インテル® マス・カーネル・ライブラリー

クックブック

インテル® MKL

資料番号: 330244-002JA

目次

著作権と商標について.....3
ヘルプとサポートについて.....6
表記規則.....7
関連情報.....8
インテル® マス・カーネル・ライブラリーのレシピ.....9

第 1 章: 定常非線形熱伝導方程式の近似解を求める

第 2 章: 一般的なブロック三重対角行列の因数分解

第 3 章: LU 因数分解されたブロック三重対角係数行列を含む連立線形方程式を解く

第 4 章: フーリエ積分の評価

第 5 章: 高速フーリエ変換を使用したコンピューター・トモグラフィー・イメージの復元

第 6 章: 金融市場のデータストリームにおけるノイズ・フィルタリング
文献目録 (英語).....38

著作権と商標について

本資料に掲載されている情報は、インテル製品の概要説明を目的としたものです。 本資料は、明示されているか否かにかかわらず、また禁反言によるとよらずにかかわらず、いかなる知的財産権のライセンスも許諾するものではありません。 製品に付属の売買契約書『Intel's Terms and Conditions of Sale』に規定されている場合を除き、インテルはいかなる責任を負うものではなく、またインテル製品の販売や使用に関する明示または黙示の保証 (特定目的への適合性、商品適格性、あらゆる特許権、著作権、その他知的財産権の非侵害性への保証を含む) に関してもいかなる責任も負いません。

「ミッション・クリティカルなアプリケーション」とは、インテル製品がその欠陥や故障によって、直接的または間接的に人身傷害や死亡事故が発生するようなアプリケーションを指します。 そのようなミッション・クリティカルなアプリケーションのためにインテル製品を購入または使用する場合は、直接的か間接的かにかかわらず、あるいはインテル製品やそのいかなる部分の設計、製造、警告にインテルまたは委託業者の過失があったかどうかにかかわらず、製造物責任、人身傷害や死亡の請求を起因とするすべての賠償請求費用、損害、費用、合理的な弁護士費用をすべて補償し、インテルおよびその子会社、委託業者および関連会社、およびそれらの役員、経営幹部、従業員に何らの損害も与えないことに同意するものとします。

インテル製品は、予告なく仕様や説明が変更されることがあります。 機能または命令の一覧で「留保」または「未定義」と記されているものがありますが、その「機能が存在しない」あるいは「性質が留保付である」という状態を設計の前提にしないでください。 これらの項目は、インテルが将来のために留保しているものです。インテルが将来これらの項目を定義したことにより、衝突が生じたり互換性が失われたりしても、インテルは一切責任を負いません。 この情報は予告なく変更されることがあります。 この情報だけに基づいて設計を最終的なものとしないでください。

本資料で説明されている製品には、エラッタと呼ばれる設計上の不具合が含まれている可能性があり、公表されている仕様とは異なる動作をする場合があります。 現在確認済みのエラッタについては、インテルまでお問い合わせください。 最新の仕様をご希望の場合や製品をご注文の場合は、お近くのインテルの営業所または販売代理店にお問い合わせください。 本資料で紹介されている資料番号付きのドキュメントや、インテルのその他の資料を入手するには、1-800-548-4725 (アメリカ合衆国) までご連絡いただくか、インテルの Web サイトを参照してください。

性能に関するテストに使用されるソフトウェアとワークロードは、性能がインテル® マイクロプロセッサー用に最適化されていることがあります。 SYSmark* や MobileMark* などの性能テストは、特定のコンピューター・システム、コンポーネント、ソフトウェア、操作、機能に基づいて行ったものです。 結果はこれらの要因によって異なります。 製品の購入を検討される場合は、他の製品と組み合わせた場合の本製品の性能など、ほかの情報や性能テストも参考にして、パフォーマンスを総合的に評価することをお勧めします。

本資料には、開発の設計段階にある製品についての情報が含まれています。

Intel、インテル、Intel ロゴは、アメリカ合衆国および / またはその他の国における Intel Corporation の商標です。

* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

Microsoft、Windows、Windows ロゴは、アメリカ合衆国および / またはその他の国における Microsoft Corporation の商標または登録商標です。

Java は、Oracle および / または関連会社の登録商標です。

サードパーティー・コンテンツ

インテル® マス・カーネル・ライブラリー (インテル® MKL) には、いくつかのサードパーティー提供のコンテンツが含まれており、それぞれ以下のライセンス規約が適用されます (敬称略)。

- © Copyright 2001 Hewlett-Packard Development Company, L.P.
- Linear Algebra PACKage (LAPACK) ルーチンのセクションには、著作権で保護された派生物の一部が含まれています。

  © 1991, 1992, and 1998 by The Numerical Algorithms Group, Ltd.

- インテル® MKL は、以下のライセンスの下で LAPACK 3.4.1 の計算、ドライバー、および補助ルーチンをサポートしています。

  Copyright © 1992-2011 The University of Tennessee and The University of Tennessee Research Foundation. All rights reserved.

  Copyright © 2000-2011 The University of California Berkeley. All rights reserved.

  Copyright © 2006-2012 The University of Colorado Denver. All rights reserved.

  Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

  - Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
  - Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer listed in this license in the documentation and/or other materials provided with the distribution.
  - Neither the name of the copyright holders nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

  The copyright holders provide no reassurances that the source code provided does not infringe any patent, copyright, or any other intellectual property rights of third parties. The copyright holders disclaim any liability to any recipient for claims brought against recipient by any third party for infringement of that parties intellectual property rights.

  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

  インテル® MKL の一部の基となった LAPACK の原版は http://www.netlib.org/lapack/index.html (英語) から入手できます。 LAPACK の開発は、E. Anderson、Z. Bai、C. Bischof、S. Blackford、J. Demmel、J. Dongarra、J. Du Croz、A. Greenbaum、S. Hammarling、A. McKenney、D. Sorensen らによって行われました。
- インテル® MKL の一部の基となった BLAS の原版は http://www.netlib.org/blas/index.html (英語) から入手できます。
- インテル® MKL の一部の基となった BLACS の原版は http://www.netlib.org/blacs/index.html (英語) から入手できます。 BLACS の開発は、Jack Dongarra と R. Clint Whaley によって行われました。
- インテル® MKL クラスター・エディションの一部の基となった ScaLAPACK の原版は http://www.netlib.org/scalapack/index.html (英語) から入手できます。 ScaLAPACK の開発は、L. S. Blackford、J. Choi、A. Cleary、E. D'Azevedo、J. Demmel、I. Dhillon、J. Dongarra、S. Hammarling、G. Henry、A. Petitet、K. Stanley、D. Walker、R. C. Whaley らによって行われました。
- インテル® MKL の一部の基となった PBLAS の原版は http://www.netlib.org/scalapack/html/pblas_qref.html (英語) から入手できます。
- インテル® MKL の PARDISO* (PARallel DIrect SOlver) の開発は、バーゼル大学のコンピューター・サイエンス学部 (http://www.unibas.ch (英語)) によって行われました。 http://www.pardiso-project.org (英語) から入手できます。
- 拡張固有値ソルバーは、Feast ソルバーパッケージを基にしており、以下のライセンスの下で配布されています。

  Copyright © 2009, The Regents of the University of Massachusetts, Amherst.
  Developed by E. Polizzi
  All rights reserved.

  Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

- 本リリースのインテル® MKL の一部の高速フーリエ変換 (FFT) 関数は、カーネギーメロン大学からライセンスを受けて、SPIRAL ソフトウェア生成システム (http://www.spiral.net/ (英語)) によって生成されました。 SPIRAL の開発は、Markus Püschel、José Moura、Jeremy Johnson、David Padua、Manuela Veloso、Bryan Singer、Jianxin Xiong、Franz Franchetti、Aca Gacic、Yevgen Voronenko、Kang Chen、Robert W. Johnson、Nick Rizzolo らによって行われました。

ヘルプとサポートについて

インテル® MKL 製品の Web サイトでは、製品機能、ホワイトペーパー、技術資料など、製品に関する最新かつ全般的な情報を提供しています。 最新情報は、http://www.intel.com/software/products/support (英語) を参照してください。

インテルでは、使い方のヒント、既知の問題点、製品のエラッタ、ライセンス情報、ユーザーフォーラムなどの多くのセルフヘルプ情報を含むサポート Web サイトを提供しています。詳しくは、http://www.intel.com/software/products/ (英語) を参照してください。

登録を行うことで、サポートサービス期間中 (通常は 1 年間)、製品アップデートとテクニカルサポートが提供されます。インテル® プレミアサポート Web サイトでは、次のサービスを利用できます。

- 問題の送信とステータスの確認
- 製品アップデートのダウンロード

製品の登録、製品サポートの利用、インテルへの問い合わせは、次のサイトにアクセスしてください: http://www.intel.com/software/products/support (英語)。

表記規則

このマニュアルでは、以下のように対象となるオペレーティング・システムを指しています。

Windows*	サポートしているすべての Windows* オペレーティング・システムで有効な情報を指します。
Linux*	サポートしているすべての Linux* オペレーティング・システムで有効な情報を指します。
OS X*	OS X* オペレーティング・システムを実行しているインテル® プロセッサー・ベースのシステムで有効な情報を指します。

このマニュアルでは、次の表記規則を使用しています。

- ルーチン名の省略形 (例えば、`cungqr/zungqr` の代わりに `?ungqr`)
- テキストとコードを区別するためのフォントの表記規則

ルーチン名の省略形

省略形 では、疑問符 "`?`" を含む名前は同様の機能を備えたグループのルーチンを表します。 各グループは通常、使用されるルーチンと 4 つの基本的なデータ型 (単精度実数、倍精度実数、単精度複素数、倍精度複素数) から成ります。 疑問符は関数の任意またはすべての種類を示します。次に例を示します。

`?swap`	ベクトル-ベクトル `?swap` ルーチンの 4 つのデータ型すべて (`sswap`、`dswap`、`cswap`、`zswap`) を指します。

フォントの表記規則

以下の フォントの表記規則が使用されています。

大文字の `COURIER`	Fortran インターフェイスの入出力パラメーターの記述で使用されるデータ型。例: `CHARACTER*1`。

小文字の `courier`	コードサンプル:
	`a(k+i,j) = matrix(i,j)`
	`a[k+i][j] = matrix[i][j];`
	および C インターフェイスのデータ型。例: `const float*`。
	データ型。例: `const float*`。

小文字と大文字の `courier`	C インターフェイスの関数名。例: `vmlSetMode`。
小文字斜体の `courier`	引数およびパラメーター記述の変数。 例: *incx*
*	コードサンプルや方程式の乗算記号として、またプログラミング言語の構文で必要な個所に使用されます。

関連情報

アプリケーションでライブラリーを使用する方法については、このドキュメントのほか、次のドキュメントも併せて参照してください。

- インテル® マス・カーネル・ライブラリー (インテル® MKL) リファレンス・マニュアル - ルーチンの機能、パラメーターの説明、インターフェイス、呼び出し構文と戻り値についてのリファレンス情報を提供します。
- インテル® マス・カーネル・ライブラリー (インテル® MKL) ユーザーズガイド。

これらのドキュメントの Web バージョンは、インテル® ソフトウェア・ドキュメント・ライブラリー (英語) から入手できます。

インテル® マス・カーネル・ライブラリーのレシピ

インテル® マス・カーネル・ライブラリー (インテル® MKL) には、行列を乗算する、連立方程式を解く、フーリエ変換を行うなど、さまざまな数値問題を解く際に役立つ多くのルーチンが含まれています。 専用のインテル® MKL ルーチンが用意されていない問題については、インテル® MKL で提供されているビルディング・ブロックを組み合わせることにより問題を解くことができます。

インテル® MKL クックブックには、より複雑な問題を解くためにインテル® MKL ルーチンを組み合わせる際に役立つ手法が含まれています。

- 「定常非線形熱伝導方程式の近似解を求める」では、インテル® MKL PARDISO、BLAS、スパース BLAS ルーチンを使用して非線形方程式の解を求める手法を紹介します。
- 「ブロック三重対角行列の因数分解」では、BLAS と LAPACK ルーチンのインテル® MKL 実装を使用します。
- 「LU 因数分解されたブロック三重対角係数行列を含む連立線形方程式を解く」では、因数分解レシピを拡張して連立方程式を解きます。
- 「フーリエ積分の評価」では、インテル® MKL 高速フーリエ変換 (FFT) インターフェイスを使用して連続するフーリエ変換積分を評価します。
- 「高速フーリエ変換を使用したコンピューター・トモグラフィー・イメージの復元」では、インテル® MKL FFT インターフェイスを使用してコンピューター・トモグラフィー・データからイメージを復元します。
- 「金融市場のデータストリームにおけるノイズ・フィルタリング」では、インテル® MKL サマリー統計ルーチンを使用してストリーミング・データの相関行列を計算します。

注
クックブックのコードサンプルは、Fortran または C で提供されています。

最適化に関する注意事項

= = = = = = = = = = =

インテル® コンパイラーは、互換マイクロプロセッサー向けには、インテル製マイクロプロセッサー向けと同等レベルの最適化が行われない可能性があります。 これには、インテル® ストリーミング SIMD 拡張命令 2 (インテル® SSE2)、インテル® ストリーミング SIMD 拡張命令 3 (インテル® SSE3)、ストリーミング SIMD 拡張命令 3 補足命令 (SSSE3) 命令セットに関連する最適化およびその他の最適化が含まれます。 インテルでは、インテル製ではないマイクロプロセッサーに対して、最適化の提供、機能、効果を保証していません。 本製品のマイクロプロセッサー固有の最適化は、インテル製マイクロプロセッサーでの使用を目的としています。 インテル® マイクロアーキテクチャーに非固有の特定の最適化は、インテル製マイクロプロセッサー向けに予約されています。 この注意事項の適用対象である特定の命令セットの詳細は、該当する製品のユーザー・リファレンス・ガイドを参照してください。

改訂 #20110804

= = = = = = = = = = =

定常非線形熱伝導方程式の近似解を求める 1

目的

熱方程式の境界値問題の解と、その解に依存する熱係数を得る。

ソリューション

固定小数点の反復アプローチ [Amos10] を使用し、インテル® MKL PARDISO によって各外部反復の線形問題を解きます。

1. CSR (Compressed Sparse Rows: 圧縮スパース行) 形式で行列構造を設定します。
2. 残差のノルムが許容差未満になるまで固定小数点の反復を行います。
   a. `pardiso` ルーチンを使用して現在の反復の線形化されたシステムを解きます。
   b. `dcopy` ルーチンを使用して、システムの解を主方程式の次の近似に設定します。
   c. 新しい近似に基づいて、行列の新しい要素を計算します。
   d. `mkl_cspblas_dcsrgemv` ルーチンを使用して現在の解の残差を計算します。
   e. `dnrm2` ルーチンを使用して残差のノルムを計算し、許容差と比較します。
3. ソルバーの内部メモリーを解放します。

ソースコード: サンプル (http://software.intel.com/en-us/mkl_cookbook_samples) の `sparse` フォルダーを参照してください。

インテル® MKL PARDISO、スパース BLAS、BLAS を使用して近似解を求める

```
CONSTRUCT_MATRIX_STRUCTURE (nx, ny, nz, &ia, &ja, &a, &error);
CONSTRUCT_MATRIX_VALUES (nx, ny, nz, us, ia, ja, a, &error);
DO WHILE res > tolerance
  phase = 13;
  PARDISO (pt, &maxfct, &mnum, &mtype, &phase, &n, a, ia, ja, &idum,
    &nrhs, iparm, &msglvl, f, u, &error );
  DCOPY (&n, u, &one, u_next, &one);
  CONSTRUCT_MATRIX_VALUES (nx, ny, nz, u_next, ia, ja, a, &error);
  MKL_CSPBLAS_DCSRGEMV (uplo, &n, a, ia, ja, u, temp );
  DAXPY (&n, &minus_one, f, &one, temp, &one);
  res = DNRM2 (&n, temp, &one);
END DO
phase = -1;
PARDISO ( pt, &maxfct, &mnum, &mtype, &phase, &n, a, ia, ja, &idum,
  &nrhs, iparm, &msglvl, f, u, &error );
```

使用するルーチン

タスク	ルーチン	説明
現在の反復の線形化されたシステムを解きます。ソルバーの内部メモリーを解放します。	PARDISO	複数の右辺を備えた 1 セットのスパース線形方程式の解を計算します。
見つかった解を主方程式の次の近似として設定します。	DCOPY	ベクトルを別のベクトルにコピーします。
現在の非線形反復の残差を計算します。	MKL_CSPBLAS_DCSRGEMV	ゼロベースでインデックスされ、CSR 形式で格納されたスパース一般行列の行列-ベクトル積を計算します。
	DAXPY	ベクトル-スカラー積を計算して結果をベクトルに追加します。
残差のノルムを計算し、停止条件と比較します。	DNRM2	ベクトルのユークリッド・ノルムを計算します。

説明

定常非線形熱伝導方程式は、非線形偏微分方程式の境界値問題として説明できます。

$$-\frac{\partial}{\partial x}\left(\mu(v)\frac{\partial v}{\partial x}\right)-\frac{\partial}{\partial y}\left(\mu(v)\frac{\partial v}{\partial y}\right)-\frac{\partial}{\partial z}\left(\mu(v)\frac{\partial v}{\partial z}\right)=1,\ (x,y,z)\in D$$

$$v\big|_{\partial D}=0$$

ここで、領域 *D* は立方体であると仮定します。*D* = $(0, 1)^3$ および *v*(*x*, *y*, *z*) は温度の未知関数です。

デモ目的のため、問題は解の熱係数の線形従属性に制限されています。

$$\mu(v)=1+10v$$

数の解を得るため、領域 *D* でグリッドステップ *h* の等距離のグリッドが選択され、偏微分方程式は差分を使用して近似されます。 このプロシージャーから [Smith86] 非線形代数方程式のシステムが得られます。

$$-u_{i,j,k-1}*\frac{m_{i,j,k-1}+m_{i,j,k}}{2}+2u_{i,j,k}*m_{i,j,k}-u_{i,j,k+1}*\frac{m_{i,j,k+1}+m_{i,j,k}}{2}$$
$$-u_{i,j-1,k}*\frac{m_{i,j-1,k}+m_{i,j,k}}{2}+2u_{i,j,k}*m_{i,j,k}-u_{i,j+1,k}*\frac{m_{i,j+1,k}+m_{i,j,k}}{2}$$
$$-u_{i+1,j,k}*\frac{m_{i+1,j,k}+m_{i,j,k}}{2}+2u_{i,j,k}*m_{i,j,k}-u_{i+1,j,k}*\frac{m_{i+1,j,k}+m_{i,j,k}}{2}$$
$$=h^2$$

$$u_{0,j,k}=u_{n,j,k}=u_{i,0,k}=u_{i,n,k}=u_{i,j,0}=u_{i,j,n}=0$$

$$m_{i,j,k}=1+10*u_{i,j,k}$$

$$i=\overline{1,n},\ j=\overline{1,n},\ k=\overline{1,n},\ n=\frac{1}{h}$$

各方程式は、7 つのグリッドポイントで、未知グリッド関数 *u* の値とそれぞれの右辺の値を関連付けます。 方程式の左辺は、解に依存するグリッド関数値と係数の線形の組み合わせとして表せます。 これらの係数から構成される行列を利用すると、方程式をベクトル-行列形式で書き直すことができます。

$A(\tilde{u})\tilde{u} = g$

係数行列 *A* は疎である (各行に非ゼロ要素が 7 つしかない) ため、この反復アルゴリズムで解くために、行列を CSR 形式の配列に格納して (『インテル® マス・カーネル・ライブラリー (インテル® MKL) リファレンス・マニュアル』の「スパース行列格納形式」を参照)、PARDISO* ソルバーを使用することは適切です。

1. *u* を初期値 u^0 に設定します。
2. 残差 *r* = *A*(*u*)*u* - *g* を計算します。
3. ||*r*|| < 許容差の場合:
   a. 方程式 *A*(*u*)*w* = *g* を *w* について解きます。
   b. *u* = *w* を設定します。
   c. 残差 *r* = *A*(*u*)*u* - *g* を計算します。

一般的なブロック三重対角行列の因数分解

2

目的

一般的なブロック三重対角行列の LU 因数分解を行う。

ソリューション

インテル® MKL LAPACK は、密行列、帯行列、三重対角行列を含む、一般的な行列の LU 因数分解用のさまざまなサブルーチンを提供しています。 この手法は、すべてのブロックが正方形で同位という条件を前提として、一般的なブロック三重対角行列の機能の範囲を拡張します。

サイズ *NB* x *NB* の正方形ブロックでブロック三重対角行列の LU 因数分解を行うには:

1. 部分 LU 因数分解を、行列の最初の 2 つのブロック行と最初の 3 つのブロック列 (*M* = 2*NB*、*N* = 3*NB*、*K* = *NB*) で構成されたサイズ *M* x *N* の長方形ブロックにシーケンシャルに適用し、最後の 1 つのブロック列が処理されるまで対角線に沿って下に移動します。

   部分 LU 因数分解: 一般的なブロック三重対角行列の LU 因数分解では、長方形の *M* x *N* 行列の部分 LU 因数分解用に別の機能が用意されていると便利です。 パラメーター *K* (ここで、$K \leq \min(M, N)$) の部分 LU 因数分解アルゴリズムは、次のステップからなります。

   a. *M* x *K* の部分行列の LU 因数分解を実行します。

   b. 三角係数行列を含む方程式を解きます。

   c. 右下の (*M* - *K*) x (*N* - *K*) ブロックを更新します。

   生成される行列は *A* = *P*(*LU* + *A1*) です。ここで、*L* は下台形 *M* x *K* 行列、*U* は上台形行列、*P* は置換 (ピボット) 行列、*A1* は最後の *M* - *K* 行と *N* - *K* 列の交差領域のみの非ゼロ要素行列です。

2. 一般的な LU 因数分解を最後の (2*NB*) x (2*NB*) ブロックに適用します。

ソースコード: サンプル (http://software.intel.com/en-us/mkl_cookbook_samples) の `BlockTDS_GE/source/dgeblttrf.f` ファイルを参照してください。

部分 LU 因数分解の実行

```
      SUBROUTINE PTLDGETRF(M, N, K, A, LDA, IPIV, INFO)
      …
          CALL DGETRF( M, K, A, LDA, IPIV, INFO )
          …
          DO I=1,K
              IF(IPIV(I).NE.I)THEN
                  CALL DSWAP(N-K, A(I,K+1), LDA, A(IPIV(I), K+1), LDA)
              END IF
          END DO
          CALL DTRSM('L','L','N','U',K,N-K,1D0, A, LDA, A(1,K+1), LDA)
          CALL DGEMM('N', 'N', M-K, N-K, K, -1D0, A(K+1,1), LDA,
     &            A(1,K+1), LDA, 1D0, A(K+1,K+1), LDA)
          …
```

ブロック三重対角行列の因数分解

```
      …
      DO K=1,N-2
C ブロック構造の 2*NB x 3*NB の部分行列 A を構成
C     (D_K   C_K 0    )
C     (B_K D_K+1 C_K+1)
      …
C 部分行列を部分的に因数分解
          CALL PTLDGETRF(2*NB, 3*NB, NB, A, 2*NB, IPIV(1,K), INFO)
C 因数分解の結果を三重対角行列のブロックを格納する配列に戻す
      …
      END DO
C ループを抜けて最後の 2*NB x 2*NB の部分行列を因数分解
      CALL DGETRF(2*NB, 2*NB, A, 2*NB, IPIV(1,N-1), INFO)
C 最後の結果を三重対角行列のブロックを格納する配列に戻す
      …
```

使用するルーチン

タスク	ルーチン	説明
M x *K* の部分行列の LU 因数分解	DGETRF	一般的な m x n 行列の LU 因数分解を計算します。
行列の行の置換	DSWAP	2 つのベクトルをスワップします。
三角係数行列を含む方程式を解く	DTRSM	三角行列を含む方程式を解きます。
右下の (*M* - *K*) x (*N* - *K*) ブロックを更新する	DGEMM	一般的な行列を含む行列-行列の積を計算します。

説明

部分 LU 因数分解では、A を長方形の *m* x *n* 行列にします。

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,k-1} & a_{1,k} & a_{1,k+1} & \cdots & a_{1,n-1} & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,k-1} & a_{2,k} & a_{2,k+1} & \cdots & a_{2,n-1} & a_{2,n} \\ a_{3,1} & a_{3,2} & a_{3,3} & \cdots & a_{3,k-1} & a_{3,k} & a_{3,k+1} & \cdots & a_{3,n-1} & a_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \iddots & \vdots & \vdots \\ a_{k-1,1} & a_{k-1,2} & a_{k-1,3} & \cdots & a_{k-1,k-1} & a_{k-1,k} & a_{k-1,k+1} & \cdots & a_{k-1,n-1} & a_{k-1,n} \\ a_{k,1} & a_{k,2} & a_{k,3} & \cdots & a_{k,k-1} & a_{k,k} & a_{k,k+1} & \cdots & a_{k,n-1} & a_{k,n} \\ a_{k+1,1} & a_{k+1,2} & a_{k+1,3} & \cdots & a_{k+1,k-1} & a_{k+1,k} & a_{k+1,k+1} & \iddots & a_{k+1,n-1} & a_{k+1,n} \\ \vdots & \vdots & \vdots & \iddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{m-1,1} & a_{m-1,2} & a_{m-1,3} & \cdots & a_{m-1,k-1} & a_{m-1,k} & a_{m-1,k-1} & \cdots & a_{m-1,n-1} & a_{m-1,n} \\ a_{m,1} & a_{m,2} & a_{m,3} & \cdots & a_{m,k-1} & a_{m,k} & a_{m,k+1} & \cdots & a_{m,n-1} & a_{m,n} \end{bmatrix}$$

注
読みやすいように、ここでは *m*、*n*、*k*、*nb* のように小文字のインデックスを使用しています。 これらのインデックスは、Fortran ソリューションおよびコードサンプルで使用されている大文字のインデックスに相当します。

行列は、*m* x *k* (ここで、0 < *k* ≤ *n*) の部分行列の LU 因数分解を使用して分解できます。 このアプリケーションでは、*m* ≤ *k* ≤ *n* または *n* = *k* ≤ *m* の場合は行列の因数分解に `?getrf` を直接使用できるため、*k* < min(*m*, *n*) です。

A はブロック行列として表現できます。

$$A=\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

ここで、A_{11} は *k* x *k* の部分行列、A_{12} は *k* x (*n* - *k*) の部分行列、A_{21} は (*m* - *k*) x *k* の部分行列、A_{22} は (*m* - *k*) x (*n* - *k*) の部分行列です。

m x *k* のパネル A_1 は次のように定義できます。

$$A_1=\begin{bmatrix} A_{11} \\ A_{21} \end{bmatrix}$$

A_1 は (`?getrf` を使用して) $A_1 = PLU$ として LU 因数分解できます。ここで、*P* は置換 (ピボット) 行列、*L* は対角要素を含む下台形行列、*U* は上三角行列です。

$$L=\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 0 \\ l_{2,1} & 1 & 0 & \cdots & 0 & 0 \\ l_{3,1} & l_{3,2} & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ l_{k-1,1} & l_{k-1,2} & l_{i-1,3} & \cdots & 1 & 0 \\ l_{k,1} & l_{k,2} & l_{k,3} & \cdots & l_{k,k-1} & 1 \\ l_{k+1,1} & l_{k+1,2} & l_{k+1,3} & \cdots & l_{k+1,k-1} & l_{k+1,k} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ l_{m-1,1} & l_{m-1,2} & l_{m-1,3} & \cdots & l_{m-1,k-1} & l_{m-1,k} \\ l_{m,1} & l_{m,2} & l_{m,3} & \cdots & l_{m,k-1} & l_{m,k} \end{bmatrix} = \begin{bmatrix} L_{11} \\ L_{21} \end{bmatrix}$$

$$U=\begin{bmatrix} u_{1,1} & u_{1,2} & u_{1,3} & \cdots & u_{1,k-1} & u_{1,k} \\ 0 & u_{2,2} & u_{2,3} & \cdots & u_{2,k-1} & u_{2,k} \\ 0 & 0 & u_{3,3} & \cdots & u_{3,k-1} & u_{3,k} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & u_{k-1,k-1} & u_{k-1,k} \\ 0 & 0 & 0 & \cdots & 0 & u_{k,k} \end{bmatrix}$$

注

L の対角要素は格納する必要がないため、A_1 を格納するために使用する配列を、*L* と *U* の要素を格納するために使用できます。

P^T を *A* の 2 つ目のパネルに適用すると次のようになります。

$$P^T A_2 = P^T \begin{bmatrix} A_{12} \\ A_{22} \end{bmatrix} = \begin{bmatrix} A'_{12} \\ A'_{22} \end{bmatrix}$$

次の方程式が得られます。

$$P^T A = \begin{bmatrix} L_{11}U & A'_{12} \\ L_{21}U & A'_{22} \end{bmatrix}$$

A''_{12} を次のように定義します。

$$A''_{12} = L_{11}^{-1} A'_{12}$$

$P^T A$ の方程式を変更します。

$$P^{T}A=\begin{bmatrix}L_{11}U & L_{11}A''_{12}\\ L_{21}U & A'_{22}\end{bmatrix}$$
$$=\begin{bmatrix}L_{11}\\ L_{21}\end{bmatrix}\begin{bmatrix}U & A''_{12}\end{bmatrix}+\begin{bmatrix}0 & 0\\ 0 & A'_{22}-L_{21}A''_{12}\end{bmatrix}$$

前の方程式と *P* を乗算すると次のようになります。

$$A=P\left(\begin{bmatrix}L_{11}\\ L_{21}\end{bmatrix}\begin{bmatrix}U & L_{11}^{-1}A'_{12}\end{bmatrix}+\begin{bmatrix}0 & 0\\ 0 & A'_{22}-L_{21}L_{11}^{-1}A'_{12}\end{bmatrix}\right)$$

この方程式は最初の行列の部分 LU 因数分解と考えることができます。

- 積 $L^{-1}{}_{11}A'_{12}$ は `?trsm` を呼び出して計算し、A_{12} に使用される配列の代わりに格納できます。 更新 A'_{22} - $L_{21}(L^{-1}{}_{11}A'_{12})$ は `?gemm` を呼び出して計算し、A_{22} に使用される配列の代わりに格納できます。
- 部分行列にすべてのランクが含まれない場合、LU 因数分解に失敗するため、このメソッドは適用できません。
- 一般的な行列の LU 因数分解とは異なり、下記に示す一般的なブロック三重対角行列の因数分解 *A = LU* は、*A = PLU* (*P* は置換行列) 形式で記述できません。 ピボットのため、左の係数 *L* は置換を含みます。 また、右の係数 *U* も複雑になります (2 つの対角線ではなく 3 つの対角線を含む)。

ロック三重対角行列の LU 因数分解では、*A* を、すべてのブロックが正方形で同位のブロック三重対角行列 n_b にします。

$$A=\begin{bmatrix}D_1 & C_1 & 0 & \cdots & 0 & 0 & 0\\ B_1 & D_2 & C_2 & \cdots & 0 & 0 & 0\\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots\\ 0 & 0 & 0 & \cdots & B_{N-2} & D_{N-1} & C_{N-1}\\ 0 & 0 & 0 & \cdots & 0 & B_{N-1} & D_N\end{bmatrix}$$

行列は *A = LU* として因数分解されます。

最初に、2 x 3 ブロックの部分行列を考えます。

$$\begin{bmatrix}D_1 & C_1 & 0\\ B_1 & D_2 & C_2\end{bmatrix}$$

この部分行列は次のように分解できます。

$$P_1^{T}\begin{bmatrix}D_1 & C_1 & 0\\ B_1 & D_2 & C_2\end{bmatrix}=\begin{bmatrix}L_{11}\\ L_{21}\end{bmatrix}\begin{bmatrix}U_{11} & U_{12} & U_{13}\end{bmatrix}+\begin{bmatrix}0 & 0 & 0\\ 0 & D'_2 & C''_2\end{bmatrix}$$

この分解は前述の部分 LU 因数分解を適用して取得できます。 ここで、$P^{T}{}_1$ は n_b 要素の順列の積であり、$2n_b$ x $2n_b$ 行列として表現できます。

$$P_1^{T}=\begin{bmatrix}P_{11}^1 & P_{12}^1\\ P_{21}^1 & P_{22}^1\end{bmatrix}$$

すべてのブロックがサイズ n_b x n_b の *N* x *N* ブロック行列を適用します。

$$\begin{bmatrix}P_{11}^1 & P_{12}^1 & 0 & \cdots & 0\\ P_{21}^1 & P_{22}^1 & 0 & \cdots & 0\\ 0 & 0 & I & \cdots & 0\\ \vdots & \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & 0 & \cdots & I\end{bmatrix}$$

分解は次のように表現できます。

$$\begin{bmatrix} P_{11}^1 & P_{12}^1 & 0 & \cdots & 0 \\ P_{21}^1 & P_{22}^1 & 0 & \cdots & 0 \\ 0 & 0 & I & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & I \end{bmatrix} \begin{bmatrix} D_1 & C_1 & 0 & \cdots & 0 & 0 & 0 \\ B_1 & D_2 & C_2 & \cdots & 0 & 0 & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & B_{N-2} & D_{N-1} & C_{N-1} \\ 0 & 0 & 0 & \cdots & 0 & B_{N-1} & D_N \end{bmatrix}$$

$$= \begin{bmatrix} L_{11} \\ L_{21} \\ 0 \\ \vdots \\ 0 \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} & U_{13} & 0 & \cdots & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & D_2' & C_2' & \cdots & 0 & 0 & 0 \\ 0 & B_2 & D_3 & C_3 & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & B_{N-2} & D_{N-1} & C_{N-1} \\ 0 & 0 & 0 & \cdots & 0 & B_{N-1} & D_N \end{bmatrix}$$

次に、方程式の右辺の行列の、2 番目と 3 番目の行の 2 x 3 ブロック行列を因数分解します。

$$P_2^{\mathrm{T}} \begin{bmatrix} D_2' & C_2' & 0 \\ B_2 & D_3 & C_3 \end{bmatrix} = \begin{bmatrix} L_{22} \\ L_{32} \end{bmatrix} \begin{bmatrix} U_{22} & U_{23} & U_{24} \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & D_3' & C_3' \end{bmatrix}$$

ここで、$P^{\mathrm{T}}{}_2$ は次のように定義されます。

$$P_2^{\mathrm{T}} = \begin{bmatrix} P_{22}^2 & P_{23}^2 \\ P_{32}^2 & P_{33}^2 \end{bmatrix}$$

分解は次のようになります。

$$\begin{bmatrix} I & 0 & 0 & 0 & \cdots & 0 \\ 0 & P_{11}^1 & P_{12}^1 & 0 & \cdots & 0 \\ 0 & P_{21}^1 & P_{22}^1 & 0 & \cdots & 0 \\ 0 & 0 & 0 & I & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & I \end{bmatrix} \begin{bmatrix} P_{11}^1 & P_{12}^1 & 0 & 0 & \cdots & 0 \\ P_{21}^1 & P_{22}^1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & I & 0 & \cdots & 0 \\ 0 & 0 & 0 & I & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & I \end{bmatrix} \begin{bmatrix} D_1 & C_1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ B_1 & D_2 & C_2 & 0 & \cdots & 0 & 0 & 0 \\ 0 & B_2 & D_3 & C_3 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & B_{N-2} & D_{N-1} & C_{N-1} \\ 0 & 0 & 0 & 0 & \cdots & 0 & B_{N-1} & D_N \end{bmatrix}$$

$$= \begin{bmatrix} I & 0 & 0 & 0 & \cdots & 0 \\ 0 & P_{11}^1 & P_{12}^1 & 0 & \cdots & 0 \\ 0 & P_{21}^1 & P_{22}^1 & 0 & \cdots & 0 \\ 0 & 0 & 0 & I & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & I \end{bmatrix} \begin{bmatrix} L_{11} \\ L_{21} \\ 0 \\ \vdots \\ 0 \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} & U_{13} & 0 & \cdots & 0 \end{bmatrix}$$

$$+ \begin{bmatrix} I & 0 & 0 & 0 & \cdots & 0 \\ 0 & P_{11}^1 & P_{12}^1 & 0 & \cdots & 0 \\ 0 & P_{21}^1 & P_{22}^1 & 0 & \cdots & 0 \\ 0 & 0 & 0 & I & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & I \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & D_2' & C_2' & 0 & \cdots & 0 & 0 & 0 \\ 0 & B_2 & D_3 & C_3 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & 0 & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & B_{N-2} & D_{N-1} & C_{N-1} \\ 0 & 0 & 0 & 0 & \cdots & 0 & B_{N-1} & D_N \end{bmatrix}$$

$$= \begin{bmatrix} I & 0 & 0 & 0 & \cdots & 0 \\ 0 & P_{11}^1 & P_{12}^1 & 0 & \cdots & 0 \\ 0 & P_{21}^1 & P_{22}^1 & 0 & \cdots & 0 \\ 0 & 0 & 0 & I & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & I \end{bmatrix} \begin{bmatrix} L_{11} \\ L_{21} \\ 0 \\ \vdots \\ 0 \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} & U_{13} & 0 & \cdots & 0 \end{bmatrix} + \begin{bmatrix} 0 \\ L_{22} \\ L_{32} \\ 0 \\ \vdots \\ 0 \end{bmatrix} \begin{bmatrix} 0 & U_{22} & U_{23} & U_{24} & 0 & \cdots & 0 \end{bmatrix}$$

$$+ \begin{bmatrix} 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & D_3' & C_3' & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & B_{N-2} & D_{N-1} & C_{N-1} \\ 0 & 0 & 0 & 0 & \cdots & 0 & B_{N-1} & D_N \end{bmatrix}$$

この表記をピボット行列に適用して方程式を単純化します。

$$\Pi_j^{\mathrm{T}} = \begin{bmatrix} I & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \iddots & \vdots \\ 0 & \cdots & I & 0 & 0 & 0 & \cdots & 0 \\ 0 & \cdots & 0 & P_{j,j}^{j} & P_{j,j+1}^{j} & 0 & \cdots & 0 \\ 0 & \cdots & 0 & P_{j+1,j}^{j} & P_{j+1,j+1}^{j} & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 0 & 0 & I & \cdots & 0 \\ \vdots & \iddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & I \end{bmatrix}, \quad j = 1, 2, \ldots, N-1$$

$$P_j^{\mathrm{T}} = \begin{bmatrix} P_{j,j}^{j} & P_{j,j+1}^{j} \\ P_{j+1,j}^{j} & P_{j+1,j+1}^{j} \end{bmatrix}$$

ここで、P^{T}_j は $2n_b$ x $2n_b$ で j 番目と (j+1) 番目の行と列の交差領域にあります。 分解は次のように簡潔に表記されます。

$$\Pi_2^{\mathrm{T}}\Pi_1^{\mathrm{T}} \begin{bmatrix} D_1 & C_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ B_1 & D_2 & C_2 & 0 & 0 & 0 & 0 & 0 \\ 0 & B_2 & D_3 & C_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & \ddots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & 0 & 0 & 0 & B_{N-2} & D_{N-1} & C_{N-1} \\ 0 & 0 & 0 & 0 & 0 & 0 & B_{N-1} & D_N \end{bmatrix}$$

$$= \Pi_2^{\mathrm{T}} \begin{bmatrix} L_{11} \\ L_{21} \\ 0 \\ \vdots \\ 0 \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} & U_{13} & 0 & \cdots & 0 \end{bmatrix} + \begin{bmatrix} 0 \\ L_{22} \\ L_{32} \\ 0 \\ \vdots \\ 0 \end{bmatrix} \begin{bmatrix} 0 & U_{22} & U_{23} & U_{24} & 0 & \cdots & 0 \end{bmatrix}$$

$$+ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & D_3' & C_3' & 0 & 0 & 0 \\ 0 & 0 & 0 & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & 0 & 0 & B_{N-2} & D_{N-1} & C_{N-1} \\ 0 & 0 & 0 & 0 & 0 & B_{N-1} & D_N \end{bmatrix}$$

ステップ N - 2 のローカル因数分解は次のようになります。

$$P_{N-2}^{\mathrm{T}} \begin{bmatrix} D_{N-2}' & C_{N-2}' & 0 \\ B_{N-2} & D_{N-1} & C_{N-1} \end{bmatrix} = \begin{bmatrix} L_{N-2,N-2} \\ L_{N-1,N-2} \end{bmatrix} \begin{bmatrix} U_{N-2,N-2} & U_{N-2,N-1} & U_{N-2,N} \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & D_{N-1}' & C_{N-1}' \end{bmatrix}$$

このステップの後、ピボット行列で乗算します。

$$\Pi_j^{\mathrm{T}} \quad j = 3, 4, \ldots, N-2$$

分解は次のようになります。

$$\Pi_{N-2}^{\mathrm{T}}\ldots\Pi_2^{\mathrm{T}}\Pi_1^{\mathrm{T}}\begin{bmatrix} D_1 & C_1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ B_1 & D_2 & C_2 & 0 & \cdots & 0 & 0 & 0 \\ 0 & B_2 & D_3 & C_3 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & B_{N-2} & D_{N-1} & C_{N-1} \\ 0 & 0 & 0 & 0 & \cdots & 0 & B_{N-1} & D_N \end{bmatrix}$$

$$= \Pi_{N-2}^{\mathrm{T}}\ldots\Pi_3^{\mathrm{T}}\Pi_2^{\mathrm{T}}\begin{bmatrix} L_{1,1} \\ L_{2,1} \\ 0 \\ \vdots \\ 0 \end{bmatrix}\begin{bmatrix} U_{11} & U_{12} & U_{13} & 0 & \cdots & 0 \end{bmatrix} + \Pi_{N-2}^{\mathrm{T}}\ldots\Pi_3^{\mathrm{T}}\begin{bmatrix} 0 \\ L_{2,2} \\ L_{3,2} \\ 0 \\ \vdots \\ 0 \end{bmatrix}\begin{bmatrix} 0 & U_{22} & U_{23} & U_{24} & 0 & \cdots & 0 \end{bmatrix}$$

$$+\ldots+\Pi_{N-2}^{\mathrm{T}}\begin{bmatrix} 0 \\ \vdots \\ L_{N-3,N-3} \\ L_{N-2,N-3} \\ 0 \\ 0 \end{bmatrix}\begin{bmatrix} 0 & \cdots & 0 & U_{N-3,N-3} & U_{N-3,N-2} & U_{N-3,N-1} & 0 \end{bmatrix}$$

$$+\begin{bmatrix} 0 \\ \vdots \\ 0 \\ L_{N-2,N-2} \\ L_{N-1,N-2} \\ 0 \end{bmatrix}\begin{bmatrix} 0 & \cdots & 0 & U_{N-2,N-2} & U_{N-2,N-1} & U_{N-2,N} \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & D'_{N-1} & C'_{N-1} \\ 0 & 0 & 0 & 0 & \cdots & B_{N-1} & D_N \end{bmatrix}$$

最後の (*N* - 1) 番目のステップでは、行列は正方形で因数分解は完了です。

$$P_{N-1}^{\mathrm{T}}\begin{bmatrix} D'_{N-1} & C'_{N-1} \\ B_{N-1} & D_N \end{bmatrix} = \begin{bmatrix} L_{N-1,N-1} & 0 \\ L_{N,N-1} & L_{N,N} \end{bmatrix}\begin{bmatrix} U_{N-1,N-1} & U_{N-1,N} \\ 0 & U_{N,N} \end{bmatrix}$$

最後のステップはそれまでのステップとピボットの構造が異なります。前の P^{T}_j (j = 1, 2, ..., N - 2) はすべて n_b 置換の積 (n_b 整数パラメーターに依存) でしたが、P^{T}_{N-1} は次数 $2n_b$ の正方行列 ($2n_b$ パラメーターに依存) に適用されます。 そのため、ピボットインデックスをすべて格納するには、長さ (N - 2) n_b + $2n_b$ = Nn_b の整数配列が必要です。

左から前の分解と Π^{T}_{N-1} を乗算すると、最終的な分解が得られます。

$$\Pi_{N-1}^{T}\Pi_{N-2}^{T}\ldots\Pi_{2}^{T}\Pi_{1}^{T}\begin{bmatrix} D_1 & C_1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ B_1 & D_2 & C_2 & 0 & \cdots & 0 & 0 & 0 \\ 0 & B_2 & D_3 & C_3 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & B_{N-2} & D_{N-1} & C_{N-1} \\ 0 & 0 & 0 & 0 & \cdots & 0 & B_{N-1} & D_N \end{bmatrix}$$

$$= \Pi_{N-1}^{T}\Pi_{N-2}^{T}\ldots\Pi_{3}^{T}\Pi_{2}^{T}\begin{bmatrix} L_{1,1} \\ L_{2,1} \\ 0 \\ \vdots \\ 0 \end{bmatrix}\begin{bmatrix} U_{11} & U_{12} & U_{13} & 0 & \cdots & 0 \end{bmatrix}$$

$$+\Pi_{N-1}^{T}\Pi_{N-2}^{T}\ldots\Pi_{3}^{T}\begin{bmatrix} 0 \\ L_{2,2} \\ L_{3,2} \\ 0 \\ \vdots \\ 0 \end{bmatrix}\begin{bmatrix} 0 & U_{22} & U_{23} & U_{24} & 0 & \cdots & 0 \end{bmatrix}$$

$$+\ldots+\Pi_{N-1}^{T}\Pi_{N-2}^{T}\begin{bmatrix} 0 \\ \vdots \\ L_{N-3,N-3} \\ L_{N-2,N-3} \\ 0 \\ 0 \end{bmatrix}\begin{bmatrix} 0 & \cdots & 0 & U_{N-3,N-3} & U_{N-3,N-2} & U_{N-3,N-1} & 0 \end{bmatrix}$$

$$+\Pi_{N-1}^{T}\begin{bmatrix} 0 \\ \vdots \\ 0 \\ L_{N-2,N-2} \\ L_{N-1,N-2} \\ 0 \end{bmatrix}\begin{bmatrix} 0 & \cdots & 0 & U_{N-2,N-2} & U_{N-2,N-1} & U_{N-2,N} \end{bmatrix}$$

$$+\begin{bmatrix} 0 \\ \vdots \\ 0 \\ 0 \\ L_{N-1,N-1} \\ L_{N,N-1} \end{bmatrix}\begin{bmatrix} 0 & \cdots & 0 & 0 & U_{N-1,N-1} & U_{N-1,N} \end{bmatrix}+\begin{bmatrix} 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ L_{N,N} \end{bmatrix}\begin{bmatrix} 0 & \cdots & 0 & 0 & 0 & U_{N,N} \end{bmatrix}$$

この分解と $\Pi_1\Pi_2\ldots\Pi_{N-1}$ を乗算すると、LU 因数分解形式で記述できます。

$$\begin{bmatrix} D_1 & C_1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ B_1 & D_2 & C_2 & 0 & \cdots & 0 & 0 & 0 \\ 0 & B_2 & D_3 & C_3 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & B_{N-2} & D_{N-1} & C_{N-1} \\ 0 & 0 & 0 & 0 & \cdots & 0 & B_{N-1} & D_N \end{bmatrix} = LU$$

$$= \left(\Pi_1\begin{bmatrix} L_{11} \\ L_{21} \\ 0 \\ \vdots \\ 0 \end{bmatrix}\Pi_1\Pi_2\begin{bmatrix} 0 \\ L_{22} \\ L_{32} \\ 0 \\ \vdots \\ 0 \end{bmatrix}\cdots\Pi_1\Pi_2\cdots\Pi_{N-2}\begin{bmatrix} 0 \\ \vdots \\ 0 \\ L_{N-2,N-2} \\ L_{N-1,N-2} \\ 0 \end{bmatrix}\Pi_1\Pi_2\cdots\Pi_{N-2}\Pi_{N-1}\begin{bmatrix} 0 \\ \vdots \\ 0 \\ L_{N-1,N-1} \\ L_{N,N-1} \end{bmatrix}\Pi_1\Pi_2\cdots\Pi_{N-2}\Pi_{N-1}\begin{bmatrix} 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ L_{N,N} \end{bmatrix}\right)$$

$$\cdot\begin{bmatrix} U_{1,1} & U_{1,2} & U_{1,3} & 0 & \cdots & 0 & 0 & 0 \\ 0 & U_{2,2} & U_{2,3} & U_{2,4} & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & U_{N-2,N-2} & U_{N-2,N-1} & U_{N-2,N} \\ 0 & 0 & 0 & 0 & \cdots & 0 & U_{N-1,N-1} & U_{N-1,N} \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & U_{N,N} \end{bmatrix}$$

この式を適用するときは、Π_j (j = 1, 2, …, N-2) はインデックス j と j+1 のブロック行に適用される n_b 転置の積ですが、$\Pi_{N\text{-}1}$ は最後の 2 つのブロック行 N-1 と N に適用される $2n_b$ 転置の積であることに注意してください。

LU 因数分解されたブロック三重対角係数行列を含む連立線形方程式を解く

3

目的

インテル® MKL LAPACK のルーチンを使用してブロック三重対角係数行列を含む連立方程式の解を求める (LAPACK にはブロック三重対角係数行列を含む式を直接解くルーチンがないため)。

ソリューション

インテル® MKL LAPACK は、LU 因数分解された係数行列を含む連立方程式を解くためのさまざまなサブルーチンを提供しています (密行列、帯行列、三重対角行列など)。 この手法は、すべてのブロックが正方形で同位という条件を前提として、このセットをブロック三重対角行列に拡張します。 ブロック三重対角行列 *A* の形式は次のとおりです。

$$A=\begin{bmatrix} D_1 & C_1 & 0 & \cdots & 0 & 0 & 0 \\ B_1 & D_2 & C_2 & \cdots & 0 & 0 & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & B_{N-2} & D_{N-1} & C_{N-1} \\ 0 & 0 & 0 & \cdots & 0 & B_{N-1} & D_N \end{bmatrix}$$

LU 因数分解された行列 *A=LU* と複数の右辺 (RHS) を含む式 *AX=F* は、2 段階で解きます (LU 因数分解の詳細は、「一般的なブロック三重対角行列の因数分解」を参照)。

1. 前方置換。ピボットを含む連立方程式 *LY=F* (*L* は下三角係数行列三角) を解きます。 因数分解されたブロック三重対角行列では、最後のブロックを除く *Y* のブロックはすべて、次の方法によりループ内で見つかります。
   a. 右辺にピボット置換を適用します。
   b. 下三角係数行列の *NB* 線形方程式を解きます (*NB* はブロックの次数)。
   c. 次のステップのために右辺を更新します。

   最後のピボットの構造 (2 つのブロック置換を連続して適用する必要がある) と係数行列の構造により、最後の 2 つのブロック・コンポーネントはループの外で見つかります。
2. 後方置換。式 *UX=Y* を解きます。 ピボットを含まないため、このステップはより単純です。 プロシージャーは、最初のステップに似ています。
   a. 三角係数行列を含む方程式を解きます。
   b. 右辺のブロックを更新します。

   前のステップとの違いは、係数行列が下三角ではなく上三角で、ループの向きが逆になっていることです。

ソースコード: サンプル (http://software.intel.com/en-us/mkl_cookbook_samples) の `BlockTDS_GE/source/dgeblttrs.f` ファイルを参照してください。

前方置換

```
! 前方置換
! ループ内で配列 F に格納されているコンポーネント Y_K を計算
      DO K = 1, N-2
          DO I = 1, NB
              IF (IPIV(I,K) .NE. I) THEN
                  CALL DSWAP(NRHS, F((K-1)*NB+I,1), LDF, F((K-1)*NB+IPIV(I,K),1), LDF)
              END IF
          END DO
          CALL DTRSM('L', 'L', 'N', 'U', NB, NRHS, 1D0, D(1,(K-1)*NB+1), NB, F((K-1)*NB+1,1), LDF)
          CALL DGEMM('N', 'N', NB, NRHS, NB, -1D0, DL(1,(K-1)*NB+1), NB, F((K-1)*NB+1,1), LDF, 1D0,
     +         F(K*NB+1,1), LDF)
      END DO

! 最後の 2 つのピボットを適用
      DO I = 1, NB
           IF (IPIV(I,N-1) .NE. I) THEN
               CALL DSWAP(NRHS, F((N-2)*NB+I,1), LDF, F((N-2)*NB+IPIV(I,N-1),1), LDF)
           END IF
      END DO

      DO I = 1, NB
           IF(IPIV(I,N)-NB.NE.I)THEN
               CALL DSWAP(NRHS, F((N-1)*NB+I,1), LDF, F((N-2)*NB+IPIV(I,N),1), LDF)
           END IF
      END DO
! ループの外で Y_N-1 と Y_N を計算して配列 F に格納
      CALL DTRSM('L', 'L', 'N', 'U', NB, NRHS, 1D0, D(1,(N-2)*NB+1), NB, F((N-2)*NB+1,1), LDF)
      CALL DGEMM('N', 'N', NB, NRHS, NB, -1D0, DL(1,(N-2)*NB +1), NB, F((N-2)*NB+1,1), LDF, 1D0,
     +     F((N-1)*NB+1,1), LDF)
```

後方置換

```
      …
! 後方置換
! ループの外で X_N を計算して配列 F に格納
      CALL DTRSM('L', 'U', 'N', 'N', NB, NRHS, 1D0, D(1,(N-1)*NB+1), NB, F((N-1)*NB+1,1), LDF)
! ループの外で X_N-1 を計算して配列 F に格納
      CALL DGEMM('N', 'N', NB, NRHS, NB, -1D0, DU1(1,(N-2)*NB +1), NB, F((N-1)*NB+1,1), LDF, 1D0,
     +    F((N-2)*NB+1,1), LDF)
      CALL DTRSM('L', 'U', 'N', 'N', NB, NRHS, 1D0, D(1,(N-2)*NB+1), NB, F((N-2)*NB+1,1), LDF)
! ループ内で配列 F に格納されているコンポーネント X_K を計算
      DO K = N-2, 1, -1
          CALL DGEMM('N','N',NB, NRHS, NB, -1D0, DU1(1,(K-1)*NB +1), NB, F(K*NB+1,1), LDF, 1D0,
     +         F((K-1)*NB+1,1), LDF)
          CALL DGEMM('N','N',NB, NRHS, NB, -1D0, DU2(1,(K-1)*NB +1), NB, F((K+1)*NB+1,1), LDF, 1D0,
     +         F((K-1)*NB+1,1), LDF)
```

```
        CALL DTRSM('L', 'U', 'N', 'N', NB, NRHS, 1D0, D(1,(K-1)*NB+1), NB, F((K-1)*NB+1,1), LDF)
    END DO
    …
```

使用するルーチン

タスク	ルーチン	説明
ピボット置換を適用する	dswap	ベクトルを別のベクトルとスワップします。
下三角係数行列と上三角係数行列を用いて連立線形方程式を解く	dtrsm	三角行列を含む方程式を解きます。
右辺のブロックを更新する	dgemm	一般的な行列を含む行列-行列の積を計算します。

説明

注
サイズ *NB* x *NB* のブロックの一般的なブロック三重対角行列は、帯域幅 4**NB*-1 の帯行列として扱い、インテル® MKL LAPACK の帯行列を因数分解するサブルーチン (?gbtrf) と、帯行列を解くサブルーチン (?gbtrs) を呼び出して解くことができます。 しかし、ブロック行列を帯行列として格納すると、帯の多くのゼロ要素が非ゼロとして扱われて計算中に処理されるため、この手法で説明したアプローチで必要な浮動小数点計算は少なくなります。 より大きな *NB* では影響も大きくなります。 帯行列をブロック三重対角行列として扱うこともできますが、 ブロックに非ゼロとして扱われる多くのゼロが含まれるため、この格納手法は効率的ではありません。 このため、帯格納手法とブロック三重対角格納手法、およびそれらのソルバーは、互いに補完的な手法として考えるべきです。

次の連立線形方程式について考えます。

$$AX = \begin{bmatrix} D_1 & C_1 & 0 & \cdots & 0 & 0 & 0 \\ B_1 & D_2 & C_2 & \cdots & 0 & 0 & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & B_{N-2} & D_{N-1} & C_{N-1} \\ 0 & 0 & 0 & \cdots & 0 & B_{N-1} & D_N \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_{N-1} \\ X_N \end{bmatrix} = \begin{bmatrix} F_1 \\ F_2 \\ \vdots \\ F_{N-1} \\ F_N \end{bmatrix}$$

ブロック三重対角係数行列 *A* は次のように因数分解されると仮定します。

$$A = L \cdot U$$

$$= \left(\Pi_1 \begin{bmatrix} L_{11} \\ L_{21} \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \Pi_1\Pi_2 \begin{bmatrix} 0 \\ L_{22} \\ L_{32} \\ 0 \\ \vdots \\ 0 \end{bmatrix} \cdots \Pi_1\Pi_2 \cdots \Pi_{N-2} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ L_{N-2,N-2} \\ L_{N-1,N-2} \\ 0 \end{bmatrix} \Pi_1\Pi_2 \cdots \Pi_{N-2}\Pi_{N-1} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 0 \\ L_{N-1,N-1} \\ L_{N,N-1} \end{bmatrix} \Pi_1\Pi_2 \cdots \Pi_{N-2}\Pi_{N-1} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ L_{NN} \end{bmatrix} \right)$$

$$\cdot \begin{bmatrix} U_{11} & U_{12} & U_{13} & 0 & \cdots & 0 & 0 & 0 \\ 0 & U_{22} & U_{23} & U_{24} & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & U_{N-2,N-2} & U_{N-2,N-1} & U_{N-2,N} \\ 0 & 0 & 0 & 0 & \cdots & 0 & U_{N-1,N-1} & U_{N-1,N} \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & U_{NN} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ \vdots \\ X_{N-1} \\ X_N \end{bmatrix} = \begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ \vdots \\ F_{N-1} \\ F_N \end{bmatrix}$$

使用している用語の定義は、「一般的なブロック三重対角行列の因数分解」を参照してください。

式は 2 つの連立線形方程式に分解されます。

$$UX = Y, LY = F$$

2 つ目の方程式を拡張します。

$$LY = \Pi_1 \begin{bmatrix} L_{11} \\ L_{21} \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} Y_1 + \Pi_1\Pi_2 \begin{bmatrix} 0 \\ L_{22} \\ L_{32} \\ 0 \\ \vdots \\ 0 \end{bmatrix} Y_2 + \cdots + \Pi_1\Pi_2 \cdots \Pi_{N-2} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ L_{N-2,N-2} \\ L_{N-1,N-2} \\ 0 \end{bmatrix} Y_{N-2}$$

$$+ \Pi_1\Pi_2 \cdots \Pi_{N-2}\Pi_{N-1} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 0 \\ L_{N-1,N-1} \\ L_{N,N-1} \end{bmatrix} Y_{N-1} + \Pi_1\Pi_2 \cdots \Pi_{N-2}\Pi_{N-1} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ L_{NN} \end{bmatrix} Y_N = \begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ \vdots \\ F_{N-1} \\ F_N \end{bmatrix}$$

Y_1 を見つけるには、最初に置換 Π_1^{T} を適用する必要があります。 この置換は、右辺の最初の 2 つのブロックのみ変更します。

$$\begin{bmatrix} F_1' \\ F_2' \\ F_3 \\ \vdots \\ F_{N-1} \\ F_N \end{bmatrix} = \Pi_1^T \begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ \vdots \\ F_{N-1} \\ F_N \end{bmatrix}$$

置換をローカルに適用します。

$$\begin{bmatrix} F_1' \\ F_2' \end{bmatrix} = P_1^T \begin{bmatrix} F_1 \\ F_2 \end{bmatrix}$$

Y_1 が見つかりました。

$$Y_1 = L_{11}^{-1} F_1'$$

Y_1 を見つけた後、同様の計算を繰り返してほかの値 (Y_2, Y_3, ..., Y_{N-2}) を見つけます。

注

Π_{N-1} の異なる構造 (「一般的なブロック三重対角行列の因数分解」を参照) は、同じ方程式を Y_{N-1} と Y_N の計算に使用できず、ループの外で計算する必要があることを意味します。

Y を見つける方程式を用いるアルゴリズムは次のとおりです。

do for $k = 1$ to $N-2$

$$\begin{bmatrix} F_k \\ F_{k+1} \end{bmatrix} := P_k^T \begin{bmatrix} F_k \\ F_{k+1} \end{bmatrix}$$

$Y_k = L_{kk}^{-1} F_k$ //using ?trsm

$F_k := F_k - L_{k,k-1} Y_{k-1}$ //update right hand side using ?gemm

end do

$$\begin{bmatrix} F_{N-1} \\ F_N \end{bmatrix} := P_N^T \begin{bmatrix} F_{N-1} \\ F_N \end{bmatrix}$$

$Y_{N-1} = L_{N-1,N-1}^{-1} F_{N-1}$ // ?trsm

$F_N := F_N - L_{N,N-1} Y_{k-1}$ // update right hand side using ?gemm

$Y_N = L_{N,N}^{-1} F_N$ // ?trsm

$UX = Y$ 方程式は次のように表現できます。

$$\begin{bmatrix} U_{11} & U_{12} & U_{13} & 0 & \cdots & 0 & 0 & 0 \\ 0 & U_{22} & U_{23} & U_{24} & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & 0 & 0 & \cdots & U_{N-2,N-2} & U_{N-2,N-1} & U_{N-2,N} \\ 0 & 0 & 0 & 0 & \cdots & 0 & U_{N-1,N-1} & U_{N-1,N} \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & U_{NN} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ \vdots \\ X_{N-1} \\ X_N \end{bmatrix} = \begin{bmatrix} Y_1 \\ Y_2 \\ Y_3 \\ \vdots \\ Y_{N-1} \\ Y_N \end{bmatrix}$$

これらの方程式を解くアルゴリズムは次のとおりです。

$X_N = U_{NN}^{-1} Y_N$ // ?trsm

$X_{N-1} = U_{N-1,N-1}^{-1}(Y_{N-1} - U_{N-1,N} X_N)$ // ?gemm followed by ?trsm

do for $k = N-1$ to 1 in steps of -1

$X_k = Y_k - U_{k,k+1} X_{k+1}$ // ?gemm

$X_k := X_k - U_{k,k+2} X_{k+2}$ // ?gemm

$X_k := U_{k,k}^{-1} X_k$ // ?trsm

end do

4 フーリエ積分の評価

目的

高速フーリエ変換 (FFT) を使用して連続するフーリエ変換積分を数値的に評価する

$$F(\xi) = \int_{-\infty}^{+\infty} f(x)\, \exp(-i\xi x)\, dx$$

ソリューション

実数値関数 *f*(*x*) が範囲 [*a*, *b*] 外のゼロで、*N* 個の等距離のポイント $x_n = a + nT/N$ (ここで、*T* = |b - a| および *n* = 0, 1, ... , *N*-1) でサンプリングされると仮定します。 FFT を使用して、ポイント $\xi_k = k2\pi/T$ (ここで、*k* = 0, 1, ... , *N*/2) の積分を評価します。

インテル® マス・カーネル・ライブラリー (インテル® MKL) の FFT インターフェイスの使用 (C/C++)

```
float *f;   // input: f[n] = f(a + n*T/N), n=0...N-1
complex *F; // output: F[k] = F(2*k*PI/T), k=0...N/2
DFTI_DESCRIPTOR_HANDLE h = NULL;
DftiCreateDescriptor(&h,DFTI_SINGLE,DFTI_REAL,1,(MKL_LONG)N);
DftiSetValue(h,DFTI_CONJUGATE_EVEN_STORAGE,DFTI_COMPLEX_COMPLEX);
DftiSetValue(h,DFTI_PLACEMENT,DFTI_NOT_INPLACE);
DftiCommitDescriptor(h);
DftiComputeForward(h,f,F);
for (int k = 0; k <= N/2; ++k)
{
  F[k] *= (T/N)*complex( cos(2*PI*a*k/T), -sin(2*PI*a*k/T) );
}
```

説明

評価は積分の階段関数近似に基づき、この派生に従います。

$$\begin{aligned} F_k = F(\xi_k) &= \int_a^b f(x)\, \exp\bigl(-i2\pi x\, k/T\bigr)\, dx \\ &\approx (T/N) \sum_{n=0}^{N-1} f(x_n)\, \exp\bigl(-i2\pi (a + nT/N)\, k/T\bigr) \\ &= (T/N) \exp\bigl(-i2\pi\, a\, k/T\bigr) \sum_{n=0}^{N-1} f_n\, \exp(-i2\pi\, kn/N). \end{aligned}$$

最後の行の合計は定義による FFT です。 関数 *f* のサポートがゼロ付近で対称的に拡張される、つまり [*a*, *b*] = [-*T*/2, *T*/2] の場合、合計の前の係数は $(T/N)(-1)^k$ になります。

関数 *f* が実数値の場合、$F(\xi_k) = \text{conj}(F(\xi_{N-k}))$ になります。 実数から複素数 FFT の最初の $N/2 + 1$ 複素数値は、実数入力とほぼ同じメモリーを占めます。共役により全体の結果を計算するのに十分です。 FFT 計算が実数から複素数への変換を行うように構成されている場合、さらに複素数から複素数 FFT の約半分の時間がかかります。

高速フーリエ変換を使用したコンピューター・トモグラフィー・イメージの復元

5

目的

高速フーリエ変換 (FFT) 関数を使用してコンピューター・トモグラフィー (CT) データから元のイメージを復元する。

ソリューション

表記規則:

- インデックス範囲の表記には MATLAB* で使われている表記規則を採用しています。
  例えば、$k=-q:q$ は、$k=-q, -q+1, -q+2,\ldots, q-1, q$ を意味します。
- $f(x)$ は関数 f のポイント x の値を意味し、$f[n]$ は離散データセット f の n 番目の要素の値を意味します。

仮定:

- 2 次元 (2D) イメージの密度 $f(x, y)$ は単位円の外部で消えます。
  $x^2 + y^2 > 1$ の場合 $f = 0$
- CT データは、角度 $\theta_j = j\pi/p$ で取得したイメージの p の投影からなります。ここで、$j = 0:p - 1$ です。
- 各投影には、次の線に沿ったイメージの積分に近似する $2q + 1$ 密度値 $g[j, l] = g(\theta_j , s_l)$ が含まれます。
  $(x, y) = (-t \sin\theta_j + s_l \cos\theta_j , t \cos\theta_j + s_l \sin\theta_j)$
  ここで、$l = -q:q$、$s_l = l/q$、t は積分パラメーターです。

離散イメージ復元アルゴリズムは次のステップからなります。

1. p 1 次元 (1D) フーリエ変換 ($j = 0:p - 1$ および $r = -q:q$) を評価します。

$$g_1(\theta_j, \pi r/q) = (2/\sqrt{2\pi}) \sum_{\ell=-q}^{q} g[j, \ell] e^{-i\pi r\ell/q}$$

2. $g_1[j, r]$ をラジアルグリッド $(\pi r/q)(\cos\theta_j , \sin\theta_j)$ からデカルトグリッド $(\xi, \eta) = (-q:q, -q:q)$ に補間し、$f_2(\xi\pi/q, \pi\eta/q)$ を取得します。
3. 逆 2 次元複素数-複素数 FFT を評価して、複素数値の復元イメージ f_1 を取得します。

$$f_1[m, n] = (2\pi)^{-1} \sum_{\xi=-q}^{q} \sum_{\eta=-q}^{q} f_2[\xi, \eta] e^{i\pi m\xi/q} e^{i\pi n\eta/q},$$

ここで、$f(m/q, n/q) \approx f_1[m, n]$ ($m = -q:q$ および $n = -q:q$) です。

ステップ 1 と 3 の計算は、インテル® MKL FFT インターフェイスを呼び出します。 ステップ 2 の計算は、インテル® MKL FFT で使用されるデータレイアウトに合わせた、単純なバージョンの補間を実装します。

元の CT イメージの復元 (C/C++)

```
// 宣言
int Nq = 2*(q+1); // インプレース r2c FFT 用の空間
void    *gmem = mkl_malloc( sizeof(float)*p*Nq, 64 );
float    *g = gmem; // g[j*Nq + ell+q]
complex *g1 = gmem; // g1[j*Nq/2 + r+q]

// g を CT データで初期化
for (int j = 0; j < p; ++j)
for (int ell = 0; ell < 2*q+1; ++ell) {
  g[j*Nq + ell+q] = get_g(theta_j,s_ell);
}

// ステップ 1: 1D 実数-複素数 FFT を構成して計算
DFTI_DESCRIPTOR_HANDLE h1 = NULL;
DftiCreateDescriptor(&h1,DFTI_SINGLE,DFTI_REAL,1,(MKL_LONG)2*q);
DftiSetValue(h1,DFTI_CONJUGATE_EVEN_STORAGE,DFTI_COMPLEX_COMPLEX);
DftiSetValue(h1,DFTI_NUMBER_OF_TRANSFORMS,(MKL_LONG)p);
DftiSetValue(h1,DFTI_INPUT_DISTANCE,(MKL_LONG)Nq);
DftiSetValue(h1,DFTI_OUTPUT_DISTANCE,(MKL_LONG)Nq/2);
DftiSetValue(h1,DFTI_FORWARD_SCALE,fscale);
DftiCommitDescriptor(h1);
DftiComputeForward(h1,g); // gmem に g1 が含まれる

// ステップ 2: g1 を f2 に補間 - ここでは省略
complex *f = mkl_malloc( sizeof(complex) * 2*q * 2*q, 64 );

// ステップ 3: 2D 複素数-複素数 FFT を構成して計算
DFTI_DESCRIPTOR_HANDLE h3 = NULL;
MKL_LONG sizes[2] = {2*q, 2*q};
DftiCreateDescriptor(&h3,DFTI_SINGLE,DFTI_COMPLEX,2,sizes);
DftiCommitDescriptor(h3);
DftiComputeBackward(h3,f); // f が複素数値で復元される
```

ソースコード、イメージファイル、メイクファイル: サンプル (http://software.intel.com/en-us/mkl_cookbook_samples) の `fft-ct` フォルダーを参照してください。

説明

このコードは、最初に `DftiComputeForward` 関数の単一の呼び出しで 1 次元フーリエ変換のバッチを計算するインテル® MKL FFT ディスクリプターを構成した後、バッチ変換を計算します。 複数の変換の距離は、対応する領域 (入力は実数、出力は複素数) の要素で設定されます。 デフォルトでは、変換はインプレースです。

メモリーのフットプリントを小さくするため、FFT は*インプレース* で計算されます。つまり、計算の結果は入力データに上書きされます。 インプレースの実数-複素数 FFT では、FFT の結果を格納するのに必要なメモリーが入力よりもやや多くなるため、入力配列は余分な空間を予約します。

ステップ 1 の入力で、配列 `g` には p x (2q+1) 実数値のデータ要素 $g(\theta_j, s_l)$ が含まれます。 このステップの出力の同じメモリーには、p x (q + 1) 複素数値の出力要素 $g_1(\theta_j, \pi r/q)$ が含まれます。 結果の複素共役部分は格納されません。そのため、配列 `g1` は、r の q + 1 値のみを指します。

g_1 から f_2 に補間するため、ステップ 3 の逆 FFT の複素数値のデータ $f_2(\xi, \eta)$ と複素数値の出力 $f_1(x, y)$ を格納する追加の配列 `f` が割り当てられます。 補間ステップはインテル® MKL 関数を呼び出しません。C++ 実装は、この手法のソースコード (`main.cpp`) の `step2_interpolation` 関数を参照してください。 補間の最も単純な実装は次のとおりです。

- 単位円の内部のすべての (ξ, η) で、最も近い $(\theta_j , \pi r/q)$ を見つけて、$g_1(\theta_j , \pi r/q)$ の値を f_2 に使用します。
- 単位円の外部のすべての (ξ, η) で、f_2 を 0 に設定します。
- 区間 $-\pi < \theta_j < 0$ に対応する (ξ, η) の場合は、実数-複素数変換の結果の共役偶数プロパティー $g_1(\theta, \omega)$=conj(g(-θ, -ω)) を使用します。

ステップ 1 の FFT は、計算された出力が $e^{i(\pi r/q)q} = (-1)^r$ で乗算される、表現区間の半分でオフセットされたデータに適用されることに注意してください。 個別のパスで修正する代わりに、補間は乗数を考慮しています。

同様に、ステップ 3 の 2D FFT はイメージの中心を隅にシフトする出力を生成し、ステップ 2 はステップ 3 に入力をシフトする過程でこれを防いでいます。

ステップ 3 は、配列 `f` に含まれる補間されたデータについて 2 次元 $(2q)$ x $(2q)$ 複素数-複素数 FFT を計算します。 この計算の後、複素数値のイメージ f_1 からビジュアルピクチャーへの変換が行われます。 CT イメージ復元を実装する完全な C++ プログラムは、この手法のソースコード (`main.cpp`) を参照してください。

金融市場のデータストリームにおけるノイズ・フィルタリング 6

目的

一部の株式の価格変動が大規模な株式ポートフォリオのほかの株式の価格変動にどのように影響を及ぼすかを特定する。

ソリューション

データ全体の依存関係を表す相関行列を、信号行列とノイズ行列の 2 つの成分に分割します。 信号行列から、株式間の依存関係を正確に評価することができます。 アルゴリズム ([Zhang12]、[Kargupta02]) は、累積データの相関行列の固有値を調べることにより、有用な情報からノイズ成分を分割する固有状態ベースのアプローチを取ります。

インテル® MKL サマリー統計は、ストリーミング・データの相関行列を計算する機能を提供します。 インテル® MKL LAPACK には、さまざまな特性や格納形式の対称行列の固有値と固有ベクトルを計算する計算ルーチン群が含まれています。

オンライン・ノイズ・フィルタリング・アルゴリズムの手順を次に示します。

1. λ_{min} と λ_{max} (ノイズ固有状態の間隔の境界) を計算します。
2. データストリームから新しいデータブロックを取得します。
3. 最新のデータブロックを使用して相関行列を更新します。
4. 間隔 [λ_{min}, λ_{max}] に属する相関行列の固有値を検索して、ノイズ成分を定義する固有値と固有ベクトルを計算します。
5. ステップ 4 で計算した固有値と固有ベクトルを組み合わせてノイズ成分の相関行列を計算します。
6. 相関行列全体からノイズ成分を取り除いて信号成分の相関行列を計算します。 さらにデータがある場合は、ステップ 2 に戻ります。

ソースコード: サンプル (http://software.intel.com/en-us/mkl_cookbook_samples) の `nf` フォルダーを参照してください。

初期化

相関解析タスクとパラメーターを初期化します。

```
VSLSSTaskPtr task;
double *x, *mean, *cor;
double W[2];
MKL_INT x_storage, cor_storage;

...

scanf("%d", &m);            // ブロックの観測数
scanf("%d", &n);            // 株式の数 (タスクの次元)

...

/* メモリーを割り当て */
nfAllocate(m, n, &x, &mean, &cor, ...);
```

```
/* サマリー統計タスク構造を初期化 */
nfInitSSTask(&m, &n, &task, x, &x_storage, mean, cor, &cor_storage, W);
...

/* メモリーの割り当て */
void nfAllocate(MKL_INT m, MKL_INT n, double **x, double **mean, double **cor,
                ...)
{
    *x = (double *)mkl_malloc(m*n*sizeof(double), ALIGN);
    CheckMalloc(*x);

    *mean = (double *)mkl_malloc(n*sizeof(double), ALIGN);
    CheckMalloc(*mean);

    *cor = (double *)mkl_malloc(n*n*sizeof(double), ALIGN);
    CheckMalloc(*cor);
...
}
/* サマリー統計タスク構造を初期化 */
void nfInitSSTask(MKL_INT *m, MKL_INT *n, VSLSSTaskPtr *task, double *x,
                  MKL_INT *x_storage, double *mean, double *cor,
                  MKL_INT *cor_storage, double *W)
{
    int status;

    /* VSL サマリー統計タスクを作成 */
    *x_storage = VSL_SS_MATRIX_STORAGE_COLS;
    status = vsldSSNewTask(task, n, m, x_storage, x, 0, 0);
    CheckSSError(status);

    /* タスクの重みのレジスター配列 */
    W[0] = 0.0;
    W[1] = 0.0;
    status = vsldSSEditTask(*task, VSL_SS_ED_ACCUM_WEIGHT, W);
    CheckSSError(status);

    /* 相関行列計算にフルストレージを使用する
       タスク・パラメーターの初期化 */
    *cor_storage = VSL_SS_MATRIX_STORAGE_FULL;
    status = vsldSSEditCovCor(*task, mean, 0, 0, cor, cor_storage);
    CheckSSError(status);
}
```

計算

データの各ブロックについてノイズ・フィルタリングのステップを実行します。

```
/* ノイズ成分を定義するしきい値を設定 */
sqrt_n_m = sqrt((double)n / (double)m);
lambda_min = (1.0 - sqrt_n_m) * (1.0 - sqrt_n_m);
lambda_max = (1.0 + sqrt_n_m) * (1.0 + sqrt_n_m);

...
/* データブロックをループ */
for (i = 0; i < n_block; i++)
{
    /* データの次の部分を読む */
    nfReadDataBlock(m, n, x, fh);
```

```
    /* "信号" と "ノイズ" 共分散の評価を更新 */
    nfKernel(m, n, lambda_min, lambda_max, x, cor, cor_copy,
             task, eval, evect, work, iwork, isuppz,
             cov_signal, cov_noise);
}
...

void nfKernel(…)
{
...

    /* FAST メソッドを使用して相関行列の評価を更新 */
    errcode = vsldSSCompute(task, VSL_SS_COR, VSL_SS_METHOD_FAST);
    CheckSSError(errcode);

...
    /* 相関行列の固有値と固有ベクトルを計算 */
    dsyevr(&jobz, &range, &uplo, &n, cor_copy, &n, &lmin, &lmax,
           &imin, &imax, &abstol, &n_noise, eval, evect, &n, isuppz,
           work, &lwork, iwork, &liwork, &info);

    /* 共分散行列の "信号" と "ノイズ" 部分を計算 */
    nfCalculateSignalNoiseCov(n, n_signal, n_noise,
        eval, evect, cor, cov_signal, cov_noise);
}

...
static int nfCalculateSignalNoiseCov(int n, int n_signal, int n_noise,
        double *eval, double *evect, double *cor, double *cov_signal,
        double *cov_noise)
{
    int i, j, nn;

    /* SYRK パラメーター */
    char uplo, trans;
    double alpha, beta;

    /* 共分散行列の "ノイズ" 部分を計算 */
    for (j = 0; j < n_noise; j++) eval[j] = sqrt(eval[j]);

    for (i = 0; i < n_noise; i++)
        for (j = 0; j < n; j++)
            evect[i*n + j] *= lambda[i];

    uplo  = 'U';
    trans = 'N';
    alpha = 1.0;
    beta  = 0.0;
    nn = n;

    if (n_noise > 0)
    {
        dsyrk(&uplo, &trans, &nn, &n_noise,  &alpha, evect, &nn,
              &beta, cov_noise, &nn);
    }
    else
    {
        for (i = 0; i < n*n; i++) cov_noise[i] = 0.0;
    }
```

```
    /* 共分散行列の "信号" 部分を計算 */
    if (n_signal > 0)
    {
        for (i = 0; i < n; i++)
            for (j = 0; j <= i; j++)
                cov_signal[i*n + j] = cor[i*n + j] - cov_noise[i*n + j];
    }
    else
    {
        for (i = 0; i < n*n; i++) cov_signal[i] = 0.0;
    }

    return 0;
}
```

リソースの解放

タスクを削除し、関連付けられたリソースを解放します。

```
errcode = vslSSDeleteTask(task);
CheckSSError(errcode);
MKL_Free_Buffers();
```

使用するルーチン

タスク	ルーチン
サマリー統計タスクを初期化して解析用のオブジェクト (データセット、サイズ (変数の数と観測の数)、格納形式)) を定義します。	vsldSSNewTask
相関行列を保持するメモリーを指定します。	vsldSSEditCovCor
処理した観測の重みの合計を保持する 2 要素の配列を指定します (データストリームの評価の正確な計算に必要です)。	vsldSSEditTask
計算タイプ (VSL_SS_COR) と計算メソッド (VSL_SS_METHOD_FAST) を指定して主計算ドライバーを呼び出します。	vsldSSCompute
タスクに関連付けられているリソースの割り当てを解除します。	vslSSDeleteTask
相関行列の固有値と固有ベクトルを計算します。	dsyevr
対称ランク-k 更新を行います。	dsyrk

説明

アルゴリズムのステップ 4 では、対称行列の固有値問題を解きます。 オンライン・ノイズ・フィルタリング・アルゴリズムを使用するには、データのノイズを定義する、事前定義間隔 [λ_{min}, λ_{max}] に属する固有値を計算する必要があります。 LAPACK ドライバールーチン `?syevr` は、この種の問題を解くデフォルトのルーチンです。 `?syevr` インターフェイスで、呼び出し元は、固有値を検索する範囲の下限と表現のペア (このケースでは λ_{min} と λ_{max}) を指定できます。

固有ベクトルは直交行列 *A* を含む配列の列として返され、固有値は対角行列 *Diag* の要素を含む配列で返されます。 ノイズ成分の相関行列は、$A*Diag*A^T$ を計算して取得できます。 しかし、ここでは、2 つの一般的な行列乗算によってノイズ相関行列を構築する代わりに、1 つの対角行列乗算と 1 つのランク n 更新操作を使用してより効率良く計算しています。

$$Cor_{noise} = ADiagA^T = A\sqrt{Diag}\,\sqrt{Diag}^T A^T = \left(A\sqrt{Diag}\right)\left(A\sqrt{Diag}\right)^T$$

ランク n 更新操作用に、インテル® MKL は BLAS 関数 `?syrk` を提供しています。

文献目録 (英語)

スパースソルバー

[Amos10] Ron Amos. *Lecture 3: Solving Equations Using Fixed Point Iterations,* University of Wisconsin CS412: Introduction to Numerical Analysis, 2010 (http://pages.cs.wisc.edu/~holzer/cs412/lecture03.pdf).

[Smith86] G.D. Smith. *Numerical Solution of Partial Differential Equations: Finite Difference Methods,* Oxford Applied Mathematics & Computing Science Series, Oxford University Press, USA, 1986.

数値演算

[Zhang12] Zhang Zhang, Andrey Nikolaev, and Victoriya Kardakova. *Optimizing Correlation Analysis of Financial Market Data Streams Using Intel® Math Kernel Library,* Intel Parallel Universe Magazine, 12: 42-48, 2012.

[Kargupta02] Hillol Kargupta, Krishnamoorthy Sivakumar, and Samiran Ghost. *A Random Matrix-Based Approach for Dependency Detection from Data Streams,* Proceedings of Principles of Data Mining and Knowledge Discovery, PKDD 2002: 250-262. Springer, New York, 2002.