数値解析
Numerical Analysis

連立一次方程式
Systems of linear equations

電気・電子情報工学系　市川　周一
Shuichi Ichikawa, Dept. EEIE

本日の内容
Index

- 連立一次方程式 (System of linear equations)
  - 連立一次方程式は，それ自体も重要な計算対象
  - 他の多くの問題において，重要な計算手段として用いられる
  - 多くの解法がある
- 掃き出し法 (row reduction),
  ガウスの消去法(Gaussian elimination)
  - 多くの選択肢がある
  - 本講義では，なるべく単純な方法を採用する

復習： ベクトルと行列
Revisited: Vectors and Matrices

- n次元ベクトル → n要素の一次元配列
- m×n 行列 → （m×n）要素の二次元配列

C言語で，どのように表現し，どのように扱うか？

$$\mathbf{x} = \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{pmatrix} \qquad \mathbf{A} = \begin{pmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,n-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,n-1} \\ \vdots & \vdots & & \vdots \\ a_{m-1,0} & a_{m-1,1} & \cdots & a_{m-1,n-1} \end{pmatrix}$$

例：　ベクトルの加算
Example: Vector Addition

- ベクトルは，配列で表現
  - 3つの配列を変数として定義
- 関数vaddの引数として，配列のアドレスを渡す
  - 値を返してもらうため
  - 参照渡し
- 配列名＝最初の要素のアドレス

$$\mathbf{x} = \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{N-1} \end{pmatrix}, \mathbf{y} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{N-1} \end{pmatrix}, \mathbf{z} = \begin{pmatrix} z_0 \\ z_1 \\ \vdots \\ z_{N-1} \end{pmatrix},$$

$$\mathbf{z} = \mathbf{x} + \mathbf{y}$$

```
#define N        3

 double x[N], y[N], z[N];

 // 途中省略

 vadd(z, x, y);
```

これと同じ意味

```
 vadd(&z[0], &x[0], &y[0]);
```

```
void vadd(double a[N], double b[N], double c[N])
{
    int i;

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

こういう計算が行われる

```
for (i = 0; i < N; i++) z[i] = x[i] + y[i];
```

復習：　2次元配列の変数宣言
Revisited: Definition of a 2D array

- 行列
  - 2次元の配列
  - 添字の順番に注意！
    - 特に正方行列でない場合
- 初期化 Initialization 可能
  - 右図参照

```
double a[M][N];
```

M行N列

$$\mathbf{A} = \begin{pmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,N-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,N-1} \\ \vdots & \vdots & & \vdots \\ a_{M-1,0} & a_{M-1,1} & \cdots & a_{M-1,N-1} \end{pmatrix}$$

```
double a[2][3] = {
    {1, 2, 3},
    {4, 5, 6}
};
```

$$\mathbf{A} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

例：　行列の加算
Example: Matrix Addition

```
double x[M][N], y[M][N], z[M][N];

// 途中省略. 行列x,yの値は設定されているとする

MatAdd(z, x, y);
```

```
void MatAdd(double a[M][N], double b[M][N], double c[M][N])
{
    int i, j;

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

問題：n元連立一次方程式
Problem: a system of n linear equations

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

$$
\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}, \quad \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix},
$$

$$
\mathbf{Ax} = \mathbf{b} \quad \Rightarrow \quad \mathbf{x} = \mathbf{A}^{-1}\mathbf{b}
$$

- 行列Aに逆行列があれば，解は$\mathbf{A}^{-1}\mathbf{b}$
  - しかし逆行列をクラメルの公式で求めると計算量が多い
- 直接逆行列を求めずに，解を求めたい
  - 欲しいのは解．
  - 逆行列ではない．
- いろいろな方法がある

掃き出し法（1）
Row reduction (1)

- 1行目を a_{11}^{-1} 倍する
  - 前提： $a_{11} \neq 0$
  - $a_{11} = 0$ なら行の入れ替えが必要
    - ピボット選択
    - とりあえず後で考える

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

$$\Downarrow$$

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

掃き出し法（2）
Row reduction (2)

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

- k行目（k≠1）から
  - （1行目）×a_{k1}を引いて
  - x_1を消去

$$\Downarrow$$

$$\begin{cases} x_1 + \dfrac{a_{12}}{a_{11}} x_2 + \cdots + \dfrac{a_{1n}}{a_{11}} x_n = \dfrac{b_1}{a_{11}} \\ \left(a_{22} - \dfrac{a_{21}a_{12}}{a_{11}}\right) x_2 + \cdots + \left(a_{2n} - \dfrac{a_{21}a_{1n}}{a_{11}}\right) x_n = b_2 - \dfrac{a_{21}b_1}{a_{11}} \\ \vdots \\ \left(a_{n2} - \dfrac{a_{n1}a_{12}}{a_{11}}\right) x_2 + \cdots + \left(a_{nn} - \dfrac{a_{n1}a_{1n}}{a_{11}}\right) x_n = b_n - \dfrac{a_{n1}b_1}{a_{11}} \end{cases}$$

$$\Leftrightarrow$$

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

掃き出し法（3）
Row reduction (3)

$$
\begin{cases}
x_1 + a_{12}^{(1)} x_2 + a_{13}^{(1)} x_3 + \cdots + a_{1n}^{(1)} x_n = b_1^{(1)} \\
\qquad a_{22}^{(1)} x_2 + a_{23}^{(1)} x_3 + \cdots + a_{2n}^{(1)} x_n = b_2^{(1)} \\
\qquad \qquad \vdots \\
\qquad a_{n2}^{(1)} x_2 + a_{n3}^{(1)} x_3 + \cdots + a_{nn}^{(1)} x_n = b_n^{(1)}
\end{cases}
$$

$$\Downarrow$$

$$
\begin{cases}
x_1 + \qquad + a_{13}^{(2)} x_3 + \cdots + a_{1n}^{(2)} x_n = b_1^{(2)} \\
\qquad x_2 + a_{23}^{(2)} x_3 + \cdots + a_{2n}^{(2)} x_n = b_2^{(2)} \\
\qquad \qquad \vdots \\
\qquad \qquad a_{n3}^{(2)} x_3 + \cdots + a_{nn}^{(2)} x_n = b_n^{(3)}
\end{cases}
\Rightarrow
\begin{cases}
x_1 = b_1^{(n)} \\
x_2 = b_2^{(n)} \\
\quad \vdots \\
x_n = b_n^{(n)}
\end{cases}
$$

- 同様にして,
  - x_2 以降の変数も消去する

掃き出し法のデータ構造 Data structure for row reduction

- 色々な方法で実現できる
- 本講義では右の表現を用いる
  - 右下の行列Aを2次元配列として実現する
  - この形なら，行列Aだけ操作すればよい
    - Aとbを両方使うより，処理が少し(だけ)簡単

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n + b_1 = 0 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n + b_2 = 0 \\ \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n + b_n = 0 \end{cases}$$

$$\mathbf{Ax} = \mathbf{0}$$

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\ \vdots & \vdots & & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} & b_n \end{pmatrix}, \quad \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \\ 1 \end{pmatrix}.$$

例1： 計算過程
Example 1

- 下の連立一次方程式を掃き出し法で解く

$$\begin{cases} 2x + y + z - 7 = 0 \\ x + 2y + z - 8 = 0 \\ x + y + 2z - 9 = 0 \end{cases}$$

$$\Downarrow$$

$$\begin{cases} x - 1 = 0 \\ y - 2 = 0 \\ z - 3 = 0 \end{cases}$$

```
2.000000 1.000000 1.000000 -7.000000
1.000000 2.000000 1.000000 -8.000000
1.000000 1.000000 2.000000 -9.000000
                ⇓
1.000000 0.500000 0.500000 -3.500000
1.000000 2.000000 1.000000 -8.000000
1.000000 1.000000 2.000000 -9.000000
                ⇓
1.000000 0.500000 0.500000 -3.500000
0.000000 1.500000 0.500000 -4.500000
0.000000 0.500000 1.500000 -5.500000
                ⇓
1.000000 0.500000 0.500000 -3.500000
0.000000 1.000000 0.333333 -3.000000
0.000000 0.500000 1.500000 -5.500000
                ⇓
1.000000 0.000000 0.333333 -2.000000
0.000000 1.000000 0.333333 -3.000000
0.000000 0.000000 1.333333 -4.000000
                ⇓
1.000000 0.000000 0.333333 -2.000000
0.000000 1.000000 0.333333 -3.000000
0.000000 0.000000 1.000000 -3.000000
                ⇓
1.000000 0.000000 0.000000 -1.000000
0.000000 1.000000 0.000000 -2.000000
0.000000 0.000000 1.000000 -3.000000
```

掃き出し法のプログラム（暫定版）
A program of row reduction (Tentative)

```
void SolveArray(x)
double x[DIM][DIM+1];
{
    int i, j, k;
    double d;

    for (i = 0; i < DIM; i++) {
        d = x[i][i];
        for (k = i; k <= DIM; k++) x[i][k] /= d;
        for (j = 0; j < DIM; j++) {
            if (i != j) {
                d = x[j][i];
                for (k = i; k <= DIM; k++) x[j][k] -= x[i][k]*d;
            }
        }
    }
}
```

こういう書き方もできる．
void SolveArray(double x[DIM][DIM+1])と同じ

第i行目を処理する

対角要素x[i][i]を1にする

要素x[j][i] (i≠j)を0にする

たったこれだけ！短いコード

復習：　代入演算子
Revisited: assignment operator

- 代入と演算を同時に指定する演算子

書き方	等価な代入文
x = y	x = y
x += y	x = x + y
x -= y	x = x - y
x *= y	x = x * y
x /= y	x = x / y
x %= y	x = x % y

```
for (i = 0; i < max; i++)
    f = f + 0.1;
```

⇕

```
for (i = 0; i < max; i++)
    f += 0.1;
```

例2: 計算過程
Example 2

- 下の連立一次方程式を掃き出し法で解く

$$\begin{cases} 2x+y+z-7=0 \\ 2x+y+2z-10=0 \\ x+2y+2z-11=0 \end{cases}$$

↓

正解 $\begin{cases} x=1 \\ y=2 \\ z=3 \end{cases}$

ところが…

```
2.000000 1.000000 1.000000 -7.000000
2.000000 1.000000 2.000000 -10.000000
1.000000 2.000000 2.000000 -11.000000

1.000000 0.500000 0.500000 -3.500000
2.000000 1.000000 2.000000 -10.000000
1.000000 2.000000 2.000000 -11.000000

1.000000 0.500000 0.500000 -3.500000
0.000000 0.000000 1.000000 -3.000000
0.000000 1.500000 1.500000 -7.500000

1.000000 0.500000 0.500000 -3.500000
0.000000 nan inf -inf
0.000000 1.500000 1.500000 -7.500000

1.000000 nan -inf inf
0.000000 nan inf -inf
0.000000 nan -inf inf

1.000000 nan -inf inf
0.000000 nan inf -inf
0.000000 nan nan nan

1.000000 nan nan nan
0.000000 nan nan nan
0.000000 nan nan nan
```

対角要素が0になってしまった

解けない!!

例3： 計算過程
Example 3

- 行を入れ替えれば解ける
  - 対角要素を非零にする

$$
\begin{cases}
2x + y + z - 7 = 0 \\
2x + y + 2z - 10 = 0 \\
x + 2y + 2z - 11 = 0
\end{cases}
$$

解けない!!

等価 ⇕ 2行目と3行目を入替

$$
\begin{cases}
2x + y + z - 7 = 0 \\
x + 2y + 2z - 11 = 0 \\
2x + y + 2z - 10 = 0
\end{cases}
$$

解ける!!

```
2.000000 1.000000 1.000000 -7.000000
1.000000 2.000000 2.000000 -11.000000
2.000000 1.000000 2.000000 -10.000000

1.000000 0.500000 0.500000 -3.500000
1.000000 2.000000 2.000000 -11.000000
2.000000 1.000000 2.000000 -10.000000

1.000000 0.500000 0.500000 -3.500000
0.000000 1.500000 1.500000 -7.500000
0.000000 0.000000 1.000000 -3.000000

1.000000 0.500000 0.500000 -3.500000
0.000000 1.000000 1.000000 -5.000000
0.000000 0.000000 1.000000 -3.000000

1.000000 0.000000 0.000000 -1.000000
0.000000 1.000000 1.000000 -5.000000
0.000000 0.000000 1.000000 -3.000000

1.000000 0.000000 0.000000 -1.000000
0.000000 1.000000 1.000000 -5.000000
0.000000 0.000000 1.000000 -3.000000

1.000000 0.000000 0.000000 -1.000000
0.000000 1.000000 0.000000 -2.000000
0.000000 0.000000 1.000000 -3.000000
```

掃き出し法のプログラム（暫定版）How to solve this problem

```
void SolveArray(x)
double x[DIM][DIM+1];
{
    int i, j, k;
    double d;

    for (i = 0; i < DIM; i++) {
        d = x[i][i];
        for (k = i; k <= DIM; k++) x[i][k] /= d;
        for (j = 0; j < DIM; j++) {
            if (i != j) {
                d = x[j][i];
                for (k = i; k <= DIM; k++) x[j][k] -= x[i][k]*d;
            }
        }
    }
}
```

x[i][i]が0にならないよう，必要ならここで行を交換する

x[i][i]が0だと，0/0 → nanになる

ここはif文の代わりにcontinueでも書ける（後述）

行の入れ替え
Change the order of equations

- d = x[i][i]; の前に次のようなコードを挿入すればOK

```
if (x[i][i] == 0.0) {
    printf("--- pivot is zero¥n");
    // exchange row
    for (j = i; j < DIM; j++) {
        if (x[j][i] != 0.0) {
            printf("--- exchanging row_%d and row_%d¥n", i, j);
            for (k = i; k <= DIM; k++) {
                d = x[i][k];
                x[i][k] = x[j][k];
                x[j][k] = d;
            }
            break;
        }
    }
    if (j == DIM) {
        // no non-zero pivot is available
        printf("There is no non-zero pivot. Aborting...¥n");
        exit(1);
    }
}
```

これは何か？ （次ページ参照）

交換できる行がなければ，ランク不足で解けないということ

プログラムの終了 （マニュアル参照）

ループの脱出：break文

- break文を実行すると，実行中のループから脱出する
  - 例：　ループで配列を探索して，発見したら次へ進む

```
 :
 for ( ; ; ) {
         :
       break;
         :
 }
 :
 :
```

脱出

For文で例を示したが，while文などでも使用可

そのfor文の次の文へ移動

ループの継続：continueの文

- continue文 → 実行中のループを，次の回に進める
  - continue → 後処理 → 条件判断 → 成立すればループ本体を実行
  - 例：　繰返しの“今回”をスキップして次に進めたいとき

最初へ

```
  :
  for ( ; ; ) {
          :
       continue;
          :
  }
  :
  :
```

For文で例を示したが，while文などでも使用可

例2:　計算過程(改良後)
After adding the code

```
2.000000 1.000000 1.000000 -7.000000
2.000000 1.000000 2.000000 -10.000000
1.000000 2.000000 2.000000 -11.000000

1.000000 0.500000 0.500000 -3.500000
2.000000 1.000000 2.000000 -10.000000
1.000000 2.000000 2.000000 -11.000000

1.000000 0.500000 0.500000 -3.500000
0.000000 0.000000 1.000000 -3.000000
0.000000 1.500000 1.500000 -7.500000

1.000000 0.500000 0.500000 -3.500000
0.000000 nan inf -inf
0.000000 1.500000 1.500000 -7.500000

1.000000 nan -inf inf
0.000000 nan inf -inf
0.000000 nan -inf inf

1.000000 nan -inf inf
0.000000 nan inf -inf
0.000000 nan nan nan

1.000000 nan nan nan
0.000000 nan nan nan
0.000000 nan nan nan
```

```
2.000000 1.000000 1.000000 -7.000000
2.000000 1.000000 2.000000 -10.000000
1.000000 2.000000 2.000000 -11.000000

1.000000 0.500000 0.500000 -3.500000
2.000000 1.000000 2.000000 -10.000000
1.000000 2.000000 2.000000 -11.000000

1.000000 0.500000 0.500000 -3.500000
0.000000 0.000000 1.000000 -3.000000
0.000000 1.500000 1.500000 -7.500000

--- pivot is zero
--- exchanging row_1 and row_2
1.000000 0.500000 0.500000 -3.500000
0.000000 1.000000 1.000000 -5.000000
0.000000 0.000000 1.000000 -3.000000

1.000000 0.000000 0.000000 -1.000000
0.000000 1.000000 1.000000 -5.000000
0.000000 0.000000 1.000000 -3.000000

1.000000 0.000000 0.000000 -1.000000
0.000000 1.000000 1.000000 -5.000000
0.000000 0.000000 1.000000 -3.000000

1.000000 0.000000 0.000000 -1.000000
0.000000 1.000000 0.000000 -2.000000
0.000000 0.000000 1.000000 -3.000000
```

行を入替！

解けた!!

結果の確認 Verification

- SolveArray実行後
  - 行列の最後の列 → $-x_i$
- 最初の行列Aを使って，$Ax \approx 0$ を確認する
  - 残差 (residue)
    - 計算誤差のチェック
  - 計算が正しくても，残差は僅かに0と異なる
    - 大きく異なればバグを疑う
    - データ次第で桁落ちもある
    - 充分小さければ可とする
- 残差を減らすための工夫もいろいろある
  - 数値計算の書籍を参照せよ

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

SolveArray

$$\begin{pmatrix} 1 & 0 & \cdots & 0 & -x_1 \\ 0 & 1 & \cdots & 0 & -x_2 \\ \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & -x_n \end{pmatrix}$$

$$\mathbf{Ax} = \mathbf{0}$$

結果の確認 CheckArray

- 連立方程式の各方程式について残差を表示する

```
void CheckArray(x, y)
double x[DIM][DIM+1];
double y[DIM][DIM+1];
{
    int i, j;
    double s;

    printf("=== Residue Check ===¥n");
    for (i = 0; i < DIM; i++) {
        s = 0.0;
        for (j = 0; j < DIM; j++) {
            s += -x[j][DIM] * y[i][j];
        }
        s += y[i][DIM];
        printf("%+e¥n", s);
    }
}
```

行列xはSolveArrayした行列，
行列yはSolveArrayする前の行列

第i番目の方程式の左辺を計算

解の第j要素

本来0であるべき → 誤差が出る

行列の複写
CopyArray

- SolveArrayすると，行列が書き変わってしまう
- CheckArrayするために，SolveArray前に複写しておく

```
int main()
{
    double A[DIM][DIM+1] = {
        { 2, 1, 1, -7 },
        { 1, 2, 1, -8 },
        { 1, 1, 2, -9 }
    };
    double AA[DIM][DIM+1];

    CopyArray(AA, A);
    SolveArray(A);
    CheckArray(A, AA);

    return 0;
}
```

```
void
CopyArray(to, from)
double to[DIM][DIM+1];
double from[DIM][DIM+1];
{
    int i, j;

    for (i = 0; i < DIM; i++) {
        for (j = 0; j <= DIM; j++) {
            to[i][j] = from[i][j];
        }
    }
}
```

実行例
Example

- 最初に行列を複写
  - あとでCheckするため
- SolveArrayの途中
  - 行列の状態を出力
- SolveArrayが終わったあと
  - 解を出力
- 最後にCheckArray
  - 残差を出力

```
-2.070705e+00 +6.809707e+00 -2.933278e+00 -1.068331e+00
-3.626145e+00 +7.728569e+00 -9.688343e+00 +1.681804e+00
-6.812627e+00 -2.325683e+00 +3.820087e+00 -8.822822e+00

+1.000000e+00 -3.288594e+00 +1.416560e+00 +5.159265e-01
-3.626145e+00 +7.728569e+00 -9.688343e+00 +1.681804e+00
-6.812627e+00 -2.325683e+00 +3.820087e+00 -8.822822e+00

+1.000000e+00 -3.288594e+00 +1.416560e+00 +5.159265e-01
+0.000000e+00 -4.196350e+00 -4.551690e+00 +3.552628e+00
+0.000000e+00 -2.472965e+01 +1.347059e+01 -5.308007e+00

+1.000000e+00 -3.288594e+00 +1.416560e+00 +5.159265e-01
+0.000000e+00 +1.000000e+00 +1.084678e+00 -8.465996e-01
+0.000000e+00 -2.472965e+01 +1.347059e+01 -5.308007e+00

+1.000000e+00 +0.000000e+00 +4.983628e+00 -2.268196e+00
+0.000000e+00 +1.000000e+00 +1.084678e+00 -8.465996e-01
+0.000000e+00 +0.000000e+00 +4.029430e+01 -2.624412e+01

+1.000000e+00 +0.000000e+00 +4.983628e+00 -2.268196e+00
+0.000000e+00 +1.000000e+00 +1.084678e+00 -8.465996e-01
+0.000000e+00 +0.000000e+00 +1.000000e+00 -6.513109e-01

+1.000000e+00 +0.000000e+00 +0.000000e+00 +9.776948e-01
+0.000000e+00 +1.000000e+00 +0.000000e+00 -1.401367e-01
+0.000000e+00 +0.000000e+00 +1.000000e+00 -6.513109e-01

=== Dumping Solution ===
-9.776948e-01
+1.401367e-01
+6.513109e-01

=== Residue Check ===
-2.220446e-16
-8.881784e-16
+0.000000e+00
```

倍精度演算としては充分な精度

【発展】 逆行列を求める
【See Also】 Matrix inversion

- 掃き出し法で，逆行列を求めることもできる．考えてみよ．
  - 下の例は，ランダムな行列の逆行列を求めた例．（よく見ればヒントになっています）

```
$ ./a.out
-2.070705e+00 +6.809707e+00 -2.933278e+00 +1.000000e+00 +0.000000e+00 +0.000000e+00
-1.068331e+00 -3.626145e+00 +7.728569e+00 +0.000000e+00 +1.000000e+00 +0.000000e+00
-9.688343e+00 +1.681804e+00 -6.812627e+00 +0.000000e+00 +0.000000e+00 +1.000000e+00

+1.000000e+00 -3.288594e+00 +1.416560e+00 -4.829274e-01 -0.000000e+00 -0.000000e+00
-1.068331e+00 -3.626145e+00 +7.728569e+00 +0.000000e+00 +1.000000e+00 +0.000000e+00
-9.688343e+00 +1.681804e+00 -6.812627e+00 +0.000000e+00 +0.000000e+00 +1.000000e+00

+1.000000e+00 -3.288594e+00 +1.416560e+00 -4.829274e-01 -0.000000e+00 -0.000000e+00
+0.000000e+00 -7.139453e+00 +9.241924e+00 -5.159265e-01 +1.000000e+00 +0.000000e+00
+0.000000e+00 -3.017923e+01 +6.911496e+00 -4.678767e+00 +0.000000e+00 +1.000000e+00

+1.000000e+00 -3.288594e+00 +1.416560e+00 -4.829274e-01 -0.000000e+00 -0.000000e+00
+0.000000e+00 +1.000000e+00 -1.294486e+00 +7.226415e-02 -1.400668e-01 -0.000000e+00
+0.000000e+00 -3.017923e+01 +6.911496e+00 -4.678767e+00 +0.000000e+00 +1.000000e+00

+1.000000e+00 +0.000000e+00 -2.840480e+00 -2.452799e-01 -4.606227e-01 -0.000000e+00
+0.000000e+00 +1.000000e+00 -1.294486e+00 +7.226415e-02 -1.400668e-01 -0.000000e+00
+0.000000e+00 +0.000000e+00 -3.215510e+01 -2.497890e+00 -4.227106e+00 +1.000000e+00

+1.000000e+00 +0.000000e+00 -2.840480e+00 -2.452799e-01 -4.606227e-01 -0.000000e+00
+0.000000e+00 +1.000000e+00 -1.294486e+00 +7.226415e-02 -1.400668e-01 -0.000000e+00
+0.000000e+00 +0.000000e+00 +1.000000e+00 +7.768255e-02 +1.314599e-01 -3.109926e-02

+1.000000e+00 +0.000000e+00 +0.000000e+00 -2.462419e-02 -8.721351e-02 -8.833685e-02
+0.000000e+00 +1.000000e+00 +0.000000e+00 +1.728232e-01 +3.010629e-02 -4.025757e-02
+0.000000e+00 +0.000000e+00 +1.000000e+00 +7.768255e-02 +1.314599e-01 -3.109926e-02

+1.000000e+00 +0.000000e+00 +1.387779e-17
+1.110223e-16 +1.000000e+00 -2.775558e-17
+4.440892e-16 +0.000000e+00 +1.000000e+00
```

行列の基本操作
Operations on a matrix

- $P_{i,j}$を左から掛ける
  - 行iと行jの交換
- $Q_{i,c}$を左から掛ける
  - 行iが定数c倍される
- $R_{i,j,c}$を左から掛ける
  - 行iに(行j)×cが加算される
- X_iをi回目の操作とする
  - X_iは$P_{i,j}$，$Q_{i,c}$，$R_{i,j,c}$のいずれか

$$Ax = b$$

$$X_k X_{k-1} \cdots X_1 A x = X_k X_{k-1} \cdots X_1 b$$

$$x = X_k X_{k-1} \cdots X_1 b$$

$$P_{i,j} = \begin{pmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 0 & & 1 & & \\ & & & \ddots & & & \\ & & 1 & & 0 & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{pmatrix} \quad (1 \le i \ne j \le n)$$

$$Q_{i,c} = \begin{pmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 1 & & & & \\ & & & c & & & \\ & & & & 1 & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{pmatrix} \quad (1 \le i \le n, c \ne 0)$$

$$R_{i,j,c} = \begin{pmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 1 & & c & & \\ & & & \ddots & & & \\ & & & & 1 & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{pmatrix} \quad (1 \le i \ne j \le n, c \ne 0)$$

逆行列の求めかた How to calculate the inverse matrix

- 掃き出し法において，
- 定数ベクタbのかわりに，
- 単位行列Eに操作を適用すれば，
- 逆行列が求まる
  - 逆行列が存在するなら！

$$Ax = b$$

$$X_k X_{k-1} \cdots X_1 Ax = X_k X_{k-1} \cdots X_1 b$$

$$x = X_k X_{k-1} \cdots X_1 b$$

$$\Downarrow$$

$$X_k X_{k-1} \cdots X_1 A = E$$

$$A^{-1} = X_k X_{k-1} \cdots X_1$$

補足

- 実社会で用いられている掃き出し法は，ここまで説明した掃き出し法とは少し違う
- 実際には，前進消去～後退代入する方法が多い

掃き出し法の別法
Another method

- これまでの説明では，係数行列を直接「対角化」した．
- 実際には，以下の2段階に分けて実行することが多い
  - 前進消去 Forward elimination
    - 係数行列を上三角化する
    - 最後の行で，未知数が1つ求まる
  - 後退代入 Back substitution
    - 求まった未知数を，上の式に代入して消去する
      - 下から2行目で，未知数が1つ求まる
    - この操作を繰り返して全ての未知数を求める
- プログラムは殆ど同じ
  - Program code is mostly similar

前進消去（1）
Forward elimination (1)

- 1行目を $a_{11}{}^{-1}$ 倍する
  - 前提： $a_{11} \neq 0$
  - $a_{11} = 0$ なら行の入れ替えが必要
    - ピボット選択
    - とりあえず後で考える
- 2～n行目からx_1を消去

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

$$\Downarrow$$

$$\begin{cases} x_1 + a_{12}^{(1)}x_2 + a_{13}^{(1)}x_3 + \cdots + a_{1n}^{(1)}x_n = b_1^{(1)} \\ \qquad a_{22}^{(1)}x_2 + a_{23}^{(1)}x_3 + \cdots + a_{2n}^{(1)}x_n = b_2^{(1)} \\ \qquad \vdots \\ \qquad a_{n2}^{(1)}x_2 + a_{n3}^{(1)}x_3 + \cdots + a_{nn}^{(1)}x_n = b_n^{(1)} \end{cases}$$

前進消去（2）
Forward elimination (2)

- 同様にして2行目以下も処理する
- k行目を処理するときは，k+1行目以降のx_kを消去
  - 1～k行目は触らない
- 係数行列は上三角行列になる
- この過程を前進消去という

$$
\begin{cases}
x_1 + a_{12}^{(1)} x_2 + a_{13}^{(1)} x_3 + \cdots + a_{1n}^{(1)} x_n = b_1^{(1)} \\
\qquad a_{22}^{(1)} x_2 + a_{23}^{(1)} x_3 + \cdots + a_{2n}^{(1)} x_n = b_2^{(1)} \\
\qquad \vdots \\
\qquad a_{n2}^{(1)} x_2 + a_{n3}^{(1)} x_3 + \cdots + a_{nn}^{(1)} x_n = b_n^{(1)}
\end{cases}
$$

$$\Downarrow$$

$$
\begin{cases}
x_1 + a_{12}^{(1)} x_2 + a_{13}^{(1)} x_3 + \cdots + a_{1n}^{(1)} x_n = b_1^{(1)} \\
\qquad x_2 + a_{23}^{(2)} x_3 + \cdots + a_{2n}^{(2)} x_n = b_2^{(2)} \\
\qquad \vdots \\
\qquad x_n = b_n^{(n)}
\end{cases}
$$

後退代入（1）
Backward substitution (1)

- x_nを1～(n-1)行目に代入して，x_nを消去する
  - n行目を使って1～(n−1)行目のx_nを消去する

$$
\begin{cases}
x_1 + a_{12}^{(1)} x_2 + a_{13}^{(1)} x_3 + \cdots + a_{1n}^{(1)} x_n = b_1^{(1)} \\
x_2 + a_{23}^{(2)} x_3 + \cdots + a_{2n}^{(2)} x_n = b_2^{(2)} \\
\vdots \\
x_{n-1} + a_{n-1,n}^{(n-1)} x_n = b_{n-1}^{(n-1)} \\
x_n = b_n^{(n)}
\end{cases}
$$

$$\Downarrow$$

$$
\begin{cases}
x_1 + a_{12}^{(1)} x_2 + \cdots + a_{1,n-1}^{(1)} x_{n-1} = b_1^{(n+1)} \\
x_2 + \cdots + a_{2,n-1}^{(2)} x_{n-1} = b_2^{(n+1)} \\
\vdots \\
x_{n-1} = b_{n-1}^{(n+1)} \\
x_n = b_n^{(n)}
\end{cases}
$$

後退代入（2）
Backward substitution (2)

- 同様にして下から変数を消去してゆく
  - x_kを1～(k-1)行目に代入して，x_kを消去する
- 最後には係数行列は対角化される

$$
\begin{cases}
x_1 + a_{12}^{(1)} x_2 + \cdots + a_{1,n-1}^{(1)} x_{n-1} &= b_1^{(n+1)} \\
x_2 + \cdots + a_{2,n-1}^{(2)} x_{n-1} &= b_2^{(n+1)} \\
&\vdots \\
x_{n-1} &= b_{n-1}^{(n+1)} \\
x_n &= b_n^{(n)}
\end{cases}
$$

$$\Downarrow$$

$$
\begin{cases}
x_1 &= b_1^{(2n)} \\
x_2 &= b_2^{(2n-1)} \\
&\vdots \\
x_{n-1} &= b_{n-1}^{(n+1)} \\
x_n &= b_n^{(n)}
\end{cases}
$$

プログラムの変更点 (1) 前進消去
Modified code (1) forward elimination

```
for (i = 0; i < DIM; i++) {
    d = x[i][i];
    for (k = i; k <= DIM; k++) x[i][k] /= d;
    for (j = 0; j < DIM; j++) {
        if (i != j) {
            d = x[j][i];
            for (k = i; k <= DIM; k++) x[j][k] -= x[i][k]*d;
        }
    }
}
```

```
for (i = 0; i < DIM; i++) {
    d = x[i][i];
    for (k = i; k <= DIM; k++) x[i][k] /= d;
    for (j = i+1; j < DIM; j++) {
        d = x[j][i];
        for (k = i; k <= DIM; k++) x[j][k] -= x[i][k]*d;
    }
}
```

前進消去 （コードは短くなる）

プログラムの変更点 (2) 後退代入
Modified code (2) backward substitution

- 前進消去の直後に，後退代入を行う
- 後退代入では係数の計算(対角化)は省略
  - 欲しいのは解なので，解だけ正しく計算すればよい

```
for (i = 0; i < DIM; i++) {
    d = x[i][i];
    for (k = i; k <= DIM; k++) x[i][k] /= d;
    for (j = i+1; j < DIM; j++) {
        d = x[j][i];
        for (k = i; k <= DIM; k++) x[j][k] -= x[i][k]*d;
    }
}
for (i = DIM-1; i >=0; i--) {
    for (j = i-1; j >=0; j--) {
        x[j][DIM] -= x[i][DIM]*x[j][i];
    }
}
```

前進消去 （コードは短くなる）

後退代入 （これだけ！）

実行例

初期状態

1列目を処理

2列目を処理

3列目を処理

誤差は小さい
答は正しい

```
% ./a.out
-2.070705e+00 +6.809707e+00 -2.933278e+00 -1.068331e+00
-3.626145e+00 +7.728569e+00 -9.688343e+00 +1.681804e+00
-6.812627e+00 -2.325683e+00 +3.820087e+00 -8.822822e+00

+1.000000e+00 -3.288594e+00 +1.416560e+00 +5.159265e-01
-3.626145e+00 +7.728569e+00 -9.688343e+00 +1.681804e+00
-6.812627e+00 -2.325683e+00 +3.820087e+00 -8.822822e+00

+1.000000e+00 -3.288594e+00 +1.416560e+00 +5.159265e-01
+0.000000e+00 -4.196350e+00 -4.551690e+00 +3.552628e+00
+0.000000e+00 -2.472965e+01 +1.347059e+01 -5.308007e+00

+1.000000e+00 -3.288594e+00 +1.416560e+00 +5.159265e-01
+0.000000e+00 +1.000000e+00 +1.084678e+00 -8.465996e-01
+0.000000e+00 -2.472965e+01 +1.347059e+01 -5.308007e+00

+1.000000e+00 -3.288594e+00 +1.416560e+00 +5.159265e-01
+0.000000e+00 +1.000000e+00 +1.084678e+00 -8.465996e-01
+0.000000e+00 +0.000000e+00 +4.029430e+01 -2.624412e+01

+1.000000e+00 -3.288594e+00 +1.416560e+00 +5.159265e-01
+0.000000e+00 +1.000000e+00 +1.084678e+00 -8.465996e-01
+0.000000e+00 +0.000000e+00 +1.000000e+00 -6.513109e-01

+1.000000e+00 -3.288594e+00 +1.416560e+00 +5.159265e-01
+0.000000e+00 +1.000000e+00 +1.084678e+00 -8.465996e-01
+0.000000e+00 +0.000000e+00 +1.000000e+00 -6.513109e-01

=== Dumping Solution ===
-9.776948e-01
+1.401367e-01
+6.513109e-01

=== Residue Check ===
-2.220446e-16
-8.881784e-16
+0.000000e+00
%
```

前進消去後

後退代入後