数値計算講義ノート５
連立一次方程式の数値解法(2)

大石進一

April 30, 2003

§1 ガウスの消去法

Lも計算する

```
for i=1 to n-1
   for j=i+1 to n
      A(j,i) = (A(j,i)/A(i,i));
      for k=i+1 to n
         A(j,k) = A(j,k) - A(j,i) * A(i,k);
```

行列計算の形に書き直すと

```
for i=1 to n-1
   A(i+1:n,i)=A(i+1:n,i)/A(i,i);
   A(i+1:n,i+1:n)=A(i+1:n,i+1:n)-A(i+1:n,i)*A(i,i+1:n);
```

定理 2　計算された後、上書きされたAの狭義下三角部分をMとして$L = I + M$、上三角部分をUとする。計算が最後まで実行されたとき(ゼロで割ることが起きなかったとき)、

$$A = LU \tag{1}$$

となる。

定理2'　Aを$n \times n$行列で正則としよう。$i = 1$から$n - 1$について$A(1 : i, 1 : i)$が正則であるとする(後にピボットを考えることによりこの条件をはずすことができる)。このとき、$A = L * U$と分解できる。ただし、Lは下三角行列でUは上三角行列である。LとUはガウスの消去法によって計算でき、手間は$2/3 * n^3 + O(n^2)$ flopsである。

$Ax = b$を解くための手間

1. $A = LU$と分解する。($2/3n^3$flopsの手間)

2. $Ly = b$を代入によって解く。(n^2flopsの手間)

3. $Ux = y$を代入によって解く。(n^2flopsの手間)

flopsについて

Since linear equation solutions for matrices of O(N) rows
and O(N) columns take O(N^3) operations, when N = 100,
about a million floating-point operations (flops) are needed.

という文章に見られるように、flopsはfloating-point operationsという
意味に用いられる。例えば、つぎのような使い方である。

To see if anything is actually gained the convergence
should be measured with respect to computation time or
flops (floating point operations). We have used flops
because flops are invariant with respect to the machine used.

一方、注意しないといけないのはflopsは

Short for floating-point operations per second, a common benchmark measurement for rating the speed of microprocessors.

というように別の意味でも用いられることである。

A megaFLOPS (MFLOPS) is equal to one million floating-point operations per second, and a gigaFLOPS (GFLOPS) is equal to one billion floating-point operations per second.

であるから、100次元の方程式をmegaFLOPS (MFLOPS)の性能のマシンで解くと1秒かかる。gigaFLOPSの性能のマシンで解けば約0.001秒である。一方、1000次元の方程式をgigaFlopsのマシンで解けば約1秒という訳である。言い換えると、1000次元の連立方程式を約1秒で解くマシンはGFLOPSの性能をもつといえる。

定理2'を証明しよう。まず、次の性質を述べる：

補題１

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \tag{2}$$

とする。ただし、Aは$m \times n$行列でA_{ij}は$m_i \times n_j$行列とする。また、

$$B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} \tag{3}$$

とする。ただし、Bは$n \times k$行列でB_{ij}は$n_i \times k_j$行列とする。このとき、

$$AB = \begin{pmatrix} A_{11} * B_{11} + A_{12} * B_{21} & A_{11} * B_{12} + A_{12} * B_{22} \\ A_{21} * B_{11} + A_{22} * B_{21} & A_{21} * B_{12} + A_{22} * B_{22} \end{pmatrix} \tag{4}$$

が成り立つ。

この補題を用いて、$A = LU$と分解できることを示そう。Aの次元のnに関する帰納法を用いる。$n = 1$のときは単に数であるので$A = 1 * A$と分解で

きる。つぎに$n-1$まで定理が正しいとして、nについて定理が成立することを示そう。

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \tag{5}$$

と書き表す。ただし、A_{11}は$n-1 \times n-1$行列、A_{12}は$n-1 \times 1$行列、A_{21}は$1 \times n-1$行列でA_{22}は1×1行列とする。A_{11}は$n-1$次元ベクトルなので帰納法の仮定から$A_{11} = L_1 * U_1$が成立する。これを使うと、

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} L_1 & 0 \\ A_{21} * \mathsf{inv}(U_1) & 1 \end{pmatrix} \begin{pmatrix} U_1 & \mathsf{inv}(L_1) * A_{12} \\ 0 & x \end{pmatrix} \tag{6}$$

と計算される。ただし、$x = A_{22} - A_{21} * \mathsf{inv}(A_{11}) * A_{12}$ 。実際、

$$\begin{aligned} & \begin{pmatrix} L_1 & 0 \\ A_{21} * \mathsf{inv}(U_1) & 1 \end{pmatrix} \begin{pmatrix} U_1 & \mathsf{inv}(L_1) * A_{12} \\ 0 & x \end{pmatrix} \\ = & \begin{pmatrix} L_1 * U_1 & L_1 * \mathsf{inv}(L_1) * A_{12} \\ A_{21} * \mathsf{inv}(U_1) * U_1 & A_{21} * \mathsf{inv}(U_1) * \mathsf{inv}(L_1) * A_{12} + x \end{pmatrix} \\ = & \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} = A \end{aligned}$$

つぎに、ガウスの消去法のアルゴリズムの手間をカウントしてみよう。

$$
\begin{aligned}
& \sum_{i=1}^{n-1} [\sum_{j=i+1}^{n} 1 + \sum_{j=i+1}^{n} \sum_{k=i}^{n} 2] \\
= & \sum_{i=1}^{n-1} [n-i + \sum_{j=i+1}^{n} 2*(n-i-1)] \\
= & \sum_{i=1}^{n-1} [n-i+2*(n-i-1)*(n-i)] \\
= & \sum_{i=1}^{n-1} [2*(n-i)*(n-i)-(n-i)] \\
= & \sum_{i=1}^{n-1} [2*i*i-i] \\
= & (2/3)*n^3 + O(n^2)
\end{aligned}
$$

ただし、

$$\sum_{i=1}^{n-1} i^2 = n^3/3 + n^2/2 + n/6 \tag{7}$$

を用いた。

§2 ピボッティング

例として

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

を考える。Aとは別にBとして

$$B = \begin{pmatrix} 10^{-4} & 1 \\ 1 & 1 \end{pmatrix}$$

を考える。簡単のため10進数で3桁に四捨五入する系を考える。BをLU分解すると

$$L = \begin{pmatrix} 1 & 0 \\ \mathrm{fl}(1/10^{-4}) & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 10^4 & 1 \end{pmatrix}$$

となる。ここまでに誤差はない。一方、

$$U = \begin{pmatrix} 10^{-4} & 1 \\ 0 & \mathrm{fl}(1 - 10^4) \end{pmatrix} = \begin{pmatrix} 10^{-4} & 1 \\ 0 & -10^4 \end{pmatrix}$$

となる。ここでは、3桁計算のため、小さな誤差が入る。ここで、LUを計算してみよう。

$$LU = \begin{pmatrix} 1 & 0 \\ 10^4 & 1 \end{pmatrix} \begin{pmatrix} 10^{-4} & 1 \\ 0 & -10^4 \end{pmatrix} = \begin{pmatrix} 10^{-4} & 1 \\ 1 & 0 \end{pmatrix} \approx A \neq B \quad (8)$$

となる。これはLU分解に大きな誤差が入ることを意味する。すなわち、この場合のLU分解は数値的に不安定である。

解決策：ピボットを行う。

部分ピボットガウスの消去法

```
for i=1 to n
    |A(k,i)|=max{i<= j <= n} |A(j,i)|となるkを求め記録する。
    if |A(k,i)|=0
        Aが特異である(に近い)と警告してアルゴリズムを終了する。
    elseif k != i
        行iと行kを交換する。
    endif
    A(i+1:n,i)=A(i+1:n,i)/A(i,i);
    A(i+1:n,i+1:n)=A(i+1:n,i+1:n)-A(i+1:n,i)*A(i,i+1:n);
```

定理３　このアルゴリズムはA＝PLUを計算する。

プログラム例としてはつぎを参照されたい。

http://www.netlib.org/clapack/double/dgetf2.c