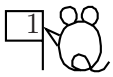
数値計算講義　第6 回
行列の計算 (1) – ガウスの消去法 –
2 次元配列の取り扱い

金子　晃（カーネンコ　アレクセイ）

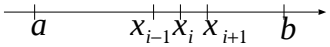
kanenko@mbk.nifty.com
alexei.kanenko@docomo.ne.jp
http://www.kanenko.com/

巨大行列の計算は数値計算の大部分を占める



巨大行列は微分方程式の離散化により生ずる
例: 2 階線型常微分方程式の境界値問題

$$\begin{cases} -\dfrac{d^2u}{dx^2} + q(x)u = f \quad \text{on } [a,b], \\ u(a) = u(b) = 0 \end{cases}$$



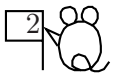
区間 $[a,b]$ を N 等分し，$h = \dfrac{b-a}{N}$, $x_i = a + hi$, $i = 0,1,2,\ldots,N$ と置く．
分点での値に $u_i = u(x_i)$, $q_i = q(x_i)$, $f_i = f(x_i)$ という記号を用いる．
各点 x_i における2 階微分を2 階の中心差分で置き換えると，

$$\begin{cases} -\dfrac{u_{i-1} + u_{i+1} - 2u_i}{h^2} + q_i u_i = f_i, \quad i = 1,2,\ldots,N-1, \\ u_0 = u_N = 0 \end{cases}$$

これは未知数 u_i に関する次のような連立一次方程式となる．

$$\begin{pmatrix} \frac{2}{h^2}+q_1 & -\frac{1}{h^2} & 0 & \cdots & 0 \\ -\frac{1}{h^2} & \frac{2}{h^2}+q_2 & -\frac{1}{h^2} & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & -\frac{1}{h^2} & \frac{2}{h^2}+q_{N-2} & -\frac{1}{h^2} \\ 0 & \cdots & 0 & -\frac{1}{h^2} & \frac{2}{h^2}+q_{N-1} \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_{N-2} \\ u_{N-1} \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_{N-2} \\ f_{N-1} \end{pmatrix}$$

行列のプログラミング



ベクトルは1 次元配列である．これは簡単.
行列は2 次元配列である．
FORTRAN では

```
      DOUBLE PRECISION A(5,5), X(5), Y(5)
```

これは次と同じ

```
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION A(5,5), X(5), Y(5)
```

C では

```
    double a[5][5], x[5], y[5];
```

行列とベクトルの掛け算 Y=AX は

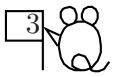
```
      DO 200  I=1,5              ! 外側のループで各成分につき計算
          Y(I)=0                 ! 内側のループは級数の和の要領で
          DO 100  J=1,5
              Y(I)=Y(I)+A(I,J)*X(J)
  100     CONTINUE
  200 CONTINUE
```

- FORTRAN では X(5) と宣言すると X(1) ∼ X(5) が使えるが,
  C では x[5] と宣言したときに使えるのは x[0] ∼ x[4] となる．

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix}$$

行列のメモリーイメージ

(実験プログラム num6-1.f, num6-1.c)



2 次元配列と言えどもメモリーの中ではリニアに並んでいる.
しかし，FORTRAN と C では並び方に重大な差が生ずる.

FORTRAN で

```
      DOUBLE PRECISION A(3,5)
```

と宣言したときは

```
A(1,1) A(2,1) A(3,1) A(1,2) A(2,2) A(3,2) A(1,3) A(2,3) ...
```

C で

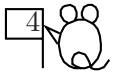
```
    double A[3][5];
```

と宣言したときは

```
A[0][0] A[0][1] ... A[0][4] A[1][0] ... A[1][4] A[2][0] ...
```

並び方の違いは普通は気にしなくてよいが，アクセススピードに関係する.
(本日の課題 6-1 参照.)
また, 大きめに取った2 次元配列の一部だけを使うときは特に注意が必要.

配列の扱いに関する注意 – 1



- 添え字の範囲: C では 0 からサイズ −1 までの範囲となる．
FORTRAN ではデフォールトで 1 からサイズまでの範囲となるが，

```
A(-1:1,0:4)
```

と宣言すれば，同じ寸法で添字の範囲を $-1 \leq i \leq 1, 0 \leq j \leq 4$ にずらせる．

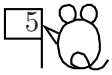
- 初期化: C では全要素に 0 がセットされるが，
FORTRAN は必ず自分でクリアする必要がある．

- 配列名を引数にすると，C 言語でも参照渡しとなり，
サブルーチンで引数の行列の基本変形などをすると，
呼び出したプログラムでその行列の内容が壊される．必要ならバックアップを．

- メインとサブで配列の寸法が合っていないと悲劇が起こる．

`main` で
`INTEGER A(3,4)` を宣言し，`0` に初期化した後
`CALL SUB(A)` をする
`sub` で
`INTEGER A(2,2)` を宣言し
`A(1,1)=11; A(2,1)=21; A(1,2)=12; A(2,2)=22` とセット
したとき，メインに戻ったときの `A` の中身はどうなっているか？

2 次元配列を引数としたときの注意　num6-2.f, num6-2.c




```
A(1,1)=11  A(1,2)=12  A(1,3)=0  A(1,4)=0
A(2,1)=21  A(2,2)=22  A(2,3)=0  A(2,4)=0
A(3,1)=0   A(3,2)=0   A(3,3)=0  A(3,4)=0
```

となってくれることが期待されるが，実際にやってみると (num6-2.f)

```
A(1,1)=11  A(1,2)=22  A(1,3)=0  A(1,4)=0
A(2,1)=21  A(2,2)=0   A(2,3)=0  A(2,4)=0
A(3,1)=12  A(3,2)=0   A(3,3)=0  A(3,4)=0
```

となる．何故か?

2 次元配列と言えども，メモリーには1 次元的に並んでいること，
サブルーチンへの参照渡しは先頭のアドレスだけであること．
の2 点から，こうなることが理解できる：

メインでの並び方:

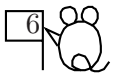
```
A(1,1) A(2,1) A(3,1) A(1,2) A(2,2) A(3,2) A(1,3) ...
```

サブでの並び方:

```
A(1,1) A(2,1) A(1,2) A(2,2)
```

同じことを C でやったらどうなるか? もう分かるね．(num6-2.c)

連立一次方程式を解く 消去法



ではいよいよ，連立一次方程式を解いてみよう．連立一次方程式

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1, \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2, \\ \qquad\qquad \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n \end{cases}$$

は，行列表現で

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \tag{1}$$

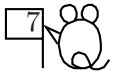
となる．どちらでも同じなので，以下後者でやる．要は，行基本変形で

$$\begin{pmatrix} a'_{11} & a'_{12} & \cdots & a'_{1n} \\ 0 & a'_{22} & & a'_{2n} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & a'_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b'_1 \\ b'_2 \\ \vdots \\ b'_n \end{pmatrix} \tag{2}$$

の形 (上三角型) に帰着し，下の方から順に求めて行く．

- (1) を (2) に帰着させる計算を **前進消去**と呼ぶ．
- (2) から x_n,...,x_2, x_1 を求める計算を **後退代入**と呼ぶ．

前進消去の際の注意



ピボット (枢軸) 選択

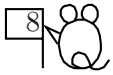
純粋数学なら，(1) で $a_{11} \neq 0$ のとき，各 $i = 2, \ldots, n$ に対し
(1) の第1 行の $\dfrac{a_{i1}}{a_{11}}$ 倍を第 i 行から引けば，第1 列を 0 にできる：

$$\left(\begin{array}{cccc|c} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} & b_n \end{array}\right) \Longrightarrow \left(\begin{array}{cccc|c} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ 0 & a'_{22} & \cdots & a'_{2n} & b'_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & a'_{n2} & \cdots & a'_{nn} & b'_n \end{array}\right)$$

しかし，数値計算では，たとい $a_{11} \neq 0$ であっても，
a_{i1}, $i = 1, \ldots, n$ の中から絶対値最大のものを探し出し，
その行を第1 行と入れ換えてから，上の計算をする．

その理由は，**桁落ち**の起こる可能性をなるべく排除するため．

例：次のような単精度の実行列は，そのまま行基本変形すると



$$\left(\begin{array}{ccc|c} 1.000000 & 1.000000 & -2.000000 & 1.000000 \\ 1.000001 & 1.000000 & -1.000000 & 1.000000 \\ 2.000000 & 1.000000 & -1.000000 & 0.000000 \end{array}\right)$$

$$\Longrightarrow \quad \left(\begin{array}{ccc|c} 1.000000 & 1.000000 & -2.000000 & 1.000000 \\ 0 & -0.000001 & 1.000002 & -0.000001 \\ 0 & -1.000000 & 3.000000 & -2.000000 \end{array}\right)$$

2 行目で桁落ちが起きているので，以後の精度が1 桁になってしまう．

行の入れ換えをしてからやると

$$\left(\begin{array}{ccc|c} 2.000000 & 1.000000 & -1.000000 & 0.000000 \\ 1.000001 & 1.000000 & -1.000000 & 1.000000 \\ 1.000000 & 1.000000 & -2.000000 & 1.000000 \end{array}\right)$$

$$\Longrightarrow \quad \left(\begin{array}{ccc|c} 2.000000 & 1.000000 & -1.000000 & 0.000000 \\ 0 & 0.4999995 & -0.4999995 & 1.000000 \\ 0 & 0.500000 & -1.500000 & 1.000000 \end{array}\right)$$

$$\Longrightarrow \quad \left(\begin{array}{ccc|c} 2.000000 & 1.000000 & -1.000000 & 0.000000 \\ 0 & 0.500000 & -1.500000 & 1.000000 \\ 0 & 0.4999995 & -0.4999995 & 1.000000 \end{array}\right)$$

$$\Longrightarrow \quad \left(\begin{array}{ccc|c} 2.000000 & 1.000000 & -1.000000 & 0.000000 \\ 0 & 0.500000 & -1.500000 & 1.000000 \\ 0 & 0 & 0.9999990 & 0.000001 \end{array}\right)$$

相変わらず x_3 の値で桁落ちが生じており，有効数字1 桁になるが，
今度はそれが x_2 や x_1 の有効桁数に影響しないので，
固定小数点の立場から見て3 変数一組での精度が保証されている．
(例えば，ベクトルの長さなどは精度が落ちないで計算できる．)

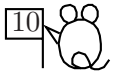
消去法のプログラムの書き方 num6-3.f


```
      SUBROUTINE SOLVE(A,B,X,N)
      DOUBLE PRECISION A(N,N),B(N),X(N),FACTOR
C 前進消去
      DO 300 I=1,N
        IMAX=I
        DO 50 K=I+1,N                              ! ピボット 選択
          IF (DABS(A(IMAX,I)).LT.DABS(A(K,I))) IMAX=K
   50   CONTINUE
        IF (IMAX.GT.I) CALL SWAP(A,B,I,IMAX,N)  ! 行交換
        DO 200 J=I+1,N
          FACTOR=A(J,I)/A(I,I)
          DO 100 K=I+1,N
            A(J,K)=A(J,K)-A(I,K)*FACTOR
  100     CONTINUE
          B(J)=B(J)-B(I)*FACTOR
  200   CONTINUE                 ! A(J,I)=0, J>I はやる必要無し
  300 CONTINUE
C 後退代入
      DO 600 I=N,1,-1
        X(I)=B(I)
        DO 500 J=I+1,N
          X(I)=X(I)-X(J)*A(I,J)
  500   CONTINUE
        X(I)=X(I)/A(I,I)
  600 CONTINUE
      RETURN
      END
```

$$\left(\begin{array}{cccc|c} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} & b_n \end{array}\right) \Longrightarrow \left(\begin{array}{cccc|c} a'_{11} & a'_{12} & \cdots & a'_{1n} & b'_1 \\ 0 & a'_{22} & \cdots & a'_{2n} & b'_2 \\ \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & a'_{nn} & b'_n \end{array}\right)$$

ピボットに選んだ行を先頭成分で割り算してから
他の行を消去してもよい.
この場合は上三角行列の対角成分がすべて1 に帰着される.
同時に $\boldsymbol{b}$ の成分も割るので，割り算の回数が減るわけではない.

計算量の見積もり



簡単のため, 行の入れ換えは無いものとする.

- 前進消去は $O(N^3)$:
  - 比較 $(N-1)+(N-2)+\cdots+1=\dfrac{N(N-1)}{2}$
  - 除算 $(N-1)+(N-2)+\cdots+1=\dfrac{N(N-1)}{2}$
  - 乗算と差 $\{(N-1)^2+(N-2)^2+\cdots+1\}+\{(N-1)+(N-2)+\cdots+1\}$
    $=\dfrac{N(N-1)(2N-1)}{6}+\dfrac{N(N-1)}{2}$
- 後退代入は $O(N^2)$:
  - 乗算と差 $(N-1)+(N-2)+\cdots+1=\dfrac{N(N-1)}{2}$
  - 除算 N

つまり, 前進消去の方が律速 (速度に対する影響を支配する) であり,
全体で $O(N^3)$ の計算量となっている.

よくある勘違い:
同じ方法で $O(N^3)$ で A の逆行列 C が求まる. 方程式の解は
$X=CB$ で計算すればよいではないか?

行列と ベクト ルの積の計算量は
- 乗算 $N \times N = N^2$, 和 $N \times (N-1)$ で $O(N^2)$ である.

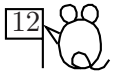
```
      DO 100 I=1,N
          Y(I)=0
          DO 100 J=1,N
             Y(I)=Y(I)+A(I,J)*X(J)
  100 CONTINUE
```

しかし, 実用でよく 現れる 行列は, 非零成分が主対角線の近く だけに
存在する **帯状行列**であることが多い. i.e. $A(I,J) = 0$ for $|I-J| > b$
b は**バンド 幅**と 呼ばれ, N に 依存し ない定数 (普通, 空間の次元に 依存).
この場合, 前進消去・ 後退代入ともに 計算量は $O(N)$ に 減少するが,
逆行列は帯状行列にならない (密行列 full matrix).
こういう 場合は, 逆行列を 掛ける 方が計算量が大きく なる.

$$\left(\begin{array}{ccccc|c} a_{11} & a_{12} & 0 & \cdots & 0 & b_1 \\ a_{21} & a_{22} & a_{23} & \ddots & \vdots & b_2 \\ 0 & \ddots & \ddots & \ddots & 0 & \vdots \\ \vdots & \ddots & \ddots & \ddots & a_{n-1,n} & \vdots \\ 0 & \cdots & 0 & a_{n,n-1} & a_{nn} & b_n \end{array}\right) \Longrightarrow \left(\begin{array}{ccccc|c} a_{11} & a_{12} & 0 & \cdots & 0 & b_1 \\ 0 & a'_{22} & a_{23} & \ddots & \vdots & b'_2 \\ 0 & \ddots & \ddots & \ddots & 0 & \vdots \\ \vdots & \ddots & \ddots & \ddots & a_{n-1,n} & \vdots \\ 0 & \cdots & 0 & 0 & a'_{nn} & b'_n \end{array}\right)$$

帯状行列は正方配列ではなく 長方形の配列 `A(N,-B:B)` に格納する.
B はバンド 幅で, 対称行列なら, `A(N,0:B)` でよい.

LR 分解 (LU 分解)



前進消去は行列に左から下三角型行列を掛けることで達成される．
$LA = R$ は上三角型となるので， $A = L^{-1}R$, 従って $A^{-1} = R^{-1}L$

- 前進消去は右辺のベクトル $\boldsymbol{b}$ に L を掛ける操作に相当，
- 後退代入はその結果 $L\boldsymbol{b}$ に R^{-1} を掛ける操作に相当．

A が帯状行列なら，L, R は帯状の下，上三角型行列となるので，
この場合は A^{-1} でなく，L, R を記憶して使えば，
$O(N)$ で方程式が解ける．

- 三角型帯状行列も逆行列をとると，同じ三角型の密行列となるので，
R^{-1} を記憶してしまうと，積の計算量は $O(N^2)$ になってしまう．
R を記憶し，後退代入で $R^{-1}\boldsymbol{b}$ を計算しなければならない．

○	上三角型行列の逆行列は，上三角型
○	下三角型行列の逆行列は，下三角型
○	対角型行列の逆行列は，対角型
×	帯状行列の逆行列は，帯状行列
×	上三角型帯状行列の逆行列は，上三角型帯状行列
×	下三角型帯状行列の逆行列は，下三角型帯状行列

配列のサイズを合わせる工夫 –1

FORTRAN の場合: 整合配列が使える.

```
SUBROUTINE SUB(A,B,N)
DOUBLE PRECISION A(N,N), B(N)
......
```

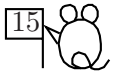
サブルーチンの引数になっている配列に限り,
変数の寸法パラメータ N で宣言することができる (**整合配列**という).
あるいは, サイズを良い加減にして書く. A(1,1) など (**偽寸法配列**という).
(後者はコンパイラによっては通らないことも有る.)

特定のサブルーチン内だけで使用する作業領域用配列のサイズは,
定数で宣言しなければならない.
この制約に対する実用的対処法:

- LAPACK や大型計算機のライブラリでは, 作業領域用配列も
メインで宣言し, 各サブルーチンに引数で渡すことで,
全体でのメモリー節約をはかっている.
- 共通仕様の PARAMETER 文で全体のサイズを統一することもできる.
しかし, C のように大域変数・定数の定義も無いので,
各サブルーチンの中で宣言が必要となり, 直す箇所が検索しやすい
という程度の効果しか無い.
C のように `#include` 宣言が無いので, 分割コンパイルの場合は更に面倒.
(⟶ FORTRAN 90 では改良された.)

配列のサイズを合わせる工夫 –2

- **C の場合:**
  - 配列のサイズを大域定数にする:
    ```
    const int N=1000; または
    #define N 1000
    ```
    分割コンパイルする場合は，これをヘッダファイル hoge.h に書き，
    各ファイルで #include "hoge.h" によりインクルードする.
  - 1 次元配列を自分で2 次元に使う.
    ```
    double A[1000000];
    ```
    としておき，
    ```
    void sub(double *A, int M, int N){
        int i,j;
        A(M*i+j);    /*  A[i][j] をアクセス */
    }
    ```
    この場合は，malloc と組み合わせ，実際に必要なサイズが分かった段階で
    1 次元の配列を確保すると更に効果的:
    ```
    double *B;
    B=(double *)malloc(sizeof(double)*M*N);
    ....
    free(B)          /* 使い終わった作業領域のメモリーを開放 */
    ```
- 使い終わった作業領域は開放しないとメモリーリークが起こる.



最初に挙げた Sturm-Liouville 型2 階常微分方程式の
斉次 Dirichlet 境界値問題

$$-u'' + q(x)u = f(x), \qquad u(0) = u(1) = 0$$

の数値解を求めてみる．
用意する函数：$q(x)$, $f(x)$

- プログラムはなるべく汎用的に書く．
- 動作を解が解析的に求まる例で確認する．

例：$q(x) = 0$, $f = 1$ のとき，解析的な解は $u = \dfrac{1}{2}x(1-x)$

計算結果は，標準出力に出る．これをリダイレクトによりファイルに落し，
gnuplot を用いてグラフに描いてみるのが，標準的手順：

```
$ g77 num6-4.f -o num6-4
$ ./num6-4 > hoge.dat      % この > はリダイレクト
$ gnuplot                  % ここから先の > は gnuplot のプロンプト
$ gnuplot > plot "hoge.dat" with linespoints
$ gnuplot > quit
```

num6-5.f は Kerosoft 社のグラフィックライブラリで直接描画している．

計算結果のファイルへの書き込み方 (続き)

- リダイレクトで，出力がコンソールにも出るようにするには,
  ```
  ./num6-4 | tee hoge.dat
  ```
- コンソールへの出力を同時にファイルに書き込むように
プログラムを作る.
(引数が与えられたときに限り，そのファイルを開くようにもできる．)
- C でのファイルの開き方の復習:

```
FILE *file;
file = fopen("hoge.dat","w");
...
fprintf(file,"%22.15lf ",X[i]);
...
fclose(file);
```

- FORTRAN でのファイルの開き方:

```
      OPEN(2,FILE='hoge.dat',ACCESS='SEQUENTIAL')
      ....
      WRITE(2,200) (X(I),I=1,N)
      ...
      CLOSE(2)
  200 FORMAT(1H ,4F18.15)
```

おまけ：行列式と逆行列の計算　num6-6.f

決定形の連立一次方程式のプログラムを少し変更すればできるので，
ついでに作っておくと後で何かと便利である．

行列式：上三角型にして，全ての主対角成分の積を返す函数とすればよい．
　ピボット選択で行交換したら，符号を変えるのを忘れないように (^^;

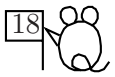
逆行列：$B(N)$ を $B(N,N)$ に変え，行基本変形を B の全ての行に
　A と同時に適用する．
　この場合は，A が上三角型になった後で，後退代入の代わりに，
　A の対角線から上の成分を消去する操作で，B を行変形し，
　最後に B の各行を A の対角成分で割る．

$$\left(\begin{array}{cccc|cccc} a_{11} & a_{12} & \cdots & a_{1n} & 1 & 0 & \cdots & 0 \\ a_{21} & a_{22} & \cdots & a_{2n} & 0 & 1 & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \ddots & 0 \\ a_{n1} & a_{n2} & \cdots & a_{nn} & 0 & \cdots & 0 & 1 \end{array}\right)$$

$$\Longrightarrow \left(\begin{array}{cccc|cccc} a'_{11} & a'_{12} & \cdots & a'_{1n} & b'_{11} & b'_{12} & \cdots & b'_{1n} \\ 0 & a'_{22} & \ddots & \vdots & b'_{21} & b'_{22} & \cdots & b'_{2n} \\ \vdots & \ddots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & a'_{nn} & b'_{n1} & b'_{n2} & \cdots & b'_{nn} \end{array}\right)$$

$$\Longrightarrow \left(\begin{array}{cccc|cccc} 1 & 0 & \cdots & 0 & b_{11} & b_{12} & \cdots & b_{1n} \\ 0 & 1 & \ddots & \vdots & b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \ddots & \ddots & 0 & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & 1 & b_{n1} & b_{n2} & \cdots & b_{nn} \end{array}\right)$$

本日の講義内容の自習課題



1 プログラム見本 `num6-1.c`, `num6-1.f` をコンパイルし，実行してみて
メモリ内部での2 次元配列データの並び方を確認する．

2 プログラム見本 `num6-2.c`, `num6-2.f` をコンパイルし，実行してみて
サイズの異なる2 次元配列データの受渡しが起こす現象を確認する．

3 `num6-4.f` と gnuplot を使って
常微分方程式の境界値問題の数値解を味わう．
`num6-5.f` も実行してみる．

`num6-5.f` は Kerosoft 社のグラフィックライブラリをリンクする．
コンパイルは第1 回でやった通り：

```
gcc -c xgrf.c      (xgrf.o を作る)
g77 num6-5.f xgrf.o -lX11 -L/usr/X11R6/lib -o num6-5
```

Cygwin のターミナルの場合, コンパイルの仕方は,
mizuka/F77 の下にある xgrfw.o を自分の作業ディレクトリにコピーし,

```
g77 num6-5.f xgrfw.o -user32 -lgdi32 -o num6-5
```

により, できた実行可能ファイル num6-5.exe を実行する．

本日の範囲の試験予想問題

問題 6.1 連立1 次方程式をガウスの消去法で解くとき，ピボット 選択を行うのはなぜか？キーワード「 桁落ち」を用いて説明せよ．

問題 6.2 ベクトル $\begin{pmatrix} x \\ y \end{pmatrix}$ に行列 $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ を N 回反復して施した結果を求めるプログラムを次のように書いた．問題点を指摘し訂正せよ．

```
for (i=0;i<N;i++){
  x=a*x+b*y;
  y=c*x+d*y;
}
```

課題 6.3 2 次元配列をコピーするプログラムを C 言語により2 重ループを用いて次のように書いた．

```
for (i=0;i<N;i++){
  for (j=0;j<N;j++){
    b[i][j]=a[i][j];
  }
}
```

(1) この書き方は正しいか？ それとも添え字の順を交換した方が速くなるか?
(2) これに対応するプログラムを FORTRAN 77 で書け．ただし高速になる順番にループを選択せよ．

問題 6.4 N 次正方行列を計数行列とする連立1 次方程式をガウスの消去法で解くとき，必要とされる計算量を乗除算の回数で見積もれ．また，巨大なサイズの連立1 次方程式の近似解を求めるのに使われる，ガウスの消去法よりも効率的な方法を述べよ．(後半は次回のテーマです．)