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・桁落ち誤差を含むことになり、計算精度上問題がある。

対角成分 a_{kk} のことを軸（ピボット）と呼んだ。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_i 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。

1

誤差の例

次の連立方程式を Gauss の消去法で解く。ただし有効数字は 4 桁 (5 桁目を四捨五入) とする。

$$-0.001x_0 + 6x_1 = 6.001 \quad (1)$$
$$3x_0 + 5x_1 = 2 \quad (2)$$

解は $(x_0, x_1) = (-1, 1)$

(1) × 3000　$-3.000x_0 + 1.800 \times 10^4 x_1 = 1.800 \times 10^4$

(2) の x_0 を消去　$1.801 \times 10^4 x_1 = 1.800 \times 10^4$

$$x_1 = \frac{1.800 \times 10^4}{1.801 \times 10^{-4}} \sim 9.994 \times 10^{-1}$$

$$x_0 = \frac{6.001 - 6.000 \times 9.994 \times 10^{-1}}{-1.000 \times 10^{-3}} \sim \frac{6.001 - 5.996}{-1.000 \times 10^{-3}} \sim -5.000$$

2

続き

軸係数が大きいものと入れ替える

$$3x_0 + 5x_2 = 2 \quad (1)$$
$$-0.001x_0 + 6x_1 = 6.001 \quad (2)$$

(1) × 0.001/3　$-0.001x_0 + 1.667 \times 10^{-3} x_1 = 6.667 \times^{-4}$

(2) の x_0 を消去　$6.002x_1 = 6.002$

$$x_1 = 1.000$$

$$x_0 = \frac{6.667 \times 10^{-4} - 1.667 \times 10^{-3} \times 1.000}{-0.001} \sim \frac{-1.000 \times 10^{-3}}{-0.001} = 1.000$$

前進消去過程においては適当な軸選択が有効

3

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、$|a_{ik}|\,(i = k, k+1, k+2, \ldots, n-1)$ 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。

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

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。

4

プログラムの骨格

ピボット選択をしない前進消去関数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_rows(a, ip, k);       /* ip行と k行の入れ替え */

  if( a[k][k]!= 0){
    for                    {
      m = a[i][k]/a[k][k];
      for
        a[i][j] =
      b[i] =
    }
  }
}
```

5

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]);

  return ip;
}
```

fabs は倍精度型値の絶対値を返す関数。math.h をインクルードして使用。

6

行の入れ替え

関数 swap_rows 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_rows(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;
}
```

7

問題 2

問題 1 を、部分軸選択を伴う Gauss の消去法で解き直せ。部分軸選択をしなかった前回の結果と比較して、近似解の誤差を評価せよ。

変数の型を double型（有効桁数16桁）から float型（同 7桁）へ変更して計算するとどうなるか確認せよ。

Gauss の消去法は万能ではない。
例えば次の 10次元連立1次方程式を数値的に解いて得られる解 **x** が、どの程度正確な解であるかを確認せよ。

$$n = 10,\ \mathbf{A} = (a_{ij}),\ \mathbf{b} = (b_i)$$
$$a_{ij} = 1.0/(i + j + 1),\ \ b_j = 1.0\ \ (i, j = 0, 1, 2, ..., n-1)$$

x が $\mathbf{Ax} = \mathbf{b}$ の解なら $\mathbf{Ax} - \mathbf{b}$ はゼロベクトルとなるはずである。

8

逆行列

行列 **A** が与えられたとき、$\mathbf{AX} = \mathbf{XA} = \mathbf{I}$ を満たす行列 **X** を
行列 **A** の逆行列と呼び $\mathbf{X} = \mathbf{A}^{-1}$ と表記する。**I** は単位行列である。

$$A = \begin{pmatrix} 3 & 2 \\ 4 & 3 \end{pmatrix} \qquad A^{-1} = \begin{pmatrix} 3 & -2 \\ -4 & 3 \end{pmatrix}$$

$$AA^{-1} = A^{-1}A = I$$

逆行列の数値解法は、連立一次方程式の解法（Gauss の消去法等）と
密接に関係している。

9

逆行列の計算

次の n 個の連立一次方程式を考える。

$\mathbf{Ax}_i = \mathbf{I}_i$　　$\mathbf{x}_i = \begin{bmatrix} x_{0i} \\ x_{1i} \\ x_{2i} \\ . \\ x_{n-1i} \end{bmatrix}$, $\mathbf{I}_i = \begin{bmatrix} 0 \\ 0 \\ 1 \\ . \\ 0 \end{bmatrix}$ ← i 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

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

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

$\mathbf{x}_i$ を並べて作った
行列 **X** は **A** の逆行列

$$\mathbf{AX} = \mathbf{I}$$

10

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、
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

$\mathbf{Ax}_i = \mathbf{I}_i$　を解いて、解 $\mathbf{x}_i$ を第 i 列に持つ行列 **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 **B** は **A** の逆行列である。

11

逆行列を計算するプログラム

```
#define SIZE 3 /* 次元 n の指定 */

double a[SIZE][SIZE] = {{2,-1,1},{-1,2,-1},{2,-2,-1}};
double e[SIZE][SIZE] = {{1,0,0},{0,1,0},{0,0,1}};
double b[SIZE][SIZE]; /* 行列 b に a の逆行列を格納 */

double a_after[SIZE][SIZE], e_after[SIZE], x[SIZE];

for(i=0; i<SIZE; i++){

  if( gauss_elimination(a, b[i], a_after, e_after) != 0){
    backward(a_after, e_after, x);
  } else printf("解けませんでした！\n");

  解 x を行列 b の第 i 列目として設定

}
```

12

問題 3

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

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。

$n = 5$, $\mathbf{A} = (a_{ij})$
$a_{ij} = 1.0/(i + j + 1)$ $(i, j = 0, 1, 2, ..., 4)$

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。

13

LU 分解

行列 **A** が下三角行列 **L** と上三角行列 **U** の積で表現できたとする。

$$\mathbf{A} = \mathbf{LU}$$

連立1次方程式 $\mathbf{Ax} = \mathbf{LUx} = \mathbf{b}$ の解 **x** は

$\mathbf{Ly} = \mathbf{b}$ の解 **y** を求めた後、$\mathbf{Ux} = \mathbf{y}$ を解いたものに等しい。

$\mathbf{Ly} = \mathbf{b}$ は下三角方程式であり、前進代入で容易に解ける。
$\mathbf{Ux} = \mathbf{y}$ は上三角方程式であり、後退代入で容易に解ける。
それぞれ計算量は n^2 のオーダー

Gauss の消去法において軸選択を行わずに上三角化可能な行列 **A** は一意に $\mathbf{A} = \mathbf{LU}$ の形に分解可能。これを LU 分解と呼ぶ。

14

LU 分解の応用

n 次元正方行列 **A** の逆行列を求めるために、先程の例では n 組の連立1次方程式を解いた。

$$\mathbf{Ax}_i = \mathbf{I}_i \qquad i = 0, 1, 2, ..., n-1$$

1 組の連立方程式を解く際の計算量は n^3 のオーダー

一度行列 **A** を **LU** 分解しておけば、n^2 の計算量で $\mathbf{x}_i$ を求めることが可能。

係数行列 **A** は不変であるが、ベクトル項 **b** のみが変わる場合に LU 分解を用いると効率的に計算可能（後述の反復法）

$$\mathbf{Ax} = \mathbf{b}$$

15

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) 1 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 **A** が上三角行列 **A'** に変形できたとき、行列 **A** の行列式 det **A** は、

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

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

上三角行列 **A'** の行列式は対角成分の積に等しい。 det **A'** は容易に計算可能。

16

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_elimination 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_det(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);
}
```

引き数を介して結果を返すのでポインタを用いる

17

続き

```
int gauss_det(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;
}
```

18

続き

行列 A を上三角化した行列 **A**' と、前進消去過程で行った行の入れ換え回数を受け取り、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)
{       /* matrix は既に上三角化されている */
  int i, sgn = -1;
  double det = 1;

  [          ]   /* 対角成分の積の計算 */

  [          ]   /* 符号の計算 */

  return  det;
}
```

19

問題 4

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

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。

20

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、真の解に必ず収束するとは限らない。

Jacobi 法、Gauss Seidel 法、など

21

反復法の例

Gauss の消去法（直接法）で第 1近似解を求めた後、反復法により
より良い近似解を求めてみる。

$\mathbf{Ax} = \mathbf{b}$ を直説法で解いた解を $\mathbf{x}^0$ とする。真の解を $\mathbf{x}^*$ とする。

近似解 $\mathbf{x}^0$ は誤差 $\delta\mathbf{x}$ を含む： $\mathbf{x}^0 = \mathbf{x}^* + \delta\mathbf{x}$

両辺に $\mathbf{A}$ をかけて、 $\mathbf{Ax}^0 = \mathbf{Ax}^* + \mathbf{A}\,\delta\mathbf{x}$

$\mathbf{Ax}^* = \mathbf{b}$ であるから $\mathbf{A}\,\delta\mathbf{x} = \mathbf{Ax}^0 - \mathbf{b}$

$\mathbf{b} - \mathbf{Ax}^0$ を残差と呼ぶ

$\mathbf{A}$ (誤差) = – (残差)

右辺は既知量である。これを $\delta\mathbf{x}$ について解く（$\delta\mathbf{x}$ も誤差を含む）。

$\mathbf{x}^1 = \mathbf{x}^0 - \delta\mathbf{x}$ としてより良い近似値 $\mathbf{x}^1$ を得る。

$\mathbf{x}^1$ を基にしてより良い近似値 $\mathbf{x}^2$ を得る ・・・

残差がある程度小さくなるまで繰り返す。

22

レポート

1) 問題 1, 2, 3, 4 を解け。
2) Gauss の消去法は、一般にどのような行列に対して有効であるか調べよ。
3) 良く用いられる反復法のアルゴリズムについて調べ、反復法が有効である行列の性質について述べよ。
4) 数値計算で用いたプログラムをレポートの最後に記載すること。

以上についてレポートを LaTeX で作成し、pdf 形式に変換したものを高須までメールに添付して提出。レポートの表紙は配付したひな形を用いること。

締め切りは 2004年 12月 31日。期限を過ぎたレポートは受け取らない。

紙媒体に印刷したレポートも高須@G311 に提出（年明けでも良い）。

23