LAPACK の dgbsv で連立1次方程式を解く

桂田 祐史

2004 年 11 月 13 日 (修正版), 2017 年 10 月 7 日 (組版仕直し)

g77 は古すぎるので、gfortran で動作確認するものか？

1 LAPACK とは

無償で公開されている (http://www.netlib.org/lapack/)、線形計算 (連立 1 次方程式、固有値問題、特異値分解) をするための、高い信頼性を誇る Fortran サブルーチン・ライブラリィであり、非常に評価が高い。これもまた著名な LINPACK, EISPACK の後継ソフトウェアと考えられる。

LAPACK のユーザーズガイドがある (http://www.netlib.org/lapack/lug/)。以前は翻訳 (Anderson 他 [1]) が買えたのだけど…

2 LAPACK の導入

きちんとした Fortran コンパイラーがあれば、とりあえず導入することは容易である。

高い実行効率を実現するには、下請けサブルーチン・ライブラリィである **BLAS** を、利用しているシステム向けの調整 (チューニング) がなされたものに置き換えればよい。

3 LAPACK の dgbsv

LAPACK の特徴は、一口に「連立方程式を解く」、「固有値問題を解く」といった要求に対して、問題に応じてきめ細かい手段が用意されていることにある。

今回は、対称だが、正定値ではない、帯行列を係数とする連立 1 次方程式を解く、というのが目標であることから、**dgbsv** の利用が適当と思われる。

dgbsv は、倍精度実数型データで与えられた (Double precision)、一般の (正値性を持たない General)、帯行列 (Band matrix) を係数行列とする連立 1 次方程式を解くサブルーチンである。

急いでコードが読みたければ

`http://www.netlib.org/cgi-bin/netlibfiles.pl?filename=/lapack/double/dgbsv.f`

にアクセスすると良い[1]。

```
SUBROUTINE DGBSV(N, KL, KU, NRHS, AB, LDAB, IPIV, B, LDB, INFO )
```

- `N` 方程式の (未知数の) 個数
- `KL` 係数行列の帯の中にある subdiagonals (対角線より下の部分) の個数
- `KU` 係数行列の帯の中にある superdiagonals (対角線より上の部分) の個数
- `NRHS` 右辺のベクトルの個数 (いくつの問題を解くか)
- `AB` 係数行列の帯の外を省略して詰め込んだ 2 次元配列
  入力だけ考えると $(\mathrm{KL}+\mathrm{KU}+1)\times\mathrm{N}$ で十分だが、
  ピボッティングありの LU 分解を行うために $(2\mathrm{KL}+\mathrm{KU}+1)\times\mathrm{N}$ 必要となる。
  `AB(LDAB,`何とか`)` と定義されているとする (LDAB $\geq$ N)。
- `LDAB` 2 次元配列の `AB` の最初の寸法 (Leading Dimension of `AB`)
- `IPIV` 行交換の結果を格納する、長さ `N` 以上の整数型配列
- `B` 入力としては連立方程式 $Ax = b_i$ の右辺 b_i を納めた 2 次元配列
  縦ベクトル b_i を並べた行列を 2 次元配列として入力する。
  dgbsv 終了後は対応する解 $x_i = A^{-1}b_i$ が納められている。
  `B(LDB,`何とか`)` と定義されているとする (LDB $\geq$ 2*KL+KU+1)。
- `LDB` 2 次元配列の `B` の最初の寸法 (leading dimension of `B`)
- `INFO` 成功したかどうか、結果をコードで返す
  0 ならば成功、負ならば第 |INFO| 番目の引数がおかしい。正ならば行列が特異であった

このサブルーチンでは、係数行列が帯の外を省略して記憶されていると仮定されている。

[1] LAPACK のソースを探すには `http://www.cs.colorado.edu/~jessup/lapack/applets/search.html` を見ると良い。

3.1 詰め込み方

$1 \le j \le \mathtt{N}$ は当たり前として

$$\mathtt{A}(i,j) = \begin{cases} \mathtt{AB}(K_L + K_U + 1 + i - j, j) & (\max(1, j - K_u) \le i \le \min(N, j + K_L) \text{ のとき}) \\ 0 & (\text{それ以外}). \end{cases}$$

(これは LAPACK のドキュメントにも書いてある。)

この逆に AB を A で表す公式も作っておく。

$$I := \{(p,j); 1 \le j \le N,\ \max(K_L{+}1, K_L{+}K_U{+}2{-}j) \le p \le \min(2K_L{+}K_U{+}1, N{+}K_L{+}K_U{+}1{-}j)\},$$

$$J := \{(p,j); 1 \le p \le 2K_L + K_U + 1,\ 1 \le j \le N\} \setminus I$$

とするとき、

$$\mathtt{AB}(p,j) = \begin{cases} \mathtt{A}(p + j - (K_L + K_U + 1), j) & ((p,j) \in I \text{ のとき}) \\ (\text{ゴミ}) & ((p,j) \in J). \end{cases}$$

3.2 C 言語で添字を 0 から始める場合

$0 \le j \le \mathtt{N} - 1$ は当たり前として

$$\mathtt{A}[i][j] = \begin{cases} \mathtt{AB}[K_L + K_U + i - j][j] & (\max(0, j - K_u) \le i \le \min(N - 1, j + K_L)) \\ 0 & (\text{それ以外}) \end{cases}$$

$$\mathtt{AB}[p][j] = \begin{cases} \mathtt{A}[p + j - (K_L + K_U)][j] & (\max(K_L + K_U - j, K_L) \le p \le \min(2K_L + K_U, N + K_L + K_U - 1 \\ (\text{ゴミ}) & (\text{それ以外}). \end{cases}$$

4 C 言語で書かれたプログラムから LAPACK を呼ぶ

4.1 g77 と gcc のコンビネーション

UNIX 環境では、伝統的に Fortran プログラムと C プログラムが相互にリンク可能なようになっている (非常に便利である)。これは Fortran コンパイラー f77 と C コンパイラー cc の作るオブジェクト・コードと使用する実行時ライブラリィに互換性があるためである。

GCC (GNU compiler collection) でも、この伝統が維持されていて、**g77** (GNU 版 f77) と **gcc** (GNU 版 cc) を利用することで、C プログラムから Fortran プログラム、例えば LAPACK のサブルーチン呼び出しが実現できる。

後で紹介するプログラム `testlapack3.c` では、以下のようにコンパイル・リンクする。

```
oyabun% gcc -c -O -I/usr/local/include testlapack3.c
oyabun% g77 -o testlapack3 testlapack3.o -L/usr/local/lib -llapack -lblas -lmatrix
```

LAPACK のライブラリィ・ファイル `liblapack.a` と、下請けの BLAS のライブラリィ・ファイル `libblas.a` と、お手製の matrix ライブラリィのライブラリィ・ファイル `libmatrix.a` は`/usr/local/lib` というディレクトリィにあるため、

`-L/usr/local/lib -llapack -lblas -lmatrix`

というリンカー・オプションを指定している。

C プログラムのコンパイルには gcc を使っているが、リンクには g77 を使っているところが小さな工夫である (Fortran プログラムは、標準の C ライブラリィ以外のライブラリィを必要とするため、gcc をオプションなしで使ったのでは必要なライブラリィがリンクされない。リンクを g77 に任せることでこの問題を気軽に回避できる)。もっともお手軽には

```
oyabun% g77 -O -I/usr/local/include -o testlapack3 testlapack3.c -L/usr/local/lib -llapack
 -lblas -lmatrix
```

とすべてを g77 に任せることも可能である。

4.2 C プログラムから Fortran のサブルーチンを呼ぶ方法

後のプログラム `testlapack3.c` で呼び出しているところは、

```
dgbsv_(&n, &kl, &ku, &nrhs, AB, &ldab, ipiv, b, &ldb, &info);
```

という一行である。

- 関数の名前が、Fortran サブルーチンの名前 `dgbsv` の末尾にアンダースコア '_' をつけた `dgbsv_` になっている。
- 配列以外の引数には、ポインター演算子であるアンパーサンド '&' を付けている。

(非常に雑な説明をすると、引数の受け渡し法に、C では call by value が、Fortran では call by reference が採用されている、ということ。)

4.3 蛇足: Fortran プログラムから C の関数を呼ぶ方法

Fortran プログラムから、C プログラム中の例えば

```
double sum(double a, double b)
{
  return a + b;
}
```

という関数を呼ぶにはどうしたらよいか?Fortran プログラムで sum と書いても、それはアセンブリー言語に変換されると、sum_ という名前に化けるので、C の関数 sum() のアセンブリー言語レベルでの名前 sum とはマッチしない。

これを解決するには、橋渡しをする C の関数 sum_() を用意すればよい。

```
double sum_(double *a, double *b)
{
  return sum(*a, *b);
}
```

Fortran プログラムから z=sum(x,y) のようにして、この sum_() を呼ぶと、ここから sum() が呼び出されて、間接的に sum() 呼び出しが実現できることになる。

4.4 Fortran の配列と C の配列

4.4.1 配列の添字がどこから始まるか

C 言語のプログラムで double a[5]; と宣言すると、

a[0], a[1], a[2], a[3], a[4]

という 5 つの要素が使えるようになる。

一方、Fortran 言語のプログラムで real*8 a(5) と宣言すると、

a(1), a(2), a(3), a(4), a(5)

という 5 つの要素が使えるようになる。

Fortran 言語のプログラムでは、添字の下限をどこから始めるか指定することが出来る。例えば `real*8 a(0:4)` と宣言すると、C と同様に

`a(0)`, `a(1)`, `a(2)`, `a(3)`, `a(4)`

という 5 つの要素が使えるようになる。添字の下限、上限として任意の整数値が指定できる。

4.4.2 2 次元配列

C 言語のプログラムで、例えば `double a[3][2];` のように 2 次元の配列の宣言をすると、配列の各要素はメモリー中に

`a[0][0]`, `a[0][1]`, `a[1][0]`, `a[1][1]`, `a[2][0]`, `a[2][1]`

の順に並ぶことになる。

一方、Fortran 言語のプログラムで、例えば `real*8 a(3,2)` のように 2 次元の配列の宣言をすると、配列の各要素はメモリー中に

`a(1,1)`, `a(2,1)`, `a(3,1)`, `a(1,2)`, `a(2,2)`, `a(3,2)`

の順に並ぶことになる。

4.4.3 2 次元配列を引数に取る Fortran サブルーチンを C プログラムから呼ぶ方法

プログラムを実行するコンピューターにとっては、メモリー内にデータがどう並んでいるかだけが問題であるので、Fortran 言語で書かれたプログラム (をコンパイルして作られたオブジェクト・コード) が「期待している」順にメモリー上にデータが並ぶように、C プログラムの側で用意してやればよい。

cmain.c

```
/* cmain.c */

#include <stdio.h>

#define M 3
#define N 3

void print_(int *row, int *col, double *A);

int main()
{
  int i,j,k,m,n;
  double a[M][N], A[M*N];
  for (i = 0; i < M; i++)
    for (j = 0; j < N; j++)
      a[i][j] = 10 * (i + 1) + (j + 1);
  k = 0;
  for (j = 0; j < N; j++)
    for (i = 0; i < M; i++)
      A[k++] = a[i][j];
  m = M; n = N;
  print_(&m, &n, A);
  return 0;
}
```

fortransub.f

```
* fortransub.f
      subroutine print(m,n,a)
      integer m,n
      real*8 a(m,n)
      integer i,j
      do i=1,m
         write(*,*) (a(i,j),j=1,n)
      end do
      end
```

コンパイル&リンク&実行

```
oyabun% make candf
gcc -W -Wall -O -I/usr/local/include -c cmain.c
g77 -O -c fortransub.f
g77 -O -o candf cmain.o fortransub.o
oyabun% ./candf
  11.  12.  13.
  21.  22.  23.
  31.  32.  33.
oyabun%
```

5 サンプル・プログラム

5.1 testlapack1.f

```
* testlapack1.f
*
*  格納の仕方の練習をかねて、dgbsv を使ってみる
*    http://www.netlib.org/cgi-bin/netlibfiles.txt?行繋ぐ
*      format=txt&blas=0&filename=lapack%2Fdouble%2Fdgbsv.f
*
*    N     方程式の個数 (=未知数の個数)
*    KL    the number of subdiagonals within the band of 某
*    KU    the number of superdiagonals within the band of 某
*    A     係数行列 (N 次正方行列)
*    AB    A の帯の外を省いて詰め込んで作った行列 (2*KL+KU+1 行, N 列)
*          成分の対応は
*            AB(KL+KU+1+i-j,j)=A(i,j)    max(1,j-KU) ≦ i ≦ min(N,j+KL)
*    LD 某 (LAPACK 用語というより LINPACK 用語) Leading Dimension of 某
*          AB は配列として AB(2*KL+KU+1,N) と定義すれば十分だが
*          それよりも大きな配列として定義することを許すと便利なことがある。
*          AB(LDAB,N) (ただし LDAB ≧ 2*KL+KU+1)
*
      program main
      implicit none
      integer n,ku,kl,nb,ldab,ldb,nrhs
      parameter (n=8,ku=2,kl=2,nb=2*kl+ku+1)
      integer i,j,p,ipiv(n),info
      real*8 a(n,n),ab(nb,n),x(n),b(n),s
      do i=1,n
         do j=1,n
            if ((i .ge. j-ku).and.(i.le.j+kl)) then
               a(i,j)=10*i+j
```

```
            else
               a(i,j)=0
            endif
         end do
      end do
      do i=1,n
         write(*,10) (a(i,j),j=1,n)
      end do
 10   format(' ',8(F5.1))
*
      do i=1,nb
         do j=1,n
            ab(i,j)=0
         end do
      end do
*
      write(*,*)
      write(*,*) ' A から AB を求める公式を用いる'
      do j=1,n
         do p=max(kl+1,kl+ku+2-j),min(nb,n+kl+ku+1-j)
            ab(p,j)=a(p+j-(kl+ku+1),j)
         end do
      end do
      do i=kl+1,nb
         write(*,10) (ab(i,j),j=1,n)
      end do
*
      do i=1,nb
         do j=1,n
            ab(i,j)=0
         end do
      end do
      write(*,*)
      write(*,*) ' AB から A を求める公式を用いる'
      do j=1,n
         do i=max(1,j-ku),min(n,j+kl)
            ab(kl+ku+1+i-j,j)=a(i,j)
         end do
      end do
      do i=kl+1,nb
         write(*,10) (ab(i,j),j=1,n)
      end do
*
      do i=1,n
         x(i)=i
      end do
*
      write(*,*)
      write(*,*) ' 普通に掛け算'
```

```
      do i=1,n
         s=0
         do j=1,n
            s=s+a(i,j)*x(j)
         end do
         b(i)=s
      end do
      write(*,*) (b(i),i=1,n)
*
      write(*,*)
      write(*,*) ' 圧縮してつめた行列で掛け算'
      do i=1,n
         b(i)=0
      end do
      do j=1,n
         do i=max(1,j-ku),min(n,j+kl)
            b(i)=b(i)+ab(kl+ku+1+i-j,j)*x(j)
         end do
      end do
      write(*,*) (b(i),i=1,n)
*
*     NRHS は右辺の個数 (ここでは 1)
*     LDAB は変数 AB の「行の数」 (ここでは =nb=2*kl+ku+1)
*     LDB は変数 B の「行の数」 (ここでは n)
      nrhs=1
      ldab=nb
      ldb=n
      write(*,*)
      write(*,*) ' call dgbsv'
      call dgbsv(n,kl,ku,nrhs,ab,ldab,ipiv,b,ldb,info)
      if (info .eq. 0) then
         write(*,*) '  successful'
      else if (info .gt. 0) then
         write(*,*) ' U is singular'
      else
         write(*,*) ' ', abs(i), '-th argument has illegal value.'
      endif
      write(*,*) ' return from dgbsv'
*
      write(*,*) (b(i),i=1,n)
*
      write(*,*) ' 結果が 1,2,3,.. となっていたら成功'
      end
```

testlapack1 の実行結果

```
oyabun% make

g77 -O -o testlapack1 testlapack1.f -L/usr/local/lib -llapack -lblas
oyabun% ./testlapack1

  11.0 12.0 13.0  0.0  0.0  0.0  0.0  0.0
  21.0 22.0 23.0 24.0  0.0  0.0  0.0  0.0
  31.0 32.0 33.0 34.0 35.0  0.0  0.0  0.0
   0.0 42.0 43.0 44.0 45.0 46.0  0.0  0.0
   0.0  0.0 53.0 54.0 55.0 56.0 57.0  0.0
   0.0  0.0  0.0 64.0 65.0 66.0 67.0 68.0
   0.0  0.0  0.0  0.0 75.0 76.0 77.0 78.0
   0.0  0.0  0.0  0.0  0.0 86.0 87.0 88.0

  A から AB を求める公式を用いる
   0.0  0.0 13.0 24.0 35.0 46.0 57.0 68.0
   0.0 12.0 23.0 34.0 45.0 56.0 67.0 78.0
  11.0 22.0 33.0 44.0 55.0 66.0 77.0 88.0
  21.0 32.0 43.0 54.0 65.0 76.0 87.0  0.0
  31.0 42.0 53.0 64.0 75.0 86.0  0.0  0.0

  AB から A を求める公式を用いる
   0.0  0.0 13.0 24.0 35.0 46.0 57.0 68.0
   0.0 12.0 23.0 34.0 45.0 56.0 67.0 78.0
  11.0 22.0 33.0 44.0 55.0 66.0 77.0 88.0
  21.0 32.0 43.0 54.0 65.0 76.0 87.0  0.0
  31.0 42.0 53.0 64.0 75.0 86.0  0.0  0.0

  普通に掛け算
  74.  230.  505.  890.  1385.  1990.  1994.  1829.

  圧縮してつめた行列で掛け算
  74.  230.  505.  890.  1385.  1990.  1994.  1829.

  call dgbsv
   successful
  return from dgbsv
  1.  2.  3.  4.  5.  6.  7.  8.
  結果が 1,2,3,.. となっていたら成功
oyabun%
```

5.2 testlapack3.c

```
/* testlapack3.c
 *
```

```
 *     格納の仕方の練習をかねて、dgbsv を使ってみる
 *    http://www.netlib.org/cgi-bin/netlibfiles.txt?行繋ぐ
 *      format=txt&blas=0&filename=lapack%2Fdouble%2Fdgbsv.f
 *
 *    N     方程式の個数 (=未知数の個数)
 *    KL    the number of subdiagonals within the band of 某
 *    KU    the number of superdiagonals within the band of 某
 *    A     係数行列 (N 次正方行列)
 *    AB    A の帯の外を省いて詰め込んで作った行列 (2*KL+KU+1 行, N 列)
 *          成分の対応は
 *            AB(KL+KU+1+i-j,j)=A(i,j)    max(1,j-KU)≦ i ≦ min(N,j+KL)
 *    LD 某 (LAPACK 用語というより LINPACK 用語) Leading Dimension of 某
 *          AB は配列として AB(2*KL+KU+1,N) と定義すれば十分だが
 *          それよりも大きな配列として定義することを許すと便利なことがある。
 *          AB(LDAB,N) (ただし LDAB≧ 2*KL+KU+1)
 */

#include <stdio.h>
#include <matrix.h>

#define _AB(i,j) AB[(j)*nb+(kl+ku+(i)-(j))]

void dgbsv_(int *n, int *kl, int *ku, int *nrhs,
            double *AB, int *ldab, int *ipiv, double *b, int *ldb, int *info);

int max(int i, int j) { return (i > j) ? i : j; }
int min(int i, int j) { return (i < j) ? i : j; }

int main()
{
  int n=8, ku=2, kl=2, nb=2*kl+ku+1, ldab, ldb, nrhs, info;
  ivector ipiv;
  int i, j;
  vector x, b;
  vector AB;
  matrix a;

  /* 変数の確保 */
  AB = new_vector(nb * n);
  ipiv = new_ivector(n);
  x = new_vector(n);
  b = new_vector(n);
  a = new_matrix(n,n);

  /* 連立 1 次方程式の問題を作る (A と 解 x を準備して、b:= A x を計算) */
  for (i = 0; i < n; i++)
    x[i] = i + 1;
  for (j = 0; j < n; j++)
    for (i = max(0,j-ku); i <= min(n-1,j+kl); i++)
```

```
    _AB(i,j) = 10 * (i+1) + (j+1);

for (i = 0; i < n; i++)
  b[i] = 0;
for (j = 0; j < n; j++)
  for (i = max(0,j-ku); i <= min(n-1,j+kl); i++)
    b[i] += _AB(i,j) * x[j];

/* A と b を表示 */
printf("A=\n");
for (i = 0; i < n; i++) {
  for (j = 0; j < n; j++)
    if (i-kl <= j && j <= i+ku)
      printf("%4.0f ", _AB(i,j));
    else
      printf("%4.0f ", 0.0);
  printf("\n");
}
printf("b=\n");
for (i = 0; i < n; i++)
  printf("%g ", b[i]);
printf("\n");

/*     NRHS は右辺の個数 (ここでは 1)
 *     LDAB は変数 AB の「行の数」 (ここでは =nb=2*kl+ku+1)
 *     LDB は変数 B の「行の数」 (ここでは n)
 */
nrhs = 1;
ldab = nb;
ldb = n;
printf("\ncall dgbsv\n");
dgbsv_(&n, &kl, &ku, &nrhs, AB, &ldab, ipiv, b, &ldb, &info);
if (info == 0)
  printf(" successful\n");
else if (info > 0)
  printf(" U is singular\n");
else
  printf("%d-th argument has illegal value.\n", abs(info));
printf(" return from dgbsv\n");

/* 計算して得た解 */
printf("\n 計算して得た解=\n");
for (i = 0; i < n; i++)
  printf("%g ", b[i]);
printf("\n");

printf(" 結果が 1,2,3,.. となっていたら成功\n");

return 0;
```

```
}
```

testlapack3 の実行結果

```
oyabun% make testlapack3
gcc -W -Wall -O -I/usr/local/include -c testlapack3.c
g77 -O -o testlapack3 testlapack3.o -L/usr/local/lib -llapack -lblas -lmatrix
oyabun% ./testlapack3

A=
  11   12   13    0    0    0    0    0
  21   22   23   24    0    0    0    0
  31   32   33   34   35    0    0    0
   0   42   43   44   45   46    0    0
   0    0   53   54   55   56   57    0
   0    0    0   64   65   66   67   68
   0    0    0    0   75   76   77   78
   0    0    0    0    0   86   87   88
b=
74 230 505 890 1385 1990 1994 1829

call dgbsv
 successful
 return from dgbsv

計算して得た解=
1 2 3 4 5 6 7 8
 結果が 1,2,3,.. となっていたら成功
oyabun%
```

A Stokes 方程式を解くプログラム

(Stokes 方程式を有限要素法で解くときのプログラムの書き方は、公開していなかったっけ…おおむね川原 [2] に従って書いてあります。)

```
/*
 * stokes3.c --- 定常 Stokes 問題の有限要素解を求めるプログラム
 *  連立 1 次方程式を解くために LAPACK を利用
 *
 *  oyabun でのコンパイル
 *   g77 -I/usr/local/include -O -o stokes3 stokes3.c -llapack -lblas -lmatrix
 */

/* 実行は ./file_name 領域データ 流速の保存ファイル名 圧力の保存ファイル名 */

#include <math.h>
#include <stdio.h>
```

```
#include <matrix.h>

#define USELAPACK
void dgbsv_(int *n, int *kl, int *ku, int *nrhs,
            double *AB, int *ldab, int *ipiv, double *b, int *ldb, int *info);
int max(int a, int b) { return (a > b) ? a : b; }
int min(int a, int b) { return (a < b) ? a : b; }
/* 一次元配列で計算するのは生で書くと大変なのでマクロで処理 */
#define AB(i,j) ab[(j)*nb+(kl+ku+(i)-(j))]

/* 関数定義*/
/* 全体係数マトリックス全体自由項ベクトルを生成する関数 */
void make_matrix(int T, vector ab, int kl, int ku, int nb,
                 vector u, double nu, double rho,
                 int **num, int *cum,
                 vector fx, vector fy, double hx, double hy)
{
  int i,j,k,I,J;
  double hx2,hy2,nurho;

  matrix A = new_matrix(9,9);
  matrix B = new_matrix(9,4);
  matrix C = new_matrix(9,4);
  matrix D = new_matrix(9,9);

  hx2 = hx*hx;  hy2 = hy*hy;
  nurho = nu * rho;

  for (i = 0; i < 9; i++) {
    for (j = 0; j < 4; j++)
      B[i][j] = C[i][j] = 0;
    for (j = 0; j < 9; j++)
      A[i][j] = D[i][j] = 0;
  }
  /* 行列 A の成分の入力*/
  for (i=0; i<=3; i++)
    A[i][i]= 28* (hx2 + hy2);

  A[4][4]=A[6][6]= 112 * hx2 + 64 * hy2;
  A[5][5]=A[7][7]= 64 * hx2 + 112 * hy2;

  for (i=0;i<=3;i++)
    A[i][8]= -16 * ( hx2 + hy2 );

  A[4][5]=A[4][7]=A[5][6]=A[6][7]= -16 * ( hx2 + hy2 );

  A[4][8]=A[6][8]= -128 * hx2 + 32 * hy2;
  A[5][8]=A[7][8]= 32 * hx2 - 128 * hy2;
```

```
A[0][5]=A[1][7]=A[2][7]=A[3][5]= 8 * hx2 + 2 * hy2;
A[0][6]=A[1][6]=A[2][4]=A[3][4]= 2 * hx2 + 8 * hy2;
A[0][4]=A[1][4]=A[2][6]=A[3][6]= 14 * hx2 - 32 * hy2;
A[0][7]=A[1][5]=A[2][5]=A[3][7]= -32 * hx2 + 14 * hy2;
A[0][1]=A[2][3]= -7 * hx2 + 4 * hy2;
A[0][3]=A[1][2]= 4 * hx2 - 7 * hy2;
A[0][2]=A[1][3]= -hx2 - hy2;
A[4][6]= 16 * hx2 - 16 * hy2;
A[5][7]= -16 * hx2 + 16 * hy2;
A[8][8]= 256 * hx2 + 256 * hy2;

for (j=0;j<9;j++)
  for (i=0;i<9;i++)
    A[i][j] = A[j][i];

for (i=0;i<9;i++)
  for (j=0;j<9;j++)
    A[i][j] = (1.0/(90*hx*hy)) * A[i][j];

 /* 行列 B の成分の入力*/
for(i=0;i<9;i++)
  for(j=0;j<4;j++)
    B[i][j]=0;

B[0][0]=B[3][3]=-25; B[1][1]=B[2][2]=25;
B[1][0]=B[2][3]=5;   B[0][1]=B[3][2]=-5;
B[4][0]=B[6][3]=20;  B[4][1]=B[6][2]=-20;
B[5][0]=B[5][3]=10;  B[5][1]=B[5][2]=50;
B[7][0]=B[7][3]=-50; B[7][1]=B[7][2]=-10;
B[8][0]=B[8][3]=40;  B[8][1]=B[8][2]=-40;

for (i=0;i<9;i++)
  for (j=0;j<4;j++)
    B[i][j] = (hy/180.0) * B[i][j];

/* 行列 C の成分の入力*/
for(i=0;i<9;i++)
  for(j=0;j<4;j++)
    C[i][j]=0;

C[0][0]=C[1][1]=-25;
C[2][2]=C[3][3]=25;
C[0][3]=C[1][2]=-5;
C[3][0]=C[2][1]=5;
C[4][0]=C[4][1]=-50;
C[4][2]=C[4][3]=-10;
C[5][1]=C[7][0]=20;
C[5][2]=C[7][3]=-20;
C[6][0]=C[6][1]=10;
```

```
 C[6][2]=C[6][3]=50;
 C[8][0]=C[8][1]=40;
 C[8][2]=C[8][3]=-40;

 for (i=0;i<9;i++)
   for (j=0;j<4;j++)
     C[i][j] = (hx/180.0) * C[i][j];

 /*   行列 D の成分の入力 */
for(i=0;i<=3;i++)
   D[i][i]=16;
 for(i=4;i<=7;i++)
   D[i][i]=64;

 for(i=0;i<=3;i++)
   D[i][8]=4;
 D[4][5]=D[4][7]=D[5][6]=D[6][7]=4;

 for(i=4;i<=7;i++)
   D[i][8]=32;

 D[0][5]=D[0][6]=D[1][6]=D[1][7]=D[2][4]=D[2][7]=D[3][4]=D[3][5]=-2;
 D[0][4]=D[0][7]=D[1][4]=D[1][5]=D[2][5]=D[2][6]=D[3][6]=D[3][7]=8;
 D[0][1]=D[0][3]=D[1][2]=D[2][3]=-4;
 D[0][2]=D[1][3]=1;
 D[4][6]=D[5][7]=-16;
 D[8][8]=256;

 for (j=0;j<9;j++)
   for (i=0;i<9;i++)
     D[i][j]=D[j][i];

for (i=0;i<9;i++)
  for (j=0;j<9;j++)
    D[i][j] = (hx*hy/900.0) * D[i][j];

/* 直接剛性法 */
 for (k=0;k<T;k++){
   for (i=0;i<9;i++){
     I = cum[num[k][i]];
     for (j=0;j<9;j++) {
       J = cum[num[k][j]];
       AB(I,J) += nurho * A[i][j];
       AB(I+1,J+1) += nurho * A[i][j];
     }

     for (j=0;j<4;j++){
       J = cum[num[k][j]];
       AB(I,J+2)   -= B[i][j];
```

```
        AB(I+1,J+2) -= C[i][j];
        AB(J+2,I)   -= B[i][j];
        AB(J+2,I+1) -= C[i][j];
      }
      for (j=0;j<9;j++) {
        u[I] += rho*D[i][j]*fx[num[k][j]];
        u[I+1] += rho*D[i][j]*fy[num[k][j]];
      }
    }
  }
  free_matrix(A);
  free_matrix(B);
  free_matrix(C);
  free_matrix(D);
}

void change(int n, vector ab, int kl, int ku, int nb,
            int k)
{
  int i,j;
  for (i = max(0,k-ku); i <= min(n-1,k+kl); i++)
    AB(i,k) = 0.0;
  for (j = max(0,k-kl); j <= min(n-1,k+ku); j++)
    AB(k,j) = 0.0;
  AB(k,k) = 1.0;
}

void bound(vector ab, int kl, int ku, int nb,
           vector u, int *b, vector bx, vector by,
           int D, int *cum, int n)
{
  int j,i,k;

  for (k = 0; k < D; k++) {
    j = cum[b[k]];
    /* j 列を移項 */
    for (i = max(0,j-ku); i <= min(n-1,j+kl); i++)
      u[i] -= AB(i,j) * bx[k];
    u[j] = bx[k];
    change(n, ab, kl, ku, nb, j);
    /* j+1 列を移項 */
    j++;
    for (i = max(0,j-ku); i <= min(n-1,j+kl); i++)
      u[i] -= AB(i,j) * by[k];
    u[j] = by[k];
    change(n, ab, kl, ku, nb, j);
  }
  change(n, ab, kl, ku, nb, 2);
  u[2] = 0.0;
```

```
}

/* main 文*/
int main(int argc, char **argv)
{
  /*************** 変数宣言**************/
  int i,j;
  /* 入力ファイル、出力ファイル*/
  FILE *f1,*f2,*f3;
  /* nelmt:要素数, nnode:総接点数, num_unknown:未知数の総数, nband:半バンド幅 */
  int nnode,nelmt,num_unknown,nband,cum_tmp,max_cum,min_cum;
  /* nDir: Dirichlet 境界上の節点の総数*/
  int nDir;
  int *cum, **num, *b;
  /* nu:粘性定数, rho:密度, hx,hy:x,y についてのメッシュ幅*/
  double nu,rho,hx,hy;
  /* bx,by:境界値, fx,fy:外力密度 */
  vector x,y,bx,by,fx,fy;
  /* 全体自由項ベクトル*/
  vector u;
  /* 全体係数マトリックス*/
  int n,ku,kl,nb,ldab,ldb,nrhs,info;
  ivector ipvt;
  vector ab;

  if (argc != 4) {
    fprintf(stderr,
            "usage: %s <入力データ> <流速データ> <圧力データ>\n", argv[0]);
    exit(1);
  }
  if ((f1 = fopen(argv[1],"r")) == NULL){
    fprintf(stderr, "Can't open %s",argv[1]);
    exit(1);
  }
  /* 総要素数, 総節点数を読む */
  fscanf(f1, "%d %d %lf %lf", &nelmt, &nnode, &hx, &hy);
  printf("要素数,総節点数,hx,hy=\n%d %d %f %f\n", nelmt, nnode, hx, hy);

  x = new_vector(nnode);
  y = new_vector(nnode);
  if (x == NULL || y == NULL) {
    fprintf(stderr, "節点座標を記憶するメモリが確保できません。");
    exit(1);
  }
  if ((num = malloc(nelmt * sizeof(void *))) == NULL) {
    fprintf(stderr, "要素と節点の対応用のメモリーが足りません\n");
    exit(1);
  }
  for (i=0;i<nelmt;i++) {
```

```
  num[i] = malloc(sizeof(int) * 9);
  if (num[i] == NULL) {
    fprintf(stderr, "要素と節点の対応用のメモリーが足りません\n");
    exit(1);
  }
}
if ((cum = malloc(sizeof(int) * nnode)) == NULL) {
  fprintf(stderr, "累積節点番号用のメモリーが足りません \n");
  exit(1);
}

/* 節点の座標、要素節点番号対応表、累積節点番号表、総節点数の読み込み */
for (i=0;i<nnode;i++)
  fscanf(f1, "%lf %lf", &x[i], &y[i]);
for (i=0;i<nelmt;i++)
  for (j=0;j<9;j++)
    fscanf(f1, "%d", &num[i][j]);
for (i=0;i<nnode;i++)
  fscanf(f1, "%d", &cum[i]); /* i:全体節点番号, cum:累積節点番号 */

fscanf(f1, "%d", &num_unknown);
printf("未知数の個数=%d\n", num_unknown);
if ((u = new_vector(num_unknown)) == NULL) {
  fprintf(stderr, "近似解を記憶するメモリーが不足しています。");
  exit(1);
}

/* 半バンド幅 nband の評価 */
nband=0;
for (i=0;i<nelmt;i++){
  max_cum=0;
  min_cum=num_unknown;
  for (j=0;j<9;j++) {
    cum_tmp=cum[num[i][j]];
    if (max_cum < cum_tmp) max_cum = cum_tmp;
    if (min_cum > cum_tmp) min_cum = cum_tmp;
  }
  if (nband < max_cum - min_cum) nband = max_cum - min_cum;
}
printf("半バンド幅=%d\n", nband);

for (i = 0; i < num_unknown; i++)
  u[i] = 0;

kl = ku = nband + 2;
nb = 2 * kl + ku + 1;
n = num_unknown;
ab = new_vector(nb * n);
ipvt = new_ivector(n);
```

```
if (ab == NULL || ipvt == NULL) {
  fprintf(stderr, "係数行列を記憶するメモリーが足りません\n");
  exit(1);
}
for (i = 0; i < (nb * n); i++)
  ab[i] = 0;

fscanf(f1, "%lf %lf", &nu, &rho);
fscanf(f1, "%d", &nDir);
printf("ν, ρ=\n%f %f\n", nu, rho);
printf("境界上の節点=%d\n", nDir);

if ((b = malloc(sizeof(int) * nDir)) == NULL){
  fprintf(stderr, "境界の番号用のメモリーが足りません。\n");
  exit(0);
}
bx = new_vector(nDir);
by = new_vector(nDir);
if (bx == NULL || by == NULL) {
  fprintf(stderr, "境界値データを読み込むためのメモリーがありません。");
  exit(1);
}
for (i = 0; i < nDir; i++)
  fscanf(f1, "%d %lf %lf", &b[i], &bx[i], &by[i]); /* 境界値 (u,v) */

fx = new_vector(nnode);
fy = new_vector(nnode);
if (fx == NULL || fy == NULL) {
  fprintf(stderr, "外力データを読み込むためのメモリーがありません。");
  exit(1);
}
for (i = 0; i < nnode; i++)
  fscanf(f1, "%lf %lf", &fx[i], &fy[i]); /* 外力の x,y 成分 */

fclose(f1);

/* 全体係数マトリックス、全体自由項ベクトルの生成 */
make_matrix(nelmt,ab,kl,ku,nb,
            u,nu,rho,num,cum,fx,fy,hx,hy);
bound(ab,kl,ku,nb,
      u,b,bx,by,nDir,cum,num_unknown);

free(num);
free_vector(fx);
free_vector(fy);

free(b); /*free(cum);*/
free_vector(bx); free_vector(by);
```

```
  /* 連立一次方程式を解く */
  nrhs = 1;
  ldab = nb;
  ldb = n;
  dgbsv_(&n, &kl, &ku, &nrhs, ab, &ldab, ipvt, u, &ldb, &info);
  if (info == 0)
    printf(" successful\n");
  else if (info > 0)
    printf(" U is singular\n");
  else
    printf("%d-th argument has illegal value.\n", abs(info));
  printf(" return from dgbsv\n");

  /* 計算結果をファイルへ出力 */
  if ((f2 = fopen(argv[2],"w")) == NULL) {
    printf("Can't open %s",argv[2]);
    exit(2);
  }
  if ((f3 = fopen(argv[3],"w")) == NULL) {
    printf("Can't open %s",argv[3]);
    exit(3);
  }
  j=0;
  for(i=0;i<num_unknown-3;){
    fprintf(f2, "%f %f %f %f\n", x[j], y[j], u[i], u[i+1]);
    if(cum[j+1]-cum[j]==3){
      fprintf(f3, "%f %f %f\n", x[j], y[j], u[i+2]);
      i++;
      if(x[j+1]!=x[j]){
        fprintf(f3,"\n");
      }
    }
    i=i+2;
    j++;
  }
  fprintf(f2, "%f %f %f %f\n", x[nnode-1], y[nnode-1], u[num_unknown-3], u[num_unknown-2]);
  fprintf(f3, "%f %f %f\n", x[nnode-1], y[nnode-1], u[num_unknown-1]);

  fclose(f2);
  fclose(f3);

  return 0;
}
```

参考文献

[1] Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Croz, J. D., Greenbaum, A., Hammarling, S., McKenney, A. and Sorensen, D.: *LAPACK User's Guide*, SIAM, Philadelphia (1992).

[2] 川原睦人：有限要素流体解析, 日科技連 (1985).