さらに進んだ話題

1 数学関数ライブラリ (math.h) の使用

数学関数ライブラリ

```
#include<stdio.h>
#include<math.h>
int main(void){

   double s, t, u, v, w, x, y;
   double e;

   // 平方根
   s = sqrt(3);
   printf("√ 3 = %f\n", s);

   // べき乗
   t = pow(2, 10);
   printf("2 の 10 乗=%f\n", t);

   //e(=2.718281828... ネピアの数) のべき乗
   e = exp(1);
   printf("e=%f\n", e);

   u = exp(2);
   printf("exp(2)=%f\n", u);

   //自然対数 (底が e)
   v = log(e);
   printf("log(e)=%f\n", v);

   w = log(100);
   printf("log(100)=%f\n", w);

   //常用対数 (底が 10)
   x = log10(10);
   printf("log10(10)=%f\n", x);

   y = log10(1000);
   printf("log10(1000)=%f\n", y);

   return 0;
}
```

実行結果

```
√ 3 = 1.732051
2 の 10 乗=1024.000000
e=2.718282
exp(2)=7.389056
log(e)=1.000000
log(100)=4.605170
log10(10)=1.000000
log10(1000)=3.000000
```

2 多元配列の応用

2.1 掃き出し法

多元配列の応用として、連立多元方程式の解法がある。この問題解法のアルゴリズムは、いろいろと知られているが、ここでは、掃き出し法を扱う。先ずは、例題を用いて、掃き出し法を説明しよう。

例題

$$
\begin{aligned}
3x + 4y - 5z &= 32 && (1)\\
4x - 5y + 3z &= 18 && (2)\\
5x - 3y - 4z &= 2 && (3)
\end{aligned}
$$

(1) 式を 3 で割って、x の係数が 1 となるようにする。

$$
\begin{aligned}
x + \frac{4}{3}y - \frac{5}{3}z &= \frac{32}{3} && (4)\\
4x - 5y + 3z &= 18 && (5)\\
5x - 3y - 4z &= 2 && (6)
\end{aligned}
$$

(5) 式を (5) $- 4 \times$ (4) 式で、(6) 式を (6) $- 5 \times$ (4) 式で置き換えて、x の項を消去する。

$$
\begin{aligned}
x + \frac{4}{3}y - \frac{5}{3}z &= \frac{32}{3} && (7)\\
-\frac{31}{3}y + \frac{29}{3}z &= -\frac{74}{3} && (8)\\
-\frac{29}{3}y + \frac{13}{3}z &= -\frac{154}{3} && (9)
\end{aligned}
$$

(8) 式に $-\frac{3}{31}$ を掛けて、y の係数を 1 にする。

$$
\begin{aligned}
x + \frac{4}{3}y - \frac{5}{3}z &= \frac{32}{3} && (10)\\
y - \frac{29}{31}z &= \frac{74}{31} && (11)\\
-\frac{29}{3}y + \frac{13}{3}z &= -\frac{154}{3} && (12)
\end{aligned}
$$

(10) 式を $(10) - \frac{4}{3} \times (11)$ 式で、(12) 式を $(12) + \frac{29}{3} \times (11)$ 式で置き換えて、y の項を消去する。

$$x - \frac{13}{31}z = \frac{232}{31} \tag{13}$$
$$y - \frac{29}{31}z = \frac{74}{31} \tag{14}$$
$$-\frac{146}{31}z = -\frac{876}{31} \tag{15}$$

(15) 式に $-\frac{31}{146}$ を掛けて、z の係数を 1 にする。

$$x - \frac{13}{31}z = \frac{232}{31} \tag{16}$$
$$y - \frac{29}{31}z = \frac{74}{31} \tag{17}$$
$$z = 6 \tag{18}$$

(16) 式を $(16) + \frac{3}{31} \times (18)$ 式で、(17) 式を $(17) + \frac{29}{31} \times (18)$ 式で置き換えて z の項を消去する。

$$x = 10 \tag{19}$$
$$y = 8 \tag{20}$$
$$z = 6 \tag{21}$$

以上の方法を配列で表現する。

[1] 解くべき方程式の係数を配列で表現する。

$$\begin{pmatrix} 3 & 4 & -5 & 32 \\ 4 & -5 & 3 & 18 \\ 5 & -3 & -4 & 2 \end{pmatrix}$$

[2] 次に、1 行 1 列の要素が 1 となるようにする。

$$\begin{pmatrix} 1 & \frac{4}{3} & -\frac{5}{3} & \frac{32}{3} \\ 4 & -5 & 3 & 18 \\ 5 & -3 & -4 & 2 \end{pmatrix}$$

[3] 2 行 1 列とと 3 行 1 列の要素を 0 にする。

$$\begin{pmatrix} 1 & \frac{4}{3} & -\frac{5}{3} & \frac{32}{3} \\ 0 & -\frac{31}{3} & \frac{29}{3} & -\frac{74}{3} \\ 0 & -\frac{29}{3} & \frac{13}{3} & -\frac{154}{3} \end{pmatrix}$$

[4] 2 行 2 列の要素を 1 にする。

$$\begin{pmatrix} 1 & \frac{4}{3} & -\frac{5}{3} & \frac{32}{3} \\ 0 & 1 & -\frac{29}{31} & \frac{74}{31} \\ 0 & -\frac{29}{3} & \frac{13}{3} & -\frac{154}{3} \end{pmatrix}$$

[5] 1行2列と3行2列の要素を0にする。

$$\begin{pmatrix} 1 & 0 & -\frac{13}{31} & \frac{232}{31} \\ 0 & 1 & -\frac{29}{31} & \frac{74}{31} \\ 0 & 0 & -\frac{146}{31} & -\frac{876}{31} \end{pmatrix}$$

[6] 3行3列の要素を1にする。

$$\begin{pmatrix} 1 & 0 & -\frac{13}{31} & \frac{232}{31} \\ 0 & 1 & -\frac{29}{31} & \frac{74}{31} \\ 0 & 0 & 1 & 6 \end{pmatrix}$$

[7] 1行3列と2行3列の要素を0にする。

$$\begin{pmatrix} 1 & 0 & 0 & 10 \\ 0 & 1 & 0 & 8 \\ 0 & 0 & 1 & 6 \end{pmatrix}$$

上のアルゴリズムを実行するプログラムは、以下の通りである。

プログラム

```
#include<stdio.h>

int main(void){
   int i, j, k;
   double p, q;
   double a[3][4] = {{3, 4, -5, 32}, {4, -5, 3, 18}, {5, -3, -4, 2}};

   for(k=0; k<3; k++){
      p=a[k][k];
      for(j=0; j<4; j++)
         a[k][j] = a[k][j]/p;

      for(i=0; i<3; i++){
         if(i!=k){
            q=a[i][k];
            for(j=0; j<4; j++)
                  a[i][j] = a[i][j]-q*a[k][j];
         }
      }
   }


   for(i=0; i<3; i++){
      for(j=0; j<4; j++)
         printf("a[%d][%d]=%9.6f   ", i, j, a[i][j]);
      printf("\n");
   }


   return 0;
}
```

実行結果

```
a[0][0]= 1.000000  a[0][1]= 0.000000  a[0][2]= 0.000000  a[0][3]=10.000000

a[1][0]=-0.000000  a[1][1]= 1.000000  a[1][2]= 0.000000  a[1][3]= 8.000000

a[2][0]=-0.000000  a[2][1]=-0.000000  a[2][2]= 1.000000  a[2][3]= 6.000000
```

一般に、3元連立1次方程式

$$a_{00}x + a_{01}y + a_{02}z = a_{03}$$
$$a_{10}x + a_{11}y + a_{12}z = a_{13}$$
$$a_{20}x + a_{21}y + a_{22}z = a_{23}$$

が与えられた場合、定数係数も含め、それら係数を取り出して2次元配列 $a[3][4]$ を作る。

$$\begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \end{pmatrix} = \begin{pmatrix} a[0][0] & a[0][1] & a[0][2] & a[0][3] \\ a[1][0] & a[1][1] & a[1][2] & a[1][3] \\ a[2][0] & a[2][1] & a[2][2] & a[2][3] \end{pmatrix}$$

この配列 (行列) に先ほど記したような操作を行い、

$$\begin{pmatrix} 1 & 0 & 0 & a'[0][3] \\ 0 & 1 & 0 & a'[1][3] \\ 0 & 0 & 1 & a'[2][3] \end{pmatrix}$$

となるようにする。すると、最後の4列目の行列要素が元の連立方程式の解である。すなわち、

$$x = a'[0][3], \quad y = a'[1][3], \quad z = a'[2][3]$$

を得る。この方法が使えるためには、対角成分 $a[0][0],\ a[1][1],\ a[2][2]$ が0でないことが必要である。

2.2 連立1次方程式の解 Cramer の公式

掃き出し法は、不完全なアルゴリズムである。多元連立1次方程式の解の完全な公式は、クラメールによって与えられた。

その前に、行列と行列式について述べる。3次の行列

$$A = \begin{pmatrix} a[0][0] & a[0][1] & a[0][2] \\ a[1][0] & a[1][1] & a[1][2] \\ a[2][0] & a[2][1] & a[2][2] \end{pmatrix}$$

の行列式 $|A|$ は、

$$\begin{aligned} |A| &= \begin{vmatrix} a[0][0] & a[0][1] & a[0][2] \\ a[1][0] & a[1][1] & a[1][2] \\ a[2][0] & a[2][1] & a[2][2] \end{vmatrix} \\ &= a[0][0]a[1][1]a[2][2] + a[0][1]a[1][2]a[2][0] + a[0][2]a[1][0]a[2][1] \\ &\quad -a[0][2]a[2][1]a[2][0] + a[0][1]a[1][0]a[2][2] + a[0][0]a[1][2]a[2][1] \end{aligned}$$

この行列式を用いて 3 元連立 1 次方程式

$$a_{00}x + a_{01}y + a_{02}z = a_{03}$$
$$a_{10}x + a_{11}y + a_{12}z = a_{13}$$
$$a_{20}x + a_{21}y + a_{22}z = a_{23}$$

の解を表すと、

$$x = \frac{\begin{vmatrix} a[0][3] & a[0][1] & a[0][2] \\ a[1][3] & a[1][1] & a[1][2] \\ a[2][3] & a[2][1] & a[2][2] \end{vmatrix}}{\begin{vmatrix} a[0][0] & a[0][1] & a[0][2] \\ a[1][0] & a[1][1] & a[1][2] \\ a[2][0] & a[2][1] & a[2][2] \end{vmatrix}}, \quad y = \frac{\begin{vmatrix} a[0][0] & a[0][3] & a[0][2] \\ a[1][0] & a[1][3] & a[1][2] \\ a[2][0] & a[2][3] & a[2][2] \end{vmatrix}}{\begin{vmatrix} a[0][0] & a[0][1] & a[0][2] \\ a[1][0] & a[1][1] & a[1][2] \\ a[2][0] & a[2][1] & a[2][2] \end{vmatrix}}$$

$$z = \frac{\begin{vmatrix} a[0][0] & a[0][1] & a[0][3] \\ a[1][0] & a[1][1] & a[1][3] \\ a[2][0] & a[2][1] & a[2][3] \end{vmatrix}}{\begin{vmatrix} a[0][0] & a[0][1] & a[0][2] \\ a[1][0] & a[1][1] & a[1][2] \\ a[2][0] & a[2][1] & a[2][2] \end{vmatrix}}$$

である。ここで、$|A| = 0$ の場合は 3 つの方程式が 1 次従属となり、解は不定、すなわち、無数の解が存在する。

3 次の行列式を求めるアルゴリズムは、以下の通りである。

3次行列式の計算

```
#include<stdio.h>

int main(void){

   int i, j, k, l;
   double me=1, mo=1;
   double se=0, so=0;
   double a[3][3];
   double det;

   printf("3行3列の行列に値を代入\n");

   for(i=0; i<3; i++){
      printf("%d行目の要素を代入\n", i+1);
      scanf("%lf %lf %lf", &a[i][0], &a[i][1], &a[i][2]);
   }

   for(k=0; k<3; k++){
      me=1;
      mo=1;
      for(i=0; i<3; i++){
         j = (k+i)%3;
         me*=a[i][j];
         l = (k+3-i)%3;
         mo*=a[i][l];
      }
      se+=me;
      so+=mo;
  }

   det=se-so;

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

   return 0;
}
```

2.3　2元連立1次方程式の解法

2元連立1次方程式

$$a_{00}x + a_{01}y = a_{02} \tag{22}$$

$$a_{10}x + a_{11}y = a_{12} \tag{23}$$

(23) 式に a_{01} 掛けて、(22) 式に a_{11} を掛けたものから引くと、

$$\begin{array}{rrrcl} a_{00}a_{11}x & + & a_{01}a_{11}y & = & a_{02}a_{11} \\ \text{-)}\ a_{01}a_{10}x & + & a_{01}a_{11}y & = & a_{12}a_{01} \\ \hline (a_{00}a_{11} - a_{01}a_{10})x & & & = & a_{02}a_{11} - a_{12}a_{01} \end{array}$$

ゆえに、

$$x = \frac{a_{02}a_{11} - a_{12}a_{01}}{a_{00}a_{11} - a_{01}a_{10}} = \frac{\begin{vmatrix} a_{02} & a_{01} \\ a_{12} & a_{11} \end{vmatrix}}{\begin{vmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{vmatrix}}$$

同様に、

$$y = \frac{a_{00}a_{12} - a_{02}a_{10}}{a_{00}a_{11} - a_{01}a_{10}} = \frac{\begin{vmatrix} a_{00} & a_{02} \\ a_{10} & a_{12} \end{vmatrix}}{\begin{vmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{vmatrix}}$$

を得る。但し、

$$a_{00}a_{11} - a_{01}a_{10} \neq 0$$

である。

$a_{00}a_{11} - a_{01}a_{10} = 0$ の場合には、方程式、

$$(a_{00}a_{11} - a_{01}a_{10})x = a_{02}a_{11} - a_{12}a_{01}$$

が成り立つのは、右辺の $a_{02}a_{11} - a_{12}a_{01} = 0$ の場合に限る。この場合には、どんな x でも方程式を満足する。もしも右辺が 0 ならば、この方程式を満足する x は存在しない。ゆえに、

$a_{02}a_{11} - a_{12}a_{01} = 0$ ならば、	解は不定である。
	すなわち、解は無限に存在する。
$a_{02}a_{11} - a_{12}a_{01} \neq 0$ ならば、	解は不能である。
	すなわち、方程式を満たす解は存在しない。

3 乱数とstdlib.h

stdlib.h は、標準ライブラリヘッダーの１つで一般のユーティリティ関数が定義されている。(擬似) 乱数を発生させる関数 rand もこの中で定義されている。乱数の種を決めるには、srand(seed) を用いる。乱数の初期値は seed の値によって決まる。

```
#include<stdio.h>
#include<stdlib.h>

int main(void){
   int i;
   int r;
   int seed= 100;

   srand(seed);

   printf("rand 関数は 0 から%d までの整数の乱数を出力する。\n",RAND_MAX);
   // 10 コの乱数を発生させる。
   for(i=0; i<10; i++){
      r = rand();
      printf("%d\n", r);
   }

   return 0;
}
```

実行結果

```
rand 関数は 0 から 32767 までの整数の乱数を出力する。
365
1216
5415
16704
24504
11254
24698
1702
23209
5629
```

0から$(n-1)$までの整数の乱数を発生させるには、rを$r\%n$で置き換えてやれば良い。

```
/*                                  */
/*    0から9までの乱数を発生        */
/*                                  */

#include<stdio.h>
#include<stdlib.h>

int main(void){
   int i;
   int r;

   int seed= 100;
   srand(seed);

   printf("0から9までの乱数を発生\n");

   for(i=0; i<20; i++){
      r = rand()%10;
      printf("%3d", r);
   }

   return 0;
}
```

実行例

```
0から9までの乱数を発生
  5  6  5  4  4  4  8  2  9  9  0  5  4  3  9  6  8  1  9  1
```

$0 < x < 1$ の乱数 x を発生させるためには、

$$x = (\text{double})(\text{rand}( ) + 1)/(\text{double})(\text{RAND\_MAX} + 2)$$

と置けば良い。

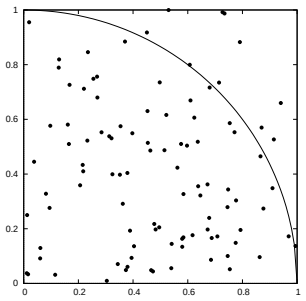
```
/*                          */
/*  0<x<1 の乱数発生        */
/*                          */

#include<stdio.h>
#include<stdlib.h>

int main(void){
   int i;
   double x;

   int seed= 100;
   srand(seed);

   printf(" 0<x<1 の乱数発生\n");
   for(i=0; i<10; i++){
      x = (double)(rand()+1)/(double)(RAND_MAX+2);
     printf("%f\n", x);
   }

   return 0;
}
```

実行例

```
 0<x<1 の乱数発生
0.011169
0.037139
0.165278
0.509781
0.747810
0.343465
0.753731
0.051970
0.708291
0.171809
```

3.1　モンテカルロ法による円周率の計算

$0 < x < 1,\ 0 < y < 1$ の正方形の内部にランダムに点をたくさん打つ。その中で、原点を中心とする半径 1 の四分円の中に落ちた点の数を数えると、その点の割合は四分円の面積 $\pi/4$ に近い値となる。このように、ランダムに点を打って、その点の数から求めたい図形の面積などを知る方法を、ギャンブルで有名な土地の名にちなんで、**モンテカルロ法**と言う。




```
#include<stdio.h>
#include<stdlib.h>

int main(void){
   int i, j, n, count=0;
   double x, y, u, pi;

   int seed= 100;
   srand(seed);

   n=10000;

   for(i=0; i<n; i++){
      x = (double)(rand()+1)/(double)(RAND_MAX+2);
      y = (double)(rand()+1)/(double)(RAND_MAX+2);
      u=x*x+y*y;
      if(u<1)  count+=1;
   }

   pi=4.0*count/(double)n;

   printf("pi=%f\n", pi);

   return 0;
}
```

実行結果

```
pi=3.156800
```