LAPACK・BLASを用いた連立1次方程式の数値計算

理化学研究所計算科学研究センター　特別研究員
寺尾　剛史

今回の目標

- 連立1次方程式の数値解法の実装（密行列）
  - LU分解
  - 残差反復改良を用いた混合精度数値計算法
- BLAS・LAPACKを用いたコードの作成に慣れる
  - 密行列に対する高性能なコードの作成

連立1次方程式の数値計算

- 正則な$n \times n$行列A、 n次ベクトルbに対して

$$Ax = b$$

  を満たす、ベクトルxを求めたい。
- 数値計算法は、**計算速度**と**精度**が求められる。
- **計算速度**はわかりやすい（実行時間を計ればよい）。
- **精度**とは、近似解$\hat{x}$に対して

$$\|\hat{x} - x\|$$

  がどれだけ小さいかである。（どう計算する？）
  - 精度保証付き数値計算
- 残差ベクトルのノルム$\|b - A\hat{x}\|$が代用されることが多い。

連立1次方程式について

- 連立1次方程式は、様々な分野に応用される。
- 密行列：
  - LU分解、コレスキー分解、QR分解
- 疎行列：
  - 定常反復法：Jacobi法、Gauss-Seidel法、SOR法
  - 非定常反復法：CG法、ICCG法、Bi-CG法
- 問題の計算コストの大部分を占める場合がある。
- 問題や係数行列で有効な手法が異なる。

LU分解

- $n \times n$の正則な行列Aに対するLU分解は、上三角行列Lと下三角行列Uを用いて

$$A = L \times U$$

の形に分解する。（計算回数：$\frac{2}{3}n^3$ flops）

- 以上のように分解できたとき、連立1次方程式$Ax = b$は

$$x = U^{-1}(L^{-1}b)$$

で計算できる。（計算回数：$O(n^2)$ flops）

- LU分解を用いた連立1次方程式の解法では、LU分解に多くのコストが掛かる。

LU分解（軸交換無し）

- $n = 3$の場合

$$A = \begin{pmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{pmatrix}$$

$$= \begin{pmatrix} u_{11} & u_{12} & u_{13} \\ l_{21}u_{11} & l_{21}u_{12} + u_{22} & l_{21}u_{13}u_{23} \\ l_{31}u_{11} & l_{31}u_{12} + l_{32}u_{22} & l_{31}u_{13} + l_{32}u_{23} + u_{33} \end{pmatrix}$$

LU分解の例（軸交換無し）

$$A = \begin{pmatrix} 5 & 6 & 7 \\ 10 & 20 & 23 \\ 15 & 50 & 67 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & l_{32} & 1 \end{pmatrix} \begin{pmatrix} 5 & 6 & 7 \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 4 & 1 \end{pmatrix} \begin{pmatrix} 5 & 6 & 7 \\ 0 & 8 & 9 \\ 0 & 0 & u_{33} \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 4 & 1 \end{pmatrix} \begin{pmatrix} 5 & 6 & 7 \\ 0 & 8 & 9 \\ 0 & 0 & 10 \end{pmatrix}$$

$$u_{22} = 20 - 2 * 6 = \mathbf{8}$$
$$u_{23} = 23 - 2 * 7 = \mathbf{9}$$
$$l_{32} = \frac{50 - 3 * 6}{u_{22}} = \mathbf{4}$$

$$u_{33} = 67 - 3 * 7 - 4 * 9 = \mathbf{8}$$

プロクラムの高速化

- 高速な数値計算を行うにはどうする？

逐次実行の高速化	並列計算
• 遅いメモリへのアクセスの削減 • キャッシュ・レジスタブロッキング	• 計算の並列化 • OpenMP、MPI

- 個人で行うには限界がある（手間もかかる）
- HPCのプロが作った数値計算ライブラリを用いることも重要！

BLAS・LAPACKとは

- **BLAS**とは、**B**asic **L**inear **A**lgebra **S**ubprogramsの略
  - 行列やベクトルの基本的な計算を行う関数群
  - 行列積やベクトルのノルム計算など
- **LAPACK**とは、**L**inear **A**lgebra **PACK**ageの略
  - 連立1次方程式や固有値問題など
- IEEE754の単精度、倍精度からなる実数、複素数を要素とするベクトルや行列をサポート
- 密行列に対するコードの高速化や簡略化ができる。

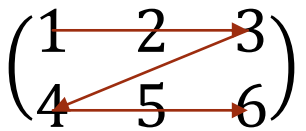
void sgesv_(N, NRHS, A, LDA, IPIV, B, LDB, INFO);

- 単精度でLU分解を用いて$B \leftarrow A^{-1}B$を解く。
- N(入力) – INTEGER
  - 行列Aの行列サイズ
- NRHS（入力）ーINTEGER
  - 右辺ベクトルBの数
- A（入力・出力）ーSINGLE PRECISION
  - （入力）係数行列
  - （出力）LU分解の計算結果
- LDA（入力）ーINTEGER
  - 行列Aの最初の次元数。$\max(n, 1)$ である必要がある。
- IPIV（出力）ーINTEGER
  - LU分解での軸交換の情報
- B（入力・出力）ーSINGLE PRECISION
  - （入力）右辺ベクトル
  - （出力） $A^{-1}B$の数値解
- LDB（入力）ーINTEGER
  - 行列Aの最初の次元数。$\max(n, 1)$ である必要がある。
- INFO（出力）ーINTEGER
  - ＝０：正常
  - ＜０：INFO=-iのときi行目の引数がおかしい
  - ＞０：INFO=iならばU(i,i)=0で計算できない

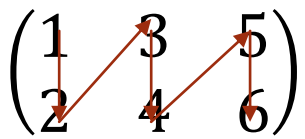
BLAS・LAPACKの注意点

- 引数はすべてポインタで渡す。
- Fortranで記述されているため、C言語では二次元配列の順番に注意する
  - 一次元配列にすれば問題ない

C： A[2][3]



Fortran：



- 今回のサンプルプログラムは、１次元配列を用いているので、気にする必要はない。
- コンパイラオプションを追加する必要がある。
  - OpenBLAS: -lopenblas, LAPACK;: -llapack , MKL: -mkl など

サンプルプログラム

- サンプルを以下からコピーしてください。

  /work/gt65/t65022/LinSys

- 次を実行してコピー：

  >> cp –r ../t65022/LinSys ./LinSys

  >> cd LinSys

  >> ls

  ex01.c　ex02.c　run.sh

- 次でコンパイル：

  >> icc –mkl –qopenmp –O3 ex01.c

- 次で実行：

  >> pjsub run.sh

課題 1

1. ex01.cをSGESV、DGESVルーチンを呼び出し完成させてください
2. 様々な行列サイズで残差と計算性能の比較を行ってください
3. 様々なスレッド数で実行してください

- 実行結果（例）：

(Single) N: 20000, Residual: 2.725259e-01, Time: 1.351563e+00

(Double) N: 20000, Residual: 2.048972e-12, Time: 2.596874e+00

混合精度計算を用いた連立1次方程式の解法

混合精度数値計算

- 複数の精度を用いた数値計算方法
- 例えば、（Single）＋（Double）で（Double）と同程度の精度で数値計算
- 主な目的は：
  - 計算性能の向上
  - 省エネ化
- 今回はLU分解を用いた混合精度計算法を紹介する

連立1次方程式に対する反復改良法

- 連立1次方程式$Ax = b$の数値解を$\hat{x}$とする
- また、$\hat{x}$の誤差を$\Delta x := x - \hat{x}$とする
- このとき、

$$A(\hat{x} + \Delta x) = b$$

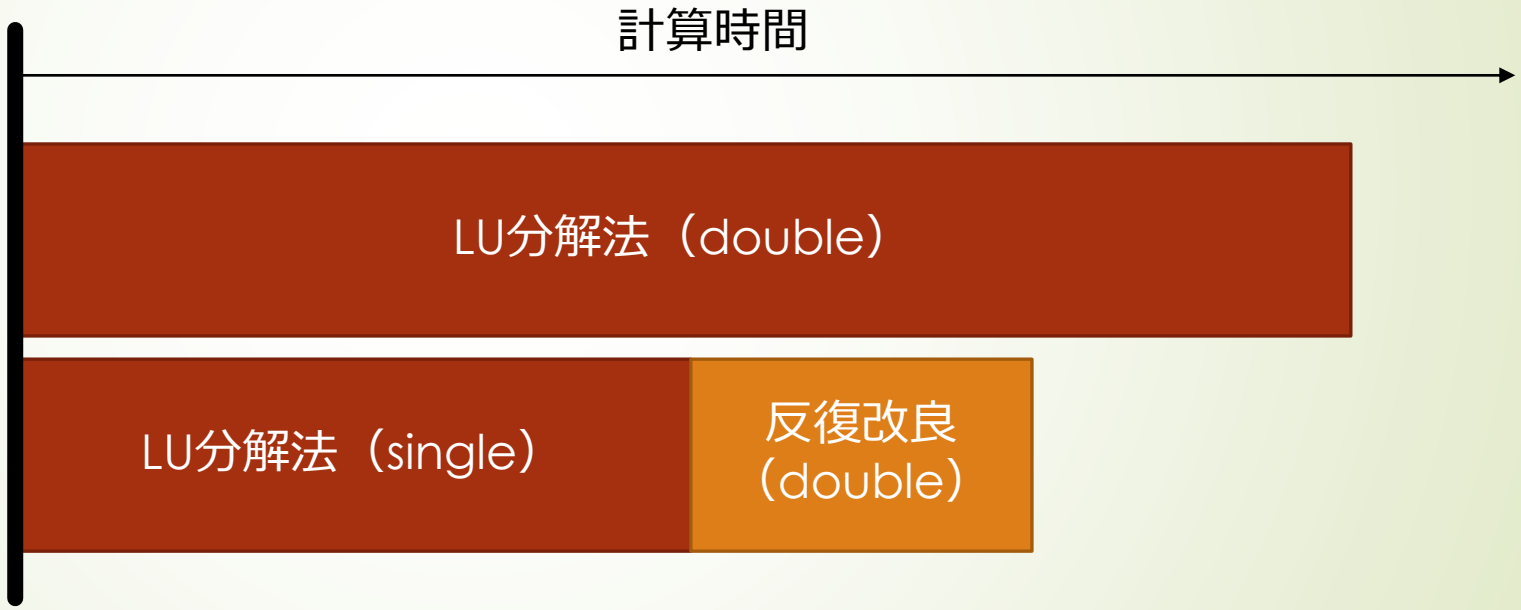
$$\Delta x = A^{-1}(b - Ax)$$

が成り立つ。

- ここで、$b - Ax$には**高精度**な行列ベクトル積が必要
- LU分解の計算結果は1回目で計算出来ているため、計算量は$O(n^2)$ flopsで計算できる。（LU分解は$2/3n^3$ flops）

混合精度計算を用いた連立1次方程式

- LU分解を用いた混合精度計算のイメージ



- 残差反復が効率的に行われる場合、計算時間を削減できる。

混合精度の利点と欠点

- 利点：
  - 主要なコストであるLU分解のコストを低減できる
  - GPUなどの低精度演算が高速な計算機ではさらに有効
- 欠点：
  - 扱える条件数$\kappa(A) \coloneqq \sigma_{max}(A)/\sigma_{min}(A)$が変化
  - 倍精度の場合は$\kappa(A) \approx 10^{16}$、単精度＋倍精度の場合は$\kappa(A) \approx 10^{7}$が限界（単精度のLU分解に依存）
  - 行列やベクトルを保存するためのメモリが必要

void dgemv_(TRANS, M, N, alpha, A, LDA, x, INCX, beta, y, INCY);

- 倍精度で$y \leftarrow \alpha Ax + \beta y$を計算。
- M（入力）ーINTEGER
  - op(A)の行数
- N（入力）ーINTEGER
  - op(A)の列数
- alpha（入力）ーDOUBLE PRECISION
  - スカラ値を指定
- A（入力）ーDOUBLE PRECISION
  - 行列Aの配列
- LDA（入力）ーINTEGER
  - 行列Aの最初の次元数。$\max(n, 1)$ である必要がある。
- x（入力）ーDOUBLE PRECISION
  - ベクトルxの配列
- INCX（入力）ーINTEGER
  - ベクトルxのインクリメント幅
- beta（入力）
  - スカラ値を指定
- y（入力・出力）
  - ベクトルyの配列
- INCY（入力）ーINTEGER
  - ベクトルyのインクリメント幅

void dgetrs_(TRANS, N, NRHS, A,LDA,IPIV, B, LDB, INFO);

- LU分解の計算結果を用いて$B \leftarrow (LU)^{-1}B$を解く。
- N(入力) – INTEGER
  - 行列Aの行列サイズ
- NRHS（入力）ーINTEGER
  - 右辺ベクトルBの数
- A（入力）ーDOUBLE PRECISION
  - （出力）LU分解の計算結果
- LDA（入力）ーINTEGER
  - 行列Aの最初の次元数。$\max(n, 1)$ である必要がある。
- IPIV（入力）ーINTEGER
  - LU分解での軸交換の情報
- B（入力・出力）ーDOUBLE PRECISION
  - （入力）右辺ベクトル
  - （出力） $A^{-1}B$の数値解
- LDB（入力）ーINTEGER
  - 行列Aの最初の次元数。$\max(n, 1)$ である必要がある。
- INFO（出力）ーINTEGER
  - ＝０：正常
  - ＜０：INFO=-iのときi行目の引数がおかしい
  - ＞０：INFO=iならばU(i,i)=0で計算できない

void dcopy_(N, X, INCX, Y, INCY);

- 倍精度配列XをYにコピー
- N(入力) – INTEGER
  - ベクトルXのサイズ
- X（入力）ーDOUBLE PRECISION
  - コピーする配列
- INCX（入力）ーINTEGER
  - Xのインクリメント幅
- Y（出力）ーDOUBLE PRECISION
  - コピーされる配列
- INCY（入力）ーINTEGER
  - Yのインクリメント幅

void daxpy_(N, alpha, X, INCX, Y, INCY);

- $y \leftarrow \alpha x + y$を計算
- N(入力) – INTEGER
  - ベクトルXのサイズ
- alpha（入力）
  - スカラ値を指定
- X（入力）ーDOUBLE PRECISION
  - コピーする配列
- INCX（入力）ーINTEGER
  - Xのインクリメント幅
- Y（出力）ーDOUBLE PRECISION
  - コピーされる配列
- INCY（入力）ーINTEGER
  - Yのインクリメント幅

アルゴリズム

- 混合精度計算を用いた連立1次方程式を解くアルゴリズム

1. $\hat{x} \approx A^{-1}b$ ：LU分解を用いて**単精度**で連立1次方程式を解く(sgesv)
2. $r \leftarrow b$：配列のコピー（dcopy）
3. $r \leftarrow r - A\hat{x}$：行列ベクトル積を**倍精度**計算(dgemv)
4. $r \leftarrow A^{-1}r$ ：LU分解の計算結果を用いて連立1次方程式を解く(dgetrsまたはsgetrs)
5. $\hat{x} \leftarrow \hat{x} + r$：数値解を更新する(daxpyまたはfor文)
6. 2～5を十分に繰り返す

- AやbなどのI情報を保存する必要があることに注意！

課題2

- ex02.cに反復改良法を実装してください
- 反復回数ごとの残差と計算性能の比較を行ってください
- 倍精度で計算した場合と混合精度で計算した場合の比較を行ってください

反復改良のコード例

単精度でLU分解法で解く

```
scopy_(&nn, A, &one, LU, &one);
scopy_(&n, b, &one, x, &one);
sgesv_(&n, &one, LU, &n, IPIV, x, &n, &info);
for (i = 0; i < n * n; i++) {
    AA[i] = A[i];
}
for (I = 0; I < n; i++) {
    xx[i] = x[i];
}
```

反復改良

```
for (i = 0; i < iter; i++) {
  for (j = 0; j < n; j++) rr[j]=b[j];
  dgemv_(&chrN, &n, &n, &dmone, AA, &n, xx, &one, &done, rr, &one);
  for (j = 0; j < n; j++) r[j]=rr[j];
  sgetrs_(&chrN, &n, &one, LU, &n, IPIV, r, &n, &info);
  for (j = 0; j < n; j++) xx[j]=xx[j]+r[j];
}
```

反復改良のコード例

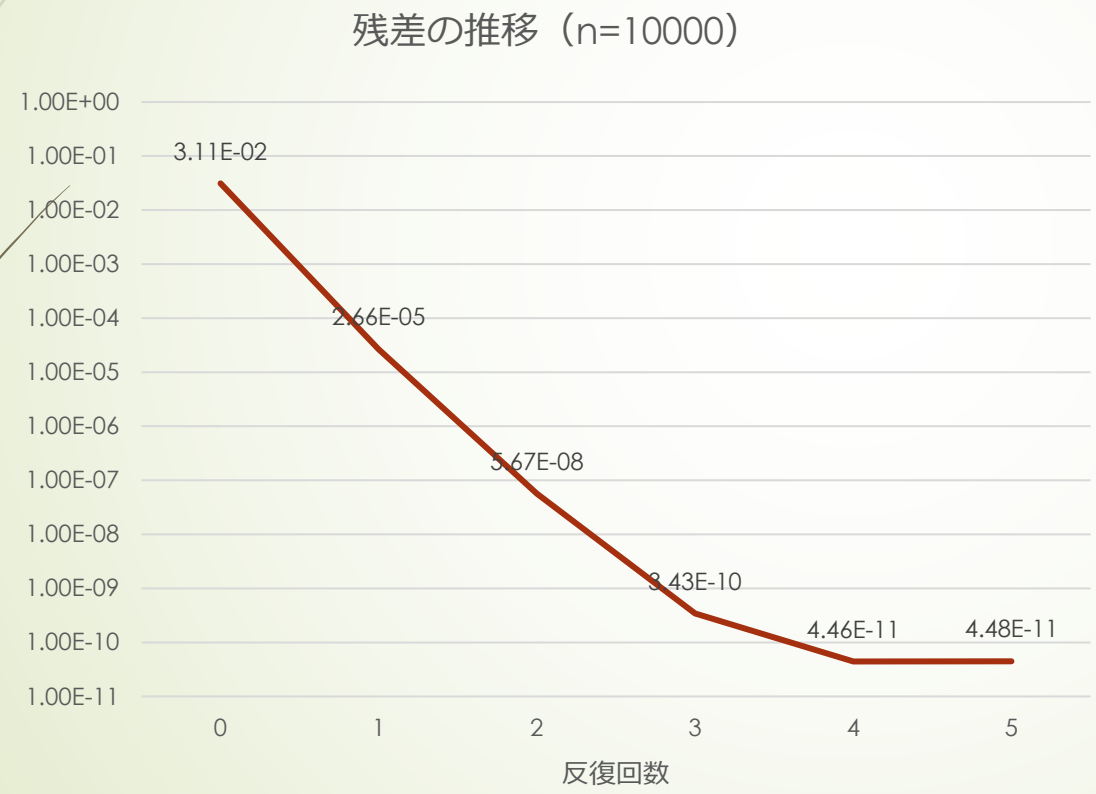
単精度でLU分解法で解く

```
scopy_(&nn, A, &one, LU, &one);
scopy_(&n, b, &one, x, &one);
sgesv_(&n, &one, LU, &n, IPIV, x, &n, &info);
for (i = 0; i < n * n; i++) {
    AA[i] = A[i];
    LU2[i] = LU[i];
}
for (I = 0; I < n; i++) {
    bb[i] = b[i];
    xx[i] = x[i];
}
```

反復改良

```
for (i = 0; i < iter; i++) {
  dcopy_(&n, bb, &one, rr, &one);
  dgemv_(&chrN, &n, &n, &dmone, AA, &n, xx, &one, &done, rr, &one);
  dgetrs_(&chrN, &n, &one, LU2, &n, IPIV, rr, &n, &info);
  daxpy_(&n, &done, rr, &one, xx, &one);
}
```

数値実験の例



- Single：3.11E-02
- Double：9.47E-11
- Mix：4.46E-11
- 4回の反復で倍精度と同等の精度で計算
- 行列サイズ、条件数で変動

おまけ

LAPACK（混合精度）

- 今回は、単精度、倍精度のLAPACKと自作の混合精度計算の比較を行った。
- 実は、混合精度のLAPACK関数は実装されている（**DSGESV：**倍精度入力で混合精度計算）
- どのようなアルゴリズムが動いているかを理解することは非常に重要。
- 今回実装した手法を改良すると、4倍精度やそれ以上の計算精度で解を求めることが出来る。