計算機実験 1

高須担当 : 2004 12/10, 12/17, 2005 1/7 の 3 回

1) 連立一次方程式の数値解法
- Gaussの消去法
- ピボット選択
- 逆行列の計算
- 行列式の計算

2) 微分方程式の数値解法
- 時間の差分化
- Euler 法
- Runge Kutta 法
- 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
レポートは LaTeXで作成すること
提出先は、高須 @ G311、提出期限は 1) 12月 31 日、2) 1 月 16 日

1

Gauss の消去法

n 次元連立一次方程式を考える。

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

$$\mathbf{Ax} = \mathbf{b} \qquad \mathbf{A} = (a_{ij}),\ \mathbf{b} = (b_i)$$

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

2

消去法のアルゴリズム

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

第 0 段

a_{00} がゼロでないとき、a_{00} を軸（ピボット）にして式 (1) 以降の x_0 の係数を消去

式 (1) から式 (0) * (a_{10}/a_{00}) を引くと、式 (1) の x_0 の係数はゼロ

式 (2) から式 (0) * (a_{20}/a_{00}) を引くと、式 (2) の x_0 の係数はゼロ

...

式 (n–1) から式 (0) * ($a_{n-1,0}/a_{00}$) を引くと、式 (n–1) の x_0 の係数はゼロ

3

アルゴリズム続き

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

第 1 段

$a^{(1)}{}_{11}$ がゼロでないとき、 $a^{(1)}{}_{11}$ を軸にして式 (2) 以降の x_1 の係数を消去

式 (2) から式 (1) * ($a^{(1)}{}_{21}/a^{(1)}{}_{11}$) を引くと、式 (2) の x_1 の係数はゼロ

式 (3) から式 (1) * ($a^{(1)}{}_{31}/a^{(1)}{}_{11}$) を引くと、式 (3) の x_1 の係数はゼロ

...

式 (n–1) から式 (1) * ($a^{(1)}{}_{n-1,1}/a^{(1)}{}_{11}$) を引くと、式 ($n$–1) の x_1 の係数はゼロ

4

続き

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

第 2 段

$a^{(2)}_{22}$ がゼロでないとき、 $a^{(2)}_{22}$ を軸にして式 (3) 以降の x_2 の係数を消去

式 (3) から式 (2) * ($a^{(2)}_{32}/a^{(2)}_{22}$) を引くと、式 (3) の x_2 の係数はゼロ

式 (4) から式 (2) * ($a^{(2)}_{42}/a^{(2)}_{22}$) を引くと、式 (4) の x_2 の係数はゼロ

...

式 (n–1) から式 (2) * ($a^{(2)}_{n-1,2}/a^{(2)}_{22}$) を引くと、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

前進消去アルゴリズム

$k = 0, 1, 2, \ldots, n-2$ に対し、以下の手順 を繰り返す

$i = k+1, k+2, \ldots, n-2, n-1$ に対し、a) + b) + c) を繰り返す

a) $m_{ik} := a^{(k)}_{ik} / a^{(k)}_{kk}$

b) $j = k+1, k+2, \ldots, n-1$ に対し $a^{(k+1)}_{ij} := a^{(k)}_{ij} - m_{ik} \times a^{(k)}_{kj}$

c) $b^{(k+1)}_{i} := b^{(k)}_{i} - m_{ik} \times b^{(k)}_{k}$

a) + b) + c) は、式 (i) から式 (k) の m_{ik}倍を引いて x_k の係数を消去

7

前進消去過程の計算量

	乗除算回数	加減算回数	
第 k 段 :	$(n+1-k)(n-k-1)$	$(n-k)(n-k-1)$	$k = 0, 1, 2, \ldots, n-2$

乗除算回数合計 $\displaystyle\sum_{k=0}^{n-2}(n-k+1)(n-k-1) \sim \frac{1}{3}O(n^3)$

加算回数合計 $\displaystyle\sum_{k=0}^{n-2}(n-k)(n-k-1) \sim \frac{1}{3}O(n^3)$

8

後退代入過程

上三角化された方程式は後退代入を繰り返すことで容易に解くことができる。

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

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

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

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

一般に　$x_i = \left(b_i - \sum_{j=i+1}^{n-1} a_{ij}x_j\right)/a_{ii}$　$(i = n-1, n-2, n-3, \ldots, 2, 1, 0)$

9

後退代入過程アルゴリズム

$i = n-1, n-2, \ldots 2, 1, 0$ に対し、次の手順により解を得る。

$$x_i = \left(b_i - \sum_{j=i+1}^{n-1} a_{ij}x_j\right)/a_{ii}$$

後退代入過程の計算量

乗除算回数　$\sum_{i=0}^{n-1}(n-i) \sim \frac{1}{2}O(n^2)$

加減算回数　$\sum_{i=0}^{n-1}(n-1-i) \sim \frac{1}{2}O(n^2)$

10

Gauss の消去法の計算量

乗除算回数　　加減算回数

$\frac{1}{3}O(n^3)$　　$\frac{1}{3}O(n^3)$

計算量は、変数の数 n の 3乗に比例して増加

11

Gauss の消去法具体例

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

第 0 段

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

第 1 段

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

12

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 $(x_0, x_1, x_2) = (0, 1, 1)$

13

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
分離してプログラムを書く。（関数を定義する）

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'** を求める関数 gauss_elimination()
2. 上三角化された **A'**, **B'** から後退代入により解を求める関数 backward()

14

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

15

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");
}
```

16

関数 gauss_elimination

関数 gauss_elimination 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）int 1 を、できなければ int 0 を返す。

```
int gauss_elimination(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 として終了
  3：引き数 a2, b2 に a_tmp, b_tmp の内容をコピー

  return success;
}
```

17

前進消去プログラム

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

式 (i) から式 (k) の a_{ik}/a_{kk} 倍を引く $(i = k+1, k+2, ..., n-1)$

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] != 0.0 ){ /* もし軸がゼロでなければ */
    for                    {
      m = a[i][k]/a[k][k];
      for
        a[i][j] =
      b[i] =
    }
  } /* end of if */
}
```

18

関数 backward

関数 backward は、引数 a と b を受け取り、後退代入によって解いた解を引数 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 backward(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
}
```

19

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

20

補足：関数の演算結果を返却する方法

C 言語では、引き数を介して関数にデータを受け渡す。関数内部の演算結果を関数呼び出し側に返すには二通りのやり方がある。

1) 関数の戻り値（返却値）として返す
2) 引数を介して返す

```
double sqr(double);

main()
{
  double x = 2.0, y = 0.0;

  y = sqr(x);
  printf("%f %f\n", x, y);
}

double sqr(double x)
{
  return x*x;
}
```

演算結果を関数の戻り値として返す例

実引数 x を関数 sqr に受け渡し、結果は関数の戻り値として y に代入される。

21

引数を介して演算結果を返す方法

関数の戻り値として演算結果を返す場合、戻り値の数は1つに限られる。また、複雑なデータ構造を戻り値とすることはややこしい（不可能ではない）

引数を介して演算結果を返す場合には、参照渡しを用いる（ポインタを使う）

```
void sqr_new(double, dobule*);

main()
{
  double x = 3.14156, y = 0.0;

  sqr_new(x, &y);

  printf("%f %f\n", x, y);
}

void sqr_new(double x, double *y)
{
  *y = x*x;
}
```

関数 sqr_new を 2 つの引数で呼び出す。第 2 引数に第 1 引数の値の 2 乗が格納される。

ポインタ変数 y の参照として x*x の値を代入

22

不良例

```
void sqr_new(double, dobule);

main()
{
  double x = 3.14156, y = 0.0;

  sqr_new(x, y);

  printf("%f %f\n", x, y);
}

void sqr_new(double x, double y)
{
  y = x*x;
}
```

正しく動かない例である

データは値渡しで関数に受け渡される

関数 sqr_new 内部のローカル変数 y の値は x*x になるが、関数呼び出し元の変数 y は不変。

値渡しでは、関数呼び出し側の引数の値を変えることはできない。

23

配列名 = ポインタ

配列名は、その配列へのポインタである。
関数の引数として配列名を与えると、参照渡しになる。
（関数内部で配列要素の値を変更すると、関数呼び出し元に反映される）

```
void manipulate_vector(double[], int size);

main()
{
  int i;
  double x[3] = {0.0, 0.0, 0.0};

  manipulate_vector(x, 3);
  for(i=0; i<3; i++)
    printf("%f¥n", x[i]);
}

void manipulate_vector(double x[], int size)
{
  int i;
  for(i=0; i<size; i++)
    x[i] += 1.0;
}
```

配列 x の内容は関数呼び出し後変更される。

関数内で配列 x の要素を変更

24

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

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

Mathematica は大文字小文字を区別するので注意

25

Gauss Jordan 法（掃き出し法）

n 次元連立一次方程式を考える。

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

ある式に別の式の定数倍を足す操作を繰り返し、対角方程式持ち込む解法が Gauss Jordan 法（別名掃き出し法）である。

26

Gauss Jordan 法のアルゴリズム

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

第 0 段

a_{00} がゼロでないとき、a_{00} を軸（ピボット）にして式 (1) 以降の x_0 の係数を消去

式 (1) から式 (0) * (a_{10}/a_{00}) を引くと、式 (1) の x_0 の係数はゼロ

式 (2) から式 (0) * (a_{20}/a_{00}) を引くと、式 (2) の x_0 の係数はゼロ

…

式 (n–1) から式 (0) * ($a_{n-1,0}/a_{00}$) を引くと、式 (n–1) の x_0 の係数はゼロ

27

アルゴリズム続き

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

第 1 段

$a^{(1)}{}_{11}$ がゼロでないとき、 $a^{(1)}{}_{11}$ を軸にして式 (1) 以外の x_1 の係数を消去

式 (0) から式 (1) * ($a^{(1)}{}_{01}/a^{(1)}{}_{11}$) を引くと、式 (0) の x_1 の係数はゼロ

式 (2) から式 (1) * ($a^{(1)}{}_{21}/a^{(1)}{}_{11}$) を引くと、式 (2) の x_1 の係数はゼロ

…

式 (n–1) から式 (1) * ($a^{(1)}{}_{n-1,1}/a^{(1)}{}_{11}$) を引くと、式 ($n$–1) の x_1 の係数はゼロ

28

続き

$$
\begin{aligned}
a_{00}x_0 &+ a_{01}x_1 + \ldots + a_{0,n-1}x_{n-1} = b_0 && (0)\\
a_{11}^{(1)}x_1 &+ a_{12}^{(1)}x_2 + \ldots + a_{1,n-1}^{(1)}x_{n-1} = b_1^{(1)} && (1)\\
a_{22}^{(2)}x_2 &+ \ldots + a_{2,n-1}^{(2)}x_{n-1} = b_2^{(2)} && (2)\\
a_{32}^{(2)}x_2 &+ \ldots + a_{3,n-1}^{(2)}x_{n-1} = b_3^{(2)} && (3)\\
a_{n-1,2}^{(2)}x_2 &+ \ldots + a_{n-1,n-1}^{(2)}x_{n-1} = b_{n-1}^{(2)} && (n-1)
\end{aligned}
$$

第 2 段

$a^{(2)}{}_{22}$ がゼロでないとき、 $a^{(2)}{}_{22}$ を軸にして式 (2) 以外の x_2 の係数を消去

式 (0) から式 (2) * $(a^{(2)}{}_{02}/a^{(2)}{}_{22})$ を引くと、式 (0) の x_2 の係数はゼロ

式 (1) から式 (2) * $(a^{(2)}{}_{12}/a^{(2)}{}_{22})$ を引くと、式 (1) の x_2 の係数はゼロ

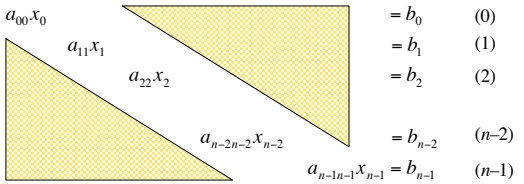
…

式 $(n-1)$ から式 (2) * $(a^{(2)}{}_{n-1,2}/a^{(2)}{}_{22})$ を引くと、式 $(n-1)$ の x_2 の係数はゼロ

29

Gauss Jordan 法の最終型

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。



係数は対角成分のみが残る。係数 a_{ij}, b_i はオリジナルの値とは異なる。

解は　$x_i = b_i / a_{ii}$　　$i = 0, 1, 2, \ldots, n-1$

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）。

30

Gauss Jordan 法の計算量

第 k 段において

乗除算回数 : $(n-1)(n-k+2)$　　加減算回数 : $(n-1)(n-k+1)$

乗除算回数合計　$\displaystyle\sum_{k=0}^{n-1}(n-1)(n-k+2) \sim \frac{1}{2}O(n^3)$

加減算回数合計　$\displaystyle\sum_{k=0}^{n-1}(n-1)(n-k+1) \sim \frac{1}{2}O(n^3)$

対角方程式から解を求める際に

乗除算回数合計　n

Gauss Jordan 法の計算量は n^3 に比例。Gauss の消去法よりも計算量は 3/2 倍多い

31

問題 1

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

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。

$n = 5$, $\mathbf{A} = (a_{ij})$, $\mathbf{b} = (b_i)$
$a_{ij} = 1.0/(i + j+1)$,　$b_j = 1$　$(i, j = 0, 1, 2, \ldots, n-1)$

の時、解 $(x_0, x_1, x_2, \ldots, x_4)$ 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）

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

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。

32

問題 1 続き

$$[a_{ij}]=\begin{bmatrix}4 & 1 & -3 & 0\\ 2 & 0 & 3 & -2\\ 0 & -1 & 3 & 2\\ 10 & -4 & 0 & -3\end{bmatrix},\quad [b_j]=\begin{bmatrix}0\\ 0\\ 25\\ 5\end{bmatrix}$$

の時、解 (x_0, x_1, x_2, x_3) を数値的及び手計算で求めよ。

$n = 10$, $\mathbf{A} = (a_{ij})$, $\mathbf{b} = (b_i)$

$a_{ij} = 1 + 1.0/(i+1) + 1.0/(j+1) + (x/10)^2$,　$b_j = 1$　$(i, j = 0, 1, 2, ..., n-1)$

の時、解 $\mathbf{x} = (x_0, x_1, x_2, ..., x_9)$ を数値的に求めよ。
$\mathbf{Ax} - \mathbf{b}$ を計算し、得られた数値解がどの程度正確な解であるか確認せよ。
$\mathbf{x}$ が真の解であれば $\mathbf{Ax} - \mathbf{b}$ はゼロベクトルとなるはずである。

Gauss の消去法及び Gauss Jordan 法について、乗除算、加減算の計算量を求めよ。

33