シミュレーション

4 年電子情報工学科
34 番 横前洸佑

提出日：2020/2/18（火）
提出期限：2020/2/17（月）10:00

1　課題 11

課題 11 では、ガウスの消去法を用いて連立方程式 (1) を解き、x_1, x_2, x_3, x_4 を求める。

$$
\begin{aligned}
x_1 + 2x_2 + x_3 + 5x_4 = 20.5 \\
8x_1 + x_2 + 3x_3 + x_4 = 14.5 \\
x_1 + 7x_2 + x_3 + x_4 = 18.5 \\
x_1 + x_2 + 6x_3 + x_4 = 9.0
\end{aligned}
\tag{1}
$$

1.1　作成したプログラム

今回作成したプログラムをソースコード 1 に示す。

ソースコード 1　課題 11 のプログラム

```c
#include<stdio.h>
#include<math.h>

void gauss(void);
void print_eq(void);

double result[4];
double eq[4][5] = {
    {1, 2, 1, 5, 20.5},
    {8, 1, 3, 1, 14.5},
    {1, 7, 1, 1, 18.5},
    {1, 1, 6, 1, 9.0}
};

int main(void)
{
    gauss();
    printf("\n結果\n");
    for(int i = 0; i < 4; i++)
        printf("x[%d] = %f\n", i + 1, result[i]);

    return (0);
}

void gauss(void)
{
    //前進削除
    for(int k = 0; k < 3; k++)
    {
        print_eq();
        printf("\n第%d列の掃き出し\n", k + 1);
        for(int i = k + 1; i < 4; i++)
        {
            double m = eq[i][k] / eq[k][k];
            for(int j = k; j < 5; j++)
            {
                eq[i][j] -= (eq[k][j] * m);
            }
        }
    }

    print_eq();

    //後退代入
    for(int i = 0; i < 4; i++)
    {
        result[i] = eq[i][4];
```

```
    }

    for(int i = 3; i >= 0; i--)
    {
        for(int j = i + 1; j <= 3; j++)
        {
            result[i] -= eq[i][j] * result[j];
        }

        result[i] /= eq[i][i];
    }
}

//二次元配列の内容を表示
void print_eq(void)
{
     for(int i = 0; i < 4; i++)
    {
        for(int j = 0; j < 5; j++)
        {
            printf("%f ", eq[i][j]);
        }
        printf("\n");
    }
}
```

ガウスの消去法の計算部分を関数として用意し、main 関数内から呼び出している。

1.2 プログラムの実行結果

実行結果を以下に示す。

```
1.000000 2.000000 1.000000 5.000000 20.500000
8.000000 1.000000 3.000000 1.000000 14.500000
1.000000 7.000000 1.000000 1.000000 18.500000
1.000000 1.000000 6.000000 1.000000 9.000000

第 1 列の掃き出し
1.000000 2.000000 1.000000 5.000000 20.500000
0.000000 -15.000000 -5.000000 -39.000000 -149.500000
0.000000 5.000000 0.000000 -4.000000 -2.000000
0.000000 -1.000000 5.000000 -4.000000 -11.500000

第 2 列の掃き出し
1.000000 2.000000 1.000000 5.000000 20.500000
0.000000 -15.000000 -5.000000 -39.000000 -149.500000
0.000000 0.000000 -1.666667 -17.000000 -51.833333
0.000000 0.000000 5.333333 -1.400000 -1.533333
```

```
第3列の掃き出し
1.000000 2.000000 1.000000 5.000000 20.500000
0.000000 -15.000000 -5.000000 -39.000000 -149.500000
0.000000 0.000000 -1.666667 -17.000000 -51.833333
0.000000 0.000000 0.000000 -55.800000 -167.400000

結果
x[1] = 1.000000
x[2] = 2.000000
x[3] = 0.500000
x[4] = 3.000000
```

1.3 考察

元の方程式に計算によって得た値を代入する。

$$\begin{aligned}
&1.0 + 2 \times 2.0 + 0.5 + 5 \times 3.0 = 20.5 \\
&8 \times 1.0 + 2.0 + 3 \times 0.5 + 3.0 = 14.5 \\
&1.0 + 7 \times 2.0 + 0.5 + 3.0 = 18.5 \\
&1.0 + 2.0 + 6 \times 0.5 + 3.0 = 9.0
\end{aligned} \tag{2}$$

式がすべて成り立つ。この結果よりプログラムは正しく動作していると言える。

2 課題 12

課題 12 では、ガウスの消去法を用いて連立方程式 (3) を解き、x_1, x_2, x_3, x_4 を求める。また、ピボット選択なしで、float 型で変数を宣言した場合と double 型で変数を宣言した場合について解いた時、ピボット選択ありで、float 型で変数を宣言した場合と double 型で変数を宣言した場合について解いた時の結果の違いを考察する。

$$\begin{aligned}
&1.0x_1 + 0.96x_2 + 0.84x_3 + 0.64x_4 = 3.44 \\
&0.96x_1 + 0.9214x_2 + 0.4406x_3 + 0.2222x_4 = 2.5442 \\
&0.84x_1 + 0.4406x_2 + 1.0x_3 + 0.3444x_4 = 2.6250 \\
&0.64x_1 + 0.2222x_2 + 0.3444x_3 + 1.0x_4 = 2.2066
\end{aligned} \tag{3}$$

2.1 作成したプログラム

今回作成したプログラムをソースコード 2 に示す。

ソースコード 2　課題 12 のプログラム

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

void gauss(void);
void print_eq(void);
void pivoting(int);

double result[4];
double eq[4][5] = {
    {1.0, 0.96, 0.84, 0.64, 3.44},
    {0.96, 0.9214, 0.4406, 0.2222, 2.5442},
    {0.84, 0.4406, 1.0, 0.3444, 2.6250},
    {0.64, 0.2222, 0.3444, 1.0, 2.2066}
    };

int main(void)
{
    gauss();
    printf("\n結果\n");
    for(int i = 0; i < 4; i++)
        printf("x[%d] = %f\n", i + 1, result[i]);

    return (0);
}


void gauss(void)
{
    //前進削除
    for(int k = 0; k < 3; k++)
    {
        print_eq();
        printf("\n第%d列の掃き出し\n", k + 1);
        for(int i = k + 1; i < 4; i++)
        {
            pivoting(k);
            double m = eq[i][k] / eq[k][k];
            for(int j = k; j < 5; j++)
            {
                eq[i][j] -= (eq[k][j] * m);
            }
        }
    }

    print_eq();

    //後退代入
    for(int i = 0; i < 4; i++)
    {
        result[i] = eq[i][4];
    }

    for(int i = 3; i >= 0; i--)
    {
        for(int j = i + 1; j <= 3; j++)
        {
            result[i] -= eq[i][j] * result[j];
        }

        result[i] /= eq[i][i];
    }
}


//二次元配列の内容を表示
void print_eq(void)
{
     for(int i = 0; i < 4; i++)
    {
        for(int j = 0; j < 5; j++)
        {
```

```
72                 printf("%f ", eq[i][j]);
73             }
74             printf("\n");
75         }
76 }
77 
78 //ピボット選択
79 void pivoting(int k)
80 {
81     if(eq[k][k] == 0.0)
82     {
83         printf("a_kkが0のためピボット選択を実行します。\n");
84         //最大値検索
85         int max_i;
86         for(int i = k; i <= (3 - k); i++)
87         {
88             double tmp = eq[i][k];
89             if(tmp < eq[i + 1][k])
90             {
91                 tmp = eq[i + 1][k];
92                 max_i = i + 1;
93             }
94         }
95 
96         //列交換
97         for(int i = 0; i < 5; i++)
98         {
99             double tmp = eq[k][i];
100            eq[k][i] = eq[max_i][i];
101            eq[max_i][i] = tmp;
102        }
103    }
104 }
```

ガウスの消去法の計算部分を関数として用意し、main 関数内から呼び出している。また、gauss() 関数内 36 行目でピボット選択を行う。

2.2 プログラムの実行結果

ピボット選択なしで、float 型で実行した結果を以下に示す。

```
1.000000 0.960000 0.840000 0.640000 3.440000
0.960000 0.921400 0.440600 0.222200 2.544200
0.840000 0.440600 1.000000 0.344400 2.625000
0.640000 0.222200 0.344400 1.000000 2.206600

第 1 列の掃き出し
1.000000 0.960000 0.840000 0.640000 3.440000
0.000000 -0.000200 -0.365800 -0.392200 -0.758200
0.000000 -0.365800 0.294400 -0.193200 -0.264600
0.000000 -0.392200 -0.193200 0.590400 0.005000
```

```
第 2 列の掃き出し
1.000000 0.960000 0.840000 0.640000 3.440000
0.000000 -0.000200 -0.365800 -0.392200 -0.758200
0.000000 0.000000 669.430725 717.235229 1386.666016
0.000000 0.000000 717.235229 769.796082 1487.031372

第 3 列の掃き出し
1.000000 0.960000 0.840000 0.640000 3.440000
0.000000 -0.000200 -0.365800 -0.392200 -0.758200
0.000000 0.000000 669.430725 717.235229 1386.666016
0.000000 0.000000 0.000000 1.342590 1.342651

結果
x[1] = 1.000012
x[2] = 1.000000
x[3] = 0.999951
x[4] = 1.000045
```

ピボット選択なしで、double 型で実行した結果を以下に示す。

```
1.000000 0.960000 0.840000 0.640000 3.440000
0.960000 0.921400 0.440600 0.222200 2.544200
0.840000 0.440600 1.000000 0.344400 2.625000
0.640000 0.222200 0.344400 1.000000 2.206600

第 1 列の掃き出し
1.000000 0.960000 0.840000 0.640000 3.440000
0.000000 -0.000200 -0.365800 -0.392200 -0.758200
0.000000 -0.365800 0.294400 -0.193200 -0.264600
0.000000 -0.392200 -0.193200 0.590400 0.005000

第 2 列の掃き出し
1.000000 0.960000 0.840000 0.640000 3.440000
0.000000 -0.000200 -0.365800 -0.392200 -0.758200
```

```
0.000000 0.000000 669.342600 717.140600 1386.483200
0.000000 0.000000 717.140600 769.694600 1486.835200

第 3 列の掃き出し
1.000000 0.960000 0.840000 0.640000 3.440000
0.000000 -0.000200 -0.365800 -0.392200 -0.758200
0.000000 0.000000 669.342600 717.140600 1386.483200
0.000000 0.000000 0.000000 1.342727 1.342727

結果
x[1] = 1.000000
x[2] = 1.000000
x[3] = 1.000000
x[4] = 1.000000
```

ピボット選択ありで、float 型で実行した結果を以下に示す。

```
1.000000 0.960000 0.840000 0.640000 3.440000
0.960000 0.921400 0.440600 0.222200 2.544200
0.840000 0.440600 1.000000 0.344400 2.625000
0.640000 0.222200 0.344400 1.000000 2.206600

第 1 列の掃き出し
1.000000 0.960000 0.840000 0.640000 3.440000
0.000000 -0.000200 -0.365800 -0.392200 -0.758200
0.000000 -0.365800 0.294400 -0.193200 -0.264600
0.000000 -0.392200 -0.193200 0.590400 0.005000

第 2 列の掃き出し
1.000000 0.960000 0.840000 0.640000 3.440000
0.000000 -0.000200 -0.365800 -0.392200 -0.758200
0.000000 0.000000 669.430725 717.235229 1386.666016
0.000000 0.000000 717.235229 769.796082 1487.031372
```

```
第3列の掃き出し
1.000000 0.960000 0.840000 0.640000 3.440000
0.000000 -0.000200 -0.365800 -0.392200 -0.758200
0.000000 0.000000 669.430725 717.235229 1386.666016
0.000000 0.000000 0.000000 1.342590 1.342651

結果
x[1] = 1.000012
x[2] = 1.000000
x[3] = 0.999951
x[4] = 1.000045
```

ピボット選択ありで、double 型で実行した結果を以下に示す。

```
1.000000 0.960000 0.840000 0.640000 3.440000
0.960000 0.921400 0.440600 0.222200 2.544200
0.840000 0.440600 1.000000 0.344400 2.625000
0.640000 0.222200 0.344400 1.000000 2.206600

第1列の掃き出し
1.000000 0.960000 0.840000 0.640000 3.440000
0.000000 -0.000200 -0.365800 -0.392200 -0.758200
0.000000 -0.365800 0.294400 -0.193200 -0.264600
0.000000 -0.392200 -0.193200 0.590400 0.005000

第2列の掃き出し
1.000000 0.960000 0.840000 0.640000 3.440000
0.000000 -0.000200 -0.365800 -0.392200 -0.758200
0.000000 0.000000 669.342600 717.140600 1386.483200
0.000000 0.000000 717.140600 769.694600 1486.835200

第3列の掃き出し
1.000000 0.960000 0.840000 0.640000 3.440000
0.000000 -0.000200 -0.365800 -0.392200 -0.758200
```

```
0.000000 0.000000 669.342600 717.140600 1386.483200
0.000000 0.000000 0.000000 1.342727 1.342727

結果
x[1] = 1.000000
x[2] = 1.000000
x[3] = 1.000000
x[4] = 1.000000
```

2.3　考察

元の方程式に計算によって得た値を代入する。

$$\begin{aligned}
&1.0 + 0.96 + 0.84 + 0.64 = 3.44 \\
&0.96 + 0.9214 + 0.4406 + 0.2222 = 2.5442 \\
&0.84 + 0.4406 + 1.0 + 0.3444 = 2.6250 \\
&0.64 + 0.2222 + 0.3444 + 1.0 = 2.2066
\end{aligned} \tag{4}$$

式がすべて成り立つ。この結果よりプログラムは正しく動作していると言える。

3　課題 13

表 1 に示す 7 組のデータに対して 2 次式で近似を行うプログラムを作成する。

表 1　データ

i	1	2	3	4	5	6	7
x_i	0.0	0.1	0.2	0.3	0.4	0.5	0.6
y_i	0.000	0.034	0.138	0.282	0.479	0.724	1.120

3.1　作成したプログラム

今回作成したプログラムをソースコード 3 に示す。

ソースコード 3　課題 13 のプログラム

```c
#include<stdio.h>
#include<math.h>

void print_eq(void);
void pivoting(int);

```

```c
#define DATA_NUM 7

double data[3][DATA_NUM] = {
    {1, 2, 3, 4, 5, 6, 7},   //i
    {0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6},   //x_i
    {0.000, 0.034, 0.138, 0.282, 0.479, 0.724, 1.120}    //y_i
};

double eq[3][4] = {0.0};

int main(void)
{
    eq[0][0] = DATA_NUM;
    for(int i = 0; i < DATA_NUM; i++)
    {
        eq[0][1] += data[1][i];
        eq[0][2] += pow(data[1][i], 2.0);
        eq[0][3] += data[2][i];
        eq[1][0] += data[1][i];
        eq[1][1] += pow(data[1][i], 2.0);
        eq[1][2] += pow(data[1][i], 3.0);
        eq[1][3] += data[1][i] * data[2][i];
        eq[2][0] += pow(data[1][i], 2.0);
        eq[2][1] += pow(data[1][i], 3.0);
        eq[2][2] += pow(data[1][i], 4.0);
        eq[2][3] += pow(data[1][i], 2.0) * data[2][i];
    }

    //前進削除
    for(int k = 0; k < 2; k++)
    {
        print_eq();
        printf("\n第%d列の掃き出し\n", k + 1);

        for(int i = k + 1; i < 3; i++)
        {
            pivoting(k);
            double m = eq[i][k] / eq[k][k];
            for(int j = k; j < 4; j++)
            {
                eq[i][j] -= (eq[k][j] * m);
            }
        }
    }

    print_eq();

    //後退代入

    double result[3] = {eq[0][3], eq[1][3], eq[2][3]};

    for(int i = 2; i >= 0; i--)
    {
        for(int j = i + 1; j <= 2; j++)
        {
            result[i] -= eq[i][j] * result[j];
        }

        result[i] /= eq[i][i];
    }

    printf("\n結果\n");
    for(int i = 0; i < 3; i++)
        printf("x^%d = %f\n", i, result[i]);

    return (0);
}


//二次元配列の内容を表示
void print_eq(void)
{
     for(int i = 0; i < 3; i++)
```

```
    {
        for(int j = 0; j < 4; j++)
        {
            printf("%f ", eq[i][j]);
        }
        printf("\n");
    }
}

//ピボット選択
void pivoting(int k)
{
    if(eq[k][k] == 0.0)
    {
        printf("a_kkが0のためピボット選択を実行します。\n");
        //最大値検索
        int max_i;
        for(int i = k; i <= (2 - k); i++)
        {
            double tmp = eq[i][k];
            if(tmp < eq[i + 1][k])
            {
                tmp = eq[i + 1][k];
                max_i = i + 1;
            }
        }

        //列交換
        for(int i = 0; i < 4; i++)
        {
            double tmp = eq[k][i];
            eq[k][i] = eq[max_i][i];
            eq[max_i][i] = tmp;
        }
    }
}
```

3.2 プログラムの実行結果

プログラムの実行結果を以下に示す。

```
7.000000 2.100000 0.910000 2.777000
2.100000 0.910000 0.441000 1.341200
0.910000 0.441000 0.227500 0.692080

第1列の掃き出し
7.000000 2.100000 0.910000 2.777000
0.000000 0.280000 0.168000 0.508100
0.000000 0.168000 0.109200 0.331070

第2列の掃き出し
7.000000 2.100000 0.910000 2.777000
0.000000 0.280000 0.168000 0.508100
```

```
0.000000 0.000000 0.008400 0.026210

結果
x^0 = 0.008333
x^1 = -0.057500
x^2 = 3.120238
```

$y = a + bx + cx^2$ の場合、a=0.008333, b=-0.0575, c=3.1202 となっているため正しく動作している。

4　課題 14-1

1 次元ランダムウォークのシミュレーションを行うプログラムを作成する。なお右へ動く確率を、p=0.5 の場合と、p=0.7 の場合について計算する。

4.1　作成したプログラム

今回作成したプログラムをソースコード 4 に示す。

ソースコード 4　課題 14-1 のプログラム

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

#define SAMPLE_CNT 10
#define DATA_NUM 10

void print_eq(void);
void pivoting(int);

double data[2][10] = {0.0};
double eq[3][4] = {0.0};

int main(void){
    double ave_xN = 0;
    double pow2_xN = 0;
    double bunsan = 0;
    double sum_xy = 0, sum_x = 0, sum_y = 0, sum_x_pow2 = 0;
    double a = 0, b = 0;
    const int STEP = 500;

    srand((unsigned int)time(NULL));

    printf("STEP, bunsan\n");
    for(int N = 1; N <= 10; N++){

        for(int k = 0; k < 10; k++){
            double sum_xN = 0, sum_pow2_xN = 0;
            for(int j = 0; j < SAMPLE_CNT; j++){
                int xN = 0;

                for(int i = 0; i < STEP * N; i++){
```

```
                    int a = rand() % 2;
                    if(a == 1){
                        xN++;
                    }else{
                        xN--;
                    }
                }

                //位置の和
                sum_xN += xN;
                //位置の2乗 の 和
                sum_pow2_xN += xN * xN;
            }

            //位置の平均
            ave_xN += sum_xN / SAMPLE_CNT;
            //位置の2乗 の 平 均
            pow2_xN += sum_pow2_xN / SAMPLE_CNT;
            //分散の和
            bunsan += (pow2_xN - pow(ave_xN, 2.0));
        }

        //分散の平均
        bunsan /= SAMPLE_CNT;
        printf("%d, %f\n", STEP * N, bunsan);
        data[0][N - 1] = STEP * N;
        data[1][N - 1] = bunsan;

        sum_xy += STEP * N * bunsan;
        sum_x += STEP * N;
        sum_y += bunsan;
        sum_x_pow2 += pow(STEP * N, 2);
    }

    a = (10 * sum_xy - sum_x * sum_y) / (10 * sum_x_pow2 - pow(sum_x, 2));
    b = (sum_x_pow2 * sum_y - sum_xy * sum_x) / (10 * sum_x_pow2 - pow(sum_x, 2));

    printf("一次近似\na = %f\nb = %f\n\n", a, b);


    eq[0][0] = DATA_NUM;
    for(int i = 0; i < DATA_NUM; i++)
    {
        eq[0][1] += data[0][i];
        eq[0][2] += pow(data[0][i], 2.0);
        eq[0][3] += data[1][i];
        eq[1][0] += data[0][i];
        eq[1][1] += pow(data[0][i], 2.0);
        eq[1][2] += pow(data[0][i], 3.0);
        eq[1][3] += data[0][i] * data[1][i];
        eq[2][0] += pow(data[0][i], 2.0);
        eq[2][1] += pow(data[0][i], 3.0);
        eq[2][2] += pow(data[0][i], 4.0);
        eq[2][3] += pow(data[0][i], 2.0) * data[1][i];
    }

    //前進削除
    for(int k = 0; k < 2; k++)
    {
        print_eq();
        printf("\n第%d列の掃き出し\n", k + 1);

        for(int i = k + 1; i < 3; i++)
        {
            pivoting(k);
            double m = eq[i][k] / eq[k][k];
            for(int j = k; j < 4; j++)
            {
                eq[i][j] -= (eq[k][j] * m);
            }
        }
    }
```

```
    print_eq();

    //後退代入

    double result[3] = {eq[0][3], eq[1][3], eq[2][3]};

    for(int i = 2; i >= 0; i--)
    {
        for(int j = i + 1; j <= 2; j++)
        {
            result[i] -= eq[i][j] * result[j];
        }

        result[i] /= eq[i][i];
    }

    printf("\n二次近似\n");
    for(int i = 0; i < 3; i++)
        printf("x^%d = %f\n", i, result[i]);

    return 0;
}



//二次元配列の内容を表示
void print_eq(void)
{
     for(int i = 0; i < 3; i++)
    {
        for(int j = 0; j < 4; j++)
        {
            printf("%f ", eq[i][j]);
        }
        printf("\n");
    }
}

//ピボット選択
void pivoting(int k)
{
    if(eq[k][k] == 0.0)
    {
        printf("a_kkが0のためピボット選択を実行します。\n");
        //最大値検索
        int max_i;
        for(int i = k; i <= (2 - k); i++)
        {
            double tmp = eq[i][k];
            if(tmp < eq[i + 1][k])
            {
                tmp = eq[i + 1][k];
                max_i = i + 1;
            }
        }

        //列交換
        for(int i = 0; i < 4; i++)
        {
            double tmp = eq[k][i];
            eq[k][i] = eq[max_i][i];
            eq[max_i][i] = tmp;
        }
    }
}
```

4.2 プログラムの実行結果

p=0.5 の場合のプログラムの実行結果を以下に示す。

```
STEP, bunsan
500, 3091.340000
1000, 11463.898000
1500, 26258.353800
2000, 46493.031380
2500, 71557.331138
3000, 94939.401114
3500, 118096.692111
4000, 165836.233211
4500, 217854.507321
5000, 267024.838732
一次近似
a = 57.651130
b = -56279.044088

10.000000 27500.000000 96250000.000000 1022615.626808
27500.000000 96250000.000000 378125000000.000000 4001247524.486351
96250000.000000 378125000000.000000 1583312500000000.000000 16746217035504.652344

第 1 列の掃き出し
10.000000 27500.000000 96250000.000000 1022615.626808
0.000000 20625000.000000 113437500000.000000 1189054550.765607
0.000000 113437500000.000000 656906250000000.000000 6903541627482.048828

第 2 列の掃き出し
10.000000 27500.000000 96250000.000000 1022615.626808
0.000000 20625000.000000 113437500000.000000 1189054550.765607
0.000000 0.000000 33000000000000.000000 363741598271.208008

二次近似
x^0 = 4344.555624
```

```
x^1 = -2.972470
x^2 = 0.011022
```

p=0.7 の場合のプログラムの実行結果を以下に示す。

```
STEP, bunsan
500, -1292039.572000
1000, -17521748.637200
1500, -85962996.179720
2000, -271856508.621972
2500, -661073320.182197
3000, -1375414664.946219
3500, -2555708081.218623
4000, -4377754633.161863
4500, -7035324404.184189
5000, -10746941452.346424
一次近似
a = -2119519.000621
b = 3115792266.803739

10.000000 27500.000000 96250000.000000 -27128849849.050407
27500.000000 96250000.000000 378125000000.000000 -118319416472704.468750
96250000.000000 378125000000.000000 1583312500000000.000000 -530299481388985536.0

第 1 列の掃き出し
10.000000 27500.000000 96250000.000000 -27128849849.050407
0.000000 20625000.000000 113437500000.000000 -43715079387815.843750
0.000000 113437500000.000000 656906250000000.000000 -269184301591875360.000000

第 2 列の掃き出し
10.000000 27500.000000 96250000.000000 -27128849849.050407
0.000000 20625000.000000 113437500000.000000 -43715079387815.843750
0.000000 0.000000 33000000000000.000000 -28751364958888224.000000
```

```
二次近似
x^0 = -1676101893.010965
x^1 = 2672375.159193
x^2 = -871.253484
```

4.3　考察

p=0.5 の場合において、横軸をステップ数、縦軸を分散とし、分散のデータ値、一次直線、二次曲線を同一のグラフにプロットすると、図 1 のようになる。

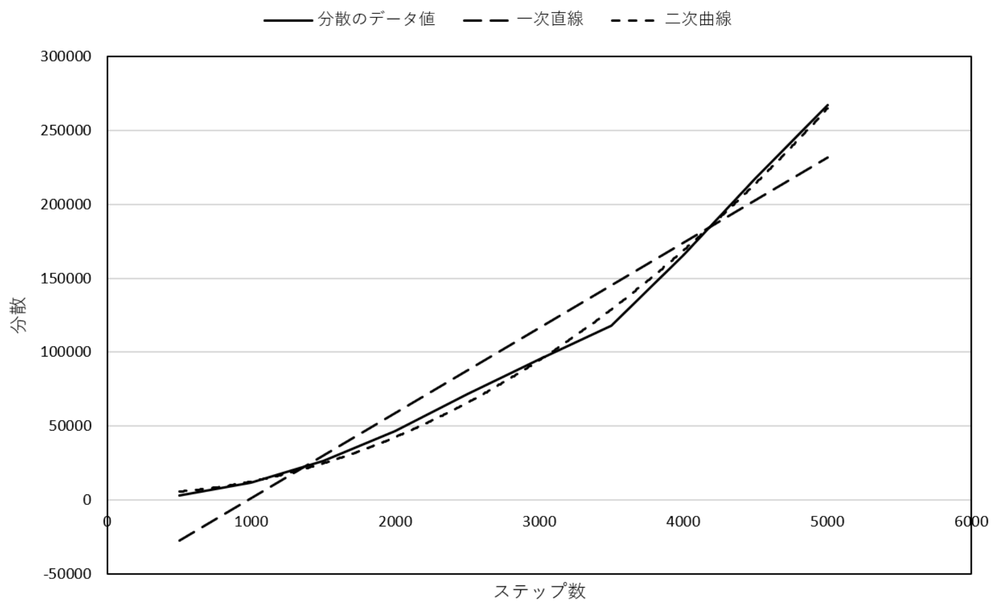

図 1　$p = 0.5$ の場合

グラフより、単調増加していることがわかる。

p=0.7 の場合においてグラフをプロットすると、図 2 のようになる。

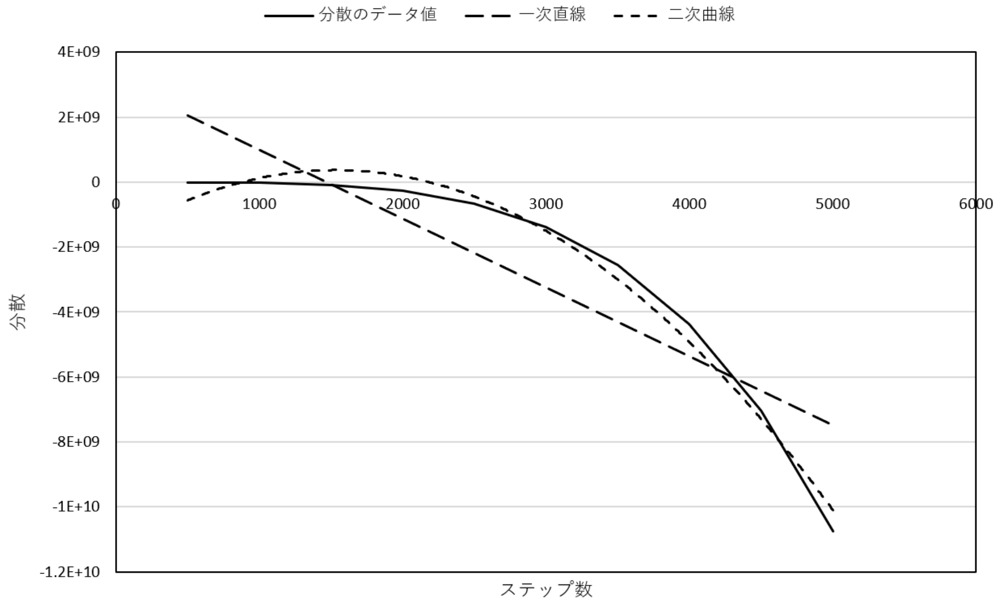

図 2　$p = 0.7$ の場合

グラフより、分散は急激に減少することがわかる。

5　課題 14-2

2 次元ランダムウォークのシミュレーションを行うプログラムを作成する。粒子を 200 個用意し、N ステップ後にどのような模様になるか調べる。今回は、N=500 とする。

5.1　作成したプログラム

今回作成したプログラムをソースコード 5 に示す。

ソースコード 5　課題 14-2 のプログラム

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

#define STEP 500

int main(void){
    srand((unsigned int)time(NULL));

    printf("x, y\n");

    for(int j = 0; j < 200; j++){
        int xN = 0;
        int yN = 0;

        for(int i = 0; i < STEP; i++){
            int a = rand() % 4;

            if(a == 0){
                xN++;
```

```
            }else if(a == 1){
                xN--;
            }else if(a == 2){
                yN++;
            }else if(a == 3){
                yN--;
            }
        }
        printf("%d, %d\n", xN, yN);
    }

    return 0;
}
```

5.2 プログラムの実行結果

プログラムの実行結果を以下に示す。なお、量が多いため、一部のみを掲載する。

```
x, y
11, 19
-17, -19
4, -14
-26, -20
9, 17
9, 7
14, 28
-10, 32
26, -16
-27, -5
13, -9
4, -26
5, 9
-6, 16
4, -10
7, 9
-13, 17
-28, -14
9, -25
-6, -28
16, -14
-10, 40
```

```
-18, -24
-21, 9
6, -14
-20, -12
-28, 2
34, 2
16, 10
11, 13
```

5.3　考察

各粒子を配置すると図 3 のようになる。

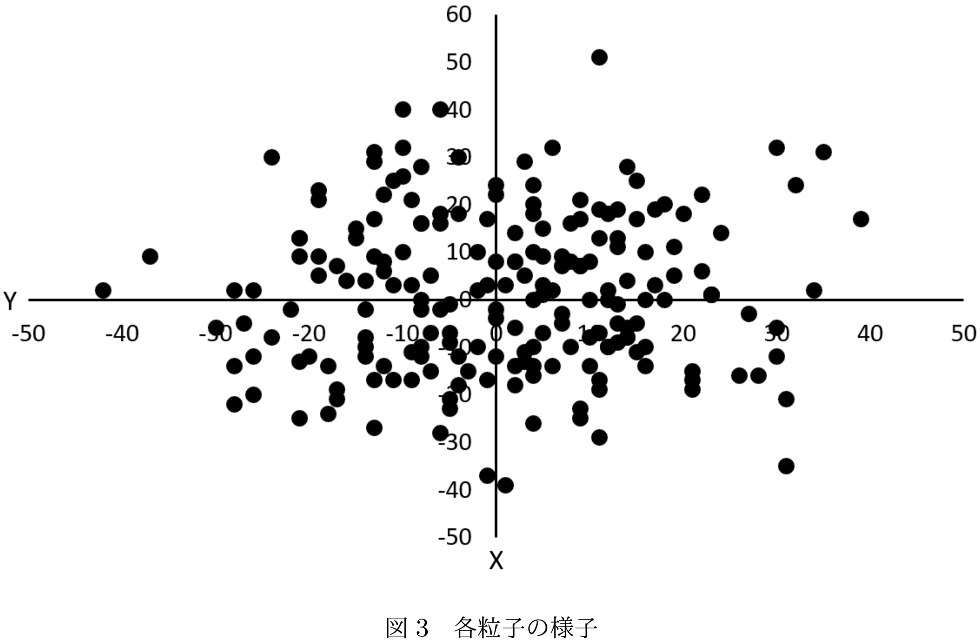

図 3　各粒子の様子

各粒子が 1 匹の蜂と考えた時、蜂の群の境界は、円のようになる。