をゼロとして扱うことで決定される。

入力パラメータ

m	INTEGER。マトリックス *A* の行数 ($m \geq 0$)。
n	INTEGER。マトリックス *A* の列数 ($n \geq 0$)。
nrhs	INTEGER。右辺の個数、すなわち *B* の列数 ($nrhs \geq 0$)。

a, b, work
REAL (sgelss の場合)
DOUBLE PRECISION (dgelss の場合)
COMPLEX (cgelss の場合)
DOUBLE COMPLEX (zgelss の場合)
配列 :
*a(lda, *)* には、*m* × *n* のマトリックス *A* を格納する。
a の第 2 次元のサイズは max(1, *n*) 以上でなければならない。
*b(ldb, *)* には、*m* × *nrhs* の右辺のマトリックス *B* を格納する。
b の第 2 次元のサイズは max(1, *nrhs*) 以上でなければならない。
work(lwork) は、ワークスペース配列。

lda
INTEGER。*a* の第 1 次元のサイズ。max(1, *m*) 以上。

ldb
INTEGER。*b* の第 1 次元のサイズ。max(1, *m*, *n*) 以上でなければならない。

rcond
REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
rcond は、*A* の有効階数を決定するのに使用する。
特異値 *s*(*i*) ≤ *rcond* * *s*(1) をゼロとして扱う。
rcond < 0 の場合、かわりにマシン精度を使用する。

lwork
INTEGER。配列 *work* のサイズ、*lwork* ≥ 1。*lwork* の推奨値は、「アプリケーション・ノート」を参照。

rwork
REAL (cgelss の場合)
DOUBLE PRECISION (zgelss の場合)
複素数の場合にのみ使用するワークスペース配列。
DIMENSION は max(1, 5*min(*m*, *n*)) 以上。

出力パラメータ

a
終了時に、*A* の先頭から min(*m*, *n*) 個の各行に、右特異ベクトルが格納される。

b
n × *nrhs* の解マトリックス *X* で上書きされる。
m ≥ *n* で *rank* = *n* の場合、*i* 番目の列の解の二乗和の誤差は、その列の *n*+1:*m* の成分の二乗和で与えられる。

s	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列 DIMENSION は max(1, min(*m*, *n*)) 以上。*A* の特異値が降順で格納される。*A* の 2- ノルムの条件数は、 $k_2(A) = s(1) / s(\min(m, n))$
rank	INTEGER *A* の有効階数、すなわち、*rcond* * *s*(1) より大きい特異値の個数。
work(1)	終了時に、*info* = 0 ならば、最適パフォーマンスに必要な *lwork* の最小値が、*work*(1) に格納される。以降の実行では、この *lwork* 値を使用する。
info	INTEGER *info* = 0 の場合は、正常終了している。 *info* = -*i* の場合は、*i* 番目のパラメータ値が不正である。 *info* = *i* の場合は、SVD の計算アルゴリズムが収束に失敗した。*i* は、中間の二重対角形式の非対角成分で、ゼロに収束しなかった成分の個数を表す。

アプリケーション・ノート

実数の場合 :

$$lwork \geq 3*\min(m, n) + \max(2*\min(m, n), \max(m, n), nrhs)$$

複素数の場合 :

$$lwork \geq 2*\min(m, n) + \max(m, n, nrhs)$$

パフォーマンスを改善するには、一般に *lwork* に上記以上の値が必要である。必要なワークスペース・サイズが不明な場合は、*lwork* に大きな値を設定して実行してみる。ルーチン終了後に *work*(1) の値を調べ、以降はその値を使用する。

?gelsd

A の特異値分解と分割統治法を使用して、
線形最小二乗問題の最小ノルム解を求める。

```
call sgelsd ( m, n, nrhs, a, lda, b, ldb, s, rcond, rank, work, lwork,
              iwork, info )
call dgelsd ( m, n, nrhs, a, lda, b, ldb, s, rcond, rank, work, lwork,
              iwork, info )
call cgelsd ( m, n, nrhs, a, lda, b, ldb, s, rcond, rank, work, lwork,
              rwork, iwork, info )
call zgelsd ( m, n, nrhs, a, lda, b, ldb, s, rcond, rank, work, lwork,
              rwork, iwork, info )
```

説明

このルーチンは、*A* の特異値分解 (SVD) を使用して、実数の線形最小二乗問題、すなわち

$$\text{minimize} \quad \| b - A x \|_2$$

の最小ノルム解を求める。*A* は `m` × `n` のマトリックスで、階数不足の可能性がある。

複数の右辺のベクトル *b* と解ベクトル *x* を、一度の呼び出しで処理できる。これらのベクトルは、`m` × `nrhs` の右辺のマトリックス *B* と `n` × `nrhx` の解マトリックス *X* の各列に格納される。

この問題は、次の 3 ステップで解く。

1. Householder 変換により係数マトリックス A を二重対角形式に縮退し、元の問題を「二重対角形式の最小二乗問題 (BLS)」に縮退する。
2. 分割統治法を用いて、BLS を解く。
3. すべての Householder 変換を逆に適用して、元の最小二乗問題を解く。

A の有効階数は、最大特異値の `rcond` 倍より小さい特異値をゼロとして扱うことで決定される。

入力パラメータ

m	INTEGER。マトリックス *A* の行数 ($m \geq 0$)。
n	INTEGER。マトリックス *A* の列数 ($n \geq 0$)。
nrhs	INTEGER。右辺の個数、すなわち *B* の列数 ($nrhs \geq 0$)。
a, b, work	REAL (sgelsd の場合) DOUBLE PRECISION (dgelsd の場合) COMPLEX (cgelsd の場合) DOUBLE COMPLEX (zgelsd の場合) 配列 : *a(lda, *)* には、$m \times n$ のマトリックス *A* を格納する。 *a* の第 2 次元のサイズは max(1, *n*) 以上でなければならない。 *b(ldb, *)* には、$m \times nrhs$ の右辺のマトリックス *B* を格納する。 *b* の第 2 次元のサイズは max(1, *nrhs*) 以上でなければならない。 *work(lwork)* は、ワークスペース配列。
lda	INTEGER。*a* の第 1 次元のサイズ。max(1, *m*) 以上。
ldb	INTEGER。*b* の第 1 次元のサイズ。max(1, *m*, *n*) 以上でなければならない。
rcond	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) *rcond* は、*A* の有効階数を決定するのに使用する。 特異値 $s(i) \leq rcond * s(1)$ をゼロとして扱う。 $rcond < 0$ の場合、かわりにマシン精度を使用する。
lwork	INTEGER。配列 *work* のサイズ、$lwork \geq 1$。*lwork* の推奨値は、「アプリケーション・ノート」を参照。
iwork	INTEGER。ワークスペース配列。*iwork* の推奨値は、「アプリケーション・ノート」を参照。
rwork	REAL (cgelsd の場合) DOUBLE PRECISION (zgelsd の場合) 複素数の場合にのみ使用されるワークスペース配列。

rwork の推奨値は、「アプリケーション・ノート」を参照。

出力パラメータ

a 終了時に、*A* が上書きされる。

b *n* × *nrhs* の解マトリックス *X* で上書きされる。

m × *n* で *rank* = *n* の場合、*i* 番目の列の解の二乗和の誤差は、その列の *n*+1:*m* の成分の二乗和で与えられる。

s REAL (単精度の場合)

DOUBLE PRECISION (倍精度の場合)

配列 DIMENSION は max(1, min(*m*, *n*)) 以上。*A* の特異値が降順で格納される。*A* の 2- ノルムの条件数は、

$$k_2(A) = s(1) / s(\min(m, n))$$

rank INTEGER

A の有効階数、すなわち、*rcond* * *s*(1) より大きい特異値の個数。

work(1) 終了時に、*info* = 0 ならば、最適パフォーマンスに必要な *lwork* の最小値が、*work*(1) に格納される。以降の実行では、この *lwork* 値を使用する。

info INTEGER

info = 0 の場合は、正常終了している。

info = -*i* の場合は、*i* 番目のパラメータ値が不正である。

info = *i* の場合は、SVD の計算アルゴリズムが収束に失敗した。*i* は、中間の二重対角形式の非対角成分で、ゼロに収束しなかった成分の個数を表す。

アプリケーション・ノート

分割統治法は、浮動小数点演算において、非常に緩い仮定を行う。すなわち、加減算においてガード桁を持つマシンで機能する。ガード桁を持たない 16 進マシンや 10 進マシンでは異常終了するのも考えられるが、そのような前例はない。

必要なワークスペースの最小値は、*m*、*n*、*nrhs* によって決まる。ワークスペース配列 *work* のサイズ *lwork* は、次に示す値以上でなければならない。

実数の場合 :

m ≥ *n* の場合、
lwork ≥ $12n + 2n*smlsiz + 8n*nlvl + n*nrhs + (smlsiz+1)^2$

m < *n* の場合、
lwork ≥ $12m + 2m*smlsiz + 8m*nlvl + m*nrhs + (smlsiz+1)^2$

複素数の場合 :

m ≥ *n* の場合、
lwork ≥ $2n + n*nrhs$

m < *n* の場合、
lwork ≥ $2m + m*nrhs$

smlsiz は `ilaenv` に返された値で、計算ツリーの一番下の部分問題の最大サイズ (通常は約 25) で、$nlvl = \mathrm{INT}(\log_2(\min(m, n)/(smlsiz+1))) + 1$。

パフォーマンスを改善するには、一般に *lwork* に上記以上の値が必要である。必要なワークスペース・サイズが不明な場合は、*lwork* に大きな値を設定して実行してみる。ルーチン終了後に *work(1)* の値を調べ、以降はその値を使用する。

ワークスペース配列 *iwork* のサイズは、$3*\min(m, n)*nlvl + 11*\min(m, n)$ 以上でなければならない。

ワークスペース配列 *rwork* (複素数の場合にのみ使用) のサイズ *lrwork* は、次の値以上でなければならない。
m ≥ *n* の場合、
lrwork ≥ $10n + 2n*smlsiz + 8n*nlvl + 3*smlsiz*nrhs + (smlsiz+1)^2$

m < *n* の場合、
lrwork ≥ $10m + 2m*smlsiz + 8m*nlvl + 3*smlsiz*nrhs + (smlsiz+1)^2$

汎用 LLS 問題

この節では、汎用線形最小二乗問題を解くための LAPACK ドライバ・ルーチンについて説明する。この節で詳しく説明するドライバ・ルーチン名を、表 5-9 に示す。

表 5-9　汎用 LLS 問題を解くためのドライバ・ルーチン

ルーチン名	機能
?gglse	汎用 RQ 因子分解を使用して、均衡制約を持つ線形最小二乗問題を解く。
?ggglm	汎用 QR 因子分解を使用して、Gauss-Markov 線形モデル問題を解く。

?gglse

汎用 *RQ* 因子分解を使用して、均衡制約を持つ線形最小二乗問題を解く。

```
call sgglse ( m, n, p, a, lda, b, ldb, c, d, x, work, lwork, info )
call dgglse ( m, n, p, a, lda, b, ldb, c, d, x, work, lwork, info )
call cgglse ( m, n, p, a, lda, b, ldb, c, d, x, work, lwork, info )
call zgglse ( m, n, p, a, lda, b, ldb, c, d, x, work, lwork, info )
```

説明

このルーチンは、次のような均衡制約を持つ線形最小二乗 (LSE) 問題を解く。

$$\text{minimize} \quad \| c - A x \|_2 \quad \text{ただし、} B x = d$$

A は $m \times n$ のマトリックス、B は $p \times n$ のマトリックス、c は与えられた m- ベクトル、d は与えられた p- ベクトルである。
ただし、$p \le n \le m+p$、

$$\text{rank}(B) = p、\ \text{rank}\begin{pmatrix} A \\ B \end{pmatrix} = n \text{ と仮定する。}$$

これらの条件により、この LSE 問題は、マトリックス *B* と *A* の汎用 *RQ* 因子分解により、一意の解が得られる問題になる。

入力パラメータ

`m` INTEGER。マトリックス *A* の行数 ($m \geq 0$)。

`n` INTEGER。マトリックス *A* と *B* の列数 ($n \geq 0$)。

`p` INTEGER。マトリックス *B* の行数 ($0 \leq p \leq n \leq m+p$)。

`a, b, c, d, work`
REAL (`sgglse` の場合)
DOUBLE PRECISION (`dgglse` の場合)
COMPLEX (`cgglse` の場合)
DOUBLE COMPLEX (`zgglse` の場合)

配列 :

`a(lda, *)` には、$m \times n$ のマトリックス *A* を格納する。`a` の第 2 次元のサイズは max(1, `n`) 以上でなければならない。

`b(ldb, *)` には、$p \times n$ のマトリックス *B* を格納する。`b` の第 2 次元のサイズは max(1, `n`) 以上でなければならない。

`c(*)` は、サイズが max(1, `m`) 以上の配列で、LSE 問題の最小二乗部分に関する右辺のベクトルを格納する。

`d(*)` は、サイズが max(1, `p`) 以上の配列で、制約式の右辺のベクトルを格納する。

`work(lwork)` は、ワークスペース配列。

`lda` INTEGER。`a` の第 1 次元のサイズ。max(1, `m`) 以上。

`ldb` INTEGER。`b` の第 1 次元のサイズ。max(1, `p`) 以上。

`lwork` INTEGER。配列 `work` のサイズ。

`lwork` ≥ max(1, `m`+`n`+`p`)。`lwork` の推奨値は、「アプリケーション・ノート」を参照。

出力パラメータ

x	REAL (sgglse の場合) DOUBLE PRECISION (dgglse の場合) COMPLEX (cgglse の場合) DOUBLE COMPLEX (zgglse の場合) 配列 DIMENSION は max(1, *n*) 以上。 終了時に、LSE 問題の解が格納される。
a, *b*, *d*	終了時に、これらの配列が上書きされる。
c	終了時に、解の二乗和の誤差が、ベクトル *c* の *n*-*p*+1 ～ *m* の成分の二乗和で与えられる。
work(1)	終了時に、*info* = 0 ならば、最適パフォーマンスに必要な *lwork* の最小値が、*work*(1) に格納される。以降の実行では、この *lwork* 値を使用する。
info	INTEGER *info* = 0 の場合は、正常終了している。 *info* = -*i* の場合は、*i* 番目のパラメータ値が不正である。

アプリケーション・ノート

最適パフォーマンスを得るには、*lwork* に次の値を設定する。

$$lwork \geq p+\min(m, n)+\max(m, n)*nb$$

nb は、?geqrf、?gerqf、?ormqr/?unmqr、?ormrq/?unmrq に対する最適ブロックサイズの上限値である。

?ggglm

汎用 *QR* 因子分解を使用して、
Gauss-Markov 線形モデル問題を解く。

```
call sggglm ( n, m, p, a, lda, b, ldb, d, x, y, work, lwork, info )
call dggglm ( n, m, p, a, lda, b, ldb, d, x, y, work, lwork, info )
call cggglm ( n, m, p, a, lda, b, ldb, d, x, y, work, lwork, info )
call zggglm ( n, m, p, a, lda, b, ldb, d, x, y, work, lwork, info )
```

説明

このルーチンは、次のような一般 Gauss-Markov 線形モデル (GLM) 問題を解く。

$$\text{minimize}_x \ \| y \|_2 \quad \text{ただし、} d = Ax + By$$

A は $n \times m$ のマトリックス、B は $n \times p$ のマトリックス、d は与えられた n- ベクトルである。
ただし、$m \le n \le m+p$、

$\text{rank}(A) = \text{m}$、$\text{rank}(\ A\ B\ ) = n$ と仮定する。

これらの仮定により、制約式が常に成立し、A と B の汎用 *QR* 因子分解により、一意の解 x と最小 2- ノルム解 y が得られる。
特に、マトリックス B が正方非特異マトリックスの場合、この GLM 問題は、次のような重み付け線形最小二乗問題と等価になる。

$$\text{minimize}_x \ \| B^{-1}(d-Ax) \|_2$$

入力パラメータ

n	INTEGER。マトリックス A と B の行数 ($n \ge 0$)。
m	INTEGER。A の列数 ($m \ge 0$)。
p	INTEGER。B の列数 ($p \ge n - m$)。
a, b, d, work	REAL (sggglm の場合) DOUBLE PRECISION (dggglm の場合) COMPLEX (cggglm の場合) DOUBLE COMPLEX (zggglm の場合)

配列 :
a(lda, *) には、$n \times m$ のマトリックス A を格納する。
a の第 2 次元のサイズは max(1, m) 以上でなければならない。
b(ldb, *) には、$n \times p$ のマトリックス B を格納する。
b の第 2 次元のサイズは max(1, p) 以上でなければならない。
d(*) は、サイズが max(1, n) 以上の配列で、GLM 式の左辺を格納する。
work(lwork) は、ワークスペース配列。

lda　INTEGER。a の第 1 次元のサイズ。max(1, n) 以上。

ldb　INTEGER。b の第 1 次元のサイズ。max(1, n) 以上。

lwork　INTEGER。配列 work のサイズ。
lwork ≥ max(1, n+m+p)。lwork の推奨値は、「アプリケーション・ノート」を参照。

出力パラメータ

x, y　REAL (sggglm の場合)
DOUBLE PRECISION (dggglm の場合)
COMPLEX (cggglm の場合)
DOUBLE COMPLEX (zggglm の場合)
配列 x(*)、y(*)。x の DIMENSION は max(1, m) 以上、y のサイズは max(1, p) 以上。
終了時に、x と y が GLM 問題の解になる。

a, b, d　終了時に、これらの配列が上書きされる。

work(1)　終了時に、info = 0 ならば、最適パフォーマンスに必要な lwork の最小値が、work(1) に格納される。

info　INTEGER
info = 0 の場合は、正常終了している。
info = -i の場合は、i 番目のパラメータ値が不正である。

アプリケーション・ノート

最適パフォーマンスを得るには、lwork に次の値を設定する。

$$lwork \geq m+\min(n, p)+\max(n, p)*nb$$

nb は、?geqrf、?gerqf、?ormqr/?unmqr、?ormrq/?unmrq に対する最適ブロックサイズ上限値である。

対称固有値問題

この節では、対称固有値問題を解くための LAPACK ドライバ・ルーチンについて説明する。対称固有値問題を解くための計算ルーチンも参照のこと。
この節で詳しく説明するドライバ・ルーチン名を、表 5-10 に示す。

表 5-10 対称固有値問題を解くためのドライバ・ルーチン

ルーチン名	機能
?syev/?heev	実対称 / エルミート・マトリックスの固有値をすべて計算し、さらにオプションで、固有ベクトルも計算する。
?syevd/?heevd	分割統治アルゴリズムを使用して、実対称 / エルミート・マトリックスの固有値をすべて計算し、さらにオプションで、固有ベクトルも計算する。
?syevx/?heevx	対称 / エルミート・マトリックスの固有値を選択的に計算し、さらにオプションで、固有ベクトルも計算する。
?syevr/?heevr	「比較的安定な表現」を使用して、実対称 / エルミート・マトリックスの固有値を選択的に計算し、さらにオプションで、固有ベクトルも計算する。
?spev/?hpev	パックド形式の実対称 / エルミート・マトリックスの固有値をすべて計算し、さらにオプションで、固有ベクトルも計算する。
?spevd/?hpevd	分割統治アルゴリズムを使用して、パックド形式の実対称 / エルミート・マトリックスの固有値をすべて計算し、さらにオプションで、固有ベクトルも計算する。
?spevx/?hpevx	パックド形式の実対称 / エルミート・マトリックスの固有値を選択的に計算し、さらにオプションで、固有ベクトルも計算する。
?sbev/?hbev	バンド形式の実対称 / エルミート・マトリックスの固有値をすべて計算し、さらにオプションで、固有ベクトルも計算する。
?sbevd/?hbevd	分割統治アルゴリズムを使用して、バンド形式の実対称 / エルミート・マトリックスの固有値をすべて計算し、さらにオプションで固有ベクトルも計算する。
?sbevx/?hbevx	バンド形式の実対称 / エルミート・マトリックスの固有値を選択的に計算し、さらにオプションで、固有ベクトルも計算する。
?stev	実対称三重対角マトリックスの固有値をすべて計算し、さらにオプションで、固有ベクトルも計算する。
?stevd	分割統治アルゴリズムを使用して、実対称三重対角マトリックスの固有値をすべて計算し、さらにオプションで、固有ベクトルも計算する。
?stevx	実対称三重対角マトリックスの固有値と固有ベクトルを選択的に計算する。
?stevr	「比較的安定な表現」を使用して、実対称三重対角マトリックスの固有値を選択的に計算し、さらにオプションで、固有ベクトルも計算する。

?syev

実対称マトリックスの固有値をすべて計算し、さらにオプションで、固有ベクトルも計算する。

```
call ssyev ( jobz, uplo, n, a, lda, w, work, lwork, info )
call dsyev ( jobz, uplo, n, a, lda, w, work, lwork, info )
```

説明

このルーチンは、実対称マトリックス A の固有値をすべて計算し、さらにオプションで、固有ベクトルも計算する。

入力パラメータ

jobz
CHARACTER*1。'N' または 'V' でなければならない。
jobz ='N' の場合、固有値のみ計算する。
jobz ='V' の場合、固有値と固有ベクトルを計算する。

uplo
CHARACTER*1。'U' または 'L' でなければならない。
uplo ='U' の場合、*a* に、A の上三角部分を格納する。
uplo ='L' の場合、*a* に、A の下三角部分を格納する。

n
INTEGER。マトリックス A の次数 ($n \geq 0$)。

a, work
REAL (ssyev の場合)
DOUBLE PRECISION (dsyev の場合)
配列 :
*a(lda, *)* は、対称マトリックス A の上または下 (*uplo* の指定に従う) 三角部分を格納する配列。
a の第 2 次元のサイズは max(1, *n*)以上でなければならない。
work(lwork) は、ワークスペース配列。

lda
INTEGER。配列 *a* の第 1 次元のサイズ。
max(1, *n*) 以上でなければならない。

lwork
INTEGER。配列 *work* のサイズ。
制約 : *lwork* ≥ max(1, 3*n*-1)。*lwork* の推奨値は、「アプリケーション・ノート」を参照。

出力パラメータ

a	*jobz* = 'V' の場合、終了時に、*info* = 0 ならば、マトリックス *A* の正規直交固有ベクトルが、配列 *a* に格納される。 *jobz* = 'N' の場合、終了時に、マトリックス *A* の下三角部分 (*uplo* = 'L' の場合) または上三角部分 (*uplo* = 'U' の場合) が、対角成分を含めて上書きされる。
w	REAL (ssyev の場合) DOUBLE PRECISION (dsyev の場合) 配列 DIMENSION は max(1, *n*) 以上。 *info* = 0 の場合、マトリックス *A* の固有値が昇順に格納される。
work(1)	終了時に、*lwork* > 0 ならば、*lwork* に必要な最小値が *work*(1) に返される。
info	INTEGER *info* = 0 の場合は、正常終了している。 *info* = -*i* の場合は、*i* 番目のパラメータ値が不正である。 *info* = *i* の場合は、このアルゴリズムが収束に失敗した。*i* は、中間の三重対角形式で、ゼロに収束しなかった成分の個数を表す。

アプリケーション・ノート

最適パフォーマンスを得るには、*lwork* に次の値を設定する。

$$lwork \geq (nb+2)*n$$

nb は、ilaenv が ?sytrd に対して返したブロックサイズである。
必要なワークスペース・サイズが不明な場合は、*lwork* に大きな値を設定して実行してみる。ルーチン終了後に *work*(1) の値を調べ、以降はその値を使用する。

?heev

エルミート・マトリックスの固有値を
すべて計算し、さらにオプションで、
固有ベクトルも計算する。

```
call cheev ( jobz, uplo, n, a, lda, w, work, lwork, rwork, info )
call zheev ( jobz, uplo, n, a, lda, w, work, lwork, rwork, info )
```

説明

このルーチンは、エルミート・マトリックス A の固有値をすべて計算し、さらにオプションで、固有ベクトルも計算する。

入力パラメータ

jobz
: CHARACTER*1。'N' または 'V' でなければならない。
jobz ='N' の場合、固有値のみ計算する。
jobz ='V' の場合、固有値と固有ベクトルを計算する。

uplo
: CHARACTER*1。'U' または 'L' でなければならない。
uplo ='U' の場合、*a* に、A の上三角部分を格納する。
uplo ='L' の場合、*a* に、A の下三角部分を格納する。

n
: INTEGER。マトリックス A の次数 ($n \geq 0$)。

a, work
: COMPLEX (cheev の場合)
DOUBLE COMPLEX (zheev の場合)
配列 :
*a(lda, *)* は、エルミート・マトリックス A の上または下 (*uplo* の指定に従う) 三角部分を格納する配列。*a* の第 2 次元のサイズは max(1, *n*)以上でなければならない。
work(lwork) は、ワークスペース配列。

lda
: INTEGER。配列 *a* の第 1 次元のサイズ。
max(1, *n*) 以上でなければならない。

lwork INTEGER。配列 *work* のサイズ。
制約 : *lwork* ≥ max(1, 2*n*-1)。*lwork* の推奨値は、「アプリケーション・ノート」を参照。

rwork REAL (cheev の場合)
DOUBLE PRECISION (zheev の場合)
ワークスペース配列 DIMENSION は max(1, 3*n*-2) 以上。

出力パラメータ

a *jobz* = 'V' の場合、終了時に、*info* = 0 ならば、マトリックス *A* の正規直交固有ベクトルが、配列 *a* に格納される。
jobz = 'N' の場合、終了時に、マトリックス *A* の下三角部分 (*uplo* = 'L' の場合) または上三角部分 (*uplo* = 'U' の場合) が、対角成分を含めて上書きされる。

w REAL (*cheev* の場合)
DOUBLE PRECISION (zheev の場合)
配列 DIMENSION は max(1, *n*) 以上。
info = 0 の場合、マトリックス *A* の固有値が昇順に格納される。

work(1) 終了時に、*lwork* > 0 ならば、*lwork* に必要な最小値が *work*(1) に返される。

info INTEGER
info = 0 の場合は、正常終了している。
info = -*i* の場合は、*i* 番目のパラメータ値が不正である。
info = *i* の場合は、このアルゴリズムが収束に失敗した。*i* は、中間の三重対角形式で、ゼロに収束しなかった成分の個数を表す。

アプリケーション・ノート

最適パフォーマンスを得るには、*lwork* に次の値を設定する。

$$lwork \geq (nb+1)*n$$

nb は、ilaenv が ?hetrd に対して返したブロックサイズである。
必要なワークスペース・サイズが不明な場合は、*lwork* に大きな値を設定して実行してみる。ルーチン終了後に *work*(1) の値を調べ、以降はその値を使用する。

?syevd

分割統治アルゴリズムを使用して、
実対称マトリックスの固有値をすべて
計算し、さらにオプションで、固有
ベクトルも計算する。

```
call ssyevd (job, uplo, n, a, lda, w, work, lwork, iwork, liwork, info)
call dsyevd (job, uplo, n, a, lda, w, work, lwork, iwork, liwork, info)
```

説明

このルーチンは、分割統治アルゴリズムを使用して、実対称マトリックス A の固有値をすべて計算し、さらにオプションで、固有ベクトルも計算する。言い換えると、A のスペクトル因子分解 $A = Z\Lambda Z^T$ が求められる。
Λ は対角成分が固有値 λ_i であるような対角マトリックス、Z は各列が固有ベクトル z_i であるような直交マトリックスである。したがって、次のように表現できる。

$$Az_i = \lambda_i z_i \quad (\text{ただし、} i = 1, 2, ..., n)$$

このルーチンは、固有ベクトルも要求された場合は、分割統治アルゴリズムを使用して、固有値と固有ベクトルを計算する。ただし、固有値のみ要求された場合は、Pal-Walker-Kahan 変形版の *QL* または *QR* アルゴリズムを使用して、固有値を計算する。

入力パラメータ

job CHARACTER*1。'N' または 'V' でなければならない。
job ='N' の場合、固有値のみ計算する。
job ='V' の場合、固有値と固有ベクトルを計算する。

uplo CHARACTER*1。'U' または 'L' でなければならない。
uplo ='U' の場合、*a* に、A の上三角部分を格納する。
uplo ='L' の場合、*a* に、A の下三角部分を格納する。

n INTEGER。マトリックス A の次数 ($n \geq 0$)。

a REAL (ssyevd の場合)
DOUBLE PRECISION (dsyevd の場合)
配列 DIMENSION は (*lda*, *)

a(lda, *) は、対称マトリックス A の上または下 (uplo の指定に従う) 三角部分を格納する配列。
a の第2次元のサイズは max(1, n)以上でなければならない。

lda
INTEGER。配列 a の第 1 次元のサイズ。
max(1, n) 以上でなければならない。

work
REAL (ssyevd の場合)
DOUBLE PRECISION (dsyevd の場合)
ワークスペース配列 DIMENSION は lwork 以上。

lwork
INTEGER。配列 work のサイズ。
次の制約がある。
$n \le 1$ の場合、lwork ≥ 1
job='N' で $n > 1$ の場合、lwork ≥ $2n+1$
job='V' で $n > 1$ の場合、lwork ≥ $3n^2+(5+2k)*n+1$ (ただし、k は、$2^k \ge n$ を満足する最小の整数)

iwork
INTEGER
ワークスペース配列 DIMENSION は liwork 以上。

liwork
INTEGER。配列 iwork のサイズ。
次の制約がある。
$n \le 1$ の場合、liwork ≥ 1
job='N' で $n > 1$ の場合、liwork ≥ 1
job='V' で $n > 1$ の場合、liwork ≥ $5n+2$

出力パラメータ

w
REAL (ssyevd の場合)
DOUBLE PRECISION (dsyevd の場合)
配列 DIMENSION は max(1, n) 以上。
info = 0 の場合、マトリックス A の固有値が昇順に格納される。
info も参照のこと。

a
job='V' の場合、終了時に、この配列は、A の固有ベクトルが入った直交マトリックス Z で上書きされる。

work(1)	終了時に、*lwork* > 0 ならば、*lwork* に必要な最小値が *work*(1) に返される。
iwork(1)	終了時に、*liwork* > 0 ならば、*liwork* に必要な最小値が *iwork*(1) に返される。
info	INTEGER *info* = 0 の場合は、正常終了している。 *info* = *i* の場合は、このアルゴリズムが収束に失敗した。*i* は、中間の三重対角形式で、ゼロに収束しなかった成分の個数を表す。 *info* = -*i* の場合は、*i* 番目のパラメータ値が不正である。

アプリケーション・ノート

計算された固有値と固有ベクトルは、正確には、$\|E\|_2 = O(\varepsilon)\,\|T\|_2$ であるようなマトリックス $T + E$ (ただし、ε はマシン精度) のものである。

このルーチンの複素数版は ?heevd である。

?heevd

分割統治アルゴリズムを使用して、
複素エルミート・マトリックスの
固有値をすべて計算し、さらに
オプションで、固有ベクトルも計算する。

```
call cheevd (job, uplo, n, a, lda, w, work, lwork, rwork, lrwork,
             iwork, liwork, info)
call zheevd (job, uplo, n, a, lda, w, work, lwork, rwork, lrwork,
             iwork, liwork, info)
```

説明

このルーチンは、分割統治アルゴリズムを使用して、複素エルミート・マトリックス A の固有値をすべて計算し、さらにオプションで、固有ベクトルも計算する。言い換えると、A のスペクトル因子分解 $A = Z\Lambda Z^H$ が求められる。
Λ は対角成分が固有値 λ_i であるような実対角マトリックス、Z は各列が固有ベクトル z_i であるような (複素) ユニタリ・マトリックスである。したがって、次のように表現できる。

$$Az_i = \lambda_i z_i \text{ (ただし、 } i = 1, 2, ..., n)$$

このルーチンは、固有ベクトルも要求された場合は、分割統治アルゴリズムを使用して、固有値と固有ベクトルを計算する。ただし、固有値のみ要求された場合は、Pal-Walker-Kahan 変形版の QL または QR アルゴリズムを使用して、固有値を計算する。

入力パラメータ

job	CHARACTER*1。'N' または 'V' でなければならない。 *job* ='N' の場合、固有値のみ計算する。 *job* ='V' の場合、固有値と固有ベクトルを計算する。
uplo	CHARACTER*1。'U' または 'L' でなければならない。 *uplo* ='U' の場合、*a* に、A の上三角部分を格納する。 *uplo* ='L' の場合、*a* に、A の下三角部分を格納する。
n	INTEGER。マトリックス A の次数 ($n \geq 0$)。

a	COMPLEX (cheevd の場合) DOUBLE COMPLEX (zheevd の場合) 配列 DIMENSION は (*lda*, *) *a*(*lda*, *) は、エルミート・マトリックス *A* の上または下 (*uplo* の指定に従う) 三角部分を格納する配列。 *a* の第 2 次元のサイズは max(1, *n*)以上でなければならない。
lda	INTEGER。配列 *a* の第 1 次元のサイズ。 max(1, *n*) 以上でなければならない。
work	COMPLEX (cheevd の場合) DOUBLE COMPLEX (zheevd の場合) ワークスペース配列 DIMENSION は *lwork* 以上。
lwork	INTEGER。配列 *work* のサイズ。 次の制約がある。 $n \le 1$ の場合、*lwork* ≥ 1 *job* ='N' で $n > 1$ の場合、*lwork* $\ge n+1$ *job* ='V' で $n > 1$ の場合、*lwork* $\ge n^2+2n$
rwork	REAL (cheevd の場合) DOUBLE PRECISION (zheevd の場合) ワークスペース配列 DIMENSION は *lrwork* 以上。
lrwork	INTEGER。配列 *rwork* のサイズ。 次の制約がある。 $n \le 1$ の場合、*lrwork* ≥ 1 *job* ='N' で $n > 1$ の場合、*lrwork* $\ge n$ *job* ='V' で $n > 1$ の場合、*lrwork* $\ge 3n^2+(4+2k)*n+1$ (ただし、k は、$2^k \ge n$ を満足する最小の整数)
iwork	INTEGER ワークスペース配列 DIMENSION は *liwork* 以上。
liwork	INTEGER。配列 *iwork* のサイズ。 次の制約がある。 $n \le 1$ の場合、*liwork* ≥ 1 *job* ='N' で $n > 1$ の場合、*liwork* ≥ 1 *job* ='V' で $n > 1$ の場合、*liwork* $\ge 5n+2$

出力パラメータ

w	REAL (cheevd の場合) DOUBLE PRECISION (zheevd の場合) 配列 DIMENSION は max(1, *n*) 以上。 *info* = 0 の場合、マトリックス *A* の固有値が昇順に格納される。 *info* も参照のこと。
a	*job* = 'V' の場合、終了時に、この配列は、*A* の固有ベクトルが入ったユニタリ・マトリックス *Z* で上書きされる。
work(1)	終了時に、*lwork* > 0 ならば、*lwork* に必要な最小値が *work(1)* の実部に返される。
rwork(1)	終了時に、*lrwork* > 0 ならば、*lrwork* に必要な最小値が *rwork(1)* に返される。
iwork(1)	終了時に、*liwork* > 0, ならば、*liwork* に必要な最小値が *iwork(1)* に返される。
info	INTEGER *info* = 0 の場合は、正常終了している。 *info* = *i* の場合は、このアルゴリズムが収束に失敗した。*i* は、中間の三重対角形式で、ゼロに収束しなかった成分の個数を表す。 *info* = -*i* の場合は、*i* 番目のパラメータ値が不正である。

アプリケーション・ノート

計算された固有値と固有ベクトルは、正確には、$\|E\|_2 = O(\varepsilon)\,\|A\|_2$ であるようなマトリックス $A + E$ (ただし、ε はマシン精度) のものである。

このルーチンの実数版は、?syevd である。
パックド形式のマトリックスに対する ?hpevd と、バンド形式のマトリックスに関する ?hbevd も参照のこと。

?syevx

対称マトリックスの固有値を選択的に計算し、さらにオプションで、固有ベクトルも計算する。

```
call ssyevx (jobz, range, uplo, n, a, lda, vl, vu, il, iu, abstol, m,
             w, z, ldz, work, lwork, iwork, ifail, info)
call dsyevx (jobz, range, uplo, n, a, lda, vl, vu, il, iu, abstol, m,
             w, z, ldz, work, lwork, iwork, ifail, info)
```

説明

このルーチンは、実対称マトリックス A の固有値を選択的に計算し、さらにオプションで、固有ベクトルも計算する。計算する固有値と固有ベクトルは、目的の固有値の範囲、または固有値のインデックスの範囲で選択できる。

入力パラメータ

jobz CHARACTER*1。'N' または 'V' でなければならない。
jobz ='N' の場合、固有値のみ計算する。
jobz ='V' の場合、固有値と固有ベクトルを計算する。

range CHARACTER*1。'A'、'V'、'I' のどれかでなければならない。
range ='A' の場合、固有値をすべて計算する。
range ='V' の場合、半開区間 (*vl*, *vu*] に含まれる固有値をすべて計算する。
range ='I' の場合、インデックスが *il* ～ *iu* の固有値を計算する。

uplo CHARACTER*1。'U' または 'L' でなければならない。
uplo ='U' の場合、*a* に、A の上三角部分を格納する。
uplo ='L' の場合、*a* に、A の下三角部分を格納する。

n INTEGER。マトリックス A の次数 ($n \geq 0$)。

a, work	REAL (ssyevx の場合) DOUBLE PRECISION (dsyevx の場合) 配列 : *a(lda, *)* は、対称マトリックス *A* の上または下 (*uplo* の指定に従う) 三角部分を格納する配列。 *a* の第 2 次元のサイズは max(1, *n*)以上でなければならない。 *work(lwork)* は、ワークスペース配列。
lda	INTEGER。配列 *a* の第 1 次元のサイズ。 max(1, *n*) 以上でなければならない。
vl, vu	REAL (ssyevx の場合) DOUBLE PRECISION (dsyevx の場合) *range* = 'V' の場合、固有値を探す区間の下限値と上限値、*vl* ≤ *vu* *range* = 'A' または 'I' の場合は、参照されない。
il, iu	INTEGER。*range* = 'I' の場合、求める固有値の最小インデックスと最大インデックス。 制約 : *n* > 0 の場合は、1 ≤ *il* ≤ *iu* ≤ *n*。*n* = 0 の場合は、*il* = 1 かつ *iu* = 0 *range* = 'A' または 'V' の場合は、参照されない。
abstol	REAL (ssyevx の場合) DOUBLE PRECISION (dsyevx の場合) 固有値の絶対誤差許容値。 詳細については、「アプリケーション・ノート」を参照。
ldz	INTEGER。出力配列 *z* の第 1 次元のサイズ。 *ldz* ≥ 1。*jobz* = 'V' の場合は、*ldz* ≥ max(1, *n*)
lwork	INTEGER。配列 *work* のサイズ。 制約 : *lwork* ≥ max(1, 8*n*)。*lwork* の推奨値は、「アプリケーション・ノート」を参照。
iwork	INTEGER。ワークスペース配列 DIMENSION は max(1, 5*n*) 以上。

出力パラメータ

a
終了時に、マトリックス *A* の下三角部分 (*uplo* = 'L' の場合) または上三角部分 (*uplo* = 'U' の場合) が、対角成分を含めて上書きされる。

m
INTEGER。見つかった固有値の総数。
$0 \le m \le n$。*range* = 'A' の場合は *m* = *n*、*range* = 'I' の場合は *m* = *iu*-*il*+*1*

w
REAL (ssyevx の場合)
DOUBLE PRECISION (dsyevx の場合)
配列 DIMENSION は max(1, *n*) 以上。
先頭の *m* 個の成分に、マトリックス *A* の選択された固有値が昇順に格納される。

z
REAL (ssyevx の場合)
DOUBLE PRECISION (dsyevx の場合)
配列 *z(ldz, *)* には、固有ベクトルが格納される。*z* の第 2 次元のサイズは max(1, *m*) 以上でなければならない。

jobz = 'V' の場合、*info* = 0 ならば、*z* の先頭 *m* 列に、マトリックス *A* の選択された固有値に対応する正規直交固有ベクトルが格納される。すなわち、*w*(i) に対応する固有ベクトルが、*z* の i 番目の列に格納される。収束に失敗した固有ベクトルは、それに対応する *z* の列に、最終的な近似固有ベクトルが格納され、この固有ベクトルのインデックスが *ifail* に返される。
jobz = 'N' の場合、*z* は参照されない。
注 : 配列 *z* には、max(1, *m*) 以上の列が提供されなければならない。*range* = 'V' の場合は、*m* の正確な値が事前にわからないため、上限値を使用する必要がある。

work(1)
終了時に、*lwork* > 0 ならば、*lwork* に必要な最小値が *work(1)* に返される。

ifail
INTEGER。配列 DIMENSION は max(1, *n*) 以上。
jobz = 'V' の場合、*info* = 0 ならば、*ifail* の先頭 *m* 成分はゼロになる。*info* > 0 ならば、収束に失敗した固有ベクトルのインデックスが *ifail* に格納される。
jobz = 'V' の場合、*ifail* は参照されない。

info INTEGER

info = 0 の場合は、正常終了している。

info = -*i* の場合は、*i* 番目のパラメータ値が不正である。

info = *i* の場合、*i* 番目の固有ベクトルが収束に失敗した。その固有ベクトルのインデックスが配列 *ifail* に格納される。

アプリケーション・ノート

最適パフォーマンスを得るには、*lwork* に次の値を設定する。

$$lwork \geq (nb+3)*n$$

nb は、ilaenv が ?sytrd と ?ormtr に対して返したブロックサイズである。必要なワークスペース・サイズが不明な場合は、*lwork* に大きな値を設定して実行してみる。ルーチン終了後に *work*(1) の値を調べ、以降はその値を使用する。

近似的な固有値でも、幅が *abstol*+ε*max(|a|, |b|) 以下の区間 [a, b] 内に納まる場合は、収束したと見なされる (ε はマシン精度)。*abstol* がゼロまたは負の場合は、かわりに ε*|*T*| を使用する。|*T*| は、*A* を三重対角形式に縮退して得られた三重対角マトリックスの 1- ノルムである。

固有値が最も正確に計算されるのは、*abstol* がゼロでなく、アンダーフローのしきい値の 2 倍、2**slamch*('S') に設定されたときである。このルーチンで *info* > 0 が返された場合は、固有ベクトルのいくつかが収束に失敗したことを意味するので、*abstol* を 2**slamch*('S') に設定して、やり直してみる。

?heevx

エルミート・マトリックスの固有値を
選択的に計算し、さらにオプションで、
固有ベクトルも計算する。

```
call cheevx (jobz, range, uplo, n, a, lda, vl, vu, il, iu, abstol, m,
             w, z, ldz, work, lwork, rwork, iwork, ifail, info)
call zheevx (jobz, range, uplo, n, a, lda, vl, vu, il, iu, abstol, m,
             w, z, ldz, work, lwork, rwork, iwork, ifail, info)
```

説明

このルーチンは、複素エルミート・マトリックス A の固有値を選択的に計算し、さらにオプションで、固有ベクトルも計算する。計算する固有値と固有ベクトルは、目的の固有値の範囲、または固有値のインデックスの範囲で選択できる。

入力パラメータ

jobz CHARACTER*1。'N' または 'V' でなければならない。
jobz ='N' の場合、固有値のみ計算する。
jobz ='V' の場合、固有値と固有ベクトルを計算する。

range CHARACTER*1。'A'、'V'、'I' のどれかでなければならない。
range ='A' の場合、固有値をすべて計算する。
range ='V' の場合、半開区間 (*vl*, *vu*) に含まれる固有値を計算する。
range ='I' の場合、インデックスが *il* ～ *iu* の固有値を計算する。

uplo CHARACTER*1。'U' または 'L' でなければならない。
uplo ='U' の場合、*a* に、A の上三角部分を格納する。
uplo ='L' の場合、*a* に、A の下三角部分を格納する。

n INTEGER。マトリックス A の次数 ($n \geq 0$)。

a, work	COMPLEX (cheevx の場合) DOUBLE COMPLEX (zheevx の場合) 配列 : *a(lda, *)* は、エルミート・マトリックス *A* の上または下 (*uplo* の指定に従う) 三角部分を格納する配列。*a* の第 2 次元のサイズは max(1, *n*)以上でなければならない。 *work(lwork)* は、ワークスペース配列。
lda	INTEGER。配列 *a* の第 1 次元のサイズ。 max(1, *n*) 以上でなければならない。
vl, vu	REAL (cheevx の場合) DOUBLE PRECISION (zheevx の場合) *range* = 'V' の場合、固有値を探す区間の下限値と上限値、*vl* ≤ *vu* *range* = 'A' または 'I' の場合は、参照されない。
il, iu	INTEGER。*range* = 'I' の場合、求める固有値の最小インデックスと最大インデックス。 制約 : *n* > 0 の場合は、1 ≤ *il* ≤ *iu* ≤ *n*。*n* = 0 の場合は、*il* = 1 かつ *iu* = 0 *range* = 'A' または 'V' の場合は、参照されない。
abstol	REAL (cheevx の場合) DOUBLE PRECISION (zheevx の場合) 固有値の絶対誤差許容値。 詳細については、「アプリケーション・ノート」を参照。
ldz	INTEGER。出力配列 *z* の第 1 次元のサイズ。 *ldz* ≥ 1。*jobz* = 'V' の場合は、*ldz* ≥ max(1, *n*)
lwork	INTEGER。配列 *work* のサイズ。 制約 : *lwork* ≥ ma(1, 2*n*-1)。*lwork* の推奨値は、「アプリケーション・ノート」を参照。
rwork	REAL (cheevx の場合) DOUBLE PRECISION (zheevx の場合) ワークスペース配列 DIMENSION は max(1, 7n) 以上。
iwork	INTEGER。ワークスペース配列 DIMENSION は max(1, 5*n*) 以上。

出力パラメータ

a
終了時に、マトリックス *A* の下三角部分 (*uplo* = 'L' の場合) または上三角部分 (*uplo* = 'U' の場合) が、対角成分を含めて上書きされる。

m
INTEGER。見つかった固有値の総数。
$0 \le m \le n$。*range* = 'A' の場合は *m* = *n*、*range* = 'I' の場合は *m* = *iu*-*il*+*1*

w
REAL (cheevx の場合)
DOUBLE PRECISION (zheevx の場合)
配列 DIMENSION は max(1, *n*) 以上。
先頭の *m* 個の成分に、マトリックス *A* の選択された固有値が昇順に格納される。

z
COMPLEX (cheevx の場合)
DOUBLE COMPLEX (zheevx の場合)
配列 *z(ldz, *)* には、固有ベクトルが格納される。
z の第 2 次元のサイズは max(1, *m*) 以上でなければならない。

jobz = 'V' の場合、*info* = 0 ならば、*z* の先頭 *m* 列に、マトリックス *A* の選択された固有値に対応する正規直交固有ベクトルが格納される。すなわち、*w*(i) に対応する固有ベクトルが、*z* の i 番目の列に格納される。収束に失敗した固有ベクトルは、それに対応する *z* の列に、最終的な近似固有ベクトルが格納され、この固有ベクトルのインデックスが *ifail* に返される。
jobz = 'N' の場合、*z* は参照されない。
注 : 配列 *z* には、max(1, *m*) 以上の列が提供されなければならない。*range* = 'V' の場合は、*m* の正確な値が事前にわからないため、上限値を使用する必要がある。

work(1)
終了時に、*lwork* > 0 ならば、*lwork* に必要な最小値が *work(1)* に返される。

ifail
INTEGER。配列 DIMENSION は max(1, *n*) 以上。
jobz = 'V' の場合、*info* = 0 ならば、*ifail* の先頭 *m* 成分はゼロになる。*info* > 0 ならば、収束に失敗した固有ベクトルのインデックスが *ifail* に格納される。
jobz = 'V' の場合、*ifail* は参照されない。

info INTEGER

info = 0 の場合は、正常終了している。

info = -*i* の場合は、*i* 番目のパラメータ値が不正である。

info = *i* の場合、*i* 番目の固有ベクトルが収束に失敗した。その固有ベクトルのインデックスが配列 *ifail* に格納される。

アプリケーション・ノート

最適パフォーマンスを得るには、*lwork* ≥ (*nb*+1)*n に設定する。*nb* は、ilaenv が ?hetrd と ?unmtr に対して返したブロックサイズである。必要なワークスペース・サイズが不明な場合は、*lwork* に大きな値を設定して実行してみる。ルーチン終了後に *work*(1) の値を調べ、以降はその値を使用する。

近似的な固有値でも、幅が *abstol*+ε*max(|a|, |b|) 以下の区間 [a, b] 内に納まる場合は、収束したと見なされる (ε はマシン精度)。*abstol* がゼロまたは負の場合は、かわりに ε*|*T*| を使用する。|*T*| は、*A* を三重対角形式に縮退して得られた三重対角マトリックスの 1- ノルムである。

固有値が最も正確に計算されるのは、*abstol* がゼロでなく、アンダーフローのしきい値の 2 倍、2*slamch('S') に設定されたときである。このルーチンで *info* > 0 が返された場合は、固有ベクトルのいくつかが収束に失敗したことを意味するので、*abstol* を 2*slamch('S') に設定して、やり直してみる。

?syevr

「比較的安定な表現」を使用して、
実対称マトリックスの固有値を選択的に
計算し、さらにオプションで、固有
ベクトルも計算する。

```
call ssyevr (jobz, range, uplo, n, a, lda, vl, vu, il, iu, abstol, m,
             w, z, ldz, isuppz, work, lwork, iwork, liwork, info)
call dsyevr (jobz, range, uplo, n, a, lda, vl, vu, il, iu, abstol, m,
             w, z, ldz, isuppz, work, lwork, iwork, liwork, info)
```

説明

このルーチンは、実対称マトリックス T の固有値を選択的に計算し、さらにオプションで、固有ベクトルも計算する。計算する固有値と固有ベクトルは、目的の固有値の範囲、または固有値のインデックスの範囲で選択できる。

可能であれば、?syevr は sstegr/dstegr を呼び出し、「比較的安定な表現」を使用して、固有スペクトルを計算する。?stegr は、固有値の計算には *dqds* アルゴリズムを使用するが、直交固有ベクトルは、種々の「良好な」LDL^T 表現 (「比較的安定な表現」とも呼ばれる) から計算する。グラム - シュミットの直交化はできる限り避ける。アルゴリズムの各ステップを詳しく説明すると次のようになる。T のまだ縮退されていない i 番目のブロックに対して、

(a) $L_i D_i L_i^T$ が比較的安定な表現になるように、$T - \sigma_i = L_i D_i L_i^T$ を計算する。

(b) *dqds* アルゴリズムによって、$L_i D_i L_i^T$ の固有値 λ_j を高い相対精度で計算する。

(c) 値が近い固有値からなるクラスタが存在する場合は、そのクラスタに近い σ_i を「選択」し、ステップ (a) に戻る。

(d) $L_i D_i L_i^T$ の近似的な固有値 λ_j に対する固有ベクトルを、階数表示のツイスト分解によって求める。

出力値に必要な精度は、入力パラメータ *abstol* で指定できる。

ルーチン ?syevr は、IEEE-754 の浮動小数点標準に準拠しているマシンでフル・スペクトルが要求された場合は、sstegr/dstegr を呼び出す。IEEE-754 に準拠しないマシンの場合や、部分スペクトルが要求された場合は、sstebz/dstebz と sstein/dstein を呼び出す。

入力パラメータ

jobz
CHARACTER*1。'N' または 'V' でなければならない。
jobz ='N' の場合、固有値のみ計算する。
jobz ='V' の場合、固有値と固有ベクトルを計算する。

range
CHARACTER*1。'A'、'V'、'I' のどれかでなければならない。
range ='A' の場合、固有値をすべて計算する。
range ='V' の場合、半開区間に含まれる固有値 λi ($vl < \lambda_i \le vu$) を計算する。
range ='I' の場合、インデックスが *il* ～ *iu* の固有値を計算する。
range ='V' または 'I' で、*iu*-*il* < *n*-1 の場合、sstebz/dstebz と sstein/dstein を呼び出す。

n
INTEGER。マトリックス *A* の次数 ($n \ge 0$)。

a, work
REAL (ssyevr の場合)
DOUBLE PRECISION (dsyevr の場合)
配列 :
*a(lda, *)* は、対称マトリックス *A* の上または下 (*uplo* の指定に従う) 三角部分を格納する配列。
a の第 2 次元のサイズは max(1, *n*) 以上でなければならない。
work(lwork) は、ワークスペース配列。

lda
INTEGER。配列 *a* の第 1 次元のサイズ。
max(1, *n*) 以上でなければならない。

vl, vu
REAL (ssyevr の場合)
DOUBLE PRECISION (dsyevr の場合)
range = 'V' の場合、固有値を探す区間の下限値と上限値。
制約 : *vl* < *vu*
range = 'A' または 'I' の場合、*vl* と *vu* は参照されない。

il, iu
INTEGER
range = 'I' の場合、求める固有値の最小インデックスと最大インデックス (インデックスは昇順)。
制約 : $n > 0$ の場合は、$1 \le il \le iu \le n$。$n = 0$ の場合は、$il = 1$ かつ $iu = 0$
range = 'A' または 'V' の場合、*il* と *iu* は参照されない。

abstol
REAL (ssyevr の場合)
DOUBLE PRECISION (dsyevr の場合)
各固有値および固有ベクトルに許される絶対誤差許容値。
jobz = 'V' の場合、出力される固有値と固有ベクトルの誤差ノルムが *abstol* 以内になる。また、異なる固有ベクトル間のドット積も *abstol* 以内になる。
abstol < $n\varepsilon\|T\|_1$ の場合は、$n\varepsilon\|T\|_1$ がかわりに使用される (ε はマシン精度)。固有値は、*abstol* とは無関係に、$\varepsilon\|T\|_1$ の精度で計算される。高い相対精度が必要な場合は、*abstol* を ?lamch('S') に設定する。

ldz
INTEGER。出力配列 *z* のリーディング・ディメンジョンのサイズ。
次の制約がある。
jobz = 'N' の場合は、$ldz \ge 1$
jobz = 'V' の場合は、$ldz \ge \max(1, n)$

lwork
INTEGER。配列 *work* のサイズ。
制約 : $lwork \ge \max(1, 26n)$。*lwork* の推奨値は、「アプリケーション・ノート」を参照。

iwork
INTEGER
ワークスペース配列 DIMENSION は *liwork*

liwork
INTEGER
配列 *iwork* のサイズ、$lwork \ge \max(1, 10n)$

出力パラメータ

a	終了時に、マトリックス *A* の下三角部分 (*uplo* = 'L' の場合) または上三角部分 (*uplo* = 'U' の場合) が、対角成分を含めて上書きされる。
m	INTEGER。見つかった固有値の総数、$0 \le m \le n$。 *range* = 'A' の場合は *m* = *n*、*range* = 'I' の場合は *m* = *iu*-*il*+1
w, *z*	REAL (ssyevr の場合) DOUBLE PRECISION (dsyevr の場合) 配列 : *w*(*) は、DIMENSION が max(1,*n*) 以上で、選択された固有値が昇順に *w*(1) ～ *w*(*m*) に格納される。 *z*(*ldz*, *)、第 2 次元のサイズは max(1, *m*) 以上。 *jobz* = 'V' の場合、*info* = 0 ならば、*z* の先頭 *m* 列に、マトリックス *T* の選択された固有値に対応する正規直交固有ベクトルが格納される。すなわち、*w*(*i*) に対応する固有ベクトルが、*z* の *i* 番目の列に格納される。 *jobz* = 'N' の場合、*z* は参照されない。 注 : 配列 *z* には、max(1, *m*) 以上の列が提供されなければならない。*range* = 'V' の場合は、*m* の正確な値が事前にわからないため、上限値を使用する必要がある。
isuppz	INTEGER 配列 DIMENSION は 2*max(1, *m*) 以上。 *z* に入っている固有ベクトルのサポート情報、すなわち *z* に入っている非ゼロ成分を示すインデックス。*i* 番目の固有ベクトルは、*isuppz*(2*i*-1) から *isuppz*(2*i*) までの成分のみ非ゼロである。 *range* = 'A'、または *range* = 'I' で *iu*-*il* = *n*-1 の場合にのみ有効である。
work(1)	終了時に、*info* = 0 ならば、*lwork* に必要な最小値が *work*(1) に返される。

iwork(1) 終了時に、*info* = 0 ならば、*liwork* に必要な最小値が *iwork(1)* に返される。

info INTEGER

info = 0 の場合は、正常終了している。

info = *-i* の場合は、*i* 番目のパラメータ値が不正である。

info = *i* の場合、内部エラーが発生した。

アプリケーション・ノート

最適パフォーマンスを得るには、*lwork* ≥ (*nb*+6)*n に設定する。*nb* は、ilaenv が ?sytrd と ?ormtr に対して返したブロックサイズである。必要なワークスペース・サイズが不明な場合は、*lwork* に大きな値を設定して実行してみる。ルーチン終了後に *work(1)* の値を調べ、以降はその値を使用する。

通常の ?stegr の実行で NaN と無限大が発生するため、NaN と無限大が IEEE 標準のデフォルト方法以外で処理される環境では、浮動小数点例外によって異常終了する可能性がある。

?heevr

「比較的安定な表現」を使用して、
エルミート・マトリックスの固有値を
選択的に計算し、さらにオプションで、
固有ベクトルも計算する。

```
call cheevr ( jobz, range, uplo, n, a, lda, vl, vu, il, iu, abstol, m,
              w, z, ldz, isuppz, work, lwork, rwork, lrwork, iwork,
              liwork, info)
call zheevr ( jobz, range, uplo, n, a, lda, vl, vu, il, iu, abstol, m,
              w, z, ldz, isuppz, work, lwork, rwork, lrwork, iwork,
              liwork, info)
```

説明

このルーチンは、複素エルミート・マトリックス T の固有値を選択的に計算し、さらにオプションで、固有ベクトルも計算する。計算する固有値と固有ベクトルは、目的の固有値の範囲、または固有値のインデックスの範囲で選択できる。

可能であれば、?heevr は cstegr/zstegr を呼び出し、「比較的安定な表現」を使用して、固有スペクトルを計算する。?stegr は、固有値の計算には *dqds* アルゴリズムを使用するが、直交固有ベクトルは、種々の「良好な」LDL^T 表現 (「比較的安定な表現」とも呼ばれる) から計算する。グラム - シュミットの直交化はできる限り避ける。アルゴリズムの各ステップを詳しく説明すると次のようになる。T のまだ縮退されていない i 番目のブロックに対して、

(a) $L_i D_i L_i^T$ が比較的安定な表現になるように、$T - \sigma_i = L_i D_i L_i^T$ を計算する。

(b) *dqds* アルゴリズムによって、$L_i D_i L_i^T$ の固有値 λ_j を高い相対精度で計算する。

(c) 値が近い固有値からなるクラスタが存在する場合は、そのクラスタに近い σ_i を「選択」し、ステップ (a) に戻る。

(d) $L_i D_i L_i^T$ の近似的な固有値 λ_j に対する固有ベクトルを、階数表示のツイスト分解によって求める。

出力値に必要な精度は、入力パラメータ *abstol* で指定できる。

ルーチン *?heevr* は、IEEE-754 の浮動小数点標準に準拠しているマシンにおいてフル・スペクトルが要求された場合は、cstegr/zstegr を呼び出す。IEEE-754 に準拠しないマシンの場合や、部分スペクトルが要求された場合は、sstebz/dstebz と cstein/zstein を呼び出す。

入力パラメータ

jobz
: CHARACTER*1。'N' または 'V' でなければならない。
job ='N' の場合、固有値のみ計算する。
job ='V' の場合、固有値と固有ベクトルを計算する。

range
: CHARACTER*1。'A'、'V'、'I' のどれかでなければならない。
range ='A' の場合、固有値をすべて計算する。
range ='V' の場合、半開区間内の固有値 λ_i ($vl < \lambda_i \le vu$) を計算する。
range ='I' の場合、インデックスが *il* ～ *iu* の固有値を計算する。
range ='V' または 'I' の場合、sstebz/dstebz と cstein/zstein を呼び出す。

n
: INTEGER。マトリックス A の次数 ($n \ge 0$)。

a, work
: COMPLEX (cheevr の場合)
DOUBLE COMPLEX (zheevr の場合)
配列 :
*a(lda, *)* は、エルミート・マトリックス A の上または下 (*uplo* の指定に従う) 三角部分を格納する配列。*a* の第 2 次元のサイズは max(1, *n*)以上でなければならない。
work(lwork) は、ワークスペース配列。

lda
: INTEGER。配列 *a* の第 1 次元のサイズ。
max(1, *n*) 以上でなければならない。

vl, vu

REAL (cheevr の場合)
DOUBLE PRECISION (zheevr の場合)
range = 'V' の場合、固有値を探す区間の下限値と上限値。
制約 : *vl* < *vu*

range = 'A' または 'I' の場合、*vl* と *vu* は参照されない。

il, iu

INTEGER
range = 'I' の場合、求める固有値の最小インデックスと最大インデックス (インデックスは昇順)。
制約 : $n > 0$ の場合は、$1 \le il \le iu \le n$。$n = 0$ の場合は、$il = 1$ かつ $iu = 0$

range = 'A' または 'V' の場合、*il* と *iu* は参照されない。

abstol

REAL (cheevr の場合)
DOUBLE PRECISION (zheevr の場合)
各固有値と固有ベクトルに許される絶対誤差許容値。*jobz* = 'V' の場合、出力される固有値と固有ベクトルの誤差ノルムが *abstol* 以内になる。また、異なる固有ベクトル間のドット積も *abstol* 以内になる。*abstol* $< n\varepsilon\|T\|_1$ の場合は、$n\varepsilon\|T\|_1$ がかわりに使用される (ε はマシン精度)。固有値は、*abstol* とは無関係に、$\varepsilon\|T\|_1$ の精度で計算される。高い相対精度が必要な場合は、*abstol* を ?lamch('S') に設定する。

ldz

INTEGER。出力配列 z のリーディング・ディメンジョンのサイズ。
次の制約がある。
jobz = 'N' の場合は、$ldz \ge 1$
jobz = 'V' の場合は、$ldz \ge \max(1, n)$

lwork

INTEGER。配列 *work* のサイズ。
制約 : $lwork \ge \max(1, 2n)$。*lwork* の推奨値は、「アプリケーション・ノート」を参照。

rwork

REAL (cheevr の場合)
DOUBLE PRECISION (zheevr の場合)
ワークスペース配列 DIMENSION は *lrwork*

lrwork INTEGER
配列 *rwork* のサイズ、*lwork* ≥ max(1, 24*n*)

iwork INTEGER
ワークスペース配列 DIMENSION は *liwork*

liwork INTEGER
配列 *iwork* のサイズ、*lwork* ≥ max(1, 10*n*)

出力パラメータ

a 終了時に、マトリックス *A* の下三角部分 (*uplo* = 'L' の場合) または上三角部分 (*uplo* = 'U' の場合) が、対角成分を含めて上書きされる。

m INTEGER。見つかった固有値の総数、0 ≤ *m* ≤ *n*。
range = 'A' の場合は *m* = *n*、*range* = 'I' の場合は *m* = *iu*-*il*+1

w REAL (cheevr の場合)
DOUBLE PRECISION (zheevr の場合)
配列 DIMENSION は max(1, *n*) 以上。選択された固有値が昇順に *w*(1) ～ *w*(*m*) に格納される。

z COMPLEX (cheevr の場合)
DOUBLE COMPLEX (zheevr の場合)
配列 *z*(*ldz*, *)。*z* の第 2 次元のサイズは、max(1, *m*) 以上でなければならない。
jobz = 'V' の場合、*info* = 0 ならば、*z* の先頭 *m* 列に、マトリックス *T* の選択された固有値に対応する正規直交固有ベクトルが格納される。すなわち、*w*(*i*) に対応する固有ベクトルが、*z* の *i* 番目の列に格納される。
jobz = 'N' の場合、*z* は参照されない。
注 : 配列 *z* には、max(1, *m*) 以上の列が提供されなければならない。*range* = 'V' の場合は、*m* の正確な値が事前にわからないため、上限値を使用する必要がある。

isuppz INTEGER
配列 DIMENSION は 2*max(1, *m*) 以上。

	z に入っている固有ベクトルのサポート情報、すなわち *z* に入っている非ゼロ成分を示すインデックス。*i* 番目の固有ベクトルは、*isuppz*(2*i*-1) から *isuppz*(2*i*) までの成分のみ非ゼロである。
work(1)	終了時に、*info* = 0 ならば、*lwork* に必要な最小値が *work(1)* に返される。
rwork(1)	終了時に、*info* = 0 ならば、*lrwork* に必要な最小値が *rwork(1)* に返される。
iwork(1)	終了時に、*info* = 0 ならば、*liwork* に必要な最小値が *iwork(1)* に返される。
info	INTEGER *info* = 0 の場合は、正常終了している。 *info* = -*i* の場合は、*i* 番目のパラメータ値が不正である。 *info* = i の場合、内部エラーが発生した。

アプリケーション・ノート

最適パフォーマンスを得るには、*lwork* ≥ (*nb*+1)***n* に設定する。*nb* は、ilaenv が ?hetrd と ?unmtr に対して返したブロックサイズである。必要なワークスペース・サイズが不明な場合は、*lwork* に大きな値を設定して実行してみる。ルーチン終了後に *work(1)* の値を調べ、以降はその値を使用する。

通常の ?stegr の実行で NaN と無限大が発生するため、NaN と無限大が IEEE 標準のデフォルト方法以外で処理される環境では、浮動小数点例外によって異常終了する可能性がある。

?spev

パックド形式の実対称マトリックスの
固有値をすべて計算し、さらに
オプションで、固有ベクトルも計算する。

```
call sspev (jobz, uplo, n, ap, w, z, ldz, work, info)
call dspev (jobz, uplo, n, ap, w, z, ldz, work, info)
```

説明

このルーチンは、パックド形式の実対称マトリックス A の固有値をすべて計算し、さらにオプションで、固有ベクトルも計算する。

入力パラメータ

jobz
CHARACTER*1。'N' または 'V' でなければならない。
job ='N' の場合、固有値のみ計算する。
job ='V' の場合、固有値と固有ベクトルを計算する。

uplo
CHARACTER*1。'U' または 'L' でなければならない。
uplo ='U' の場合、*ap* に、パックド形式の A の上三角部分を格納する。
uplo ='L' の場合、*ap* に、パックド形式の A の下三角部分を格納する。

n
INTEGER。マトリックス A の次数 ($n \geq 0$)。

ap, work
REAL (sspev の場合)
DOUBLE PRECISION (dspev の場合)
配列 :
ap(*) には、対称マトリックスAの上または下 (*uplo* の指定に従う) 三角部分をパックド形式で格納する。
ap のサイズは、 max(1, *n**(*n*+1)/2) 以上でなければならない。
work(*) は、ワークスペース配列 DIMENSION は max(1, 3*n*) 以上。

ldz INTEGER。出力配列 z のリーディング・ディメンジョンのサイズ。
次の制約がある。
jobz = 'N' の場合、*ldz* ≥ 1
jobz = 'V' の場合、*ldz* ≥ max(1, *n*)

出力パラメータ

w, z REAL (sspev の場合)
DOUBLE PRECISION (dspev の場合)
配列 :
w(*)、DIMENSION は max(1, *n*) 以上。
info = 0 の場合、*w* に、マトリックス *A* の固有値が昇順に格納される。
z(*ldz*, *)、*z* の第 2 次元のサイズは max(1, *n*) 以上でなければならない。
jobz = 'V' の場合、*info* = 0 ならば、*z* に、マトリックス *A* の正規直交固有ベクトルが格納される。そのとき、*w*(*i*) に対応する固有ベクトルが、*z* の *i* 番目の列に格納される。
jobz = 'N' の場合、*z* は参照されない。

ap 終了時に、この配列は、三重対角形式に縮退したときに生成された値で上書きされる。すなわち、生成された三重対角マトリックスの対角成分と非対角成分によって、A の対応する成分が上書きされる。

info INTEGER
info = 0 の場合は、正常終了している。
info = -*i* の場合は、*i* 番目のパラメータ値が不正である。
info = *i* の場合は、このアルゴリズムが収束に失敗した。*i* は、中間の三重対角形式で、ゼロに収束しなかった成分の個数を表す。

?hpev

パックド形式のエルミート・
マトリックスの固有値をすべて計算し、
さらにオプションで、固有ベクトルも
計算する。

```
call chpev (jobz, uplo, n, ap, w, z, ldz, work, rwork, info)
call zhpev (jobz, uplo, n, ap, w, z, ldz, work, rwork, info)
```

説明

このルーチンは、パックド形式のエルミート・マトリックス *A* の固有値をすべて計算し、さらにオプションで、固有ベクトルも計算する。

入力パラメータ

jobz
CHARACTER*1。'N' または 'V' でなければならない。
job ='N' の場合、固有値のみ計算する。
job ='V' の場合、固有値と固有ベクトルを計算する。

uplo
CHARACTER*1。'U' または 'L' でなければならない。
uplo ='U' の場合、*ap* に、パックド形式の *A* の上三角部分を格納する。
uplo ='L' の場合、*ap* に、パックド形式の *A* の下三角部分を格納する。

n
INTEGER。マトリックス *A* の次数 ($n \geq 0$)。

ap, work
COMPLEX (chpev の場合)
DOUBLE COMPLEX (zhpev の場合)
配列 :
ap(*) には、エルミート・マトリックス *A* の上または下 (*uplo* の指定に従う) 三角部分をパックド形式で格納する。*ap* のサイズは、max(1, n*(n+1)/2) 以上でなければならない。
work(*) は、ワークスペース配列 DIMENSION は max(1, 2n-1) 以上。

ldz INTEGER。出力配列 *z* のリーディング・ディメンジョンのサイズ。
次の制約がある。
jobz ='N' の場合、*ldz* ≥ 1
jobz ='V' の場合、*ldz* ≥ max(1, *n*)

rwork REAL (chpev の場合)
DOUBLE PRECISION (zhpev の場合)
ワークスペース配列 DIMENSION は max(1, 3*n*-2) 以上。

出力パラメータ

w REAL (chpev の場合)
DOUBLE PRECISION (zhpev の場合)
配列 DIMENSION は max(1, *n*) 以上。
info = 0 の場合、*w* に、マトリックス *A* の固有値が昇順に格納される。

z COMPLEX (chpev の場合)
DOUBLE COMPLEX (zhpev の場合)
配列 z(*ldz*, *)。*z* の第 2 次元のサイズは max(1, *n*) 以上でなければならない。
jobz ='V' の場合、*info* = 0 ならば、*z* に、マトリックス *A* の正規直交固有ベクトルが格納される。そのとき、*w*(*i*) に対応する固有ベクトルが、*z* の *i* 番目の列に格納される。
jobz ='N' の場合、*z* は参照されない。

ap 終了時に、この配列は、三重対角形式に縮退したときに生成された値で上書きされる。すなわち、生成された三重対角マトリックスの対角成分と非対角成分によって、A の対応する成分が上書きされる。

info INTEGER
info = 0 の場合は、正常終了している。
info = -*i* の場合は、*i* 番目のパラメータ値が不正である。
info = *i* の場合は、このアルゴリズムが収束に失敗した。*i* は、中間の三重対角形式で、ゼロに収束しなかった成分の個数を表す。

?spevd

分割統治アルゴリズムを使用して、
パックド形式の実対称マトリックスの
固有値をすべて計算し、さらに
オプションで、固有ベクトルも計算する。

```
call sspevd (job, uplo, n, ap, w, z, ldz, work, lwork, iwork, liwork,
             info)
call dspevd (job, uplo, n, ap, w, z, ldz, work, lwork, iwork, liwork,
             info)
```

説明

このルーチンは、パックド形式の実対称マトリックス A の固有値をすべて計算し、さらにオプションで、固有ベクトルも計算する。言い換えると、A のスペクトル因子分解 $A = Z\Lambda Z^T$ が求められる。
Λ は対角成分が固有値 λ_i であるような対角マトリックス、Z は各列が固有ベクトル z_i であるような直交マトリックスである。したがって、次のように表現できる。

$$Az_i = \lambda_i z_i \text{ (ただし、} i = 1, 2, ..., n)$$

このルーチンは、固有ベクトルも要求された場合は、分割統治アルゴリズムを使用して、固有値と固有ベクトルを計算する。ただし、固有値のみ要求された場合は、Pal-Walker-Kahan 変形版の *QL* または *QR* アルゴリズムを使用して、固有値を計算する。

入力パラメータ

job CHARACTER*1。'N' または 'V' でなければならない。
job ='N' の場合、固有値のみ計算する。
job ='V' の場合、固有値と固有ベクトルを計算する。

uplo CHARACTER*1。'U' または 'L' でなければならない。
uplo ='U' の場合、*ap* に、パックド形式の A の上三角部分を格納する。
uplo ='L' の場合、*ap* に、パックド形式の A の下三角部分を格納する。

n INTEGER。マトリックス A の次数 ($n \geq 0$)。

ap, work
REAL (sspevd の場合)
DOUBLE PRECISION (dspevd の場合)
配列 :
ap(*) には、対称マトリックス A の上または下 (*uplo* の指定に従う) 三角部分をパックド形式で格納する。
ap のサイズは、max(1, *n**(*n*+1)/2) 以上でなければならない。
work(*) は、ワークスペース配列 DIMENSION は *lwork* 以上。

ldz
INTEGER。出力配列 *z* のリーディング・ディメンジョンのサイズ。
次の制約がある。
job = 'N' の場合、*ldz* ≥ 1
job = 'V' の場合、*ldz* ≥ max(1, *n*)

lwork
INTEGER。配列 *work* のサイズ。
次の制約がある。
n ≤ 1 の場合、*lwork* ≥ 1
job = 'N' で *n* > 1 の場合、*lwork* ≥ 2*n*
job = 'V' で *n* > 1 の場合、*lwork* ≥ $2n^2$+ (5+2*k*) **n*+1
(ただし、*k* は、$2^k \geq n$ を満足する最小の整数)
lwork = -1 の場合はワークスペースのクエリとみなされ、ルーチンは *work* 配列の最適サイズだけを計算し、その値を *work* 配列の最初のエントリとして返し、xerbla は *lwork* に関するエラー・メッセージを生成しない。

iwork
INTEGER
ワークスペース配列 DIMENSION は *liwork* 以上。

liwork
INTEGER。配列 *iwork* のサイズ。
次の制約がある。
n ≤ 1 の場合、*liwork* ≥ 1
job = 'N' で *n* > 1 の場合、*liwork* ≥ 1
job = 'V' で *n* > 1 の場合、*liwork* ≥ 5*n*+2
liwork = -1 の場合はワークスペースのクエリとみなされ、ルーチンは *iwork* 配列の最適サイズだけを計算し、その値を *iwork* 配列の最初のエントリとして返し、xerbla は *liwork* に関するエラー・メッセージを生成しない。

出力パラメータ

w, z
REAL (sspevd の場合)
DOUBLE PRECISION (dspevd の場合)
配列 :
w(*)、DIMENSION は max(1, *n*) 以上。
info = 0 の場合、マトリックス *A* の固有値が昇順に格納される。*info* も参照のこと。
*z(ldz, *)*、*z* の第 2 次元のサイズは、
job = 'N' の場合は、1 以上。
job = 'V' の場合は、max(1, *n*) 以上。
job = 'V' の場合、この配列は、*A* の固有ベクトルの入った直交マトリックス *Z* で上書きされる。
job = 'N' の場合、*z* は参照されない。

ap
終了時に、この配列は、三重対角形式に縮退したときに生成された値で上書きされる。すなわち、生成された三重対角マトリックスの対角成分と非対角成分によって、*A* の対応する成分が上書きされる。

work(1)
終了時に、*info* = 0 ならば、*lwork* に最適な値が *work(1)* に返される。

iwork(1)
終了時に、*info* = 0 ならば、*liwork* に最適な値が *iwork(1)* に返される。

info
INTEGER
info = 0 の場合は、正常終了している。
info = *i* の場合は、このアルゴリズムが収束に失敗した。*i* は、中間の三重対角形式で、ゼロに収束しなかった成分の個数を表す。
info = -*i* の場合は、*i* 番目のパラメータ値が不正である。

アプリケーション・ノート

計算された固有値と固有ベクトルは、正確には、$\|E\|_2 = O(\varepsilon)\,\|T\|_2$ であるようなマトリックス $T + E$ (ただし、ε はマシン精度) のものである。

このルーチンの複素数版は ?hpevd である。

フル形式のマトリックスに対する ?syevd と、バンド形式のマトリックスに関する ?sbevd も参照のこと。

?hpevd

分割統治アルゴリズムを使用して、
パックド形式の複素エルミート・
マトリックスの固有値をすべて計算し、
さらにオプションで、固有ベクトルも
計算する。

```
call chpevd (job, uplo, n, ap, w, z, ldz, work, lwork, rwork, lrwork,
             iwork, liwork, info)
call zhpevd (job, uplo, n, ap, w, z, ldz, work, lwork, rwork, lrwork,
             iwork, liwork, info)
```

説明

このルーチンは、パックド形式の複素エルミート・マトリックス A の固有値をすべて計算し、さらにオプションで、固有ベクトルも計算する。言い換えると、A のスペクトル因子分解 $A = Z\Lambda Z^H$ が求められる。Λ は対角成分が固有値 λi であるような実対角マトリックス、Z は各列が固有ベクトル z_i であるような (複素) ユニタリ・マトリックスである。したがって、次のように表現できる。

$$Az_i = \lambda_i z_i \text{ (ただし、} i = 1, 2, ..., n)$$

このルーチンは、固有ベクトルも要求された場合は、分割統治アルゴリズムを使用して、固有値と固有ベクトルを計算する。ただし、固有値のみ要求された場合は、Pal-Walker-Kahan 変形版の QL または QR アルゴリズムを使用して、固有値を計算する。

入力パラメータ

job	CHARACTER*1。'N' または 'V' でなければならない。 *job*='N' の場合、固有値のみ計算する。 *job*='V' の場合、固有値と固有ベクトルを計算する。
uplo	CHARACTER*1。'U' または 'L' でなければならない。

uplo = 'U' の場合、ap に、パックド形式の A の上三角部分を格納する。
uplo = 'L' の場合、ap に、パックド形式の A の下三角部分を格納する。

n　INTEGER。マトリックス A の次数 ($n \geq 0$)。

ap, work　COMPLEX (chpevd の場合)
DOUBLE COMPLEX (zhpevd の場合)
配列 :
ap(*) には、エルミート・マトリックス A の上または下 (uplo の指定に従う) 三角部分をパックド形式で格納する。ap のサイズは、max(1, n*(n+1)/2) 以上でなければならない。
work(*) は、ワークスペース配列 DIMENSION は lwork 以上。

ldz　INTEGER。出力配列 z のリーディング・ディメンジョンのサイズ。
次の制約がある。
job = 'N' の場合、ldz ≥ 1
job = 'V' の場合、ldz ≥ max(1, n)

lwork　INTEGER。配列 work のサイズ。
次の制約がある。
$n \leq 1$ の場合、lwork ≥ 1
job = 'N' で $n > 1$ の場合、lwork ≥ n
job = 'V' で $n > 1$ の場合、lwork ≥ 2n

rwork　REAL (chpevd の場合)
DOUBLE PRECISION (zhpevd の場合)
ワークスペース配列 DIMENSION は lrwork 以上。

lrwork　INTEGER。配列 rwork のサイズ。
次の制約がある。
$n \leq 1$ の場合、lrwork ≥ 1
job = 'N' で $n > 1$ の場合、lrwork ≥ n
job = 'V' で $n > 1$ の場合、lrwork ≥ $3n^2 + (4+2k) * n + 1$
(ただし、k は、$2^k \geq n$ を満足する最小の整数)。

iwork　INTEGER
ワークスペース配列 DIMENSION は liwork 以上。

liwork INTEGER。配列 *iwork* のサイズ。
次の制約がある。
$n \le 1$ の場合、*liwork* ≥ 1
job = 'N' で $n > 1$ の場合、*liwork* ≥ 1
job = 'V' で $n > 1$ の場合、*liwork* $\ge 5n+2$

出力パラメータ

w REAL (chpevd の場合)
DOUBLE PRECISION (zhpevd の場合)
配列 DIMENSION は max(1, *n*) 以上。
info = 0 の場合、マトリックス *A* の固有値が昇順に格納される。*info* も参照のこと。

z COMPLEX (chpevd の場合)
DOUBLE COMPLEX (zhpevd の場合)
配列 DIMENSION は *(ldz, *)*。*z* の第 2 次元のサイズは、次の値でなければならない。
job = 'N' の場合は、1 以上。
job = 'V' の場合は、max(1, *n*) 以上。
job = 'V' の場合、この配列は、*A* の固有ベクトルの入ったユニタリ・マトリックス Z で上書きされる。
job = 'N' の場合、*z* は参照されない。

ap 終了時に、この配列は、三重対角形式に縮退したときに生成された値で上書きされる。すなわち、生成された三重対角マトリックスの対角成分と非対角成分によって、A の対応する成分が上書きされる。

work(1) 終了時に、*lwork* > 0 ならば、*lwork* に必要な最小値が *work(1)* の実部に返される。

rwork(1) 終了時に、*lrwork* > 0 ならば、*lrwork* に必要な最小値が *rwork(1)* に返される。

iwork(1) 終了時に、*liwork* > 0 ならば、*liwork* に必要な最小値が *iwork(1)* に返される。

info INTEGER
info = 0 の場合は、正常終了している。
info = *i* の場合は、このアルゴリズムが収束に失敗した。*i* は、中間の三重対角形式で、ゼロに収束しなかった成分の個数を表す。

info = *-i* の場合は、*i* 番目のパラメータ値が不正である。

アプリケーション・ノート

計算された固有値と固有ベクトルは、正確には、$\|E\|_2 = O(\varepsilon)\,\|T\|_2$ であるようなマトリックス $T + E$ (ただし、ε はマシン精度) のものである。

このルーチンの実数版は、?spevd である。

フル形式のマトリックスに対する ?heevd と、バンド形式のマトリックスに関する ?hbevd も参照のこと。

?spevx

パックド形式の実対称マトリックスの
固有値を選択的に計算し、さらに
オプションで、固有ベクトルも計算する。

```
call sspevx (jobz, range, uplo, n, ap, vl, vu, il, iu, abstol, m, w,
             z, ldz, work, iwork, ifail, info)
call dspevx (jobz, range, uplo, n, ap, vl, vu, il, iu, abstol, m, w,
             z, ldz, work, iwork, ifail, info)
```

説明

パックド形式の実対称マトリックス A の固有値を選択的に計算し、さらにオプションで、固有ベクトルも計算する。計算する固有値と固有ベクトルは、目的の固有値の範囲、または固有値のインデックスの範囲で選択できる。

入力パラメータ

jobz CHARACTER*1。'N' または 'V' でなければならない。
job ='N' の場合、固有値のみ計算する。
job ='V' の場合、固有値と固有ベクトルを計算する。

range CHARACTER*1。'A'、'V'、'I' のどれかでなければならない。
range ='A' の場合、固有値をすべて計算する。
range ='V' の場合、半開区間内の固有値 λ_i ($vl < \lambda_i \le vu$) を計算する。
range ='I' の場合、インデックスが *il* ～ *iu* の固有値を計算する。

uplo CHARACTER*1。'U' または 'L' でなければならない。
uplo ='U' の場合、*ap* に、パックド形式の A の上三角部分を格納する。
uplo ='L' の場合、*ap* に、パックド形式の A の下三角部分を格納する。

n INTEGER。マトリックス A の次数 ($n \ge 0$)。

ap, work

REAL (sspevx の場合)
DOUBLE PRECISION (dspevx の場合)
配列 :
ap(*) には、対称マトリックス *A* の上または下 (*uplo* の指定に従う) 三角部分をパックド形式で格納する。
ap のサイズは、max(1, *n**(*n*+1)/2) 以上でなければならない。
work(*) は、ワークスペース配列 DIMENSION は max(1, 8*n*) 以上。

vl, vu

REAL (sspevx の場合)
DOUBLE PRECISION (dspevx の場合)
range ='V' の場合、固有値を探す区間の下限値と上限値。
制約 : *vl*< *vu*
range ='A' または 'I' の場合、*vl* と *vu* は参照されない。

il, iu

INTEGER
range ='I' の場合、求める固有値の最小インデックスと最大インデックス (インデックスは昇順)。
制約 : *n* > 0 の場合は、1 ≤ *il* ≤ *iu* ≤ *n*。*n* = 0 の場合は、*il* = 1 かつ *iu* = 0
range ='A' または 'V' の場合、*il* と *iu* は参照されない。

abstol

REAL (sspevx の場合)
DOUBLE PRECISION (dspevx の場合)
各固有値に許される絶対誤差許容値。誤差許容値の詳細は、「アプリケーション・ノート」を参照。

ldz

INTEGER。出力配列 *z* のリーディング・ディメンジョンのサイズ。
次の制約がある。
jobz ='N' の場合、*ldz* ≥ 1
jobz ='V' の場合、*ldz* ≥ max(1, *n*)

iwork

INTEGER
ワークスペース配列 DIMENSION は max(1, 5*n*) 以上。

出力パラメータ

ap
終了時に、この配列は、三重対角形式に縮退したときに生成された値で上書きされる。すなわち、生成された三重対角マトリックスの対角成分と非対角成分によって、*A* の対応する成分が上書きされる。

m
INTEGER。見つかった固有値の総数、$0 \le m \le n$。
range = 'A' の場合は *m* = *n*、*range* = 'I' の場合は *m* = *iu*-*il*+1

w, z
REAL (sspevx の場合)
DOUBLE PRECISION (dspevx の場合)
配列 :
w(*)、サイズは max(1, *n*) 以上。
info = 0 の場合、選択された *A* の固有値が昇順に格納される。
z*(ldz, *)*、*z* の第 2 次元のサイズは max(1, *m*) 以上でなければならない。
jobz = 'V' の場合、*info* = 0 ならば、*z* の先頭 *m* 列に、マトリックス *A* の選択された固有値に対応する正規直交固有ベクトルが格納される。すなわち、*w*(*i*) に対応する固有ベクトルが、*z* の *i* 番目の列に格納される。収束に失敗した固有ベクトルは、それに対応する *z* の列に、最終的な近似固有ベクトルが格納され、この固有ベクトルのインデックスが *ifail* に返される。
jobz = 'N' の場合、*z* は参照されない。
注 : 配列 *z* には、max(1, *m*) 以上の列が提供されなければならない。*range* = 'V' の場合は、*m* の正確な値が事前にわからないため、上限値を使用する必要がある。

ifail
INTEGER。配列 DIMENSION は max(1, *n*) 以上。
jobz = 'V' の場合、*info* = 0 ならば、*ifail* の先頭 *m* 成分はゼロになる。*info* > 0 ならば、収束に失敗した固有ベクトルのインデックスが *ifail* に格納される。
jobz = 'N' の場合、*ifail* は参照されない。

info INTEGER

info = 0 の場合は、正常終了している。

info = *-i* の場合は、*i* 番目のパラメータ値が不正である。

info = *i* の場合、*i* 番目の固有ベクトルが収束に失敗した。その固有ベクトルのインデックスが配列 *ifail* に格納される。

アプリケーション・ノート

近似的な固有値でも、幅が *abstol*+ε*max(|a|, |b|) 以下の区間 [a, b] 内に納まる場合は、収束したと見なされる (ε はマシン精度)。*abstol* がゼロまたは負の場合は、かわりに ε*$\|T\|_1$ を使用する。*T* は、*A* を三重対角形式に縮退して得られた三重対角マトリックスである。

固有値が最も正確に計算されるのは、*abstol* がゼロでなく、アンダーフローのしきい値の 2 倍、2*?lamch('S') に設定されたときである。このルーチンで *info* > 0 が返された場合は、固有ベクトルのいくつかが収束に失敗したことを意味するので、*abstol* を 2*?lamch('S') に設定して、やり直してみる。

?hpevx

パックド形式のエルミート・
マトリックスの固有値を選択的に計算し、
さらにオプションで、固有ベクトルも
計算する。

```
call chpevx (jobz, range, uplo, n, ap, vl, vu, il, iu, abstol, m, w,
             z, ldz, work, rwork, iwork, ifail, info)
call zhpevx (jobz, range, uplo, n, ap, vl, vu, il, iu, abstol, m, w,
             z, ldz, work, rwork, iwork, ifail, info)
```

説明

このルーチンは、パックド形式の複素エルミート・マトリックス A の固有値を選択的に計算し、さらにオプションで、固有ベクトルも計算する。計算する固有値と固有ベクトルは、目的の固有値の範囲、または固有値のインデックスの範囲で選択できる。

入力パラメータ

jobz CHARACTER*1。'N' または 'V' でなければならない。
job ='N' の場合、固有値のみ計算する。
job ='V' の場合、固有値と固有ベクトルを計算する。

range CHARACTER*1。'A'、'V'、'I' のどれかでなければならない。
range ='A' の場合、固有値をすべて計算する。
range ='V' の場合、半開区間内の固有値 λ_i ($vl < \lambda_i \le vu$) を計算する。
range ='I' の場合、インデックスが *il* ～ *iu* の固有値を計算する。

uplo CHARACTER*1。'U' または 'L' でなければならない。
uplo ='U' の場合、*ap* に、パックド形式の A の上三角部分を格納する。
uplo ='L' の場合、*ap* に、パックド形式の A の下三角部分を格納する。

n INTEGER。マトリックス A の次数 ($n \ge 0$)。

ap, work

COMPLEX (chpevx の場合)
DOUBLE COMPLEX (zhpevx の場合)
配列 :
ap(*) には、エルミート・マトリックス *A* の上または下 (*uplo* の指定に従う) 三角部分をパックド形式で格納する。*ap* のサイズは、max(1, *n**(*n*+1)/2) 以上でなければならない。

work(*) は、ワークスペース配列 DIMENSION は max(1, 2*n*) 以上。

vl, vu

REAL (chpevx の場合)
DOUBLE PRECISION (zhpevx の場合)
range ='V' の場合、固有値を探す区間の下限値と上限値。
制約 : *vl*< *vu*
range ='A' または 'I' の場合、*vl* と *vu* は参照されない。

il, iu

INTEGER
range ='I' の場合、求める固有値の最小インデックスと最大インデックス (インデックスは昇順)。
制約 : *n* > 0 の場合は、1 ≤ *il* ≤ *iu* ≤ *n*。*n* = 0 の場合は、*il* = 1 かつ *iu* = 0
range ='A' または 'V' の場合、*il* と *iu* は参照されない。

abstol

REAL (chpevx の場合)
DOUBLE PRECISION (zhpevx の場合)
各固有値に許される絶対誤差許容値。誤差許容値の詳細は、「アプリケーション・ノート」を参照。

ldz

INTEGER。出力配列 *z* のリーディング・ディメンジョンのサイズ。
次の制約がある。
jobz ='N' の場合、*ldz* ≥ 1
jobz ='V' の場合、*ldz* ≥ max(1, *n*)

rwork

REAL (chpevx の場合)
DOUBLE PRECISION (zhpevx の場合)
ワークスペース配列 DIMENSION は max(1, 7*n*) 以上。

iwork

INTEGER
ワークスペース配列 DIMENSION は max(1, 5*n*) 以上。

出力パラメータ

ap
終了時に、この配列は、三重対角形式に縮退したときに生成された値で上書きされる。すなわち、生成された三重対角マトリックスの対角成分と非対角成分によって、*A* の対応する成分が上書きされる。

m
INTEGER。見つかった固有値の総数、$0 \le m \le n$。
range = 'A' の場合は
m = *n*、*range* = 'I' の場合は *m* = *iu*-*il*+1

w
REAL (chpevx の場合)
DOUBLE PRECISION (zhpevx の場合)
配列 DIMENSION は max(1, *n*) 以上。*info* = 0 の場合、選択された *A* の固有値が昇順に格納される。

z
COMPLEX (chpevx の場合)
DOUBLE COMPLEX (zhpevx の場合)
配列 *z(ldz, *)*、*z* の第 2 次元のサイズは max(1, *m*) 以上でなければならない。
jobz = 'V' の場合、*info* = 0 ならば、*z* の先頭 *m* 列に、マトリックス *A* の選択された固有値に対応する正規直交固有ベクトルが格納される。すなわち、*w*(*i*) に対応する固有ベクトルが、*z* の *i* 番目の列に格納される。収束に失敗した固有ベクトルは、それに対応する *z* の列に、最終的な近似固有ベクトルが格納され、この固有ベクトルのインデックスが *ifail* に返される。
jobz = 'N' の場合、*z* は参照されない。
注 : 配列 *z* には、max(1, *m*) 以上の列が提供されなければならない。*range* = 'V' の場合は、*m* の正確な値が事前にわからないため、上限値を使用する必要がある。

ifail
INTEGER。配列 DIMENSION は max(1, *n*) 以上。
jobz = 'V' の場合、*info* = 0 ならば、*ifail* の先頭 *m* 成分はゼロになる。*info* > 0 ならば、収束に失敗した固有ベクトルのインデックスが *ifail* に格納される。
jobz = 'N' の場合、*ifail* は参照されない。

info INTEGER

info = 0 の場合は、正常終了している。

info = *-i* の場合は、i 番目のパラメータ値が不正である。

info = *i* の場合、*i* 番目の固有ベクトルが収束に失敗した。その固有ベクトルのインデックスが配列 *ifail* に格納される。

アプリケーション・ノート

近似的な固有値でも、幅が *abstol*+ε*max(|a|, |b|) 以下の区間 [a, b] 内に納まる場合は、収束したと見なされる (ε はマシン精度)。*abstol* がゼロまたは負の場合は、かわりに $\varepsilon*\|T\|_1$ を使用する。*T* は、*A* を三重対角形式に縮退して得られた三重対角マトリックスである。

固有値が最も正確に計算されるのは、*abstol* がゼロでなく、アンダーフローのしきい値の 2 倍、2*?lamch('S') に設定されたときである。このルーチンで *info* > 0 が返された場合は、固有ベクトルのいくつかが収束に失敗したことを意味するので、*abstol* を 2*?lamch('S') に設定して、やり直してみる。

?sbev

バンド形式の実対称マトリックスの
固有値をすべて計算し、さらに
オプションで、固有ベクトルも計算する。

```
call ssbev (jobz, uplo, n, kd, ab, ldab, w, z, ldz, work, info)
call dsbev (jobz, uplo, n, kd, ab, ldab, w, z, ldz, work, info)
```

説明

このルーチンは、バンド形式の実対称マトリックス *A* の固有値をすべて計算し、さらにオプションで、固有ベクトルも計算する。

入力パラメータ

jobz
CHARACTER*1。'N' または 'V' でなければならない。
jobz ='N' の場合、固有値のみ計算する。
jobz ='V' の場合、固有値と固有ベクトルを計算する。

uplo
CHARACTER*1。'U' または 'L' でなければならない。
uplo ='U' の場合、*ab* に *A* の上三角部分を格納する。
uplo ='L' の場合、*ab* に *A* の下三角部分を格納する。

n
INTEGER。マトリックス *A* の次数 ($n \geq 0$)。

kd
INTEGER。*A* の優対角線または劣対角線の本数 ($kd \geq 0$)。

ab, work
REAL (ssbev の場合)
DOUBLE PRECISION (dsbev の場合)
配列 :
ab (*ldab*, *) は、対称マトリックス *A* の上または下 (*uplo* の指定に従う) 三角部分をバンド形式で格納する配列。
*ab*の第2次元のサイズはmax(1, *n*)以上でなければならない。
work (*) は、ワークスペース配列。
work のサイズは、max(1, 3*n*-2) 以上でなければならない。

ldab INTEGER。*ab* のリーディング・ディメンジョンのサイズ。*kd* +1 以上でなければならない。

ldz INTEGER。出力配列 *z* のリーディング・ディメンジョンのサイズ。
次の制約がある。
jobz ='N' の場合、*ldz* ≥ 1
jobz ='V' の場合、*ldz* ≥ max(1, *n*)

出力パラメータ

w, z REAL (ssbev の場合)
DOUBLE PRECISION (dsbev の場合)
配列 :
w(*)、DIMENSION は max(1, *n*) 以上。
info = 0 の場合、マトリックス *A* の固有値が昇順に格納される。
*z(ldz, *)*、*z* の第 2 次元のサイズは max(1, *n*) 以上でなければならない。
jobz ='V' の場合、*info* = 0 ならば、*z* に、マトリックス *A* の正規直交固有ベクトルが格納される。そのとき、*w*(*i*) に対応する固有ベクトルが、*z* の *i* 番目の列に格納される。
jobz ='N' の場合、*z* は参照されない。

ab 終了時に、この配列は、三重対角形式に縮退したときに生成された値で上書きされる。*uplo* ='U' の場合、三重対角マトリックス *T* の第一優対角成分と対角成分が、*ab* の 2 つの行 (*kd* と *kd*+1) に返される。
uplo ='L' の場合、*T* の対角成分と第一劣対角成分が、*ab* の先頭 2 行に返される。

info INTEGER
info = 0 の場合は、正常終了している。
info = -*i* の場合は、*i* 番目のパラメータ値が不正である。
info = *i* の場合は、このアルゴリズムが収束に失敗した。
i は、中間の三重対角形式で、ゼロに収束しなかった成分の個数を表す。

?hbev

バンド形式のエルミート・マトリックスの
固有値をすべて計算し、さらに
オプションで、固有ベクトルも計算する。

```
call chbev(jobz, uplo, n, kd, ab, ldab, w, z, ldz, work, rwork,info)
call zhbev(jobz, uplo, n, kd, ab, ldab, w, z, ldz, work, rwork,info)
```

説明

このルーチンは、バンド形式の複素エルミート・マトリックス A の固有値をすべて計算し、さらにオプションで、固有ベクトルも計算する。

入力パラメータ

jobz
CHARACTER*1。'N' または 'V' でなければならない。
jobz ='N' の場合、固有値のみ計算する。
jobz ='V' の場合、固有値と固有ベクトルを計算する。

uplo
CHARACTER*1。'U' または 'L' でなければならない。
uplo ='U' の場合、*ab* に A の上三角部分を格納する。
uplo ='L' の場合、*ab* に A の下三角部分を格納する。

n
INTEGER。マトリックス A の次数 ($n \geq 0$)。

kd
INTEGER。A の優対角線または劣対角線の本数 ($kd \geq 0$)。

ab, work
COMPLEX (chbev の場合)
DOUBLE COMPLEX (zhbev の場合)
配列 :
ab (*ldab*, *) は、エルミート・マトリックス A の上または下 (*uplo* の指定に従う) 三角部分をバンド形式で格納する配列。
*ab*の第2次元のサイズはmax(1, *n*)以上でなければならない。
work (*) は、ワークスペース配列。
work のサイズは、max(1, *n*) 以上でなければならない。

ldab INTEGER。*ab* のリーディング・ディメンジョンのサイズ。*kd* +1 以上でなければならない。

ldz INTEGER。出力配列 *z* のリーディング・ディメンジョンのサイズ。
次の制約がある。
jobz ='N' の場合、*ldz* ≥ 1
jobz ='V' の場合、*ldz* ≥ max(1, *n*)

rwork REAL (chbev の場合)
DOUBLE PRECISION (zhbev の場合)
ワークスペース配列 DIMENSION は max(1, 3*n*-2) 以上。

出力パラメータ

w REAL (chbev の場合)
DOUBLE PRECISION (zhbev の場合)
配列 DIMENSION は max(1, *n*) 以上。*info* = 0 の場合、固有値が昇順で格納される。

z COMPLEX (chbev の場合)
DOUBLE COMPLEX (zhbev の場合)
配列 *z(ldz, *)*、*z* の第 2 次元のサイズは max(1, *n*) 以上でなければならない。*jobz* ='V' の場合、*info* = 0 ならば、*z* に、マトリックス *A* の正規直交固有ベクトルが格納される。そのとき、*w*(*i*) に対応する固有ベクトルが、*z* の *i* 番目の列に格納される。
jobz ='N' の場合、*z* は参照されない。

ab 終了時に、この配列は、三重対角形式に縮退したときに生成された値で上書きされる。*uplo* ='U' の場合、三重対角マトリックス *T* の第一優対角成分と対角成分が、*ab* の 2 つの行 (*kd* と *kd*+1) に返される。
uplo ='L' の場合、*T* の対角成分と第一劣対角成分が、*ab* の先頭 2 行に返される。

info INTEGER
info = 0 の場合は、正常終了している。
info = -*i* の場合は、*i* 番目のパラメータ値が不正である。
info = *i* の場合は、このアルゴリズムが収束に失敗した。
i は、中間の三重対角形式で、ゼロに収束しなかった成分の個数を表す。

?sbevd

分割統治アルゴリズムを使用して、
バンド形式の実対称マトリックスの
固有値をすべて計算し、さらに
オプションで固有ベクトルも計算する。

```
call ssbevd (job, uplo, n, kd, ab, ldab, w, z, ldz, work, lwork,
             iwork, liwork, info)
call dsbevd (job, uplo, n, kd, ab, ldab, w, z, ldz, work, lwork,
             iwork, liwork, info)
```

説明

このルーチンは、バンド形式の実対称マトリックス A の固有値をすべて計算し、さらにオプションで固有ベクトルも計算する。言い換えると、次のような A のスペクトル因子分解が求められる。

$$A = Z\Lambda Z^T$$

Λ は対角成分が固有値 λ_i であるような対角マトリックス、Z は各列が固有ベクトル z_i であるような直交マトリックスである。したがって、次のように表現できる。

$$Az_i = \lambda_i z_i \text{ (ただし、} i = 1, 2, ..., n)$$

このルーチンは、固有ベクトルも要求された場合は、分割統治アルゴリズムを使用して、固有値と固有ベクトルを計算する。ただし、固有値のみ要求された場合は、Pal-Walker-Kahan 変形版の *QL* または *QR* アルゴリズムを使用して、固有値を計算する。

入力パラメータ

job CHARACTER*1。'N' または 'V' でなければならない。
job ='N' の場合、固有値のみ計算する。
job ='V' の場合、固有値と固有ベクトルを計算する。

uplo CHARACTER*1。'U' または 'L' でなければならない。
uplo ='U' の場合、*ab* に A の上三角部分を格納する。
uplo ='L' の場合、*ab* に A の下三角部分を格納する。

n	INTEGER。マトリックス A の次数 ($n \geq 0$)。
kd	INTEGER。A の優対角線または劣対角線の本数 ($kd \geq 0$)。
ab, work	REAL (ssbevd の場合) DOUBLE PRECISION (dsbevd の場合) 配列 : *ab* (*ldab, **) は、対称マトリックス A の上または下 (*uplo* の指定に従う) 三角部分をバンド形式で格納する配列。 *ab*の第2次元のサイズは max(1, *n*)以上でなければならない。 *work* (*) は、ワークスペース配列。 *work* のサイズは、*lwork* 以上でなければならない。
ldab	INTEGER。*ab* のリーディング・ディメンジョンのサイズ。*kd*+1 以上でなければならない。
ldz	INTEGER。出力配列 *z* のリーディング・ディメンジョンのサイズ。 次の制約がある。 *job* ='N' の場合、*ldz* ≥ 1 *job* ='V' の場合、*ldz* ≥ max(1, *n*)
lwork	INTEGER。配列 *work* のサイズ。 次の制約がある。 *n* ≤ 1 の場合、*lwork* ≥ 1 *job* ='N' で *n* > 1 の場合、*lwork* ≥ 2*n* *job* ='V' で *n* > 1 の場合、*lwork* ≥ $3n^2$+ (4+2*k*) **n*+1 (ただし、*k* は、$2^k \geq n$ を満足する最小の整数)
iwork	INTEGER ワークスペース配列 DIMENSION は *liwork* 以上。
liwork	INTEGER。配列 *iwork* のサイズ。 次の制約がある。 *n* ≤ 1 の場合、*liwork* ≥ 1 *job* ='N' で *n* > 1 の場合、*liwork* ≥ 1 *job* ='V' で *n* > 1 の場合、*liwork* ≥ 5*n*+2

出力パラメータ

w, z	REAL (ssbevd の場合) DOUBLE PRECISION (dsbevd の場合) 配列 : *w(*)*、DIMENSION は max(1, *n*) 以上。 *info* = 0 の場合、マトリックス *A* の固有値が昇順に格納される。*info* も参照のこと。 *z(ldz, *)*、*z* の第 2 次元のサイズは、 *job* = 'N' の場合は、1 以上。 *job* = 'V' の場合は、max(1, *n*) 以上。 *job* = 'V' の場合、この配列は、*A* の固有ベクトルの入った直交マトリックス *Z* で上書きされる。すなわち、固有値 *w(i)* に対応する固有ベクトルが、*z* の *i* 番目の列に格納される。 *job* = 'N' の場合、*z* は参照されない。
ab	終了時に、この配列は、三重対角形式に縮退したときに生成された値で上書きされる。
work(1)	終了時に、*lwork* > 0 ならば、*lwork* に必要な最小値が *work(1)* に返される。
iwork(1)	終了時に、*liwork* > 0 ならば、*liwork* に必要な最小値が *iwork(1)* に返される。
info	INTEGER *info* = 0 の場合は、正常終了している。 *info* = *i* の場合は、このアルゴリズムが収束に失敗した。*i* は、中間の三重対角形式で、ゼロに収束しなかった成分の個数を表す。 *info* = -*i* の場合は、*i* 番目のパラメータ値が不正である。

アプリケーション・ノート

計算された固有値と固有ベクトルは、正確には、$\|E\|_2 = O(\varepsilon)\,\|T\|_2$ であるようなマトリックス $T + E$ (ただし、ε はマシン精度) のものである。

このルーチンの複素数版は ?hbevd である。

フル形式のマトリックスに対する ?syevd と、パックド形式のマトリックスに関する ?spevd も参照のこと。

?hbevd

分割統治アルゴリズムを使用して、
バンド形式の複素エルミート・
マトリックスの固有値をすべて計算し、
さらにオプションで固有ベクトルも
計算する。

```
call chbevd (job, uplo, n, kd, ab, ldab, w, z, ldz, work, lwork,
             rwork, lrwork, iwork, liwork, info)
call zhbevd (job, uplo, n, kd, ab, ldab, w, z, ldz, work, lwork,
             rwork, lrwork, iwork, liwork, info)
```

説明

このルーチンは、バンド形式の複素エルミート・マトリックス A の固有値をすべて計算し、さらにオプションで固有ベクトルも計算する。言い換えると、A のスペクトル因子分解 $A = Z\Lambda Z^H$ が求められる。
Λ は対角成分が固有値 λ_i であるような実対角マトリックス、Z は各列が固有ベクトル z_i であるような (複素) ユニタリ・マトリックスである。したがって、次のように表現できる。

$$Az_i = \lambda_i z_i \text{ (ただし、} i = 1, 2, ..., n)$$

このルーチンは、固有ベクトルも要求された場合は、分割統治アルゴリズムを使用して、固有値と固有ベクトルを計算する。ただし、固有値のみ要求された場合は、Pal-Walker-Kahan 変形版の *QL* または *QR* アルゴリズムを使用して、固有値を計算する。

入力パラメータ

job CHARACTER*1。'N' または 'V' でなければならない。
job ='N' の場合、固有値のみ計算する。
job ='V' の場合、固有値と固有ベクトルを計算する。

uplo CHARACTER*1。'U' または 'L' でなければならない。
uplo ='U' の場合、*ab* に A の上三角部分を格納する。
uplo ='L' の場合、*ab* に A の下三角部分を格納する。

n INTEGER。マトリックス A の次数 ($n \geq 0$)。

kd	INTEGER。*A* の優対角線または劣対角線の本数 ($kd \geq 0$)。
ab, work	COMPLEX (chbevd の場合) DOUBLE COMPLEX (zhbevd の場合) 配列 : *ab* (*ldab, **) は、エルミート・マトリックス *A* の上または下 (*uplo* の指定に従う) 三角部分をバンド形式で格納する配列。 *ab*の第2次元のサイズはmax(1, *n*)以上でなければならない。 *work* (*) は、ワークスペース配列。 *work* のサイズは、*lwork* 以上でなければならない。
ldab	INTEGER。*ab* のリーディング・ディメンジョンのサイズ。*kd*+1 以上でなければならない。
ldz	INTEGER。出力配列 *z* のリーディング・ディメンジョンのサイズ。 次の制約がある。 *job* = 'N' の場合、$ldz \geq 1$ *job* = 'V' の場合、$ldz \geq \max(1, n)$
lwork	INTEGER。配列 *work* のサイズ。 次の制約がある。 $n \leq 1$ の場合、$lwork \geq 1$ *job* = 'N' で $n > 1$ の場合、$lwork \geq n$ *job* = 'V' で $n > 1$ の場合、$lwork \geq 2n^2$
rwork	REAL (chbevd の場合) DOUBLE PRECISION (zhbevd の場合) ワークスペース配列 DIMENSION は *lrwork* 以上。
lrwork	INTEGER。配列 *rwork* のサイズ。 次の制約がある。 $n \leq 1$ の場合、$lrwork \geq 1$ *job* = 'N' で $n > 1$ の場合、$lrwork \geq n$ *job* = 'V' で $n > 1$ の場合、$lrwork \geq 3n^2 + (4+2k) * n + 1$ (ただし、k は、$2^k \geq n$ を満足する最小の整数)
iwork	INTEGER ワークスペース配列 DIMENSION は *liwork* 以上。

liwork INTEGER。配列 *iwork* のサイズ。
次の制約がある。
job = 'N' または $n \le 1$ の場合、*liwork* ≥ 1
job = 'V' で $n > 1$ の場合、*liwork* $\ge 5n+2$

出力パラメータ

w REAL (chbevd の場合)
DOUBLE PRECISION (zhbevd の場合)
配列 DIMENSION は max(1, *n*) 以上。
info = 0 の場合、マトリックス *A* の固有値が昇順に格納される。*info* も参照のこと。

z COMPLEX (chbevd の場合)
DOUBLE COMPLEX (zhbevd の場合)
配列 DIMENSION は *(ldz, *)*。*z* の第 2 次元のサイズは、次の値でなければならない。
job = 'N' の場合は、1 以上。
job = 'V' の場合は、max(1, *n*) 以上。
job = 'V' の場合、この配列は、*A* の固有ベクトルの入ったユニタリ・マトリックス *Z* で上書きされる。すなわち、固有値 *w(i)* に対応する固有ベクトルが、*z* の *i* 番目の列に格納される。
job = 'N' の場合、*z* は参照されない。

ab 終了時に、この配列は、三重対角形式に縮退したときに生成された値で上書きされる。

work(1) 終了時に、*lwork* > 0 ならば、*lwork* に必要な最小値が *work(1)* の実部に返される。

rwork(1) 終了時に、*lrwork* > 0 ならば、*lrwork* に必要な最小値が *rwork(1)* に返される。

iwork(1) 終了時に、*liwork* > 0 ならば、*liwork* に必要な最小値が *iwork(1)* に返される。

info INTEGER
info = 0 の場合は、正常終了している。
info = *i* の場合は、このアルゴリズムが収束に失敗した。*i* は、中間の三重対角形式で、ゼロに収束しなかった成分の個数を表す。

info = *-i* の場合は、*i* 番目のパラメータ値が不正である。

アプリケーション・ノート

計算された固有値と固有ベクトルは、正確には、$\|E\|_2 = O(\varepsilon)\,\|T\|_2$ であるようなマトリックス $T + E$ (ただし、ε はマシン精度) のものである。

このルーチンの実数版は、?sbevd である。

フル形式のマトリックスに対する ?heevd と、パックド形式のマトリックスに関する ?hpevd も参照のこと。

?sbevx

バンド形式の実対称マトリックスの
固有値を選択的に計算し、さらに
オプションで、固有ベクトルも計算する。

```
call ssbevx ( jobz, range, uplo, n, kd, ab, ldab, q, ldq, vl, vu, il,
              iu, abstol, m, w, z, ldz, work, iwork, ifail, info)
call dsbevx ( jobz, range, uplo, n, kd, ab, ldab, q, ldq, vl, vu, il,
              iu, abstol, m, w, z, ldz, work, iwork, ifail, info)
```

説明

このルーチンは、バンド形式の実対称マトリックス A の固有値を選択的に計算し、さらにオプションで、固有ベクトルも計算する。計算する固有値と固有ベクトルは、目的の固有値の範囲、または固有値のインデックスの範囲で選択できる。

入力パラメータ

jobz CHARACTER*1。'N' または 'V' でなければならない。
job ='N' の場合、固有値のみ計算する。
job ='V' の場合、固有値と固有ベクトルを計算する。

range CHARACTER*1。'A'、'V'、'I' のどれかでなければならない。
range ='A' の場合、固有値をすべて計算する。
range ='V' の場合、半開区間内の固有値 λ_i ($vl < \lambda_i \le vu$) を計算する。
range ='I' の場合、インデックスが *il* ～ *iu* の固有値を計算する。

uplo CHARACTER*1。'U' または 'L' でなければならない。
uplo ='U' の場合、*ab* に A の上三角部分を格納する。
uplo ='L' の場合、*ab* に A の下三角部分を格納する。

n INTEGER。マトリックス A の次数 ($n \ge 0$)。

kd INTEGER。A の優対角線または劣対角線の本数 ($kd \ge 0$)。

ab, work

REAL (ssbevx の場合)
DOUBLE PRECISION (dsbevx の場合)
配列 :
ab (*ldab*, *) は、対称マトリックス *A* の上または下 (*uplo* の指定に従う) 三角部分をバンド形式で格納する配列。
*ab*の第2次元のサイズはmax(1, *n*)以上でなければならない。
work (*) は、ワークスペース配列。
work のサイズは、max(1, 7*n*) 以上でなければならない。

ldab

INTEGER。*ab* のリーディング・ディメンジョンのサイズ。*kd* + 1 以上でなければならない。

vl, vu

REAL (ssbevx の場合)
DOUBLE PRECISION (dsbevx の場合)
range = 'V' の場合、固有値を探す区間の下限値と上限値。
制約 : *vl* < *vu*
range = 'A' または 'I' の場合、*vl* と *vu* は参照されない。

il, iu

INTEGER
range = 'I' の場合、求める固有値の最小インデックスと最大インデックス (インデックスは昇順)。
制約 : *n* > 0 の場合は、1 ≤ *il* ≤ *iu* ≤ *n*。*n* = 0 の場合は、*il* = 1 かつ *iu* = 0
range = 'A' または 'V' の場合、*il* と *iu* は参照されない。

abstol

REAL (chpevx の場合)
DOUBLE PRECISION (zhpevx の場合)
各固有値に許される絶対誤差許容値。誤差許容値の詳細は、「アプリケーション・ノート」を参照。

ldq, ldz

INTEGER。出力配列 *q* と *z* のリーディング・ディメンジョンのサイズ。次の制約がある。
ldq ≥ 1, *ldz* ≥ 1
jobz = 'V' の場合は、*ldq* ≥ max(1, *n*)、
ldz ≥ max(1, *n*)

iwork

INTEGER
ワークスペース配列 DIMENSION は max(1, 5*n*) 以上。

出力パラメータ

m

INTEGER。見つかった固有値の総数、$0 \le m \le n$。
range = 'A' の場合は *m* = *n*、*range* = 'I' の場合は
m = *iu*-*il*+1

w, *z*

REAL (ssbevx の場合)
DOUBLE PRECISION (dsbevx の場合)
配列 :
w(*)、サイズは max(1, *n*) 以上。
w の先頭 *m* 成分に、マトリックス *A* の選択された固有値が昇順に格納される。

z(*ldz*, *)、*z* の第 2 次元のサイズは max(1, *m*) 以上でなければならない。
jobz = 'V' の場合、*info* = 0 ならば、*z* の先頭 *m* 列に、マトリックス *A* の選択された固有値に対応する正規直交固有ベクトルが格納される。すなわち、*w*(*i*) に対応する固有ベクトルが、*z* の *i* 番目の列に格納される。収束に失敗した固有ベクトルは、それに対応する *z* の列に、最終的な近似固有ベクトルが格納され、この固有ベクトルのインデックスが *ifail* に返される。
jobz = 'N' の場合、*z* は参照されない。
注 : 配列 *z* には、max(1, *m*) 以上の列が提供されなければならない。*range* = 'V' の場合は、*m* の正確な値が事前にわからないため、上限値を使用する必要がある。

ab

終了時に、この配列は、三重対角形式に縮退したときに生成された値で上書きされる。*uplo* = 'U' の場合、三重対角マトリックス *T* の第一優対角成分と対角成分が、*ab* の 2 つの行 (*kd* と *kd*+1) に返される。
uplo = 'L' の場合、*T* の対角成分と第一劣対角成分が、*ab* の先頭 2 行に返される。

ifail

INTEGER
配列 DIMENSION は max(1, *n*) 以上。
jobz = 'V' の場合、*info* = 0 ならば、*ifail* の先頭 *m* 成分はゼロになる。*info* > 0 ならば、収束に失敗した固有ベクトルのインデックスが *ifail* に格納される。

jobz = 'N' の場合、*ifail* は参照されない。

info INTEGER

info = 0 の場合は、正常終了している。
info = -*i* の場合は、*i* 番目のパラメータ値が不正である。
info = *i* の場合、*i* 番目の固有ベクトルが収束に失敗した。その固有ベクトルのインデックスが配列 *ifail* に格納される。

アプリケーション・ノート

近似的な固有値でも、幅が *abstol*+ε*max(|a|, |b|) 以下の区間 [a, b] 内に納まる場合は、収束したと見なされる (ε はマシン精度)。*abstol* がゼロまたは負の場合は、かわりに $\varepsilon*\|T\|_1$ を使用する。*T* は、*A* を三重対角形式に縮退して得られた三重対角マトリックスである。
固有値が最も正確に計算されるのは、*abstol* がゼロでなく、アンダーフローのしきい値の 2 倍、2*?lamch('S') に設定されたときである。このルーチンで *info* > 0 が返された場合は、固有ベクトルのいくつかが収束に失敗したことを意味するので、*abstol* を 2*?lamch('S') に設定して、やり直してみる。

?hbevx

バンド形式のエルミート・マトリックスの
固有値を選択的に計算し、さらに
オプションで、固有ベクトルも計算する。

```
call chbevx ( jobz, range, uplo, n, kd, ab, ldab, q, ldq, vl, vu, il,
              iu, abstol, m, w, z, ldz, work, rwork, iwork, ifail,
              info)
call zhbevx ( jobz, range, uplo, n, kd, ab, ldab, q, ldq, vl, vu, il,
              iu, abstol, m, w, z, ldz, work, rwork, iwork, ifail,
              info)
```

説明

このルーチンは、バンド形式の複素エルミート・マトリックス A の固有値を選択的に計算し、さらにオプションで、固有ベクトルも計算する。計算する固有値と固有ベクトルは、目的の固有値の範囲、または固有値のインデックスの範囲で選択できる。

入力パラメータ

jobz CHARACTER*1。'N' または 'V' でなければならない。
job ='N' の場合、固有値のみ計算する。
job ='V' の場合、固有値と固有ベクトルを計算する。

range CHARACTER*1。'A'、'V'、'I' のどれかでなければならない。
range ='A' の場合、固有値をすべて計算する。
range ='V' の場合、半開区間内の固有値 λ_i ($vl < \lambda_i \le vu$) を計算する。
range ='I' の場合、インデックスが *il* ～ *iu* の固有値を計算する。

uplo CHARACTER*1。'U' または 'L' でなければならない。
uplo ='U' の場合、*ab* に A の上三角部分を格納する。
uplo ='L' の場合、*ab* に A の下三角部分を格納する。

n INTEGER。マトリックス A の次数 ($n \ge 0$)。

kd INTEGER。A の優対角線または劣対角線の本数 ($kd \ge 0$)。

ab, work

COMPLEX (chbevx の場合)
DOUBLE COMPLEX (zhbevx の場合)
配列 :
ab (*ldab*, *) は、エルミート・マトリックス *A* の上または下 (*uplo* の指定に従う) 三角部分をバンド形式で格納する配列。
*ab*の第2次元のサイズはmax(1, *n*)以上でなければならない。
work (*) は、ワークスペース配列。
work のサイズは、max(1, *n*) 以上でなければならない。

ldab

INTEGER。*ab* のリーディング・ディメンジョンのサイズ。*kd* + 1 以上でなければならない。

vl, vu

REAL (chbevx の場合)
DOUBLE PRECISION (zhbevx の場合)
range = 'V' の場合、固有値を探す区間の下限値と上限値。
制約 : *vl* < *vu*
range = 'A' または 'I' の場合、*vl* と *vu* は参照されない。

il, iu

INTEGER
range = 'I' の場合、求める固有値の最小インデックスと最大インデックス (インデックスは昇順)。
制約 : *n* > 0 の場合は、1 ≤ *il* ≤ *iu* ≤ *n*。*n* = 0 の場合は、*il* = 1 かつ *iu* = 0
range = 'A' または 'V' の場合、*il* と *iu* は参照されない。

abstol

REAL (chbevx の場合)
DOUBLE PRECISION (zhbevx の場合)
各固有値に許される絶対誤差許容値。誤差許容値の詳細は、「アプリケーション・ノート」を参照。

ldq, ldz

INTEGER。出力配列 *q* と *z* のリーディング・ディメンジョンのサイズ。次の制約がある。
ldq ≥ 1, *ldz* ≥ 1
jobz = 'V' の場合は、*ldq* ≥ max(1, *n*)、
ldz ≥ max(1, *n*)

rwork REAL (chbevx の場合)
DOUBLE PRECISION (zhbevx の場合)
ワークスペース配列 DIMENSION は max(1, 7*n*) 以上。

iwork INTEGER
ワークスペース配列 DIMENSION は max(1, 5*n*) 以上。

出力パラメータ

m INTEGER。見つかった固有値の総数、$0 \le m \le n$。
range ='A' の場合は *m* = *n*、*range* ='I' の場合は *m* = *iu*-*il*+1

w REAL (chbevx の場合)
DOUBLE PRECISION (zhbevx の場合)
配列 DIMENSION は max(1, *n*) 以上。
先頭の *m* 個の成分に、マトリックス *A* の選択された固有値が昇順に格納される。

z COMPLEX (chbevx の場合)
DOUBLE COMPLEX (zhbevx の場合)
配列 *z(ldz, *)*、*z* の第 2 次元のサイズは max(1, *m*) 以上でなければならない。
jobz ='V' の場合、*info* = 0 ならば、*z* の先頭 *m* 列に、マトリックス *A* の選択された固有値に対応する正規直交固有ベクトルが格納される。すなわち、*w*(*i*) に対応する固有ベクトルが、*z* の *i* 番目の列に格納される。収束に失敗した固有ベクトルは、それに対応する *z* の列に、最終的な近似固有ベクトルが格納され、この固有ベクトルのインデックスが *ifail* に返される。
jobz ='N' の場合、*z* は参照されない。
注 : 配列 *z* には、max(1, *m*) 以上の列が提供されなければならない。*range* ='V' の場合は、*m* の正確な値が事前にわからないため、上限値を使用する必要がある。

ab 終了時に、この配列は、三重対角形式に縮退したときに生成された値で上書きされる。*uplo* ='U' の場合、三重対角マトリックス *T* の第一優対角成分と対角成分が、*ab* の 2 つの行 (*kd* と *kd*+1) に返される。*uplo* ='L' の場合、*T* の対角成分と第一劣対角成分が、*ab* の先頭 2 行に返される。

ifail INTEGER

配列 DIMENSION は max(1, *n*) 以上。

jobz = 'V' の場合、*info* = 0 ならば、*ifail* の先頭 *m* 成分はゼロになる。*info* > 0 ならば、収束に失敗した固有ベクトルのインデックスが *ifail* に格納される。

jobz = 'N' の場合、*ifail* は参照されない。

info INTEGER

info = 0 の場合は、正常終了している。

info = -*i* の場合は、*i* 番目のパラメータ値が不正である。

info = *i* の場合、*i* 番目の固有ベクトルが収束に失敗した。その固有ベクトルのインデックスが配列 *ifail* に格納される。

アプリケーション・ノート

近似的な固有値でも、幅が *abstol*+ε*max(|a|, |b|) 以下の区間 [a, b] 内に納まる場合は、収束したと見なされる (ε はマシン精度)。*abstol* がゼロまたは負の場合は、かわりに $\varepsilon*\|T\|_1$ を使用する。*T* は、*A* を三重対角形式に縮退して得られた三重対角マトリックスである。

固有値が最も正確に計算されるのは、*abstol* がゼロでなく、アンダーフローのしきい値の 2 倍、2*?lamch('S') に設定されたときである。このルーチンで *info* > 0 が返された場合は、固有ベクトルのいくつかが収束に失敗したことを意味するので、*abstol* を 2*?lamch('S') に設定して、やり直してみる。

?stev

実対称三重対角マトリックスの固有値を
すべて計算し、さらにオプションで、
固有ベクトルも計算する。

```
call sstev (jobz, n, d, e, z, ldz, work, info)
call dstev (jobz, n, d, e, z, ldz, work, info)
```

説明

このルーチンは、実対称三重対角マトリックス A の固有値をすべて計算し、さらにオプションで、固有ベクトルも計算する。

入力パラメータ

jobz
CHARACTER*1。'N' または 'V' でなければならない。
jobz ='N' の場合、固有値のみ計算する。
jobz ='V' の場合、固有値と固有ベクトルを計算する。

n
INTEGER。マトリックス A の次数 ($n \geq 0$)。

d, e, work
REAL (sstev の場合)
DOUBLE PRECISION (dstev の場合)
配列 :
d(*) には、三重対角マトリックス A の *n* 個の対角成分を格納する。
d のサイズは、max(1, *n*) 以上でなければならない。
e(*) には、三重対角マトリックス A の *n*-1 個の劣対角成分を格納する。
e のサイズは、max(1, *n*) 以上でなければならない。この配列の *n* 番目の成分は、ワークスペースとして使用される。
work(*) は、ワークスペース配列。
work のサイズは、max(1, 2*n*-2) 以上でなければならない。
jobz ='N' の場合、*work* は参照されない。

ldz INTEGER。出力配列 *z* のリーディング・ディメンジョンのサイズ。
ldz ≥ 1。*jobz* = 'V' の場合は、*ldz* ≥ max(1, *n*)

出力パラメータ

d 終了時に、*info* = 0 の場合、マトリックス *A* の固有値が昇順に格納される。

z REAL (sstev の場合)
DOUBLE PRECISION (dstev の場合)
配列 DIMENSION は (*ldz*, *)
z の第 2 次元のサイズは max(1, *n*) 以上でなければならない。
jobz = 'V' の場合、*info* = 0 ならば、マトリックス *A* の正規直交固有ベクトルが *z* に格納される。すなわち、*d*(*i*) に対応する固有ベクトルが、*z* の *i* 番目の列に格納される。
job = 'N' の場合、*z* は参照されない。

e 終了時に、この配列は、中間結果で上書きされる。

info INTEGER
info = 0 の場合は、正常終了している。
info = -*i* の場合は、*i* 番目のパラメータ値が不正である。
info = *i* の場合は、このアルゴリズムが収束に失敗した。
e の *i* 個の成分が、ゼロに収束しなかった。

?stevd

分割統治アルゴリズムを使用して、実対称三重対角マトリックスの固有値をすべて計算し、さらにオプションで、固有ベクトルも計算する。

```
call sstevd (job, n, d, e, z, ldz, work, lwork, iwork, liwork, info)
call dstevd (job, n, d, e, z, ldz, work, lwork, iwork, liwork, info)
```

説明

このルーチンは、実対称三重対角マトリックス T の固有値をすべて計算し、さらにオプションで、固有ベクトルも計算する。言い換えると、T のスペクトル因子分解 $T = Z\Lambda Z^T$ が求められる。
Λ は対角成分が固有値 λ_i であるような対角マトリックス、Z は各列が固有ベクトル z_i であるような直交マトリックスである。したがって、次のように表現できる。

$$Tz_i = \lambda_i z_i \text{ (ただし、} i = 1, 2, ..., n)$$

このルーチンは、固有ベクトルも要求された場合は、分割統治アルゴリズムを使用して、固有値と固有ベクトルを計算する。ただし、固有値のみ要求された場合は、Pal-Walker-Kahan 変形版の *QL* または *QR* アルゴリズムを使用して、固有値を計算する。

このルーチンの複素数版はない。

入力パラメータ

job	CHARACTER*1。'N' または 'V' でなければならない。 *job* = 'N' の場合、固有値のみ計算する。 *job* = 'V' の場合、固有値と固有ベクトルを計算する。
n	INTEGER。マトリックス T の次数 ($n \geq 0$)
d, e, work	REAL (sstevd の場合) DOUBLE PRECISION (dstevd の場合) 配列 :

d(*) には、三重対角マトリックス T の n 個の対角成分を格納する。
d のサイズは、max(1, n) 以上でなければならない。
e(*) には、T の n-1 個の非対角成分を格納する。
e のサイズは、max(1, n) 以上でなければならない。この配列の n 番目の成分は、ワークスペースとして使用される。
work(*) は、ワークスペース配列。
work のサイズは、lwork 以上でなければならない。

ldz
INTEGER。出力配列 z のリーディング・ディメンジョンのサイズ。
次の制約がある。
job='N' の場合、ldz ≥ 1
job='V' の場合、ldz ≥ max(1, n)

lwork
INTEGER。配列 work のサイズ。
次の制約がある。
job='N' または n ≤ 1 の場合、lwork ≥ 1
job='V' で n > 1 の場合、lwork ≥ $2n^2$+(3+2k)*n+1
(ただし、k は、2^k ≥ n を満足する最小の整数)

iwork
INTEGER
ワークスペース配列 DIMENSION は liwork 以上。

liwork
INTEGER。配列 iwork のサイズ。
次の制約がある。
job='N' または n ≤ 1 の場合、liwork ≥ 1
job='V' で n > 1 の場合、liwork ≥ 5n+2

出力パラメータ

d
終了時に、info = 0 の場合、マトリックス T の固有値が昇順に格納される。
info も参照のこと。

z
REAL (sstevd の場合)
DOUBLE PRECISION (dstevd の場合)
配列 DIMENSION は (ldz, *)
z の第 2 次元のサイズは、次の値でなければならない。
job='N' の場合は、1 以上。
job='V' の場合は、max(1, n) 以上。

	job = 'V' の場合、この配列は、*T* の固有ベクトルの入った直交マトリックス *Z* で上書きされる。*job* = 'N' の場合、*z* は参照されない。
e	終了時に、この配列は、中間結果で上書きされる。
work(1)	終了時に、*lwork* > 0 ならば、*lwork* に必要な最小値が *work(1)* に返される。
iwork(1)	終了時に、*liwork* > 0 ならば、*liwork* に必要な最小値が *iwork(1)* に返される。
info	INTEGER *info* = 0 の場合は、正常終了している。 *info* = *i* の場合は、このアルゴリズムが収束に失敗した。*i* は、中間の三重対角形式で、ゼロに収束しなかった成分の個数を表す。 *info* = -*i* の場合は、*i* 番目のパラメータ値が不正である。

アプリケーション・ノート

計算された固有値と固有ベクトルは、正確には、$|E\|_2 = O(\varepsilon)\,\|T\|_2$ であるようなマトリックス $T + E$ (ただし、ε はマシン精度) のものである。

正確な固有値を λ_i とし、それに対する計算値を μ_i とすると、その関係は次のようになる。

$$|\mu_i - \lambda_i| \le c(n)\varepsilon\,\|T\|_2$$

$c(n)$ は、n の漸増関数である。

これに対応する正確な固有ベクトルを z_i とし、それに対する計算値を w_i とすると、その 2 つの角度 $\theta(z_i, w_i)$ は次のように表される。

$$\theta(z_i, w_i) \le c(n)\varepsilon\,\|T\|_2 / \min_{i \ne j}|\lambda_i - \lambda_j|$$

このように、計算された固有ベクトルの精度は、それが対応する固有値と、他のすべての固有値との隔たりに依存するのがわかる。

?stevx

実対称三重対角マトリックスの固有値と
固有ベクトルを選択的に計算する。

```
call sstevx ( jobz, range, n, d, e, vl, vu, il, iu, abstol, m, w, z,
              ldz, work, iwork, ifail, info)
call dstevx ( jobz, range, n, d, e, vl, vu, il, iu, abstol, m, w, z,
              ldz, work, iwork, ifail, info)
```

説明

このルーチンは、実対称三重対角マトリックス A の固有値を選択的に計算し、さらにオプションで、固有ベクトルも計算する。計算する固有値と固有ベクトルは、目的の固有値の範囲、または固有値のインデックスの範囲で選択できる。

入力パラメータ

jobz CHARACTER*1。'N' または 'V' でなければならない。
job ='N' の場合、固有値のみ計算する。
job ='V' の場合、固有値と固有ベクトルを計算する。

range CHARACTER*1。'A'、'V'、'I' のどれかでなければならない。
range ='A' の場合、固有値をすべて計算する。
range ='V' の場合、半開区間内の固有値 λ_i ($vl < \lambda_i \le vu$) を計算する。
range ='I' の場合、インデックスが *il* ～ *iu* の固有値を計算する。

n INTEGER。マトリックス A の次数 ($n \ge 0$)

d, e, work REAL (sstevx の場合)
DOUBLE PRECISION (dstevx の場合)
配列 :

d(*) には、三重対角マトリックス *A* の *n* 個の対角成分を格納する。
d のサイズは、max(1, *n*) 以上でなければならない。
e(*) には、*A* の *n*-1 個の劣対角成分を格納する。
e のサイズは、max(1, *n*) 以上でなければならない。この配列の *n* 番目の成分は、ワークスペースとして使用される。
work(*) は、ワークスペース配列。
work のサイズは、max(1, 5*n*) 以上でなければならない。

vl, vu
REAL (sstevx の場合)
DOUBLE PRECISION (dstevx の場合)
range = 'V' の場合、固有値を探す区間の下限値と上限値。
制約 : *vl* < *vu*
range = 'A' または 'I' の場合、*vl* と *vu* は参照されない。

il, iu
INTEGER
range = 'I' の場合、求める固有値の最小インデックスと最大インデックス (インデックスは昇順)。
制約 : *n* > 0 の場合は、1 ≤ *il* ≤ *iu* ≤ *n*。*n* = 0 の場合は、*il* = 1 かつ *iu* = 0
range = 'A' または 'V' の場合、*il* と *iu* は参照されない。

abstol
REAL (sstevx の場合)
DOUBLE PRECISION (dstevx の場合)
各固有値に許される絶対誤差許容値。誤差許容値の詳細は、「アプリケーション・ノート」を参照。

ldz
INTEGER。出力配列 *z* のリーディング・ディメンジョンのサイズ。
ldz ≥ 1。*jobz* = 'V' の場合は、*ldz* ≥ max(1, *n*)

iwork
INTEGER
ワークスペース配列 DIMENSION は max(1, 5*n*) 以上。

出力パラメータ

m
INTEGER。見つかった固有値の総数、$0 \le m \le n$。
range = 'A' の場合は *m* = *n*、*range* = 'I' の場合は *m* = *iu*-*il*+1

w, z
REAL (sstevx の場合)
DOUBLE PRECISION (dstevx の場合)
配列 :
w(*)、DIMENSION は max(1, *n*) 以上。
w の先頭 *m* 成分に、マトリックス *A* の選択された固有値が昇順に格納される。

*z(ldz, *)*、*z* の第 2 次元のサイズは max(1, *m*) 以上でなければならない。
jobz = 'V' の場合、*info* = 0 ならば、*z* の先頭 *m* 列に、マトリックス *A* の選択された固有値に対応する正規直交固有ベクトルが格納される。すなわち、*w*(*i*) に対応する固有ベクトルが、*z* の *i* 番目の列に格納される。収束に失敗した固有ベクトルは、それに対応する *z* の列に、最終的な近似固有ベクトルが格納され、この固有ベクトルのインデックスが *ifail* に返される。
jobz = 'N' の場合、*z* は参照されない。
注 : 配列 *z* には、max(1, *m*) 以上の列が提供されなければならない。*range* = 'V' の場合は、*m* の正確な値が事前にわからないため、上限値を使用する必要がある。

d, e
終了時に、この 2 つの配列には、固有値を計算するときのオーバーフローまたはアンダーフローを避けるために、選択された定係数が掛けられることがある。

ifail
INTEGER
配列 DIMENSION は max(1, *n*) 以上。
jobz = 'V' の場合、*info* = 0 ならば、*ifail* の先頭 *m* 成分はゼロになる。*info* > 0 ならば、収束に失敗した固有ベクトルのインデックスが *ifail* に格納される。
jobz = 'N' の場合、*ifail* は参照されない。

info INTEGER

info = 0 の場合は、正常終了している。

info = *-i* の場合は、*i* 番目のパラメータ値が不正である。

info = *i* の場合、*i* 番目の固有ベクトルが収束に失敗した。その固有ベクトルのインデックスが配列 *ifail* に格納される。

アプリケーション・ノート

近似的な固有値でも、幅が *abstol*+ε*max(|a|, |b|) 以下の区間 [a, b] 内に納まる場合は、収束したと見なされる (ε はマシン精度)。*abstol* がゼロまたは負の場合は、かわりに $\varepsilon*\|A\|_1$ を使用する。

固有値が最も正確に計算されるのは、*abstol* がゼロでなく、アンダーフローのしきい値の 2 倍、2*?lamch('S') に設定されたときである。このルーチンで *info* > 0 が返された場合は、固有ベクトルのいくつかが収束に失敗したことを意味するので、*abstol* を 2*?lamch('S') に設定して、やり直してみる。

?stevr

「比較的安定な表現」を使用して、実対称三重対角マトリックスの固有値を選択的に計算し、さらにオプションで、固有ベクトルも計算する。

```
call sstevr ( jobz, range, n, d, e, vl, vu, il, iu, abstol, m, w, z,
              ldz, isuppz, work, lwork, iwork, liwork, info)
call dstevr ( jobz, range, n, d, e, vl, vu, il, iu, abstol, m, w, z,
              ldz, isuppz, work, lwork, iwork, liwork, info)
```

説明

このルーチンは、実対称三重対角マトリックス T の固有値を選択的に計算し、さらにオプションで、固有ベクトルも計算する。計算する固有値と固有ベクトルは、目的の固有値の範囲、または固有値のインデックスの範囲で選択できる。

可能であれば、?stevr は sstegr/dstegr を呼び出し、「比較的安定な表現」を使用して、固有スペクトルを計算する。?stegr は、固有値の計算には *dqds* アルゴリズムを使用するが、直交固有ベクトルは、種々の「良好な」LDL^T 表現 (「比較的安定な表現」とも呼ばれる) から計算する。グラム - シュミットの直交化はできる限り避ける。アルゴリズムの各ステップを詳しく説明すると次のようになる。T のまだ縮退されていない i 番目のブロックに対して、

(a) $L_i D_i L_i^T$ が比較的安定な表現になるように、$T - \sigma_i = L_i D_i L_i^T$ を計算する。

(b) *dqds* アルゴリズムによって、$L_i D_i L_i^T$ の固有値 λ_j を高い相対精度で計算する。

(c) 値が近い固有値からなるクラスタが存在する場合は、そのクラスタに近い σ_i を「選択」し、ステップ (a) に戻る。

(d) $L_i D_i L_i^T$ の近似的な固有値 λ_j に対する固有ベクトルを、階数表示のツイスト分解によって求める。

出力値に必要な精度は、入力パラメータ *abstol* で指定できる。

ルーチン ?stevr は、IEEE-754 の浮動小数点標準に準拠しているマシンにおいてフル・スペクトルが要求された場合は、sstegr/dstegr を呼び出す。IEEE-754 に準拠しないマシンの場合や、部分スペクトルが要求された場合は、sstebz/dstebz と sstein/dstein を呼び出す。

入力パラメータ

jobz
CHARACTER*1。'N' または 'V' でなければならない。
jobz ='N' の場合、固有値のみ計算する。
jobz ='V' の場合、固有値と固有ベクトルを計算する。

range
CHARACTER*1。'A'、'V'、'I' のどれかでなければならない。
range ='A' の場合、固有値をすべて計算する。
range ='V' の場合、半開区間内の固有値 λ_i ($vl < \lambda_i \le vu$) を計算する。
range ='I' の場合、インデックスが *il* ～ *iu* の固有値を計算する。
range ='V' または 'I' で、$iu-il < n-1$ の場合、sstebz/dstebz と sstein/dstein を呼び出す。

n
INTEGER。マトリックス T の次数 ($n \ge 0$)

d, e, work
REAL (sstevr の場合)
DOUBLE PRECISION (dstevr の場合)
配列 :
d(*) には、三重対角マトリックス T の *n* 個の対角成分を格納する。
d のサイズは、max(1, *n*) 以上でなければならない。
e(*) には、A の *n*-1 個の劣対角成分を格納する。
e のサイズは、max(1, *n*) 以上でなければならない。この配列の *n* 番目の成分は、ワークスペースとして使用される。
work(*lwork*) は、ワークスペース配列。

vl, vu	REAL (sstevr の場合) DOUBLE PRECISION (dstevr の場合) *range* = 'V' の場合、固有値を探す区間の下限値と上限値。 制約 : *vl* < *vu* *range* = 'A' または 'I' の場合、*vl* と *vu* は参照されない。
il, iu	INTEGER *range* = 'I' の場合、求める固有値の最小インデックスと最大インデックス (インデックスは昇順)。 制約 : $n > 0$ の場合は、$1 \le il \le iu \le n$。$n = 0$ の場合は、$il = 1$ かつ $iu = 0$ *range* = 'A' または 'V' の場合、*il* と *iu* は参照されない。
abstol	REAL (ssyevr の場合) DOUBLE PRECISION (dsyevr の場合) 各固有値と固有ベクトルに許される絶対誤差許容値。*jobz* = 'V' の場合、出力される固有値と固有ベクトルの誤差ノルムが *abstol* 以内になる。また、異なる固有ベクトル間のドット積も *abstol* 以内になる。*abstol* < $n\varepsilon\|T\|_1$ の場合は、$n\varepsilon\|T\|_1$ がかわりに使用される (ε はマシン精度)。固有値は、*abstol* とは無関係に、$\varepsilon\|T\|_1$ の精度で計算される。高い相対精度が必要な場合は、*abstol* を ?lamch('S') に設定する。
ldz	INTEGER。出力配列 *z* のリーディング・ディメンジョンのサイズ。 次の制約がある。 *jobz* = 'N' の場合は、$ldz \ge 1$ *jobz* = 'V' の場合は、$ldz \ge \max(1, n)$
lwork	INTEGER。配列 *work* のサイズ。 制約 : $lwork \ge \max(1, 20n)$
iwork	INTEGER ワークスペース配列 DIMENSION は *liwork*
liwork	INTEGER。配列 *iwork* のサイズ、$lwork \ge \max(1, 10n)$

出力パラメータ

m
INTEGER。見つかった固有値の総数、$0 \le m \le n$。
range = 'A' の場合は *m* = *n*、*range* = 'I' の場合は *m* = *iu*-*il*+1

w, z
REAL (sstevr の場合)
DOUBLE PRECISION (dstevr の場合)
配列 :
w(*)、DIMENSION は max(1, *n*) 以上。
w の先頭 *m* 成分に、マトリックス *T* の選択された固有値が昇順に格納される。

*z(ldz, *)*、*z* の第 2 次元のサイズは max(1, *m*) 以上でなければならない。
jobz = 'V' の場合、*info* = 0 ならば、*z* の先頭 *m* 列に、マトリックス *T* の選択された固有値に対応する正規直交固有ベクトルが格納される。すなわち、*w*(*i*) に対応する固有ベクトルが、*z* の *i* 番目の列に格納される。
jobz = 'N' の場合、*z* は参照されない。
注 : 配列 *z* には、max(1, *m*) 以上の列が提供されなければならない。*range* = 'V' の場合は、*m* の正確な値が事前にわからないため、上限値を使用する必要がある。

d, e
終了時に、この 2 つの配列には、固有値を計算するときのオーバーフローまたはアンダーフローを避けるために、選択された定係数が掛けられることがある。

isuppz
INTEGER
配列 DIMENSION は 2*max(1, *m*) 以上。
z に入っている固有ベクトルのサポート情報、すなわち *z* に入っている非ゼロ成分を示すインデックス。*i* 番目の固有ベクトルは、*isuppz*(2*i*-1) から *isuppz*(2*i*) までの成分のみ非ゼロである。
range = 'A'、または *range* = 'I' で *iu*-*il* = *n*-1 の場合にのみ有効である。

work(1)
終了時に、*info* = 0 ならば、*lwork* に必要な最小値が *work(1)* に返される。

iwork(1)	終了時に、*info* = 0 ならば、*liwork* に必要な最小値が *iwork(1)* に返される。
info	INTEGER *info* = 0 の場合は、正常終了している。 *info* = *-i* の場合は、*i* 番目のパラメータ値が不正である。 *info* = *i* の場合、内部エラーが発生した。

アプリケーション・ノート

通常の ?stegr の実行で NaN と無限大が発生するため、NaN と無限大が IEEE 標準のデフォルト方法以外で処理される環境では、浮動小数点例外によって異常終了する可能性がある。

非対称固有値問題

本節では、非対称固有値問題を解くために使用する LAPACK ドライバ・ルーチンについて説明する。これらの問題を解くために呼び出すことが可能な計算ルーチン も参照のこと。
表 5-12 で、ルーチンの詳細を説明する。

表 5-11 非対称固有値問題を解くためのドライバ・ルーチン

ルーチン名	実行する演算
?gees	一般マトリックスの固有値と Schur 因子分解を計算し、指定された固有値が Schur 形式の左上にくるように因子分解を順序付けする。
?geesx	一般マトリックスの固有値と Schur 因子分解を計算し、因子分解の順序付けと、条件数の逆数の計算を実行する。
?geev	一般マトリックスの固有値と左固有ベクトルおよび右固有ベクトルを計算する。
?geevx	あらかじめマトリックスを平衡化して一般マトリックスの固有値と左固有ベクトルおよび右固有ベクトルを計算し、固有値と右固有ベクトルに対する条件数の逆数を計算する。

?gees

一般マトリックスの固有値と Schur 因子分解を計算し、指定された固有値が Schur 形式の左上にくるように因子分解を順序付けする。

```
call sgees ( jobvs, sort, select, n, a, lda, sdim, wr, wi, vs, ldvs,
             work, lwork, bwork, info)
call dgees ( jobvs, sort, select, n, a, lda, sdim, wr, wi, vs, ldvs,
             work, lwork, bwork, info)
call cgees ( jobvs, sort, select, n, a, lda, sdim, w, vs, ldvs, work,
             lwork, rwork, bwork, info)
call zgees ( jobvs, sort, select, n, a, lda, sdim, w, vs, ldvs, work,
             lwork, rwork, bwork, info)
```

説明

このルーチンは、*n* × *n* の実数 / 複素数の非対称マトリックス *A* について、固有値、実数 Schur 形式 *T* を計算し、オプションで Schur ベクトル *Z* のマトリックスを計算する。これにより、Schur 因子分解 $A = ZTZ^H$ が提供される。

また、このルーチンは、指定された固有値が左上にくるように、実数 Schur/Schur 形式の対角上の固有値を順序付けることもできる。*Z* の先頭の列は、指定された固有値に対応する不変部分空間の直交基を形成する。

1 × 1 ブロックと 2 × 2 ブロックを持つ上準三角マトリックスの場合、実数マトリックスは、実数 Schur 形式となる。2 × 2 ブロックは、以下の形式で標準化される。

$$\begin{pmatrix} a & b \\ c & a \end{pmatrix}$$

$b*c<0$ である。このようなブロックの固有値は、$a \pm \sqrt{bc}$ となる。

複素マトリックスは、上三角の場合は Schur 形式になる。

入力パラメータ

jobvs
: CHARACTER*1。'N' または 'V' でなければならない。
jobvs ='N' の場合、Schur ベクトルは計算されない。
jobvs ='V' の場合、Schur ベクトルは計算される。

sort
: CHARACTER*1。'N' または 'S' でなければならない。
Schur 形式の対角上の固有値を順序付けるかどうかを指定する。
sort ='N' の場合、固有値は順序付けされない。
sort ='S' の場合、固有値は順序付けされる (*select* を参照)。

select
: 実数型の場合、2 つの REAL 引数の LOGICAL FUNCTION。
複素数型の場合、1 つの COMPLEX 引数の LOGICAL FUNCTION
select は、呼び出しサブルーチン内で EXTERNAL として宣言する必要がある。
sort ='S' の場合、*select* は、Schur 形式の左上に対してソートを実行する固有値を選択するために使用される。
sort ='N' の場合、*select* は参照されない。

実数型の場合 :
select(*wr*(j), *wi*(j)) が真の場合は、固有値 *wr*(j)+$\sqrt{-1}$ **wi*(j) が選択される。つまり、固有値の複素共役ペアの一方を選択すると、複素数の固有値が両方選択される。順序付けを行うと、複素数の固有値の値が変更される可能性がある (特に、固有値が悪い条件の場合)。そのため、順序付けを実行した後、選択した複素数の固有値は、*select*(*wr*(j), *wi*(j)) =.TRUE. を満たさないときがある。この場合、*info* には、*n*+2 が設定される (以下の *info* を参照)。
複素数型の場合 :
select(*w*(j)) が真の場合は、固有値 *w*(j) が選択される。

n — INTEGER。マトリックス *A* の次数 ($n \geq 0$)

a, work — REAL (sgees の場合)
DOUBLE PRECISION (dgees の場合)
COMPLEX (cgees の場合)
DOUBLE COMPLEX (zgees の場合)
配列 :
a(*lda*, *) は、$n \times n$ のマトリックス *A* を格納する配列である。
a の第 2 次元は、max(1, *n*) 以上でなければならない。
work(*lwork*) は、ワークスペース配列である。

lda — INTEGER。配列 *a* の第 1 次元。
max(1, *n*) 以上でなければならない。

ldvs — INTEGER。出力配列 *vs* のリーディング・ディメンジョン。
次の制約がある。
$ldvs \geq 1$
$ldvs \geq \max(1, n)$ (*jobvs* ='V' の場合)

lwork — INTEGER。配列 *work* の次元。
次の制約がある。
$lwork \geq \max(1, 3n)$ (実数型の場合)
$lwork \geq \max(1, 2n)$ (複素数型の場合)

rwork — REAL (cgees の場合)
DOUBLE PRECISION (zgees の場合)
ワークスペース配列 DIMENSION は、max(1, *n*) 以上。
複素数型でのみ使用される。

bwork LOGICAL。
ワークスペース配列 DIMENSION は、max(1, *n*) 以上。
sort ='N' の場合は参照されない。

出力パラメータ

a 終了時、この配列は、実数 Schur/Schur 形式 *T* で上書きされる。

sdim INTEGER
sort ='N' の場合、*sdim*= 0 である。
sort ='S' の場合、*sdim* は、*select* が真の固有値の数 (ソート後の) に等しくなる。
実数型の場合、どちらかの固有値について *select* が真の複素共役ペアは、2 としてカウントされるのに注意する必要がある。

wr, wi REAL (sgees の場合)
DOUBLE PRECISION (dgees の場合)
配列 DIMENSION はそれぞれ、max(1, *n*) 以上。
出力実数 Schur 形式 *T* の対角上に格納されるのと同じ順序で、計算された固有値の実数部と虚数部をそれぞれ格納する。固有値の複素共役ペアは、虚数部が正である固有値に続いて格納される。

w COMPLEX (cgees の場合)
DOUBLE COMPLEX (zgees の場合)
配列 DIMENSION は、max(1, *n*) 以上。
計算された固有値が格納される。一連の固有値は、出力 Schur 形式 *T* の対角上に格納されるのと同じ順序で格納する。

vs REAL (sgees の場合)
DOUBLE PRECISION (dgees の場合)
COMPLEX (cgees の場合)
DOUBLE COMPLEX (zgees の場合)
配列 *vs*(*ldvs*, *);*vs* の第 2 次元は、max(1, *n*) 以上でなければならない。

jobvs = 'V' の場合、vs には、Schur ベクトルの直交 / ユニタリ・マトリックス Z が格納される。

jobvs = 'N' の場合、vs は参照されない。

work(1) 終了時に info = 0 の場合、work(1) は、lwork の必要最小サイズを戻す。

info INTEGER

info = 0 の場合は、正常に終了している。

info = -i の場合は、i 番目のパラメータの値が不正である。

info = i、かつ
i ≤ n :

QR アルゴリズムが固有値をすべて計算できなかった。収束した固有値は、wr と wi (実数型の場合) または w (複素数型の場合) の 1:ilo-1 と i+1:n の成分に格納される。jobvs = 'V' の場合、vs には、*A* をその部分収束 Schur 形式に縮退させるマトリックスを格納する。

i = n+1 :

いくつかの固有値が接近しすぎて分離できなかったため、固有値を順序変更できなかった (非常に悪い条件が問題)。

i = n+2 :

順序変更後、丸め操作によりいくつかの複素数の固有値の値が変更されたため、Schur 形式の先頭の固有値が select = .TRUE. を満たさなくなった。これは、スケーリングによるアンダーフローによっても引き起こされる。

アプリケーション・ノート

配列 work に必要なワークスペースの大きさがわからない場合は、最初の実行で lwork を大きめの値に設定する。実行の終了時に work(1) を調べて、これ以降の実行にはその値を使用するとよい。

?geesx

一般マトリックスの固有値と Schur 因子分解を計算し、因子分解の順序付けと、条件数の逆数の計算を実行する。

```
call sgeesx(jobvs, sort, select, sense, n, a, lda, sdim, wr, wi, vs,
            ldvs, rconde, rcondv, work, lwork, iwork, liwork, bwork,
            info)
call dgeesx(jobvs, sort, select, sense, n, a, lda, sdim, wr, wi, vs,
            ldvs, rconde, rcondv, work, lwork, iwork, liwork, bwork,
            info)
call cgeesx(jobvs, sort, select, sense, n, a, lda, sdim, w, vs,
            ldvs, rconde, rcondv, work, lwork, rwork, bwork, info)
call zgeesx(jobvs, sort, select, sense, n, a, lda, sdim, w, vs,
            ldvs, rconde, rcondv, work, lwork, rwork, bwork, info)
```

説明

このルーチンは、`n` × `n` の実数 / 複素数の非対称マトリックス A について、固有値、実数 Schur/Schur 形式 T を計算し、オプションで Schur ベクトル Z のマトリックスを計算する。これにより、Schur 因子分解 $A=ZTZ^H$ が提供される。

このルーチンは、指定された固有値が左上にくるように、実数 Schur/Schur 形式の対角上の固有値を順序付けもできる。また、指定した固有値の平均に対する条件数の逆数 (`rconde`) を計算したり、指定した固有値に対応する右不変部分空間に対する条件数の逆数 (`rcondv`) も計算できる。Z の先頭の列は、この不変部分空間の直交基を形成する。

条件数の逆数 `rconde`、`rcondv` の詳細は、4.10 節の [*LUG*] を参照のこと (これらの大きさはそれぞれ、s と sep と呼ぶ)。

1 × 1 ブロックと 2 × 2 ブロックを持つ上準三角マトリックスの場合、実数マトリックスは、実数 Schur 形式となる。2 × 2 ブロックは、次の形式で標準化される。

$$\begin{pmatrix} a & b \\ c & a \end{pmatrix}$$

ここで、$b*c<0$ である。このようなブロックの固有値は、$a \pm \sqrt{bc}$ となる。

複素マトリックスは、上三角の場合は Schur 形式になる。

入力パラメータ

jobvs CHARACTER*1。'N' または 'V' でなければならない。
jobvs ='N' の場合、Schur ベクトルは計算されない。
jobvs ='V' の場合、Schur ベクトルは計算される。

sort CHARACTER*1。'N' または 'S' でなければならない。
Schur 形式の対角上の固有値を順序付けるかどうかを指定する。
sort ='N' の場合、固有値は順序付けされない。
sort ='S' の場合、固有値は順序付けされる (*select* を参照)。

select 実数型の場合、2 つの REAL 引数の LOGICAL FUNCTION。
複素数型の場合、1 つの *COMPLEX* 引数の LOGICAL FUNCTION
select は、呼び出しサブルーチン内で EXTERNAL として宣言する必要がある。
sort ='S' の場合、*select* は、Schur 形式の左上に対してソートを実行する固有値を選択するために使用される。
sort ='N' の場合、*select* は参照されない。
実数型の場合 :
select(*wr*(j), *wi*(j)) が真の場合は、固有値 *wr*(j)+$\sqrt{-1}$ **wi*(j) が選択される。つまり、固有値の複素共役ペアの一方を選択すると、複素数の固有値が両方選択される。順序付けを行うと、複素数の固有値の値が変更される可能性がある (特に、固有値が悪い条件の場合)。そのため、順序付けを実行した後、選択した複素数の固有値は、*select*(*wr*(j), *wi*(j)) =.TRUE. を満たさないときがある。この場合、*info* には、*n*+2 が設定される (以下の *info* を参照)。
複素数型の場合 :
select(*w*(j)) が真の場合は、固有値 *w*(j) が選択される。

sense	CHARACTER*1。'N'、'E'、'V'、または 'B' でなければならない。 どの条件数の逆数を計算するのかを決定する。 *sense* ='N' の場合、計算は実行されない。 *sense* ='E' の場合、指定された固有値の平均についてのみ計算が実行される。 *sense* ='V' の場合、指定された右不変部分空間についてのみ計算が実行される。 *sense* ='B' の場合、両方について計算が実行される。 *sense* が 'E'、'V'、または 'B' の場合、*sort* は 'S' に等しくなければならない。
n	INTEGER。マトリックス A の次数 ($n \geq 0$)
a, work	REAL (sgeesx の場合) DOUBLE PRECISION (dgeesx の場合) COMPLEX (cgeesx の場合) DOUBLE COMPLEX (zgeesx の場合) 配列 : *a(lda, *)* は、$n \times n$ のマトリックス A を格納する配列である。 *a* の第 2 次元は、max(1, *n*) 以上でなければならない。 *work(lwork)* は、ワークスペース配列である。
lda	INTEGER。配列 *a* の第 1 次元。 max(1, *n*) 以上でなければならない。
ldvs	INTEGER。出力配列 *vs* のリーディング・ディメンジョン。 次の制約がある。 $ldvs \geq 1$ $ldvs \geq \max(1, n)$ (*jobvs* ='V' の場合)
lwork	INTEGER。配列 *work* の次元。 次の制約がある。 $lwork \geq \max(1, 3n)$ (実数型の場合) $lwork \geq \max(1, 2n)$ (複素数型の場合) また、*sense* ='E'、'V'、または 'B' の場合、次のようになる。 $lwork \geq n+2*sdim*(n-sdim)$ (実数型の場合) $lwork \geq 2*sdim*(n-sdim)$ (複素数型の場合)

sdim は、このルーチンで計算される指定された固有値の数である。また、2**sdim**(*n*-*sdim*) ≤ *n***n*/2 である。
性能を上げるには、一般に *lwork* を大きくする必要がある。

iwork INTEGER
ワークスペース配列 DIMENSION (*liwork*)。実数型でのみ使用される。
sense ='N' または 'E' の場合は参照されない。

liwork INTEGER。配列 *iwork* の次元。実数型でのみ使用される。
次の制約がある。
liwork ≥ 1
sense ='V' または 'B' の場合、
liwork ≥ *sdim**(*n*-*sdim*) である。

rwork REAL (cgeesx の場合)
DOUBLE PRECISION (zgeesx の場合)
ワークスペース配列 DIMENSION は、max(1, *n*) 以上。
複素数型でのみ使用される。

bwork LOGICAL。
ワークスペース配列 DIMENSION は、max(1, *n*) 以上。
sort ='N' の場合は参照されない。

出力パラメータ

a 終了時、この配列は、実数 Schur/Schur 形式 *T* で上書きされる。

sdim INTEGER
sort ='N' の場合、*sdim*= 0 である。
sort ='S' の場合、*sdim* は、*select* が真の固有値の数 (ソート後の) に等しくなる。
実数型の場合、どちらかの固有値について *select* が真の複素共役ペアは、2 としてカウントされるのに注意する必要がある。

wr, wi REAL (sgeesx の場合)
DOUBLE PRECISION (dgeesx の場合)
配列 DIMENSION はそれぞれ、max(1, *n*) 以上。
計算された固有値の実数部と虚数部は、出力実数 Schur 形式 *T* の対角上に格納されるのと同じ順序でそれぞれ格納される。

固有値の複素共役ペアは、虚数部が正である固有値に続いて格納される。

w

COMPLEX (cgeesx の場合)
DOUBLE COMPLEX (zgeesx の場合)
配列 DIMENSION は、max(1, *n*) 以上。
計算された固有値が格納される。固有値は、出力 Schur 形式 *T* の対角上に格納されるのと同じ順序で格納される。

vs

REAL (sgeesx の場合)
DOUBLE PRECISION (dgeesx の場合)
COMPLEX (cgeesx の場合)
DOUBLE COMPLEX (zgeesx の場合)
配列 *vs*(*ldvs*, *);*vs* の第 2 次元は、max(1, *n*) 以上でなければならない。

jobvs ='V' の場合、*vs* には、Schur ベクトルの直交 / ユニタリ・マトリックス Z が格納される。
jobvs ='N' の場合、*vs* は参照されない。

rconde, rcondv

REAL (単精度型の場合)
DOUBLE PRECISION (倍精度型の場合)
sense ='E' または 'B' の場合、*rconde* には、指定した固有値の平均に対する条件数の逆数が格納される。
sense ='N' または 'V' の場合、*rconde* は参照されない。

sense ='V' または 'B' の場合、*rcondv* には、指定した右不変部分空間に対する条件数の逆数が格納される。*sense* ='N' または 'E' の場合、*rcondv* は参照されない。

work(1)

終了時に *info* = 0 の場合、*work(1)* は、*lwork* の必要最小サイズを戻す。

info

INTEGER
info = 0 の場合は、正常に終了している。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。

info = *i*、かつ
i ≤ *n* :

QR アルゴリズムが固有値をすべて計算できなかった。収束した固有値は、*wr* と *wi* (実数型の場合) または *w*(複素数型の場合) の 1:*ilo*-1 と *i*+1:*n* の成分に格納される。*jobvs* ='V' の場合、*vs* には、*A* をその部分収束 Schur 形式に縮退させる変換を格納する。

i = *n*+1 :

いくつかの固有値が接近しすぎて分離できなかったため、固有値を順序変更できなかった (非常に悪い条件が問題)。

i = *n*+2 :

順序変更後、丸め操作によりいくつかの複素数の固有値の値が変更されたため、Schur 形式の先頭の固有値が *select* =.TRUE. を満たさなくなった。これは、スケーリングによるアンダーフローによっても引き起こされる。

アプリケーション・ノート

配列 *work* に必要なワークスペースの大きさがわからない場合は、最初の実行で *lwork* を大きめの値に設定する。実行の終了時に *work*(1) を調べて、これ以降の実行にはその値を使用するとよい。

?geev

一般マトリックスの固有値と左固有ベクトルおよび右固有ベクトルを計算する。

```
call sgeev ( jobvl, jobvr, n, a, lda, wr, wi, vl, ldvl, vr, ldvr,
             work, lwork, info)
call dgeev ( jobvl, jobvr, n, a, lda, wr, wi, vl, ldvl, vr, ldvr,
             work, lwork, info)
call cgeev ( jobvl, jobvr, n, a, lda, w, vl, ldvl, vr, ldvr, work,
             lwork, rwork, info)
call zgeev ( jobvl, jobvr, n, a, lda, w, vl, ldvl, vr, ldvr, work,
             lwork, rwork, info)
```

説明

このルーチンは、$n \times n$ の実数 / 複素数の非対称マトリックス A について、固有値を計算し、オプションで左 / 右の固有ベクトルを計算する。A の右固有ベクトル $v(j)$ は、以下を満足する。

$$A*v(j) = \lambda(j)*v(j)$$

$\lambda(j)$ は、その固有値である。

A の左固有ベクトル $u(j)$ は、以下を満足する。

$$u(j)^H*A = \lambda(j)*u(j)^H$$

$u(j)^H$ は、$u(j)$ の共役転置を示す。
計算された固有ベクトルは、1 に等しいユークリッド・ノルムと最大コンポーネント実数を持つように正規化される。

入力パラメータ

jobvl
: CHARACTER*1。'N' または 'V' でなければならない。
jobvl ='N' の場合、A の左固有ベクトルは計算されない。
jobvl ='V' の場合、A の左固有ベクトルは計算される。

jobvr CHARACTER*1。'N' または 'V' でなければならない。
jobvr ='N' の場合、*A* の右固有ベクトルは計算されない。
jobvr ='V' の場合、*A* の右固有ベクトルは計算される。

n INTEGER。マトリックス *A* の次数 ($n \geq 0$)。

a, work REAL (sgeev の場合)
DOUBLE PRECISION (dgeev の場合)
COMPLEX (cgeev の場合)
DOUBLE COMPLEX (zgeev の場合)
配列 :
*a(lda, *)* は、$n \times n$ のマトリックス *A* を格納する配列である。
a の第 2 次元は、max(1, *n*) 以上でなければならない。
work(lwork) は、ワークスペース配列である。

lda INTEGER。配列 *a* の第 1 次元。
max(1, *n*) 以上でなければならない。

ldvl, ldvr INTEGER。それぞれ、出力配列 *vl* と *vr* のリーディング・ディメンジョン。
次の制約がある。
ldvl ≥ 1、*ldvr* ≥ 1
jobvl ='V' の場合、*ldvl* $\geq$ max(1, *n*)
jobvr ='V' の場合、*ldvr* $\geq$ max(1, *n*)

lwork INTEGER。配列 *work* の次元。
次の制約がある。
lwork $\geq$ max(1, 3*n*)。*jobvl* ='V' または *jobvr* ='V' の場合、
lwork $\geq$ max(1, 4*n*) (実数型の場合)
lwork $\geq$ max(1, 2*n*) (複素数型の場合)
性能を上げるには、一般に *lwork* を大きくする必要がある。

rwork REAL (cgeev の場合)
DOUBLE PRECISION (zgeev の場合)
ワークスペース配列 DIMENSION は、max(1, 2*n*) 以上。
複素数型でのみ使用される。

出力パラメータ

a
終了時、この配列は、中間結果で上書きされる。

wr, wi
REAL (sgeev の場合)
DOUBLE PRECISION (dgeev の場合)
配列 DIMENSION はそれぞれ、max(1, *n*) 以上。
計算された固有値の実数部と虚数部をそれぞれ格納される。固有値の複素共役ペアは、虚数部が正である固有値に続いて格納される。

w
COMPLEX (cgeev の場合)
DOUBLE COMPLEX (zgeev の場合)
配列 DIMENSION は、max(1, *n*) 以上。
計算された固有値が格納される。

vl, vr
REAL (sgeev の場合)
DOUBLE PRECISION (dgeev の場合)
COMPLEX (cgeev の場合)
DOUBLE COMPLEX (zgeev の場合)
配列 :
vl(*ldvl*, *); *vl* の第 2 次元は、max(1, *n*) 以上でなければならない。
jobvl ='V' の場合、左固有ベクトル *u*(j) は、それらの固有値と同じ順序で、*vl* の列に次々と格納される。*jobvl* ='N' の場合、*vl* は参照されない。
実数型の場合 :
j 番目の固有値が実数の場合、*u*(j) = *vl*(:, j) (*vl* の j 番目の列) である。j 番目と (j+1) 番目の固有値が複素共役ペアとなる場合は、*u*(j) = *vl*(:, j) + *i***vl*(:, j+1)、*u*(j+1) = *vl*(:, j) - *i***vl*(:, j+1) である。($i=\sqrt{-1}$ である)
複素数型の場合 :
u(j) = *vl*(:, j) (*vl* の j 番目の列)
vr(*ldvr*, *);*vr* の第 2 次元は、max(1, *n*) 以上でなければならない。
jobvr ='V' の場合、右固有ベクトル *v*(j) は、それらの固有値と同じ順序で、*vr* の列に次々と格納される。*jobvr* ='N' の場合、*vr* は参照されない。

実数型の場合 :
j 番目の固有値が実数の場合、$v(j) = vr(:, j)$ (*vr* の j 番目の列) である。j 番目と (j+1) 番目の固有値が複素共役ペアとなる場合は、$v(j) = vr(:, j) + i * vr(:, j+1)$、$v(j+1) = vr(:, j) - i * vr(:, j+1)$ である。($i=\sqrt{-1}$ である)

複素数型の場合 :
$v(j) = vr(:, j)$ (*vr* の j 番目の列)

work(1) 終了時に *info* = 0 の場合、*work(1)* は、*lwork* の必要最小サイズを戻す。

info INTEGER

info = 0 の場合は、正常に終了している。

info = *-i* の場合は、*i* 番目のパラメータの値が不正である。

info = *i* の場合は、*QR* アルゴリズムが固有値をすべて計算できず、固有ベクトルが計算されなかった。収束した固有値は、*wr* と *wi* (実数型の場合) または *w*(複素数型の場合) の *i*+1:*n* の成分に格納される。

アプリケーション・ノート

配列 *work* に必要なワークスペースの大きさがわからない場合は、最初の実行で *lwork* を大きめの値に設定する。実行の終了時に *work(1)* を調べて、これ以降の実行にはその値を使用するとよい。

?geevx

あらかじめマトリックスを平衡化して
一般マトリックスの固有値と左固有
ベクトルおよび右固有ベクトルを計算し、
固有値と右固有ベクトルに対する条件数の
逆数を計算する。

```
call sgeevx ( balanc, jobvl, jobvr, sense, n, a, lda, wr, wi, vl,
              ldvl, vr, ldvr, ilo, ihi, scale, abnrm, rconde, rcondv,
              work, lwork, iwork, info)
call dgeevx ( balanc, jobvl, jobvr, sense, n, a, lda, wr, wi, vl,
              ldvl, vr, ldvr, ilo, ihi, scale, abnrm, rconde, rcondv,
              work, lwork, iwork, info)
call cgeevx ( balanc, jobvl, jobvr, sense, n, a, lda, w, vl, ldvl, vr,
              ldvr, ilo, ihi, scale, abnrm, rconde, rcondv, work,
              lwork, rwork, info)
call zgeevx ( balanc, jobvl, jobvr, sense, n, a, lda, w, vl, ldvl, vr,
              ldvr, ilo, ihi, scale, abnrm, rconde, rcondv, work,
              lwork, rwork, info)
```

説明

このルーチンは、*n* × *n* の実数 / 複素数の非対称マトリックス *A* について、固有値を計算し、オプションで左 / 右の固有ベクトルを計算する。

また、このルーチンは、平衡化変換を計算して、固有値と固有ベクトル (*ilo*、*ihi*、*scale*、*abnrm*)、固有値に対する条件数の逆数 (*rconde*)、右固有ベクトルに対する条件数の逆数 (*rcondv*) の条件付けも改善できる。

A の右固有ベクトル *v*(j) は、以下を満足する。

$$A*v(j) = \lambda(j)*v(j)$$

λ(j) は、その固有値である。

A の左固有ベクトル *u*(j) は、以下を満足する。

$$u(j)^H*A = \lambda(j)*u(j)^H$$

$u(j)^H$ は、$u(j)$ の共役転置を示す。
計算された固有ベクトルは、1 に等しいユークリッド・ノルムと最大コンポーネント実数を持つように正規化される。

マトリックスの平衡化は、行と列を置換してマトリックスを上三角により近づけ、対角相似変換 $D A D^{-1}$ (D は対角マトリックス) を適用して行と列をノルム内でより近づけ、その固有値と固有ベクトルの条件数をより小さくするのを意味する。計算された条件数の逆数は、平衡化されたマトリックスに一致する。
行と列を置換しても条件数は変更されないが (正確な演算において)、対角をスケーリングすると条件数は変更される。平衡化の詳細は、4.10 節の [*LUG*] を参照。

入力パラメータ

balanc
CHARACTER*1。'N'、'P'、'S'、または 'B' でなければならない。
入力マトリックスを対角的にスケーリングする方法、または入力マトリックスを置換してその固有値の条件付けを改善する方法を指定する。

balanc ='N' の場合、対角的なスケーリングまたは置換は実行されない。

balanc ='P' の場合、置換を実行してマトリックスを上三角により近づける。対角的なスケーリングは実行されない。

balanc ='S' の場合、マトリックスに対して対角的なスケーリングが実行される。つまり、A が $D A D^{-1}$ で置き換えられる (D は、A の行と列をノルム内でより等しくするために選択された対角マトリックス)。置換は実行されない。

balanc ='B' の場合、対角的なスケーリングが実行されるとともに、A が置換される。

計算された条件数の逆数は、平衡化または置換を実行後のマトリックスに使用される。置換を実行しても条件数は変更されないが (正確な演算では)、平衡化を実行すると条件数が変更される。

jobvl
CHARACTER*1。'N' または 'V' でなければならない。
jobvl ='N' の場合、A の左固有ベクトルは計算されない。

jobvl ='V' の場合、A の左固有ベクトルは計算される。

sense ='E' または 'B' の場合、*jobvl* は 'V' でなければならない。

jobvr
CHARACTER*1。'N' または 'V' でなければならない。
jobvr ='N' の場合、*A* の右固有ベクトルは計算されない。
jobvr ='V' の場合、*A* の右固有ベクトルは計算される。
sense ='E' または 'B' の場合、*jobvr* は 'V' でなければならない。

sense
CHARACTER*1。'N'、'E'、'V'、または 'B' でなければならない。
どの条件数の逆数を計算するのかを決定する。
sense ='N' の場合、計算は実行されない。
sense ='E' の場合、固有値についてのみ計算が実行される。
sense ='V' の場合、右固有ベクトルについてのみ計算が実行される。
sense ='B' の場合、固有値と右固有ベクトルについて計算が実行される。
sense が 'E' または 'B' の場合、左固有ベクトルと右固有ベクトルの両方についても計算を実行する必要がある (*jobvl* ='V' と *jobvr* ='V')。

n
INTEGER。マトリックス *A* の次数 ($n \geq 0$)。

a, work
REAL (sgeevx の場合)
DOUBLE PRECISION (dgeevx の場合)
COMPLEX (cgeevx の場合)
DOUBLE COMPLEX (zgeevx の場合)
配列 :
*a(lda, *)* は、$n \times n$ のマトリックス *A* を格納する配列である。
a の第 2 次元は、max(1, *n*) 以上でなければならない。
work(lwork) は、ワークスペース配列である。

lda
INTEGER。配列 *a* の第 1 次元。
max(1, *n*) 以上でなければならない。

ldvl, ldvr
INTEGER。それぞれ、出力配列 *vl* と *vr* のリーディング・ディメンジョン。次の制約がある。
$ldvl \geq 1$、$ldvr \geq 1$
jobvl ='V' の場合、$ldvl \geq \max(1, n)$
jobvr ='V' の場合、$ldvr \geq \max(1, n)$

lwork
INTEGER。配列 *work* の次元。
実数型の場合 :
sense ='N' または 'E' の場合、$lwork \geq \max(1, 2n)$ であり、*jobvl* ='V' または *jobvr* ='V' の場合、

lwork ≥ 3*n* ; *sense* = 'V' または 'B' の場合、
lwork ≥ *n*(*n*+6) である。
性能を上げるには、一般に *lwork* を大きくする必要がある。

複素数型の場合 :
sense = 'N' または 'E' の場合、*lwork* ≥ max(1, 2*n*)、
sense = 'V' または 'B' の場合、*lwork* ≥ n^2+2*n* である。
性能を上げるには、一般に *lwork* を大きくする必要がある。

rwork REAL (cgeevx の場合)
DOUBLE PRECISION (zgeevx の場合)
ワークスペース配列 DIMENSION は、max(1, 2*n*) 以上。
複素数型でのみ使用される。

iwork INTEGER
ワークスペース配列 DIMENSION は、max(1, 2*n*-2) 以上。
実数型でのみ使用される。*sense* = 'N' または 'E' の場合は参照されない。

出力パラメータ

a 終了時、この配列は上書きされる。*jobvl* = 'V' または *jobvr* = 'V' の場合、平衡化した入力マトリックス *A* の実数 Schur/Schur 形式が格納される。

wr, wi REAL (sgeevx の場合)
DOUBLE PRECISION (dgeevx の場合)
配列 DIMENSION はそれぞれ、max(1, *n*) 以上。
計算された固有値の実数部と虚数部がそれぞれ格納される。固有値の複素共役ペアは、虚数部が正である固有値に続いて格納される。

w COMPLEX (cgeevx の場合)
DOUBLE COMPLEX (zgeevx の場合)
配列 DIMENSION は、max(1, *n*) 以上。
計算された固有値が格納される。

vl, vr REAL (sgeevx の場合)
DOUBLE PRECISION (dgeevx の場合)
COMPLEX (cgeevx の場合)
DOUBLE COMPLEX (zgeevx の場合)

配列 :
`vl(ldvl, *)`; `vl` の第 2 次元は、max(1, `n`) 以上でなければならない。
`jobvl` = `'V'` の場合、左固有ベクトル u(j) は、それらの固有値と同じ順序で、`vl` の列に次々と格納される。`jobvl` = `'N'` の場合、`vl` は参照されない。
実数型の場合 :
j 番目の固有値が実数の場合、u(j) = `vl`(:, j) (`vl` の j 番目の列) である。j 番目と (j+1) 番目の固有値が複素共役ペアとなる場合は、u(j) = `vl`(:, j) + i* `vl`(:, j+1)、u(j+1) = `vl`(:, j) - i* `vl`(:, j+1) である。($i=\sqrt{-1}$ である)
複素数型の場合 :
u(j) = `vl`(:, j) (`vl` の j 番目の列)
`vr(ldvr, *)`;`vr` の第 2 次元は、max(1, `n`) 以上でなければならない。
`jobvr` = `'V'` の場合、右固有ベクトル v(j) は、それらの固有値と同じ順序で、`vr` の列に次々と格納される。`jobvr` = `'N'` の場合、`vr` は参照されない。
実数型の場合 :
j 番目の固有値が実数の場合、v(j) = `vr`(:, j) (`vr` の j 番目の列) である。j 番目と (j+1) 番目の固有値が複素共役ペアとなる場合は、v(j) = `vr`(:, j) + i* `vr`(:, j+1)、v(j+1) = `vr`(:, j) - i* `vr`(:, j+1) である。($i=\sqrt{-1}$ である)
複素数型の場合 :
v(j) = `vr`(:, j) (`vr` の j 番目の列)

`ilo, ihi` — `INTEGER`
`ilo` と `ihi` は、A を平衡化したときに決定される整数値である。
平衡化した A(i, j) = 0 である。(i > j と j = 1, ..., `ilo`-1 または i = `ihi`+1, ..., `n` の場合)
`balanc` = `'N'` または `'S'` の場合、`ilo` = 1 と `ihi` = `n` である。

`scale` — `REAL` (単精度型の場合)
`DOUBLE PRECISION` (倍精度型の場合)
配列 `DIMENSION` は、max(1, `n`) 以上。

A の平衡化時に適用された置換とスケーリング係数の各成分が格納される。*P*(j) が、行と列 j で交換された行と列のインデックスである場合、*D*(j) は、行と列 j に適用されたスケーリング係数であり、

scale(j) = *P*(j)、(j = 1, ..., *ilo*-1)
= *D*(j)、(j = *ilo*, ..., *ihi*)
= *P*(j)、(j = *ihi*+1, ..., *n*) である。

交換の実行順序は、*n* ～ *ihi*+1、次に、1 ～ *ilo*-1 になる。

abnrm
REAL (単精度型の場合)
DOUBLE PRECISION (倍精度型の場合)
平衡化したマトリックスの 1- ノルム (任意の列の要素に対する絶対値の合計の最大値)。

rconde, rcondv
REAL (単精度型の場合)
DOUBLE PRECISION (倍精度型の場合)
配列 DIMENSION はそれぞれ、max(1, *n*) 以上。
rconde(j) は、j 番目の固有値の条件数の逆数である。
rcondv(j) は、j 番目の右固有ベクトルの条件数の逆数である。

work(1)
終了時に *info* = 0 の場合、*work(1)* は、*lwork* の必要最小サイズを戻す。

info
INTEGER
info = 0 の場合は、正常に終了している。
info = *-i* の場合は、*i* 番目のパラメータの値が不正である。
info = *i* の場合は、*QR* アルゴリズムが固有値をすべて計算できず、固有ベクトルまたは条件数が計算されなかった。収束した固有値は、*wr* と *wi* (実数型の場合) または *w* (複素数型の場合) の 1:*ilo*-1 と *i*+1:*n* の成分に格納される。

アプリケーション・ノート

配列 *work* に必要なワークスペースの大きさがわからない場合は、最初の実行で *lwork* を大きめの値に設定する。実行の終了時に *work(1)* を調べて、これ以降の実行にはその値を使用するとよい。

特異値分解

本節では、特異値問題を解くために使用する LAPACK ドライバ・ルーチンについて説明する。これらの問題を解くために呼び出すのが可能な計算ルーチンも参照のこと。
表 5-12 に、ルーチンの詳細を示す。

表 5-12 特異値分解用のドライバ・ルーチン

ルーチン名	実行する演算
?gesvd	一般矩形マトリックスの特異値分解を計算する。
?gesdd	分割統治法を使用して、一般矩形マトリックスの特異値分解を計算する。
?ggsvd	一般矩形マトリックス・ペアの汎用特異値分解を計算する。

?gesvd

一般矩形マトリックスの特異値分解を計算する。

```
call sgesvd ( jobu, jobvt, m, n, a, lda, s, u, ldu, vt, ldvt, work,
              lwork, info)
call dgesvd ( jobu, jobvt, m, n, a, lda, s, u, ldu, vt, ldvt, work,
              lwork, info)
call cgesvd ( jobu, jobvt, m, n, a, lda, s, u, ldu, vt, ldvt, work,
              lwork, rwork, info)
call zgesvd ( jobu, jobvt, m, n, a, lda, s, u, ldu, vt, ldvt, work,
              lwork, rwork, info)
```

説明

このルーチンは、実数 / 複素数マトリックス A ($m \times n$) の特異値分解 (SVD) を計算し、オプションで左特異ベクトルまたは右特異ベクトルを計算する。SVD は、次のように記述される。

$$A = U \Sigma V^H$$

Σはその min(m, n) 対角成分を除いてゼロの $m \times n$ のマトリックス、U は $m \times m$ の直交 / ユニタリ・マトリックス、V は $n \times n$ の直交 / ユニタリ・マトリックスである。Σ の対角成分は、A の特異値である。これらは、実数かつ非負であり、降順で戻される。U と V の先頭から min(m, n) の列は、A の左特異ベクトルと右特異ベクトルである。
ルーチンは、V ではなく、V^H を戻すことに注意が必要である。

入力パラメータ

jobu
: CHARACTER*1。'A'、'S'、'O'、または 'N' でなければならない。
マトリックス U のすべてまたは一部を計算するオプションを指定する。
jobu ='A' の場合、U の m 個の列がすべて配列 *u* に戻される。
jobu ='S' の場合、U の先頭から min(m, n) の列 (左特異ベクトル) が配列 *u* に戻される。
jobu ='O' の場合、U の先頭から min(m, n) の列 (左特異ベクトル) が配列 *a* 上で上書きされる。
jobu ='N' の場合、U の列 (左特異ベクトル) は計算されない。

jobvt
: CHARACTER*1。'A'、'S'、'O'、または 'N' でなければならない。
マトリックス V^H のすべてまたは一部を計算するオプションを指定する。
jobvt ='A' の場合、V^H の n 個の行がすべて配列 *vt* に戻される。
jobvt ='S' の場合、V^H の先頭から min(m, n) の行 (右特異ベクトル) が配列 *vt* に戻される。
jobvt ='O' の場合、V^H の先頭から min(m, n) の行 (右特異ベクトル) が配列 *a* 上で上書きされる。
jobvt ='N' の場合、V^H の行 (右特異ベクトル) は計算されない。
jobvt と *jobu* は、どちらも 'O' にできない。

m
: INTEGER。マトリックス A の行数 ($m \geq 0$)。

n
: INTEGER。A の列数 ($n \geq 0$)。

a, work
REAL (sgesvd の場合)
DOUBLE PRECISION (dgesvd の場合)
COMPLEX (cgesvd の場合)
DOUBLE COMPLEX (zgesvd の場合)
配列 :
*a(lda, *)* は、$m \times n$ のマトリックス A を格納する配列である。
a の第 2 次元は、max(1, *n*) 以上でなければならない。
work(lwork) は、ワークスペース配列である。

lda
INTEGER。配列 *a* の第 1 次元。
max(1, *m*) 以上でなければならない。

ldu, ldvt
INTEGER。それぞれ、出力配列 *u* と *vt* のリーディング・ディメンジョン。次の制約がある。
ldu ≥ 1 ; *ldvt* ≥ 1
jobu ='S' または 'A' の場合、*ldu* ≥ *m*
jobvt ='A' の場合、*ldvt* ≥ *n*
jobvt ='S' の場合、*ldvt* ≥ min(*m*, *n*)

lwork
INTEGER。配列 *work* の次元。*lwork* ≥ 1 である。
次の制約がある。
lwork ≥ max(3*min(*m*, *n*)+max(*m*, *n*), 5*min(*m*, *n*)) (実数型の場合)
lwork ≥ 2*min(*m*, *n*)+max(*m*, *n*) (複素数型の場合)
性能を上げるには、一般に *lwork* を大きくする必要がある。

rwork
REAL (cgesvd の場合)
DOUBLE PRECISION (zgesvd の場合)
ワークスペース配列 DIMENSION は、
max(1, 5*min(*m*, *n*)) 以上。複素数型でのみ使用される。

出力パラメータ

a
終了時に *jobu* ='O' の場合、*a* は、U の先頭から min(*m*, *n*) の列 (列方向に格納される左特異ベクトル) で上書きされる。
jobvt ='O' の場合、*a* は、V^H の先頭から min(*m*, *n*) の行 (行方向に格納される右特異ベクトル) で上書きされる。

	jobu ≠'O' かつ *jobvt* ≠'O' の場合、*a* の内容は破棄される。
s	REAL (単精度型の場合) DOUBLE PRECISION (倍精度型の場合) 配列 DIMENSION は、max(1, min(*m*, *n*)) 以上。 *s*(i) ≥ *s*(i+1) となるようにソートされて *A* の特異値が格納される。
u, *vt*	REAL (sgesvd の場合) DOUBLE PRECISION (dgesvd の場合) COMPLEX (cgesvd の場合) DOUBLE COMPLEX (zgesvd の場合) 配列 : *u*(*ldu*, *); *jobu* ='A' の場合、*u* の第 2 次元は、max(1, *m*) 以上でなければならない。また、*jobu* ='S' の場合、*u* の第 2 次元は、max(1, min(*m*, *n*)) 以上でなければならない。 *jobu* ='A' の場合、*u* には、*m* × *m* の直交 / ユニタリ・マトリックス *U* が格納される。 *jobu* ='S' の場合、*u* には、*U* の先頭から min(*m*, *n*) の列 (列方向に格納される左特異ベクトル) が格納される。 *jobu* ='N' または 'O' の場合、*u* は参照されない。 *vt*(*ldvt*, *); *vt* の第 2 次元は、max(1, *n*) 以上でなければならない。 *jobvt* ='A' の場合、*vt* には、*n* × *n* の直交 / ユニタリ・マトリックス V^H が格納される。 *jobvt* ='S' の場合、*vt* には、V^H の先頭から min(*m*, *n*) の行 (行方向に格納される右特異ベクトル) が格納される。 *jobvt* ='N' または 'O' の場合、*vt* は参照されない。
work	終了時に *info* = 0 の場合、*work*(1) は、*lwork* の必要最小サイズを戻す。 **実数型の場合 :** *info* > 0 の場合、*work*(2:min(*m*, *n*)) には、対角が (必ずしもソートされていない) *s* 内にある上二重対角マトリックス *B* の非収束優対角要素が格納される。*B* は、*A* = *u* * *B* * *vt* を満足するため、*A* と同じ特異値を持ち、*u* と *vt* で関連付けられた特異ベクトルを持つ。

rwork
終了時 (複素数型の場合)、*info* > 0 であれば、*rwork*(1:min(*m*, *n*)-1) には、対角が (必ずしもソートされていない) *s* 内にある上二重対角マトリックス *B* の非収束優対角要素が格納される。*B* は、*A* = *u* * *B* * *vt* を満足するため、*A* と同じ特異値を持ち、*u* と *vt* で関連付けられた特異ベクトルを持つ。

info
INTEGER
info = 0 の場合は、正常に終了している。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。
info = *i* で、?bdsqr が収束していなかった場合、*i* は、ゼロに収束しなかった中間二重対角形式 *B* の優対角の数である。

アプリケーション・ノート

配列 *work* に必要なワークスペースの大きさがわからない場合は、最初の実行で *lwork* を大きめの値に設定する。実行の終了時に *work*(1) を調べて、これ以降の実行にはその値を使用するとよい。

?gesdd

分割統治法を使用して、一般矩形
マトリックスの特異値分解を計算する。

```
call sgesdd ( jobz, m, n, a, lda, s, u, ldu, vt, ldvt, work, lwork,
              iwork, info)
call dgesdd ( jobz, m, n, a, lda, s, u, ldu, vt, ldvt, work, lwork,
              iwork, info)
call cgesdd ( jobz, m, n, a, lda, s, u, ldu, vt, ldvt, work, lwork,
              rwork, iwork, info)
call zgesdd ( jobz, m, n, a, lda, s, u, ldu, vt, ldvt, work, lwork,
              rwork, iwork, info)
```

説明

このルーチンは、実数 / 複素数マトリックス A ($m \times n$) の特異値分解 (SVD) を計算し、オプションで左特異ベクトルまたは右特異ベクトルを計算する。特異ベクトルを計算する場合は、分割統治アルゴリズムを使用する。
SVD は、次のように記述される。

$$A = U \Sigma V^H$$

Σ はその min(m, n) 対 角成分を除いてゼロの $m \times n$ のマトリックス、U は $m \times m$ の直交 / ユニタリ・マトリックス、V は $n \times n$ の直交 / ユニタリ・マトリックスである。Σ の対角成分は、A の特異値である。これらは、実数かつ非負であり、降順で戻される。U と V の先頭から min(m, n) の列は、A の左特異ベクトルと右特異ベクトルである。
ルーチンは、V ではなく、V^H を戻すことに注意が必要である。

入力パラメータ

jobz CHARACTER*1。'A'、'S'、'O'、または 'N' でなければならない。
マトリックス U のすべてまたは一部を計算するオプションを指定する。

	jobz = 'A' の場合、U の *m* 個の列、V^T の *n* 個の行は、すべて配列 *u* と *vt* に戻される。 *jobz* = 'S' の場合、U の先頭から min(*m*, *n*) の列、V^T の先頭から min(m, n) の行は、配列 *u* と *vt* に戻される。 *jobz* = 'O' の場合、 *m* ≥ *n* であれば、U の先頭から *n* 個の列が配列 *a* 上で上書きされ、V^T の行がすべて配列 *vt* に戻される。 *m* < *n* であれば、U の列がすべて配列 *u* に戻され、V^T の先頭から *m* 個の行が配列 *vt* 内で上書きされる。 *jobz* = 'N' の場合、U の列、または V^T の行は計算されない。
m	INTEGER。マトリックス A の行数 (*m* ≥ 0)。
n	INTEGER。A の列数 (*n* ≥ 0)。
a, *work*	REAL (sgesdd の場合) DOUBLE PRECISION (dgesdd の場合) COMPLEX (cgesdd の場合) DOUBLE COMPLEX (zgesdd の場合) 配列 : *a*(*lda*, *) は、*m* × *n* のマトリックス A を格納する配列である。 *a* の第 2 次元は、max(1, *n*) 以上でなければならない。 *work*(*lwork*) は、ワークスペース配列である。
lda	INTEGER。配列 *a* の第 1 次元。 max(1, *m*) 以上でなければならない。
ldu, *ldvt*	INTEGER。それぞれ、出力配列 *u* と *vt* のリーディング・ディメンジョン。次の制約がある。 *ldu* ≥ 1 ; *ldvt* ≥ 1 *jobz* = 'S' または 'A' の場合、または *jobz* = 'O' の場合、*m* < *n* であれば、*ldu* ≥ *m* *jobz* = 'A'、または *jobz* = 'O' の場合、*m* ≥ *n* であれば、*ldvt* ≥ *n* *jobz* = 'S' の場合、*ldvt* ≥ min(*m*, *n*) である。

lwork INTEGER。配列 work の次元。*work*; *lwork* ≥ 1 である。*lwork* の推奨値は、「アプリケーション・ノート」を参照。

rwork REAL (cgesdd の場合)
DOUBLE PRECISION (zgesdd の場合)
jobz ='N' の場合、ワークスペース配列 DIMENSION は、max(1, 5*min(*m*, *n*)) 以上。それ以外の場合、*rwork* の次元は、$5*(\min(m, n))^2 + 7*\min(m, n)$ 以上でなければならない。この配列は、複素数型でのみ使用される。

iwork INTEGER。ワークスペース配列 DIMENSION は、max(1, 8*min(*m*, *n*)) 以上。

出力パラメータ

a 終了時、
jobz ='O' の場合、*m* ≥ *n* であれば、*a* は、*U* の先頭から *n* 個の列 (列方向に格納される左特異ベクトル) で上書きされる。*m* < *n* であれば、*a* は、V^T の先頭から *m* 個の行 (行方向に格納される右特異ベクトル) で上書きされる。
jobz ≠'O' の場合、*a* の内容は破棄される。

s REAL (単精度型の場合)
DOUBLE PRECISION (倍精度型の場合)
配列 DIMENSION は、max(1, min(*m*, *n*)) 以上。
s(i) ≥ *s*(i+1)となるようにソートされて*A*の特異値が格納される。

u, vt REAL (sgesdd の場合)
DOUBLE PRECISION (dgesdd の場合)
COMPLEX (cgesdd の場合)
DOUBLE COMPLEX (zgesdd の場合)
配列 :
u(*ldu*, *); *u* の第 2 次元は、max(1, *m*) 以上でなければならない (*jobz* ='A' または *jobz* ='O' であり、*m* < *n* の場合)。
jobz ='S' の場合、*u* の第 2 次元は、max(1, min(*m*, *n*)) 以上でなければならない。
jobz ='A' または *jobz* ='O' の場合、*m* < *n* であれば、*u* には、*m* × *m* の直交 / ユニタリ・マトリックス *U* が格納される。
jobz ='S' の場合、*u* には、*U* の先頭から min(*m*, *n*) の列 (列方向に格納される左特異ベクトル) が格納される。

jobz = 'O' かつ $m \geq n$、または jobz = 'N' の場合、u は参照されない。

vt(ldvt, *); vt の第 2 次元は、max(1, n) 以上でなければならない。

jobz = 'A' または jobz = 'O' の場合、$m \geq n$ であれば、vt には、$n \times n$ の直交 / ユニタリ・マトリックス V^T が格納される。

jobz = 'S' の場合、vt には、V^T の先頭から min(m, n) の行 (行方向に格納される右特異ベクトル) が格納される。

jobz = 'O' かつ $m < n$、または jobz = 'N' の場合、vt は参照されない。

work(1) 終了時に info = 0 の場合、work(1) は、lwork の必要最小サイズを戻す。

info INTEGER

info = 0 の場合は、正常に終了している。

info = -i の場合は、i 番目のパラメータの値が不正である。

info = i で、?bdsdc が収束していなかった場合は、更新プロセスが失敗した。

アプリケーション・ノート

実数型の場合 :

jobz = 'N' の場合、$lwork \geq 3*min(m, n) + max(max(m, n), 6*min(m, n))$

jobz = 'O' の場合、$lwork \geq 3*(min(m, n))^2 + max(max(m, n), 5*(min(m, n))^2 + 4*min(m, n))$

jobz = 'S' または 'A' の場合、$lwork \geq 3*(min(m, n))^2 + max(max(m, n), 4*(min(m, n))^2 + 4*min(m, n))$

複素数型の場合 :

jobz = 'N' の場合、$lwork \geq 2*min(m, n) + max(m, n)$

jobz = 'O' の場合、$lwork \geq 2*(min(m, n))^2 + max(m, n) + 2*min(m, n)$

jobz = 'S' または 'A' の場合、$lwork \geq (min(m, n))^2 + max(m, n) + 2*min(m, n)$

性能を上げるには、一般に lwork を大きくする必要がある。
配列 work に必要なワークスペースの大きさがわからない場合は、最初の実行で lwork を大きめの値に設定する。実行の終了時に work(1) を調べて、これ以降の実行にはその値を使用するとよい。

?ggsvd

一般矩形マトリックス・ペアの汎用特異値
分解を計算する。

```
call sggsvd ( jobu, jobv, jobq, m, n, p, k, l, a, lda, b, ldb, alpha,
              beta, u, ldu, v, ldv, q, ldq, work, iwork, info)
call dggsvd ( jobu, jobv, jobq, m, n, p, k, l, a, lda, b, ldb, alpha,
              beta, u, ldu, v, ldv, q, ldq, work, iwork, info)
call cggsvd ( jobu, jobv, jobq, m, n, p, k, l, a, lda, b, ldb, alpha,
              beta, u, ldu, v, ldv, q, ldq, work, rwork, iwork, info)
call zggsvd ( jobu, jobv, jobq, m, n, p, k, l, a, lda, b, ldb, alpha,
              beta, u, ldu, v, ldv, q, ldq, work, rwork, iwork, info)
```

説明

このルーチンは、$m \times n$ の実数 / 複素数マトリックス A と $p \times n$ の実数 / 複素数マトリックス B の汎用特異値分解 (GSVD) を計算する。

$$U^H A Q = D_1 * (0 \quad R)、V^H B Q = D_2 * (0 \quad R)$$

U、V、Q は、直交 / ユニタリ・マトリックスである。

$k+l=$ がマトリックス $(A^H, B^H)^H$ の有効な数値ランクの場合、R は、$(k+l) \times (k+l)$ の非特異上三角マトリックスとなり、D_1 と D_2 は、$m \times (k+l)$の"対角"マトリックスと$p \times (k+l)$の"対角"マトリックスとなる。D_1 と D_2 はそれぞれ、次の構造で示される。

$m-k-l \geq 0$ の場合、

$$D_1 = \begin{array}{c} \\ k \\ l \\ m-k-l \end{array} \begin{array}{c} \begin{array}{cc} k & l \end{array} \\ \begin{pmatrix} I & 0 \\ 0 & C \\ 0 & 0 \end{pmatrix} \end{array}$$

$$D_2 = \begin{array}{c} \\ l \\ p-l \end{array} \begin{array}{c} \begin{array}{cc} k & l \end{array} \\ \begin{pmatrix} 0 & S \\ 0 & 0 \end{pmatrix} \end{array}$$

$$
(0 \quad R) = \begin{array}{c} \\ k \\ l \end{array} \begin{array}{c} \begin{array}{ccc} n-k-l & k & l \end{array} \\ \begin{pmatrix} 0 & R_{11} & R_{12} \\ 0 & 0 & R_{22} \end{pmatrix} \end{array}
$$

C = diag (alpha(k+1), ..., alpha(k+1))
S = diag (beta(k+1), ..., beta(k+1))
$C^2 + S^2 = \mathrm{I}$

R は、終了時、a(1:k+1, n-k-1+1:n) に格納される。

m-k-1 < 0 の場合、

$$
D_1 = \begin{array}{c} \\ k \\ m-k \end{array} \begin{array}{c} \begin{array}{ccc} k & m-k & k+l-m \end{array} \\ \begin{pmatrix} I & 0 & 0 \\ 0 & C & 0 \end{pmatrix} \end{array}
$$

$$
D_2 = \begin{array}{c} \\ m-k \\ k+l-m \\ p-l \end{array} \begin{array}{c} \begin{array}{ccc} k & m-k & k+l-m \end{array} \\ \begin{pmatrix} 0 & S & 0 \\ 0 & 0 & I \\ 0 & 0 & 0 \end{pmatrix} \end{array}
$$

$$
(0 \quad R) = \begin{array}{c} \\ k \\ m-k \\ k+l-m \end{array} \begin{array}{c} \begin{array}{cccc} n-k-l & k & m-k & k+l-m \end{array} \\ \begin{pmatrix} 0 & R_{11} & R_{12} & R_{13} \\ 0 & 0 & R_{22} & R_{23} \\ 0 & 0 & 0 & R_{33} \end{pmatrix} \end{array}
$$

C = diag (alpha(k+1), ..., alpha(m)),
S = diag (beta(k+1), ..., beta(m)),
$C^2 + S^2 = \mathrm{I}$

終了時、 $\begin{pmatrix} R_{11} R_{12} R_{13} \\ 0 \;\; R_{22} R_{23} \end{pmatrix}$ は $a(1{:}m, n\text{-}k\text{-}l\text{+}1{:}n)$ に格納され、R_{33} は

$b(m\text{-}k\text{+}1{:}l, n\text{+}m\text{-}k\text{-}l\text{+}1{:}n)$ に格納される。

このルーチンは、C、S、R を計算し、オプションで直交 / ユニタリ変換マトリックス U、V、Q を計算する。
特に、B が $n \times n$ の非特異マトリックスの場合、A と B の GSVD は、暗黙的に AB^{-1} の SVD を提供する。

$$AB^{-1} = U(D_1 D_2^{-1}) V^H$$

$(A^H, B^H)^H$ が直交列を持っている場合は、A と B の GSVD も A と B の CS 分解に等しくなる。更に、GSVD を使用すると、固有値問題の解を導き出せる。

$$A^H A x = \lambda B^H B x$$

入力パラメータ

jobu CHARACTER*1。'U' または 'N' でなければならない。
jobu ='U' の場合、直交 / ユニタリ・マトリックス U が計算される。
jobu ='N' の場合、U は計算されない。

jobv CHARACTER*1。'V' または 'N' でなければならない。
jobv ='V' の場合、直交 / ユニタリ・マトリックス V が計算される。
jobv ='N' の場合、V は計算されない。

jobq CHARACTER*1。'Q' または 'N' でなければならない。
jobq ='Q' の場合、直交 / ユニタリ・マトリックス Q が計算される。
jobq ='N' の場合、Q は計算されない。

m INTEGER。マトリックス A の行数 ($m \geq 0$)。

n INTEGER。マトリックス A と B の列数 ($n \geq 0$)。

p INTEGER。マトリックス B の行数 B($p \geq 0$)。

a, b, work REAL (sggsvd の場合)
DOUBLE PRECISION (dggsvd の場合)
COMPLEX (cggsvd の場合)
DOUBLE COMPLEX (zggsvd の場合)

配列 :
a(lda, *) には、m × n のマトリックス A が格納される。
a の第 2 次元は、max(1, n) 以上でなければならない。
b(ldb, *) には、p × n のマトリックス B が格納される。
b の第 2 次元は、max(1, n) 以上でなければならない。
work(*) は、ワークスペース配列である。work の次元は、max(3n, m, p)+n 以上でなければならない。

lda
INTEGER。a の第 1 次元。max(1, m) 以上でなければならない。

ldb
INTEGER。b の第 1 次元。max(1, p) 以上でなければならない。

ldu
INTEGER。配列 u の第 1 次元。
jobu = 'U' の場合は、ldu ≥ max(1, m) 、それ以外の場合は、ldu ≥ 1

ldv
INTEGER。配列 v の第 1 次元。
jobv = 'V' の場合は、ldv ≥ max(1, p) 、それ以外の場合は、ldv ≥ 1

ldq
INTEGER。配列 q の第 1 次元。
jobq = 'Q' の場合は、ldq ≥ max(1, n) 、それ以外の場合は、ldq ≥ 1

iwork
INTEGER
ワークスペース配列 DIMENSION は、max(1, n) 以上。

rwork
REAL (cggsvd の場合)
DOUBLE PRECISION (zggsvd の場合)
ワークスペース配列 DIMENSION は、max(1, 2n) 以上。
複素数型でのみ使用される。

出力パラメータ

k, l
INTEGER。終了時、k と l は、サブブロックの次元を指定する。k+l の合計は、$(A^H, B^H)^H$ の有効数値ランクに等しい。

a
終了時、a には、三角マトリックス R または R の一部が格納される。

b
終了時、m-k-l < 0 の場合、b には、三角マトリックス R の一部が格納される。

alpha, beta
REAL (単精度型の場合)
DOUBLE PRECISION (倍精度型の場合)
配列 DIMENSION はそれぞれ、max(1, n) 以上。
A と B の汎用特異値ペアが格納される。

alpha(1:k) = 1、
beta(1:k) = 0、
および、m-k-l ≥ 0 の場合、
alpha(k+1:k+l) = C、
beta(k+1:k+l) = S、
または、m-k-l < 0 の場合、
alpha(k+1:m) = C、alpha(m+1:k+l) = 0
beta(k+1:m) = S、beta(m+1:k+l) = 1
および、
alpha(k+l+1:n) = 0
beta(k+l+1:n) = 0

u, v, q
REAL (sggsvd の場合)
DOUBLE PRECISION (dggsvd の場合)
COMPLEX (cggsvd の場合)
DOUBLE COMPLEX (zggsvd の場合)
配列 :
u(ldu, *); u の第 2 次元は、max(1, m) 以上でなければならない。
jobu='U' の場合、u には、m × m の直交 / ユニタリ・マトリックス U が格納される。
jobu='N' の場合、u は参照されない。
v(ldv, *); v の第 2 次元は、max(1, p) 以上でなければならない。
jobv='V' の場合、v には、p × p の直交 / ユニタリ・マトリックス V が格納される。
jobv='N' の場合、v は参照されない。
q(ldq, *); q の第 2 次元は、max(1, n) 以上でなければならない。
jobq='Q' の場合、q には、n × n の直交 / ユニタリ・マトリックス Q が格納される。
jobq='N' の場合、q は参照されない。

iwork
終了時、iwork には、ソート情報が格納される。

info INTEGER

info = 0 の場合は、正常に終了している。

info = -*i* の場合は、*i* 番目のパラメータの値が不正である。

info = 1 の場合は、Jacobi 型プロシージャが収束できなかった。詳細は、サブルーチン ?tgsja を参照。

汎用対称固有値問題

本節では、汎用対称固有値問題を解くために使用する LAPACK ドライバ・ルーチンについて説明する。これらの問題を解くために呼び出すのが可能な計算ルーチンも参照のこと。
表 5-13 に、ルーチンの詳細を示す。

表 5-13 汎用対称固有値問題を解くためのドライバ・ルーチン

ルーチン名	実行する演算
?sygv /?hegv	実数 / 複素数の汎用対称 / エルミート固有値問題について、すべての固有値を計算し、オプションですべての固有ベクトルを計算する。
?sygvd /?hegvd	実数 / 複素数の汎用対称 / エルミート固有値問題について、すべての固有値を計算し、オプションですべての固有ベクトルを計算する。固有ベクトルを計算する場合は、分割統治法を使用する。
?sygvx /?hegvx	実数 / 複素数の汎用対称 / エルミート固有値問題について、指定された固有値を計算し、オプションで、指定された固有ベクトルを計算する。
?spgv / ?hpgv	パックド格納形式のマトリックスに関する実数 / 複素数の汎用対称 / エルミート固有値問題について、すべての固有値を計算し、オプションですべての固有ベクトルを計算する。
?spgvd /?hpgvd	パックド格納形式のマトリックスに関する実数 / 複素数の汎用対称 / エルミート固有値問題について、すべての固有値を計算し、オプションですべての固有ベクトルを計算する。固有ベクトルを計算する場合は、分割統治法を使用する。
?spgvx /?hpgvx	パックド格納形式のマトリックスに関する実数 / 複素数の汎用対称 / エルミート固有値問題について、指定された固有値を計算し、オプションで、指定された固有ベクトルを計算する。
?sbgv /?hbgv	バンド・マトリックスに関する実数 / 複素数の汎用対称 / エルミート固有値問題について、すべての固有値を計算し、オプションですべての固有ベクトルを計算する。
?sbgvd /?hbgvd	バンド・マトリックスに関する実数 / 複素数の汎用対称 / エルミート固有値問題について、すべての固有値を計算し、オプションですべての固有ベクトルを計算する。固有ベクトルを計算する場合は、分割統治法を使用する。
?sbgvx /?hbgvx	バンド・マトリックスに関する実数 / 複素数の汎用対称 / エルミート固有値問題について、指定された固有値を計算し、オプションで、指定された固有ベクトルを計算する。

?sygv

実数の汎用対称固有値問題について、
すべての固有値を計算し、オプションで
すべての固有ベクトルを計算する。

```
call ssygv ( itype, jobz, uplo, n, a, lda, b, ldb, w, work, lwork,
             info )
call dsygv ( itype, jobz, uplo, n, a, lda, b, ldb, w, work, lwork,
             info )
```

説明

このルーチンは、次の形式で示される実数の汎用対称固有値問題について、すべての固有値を計算し、オプションですべての固有ベクトルを計算する。

$$Ax = \lambda Bx、ABx = \lambda x、または\ BAx = \lambda x$$

A と B は対称であると仮定する。また、B は正値である。

入力パラメータ

itype INTEGER。1、2、または 3 でなければならない。
解決する問題のタイプを指定する。
itype = 1 の場合、問題のタイプは $Ax = \lambda Bx$ である。
itype = 2 の場合、問題のタイプは $ABx = \lambda x$ である。
itype = 3 の場合、問題のタイプは $BAx = \lambda x$ である。

jobz CHARACTER*1。'N' または 'V' でなければならない。
jobz ='N' の場合、固有値のみを計算する。
jobz ='V' の場合、固有値と固有ベクトルを計算する。

uplo CHARACTER*1。'U' または 'L' でなければならない。
uplo ='U' の場合、配列 *a* と *b* は、A と B の上三角を格納する。
uplo ='L' の場合、配列 *a* と *b* は、A と B の下三角を格納する。

n INTEGER。マトリックス A と B の次数 ($n \geq 0$)。

a, b, work
REAL (ssygv の場合)
DOUBLE PRECISION (dsygv の場合)
配列 :
a(*lda*, *) には、*uplo* の値に従って、対称マトリックス A の上三角または下三角が格納される。
a の第 2 次元は、max(1, *n*) 以上でなければならない。
b(*ldb*, *) には、*uplo* の値に従って、正値対称マトリックス B の上三角または下三角が格納される。
b の第 2 次元は、max(1, *n*) 以上でなければならない。
work(*lwork*) は、ワークスペース配列である。

lda
INTEGER。*a* の第 1 次元。max(1, *n*) 以上でなければならない。

ldb
INTEGER。*b* の第 1 次元。max(1, *n*) 以上でなければならない。

lwork
INTEGER。配列 *work* の次元。*lwork* ≥ max(1, 3*n*-1)
lwork の推奨値は、「アプリケーション・ノート」を参照。

出力パラメータ

a
終了時、*jobz* = 'V' を指定した場合に *info* = 0 であると、*a* には、固有ベクトルのマトリックス Z が格納される。固有ベクトルは、次のように正規化される。
itype = 1 または 2 の場合、$Z^T B Z = I$
itype = 3 の場合、$Z^T B^{-1} Z = I$
jobz = 'N' を指定すると、終了時、A の上三角 (*uplo* = 'U' の場合)、または下三角 (*uplo* = 'L' の場合) が対角を含めて破棄される。

b
終了時に *info* ≤ *n* の場合、マトリックスを格納している *b* の一部が、コルスキー因子分解 $B = U^T U$ または $B = L L^T$ からの三角係数 U または L で上書きされる。

w
REAL (ssygv の場合)
DOUBLE PRECISION (dsygv の場合)
配列 DIMENSION は、max(1, *n*) 以上。
info = 0 の場合、固有値が昇順で格納される。

work(1) 終了時に *info* = 0 の場合、*work(1)* は、*lwork* の必要最小サイズを戻す。

info INTEGER

info = 0 の場合は、正常に終了している。

info = -*i* の場合は、*i* 番目の引数の値が不正である。

info > 0 の場合は、spotrf/dpotrf と ssyev/dsyev により、エラー・コードが戻される。

info = *i* ≤ *n* の場合は、ssyev/dsyev が収束せず、中間三重対角の *i* 個の対角外の成分がゼロに収束しなかった。

info = *n* + *i*(1 ≤ *i* ≤ *n*) の場合は、*B* について先頭から次数 *i* の小行列が正値でない。また、*B* の因子分解が完了できず、固有値または固有ベクトルが計算されなかった。

アプリケーション・ノート

性能を最大限に上げるには、*lwork* ≥ (*nb*+2)***n* を使用する (*nb* は、ilaenv により戻される ssytrd/dsytrd のブロック・サイズである)。配列 *work* に必要なワークスペースの大きさがわからない場合は、最初の実行で *lwork* を大きめの値に設定する。実行の終了時に *work*(1) を調べて、これ以降の実行にはその値を使用するとよい。

?hegv

複素数の汎用エルミート固有値問題について、すべての固有値を計算し、オプションですべての固有ベクトルを計算する。

```
call chegv ( itype, jobz, uplo, n, a, lda, b, ldb, w, work, lwork,
             rwork, info )
call zhegv ( itype, jobz, uplo, n, a, lda, b, ldb, w, work, lwork,
             rwork, info )
```

説明

このルーチンは、次の形式で示される複素数の汎用エルミート固有値問題について、すべての固有値を計算し、オプションですべての固有ベクトルを計算する。

$$Ax = \lambda Bx、ABx = \lambda x、または\ BAx = \lambda x$$

A と B はエルミートであると仮定する。また、B は正値である。

入力パラメータ

itype INTEGER。1、2、または 3 でなければならない。
解決する問題のタイプを指定する。
itype = 1 の場合、問題のタイプは $Ax = \lambda Bx$ である。
itype = 2 の場合、問題のタイプは $ABx = \lambda x$ である。
itype = 3 の場合、問題のタイプは $BAx = \lambda x$ である。

jobz CHARACTER*1。'N' または 'V' でなければならない。
jobz ='N' の場合、固有値のみを計算する。
jobz ='V' の場合、固有値と固有ベクトルを計算する。

uplo CHARACTER*1。'U' または 'L' でなければならない。
uplo ='U' の場合、配列 *a* と *b* は、A と B の上三角を格納する。
uplo ='L' の場合、配列 *a* と *b* は、A と B の下三角を格納する。

n INTEGER。マトリックス A と B の次数 ($n \geq 0$)。

a, b, work
: COMPLEX (chegv の場合)
DOUBLE COMPLEX (zhegv の場合)
配列 :
*a(lda, *)* には、*uplo* の値に従って、エルミート・マトリックス A の上三角または下三角が格納される。
a の第 2 次元は、max(1, *n*) 以上でなければならない。
*b(ldb, *)* には、*uplo* の値に従って、正値エルミート・マトリックス *B* の上三角または下三角が格納される。
b の第 2 次元は、max(1, *n*) 以上でなければならない。
work(lwork) は、ワークスペース配列である。

lda
: INTEGER。*a* の第 1 次元。max(1, *n*) 以上でなければならない。

ldb
: INTEGER。*b* の第 1 次元。max(1, *n*) 以上でなければならない。

lwork
: INTEGER。配列 *work* の次元。
lwork ≥ max(1, 2*n*-1)
lwork の推奨値は、「アプリケーション・ノート」を参照。

rwork
: REAL (chegv の場合)
DOUBLE PRECISION (zhegv の場合)
ワークスペース配列 DIMENSION は、max(1, 3*n*-2) 以上。

出力パラメータ

a
: 終了時、*jobz* = 'V' を指定した場合に *info* = 0 であると、*a* には、固有ベクトルのマトリックス *Z* が格納される。固有ベクトルは、次のように正規化される。
itype = 1 または 2 の場合、$Z^H B Z = I$
itype = 3 の場合、$Z^H B^{-1} Z = I$
jobz = 'N' を指定すると、終了時、*A* の上三角 (*uplo* = 'U' の場合)、または下三角 (*uplo* = 'L' の場合) が対角を含めて破棄される。

b
: 終了時に *info* ≤ *n* の場合、マトリックスを格納している *b* の一部が、コルスキー因子分解 $B = U^H U$ または $B = L L^H$ からの三角係数 *U* または *L* で上書きされる。

w REAL (chegv の場合)
DOUBLE PRECISION (zhegv の場合)
配列 DIMENSION は、max(1, *n*) 以上。
info = 0 の場合、固有値が昇順で格納される。

work(1) 終了時に *info* = 0 の場合、*work(1)* は、*lwork* の必要最小サイズを戻す。

info INTEGER
info = 0 の場合は、正常に終了している。
info = -*i* の場合は、*i* 番目の引数の値が不正である。
info > 0 の場合は、cpotrf/zpotrf と cheev/zheev により、エラー・コードが戻される。

info = *i* ≤ *n* の場合は、cheev/zheev が収束せず、中間三重対角の *i* 個の対角外の成分がゼロに収束しなかった。

info = *n* + *i*(1 ≤ *i* ≤ *n*) の場合は、*B* について先頭から次数 *i* の小行列が正値でない。また、*B* の因子分解が完了できず、固有値または固有ベクトルが計算されなかった。

アプリケーション・ノート

性能を最大限に上げるには、*lwork* ≥ (*nb*+1)***n* を使用する (*nb* は、ilaenv により戻される chetrd/zhetrd のブロック・サイズである)。配列 *work* に必要なワークスペースの大きさがわからない場合は、最初の実行で *lwork* を大きめの値に設定する。実行の終了時に *work(1)* を調べて、これ以降の実行にはその値を使用するとよい。

?sygvd

実数の汎用対称固有値問題について、
すべての固有値を計算し、オプションで
すべての固有ベクトルを計算する。
固有ベクトルを計算する場合は、分割
統治法を使用する。

```
call ssygvd ( itype, jobz, uplo, n, a, lda, b, ldb, w, work, lwork,
              iwork, liwork, info )
call dsygvd ( itype, jobz, uplo, n, a, lda, b, ldb, w, work, lwork,
              iwork, liwork, info )
```

説明

このルーチンは、次の形式で示される実数の汎用対称固有値問題について、すべての固有値を計算し、オプションですべての固有ベクトルを計算する。

$$Ax = \lambda Bx、ABx = \lambda x、または\ BAx = \lambda x$$

A と B は対称であると仮定する。また、B は正値である。

固有ベクトルを計算する場合は、分割統治アルゴリズムを使用する。

入力パラメータ

itype　INTEGER。1、2、または 3 でなければならない。
解決する問題のタイプを指定する。
itype = 1 の場合、問題のタイプは $Ax = \lambda Bx$ である。
itype = 2 の場合、問題のタイプは $ABx = \lambda x$ である。
itype = 3 の場合、問題のタイプは $BAx = \lambda x$ である。

jobz　CHARACTER*1。'N' または 'V' でなければならない。
jobz ='N' の場合、固有値のみを計算する。
jobz ='V' の場合、固有値と固有ベクトルを計算する。

uplo	CHARACTER*1。'U' または 'L' でなければならない。 *uplo* ='U' の場合、配列 *a* と *b* は、*A* と *B* の上三角を格納する。 *uplo* ='L' の場合、配列 *a* と *b* は、*A* と *B* の下三角を格納する。
n	INTEGER。マトリックス *A* と *B* の次数 ($n \geq 0$)。
a, b, work	REAL (ssygvd の場合) DOUBLE PRECISION (dsygvd の場合) 配列 : *a(lda, *)* には、*uplo* の値に従って、対称マトリックス *A* の上三角または下三角が格納される。 *a* の第 2 次元は、max(1, *n*) 以上でなければならない。 *b(ldb, *)* には、*uplo* の値に従って、正値対称マトリックス *B* の上三角または下三角が格納される。 *b* の第 2 次元は、max(1, *n*) 以上でなければならない。 *work(lwork)* は、ワークスペース配列である。
lda	INTEGER。*a* の第 1 次元。max(1, *n*) 以上でなければならない。
ldb	INTEGER。*b* の第 1 次元。max(1, *n*) 以上でなければならない。
lwork	INTEGER。配列 *work* の次元。 次の制約がある。 $n \leq 1$ の場合、*lwork* ≥ 1 *jobz* ='N' かつ $n > 1$ の場合、*lwork* $\geq 2n+1$ *jobz* ='V' かつ $n > 1$ の場合、*lwork* $\geq 2n^2+6n+1$
iwork	INTEGER ワークスペース配列 DIMENSION(*liwork*)
liwork	INTEGER。配列 *iwork* の次元。 次の制約がある。 $n \leq 1$ の場合、*liwork* ≥ 1 *jobz* ='N' かつ $n > 1$ の場合、*liwork* ≥ 1 *jobz* ='V' かつ $n > 1$ の場合、*liwork* $\geq 5n+3$

出力パラメータ

a

終了時、*jobz* = 'V' を指定した場合に *info* = 0 であると、*a* には、固有ベクトルのマトリックス *Z* が格納される。固有ベクトルは、次のように正規化される。

itype = 1 または 2 の場合、$Z^T B Z = I$

itype = 3 の場合、$Z^T B^{-1} Z = I$

jobz = 'N' を指定すると、終了時、*A* の上三角 (*uplo* = 'U' の場合)、または下三角 (*uplo* = 'L' の場合) が対角を含めて破棄される。

b

終了時に *info* ≤ *n* の場合、マトリックスを格納している *b* の一部が、コルスキー因子分解 $B = U^T U$ または $B = L L^T$ からの三角係数 *U* または *L* で上書きされる。

w

REAL (ssygvd の場合)

DOUBLE PRECISION (dsygvd の場合)

配列 DIMENSION は、max(1, *n*) 以上。

info = 0 の場合、固有値が昇順で格納される。

work(1)

終了時に *info* = 0 の場合、*work(1)* は、*lwork* の必要最小サイズを戻す。

iwork(1)

終了時に *info* = 0 の場合、*iwork(1)* は、*liwork* の必要最小サイズを戻す。

info

INTEGER

info = 0 の場合は、正常に終了している。

info = -*i* の場合は、*i* 番目の引数の値が不正である。

info > 0 の場合は、spotrf/dpotrf と ssyev/dsyev により、エラー・コードが戻される。

info = *i* ≤ *n* の場合は、ssyev/dsyev が収束せず、中間三重対角の *i* 個の対角外の成分がゼロに収束しなかった。

info = *n* + *i*(1 ≤ *i* ≤ *n*) の場合は、*B* について先頭から次数 *i* の小行列が正値でない。また、*B* の因子分解が完了できず、固有値または固有ベクトルが計算されなかった。

?hegvd

複素数の汎用エルミート固有値問題について、すべての固有値を計算し、オプションですべての固有ベクトルを計算する。固有ベクトルを計算する場合は、分割統治法を使用する。

```
call chegvd ( itype, jobz, uplo, n, a, lda, b, ldb, w, work, lwork,
              rwork, lrwork, iwork, liwork, info )
call zhegvd ( itype, jobz, uplo, n, a, lda, b, ldb, w, work, lwork,
              rwork, lrwork, iwork, liwork, info )
```

説明

このルーチンは、次の形式で示される複素数の汎用エルミート固有値問題について、すべての固有値を計算し、オプションですべての固有ベクトルを計算する。

$$Ax = \lambda Bx、ABx = \lambda x、または\ BAx = \lambda x$$

A と B はエルミートであると仮定する。また、B は正値である。
固有ベクトルを計算する場合は、分割統治アルゴリズムを使用する。

入力パラメータ

itype INTEGER。1、2、または 3 でなければならない。
解決する問題のタイプを指定する。
itype = 1 の場合、問題のタイプは $Ax = \lambda Bx$ である。
itype = 2 の場合、問題のタイプは $ABx = \lambda x$ である。
itype = 3 の場合、問題のタイプは $BAx = \lambda x$ である。

jobz CHARACTER*1。'N' または 'V' でなければならない。
jobz ='N' の場合、固有値のみを計算する。
jobz ='V' の場合、固有値と固有ベクトルを計算する。

uplo	CHARACTER*1。'U' または 'L' でなければならない。 *uplo* ='U' の場合、配列 *a* と *b* は、*A* と *B* の上三角を格納する。 *uplo* ='L' の場合、配列 *a* と *b* は、*A* と *B* の下三角を格納する。
n	INTEGER。マトリックス *A* と *B* の次数 ($n \geq 0$)。
a, b, work	COMPLEX (chegvd の場合) DOUBLE COMPLEX (zhegvd の場合) 配列 : *a(lda, *)* には、*uplo* の値に従って、エルミート・マトリックス *A* の上三角または下三角が格納される。 *a* の第 2 次元は、max(1, *n*) 以上でなければならない。 *b(ldb, *)* には、*uplo* の値に従って、正値エルミート・マトリックス *B* の上三角または下三角が格納される。 *b* の第 2 次元は、max(1, *n*) 以上でなければならない。 *work(lwork)* は、ワークスペース配列である。
lda	INTEGER。*a* の第 1 次元。max(1, *n*) 以上でなければならない。
ldb	INTEGER。*b* の第 1 次元。max(1, *n*) 以上でなければならない。
lwork	INTEGER。配列 *work* の次元。 次の制約がある。 $n \leq 1$ の場合、*lwork* ≥ 1 *jobz* ='N' かつ $n > 1$ の場合、*lwork* $\geq n+1$ *jobz* ='V' かつ $n > 1$ の場合、*lwork* $\geq n^2+2n$
rwork	REAL (chegvd の場合) DOUBLE PRECISION (zhegvd の場合) ワークスペース配列 DIMENSION (*lrwork*)
lrwork	INTEGER。配列 *rwork* の次元。 次の制約がある。 $n \leq 1$ の場合、*lrwork* ≥ 1 *jobz* ='N' かつ $n > 1$ の場合、*lrwork* $\geq n$ *jobz* ='V' かつ $n > 1$ の場合、*lrwork* $\geq 2n^2+5n+1$
iwork	INTEGER ワークスペース配列 DIMENSION (*liwork*)

liwork INTEGER。配列 *iwork* の次元。
次の制約がある。
$n \le 1$ の場合、*liwork* ≥ 1
jobz ='N' かつ $n > 1$ の場合、*liwork* ≥ 1
jobz ='V' かつ $n > 1$ の場合、*liwork* $\ge 5n+3$

出力パラメータ

a 終了時、*jobz* ='V' を指定した場合に *info* = 0 であると、*a* には、固有ベクトルのマトリックス Z が格納される。固有ベクトルは、次のように正規化される。
itype = 1 または 2 の場合、$Z^H B Z = I$
itype = 3 の場合、$Z^H B^{-1} Z = I$
jobz ='N' を指定すると、終了時、A の上三角 (*uplo* ='U' の場合)、または下三角 (*uplo* ='L' の場合) が対角を含めて破棄される。

b 終了時に *info* $\le n$ の場合、マトリックスを格納している *b* の一部が、コルスキー因子分解 $B = U^H U$ または $B = L L^H$ からの三角係数 U または L で上書きされる。

w REAL (chegvd の場合)
DOUBLE PRECISION (zhegvd の場合)
配列 DIMENSION は、max(1, *n*) 以上。
info = 0 の場合、固有値が昇順で格納される。

work(1) 終了時に *info* = 0 の場合、*work(1)* は、*lwork* の必要最小サイズを戻す。

rwork(1) 終了時に *info* = 0 の場合、*rwork(1)* は、*lrwork* の必要最小サイズを戻す。

iwork(1) 終了時に *info* = 0 の場合、*iwork(1)* は、*liwork* の必要最小サイズを戻す。

info INTEGER
info = 0 の場合は、正常に終了している。
info = -*i* の場合は、*i* 番目の引数の値が不正である。
info > 0 の場合は、cpotrf/zpotrf と cheev/zheev により、エラー・コードが戻される。

info = *i* ≤ *n* の場合は、cheev/zheev が収束せず、中間三重対角の *i* 個の対角外の成分がゼロに収束しなかった。

info = *n* + *i*(1 ≤ *i* ≤ *n*) の場合は、*B* について先頭から次数 *i* の小行列が正値でない。また、*B* の因子分解が完了できず、固有値または固有ベクトルが計算されなかった。

?sygvx

実数の汎用対称固有値問題について、
指定された固有値を計算し、
オプションで、指定された固有ベクトルを
計算する。

```
call ssygvx(itype, jobz, range, uplo, n, a, lda, b, ldb, vl, vu, il,
            iu, abstol, m, w, z, ldz, work, lwork, iwork, ifail, info)
call dsygvx(itype, jobz, range, uplo, n, a, lda, b, ldb, vl, vu, il,
            iu, abstol, m, w, z, ldz, work, lwork, iwork, ifail, info)
```

説明

このルーチンは、次の形式で示される実数の汎用対称固有値問題について、指定された固有値を計算し、オプションで、指定された固有ベクトルを計算する。

$$Ax = \lambda Bx、ABx = \lambda x、または\ BAx = \lambda x$$

A と B は対称であると仮定する。また、B は正値である。
固有値と固有ベクトルを選択するには、希望する固有値の値の範囲かインデックスの範囲を指定する。

入力パラメータ

itype INTEGER。1、2、または 3 でなければならない。
解決する問題のタイプを指定する。
itype = 1 の場合、問題のタイプは $Ax = \lambda Bx$ である。
itype = 2 の場合、問題のタイプは $ABx = \lambda x$ である。
itype = 3 の場合、問題のタイプは $BAx = \lambda x$ である。

jobz CHARACTER*1。'N' または 'V' でなければならない。
jobz ='N' の場合、固有値のみを計算する。
jobz ='V' の場合、固有値と固有ベクトルを計算する。

range CHARACTER*1。'A'、'V'、または 'I' でなければならない。

	range = 'A' の場合、ルーチンは固有値をすべて計算する。 *range* = 'V' の場合、ルーチンは半開区間 *vl* < $\lambda_i \le$ *vu* における固有値 λ_i を計算する。 *range* = 'I' の場合、ルーチンはインデックスが *il* ～ *iu* である固有値を計算する。
uplo	CHARACTER*1。'U' または 'L' でなければならない。 *uplo* = 'U' の場合、配列 *a* と *b* は、*A* と *B* の上三角を格納する。 *uplo* = 'L' の場合、配列 *a* と *b* は、*A* と *B* の下三角を格納する。
n	INTEGER。マトリックス *A* と *B* の次数 (*n* ≥ 0)。
a, b, work	REAL (ssygvx の場合) DOUBLE PRECISION (dsygvx の場合) 配列 : *a(lda, *)* には、*uplo* の値に従って、対称マトリックス *A* の上三角または下三角が格納される。 *a* の第 2 次元は、max(1, *n*) 以上でなければならない。 *b(ldb, *)* には、*uplo* の値に従って、正値対称マトリックス *B* の上三角または下三角が格納される。 *b* の第 2 次元は、max(1, *n*) 以上でなければならない。 *work(lwork)* は、ワークスペース配列である。
lda	INTEGER。*a* の第 1 次元。max(1, *n*) 以上でなければならない。
ldb	INTEGER。*b* の第 1 次元。max(1, *n*) 以上でなければならない。
vl, vu	REAL (ssygvx の場合) DOUBLE PRECISION (dsygvx の場合) *range* = 'V' の場合、固有値を検索する区間の上限と下限を指定する。 次の制約がある。 *vl* < *vu* *range* = 'A' または 'I' の場合、*vl* と *vu* は参照されない。

il, iu INTEGER

range = 'I' の場合、最小の固有値から最大の固有値のインデックスが昇順で戻される。次の制約がある。$1 \le il \le iu \le n$ ($n > 0$ の場合)、*il*=1 と *iu* = 0 ($n = 0$ の場合)

range = 'A' または 'V' の場合、*il* と *iu* は参照されない。

abstol REAL (ssygvx の場合)

DOUBLE PRECISION (dsygvx の場合)

固有値に対する絶対エラー許容値。

詳しくは、「アプリケーション・ノート」を参照。

ldz INTEGER。出力配列 *z* のリーディング・ディメンジョン。

次の制約がある。

$ldz \ge 1$; *jobz* = 'V' の場合、$ldz \ge \max(1, n)$。

lwork INTEGER。配列 *work* の次元。

$lwork \ge \max(1, 8n)$ でなければならない。

lwork の推奨値は、「アプリケーション・ノート」を参照。

iwork INTEGER

ワークスペース配列 DIMENSION は、$\max(1, 5n)$ 以上。

出力パラメータ

a 終了時、*A* の上三角 (*uplo* = 'U' の場合)、または下三角 (*uplo* = 'L' の場合) が対角を含めて上書きされる。

b 終了時に $info \le n$ の場合、マトリックスを格納している *b* の一部が、コルスキー因子分解 $B = U^T U$ または $B = L L^T$ からの三角係数 *U* または *L* で上書きされる。

m INTEGER。検出された固有値の総数 ($0 \le m \le n$)。

range = 'A' の場合は $m = n$、*range* = 'I' の場合は $m = iu - il + 1$ である。

w, z REAL (ssygvx の場合)

DOUBLE PRECISION (dsygvx の場合)

配列 :

w(*)、DIMENSION は、max(1, n) 以上。
w の先頭から m 個の成分には、指定された固有値が昇順で格納される。

z(ldz, *)。z の第 2 次元は、max(1, m) 以上でなければならない。
jobz='V' を指定した場合に info=0 であると、w(i) に関連する固有ベクトルを格納している z の i 番目の列とともに、指定された固有値に対応するマトリックス A の正規直交固有ベクトルが、z の先頭から m 個の列に格納される。固有ベクトルは、次のように正規化される。
itype=1 または 2 の場合、$Z^T B Z = I$
itype=3 の場合、$Z^T B^{-1} Z = I$

jobz='N' の場合、z は参照されない。
固有ベクトルが収束できない場合、z のその列には、その固有ベクトルに対する最新の近似値が格納され、固有ベクトルのインデックスは ifail に戻される。
注 : 配列 z には、max(1, m) 以上の列を指定する必要がある。range='V' の場合、m の正確な値は事前にはわからないため、上限を使用する必要がある。

work(1)　終了時に info=0 の場合、work(1) は、lwork の必要最小サイズを戻す。

ifail　INTEGER
配列 DIMENSION は、max(1, n) 以上。
jobz='V' を指定した場合に info=0 であると、ifail の先頭から m 個の成分はゼロになる。
info>0 の場合、ifail には、収束しなかった固有ベクトルのインデックスが格納される。
jobz='N' の場合、ifail は参照されない。

info　INTEGER
info=0 の場合は、正常に終了している。
info=-i の場合は、i 番目の引数の値が不正である。
info>0 の場合は、spotrf/dpotrf と ssyevx/dsyevx により、エラー・コードが戻される。

info = *i* ≤ *n* の場合は、ssyevx/dsyevx が収束せず、*i* 個の固有ベクトルが収束しなかった。これらのインデックスは、配列 *ifail* に格納される。
info = *n* + *i*(1 ≤ *i* ≤ *n*) の場合は、*B* について先頭から次数 *i* の小行列が正値でない。また、*B* の因子分解が完了できず、固有値または固有ベクトルが計算されなかった。

アプリケーション・ノート

abstol + ε * max(|a|, |b|) 以下の幅の区間 [a, b] 内に存在していると判別された場合 (ε はマシン精度)、近似固有値は、収束したものとして受け入れられる。*abstol* がゼロ以下の場合、$\varepsilon * \|T\|_1$ は所定の場所で使用される (*T* は、*A* を三重対角形式に縮退して獲得した三重対角マトリックス)。固有値が最も正確に計算されるのは、*abstol* にアンダーフローしきい値 2*?lamch('S') (ゼロ以外) の 2 倍の値を設定したときである。このルーチンで *info* > 0 が戻された場合は、収束しなかった固有ベクトルがいくつかあるのを示している。その場合は、*abstol* に 2*?lamch('S') を設定する。

性能を最大限に上げるには、*lwork* ≥ (*nb*+3)*n を使用する (*nb* は、ilaenv により戻される ssytrd/dsytrd のブロック・サイズである)。配列 *work* に必要なワークスペースの大きさがわからない場合は、最初の実行で *lwork* を大きめの値に設定する。実行の終了時に *work*(1) を調べて、これ以降の実行にはその値を使用するとよい。

?hegvx

複素数の汎用エルミート固有値問題について、指定された固有値を計算し、オプションで、指定された固有ベクトルを計算する。

```
call chegvx ( itype, jobz, range, uplo, n, a, lda, b, ldb, vl, vu, il,
              iu, abstol, m, w, z, ldz, work, lwork, rwork, iwork,
              ifail, info)
call zhegvx ( itype, jobz, range, uplo, n, a, lda, b, ldb, vl, vu, il,
              iu, abstol, m, w, z, ldz, work, lwork, rwork, iwork,
              ifail, info)
```

説明

このルーチンは、次の形式で示される複素数の汎用エルミート固有値問題について、指定された固有値を計算し、オプションで、指定された固有ベクトルを計算する。

$$Ax = \lambda Bx、ABx = \lambda x、または\ BAx = \lambda x$$

A と B はエルミートであると仮定する。また、B は正値である。
固有値と固有ベクトルを選択するには、希望する固有値の値の範囲かインデックスの範囲を指定する。

入力パラメータ

itype INTEGER。1、2、または 3 でなければならない。
解決する問題のタイプを指定する。
itype = 1 の場合、問題のタイプは $Ax = \lambda Bx$ である。
itype = 2 の場合、問題のタイプは $ABx = \lambda x$ である。
itype = 3 の場合、問題のタイプは $BAx = \lambda x$ である。

jobz CHARACTER*1。'N' または 'V' でなければならない。
jobz ='N' の場合、固有値のみを計算する。
jobz ='V' の場合、固有値と固有ベクトルを計算する。

range CHARACTER*1。'A'、'V'、または 'I' でなければならない。

	range = 'A' の場合、ルーチンは固有値をすべて計算する。 *range* = 'V' の場合、ルーチンは半開区間 $vl < \lambda_i \le vu$ における固有値 λ_i を計算する。 *range* = 'I' の場合、ルーチンはインデックスが *il* ～ *iu* である固有値を計算する。
uplo	CHARACTER*1。'U' または 'L' でなければならない。 *uplo* = 'U' の場合、配列 *a* と *b* は、*A* と *B* の上三角を格納する。 *uplo* = 'L' の場合、配列 *a* と *b* は、*A* と *B* の下三角を格納する。
n	INTEGER。マトリックス *A* と *B* の次数 ($n \ge 0$)。
a, b, work	COMPLEX (chegvx の場合) DOUBLE COMPLEX (zhegvx の場合) 配列 : *a(lda, *)* には、*uplo* の値に従って、エルミート・マトリックス *A* の上三角または下三角が格納される。 *a* の第 2 次元は、max(1, *n*) 以上でなければならない。 *b(ldb, *)* には、*uplo* の値に従って、正値エルミート・マトリックス *B* の上三角または下三角が格納される。 *b* の第 2 次元は、max(1, *n*) 以上でなければならない。 *work(lwork)* は、ワークスペース配列である。
lda	INTEGER。*a* の第 1 次元。max(1, *n*) 以上でなければならない。
ldb	INTEGER。*b* の第 1 次元。max(1, *n*) 以上でなければならない。
vl, vu	REAL (chegvx の場合) DOUBLE PRECISION (zhegvx の場合) *range* = 'V' の場合、固有値を検索する区間の上限と下限を指定する。 次の制約がある。 *vl* < *vu* *range* = 'A' または 'I' の場合、*vl* と *vu* は参照されない。

il, iu
INTEGER
range = 'I' の場合、最小の固有値から最大の固有値のインデックスが昇順で戻される。次の制約がある。
$1 \le il \le iu \le n$ ($n > 0$ の場合)、il=1 と $iu = 0$ ($n = 0$ の場合)
range = 'A' または 'V' の場合、*il* と *iu* は参照されない。

abstol
REAL (chegvx の場合)
DOUBLE PRECISION (zhegvx の場合)
固有値に対する絶対エラー許容値。
詳細は、「アプリケーション・ノート」を参照。

ldz
INTEGER。出力配列 *z* のリーディング・ディメンジョン。
次の制約がある。
$ldz \ge 1$。*jobz* = 'V' の場合、$ldz \ge \max(1, n)$

lwork
INTEGER。配列 *work* の次元。
$lwork \ge \max(1, 2n-1)$ でなければならない。
lwork の推奨値は、「アプリケーション・ノート」を参照。

rwork
REAL (chegvx の場合)
DOUBLE PRECISION (zhegvx の場合)
ワークスペース配列 DIMENSION は、$\max(1, 7n)$ 以上。

iwork
INTEGER
ワークスペース配列 DIMENSION は、$\max(1, 5n)$ 以上。

出力パラメータ

a
終了時、A の上三角 (*uplo* = 'U' の場合)、または下三角 (*uplo* = 'L' の場合) が対角を含めて上書きされる。

b
終了時に $info \le n$ の場合、マトリックスを格納している *b* の一部が、コルスキー因子分解 $B = U^H U$ または $B = L L^H$ からの三角係数 U または L で上書きされる。

m
INTEGER。検出された固有値の総数 ($0 \le m \le n$)。
range = 'A' の場合は $m = n$、*range* = 'I' の場合は $m = iu-il+1$ である。

w
REAL (chegvx の場合)
DOUBLE PRECISION (zhegvx の場合)
配列 DIMENSION は、max(1, *n*) 以上。
w の先頭から *m* 個の成分には、指定された固有値が昇順で格納される。

z
COMPLEX (chegvx の場合)
DOUBLE COMPLEX (zhegvx の場合)
配列 *z(ldz, *)*。*z* の第 2 次元は、max(1, *m*) 以上でなければならない。
jobz = 'V' を指定した場合に *info* = 0 であると、*w*(*i*) に関連する固有ベクトルを格納している *z* の *i* 番目の列とともに、指定された固有値に対応するマトリックス *A* の正規直交固有ベクトルが、*z* の先頭から *m* 個の列に格納される。固有ベクトルは、次のように正規化される。
itype = 1 または 2 の場合、$Z^H B Z = I$
itype = 3 の場合、$Z^H B^{-1} Z = I$
jobz = 'N' の場合、*z* は参照されない。
固有ベクトルが収束できない場合、*z* のその列には、その固有ベクトルに対する最新の近似値が格納され、固有ベクトルのインデックスは *ifail* に戻される。
注 : 配列 *z* には、max(1, *m*) 以上の列を指定する必要がある。*range* = 'V' の場合、*m* の正確な値は事前にはわからないため、上限を使用する必要がある。

work(1)
終了時に *info* = 0 の場合、*work(1)* は、*lwork* の必要最小サイズを戻す。

ifail
INTEGER
配列 DIMENSION は、max(1, *n*) 以上。
jobz = 'V' を指定した場合に *info* = 0 であると、*ifail* の先頭から *m* 個の成分はゼロになる。
info > 0 の場合、*ifail* には、収束しなかった固有ベクトルのインデックスが格納される。
jobz = 'N' の場合、*ifail* は参照されない。

info INTEGER

info = 0 の場合は、正常に終了している。

info = -*i* の場合は、*i* 番目の引数の値が不正である。

info > 0 の場合は、cpotrf/zpotrf と cheevx/zheevx により、エラー・コードが戻される。

info = *i* ≤ *n* の場合は、cheevx/zheevx が収束せず、*i* 個の固有ベクトルが収束しなかった。これらのインデックスは、配列 *ifail* に格納される。

info = *n* + *i*(1 ≤ *i* ≤ *n*) の場合は、*B* について先頭から次数 *i* の小行列が正値でない。また、*B* の因子分解が完了できず、固有値または固有ベクトルが計算されなかった。

アプリケーション・ノート

abstol + ε * max(|a|, |b|) 以下の幅の区間 [a, b] 内に存在していると判別された場合 (ε はマシン精度)、近似固有値は、収束したものとして受け入れられる。*abstol* がゼロ以下の場合、$\varepsilon * \|T\|_1$ は所定の場所で使用される (*T* は、*A* を三重対角形式に縮退して獲得した三重対角マトリックス)。固有値が最も正確に計算されるのは、*abstol* にアンダーフローしきい値 2*?lamch('S') (ゼロ以外) の 2 倍の値を設定したときである。このルーチンで *info* > 0 が戻された場合は、収束しなかった固有ベクトルがいくつかあるのを示している。その場合は、*abstol* に 2*?lamch('S') を設定する。

性能を最大限に上げるには、*lwork* ≥ (*nb*+1)*n を使用する (*nb* は、ilaenv により戻される chetrd/zhetrd のブロック・サイズである)。配列 *work* に必要なワークスペースの大きさがわからない場合は、最初の実行で *lwork* を大きめの値に設定する。実行の終了時に *work*(1) を調べて、これ以降の実行にはその値を使用するとよい。

?spgv

パックド格納形式のマトリックスに関する実数の汎用対称固有値問題について、すべての固有値を計算し、オプションですべての固有ベクトルを計算する。

```
call sspgv ( itype, jobz, uplo, n, ap, bp, w, z, ldz, work, info )
call dspgv ( itype, jobz, uplo, n, ap, bp, w, z, ldz, work, info )
```

説明

このルーチンは、次の形式で示される実数の汎用対称固有値問題について、すべての固有値を計算し、オプションですべての固有ベクトルを計算する。

$$Ax = \lambda Bx、ABx = \lambda x、または\ BAx = \lambda x$$

A と B は対称であり、パックド形式で格納されると仮定する。また、B は正値である。

入力パラメータ

itype INTEGER。1、2、または 3 でなければならない。
解決する問題のタイプを指定する。
itype = 1 の場合、問題のタイプは $Ax = \lambda Bx$ である。
itype = 2 の場合、問題のタイプは $ABx = \lambda x$ である。
itype = 3 の場合、問題のタイプは $BAx = \lambda x$ である。

jobz CHARACTER*1。'N' または 'V' でなければならない。
jobz ='N' の場合、固有値のみを計算する。
jobz ='V' の場合、固有値と固有ベクトルを計算する。

uplo CHARACTER*1。'U' または 'L' でなければならない。
uplo ='U' の場合、配列 *ap* と *bp* は、A と B の上三角を格納する。
uplo ='L' の場合、配列 *ap* と *bp* は、A と B の下三角を格納する。

n INTEGER。マトリックス A と B の次数 ($n \geq 0$)。

ap, bp, work
REAL (sspgv の場合)
DOUBLE PRECISION (dspgv の場合)
配列 :
ap(*) には、*uplo* の値に従って、対称マトリックス *A* の上三角または下三角がパックド形式で格納される。*ap* の次元は、max(1, *n**(*n*+1)/2) 以上でなければならない。
bp(*) には、*uplo* の値に従って、対称マトリックス *B* の上三角または下三角がパックド形式で格納される。*bp* の次元は、max(1, *n**(*n*+1)/2) 以上でなければならない。
work(*) は、ワークスペース配列 DIMENSION であり、max(1, 3*n*) 以上。

ldz
INTEGER。出力配列 *z* のリーディング・ディメンジョン。
ldz ≥ 1 である。*jobz* = 'V' の場合、*ldz* ≥ max(1, *n*) である。

出力パラメータ

ap
終了時、*ap* の内容は上書きされる。

bp
終了時、コルスキー因子分解 $B = U^T U$ または $B = L L^T$ からの三角係数 U または L が、B と同じ格納形式で格納される。

w, z
REAL (sspgv の場合)
DOUBLE PRECISION (dspgv の場合)
配列 :
w(*)、DIMENSION は、max(1, *n*) 以上。
info = 0 の場合は、固有値が昇順で格納される。
*z(ldz, *)*。*z* の第 2 次元は、max(1, *n*) 以上でなければならない。
jobz = 'V' を指定した場合に *info* = 0 であると、*z* には、固有ベクトルのマトリックス Z が格納される。固有ベクトルは、次のように正規化される。
itype = 1 または 2 の場合、$Z^T B Z = I$
itype = 3 の場合、$Z^T B^{-1} Z = I$
jobz = 'N' の場合、*z* は参照されない。

info INTEGER

info = 0 の場合は、正常に終了している。

info = -*i* の場合は、*i* 番目の引数の値が不正である。

info > 0 の場合は、spptrf/dpptrf と sspev/dspev により、エラー・コードが戻される。

info = *i* ≤ *n* の場合は、sspev/dspev が収束せず、中間三重対角の *i* 個の対角外の成分がゼロに収束しなかった。

info = *n* + *i*(1 ≤ *i* ≤ *n*) の場合は、*B* について先頭から次数 *i* の小行列が正値でない。また、*B* の因子分解が完了できず、固有値または固有ベクトルが計算されなかった。

?hpgv

パックド格納形式のマトリックスに関する複素数の汎用エルミート固有値問題について、すべての固有値を計算し、オプションですべての固有ベクトルを計算する。

```
call chpgv ( itype, jobz, uplo, n, ap, bp, w, z, ldz, work, rwork,
             info )
call zhpgv ( itype, jobz, uplo, n, ap, bp, w, z, ldz, work, rwork,
             info )
```

説明

このルーチンは、次の形式で示される複素数の汎用エルミート固有値問題について、すべての固有値を計算し、オプションですべての固有ベクトルを計算する。

$$Ax = \lambda Bx、ABx = \lambda x、または\ BAx = \lambda x$$

A と B はエルミートであり、パックド形式で格納されると仮定する。また、B は正値である。

入力パラメータ

itype
: INTEGER。1、2、または 3 でなければならない。
解決する問題のタイプを指定する。
itype = 1 の場合、問題のタイプは $Ax = \lambda Bx$ である。
itype = 2 の場合、問題のタイプは $ABx = \lambda x$ である。
itype = 3 の場合、問題のタイプは $BAx = \lambda x$ である。

jobz
: CHARACTER*1。'N' または 'V' でなければならない。
jobz ='N' の場合、固有値のみを計算する。
jobz ='V' の場合、固有値と固有ベクトルを計算する。

uplo
: CHARACTER*1。'U' または 'L' でなければならない。
uplo ='U' の場合、配列 *ap* と *bp* は、A と B の上三角を格納する。
uplo ='L' の場合、配列 *ap* と *bp* は、A と B の下三角を格納する。

n INTEGER。マトリックス A と B の次数 ($n \geq 0$)。

ap, bp, work COMPLEX (chpgv の場合)
DOUBLE COMPLEX (zhpgv の場合)
配列 :
ap(*) には、*uplo* の値に従って、エルミート・マトリックス A の上三角または下三角がパックド形式で格納される。*ap* の次元は、max(1, *n**(*n*+1)/2) 以上でなければならない。
bp(*) には、*uplo* の値に従って、エルミート・マトリックス B の上三角または下三角がパックド形式で格納される。*bp* の次元は、max(1, *n**(*n*+1)/2) 以上でなければならない。
work(*) は、ワークスペース配列 DIMENSION であり、max(1, 2*n*-1) 以上。

ldz INTEGER。出力配列 *z* のリーディング・ディメンジョン。
ldz ≥ 1 である。*jobz* = 'V' の場合、*ldz* ≥ max(1, *n*) である。

rwork REAL (chpgv の場合)
DOUBLE PRECISION (zhpgv の場合)
ワークスペース配列 DIMENSION は、max(1, 3*n*-2) 以上。

出力パラメータ

ap 終了時、*ap* の内容は上書きされる。

bp 終了時、コルスキー因子分解 $B = U^H U$ または $B = L L^H$ からの三角係数 U または L が、B と同じ格納形式で格納される。

w REAL (chpgv の場合)
DOUBLE PRECISION (zhpgv の場合)
配列 DIMENSION は、max(1, *n*) 以上。
info = 0 の場合は、固有値が昇順で格納される。

z COMPLEX (chpgv の場合)
DOUBLE COMPLEX (zhpgv の場合)
配列 *z(ldz, *)*。*z* の第 2 次元は、max(1, *n*) 以上でなければならない。
jobz = 'V' を指定した場合に *info* = 0 であると、*z* には、固有ベクトルのマトリックス Z が格納される。固有ベクトルは、次のように正規化される。

itype = 1 または 2 の場合、$Z^H B Z = I$
itype = 3 の場合、$Z^H B^{-1} Z = I$

jobz = 'N' の場合、*z* は参照されない。

info INTEGER

info = 0 の場合は、正常に終了している。
info = -*i* の場合は、*i* 番目の引数の値が不正である。
info > 0 の場合は、cpptrf/zpptrf と chpev/zhpev により、エラー・コードが戻される。

info = *i* ≤ *n* の場合は、chpev/zhpev が収束せず、中間三重対角の *i* 個の対角外の成分がゼロに収束しなかった。

info = *n* + *i*(1 ≤ *i* ≤ *n*) の場合は、*B* について先頭から次数 *i* の小行列が正値でない。また、*B* の因子分解が完了できず、固有値または固有ベクトルが計算されなかった。

?spgvd

パックド格納形式のマトリックスに関する実数の汎用対称固有値問題について、すべての固有値を計算し、オプションですべての固有ベクトルを計算する。固有ベクトルを計算する場合は、分割統治法を使用する。

```
call sspgvd ( itype, jobz, uplo, n, ap, bp, w, z, ldz, work, lwork,
              iwork, liwork, info )
call dspgvd ( itype, jobz, uplo, n, ap, bp, w, z, ldz, work, lwork,
              iwork, liwork, info )
```

説明

このルーチンは、次の形式で示される実数の汎用対称固有値問題について、すべての固有値を計算し、オプションですべての固有ベクトルを計算する。

$$Ax = \lambda Bx、ABx = \lambda x、または\ BAx = \lambda x$$

A と B は対称であり、パックド形式で格納されると仮定する。また、B は正値である。固有ベクトルを計算する場合は、分割統治アルゴリズムを使用する。

入力パラメータ

itype INTEGER。1、2、または 3 でなければならない。
解決する問題のタイプを指定する。
itype = 1 の場合、問題のタイプは $Ax = \lambda Bx$ である。
itype = 2 の場合、問題のタイプは $ABx = \lambda x$ である。
itype = 3 の場合、問題のタイプは $BAx = \lambda x$ である。

jobz CHARACTER*1。'N' または 'V' でなければならない。
jobz ='N' の場合、固有値のみを計算する。
jobz ='V' の場合、固有値と固有ベクトルを計算する。

uplo CHARACTER*1。'U' または 'L' でなければならない。
uplo ='U' の場合、配列 *ap* と *bp* は、*A* と *B* の上三角を格納する。
uplo ='L' の場合、配列 *ap* と *bp* は、*A* と *B* の下三角を格納する。

n INTEGER。マトリックス *A* と *B* の次数 ($n \geq 0$)。

ap, bp, work REAL (sspgvd の場合)
DOUBLE PRECISION (dspgvd の場合)
配列 :
ap(*) には、*uplo* の値に従って、対称マトリックス *A* の上三角または下三角がパックド形式で格納される。*ap* の次元は、max(1, *n**(*n*+1)/2) 以上でなければならない。
bp(*) には、*uplo* の値に従って、対称マトリックス *B* の上三角または下三角がパックド形式で格納される。*bp* の次元は、max(1, *n**(*n*+1)/2) 以上でなければならない。
work(lwork) は、ワークスペース配列である。

ldz INTEGER。出力配列 *z* のリーディング・ディメンジョン。
ldz ≥ 1 である。*jobz* ='V' の場合、*ldz* $\geq \max(1, n)$ である。

lwork INTEGER。配列 *work* の次元。
次の制約がある。
$n \leq 1$ の場合、*lwork* ≥ 1
jobz ='N' かつ $n > 1$ の場合、*lwork* $\geq 2n$
jobz ='V' かつ $n > 1$ の場合、*lwork* $\geq 2n^2+6n+1$

iwork INTEGER
ワークスペース配列 DIMENSION (*liwork*)

liwork INTEGER。配列 *iwork* の次元。
次の制約がある。
$n \leq 1$ の場合、*liwork* ≥ 1
jobz ='N' かつ $n > 1$ の場合、*liwork* ≥ 1
jobz ='V' かつ $n > 1$ の場合、*liwork* $\geq 5n+3$

出力パラメータ

ap 終了時、*ap* の内容は上書きされる。

bp
終了時、コルスキー因子分解 $B = U^T U$ または $B = L L^T$ からの三角係数 U または L が、B と同じ格納形式で格納される。

w, z
REAL (sspgv の場合)
DOUBLE PRECISION (dspgv の場合)
配列 :
w(*)、DIMENSION は、max(1, *n*) 以上。
info = 0 の場合は、固有値が昇順で格納される。
*z(ldz, *)*。*z* の第 2 次元は、max(1, *n*) 以上でなければならない。
jobz = 'V' を指定した場合に *info* = 0 であると、*z* には、固有ベクトルのマトリックス Z が格納される。
固有ベクトルは、次のように正規化される。
itype = 1 または 2 の場合、$Z^T B Z = I$
itype = 3 の場合、$Z^T B^{-1} Z = I$
jobz = 'N' の場合、*z* は参照されない。

work(1)
終了時に *info* = 0 の場合、*work(1)*、*lwork* の必要最小サイズを戻す。

iwork(1)
終了時に *info* = 0 の場合、*iwork(1)* は、*liwork* の必要最小サイズを戻す。

info
INTEGER
info = 0 の場合は、正常に終了している。
info = -*i* の場合は、*i* 番目の引数の値が不正である。
info > 0 の場合は、spptrf/dpptrf と sspevd/dspevd により、エラー・コードが戻される。
info = *i* ≤ *n* の場合は、sspevd/dspevd が収束せず、中間三重対角の *i* 個の対角外の成分がゼロに収束しなかった。
info = *n* + *i*(1 ≤ *i* ≤ *n*) の場合は、*B* について先頭から次数 *i* の小行列が正値でない。また、*B* の因子分解が完了できず、固有値または固有ベクトルが計算されなかった。

?hpgvd

パックド格納形式のマトリックスに関する複素数の汎用エルミート固有値問題について、すべての固有値を計算し、オプションですべての固有ベクトルを計算する。固有ベクトルを計算する場合は、分割統治法を使用する。

```
call chpgvd ( itype, jobz, uplo, n, ap, bp, w, z, ldz, work, lwork,
              rwork, lrwork, iwork, liwork, info )
call zhpgvd ( itype, jobz, uplo, n, ap, bp, w, z, ldz, work, lwork,
              rwork, lrwork, iwork, liwork, info )
```

説明

このルーチンは、次の形式で示される複素数の汎用エルミート固有値問題について、すべての固有値を計算し、オプションですべての固有ベクトルを計算する。

$$Ax = \lambda Bx、ABx = \lambda x、または\ BAx = \lambda x$$

A と B はエルミートであり、パックド形式で格納されると仮定する。また、B は正値である。固有ベクトルを計算する場合は、分割統治アルゴリズムを使用する。

入力パラメータ

itype INTEGER。1、2、または 3 でなければならない。
解決する問題のタイプを指定する。
itype = 1 の場合、問題のタイプは $Ax = \lambda Bx$ である。
itype = 2 の場合、問題のタイプは $ABx = \lambda x$ である。
itype = 3 の場合、問題のタイプは $BAx = \lambda x$ である。

jobz CHARACTER*1。'N' または 'V' でなければならない。
jobz ='N' の場合、固有値のみを計算する。
jobz ='V' の場合、固有値と固有ベクトルを計算する。

uplo CHARACTER*1。'U' または 'L' でなければならない。
uplo ='U' の場合、配列 *ap* と *bp* は、*A* と *B* の上三角を格納する。
uplo ='L' の場合、配列 *ap* と *bp* は、*A* と *B* の下三角を格納する。

n INTEGER。マトリックス *A* と *B* の次数 ($n \geq 0$)。

ap, bp, work COMPLEX (chpgvd の場合)
DOUBLE COMPLEX (zhpgvd の場合)
配列 :
ap(*) には、*uplo* の値に従って、エルミート・マトリックス *A* の上三角または下三角がパックド形式で格納される。*ap* の次元は、max(1, *n**(*n*+1)/2) 以上でなければならない。
bp(*) には、*uplo* の値に従って、エルミート・マトリックス *B* の上三角または下三角がパックド形式で格納される。*bp* の次元は、max(1, *n**(*n*+1)/2) 以上でなければならない。
work(lwork) は、ワークスペース配列である。

ldz INTEGER。出力配列 *z* のリーディング・ディメンジョン。
ldz ≥ 1 である。*jobz* ='V' の場合、*ldz* $\geq$ max(1, *n*) である。

lwork INTEGER。配列 *work* の次元。
次の制約がある。
$n \leq 1$ の場合、*lwork* ≥ 1
jobz ='N' かつ $n > 1$ の場合、*lwork* $\geq n$
jobz ='V' かつ $n > 1$ の場合、*lwork* $\geq 2n$

rwork REAL (chpgvd の場合)
DOUBLE PRECISION (zhpgvd の場合)
ワークスペース配列 DIMENSION (*lrwork*)

lrwork INTEGER。配列 *rwork* の次元。
次の制約がある。
$n \leq 1$ の場合、*lrwork* ≥ 1
jobz ='N' かつ $n > 1$ の場合、*lrwork* $\geq n$
jobz ='V' かつ $n > 1$ の場合、*lrwork* $\geq 2n^2+5n+1$

iwork INTEGER
ワークスペース配列 DIMENSION (*liwork*)

liwork INTEGER。配列 *iwork* の次元。
次の制約がある。
$n \le 1$ の場合、*liwork* ≥ 1
jobz = 'N' かつ $n > 1$ の場合、*liwork* ≥ 1
jobz = 'V' かつ $n > 1$ の場合、*liwork* $\ge 5n+3$

出力パラメータ

ap 終了時、*ap* の内容は上書きされる。

bp 終了時、コルスキー因子分解 $B = U^H U$ または $B = L L^H$ からの三角係数 U または L が、B と同じ格納形式で格納される。

w REAL (chpgvd の場合)
DOUBLE PRECISION (zhpgvd の場合)
配列 DIMENSION は、max(1, *n*) 以上。
info = 0 の場合は、固有値が昇順で格納される。

z COMPLEX (chpgvd の場合)
DOUBLE COMPLEX (zhpgvd の場合)
配列 *z(ldz, *)*。*z* の第 2 次元は、max(1, *n*) 以上でなければならない。
jobz = 'V' を指定した場合に *info* = 0 であると、*z* には、固有ベクトルのマトリックス Z が格納される。
固有ベクトルは、次のように正規化される。
itype = 1 または 2 の場合、$Z^H B Z = I$
itype = 3 の場合、$Z^H B^{-1} Z = I$
jobz = 'N' の場合、*z* は参照されない。

work(1) 終了時に *info* = 0 の場合、*work(1)* は、*lwork* の必要最小サイズを戻す。

rwork(1) 終了時に *info* = 0 の場合、*rwork(1)* は、*lrwork* の必要最小サイズを戻す。

iwork(1) 終了時に *info* = 0 の場合、*iwork(1)* は、*liwork* の必要最小サイズを戻す。

info INTEGER

info = 0 の場合は、正常に終了している。

info = -*i* の場合は、*i* 番目の引数の値が不正である。

info > 0 の場合は、cpptrf/zpptrf と chpevd/zhpevd により、エラー・コードが戻される。

info = *i* ≤ *n* の場合は、chpevd/zhpevd が収束せず、中間三重対角の *i* 個の対角外の成分がゼロに収束しなかった。

info = *n* + *i*(1 ≤ *i* ≤ *n*) の場合は、*B* について先頭から次数 *i* の小行列が正値でない。また、*B* の因子分解が完了できず、固有値または固有ベクトルが計算されなかった。

?spgvx

パックド格納形式のマトリックスに関する実数の汎用対称固有値問題について、指定された固有値を計算し、オプションで、指定された固有ベクトルを計算する。

```
call sspgvx ( itype, jobz, range, uplo, n, ap, bp, vl, vu, il, iu,
              abstol, m, w, z, ldz, work, iwork, ifail, info )
call dspgvx ( itype, jobz, range, uplo, n, ap, bp, vl, vu, il, iu,
              abstol, m, w, z, ldz, work, iwork, ifail, info )
```

説明

このルーチンは、次の形式で示される実数の汎用対称固有値問題について、指定された固有値を計算し、オプションで、指定された固有ベクトルを計算する。

$$Ax = \lambda Bx、ABx = \lambda x、または\ BAx = \lambda x$$

A と B は対称であり、パックド形式で格納されると仮定する。また、B は正値である。
固有値と固有ベクトルを選択するには、希望する固有値の値の範囲かインデックスの範囲を指定する。

入力パラメータ

itype INTEGER。1、2、または 3 でなければならない。
解決する問題のタイプを指定する。
itype = 1 の場合、問題のタイプは $Ax = \lambda Bx$ である。
itype = 2 の場合、問題のタイプは $ABx = \lambda x$ である。
itype = 3 の場合、問題のタイプは $BAx = \lambda x$ である。

jobz CHARACTER*1。'N' または 'V' でなければならない。
jobz ='N' の場合、固有値のみを計算する。
jobz ='V' の場合、固有値と固有ベクトルを計算する。

range CHARACTER*1。'A'、'V'、または 'I' でなければならない。

	range = 'A' の場合、ルーチンは固有値をすべて計算する。 *range* = 'V' の場合、ルーチンは半開区間 $vl < \lambda_i \le vu$ における固有値 λ_i を計算する。 *range* = 'I' の場合、ルーチンはインデックスが *il* ～ *iu* である固有値を計算する。
uplo	CHARACTER*1。'U' または 'L' でなければならない。 *uplo* = 'U' の場合、配列 *ap* と *bp* は、*A* と *B* の上三角を格納する。 *uplo* = 'L' の場合、配列 *ap* と *bp* は、*A* と *B* の下三角を格納する。
n	INTEGER。マトリックス *A* と *B* の次数 ($n \ge 0$)。
ap, bp, work	REAL (sspgvx の場合) DOUBLE PRECISION (dspgvx の場合) 配列 : *ap(*)* には、*uplo* の値に従って、対称マトリックス *A* の上三角または下三角がパックド形式で格納される。*ap* の次元は、max(1, *n**(*n*+1)/2) 以上でなければならない。 *bp(*)* には、*uplo* の値に従って、対称マトリックス *B* の上三角または下三角がパックド形式で格納される。*bp* の次元は、max(1, *n**(*n*+1)/2) 以上でなければならない。 *work(*)* は、ワークスペース配列 DIMENSION であり、max(1, 8*n*) 以上。
vl, vu	REAL (sspgvx の場合) DOUBLE PRECISION (dspgvx の場合) *range* = 'V' の場合、固有値を検索する区間の上限と下限を指定する。 次の制約がある。 $vl < vu$ *range* = 'A' または 'I' の場合、*vl* と *vu* は参照されない。
il, iu	INTEGER *range* = 'I' の場合、最小の固有値から最大の固有値のインデックスが昇順で戻される。次の制約がある。 $1 \le il \le iu \le n$ ($n > 0$ の場合)、*il*=1 と $iu = 0$ ($n = 0$ の場合) *range* = 'A' または 'V' の場合、*il* と *iu* は参照されない。

abstol REAL (sspgvx の場合)
DOUBLE PRECISION (dspgvx の場合)
固有値に対する絶対エラー許容値。
詳細は、「アプリケーション・ノート」を参照。

ldz INTEGER。出力配列 *z* のリーディング・ディメンジョン。
次の制約がある。
ldz ≥ 1。*jobz* ='V' の場合、*ldz* ≥ max(1, *n*)

iwork INTEGER
ワークスペース配列 DIMENSION は、max(1, 5*n*) 以上。

出力パラメータ

ap 終了時、*ap* の内容は上書きされる。

bp 終了時、コルスキー因子分解 $B = U^T U$ または $B = L L^T$ からの三角係数 U または L が、B と同じ格納形式で格納される。

m INTEGER。検出された固有値の総数 (0 ≤ *m* ≤ *n*)。
range ='A' の場合は *m* = *n*、*range* ='I' の場合は *m* = *iu*-*il*+1 である。

w, z REAL (sspgvx の場合)
DOUBLE PRECISION (dspgvx の場合)
配列 :
w(*)、DIMENSION は、max(1, *n*) 以上。
info = 0 の場合は、固有値が昇順で格納される。
z(*ldz*, *)。*z* の第 2 次元は、max(1, *n*) 以上でなければならない。
jobz ='V' を指定した場合に *info* = 0 であると、*w*(*i*) に関連する固有ベクトルを格納している *z* の *i* 番目の列とともに、指定された固有値に対応するマトリックス A の正規直交固有ベクトルが、*z* の先頭から *m* 個の列に格納される。固有ベクトルは、次のように正規化される。
itype = 1 または 2 の場合、$Z^T B Z = I$
itype = 3 の場合、$Z^T B^{-1} Z = I$
jobz ='N' の場合、*z* は参照されない。
固有ベクトルが収束できない場合、*z* のその列には、その固有ベクトルに対する最新の近似値が格納され、固有ベクトルのインデックスは *ifail* に戻される。

注 : 配列 *z* には、max(1, *m*) 以上の列を指定する必要がある。*range* = 'V' の場合、*m* の正確な値は事前にはわからないため、上限を使用する必要がある。

ifail INTEGER

配列 DIMENSION は、max(1, *n*) 以上。
jobz = 'V' を指定した場合に *info* = 0 であると、*ifail* の先頭から *m* 個の成分はゼロになる。*info* > 0 の場合、*ifail* には、収束しなかった固有ベクトルのインデックスが格納される。
jobz = 'N' の場合、*ifail* は参照されない。

info INTEGER

info = 0 の場合は、正常に終了している。
info = -*i* の場合は、*i* 番目の引数の値が不正である。
info > 0 の場合は、spptrf/dpptrf と sspevx/dspevx により、エラー・コードが戻される。

info = *i* ≤ *n* の場合は、sspevx/dspevx が収束せず、*i* 個の固有ベクトルが収束しなかった。これらのインデックスは、配列 *ifail* に格納される。
info = *n* + *i*(1 ≤ *i* ≤ *n*) の場合は、*B* について先頭から次数 *i* の小行列が正値でない。また、*B* の因子分解が完了できず、固有値または固有ベクトルが計算されなかった。

アプリケーション・ノート

abstol + ε * max(|a|, |b|) 以下の幅の区間 [a, b] 内に存在していると判別された場合 (ε はマシン精度)、近似固有値は、収束したものとして受け入れられる。*abstol* がゼロ以下の場合、$\varepsilon * \|T\|_1$ は所定の場所で使用される (*T* は、*A* を三重対角形式に縮退して獲得した三重対角マトリックス)。固有値が最も正確に計算されるのは、*abstol* にアンダーフローしきい値 2*?lamch('S') (ゼロ以外) の 2 倍の値を設定したときである。このルーチンで *info* > 0 が戻された場合は、収束しなかった固有ベクトルがいくつかあるのを示している。その場合は、*abstol* に 2*?lamch('S') を設定する。

?hpgvx

パックド格納形式のマトリックスに関する汎用エルミート固有値問題について、指定された固有値を計算し、オプションで、指定された固有ベクトルを計算する。

```
call chpgvx ( itype, jobz, range, uplo, n, ap, bp, vl, vu, il, iu,
              abstol, m, w, z, ldz, work, rwork, iwork, ifail, info )
call zhpgvx ( itype, jobz, range, uplo, n, ap, bp, vl, vu, il, iu,
              abstol, m, w, z, ldz, work, rwork, iwork, ifail, info )
```

説明

このルーチンは、次の形式で示される複素数の汎用エルミート固有値問題について、指定された固有値を計算し、オプションで、指定された固有ベクトルを計算する。

$$Ax = \lambda Bx、ABx = \lambda x、または\ BAx = \lambda x$$

A と B はエルミートであり、パックド形式で格納されると仮定する。また、B は正値である。
固有値と固有ベクトルを選択するには、希望する固有値の値の範囲かインデックスの範囲を指定する。

入力パラメータ

itype INTEGER。1、2、または 3 でなければならない。
解決する問題のタイプを指定する。
itype = 1 の場合、問題のタイプは $Ax = \lambda Bx$ である。
itype = 2 の場合、問題のタイプは $ABx = \lambda x$ である。
itype = 3 の場合、問題のタイプは $BAx = \lambda x$ である。

jobz CHARACTER*1。'N' または 'V' でなければならない。
jobz ='N' の場合、固有値のみを計算する。
jobz ='V' の場合、固有値と固有ベクトルを計算する。

range CHARACTER*1。'A'、'V'、または 'I' でなければならない。
range ='A' の場合、ルーチンは固有値をすべて計算する。
range ='V' の場合、ルーチンは半開区間 $vl < \lambda_i \le vu$ における固有値 λ_i を計算する。

range = 'I' の場合、ルーチンはインデックスが il ～ iu である固有値を計算する。

uplo　CHARACTER*1。'U' または 'L' でなければならない。
uplo = 'U' の場合、配列 ap と bp は、A と B の上三角を格納する。
uplo = 'L' の場合、配列 ap と bp は、A と B の下三角を格納する。

n　INTEGER。マトリックス A と B の次数 ($n \geq 0$)。

ap, bp, work　COMPLEX (chpgvx の場合)
DOUBLE COMPLEX (zhpgvx の場合)
配列 :
ap(*) には、uplo の値に従って、エルミート・マトリックス A の上三角または下三角がパックド形式で格納される。ap の次元は、max(1, n*(n+1)/2) 以上でなければならない。
bp(*) には、uplo の値に従って、エルミート・マトリックス B の上三角または下三角がパックド形式で格納される。bp の次元は、max(1, n*(n+1)/2) 以上でなければならない。
work(*) は、ワークスペース配列 DIMENSION であり、max(1, 2n) 以上。

vl, vu　REAL (chpgvx の場合)
DOUBLE PRECISION (zhpgvx の場合)
range = 'V' の場合、固有値を検索する区間の上限と下限を指定する。
次の制約がある。
vl < vu
range = 'A' または 'I' の場合、vl と vu は参照されない。

il, iu　INTEGER
range = 'I' の場合、戻される最小固有値と最大固有値に対する昇順のインデックス。次の制約がある。
$1 \leq il \leq iu \leq n$ ($n > 0$ の場合)、il=1 と iu=0 ($n = 0$ の場合)
range = 'A' または 'V' の場合、il と iu は参照されない。

abstol　REAL (chpgvx の場合)
DOUBLE PRECISION (zhpgvx の場合)
固有値に対する絶対エラー許容値。
詳細は、「アプリケーション・ノート」を参照。

ldz INTEGER。出力配列 *z* のリーディング・ディメンジョン。
ldz ≥ 1。*jobz* = 'V' の場合、*ldz* ≥ max(1, *n*)

rwork REAL (chpgvx の場合)
DOUBLE PRECISION (zhpgvx の場合)
ワークスペース配列 DIMENSION は、max(1, 7*n*) 以上。

iwork INTEGER
ワークスペース配列 DIMENSION は、max(1, 5*n*) 以上。

出力パラメータ

ap 終了時、*ap* の内容は上書きされる。

bp 終了時、コルスキー因子分解 $B = U^H U$ または $B = L L^H$ からの三角係数 U または L が、B と同じ格納形式で格納される。

m INTEGER。検出された固有値の総数 (0 ≤ *m* ≤ *n*)。
range = 'A' の場合は *m* = *n*、*range* = 'I' の場合は *m* = *iu*-*il*+1 である。

w REAL (chpgvx の場合)
DOUBLE PRECISION (zhpgvx の場合)
配列 DIMENSION は、max(1, *n*) 以上。
info = 0 の場合は、固有値が昇順で格納される。

z COMPLEX (chpgvx の場合)
DOUBLE COMPLEX (zhpgvx の場合)
配列 *z(ldz, *)*。*z* の第 2 次元は、max(1, *n*) 以上でなければならない。
jobz = 'V' を指定した場合に *info* = 0 であると、*w*(*i*) に関連する固有ベクトルを格納している *z* の *i* 番目の列とともに、指定された固有値に対応するマトリックス A の正規直交固有ベクトルが、*z* の先頭から *m* 個の列に格納される。固有ベクトルは、次のように正規化される。
itype = 1 または 2 の場合、$Z^H B Z = I$
itype = 3 の場合、$Z^H B^{-1} Z = I$
jobz = 'N' の場合、*z* は参照されない。

固有ベクトルが収束できない場合、*z* のその列には、その固有ベクトルに対する最新の近似値が格納され、固有ベクトルのインデックスは *ifail* に戻される。
注 : 配列 *z* には、max(1, *m*) 以上の列を指定する必要がある。*range* = 'V' の場合、*m* の正確な値は事前にはわからないため、上限を使用する必要がある。

ifail　INTEGER

配列 DIMENSION は、max(1, *n*) 以上。
jobz = 'V' を指定した場合に *info* = 0 であると、*ifail* の先頭から *m* 個の成分はゼロになる。
info > 0 の場合、*ifail* には、収束しなかった固有ベクトルのインデックスが格納される。
jobz = 'N' の場合、*ifail* は参照されない。

info　INTEGER

info = 0 の場合は、正常に終了している。
info = -*i* の場合は、*i* 番目の引数の値が不正である。
info > 0 の場合は、cpptrf/zpptrf と chpevx/zhpevx により、エラー・コードが戻される。

info = *i* ≤ *n* の場合は、chpevx/zhpevx が収束せず、*i* 個の固有ベクトルが収束しなかった。これらのインデックスは、配列 *ifail* に格納される。
info = *n* + *i*(1 ≤ *i* ≤ *n*) の場合は、*B* について先頭から次数 *i* の小行列が正値でない。また、*B* の因子分解が完了できず、固有値または固有ベクトルが計算されなかった。

アプリケーション・ノート

abstol + ε * max(|a|, |b|) 以下の幅の区間 [a, b] 内に存在していると判別された場合 (ε はマシン精度)、近似固有値は、収束したものとして受け入れられる。*abstol* がゼロ以下の場合、$\varepsilon * \|T\|_1$ は所定の場所で使用される (*T* は、*A* を三重対角形式に縮退して獲得した三重対角マトリックス)。固有値が最も正確に計算されるのは、*abstol* にアンダーフローしきい値 2*?lamch('S') (ゼロ以外) の 2 倍の値を設定したときである。このルーチンで *info* > 0 が戻された場合は、収束しなかった固有ベクトルがいくつかあるのを示している。その場合は、*abstol* に 2*?lamch('S') を設定する。

?sbgv

バンド・マトリックスに関する実数の汎用対称固有値問題について、すべての固有値を計算し、オプションですべての固有ベクトルを計算する。

```
call ssbgv ( jobz, uplo, n, ka, kb, ab, ldab, bb, ldbb, w, z, ldz,
             work, info )
call dsbgv ( jobz, uplo, n, ka, kb, ab, ldab, bb, ldbb, w, z, ldz,
             work, info )
```

説明

このルーチンは、$Ax = \lambda Bx$ の形式で示される実数の汎用対称バンド固有値問題について、すべての固有値を計算し、オプションですべての固有ベクトルを計算する。A と B は対称とバンドであると仮定する。また、B は正値である。

入力パラメータ

jobz CHARACTER*1。'N' または 'V' でなければならない。
jobz ='N' の場合、固有値のみを計算する。
jobz ='V' の場合、固有値と固有ベクトルを計算する。

uplo CHARACTER*1。'U' または 'L' でなければならない。
uplo ='U' の場合、配列 *ab* と *bb* は、A と B の上三角を格納する。
uplo ='L' の場合、配列 *ab* と *bb* は、A と B の下三角を格納する。

n INTEGER。マトリックス A と B の次数 ($n \geq 0$)。

ka INTEGER。A の優対角成分または劣対角成分の数 ($ka \geq 0$)。

kb INTEGER。B の優対角成分または劣対角成分の数 ($kb \geq 0$)。

ab, bb, work
REAL (ssbgv の場合)
DOUBLE PRECISION (dsbgv の場合)
配列 :
ab (*ldab*, *)) は、対称マトリックス A の上三角部分または下三角部分を (*uplo* の値に従って) バンド格納形式で格納する配列である。
配列 *ab* の第 2 次元は、max(1, *n*) 以上でなければならない。
bb (*ldbb*, *)) は、対称マトリックス B の上三角部分または下三角部分を (*uplo* の値に従って) バンド格納形式で格納する配列である。
配列 *bb* の第 2 次元は、max(1, *n*) 以上でなければならない。
work(*)) は、ワークスペース配列 DIMENSION であり、max(1, 3*n*) 以上。

ldab
INTEGER。配列 *ab* の第 1 次元。*ka*+1 以上でなければならない。

ldbb
INTEGER。配列 *bb* の第 1 次元。*kb*+1 以上でなければならない。

ldz
INTEGER。出力配列 *z* のリーディング・ディメンジョン。
ldz ≥ 1 である。*jobz* ='V' の場合、*ldz* ≥ max(1, *n*) である。

出力パラメータ

ab
終了時、*ab* の内容は上書きされる。

bb
終了時、spbstf/dpbstf で戻された分割コルスキー因子分解 $B = S^T S$ からの係数 S が格納される。

w, z
REAL (ssbgv の場合)
DOUBLE PRECISION (dsbgv の場合)
配列 :
w(*)、DIMENSION は、max(1, *n*) 以上。
info = 0 の場合は、固有値が昇順で格納される。
z(*ldz*, *)。*z* の第 2 次元は、max(1, *n*) 以上でなければならない。
jobz ='V' を指定した場合に *info* = 0 であると、*z* には、*w*(*i*) に関連する固有ベクトルを格納している *z* の *i* 番目の列とともに、固有ベクトルのマトリックス Z が格納される。固有ベクトルは、$Z^T B Z = I$ のように正規化される。

jobz = 'N' の場合、*z* は参照されない。

info INTEGER

info = 0 の場合は、正常に終了している。

info = -*i* の場合は、*i* 番目の引数の値が不正である。

info > 0 であり、

i ≤ *n* の場合は、アルゴリズムが収束せず、中間三重対角の *i* 個の対角外の成分がゼロに収束しなかった。

info = *n* + *i*(1 ≤ *i* ≤ *n*) の場合は、spbstf/dpbstf により *info* = *i* が戻され、*B* が正値でない。また、*B* の因子分解が完了できず、固有値または固有ベクトルが計算されなかった。

?hbgv

バンド・マトリックスに関する複素数の
汎用エルミート固有値問題について、
すべての固有値を計算し、オプションで
すべての固有ベクトルを計算する。

```
call chbgv ( jobz, uplo, n, ka, kb, ab, ldab, bb, ldbb, w, z, ldz,
             work, rwork, info )
call zhbgv ( jobz, uplo, n, ka, kb, ab, ldab, bb, ldbb, w, z, ldz,
             work, rwork, info )
```

説明

このルーチンは、$Ax = \lambda Bx$ の形式で示される複素数の汎用エルミート・バンド固有値問題について、すべての固有値を計算し、オプションですべての固有ベクトルを計算する。A と B はエルミートとバンドであると仮定する。また、B は正値である。

入力パラメータ

jobz CHARACTER*1。'N' または 'V' でなければならない。
jobz ='N' の場合、固有値のみを計算する。
jobz ='V' の場合、固有値と固有ベクトルを計算する。

uplo CHARACTER*1。'U' または 'L' でなければならない。
uplo ='U' の場合、配列 *ab* と *bb* は、A と B の上三角を格納する。
uplo ='L' の場合、配列 *ab* と *bb* は、A と B の下三角を格納する。

n INTEGER。マトリックス A と B の次数 ($n \geq 0$)。

ka INTEGER。A の優対角成分または劣対角成分の数 ($ka \geq 0$)。

kb INTEGER。B の優対角成分または劣対角成分の数 ($kb \geq 0$)。

ab, bb, work
: COMPLEX (chbgv の場合)
DOUBLE COMPLEX (zhbgv の場合)
配列 :
ab (*ldab*, *) は、エルミート・マトリックス A の上三角部分または下三角部分を (*uplo* の値に従って) バンド格納形式で格納する配列である。
配列 *ab* の第 2 次元は、max(1, *n*) 以上でなければならない。
bb (*ldbb*, *) は、エルミート・マトリックス B の上三角部分または下三角部分を (*uplo* の値に従って) バンド格納形式で格納する配列である。
配列 *bb* の第 2 次元は、max(1, *n*) 以上でなければならない。
work(*) は、ワークスペース配列 DIMENSION であり、max(1, *n*) 以上。

ldab
: INTEGER。配列 *ab* の第 1 次元。*ka*+1 以上でなければならない。

ldbb
: INTEGER。配列 *bb* の第 1 次元。*kb*+1 以上でなければならない。

ldz
: INTEGER。出力配列 *z* のリーディング・ディメンジョン。
ldz ≥ 1 である。*jobz* = 'V' の場合、*ldz* ≥ max(1, *n*) である。

rwork
: REAL (chbgv の場合)
DOUBLE PRECISION (zhbgv の場合)
ワークスペース配列 DIMENSION は、max(1, 3*n*) 以上。

出力パラメータ

ab
: 終了時、*ab* の内容は上書きされる。

bb
: 終了時、cpbstf/zpbstf で戻された分割コルスキー因子分解 $B = S^H S$ からの係数 S が格納される。

w
: REAL (chbgv の場合)
DOUBLE PRECISION (zhbgv の場合)
配列 DIMENSION は、max(1, *n*) 以上。
info = 0 の場合は、固有値が昇順で格納される。

z COMPLEX (chbgv の場合)
DOUBLE COMPLEX (zhbgv の場合)
配列 *z(ldz, *)*。*z* の第 2 次元は、max(1, *n*) 以上でなければならない。
jobz ='V' を指定した場合に *info* = 0 であると、*z* には、*w*(*i*) に関連する固有ベクトルを格納している *z* の *i* 番目の列とともに、固有ベクトルのマトリックス Z が格納される。固有ベクトルは、$Z^H B Z = I$ のように正規化される。
jobz ='N' の場合、*z* は参照されない。

info INTEGER
info = 0 の場合は、正常に終了している。
info = -*i* の場合は、*i* 番目の引数の値が不正である。
info > 0 であり、
i ≤ *n* の場合は、アルゴリズムが収束せず、中間三重対角の *i* 個の対角外の成分がゼロに収束しなかった。
info = *n* + *i*(1 ≤ *i* ≤ *n*) の場合は、cpbstf/zpbstf により *info* = *i* が戻され、B が正値でない。また、B の因子分解が完了できず、固有値または固有ベクトルが計算されなかった。

?sbgvd

バンド・マトリックスに関する実数の汎用対称固有値問題について、すべての固有値を計算し、オプションですべての固有ベクトルを計算する。固有ベクトルを計算する場合は、分割統治法を使用する。

```
call ssbgvd ( jobz, uplo, n, ka, kb, ab, ldab, bb, ldbb, w, z, ldz,
              work, lwork, iwork, liwork, info )
call dsbgvd ( jobz, uplo, n, ka, kb, ab, ldab, bb, ldbb, w, z, ldz,
              work, lwork, iwork, liwork, info )
```

説明

このルーチンは、$Ax = \lambda Bx$ の形式で示される実数の汎用対称バンド固有値問題について、すべての固有値を計算し、オプションですべての固有ベクトルを計算する。A と B は対称とバンドであると仮定する。また、B は正値である。固有ベクトルを計算する場合は、分割統治アルゴリズムを使用する。

入力パラメータ

jobz CHARACTER*1。'N' または 'V' でなければならない。
jobz ='N' の場合、固有値のみを計算する。
jobz ='V' の場合、固有値と固有ベクトルを計算する。

uplo CCHARACTER*1。'U' または 'L' でなければならない。
uplo ='U' の場合、配列 *ab* と *bb* は、A と B の上三角を格納する。
uplo ='L' の場合、配列 *ab* と *bb* は、A と B の下三角を格納する。

n INTEGER。マトリックス A と B の次数 ($n \geq 0$)。

ka INTEGER。A の優対角成分または劣対角成分の数 ($ka \geq 0$)。

kb INTEGER。*B* の優対角成分または劣対角成分の数 (*kb* ≥ 0)。

ab, bb, work REAL (ssbgvd の場合)
DOUBLE PRECISION (dsbgvd の場合)
配列 :
ab (*ldab*, *) は、対称マトリックス *A* の上三角部分または下三角部分を (*uplo* の値に従って) バンド格納形式で格納する配列である。
配列 *ab* の第 2 次元は、max(1, *n*) 以上でなければならない。
bb (*ldbb*, *) は、対称マトリックス *B* の上三角部分または下三角部分を (*uplo* の値に従って) バンド格納形式で格納する配列である。
配列 *bb* の第 2 次元は、max(1, *n*) 以上でなければならない。
work(*lwork*) は、ワークスペース配列である。

ldab INTEGER。配列 *ab* の第 1 次元。*ka*+1 以上でなければならない。

ldbb INTEGER。配列 *bb* の第 1 次元。*kb*+1 以上でなければならない。

ldz INTEGER。出力配列 *z* のリーディング・ディメンジョン。
ldz ≥ 1 である。*jobz* ='V' の場合、*ldz* ≥ max(1, *n*) である。

lwork INTEGER。配列 *work* の次元。
次の制約がある。
n ≤ 1 の場合、*lwork* ≥ 1
jobz ='N' かつ *n* > 1 の場合、*lwork* ≥ 3*n*
jobz ='V' かつ *n* > 1 の場合、*lwork* ≥ $2n^2+5n+1$

iwork INTEGER
ワークスペース配列 DIMENSION(*liwork*)

liwork INTEGER。配列 *iwork* の次元。
次の制約がある。
n ≤ 1 の場合、*liwork* ≥ 1
jobz ='N' かつ *n* > 1 の場合、*liwork* ≥ 1
jobz ='V' かつ *n* > 1 の場合、*liwork* ≥ 5*n*+3

出力パラメータ

ab
終了時、*ab* の内容は上書きされる。

bb
終了時、spbstf/dpbstf で戻された分割コルスキー因子分解 $B = S^T S$ からの係数 S が格納される。

w, z
REAL (ssbgvd の場合)
DOUBLE PRECISION (dsbgvd の場合)
配列 :
w(*)、DIMENSION は、max(1, *n*) 以上。
info = 0 の場合は、固有値が昇順で格納される。
z(*ldz*, *)。*z* の第 2 次元は、max(1, *n*) 以上でなければならない。
jobz = 'V' を指定した場合に *info* = 0 であると、*z* には、*w*(*i*) に関連する固有ベクトルを格納している *z* の *i* 番目の列とともに、固有ベクトルのマトリックス Z が格納される。固有ベクトルは、$Z^T B Z = I$ のように正規化される。
jobz = 'N' の場合、*z* は参照されない。

work(1)
終了時に *info* = 0 の場合、*work(1)* は、*lwork* の必要最小サイズを戻す。

iwork(1)
終了時に *info* = 0 の場合、*iwork(1)* は、*liwork* の必要最小サイズを戻す。

info
INTEGER
info = 0 の場合は、正常に終了している。
info = -*i* の場合は、*i* 番目の引数の値が不正である。
info > 0 であり、
$i \le n$ の場合は、アルゴリズムが収束せず、中間三重対角の *i* 個の対角外の成分がゼロに収束しなかった。
info = *n* + *i*($1 \le i \le n$) の場合は、spbstf/dpbstf により *info* = *i* が戻され、B が正値でない。また、B の因子分解が完了できず、固有値または固有ベクトルが計算されなかった。

?hbgvd

バンド・マトリックスに関する複素数の汎用エルミート固有値問題について、すべての固有値を計算し、オプションですべての固有ベクトルを計算する。固有ベクトルを計算する場合は、分割統治法を使用する。

```
call chbgvd ( jobz, uplo, n, ka, kb, ab, ldab, bb, ldbb, w, z, ldz,
              work, lwork, rwork, lrwork, iwork, liwork, info )
call zhbgvd ( jobz, uplo, n, ka, kb, ab, ldab, bb, ldbb, w, z, ldz,
              work, lwork, rwork, lrwork, iwork, liwork, info )
```

説明

このルーチンは、$Ax = \lambda Bx$ の形式で示される複素数の汎用エルミート・バンド固有値問題について、すべての固有値を計算し、オプションですべての固有ベクトルを計算する。A と B はエルミートとバンドであると仮定する。また、B は正値である。固有ベクトルを計算する場合は、分割統治アルゴリズムを使用する。

入力パラメータ

jobz CHARACTER*1。'N' または 'V' でなければならない。
jobz ='N' の場合、固有値のみを計算する。
jobz ='V' の場合、固有値と固有ベクトルを計算する。

uplo CHARACTER*1。'U' または 'L' でなければならない。
uplo ='U' の場合、配列 *ab* と *bb* は、A と B の上三角を格納する。
uplo ='L' の場合、配列 *ab* と *bb* は、A と B の下三角を格納する。

n INTEGER。マトリックス A と B の次数 ($n \geq 0$)。

ka INTEGER。A の優対角成分または劣対角成分の数 ($ka \geq 0$)。

kb	INTEGER。*B* の優対角成分または劣対角成分の数 (*kb* ≥ 0)。
ab, bb, work	COMPLEX (chbgvd の場合) DOUBLE COMPLEX (zhbgvd の場合) 配列 : *ab* (*ldab, **) は、エルミート・マトリックス *A* の上三角部分または下三角部分を (*uplo* の値に従って) バンド格納形式で格納する配列である。 配列 *ab* の第 2 次元は、max(1, *n*) 以上でなければならない。 *bb* (*ldbb, **) は、エルミート・マトリックス *B* の上三角部分または下三角部分を (*uplo* の値に従って) バンド格納形式で格納する配列である。 配列 *bb* の第 2 次元は、max(1, *n*) 以上でなければならない。 *work(lwork)* は、ワークスペース配列である。
ldab	INTEGER。配列 *ab* の第 1 次元。*ka*+1 以上でなければならない。
ldbb	INTEGER。配列 *bb* の第 1 次元。*kb*+1 以上でなければならない。
ldz	INTEGER。出力配列 *z* のリーディング・ディメンジョン。 *ldz* ≥ 1 である。*jobz* = 'V' の場合、 *ldz* ≥ max(1, *n*) である。
lwork	INTEGER。配列 *work* の次元。 次の制約がある。 *n* ≤ 1 の場合、*lwork* ≥ 1 *jobz* = 'N' かつ *n* > 1 の場合、*lwork* ≥ *n* *jobz* = 'V' かつ *n* > 1 の場合、*lwork* ≥ $2n^2$
rwork	REAL (chbgvd の場合) DOUBLE PRECISION (zhbgvd の場合) ワークスペース配列 DIMENSION (*lrwork*)
lrwork	INTEGER。配列 *rwork* の次元。

次の制約がある。
$n \le 1$ の場合、*lrwork* ≥ 1
jobz = 'N' かつ $n > 1$ の場合、*lrwork* $\ge n$
jobz = 'V' かつ $n > 1$ の場合、*lrwork* $\ge 2n^2+5n+1$

iwork INTEGER
ワークスペース配列 DIMENSION(*liwork*)

liwork INTEGER。配列 *iwork* の次元。
次の制約がある。
$n \le 1$ の場合、*liwork* ≥ 1
jobz = 'N' かつ $n > 1$ の場合、*liwork* ≥ 1
jobz = 'V' かつ $n > 1$ の場合、*liwork* $\ge 5n+3$

出力パラメータ

ab 終了時、*ab* の内容は上書きされる。

bb 終了時、cpbstf/zpbstf で戻された分割コルスキー因子分解 $B = S^H S$ からの係数 S が格納される。

w REAL (chbgvd の場合)
DOUBLE PRECISION (zhbgvd の場合)
配列 DIMENSION は、max(1, *n*) 以上。
info = 0 の場合は、固有値が昇順で格納される。

z COMPLEX (chbgvd の場合)
DOUBLE COMPLEX (zhbgvd の場合)
配列 *z(ldz, *)*。*z* の第 2 次元は、max(1, *n*) 以上でなければならない。
jobz = 'V' を指定した場合に *info* = 0 であると、*z* には、*w*(*i*) に関連する固有ベクトルを格納している *z* の *i* 番目の列とともに、固有ベクトルのマトリックス Z が格納される。固有ベクトルは、$Z^H B Z = I$ のように正規化される。
jobz = 'N' の場合、*z* は参照されない。

work(1) 終了時に *info* = 0 の場合、*work(1)* は、*lwork* の必要最小サイズを戻す。

rwork(1) 終了時に *info* = 0 の場合、*rwork(1)* は、*lrwork* の必要最小サイズを戻す。

iwork(1) 終了時に *info* = 0 の場合、*iwork(1)* は、*liwork* の必要最小サイズを戻す。

info INTEGER

info = 0 の場合は、正常に終了している。

info = -*i* の場合は、*i* 番目の引数の値が不正である。

info > 0 であり、

i ≤ *n* の場合は、アルゴリズムが収束せず、中間三重対角の *i* 個の対角外の成分がゼロに収束しなかった。

info = *n* + *i*(1 ≤ *i* ≤ *n*) の場合は、cpbstf/zpbstf により *info* = *i* が戻され、*B* が正値でない。また、*B* の因子分解が完了できず、固有値または固有ベクトルが計算されなかった。

?sbgvx

バンド・マトリックスに関する実数の
汎用対称固有値問題について、指定された
固有値を計算し、オプションで、
指定された固有ベクトルを計算する。

```
call ssbgvx ( jobz, range, uplo, n, ka, kb, ab, ldab, bb, ldbb, q,
              ldq, vl, vu, il, iu, abstol, m, w, z, ldz, work, iwork,
              ifail, info )
call dsbgvx ( jobz, range, uplo, n, ka, kb, ab, ldab, bb, ldbb, q,
              ldq, vl, vu, il, iu, abstol, m, w, z, ldz, work, iwork,
              ifail, info )
```

説明

このルーチンは、$Ax = \lambda Bx$ の形式で示される実数の汎用対称バンド固有値問題について、指定された固有値を計算し、オプションで、指定された固有ベクトルを計算する。A と B は対称とバンドであると仮定する。また、B は正値である。
固有値と固有ベクトルを選択するには、希望する固有値について、すべての固有値、値の範囲、またはインデックスの範囲を指定する。

入力パラメータ

jobz CHARACTER*1。'N' または 'V' でなければならない。
jobz ='N' の場合、固有値のみを計算する。
jobz ='V' の場合、固有値と固有ベクトルを計算する。

range CHARACTER*1。'A'、'V'、または 'I' でなければならない。
range ='A' の場合、ルーチンは固有値をすべて計算する。
range ='V' の場合、ルーチンは半開区間 $vl < \lambda_i \le vu$ における固有値 λ_i を計算する。
range ='I' の場合、ルーチンはインデックスが *il* ～ *iu* である固有値を計算する。

uplo CHARACTER*1。'U' または 'L' でなければならない。

	uplo = 'U' の場合、配列 *ab* と *bb* は、*A* と *B* の上三角を格納する。 *uplo* = 'L' の場合、配列 *ab* と *bb* は、*A* と *B* の下三角を格納する。
n	INTEGER。マトリックス *A* と *B* の次数 ($n \geq 0$)。
ka	INTEGER。*A* の優対角成分または劣対角成分の数 ($ka \geq 0$)。
kb	INTEGER。*B* の優対角成分または劣対角成分の数 ($kb \geq 0$)。
ab, bb, work	REAL (ssbgvx の場合) DOUBLE PRECISION (dsbgvx の場合) 配列 : *ab* (*ldab*, *) は、対称マトリックス *A* の上三角部分または下三角部分を (*uplo* の値に従って) バンド格納形式で格納する配列である。 配列 *ab* の第 2 次元は、max(1, *n*) 以上でなければならない。 *bb* (*ldbb*, *) は、対称マトリックス *B* の上三角部分または下三角部分を (*uplo* の値に従って) バンド格納形式で格納する配列である。 配列 *bb* の第 2 次元は、max(1, n) 以上でなければならない。 *work*(*) は、ワークスペース配列 DIMENSION であり、max(1, 7*n*) 以上。
ldab	INTEGER。配列 *ab* の第 1 次元。*ka*+1 以上でなければならない。
ldbb	INTEGER。配列 *bb* の第 1 次元。*kb*+1 以上でなければならない。
vl, vu	REAL (ssbgvx の場合) DOUBLE PRECISION (dsbgvx の場合) *range* = 'V' の場合、固有値を検索する区間の上限と下限を指定する。 次の制約がある。 *vl* < *vu* *range* = 'A' または 'I' の場合、*vl* と *vu* は参照されない。

il, iu INTEGER
range = 'I' の場合、戻される最小固有値と最大固有値に対する昇順のインデックス。次の制約がある。
$1 \le il \le iu \le n$($n > 0$ の場合)、*il*=1 および *iu* = 0 ($n = 0$ の場合)。
range = 'A' または 'V' の場合、*il* と *iu* は参照されない。

abstol REAL (ssbgvx の場合)
DOUBLE PRECISION (dsbgvx の場合)
固有値に対する絶対エラー許容値。
詳細は、「アプリケーション・ノート」を参照。

ldz INTEGER。出力配列 *z* のリーディング・ディメンジョン。
$ldz \ge 1$。*jobz* = 'V' の場合、$ldz \ge \max(1, n)$

ldq INTEGER。出力配列 *q* のリーディング・ディメンジョン。
$ldq \ge 1$。*jobz* = 'V' の場合、$ldq \ge \max(1, n)$

iwork INTEGER
ワークスペース配列 DIMENSION は、$\max(1, 5n)$ 以上。

出力パラメータ

ab 終了時、*ab* の内容は上書きされる。

bb 終了時、spbstf/dpbstf で戻された分割コルスキー因子分解 $B = S^T S$ からの係数 S が格納される。

m INTEGER。検出された固有値の総数 ($0 \le m \le n$)。
range = 'A' の場合は $m = n$、*range* = 'I' の場合は $m = iu - il + 1$ である。

w, z, q REAL (ssbgvx の場合)
DOUBLE PRECISION (dsbgvx の場合)
配列 :
w(*)、DIMENSION は、$\max(1, n)$ 以上。
info = 0 の場合は、固有値が昇順で格納される。
*z(ldz, *)*。*z* の第 2 次元は、$\max(1, n)$ 以上でなければならない。
jobz = 'V' を指定した場合に *info* = 0 であると、*z* には、*w*(*i*) に関連する固有ベクトルを格納している *z* の *i* 番目の列とともに、固有ベクトルのマトリックス Z が格納される。固有ベクトルは、$Z^T B Z = I$ のように正規化される。

jobz = 'N' の場合、z は参照されない。

q(ldq, *)。q の第 2 次元は、max(1, n) 以上でなければならない。

jobz = 'V' の場合、q には、$Ax = \lambda Bx$ を標準形式、すなわち $Cx = \lambda x$ に縮退させて、C を三重対角形式に縮退させるために使用する $n \times n$ のマトリックスが格納される。

jobz = 'N' の場合、q は参照されない。

ifail INTEGER

配列 DIMENSION は、max(1, n) 以上。

jobz = 'V' を指定した場合に info = 0 であると、ifail の先頭から m 個の成分はゼロになる。

info > 0 の場合、ifail には、収束しなかった固有ベクトルのインデックスが格納される。

jobz = 'N' の場合、ifail は参照されない。

info INTEGER

info = 0 の場合は、正常に終了している。

info = -i の場合は、i 番目の引数の値が不正である。

info > 0 であり、

- $i \le n$ の場合は、アルゴリズムが収束せず、中間三重対角の i 個の対角外の成分がゼロに収束しなかった。
- $info = n + i (1 \le i \le n)$ の場合は、spbstf/dpbstf により info = i が戻され、B が正値でない。また、B の因子分解が完了できず、固有値または固有ベクトルが計算されなかった。

アプリケーション・ノート

abstol + ε * max(|a|, |b|) 以下の幅の区間 [a, b] 内に存在していると判別された場合 (ε はマシン精度)、近似固有値は、収束したものとして受け入れられる。abstol がゼロ以下の場合、$\varepsilon * \|T\|_1$ は所定の場所で使用される (T は、A を三重対角形式に縮退して獲得した三重対角マトリックス)。

固有値が最も正確に計算されるのは、abstol にアンダーフローしきい値 2*?lamch('S') (ゼロ以外) の 2 倍の値を設定したときである。このルーチンで info > 0 が戻された場合は、収束しなかった固有ベクトルがいくつかあるのを示している。その場合は、abstol に 2*?lamch('S') を設定する。

?hbgvx

バンド・マトリックスに関する複素数の
汎用エルミート固有値問題について、
指定された固有値を計算し、オプション
で、指定された固有ベクトルを計算する。

```
call chbgvx ( jobz, range, uplo, n, ka, kb, ab, ldab, bb, ldbb, q,
              ldq, vl, vu, il, iu, abstol, m, w, z, ldz, work, rwork,
              iwork, ifail, info )
call zhbgvx ( jobz, range, uplo, n, ka, kb, ab, ldab, bb, ldbb, q,
              ldq, vl, vu, il, iu, abstol, m, w, z, ldz, work, rwork,
              iwork, ifail, info )
```

説明

このルーチンは、$Ax = \lambda Bx$ の形式で示される複素数の汎用エルミート・バンド固有値問題について、指定された固有値を計算し、オプションで、指定された固有ベクトルを計算する。A と B はエルミートとバンドであると仮定する。また、B は正値である。
固有値と固有ベクトルを選択するには、希望する固有値について、すべての固有値、値の範囲、またはインデックスの範囲を指定する。

入力パラメータ

jobz CHARACTER*1。'N' または 'V' でなければならない。
jobz ='N' の場合、固有値のみを計算する。
jobz ='V' の場合、固有値と固有ベクトルを計算する。

range CHARACTER*1。'A'、'V'、または 'I' でなければならない。
range ='A' の場合、ルーチンは固有値をすべて計算する。
range ='V' の場合、ルーチンは半開区間 $vl < \lambda_i \le vu$ における固有値 λ_i を計算する。
range ='I' の場合、ルーチンはインデックスが *il* ～ *iu* である固有値を計算する。

uplo CHARACTER*1。'U' または 'L' でなければならない。

uplo = 'U' の場合、配列 ab と bb は、A と B の上三角を格納する。
uplo = 'L' の場合、配列 ab と bb は、A と B の下三角を格納する。

n — INTEGER。マトリックス A と B の次数 (n ≥ 0)。

ka — INTEGER。A の優対角成分または劣対角成分の数 (ka ≥ 0)。

kb — INTEGER。B の優対角成分または劣対角成分の数 (kb ≥ 0)。

ab, bb, work — COMPLEX (chbgvx の場合)
DOUBLE COMPLEX (zhbgvx の場合)
配列 :
ab (ldab, *) は、エルミート・マトリックス A の上三角部分または下三角部分を (uplo の値に従って) バンド格納形式で格納する配列である。
配列 ab の第 2 次元は、max(1, n) 以上でなければならない。
bb (ldbb, *) は、エルミート・マトリックス B の上三角部分または下三角部分を (uplo の値に従って) バンド格納形式で格納する配列である。
配列 bb の第 2 次元は、max(1, n) 以上でなければならない。
work(*) は、ワークスペース配列 DIMENSION であり、max(1, n) 以上。

ldab — INTEGER。配列 ab の第 1 次元。ka+1 以上でなければならない。

ldbb — INTEGER。配列 bb の第 1 次元。kb+1 以上でなければならない。

vl, vu — REAL (chbgvx の場合)
DOUBLE PRECISION (zhbgvx の場合)
range = 'V' の場合、固有値を検索する区間の上限と下限を指定する。
次の制約がある。
vl < vu
range = 'A' または 'I' の場合、vl と vu は参照されない。

il, iu INTEGER
range = 'I' の場合、戻される最小固有値と最大固有値に対する昇順のインデックス。次の制約がある。
$1 \le il \le iu \le n$($n > 0$ の場合)、*il*=1 および *iu* = 0 ($n = 0$ の場合)
range = 'A' または 'V' の場合、*il* と *iu* は参照されない。

abstol REAL (chbgvx の場合)
DOUBLE PRECISION (zhbgvx の場合)
固有値に対する絶対エラー許容値。
詳細は、「アプリケーション・ノート」を参照。

ldz INTEGER。出力配列 *z* のリーディング・ディメンジョン。
$ldz \ge 1$。*jobz* = 'V' の場合、$ldz \ge \max(1, n)$

ldq INTEGER。出力配列 *q* のリーディング・ディメンジョン。
$ldq \ge 1$。*jobz* = 'V' の場合、$ldq \ge \max(1, n)$

rwork REAL (chbgvx の場合)
DOUBLE PRECISION (zhbgvx の場合)
ワークスペース配列 DIMENSION は、$\max(1, 7n)$ 以上。

iwork INTEGER
ワークスペース配列 DIMENSION は、$\max(1, 5n)$ 以上。

出力パラメータ

ab 終了時、*ab* の内容は上書きされる。

bb 終了時、cpbstf/zpbstf で戻された分割コルスキー因子分解 $B = S^H S$ からの係数 S が格納される。

m INTEGER。検出された固有値の総数 ($0 \le m \le n$)。
range = 'A' の場合は $m = n$、*range* = 'I' の場合は $m = iu - il + 1$ である。

w REAL (chbgvx の場合)
DOUBLE PRECISION (zhbgvx の場合)
配列 *w*(*)、DIMENSION は、$\max(1, n)$ 以上。
info = 0 の場合は、固有値が昇順で格納される。

z, q COMPLEX (chbgvx の場合)
DOUBLE COMPLEX (zhbgvx の場合)
配列 :

z(ldz, *)。z の第 2 次元は、max(1, n) 以上でなければならない。

jobz='V' を指定した場合に info=0 であると、z には、w(i) に関連する固有ベクトルを格納している z の i 番目の列とともに、固有ベクトルのマトリックス Z が格納される。固有ベクトルは、$Z^H B Z = I$ のように正規化される。

jobz='N' の場合、z は参照されない。

q(ldq, *)。q の第 2 次元は、max(1, n) 以上でなければならない。

jobz='V' の場合、q には、$Ax = \lambda Bx$ を標準形式、すなわち $Cx = \lambda x$ に縮退させて、C を三重対角形式に縮退させるために使用する $n \times n$ のマトリックスが格納される。

jobz='N' の場合、q は参照されない。

ifail INTEGER

配列 DIMENSION は、max(1, n) 以上。

jobz='V' を指定した場合に info=0 であると、ifail の先頭から m 個の成分はゼロになる。info>0 の場合、ifail には、収束しなかった固有ベクトルのインデックスが格納される。

jobz='N' の場合、ifail は参照されない。

info INTEGER

info=0 の場合は、正常に終了している。

info=-i の場合は、i 番目の引数の値が不正である。

info>0 であり、

- $i \le n$ の場合は、アルゴリズムが収束せず、中間三重対角の i 個の対角外の成分がゼロに収束しなかった。
- $info = n + i (1 \le i \le n)$ の場合は、cpbstf/zpbstf により info=i が戻され、B が正値でない。また、B の因子分解が完了できず、固有値または固有ベクトルが計算されなかった。

アプリケーション・ノート

abstol + ε * max(|a|, |b|) 以下の幅の区間 [a, b] 内に存在していると判別された場合 (ε はマシン精度)、近似固有値は、収束したものとして受け入れられる。abstol がゼロ以下の場合、$\varepsilon * \|T\|_1$ は所定の場所で使用される (T は、A を三重対角形式に縮退して獲得した三重対角マトリックス)。

固有値が最も正確に計算されるのは、*abstol* にアンダーフローしきい値 2*?lamch('S') (ゼロ以外) の 2 倍の値を設定したときである。このルーチンで *info* > 0 が戻された場合は、収束しなかった固有ベクトルがいくつかあるのを示している。その場合は、*abstol* に 2*?lamch('S') を設定する。

汎用非対称固有値問題

本節では、汎用非対称固有値問題を解くために使用する LAPACK ドライバ・ルーチンについて説明する。これらの問題を解くために呼び出すのが可能な計算ルーチンも参照のこと。
表 5-14 は、ルーチンの詳細について説明している。

表 5-14 汎用非対称固有値問題を解くためのドライバ・ルーチン

ルーチン名	実行する演算
?gges	非対称マトリックスのペアについて、汎用固有値、Schur 形式、左 / 右の Schur ベクトルを計算する。
?ggesx	汎用固有値、Schur 形式を計算し、オプションで Schur ベクトルの左 / 右のマトリックスを計算する。
?ggev	非対称マトリックスのペアについて、汎用固有値、左 / 右の汎用固有ベクトルを計算する。
?ggevx	汎用固有値を計算し、オプションで左 / 右の汎用固有ベクトルを計算する。

?gges

非対称マトリックスのペアについて、汎用固有値、Schur 形式、左 / 右の Schur ベクトルを計算する。

```
call sgges ( jobvsl, jobvsr, sort, selctg, n, a, lda, b, ldb, sdim,
             alphar, alphai, beta, vsl, ldvsl, vsr, ldvsr, work,
             lwork, bwork, info )
call dgges ( jobvsl, jobvsr, sort, selctg, n, a, lda, b, ldb, sdim,
             alphar, alphai, beta, vsl, ldvsl, vsr, ldvsr, work,
             lwork, bwork, info )
call cgges ( jobvsl, jobvsr, sort, selctg, n, a, lda, b, ldb, sdim,
             alpha, beta, vsl, ldvsl, vsr, ldvsr, work, lwork, rwork,
             bwork, info )
call zgges ( jobvsl, jobvsr, sort, selctg, n, a, lda, b, ldb, sdim,
             alpha, beta, vsl, ldvsl, vsr, ldvsr, work, lwork, rwork,
             bwork, info )
```

説明

このルーチンは、$n \times n$ の実数 / 複素数非対称マトリックス (*A, B*) のペアについて、汎用固有値、汎用実数 / 複素数 Schur 形式 (*S, T*) を計算し、オプションで Schur ベクトルの左 / 右のマトリックス (`vsl` と `vsr`) を計算する。これにより、汎用 Schur 因子分解

$$(A, B) = (vsl*S*vsr^H, vsl*T*vsr^H)$$

が提供される。

また、このルーチンは、固有値の選択した束が上準三角マトリックス *S* と上三角マトリックス *T* の主対角ブロックにくるように固有値を順序付けられる。これにより、`vsl` と `vsr` の先頭の列は、対応する左右の固有空間 (収縮部分空間) に対して直交 / ユニタリ基を形成する (汎用固有値だけが必要な場合は、より高速なドライバ `?ggev` を代わりに使用できる)。
マトリックス (*A, B*) のペアの汎用固有値は、*A* - *w***B* が特異であるようなスカラー *w* または比 *alpha* / *beta* = *w* である。*beta*=0 または両方がゼロである妥当な解釈が存在するため、通常は、ペア (*alpha*, *beta*) として表現される。
T が非負の対角を持つ上三角で S が 1 × 1 と 2 × 2 のブロックを持つブロック上三角である場合、マトリックス (*S, T*) のペアは、汎用実数 Schur 形式である。1 × 1 ブロックは実数の汎用固有値に対応するが、*S* の
2 × 2 ブロックは、*T* の対応する成分を次の形式にすることで " 標準化される "。

$$\begin{pmatrix} a & 0 \\ 0 & b \end{pmatrix}$$

また、*S* と *T* の対応する 2 × 2 ブロックのペアは、汎用固有値の複素共役ペアを持つ。
S と *T* が上三角であり、*T* の対角が非負の実数の場合、マトリックス (*S, T*) のペアは、汎用複素数 Schur 形式になる。

入力パラメータ

jobvsl
CHARACTER*1。'N' または 'V' でなければならない。
jobvsl ='N' の場合、左 Schur ベクトルは計算されない。
jobvsl ='V' の場合、左 Schur ベクトルは計算される。

jobvsr
CHARACTER*1。'N' または 'V' でなければならない。
jobvsr ='N' の場合、右 Schur ベクトルは計算されない。
jobvsr ='V' の場合、右 Schur ベクトルは計算される。

sort
CHARACTER*1。'N' または 'S' でなければならない。
汎用 Schur 形式の対角上の固有値を順序付けるかどうかを指定する。
sort ='N' の場合、固有値は順序付けされない。
sort ='S' の場合、固有値は順序付けされる (*selctg* を参照)。

selctg
3 つの REAL 引数の LOGICAL FUNCTION (実数型の場合)
2 つの COMPLEX 引数の LOGICAL FUNCTION (複素数型の場合)
selctg は、呼び出しサブルーチン内で EXTERNAL として宣言する必要がある。
sort ='S' の場合、*selctg* は、Schur 形式の左上に対してソートを実行する固有値を選択するために使用される。
sort ='N' の場合、*selctg* は参照されない。

実数型の場合 :
selctg(*alphar*(j), *alphai*(j), *beta*(j)) が真の場合は、固有値 (*alphar*(j) + *alphai*(j))/*beta*(j) が選択される。つまり、固有値の複素共役ペアの一方が選択されると、複素数の固有値が両方選択される。
悪条件の場合、指定された複素数の固有値は、順序付けの実行後、*selctg*(*alphar*(j), *alphai*(j), *beta*(j)) =.TRUE. を満たさないことがある。その場合は、*info* に *n*+2 を設定する。

複素数型の場合 :
selctg(*alpha*(j), *beta*(j)) が真の場合は、固有値 *alpha*(j) / *beta*(j) が選択される。順序付けを行うと、複素数の固有値の値が変更される可能性がある (特に、固有値が悪い条件の場合)。そのため、順序付けを実行した後、指定された複素数の固有値は、*selctg*(*alpha*(j), *beta*(j)) = .TRUE. を満たさないことがある。その場合は、*info* に *n*+2 が設定される (以下の *info* を参照)。

n — INTEGER。マトリックス *A*、*B*、*vsl*、*vsr* の次数 ($n \geq 0$)。

a, b, work — REAL (sgges の場合)
DOUBLE PRECISION (dgges の場合)
COMPLEX (cgges の場合)
DOUBLE COMPLEX (zgges の場合)
配列 :
a(*lda*, *) は、$n \times n$ のマトリックス *A* (マトリックスのペアの 1 番目) を格納する配列である。
a の第 2 次元は、max(1, *n*) 以上でなければならない。
b(*ldb*, *) は、$n \times n$ のマトリックス *B* (マトリックスのペアの 2 番目) を格納する配列である。
b の第 2 次元は、max(1, *n*) 以上でなければならない。
work(*lwork*) は、ワークスペース配列である。

lda — INTEGER。配列 *a* の第 1 次元。max(1, *n*) 以上でなければならない。

ldb — INTEGER。配列 *b* の第 1 次元。max(1, *n*) 以上でなければならない。

ldvsl, ldvsr — INTEGER。それぞれ、出力マトリックス *vsl* と *vsr* の第 1 次元。
次の制約がある。
ldvsl ≥ 1。*jobvsl* = 'V' の場合、
ldvsl ≥ max(1, *n*)
ldvsr ≥ 1。*jobvsr* = 'V' の場合、
ldvsr ≥ max(1, *n*)

lwork — INTEGER。配列 *work* の次元。

lwork ≥ max(1, 8*n*+16) (実数型の場合)
lwork ≥ max(1, 2*n*) (複素数型の場合)
性能を上げるには、一般に *lwork* を大きくする必要がある。

rwork REAL (cgges の場合)
DOUBLE PRECISION (zgges の場合)
ワークスペース配列 DIMENSION は、max(1, 8*n*) 以上。この配列は、複素数型でのみ使用される。

bwork LOGICAL。
ワークスペース配列 DIMENSION は、max(1, *n*) 以上。
sort ='N' の場合は参照されない。

出力パラメータ

a 終了時、この配列は、その汎用 Schur 形式 *S* で上書きされる。

b 終了時、この配列は、その汎用 Schur 形式 *T* で上書きされる。

sdim INTEGER
sort ='N' の場合、*sdim*= 0
sort ='S' の場合、*sdim* は、*selctg* が真の固有値の数 (ソート後の) に等しくなる。
実数型の場合、どちらかの固有値について *selctg* が真の複素共役ペアは、2 としてカウントされるのに注意が必要である。

alphar, alphai REAL (sgges の場合)
DOUBLE PRECISION (dgges の場合)
配列 DIMENSION はそれぞれ、max(1, *n*) 以上。実数型の汎用固有値を形成する値が格納される。
beta を参照。

alpha COMPLEX (cgges の場合)
DOUBLE COMPLEX (zgges の場合)
配列 DIMENSION は、max(1, *n*) 以上。
複素数型の汎用固有値を形成する値が格納される。
beta を参照。

beta REAL (sgges の場合)
DOUBLE PRECISION (dgges の場合)
COMPLEX (cgges の場合)

DOUBLE COMPLEX (zgges の場合)
配列 DIMENSION は、max(1, *n*) 以上。
実数型の場合 :
終了時、(*alphar*(j) + *alphai*(j)*i)/*beta*(j)、j=1, ..., *n*, は、汎用固有値になる。
alphar(j) + *alphai*(j)*i と *beta*(j)、j=1, ..., *n* は、(*A, B*) の実数汎用 Schur 形式の 2 × 2 の対角ブロックを複素数のユニタリ変換を使用して三角形式に更に縮退する場合に生じる複素数 Schur 形式 (*S, T*) の対角である。*alphai*(j) がゼロの場合は、j 番目の固有値が実数であり、正の場合は、j 番目と (j+1) 番目の固有値が複素共役ペアである (*alphai*(j+1) は負)。
複素数型の場合 :
終了時、*alpha*(j)/*beta*(j)、j=1, ..., *n*, は、汎用固有値になる。*alpha*(j)、j=1, ..., *n*, と *beta*(j)、j=1, ..., *n*, は、cgges/zgges による複素数の Schur 形式 (*S, T*) 出力の対角である。*beta*(j) は、非負の実数になる。
次のアプリケーション・ノートも参照のこと。

vsl, vsr

REAL (sgges の場合)
DOUBLE PRECISION (dgges の場合)
COMPLEX (cgges の場合)
DOUBLE COMPLEX (zgges の場合)
配列 :
*vsl(ldvsl, *)*、*vsl* の第 2 次元は、max(1, *n*) 以上でなければならない。
jobvsl ='V' の場合、この配列には、左 Schur ベクトルが格納される。
jobvsl ='N' の場合、*vsl* は参照されない。
*vsr(ldvsr, *)*、*vsr* の第 2 次元は、max(1, *n*) 以上でなければならない。
jobvsr ='V' の場合、この配列には、右 Schur ベクトルが格納される。
jobvsr ='N' の場合、*vsr* は参照されない。

work(1) 終了時に *info* = 0 の場合、*work(1)* は、*lwork* の必要最小サイズを戻す。

info INTEGER

info = 0 の場合は、正常に終了している。

info = *-i* の場合は、*i* 番目のパラメータの値が不正である。

info = *i* であり、*i* ≤ *n* の場合 :

QZ の繰り返しが失敗した。(*A*, *B*) は Schur 形式でないが、*alphar*(j)、*alphai*(j) (実数型の場合)、または *alpha*(j) (複素数型の場合)、および *beta*(j)、j=*info*+1, ..., *n* は正しい。

i > *n* の場合 : 一般に LAPACK の問題を示すエラー。

i = *n*+1 の場合 : *?hgeqz* で *QZ* の繰り返し以外の操作に失敗。

i = *n*+2 の場合 : 順序変更後、丸め操作によりいくつかの複素数の固有値の値が変更され、その結果、汎用 Schur 形式の先頭の固有値が *selctg* = *.TRUE.* を満たさなくなった。これは、スケーリングによっても引き起こされる。

i = *n*+3 の場合 : *?tgsen* で順序変更に失敗。。

アプリケーション・ノート

配列 *work* に必要なワークスペースの大きさがわからない場合は、最初の実行で *lwork* を大きめの値に設定する。実行の終了時に *work(1)* を調べて、これ以降の実行にはその値を使用するとよい。

一般に、商 *alphar*(j)/*beta*(j) と *alphai*(j)/*beta*(j) は、簡単にオーバーフローまたはアンダーフローし、*beta*(j) がゼロになる場合もある。したがって、単純に比を計算するのは避けなければならない。ただし、*alphar* と *alphai* は、大きさがノルム (*A*) より常に小さくなり、通常はノルム (*A*) と比較できる。また、*beta* は、ノルム (*B*) より常に小さくなり、通常はノルム (*B*) と比較できる。

?ggesx

汎用固有値、Schur 形式を計算し、
オプションで Schur ベクトルの左 / 右の
マトリックスを計算する。

```
call sggesx (jobvsl, jobvsr, sort, selctg, sense, n, a, lda, b, ldb,
             sdim, alphar, alphai, beta, vsl, ldvsl, vsr, ldvsr,
             rconde, rcondv, work, lwork, iwork, liwork, bwork, info )
call dggesx (jobvsl, jobvsr, sort, selctg, sense, n, a, lda, b, ldb,
             sdim, alphar, alphai, beta, vsl, ldvsl, vsr, ldvsr,
             rconde, rcondv, work, lwork, iwork, liwork, bwork, info )
call cggesx (jobvsl, jobvsr, sort, selctg, sense, n, a, lda, b, ldb,
             sdim, alpha, beta, vsl, ldvsl, vsr, ldvsr, rconde,
             rcondv, work, lwork, rwork, iwork, liwork, bwork, info )
call zggesx (jobvsl, jobvsr, sort, selctg, sense, n, a, lda, b, ldb,
             sdim, alpha, beta, vsl, ldvsl, vsr, ldvsr, rconde,
             rcondv, work, lwork, rwork, iwork, liwork, bwork, info )
```

説明

このルーチンは、`n` × `n` の実数 / 複素数非対称マトリックス (*A*, *B*) のペアについて、汎用固有値、汎用実数 / 複素数 Schur 形式 (*S*, *T*) を計算し、オプションで Schur ベクトルの左 / 右のマトリックス (`vsl` と `vsr`) を計算する。これにより、汎用 Schur 因子分解

$$(A, B) = (vsl * S * vsr^H, vsl * T * vsr^H)$$

が提供される。

また、このルーチンは、固有値の選択した束が上準三角マトリックス *S* と上三角マトリックス *T* の主対角ブロックにくるように固有値を順序付けられる。また、指定された固有値の平均に対する条件数の逆数 (`rconde`) を計算し、指定された固有値に対応する右と左の収縮部分空間に対する条件数の逆数 (`rcondv`) を計算することもできる。`vsl` と `vsr` の先頭の列は、対応する左と右の固有空間 (収縮部分空間) に対する直交 / ユニタリ基を形成する。

マトリックス (*A, B*) のペアの汎用固有値は、*A* - *w***B* が特異であるようなスカラー *w* または比 *alpha* / *beta* = *w* である。*beta*=0 または両方がゼロである妥当な解釈が存在するため、通常は、ペア (*alpha*, *beta*) として表現される。
T が非負の対角を持つ上三角で *S* が 1 × 1 と 2 × 2 のブロックを持つブロック上三角である場合、マトリックス (*S, T*) のペアは、汎用実数 Schur 形式である。1 × 1 ブロックは実数の汎用固有値に対応するが、S の 2 × 2 ブロックは、*T* の対応する成分を次の形式にすると " 標準化される "。

$$\begin{pmatrix} a & 0 \\ 0 & b \end{pmatrix}$$

また、*S* と *T* の対応する 2 × 2 ブロックのペアは、汎用固有値の複素共役ペアを持つ。
S と *T* が上三角であり、*T* の対角が非負の実数の場合、マトリックス (*S, T*) のペアは、汎用複素数 Schur 形式になる。

入力パラメータ

jobvsl
: CHARACTER*1。'N' または 'V' でなければならない。
jobvsl ='N' の場合、左 Schur ベクトルは計算されない。
jobvsl ='V' の場合、左 Schur ベクトルは計算される。

jobvsr
: CHARACTER*1。'N' または 'V' でなければならない。
jobvsr ='N' の場合、右 Schur ベクトルは計算されない。
jobvsr ='V' の場合、右 Schur ベクトルは計算される。

sort
: CHARACTER*1。'N' または 'S' でなければならない。
汎用 Schur 形式の対角上の固有値を順序付けるかどうかを指定する。
sort ='N' の場合、固有値は順序付けされない。
sort ='S' の場合、固有値は順序付けされる (*selctg* を参照)。

selctg

3 つの REAL 引数の LOGICAL FUNCTION (実数型の場合)
2 つの COMPLEX 引数の LOGICAL FUNCTION (複素数型の場合)

selctg は、呼び出しサブルーチン内で EXTERNAL として宣言する必要がある。

sort ='S' の場合、*selctg* は、Schur 形式の左上に対してソートを実行する固有値を選択するために使用される。

sort ='N' の場合、*selctg* は参照されない。

実数型の場合 :

selctg(*alphar*(j), *alphai*(j), *beta*(j)) が真の場合は、固有値 (*alphar*(j) + *alphai*(j))/*beta*(j)) が選択される。つまり、固有値の複素共役ペアの一方が選択されると、複素数の固有値が両方選択される。
悪条件の場合、指定された複素数の固有値は、順序付けの実行後、*selctg*(*alphar*(j), *alphai*(j), *beta*(j)) =.TRUE. を満たさないことがある。その場合は、*info* に *n*+2 が設定される。

複素数型の場合 :

selctg(*alpha*(j), *beta*(j)) が真の場合は、固有値 *alpha*(j) / *beta*(j) が選択される。順序付けを行うと、複素数の固有値の値が変更される可能性がある (特に、固有値が悪い条件の場合)。そのため、順序付けを実行した後、指定された複素数の固有値は、*selctg*(*alpha*(j), *beta*(j)) =.TRUE. を満たさないことがある。その場合は、*info* に *n*+2 が設定される (以下の *info* を参照)。

sense

CHARACTER*1。'N'、'E'、'V'、または 'B' でなければならない。
どの条件数の逆数を計算するのかを決定する。

sense ='N' の場合、計算は実行されない。

sense ='E' の場合、指定された固有値の平均についてのみ計算が実行される。

sense ='V' の場合、指定された収縮部分空間についてのみ計算が実行される。

	sense = 'B' の場合、両方について計算が実行される。 *sense* が 'E'、'V'、または 'B' の場合、*sort* は 'S' に等しくなければならない。
n	INTEGER。マトリックス *A*、*B*、*vsl*、*vsr* の次数 ($n \geq 0$)。
a, b, work	REAL (sggesx の場合) DOUBLE PRECISION (dggesx の場合) COMPLEX (cggesx の場合) DOUBLE COMPLEX (zggesx の場合) 配列 : *a(lda, *)* は、$n \times n$ のマトリックス *A* (マトリックスのペアの 1 番目) を格納する配列である。 *a* の第 2 次元は、max(1, *n*) 以上でなければならない。 *b(ldb, *)* は、$n \times n$ のマトリックス *B* (マトリックスのペアの 2 番目) を格納する配列である。 *b* の第 2 次元は、max(1, *n*) 以上でなければならない。 *work(lwork)* は、ワークスペース配列である。
lda	INTEGER。配列 *a* の第 1 次元。max(1, *n*) 以上でなければならない。
ldb	INTEGER。配列 *b* の第 1 次元。max(1, *n*) 以上でなければならない。
ldvsl, ldvsr	INTEGER。それぞれ、出力マトリックス *vsl* と *vsr* の第 1 次元。 次の制約がある。 *ldvsl* ≥ 1。*jobvsl* = 'V' の場合、*ldvsl* ≥ max(1, *n*) *ldvsr* ≥ 1。*jobvsr* = 'V' の場合、*ldvsr* ≥ max(1, *n*)
lwork	INTEGER。配列 *work* の次元。 **実数型の場合 :** *lwork* ≥ max(1, 8(*n*+1)+16) *sense* = 'E'、'V'、または 'B' の場合、 *lwork* ≥ max(8(*n*+1)+16) , 2**sdim**(*n*-*sdim*)) **複素数型の場合 :** *lwork* ≥ max(1, 2*n*) *sense* = 'E'、'V'、または 'B' の場合、 *lwork* ≥ max(2*n* , 2**sdim**(*n*-*sdim*))

性能を上げるには、一般に *lwork* を大きくする必要がある。

rwork REAL (cggesx の場合)
DOUBLE PRECISION (zggesx の場合)
ワークスペース配列 DIMENSION は、max(1, 8*n*) 以上。
この配列は、複素数型でのみ使用される。

iwork INTEGER
ワークスペース配列 DIMENSION(*liwork*)
sense ='N' の場合は参照されない。

liwork INTEGER。配列 *iwork* の次元。
liwork ≥ *n*+6 (実数型の場合)
liwork ≥ *n*+2 (複素数型の場合)

bwork LOGICAL
ワークスペース配列 DIMENSION は、max(1, *n*) 以上。
sort ='N' の場合は参照されない。

出力パラメータ

a 終了時、この配列は、その汎用 Schur 形式 *S* で上書きされる。

b 終了時、この配列は、その汎用 Schur 形式 *T* で上書きされる。

sdim INTEGER
sort ='N' の場合、*sdim*= 0
sort ='S' の場合、*sdim* は、*selctg* が真の固有値の数 (ソート後の) に等しくなる。
実数型の場合、どちらかの固有値について *selctg* が真の複素共役ペアは、2 としてカウントされるのに注意が必要である。

alphar, alphai REAL (sggesx の場合)
DOUBLE PRECISION (dggesx の場合)
配列 DIMENSION はそれぞれ、max(1, *n*) 以上。
実数型の汎用固有値を形成する値が格納される。
beta を参照。

alpha

COMPLEX (cggesx の場合)
DOUBLE COMPLEX (zggesx の場合)
配列 DIMENSION は、max(1, *n*) 以上。複素数型の汎用固有値を形成する値が格納される。
beta を参照。

beta

REAL (sggesx の場合)
DOUBLE PRECISION (dggesx の場合)
COMPLEX (cggesx の場合)
DOUBLE COMPLEX (zggesx の場合)
配列 DIMENSION は、max(1, *n*) 以上。
実数型の場合 :
終了時、(*alphar*(j) + *alphai*(j)*i)/*beta*(j)、j=1, ..., *n*, は、汎用固有値になる。
alphar(j) + *alphai*(j)*i と *beta*(j)、j=1, ..., *n* は、(*A, B*) の実数汎用 Schur 形式の 2 × 2 の対角ブロックを複素数のユニタリ変換を使用して三角形式に更に縮退する場合に生じる複素数 Schur 形式 (*S, T*) の対角である。*alphai*(j) がゼロの場合は、j 番目の固有値が実数であり、正の場合は、j 番目と (j+1) 番目の固有値が複素共役ペアである (*alphai*(j+1) は負)。
複素数型の場合 :
終了時、*alpha*(j)/*beta*(j)、j=1, ..., *n*, は、汎用固有値になる。*alpha*(j)、j=1, ..., *n*, と *beta*(j)、j=1, ..., *n*, は、cggesx/zggesx による複素数の Schur 形式 (*S, T*) 出力の対角である。*beta*(j) は、非負の実数になる。
次のアプリケーション・ノートも参照のこと。

vsl, vsr

REAL (sggesx の場合)
DOUBLE PRECISION (dggesx の場合)
COMPLEX (cggesx の場合)
DOUBLE COMPLEX (zggesx の場合)
配列 :
*vsl(ldvsl, *)*、*vsl* の第 2 次元は、max(1, *n*) 以上でなければならない。

jobvsl = 'V' の場合、この配列には、左 Schur ベクトルが格納される。
jobvsl = 'N' の場合、*vsl* は参照されない。
*vsr(ldvsr, *)*、*vsr* の第 2 次元は、max(1, *n*) 以上でなければならない。
jobvsr = 'V' の場合、この配列には、右 Schur ベクトルが格納される。
jobvsr = 'N' の場合、*vsr* は参照されない。

rconde, rcondv REAL (単精度型の場合)
DOUBLE PRECISION (倍精度型の場合)
配列は、それぞれ DIMENSION(2)
sense = 'E' または 'B' の場合、*rconde*(1) と *rconde*(2) には、指定された固有値の平均に対する条件数の逆数が格納される。
sense = 'N' または 'V' の場合は参照されない。
sense = 'V' または 'B' の場合、*rcondv*(1) と *rcondv*(2) には、指定された収縮部分空間に対する条件数の逆数が格納される。
sense = 'N' または 'E' の場合は参照されない。

work(1) 終了時に *info* = 0 の場合、*work(1)* は、*lwork* の必要最小サイズを戻す。

info INTEGER
info = 0 の場合は、正常に終了している。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。
info = *i* であり、
i ≤ *n* の場合 :
QZ の繰り返しが失敗した。(*A*, *B*) は Schur 形式でないが、*alphar*(j)、*alphai*(j) (実数型の場合)、または *alpha*(j) (複素数型の場合)、および *beta*(j)、j=*info*+1, ..., *n* は正しい。
i > *n* の場合 : 一般に LAPACK の問題を示すエラー。
i = *n*+1 の場合 : ?hgeqz で *QZ* の繰り返し以外の操作に失敗した。

i = *n*+2 の場合 : 順序変更後、丸め操作によりいくつかの複素数の固有値の値が変更され、その結果、汎用 Schur 形式の先頭の固有値が *selctg* = .TRUE. を満たさなくなった。これは、スケーリングによっても引き起こされる。

i = *n*+3 の場合 : ?tgsen で順序変更に失敗した。

アプリケーション・ノート

配列 *work* に必要なワークスペースの大きさがわからない場合は、最初の実行で *lwork* を大きめの値に設定する。実行の終了時に *work*(1) を調べて、これ以降の実行にはその値を使用するとよい。

一般に、商 *alphar*(j)/*beta*(j) と *alphai*(j)/*beta*(j) は、簡単にオーバーフローまたはアンダーフローし、*beta*(j) がゼロになる場合もある。したがって、単純に比を計算するのは避けなければならない。ただし、*alphar* と *alphai* は、大きさがノルム (*A*) より常に小さくなり、通常はノルム (*A*) と比較できる。また、*beta* は、ノルム (*B*) より常に小さくなり、通常はノルム (*B*) と比較できる。

?ggev

非対称マトリックスのペアについて、
汎用固有値、左 / 右の汎用固有ベクトルを
計算する。

```
call sggev ( jobvl, jobvr, n, a, lda, b, ldb, alphar, alphai, beta,
             vl, ldvl, vr, ldvr, work, lwork, info )
call dggev ( jobvl, jobvr, n, a, lda, b, ldb, alphar, alphai, beta,
             vl, ldvl, vr, ldvr, work, lwork, info )
call cggev ( jobvl, jobvr, n, a, lda, b, ldb, alpha, beta, vl, ldvl,
             vr, ldvr, work, lwork, rwork, info )
call zggev ( jobvl, jobvr, n, a, lda, b, ldb, alpha, beta, vl, ldvl,
             vr, ldvr, work, lwork, rwork, info )
```

説明

このルーチンは、`n` × `n` の実数 / 複素数非対称マトリックス (*A*, B) のペアについて、汎用固有値を計算し、オプションで左 / 右の汎用固有ベクトルを計算する。

マトリックス (*A*, *B*) のペアの汎用固有値は、*A* - λ***B* が特異であるようなスカラー λ または比 *alpha* / *beta* = λ である。*beta*=0 および両方がゼロである妥当な解釈が存在するため、通常は、ペア (*alpha*, *beta*) として表現される。
(*A*, *B*) の汎用固有値 λ(j) に対応する右汎用固有ベクトル *v*(j) は、以下を満足する。

$$A*v(j) = \lambda(j)*B*v(j)$$

(*A*, *B*) の汎用固有値 λ(j) に対応する左汎用固有ベクトル *u*(j) は、以下を満足する。

$$u(j)^H*A = \lambda(j)*u(j)^H*B$$

$u(j)^H$ は、*u*(j) の共役転置を示す。

入力パラメータ

jobvl
CHARACTER*1。'N' または 'V' でなければならない。
jobvl ='N' の場合、左汎用固有ベクトルは計算されない。
jobvl ='V' の場合、左汎用固有ベクトルは計算される。

jobvr
CHARACTER*1。'N' または 'V' でなければならない。
jobvr ='N' の場合、右汎用固有ベクトルは計算されない。
jobvr ='V' の場合、右汎用固有ベクトルは計算される。

n
INTEGER。マトリックス A、B、*vl*、*vr* の次数 ($n \geq 0$)

a, b, work
REAL (sggev の場合)
DOUBLE PRECISION (dggev の場合)
COMPLEX (cggev の場合)
DOUBLE COMPLEX (zggev の場合)
配列 :
*a(lda, *)* は、$n \times n$ のマトリックス A (マトリックスのペアの 1 番目) を格納する配列である。
a の第 2 次元は、max(1, *n*) 以上でなければならない。
*b(ldb, *)* は、$n \times n$ のマトリックス B (マトリックスのペアの 2 番目) を格納する配列である。
b の第 2 次元は、max(1, *n*) 以上でなければならない。
work(lwork) は、ワークスペース配列である。

lda
INTEGER。配列 *a* の第 1 次元。max(1, *n*) 以上でなければならない。

ldb
INTEGER。配列 *b* の第 1 次元。max(1, *n*) 以上でなければならない。

ldvl, ldvr
INTEGER。それぞれ、出力マトリックス *vl* と *vr* の第 1 次元。次の制約がある。
ldvl ≥ 1。*jobvl* ='V' の場合、*ldvl* ≥ max(1, *n*)
ldvr ≥ 1。*jobvr* ='V' の場合、*ldvr* ≥ max(1, *n*)

lwork
INTEGER。配列 *work* の次元。

lwork ≥ max(1, 8*n*+16) (実数型の場合)
lwork ≥ max(1, 2*n*) (複素数型の場合)
性能を上げるには、一般に *lwork* を大きくする必要がある。

rwork REAL (cggev の場合)
DOUBLE PRECISION (zggev の場合)
ワークスペース配列 DIMENSION は、max(1, 8*n*) 以上。この配列は、複素数型でのみ使用される。

出力パラメータ

a, b 終了時、これらの配列は上書きされる。

alphar, alphai REAL (sggev の場合)
DOUBLE PRECISION (dggev の場合)
配列 DIMENSION はそれぞれ、max(1, *n*) 以上。実数型の汎用固有値を形成する値が格納される。
beta を参照。

alpha COMPLEX (cggev の場合)
DOUBLE COMPLEX (zggev の場合)
配列 DIMENSION は、max(1, *n*) 以上。複素数型の汎用固有値を形成する値が格納される。
beta を参照。

beta REAL (sggev の場合)
DOUBLE PRECISION (dggev の場合)
COMPLEX (cggev の場合)
DOUBLE COMPLEX (zggev の場合)
配列 DIMENSION は、max(1, *n*) 以上。
実数型の場合 :
終了時、(*alphar*(j) + *alphai*(j)*i)/*beta*(j)、j=1, ..., *n*, は、汎用固有値になる。
alphai(j) がゼロの場合は、j 番目の固有値が実数であり、正の場合は、j 番目と (j+1) 番目の固有値が複素共役ペアである (*alphai*(j+1) は負)。
複素数型の場合 :
終了時、*alpha*(j)/*beta*(j)、j=1, ..., *n*, は、汎用固有値になる。
次の「アプリケーション・ノート」も参照のこと。

vl, vr

REAL (sggev の場合)
DOUBLE PRECISION (dggev の場合)
COMPLEX (cggev の場合)
DOUBLE COMPLEX (zggev の場合)
配列 :
vl(*ldvl*, *)); *vl* の第 2 次元は、max(1, *n*) 以上でなければならない。

jobvl ='V' の場合、左汎用固有ベクトル *u*(j) は、それらの固有値と同じ順序で、*vl* の列に次々と格納される。各固有ベクトルは、最大のコンポーネントが abs(Re) + abs(Im) = 1 となるようにスケーリングされる。*jobvl* ='N' の場合、*vl* は参照されない。

実数型の場合 :
j 番目の固有値が実数の場合、*u*(j) = *vl*(:, j) (*vl* の j 番目の列) である。j 番目と (j+1) 番目の固有値が複素共役ペアとなる場合は、*u*(j) = *vl*(:, j) + *i***vl*(:, j+1)、*u*(j+1) = *vl*(:, j) - *i***vl*(:, j+1) である。(*i*=$\sqrt{-1}$ である)

複素数型の場合 :
u(j) = *vl*(:, j) (*vl* の j 番目の列)

vr(*ldvr*, *);*vr* の第 2 次元は、max(1, *n*) 以上でなければならない。

jobvr ='V' の場合、右汎用固有ベクトル *v*(j) は、それらの固有値と同じ順序で、*vr* の列に次々と格納される。各固有ベクトルは、最大のコンポーネントが abs(Re) + abs(Im) = 1 となるようにスケーリングされる。*jobvr* ='N' の場合、*vr* は参照されない。

実数型の場合 :
j 番目の固有値が実数の場合、*v*(j) = *vr*(:, j) (*vr* の j 番目の列) である。j 番目と (j+1) 番目の固有値が複素共役ペアとなる場合は、*v*(j) = *vr*(:, j) + *i***vr*(:, j+1)、*v*(j+1) = *vr*(:, j) - *i***vr*(:, j+1) である。

複素数型の場合 :
v(j) = *vr*(:, j) (*vr* の j 番目の列)

work(1)

終了時に *info* = 0 の場合、*work(1)* は、*lwork* の必要最小サイズを戻す。

info INTEGER

info = 0 の場合は、正常に終了している。

info = -*i* の場合は、*i* 番目のパラメータの値が不正である。

info = *i* であり、

i ≤ *n* の場合 :

QZ の繰り返しが失敗した。固有ベクトルは計算されなかったが、*alphar*(j)、*alphai*(j) (実数型の場合)、または *alpha*(j) (複素数型の場合)、および *beta*(j)、j=*info*+1, ..., *n* は正しい。

i > *n* の場合 : 一般に LAPACK の問題を示すエラー。

i = *n*+1 の場合: ?hgeqz で *QZ* の繰り返し以外の操作に失敗した。

i = *n*+2 の場合 : ?tgevc からエラーが戻された。

アプリケーション・ノート

配列 *work* に必要なワークスペースの大きさがわからない場合は、最初の実行で *lwork* を大きめの値に設定する。実行の終了時に *work*(1) を調べて、これ以降の実行にはその値を使用するとよい。

一般に、商 *alphar*(j)/*beta*(j) と *alphai*(j)/*beta*(j) は、簡単にオーバーフローまたはアンダーフローし、*beta*(j) がゼロになる場合もある。したがって、単純に比を計算するのは避けなければならない。ただし、*alphar* と *alphai* (実数型の場合)、または *alpha* (複素数型の場合) は、大きさがノルム (*A*) より常に小さくなり、通常はノルム (*A*) と比較できる。また、*beta* は、ノルム (*B*) より常に小さくなり、通常はノルム (*B*) と比較できる。

?ggevx

汎用固有値を計算し、オプションで
左 / 右の汎用固有ベクトルを計算する。

```
call sggevx ( balanc, jobvl, jobvr, sense, n, a, lda, b, ldb, alphar,
              alphai, beta, vl, ldvl, vr, ldvr, ilo, ihi, lscale,
              rscale, abnrm, bbnrm, rconde, rcondv, work, lwork,
              iwork, bwork, info)
call dggevx ( balanc, jobvl, jobvr, sense, n, a, lda, b, ldb, alphar,
              alphai, beta, vl, ldvl, vr, ldvr, ilo, ihi, lscale,
              rscale, abnrm, bbnrm, rconde, rcondv, work, lwork,
              iwork, bwork, info)
call cggevx ( balanc, jobvl, jobvr, sense, n, a, lda, b, ldb, alpha,
              beta, vl, ldvl, vr, ldvr, ilo, ihi, lscale, rscale,
              abnrm, bbnrm, rconde, rcondv, work, lwork, rwork,
              iwork, bwork, info)
call zggevx ( balanc, jobvl, jobvr, sense, n, a, lda, b, ldb, alpha,
              beta, vl, ldvl, vr, ldvr, ilo, ihi, lscale, rscale,
              abnrm, bbnrm, rconde, rcondv, work, lwork, rwork,
              iwork, bwork, info)
```

説明

このルーチンは、$n \times n$ の実数 / 複素数非対称マトリックス (*A*, *B*) のペアについて、汎用固有値を計算し、オプションで左 / 右の汎用固有ベクトルを計算する。

また、このルーチンは、平衡化のための変換を計算し、固有値と固有ベクトル (*ilo*、*ihi*、*lscale*、*rscale*、*abnrm*、*bbnrm*)、固有値に対する条件数の逆数 (*rconde*)、右固有ベクトルに対する条件数の逆数 (*rcondv*) の条件付けを改善できる。

マトリックス (*A*, *B*) のペアの汎用固有値は、$A - \lambda * B$ が特異であるようなスカラー λ または比 *alpha* / *beta* = λ である。*beta*=0 および両方がゼロである妥当な解釈が存在するため、通常は、ペア (*alpha*, *beta*) として表現される。

(*A*, *B*) の汎用固有値 λ(j) に対応する右汎用固有ベクトル v(j) は、以下を満足する。

$$A*v(j) = \lambda(j)*B*v(j)$$

(*A*, *B*) の汎用固有値 λ(j) に対応する左汎用固有ベクトル *u*(j) は、以下を満足する。

$$u(j)^H*A = \lambda(j)*u(j)^H*B$$

$u(j)^H$ は、*u*(j) の共役転置を示す。

入力パラメータ

balanc
: CHARACTER*1。'N'、'P'、'S'、または 'B' でなければならない。
実行する平衡化オプションを指定する。
balanc ='N' の場合、対角的なスケーリングまたは置換は実行されない。
balanc ='P' の場合、置換のみが実行される。
balanc ='S' の場合、スケーリングのみが実行される。
balanc ='B' の場合、置換とスケーリングがともに実行される。
計算された条件数の逆数は、平衡化または置換を実行後のマトリックスに使用される。置換を実行しても条件数は変更されないが (正確な演算では)、平衡化を実行すると条件数が変更される。

jobvl
: CHARACTER*1。'N' または 'V' でなければならない。
jobvl ='N' の場合、左汎用固有ベクトルは計算されない。
jobvl ='V' の場合、左汎用固有ベクトルは計算される。

jobvr
: CHARACTER*1。'N' または 'V' でなければならない。
jobvr ='N' の場合、右汎用固有ベクトルは計算されない。
jobvr ='V' の場合、右汎用固有ベクトルは計算される。

sense
: CHARACTER*1。'N'、'E'、'V'、または 'B' でなければならない。
どの条件数の逆数を計算するのかを決定する。

sense = 'N' の場合、計算は実行されない。
sense = 'E' の場合、固有値のみ計算が実行される。
sense = 'V' の場合、固有ベクトルのみ計算が実行される。
sense = 'B' の場合、固有値と固有ベクトルの計算が実行される。

n INTEGER。マトリックス *A*、*B*、*vl*、*vr* の次数 ($n \geq 0$)。

a, b, work REAL (sggevx の場合)
DOUBLE PRECISION (dggevx の場合)
COMPLEX (cggevx の場合)
DOUBLE COMPLEX (zggevx の場合)
配列 :
*a(lda, *)* は、$n \times n$ のマトリックス *A* (マトリックスのペアの 1 番目) を格納する配列である。
a の第 2 次元は、max(1, *n*) 以上でなければならない。
*b(ldb, *)* は、$n \times n$ のマトリックス *B* (マトリックスのペアの 2 番目) を格納する配列である。
b の第 2 次元は、max(1, *n*) 以上でなければならない。
work(lwork) は、ワークスペース配列である。

lda INTEGER。配列 *a* の第 1 次元。max(1, *n*) 以上でなければならない。

ldb INTEGER。配列 *b* の第 1 次元。max(1, *n*) 以上でなければならない。

ldvl, ldvr INTEGER。それぞれ、出力マトリックス *vl* と *vr* の第 1 次元。
次の制約がある。
ldvl ≥ 1。*jobvl* = 'V' の場合、*ldvl* ≥ max(1, *n*)
ldvr ≥ 1。*jobvr* = 'V' の場合、*ldvr* ≥ max(1, *n*)

lwork INTEGER。配列 *work* の次元。
実数型の場合 :
lwork ≥ max(1, 6*n*)
sense = 'E' の場合、*lwork* ≥ 12*n*
sense = 'V' または 'B' の場合、*lwork* ≥ $2n^2 + 12n + 16$

複素数型の場合 :
lwork ≥ max(1, 2*n*)
sense = 'N' または 'E' の場合、*lwork* ≥ 2*n*
sense = 'V' または 'B' の場合、*lwork* ≥ $2n^2 + 2n$

rwork REAL (cggevx の場合)
DOUBLE PRECISION (zggevx の場合)
ワークスペース配列 DIMENSION は、max(1, 6*n*) 以上。
この配列は、複素数型でのみ使用される。

iwork INTEGER
ワークスペース配列 DIMENSION は、実数型の場合は (*n*+6) 以上、複素数型の場合は (*n*+2) 以上。
sense = 'E' の場合は参照されない。

bwork LOGICAL
ワークスペース配列 DIMENSION は、max(1, *n*) 以上。
sense = 'N' の場合は参照されない。

出力パラメータ

a, b 終了時、これらの配列は上書きされる。
jobvl = 'V' または *jobvr* = 'V'(またはその両方) の場合、*a* には、平衡化した入力 *A* と *B* の実数 Schur 形式の 1 番目の部分が格納され、*b* には、その 2 番目の部分が格納される。

alphar, alphai REAL (sggevx の場合)
DOUBLE PRECISION (dggevx の場合)
配列 DIMENSION はそれぞれ、max(1, *n*) 以上。
実数型の汎用固有値を形成する値が格納される。
beta を参照。

alpha COMPLEX (cggevx の場合)
DOUBLE COMPLEX (zggevx の場合)
配列 DIMENSION は、max(1, *n*) 以上。
複素数型の汎用固有値を形成する値が格納される。
beta を参照。

beta REA L(sggevx の場合)
DOUBLE PRECISION (dggevx の場合)
COMPLEX (cggevx の場合)
DOUBLE COMPLEX (zggevx の場合)

配列 DIMENSION は、max(1, *n*) 以上。

実数型の場合 :

終了時、(*alphar*(j) + *alphai*(j)*i)/*beta*(j)、j=1, ..., *n*, は、汎用固有値になる。

alphai(j) がゼロの場合は、j 番目の固有値が実数であり、正の場合は、j 番目と (j+1) 番目の固有値が複素共役ペアである (*alphai*(j+1) は負)。

複素数型の場合 :

終了時、*alpha*(j)/*beta*(j)、j=1, ..., *n*, は、汎用固有値になる。

次の「アプリケーション・ノート」も参照のこと。

vl, vr

REAL (sggevx の場合)

DOUBLE PRECISION (dggevx の場合)

COMPLEX (cggevx の場合)

DOUBLE COMPLEX (zggevx の場合)

配列 :

vl(*ldvl*, *)); *vl* の第 2 次元は、max(1, *n*) 以上でなければならない。

jobvl ='V' の場合、左汎用固有ベクトル *u*(j) は、それらの固有値と同じ順序で、*vl* の列に次々と格納される。各固有ベクトルは、最大のコンポーネントが abs(Re) + abs(Im) = 1 となるようにスケーリングされる。*jobvl* ='N' の場合、*vl* は参照されない。

実数型の場合 :

j 番目の固有値が実数の場合、*u*(j) = *vl*(:, j) (*vl* の j 番目の列) である。j 番目と (j+1) 番目の固有値が複素共役ペアとなる場合は、*u*(j) = *vl*(:, j) + *i***vl*(:, j+1)、*u*(j+1) = *vl*(:, j) - *i***vl*(:, j+1) である。($i=\sqrt{-1}$ である)

複素数型の場合 :

u(j) = *vl*(:, j) (*vl* の j 番目の列)

vr(*ldvr*, *);*vr* の第 2 次元は、max(1, *n*) 以上でなければならない。

jobvr ='V' の場合、右汎用固有ベクトル *v*(j) は、それらの固有値と同じ順序で、*vr* の列に次々と格納される。各固有ベクトルは、最大のコンポーネントが abs(Re) + abs(Im) = 1 となるようにスケーリングされる。*jobvr* ='N' の場合、*vr* は参照されない。

実数型の場合 :
j 番目の固有値が実数の場合、v(j) = `vr`(:, j) (`vr` の j 番目の列) である。j 番目と (j+1) 番目の固有値が複素共役ペアとなる場合は、v(j) = `vr`(:, j) + i* `vr`(:, j+1)、v(j+1) = `vr`(:, j) - i* `vr`(:, j+1) である。

複素数型の場合 :
v(j) = `vr`(:, j) (`vr` の j 番目の列)

`ilo, ihi` `INTEGER`

`ilo` と `ihi` は、終了時に A(i, j) = 0 および B(i, j) = 0 (i > j かつ j = 1, ..., `ilo`-1 または i = `ihi`+1, ..., `n` の場合) であるような整数値である。
`balanc` = `'N'` または `'S'` の場合、`ilo` = 1 と `ihi` = `n` である。

`lscale, rscale` `REAL` (単精度型の場合)
`DOUBLE PRECISION` (倍精度型の場合)
配列 `DIMENSION` はそれぞれ、max(1, `n`) 以上。
`lscale` には、A と B の左側に適用される置換およびスケーリング係数の各成分が格納される。PL(j) は行 j と交換される行のインデックスであり、DL(j) は行 j に適用されるスケーリング係数である。したがって、

`lscale`(j) = PL(j)、j = 1, ..., `ilo`-1
= DL(j)、j = `ilo`, ..., `ihi`
= PL(j)、j = `ihi`+1, ..., `n`

交換を実行する順序は、`n` ～ `ihi`+1、次に 1 ～ `ilo`-1 である。

`rscale` には、A と B の右側に適用される置換およびスケーリング係数の各成分が格納される。PR(j) は列 j と交換される列のインデックスであり、DR(j) は列 j に適用されるスケーリング係数である。したがって、

`rscale`(j) = PR(j)、j = 1, ..., `ilo`-1
= DR(j)、j = `ilo`, ..., `ihi`
= PR(j)、j = `ihi`+1, ..., `n`

交換を実行する順序は、*n* ～ *ihi*+1、次に 1 ～ *ilo*-1 である。

abnrm, bbnrm
REAL (単精度型の場合)
DOUBLE PRECISION (倍精度型の場合)
それぞれ、平衡化したマトリックス *A* と *B* の 1- ノルムである。

rconde, rcondv
REAL (単精度型の場合)
DOUBLE PRECISION (倍精度型の場合)
配列 DIMENSION はそれぞれ、max(1, *n*) 以上。
sense ='E' または 'B' の場合、*rconde* には、指定された固有値の条件数の逆数を格納する。これらは、配列の連続する成分に格納される。固有値の複素共役ペアの場合、*rconde* の 2 つの連続する成分には同じ値が設定される。したがって、*rconde*(j)、*rcondv*(j)、*vl* と *vr* の j 番目の列はすべて、同じ固有ペアに対応している (ただし、選択しなかった固有ペアが存在する場合は、一般に、j 番目の固有ペアにはならない)。
sense ='V' の場合、*rconde* は参照されない。
sense ='V' または 'B' の場合、*rcondv* には、指定された固有ベクトルについての見積もりの条件数の逆数が格納される。これらは、配列の連続する成分に格納される。複素数の固有ベクトルの場合、*rcondv* の 2 つの連続する成分には同じ値が設定される。*rcondv*(j) を計算するために固有値の順序を変更できない場合は、*rcondv*(j) にゼロが設定される。これは、真の値が非常に小さい場合にだけ発生する可能性がある。
sense ='E' の場合、*rcondv* は参照されない。

work(1)
終了時に *info* = 0 の場合、*work(1)* は、*lwork* の必要最小サイズを戻す。

info
INTEGER
info = 0 の場合は、正常に終了している。
info = -*i* の場合は、*i* 番目のパラメータの値が不正である。
info = *i* であり、
i ≤ *n* の場合 :

QZ の繰り返しが失敗した。固有ベクトルは計算されなかったが、*alphar*(j)、*alphai*(j) (実数型の場合)、または *alpha*(j) (複素数型の場合)、および *beta*(j)、j=*info*+1, ..., *n* は正しい。

i > *n* の場合 : 一般に LAPACK の問題を示すエラー。

i = *n*+1 の場合 : ?hgeqz で *QZ* の繰り返し以外の操作に失敗した。

i = *n*+2 の場合 : ?tgevc からエラーが戻された。

アプリケーション・ノート

配列 *work* に必要なワークスペースの大きさがわからない場合は、最初の実行で *lwork* を大きめの値に設定する。実行の終了時に *work*(1) を調べて、これ以降の実行にはその値を使用するとよい。

一般に、商 *alphar*(j)/*beta*(j) と *alphai*(j)/*beta*(j) は、簡単にオーバーフローまたはアンダーフローし、*beta*(j) がゼロになる場合もある。したがって、単純に比を計算するのは避けなければならない。ただし、*alphar* と *alphai* (実数型の場合) または *alpha* (複素数型の場合) は、大きさがノルム (*A*) より常に小さくなり、通常はノルム (*A*) と比較できる。また、*beta* は、ノルム (*B*) より常に小さくなり、通常はノルム (*B*) と比較できる。

参考文献

[LUG] E. Anderson, Z. Bai et al. *LAPACK User's Guide.* Third edition, SIAM, Philadelphia, 1999.

[Golub96] G.Golub, C. Van Loan. *Matrix Computations.* Johns Hopkins University Press, Baltimore, third edition, 1996.

LAPACK 補助ルーチン

6

この章では LAPACK 補助ルーチンに関するインテル ® マス・カーネル・ライブラリの実装について説明する。ライブラリは実数と複素の両方のデータに対応した補助ルーチンで構成される。

LAPACK 補助ルーチンで使用されているルーチン名の規則、数学表記、マトリックスの格納形式は、これまでの章に記載されているドライバおよび計算ルーチンのものと同一である。

ルーチンはルーチン名または関数グループ名 (常に ? から始まる) のアルファベット順に並べられている。

?lacgv

複素ベクトルを共役する。

```
call clacgv (n, x, incx)
call zlacgv (n, x, incx)
```

説明

このルーチンは、長さ *n*、増分 *incx* を持つ複素ベクトル *x* を共役する。(付録 A の「BLAS のベクトル引数」を参照)。

入力パラメータ

n	INTEGER。ベクトル *x* の長さ ($n \geq 0$)。
x	COMPLEX (clacgv の場合) COMPLEX*16 (zlacgv の場合) 配列、次元は (1+(*n*-1)* |*incx*|) 共役させる長さ *n* のベクトルを格納する。

incx INTEGER。*x* の連続する成分の間隔。

出力パラメータ

x conjg(*x*) で上書きされる。

?lacrm

複素マトリックスに正方実数
マトリックスを乗算する。

```
call clacrm (m, n, a, lda, b, ldb, c, ldc, rwork)
call zlacrm (m, n, a, lda, b, ldb, c, ldc, rwork)
```

説明

このルーチンは、次の形式の単純なマトリックス - マトリックス操作を実行する。

$$C = A * B,$$

ここで A は $m \times n$ で複素、B は $n \times n$ で実数、C は $m \times n$ で複素である。

入力パラメータ

m INTEGER。マトリックス A の行数とマトリックス C の行数 ($m \geq 0$)。

n INTEGER。マトリックス B の列数と行数およびマトリックス C の列数 ($n \geq 0$)。

a COMPLEX (clacrm の場合)
COMPLEX*16 (zlacrm の場合)
配列 DIMENSION は *(lda, n)*。$m \times n$ マトリックス A が格納される。

lda INTEGER。配列 *a* のリーディング・ディメンジョン。*lda* $\geq \max(1, m)$

b REAL (clacrm の場合)
DOUBLE PRECISION (zlacrm の場合)
配列 DIMENSION は (*ldb*, *n*)。*n* × *n* マトリックス *B* が格納される。

ldb INTEGER。配列 *b* のリーディング・ディメンジョン。*ldb* ≥ max(1, *n*)

ldc INTEGER。出力配列 *c* のリーディング・ディメンジョン。*ldc* ≥ max(1, *n*)

rwork REAL (clacrm の場合)
DOUBLE PRECISION (zlacrm の場合)
ワークスペース配列 DIMENSION は (2*m*n)

出力パラメータ

c COMPLEX (clacrm の場合)
COMPLEX*16 (zlacrm の場合)
配列 DIMENSION は (*ldc*, *n*)。*m* × *n* マトリックス *C* が格納される。

?lacrt

複素ベクトル対の線形変換を
実行する。

```
call clacrt (n, cx, incx, cy, incy, c, s)
call zlacrt (n, cx, incx, cy, incy, c, s)
```

説明

このルーチンは次の変換を実行する。

$$\begin{pmatrix} c & s \\ -s & c \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} \Rightarrow \begin{pmatrix} x \\ y \end{pmatrix}$$

C と *S* は複素スカラー、*x* と *y* は複素ベクトルである。

入力パラメータ

n	INTEGER。ベクトル *cx* とベクトル *cy* の成分の数 ($n \geq 0$)。
cx, cy	COMPLEX (clacrt の場合) COMPLEX*16 (zlacrt の場合) 配列、次元は (*n*) それぞれ入力ベクトル *x* と *y* を格納する。
incx	INTEGER。*cx* の連続する成分間の増分。
incy	INTEGER。*cy* の連続する成分間の増分。
c, s	COMPLEX (clacrt の場合) COMPLEX*16 (zlacrt の場合) 変換マトリックスを定義する複素スカラー

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix}$$

出力パラメータ

cx	*c***x* + *s***y* で上書きされる。
cy	-*s***x* + *c***y* で上書きされる。

?laesy

2 × 2 の複素対称マトリックスの固有値と固有ベクトルを計算し、固有ベクトルのマトリックスのノルムがしきい値より大きいことを確認する。

```
call claesy (a, b, c, rt1, rt2, evscal, cs1, sn1)
call zlaesy (a, b, c, rt1, rt2, evscal, cs1, sn1)
```

説明

このルーチンは、固有ベクトルのマトリックスのノルムがしきい値より大きければ、次の 2 × 2 対称マトリックスの固有分解を実行する。

$$\begin{bmatrix} a & b \\ b & c \end{bmatrix}$$

rt1 は絶対値が大きい方の固有値、*rt2* は絶対値が小さい方の固有値である。固有ベクトルが計算されると、(*cs1*, *sn1*) は *rt1* の単位固有ベクトルで上書きされ、ゆえに、

$$\begin{bmatrix} cs1 & sn1 \\ -sn1 & cs1 \end{bmatrix} \cdot \begin{bmatrix} a & b \\ b & c \end{bmatrix} \cdot \begin{bmatrix} cs1 & -sn1 \\ sn1 & cs1 \end{bmatrix} = \begin{bmatrix} rt1 & 0 \\ 0 & rt2 \end{bmatrix}$$

入力パラメータ

a, b, c
: COMPLEX (claesy の場合)
COMPLEX*16 (zlaesy の場合)
入力マトリックスの成分。

出力パラメータ

rt1, rt2
: COMPLEX (claesy の場合)
COMPLEX*16 (zlaesy の場合)
それぞれ、大きい方 (モジュラス) の固有値、小さい方 (モジュラス) の固有値。

evscal
: COMPLEX (claesy の場合)
COMPLEX*16 (zlaesy の場合)
固有ベクトル・マトリックスを正規直交化するためにスケーリングした複素値。*evscal* がゼロの場合、固有ベクトルは計算されていない。これには 2 つの理由が考えられる。2 × 2 マトリックスを対角化できなかったか、スケーリング前の固有ベクトルのマトリックスのノルムがしきい値 thresh (0.1E0 に設定) よりも大きかったからである。

cs1, sn1	COMPLEX (claesy の場合) COMPLEX*16 (zlaesy の場合) *evscal* がゼロでない場合、(*cs1*, *sn1*) は *rt1* の単位右固有ベクトルである。

?rot

複素ベクトル対に実余弦と複素正弦を用いて面回転を適用する。

```
call crot (n, cx, incx, cy, incy, c, s)
call zrot (n, cx, incx, cy, incy, c, s)
```

説明

このルーチンは面回転を適用する。ここで余弦 (*c*) は実数、正弦 (*s*) は複素数、ベクトル *cx* と *cy* は複素数である。このルーチンと実数において等価な関数が BLAS にある (第 2 章の「?rot」を参照)。

入力パラメータ

n	INTEGER。ベクトル *cx* とベクトル *cy* の成分の数。
cx, cy	COMPLEX (crot の場合) COMPLEX*16 (zrot の場合) 次元 (*n*) の配列で、それぞれ入力ベクトル *x* と *y* を格納する。
incx	INTEGER。*cx* の連続する成分間の増分。
incy	INTEGER。*cy* の連続する成分間の増分。
c	REAL (crot の場合) DOUBLE PRECISION (zrot の場合)

s COMPLEX (crot の場合)
COMPLEX*16 (zrot の場合)
回転を定義する値で、

$$\begin{bmatrix} c & s \\ -\mathrm{conjg}(s) & c \end{bmatrix}$$

ここで、$c*c + s*\mathrm{conjg}(s) = 1.0$

出力パラメータ

cx $c*x + s*y$ で上書きされる。

cy $-\mathrm{conjg}(s)*x + c*y$ で上書きされる。

?spmv

複素対称パックド・マトリックスを
使用して複素ベクトルに対する
マトリックス・ベクトル積を計算する。

```
call cspmv ( uplo, n, alpha, ap, x, incx, beta, y, incy )
call zspmv ( uplo, n, alpha, ap, x, incx, beta, y, incy )
```

説明

これらのルーチンは、次のように定義されるマトリックス・ベクトル演算を実行する。

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

alpha と *beta* は複素スカラー、

x と *y* は、*n* 個の成分を持つ複素ベクトル、

a は、パックド形式で与えられる $n \times n$ の複素対称マトリックスである。

このルーチンと実数において等価な関数が BLAS にある (第 2 章の「?spmv」を参照)。

入力パラメータ

uplo
CHARACTER*1。次に示すように、マトリックス *A* の上三角部分と下三角部分のどちらをパックド形式の配列 *ap* に与えるかを指定する。

uplo = 'U' または 'u' の場合、マトリックス *A* の上三角部分を配列 *ap* に格納する。

uplo = 'L' または 'l' の場合、マトリックス *A* の下三角部分を配列 *ap* に格納する。

n
INTEGER。マトリックス *A* の次数を指定する。*n* の値はゼロ以上でなければならない。

alpha, beta
COMPLEX (cspmv の場合)
COMPLEX*16 (zspmv の場合)
複素スカラー *alpha* と *beta* を指定する。*beta* をゼロとして与えた場合、入力 *y* を設定する必要はない。

ap
COMPLEX (cspmv の場合)
COMPLEX*16 (zspmv の場合)
配列 DIMENSION は少なくとも ((*n**(*n* + 1))/2)。
uplo = 'U' または 'u' と指定した場合は、*ap*(1) に *a*(1, 1) が、*ap*(2) に *a*(1, 2) が、*ap*(3) に *a*(2, 2) がそれぞれ格納されるように、配列 *ap* には順番にパックされた対称マトリックスの上三角部分を列ごとに格納しなければならない。また、*uplo* = 'L' または 'l' と指定した場合は、*ap*(1) に *a*(1, 1) が、*ap*(2) に *a*(2, 1) が、*ap*(3) に *a*(3, 1) がそれぞれ格納されるように、配列 *ap* には順番にパックされた対称マトリックスの下三角部分を列ごとに格納しなければならない。

x
COMPLEX (cspmv の場合)
COMPLEX*16 (zspmv の場合)
配列 DIMENSION は少なくとも (1 + (*n* - 1)*abs(*incx*))。増分された配列 *x* に *n* 個の成分を持つベクトル *x* を格納しなければならない。

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

y COMPLEX (cspmv の場合)
COMPLEX*16 (zspmv の場合)
配列 DIMENSION は少なくとも (1 + (*n* - 1)*abs(*incy*))。増分された配列 *y* に *n* 個の成分を持つベクトル *y* を格納しなければならない。

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

出力パラメータ

y 更新されたベクトル *y* によって上書きされる。

?spr

対称パックド・マトリックスの
複素数対称階数 1 の更新を実行する。

```
call cspr( uplo, n, alpha, x, incx, ap )
call zspr( uplo, n, alpha, x, incx, ap )
```

説明

?spr ルーチンは、次のように定義されるマトリックス・ベクトル演算を実行する。

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

alpha は複素スカラー、

x は、*n* 個の成分を持つ複素ベクトル、

a は、パックド形式で与えられる *n* × *n* の複素対称マトリックスである。

このルーチンと実数において等価な関数が BLAS にある (第 2 章「?spr」を参照)。

入力パラメータ

uplo CHARACTER*1。次に示すように、マトリックス *A* の上三角部分と下三角部分のどちらをパックド形式の配列 *ap* に与えるかを指定する。

	uplo = 'U' または 'u' の場合、マトリックス *A* の上三角部分を配列 *ap* に格納する。 *uplo* = 'L' または 'l' の場合、マトリックス *A* の下三角部分を配列 *ap* に格納する。
n	INTEGER。マトリックス *a* の次数を指定する。*n* の値はゼロ以上でなければならない。
alpha	COMPLEX (cspr の場合) COMPLEX*16 (zspr の場合) スカラー *alpha* を指定する。
x	COMPLEX (cspr の場合) COMPLEX*16 (zspr の場合) 配列 DIMENSION は少なくとも (1 + (*n* - 1)*abs(*incx*))。増分された配列 *x* に *n* 個の成分を持つベクトル *x* を格納しなければならない。
incx	INTEGER。*x* の成分に対する増分を指定する。*incx* の値はゼロであってはならない。
ap	COMPLEX (cspr の場合) COMPLEX*16 (zspr の場合) 配列 DIMENSION は少なくとも ((*n**(*n* + 1))/2)。 *uplo* = 'U' または 'u' と指定した場合は、*ap*(1) に *a*(1, 1) が、*ap*(2) に *a*(1, 2) が、*ap*(3) に *a*(2, 2) がそれぞれ格納されるように、配列 *ap* には順番にパックされた対称マトリックスの上三角部分を列ごとに格納しなければならない。 *uplo* = 'L' または 'l' と指定した場合は、*ap*(1) に *a*(1, 1) が、*ap*(2) に *a*(2, 1) が、*ap*(3) に *a*(3, 1) がそれぞれ格納されるように、配列 *ap* には順番にパックされた対称マトリックスの下三角部分を列ごとに格納しなければならない。 直交成分の虚数部分はゼロとみなすため設定する必要はなく、また、出力ではゼロに設定される。

出力パラメータ

ap　　*uplo* = 'U' または 'u' の場合は更新されたマトリックスの上三角部分によって上書きされる。

uplo = 'L' または 'l' の場合は更新されたマトリックスの下三角部分によって上書きされる。

?symv

複素対称マトリックスに対して
マトリックス・ベクトル積を計算する。

```
call csymv ( uplo, n, alpha, a, lda, x, incx, beta, y, incy )
call zsymv ( uplo, n, alpha, a, lda, x, incx, beta, y, incy )
```

説明

これらのルーチンは、次のように定義されるマトリックス・ベクトル演算を実行する。

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

alpha と *beta* は、複素スカラー、

x と *y* は、*n* 個の成分を持つ複素ベクトル、

a は *n* × *n* の対称複素マトリックスである。

このルーチンと実数において等価な関数が BLAS にある (第 2 章「?symv」を参照)。

入力パラメータ

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

	uplo = 'U' または 'u' の場合、配列 *a* の上三角部分が参照される。 *uplo* = 'L' または 'l' の場合、配列 *a* の下三角部分が参照される。
n	INTEGER。マトリックス *A* の次数を指定する。*n* の値はゼロ以上でなければならない。
alpha, beta	COMPLEX (csymv の場合) COMPLEX*16 (zsymv の場合) スカラー *alpha* と *beta* を指定する。*beta* をゼロとして与える場合は入力 *y* を設定する必要はない。
a	COMPLEX (csymv の場合) COMPLEX*16 (zsymv の場合) 配列 DIMENSION は (*lda, n*)。*uplo* = 'U' または 'u' と指定した場合は、配列 *a* の先頭の *n* × *n* の上三角部分には対称マトリックスの上三角部分を格納しなければならず、*a* の厳密な意味での下三角部分は参照されない。また、*uplo* = 'L' または 'l' と指定した場合は、配列 *a* の先頭の *n* × *n* の下三角部分には対称マトリックスの下三角部分を格納しなければならず、*a* の厳密な意味での上三角部分は参照されない。
lda	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、*a* の第 1 次元を指定する。*lda* の値は、少なくとも max(1, *n*) でなければならない。
x	COMPLEX (csymv の場合) COMPLEX*16 (zsymv の場合) 配列 DIMENSION は少なくとも (1 + (*n* - 1)*abs(*incx*))。増分された配列 *x* に *n* 個の成分を持つベクトル *x* を格納しなければならない。
incx	INTEGER。*x* の成分に対する増分を指定する。*incx* の値はゼロであってはならない。
y	COMPLEX (csymv の場合) COMPLEX*16 (zsymv の場合)

配列 DIMENSION は少なくとも (1 + (*n* - 1)*abs(*incy*))。増分された配列 *y* に *n* 個の成分を持つベクトル *y* を格納しなければならない。

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

出力パラメータ

y 更新されたベクトル *y* によって上書きされる。

?syr

複素対称マトリックスの対称階数 1 の更新を実行する。

```
call csyr( uplo, n, alpha, x, incx, a, lda )
call zsyr( uplo, n, alpha, x, incx, a, lda )
```

説明

このルーチンは、次のように定義される対称階数 1 の演算を実行する。

*a := alpha*x*x' + a,*

alpha は、複素スカラー、

x は、*n* 個の成分を持つ複素ベクトル、

a は、*n* × *n* の複素対称マトリックスである。

このルーチンと実数において等価な関数が BLAS にある (第 2 章の「?syr」を参照)。

入力パラメータ

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

	uplo = 'U' または 'u' の場合、配列 *a* の上三角部分が参照される。 *uplo* = 'L' または 'l' の場合、配列 *a* の下三角部分が参照される。
n	INTEGER。マトリックス *a* の次数を指定する。*n* の値はゼロ以上でなければならない。
alpha	COMPLEX (csyr の場合) COMPLEX*16 (zsyr の場合) スカラー *alpha* を指定する。
x	COMPLEX (csyr の場合) COMPLEX*16 (zsyr の場合) 配列 DIMENSION は少なくとも (1 + (*n* - 1)*abs(*incx*))。増分された配列 *x* に *n* 個の成分を持つベクトル *x* を格納しなければならない。
incx	INTEGER。*x* の成分に対する増分を指定する。*incx* の値はゼロであってはならない。
a	COMPLEX (csyr の場合) COMPLEX*16 (zsyr の場合) 配列 DIMENSION は (*lda*, *n*)。*uplo* = 'U' または 'u' と指定した場合は、配列 *a* の先頭の *n* × *n* の上三角部分には対称マトリックスの上三角部分を格納しなければならず、*a* の厳密な意味での下三角部分は参照されない。 *uplo* = 'L' または 'l' と指定した場合は、配列 *a* の先頭の *n* × *n* の下三角部分には対称マトリックスの下三角部分を格納しなければならず、*a* の厳密な意味での上三角部分は参照されない。
lda	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、*a* の第 1 次元を指定する。*lda* の値は、少なくとも max(1, *n*) でなければならない。

出力パラメータ

a

uplo = 'U' または 'u' の場合、配列 *a* の上三角部分が更新されたマトリックスの上三角部分によって上書きされる。

uplo = 'L' または 'l' の場合、配列 *a* の下三角部分が更新されたマトリックスの下三角部分によって上書きされる。

i?max1

実数部分が最大絶対値を持つベクトルの成分のインデックスを検出する。

```
index = icmax1 ( n, cx, incx )
index = izmax1 ( n, cx, incx )
```

説明

複素ベクトル *cx* が与えられたとき、i?max1 関数は実数部分が最大絶対値を持つベクトル成分のインデックスを返す。これら関数は BLAS 関数 icamax/izamax を元にしているが、実数部分の絶対値を使用している。clacon/zlacon との併用を意図している。

入力パラメータ

n

INTEGER。ベクトル *cx* の要素の数を指定する。

cx

COMPLEX (icmax1 の場合)
COMPLEX*16 (izmax1 の場合)
配列 DIMENSION は少なくとも
(1+(*n*-1)*abs(*incx*))
入力ベクトルが格納される。

incx INTEGER。*cx* の連続する成分の間隔を指定する。

出力パラメータ

index INTEGER。実数部分が最大絶対値を持つベクトル成分のインデックスが格納される。

ilaenv

アルゴリズム性能のチューニング値を
戻す環境問い合わせ関数。

```
value = ilaenv ( ispec, name, opts, n1, n2, n3, n4 )
```

説明

問い合わせ関数 *ilaenv* はローカル環境に対する問題依存パラメータを選択するために LAPACK ルーチンから呼び出される。パラメータの説明は *ispec* を参照のこと。

このバージョンは、現在の多くのコンピュータ上で最適ではないが、良好な性能を得る一連のパラメータを与える。ユーザは、引数中のオプションと問題サイズ情報を使って、使用対象のマシンに対するチューニング・パラメータを設定するために、このサブルーチンを変更することが奨励される。

このルーチンはすべて小文字にすると正しく機能しない。すべて大文字で使用することは許される。

入力パラメータ

ispec INTEGER。ilaenv の戻り値として得たいパラメータを指定する。

= 1: 最適ブロック・サイズ。戻り値が 1 のとき、非ブロック化アルゴリズムで最適な性能が得られる。

= 2: ブロック・ルーチンを使用するべき最小ブロック・サイズ。使用できるブロック・サイズがこの値未満の場合は、非ブロック化ルーチンを使用すべきである。

= 3: クロスオーバー点 (ブロック・ルーチンでは、この値未満の *N* に対して、非ブロック化ルーチンを使用すべきである)。

= 4: シフト数。非対称の固有値ルーチンで使用される。

= 5: ブロック化が使用される最小の列次元。矩形ブロックは少なくとも `k` × `m` の次元を持っていなければならず、ここで `k` は `ilaenv(2, ...)` で与えられ、`m` は `ilaenv(5, ...)` で与えられる。

= 6: SVD に対するクロスオーバー点 (*m* × *n* のマトリックスを二重対角形式に縮退する場合、max(`m, n`)/min(`m, n`) がこの値を超えたなら、マトリックスを三角形式に縮退するために最初に *QR* 因子分解が使用される)。

= 7: プロセッサの個数。

= 8: 非対称固有値問題に対するマルチ。シフト *QR* および *QZ* 法のクロスオーバー点

= 9: 分割統治アルゴリズムにおける計算ツリーの一番下にある部分問題の最大サイズ (`?gelsd` と `?gesdd` で使用する)。

= 10: トラップを生じないと信頼される IEEE NaN 演算。

= 11: トラップを生じないと信頼される無限大演算。

name `CHARACTER*(*)`。呼び出しサブルーチンの名称で、大文字でも小文字でもよい。

opts `CHARACTER*(*)`。サブルーチン *name* に対する文字オプションを単一文字列に連結したもの。たとえば、*uplo* = `'U'`、`trans` = `'T'`、`diag` = `'N'` の三角ルーチンは `opts` = `'UTN'` として指定してもよい。

n1, n2, n3, n4 `INTEGER`。サブルーチン *name* に対する問題の次元。すべてのパラメータが必要でない場合もある。

出力パラメータ

value INTEGER

value ≥ 0 の場合、*ispec* で指定されたパラメータ値が返される。

value = -k < 0 の場合、k 番目の引数が不正な値であることを示す。

アプリケーション・ノート

LAPACK ルーチンから ilaenv を呼び出すときには以下の規則が使用される。

1) *opts* は、*ispec* で指定したパラメータ値を求めるために使用しない場合でも、*name* の引数リストと同じ順番でサブルーチン *name* のすべての文字オプションを連結したものである。

2) 問題次元 *n1, n2, n3, n4* は *name* の引数リストと同じ順番で指定する。*n1* が最初に使用され、次に *n2* が使われ、以下同様に続く。使用されない問題次元には値 -1 が渡される。

3) ilaenv が戻すパラメータ値は呼び出しサブルーチン内で有効性をチェックする。たとえば、strtri の最適ブロック・サイズを検索するには ilaenv を次のように使用する。

```
nb = ilaenv( 1, 'strtri', uplo // diag, n, -1, -1, -1 )
if( nb.le.1 ) nb = max( 1, n )
```

lsame

2 つの文字の同一性を大文字 / 小文字に関わらずテストする。

```
val = lsame ( ca, cb )
```

説明

この論理関数は、*ca* と *cb* が大文字 / 小文字に関わらず同一の文字のときに .TRUE. を返す。

入力パラメータ

ca, cb CHARACTER*1。比較する単一文字を指定する。

出力パラメータ

val LOGICAL。比較結果を示す。

lsamen

2 つの文字列の同一性を大文字 / 小文字に関わらずテストする。

```
val = lsamen ( n, ca, cb )
```

説明

この論理関数は、文字列 *ca* の最初の *n* 文字と文字列 *cb* の最初の *n* 文字が大文字 / 小文字に関わらず、等しいかどうかをテストする。lsamen 関数は大文字 / 小文字に関わらず、*ca* と *cb* が等価のときに .TRUE. を返し、それ以外では .FALSE. を返す。len(*ca*) または len(*cb*) が *n* 未満のときにも、lsamen は .FALSE. を返す。

入力パラメータ

n INTEGER。*ca* と *cb* を比較する文字数。

ca, cb CHARACTER*(*)。少なくとも長さ *n* を持った 2 つの比較する文字列を指定する。それぞれの文字列は先頭の *n* 文字のみアクセスされる。

出力パラメータ

val LOGICAL。比較結果を示す。

?sum1

真の絶対値を用いて複素ベクトルの
1- ノルムを実行する。

```
res = scsum1 ( n, cx, incx )
res = dzsum1 ( n, cx, incx )
```

説明

複素ベクトル cx が与えられたとき、scsum1/dzsum1 関数は、ベクトル成分の絶対値の合計を取り、それぞれ単精度 / 倍精度の結果を返す。これら関数はレベル 1 BLAS の scasum/dzasum を元にしているが、真の絶対値を使用し、clacon/zlacon と併せて使用するように設計されている。

入力パラメータ

n	INTEGER。ベクトル *cx* の要素の数を指定する。
cx	COMPLEX (scsum1 の場合) COMPLEX*16 (dzsum1 の場合) 配列 DIMENSION は少なくとも (1+(*n*-1)*abs(*incx*)) 成分を合計する入力ベクトルを格納する。
incx	INTEGER。*cx* の連続する成分間の間隔を指定する (*incx* > 0)。

出力パラメータ

res	REAL (scsum1 の場合) DOUBLE PRECISION (dzsum1 の場合) 絶対値の合計が格納される。

?gbtf2

非ブロック化バージョンのアルゴリズムを使って一般バンド・マトリックスの *LU* 因子分解を計算する。

```
call sgbtf2 ( m, n, kl, ku, ab, ldab, ipiv, info )
call dgbtf2 ( m, n, kl, ku, ab, ldab, ipiv, info )
call cgbtf2 ( m, n, kl, ku, ab, ldab, ipiv, info )
call zgbtf2 ( m, n, kl, ku, ab, ldab, ipiv, info )
```

説明

このルーチンは、*kl* 個の劣対角と *ku* 個の優対角を含む $m \times n$ の一般実数 / 複素バンド・マトリックス *A* の *LU* 因子分解を実行する。このルーチンは行交換を伴なう部分ピボッティングを使用しており、また、非ブロック化ルーチンのアルゴリズムを実装している。レベル 2 BLAS を呼び出す。?gbtrf も参照のこと。

入力パラメータ

m	INTEGER。マトリックス *A* の行数 ($m \geq 0$)。
n	INTEGER。*A* の列数 $n \geq 0$)。
kl	INTEGER。*A* のバンドの中の劣対角成分の数 ($kl \geq 0$)。
ku	INTEGER。*A* のバンドの中の優対角成分の数 ($ku \geq 0$)。
ab	REAL (sgbtf2 の場合) DOUBLE PRECISION (dgbtf2 の場合) COMPLEX (cgbtf2 の場合) COMPLEX*16 (zgbtf2 の場合) 配列 DIMENSION は (*ldab*, *) 配列 *ab* にはマトリックス *A* をバンド格納形式で格納する (「マトリックス引数」を参照)。 *ab*の第2次元は少なくともmax(1, *n*)でなければならない。
ldab	INTEGER。配列 *ab* の第 1 次元。($ldab \geq 2kl + ku + 1$)

出力パラメータ

ab
因数分解の詳細で上書きされる。U の対角成分と *kl* + *ku* 個の優対角成分は *ab* の最初の 1 + *kl* + *ku* 行に格納される。因子分解中に使用された乗数は次の *kl* 行に格納される。

ipiv
INTEGER
配列 DIMENSION は少なくとも max(1, min(*m*, *n*))
ピボット (要) のインデックス：行 *i* は *ipiv*(*i*) と交換されている。

info
INTEGER。*info* = 0 の場合、正常に終了したことを示す。
info = -*i* の場合は *i* 番目のパラメータの値が不正だったことを示す。
info = *i* の場合は u_{ii} はゼロである。因子分解は完了したが U は完全に特異である。連立 1 次方程式の解の算出に係数 U を使用すると 0 による除算が発生する。

?gebd2

非ブロック化アルゴリズムを使って一般
マトリックスを二重対角形式に縮退させる。

```
call sgebd2 ( m, n, a, lda, d, e, tauq, taup, work, info )
call dgebd2 ( m, n, a, lda, d, e, tauq, taup, work, info )
call cgebd2 ( m, n, a, lda, d, e, tauq, taup, work, info )
call zgebd2 ( m, n, a, lda, d, e, tauq, taup, work, info )
```

説明

このルーチンは直交 / ユニタリ変換を用いて、$m \times n$ の一般マトリックス A を上または下二重対角形式 B に縮退させる。

$$Q' A P = B$$

$m \geq n$ の場合、B は上二重対角である。$m < n$ の場合、B は下二重対角である。

このルーチンでは、マトリックス Q および P を明示的な形式では表現せず、基本リフレクタの積として表現する。

$m \geq n$ の場合、

$Q = H(1)H(2) \ldots \mathrm{H}(n)$ および $\mathrm{P} = G(1)G(2) \ldots G(n-1)$

$m < n$ の場合、

$Q = H(1)H(2) \ldots \mathrm{H}(m-1)$ および $\mathrm{P} = G(1)G(2) \ldots G(m)$

それぞれの $H(\mathrm{i})$ と $G(\mathrm{i})$ は形式を持っている。

$H(\mathrm{i}) = I - tauq*v*v'$ および $G(\mathrm{i}) = I - taup*u*u'$

tauq と *taup* はスカラー (sgebd2/dgebd2 では実数、cgebd2/zgebd2 では複素)、*v* と *u* はベクトルである (sgebd2/dgebd2 では実数、cgebd2/zgebd2 では複素)。

入力パラメータ

m	INTEGER。マトリックス A の行数 ($m \geq 0$)。
n	INTEGER。A の列数 ($n \geq 0$)。
a, *work*	REAL (sgebd2 の場合) DOUBLE PRECISION (dgebd2 の場合) COMPLEX (cgebd2 の場合) COMPLEX*16 (zgebd2 の場合)

配列：
a(lda, *) には縮退する $m \times n$ の一般マトリックス A を格納する。a の第 2 次元は、少なくとも max(1, n) でなければならない。
work(*) はワークスペース配列、work の値は、少なくとも max(1, m, n) でなければならない。

lda　INTEGER。a の第 1 次元。max(1, m) 以上であること。

出力パラメータ

a　$m \geq n$ の場合、a の対角および第 1 の優対角成分は上二重対角マトリックス B によって上書きされる。対角よりも下の成分は、配列 tauq とともに、基本リフレクタの積として直交 / ユニタリのマトリックス Q を表現する。また、第 1 の優対角よりも上の成分は、配列 taup とともに、基本リフレクタの積として直交 / ユニタリのマトリックス P を表現する。
$m < n$ の場合、a の対角および第 1 の劣対角成分は下二重対角マトリックス B によって上書きされる。第 1 の劣対角よりも下の成分は、配列 tauq とともに、基本リフレクタの積として直交 / ユニタリのマトリックス Q を表現する。また、対角よりも上の成分は、配列 taup とともに、基本リフレクタの積として直交 / ユニタリのマトリックス P を表現する。

d　REAL (単精度の場合)、
DOUBLE PRECISION (倍精度の場合)
配列 DIMENSION は少なくとも max(1, min(m, n))
二重対角マトリックス B の対角成分 (d(i) = a(i, i)) が格納される。

e　REAL (単精度の場合)、D
OUBLE PRECISION (倍精度の場合)
配列 DIMENSION は少なくとも max(1, min(m, n) - 1)
二重対角マトリックス B の対角外の成分が次のように格納される。

	m ≥ *n* のとき、i = 1, 2, ..., *n*-1 に対して *e*(i) = *a*(i, i+1)、 *m* < *n* のとき、i = 1, 2, ..., *m*-1 に対して *e*(i) = *a*(i+1, i)
tauq, taup	REAL (sgebd2 の場合) DOUBLE PRECISION (dgebd2 の場合) COMPLEX (cgebd2 の場合) COMPLEX*16 (zgebd2 の場合) 配列 DIMENSION は少なくとも max(1, min(*m*, *n*)) 直交 / ユニタリ・マトリックス *Q* と *P* をそれぞれ表す基本リフレクタのスカラー係数が格納される。
info	INTEGER *info* = 0 の場合、正常に終了したことを示す。 *info* = -*i* の場合 *i* 番目のパラメータの値が不正だったことを示す。

?gehd2

非ブロック化アルゴリズムを使用して
一般正方マトリックスを上 Hessenberg
形式に縮退させる。

```
call sgehd2 ( n, ilo, ihi, a, lda, tau, work, info )
call dgehd2 ( n, ilo, ihi, a, lda, tau, work, info )
call cgehd2 ( n, ilo, ihi, a, lda, tau, work, info )
call zgehd2 ( n, ilo, ihi, a, lda, tau, work, info )
```

説明

このルーチンは、直交またはユニタリの相似変換 $Q'A\,Q = H$ によって、実数 / 複素一般マトリックス A を上 Hessenberg 形式 H に縮退させる。

このルーチンではマトリックス Q を明示的に表現しない。代わりに、Q は**基本リフレクタ**の積として表現される。

入力パラメータ

n	INTEGER。マトリックス A の次数 ($n \geq 0$)。
ilo, ihi	INTEGER。A はすでに行 1:*ilo* -1 と列 *ihi*+1:*n* で上三角になっていると仮定する。*?gebal* の出力を A として使用する場合は、*ilo* と *ihi* には *?gebal* ルーチンが返した値を使用しなければならない。それ以外の場合は *ilo* = 1 かつ *ihi* = *n* を設定する。 $1 \leq ilo \leq ihi \leq \max(1, n)$ の制約がある。
a, work	REAL (sgehd2 の場合) DOUBLE PRECISION (dgehd2 の場合) COMPLEX (cgehd2 の場合) COMPLEX*16 (zgehd2 の場合) 配列： *a* (*lda*, *) には縮退させる $n \times n$ のマトリックス A を格納する。*a* の第 2 次元は、少なくとも max(1, *n*) でなければならない。 *work* (*n*) は、ワークスペース配列。

lda	INTEGER。*a* の第 1 次元。max(1, *n*) 以上であること。

出力パラメータ

a	*A* の上三角と第 1 の劣対角は、上 Hessenberg マトリックス *H* および第 1 の劣対角よりも下の成分で上書きされ、配列 *tau* とともに、基本リフレクタの積として直交 / ユニタリのマトリックス *Q* を表現する。次の**アプリケーション・ノート**を参照。
tau	REAL (sgehd2 の場合) DOUBLE PRECISION (dgehd2 の場合) COMPLEX (cgehd2 の場合) COMPLEX*16 (zgehd2 の場合) 配列 DIMENSION は少なくとも max (1, *n*-1) 基本リフレクタのスカラー係数が格納される。次の**アプリケーション・ノート**を参照。
info	INTEGER *info* = 0 の場合、正常に終了したことを示す。 *info* = -*i* の場合、*i* 番目のパラメータの値が不正であったことを示す。

アプリケーション・ノート

マトリックス *Q* は (*ihi* - *ilo*) の基本リフレクタの積として表現される。

$Q = H(ilo)\, H(ilo+1)\, .\, .\, H(ihi-1)$

それぞれの *H*(i) は形式を持っている。

$H(i) = I - tau * v * v'$

tau は実数 / 複素スカラー、*v* は実数 / 複素ベクトルで、
v(1:i) = 0, *v*(i+1) = 1 および *v*(*ihi*+1:*n*) = 0 である。

ルーチン終了時には、*v*(i+2:*ihi*) は *a*(i+2:*ihi*, i) に格納され、*tau* は *tau*(i) に格納される。

n = 7、*ilo* = 2、*ihi* = 6 の場合、*a* の内容を次の例に示す。

入力

$$\begin{bmatrix} a & a & a & a & a & a & a \\ & a & a & a & a & a & a \\ & & a & a & a & a & a \\ & & & a & a & a & a \\ & & & & a & a & a \\ & & & & & a & a \\ & & & & & & a \end{bmatrix}$$

出力

$$\begin{bmatrix} a & a & h & h & h & h & a \\ & a & h & h & h & h & a \\ & h & h & h & h & h & h \\ & v_2 & h & h & h & h & h \\ & v_2 & v_3 & h & h & h & h \\ & v_2 & v_3 & v_4 & h & h & h \\ & & & & & & a \end{bmatrix}$$

ここで a は元のマトリックス A の成分を表し、h は上 Hessenberg マトリックス H の更新された成分を表し、v_i は $H(i)$ を定義するベクトルの成分を表している。

?gelq2

非ブロック化アルゴリズムを使用して
一般矩形マトリックスの *LQ* 因子分解を
計算する。

```
call sgelq2 ( m, n, a, lda, tau, work, info )
call dgelq2 ( m, n, a, lda, tau, work, info )
call cgelq2 ( m, n, a, lda, tau, work, info )
call zgelq2 ( m, n, a, lda, tau, work, info )
```

説明

このルーチンは実数 / 複素の $m \times n$ マトリックス A の *LQ* 因子分解を $A = L\,Q$ として計算する。

このルーチンではマトリックス Q を明示的な形式では表現しない。代わりに、Q は min(m, n) の**基本リフレクタ**の積として表現される。

$Q = H(\mathrm{k}) \ldots H(2)\,H(1)$ (または複素の場合、$Q = H(\mathrm{k})' \ldots H(2)'\,H(1)'$)、ここで k = min($m$, n)

それぞれの H(i) は形式を持っている。

$H(\mathrm{i}) = I - tau*v*v'$

tau は `tau`(i) に格納される実数 / 複素スカラー、v は v(1:i-1) = 0 および v(i) = 1 を持つ実数 / 複素ベクトルである。

ルーチン終了時に v(i+1:n) は `a`(i, i+1:n) に格納される。

入力パラメータ

`m`	`INTEGER`。マトリックス A の行数 ($m \geq 0$)。
`n`	`INTEGER`。A の列数 ($n \geq 0$)。
`a, work`	`REAL` (`sgelq2` の場合) `DOUBLE PRECISION` (`dgelq2` の場合) `COMPLEX` (`cgelq2` の場合) `COMPLEX*16` (`zgelq2` の場合)

配列：
a(*lda*, *)には *m* × *n* のマトリックス *A* を格納する。*a* の第 2 次元は、少なくとも max(1, *n*) でなければならない。
work(*m*) は、ワークスペース配列。

lda INTEGER。*a* の第 1 次元。max(1, *m*) 以上であること。

出力パラメータ

a 因子分解のデータによって、次のように上書きされる。
配列 *a* の対角成分とその下の成分は *m* × *n*(*n*, *m*) の下台形マトリックス *L* (*n* ≥ *m* の場合、*L* は下三角マトリックス) で上書きされる。対角よりも上の成分は、配列 *tau* とともに、min(*n*, *m*) の基本リフレクタ積として直交 / ユニタリのマトリックス *Q* を表現する。

tau REAL (sgelq2 の場合)
DOUBLE PRECISION (dgelq2 の場合)
COMPLEX (cgelq2 の場合)
COMPLEX*16 (zgelq2 の場合)
配列 DIMENSION は少なくとも max (1, min(*m*, *n*))
基本リフレクタのスカラー係数が入る。

info INTEGER
info = 0 の場合、正常に終了したことを示す。
info = -*i* の場合、*i* 番目のパラメータの値が不正であったことを示す。

?geql2

非ブロック化アルゴリズムを使用して
一般矩形マトリックスの *QL* 因子分解を
計算する。

```
call sgeql2 ( m, n, a, lda, tau, work, info )
call dgeql2 ( m, n, a, lda, tau, work, info )
call cgeql2 ( m, n, a, lda, tau, work, info )
call zgeql2 ( m, n, a, lda, tau, work, info )
```

説明

このルーチンは実数 / 複素の $m \times n$ マトリックス A の QL 因子分解を $A = QL$ として計算する。

このルーチンはマトリックス Q を明示的な形式では表現しない。代わりに、Q は $\min(m, n)$ の**基本リフレクタ**の積として表現される。

$Q = H(k) \ldots H(2)\, H(1)$、ここで $k = \min(m, n)$

それぞれの $H(i)$ は形式を持っている。

$H(i) = I - tau * v * v'$

tau は `tau`(i) に格納される実数 / 複素スカラー、v は $v(m\text{-}k\text{+}i\text{+}1{:}m) = 0$ および $v(m\text{-}k\text{+}i) = 1$ を持つ実数 / 複素ベクトルである。

ルーチン終了時には $v(1{:}m\text{-}k\text{+}i\text{-}1)$ は $a(1{:}m\text{-}k\text{+}i\text{-}1,\ n\text{-}k\text{+}i)$ に格納される。

入力パラメータ

`m`	`INTEGER`。マトリックス A の行数 ($m \geq 0$)。
`n`	`INTEGER`。A の列数 ($n \geq 0$)。
`a, work`	`REAL` (`sgeql2` の場合) `DOUBLE PRECISION` (`dgeql2` の場合) `COMPLEX` (`cgeql2` の場合) `COMPLEX*16` (`zgeql2` の場合)

配列：
a(lda, *) には m × n のマトリックス A を格納する。a の第 2 次元は、少なくとも max(1, n) でなければならない。
work(m) は、ワークスペース配列。

lda　INTEGER。a の第 1 次元。max(1, m) 以上であること。

出力パラメータ

a　因子分解のデータによって、次のように上書きされる。
m ≥ n の場合は、部分配列 a(m-n+1:m, 1:n) の下三角部分に n × n の下三角マトリックス L が格納される。
m < n の場合は、(n-m) 番目の優対角成分とその下の各成分に m × n の下台形マトリックス L が格納される。
残りの成分は、配列 tau とともに、基本リフレクタの積として直交 / ユニタリ・マトリックス Q を表現する。

tau　REAL (sgeql2 の場合)
DOUBLE PRECISION (dgeql2 の場合)
COMPLEX (cgeql2 の場合)
COMPLEX*16 (zgeql2 の場合)
配列 DIMENSION は少なくとも max(1, min(m,n))
基本リフレクタのスカラー係数が入る。

info　INTEGER
info = 0 の場合、正常に終了したことを示す。
info = -i の場合、i 番目のパラメータの値が不正であったことを示す。

?geqr2

非ブロックアルゴリズムを使用して一般
矩形マトリックスの *QR* 因子分解を
計算する。

```
call sgeqr2 ( m, n, a, lda, tau, work, info )
call dgeqr2 ( m, n, a, lda, tau, work, info )
call cgeqr2 ( m, n, a, lda, tau, work, info )
call zgeqr2 ( m, n, a, lda, tau, work, info )
```

説明

このルーチンは実数 / 複素の $m \times n$ マトリックス A の QR 因子分解を $A = QR$ として計算する。

このルーチンはマトリックス Q を明示的な形式では表現しない。代わりに、Q は min(m, n) の**基本リフレクタ**の積として表現される。

$Q = H(1)H(2) \ldots H(k)$、ここで $k = \min(m, n)$

それぞれの $H(i)$ は形式を持っている。

$H(i) = I - tau * v * v'$

tau は *tau*(i) に格納される実数 / 複素スカラー、v は $v(1:i-1) = 0$ および $v(i) = 1$ を持つ実数 / 複素ベクトルである。

ルーチン終了時に $v(i+1:m)$ は *a*(i+1:*m*, i) に格納される。

入力パラメータ

m	INTEGER。マトリックス A の行数 ($m \geq 0$)。
n	INTEGER。A の列数 ($n \geq 0$)。
a, *work*	REAL (sgeqr2 の場合) DOUBLE PRECISION (dgeqr2 の場合) COMPLEX (cgeqr2 の場合) COMPLEX*16 (zgeqr2 の場合)

配列：
a(lda, *) には m × n のマトリックス A を格納する。a の第 2 次元は、少なくとも max(1, n) でなければならない。
work(n) は、ワークスペース配列。

lda　INTEGER。a の第 1 次元。max(1, m) 以上であること。

出力パラメータ

a　因子分解のデータによって、次のように上書きされる。
マトリックスの対角成分とその上の成分は、min(n, m) × n の上台形マトリックス R によって上書きされる (m ≥ n のとき R は上三角マトリックス)。対角よりも下の成分は、配列 tau とともに、基本リフレクタの積として直交 / ユニタリ・マトリックス Q を表現する。

tau　REAL (sgeqr2 の場合)
DOUBLE PRECISION (dgeqr2 の場合)
COMPLEX (cgeqr2 の場合)
COMPLEX*16 (zgeqr2 の場合)
配列 DIMENSION は少なくとも max(1, min(m, n))
基本リフレクタのスカラー係数が入る。

info　INTEGER
info = 0 の場合、正常に終了したことを示す。
info = -i の場合、i 番目のパラメータの値が不正であったことを示す。

?gerq2

非ブロック化アルゴリズムを使用して
一般矩形マトリックスの *RQ* 因子分解を
計算する。

```
call sgerq2 ( m, n, a, lda, tau, work, info )
call dgerq2 ( m, n, a, lda, tau, work, info )
call cgerq2 ( m, n, a, lda, tau, work, info )
call zgerq2 ( m, n, a, lda, tau, work, info )
```

説明

このルーチンは実数 / 複素の $m \times n$ マトリックス A の RQ 因子分解を $A = RQ$ として計算する。

このルーチンはマトリックス Q を明示的な形式では表現しない。代わりに、Q は min(m, n) の**基本リフレクタ**の積として表現される。

$Q = H(1)H(2) \ldots H(k)$、ここで $k = \min(m, n)$

それぞれの $H(i)$ は形式を持っている。

$H(i) = I - tau * v * v'$

tau は *tau*(i) に格納される実数 / 複素スカラー、v は $v(n-k+i+1:n) = 0$ および $v(n-k+i) = 1$ を持つ実数 / 複素ベクトルである。

ルーチン終了時に $v(1:n-k+i-1)$ は $a(m-k+i, 1:n-k+i-1)$ に格納される。

入力パラメータ

m	INTEGER。マトリックス A の行数 ($m \geq 0$)。
n	INTEGER。A の列数 ($n \geq 0$)。
a, work	REAL (sgerq2 の場合) DOUBLE PRECISION (dgerq2 の場合) COMPLEX (cgerq2 の場合) COMPLEX*16 (zgerq2 の場合)

配列：
a(lda, *) には m × n のマトリックス A を格納する。a の第 2 次元は、少なくとも max(1, n) でなければならない。
work(m) は、ワークスペース配列。

lda　INTEGER。a の第 1 次元。max(1, m) 以上であること。

出力パラメータ

a　因子分解のデータによって、次のように上書きされる。
m ≤ n の場合は、部分配列 a(1:m, n-m+1:n) の上三角部分に、m × m の上三角マトリックス R が格納される。
m > n の場合は、(m-n) 番目の劣対角成分とその上の各成分に、m × n の上台形マトリックス R が格納される。残りの成分は、配列 tau とともに、min(m, n) 個の基本リフレクタの積として直交 / ユニタリ・マトリックス Q を表現する。

tau　REAL (sgerq2 の場合)
DOUBLE PRECISION (dgerq2 の場合)
COMPLEX (cgerq2 の場合)
COMPLEX*16 (zgerq2 の場合)
配列 DIMENSION は少なくとも max(1, min(m, n))
基本リフレクタのスカラー係数が入る。

info　INTEGER
info = 0 の場合、正常に終了したことを示す。
info = -i の場合、i 番目のパラメータの値が不正であったことを示す。

?gesc2

?getc2 で計算された完全ピボッティングによる *LU* 因子分解を使用して連立一次方程式を解く。

```
call sgesc2 ( n, a, lda, rhs, ipiv, jpiv, scale )
call dgesc2 ( n, a, lda, rhs, ipiv, jpiv, scale )
call cgesc2 ( n, a, lda, rhs, ipiv, jpiv, scale )
call zgesc2 ( n, a, lda, rhs, ipiv, jpiv, scale )
```

説明

このルーチンは次の連立一次方程式を、

$$AX = scale * RHS$$

?getc2で計算された完全ピボッティングによる*LU*因子分解を使用して、一般 $n \times n$ マトリックス *A* について解く。

入力パラメータ

n	INTEGER。マトリックス *A* の次数。
a, *rhs*	REAL (sgesc2 の場合) DOUBLE PRECISION (dgesc2 の場合) COMPLEX (cgesc2 の場合) COMPLEX*16 (zgesc2 の場合) 配列： *a*(*lda*, *)には?getc2で計算された $n \times n$ マトリックス *A* の因子分解の *LU* 部分を格納する。 $A = P\,L\,U\,Q$ *a* の第 2 次元は、少なくとも max(1, *n*) でなければならない。 *rhs*(*n*) には連立一次方程式の右辺ベクトルを格納する。
lda	INTEGER。*a* の第 1 次元。max(1, *n*) 以上であること。

ipiv INTEGER

配列 DIMENSION は少なくとも max(1, *n*)
ピボット・インデックスである。1 ≤ i ≤ *n* で、マトリックスの行 i は行 *ipiv*(i) と交換されている。

jpiv INTEGER

配列 DIMENSION は少なくとも max(1, *n*)
ピボット・インデックスである。1 ≤ j ≤ *n* で、マトリックスの列 j は列 *jpiv*(j) と交換されている。

出力パラメータ

rhs 解のベクトル *X* で上書きされる。

scale REAL (sgesc2/cgesc2 の場合)
DOUBLE PRECISION (dgesc2/zgesc2 の場合)
スケール係数が格納される。*scale* は解におけるオーバーフローを防ぐため 0 ≤ *scale* ≤ 1 の範囲で選択される。

?getc2

一般 $n \times n$ マトリックスの完全
ピボッティングによる *LU* 因子分解を
計算する。

```
call sgetc2 ( n, a, lda, ipiv, jpiv, info )
call dgetc2 ( n, a, lda, ipiv, jpiv, info )
call cgetc2 ( n, a, lda, ipiv, jpiv, info )
call zgetc2 ( n, a, lda, ipiv, jpiv, info )
```

説明

このルーチンは一般 $n \times n$ マトリックスの完全ピボッティングによる *LU* 因子分解を計算する。因子分解は形式 $A = P*L*U*Q$ を持ち、*P* と *Q* は置換マトリックス、*L* は単位対角成分を持つ下三角、*U* は上三角である。

入力パラメータ

n	INTEGER。マトリックス *A* の次数 ($n \geq 0$)。
a	REAL (sgetc2 の場合) DOUBLE PRECISION (dgetc2 の場合) COMPLEX (cgetc2 の場合) COMPLEX*16 (zgetc2 の場合) 配列 *a(lda, *)* には因子分解する $n \times n$ マトリックス *A* を格納する。 *a* の第 2 次元は、少なくとも max(1, *n*) でなければならない。
lda	INTEGER。*a* の第 1 次元。max(1, *n*) 以上であること。

出力パラメータ

a	因子分解 $A = P*L*U*Q$ で得られた係数 *L* と *U* で上書きされる。*L* の単位対角成分は格納されない。*U*(k, k) が *smin* 未満の場合、*U*(k, k) が *smin* の値として与えられる。すなわち、非特異の摂動連立方程式を与える。

ipiv INTEGER
配列 DIMENSION は少なくとも max(1, *n*)
ピボット・インデックスである。$1 \le i \le n$ で、マトリックスの行 i は行 *ipiv*(i) と交換されている。

jpiv INTEGER
配列 DIMENSION は少なくとも max(1, *n*)
ピボット・インデックスである。$1 \le j \le n$ で、マトリックスの列 j は列 *jpiv*(j) と交換されている。

info INTEGER
info = 0 の場合、正常に終了したことを示す。
info = *k* > 0 の場合、$Ax = b$ で x を解こうとすると $U(k, k)$ はおそらくオーバーフローを生む。そこで、オーバーフローを防止するために U は摂動となる。

?getf2

一般 $m \times n$ マトリックスの LU 因子分解を、行交換を伴なう部分ピボッティングを用いて計算する (非ブロック化アルゴリズム)。

```
call sgetf2 ( m, n, a, lda, ipiv, info )
call dgetf2 ( m, n, a, lda, ipiv, info )
call cgetf2 ( m, n, a, lda, ipiv, info )
call zgetf2 ( m, n, a, lda, ipiv, info )
```

説明

このルーチンは一般 $m \times n$ マトリックス A の LU 因子分解を、行交換を伴なう部分ピボッティングを用いて計算する。因子分解は次の形式を持つ。

$$A = PLU$$

P は置換マトリックス、*L* は単位対角成分を持つ下三角 (*m* > *n* の場合下台形)、*U* は上三角である (*m* < *n* の場合上台形)。

入力パラメータ

m INTEGER。マトリックス *A* の行数 (*m* ≥ 0)。

n INTEGER。*A* の列数 (*n* ≥ 0)。

a REAL (sgetf2 の場合)
DOUBLE PRECISION (dgetf2 の場合)
COMPLEX (cgetf2 の場合)
COMPLEX*16 (zgetf2 の場合)
配列 DIMENSION は (*lda*, *)。因数分解の対象となるマトリックス *A* が格納される。*a* の第 2 次元は、少なくとも max(1, *n*) でなければならない。

lda INTEGER。*a* の第 1 次元。max(1, *m*) 以上であること。

出力パラメータ

a *L* と *U* によって上書きされる。*L* の単位対角成分は格納されない。

ipiv INTEGER
配列 DIMENSION は少なくとも max(1, min(*m*, *n*))
ピボット・インデックスである。1 ≤ i ≤ *n* で、マトリックスの行 i は行 *ipiv*(*i*) と交換されている。

info INTEGER。*info* = 0 の場合、正常に終了したことを示す。
info = -*i* の場合、*i* 番目のパラメータの値が不正であったことを示す。
info = *i* > 0 の場合、u_{ii} はゼロである。因子分解は完了したが *U* は完全に特異である。連立 1 次方程式の解の算出に係数 *U* を使用すると 0 による除算が発生する。

?gtts2

?gttrf によって行われた *LU* 因子分解を
使用して、三重対角マトリックスを
係数行列とする連立 1 次方程式を解く。

```
call sgtts2 (itrans, n, nrhs, dl, d, du, du2, ipiv, b, ldb)
call dgtts2 (itrans, n, nrhs, dl, d, du, du2, ipiv, b, ldb)
call cgtts2 (itrans, n, nrhs, dl, d, du, du2, ipiv, b, ldb)
call zgtts2 (itrans, n, nrhs, dl, d, du, du2, ipiv, b, ldb)
```

説明

このルーチンは、以下に示す複数の右辺を持つ以下の連立 1 次方程式のうち 1 つを *X* について解く。

$AX = B$　　$A^TX = B$　または　$A^HX = B$ (複素マトリックスのみ)、
A は三重対角マトリックスで、?gttrf で計算した *LU* 因子分解を使用する。

入力パラメータ

itrans　　INTEGER。ゼロ、1、または 2 のいずれかでなければならない。
方程式の形式を指定する。
itrans = 0 の場合、$AX = B$ (転置なし)
itrans = 1 の場合、$A^TX = B$ (転置)
itrans = 2 の場合、$A^HX = B$ (共役転置)

n　　INTEGER。マトリックス *A* の次数 ($n \geq 0$)。

nrhs　　INTEGER。右辺の個数。すなわち、*B* の列の数 ($nrhs \geq 0$)。

dl, d, du, du2, b　　REAL (sgtts2 の場合)
DOUBLE PRECISION (dgtts2 の場合)
COMPLEX (cgtts2 の場合)
COMPLEX*16 (zgtts2 の場合)
配列：*dl*(*n* - 1)、*d*(*n*)、*du*(*n* - 1)、*du2*(*n* - 2)、*b*(*ldb*, *nrhs*)
配列 *dl* には *A* の *LU* 因子分解で得られたマトリックス *L* を定義する (*n* - 1) 個の乗数を格納する。

配列 *d* には *A* の *LU* 因子分解で得られた上三角マトリックス *U* の *n* 個の対角成分を格納する。
配列 *du* には *U* の最初の優対角成分の (*n* - 1) 個の成分を格納する。
配列 *du2* には *U* の 2 番目の優対角成分の (*n* - 2) 個の成分を格納する。
配列 *b* にはマトリックス *B* を格納する。このマトリックスの列は連立方程式の右辺である。

ldb INTEGER。*b* のリーディング・ディメンジョン。
ldb ≥ max(1, *n*) でなければならない。

ipiv INTEGER
配列 DIMENSION は (*n*)
?gttrf によって返されるピボット・インデックス配列。

出力パラメータ

b 解のマトリックス *X* によって上書きされる。

?labad

指数範囲がきわめて大きい場合に
アンダーフローおよびオーバーフロー
しきい値の平方根を返す。

```
call slabad ( small, large )
call dlabad ( small, large )
```

説明

このルーチンは、アンダーフローとオーバーフローに対して slamch/dlamch によって計算された値を入力として受け取り、*large* のログが充分に大きい場合にそれら値のそれぞれの平方根を返す。このサブルーチンは、Cray のような大きな指数範囲を持つマシンを判別するために使用し、アンダーフローとオーバーフローのリミット値を ?lamch で計算された値の平方根に再定義する。Cray に見られるような上半分の指数範囲における計算能力が弱い場合でも ?lamch は補正を行わないため、このサブルーチンが必要となる。

入力パラメータ

small
REAL (slabad の場合)
DOUBLE PRECISION (dlabad の場合)
?lamch によって計算されたアンダーフローしきい値。

large
REAL (slabad の場合)
DOUBLE PRECISION (dlabad の場合)
?lamch によって計算されたオーバーフローしきい値。

出力パラメータ

small
log10(*large*)が充分に大きい場合は*small*の平方根で上書きされ、そうでない場合は変更されない。

large
log10(*large*)が充分に大きい場合は*large*の平方根で上書きされ、そうでない場合は変更されない。

?labrd

一般マトリックスの最初の nb 行と列を
二重対角形式に縮退させる。

```
call slabrd ( m, n, nb, a, lda, d, e, tauq, taup, x, ldx, y, ldy )
call dlabrd ( m, n, nb, a, lda, d, e, tauq, taup, x, ldx, y, ldy )
call clabrd ( m, n, nb, a, lda, d, e, tauq, taup, x, ldx, y, ldy )
call zlabrd ( m, n, nb, a, lda, d, e, tauq, taup, x, ldx, y, ldy )
```

説明

このルーチンは、直交 / ユニタリ変換 $Q'AP$ によって一般マトリックス A の最初の nb 行と列を上または下二重対角形式に縮退させ、A の縮退されていない部分に変換を適用するために必要なマトリックス X と Y を返す。

$m \geq n$ の場合、A は上二重対角形式に縮退され、$m < n$ の場合、下二重対角形式に縮退される。

マトリックス Q と P は基本リフレクタの積として表現される。

$Q = H(1)\ H(2)\ .\ .\ H(nb)$ および $P = G(1)\ G(2)\ .\ .\ G(nb)$

それぞれの $H(i)$ と $G(i)$ は形式を持っている。持っている。

$H(i) = I - tauq*v*v'$ および $G(i) = I - taup*u*u'$

$tauq$ と $taup$ はスカラー、v と u はベクトルである。

ベクトル v と u の成分は $m \times n$ のマトリックス V と $nb \times n$ のマトリックス U' を形成する。マトリックス A の縮退されていない部分に次の形式のブロック更新を使って変換を適用するために、これらのマトリックスとマトリックス X および Y が必要となる。

$A := A - V*Y' - X*U'$ このルーチンは ?gebrd の補助ルーチンである。

入力パラメータ

m	INTEGER。マトリックス A 内の行数 ($m \geq 0$)。
n	INTEGER。A の列数 ($n \geq 0$)。

nb INTEGER。縮退の対象となる *A* の先頭の行と列の数。

a REAL (slabrd の場合)
DOUBLE PRECISION (dlabrd の場合)
COMPLEX (clabrd の場合)
COMPLEX*16 (zlabrd の場合)
配列 *a(lda, *)* には縮退されるマトリックス *A* を格納する。
a の第 2 次元は、少なくとも max(1, *n*) でなければならない。

lda INTEGER。*a* の第 1 次元。max(1, *m*) 以上であること。

ldx INTEGER。出力配列 *x* の第 1 次元のサイズ。max(1, *m*) 以上でなければならない。

ldy INTEGER。出力配列 *y* の第 1 次元のサイズ。max(1, *n*) 以上でなければならない。

出力パラメータ

a マトリックスの最初の *nb* 行と列は上書きされる。配列の残りの部分は変更されない。

$m \geq n$ の場合、最初の *nb* 列の対角線上とその下の成分は、配列 *tauq* とともに、基本リフレクタの積として直交 / ユニタリ・マトリックス *Q* を表現する。また、最初の *nb* 行の対角よりも上の成分は、配列 *taup* とともに、基本リフレクタの積として直交 / ユニタリ・マトリックス *P* を表現する。

$m < n$ の場合、最初の *nb* 列の対角よりも下の成分は、配列 *tauq* とともに、基本リフレクタの積として直交 / ユニタリ・マトリックス *Q* を表現する。また、最初の *nb* 行の対角線上とその上の成分は、配列 *taup* とともに、基本リフレクタの積として直交 / ユニタリ・マトリックス *P* を表現する。

d, e REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
配列 DIMENSION はそれぞれ、(*nb*)

配列 *d* には縮退されたマトリックスの最初の *nb* 行と列の対角成分が格納される。
d(i) = *a*(i, i)
配列 *e* には縮退されたマトリックスの最初の *nb* 行と列の対角以外の成分が格納される。

tauq, taup
REAL (slabrd の場合)
DOUBLE PRECISION (dlabrd の場合)
COMPLEX (clabrd の場合)
COMPLEX*16 (zlabrd の場合)
配列 DIMENSION はそれぞれ、(*nb*)
直交 / ユニタリ・マトリックス *Q* と *P* をそれぞれ表す基本リフレクタのスカラー係数が格納される。

x, y
REAL (slabrd の場合)
DOUBLE PRECISION (dlabrd の場合)
COMPLEX (clabrd の場合)
COMPLEX*16 (zlabrd の場合)
配列、次元は、*x*(*ldx*, *nb*), *y*(*ldy*, *nb*)
配列 *x* には、*A* の縮退されていない部分を更新するために必要となる *m* × *nb* のマトリックス *X* が格納される。
配列 *y* には、*A* の縮退されていない部分を更新するために必要となる *n* × *nb* のマトリックス *Y* が格納される。

アプリケーション・ノート

m ≥ *n* の場合、基本リフレクタ *H*(i) と *G*(i) に対して、

v(1:i-1) = 0、*v*(i) = 1、*v*(i:*m*) は *a*(i:*m*, i) に格納される。
u(1:i) = 0、*u*(i+1) = 1、*u*(i+1:*n*) は *a*(i, i+1:*n*) に格納される。
tauq は *tauq*(i) に、*taup* は *taup*(i) に格納される。

m < *n* の場合、

v(1:i) = 0、*v*(i+1) = 1、*v*(i+1:*m*) は *a*(i+2:*m*, i) に格納される。
u(1:i-1) = 0、*u*(i) = 1、*u*(i:*n*) は *a*(i, i+1:*n*) に格納される。
tauq は *tauq*(i) に、*taup* は *taup*(i) に格納される。

ルーチン終了後の *a* の内容は、*nb* = 2 において次の例に示される。

$m = 6$、$n = 5$ ($m > n$)

$$\begin{bmatrix} 1 & 1 & u_1 & u_1 & u_1 \\ v_1 & 1 & 1 & u_2 & u_2 \\ v_1 & v_2 & a & a & a \\ v_1 & v_2 & a & a & a \\ v_1 & v_2 & a & a & a \\ v_1 & v_2 & a & a & a \end{bmatrix}$$

$m = 5$、$n = 6$ ($m < n$)

$$\begin{bmatrix} 1 & u_1 & u_1 & u_1 & u_1 & u_1 \\ 1 & 1 & u_2 & u_2 & u_2 & u_2 \\ v_1 & 1 & a & a & a & a \\ v_1 & v_2 & a & a & a & a \\ v_1 & v_2 & a & a & a & a \end{bmatrix}$$

a は元のマトリックスの変更されていない成分を意味し、v_i は $H(i)$ を定義するベクトルの成分を意味し、u_i は $G(i)$ を定義するベクトルの成分を意味する。

?lacon

マトリックス・ベクトル積の評価に
リバース・コミュニケーションを使用して
正方マトリックスの 1- ノルムを推定する。

```
call slacon ( n, v, x, isgn, est, kase )
call dlacon ( n, v, x, isgn, est, kase )
call clacon ( n, v, x, est, kase )
call zlacon ( n, v, x, est, kase )
```

説明

このルーチンは実数 / 複素の正方マトリックス A の 1- ノルムを推定する。マトリックス・ベクトル積の評価にはリバース・コミュニケーションが使用される。

入力パラメータ

n INTEGER。マトリックス A の次数 ($n \geq 1$)。

v, x REAL (slacon の場合)
DOUBLE PRECISION (dlaconb の場合)
COMPLEX (clacon の場合)
COMPLEX*16 (zlacon の場合)
配列 DIMENSION はそれぞれ、(*n*)
v は、ワークスペース配列。
x は中間戻り後は入力として使用される。

isgn INTEGER。ワークスペース配列 DIMENSION (*n*)、実数の場合のみ使用される。

kase INTEGER
?lacon を初めて呼び出すときは *kase* はゼロでなければならない。

出力パラメータ

est REAL (slacon/clacon の場合)
DOUBLE PRECISION (dlacon/zlacon の場合)
ノルム (*A*) の推定 (下限)。

kase 中間戻りでは、*kase* は 1 か 2 となり、*x* が *A* **x* または *A*′**x* によって上書きされるかどうかを示す。?lacon からの最終戻りでは、*kase* は再びゼロになる。

v 最終戻りでは *v* = *A**w、ここで
est = norm(*v*)/norm(*w*) (*w* は返されない)。

x 中間戻りでは、*x* は次の値で上書きされる。
A **x* 、*kase* = 1 の場合、
A′**x*、*kase* = 2 の場合、
(複素では *A*′ は *A* の共役転置)、また、他のパラメータは変更しない状態で ?lacon を再度呼び出さなければならない。

?lacpy

1 つの 2 次元配列の一部または全部を
他にコピーする。

```
call slacpy ( uplo, m, n, a, lda, b, ldb )
call dlacpy ( uplo, m, n, a, lda, b, ldb )
call clacpy ( uplo, m, n, a, lda, b, ldb )
call zlacpy ( uplo, m, n, a, lda, b, ldb )
```

説明

このルーチンは 2 次元マトリックス *A* の一部または全部を他のマトリックス *B* にコピーする。

入力パラメータ

uplo CHARACTER*1

マトリックス *A* のうち、*B* にコピーする部分を指定する。

uplo = 'U' の場合、*A* の上三角部分をコピーする。

uplo = 'L' の場合、*A* の下三角部分をコピーする。

それ以外の場合は、マトリックス *A* の全部がコピーする。

m INTEGER。マトリックス *A* の行数 ($m \geq 0$)。

n INTEGER。*A* の列数 ($n \geq 0$)。

a REAL (slacpy の場合)

DOUBLE PRECISION (dlacpy の場合)

COMPLEX (clacpy の場合)

COMPLEX*16 (zlacpy の場合)

配列 *a*(*lda*, *) には $m \times n$ のマトリックス *A* を格納する。

a の第 2 次元は、少なくとも max(1, *n*) でなければならない。

uplo = 'U' の場合、三角または台形の上側だけがアクセスされる。*uplo* = 'L' の場合、三角または台形の下側だけがアクセスされる。

lda INTEGER。*a* の第 1 次元。*lda* ≥ max(1, *m*)

ldb INTEGER。出力配列 *b* の第 1 次元のサイズ。
ldb ≥ max(1, *m*)

出力パラメータ

b REAL (slacpy の場合)
DOUBLE PRECISION (dlacpy の場合)
COMPLEX (clacpy の場合)
COMPLEX*16 (zlacpy の場合)
配列 *b*(*ldb*, *) には *m* × *n* のマトリックス *B* が格納される。
b の第 2 次元は少なくとも max(1, *n*) でなければならない。
uplo で指定された部分において *B* = *A* で上書きされる。

?ladiv

不必要なオーバーフローを回避して、
実数演算で複素除算を実行する。

```
call sladiv ( a, b, c, d, p, q )
call dladiv ( a, b, c, d, p, q )
res = cladiv ( x, y )
res = zladiv ( x, y )
```

説明

ルーチン sladiv/dladiv は、次式として複素除算を実数演算で実行する。

$$p + iq = \frac{a + ib}{c + id}$$

複素関数 cladiv/zladiv は結果を次式として計算する。

$$res = x/y$$

x と *y* は複素である。*x* / *y* の計算は、結果がオーバーフローしない限り中間過程でオーバーフローは生じない。

入力パラメータ

a, b, c, d	REAL (sladiv の場合) DOUBLE PRECISION (dladiv の場合) 上の式にあるスカラー *a*、*b*、*c*、*d* (実数型のみ)
x, y	COMPLEX (cladiv の場合) COMPLEX*16 (zladiv の場合) 複素スカラー *x* と *y* (複素型のみ)

出力パラメータ

p, q	REAL (sladiv の場合) DOUBLE PRECISION (dladiv の場合) 上の式にあるスカラー *p* と *q* (実数型のみ)
res	COMPLEX (cladiv の場合) DOUBLE COMPLEX (zladiv の場合) *x* / *y* の除算の結果が格納される。

?lae2

2 × 2 対称マトリックスの固有値を
計算する。

```
call slae2 ( a, b, c, rt1, rt2 )
call dlae2 ( a, b, c, rt1, rt2 )
```

説明

ルーチン slae2/dlae2 は 2 × 2 対称マトリックスの固有値を計算する。

$$\begin{bmatrix} a & b \\ b & c \end{bmatrix}$$

出力において、*rt1* は大きい方の絶対値の固有値、*rt2* は小さい方の絶対値の固有値で上書きされる。

入力パラメータ

a, b, c　　REAL (slae2 の場合)
DOUBLE PRECISION (dlae2 の場合)
上記 2 × 2 マトリックスの成分 *a*、*b*、*c*

出力パラメータ

rt1, rt2　　REAL (slae2 の場合)
DOUBLE PRECISION (dlae2 の場合)
大きい方の絶対値、小さい方の絶対値に対してそれぞれ計算された固有値。

アプリケーション・ノート

rt1 はオーバーフロー / アンダーフローが起こらなければ、わずかな ulp (最小位置単位) に対して正確である。*rt2* は行列式 *a**c-b**b* で大幅な桁落ちがある場合に不正確となる可能性がある。*rt2* を精度高く計算するには、あらゆる場合で高精度または適切な丸めまたは適切な切捨て計算が必要になる。

オーバーフローは、*rt1* がオーバーフローに係数 5 を掛けた値の範囲内にあるときに起こり得る。アンダーフローは入力データがゼロか *underflow_threshold*/macheps を超えた場合には影響とはならない。

?laebz

実数対称三重対角マトリックスの
指定された値以下の固有値の個数を
計算し、また ?stebz ルーチンが
必要とする他の処理を実行する。

```
call slaebz( ijob, nitmax, n, mmax, minp, nbmin, abstol, reltol,
             pivmin, d, e, e2, nval, ab, c, mout, nab, work, iwork,
             info )
call dlaebz( ijob, nitmax, n, mmax, minp, nbmin, abstol, reltol,
             pivmin, d, e, e2, nval, ab, c, mout, nab, work, iwork,
             info )
```

説明

ルーチン ?laebz は関数 $N(w)$ を使った反復ループで構成される。関数 $N(w)$ は、対称三重対角マトリックス T から、引数 w 以下の固有値を計数する。ループには 2 つのタイプがある。

ijob = 1、このあと続けて、

ijob = 2: 区間リストを入力として受け取り、元の区間の和集合と同一の固有値の和集合を持つ十分小さい区間のリストを返す。入力区間は (*ab*(j, 1), *ab*(j, 2)], j=1, ..., *minp* である。出力区間 (*ab*(j, 1), *ab*(j, 2)) は固有値 *nab*(j, 1)+1, ..., *nab*(j, 2) を含み、ここで $1 \le j \le$ *mout* である。

ijob = 3: 各入力区間(*ab*(j, 1), *ab*(j, 2))内で$N(w(j))$=*nval*(j)となる$w(j)$点に対してバイナリ・サーチを実行し、サーチでは *c*(j) を開始点として使用する。そのような $w(j)$ が見つかった場合、出力は *ab*(j, 1)=*ab*(j, 2)=w となる。$w(j)$ が見つからなかった場合、出力 (*ab*(j, 1), *ab*(j, 2)] は、初期区間外に点がない限り $N(w)$ が *nval*(j) を飛び越える点を含んだ小区間となる。

前記区間はどのような場合でも半開区間である。すなわち、形式 (*a*,*b*] において *b* は含まれるが *a* は含まれないことに注意する。

アンダーフローを防ぐために、マトリックスを、その最大の成分が *overflow***(1/2) * *underflow***(1/4) よりも絶対値において超えないようにスケールしなければならない。小さい固有値の計算を最も精度高く行うには、マトリックスを上記条件よりもはるかに小さくスケールしてはならない。

注 : 通常、引数の妥当性は確認されない。

入力パラメータ

ijob
INTEGER。処理方法を指定する。
= 1: 初期区間に対して *nab* を計算する。
= 2: *T* の固有値を探すために二分法の反復を実行する。
= 3: *N*(*w*) を反転するため、すなわち、指定した個数の *T* の固有値を左辺に持つ点を探すため、二分法の反復を実行する。
そのほかの値の場合、?laebz は *info* = -1 を返す。

nitmax
INTEGER
実行する二分法の「レベル」の最大数で、すなわち、幅 *W* の区間を 2^(-*nitmax*) * *W* よりも小さくしない。*nitmax* 反復後にすべての区間が収束しなかった場合、*info* には非収束区間数が設定される。

n
INTEGER
三重対角マトリックス *T* の次元 *n*。この値は、少なくとも 1 でなければならない。

mmax
INTEGER
区間の最大数。*mmax* より大きい区間が生成された場合、?laebz は *info* = *mmax*+1 を返して終了する。

minp
INTEGER
区間の初期数。*mmax* よりも大きくてはならない。

nbmin INTEGER
ベクトル・ループを使って処理されるべき、区間の最小値。ゼロの場合、スカラー・ループのみが使用される。

abstol REAL (slaebz の場合)
DOUBLE PRECISION (dlaebz の場合)
区間の最小 (絶対) 幅。区間が *abstol* よりも狭い場合、または (大きさで) 大きい方の終点の *reltol* 倍より狭い場合、区間は充分に小さい、すなわち収束としてみなされる。この値は、少なくとも 0 でなければならない。

reltol REAL (slaebz の場合)
DOUBLE PRECISION (dlaebz の場合)
区間の最小相対幅。区間が *abstol* よりも狭い場合、または (大きさで) 大きい方の終点の *reltol* 倍より狭い場合、区間は充分に小さい、すなわち収束としてみなされる。少なくとも *radix*machine epsilon* でなければならない。

pivmin REAL (slaebz の場合)
DOUBLE PRECISION (dlaebz の場合)
Sturm シーケンス・ループにおける「ピボット」の最小絶対値。この値は少なくとも |e(j)**2| * *safe_min* で、かつ少なくとも *safe_min* でなければならず、ここで *safe_min* は少なくともオーバーフローなしで除算できる最小値である。

d, e, e2 REAL (slaebz の場合)
DOUBLE PRECISION (dlaebz の場合)
配列、次元はそれぞれ、(*n*)
配列 *d* には三重対角マトリックス *T* の対角成分を格納する。
配列 *e* には 1 から *n*-1 の位置にある三重対角マトリックス *T* の対角外成分を格納する。*e*(*n*) は任意である。
配列 *e2* には三重対角マトリックス *T* の対角外成分の正方を格納する。*e2*(*n*) は無視される。

nval INTEGER
配列、次元は (*minp*)
ijob = 1 または 2 の場合は参照されない。
ijob = 3 の場合、$N(w)$ の求める値。

ab REAL (slaebz の場合)
DOUBLE PRECISION (dlaebz の場合)
配列、次元は (*mmax*, 2)
区間の終点。*ab*(j, 1) は j 番目の区間の左側終点 a(j)、*ab*(j, 2) は j 番目の区間の右側終点 b(j)

c REAL (slaebz の場合)
DOUBLE PRECISION (dlaebz の場合)
配列、次元は (*mmax*)
ijob = 1 の場合は無視される。
ijob = 2 の場合、ワークスペース。
ijob = 3 の場合、*c*(j) はバイナリ・サーチにおける最初の検索点に初期化されなければならない。

nab INTEGER
配列、次元は (*mmax*, 2)
ijob = 2の場合は*nab*(i, j)を設定しなければならない。次の条件を満足する必要がある。
N(*ab*(i, 1)) ≤ *nab*(i, 1) ≤ *nab*(i, 2) ≤ N(*ab*(i, 2))、これは区間 i で、固有値 *nab*(i, 1)+1, ..., *nab*(i, 2) のみが考慮されることを意味する。通常、先に *ijob* = 1 で ?laebz を呼び出しておけば、*nab*(i, j) = N(*ab*(i, j)) となる。
ijob = 3 の場合、?laebz を呼び出す前に、*nab* に別の値を通常は設定しなければらならない。

work REAL (slaebz の場合)
DOUBLE PRECISION (dlaebz の場合)
ワークスペース配列、次元は (*mmax*)

iwork INTEGER
ワークスペース配列、次元は (*mmax*)

出力パラメータ

nval
nval の成分は *ab* 内の区間に対応して順序が変更される。そのため、出力の *nval*(j) は一般に入力の *nval*(j) とは同一ではないが、出力の区間 (*ab*(j, 1), *ab*(j, 2)] に対応する。

ab
入力区間は一般に、計算によって、変更、分割、または順序の変更が生じる。

mout
INTEGER
ijob = 1 の場合、区間内の固有値の個数。
ijob = 2 または 3 の場合、区間出力の個数
ijob = 3 の場合、*mout* は *minp* と等しい。

nab
ijob = 1 の場合、*nab*(i, j) は N(*ab*(i, j)) に設定される。
ijob = 2 の場合、*nab*(i, j) には max(*na*(*k*), min(*nb*(*k*), *N*(*ab*(i, j)))) が格納され、*k* は出力区間 (*ab*(j, 1), *ab*(j, 2)] をもたらした入力区間のインデックス、*na*(*k*) と *nb*(*k*) は *nab*(*k*, 1) と *nab*(*k*, 2) の入力値である。
ijob = 3 の場合、*nab*(i, 1) が変更されない (すなわち出力値は入力値と同じ (モジュロの順序変更、*nval* と *ab* 参照)) すべてのサーチ点 w に対して *N*(*w*) > *nval*(i) でない限り、または、*nab*(i, 2) が変更されないすべてのサーチ点 *w* に対して *N*(*w*) < *nval*(i) でない限り、*nab*(i, j) には *N*(*ab*(i, j)) が格納される。

info
INTEGER
0: すべての区間は収束した。
1 ～ *mmax*: 最後の *info* 区間は収束しなかった。
mmax+1: *mmax* を超える区間が生成された。

アプリケーション・ノート

このルーチンは他の LAPACK ルーチンからの呼び出しのみを想定しているため、インターフェイスが使いにくくなっている。目的は次の 2 つである。

(a) 固有値を探す。?laebz は 1 つ以上の初期区間を *ab* に持っていなければならず、また ?laebz を *ijob*=1 で呼び出さなければならない。これは *nab* を設定し、また固有値をカウントする。固有値を持たない区間は通常この時点で捨てられる。また、区間 i のすべての固有値が必要ではない場合、*nab*(i, 1) を増やすか *nab*(i, 2) を減らせる。たとえば、最大固有値を得るために *nab*(i, 1)=*nab*(i, 2)-1 と設定する。次に、*ijob*=2 とし、*mmax* を *ijob*=1 の呼び出しで得た *mout* 値以上にして ?laebz を呼び出す。*ijob*=2 の呼び出し後、固有値 *nab*(i, 1)+1 から *nab*(i, 2) は、*abstol* と *reltol* で指定された許容値で、およそ *ab*(i, 1) (または *ab*(i, 2)) となる。

(b) 固有値 w(f), ..., w(l) を含む区間 (a', b'] を探す。ここでは Gershgorin 区間から開始する。*ab* に 2 個のサーチ区間を設定し、どちらも最初は (a, b) とする。*nval* の 1 つの成分は f-1 を、その他の成分には 1 を格納しなければならない一方で、*c* には a と b がそれぞれ格納されなければならない。求めた区間が (a, b) にない場合をエラーとするため、*nab*(i, 1) は -1 でなければならず、*nab*(i, 2) は *n*+1 でなければならない。そして、*ijob*=3 で ?laebz を呼び出す。出力で、w(f-1) < w(f) ならば、区間の 1 つ j は *ab*(j, 1)=*ab*(j, 2) と *nab*(j, 1)=nab(j, 2)=f-1 を持ち、一方、指定された許容値に対して、w(f-k)=...=w(f+r)、k > 0 および r ≥ 0 ならば、区間は N(*ab*(j, 1))=*nab*(j, 1)=f-k と N(*ab*(j, 2))=*nab*(j, 2)=f+r を持つ。w(l) < w(l+1) と w(l-r)=...=w(l+k) は同様に取り扱われる。

?laed0

?stedc によって使用される。縮退されていない対称三重対角マトリックスの固有値と対応する固有ベクトルを分割統治法を使用してすべて求める。

```
call slaed0(icompq, qsiz, n, d, e, q, ldq, qstore, ldqs, work, iwork,
            info)
call dlaed0(icompq, qsiz, n, d, e, q, ldq, qstore, ldqs, work, iwork,
            info)
call claed0(qsiz, n, d, e, q, ldq, qstore, ldqs, rwork, iwork,
            info)
call zlaed0(qsiz, n, d, e, q, ldq, qstore, ldqs, rwork, iwork,
            info)
```

説明

実数型の本ルーチンは、対称三重対角マトリックスの固有値と (オプションとして) 対応する固有ベクトルを分割統治法を使用してすべて求める。

複素型の claed0/zlaed0 は、密またはバンド・エルミート・マトリックスを縮退して得られる 1 個の対角ブロックである対称三重対角マトリックスのすべての固有値と、密またはバンド・マトリックスの対応する固有ベクトルを求める

入力パラメータ

icompq INTEGER。実数型でのみ使用される。
icompq = 0 の場合、固有値のみ計算する。
icomp = 1 の場合、元の密対称マトリックスの固有ベクトルも計算する。配列 *q* には、元のマトリックスから三重対角形式への縮退に使用した直交マトリックスを格納しなければならない。
icomp = 2 の場合、三重対角マトリックスの固有値と固有ベクトルを計算する。

qsiz INTEGER

フル・マトリックスから三重対角マトリックスへの縮退に使用した直交 / ユニタリ・マトリックスの次元で、*qsiz* ≥ *n* (実数型の場合。*icompq* = 1 の場合は *qsiz* ≥ *n*)

n INTEGER。対称三重対角マトリックスの次元 (*n* ≥ 0)。

d, e, rwork REAL (単精度型)、
DOUBLE PRECISION (倍精度型)
配列：
d(*) には三重対角マトリックスの主対角を格納する。
d(*) の次元は少なくとも max(1, *n*) でなければならない。
e(*) には三重対角マトリックスの対角外の成分を格納する。
e の次元は少なくとも max(1, *n*-1) でなければならない。
rwork(*) は複素型でのみ使用されるワークスペース配列である。*rwork* の次元は、少なくとも (1 +3*n*+2*n*lg(*n*)+3n^2) でなければならず、ここで lg(*n*) = 2^k ≥ *n* となる最小の整数 k

q, qstore REAL (slaed0 の場合)
DOUBLE PRECISION (dlaed0 の場合)
COMPLEX (claed0 の場合)
COMPLEX*16 (zlaed0 の場合)
配列 : *q*(*ldq*, *), *qstore*(*ldqs*, *)。これらの配列の第 2 次元は少なくとも max(1, *n*) でなければならない。
実数型の場合
icompq = 0 の場合、配列 *q* は参照されない。
icompq = 1 の場合、*q* は、現時点で分解の対象となるフル・マトリックスの部分集合に対応する、フル・マトリックスから三重対角形式への縮退に使用する直交マトリックスの列の部分集合を格納する。
icompq = 2 の場合、*q* は単位マトリックスを格納する。配列 *qstore* はワークスペース配列で、*icomp* = 1 の場合のみ参照される。更新マトリックス乗算の実行時に固有ベクトル・マトリックスの部分を格納するために使用される。

複素型の場合。
q には列がユニタリに直交している *qsiz* × *n* マトリックスを格納しなければならない。これは、フル密エルミート・マトリックスを (縮退可能な) 対称三重対角マトリックスに縮退させるユニタリ・マトリックスの一部分である。配列 *qstore* は、更新マトリックス乗算の実行時に固有ベクトル・マトリックスの部分を格納するためにワークスペースとして使用される。

ldq INTEGER。配列 *q* の第 1 次元。*ldq* ≥ max(1, *n*)

ldqs INTEGER。配列 *qstore* の第 1 次元。*ldqs* ≥ max(1, *n*)

work REAL (slaed0 の場合)
DOUBLE PRECISION (dlaed0 の場合)
ワークスペース配列で実数型でのみ使用される。
icompq = 0 または 1 の場合、*work* の次元は少なくとも $(1 + 3n + 2n\lg(n) + 2n^2)$ でなければならず、ここで $\lg(n) = 2^k \geq n$ となる最小の整数 *k* である。
icompq = 2 の場合、*work* の次元は少なくとも $(4n + n^2)$ でなければならない。

iwork INTEGER
ワークスペース配列。
実数型で *icomp* = 0 または 1 の場合、および複素型の場合、*iwork* の次元は少なくとも $(6 + 6n + 5n\lg(n))$ でなければならない。
実数型の場合は、*icompq* = 2 ならば *iwork* の次元は少なくとも $(3 + 5n)$ でなければならない。

出力パラメータ

d 昇順に並べられた固有値によって上書きされる。

e 配列を削除する。

q *icompq* = 2 の場合、*q* は三重対角マトリックスの固有ベクトルで上書きされる。

info INTEGER

info = 0 の場合、正常に終了したことを示す。
info = -*i* の場合、*i* 番目のパラメータの値が不正だったことを示す。
info = *i* > 0 の場合、*i*/(*n*+1) から *mod*(*i*, *n*+1) の行と列からなる部分マトリックスの操作中に、アルゴリズムが固有値の計算に失敗したことを示す。

?laed1

sstedc/dstedc によって使用される。
階数 1 対称マトリックスによる更新後に対角マトリックスの更新された固有連立方程式を計算する。元のマトリックスが三重対角の場合に使用される。

```
call slaed1( n, d, q, ldq, indxq, rho, cutpnt, work, iwork, info)
call dlaed1( n, d, q, ldq, indxq, rho, cutpnt, work, iwork, info)
```

説明

ルーチン ?laed1 は、階数 1 対称マトリックスによる更新後に対角マトリックスの更新された固有連立方程式を計算する。このルーチンは、三重対角マトリックスのすべての固有値と固有ベクトルを必要とする固有問題に対してのみ使用される。?laed7 は、フル対称マトリックスの固有値のみを必要とする場合、または (三重対角形式に縮退された) 固有値と固有ベクトルを必要とする場合を取り扱う。

$$T = Q(\text{in})\ (\ D(\text{in}) + \mathit{rho} * z*z'\ )\ Q'(\text{in}) = Q(\text{out}) * D(\text{out}) * Q'(\text{out})$$

$z = Q'u$ で、u は cutpnt 番目と ($cutpnt$ + 1) 番目の成分が 1 で残りがゼロの長さ n のベクトルである。元のマトリックスの固有ベクトルは Q に格納され、固有値は D に格納される。アルゴリズムは次の 3 過程で構成される。

最初の過程では、固有値が複数ある場合、または z 内にゼロがある場合、問題の大きさの収縮が行われる。そのような場合ごとに、永年方程式問題の次元は 1 だけ縮小される。この過程はルーチン ?laed2 によって実行される。

次の過程では更新された固有値を計算する。これは、ルーチン ?laed4 (?laed3 として呼び出される) を介して永年方程式の根を見つけることによって行われる。このルーチンは現在の問題の固有ベクトルも計算する。

最後の過程では、更新された固有値を直接使用して更新された固有ベクトルを計算する。現在の問題に対する固有ベクトルは、全体問題から得られる固有ベクトルで乗算される。

入力パラメータ

n	INTEGER。対称三重対角マトリックスの次元 ($n \geq 0$)。
d, q, work	REAL (slaed1 の場合) DOUBLE PRECISION (dlaed1 の場合) 配列： *d(*)* には階数 1 摂動マトリックスの固有値を格納する。*d* の次元は少なくとも max(1, *n*) でなければならない。 *q(ldq, *)* には階数 1 摂動マトリックスの固有ベクトルを格納する。*q* の第 2 次元は少なくとも max(1, *n*) でなければならない。 *work(*)* は、ワークスペース配列、次元は ($4n+n^2$) 以上。
ldq	INTEGER。配列 *q* の第 1 次元。$ldq \geq \max(1, n)$
indxq	INTEGER。配列、次元は (n) *d* に入っている 2 つの部分問題を昇順で別々にソートする置換。
rho	REAL (slaed1 の場合) DOUBLE PRECISION (dlaed1 の場合) 階数 1 更新を生成するために使用する劣対角エントリ。
cutpnt	INTEGER 先頭の部分マトリックスにある最後の固有値の位置。$\min(1, n) \leq cutpnt \leq n/2$
iwork	INTEGER。ワークスペース配列、次元は ($4n$)

出力パラメータ

d	修復されたマトリックスの固有値で上書きされる。

q	*q* は修復された三重対角マトリックスの固有ベクトルで上書きされる。
indxq	ソートされた順に部分問題を再統合する置換で上書きされ、すなわち、*d*(*indxq*(i = 1, *n*)) は昇順となる。
info	INTEGER *info* = 0 の場合、正常に終了したことを示す。 *info* = -*i* の場合、*i* 番目のパラメータの値が不正だったことを示す。 *info* = 1 の場合、固有値は収束しなかったことを示す。

?laed2

sstedc/dstedc によって使用される。
固有値を併合し永年方程式を収縮させる。
元のマトリックスが三重対角の場合に
使用される。

```
call slaed2( k, n, n1, d, q, ldq, indxq, rho, z, dlamda, w, q2, indx,
             indxc, indxp, coltyp, info)
call dlaed2( k, n, n1, d, q, ldq, indxq, rho, z, dlamda, w, q2, indx,
             indxc, indxp, coltyp, info)
```

説明

このルーチン ?laed2 は 2 つの固有値のセットをソートされた単一のセットに併合する。続いて、問題の大きさの収縮を試みる。収縮が起こり得るには 2 つの場合がある。2 個以上の固有値が互いに近い場合、または、*z* ベクトルに微小エントリがある場合である。そのような場合ごとに関連する永年方程式問題の次元は 1 だけ縮小される。

入力パラメータ

k	INTEGER。収縮されていない固有値の個数、および関連する永年方程式の次数 ($0 \le k \le n$)。
n	INTEGER。対称三重対角マトリックスの次元 ($n \ge 0$)。
n1	INTEGER。先頭の部分マトリックスにある最後の固有値の位置で、$\min(1, n) \le n1 \le n/2$
d, q, z	REAL (slaed2 の場合) DOUBLE PRECISION (dlaed2 の場合) 配列： *d*(*) には結合対象となる 2 つの部分マトリックスの固有値を格納する。 *d* の次元は少なくとも $\max(1, n)$ でなければならない。

	q(ldq, *) には、(1, 1), (n1, n1) と (n1+1, n1+1), (n, n) の隅にある 2 つの平方ブロック内の 2 個の部分マトリックスの固有ベクトルを格納する。q の第 2 次元は少なくとも max(1, n) でなければならない。 z(*) には更新ベクトルを格納する(最初の部分固有ベクトル・マトリックスの最後の行と、2 番目の部分固有ベクトル・マトリックスの最初の行)。
ldq	INTEGER。配列 q の第 1 次元。ldq ≥ max(1, n)
indxq	INTEGER。配列、次元は (n) d に入っている 2 つの部分問題を昇順で別々にソートする置換。この置換の後半にある成分は、それら値に加算された n1 を最初に持っていなければならない。
rho	REAL (slaed2 の場合) DOUBLE PRECISION (dlaed2 の場合) 今回再結合の対象となっている 2 つの部分マトリックスを、そもそも分割する階数 1 切断に関連した対角外成分。
indx, indxp	INTEGER ワークスペース配列、次元はそれぞれ、(n) 配列 indx には dlamda の内容を昇順にソートするために使用した置換を格納する。 配列 indxp には d の収縮された値を配列の終わりに配置するために使用した置換を格納する。 indxp(1:k) は収縮されていない d 値を指し、 indxp(k+1:n) は収縮された固有値を指す。
coltyp	INTEGER。ワークスペース配列、次元は (n) 実行中に、q2 マトリックス中の次のタイプの列を示すラベルである。 1 : 上半分のみ非ゼロ 2 : 密 3 : 下半分のみ非ゼロ 4 : 収縮

出力パラメータ

d	*d*には後続の(*n*-*k*)個の更新された固有値(収縮された)が昇順で上書きされる。
q	*q* の最後の *n*-*k* 列には、後続の (*n*-*k*) 個の更新された固有値 (収縮された) が上書きされる。
indxq	壊される。
rho	?laed3 が必要とした値に *rho* は変更されている。
dlamda, w, q2	REAL (slaed2 の場合) DOUBLE PRECISION (dlaed2 の場合) 配列 : *dlamda*(*n*), *w*(*n*), *q2*(*n1*^{2}+(*n*-*n1*)^2) 配列 *dlamda* には最初の *k* 個の固有値のコピーが格納され、永年方程式を形成するために ?laed3 で使用される。 配列 *w* には収縮更新された最終的な *z* ベクトルの最初の *k* 個の値が格納され、?laed3 に渡される。 配列 *q2* には最初の *k* 個の固有ベクトルのコピーが格納され、新しい固有ベクトルを解くために ?laed3 のマトリックス乗算 (sgemm/dgemm) で使用される。
indxc	INTEGER。配列、次元は (*n*) 収縮された *q* マトリックスの列を 3 つのグループに整理するために使用された置換。第 1 のグループには *n1* とその上にのみ非ゼロ成分が格納され、第 2 のグループには *n1* より下にのみ非ゼロ成分が格納され、第 3 のグループは密である。
coltyp	*coltyp*(i) は、i = 1 から 4 のみに対するタイプ i の列数で上書きされる (タイプの定義については**入力パラメータ**の *coltyp* の説明を参照)。
info	INTEGER *info* = 0 の場合、正常に終了したことを示す。 *info* = -*i* の場合、*i* 番目のパラメータの値が不正だったことを示す。

?laed3

sstedc/dstedc で使用される。
永年方程式の根を探し固有ベクトルを
更新する。元のマトリックスが三重対角の
場合に使用される。

```
call slaed3( k, n, n1, d, q, ldq, rho, dlamda, q2, indx, ctot, w, s,
             info)
call dlaed3( k, n, n1, d, q, ldq, rho, dlamda, q2, indx, ctot, w, s,
             info)
```

説明

ルーチン ?laed3 は、1 と *k* の範囲に対して、*d*、*w*、*rho* の値によって定義されているとおり、永年方程式の根を探す。?laed4 を適切に呼び出し、ここで解かれる $k \times k$ 連立方程式の固有ベクトルのマトリックスによって結合される固有方程式ペアから固有ベクトルのマトリックスを乗算して、その固有ベクトルを更新する。

このコードは浮動小数点演算に対してきわめて緩い仮定を行う。加減算においてガード桁を持つマシン、または Cray X-MP、Cray Y-MP、Cray C-90、または Cray-2 のような減算のガード桁を持たないバイナリ・マシンで動作する。ガード桁を持たない 16 進マシンや 10 進マシンでは異常終了することが考えられるが、そのような前例はない

入力パラメータ

k	INTEGER。?laed4 で解かれる有利関数の項目数 ($k \geq 0$)。
n	INTEGER。*q* マトリックスの行と列の数。$n \geq k$ (収縮で $n > k$ になる場合がある)。
n1	INTEGER。先頭の部分マトリックス内の最後の固有値の位置で、$\min(1, n) \leq n1 \leq n/2$
q	REAL (slaed3 の場合) DOUBLE PRECISION (dlaed3 の場合) 配列 q(ldq, *)。*q* の第 2 次元は少なくとも max(1, *n*) でなければならない。

初期時には、この配列の最初の *k* 列はワークスペースとして使用される。

ldq INTEGER。配列 *q* の第 1 次元。*ldq* ≥ max(1, *n*)

rho REAL (slaed3 の場合)
DOUBLE PRECISION (dlaed3 の場合)
階数 1 の更新方程式にあるパラメータの値。*rho* ≥ 0 が必要。

dlamda, q2, w REAL (slaed3 の場合)
DOUBLE PRECISION (dlaed3 の場合)
配列 : *dlamda(k)*, *q2(ldq2, *)*, *w(k)*

配列 *dlamda* の最初の *k* 個の成分には収縮された更新問題の古い根を格納する。これらは永年方程式の極である。

配列 *q2* の最初の *k* 列には、分割問題に対する収縮されていない固有ベクトルを格納する。*q2* の第 2 次元は少なくとも max(1, *n*) でなければならない。

配列 *w* の最初の *k* 個の成分には、収縮調整更新ベクトルの成分を格納する。

indx INTEGER。配列、次元は (*n*)
収縮された *q* マトリックスの列を 3 つのグループに整理するために使用された置換 (?laed2 参照)。
?laed4 によって求められた固有ベクトルの行は、マトリックス乗算が行われる前に同様に置換されなければならない。

ctot INTEGER。配列、次元は (4)
indx に記述されている、*q* に含まれる列のタイプのそれぞれの合計。4 番目の列タイプは収縮された任意の列である。

s REAL (slaed3 の場合)
DOUBLE PRECISION (dlaed3 の場合)
ワークスペース配列、次元は (*n1*+1)*k

連立方程式を更新するために過去に蓄積された固有ベクトルで乗算される、修復されたマトリックスの固有ベクトルを格納する。

出力パラメータ

d	REAL (slaed3 の場合) DOUBLE PRECISION (dlaed3 の場合) 配列、次元は少なくとも max(1, *n*) *d*(i) には 1 ≤ i ≤ *k* に対する更新された固有値が格納される。
q	*q* の列 1 から *k* は更新された固有ベクトルで上書きされる。
dlamda	前述のとおり、Cray X-MP、Cray Y-MP、Cray-2、または Cray C-90 では、ゼロに設定された最下位ビットを持っていると変更される場合がある。
w	削除される。
info	INTEGER *info* = 0 の場合、正常に終了したことを示す。 *info* = -*i* の場合、i 番目のパラメータの値が不正だったことを示す。 *info* = 1 の場合、固有値は収束しなかったことを示す。

?laed4

sstedc/dstedc によって使用される。
永年方程式の単一根を見つける。

```
call slaed4 ( n, i, d, z, delta, rho, dlam, info )
call dlaed4 ( n, i, d, z, delta, rho, dlam, info )
```

説明

このサブルーチンは、成分が配列 *d* で与えられている対角マトリックスに対する、対称階数 1 の更新の *i* 番目の更新された固有値を計算し、ここで、

i < j では $D(i) < D(j)$

ここで *rho* > 0。これは呼び出しルーチンによって整頓され、また、一般性が失われることはない。階数 1 の更新された連立方程式は、

diag(D) + *rho* * Z * transpose(Z)

Z のユークリッド・ノルムを 1 と仮定する。

この方法は、単純補間の有利関数による永久方程式内の有利関数の近似を含む。

入力パラメータ

n	INTEGER。全配列の長さ。
i	INTEGER。計算される固有値のインデックスで、$1 \le i \le n$
d, z	REAL (slaed4 の場合) DOUBLE PRECISION (dlaed4 の場合) 配列、次元はそれぞれ、(*n*) 配列 *d* には元の固有値を格納する。それらは、i < j では $d(i) < d(j)$ のように順序どおりに並べられていると仮定される。 配列 *z* には更新ベクトル Z の成分を格納する。

rho REAL (slaed4 の場合)
DOUBLE PRECISION (dlaed4 の場合)
対称更新式におけるスカラー。

出力パラメータ

delta REAL (slaed4 の場合)
DOUBLE PRECISION (dlaed4 の場合)
配列、次元は (*n*)
$n \neq 1$ の場合、*delta* の j 番目の成分に (*d*(j)- *lambda_i*) が格納される。$n = 1$ の場合、*delta*(1) = 1。ベクトル *delta* には固有ベクトルを構成するために必要な情報が格納される。

dlam REAL (slaed4 の場合)
DOUBLE PRECISION (dlaed4 の場合)
計算された *lambda_i*、*i* 番目の更新された固有値。

info INTEGER
info = 0 の場合、正常に終了したことを示す。
info = 1 の場合、更新処理に失敗したことを示す。

?laed5

sstedc/dstedc によって使用される。
2 × 2 の永年方程式を解く。

```
call slaed5 ( i, d, z, delta, rho, dlam )
call dlaed5 ( i, d, z, delta, rho, dlam )
```

説明

このサブルーチンは、2 × 2 対角マトリックスの対称階数 1 の更新の *i* 番目の固有値を計算する。

diag(*D*) + *rho* * *Z* * transpose(*Z*)

配列 *D* の対角成分は次を満たすものと仮定される。

i < j では *D*(i) < *D*(j)

さらに、*rho* > 0、ベクトル Z のユークリッド・ノルムを 1 と仮定する。

入力パラメータ

i INTEGER。計算される固有値のインデックスで $1 \le i \le 2$

d, z REAL (slaed5 の場合)
DOUBLE PRECISION (dlaed5 の場合)
配列、次元はそれぞれ、(2)。
配列 *d* には元の固有値を格納する。*d*(1) < *d*(2) と仮定する。
配列 *z* には更新ベクトルの成分を格納する。

rho REAL (slaed5 の場合)
DOUBLE PRECISION (dlaed5 の場合)
対称更新式におけるスカラー。

出力パラメータ

delta REAL (slaed5 の場合)
DOUBLE PRECISION (dlaed5 の場合)
配列、次元は (2)。
ベクトル *delta* には固有ベクトルを構成するために必要な情報が格納される。

dlam REAL (slaed5 の場合)
DOUBLE PRECISION (dlaed5 の場合)
計算された *lambda_i*、*i* 番目の更新固有値。

?laed6

sstedc/dstedc で使用される。
永久方程式の解の中から Newton ステップの 1 つを計算する。

```
call slaed6(kniter, orgati, rho, d, z, finit, tau, info)
call dlaed6(kniter, orgati, rho, d, z, finit, tau, info)
```

説明

このルーチンは次の式の正または負の根 (元にもっとも近い) を計算する。

$$f(x) = rho + \frac{z(1)}{d(1)-x} + \frac{z(2)}{d(2)-x} + \frac{z(3)}{d(3)-x}$$

orgati = .TRUE. の場合、根は *d*(2) と *d*(3) の間にあると仮定される。それ以外では *d*(1) と *d*(2) の間にあると仮定される。
このルーチンは ?laed4 から必要に応じて呼び出される。多くの場合、根の探索は大きさにおいて最も小さいが、きわめて稀な状況ではないと考えられる。

入力パラメータ

kniter
INTEGER
大きさに関して ?laed4 を参照。

orgati
LOGICAL
orgati = .TRUE. の場合、必要となる根は *d*(2) と *d*(3) の間にあり、それ以外では *d*(1) と *d*(2) の間にある。詳細は ?laed4 を参照。

rho
REAL (slaed6 の場合)
DOUBLE PRECISION (dlaed6 の場合)
上記の *f(x)* の式を参照。

d, z
REAL (slaed6 の場合)
DOUBLE PRECISION (dlaed6 の場合)
配列、次元はそれぞれ、(3)
d(1) < *d*(2) < *d*(3) を満足する配列 *d*
配列 *z* の各成分は正でなければならない。

finit
REAL (slaed6 の場合)
DOUBLE PRECISION (dlaed6 の場合)
ゼロにおける *f(x)* の値。このルーチン内部を評価した値よりも正確であること (そのようなことを望む場合)。

出力パラメータ

tau REAL (slaed6 の場合)
DOUBLE PRECISION (dlaed6 の場合)
f(x) に対する式の根。

info INTEGER
info = 0 の場合、正常に終了したことを示す。
info = 1 の場合、収束に失敗したことを示す。

?laed7

?stedc によって使用される。階数 1 対称マトリックスによる更新後に対角マトリックスの更新された固有連立方程式を計算する。元のマトリックスが密の場合に使用される。

```
call slaed7( icompq, n, qsiz, tlvls, curlvl, curpbm, d, q, ldq, indxq,
             rho, cutpnt, qstore, qptr, prmptr, perm, givptr, givcol,
             ivnum, work, iwork, info )
call dlaed7( icompq, n, qsiz, tlvls, curlvl, curpbm, d, q, ldq, indxq,
             rho, cutpnt, qstore, qptr, prmptr, perm, givptr, givcol,
             givnum, work, iwork, info )
call claed7( n, cutpnt, qsiz, tlvls, curlvl, curpbm, d, q, ldq, rho,
             indxq, qstore, qptr, prmptr, perm, givptr, givcol,
             givnum, work, rwork, iwork, info )
call zlaed7( n, cutpnt, qsiz, tlvls, curlvl, curpbm, d, q, ldq, rho,
             indxq, qstore, qptr, prmptr, perm, givptr, givcol,
             givnum, work, rwork, iwork, info )
```

説明

ルーチン ?laed7 は、階数 1 対称マトリックスによる更新後に対角マトリックスの更新された固有連立方程式を計算する。このルーチンは、三重対角形式に縮退された密対称 / エルミート・マトリックスのすべての固有値と、オプションで固有ベクトルを必要とする固有問題に対してのみ使用される。実数型では、slaed1/dlaed1 は、対称三重対角マトリックスのすべての固有値と固有ベクトルが必要な場合を取り扱う。

$$T = Q(\text{in})\ (\ D(\text{in}) + \mathit{rho} * z*z'\ )\ Q'(\text{in}) = Q(\text{out}) * D(\text{out}) * Q'(\text{out})$$

$z = Q'u$ で、u は *cutpnt* 番目と (*cutpnt* + 1) 番目の成分が 1 で残りがゼロの長さ *n* のベクトルである。元のマトリックスの固有ベクトルは Q に格納され、固有値は D に格納される。アルゴリズムは次の 3 過程で構成される。

最初の過程では、固有値が複数ある場合、または z ベクトル内にゼロがある場合、問題の大きさの収縮が行われる。そのような場合ごとに、永年方程式問題の次元は 1 だけ縮小される。この過程はルーチン slaed8/dlaed8 (実数型)、またはルーチン slaed2/dlaed2 (複素型) によって実行される。

次の過程では更新された固有値を計算する。これは、ルーチン ?laed4 (?laed3 または ?laed9 として呼び出される) を介して永年方程式の根を見つけることによって行われる。このルーチンは現在の問題の固有ベクトルも計算する。

最後の過程では、更新された固有値を直接使用して更新された固有ベクトルを計算する。現在の問題に対する固有ベクトルは、全体問題から得られる固有ベクトルで乗算される。

入力パラメータ

icompq INTEGER。実数型でのみ使用される。
icompq = 0 の場合、固有値のみ計算する。
icompq = 1 の場合、元の密対称マトリックスの固有ベクトルも計算する。配列 *q* には、元のマトリックスから三重対角形式への縮退に使用した直交マトリックスを格納しなければならない。

n INTEGER。対称三重対角マトリックスの次元 ($n \geq 0$)。

cutpnt INTEGER。先頭の部分マトリックスにある最後の固有値の位置。$\min(1, n) \leq cutpnt \leq n$

qsiz INTEGER。フル・マトリックスから三重対角マトリックスへの縮退に使用した直交 / ユニタリ・マトリックスの次元で、$qsiz \geq n$ (実数型の場合。*icompq* = 1 の場合は $qsiz \geq n$)

tlvls INTEGER。分割統治ツリー全体の併合レベルの合計数。

curlvl INTEGER。併合ルーチン全体の現在のレベルで、$0 \leq curlvl \leq tlvls$

curpbm INTEGER。併合ルーチン全体の現在のレベルにおける現在の問題 (左上から右下に向かって計数)。

d
REAL (slaed7/claed7 の場合)
DOUBLE PRECISION (dlaed7/zlaed7 の場合)
配列、次元は少なくとも max(1, *n*)
配列 *d*(*) には階数 1 置換マトリックスの固有値を格納する。

q, work
REAL (slaed7 の場合)
DOUBLE PRECISION (dlaed7 の場合)
COMPLEX (claed7 の場合)
COMPLEX*16 (zlaed7 の場合)
配列：
q(*ldq*, *) には階数1置換マトリックスの固有ベクトルを格納する。*q* の第 2 次元は少なくとも max(1, *n*) でなければならない。
work(*) はワークスペース配列、次元は、実数の場合は (3*n*+*qsiz***n*) 以上、複素数の場合は (*qsiz***n*) 以上。

ldq
INTEGER。配列 *q* の第 1 次元。*ldq* ≥ max(1, *n*)

rho
REAL (slaed7/claed7 の場合)
DOUBLE PRECISION (dlaed7/zlaed7 の場合)
階数 1 更新を生成するために使用された部分対角成分。

qstore
REAL (slaed7/claed7 の場合)
DOUBLE PRECISION (dlaed7/zlaed7 の場合)
配列、次元は (n^2+1)。出力パラメータとしても働く。分割統治中に遭遇した部分マトリックスの固有ベクトルをパックして格納する。*qptr* は部分マトリックスの始まりを指す。

qptr
INTEGER。配列、次元は (*n*+2)。出力パラメータとしても働く。
qstore に格納されている部分マトリックスの始まりを指すインデックス・リスト。部分マトリックスは、分割統治ツリーの左下を先頭にして、左から右へ、下から上へ番号が振られる。

prmptr, perm, givptr

INTEGER。配列、次元はそれぞれ、(n lg$n$)

配列 *prmptr*(*) には、あるレベルの置換が *perm* 内のどこに格納されているかを示すポインタ・リストを格納する。*prmptr*(i+1) - *prmptr*(i) は置換のサイズを表し、また収縮されていない全問題の大きさも表す。

配列 *perm*(*) には各固有ブロックに適用される (収縮とソートによる) 置換を格納する。

配列 *givptr*(*) には、あるレベルの *Givens* 回転が *givcol* 内のどこに格納されているかを示すポインタ・リストを格納する。*givptr*(i+1) - *givptr*(i) は Givens 回転の回数を示す。

givcol

INTEGER。配列、次元は (2, n lg$n$)

数のそれぞれのペアは Givens 回転を生じる列ペアを示す。

givnum

REAL (slaed7/claed7 の場合)

DOUBLE PRECISION (dlaed7/zlaed7 の場合)

配列、次元は (2, n lg$n$)

それぞれの数は対応する Givens 回転で使用される S 値を示す。

iwork

INTEGER。ワークスペース配列、次元は (4n)

rwork

REAL (claed7 の場合)

DOUBLE PRECISION (zlaed7 の場合)

ワークスペース配列、次元は (3n+2*qsiz**n)。複素型でのみ使用される。

出力パラメータ

d

修復されたマトリックスの固有値で上書きされる。

q

q は修復された三重対角マトリックスの固有ベクトルで上書きされる。

indxq INTEGER。配列、次元は (*n*)
ソートされた順に部分問題を再統合する置換で上書きされ、すなわち、*d*(*indxq*(i = 1, *n*)) は昇順となる。

info INTEGER
info = 0 の場合、正常に終了したことを示す。
info = -*i* の場合、*i* 番目のパラメータの値が不正だったことを示す。
info = 1 の場合、固有値は収束しなかったことを示す。

?laed8

?stedc で使用される。固有値を併合し
永年方程式を収縮する。元の
マトリックスが密のときに使用される。

```
call slaed8( icompq, k, n, qsiz, d, q, ldq, indxq, rho, cutpnt, z,
             lamda, q2, ldq2, w, perm, givptr, givcol, givnum, indxp,
             indx, info )
call dlaed8( icompq, k, n, qsiz, d, q, ldq, indxq, rho, cutpnt, z,
             dlamda, q2, ldq2, w, perm, givptr, givcol, givnum,
             indxp, indx, info )
call claed8( k, n, qsiz, q, ldq, d, rho, cutpnt, z, dlamda, q2, ldq2,
             w, indxp, indx, indxq, perm, givptr, givcol, givnum,
             info )
call zlaed8( k, n, qsiz, q, ldq, d, rho, cutpnt, z, dlamda, q2, ldq2,
             w, indxp, indx, indxq, perm, givptr, givcol, givnum,
             info )
```

説明

このルーチンは 2 つの固有値のセットをソートされた単一のセットに併合する。続いて、問題の大きさの収縮を試みる。収縮が起こり得るには 2 つの場合がある。2 個以上の固有値が互いに近い場合、または、z ベクトルに微小エントリがある場合である。そのような場合ごとに、関連する永年方程式問題の次元は 1 だけ縮小される。

入力パラメータ

icompq INTEGER。実数型でのみ使用される。

icompq = 0 の場合、固有値のみ計算する。
icompq = 1 の場合、元の密対称マトリックスの固有ベクトルも計算される。配列 *q* には、元のマトリックスから三重対角形式への縮退に使用した直交マトリックスを格納しなければならない。

n INTEGER。対称三重対角マトリックスの次元 ($n \geq 0$)。

cutpnt	INTEGER。先頭の部分マトリックスにある最後の固有値の位置。min(1, *n*) ≤ *cutpnt* ≤ *n*
qsiz	INTEGER。フル・マトリックスから三重対角マトリックスへの縮退に使用した直交 / ユニタリ・マトリックスの次元で、*qsiz* ≥ *n* (実数型の場合。*icompq* = 1 の場合は *qsiz* ≥ *n*)
d, z	REAL (slaed8/claed8 の場合) DOUBLE PRECISION (dlaed8/zlaed8 の場合) 配列、次元はそれぞれ、max(1, *n*) 以上。 配列 *d*(*) には結合する 2 つの部分マトリックスの固有値を格納する。 *z*(*) には更新ベクトルを格納する (最初の部分固有ベクトル・マトリックスの最後の行と、2 番目の部分固有ベクトル・マトリックスの最初の行)。*z* の内容は更新過程で壊される。
q	REAL (slaed8 の場合) DOUBLE PRECISION (dlaed8 の場合) COMPLEX (claed8 の場合) COMPLEX*16 (zlaed8 の場合) 配列 *q(ldq, *)*。*q* の第 2 次元は少なくとも max(1, *n*) でなければならない。*q* には、他の部分的に解かれた固有連立方程式とのマトリックス乗算によって、すでに更新された部分的に解かれた連立方程式の固有ベクトルを格納する。 実数型の場合は、*icompq* = 0 ならば *q* は参照されない。
ldq	INTEGER。配列 *q* の第 1 次元。*ldq* ≥ max(1, *n*)
ldq2	INTEGER。出力配列 *q2* の第 1 次元のサイズ。*ldq2* ≥ max(1, *n*)
indxq	INTEGER。配列、次元は (*n*) *d* に入っている 2 つの部分問題を昇順で別々にソートする置換。この置換の後半にある成分は、精度を得るために、それら値に加算された *cutpnt* を最初に持っていなければならない。

rho REAL (slaed8/claed8 の場合)
DOUBLE PRECISION (dlaed8/zlaed8 の場合)
今回再結合の対象となっている 2 つの部分マトリックスを、そもそも分割する階数 1 切断に関連した対角外成分。

出力パラメータ

k INTEGER。非収縮の固有値の数、関連永年方程式の次数。

d 後続の (*n*-*k*) 個の更新された固有値 (収縮された) が昇順で上書きされる。

q *q* の最後の *n*-*k* 列には、後続の (*n*-*k*) 個の更新された固有値 (収縮された) が上書きされる。

rho ?laed3 が必要とした値に *rho* は変更されている。

dlamda, w REAL (slaed8/claed8 の場合)
DOUBLE PRECISION (dlaed8/zlaed8 の場合)
配列、次元はそれぞれ、(*n*)
配列 *dlamda*(*) には最初の *k* 個の固有値のコピーが格納され、永年方程式を形成するために ?laed3 で使用される。
配列 *w*(*) は収縮更新された最終的な z ベクトルの最初の *k* 個の値を保持し、?laed3 に渡される。

q2 REAL (slaed8 の場合)
DOUBLE PRECISION (dlaed8 の場合)
COMPLEX (claed8 の場合)
COMPLEX*16 (zlaed8 の場合)
配列 *q2*(*ldq2, *)。*q2* の第 2 次元は少なくとも max(1, *n*) でなければならない。
新しい固有値の更新のために、slaed7/dlaed7 のマトリックス乗算 (sgemm/dgemm) で使用される最初の *k* 個の固有ベクトルのコピーが格納される。
実数型の場合は、*icompq* = 0 ならば *q2* は参照されない。

indxp, indx INTEGER。ワークスペース配列、次元はそれぞれ、(*n*)

配列 *indxp*(*) には、*d* の収縮した値を配列の終わりに配置するために使用した置換を格納する。
indxp(1:*k*) は収縮されていない *d* 値を指し、
indxp(*k*+1:*n*) は収縮された固有値を指す。
配列 *indx*(*) には *d* の内容を昇順にソートするために使用した置換を格納する。

perm INTEGER。配列、次元は (*n*)
各固有ブロックに適用される (収縮とソートによる) 置換が格納される。

givptr INTEGER。この部分問題で実行された Givens 回転の回数が格納される。

givcol INTEGER。配列、次元は (2, *n*)
数のそれぞれのペアは Givens 回転を生じる列ペア。

givnum REAL (slaed8/claed8 の場合)
DOUBLE PRECISION (dlaed8/zlaed8 の場合)
配列、次元は (2, *n*)
それぞれの数は対応する Givens 回転で使用される *S* 値を示す。

info INTEGER
info = 0 の場合、正常に終了したことを示す。
info = -*i* の場合、*i* 番目のパラメータの値が不正だったことを示す。

?laed9

sstedc/dstedc で使用される。
永年方程式の根を探し固有ベクトルを
更新する。元のマトリックスが密の場合に
使用される。

```
call slaed9( k, kstart, kstop, n, d, q, ldq, rho, dlamda, w, s, lds,
             info )
call dlaed9( k, kstart, kstop, n, d, q, ldq, rho, dlamda, w, s, lds,
             info )
```

説明

ルーチン ?laed3 は、*kstart* と *kstop* の範囲で、*d*、*w*、*rho* の値によって定義されているとおり、永年方程式の根を探す。slaed4/dlaed4 への適切な呼び出しを行い、続いて、z ベクトルの次のレベルの計算で用いる固有ベクトルの新しいマトリックスを格納する。

入力パラメータ

k	INTEGER。slaed4/dlaed4 で解かれる有利関数の項目数 ($k \geq 0$)。
kstart, kstop	INTEGER。更新された固有値 *lambda*(i)、*kstart* ≤ i ≤ *kstop* で計算される。 $1 \leq kstart \leq kstop \leq k$
n	INTEGER。*q* マトリックスの行と列の数。$n \geq k$ (収縮で $n > k$ になる場合がある)。
q	REAL (slaed9 の場合) DOUBLE PRECISION (dlaed9 の場合) ワークスペース配列、次元は (*ldq*, *)。*q* の第 2 次元は少なくとも max(1, *n*) でなければならない。
ldq	INTEGER。配列 *q* の第 1 次元。$ldq \geq \max(1, n)$

rho
REAL (slaed9 の場合)
DOUBLE PRECISION (dlaed9 の場合)
階数 1 更新方程式のパラメータの値。*rho* ≥ 0 が必要。

dlamda, w
REAL (slaed9 の場合)
DOUBLE PRECISION (dlaed9 の場合)
配列、次元はそれぞれ、(*k*)
配列 *dlamda*(*) の最初の *k* 個の成分には収縮された更新問題の古い根を格納する。これらは永年方程式の極である。
配列 *w*(*) の最初の *k* 個の成分には、収縮調整更新ベクトルの成分を格納する。

lds
INTEGER。出力配列 *s* の第 1 次元のサイズ。
lds ≥ max(1, *k*)

出力パラメータ

d
REAL (slaed9 の場合)
DOUBLE PRECISION (dlaed9 の場合)
配列、次元は (*n*)。*d*(i) には *kstart* ≤ i ≤ *kstop* に対する更新された固有値が格納される。

s
REAL (slaed9 の場合)
DOUBLE PRECISION (dlaed9 の場合)
配列、次元は (*lds*, *)。*s* の第 2 次元は少なくとも max(1, *k*) でなければならない。
続く z ベクトル計算のために格納され、また、連立方程式の更新のために過去に蓄積された固有ベクトルで乗算される、修復されたマトリックスの固有ベクトルが格納される。

info
INTEGER
info = 0 の場合、正常に終了したことを示す。
info = -*i* の場合、*i* 番目のパラメータの値が不正だったことを示す。
info = 1 の場合、固有値は収束しなかったことを示す。

?laeda

?stedc で使用される。対角
マトリックスの階数 1 更新を決定する
Z ベクトルを計算する。元の
マトリックスが密のときに使用される。

```
call slaeda( n, tlvls, curlvl, curpbm, prmptr, perm, givptr, givcol,
             givnum, q, qptr, z, ztemp, info )
call dlaeda( n, tlvls, curlvl, curpbm, prmptr, perm, givptr, givcol,
             givnum, q, qptr, z, ztemp, info )
```

説明

?laeda ルーチンは、*curpbm* 番目問題に対する *tlvls* ステップを用いて、併合過程の *culvl* 番目のステップ中の併合ステップに対応した z ベクトルを計算する。

入力パラメータ

n INTEGER。対称三重対角マトリックスの次元 ($n \geq 0$)。

tlvls INTEGER。分割統治ツリー全体の併合レベルの合計数。

curlvl INTEGER。併合ルーチン全体の現在のレベルで、$0 \leq curlvl \leq tlvls$

curpbm INTEGER。併合ルーチン全体の現在のレベルにおける現在の問題 (左上から右下に向かって計数)。

prmptr, perm, givptr
INTEGER。配列、次元はそれぞれ、($n \lg n$)
配列 *prmptr*(*) には、あるレベルの置換が *perm* 内のどこに格納されているかを示すポインタ・リストを格納する。*prmptr*(i+1) - *prmptr*(i) は置換のサイズを表し、また収縮されていない全問題の大きさも表す。

配列 *perm*(*) には各固有ブロックに適用される (収縮とソートによる) 置換を格納する。

配列 *givptr*(*) には、あるレベルの *Givens* 回転が *givcol* 内のどこに格納されているかを示すポインタ・リストを格納する。*givptr*(i+1) - *givptr*(i) は Givens 回転の回数を示す。

givcol INTEGER。配列、次元は (2, $n \lg n$)
数のそれぞれのペアは Givens 回転を生じる列ペアを示す。

givnum REAL (slaeda の場合)
DOUBLE PRECISION (dlaeda の場合)
配列、次元は (2, $n \lg n$)
それぞれの数は対応する Givens 回転で使用される *S* 値を示す。

q REAL (slaeda の場合)
DOUBLE PRECISION (dlaeda の場合)
配列、次元は (n^2)
以前のレベルから得られる平方固有ブロックを格納する。ブロックの開始位置は *qptr* で与えられる。

qptr INTEGER。配列、次元は (n+2)。固有ブロックが *q* のどこに格納されているかを示すポインタのリストを格納する。sqrt(*qptr*(i+1) - *qptr*(i)) はブロックの大きさを示す。

ztemp REAL (slaeda の場合)
DOUBLE PRECISION (dlaeda の場合)
ワークスペース配列、次元は (n)

出力パラメータ

z REAL (slaeda の場合)
DOUBLE PRECISION (dlaeda の場合)
配列、次元は (n)。更新ベクトルが格納される (最初の部分固有ベクトル・マトリックスの最後の行と、2 番目の部分固有ベクトル・マトリックスの最初の行)。

info INTEGER

info = 0 の場合、正常に終了したことを示す。
info = -*i* の場合、*i* 番目のパラメータの値が不正だったことを示す。

?laein

上 Hessenberg マトリックスの指定された右または左固有ベクトルを逆反復法を用いて計算する。

```
call slaein( rightv, noinit, n, h, ldh, wr, wi, vr, vi, b, ldb, work,
             eps3, smlnum, bignum, info )
call dlaein( rightv, noinit, n, h, ldh, wr, wi, vr, vi, b, ldb, work,
             eps3, smlnum, bignum, info )
call claein( rightv, noinit, n, h, ldh, w, v, b, ldb, rwork, eps3,
             smlnum, info )
call zlaein( rightv, noinit, n, h, ldh, w, v, b, ldb, rwork, eps3,
             smlnum, info )
```

説明

?laein ルーチンは、実数の上 Hessenberg マトリックス H (実数型 laein/dlaein) の固有値 (*wr*, *wi*)、または複素の上 Hessenberg マトリックス H (複素型 claein/zlaein) の固有値 *w* に対応した右または左固有ベクトルを逆反復法を使用して探す。

入力パラメータ

rightv LOGICAL。
rightv = .TRUE. の場合、右固有ベクトルを計算する。
rightv = .FALSE. の場合、左固有ベクトルを計算する。

noinit LOGICAL。
noinit = .TRUE. の場合、(*vr*, *vi*) または *v* (複素型) において初期ベクトルは与えられない。
noinit = .FALSE. の場合、(*vr*, *vi*) または *v* (複素型) において初期ベクトルが与えられる。

n INTEGER。マトリックス H の次数 ($n \geq 0$)。

h
REAL (slaein の場合)
DOUBLE PRECISION (dlaein の場合)
COMPLEX (claein の場合)
COMPLEX*16 (zlaein の場合)
配列 *h*(*ldh*, *)。*h* の第 2 次元は少なくとも max(1, *n*) でなければならない。Hessenberg 上マトリックス *H* が格納される。

ldh
INTEGER。配列 *h* の第 1 次元。*ldh* ≥ max(1, *n*)

wr, wi
REAL (slaein の場合)
DOUBLE PRECISION (dlaein の場合)
その対応する右または左の固有ベクトルが、計算対象となっている *H* の固有値の実数部分および虚数部分 (実数型ルーチン)。

w
COMPLEX (claein の場合)
COMPLEX*16 (zlaein の場合)
その対応する右または左の固有ベクトルが計算対象となっている *H* の固有値 (複素型ルーチン)。

vr, vi
REAL (slaein の場合)
DOUBLE PRECISION (dlaein の場合)
配列、次元はそれぞれ、(*n*)。実数型でのみ使用される。
noinit = .FALSE. かつ *wi* = 0.0 の場合、実数固有値 *wr* を使った逆反復のために、*vr* には実数開始ベクトルを格納しなければならない。
noinit = .FALSE. かつ *wi* ≠ 0.0 の場合、複素固有値 (*wr*, *wi*) を使った逆反復のために、*vr* と *vi* には複素開始ベクトルの実数と虚数部分を格納しなければならない。それ以外の場合は *vr* と *vi* を設定する必要はない。

v
COMPLEX (claein の場合)
COMPLEX*16 (zlaein の場合)
配列、次元は (*n*)。複素型でのみ使用される。
noinit = .FALSE. 場合、*v* には逆反復のために開始ベクトルを格納しなければならない。それ以外の場合は *v* を設定する必要はない。

b
REAL (slaein の場合)
DOUBLE PRECISION (dlaein の場合)
COMPLEX (claein の場合)
COMPLEX*16 (zlaein の場合)
ワークスペース配列 *b*(*ldb*, *)。*b* の第 2 次元は少なくとも max(1, *n*) でなければならない。

ldb
INTEGER。配列 *b* の第 1 次元。
実数型は *ldb* ≥ *n*+1
複素型は *ldb* ≥ max(1, *n*)

work
REAL (slaein の場合)
DOUBLE PRECISION (dlaein の場合)
ワークスペース配列、次元は (*n*)。実数型でのみ使用される。

rwork
REAL (claein の場合)
DOUBLE PRECISION (zlaein の場合)
ワークスペース配列、次元は (*n*)。複素型でのみ使用される。

eps3, smlnum
REAL (slaein/claein の場合)
DOUBLE PRECISION (dlaein/zlaein の場合)
eps3 は小さなマシン依存値で、値が近い固有値を摂動するため、およびゼロ・ピボットを交換するために使われる。*smlnum* はマシン依存値でアンダーフローしきい値に近い。

bignum
REAL (slaein の場合)
DOUBLE PRECISION (dlaein の場合)
bignum はマシン依存値でオーバーフローしきい値に近い。実数型でのみ使用される。

出力パラメータ

vr, vi
wi = 0.0(実数固有値) の場合、*vr* は計算された実数固有ベクトルで上書きされ、*wi* ≠ 0.0 (複素固有値) の場合、*vr* と *vi* は計算された複素固有ベクトルの実数部と虚数部で上書きされる。固有ベクトルは最大の大きさの成分が大きさ 1 を持つように正規化される。ここで、複素値 (*x*, *y*) の大きさは |*x*| + |*y*| として求められる。*wi* = 0.0 の場合、*vi* は参照されない。

v	*v* は、最大の大きさの成分が大きさ 1 を持つように正規化された、計算された固有ベクトルで上書きされる。ここで、複素値 (x, y) の大きさは $\|x\| + \|y\|$ として求められる。
info	INTEGER *info* = 0 の場合、正常に終了したことを示す。 *info* = 1 の場合、逆反復は収束しなかったことを示す。実数型の場合、*vr* は最後の反復が設定され、*vi* も *wi* ≠ 0.0 の場合に同じである。複素型の場合、*v* には最後の反復が設定される。

?laev2

2 × 2 の対称 / エルミート・マトリックスの
固有値と固有ベクトルを計算する。

```
call slaev2 (a, b, c, rt1, rt2, cs1, sn1)
call dlaev2 (a, b, c, rt1, rt2, cs1, sn1)
call claev2 (a, b, c, rt1, rt2, cs1, sn1)
call zlaev2 (a, b, c, rt1, rt2, cs1, sn1)
```

説明

このルーチンは、対称マトリックス、

$\begin{bmatrix} a & b \\ b & c \end{bmatrix}$ (slaev2/dlaev2 の場合) または

エルミート・マトリックス $\begin{bmatrix} a & b \\ \mathrm{conjg}(b) & c \end{bmatrix}$ (claev2/zlaev2 の場合)

に対して固有分解を実行する。

rt1 は大きい方の絶対値の固有値で上書きされ、*rt2* は小さい方の絶対値の固有値で上書きされ、(*cs1*, *sn1*) は *rt1* に対する単位右固有ベクトルで次の分解を与える。

$$\begin{bmatrix} cs1 & sn1 \\ -sn1 & cs1 \end{bmatrix} \cdot \begin{bmatrix} a & b \\ b & c \end{bmatrix} \cdot \begin{bmatrix} cs1 & -sn1 \\ sn1 & cs1 \end{bmatrix} = \begin{bmatrix} rt1 & 0 \\ 0 & rt2 \end{bmatrix}$$

(slaev2/dlaev2 の場合)、
または

$$\begin{bmatrix} cs1 & \mathrm{conjg}(sn1) \\ -sn1 & cs1 \end{bmatrix} \cdot \begin{bmatrix} a & b \\ \mathrm{conjg}(b) & c \end{bmatrix} \cdot \begin{bmatrix} cs1 & -\mathrm{conjg}(sn1) \\ sn1 & cs1 \end{bmatrix} = \begin{bmatrix} rt1 & 0 \\ 0 & rt2 \end{bmatrix}$$

(claev2/zlaev2 の場合)

入力パラメータ

a, b, c
: REAL (slaev2 の場合)
DOUBLE PRECISION (dlaev2 の場合)
COMPLEX (claev2 の場合)
COMPLEX*16 (zlaev2 の場合)
入力マトリックスの成分。

出力パラメータ

rt1, rt2
: REAL (slaev2/claev2 の場合)
DOUBLE PRECISION (dlaev2/zlaev2 の場合)
それぞれ、大きい方および小さい方の値の固有値。

cs1
: REAL (slaev2/claev2 の場合)
DOUBLE PRECISION (dlaev2/zlaev2 の場合)

sn1
: REAL (slaev2 の場合)
DOUBLE PRECISION (dlaev2 の場合)
COMPLEX (claev2 の場合)
COMPLEX*16 (zlaev2 の場合)
ベクトル (*cs1*, *sn1*) は *rt1* に対する単位右固有ベクトル。

アプリケーション・ノート

rt1 はオーバーフロー / アンダーフローが起こらなければ、わずかな ulp (最小位置単位) に対して正確である。*rt2* は行列式 *a***c*-*b***b* で、大幅な桁落ちがある場合に不正確となる可能性がある。*rt2* を精度高く計算するには、あらゆる場合で高精度または適切な丸めまたは適切な切捨て計算が必要になる。*cs1* と *sn1* はオーバーフロー / アンダーフローが起こらなければわずかな *ulp* に対して正確である。オーバーフローは、*rt1* がオーバーフローに係数 5 を掛けた値の範囲内にあるときに起こる可能性がある。アンダーフローは入力データがゼロか *underflow_threshold* / macheps を超えた場合には影響とはならない。

?laexc

Schur 標準形となっている実数の上準三角
マトリックスの隣接対角ブロックを、
直交相似変換によって交換する。

```
call slaexc ( wantq, n, t, ldt, q, ldq, j1, n1, n2, work, info )
call dlaexc ( wantq, n, t, ldt, q, ldq, j1, n1, n2, work, info )
```

説明

このルーチンは、上準三角マトリックス T にある次数 1 または 2 の隣接対角ブロック T_{11} と T_{22} を、直交相似変換によって交換する。
T は Schur 標準形でなければならず、すなわち、1×1 と 2×2 の対角ブロックを持つブロック上三角である。それぞれの 2×2 対角ブロックは、対角成分が等しく、かつ対角外成分が逆の符号を持つ。

入力パラメータ

wantq
: LOGICAL。
wantq = .TRUE. の場合、マトリックス Q 内の変換を蓄積する。
wantq = .FALSE. の場合、変換を蓄積しない。

n
: INTEGER。マトリックス T の次数 ($n \geq 0$)。

t, q
: REAL (slaexc の場合)
DOUBLE PRECISION (dlaexc の場合)
配列：
*t(ldt, *)* には、上準三角マトリックス T を Shur 標準形で格納する。
t の第 2 次元は少なくとも max(1, *n*) でなければならない。

wantq = .TRUE. の場合、q(ldq, *) には直交マトリックス Q を格納する。wantq = .FALSE. の場合、q は参照されない。
q の第 2 次元は少なくとも max(1, n) でなければならない。

ldt　INTEGER。t の第 1 次元。max(1, n) 以上であること。

ldq　INTEGER。q の第 1 次元。
wantq= .FALSE. の場合、ldq ≥ 1
wantq = .TRUE. の場合、ldq ≥ max(1, n)

j1　INTEGER。第 1 のブロック T_{11} の第 1 行のインデックス。

n1　INTEGER。第 1 のブロック T_{11} の次数 (n1 = 0、1、または 2)。

n2　INTEGER。第 2 のブロック T_{22} の次数 (n1 = 0、1、または 2)。

work　REAL (slaexC の場合)
DOUBLE PRECISION (dlaexc の場合)
ワークスペース配列 DIMENSION は (n)

出力パラメータ

t　schur 標準形を持つ更新されたマトリックス T で上書きされる。

q　wantq = .TRUE. の場合、更新されたマトリックス Q で上書きされる。

info　INTEGER
info = 0 の場合、正常に終了したことを示す。
info = 1 の場合、変換されたマトリックス T は Schur 形式に遠すぎたことを示す。ブロックは交換されず、また T と Q は変更されない。

?lag2

2 × 2 一般固有値問題の固有値を、
オーバーフロー / アンダーフローが
発生しないように必要に応じて
スケーリングを行い計算する。

```
call slag2 ( a, lda, b, ldb, safmin, scale1, scale2, wr1, wr2, wi )
call dlag2 ( a, lda, b, ldb, safmin, scale1, scale2, wr1, wr2, wi )
```

説明

このルーチンは、2 × 2 一般固有値問題 $A - w B$ の固有値を、オーバーフロー / アンダーフローが発生しないように必要に応じてスケーリングを行い計算する。スケール係数 s によって固有値の式は次のように変更される。

$s A - w B$、

s は、w、$w B$、$s A$ がオーバーフローを起こさないように、かつ可能ならばアンダーフローも起こさないように選ばれた非負のスケール係数である。

入力パラメータ

a, b	REAL (slag2 の場合) DOUBLE PRECISION (dlag2 の場合) 配列： *a*(*lda*, 2) には 2 × 2 のマトリックス A を格納する。この 1- ノルムは 1/*safmin* よりも小さいとする。 sqrt(*safmin*)*norm(A) より小さい成分はゼロとして取り扱われる。 *b*(*ldb*, 2) には 2 × 2 上三角マトリックス B を格納する。この 1- ノルムは 1/*safmin* よりも小さいとする。対角は B の (絶対値で) 最大成分の少なくとも sqrt(*safmin*) 倍でなければならない。もし対角がこれよりも小さい場合は、対角に変わって +/- sqrt(*safmin*) が使用される。
lda	INTEGER。*a* の第 1 次元。*lda* ≥ 2

ldb INTEGER。*b* の第 1 次元。*ldb* ≥ 2

safmin REAL (slag2 の場合)
DOUBLE PRECISION (dlag2 の場合)
1/*safmin* をオーバーフローさせない最も小さな正の値。(これは常に ?lamch('S') でなければならない。?lamch を頻繁に呼び出すのを避ける引数である。)

出力パラメータ

scale1 REAL (slag2 の場合)
DOUBLE PRECISION (dlag2 の場合)
第 1 の固有値を定義する固有値の式で、オーバーフロー / アンダーフローを防ぐために使用されたスケール係数。固有値が複素の場合、固有値は (*wr1* +/- *wi* ***i***) / *scale1* (マシンの指数範囲を超えるときがある) で、*scale1*=*scale2* となり、*scale1* は常に正となる。
固有値が実数の場合、第 1(実数) の固有値は *wr1* / *scale1* となるが、これはオーバーフローかアンダーフローを起こすときがあり、実際、正確な固有値が充分に大きい場合、*scale1* はゼロかアンダーフローしきい値未満になることがある。

scale2 REAL (slag2 の場合)
DOUBLE PRECISION (dlag2 の場合)
第 2 の固有値を定義する固有値の式で、オーバーフロー / アンダーフローを防ぐために使用されたスケール係数。固有値が複素の場合、*scale1*=*scale2* となる。固有値が実数の場合、第 2(実数) の固有値は *wr2*/*scale2* となるが、これはオーバーフローかアンダーフローを起こすときがあり、実際、正確な固有値が充分に大きいと、*scale2* はゼロかアンダーフローしきい値未満になることがある。

wr1 REAL (slag2 の場合)
DOUBLE PRECISION (dlag2 の場合)
固有値が実数の場合、*wr1* は AB^{-1} の (2, 2) 成分に最も近い固有値の *scale1* 倍となる。固有値が複素の場合、*wr1* は固有値の実数部の *scale1* 倍となり *wr1* = *wr2* である。

wr2
REAL (slag2 の場合)
DOUBLE PRECISION (dlag2 の場合)
固有値が実数の場合、*wr2* はその他の固有値の *scale2* 倍になる。固有値が複素の場合、*wr1* は固有値の実数部の *scale1* 倍となり *wr1* = *wr2* である。

wi
REAL (slag2 の場合)
DOUBLE PRECISION (dlag2 の場合)
固有値が実数の場合、*wi* はゼロである。固有値が複素の場合、*wi* は固有値の虚数部の *scale1* 倍となる。*wi* は常に非負である。

?lags2

2×2 直交マトリックス U、V、Q を計算し、変換された A と B の行が平行であるような A と B にそれらを適用する。

```
call slags2 ( upper, a1, a2, a3, b1, b2, b3, csu, snu, csv, snv, csq,
              snq )
call dlags2 ( upper, a1, a2, a3, b1, b2, b3, csu, snu, csv, snv, csq,
              snq )
```

説明

このルーチンは 2×2 直交マトリックス U、V、Q を計算し、`upper` = `.TRUE.` であれば、

$$U'*A*Q = U'*\begin{bmatrix} A_1 & A_2 \\ 0 & A_3 \end{bmatrix}*Q = \begin{bmatrix} \mathrm{x} & 0 \\ \mathrm{x} & \mathrm{x} \end{bmatrix}$$

および

$$V'*B*Q = V'*\begin{bmatrix} B_1 & B_2 \\ 0 & B_3 \end{bmatrix}*Q = \begin{bmatrix} \mathrm{x} & 0 \\ \mathrm{x} & \mathrm{x} \end{bmatrix}$$

または、`upper` = `.FALSE.` であれば、

$$U'*A*Q = U'*\begin{bmatrix} A_1 & 0 \\ A_2 & A_3 \end{bmatrix}*Q = \begin{bmatrix} \mathrm{x} & \mathrm{x} \\ 0 & \mathrm{x} \end{bmatrix}$$

および

$$V'*B*Q = V'*\begin{bmatrix} B_1 & 0 \\ B_2 & B_3 \end{bmatrix}*Q = \begin{bmatrix} \mathrm{x} & \mathrm{x} \\ 0 & \mathrm{x} \end{bmatrix}$$

変換された A と B の行は平行で、ここで

$$U = \begin{bmatrix} csu & snu \\ -snu & csu \end{bmatrix}, \quad V = \begin{bmatrix} csv & snv \\ -snv & csv \end{bmatrix}, \quad Q = \begin{bmatrix} csq & snq \\ -snq & csq \end{bmatrix}$$

Z' は Z の転置を表わす。

入力パラメータ

upper
: LOGICAL。
upper = .TRUE. の場合、入力マトリックス A と B は上三角である。
upper = .FALSE. の場合、入力マトリックス A と B は下三角である。

a1, a2, a3
: REAL (slags2 の場合)
DOUBLE PRECISION (dlags2 の場合)
a1、*a2*、*a3* には入力 2 × 2 上 (下) 三角マトリックス A の成分を格納する。

b1, b2, b3
: REAL (slags2 の場合)
DOUBLE PRECISION (dlags2 の場合)
b1、*b2*、*b3* には入力 2 × 2 上 (下) 三角マトリックス B の成分を格納する。

出力パラメータ

csu, snu
: REAL (slags2 の場合)
DOUBLE PRECISION (dlags2 の場合)
目的の直交マトリックス U

csv, snv
: REAL (slags2 の場合)
DOUBLE PRECISION (dlags2 の場合)
目的の直交マトリックス V

csq, snq
: REAL (slags2 の場合)
DOUBLE PRECISION (dlags2 の場合)
目的の直交マトリックス Q

?lagtf

行交換を伴なう部分ピボット演算を用いて
マトリックス T-λI の LU 因子分解を
計算する。T は一般三重対角マトリックス、
λ はスカラー。

```
call slagtf ( n, a, lambda, b, c, tol, d, in, info )
call dlagtf ( n, a, lambda, b, c, tol, d, in, info )
```

説明

このルーチンは次のようにマトリックス (T - `lambda`*I) を因子分解する。T は `n` × `n` 三重対角マトリックス、`lambda` はスカラーである。

T - `lambda`*$I = P\ L\ U$、

P は置換マトリックス、L は非ゼロの劣対角成分を列あたり多くても 1 つしか持たない単位下三重対角マトリックス、U は非ゼロの優対角成分を列あたり多くても 1 つしか持たない上三重対角マトリックスである。因子分解は部分ピボット演算と黙示的な行スケーリングを用いたガウス消去によって実行される。T の固有ベクトルを逆反復によって取得するために、`?lagtf` と `?lagts` が使用できるようにパラメータ `lambda` はルーチンに格納されている。

入力パラメータ

`n`	`INTEGER`。マトリックス T の次数 ($n \geq 0$)。
`a, b, c`	`REAL` (`slagtf` の場合) `DOUBLE PRECISION` (`dlagtf` の場合) 配列、次元は、`a`(n), `b`(n-1), `c`(n-1) `a(*)` にはマトリックス T の対角成分を格納しなければならない。 `b(*)` にはマトリックス T の (n-1) 個の優対角成分を格納しなければならない。 `c(*)` にはマトリックス T の (n-1) 個の劣対角成分を格納しなければならない。

tol
REAL (slagtf の場合)
DOUBLE PRECISION (dlagtf の場合)
マトリックス (*T* - *lambda***I*) がほとんど特異値かどうかを示すために使われる相対許容値。*tol* は通常、*T* の成分におけるおよその最大相対誤差として選択される。たとえば、*T* の成分がおよそ 4 桁の有効数字で正しい場合、*tol* はおよそ 5*10-4 に設定しなければならない。*tol* が相対マシン精度 eps よりも小さく与えられた場合は *tol* の代わりに eps の値が使われる。

出力パラメータ

a
T を因子分解した上三角マトリックス *U* の *n* 個の対角成分で上書きされる。

b
T を因子分解したマトリックス *U* の *n*-1 個の優対角成分で上書きされる。

c
T を因子分解したマトリックス *L* の *n*-1 個の劣対角成分で上書きされる。

d
REAL (slagtf の場合)
DOUBLE PRECISION (dlagtf の場合)
配列、次元は (*n*-2)
T を因子分解したマトリックス *U* の *n*-2 個の第 2 の優対角成分で上書きされる。

in
INTEGER
配列、次元は (*n*)
in には置換マトリックス *P* の詳細が格納される。消去の *k* 番目の過程で交換が起こった場合は *in*(k) = 1 となり、それ以外では *in*(k) = 0 となる。成分 *in*(*n*) は以下を満たすような最小の正の整数を返す。

abs(*u*(j, j)) ≤ norm((*T* - *lambda***I*)(j))* *tol*、

norm(*A*(j)) はマトリックス *A* の j 行目の絶対値の総和を示す。そのような j が存在しない場合は *in*(*n*) はゼロとして返される。*in*(*n*) が正で返された場合、*U* の対角成分は小さく、(*T* - *lambda***I*) が特異値またはほとんど特異値であることを示す。

info INTEGER

info = 0 の場合、正常に終了したことを示す。
info = -*k* の場合、*k* 番目のパラメータの値が不正だったことを示す。

?lagtm

$C = \alpha AB+\beta C$ の形式で示される
マトリックス=マトリックス積を実行し、
ここで A は三重対角マトリックス、B と
C は矩形マトリックス、α と β は
スカラーで 0、1、または -1 である。

```
call slagtm( trans, n, nrhs, alpha, dl, d, du, x, ldx, beta, b, ldb)
call dlagtm( trans, n, nrhs, alpha, dl, d, du, x, ldx, beta, b, ldb)
call clagtm( trans, n, nrhs, alpha, dl, d, du, x, ldx, beta, b, ldb)
call zlagtm( trans, n, nrhs, alpha, dl, d, du, x, ldx, beta, b, ldb)
```

説明

このルーチンは次の形式で示されるマトリックス・ベクトル積を実行する。

$B := alpha*A*X + beta*B$

ここで A は次数 n の三重対角マトリックス、B と X は $n \times nrhs$ マトリックス、*alpha* と *beta* は実数のスカラーで、それぞれ 0.、1.、または -1. である。

入力パラメータ

trans	CHARACTER*1。'N'、'T'、または 'C' のいずれかでなければならない。 式の形式を示す。 *trans* = 'N' の場合、$B := alpha*A*X + beta*B$ (転置なし) *trans* = 'T' の場合、$B := alpha*A^T*X + beta*B$ (転置) *trans* = 'C' の場合、$B := alpha*A^H*X + beta*B$ (共役転置)
n	INTEGER。マトリックス A の次数 ($n \geq 0$)。
nrhs	INTEGER。右辺の個数。すなわち、X と B にある列の数 ($nrhs \geq 0$)。

alpha, beta
REAL (slagtm/clagtm の場合)
DOUBLE PRECISION (dlagtm/zlagtm の場合)
スカラー α と β。*alpha* は 0.、1.、または -1. のいずれかでなければならず、それ以外の場合は 0 とみなされる。*beta* は 0.、1.、または -1. のいずれかでなければならず、それ以外の場合は 1 とみなされる。

dl, d, du
REAL (slagtm の場合)
DOUBLE PRECISION (dlagtm の場合)
COMPLEX (clagtm の場合)
COMPLEX*16 (zlagtm の場合)
配列 : *dl*(*n* - 1), *d*(*n*), *du*(*n* - 1)
配列 *dl* には *T* の (*n*-1) 個の劣対角成分を格納する。
配列 *d* には *T* の *n* 個の対角成分を格納する。
配列 *du* には (*n*-1) 個の優対角成分を格納する。

x, b
REAL (slagtm の場合)
DOUBLE PRECISION (dlagtm の場合)
COMPLEX (clagtm の場合)
COMPLEX*16 (zlagtm の場合)
配列 :
x(*ldx*, *)には *n* × *nrhs* のマトリックス *X* を格納する。*x* の第 2 次元 は少なくとも max(1, *nrhs*) でなければならない。
b(*ldb*, *) には *n* × *nrhs* のマトリックス *B* を格納する。*b* の第 2 次元 は少なくとも max(1, *nrhs*) でなければならない。

ldx
INTEGER。配列 *x* のリーディング・ディメンジョン。
ldx ≥ max(1, *n*)

ldb
INTEGER。配列 *b* のリーディング・ディメンジョン。
ldb ≥ max(1, *n*)

出力パラメータ

b
次のマトリックス式で上書きされる。
B := *alpha***A***X* + *beta***B*

?lagts

?lagtf で計算された *LU* 因子分解を用いて
連立方程式 $(T-\lambda I)x = y$ または
$(T-\lambda I)^T x = y$ を解く。ここで *T* は一般
三重対角マトリックス、λ はスカラー。

```
call slagts ( job, n, a, b, c, d, in, y, tol, info )
call dlagts ( job, n, a, b, c, d, in, y, tol, info )
```

説明

このルーチンは次のどちらかの連立方程式を *x* について解くために使用される。

$(T - lambda*I)*x = y$　または　$(T - lambda*I)'*x = y$、

ここで *T* は $n \times n$ の三重対角マトリックスで、ルーチン ?lagtf で

$T - lambda*I = P\ L\ U$、

として計算された $(T - lambda*I)$ の因子分解から続く。

解く式は引数 *job* によって選択し、どちらの場合もゼロへの摂動か *U* のきわめて小さい対角成分のオプションがあり、このオプションは逆反復のようなアプリケーションでの使用を意図したものである。

入力パラメータ

job　INTEGER。?lagts で実行される内容を次の中から指定する。
= 1 の場合、式 $(T - lambda*I)x = y$ を解くが *U* の対角成分を摂動させない。
= -1 の場合、式 $(T - lambda*I)x = y$ を解き、かつ、オーバーフローが起きるなら *U* の対角成分を摂動させる。下記の引数 *tol* を参照のこと。

= 2 の場合、式 $(T$ - *lambda*$*I)'x = y$ を解くが U の対角成分を摂動させない。

= -2 の場合、$(T$ - *lambda*$*I)'x = y$ 解き、かつ、オーバーフローが起きるなら U の対角成分を摂動させる。下記の引数 *tol* を参照のこと。

n INTEGER。マトリックス T の次数 ($n \geq 0$)。

a, b, c, d REAL (slagts の場合)
DOUBLE PRECISION (dlagts の場合)
配列、次元は、*a*(*n*), *b*(*n*-1), *c*(*n*-1), *d*(*n*-2)
a(*) には、?lagtf から返されたとおり、U の対角成分を格納しなければならない。
b(*) には、?lagtf から返されたとおり、U の第 1 優対角成分を格納しなければならない。
c(*) には、?lagtf から返されたとおり、L の劣対角成分を格納しなければならない。
d(*) には、?lagtf から返されたとおり、U の第 2 優対角成分を格納しなければならない。

in INTEGER
配列、次元は (*n*)
in(*) には、?lagtf から返されたとおり、マトリックス P の詳細を格納しなければならない。

y REAL (slagts の場合)
DOUBLE PRECISION (dlagts の場合)
配列、次元は (*n*)。ベクトル y の右辺である。

tol REAL (slagtf の場合)
DOUBLE PRECISION (dlagtf の場合)
job < 0 の場合、*tol* は、U のきわめて小さな対角成分に対して作られる最小摂動でなければならない。
tol は通常およそ *eps**norm(U) として選ばれ、ここで *eps* は相対マシン精度であるが、*tol* が 0 未満として与えられた場合は *eps**max(abs(u(i, j))) に設定しなおされる。*job* > 0 の場合、*tol* は参照されない。

出力パラメータ

y	*y* は解ベクトル *x* によって上書きされる。
tol	上記**入力パラメータ**の項で説明したとおり、*tol* が入力において 0 未満の場合のみ *tol* は変更される。それ以外では *tol* は変更されない。
info	INTEGER *info* = 0 の場合、正常に終了したことを示す。 *info* = -*i* の場合、*i* 番目のパラメータの値が不正だったことを示す。 *info* = *i* > 0 の場合、解ベクトル *x* の *i* 番目の成分の計算でオーバーフローが起こったことを示す。これは *job* が正として与えられ、*U* の対角成分がきわめて小さいかベクトル *y* の右辺の成分がきわめて大きい場合にのみ起こる

?lagv2

実数の 2 × 2 行列束 (*A*, *B*) の汎用 Schur 因子分解を計算する。ここで *B* は上三角である。

```
call slagv2 ( a, lda, b, ldb, alphar, alphai, beta, csl, snl, csr, snr )
call dlagv2 ( a, lda, b, ldb, alphar, alphai, beta, csl, snl, csr, snr )
```

説明

このルーチンは 2 × 2 行列束 (*A*, *B*) の汎用 Schur 因子分解を計算する。ここで *B* は上三角である。ルーチンは、次を満たすような *csl*, *snl* と *csr*, *snr* で与えられる直交 (回転) マトリックスを計算する。

1) 束 (*A*, *B*) が 2 個の実数固有値 (0/0 または 1/0 タイプを含む) を持つ場合、

$$\begin{bmatrix} a_{11} & a_{12} \\ 0 & a_{22} \end{bmatrix} = \begin{bmatrix} csl & snl \\ -snl & csl \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} csr & -snr \\ snr & csr \end{bmatrix}$$

$$\begin{bmatrix} b_{11} & b_{12} \\ 0 & b_{22} \end{bmatrix} = \begin{bmatrix} csl & snl \\ -snl & csl \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ 0 & b_{22} \end{bmatrix} \begin{bmatrix} csr & -snr \\ snr & csr \end{bmatrix}$$

2) 束 (*A*, *B*) が複素共役固有値のペアを持つ場合、

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = \begin{bmatrix} csl & snl \\ -snl & csl \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} csr & -snr \\ snr & csr \end{bmatrix}$$

$$\begin{bmatrix} b_{11} & 0 \\ 0 & b_{22} \end{bmatrix} = \begin{bmatrix} csl & snl \\ -snl & csl \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ 0 & b_{22} \end{bmatrix} \begin{bmatrix} csr & -snr \\ snr & csr \end{bmatrix}$$

ここで $b_{11} \geq b_{22} > 0$

入力パラメータ

a, b
REAL (slagv2 の場合)
DOUBLE PRECISION (dlagv2 の場合)
配列：
a(*lda*, 2) には 2 × 2 マトリックス *A* を格納する。
b(*ldb*, 2) には上三角 2 × 2 マトリックス *B* を格納する。

lda
INTEGER。配列 *a* のリーディング・ディメンジョン。
lda ≥ 2

ldb
INTEGER。配列 *b* のリーディング・ディメンジョン。
ldb ≥ 2

出力パラメータ

a
a は汎用 Schur 形式の "*A*- 部分 " で上書きされる。

b
b は汎用 Schur 形式の "*B*- 部分 " で上書きされる。

alphar, alphai, beta
REAL (slagv2 の場合)
DOUBLE PRECISION (dlagv2 の場合)
配列、次元はそれぞれ、(2)
(*alphar*(k)+*i***alphai*(k))/*beta*(k) は束 (*A, B*) の固有値で、k = 1, 2 および *i* = sqrt(-1) である。
beta(k) はゼロとなることがある点に注意する。

csl, snl
REAL (slagv2 の場合)
DOUBLE PRECISION (dlagv2 の場合)
それぞれ、左回転マトリックスの余弦と正弦である。

csr, snr
REAL (slagv2 の場合)
DOUBLE PRECISION (dlagv2 の場合)
それぞれ、右回転マトリックスの余弦と正弦である。

?lahqr

ダブル・シフト / シングル・シフト QR アルゴリズムを用いて、上 Hessenberg マトリックスの固有値と Schur 因子分解を計算する。

```
call slahqr ( wantt, wantz, n, ilo, ihi, h, ldh, wr, wi, iloz, ihiz,
              z, ldz, info )
call dlahqr ( wantt, wantz, n, ilo, ihi, h, ldh, wr, wi, iloz, ihiz,
              z, ldz, info )
call clahqr ( wantt, wantz, n, ilo, ihi, h, ldh, w, iloz, ihiz, z,
              ldz, info )
call zlahqr ( wantt, wantz, n, ilo, ihi, h, ldh, w, iloz, ihiz, z,
              ldz, info )
```

説明

このルーチンは ?hseqr から呼び出される補助ルーチンで、*ilo* から *ihi* の行と列にある Hessenberg 部分マトリックスを対象として、?hseqr によって過去に計算された固有値と Schur 分解を更新する。

入力パラメータ

wantt LOGICAL。
wantt = .TRUE. の場合、すべての Schur 形式 T が必要である。
wantt = .FALSE. の場合、固有値のみが必要である。

wantz LOGICAL。
wantz = .TRUE. の場合、Schur ベクトル Z のマトリックスが必要である。
wantz = .FALSE. の場合、Schur ベクトルは必要ない。

n INTEGER。マトリックス H の次数 ($n \geq 0$)。

ilo, ihi INTEGER
H は、行と列が *ihi*+1:*n* の上準三角になっているとする。また、H(*ilo*, *ilo*-1) = 0 (unless *ilo* = 1) であるとする。ルーチン ?lahqr は基本的に行と列が *ilo* から *ihi* の Hessenberg 部分マトリックスを対象とするが、*wantt* = .TRUE. の場合は H のすべてに変換が適用される。

制約は以下のとおりである。
$1 \le ilo \le \max(1, ihi)$; $ihi \le n$

h, z
REAL (slahqr の場合)
DOUBLE PRECISION (dlahqr の場合)
COMPLEX (clahqr の場合)
COMPLEX*16 (zlahqr の場合)
配列：
*h(ldh, *)* には上HessenbergマトリックスHを格納する。
h の第2次元は少なくとも max(1, *n*) でなければならない。
*z(ldz, *)*
wantz = .TRUE. の場合、*z* には ?hseqr で蓄積された変換の現在のマトリックス Z を格納しなければならない。
wantz = .FALSE. の場合、*z* は参照されない。
z の第2次元は少なくとも max(1, *n*) でなければならない。

ldh
INTEGER。*h* の第 1 次元。max(1, *n*) 以上であること。

ldz
INTEGER。*z* の第 1 次元。max(1, *n*) 以上であること。

iloz, ihiz
INTEGER。*wantz* = .TRUE. の場合、変換が適用されなければならない Z の行を指定する。
$1 \le iloz \le ilo$; $ihi \le ihiz \le n$

出力パラメータ

h
wantt = .TRUE. の場合、H は行と列が *ilo*:*ihi* にある上準三角 (複素型では上三角) で、標準形式の 2×2 対角ブロックを持つ。*wantt* = .FALSE. の場合、H の内容は規定されない。

wr, wi
REAL (slahqr の場合)
DOUBLE PRECISION (dlahqr の場合)
配列 DIMENSION はそれぞれ、max (1, *n*) 以上。実数型でのみ使用される。

計算された固有値 *ilo* から *ihi* の実数部分と虚数部分はそれぞれ *wr* と *wi* の対応する成分に格納される。2 個の固有値が複素共役ペアとして計算された場合は、それらは *wr* と *wi* の連続する成分に格納され、すなわち i 番目と (i+1) 番目で *wi*(i) > 0 かつ *wi*(i+1) < 0 である。*wantt* = .TRUE. の場合、固有値は *H* に返された Schur 形式の対角と同じ順序で格納され、*wr*(i) = *H*(i, i)、また 2 × 2 対角ブロックでは *H*(i:i+1, i:i+1) で、*wi*(i) = sqrt(*H*(i+1, i)*H*(i, i+1)) および *wi*(i+1) = -*wi*(i) である。

w

COMPLEX (clahqr の場合)
COMPLEX*16 (zlahqr の場合)
配列 DIMENSION は少なくとも max (1, *n*)。複素型でのみ使用される。
計算された固有値 *ilo* から *ihi* は対応する *w* の成分に格納される。
wantt = .TRUE. の場合、固有値は *H* に返された Schur 形式の対角と同じ順序で格納され、*wr*(i) = *H*(i, i)。

z

wantz = .TRUE. の場合、*z* は更新されている。変換は部分マトリックス *Z*(*iloz*:*ihiz*, *ilo*:*ihi*) にのみ適用されている。

info

INTEGER
info = 0 の場合、正常に終了したことを示す。
info = *i* > 0 の場合、?lahqr はすべての固有値 *ilo* から *ihi* を、合計 30*(*ihi*-*ilo*+1) 回以内の反復で計算できなかったことを示す。*wr* と *wi* (slahqr/dlahqr の場合) または (clahqr/zlahqr) の成分 i+1:*ihi* には、正常に終了したそれら固有値が格納される。

?lahrd

k 番目の劣対角よりも下の成分がゼロになるように一般矩形マトリックスの最初の *nb* 列を縮退させ、*A* の縮退されていない部分に変換を適用するために必要となる補助マトリックスを返す。

```
call slahrd ( n, k, nb, a, lda, tau, t, ldt, y, ldy )
call dlahrd ( n, k, nb, a, lda, tau, t, ldt, y, ldy )
call clahrd ( n, k, nb, a, lda, tau, t, ldt, y, ldy )
call zlahrd ( n, k, nb, a, lda, tau, t, ldt, y, ldy )
```

説明

このルーチンは、*k* 番目の劣対角よりも下の要素がゼロになるように、実数 / 複素一般 $n \times (n-k+1)$ マトリックス *A* の最初の *nb* 列を縮退させる。この縮退は直交 / ユニタリ相似変換 $Q'AQ$ によって実行される。ルーチンは、ブロック・リフレクタ $I - V T V'$ として *Q* を定義するマトリックス *V* と *T*、およびマトリックス $Y = A V T$ を返す。

マトリックス *Q* は *nb* 個の基本リフレクタの積として表現される。
$Q = H(1) H(2) \ldots H(nb)$

それぞれの $H(i)$ は形式を持つ。

$H(i) = I - tau*v*v'$

tau は実数 / 複素スカラー、*v* は実数 / 複素ベクトルである。

これは `?gehrd` から呼び出される補助ルーチンである。

入力パラメータ

n	`INTEGER`。マトリックス *A* の次数 ($n \geq 0$)。
k	`INTEGER`。縮退に対するオフセット。最初の *nb* 列にある *k* 番目の劣対角よりも下の成分がゼロに縮退される。
nb	`INTEGER`。縮退させる列の数。

a REAL (slahrd の場合)
DOUBLE PRECISION (dlahrd の場合)
COMPLEX (clahrd の場合)
COMPLEX*16 (zlahrd の場合)
a(*lda*, *n*-*k*+1) には縮退対象の *n* × (*n*-*k*+1) 一般マトリックス *A* を格納する。

lda INTEGER。*a* の第 1 次元。max(1, *n*) 以上であること。

ldt INTEGER。出力配列 *t* の第 1 次元のサイズ。max(1, *nb*) 以上でなければならない。

ldy INTEGER。出力配列 *y* の第 1 次元のサイズ。max(1, *n*) 以上でなければならない。

出力パラメータ

a 最初の *nb* 列にある *k* 番目の劣対角成分とその上の成分は縮退されたマトリックスの対応する成分で上書きされる。*k* 番目の劣対角よりも下の成分は配列 *tau* で上書きされ、基本リフレクタの積としてマトリックス *Q* を表現する。そのほかの列は変更されない。次の**アプリケーション・ノート**を参照。

tau REAL (slahrd の場合)
DOUBLE PRECISION (dlahrd の場合)
COMPLEX (clahrd の場合)
COMPLEX*16 (zlahrd の場合)
配列 DIMENSION は (*nb*)
基本リフレクタのスカラー係数が入る。

t, *y* REAL (slahrd の場合)
DOUBLE PRECISION (dlahrd の場合)
COMPLEX (clahrd の場合)
COMPLEX*16 (zlahrd の場合)
配列、次元は、*t*(*ldt*, *nb*), *y*(*ldy*, *nb*)
配列 *t* には上三角マトリックス *T* が格納される。
配列 *y* には *n* × *nb* のマトリックス *Y* が格納される。

アプリケーション・ノート

基本リフレクタ H(i) に対して、

ルーチン終了時に v(1:i+k-1) = 0, v(i+k) = 1; v(i+k+1:n) は a(i+k+1:n, i) に格納され、*tau* は `tau`(i) に格納される。

ベクトル v の成分は、マトリックスの縮退されていない部分に変換を適用するために、次の形式の更新を用いて、T と Y とともに必要な (n-k+1) × nb のマトリックス V を形成する。
$A := (I - V T V') * (A - Y V')$

n = 7、k = 3、nb = 2 におけるルーチン終了時の A の内容を以下に示す。

$$\begin{bmatrix} a & h & a & a & a \\ a & h & a & a & a \\ a & h & a & a & a \\ h & h & a & a & a \\ v_1 & h & a & a & a \\ v_1 & v_2 & a & a & a \\ v_1 & v_2 & a & a & a \end{bmatrix}$$

a は元のマトリックス A の成分、h は上 Hessenberg マトリックス H の変更された成分、v_i は H(i) を定義するベクトルの成分を示す。

?laic1

増加条件推定を 1 ステップ適用する。

```
call slaic1 ( job, j, x, sest, w, gamma, sestpr, s, c )
call dlaic1 ( job, j, x, sest, w, gamma, sestpr, s, c )
call claic1 ( job, j, x, sest, w, gamma, sestpr, s, c )
call zlaic1 ( job, j, x, sest, w, gamma, sestpr, s, c )
```

説明

ルーチン ?laic1 は最も簡単な形で増加条件推定を 1 ステップ適用する。

$\|x\|_2 = 1$ ($\|a\|_2$ とは a の 2- ノルムを示す) の x を、

$\|L^*x\|_2 = sest$

を満たすような $j \times j$ の下三角マトリックス L のおおよその特異ベクトルとする。

次に ?laic1 は、ベクトル

$$xhat = \begin{bmatrix} s^*x \\ c \end{bmatrix}$$

が

$$Lhat = \begin{bmatrix} L & 0 \\ w' & gamma \end{bmatrix}$$

のおおよその特異ベクトルを、

$\|Lhat\,^*xhat\|_2 = sestpr$

の意味で満たす sestpr、s、c を計算する。

job によって最大または最小特異値に対する推定が計算される。

$[s\ c]'$ と $sestpr^2$ は、次の連立方程式の固有ペアである点に注意する (slaic1/claic の場合)。

$$\mathrm{diag}(sest*sest, 0) + [alpha \quad gamma] \quad * \begin{bmatrix} alpha \\ gamma \end{bmatrix}$$

ここで、$alpha = x'*w$、

または、次の連立方程式の固有ペアである (claic1/zlaic の場合)。

$$\mathrm{diag}(sest*sest, 0) + [alpha \quad gamma] \quad * \begin{bmatrix} \mathrm{conjg}(alpha) \\ \mathrm{conjg}(gamma) \end{bmatrix}$$

ここで、$alpha = \mathrm{conjg}(x)'*w$

入力パラメータ

job INTEGER

job = 1 の場合、最大特異値に対する推定が計算される。

job = 2 の場合、最小特異値に対する推定が計算される。

j INTEGER。*x* と *w* の長さ。

x, w REAL (slaic1 の場合)

DOUBLE PRECISION (dlaic1 の場合)

COMPLEX (claic1 の場合)

COMPLEX*16 (zlaic1 の場合)

配列、次元はそれぞれ、(*j*)

それぞれ、ベクトル *x* と *w* を格納する。

sest REAL (slaic1/claic1 の場合)

DOUBLE PRECISION (dlaic1/zlaic1 の場合)

j × *j* のマトリックス *L* の推定された特異値。

gamma REAL (slaic1 の場合)

DOUBLE PRECISION (dlaic1 の場合)

COMPLEX (claic1 の場合)

COMPLEX*16 (zlaic1 の場合)

対角成分 *gamma*

出力パラメータ

sestpr REAL (slaic1/claic1 の場合)
DOUBLE PRECISION (dlaic1/zlaic1 の場合)
(j+1) × (j+1) のマトリックス *Lhat* の推定された特異値。

s, c REAL (slaic1 の場合)
DOUBLE PRECISION (dlaic1 の場合)
COMPLEX (claic1 の場合)
COMPLEX*16 (zlaic1 の場合)
xhat の構成に必要な正弦と余弦。

?laln2

指定された形式の 1 × 1 または 2 × 2 の
線形連立方程式を解く。

```
call slaln2( ltrans, na, nw, smin, ca, a, lda, d1, d2, b, ldb, wr, wi,
             x, ldx, scale, xnorm, info )
call dlaln2( ltrans, na, nw, smin, ca, a, lda, d1, d2, b, ldb, wr, wi,
             x, ldx, scale, xnorm, info )
```

説明

このルーチンは次の形式の連立方程式を、可能なスケーリング (s) と A の摂動 (A' は A の転置) を使って解く。

$(ca\,A - w\,D)\,X = s\,B$ または $(ca\,A' - w\,D)\,X = s\,B$

A は $na \times na$ の実数マトリックス、ca は実数スカラー、D は $na \times na$ の実数対角マトリックス、w は実数または複素値、X と B は $na \times 1$ のマトリックスで w が実数の場合は実数、w が複素の場合は複素である。パラメータ na は 1 か 2 である。

w が複素の場合、X と B は $na \times 2$ のマトリックスとして表現され、それぞれの第 1 の列が実数部、第 2 の列が虚数部となる。

このルーチンは X をオーバーフローせずに計算できるようにスケール係数 s (≤ 1) を選択する。X は、norm($ca\,A - w\,D$)*norm(X) がオーバーフロー未満になるように、必要に応じてさらにスケーリングされる。

($ca\,A - w\,D$) の両方の特異値が $smin$ よりも小さい場合、$smin * I$ (ここで I は単位を表わす) が ($ca\,A - w\,D$) の代わりに使用される。1 つの固有値が $smin$ よりも小さい場合は、最小特異値がおおよそ $smin$ になるように ($ca\,A - w\,D$) の 1 つの成分が摂動される。両方の特異値が少なくとも $smin$ ならば、($ca\,A - w\,D$) は摂動されない。
どのような場合でも、摂動は、大きくとも
max($smin$, ulp * norm($ca\,A - w\,D$)) の小さな倍数となる。

特異値は無限ノルム近似によって計算され、そのため、2 程度の係数に対してのみ正確となる。

注：入力の大きさは、妥当な係数によってオーバーフローよりも小さいと仮定される (*bignum* 参照)。

入力パラメータ

trans LOGICAL。
trans = .TRUE. の場合、*A*- 転置が使用される。
trans = .FALSE. の場合、*A* が使用される (転置なし)。

na INTEGER。マトリックス *A* のサイズである。1 か 2 のみを取り得る。

nw INTEGER。*w* が実数の場合、このパラメータは 1、*w* が複素数の場合、パラメータは 2 でなければならない。1 か 2 のみを取り得る。

smin REAL (slaln2 の場合)
DOUBLE PRECISION (dlaln2 の場合)
A の特異値における目的の下限。
これはアンダーフローまたはオーバーフローに対して安全な距離を持っていなければならない。たとえば、(*underflow/machine_precision*) と (*machine_precision* * *overflow*) の間とする。(*bignum* と *ulp* を参照)

ca REAL (slaln2 の場合)
DOUBLE PRECISION (dlaln2 の場合)
A に乗算される係数。

a REAL (slaln2 の場合)
DOUBLE PRECISION (dlaln2 の場合)
配列 DIMENSION は (*lda*, *na*)。*na* × *na* のマトリックス *A*

lda INTEGER。*a* のリーディング・ディメンジョン。*na* 以上でなければならない。

d1, d2 REAL (slaln2 の場合)
DOUBLE PRECISION (dlaln2 の場合)
それぞれ対角マトリックス *D* にある (1, 1) と (2, 2) の成分。*nw* = 1 の場合は *d2* は使用されない。

b REAL (slaln2 の場合)
DOUBLE PRECISION (dlaln2 の場合)
配列 DIMENSION は (*ldb*, *nw*)。*na* × *nw* のマトリックス *B* (右辺)。*nw* = 2 (*w* が複素) の場合、列 1 には *B* の実数部分を格納し、列 2 には虚数部分を格納する。

ldb INTEGER。*b* のリーディング・ディメンジョン。*na* 以上でなければならない。

wr, wi REAL (slaln2 の場合)
DOUBLE PRECISION (dlaln2 の場合)
それぞれスカラー *w* の実数部と虚数部。
nw = 1 の場合は *wi* は参照されない。

ldx INTEGER。出力配列 *x* のリーディング・ディメンジョン。*na* 以上でなければならない。

出力パラメータ

x REAL (slaln2 の場合)
DOUBLE PRECISION (dlaln2 の場合)
配列 DIMENSION は (*ldx*, *nw*)。このルーチンによって計算された *na* × *nw* のマトリックス *X* (未知)。
nw = 2 の場合 (*w* が複素)、列 1 には *X* の実数部が格納され、列 2 には虚数部が格納される。

scale REAL (slaln2 の場合)
DOUBLE PRECISION (dlaln2 の場合)
X の計算でオーバーフローを起こさないように *B* に乗算されるべきスケール係数。そのため、(*ca A* - *w D*) *X* は *B* ではなく *scale***B* となる (*A* の摂動は無視)。
大きくとも 1 である。

xnorm REAL (slaln2 の場合)
DOUBLE PRECISION (dlaln2 の場合)
X が *na* × *nw* の実数マトリックスとしてみなされる場合、*X* の無限ノルム。

info INTEGER
エラー・フラグ。エラーが生じなかった場合はゼロに、引数にエラーがあった場合は負に、(*ca A* - *w D*) を摂動しなければならない場合は正になる。

取り得る値は次のとおりである。
info = 0 の場合、正常に終了したことを示し、
(*ca* A - w D) を摂動する必要がなかった。
info = 1 の場合、最小(または唯一の)特異値を *smin* よりも大きくするために、(*ca* A - w D) を摂動する必要があったことを示す。

注：処理速度を勘案してこのルーチンでは入力のエラーをチェックしない。

?lals0

最小二乗問題を分割統治 SVD 法を
使用して解く過程で乗率を逆に適用する。
?gelsd によって使用される。

```
call slals0( icompq, nl, nr, sqre, nrhs, b, ldb, bx, ldbx, perm,
             givptr, givcol, ldgcol, givnum, ldgnum, poles, difl,
             difr, z, k, c, s, work, info )
call dlals0( icompq, nl, nr, sqre, nrhs, b, ldb, bx, ldbx, perm,
             givptr, givcol, ldgcol, givnum, ldgnum, poles, difl,
             difr, z, k, c, s, work, info )
call clals0 ( icompq, nl, nr, sqre, nrhs, b, ldb, bx, ldbx, perm,
             givptr, givcol, ldgcol, givnum, ldgnum, poles, difl,
             difr, z, k, c, s, rwork, info )
call zlals0 ( icompq, nl, nr, sqre, nrhs, b, ldb, bx, ldbx, perm,
             givptr, givcol, ldgcol, givnum, ldgnum, poles, difl,
             difr, z, k, c, s, rwork, info )
```

説明

このルーチンは、分割統治 SVD 法を使用して最小二乗問題を解く過程で、右辺マトリックス *B* に行を追加した対角マトリックスの左または右特異ベクトル・マトリックスのいずれかの乗率を逆に適用する。

左特異ベクトル・マトリックスにおいて、3 タイプの直交マトリックスが関係する。

(1L) Givens 回転。回転の回数は *givptr* に格納する。回転が適用された列 / 行のペアは *givcol* に格納する。これら回転の *c* 値と *s* 値は *givnum* に格納する。

(2L) 置換。*B* の (*nl*+1) 番目の行が第 1 の行に移動される行であり、*j* = 2:*n* において、*B* の *perm*(j) 番目の行が j 番目の行に移動される行である。

(3L) 残りのマトリックスの左特異ベクトル・マトリックス。

右特異ベクトル・マトリックスにおいて、4 タイプの直交マトリックスが関係する。

(1R) 残りのマトリックスの右特異ベクトル・マトリックス。

(2R) *sqre* = 1 の場合、右ヌル空間を生成するために 1 回の追加 Givens 回転。

(3R) (2L) の逆変換。

(4R) (1L) の逆変換。

入力パラメータ

icompq	INTEGER。特異ベクトルを因子分解された形式で計算するかを指定する。 *icompQ* = 0 の場合、左特異ベクトル・マトリックス。 *icompq* = 1 の場合、右特異ベクトル・マトリックス。
nl	INTEGER。上ブロックの行次元。*nl* ≥ 1
nr	INTEGER。下ブロックの行次元。*nr* ≥ 1
sqre	INTEGER *sqre* = 0 の場合、下ブロックは *nr* × *nr* の正方マトリックス。 *sqre* = 1 の場合、下ブロックは *nr* × (*nr*+1) の矩形マトリックス。二重対角マトリックスは行次元 *n* = *nl* + *nr* + 1、列次元 *m* = *n* + *sqre* を持つ。

nrhs INTEGER。*b* と *bx* の列の数。1 以上でなければならない。

b REAL (slals0 の場合)
DOUBLE PRECISION (dlals0 の場合)
COMPLEX (clals0 の場合)
COMPLEX*16 (zlals0 の場合)
配列 DIMENSION は (*ldb*, *nrhs*)。最小二乗問題の右辺を行 1 から *m* に格納する。

ldb INTEGER。*b* のリーディング・ディメンジョン。max(1, max(*m*, *n*)) 以上でなければならない。

bx REAL (slals0 の場合)
DOUBLE PRECISION (dlals0 の場合)
COMPLEX (clals0 の場合)
COMPLEX*16 (zlals0 の場合)
ワークスペース配列 DIMENSION は (*ldbx*, *nrhs*)

ldbx INTEGER。*bx* のリーディング・ディメンジョン。

perm INTEGER
配列 DIMENSION は (*n*)。2 個のブロックに適用される (収縮とソートによる) 置換。

givptr INTEGER。この部分問題で実行された Givens 回転の回数。

givcol INTEGER
配列 DIMENSION は (*ldgcol*, 2)。Givens 回転に関与した行 / 列のペアを示す数値のペア。

ldgcol INTEGER。*givcol* のリーディング・ディメンジョン。*n* 以上でなければならない。

givnum
REAL (slals0/clals0 の場合)
DOUBLE PRECISION (dlals0/zlals0 の場合)
配列 DIMENSION は (*ldgnum*, 2)。対応する Givens 回転で使用された *c* 値または *s* 値を示す数。

ldgnum
INTEGER。配列 *difr*、*poles*、*givnum* のリーディング・ディメンジョン。*k* 以上でなければならない。

poles
REAL (slals0/clals0 の場合)
DOUBLE PRECISION (dlals0/zlals0 の場合)
配列 DIMENSION は (*ldgnum*, 2)。*poles*(1:*k*, 1) には永年方程式を解いて得られた新しい特異値を格納し、*poles*(1:*k*, 2) には永年方程式内の極を含む配列を格納する。

difl
REAL (slals0/clals0 の場合)
DOUBLE PRECISION (dlals0/zlals0 の場合)
配列 DIMENSION は (*k*)。*difl*(*i*) には *i* 番目の更新された (収縮されていない) 特異値と *i* 番目の (収縮されていない) 古い特異値の距離を格納する。

difr
REAL (slals0/clals0 の場合)
DOUBLE PRECISION (dlals0/zlals0 の場合)
配列 DIMENSION は (*ldgnum*, 2)。*difr*(*i*, 1) には、*i* 番目の更新された (収縮されていない) 特異値と *i*+*1*番目の(収縮されていない)古い特異値の距離を格納する。*difr*(*i*, 2) には *i* 番目の右特異ベクトルに対する正規化係数を格納する。

z
REAL (slals0/clals0 の場合)
DOUBLE PRECISION (dlals0/zlals0 の場合)
配列 DIMENSION は (*k*)。収縮調整された更新行ベクトルの成分を格納する。

k
INTEGER。非収縮マトリックスの次元を格納する。これは関連する永年方程式の次数である。$1 \le k \le n$

c REAL (slals0/clals0 の場合)
DOUBLE PRECISION (dlals0/zlals0 の場合)
sqre = 0 の場合はガーベッジを格納し、*sqre* = 1 の場合は右ヌル空間に関係する Givens 回転の *c* 値を格納する。

s REAL (slals0/clals0 の場合)
DOUBLE PRECISION (dlals0/zlals0 の場合)
sqre = 0 の場合はガーベッジを格納し、*sqre* = 1 の場合は右ヌル空間に関係する Givens 回転の *s* 値を格納する。

work REAL (slals0 の場合)
DOUBLE PRECISION (dlals0 の場合)
ワークスペース配列 DIMENSION は (*k*)。実数型でのみ使用される。

rwork REAL (clals0 の場合)
DOUBLE PRECISION (zlals0 の場合)
ワークスペース配列 DIMENSION は (*k**(1+*nrhs*) + 2**nrhs*)。複素型でのみ使用される。

出力パラメータ

b 行 1 から *n* は解 *X* で上書きされる。

info INTEGER
info = 0 の場合、正常に終了したことを示す。
info = -*i* < 0 の場合、*i* 番目の引数の値が不正だったことを示す。

?lalsa

コンパクト形式で係数マトリックスの
SVD を計算する。?gelsd で使用される。

```
call slalsa ( icompq, smlsiz, n, nrhs, b, ldb, bx, ldbx, u, ldu, vt,
              k, difl, difr, z, poles, givptr, givcol, ldgcol, perm,
              givnum, c, s, work, iwork, info )
call dlalsa ( icompq, smlsiz, n, nrhs, b, ldb, bx, ldbx, u, ldu, vt,
              k, difl, difr, z, poles, givptr, givcol, ldgcol, perm,
              givnum, c, s, work, iwork, info )
call clalsa ( icompq, smlsiz, n, nrhs, b, ldb, bx, ldbx, u, ldu, vt,
              k, difl, difr, z, poles, givptr, givcol, ldgcol, perm,
              givnum, c, s, rwork, iwork, info )
call zlalsa ( icompq, smlsiz, n, nrhs, b, ldb, bx, ldbx, u, ldu, vt,
              k, difl, difr, z, poles, givptr, givcol, ldgcol, perm,
              givnum, c, s, rwork, iwork, info )
```

説明

このルーチンは最小二乗問題を解く中間の過程であり、コンパクト形式で係数マトリックスの SVD を計算する。特異ベクトルは単純な直交マトリックスの積として計算される。

ルーチン ?lalsa は、*icompQ* = 0 の場合は上二重対角マトリックスの左特異ベクトル・マトリックスの逆を右辺に適用する。*icompq* = 1 の場合は右特異ベクトル・マトリックスを右辺に適用する。特異ベクトル・マトリックスは ?lalsa によってコンパクト形式で生成される。

入力パラメータ

icompq INTEGER。左または右のどちらの特異ベクトル・マトリックスを対象とするか指定する。
iconpq = 0 の場合、左特異ベクトル・マトリックスを使用する。
iconpq = 1 の場合、右特異ベクトル・マトリックスを使用する。

smlsiz INTEGER。計算ツリーの一番下の部分問題の最大サイズ。

n INTEGER。上二重対角マトリックスの行と列の次元。

nrhs INTEGER。*b* と *bx* の列の数。1 以上でなければならない。

b REAL (slalsa の場合)
DOUBLE PRECISION (dlalsa の場合)
COMPLEX (clalsa の場合)
COMPLEX*16 (zlalsa の場合)
配列 DIMENSION は (*ldb*, *nrhs*)。最小二乗問題の右辺を行 1 から *m* に格納する。

ldb INTEGER。呼び出し元のサブプログラムの *b* のリーディング・ディメンジョン。max(1, max(*m*, *n*)) 以上でなければならない。

ldbx INTEGER。出力配列 *bx* のリーディング・ディメンジョン。

u REAL (slalsa/clalsa の場合)
DOUBLE PRECISION (dlalsa/zlalsa の場合)
配列 DIMENSION は (*ldu*, *smlsiz*)。*u* には、一番下のレベルにあるすべての部分問題の左特異ベクトル・マトリックスを格納する。

ldu INTEGER、*ldu* ≥ *n*。配列 *u*、*vt*、*difl*、*difr*、*poles*、*givnum*、*z* のリーディング・ディメンジョン。

vt REAL (slalsa/clalsa の場合)
DOUBLE PRECISION (dlalsa/zlalsa の場合)
配列 DIMENSION は (*ldu*, *smlsiz* +1)。一番下のレベルにあるすべての部分問題の右特異ベクトル・マトリックスを格納する。

k INTEGER。配列 DIMENSION は (*n*)

difl
REAL (slalsa/clalsa の場合)
DOUBLE PRECISION (dlalsa/zlalsa の場合)
配列 DIMENSION は (*ldu*, *nlvl*)、ここで
nlvl = int($\log_2$ (*n* /(*smlsiz*+1))) + 1

difr
REAL (slalsa/clalsa の場合)
DOUBLE PRECISION (dlalsa/zlalsa の場合)
配列 DIMENSION は (*lldu*, 2**nlvl*)。*difl*(*, *i*) と *difr*(*, 2*i* -1) には、*i* 番目のレベルの特異値と (*i*-1) 番目のレベルの特異値の距離を格納する。*difr*(*, 2*i*) には *i* 番目のレベルの部分問題の右特異ベクトル・マトリックスに対する正規化係数を格納する。

z
REAL (slalsa/clalsa の場合)
DOUBLE PRECISION (dlalsa/zlalsa の場合)
配列 DIMENSION は (*ldu*, *nlvl*)。*z*(1, *i*) には *i* 番目レベルの部分問題に対する収縮調整された更新行ベクトルの成分を格納する。

poles
REAL (slalsa/clalsa の場合)
DOUBLE PRECISION (dlalsa/zlalsa の場合)
配列 DIMENSION は (*ldu*, 2**nlvl*)
poles(*, 2*i*-1: 2*i*) には *i* 番目レベルの永年方程式に関係する新しい特異値と古い特異値を格納する。

givptr
INTEGER
配列 DIMENSION は (*n*)
givptr(*i*) には計算ツリーの *i* 番目問題で実行された Givens 回転の回数を格納する。

givcol
INTEGER
配列 DIMENSION は (*ldgcol*, 2**nlvl*)。それぞれの *i* で、*igivcol*(*, 2*i*-1: 2*i*) には計算ツリーの *i* 番目レベルで実行された Givens 回転の位置を格納する。

ldgcol INTEGER、*ldgcol* ≥ *n*。配列 *givcol*、*perm* のリーディング・ディメンジョン。

perm INTEGER。配列 DIMENSION は (*ldgcol*, *nlvl*)
perm(*, *i*) には計算ツリーの *i* 番目レベルで実行された置換を格納する。

givnum REAL (slalsa/clalsa の場合)
DOUBLE PRECISION (dlalsa/zlalsa の場合)
配列 DIMENSION は (*ldu*, 2**nlvl*)
givnum(*, 2*i*-1 : 2*i*) には計算ツリーの *i* 番目レベルで実行された Givens 回転の *c* 値と *s* 値を格納する。

c REAL (slalsa/clalsa の場合)
DOUBLE PRECISION (dlalsa/zlalsa の場合)
配列 DIMENSION は (*n*)。*i* 番目の部分問題が正方ではない場合、*c*(*i*) には *i* 番目の部分問題の右ヌル空間に関係する Givens 回転の *c* 値を格納する。

s REAL (slalsa/clalsa の場合)
DOUBLE PRECISION (dlalsa/zlalsa の場合)
配列 DIMENSION は (*n*)。*i* 番目の部分問題が正方ではない場合、*s*(*i*) には *i* 番目の部分問題の右ヌル空間に関係する Givens 回転の *s* 値を格納する。

work REAL (slalsa の場合)
DOUBLE PRECISION (dlalsa の場合)
ワークスペース配列 DIMENSION は少なくとも (*n*)。実数型でのみ使用される。

rwork REAL (clalsa の場合)
DOUBLE PRECISION (zlalsa の場合)
ワークスペース配列 DIMENSION は少なくとも max(*n*, (*smlsz*+1)**nrhs**3)。複素型でのみ使用される。

iwork INTEGER
ワークスペース配列 DIMENSION は少なくとも (3*n*)

出力パラメータ

b	行 1 から *n* は解 *X* で上書きされる。
bx	REAL (slalsa の場合) DOUBLE PRECISION (dlalsa の場合) COMPLEX (clalsa の場合) COMPLEX*16 (zlalsa の場合) 配列 DIMENSION は (*ldbx*, *nrhs*)。*b* に対する左または右の特異ベクトル・マトリックスの適用結果で上書きされる。
info	INTEGER *info* = 0 の場合、正常に終了したことを示す。 *info* = -*i* < 0 の場合、*i* 番目の引数の値が不正だったことを示す。

?lalsd

最小二乗問題を解くために *A* の特異値分解を使用する。

```
call slalsd ( uplo, smlsiz, n, nrhs, d, e, b, ldb, rcond, rank, work,
              iwork, info )
call dlalsd ( uplo, smlsiz, n, nrhs, d, e, b, ldb, rcond, rank, work,
              iwork, info )
call clalsd ( uplo, smlsiz, n, nrhs, d, e, b, ldb, rcond, rank, work,
              rwork, iwork, info )
call zlalsd ( uplo, smlsiz, n, nrhs, d, e, b, ldb, rcond, rank, work,
              rwork, iwork, info )
```

説明

このルーチンは、*AX-B* の各列のユークリッド・ノルムを最小化するために *X* を求める最小二乗問題を *A* の特異値分解を使用して解き、ここで *A* は *n* × *n* の上二重対角、*X* と *B* は *n* × *nrhs* である。解 *X* は *B* を上書きする。

最大特異値の *rcond* 倍より小さな *A* の特異値は、最小二乗問題を解く過程でゼロとして扱われる。この場合は最小ノルム解が返される。実際の特異値は *d* に昇順で格納され返される。

このコードは浮動小数点演算に対してきわめて緩い仮定を行う。加減算においてガード桁を持つマシン、または Cray X-MP、Cray Y-MP、Cray C-90、または Cray-2 のような減算のガード桁を持たないバイナリ・マシンで動作する。

ガード桁を持たない 16 進マシンや 10 進マシンでは異常終了することが考えられるが、そのような前例はない

入力パラメータ

uplo CHARACTER*1
uplo = 'U' の場合、*d* と *e* は上二重対角マトリックスを定義する。
uplo = 'L' の場合、*d* と *e* は下二重対角マトリックスを定義する。

smlsiz INTEGER。計算ツリーの一番下の部分問題の最大サイズ。

n INTEGER。二重対角マトリックスの次元。$n \geq 0$

nrhs INTEGER。*B* の列の数。1 以上でなければならない。

d REAL (slalsd/clalsd の場合)
DOUBLE PRECISION (dlalsd/zlalsd の場合)
配列 DIMENSION は (*n*)。*d* には二重対角マトリックスの主対角を格納する。

e REAL (slalsd/clalsd の場合)
DOUBLE PRECISION (dlalsd/zlalsd の場合)
配列 DIMENSION は (*n*-1)。入力パラメータとして、二重対角マトリックスの優対角成分を格納する。ルーチン終了時に内容は破壊される。

b REAL (slalsd の場合)
DOUBLE PRECISION (dlalsd の場合)
COMPLEX (clalsd の場合)
COMPLEX*16 (zlalsd の場合)

配列 DIMENSION は (*ldb*, *nrhs*)。入力パラメータとして *b* には最小二乗問題の右辺を格納する。出力パラメータとして *b* には解 *X* が格納される。

ldb
INTEGER。呼び出し元のサブプログラムの *b* のリーディング・ディメンジョン。max(1, *n*) 以上でなければならない。

rcond
REAL (slalsd/clalsd の場合)
DOUBLE PRECISION (dlalsd/zlalsd の場合)
最大特異値の *rcond* 倍より小さいか等しい *A* の特異値は、最小二乗問題を解く過程でゼロとして扱われる。
rcond が負の場合はマシン精度が代わりに使用される。
たとえば、diag(*S*)**X*=*B* が最小二乗問題で diag(*S*) が特異値の二重対角マトリックスの場合、解は、S(*i*) が *rcond* *max(*S*) より大きい場合は *X*(*i*) = *B*(*i*) / *S*(*i*) if *S*(*i*) に、*S*(*i*) が *rcond* *max(*S*) より小さいか等しい場合は *X*(*i*) = 0 になる。

rank
INTEGER。最大特異値の *rcond* 倍より大きい *A* の特異値の個数。

work
REAL (slalsd の場合)
DOUBLE PRECISION (dlalsd の場合)
COMPLEX (clalsd の場合)
COMPLEX*16 (zlalsd の場合)
ワークスペース配列。
実数型の DIMENSION で少なくとも、
$(9n+2n*smlsiz+8n*nlvl+n*nrhs+(smlsiz+1)^2)$、
ここで
$nlvl = \max(0, \mathrm{int}(\log_2(n / (smlsiz+1))) + 1)$
複素型の DIMENSION で少なくとも (n*nrhs)

rwork
REAL (clalsd の場合)
DOUBLE PRECISION (zlalsd の場合)
ワークスペース配列で複素型でのみ使用される。
DIMENSION は少なくとも $(9n + 2n*smlsiz + 8n*nlvl + 3*mlsiz*nrhs + (smlsiz+1)^2)$、
ここで
$nlvl = \max(0, \mathrm{int}(\log_2(\min(m, n)/(smlsiz+1))) + 1)$

iwork INTEGER
ワークスペース配列 DIMENSION は少なくとも
(3*n***nlvl* + 11*n*)

出力パラメータ

d *info* = 0 の場合、*d* には二重対角マトリックスの特異値が格納される。

b *b* には解 *X* が格納される。

info INTEGER
info = 0 の場合、正常に終了したことを示す。
info = -*i* < 0 の場合、*i* 番目の引数の値が不正だったことを示す。
info = *i* > 0 の場合、*info*/(*n*+1) から mod(*info*, *n*+1) までの行と列にある部分マトリックスの操作中に、アルゴリズムが特異値の計算に失敗したことを示す。

?lamch

浮動小数点演算に対するマシン・パラメータを決定する。

```
val = slamch ( cmach )
val = dlamch ( cmach )
```

説明

関数 ?lamch は、単精度および倍精度のマシン・パラメータを決定する。

入力パラメータ

cmach CHARACTER*1。?lamch が返す値を指定する。
= '*E*' または '*e*', *val* = *eps*
= '*S*' または '*s* , *val* = *sfmin*
= '*B*' または '*b*', *val* = *base*
= '*P*' または '*p*', *val* = *eps***base*

= 'N' または 'n', val = t
= 'R' または 'r', val = rnd
= 'M' または 'm', val = emin
= 'U' または 'u', val = rmin
= 'L' または 'l', val = emax
= 'O' または 'o', val = rmax
ここで
eps = 相対マシン精度。
sfmin = 1/sfmin がオーバーフローしないような安全な最小値。
base = マシンの基数。
prec = eps*base
t = 仮数における (基数) 桁数。
rnd = 丸めが追加で発生した場合 1.0、それ以外では 0.0
emin = (緩やかに) アンダーフローを起こす前の最小指数。
rmin = *underflow_threshold* - base**(emin-1)
emax = オーバーフローを起こす前の最大指数
rmax = *overflow_threshold* - (base**emax)*(1-eps)

出力パラメータ

val　REAL (slamch の場合)
DOUBLE PRECISION (dlamch の場合)
関数の戻り値。

?lamc1

?lamc2 から呼び出される。beta、t、rnd、ieee1 で与えられるマシン・パラメータを決定する。

```
call slamc1 ( beta, t, rnd, ieee1 )
call dlamc1 ( beta, t, rnd, ieee1 )
```

説明

ルーチン ?lamc1 は *beta*、*t*、*rnd*、*ieee1* で与えられるマシン・パラメータを決定する。

出力パラメータ

beta INTEGER。マシンの基数。

t INTEGER。仮数における (*beta*) 桁数。

rnd LOGICAL。
適切な丸め (*rnd* = .TRUE.) または切捨て (*rnd* = .FALSE.) が追加で発生したかを示す。マシンの演算性能を決める信頼性のある指標ではない。

ieee1 LOGICAL。
ieee の「round to nearest」形式で丸めが発生したかどうかを示す。

?lamc2

?lamch で使用される。引数リストで指定されたマシン・パラメータを決定する。

```
call slamc2 ( beta, t, rnd, eps, emin, rmin, emax, rmax )
call dlamc2 ( beta, t, rnd, eps, emin, rmin, emax, rmax )
```

説明

ルーチン ?lamc2 は引数リストで指定されたマシン・パラメータを決定する。

出力パラメータ

beta INTEGER。マシンの基数。

t INTEGER。仮数における (*beta*) 桁数。

rnd	LOGICAL。 適切な丸め (*rnd* = .TRUE.) または切捨て (*rnd* = .FALSE.) が追加で発生したかを示す。マシンの演算性能を決める信頼性のある指標ではない。
eps	REAL (slamc2 の場合) DOUBLE PRECISION (dlamc2 の場合) 以下を満たすような最小の正の値。 *fl*(1.0 - *eps*) < 1.0、 *fl* は計算された値を示す。
emin	INTEGER。(緩やかに) アンダーフローを起こす前の最小指数。
rmin	REAL (slamc2 の場合) DOUBLE PRECISION (dlamc2 の場合) マシンに対する最小の正規化された数で、 $base^{emin-1}$ で与えられ、*base* は *beta* の浮動小数点値である。
emax	INTEGER。オーバーフローを起こす前の最大指数
rmax	REAL (slamc2 の場合) DOUBLE PRECISION (dlamc2 の場合) マシンに対する最大の正の数で、$base^{emax}(1 - eps)$ で与えられ、*base* は *beta* の浮動小数点値である。

?lamc3

?lamc1 ～ ?lamc5 から呼び出される。
a と *b* の加算を実行する前に *a* と *b* を格納させる。

```
val = slamc3 (a, b)
val = dlamc3 (a, b)
```

説明

このルーチンは、*a* と *b* の加算を実行する前に *a* と *b* を格納させる。オプティマイザがこれらの値をレジスタに保持している場合に使用する。

入力パラメータ

a, b
REAL (slamc3 の場合)
DOUBLE PRECISION (dlamc3 の場合)
a と *b* の値。

出力パラメータ

val
REAL (slamc3 の場合)
DOUBLE PRECISION (dlamc3 の場合)
a と *b* を加算した結果。

?lamc4

?lamc2 のサービス・ルーチンである。

```
call slamc4 (emin, start, base)
call dlamc4 (emin, start, base)
```

説明

このルーチンは ?lamc2 のサービス・ルーチンである。

入力パラメータ

start
REAL (slamc4 の場合)
DOUBLE PRECISION (dlamc4 の場合)
emin を決定する開始点。

base
INTEGER。マシンの基数。

出力パラメータ

emin INTEGER。(緩やかに) アンダーフローを起こす前の最小指数で、*a* = *start* と設定し、前の *a* が回復できなくなるまで基数による除算を行って計算される。

?lamc5

?lamc2 から呼び出される。
オーバーフローしないマシンの最大浮動小数点値の計算を試みる。

```
call slamc5 ( beta, p, emin, ieee, emax, rmax )
call dlamc5 ( beta, p, emin, ieee, emax, rmax )
```

説明

ルーチン ?lamc5 は、オーバーフローしない最大マシン浮動小数点値 *rmax* の計算を試みる。*emax* + abs(*emin*) の合計はおよそ 2 のべき乗であると仮定する。たとえば、Cyber 205 (*emin* = -28625, *emax* = 28718) のように、この仮定が満たされない場合、マシンでの実行は失敗する。*emin* にオーバーフローするであろう大きすぎる値 (すなわちゼロに近い) を与えても同様にマシンでの実行は失敗する。

入力パラメータ

beta INTEGER。浮動小数点演算の基数。

p INTEGER。浮動小数点演算の仮数における基数 *beta* の桁数。

emin INTEGER。(緩やかに) アンダーフローを起こす前の最小指数。

ieee LOGICAL。演算システムが IEEE スタンダードに準拠していると考えられるかどうかを示す論理フラグ。

出力パラメータ

emax INTEGER。オーバーフローを起こす前の最大指数。

rmax REAL (slamc5 の場合)
DOUBLE PRECISION (dlamc5 の場合)
最大マシン浮動小数点値。

?lamrg

2 つの独立するソートされたセットの
成分を単一のソートされたセットに昇順で
併合する置換リストを生成する。

```
call slamrg ( n1, n2, a, strd1, strd2, index )
call dlamrg ( n1, n2, a, strd1, strd2, index )
```

説明

このルーチンは、*a* (2 つの独立したソートされたセットで構成) の成分を単一のソートされたセットに昇順で併合する置換リストを生成する。

入力パラメータ

n1, n2 INTEGER
これら引数には、併合対象となる 2 つのソートされたリストのそれぞれの長さを格納する。

a REAL (slamrg の場合)
DOUBLE PRECISION (dlamrg の場合)
配列 DIMENSION は (*n1*+*n2*)
a の最初の *n1* 個の成分には、昇順または降順のどちらかでソートされる数値のリストを格納する。最後の *n2* 個の成分も同様である。

strd1, strd2 INTEGER
配列 *a* を介して取られるストライドである。可能なストライドは 1 と -1 である。これらは *a* の部分集合が昇順 (*strdx* = 1)、または降順 (*strdx* = -1) でソートされることを示す。

出力パラメータ

index INTEGER
配列 DIMENSION は (*n1*+*n2*)
i = 1, *n1*+*n2* に対して *b*(*i*) = *a*(*index*(*i*)) ならば、この配列には置換が格納され、*b* は昇順でソートされる。

?langb

一般バンド・マトリックスの 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。

```
val = slangb ( norm, n, kl, ku, ab, ldab, work )
val = dlangb ( norm, n, kl, ku, ab, ldab, work )
val = clangb ( norm, n, kl, ku, ab, ldab, work )
val = zlangb ( norm, n, kl, ku, ab, ldab, work )
```

説明

この関数は、*kl* 個の劣対角と *ku* 個の優対角を持つ $n \times n$ バンド・マトリックス A の、1- ノルムの値、または Frobenius ノルムの値、または無限ノルムの値、または最大絶対値の成分を返す。

関数の戻り値は *val* に格納される。

val = max(abs(A_{ij}))、 *norm* = 'M' または 'm' の場合

= norm1(A)、 *norm* = '1' または 'O' または 'o' の場合

= normI(A)、 *norm* = 'I' または 'i' の場合

= normF(A)、 *norm* = 'F', 'f', 'E' または 'e' の場合

norm1 はマトリックスの 1- ノルム (最大列合計)、normI はマトリックスの無限ノルム (最大行合計)、normF はマトリックスの Frobenius ノルム (二乗和の平方根) を示す。max(abs(A_{ij})) はマトリックス・ノルムではない点に注意する。

入力パラメータ

norm CHARACTER*1。ルーチンが返す値を上述のように指定する。

n INTEGER。マトリックス A の次数。
$n \geq 0$。$n = 0$ と指定した場合、?langb はゼロに設定される。

kl INTEGER。マトリックス A の劣対角成分の数。$kl \geq 0$

ku INTEGER。マトリックス A の優対角成分の数。$ku \geq 0$

ab REAL (slangb の場合)
DOUBLE PRECISION (dlangb の場合)
COMPLEX (clangb の場合)
COMPLEX*16 (zlangb の場合)
配列 DIMENSION は (*ldab*, *n*)。バンド・マトリックス A で、行 1 から $kl+ku+1$ に格納する。A の j 番目の列は配列 *ab* の j 番目の列に次のように格納する。
$ab(ku+1+i-j, j) = a(i, j)$
$\max(1, j-ku) \leq i \leq \min(n, j+kl)$ の場合。

ldab INTEGER。配列 *ab* のリーディング・ディメンジョン。
$ldab \geq kl+ku+1$

work REAL (slangb/clangb の場合)
DOUBLE PRECISION (dlangb/zlangb の場合)
ワークスペース配列で DIMENSION (*lwork*)
norm = '*i*' のとき $lwork \geq n$。それ以外では *work* は参照されない。

出力パラメータ

val REAL (slangb/clangb の場合)
DOUBLE PRECISION (dlangb/zlangb の場合)
関数の戻り値。

?lange

一般矩形マトリックスの 1- ノルムの値、
Frobenius ノルムの値、無限ノルムの値、
あるいは最大絶対値の成分を返す。

```
val = slange ( norm, m, n, a, lda, work )
val = dlange ( norm, m, n, a, lda, work )
val = clange ( norm, m, n, a, lda, work )
val = zlange ( norm, m, n, a, lda, work )
```

説明

関数 ?lange は、実数 / 複素マトリックス A の、1- ノルムの値、または Frobenius ノルムの値、または無限ノルムの値、または最大絶対値の成分を返す。

関数の戻り値は *val* に格納される。

val $= \max(\mathrm{abs}(A_{ij}))$、 *norm* = 'M' または 'm' の場合

$=$ norm1(A)、 *norm* = '1' または 'O' または 'o' の場合

$=$ normI(A)、 *norm* = 'I' または 'i' の場合

$=$ normF(A)、 *norm* = 'F', 'f', 'E' または 'e' の場合

norm1 はマトリックスの 1- ノルム (最大列合計)、normI はマトリックスの無限ノルム (最大行合計)、normF はマトリックスの Frobenius ノルム (二乗和の平方根) を示す。$\max(\mathrm{abs}(A_{ij}))$ はマトリックス・ノルムではない点に注意する。

入力パラメータ

norm CHARACTER*1。ルーチン ?lange が返す値を上述のように指定する。

m INTEGER。マトリックス A の行数
$m \geq 0$。$m = 0$ と指定した場合、?lange はゼロに設定される。

n INTEGER。マトリックス A の列数。
$n \geq 0$。$n = 0$ と指定した場合、?lange はゼロに設定される。

a REAL (slange の場合)
DOUBLE PRECISION (dlange の場合)
COMPLEX (clange の場合)
COMPLEX*16 (zlange の場合)
配列 DIMENSION は (*lda*, *n*)。$m \times n$ のマトリックス A

lda INTEGER。配列 *a* のリーディング・ディメンジョン。
$lda \geq \max(m, 1)$

work REAL (slange と clange の場合)
DOUBLE PRECISION (dlange と zlange の場合)
ワークスペース配列で DIMENSION (*lwork*)
norm = 'I' のとき $lwork \geq m$。それ以外では *work* は参照されない。

出力パラメータ

val REAL (slange/clange の場合)
DOUBLE PRECISION (dlange/zlange の場合)
関数の戻り値。

?langt

一般三重対角マトリックスの 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。

```
val = slangt ( norm, n, dl, d, du )
val = dlangt ( norm, n, dl, d, du )
val = clangt ( norm, n, dl, d, du )
val = zlangt ( norm, n, dl, d, du )
```

説明

このルーチンは、実数 / 複素三重対角マトリックス *A* の、1- ノルムの値、または Frobenius ノルムの値、または無限ノルムの値、または最大絶対値の成分を返す。

関数の戻り値は val に格納される。

val = max(abs(A_{ij}))、 *norm* = 'M' または 'm' の場合

= norm1(*A*)、 *norm* = '1' または 'O' または 'o' の場合

= normI(*A*)、 *norm* = 'I' または 'i' の場合

= normF(*A*)、 *norm* = 'F', 'f', 'E' または 'e' の場合

norm1 はマトリックスの 1- ノルム (最大列合計)、normI はマトリックスの無限ノルム (最大行合計)、normF はマトリックスの Frobenius ノルム (二乗和の平方根) を示す。max(abs(A_{ij})) はマトリックス・ノルムではない点に注意する。

入力パラメータ

norm CHARACTER*1。ルーチン ?langt が返す値を上述のように指定する。

n INTEGER。マトリックス *A* の次数。
$n \geq 0$。$n = 0$ と指定した場合、?langt はゼロに設定される。

dl, d, du REAL (slangt の場合)
DOUBLE PRECISION (dlangt の場合)
COMPLEX (clangt の場合)
COMPLEX*16 (zlangt の場合)
配列 : *dl* (*n*-1), *d* (*n*), *du* (*n*-1)
配列 *dl* には *A* の (*n*-1) 個の劣対角成分を格納する。
配列 *d* には *A* の対角成分を格納する。
配列 *du* には *A* の (*n*-1) 個の優対角成分を格納する。

出力パラメータ

val REAL (slangt/clangt の場合)
DOUBLE PRECISION (dlangt/zlangt の場合)
関数の戻り値。

?lanhs

上 Hessenberg マトリックスの 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。

```
val = slanhs ( norm, n, a, lda, work )
val = dlanhs ( norm, n, a, lda, work )
val = clanhs ( norm, n, a, lda, work )
val = zlanhs ( norm, n, a, lda, work )
```

説明

関数 ?lanhs は、Hessenberg マトリックス A の、1- ノルムの値、または Frobenius ノルムの値、または無限ノルムの値、または最大絶対値の成分を返す。

関数の戻り値は *val* に格納される。

val = max(abs(A_{ij}))、 *norm* = 'M' または 'm' の場合

= norm1(A)、 *norm* = '1' または 'O' または 'o' の場合

= normI(A)、 *norm* = 'I' または 'i' の場合

= normF (A)、 *norm* = 'F', 'f', 'E' または 'e')

norm1 はマトリックスの 1- ノルム (最大列合計)、normI はマトリックスの無限ノルム (最大行合計)、normF はマトリックスの Frobenius ノルム (二乗和の平方根) を示す。max(abs(A_{ij})) はマトリックス・ノルムではない点に注意する。

入力パラメータ

norm CHARACTER*1。ルーチン ? lanhs が返す値を上述のように指定する。

n INTEGER。マトリックス A の次数。
$n \geq 0$。$n = 0$ と指定した場合、?lanhs はゼロに設定される。

a REAL (slanhs の場合)
DOUBLE PRECISION (dlanhs の場合)
COMPLEX (clanhs の場合)
COMPLEX*16 (zlanhs の場合)
配列 DIMENSION は (*lda*, *n*)。$n \times n$ の上 Hessenberg マトリックス A。A のうち最初の劣対角から下の部分は参照されない。

lda INTEGER。配列 *a* のリーディング・ディメンジョン。
$lda \geq \max(n, 1)$

work REAL (slanhs と clanhs の場合)
DOUBLE PRECISION (dlange と zlange の場合)
ワークスペース配列で DIMENSION (*lwork*)
norm = 'I' のとき $lwork \geq n$。それ以外では *work* は参照されない。

出力パラメータ

val REAL (slanhs/clanhs の場合)
DOUBLE PRECISION (dlanhs/zlanhs の場合)
関数の戻り値。

?lansb

対称バンド・マトリックスの 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。

```
val = slansb ( norm, uplo, n, k, ab, ldab, work )
val = dlansb ( norm, uplo, n, k, ab, ldab, work )
val = clansb ( norm, uplo, n, k, ab, ldab, work )
val = zlansb ( norm, uplo, n, k, ab, ldab, work )
```

説明

関数 ?lansb は、*k* 個の優対角を持つ *n* × *n* 実数 / 複素対称バンド・マトリックス *A* の、1- ノルムの値、または Frobenius ノルムの値、または無限ノルムの値、または最大絶対値の成分を返す。

関数の戻り値は *val* に格納される。

val = max(abs(A_{ij}))、 *norm* = 'M' または 'm' の場合

= norm1(*A*)、 *norm* = '1' または 'O' または 'o' の場合

= normI(*A*)、 *norm* = 'I' または 'i' の場合

= normF(*A*)、 *norm* = 'F', 'f', 'E' または 'e' の場合

norm1 はマトリックスの 1- ノルム (最大列合計)、normI はマトリックスの無限ノルム (最大行合計)、normF はマトリックスの Frobenius ノルム (二乗和の平方根) を示す。max(abs(A_{ij})) はマトリックス・ノルムではない点に注意する。

入力パラメータ

norm CHARACTER*1。ルーチン ? lansb が返す値を上述のように指定する。

uplo CHARACTER*1。バンド・マトリックス *A* の上三角部分または下三角部分のどちらを与えるか指定する。
uplo = 'U' の場合、上三角部分を与える。
uplo = 'L' の場合、下三角部分を与える。

n INTEGER。マトリックス *A* の次数。*n* ≥ 0
n = 0 と指定した場合、?lansb はゼロに設定される。

k INTEGER。バンド・マトリックス *A* の優対角または劣対角の個数。*k* ≥ 0

ab REAL (slansb の場合)
DOUBLE PRECISION (dlansb の場合)
COMPLEX (clansb の場合)
COMPLEX*16 (zlansb の場合)
配列 DIMENSION は (*ldab*, *n*)。対称バンド・マトリックス *A* の上三角または下三角で、*ab* の最初の *k*+1 行に格納する。

A の j 番目の列は配列 ab の j 番目の列に次のように格納する。
uplo = 'U' の場合、max(1, j-k) ≤ i ≤ j に対して ab(k+1+i-j, j) = a(i, j)
uplo = 'L' の場合、j ≤ i ≤ min(n, j+k) に対して ab(1+i-j, j) = a(i, j)

ldab　INTEGER。配列 ab のリーディング・ディメンジョン。ldab ≥ k+1

work　REAL (slansb と clansb の場合)
DOUBLE PRECISION (dlansb と zlansb の場合)
ワークスペース配列で DIMENSION (lwork)
norm = 'I' または '1' または 'O' のとき lwork ≥ n。それ以外では work は参照されない。

出力パラメータ

val　REAL (slansb/clansb の場合)
DOUBLE PRECISION (dlansb/zlansb の場合)
関数の戻り値。

?lanhb

エルミート・バンド・マトリックスの 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。

```
val = clanhb ( norm, uplo, n, k, ab, ldab, work )
val = zlanhb ( norm, uplo, n, k, ab, ldab, work )
```

説明

このルーチンは、k 個の優対角を持つ n × n エルミート・バンド・マトリックス A の、1- ノルムの値、または Frobenius ノルムの値、または無限ノルムの値、または最大絶対値の成分を返す。

関数の戻り値は val に格納される。

val = max(abs(A_{ij}))、 *norm* = 'M' または 'm' の場合

= norm1(A)、 *norm* = '1' または 'O' または 'o' の場合

= normI(A)、 *norm* = 'I' または 'i' の場合

= normF(A)、 *norm* = 'F', 'f', 'E' または 'e' の場合

norm1 はマトリックスの 1- ノルム (最大列合計)、normI はマトリックスの無限ノルム (最大行合計)、normF はマトリックスの Frobenius ノルム (二乗和の平方根) を示す。max(abs(A_{ij})) はマトリックス・ノルムではない点に注意する。

入力パラメータ

norm CHARACTER*1。ルーチン ?lanhb が返す値を上述のように指定する。

uplo CHARACTER*1。バンド・マトリックス A の上三角部分または下三角部分のどちらを与えるか指定する。
uplo = 'U' の場合、上三角部分を与える。
uplo = 'L' の場合、下三角部分を与える。

n INTEGER。マトリックス A の次数。$n \geq 0$
$n = 0$ と指定した場合、?lanhb はゼロに設定される。

k INTEGER。バンド・マトリックス A の優対角または劣対角の個数。$k \geq 0$

ab COMPLEX (clanhb の場合)
COMPLEX*16 (zlanhb の場合)
配列 DIMENSION は (*ldab*, *n*)。エルミート・バンド・マトリックス A の上三角または下三角で、*ab* の最初の *k*+1 行に格納する。
A の *j* 番目の列は配列 *ab* の *j* 番目の列に次のように格納する。
uplo = 'U' の場合、$\max(1, j-k) \leq i \leq j$ に対して
$ab(k+1+i-j, j) = a(i, j)$
uplo = 'L' の場合、$j \leq i \leq \min(n, j+k)$ に対して
$ab(1+i-j, j) = a(i, j)$
対角成分の虚数部分は設定する必要はなくゼロとして仮定される点に注意する。

ldab INTEGER。配列 *ab* のリーディング・ディメンジョン。
$ldab \geq k+1$

work REAL (clanhb の場合)
DOUBLE PRECISION (zlanhb の場合)
ワークスペース配列で DIMENSION (*lwork*)
norm = 'I' または '1' または 'O' のとき *lwork* ≥ *n*。それ以外では *work* は参照されない。

出力パラメータ

val REAL (slanhb/clanhb の場合)
DOUBLE PRECISION (dlanhb/zlanhb の場合)
関数の戻り値。

?lansp

パック形式で与えられる対称マトリックスの 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。

```
val = slansp ( norm, uplo, n, ap, work )
val = dlansp ( norm, uplo, n, ap, work )
val = clansp ( norm, uplo, n, ap, work )
val = zlansp ( norm, uplo, n, ap, work )
```

説明

関数 ?lansp は、パック形式で与えられる実数 / 複素対称マトリックス *A* の、1- ノルムの値、または Frobenius ノルムの値、または無限ノルムの値、または最大絶対値の成分を返す。

関数の戻り値は *val* に格納される。

val = max(abs(A_{ij}))、 *norm* = 'M' または 'm' の場合
= norm1(*A*)、 *norm* = '1' または 'O' または 'o' の場合
= normI(*A*)、 *norm* = 'I' または 'i' の場合

= normF(A)、　　*norm* = 'F', 'f', 'E' または 'e' の場合

norm1 はマトリックスの 1- ノルム (最大列合計)、normI はマトリックスの無限ノルム (最大行合計)、normF はマトリックスの Frobenius ノルム (二乗和の平方根) を示す。max(abs(A_{ij})) はマトリックス・ノルムではない点に注意する。

入力パラメータ

norm　　CHARACTER*1。ルーチン ?lansp が返す値を上述のように指定する。

uplo　　CHARACTER*1。対称マトリックス A の上三角部分または下三角部分のどちらを与えるか指定する。
uplo = 'U' の場合、A の上三角部分を与える。
uplo = 'L' の場合、A の下三角部分を与える。

n　　INTEGER。マトリックス A の次数。$n \geq 0$。$n = 0$ と指定した場合、?lansp はゼロに設定される。

ap　　REAL (slansp の場合)
DOUBLE PRECISION (dlansp の場合)
COMPLEX (clansp の場合)
COMPLEX*16 (zlansp の場合)
配列 DIMENSION は (n(n+1)/2)。線形配列に列方向にパックされた対称マトリックス A の上三角または下三角。A の j 番目の列は配列 *ap* に次のように格納する。
uplo = 'U' の場合、$1 \leq i \leq j$ に対して
$ap(i+(j-1)j/2) = A(i, j)$
uplo = 'L' の場合、$j \leq i \leq n$ に対して
$ap(i+(j-1)(2n-j)/2) = A(i, j)$

work　　REAL (slansp と clansp の場合)
DOUBLE PRECISION (dlansp と zlansp の場合)
ワークスペース配列で DIMENSION (*lwork*)
norm = 'I' または '1' または 'O' のとき $lwork \geq n$。それ以外では *work* は参照されない。

出力パラメータ

val　　REAL (slansp/clansp の場合)
DOUBLE PRECISION (dlansp/zlansp の場合)
関数の戻り値。

?lanhp

パック形式で与えられる複素エルミート・マトリックスの 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。

```
val = clanhp ( norm, uplo, n, ap, work )
val = zlanhp ( norm, uplo, n, ap, work )
```

説明

関数 ?lanhp は、パック形式で与えられる複素エルミート・マトリックス A の、1- ノルムの値、または Frobenius ノルムの値、または無限ノルムの値、または最大絶対値の成分を返す。

関数の戻り値は *val* に格納される。

val = max(abs(A_{ij}))、 *norm* = 'M' または 'm' の場合

= norm1(A)、 *norm* = '1' または 'O' または 'o' の場合

= norm1(A)、 *norm* = 'I' または 'i' の場合

= norm1(A)、 *norm* = 'F', 'f', 'E' または 'e' の場合

norm1 はマトリックスの 1- ノルム (最大列合計)、normI はマトリックスの無限ノルム (最大行合計)、normF はマトリックスの Frobenius ノルム (二乗和の平方根) を示す。max(abs(A_{ij})) はマトリックス・ノルムではない点に注意する。

入力パラメータ

norm CHARACTER*1。ルーチン ?lanhp が返す値を上述のように指定する。

uplo CHARACTER*1。エルミート・マトリックス A の上三角部分または下三角部分のどちらを与えるか指定する。
uplo = 'U' の場合、A の上三角部分を与える。
uplo = 'L' の場合、A の下三角部分を与える。

n INTEGER。マトリックス *A* の次数。
$n \geq 0$。$n = 0$ と指定した場合、?lanhp はゼロに設定される。

ap COMPLEX (clanhp の場合)
COMPLEX*16 (zlanhp の場合)
配列 DIMENSION は $(n(n+1)/2)$。線形配列に列方向にパックされたエルミート・マトリックス *A* の上三角または下三角。*A* の *j* 番目の列は配列 *ap* に次のように格納する。
uplo = 'U' の場合、$1 \leq i \leq j$ に対して
$ap(i + (j-1)j/2) = A(i, j)$
uplo = 'L' の場合、$j \leq i \leq n$ に対して
$ap(i + (j-1)(2n-j)/2) = A(i, j)$

work REAL (clanhp の場合)
DOUBLE PRECISION (zlanhp の場合)
ワークスペース配列で DIMENSION (*lwork*)
norm = 'I' または '1' または 'O' のとき $lwork \geq n$。それ以外では *work* は参照されない。

出力パラメータ

val REAL (clanhp の場合)
DOUBLE PRECISION (zlanhp の場合)
関数の戻り値。

?lanst/?lanht

実数対称または複素エルミート三重対角マトリックスの 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。

```
val = slanst ( norm, n, d, e )
val = dlanst ( norm, n, d, e )
val = clanht ( norm, n, d, e )
val = zlanht ( norm, n, d, e )
```

説明

関数 ?lanst/?lanht は、実数対称または複素エルミート三重対角マトリックス A の、1- ノルムの値、または Frobenius ノルムの値、または無限ノルムの値、または最大絶対値の成分を返す。

関数の戻り値は *val* に格納される。

val = max(abs(A_{ij}))、 *norm* = 'M' または 'm' の場合

= norm1(A)、 *norm* = '1' または 'O' または 'o' の場合

= norm1(A)、 *norm* = 'I' または 'i' の場合

= norm1(A)、 *norm* = 'F', 'f', 'E' または 'e' の場合

norm1 はマトリックスの 1- ノルム (最大列合計)、normI はマトリックスの無限ノルム (最大行合計)、normF はマトリックスの Frobenius ノルム (二乗和の平方根) を示す。max(abs(A_{ij})) はマトリックス・ノルムではない点に注意する。

入力パラメータ

norm CHARACTER*1。ルーチン ?lanst/?lanht が返す値を上述のように指定する。

n INTEGER。マトリックス A の次数。
$n \geq 0$。$n = 0$ と指定した場合、?lanst/?lanht はゼロに設定される。

d REAL (slanst/clanht の場合)
DOUBLE PRECISION (dlanst/zlanht の場合)
配列 DIMENSION は (n)。A の対角成分。

e REAL (slanst の場合)
DOUBLE PRECISION (dlanst の場合)
COMPLEX (clanht の場合)
COMPLEX*16 (zlanht の場合)
配列 DIMENSION は (n-1)。A の (n-1) 個の劣対角成分または優対角成分。

出力パラメータ

val REAL (slanst/clanht の場合)
DOUBLE PRECISION (dlanst/zlanht の場合)
関数の戻り値。

?lansy

実数 / 複素対称マトリックスの 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。

```
val = slansy ( norm, uplo, n, a, lda, work )
val = dlansy ( norm, uplo, n, a, lda, work )
val = clansy ( norm, uplo, n, a, lda, work )
val = zlansy ( norm, uplo, n, a, lda, work )
```

説明

関数 ?lansy は、実数 / 複素対称マトリックス A の、1- ノルムの値、または Frobenius ノルムの値、または無限ノルムの値、または最大絶対値の成分を返す。

関数の戻り値は *val* に格納される。

val = max(abs(A_{ij}))、 *norm* = 'M' または 'm' の場合

= norm1(A)、 *norm* = '1' または 'O' または 'o' の場合

= normI(A)、 *norm* = 'I' または 'i' の場合

= normF(A)、 *norm* = 'F', 'f', 'E' または 'e' の場合

norm1 はマトリックスの 1- ノルム (最大列合計)、normI はマトリックスの無限ノルム (最大行合計)、normF はマトリックスの Frobenius ノルム (二乗和の平方根) を示す。max(abs(A_{ij})) はマトリックス・ノルムではない点に注意する。

入力パラメータ

norm CHARACTER*1。ルーチン ?lansy が返す値を上述のように指定する。

uplo CHARACTER*1。対称マトリックス *A* の上三角部分と下三角部分のどちらを参照させるか指定する。
= 'U' の場合、*A* の上三角部分を参照させる。
= 'L' の場合、*A* の下三角部分を参照させる。

n INTEGER。マトリックス *A* の次数。$n \geq 0$
$n = 0$ と指定した場合、?lansy はゼロに設定される。

a REAL (slansy の場合)
DOUBLE PRECISION (dlansy の場合)
COMPLEX (clansy の場合)
COMPLEX*16 (zlansy の場合)
配列 DIMENSION は (*lda*, *n*)。対称マトリックス *A*。
uplo = 'U' の場合、*a* の先頭の $n \times n$ 上三角部分にマトリックス *A* の上三角部分を格納する。厳密な下三角部分は参照されない。
uplo = 'L' の場合、*a* の先頭の $n \times n$ 下三角部分にマトリックス *A* の下三角部分を格納する。厳密な上三角部分は参照されない。

lda INTEGER。配列 *a* のリーディング・ディメンジョン。
$lda \geq \max(n, 1)$

work REAL (slansy と clansy の場合)
DOUBLE PRECISION (dlansy と lansy の場合)
ワークスペース配列で DIMENSION (*lwork*)
norm = 'I' または '1' または 'O' のとき $lwork \geq n$。それ以外では *work* は参照されない。

出力パラメータ

val REAL (slansy/clansy の場合)
DOUBLE PRECISION (dlansy/zlansy の場合)
関数の戻り値。

?lanhe

複素エルミート・マトリックスの
1- ノルムの値、Frobenius ノルムの値、
無限ノルムの値、あるいは最大絶対値の
成分を返す。

```
val = clanhe ( norm, uplo, n, a, lda, work )
val = zlanhe ( norm, uplo, n, a, lda, work )
```

説明

関数 ?lanhe は、複素エルミート・マトリックス A の、1- ノルムの値、または Frobenius ノルムの値、または無限ノルムの値、または最大絶対値の成分を返す。

関数の戻り値は *val* に格納される。

val = max(abs(A_{ij}))、 *norm* = 'M' または 'm' の場合

= norm1(A)、 *norm* = '1' または 'O' または 'o' の場合

= normI(A)、 *norm* = 'I' または 'i' の場合

= normF(A)、 *norm* = 'F', 'f', 'E' または 'e' の場合

norm1 はマトリックスの 1- ノルム (最大列合計)、normI はマトリックスの無限ノルム (最大行合計)、normF はマトリックスの Frobenius ノルム (二乗和の平方根) を示す。max(abs(A_{ij})) はマトリックス・ノルムではない点に注意する。

入力パラメータ

norm CHARACTER*1。ルーチン ?lanhe が返す値を上述のように指定する。

uplo CHARACTER*1。エルミート・マトリックス A の上三角部分と下三角部分のどちらを参照させるか指定する。
= 'U' の場合、A の上三角部分を参照させる。
= 'L' の場合、A の下三角部分を参照させる。

n INTEGER。マトリックス *A* の次数。
n ≥ 0。*n* = 0 と指定した場合、?lanhe はゼロに設定される。

a COMPLEX (clanhe の場合)
COMPLEX*16 (zlanhe の場合)
配列 DIMENSION は (*lda*, *n*)。エルミート・マトリックス *A*。
uplo = 'U' の場合、*a* の先頭の *n* × *n* 上三角部分にマトリックス *A* の上三角部分を格納する。厳密な下三角部分は参照されない。
uplo = 'L' の場合、*a* の先頭の *n* × *n* 下三角部分にマトリックス *A* の下三角部分を格納する。厳密な上三角部分は参照されない。

lda INTEGER。配列 *a* のリーディング・ディメンジョン。
lda ≥ max(*n*, 1)

work REAL (clanhe の場合)
DOUBLE PRECISION (zlanhe の場合)
ワークスペース配列で DIMENSION (*lwork*)
norm = 'I' または '1' または 'O' のとき *lwork* ≥ *n*。それ以外では *work* は参照されない。

出力パラメータ

val REAL (clanhe の場合)
DOUBLE PRECISION (zlanhe の場合)
関数の戻り値。

?lantb

三角バンド・マトリックスの 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。

```
val = slantb ( norm, uplo, diag, n, k, ab, ldab, work )
val = dlantb ( norm, uplo, diag, n, k, ab, ldab, work )
val = clantb ( norm, uplo, diag, n, k, ab, ldab, work )
val = zlantb ( norm, uplo, diag, n, k, ab, ldab, work )
```

説明

関数 ?lantb は、(k+1) 個の対角を持つ $n \times n$ 三角バンド・マトリックス A の、1- ノルムの値、または Frobenius ノルムの値、または無限ノルムの値、または最大絶対値の成分を返す。

関数の戻り値は *val* に格納される。

val = max(abs(A_{ij}))、 *norm* = 'M' または 'm' の場合

= norm1(A)、 *norm* = '1' または 'O' または 'o' の場合

= normI(A)、 *norm* = 'I' または 'i' の場合

= normF(A)、 *norm* = 'F', 'f', 'E' または 'e' の場合

norm1 はマトリックスの 1- ノルム (最大列合計)、normI はマトリックスの無限ノルム (最大行合計)、normF はマトリックスの Frobenius ノルム (二乗和の平方根) を示す。max(abs(A_{ij})) はマトリックス・ノルムではない点に注意する。

入力パラメータ

norm CHARACTER*1。ルーチン ?lantb が返す値を上述のように指定する。

uplo CHARACTER*1。マトリックス A が上三角か下三角かを指定する。
= 'U' の場合、上三角。
= 'L' の場合、下三角。

diag CHARACTER*1。マトリックス A が単位三角かどうかを指定する。
= 'N' の場合、単位三角ではない
= 'U' の場合、単位三角

n INTEGER。マトリックス A の次数。
$n \geq 0$。$n = 0$ と指定した場合、?lantb はゼロに設定される。

k INTEGER。*uplo* = 'U' の場合はマトリックス A の優対角成分の数、*uplo* = 'L' の場合はマトリックス A の劣対角成分の数。$k \geq 0$

ab
REAL (slantb の場合)
DOUBLE PRECISION (dlantb の場合)
COMPLEX (clantb の場合)
COMPLEX*16 (zlantb の場合)
配列 DIMENSION は (*ldab*, *n*)。上または下三角バンド・マトリックス *A* で、*ab* の最初の *k*+1 行に格納する。*A* の *j* 番目の列は配列 *ab* の *j* 番目の列に次のように格納する。
uplo = 'U' の場合、max(1, *j*-*k*) ≤ *i* ≤ *j* に対して
ab(*k*+1+*i*-*j*, *j*) = *a*(*i*, *j*)
uplo = 'L' の場合、*j* ≤ *i* ≤ min(*n*, *j*+*k*) に対して
ab(1+*i*-*j*, *j*) = *a*(*i*, *j*)
diag = 'U' の場合、マトリックス *A* の対角成分に対応する配列 *ab* の成分は参照されないが、1 として仮定される。

ldab
INTEGER。配列 *ab* のリーディング・ディメンジョン。
ldab ≥ *k*+1

work
REAL (slantb と clantb の場合)
DOUBLE PRECISION (dlantb と zlantb の場合)
ワークスペース配列で DIMENSION (*lwork*)
norm = 'I' のとき *lwork* ≥ *n*。それ以外では *work* は参照されない。

出力パラメータ

val
REAL (slantb/clantb の場合)
DOUBLE PRECISION (dlantb/zlantb の場合)
関数の戻り値。

?lantp

パック形式で与えられる三角マトリックスの 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。

```
val = slantp ( norm, uplo, diag, n, ap, work )
val = dlantp ( norm, uplo, diag, n, ap, work )
val = clantp ( norm, uplo, diag, n, ap, work )
val = zlantp ( norm, uplo, diag, n, ap, work )
```

説明

関数 ?lantp は、パック形式で与えられる三角マトリックス A の、1- ノルムの値、または Frobenius ノルムの値、または無限ノルムの値、または最大絶対値の成分を返す。関数の戻り値は *val* に格納される。

val = max(abs(A_{ij}))、 *norm* = 'M' または 'm' の場合

= norm1(A)、 *norm* = '1' または 'O' または 'o' の場合

= normI(A)、 *norm* = 'I' または 'i' の場合

= normF(A)、 *norm* = 'F', 'f', 'E' または 'e' の場合

norm1 はマトリックスの 1- ノルム (最大列合計)、normI はマトリックスの無限ノルム (最大行合計)、normF はマトリックスの Frobenius ノルム (二乗和の平方根) を示す。max(abs(A_{ij})) はマトリックス・ノルムではない点に注意する。

入力パラメータ

norm CHARACTER*1。ルーチン ?lantp が返す値を上述のように指定する。

uplo CHARACTER*1。マトリックス *A* が上三角か下三角かを指定する。
= 'U' の場合、上三角。
= 'L' の場合、下三角。

diag CHARACTER*1。マトリックス *A* が単位三角かどうかを指定する。
= 'N' の場合、単位三角ではない。
= 'U' の場合、単位三角。

n INTEGER。マトリックス *A* の次数。
$n \geq 0$。$n = 0$ と指定した場合、?lantp はゼロに設定される。

ap REAL (slantp の場合)
DOUBLE PRECISION (dlantp の場合)
COMPLEX (clantp の場合)
COMPLEX*16 (zlantp の場合)
配列 DIMENSION は $(n(n+1)/2)$。線形配列に列方向にパックされた上または下三角マトリックス *A*。*A* の *j* 番目の列は配列 *ap* に次のように格納する。
uplo = 'U' の場合、$1 \leq i \leq j$ に対して
$AP(i + (j-1)j/2) = a(i, j)$
uplo = 'L' の場合、$j \leq i \leq n$ に対して
$ap(i + (j-1)(2n-j)/2) = a(i, j)$
diag = 'U' の場合、マトリックス *A* の対角成分に対応する配列 *ap* の成分は参照されないが、1 として仮定される。

work REAL (slantp と clantp の場合)
DOUBLE PRECISION (dlantp と zlantp の場合)
ワークスペース配列で DIMENSION (*lwork*)
norm = 'I' のとき $lwork \geq n$。それ以外では *work* は参照されない。

出力パラメータ

val REAL (slantp/clantp の場合)
DOUBLE PRECISION (dlantp/zlantp の場合)
関数の戻り値。

?lantr

台形または三角マトリックスの
1- ノルムの値、Frobenius ノルムの値、
無限ノルムの値、あるいは最大絶対値の
成分を返す。

```
val = slantr ( norm, uplo, diag, m, n, a, lda, work )
val = dlantr ( norm, uplo, diag, m, n, a, lda, work )
val = clantr ( norm, uplo, diag, m, n, a, lda, work )
val = zlantr ( norm, uplo, diag, m, n, a, lda, work )
```

説明

関数 ?lantr は、台形または三角マトリックス A の、1- ノルムの値、または Frobenius ノルムの値、または無限ノルムの値、または最大絶対値の成分を返す。

関数の戻り値は *val* に格納される。

val = max(abs(A_{ij}))、 *norm* = 'M' または 'm' の場合

= norm1(A)、 *norm* = '1' または 'O' または 'o' の場合

= normI(A)、 *norm* = 'I' または 'i' の場合

= normF(A)、 *norm* = 'F', 'f', 'E' または 'e' の場合

norm1 はマトリックスの 1- ノルム (最大列合計)、normI はマトリックスの無限ノルム (最大行合計)、normF はマトリックスの Frobenius ノルム (二乗和の平方根) を示す。max(abs(A_{ij})) はマトリックス・ノルムではない点に注意する。

入力パラメータ

norm CHARACTER*1。ルーチン ?lantr が返す値を上述のように指定する。

uplo CHARACTER*1。マトリックス *A* が上台形か下台形かを指定する。
= 'U' の場合、上台形。
= 'L' の場合、下台形。
m = *n* の場合、*A* は台形ではなく三角であることに注意する。

diag CHARACTER*1。マトリックス *A* が単位対角を持っているかどうかを指定する。
= 'N' の場合、単位対角ではない。
= 'U' の場合、単位対角。

m INTEGER。マトリックス *A* の行数。
m ≥ 0、*uplo* = 'U' の場合はさらに *m* ≤ *n*
m = 0 と指定した場合、?lantr はゼロに設定される。

n INTEGER。マトリックス *A* の列数。
n ≥ 0、*uplo* = 'L' の場合はさらに *n* ≤ *m*
n = 0 と指定した場合、?lantr はゼロに設定される。

a REAL (slantr の場合)
DOUBLE PRECISION (dlantr の場合)
COMPLEX (clantr の場合)
COMPLEX*16 (zlantr の場合)
配列 DIMENSION は (*lda*, *n*)
台形マトリックス *A* (*m* = *n* の場合 *A* は三角)。
uplo = 'U' の場合、配列 *a* の先頭の *m* × *n* 上台形部分には上台形マトリックスを格納する。厳密な下三角部分は参照されない。
uplo = 'L' の場合、配列 *a* の先頭の *m* × *n* 下台形部分には下台形マトリックスを格納する。厳密な上三角部分は参照されない。
diag = 'U' の場合、*a* の対角成分は参照されず 1 と仮定される点に注意する。

lda INTEGER。配列 *a* のリーディング・ディメンジョン。
lda ≥ max(*m*, 1)

work REAL (slantr/clantrp の場合)
DOUBLE PRECISION (dlantr/zlantr の場合)
ワークスペース配列で DIMENSION (*lwork*)
norm = 'I' のとき *lwork* ≥ *m*。それ以外では *work* は参照されない。

出力パラメータ

val REAL (slantr/clantrp の場合)
DOUBLE PRECISION (dlantr/zlantr の場合)
関数の戻り値。

?lanv2

2 × 2 実数非対称マトリックスの Schur 因子分解を標準形式で計算する。

```
call slanv2 ( a, b, c, d, rt1r, rt1i, rt2r, rt2i, cs, sn )
call dlanv2 ( a, b, c, d, rt1r, rt1i, rt2r, rt2i, cs, sn )
```

説明

このルーチンは 2 × 2 実数非対称マトリックスの Schur 因子分解を標準形式で計算する。

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} cs & -sn \\ sn & cs \end{bmatrix} \begin{bmatrix} aa & bb \\ cc & dd \end{bmatrix} \begin{bmatrix} cs & sn \\ -sn & cs \end{bmatrix}$$

ここで、次のいずれかである。

1. aa と dd がマトリックスの実数固有値となるように $cc = 0$、または、
2. $aa \pm \mathrm{sqrt}(bb*cc)$ が複素共役固有値となるように $aa = dd$ と $bb*cc < 0$

このルーチンは、実数固有値の計算において桁落ち誤差のリスクを低減するために、また可能ならば abs(*rt1r*) ≥ abs(*rt2r*) を保証するために調整されている。

入力パラメータ

a, b, c, d
: REAL (slanv2 の場合)
DOUBLE PRECISION (dlanv2 の場合)
入力マトリックスの成分。

出力パラメータ

a, b, c, d
: 標準化された Schur 形式の成分で上書きされる。

rt1r, rt1i, rt2r, rt2i,
: REAL (slanv2 の場合)
DOUBLE PRECISION (dlanv2 の場合)
固有値の実数部と虚数部。固有値が複素共役ペアの場合、*rt1i* > 0

cs, sn
: REAL (slanv2 の場合)
DOUBLE PRECISION (dlanv2 の場合)
回転マトリックスのパラメータ

?lapll

2 つのベクトルの線形従属性を測定する。

```
call slapll ( n, x, incx, y, incy, ssmin )
call dlapll ( n, x, incx, y, incy, ssmin )
call clapll ( n, x, incx, y, incy, ssmin )
call zlapll ( n, x, incx, y, incy, ssmin )
```

説明

長さ *n* の 2 つの列ベクトル *x* と *y* が与えられ、A = (*x y*) を *n* × 2 マトリックスとする。

ルーチン ?lapll は最初に A の QR 因子分解を $A = QR$ として計算し、次に 2×2 上三角マトリックス R の SVD を計算する。R の小さい方の特異値は *ssmin* で返され、ベクトル x と y の線形従属性の測定結果として使用される。

入力パラメータ

n INTEGER。ベクトル x およびベクトル y の長さ。

x REAL (slapll の場合)
DOUBLE PRECISION (dlapll の場合)
COMPLEX (clapll の場合)
COMPLEX*16 (zlapll の場合)
配列 DIMENSION は (1+(*n*-1)*incx*)
x には *n*- ベクトル x を格納する。

y REAL (slapll の場合)
DOUBLE PRECISION (dlapll の場合)
COMPLEX (clapll の場合)
COMPLEX*16 (zlapll の場合)
配列 DIMENSION は (1+(*n*-1)*incy*)。*y* には *n*- ベクトル y を格納する。

incx INTEGER。x の連続する成分間の増分で、*incx* > 0

incy INTEGER。y の連続する成分間の増分で、*incy* > 0

出力パラメータ

x *x* は上書きされる。

y *y* は上書きされる。

ssmin REAL (slapll/clapll の場合)
DOUBLE PRECISION (dlapll/zlapll の場合)
n × 2 のマトリックス $A = (x\ y)$ の最小固有値。

?lapmt

マトリックスの列に対して順方向または
逆方向置換を実行する。

```
call slapmt ( forwrd, m, n, x, ldx, k )
call dlapmt ( forwrd, m, n, x, ldx, k )
call clapmt ( forwrd, m, n, x, ldx, k )
call zlapmt ( forwrd, m, n, x, ldx, k )
```

説明

ルーチン ?lapmt は、整数 1, ..., *n* の置換 *k*(1), *k*(2), ..., k(*n*) で指定されたとおりに、*m* × *n* マトリックス *X* の列を再配置する。

forwrd = .TRUE. の場合、順方向置換である。

j = 1, 2, ..., *n* では、*X*(*, *k*(*j*)) は *X*(*, *j*) に移動される。

forwrd = .FALSE. の場合、逆方向置換である。

j = 1, 2, ..., *n* では、*X*(*, *j*) は *X*(*, *k*(*j*)) に移動される。

入力パラメータ

forwrd	LOGICAL。 *forwrd* = .TRUE. の場合、順方向置換。 *forwrd* = .FALSE. の場合、逆方向置換。
m	INTEGER。マトリックス *X* の行数。*m* ≥ 0
n	INTEGER。マトリックス *X* の列数。*n* ≥ 0
x	REAL (slapmt の場合) DOUBLE PRECISION (dlapmt の場合) COMPLEX (clapmt の場合) COMPLEX*16 (zlapmt の場合) 配列 DIMENSION は (*ldx*, *n*)。*m* × *n* マトリックス *X* を格納する。

ldx INTEGER。配列 *x* のリーディング・ディメンジョン。
ldx ≥ max(1, *m*)

k INTEGER
配列 DIMENSION は (*n*)。*k* には置換ベクトルを格納する。

出力パラメータ

x *x* は置換されたマトリックス *X* で上書きされる。

?lapy2

sqrt(x^2+y^2) を返す。

```
val = slapy2 ( x, y )
val = dlapy2 ( x, y )
```

説明

関数 ?lapy2 は、無用なオーバーフローや有害なアンダーフローを発生させないようにしながら、sqrt(x^2+y^2) を返す。

入力パラメータ

x, y REAL (slapy2 の場合)
DOUBLE PRECISION (dlapy2 の場合)
入力値 *x* と *y* を指定する。

出力パラメータ

val REAL (slapy2 の場合)
DOUBLE PRECISION (dlapy2 の場合)
関数の戻り値。

?lapy3

sqrt(x^2+y^2+z^2) を返す。

```
val = slapy3 ( x, y, z )
val = dlapy3 ( x, y, z )
```

説明

関数 ?lapy3 は、無用なオーバーフローや有害なアンダーフローを発生させないようにしながら、sqrt(x^2+y^2+z^2) を返す。

入力パラメータ

x, y, z	REAL (slapy3 の場合) DOUBLE PRECISION (dlapy3 の場合) 入力値 *x*、*y*、*z* を指定する。

出力パラメータ

val	REAL (slapy3 の場合) DOUBLE PRECISION (dlapy3 の場合) 関数の戻り値。

?laqgb

?gbequ で計算された行と列のスケール係数を使って一般バンド・マトリックスをスケーリングする。

```
call slaqgb ( m, n, kl, ku, ab, ldab, r, c, rowcnd, colcnd, amax,
              equed )
call dlaqgb ( m, n, kl, ku, ab, ldab, r, c, rowcnd, colcnd, amax,
              equed )
call claqgb ( m, n, kl, ku, ab, ldab, r, c, rowcnd, colcnd, amax,
              equed )
call zlaqgb ( m, n, kl, ku, ab, ldab, r, c, rowcnd, colcnd, amax,
              equed )
```

説明

このルーチンは、*kl* 個の劣対角と *ku* 個の優対角を持つ一般 $m \times n$ バンド・マトリックス A を、ベクトル *r* と *c* に格納されている行と列のスケール係数を用いて平衡化する。

入力パラメータ

m	INTEGER。マトリックス A の行数。$m \geq 0$
n	INTEGER。マトリックス A の列数。$n \geq 0$
kl	INTEGER。A のバンド内にある劣対角の数。$kl \geq 0$
ku	INTEGER。A のバンド内にある優対角の数。$ku \geq 0$
ab	REAL (slaqgb の場合) DOUBLE PRECISION (dlaqgb の場合) COMPLEX (claqgb の場合) COMPLEX*16 (zlaqgb の場合) 配列 DIMENSION は (*ldab*, *n*)。マトリックス A をバンド格納形式で行 1 から *kl*+*ku*+1 に格納する。A の *j* 番目の列は配列 *ab* の *j* 番目の列に次のように格納する。$\max(1, j-ku) \leq i \leq \min(m, j+kl)$ に対して $ab(ku+1+i-j, j) = A(i, j)$
ldab	INTEGER。配列 *ab* のリーディング・ディメンジョン。 $lda \geq kl+ku+1$
amax	REAL (slaqgb/claqgb の場合) DOUBLE PRECISION (dlaqgb/zlaqgb の場合) 最大マトリックス成分の絶対値。

出力パラメータ

ab
A と同じ格納形式の平衡化マトリックスで上書きされる。
平衡化マトリックスの形式は *equed* を参照のこと。

r, c
REAL (slaqgb/claqgb の場合)
DOUBLE PRECISION (dlaqgb/zlaqgb の場合)
配列 : *r* (*m*), *c* (*n*)。それぞれ、*A* に対する行と列のスケール係数。

rowcnd
REAL (slaqgb/claqgb の場合)
DOUBLE PRECISION (dlaqgb/zlaqgb の場合)
最小の *r*(*i*) を最大の *r*(*i*) で割った値。

colcnd
REAL (slaqgb/claqgb の場合)
DOUBLE PRECISION (dlaqgb/zlaqgb の場合)
最小の *c*(*i*) を最大の *c*(*i*) で割った値。

equed
CHARACTER*1
実行された平衡化の形式を表わす。
equed = 'N' の場合、平衡化は行われていない。
equed = 'R' の場合、行の平衡化、すなわち、*A* は diag(*r*) で事前乗算された。
equed = 'C' の場合、列の平衡化、すなわち、*A* は diag(*c*) で事後乗算された。
equed = 'B' の場合、行と列の平衡化、すなわち、*A* は diag(*r*)*A*diag(*c*) で置き換えられた。

アプリケーション・ノート

このルーチンは内部パラメータ *thresh*、*large*、*small* を使用する。*thresh* はしきい値で、行または列のスケール係数の比率にもとづいた行または列のスケーリングの実行判定に使用される。
rowcnd < *thresh* の場合は行スケーリングが実行され、*colcnd* < *thresh* の場合は列スケーリングが実行される。*large* と *small* はそれぞれしきい値で、最大マトリックス成分の絶対サイズにもとづいた行スケーリングの実行判定に使用される。*amax* > *large* または *amax* < *small* の場合に行スケーリングが実行される。

?laqge

?geequ で計算された行と列のスケール係数を使って一般矩形マトリックスをスケーリングする。

```
call slaqge ( m, n, a, lda, r, c, rowcnd, colcnd, amax, equed )
call dlaqge ( m, n, a, lda, r, c, rowcnd, colcnd, amax, equed )
call claqge ( m, n, a, lda, r, c, rowcnd, colcnd, amax, equed )
call zlaqge ( m, n, a, lda, r, c, rowcnd, colcnd, amax, equed )
```

説明

このルーチンは、一般 $m \times n$ マトリックス A を、ベクトル r と c に格納されている行と列のスケール係数を用いて平衡化する。

入力パラメータ

m	INTEGER。マトリックス A の行数。$m \geq 0$
n	INTEGER。マトリックス A の列数。$n \geq 0$
a	REAL (slaqge の場合) DOUBLE PRECISION (dlaqge の場合) COMPLEX (claqge の場合) COMPLEX*16 (zlaqge の場合) 配列 DIMENSION は (*lda*, *n*)。$m \times n$ のマトリックス A を格納する。
lda	INTEGER。配列 A のリーディング・ディメンジョン。$lda \geq \max(m, 1)$
r	REAL (slanqge/claqge の場合) DOUBLE PRECISION (dlaqge/zlaqge の場合) 配列 DIMENSION は (*m*)。A に対する行スケール係数。

c REAL (slanqge/claqge の場合)
DOUBLE PRECISION (dlaqge/zlaqge の場合)
配列 DIMENSION は (*n*)。*A* に対する列スケール係数。

rowcnd REAL (slanqge/claqge の場合)
DOUBLE PRECISION (dlaqge/zlaqge の場合)
最小の *r*(*i*) を最大の *r*(*i*) で割った値。

colcnd REAL (slanqge/claqge の場合)
DOUBLE PRECISION (dlaqge/zlaqge の場合)
最小の *c*(*i*) を最大の *c*(*i*) で割った値。

amax REAL (slanqge/claqge の場合)
DOUBLE PRECISION (dlaqge/zlaqge の場合)
最大マトリックス成分の絶対値。

出力パラメータ

a 平衡化されたマトリックスで上書きされる。
平衡化されたマトリックスの形式は *equed* を参照のこと。

equed CHARACTER*1
実行された平衡化の形式を表わす。
equed = 'N' の場合、平衡化は行われていない。
equed = 'R' の場合、行の平衡化、すなわち、*A* は diag(*r*) で事前乗算された。
equed = 'C' の場合、列の平衡化、すなわち、*A* は diag(*c*) で事後乗算された。
equed = 'B' の場合、行と列の平衡化、すなわち、*A* は diag(*r*)*A*diag(*c*) で置き換えられた。

アプリケーション・ノート

このルーチンは内部パラメータ *thresh*、*large*、*small* を使用する。*thresh* はしきい値で、行または列のスケール係数の比率にもとづいた行または列のスケーリングの実行判定に使用される。
rowcnd < *thresh* の場合は行スケーリングが実行され、*colcnd* < *thresh* の場合は列スケーリングが実行される。*large* と *small* はそれぞれしきい値で、最大マトリックス成分の絶対サイズにもとづいた行スケーリングの実行判定に使用される。*amax* > *large* または *amax* < *small* の場合に行スケーリングが実行される。

?laqp2

マトリックス・ブロックに対して
列ピボット演算を用いた *QR* 因子分解を
計算する。

```
call slaqp2 ( m, n, offset, a, lda, jpvt, tau, vn1, vn2, work )
call dlaqp2 ( m, n, offset, a, lda, jpvt, tau, vn1, vn2, work )
call claqp2 ( m, n, offset, a, lda, jpvt, tau, vn1, vn2, work )
call zlaqp2 ( m, n, offset, a, lda, jpvt, tau, vn1, vn2, work )
```

説明

このルーチンはブロック A(*offset*+1:*m*, 1:*n*) に対して列ピボット演算を用いた *QR* 因子分解を計算する。ブロック A(1:*offset*, 1:*n*) はピボットされるが因子分解はされない。

入力パラメータ

m	INTEGER。マトリックス A の行数。$m \geq 0$
n	INTEGER。マトリックス A の列数。$n \geq 0$
offset	INTEGER。因子分解はしないがピボット演算すべきマトリックス A の行数。$offset \geq 0$
a	REAL (slaqp2 の場合) DOUBLE PRECISION (dlaqp2 の場合) COMPLEX (claqp2 の場合) COMPLEX*16 (zlaqp2 の場合) 配列 DIMENSION は (*lda*, *n*)。$m \times n$ のマトリックス A を格納する。
lda	INTEGER。配列 A のリーディング・ディメンジョン。$lda \geq \max(1, m)$

jpvt INTEGER
配列 DIMENSION は (*n*)。*jpvt*(*i*) ≠ 0 の場合、*A* の *i* 番目の列は *A***P* の先頭 (先頭の列) に置換される。*jpvt*(*i*) = 0 の場合、*A* の *i* 番目の列はフリー列になる。

vn1, vn2 REAL (slaqp2/claqp2 の場合)
DOUBLE PRECISION (dlaqp2/zlaqp2 の場合)
配列 DIMENSION はそれぞれ、(*n*)。それぞれ、部分的な列ノルムと完全な列ノルムを持つベクトルを格納する。

work REAL (slaqp2 の場合)
DOUBLE PRECISION (dlaqp2 の場合)
COMPLEX (claqp2 の場合)
COMPLEX*16 (zlaqp2 の場合)
ワークスペース配列 DIMENSION は (*n*)

出力パラメータ

a ブロック *A* (*offset*+1:*m*, 1:*n*) の上三角には得られた三角係数が上書きされる。対角より下のブロック *A* (*offset*+1:*m*, 1:*n*) の成分は、配列 *tau* とともに、基本リフレクタの積として直交マトリックス *Q* を表現する。ブロック *A* (1:*offset*, 1:*n*) はピボットされるが因子分解はされない。

jpvt *jpvt*(*i*) = *k* の場合、*A* の *k* 番目の列が *A***P* の *i* 番目の列になる。

tau REAL (slaqp2 の場合)
DOUBLE PRECISION (dlaqp2 の場合)
COMPLEX (claqp2 の場合)
COMPLEX*16 (zlaqp2 の場合)
配列 DIMENSION は (min(*m*, *n*))。基本リフレクタのスカラー係数。

vn1, vn2 それぞれ部分的な列ノルムと完全な列ノルムを持つベクトルが格納される。

?laqps

BLAS レベル 3 を使って実数 *m* × *n* マトリックス *A* に対して列ピボット演算を用いた *QR* 因子分解のステップを計算する。

```
call slaqps ( m, n, offset, nb, kb, a, lda, jpvt, tau, vn1, vn2, auxv,
              f, ldf )
call dlaqps ( m, n, offset, nb, kb, a, lda, jpvt, tau, vn1, vn2, auxv,
              f, ldf )
call claqps ( m, n, offset, nb, kb, a, lda, jpvt, tau, vn1, vn2, auxv,
              f, ldf )
call zlaqps ( m, n, offset, nb, kb, a, lda, jpvt, tau, vn1, vn2, auxv,
              f, ldf )
```

説明

このルーチンは、BLAS レベル 3 を使って、実数 *m* × *n* マトリックス *A* に対して列ピボット演算を用いた *QR* 因子分解を計算する。このルーチンは、行 *offset*+1 からから始めて *A* の *nb* 列の因子分解を試み、BLAS レベル 3 ルーチン ?gemm を使ってマトリックス全体を更新する。

大幅な桁落ちの発生によって ?laqps が *nb* 列の因子分解を実行できない場合がある。実際に因子分解された列数は *kb* に返される。

ブロック *A* (1:*offset*, 1:*n*) はピボットされるが因子分解はされない。

入力パラメータ

m	INTEGER。マトリックス *A* の行数。*m* ≥ 0
n	INTEGER。マトリックス *A* の列数。*n* ≥ 0
offset	INTEGER。前のステップですでに因子分解された *A* の行数。
nb	INTEGER。因子分解する列数。

a
REAL (slaqps の場合)
DOUBLE PRECISION (dlaqps の場合)
COMPLEX (claqps の場合)
COMPLEX*16 (zlaqps の場合)
配列 DIMENSION は (*lda*, *n*)
$m \times n$ のマトリックス A を格納する。

lda
INTEGER。配列 *a* のリーディング・ディメンジョン。
lda ≥ max(1, *m*)

jpvt
INTEGER
配列 DIMENSION は (*n*)。*jpvt*(*i*) = *k* の場合、フル・マトリックス A の列 *k* は AP の位置 *i* に置換されている。

vn1, vn2
REAL (slaqps/claqps の場合)
DOUBLE PRECISION (dlaqps/zlaqps の場合)
配列 DIMENSION はそれぞれ、(*n*)。それぞれ、部分的な列ノルムと完全な列ノルムを持つベクトルを格納する。

auxv
REAL (slaqps の場合)
DOUBLE PRECISION (dlaqps の場合)
COMPLEX (claqps の場合)
COMPLEX*16 (zlaqps の場合)
配列 DIMENSION は (*nb*)。補助ベクトル。

f
REAL (slaqps の場合)
DOUBLE PRECISION (dlaqps の場合)
COMPLEX (claqps の場合)
COMPLEX*16 (zlaqps の場合)
配列 DIMENSION は (*ldf*, *nb*)。マトリックス
$F' = L*Y'*A$

ldf
INTEGER。配列 *f* のリーディング・ディメンジョン。
ldf ≥ max(1, *n*)

出力パラメータ

kb
INTEGER。実際に因子分解された列数。

a
ブロック A (*offset*+1:*m*, 1:*kb*) は得られた三角係数が上書きされる。また、block A (1:*offset*, 1:*n*) はピボットされているが因子分解はされていない。マトリックスの残りの部分 block A (*offset*+1:*m*, *kb*+1:*n*) は更新されている。

jpvt	INTEGER。配列 DIMENSION は (*n*)。*jpvt*(*i*) = *k* の場合、フル・マトリックス *A* の列 *k* は *AP* の位置 *i* に置換されている。
tau	REAL (slaqps の場合) DOUBLE PRECISION (dlaqps の場合) COMPLEX (claqps の場合) COMPLEX*16 (zlaqps の場合) 配列 DIMENSION は (*kb*)。基本リフレクタのスカラー係数。
vn1, vn2	それぞれ、部分的な列ノルムと完全な列ノルムを持つベクトル。
auxv	補助ベクトル。
f	マトリックス $F' = L*Y'*A$

?laqsb

?pbequ で計算されたスケール係数を用いて対称 / エルミート・バンド・マトリックスをスケーリングする。

```
call slaqsb ( uplo, n, kd, ab, ldab, s, scond, amax, equed )
call dlaqsb ( uplo, n, kd, ab, ldab, s, scond, amax, equed )
call claqsb ( uplo, n, kd, ab, ldab, s, scond, amax, equed )
call zlaqsb ( uplo, n, kd, ab, ldab, s, scond, amax, equed )
```

説明

このルーチンはベクトル *s* に格納されているスケール係数を使って対称バンド・マトリックス *A* を平衡化する。

入力パラメータ

uplo CHARACTER*1。対称マトリックス *A* の上三角部分と下三角部分のどちらが格納されるかを指定する。
uplo = 'U' の場合は、上三角。
uplo = 'L' の場合は、下三角。

n INTEGER。マトリックス *A* の次数。$n \geq 0$

kd INTEGER。*uplo* = 'U' の場合はマトリックス *A* の優対角成分の数、*uplo* = 'L' の場合はマトリックス *A* の劣対角成分の数。$kd \geq 0$

ab REAL (slaqsb の場合)
DOUBLE PRECISION (dlaqsb の場合)
COMPLEX (claqsb の場合)
COMPLEX*16 (zlaqsb の場合)
配列 DIMENSION は (*ldab*, *n*)。対称バンド・マトリックス *A* の上三角または下三角で、配列の最初の *kd*+1 行に格納する。*A* の *j* 番目の列は配列 *ab* の *j* 番目の列に次のように格納する。
uplo = 'U' の場合、$\max(1, j-kd) \leq i \leq j$ では
$ab(kd+1+i-j, j) = A(i, j)$
uplo = 'L' の場合、$j \leq i \leq \min(n, j+kd)$ では
$ab(1+i-j, j) = A(i, j)$

ldab INTEGER。配列 *ab* のリーディング・ディメンジョン。
$ldab \geq kd+1$

scond REAL (slaqsb/claqsb の場合)
DOUBLE PRECISION (dlaqsb/zlaqsb の場合)
最小の *s*(*i*) を最大の *s*(*i*) で割った値。

amax REAL (slaqsb/claqsb の場合)
DOUBLE PRECISION (dlaqsb/zlaqsb の場合)
最大マトリックス成分の絶対値。

出力パラメータ

ab *info* = 0 の場合、バンド・マトリックス *A* のコルスキー因子分解 $A = U' U$ または $A = L L'$ から得られる三角係数 *U* または *L* が、*A* と同じ格納形式で格納される。

s	REAL (slaqsb/claqsb の場合) DOUBLE PRECISION (dlaqsb/zlaqsb の場合) 配列 DIMENSION は (*n*)。*A* のスケール係数。
equed	CHARACTER*1 平衡化が行われたかどうかを示す。 *equed* = 'N' の場合は、平衡化は行われていない。 *equed* = 'Y' の場合は、平衡化が行われ、*A* は diag(*s*)**A**diag(*s*) で置き換えられている。

アプリケーション・ノート

このルーチンは内部パラメータ *thresh*、*large*、*small* を使用する。*thresh* はしきい値で、スケール係数の比率にもとづいたスケーリングの実行判定に使用される。*scond* < *thresh* の場合にスケーリングが実行される。*large* と *small* はそれぞれしきい値で、最大マトリックス成分の絶対サイズにもとづいたスケーリングの実行判定に使用される。*amax* > *large* または *amax* < *small* の場合にスケーリングが実行される。

?laqsp

?ppequ で計算されたスケール係数を用いてパックド格納形式にある対称 / エルミート・マトリックスをスケーリングする。

```
call slaqsp ( uplo, n, ap, s, scond, amax, equed )
call dlaqsp ( uplo, n, ap, s, scond, amax, equed )
call claqsp ( uplo, n, ap, s, scond, amax, equed )
call zlaqsp ( uplo, n, ap, s, scond, amax, equed )
```

説明

ルーチン ?laqsp は、ベクトル *s* に格納されているスケール係数を使って対称マトリックス *A* を平衡化する。

入力パラメータ

uplo CHARACTER*1。対称マトリックス *A* の上三角部分と下三角部分のどちらが格納されるかを指定する。
uplo = 'U' の場合は、上三角。
uplo = 'L' の場合は、下三角。

n INTEGER。マトリックス *A* の次数。$n \geq 0$

ap REAL (slaqsp の場合)
DOUBLE PRECISION (dlaqsp の場合)
COMPLEX (claqsp の場合)
COMPLEX*16 (zlaqsp の場合)
配列 DIMENSION は $(n(n+1)/2)$。線形配列に列方向にパックされた対称マトリックス *A* の上三角または下三角を格納する。A の *j* 番目の列は配列 *ap* に次のように格納する。
uplo = 'U' の場合、$1 \leq i \leq j$ では
$ap(i + (j-1)j/2) = A(i, j)$
uplo = 'L' の場合、$j \leq i \leq n$ では
$ap(i + (j-1)(2n-j)/2) = A(i, j)$

s REAL (slaqsp/claqsp の場合)
DOUBLE PRECISION (dlaqsp/zlaqsp の場合)
配列 DIMENSION は (n)。*A* のスケール係数。

scond REAL (slaqsp/claqsp の場合)
DOUBLE PRECISION (dlaqsp/zlaqsp の場合)
最小の $s(i)$ を最大の $s(i)$ で割った値。

amax REAL (slaqsp/claqsp の場合)
DOUBLE PRECISION (dlaqsp/zlaqsp の場合)
最大マトリックス成分の絶対値。

出力パラメータ

ap *A* と同じ形式で平衡化されたマトリックス diag(*s*)**A**diag(*s*) で上書きされる。

equed CHARACTER*1
平衡化が行われたかどうかを示す。
equed = 'N' の場合は、平衡化は行われていない。
equed = 'Y' の場合は、平衡化が行われ、*A* は diag(*s*)**A**diag(*s*) で置き換えられている。

アプリケーション・ノート

このルーチンは内部パラメータ *thresh*、*large*、*small* を使用する。*thresh* はしきい値で、スケール係数の比率にもとづいたスケーリングの実行判定に使用される。*scond* < *thresh* の場合にスケーリングが実行される。*large* と *small* はそれぞれしきい値で、最大マトリックス成分の絶対サイズにもとづいたスケーリングの実行判定に使用される。*amax* > *large* または *amax* < *small* の場合にスケーリングが実行される。

?laqsy

?poequ で計算されたスケール係数を用いて対称 / エルミート・マトリックスをスケーリングする。

```
call slaqsy ( uplo, n, a, lda, s, scond, amax, equed )
call dlaqsy ( uplo, n, a, lda, s, scond, amax, equed )
call claqsy ( uplo, n, a, lda, s, scond, amax, equed )
call zlaqsy ( uplo, n, a, lda, s, scond, amax, equed )
```

説明

このルーチンは、ベクトル *s* に格納されているスケール係数を使って対称マトリックス *A* を平衡化する。

入力パラメータ

uplo CHARACTER*1。対称マトリックス *A* の上三角部分と下三角部分のどちらが格納されるかを指定する。
uplo = 'U' の場合は、上三角。
uplo = 'L' の場合は、下三角。

n INTEGER。マトリックス *A* の次数。$n \geq 0$

a REAL (slaqsy の場合)
DOUBLE PRECISION (dlaqsy の場合)
COMPLEX (claqsy の場合)
COMPLEX*16 (zlaqsy の場合)

配列 DIMENSION は (*lda*, *n*)。対称マトリックス *A*。
uplo = 'U' の場合、先頭の *n* × *n* 上三角部分にマトリックス *A* の上三角部分を格納する。厳密な下三角部分は参照されない。
uplo = 'L' の場合、先頭の *n* × *n* 下三角部分にマトリックス *A* の下三角部分を格納する。厳密な上三角部分は参照されない。

lda INTEGER。配列 *a* のリーディング・ディメンジョン。
lda ≥ max(*n*, 1)

s REAL (slaqsy/claqsy の場合)
DOUBLE PRECISION (dlaqsy/zlaqsy の場合)
配列 DIMENSION は (*n*)。*A* のスケール係数。

scond REAL (slaqsy/claqsy の場合)
DOUBLE PRECISION (dlaqsy/zlaqsy の場合)
最小の *s*(*i*) を最大の *s*(*i*) で割った値。

amax REAL (slaqsy/claqsy の場合)
DOUBLE PRECISION (dlaqsy/zlaqsy の場合)
最大マトリックス成分の絶対値。

出力パラメータ

a *equed* = 'Y' の場合、平衡化されたマトリックス、diag(*s*)*A*diag(*s*) が格納される。

equed CHARACTER*1
平衡化が行われたかどうかを示す。
equed = 'N' の場合は、平衡化は行われていない。
equed = 'Y' の場合は、平衡化が行われ、*A* は diag(*s*)*A*diag(*s*) で置き換えられている。

アプリケーション・ノート

このルーチンは内部パラメータ *thresh*、*large*、*small* を使用する。*thresh* はしきい値で、スケール係数の比率にもとづいたスケーリングの実行判定に使用される。*scond* < *thresh* の場合にスケーリングが実行される。*large* と *small* はそれぞれしきい値で、最大マトリックス成分の絶対サイズにもとづいたスケーリングの実行判定に使用される。*amax* > *large* または *amax* < *small* の場合にスケーリングが実行される。

?laqtr

実数準三角連立方程式または特殊な形式の
複素準三角連立方程式を実数演算で解く。

```
call slaqtr ( ltran, lreal, n, t, ldt, b, w, scale, x, work, info )
call dlaqtr ( ltran, lreal, n, t, ldt, b, w, scale, x, work, info )
```

説明

ルーチン `?laqtr` は実数準三角連立方程式、

op(T) * p = `scale`* c、`lreal` = `.TRUE.` の場合

または複素準三角連立方程式、

op($T + iB$)*($p+iq$) = `scale`*($c+id$)、`lreal` = `.FALSE.` の場合

を実数演算で解き、ここで T は上準三角である。

`lreal` = `.FALSE.` の場合、T の最初の対角ブロックは 1×1 でなければならず、また、B は次のような特殊な構造のマトリックスである。

$$B = \begin{bmatrix} b_1 & b_2 & \dots & \dots & b_n \\ & w & & & \\ & & w & & \\ & & & \dots & \\ & & & & w \end{bmatrix}$$

op(A) = A または A'。ここで A' はマトリックス A の共役転置である。

入力では、

$x = \begin{bmatrix} c \\ d \end{bmatrix}$ 、出力では、$x = \begin{bmatrix} p \\ q \end{bmatrix}$

このルーチンはルーチン ?trsna における条件数推定に用いられる。

入力パラメータ

ltran LOGICAL。
ltran は共役転置のオプションを指定する。
= .FALSE. の場合、$\mathrm{op}(T + iB) = T + iB$、
= .TRUE. の場合、$\mathrm{op}(T + iB) = (T + iB)'$。

lreal LOGICAL。
lreal は入力マトリックスの構造を指定する。
= .FALSE. の場合、入力は複素、
= .TRUE. の場合、入力は実数。

n INTEGER。*n* は $T + iB$ の次数を指定する。$n \geq 0$

t REAL (slaqtr の場合)
DOUBLE PRECISION (dlaqtr の場合)
配列 DIMENSION は (*ldt*, *n*)。*t* にはマトリックスを Schur 標準形で格納する。*lreal* = .FALSE. の場合、*t* の最初の対角ブロックは 1×1 でなければならない。

ldt INTEGER。マトリックス T のリーディング・ディメンジョン。*ldt* $\geq \max(1, n)$

b REAL (slaqtr の場合)
DOUBLE PRECISION (dlaqtr の場合)
配列 DIMENSION は (*n*)。*b* には上述のようにマトリックス B を形成するための成分を格納する。
lreal = .TRUE. の場合、*b* は参照されない。

w REAL (slaqtr の場合)
DOUBLE PRECISION (dlaqtr の場合)
w にはマトリックス B の対角成分を格納する。
lreal = .TRUE. の場合、*w* は参照されない。

x REAL (slaqtr の場合)
DOUBLE PRECISION (dlaqtr の場合)
配列 DIMENSION は (2*n*)。*x* には連立方程式の右辺を格納する。

work REAL (slaqtr の場合)
DOUBLE PRECISION (dlaqtr の場合)
ワークスペース配列、次元は (*n*)

出力パラメータ

scale REAL (slaqtr の場合)
DOUBLE PRECISION (dlaqtr の場合)
scale にはスケール係数が格納される。

x *x* は解で上書きされる。

info INTEGER
info = 0 の場合、正常に終了したことを示す。
info = 1 の場合、非特異点を維持するために、ある対角 1 × 1 ブロックが小さい数 smin によって摂動されたことを示す。
info = 2 の場合、非特異点を維持するために、ある対角 2 × 2 ブロックが ?laln2 に含まれる小さい数によって摂動されたことを示す。

注：実行速度を勘案してこのルーチンでは入力のエラーをチェックしない。

?lar1v

三重対角マトリックス LDL^T - σ*I の
行 b1 から bn にある部分マトリックスに
対して逆行列の (スケールされた)
r 番目の列を計算する。

```
call slar1v ( n, b1, bn, sigma, d, l, ld, lld, gersch, z, ztz, mingma,
              r, isuppz, work )
call dlar1v ( n, b1, bn, sigma, d, l, ld, lld, gersch, z, ztz, mingma,
              r, isuppz, work )
call clar1v ( n, b1, bn, sigma, d, l, ld, lld, gersch, z, ztz, mingma,
              r, isuppz, work )
call zlar1v ( n, b1, bn, sigma, d, l, ld, lld, gersch, z, ztz, mingma,
              r, isuppz, work )
```

説明

ルーチン ?lar1v は、三重対角マトリックス LDL^T-σ*I の行 b1 から bn にある部分マトリックスに対して、逆行列の (スケールされた) r 番目の列を計算する。

この計算は次のようなステップで実行される。

1. 定常 qd 変換、LDL^T - σ*I = L(+) D(+) L(+)^T
2. 進行 qd 変換、LDL^T - σ*I = U(-) D(-) U(-)^T
3. 記変換の組み合わせと、大きさで最大 (のうちの 1 つ) となる逆行列の対角成分の場所を示すインデックスとして r を選択すれば、LDL^T - σ*I の逆行列の対角成分を計算する。
4. 逆行列の (スケールされた) r 番目の列を、定常変換の上部分と進行変換の下部分との組み合わせで得られるツイスト分解を使用して計算する。

入力パラメータ

n	INTEGER。マトリックス LDL^T の次数。
b1	INTEGER。LDL^T の部分マトリックスの最初のインデックス。

bn INTEGER。LDL^T の部分マトリックスの最後のインデックス。

sigma REAL (slar1v/clar1v の場合)
DOUBLE PRECISION (dlar1v/zlar1v の場合)
シフト。初期時点の *r* = 0 の場合、*sigma* は LDL^T の固有値の良好な近似でなければならない。

l REAL (slar1v/clar1v の場合)
DOUBLE PRECISION (dlar1v/zlar1v の場合)
配列 DIMENSION は (*n*-1)。単位二重対角マトリックス *L* の (*n*-1) 個の劣対角成分を成分 1 から *n*-1 に格納する。

d REAL (slar1v/clar1v の場合)
DOUBLE PRECISION (dlar1v/zlar1v の場合)
配列 DIMENSION は (*n*)。対角マトリックス *D* の *n* 個の対角成分。

ld REAL (slar1v/clar1v の場合)
DOUBLE PRECISION (dlar1v/zlar1v の場合)
配列 DIMENSION は (*n*-1)。*n*-1 個の成分 L_i*D_i

lld REAL (slar1v/clar1v の場合)
DOUBLE PRECISION (dlar1v/zlar1v の場合)
配列 DIMENSION は (*n*-1)。*n*-1 個の成分 $L_i*L_i*D_i$

gersch REAL (slar1v/clar1v の場合)
DOUBLE PRECISION (dlar1v/zlar1v の場合)
配列 DIMENSION は (2*n*)。*n* 個の Gerschgorin 間隔。*r* がゼロとして入力された場合、これらは *r* の初期検索を制限するために使用される。

r INTEGER
最初の入力では *r* はゼロとする。出力では大きさにおいて最大の逆行列の対角成分の場所を示すインデックスとなる。その後の反復では出力と同じ値を入力に与えなければならない。

work REAL (slar1v/clar1v の場合)
DOUBLE PRECISION (dlar1v/zlar1v の場合)
ワークスペース配列 DIMENSION は (4*n*)

出力パラメータ

z	REAL (slar1v の場合) DOUBLE PRECISION (dlar1v の場合) COMPLEX (clar1v の場合) COMPLEX*16 (zlar1v の場合) 配列 DIMENSION は (*n*)。逆行列の (スケールされた) *r* 番目の列。*z*(*r*) は 1 として返される。
ztz	REAL (slar1v/clar1v の場合) DOUBLE PRECISION (dlar1v/zlar1v の場合) *z* のノルムの平方。
mingma	REAL (slar1v/clar1v の場合) DOUBLE PRECISION (dlar1v/zlar1v の場合) LDL^T - σ*I の逆行列の (大きさにおいて) 最大の対角成分の逆数。
r	大きさでは最大となる逆行列の対角成分の場所を示すインデックス。
isuppz	INTEGER 配列 DIMENSION は (2)。*z* にあるベクトルのサポート情報、すなわち、ベクトル *z* は成分 *isuppz*(1) から *isuppz*(2) のみ非ゼロ。

?lar2v

一連の 2 × 2 対称 / エルミート・マトリックスに、実数余弦と実数 / 複素正弦を持つ面回転のベクトルを両側から適用する。

```
call slar2v ( n, x, y, z, incx, c, s, incc )
call dlar2v ( n, x, y, z, incx, c, s, incc )
call clar2v ( n, x, y, z, incx, c, s, incc )
call zlar2v ( n, x, y, z, incx, c, s, incc )
```

説明

ルーチン ?lar2v は、ベクトル x、y、z の成分で定義される一連の 2×2 実数対称または複素エルミート・マトリックスに、実数余弦を持つ実数 / 複素面回転のベクトルを両側から適用する。$i = 1, 2, ..., n$ では、

$$\begin{bmatrix} x_i & z_i \\ \mathrm{conjg}(z_i) & y_i \end{bmatrix} := \begin{bmatrix} c(i) & \mathrm{conjg}(s(i)) \\ -s(i) & c(i) \end{bmatrix} \begin{bmatrix} x_i & z_i \\ \mathrm{conjg}(z_i) & y_i \end{bmatrix} \begin{bmatrix} c(i) & -\mathrm{conjg}(s(i)) \\ s(i) & c(i) \end{bmatrix}$$

入力パラメータ

n	INTEGER。適用する面回転の回数。
x, y, z	REAL (slar2v の場合) DOUBLE PRECISION (dlar2v の場合) COMPLEX (clar2v の場合) COMPLEX*16 (zlar2v の場合) 配列 DIMENSION はそれぞれ、(1+(*n*-1)**incx*) それぞれ、ベクトル x、y、z を格納する。?lar2v のすべての型で x と y は実数とする。
incx	INTEGER。*x*、*y*、*z* の成分間の増分。*incx* > 0
c	REAL (slar2v/clar2v の場合) DOUBLE PRECISION (dlar2v/zlar2v の場合) 配列 DIMENSION は (1+(*n*-1)**incc*)。面回転の余弦。
s	REAL (slar2v の場合) DOUBLE PRECISION (dlar2v の場合) COMPLEX (clar2v の場合) COMPLEX*16 (zlar2v の場合) 配列 DIMENSION は (1+(*n*-1)**incc*)。面回転の正弦。
incc	INTEGER。*c* と *s* の成分間の増分。*incc* > 0

出力パラメータ

x, y, z	ベクトル *x*、*y*、*z* は変換結果で上書きされる。

?larf

一般矩形マトリックスに基本リフレクタを適用する。

```
call slarf ( side, m, n, v, incv, tau, c, ldc, work )
call dlarf ( side, m, n, v, incv, tau, c, ldc, work )
call clarf ( side, m, n, v, incv, tau, c, ldc, work )
call zlarf ( side, m, n, v, incv, tau, c, ldc, work )
```

説明

このルーチンは、実数 / 複素基本リフレクタ *H* を、実数 / 複素 *m* × *n* マトリックス *C* に、左または右のいずれかから適用する。*H* は次の形式で表現される。

H = *I* - *tau* * *v* * *v*'

tau は実数 / 複素スカラーで、*v* は実数 / 複素ベクトルである。

tau = 0 の場合、*H* は単位マトリックスをとる。

clarf/zlarf で *H*′ (*H* の共役転置) を適用するには *tau* の代わりに conjg(*tau*) を与える。

入力パラメータ

side	CHARACTER*1 *side* = 'L' の場合、形式 *H* *C* *side* = 'R' の場合、形式 *C* *H*
m	INTEGER。マトリックス *C* の行数。
n	INTEGER。マトリックス *C* の列数。
v	REAL (slarf の場合) DOUBLE PRECISION (dlarf の場合) COMPLEX (clarf の場合)

COMPLEX*16 (zlarf の場合)
配列 DIMENSION は
(1 + (*m*-1)*abs(*incv*)) (*side* = 'L' の場合)
(1 + (*n*-1)*abs(*incv*)) (*side* = 'R' の場合)
H の表現におけるベクトル *v*。*tau* = 0 の場合、*v* は参照されない。

incv INTEGER。*v* の成分間の増分。*incv* ≠ 0

tau REAL (slarf の場合)
DOUBLE PRECISION (dlarf の場合)
COMPLEX (clarf 場合)
COMPLEX*16 (zlarf の場合)
H の表現における値 *tau*

c REAL (slarf の場合)
DOUBLE PRECISION (dlarf の場合)
COMPLEX (clarf の場合)
COMPLEX*16 (zlarf の場合)
配列 DIMENSION は (*ldc*, *n*)
m × *n* のマトリックス *C* を格納する。

ldc INTEGER。配列 *C* のリーディング・ディメンジョン。
ldc ≥ max(1, *m*)

work REAL (slarf の場合)
DOUBLE PRECISION (dlarf の場合)
COMPLEX (clarf の場合)
COMPLEX*16 (zlarf の場合)
ワークスペース 配列 DIMENSION は
side = 'L' の場合では (*n*)、または
side = 'R' の場合では (*m*)

出力パラメータ

c *side* = 'L' の場合はマトリックス *H***C* で、*side* = 'R' の場合はマトリックス *C***H* で上書きされる。

?larfb

ブロック・リフレクタまたはその
転置 / 共役転置を一般矩形マトリックスに
適用する。

```
call slarfb ( side, trans, direct, storev, m, n, k, v, ldv, t, ldt, c,
              ldc, work, ldwork )
call dlarfb ( side, trans, direct, storev, m, n, k, v, ldv, t, ldt, c,
              ldc, work, ldwork )
call clarfb ( side, trans, direct, storev, m, n, k, v, ldv, t, ldt, c,
              ldc, work, ldwork )
call zlarfb ( side, trans, direct, storev, m, n, k, v, ldv, t, ldt, c,
              ldc, work, ldwork )
```

説明

ルーチン ?larfb は、複素ブロック・リフレクタ H またはその転置 H' を、複素 $m \times n$ マトリックス C に、左または右のいずれかから適用する。

入力パラメータ

side CHARACTER*1

side = 'L' の場合、H または H' を左から適用する。
side = 'R' の場合、H または H' を右から適用する。

trans CHARACTER*1

trans = 'N' の場合、H を適用する (転置なし)
trans = 'C' の場合、H' を適用する (共役転置)

direct CHARACTER*1。基本リフレクタの積からどのように H が表現されているかを示す。

direct = 'F' の場合、$H = H(1)\,H(2)\,...\,H(k)$ (順方向)
direct = 'B' の場合、$H = H(k)\,...\,H(2)\,H(1)$ (逆方向)

storev CHARACTER*1。基本リフレクタを定義するベクトルがどのように格納されているかを示す。

storev = 'C' の場合、列方向。
storev = 'R' の場合、行方向。

m	INTEGER。マトリックス C の行数。
n	INTEGER。マトリックス *C* の列数。
k	INTEGER。マトリックス *T* の次数 (積がブロック・リフレクタを定義する基本リフレクタの個数に等しい)。
v	REAL (slarfb の場合) DOUBLE PRECISION (dlarfb の場合) COMPLEX (clarfb の場合) COMPLEX*16 (zlarfb の場合) 配列 DIMENSION は (*ldv*, *k*) *storev* = 'C' の場合 (*ldv*, *m*) *storev* = 'R' および *side* = 'L' の場合 (*ldv*, *n*) *storev* = 'R' および *side* = 'R' の場合 マトリックス *V*
ldv	INTEGER 配列 v のリーディング・ディメンジョン。 *storev* = 'C' および *side* = 'L' の場合、*ldv* ≥ max(1, *m*)、 *storev* = 'C' および *side* = 'R' の場合、*ldv* ≥ max(1, *n*)、 *storev* = 'R' の場合、 *ldv* ≥ *k*
t	REAL (slarfb の場合) DOUBLE PRECISION (dlarfb の場合) COMPLEX (clarfb の場合) COMPLEX*16 (zlarfb の場合) 配列 DIMENSION は (*ldt*, *k*) ブロック・リフレクタの表現にある *k* × *k* マトリックス *T* の三角を格納する。
ldt	INTEGER。配列 *t* のリーディング・ディメンジョン。 *ldt* ≥ *k*
c	REAL (slarfb の場合) DOUBLE PRECISION (dlarfb の場合) COMPLEX (clarfb の場合) COMPLEX*16 (zlarfb の場合) 配列 DIMENSION は (*ldc*, *n*) *m* × *n* のマトリックス *C*

ldc	INTEGER。配列 *c* のリーディング・ディメンジョン。 *ldc* ≥ max(1, *m*)
work	REAL (slarfb の場合) DOUBLE PRECISION (dlarfb の場合) COMPLEX (clarfb の場合) COMPLEX*16 (zlarfb の場合) ワークスペース配列 DIMENSION は (*ldwork*, *k*)
ldwork	INTEGER。配列 *work* のリーディング・ディメンジョン。 *side* = 'L' の場合、*ldwork* ≥ max(1, *n*)、 *side* = 'R' の場合、*ldwork* ≥ max(1, *m*)

出力パラメータ

c	終了時に、*c* は *H***C* または *H*′**C*、あるいは *C***H* または *C***H*′ によって上書きされる。

?larfg

基本リフレクタ (Householder マトリックス) を生成する

```
call slarfg ( n, alpha, x, incx, tau )
call dlarfg ( n, alpha, x, incx, tau )
call clarfg ( n, alpha, x, incx, tau )
call zlarfg ( n, alpha, x, incx, tau )
```

説明

ルーチン ?larfg は、次を満たすような次数 *n* の実数 / 複素基本リフレクタ *H* を生成する。

$$H'*\begin{bmatrix} alpha \\ x \end{bmatrix} = \begin{bmatrix} beta \\ 0 \end{bmatrix} 、H'*H = I$$

alpha と *beta* はスカラー (*beta* はすべての型で実数)、*x* は (*n*-1) 成分で構成される実数 / 複素ベクトルである。

H は次の形式で表現される。

$$H = I - tau * \begin{bmatrix} 1 \\ v \end{bmatrix} * \begin{bmatrix} 1 & v' \end{bmatrix}$$

tau は実数/複素スカラー、v は実数/複素の (*n*-1) 成分で構成されるベクトルである。clarfg/zlarfg では、*H* はエルミートではない点に注意する。

x の成分がすべてゼロの場合 (かつ、複素型で *alpha* が実数)、*tau* = 0 となり、*H* は単位マトリックスをとる。

それ以外では、1 ≤ *tau* ≤ 2 (実数型の場合)、または、
1 ≤ Re(*tau*) ≤ 2 かつ abs(*tau*-1) ≤ 1 (複素型の場合)

入力パラメータ

n	INTEGER。基本リフレクタの次数。
alpha	REAL (slarfg の場合) DOUBLE PRECISION (dlarfg の場合) COMPLEX (clarfg の場合) COMPLEX*16 (zlarfg の場合) *alpha* の値を格納する。
x	REAL (slarfg の場合) DOUBLE PRECISION (dlarfg の場合) COMPLEX (clarfg の場合) COMPLEX*16 (zlarfg の場合) 配列 DIMENSION は (1 + (*n*-2)*abs(*incx*)) ベクトル *x* を格納する。
incx	INTEGER *x* の成分間の増分。*incx* > 0

出力パラメータ

alpha	*beta* の値で上書きされる。
x	ベクトル *v* で上書きされる。

tau REAL (slarfg の場合)
DOUBLE PRECISION (dlarfg の場合)
COMPLEX (clarfg の場合)
COMPLEX*16 (zlarfg の場合)
tau の値。

?larft

ブロック・リフレクタ $H = I - VTV^H$ の
三角係数 T を生成する。

```
call slarft ( direct, storev, n, k, v, ldv, tau, t, ldt )
call dlarft ( direct, storev, n, k, v, ldv, tau, t, ldt )
call clarft ( direct, storev, n, k, v, ldv, tau, t, ldt )
call zlarft ( direct, storev, n, k, v, ldv, tau, t, ldt )
```

説明

ルーチン ?larft は、次数 *n* の実 / 複素ブロック・リフレクタ H の三角係数 T を生成する。ブロック・リフレクタ H は *k* 個の基本リフレクタの積として定義される

direct = 'F' の場合、$H = H(1)\,H(2)\,...\,H(k)$、$T$ は上三角。

direct = 'B' の場合、$H = H(k)\,...\,H(2)\,H(1)$、$T$ は下三角。

storev = 'C' の場合、基本リフレクタ $H(i)$ を定義するベクトルは配列 *v* の i 番目の列に格納され、$H = I - V*T*V'$ である。

storev = 'R' の場合、基本リフレクタ $H(i)$ を定義するベクトルは配列 *v* の i 番目の行に格納され、$H = I - V'*T*V$ である。

入力パラメータ

direct CHARACTER*1。ブロック・リフレクタを生成するために基本リフレクタを乗算する次数を指定する。
= 'F' の場合、$H = H(1)\,H(2)\,...\,H(k)$ (順方向)
= 'B' の場合、$H = H(k)\,...\,H(2)\,H(1)$ (逆方向)

storev CHARACTER*1。基本リフレクタを定義するベクトルがどのように格納されているかを指定する。(次の**アプリケーション・ノート**も参照)
= 'C' の場合、列方向。
= 'R' の場合、行方向。

n INTEGER。ブロック・リフレクタ H の次数。$n \geq 0$

k INTEGER。三角係数 T の次数 (基本リフレクタの個数に等しい)。$k \geq 1$

v REAL (slarft の場合)
DOUBLE PRECISION (dlarft の場合)
COMPLEX (clarft の場合)
COMPLEX*16 (zlarft の場合)
配列 DIMENSION は
storev = 'C' の場合では (*ldv*, *k*)、または
storev = 'R' の場合では (*ldv*, *n*)
マトリックス V

ldv INTEGER。配列 *v* のリーディング・ディメンジョン。
storev = 'C' の場合、*ldv* ≥ max(1, *n*)、
storev = 'R' の場合、*ldv* ≥ *k*

tau REAL (slarft の場合)
DOUBLE PRECISION (dlarft の場合)
COMPLEX (clarft の場合)
COMPLEX*16 (zlarft の場合)
配列 DIMENSION は (*k*)。*tau*(*i*) には、基本リフレクタ H(*i*) のスカラー係数が入っていなければならない。

ldt INTEGER。出力配列 *t* のリーディング・ディメンジョン。*ldt* ≥ *k*

出力パラメータ

t REAL (slarft の場合)
DOUBLE PRECISION (dlarft の場合)
COMPLEX (clarft の場合)
COMPLEX*16 (zlarft の場合)
配列 DIMENSION は (*ldt*, *k*)。ブロック・リフレクタの $k \times k$ 三角係数 T。*direct* = 'F' の場合、T は上三角。*direct* = 'B' の場合、T は下三角。配列の残りの部分は使用されない。

v マトリックス *V*

アプリケーション・ノート

マトリックス *V* の形状と *H*(*i*) を定義するベクトルの格納形式を、*n* = 5、*k* = 3 において次の図に示す。1 に等しい成分は格納されない。対応する配列成分は変更されるが、出力で元に戻される。配列の残りの部分は使用されない。

direct = 'F' および *storev* = 'C'

$$\begin{bmatrix} 1 & & \\ v_1 & 1 & \\ v_1 & v_2 & 1 \\ v_1 & v_2 & v_3 \\ v_1 & v_2 & v_3 \end{bmatrix}$$

direct = 'F' および *storev* = 'R'

$$\begin{bmatrix} 1 & v_1 & v_1 & v_1 & v_1 \\ & 1 & v_2 & v_2 & v_2 \\ & & 1 & v_3 & v_3 \end{bmatrix}$$

direct = 'B' および *storev* = 'C'

$$\begin{bmatrix} v_1 & v_2 & v_3 \\ v_1 & v_2 & v_3 \\ 1 & v_2 & v_3 \\ & 1 & v_3 \\ & & 1 \end{bmatrix}$$

direct = 'B' および *storev* = 'R

$$\begin{bmatrix} v_1 & v_1 & 1 & & \\ v_2 & v_2 & v_2 & 1 & \\ v_3 & v_3 & v_3 & v_3 & 1 \end{bmatrix}$$

?larfx

リフレクタが次数 ≤ 10 を持つ場合、ループの繰り返しを避けながら、基本リフレクタを一般矩形マトリックスに適用する。

```
call slarfx ( side, m, n, v, tau, c, ldc, work )
call dlarfx ( side, m, n, v, tau, c, ldc, work )
call clarfx ( side, m, n, v, tau, c, ldc, work )
call zlarfx ( side, m, n, v, tau, c, ldc, work )
```

説明

ルーチン ?larfx は、実数 / 複素基本リフレクタ H を、実数 / 複素 $m \times n$ マトリックス C に、左辺または右辺から適用する。

H は次の形式で表現される。

$H = I$ - *tau***v***v*、

tau は実数 / 複素スカラー、v は実数 / 複素ベクトルである。

tau = 0 の場合、H は単位マトリックスをとる。

入力パラメータ

side	CHARACTER*1 *side* = 'L' の場合、形式 H *C *side* = 'R' の場合、形式 C *H
m	INTEGER。マトリックス C の行数。
n	INTEGER。マトリックス C の列数。
v	REAL (slarfx の場合) DOUBLE PRECISION (dlarfx の場合) COMPLEX (clarfx の場合) COMPLEX*16 (zlarfx の場合) 配列 DIMENSION は *side* = 'L' の場合では (*m*)、または *side* = 'R' の場合では (*n*) H の表現におけるベクトル v

tau REAL (slarfx の場合)
DOUBLE PRECISION (dlarfx の場合)
COMPLEX (clarfx の場合)
COMPLEX*16 (zlarfx の場合)
H の表現における値 *tau*

c REAL (slarfx の場合)
DOUBLE PRECISION (dlarfx の場合)
COMPLEX (clarfx の場合)
COMPLEX*16 (zlarfx の場合)
配列 DIMENSION は (*ldc*, *n*)。*m* × *n* のマトリックス C を格納する。

ldc INTEGER。配列 *C* のリーディング・ディメンジョン。*lda* ≥ (1, *m*)

work REAL (slarfx の場合)
DOUBLE PRECISION (dlarfx の場合)
COMPLEX (clarfx の場合)
COMPLEX*16 (zlarfx の場合)
ワークスペース 配列 DIMENSION は
(*n*) *side* = 'L' の場合、または
(*m*) *side* = 'R' の場合。
H が次数 < 11 を持つ場合は *work* は参照されない。

出力パラメータ

c *side* = 'L' の場合はマトリックス *H***C* で、*side* = 'R' の場合はマトリックス *C***H* で上書きされる。

?largv

実数余弦および実数 / 複素正弦を持つ
面回転のベクトルを生成する。

```
call slargv ( n, x, incx, y, incy, c, incc )
call dlargv ( n, x, incx, y, incy, c, incc )
call clargv ( n, x, incx, y, incy, c, incc )
call zlargv ( n, x, incx, y, incy, c, incc )
```

説明

このルーチンは、実数 / 複素ベクトル x と y の成分で決定される、実数余弦を持った実数 / 複素面回転のベクトルを生成する。

slargv/dlargv の場合

$$\begin{bmatrix} c(i) & s(i) \\ -s(i) & c(i) \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} = \begin{bmatrix} a_i \\ 0 \end{bmatrix}$$ 、$i = 1, 2, ..., n$ の場合

clargv/zlargv の場合

$$\begin{bmatrix} c(i) & s(i) \\ -\mathrm{conjg}(s(i)) & c(i) \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} = \begin{bmatrix} r_i \\ 0 \end{bmatrix}$$ 、$i = 1, 2, ..., n$ の場合

ここで $c(i)^2 + \mathrm{abs}(s(i))^2 = 1$。また、次の規則が使用される (clartg/zlartg と同じだが、BLAS レベル 1 ルーチン crotg/zrotg とは異なる)。

$y_i = 0$ の場合、$c(i) = 0$ かつ $s(i) = 0$

$x_i = 0$ の場合、$c(i) = 0$、かつ r_i が実数となるように $s(i)$ が選択される。

入力パラメータ

n	INTEGER。適用する面回転の回数。
x, y	REAL (slargv の場合) DOUBLE PRECISION (dlargv の場合) COMPLEX (clargv の場合) COMPLEX*16 (zlargv の場合) 配列 DIMENSION はそれぞれ、(1 + (*n*-1)**incx*) および (1 + (*n*-1)**incy*) ベクトル x、y を格納する。
incx	INTEGER。*x* の成分間の増分。*incx* > 0

incy	INTEGER。*y* の成分間の増分。*incy* > 0
incc	INTEGER。出力配列 *c* の分間の増分。*incc* > 0

出力パラメータ

x	$i = 1, ..., n$ に対して、*x*(*i*) は a_i (実数型) または r_i (複素型) で上書きされる。
y	面回転の正弦 *s*(*i*) で上書きされる。
c	REAL (slargv/clargv の場合) DOUBLE PRECISION (dlargv/zlargv の場合) 配列 DIMENSION は (1 + (*n*-1)**incc*)。面回転の余弦。

?larnv

乱数ベクトルを一様分布または正規分布で返す。

```
call slarnv ( idist, iseed, n, x )
call dlarnv ( idist, iseed, n, x )
call clarnv ( idist, iseed, n, x )
call zlarnv ( idist, iseed, n, x )
```

説明

ルーチン ?larnv は、*n* 個の実数 / 複素乱数のベクトルを一様分布または正規分布で返す。

このルーチンは、一様分布 (0, 1) の実数乱数を生成するために、ベクトル化可能なコードを使った最大 128 のバッチ処理中に補助ルーチン ?laruv を呼び出す。一様分布から正規分布への数値変換は Box-Muller 法を使用する。

入力パラメータ

idist INTEGER。乱数の分布を指定する。
slarnv と dlanrv の場合。
= 1 の場合、一様分布 (0, 1)
= 2 の場合、一様分布 (-1, 1)
= 3 の場合、正規分布 (0, 1)。
clarnv と zlanrv の場合。
= 1 の場合、実数部分と虚数部分がそれぞれ一様分布 (0, 1)
= 2 の場合、実数部分と虚数部分がそれぞれ一様分布 (-1, 1)
= 3 の場合、実数部分と虚数部分がそれぞれ正規分布 (0, 1)
= 4 の場合、円盤 abs(z) < 1 上で一様分布
= 5 の場合、円 abs(z) = 1 上で一様分布。

iseed INTEGER
配列 DIMENSION は (4)
乱数生成器の種 (シード) を与える。配列成分は 0 から 4095 の間でなければならない。また *iseed*(4) は奇数でなければならない。

n INTEGER。生成する乱数の個数。

出力パラメータ

x REAL (slarnv の場合)
DOUBLE PRECISION (dlarnv の場合)
COMPLEX (clarnv の場合)
COMPLEX*16 (zlarnv の場合)
配列 DIMENSION は (*n*)。生成された乱数。

iseed 更新された種。

?larrb

より高い精度で固有値を見つけるための
限定二分法を与える。

```
call slarrb ( n, d, l, ld, lld, ifirst, ilast, sigma, reltol, w, wgap,
              werr, work, iwork, info )
call dlarrb ( n, d, l, ld, lld, ifirst, ilast, sigma, reltol, w, wgap,
              werr, work, iwork, info )
```

説明

このルーチンは、比較的安定な表現 (RRR) の LDL^T が与えられた状態で、より高い精度で LDL^T の固有値 *w*(*ifirst*) から *w*(*ilast*) を見つけるために、「限定された」二分法を実行する。間隔 [**左** , **右**] は、それらの中点と部分幅を配列 *w* と *werr* にそれぞれ格納することによって維持される。

入力パラメータ

n	INTEGER。マトリックスの次数。
d	REAL (slarrb の場合) DOUBLE PRECISION (dlarrb の場合) 配列 DIMENSION は (*n*)。対角マトリックス *D* の *n* 個の対角成分。
l	REAL (slarrb の場合) DOUBLE PRECISION (dlarrb の場合) 配列 DIMENSION は (*n*-1)。単位二重対角マトリックス *L* の *n*-1 個の劣対角成分。
ld	REAL (slarrb の場合) DOUBLE PRECISION (dlarrb の場合) 配列 DIMENSION は (*n*-1)。*n*-1 個の成分 L_i*Di
lld	REAL (slarrb の場合) DOUBLE PRECISION (dlarrb の場合) 配列 DIMENSION は (*n*-1)。*n*-1 個の成分 L_i*L_i*D_i
ifirst	INTEGER。クラスタ内の最初の固有値のインデックス。

ilast INTEGER。クラスタ内の最後の固有値のインデックス。

sigma REAL (slarrb の場合)
DOUBLE PRECISION (dlarrb の場合)
LDL^T を形成するために使用されたシフト。(?larrf を参照)

reltol REAL (slarrb の場合)
DOUBLE PRECISION (dlarrb の場合)
相対許容値。

w REAL (slarrb の場合)
DOUBLE PRECISION (dlarrb の場合)
配列 DIMENSION は (*n*)。*w*(*ifirst*) から *w*(*ilast*) には LDL^T の固有値に対応した推定を格納する。

wgap REAL (slarrb の場合)
DOUBLE PRECISION (dlarrb の場合)
配列 DIMENSION は (*n*)。LDL^T の固有値間の隔たり。

werr REAL (slarrb の場合)
DOUBLE PRECISION (dlarrb の場合)
配列 DIMENSION は (*n*)。*werr*(*ifirst*) から *werr*(*ilast*) には推定 *w*(*ifirst*) から *w*(*ilast*) の誤差を格納する。

work REAL (slarrb の場合)
DOUBLE PRECISION (dlarrb の場合)
ワークスペース配列。このパラメータはルーチン内では使用されない。

iwork INTEGER
ワークスペース配列 DIMENSION は (2*n*)

出力パラメータ

w 固有値の推定の精度を改善した結果が上書きされる。

wgap 微小な隔たりが変更される。

werr 推定 *w*(*ifirst*) から *w*(*ilast*) の誤差に対して精度を改善した結果が上書きされる。

info INTEGER
エラーフラグ。ルーチンではこのパラメータは設定されない。

?larre

三重対角マトリックス T が与えられたとき、
小さい対角外成分をゼロに設定し、
縮退されていないブロック T_i に対して
基本表現と固有値を探す。

```
call slarre ( n, d, e, tol, nsplit, isplit, m, w, woff, gersch, work,
              info )
call dlarre ( n, d, e, tol, nsplit, isplit, m, w, woff, gersch, work,
              info )
```

説明

三重対角マトリックス T が与えられたとき、このルーチンは「小さい」対角外成分をゼロに設定し、縮退されていないブロック T_i に対して、

- 数 σ_i
- 基本 $T_i - \sigma_i I = L_i D_i L_i^T$ 表現
- 各 $L_i D_i L_i^T$ の固有値

を探す。

求められた表現と固有値は対称三重対角マトリックスの固有値を計算するために ?stegr で使用される。現時点で基本表現は正定値または負定値として制限されており、定値マトリックスの固有値は *dqds* アルゴリズムによって求められる (サブルーチン ?lasq2)。また ?larre は、各 $L_i D_i L_i^T$ に対する *n* 個の Gerschgorin 間隔を出力する利点を持つ。

入力パラメータ

n INTEGER。マトリックスの次数。

d REAL (slarre の場合)
DOUBLE PRECISION (dlarre の場合)
配列 DIMENSION は (*n*)。三重対角マトリックス T の *n* 個の対角成分を格納する。

e REAL (slarre の場合)
DOUBLE PRECISION (dlarre の場合)
配列 DIMENSION は (*n*)。三重対角マトリックス T の (*n*-1) 個の劣対角成分を格納する。*e*(*n*) は設定する必要はない。

tol REAL (slarre の場合)
DOUBLE PRECISION (dlarre の場合)
分割のしきい値。入力において $|e(i)| < tol$ の場合、マトリックス T は小さいブロックに分割される。

nsplit INTEGER。ブロック T を分割する数。$1 \le nsplit \le n$

work REAL (slarre の場合)
DOUBLE PRECISION (dlarre の場合)
ワークスペース配列 DIMENSION は (4*n)

出力パラメータ

d 対角マトリックス D_i の *n* 個の対角成分で上書きされる。

e 単位二重対角マトリックス L_i の劣対角成分で上書きされる。

isplit INTEGER
配列 DIMENSION は (2*n*)。T を部分マトリックスに分割した分割点。第 1 の部分マトリックスは 1 から *isplit*(1) の行 / 列で構成され、第 2 の部分マトリックスは *isplit*(1)+1 から *isplit*(2) の行 / 列で構成され、以下同様であり、*nsplit* 番目は *isplit*(*nsplit*-1)+1 から *isplit*(*nsplit*) = *n* の行 / 列で構成される。

m INTEGER。求められた (すべての $L_i D_i L_i^T$ の) 固有値の総数。

w	REAL (slarre の場合) DOUBLE PRECISION (dlarre の場合) 配列 DIMENSION は (*n*)。最初の *m* 個の要素には固有値が格納される。各ブロック $L_i D_i L_i^T$ の各固有値は昇順でソートされる。
woff	REAL (slarre の場合) DOUBLE PRECISION (dlarre の場合) 配列 DIMENSION は (*n*)。*nsplit* 基本点 σ_i
gersch	REAL (slarre の場合) DOUBLE PRECISION (dlarre の場合) 配列 DIMENSION は (2*n*)。*n* 個の Gerschgorin 間隔。
info	INTEGER。?lasq2 のエラーコード。

?larrf

少なくとも 1 つの固有値が相対的に
分離されているような新しい比較的安定な
表現を探す。

```
call slarrf ( n, d, l, ld, lld, ifirst, ilast, w, dplus, lplus, work,
              iwork, info )
call dlarrf ( n, d, l, ld, lld, ifirst, ilast, w, dplus, lplus, work,
              iwork, info )
```

説明

初期表現 LDL^T と (相対的に) 値が近いそれらの固有値のクラスタ *w*(*ifirst*), *w*(*ifirst*+1), ... *w*(*ilast*) が与えられたとき、ルーチン ?larrf は、新しい比較的安定な表現を探す。

$$LDL^T - \sigma_i I = L(+)D(+)L(+)^T$$

$L(+)D(+)L(+)^T$ の少なくとも 1 つの固有値が相対的に分離されている。

入力パラメータ

n INTEGER。マトリックスの次数。

d REAL (slarrf の場合)
DOUBLE PRECISION (dlarrf の場合)
配列 DIMENSION は (*n*)。対角マトリックス *D* の *n* 個の対角成分。

l REAL (slarrf の場合)
DOUBLE PRECISION (dlarrf の場合)
配列 DIMENSION は (*n*-1)。単位二重対角マトリックス *L* の (*n*-1) 個の劣対角成分。

ld REAL (slarrf の場合)
DOUBLE PRECISION (dlarrf の場合)
配列 DIMENSION は (*n*-1)。*n*-1 個の成分 L_i*D_i

lld REAL (slarrf の場合)
DOUBLE PRECISION (dlarrf の場合)
配列 DIMENSION は (*n*-1)。*n*-1 個の成分 $L_i*L_i*D_i$

ifirst INTEGER。クラスタ内の最初の固有値のインデックス。

ilast INTEGER。クラスタ内の最後の固有値のインデックス。

w REAL (slarrf の場合)
DOUBLE PRECISION (dlarrf の場合)
配列 DIMENSION は (*n*)。LDL^T の固有値を昇順で格納する。*w*(*ifirst*) から *w*(*ilast*) が比較的安定な表現のクラスタを形成する。

sigma REAL (slarrf の場合)
DOUBLE PRECISION (dlarrf の場合)
$L(+)D(+)L(+)^T$ を形成するために使用されるシフト。

work REAL (slarrf の場合)
DOUBLE PRECISION (dlarrf の場合)
ワークスペース配列。

出力パラメータ

w	*w*(*ifirst*) から *w*(*ilast*) は、$L(+)D(+)L(+)^T$ の対応する固有値の推定で上書きされる。
dplus	REAL (slarrf の場合) DOUBLE PRECISION (dlarrf の場合) 配列 DIMENSION は (*n*)。対角マトリックス $D(+)$ の *n* 個の対角成分。
lplus	REAL (slarrf の場合) DOUBLE PRECISION (dlarrf の場合) 配列 DIMENSION は (*n*)。*lplus* の最初の (*n*-1) 個の成分には単位二重対角マトリックス $L(+)$ の劣対角成分が格納される。*lplus*(*n*) は *sigma* に設定される。

?larrv

L、D、$L\,D\,L^T$ の固有値が与えられたとき、三重対角マトリックス $T = L\,D\,L^T$ の固有ベクトルを計算する。

```
call slarrv ( n, d, l, isplit, m, w, iblock, gersch, tol, z, ldz,
              isuppz, work, iwork, info )
call dlarrv ( n, d, l, isplit, m, w, iblock, gersch, tol, z, ldz,
              isuppz, work, iwork, info )
call clarrv ( n, d, l, isplit, m, w, iblock, gersch, tol, z, ldz,
              isuppz, work, iwork, info )
call zlarrv ( n, d, l, isplit, m, w, iblock, gersch, tol, z, ldz,
              isuppz, work, iwork, info )
```

説明

ルーチン ?larrv は、L、D、$L\,D\,L^T$ の固有値が与えられたとき、三重対角マトリックス $T = L\,D\,L^T$ の固有ベクトルを計算する。入力の固有値は L と D の成分に対して相対的に高い精度を持っていなければならない。目的とする出力精度は入力パラメータ *tol* で指定できる。

入力パラメータ

n INTEGER。マトリックスの次数。$n \geq 0$

d REAL (slarrv/clarrv の場合)
DOUBLE PRECISION (dlarrv/zlarrv の場合)
配列 DIMENSION は (*n*)。対角マトリックス *D* の *n* 個の対角成分を格納する。

l REAL (slarrv/clarrv の場合)
DOUBLE PRECISION (dlarrv/zlarrv の場合)
配列 DIMENSION は (*n*-1)。単位対角マトリックス *L* の (*n*-1) 個の劣対角成分を *l* の成分 *1* から *n*-1 に格納する。*l*(*n*) を設定する必要はない。

isplit INTEGER
配列 DIMENSION は (*n*)。*T* を部分マトリックスに分割する分割点。第 1 の部分マトリックスは 1 から *isplit*(1) の行 / 列で構成され、第 2 の部分マトリックスは *isplit*(1)+1 から *isplit*(2) の行 / 列で構成され、以下同様。

tol REAL (slarrv/clarrv の場合)
DOUBLE PRECISION (dlarrv/zlarrv の場合)
固有値 / 固有ベクトルに対する絶対誤差許容値。
入力固有値の誤差は *tol* を上限としれなければならない。
固有ベクトル出力は *tol* を上限とする誤差ノルムを持ち、また、異なる固有ベクトル間のドット積は *tol* に制限される。*tol* は少なくとも $n*eps*|T|$ でなければならず、ここで *eps* はマシン精度、$|T|$ は三重対角マトリックスの 1- ノルムである。

m INTEGER。求められた固有値の個数。
$0 \leq m \leq n$。*range* = 'A' の場合は $m = n$、*range* = 'I' の場合は $m = iu - il + 1$

w REAL (slarrv/clarrv の場合)
DOUBLE PRECISION (dlarrv/zlarrv の場合)
配列 DIMENSION は (*n*)。*w* の最初の *m* 個の成分には固有ベクトルを計算する固有値を格納する。固有値は分割されたブロックごとにグループ分けし、ブロック内で最小から最大に並べられていなければならない (?larre の出力配列 *w* がここで要求される)。
w の誤差は *tol* を上限としなければならない。

iblock INTEGER
配列 DIMENSION は (*n*)。*w* 内の対応する固有値に関係した部分マトリックス・インデックス。固有値 *w*(*i*) が第 1 の部分マトリックスの先頭から属する場合は *iblock*(*i*) = 1、*w*(*i*) が第 2 のマトリックスに属する場合は *iblock*(*i*) = 2、以下同様。

ldz INTEGER。出力配列 *z* のリーディング・ディメンジョン。*ldz* ≥ 1、*jobz* = 'V' の場合は *ldz* ≥ max(1, *n*)。

work REAL (slarrv/clarrv の場合)
DOUBLE PRECISION (dlarrv/zlarrv の場合)
ワークスペース配列 DIMENSION は (13*n*)。

iwork INTEGER
ワークスペース配列 DIMENSION は (6*n*)。

出力パラメータ

d *d* は上書きされることがある。

l *l* は上書きされる。

z REAL (slarrv の場合)
DOUBLE PRECISION (dlarrv の場合)
COMPLEX (clarrv の場合)
COMPLEX*16 (zlarrv の場合)
配列 DIMENSION は (*ldz*, max(1, *m*))。
jobz = 'V' の場合、*info* = 0 ならば、マトリックス *T* の正規直交固有ベクトルのうち、選択された固有値に対応するものが *z* の最初の *m* 列に格納される。すなわち、*w*(*i*) に対応する固有ベクトルが *z* の *i* 番目の列に入る。

jobz = 'N' の場合、*z* は参照されない。

注：配列 *z* には max(1, *m*) 以上の列が与えられなければならない。*range* = 'V' の場合は *m* の正確な値が事前にわからないため、上限値を使用する必要がある。

isuppz INTEGER

配列 DIMENSION は (2*max(1, *m*))。*z* に入っている固有ベクトルのサポート情報、すなわち、*z* に入っている非ゼロ成分を示すインデックス。*i* 番目の固有ベクトルは *isuppz*(2*i*-1) から *isuppz*(2*i*) までの成分のみ非ゼロである。

info INTEGER

info = 0 の場合、正常に終了したことを示す。
info = -*i* < 0 の場合、*i* 番目の引数の値が不正だったことを示す。
info > 0 の場合、*info* = 1 ならば ?larrb で内部エラーが発生したことを示す。
info = 2 ならば ?stein で内部エラーが発生したことを示す。

?lartg

実数余弦と実数 / 複素正弦を持つ面回転を生成する。

```
call slartg ( f, g, cs, sn, r )
call dlartg ( f, g, cs, sn, r )
call clartg ( f, g, cs, sn, r )
call zlartg ( f, g, cs, sn, r )
```

説明

このルーチンは、次のように面回転を生成する。

$$\begin{bmatrix} cs & sn \\ -\mathrm{conjg}(sn) & cs \end{bmatrix} \cdot \begin{bmatrix} f \\ g \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix}$$

ここで $cs^2 + |sn|^2 = 1$

このルーチンは BLAS レベル 1 ルーチン ?rotg の実行速度が遅く精度が高いバージョンであるが、次の点で相違がある。

slartg/dlartg の場合。

f と g は出力時に変更されない。

$g = 0$ の場合、$cs = 1$ と $sn = 0$

$f = 0$ かつ $g \neq 0$ の場合、浮動小数点演算は実行されず $cs = 0$ と $sn = 1$ (対角にゼロがある場合、?bdsqr の動作を省く)。

f が大きさで g を超えた場合、cs は正となる。

clartg/zlartg の場合。

f と g は出力時に変更されない。

$g = 0$ の場合、$cs = 1$ と $sn = 0$

$f = 0$ の場合、$cs = 0$、と r が実数になるように sn が選択される。

入力パラメータ

f, g	REAL (slartg の場合) DOUBLE PRECISION (dlartg の場合) COMPLEX (clartg の場合) COMPLEX*16 (zlartg の場合) 回転対象のベクトルの第 1 および第 2 成分。

出力パラメータ

cs	REAL (slartg/clartg の場合) DOUBLE PRECISION (dlartg/zlartg の場合) 回転の余弦。

sn REAL (slartg の場合)
DOUBLE PRECISION (dlartg の場合)
COMPLEX (clartg の場合)
COMPLEX*16 (zlartg の場合)
回転の正弦。

r REAL (slartg の場合)
DOUBLE PRECISION (dlartg の場合)
COMPLEX (clartg の場合)
COMPLEX*16 (zlartg の場合)
回転されたベクトルの非ゼロ成分。

?lartv

実数余弦と実数 / 複素制限を持つ面回転の
ベクトルをベクトル対の成分に
適用する。

```
call slartv ( n, x, incx, y, incy, c, s, incc )
call dlartv ( n, x, incx, y, incy, c, s, incc )
call clartv ( n, x, incx, y, incy, c, s, incc )
call zlartv ( n, x, incx, y, incy, c, s, incc )
```

説明

このルーチンは、実数余弦を持つ実数 / 複素面回転を実数 / 複素ベクトル x と y の成分に適用する。i = 1, 2, ..., *n* では、

$$\begin{bmatrix} x_i \\ y_i \end{bmatrix} := \begin{bmatrix} c(i) & s(i) \\ -\mathrm{conjg}(s(i)) & c(i) \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix}$$

入力パラメータ

n INTEGER。適用する面回転の回数。

x, y	REAL (slartv の場合) DOUBLE PRECISION (dlartv の場合) COMPLEX (clartv の場合) COMPLEX*16 (zlartv の場合) 配列 DIMENSION はそれぞれ (1 + (*n*-1)**incx*) と (1 + (*n*-1)**incy*)。入力ベクトル *x* と *y*
incx	INTEGER。*x* の成分間の増分。*incx* > 0
incy	INTEGER。*y* の成分間の増分。*incy* > 0
c	REAL (slartv/clartv の場合) DOUBLE PRECISION (dlartv/zlartv の場合) 配列 DIMENSION は (1 + (*n*-1)**incc*)。面回転の余弦。
s	REAL (slartv の場合) DOUBLE PRECISION (dlartv の場合) COMPLEX (clartv の場合) COMPLEX*16 (zlartv の場合) 配列 DIMENSION は (1 + (*n*-1)**incc*)。面回転の正弦。
incc	INTEGER。*c* と *s* の成分間の増分。*incc* > 0

出力パラメータ

x, y	回転されたベクトル *x* と *y*

?laruv

n 個の実数乱数のベクトルを一様分布で返す。

```
call slaruv ( iseed, n, x )
call dlaruv ( iseed, n, x )
```

説明

ルーチン ?laruv は *n* 個の実数乱数のベクトルを一様分布 (0, 1) で返す (*n* ≤ 128)。このルーチンは ?larnv から呼び出される補助ルーチンである。

入力パラメータ

iseed INTEGER
配列 DIMENSION は (4)。乱数生成器の種 (シード) を格納する。配列成分は 0 から 4095 の間でなければならない。また *iseed*(4) は奇数でなければならない。

n INTEGER。生成する乱数の個数。*n* ≤ 128

出力パラメータ

x REAL (slaruv の場合)
DOUBLE PRECISION (dlaruv の場合)
配列 DIMENSION は (*n*)。生成された乱数。

seed 種は更新される。

?larz

(?tzrzf が返したとおりの) 基本リフレクタを一般マトリックスに適用する。

```
call slarz ( side, m, n, l, v, incv, tau, c, ldc, work )
call dlarz ( side, m, n, l, v, incv, tau, c, ldc, work )
call clarz ( side, m, n, l, v, incv, tau, c, ldc, work )
call zlarz ( side, m, n, l, v, incv, tau, c, ldc, work )
```

説明

ルーチン ?larz は、実数 / 複素基本リフレクタ *H* を、実数 / 複素 *m* × *n* マトリックス *C* に、左または右のいずれかから適用する。

H は次の形式で表現される。

H = *I* - *tau* * *v* * *v*'、

tau は実数 / 複素スカラー、*v* は実数 / 複素ベクトルである。

tau = 0 の場合、*H* は単位マトリックスをとる。

複素型で *H*′ (*H* の共役転置) を適用するには *tau* の代わりに conjg(*tau*) を与える。

H は ?tzrzf から返されたとおり、*k* 個の基本リフレクタの積である。

入力パラメータ

side	CHARACTER*1 *side* = 'L' の場合、形式 *H***C* *side* = 'R' の場合、形式 *C***H*
m	INTEGER。マトリックス *C* の行数。
n	INTEGER。マトリックス *C* の列数。
l	INTEGER。Householder ベクトルとして意味を持った部分が格納されているベクトル *v* の成分数。 *side* = 'L' の場合は *m* ≥ *l* ≥ 0、*side* = 'R' の場合は *n* ≥ *l* ≥ 0
v	REAL (slarz の場合) DOUBLE PRECISION (dlarz の場合) COMPLEX (clarz の場合) COMPLEX*16 (zlarz の場合) 配列 DIMENSION は (1 + (*l*-1)*abs(*incv*)) ?tzrzf から返されたとおり、*H* の表現中にあるベクトル *v*。*tau* = 0 の場合、*v* は使用されない。
incv	INTEGER。*v* の成分間の増分。*incv* ≠ 0
tau	REAL (slarz の場合) DOUBLE PRECISION (dlarz の場合) COMPLEX (clarz の場合) COMPLEX*16 (zlarz の場合) *H* の表現中にある値 *tau*

c	REAL (slarz の場合) DOUBLE PRECISION (dlarz の場合) COMPLEX (clarz の場合) COMPLEX*16 (zlarz の場合) 配列 DIMENSION は (*ldc*, *n*) *m* × *n* のマトリックス *C* を格納する。
ldc	INTEGER。配列 *c* のリーディング・ディメンジョン。 *ldc* ≥ max(1, *m*)
work	REAL (slarz の場合) DOUBLE PRECISION (dlarz の場合) COMPLEX (clarz の場合) COMPLEX*16 (zlarz の場合) ワークスペース 配列 DIMENSION は (*n*) *side* = 'L' の場合、または (*m*) *side* = 'R' の場合。

出力パラメータ

c	*c* は、*side* = 'L' の場合はマトリックス *H***C* で、 *side* = 'R'の場合はマトリックス*C***H*で上書きされる。

?larzb

ブロック・リフレクタまたはその
転置 / 共役転置を一般矩形マトリックスに
適用する。

```
call slarzb ( side, trans, direct, storev, m, n, k, l, v, ldv, t, ldt,
              c, ldc, work, ldwork )
call dlarzb ( side, trans, direct, storev, m, n, k, l, v, ldv, t, ldt,
              c, ldc, work, ldwork )
call clarzb ( side, trans, direct, storev, m, n, k, l, v, ldv, t, ldt,
              c, ldc, work, ldwork )
call zlarzb ( side, trans, direct, storev, m, n, k, l, v, ldv, t, ldt,
              c, ldc, work, ldwork )
```

説明

このルーチンは実数 / 複素ブロック・リフレクタ H またはその転置 H^T (または複素型の場合 H^H) を、実数 / 複素分布 $m \times n$ マトリックス C に左または右のいずれかから適用する。
現時点で、*storev* = 'R' と *direct* = 'B' のみがサポートされている。

入力パラメータ

side CHARACTER*1
side = 'L' の場合、H または H' を左から適用する。
side = 'R' の場合、H または H' を右から適用する。

trans CHARACTER*1
trans = 'N' の場合、H を適用する (転置なし)。
trans = 'C' の場合、H' を適用する (共役転置)。

direct CHARACTER*1。基本リフレクタの積からどのように H が形成されているかを示す。
= 'F' の場合、$H = H(1)\,H(2)\,...\,H(k)$ (順方向、現時点でこの機能はサポートされていない)。
= 'B' の場合、$H = H(k)\,...\,H(2)\,H(1)$ (逆方向)。

storev CHARACTER*1。基本リフレクタを定義するベクトルがどのように格納されているかを示す。
= 'C' の場合、列方向 (現時点でこの機能はサポートされていない)。
= 'R' の場合、行方向。

m INTEGER。マトリックス C の行数。

n INTEGER。マトリックス C の列数。

k INTEGER。マトリックス T の次数 (積がブロック・リフレクタを定義する基本リフレクタの個数に等しい)。

l INTEGER。Householder ベクトルとして意味を持った部分が格納されているマトリックス V の列数。
side = 'L' の場合は $m \geq l \geq 0$、*side* = 'R' の場合は $n \geq l \geq 0$

v REAL (slarzb の場合)
DOUBLE PRECISION (dlarzb の場合)
COMPLEX (clarzb の場合)
COMPLEX*16 (zlarzb の場合)

配列 DIMENSION は (*ldv*, *nv*)
storev = 'C' の場合は *nv* = *k*、*storev* = 'R' の場合は *nv* = *l*

ldv INTEGER。配列 *v* のリーディング・ディメンジョン。
storev = 'C' の場合 *ldv* ≥ *l*、*storev* = 'R' の場合 *ldv* ≥ *k*

t REAL (slarzb の場合)
DOUBLE PRECISION (dlarzb の場合)
COMPLEX (clarzb の場合)
COMPLEX*16 (zlarzb の場合)
配列 DIMENSION は (*ldt*, *k*)。ブロック・リフレクタの表現中にある三角 $k \times k$ マトリックス T

ldt INTEGER。配列 *t* のリーディング・ディメンジョン。
ldt ≥ *k*

c REAL (slarzb の場合)
DOUBLE PRECISION (dlarzb の場合)
COMPLEX (clarzb の場合)
COMPLEX*16 (zlarzb の場合)
配列 DIMENSION は (*ldc*, *n*)。$m \times n$ マトリックス C を格納する。

ldc INTEGER。配列 *c* のリーディング・ディメンジョン。
ldc ≥ max(1, *m*)

work REAL (slarzb の場合)
DOUBLE PRECISION (dlarzb の場合)
COMPLEX (clarzb の場合)
COMPLEX*16 (zlarzb の場合)
ワークスペース配列 DIMENSION は (*ldwork*, *k*)

ldwork INTEGER。配列 *work* のリーディング・ディメンジョン。
side = 'L' の場合、*ldwork* ≥ max(1, *n*)、
side = 'R' の場合、*ldwork* ≥ max(1, *m*)

出力パラメータ

c *c* は、$H*C$、または $H'*C$、または $C*H$、または $C*H'$ で上書きされる。

?larzt

ブロック・リフレクタ $H = I\text{-}VTV^H$ の
三角係数 T を生成する。

```
call slarzt ( direct, storev, n, k, v, ldv, tau, t, ldt )
call dlarzt ( direct, storev, n, k, v, ldv, tau, t, ldt )
call clarzt ( direct, storev, n, k, v, ldv, tau, t, ldt )
call zlarzt ( direct, storev, n, k, v, ldv, tau, t, ldt )
```

説明

このルーチンは、*k* 個の基本リフレクタの積として定義されている次数 > *n* の実 / 複素ブロック・リフレクタ H の三角係数 T を生成する。
direct = 'F' の場合、$H = H(1)\,H(2)\ldots H(k)$、$T$ は上三角。
direct = 'B' の場合、$H = H(k)\ldots H(2)\,H(1)$、$T$ は下三角。
storev = 'C' の場合、基本リフレクタ $H(i)$ を定義するベクトルは配列 *v* の *i* 番目の列に格納され、$H = I - V*T*V'$ である。
storev = 'R' の場合、基本リフレクタ $H(i)$ を定義するベクトルは配列 *v* の *i* 番目の行に格納され、$H = I - V'*T*V$ である。
現時点で、*storev* = 'R' と *direct* = 'B' のみがサポートされている。

入力パラメータ

direct
CHARACTER*1。ブロック・リフレクタの生成で基本リフレクタが乗算される次数を指定する。
direct = 'F' の場合、$H = H(1)\,H(2)\ldots H(k)$ (順方向、現時点でこの機能はサポートされていない)
direct = 'B' の場合、$H = H(k)\ldots H(2)\,H(1)$ (逆方向)

storev
CHARACTER*1。基本リフレクタを定義するベクトルがどのように格納されているかを指定する。(次のアプリケーション・ノートも参照):
storev = 'C' の場合、列方向 (現時点でこの機能はサポートされていない)。
storev = 'R' の場合、行方向。

n INTEGER。ブロック・リフレクタ H の次数。$n \geq 0$

k INTEGER。三角係数 T の次数 (基本リフレクタの個数に等しい)。$k \geq 1$

v REAL (slarzt の場合)
DOUBLE PRECISION (dlarzt の場合)
COMPLEX (clarzt の場合)
COMPLEX*16 (zlarzt の場合)
配列 DIMENSION は
(*ldv*, *k*) *storev* = 'C' の場合
(*ldv*, *n*) *storev* = 'R' の場合
マトリックス V

ldv INTEGER。配列 *v* のリーディング・ディメンジョン。
storev = 'C' の場合、*ldv* ≥ max(1, *n*)、
storev = 'R' の場合、*ldv* ≥ *k*

tau REAL (slarzt の場合)
DOUBLE PRECISION (dlarzt の場合)
COMPLEX (clarzt の場合)
COMPLEX*16 (zlarzt の場合)
配列 DIMENSION は (*k*)。*tau*(*i*) には、基本リフレクタ H(*i*) のスカラー係数が入っていなければならない。

ldt INTEGER。出力配列 *t* のリーディング・ディメンジョン。*ldt* ≥ *k*

出力パラメータ

t REAL (slarzt の場合)
DOUBLE PRECISION (dlarzt の場合)
COMPLEX (clarzt の場合)
COMPLEX*16 (zlarzt の場合)
配列 DIMENSION は (*ldt*, *k*)。ブロック・リフレクタの *k* × *k* 三角係数 T。*direct* = 'F' の場合、T は上三角。*direct* = 'B' の場合、T は下三角。配列の残りの部分は使用されない。

v マトリックス V。次のアプリケーション・ノートを参照。

アプリケーション・ノート

マトリックス V の形状と $H(i)$ を定義するベクトルの格納形式の例を、$n = 5$、$k = 3$ において次の図に示す。1 に等しい成分は格納されない。対応する配列成分は変更されるが、出力で元に戻される。配列の残りの部分は使用されない。

direct = 'F' および *storev* = 'C':

$$V = \begin{bmatrix} v_1 & v_2 & v_3 \\ v_1 & v_2 & v_3 \\ v_1 & v_2 & v_3 \\ v_1 & v_2 & v_3 \\ v_1 & v_2 & v_3 \end{bmatrix}$$

$$\begin{matrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ 1 & \cdot & \cdot \\ & 1 & \cdot \\ & & 1 \end{matrix}$$

direct = 'F' および *storev* = 'R':

____V___

/ \

$$\begin{bmatrix} v_1 & v_1 & v_1 & v_1 & v_1 & \cdot & \cdot & \cdot & \cdot & 1 \\ v_2 & v_2 & v_2 & v_2 & v_2 & \cdot & \cdot & \cdot & 1 & \\ v_3 & v_3 & v_3 & v_3 & v_3 & \cdot & \cdot & 1 & & \end{bmatrix}$$

direct = 'B' および *storev* = 'C':

$$\begin{matrix} 1 & & \\ \cdot & 1 & \\ \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{matrix}$$

$$V = \begin{bmatrix} v_1 & v_2 & v_3 \\ v_1 & v_2 & v_3 \\ v_1 & v_2 & v_3 \\ v_1 & v_2 & v_3 \\ v_1 & v_2 & v_3 \end{bmatrix}$$

direct = 'B' および *storev* = 'R':

____V___

/ \

$$\begin{bmatrix} 1 & \cdot & \cdot & \cdot & \cdot & v_1 & v_1 & v_1 & v_1 & v_1 \\ \cdot & 1 & \cdot & \cdot & \cdot & v_2 & v_2 & v_2 & v_2 & v_2 \\ \cdot & \cdot & 1 & \cdot & \cdot & v_3 & v_3 & v_3 & v_3 & v_3 \end{bmatrix}$$

?las2

2 × 2 三角マトリックスの特異値を
計算する。

```
call slas2 ( f, g, h, ssmin, ssmax )
call dlas2 ( f, g, h, ssmin, ssmax )
```

説明

ルーチン ?las2 は 2 × 2 のマトリックス、

$$\begin{bmatrix} f & g \\ 0 & h \end{bmatrix}$$

の特異値を計算する。

出力では、*ssmin* に小さい方の特異値、*ssmax* に大きい方の特異値が格納される。

入力パラメータ

f, g, h　REAL (slas2 の場合)
DOUBLE PRECISION (dlas2 の場合)
それぞれ、2 × 2 マトリックスの (1,1)、(1,2)、(2,2) の成分。

出力パラメータ

ssmin, ssmax　REAL (slas2 の場合)
DOUBLE PRECISION (dlas2 の場合)
それぞれ、小さい方の特異値と大きい方の特異値。

アプリケーション・ノート

オーバーフロー / アンダーフローの場合を除いて、加算 / 減算にガード桁がなくとも、すべての出力の大きさは最後の位置からわずかな単位以内 (*ulp*) で正確である。

IEEE 演算で、1 つのマトリックス成分が無限でもコードは正しく動作する。

最大特異値自身がオーバーフローしない限り、あるいは最大特異値がオーバーフローからわずかな *ulp* 範囲内にある限り、オーバーフローは起こらない。(Cray のように部分オーバーフローを持つマシンでは、最大特異値がオーバーフローに対して 2 の因数以内にあるとオーバーフローが起こる場合がある。)

アンダーフローは緩やかであればアンダーフローは無害である。そうでない場合、計算結果が、アンダーフローしきい値に近いサイズの摂動によって変更されたマトリックスに一致するときがある。

?lascl

一般矩形マトリックスに c_{to}/c_{from} として定義される実数スカラーを乗算する。

```
call slascl ( type, kl, ku, cfrom, cto, m, n, a, lda, info )
call dlascl ( type, kl, ku, cfrom, cto, m, n, a, lda, info )
call clascl ( type, kl, ku, cfrom, cto, m, n, a, lda, info )
call zlascl ( type, kl, ku, cfrom, cto, m, n, a, lda, info )
```

説明

ルーチン `?lascl` は `m` × `n` 実数 / 複素マトリックス *A* に実数スカラー `cto`/`cfrom` を乗算する。最終結果 `cto`**A*(*i*, *j*)/`cfrom` がオーバーフロー / アンダーフローしない限り、演算はオーバーフロー / アンダーフローなしで実行される。*A* の形式は `type` によって、フル、上三角、下三角、上 Hessenberg、またはバンドから指定される。

入力パラメータ

`type` `CHARACTER*1`。`type` は入力マトリックスの格納形式を示る。
= '`G`' の場合、*A* はフル・マトリックス。
= '`L`' の場合、*A* は下三角マトリックス。
= '`U`' の場合、*A* は上三角マトリックス。
= '`H`' の場合、*A* は上 Hessenberg マトリックス。
= '`B`' の場合、*A* は下バンド幅 `kl` と上バンド幅 `ku` を持つ対称バンド・マトリックスで、下半分のみ格納される。

= 'Q' の場合、A は下バンド幅 *kl* と上バンド幅 *ku* を持つ対称バンド・マトリックスで、上半分のみ格納される。
= 'Z' の場合、A は下バンド幅 *kl* と上バンド幅 *ku* を持つ対称バンド・マトリックス。

kl INTEGER。A の下バンド幅。*type* = 'B'、'Q'、または 'Z' の場合のみ参照される。

ku INTEGER。A の上バンド幅。*type* = 'B'、'Q'、または 'Z' の場合のみ参照される。

cfrom, cto REAL (slascl/clascl の場合)
DOUBLE PRECISION (dlascl/zlascl の場合)
マトリックス A に乗算する *cto*/*cfrom*。最終結果 *cto**$A(i, j)$/*cfrom* がオーバーフロー / アンダーフローなしで表現される場合、$A(i, j)$ はオーバーフロー / アンダーフローなしで計算される。*cfrom* は非ゼロでなければならない。

m INTEGER。マトリックス A の行数。$m \geq 0$

n INTEGER。マトリックス A の列数。$n \geq 0$

a REAL (slascl の場合)
DOUBLE PRECISION (dlascl の場合)
COMPLEX (clascl の場合)
COMPLEX*16 (zlascl の場合)
配列 DIMENSION は (*lda*, *m*)。*cto*/*cfrom* が乗算されるマトリックス。格納形式については *type* を参照。

lda INTEGER。配列 *a* のリーディング・ディメンジョン。$lda \geq \max(1, m)$

出力パラメータ

a 乗算されたマトリックス A

info INTEGER
info = 0 の場合、正常に終了したことを示す。
info = -*i* < 0 の場合、*i* 番目のパラメータの値が不正だったことを示す。

?lasd0

対角 *d* と対角外 *e* を持つ実数上二重対角
n × *m* マトリックス *B* の特異値を
計算する。?bdsdc で使用される。

```
call slasd0 ( n, sqre, d, e, u, ldu, vt, ldvt, smlsiz, iwork, work,
              info )
call dlasd0 ( n, sqre, d, e, u, ldu, vt, ldvt, smlsiz, iwork, work,
              info )
```

説明

ルーチン ?lasd0 は、分割統治法を使って、対角 *d* と対角外 *e* を持つ実数上二重対角 *n* × *m* マトリックス *B* の特異値分解 (SVD) を計算する。ここで *m* = *n* + *sqre*

このアルゴリズムは、*B* = *U***S***VT* であるような直交マトリックス *U* と *VT* を計算する。特異値 *s* は *d* に上書きされる。

関連するサブルーチン ?lasda は、特異値のみを計算し、オプションで特異ベクトルをコンパクト形式で計算する。

入力パラメータ

n INTEGER。上二重対角マトリックスの行次元を格納する。主対角配列 *d* の次元でもある。

sqre INTEGER。二重対角マトリックスの列のサイズを指定する。
sqre = 0 の場合、二重対角マトリックスは列次元 *m* = *n* を持つ。
sqre = 1 の場合、二重対角マトリックスは列次元 *m* = *n*+1 を持つ。

d REAL (slasd0 の場合)
DOUBLE PRECISION (dlasd0 の場合)
配列 DIMENSION は (*n*)。*d* には二重対角マトリックスの主対角を格納する。

e REAL (slasd0 の場合)
DOUBLE PRECISION (dlasd0 の場合)
配列 DIMENSION は (*m*-1)。二重対角マトリックスの劣対角成分を格納する。出力において *e* の内容は壊される。

ldu INTEGER。出力配列 *u* のリーディング・ディメンジョンを格納する。

ldvt INTEGER。出力配列 *vt* のリーディング・ディメンジョンを格納する。

smlsiz INTEGER。計算ツリーの一番下の部分問題の最大サイズを格納する。

iwork INTEGER
ワークスペース配列 DIMENSION は少なくとも (8*n*) でなければならない。

work REAL (slasd0 の場合)
DOUBLE PRECISION (dlasd0 の場合)
ワークスペース配列 DIMENSION は少なくとも ($3m^2 + 2m$) でなければならない。

出力パラメータ

d *info* = 0 の場合、*d* には二重対角マトリックスの特異値が上書きされる。

u REAL (slasd0 の場合)
DOUBLE PRECISION (dlasd0 の場合)
配列 DIMENSION は少なくとも (*ldq*, *n*)。*u* には左特異値が格納される。

vt REAL (slasd0 の場合)
DOUBLE PRECISION (dlasd0 の場合)
配列 DIMENSION は少なくとも (*ldvt*, *m*)。*vt*' には右特異値が格納される。

info	INTEGER *info* = 0 の場合、正常に終了したことを示す。 *info* = -*i* < 0 の場合、*i* 番目の引数の値が不正であったことを示す。 *info* = 1 の場合、特異値が収束しなかったことを示す。

?lasd1

指定サイズの上二重対角マトリックス *B* の SVD を計算する。?bdsc で使用される。

```
call slasd1 ( nl, nr, sqre, d, alpha, beta, u, ldu, vt, ldvt, idxq,
              iwork, work, info )
call dlasd1 ( nl, nr, sqre, d, alpha, beta, u, ldu, vt, ldvt, idxq,
              iwork, work, info )
```

説明

このルーチンは上二重対角 $n \times m$ マトリックス *B* の SVD を計算する。ここで $n = nl + nr + 1$ および $m = n + sqre$。ルーチン ?lasd1 は ?lasd0 から呼び出される。

関連サブルーチン ?lasd7 は特異値 (および因子分解形式の特異ベクトル) を目的とする場合を扱う。
?lasd1 は次のように SVD を計算する。

$$B = U(in) * \begin{bmatrix} D1(in) & 0 & 0 & 0 \\ Z1' & a & Z2' & b \\ 0 & 0 & D2(in) & 0 \end{bmatrix} * VT(in)$$

$$= U(out) * (D(out) \;\; 0) * VT(out)$$

$Z' = (Z1' \; a \; Z2' \; b) = u' \; VT'$、また u は、nl+1 と nl+2 番目の成分に *alpha* と *beta* を持ち他はゼロである次元 m のベクトル。$sqre = 0$ の場合、成分 b は空である。

元のマトリックスの左特異ベクトルは *u* に格納され、右特異ベクトルの転置は *vt* に格納され、特異値は *d* に格納される。このアルゴリズムは 3 つの過程で構成される。

第 1 の過程では、複数の特異値がある場合、または Z ベクトルにゼロがある場合、問題の大きさの収縮を行う。そのような場合ごとに永年方程式問題の次元は 1 だけ縮退される。この過程はルーチン ?lasd2 によって処理される。

第 2 の過程では更新された特異値を計算する。これは、ルーチン ?lasd4 (?lasd3 として呼び出される) を介して永年方程式の根の平方根を見つけることによって行われる。このルーチンは現在の問題の特異ベクトルも計算する。

最後の過程では、更新された特異値を直接使用して更新された特異ベクトルを計算する。現在の問題に対する特異ベクトルは、全体問題から得られる特異ベクトルで乗算される。

入力パラメータ

nl INTEGER。上ブロックの行次元。*nl* ≥ 1

nr INTEGER。下ブロックの行次元。*nr* ≥ 1

sqre INTEGER

sqre = 0 の場合、下ブロックは *nr* × *nr* 平方マトリックスである。
sqre = 1 の場合、下ブロックは *nr* × (*nr*+1) 矩形マトリックスである。二重対角マトリックスは行次元 *n* = *nl* + *nr* + 1 と列次元 *m* = *n* + *sqre* を持つ。

d REAL (slasd1 の場合)
DOUBLE PRECISION (dlasd1 の場合)
配列 DIMENSION は (*n* = *nl*+*nr*+1)。*d*(1:*nl*, 1:*nl*) には上ブロックの特異値を格納し、*d*(*nl*+2:*n*) には下ブロックの特異値を格納する。

alpha
REAL (slasd1 の場合)
DOUBLE PRECISION (dlasd1 の場合)
追加した行に関連する対角成分。

beta
REAL (slasd1 の場合)
DOUBLE PRECISION (dlasd1 の場合)
追加した行に関連する対角外成分。

u
REAL (slasd1 の場合)
DOUBLE PRECISION (dlasd1 の場合)
配列 DIMENSION は (*ldu*, *n*)。*u*(1:*nl*, 1:*nl*) には上ブロックの左特異ベクトルを格納し、*u*(*nl*+2:*n*, *nl*+2:*n*) には下ブロックの左特異ベクトルを格納する。

ldu
INTEGER。配列 *u* のリーディング・ディメンジョン。*ldu* ≥ max(1, *n*)

vt
REAL (slasd1 の場合)
DOUBLE PRECISION (dlasd1 の場合)
配列 DIMENSION は (*ldvt*, *m*)、ここで *m* = *n* + *sqre*
vt(1:*nl*+1, 1:*nl*+1)' には上ブロックの右特異ベクトルを格納し、*vt*(*nl*+2:*m*, *nl*+2:*m*)' には下ブロックの右特異ベクトルを格納する。

ldvt
INTEGER。配列 *vt* のリーディング・ディメンジョン。*ldvt* ≥ max(1, *m*)

iwork
INTEGER
ワークスペース配列 DIMENSION は (4*n*)

work
REAL (slasd1 の場合)
DOUBLE PRECISION (dlasd1 の場合)
ワークスペース配列 DIMENSION は ($3m^2 + 2m$)

出力パラメータ

d
d(1:*n*) には、変更されたマトリックスの特異値が格納される。

u	*u* には二重対角マトリックスの左特異ベクトルが格納される。
vt	*vt*' には二重対角マトリックスの右特異ベクトルが格納される。
idxq	INTEGER 配列 DIMENSION は (*n*)。解かれたばかりの部分問題をソートされた順に再統合する置換が上書きされる。すなわち、*d*(*idxq*(*i* = 1, *n*)) は昇順となる。
info	INTEGER *info* = 0 の場合、正常に終了したことを示す。 *info* = -*i* < 0 の場合、*i* 番目の引数の値が不正であったことを示す。 *info* = 1 の場合、特異値が収束しなかったことを示す。

?lasd2

2 つの特異値のセットをソートされた単一のセットに併合する。?bdsdc で使用される。

```
call slasd2 ( nl, nr, sqre, k, d, z, alpha, beta, u, ldu, vt, ldvt,
              dsigma, u2, ldu2, vt2, ldvt2, idxp, idx, idxc, idxq,
              coltyp, info )
call dlasd2 ( nl, nr, sqre, k, d, z, alpha, beta, u, ldu, vt, ldvt,
              dsigma, u2, ldu2, vt2, ldvt2, idxp, idx, idxc, idxq,
              coltyp, info )
```

説明

ルーチン ?lasd2 は 2 つの特異値のセットをソートされた単一のセットに併合する。続いて、問題の大きさの収縮を試みる。収縮が起こるには 2 つの場合がある。2 個以上の特異値が互いに近い場合、または、*Z* ベクトルに微小成分がある場合である。そのような場合ごとに関連する永年方程式問題の次元は 1 だけ縮退される。

ルーチン ?lasd2 は ?lasd1 から呼び出される。

入力パラメータ

nl	INTEGER。上ブロックの行次元。*nl* ≥ 1
nr	INTEGER。下ブロックの行次元。*nr* ≥ 1
sqre	INTEGER *sqre* = 0 の場合、下ブロックは *nr* × *nr* 平方マトリックスである。 *sqre* = 1 の場合、下ブロックは *nr* × (*nr*+1) 矩形マトリックスである。二重対角マトリックスは *n* = *nl* + *nr* + 1個の行と *m* = *n* + *sqre* ≥ *n*個の列を持つ。
d	REAL (slasd2 の場合) DOUBLE PRECISION (dlasd2 の場合) 配列 DIMENSION は (*n*)。*d* には結合を行う 2 つの部分マトリックスの特異値を格納する。
alpha	REAL (slasd2 の場合) DOUBLE PRECISION (dlasd2 の場合) 追加した行に関連する対角成分。
beta	REAL (slasd2 の場合) DOUBLE PRECISION (dlasd2 の場合) 追加した行に関連する対角外成分。
u	REAL (slasd2 の場合) DOUBLE PRECISION (dlasd2 の場合) 配列 DIMENSION は (*ldu*, *n*)。*u* には、(1, 1), (*n1*, *n1*) と (*n1*+2, *n1*+2), (*n*,*n*) の隅にある 2 つの平方ブロック内の 2 個の部分マトリックスの左特異ベクトルを格納する。
ldu	INTEGER。配列 *u* のリーディング・ディメンジョン。 *ldu* ≥ *n*

ldu2 INTEGER。出力配列 *u2* のリーディング・ディメンジョン。*ldu2* ≥ *n*

vt REAL (slasd2 の場合)
DOUBLE PRECISION (dlasd2 の場合)
配列 DIMENSION は (*ldvt*, *m*)。*vt*' には、(1, 1), (*nl*+1, *nl*+1) と (*nl*+2, *nl*+2), (*m*, *m*) の隅にある 2 つの平方ブロック内の 2 個の部分マトリックスの右特異ベクトルを格納する。

ldvt INTEGER。配列 *vt* のリーディング・ディメンジョン。*ldvt* ≥ *m*

ldvt2 INTEGER。出力配列 *vt2* のリーディング・ディメンジョン。*ldvt2* ≥ *m*

idxp INTEGER
ワークスペース配列 DIMENSION は (*n*)。この配列には、収縮された *d* の値を配列の終わりに配置した置換を格納する。出力で、*idxp*(2:*k*) は収縮されていない *d* 値を指し、*idxp*(*k*+1:*n*) は収縮された特異値を指す。

idx INTEGER
ワークスペース配列 DIMENSION は (*n*)。この配列には *d* の内容を昇順にソートした置換を格納する。

coltyp INTEGER
ワークスペース配列 DIMENSION は (*n*)。ワークスペースとして、*u2* マトリックス内の列、または *vt2* マトリックス内の行を示すラベルを次のタイプから格納する。
1 : 上半分のみ非ゼロ
2 : 下半分のみ非ゼロ
3 : 密
4 : 収縮

idxq INTEGER
配列 DIMENSION は (*n*)。*d* に入っている 2 つの部分問題を昇順で別々にソートする置換。この置換の前半にある成分は、最初に 1 つの位置だけ後ろに動かされなければならない。また、後半にある成分は、最初にそれら値に *nl*+1 を加算しなければならない。

出力パラメータ

k
INTEGER。収縮されていないマトリックスの次元が格納される。これは関連する永年方程式の次数である。$1 \le k \le n$

d
d には、後続の (*n*-*k*) 個の更新された特異値 (収縮された) が昇順で上書きされる。

u
u の最後の *n*-*k* 個の列には、後続の (*n*-*k*) 個の更新された特異値 (収縮された) が上書きされる。

z
REAL (slasd2 の場合)
DOUBLE PRECISION (dlasd2 の場合)
配列 DIMENSION は (*n*)。*z* は永年方程式内の更新列ベクトルで上書きされる。

dsigma
REAL (slasd2 の場合)
DOUBLE PRECISION (dlasd2 の場合)
配列 DIMENSION は (*n*)。永年方程式内の対角成分 (*k*-1 個の特異値と 1 つのゼロ) のコピーが格納される。

u2
REAL (slasd2 の場合)
DOUBLE PRECISION (dlasd2 の場合)
配列 DIMENSION は (*ldu2*, *n*)。最初の *k*-1 個の左特異ベクトルのコピーが格納される。これは新しい左特異ベクトルを解くために ?lasd3 でのマトリックス乗算 (?gemm) で使用される。*u2* は 4 つのブロックに配置される。第 1 のブロックは列 *nl*+1 が 1 で他はゼロである。第 2 のブロックには *nl* とその上のみに非ゼロ成分が格納される。第 3 のブロックには *nl*+1 よりも下のみに非ゼロ成分が格納される。第 4 のブロックは密。

vt

vt' の最後の *n*-*k* 個の列は、後続の (*n*-*k*) 個の更新された右特異値 (収縮された) で上書きされる。*sqre* = 1 の場合、*vt* の最後の行は右ヌル空間にかかる。

vt2

REAL (slasd2 の場合)

DOUBLE PRECISION (dlasd2 の場合)

配列 DIMENSION は (*ldvt2*, *n*)。*vt2*' には最初の *k* 個の右特異ベクトルのコピーが格納される。これは新しい右特異ベクトルを解くために ?lasd3 でのマトリックス乗算 (?gemm) で使用される。*vt2* は 3 つのブロックに配置される。第 1 のブロックには *sigma* 内の特殊 0 対角成分に対応する行が格納される。第 2 のブロックには *nl* +1 とその前のみに非ゼロが格納される。第 3 のブロックには *nl* +2 とその後ろのみに非ゼロが格納される。

idxc

INTEGER

配列 DIMENSION は (*n*)。収縮された *U* マトリックスの列を 3 つのグループに配置する置換が格納される。第 1 のグループには *nl* とその上に非ゼロ成分が格納され、第 2 のグループには *nl*+2 よりも下のみに非ゼロ成分が格納され、第 3 は密である。

coltyp

coltyp(*i*)が*i*番目のタイプ列の次元となる次元4の配列で上書きされる。

info

INTEGER

info = 0 の場合、正常に終了したことを示す。

info = -*i* < 0 の場合、*i* 番目の引数の値が不正であったことを示す。

?lasd3

D と *Z* 内の値によって定義されている永年方程式に対して根のすべての平方根を求め、次にマトリックス乗算によって特異ベクトルを更新する。?bdsdc で使用される。

```
call slasd3 ( nl, nr, sqre, k, d, q, ldq, dsigma, u, ldu, u2, ldu2,
              vt, ldvt, vt2, ldvt2, idxc, ctot, z, info )
call dlasd3 ( nl, nr, sqre, k, d, q, ldq, dsigma, u, ldu, u2, ldu2,
              vt, ldvt, vt2, ldvt2, idxc, ctot, z, info )
```

説明

ルーチン ?lasd3 は、*D* と *Z* 内の値によって定義されている永年方程式に対して根のすべての平方根を求める。?lasd4 に対する適切な呼び出しを実行し、続いてマトリックス乗算によって特異ベクトルを更新する。ルーチン ?lasd3 は ?lasd1 から呼び出される。

入力パラメータ

nl	INTEGER。上ブロックの行次元。$nl \geq 1$
nr	INTEGER。下ブロックの行次元。$nr \geq 1$
sqre	INTEGER $sqre = 0$ の場合、下ブロックは $nr \times nr$ 平方マトリックスである。 $sqre = 1$ の場合、下ブロックは $nr \times (nr+1)$ 矩形マトリックスである。二重対角マトリックスは $n = nl + nr + 1$ 個の行と $m = n + sqre \geq n$ 個の列を持つ。
k	INTEGER。永年方程式の大きさで、$1 \leq k \leq n$

q
REAL (slasd3 の場合)
DOUBLE PRECISION (dlasd3 の場合)
ワークスペース配列 DIMENSION は少なくとも
(*ldq*, *k*)

ldq
INTEGER。配列 *q* のリーディング・ディメンジョン。
ldq ≥ *k*

dsigma
REAL (slasd3 の場合)
DOUBLE PRECISION (dlasd3 の場合)
配列 DIMENSION は (*k*)。この配列の先頭の *k* 個の成分には収縮された更新問題の古い根を格納する。これらは永年方程式の極である。

u
REAL (slasd3 の場合)
DOUBLE PRECISION (dlasd3 の場合)
配列 DIMENSION は (*ldu*, *n*)。このマトリックスの最後の *n*-*k* 個の列には収縮された左特異ベクトルを格納する。

ldu
INTEGER。配列 *u* のリーディング・ディメンジョン。
ldu ≥ *n*

u2
REAL (slasd3 の場合)
DOUBLE PRECISION (dlasd3 の場合)
配列 DIMENSION は (*ldu2*, *n*)。このマトリックスの先頭の *k* 個の列には分割問題に対する収縮されていない左特異ベクトルを格納する。

ldu2
INTEGER。配列 *u2* のリーディング・ディメンジョン。
ldu2 ≥ *n*

vt
REAL (slasd3 の場合)
DOUBLE PRECISION (dlasd3 の場合)
配列 DIMENSION は (*ldvt*, *m*)。*vt*' の最後の *m*-*k* 個の列には収縮された右特異ベクトルを格納する。

ldvt
INTEGER。配列 *vt* のリーディング・ディメンジョン。
ldvt ≥ *n*

vt2
REAL (slasd3 の場合)
DOUBLE PRECISION (dlasd3 の場合)
配列 DIMENSION は (*ldvt2*, *n*)。*vt2*' の先頭の *k* 個の列には分割問題に対する収縮されていない右特異ベクトルを格納する。

ldvt2 INTEGER。配列 *vt2* のリーディング・ディメンジョン。*ldvt2* ≥ *n*

idxc INTEGER

配列 DIMENSION は (*n*)。*u* の列 (および *vt* の行) を 3 つのグループに配置する置換。第 1 のグループには *nl*+1 とその上 (または前) のみに非ゼロ成分が格納される。第 2 のグループには *nl*+2 とその下 (または後ろ) にのみ非ゼロ成分が格納される。第 3 のグループは密である。ただし、*u* の最初の列と *vt* の最初の行は別々に扱われる。?lasd4 で求められた特異ベクトルの行は、マトリックス乗算が実行される前に、同様に置換されなければならない。

ctot INTEGER

配列 DIMENSION は (4)。*idxc* に記述されているとおり、*u* に含まれる列 (または *vt* に含まれる行) のタイプのそれぞれの合計。4 番目の列タイプは収縮された任意の列である。

z REAL (slasd3 の場合)

DOUBLE PRECISION (dlasd3 の場合)

配列 DIMENSION は (*k*)。この配列の先頭の *k* 個の成分には収縮調整された更新行ベクトルの成分を格納する。

出力パラメータ

d REAL (slasd3 の場合)

DOUBLE PRECISION (dlasd3 の場合)

配列 DIMENSION は (*k*)。永年方程式の根の平方根が昇順で上書きされる。

info INTEGER

info = 0 の場合、正常に終了したことを示す。

info = -*i* < 0 の場合、*i* 番目の引数の値が不正であったことを示す。

info = 1 の場合、特異値が収束しなかったことを示す。

アプリケーション・ノート

このコードは浮動小数点演算に対してきわめて緩い仮定を行う。加減算ではガード桁を持つマシン、または Cray X-MP、Cray Y-MP、Cray C-90、または Cray-2 のような減算のガード桁を持たないバイナリ・マシンで動作する。ガード桁を持たない 16 進マシンや 10 進マシンでは異常終了することが考えられるが、そのような前例はない

?lasd4

正値対角マトリックスに対する正値対称階数 1 更新の、*i* 番目の更新された固有値の平方根を計算する。?bdsdc で使用される。

```
call slasd4 ( n, i, d, z, delta, rho, sigma, work, info )
call dlasd4 ( n, i, d, z, delta, rho, sigma, work, info )
```

説明

このルーチンは、正値対角マトリックスに対する正値対称階数 1 の更新の、*i* 番目の更新された固有値の平方根を計算する。正値対角マトリックスの成分は、配列 *d* の中の対応する成分の平方として与えられ、$i<j$ では $0 \le d(i) < d(j)$、および *rho* > 0 である。これは呼び出しルーチンによって配置され、一般性が失われることはない。階数 1 の更新された連立方程式はゆえに、

diag(*d*)* diag(*d*) + *rho* *Z * Z_transpose

Z のユークリッド・ノルムを 1 とする。この方法には、単純補間の有利関数を用いた永久方程式内の有利関数の近似が含まれる。

入力パラメータ

n INTEGER。全配列の長さ。

i
INTEGER。計算する固有値のインデックス。
$1 \le i \le n$

d
REAL (slasd4 の場合)
DOUBLE PRECISION (dlasd4 の場合)
配列 DIMENSION は (*n*)
元の固有値。$i < j$ では $0 \le d(i) < d(j)$ のように順序どおりに並べられていると仮定される。

z
REAL (slasd4 の場合)
DOUBLE PRECISION (dlasd4 の場合)
配列 DIMENSION は (*n*)
更新ベクトルの成分。

rho
REAL (slasd4 の場合)
DOUBLE PRECISION (dlasd4 の場合)
対称更新式におけるスカラー。

work
REAL (slasd4 の場合)
DOUBLE PRECISION (dlasd4 の場合)
ワークスペース配列 DIMENSION は (*n*)
$n \ne 1$ の場合、*work* の j 番目の成分に ($d(j)$ + *sigma_i*) を格納する。$n = 1$ の場合、*work*(1) = 1

出力パラメータ

delta
REAL (slasd4 の場合)
DOUBLE PRECISION (dlasd4 の場合)
配列 DIMENSION は (*n*)
$n \ne 1$ の場合、*delta* の j 番目の成分に ($d(j)$ - *sigma_i*) が格納される。$n = 1$ の場合、*delta*(1) = 1。ベクトル *delta* には(特異)固有ベクトルを構成するために必要な情報が格納される。

sigma
REAL (slasd4 の場合)
DOUBLE PRECISION (dlasd4 の場合)
計算された λ_i、i 番目の更新された固有値。

info
INTEGER
= 0 の場合、正常に終了したことを示す。
> 0 の場合で *info* = 1 ならば、更新処理に失敗したことを示す。

?lasd5

2 × 2 対角マトリックスの正値対称階数 1
更新から、*i* 番目の固有値の平方根を
計算する。?bdsdc で使用される。

```
call slasd5 ( i, d, z, delta, rho, dsigma, work )
call dlasd5 ( i, d, z, delta, rho, dsigma, work )
```

説明

このルーチンは、2 × 2 対角マトリックスの正値対称階数 1 更新から、*i* 番目の固有値の平方根を計算する。

diag(*d*)* diag(*d*) + *rho* *Z * Z_transpose

配列 *d* の対角成分は $i < j$ において $0 \le d(i) < d(j)$ を満たすものとする。合わせて、*rho* > 0、ベクトル Z のユークリッド・ノルムは 1 であるとする。

入力パラメータ

i INTEGER。計算される固有値のインデックス。
i = 1 または *i* = 2

d REAL (slasd5 の場合)
DOUBLE PRECISION (dlasd5 の場合)
配列 DIMENSION は (2)
元の固有値。$0 \le d(1) < d(2)$ とする。

z REAL (slasd5 の場合)
DOUBLE PRECISION (dlasd5 の場合)
配列 DIMENSION は (2)
更新ベクトルの成分。

rho REAL (slasd5 の場合)
DOUBLE PRECISION (dlasd5 の場合)
対称更新式におけるスカラー。

work REAL (slasd5 の場合)
DOUBLE PRECISION (dlasd5 の場合)
ワークスペース配列 DIMENSION は (2)
work の *j* 番目の成分には (*d*(*j*) + *sigma_i*) を格納する。

出力パラメータ

delta REAL (slasd5 の場合)
DOUBLE PRECISION (dlasd5 の場合)
配列 DIMENSION は (2)
delta の *j* 番目の成分に ($d(j) - \lambda_i$) が格納される。ベクトル *delta* には固有ベクトルを構成するために必要な情報が格納される。

dsigma REAL (slasd5 の場合)
DOUBLE PRECISION (dlasd5 の場合)
計算された λ_i、*i* 番目の更新された固有値。

?lasd6

2 つの小さいマトリックスを行追加によって併合して得た、更新された上二重対角マトリックスの *SVD* を計算する。
?bdsdc で使用される。

```
call slasd6 ( icompq, nl, nr, sqre, d, vf, vl, alpha, beta, idxq,
              perm, givptr, givcol, ldgcol, givnum, ldgnum, poles,
              difl, difr, z, k, c, s, work, iwork, info)
call dlasd6 ( icompq, nl, nr, sqre, d, vf, vl, alpha, beta, idxq,
              perm, givptr, givcol, ldgcol, givnum, ldgnum, poles,
              difl, difr, z, k, c, s, work, iwork, info)
```

説明

ルーチン ?lasd6 は、2 つの小さいマトリックスを行追加によって併合して得た、更新された上二重対角マトリックス *B* の *SVD* を計算する。このルーチンは、特異値すべてと、オプションで因子分解形式の特異ベクトル・マトリックスを必要とする問題でのみ使用される。*B* は $n \times m$ のマトリックスで、$n = nl + nr + 1$ かつ $m = n + sqre$ である。関連サブルーチン ?lasd1 は、二重対角マトリックスのすべての特異値と特異ベクトルが必要な場合を扱う。?lasd6 は次のように *SVD* を計算する。

$$B = U(in) * \begin{bmatrix} D1(in) & 0 & 0 & 0 \\ Z1' & a & Z2' & b \\ 0 & 0 & D2(in) & 0 \end{bmatrix} * VT(in)$$

$$= U(out) * (D(out) \;\; 0) * VT(out)$$

$Z = (Z1'\ a\ Z2'\ b) = u'\ VT'$、$u$ は次元 m のベクトルで、nl+1 番目と nl+2 番目の成分に *alpha* と *beta* を持ち他はゼロである。$sqre = 0$ の場合、成分 *b* は空である。

B の特異値は、下ブロックの全右特異ベクトルの最初の成分 *D1* と、上ブロックの全右特異ベクトルの最後の成分 *D2* を使って計算できる。これら成分は ?lasd6 の中で、それぞれ *vf* と *vl* に格納され更新される。そのため *U* と *VT* は明示的には参照されない。

特異値は *D* に格納される。アルゴリズムは 2 つの過程で構成される。第 1 の過程では、複数の特異値がある場合、または *Z* ベクトルにゼロがある場合、問題の大きさの収縮を行う。そのような場合ごとに永年方程式問題の次元は 1 だけ縮小される。この過程はルーチン ?lasd7 によって実行される。

第 2 の過程では更新された特異値を計算する。これは、ルーチン ?lasd4 (?lasd8 として呼び出される) を介して永年方程式の根を見つけることによって行われる。また、このルーチンは、*vf* と *vl* を更新し、更新された特異値と古い特異値の距離を計算する。?lasd6 は ?lasda から呼び出される。

入力パラメータ

icompq INTEGER。因子分解形式で特異ベクトルを計算するかどうかを指定する。
= 0 の場合、特異値のみ計算する。
= 1 の場合、合わせて特異ベクトルを因子分解形式で計算する。

nl INTEGER。上ブロックの次元。*nl* ≥ 1

nr INTEGER。下ブロックの次元。*nr* ≥ 1

sqre INTEGER
= 0 の場合、下ブロックは *nr* × *nr* 平方マトリックスである。
= 1 の場合、下ブロックは *nr* × (*nr*+1) 矩形マトリックスである。
二重対角マトリックスは行次元 *n* = *nl* + *nr* + 1 と列次元 *m* = *n* + *sqre* を持つ。

d REAL (slasd6 の場合)
DOUBLE PRECISION (dlasd6 の場合)
配列 DIMENSION は (*nl*+*nr*+1)。*d*(1:*nl*, 1:*nl*) には上ブロックの特異値を格納し、*d*(*nl*+2:*n*) には下ブロックの特異値を格納する。

vf REAL (slasd6 の場合)
DOUBLE PRECISION (dlasd6 の場合)
配列 DIMENSION は (*m*)。*vf*(1:*nl*+1) には上ブロックの全右特異ベクトルの最初の成分を格納し、*vf*(*nl*+2:*m*) には下ブロックの全右左特異ベクトルの最初の成分を格納する。

vl REAL (slasd6 の場合)
DOUBLE PRECISION (dlasd6 の場合)
配列 DIMENSION は (*m*)。*vl*(1:*nl*+1) には上ブロックの全右特異ベクトルの最後の成分を格納し、*vl*(*nl*+2:*m*) には下ブロックの全右左特異ベクトルの最後の成分を格納する。

alpha REAL (slasd6 の場合)
DOUBLE PRECISION (dlasd6 の場合)
追加した行に関連する対角成分。

beta REAL (slasd6 の場合)
DOUBLE PRECISION (dlasd6 の場合)
追加した行に関連する対角外成分。

ldgcol INTEGER。出力配列 *givcol* のリーディング・ディメンジョン。*n* 以上でなければならない。

ldgnum INTEGER。出力配列 *givnum*、*poles* のリーディング・ディメンジョン。*n* 以上でなければならない。

work REAL (slasd6 の場合)
DOUBLE PRECISION (dlasd6 の場合)
ワークスペース配列 DIMENSION は (4*m*)

iwork INTEGER
ワークスペース 配列 DIMENSION は (3*n*)

出力パラメータ

d *d*(1:*n*) は更新されたマトリックスの固有値で上書きされる。

vf *vf* は二重対角マトリックスの全右特異ベクトルの最初の成分で上書きされる。

vl *vl* は二重対角マトリックスの全右特異ベクトルの最後の成分で上書きされる。

idxq INTEGER
配列 DIMENSION は (*n*)。解かれたばかりの部分問題をソートされた順に再統合する置換で上書きされる。すなわち、*d*(*idxq*(*i* = 1, *n*)) は昇順となる。

perm INTEGER
配列 DIMENSION は (*n*)。各固有ブロックに適用される (収縮とソートによる) 置換が格納される。
icompq = 0 の場合は参照されない。

givptr
INTEGER。この部分問題で実行された Givens 回転の回数が格納される。*icompq* = 0 の場合は参照されない。

givcol
INTEGER
配列 DIMENSION は (*ldgcol*, 2)。それぞれの数のペアは Givens 回転で対象となる列ペアを示す。
icompq = 0 の場合は参照されない。

givnum
REAL (slasd6 の場合)
DOUBLE PRECISION (dlasd6 の場合)
配列 DIMENSION は (*ldgnum*, 2)。それぞれの値は対応する Givens 回転で使用される *C* 値または *S* 値を示す。*icompq* = 0 の場合は参照されない。

poles
REAL (slasd6 の場合)
DOUBLE PRECISION (dlasd6 の場合)
配列 DIMENSION は (*ldgnum*, 2)。配列 *poles*(1, *) には永年方程式を解いて得られた新しい特異値が格納される。配列 *poles*(2, *) には永年方程式内の極が格納される。*icompq* = 0 の場合は参照されない。

difl
REAL (slasd6 の場合)
DOUBLE PRECISION (dlasd6 の場合)
配列 DIMENSION は (*n*)。*difl*(*i*) には *i* 番目の更新された (収縮されていない) 特異値と *i* 番目の (収縮されていない) 古い特異値との距離が格納される。

difr
REAL (slasd6 の場合)
DOUBLE PRECISION (dlasd6 の場合)
配列、
DIMENSION は (*ldgnum*, 2) (*icompq* = 1)
DIMENSION は (*n*) (*icompq* = 0)
difr(*i*, 1) には i 番目の更新された (収縮されていない) 特異値と *i*+1 番目の (収縮されていない) 古い特異値との距離が格納される。*icompq* = 1 の場合、配列 *difr*(1:*k*, 2) には右特異ベクトル・マトリックスに対する正規化係数が格納される。
difl と *difl* の詳細は ?lasd8 を参照のこと。

z REAL (slasd6 の場合)
DOUBLE PRECISION (dlasd6 の場合)
配列 DIMENSION は (*m*)
この配列の先頭成分には収縮調整された更新行ベクトルが格納される。

k INTEGER。収縮されていないマトリックスの次元が格納される。これは関連する永年方程式の次数である。$1 \le k \le n$

c REAL (slasd6 の場合)
DOUBLE PRECISION (dlasd6 の場合)
sqre = 0 の場合はガーベッジが格納され、*sqre* = 1 の場合は右ヌル空間に関係する Givens 回転の *C* 値が *c* に格納される。

s REAL (slasd6 の場合)
DOUBLE PRECISION (dlasd6 の場合)
sqre = 0 の場合はガーベッジが格納され、*sqre* = 1 の場合は右ヌル空間に関係する Givens 回転の *S* 値が *s* に格納される。

info INTEGER
= 0 の場合、正常に終了したことを示す。
< 0 の場合、*info* = -*i* は *i* 番目の引数の値が不正だったことを示す。
> 0 の場合、*info* = 1 は固有値が収束しなかったことを示す。

?lasd7

2 つの特異値のセットをソートされた単一のセットに併合する。続いて、問題の大きさの収縮を試みる。?bdsdc で使用される。

```
call slasd7 ( icompq, nl, nr, sqre, k, d, z, zw, vf, vfw, vl, vlw,
              alpha, beta, dsigma, idx, idxp, idxq, perm, givptr,
              givcol, ldgcol, givnum, ldgnum, c, s, info )
call dlasd7 ( icompq, nl, nr, sqre, k, d, z, zw, vf, vfw, vl, vlw,
              alpha, beta, dsigma, idx, idxp, idxq, perm, givptr,
              givcol, ldgcol, givnum, ldgnum, c, s, info )
```

説明

ルーチン ?lasd7 は 2 つの特異値のセットをソートされた単一のセットに併合する。続いて、問題の大きさの収縮を試みる。収縮が起こり得るには 2 つの場合がある。2 個以上の特異値が互いに近い場合、または、Z ベクトルに微小成分がある場合である。そのような条件が発生するごとに、関連する永年方程式問題の次元は 1 だけ縮小される。?lasd7 は ?lasd6 から呼び出される。

入力パラメータ

icompq INTEGER。コンパクト形式で特異ベクトルを計算するかどうかを指定する。
= 0 の場合、特異値のみ計算する。
= 1 の場合、上二重対角マトリックスの特異ベクトルをコンパクト形式で計算する。

nl INTEGER。上ブロックの次元。$nl \geq 1$

nr INTEGER。下ブロックの次元。$nr \geq 1$

sqre INTEGER
= 0 の場合、下ブロックは $nr \times nr$ 平方マトリックスである。
= 1 の場合、下ブロックは $nr \times (nr+1)$ 矩形マトリックスである。二重対角マトリックスは行次元 $n = nl + nr + 1$ と列次元 $m = n + sqre \geq n$ を持つ。

d REAL (slasd7 の場合)
DOUBLE PRECISION (dlasd7 の場合)
配列 DIMENSION は (*n*)。*d* には結合対象の 2 つの部分マトリックスの特異値を格納する。

zw REAL (slasd7 の場合)
DOUBLE PRECISION (dlasd7 の場合)
配列 DIMENSION は (*m*)。*z* 用のワークスペース。

vf
REAL (slasd7 の場合)
DOUBLE PRECISION (dlasd7 の場合)
配列 DIMENSION は (*m*)。*vf*(1:*nl*+1) には上ブロックの全右特異ベクトルの最初の成分を格納し、*vf*(*nl*+2:*m*) には下ブロックの全右左特異ベクトルの最初の成分を格納する。

vfw
REAL (slasd7 の場合)
DOUBLE PRECISION (dlasd7 の場合)
配列 DIMENSION は (*m*)。*vf* 用のワークスペース。

vl
REAL (slasd7 の場合)
DOUBLE PRECISION (dlasd7 の場合)
配列 DIMENSION は (*m*)。*vl*(1:*nl*+1) には上ブロックの全右特異ベクトルの最後の成分を格納し、*vl*(*nl*+2:*m*) には下ブロックの全右左特異ベクトルの最後の成分を格納する。

vlw
REAL (slasd7 の場合)
DOUBLE PRECISION (dlasd7 の場合)
配列 DIMENSION は (*m*)。*vl* 用のワークスペース。

alpha
REAL (slasd7 の場合)
DOUBLE PRECISION (dlasd7 の場合)
追加した行に関連する対角成分。

beta
REAL (slasd7 の場合)
DOUBLE PRECISION (dlasd7 の場合)
追加した行に関連する対角外成分。

idx
INTEGER
ワークスペース配列 DIMENSION は (*n*)。*d* の内容を昇順でソートしたときに使用した置換を格納する。

idxp INTEGER
ワークスペース配列 DIMENSION は (*n*)。収縮した *d* の値を配列の終わりに配置した置換を格納する。

idxq INTEGER
配列 DIMENSION は (*n*)。*d* に入っている 2 つの部分問題を昇順で別々にソートする置換。この置換の前半にある成分は、最初に 1 つの位置だけ後ろに動かされなければならない。また、この置換の後半にある成分は、最初にそれら値に *nl*+1 を加算しなければならない。

ldgcol INTEGER。出力配列 *givcol* のリーディング・ディメンジョン。*n* 以上でなければならない。

ldgnum INTEGER。出力配列 *givnum* のリーディング・ディメンジョン。*n* 以上でなければならない。

出力パラメータ

k INTEGER。収縮されていないマトリックスの次元が格納される。これは関連する永年方程式の次数である。$1 \le k \le n$

d *d* は、後続の (*n*-*k*) 個の更新された特異値 (収縮された) の昇順で上書きされる。

z REAL (slasd7 の場合)
DOUBLE PRECISION (dlasd7 の場合)
配列 DIMENSION は (*m*)。*z* は永年方程式内の更新行ベクトルで上書きされる。

vf *vf* は二重対角マトリックスの全右特異値の最初の成分で上書きされる。

vl *vl* は二重対角マトリックスの全右特異値の最後の成分で上書きされる。

dsigma REAL (slasd7 の場合)
DOUBLE PRECISION (dlasd7 の場合)
配列 DIMENSION は (*n*)。永年方程式内の対角成分 (*k*-1 個の特異値と 1 個のゼロ) のコピーが格納される。

idxp *idxp*(2:*k*) は収縮されていない *d* 値を指し、*idxp*(*k*+1:*n*) は収縮された特異値を指す。

perm INTEGER
配列 DIMENSION は (*n*)。各固有ブロックに適用される (収縮とソートによる) 置換が格納される。*icompq* = 0 の場合は参照されない。

givptr INTEGER。この部分問題で実行された Givens 回転の回数が格納される。*icompq* = 0 の場合は参照されない。

givcol INTEGER
配列 DIMENSION は (*ldgcol*, 2)。それぞれの数のペアは Givens 回転で対象となる列ペアを示す。*icompq* = 0 の場合は参照されない。

givnum REAL (slasd7 の場合)
DOUBLE PRECISION (dlasd7 の場合)
配列 DIMENSION は (*ldgnum*, 2)。それぞれの値は対応する Givens 回転で使用される *C* 値または *S* 値を示す。*icompq* = 0 の場合は参照されない。

c REAL (slasd7 の場合)
DOUBLE PRECISION (dlasd7 の場合)
sqre = 0 の場合はガーベッジが格納され、*sqre* = 1 の場合は右ヌル空間に関係する Givens 回転の *C* 値が *c* に格納される。

s REAL (slasd7 の場合)
DOUBLE PRECISION (dlasd7 の場合)
sqre = 0 の場合はガーベッジが格納され、*sqre* = 1 の場合は右ヌル空間に関係する Givens 回転の *S* 値が *s* に格納される。

info INTEGER
= 0 の場合、正常に終了したことを示す。
< 0 の場合、*info* = -*i* は *i* 番目の引数の値が不正だったことを示す。

?lasd8

永年方程式の根の平方根を求め、*D* の各成分に対して最も近い 2 つの極までの距離を格納する。?bdsdc で使用される。

```
call slasd8 ( icompq, k, d, z, vf, vl, difl, difr, lddifr, dsigma,
              work, info )
call dlasd8 ( icompq, k, d, z, vf, vl, difl, difr, lddifr, dsigma,
              work, info )
```

説明

ルーチン ?lasd8 は、*dsigma* と *z* の値によって定義されている永年方程式に対して根の平方根を求める。?lasd4 に対する適切な呼び出しを実行し、続いて *d* の各成分に対して最も近い 2 つの極までの距離 (*dsigma* 内の成分) を格納する。また、元の二重対角マトリックスの全右特異ベクトルの最初と最後の成分である配列 *vf* と *vl* を更新する。?lasd8 は ?lasd6 から呼び出される。

入力パラメータ

icompq INTEGER。呼び出されたルーチン内で、因子分解形式で特異ベクトルを計算するかどうかを指定する。
= 0 の場合、特異値のみ計算する。
= 1 の場合、合わせて特異ベクトルを因子分解形式で計算する。

k INTEGER。?lasd4 で解かれる有利関数の項目数。
$k \geq 1$

z REAL (slasd8 の場合)
DOUBLE PRECISION (dlasd8 の場合)
配列 DIMENSION は (*k*)。この配列の先頭の *k* 個の成分には収縮調整された更新行ベクトルの成分を格納する。

vf REAL (slasd8 の場合)
DOUBLE PRECISION (dlasd8 の場合)
配列 DIMENSION は (*k*)。*vf* には dbede8 から渡された情報を格納する。

vl REAL (slasd8 の場合)
DOUBLE PRECISION (dlasd8 の場合)
配列 DIMENSION は (*k*)。*vl* には dbede8 から渡された情報を格納する。

lddifr INTEGER。出力配列 *difr* のリーディング・ディメンジョン。*k* 以上でなければならない。

dsigma REAL (slasd8 の場合)
DOUBLE PRECISION (dlasd8 の場合)
配列 DIMENSION は (*k*)。この配列の先頭の *k* 個の成分には収縮された更新問題の古い根を格納する。これらは永年方程式の極である。

work REAL (slasd8 の場合)
DOUBLE PRECISION (dlasd8 の場合)
ワークスペース配列 DIMENSION は少なくとも (3*k*)

出力パラメータ

d REAL (slasd8 の場合)
DOUBLE PRECISION (dlasd8 の場合)
配列 DIMENSION は (*k*)。*d* は更新された特異ベクトルで上書きされる。

vf *vf* は二重対角マトリックスの全右特異ベクトルの最初の成分の先頭の *k* 個の成分で上書きされる。

vl *vl* は二重対角マトリックスの全右特異ベクトルの最後の成分の先頭の *k* 個の成分で上書きされる。

difl REAL (slasd8 の場合)
DOUBLE PRECISION (dlasd8 の場合)
配列 DIMENSION は (*k*)。*difl*(*i*) = *d*(*i*)-*dsigma*(*i*) が格納される。

difr REAL (slasd8 の場合)
DOUBLE PRECISION (dlasd8 の場合)
配列、
icompq = 1 の場合、DIMENSION は (*lddifr*, 2) および
icompq = 0 の場合、DIMENSION は (*k*)
difr(*i*, 1) = *d*(*i*) - *dsigma*(*i*+1) が格納される。
difr(*k*, 1) は定義されてなく参照されない。
icompq = 1 の場合、配列 *difr*(1:*k*, 2) には右特異ベクトル・マトリックスに対する正規化係数が格納される。

info INTEGER
= 0 の場合、正常に終了したことを示す。
< 0 の場合、*info* = -*i* は *i* 番目の引数の値が不正だったことを示す。
> 0 の場合、*info* = 1 は固有値が収束しなかったことを示す。

?lasd9

永年方程式の根の平方根を求め、*D* の各成分に対して最も近い 2 つの極までの距離を格納する。?bdsdc で使用される。

```
call slasd9 ( icompq, ldu, k, d, z, vf, vl, difl, difr, sigma, work,
              info )
call dlasd9 ( icompq, ldu, k, d, z, vf, vl, difl, difr, dsigma, work,
              info )
```

説明

ルーチン ?lasd9 は、*dsigma* と *z* の値によって定義されている永年方程式に対して根の平方根を求める。?lasd4 に対する適切な呼び出しを実行し、続いて *d* の各成分に対して最も近い 2 つの極までの距離 (*dsigma* 内の成分) を格納する。また、元の二重対角マトリックスの全右特異ベクトルの最初と最後の成分である配列 *vf* と *vl* を更新する。?lasd9 は ?lasd7 から呼び出される。

入力パラメータ

icompq INTEGER。呼び出されたルーチン内で、因子分解形式で特異ベクトルを計算するかどうかを指定する。
icompq = 0 の場合、特異値のみ計算する。
icompq = 1 の場合、合わせて特異ベクトルを因子分解形式で計算する。

k INTEGER。slasd4 で解かれる有利関数の項目数。
$k \geq 1$

dsigma REAL (slasd9 の場合)
DOUBLE PRECISION (dlasd9 の場合)
配列 DIMENSION は (*k*)。この配列の先頭の *k* 個の成分には収縮された更新問題の古い根を格納する。これらは永年方程式の極である。

z REAL (slasd9 の場合)
DOUBLE PRECISION (dlasd9 の場合)
配列 DIMENSION は (*k*)。この配列の先頭の *k* 個の成分には収縮調整された更新行ベクトルの成分を格納する。

vf REAL (slasd9 の場合)
DOUBLE PRECISION (dlasd9 の場合)
配列 DIMENSION は (*k*)。*vf* には sbede8 から渡された情報を格納する。

vl REAL (slasd9 の場合)
DOUBLE PRECISION (dlasd9 の場合)
配列 DIMENSION は (*k*)。*vl* には sbede8 から渡された情報を格納する。

work REAL (slasd9 の場合)
DOUBLE PRECISION (dlasd9 の場合)
ワークスペース配列 DIMENSION は少なくとも (3*k*)

出力パラメータ

d
REAL (slasd9 の場合)
DOUBLE PRECISION (dlasd9 の場合)
配列 DIMENSION は (*k*)。*d*(*i*) は更新された特異ベクトルで上書きされる。

vf
vf は二重対角マトリックスの全右特異ベクトルの最初の成分の先頭の *k* 個の成分で上書きされる。

vl
vl は二重対角マトリックスの全右特異ベクトルの最後の成分の先頭の *k* 個の成分で上書きされる。

difl
REAL (slasd9 の場合)
DOUBLE PRECISION (dlasd9 の場合)
配列 DIMENSION は (*k*)
difl(*i*) = *d*(*i*)-*dsigma*(*i*) が格納される。

difr
REAL (slasd9 の場合)
DOUBLE PRECISION (dlasd9 の場合)
配列、
icompq = 1 の場合、DIMENSION は (*ldu*, 2) および
icompq = 0 の場合、DIMENSION は (*k*)
difr(*i*, 1) = *d*(*i*)- *dsigma*(*i*+1) が格納される。
difr(*k*, 1) は定義されてなく参照されない。
icompq = 1 の場合、配列 *difr*(1:*k*, 2) には右特異ベクトル・マトリックスに対する正規化係数が格納される。

info
INTEGER
= 0 の場合、正常に終了したことを示す。
< 0 の場合、*info* = -*i* は *i* 番目の引数の値が不正だったことを示す。
> 0 の場合、*info* = 1 は固有値が収束しなかったことを示す。

?lasda

対角 *d* と対角外 *e* を持つ実数上二重対角マトリックスの特異値分解 (*SVD*) を計算する。

```
 call slasda ( icompq, smlsiz, n, sqre, d, e, u, ldu, vt, k, difl,
               difr, z, poles, givptr, givcol, ldgcol, perm, givnum,
               c, s, work, iwork, info )
call dlasda ( icompq, smlsiz, n, sqre, d, e, u, ldu, vt, k, difl,
               difr, z, poles, givptr, givcol, ldgcol, perm, givnum,
               c, s, work, iwork, info )
```

説明

ルーチン ?lasda は、分割統治法を使って、対角 *d* と対角外 *e* を持つ実数上二重対角 $n \times m$ マトリックス *B* の特異値分解 (*SVD*) を計算する。ここで $m = n+sqre$。このアルゴリズムは *SVD* で特異値 $B = U*S*VT$ を計算する。直交マトリックス *U* と *VT* の計算はオプションであり、コンパクト形式で計算される。関連するサブルーチン ?lasd0 は特異値と特異ベクトルを黙示的な形式で計算する。

入力パラメータ

icompq INTEGER。コンパクト形式で特異ベクトルを計算するかどうかを指定する。
= 0 の場合、特異値のみ計算する。
= 1 の場合、上二重対角マトリックスの特異ベクトルをコンパクト形式で計算する。

smlsiz INTEGER。計算ツリーの一番下の部分問題の最大サイズ。

n INTEGER。上二重対角マトリックスの行次元。これはまた、主対角配列 *d* の次元でもある。

sqre INTEGER。二重対角マトリックスの列のサイズを指定する。
sqre = 0 の場合、二重対角マトリックスは列次元 *m* = *n* を持つ。
sqre = 1 の場合、二重対角マトリックスは列次元 *m* = *n* + 1 を持つ。

d REAL (slasda の場合)
DOUBLE PRECISION (dlasda の場合)
配列 DIMENSION は (*n*)。*d* には二重対角マトリックスの主対角を格納する。

e REAL (slasda の場合)
DOUBLE PRECISION (dlasda の場合)
配列 DIMENSION は (*m* -1)。二重対角マトリックスの劣対角成分を格納する。出力において内容は壊される。

ldu INTEGER。配列 *u*、*vt*、*difl*、*difr*、*poles*、*givnum*、*z* のリーディング・ディメンジョン。
ldu ≥ *n*

ldgcol INTEGER。配列 *givcol*、*perm* のリーディング・ディメンジョン。*ldgcol* ≥ *n*

work REAL (slasda の場合)
DOUBLE PRECISION (dlasda の場合)
ワークスペース配列 DIMENSION は
$(6n + (smlsiz + 1)^2)$

iwork INTEGER
ワークスペース配列 DIMENSION は少なくとも (7*n*) でなければならない。

出力パラメータ

d *info* = 0 の場合、二重対角マトリックスの特異値が格納される。

u REAL (slasda の場合)
DOUBLE PRECISION (dlasda の場合)
配列 DIMENSION は (*ldu*, *smlsiz*)、*icompq* = 1 の場合

icompq = 0 の場合は参照されない。
icompq = 1 の場合、u には一番下のレベルの全部分問題の左特異ベクトル・マトリックスが格納される。

vt
REAL (slasda の場合)
DOUBLE PRECISION (dlasda の場合)
配列、icompq = 1 の場合は DIMENSION は (ldu, smlsiz+1)。icompq = 0 の場合は参照されない。
icompq = 1 の場合、vt には一番下のレベルの全部分問題の左特異ベクトル・マトリックスが格納される。

k
INTEGER
配列、
icompq = 1 の場合、DIMENSION は (n)、および
icompq = 0 の場合、DIMENSION は (1)
icompq = 1 の場合、k(i) には計算ツリーにおける i 番目の永年方程式の次元が格納される。

difl
REAL (slasda の場合)
DOUBLE PRECISION (dlasda の場合)
配列 DIMENSION は (ldu, nlvl)、ここで
nlvl = floor ($\log_2$ (n/smlsiz))

difr
REAL (slasda の場合)
DOUBLE PRECISION (dlasda の場合)
配列、
icompq = 1 の場合、DIMENSION は (ldu, 2 nlvl)、
および icompq = 0 の場合、DIMENSION は (n)
icompq = 1 の場合、difl(1:n, i) と difr(1:n, 2i -1) には、i 番目のレベルの特異値と (i-1) 番目のレベルの特異値との距離が記録され、difr(1:n, 2i) には右特異ベクトル・マトリックスに対する正規化係数が格納される。詳細は ?lasd8 を参照のこと。

z
REAL (slasda の場合)
DOUBLE PRECISION (dlasda の場合)
配列、
icompq = 1 の場合、DIMENSION は (ldu, nlvl)、および
icompq = 0 の場合、DIMENSION は (n)

z(1, *i*) の最初の *k* 個の成分には、*i* 番目のレベルの部分問題に対する収縮調整された更新行ベクトルの成分が格納される。

poles
REAL (slasda の場合)
DOUBLE PRECISION (dlasda の場合)
配列、*icompq* = 1 の場合は DIMENSION は (*ldu*, 2***nlvl*)。*icompq* = 0 の場合は参照されない。*icompq* = 1 の場合、*poles*(1, 2*i*-1) と *poles*(1, 2*i*) には *i* 番目レベルの永年方程式に関係する新しい特異値と古い特異値が格納される。

givptr
INTEGER
配列、*icompq* = 1 の場合は DIMENSION は (*n*)
icompq = 0 の場合は参照されない。*icompq* = 1 の場合、*givptr*(*i*) には計算ツリーの *i* 番目問題で実行された Givens 回転の回数が格納される。

givcol
INTEGER
配列、*icompq* = 1 の場合は DIMENSION は (*ldgcol*, 2***nlvl*)。*icompq* = 0 の場合は参照されない。*icompq* = 1 の場合、*givcol*(1, 2*i*-1) と *givcol*(1, 2*i*) には計算ツリーの *i* 番目問題で実行された Givens 回転の位置が格納される。

perm
INTEGER
配列、*icompq* = 1 の場合は DIMENSION は (*ldgcol*, *nlvl*)。*icompq* = 0 の場合は参照されない。*icompq* = 1 の場合、*perm*(1, *i*) には計算ツリーの *i* 番目問題で実行された置換が格納される。

givnum
REAL (slasda の場合)
DOUBLE PRECISION (dlasda の場合)
配列、*icompq* = 1 の場合は DIMENSION は (*ldu*, 2***nlvl*)。*icompq* = 0 の場合は参照されない。*icompq* = 1 の場合、*givnum*(1, 2*i*-1) と *givnum*(1, 2*i*) には計算ツリーの *i* 番目問題で実行された Givens 回転の *C* 値と *S* 値が格納される。

c REAL (slasda の場合)
DOUBLE PRECISION (dlasda の場合)
配列、
icompq = 1 の場合、DIMENSION は (*n*)、および
icompq = 0 の場合、DIMENSION は (1)
icompq = 1 で *i* 番目の部分問題が正方ではない場合、*c*(*i*) には *i* 番目の部分問題の右ヌル空間に関係する Givens 回転の *C* 値が格納される。

s REAL (slasda の場合)
DOUBLE PRECISION (dlasda の場合)
配列、
icompq = 1 の場合、DIMENSION は (*n*)、および
icompq = 0 の場合、DIMENSION は (1)
icompq = 1 で *i* 番目の部分問題が正方ではない場合、*s*(*i*) には *i* 番目の部分問題の右ヌル空間に関係する Givens 回転の *S* 値が格納される。

info INTEGER
= 0 の場合、正常に終了したことを示す。
< 0 の場合、*info* = -*i* は *i* 番目の引数の値が不正だったことを示す。
> 0 の場合、*info* = 1 は固有値が収束しなかったことを示す。

?lasdq

対角 *d* と対角外 *e* を持つ実二重対角マトリックスを計算する。?bdsdc で使用される。

```
call slasdq ( uplo, sqre, n, ncvt, nru, ncc, d, e, vt, ldvt, u, ldu,
              c, ldc, work, info )
call dlasdq ( uplo, sqre, n, ncvt, nru, ncc, d, e, vt, ldvt, u, ldu,
              c, ldc, work, info )
```

説明

ルーチン ?lasdq は、対角 *d* と対角外 *e* を持つ実数 (上または下) 二重対角マトリックスの特異値分解 (*SVD*) を、必要に応じて変換の蓄積を行いながら計算する。*B* を入力の二重対角マトリックスとしたとき、このアルゴリズムは $B = Q\,S\,P'$ であるような直交マトリックス *Q* と *P* を計算する (*P'* は *P* の転置)。特異値 *S* は *d* に上書きされる。

必要に応じて入力マトリックス *U* は *UQ* に変更される。

必要に応じて入力マトリックス *VT* は *P' VT* に変更される。

必要に応じて入力マトリックス *C* は *Q' C* に変更される。

入力パラメータ

uplo CHARACTER*1。*uplo* は入力二重対角マトリックスが上二重対角か下二重対角かを指定する。
uplo = 'U' または 'u' の場合、*B* は上二重対角。
uplo = 'L' または 'l' の場合、*B* は下二重対角。

sqre INTEGER
= 0 の場合、入力マトリックスは $n \times n$
= 1 の場合、*uplu* = 'U' ならば入力マトリックスは $n \times (n+1)$、*uplu* = 'L' ならば $(n+1) \times n$。二重対角マトリックスは $n = nl + nr + 1$ の行と $m = n + sqre \geq n$ の列を持つ。

n INTEGER。*n* はマトリックス内の行数と列数を指定する。
n は 0 以上でなければならない。

ncvt INTEGER。*ncvt* はマトリックス *VT* の列数を指定する。
ncvt は 0 以上でなければならない。

nru INTEGER。*nru* はマトリックス *U* の行数を指定する。
nru は 0 以上でなければならない。

ncc INTEGER。*ncc* はマトリックス *C* の列数を指定する。
ncc は 0 以上でなければならない。

d REAL (slasdq の場合)
DOUBLE PRECISION (dlasdq の場合)
配列 DIMENSION は (*n*)。*d* には、*SVD* の実行対象である二重対角マトリックスの対角成分を格納する。

e
REAL (slasdq の場合)
DOUBLE PRECISION (dlasdq の場合)
配列 DIMENSION は、*sqre* = 0 の場合は (*n*-1)、*sqre* = 1 の場合は (*n*)。*e* の成分には、*SVD* の実行対象である二重対角マトリックスの対角外成分を格納する。

vt
REAL (slasdq の場合)
DOUBLE PRECISION (dlasdq の場合)
配列 DIMENSION は (*ldvt*, *ncvt*)。出力時に *P'* によって事前乗算されるマトリックスを格納する。
sqre = 0 ならば次元は *n* × *ncvt*、*sqre* = 1 ならば次元は (*n*+1) × *ncvt* (*ncvt* = 0 の場合は参照されない)。

ldvt
INTEGER。呼び出し (サブ) プログラムで宣言されたとおり、*ldvt* には *vt* のリーディング・ディメンジョンを格納する。
ldvt の値は、少なくとも 1 でなければならない。さらに *ncvt* が非ゼロならば、*ldvt* は少なくとも *n* でなければならない。

u
REAL (slasdq の場合)
DOUBLE PRECISION (dlasdq の場合)
配列 DIMENSION は (*ldu*, *n*)。出力時に *Q* によって事後乗算されるマトリックスを格納する。*sqre* = 0 ならば次元は *nru* × *n*、*sqre* = 1 ならば次元は *nru* × (*n*+1) (*nru* = 0 の場合は参照されない)。

ldu
INTEGER。呼び出し (サブ) プログラムで宣言されたとおり、*ldu* には *u* のリーディング・ディメンジョンを格納する。
ldu は max(1, *nru*) 以上でなければならない。

c
REAL (slasdq の場合)
DOUBLE PRECISION (dlasdq の場合)
配列 DIMENSION は (*ldc*, *ncc*)。出力時に *Q'* によって事前乗算される *n* × *ncc* のマトリックスを格納する。
sqre = 0 ならば次元は *n* × *ncc*、*sqre* = 1 ならば次元は (*n*+1) × *ncc* (*ncc* = 0 の場合は参照されない)。

ldc
INTEGER。呼び出し (サブ) プログラムで宣言されたとおり、*ldc* には *c* のリーディング・ディメンジョンを格納する。
ldc は 1 以上でなければならない。さらに *ncc* が非ゼロならば、*ldc* は少なくとも *n* でなければならない。

work REAL (slasdq の場合)
DOUBLE PRECISION (dlasdq の場合)
配列 DIMENSION は (4*n*)。ワークスペース配列。
ncvt、*nru*、または *ncc* のうち 1 つが非ゼロで、かつ、*n* が少なくとも 2 の場合にのみ参照される。

出力パラメータ

d *d* には特異値が昇順で上書きされる。

e 通常終了時は *e* には *0* が格納される。アルゴリズムが収束しない場合、*d* と *e* には、入力として与えられたマトリックスと直交的に等価な二重対角マトリックスの対角と劣対角成分が格納される。

vt 終了時に、マトリックスは P' によって事前に乗算されている。

u 終了時に、マトリックスは Q によって事後に乗算されている。

c 終了時に、マトリックスは Q' によって事前に乗算されている。

info INTEGER。ゼロは正常に終了したことを示す。
info < 0 の場合、-*info* 番目の引数の値が不正であったことを示す。*info* > 0 の場合、アルゴリズムが収束しなかったことを示し、*info* は収束しなかった優対角の個数を表わす。

?lasdt

二重対角分割統治に対する部分問題の
ツリーを生成する。?bdsdc で使用される。

```
call slasdt ( n, lvl, nd, inode, ndiml, ndimr, msub )
call dlasdt ( n, lvl, nd, inode, ndiml, ndimr, msub )
```

説明

このルーチンは二重対角分割統治に対する部分問題のツリーを生成する。

入力パラメータ

n INTEGER。二重対角マトリックスの対角成分の個数を格納する。

msub INTEGER。一番下のツリーにある各部分問題が取り得る最大行次元を格納する。

出力パラメータ

lvl INTEGER。計算ツリーのレベル数が格納される。

nd INTEGER。ツリーのノード数が格納される。

inode INTEGER
配列 DIMENSION は (*n*)。部分問題の中央が格納される。

ndiml INTEGER
配列 DIMENSION は (*n*)。左の子の列次元が格納される。

ndimr INTEGER
配列 DIMENSION は (*n*)。右の子の列次元が格納される。

?laset

マトリックスの対角外成分と対角成分を
指定された値に初期化する。

```
call slaset ( uplo, m, n, alpha, beta, a, lda )
call dlaset ( uplo, m, n, alpha, beta, a, lda )
call claset ( uplo, m, n, alpha, beta, a, lda )
call zlaset ( uplo, m, n, alpha, beta, a, lda )
```

説明

このルーチンは、$m \times n$ マトリックス A で、対角を *beta* に、対角外を *alpha* に初期化する。

入力パラメータ

uplo
CHARACTER*1。設定対象のマトリックスの部分を指定する。
uplo = 'U' の場合、上三角部分が設定される。A の厳密な下三角部分は変更されない。
uplo = 'L' の場合、下三角部分が設定される。A の厳密な上三角部分は変更されない。それ以外ではマトリックス A のすべての部分が設定される。

m
INTEGER。マトリックス A の行数。$m \geq 0$

n
INTEGER。マトリックス A の列数。$n \geq 0$

alpha, beta
REAL (slaset の場合)
DOUBLE PRECISION (dlaset の場合)
COMPLEX (claset の場合)
COMPLEX*16 (zlaset の場合)
それぞれ、対角外成分と対角成分に設定する定数。

a
REAL (slaset の場合)
DOUBLE PRECISION (dlaset の場合)
COMPLEX (claset の場合)
COMPLEX*16 (zlaset の場合)
配列 DIMENSION は (*lda*, *n*)
$m \times n$ のマトリックス A を格納する。

lda
INTEGER。配列 A のリーディング・ディメンジョン。
lda $\geq \max(1, m)$

出力パラメータ

a
A の先頭の $m \times n$ 部分マトリックスは次のように設定される。
uplo = 'U' の場合、$A(i, j) =$ *alpha*、$1 \leq i \leq j\text{-}1$、$1 \leq j \leq n$、
uplo = 'L' の場合、$A(i, j) =$ *alpha*、$j\text{+}1 \leq i \leq m$、$1 \leq j \leq n$、

それ以外の場合、$A(i, j) = alpha$、$1 \le i \le m$、
$1 \le j \le n, i \ne j$、
また、すべての uplo で、$A(i, i) = beta$、
$1 \le i \le \min(m, n)$

?lasq1

実数平方二重対角マトリックスの特異値を計算する。?bdsqr で使用される。

```
call slasq1 ( n, d, e, work, info )
call dlasq1 ( n, d, e, work, info )
```

説明

ルーチン ?lasq1 は、対角 d と対角外 e を持つ実数 $n \times n$ 二重対角マトリックスの特異値を計算する。特異値は相対的に高い精度で計算され、非正規化、アンダーフロー、オーバフローは発生しない。

入力パラメータ

n	INTEGER。マトリックス内の行と列の数。$n \ge 0$
d	REAL (slasq1 の場合) DOUBLE PRECISION (dlasq1 の場合) 配列 DIMENSION は (n)。d には、*SVD* の実行対象である二重対角マトリックスの対角成分を格納する。
e	REAL (slasq1 の場合) DOUBLE PRECISION (dlasq1 の場合) 配列 DIMENSION は (n)。成分 e(1:n-1) には、*SVD* の実行対象である二重対角マトリックスの対角外成分を格納する。

work REAL (slasq1 の場合)
DOUBLE PRECISION (dlasq1 の場合)
ワークスペース配列 DIMENSION は (4*n*)

出力パラメータ

d *d* には特異値が降順で上書きされる。

e *e* は上書きされる。

info INTEGER
= 0 の場合、正常に終了したことを示す。
< 0 の場合、*info* = -*i* は *i* 番目の引数の値が不正だったことを示す。
> 0 の場合、アルゴリズムが失敗したことを示す。
= 1 の場合、分割は *e* に格納されている正の値でマークされた。
= 2 の場合、*z* の現在のブロックは 30*n* 回の反復 (内側の while ループ) によって対角化されなかった。
= 3 の場合、外側の while ループの終了条件を満たさなかった (プログラムは *n* を超える縮退されていないブロックを生成した)。

?lasq2

相対的に高い精度で、*qd* 配列 *z* に関係する対称正値三重対角マトリックスのすべての固有値を計算する。?bdsqr と ?stegr で使用される。

```
call slasq2 ( n, z, info )
call dlasq2 ( n, z, info )
```

説明

ルーチン ?lasd2 は、*qd* 配列 *z* に関係する対称正値三重対角マトリックスのすべての固有値を計算する。固有値は相対的に高い精度で計算され、非正規化、アンダーフロー、オーバフローは発生しない。

三重対角マトリックスに対する *z* の関係を理解するには、*L* を劣対角 *z*(2, 4, 6, ...) を持つ単位下二重対角マトリックスとし、*U* を対角 *z*(1, 3, 5, ...) とその上に 1 を持つ上二重対角マトリックスとする。三重対角は *LU* となる。または、類似した対称三重対角としてもよい。

入力パラメータ

n INTEGER。マトリックス内の行数と列数。$n \geq 0$

z REAL (slasq2 の場合)
DOUBLE PRECISION (dlasq2 の場合)
配列 DIMENSION は (4*n*)。*z* には *qd* 配列を格納する。

出力パラメータ

z 成分 1 から *n* には固有値が降順で、*z*(2*n*+1) には対角和が、*z*(2*n*+2) には固有値の合計が格納される。
n > 2 の場合、*z*(2*n*+3) には反復回数が、*z*(2*n*+4) には ndivs/nin^2 が、*z*(2*n*+5) には失敗したシフトのパーセントが格納される。

info INTEGER
= 0 の場合、正常に終了したことを示す。
< 0 の場合、*i* 番目の引数がスカラーで値が不正だった場合は *info* = -*i*、*i* 番目の引数が配列で *j* 番目の成分の値が不正だった場合は *info* = -(*i**100+*j*)
> 0 の場合、アルゴリズムが失敗したことを示す。
= 1 の場合、分割は *e* に格納されている正値でマークされた。
= 2 の場合、*z* の現在のブロックは 30**n* 回の反復 (内側 while ループ) によって対角化されなかった。
= 3 の場合、外側 while ループの終了条件を満たさなかった (プログラムは *n* を超える縮退されていないブロックを生成した)。

アプリケーション・ノート

ルーチン ?lasq2 は論理変数 *ieee* を定義する。マシンが IEEE-754 浮動小数点スタンダードに従って無限および NaN を取り扱う場合は .TRUE. となり、そうでない場合は .FALSE. となる。変数は ?lasq3 に渡される。

?lasq3

収縮をチェックし、シフトを計算し、
dqds を呼び出す。?bdsqr で使用される。

```
call slasq3 ( i0, n0, z, pp, dmin, sigma, desig, qmax, nfail, iter,
              ndiv, ieee )
call dlasq3 ( i0, n0, z, pp, dmin, sigma, desig, qmax, nfail, iter,
              ndiv, ieee )
```

説明

ルーチン ?lasq3 は、収縮をチェックし、シフト (*tau*) を計算し、*dqds* を呼び出す。失敗した場合はシフトを変更して出力が正になるまで繰り返す。

入力パラメータ

i0	INTEGER。最初のインデックス。
n0	INTEGER。最後のインデックス。
z	REAL (slasq3 の場合) DOUBLE PRECISION (dlasq3 の場合) 配列 DIMENSION は (4*n*)。*z* には *qd* 配列を格納する。
pp	INTEGER ping のとき *pp* = 0、pong のとき *pp* = 1
desig	REAL (slasq3 の場合) DOUBLE PRECISION (dlasq3 の場合) *sigma* の下位次数部分。

qmax REAL (slasq3 の場合)
DOUBLE PRECISION (dlasq3 の場合)
q の最大値。

ieee LOGICAL。IEEE または非 IEEE 演算を示すフラグ (?lasq5 に渡される)。

出力パラメータ

dmin REAL (slasq3 の場合)
DOUBLE PRECISION (dlasq3 の場合)
d の最小値。

sigma REAL (slasq3 の場合)
DOUBLE PRECISION (dlasq3 の場合)
現在のセグメントで使用されるシフトの合計。

desig *sigma* の下位次数部分。

nfail INTEGER。シフト回数が大きすぎたことを示す。

iter INTEGER。反復の回数。

ndiv INTEGER。除算の回数。

ttype INTEGER。シフトのタイプ。

?lasq4

以前の変換で得られた *d* の値を用いて
最小固有値に対する近似を計算する。
?bdsqr で使用される。

```
call slasq4 ( i0, n0, z, pp, n0in, dmin, dmin1, dmin2, dn, dn1, dn2,
              tau, ttype )
call dlasq4 ( i0, n0, z, pp, n0in, dmin, dmin1, dmin2, dn, dn1, dn2,
              tau, ttype )
```

説明

このルーチンは、以前の変換で得られた *d* の値を用いて、最小固有値に対する近似 *tau* を計算する。

入力パラメータ

i0	INTEGER。最初のインデックス。
n0	INTEGER。最後のインデックス。
z	REAL (slasq4 の場合) DOUBLE PRECISION (dlasq4 の場合) 配列 DIMENSION は (4*n*)。*z* には *qd* 配列を格納する。
pp	INTEGER。ping のとき *pp* = 0、pong のとき *pp* = 1
noin	INTEGER。*eigtest* の開始にある *n0* の値。
dmin	REAL (slasq4 の場合) DOUBLE PRECISION (dlasq4 の場合) *d* の最小値。
dmin1	REAL (slasq4 の場合) DOUBLE PRECISION (dlasq4 の場合) *d*(*n0*) を除く *d* の最小値。
dmin2	REAL (slasq4 の場合) DOUBLE PRECISION (dlasq4 の場合) *d*(*n0*) と *d*(*n0*-1) を除く *d* の最小値。
dn	REAL (slasq4 の場合) DOUBLE PRECISION (dlasq4 の場合) *d*(*n*) が格納される。
dn1	REAL (slasq4 の場合) DOUBLE PRECISION (dlasq4 の場合) *d*(*n*-1) が格納される。
dn2	REAL (slasq4 の場合) DOUBLE PRECISION (dlasq4 の場合) *d*(*n*-2) が格納される。

出力パラメータ

tau REAL (slasq4 の場合)
DOUBLE PRECISION (dlasq4 の場合)
シフトである。

ttype INTEGER。シフトのタイプ。

?lasq5

ping-pong 形式で *dqds* 変換を 1 つ計算する。?bdsqr と ?stegr で使用される。

```
call slasq5 ( i0, n0, z, pp, tau, dmin, dmin1, dmin2, dn, dnm1, dnm2,
              ieee )
call dlasq5 ( i0, n0, z, pp, tau, dmin, dmin1, dmin2, dn, dnm1, dnm2,
              ieee )
```

説明

このルーチンは ping-pong 形式で *dqds* 変換を 1 つ計算する。IEEE マシンと非 IEEE マシンに対応する。

入力パラメータ

i0 INTEGER。最初のインデックス。

n0 INTEGER。最後のインデックス。

z REAL (slasq5 の場合)
DOUBLE PRECISION (dlasq5 の場合)
配列 DIMENSION は (4*n*)。*z* には *qd* 配列を格納する。余分な引数を防ぐため *z*(4**n0*) に *emin* を格納する。

pp INTEGER。ping のとき *pp* = 0、pong のとき *pp* = 1

tau REAL (slasq5 の場合)
DOUBLE PRECISION (dlasq5 の場合)
シフトである。

ieee LOGICAL。IEEE または非 IEEE 演算を示すフラグ。

出力パラメータ

dmin REAL (slasq5 の場合)
DOUBLE PRECISION (dlasq5 の場合)
d の最小値。

dmin1 REAL (slasq5 の場合)
DOUBLE PRECISION (dlasq5 の場合)
d(*n0*) を除く *d* の最小値。

dmin2 REAL (slasq5 の場合)
DOUBLE PRECISION (dlasq5 の場合)
d(*n0*) と *d*(*n0*-1) を除く *d* の最小値。

dn REAL (slasq5 の場合)
DOUBLE PRECISION (dlasq5 の場合)
d の最後の値である *d*(*n0*) が格納される。

dnm1 REAL (slasq5 の場合)
DOUBLE PRECISION (dlasq5 の場合)
d(*n0*-1) が格納される。

dnm2 REAL (slasq5 の場合)
DOUBLE PRECISION (dlasq5 の場合)
d(*n0*-2) が格納される。

?lasq6

ping-pong 形式で *dqds* 変換を 1 つ
計算する。?bdsqr と ?stegr で
使用される。

```
call slasq6 ( i0, n0, z, pp, dmin, dmin1, dmin2, dn, dnm1, dnm2 )
call dlasq6 ( i0, n0, z, pp, dmin, dmin1, dmin2, dn, dnm1, dnm2 )
```

説明

ルーチン ?lasq6 は、アンダーフローとオーバーフローに対する保護を行いながら、ping-pong 形式で *dqd* (ゼロに等しいシフト) 変換を 1 つ計算する。

入力パラメータ

i0 INTEGER。最初のインデックス。

n0 INTEGER。最後のインデックス。

z REAL (slasq6 の場合)
DOUBLE PRECISION (dlasq6 の場合)
配列 DIMENSION は (4*n*)。*z* には *qd* 配列を格納する。余分な引数を防ぐため *z*(4**n0*) に *emin* を格納する。

pp INTEGER。ping のとき *pp* = 0、pong のとき *pp* = 1

出力パラメータ

dmin REAL (slasq6 の場合)
DOUBLE PRECISION (dlasq6 の場合)
d の最小値。

dmin1 REAL (slasq6 の場合)
DOUBLE PRECISION (dlasq6 の場合)
d(*n0*) を除く *d* の最小値。

dmin2 REAL (slasq6 の場合)
DOUBLE PRECISION (dlasq6 の場合)
d(*n0*) と *d*(*n0*-1) を除く *d* の最小値。

dn REAL (slasq6 の場合)
DOUBLE PRECISION (dlasq6 の場合)
d の最後の値である *d*(*n0*) が格納される。

dnm1 REAL (slasq6 の場合)
DOUBLE PRECISION (dlasq6 の場合)
d(*n0*-1) が格納される。

dnm2 REAL (slasq6 の場合)
DOUBLE PRECISION (dlasq6 の場合)
d(*n0*-2) が格納される。

?lasr

一般矩形マトリックスに一連の面回転を
適用する。

```
call slasr ( side, pivot, direct, m, n, c, s, a, lda )
call dlasr ( side, pivot, direct, m, n, c, s, a, lda )
call clasr ( side, pivot, direct, m, n, c, s, a, lda )
call zlasr ( side, pivot, direct, m, n, c, s, a, lda )
```

説明

このルーチンは次の変換を実行する。

side = 'L' または 'l' のとき $A := P\,A$ (左辺側)
side = 'R' または 'r' のとき $A := A\,P'$ (右辺側)

A は $m \times n$ の実数マトリックス、P は直交マトリックスで、パラメータ *pivot* と *direct* によって次のように定義される一連の面回転で構成される (*side* = 'L' または 'l' のとき $z = m$、*side* = 'R' または 'r' のとき $z = n$)。

direct = 'F' または 'f' のとき (順方向順)、

$$P = P(z - 1) \ldots P(2)\,P(1),$$

direct = 'B' または 'b' のとき (逆方向順)、

$$P = P(1)\,P(2) \ldots P(z - 1),$$

$P(k)$ は次の面に対する面回転マトリックスである。

pivot = 'V' または 'v' (可変ピボット) のとき、面 $(k,\ k+1)$
pivot = 'T' または 't' (上ピボット) のとき、面 $(1, k+1)$
pivot = 'B' または 'b' (下ピボット) のとき、面 $(k,\ z)$

$c(k)$ と $s(k)$ には $P(k)$ を定義する余弦と正弦が格納されていなければならない。マトリックス $P(k)$ の 2×2 面回転部分 $R(k)$ は、次の形式とみなす。

$$R(k) = \begin{bmatrix} c(k) & s(k) \\ -s(k) & c(k) \end{bmatrix}$$

入力パラメータ

side CHARACTER*1。面回転マトリックス P を A の左または右に適用するかを指定する。
= 'L' の場合、左、$A := P A$ を計算する。
= 'R' の場合、右、$A := A P'$ を計算する。

direct CHARACTER*1。P が面回転の順方向順または逆方向順かを指定する。
= 'F' の場合、順方向、$P = P(z - 1) \ldots P(2) P(1)$
= 'B' の場合、逆方向、$P = P(1) P(2) \ldots P(z - 1)$

pivot CHARACTER*1。面回転マトリックス $P(k)$ の面を指定する。
= 'V' の場合、可変ピボット、面 $(k, k+1)$
= 'T' の場合、上ピボット、面 $(1, k+1)$
= 'B' の場合、下ピボット、面 (k, z)

m INTEGER。マトリックス A の行数。
$m \leq 1$ の場合、イミディエイト・リターンが有効。

n INTEGER。マトリックス A の列数。
$n \leq 1$ の場合、イミディエイト・リターンが有効。

c, s REAL (slasr/clasr の場合)
DOUBLE PRECISION (dlasr/zlasr の場合)
配列 DIMENSION は、
side = 'L' の場合 (m-1)
side = 'R' の場合 (n-1)
$c(k)$ と $s(k)$ には上述のようにマトリックス $P(k)$ を定義する余弦と正弦を格納する。

a REAL (slasr の場合)
DOUBLE PRECISION (dlasr の場合)
COMPLEX (clasr の場合)
COMPLEX*16 (zlasr の場合)
配列 DIMENSION は (*lda*, *n*)。$m \times n$ のマトリックス A

lda INTEGER。配列 A のリーディング・ディメンジョン。
$lda \geq \max(1, m)$

出力パラメータ

a *side* = 'R' の場合は *PA*、*side* = 'L' の場合は *AP*' で上書きされる。

?lasrt

数を昇順または降順でソートする。

```
call slasrt ( id, n, d, info )
call dlasrt ( id, n, d, info )
```

説明

?lasrt は、*d* に格納されている数を昇順 (*id* = 'I' の場合) または降順 (*id* = 'D' の場合) でソートする。このルーチンはクイック・ソートを使用するが、*size* ≤ 20 の配列では挿入ソートが使われる。スタックの次元によって *n* はおよそ 2^{32} に制限される。

入力パラメータ

id CHARACTER*1
= 'I' の場合、*d* を昇順でソートする。
= 'D' の場合、*d* を降順でソートする。

n INTEGER。配列 *d* の長さ。

d REAL (slasrt の場合)
DOUBLE PRECISION (dlasrt の場合)
ソートする配列を格納する。

出力パラメータ

d *id* の設定によって、*d* は昇順 ($d(1) \le ... \le d(n)$) または降順 ($d(1) \ge ... \ge d(n)$) となる。

info INTEGER
= 0 の場合、正常に終了したことを示す。
< 0 の場合、*info* = -*i* は *i* 番目の引数の値が不正だったことを示す。

?lassq

スケーリング形式で表現された二乗和を
更新する。

```
call slassq ( n, x, incx, scale, sumsq )
call dlassq ( n, x, incx, scale, sumsq )
call classq ( n, x, incx, scale, sumsq )
call zlassq ( n, x, incx, scale, sumsq )
```

説明

実数ルーチン `slassq/dlassq` は、次のように値 *scl* と *smsq* を返す。

$$scl^2 * smsq = x(1)^2 + ... + x(n)^2 + scale^2 * sumsq$$

ここで、$x(i) = x(1 + (i - 1)\ incx)$
sumsq の値は非負であるとし、*scl* は次の値を返す。

$$scl = \max(scale, \mathrm{abs}(x(i)))$$

scale と *sumsq* は `scale` と `sumsq` によって与えられなければならない。
また *scl* と *smsq* は `scale` と `sumsq` にそれぞれ上書きされる。

複素ルーチン `classq/zlassq` は、次のように値 *scl* と *ssq* を返す。

$$scl^2 * ssq = x(1)^2 + ... + x(n)^2 + scale^2 * sumsq$$

ここで、$x(i) = \mathrm{abs}\ (x(1 + (i - 1)\ incx))$
sumsq の値は少なくともユニティであるとみなされ、*ssq* の値は、次のように満たされる。

$$1.0 \le ssq \le sumsq + 2n$$

scale の値は非負であるとし、*scl* は次の値を返す。

$$scl = \max(scale, \mathrm{abs}(\mathrm{real}(x(i))), \mathrm{abs}(\mathrm{aimag}(x(i))))$$

scale と *sumsq* は `scale` と `sumsq` によって与えられなければならない。
また *scl* と *ssq* は `scale` と `sumsq` にそれぞれ上書きされる。

`?lassq` の全ルーチンは唯一、ベクトル *x* を出力に渡す。

入力パラメータ

n
INTEGER。ベクトル *x* で使用する成分の個数。

x
REAL (slassq の場合)
DOUBLE PRECISION (dlassq の場合)
COMPLEX (classq の場合)
COMPLEX*16 (zlassq の場合)
スケーリングされた二乗和を計算するベクトル。
$x(i) = x(1 + (i - 1)\ incx),\ 1 \le i \le n$

incx
INTEGER。ベクトル *x* の連続する値の増分。*incx* > 0

scale
REAL (slassq/classq の場合)
DOUBLE PRECISION (dlassq/zlassq の場合)
上の式で示した *scale* の値を格納する。

sumsq
REAL (slassq/classq の場合)
DOUBLE PRECISION (dlassq/zlassq の場合)
上の式で示した *sumsq* の値を格納する。

出力パラメータ

scale
scale は、二乗和に対するスケール係数 *sc1* で上書きされる。

sumsq
実数型の場合。
sumsq は上の式で示した *smsq* の値で上書きされる。
複素型の場合。
sumsq は上の式で示した *ssq* の値で上書きされる。

?lasv2

2 × 2 三角マトリックスの特異値分解を
計算する。

```
call slasv2 ( f, g, h, ssmin, ssmax, snr, csr, snl, csl )
call dlasv2 ( f, g, h, ssmin, ssmax, snr, csr, snl, csl )
```

説明

ルーチン ?lasv2 は 2 × 2 三角マトリックスの特異値分解を計算する。

$$\begin{bmatrix} f & g \\ 0 & h \end{bmatrix}$$

出力において、abs(*ssmax*) には大きいほうの特異値が、abs(*ssmin*) には小さいほうの特異値が、(*csl*, *snl*) と (*csr*, *snr*) には abs(*ssmax*) に対する左と右の特異ベクトルが格納され、分解を与える。

$$\begin{bmatrix} csl & snl \\ -snl & csl \end{bmatrix} \begin{bmatrix} f & g \\ 0 & h \end{bmatrix} \begin{bmatrix} csr & -snr \\ snr & csr \end{bmatrix} = \begin{bmatrix} ssmax & 0 \\ 0 & ssmin \end{bmatrix}$$

入力パラメータ

f, g, h
: REAL (slasv2 の場合)
DOUBLE PRECISION (dlasv2 の場合)
それぞれ 2 × 2 マトリックスの (1,1)、(1,2)、(2,2) の成分。

出力パラメータ

ssmin, ssmax
: REAL (slasv2 の場合)
DOUBLE PRECISION (dlasv2 の場合)
それぞれ、abs(*ssmin*) と abs(*ssmax*) は小さいほうの固有値および大きいほうの特異値。

snl, csl	REAL (slasv2 の場合) DOUBLE PRECISION (dlasv2 の場合) ベクトル (*csl*, *snl*) は特異値 abs(*ssmax*) に対する単位左特異ベクトル。
snr, csr	REAL (slasv2 の場合) DOUBLE PRECISION (dlasv2 の場合) ベクトル (*csr*, *sn*) は特異値 abs(*ssmax*) に対する単位右特異ベクトル。

アプリケーション・ノート

入力パラメータは、出力パラメータにエイリアスされるときがある。
オーバーフロー / アンダーフローの場合を除いて、減算にガード桁がなくとも、すべての出力の大きさは最後の位置からわずかな単位以内 (ulp) で正確である。

IEEE 演算で、1 つのマトリックス成分が無限でもコードは正しく動作する。
最大特異値自身がオーバーフローしない限り、あるいは最大特異値がオーバーフローからわずかな ulp 範囲内にある限り、オーバーフローは起こらない。(Cray のように部分オーバーフローを持つマシンでは、最大特異値がオーバーフローに対して 2 の因数以内にあるとオーバーフローが起こることがある。)
アンダーフローは緩やかであればアンダーフローは無害である。そうでない場合、結果は、アンダーフローしきい値に近いサイズの摂動によって変更されたマトリックスに一致することがある。

?laswp

一般矩形マトリックスに対して一連の
行交換を実行する。

```
call slaswp ( n, a, lda, k1, k2, ipiv, incx )
call dlaswp ( n, a, lda, k1, k2, ipiv, incx )
call claswp ( n, a, lda, k1, k2, ipiv, incx )
call zlaswp ( n, a, lda, k1, k2, ipiv, incx )
```

説明

このルーチンはマトリックス *A* に一連の行交換を実行する。行交換は *A* の行 *k1* から *k2* のそれぞれに対して開始される、

入力パラメータ

n	INTEGER。マトリックス *A* の列数。
a	REAL (slaswp の場合) DOUBLE PRECISION (dlaswp の場合) COMPLEX (claswp の場合) COMPLEX*16 (zlaswp の場合) 配列 DIMENSION は (*lda*, *n*) 行交換を適用する列次元 *n* のマトリックスを格納する。
lda	INTEGER。配列 *a* のリーディング・ディメンジョン。
k1	INTEGER。*ipiv* の行交換を適用する最初の成分。
k2	INTEGER。*ipiv* の行交換を適用する最後の成分。
ipiv	INTEGER 配列 DIMENSION は (*m** abs(*incx*)) ピボット・インデックスのベクトル。*ipiv* の位置 *k1* から *k2* にある成分のみがアクセスされる。 *ipiv*(*k*) = 1 には行 *k* と *l* が交換される意味がある。
incx	INTEGER。*ipiv* の連続する値の増分。*ipiv* が負の場合、ピボットは逆順に適用される。

出力パラメータ

a	置換されたマトリックスで上書きされる。

?lasy2

マトリックスの次数が 1 または 2 の
シルベスター・マトリックス式を解く。

```
call slasy2 ( ltranl, ltranr, isgn, n1, n2, tl, ldtl, tr,ldtr, b, ldb,
              scale, x, ldx, xnorm, info )
call dlasy2 ( ltranl, ltranr, isgn, n1, n2, tl, ldtl, tr,ldtr, b, ldb,
              scale, x, ldx, xnorm, info )
```

説明

このルーチンは、次式で *n1* × *n2* マトリックス X を解く。
1 ≤ *n1*、*n2* ≤ 2

$$\mathrm{op}(TL) * X + isgn * X * \mathrm{op}(TR) = scale * B$$

ここで
TL は *n1* × *n1*、
TR は *n2* × *n2*、
B は *n1* × *n2*、
また、*isgn* = 1 または -1、op(*T*) = *T* または *T*'、ここで *T*' は *T* の転置を表わす。

入力パラメータ

ltranl LOGICAL。
ltranl は op(*TL*) を指定する。
= .FALSE., op(*TL*) = *TL*,
= .TRUE., op(*TL*) = *TL*'.

ltranr LOGICAL。
ltranr は op(TR) を指定する。
= .FALSE., op(*TR*) = *TR*,
= .TRUE., op(*TR*) = *TR*'.

isgn INTEGER。*isgn* は前述のとおり式の符号を指定する。
isgn は 1 または -1 を取り得る。

n1 INTEGER。*n1* はマトリックス *TL* の次数を指定する。
n1 は、0、1、2 を取り得る。

n2 INTEGER
n2 はマトリックス *TR* の次数を指定する。*n2* は、0、1、2 を取り得る。

t1 REAL (slasy2 の場合)
DOUBLE PRECISION (dlasy2 の場合)
配列 DIMENSION は (*ldt1*, 2)。*t1* には *n1* × *n1* のマトリックス *TL* を格納する。

ldt1 INTEGER。マトリックス *t1* のリーディング・ディメンジョン。*ldt1* ≥ max(1, *n1*)

tr REAL (slasy2 の場合)
DOUBLE PRECISION (dlasy2 の場合)
配列 DIMENSION は (*ldtr*, 2)。*tr* には *n2* × *n2* のマトリックス *TR* を格納する。

ldtr INTEGER
マトリックス *tr* のリーディング・ディメンジョン。*ldtr* ≥ max(1, *n2*)

b REAL (slasy2 の場合)
DOUBLE PRECISION (dlasy2 の場合)
配列 DIMENSION は (*ldb*, 2)。*n1* × *n2* のマトリックス *b* には式の右辺を格納する。

ldb INTEGER
マトリックス *b* のリーディング・ディメンジョン。*ldb* ≥ max(1, *n1*)

ldx INTEGER
出力マトリックス *x* のリーディング・ディメンジョン。*ldx* ≥ max(1, *n1*)

出力パラメータ

scale REAL (slasy2 の場合)
DOUBLE PRECISION (dlasy2 の場合)
終了時に、*scale* にはスケール係数が格納される。*scale* は 1 以下に選択され解のオーバーフローを防ぐ。

x	REAL (slasy2 の場合) DOUBLE PRECISION (dlasy2 の場合) 配列 DIMENSION は (*ldx*, 2)。*x* には *n1* × *n2* の解が格納される。
xnorm	REAL (slasy2 の場合) DOUBLE PRECISION (dlasy2 の場合) *xnorm* には解の無限ノルムが格納される。
info	INTEGER。*info* には次の値が設定される。 0 の場合、正常に終了したことを示す。 1 の場合、*TL* と *TR* は近すぎる固有値を持つため、*TL* と *TR* は非特異方程式を得るために摂動された。

注：実行速度を考慮してこのルーチンでは入力のエラーをチェックしない。

?lasyf

対角ピボット演算法を使って実数 / 複素対称マトリックスの部分因子分解を計算する。

```
call slasyf ( uplo, n, nb, kb, a, lda, ipiv, w, ldw, info)
call dlasyf ( uplo, n, nb, kb, a, lda, ipiv, w, ldw, info)
call clasyf ( uplo, n, nb, kb, a, lda, ipiv, w, ldw, info)
call zlasyf ( uplo, n, nb, kb, a, lda, ipiv, w, ldw, info)
```

説明

ルーチン ?lasyf は、Bunch-Kaufman 対角ピボット演算法を使って、実数 / 複素対称マトリックス *A* の部分因子分解を計算する。
部分因子分解は以下の形式を持つ。

$$A = \begin{bmatrix} I & U_{12} \\ 0 & U_{22} \end{bmatrix} \begin{bmatrix} A_{11} & 0 \\ 0 & D \end{bmatrix} \begin{bmatrix} I & 0 \\ U_{12}' & U_{22}' \end{bmatrix}$$ uplo = 'U' の場合、または

$$A = \begin{bmatrix} L_{11} & 0 \\ L_{21} & I \end{bmatrix} \begin{bmatrix} D & 0 \\ 0 & A_{22} \end{bmatrix} \begin{bmatrix} L_{11}' & L_{21}' \\ 0 & I \end{bmatrix}$$ uplo = 'L' の場合

ここで D の次数は大きくとも nb である。実際の次数は引数 kb で返され、nb か nb-1、あるいは $n \le nb$ の場合は n である。

このルーチンは ?sytrf から呼び出される補助ルーチンである。部分マトリックス A_{11} (uplo = 'U' の場合) または A_{22} (uplo = 'L' の場合) を更新するためにブロック化コード (レベル 3BLAS の呼び出し) が使用されている。

入力パラメータ

uplo　CHARACTER*1
対称マトリックス A の上三角部分と下三角部分のどちらが格納されるかを指定する。
= 'U': 上三角
= 'L': 下三角

n　INTEGER
マトリックス A の次数。$n \ge 0$

nb　INTEGER
因子分解すべきマトリックス A の最大列。2×2 ピボット・ブロックを許すために、nb は小さくとも 2 でなければならない。

a　REAL (slasyf の場合)
DOUBLE PRECISION (dlasyf の場合)
COMPLEX (clasyf の場合)
COMPLEX*16 (zlasyf の場合)
配列 DIMENSION は (lda, n)。対称マトリックス A を格納する。uplo = 'U' の場合、a の先頭の $n \times n$ 上三角部分にマトリックス A の上三角部分を格納する。a の厳密な下三角部分は参照されない。
uplo = 'L' の場合、a の先頭の $n \times n$ 下三角部分にマトリックス A の下三角部分を格納する。a の厳密な上三角部分は参照されない。

lda INTEGER
配列 *a* のリーディング・ディメンジョン。
lda ≥ max(1, *n*)

w REAL (slasyf の場合)
DOUBLE PRECISION (dlasyf の場合)
COMPLEX (clasyf の場合)
COMPLEX*16 (zlasyf の場合)
ワークスペース配列 DIMENSION は (*ldw*, *nb*)

ldw INTEGER
配列 *w* のリーディング・ディメンジョン。
ldw ≥ max(1, *n*)

出力パラメータ

kb INTEGER
実際に因子分解された *A* の列数。*kb* は *nb*-1、*nb*、または *n* ≤ *nb* は *n* のいずれかである。

a *a* は部分因子分解の詳細で上書きされる。

ipiv INTEGER
配列 DIMENSION は (*n*)。交換の結果と *D* のブロック構造の各成分が格納される。
uplo = 'U' の場合、*ipiv* の最後の *kb* 個の成分のみが設定される。
uplo = 'L' の場合、最初の *kb* 個の成分のみが設定される。
ipiv(*k*) > 0 の場合、*k* 番目の行と列と *ipiv*(*k*) 番目の行と列は交換された。*D*(*k*, *k*) は 1 × 1 の対角ブロックである。
uplo = 'U' かつ *ipiv*(*k*) = *ipiv*(*k*-1) < 0 の場合、*k*-1 番目の行と列と -*ipiv*(*k*) 番目の行と列は交換された。*D*(*k*-1:*k*, *k*-1:*k*) は 2 × 2 の対角ブロックである。
uplo = 'L' かつ *ipiv*(*k*) = *ipiv*(*k*+1) < 0 の場合、*k*+1 番目の行と列と -*ipiv*(*k*) 番目の行と列は交換された。*D*(*k*:*k*+1, *k*:*k*+1) は 2 × 2 の対角ブロックである。

info INTEGER
= 0 の場合、正常に終了したことを示す。
> 0 の場合、*info* = *k* ならば $D(k, k)$ は完全にゼロである。因子分解は完了したが、ブロック対角マトリックス D は完全に特異である。

?lahef

対角ピボット演算法を使って複素エルミート無限マトリックスの部分因子分解を計算する。

```
call clahef ( uplo, n, nb, kb, a, lda, ipiv, w, ldw, info)
call zlahef ( uplo, n, nb, kb, a, lda, ipiv, w, ldw, info)
```

説明

ルーチン ?lahef は、Bunch-Kaufman 対角ピボット演算法を使って、複素エルミート・マトリックス A の部分因子分解を計算する。部分因子分解は次の形式を持つ。

$$A = \begin{bmatrix} I & U_{12} \\ 0 & U_{22} \end{bmatrix} \begin{bmatrix} A_{11} & 0 \\ 0 & D \end{bmatrix} \begin{bmatrix} I & 0 \\ U_{12}' & U_{22}' \end{bmatrix}$$ *uplo* = 'U' の場合、または

$$A = \begin{bmatrix} L_{11} & 0 \\ L_{21} & I \end{bmatrix} \begin{bmatrix} D & 0 \\ 0 & A_{22} \end{bmatrix} \begin{bmatrix} L_{11}' & L_{21}' \\ 0 & I \end{bmatrix}$$ *uplo* = 'L' の場合

D の次数は大きくとも *nb* である。実際の次数は引数 *kb* で返され、*nb* か *nb* -1、あるいは $n \le nb$ の場合は *n* である。なお、U' は U の共役転置を表わす。

このルーチンは ?hetrf から呼び出される補助ルーチンである。部分マトリックス A_{11} (*uplo* = 'U' の場合) または A_{22} (*uplo* = 'L' の場合) を更新するためにブロック化コード (レベル 3 BLAS の呼び出し) が使用されている。

入力パラメータ

uplo CHARACTER*1

エルミート・マトリックス *A* の上三角部分と下三角部分のどちらが格納されるかを指定する。
= 'U' の場合、上三角
= 'L' の場合、下三角

n INTEGER

マトリックス *A* の次数。$n \geq 0$

nb INTEGER

因子分解すべきマトリックス *A* の最大列。2 × 2 ピボット・ブロックを許すために、*nb* は小さくとも 2 でなければならない。

a COMPLEX (clahef の場合)
COMPLEX*16 (zlahef の場合)

配列 DIMENSION は (*lda*, *n*)
エルミート・マトリックス *A* を格納する。
uplo = 'U' の場合、*a* の先頭の $n \times n$ 上三角部分にマトリックス *A* の上三角部分を格納する。*a* の厳密な下三角部分は参照されない。
uplo = 'L' の場合、*a* の先頭の $n \times n$ 下三角部分にマトリックス *A* の下三角部分を格納する。*a* の厳密な上三角部分は参照されない。

lda INTEGER

配列 *a* のリーディング・ディメンジョン。
$lda \geq \max(1, n)$

w COMPLEX (clahef の場合)
COMPLEX*16 (zlahef の場合)

ワークスペース配列 DIMENSION は (*ldw*, *nb*)

ldw INTEGER

配列 *w* のリーディング・ディメンジョン。
$ldw \geq \max(1, n)$

出力パラメータ

kb INTEGER

実際に因子分解された *A* の列数。*kb* は *nb*-1, *nb*, または $n \leq nb$ は *n* のいずれかである。

a	*a* は部分因子分解の詳細で上書きされる。
ipiv	INTEGER 配列 DIMENSION は (*n*)。交換の結果と *D* のブロック構造の各成分が格納される。 *uplo* = 'U' の場合、*ipiv* の最後の *kb* 個の成分のみが設定される。 *uplo* = 'L' の場合、最初の *kb* 個の成分のみが設定される。 *ipiv*(*k*) > 0 の場合、*k* 番目の行と列と *ipiv*(*k*) 番目の行と列は交換された。*D*(*k*, *k*) は 1 × 1 の対角ブロックである。 *uplo* = 'U' かつ *ipiv*(*k*) = *ipiv*(*k*-1) < 0 の場合、*k*-1 番目の行と列と -*ipiv*(*k*) 番目の行と列は交換された。*D*(*k*-1:*k*, *k*-1:*k*) は 2 × 2 の対角ブロックである。 *uplo* = 'L' かつ *ipiv*(*k*) = *ipiv*(*k*+1) < 0 の場合、*k*+1 番目の行と列と -*ipiv*(*k*) 番目の行と列は交換された。*D*(*k*:*k*+1, *k*:*k*+1) は 2 × 2 の対角ブロックである。
info	INTEGER = 0 の場合、正常に終了したことを示す。 > 0 の場合、*info* = *k* ならば *D*(*k*, *k*) は完全にゼロである。因子分解は完了したが、ブロック対角マトリックス *D* は完全に特異である。

?latbs

三角バンド連立方程式を解く。

```
call slatbs ( uplo, trans, diag, normin, n, kd, ab, ldab, x, scale,
              cnorm, info )
call dlatbs ( uplo, trans, diag, normin, n, kd, ab, ldab, x, scale,
              cnorm, info )
call clatbs ( uplo, trans, diag, normin, n, kd, ab, ldab, x, scale,
              cnorm, info )
call zlatbs ( uplo, trans, diag, normin, n, kd, ab, ldab, x, scale,
              cnorm, info )
```

説明

このルーチンは、次の三角法をオーバーフローを防ぐためにスケーリングして解く。

$Ax = s\,b$、または $A^{T}x = s\,b$、または $A^{H}x = s\,b$ (複素型の場合)

ここで A は上または下三角バンド・マトリックスである。A^{T} は A の転置を表わし、A^{H} は A の共役転置を表わし、x と b は *n* 成分のベクトル、s はスケール係数で、x の成分がオーバーフローしきい値未満になるように通常は 1 以下に選択される。スケーリングされていない問題がオーバーフローを起こさない場合は、レベル 2 の BLAS ルーチン `?tbsv` が呼び出される。マトリックス A が特異 (任意の j に対して $A(j, j) = 0$) の場合、s は 0 に設定され、$Ax = 0$ に対するゼロでない解が返される。

入力パラメータ

uplo CHARACTER*1

マトリックス A が上三角か下三角かを指定する。
= 'U' の場合、上三角
= 'L' の場合、下三角

trans CHARACTER*1

A に適用する演算を指定する。
= 'N' の場合、$Ax = s\,b$ を解く (転置なし)
= 'T' の場合、$A^{T}x = s\,b$ を解く (転置)
= 'C' の場合、$A^{H}x = s\,b$ を解く (共役転置)

diag CHARACTER*1

マトリックス A が単位三角かどうかを指定する。
= 'N' の場合、単位三角ではない
= 'U' の場合、単位三角

normin CHARACTER*1

cnorm を設定したかどうかを指定する。
= 'Y' の場合、*cnorm* には列ノルムを格納した。
= 'N' の場合、*cnorm* は設定していない。ノルムが計算され出力時に *cnorm* に格納される。

n INTEGER

マトリックス A の次数。$n \geq 0$

kd INTEGER
三角マトリックス A の劣対角または優対角の個数。
$kd \geq 0$

ab REAL (slatbs の場合)
DOUBLE PRECISION (dlatbs の場合)
COMPLEX (clatbs の場合)
COMPLEX*16 (zlatbs の場合)
配列 DIMENSION は (*ldab*, *n*)。上三角または下三角バンド・マトリックス A で、配列の最初の *kd*+1 行に格納する。A の *j* 番目の列を配列 *ab* の *j* 番目の列に次のように格納する。
uplo = 'U' の場合、$\max(1, j-kd) \leq i \leq j$ に対して
$ab(kd+1+i-j, j) = A(i, j)$
uplo = 'L' の場合、$j \leq i \leq \min(n, j+kd)$ に対して
$ab(1+i-j, j) = A(i, j)$

ldab INTEGER
配列 *ab* のリーディング・ディメンジョン。
$ldab \geq kd+1$

x REAL (slatbs の場合)
DOUBLE PRECISION (dlatbs の場合)
COMPLEX (clatbs の場合)
COMPLEX*16 (zlatbs の場合)
配列 DIMENSION は (*n*)
三角法の右辺 b を格納する。

cnorm REAL (slatbs/clatbs の場合)
DOUBLE PRECISION (dlatbs/zlatbs の場合)
配列 DIMENSION は (*n*)
normin = 'Y' の場合、*cnorm* は入力引数となり
cnorm(*j*) には A の *j* 番目列の対角外部分を格納する。
trans = 'N' の場合、*cnorm*(*j*) は無限ノルムに等しいか大きくなければならない。*trans* = 'T' または 'C' の場合、*cnorm*(*j*) は 1- ノルムに等しいか大きくなければならない。

出力パラメータ

scale	REAL (slatbs/clatbs の場合) DOUBLE PRECISION (dlatbs/zlatbs の場合) 上述のとおり三角法に対するスケール係数 *s*。 *scale* = 0 はマトリックス *A* が特異であるか不適切にスケーリングされたことを示し、ベクトル x は A*x* = 0 に対する完全または近似解となる。
cnorm	*normin* = 'N' の場合、*cnorm* は出力引数となり *cnorm*(*j*) は *A* の *j* 番目列の対角外部分の 1- ノルムを返す。
info	INTEGER = 0 の場合、正常に終了したことを示す。 < 0 の場合、*info* = -*k* は *k* 番目の引数の値が不正だったことを示す。

?latdf

?getc2 による *n* × *n* マトリックスの *LU* 因子分解を使い、Dif 推定値の逆数に対する影響を計算する。

```
call slatdf ( ijob, n, z, ldz, rhs, rdsum, rdscal, ipiv, jpiv )
call dlatdf ( ijob, n, z, ldz, rhs, rdsum, rdscal, ipiv, jpiv )
call clatdf ( ijob, n, z, ldz, rhs, rdsum, rdscal, ipiv, jpiv )
call zlatdf ( ijob, n, z, ldz, rhs, rdsum, rdscal, ipiv, jpiv )
```

説明

ルーチン ?latdf は、?getc2 で計算された *n* × *n* マトリックスの *LU* 因子分解を使い、*x* に対して *Zx* = *b* を解き、さらに *x* のノルムが可能な限り大きくなるような右辺 *b* を選択して、Dif 推定値の逆数に対する影響を計算する。入力において、*rhs* = *b* にはこれまでに部分式から解いた影響を格納し、出力において *rhs* = *x* となる。

?getc2 で返される Z の因子分解は形式 $Z = P\,L\,U\,Q$ を持っており、P と Q は置換マトリックスである。L は単位対角成分を持つ下三角、U は上三角である。

入力パラメータ

ijob INTEGER

ijob = 2 の場合、最初に ?gecon を使って Z のおおよそのヌル・ベクトル e を計算し、続いて e は正規化され、最後に 2- ノルム (x) に大きな値を与える符号を使って $Zx = \pm e - f$ を解く。このオプションはデフォルトよりも 5 倍程度負荷が高い。

ijob ≠ 2 (デフォルト) の場合、右辺 b の全成分が +1 または -1 のいずれかとして選択される局所的先読み法。

n INTEGER

マトリックス Z の列数。

z REAL (slatdf/clatdf の場合)

DOUBLE PRECISION (dlatdf/zlatdf の場合)

配列 DIMENSION は (*ldz*, *n*)

n × *n* マトリックス Z を ?getc2 で計算した因子分解の LU 部分を格納する。$Z = P\,L\,U\,Q$

ldz INTEGER

配列 *z* のリーディング・ディメンジョン。

lda ≥ max(1, *n*)

rhs REAL (slatdf/clatdf の場合)

DOUBLE PRECISION (dlatdf/zlatdf の場合)

配列 DIMENSION は (*n*)

rhs には他の部分式で得られた影響を格納する。

rdsum REAL (slatdf/clatdf の場合)

DOUBLE PRECISION (dlatdf/zlatdf の場合)

?tgsyl で得られた Dif 推定に対する、計算された影響の二乗和を格納する。スケール係数 *rdscal* は因子分解されている。

trans = 'T' の場合、rdsum は使われない。
rdsum は ?tgsy2 が ?tgsyl から呼び出されたときのみ意味を持つことに注意。

rdscal
REAL (slatdf/clatdf の場合)
DOUBLE PRECISION (dlatdf/zlatdf の場合)
rdsum のオーバーフローを防ぐために使用されるスケール係数を格納する。trans = 'T' の場合、rdscal は使われない。
rdscal は ?tgsy2 が ?tgsyl から呼び出されたときのみ意味を持つことに注意する。

ipiv
INTEGER
配列 DIMENSION は (n)
ピボット・インデックス。$1 \le i \le n$ に対して、マトリックスの行 i は行 ipiv(i) と交換されている。

jpiv
INTEGER
配列 DIMENSION は (n)
ピボット・インデックス。$1 \le j \le n$ に対して、マトリックスの列 j は列 jpiv(j) と交換されている。

出力パラメータ

rhs
rhs には、ijob の値に対応する成分で構成される、部分式の解が上書きされる

rdsum
対応する二乗和は、現在の部分式から得られた影響によって更新される。
trans = 'T' の場合、rdsum は使われない。

rdscal
rdscal は、rdsum に格納されている現在の影響に関して更新される。

?latps

パックド形式で格納されている
マトリックスを持った三角連立方程式を
解く。

```
call slatps (uplo, trans, diag, normin, n, ap, x, scale, cnorm, info)
call dlatps (uplo, trans, diag, normin, n, ap, x, scale, cnorm, info)
call clatps (uplo, trans, diag, normin, n, ap, x, scale, cnorm, info)
call zlatps (uplo, trans, diag, normin, n, ap, x, scale, cnorm, info)
```

説明

ルーチン ?latps は、次の三角法のうち 1 つをオーバーフローを防ぐためにスケーリングして解く。

$Ax = s\,b$、または $A^{T}x = s\,b$、または $A^{H}x = s\,b$ (複素型の場合)

A はパックド形式で格納されている上または下三角マトリックスである。A^{T} は A の転置を表わし、A^{H} は A の共役転置を表わし、x と b は n 成分のベクトル、s はスケール係数で、x の成分がオーバーフローしきい値未満になるように通常は 1 以下に選択される。スケーリングされていない問題がオーバーフローを起こさない場合は、レベル 2 の BLAS ルーチン ?tpsv が呼び出される。マトリックス A が特異 (任意の j に対して $A(j, j) = 0$) の場合、s は 0 に設定され $Ax = 0$ に対するゼロでない解が返される

入力パラメータ

uplo CHARACTER*1
マトリックス A が上三角か下三角かを指定する。
= 'U' の場合、上三角
= 'L' の場合、下三角

trans CHARACTER*1
A に適用する演算を指定する。
= 'N' の場合、$Ax = s\,b$ を解く (転置なし)
= 'T' の場合、$A^{T}x = s\,b$ を解く (転置)
= 'C' の場合、$A^{H}x = s\,b$ を解く (共役転置)

diag CHARACTER*1
マトリックス *A* が単位三角かどうかを指定する。
= 'N' の場合、単位三角ではない
= 'U' の場合、単位三角

normin CHARACTER*1
cnorm を設定したかどうかを指定する。
= 'Y' の場合、*cnorm* には列ノルムを格納した。
= 'N' の場合、*cnorm* は設定していない。ノルムが計算され、出力時に *cnorm* に格納される。

n INTEGER
マトリックス *A* の次数。$n \geq 0$

ap REAL (slatps の場合)
DOUBLE PRECISION (dlatps の場合)
COMPLEX (clatps の場合)
COMPLEX*16 (zlatps の場合)
配列 DIMENSION は $(n(n+1)/2)$。上三角または下三角バンド・マトリックス *A* で、線形配列にカラム方向にパックされている。*A* の *j* 番目の列は配列 *ap* に次のように格納する。
uplo = 'U' の場合、$1 \leq i \leq j$ に対して
$ap(i + (j-1)j/2) = A(i, j)$
uplo = 'L' の場合、$j \leq i \leq n$ に対して
$ap(i + (j-1)(2n-j)/2) = A(i, j)$

x REAL (slatps の場合)
DOUBLE PRECISION (dlatps の場合)
COMPLEX (clatps の場合)
COMPLEX*16 (zlatps の場合)
配列 DIMENSION は (n)
三角法の右辺 *b* を格納する。

cnorm REAL (slatps/clatps の場合)
DOUBLE PRECISION (dlatps/zlatps の場合)
配列 DIMENSION は (n)
normin = 'Y' の場合、*cnorm* は入力引数となり、*cnorm*(*j*) には *A* の *j* 番目列の対角外部分を格納する。*trans* = 'N' の場合、cnorm(*j*) は無限ノルムに等しいか大きくなければならない。*trans* = 'T' または 'C' の場合、*cnorm*(*j*) は 1- ノルムに等しいか大きくなければならない。

出力パラメータ

x	終了時に、*x* は解のベクトル x によって上書きされる。
scale	REAL (slatps/clatps の場合) DOUBLE PRECISION (dlatps/zlatps の場合) 上述のとおり三角法に対するスケール係数 *s*。 *scale* = 0 の場合、マトリックス *A* は特異であるか不適切にスケーリングされたことを示し、ベクトル *x* は $Ax = 0$ に対する完全または近似解となる。
cnorm	*normin* = 'N' の場合、*cnorm* は出力引数となり *cnorm*(*j*) は *A* の *j* 番目列の対角外部分の 1- ノルムを返す。
info	INTEGER = 0 の場合、正常に終了したことを示す。 < 0 の場合、*info* = -*k* は *k* 番目の引数の値が不正だったことを示す。

?latrd

直交 / ユニタリ相似変換を用いて、
対称 / エルミート・マトリックス *A* の
最初の *nb* 列と *nb* 行を実数三重
対角形式に縮退する。

```
call slatrd ( uplo, n, nb, a, lda, e, tau, w, ldw )
call dlatrd ( uplo, n, nb, a, lda, e, tau, w, ldw )
call clatrd ( uplo, n, nb, a, lda, e, tau, w, ldw )
call zlatrd ( uplo, n, nb, a, lda, e, tau, w, ldw )
```

説明

ルーチン ?latrd は、実数対称または複素エルミート・マトリックス *A* の *nb* 行と *nb* 列を、直交 / ユニタリ相似変換 $Q' A Q$ によって対称 / エルミート実数三重対角形式に縮退し、*A* の縮退されていない部分に変換を適用するために必要となる *V* と *W* を返す。

uplo = 'U' の場合、?latrd は、上三角が与えられるマトリックスに対して最後の *nb* 行と *nb* 列の縮退を実行する。

uplo = 'L' の場合、?latrd は、下三角が与えられるマトリックスに対して最初の *nb* 行と *nb* 列の縮退を実行する。

入力パラメータ

uplo
CHARACTER。
対称 / エルミート・マトリックス *A* の上三角または下三角のどちらを格納するか指定する。
= 'U': 上三角
= 'L': 下三角

n
INTEGER
マトリックス *A* の次数。

nb
INTEGER
縮退する行数と列数。

a
REAL (slatrd の場合)
DOUBLE PRECISION (dlatrd の場合)
COMPLEX (clatrd の場合)
COMPLEX*16 (zlatrd の場合)
配列 DIMENSION は (*lda*, *n*)
対称 / エルミート・マトリックス *A* を格納する。
uplo = 'U' の場合、*a* の先頭の *n* × *n* 上三角部分にマトリックス *A* の上三角部分を格納する。厳密な下三角部分は参照されない。*uplo* = 'L' の場合、*a* の先頭の *n* × *n* 下三角部分にマトリックス *A* の下三角部分を格納する。厳密な上三角部分は参照されない。

lda
INTEGER
配列 *a* のリーディング・ディメンジョン。
lda ≥ (1, *n*)

ldw INTEGER
出力配列 *w* のリーディング・ディメンジョン。
ldw ≥ max(1, *n*)

出力パラメータ

a *uplo* = 'U' の場合、最後の *nb* 列が三重対角形式に縮退され、*a* の対角成分は三重対角形式の対角成分で上書きされる。対角よりも上の成分は配列 *tau* とともに基本リフレクタの積として直交 / ユニタリのマトリックス *Q* を表現する。
uplo = 'L' の場合、最初の *nb* 列が三重対角形式に縮退され、*a* の対角成分は三重対角形式の対角成分で上書きされる。対角よりも下の成分は配列 *tau* とともに基本リフレクタの積として直交 / ユニタリのマトリックス *Q* を表現する。

e REAL (slatrd/clatrd の場合)
DOUBLE PRECISION (dlatrd/zlatrd の場合)
uplo = 'U' の場合、*e*(*n*-*nb*:*n*-1) には縮退されたマトリックスの最後の *nb* 列の優対角成分が格納される。
uplo = 'L' の場合、*e*(1:*nb*) には縮退されたマトリックスの最初の *nb* 列の劣対角成分が格納される。

tau REAL (slatrd の場合)
DOUBLE PRECISION (dlatrd の場合)
COMPLEX (clatrd の場合)
COMPLEX*16 (zlatrd の場合)
配列 DIMENSION は (*lda*, *n*)
基本リフレクタのスケール係数で、*uplo* = 'U' の場合は *tau*(*n*-*nb*:*n*-1) に、*uplo* = 'L' の場合は *tau*(1:*nb*) に格納される。

w REAL (slatrd の場合)
DOUBLE PRECISION (dlatrd の場合)
COMPLEX (clatrd の場合)
COMPLEX*16 (zlatrd の場合)

配列 DIMENSION は (*lda*, *n*)
A の縮退されていない部分の更新に必要な *n* × *nb* マトリックス *W*

アプリケーション・ノート

uplo = 'U' の場合、マトリックス *Q* は基本リフレクタの積として表現される。

$$Q = H(n)\,H(n-1)\,\ldots\,H(n-nb+1)$$

それぞれの *H*(*i*) は次のような形式を持つ。

$$H(i) = I - tau * v * v'$$

tau は実数 / 複素スカラー、*v* は実数 / 複素ベクトルで *v*(*i*:*n*) = 0 と *v*(*i*-1) = 1。出力において、v(1:*i*-1) は *a*(1:*i*-1, *i*) に格納され、*tau* は *tau*(*i*-1) に格納される。

uplo = 'L' の場合、マトリックス *Q* は基本リフレクタの積として表現される。

$$Q = H(1)\,H(2)\,\ldots\,H(nb)$$

それぞれの *H*(*i*) は次のような形式を持つ。

$$H(i) = I - tau * v * v'$$

tau は実数 / 複素スカラー、*v* は実数 / 複素ベクトルで *v*(1:*i*) = 0 と *v*(*i*+1) = 1。出力において、*v*(*i*+1:n) は *a*(*i*+1:*n*, *i*) に格納され、*tau* は *tau*(*i*) に格納される。

ベクトル *v* の成分は、マトリックスの縮退されていない部分に変換を適用するために、マトリックス *W* とともに必要となる *n* × *nb* のマトリックス *V* を、形式の対称 / エルミート階数 -2k 更新を使用して形成する。
A := *A* - *VW*' - *WV*'.

a の出力の例を、*n* = 5、*nb* = 2 で以下に示す。

uplo = 'U' の場合　　*uplo* = 'L' の場合

$$\begin{bmatrix} a & a & a & v_4 & v_5 \\ & a & a & v_4 & v_5 \\ & & a & 1 & v_5 \\ & & & d & 1 \\ & & & & d \end{bmatrix} \qquad \begin{bmatrix} d & & & & \\ 1 & d & & & \\ v_1 & 1 & a & & \\ v_1 & v_2 & a & a & \\ v_1 & v_2 & a & a & a \end{bmatrix}$$

d は縮退されたマトリックスの対角成分、a は変更されていない元のマトリックスの成分、v_i は $H(i)$ を定義するベクトルの成分を表わす。

?latrs

オーバフローを防ぐために設定されたスケール係数を持つ三角連立方程式を解く。

```
call slatrs ( uplo, trans, diag, normin, n, a, lda, x, scale, cnorm,
              info )
call dlatrs ( uplo, trans, diag, normin, n, a, lda, x, scale, cnorm,
              info )
call clatrs ( uplo, trans, diag, normin, n, a, lda, x, scale, cnorm,
              info )
call zlatrs ( uplo, trans, diag, normin, n, a, lda, x, scale, cnorm,
              info )
```

説明

このルーチンは、次の三角法をオーバーフローを防ぐためにスケーリングして解く。

$Ax = s\,b$、または $A^T x = s\,b$、または $A^H x = s\,b$ (複素型の場合)

A は上または下三角マトリックスである。A^T は A の転置を表わし、A^H は A の共役転置を表わし、x と b は `n` 成分のベクトル、s はスケール係数で、x の成分がオーバーフローしきい値未満になるように通常は 1 以下に選択される。スケーリングされていない問題がオーバーフローを起こさない場合は、レベル 2 の BLAS ルーチン `? trsv` が呼び出される。マトリックス A が特異 (任意の `j` に対して $A(j, j) = 0$) の場合、s は 0 に設定され $Ax = 0$ に対するゼロでない解が返される。

入力パラメータ

uplo CHARACTER*1

マトリックス A が上三角か下三角かを指定する。
= 'U' の場合、上三角
= 'L' の場合、下三角

trans CHARACTER*1

A に適用する演算を指定する。
= 'N' の場合、$Ax = s\,b$ を解く (転置なし)
= 'T' の場合、$A^{T}x = s\,b$ を解く (転置)
= 'C' の場合、$A^{H}x = s\,b$ を解く (共役転置)

diag CHARACTER*1

マトリックス A が単位三角かどうかを指定する。
= 'N' の場合、単位三角ではない
= 'U' の場合、単位三角

normin CHARACTER*1

cnorm を設定したかどうかを指定する。
= 'Y' の場合、*cnorm* には列ノルムを格納した。
= 'N' の場合、*cnorm* は設定していない。ノルムが計算され出力時に *cnorm* に格納される。

n INTEGER

マトリックス A の次数。$n \geq 0$

a REAL (slatrs の場合)
DOUBLE PRECISION (dlatrs の場合)
COMPLEX (clatrs の場合)
COMPLEX*16 (zlatrs の場合)

配列 DIMENSION は (*lda*, *n*)。三角マトリックス A を格納する。*uplo* = 'U' の場合、配列 *a* の先頭の $n \times n$ 上三角部分に上三角マトリックスを格納する。*a* の厳密な下三角部分は参照されない。*uplo* = 'L' の場合、配列 *a* の先頭の $n \times n$ 下三角部分に下三角マトリックスを格納する。*a* の厳密な上三角部分は参照されない。*diag* = 'U' の場合、*a* の対角成分も参照されず 1 とみなされる。

lda INTEGER
配列 *a* のリーディング・ディメンジョン。
lda ≥ max (1, *n*)

x REAL (slatrs の場合)
DOUBLE PRECISION (dlatrs の場合)
COMPLEX (clatrs の場合)
COMPLEX*16 (zlatrs の場合)
配列 DIMENSION は (*n*)。三角法の右辺 *b* を格納する。

cnorm REAL (slatrs/clatrs の場合)
DOUBLE PRECISION (dlatrs/zlatrs の場合)
配列 DIMENSION は (*n*)。*normin* = 'Y' の場合、*cnorm* は入力引数となり *cnorm*(*j*) には *A* の *j* 番目列の対角外部分を格納する。*trans* = 'N' の場合、*cnorm*(*j*) は無限ノルムに等しいか大きくなければならない。*trans* = 'T' または 'C' の場合、*cnorm*(*j*) は 1- ノルムに等しいか大きくなければならない。

出力パラメータ

x 終了時に、*x* は解のベクトル x によって上書きされる。

scale REAL (slatrs/clatrs の場合)
DOUBLE PRECISION (dlatrs/zlatrs の場合)
配列 DIMENSION は (*lda*, *n*)。上述のとおり三角法に対するスケール係数 *s*。
scale = 0 はマトリックス *A* が特異であるか不適切にスケーリングされたことを示し、ベクトル *x* は $Ax = 0$ に対する完全または近似解となる。

cnorm *normin* = 'N' の場合、*cnorm* は出力引数となり *cnorm*(*j*) は *A* の *j* 番目列の対角外部分の 1- ノルムを返す。

info INTEGER
= 0 の場合、正常に終了したことを示す。
< 0 の場合、*info* = -*k* は *k* 番目の引数の値が不正だったことを示す。

アプリケーション・ノート

x に対するおおまかな制限が計算される。これがオーバーフロー未満の場合は ?trsv が呼び出され、そうでなければ演算ごとにオーバーフローまたはゼロ除算をチェックする特定のコードが使われる。

$Ax = b$ を解くうえで列方向の格納形式が使用される。A が下三角の場合、基本アルゴリズムは次のようになる。

$x[1:n] := b[1:n]$
for $j = 1, ..., n$
$x(j) := x(j) / A(j, j)$
$x[j+1:n] := x[j+1:n] - x(j)*A[j+1:n, j]$
end

ループを j 回反復した後、x の成分に対する制限を定義する。
$M(j) = x[1:j]$ に対する制限
$G(j) = x[j+1:n]$ に対する制限
初期時は、$M(0) = 0$ および $G(0) = \max\{x(i),\ i = 1, ..., n\}$ とする。

j+1 回目の反復で、以下を得る。

$$M(j+1) \le G(j) / |A(j+1, j+1)|$$
$$G(j+1) \le G(j) + M(j+1)*|A[j+2:n, j+1]|$$
$$\le G(j)\,(1 + cnorm(j+1) / |A(j+1, j+1)|\ ,$$

$cnorm(j+1)$ は、対角を含めない状態で A の列 j+1 の無限ノルムに等しいか大きい。ゆえに、

$$G(j) \le G(0) \prod_{1 \le i \le j} (1 + cnorm(i)/|A(i, i)|)$$

および、

$$|x(j)| \le (G(0)/|A(j, j)|) \prod_{1 \le i \le j} (1 + cnorm(i)/|A(i, i)|)$$

$|x(j)| \le M(j)$ であるから、$j = 1, .., n$ における最大 $M(j)$ の逆数が $\max(underflow, 1/overflow)$ よりも大きい場合はレベル 2 の BLAS ルーチン ?trsv を使用する。
$x(j)$ に対する制限もまた、オーバーフローの可能性のない、列方向の格納形式でのステップの実行判断に使用される。計算された制限が大きな定数よりも大きい場合、オーバーフローを防ぐために x はスケーリングされる。しかし、制限がオーバーフローする場合、x は 0 に、$x(j)$ は 1 に、scale は 0 にそれぞれ設定され、$Ax = 0$ に対するゼロでない解が返される。

同様に、列方向の格納形式が $A^{T}x = b$ または $A^{H}x = b$ を解くために使用される。A が上三角の場合、基本アルゴリズムは次のようになる。

for j = 1, ..., n
$x(j)$:= ($b(j)$ - $A[1:j-1, j]'$ $x[1:j-1]$) / $A(j, j)$
end

2 つの制限が同時に計算される。
$G(i) = (b(i)A[1:i-1, i]'*x[1:i-1])$ に対する制限、$1 \le i \le j$
$M(j) = x(i)$, $1 \le i \le j$ に対する制限
初期値は $G(0) = 0$、$M(0) = \max\{b(i), i=1, .., n\}$ で、ここで $j \ge 1$ に対し $G(j) \ge G(j-1)$ と $M(j) \ge M(j-1)$ の制約を追加する。

よって、$x(j)$ に対する制限は、

$$M(j) \le M(j-1) *(1 + cnorm(j)) / |A(j, j)|$$

$$\le M(0) \prod_{1 \le i \le j} (1 + cnorm(i) / |A(i, i)|)$$

これによって、$1/M(n)$ と $1/G(n)$ の両方が $\max(underflow, 1/overflow)$ を超えている場合でも安全に `?trsv` を呼び出すことができる。

?latrz

上台形マトリックスを直交 / ユニタリ
変換によって因子分解する。

```
call slatrz ( m, n, l, a, lda, tau, work )
call dlatrz ( m, n, l, a, lda, tau, work )
call clatrz ( m, n, l, a, lda, tau, work )
call zlatrz ( m, n, l, a, lda, tau, work )
```

説明

ルーチン ?latrz は、次の $m \times (m+l)$ の実数 / 複素上台形マトリックスを、

$[\ A1\ \ A2\ ] = [\ A(1{:}m,\ 1{:}m)\ \ \ A(1{:}m,\ n-l+1{:}n)\ ]$

$(R\ \ \ 0)*Z$ として、直交 / ユニタリ変換を使用して因子分解する。Z は $(m+l) \times (m+l)$ 直交 / ユニタリ・マトリックス、R と $A1$ は $m \times m$ の上三角マトリックスである。

入力パラメータ

m	INTEGER マトリックス A の行数。$m \geq 0$
n	INTEGER マトリックス A の列数。$n \geq 0$
l	INTEGER Householder ベクトルとして意味を持った部分が格納されているマトリックス A の列数。$n-m \geq l \geq 0$
a	REAL (slatrz の場合) DOUBLE PRECISION (dlatrz の場合) COMPLEX (clatrz の場合) COMPLEX*16 (zlatrz の場合) 配列 DIMENSION は (*lda*, *n*) 配列 *a* 先頭の $m \times n$ 上台形部分には因子分解するマトリックスを格納しておかなければならない。
lda	INTEGER 配列 *a* のリーディング・ディメンジョン。 $lda \geq \max(1, m)$
work	REAL (slatrz の場合) DOUBLE PRECISION (dlatrz の場合) COMPLEX (clatrz の場合) COMPLEX*16 (zlatrz の場合) ワークスペース配列 DIMENSION は (*m*)

出力パラメータ

a	a の先頭の m × m 上三角部分には上三角マトリックス R が格納される。a の最初の m 行の n-l+1 から n までの成分は、配列 tau とともに、基本リフレクタの積として直交 / ユニタリ・マトリックス Z を表現する。
tau	REAL (slatrz の場合) DOUBLE PRECISION (dlatrz の場合) COMPLEX (clatrz の場合) COMPLEX*16 (zlatrz の場合) 配列 DIMENSION は (m)。基本リフレクタのスケール係数。

アプリケーション・ノート

因子分解は Householder 法を用いて得る。A の (m - k + 1) 番目の行にゼロを導入するために使用される k 番目の変換マトリックス Z(k) は次の形式で与えられる。

$$Z(k) = \begin{bmatrix} I & 0 \\ 0 & T(k) \end{bmatrix}$$

ここで

$$T(k) = I - tau * u(k) * u(k)', \qquad u(k) = \begin{bmatrix} 1 \\ 0 \\ z(k) \end{bmatrix}$$

tau はスカラー、z(k) は 1 成分のベクトルである。tau と z(k) は、A2 の k 番目の行の成分をゼロにするために選択される。スカラー tau は tau の k 番目の成分で返され、ベクトル u(k) は、z(k) の成分が a(k, l+1), ..., a(k, n) に格納されるような A2 の k 番目の行で返される。R の成分は A1 の上三角部分で返される。
Z は次式で与えられる。

$$Z = Z(1)\,Z(2) \ldots Z(m)$$

?lauu2

積 UU^H または LH^L を計算する。ここで
U と L は上三角または下三角マトリックス
(非ブロック化アルゴリズム)。

```
call slauu2 ( uplo, n, a, lda, info )
call dlauu2 ( uplo, n, a, lda, info )
call clauu2 ( uplo, n, a, lda, info )
call zlauu2 ( uplo, n, a, lda, info )
```

説明

ルーチン ?lauu2 は積 UU' または LL' を計算する。ここで、三角係数 U または L は、それぞれ配列 *a* の上三角または下三角部分に格納される。
uplo = 'U' または 'U' の場合、*a* の係数 U は結果の上三角で上書きされる。
uplo = 'L' または 'l' の場合、*a* の係数 L は結果の下三角で上書きされる。

このルーチンはアルゴリズムの非ブロック化形式で、レベル 2 BLAS を呼び出す。

入力パラメータ

uplo CHARACTER*1
配列 *a* に格納されている三角係数が上三角か下三角かを指定する。
= 'U' の場合、上三角
= 'L' の場合、下三角

n INTEGER
三角係数 U または L の次数。$n \geq 0$

a REAL (slauu2 の場合)
DOUBLE PRECISION (dlauu2 の場合)
COMPLEX (clauu2 の場合)

COMPLEX*16 (zlauu2 の場合)
配列 DIMENSION は (*lda*, *n*)。三角係数 *U* または *L* を格納する。

lda INTEGER
配列 *a* のリーディング・ディメンジョン。
lda ≥ max(1, *n*)

出力パラメータ

a *uplo* = 'U' の場合、*a* の上三角は積 *UU'* の上三角で上書きされる。
uplo = 'L' の場合、*a* の下三角は積 *LL'* の下三角で上書きされる。

info INTEGER
= 0 の場合、正常に終了したことを示す。
< 0 の場合、*info* = -*k* は *k* 番目の引数の値が不正だったことを示す。

?lauum

積 UU^H または LH^L を計算する。ここで、
U と *L* は上三角または下三角
マトリックス。

```
call slauum ( uplo, n, a, lda, info )
call dlauum ( uplo, n, a, lda, info )
call clauum ( uplo, n, a, lda, info )
call zlauum ( uplo, n, a, lda, info )
```

説明

ルーチン ?lauum は積 *UU'* または *L'L* を計算する。三角係数 *U* または *L* は、配列 *a* の上三角または下三角部分に格納される。
uplo = 'U' または 'u' の場合、*a* の係数 *U* は結果の上三角で上書きされる。
uplo = 'L' または 'l' の場合、*a* の係数 *L* は結果の下三角で上書きされる。

これはアルゴリズムの非ブロック化形式で、レベル 3 BLAS を呼び出す。

入力パラメータ

uplo CHARACTER*1
配列 *a* に格納されている三角係数が上三角か下三角かを指定する。
= 'U' の場合、上三角
= 'L' の場合、下三角

n INTEGER
三角係数 *U* または *L* の次数。$n \geq 0$

a REAL (slauum の場合)
DOUBLE PRECISION (dlauum の場合)
COMPLEX (clauum の場合)
COMPLEX*16 (zlauum の場合)
配列 DIMENSION は (*lda*, *n*)。三角係数 *U* または *L* を格納する。

lda INTEGER
配列 *a* のリーディング・ディメンジョン。
$lda \geq \max(1, n)$

出力パラメータ

a *uplo* = 'U' の場合、*a* の上三角は積 *UU'* の上三角で上書きされる。
uplo = 'L' の場合、*a* の下三角は積 *LL'* の下三角で上書きされる。

info INTEGER
= 0 の場合、正常に終了したことを示す。
< 0 の場合、*info* = -*k* は *k* 番目の引数の値が不正だったことを示す。

?org2l/?ung2l

?geqlf で求めた *QL* 因子分解から、全部または一部の直交 / ユニタリ・マトリックス *Q* を生成する (非ブロック化アルゴリズム)。

```
call sorg2l ( m, n, k, a, lda, tau, work, info )
call dorg2l ( m, n, k, a, lda, tau, work, info )
call cung2l ( m, n, k, a, lda, tau, work, info )
call zung2l ( m, n, k, a, lda, tau, work, info )
```

説明

ルーチン ?org2l/?ung2l は、直交列を持つ $m \times n$ の実数 / 複素マトリックス *Q* を生成する。これは、?geqlf から返されたとおり、次数 *m* の *k* 個の基本リフレクタの積の最後の *n* 列として定義される。

$$Q = H(k) \ldots H(2)\, H(1)$$

入力パラメータ

m INTEGER
マトリックス *Q* の行数。$m \geq 0$

n INTEGER
マトリックス *Q* の列数。$m \geq n \geq 0$

k INTEGER
基本リフレクタ (その積でマトリックス *Q* を定義) の個数。$n \geq k \geq 0$

a REAL (sorg2l の場合)
DOUBLE PRECISION (dorg2l の場合)
COMPLEX (cung2l の場合)
COMPLEX*16 (zung2l の場合)
配列 DIMENSION は (*lda*, *n*)
a の (*n*- *k*+*i*) 番目の列には、?geqlf によって配列引数 *a* の最後の *k* 列に返されたとおりに、基本リフレクタ *H*(*i*) を定義するベクトルを格納しておかなければならない。ここで、$i = 1, 2, ..., k$

lda INTEGER
配列 *a* の第 1 次元。*lda* ≥ max(1, *m*)

tau REAL (sorg2l の場合)
DOUBLE PRECISION (dorg2l の場合)
COMPLEX (cung2l の場合)
COMPLEX*16 (zung2l の場合)
配列 DIMENSION は (*k*)
tau(*i*) には、?geqlf から返されたとおりに、基本リフレクタ *H*(*i*) のスカラー係数を格納しておかなければならない。

work REAL (sorg2l の場合)
DOUBLE PRECISION (dorg2l の場合)
COMPLEX (cung2l の場合)
COMPLEX*16 (zung2l の場合)
ワークスペース配列 DIMENSION は (*n*)

出力パラメータ

a *m* × *n* のマトリックス *Q* で上書きされる。

info INTEGER
= 0 の場合、正常に終了したことを示す。
< 0 の場合、*info* = -*i* は *i* 番目の引数の値が不正だったことを示す。

?org2r/?ung2r

?geqrf で求めた *QR* 因子分解から、全部または一部の直交 / ユニタリ・マトリックス *Q* を生成する (非ブロック化アルゴリズム)。

```
call sorg2r ( m, n, k, a, lda, tau, work, info )
call dorg2r ( m, n, k, a, lda, tau, work, info )
call cung2r ( m, n, k, a, lda, tau, work, info )
call zung2r ( m, n, k, a, lda, tau, work, info )
```

説明

ルーチン ?org2r/?ung2r は、直交列を持つ $m \times n$ の実数 / 複素マトリックス Q を生成する。これは、?geqrf から返されたとおり、次数 m の k 個の基本リフレクタの積の最初の n 列として定義される。

$$Q = H(1)\,H(2)\,...\,H(k)$$

入力パラメータ

m INTEGER
マトリックス Q の行数。$m \geq 0$

n INTEGER
マトリックス Q の列数。$m \geq n \geq 0$

k INTEGER
基本リフレクタ (その積でマトリックス Q を定義) の個数。$n \geq k \geq 0$

a REAL (sorg2r の場合)
DOUBLE PRECISION (dorg2r の場合)
COMPLEX (cung2r の場合)
COMPLEX*16 (zung2r の場合)
配列 DIMENSION は (*lda*, *n*)
a の i 番目の列には、?geqrf によって配列引数 *a* の最初の *k* 列に返されたとおりに、基本リフレクタ $H(i)$ を定義するベクトルを格納しておかなければならない。ここで、$i = 1, 2, ..., k$

lda INTEGER
配列 *a* の第 1 次元。$lda \geq \max(1, m)$

tau REAL (sorg2r の場合)
DOUBLE PRECISION (dorg2r の場合)
COMPLEX (cung2r の場合)
COMPLEX*16 (zung2r の場合)

配列 DIMENSION は (*k*)
tau(*i*) には、?geqrf から返されたとおりに、基本リフレクタ *H*(*i*) のスカラー係数を格納しておかなければならない。

work REAL (sorg2r の場合)
DOUBLE PRECISION (dorg2r の場合)
COMPLEX (cung2r の場合)
COMPLEX*16 (zung2r の場合)
ワークスペース配列 DIMENSION は (*n*)

出力パラメータ

a *m* × *n* のマトリックス *Q* で上書きされる。

info INTEGER
= 0 の場合、正常に終了したことを示す。
< 0 の場合、*info* = -*i* は *i* 番目の引数の値が不正だったことを示す。

?orgl2/?ungl2

?gelqf で求めた *LQ* 因子分解から、全部または一部の直交 / ユニタリ・マトリックス *Q* を生成する (非ブロック化アルゴリズム)。

```
call sorgl2 ( m, n, k, a, lda, tau, work, info )
call dorgl2 ( m, n, k, a, lda, tau, work, info )
call cungl2 ( m, n, k, a, lda, tau, work, info )
call zungl2 ( m, n, k, a, lda, tau, work, info )
```

説明

ルーチン ?org2l/?ung2l は、直交行を持つ *m* × *n* の実数 / 複素マトリックス *Q* を生成する。これは、?gelqf から返されたとおり、次数 *n* の *k* 個の基本リフレクタの積の最初の *m* 行として定義される。

$$Q = H(k) \ldots H(2)\, H(1) \text{ または } Q = H(k)' \ldots H(2)'\, H(1)'$$

入力パラメータ

m INTEGER
マトリックス Q の行数。$m \geq 0$

n INTEGER
マトリックス Q の列数。$n \geq m$

k INTEGER
基本リフレクタ (その積でマトリックス Q を定義) の個数。$m \geq k \geq 0$

a REAL (sorgl2 の場合)
DOUBLE PRECISION (dorgl2 の場合)
COMPLEX (cungl2 の場合)
COMPLEX*16 (zungl2 の場合)
配列 DIMENSION は (*lda*, *n*)
a の i 番目の行には、?gelqf によって配列引数 *a* の最初の *k* 行に返されたとおりに、基本リフレクタ $H(i)$ を定義するベクトルを格納しておかなければならない。ここで、$i = 1, 2, ..., k$

lda INTEGER
配列 *a* の第 1 次元。$lda \geq \max(1, m)$

tau REAL (sorgl2 の場合)
DOUBLE PRECISION (dorgl2 の場合)
COMPLEX (cungl2 の場合)
COMPLEX*16 (zungl2 の場合)
配列 DIMENSION は (*k*)
tau(i) には、?gelqf から返されたとおりに、基本リフレクタ $H(i)$ のスカラー係数を格納しておかなければならない。

work REAL (sorgl2 の場合)
DOUBLE PRECISION (dorgl2 の場合)
COMPLEX (cungl2 の場合)
COMPLEX*16 (zungl2 の場合)
ワークスペース配列 DIMENSION は (*m*)

出力パラメータ

a $m \times n$ のマトリックス Q で上書きされる。

info	INTEGER = 0 の場合、正常に終了したことを示す。 < 0 の場合、*info* = -*i* は *i* 番目の引数の値が不正だったことを示す。

?orgr2/?ungr2

?gerqf で求めた *RQ* 因子分解から、全部または一部の直交 / ユニタリ・マトリックス *Q* を生成する (非ブロック化アルゴリズム)。

```
call sorgr2 ( m, n, k, a, lda, tau, work, info )
call dorgr2 ( m, n, k, a, lda, tau, work, info )
call cungr2 ( m, n, k, a, lda, tau, work, info )
call zungr2 ( m, n, k, a, lda, tau, work, info )
```

説明

ルーチン ?orgr2/?ungr2 は、直交行を持つ $m \times n$ の実数のマトリックス Q を生成する。これは、?gerqf から返されたとおり、次数 n の k 個の基本リフレクタの積の最後の m 行として定義される。

$Q = H(1)\,H(2)\ldots H(k)$ または $Q = H(1)'\,H(2)'\ldots H(k)'$

入力パラメータ

m	INTEGER。マトリックス Q の行数。$m \geq 0$
n	INTEGER マトリックス Q の列数。$n \geq m$
k	INTEGER 基本リフレクタ (その積でマトリックス Q を定義) の個数。$m \geq k \geq 0$
a	REAL (sorgr2 の場合) DOUBLE PRECISION (dorgr2 の場合) COMPLEX (cungr2 の場合)

COMPLEX*16 (zungr2 の場合)
配列 DIMENSION は (*lda*, *n*)
a の (*m*-*k*+*i*) 番目の行には、?gerqf によって配列引数 *a* の最後の *k* 行に返されたとおりに、基本リフレクタ $H(i)$ を定義するベクトルを格納しておかなければならない。ここで、$i = 1, 2, ..., k$

lda INTEGER
配列 *a* の第 1 次元。*lda* ≥ max(1, *m*)

tau REAL (sorgr2 の場合)
DOUBLE PRECISION (dorgr2 の場合)
COMPLEX (cungr2 の場合)
COMPLEX*16 (zungr2 の場合)
配列 DIMENSION は (*k*)。*tau*(*i*) には、?gerqf から返されたとおりに、基本リフレクタ $H(i)$ のスカラー係数を格納しておかなければならない。

work REAL (sorgr2 の場合)
DOUBLE PRECISION (dorgr2 の場合)
COMPLEX (cungr2 の場合)
COMPLEX*16 (zungr2 の場合)
ワークスペース配列 DIMENSION は (*m*)

出力パラメータ

a *m* × *n* のマトリックス Q で上書きされる。

info INTEGER
= 0 の場合、正常に終了したことを示す。
< 0 の場合、*info* = -*i* は *i* 番目の引数の値が不正だったことを示す。

?orm2l/?unm2l

一般マトリックスと ?geqlf で求めた *QL* 因子分解の直交 / ユニタリ・マトリックスを乗算する (非ブロック化アルゴリズム)。

```
call sorm2l ( side, trans, m, n, k, a, lda, tau, c, ldc, work, info )
call dorm2l ( side, trans, m, n, k, a, lda, tau, c, ldc, work, info )
call cunm2l ( side, trans, m, n, k, a, lda, tau, c, ldc, work, info )
call zunm2l ( side, trans, m, n, k, a, lda, tau, c, ldc, work, info )
```

説明

ルーチン ?orm2l/?unm2l は、一般実数 / 複素 $m \times n$ マトリックス C を以下で上書きする。

side = 'L' かつ *trans* = 'N' の場合、$Q*C$、
side = 'L' かつ、*trans* = 'T' (実数型) または
trans = 'C' (複素型) の場合、$Q'*C$、
side = 'R' かつ *trans* = 'N' の場合、$C*Q$、
side = 'R' かつ、*trans* = 'T' (実数型) または
trans = 'C' (複素型) の場合、$C*Q'$

Q は、?geqlf から返されたとおり、k 個の基本リフレクタの積として定義される実数直交または複素ユニタリ・マトリックスである。

$$Q = H(k) \ldots H(2)\, H(1)$$

Q の次数は、*side* = 'L' の場合は m、*side* = 'R' の場合は n である。

入力パラメータ

side CHARACTER*1
= 'L' の場合、Q または Q' を左から適用する。
= 'R' の場合、Q または Q' を右から適用する。

trans CHARACTER*1
= 'N' の場合、Q を適用する (転置なし)。
= 'T' の場合、Q' を適用する (転置、実数型の場合)。
= 'C' の場合、Q' を適用する (共役転置、複素型の場合)。

m INTEGER
マトリックス C の行数。$m \geq 0$

n INTEGER
マトリックス C の列数。$n \geq 0$

k INTEGER
積がマトリックス Q を定義する基本リフレクタの個数。
side = 'L' の場合、$m \geq k \geq 0$、
side = 'R' の場合、$n \geq k \geq 0$

a REAL (sorm2l の場合)
DOUBLE PRECISION (dorm2l の場合)
COMPLEX (cunm2l の場合)
COMPLEX*16 (zunm2l の場合)
配列 DIMENSION は (*lda*, *k*)
a の i 番目の列には、?geqlf によって配列引数 *a* の最後の *k* 列に返されたとおりに、基本リフレクタ $H(i)$ を定義するベクトルを格納しておかなければならない。ここで、$i = 1, 2, ..., k$。配列 *a* はルーチンにより変更されるが終了時に復元される。

lda INTEGER
配列 *a* のリーディング・ディメンジョン。
side = 'L' の場合 $lda \geq \max(1, m)$、
side = 'R' の場合 $lda \geq \max(1, n)$

tau REAL (sorm2l の場合)
DOUBLE PRECISION (dorm2l の場合)
COMPLEX (cunm2l の場合)
COMPLEX*16 (zunm2l の場合)
配列 DIMENSION は (*k*)。*tau*(i) には、?geqlf から返されたとおりに、基本リフレクタ $H(i)$ のスカラー係数を格納しておかなければならない。

c REAL (sorm2l の場合)
DOUBLE PRECISION (dorm2l の場合)
COMPLEX (cunm2l の場合)
COMPLEX*16 (zunm2l の場合)
配列 DIMENSION は (*ldc*, *n*)。*m* × *n* のマトリックス *C* を格納する。

ldc INTEGER
配列 *C* のリーディング・ディメンジョン。
ldc ≥ max(1, *m*)

work REAL (sorm2l の場合)
DOUBLE PRECISION (dorm2l の場合)
COMPLEX (cunm2l の場合)
COMPLEX*16 (zunm2l の場合)
ワークスペース 配列 DIMENSION は
side = 'L' の場合、(*n*)
side = 'R' の場合、(*m*)

出力パラメータ

c *c* は *QC* または *Q'C*、あるいは *CQ'* または *CQ* によって上書きされる。

info INTEGER
= 0 の場合、正常に終了したことを示す。
< 0 の場合、*info* = -*i* は *i* 番目の引数の値が不正だったことを示す。

?orm2r/?unm2r

一般マトリックスと ?geqrf で求めた *QR* 因子分解の直交 / ユニタリ・マトリックスとを乗算する (非ブロック化アルゴリズム)。

```
call sorm2r ( side, trans, m, n, k, a, lda, tau, c, ldc, work, info )
call dorm2r ( side, trans, m, n, k, a, lda, tau, c, ldc, work, info )
call cunm2r ( side, trans, m, n, k, a, lda, tau, c, ldc, work, info )
call zunm2r ( side, trans, m, n, k, a, lda, tau, c, ldc, work, info )
```

説明

ルーチン ?orm2r/?unm2r は、一般実数 / 複素 $m \times n$ マトリックス C を以下で上書きする。

side = 'L' かつ trans = 'N' の場合、$Q*C$、
side = 'L' かつ、trans = 'T' (実数型) または
trans = 'C' (複素型) の場合、$Q'*C$、
side = 'R' かつ trans = 'N' の場合、$C*Q$、
side = 'R' かつ、trans = 'T' (実数型) または
trans = 'C' (複素型) の場合、$C*Q'$

Q は、?geqrf から返されたとおり、k 個の基本リフレクタの積として定義される実数直交または複素ユニタリ・マトリックスである。

$$Q = H(1)\,H(2)\,...\,H(k)$$

Q の次数は、side = 'L' の場合は m、side = 'R' の場合は n である。

入力パラメータ

side CHARACTER*1
= 'L' の場合、Q または Q' を左から適用する。
= 'R' の場合、Q または Q' を右から適用する。

trans CHARACTER*1
= 'N' の場合、Q を適用する (転置なし)。
= 'T' の場合、Q' を適用する (転置、実数型の場合)。
= 'C' の場合、Q' を適用する (共役転置、複素型の場合)

m INTEGER
マトリックス C の行数。$m \geq 0$

n INTEGER
マトリックス C の列数。$n \geq 0$

k INTEGER
基本リフレクタ (その積でマトリックス Q を定義) の個数。
side = 'L' の場合、$m \geq k \geq 0$、
side = 'R' の場合、$n \geq k \geq 0$

a
REAL (sorm2r の場合)
DOUBLE PRECISION (dorm2r の場合)
COMPLEX (cunm2r の場合)
COMPLEX*16 (zunm2r の場合)
配列 DIMENSION は (*lda*, *k*)
a の *i* 番目の列には、?geqrf によって配列引数 *a* の最初の *k* 列に返されたとおりに、基本リフレクタ $H(i)$ を定義するベクトルを格納しておかなければならない。ここで、$i = 1, 2, ..., k$。配列 *a* はルーチンにより変更されるが終了時に復元される。

lda
INTEGER
配列 *a* のリーディング・ディメンジョン。
side = 'L' の場合 $lda \geq \max(1, m)$、
side = 'R' の場合 $lda \geq \max(1, n)$

tau
REAL (sorm2r の場合)
DOUBLE PRECISION (dorm2r の場合)
COMPLEX (cunm2r の場合)
COMPLEX*16 (zunm2r の場合)
配列 DIMENSION は (*k*)
tau(i) には、?geqrf から返されたとおりに、基本リフレクタ $H(i)$ のスカラー係数を格納しておかなければならない。

c
REAL (sorm2r の場合)
DOUBLE PRECISION (dorm2r の場合)
COMPLEX (cunm2r の場合)
COMPLEX*16 (zunm2r の場合)
配列 DIMENSION は (*ldc*, *n*)。$m \times n$ のマトリックス C を格納する。

ldc
INTEGER
配列 *c* のリーディング・ディメンジョン。
$ldc \geq \max(1, m)$

work
REAL (sorm2r の場合)
DOUBLE PRECISION (dorm2r の場合)
COMPLEX (cunm2r の場合)
COMPLEX*16 (zunm2r の場合)
ワークスペース配列 DIMENSION は
side = 'L' の場合、(*n*)
side = 'R' の場合、(*m*)

出力パラメータ

c	*c* は *QC* または *Q'C*、あるいは *CQ'* または *CQ* によって上書きされる。
info	INTEGER = 0 の場合、正常に終了したことを示す。 < 0 の場合、*info* = -*i* は *i* 番目の引数の値が不正だったことを示す。

?orml2/?unml2

一般マトリックスと ?gelqf で求めた *LQ* 因子分解の直交 / ユニタリ・マトリックスとを乗算する (非ブロック化アルゴリズム)。

```
call sorml2 ( side, trans, m, n, k, a, lda, tau, c, ldc, work, info )
call dorml2 ( side, trans, m, n, k, a, lda, tau, c, ldc, work, info )
call cunml2 ( side, trans, m, n, k, a, lda, tau, c, ldc, work, info )
call zunml2 ( side, trans, m, n, k, a, lda, tau, c, ldc, work, info )
```

説明

ルーチン ?orml2/?unml2 は、一般実数 / 複素 $m \times n$ マトリックス *C* を以下で上書きする。

side = 'L' かつ *trans* = 'N' の場合、*Q***C*、
side = 'L' かつ、*trans* = 'T'(実数型) または
trans = 'C' (複素型) の場合、*Q'***C*、
side = 'R' かつ *trans* = 'N' の場合、*C***Q*、
side = 'R' かつ、*trans* = 'T' (実数型) または
trans = 'C' (複素型) の場合、*C***Q'*

Q は、?gelqf から返されたとおり、*k* 個の基本リフレクタの積として定義される実数直交または複素ユニタリ・マトリックスである。

$$Q = H(k) \ldots H(2)\, H(1) \text{ または } Q = H(k)' \ldots H(2)'\, H(1)'$$

Q の次数は、*side* = 'L' の場合は *m*、*side* = 'R' の場合は *n* である。

入力パラメータ

side CHARACTER*1
= 'L' の場合、Q または Q' を左から適用する。
= 'R' の場合、Q または Q' を右から適用する。

trans CHARACTER*1
= 'N' の場合、Q を適用する (転置なし)。
= 'T' の場合、Q' を適用する (転置、実数型の場合)。
= 'C' の場合、Q' を適用する (共役転置、複素型の場合)。

m INTEGER
マトリックス C の行数。$m \geq 0$

n INTEGER
マトリックス C の列数。$n \geq 0$

k INTEGER
基本リフレクタ (その積でマトリックス Q を定義) の個数。
side = 'L' の場合、$m \geq k \geq 0$、
side = 'R' の場合、$n \geq k \geq 0$

a REAL (sorml2 の場合)
DOUBLE PRECISION (dorml2 の場合)
COMPLEX (cunml2 の場合)
COMPLEX*16 (zunml2 の場合)
配列 DIMENSION は
(*lda*, *m*) (*side* = 'L' の場合)
(*lda*, *n*) (*side* = 'R' の場合)
a の i 番目の行には、?gelqf によって配列引数 *a* の最初の *k* 行に返されたとおりに、基本リフレクタ $H(i)$ を定義するベクトルを格納しておかなければならない。ここで、$i = 1, 2, ..., k$。配列 *a* はルーチンにより変更されるが終了時に復元される。

lda INTEGER
配列 *a* のリーディング・ディメンジョン。
lda ≥ max(1, *k*)

tau REAL (sorml2 の場合)
DOUBLE PRECISION (dorml2 の場合)
COMPLEX (cunml2 の場合)
COMPLEX*16 (zunml2 の場合)

配列 DIMENSION は (*k*)
tau(*i*) には、?gelqf から返されたとおりに、基本リフレクタ *H*(*i*) のスカラー係数を格納しておかなければならない。

c REAL (sorml2 の場合)
DOUBLE PRECISION (dorml2 の場合)
COMPLEX (cunml2 の場合)
COMPLEX*16 (zunml2 の場合)
配列 DIMENSION は (*ldc*, *n*)
m × *n* のマトリックス *C* を格納する。

ldc INTEGER
配列 *c* のリーディング・ディメンジョン。
ldc ≥ max(1, *m*)

work REAL (sorml2 の場合)
DOUBLE PRECISION (dorml2 の場合)
COMPLEX (cunml2 の場合)
COMPLEX*16 (zunml2 の場合)
ワークスペース配列 DIMENSION は
side = 'L' の場合、(*n*)
side = 'R' の場合、(*m*)

出力パラメータ

c *c* は *QC* または *Q'C*、あるいは *CQ'* または *CQ* によって上書きされる。

info INTEGER
= 0 の場合、正常に終了したことを示す。
< 0 の場合、*info* = -*i* は *i* 番目の引数の値が不正だったことを示す。

?ormr2/?unmr2

一般マトリックスと ?gerqf で求めた *RQ* 因子分解の直交 / ユニタリ・マトリックスとを乗算する (非ブロック化アルゴリズム)。

```
call sormr2 ( side, trans, m, n, k, a, lda, tau, c, ldc, work, info )
call dormr2 ( side, trans, m, n, k, a, lda, tau, c, ldc, work, info )
call cunmr2 ( side, trans, m, n, k, a, lda, tau, c, ldc, work, info )
call zunmr2 ( side, trans, m, n, k, a, lda, tau, c, ldc, work, info )
```

説明

ルーチン ?ormr2/?unmr2 は、一般の実数 / 複素 $m \times n$ マトリックス C を以下で上書きする。

side = 'L' かつ trans = 'N' の場合、$Q*C$、
side = 'L' かつ、trans = 'T'(実数型) または
trans = 'C' (複素型) の場合、$Q'*C$、
side = 'R' かつ trans = 'N' の場合、$C*Q$、
side = 'R' かつ、trans = 'T' (実数型) または
trans = 'C' (複素型) の場合、$C*Q'$

Q は、?gerqf から返されたとおり、k 個の基本リフレクタの積として定義される実数直交または複素ユニタリ・マトリックスである。

$$Q = H(1)\,H(2)\ldots H(k) \text{ または } Q = H(1)'\,H(2)'\ldots H(k)'$$

Q の次数は、side = 'L' の場合は m、side = 'R' の場合は n である。

入力パラメータ

side　CHARACTER*1
= 'L' の場合、Q または Q' を左から適用する。
= 'R' の場合、Q または Q' を右から適用する。

trans　CHARACTER*1
= 'N' の場合、Q を適用する (転置なし)。
= 'T' の場合、Q' を適用する (転置、実数型の場合)。
= 'C' の場合、Q' を適用する (共役転置、複素型の場合)。

m　INTEGER
マトリックス C の行数。$m \geq 0$

n　INTEGER
マトリックス C の列数。$n \geq 0$

k INTEGER
基本リフレクタ (その積でマトリックス Q を定義) の個数。
side = 'L' の場合、$m \geq k \geq 0$、
side = 'R' の場合、$n \geq k \geq 0$

a REAL (sormr2 の場合)
DOUBLE PRECISION (dormr2 の場合)
COMPLEX (cunmr2 の場合)
COMPLEX*16 (zunmr2 の場合)
配列 DIMENSION は
side = 'L' の場合、(*lda*, *m*)
side = 'R' の場合、(*lda*, *n*)
a の i 番目の行には、?gerqf によって配列引数 *a* の最後の *k* 行に返されたとおりに、基本リフレクタ $H(i)$ を定義するベクトルを格納しておかなければならない。ここで、$i = 1, 2, ..., k$。配列 *a* はルーチンにより変更されるが終了時に復元される。

lda INTEGER
配列 *a* のリーディング・ディメンジョン。
$lda \geq \max(1, k)$

tau REAL (sormr2 の場合)
DOUBLE PRECISION (dormr2 の場合)
COMPLEX (cunmr2 の場合)
COMPLEX*16 (zunmr2 の場合)
配列 DIMENSION は (*k*)
tau(i) には、?gerqf から返されたとおりに、基本リフレクタ $H(i)$ のスカラー係数を格納しておかなければならない。

c REAL (sormr2 の場合)
DOUBLE PRECISION (dormr2 の場合)
COMPLEX (cunmr2 の場合)
COMPLEX*16 (zunmr2 の場合)
配列 DIMENSION は (*ldc*, *n*)
$m \times n$ のマトリックス C を格納する。

ldc INTEGER
配列 C のリーディング・ディメンジョン。
$ldc \geq \max(1, m)$

work REAL (sormr2 の場合)
DOUBLE PRECISION (dormr2 の場合)
COMPLEX (cunmr2 の場合)
COMPLEX*16 (zunmr2 の場合)
ワークスペース配列 DIMENSION は
side = 'L' の場合、(*n*)
side = 'R' の場合、(*m*)

出力パラメータ

c *c* は *QC* または *Q'C*、あるいは *CQ'* または *CQ* によって上書きされる。

info INTEGER
= 0 の場合、正常に終了したことを示す。
< 0 の場合、*info* = -*i* は *i* 番目の引数の値が不正だったことを示す。

?ormr3/?unmr3

一般マトリックスと ?tzrzf で求めた *RZ* 因子分解の直交 / ユニタリ・マトリックスとを乗算する (非ブロック化アルゴリズム)。

```
call sormr3 (side, trans, m, n, k, l, a, lda, tau, c, ldc, work, info)
call dormr3 (side, trans, m, n, k, l, a, lda, tau, c, ldc, work, info)
call cunmr3 (side, trans, m, n, k, l, a, lda, tau, c, ldc, work, info)
call zunmr3 (side, trans, m, n, k, l, a, lda, tau, c, ldc, work, info)
```

説明

このルーチン ?ormr3/?unmr3 は、一般実数 / 複素 $m \times n$ マトリックス *C* を以下で上書きする。

side = 'L' かつ *trans* = 'N' の場合、*Q***C*、
side = 'L' かつ、*trans* = 'T'(実数型) または
trans = 'C' (複素型) の場合、*Q'***C*、

side = 'R' かつ trans = 'N' の場合、$C*Q$、
side = 'R' かつ、trans = 'T' (実数型) または
trans = 'C' (複素型) の場合、$C*Q'$

Q は、? tzrzf から返されたとおり、k 個の基本リフレクタの積として定義される実数直交または複素ユニタリ・マトリックスである。

$$Q = H(1)\,H(2)\,...\,H(k)$$

Q の次数は、side = 'L' の場合は m、side = 'R' の場合は n である。

入力パラメータ

side　CHARACTER*1
= 'L' の場合、Q または Q' を左から適用する。
= 'R' の場合、Q または Q' を右から適用する。

trans　CHARACTER*1
= 'N' の場合、Q を適用する (転置なし)。
= 'T' の場合、Q' を適用する (転置、実数型の場合)。
= 'C' の場合、Q' を適用する (共役転置、複素型の場合)。

m　INTEGER
マトリックス C の行数。$m \geq 0$

n　INTEGER
マトリックス C の列数。$n \geq 0$

k　INTEGER
基本リフレクタ (その積でマトリックス Q を定義) の個数。
side = 'L' の場合、$m \geq k \geq 0$、
side = 'R' の場合、$n \geq k \geq 0$

l　INTEGER
Householder リフレクタとして意味を持った部分が格納されているマトリックス A の列数。
side = 'L' の場合、$m \geq l \geq 0$、
side = 'R' の場合、$n \geq l \geq 0$

a
REAL (sormr3 の場合)
DOUBLE PRECISION (dormr3 の場合)
COMPLEX (cunmr3 の場合)
COMPLEX*16 (zunmr3 の場合)
配列 DIMENSION は
side = 'L' の場合、(*lda*, *m*)
side = 'R' の場合、(*lda*, *n*)
a の *i* 番目の行には、?tzrzf によって配列引数 *a* の最後の *k* 行に返されたとおりに、基本リフレクタ $H(i)$ を定義するベクトルを格納しておかなければならない。ここで、$i = 1, 2, ..., k$。配列 *a* はルーチンにより変更されるが終了時に復元される。

lda
INTEGER
配列 *a* のリーディング・ディメンジョン。
$lda \geq \max(1, k)$

tau
REAL (sormr3 の場合)
DOUBLE PRECISION (dormr3 の場合)
COMPLEX (cunmr3 の場合)
COMPLEX*16 (zunmr3 の場合)
配列 DIMENSION は (*k*)
$tau(i)$ には、?tzrzf から返されたとおりに、基本リフレクタ $H(i)$ のスカラー係数を格納しておかなければならない。

c
REAL (sormr3 の場合)
DOUBLE PRECISION (dormr3 の場合)
COMPLEX (cunmr3 の場合)
COMPLEX*16 (zunmr3 の場合)
配列 DIMENSION は (*ldc*, *n*)
$m \times n$ のマトリックス C を格納する。

ldc
INTEGER
配列 *c* のリーディング・ディメンジョン。
$ldc \geq \max(1, m)$

work
REAL (sormr3 の場合)
DOUBLE PRECISION (dormr3 の場合)
COMPLEX (cunmr3 の場合)
COMPLEX*16 (zunmr3 の場合)
ワークスペース配列 DIMENSION は
side = 'L' の場合、(*n*)
side = 'R' の場合、(*m*)

出力パラメータ

c — *c* は *QC* または *Q'C*、あるいは *CQ'* または *CQ* によって上書きされる。

info — INTEGER
= 0 の場合、正常に終了したことを示す。
< 0 の場合、*info* = -*i* は *i* 番目の引数の値が不正だったことを示す。

?pbtf2

正値対称 / エルミート・マトリックスの
コルスキー因子分解を計算する
(非ブロック化アルゴリズム)。

```
call spbtf2 ( uplo, n, kd, ab, ldab, info )
call dpbtf2 ( uplo, n, kd, ab, ldab, info )
call cpbtf2 ( uplo, n, kd, ab, ldab, info )
call zpbtf2 ( uplo, n, kd, ab, ldab, info )
```

説明

このルーチンは、実数正値対称または複素エルミートのバンド・マトリックス *A* に対してコルスキー因子分解を行う。因子分解は以下の形式を持つ。
A = *U'U*、*uplo* = 'U' の場合、または
A = *LL'*、*uplo* = 'L' の場合

U は上三角マトリックス、*U'* は *U* の転置、*L* は下三角である。
このルーチンはアルゴリズムの非ブロック化版で、レベル 2 BLAS を呼び出す。

入力パラメータ

uplo — CHARACTER*1
対称 / エルミート・マトリックス *A* の上三角または下三角部分のどちらを格納するか指定する。
= 'U' の場合、上三角
= 'L' の場合、下三角

n INTEGER
マトリックス *A* の次数。*n* ≥ 0

kd INTEGER
uplo = 'U' の場合はマトリックス *A* の優対角成分の個数、*uplo* = 'L' の場合はマトリックス *A* の劣対角成分の個数。*kd* ≥ 0

ab REAL (spbtf2 の場合)
DOUBLE PRECISION (dpbtf2 の場合)
COMPLEX (cpbtf2 の場合)
COMPLEX*16 (zpbtf2 の場合)
配列 DIMENSION は (*ldab*, *n*)
対称 / エルミート・バンド・マトリックス *A* の上三角または下三角を、配列の最初の *kd*+1 行に格納する。*A* の *j* 番目の列は配列 *ab* の *j* 番目の列に次のように格納する。
uplo = 'U' の場合、max(1, *j*-*kd*) ≤ *i* ≤ *j* に対して
ab(*kd*+1+*i*-*j*, *j*) = *A*(*i*, *j*)
uplo = 'L' の場合、*j* ≤ *i* ≤ min(*n*, *j*+*kd*) に対して
ab(1+*i*-*j*, *j*) = *A*(*i*, *j*)

ldab INTEGER
配列 *ab* のリーディング・ディメンジョン。
ldab ≥ *kd*+1

出力パラメータ

ab *info* = 0 の場合、バンド・マトリックス *A* のコルスキー因子分解 *A* = *U*' *U* または *A* = *L L*' で得られた三角係数 *U* または *L* が、*A* と同じ格納形式で格納される。

info INTEGER
= 0 の場合、正常に終了したことを示す。
< 0 の場合、*info* = -*k* は *k* 番目の引数の値が不正だったことを示す。
> 0 の場合、*info* = *k* は次数 *k* の先頭の小行列式が正定値でないため因子分解を完了できなかったことを示す。

?potf2

正値対称 / エルミート・マトリックスの
コルスキー因子分解を行う (非ブロック化
アルゴリズム)。

```
call spotf2 ( uplo, n, a, lda, info )
call dpotf2 ( uplo, n, a, lda, info )
call cpotf2 ( uplo, n, a, lda, info )
call zpotf2 ( uplo, n, a, lda, info )
```

説明

ルーチン ?potf2 は、実数正値対称または複素エルミートのマトリックス A に対してコルスキー因子分解を行う。因子分解は以下の形式を持つ。
$A = U'U$、*uplo* = 'U' の場合、または
$A = LL'$、*uplo* = 'L' の場合
ここで U は上三角マトリックスで、L は下三角である。

このルーチンはアルゴリズムの非ブロック化版で、レベル 2 BLAS を呼び出す。

入力パラメータ

uplo CHARACTER*1
対称 / エルミート・マトリックス A の上三角または下三角部分のどちらを格納するか指定する。
= 'U' の場合、上三角
= 'L' の場合、下三角

n INTEGER
マトリックス A の次数。$n \geq 0$

a REAL (spotf2 の場合)
DOUBLE PRECISION (dpotf2 の場合)
COMPLEX (cpotf2 の場合)
COMPLEX*16 (zpotf2 の場合)
配列 DIMENSION は (*lda*, *n*)
対称 / エルミート・マトリックス A を格納する。

uplo = 'U' の場合、a の先頭の $n \times n$ 上三角部分にマトリックス A の上三角部分を格納する。a の厳密な下三角部分は参照されない。
uplo = 'L' の場合、a の先頭の $n \times n$ 下三角部分にマトリックス A の下三角部分を格納する。a の厳密な上三角部分は参照されない。

lda　INTEGER
配列 a のリーディング・ディメンジョン。
$lda \geq \max(1, n)$

出力パラメータ

a　info = 0 の場合、コルスキー因子分解 $A = U' U$ または $A = L L'$ により得られた係数 U または L で上書きされる。

info　INTEGER
= 0 の場合、正常に終了したことを示す。
< 0 の場合、info = -k は k 番目の引数の値が不正だったことを示す。
> 0 の場合、info = k は次数 k の先頭の小行列式が正定値でないため因子分解を完了できなかったことを示す。

?ptts2

?pttrf で計算した L、D、L^H 因子分解を用いて、形式 $AX = B$ の三重対角連立方程式を解く。

```
call sptts2 ( n, nrhs, d, e, b, ldb )
call dptts2 ( n, nrhs, d, e, b, ldb )
call cptts2 ( iuplo, n, nrhs, d, e, b, ldb )
call zptts2 ( iuplo, n, nrhs, d, e, b, ldb )
```

説明

ルーチン ?ptts2 は次の形式の三重対角連立方程式を解く。

$A X = B$

実数型の sptts2/dptts2 は spttrf/dpttrf で計算された A の $L D L'$ 因子分解を使用する。複素数型の cptts2/zptts2 は cpttrf/zpttrf で計算された A の $U'D U$ または $L D L'$ 因子分解を使用する。
D はベクトル *d* で指定される対角マトリックス、U (または L) はその優対角成分 (劣対角成分) がベクトル *e* で指定されている単位二重対角マトリックス、X と B は $n \times nrhs$ のマトリックスである。

入力パラメータ

iuplo INTEGER。複素型でのみ使われる。
因子分解の形式と、ベクトル *e* が上二重単位対角係数 U の優対角成分か下二重単位対角係数 L の劣対角成分のどちらであるかを指定する。
= 1 の場合、$A = U' D U$、*e* は U の優対角成分。
= 0 の場合、$A = L D L'$、*e* は L の劣対角成分。

n INTEGER
三重対角マトリックス A の次数。$n \geq 0$

nrhs INTEGER
右辺の数、すなわち、マトリックス B の列数。
$nrhs \geq 0$

d REAL (sptts2/cptts2 の場合)
DOUBLE PRECISION (dptts2/zptts2 の場合)
配列 DIMENSION は (*n*)
A の因子分解で得られる対角マトリックス D の *n* 対角成分

e REAL (sptts2 の場合)
DOUBLE PRECISION (dptts2 の場合)
COMPLEX (cptts2 の場合)
COMPLEX*16 (zptts2 の場合)
配列 DIMENSION は (*n*-1)
A の LDL' 因子分解から得られる単位二重対角係数 L の (*n*-1) 個の劣対角成分を格納する (実数型、また *iuplo* = 0 の場合は複素型)。

複素型において *iuplo* = 1 の場合、*e* には因子分解 $A = U^{H}DU$ から得られる単位二重対角係数 U の (*n*-1) 個の優対角成分を格納する。

b REAL (sptts2/cptts2 の場合)
DOUBLE PRECISION (dptts2/zptts2 の場合)
配列 DIMENSION は (*ldb*, *nrhs*)
連立方程式に対するベクトル B の右辺。

ldb INTEGER
配列 B のリーディング・ディメンジョン。
ldb ≥ max(1, *n*)

出力パラメータ

b 解ベクトル X で上書きされる。

?rscl

ベクトルに実数スカラーの逆数を乗算する。

```
call srscl ( n, sa, sx, incx )
call drscl ( n, sa, sx, incx )
call csrscl ( n, sa, sx, incx )
call zdrscl ( n, sa, sx, incx )
```

説明

ルーチン ?rscl は、*n* 成分の実数 / 複素ベクトル x に実数スカラー $1/a$ を乗算する。最終結果 x/a がオバーフローまたはアンダアフローを起こさない限り、演算はオバーフローまたはアンダアフローを起こすことなく実行される。

入力パラメータ

n INTEGER
ベクトル x の成分の個数。

sa REAL (srscl/csrscl の場合)
DOUBLE PRECISION (drscl/zdrscl の場合)
ベクトル x の各成分の除算に使用されるスカラー a。
sa は ≥ 0 でなければならない。そうしないとサブルーチンでゼロの除算が発生する。

sx REAL (srscl の場合)
DOUBLE PRECISION (drscl の場合)
COMPLEX (csrscl の場合)
COMPLEX*16 (zdrscl の場合)
配列 DIMENSION は (1+(n-1)*abs(*incx*))
ベクトル x の n 成分。

incx INTEGER
ベクトル *sx* の連続する値の間の増加分。
incx > 0 の場合、*sx*(1) = x(1)、$1 < i \leq n$ に対して
sx(1+(i-1)* *incx*) = x(i)

出力パラメータ

sx 結果 x/a で上書きされる。

?sygs2/?hegs2

?potrf で得られた因子分解の結果を用いて、対称 / エルミート汎用固有値問題を標準形式に縮退させる (非ブロック化アルゴリズム)。

```
call ssygs2 ( itype, uplo, n, a, lda, b, ldb, info )
call dsygs2 ( itype, uplo, n, a, lda, b, ldb, info )
call chegs2 ( itype, uplo, n, a, lda, b, ldb, info )
call zhegs2 ( itype, uplo, n, a, lda, b, ldb, info )
```

説明

ルーチン ?sygs2/?hegs2 は、実数対称または複素エルミートの汎用固有値問題を標準化形式に縮退させる。

itype = 1 の場合、問題は

$$Ax = \lambda Bx$$

となり、A は inv(U')*A*inv(U) または inv(L)*A*inv(L') で上書きされる。

itype = 2 または 3 の場合、問題は

$$ABx = \lambda x \text{ または } B\,Ax = \lambda x$$

となり、A は UAU' または $L'AL$ で上書きされる。B は、$U'U$ または LL' として、?potrf によって事前に因子分解されていなければならない。

入力パラメータ

itype INTEGER

= 1 の場合、inv(U')*A*inv(U) または inv(L)*A*inv(L') を計算する。

= 2 または 3 の場合、UAU' または $L'AL$ を計算する。

uplo CHARACTER

対称 / エルミート・マトリックス A の上三角部分または下三角部分のどちらが格納されているか、および B がどのように因子分解されたかを指定する。

= 'U' の場合、上三角

= 'L' の場合、下三角

n INTEGER

マトリックス A と B の次数。$n \geq 0$

a REAL (ssygs2 の場合)

DOUBLE PRECISION (dsygs2 の場合)

COMPLEX (chegs2 の場合)

COMPLEX*16 (zhegs2 の場合)

配列 DIMENSION は (*lda*, *n*)

対称 / エルミート・マトリックス A を格納する。

uplo = 'U' の場合、*a* の先頭の $n \times n$ 上三角部分にマトリックス A の上三角部分を格納する。*a* の厳密な下三角部分は参照されない。

uplo = 'L' の場合、*a* の先頭の $n \times n$ 下三角部分にマトリックス A の下三角部分を格納する。*a* の厳密な上三角部分は参照されない。

lda INTEGER
配列 *a* のリーディング・ディメンジョン。
lda ≥ max(1, *n*)

b REAL (ssygs2 の場合)
DOUBLE PRECISION (dsygs2 の場合)
COMPLEX (chegs2 の場合)
COMPLEX*16 (zhegs2 の場合)
配列 DIMENSION は (*ldb*, *n*)
?potrf によって返された、*B* のコルスキー因子分解で得られた三角係数。

ldb INTEGER
配列 *B* のリーディング・ディメンジョン。
ldb ≥ max(1, *n*)

出力パラメータ

a *info* = 0 の場合、変換後のマトリックスが *A* と同じ形式で格納される。

info INTEGER
= 0 の場合、正常に終了したことを示す。
< 0 の場合、*info* = -*i* は *i* 番目の引数の値が不正だったことを示す。

?sytd2/?hetd2

直交 / ユニタリ相似変換を用いて、
対称 / エルミート・マトリックスを
実数対称三重対角形式に縮退させる
(非ブロック化アルゴリズム)。

```
call ssytd2 ( uplo, n, a, lda, d, e, tau, info )
call dsytd2 ( uplo, n, a, lda, d, e, tau, info )
call chetd2 ( uplo, n, a, lda, d, e, tau, info )
call zhetd2 ( uplo, n, a, lda, d, e, tau, info )
```

説明

ルーチン ?sytd2/?hetd2 は、直交 / ユニタリ相似変換 ($Q'AQ = T$) を用いて、実数対称 / 複素エルミート・マトリックス A を実数対称三重対角形式 T に縮退させる。

入力パラメータ

uplo CHARACTER*1

対称 / エルミート・マトリックス A の上三角部分または下三角部分のどちらが格納されているかを指定する。
= 'U' の場合、上三角
= 'L' の場合、下三角

n INTEGER

マトリックス A の次数。$n \geq 0$

a REAL (ssytd2 の場合)
DOUBLE PRECISION (dsytd2 の場合)
COMPLEX (chetd2 の場合)
COMPLEX*16 (zhetd2 の場合)

配列 DIMENSION は (*lda*, *n*)
対称 / エルミート・マトリックス A を格納する。
uplo = 'U' の場合、*a* の先頭の $n \times n$ 上三角部分にマトリックス A の上三角部分を格納する。*a* の厳密な下三角部分は参照されない。
uplo = 'L' の場合、*a* の先頭の $n \times n$ 下三角部分にマトリックス A の下三角部分を格納する。*a* の厳密な上三角部分は参照されない。

lda INTEGER

配列 *a* のリーディング・ディメンジョン。
$lda \geq \max(1, n)$

出力パラメータ

a *uplo* = 'U' の場合、*a* の対角成分と最初の優対角成分は三重対角マトリックス T の対応する成分で上書きされる。最初の優対角成分より上の成分は、配列 *tau* とともに、基本リフレクタの積として直交 / ユニタリ・マトリックス Q を表現する。

uplo = 'L' の場合、*a* の対角成分と最初の劣対角成分は三重対角マトリックス *T* の対応する成分で上書きされる。最初の劣対角成分より下の成分は、配列 *tau* とともに、基本リフレクタの積として直交 / ユニタリ・マトリックス *Q* を表現する。

d
REAL (ssytd2/chetd2 の場合)
DOUBLE PRECISION (dsytd2/zhetd2 の場合)
配列 DIMENSION は (*n*)
三重対角マトリックス *T* の対角成分。*d*(*i*) = *a*(*i*, *i*)

e
REAL (ssytd2/chetd2 の場合)
DOUBLE PRECISION (dsytd2/zhetd2 の場合)
配列 DIMENSION は (*n*-1)
三重対角マトリックス *T* の対角線外の成分。
e(*i*) = *a*(*i*, *i*+1) (*uplo* = 'U' の場合)
e(*i*) = *a*(*i*+1, i) (*uplo* = 'L' の場合)

tau
REAL (ssytd2 の場合)
DOUBLE PRECISION (dsytd2 の場合)
COMPLEX (chetd2 の場合)
COMPLEX*16 (zhetd2 の場合)
配列 DIMENSION は (*n*-1)
基本リフレクタのスカラー係数。

info
INTEGER
= 0 の場合、正常に終了したことを示す。
< 0 の場合、*info* = -*i* は *i* 番目の引数の値が不正だったことを示す。

?sytf2

対角ピボット演算法を使用して、
実数 / 複素不定値対称マトリックスの
因子分解を計算する (非ブロック化
アルゴリズム)。

```
call ssytf2 ( uplo, n, a, lda, ipiv, info )
call dsytf2 ( uplo, n, a, lda, ipiv, info )
call ñsytf2 ( uplo, n, a, lda, ipiv, info )
call zsytf2 ( uplo, n, a, lda, ipiv, info )
```

説明

ルーチン ?sytf2 は、Bunch-Kaufman 対角ピボット演算法を使用して、実数 / 複素不定値対称マトリックス A の因子分解を計算する。

$A = U D U'$ または $A = L D L'$

U (または L) は置換マトリックスと単位上 (下) 三角マトリックスの積、U' は U の転置、D は 1×1 と 2×2 の対角ブロックを持つ対称ブロック対角マトリックスである。

このルーチンはアルゴリズムの非ブロック化形式で、レベル 2 BLAS を呼び出す。

入力パラメータ

uplo CHARACTER*1
対称マトリックス A の上三角部分または下三角部分のどちらが格納されているかを指定する。
= 'U' の場合、上三角
= 'L' の場合、下三角

n INTEGER
マトリックス A の次数。$n \geq 0$

a REAL (ssytf2 の場合)
DOUBLE PRECISION (dsytf2 の場合)
COMPLEX (csytf2 の場合)
COMPLEX*16 (zsytf2 の場合)
配列 DIMENSION は (*lda*, *n*)

対称マトリックス A を格納する。
uplo = 'U' の場合、*a* の先頭の $n \times n$ 上三角部分にマトリックス A の上三角部分を格納する。*a* の厳密な下三角部分は参照されない。
uplo = 'L' の場合、*a* の先頭の $n \times n$ 下三角部分にマトリックス A の下三角部を格納する。*a* の厳密な上三角部分は参照されない。

lda INTEGER
配列 *a* のリーディング・ディメンジョン。
lda ≥ max(1, *n*)

出力パラメータ

a ブロック対角マトリックス D と、係数 U または L を得るために使用された乗数が上書きされる。

ipiv INTEGER
配列 DIMENSION は (*n*)
交換の結果と D のブロック構造の各成分が格納される。
ipiv(k) > 0 の場合、*k* 番目の行と列と *ipiv*(k) 番目の行と列は交換された。$D(k, k)$ は 1×1 の対角ブロックである。
uplo = 'U' かつ *ipiv*(k) = *ipiv*(k-1) < 0 の場合、k-1 番目の行と列と -*ipiv*(k) 番目の行と列は交換された。
$D(k\text{-}1{:}k, k\text{-}1{:}k)$ は 2×2 の対角ブロックである。
uplo = 'L' かつ *ipiv*(k) = *ipiv*(k+1) < 0 の場合、k+1 番目の行と列と -*ipiv*(k) 番目の行と列は交換された。
$D(k{:}k+1, k{:}k+1)$ は 2×2 の対角ブロックである。

info INTEGER
= 0 の場合、正常に終了したことを示す。
< 0 の場合、*info* = -*k* は *k* 番目の引数の値が不正だったことを示す。
> 0 の場合、*info* = *k* は $D(k, k)$ が完全に 0 であることを示す。因子分解は完了したが、ブロック対角マトリックス D は完全に特異で、連立方程式の解の算出に D を使用すると 0 での除算が発生する。

?hetf2

対角ピボット演算法を使用して、複素エルミート・マトリックスの因子分解を計算する (非ブロック化アルゴリズム)。

```
call chetf2 ( uplo, n, a, lda, ipiv, info )
call zhetf2 ( uplo, n, a, lda, ipiv, info )
```

説明

このルーチンは、Bunch-Kaufman 対角ピボット演算法を使用して、複素エルミート・マトリックス A の因子分解を計算する。

$A = U D U'$ または $A = L D L'$

U (または L) は置換マトリックスと単位上 (下) 三角マトリックスの積、U' は U の共役転置、D は 1×1 と 2×2 の対角ブロックを持つエルミート・ブロック対角マトリックスである。

このルーチンはアルゴリズムの非ブロック化形式で、Level 2 BLAS を呼び出す。

入力パラメータ

uplo CHARACTER*1
エルミート・マトリックス A の上三角部分または下三角部分のどちらが格納されているかを指定する。
= 'U' の場合、上三角
= 'L' の場合、下三角

n INTEGER
マトリックス A の次数。$n \geq 0$

a COMPLEX (chetf2 の場合)
COMPLEX*16 (zhetf2 の場合)
配列 DIMENSION は (*lda*, *n*)
エルミート・マトリックス A を格納する。
uplo = 'U' の場合、*a* の先頭の $n \times n$ 上三角部分にマトリックス A の上三角部分を格納する。*a* の厳密な下三角部分は参照されない。

uplo = 'L' の場合、a の先頭の n × n 下三角部分にマトリックス A の下三角部を格納する。a の厳密な上三角部分は参照されない。

lda　INTEGER

配列 a のリーディング・ディメンジョン。
lda ≥ max(1, n)

出力パラメータ

a　ブロック対角マトリックス D と、係数 U または L を得るために使用された乗数が上書きされる。

ipiv　INTEGER

配列 DIMENSION は (n)
交換の結果と D のブロック構造の各成分。
ipiv(k) > 0 の場合、k 番目の行と列と ipiv(k) 番目の行と列は交換された。D(k, k) は 1 × 1 の対角ブロックである。
uplo = 'U' かつ ipiv(k) = ipiv(k-1) < 0 の場合、k-1 番目の行と列と -ipiv(k) 番目の行と列は交換された。D(k-1:k, k-1:k) は 2 × 2 の対角ブロックである。
uplo = 'L' かつ ipiv(k) = ipiv(k+1) < 0 の場合、k+1 番目の行と列と -ipiv(k) 番目の行と列は交換された。D(k:k+1, k:k+1) は 2 × 2 の対角ブロックである。

info　INTEGER

= 0 の場合、正常に終了したことを示す。
< 0 の場合、info = -k は k 番目の引数の値が不正だったことを示す。
> 0 の場合、info = k は D(k, k) が完全に 0 であることを示す。因子分解は完了したが、ブロック対角マトリックス D は完全に特異で、連立方程式の解の算出に D を使用すると 0 での除算が発生する。

?tgex2

直交 / ユニタリ等価変換を使用して、(準) 上三角マトリックスのペア中の隣接対角ブロックを交換する。

```
call stgex2 ( wantq, wantz, n, a, lda, b, ldb, q, ldq, z, ldz, j1, n1,
              n2, work, lwork, info )
call dtgex2 ( wantq, wantz, n, a, lda, b, ldb, q, ldq, z, ldz, j1, n1,
              n2, work, lwork, info )
call ctgex2 ( wantq, wantz, n, a, lda, b, ldb, q, ldq, z, ldz, j1,
              info )
call ztgex2 ( wantq, wantz, n, a, lda, b, ldb, q, ldq, z, ldz, j1,
              info )
```

説明

実数ルーチン `stgex2/dtgex2` は、直交等価変換を用いて、(準) 上三角マトリックスのペア (*A*, *B*) の中の 1 × 1 の隣接対角ブロック (*A*11, *B*11) と 2 × 2 の隣接対角ブロック (*A*22, *B*22) を交換する。(*A*, *B*) は汎用実数 Schur 標準形 (`sgges/dgges` により返されたとおり) でなければならない。すなわち、*A* は 1 × 1 と 2 × 2 の対角ブロックを持つ上三角ブロックである。*B* は上三角である。

複素ルーチン `ctgex2/ztgex2` は、ユニタリ等価変換を用いて、上三角マトリックスのペア (*A*, *B*) の中の 1 × 1 の隣接対角ブロック (*A*11, *B*11) と 2 × 2 の隣接対角ブロック (*A*22, *B*22) を交換する。(*A*, *B*) は汎用 Schur 標準形でなければならない。すなわち、*A* と *B* はどちらも上三角である。

すべてのルーチンは、汎用 Schur ベクトルの *Q* と *Z* をオプションで更新する。

Q(in) **A*(in)**Z*(in)' = *Q*(out)**A*(out)* *Z*(out)'
Q(in)**B*(in)**Z*(in)' = *Q*(out)**B*(out)**Z*(out)'

入力パラメータ

wantq LOGICAL。
wantq = .TRUE. の場合、左変換マトリックス Q を更新する。
wantq = .FALSE. の場合、Q を更新しない。

wantz LOGICAL。
wantz = .TRUE. の場合、右変換マトリックス Z を更新する。
wantz = .FALSE. の場合、Z を更新しない。

n INTEGER
マトリックス A と B の次数。

a, b REAL (stgex2 の場合)
DOUBLE PRECISION (dtgex2 の場合)
COMPLEX (ctgex2 の場合)
COMPLEX*16 (ztgex2 の場合)
配列 DIMENSION はそれぞれ、(*lda*, *n*) および (*ldb*, *n*) ペア (A, B) となっているマトリックス A と B を格納する。

lda INTEGER
配列 *a* のリーディング・ディメンジョン。
lda ≥ max(1, *n*)

ldb INTEGER
配列 *b* のリーディング・ディメンジョン。
ldb ≥ max(1, *n*)

q, z REAL (stgex2 の場合)
DOUBLE PRECISION (dtgex2 の場合)
COMPLEX (ctgex2 の場合)
COMPLEX*16 (ztgex2 の場合)
配列 DIMENSION はそれぞれ、(*ldq*, *n*) および (*ldz*, *n*)
wantq = .TRUE. の場合は *q* に直交 / ユニタリ・マトリックス Q を格納し、*wantz* = .TRUE. の場合は *z* に直交 / ユニタリ・マトリックス Z を格納する。

ldq INTEGER
配列 *q* のリーディング・ディメンジョン。*ldq* ≥ 1
wantq = .TRUE. の場合、*ldq* ≥ *n*

ldz INTEGER
配列 *z* のリーディング・ディメンジョン。*ldz* ≥ 1
wantz = .TRUE. の場合、*ldz* ≥ *n*

j1 INTEGER
最初のブロック (*A*11, *B*11) へのインデックス。
1 ≤ *j1* ≤ *n*

n1 INTEGER。実数型でのみ使用される。
最初のブロック (*A*11, *B*11) の次数。*n1* = 0, 1 または 2

n2 INTEGER。実数型でのみ使用される。
2 番目のブロック (*A*22, *B*22) の次数。*n2* = 0, 1 また 2

work REAL (stgex2 の場合)
DOUBLE PRECISION (dtgex2 の場合)
ワークスペース配列 DIMENSION は (*lwork*)
実数型でのみ使用される。

lwork INTEGER
配列 *work* の次元。
lwork ≥ max(*n**(*n2*+*n1*), 2*(*n2*+*n1*)^2)

出力パラメータ

a 更新されたマトリックス *A* で上書きされる。

b 更新されたマトリックス *B* で上書きされる。

q 更新されたマトリックス *Q* で上書きされる。
wantq = .FALSE. の場合は参照されない。

z 更新されたマトリックス *Z* で上書きされる。
wantz = .FALSE. の場合は参照されない。

info INTEGER
= 0 の場合、正常に終了したことを示す。
stgex2/dtgex2 の場合。
info = 1 の場合、変換されたマトリックス (*A, B*) が汎用 Schur 形式からかけ離れていることを示す。ブロックは交換されず、(*A, B*) と (*Q, Z*) は変更されない。すなわち、この問題は悪条件である。*Info* = -16 の場合、*lwork* が小さすぎることを示す。*lwork* の適切な値は *work*(1) に返される。

ctgex2/ztgex2 の場合。
info = 1 の場合、変換されたマトリックス (*A*, *B*) が汎用 Schur 形式からかけ離れていることを示す。すなわち、この問題は悪条件である。(*A*, *B*) は部分的に順序が変更されている可能性があり、*ilst* は移動対象ブロックの現在の位置の最初の行を指している。

?tgsy2

汎用シルベスター式を解く (非ブロック化アルゴリズム)。

```
call stgsy2 ( trans, ijob, m, n, a, lda, b, ldb, c, ldc, d, ldd, e,
              lde, f, ldf, scale, rdsum, rdscal, iwork, pq, info )
call dtgsy2 ( trans, ijob, m, n, a, lda, b, ldb, c, ldc, d, ldd, e,
              lde, f, ldf, scale, rdsum, rdscal, iwork, pq, info )
call ctgsy2 ( trans, ijob, m, n, a, lda, b, ldb, c, ldc, d, ldd, e,
              lde, f, ldf, scale, rdsum, rdscal, info )
call ztgsy2 ( trans, ijob, m, n, a, lda, b, ldb, c, ldc, d, ldd, e,
              lde, f, ldf, scale, rdsum, rdscal, info )
```

説明

ルーチン ?tgsy2 は、レベル 1 BLAS とレベル 2 BLAS を使用して、次の汎用シルベスター式を解く。

$$AR - LB = \text{scale} * C \qquad (1)$$

$$DR - LE = \text{scale} * F、$$

R と *L* は未知の $m \times n$ マトリックス、(*A*, *D*)、(*B*, *E*)、(*C*, *F*) はそれぞれ $m \times m$、$n \times n$、$m \times n$ の与えられたマトリックスのペアである。
stgsy2/dtgsy2 の場合、ペア (*A*, *D*) と (*B*, *E*) は汎用 Schur 標準形でなければならない。すなわち、*A* と *B* は上準三角マトリックスで、*D* と *E* は上三角マトリックスでなければならない。ctgsy2/ztgsy2 の場合、マトリックス *A* と *B* と *D* と *E* は上三角マトリックスである (すなわち、(*A*, *D*) と (*B*, *E*) は汎用 Schur 形)。

(C, F) は解 (R, L) で上書きされる。$0 \le$ *scale* ≤ 1 はオーバーフローを避けるために選択された出力スケール係数である。

マトリックス表記では、方程式 (1) を解くことは次式を解くことに対応する。

$$Zx = \text{scale} * b$$

ただし、Z は以下のように定義される。

$$Z = \begin{bmatrix} \text{kron}(I_n, A) & -\text{kron}(B', I_m) \\ \text{kron}(I_n, D) & -\text{kron}(E', I_m) \end{bmatrix} \qquad (2)$$

I_k はサイズが k の単位マトリックス、X' は X の転置である。
kron(X, Y) はマトリックス X と Y の Kronecker 積を表わす。

trans = 'T' の場合、次の転置 (共役転置) 式を y について解く。

$$Z'y = \text{scale} * b$$

これは、以下の式を R、L に対して解くことと等価である。

$$\begin{aligned} A'R + D'L &= \text{scale} * C \\ RB' + LE' &= \text{scale} * (-F) \end{aligned} \qquad (3)$$

上の事例は、?lacon による逆コミュニケーションを使用した、
Dif$[(A, D), (B, E)]$ = *sigma_min*(Z) の推定の計算に用いられる。

?tgsy2 はまた (*ijob* ≥ 1 に対して)、?tgsyl における 2 つのマトリックス・ペア間の隔たりの上限計算に影響する。入力 (A, D)、(B, E) は、?tgsyl においてマトリックス・ペアの部分行列束である。詳細は ?tgsyl を参照。

入力パラメータ

trans CHARACTER。
trans = 'N' の場合、汎用シルベスター方程式 (1) を解く。
trans = 'T' の場合、転置された式 (3) を解く。

ijob INTEGER
実行する機能を指定する。
ijob = 0 の場合、(1) のみを解く。
ijob = 1 の場合、この部分式を始点として、2 個のマトリックス・ペア間の隔たりの Frobenius ノルムにもとづく推定までの影響が計算される (先読み法が用いられる)。

ijob = 2 の場合、この部分式を始点として、2 個のマトリックス・ペア間の隔たりの Frobenius ノルムにもとづく推定までの影響が計算される (部分式に対して ?gecon が用いられる)。
trans = 'T' の場合は参照されない。

m
INTEGER
m は、*A* と *D* の次数、および *C*、*F*、*R*、*L* の行次元を指定する。

n
INTEGER
n は、*B* と *E* の次数、および *C*、*F*、*R*、*L* の列次元を指定する。

a, b
REAL (stgsy2 の場合)
DOUBLE PRECISION (dtgsy2 の場合)
COMPLEX (ctgsy2 の場合)
COMPLEX*16 (ztgsy2 の場合)
配列 DIMENSION はそれぞれ、(*lda*, *m*) および (*ldb*, *n*)。*a* には上 (準) 三角マトリックス *A* を、*b* には上 (準) 三角マトリックス *B* を格納する。

lda
INTEGER
配列 *a* のリーディング・ディメンジョン。
lda ≥ max(1, *m*)

ldb
INTEGER
配列 *b* のリーディング・ディメンジョン。
ldb ≥ max(1, *n*)

c, f
REAL (stgsy2 の場合)
DOUBLE PRECISION (dtgsy2 の場合)
COMPLEX (ctgsy2 の場合)
COMPLEX*16 (ztgsy2 の場合)
配列 DIMENSION はそれぞれ、(*ldc*, *n*) および (*ldf*, *n*)。*c* には (1) の最初のマトリックス方程式の右辺を、*f* には (1) の 2 番目のマトリックス方程式の右辺を格納する。

ldc INTEGER
配列 *c* のリーディング・ディメンジョン。
ldc ≥ max(1, *m*)

d, e REAL (stgsy2 の場合)
DOUBLE PRECISION (dtgsy2 の場合)
COMPLEX (ctgsy2 の場合)
COMPLEX*16 (ztgsy2 の場合)
配列 DIMENSION はそれぞれ、(*ldd*, *m*) および (*lde*, *n*)。*d* には上三角マトリックス *D* を、*e* には上三角マトリックス *E* を格納する。

ldd INTEGER
配列 *d* のリーディング・ディメンジョン。
ldd ≥ max(1, *m*)

lde INTEGER
配列 *e* のリーディング・ディメンジョン。
lde ≥ max(1, *n*)

ldf INTEGER
配列 *f* のリーディング・ディメンジョン。
ldf ≥ max(1, *m*)

rdsum REAL (stgsy2/ctgsy2 の場合)
DOUBLE PRECISION (dtgsy2/ztgsy2 の場合)
?tgsyl で得られた Dif 推定に対する、計算された影響の二乗和を格納する。ここでスケール係数 *rdscal* は因子分解されている。

rdscal REAL (stgsy2/ctgsy2 の場合)
DOUBLE PRECISION (dtgsy2/ztgsy2 の場合)
rdsum のオーバーフローを防ぐために使用されるスケール係数を格納する。

iwork INTEGER。実数型でのみ使用される。
ワークスペース配列 DIMENSION は (*m*+*n*+2)

出力パラメータ

c *ijob* = 0 の場合、*c* は解 *R* で上書きされる。

f *ijob* = 0 の場合、*f* は解 *L* で上書きされる。

scale REAL (stgsy2/ctgsy2 の場合)
DOUBLE PRECISION (dtgsy2/ztgsy2 の場合)
$0 \le$ *scale* ≤ 1 が格納される。$0 <$ *scale* < 1 の場合、解 R と解 L (入力時は C と F) にはそれぞれ、わずかに摂動する系の解が入るが、入力マトリックス A、B、D、E は変更されない。*scale* = 0 の場合、R と L にはそれぞれ $C = F = 0$ の均一な系の解が入る。通常は *scale* = 1 である。

rdsum 対応する二乗和は、現在の部分式から得られた影響によって更新される。
trans = 'T' の場合、*rdsum* は使われない。
*rdsum*は ?tgsy2 が ?tgsyl から呼び出されたときのみ意味を持つことに注意する。

rdscal *rdscal* は、*rdsum* に格納されている現在の影響に関して更新される。
trans = 'T' の場合、*rdscal* は使われない。
rdscal は ?tgsy2 が ?tgsyl から呼び出されたときのみ意味を持つことに注意する。

pq INTEGER。実数型でのみ使用される。ルーチン stgsy2/dtgsy2 で解かれた部分式 (2×2、4×4、8×8 のサイズ) の個数が格納される。

info INTEGER
ルーチン終了時、*info* が以下の場合、
= 0 の場合、正常に終了したことを示す。
< 0 の場合、*info* = -*i* は *i* 番目の引数の値が不正だったことを示す。
> 0 の場合、マトリックスのペア (A, D) と (B, E) は、同じ固有値か、きわめて近い固有値を持つ。

?trti2

三角マトリックスの逆行列を計算する
(非ブロック化アルゴリズム)。

```
call strti2 ( uplo, diag, n, a, lda, info )
call dtrti2 ( uplo, diag, n, a, lda, info )
call ctrti2 ( uplo, diag, n, a, lda, info )
call ztrti2 ( uplo, diag, n, a, lda, info )
```

説明

ルーチン ?trti2 は実数 / 複素の上三角または下三角マトリックスの逆行列を計算する。

このルーチンは、アルゴリズムの Level 2 BLAS バージョンである。

入力パラメータ

uplo CHARACTER*1
マトリックス *A* が上または下三角のどちらかを指定する。
= 'U' の場合、上三角
= 'L' の場合、下三角

diag CHARACTER*1
マトリックス *A* が単位三角かどうかを指定する。
= 'N' の場合、単位三角ではない。
= 'U' の場合、単位三角。

n INTEGER
マトリックス *A* の次数。$n \geq 0$

a REAL (strti2 の場合)
DOUBLE PRECISION (dtrti2 の場合)
COMPLEX (ctrti2 の場合)
COMPLEX*16 (ztrti2 の場合)
配列 DIMENSION は (*lda*, *n*)
三角マトリックス *A* を格納する。*uplo* = 'U' の場合、配列 *a* の先頭の $n \times n$ 上三角部分に上三角マトリックスを格納する。厳密な下三角部分は参照されない。

uplo = 'L' の場合、配列 *a* の先頭の *n* × *n* 下三角部分に下三角マトリックスを格納する。厳密な上三角部は参照されない。*diag* = 'U' の場合も対角成分は参照されず 1 とみなされる。

lda INTEGER
配列 *a* のリーディング・ディメンジョン。
lda ≥ max(1,*n*)

出力パラメータ

a 元のマトリックスの (三角) 逆マトリックスで、同じ格納形式で上書きされる。

info INTEGER
= 0 の場合、正常に終了したことを示す。
< 0 の場合、*info* = -*k* は *k* 番目の引数の値が不正だったことを示す。

xerbla

LAPACK ルーチンから呼び出される
エラー処理ルーチン。

```
call xerbla ( srname, info )
```

説明

ルーチン *xerbla* は LAPACK ルーチンのエラー処理ルーチンである。LAPACK ルーチンで不正な入力パラメータの値があったときに呼び出される。メッセージを出力し実行を停止する。
LAPACK を導入する場合に、システム固有の例外処理機能を呼び出すのであれば、*stop* 文の変更を考慮するとよい。

入力パラメータ

*srname*CHARACTER*6。
xerbla を呼び出したルーチンの名前。

info INTEGER
呼び出しルーチンのパラメータ・リストにおいて、不正なパラメータの位置。

ベクトル数学関数

7

本章では、ベクトル引数に基づいて基本関数を計算するベクトル数学関数ライブラリ (VML) について説明する。VML は、インテル ® マス・カーネル・ライブラリは、マス・カーネル・ライブラリの必須部分である。ここでは、これらの関数の説明を簡単にするために、VML の用語を使用する。

VML には、ベクトルを演算するためのコアとなる高級な数学関数のセット (累乗、三角方程式、指数、双曲線など) が、高度な最適化をし実装されている。

VML により、非線形プログラミング・ソフトウェアや積分計算などの多くのプログラムの性能が大幅に向上する。

VML 関数は、実行する演算に応じて、以下のグループに分けられる。

- VML 数学関数。単位増分インデックスを使用するベクトルに基づいて、基本関数 (正弦、余弦、指数、対数など) の値を計算する。
- VML Pack/Unpack 関数。正増分インデックス、ベクトル・インデックス、マスク・インデックスを使用するベクトルへの変換、およびその逆の変換を実行する (ベクトル・インデックス方式の詳細は、付録 A を参照のこと)。
- VML サービス関数。精密モードの設定 / 取得、エラー・コードの設定 / 取得を行う。

VML 数学関数は、入力ベクトルを引数として処理し、それぞれの基本関数の値を成分単位で計算して、結果を出力ベクトルに戻す。

データ・タイプと精密モード

VML の数学ベクトル関数と pack/unpack ベクトル関数は、単精度実数データと倍精度実数データのベクトル引数に対応するように実装されている。ライブラリには、すべての関数に対する Fortran インターフェイスと C インターフェイスが、VML サービス関数とともに用意されている。Fortran インターフェイスと C インターフェイスにおける関数の命名方法と呼び出し方法の違いは、この後の関数の命名規則の項で説明する。

VML の各ベクトル関数は、(各データ形式ごとに) 高精度 (HA) と低精度 (LA) の 2 つのモードで機能できる。多くの関数の場合、LA モードを使用すると、精度を落とす代わりに性能が向上する。
しかし、精度を落としても性能がほとんど向上しないときもあるため、1 つの関数に両方のモードが用意されている。エラーの動作は、HA モードと LA モードのどちらを選択しているかだけではなく、ソフトウェアを実行するプロセッサによっても変わってくる。
また、特殊な値に対する動作は、関数で HA モードと LA モードのどちらを使用しているかにより異なる場合がある。精度に対する動作は、VML のリリース・ノートを参照のこと。

HA モードと LA モードを切り替えるには、`vmlSetMode(mode)` を使用する (表 7-11 を参照)。関数 `vmlGetMode()` は、現在使用しているモードを戻す。デフォルトでは、高精度 (HA) モードが使用される。

関数規則

Fortran インターフェイスの場合、VML 関数のフルネームには小文字しか含まれない。C インターフェイスの場合は、小文字と大文字が混在して含まれる。

注：関数命名規則は本マニュアルの関数記述に従った。ライブラリ中の実際の関数名は、小文字と大文字の使用に関して若干異なる部分がある。サポート対象のコンパイラ要件を満足する。

VML 数学関数と pack/unpack 関数のフルネームは、以下の形式で指定する。

```
v <p> <name> <mod>
```

先頭文字の v は、関数が VML に属していることを示すプリフィックスである。
<p> フィールドは、以下のデータ・タイプを指定する精度プリフィックスである。

- s　REAL (Fortran インターフェイスの場合)、float (C インターフェイスの場合)
- d　DOUBLE PRECISION (Fortran インターフェイスの場合)、double (C インターフェイスの場合)

<name> フィールドには、関数の短縮名を指定する。C インターフェイスの場合は、一部大文字が含まれる。(表 7-2、表 7-9 を参照)。

<mod> フィールド (C インターフェイスの場合は大文字で記述) は、pack/unpack 関数でのみ指定する。次に示すインデックス方式を指定する。

- i　正増分を使用するインデックス
- v　インデックス・ベクトルを使用するインデックス
- m　マスク・ベクトルを使用するインデックス

VML サービス関数のフルネームは、次の形式で指定する。

```
vml <name>
```

vml は関数が VML に属しているのを示すプリフィックスであり、<name> は関数の短縮名である (C インターフェイスの場合は一部大文字が含まれる。表 7-10 を参照のこと)。
アプリケーション・プログラムから VML 関数を呼び出すには、従来の関数呼び出しを使用する。例えば、単精度データの VML 指数関数は、次のように呼び出せる。

```
call vsexp ( n, a, y )
```
(Fortran インターフェイスの場合)

```
vsExp ( n, a, y );
```
(C インターフェイスの場合)

関数インターフェイス

VML 関数へのインターフェイスには、関数のフルネームと一連の引数が含まれる。
それぞれの VML 関数グループにおける Fortran インターフェイスと C インターフェイスの説明を以下に示す。関数によっては、引数として *a* と *b* の 2 つの入力ベクトルを持つものがある (Div、Pow、Atan2)。また、SinCos 関数は、*y* と *z* の 2 つの出力ベクトルを持つ。

VML 数学関数 :

Fortran:

```
call v<p><name>( n, a, y )
call v<p><name>( n, a, b, y )
call v<p><name>( n, a, y, z )
```

C:

```
v<p><name>( n, a, y );
v<p><name>( n, a, b, y );
v<p><name>( n, a, y, z );
```

Pack 関数 :

Fortran:

```
call v<p>packi( n, a, inca, y )
call v<p>packv( n, a, ia, y )
call v<p>packm( n, a, ma, y )
```

C:

```
v<p>PackI( n, a, inca, y );
v<p>PackV( n, a, ia, y );
v<p>PackM( n, a, ma, y );
```

Unpack 関数 :

Fortran:

```
call v<p>unpacki( n, a, y, incy )
call v<p>unpackv( n, a, y, iy )
call v<p>unpackm( n, a, y, my )
```

C:

```
v<p>UnpackI( n, a, y, incy );
v<p>UnpackV( n, a, y, iy );
v<p>UnpackM( n, a, y, my );
```

サービス関数 :

Fortran:

```
oldmode = vmlsetmode( mode )
mode    = vmlgetmode( )
olderr  =  vmlseterrstatus ( err )
```

```
err      =  vmlgeterrstatus( )
olderr   =  vmlclearerrstatus( )
oldcallback = vmlseterrorcallback( callback )
callback    = vmlgeterrorcallback( )
oldcallback = vmlclearerrorcallback( )
```

C:

```
oldmode = vmlSetMode( mode );
mode    = vmlGetMode( void );
olderr  = vmlSetErrStatus ( err );
err     = vmlGetErrStatus(void);
olderr  = vmlClearErrStatus(void);
oldcallback = vmlSetErrorCallBack(callback );
callback    = vmlGetErrorCallBack( void );
oldcallback = vmlClearErrorCallBack(void );
```

入力パラメータ：

n	計算する成分の数
a	1 番目の入力ベクトル
b	2 番目の入力ベクトル
inca	入力ベクトル *a* のベクトル増分
ia	入力ベクトル *a* のインデックス・ベクトル
ma	入力ベクトル *a* のマスク・ベクトル
incy	出力ベクトル *y* のベクトル増分
iy	出力ベクトル *y* のインデックス・ベクトル
my	出力ベクトル *y* のマスク・ベクトル
err	エラー・コード
mode	VML モード
callback	コールバック関数のアドレス

出力パラメータ：

y	1 番目の出力ベクトル
z	2 番目の出力ベクトル
err	前のエラー・コード
mode	VML モード
olderr	前のエラー・コード

oldmode	前の VML モード
oldcallback	前のコールバック関数のアドレス

各関数で使用するパラメータのデータ・タイプは、それぞれの関数の説明の項で記述する。VML 数学関数はすべて、インプレース演算を実行できる。つまり、同一のベクトルを入力パラメータとしても出力パラメータとしても使用可能である。これは、2 つの入力ベクトルを持つ関数にも当てはまる。この場合、2 つの入力パラメータの一方は、出力ベクトルで上書きできる。2 つの出力ベクトルを持つ関数の場合、出力ベクトルの一方は、入力ベクトルと一致する。

ベクトル・インデックス方式

現在の VML 数学関数は、単位増分を使用しないと機能しない。他の増分を使用する配列、またはより複雑なインデックスを使用する配列を収容するには、成分を収集して連続するベクトルを生成し、計算の完了後、それらを分散すればよい。
VML では、ベクトル成分の収集 / 分散に以下のインデックス方式を使用する。

- 正増分
- インデックス・ベクトル
- マスク・ベクトル

関数で使用するインデックス方式は、インデックス修飾子で指定する (関数の命名規則の <mod> フィールドの説明を参照)。インデックス方式の詳細は、付録 A の VML のベクトル引数を参照のこと。

エラー診断

VML ライブラリは、独自のエラー・ハンドラを備えている。C インターフェイスと Fortran インターフェイスでは、異なる点が 1 つだけある。それは、Fortran インターフェイスでは、VML 関数がエラーを検出した後で、インテル MKL エラー報告ルーチン XERBLA の呼び出しが可能なことである。このルーチンは、VML_STATUS_BADSIZE と VML_STATUS_BADMEM 入力エラーに関する情報を取得する(表7-13を参照)。

VML エラー・ハンドラは、以下のような特性を持つ。

1) エラー・ステータス (vmlErrStatus) グローバル変数が、VML 関数をそれぞれ呼び出した後で設定される。この変数が取り得る値については、表 7-13 に示している。

2) VML モードに応じて、エラー・ハンドラ関数は以下を呼び出す。

- errno 変数設定。errno が取り得る値を表 7-1 に示す。
- stderr ストリームへのエラー・テキスト情報の書き込み
- エラー時の適切な例外の生成 (必要な場合)
- 追加のエラー・ハンドラ・コールバック関数の呼び出し

表 7-1 errno 変数の値の設定

errno の値	説明
0	検出されたエラーがない。
EINVAL	配列の次元が正でない。
EACCES	NULL ポインタが渡された。
EDOM	配列の値の少なくとも 1 つが、定義の範囲外。
ERANGE	配列の値の少なくとも 1 つが、特異点、オーバーフロー、またはアンダーフローを引き起こした。

VML 数学関数

この項では、単位増分を使用する実数のベクトル引数に基づいて基本数学関数の値を計算する VML 関数について説明する。
それぞれの関数グループの説明では、短縮名と機能の簡単な説明を記載するとともに、Fortran インターフェイスと C インターフェイスの両方に対応した各データ・タイプの呼び出しシーケンスを紹介し、入力 / 出力引数の説明も記載している。

すべての VML 数学関数において、パラメータの入力範囲は、それぞれのデータ・タイプに定められた一連の値の定義の数学的な範囲に等しい。VML 関数の一部、具体的には、Div、Exp、Sinh、Cosh、Pow は、結果がオーバーフローする場合がある。これらの関数における各精度でのオーバーフローの条件となるそれぞれの入力しきい値については、関数の説明の項で記述する。これらの仕様では、FLT_MAX は単精度データ・タイプで表現可能な最大数を表し、DBL_MAX は倍精度データ・タイプで表現可能な最大数を表す。

表 7-2 は、使用可能な数学関数とそれに関連するデータ・タイプをまとめたものである。

表 7-2　VML 数学関数

関数の短縮名	データ・タイプ	説明
累乗関数と累乗根関数		
Inv	s, d	ベクトル成分の逆転
Div	s, d	1 つのベクトルの成分を 2 番目のベクトルの成分で割る
Sqrt	s, d	ベクトル成分の平方根
InvSqrt	s, d	ベクトル成分の逆平方根
Cbrt	s, d	ベクトル成分の立方根
InvCbrt	s, d	ベクトル成分の逆立方根
Pow	s, d	各ベクトル成分を指定された値で累乗する
Powx	s, d	各ベクトル成分を定数で累乗する

表 7-2 VML 数学関数 (続き)

関数の短縮名	データ・タイプ	説明
指数関数と対数関数		
Exp	s, d	ベクトル成分の指数
Ln	s, d	ベクトル成分の自然対数
Log10	s, d	ベクトル成分の 10 進対数
三角方程式関数		
Cos	s, d	ベクトル成分の余弦
Sin	s, d	ベクトル成分の正弦
SinCos	s, d	ベクトル成分の正弦および余弦
Tan	s, d	ベクトル成分の正接
Acos	s, d	ベクトル成分の逆余弦
Asin	s, d	ベクトル成分の逆正弦
Atan	s, d	ベクトル成分の逆正接
Atan2	s, d	2 つのベクトルの成分の四分円逆正接
双曲線関数		
Cosh	s, d	ベクトル成分の双曲余弦
Sinh	s, d	ベクトル成分の双曲正弦
Tanh	s, d	ベクトル成分の双曲正接
Acosh	s, d	ベクトル成分の逆双曲余弦 (非負)
Asinh	s, d	ベクトル成分の逆双曲正弦
Atanh	s, d	ベクトル成分の逆双曲正接
特殊関数		
Erf	s, d	ベクトル成分の誤差関数値
Erfc	s, d	ベクトル成分の相補誤差関数

Inv

ベクトルの逆転を成分単位で実行する。

Fortran:

```
call vsinv( n, a, y )
call vdinv( n, a, y )
```

C:

```
vsInv( n, a, y );
vdInv( n, a, y );
```

入力パラメータ

Fortran:

n INTEGER, INTENT(IN)。計算する成分の数を指定する。

a REAL, INTENT(IN) (vsinv の場合)
DOUBLE PRECISION, INTENT(IN) (vdinv の場合)
配列、入力ベクトル a を指定する。

C:

n int。計算する成分の数を指定する。

a const float* (vsInv の場合)
const double* (vdInv の場合)
入力ベクトル a を格納する配列へのポインタ。

出力パラメータ

Fortran:

y REAL (vsinv の場合)
DOUBLE PRECISION (vdinv の場合)
配列、出力ベクトル y を指定する。

C:

y	float* (vsInv の場合) double* (vdInv の場合) 出力ベクトル y を格納する配列へのポインタ。

Div

ベクトル *a* をベクトル *b* で成分単位で割る。

Fortran:

```
call vsdiv( n, a, b, y )
call vddiv( n, a, b, y )
```

C:

```
vsDiv( n, a, b, y );
vdDiv( n, a, b, y );
```

入力パラメータ

Fortran:

n	INTEGER, INTENT(IN)。計算する成分の数を指定する。
a、*b*	REAL, INTENT(IN) (vsdiv の場合) DOUBLE PRECISION, INTENT(IN) (vddiv の場合) 配列、入力ベクトル a と b を指定する。

C:

n	int。計算する成分の数を指定する。
a、*b*	const float* (vsDiv の場合) const double* (vdDiv の場合) 入力ベクトル a と b を格納する配列へのポインタ。

表 7-3 Div 関数における各精度でのオーバーフローしきい値

データ・タイプ	入力パラメータに対するしきい値制限
単精度	abs(*a*[i]) < abs(*b*[i]) * FLT_MAX
倍精度	abs(*a*[i]) < abs(*b*[i]) * DBL_MAX

出力パラメータ

Fortran:

y REAL (vsdiv の場合)
DOUBLE PRECISION (vddiv の場合)
配列、出力ベクトル y を指定する。

C:

y float* (vsDiv の場合)
double* (vdDiv の場合)
出力ベクトル y を格納する配列へのポインタ。

Sqrt

ベクトル成分の平方根を計算する。

Fortran:

```
call vssqrt( n, a, y )
call vdsqrt( n, a, y )
```

C:

```
vsSqrt( n, a, y );
vdSqrt( n, a, y );
```

入力パラメータ

Fortran:

n INTEGER, INTENT(IN)。計算する成分の数を指定する。

a REAL, INTENT(IN) (vssqrt の場合)
DOUBLE PRECISION, INTENT(IN) (vdsqrt の場合)
配列、入力ベクトル a を指定する。

C:

n int。計算する成分の数を指定する。

a const float* (vsSqrt の場合)
const double* (vdSqrt の場合)
入力ベクトル a を格納する配列へのポインタ。

出力パラメータ

Fortran:

y REAL (vssqrt の場合)
DOUBLE PRECISION (vdsqrt の場合)
配列、出力ベクトル y を指定する。

C:

y float* (vsSqrt の場合)
double* (vdSqrt の場合)
出力ベクトル y を格納する配列へのポインタ。

InvSqrt

ベクトル成分の逆平方根を計算する。

Fortran:

```
call vsinvsqrt( n, a, y )
call vdinvsqrt( n, a, y )
```

C:

```
vsInvSqrt( n, a, y );
vdInvSqrt( n, a, y );
```

入力パラメータ

Fortran:

n INTEGER, INTENT(IN)。計算する成分の数を指定する。

a REAL, INTENT(IN) (vsinvsqrt の場合)
DOUBLE PRECISION, INTENT(IN) (vdinvsqrt の場合)
配列、入力ベクトル a を指定する。

C:

n int。計算する成分の数を指定する。

a const float* (vsInvSqrt の場合)
const double* (vdInvSqrt の場合)
入力ベクトル a を格納する配列へのポインタ。

出力パラメータ

Fortran:

y REAL (vsinvsqrt の場合)
DOUBLE PRECISION (vdinvsqrt の場合)
配列、出力ベクトル y を指定する。

C:

y float* (vsInvSqrt の場合)
double* (vdInvSqrt の場合)
出力ベクトル y を格納する配列へのポインタ。

Cbrt

ベクトル成分の立方根を計算する。

Fortran:

```
call vscbrt( n, a, y )
call vdcbrt( n, a, y )
```

C:

```
vsCbrt( n, a, y );
vdCbrt( n, a, y );
```

入力パラメータ

Fortran:

n INTEGER, INTENT(IN)。計算する成分の数を指定する。

a REAL, INTENT(IN) (vscbrt の場合)
DOUBLE PRECISION, INTENT(IN) (vdcbrt の場合)
配列、入力ベクトル a を指定する。

C:

n int。計算する成分の数を指定する。

a const float* (vsCbrt の場合)
const double* (vdCbrt の場合)
入力ベクトル a を格納する配列へのポインタ。

出力パラメータ

Fortran:

y REAL (vscbrt の場合)
DOUBLE PRECISION (vdcbrt の場合)
配列、出力ベクトル y を指定する。

C:

y float* (vsCbrt の場合)
double* (vdCbrt の場合)
出力ベクトル y を格納する配列へのポインタ。

InvCbrt

ベクトル成分の逆立方根を計算する。

Fortran:

```
call vsinvcbrt( n, a, y )
call vdinvcbrt( n, a, y )
```

C:

```
vsInvCbrt( n, a, y );
vdInvCbrt( n, a, y );
```

入力パラメータ

Fortran:

n INTEGER, INTENT(IN)。計算する成分の数を指定する。

a REAL, INTENT(IN) (vsinvcbrt の場合)
DOUBLE PRECISION, INTENT(IN) (vdinvcbrt の場合)
配列、入力ベクトル a を指定する。

C:

n int。計算する成分の数を指定する。

a const float* (vsInvCbrt の場合)
const double* (vdInvCbrt の場合)
入力ベクトル a を格納する配列へのポインタ。

出力パラメータ

Fortran:

y REAL (vsinvcbrt の場合)
DOUBLE PRECISION (vdinvcbrt の場合)
配列、出力ベクトル y を指定する。

C:

y float* (vsInvCbrt の場合)
double* (vdInvCbrt の場合)
出力ベクトル y を格納する配列へのポインタ。

Pow

2 つのベクトルの成分において、*a* の *b* 乗を計算する。

Fortran:

```
call vspow( n, n, n, n )
call vdpow( n, n, n, n )
```

C:

```
vsPow( n, a, b, y );
vdPow( n, a, b, y );
```

入力パラメータ

Fortran:

n	INTEGER, INTENT(IN)。計算する成分の数を指定する。
a、*b*	REAL, INTENT(IN) (vspow の場合) DOUBLE PRECISION, INTENT(IN) (vdpow の場合) 配列、入力ベクトル a と b を指定する。

C:

n	int。計算する成分の数を指定する。
a、*b*	const float* (vsPow の場合) const double* (vdPow の場合) 入力ベクトル a と b を格納する配列へのポインタ。

表 7-4 Pow 関数における各精度でのオーバーフローしきい値

データ・タイプ	入力パラメータに対するしきい値制限
単精度	abs(*a*[i]) < (FLT_MAX) 1/*b*[i]
倍精度	abs(*a*[i]) < (DBL_MAX) 1/*b*[i]

出力パラメータ

Fortran:

y	REAL (vspow の場合) DOUBLE PRECISION (vdpow の場合) 配列、出力ベクトル y を指定する。

C:

y	float* (vsPow の場合) double* (vdPow の場合) 出力ベクトル y を格納する配列へのポインタ。

説明

関数 Pow には、パラメータ *a* と *b* の入力範囲について、一定の制限がある。具体的には、*a*[i] が正の場合、*b*[i] には任意の値を指定できる。*a*[i] が負またはゼロの場合、*b*[i] の値は整数 (正または負) でなければならない。

Powx

各ベクトル成分を定数で累乗する。

Fortran:

```
call vspowx( n, a, b, y )
call vdpowx( n, a, b, y )
```

C:

```
vsPowx( n, a, b, y );
vdPowx( n, a, b, y );
```

入力パラメータ

Fortran:

n	INTEGER, INTENT(IN)。計算する成分の数を指定する。
a, *b*	REAL, INTENT(IN) (vspowx の場合)

DOUBLE PRECISION, INTENT(IN) (vdpowx の場合)
配列 a は入力ベクトルを指定する。
スカラー値 b は累乗の定数。

C:

n　int。計算する成分の数を指定する。

a　const float*　(vsPowx の場合)
const double*　(vdPowx の場合)
入力ベクトル a を格納する配列へのポインタ。

b　const float for (vsPowx の場合)
const double (vdPowx の場合)
累乗の定数 b

表 7-5　Powx 関数における各精度でのオーバーフローしきい値

データ・タイプ	入力パラメータに対するしきい値制限
単精度	abs(a[i]) < (FLT_MAX) $^{1/b}$
倍精度	abs(a[i]) < (DBL_MAX) $^{1/b}$

出力パラメータ

Fortran:

y　REAL (vspowx の場合)
DOUBLE PRECISION (vdpowx の場合)
配列、出力ベクトル y を指定する。

C:

y　float*　(vspowx の場合)
double*　(vdpowx の場合)
出力ベクトル y を格納する配列へのポインタ。

説明

関数 Powx には、パラメータ a および b の入力範囲について、一定の制限がある。具体的には、a[i] が正の場合、b には任意の値を指定できる。a[i] が負またはゼロの場合、b の値は整数 (正または負) でなければならない。

Exp

ベクトル成分の指数を計算する。

Fortran:

```
call vsexp( n, a, y )
call vdexp( n, a, y )
```

C:

```
vsExp( n, a, y );
vdExp( n, a, y );
```

入力パラメータ

Fortran:

n	INTEGER, INTENT(IN)。計算する成分の数を指定する。
a	REAL, INTENT(IN) (vsexp の場合) DOUBLE PRECISION, INTENT(IN) (vdexp の場合) 配列、入力ベクトル a を指定する。

C:

n	int。計算する成分の数を指定する。
a	const float* (vsExp の場合) const double* (vdExp の場合) 入力ベクトル a を格納する配列へのポインタ。

表 7-6　Exp 関数における各精度でのオーバーフローしきい値

データ・タイプ	入力パラメータに対するしきい値制限
単精度	a[i] < Ln(FLT_MAX)
倍精度	a[i] < Ln(DBL_MAX)

出力パラメータ

Fortran:

y　　REAL (vsexp の場合)
DOUBLE PRECISION (vdexp の場合)
配列、出力ベクトル y を指定する。

C:

y　　float*　(vsExp の場合)
double*　(vdExp の場合)
出力ベクトル y を格納する配列へのポインタ。

Ln

ベクトル成分の自然対数を計算する。

Fortran:

```
call vsln( n, a, y )
call vdln( n, a, y )
```

C:

```
vsLn( n, a, y );
vdLn( n, a, y );
```

入力パラメータ

Fortran:

n　　INTEGER, INTENT(IN)。計算する成分の数を指定する。

a REAL, INTENT(IN) (vsln の場合)
DOUBLE PRECISION, INTENT(IN) (vdln の場合)
配列、入力ベクトル a を指定する。

C:

n int。計算する成分の数を指定する。

a const float* (vsLn の場合)
const double* (vdLn の場合)
入力ベクトル a を格納する配列へのポインタ。

出力パラメータ

Fortran:

y REAL (vsln の場合)
DOUBLE PRECISION (vdln の場合)
配列、出力ベクトル y を指定する。

C:

y float* (vsLn の場合)
double* (vdLn の場合)
出力ベクトル y を格納する配列へのポインタ。

Log10

ベクトル成分の 10 進対数を計算する。

Fortran:

```
call vslog10( n, a, y )
call vdlog10( n, a, y )
```

C:

```
vsLog10( n, a, y );
vdLog10( n, a, y );
```

入力パラメータ

Fortran:

n INTEGER, INTENT(IN)。計算する成分の数を指定する。

a REAL, INTENT(IN) (vslog10 の場合)
DOUBLE PRECISION, INTENT(IN) (vdlog10 の場合)
配列、入力ベクトル a を指定する。

C:

n int。計算する成分の数を指定する。

a const float* (vsLog10 の場合)
const double* (vdLog10 の場合)
入力ベクトル a を格納する配列へのポインタ。

出力パラメータ

Fortran:

y REAL (vslog10 の場合)
DOUBLE PRECISION (vdlog10 の場合)
配列、出力ベクトル y を指定する。

C:

y float* (vsLog10 の場合)
double* (vdLog10 の場合)
出力ベクトル y を格納する配列へのポインタ。

Cos

ベクトル成分の余弦を計算する。

Fortran:

```
call vscos( n, a, y )
call vdcos( n, a, y )
```

C:

```
vsCos( n, a, y );
vdCos( n, a, y );
```

入力パラメータ

Fortran:

n INTEGER, INTENT(IN)。計算する成分の数を指定する。

a REAL, INTENT(IN) (vscos の場合)
DOUBLE PRECISION, INTENT(IN) (vdcos の場合)
配列、入力ベクトル a を指定する。

C:

n int。計算する成分の数を指定する。

a const float* (vsCos の場合)
const double* (vdCos の場合)
入力ベクトル a を格納する配列へのポインタ。

出力パラメータ

Fortran:

y REAL (vscos の場合)
DOUBLE PRECISION (vdcos の場合)
配列、出力ベクトル y を指定する。

C:

y float* (vsCos の場合)
double* (vdCos の場合)
出力ベクトル y を格納する配列へのポインタ。

Sin

ベクトル成分の正弦を計算する。

Fortran:

```
call vssin( n, a, y )
call vdsin( n, a, y )
```

C:

```
vsSin( n, a, y );
vdSin( n, a, y );
```

入力パラメータ

Fortran:

n INTEGER, INTENT(IN)。計算する成分の数を指定する。

a REAL, INTENT(IN) (vssin の場合)
DOUBLE PRECISION, INTENT(IN) (vdsin の場合)
配列、入力ベクトル a を指定する。

C:

n int。計算する成分の数を指定する。

a const float* (vsSin の場合)
const double* (vdSin の場合)
入力ベクトル a を格納する配列へのポインタ。

出力パラメータ

Fortran:

y REAL (vssin の場合)
DOUBLE PRECISION (vdsin の場合)
配列、出力ベクトル y を指定する。

C:

y float* (vsSin の場合)
double* (vdSin の場合)
出力ベクトル y を格納する配列へのポインタ。

SinCos

ベクトル成分の正弦および余弦を
計算する。

Fortran:

```
call vssincos( n, a, y, z )
call vdsincos( n, a, y, z )
```

C:

```
vsSinCos( n, a, y, z );
vdSinCos( n, a, y, z );
```

入力パラメータ

Fortran:

n INTEGER, INTENT(IN)。計算する成分の数を指定する。

a REAL, INTENT(IN) (vssincos の場合)
DOUBLE PRECISION, INTENT(IN) (vdsincos の場合)
配列、入力ベクトル a を指定する。

C:

n int。計算する成分の数を指定する。

a const float* (vsSinCos の場合)
const double* (vdSinCos の場合)
入力ベクトル a を格納する配列へのポインタ。

出力パラメータ

Fortran:

y、*z* REAL (vssincos の場合)
DOUBLE PRECISION (vdsincos の場合)

配列、出力ベクトル y (正弦値の場合) と出力ベクトル z (余弦値の場合) を指定する。

C:

y、*z* float* (vsSinCos の場合)
double* (vdSinCos の場合)
出力ベクトル y (正弦値の場合) と出力ベクトル z (余弦値の場合) を格納する配列へのポインタ。

Tan

ベクトル成分の正接を計算する。

Fortran:

```
call vstan( n, a, y )
call vdtan( n, a, y )
```

C:

```
vsTan( n, a, y );
vdTan( n, a, y );
```

入力パラメータ

Fortran:

n INTEGER, INTENT(IN)。計算する成分の数を指定する。

a REAL, INTENT(IN) (vstan の場合)
DOUBLE PRECISION, INTENT(IN) (vdtan の場合)
配列、入力ベクトル a を指定する。

C:

n int。計算する成分の数を指定する。

a const float* (vsTan の場合)
const double* (vdTan の場合)
入力ベクトル a を格納する配列へのポインタ。

出力パラメータ

Fortran:

y REAL (vstan の場合)
DOUBLE PRECISION (vdtan の場合)
配列、出力ベクトル y を指定する。

C:

y float* (vsTan の場合)
double* (vdTan の場合)
出力ベクトル y を格納する配列へのポインタ。

Acos

ベクトル成分の逆余弦を計算する。

Fortran:

```
call vsacos( n, a, y )
call vdacos( n, a, y )
```

C:

```
vsAcos( n, a, y );
vdAcos( n, a, y );
```

入力パラメータ

Fortran:

n INTEGER, INTENT(IN)。計算する成分の数を指定する。

a REAL, INTENT(IN) (vsacos の場合)
DOUBLE PRECISION, INTENT(IN) (vdacos の場合)
配列、入力ベクトル a を指定する。

C:

n int。計算する成分の数を指定する。

a const float* (vsAcos の場合)
const double* (vdAcos の場合)
入力ベクトル a を格納する配列へのポインタ。

出力パラメータ

Fortran:

y REAL (vsacos の場合)
DOUBLE PRECISION (vdacos の場合)
配列、出力ベクトル y を指定する。

C:

y float* (vsAcos の場合)
double* (vdAcos の場合)
出力ベクトル y を格納する配列へのポインタ。

Asin

ベクトル成分の逆正弦を計算する。

Fortran:

```
call vsasin( n, a, y )
call vdasin( n, a, y )
```

C:

```
vsAsin( n, a, y );
vdAsin( n, a, y );
```

入力パラメータ

Fortran:

n INTEGER, INTENT(IN)。計算する成分の数を指定する。

a REAL, INTENT(IN) (vsasin の場合)
DOUBLE PRECISION, INTENT(IN) (vdasin の場合)
配列、入力ベクトル a を指定する。

C:

n int。計算する成分の数を指定する。

a const float* (vsAsin の場合)
const double* (vdAsin の場合)
入力ベクトル a を格納する配列へのポインタ。

出力パラメータ

Fortran:

y REAL (vsasin の場合)
DOUBLE PRECISION (vdasin の場合)
配列、出力ベクトル y を指定する。

C:

y float* (vsAsin の場合)
double* (vdAsin の場合)
出力ベクトル y を格納する配列へのポインタ。

Atan

ベクトル成分の逆正接を計算する。

Fortran:

```
call vsatan( n, a, y )
call vdatan( n, a, y )
```

C:

```
vsAtan( n, a, y );
vdAtan( n, a, y );
```

入力パラメータ

Fortran:

n INTEGER, INTENT(IN)。計算する成分の数を指定する。

a REAL, INTENT(IN) (vsatan の場合)
DOUBLE PRECISION, INTENT(IN) (vdatan の場合)
配列、入力ベクトル a を指定する。

C:

n int。計算する成分の数を指定する。

a const float* (vsAtan の場合)
const double* (vdAtan の場合)
入力ベクトル a を格納する配列へのポインタ。

出力パラメータ

Fortran:

y REAL (vsatan の場合)
DOUBLE PRECISION (vdatan の場合)
配列、出力ベクトル y を指定する。

C:

y	float* (vsAtan の場合) double* (vdAtan の場合) 出力ベクトル y を格納する配列へのポインタ。

Atan2

2 つのベクトルの成分の四分円逆正接を計算する。

Fortran:

```
call vsatan2( n, a, b, y )
call vdatan2( n, a, b, y )
```

C:

```
vsAtan2( n, a, b, y );
vdAtan2( n, a, b, y );
```

入力パラメータ

Fortran:

n	INTEGER, INTENT(IN)。計算する成分の数を指定する。
a、*b*	REAL, INTENT(IN) (vsatan2 の場合) DOUBLE PRECISION, INTENT(IN) (vdatan2 の場合) 配列、入力ベクトル a と b を指定する。

C:

n	int。計算する成分の数を指定する。
a、*b*	const float* (vsAtan2 の場合) const double* (vdAtan2 の場合) 入力ベクトル a と b を格納する配列へのポインタ。

出力パラメータ

Fortran:

y	REAL (vsatan2 の場合) DOUBLE PRECISION (vdatan2 の場合) 配列、出力ベクトル y を指定する。

C:

y	float* (vsAtan2 の場合) double* (vdAtan2 の場合) 出力ベクトル y を格納する配列へのポインタ。

出力ベクトル y の成分は、a[i] / b[i] の四分円逆正接として計算される。

Cosh

ベクトル成分の双曲余弦を計算する。

Fortran:

```
call vscosh( n, a, y )
call vdcosh( n, a, y )
```

C:

```
vsCosh( n, a, y );
vdCosh( n, a, y );
```

入力パラメータ

Fortran:

n	INTEGER, INTENT(IN)。計算する成分の数を指定する。

a REAL, INTENT(IN) (vscosh の場合)
DOUBLE PRECISION, INTENT(IN) (vdcosh の場合)
配列、入力ベクトル a を指定する。

C:

n int。計算する成分の数を指定する。

a const float* (vsCosh の場合)
const double* (vdCosh の場合)
入力ベクトル a を格納する配列へのポインタ。

表 7-7 Cosh 関数における各精度でのオーバーフローしきい値

データ・タイプ	入力パラメータに対するしきい値制限
単精度	-Ln(FLT_MAX)-Ln2 < *a*[i] < Ln(FLT_MAX)+Ln2
倍精度	-Ln(DBL_MAX)-Ln2 < *a*[i] < Ln(DBL_MAX)+Ln2

出力パラメータ

Fortran:

y REAL (vscosh の場合)
DOUBLE PRECISION (vdcosh の場合)
配列、出力ベクトル y を指定する。

C:

y float* (vsCosh の場合)
double* (vdCosh の場合)
出力ベクトル y を格納する配列へのポインタ。

Sinh

ベクトル成分の双曲正弦を計算する。

Fortran:

```
call vssinh( n, a, y )
call vdsinh( n, a, y )
```

C:

```
vsSinh( n, a, y );
vdSinh( n, a, y );
```

入力パラメータ

Fortran:

n	INTEGER, INTENT(IN)。計算する成分の数を指定する。
a	REAL, INTENT(IN) (vssinh の場合) DOUBLE PRECISION, INTENT(IN) (vdsinh の場合) 配列、入力ベクトル a を指定する。

C:

n	int。計算する成分の数を指定する。
a	const float* (vsSinh の場合) const double* (vdSinh の場合) 入力ベクトル a を格納する配列へのポインタ。

表 7-8　Sinh 関数における各精度でのオーバーフローしきい値

データ・タイプ	入力パラメータに対するしきい値制限
単精度	-Ln(FLT_MAX)-Ln2 < a[i] < Ln(FLT_MAX)+Ln2
倍精度	-Ln(DBL_MAX)-Ln2 < a[i] < Ln(DBL_MAX)+Ln2

出力パラメータ

Fortran:

y　　REAL (vssinh の場合)
DOUBLE PRECISION (vdsinh の場合)
配列、出力ベクトル y を指定する。

C:

y　　float*　(vsSinh の場合)
double*　(vdSinh の場合)
出力ベクトル y を格納する配列へのポインタ。

Tanh

ベクトル成分の双曲正接を計算する。

Fortran:

```
call vstanh( n, a, y )
call vdtanh( n, a, y )
```

C:

```
vsTanh( n, a, y );
vdTanh( n, a, y );
```

入力パラメータ

Fortran:

n　　INTEGER, INTENT(IN)。計算する成分の数を指定する。

a REAL, INTENT(IN) (vstanh の場合)
DOUBLE PRECISION, INTENT(IN) (vdtanh の場合)
配列、入力ベクトル a を指定する。

C:

n int。計算する成分の数を指定する。

a const float* (vsTanh の場合)
const double* (vdTanh の場合)
入力ベクトル a を格納する配列へのポインタ。

出力パラメータ

Fortran:

y REAL (vstanh の場合)
DOUBLE PRECISION (vdtanh の場合)
配列、出力ベクトル y を指定する。

C:

y float* (vsTanh の場合)
double* (vdTanh の場合)
出力ベクトル y を格納する配列へのポインタ。

Acosh

ベクトル成分の逆双曲余弦 (非負) を
計算する。

Fortran:

```
call vsacosh( n, a, y )
call vdacosh( n, a, y )
```

C:

```
vsAcosh( n, a, y );
vdAcosh( n, a, y );
```

入力パラメータ

Fortran:

n INTEGER, INTENT(IN)。計算する成分の数を指定する。

a REAL, INTENT(IN) (vsacosh の場合)
DOUBLE PRECISION, INTENT(IN) (vdacosh の場合)
配列、入力ベクトル a を指定する。

C:

n int。計算する成分の数を指定する。

a const float* (vsAcosh の場合)
const double* (vdAcosh の場合)
入力ベクトル a を格納する配列へのポインタ。

出力パラメータ

Fortran:

y REAL (vsacosh の場合)
DOUBLE PRECISION (vdacosh の場合)
配列、出力ベクトル y を指定する。

C:

y float* (vsAcosh の場合)
double* (vdAcosh の場合)
出力ベクトル y を格納する配列へのポインタ。

Asinh

ベクトル成分の逆双曲正弦を計算する。

Fortran:

```
call vsasinh( n, a, y )
call vdasinh( n, a, y )
```

C:

```
vsAsinh( n, a, y );
vdAsinh( n, a, y );
```

入力パラメータ

Fortran:

n INTEGER, INTENT(IN)。計算する成分の数を指定する。

a REAL, INTENT(IN) (vsasinh の場合)
DOUBLE PRECISION, INTENT(IN) (vdasinh の場合)
配列、入力ベクトル a を指定する。

C:

n int。計算する成分の数を指定する。

a const float* (vsAsinh の場合)
const double* (vdAsinh の場合)
入力ベクトル a を格納する配列へのポインタ。

出力パラメータ

Fortran:

y REAL (vsasinh の場合)
DOUBLE PRECISION (vdasinh の場合)
配列、出力ベクトル y を指定する。

C:

y float* (vsAsinh の場合)
double* (vdAsinh の場合)
出力ベクトル y を格納する配列へのポインタ。

Atanh

ベクトル成分の逆双曲正接を計算する。

Fortran:

```
call vsatanh( n, a, y )
call vdatanh( n, a, y )
```

C:

```
vsAtanh( n, a, y );
vdAtanh( n, a, y );
```

入力パラメータ

Fortran:

n INTEGER, INTENT(IN)。計算する成分の数を指定する。

a REAL, INTENT(IN) (vsatanh の場合)
DOUBLE PRECISION, INTENT(IN) (vdatanh の場合)
配列、入力ベクトル a を指定する。

C:

n int。計算する成分の数を指定する。

a const float* (vsAtanh の場合)
const double* (vdAtanh の場合)
入力ベクトル a を格納する配列へのポインタ。

出力パラメータ

Fortran:

y REAL (vsatanh の場合)
DOUBLE PRECISION (vdatanh の場合)
配列、出力ベクトル y を指定する。

C:

y float* (vsAtanh の場合)
double* (vdAtanh の場合)
出力ベクトル y を格納する配列へのポインタ。

Erf

ベクトル成分の誤差関数を計算する。

Fortran:

```
call vserf( n, a, y )
call vderf( n, a, y )
```

C:

```
vsErf( n, a, y );
vdErf( n, a, y );
```

入力パラメータ

Fortran:

n	INTEGER, INTENT(IN)。計算する成分の数を指定する。
a	REAL, INTENT(IN) (vserf の場合) DOUBLE PRECISION, INTENT(IN) (vderf の場合) 配列、入力ベクトル a を指定する。

C:

n	int。計算する成分の数を指定する。
a	const float* (vsErf の場合) const double* (vdErf の場合) 入力ベクトル a を格納する配列へのポインタ。

出力パラメータ

Fortran:

y	REAL (vserf の場合) DOUBLE PRECISION (vderf の場合) 配列、出力ベクトル y を指定する。

C:

y	float* (vsErf の場合) double* (vdErf の場合) 出力ベクトル y を格納する配列へのポインタ。

説明

関数 Erf は、入力ベクトル *a* の成分に対して誤差関数を計算し、結果を出力ベクトル *y* として書き出す。

誤差関数は次のように定義される。

$$\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}dt$$

Erfc

ベクトル成分の相補誤差関数を計算する。

Fortran:

```
call vserfc( n, a, y )
call vderfc( n, a, y )
```

C:

```
vsErfc( n, a, y );
vdErfc( n, a, y );
```

入力パラメータ

Fortran:

n INTEGER, INTENT(IN)。計算する成分の数を指定する。

a REAL, INTENT(IN) (vserfc の場合)
DOUBLE PRECISION, INTENT(IN) (vderfc の場合)
配列、入力ベクトル a を指定する。

C:

n int。計算する成分の数を指定する。

a const float* (vsErfc の場合)
const double* (vdErfc の場合)
入力ベクトル a を格納する配列へのポインタ。

出力パラメータ

Fortran:

y REAL (vserfc の場合)
DOUBLE PRECISION (vderfc の場合)
配列、出力ベクトル y を指定する。

C:

y float* (vsErfc の場合)
double* (vdErfc の場合)
出力ベクトル y を格納する配列へのポインタ。

説明

関数 `Erfc` は、入力ベクトル *a* の成分に対して相補誤差関数を計算し、結果を出力ベクトル *y* として書き出す。

相補誤差関数は次のように定義される。

$$\mathrm{erfc}(x) = 1 - \mathrm{erf}(x)$$

または書き換えると、

$$\mathrm{erfc}(x) = \frac{2}{\sqrt{\pi}}\int_x^{\infty} e^{-t^2}dt$$

VML Pack/Unpack 関数

この項では、単位増分を使用するベクトルから正増分インデックス、ベクトル・インデックス、マスク・インデックスを使用するベクトルへの変換、およびその逆の変換を実行する VML 関数について説明する (ベクトル・インデックス方式の詳細は、付録 A を参照のこと)。

表 7-9 は、使用可能な VML Pack/Unpack 関数とそれに関連するデータ・タイプとインデックス方式をまとめたものである。

表 7-9　VML Pack/Unpack 関数

関数の短縮名	データ・タイプ	インデックス方式	説明
Pack	s, d	I,V,M	各種の方式でインデックス化された配列の成分を収集する。
Unpack	s, d	I,V,M	各種のインデックスを使用する配列にベクトル成分を分散する。

Pack

指定されたインデックスを使用する配列の成分を、単位増分を使用するベクトルにコピーする。

Fortran:

```
call vspacki( n, a, inca, y  )
call vspackv( n, a, ia,   y  )
call vspackm( n, a, ma,   y  )
call vdpacki( n, a, inca, y )
call vdpackv( n, a, ia,   y )
call vdpackm( n, a, ma,   y )
```

C:

```
vsPackI( n, a, inca, y );
vsPackV( n, a, ia,   y );
vsPackM( n, a, ma,   y );
vdPackI( n, a, inca, y );
vdPackV( n, a, ia,   y );
vdPackM( n, a, ma,   y );
```

入力パラメータ

Fortran:

n INTEGER, INTENT(IN)。計算する成分の数を指定する。

a REAL, INTENT(IN) (vspacki、vspackv、vspackm の場合)
DOUBLE PRECISION, INTENT(IN) (vdpacki、vdpackv、vdpackm の場合)
配列 DIMENSION は、
vspacki の場合 (1 + (*n*-1)**inca*) 以上、
vspackv の場合 max(*n*, max(*ia*[j])) (j=0, …, *n*-1) 以上、
vspackm の場合 *n* 以上。入力ベクトル a を指定する。

inca INTEGER, INTENT(IN) (vspacki、vdpacki の場合)。
a の成分の増分を指定する。

ia INTEGER, INTENT(IN) (vspackv、vdpackv の場合)。
配列 DIMENSION は *n* 以上。
a の成分のインデックス・ベクトルを指定する。

ma INTEGER, INTENT(IN) (vspackm、vdpackm の場合)。
配列 DIMENSION は *n* 以上。
a の成分のマスク・ベクトルを指定する。

C:

n int。計算する成分の数を指定する。

a const float* (vsPackI、vsPackV、vsPackM の場合)
const double* (vdPackI、vdPackV、vdPackM の場合)
入力ベクトル a を格納する配列へのポインタを指定する。
配列のサイズは、以下でなければならない。

vsPackI の場合、(1 + (*n*-1)**inca*) 以上、
vsPackV の場合、max(*n*, max(*ia*[j]))
(j=0, …, *n*-1) 以上、
vsPackM の場合、*n* 以上。

inca int (vsPackI、vdPackI の場合)。
a の成分の増分を指定する。

ia const int* (vsPackV、vdPackV の場合)。*a* の成分のインデックス・ベクトルを格納するサイズが *n* 以上の配列へのポインタを指定する。

ma const int* (vsPackM、vdPackM の場合)。*a* の成分のマスク・ベクトルを格納するサイズが *n* 以上の配列へのポインタを指定する。

出力パラメータ

Fortran:

y REAL (vspacki、vspackv、vspackm の場合)
DOUBLE PRECISION (vdpacki、vdpackv、vdpackm の場合)
配列 DIMENSION は *n* 以上。出力ベクトル y を指定する。

C:

y float* (vsPackI、vsPackV、vsPackM の場合)
double* (vdPackI、vdPackV、vdPackM の場合)
出力ベクトル y を格納するサイズが *n* 以上の配列へのポインタを指定する。

Unpack

単位増分を使用するベクトルの成分を、
指定されたインデックスを使用する配列に
コピーする。

Fortran:

```
call vsunpacki( n, a, y, incy )
call vsunpackv( n, a, y, iy )
call vsunpackm( n, a, y, my )
```

```
call vdunpacki( n, a, y, incy )
call vdunpackv( n, a, y, iy )
call vdunpackm( n, a, y, my )
```

C:

```
vsUnpackI( n, a, y, incy );
vsUnpackV( n, a, y, iy );
vsUnpackM( n, a, y, my );
vdUnpackI( n, a, y, incy );
vdUnpackV( n, a, y, iy );
vdUnpackM( n, a, y, my );
```

入力パラメータ

Fortran:

n INTEGER, INTENT(IN)。計算する成分の数を指定する。

a REAL, INTENT(IN) (vsunpacki、vsunpackv、vsunpackm の場合)
DOUBLE PRECISION, INTENT(IN) (vdunpacki、vdunpackv、vdunpackm の場合)。
配列 DIMENSION は *n* 以上。入力ベクトル a を指定する。

incy INTEGER, INTENT(IN) (vsunpacki、vdunpacki の場合)。*y* の成分の増分を指定する。

iy INTEGER, INTENT(IN) (vsunpackv、vdunpackv の場合)。配列 DIMENSION は *n* 以上。*y* の成分のインデックス・ベクトルを指定する。

my INTEGER, INTENT(IN) (vsunpackm、vdunpackm の場合)。べは *n* 以上。*y* の成分のマスク・ベクトルを指定する。

C:

n int。計算する成分の数を指定する。

a const float* (vsUnpackI、vsUnpackV、vsUnpackM の場合)
const double* (vdUnpackI、vdUnpackV、vdUnpackM の場合)
入力ベクトル a を格納するサイズが *n* 以上の配列へのポインタを指定する。

incy int (vsUnpackI、vdUnpackI の場合)。
y の成分の増分を指定する。

iy const int* (vsUnpackV、vdUnpackV の場合)。*a* の成分のインデックス・ベクトルを格納するサイズが *n* 以上の配列へのポインタを指定する。

my const int* (vsUnpackM、vdUnpackM の場合)。*a* の成分のマスク・ベクトルを格納するサイズが *n* 以上の配列へのポインタを指定する。

出力パラメータ

Fortran:

y REAL (vsunpacki、vsunpackv、vsunpackm の場合)
DOUBLE PRECISION (vdunpacki、vdunpackv、vdunpackm の場合)。
べは、
vsunpacki の場合、(1 + (*n*-1)*incy) 以上、
vsunpackv の場合、max(*n*, max(*iy*[j])) (j=0, …, *n*-1) 以上、
vsunpackm の場合、*n* 以上。
出力ベクトル y を指定する。

C:

y float* (vsUnpackI、vsUnpackV、vsUnpackM の場合)
double* (vdUnpackI、vdUnpackV、vdUnpackM の場合)
出力ベクトル y を格納する配列へのポインタを指定する。
配列のサイズは、以下でなければならない。
vsUnPackI の場合、(1 + (*n*-1)*incy) 以上、
vsUnPackV の場合、max(*n*, max(*ia*[j])) (j=0, …, *n*-1) 以上、
vsUnPackM の場合、*n* 以上。

VML サービス関数

この項では、精密モードの設定 / 取得、エラー・コードの設定 / 取得を実行する VML 関数について説明する。これらの関数はすべて、Fortran インターフェイスと C インターフェイスの両方で使用できる。
表 7-10 は、使用可能な VML サービス関数とそれについての簡単な説明をまとめたものである。

表 7-10　VML サービス関数

関数の短縮名	説明
SetMode	VML モードを設定する
GetMode	VML モードを取得する
SetErrStatus	VML エラー・ステータスを設定する
GetErrStatus	VML エラー・ステータスを取得する
ClearErrStatus	VML エラー・ステータスをクリアする
SetErrorCallBack	追加のエラー・ハンドラ・コールバック関数を設定する
GetErrorCallBack	追加のエラー・ハンドラ・コールバック関数を取得する
ClearErrorCallBack	追加のエラー・ハンドラ・コールバック関数を削除する

SetMode

mode パラメータに従って VML 関数の新しいモードを設定し、前の VML モードを *oldmode* に保存する。

Fortran:

```
oldmode = vmlsetmode( mode )
```

C:

```
oldmode = vmlSetMode( mode );
```

入力パラメータ

Fortran:

mode INTEGER, INTENT(IN)。設定する VML モードを指定する。

C:

mode int。設定する VML モードを指定する。

出力パラメータ

Fortran:

oldmode INTEGER。前の VML モードを指定する。

C:

oldmode int。前の VML モードを指定する。

説明

mode パラメータは、精度、FPU、エラー処理オプションを制御する。表 7-11 は、*mode* パラメータの値をまとめたものである。*mode* パラメータに指定できるその他の値はすべて、これらの値から得られる。その方法は、ビット単位 OR (|) という演算を使用して、精度用の値、FPU 用の値、エラー制御オプション用の値を 1 つずつ組み合わせる。*mode* パラメータのデフォルト値は、VML_HA | VML_ERRMODE_DEFAULT である。したがって、デフォルトでは、現在の FPU 制御ワード (FPU 精度および丸め方式) が使用される。

異なる FPU 精度または丸め方式が必要な場合、VML 数学関数は、これらのオプションを自動的に変更し、前の値を復元する。*mode* パラメータを使用すると、同じ精度の設定で機能する各 VML 数学関数内の内部 FPU モードの切り替えを最小限に抑えられる。そのためには、*mode* パラメータに VML_FLOAT_CONSISTENT (単精度関数の場合)、または VML_DOUBLE_CONSISTENT (倍精度関数の場合) を設定する。それぞれの関数グループで、*mode* パラメータにこれらの値を設定するのは最適な選択となる。なぜなら、これらの値は、ほとんどの VML 数学関数で必須だからである。前の FPU モードを復元する必要がある場合は、実行の完了後、*mode* に VML_RESTORE を設定する。

表 7-11 *mode* パラメータの値

mode の値	説明
精度の制御	
VML_HA	VML 関数の高精度モードを使用する
VML_LA	VML 関数の低精度モードを使用する
追加 FPU モードの制御	
VML_FLOAT_CONSISTENT	単精度関数に最適な FPU モード (制御ワード) を設定し、前の FPU モードを保存する
VML_DOUBLE_CONSISTENT	倍精度関数に最適な FPU モード (制御ワード) を設定し、前の FPU モードを保存する
VML_RESTORE	前に保存した FPU モードを復元する
エラー・モードの制御	
VML_ERRMODE_IGNORE	計算エラーに対するアクションを設定しない
VML_ERRMODE_ERRNO	エラー時に errno 変数を設定する
VML_ERRMODE_STDERR	エラー時にエラー・テキスト情報を stderr に書き込む
VML_ERRMODE_EXCEPT	エラー時に例外を生成する
VML_ERRMODE_CALLBACK	エラー時に追加のエラー・ハンドラ関数を呼び出す
VML_ERRMODE_DEFAULT	エラー時に errno 変数を設定し、例外を生成して、追加のエラー・ハンドラ関数を呼び出す

例

mode パラメータにさまざまな値を指定して関数 vmlSetMode() を呼び出す例をいくつか以下に示す。

Fortran:

```
oldmode = vmlsetmode( VML_LA )
call vmlsetmode( IOR(VML_LA, IOR(VML_FLOAT_CONSISTENT,
        VML_ERRMODE_IGNORE )))
call vmlsetmode( VML_RESTORE)
```

C:

```
vmlSetMode( VML_LA );
vmlSetMode( VML_LA | VML_FLOAT_CONSISTENT | VML_ERRMODE_IGNORE );
vmlSetMode( VML_RESTORE);
```

GetMode

VML モードを取得する。

Fortran:

```
mod  = vmlgetmode()
```

C:

```
mod  = vmlGetMode( void );
```

出力パラメータ

Fortran:

mod INTEGER。パックド *mode* パラメータを指定する。

C:

mod int。パックド *mode* パラメータを指定する。

説明

関数 vmlGetMode() は、精度、FPU、エラー処理オプションを制御する VML mode パラメータを戻す。mod 変数の値は、表 7-11 で示した値の組み合わせで示される。表 7-12 のそれぞれのマスクを使用すると、これらの値の一部を得られる。例を以下に示す。

Fortran:

```
mod  = vmlgetmode()
accm = IAND(mod, VML_ACCURACY_MASK)
fpum = IAND(mod, VML_FPUMODE_MASK)
errm = IAND(mod, VML_ERRMODE_MASK)
```

C:

```
accm = vmlGetMode(void )& VML_ACCURACY_MASK;
fpum = vmlGetMode(void )& VML_FPUMODE _MASK;
errm = vmlGetMode(void )& VML_ERRMODE _MASK;
```

表 7-12 *mode* パラメータのマスクの値

マスクの値	説明
VML_ACCURACY_MASK	精密 *mode* を選択した場合のマスクを指定する。
VML_FPUMODE_MASK	FPU *mode* を選択した場合のマスクを指定する。
VML_ERRMODE_MASK	エラー *mode* を選択した場合のマスクを指定する。

SetErrStatus

err に従って VML エラー・ステータスを設定し、前の VML エラー・ステータスを *olderr* に保存する。

Fortran:

```
olderr = vmlseterrstatus( err )
```

C:

```
olderr = vmlSetErrStatus( err );
```

入力パラメータ

Fortran:

err INTEGER, INTENT(IN)。設定する VML エラー・ステータスを指定する。

C:

err int。設定する VML エラー・ステータスを指定する。

出力パラメータ

Fortran:

olderr INTEGER。前の VML エラー・ステータスを指定する。

C:

olderr int。前の VML エラー・ステータスを指定する。

表 7-13 は、*err* パラメータに指定可能な値をまとめたものである。

表 7-13 VML エラー・ステータスの値

エラー・ステータス	説明
VML_STATUS_OK	実行が正常に完了した。
VML_STATUS_BADSIZE	配列の次元が正でない。
VML_STATUS_BADMEM	NULL ポインタが渡される。
VML_STATUS_ERRDOM	配列の値の少なくとも 1 つが、定義の範囲外。
VML_STATUS_SING	配列の値の少なくとも 1 つが、特異点を引き起こした。
VML_STATUS_OVERFLOW	計算プロセスでオーバーフローが発生した。
VML_STATUS_UNDERFLOW	計算プロセスでアンダーフローが発生した。

例 :

```
vmlSetErrStatus( VML_STATUS_OK );
vmlSetErrStatus( VML_STATUS_ERRDOM );
vmlSetErrStatus( VML_STATUS_UNDERFLOW );
```

GetErrStatus

VML エラー・ステータスを取得する。

Fortran:

```
err = vmlgeterrstatus( )
```

C:

```
err  = vmlGetErrStatus( void );
```

出力パラメータ

Fortran:

err INTEGER。VML エラー・ステータスを指定する。

C:

err int。VML エラー・ステータスを指定する。

ClearErrStatus

VML エラー・ステータスに VML_STATUS_OK を設定し、前の VML エラー・ステータスを *olderr* に保存する。

Fortran:

```
olderr = vmlclearerrstatus( )
```

C:

```
olderr = vmlClearErrStatus( void );
```

出力パラメータ

Fortran:

olderr INTEGER。前の VML エラー・ステータスを指定する。

C:

olderr int。前の VML エラー・ステータスを指定する。

SetErrorCallBack

追加のエラー・ハンドラ・コールバック関数を設定し、古いコールバック関数を取得する。

Fortran:

```
oldcallback = vmlseterrorcallback( callback )
```

C:

```
oldcallback = vmlSetErrorCallBack( callback );
```

入力パラメータ

Fortran:

callback コールバック関数のアドレス。
コールバック関数は、以下の形式で示される。

```
INTEGER FUNCTION ERRFUNC(par)
 TYPE (ERROR_STRUCTURE) par
 ! ...
 ! user error processing
 ! ...
 ERRFUNC = 0
 ! if ERRFUNC  = 0 - standard VML error handler
 ! is called after the callback
 ! if ERRFUNC != 0 - standard VML error handler
 ! is not called
END
```

渡されるエラー構造は、次のように定義される。

```
TYPE ERROR_STRUCTURE
     SEQUENCE
     INTEGER*4 ICODE
     INTEGER*4 IINDEX
     REAL*8 DBA1
     REAL*8 DBA2
     REAL*8 DBR1
     REAL*8 DBR2
     CHARACTER(64) CFUNCNAME
     INTEGER*4 IFUNCNAMELEN
 END TYPE ERROR_STRUCTURE
```

C:

callback コールバック関数へのポインタ。
コールバック関数は、次の形式で示される。

```
static int __stdcall
MyHandler(DefVmlErrorContext*
pContext)
{
     /* Handler body */
};
```

渡されるエラー構造は、次のように定義される。

```
typedef struct _DefVmlErrorContext
{
    int iCode;            /* Error status
                             value */
    int iIndex;           /* Index for bad array
                             element, or bad
                             array dimension, or
                             bad array pointer */
    double dbA1;           * Error argument 1 */
    double dbA2;          /* Error argument 2 */
    double dbR1;          /* Error result 1 */
    double dbR2;          /* Error result 2 */
    char cFuncName[64];   /* Function name */
    int iFuncNameLen;     /* Length of function
                             name*/
} DefVmlErrorContext;
```

出力パラメータ

Fortran:

oldcallback 前のコールバック関数のアドレス。

C:

oldcallback 前のコールバック関数へのポインタ。

説明

VML_ERRMODE_CALLBACK エラー・モードが設定されている場合、VML 数学関数のエラーが発生すると、そのたびにコールバック関数が呼び出される (表 7-11 を参照)。

デフォルトの空のコールバック関数ではなく、独自のコールバック関数を定義する必要がある場合は、vmlSetErrorCallBack() 関数を使用する。

コールバック関数の入力構造には、検出されたエラーに関して次の情報が含まれる。

- エラーを引き起こした入力値
- この値のロケーション (配列インデックス)
- 計算された結果値
- エラー・コード
- エラーが発生した関数の名前

コールバック関数には、独自のエラー処理を組み込める。例えば、渡された結果値を訂正し、それを戻して計算を再開するような処理を組み込むおも可能である。コールバック関数の後に標準エラー・ハンドラが呼び出されるのは、0 が戻された場合だけである。

GetErrorCallBack

追加のエラー・ハンドラ・コールバック関数を取得する。

Fortran:

```
fun = vmlgeterrorcallback( )
```

C:

```
fun = vmlGetErrorCallBack( void );
```

出力パラメータ

Fortran:

fun コールバック関数のアドレス。

C:

fun コールバック関数へのポインタ。

ClearErrorCallBack

追加のエラー・ハンドラ・コールバック関数を削除し、前のコールバック関数を検索する。

Fortran:

```
oldcallback = vmlclearerrorcallback( )
```

C:

```
oldcallback = vmlClearErrorCallBack( void );
```

出力パラメータ

Fortran:

oldcallback `INTEGER`。前のコールバック関数のアドレス。

C:

oldcallback `int`。前のコールバック関数へのポインタ。

統計的分布のベクトル生成器 8

本章では、ベクトル統計ライブラリ (VSL) として知られる、擬似乱数ベクトルの生成を目的として設計されたインテル ® マス・カーネル・ライブラリの機能について説明する。

ベクトル統計ライブラリ (VSL) 関数 (第 7 章参照) は基本的な連続分布または離散分布に対応した擬似乱数生成サブルーチン群で構成される。VSL サブルーチンは高度に最適化された基本乱数生成器とベクトル数学関数ライブラリ VML を呼び出して最高レベルの性能を実現している。

VSL サブルーチンは 3 種類のカテゴリーに分類できる。

- 一様分布、正規 (ガウス) 分布、二項分布など、さまざまなタイプの統計的分布に対応した擬似乱数生成器。生成器の詳細は「擬似乱数生成器」の節で述べる。
- 乱数ストリームを取り扱う基本サブルーチン。生成、初期化、削除、コピー、基本生成器のインデックス取得がある。サブルーチンの詳細は「サービス・サブルーチン」の節で述べる。
- 基本生成器を登録するサブルーチン、および登録された生成器のプロパティを取得するサブルーチン用登録。詳細は「アドバンスト・サービス・サブルーチン」の節で述べる。

カテゴリーの後半 2 つをサービス・サブルーチンと呼ぶ。

規則

この章で擬似乱数生成器 (RNG) とは、与えられた確率分布関数を使用して、全体集合からランダムなサンプリングとして解釈される数列を生成する、数論上の決定論的アルゴリズムを意味する乱数は決定論的アルゴリズムを用いて生成されるため、それらは真のランダムではなく擬似ランダムとして呼ぶ必要がある。各生成器も同様に擬似乱数生成器と呼ぶ。しかし本章では、文脈上必要でない限り、真の乱数と擬似乱数との間にも、真の乱数生成器と擬似乱数生成器との間にも、特定の相違はないものとする。同様に、**乱数**と**変量**、**統計的分布**と**確率的分布**も、それぞれ区別しない。

数論上のアルゴリズム A と初期条件 I の選択によって乱数の固有シーケンスが決まる。これをランダム・ストリームと呼ぶ。$\langle A, I \rangle$ のペアをランダム・ストリーム・ステートと呼ぶ。VSL では、FORTRAN インターフェイスでは `TYPE (VSL_STREAM_STATE)` 構造体として表される**ストリーム・ディスクリプタ**によって、C インターフェイスでは `VSLStreamStatePtr` ポインタによって、ストリームが特定される。

一様分布以外の各生成器は、離散分布と連続分布の両方とも、基本乱数生成器 (BRNG) と呼ばれる一様分布生成器を元に構築されている。一様分布以外の擬似乱数は、一様分布の擬似乱数にしかるべき変換を行って取得する。一般的な変換手法には、累積分布関数 (CDF) の逆関数、受容 / 棄却法、その併用がある。ある分布タイプには複数の生成方法が実装されている。

擬似乱数生成の VSL サブルーチンは、ストリーム・ディスクリプタと分布パラメータを入力として受け取り、指定された分布に従う擬似乱数ベクトルを結果として書き出す。一部の統計的分布では、入力パラメータ範囲によって効率が異なる複数の生成法を選択できる。すなわち、入力範囲によって生成法を変えると最も効率的な生成器が得られる。サブルーチンで使う生成法を指定するには、入力パラメータでメソッド番号を指定しなければならない。各生成器で利用可能な生成法は「擬似乱数生成器」の節に記載されている。

本章でマルチプロセッサ・システム、処理ノード、またはプロセッサとは、共有メモリまたは分散メモリ、またはその組み合わせを持つ、任意の構成のシステムを指す。とくに処理ノードまたはプロセッサは、独立した並列処理を実行する能力を持った処理ユニットを意味する (これには、物理的なプロセッサ、クラスタ・ノード、論理的な並列プロセスなどがある)。

数学的表記

本文では次の表記を使用する。

N	自然数の集合 $N = \{1, 2, 3 \ldots\}$
Z	整数の集合 $Z = \{\ldots -3, -2, -1, 0, 1, 2, 3 \ldots\}$
R	実数の集合
$\lfloor a \rfloor$	a の切捨て値 (a よりも小さいか等しい最大の整数)
$\oplus$ または **xor**	ビット単位の排他的 OR
C_α^k または $\binom{\alpha}{k}$	二項係数または組み合わせ ($\alpha \in R$, $\alpha \geq 0$; $k \in N \cup \{0\}$)。$C_\alpha^0 = 1$ 。$\alpha \geq k$ では二項係数は次のように定義される。 $C_\alpha^k = \frac{\alpha(\alpha-1)\ldots(\alpha-k+1)}{k!}$ $\alpha < k$ 。では $C_\alpha^k = 0$ となる。
$\Phi(x)$	累積ガウス分布関数 $\Phi(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} exp\left(-\frac{y^2}{2}\right) dy$ 、$-\infty < x < +\infty$ 。 $\Phi(-\infty) = 0$, $\Phi(+\infty) = 1$ に対して。
LCG(a, c, m)	線形合同生成器 $x_{n+1} = (ax_n + c) \bmod m$ 、ここで a は乗数、c は加数、m は生成器の法である。

MCG(*a*, *m*)	乗算合同生成器 $x_{n+1} = (ax_n) \bmod m$ は、線形合同生成器で *c* が 0 の場合である。
GFSR(*p*, *q*)	GFSR 乱数生成器 $x_n = x_{n-p} \oplus x_{n-q}$。

命名規則

FORTRAN 用の VSL 関数名はすべて小文字である。C 用の関数名は小文字と大文字で構成される。

注：関数命名規則は本マニュアルの関数記述に従った。ライブラリ中の実際の関数名は小文字と大文字の使用に関して若干異なる部分がある。サポート対象のコンパイラ要件を満足する。

The names of generator subroutines have the following structure:

v<*type of result*>rng<*distribution*>	(FORTRAN インターフェイスの場合)
v<*type of result*>Rng<*distribution*>	(C インターフェイスの場合)

ここで v は VSL ベクトル関数のプリフィックスである。<*type of result*> フィールドは s、d、i のいずれかで、次のタイプから 1 つを指定する。

s	REAL (FORTRAN インターフェイスの場合) float (C インターフェイスの場合)
d	DOUBLE PRECISION (FORTRAN インターフェイスの場合) double (C インターフェイスの場合)
i	INTEGER (FORTRAN インターフェイスの場合) int (C インターフェイスの場合)

プリフィックス s と d は連続分布にのみ適用され、プリフィックス i は離散型にのみ適用される。プリフィックス rng はサブルーチンが擬似乱数生成器であることを示し、<*distribution*> フィールドは統計的分布のタイプを示す。

サービス・サブルーチン名は次の規則に従う。

```
vsl<name>
```

ここで vsl は VSL サービス関数のプリフィックスである。<name> フィールドには関数の短縮名が入る。サービス・サブルーチンの詳細は「サービス・サブルーチン」の節と「アドバンスト・サービス・サブルーチン」の節に記載されている。

指定された乱数分布を処理する各生成器サブルーチンのプロトタイプは次の構造に適合する。

```
<function name>( method, stream, n, r, [<distribution
parameters>] )
```

- *method* は生成法を指定する数値である。このパラメータの詳細は「擬似乱数生成器」の節に記載されている。
- *stream* はランダム・ストリーム・ディスクリプタを定義する。非ゼロでなければならない。ランダム・ストリームとその使い方は、後述する「ランダム・ストリーム」と「サービス・サブルーチン」に記載されている。
- *n* は生成させる擬似乱数の個数を指定する。*n* がゼロかゼロ以下の場合、乱数は生成されない。また *n* が負の場合はエラー条件がセットされる。
- *r* は生成した乱数の出力先配列を指定する。配列の次元は、少なくとも *n* 個の擬似乱数を格納できるだけの大きさがなくてはならない。

<distribution parameters> フィールドの追加パラメータは生成器サブルーチンごとに異なっており、詳細は「擬似乱数生成器」の節に記載されている。

擬似乱数生成器を呼び出すにはそれぞれの VSL サブルーチンをコールする。たとえば、平均値 *a*、標準偏差 *sigma* を持つ正規 (ガウス) 分布に従った *n* 個の独立かつ擬似乱数としてみなされるベクトル *r* を取得するには次のように記述する。

FORTRAN インターフェイスの場合

```
call vsrnggaussian( method, stream, n, r, a, sigma )
```

C インターフェイスの場合

```
vsRngGaussian( method, stream, n, r, a, sigma )
```

基本擬似乱数生成器

基本乱数生成器 (BRNG) は、一様分布の擬似乱数を求める主要な方法として広く使用されている。

VSL は処理速度と品質が異なる複数の基本生成器を提供する。すなわち、32 ビット乗算合同生成器 *MCG (1132489760,* 2^{31} *- 1)* [L'Ecuyer99]、32 ビット *GFSR (250,103)* [Kirkpatrick81]、複合再帰生成器の組み合わせ *MRG-32k3a* [L'Ecuyer99a]、NAG 数値ライブラリ [NAG] から 59 ビット乗算合同生成器 *MCG (*13^{13}*,* 2^{59}*)* と Wichmann-Hill 生成器 (実際は 273 個の基本生成器の集合)。与えられた処理タスクに対して、ある基本生成器が適しているかどうかを事前に見極めるのは、本質的に困難である。したがって信頼性の高い結果を得るには、実際の計算処理に先立って、基本生成器に対して一連の統計的検定を実施する必要がある。生成器の性能比較解析と検定結果は VSLNotes に記載されている。

ユーザは独自の基本生成器も作成できる。VSL では、「アドバンスト・サービス・サブルーチン」の節に記載のとおり、ユーザ定義の生成器を登録する手段を設けている。

一部の基本生成器では、マルチプロセッサ処理に対応して、複数の独立したランダム・ストリームを生成するリープフロッグ法とブロック分割法の 2 種類が提供される。並列処理向けに設計された生成器の特性は [Coddington94] に詳細が述べられている。

生成器の特性と検定結果の詳細は VSLNotes を参照のこと。

ランダム・ストリーム

次の場合、単一アプリケーションで複数のランダム・ストリームを使用することがある。

1 つは、マルチプロセッサ・システムの処理ノードごとにランダム・データを与えたい場合である。次のような方策が考えられる。

- 各処理ノードで個別の基本生成器を使用し、各ランダム・ストリームを異なる基本生成器で生成する

- 1 つの基本生成器をすべての処理ノードに適用し、リープフロッグ法またはブロック分割法を用いて複数の独立したランダム・ストリームを作成する。
- 上記 2 つの方法を併用し、各ノードでは個別の生成器を使用し、全ノードにストリームを与える場合は単一の生成器を使って独立したストリームを生成する。

もう 1 つの理由は、多くのモンテカルロ・シミュレーションでは一層のランダムさを要求するためである。ランダム・ストリームをモデルの異なる要素に割り当てたり、分散減少法を実行するには、簡単な実証が必要である [Bratley87]。

どちらの場合も、異なるランダム・ストリーム間の相関が最終結果の信頼性に影響を与え得る。

データ・タイプ

ストリーム・ステート。 乱数は、ストリーム・ステートの概念を用いて、ある個数だけ生成される。ストリーム・ステートはストリーム生成サブルーチンの呼び出しによって生成される構造体であり、ストリーム・ステート・ディスクリプタを使用してアクセスする。

FORTRAN

```
TYPE VSL_STREAM_STATE
    INTEGER*4 descriptor1
    INTEGER*4 descriptor2
END TYPE VSL_STREAM_STATE
```

C

```
typedef (void*) VSLStreamStatePtr;
```

ユーザ定義生成器のストリーム・ステート構造体のフォーマットは、「アドバンスト・サービス・サブルーチン」の節を参照のこと。

サービス・サブルーチン

ストリームを取り扱うサブルーチン群であり、ストリームの作成、削除、コピー、基本生成器のインデックス取得がある。

表 8-1 利用可能なサービス・サブルーチンの全リスト

表 8-1 サービス・サブルーチン

サブルーチン	説明
NewStream	ランダム・ストリームを作成し初期化する
NewStreamEx	複数の初期化条件を必要とする生成器に対し、ランダム・ストリームを作成し初期化する
DeleteStream	以前に生成したストリームを削除する
CopyStream	別のストリームにコピーする
CopyStreamState	ランダム・ストリーム・ステートのコピーを作成する
LeapfrogStream	*nstreams* ノードで構成されるクラスタの *k* 番目の処理ノードに与えるストリームを、リープフロッグ法で初期化する
SkipAheadStream	ストリームをブロック分割法で初期化する
GetStreamStateBrng	指定のランダム・ストリームを生成した基本生成器のインデックスを取得する
GetNumRegBrng	登録されている基本生成器の数を取得する

注：上記の表では関数名の vsl プリフィックスを省略した。このプリフィックスは、関数のリファレンスでは関数プロトタイプとコード例とともに常に使用される。

生成器を用いた処理は、基本的な 3 つのステップで構成する。

1. ストリームを作成し初期化する (NewStream、NewStreamEx、CopyStream、CopyStreamState、LeapfrogStream、SkipAheadStream)。
2. 指定された分布を用いて擬似乱数を生成する。「擬似乱数生成器」を参照。
3. ストリームを削除する (DeleteStream)。

ストリームは同時に複数を作成可能で、1 つまたは複数の生成器からストリーム・ステートを用いて擬似乱数データを取得できる。最後に DeleteStream 関数を用いてすべてのストリームを削除しなければならない。

NewStream

ランダム・ストリームを作成し
初期化する。

Fortran:

```
call vslnewstream( stream, brng, seed )
```

C:

```
vslNewStream( stream, brng, seed )
```

説明

この関数は、*brng* 値を持つ基本生成器のストリームを新規に作成し、32 ビットの種 (シード) を使用して初期化を行う。また、この関数は、複数の初期化条件を伴う生成器にも適用可能である。基本生成器ごとのストリーム初期化の詳細は VSLNotes を参照のこと。

入力パラメータ

FORTRAN:

brng	INTEGER, INTENT(IN)。ストリームを初期化する基本生成器のインデックス値。
seed	INTEGER, INTENT(IN)。ストリームの初期条件。

C:

brng	int。ストリームを初期化する基本生成器のインデックス値。
seed	unsigned int。ストリームの初期条件

出力パラメータ

FORTRAN:

stream	TYPE(VSL_STREAM_STATE), INTENT(OUT)。ストリーム・ステート・ディスクリプタ。

C:

stream	VSLStreamStatePtr*。ストリーム・ステート構造体へのポインタ。

NewStreamEx

複数の初期化条件を使用する生成器で
ランダム・ストリームを作成し
初期化する。

Fortran:

```
call vslnewstreamex( stream, brng, n, params )
```

C:

```
vslNewStreamEx( stream, brng, n, params  )
```

説明

この関数は、入力引数が複数の初期化パラメータで構成される基本生成器に複数の初期条件を設定する。このサブルーチンは特別に必要な場合を除いては使用しない。可能なかぎり、32 ビットの単一初期条件のみを扱う vslNewStream を、vslNewStreamEx に相当する関数として使用する。ただし、GFSR の種テーブルの初期化には vslNewStreamEx を使用する。詳細は「VSLNotes」を参照のこと。

入力パラメータ

FORTRAN:

brng	INTEGER, INTENT(IN)。ストリームを初期化する基本生成器のインデックス値。
n	INTEGER, INTENT(IN)。*params* に含まれる初期条件の数。
params	INTEGER, INTENT(IN)。基本生成器 *brng* がストリームの初期化に必要とする初期条件の配列。

C:

brng	int。ストリームを初期化する基本生成器のインデックス値。
n	int。*params* に含まれる初期条件の数。
params	const unsigned int[]。基本生成器 *brng* がストリームの初期化に必要とする初期条件の配列。

出力パラメータ

FORTRAN:

stream	TYPE(VSL_STREAM_STATE), INTENT(OUT)。ストリーム・ステート・ディスクリプタ。

C:

stream	VSLStreamStatePtr*。ストリーム・ステート構造体へのポインタ。

DeleteStream

ランダム・ストリームを削除する。

Fortran:

```
call vsldeletestream( stream )
```

C:

```
vslDeleteStream( stream )
```

説明

この関数は、いずれかの初期化関数で作成されたランダム・ストリームを削除する。

入力 / 出力パラメータ

FORTRAN:

stream	TYPE(VSL_STREAM_STATE), INTENT(INOUT)。削除するストリームのディスクリプタ。非ゼロであること。

C:

stream	VSLStreamStatePtr*。ストリーム・ステート構造体へのポインタ。非ゼロであること。ストリームの削除に成功すると *stream* ポインタには *NULL* がセットされる。

CopyStream

ランダム・ストリームをコピーする。

Fortran:

```
call vslcopystream( newstream, srcstream )
```

C:

```
vslCopyStream( newstream, srcstream )
```

説明

この関数は *srcstream* の完全なコピーを作成し *newstream* のディスクリプタに格納する。

入力パラメータ

FORTRAN:

scrstream `TYPE(VSL_STREAM_STATE), INTENT(IN)`。
Descriptor of the stream to be copied.

C:

srcstream `VSLStreamStatePtr`。コピーするストリーム・ステート構造体へのポインタ。

出力パラメータ

FORTRAN:

newstream `TYPE(VSL_STREAM_STATE), INTENT(OUT)`。
Descriptor of the stream copy.

C:

newstream `VSLStreamStatePtr*`。コピーされたストリーム・ステート構造体へのポインタ。

CopyStreamState

ランダム・ストリーム・ステートのコピーを作成する。

Fortran:

```
call vslcopystreamstate( deststream, srcstream )
```

C:

```
vslCopyStreamState( deststream, srcstream )
```

説明

この関数は *srcstream* から既存の *deststream* ストリームへストリーム・ステートをコピーする。両方のストリームは同一の基本生成器を使って作成されていなければならない。*deststream* ストリームを作成した BRNG インデックスが *srcstream* ストリームを作成した BRNG インデックスと異なっている場合はエラー・メッセージが出力される。

CopyStream 関数は新規のストリームを作成して *srcstream* からストリーム・ステートと他のデータをコピーするが、CopyStreamState 関数は *srcstream* ストリーム・ステート・データのみ作成済みの *deststream* ストリームにコピーする。

入力パラメータ

FORTRAN:

scrstream	TYPE(VSL_STREAM_STATE), INTENT(IN)。コピー対象のステートを持つストリームのディスクリプタ。

C:

srcstream	VSLStreamStatePtr。コピー対象のストリーム・ステートを持つストリーム・ステート構造体へのポインタ。

出力パラメータ

FORTRAN:

deststream	TYPE(VSL_STREAM_STATE), INTENT(IN)。*scrstream* ストリームのステートがコピーされたコピー先ストリームのディスクリプタ。

C:

deststream	VSLStreamStatePtr。ストリーム・ステートがコピーされたストリーム・ステート構造体へのポインタ。

LeapfrogStream

リープフロッグ法を使用して、*nstreams* ノード・クラスタの k 番目の処理ノードに対するストリームを初期化する。

Fortran:

```
call vslleapfrogstream( stream, k, nstreams )
```

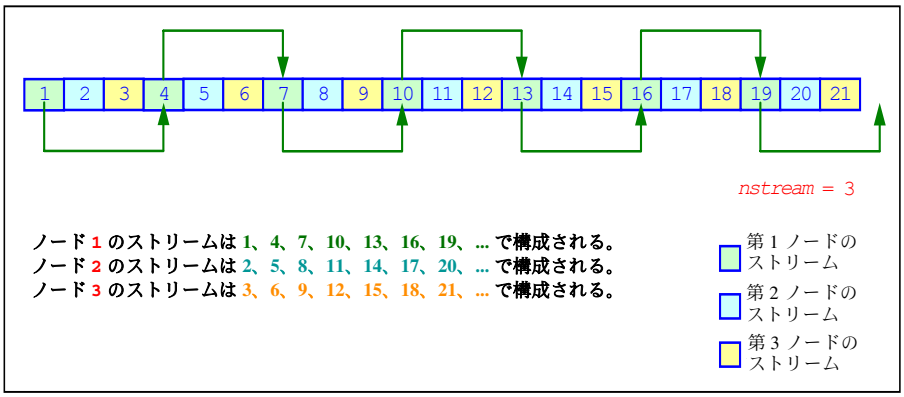
C:

```
vslLeapfrogtream( stream, k, nstreams )
```

説明

この関数はリープフロッグ法 (図 8-1) を使用して、処理ノード k に対し個別のランダム・ストリームを作成する。$k, 0 \le k < nstreams$、*nstreams* は対象の処理ノードの最大値である。

図 8-1 リープフロッグ法



以下に、リープフロッグ法を用いて 3 つの独立したストリームを初期化するコード例を示す。

例 8-1 リープフロッグ法の FORTRAN コード

```
…
type(VSL_STREAM_STATE)stream1
type(VSL_STREAM_STATE)stream2
type(VSL_STREAM_STATE)stream3

! Creating 3 identical streams
call vslnewstream(stream1, VSL_BRNG_MCG31, 174)
call vslcopystream(stream2, stream1)
call vslcopystream(stream3, stream1)

! Leapfrogging the streams
call vslleapfrogstream(stream1, 0, 3)
call vslleapfrogstream(stream2, 1, 3)
call vslleapfrogstream(stream3, 2, 3)

! Generating random numbers
…
! Deleting the streams
call vsldeletestream(stream1)
call vsldeletestream(stream2)
call vsldeletestream(stream3)
…
```

例 8-2 リープフロッグ法の C コード

```
…
VSLStreamStatePtr stream1;
VSLStreamStatePtr stream2;
VSLStreamStatePtr stream3;

/* Creating 3 identical streams */
vslNewStream(&stream1, VSL_BRNG_MCG31, 174);
vslCopyStream(&stream2, stream1);
vslCopyStream(&stream3, stream1);

/* Leapfrogging the streams */
vslLeapfrogStream(stream1, 0, 3);
vslLeapfrogStream(stream2, 1, 3);
vslLeapfrogStream(stream3, 2, 3);

/* Generating random numbers */
…
/* Deleting the streams */
vslDeleteStream(&stream1);
vslDeleteStream(&stream2);
vslDeleteStream(&stream3);
…
```

入力パラメータ

FORTRAN:

stream `TYPE(VSL_STREAM_STATE), INTENT(IN)`。リープフロッグ法を適用するストリームのディスクリプタ。

k `INTEGER, INTENT(IN)`。処理ノードのインデックス、またはストリーム番号。

nstreams `INTEGER, INTENT(IN)`。処理ノードの最大値、または独立したストリーム数。

C:

stream	VSLStreamStatePtr。リープフロッグ法を適用するストリーム・ステート構造体へのポインタ。
k	int。処理ノードのインデックス、またはストリーム番号。
nstreams	int。処理ノードの最大値、または独立したストリーム数。

SkipAheadStream

ブロック分割法を用いてストリームを初期化する。

Fortran:

```
call vslskipaheadstream( stream, nskip )
```

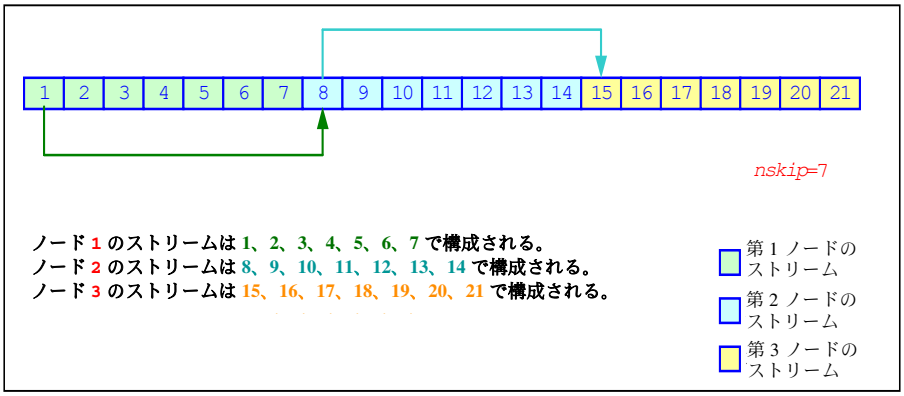
C:

```
vslSkipAheadStream( stream, nskip )
```

説明

この関数は、指定された処理ノードに与える個々のランダム・ストリームを、ブロック分割法を用いて初期化する (図 8-2 参照)。処理ノードの最大値に上限はない。指定されたストリームで、スキップする要素の最大値が *nskip* である。

図 8-2　ブロック分割法



以下に、SkipAheadStream 関数を用いて 3 つの独立したストリームを初期化するコード例を示す。

例 8-3　ブロック分割法の FORTRAN コード

```
…
TYPE(VSL_STREAM_STATE)stream1
TYPE(VSL_STREAM_STATE)stream2
TYPE(VSL_STREAM_STATE)stream3

! Creating the 1st stream
call vslnewstream(stream1, VSL_BRNG_MCG31, 174)

! Skipping ahead by 7 elements the 2nd stream
call vslcopystream(stream2, stream1);
call vslskipaheadstream(stream2, 7);

! Skipping ahead by 7 elements the 3rd stream
call vslcopystream(stream3, stream2);
call vslskipaheadstream(stream3, 7);

! Generating random numbers
…
! Deleting the streams
call vsldeletestream(stream1)
call vsldeletestream(stream2)
call vsldeletestream(stream3)
…
```

例 8-4 ブロック分割法の C コード

```
VSLStreamStatePtr stream1;
VSLStreamStatePtr stream2;
VSLStreamStatePtr stream3;

/* Creating the 1st stream */
vslNewStream(&stream1, VSL_BRNG_MCG31, 174);

/* Skipping ahead by 7 elements the 2nd stream */
vslCopyStream(&stream2, stream1);
vslSkipAheadStream(stream2, 7);

/* Skipping ahead by 7 elements the 3rd stream */
vslCopyStream(&stream3, stream2);
vslSkipAheadStream(stream3, 7);

/* Generating random numbers */
…
/* Deleting the streams */
vslDeleteStream(&stream1);
vslDeleteStream(&stream2);
vslDeleteStream(&stream3);
…
```

入力パラメータ

FORTRAN:

stream `TYPE(VSL_STREAM_STATE), INTENT(IN)`。ブロック分割法を適用するストリームのディスクリプタ。

nskip `INTEGER, INTENT(IN)`。スキップする要素数。

C:

stream `VSLStreamStatePtr`。ブロック分割法を適用するストリーム・ステート構造体へのポインタ。

nskip `int`。スキップする要素数。

GetStreamStateBrng

指定されたランダム・ストリームを
生成した基本生成器のインデックス値を
返す。

Fortran:

```
brng = vslgetstreamstatebrng( stream )
```

C:

```
brng = vslGetStreamStateBrng( stream )
```

説明

この関数は、指定されたランダム・ストリームの生成に使用された基本生成器のインデックスを返す。

入力パラメータ

FORTRAN:

stream	TYPE(VSL_STREAM_STATE), INTENT(IN)。ストリーム・ステートのディスクリプタ。

C:

stream	VSLStreamStatePtr。ストリーム・ステート構造体へのポインタ。

出力パラメータ

FORTRAN:

brng	INTEGER。ストリームの生成に使用された基本生成器のインデックス値。値が負の場合はエラーを表す。

C:

brng int。ストリームの生成に使用された基本生成器のインデックス値。値が負の場合はエラーを表す。

GetNumRegBrng

現在登録されている基本生成器の数を取得する。

Fortran:

```
nregbrng = vslgetnumregbrngs( )
```

C:

```
nregbrng = vslGetNumRegBrngs( void )
```

説明

この関数は現在登録されている基本生成器の数を取得する。ユーザがユーザ定義生成器を登録すると、登録されている基本生成器の数はインクリメントされる。登録できる基本生成器の上限は VSL_MAX_REG_BRNGS によって決まる。

出力パラメータ

FORTRAN:

nregbrngs INTEGER。関数呼び出しを発した時点で登録されている基本生成器の数。

C:

nregbrngs int。関数呼び出しを発した時点で登録されている基本生成器の数。

擬似乱数生成器

ここでは複数の分布タイプに対応した乱数生成 VSL サブルーチンについて説明する。それぞれの関数は基本となる分布タイプによってグループに分けられ、機能概要に加えて FORTRAN と C インターフェイスの呼び出し手順仕様および入力と出力パラメータの説明が記載される。擬似乱数生成器サブルーチンを、データ・タイプと出力分布と併せて、表 8-2 と表 8-3 に一覧表示した。

表 8-2 連続分布生成器

分布のタイプ	データ・タイプ	説明
Uniform	s, d	区間 (*a*、*b*) に対する連続一様分布
Gaussian_	s, d	正規 (ガウス) 分布
Exponential	s, d	指数分布
Laplace	s, d	ラプラス分布 (両側指数分布)
Weibull	s, d	ワイブル分布
Cauchy	s, d	コーシー分布
Rayleigh	s, d	レイリー分布
Lognormal	s, d	対数正規分布
Gumbel	s, d	ガンベル (極値) 分布

表 8-3 離散分布生成器

分布のタイプ	データ・タイプ	説明
Uniform	i	区間 [*a*、*b*) に対する離散一様分布
UniformBits	i	一様ビット分布を持つ整数乱数生成器
Bernoulli	i	ベルヌーイ分布
Geometric	i	幾何分布
Binomial	i	二項分布
Hypergeometric	i	超幾何分布

表 8-3　離散分布生成器 (続き)

分布のタイプ	データ・タイプ	説明
Poisson	i	ポワソン分布
NegBinomial	i	負の二項分布、またはパスカル分布

連続分布

連続分布の擬似乱数を生成するルーチンについて説明する。

Uniform

一様分布に従う擬似乱数を生成する。

Fortran:

```
call vsrnguniform( method, stream, n, r, a, b )
call vdrnguniform( method, stream, n, r, a, b )
```

C:

```
vsRngUniform( method, stream, n, r, a, b )
vdRngUniform( method, stream, n, r, a, b )
```

説明

この関数は、区間 (a、b) を持つ一様分布の擬似乱数を生成する。
a、b は、それぞれ区間の始まりと終わりであり、$a, b \in R$; $a < b$ 。

確率密度関数は次のように示される。

$$f_{a,b}(x) = \begin{cases} \dfrac{1}{b-a}, & x \in (a, b) \\ 0, & x \notin (a, b) \end{cases}, \quad -\infty < x < +\infty$$

累積分布関数は次のように示される。

$$F_{a,b}(x) = \begin{cases} 0, & x < a \\ \dfrac{x-a}{b-a}, & a \le x < b \\ 1, & x \ge b \end{cases}, \quad -\infty < x < +\infty$$

入力パラメータ

FORTRAN:

method INTEGER, INTENT(IN)。生成方法。このパラメータはダミーで一様分布の場合は 0 を設定する。

stream TYPE (VSL_STREAM_STATE), INTENT(IN)。ストリーム・ステート構造体のディスクリプタ。

n INTEGER, INTENT(IN)。生成させる乱数の個数。

a REAL, INTENT(IN) (vsrnguniform の場合)。

INTENT(IN) (vdrnguniform の場合)。

区間の開始 *a*。

b REAL, INTENT(IN) (vsrnguniform の場合)。

DOUBLE PRECISION, INTENT(IN) (vdrnguniform の場合)。

区間の終了 *b*。

C:

method int。生成方法。このパラメータはダミーで一様分布の場合は 0 を設定する。

stream VSLStreamStatePtr。ストリーム・ステート構造体へのポインタ。

n	int。生成させる乱数の個数。
a	float (vsRngUniform の場合)。 double (vdRngUniform の場合)。 区間の開始 *a*。
b	float (vsRngUniform の場合)。 double (vdRngUniform の場合)。 区間の終了 *b*。

出力パラメータ

FORTRAN:

r	REAL, INTENT(OUT) (vsrnguniform の場合) DOUBLE PRECISION, INTENT(OUT) (vdrnguniform の場合) 区間 (*a*、*b*) を持つ一様分布に従った *n* 個の擬似乱数のベクトル。

C:

r	float* (vsRngUniform の場合) double* (vdRngUniform の場合) 区間 (*a*、*b*) を持つ一様分布に従った *n* 個の擬似乱数のベクトル。

Gaussian

正規分布に従う擬似乱数を生成する。

Fortran:

```
call vsrnggaussian( method, stream, n, r, a, sigma )
call vdrnggaussian( method, stream, n, r, a, sigma )
```

C:

```
vsRngGaussian( method, stream, n, r, a, sigma )
vdRngGaussian( method, stream, n, r, a, sigma )
```

説明

この関数は、平均値 a、標準偏差 σ の正規 (ガウス) 分布の擬似乱数を生成する。ここで

$a, \sigma \in R$; $\sigma > 0$

確率密度関数は次のように示される。

$$f_{a,\sigma}(x) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(x-a)^2}{2\sigma^2}\right), \quad -\infty < x < +\infty$$

累積分布関数は次のように示される。

$$F_{a,\sigma}(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(y-a)^2}{2\sigma^2}\right)dy, \quad -\infty < x < +\infty$$

累積分布関数 $F_{a,\sigma}(x)$ は、標準的な正規分布 $\Phi(x)$ を用いて次のようにも表せる。

$$F_{a,\sigma}(x) = \Phi((x-a)/\sigma)$$

入力パラメータ

FORTRAN:

method	INTEGER, INTENT(IN)。生成方法。
stream	TYPE (VSL_STREAM_STATE), INTENT(IN)。ストリーム・ステート構造体のディスクリプタ
n	INTEGER, INTENT(IN)。生成させる乱数の個数。

a

REAL, INTENT(IN) (vsrnggaussian の場合)

DOUBLE PRECISION, INTENT(IN) (vdrnggaussian の場合)

平均値 *a*。

sigma

REAL, INTENT(IN) (vsrnggaussian の場合)

DOUBLE PRECISION, INTENT(IN) (vdrnggaussian の場合)

標準偏差 σ 。

C:

method

int。生成方法。

stream

VSLStreamStatePtr。ストリーム・ステート構造体へのポインタ。

n

int。生成させる乱数の個数。

a

float (vsRngGaussian の場合)

double (vdRngGaussian の場合)

平均値 *a*。

sigma

float (vsRngGaussian の場合)

double (vdRngGaussian の場合)

標準偏差 σ 。

出力パラメータ

FORTRAN:

r

REAL, INTENT(OUT) (vsrnggaussian の場合)

DOUBLE PRECISION, INTENT(OUT) (vdrnggaussian の場合)

正規分布に従った *n* 個の乱数のベクトル。

C:

r	float* (vsRngGaussian の場合) double* (vdRngGaussian の場合) 正規分布に従った *n* 個の乱数のベクトル。

Exponential

指数分布に従う擬似乱数を生成する。

Fortran:

```
call vsrngexponential( method, stream, n, r, a, beta )
call vdrngexponential( method, stream, n, r, a, beta )
```

C:

```
vsRngExponential( method, stream, n, r, a, beta )
vdRngExponential( method, stream, n, r, a, beta )
```

説明

この関数は、位置母数 *a* 、尺度母数 β を持つ指数分布の擬似乱数を生成する。

$a, \beta \in R\ ;\ \beta > 0$

確率密度関数は次のように示される。

$$f_{a,\beta}(x) = \begin{cases} \frac{1}{\beta}\exp((-(x-a))/\beta), & x \geq a \\ 0, & x < a \end{cases}, \quad -\infty < x < +\infty$$

累積分布関数は次のように示される。

$$F_{a,\beta}(x) = \begin{cases} 1-\exp((-(x-a))/\beta), & x \geq a \\ 0, & x < a \end{cases}, \quad -\infty < x < +\infty$$

入力パラメータ

FORTRAN:

method	INTEGER, INTENT(IN)。生成方法。
stream	TYPE (VSL_STREAM_STATE), INTENT(IN)。ストリーム・ステート構造体のディスクリプタ。
n	INTEGER, INTENT(IN)。生成させる乱数の個数。
a	REAL, INTENT(IN) (vsrngexponential の場合) DOUBLE PRECISION, INTENT(IN) (vdrngexponential の場合) 位置母数 *a*。
beta	REAL, INTENT(IN) (vsrngexponential の場合) DOUBLE PRECISION, INTENT(IN) (vdrngexponential の場合) 尺度母数 β 。

C:

method	int。生成方法。
stream	VSLStreamStatePtr。ストリーム・ステート構造体へのポインタ。
n	int。生成させる乱数の個数。
a	float (vsRngExponential の場合) double (vdRngExponential の場合) 位置母数 *a*。

beta float (vsRngExponential の場合)

double (vdRngExponential の場合)

尺度母数 β 。

出力パラメータ

FORTRAN:

r REAL, INTENT(OUT) (vsrngexponential の場合)

DOUBLE PRECISION, INTENT(OUT) (vdrngexponential の場合)

指数分布に従った *n* 個の擬似乱数のベクトル。

C:

r float* (vsRngExponential の場合)

double* (vdRngExponential の場合)

指数分布に従った *n* 個の擬似乱数のベクトル。

Laplace

ラプラス分布に従う擬似乱数を生成する。

Fortran:

```
call vsrnglaplace( method, stream, n, r, a, beta )
call vdrnglaplace( method, stream, n, r, a, beta )
```

C:

```
vsRngLaplace( method, stream, n, r, a, beta )
vdRngLaplace( method, stream, n, r, a, beta )
```

説明

この関数は、平均値 a 、尺度母数 β を持つラプラス分布の擬似乱数を生成する。ここで $a, \beta \in R$; $\beta > 0$ 。尺度母数から標準偏差が次のように決まる。

$$\sigma = \beta\sqrt{2}$$

確率密度関数は次のように示される。

$$f_{a,\beta}(x) = \frac{1}{\sqrt{2}\beta}\exp\left(-\frac{|x-a|}{\beta}\right), \quad -\infty < x < +\infty$$

累積分布関数は次のように示される。

$$F_{a,\beta}(x) = \begin{cases} \frac{1}{2}\exp\left(-\frac{|x-a|}{\beta}\right), & x < a \\ 1 - \frac{1}{2}\exp\left(-\frac{|x-a|}{\beta}\right), & x \geq a \end{cases}, \quad -\infty < x < +\infty$$

入力パラメータ

FORTRAN:

method	INTEGER, INTENT(IN)。生成方法。
stream	TYPE (VSL_STREAM_STATE), INTENT(IN)。ストリーム・ステート構造体のディスクリプタ。
n	INTEGER, INTENT(IN)。生成させる乱数の個数。
a	REAL, INTENT(IN) (vsrnglaplace の場合) DOUBLE PRECISION, INTENT(IN) (vdrnglaplace の場合) 平均値 *a*。

beta

REAL, INTENT(IN) (vsrnglaplaceÇÃ 場合)

DOUBLE PRECISION, INTENT(IN) (vdrnglaplace の場合)

尺度母数 β 。

C:

method

int。生成方法。

stream

VSLStreamStatePtr。ストリーム・ステート構造体へのポインタ。

n

int。生成させる乱数の個数。

a

float (vsRngLaplace の場合)

double (vdRngLaplace の場合)

平均値 *a*。

beta

float (vsRngLaplace の場合)

double (vdRngLaplace の場合)

尺度母数 β 。

出力パラメータ

FORTRAN:

r

REAL, INTENT(OUT) (vsrnglaplace の場合)

DOUBLE PRECISION, INTENT(OUT) (vdrnglaplace の場合)

ラプラス分布に従った *n* 個の擬似乱数のベクトル。

C:

r

float* (vsRngLaplace の場合)

double* (vdRngLaplace の場合)

ラプラス分布に従った *n* 個の擬似乱数のベクトル。

Weibull

ワイブル分布に従う擬似乱数を生成する。

Fortran:

```
call vsrngweibull( method, stream, n, r, alpha, a, beta )
call vdrngweibull( method, stream, n, r, alpha, a, beta )
```

C:

```
vsRngWeibull( method, stream, n, r, alpha, a, beta )
vdRngWeibull( method, stream, n, r, alpha, a, beta )
```

説明

この関数は、位置母数 a 、尺度母数 β 、形状母数 α を持つワイブル分布の擬似乱数を生成する。ここで $\alpha, \beta, a \in R$; $\alpha > 0$; $\beta > 0$ 。

確率密度関数は次のように示される。

$$f_{a,\,\alpha,\,\beta}(x) = \begin{cases} \dfrac{\alpha}{\beta^{\alpha}}(x-a)^{\alpha-1}\exp\left(-\left(\dfrac{x-a}{\beta}\right)^{\alpha}\right), & x \geq a \\ 0, & x < a \end{cases}$$

累積分布関数は次のように示される。

$$F_{a,\,\alpha,\,\beta}(x) = \begin{cases} 1-\exp\left(-\left(\dfrac{x-a}{\beta}\right)^{\alpha}\right), & x \geq a \\ 0, & x < a \end{cases}, \quad -\infty < x < +\infty$$

入力パラメータ

FORTRAN:

method	INTEGER, INTENT(IN)。生成方法。
stream	TYPE (VSL_STREAM_STATE), INTENT(IN)。ストリーム・ステート構造体のディスクリプタ。
n	INTEGER, INTENT(IN)。生成させる乱数の個数。
alpha	REAL, INTENT(IN) (vsrngweibull の場合) DOUBLE PRECISION, INTENT(IN) (vdrngweibull の場合) 形状母数 α 。
a	REAL, INTENT(IN) (vsrngweibull の場合) DOUBLE PRECISION, INTENT(IN) (vdrngweibull の場合) 位置母数 *a*。
beta	REAL, INTENT(IN) (vsrngweibull の場合) DOUBLE PRECISION, INTENT(IN) (vdrngweibull の場合) 尺度母数 β 。

C:

method	int。生成方法。
stream	VSLStreamStatePtr。ストリーム・ステート構造体へのポインタ。
n	int。生成させる乱数の個数。
alpha	float (vsRngWeibull の場合) double (vdRngWeibull の場合) 形状母数 α 。

a float (vsRngWeibull の場合)

double (vdRngWeibull の場合)

位置母数 *a*。

beta float (vsRngWeibull の場合)

double (vdRngWeibull の場合)

尺度母数 β 。

出力パラメータ

FORTRAN:

r REAL, INTENT(OUT) (vsrngweibull の場合)

DOUBLE PRECISION, INTENT(OUT) (vdrngweibull の場合)

ワイブル分布に従った *n* 個の擬似乱数のベクトル。

C:

r float* (vsRngWeibull の場合)

double* (vdRngWeibull の場合)

ワイブル分布に従った *n* 個の擬似乱数のベクトル。

Cauchy

コーシー分布に従う擬似乱数を生成する。

Fortran:

```
call vsrngcauchy( method, stream, n, r, a, beta )
call vdrngcauchy( method, stream, n, r, a, beta )
```

C:

```
vsRngCauchy( method, stream, n, r, a, beta )
vdRngCauchy( method, stream, n, r, a, beta )
```

説明

この関数は、位置母数 *a* 、尺度母数 β を持つコーシー分布の擬似乱数を生成する。ここで $a, \beta \in R$; $\beta > 0$ 。

確率密度関数は次のように示される。

$$f_{a,\beta}(x) = \frac{1}{\pi\beta\left(1+\left(\frac{x-a}{\beta}\right)^2\right)}, -\infty < x < +\infty$$

累積分布関数は次のように示される。

$$F_{a,\beta}(x) = \frac{1}{2} + \frac{1}{\pi} arctan\left(\frac{x-a}{\beta}\right), \quad -\infty < x < +\infty$$

入力パラメータ

FORTRAN:

method	INTEGER, INTENT(IN)。生成方法。
stream	TYPE (VSL_STREAM_STATE), INTENT(IN)。ストリーム・ステート構造体のディスクリプタ。
n	INTEGER, INTENT(IN)。生成させる乱数の個数。
a	REAL, INTENT(IN) (vsrngcauchy の場合) DOUBLE PRECISION, INTENT(IN) (vdrngcauchy の場合) 位置母数 *a*。

beta

REAL, INTENT(IN) (vsrngcauchy の場合)

DOUBLE PRECISION, INTENT(IN) (vdrngcauchy の場合)

尺度母数 β 。

C:

method

int。生成方法。

stream

VSLStreamStatePtr。ストリーム・ステート構造体へのポインタ。

n

int。生成させる乱数の個数。

a

float (vsRngCauchy の場合)

double (vdRngCauchy の場合)

位置母数 *a*。

beta

float (vsRngCauchy の場合)

double (vdRngCauchy の場合)

尺度母数 β 。

出力パラメータ

FORTRAN:

r

REAL, INTENT(OUT) (vsrngcauchy の場合)

DOUBLE PRECISION, INTENT(OUT) (vdrngcauchy の場合)

コーシー分布に従った *n* 個の擬似乱数のベクトル。

C:

r

float* (vsRngCauchy の場合)

double* (vdRngCauchy の場合)

コーシー分布に従った *n* 個の擬似乱数のベクトル。

Rayleigh

レイリー分布に従う擬似乱数を生成する。

Fortran:

```
call vsrngrayleigh( method, stream, n, r, a, beta )
call vdrngrayleigh( method, stream, n, r, a, beta )
```

C:

```
vsRngRayleigh( method, stream, n, r, a, beta )
vdRngRayleigh( method, stream, n, r, a, beta )
```

説明

この関数は、位置母数 a 、尺度母数 β を持つレイリー分布の擬似乱数を生成する。ここで $a, \beta \in R$; $\beta > 0$ 。

レイリー分布は Weibull 分布の一種であり、形状母数 $\alpha = 2$ のときである。

確率密度関数は次のように示される。

$$f_{a,\beta}(x) = \begin{cases} \dfrac{2(x-a)}{\beta^2}\exp\left(-\dfrac{(x-a)^2}{\beta^2}\right), & x \geq a \\ 0, & x < a \end{cases}, \quad -\infty < x < +\infty$$

累積分布関数は次のように示される。

$$F_{a,\beta}(x) = \begin{cases} 1-\exp\left(-\dfrac{(x-a)^2}{\beta^2}\right), & x \geq a \\ 0, & x < a \end{cases}, \quad -\infty < x < +\infty$$

入力パラメータ

FORTRAN:

method	INTEGER, INTENT(IN)。生成方法。
stream	TYPE (VSL_STREAM_STATE), INTENT(IN)。ストリーム・ステート構造体のディスクリプタ。
n	INTEGER, INTENT(IN)。生成させる乱数の個数。
a	REAL, INTENT(IN) (vsrngrayleigh の場合) DOUBLE PRECISION, INTENT(IN) (vdrngrayleigh の場合) 位置母数 *a*。
beta	REAL, INTENT(IN) (vsrngrayleigh の場合) DOUBLE PRECISION, INTENT(IN) (vdrngrayleigh の場合) 尺度母数 β 。

C:

method	int。生成方法。
stream	VSLStreamStatePtr。ストリーム・ステート構造体へのポインタ。
n	int。生成させる乱数の個数。
a	float (vsRngRayleigh の場合) double (vdRngRayleigh の場合) 位置母数 *a*。
beta	float (vsRngRayleigh の場合) double (vdRngRayleigh の場合) 尺度母数 β 。

出力パラメータ

FORTRAN:

r	REAL, INTENT(OUT) (vsrngrayleigh の場合) DOUBLE PRECISION, INTENT(OUT) (vdrngrayleigh の場合) レイリー分布に従った *n* 個の擬似乱数のベクトル。

C:

r	float* (vsRngRayleigh の場合) double* (vdRngRayleigh の場合) レイリー分布に従った *n* 個の擬似乱数のベクトル。

Lognormal

対数正規分布に従う擬似乱数を生成する。

Fortran:

```
call vsrnglognormal( method, stream, n, r, a, sigma, b, beta )
call vdrnglognormal( method, stream, n, r, a, sigma, b, beta )
```

C:

```
vsRngLognormal( method, stream, n, r, a, sigma, b, beta )
vdRngLognormal( method, stream, n, r, a, sigma, b, beta )
```

説明

この関数は、分布の平均 a 、正規分布の標準偏差 σ 、位置母数 b 、尺度母数 β を持つ対数正規分布の擬似乱数を生成する。ここで、$a, \sigma, b, \beta \in R$; $\sigma > 0$; $\beta > 0$ 。

確率密度関数は次のように示される。

$$f_{a,\sigma,b,\beta}(x) = \begin{cases} \dfrac{1}{\sigma(x-b)\sqrt{2\pi}} \exp\left(-\dfrac{[\ln((x-b)/\beta)-a]^2}{2\sigma^2}\right), & x > b \\ 0, & x \le b \end{cases}$$

累積分布関数は次のように示される。

$$F_{a,\sigma,b,\beta}(x) = \begin{cases} \Phi((\ln((x-b)/\beta)-a)/\sigma), & x > b \\ 0, & x \le b \end{cases}$$

入力パラメータ

FORTRAN:

method	INTEGER, INTENT(IN)。生成方法。
stream	TYPE (VSL_STREAM_STATE), INTENT(IN)。ストリーム・ステート構造体のディスクリプタ。
n	INTEGER, INTENT(IN)。生成させる乱数の個数。
a	REAL, INTENT(IN) (vsrnglognormal の場合) DOUBLE PRECISION, INTENT(IN) (vdrnglognormal の場合) 正規分布の平均 *a*。

sigma REAL, INTENT(IN) (vsrnglognormal の場合)

DOUBLE PRECISION, INTENT(IN) (vdrnglognormal の場合)

正規分布の標準偏差 σ。

b REAL, INTENT(IN) (vsrnglognormal の場合)

DOUBLE PRECISION, INTENT(IN) (vdrnglognormal の場合)

位置母数 *b*。

beta REAL, INTENT(IN) (vsrnglognormal の場合)

DOUBLE PRECISION, INTENT(IN) (vdrnglognormal の場合)

尺度母数 β。

C:

method int。生成方法。

stream VSLStreamStatePtr。ストリーム・ステート構造体へのポインタ。

n int。生成させる乱数の個数。

a float (vsRngLognormal の場合)

double (vdRngLognormal の場合)

正規分布の平均 *a*。

sigma float (vsRngLognormal の場合)

double (vdRngLognormal の場合)

正規分布の標準偏差 σ。

b float (vsRngLognormal の場合)

double (vdRngLognormal の場合)

位置母数 *b*。

beta float (vsRngLognormal の場合)

double (vdRngLognormal の場合)

尺度母数 β 。

出力パラメータ

FORTRAN:

r REAL, INTENT(OUT) (vsrnglognormal の場合)

DOUBLE PRECISION, INTENT(OUT) (vdrnglognormal の場合)

対数正規分布に従った *n* 個の擬似乱数のベクトル。

C:

r float* (vsRngLognormal の場合)

double* (vdRngLognormal の場合)

対数正規分布に従った *n* 個の擬似乱数のベクトル。

Gumbel

ガンベル分布に従う擬似乱数を生成する。

Fortran:

```
call vsrnggumbel( method, stream, n, r, a, beta )
call vdrnggumbel( method, stream, n, r, a, beta )
```

C:

```
vsRngGumbel( method, stream, n, r, a, beta )
vdRngGumbel( method, stream, n, r, a, beta )
```

説明

この関数は、位置母数 a 、尺度母数 β を持つガンベル分布の擬似乱数を生成する。ここで、$a, \beta \in R$; $\beta > 0$ 。

確率密度関数は次のように示される。

$$f_{a,\beta}(x) = \frac{1}{\beta}\exp\left(\frac{x-a}{\beta}\right)\exp(-\exp((x-a)/\beta)), \quad -\infty < x < +\infty$$

累積分布関数は次のように示される。

$$F_{a,\beta}(x) = 1 - \exp(-\exp((x-a)/\beta)), \quad -\infty < x < +\infty$$

入力パラメータ

FORTRAN:

method	INTEGER, INTENT(IN)。生成方法。
stream	TYPE (VSL_STREAM_STATE), INTENT(IN)。ストリーム・ステート構造体のディスクリプタ。
n	INTEGER, INTENT(IN)。生成させる乱数の個数。
a	REAL, INTENT(IN) (vsrnggumbel の場合) DOUBLE PRECISION, INTENT(IN) (vdrnggumbel の場合) 位置母数 *a*。
beta	REAL, INTENT(IN) (vsrnggumbel の場合) DOUBLE PRECISION, INTENT(IN) (vdrnggumbel の場合) 尺度母数 β 。

C:

method	int。生成方法。

stream	VSLStreamStatePtr。ストリーム・ステート構造体へのポインタ。
n	int。生成させる乱数の個数。
a	float (vsRngGumbel の場合) double (vdRngGumbel の場合) 位置母数 *a*。
beta	float (vsRngGumbel の場合) double (vdRngGumbel の場合) 尺度母数 β 。

出力パラメータ

FORTRAN:

r	REAL, INTENT(OUT) (vsrnggumbel の場合) DOUBLE PRECISION, INTENT(OUT) (vdrnggumbel の場合) ガンベル分布に従った *n* 個の擬似乱数のベクトル。

C:

r	float* (vsRngGumbel の場合) double* (vdRngGumbel の場合) ガンベル分布に従った *n* 個の擬似乱数のベクトル。

離散分布

離散分布の擬似乱数を生成するルーチンについて説明する。

Uniform

区間 $[a, b)$ を持つ一様離散分布に従った
擬似乱数を生成する。

Fortran:

```
call virnguniform( method, stream, n, r, a, b )
```

C:

```
viRngUniform( method, stream, n, r, a, b )
```

説明

この関数は区間 $[a, b)$ を持つ一様離散分布に従った擬似乱数を生成する。a、b は、それぞれ区間の始まりと終わりであり、$a, b \in Z$; $a < b$。

確率分布は次のように示される。

$$P(X = k) = \frac{1}{b - a}, \quad k \in \{a, a+1, \ldots, b-1\}$$

累積分布関数は次のように示される。

$$F_{a,b}(x) = \begin{cases} 0, & x < a \\ \dfrac{\lfloor x - a + 1 \rfloor}{b - a}, & a \le x < b, \quad x \in R \\ 1, & x \ge b \end{cases}$$

入力パラメータ

FORTRAN:

method INTEGER, INTENT(IN)。生成方法。

stream	TYPE (VSL_STREAM_STATE), INTENT(IN)。ストリーム・ステート構造体のディスクリプタ。
n	INTEGER, INTENT(IN)。生成させる乱数の個数。
a	INTEGER, INTENT(IN)。区間の開始 *a*。
b	INTEGER, INTENT(IN)。区間の終了 *b*。

C:

method	int。生成方法。
stream	VSLStreamStatePtr。ストリーム・ステート構造体へのポインタ。
n	int。生成させる乱数の個数。
a	int。区間の開始 *a*。
b	int。区間の終了 *b*。

出力パラメータ

FORTRAN:

r	INTEGER, INTENT(OUT)。区間 [*a*、*b*) を持つ一様離散分布に従った *n* 個の擬似乱数のベクトル。

C:

r	int*。区間 [*a*、*b*) を持つ一様離散分布に従った *n* 個の擬似乱数のベクトル。

UniformBits

一様ビット分布に従う擬似乱数を
生成する。

Fortran:

```
call virnguniformbits( method, stream, n, r )
```

C:

```
viRngUniformBits( method, stream, n, r )
```

説明

この関数は一様ビット分布に従う整数の乱数を生成する。一様分布の乱数生成器は、整数値に対する剰余算の漸化式として表せる。各整数は、それぞれ複数のビットを持つベクトルとして扱えることは明らかである。真の乱数を発する生成器ではこれらビットはランダムであるが、擬似乱数生成器ではランダムさは失われる。たとえば線形合同生成器には、上位ビットに比べて下位ビットのランダムさが少ないというよく知られた欠点がある (たとえば [Knuth81] を参照)。この理由から、この関数を使用する場合は注意が必要である。一般に 32 ビット LCG では、整数値の上位 24 ビットのみが真にランダムであるとみなされる。詳細は VSLNotes を参照のこと。

入力パラメータ

FORTRAN:

method	INTEGER, INTENT(IN)。生成方法。virnguniformbitsではダミーのパラメータ。ゼロにしなければならない。
stream	TYPE (VSL_STREAM_STATE), INTENT(IN)。ストリーム・ステート構造体のディスクリプタ。
n	INTEGER, INTENT(IN)。生成させる乱数の個数。

C:

method	int。生成方法。viRngUniformBits ではダミーのパラメータ。ゼロにしなければならない。
stream	VSLStreamStatePtr。ストリーム・ステート構造体へのポインタ。
n	int。生成させる乱数の個数。

出力パラメータ

FORTRAN:

r	INTEGER, INTENT(OUT)。*n* 個の擬似乱数のベクトル。ストリームが 64 ビットまたは 128 ビットの生成器で生成された場合、各整数値は 2 個または 4 個の *r* の要素でそれぞれ表される。各整数が占有するバイト数は VSL_BRNG_PROPERTIES 構造体の *wordsize* フィールドで指定される。値を格納するために実際に使用される合計ビット数は同 VSL_BRNG_PROPERTIES 構造体の *nbits* フィールドで指定される。VSL_BRNG_PROPERTIES の詳細は「アドバンスト・サービス・サブルーチン」を参照のこと。

C:

r	unsigned int*。*n* 個の擬似乱数のベクトル。ストリームが 64 ビットまたは 128 ビットの生成器で生成された場合、各整数値は 2 個または 4 個の *r* の要素でそれぞれ表される。各整数が占有するバイト数は VSLBrngProperties 構造体の *WordSize* フィールドで指定される。値を格納するために実際に使用される合計ビット数は同 VSLBrngProperties 構造体の *NBits* フィールドで指定される。VSL_BRNG_PROPERTIES の詳細は「アドバンスト・サービス・サブルーチン」を参照のこと。

Bernoulli

ベルヌーイ分布に従う擬似乱数を生成する。

Fortran:

```
call virngbernoulli( method, stream, n, r, p )
```

C:

```
viRngBernoulli( method, stream, n, r, p )
```

説明

この関数は、単一の試行成功確率を p とするベルヌーイ分布に従った擬似乱数を生成する。ここで

$p \in R$; $0 \leq p \leq 1$ 。

変数はベルヌーイ分布と呼ばれ、試行の結果 1 となる成功確率は *p*、0 となる確立は *1-p* である。

確率分布は次のように示される。

$P(X = 1) = p$,

$P(X = 0) = 1 - p$

累積分布関数は次のように示される。

$$F_p(x) = \begin{cases} 0, & x < 0 \\ 1 - p, & 0 \le x < 1 \\ 1, & x \ge 1 \end{cases}, \quad x \in R$$

入力パラメータ

FORTRAN:

method	INTEGER, INTENT(IN)。生成方法。
stream	TYPE (VSL_STREAM_STATE), INTENT(IN)。ストリーム・ステート構造体のディスクリプタ。
n	INTEGER, INTENT(IN)。生成させる乱数の個数。
p	DOUBLE PRECISION, INTENT(IN) 。試行の成功確率 *p*。

C:

method	int。生成方法。
stream	VSLStreamStatePtr。ストリーム・ステート構造体へのポインタ。
n	int。生成させる乱数の個数。
p	double。試行の成功確率 *p*。

出力パラメータ

FORTRAN:

r	INTEGER, INTENT(OUT)。ベルヌーイ分布に従った *n* 個の擬似乱数のベクトル。

C:

r	int*。ベルヌーイ分布に従った *n* 個の擬似乱数のベクトル。

Geometric

幾何分布に従う擬似乱数を生成する。

Fortran:

```
call virnggeometric( method, stream, n, r, p )
```

C:

```
viRngGeometric( method, stream, n, r, p )
```

説明

この関数は、試行の成功確率を p とする幾何分布に従った擬似乱数を生成する。ここで $p \in R$; $0 < p < 1$ 。

幾何分布の変数は最初の成功が得られるまでの独立したベルヌーイ試行の回数である。ベルヌーイ試行の成功確率は *p* である。

確率分布は次のように示される。

$$P(X = k) = p \cdot (1-p)^k, \quad k \in \{0, 1, 2, \ldots\}$$

累積分布関数は次のように示される。

$$F_p(x) = \begin{cases} 0, & x < 0 \\ 1-(1-p)^{\lfloor x+1 \rfloor}, & x \geq 0 \end{cases}, \quad x \in R$$

入力パラメータ

FORTRAN:

method	INTEGER, INTENT(IN)。生成方法。
stream	TYPE (VSL_STREAM_STATE), INTENT(IN)。ストリーム・ステート構造体のディスクリプタ。
n	INTEGER, INTENT(IN)。生成させる乱数の個数。
p	DOUBLE PRECISION, INTENT(IN) 。試行の成功確率 *p*。

C:

method	int。生成方法。
stream	VSLStreamStatePtr。ストリーム・ステート構造体へのポインタ。
n	int。生成させる乱数の個数。
p	double。試行の成功確率 *p*。

出力パラメータ

FORTRAN:

r	INTEGER, INTENT(OUT)。幾何分布に従った *n* 個の擬似乱数のベクトル。

C:

r	int*。幾何分布に従った *n* 個の擬似乱数のベクトル。

Binomial

二項分布に従う擬似乱数を生成する。

Fortran:

```
call virngbinomial( method, stream, n, r, ntrial, p )
```

C:

```
viRngBinomial( method, stream, n, r, ntrial, p )
```

説明

この関数は、独立したベルヌーイ試行回数 m と単一の試行成功確率 p を持つ二項分布に従った擬似乱数を生成する。ここで $p \in R$; $0 \le p \le 1$, $m \in N$ 。

二項分布の変数は、単一の試行成功確率 *p* を持つベルヌーイ試行を、独立に m 回行ったときの成功回数として表される。

確率分布は次のように示される。

$$P(X = k) = C_m^k p^k (1-p)^{m-k}, \quad k \in \{0, 1, \ldots, m\}$$

累積分布関数は次のように示される。

$$F_{m,p}(x) = \begin{cases} 0, & x < 0 \\ \displaystyle\sum_{k=0}^{\lfloor x \rfloor} C_m^k p^k (1-p)^{m-k}, & 0 \le x < m, \quad x \in R \\ 1, & x \ge m \end{cases}$$

入力パラメータ

FORTRAN:

method	INTEGER, INTENT(IN)。生成方法。
stream	TYPE (VSL_STREAM_STATE), INTENT(IN)。ストリーム・ステート構造体のディスクリプタ。

n	INTEGER, INTENT(IN)。生成させる乱数の個数。
ntrial	INTEGER, INTENT(IN)。独立した試行回数 *m*。
p	DOUBLE PRECISION, INTENT(IN)。単一試行の成功確率 *p*。

C:

method	int。生成方法。
stream	VSLStreamStatePtr。ストリーム・ステート構造体へのポインタ。
n	int。生成させる乱数の個数。
ntrial	int。独立した試行回数 *m*。
p	double。単一試行の成功確率 *p*。

出力パラメータ

FORTRAN:

r	INTEGER, INTENT(OUT)。二項分布に従った *n* 個の擬似乱数のベクトル。

C:

r	int*。二項分布に従った *n* 個の擬似乱数のベクトル。

Hypergeometric

超幾何分布に従う擬似乱数を生成する。

Fortran:

```
call virnghypergeometric( method, stream, n, r, l, s, m )
```

C:

```
viRngHypergeometric( method, stream, n, r, l, s, m )
```

説明

この関数は、母集団の大きさ l、抽出の大きさ s、母集団内の「あたり」の要素数 m を持つ超幾何分布に従った擬似乱数を生成する。ここで $l, m, s \in N \cup \{0\}$; $l \geq max(s, m)$。

母集団 *l* の要素は、*m* 個の「あたり」要素と *l-m* 個の「はずれ」要素で構成されているとする。この母集団から復元なしで *s* 個の要素を試行抽出すると、*s* 個の要素の中に *k* 個の「あたり」を含んでいる確率を示す超幾何分布が得られる。

確率分布は次のように示される。

$$P(X = k) = \frac{C_m^k C_{l-m}^{s-k}}{C_l^s}, \quad k \in \{max(0, s+m-l), \ldots, min(s, m)\}$$

累積分布関数は次のように示される。

$$F_{l,s,m}(x) = \begin{cases} 0, & x < max(0, s+m-l) \\ \displaystyle\sum_{k=max(0, s+m-l)}^{\lfloor x \rfloor} \frac{C_m^k C_{l-m}^{s-k}}{C_l^s}, & max(0, s+m-l) \leq x \leq min(s, m) \\ 1, & x > min(s, m) \end{cases}$$

入力パラメータ

FORTRAN:

method	INTEGER, INTENT(IN)。生成方法。
stream	TYPE (VSL_STREAM_STATE), INTENT(IN)。ストリーム・ステート構造体のディスクリプタ。
n	INTEGER, INTENT(IN)。生成させる乱数の個数。
l	INTEGER, INTENT(IN)。母集団の大きさ *l*。
s	INTEGER, INTENT(IN)。復元なしに抽出する個数 *s*。
m	INTEGER, INTENT(IN)。「あたり」要素の個数 *m*。

C:

method	int。生成方法。
stream	VSLStreamStatePtr。ストリーム・ステート構造体へのポインタ。
n	int。生成させる乱数の個数。
l	int。母集団の大きさ *l*。
s	int。復元なしに抽出する個数 *s*。
m	int。「あたり」要素の個数 *m*。

出力パラメータ

FORTRAN:

r	INTEGER, INTENT(OUT)。超幾何分布に従った *n* 個の擬似乱数のベクトル。

C:

r	int*。超幾何分布に従った *n* 個の擬似乱数のベクトル。

Poisson

ポアソン分布に従う擬似乱数を生成する。

Fortran:

```
call virngpoisson( method, stream, n, r, lambda )
```

C:

```
viRngPoisson( method, stream, n, r, lambda )
```

説明

この関数は、分布パラメータを λ とするポアソン分布に従った擬似乱数を生成する。ここで $\lambda \in R;\ \lambda > 0$ 。

確率分布は次のように示される。

$$P(X = k) = \frac{\lambda^k e^{-\lambda}}{k!}, \quad k \in \{0, 1, 2, \ldots\}$$

累積分布関数は次のように示される。

$$F_\lambda(x) = \begin{cases} \displaystyle\sum_{k=0}^{\lfloor x \rfloor} \frac{\lambda^k e^{-\lambda}}{k!}, & x \geq 0 \\ 0, & x < 0 \end{cases}, \quad x \in R$$

入力パラメータ

FORTRAN:

method	INTEGER, INTENT(IN)。生成方法。
stream	TYPE (VSL_STREAM_STATE), INTENT(IN)。ストリーム・ステート構造体のディスクリプタ。
n	INTEGER, INTENT(IN)。生成させる乱数の個数。
lambda	DOUBLE PRECISION, INTENT(IN) 。分布パラメータλ。

C:

method	int。生成方法。
stream	VSLStreamStatePtr。ストリーム・ステート構造体へのポインタ。
n	int。生成させる乱数の個数。
lambda	double。分布パラメータλ。

出力パラメータ

FORTRAN:

r	INTEGER, INTENT(OUT)。ポアソン分布に従った *n* 個の擬似乱数のベクトル。

C:

r	int*。ポアソン分布に従った *n* 個の擬似乱数のベクトル。

NegBinomial

負の二項分布に従う擬似乱数を生成する。

Fortran:

```
call virngnegbinomial( method, stream, n, r, a, p )
```

C:

```
viRngNegBinomial( method, stream, n, r, a, p )
```

説明

この関数は、分布パラメータが a と p を持つ負の二項分布に従った擬似乱数を生成する。ここで、$p, a \in R$; $0 < p < 1$; $a > 0$ 。

第 1 の分布パラメータが $a \in N$ なら、この分布はパスカル分布と等しくなる。この分布は $a \in N$ では、成功確率 p のベルヌーイ試行のときに、a 番目の成功を得るまでの見込まれる試行回数として解釈できる。

確率分布は次のように示される。

$$P(X = k) = C^{k}_{a+k-1} p^{a} (1-p)^{k}, \quad k \in \{0, 1, 2, \ldots\}$$

累積分布関数は次のように示される

$$F_{a,p}(x) = \begin{cases} \sum_{k=0}^{\lfloor x \rfloor} C^{k}_{a+k-1} p^{a} (1-p)^{k}, & x \geq 0 \\ 0, & x < 0 \end{cases}, \quad x \in R$$

入力パラメータ

FORTRAN:

method	INTEGER, INTENT(IN)。生成方法。
stream	TYPE (VSL_STREAM_STATE), INTENT(IN)。ストリーム・ステート構造体のディスクリプタ。
n	INTEGER, INTENT(IN)。生成させる乱数の個数。
a	DOUBLE PRECISION, INTENT(IN)。第 1 の分布パラメータ *a*。
p	DOUBLE PRECISION, INTENT(IN)。第 2 の分布パラメータ *p*。

C:

method	int。生成方法。
stream	VSLStreamStatePtr。ストリーム・ステート構造体へのポインタ。
n	int。生成させる乱数の個数。
a	double。第 1 の分布パラメータ *a*。
p	double。第 2 の分布パラメータ *p*。

出力パラメータ

FORTRAN:

r	INTEGER, INTENT(OUT)。負の二項分布に従った *n* 個の擬似乱数のベクトル。

C:

r	int*。負の二項分布に従った *n* 個の擬似乱数のベクトル。

アドバンスト・サービス・サブルーチン

この節では、基本生成器の登録を行うサブルーチンと、登録済み基本生成器のプロパティを取得するサブルーチンについて説明する。

表 8-4 アドバンスト・サービス・サブルーチン

サブルーチン	説明
RegisterBrng	ユーザ定義の基本生成器を登録する。
GetBrngProperties	指定された番号の基本生成器に関するプロパティを構造体で返す。

データ・タイプ

ここで記載のサブルーチン群は、基本生成器のプロパティを定義する構造体を参照する。

例 8-5 FORTRAN バージョン

```
TYPE VSL_BRNG_PROPERTIES
              INTEGER streamstatesize
              INTEGER nseeds
              INTEGER includeszero
              INTEGER wordsize
              INTEGER nbits
              INTEGER initstream
              INTEGER sbrng
              INTEGER dbrng
              INTEGER ibrng
END TYPE VSL_BRNG_PROPERTIES
```

例 8-6 C バージョン

```
typedef struct _VSLBRngProperties {
              int              StreamStateSize;
              int              NSeeds;
              int              IncludesZero;
              int              WordSize;
              int              NBits;
              InitStreamPtr    InitStream;
              sBRngPtr         sBRng;
              dBRngPtr         dBRng;
              iBRngPtr         iBRng;
} VSLBRngProperties;
```

例 8-7 関数へのポインタ

```
typedef   int (*InitStreamPtr)( int method, void * stream, int n,
                                const unsigned int params[] );
typedef   void (*sBRngPtr)( void * stream, int n, float r[],
                            float a, float b );
typedef   void (*dBRngPtr)( void * stream, int n, double r[],
                            double a, double b );
typedef   void (*iBRngPtr)( void * stream, int n,
                            unsigned int r[] );
```

表 8-5 フィールドの説明

フィールド	説明
FORTRAN: streamstatesize C: StreamStateSize	バイトを単位とする、指定された基本生成器に対するストリーム・ステート構造体のサイズ。
FORTRAN: nseeds C: NSeeds	指定された基本生成器に対するストリーム・ステート構造体の初期化に必要な、32 ビット初期条件 (種) の個数。
FORTRAN: includeszero C: IncludesZero	生成器が擬似乱数 0^1 を生成可能であることを示すフラグ値。
FORTRAN: wordsize C: WordSize	バイトを単位とする、整数値計算で使用されるマシン・ワード・サイズ。取り得る値は、32 ビット、64 ビット、128 ビット生成器で、それぞれ 4、8、16。
FORTRAN: nbits C: NBits	整数算で擬似乱数を表すために必要となるビット数。たとえば、48 ビット擬似乱数は 64 ビット (8 バイト) メモリ・ロケーションに格納される点に注意する。この場合を例に取ると、WordSize (その擬似乱数の格納に使用されるバイト数) は 8 となるが、NBits はその値によって占有される実際のビット数であるから 48 となる。

表 8-5　フィールドの説明（続き）

フィールド	説明

FORTRAN:
`initstream`
C:
`InitStream`
	指定された基本生成器の初期化サブルーチンに対するポインタ。

FORTRAN:
`sbrng`
C:
`sBRng`
	区間 (*a*, *b*) を持つ一様分布を単精度実数で処理する基本生成器に対するポインタ (FORTRAN では `REAL`、C では `float`)。

FORTRAN:
`dbrng`
C:
`dBRng`
	区間 (*a*, *b*) を持つ一様分布を倍精度実数で処理する基本生成器に対するポインタ (FORTRAN では `DOUBLE PRECISION`、C では `double`)。

FORTRAN:
`ibrng`
C:
`iBRng`
	一様ビット分布[2] を整数で処理する基本生成器に対するポインタ (FORTRAN では `INTEGER`、C では `unsigned int`)。

1. たとえば GFSR のような一部の生成器はゼロの擬似乱数を生成する可能性を持っている。一方、乗算合同生成器などはゼロを生成することはあり得ない。ゼロが生成される確率はきわめて低いため、一般にこのような情報にはそれほど意味はない。しかし、一部の非一様分布生成器では、基本生成器がゼロを生成する可能性があるのは、無限に近い大きな値を生成する可能性もあることを意味する (オーバーフロー)。たとえソフトウェアがオーバーフローを＋∞や－∞として正しく取り扱ったとしても、このような性質には注意が必要であり、結果の確認が求められる。無限に近い大きな値が計算に影響を与える場合、生成されたベクトルからそのような値を取り除くか、あるいはゼロを生成しない安全な生成器を使用する必要がある。
2. 擬似乱数の単一ビットとビット・グループとの演算を許している特有の生成器である。

RegisterBrng

ユーザ定義の基本生成器を登録する。

Fortran:

```
brng = vslregisterbrng( properties )
```

C:

```
brng = vslRegisterBrng( properties )
```

説明

登録手順の例は各システムの vsl\examples ディレクトリにある。

入力パラメータ

FORTRAN:

properties	TYPE (VSL_BRNG_PROPERTIES), INTENT(IN)。登録する基本生成器のプロパティを含む構造体。

C:

properties	VSLBrngProperties*。登録する基本生成器のプロパティを含む構造体。

出力パラメータ

FORTRAN:

brng	INTEGER。登録された基本生成器の番号 (インデックス値) で、識別に使用する。負の値が返ってきた場合は登録時にエラーが発生したことを示す。

C:

brng int。登録された基本生成器の番号 (インデックス値) で、識別に使用する。負の値が返ってきた場合は登録時にエラーが発生したことを示す。

GetBrngProperties

指定した基本生成器のプロパティを構造体で返す。

Fortran:

```
call vslgetbrngproperties( brng, properties )
```

C:

```
call vslGetBrngProperties( brng, properties )
```

入力パラメータ

FORTRAN:

brng INTEGER, INTENT(IN)。登録されている基本生成器の番号 (インデックス)。

C:

brng int。登録されている基本生成器の番号 (インデックス)。

出力パラメータ

FORTRAN:

properties TYPE (VSL_BRNG_PROPERTIES), INTENT(OUT)。番号 *brng* を持つ生成器のプロパティを含んだ構造体。

C:

properties	VSLBrngProperties*。番号 *brng* を持つ生成器のプロパティを含んだ構造体。

ユーザ定義生成器のフォーマット

RegisterBrng 関数を用いてユーザ定義基本生成器を登録するには、整数値で実装されている生成器のポインタを *iBrng* に、単精度値および倍精度値で実装されている生成器のポインタをそれぞれ *sBrng* と *dBrng* に、ストリーム初期化サブルーチンのポインタを *InitStream* に、それぞれ渡さなければならない。以下のセクションに、入力引数と出力引数の使い方と合わせ各関数定義の推奨例を記載している。ユーザ定義生成器の登録手順例は各システムの vsl\examples ディレクトリにある。

InitStream

FORTRAN:

```
INTEGER FUNCTION mybrnginitstream( method, stream, n, params )
      INTEGER, INTENT (IN):: method
      TYPE(MYSTREAM_STATE), INTENT (INOUT):: stream
      INTEGER, INTENT (IN):: n
      INTEGER, INTENT (IN):: params
! Initialize the stream
      …
END SUBROUTINE mybrnginitstream
```

C:

```
int MyBrngInitStream( int method, VSLStreamStatePtr stream,
      int n, const unsigned int params[] )
{
            /* Initialize the stream */
            …
} /* MyBrngInitStream */
```

説明

ユーザ定義生成器の初期化サブルーチンは、指定された初期化 *method*、初期条件 *params*、引数 *n* に従って *stream* を初期化しなければならない。*method* の値によって初期化方法が指定される。

- *method* が 0 の場合、すべての基本生成器がサポートしなければならない標準的な生成方法を用いて初期化を行う。ここで関数は、*stream* 構造体は過去に初期化されていないと仮定する。*n* は、初期化条件として *params* を介して渡される 32 ビット値の個数を表す。関数に渡される初期条件の個数が生成器の初期化に足りない場合でもエラーではないことに注意が必要である。このような状況が起こった場合、基本生成器は、不足している初期条件に代わってデフォルト値を用いて初期化を行わなければならない。
- *method* が 1 の場合はリープフロッグ法による生成となり、*n* は処理ノード (独立ストリーム) 数を指定する。*stream* は標準生成法で初期化されていると仮定する。*params* は処理ノードを指定する単一要素のみで構成される。生成器がリープフロッグ法をサポートしない場合、関数はエラーコード VSL_ERROR_LEAPFROG_UNSUPPORTED を返さなければならない。
- *method* が 2 の場合はブロック分割法による生成となる。上記と同様に、*stream* は標準生成法で初期化されていると仮定する。*params* は使用されない。*n* はスキップする要素数を指定する。生成器がブロック分割法をサポートしていない場合、関数はエラーコード VSL_ERROR_SKIPAHEAD_UNSUPPORTED を返さなければならない。

リープフロッグ法とブロック分割法の詳細は、LeapfrogStream および SkipAheadStream をそれぞれ参照のこと。

ストリーム・ステート構造体はそれぞれの生成器ごとに固有である。ただし、すべての生成器で共通なフィールドはすべての構造体が持っている。

FORTRAN:

```
type(mystream_state)
        INTEGER*4     reserved1
        INTEGER*4     reserved2
        INTEGER*4     reserved3
        INTEGER*4     reserved4
        [ fields specific for the given generator ]
end type mystream_state
```

C:

```
typedef struct
{
        uint64   Reserved1;
        uint64   Reserved2;
        [ fields specific for the given generator ]
} MyStreamState
```

フィールド *Reserved1* と *Reserved2* は非公開目的のために予約されており、ユーザは変更してはならない。構造体に専用のフィールドを追加する場合は次の規則に従うこと。

- フィールドは生成器の最新ステートを完全に記述しなくてはならない。たとえば、線形合同生成器のステートは単一の初期条件しか持たない。
- 生成器がリープフロッグ法とブロック分割法の両方に対応するのであれば、独立したストリームを識別するために追加フィールドを導入すべきである。たとえば $LCG(a, c, m)$ では、初期条件とは別に、乗数 a^k と、加数 $(a^k - 1)c/(a - 1)$ の 2 つの追加フィールドを規定する必要がある。

詳細は [Knuth81] と [Gentle98] を参照のこと。登録手順の例は各システムの `vsl\examples` ディレクトリにある。

iBRng

FORTRAN:

```
SUBROUTINE imybrng( stream, n, r )
      TYPE(MYSTREAM_STATE), INTENT(INOUT):: stream
      INTEGER, INTENT(IN)    :: n
      INTEGER, DIMENSION(*), INTENT(OUT) :: r
! Generating integer random numbers
! Pay attention to word size needed to
! store one random number
      DO i = 1, n
             R(I) = …
      END DO
! Update stream state
END SUBROUTINE imybrng
```

C:

```
void iMyBrng( VSLStreamStatePtr stream, int n,
              unsigned int r[] )
{
       int   i;    /* Loop variable */
      /* Generating integer random numbers */
      /* Pay attention to word size needed to
         store only random number */
      for( i = 0; i < n; i++ )
      {
              r[i] = …
      }
      /* Update stream state */
      …
} /* iMyBrng */
```

注：64 ビットと 128 ビットの生成器を使用する場合、擬似乱数ベクトル *r* に値を正しく格納するために桁容量を考慮すること。たとえば、1 個の 64 ビット値は *r* の 2 個の要素を必要とし、1 番目の要素に下位 32 ビット、2 番目の要素に上位 32 ビットが格納される。同様に 128 ビット値では 4 個の *r* 要素が使われる。

sBRng

FORTRAN:

```
SUBROUTINE smybrng( stream, n, r, a, b )
      TYPE(MYSTREAM_STATE), INTENT(INOUT):: stream
      INTEGER, INTENT(IN)    :: n
      REAL, DIMENSION(n), INTENT(OUT)    :: r
      REAL, INTENT(IN)    :: a
      REAL, INTENT(IN)    :: b
! Generating real (a,b) random numbers
      DO i = 1, n
             R(I) = …
      END DO
! Update stream state
END SUBROUTINE smybrng
```

C:

```
void sMyBrng( VSLStreamStatePtr stream, int n, float r[],
              float a, float b )
{
       int    i;     /* Loop variable */
      /* Generating float (a,b) random numbers */
      for ( i = 0; i < n; i++ )
      {
             r[i] = …
      }
      /* Update stream state */
      …
} /* sMyBrng */
```

dBRng

FORTRAN:

```
SUBROUTINE dmybrng( stream, n, r, a, b )
      TYPE(MYSTREAM_STATE), INTENT(INOUT) :: stream
      INTEGER, INTENT(IN)     :: n
      DOUBLE PRECISION, DIMENSION(n), INTENT(OUT)  :: r
      REAL, INTENT(IN)  :: a
      REAL, INTENT(IN)  :: b
! Generating double precision (a,b) random numbers
      DO i = 1, n
             R(I) = …
      END DO
! Update stream state
       …
END SUBROUTINE dmybrng
```

C:

```
void dMyBrng( VSLStreamStatePtr stream, int n, double r[],
              double a, double b )
{
       int   i;    /* Loop variable */
      /* Generating double (a,b) random numbers */
      for ( i = 0; i < n; i++ )
      {
              r[i] = …
      }
      /* Update stream state */
      …
} /* dMyBrng */
```

アドバンスト DFT 関数

9

離散フーリエ変換 (DFT) を計算する高速フーリエ変換 (FFT) アルゴリズムは、多くの科学技術計算分野で不可欠なツールとなっている。本章ではインテル® マス・カーネル・ライブラリに新たに追加された DFT 関数を説明する。これら関数は、使い勝手のよいアプリケーション・プログラマ・インターフェイス (API) として実装され、FFT によって高速に DFT 計算を実行する。

インテル・マス・カーネル・ライブラリの離散フーリエ変換関数ライブラリは、一次元、二次元、多次元 (最大 7 次) のルーチンを提供する。

Fortran インターフェイスと C インターフェイスの両方がすべての変換関数に用意されている。

インテル・マス・カーネル・ライブラリは本マニュアルの 3 章で述べている FFT インターフェイスを引き続きサポートするが、従来のユーザ・アプリケーションは DFT 新関数に移行するのが望ましい。DFT ルーチンは従来の FFT ルーチンと異なり、最大 7 次元の変換と 2 の累乗混合基数以外の変換長をサポートする。

インテル・マス・カーネル・ライブラリに実装されているすべての DFT 関数を次の表に示す。

表 9-1　インテル® マス・カーネル・ライブラリの DFT 関数

関数名	演算
ディスクリプタ操作関数	
DftiCreateDescriptor	ディスクリプタ・データ構造にメモリを割り当て、デフォルト構成設定を用いて具体化する。
DftiCommitDescriptor	実際の DFT 演算を実行可能にするすべての初期化を実行する。
DftiCopyDescriptor	既存のディスクリプタをコピーする。
DftiFreeDescriptor	ディスクリプタに割り当てられていたメモリを解放する。

表 9-1　インテル® マス・カーネル・ライブラリの DFT 関数 (続き)

関数名	演算
DFT 計算関数	
DftiComputeForward	順方向 DFT を計算する。
DftiComputeBackward	逆方向 DFT を計算する。
ディスクリプタ構成関数	
DftiSetValue	特定の 1 個の構成パラメータを指定された構成値で設定する。
DftiGetValue	特定の 1 個の構成パラメータの構成値を取得する。
ステータス確認関数	
DftiErrorClass	ステータスが定義済みクラスのエラーを反映しているか確認する。
DftiErrorMessage	エラー・メッセージを生成する。

DFT 関数の説明に続いて、構成設定 (「構成設定」を参照) および使用されるさまざまな構成パラメータを説明する。

DFT の計算

本章で後述する DFT 関数は Fortran インターフェイスと C インターフェイスで実装される。Fortran とは Fortran 95 を意味する。DFT インターフェイスは、Fortran 77 にはない Fortran 95 のさまざまな新機能に強く依存している。

注：Fortran では明示的な関数インターフェイスに続くデータ配列は、どの変換タイプに対しても一次元として定義されなければならない。

本章で示される資料では、C のネイティブ複素タイプの有効性は C9X で規定されているものと仮定している。

それぞれの関数を説明する前に使用法の例を示す。

一般的な状況では 1 回の DFT 計算は 4 個の関数コールで成立すると考えられる。この例では一次元の計算を 2 回行っている。

例 9-1　一次元 DFT (Fortran インターフェイス)

```
! Fortran example.
! 1D complex to complex, and real to conjugate even
Use MKL_DFTI
Complex :: X(32)
Real :: Y(34)
type(DFTI_DESCRIPTOR), POINTER :: My_Desc1_Handle, My_Desc2_Handle
Integer :: Status
...put input data into X(1),...,X(32); Y(1),...,Y(32)

! Perform a complex to complex transform
Status = DftiCreateDescriptor( My_Desc1_Handle, DFTI_SINGLE,
          DFTI_COMPLEX, 1, 32 )
Status = DftiCommitDescriptor( My_Desc1_Handle )
Status = DftiComputeForward( My_Desc1_Handle, X )
Status = DftiFreeDescriptor(My_Desc1_Handle)
! result is given by {X(1),X(2),...,X(32)}

! Perform a real to complex conjugate even transform
Status = DftiCreateDescriptor(My_Desc2_Handle, DFTI_SINGLE,
          DFTI_REAL, 1, 32)
Status = DftiCommitDescriptor(My_Desc2_Handle)
Status = DftiComputeForward(My_Desc2_Handle, Y)
Status = DftiFreeDescriptor(My_Desc2_Handle)
! result is given by {Y(1)+iY(2), Y(3)+iY(4), ..., Y(33)+iY(34),
! Y(31)-iY(32), Y(29)-iY(30), ..., Y(3)-iY(4).
```

例 9-2　一次元 DFT (C インターフェイス)

```
/* C example, float _Complex is defined in C9X */
#include "mkl_dfti.h"
float _Complex x[32];
float y[34];
DFTI_DESCRIPTOR *my_desc1_handle, *my_desc2_handle;
/* .... or alternatively
DFTI_DESCRIPTOR_HANDLE my_desc1_handle, my_desc2_handle; */

long status;
...put input data into x[0],...,x[31]; y[0],...,y[31]
status = DftiCreateDescriptor( &my_desc1_handle, DFTI_SINGLE,
          DFTI_COMPLEX, 1, 32);
status = DftiCommitDescriptor( my_desc1_handle );
status = DftiComputeForward( my_desc1_handle, x);
status = DftiFreeDescriptor(&my_desc1_handle);
/* result is x[0], ..., x[31] */
status = DftiCreateDescriptor( &my_desc2_handle, DFTI_SINGLE,
          DFTI_REAL, 1, 32);
status = DftiCommitDescriptor( my_desc2_handle);
status = DftiComputeForward( my_desc2_handle, y);
status = DftiFreeDescriptor(&my_desc2_handle);
/* y[0]+iy[1], ..., y[32]+iy[33], y[30]-iy[31], ..., y[2]-iy[3] */
```

次に示すのは単純な二次元変換を 2 回行った例である。計算関数のデータと結果パラメータはすべて擬寸法階数 1 の配列 `DIMENSION(0:*)` として宣言されている点に注意する。そのため、`EQUVALENCE` ステートメントか Fortran のその他の手段を用いて二次元配列を一次元に変換する必要がある。

例 9-3　二次元 DFT (Fortran インターフェイス)

```
! Fortran example.
! 2D complex to complex, and real to conjugate even
Use MKL_DFTI
Complex ::  X_2D(32,100)
Real :: Y_2D(34, 102)
Complex ::  X(3200)
Real :: Y(3468)
Equivalence (X_2D, X)
Equivalence (Y_2D, Y)
type(DFTI_DESCRIPTOR), POINTER :: My_Desc1_Handle, My_Desc2_Handle
Integer :: Status, L(2)
...put input data into X_2D(j,k), Y_2D(j,k), 1<=j=32,1<=k<=100
...set L(1) = 32, L(2) = 100
...the transform is a 32-by-100

! Perform a complex to complex transform
Status = DftiCreateDescriptor( My_Desc1_Handle, DFTI_SINGLE,
          DFTI_COMPLEX, 2, L)
Status = DftiCommitDescriptor( My_Desc1_Handle)
Status = DftiComputeForward( My_Desc1_Handle, X)
Status = DftiFreeDescriptor(My_Desc1_Handle)
! result is given by X_2D(j,k), 1<=j<=32, 1<=k<=100

! Perform a real to complex conjugate even transform
Status = DftiCreateDescriptor( My_Desc2_Handle, DFTI_SINGLE,
          DFTI_REAL, 2, L)
Status = DftiCommitDescriptor( My_Desc2_Handle)
Status = DftiComputeForward( My_Desc2_Handle, Y)
Status = DftiFreeDescriptor(My_Desc2_Handle)
! result is given by the complex value z(j,k) 1<=j<=32; 1<=k<=100
where
! z(j,k) = Y_2D(2j-1,k) + iY_2D(2j,k) 1<=j<=17; 1<=k<=100
! z(j,k) = Y_2D(2(34-j)-1,k) - iY_2D(2(34-j),k) 18<=j<=32; 1<=k<=100
```

例 9-4 二次元 DFT (C インターフェイス)

```
/* C example */
#include "mkl_dfti.h"
float _Complex x[32][100];
float y[34][102];
DFTI_DESCRIPTOR_HANDLE my_desc1_handle, my_desc2_handle;
/* or alternatively
DFTI_DESCRIPTOR *my_desc1_handle, *my_desc2_handle; */
long status, l[2];
...put input data into x[j][k] 0<=j<=31, 0<=k<=99
...put input data into y[j][k] 0<=j<=31, 0<=k<=99
l[0] = 32; l[1] = 100;
status = DftiCreateDescriptor( &my_desc1_handle, DFTI_SINGLE,
          DFTI_COMPLEX, 2, l);
status = DftiCommitDescriptor( my_desc1_handle);
status = DftiComputeForward( my_desc1_handle, x);
status = DftiFreeDescriptor(&my_desc1_handle);
/* result is the complex value x[j][k], 0<=j<=31, 0<=k<=99 */
status = DftiCreateDescriptor( &my_desc2_handle, DFTI_SINGLE,
          DFTI_REAL, 2, l);
status = DftiCommitDescriptor( my_desc2_handle);
status = DftiComputeForward( my_desc2_handle, y);
status = DftiFreeDescriptor(&my_desc2_handle);
/* result is the complex value z(j,k) 0<=j<=31; 0<=k<=99
/* z(j,k) = y[2j][k] + iy[2j+1][k] 0<=j<=16; 0<=k<=99 */
/* z(j,k) = y[2(32-j)][k] - iy[2(32-j)+1][k] 17<=j<=31; 1<=k<=100 */
```

生成された DFTI_DESCRIPTOR のレコード・タイプは計算される DFT の長さと領域の情報を含んでいる。さらに 構成パラメータの多くの設定も含んでいる。上の例ではこれらすべてのパラメータに対してデフォルト設定を使用しており、たとえば次のような項目である。

- 計算される DFT は倍率因子を持っていない

- 変換されるデータは 1 セットのみ
- データはメモリに連続して格納
- 順方向変換は、$e^{-i2\pi jk/n}$ ではなく $e^{+i2\pi jk/n}$ を用いた式として定義される
- 複素データはネイティブ複素データ・タイプに格納される
- 計算結果は入力データに上書き (イン・プレース) される、ほか

これらのデフォルト設定が適当でない場合は、DftiSetValue 関数を用いて 1 つずつ変更できる。たとえば入力データを DFT 計算後にも保全しておくには、6 番目の設定をデフォルトの「イン・プレース」から「ノット・イン・プレース」に変更する必要がある。以下にコードの例を示す。

例 9-5 デフォルト設定の変更 (Fortran)

```
! Fortran example
! 1D complex to complex, not in place
Use MKL_DFTI
Complex :: X_in(32), X_out(32)
type(DFTI_DESCRIPTOR), POINTER :: My_Desc_Handle
Integer :: Status
...put input data into X_in(j), 1<=j<=32
Status = DftiCreateDescriptor( My_Desc_Handle, DFTI_SINGLE,
          DFTI_COMPLEX, 1, 32)
Status = DftiSetValue( My_Desc_Handle, DFTI_PLACEMENT,
DFTI_NOT_INPLACE)
Status = DftiCommitDescriptor( My_Desc_Handle)
Status = DftiComputeForward( My_Desc_Handle, X_in, X_out)
Status = DftiFreeDescriptor (My_Desc_Handle)
! result is X_out(1),X_out(2),...,X_out(32)
```

例 9-6 デフォルト設定の変更 (C)

```
/* C example */
#include "mkl_dfti.h"
float  _Complex x_in[32], x_out[32];
DFTI_DESCRIPTOR_HANDLE my_desc_handle;
/* or alternatively
DFTI_DESCRIPTOR *my_desc_handle; */
long status;
...put input data into x_in[j], 0 <= j < 32
status = DftiCreateDescriptor( &my_desc_handle, DFTI_SINGLE,
DFTI_COMPLEX, 1, 32);
status = DftiSetValue( my_desc_handle, DFTI_PLACEMENT,
DFTI_NOT_INPLACE);
status = DftiCommitDescriptor( my_desc_handle);
status = DftiComputeForward( my_desc_handle, x_in, x_out);
status = DftiFreeDescriptor(&my_desc_handle);
/* result is x_out[0], x_out[1], ..., x_out[31] */
```

インテル・マス・カーネル・ライブラリでは、1 つの単一データ構造、すなわちディスクリプタを使用して、パラメータを個別に変更できるようにした自由度の高い手法を DFT 計算に適用している。これによって機能の拡張性と使いやすさを実現した。

DFT インターフェイス

アドバンスト DFT 関数を使用するには、Fortran では "use" ステートメントを使用して MKL_DFTI モジュールをアクセスする必要がある。C では "include" ステートメントを使ってヘッダ・ファイル mkl_dfti.h をアクセスする必要がある。

Fortran インターフェイスでは、派生型 DFTI_DESCRIPTOR、さまざまな構成パラメータの名称とそれらの取り得る値を表す名前付き定数、Fortran 95 の汎用機能性を介したさまざまなオーバーロード関数が与えられる。

C インターフェイスでは、構造タイプ DFTI_DESCRIPTOR、マクロ定義

#define DFTI_DESCRIPTOR_HANDLE DFTI_DESCRIPTOR は、

DFTI_CONFIG_PARAM と DFTI_CONFIG_VALUE の 2 個の列挙型の名前付き定数、関数によっては異なる個数の入力引数を受け付ける種々の関数が与えられる。

注：現時点のライブラリ実装では、本章の続くセクションで記載されている関数あるいは機能の一部がサポートされていない場合がある。実装に依存して除外される項目については、使用しているライブラリ・バージョンのリリース・ノートを参照のこと。

インテル・マス・カーネル・ライブラリでは DFT 関数を 4 種類の大きなカテゴリーに分けている。

1. **ディスクリプタ操作**。このカテゴリーは 4 個の関数で構成される。1 つ目は DFT ディスクリプタを生成しストレージを動的に割り当てる関数である。この関数はユーザが与える入力値によってディスクリプタをデフォルト設定に構成する。
   2 番目はディスクリプタを設定にもとづいて「遂行」させる関数である。実際には、すべての必要な事前計算が実行されることを意味する。この事前計算には、入力長さの因数分解と必要なすべての回転因子の計算が含まれる。3 番目はディスクリプタのコピーを作成する関数、4 つ目はディスクリプタ情報に割り当てられていたすべてのメモリを解放する関数である。

2. **DFT 計算**。このカテゴリーは 2 個の関数で構成される。1 つ目は順方向 DFT 計算を実行する関数で、2 つ目は逆方向 DFT 計算を実行する関数である。
3. **ディスクリプタ構成**。このカテゴリーは 2 個の関数で構成される。1 つ目の関数は特定の 1 個の値を、多くの構成パラメータのうち変更可能な (わずかにあるか、まったくない) パラメータの 1 つに設定する。2 つ目の関数は、これら構成パラメータの任意の 1 つの現在値を取得する (すべてが読み出し可能)。読み出し可能は、パラメータ数は多いが扱えるのは 1 回の関数呼び出しにつき 1 個である。
4. **ステータス確認**。上記の 3 種類のカテゴリーに記述されている各関数は演算のステータスを表す整数値を返す。
戻り値が非ゼロの場合はある種の問題が起きたことを意味する。
インテル・マス・カーネル・ライブラリでは将来のリリースにおける拡張を想定して、現在の DFT インターフェイスは単一の論理ステータス・クラス関数および単純なステータス・メッセージ生成関数のみを提供している。

ステータス確認関数

ディスクリプタ操作、DFT 計算、ディスクリプタ構成のカテゴリーに含まれる各関数は、演算のステータスを示す整数値を返す。ここで述べる 2 個の関数は、このステータスを確認する関数である。1 つ目の関数は論理関数で定義済みクラスのエラーをステータスが反映しているか確認し、2 つ目の関数はエラー・メッセージ関数で文字列を返す。

ErrorClass

ステータスが定義済みクラスのエラーを
反映しているか確認する。

使用方法

```
! Fortran
Predicate = DftiErrorClass( Status, Error_Class )
/* C */
predicate = DftiErrorClass( status, error_class );
```

説明

インテル・マス・カーネル・ライブラリの DFT インターフェイスには表 9-2 に示す定義済みエラー・クラスが与えられている。これらは名前付き定数で、Fortran では INTEGER タイプ、C では long タイプである。

表 9-2　定義済みエラー・クラス

名前付き定数	意味
DFTI_NO_ERROR	エラーなし
DFTI_MEMORY_ERROR	通常、メモリ割り当てに関係する
DFTI_INVALID_CONFIGURATION	構成パラメータの 1 つ以上が無効な設定
DFTI_INCONSISTENT_CONFIGURATION	構成パラメータまたは入力パラメータが不整合
DFTI_MULTITHREADED_ERROR	通常、OMP ルーチンのエラー戻り値に関係する
DFTI_BAD_DESCRIPTOR	ディスクリプタを計算に使用できない
DFTI_UNIMPLEMENTED	正当な設定が未実装、実装依存
DFTI_MKL_INTERNAL_ERROR	ライブラリ内部エラー

定義済みエラー・クラスの確認では、指定したエラー・クラスに対して DFTI_ERROR_CLASS を適用して、その結果ステータス戻り値が .TRUE. または .FALSE. かを確認することが正しい使用方法である点に注意する。ステータスと定義済みクラスの直接比較は使用方法としては適切ではない。

例 9-7 ステータス確認関数の使用

```
from C language:

DFTI_DESCRIPTOR_HANDLE desc;
long status, class_error, value;
char* error_message;
 . . . descriptor creation and other code
status = DftiGetValue( desc, DFTI_PRECISION, &value); //
//or any DFTI function

class_error = DftiErrorClass(status, DFTI_ERROR_CLASS);
if (! class_error) {
printf ("status is not a member of Predefined Error
Class\n");
} else {
error_message = DftiErrorMessage(status);
printf("error_message = %s \n", error_message);
}
. . .
from Fortran:

type(DFTI_DESCRIPTOR), POINTER :: desc
integer value, status
character(DFTI_MAX_MESSAGE_LENGTH) error_message
logical class_error
 . . . descriptor creation and other code
status = DftiGetValue( desc, DFTI_PRECISION, value)

class_error = DftiErrorClass(status, DFTI_ERROR_CLASS)
if (.not. class_error) then
print *, 'status is not a member of Predefined Error Class '
else
error_message  = DftiErrorMessage(status)
print *, 'error_message = ', error_message
endif
```

インターフェイスとプロトタイプ

```
//Fortran interface
INTERFACE DftiErrorClass
//Note that the body provided here is to illustrate the different
```

```
//argument list and types of dummy arguments. The interface
//does not guarantee what the actual function names are.
//Users can only rely on the function name following the
//keyword INTERFACE
 FUNCTION some_actual_function_8( Status, Error_Class )
   LOGICAL some_actual_function_8
   INTEGER, INTENT(IN) :: Status, Error_Class
 END FUNCTION some_actual_function_8
END INTERFACE DftiErrorClass

/* C prototype */
long DftiErrorClass( long , long );
```

ErrorMessage

エラー・メッセージを生成する。

使用方法

```
! Fortran
ERROR_MESSAGE = DftiErrorMessage( Status )
/* C */
error_message = DftiErrorMessage( status );
```

説明

このエラー・メッセージ関数はエラー・メッセージ文字列を生成する。文字列の最大長さは Fortran では名前付き定数 DFTI_MAX_MESSAGE_LENGTH で与えられる。実際のエラー・メッセージは実装依存である。Fortran の場合、ユーザは DFTI_MAX_MESSAGE_LENGTH を最大長とした文字列を使用する。C ではこの関数は文字列へのポインタを返す。すなわち
デリミタ '0' を持つ文字配列である。

例 9-7 にこの関数の実装例を示す。

インターフェイスとプロトタイプ

```
//Fortran interface
INTERFACE DftiErrorMessage
//Note that the body provided here is to illustrate the different
//argument list and types of dummy arguments. The interface
//does not guarantee what the actual function names are.
//Users can only rely on the function name following the
//keyword INTERFACE
 FUNCTION some_actual_function_9( Status, Error_Class )
  CHARACTER(LEN=DFTI_MAX_MESSAGE_LENGTH) some_actual_function_9( Status )
  INTEGER, INTENT(IN) :: Status
 END FUNCTION some_actual_function_9
END INTERFACE DftiErrorMessage

/* C prototype */
char *DftiErrorMessage( long );
```

ディスクリプタ操作

このカテゴリーは、ディスクリプタの生成、ディスクリプタの遂行、ディスクリプタのコピー、ディスクリプタの解放、の 4 個の関数で構成される。

CreateDescriptor

ディスクリプタ・データ構造にメモリを割り当て、デフォルト構成設定で具体化する。

使用方法

```
! Fortran
Status = DftiCreateDescriptor( Desc_Handle,            &
                               Precision,              &
                               Forward_Domain,         &
                               Dimension,              &
                               Length )

/* C */
status = DftiCreateDescriptor( &desc_handle,
                               precision,
                               forward_domain,
                               dimension,
                               length );
```

説明

この関数はディスクリプタ・データ構造にメモリを割り当て、変換対象の精度、領域、次元、長さをそれぞれのデフォルト構成設定を使って具体化する。領域は順方向変換での領域として解釈される。メモリは動的に割り当てられるため、出力結果として生成されたディスクリプタのポインタが得られる。この関数は、DFT を計算する従来型の多くのソフトウェア・パッケージやライブラリに見られる「初期化」ルーチンとはやや異なっている。ユーザの要求によって値設定関数 DftiSetValue を通してデフォルト構成設定を変更できることから、この関数は回転因子計算のような計算量の多い計算はおそらく実行しない。

精度と (順方向) 領域は DFT インターフェイスで構成値に与えられる名前付き定数によって指定される。精度の選択肢は DFTI_SINGLE か DFTI_DOUBLE である。(順方向) 領域の選択肢は DFTI_COMPLEX、DFTI_REAL、または DFTI_CONJUGATE_EVEN である。表 9-5 に構成値に対する名前付き定数の全リストを示す。

次元は変換の次元を表す単純な正の整数である。長さは、一次元変換では単純な正の整数、多次元変換では長さ次元に対応した正の整数を含む整数配列 (C ではポインタ) である。

この関数は正常に終了すると DFTI_NO_ERROR を返す。戻り値ステータスの詳細は「ステータス確認関数」を参照のこと。

インターフェイスとプロトタイプ

```
!Fortran interface.
INTERFACE DftiCreateDescriptor
!Note that the body provided here is to illustrate the different
!argument list and types of dummy arguments. The interface
!does not guarantee what the actual function names are.
!Users can only rely on the function name following the
 !keyword INTERFACE
 FUNCTION some_actual_function_1D( Desc_Handle, Prec, Dom, Dim, Length )
    INTEGER :: some_actual_function_1D
    TYPE(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
    INTEGER, INTENT(IN) :: Prec, Dom
    INTEGER, INTENT(IN) :: Dim, Length
 END FUNCTION some_actual_function_1D

 FUNCTION some_actual_function_HIGHD( Desc_Handle, Prec, Dom, Dim, Length )
    INTEGER :: some_actual_function_HIGHD
    TYPE(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
    INTEGER, INTENT(IN) :: Prec, Dom
    INTEGER, INTENT(IN) :: Dim, Length(*)
 END FUNCTION some_actual_function_HIGHD
END INTERFACE DftiCreateDescriptor
```

関数はオーバーロードであるとともに、長さの実際の引数はスカラーまたは階数 1 配列を取り得る点に注意する。

```
/* C prototype */
long DftiCreateDescriptor( DFTI_DESCRIPTOR_HANDLE *,
                           DFTI_CONFIG_PARAM ,
                           DFTI_CONFIG_PARAM ,
                           long ,
                           ... );
```

可変引数機能は、スカラー (`long`) または配列 (`long *`) で表現される長さの引数に対処するために使用される。

CommitDescriptor

実際の DFT 計算を実現可能にする
すべての具体化を実行する。

使用方法

```
! Fortran
Status = DftiCommitDescriptor( Desc_Handle )
/* C */
status = DftiCommitDescriptor( desc_handle );
```

説明

DFT インターフェイスでは、ディスクリプタを使って DFT 計算を実行する前に、生成されたディスクリプタを遂行する関数を呼び出す必要がある。通常この遂行処理では、実際の DFT 計算を実行可能にするすべての初期化が実行される。近代的な実装では、高度に効率化された計算方法を検索するために、さまざまな長さの入力を用いた因数分解を行う調査が含まれる場合がある。

遂行されたディスクリプタの構成パラメータを値設定関数 (「ディスクリプタの構成」を参照) を介して変更した場合は、計算関数を呼び出す前にディスクリプタの再遂行を必要とする。通常は、この遂行関数コールのすぐあとに計算関数コールを続ける (「DFT の計算」を参照)。

この関数は正常に終了すると DFTI_NO_ERROR を返す。戻り値ステータスの詳細は「ステータス確認関数」を参照のこと。

インターフェイスとプロトタイプ

```
! Fortran interface
INTERFACE DftiCommitDescriptor
!Note that the body provided here is to illustrate the different
!argument list and types of dummy arguments. The interface
!does not guarantee what the actual function names are.
!Users can only rely on the function name following the
!keyword INTERFACE
 FUNCTION some_actual function_1 ( Desc_Handle )
    INTEGER :: some_actual function_1
    TYPE(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
 END FUNCTION some_actual function_1
END INTERFACE DftiCommitDescriptor

/* C prototype */
long DftiCommitDescriptor( DFTI_DESCRIPTOR_HANDLE );
```

CopyDescriptor

既存のディスクリプタをコピーする。

使用方法

```
! Fortran
Status = DftiCopyDescriptor( Desc_Handle_Original,
                             Desc_Handle_Copy )
/* C */
status = DftiCopyDescriptor( desc_handle_original,
                             &desc_handle_copy );
```

説明

この関数は既存ディスクリプタのコピーを作成し、それに対するポインタを与える。この関数の目的は、ディスクリプタ解放関数 DftiFreeDescriptor によって元のディスクリプタが破壊された場合でも、元のディスクリプタの全情報を維持するためである。

この関数は正常に終了すると DFTI_NO_ERROR を返す。戻り値ステータスの詳細は「ステータス確認関数」を参照のこと。

インターフェイスとプロトタイプ

```
! Fortran interface
INTERFACE DftiCopyDescriptor
! Note that the body provided here is to illustrate the different
!argument list and types of dummy arguments. The interface
!does not guarantee what the actual function names are.
!Users can only rely on the function name following the
!keyword INTERFACE
 FUNCTION some_actual_function_2( Desc_Handle_Original,
                                          Desc_Handle_Copy )
    INTEGER :: some_actual_function_2
    TYPE(DFTI_DESCRIPTOR), POINTER :: Desc_Handle_Original,
Desc_Handle_Copy
 END FUNCTION some_actual_function_2
END INTERFACE DftiCopyDescriptor

/* C prototype */
long DftiCopyDescriptor( DFTI_DESCRIPTOR_HANLDE,
DFTI_DESCRIPTOR_HANDLE * );
```

FreeDescriptor

ディスクリプタに割り当てられたメモリを解放する。

使用方法

```
! Fortran
Status = DftiFreeDescriptor( Desc_Handle )
/* C */
status = DftiFreeDescriptor( &desc_handle );
```

説明

この関数はディスクリプタに割り当てられているすべてのメモリを解放する。

この関数は正常に終了すると DFTI_NO_ERROR を返す。戻り値ステータスの詳細は「ステータス確認関数」を参照のこと。

インターフェイスとプロトタイプ

```
! Fortran interface
INTERFACE DftiFreeDescriptor
//Note that the body provided here is to illustrate the different
//argument list and types of dummy arguments. The interface
//does not guarantee what the actual function names are.
//Users can only rely on the function name following the
//keyword INTERFACE
 FUNCTION some_actual_function_3( Desc_Handle )
    INTEGER :: some_actual_function_3
    TYPE(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
 END FUNCTION some_actual_function_3
END INTERFACE DftiFreeDescriptor

/* C prototype */
long DftiFreeDescriptor( DFTI_DESCRIPTOR_HANDLE * );
```

DFT ComputationDFT の計算

このカテゴリーは、順方向変換の計算、逆方向変換の計算の 2 つの関数で構成される。

ComputeForward

順方向 DFT を計算する。

使用方法

```
! Fortran
Status = DftiComputeForward( Desc_Handle, X_inout )
Status = DftiComputeForward( Desc_Handle, X_in, X_out )
Status = DftiComputeForward( Desc_Handle, X_inout, Y_inout )
Status = DftiComputeForward( Desc_Handle, X_in, Y_in, X_out, Y_out )
/* C */
status = DftiComputeForward( desc_handle, x_inout );
status = DftiComputeForward( desc_handle, x_in, x_out );
status = DftiComputeForward( desc_handle, x_inout, y_inout );
status = DftiComputeForward( desc_handle, x_in, y_in, x_out, y_out );
```

説明

実際の DFT 計算はディスクリプタの生成と遂行を完了した後に実行する。`DftiComputeForward` 関数は順方向 DFT を計算する。デフォルトは因子 $e^{-i2\pi/n}$ を用いた変換である (正符号の付いたものの代わりに)。構成の自由度が高いため、入力データを複数の方法で表すことができ、また出力結果を複数の方法で配置できる。そのため、入力パラメータの個数とタイプは多様である。このような多様性は Fortran 95 の一般関数機能によって実現される。データと結果パラメータは、すべて擬寸法階数 1 の配列 `DIMENSION(0:*)` として宣言される。

この関数は正常に終了すると `DFTI_NO_ERROR` を返す。戻り値ステータスの詳細は「ステータス確認関数」を参照のこと。

インターフェイスとプロトタイプ

```
//Fortran interface.
INTERFACE DftiComputeFoward
//Note that the body provided here is to illustrate the different
//argument list and types of dummy arguments. The interface
//does not guarantee what the actual function names are.
//Users can only rely on the function name following the
//keyword INTERFACE
 // One argument single precision complex
 FUNCTION some_actual_function_4_C( Desc_Handle, X )
    INTEGER :: some_actual_function_4_C
    TYPE(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
    COMPLEX, INTENT(INOUT) :: X(*)
 END FUNCTION some_actual_function_4_C
 // One argument double precision complex
 FUNCTION some_actual_function_4_Z( Desc_Handle, X )
    INTEGER :: some_actual_function_4_Z
    TYPE(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
    COMPLEX (Kind((0D0,0D0))), INTENT(INOUT) :: X(*)
 END FUNCTION some_actual_function_4_Z
 // One argument single precision real
 FUNCTION some_actual_function_4_R( Desc_Handle, X )
    INTEGER :: some_actual_function_4_R
    TYPE(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
    REAL, INTENT(INOUT) :: X(*)
 END FUNCTION some_actual_function_4_R
 // One argument double precision real
 ...
 // Two argument single precision complex
 ...
 ...
 // Four argument double precision real
 FUNCTION some_actual_function_4_DDDD( Desc_Handle, X1_In, X2_In,
                                       Y1_Out, Y2_Out )
```

```
    INTEGER :: some_actual_function_4_DDDD
    TYPE(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
    REAL (Kind(0D0)), INTENT(IN) :: X1_In(*), X2_In(*)
    REAL (Kind(0D0)), INTENT(OUT) :: Y1_Out(*), Y2_Out(*)
  END FUNCTION some_actual_function_4_DDDD
END INTERFACE DftiComputeFoward

/* C prototype */
long DftiComputeForward( DFTI_DESCRIPTOR_HANDLE,
                         void *,
                         ... );
```

DFT インターフェイスはその実装で、規則的な「ストライド」パターンで線形にメモリに格納されるものとしてデータが扱われることを前提としている (詳細は「ストライド」、[3] を参照のこと)。関数は最初の要素の開始アドレスを期待する。そのため、

ディスクリプタ単体は、引数の個数と存在すべきタイプを正確に決めるために必要な充分な情報を含んでいる。実際の実装では、この情報を使って入力の矛盾を確認することがある。

ComputeBackward

逆方向 DFT を計算する。

使用方法

```
! Fortran
Status = DftiComputeBackward( Desc_Handle, X_inout )
Status = DftiComputeBackward( Desc_Handle, X_in, X_out )
Status = DftiComputeBackward( Desc_Handle, X_inout, Y_inout )
Status = DftiComputeBackward( Desc_Handle, X_in, Y_in, X_out, Y_out )
/* C */
status = DftiComputeBackward( desc_handle, x_inout );
status = DftiComputeBackward( desc_handle, x_in, x_out );
status = DftiComputeBackward( desc_handle, x_inout, y_inout );
status = DftiComputeBackward( desc_handle, x_in, y_in, x_out, y_out );
```

説明

実際の DFT 計算はディスクリプタの生成と遂行を完了した後に実行する。DftiComputeBackward 関数は逆方向 DFT を計算する。デフォルトは因子 $e^{i2\pi/n}$ を用いた変換である (負符号の付いたものの代わりに)。構成の自由度が高いため、入力データを複数の方法で表すことができ、また出力結果を複数の方法で配置できる。そのため、入力パラメータの個数とタイプは多様である。このような多様性は Fortran 95 の一般関数機能によって実現される。データと結果パラメータは、すべて擬寸法階数 1 の配列 DIMENSION(0:*) として宣言される。

この関数は正常に終了すると DFTI_NO_ERROR を返す。戻り値ステータスの詳細は「ステータス確認関数」を参照のこと。

インターフェイスとプロトタイプ

```
//Fortran interface.
INTERFACE DftiComputeBackward
//Note that the body provided here is to illustrate the different
//argument list and types of dummy arguments. The interface
//does not guarantee what the actual function names are.
//Users can only rely on the function name following the
//keyword INTERFACE
 // One argument single precision complex
 FUNCTION some_actual_function_5_C( Desc_Handle, X )
    INTEGER :: some_actual_function_5_C
    TYPE(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
    COMPLEX, INTENT(INOUT) :: X(*)
 END FUNCTION some_actual_function_5_C
 // One argument double precision complex
 FUNCTION some_actual_function_5_Z( Desc_Handle, X )
    INTEGER :: some_actual_function_5_Z
    TYPE(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
    COMPLEX (Kind((0D0,0D0))), INTENT(INOUT) :: X(*)
```

```
  END FUNCTION some_actual_function_5_Z
  // One argument single precision real
  FUNCTION some_actual_function_5_R( Desc_Handle, X )
     INTEGER :: some_actual_function_5_R
     TYPE(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
     REAL, INTENT(INOUT) :: X(*)
  END FUNCTION some_actual_function_5_R
  // One argument double precision real
  ...
  // Two argument single precision complex
  ...
  ...
  // Four argument double precision real
  FUNCTION some_actual_function_5_DDDD( Desc_Handle, X1_In, X2_In,
                                        Y1_Out, Y2_Out )
     INTEGER :: some_actual_function_5_DDDD
     TYPE(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
     REAL (Kind(0D0)), INTENT(IN) :: X1_In(*), X2_In(*)
     REAL (Kind(0D0)), INTENT(OUT) :: Y1_Out(*), Y2_Out(*)
  END FUNCTION some_actual_function_5_DDDD
END INTERFACE DftiComputeBackward


/* C prototype */
long DftiComputeBackward( DFTI_DESCRIPTOR_HANDLE,
                          void *,
                          ... );
```

DFT インターフェイスはその実装で、規則的な「ストライド」パターンで線形にメモリに格納されるものとしてデータが扱われることを前提としている (詳細は「ストライド」、[3] を参照のこと)。関数は最初の要素の開始アドレスを期待する。そのため、Fortran では擬寸法宣言を使用する。

ディスクリプタ単体は、引数の個数と存在すべきタイプを正確に決めるために必要な充分な情報を含んでいる。実際の実装では、この情報を使って入力の矛盾を確認する場合がある。

ディスクリプタの構成

このカテゴリーは、特定の 1 個の構成パラメータに適切な値を設定する値設定関数 DftiSetValue と、特定の 1 個の構成パラメータ値を読み出す値取得関数 DftiGetValue の 2 つで構成される。すべての構成パラメータは読み出し可能であるが、一部はユーザが設定できない。構成パラメータには、バージョン番号、動的な情報など、実装に固有な固定情報が含まれているが、固定情報は関数の実行時の実装によって求められている。

表 9-3　設定可能な構成パラメータ

名前付き定数	値のタイプ	意味
もっとも共通的な構成で、デフォルトはなく、明示的な設定が必要		
DFTI_PRECISION	名前付き定数	計算精度
DFTI_FORWARD_DOMAIN	名前付き定数	順方向変換の領域
DFTI_DIMENSION	整数スカラー	変換の次元
DFTI_LENGTHS	整数スカラー / 配列	各次元の長さ
複数の変換とデータ表現を含む共通的な構成		
DFTI_NUMBER_OF_TRANSFORMS	整数スカラー	変換の回数
DFTI_FORWARD_SIGN	名前付き定数	順方向変換の定義
DFTI_FORWARD_SCALE	浮動小数点スカラー	順方向変換の倍率因子
DFTI_BACKWARD_SCALE	浮動小数点スカラー	逆方向変換の倍率因子
DFTI_PLACEMENT	名前付き定数	計算結果の配置
DFTI_COMPLEX_STORAGE	名前付き定数	格納方法、複素領域データ
DFTI_REAL_STORAGE	名前付き定数	格納方法、実数領域データ
DFTI_CONJUGATE_EVEN_STORAGE	名前付き定数	格納方法、共役偶領域データ
DFTI_DESCRIPTOR_NAME	文字列	DFTI_MAX_NAME_LENGTH より短いこと
DFTI_PACKED_FORMAT	名前付き定数	パック形式、実数領域データ

表 9-3　設定可能な構成パラメータ (続き)

名前付き定数	値のタイプ	意味
データのストライドに関する構成		
DFTI_INPUT_DISTANCE	整数スカラー	複数変換、最初の要素の距離
DFTI_OUTPUT_DISTANCE	整数スカラー	複数変換、最初の要素の距離
DFTI_INPUT_STRIDES	整数配列	入力データのストライド情報
DFTI_OUTPUT_STRIDES	整数配列	出力データのストライド情報
その他の構成		
DFTI_INITIALIZATION_EFFORT	名前付き定数	計算方法の動的検索
DFTI_ORDERING	名前付き定数	データ順のスクランブル
DFTI_WORKSPACE	名前付き定数	補助ストレージを使用しない計算
DFTI_TRANSPOSE	名前付き定数	次元のスクランブル

これら各構成パラメータは MKL_DFTI モジュール内で名前付き定数によって区別される。C では、これら名前付き定数は列挙型 DFTI_CONFIG_PARAM を持つ。ユーザによって設定可能な構成パラメータのリストを表 9-3 に示す。構成パラメータのうち読み出し専用のリストを表 9-4 に示す。なお、すべてのパラメータは読み出し可能である。これらのパラメータのほとんどは自明であるが、残りについては関連した関数に詳細を記載している。

表 9-4　読み出し専用構成パラメータ

名前付き定数	値のタイプ	意味
DFTI_COMMIT_STATUS	名前付き定数	ディスクリプタが遂行されているかを示す
DFTI_VERSION	文字列	インテル・マス・カーネル・ライブラリのバージョン番号
DFTI_FORWARD_ORDERING	整数ポインタ	整数配列へのポインタ (「順序指定」を参照)
DFTI_BACKWARD_ORDERING	整数ポインタ	整数配列へのポインタ (「順序指定」を参照)

構成パラメータにはさまざまな値に設定される。これらの値の一部は、整数値 (たとえば同時変換要求数) や単精度値 (たとえば、順方向変換に適用したい倍率因子) のようなネイティブ・データ・タイプで規定される。

その他の構成値は性質において個別であり (たとえば、順方向変換の領域)、またそのため、DFTI モジュールでは名前付き定数として与えられる。C では、これらの名前付き定数は列挙型 DFTI_CONFIG_VALUE を持つ。この種の構成値で使用される名前付き定数の全リストを表 9-5 に示す。

表 9-5　名前付き定数構成値

名前付き定数	意味
DFTI_SINGLE	単精度
DFTI_DOUBLE	倍精度
DFTI_COMPLEX	複素領域
DFTI_REAL	実数領域
DFTI_CONJUGATE_EVEN	共役奇領域
DFTI_NEGATIVE	順方向変換を定義する符号
DFTI_POSITIVE	順方向変換を定義する符号
DFTI_INPLACE	出力は入力を上書き
DFTI_NOT_INPLACE	出力は入力を上書きしない
DFTI_COMPLEX_COMPLEX	格納方法 (「格納体系」参照)
DFTI_REAL_REAL	格納方法 (「格納体系」参照)
DFTI_COMPLEX_REAL	格納方法 (「格納体系」参照)
DFTI_REAL_COMPLEX	格納方法 (「格納体系」参照)
DFTI_HIGH	上位設定、初期化努力に関係する
DFTI_MEDIUM	中位設定、初期化努力に関係する
DFTI_LOW	下位設定、初期化努力に関係する
DFTI_COMMITTED	ディスクリプタの遂行ステータス
DFTI_UNCOMMITTED	ディスクリプタの遂行ステータス
DFTI_ORDERED	順方向領域および逆方向領域で順序どおりのデータ
DFTI_BACKWARD_SCRAMBLED	逆方向領域 (順方向変換による) でスクランブルされたデータ

表 9-5　名前付き定数構成値 (続き)

名前付き定数	意味
DFTI_FORWARD_SCRAMBLED	順方向領域 (逆方向変換による) でスクランブルされたデータ
DFTI_ALLOW	有用な場合に要求または利用を許可
DFTI_AVOID	実用的な場合に要求または利用を回避
DFTI_NONE	転置なしを指定
DFTI_CCS_FORMAT	パック形式、実数データ (「パックド形式」参照)
DFTI_PACK_FORMAT	パック形式、実数データ (「パックド形式」参照)
DFTI_PERM_FORMAT	パック形式、実数データ (「パックド形式」参照)
DFTI_VERSION_LENGTH	ライブラリ・バージョン長の文字数
DFTI_MAX_NAME_LENGTH	ディスクリプタ名の最大長
DFTI_MAX_MESSAGE_LENGTH	ステータス・メッセージの最大長

性質において個別なこれら構成パラメータで設定可能な値を表 9-6 に示す。

表 9-6　個別の構成パラメータの設定

名前付き定数	設定可能な値
DFTI_PRECISION	DFTI_SINGLE、または DFTI_DOUBLE (デフォルト値なし)
DFTI_FORWARD_DOMAIN	DFTI_COMPLEX、または DFTI_REAL、または DFTI_CONJUGATE_EVEN (デフォルト値なし)
DFTI_FORWARD_SIGN	DFTI_NEGATIVE (デフォルト)、または DFTI_POSITIVE
DFTI_PLACEMENT	DFTI_INPLACE (デフォルト) または DFTI_NOT_INPLACE
DFTI_COMPLEX_STORAGE	DFTI_COMPLEX_COMPLEX (デフォルト)、または DFTI_COMPLEX REAL、または DFTI_REAL_REAL
DFTI_REAL_STORAGE	DFTI_REAL_REAL (デフォルト)、または DFTI_REAL_COMPLEX
DFTI_CONJUGATE_EVEN_STORAGE	DFTI_COMPLEX_COMPLEX、または

表 9-6 個別の構成パラメータの設定 (続き)

名前付き定数	設定可能な値
DFTI_PACKED_FORMAT	DFTI_COMPLEX_REAL (デフォルト) 、または DFTI_REAL_REAL (1-D 変換のみ) DFTI_CCS_FORMAT (デフォルト)、または、 DFTI_PACK_FORMAT、または DFTI_PERM_FORMAT

表 9-7 に設定可能な構成パラメータのデフォルト値のリストを示す。

表 9-7 設定可能パラメータのデフォルト構成値

名前付き定数	デフォルト値
DFTI_NUMBER_OF_TRANSFORMS	1
DFTI_FORWARD_SIGN	DFTI_NEGATIVE
DFTI_FORWARD_SCALE	1.0
DFTI_BACKWARD_SCALE	1.0
DFTI_PLACEMENT	DFTI_INPLACE
DFTI_COMPLEX_STORAGE	DFTI_COMPLEX_COMPLEX
DFTI_REAL_STORAGE	DFTI_REAL_REAL
DFTI_CONJUGATE_EVEN_STORAGE	DFTI_COMPLEX_REAL
DFTI_PACKED_FORMAT	DFTI_CCS_FORMAT
DFTI_DESCRIPTOR_NAME	名前なし、長さゼロの文字列
DFTI_INPUT_DISTANCE	0
DFTI_OUTPUT_DISTANCE	0
DFTI_INPUT_STRIDES	次元、長さ、格納に従って厳密にパックされている
DFTI_OUTPUT_STRIDES	同上。詳細は「ストライド」参照
DFTI_INITIALIZATION_EFFORT	DFTI_MEDIUM
DFTI_ORDERING	DFTI_ORDERED
DFTI_WORKSPACE	DFTI_ALLOW
DFTI_TRANSPOSE	DFTI_NONE

SetValue

指定された構成値を用いて特定の
1 個の構成パラメータを設定する。

使用方法

```
! Fortran
Status = DftiSetValue( Desc_Handle,          &
                       Config_Param,         &
                       Config_Val )
/* C */
Status = DftiSetValue( desc_handle,
                       config_param,
                       config_val );
```

説明

この関数は特定の 1 個の構成パラメータを指定された構成値で設定する。構成パラメータは表 9-3 に示される名前付き定数のうちの 1 つであり、構成値は、対応する名前付き定数かネイティブ・タイプのどちらかの適切なタイプによって設定される。設定の詳細は「構成設定」を参照のこと。

この関数は正常に終了すると DFTI_NO_ERROR を返す。戻り値ステータスの詳細は「ステータス確認関数」を参照のこと。

インターフェイスとプロトタイプ

```
! Fortran interface
INTERFACE DftiSetValue
//Note that the body provided here is to illustrate the different
//argument list and types of dummy arguments. The interface
//does not guarantee what the actual function names are.
//Users can only rely on the function name following the
```

```
//keyword INTERFACE
 FUNCTION some_actual_function_6_INTVAL( Desc_Handle, Config_Param,
INTVAL )
    INTEGER :: some_actual_function_6_INTVAL
    Type(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
    INTEGER, INTENT(IN) :: Config_Param
    INTEGER, INTENT(IN) :: INTVAL
 END FUNCTION some_actual_function_6_INTVAL

 FUNCTION some_actual_function_6_SGLVAL( Desc_Handle, Config_Param,
SGLVAL )
    INTEGER :: some_actual_function_6_SGLVAL
    Type(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
    INTEGER, INTENT(IN) :: Config_Param
    REAL, INTENT(IN) :: SGLVAL
 END FUNCTION some_actual_function_6_SGLVAL

 FUNCTION some_actual_function_6_DBLVAL( Desc_Handle, Config_Param,
DBLVAL )
    INTEGER :: some_actual_function_6_DBLVAL
    Type(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
    INTEGER, INTENT(IN) :: Config_Param
    REAL (KIND(0D0)), INTENT(IN) :: DBLVAL
 END FUNCTION some_actual_function_6_DBLVAL

 FUNCTION some_actual_function_6_INTVEC( Desc_Handle, Config_Param,
INTVEC )
    INTEGER :: some_actual_function_6_INTVEC
    Type(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
    INTEGER, INTENT(IN) :: Config_Param
    INTEGER, INTENT(IN) :: INTVEC(*)
 END FUNCTION some_actual_function_6_INTVEC

 FUNCTION some_actual_function_6_CHARS( Desc_Handle, Config_Param,
CHARS )
    INTEGER :: some_actual_function_6_CHARS
    Type(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
```

```
    INTEGER, INTENT(IN) :: Config_Param
    CHARCTER(*), INTENT(IN) :: CHARS
  END FUNCTION some_actual_function_6_CHARS
END INTERFACE DftiSetValue


/* C prototype */
long DftiSetValue( DFTI_DESCRIPTOR_HANDLE,
                   DFTI_CONFIG_PARAM ,
                   ... );
```

GetValue

特定の 1 個の構成パラメータの構成値を取得する。

使用方法

```
! Fortran
Status = DftiGetValue( Desc_Handle,          &
                       Config_Param,         &
                       Config_Val )
/* C */
status = DftiGetValue( desc_handle,
                       config_param,
                       &config_val );
```

説明

この関数は特定の 1 個の構成パラメータを取得する。構成パラメータは表 9-3 または表 9-4 に示される名前付き定数の中の 1 つであり、構成値は、対応する名前付き定数かネイティブ・タイプのどちらかの適切なタイプによって取得される。

この関数は正常に終了すると DFTI_NO_ERROR を返す。戻り値ステータスの詳細は「ステータス確認関数」を参照のこと。

インターフェイスとプロトタイプ

```
! Fortran interface
INTERFACE DftiGetValue
//Note that the body provided here is to illustrate the different
//argument list and types of dummy arguments. The interface
//does not guarantee what the actual function names are.
//Users can only rely on the function name following the
//keyword INTERFACE
 FUNCTION some_actual_function_7_INTVAL( Desc_Handle, Config_Param,
INTVAL )
    INTEGER :: some_actual_function_7_INTVAL
    Type(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
    INTEGER, INTENT(IN) :: Config_Param
    INTEGER, INTENT(OUT) :: INTVAL
 END FUNCTION DFTI_GET_VALUE_INTVAL

 FUNCTION some_actual_function_7_SGLVAL( Desc_Handle, Config_Param,
SGLVAL )
    INTEGER :: some_actual_function_7_SGLVAL
    Type(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
    INTEGER, INTENT(IN) :: Config_Param
    REAL, INTENT(OUT) :: SGLVAL
 END FUNCTION some_actual_function_7_SGLVAL

 FUNCTION some_actual_function_7_DBLVAL( Desc_Handle, Config_Param,
DBLVAL )
    INTEGER :: some_actual_function_7_DBLVAL
    Type(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
    INTEGER, INTENT(IN) :: Config_Param
    REAL (KIND(0D0)), INTENT(OUT) :: DBLVAL
 END FUNCTION some_actual_function_7_DBLVAL

 FUNCTION some_actual_function_7_INTVEC( Desc_Handle, Config_Param,
INTVEC )
    INTEGER :: some_actual_function_7_INTVEC
    Type(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
    INTEGER, INTENT(IN) :: Config_Param
```

```
    INTEGER, INTENT(OUT) :: INTVEC(*)
  END FUNCTION some_actual_function_7_INTVEC

  FUNCTION some_actual_function_7_INTPNT( Desc_Handle, Config_Param,
INTPNT )
    INTEGER :: some_actual_function_7_INTPNT
    Type(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
    INTEGER, INTENT(IN) :: Config_Param
    INTEGER, DIMENSION(*), POINTER :: INTPNT
  END FUNCTION some_actual_function_7_INTPNT

  FUNCTION some_actual_function_7_CHARS( Desc_Handle, Config_Param,
CHARS )
    INTEGER :: some_actual_function_7_CHARS
    Type(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
    INTEGER, INTENT(IN) :: Config_Param
    CHARCTER(*), INTENT(OUT):: CHARS
  END FUNCTION some_actual_function_7_CHARS
END INTERFACE DftiGetValue

/* C prototype */
long DftiGetValue( DFTI_DESCRIPTOR_HANDLE,
                   DFTI_CONFIG_PARAM ,
                   ... );
```

構成設定

変換精度

構成パラメータ DFTI_PRECISION は変換の実行精度が浮動小数点であることを示す。設定が DFTI_SINGLE の場合は単精度が有効で、設定が DFTI_DOUBLE の場合は倍精度が有効となる。データがこの精度で示される意味であり、すなわち計算はこの精度で実行され、結果はこの精度で与えられる。これはデフォルト値を持たない設定可能な 4 つの構成パラメータのうちの 1 個である。ユーザの明示的な設定が必要で、ディスクリプタ生成関数 DftiCreateDescriptor の呼び出しが最も便利である。

順方向変換

離散フーリエ変換の一般の形は次のとおりである。

$$z_{k_1, k_2, \ldots, k_d} = \sigma \times \sum_{j_d=0}^{n_d-1} \ldots \sum_{j_2=0}^{n_2-1} \sum_{j_1=0}^{n_1-1} w_{j_1, j_2, \ldots, j_d} \exp\left(\delta i 2\pi \sum_{l=1}^{d} j_l k_l / n_l\right) \quad (7.1)$$

$d(k_l = 0, \pm 1, \pm 2, \ldots)$、ここでσは任意実数値の倍率因子、またδ = ±1 である。デフォルトでは順方向変換はσ = 1 およびδ = –1 として定義される。最も一般的な状況では、順方向変換の領域、すなわち入力 (周期的) シーケンス $\{w_{j_1, j_2, \ldots, j_d}\}$ が属するセットは、複素値シーケンス、実数シーケンス、複素共役偶シーケンスのいずれかのセットである。構成パラメータ DFTI_FORWARD_DOMAIN は順方向変換の領域を示す。この黙示性によって、DFT の数学的な特性から逆方向変換の領域が指定される点に注意する。詳細は表 9-8 を参照のこと。

表 9-8　順方向領域と逆方向領域の対応

	順方向領域	黙示的な逆方向領域
複素	(DFTI_COMPLEX)	複素
実数	(DFTI_REAL)	共役偶
共役偶	(DFTI_CONJUGATE_EVEN)	実数

実領域での変換において、一部のソフトウェア・パッケージは単一の「実数から複素数」変換のみを提供している。これは順方向変換での共役偶領域が本質的に省略されている。順方向領域構成パラメータ DFTI_FORWARD_DOMAIN は、デフォルトを持たない 4 個の構成パラメータのうちの 2 個目である。

変換の次元と長さ

変換の次元は Integer タイプの整数スカラーで表される正の整数値である。一次元変換では変換長さは Integer タイプの整数スカラーで表される正の整数で指定される。多次元 (≥ 2) 変換では各次元の長さは整数配列で与えられる。DFTI_DIMENSION と DFTI_LENGTHS は、デフォルトを持たない 4 個の構成パラメータのうちの残りの 2 個である。

これまでに述べたように、これら 4 個のパラメータはデフォルト値を持っていない。これらはディスクリプタ生成関数で設定するのがもっとも妥当である。次元と長さの構成値は、DftiSetValue 関数では設定はできず、ディスクリプタ生成関数でしか設定できない。残りの 2 個の構成値は、DftiSetValue 関数を介して変更ができるが一般的な方法ではない。

注意：次元と長さを変更するとストライド値を不正にしてしまうことがある。その場合、ユーザはストライドの再構成を行ったほうがよい (「ストライド」参照)。

変換回数

状況によっては、ユーザは同じ次元、同じ長さの DFT 変換を複数回実行したい場合がある。もっとも一般的な状況は、同じ長さを持つ複数の一次元データの変換であると考えられる。このパラメータのデフォルト値は 1 で、Fortran では Integer データ・タイプ、C では long データ・タイプによって、正の整数として設定する。データセットは共通要素を持たない。回数値が 1 より大きい場合は距離パラメータを必須とする。

符号と倍率

離散フーリエ変換の一般的な形式は式 (7.1) で $k_1 = 0, \pm 1, \pm 2, \ldots$ 、として与えられ、ここで σ は任意実数値の倍率因子、また $\delta = \pm 1$ である。デフォルトでは、順方向変換は $\sigma = 1$ と $\delta = -1$ で定義され、逆方向変換は $\sigma = 1$ と $\sigma = 1$ で定義される。ユーザは δ の符号を DFTI_NEGATIVE (デフォルト) または DFTI_POSITIVE に変更すれば、順方向変換の定義を変更できる。逆方向変換の符号は、順方向変換に対する負の符号として黙示的に定義されている。

順方向変換と逆方向変換は、それぞれの倍率因子σのデフォルト値が 1 という点で互いに関連している。ユーザは 2 個の構成パラメータ DFTI_FORWARD_SCALE と DFTI_BACKWARD_SCALE によって、どちらか、または両方の設定が可能である。たとえば長さ *n* の一次元の変換で、順方向変換の倍率はデフォルトの 1 を使用し、一方、逆方向変換の倍率を 1/*n* に設定すれば、逆方向変換を順方向変換の逆にできる。

倍率因数構成パラメータは、DFTI_PRECISION と同じ精度の実数浮動小数点データ・タイプによって設定しなければならない。

注：符号構成はサポートされていない。順方向変換は $\delta = -1$ として定義される。

結果の配置

デフォルトでは、計算関数は入力データを出力結果で上書きする。すなわち、構成パラメータ DFTI_PLACEMENT のデフォルト設定は DFTI_INPLACE である。ユーザは、これを DFTI_NOT_INPLACE に変更可能である。

パックド形式

実数の順方向変換 (たとえば周波数領域) の結果は、***Pack***、***Perm***、または ***ccs*** のパックド形式によって表現される。実数データの DFT 変換の対称性によってデータはパックが可能になっている。

ccs形式は順方向DFTから得られる出力複素信号の前半の値を格納する。CCS 形式に格納される信号は 1 個の複素要素分だけ長いことに注意する。CCS 形式では、DFT の出力標本は表 9-9 に示されるように整列配置される。

Pack 形式は複素共役対称シーケンスをコンパクトに表現したものである。この形式の欠点は、実際の DFT アルゴリズムで使用されている自然な形式とは異なることである (この「自然」とは配列が複素 DFT に自然を意味する)。Pack 形式では DFT の出力標本は表 9-9 に示すように整列配置される。

Perm 形式は、偶数長の Pack 形式に対しては任意の置換であり、奇数長の Pack 形式に対しては同一である。Perm 形式では DFT の出力標本は表 9-9 のように整列配置される。

表 9-9　パック形式出力標本

(N = S*2) の場合

DFT 実数	**0**	**1**	**2**	**3**	**...**	**N-2**	**N-1**	**N**	**N+1**
	0	**1**					**(2S-1)**	**2S**	**(2S+1)**
CCS	R_0	0	R_1	I_1	...	$R_{N/2-1}$	$I_{N/2-1}$	$R_{N/2}$	0
Pack	R_0	R_1	I_1	R_2	...	$I_{N/2-1}$	$R_{N/2}$		
Perm	R_0	$R_{n/2}$	R_1	I_1	...	$R_{N/2-1}$	$I_{N/2-1}$		

(N = S*2 + 1) の場合

DFT 実数	**0**	**1**	**2**	**3**	**...**	**N-4**	**N-3**	**N-2**	**N-1**	**N**	**N+1**
	0	**1**							**2S**	**(2S+1)**	
CCS	R_0	0	R_1	I_1	...	I_{S_2}	R_{S-1}	I_{S_1}	R_S	I_S	
Pack	R_0	R_1	I_1	R_2	...	R_{S-1}	I_{S_1}	R_{S-1}	I_S		
Perm	R_0	R_1	I_1	R_2	...	R_{S-1}	I_{S_1}	R_{S-1}	I_S		

表 9-10 と表 9-11 も参照のこと。

格納体系

DFTI_COMPLEX、DFTI_REAL、DFTI_CONJUGATE_EVEN (これは順方向の場合だが逆方向演算も同様) の 3 種類の領域の計算に対して、格納体系のサブセットを DFTI_COMPLEX_COMPLEX、DFTI_COMPLEX_REAL、DFTI_REAL_COMPLEX、DFTI_REAL_REAL の 4 つの中から指定できる。格納体系を説明するために例を以下に示す。この格納体系の定義の背景にある理論的根拠は文献 [3] を参照されたい。

注：データは Fortran 形式でのみ格納され、すなわち、実数部と虚数部は隣接して格納される。

複素領域の格納体系

この設定は構成パラメータ DFTI_COMPLEX_STORAGE に登録される。設定可能な 3 種類の値は、DFTI_COMPLEX_COMPLEX、DFTI_COMPLEX_REAL、または DFTI_REAL_REAL である。一次元、長さ *n* の形式の変換を考える。

$$z_k = \sum_{j=0}^{n-1} w_j e^{-i2\pi jk/n} \quad 、 \qquad w_j, z_k \in \mathrm{C}$$

ストライドはデフォルト値 (単位ストライド) を持ち、DFTI_PLACEMENT 設定はデフォルトのイン・プレース (入力上書き) であると仮定する。

1. **DFTI_COMPLEX_COMPLEX 格納体系**。典型的な使用法は次のとおりである。

```
COMPLEX :: X(0:n-1)
...some other code...
Status = DftiComputeForward( Desc_Handle, X )
```

入力において、

$X(j) = w_j$, $j = 0, 1, ..., n-1$

出力において、

$X(k) = z_k$, $k = 0, 1, ..., n-1$

2. **DFTI_COMPLEX_REAL 格納体系**。典型的な使用法は次のとおりである。

```
REAL :: X(0:2*n-1)
...some other code...
Status = DftiComputeForward( Desc_Handle, X )
```

入力において、

X(2*j) = Re(w_j) , X(2*j+1) = Im(w_j) , j = 0, 1, ..., n-1

出力において、

X(2*k) = Re(z_k) , X(2*k+1) = Im(z_k) , k = 0, 1, ..., n-1

記号 Re(w_j) と Im(w_j) は複素数 w_j の実数部と虚数部である。

3. DFTI_REAL_REAL 格納体系。典型的な使用法は次のとおりである。

```
REAL :: X(0:n-1), Y(0:n-1)
...some other code...
Status = DftiComputeForward( Desc_Handle, X, Y )
```

入力において、

X(j) = Re(w_j) , Y(j) = Im(w_j) , j = 0, 1, ..., n-1

出力において、

X(k) = Re(z_k) , Y(k) = Im(z_k) , k = 0, 1, ..., n-1

実数領域と共役偶領域の格納体系

これら領域に対する格納体系の設定は構成パラメータ DFTI_REAL_STORAGE と DFTI_CONJUGATE_EVEN に登録される。順方向の実数領域は逆方向の共役偶領域に一致することから、これらは一緒に扱われる。一次元実数を共役偶に変換する例を使用した。可能であればイン・プレース計算を仮定している (すなわち、入力データ・タイプが出力データ・タイプに一致する場合)。

一次元、長さ n の形式の変換を考える。

$$z_k = \sum_{j=0}^{n-1} w_j e^{-i2\pi jk/n} \quad 、 \qquad w_j \in \mathrm{R}、\quad z_k \in \mathrm{C}$$

これは対称である。

n が偶数の場合：z(n/2+i)=conjg(z(n/2-i))、$1 \le i \le n/2-1$ 、またさらに、z(0) と z(n/2) は実数値。

n が奇数の場合：z(m+i)=conjg(z(m-i+1))、$1 \le i \le m$ 、またさらに、z(0) は実数値。

m = floor(n/2)

表 9-10 順方向変換での複素＝複素 DFT および実数＝複素 DFT の格納効果の比較

N=8

入力ベクトル			出力ベクトル				
complex DFT		real DFT	complex DFT		real DFT		
複素データ		実数データ	複素データ		実数データ		
実数部	虚数部		実数部	虚数部	CCS	Pack	Perm
w0	0.000000	w0	z0	0.000000	z0	z0	z0
w1	0.000000	w1	Re(z1)	Im(z1)	0.000000	Re(z1)	z4
w2	0.000000	w2	Re(z2)	Im(z2)	Re(z1)	Im(z1)	Re(z1)
w3	0.000000	w3	Re(z3)	Im(z3)	Im(z1)	Re(z2)	Im(z1)
w4	0.000000	w4	z4	0.000000	Re(z2)	Im(z2)	Re(z2)
w5	0.000000	w5	Re(z3)	-Im(z3)	Im(z2)	Re(z3)	Im(z2)
w6	0.000000	w6	Re(z2)	-Im(z2)	Re(z3)	Im(z3)	Re(z3)
w7	0.000000	w7	Re(z1)	-Im(z1)	Im(z3)	z4	Im(z3)
					z4		
					0.000000		

N=7

入力ベクトル			出力ベクトル				
Complex DFT		Real DFT	complex DFT		real DFT		
複素データ		実数データ	複素データ		実数データ		
実数部	虚数部		実数部	虚数部	CCS	Pack	Perm
w0	0.000000	w0	z0	0.000000	z0	z0	z0
w1	0.000000	w1	Re(z1)	Im(z1)	0.000000	Re(z1)	Re(z1)
w2	0.000000	w2	Re(z2)	Im(z2)	Re(z1)	Im(z1)	Im(z1)

N=7

入力ベクトル			出力ベクトル				
Complex DFT		Real DFT	complex DFT		real DFT		
複素データ		実数データ	複素データ		実数データ		
実数部	虚数部		実数部	虚数部	CCS	Pack	Perm
w3	0.000000	w3	Re(z3)	Im(z3)	Im(z1)	Re(z2)	Re(z2)
w4	0.000000	w4	Re(z3)	-Im(z3)	Re(z2)	Im(z2)	Im(z2)
w5	0.000000	w5	Re(z2)	-Im(z2)	Im(z2)	Re(z3)	Re(z3)
w6	0.000000	w6	Re(z1)	-Im(z1)	Re(z3)	Im(z3)	Im(z3)
					Im(z3)		

表 9-11　逆方向変換での複素＝複素 DFT および複素＝実数 DFT の格納効果の比較

N=8

出力ベクトル			入力ベクトル				
Complex DFT		Real DFT	complex DFT				
複素データ		実数データ	複素データ				
実数部	虚数部		実数部	虚数部	CCS	Pack	Perm
w0	0.000000	w0	z0	0.000000	z0	z0	z0
w1	0.000000	w1	Re(z1)	Im(z1)	0.000000	Re(z1)	z4
w2	0.000000	w2	Re(z2)	Im(z2)	Re(z1)	Im(z1)	Re(z1)
w3	0.000000	w3	Re(z3)	Im(z3)	Im(z1)	Re(z2)	Im(z1)
w4	0.000000	w4	z4		Re(z2)	Im(z2)	Re(z2)
w5	0.000000	w5	Re(z3)	-Im(z3)	Im(z2)	Re(z3)	Im(z2)
w6	0.000000	w6	Re(z2)	-Im(z2)	Re(z3)	Im(z3)	Re(z3)
w7	0.00000	w7	Re(z1)	-Im(z1)	Im(z3)	z4	Im(z3)
					z4		
					0.000000		

N=7

出力ベクトル			入力ベクトル				
Complex DFT		Real DFT	complex DFT		real DFT		
複素データ		実数データ	複素データ		実数データ		
実数部	虚数部		実数部	虚数部	CCS	Pack	Perm
w0	0.000000	w0	z0	0.000000	z0	z0	z0
w1	0.000000	w1	Re(z1)	Im(z1)	0.000000	Re(z1)	Re(z1)
w2	0.000000	w2	Re(z2)	Im(z2)	Re(z1)	Im(z1)	Im(z1)
w3	0.000000	w3	Re(z3)	Im(z3)	Im(z1)	Re(z2)	Re(z2)
w4	0.000000	w4	Re(z3)	-Im(z3)	Re(z2)	Im(z2)	Im(z2)
w5	0.000000	w5	Re(z2)	-Im(z2)	Im(z2)	Re(z3)	Re(z3)

N=7

出力ベクトル			入力ベクトル				
Complex DFT		Real DFT	complex DFT		real DFT		
複素データ		実数データ	複素データ		実数データ		
実数部	虚数部		実数部	虚数部	CCS	Pack	Perm
w6	0.000000	w6	Re(z1)	-Im(z1)	Re(z3)	Im(z3)	Im(z3)
					Im(z3)		

ストライドはデフォルト値 (単位ストライド) であると仮定する。

この複素共役対称ベクトルは、パックド形式に依存して、大きさ m+1 の複素配列、または大きさ 2m+2 か 2m の実数配列に格納できる。

1. 実数領域では DFTI_REAL_REAL、共役偶領域では DFTI_COMPLEX_COMPLEX。典型的な使用法は次のとおりである。

```
// m = floor( n/2 )
REAL :: X(0:n-1)
COMPLEX :: Y(0:m)
...some other code...
...out of place transform...
Status = DftiComputeForward( Desc_Handle, X, Y )
```

入力において、

X(j) $= w_j$, $j = 0, 1, ..., n-1$

出力において、

Y(k) $= z_k$, $k = 0, 1, ..., m$

2. 実数領域では DFTI_REAL_REAL、共役偶領域では DFTI_COMPLEX_REAL。典型的な使用法は次のとおりである。

```
// m = floor( n/2 )
REAL :: X(0:2*m+1)
...some other code...
...assuming inplace...
Status = DftiComputeForward( Desc_Handle, X )
```

入力において、

X(j) = w_j , j = 0, 1, ..., n-1

出力において、

出力データは、Pack、Perm、または CCS のどれか 1 つの形式で格納される (「パックド形式」を参照)。

CCS 形式：x(2*k) = Re(z_k) , x(2*k+1) = Im(z_k) , k = 0, 1, ..., m

Pack 形式：偶数 n： x(0) = Re(z_0), x(2*k-1) = Re(z_k), x(2*k) = Im(z_k), k = 1, ..., m-1, and x(n-1) = Re(z_m)

奇数 n：x(0) = Re(z_0), x(2*k-1) = Re(z_k), x(2*k) = Im(z_k), k = 1, ..., m

Perm 形式：偶数 n：x(0) = Re(z_0), x(1) = Re(z_m), x(2*k) = Re(z_k) , x(2*k+1) = Im(z_k) , k = 1, ..., m-1、

奇数 n：x(0) = Re(z_0), x(2*k-1) = Re(z_k), x(2*k) = Im(z_k), k = 1, ..., m

3. 実数領域では DFTI_REAL_REAL、共役偶領域では DFTI_REAL_REAL。
共役偶領域に対するこの格納体系は一次元変換にのみ適用できる。典型的な使用法は次のとおりである。

```
// m = floor( n/2 )
REAL :: X(0:n-1)
...some other code...
...assuming inplace...
Status = DftiComputeForward( Desc_Handle, X )
```

入力において、

X(j) = w_j , j = 0, 1, ..., n-1

出力において、

X(k) = Re(z_k) , k = 0, 1, ..., m

および

X(n-k) = Im(z_k) , k = 1, 2, ..., m-1

4. 実数領域では DFTI_REAL_COMPLEX、共役偶領域では DFTI_COMPLEX_COMPLEX。
典型的な使用法は次のとおりである。

```
// m = floor( n/2 )
COMPLEX :: X(0:n-1)
...some other code...
...inplace transform...
Status = DftiComputeForward( Desc_Handle, X )
```

入力において、

X(j) $= w_j$, $j = 0, 1, ..., n-1$

すなわち、X(j) の虚数部はゼロである。出力において、

Y(k) $= z_k$, $k = 0, 1, ..., m$

ここで m は $\lfloor n/2 \rfloor$

5．実数領域では DFTI_REAL_COMPLEX、共役偶領域では DFTI_COMPLEX_REAL。典型的な使用法は次のとおりである。

```
// m = floor( n/2 )
COMPLEX :: X(0:n-1)
REAL :: Y(0:2*m+1)
...some other code...
...not inplace...
Status = DftiComputeForward( Desc_Handle, X, Y )
```

入力において、

X(j) $= w_j$, $j = 0, 1, ..., n-1$

出力において、

出力データは、Pack、Perm、または CCS のどれか 1 つの形式で格納される (「パックド形式」を参照)。

CCS 形式：Y(2*k) = Re(z_k) , Y(2*k+1) = Im(z_k) , k = 0, 1, ..., m

Pack 形式：偶数 n：Y(0) = Re(z_0), Y(2*k-1) = Re(z_k), Y(2*k) = Im(z_k), k = 1, ..., m-1, Y(n-1) = Re(z_m)

奇数 n：Y(0) = Re(z_0), Y(2*k-1) = Re(z_k), Y(2*k) = Im(z_k), k = 1, ..., m

Perm 形式：偶数 n：Y(0) = Re(z_0), Y(1) = Re(z_m), Y(2*k) = Re(z_k) , Y(2*k+1) = Im(z_k) , k = 1, ..., m-1、

奇数 n：Y(0) = Re(z_0), Y(2*k-1) = Re(z_k), Y(2*k) = Im(z_k), k = 1, ..., m

6．実数領域では DFTI_REAL_COMPLEX、共役偶領域では DFTI_REAL_REAL。共役偶領域に対するこの格納体系は一次元変換にのみ適用できる。典型的な使用法は次のとおりである。

```
// m = floor( n/2 )
COMPLEX :: X(0:n-1)
```

```
REAL :: Y(0:n-1)
...some other code...
...not inplace...
Status = DftiComputeForward( Desc_Handle, X, Y )
```

入力において、

X(j) $= w_j$, $j = 0, 1, ..., n-1$

出力において、

Y(k) $= \mathrm{Re}(z_k)$, $k = 0, 1, ..., m$

および

Y(n-k) $= \mathrm{Im}(z_k)$, $k = 1, 2, ..., m-1$

入力と出力との距離

インテル・マス・カーネル・ライブラリの DFT インターフェイスは複数回の変換計算を許している。そのため、複数セットのデータのデータ分布をユーザが指定できる必要がある。これは、連続するデータ・セットの最初のデータ要素同士の距離で実現される。このパラメータは変換回数が 1 より大きい場合は必須である。データ・セットは何ら共通要素は持っていない。次の例で仕様を説明する。X(0:31, 1)、X(0:31, 2)、X(0:31, 3) に格納された3つの長さ32の複素シーケンスに順方向DFTを計算する場合を考える。結果は配列 Y(0:63, 3) の位置 Y(0:31, k)、k = 1, 2, 3 に格納されるとする。以上から、入力距離は 32 になり、一方の出力距離は 64 になる。計算関数でのデータと結果パラメータは、すべて擬寸法階数 1 の配列 DIMENSION(0:*) として宣言されている点に注意する。そのため二次元配列は EQUIVALENCE 文か Fortran のその他の機能を用いて一次元配列に変換しなければならない。以下にコードの一部を示す。

```
Complex ::  X_2D(0:31,3), Y_2D(0:63, 3)
Complex ::  X(96), Y(192)
Equivalence (X_2D, X)
Equivalence (Y_2D, Y)
...................
Status = DftiCreateDescriptor(Desc_Handle, DFTI_SINGLE,
                              DFTI_COMPLEX, 1, 32)
Status = DftiSetValue(Desc_Handle, DFTI_NUMBER_OF_TRANSFORM, 3)
```

```
Status = DftiSetValue(Desc_Handle, DFTI_INPUT_DISTANCE, 32)
Status = DftiSetValue(Desc_Handle, DFTI_OUTPUT_DISTANCE, 64)
Status = DftiSetValue(Desc_Handle, DFTI_PLACEMENT, DFTI_NOT_INPLACE)
Status = DftiCommitDescriptor(Desc_Handle)
Status = DftiComputeForward(Desc_Handle, X, Y)
Status = DftiFreeDescriptor(Desc_Handle)
```

ストライド

DFT インターフェイスは、データ・セットの複数回変換のサポートに加えて、各データ・セット内のデータに対して単位ストライド以外のストライド分布をサポートしている。シーケンス x_j, $0 \le j < 32$ において長さ 32 の DFT が計算される次のような状況を考える。これら値の実際の位置は、配列 `x(1:68)` の `x(5)`、`x(7)`、...、`x(67)` である。DFT インターフェイスによって実現されるストライドは、データ配列の最初の要素からの変位 L_0 (この場合 4) と、続く要素の一定の距離 L_1 (この場合 2) で構成される。よって Fortran 配列 `x` は次のようになる。

x_j = `x`$(1 + L_0 + L_1 * j)$ = `x`$(5 + L_1 * j)$

ストライド・ベクトル (4,2) は長さ 2、階数 1 の整数配列によって与えられる。

```
COMPLEX :: X(68)
INTEGER :: Stride(2)
...................
Status = DftiCreateDescriptor(Desc_Handle, DFTI_SINGLE,
                              DFTI_COMPLEX, 1, 32)
Stride = (/ 4, 2 /)
Status = DftiSetValue(Desc_Handle, DFTI_INPUT_STRIDE, Stride)
Status = DftiSetValue(Desc_Handle, DFTI_OUTPUT_STRIDE, Stride)
Status = DftiCommitDescriptor(Desc_Handle)
Status = DftiComputeForward(Desc_Handle, X)
Status = DftiFreeDescriptor(Desc_Handle)
```

一般に d 次元変換ではストライドは長さ $d+1$ の整数ベクトル $(L_0, L_1, L_2, ..., L_d)$ で与えられ、それぞれの意味は次のとおりである。

L_0 = 最初の配列要素からの変位

L_1 = 最初の次元の連続するデータ要素間の距離

L_2 = 2 番目の次元の連続するデータ要素間の距離

... = ...

$L_d = d$ 番目の次元の連続するデータ要素間の距離

d 次元のデータ・シーケンス、

$x_{j_1, j_2, \ldots, j_d}$ 、 $0 \le j_i < J_i$ 、 $1 \le i \le d$

は、階数 1 の配列 x に以下のマッピングによって格納される。

$x_{j_1, j_2, \ldots, j_d} = \mathrm{x}\,(\text{first index} + L_0 + j_1 L_1 + j_2 L_2 + \ldots + j_d L_d)$

複数回の変換では、値 L_0 は最初のデータ・シーケンスに適用され、L_j、j=1、2、...、d はすべてのデータ・シーケンスに適用される。

単一の一次元シーケンスの場合、L_1 は単純に通常のストライドとなる。一般的な多次元の状態でのストライドのデフォルト設定は、シーケンスが配列内に厳密に分布している場合に相当する。

$$L_1 = 1, L_2 = J_1, L_3 = J_1 J_2, \ldots, L_d = \prod_{i=1}^{d-1} J_i$$

入力データと出力データのそれぞれは関連するストライドを有する。デフォルトは、データが言語に自然な方法でメモリに隣接して格納されるように設定されている。

最後に、20 × 40 の 2 次元 DFT を一次元変換を明示的に用いて計算した不自然な例を考える。計算関数のデータと結果パラメータは、擬寸法階数 1 の配列 DIMENSION(0:*) として宣言されている点に注意する。そのため、2 次元配列は EQUIVALENCE 文か Fortran のその他の機能で一次元配列に変換しなければならない。

```
! Fortran
Complex ::  X_2D(20,40),
Complex ::  X(800)
Equivalence (X_2D, X)
INTEGER :: STRIDE(2)
type(DFTI_DESCRIPTOR), POINTER :: Desc_Handle_Dim1
type(DFTI_DESCRIPTOR), POINTER :: Desc_Handle_Dim2
...
Status = DftiCreateDescriptor( Desc_Handle_Dim1, DFTI_SINGLE,
                               DFTI_COMPLEX, 1, 20 )
```

```
Status = DftiCreateDescriptor( Desc_Handle_Dim2, DFTI_SINGLE,
                                DFTI_COMPLEX, 1, 40 )

! perform 40 one-dimensional transforms along 1st dimension
Status = DftiSetValue( Desc_Handle_Dim1, DFTI_NUMBER_OF_TRANSFORMS, 40 )
Status = DftiSetValue( Desc_Handle_Dim1, DFTI_INPUT_DISTANCE, 20 )
Status = DftiSetValue( Desc_Handle_Dim1, DFTI_OUTPUT_DISTANCE, 20 )
Status = DftiCommitDescriptor( Desc_Handle_Dim1 )
Status = DftiComputeForward( Desc_Handle_Dim1, X )

! perform 20 one-dimensional transforms along 2nd dimension
Stride(1) = 0; Stride(2) = 20
Status = DftiSetValue( Desc_Handle_Dim2, DFTI_NUMBER_OF_TRANSFORMS, 20 )
Status = DftiSetValue( Desc_Handle_Dim2, DFTI_INPUT_DISTANCE, 1 )
Status = DftiSetValue( Desc_Handle_Dim2, DFTI_OUTPUT_DISTANCE, 1 )
Status = DftiSetValue( Desc_Handle_Dim2, DFTI_INPUT_STRIDES, Stride )
Status = DftiSetValue( Desc_Handle_Dim2, DFTI_OUTPUT_STRIDES, Stride )
Status = DftiCommitDescriptor( Desc_Handle_Dim2 )
Status = DftiComputeForward( Desc_Handle_Dim2, X )
Status = DftiFreeDescriptor( Desc_Handle_Dim1 )
Status = DftiFreeDescriptor( Desc_Handle_Dim2 )

/* C */
float _Complex x[20][40];
long stride[2];
DFTI_DESCRIPTOR_HANDLE Desc_Handle_Dim1;
DFTI_DESCRIPTOR_HANDLE Desc_Handle_Dim2;
...
status = DftiCreateDescriptor( &desc_handle_dim1, DFTI_SINGLE,
                               DFTI_COMPLEX, 1, 20 );
status = DftiCreateDescriptor( &desc_handle_dim2, DFTI_SINGLE,
                               DFTI_COMPLEX, 1, 40 );

/* perform 40 one-dimensional transforms along 1st dimension */
/* note that the 1st dimension data are not unit-stride */
stride[0] = 0; stride[1] = 40;
status = DftiSetValue( desc_handle_dim1, DFTI_NUMBER_OF_TRANSFORMS, 40 );
```

```
status = DftiSetValue( desc_handle_dim1, DFTI_INPUT_DISTANCE, 1 );
status = DftiSetValue( desc_handle_dim1, DFTI_OUTPUT_DISTANCE, 1 );
status = DftiSetValue( desc_handle_dim1, DFTI_INPUT_STRIDES, stride );
status = DftiSetValue( desc_handle_dim1, DFTI_OUTPUT_STRIDES, stride );
status = DftiCommitDescriptor( desc_handle_dim1 );
status = DftiComputeForward( desc_handle_dim1, x );

/* perform 20 one-dimensional transforms along 2nd dimension */
/* note that the 2nd dimension is unit stride */
status = DftiSetValue( desc_handle_dim2, DFTI_NUMBER_OF_TRANSFORMS, 20 );
status = DftiSetValue( desc_handle_dim2, DFTI_INPUT_DISTANCE, 40 );
status = DftiSetValue( desc_handle_dim2, DFTI_OUTPUT_DISTANCE, 40 );
status = DftiCommitDescriptor( desc_handle_dim2 );
status = DftiComputeForward( desc_handle_dim2, x );
status = DftiFreeDescriptor( &Desc_Handle_Dim1 );
status = DftiFreeDescriptor( &Desc_Handle_Dim2 );
```

初期化努力

DFT 計算用の高速アルゴリズム (FFT) を構築するための近代的なアプローチでは、後続するデータに対して高い計算効率を得るために、FFT アルゴリズムの初期化 (準備) により多くの努力を費やす傾向がある。インテル・マス・カーネル・ライブラリのアドバンスト DFT 関数では、DFTI_INITIALIZATION_EFFORT 構成パラメータを使ってこの状況に対応している。3 種類の構成値は DFTI_LOW、DFTI_MEDIUM (デフォルト)、DFTI_HIGH である。この設定は、DFT インターフェイスの固有の実装によって使用できない場合があることに注意されたい (実装の詳細は**マス・カーネル・ライブラリのリリース・ノート**参照)。

順序指定

数々の FFT アルゴリズムには、処理時間を要する明示的な置換段階が適用されていることはよく知られている [4]]。ただし、スクランブルされた入力データに DFT を適用するか、あるいはスクランブルした DFT 結果出力を許せば、上記段階を回避できる。畳み込みとパワー・スペクトラム計算のようなアプリケーションでは、結果またはデータの順序は重要ではないため、性能向上が得られるのであればスクランブル順の許可は魅力的である。インテル・マス・カーネル・ライブラリでは次の 3 つのオプションが利用可能である。

1. DFTI_ORDERED：順方向変換データは順序どおり、逆方向変換データは順序どおり (デフォルト・オプション)。
2. DFTI_BACKWARD_SCRAMBLED：順方向変換データは順序どおり、逆方向変換データはスクランブル。
3. DFTI_FORWARD_SCRAMBLED：順方向データはスクランブル、逆方向データは順序どおり。

表 9-12 に、この構成設定の効果を示す。

表 9-12 スクランブル順の変換

	DftiComputeForward	DftiComputeBackward
DFTI_ORDERING	**入力 → 出力**	**入力 → 出力**
DFTI_ORDERED	順序どおり → 順序どおり	順序どおり → 順序どおり
DFTI_BACKWARD_SCRAMBLED	順序どおり → スクランブル	スクランブル → 順序どおり
DFTI_FORWARD_SCRAMBLED	スクランブル → 順序どおり	順序どおり → スクランブル

後半の 2 個のオプションは「可能ならスクランブル順を許す」意味である点に注意する。データのスクランブルを許しても性能向上が得られない状況もあり、そのため実装によっては示唆を無視する場合がある。厳密には、些細であるが、普通の順序もスクランブルされた順序の 1 つである。

順序指定の設定がデフォルトの DFTI_ORDERED 以外の場合、入力データと出力データの実際の順序指定は次のように取得できる。順方向領域でのデータの指定順は、構成パラメータ DFTI_FORWARD_ORDERING を読み出す (取得する) ことで得られる。逆方向領域のデータの指定順は、構成パラメータ DFTI_BACKWARD_ORDERING を読み出す (取得する) ことで得られる。構成値は整数ベクトルであり、そのため整数配列へのポインタとして与えられる。ここで、これら整数値によって、実際のデータ・スクランブルがどのように指定されているかを説明する。

すべてのスクランブルは 1 つの単一次元に沿った桁反転に関係する。正確には、長さ J は K 次の因数 $D_1, D_2, ..., D_K$ に因数分解される。任意のインデックス $i, 0 \le i < n$ は、$0 \le i_l < D_l$、かつ

$$i=i_1+i_2D_1+i_3D_1D_2+...+i_KD_1D_2...D_{K-1}$$

となる K 桁の $i_1, i_2, ..., i_K$ として独自に表現される。

桁反転置換 scram(i) は次のように与えられる。

$$\mathrm{scram}(i) = i_K + i_{K-1}D_K + i_{K-2}D_K D_{K-1} + \ldots + i_1 D_K D_{K-1} \ldots D_2$$

J の因数分解が単一の因子 J になる場合は何のスクランブルも行われないことに相当し、scram(i) = i である。因数分解は実行される「バタフライ」段階の回数に正確に一致する必要はない点に注意する。実際に、計算ルーチンが初期化段階にあるときに、一部または全次元のスクランブル順が性能向上に結びつくかどうかが決まる。桁反転の対象となる桁はディスクリプタ内に登録かつ格納される。これら桁は、整数配列へのポインタを返す対応する問い合わせルーチンを呼び出すことで取得できる。最初の要素は $K^{(1)}$ で第一の次元の桁数を示し、次に対応する桁の $K^{(1)}$ の値が続く。次元が 1 よりも大きい場合、次の整数値は $K^{(2)}$、...、と続いていく。

mod-p sort [4] のような単純な置換は桁反転の特別な場合である。そのためこのオプションは、「6 段階」または「4 段階」フレームワークを介した一次元 DFT の高性能実装に利用価値がある。

ユーザは順方向データと逆方向データのスクランブル設定を確認できる。この情報は複数のシーケンス ($K, D_1, D_2, \ldots, D_K$) を含む整数ベクトルとして返され、それぞれが各次元である。最初の要素は D がいくつ続くかを示している。問い合わせルーチンはメモリを割り当て、この情報を書き込んだあと、メモリ位置へのポインタを返す。

ワークスペース

一部の FFT アルゴリズムは反転処理においてスクラッチ空間を必要としない。ユーザは `DFTI_WORKSPACE` オプションとして `DFTI_ALLOW` (デフォルト) か `DFTI_AVOID` の設定を選択できる。`DFTI_AVOID` は「可能ならば使わない」意味である点に注意する。したがって、設定に関わらず実装はワークスペースを柔軟に利用できる。

転置

高次変換の結果を転置の形式で表現可能にするオプションである。デフォルト設定は `DFTI_NONE` だが、`DFTI_ALLOW` に設定可能である。スクランブル順と同様に、高次変換では、結果を転置形式で与えるようにすると性能が得られる場合がある。

DFT インターフェイスは、性能向上が可能な場合に対応して、転置形式で結果を出力するオプションを備えている。一般的なストライド仕様は転置表現に自然に適合することから、このオプションを利用すると、出力のストライドがユーザによって元々指定されたストライドと異なることが起こる。2 次元結果 $y_{j_1, j_2}, 0 \le j_i < n_i$ が見込まれる例を考えてみる。はじめにユーザは、汎用ストライド L_1=1 および L_2=n_1 を用いて結果を (フラットな) 配列 y に割り当てるように指定した。転置オプションを使用すると実際の計算結果はストライド L_1=n_2 と L_2=1 で y に返される場合がある。これらストライドは対応する問い合わせ関数で取得できる。3 次元以上では、転置は次元の任意な置換を意味する点に注意する。

ルーチンおよび関数の引数

A

BLAS ルーチンの主要な引数は、ベクトルとマトリックスである。一方、VML 関数はベクトル引数のみで動作する。
この後の各項では、これらの引数それぞれについて、例を挙げながら説明する。

BLAS のベクトル引数

ベクトル引数は、1 次元の配列で渡される。配列の次元 (長さ) とベクトルの増分は、整数の変数として渡される。長さによって、ベクトル内の成分の数が決まる。増分 (ストライドとも呼ばれる) によって、ベクトルの成分間の間隔と、ベクトルが渡される配列内の成分の順序が決まる。

長さが *n* で増分が *incx* のベクトルは、その値が次のように定義される 1 次元の配列で渡される。

x(1), *x*(1+|*incx*|), ..., *x*(1+(*n*-1)* |*incx*|)

incx が正の場合は、配列 *x* 内の成分は昇順に格納される。*incx* が負の場合は、配列 *x* 内の成分は降順に格納され、また最初の成分は *x*(1+(*n*-1)* |*incx*|) として定義される。*incx* がゼロの場合は、ベクトルの全成分は同じ値 *x*(1) をとる。ベクトルを格納する 1 次元配列の次元は、常に、*idimx* = 1 + (*n*-1)* |*incx* | 以上でなければならない。

例 A-1　1 次元の実数配列

`x(1:7)` が、次の 1 次元の実数配列とする。

```
x = (1.0, 3.0, 5.0, 7.0, 9.0, 11.0, 13.0)
```

`incx =2` かつ `n = 3` の場合は、このベクトル引数の成分を始めから終わりまで並べると、`(1.0, 5.0, 9.0)` になる。

`incx = -2` かつ `n = 4` の場合は、このベクトル成分を始めから終わりまで並べると、`(13.0, 9.0, 5.0, 1.0)` になる。

`incx = 0` かつ `n = 4` の場合は、このベクトル成分を始めから終わりまで並べると、`(1.0, 1.0, 1.0, 1.0)` になる。

マトリックスの 1 次元下部構造 (行、列、対角成分など) は、開始アドレスと増分を指定してベクトル引数として渡せる。Fortran では、`m` 行 `n` 列のマトリックスは、列主体の順序に基づいて格納される。つまり、同じ列内の成分間の増分は `1`、同じ行内の成分間の増分は `m`、同じ対角成分上の成分間の増分は `m + 1` になる。

例 A-2　2 次元の実マトリックス

`a` が、`REAL A (5, 4)` として宣言された 5 × 4 の実マトリックスとする。

`a` の 3 番目の列を 2.0 でスケーリングするには、BLAS ルーチン `sscal` を次の呼び出しシーケンスで使用する。

```
call sscal (5, 2.0, a(1,3), 1)
```

2 番目の行をスケーリングするには、次のステートメントを使用する。

```
call sscal (4, 2.0, a(2, 1), 5)
```

A の主対角成分を 2.0 でスケーリングするには、次のステートメントを使用する。

```
call sscal (5, 2.0, a(1, 1), 6)
```

注：デフォルトのベクトル引数は 1 とみなされる。

VML のベクトル引数

VML 数学関数のベクトル引数は、単位ベクトル増分を使用する 1 次元配列に渡される。つまり、長さ n のベクトルは、a[0], a[1], ..., a[n-1] として値が定義される配列 a に連続して渡される (C インターフェイスの場合)。
他の増分を使用する配列、またはより複雑なインデックスを使用する配列を収容するため、VML では、pack/unpack 関数を補助的に用意している。pack/unpack 関数は、配列の成分を収集して連続するベクトルを生成し、計算の完了後、それらを分散するものである。

一般に、ベクトル成分が 1 次元配列 a に次のように格納されており、

$a[m_0], a[m_1], ..., a[m_{n-1}]$

配列 y に次のように再グループ化する必要がある場合、

$y[k_0], y[k_1], ..., y[k_{n-1}],$

VML pack/unpack 関数では、以下のいずれかのインデックス方式を使用できる。

正増分インデックス

$k_j = incy * j, m_j = inca * j, j = 0 ,..., n-1$

制約 : incy > 0、inca > 0
例えば、incy = 1 を設定すると、配列の成分が収集されて連続するベクトルが生成される。

この方式は、負とゼロの増分が許可されない点を除けば、BLAS で使用される方式と同じである。

インデックス・ベクトル・インデックス

$k_j = iy[j], m_j = ia[j], j = 0 ,..., n-1,$

ia と iy は長さ n の配列であり、それぞれ入力配列 a 用のインデックス・ベクトルと出力配列 y 用のインデックス・ベクトルを格納する。

マスク・ベクトル・インデックス

インデックス k_j、m_j は、以下の条件を満たす。

$my[k_j] \neq 0, ma[m_j] \neq 0 , j = 0,..., n-1,$

ma と my は、それぞれ入力配列 a 用のマスク・ベクトル、出力配列 y 用のマスク・ベクトルを格納する配列である。

マトリックス引数

インテル® マス・カーネル・ライブラリ・ルーチンのマトリックス引数は、以下に挙げる格納形式を使用して、1 次元または 2 次元の配列に格納できる。

- 通常のフル格納 (2 次元配列に格納)
- エルミート・マトリックス、対称マトリックス、または三角マトリックス用のパックド格納 (1 次元配列に格納)
- バンド・マトリックス用のバンド格納 (2 次元配列に格納)

フル格納は、次のような明確な形式である。マトリックス A は、2 次元配列 `a` に格納される。このとき、マトリックスの成分 a_{ij} は、配列の成分 `a(i, j)` に格納される。

マトリックスが、**三角マトリックス** (引数 `uplo` に指定した値に従って、上三角マトリックスまたは下三角マトリックス) の場合は、関連する三角成分だけが格納される。配列の残りの成分は、設定する必要はない。

対称マトリックスまたはエルミート・マトリックスを処理するルーチンを使用すると、マトリックスの上三角または下三角のいずれかを、配列の対応する成分に格納できる。

`uplo='U'` の場合は、$i \le j$ に対して、a_{ij} が `a(i, j)` に格納される。
`a` の他の成分は、設定する必要はない。

`uplo='L'` の場合は、$j \le i$ に対して、a_{ij} が `a(i, j)` に格納される。
`a` の他の成分は、設定する必要はない。

パックド格納を使用すると、対称マトリックス、エルミート・マトリックス、または三角マトリックスをよりコンパクトに格納できる。関連する三角部分 (この場合も、引数 `uplo` で指定) は、列ごとに 1 次元配列 `ap` にパックされる。

`uplo='U'` の場合は、$i \le j$ に対して、a_{ij} が `ap(i+j(j-1)/2)` に格納される。

`uplo='L'` の場合は、$j \le i$ に対して、a_{ij} が `ap(i+(2*n-j)*(j-1)/2)` に格納される。

LAPACK ルーチンの説明では、パックド形式のマトリックスを保持する配列は、名前の末尾に `p` が付いている。

バンド格納では、`kl` 個の非ゼロの劣対角成分と `ku` 個の優対角成分を持つ `m` 行 `n` 列のバンド・マトリックスは、(`kl`+`ku`+1) 行 `n` 列の 2 次元配列 `ab` にコンパクトに格納される。マトリックスの各列は配列の対応する列に、マトリックスの各対角成分は配列の各行にそれぞれ格納される。したがって、a_{ij} は、$\max(n, j-ku) \le i \le \min(n, j+kl)$ に対して、`ab(ku+1+i-j, j)` に格納される。

バンド格納形式は、*kl* と *ku* がマトリックスのサイズ *n* よりはるかに小さい場合にのみ使用する。(ルーチンは、*kl* と *ku* がどのような値であっても正常に動作するが、マトリックスが実際にバンド化されていない場合は、バンド格納を使用すると効率が低下する。)

一般バンド・マトリックスを LU **因子分解**用に入力として与える場合は、行の交換の結果として埋め込みによって生成される優対角成分をさらに *kl* 個だけ格納するためのスペースがなくてはならない。つまり、マトリックスは上記の形式で格納されるが、*kl* + *ku* 個の優対角成分を持つことにならない。

バンド格納方式の例を以下に示す (*m* = *n* = 6, *kl* = 2, *ku* = 1 の場合)。

バンド・マトリックス A

$$\begin{bmatrix} a_{11} & a_{12} & 0 & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 & 0 & 0 \\ a_{31} & a_{32} & a_{33} & a_{34} & 0 & 0 \\ 0 & a_{42} & a_{43} & a_{44} & a_{45} & 0 \\ 0 & 0 & a_{53} & a_{54} & a_{55} & a_{56} \\ 0 & 0 & 0 & a_{64} & a_{65} & a_{66} \end{bmatrix}$$

A のバンド格納

$$\begin{matrix} * & * & * & + & + & + \\ * & * & + & + & + & + \\ * & a_{12} & a_{23} & a_{34} & a_{45} & a_{56} \\ a_{11} & a_{22} & a_{33} & a_{44} & a_{55} & a_{66} \\ a_{21} & a_{32} & a_{43} & a_{54} & a_{65} & * \\ a_{31} & a_{42} & a_{53} & a_{64} & * & * \end{matrix}$$

* でマークされた配列の成分はルーチンで使用されない。+ でマークされた成分は入力時に設定する必要はないが、結果を格納するために LU 因子分解ルーチンで必要となる。入力配列は、終了時に LU 因子分解の各成分で次のように上書きされる。

$$\begin{matrix} * & * & * & u_{14} & u_{25} & u_{36} \\ * & * & u_{13} & u_{24} & u_{35} & u_{46} \\ * & u_{12} & u_{23} & u_{34} & u_{45} & u_{56} \\ u_{11} & u_{22} & u_{33} & u_{44} & u_{55} & u_{66} \\ m_{21} & m_{32} & m_{43} & m_{54} & m_{65} & * \\ m_{31} & m_{42} & m_{53} & m_{64} & * & * \end{matrix}$$

u_{ij} は上三角マトリックス U の各成分であり、m_{ij} は因子分解で使用される乗数である。

三角バンド・マトリックスは、上三角マトリックスの場合は `kl`= 0、下三角マトリックスの場合は `ku` = 0 として同じ形式で格納される。`k` 個の劣対角成分または優対角成分を持つ対称バンド・マトリックスまたはエルミート・バンド・マトリックスに対しては、引数 `uplo` で指定したとおりに、上三角マトリックスまたは下三角マトリックスだけを格納しなければならない。

`uplo='U'` の場合は、$\max(1, j-k) \le i \le j$ に対して、a_{ij} は `ab(k+1+i-j, j)` に格納される。
`uplo='L'` の場合は、$j \le i \le \min(n, j+k)$ に対して、a_{ij} は `ab(1+i-j, j)` に格納される。

LAPACK ルーチンの説明では、バンド格納でマトリックスを保持する配列は、名前の末尾に `b` が付いている。

Fortran では、列を主体とする順序での格納方法が想定されている。つまり、同じ列の成分が続けて格納されることになる。

2 次元配列の引数には、リーディング・ディメンジョン (同じ行内の成分間の格納位置の数を指定する)、行数、列数の 3 つの特性が関連付けられるのが普通である。フル格納のマトリックスに対しては、配列のリーディング・ディメンジョンは、少なくともマトリックス内の行数と同じ大きさでなければならない。

文字転置パラメータは、マトリックス引数を通常形式と転置形式のどちらで使用するか、あるいは、複素マトリックスの場合はマトリックスの共役転置を使用するかどうかを指示するために頻繁に使用される。
これら 3 つのケースに対する転置パラメータの値は、次のようになる。

`'N'` または `'n'`	通常形式 (共役や転置は使用しない)
`'T'` または `'t'`	転置
`'C'` または `'c'`	共役転置

例 A-3　2 次元複素配列

`A (1:5, 1:4)` は、次のマトリックスで表される 2 次元の複素配列とする。

$$\begin{bmatrix} (1.1, 0.11) & (1.2, 0.12) & (1.3, 0.13) & (1.4, 0.14) \\ (2.1, 0.21) & (2.2, 0.22) & (2.3, 0.23) & (2.4, 0.24) \\ (3.1, 0.31) & (3.2, 0.32) & (3.3, 0.33) & (3.4, 0.34) \\ (4.1, 0.41) & (4.2, 0.42) & (4.3, 0.43) & (4.4, 0.44) \\ (5.1, 0.51) & (5.2, 0.52) & (5.3, 0.53) & (5.4, 0.54) \end{bmatrix}$$

transa が転置パラメータ、*m* が行数、*n* が列数、*lda* がリーディング・ディメンジョンとする。このとき、

`transa = 'N'`、`m = 4`、`n = 2`、`lda = 5` の場合は、マトリックス引数は次のようになる。

$$\begin{bmatrix} (1.1, 0.11) & (1.2, 0.12) \\ (2.1, 0.21) & (2.2, 0.22) \\ (3.1, 0.31) & (3.2, 0.32) \\ (4.1, 0.41) & (4.2, 0.42) \end{bmatrix}$$

`transa = 'T'`、`m = 4`、`n = 2`、`lda =5` の場合は、

マトリックス引数は次のようになる。

$$\begin{bmatrix} (1.1, 0.11) & (2.1, 0.21) & (3.1, 0.31) & (4.1, 0.41) \\ (1.2, 0.12) & (2.2, 0.22) & (3.2, 0.32) & (4.2, 0.42) \end{bmatrix}$$

`transa = 'C'`、`m = 4`、`n = 2`、`lda =5` の場合は、

マトリックス引数は次のようになる。

$$\begin{bmatrix} (1.1, -0.11) & (2.1, -0.21) & (3.1, -0.31) & (4.1, -0.41) \\ (1.2, -0.12) & (2.2, -0.22) & (3.2, -0.32) & (4.2, -0.42) \end{bmatrix}$$

リーディング・ディメンジョンの値として、2 次元配列の宣言で指定した行数とは異なる値を使用する場合は、特に注意が必要である。たとえば、上記の配列 *A* を `COMPLEX A (5, 4)` として宣言するものとする。

このとき、*transa* = 'N'、*m* = 3、*n* = 4、*lda* = 4 の場合は、マトリックス引数は次のようになる。

$$\begin{bmatrix} (1.1, 0.11) & (5.1, 0.51) & (4.2, 0.42) & (3.3, 0.33) \\ (2.1, 0.21) & (1.2, 0.12) & (5.2, 0.52) & (4.3, 0.43) \\ (3.1, 0.31) & (2.2, 0.22) & (1.3, 0.13) & (5.3, 0.53) \end{bmatrix}$$

コード例

B

本付録では、BLAS のルーチンおよび関数を使用するコード例を示す。

例 B-1　BLAS レベル 1 の関数の使用法

?dot の説明

次の例は、BLAS レベル 1 の関数 sdot の呼び出し方法を表す。この関数は、2 つの単精度実数ベクトル *x* と *y* のスカラー積を計算するベクトル・ベクトル演算を実行する。

パラメータ

n	ベクトル *x* と *y* の次数を指定する。
incx	*x* の成分に対する増分を指定する。
incy	*y* の成分に対する増分を指定する。

```
program dot_main
real x(10), y(10), sdot, res
integer n, incx, incy, i
external sdot
n = 5
incx = 2
incy = 1
do i = 1, 10
  x(i) = 2.0e0
  y(i) = 1.0e0
end do
```

続く ☛

例 B-1 BLAS レベル 1 の関数の使用法 (続き)

```
res = sdot (n, x, incx, y, incy)
print*, 'SDOT = ', res
end
```

このプログラムを実行すると、次の行が出力される。

```
SDOT = 10.000
```

例 B-2 BLAS レベル 1 のルーチンの使用法

?copy の説明

次の例は、BLAS レベル 1 のルーチン scopy の呼び出し方法を表す。このルーチンは、2 つの単精度実数ベクトル *x* を *y* にコピーするベクトル・ベクトル演算を実行する。

パラメータ

n	ベクトル *x* と *y* の次数を指定する。
incx	*x* の成分に対する増分を指定する。
incy	*y* の成分に対する増分を指定する。

```
program copy_main
real x(10), y(10)
integer n, incx, incy, i
n = 3
incx = 3
incy = 1
do i = 1, 10
  x(i) = i
end do
call scopy (n, x, incx, y, incy)
print*, 'Y = ', (y(i), i = 1, n)
end
```

このプログラムを実行すると、次の行が出力される。

```
Y = 1.00000 4.00000 7.00000
```

例 B-3　BLAS レベル 2 のルーチンの使用法

?ger の説明

次の例は、BLAS レベル 2 のルーチン sger の呼び出し方法を表す。このルーチンは、次に示すマトリックス・ベクトル演算を実行する。

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

パラメータ

alpha	スカラー *alpha* を指定する。
x	*m* 個の成分を持つベクトル。
y	*n* 個の成分を持つベクトル。
a	*m* 行 *n* 列のマトリックス。

```
program ger_main
real a(5,3), x(10), y(10), alpha
integer m, n, incx, incy, i, j, lda
m = 2
n = 3
lda = 5
incx = 2
incy = 1
alpha = 0.5
do i = 1, 10
  x(i) = 1.0
  y(i) = 1.0
end do
do i = 1, m
  do j = 1, n
    a(i,j) = j
  end do
end do
call sger (m, n, alpha, x, incx, y, incy, a, lda)
print*, 'Matrix A: '
do i = 1, m
  print*, (a(i,j), j = 1, n)
end do
end
```

続く ☛

例 B-3 BLAS レベル 2 のルーチンの使用法 (続き)

このプログラムを実行すると、次に示すマトリックス *a* が出力される。

マトリックス A:

1.50000 2.50000 3.50000

1.50000 2.50000 3.50000

例 B-4 BLAS レベル 3 のルーチンの使用法

次の例は、BLAS レベル 3 のルーチン ssymm の呼び出し方法を表す。このルーチンは、次に示すマトリックス・マトリックス演算を実行する。

?symm の説明

*c := alpha*a*b' + beta*c.*

パラメータ

alpha	スカラー *alpha* を指定する。
beta	スカラー *beta* を指定する。
a	対称マトリックス。
b	*m* 行 *n* 列のマトリックス。
c	*m* 行 *n* 列のマトリックス。

```
program symm_main
real a(3,3), b(3,2), c(3,3), alpha, beta
integer m, n, lda, ldb, ldc, i, j
character uplo, side
uplo = 'u'
side = 'l'
m = 3
n = 2
lda = 3
ldb = 3
ldc = 3
alpha = 0.5
beta = 2.0
```

続く ☛

例 B-4　BLAS レベル 3 のルーチンの使用法 (続き)

```
do i = 1, m
  do j = 1, m
    a(i,j) = 1.0
  end do
end do
do i = 1, m
  do j = 1, n
    c(i,j) = 1.0
    b(i,j) = 2.0
  end do
end do
call ssymm (side, uplo, m, n, alpha, a, lda, b, ldb, beta,
c, ldc)
print*, 'Matrix C: '
do i = 1, m
  print*, (c(i,j), j = 1, n)
end do
end
```

このプログラムを実行すると、次に示すマトリックス *c* が出力される。

マトリックス C:

5.00000 5.00000

5.00000 5.00000

5.00000 5.00000

例 B-5 BLAS レベル 1 の複素関数を C から呼び出す

次の例は、C プログラムから BLAS レベル 1 の複素関数 zdotc() を呼び出す方法を表す。この関数は、2 つの倍精度複素ベクトルのドット積を計算する。

この例では、複素数型のドット積が構造体 c に返される。

```
#define N 5
void main()
{

  int n, inca = 1, incb = 1, i;
  typedef struct{ double re; double im; } complex16;
  complex16 a[N], b[N], c;
  void zdotc();
  n = N;
  for( i = 0; i < n; i++ ){
    a[i].re = (double)i; a[i].im = (double)i * 2.0;
    b[i].re = (double)(n - i); b[i].im = (double)i * 2.0;
  }
  zdotc( &c, &n, a, &inca, b, &incb );
  printf( "The complex dot product is: ( %6.2f, %6.2f )\n",
c.re, c.im );
}
```

注：C プログラムから BLAS を直接呼び出す代わりに、CBLAS インターフェイスの使用も可能である。これは、C から BLAS を呼び出す方法としてサポートされている。CBLAS の詳細については、付録 C「BLAS に対する CBLAS インターフェイス」を参照。

BLAS に対する CBLAS インターフェイス

C

本付録では、CBLAS について説明する。CBLAS は、インテル ® MKL に導入された BLAS (Basic Linear Algebra Subprograms) に対する C インターフェイスである。

BLAS の場合と同じように、CBLAS インターフェイスには、以下に示すレベルの関数が含まれる。

- 「レベル 1 の CBLAS」(ベクトル・ベクトル演算)
- 「レベル 2 の CBLAS」(マトリックス・ベクトル演算)
- 「レベル 3 の CBLAS」(マトリックス・マトリックス演算)
- 「スパース CBLAS」(スパース・ベクトルでの演算)

C インターフェイスを得るには、Fortran ルーチン名の先頭に `cblas_` を付ける (たとえば、`dasum` は `cblas_dasum` となる)。すべての CBLAS 関数は、小文字で表す。

複素関数 `?dotc` と `?dotu` は、CBLAS のサブルーチン (void 関数) になる。これらの関数は、最後のパラメータとして追加される void ポインタを介して複素数型の結果を返す。これらの関数の CBLAS 名には、末尾に _sub を付ける。たとえば、BLAS 関数 `cdotc` は `cblas_cdotc_sub` に対応する。

CBLAS の引数

CBLAS 関数の引数は、以下の規則に従る。

- 入力引数は、`const` 修飾子で宣言する。
- 非複素数のスカラー入力引数は、値で渡される。
- 複素数のスカラー入力引数は、void ポインタとして渡される。
- 配列の引数は、アドレスで渡される。
- 出力のスカラー引数は、アドレスで渡される。
- BLAS のキャラクタ引数は、該当する列挙型で置き換えられる。

- レベル 2 とレベル 3 のルーチンは、最初の引数として型 CBLAS_ORDER の追加パラメータをとる。このパラメータは、2 次元配列が行主体 (CblasRowMajor) か列主体 (CblasColMajor) のどちらであるかを指定する。

列挙型

CBLAS インターフェイスは、以下に示す列挙型を使用する。

```
enum CBLAS_ORDER {
   CblasRowMajor=101,      /* row-major arrays */
   CblasColMajor=102};     /* column-major arrays */
enum CBLAS_TRANSPOSE {
   CblasNoTrans=111,       /* trans='N' */
   CblasTrans=112,         /* trans='T' */
   CblasConjTrans=113};    /* trans='C' */
enum CBLAS_UPLO {
   CblasUpper=121,         /* uplo ='U' */
   CblasLower=122};        /* uplo ='L' */
enum CBLAS_DIAG {
   CblasNonUnit=131,       /* diag ='N' */
   CblasUnit=132};         /* diag ='U' */
enum CBLAS_SIDE {
   CblasLeft=141,          /* side ='L' */
   CblasRight=142};        /* side ='R' */
```

レベル 1 の CBLAS

基本的なベクトル・ベクトル演算を実行する「BLAS レベル 1 のルーチンと関数」へのインターフェイスである。

?asum

```
float cblas_sasum(const int N, const float *X, const int incX);
double cblas_dasum(const int N, const double *X, const int incX);
float cblas_scasum(const int N, const void *X, const int incX);
double cblas_dzasum(const int N, const void *X, const int incX);
```

?axpy

```
void cblas_saxpy(const int N, const float alpha, const float *X,
const int incX, float *Y, const int incY);
void cblas_daxpy(const int N, const double alpha, const double *X,
const int incX, double *Y, const int incY);
void cblas_caxpy(const int N, const void *alpha, const void *X,
const int incX, void *Y, const int incY);
void cblas_zaxpy(const int N, const void *alpha, const void *X,
const int incX, void *Y, const int incY);
```

?copy

```
void cblas_scopy(const int N, const float *X, const int incX, float
*Y, const int incY);
void cblas_dcopy(const int N, const double *X, const int incX,
double *Y, const int incY);
void cblas_ccopy(const int N, const void *X, const int incX, void
*Y, const int incY);
void cblas_zcopy(const int N, const void *X, const int incX, void
*Y, const int incY);
```

?dot

```
float cblas_sdot(const int N, const float *X, const int incX,
const float *Y, const int incY);
double cblas_ddot(const int N, const double *X, const int incX,
const double *Y, const int incY);
```

?sdot

```
float cblas_sdsdot(const int N, const float *SB, const float *SX,
const int incX, const float *SY, const int incY);
double cblas_dsdot(const int N, const float *SX, const int incX,
const float *SY, const int incY);
```

?dotc

```
void cblas_cdotc_sub(const int N, const void *X, const int incX,
const void *Y, const int incY, void *dotc);
void cblas_zdotc_sub(const int N, const void *X, const int incX,
const void *Y, const int incY, void *dotc);
```

?dotu

```
void cblas_cdotu_sub(const int N, const void *X, const int incX,
const void *Y, const int incY, void *dotu);
void cblas_zdotu_sub(const int N, const void *X, const int incX,
const void *Y, const int incY, void *dotu);
```

?nrm2

```
float cblas_snrm2(const int N, const float *X, const int incX);
double cblas_dnrm2(const int N, const double *X, const int incX);
float cblas_scnrm2(const int N, const void *X, const int incX);
double cblas_dznrm2(const int N, const void *X, const int incX);
```

?rot

```
void cblas_srot(const int N, float *X, const int incX, float *Y,
const int incY, const float c, const float s);
void cblas_drot(const int N, double *X, const int incX, double
*Y,const int incY, const double c, const double s);
```

?rotg

```
void cblas_srotg(float *a, float *b, float *c, float *s);
void cblas_drotg(double *a, double *b, double *c, double *s);
```

?rotm

```
void cblas_srotm(const int N, float *X, const int incX, float *Y,
const int incY, const float *P);
void cblas_drotm(const int N, double *X, const int incX, double *Y,
const int incY, const double *P);
```

?rotmg

```
void cblas_srotmg(float *d1, float *d2, float *b1, const float b2,
float *P);
void cblas_drotmg(double *d1, double *d2, double *b1, const double
b2, double *P);
```

?scal

```
void cblas_sscal(const int N, const float alpha, float *X, const int
incX);
void cblas_dscal(const int N, const double alpha, double *X, const
int incX);
void cblas_cscal(const int N, const void *alpha, void *X, const int
incX);
void cblas_zscal(const int N, const void *alpha, void *X, const int
incX);
void cblas_csscal(const int N, const float alpha, void *X, const int
incX);
void cblas_zdscal(const int N, const double alpha, void *X, const
int incX);
```

?swap

```
void cblas_sswap(const int N, float *X, const int incX, float *Y,
const int incY);
void cblas_dswap(const int N, double *X, const int incX, double *Y,
const int incY);
void cblas_cswap(const int N, void *X, const int incX, void *Y,
const int incY);
void cblas_zswap(const int N, void *X, const int incX, void *Y,
const int incY);
```

i?amax

```
CBLAS_INDEX cblas_isamax(const int N, const float *X, const int
incX);
CBLAS_INDEX cblas_idamax(const int N, const double *X, const int
incX);
CBLAS_INDEX cblas_icamax(const int N, const void *X, const int
incX);
CBLAS_INDEX cblas_izamax(const int N, const void *X, const int
incX);
```

i?amin

```
CBLAS_INDEX cblas_isamin(const int N, const float *X, const int
incX);
CBLAS_INDEX cblas_idamin(const int N, const double *X, const int
incX);
CBLAS_INDEX cblas_icamin(const int N, const void *X, const int
incX);
CBLAS_INDEX cblas_izamin(const int N, const void *X, const int
incX);
```

レベル 2 の CBLAS

基本的なマトリックス・ベクトル演算を実行する「BLAS レベル 2 のルーチン」へのインターフェイスである。このグループに属する各 C ルーチンは、型 CBLAS_ORDER の追加パラメータ (最初の引数) をとる。このパラメータは、2 次元の配列が、列主体か行主体のどちらの格納方式を使用するかを決定する。

?gbmv

```
void cblas_sgbmv(const enum CBLAS_ORDER order, const enum
CBLAS_TRANSPOSE TransA, const int M, const int N, const int KL,
const int KU, const float alpha, const float *A, const int lda,
const float *X, const int incX, const float beta, float *Y, const
int incY);
void cblas_dgbmv(const enum CBLAS_ORDER order, const enum
CBLAS_TRANSPOSE TransA, const int M, const int N, const int KL,
const int KU, const double alpha, const double *A, const int lda,
const double *X, const int incX, const double beta, double *Y, const
int incY);
void cblas_cgbmv(const enum CBLAS_ORDER order, const enum
CBLAS_TRANSPOSE TransA, const int M, const int N, const int KL,
const int KU, const void *alpha, const void *A, const int lda, const
void *X, const int incX, const void *beta, void *Y, const int incY);
void cblas_zgbmv(const enum CBLAS_ORDER order, const enum
CBLAS_TRANSPOSE TransA, const int M, const int N, const int KL,
const int KU, const void *alpha, const void *A, const int lda, const
void *X, const int incX, const void *beta, void *Y, const int incY);
```

?gemv

```
void cblas_sgemv(const enum CBLAS_ORDER order, const enum
CBLAS_TRANSPOSE TransA, const int M, const int N, const float alpha,
const float *A, const int lda, const float *X, const int incX, const
float beta, float *Y, const int incY);
void cblas_dgemv(const enum CBLAS_ORDER order, const enum
CBLAS_TRANSPOSE TransA, const int M, const int N, const double
alpha, const double *A, const int lda, const double *X, const int
incX, const double beta, double *Y, const int incY);
void cblas_cgemv(const enum CBLAS_ORDER order, const enum
CBLAS_TRANSPOSE TransA, const int M, const int N, const void *alpha,
const void *A, const int lda, const void *X, const int incX, const
void *beta, void *Y, const int incY);
void cblas_zgemv(const enum CBLAS_ORDER order, const enum
CBLAS_TRANSPOSE TransA, const int M, const int N, const void *alpha,
const void *A, const int lda, const void *X, const int incX, const
void *beta, void *Y, const int incY);
```

?ger

```
void cblas_sger(const enum CBLAS_ORDER order, const int M, const int N, const float alpha, const float *X, const int incX, const float *Y, const int incY, float *A, const int lda);
void cblas_dger(const enum CBLAS_ORDER order, const int M, const int N, const double alpha, const double *X, const int incX, const double *Y, const int incY, double *A, const int lda);
```

?gerc

```
void cblas_cgerc(const enum CBLAS_ORDER order, const int M, const int N, const void *alpha, const void *X, const int incX, const void *Y, const int incY, void *A, const int lda);
void cblas_zgerc(const enum CBLAS_ORDER order, const int M, const int N, const void *alpha, const void *X, const int incX, const void *Y, const int incY, void *A, const int lda);
```

?geru

```
void cblas_cgeru(const enum CBLAS_ORDER order, const int M, const int N, const void *alpha, const void *X, const int incX, const void *Y, const int incY, void *A, const int lda);
void cblas_zgeru(const enum CBLAS_ORDER order, const int M, const int N, const void *alpha, const void *X, const int incX, const void *Y, const int incY, void *A, const int lda);
```

?hbmv

```
void cblas_chbmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const int N, const int K, const void *alpha, const void *A, const int lda, const void *X, const int incX, const void *beta, void *Y, const int incY);
void cblas_zhbmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const int N, const int K, const void *alpha, const void *A, const int lda, const void *X, const int incX, const void *beta, void *Y, const int incY);
```

?hemv

```
void cblas_chemv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const int N, const void *alpha, const void *A, const int lda, const void *X, const int incX, const void *beta, void *Y, const int incY);
void cblas_zhemv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const int N, const void *alpha, const void *A, const int lda, const void *X, const int incX, const void *beta, void *Y, const int incY);
```

?her

```
void cblas_cher(const enum CBLAS_ORDER order, const enum CBLAS_UPLO
Uplo, const int N, const float alpha, const void *X, const int incX,
void *A, const int lda);
void cblas_zher(const enum CBLAS_ORDER order, const enum CBLAS_UPLO
Uplo, const int N, const double alpha, const void *X, const int
incX, void *A, const int lda);
```

?her2

```
void cblas_cher2(const enum CBLAS_ORDER order, const enum CBLAS_UPLO
Uplo, const int N, const void *alpha, const void *X, const int incX,
const void *Y, const int incY, void *A, const int lda);
void cblas_zher2(const enum CBLAS_ORDER order, const enum CBLAS_UPLO
Uplo, const int N, const void *alpha, const void *X, const int incX,
const void *Y, const int incY, void *A, const int lda);
```

?hpmv

```
void cblas_chpmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO
Uplo, const int N, const void *alpha, const void *Ap, const void *X,
const int incX, const void *beta, void *Y, const int incY);
void cblas_zhpmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO
Uplo, const int N, const void *alpha, const void *Ap, const void *X,
const int incX, const void *beta, void *Y, const int incY);
```

?hpr

```
void cblas_chpr(const enum CBLAS_ORDER order, const enum CBLAS_UPLO
Uplo, const int N, const float alpha, const void *X, const int incX,
void *A);
void cblas_zhpr(const enum CBLAS_ORDER order, const enum CBLAS_UPLO
Uplo, const int N, const double alpha, const void *X, const int
incX, void *A);
```

?hpr2

```
void cblas_chpr2(const enum CBLAS_ORDER order, const enum CBLAS_UPLO
Uplo, const int N, const void *alpha, const void *X, const int incX,
const void *Y, const int incY, void *Ap);
void cblas_zhpr2(const enum CBLAS_ORDER order, const enum CBLAS_UPLO
Uplo, const int N, const void *alpha, const void *X, const int incX,
const void *Y, const int incY, void *Ap);
```

?sbmv

```
void cblas_ssbmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO
Uplo, const int N, const int K, const float alpha, const float *A,
const int lda, const float *X, const int incX, const float beta,
float *Y, const int incY);
void cblas_dsbmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO
Uplo, const int N, const int K, const double alpha, const double *A,
const int lda, const double *X, const int incX, const double beta,
double *Y, const int incY);
```

?spmv

```
void cblas_sspmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO
Uplo, const int N, const float alpha, const float *Ap, const float
*X, const int incX, const float beta, float *Y, const int incY);
void cblas_dspmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO
Uplo, const int N, const double alpha, const double *Ap, const
double *X, const int incX, const double beta, double *Y, const int
incY);
```

?spr

```
void cblas_sspr(const enum CBLAS_ORDER order, const enum CBLAS_UPLO
Uplo, const int N, const float alpha, const float *X, const int
incX, float *Ap);
void cblas_dspr(const enum CBLAS_ORDER order, const enum CBLAS_UPLO
Uplo, const int N, const double alpha, const double *X, const int
incX, double *Ap);
```

?spr2

```
void cblas_sspr2(const enum CBLAS_ORDER order, const enum CBLAS_UPLO
Uplo, const int N, const float alpha, const float *X, const int
incX, const float *Y, const int incY, float *A);
void cblas_dspr2(const enum CBLAS_ORDER order, const enum CBLAS_UPLO
Uplo, const int N, const double alpha, const double *X, const int
incX, const double *Y, const int incY, double *A);
```

?symv

```
void cblas_ssymv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO
Uplo, const int N, const float alpha, const float *A, const int lda,
const float *X, const int incX, const float beta, float *Y, const
int incY);
void cblas_dsymv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO
Uplo, const int N, const double alpha, const double *A, const int
lda, const double *X, const int incX, const double beta, double *Y,
const int incY);
```

?syr

```
void cblas_ssyr(const enum CBLAS_ORDER order, const enum CBLAS_UPLO
Uplo, const int N, const float alpha, const float *X, const int
incX, float *A, const int lda);
void cblas_dsyr(const enum CBLAS_ORDER order, const enum CBLAS_UPLO
Uplo, const int N, const double alpha, const double *X, const int
incX, double *A, const int lda);
```

?syr2

```
void cblas_ssyr2(const enum CBLAS_ORDER order, const enum CBLAS_UPLO
Uplo, const int N, const float alpha, const float *X, const int
incX, const float *Y, const int incY, float *A, const int lda);
void cblas_dsyr2(const enum CBLAS_ORDER order, const enum CBLAS_UPLO
Uplo, const int N, const double alpha, const double *X, const int
incX, const double *Y, const int incY, double *A, const int lda);
```

?tbmv

```
void cblas_stbmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO
Uplo, const enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag,
const int N, const int K, const float *A, const int lda, float *X,
const int incX);
void cblas_dtbmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO
Uplo, const enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag,
const int N, const int K, const double *A, const int lda, double *X,
const int incX);
void cblas_ctbmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO
Uplo, const enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag,
const int N, const int K, const void *A, const int lda, void *X,
const int incX);
void cblas_ztbmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO
Uplo, const enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag,
const int N, const int K, const void *A, const int lda, void *X,
const int incX);
```

?tbsv

```
void cblas_stbsv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO
Uplo, const enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag,
const int N, const int K, const float *A, const int lda, float *X,
const int incX);
void cblas_dtbsv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO
Uplo, const enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag,
const int N, const int K, const double *A, const int lda, double *X,
const int incX);
void cblas_ctbsv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO
Uplo, const enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag,
const int N, const int K, const void *A, const int lda, void *X,
const int incX);
void cblas_ztbsv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO
Uplo, const enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag,
const int N, const int K, const void *A, const int lda, void *X,
const int incX);
```

?tpmv

```
void cblas_stpmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO
Uplo, const enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag,
const int N, const float *Ap, float *X, const int incX);
void cblas_dtpmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO
Uplo, const enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag,
const int N, const double *Ap, double *X, const int incX);
void cblas_ctpmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO
Uplo, const enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag,
const int N,const void *Ap, void *X, const int incX);
void cblas_ztpmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO
Uplo, const enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag,
const int N,const void *Ap, void *X, const int incX);
```

?tpsv

```
void cblas_stpsv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO
Uplo, const enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag,
const int N,const float *Ap, float *X, const int incX);
void cblas_dtpsv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO
Uplo, const enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag,
const int N,const double *Ap, double *X, const int incX);
void cblas_ctpsv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO
Uplo, const enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag,
const int N,const void *Ap, void *X, const int incX);
void cblas_ztpsv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO
Uplo, const enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag,
const int N,const void *Ap, void *X, const int incX);
```

?trmv

```
void cblas_strmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO
Uplo, const enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag,
const int N,const float *A, const int lda, float *X, const int
incX);
void cblas_dtrmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO
Uplo, const enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag,
const int N,const double *A, const int lda, double *X, const int
incX);
void cblas_ctrmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO
Uplo, const enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag,
const int N,const void *A, const int lda, void *X, const int incX);
void cblas_ztrmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO
Uplo, const enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag,
const int N,const void *A, const int lda, void *X, const int incX);
```

?trsv

```
void cblas_strsv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO
Uplo, const enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag,
const int N,const float *A, const int lda, float *X, const int
incX);
void cblas_dtrsv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO
Uplo, const enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag,
const int N,const double *A, const int lda, double *X, const int
incX);
void cblas_ctrsv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO
Uplo, const enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag,
const int N,const void *A, const int lda, void *X, const int incX);
void cblas_ztrsv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO
Uplo, const enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag,
const int N,const void *A, const int lda, void *X, const int incX);
```

レベル 3 の CBLAS

基本的なマトリックス・マトリックス演算を実行する「BLAS レベル 3 のルーチン」へのインターフェイスである。このグループに属する各 C ルーチンは、型 CBLAS_ORDER の追加パラメータ (最初の引数) をとる。このパラメータは、2 次元の配列が、列主体か行主体のどちらの格納方式を使用するかを決定する。

?gemm

```
void cblas_sgemm(const enum CBLAS_ORDER Order, const enum
CBLAS_TRANSPOSE TransA, const enum CBLAS_TRANSPOSE TransB, const int
M, const int N, const int K, const float alpha, const float *A, const
int lda, const float *B, const int ldb, const float beta, float *C,
const int ldc);
```

```
void cblas_dgemm(const enum CBLAS_ORDER Order, const enum
CBLAS_TRANSPOSE TransA, const enum CBLAS_TRANSPOSE TransB, const int
M, const int N, const int K, const double alpha, const double *A,
const int lda, const double *B, const int ldb, const double beta,
double *C, const int ldc);
```

```
void cblas_cgemm(const enum CBLAS_ORDER Order, const enum
CBLAS_TRANSPOSE TransA, const enum CBLAS_TRANSPOSE TransB, const int
M, const int N, const int K, const void *alpha, const void *A, const
int lda, const void *B, const int ldb, const void *beta, void *C,
const int ldc);
```

```
void cblas_zgemm(const enum CBLAS_ORDER Order, const enum
CBLAS_TRANSPOSE TransA, const enum CBLAS_TRANSPOSE TransB, const int
M, const int N, const int K, const void *alpha, const void *A, const
int lda, const void *B, const int ldb, const void *beta, void *C,
const int ldc);
```

?hemm

```
void cblas_chemm(const enum CBLAS_ORDER Order, const enum CBLAS_SIDE
Side, const enum CBLAS_UPLO Uplo, const int M, const int N, const
void *alpha, const void *A, const int lda, const void *B, const int
ldb, const void *beta, void *C, const int ldc);
```

```
void cblas_zhemm(const enum CBLAS_ORDER Order, const enum CBLAS_SIDE
Side, const enum CBLAS_UPLO Uplo, const int M, const int N, const
void *alpha, const void *A, const int lda, const void *B, const int
ldb, const void *beta, void *C, const int ldc);
```

?herk

```
void cblas_cherk(const enum CBLAS_ORDER Order, const enum CBLAS_UPLO
Uplo, const enum CBLAS_TRANSPOSE Trans, const int N, const int K,
const float alpha, const void *A, const int lda, const float beta,
void *C, const int ldc);
void cblas_zherk(const enum CBLAS_ORDER Order, const enum CBLAS_UPLO
Uplo, const enum CBLAS_TRANSPOSE Trans, const int N, const int K,
const double alpha, const void *A, const int lda, const double beta,
void *C, const int ldc);
```

?herk

```
void cblas_cher2k(const enum CBLAS_ORDER Order, const enum
CBLAS_UPLO Uplo, const enum CBLAS_TRANSPOSE Trans, const int N,
const int K, const void *alpha, const void *A, const int lda, const
void *B, const int ldb, const float beta, void *C, const int ldc);
void cblas_zher2k(const enum CBLAS_ORDER Order, const enum
CBLAS_UPLO Uplo, const enum CBLAS_TRANSPOSE Trans, const int N,
const int K, const void *alpha, const void *A, const int lda, const
void *B, const int ldb, const double beta, void *C, const int ldc);
```

?symm

```
void cblas_ssymm(const enum CBLAS_ORDER Order, const enum CBLAS_SIDE
Side, const enum CBLAS_UPLO Uplo, const int M, const int N, const
float alpha, const float *A, const int lda, const float *B, const
int ldb, const float beta, float *C, const int ldc);
void cblas_dsymm(const enum CBLAS_ORDER Order, const enum CBLAS_SIDE
Side, const enum CBLAS_UPLO Uplo, const int M, const int N, const
double alpha, const double *A, const int lda, const double *B, const
int ldb, const double beta, double *C, const int ldc);
void cblas_csymm(const enum CBLAS_ORDER Order, const enum CBLAS_SIDE
Side, const enum CBLAS_UPLO Uplo, const int M, const int N, const
void *alpha, const void *A, const int lda, const void *B, const int
ldb, const void *beta, void *C, const int ldc);
void cblas_zsymm(const enum CBLAS_ORDER Order, const enum CBLAS_SIDE
Side, const enum CBLAS_UPLO Uplo, const int M, const int N, const
void *alpha, const void *A, const int lda, const void *B, const int
ldb, const void *beta, void *C, const int ldc);
```

?syrk

```
void cblas_ssyrk(const enum CBLAS_ORDER Order, const enum CBLAS_UPLO
Uplo, const enum CBLAS_TRANSPOSE Trans, const int N, const int K,
const float alpha, const float *A, const int lda, const float beta,
float *C, const int ldc);
void cblas_dsyrk(const enum CBLAS_ORDER Order, const enum CBLAS_UPLO
Uplo, const enum CBLAS_TRANSPOSE Trans, const int N, const int K,
const double alpha, const double *A, const int lda, const double
beta, double *C, const int ldc);
void cblas_csyrk(const enum CBLAS_ORDER Order, const enum CBLAS_UPLO
Uplo, const enum CBLAS_TRANSPOSE Trans, const int N, const int K,
const void *alpha, const void *A, const int lda, const void *beta,
void *C, const int ldc);
void cblas_zsyrk(const enum CBLAS_ORDER Order, const enum CBLAS_UPLO
Uplo, const enum CBLAS_TRANSPOSE Trans, const int N, const int K,
const void *alpha, const void *A, const int lda, const void *beta,
void *C, const int ldc);
```

?syr2k

```
void cblas_ssyr2k(const enum CBLAS_ORDER Order, const enum
CBLAS_UPLO Uplo, const enum CBLAS_TRANSPOSE Trans, const int N,
const int K, const float alpha,const float *A, const int lda, const
float *B, const int ldb, const float beta, float *C, const int ldc);
void cblas_dsyr2k(const enum CBLAS_ORDER Order, const enum
CBLAS_UPLO Uplo, const enum CBLAS_TRANSPOSE Trans, const int N,
const int K, const double alpha, const double *A, const int lda,
const double *B, const int ldb, const double beta, double *C, const
int ldc);
void cblas_csyr2k(const enum CBLAS_ORDER Order, const enum
CBLAS_UPLO Uplo, const enum CBLAS_TRANSP SE Trans, const int N,
const int K, const void *alpha,const void *A, const int lda, const
void *B, const int ldb, const void *beta, void *C, const int ldc);
void cblas_zsyr2k(const enum CBLAS_ORDER Order, const enum
CBLAS_UPLO Uplo, const enum CBLAS_TRANSPOSE Trans, const int N,
const int K, const void *alpha, const void *A, const int lda, const
void *B, const int ldb, const void *beta, void *C, const int ldc);
```

?trmm

```
void cblas_strmm(const enum CBLAS_ORDER Order, const enum CBLAS_SIDE
Side, const enum CBLAS_UPLO Uplo, const enum CBLAS_TRANSPOSE TransA,
const enum CBLAS_DIAG Diag, const int M, const int N, const float
alpha, const float *A, const int lda, float *B, const int ldb);
void cblas_dtrmm(const enum CBLAS_ORDER Order, const enum CBLAS_SIDE
Side, const enum CBLAS_UPLO Uplo, const enum CBLAS_TRANSPOSE TransA,
const enum CBLAS_DIAG Diag, const int M, const int N, const double
alpha, const double *A, const int lda, double *B, const int ldb);
void cblas_ctrmm(const enum CBLAS_ORDER Order, const enum CBLAS_SIDE
Side, const enum CBLAS_UPLO Uplo, const enum CBLAS_TRANSPOSE TransA,
const enum CBLAS_DIAG Diag, const int M, const int N, const void
*alpha, const void *A, const int lda, void *B, const int ldb);
void cblas_ztrmm(const enum CBLAS_ORDER Order, const enum CBLAS_SIDE
Side, const enum CBLAS_UPLO Uplo, const enum CBLAS_TRANSPOSE TransA,
const enum CBLAS_DIAG Diag, const int M, const int N, const void
*alpha, const void *A, const int lda, void *B, const int ldb);
```

?trsm

```
void cblas_strsm(const enum CBLAS_ORDER Order, const enum CBLAS_SIDE
Side, const enum CBLAS_UPLO Uplo, const enum CBLAS_TRANSPOSE TransA,
const enum CBLAS_DIAG Diag, const int M, const int N, const float
alpha, const float *A, const int lda, float *B, const int ldb);
void cblas_dtrsm(const enum CBLAS_ORDER Order, const enum CBLAS_SIDE
Side, const enum CBLAS_UPLO Uplo, const enum CBLAS_TRANSPOSE TransA,
const enum CBLAS_DIAG Diag, const int M, const int N, const double
alpha, const double *A, const int lda, double *B, const int ldb);
void cblas_ctrsm(const enum CBLAS_ORDER Order, const enum CBLAS_SIDE
Side, const enum CBLAS_UPLO Uplo, const enum CBLAS_TRANSPOSE TransA,
const enum CBLAS_DIAG Diag, const int M, const int N, const void
*alpha, const void *A, const int lda, void *B, const int ldb);
void cblas_ztrsm(const enum CBLAS_ORDER Order, const enum CBLAS_SIDE
Side, const enum CBLAS_UPLO Uplo, const enum CBLAS_TRANSPOSE TransA,
const enum CBLAS_DIAG Diag, const int M, const int N, const void
*alpha, const void *A, const int lda, void *B, const int ldb);
```

スパース CBLAS

「スパース BLAS のルーチンと関数」に対するインターフェイスであり、圧縮形式で格納されたスパース・ベクトルに共通的なベクトル演算を実行する。

すべてのインデックス・パラメータ *indx* は C タイプで表記され [0..*N*-1] の範囲を取る。

?axpyi

```
void cblas_saxpyi(const int N, const float alpha,
const float *X, const int *indx, float *Y);
void cblas_daxpyi(const int N, const double alpha,
const double *X, const int *indx, double *Y);
void cblas_caxpyi(const int N, const void *alpha,
const void *X, const int *indx, void *Y);
void cblas_zaxpyi(const int N, const void *alpha,
const void *X, const int *indx, void *Y);
```

?doti

```
float  cblas_sdoti(const int N, const float *X,
const int *indx, const float *Y);
double cblas_ddoti(const int N, const double *X,
const int *indx, const double *Y);
```

?dotci

```
void   cblas_cdotci_sub(const int N, const void *X, const int *indx,
const void *Y, void *dotui);
void   cblas_zdotci_sub(const int N, const void *X, const int *indx,
const void *Y, void *dotui);
```

?dotui

```
void   cblas_cdotui_sub(const int N, const void *X, const int *indx,
const void *Y, void *dotui);
void   cblas_zdotui_sub(const int N, const void *X, const int *indx,
const void *Y, void *dotui);
```

?gthr

```
void cblas_sgthr(const int N, const float *Y, float *X,
const int *indx);
void cblas_dgthr(const int N, const double *Y, double *X,
const int *indx);
void cblas_cgthr(const int N, const void *Y, void *X,
const int *indx);
void cblas_zgthr(const int N, const void *Y, void *X,
const int *indx);
```

?gthrz

```
void cblas_sgthrz(const int N, float *Y, float *X,
const int *indx);
void cblas_dgthrz(const int N, double *Y, double *X,
const int *indx);
void cblas_cgthrz(const int N, void *Y, void *X,
const int *indx);
void cblas_zgthrz(const int N, void *Y, void *X,
const int *indx);
```

?roti

```
void cblas_sroti(const int N, float *X, const int *indx,
float *Y, const float c, const float s);
void cblas_droti(const int N, double *X, const int *indx,
double *Y, const double c, const double s);
```

?sctr

```
void cblas_ssctr(const int N, const float *X, const int *indx, float
*Y);
void cblas_dsctr(const int N, const double *X, const int *indx,
double *Y);
void cblas_csctr(const int N, const void *X, const int *indx, void
*Y);
void cblas_zsctr(const int N, const void *X, const int *indx, void
*Y);
```

用語集

A^H	一般マトリックス A の共役を表す。 共役マトリックスも参照。
A^T	一般マトリックス A の転置を表す。 転置も参照。
バンド・ マトリックス	$\|i-j\| > l$ に対して $a_{ij} = 0$ となる m 行 n 列の一般マトリックス A。ここで、$1 < l < \min(m, n)$。たとえば、三角マトリックスは、いずれもバンド・マトリックスである。
バンド格納	バンド・マトリックスに対する特殊な格納形式。マトリックスは、2 次元配列に格納される。このとき、マトリックスの各列は配列の対応する列に、マトリックスの各**対角成分**は配列の各行に格納される。
BLAS	Basic Linear Algebra の略。これらのサブプログラムには、ベクトル演算、マトリックス - ベクトル演算、マトリックス - マトリックス演算が導入されている。
Bunch-Kaufman 因子分解	実数称マトリックスあるいは複素エルミート・マトリックス A を形式 $A = PUDU^HP^T$（または $A = PLDL^HP^T$）で表したもの。P は置換マトリックス、U と L は単位対角成分を持つ上および下三角マトリックス、D は 1×1 と 2×2 の対角ブロックを持つエルミート・ブロック対角マトリックスである。U と L は、D の 2×2 のブロックに対応する 2×2 の単位対角ブロックを持つ。

c	ルーチン名の最初の文字に使われている場合は、単精度の複素数データ・タイプを使用するのを示す。
CBLAS	BLAS に対する C インターフェイス。BLAS を参照。
コルスキー因子分解	正値対称マトリックス、あるいは複素数データの場合は正値エルミート・マトリックス A を、形式 $A = U^H U$ か $A = LL^H$ で表したもの。L は下三角マトリックス、U は上三角マトリックスである。
条件数	所定の正方マトリックス A に対して、次のように定義される数 $\kappa(A)$：$\kappa(A) = \|A\| \, \|A^{-1}\|$。
共役マトリックス	所定の一般マトリックス A に対して、次のように定義されるマトリックス A^H：$(A^H)_{ij} = (a_{ji})^*$。
共役数	複素数 $z = a + bi$ の共役は、$z^* = a - bi$ になる。
d	ルーチン名の最初の文字に使われている場合は、倍精度の実数データ・タイプを使用するのを示す。
ドット積	$x \cdot y$ で表され、特定のベクトル x と y に対して次のように定義される数：$x \cdot y = \sum_i x_i y_i$ x_i と y_i はそれぞれ、x と y の i 番目の要素を表す。
倍精度	浮動小数点データ・タイプの 1 つ。インテル® アーキテクチャ・プロセッサでは、このデータ・タイプは $2.23*10^{-308} < \|x\| < 1.79*10^{308}$ の範囲にある実数 x を格納できる。 このデータ・タイプに対するマシン精度 ε は、およそ 10^{-15} になる。つまり、倍精度の数は、通常は、15 以下の有効 10 進桁を格納できる。 詳細は、『Pentium® Processor Family Developer's Manual, Volume 3: Architecture and Programming Manual』を参照。
固有値	固有値問題を参照。

固有値問題	所定の正方マトリックス A に対して、$Ax = \lambda x$ となるような非ゼロのベクトル x と値 λ を求める問題。値 λ は、マトリックス A の**固有値**と呼ばれ、ベクトル x はマトリックス A の**固有ベクトル**と呼ばれる。
固有ベクトル	固有値問題を参照。
基本リフレクタ（Householder マトリックス）	一般形式 $H = I - \tau vv^T$ のマトリックス。v は列ベクトル、τ はスカラーである。LAPACK では、基本リフレクタは、たとえばマトリックス Q を QR 因子分解で表すために使用する（マトリックス Q は、基本リフレクタの積として表す）。
因子分解	マトリックスをマトリックス同士の積として表すこと。Bunch-Kaufman 因子分解、コルスキー因子分解 LU 因子分解、LQ 因子分解、QR 因子分解、Schur 因子分解も参照。
FFT	Fast Fourier Transforms（高速フーリエ変換）の略。本書の第 3 章を参照。
フル格納	あらゆる種類のマトリックスの格納が可能な格納形式。マトリックス A は、2 次元の配列 `a` に格納される。このとき、マトリックスの要素 a_{ij} は、配列の要素 `a(i, j)` に格納される。
エルミート・マトリックス	その共役マトリックス A^H に等しい正方マトリックス A。共役 A^H は、次のように定義される：$(A^H)_{ij} = (a_{ji})^*$。
I	単位マトリックスを参照。
単位マトリックス	その対角要素が 1、対角線外の要素が 0 である正方マトリックス I。任意のマトリックス A に対し、$AI = A$ と $IA = A$。
インプレース	演算の修飾子。演算をインプレースで実行する関数では、その入力を配列から読み取り、その出力を同じ配列に返す。

逆マトリックス	A^{-1} のように表し、所定の正方マトリックス A に対して次のように定義されるマトリックス：$AA^{-1} = A^{-1}A = I$ 特異マトリックス A に対しては、A^{-1} は存在しない。
LQ 因子分解	m 行 n 列のマトリックス A を、$A = LQ$ あるいは $A = (L\ 0)\ Q$ として表したもの。Q は、n 行 n 列の直交（ユニタリ）マトリックスである。$m \le n$ に対しては、L は実数の対角要素を持つ m 行 m 列の下三角マトリックスである。$m > n$ に対しては、次のようになる。 $L = \begin{bmatrix} L_1 \\ L_2 \end{bmatrix}$ L_1 は、n 行 n 列の下三角マトリックス、L_2 は矩形マトリックスである。
LU 因子分解	m 行 n 列の一般マトリックス A を、$A = PLU$ として表したもの。P は置換マトリックス、L は単位対角要素を持つ下三角マトリックス（$m > n$ の場合は、下台形マトリックス）、U は上三角マトリックス（$m < n$ の場合は上台形マトリックス）である。
マシン精度	マシンによる実数表現の精度を決定する数 ε。インテル® アーキテクチャでは、マシン精度は単精度のデータに対しては約 10^{-7} に、倍精度のデータに対しては約 10^{-15} になる。この精度は、マシンによる実数の表現における有効 10 進桁数も決定する。倍精度と単精度も参照。
MKL	Math Kernel Library（マス・カーネル・ライブラリ）の略。
直交マトリックス	その転置と逆が等しくなる、つまり $A^T = A^{-1}$ であるような実数の正方マトリックス A。したがって、$AA^T = A^TA = I$。直交マトリックスでは、すべての固有値は絶対値 1 を持つ。
パックド格納	対称マトリックス、エルミート・マトリックス、三角マトリックスをよりコンパクトに格納できる格納形式。マトリックスの上三角または下三角が、列ごとに 1 次元配列にパックされる。

正値マトリックス	非ゼロの任意のベクトル x に対して、$Ax \cdot x >$ であるような正方マトリックス A。・は、ドット積を表す。
QR 因子分解	m 行 n 列のマトリックス A を、$A = QR$ として表したもの。Q は m 行 m 列の直交（ユニタリ）マトリックス、R は実数の対角要素を持つ n 行 n 列の上三角マトリックス（$m \geq n$ の場合）あるいは上台形マトリックス（$m < n$ の場合）である。
s	ルーチン名の最初の文字に使われている場合は、単精度の実数データ・タイプを使用することを示す。
Schur 因子分解	正方マトリックス A を $A = ZTZ^H$ の形式で表したもの。T は、Schur 形式と呼ばれる A の上疑似三角マトリックス（複素数 A に対しては、三角マトリックス）である。マトリックス Z は、直交マトリックス（複素数 A に対してはユニタリ・マトリックス）である。Z の列は、Schur ベクトルと呼ばれる。
単精度	浮動小数点データ・タイプの 1 つ。インテル ® アーキテクチャ・プロセッサでは、このデータ・タイプは $1.18*10^{-38} < \lvert x \rvert < 3.40*10^{38}$ の範囲にある実数 x を格納できる。 このデータ・タイプに対するマシン精度 ε は、およそ 10^{-7} になる。つまり、単精度の数は、通常は、7 以下の有効 10 進桁を格納できる。詳細は、『Pentium® Processor Family Developer's Manual, Volume 3: Architecture and Programming Manual』を参照。
特異マトリックス	その行列式がゼロであるようなマトリックス。A が特異マトリックスとすると、逆 A^{-1} は存在せず、また連立方程式 $Ax = b$ は独自の解を持たない（つまり、解が存在しないか無限数の解が存在する）。
特異値	所定の一般マトリックス A に対し、マトリックス AA^H の固有値として定義される数。SVD も参照。
SMP	Symmetric MultiProcessing（対称マルチプロセシング）の略。MKL では、SMP によって提供される並列化によって性能を改善している。

スパース BLAS	スパース・ベクトルに対して基本的なベクトル演算を実行するルーチン。スパース BLAS ルーチンは、ベクトルが粗であるのを利用し、ベクトルのゼロ以外の要素だけを格納できる。BLAS を参照。
スパース・ベクトル	成分の大半がゼロであるベクトル。
格納形式	マトリックスを格納する方法。フル格納、パックド格納、バンド格納を参照。
SVD	Singular Value Decomposition（特異値分解）の略。第 5 章の特異値分解も参照。
対称マトリックス	$a_{ij} = a_{ji}$ である正方マトリックス A。
転置	所定のマトリックス A の転置は、$(A^T)_{ij} = a_{ji}$ となるマトリックス A^T になる（A の行が A^T の列に、A の列が A^T の行になる）。
台形マトリックス	$A = (A_1 A_2)$ であるマトリックス A。A_1 は上三角マトリックス、A_2 は矩形マトリックスである。
三角マトリックス	その劣対角要素（優対角要素）すべてがゼロである場合、マトリックス A は上（下）三角マトリックスと呼ばれる。したがって、上三角マトリックスに対しては $i > j$ の場合に $a_{ji} = 0$ に、下三角マトリックスに対しては $i < j$ の場合に $a_{ji} = 0$ になる。
三重対角マトリックス	非ゼロ要素が、3 つの対角成分（主対角成分、最初の劣対角成分、最初の優対角成分）だけに存在するマトリックス。
ユニタリ・マトリックス	その共役と逆が等しくなる複素数型の正方マトリックス A。つまり、$A^H = A^{-1}$ となり、したがって $AA^H = A^H A = I$ となる。ユニタリ・マトリックスのすべての固有値は、絶対値 1 を持つ。
VML	Vector Mathematical Library (ベクトル数学ライブラリ) の略。 本書の第 6 章を参照。
z	ルーチン名の最初の文字に使われている場合は、倍精度の複素数データ・タイプを使用するのを示す。

索引

ルーチン

?asum, 2-5
?axpy, 2-6
?axpyi, 2-118
?bdsqr, 5-94, 5-98
?copy, 2-7
?dot, 2-8
?dotc, 2-10
?dotci, 2-121
?doti, 2-120
?dotu, 2-11
?dotui, 2-122
?fft1d, 3-4, 3-8
?fft1dc, 3-5, 3-10, 3-15
?fft2d, 3-19, 3-22, 3-28
?fft2dc, 3-20, 3-24, 3-29
?gbbrd, 5-79
?gbcon, 4-65
?gbmv, 2-26
?gbrfs, 4-95
?gbtf2, 6-21
?gbtrf, 4-10
?gbtrs, 4-36
?gebak, 5-193
?gebal, 5-190
?gebd2, 6-23
?gebrd, 5-76
?gecon, 4-63
?gees, 5-379
?geesx, 5-384
?geev, 5-390
?geevx, 5-394
?gehd2, 6-26
?gehrd, 5-178
?gelq2, 6-29
?gelqf, 5-25, 5-36
?gels, 5-279
?gelsd, 5-289
?gelss, 5-286
?gelsy, 5-282
?gemm, 2-85
?gemv, 2-29
?geql2, 6-31
?geqpf, 5-11, 5-14
?geqr2, 6-33
?geqrf, 5-8, 5-48, 5-60, 5-68, 5-71
?ger, 2-32
?gerc, 2-33
?gerfs, 4-92, 4-98
?gerq2, 6-35
?geru, 2-35
?gesc2, 6-37
?gesdd, 5-405
?gesvd, 5-400
?getc2, 6-39
?getf2, 6-40
?getrf, 4-7
?getri, 4-133
?getrs, 4-34
?ggbak, 5-233
?ggbal, 5-230
?gges, 5-482
?ggesx, 5-489
?ggev, 5-497
?ggevx, 5-502
?ggglm, 5-296
?gghrd, 5-226
?gglse, 5-293
?ggsvd, 5-409
?ggsvp, 5-267
?gtcon, 4-67
?gthr, 2-123
?gthrz, 2-124
?gttrf, 4-12
?gttrs, 4-38, 6-42
?hbev, 5-348
?hbevd, 5-353
?hbevx, 5-361
?hbgst, 5-169
?hbgv, 5-463
?hbgvd, 5-469

?hbgvx, 5-477
?hbtrd, 5-130
?hecon, 4-80
?heev, 5-301
?heevd, 5-306
?heevr, 5-322
?heevx, 5-313
?hegst, 5-160
?hegv, 5-419
?hegvd, 5-425
?hegvx, 5-434
?hemm, 2-88
?hemv, 2-40
?her, 2-42
?her2, 2-44
?her2k, 2-94
?herfs, 4-116
?herk, 2-91
?hetrd, 5-111
?hetrf, 4-25
?hetri, 4-141
?hetrs, 4-51
?hgeqz, 5-235
?hpcon, 4-84
?hpev, 5-329
?hpevd, 5-334
?hpevx, 5-342
?hpgst, 5-164
?hpgv, 5-442
?hpgvd, 5-448
?hpgvx, 5-456
?hpmv, 2-46
?hpr, 2-49
?hpr2, 2-51
?hprfs, 4-122
?hptrd, 5-122
?hptrf, 4-31
?hptri, 4-145
?hptrs, 4-55
?hsein, 5-199
?hseqr, 5-195
?labrd, 6-45
?lacgv, 6-1
?lacon, 6-48
?lacpy, 6-50
?lacrm, 6-2
?lacrt, 6-3
?ladiv, 6-51
?lae2, 6-52
?laebz, 6-54
?laed0, 6-60
?laed1, 6-64
?laed2, 6-67
?laed3, 6-70
?laed4, 6-73
?laed5, 6-74
?laed6, 6-75
?laed7, 6-78
?laed8, 6-83
?laed9, 6-87
?laeda, 6-89
?laein, 6-92
?laesy, 6-4
?laev2, 6-95
?laexc, 6-98
?lag2, 6-100
?lags2, 6-103
?lagtf, 6-105
?lagtm, 6-108
?lagts, 6-110
?lagv2, 6-113
?lahqr, 6-115
?lahrd, 6-118
?laic1, 6-121
?laln2, 6-124
?lals0, 6-127
?lalsa, 6-132
?lalsd, 6-136
?lamc1, 6-140
?lamc2, 6-141
?lamc3, 6-142
?lamc4, 6-143
?lamc5, 6-144
?lamch, 6-139
?lamrg, 6-145
?langb, 6-146
?lange, 6-148
?langt, 6-149
?lanhb, 6-154
?lanhe, 6-163
?lanhp, 6-158
?lanhs, 6-151
?lansb, 6-152
?lansp, 6-156

?lanst/?lanht, 6-159
?lansy, 6-161
?lantb, 6-164
?lantp, 6-167
?lantr, 6-169
?lanv2, 6-171
?lapll, 6-172
?lapmt, 6-174
?lapy2, 6-175
?lapy3, 6-176
?laqgb, 6-176
?laqge, 6-179
?laqp2, 6-181
?laqps, 6-183
?laqsb, 6-185
?laqsp, 6-187
?laqsy, 6-189
?laqtr, 6-191
?lar1v, 6-194
?lar2v, 6-196
?larf, 6-198
?larfb, 6-200
?larfg, 6-202
?larft, 6-204
?larfx, 6-207
?largv, 6-208
?larnv, 6-210
?larrb, 6-212
?larre, 6-214
?larrf, 6-216
?larrv, 6-218
?lartg, 6-221
?lartv, 6-223
?laruv, 6-224
?larz, 6-225
?larzb, 6-227
?larzt, 6-230
?las2, 6-233
?lascl, 6-234
?lasd0, 6-236
?lasd1, 6-238
?lasd2, 6-241
?lasd3, 6-246
?lasd4, 6-249
?lasd5, 6-251
?lasd6, 6-252
?lasd7, 6-257
?nrm2, 2-12
?opgtr, 5-119
?opmtr, 5-120
?orgbr, 5-82
?orghr, 5-180
?orglq, 5-28, 5-38, 5-40
?orgqr, 5-17, 5-50, 5-52
?orgtr, 5-107
?ormbr, 5-85
?ormhr, 5-182
?ormlq, 5-30, 5-42, 5-45, 5-54, 5-57, 5-62, 5-65
?ormqr, 5-19
?ormtr, 5-109
?pbcon, 4-74
?pbrfs, 4-107
?pbstf, 5-172
?pbtrf, 4-18
?pbtrs, 4-45
?pocon, 4-70
?porfs, 4-101, 4-110
?potrf, 4-14
?potri, 4-135
?potrs, 4-41
?ppcon, 4-72
?pprfs, 4-104
?pptrf, 4-16
?pptri, 4-137
?pptrs, 4-43
?ptcon, 4-76
?pteqr, 5-146
?pttrf, 4-20
?pttrs, 4-47
?rot, 2-13
?rot (複素), 6-6
?rotg, 2-15
?roti, 2-125
?rotm, 2-16
?rotmg, 2-18
?sbev, 5-346
?sbevd, 5-350
?sbevx, 5-357
?sbgst, 5-166
?sbgv, 5-460
?sbgvd, 5-466
?sbgvx, 5-473
?sbmv, 2-53
?sbtrd, 5-128

?scal, 2-20
?sctr, 2-126
?sdot, 2-9
?spcon, 4-82
?spev, 5-327
?spevd, 5-331
?spevx, 5-338
?spgst, 5-162
?spgv, 5-439
?spgvd, 5-445
?spgvx, 5-452
?spmv, 2-56
?spr, 2-58, 6-9
?spr2, 2-60
?sprfs, 4-119
?sptrd, 5-117
?sptrf, 4-28
?sptri, 4-143
?sptrs, 4-53
?stebz, 5-141, 5-149
?stein, 5-152
?steqr, 5-134, 5-137
?sterf, 5-132
?stev, 5-365
?stevd, 5-367
?stevr, 5-374
?stevx, 5-370
?sum1, 6-20
?swap, 2-21
?sycon, 4-78, 5-154
?syev, 5-299
?syevd, 5-303
?syevr, 5-317
?syevx, 5-309
?sygst, 5-158
?sygv, 5-416
?sygvd, 5-422
?sygvx, 5-429
?symm, 2-98
?symv, 2-62
?symv (複素), 6-11
?syr, 2-64
?syr (複素), 6-13
?syr2, 2-66
?syr2k, 2-105
?syrfs, 4-113
?syrk, 2-102
?sytrd, 5-105
?sytrf, 4-22
?sytri, 4-139
?sytrs, 4-49
?tbcon, 4-90
?tbmv, 2-68
?tbsv, 2-71
?tbtrs, 4-61
?tgevc, 5-242
?tgexc, 5-247
?tgsen, 5-250
?tgsja, 5-271
?tgsna, 5-261
?tgsyl, 5-256
?tpcon, 4-88
?tpmv, 2-74
?tprfs, 4-127
?tpsv, 2-77
?tptri, 4-148
?tptrs, 4-59
?trcon, 4-86
?trevc, 5-205
?trexc, 5-215
?trmm, 2-109
?trmv, 2-79
?trrfs, 4-124
?trsen, 5-217
?trsm, 2-112
?trsna, 5-210
?trsv, 2-81
?trsyl, 5-222
?trtri, 4-147
?trtrs, 4-57
?ungbr, 5-88
?unghr, 5-185
?unglq, 5-32
?ungqr, 5-21
?ungtr, 5-113
?unmbr, 5-91
?unmhr, 5-187
?unmlq, 5-34
?unmqr, 5-23
?unmtr, 5-115
?upgtr, 5-124
?upmtr, 5-125

数字

1 次元 FFT, 3-1
格納の効果 , 3-8, 3-13, 9-42, 9-44
逆方向の FFT を計算する、複素数の入力データ , 3-13, 3-15
グループ , 3-2
実数から複素数への , 3-7
順方向の FFT を計算する、実数の入力データ , 3-8, 3-10
複素数から実数への , 3-12
複素数から複素数への , 3-3
複素数シーケンス , 3-10, 3-11, 3-14, 3-16
複素ベクトルの順方向または逆方向の FFT を計算する , 3-4, 3-5
1 次方程式の解の誤差
一般マトリックス , 4-92, 4-98
バンド格納 , 4-95
エルミート・マトリックス , 4-116
パックド格納 , 4-122
三角マトリックス , 4-124
パックド格納 , 4-127
バンド格納 , 4-130
正値エルミート・マトリックス , 4-101, 4-110
パックド格納 , 4-104
バンド格納 , 4-107
正値対称マトリックス , 4-101, 4-110
パックド格納 , 4-104
バンド格納 , 4-107
対称マトリックス , 4-113
パックド格納 , 4-119
1 次方程式の解の精度の改善
一般マトリックス , 4-98, 4-92
エルミート・マトリックス , 4-116
パックド格納 , 4-122
正値エルミート・マトリックス , 4-101, 4-110
パックド格納 , 4-104
バンド格納 , 4-107
正値対称マトリックス , 4-101, 4-110
パックド格納 , 4-104
バンド格納 , 4-107
対称マトリックス , 4-113
パックド格納 , 4-119
バンド・マトリックス , 4-95
1 次方程式を解く。1 次方程式を参照
2 次元 FFT, 3-17
逆方向の FFT を計算する、複素数の入力データ , 3-28, 3-29
グループ , 3-17
式 , 3-18
実数から複素数への , 3-21
順方向の FFT を計算する、実数の入力データ , 3-22, 3-24
順方向または逆方向の FFT を計算する , 3-19, 3-20
データ格納の型 , 3-18
データ構造の要件 , 3-18
複素数から実数への , 3-27
複素数から複素数への , 3-18

B

Bernoulli, 8-51
Binomial, 8-55
BLAS ルーチン
ベクトル引数 , A-1
マトリックス引数 , A-4
ルーチン・グループ , 1-6, 2-1
BLAS レベル 1 の関数
?asum, 2-4, 2-5
?dot, 2-4, 2-8
?dotc, 2-4, 2-10
?dotu, 2-4, 2-11
?nrm2, 2-4, 2-12
?sdot, 2-9
i?amax, 2-4, 2-22
i?amin, 2-4, 2-23
コード例 , B-1, B-2
BLAS レベル 1 のルーチン
?axpy, 2-4, 2-6
?copy, 2-4, 2-7
?rot, 2-4, 2-13
?rotg, 2-4, 2-15
?rotm, 2-16
?rotmg, 2-18
?scal, 2-4, 2-20
?swap, 2-4, 2-21
コード例 , B-2
BLAS レベル 2 のルーチン
?gbmv, 2-25, 2-26
?gemv, 2-25, 2-29
?ger, 2-25, 2-32
?gerc, 2-25, 2-33
?geru, 2-25, 2-35
?hbmv, 2-25, 2-37
?hemv, 2-25, 2-40
?her, 2-25, 2-42
?her2, 2-25, 2-44

?hpmv, 2-25, 2-46
?hpr, 2-25, 2-49
?hpr2, 2-25, 2-51
?sbmv, 2-25, 2-53
?spmv, 2-25, 2-56
?spr, 2-25, 2-58
?spr2, 2-25, 2-60
?symv, 2-25, 2-62
?syr, 2-25, 2-64
?syr2, 2-25, 2-66
?tbmv, 2-26, 2-68
?tbsv, 2-26, 2-71
?tpmv, 2-26, 2-74
?tpsv, 2-26, 2-77
?trmv, 2-26, 2-79
?trsv, 2-26, 2-81
コード例 , B-3, B-4
BLAS レベル 3 のルーチン
?gemm, 2-84, 2-85
?hemm, 2-84, 2-88
?her2k, 2-84, 2-94
?herk, 2-84, 2-91
?symm, 2-84, 2-98
?syr2k, 2-84, 2-105
?syrk, 2-84, 2-102
?trmm, 2-84, 2-109
?trsm, 2-84, 2-112
コード例 , B-4, B-5
Bunch-Kaufman 因子分解 , 4-7
エルミート・マトリックス , 4-25
パックド格納 , 4-31
対称マトリックス , 4-22
パックド格納 , 4-28

C

Cauchy, 8-36
CBLAS, C-1
スパース BLAS, C-18
引数 , C-1
レベル 1 (ベクトル演算), C-3
レベル 2 (マトリックス・ベクトル演算), C-6
レベル 3 (マトリックス・マトリックス演算), C-14
CommitDescriptor, 9-17
ComputeBackward, 9-23
ComputeForward, 9-21
CopyDescriptor, 9-18
CopyStream, 8-12
CopyStreamState, 8-13
CreateDescriptor, 9-15
C インターフェイス , 3-3

D

DeleteStream, 8-12
DFTI の構成パラメータ , 9-6
DFTI のステータス確認 , 9-10
DFTI のディスクリプタ構成 , 9-10
DFTI のディスクリプタ操作 , 9-9
DFT 計算 , 9-10
DFT ルーチン
DFT 計算
ComputeBackward, 9-23
ComputeForward, 9-21
ステータス確認
ErrorClass, 9-11
ErrorMessage, 9-13
ディスクリプタ構成
GetValue, 9-33
SetValue, 9-31
ディスクリプタ操作
CommitDescriptor, 9-17
CopyDescriptor, 9-18
CreateDescriptor, 9-15
FreeDescriptor, 9-20

E

ErrorClass, 9-11
ErrorMessage, 9-13
Exponential, 8-29

F

FFT に対するデータ構造の要件 , 3-3
FFT。高速フーリエ変換を参照
FreeDescriptor, 9-20

G

Gaussian, 8-26
Geometric, 8-53
GetBrngProperties, 8-67
GetNumRegBrng, 8-22
GetStreamStateBrng, 8-21
GetValue, 9-33
GFSR, 8-4
Givens 回転

スパース・ベクトルの , 2-125
パラメータ , 2-15
変形 Givens 変換のパラメータ , 2-18
Gumbel, 8-44

H

Hessenberg 上マトリックス , 5-174, 5-225
Hypergeometric, 8-57

I

i?amax, 2-22
i?amin, 2-23
i?max1, 6-15
ilaenv, 6-16

L

LAPACK ルーチン
LQ 因子分解
?gelqf, 5-25, 5-36
?orglq, 5-28, 5-38, 5-40
?ormlq, 5-30, 5-42, 5-45, 5-54, 5-57, 5-62, 5-65
?unglq, 5-32
?unmlq, 5-34
QR 因子分解
?geqpf, 5-11, 5-14
?geqrf, 5-8, 5-48, 5-60, 5-68, 5-71
?orgqr, 5-17, 5-50, 5-52
?ormqr, 5-19
?ungqr, 5-21
?unmqr, 5-23
解の精度の改善と誤差の推定
?gbrfs, 4-95
?gerfs, 4-92, 4-98
?herfs, 4-116
?hprfs, 4-122
?pbrfs, 4-107
?porfs, 4-101, 4-110
?pprfs, 4-104
?sprfs, 4-119
?syrfs, 4-113
?tbrfs, 4-130
?tprfs, 4-127
?trrfs, 4-124
三角因子分解
?gbtrf, 4-10
?getrf, 4-7
?gttrf, 4-12
?hetrf, 4-25
?hptrf, 4-31
?pbtrf, 4-18
?potrf, 4-14
?pptrf, 4-16
?pttrf, 4-20
?sptrf, 4-28
?sytrf, 4-22
条件数の推定
?gbcon, 4-65
?gecon, 4-63
?gtcon, 4-67
?hecon, 4-80
?hpcon, 4-84
?pbcon, 4-74
?pocon, 4-70
?ppcon, 4-72
?ptcon, 4-76
?spcon, 4-82
?sycon, 4-78, 5-154
?tbcon, 4-90
?tpcon, 4-88
?trcon, 4-86
シルベスター式
?trsyl, 5-222
対称固有値問題
?hbevd, 5-353
?hbtrd, 5-130
?heevd, 5-306
?hetrd, 5-111
?hpevd, 5-334
?hptrd, 5-122
?opgtr, 5-119
?opmtr, 5-120
?orgtr, 5-107
?ormtr, 5-109
?pteqr, 5-146
?sbevd, 5-350
?sbtrd, 5-128
?spevd, 5-331
?sptrd, 5-117
?stebz, 5-141, 5-149
?stein, 5-152
?steqr, 5-134, 5-137
?sterf, 5-132
?stevd, 5-367
?syevd, 5-303
?sytrd, 5-105

?ungtr, 5-113
?unmtr, 5-115
?upgtr, 5-124
?upmtr, 5-125
特異値分解
?bdsqr, 5-94, 5-98
?gbbrd, 5-79
?gebrd, 5-76
?orgbr, 5-82
?ormbr, 5-85
?ungbr, 5-88
?unmbr, 5-91
汎用固有値問題
?hbgst, 5-169
?hegst, 5-160
?hpgst, 5-164
?pbstf, 5-172
?sbgst, 5-166
?spgst, 5-162
?sygst, 5-158
非対称固有値問題
?gebak, 5-193
?gebal, 5-190
?gehrd, 5-178
?hsein, 5-199
?hseqr, 5-195
?orghr, 5-180
?ormhr, 5-182
?trevc, 5-205
?trexc, 5-215
?trsen, 5-217
?trsna, 5-210
?unghr, 5-185
?unmhr, 5-187
マトリックスの反転
?getri, 4-133
?hetri, 4-141
?hptri, 4-145
?potri, 4-135
?pptri, 4-137
?sptri, 4-143
?sytri, 4-139
?tptri, 4-148
?trtri, 4-147
連立方程式を解く
?gbtrs, 4-36
?getrs, 4-34
?gttrs, 4-38, 6-42
?hetrs, 4-51
?hptrs, 4-55
?pbtrs, 4-45
?potrs, 4-41
?pptrs, 4-43
?pttrs, 4-47
?sptrs, 4-53
?sytrs, 4-49
?tbtrs, 4-61
?tptrs, 4-59
?trtrs, 4-57
Laplace, 8-31
LeapfrogStream, 8-15
Lognormal, 8-41
LQ 因子分解 , 5-6
成分を求める
直交マトリックス Q, 5-28, 5-38, 5-40
ユニタリ・マトリックス Q の , 5-32
lsame, 6-18
lsamen, 6-19
LU 因子分解 , 4-7
三重対角マトリックス , 4-12
バンド・マトリックス , 4-10

N

NegBinomial, 8-61
NewStream, 8-9
NewStreamEx, 8-10

P

Poisson, 8-59

Q

QR 因子分解 , 5-6
成分を求める
直交マトリックス Q, 5-17, 5-50, 5-52
ユニタリ・マトリックス Q の , 5-21
ピボット (要) 演算あり , 5-11, 5-14

R

Rayleigh, 8-39
RegisterBrng, 8-66

S

SetValue, 9-31
SkipAheadStream, 8-18
SVD (特異値分解), 5-74

U

Uniform (分布), 8-47
Uniform (連続), 8-24
UniformBits, 8-49

V

VML, 7-1
VML Pack 関数 , 7-43
VML Unpack 関数 , 7-45
VML 関数
- pack/unpack 関数
  - Pack, 7-43
  - Unpack, 7-45
- VML サービス関数
  - ClearErrorCallBack, 7-58
  - ClearErrStatus, 7-54
  - GetErrorCallBack, 7-57
  - GetErrStatus, 7-54
  - GetMode, 7-51
  - SetErrorCallBack, 7-55
  - SetErrStatus, 7-52
  - SetMode, 7-48
- 数学関数
  - Acos, 7-28
  - Acosh, 7-36
  - Asin, 7-29
  - Asinh, 7-37
  - Atan, 7-30
  - Atan2, 7-31
  - Atanh, 7-38
  - Cbrt, 7-15
  - Cos, 7-23
  - Cosh, 7-32
  - Div, 7-11
  - Erf, 7-39
  - Erfc, 7-41
  - Exp, 7-20
  - Inv, 7-10
  - InvCbrt, 7-16
  - InvSqrt, 7-13
  - Ln, 7-21
  - Log10, 7-22
  - Pow, 7-17
  - Powx, 7-18
  - Sin, 7-24
  - SinCos, 7-25
  - Sinh, 7-34
  - Sqrt, 7-12
  - Tan, 7-27
  - Tanh, 7-35

VML のエラー診断 , 7-6
VML の関数の命名規則 , 7-2
VML の精密モード , 7-2
VSL 関数
- アドバンスト・サービス・サブルーチン
  - GetBrngProperties, 8-67
  - RegisterBrng, 8-66
- サービス・サブルーチン
  - CopyStream, 8-12
  - CopyStreamState, 8-13
  - DeleteStream, 8-12
  - GetStreamStateBrng, 8-21, 8-22
  - LeapfrogStream, 8-15
  - NewStream, 8-9
  - NewStreamEx, 8-10
  - SkipAheadStream, 8-18
- 生成器サブルーチン , 8-61
  - Bernoulli, 8-51
  - Binomial, 8-55
  - Cauchy, 8-36
  - Exponential, 8-29
  - Gaussian, 8-26
  - Geometric, 8-53
  - Gumbel, 8-44
  - Hypergeometric, 8-57
  - Laplace, 8-31
  - Lognormal, 8-41
  - Poisson, 8-59
  - Rayleigh, 8-39
  - Uniform (分布), 8-47
  - Uniform (連続), 8-24
  - UniformBits, 8-49
  - Weibull, 8-34

W

Weibull, 8-34

X

XERBLA、エラー報告ルーチン , 2-1

あ

圧縮形式のスパース・ベクトルの成分をフル格納形式に分散させる , 2-126
圧縮形式のスパース・ベクトルをフル格納形式に変換する , 2-126

い

一般マトリックス
LQ 因子分解 , 5-25, 5-36
LU 因子分解 , 4-7
バンド格納 , 4-10
QR 因子分解 , 5-8, 5-48, 5-60, 5-68, 5-71
ピボット (要) 演算あり , 5-11, 5-14
階数 1 の更新 , 2-32
階数 1 の更新、共役 , 2-33
階数 1 の更新、非共役 , 2-35
固有値問題 , 5-174, 5-225
条件数の推定 , 4-63
バンド格納 , 4-65
スカラー・マトリックス・マトリックスの積 , 2-85
マトリックスの反転 , 4-133
マトリックス・ベクトルの積 , 2-29
バンド格納 , 2-26
連立 1 次方程式を解く , 4-34
バンド格納 , 4-36
因子分解
Bunch-Kaufman, 4-7
LU, 4-7
コルスキー , 4-7
三角因子分解も参照
直交 (LQ、QR), 5-7

え

エラー報告ルーチン、XERBLA, 2-1
エルミート・マトリックス , 5-101, 5-157
Bunch-Kaufman 因子分解 , 4-25
パックド格納 , 4-31
階数 1 の更新 , 2-42
パックド格納 , 2-49
階数 2k の更新 , 2-94
階数 2 の更新 , 2-44
パックド格納 , 2-51
階数 n の更新 , 2-91
条件数の推定 , 4-80
パックド格納 , 4-84
スカラー・マトリックス・マトリックスの積 , 2-88
汎用固有値問題 , 5-157
マトリックスの反転 , 4-141
パックド格納 , 4-145
マトリックス・ベクトルの積 , 2-40
パックド格納 , 2-46
バンド格納 , 2-37
連立 1 次方程式を解く , 4-51
パックド格納 , 4-55

か

階数 1 の更新
一般マトリックス , 2-32
エルミート・マトリックス , 2-42
パックド格納 , 2-49
共役、一般マトリックス , 2-33
対称マトリックス , 2-64
パックド格納 , 2-58
非共役、一般マトリックス , 2-35
階数 2k の更新
エルミート・マトリックス , 2-94
対称マトリックス , 2-105
階数 2 の更新
エルミート・マトリックス , 2-44
パックド格納 , 2-51
対称マトリックス , 2-66
パックド格納 , 2-60
階数 n の更新
エルミート・マトリックス , 2-91
対称マトリックス , 2-102
回転
Givens 回転に対するパラメータ , 2-15
スパース・ベクトルの , 2-125
変形 Givens 変換のパラメータ , 2-18
変形面における点の , 2-16
面における点の , 2-13

き

擬似乱数 , 8-1
規則
LAPACK, 4-2
VML, 7-2

け

計算ルーチン , 5-6
検出する
最小絶対値を持つベクトル成分を , 2-23
最大絶対値を持つベクトルの成分を , 2-22

こ

合計
スパース・ベクトルとフル格納形式のベクトルの , 2-118
ベクトルの , 2-6
ベクトルの成分の大きさの , 2-5
更新する
階数 1
一般マトリックス , 2-32
エルミート・マトリックス , 2-42
パックド格納 , 2-49
対称マトリックス , 2-64
パックド格納 , 2-58
階数 1、共役
一般マトリックス , 2-33
階数 1、非共役
一般マトリックス , 2-35
階数 2
エルミート・マトリックス , 2-44
パックド格納 , 2-51
対称マトリックス , 2-66
パックド格納 , 2-60
階数 2k
エルミート・マトリックス , 2-94
対称マトリックス , 2-105
階数 n
エルミート・マトリックス , 2-91
対称マトリックス , 2-102
高速フーリエ変換 , 1-3
C インターフェイス , 3-3
データ格納の型 , 3-2
データ構造の要件 , 3-3
ルーチン
?fft1d, 3-4, 3-8, 3-13
?fft1dc, 3-5, 3-10, 3-15
?fft2d, 3-19, 3-22, 3-28
?fft2dc, 3-20, 3-24, 3-29
コード例
BLAS レベル 1 の関数 , B-1
BLAS レベル 1 のルーチン , B-2
BLAS レベル 2 のルーチン , B-3
BLAS レベル 3 のルーチン , B-4
固有値問題
一般マトリックス , 5-174, 5-225
エルミート・マトリックス , 5-101
対称マトリックス , 5-101
汎用形式 , 5-157
固有値。固有値問題を参照
固有ベクトル。固有値問題を参照
コルスキー因子分解
正値エルミート・マトリックス , 4-14
パックド格納 , 4-16
バンド格納 , 4-18
正値対称マトリックス , 4-14
パックド格納 , 4-16
バンド格納 , 4-18

さ

三角因子分解
一般マトリックス , 4-7
エルミート・マトリックス , 4-25
パックド格納 , 4-31
三重対角マトリックス , 4-12
正値エルミート・マトリックス , 4-14
三重対角マトリックス , 4-20
パックド格納 , 4-16
バンド格納 , 4-18
正値対称マトリックス , 4-14
三重対角マトリックス , 4-20
パックド格納 , 4-16
バンド格納 , 4-18
対称マトリックス , 4-22
パックド格納 , 4-28
バンド・マトリックス , 4-10
三角マトリックス , 5-174, 5-225
条件数の推定 , 4-86
パックド格納 , 4-88
バンド格納 , 4-90
スカラー・マトリックス・マトリックスの積 , 2-109
マトリックスの反転 , 4-147
パックド格納 , 4-148
マトリックス・ベクトルの積 , 2-79
パックド格納 , 2-74

バンド格納 , 2-68
連立 1 次方程式を解く , 2-81, 4-57
パックド格納 , 2-77, 4-59
バンド格納 , 2-71, 4-61
三重対角マトリックス , 5-101
条件数の推定 , 4-67
連立 1 次方程式を解く , 4-38, 4-47, 6-42

し

次元 , A-1
実数から複素数への 1 次元 FFT, 3-7
実数から複素数への 2 次元 FFT, 3-21
準三角マトリックス , 5-174, 5-225
順方向または逆方向の FFT, 3-4, 3-5, 3-19, 3-20
条件数
一般マトリックス , 4-63
エルミート・マトリックス , 4-80
パックド格納 , 4-84
三角マトリックス , 4-86
パックド格納 , 4-88
バンド格納 , 4-90
三重対角マトリックス , 4-67
正値エルミート・マトリックス , 4-70
三重対角 , 4-76
パックド格納 , 4-72
バンド格納 , 4-74
正値対称マトリックス , 4-70
三重対角 , 4-76
パックド格納 , 4-72
バンド格納 , 4-74
対称マトリックス , 4-78, 5-154
パックド格納 , 4-82
バンド・マトリックス , 4-65
シルベスター式 , 5-222

す

スカラー・マトリックスの積 , 2-85, 2-88, 2-98
スカラー・マトリックス・マトリックスの積 , 2-88
一般マトリックス , 2-85
三角マトリックス , 2-109
対称マトリックス , 2-98
ストライド。増分を参照
ストリーム・ステート , 8-7
ストリーム・ディスクリプタ , 8-2
スパース BLAS, 2-116
データ・タイプ , 2-117
命名規則 , 2-117
スパース BLAS のルーチンと関数 , 2-117
?axpyi, 2-118
?dotci, 2-121
?doti, 2-120
?dotui, 2-122
?gthr, 2-123
?gthrz, 2-124
?roti, 2-125
?sctr, 2-126
スパース・ベクトル , 2-116
BLAS レベル 1 のルーチンに渡される , 2-118
Givens 回転 , 2-125
圧縮形式 , 2-116
圧縮形式に変換する , 2-123, 2-124
加算とスケーリング , 2-118
実数型ドット積 , 2-120
スケーリング , 2-118
ノルム , 2-118
複素共役のドット積 , 2-121
複素非共役のドット積 , 2-122
フル格納形式 , 2-116
フル格納形式に変換する , 2-126
スパース・ベクトルの成分を圧縮形式に集積する , 2-123
また、これらの成分にゼロを書き込む , 2-124
スパース・ベクトルを圧縮格納形式に変換する , 2-123
また、元のベクトルにゼロを書き込む , 2-124

せ

正値エルミート・マトリックス
コルスキー因子分解 , 4-14
パックド格納 , 4-16
バンド格納 , 4-18
条件数の推定 , 4-70
パックド格納 , 4-72
バンド格納 , 4-74
マトリックスの反転 , 4-135
パックド格納 , 4-137
連立 1 次方程式を解く , 4-41
パックド格納 , 4-43
バンド格納 , 4-45

正値対称マトリックス
コルスキー因子分解 , 4-14
パックド格納 , 4-16
バンド格納 , 4-18
条件数の推定 , 4-70
三重対角マトリックス , 4-76
パックド格納 , 4-72
バンド格納 , 4-74
マトリックスの反転 , 4-135
パックド格納 , 4-137
連立 1 次方程式を解く , 4-41
パックド格納 , 4-43
バンド格納 , 4-45
正値マトリックス
汎用固有値問題 , 5-158
積
スカラー・マトリックス
一般マトリックス , 2-85
エルミート・マトリックス , 2-88
スカラー・マトリックス・マトリックス
一般マトリックス , 2-85
エルミート・マトリックス , 2-88
三角マトリックス , 2-109
対称マトリックス , 2-98
ドット積も参照
ベクトル・スカラーの , 2-20
マトリックス・ベクトルの
一般マトリックス , 2-29
バンド格納 , 2-26
エルミート・マトリックス , 2-40
パックド格納 , 2-46
バンド格納 , 2-37
三角マトリックス , 2-79
パックド格納 , 2-74
バンド格納 , 2-68
対称マトリックス , 2-62
パックド格納 , 2-56
バンド格納 , 2-53
線形合同生成器 , 8-3

そ

増分 , A-1

た

対応プラットフォーム , 1-5
対称マトリックス , 5-101, 5-157
Bunch-Kaufman 因子分解 , 4-22
パックド格納 , 4-28
階数 1 の更新 , 2-64
パックド格納 , 2-58
階数 2k の更新 , 2-105
階数 2 の更新 , 2-66
パックド格納 , 2-60
階数 n の更新 , 2-102
条件数の推定 , 4-78, 5-154
パックド格納 , 4-82
スカラー・マトリックス・マトリックスの積 , 2-98
汎用固有値問題 , 5-157
マトリックスの反転 , 4-139
パックド格納 , 4-143
マトリックス・ベクトルの積 , 2-62
パックド格納 , 2-56
バンド格納 , 2-53
連立 1 次方程式を解く , 4-49
パックド格納 , 4-53

ち

直交マトリックス , 5-74, 5-101, 5-174, 5-225

て

データ・タイプ
VML の , 7-2
省略表記 , 1-8
点
変形面における回転 , 2-16
面における回転 , 2-13
転置パラメータ , A-6

と

登録済み基本生成器 , 8-63
特異値分解 , 5-74
LAPACK ルーチン、特異値分解も参照のこと
ドット積
実数型スパース・ベクトル , 2-120
実ベクトル , 2-8
実ベクトル (extended precision), 2-9

複素共役スパース・ベクトル , 2-121
複素共役ベクトル , 2-10
複素スパース・ベクトル , 2-122
複素非共役ベクトル , 2-11
ドライバ・ルーチン , 5-278

な

長さ。次元を参照

に

二重対角マトリックス , 5-74

は

パック形式 , 9-38
パックド格納形式 , A-4
パラメータ
Givens 回転に対する , 2-15
変形 Givens 変換 , 2-18
バンド格納形式 , A-4
汎用固有値問題 , 5-157
LAPACK ルーチン、汎用固有値問題も参照のこと
実対称問題 , 5-158
パックド格納 , 5-162
バンド格納 , 5-166
複素エルミート問題 , 5-160
パックド格納 , 5-164
バンド格納 , 5-169
汎用固有値問題の縮退 , 5-158

ひ

引数
スパース・ベクトル , 2-116
ベクトルの , A-1
マトリックス , A-4

ふ

フォント表記規則 , 1-8
複素数から実数への 1 次元 FFT, 3-12
複素数から実数への 2 次元 FFT, 3-27
複素数から複素数への 1 次元 FFT, 3-3
複素数から複素数への 2 次元 FFT, 3-18
フル格納形式 , A-4
ブロック分割法 , 8-6
分割コルスキー因子分解 (バンド・マトリックス), 5-172

へ

ベクトル
Givens 回転 , 2-15
交換 , 2-21
コピーする , 2-7
最小絶対値を持つ成分 , 2-23
最大絶対値を持つ成分 , 2-22
スパース・ベクトル , 2-117
点の回転 , 2-13
ドット積
実ベクトル , 2-8
複素共役ベクトル , 2-10
複素非共役ベクトル , 2-11
ベクトル・スカラーの積 , 2-20
ベクトルの合計 , 2-6
ベクトルの成分の大きさを加算する , 2-5
ベクトルの線形結合 , 2-6
変形 Givens 変換のパラメータ , 2-18
変形面における点の回転 , 2-16
ユークリッド・ノルム , 2-12
ベクトル・インデックス , 7-6
ベクトル数学関数 , 7-8
10 進対数 , 7-22
complementary error function value, 7-41
逆正弦 , 7-29
逆正接 , 7-30
逆双曲正弦 , 7-37
逆双曲正接 , 7-38
逆双曲余弦 , 7-36
逆転 , 7-10
逆平方根 , 7-13
逆余弦 , 7-28
逆立方根 , 7-16
誤差関数 , 7-39
指数 , 7-20
自然対数 , 7-21
四分円逆正接 , 7-31
乗 , 7-17
乗 (定数), 7-18
正弦 , 7-24
正弦および余弦 , 7-25
正接 , 7-27
双曲正弦 , 7-34

双曲正接 , 7-35
双曲余弦 , 7-32
平方根 , 7-12
余弦 , 7-23
立方根 , 7-15
割る , 7-11
ベクトル・スカラーの積 , 2-20
スパース・ベクトル , 2-118
ベクトル成分の最小絶対値 , 2-23
ベクトル成分の絶対値
最小の , 2-23
最大の , 2-22
ベクトルの交換 , 2-21
ベクトルの成分の大きさを加算する , 2-5
ベクトルの線形結合 , 2-6
ベクトルの統計関数
Bernoulli, 8-51
Binomial, 8-55
Cauchy, 8-36
CopyStream, 8-12
CopyStreamState, 8-13
DeleteStream, 8-12
Exponential, 8-29
Gaussian, 8-26
Geometric, 8-53
GetBrngProperties, 8-67
GetNumRegBrng, 8-22
GetStreamStateBrng, 8-21
Gumbel, 8-44
Hypergeometric, 8-57
Laplace, 8-31
LeapfrogStream, 8-15
Lognormal, 8-41
NegBinomial, 8-61
NewStream, 8-9
NewStreamEx, 8-10
Poisson, 8-59
Rayleigh, 8-39
RegisterBrng, 8-66
SkipAheadStream, 8-18
Uniform (分布), 8-47
Uniform (連続), 8-24
UniformBits, 8-49
Weibull, 8-34
ベクトル引数 , A-1
スパース・ベクトル , 2-116
増分 , A-1
デフォルトの , A-2
長さ , A-1
配列の次元 , A-1
マトリックスの 1 次元下部構造 , A-2
例 , A-2
ベクトルをコピーする , 2-7

ま

マトリックスの 1 次元下部構造 , A-2
マトリックスの逆行列を計算する。マトリックスの反転を参照
マトリックスの反転
一般マトリックス , 4-133
エルミート・マトリックス , 4-141
パックド格納 , 4-145
三角マトリックス , 4-147
パックド格納 , 4-148
正値エルミート・マトリックス , 4-135
パックド格納 , 4-137
正値対称マトリックス , 4-135
パックド格納 , 4-137
対称マトリックス , 4-139
パックド格納 , 4-143
マトリックスの平衡化 , 5-190
マトリックスの方程式
AX = B, 2-112, 4-5, 4-33
マトリックス引数 , A-4
行数 , A-6
転置パラメータ , A-6
リーディング・ディメンジョン , A-6
例 , A-7
列主体の順序 , A-2, A-6
列数 , A-6
マトリックス・ベクトル演算
階数 1 の更新 , 2-32, 2-33, 2-35
エルミート・マトリックス , 2-42
パックド格納 , 2-49
対称マトリックス , 2-64
パックド格納 , 2-58
階数 2 の更新
エルミート・マトリックス , 2-44
パックド格納 , 2-51
対称マトリックス , 2-66
パックド格納 , 2-60
積 , 2-26, 2-29
エルミート・マトリックス , 2-40

パックド格納 , 2-46
バンド格納 , 2-37
三角マトリックス , 2-79
パックド格納 , 2-74
バンド格納 , 2-68
対称マトリックス , 2-62
パックド格納 , 2-56
バンド格納 , 2-53
マトリックス・マトリックス演算
階数 2k の更新
エルミート・マトリックス , 2-94
対称マトリックス , 2-105
階数 n の更新
エルミート・マトリックス , 2-91
対称マトリックス , 2-102
スカラー・マトリックス・マトリックスの積
エルミート・マトリックス , 2-88
三角マトリックス , 2-109
対称マトリックス , 2-98
積
一般マトリックス , 2-85

め

命名規則 , 1-8
BLAS, 2-2
LAPACK, 4-2, 5-4
VML, 7-2
スパース BLAS, 2-117

ゆ

ユークリッド・ノルム
ベクトルの , 2-12
ユニタリ・マトリックス , 5-74, 5-101, 5-174, 5-225

ら

乱数生成器 , 8-1
ランダム・ストリーム , 8-2

り

リーディング・ディメンジョン , A-6
リープフロッグ法 , 8-6
離散フーリエ変換
CommitDescriptor, 9-17
ComputeBackward, 9-23
ComputeForward, 9-21
CopyDescriptor, 9-18
CreateDescriptor, 9-15
ErrorClass, 9-11
ErrorMessage, 9-13
FreeDescriptor, 9-20
GetValue, 9-33
SetValue, 9-31
離散分布生成器 , 8-23

る

ルーチンの命名規則
BLAS, 2-2
スパース BLAS, 2-117

れ

連続分布生成器 , 8-23
連立 1 次方程式
三角マトリックスでの , 2-81
パックド格納 , 2-77
バンド格納 , 2-71
連立 1 次方程式。1 次方程式を参照
連立方程式、解く
一般マトリックス , 4-34
バンド格納 , 4-36
エルミート・マトリックス , 4-51
パックド格納 , 4-55
三角マトリックス , 4-57
パックド格納 , 4-59
バンド格納 , 4-61
三重対角マトリックス , 4-38, 4-47, 6-42
正値エルミート・マトリックス , 4-41
パックド格納 , 4-43
バンド格納 , 4-45
正値対称マトリックス , 4-41
パックド格納 , 4-43
バンド格納 , 4-45
対称マトリックス , 4-49
パックド格納 , 4-53