計算機実験 1

高須担当 : 2003. 12/12, 12/19, 2004. 1/9, 1/23 の 4 回

連立一次方程式の数値解法	微分方程式の数値解法
• Gaussの消去法	• 時間の差分化
• ピボット選択	• Euler 法
• 逆行列の計算	• Runge Kutta 法
• 行列式の計算	• 数値解の視覚化

上記の課題を C 言語を用いてプログラムする。

各問題を解いてレポートを作成し、ソースプログラムとともに提出。
提出先は、高須 @ G311、提出期限は 1) 12月 31 日、2) 1 月 29 日

Gauss の消去法

n 次元連立一次方程式を考える。

$$\begin{aligned}
&a_{00}x_0 + a_{01}x_1 + a_{02}x_2 + \ldots + a_{0n-1}x_{n-1} = b_0\\
&a_{10}x_0 + a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n-1}x_{n-1} = b_1\\
&a_{20}x_0 + a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n-1}x_{n-1} = b_2\\
&\vdots\\
&a_{n-10}x_0 + a_{n-11}x_1 + a_{n-12}x_2 + \ldots + a_{n-1n-1}x_{n-1} = b_{n-1}
\end{aligned}$$

Ax = b

係数 a_{ij} および b_i は C 言語の配列として表現する。
本講義では、添え字 i, j の範囲は 0 ... n–1 で考える。

消去法のアルゴリズム

$$\begin{aligned}
&a_{00}x_0 + a_{01}x_1 + a_{02}x_2 + \ldots + a_{0n-1}x_{n-1} = b_0 && (0)\\
&a_{10}x_0 + a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n-1}x_{n-1} = b_1 && (1)\\
&a_{20}x_0 + a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n-1}x_{n-1} = b_2 && (2)\\
&\vdots\\
&a_{n-10}x_0 + a_{n-11}x_1 + a_{n-12}x_2 + \ldots + a_{n-1n-1}x_{n-1} = b_{n-1} && (n\text{–}1)
\end{aligned}$$

第 0 段

a_{00} がゼロでないとき、a_{00} を軸（ピボット）にして x_0 の係数を消去

式 (1) に、式 (0) * ($-a_{10}/a_{00}$) を足すと、式 (1) の x_0 の係数はゼロになる。

式 (2) に、式 (0) * ($-a_{20}/a_{00}$) を足すと、式 (2) の x_0 の係数はゼロになる。

...

式 (n–1) に、式 (0) * ($-a_{n-10}/a_{00}$) を足すと、式 (n–1) の x_0 の係数はゼロ。

アルゴリズム続き

$$\begin{aligned}
&a_{00}x_0 + a_{01}x_1 + a_{02}x_2 + \ldots + a_{0n-1}x_{n-1} = b_0 && (0)\\
&a'_{11}x_1 + a'_{12}x_2 + \ldots + a'_{1n-1}x_{n-1} = b'_1 && (1)\\
&a'_{21}x_1 + a'_{22}x_2 + \ldots + a'_{2n-1}x_{n-1} = b'_2 && (2)\\
&a'_{31}x_1 + a'_{32}x_2 + \ldots + a'_{3n-1}x_{n-1} = b'_3 && (3)\\
&\vdots\\
&a'_{n-1}x_1 + a'_{n-12}x_2 + \ldots + a'_{n-1n-1}x_{n-1} = b'_{n-1} && (n\text{–}1)
\end{aligned}$$

第 1 段

a'_{11} がゼロでないとき　　a'_{11} を軸（ピボット）にして x_1 の係数を消去

式 (2) に、式 (1) * ($-a'_{21}/a'_{11}$) を足すと、式 (2) の x_1 の係数はゼロになる。

式 (3) に、式 (1) * ($-a'_{31}/a'_{11}$) を足すと、式 (3) の x_1 の係数はゼロになる。

...

式 (n–1) に、式 (1) * ($-a'_{n-11}/a'_{11}$) を足すと、式 (n–1) の x_1 の係数はゼロ。

続き

$$
\begin{aligned}
a_{00}x_0 + a_{01}x_1 + a_{02}x_2 + \ldots \quad & + a_{0n-1}x_{n-1} = b_0 && (0)\\
a'_{11}x_1 + a'_{12}x_2 + \ldots \quad & + a'_{1n-1}x_{n-1} = b'_1 && (1)\\
a''_{22}x_2 + \ldots \quad & + a''_{2n-1}x_{n-1} = b''_2 && (2)\\
a''_{32}x_2 + \ldots \quad & + a''_{3n-1}x_{n-1} = b''_3 && (3)\\
a''_{n-12}x_2 + \ldots \quad & + a''_{n-1n-1}x_{n-1} = b''_{n-1} && (n-1)
\end{aligned}
$$

第 2 段

a''_{22} がゼロでないとき、a''_{22} を軸（ピボット）にして x_2 の係数を消去

式 (3) に、式 (2) * $(-a''_{32}/a''_{22})$ を足すと、式 (3) の x_2 の係数はゼロ。

式 (4) に、式 (2) * $(-a''_{42}/a''_{22})$ を足すと、式 (4) の x_2 の係数はゼロ。

…

式 $(n-1)$ に、式 (2) * $(-a''_{n-12}/a''_{22})$ を足すと、式 $(n-1)$ の x_2 の係数はゼロ。

5

前進消去過程

先程の操作を繰り返すと最終的に以下の形に変形できる。

$$
\begin{aligned}
a_{00}x_0 + a_{01}x_1 + a_{02}x_2 + \ldots \quad & + a_{0n-1}x_{n-1} = b_0 && (0)\\
a_{11}x_1 + a_{12}x_2 + \ldots \quad & + a_{1n-1}x_{n-1} = b_1 && (1)\\
a_{22}x_2 + \ldots \quad & + a_{2n-1}x_{n-1} = b_2 && (2)\\
a_{n-2n-2}x_{n-2} + a_{n-2n-1}x_{n-1} &= b_{n-2} && (n-2)\\
a_{n-1n-1}x_{n-1} &= b_{n-1} && (n-1)
\end{aligned}
$$

この形式を上三角方程式と呼ぶ。以上の過程を前進消去過程と呼ぶ。

第 1 行以下の係数 a_{ij}, b_i は、オリジナルの係数とは異なることに注意。

途中でゼロになる軸があると失敗する（ゼロでの割算はできないため）。
対処法は後述。

6

後退代入過程

連立一次方程式を前進消去過程により上三角方程式に変形できれば、
解は後退代入を繰り返すことで容易に計算できる。

$$
\begin{aligned}
a_{00}x_0 + a_{01}x_1 + a_{02}x_2 + \ldots \quad & + a_{0n-1}x_{n-1} = b_0 && (0)\\
a_{11}x_1 + a_{12}x_2 + \ldots \quad & + a_{1n-1}x_{n-1} = b_1 && (1)\\
a_{22}x_2 + \ldots \quad & + a_{2n-1}x_{n-1} = b_2 && (2)\\
a_{n-2n-2}x_{n-2} + a_{n-2n-1}x_{n-1} &= b_{n-2} && (n-2)\\
a_{n-1n-1}x_{n-1} &= b_{n-1} && (n-1)
\end{aligned}
$$

式 $(n-1)$より　$x_{n-1} = b_{n-1}/a_{n-1n-1}$

式 $(n-2)$より　$x_{n-2} = (b_{n-2} - a_{n-2n-1}x_{n-1})/a_{n-2n-2}$

式 $(n-3)$より　$x_{n-3} = (b_{n-3} - a_{n-3n-1}x_{n-1} - a_{n-3n-2}x_{n-2})/a_{n-3n-3}$

一般に　$x_i = (b_i - \sum_{j=i+1}^{n-1} a_{ij}x_j)/a_{ii} \quad (i = n-1, n-2, n-3, \ldots, 1)$

7

具体例

$$
\begin{aligned}
2x_0 - x_1 + x_2 &= 0 && (0)\\
-x_0 + 2x_1 - x_2 &= 1 && (1)\\
2x_0 - 2x_1 - x_2 &= -3 && (2)
\end{aligned}
\qquad
A = \begin{bmatrix} 2 & -1 & 1 \\ -1 & 2 & -1 \\ 2 & -2 & -1 \end{bmatrix},\quad B = \begin{bmatrix} 0 \\ 1 \\ 3 \end{bmatrix}
$$

第 0 段

$$
\begin{aligned}
2x_0 - x_1 + x_2 &= 0 && (0)\\
(1) + (0)\times\frac{1}{2} \rightarrow \frac{3}{2}x_1 - \frac{1}{2}x_2 &= 1 && (1)\\
(2) + (0)\times(-1) \rightarrow -x_1 - 3x_2 &= -3 && (2)
\end{aligned}
$$

第 1 段

$$
\begin{aligned}
2x_0 - x_1 + x_2 &= 0 && (0)\\
\frac{3}{2}x_1 - \frac{1}{2}x_2 &= 1 && (1)\\
(2) + (1)\times\frac{2}{3} \rightarrow -\frac{7}{3}x_2 &= -\frac{7}{3} && (2)
\end{aligned}
$$

8

続き

$$\begin{aligned} 2x_0 - x_1 + x_2 &= 0 \quad (0)\\ \frac{3}{2}x_1 - \frac{1}{2}x_2 &= 1 \quad (1)\\ -\frac{7}{3}x_2 &= -\frac{7}{3} \quad (2) \end{aligned}$$

上三角方程式を後退代入により解く

(2) から　$x_2 = -\frac{7}{3}/(-\frac{7}{3}) = 1$

(1) から　$x_1 = (1+\frac{1}{2}x_2)/\frac{3}{2} = (1+\frac{1}{2})/\frac{3}{2} = 1$

(0) から　$x_0 = (0 + x_1 - x_2)/2 = (0+1-1)/2 = 0$

解は $(x_0, x_1, x_2) = (0, 1, 1)$

9

数値計算のプログラム

連立方程式の係数、行列 **A** とベクトル **B** は、それぞれ 2 次元配列 a_{ij} 、 1 次元配列 b_i として与える。$(i, j = 0, 1, 2, ..., n-1)$

取り組むべき問題を明確にするため、前進消去過程と後退代入過程を分けてプログラムする。（関数を用いる）

1. 与えられた係数 **A**, **B** に対して前進消去を行い、上三角化された **A'**, **B'** を求める。
2. 上三角化された **A'**, **B'** から後退代入により解を求める。

10

プログラムの骨格（例）

連立一次方程式の次元を n、
係数 a_{ij}, b_i は配列に格納されているとする。$(i, j = 0, 1, 2, ..., n-1)$

```
#define SIZE 3 /* 次元 n の指定 */

double a[SIZE][SIZE] = {{2,-1,1},{-1,2,-1},{2,-2,-1}};
double b[SIZE]={0,1,-3};

main()
{
  double a_after[SIZE][SIZE], b_after[SIZE], x[SIZE];

  report(a, b);
  if( gauss(a, b, a_after, b_after) != 0){
    report(a_after, b_after);
    solve(a_after, b_after, x);
  } else printf("解けませんでした！\n");
}
```

11

関数の説明

関数 report は、行列（ 2 次元配列）とベクトル（ 1 次元配列）を受け取り、その内容を表示する。

```
void report(double matrix[SIZE][SIZE], double vector[SIZE])
{
  int i, j;

  for(i=0; i<SIZE; i++){
    for(j=0; j<SIZE; j++)
      printf("%f ", matrix[i][j]);
    printf(" = %f\n", vector[i]);
  }
  printf("\n");
}
```

12

続き

関数 gauss は、引数 a と b を受け取って前進消去を行い、結果を引数 a2, b2 を通じて返す。上三角化できれば（ゼロになる軸がなければ）1 (int) を、できなければ 0 (int) を返す。

```
int gauss(double a[SIZE][SIZE], double b[SIZE],double
                            a2[SIZE][SIZE], double b2[SIZE])
{
  double a_tmp[SIZE][SIZE], b_tmp[SIZE];
  int success = 1;

  手順
  １：ローカル変数 a_tmp, b_tmp に引数 a, b の内容をコピー。
  ２：a_tmp, b_tmp を上三角化。できなければ success = 0;
  ３：引き数 a2, b2 に a_tmp, b_tmp の内容をコピー。

  return(success);
}
```

13

前進消去のアルゴリズム

第 k 段：式 (k) に注目 ($k = 0, 1, 2, ..., n-1$)

式 (i) に式 (k) の $-a_{ik}/a_{kk}$ 倍を足す ($i = k+1, k+2, ..., n-1$)

具体的には $\begin{cases} a'_{ij} = a_{ij} - a_{ik}/a_{kk} * a_{kj} & (j = 0, 1, 2, ..., n-1) \\ b'_i = b_i - a_{ik}/a_{kk} * b_k & j = k+1 \text{ から始めると効率が良い} \end{cases}$

```
for(k=0; k<SIZE; k++){
  if( a[k][k] != 0){ /* もし軸がゼロでなければ */
    for(自 分 で 考 え て ね){
      alpha = -a[i][k]/a[k][k];
      for(自 分 で 考 え て ね)
        a[i][j] = 自 分 で 考 え て ね
      b[i] = 自 分 で 考 え て ね
    }
  }
}
```

14

続き

関数 solve は、引数 a と b を受け取り、後退代入により解いた解を引数 solution として返す。a と b は既に上三角化されているものとする。

```
void solve(double a[SIZE][SIZE], double b[SIZE],
                                 double solution[SIZE])
{
  int i;
  double x[SIZE];

  手順
  １：引数 a, b に対して後退代入を行い、解を計算（配列 x に格納）
  ２：x の内容を引き数 solution にコピー
}
```

15

数値計算具体例

$$A = \begin{bmatrix} 2 & -1 & 1 \\ -1 & 2 & -1 \\ 2 & -2 & -1 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 1 \\ -3 \end{bmatrix}$$ の場合の計算結果

```
% ./a.out
2.000000 -1.000000 1.000000  = 0.000000
-1.000000 2.000000 -1.000000  = 1.000000
2.000000 -2.000000 -1.000000  = -3.000000

2.000000 -1.000000 1.000000  = 0.000000
0.000000 1.500000 -0.500000  = 1.000000
0.000000 0.000000 -2.333333  = -2.333333

0.000000
1.000000
1.000000
```

元式の表示

上三角化した式の表示

解の表示

16

Mathematica で連立方程式を解く

% /usr/local/bin/mathematica で Mathematica を起動。

```
In[8]:=
a = {{2,-1,1},{-1,0,-1},{2,-2,-1}}
b = {0,1,-3}
Out[8]=
{{2, -1, 1}, {-1, 0, -1}, {2, -2, -1}}
Out[9]=
{0, 1, -3}
In[10]:=
Inverse[a]
Out[10]=
{{-2, -3, 1}, {-3, -4, 1}, {2, 2, -1}}
In[11]:=
sol = Inverse[a].b
Out[11]=
{-6, -7, 5}
In[12]:=
a.sol
Out[12]=
{0, 1, -3}
```

太字の部分を入力し、シフト＋リターンで実行する。

行列 a とベクトル b の定義

行列 a の逆行列を Inverse[a] で求める。

ベクトル b に左から行列 a の逆行列をかけたものを sol とする。行列とベクトルの積はドット . 。

sol が解であることを確認。

17

問題 1

$$[a_{ij}]=\begin{bmatrix}1&1&0\\1&0&2\\0&1&1\end{bmatrix},\quad [b_j]=\begin{bmatrix}0\\1\\-1\end{bmatrix}$$

の時、 $\mathbf{Ax}=\mathbf{b}$ の解 (x_0, x_1, x_2) を数値的に求めよ。これを手計算の結果と比較せよ。

$n=5$

$a_{ij}=1/(i+j+1),\ b_j=1\ \ (i, j=0, 1, 2, ..., n-1)$

の時、解 $(x_0, x_1, x_2, ..., x_4)$ を数値的に求めよ。
厳密解は $(5, -120, 630, -1120, 630)$ である。
（ Mathematica で確認せよ）

$$[a_{ij}]=\begin{bmatrix}1&1&1&-1\\1&1&-1&1\\1&-1&1&1\\-1&1&1&1\end{bmatrix},\quad [b_j]=\begin{bmatrix}1\\2\\3\\4\end{bmatrix}$$

の時、解 (x_0, x_1, x_2, x_3) を数値的および手計算で求めよ。数値的に解が求められない場合、その理由を考えよ。

18

ピボットの選択

Gauss の消去法では、前進消去の過程で対角成分 a_{kk} がゼロになると都合が悪い（a_{kk} による割り算が含まれているため、上三角化できなくなる）。

また、対角成分 a_{kk} がゼロでなくても、他の要素に比べて非常に小さな値である場合、a_{kk}での割り算は大きな丸め誤差を含むことになり、計算精度の点からよろしくない。

対角成分 a_{kk} のことを軸（ピボット）と呼ぶ。これがゼロもしくは小さな値にならないように、行もしくは列の入れ替えをすることを軸選択（ピボット選択）という。行だけの入れ替えを行うものを部分ピボット選択、行と列の入れかを行うものを完全ピボット選択と呼ぶ。

列の入れ替えは変数 x_j の入れ替えを伴うため、処理がややこしくなる。本講義では行の入れ替えのみを行う部分ピボット選択を取り扱う。

19

部分ピボット選択のアルゴリズム

第 k 段の過程（第 k 行に注目中）で、$|a_{ik}|\ (i=k, k+1, k+2, ..., n-1)$ が最大になる行番号を ip とする。$ip \neq k$ なら k 行と ip 行を入れ替える。

この操作により、ピボット a_{kk} がゼロもしくは他の要素よりも極端に小さな値になることを避けることができる。

$$\begin{aligned}
&a_{00}x_0+a_{01}x_1+a_{02}x_2+\ldots &&+a_{0n-1}x_{n-1}=b_0 &&(0)\\
&a_{10}x_0+a_{11}x_1+a_{12}x_2+\ldots &&+a_{1n-1}x_{n-1}=b_1 &&(1)\\
&a_{20}x_0+a_{21}x_1+a_{22}x_2+\ldots &&+a_{2n-1}x_{n-1}=b_2 &&(2)\\
&\cdot\\
&\cdot\\
&a_{n-10}x_0+a_{n-11}x_1+a_{n-12}x_2+\ldots+a_{n-1n-1}x_{n-1}=b_{n-1} &&&(n-1)
\end{aligned}$$

例えば、第 0 段において、a_{20} が絶対値最大であれば、第 0 行と第 2 行を入れ替える。入れ換え後の $a'_{00}\ (=a_{20})$ をピボットとして x_0 の係数を消去。

20

プログラムの骨格

ピボット選択をしない先程のプログラムを少し修正するだけで可能。

```
for(k=0; k<SIZE; k++){

  ip = select_pivot(a, k); /* ピボットの選択 */
  swap_row(a, ip, k);      /* ip行と k行の入れ替え */

  if( a[k][k]!= 0){
    for 自 分 で 考 え て ね {
      alpha = -a[i][k]/a[k][k];
      for 自 分 で 考 え て ね
        a[i][j] = 自 分 で 考 え て ね
      b[i] = 自 分 で 考 え て ね
    }
  }
}
```

21

ピボットの探索

関数 select_pivot は行列 matrix の k 段のピボット (int) を返す関数

```
int select_pivot(double matrix[SIZE][SIZE], int k)
{
  int ip = k;
  double pivot = fabs(matrix[k][k]);

  自 分 で 考 え て ね

  return(ip);
}
```

22

行の入れ替え

関数 swap_row は行列 a とベクトル b の第 *ip* 行と第 *k* 行を入れ替える。

```
void swap_row(double a[SIZE][SIZE],double b[SIZE],
                                        int ip, int k)
{
  int j;
  double a_tmp[SIZE], b_tmp;

  for(j=0; j<SIZE; j++) a_tmp[j] = a[ip][j];
  b_tmp = b[ip];

  for(j=0; j<SIZE; j++) a[ip][j] = a[k][j];
  b[ip] = b[k];

  for(j=0; j<SIZE; j++) a[k][j] = a_tmp[j];
  b[k] = b_tmp;
}
```

23

逆行列の計算

次の n 個の連立一次方程式を考える。

$$\mathbf{A}\mathbf{x}_i = \mathbf{I}_i \qquad \mathbf{x}_i = \begin{bmatrix} x_{0i} \\ x_{1i} \\ x_{2i} \\ . \\ x_{n-1i} \end{bmatrix}, \quad \mathbf{I}_i = \begin{bmatrix} 0 \\ 0 \\ 1 \\ . \\ 0 \end{bmatrix} \leftarrow i \text{ 番目の要素のみが 1 のベクトル}$$

$i = 0, 1, 2, ..., n-1$

これを行列の形で書き下すと

$$A \begin{bmatrix} x_{00} & x_{01} & \dots & x_{0n-1} \\ x_{10} & x_{11} & \dots & x_{1n-1} \\ x_{20} & x_{21} & \dots & x_{2n-1} \\ . & . & . & . \\ x_{n-10} & x_{n-11} & \dots & x_{n-1n-1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ . & . & . & \dots & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$\mathbf{x}_0$ $\mathbf{x}_1$ $\mathbf{x}_{n-1}$ 単位行列

24

続き

先程の n 個の連立方程式の解 $\mathbf{x}_i$ を縦に並べてできる行列は、
$\mathbf{A}$ の逆行列になっている。

逆行列を求めるアルゴリズム

$i = 0, 1, 2, 3, ..., n-1$ について

$\mathbf{A}\mathbf{x}_i = \mathbf{I}_i$　を解いて、解 $\mathbf{x}_i$ を第 i 列に持つ行列 $\mathbf{B}$ を作る

解けなければ逆行列は存在しない。

$\mathbf{B} = \{ \mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_{n-1} \}$

行列 $\mathbf{B}$ は $\mathbf{A}$ の逆行列である。

25

問題 2

$\mathbf{A} = \begin{bmatrix} 1 & -3 \\ 2 & 1 \end{bmatrix}$　の逆行列 $\mathbf{A}^{-1}$を数値的に求め、手計算の結果と比較せよ。
$\mathbf{A}\,\mathbf{A}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$ となることを確認せよ。

$\mathbf{A} = \begin{bmatrix} 1 & 1 & 1 & -1 \\ 1 & 1 & -1 & 1 \\ 1 & -1 & 1 & 1 \\ -1 & 1 & 1 & 1 \end{bmatrix}$　の逆行列 $\mathbf{A}^{-1}$を数値的に求めよ。
$\mathbf{A}\,\mathbf{A}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$ となることを確認せよ。

$\mathbf{A} = \{a_{ij}\}$
$a_{ij} = 1/(i+j+1)\ (i, j = 0, 1, 2, ..., 4)$

の逆行列 $\mathbf{A}^{-1}$を数値的に求めよ。
$\mathbf{A}\,\mathbf{A}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$ となることを確認せよ。

26

行列式の計算

行列式の性質として次がある（線形代数より）

1）ある行に定数をかけたものを別の行に加えても行列式は不変
2）２つの行を入れ替えると符号 (±) が変わる。

以上の性質を用いると、Gauss の前進消去過程により行列 $\mathbf{A}$ が上三角行列 $\mathbf{A}'$ に変形できたとき、行列 $\mathbf{A}$ の行列式 $\det \mathbf{A}$ は、

$$\det \mathbf{A} = (-1)^p \det \mathbf{A}'$$

で与えられる。ここで p は、軸選択により前進消去の過程で行を入れ替えた回数である。

上三角行列 $\mathbf{A}'$ の行列式は、対角成分の積。 $\det \mathbf{A}'$ は容易に計算可能。

27

行列式を計算するプログラム

既に作成した前進消去過程を行う関数 gauss を修正し、行を入れ替えた回数も引き数を通じて返すようにする。

```
main()
{
  double a_after[SIZE][SIZE], b_after[SIZE], x[SIZE];
  int swap_count; /* 行の入れ替え数を格納する変数 */

  report(a, b);
  if( gauss(a, b, a_after, b_after, &swap_count) != 0){
    report(a_after, b_after);
    det = determinant(a_after, swap_count);
  } else det = 0; /* 上三角化できなければ det = 0 */

  printf("det = %f\n", det);
}
```

28

続き

```
int gauss(double a[SIZE][SIZE], double b[SIZE],
       double a2[SIZE][SIZE], double b2[SIZE], int *swap_count)
{
  double a_tmp[SIZE][SIZE], b_tmp[SIZE];
  int success = 1, count=0;

  手順
  １：ローカル変数 a_tmp, b_tmp に引き数 a, b の内容をコピー。
  ２：a_tmp, b_tmp を上三角化。行の入れ替えがあれば count++;
     上三角化できなければ success = 0;
  ３：引き数 a2, b2 に a_tmp, b_tmp の内容をコピー。

  *swap_count = count; /* ポインタを介して値を返却 */

  return(success);
}
```

29

続き

行列 **A** を上三角化した行列 **A'** と、行の入れ換えの回数を受け取って、行列 **A** の行列式を戻り値として返却する関数 determinant

```
double determinant(double matrix[SIZE][SIZE], int count)
{
  int i, sgn=-1;
  double det = 1;

  自 分 で 考 え て ね

  /* 対角成分の積の計算 */

  /* 符号の計算 */

  return(det);
}
```

30

問題 3

$$\mathbf{A}=\begin{bmatrix}1 & 1 & 0\\ 1 & 0 & 2\\ 0 & 1 & 1\end{bmatrix}$$

の行列式 det **A** を数値計算で求め、手計算の結果と比較せよ。

$$\mathbf{A}=\begin{bmatrix}1 & 2 & 3 & 4\\ 1^2 & 2^2 & 3^2 & 4^2\\ 1^3 & 2^3 & 3^3 & 4^3\\ 1^4 & 2^4 & 3^4 & 4^4\end{bmatrix}$$

の行列式 det **A** を数値計算で求め、手計算の結果と比較せよ。

成分が $a_{ij}=1+i+j \quad (i,j=0,1,2,...,4)$ である行列 **A** の行列式 det **A** を数値計算で求め、手計算の結果と比較せよ。

31

連立方程式の数値解法に関する２つのアプローチ

直接法（直接解法）

加減乗除などの操作を有限回組み合わせて解を求めるアプローチ。

ある意味直感的。具体的な解析手順と対比させやすい。

丸め誤差の累積が避けられない。

Gauss の消去法、Gauss Jordan 法、LU 分解、など

反復法（反復解法）

ある操作を反復して解の近似値を真の値に近づけていくアプローチ。

一般に、真の解に必ずしも収束するとは限らない。

Jacobi 法、Gauss Seidel 法、など

１）問題 1, 2, 3 を解き、ソースプログラムとともに提出。
２）連立一次方程式の数値解法に関するさまざまな手法について調べてレポートにまとめて提出。

32