シミュレーション
—連立方程式と最小二乗法—

学籍番号：16426
4 年 電子情報工学科 23 番
福澤 大地

提出日：2020 年 2 月 15 日

1　目的

ガウスの消去法を用いて、連立一次方程式の解を求めるプログラムを作成する。その中で、ピボット選択などの工夫により精度を向上させる手法を検討する。

さらに最小二乗法により、一次、二次で近似した方程式を導き、ランダムウォークのステップ数と分散の関係などを考察する。

2　開発環境

プログラムの開発、実行を行った環境を表 1 に示す。

表 1　開発環境

CPU	Intel Core i5–7400 @ 3.0GHz
メモリ	8GB
OS	Microsoft Windows 10 Home
システム	64bit
コンパイラ	GCC 7.4.0

3　課題 11：ガウスの消去法

3.1　課題内容

式 (1) の連立方程式をガウスの消去法で解くプログラムを作成する。

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

3.2　プログラムリスト

課題 11 のプログラムリストをリスト 1 に示す。

リスト 1　課題 11 のプログラム

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

#define N 4

void gauss(double[N][N + 1]);
void dispMatrix(double[N][N + 1]);

int main(void) {
    double X[N][N + 1] = {
```

```c
        {1, 2, 1, 5, 20.5},
        {8, 1, 3, 1, 14.5},
        {1, 7, 1, 1, 18.5},
        {1, 1, 6, 1, 9.0}
    };

    gauss(X);
    dispMatrix(X);

    return 0;
}

// ガウスの消去法
void gauss(double X[N][N + 1]) {
    int i, j, k;

    // 前進消去
    for (i = 0; i < N; i++) {
        double temp;

        temp = X[i][i];
        for (j = 0; j < N + 1; j++)
            X[i][j] /= temp;

        for (j = i + 1; j < N; j++) {
            temp = X[j][i];
            for (k = 0; k < N + 1; k++)
                X[j][k] -= X[i][k] * temp;
        }
    }

    // 後退代入
    for (i = N - 1; i >= 1; i--) {
        for (j = i; j < N; j++) {
            X[i - 1][N] -= X[j][N] * X[i - 1][j];
            X[i - 1][j] = 0;
        }
    }
}

// 行列を表示
void dispMatrix(double X[N][N + 1]) {
    int i, j;

    for (i = 0; i < N; i++) {
        for (j = 0; j < N + 1; j++) {
            if (j != N)
                printf("%6.2f ", X[i][j]);
            else
                printf("| %6.2f\n", X[i][j]);
        }
    }
}
```

3.3　実行結果

課題 11 の実行結果をリスト 2 に示す。リスト 2 より、$x_1 = 1.0$, $x_2 = 2.0$, $x_2 = 0.5$, $x_3 = 3.0$ であることが分かる。

リスト 2　課題 11 の実行結果

```
 1.00   0.00   0.00   0.00 |   1.00
-0.00   1.00   0.00   0.00 |   2.00
-0.00  -0.00   1.00   0.00 |   0.50
-0.00  -0.00  -0.00   1.00 |   3.00
```

3.4　考察

プログラムによって得られた結果を式 (1) に代入すると、式 (2)–(5) のようになる。いずれも元の方程式を満たしているので、正しい実行結果が得られた。

$$\begin{aligned} &x_1 + 2x_2 + x_3 + 5x_4 \\ =&1 \times 1.0 + 2 \times 2.0 + 1 \times 0.5 + 5 \times 3.0 \\ =&20.5 \end{aligned} \tag{2}$$

$$\begin{aligned} &x_1 + 2x_2 + x_3 + 5x_4 \\ =&8 \times 1.0 + 1 \times 2.0 + 3 \times 0.5 + 1 \times 3.0 \\ =&14.5 \end{aligned} \tag{3}$$

$$\begin{aligned} &x_1 + 2x_2 + x_3 + 5x_4 \\ =&1 \times 1.0 + 7 \times 2.0 + 1 \times 0.5 + 1 \times 3.0 \\ =&18.5 \end{aligned} \tag{4}$$

$$\begin{aligned} &x_1 + 2x_2 + x_3 + 5x_4 \\ =&1 \times 1.0 + 1 \times 2.0 + 6 \times 0.5 + 1 \times 3.0 \\ =&9.0 \end{aligned} \tag{5}$$

4　課題 12：ピボット選択

4.1　課題内容

式 (6) の連立方程式をガウスの消去法で解くプログラムを作成する。ピボット選択あり/なしそれぞれの場合に対して、float 型/double 型で演算を行い、結果の違いについて考察を行う。

$$
\begin{cases}
1.0x_1 + 0.96x_2 + 0.84x_3 + 0.64x_4 & = 3.44 \\
0.96x_1 + 0.9214x_2 + 0.4406x_3 + 0.2222x_4 & = 2.5442 \\
0.84x_1 + 0.4406x_2 + 1.0x_3 + 0.3444x_4 & = 2.6250 \\
0.64x_1 + 0.2222x_2 + 0.3444x_3 + 1.0x_4 & = 2.2066
\end{cases}
\tag{6}
$$

4.2 プログラムリスト

課題 12 のプログラムリストをリスト 3 に示す。

6 行目をコメントアウトすると、ピボット選択をしなくなり、7 行目をコメントアウトすると、float 型で演算を行うようになる。

リスト 3　課題 12 のプログラム

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4  #include <float.h>
5  
6  #define PIVOT
7  #define DOUBLE
8  
9  #define N 4
10 
11 #ifdef DOUBLE
12     typedef double type;
13 #else
14     typedef float type;
15 #endif
16 
17 void gauss(type[N][N + 1]);
18 void dispMatrix(type[N][N + 1]);
19 
20 int main(void) {
21     type X[N][N + 1] = {
22         {1.0,  0.96,   0.84,   0.64,   3.44},
23         {0.96, 0.9214, 0.4406, 0.2222, 2.5442},
24         {0.84, 0.4406, 1.0,    0.3444, 2.6250},
25         {0.64, 0.2222, 0.3444, 1.0,    2.2066}
26     };
27 
28     gauss(X);
29     dispMatrix(X);
30 
31     return 0;
32 }
33 
34 // ガウスの消去法
35 void gauss(type X[N][N + 1]) {
36     int i, j, k;
37 
38     // 前進消去
39     for (i = 0; i < N; i++) {
40         type temp;
41 
```

```c
#ifdef PIVOT
        // ピボット選択
        for (j = i; j < N; j++) {
            if (X[j][i] > X[i][i]) {
                for (k = 0; k < N + 1; k++) {
                    temp = X[i][k];
                    X[i][k] = X[j][k];
                    X[j][k] = temp;
                }
            }
        }
#endif

        temp = X[i][i];
        for (j = 0; j < N + 1; j++)
            X[i][j] /= temp;

        for (j = i + 1; j < N; j++) {
            temp = X[j][i];
            for (k = 0; k < N + 1; k++)
                X[j][k] -= X[i][k] * temp;
        }
    }

    // 後退代入
    for (i = N - 1; i >= 1; i--) {
        for (j = i; j < N; j++) {
            X[i - 1][N] -= X[j][N] * X[i - 1][j];
            X[i - 1][j] = 0;
        }
    }
}

// 行列を表示
void dispMatrix(type X[N][N + 1]) {
    int i, j;

    for (i = 0; i < N; i++) {
        for (j = 0; j < N + 1; j++) {
            if (j != N)
                printf("%6.2f ", X[i][j]);
            else
                printf("| %24.20f\n", X[i][j]);
        }
    }
}
```

4.3 実行結果

ピボット選択なし/float 型の場合の実行結果をリスト 4, ピボット選択なし/double 型の場合の実行結果をリスト 5, ピボット選択あり/float 型の場合の実行結果をリスト 6, ピボット選択あり/double 型の場合の実行結果をリスト 7 に示す。

リスト 4　ピボット選択なし/float 型の実行結果

```
 1.00   0.00   0.00   0.00 |   1.00003778934478759766
-0.00   1.00   0.00   0.00 |   0.99998271465301513672
 0.00   0.00   1.00   0.00 |   0.99991285800933837891
 0.00   0.00   0.00   1.00 |   1.00008165836334228516
```

リスト 5　ピボット選択なし/double 型の実行結果

```
 1.00   0.00   0.00   0.00 |   1.00000000000014521717
-0.00   1.00   0.00   0.00 |   0.99999999999981215026
 0.00   0.00   1.00   0.00 |   1.00000000000014499513
 0.00   0.00   0.00   1.00 |   0.99999999999986455279
```

リスト 6　ピボット選択あり/float 型の実行結果

```
 1.00   0.00   0.00   0.00 |   1.00024843215942382812
-0.00   1.00   0.00   0.00 |   0.99976122379302978516
 0.00   0.00   1.00   0.00 |   0.99992024898529052734
-0.00  -0.00  -0.00   1.00 |   1.00007486343383789062
```

リスト 7　ピボット選択あり/double 型の実行結果

```
 1.00   0.00   0.00   0.00 |   0.99999999999990685229
-0.00   1.00   0.00   0.00 |   1.00000000000005018208
 0.00   0.00   1.00   0.00 |   1.00000000000018673951
-0.00  -0.00  -0.00   1.00 |   0.99999999999982547294
```

4.4　考察

リスト 4–7 を見ると、float 型で演算を行った場合に比べ、double 型で演算を行った場合の方が断然精度良く計算できていることが分かる。

しかし、ピボット選択ありとなしの場合を比較すると、はっきりとした精度の良し悪しは分からない。このことから、ピボット選択の効力を発揮できるかどうかはデータによるということが分かる。

5　課題 13：最小二乗法

5.1　課題内容

表 2 に示すデータが与えられたとき、最小二乗法により近似した二次方程式を求めるプログラムを作成する。

表 2　最小二乗法を行うデータ

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

5.2　プログラムリスト

課題 13 のプログラムリストをリスト 8 に示す。課題 12 で作成したガウスの消去法を行う関数を流用している。

リスト 8　課題 13 のプログラム

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

#define N 7

void quadratic(double[N], double[N], double *, double *, double *);
void gauss(double[3][4]);

int main(void) {
    double a, b, c;
    double x[N] = {0.0,   0.1,   0.2,   0.3,   0.4,   0.5,   0.6};
    double y[N] = {0.000, 0.034, 0.138, 0.282, 0.479, 0.724, 1.120};

    quadratic(x, y, &a, &b, &c);
    printf("y = %.8f x^2 + %.8f x + %.8f\n", a, b, c);

    return 0;
}

// 二次方程式による最小二乗法
void quadratic(double x[N], double y[N], double *a, double *b, double *c) {
    int i;
    double sum_1, sum_x, sum_x2, sum_x3, sum_x4, sum_y, sum_xy, sum_x2y;
    sum_1 = sum_x = sum_x2 = sum_x3 = sum_x4 = sum_y = sum_xy = sum_x2y = 0;

    // 和を計算
    for (i = 0; i < N; i++) {
        sum_1 += 1;
        sum_x += x[i];
        sum_x2 += pow(x[i], 2);
        sum_x3 += pow(x[i], 3);
        sum_x4 += pow(x[i], 4);
        sum_y += y[i];
        sum_xy += x[i] * y[i];
        sum_x2y += pow(x[i], 2) * y[i];
    }

    // ガウスの消去法
    double X[3][4] = {
        {sum_x4, sum_x3, sum_x2, sum_x2y},
        {sum_x3, sum_x2, sum_x,  sum_xy},
        {sum_x2, sum_x,  sum_1,  sum_y}
    };

    gauss(X);

    // 結果を返却
    *a = X[0][3];
```

```c
    *b = X[1][3];
    *c = X[2][3];
}

// ガウスの消去法
void gauss(double X[3][4]) {
    int i, j, k;

    // 前進消去
    for (i = 0; i < 3; i++) {
        double temp;

        // ピボット選択
        for (j = i; j < 3; j++) {
            if (X[j][i] > X[i][i]) {
                for (k = 0; k < 4; k++) {
                    temp = X[i][k];
                    X[i][k] = X[j][k];
                    X[j][k] = temp;
                }
            }
        }

        temp = X[i][i];
        for (j = 0; j < 4; j++)
            X[i][j] /= temp;

        for (j = i + 1; j < 3; j++) {
            temp = X[j][i];
            for (k = 0; k < 3 + 1; k++)
                X[j][k] -= X[i][k] * temp;
        }
    }

    // 後退代入
    for (i = 2; i >= 1; i--) {
        for (j = i; j < 3; j++) {
            X[i - 1][3] -= X[j][3] * X[i - 1][j];
            X[i - 1][j] = 0;
        }
    }
}
```

5.3 実行結果

課題 13 の実行結果をリスト 9 に示す。

リスト 9 課題 13 の実行結果

```
y = 3.12023810 x^2 + -0.05750000 x + 0.00833333
```

5.4　考察

表 2 のデータと、実行結果の二次方程式をプロットしたグラフを図 1 に示す。図 1 を見ると、正しく近似できているので、正しい実行結果が得られた。

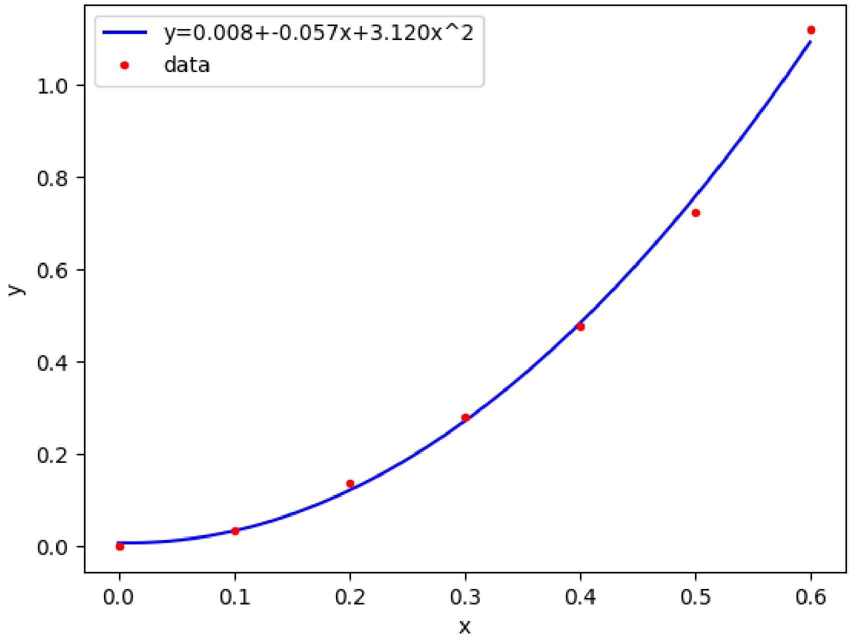

図 1　課題 13 のグラフ

6　課題 14-1：一次元ランダムウォーク

6.1　課題内容

一次元のランダムウォークを行うプログラムを作成し、ステップ数と分散の関係を一次と二次の最小二乗法により調査する。また、正の方向に移動する確率 p を変更した場合、この関係はどのように変わるのか考察する。

6.2　プログラムリスト

課題 14-1 のプログラムリストをリスト 10 に示す。課題 13 で作成した最小二乗法を行う関数を流用している。

リスト 10　課題 14-1 のプログラム

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

#define PLOT_N   50
```

```c
#define DATA_N   100
#define STEP_MAX 5000

#define P 0.5
#define L 1.0

void random_walk(double[], int, double, double);
void linear(double[], double[], int, double *, double *);
void quadratic(double[], double[], int, double *, double *, double *);
void gauss(double[3][4]);
double variance(double[], int);
double covariance(double[], double[], int);
double average(double[], int);

int main(void) {
    int i, j;
    double a, b, c;
    int n[PLOT_N];
    double nd[PLOT_N];
    double x[STEP_MAX + 1] = {0};
    double V[PLOT_N];

    srand(time(NULL));

    // nを決定
    for (i = 0; i < PLOT_N; i++) {
        n[i] = (i + 1) * 100;
        nd[i] = (double)n[i];
    }

    // ランダムウォークの分散を記録
    for (i = 0; i < PLOT_N; i++) {
        double sum = 0;

        for (j = 0; j < DATA_N; j++) {
            random_walk(x, n[i], P, L);
            sum += variance(x + 1, n[i]);
        }

        V[i] = sum / DATA_N;
    }

    // 結果を出力
    for (i = 0; i < PLOT_N; i++)
        printf("(n, V) = (%4d, %11f)\n", n[i], V[i]);
    printf("\n");

    // 最小二乗法
    linear(nd, V, PLOT_N, &a, &b);
    printf("y = %fx + %f\n", a, b);

    quadratic(nd, V, PLOT_N, &a, &b, &c);
    printf("y = %fx^2 + %fx + %f\n", a, b, c);

    return 0;
}
```

```c

// ランダムウォーク
void random_walk(double x[], int n, double p, double l) {
    int i;

    for (i = 0; i < n; i++)
        x[i + 1] = x[i] + ((double)rand() / RAND_MAX < p ? l : -l);
}

// 一次方程式による最小二乗法
void linear(double x[], double y[], int n, double *a, double *b) {
    double V_x = variance(x, n), V_xy = covariance(x, y, n);
    double ave_x = average(x, n), ave_y = average(y, n);

    *a = V_xy / V_x;
    *b = ave_y - *a * ave_x;
}

// 二次方程式による最小二乗法
void quadratic(double x[], double y[], int n, double *a, double *b, double *c) {
    int i;
    double sum_1, sum_x, sum_x2, sum_x3, sum_x4, sum_y, sum_xy, sum_x2y;
    sum_1 = sum_x = sum_x2 = sum_x3 = sum_x4 = sum_y = sum_xy = sum_x2y = 0;

    // 和を計算
    for (i = 0; i < n; i++) {
        sum_1 += 1;
        sum_x += x[i];
        sum_x2 += pow(x[i], 2);
        sum_x3 += pow(x[i], 3);
        sum_x4 += pow(x[i], 4);
        sum_y += y[i];
        sum_xy += x[i] * y[i];
        sum_x2y += pow(x[i], 2) * y[i];
    }

    // ガウスの消去法
    double X[3][4] = {
        {sum_x4, sum_x3, sum_x2, sum_x2y},
        {sum_x3, sum_x2, sum_x,  sum_xy},
        {sum_x2, sum_x,  sum_1,  sum_y}
    };

    gauss(X);

    // 結果を返却
    *a = X[0][3];
    *b = X[1][3];
    *c = X[2][3];
}

// ガウスの消去法
void gauss(double X[3][4]) {
    int i, j, k;

    // 前進消去
```

```
    for (i = 0; i < 3; i++) {
        double temp;

        // ピボット選択
        for (j = i; j < 3; j++) {
            if (X[j][i] > X[i][i]) {
                for (k = 0; k < 4; k++) {
                    temp = X[i][k];
                    X[i][k] = X[j][k];
                    X[j][k] = temp;
                }
            }
        }

        temp = X[i][i];
        for (j = 0; j < 4; j++)
            X[i][j] /= temp;

        for (j = i + 1; j < 3; j++) {
            temp = X[j][i];
            for (k = 0; k < 3 + 1; k++)
                X[j][k] -= X[i][k] * temp;
        }
    }

    // 後退代入
    for (i = 2; i >= 1; i--) {
        for (j = i; j < 3; j++) {
            X[i - 1][3] -= X[j][3] * X[i - 1][j];
            X[i - 1][j] = 0;
        }
    }
}

// 分散
double variance(double x[], int n) {
    return covariance(x, x, n);
}

// 共分散
double covariance(double x[], double y[], int n) {
    int i;
    double sum = 0;
    double ave_x = average(x, n), ave_y = average(y, n);

    for (i = 0; i < n; i++)
        sum += (x[i] - ave_x) * (y[i] - ave_y);

    return sum / n;
}

// 平均
double average(double x[], int n) {
    int i;
    double sum = 0;

```

```c
    for (i = 0; i < n; i++)
        sum += x[i];

    return sum / n;
}
```

6.3 実行結果

$p = 0.5$ のときの実行結果をリスト 11 に、$p = 0.7$ のときの実行結果をリスト 12 に示す。

リスト 11　p0.5 のときの実行結果

```
(n, V) = ( 100,   15.523884)
(n, V) = ( 200,   32.598991)
(n, V) = ( 300,   47.586104)
(n, V) = ( 400,   61.323219)
(n, V) = ( 500,   78.533024)
(n, V) = ( 600,   98.516329)
(n, V) = ( 700,  116.542794)
(n, V) = ( 800,  131.299920)
(n, V) = ( 900,  154.414567)
(n, V) = (1000,  147.934357)
(n, V) = (1100,  189.357535)
(n, V) = (1200,  169.977783)
(n, V) = (1300,  209.623666)
(n, V) = (1400,  226.605759)
(n, V) = (1500,  253.670260)
(n, V) = (1600,  268.832560)
(n, V) = (1700,  279.689249)
(n, V) = (1800,  277.343430)
(n, V) = (1900,  345.269809)
(n, V) = (2000,  333.731190)
(n, V) = (2100,  355.947362)
(n, V) = (2200,  345.883745)
(n, V) = (2300,  421.436965)
(n, V) = (2400,  375.919495)
(n, V) = (2500,  375.621363)
(n, V) = (2600,  404.488618)
(n, V) = (2700,  431.486227)
(n, V) = (2800,  508.233071)
(n, V) = (2900,  448.499493)
(n, V) = (3000,  495.616177)
(n, V) = (3100,  514.785516)
(n, V) = (3200,  524.807244)
(n, V) = (3300,  617.592151)
(n, V) = (3400,  595.879307)
(n, V) = (3500,  519.094713)
```

```
(n, V) = (3600,  614.621791)
(n, V) = (3700,  568.387483)
(n, V) = (3800,  560.936808)
(n, V) = (3900,  695.592825)
(n, V) = (4000,  714.939175)
(n, V) = (4100,  763.579925)
(n, V) = (4200,  718.312947)
(n, V) = (4300,  626.020558)
(n, V) = (4400,  736.293271)
(n, V) = (4500,  762.235101)
(n, V) = (4600,  957.430561)
(n, V) = (4700,  785.787048)
(n, V) = (4800,  870.550813)
(n, V) = (4900,  784.626573)
(n, V) = (5000,  762.585985)

y = 0.170861x + -9.783985
y = 0.000002x^2 + 0.160822x + -1.083632
```

リスト 12　p0.7 のときの実行結果

```
(n, V) = ( 100,  148.665644)
(n, V) = ( 200,  555.972195)
(n, V) = ( 300, 1158.931700)
(n, V) = ( 400, 2180.599354)
(n, V) = ( 500, 3518.090981)
(n, V) = ( 600, 4985.400662)
(n, V) = ( 700, 6607.519793)
(n, V) = ( 800, 8601.677295)
(n, V) = ( 900, 10855.399123)
(n, V) = (1000, 14048.736173)
(n, V) = (1100, 16328.609926)
(n, V) = (1200, 19027.811397)
(n, V) = (1300, 22179.548714)
(n, V) = (1400, 26209.735436)
(n, V) = (1500, 30024.733010)
(n, V) = (1600, 34614.080243)
(n, V) = (1700, 38568.974709)
(n, V) = (1800, 43638.861840)
(n, V) = (1900, 47951.544745)
(n, V) = (2000, 53131.616736)
(n, V) = (2100, 59248.946839)
(n, V) = (2200, 64108.926276)
(n, V) = (2300, 70945.047973)
(n, V) = (2400, 78243.558186)
(n, V) = (2500, 83326.828883)
```

```
(n, V) = (2600, 89868.918849)
(n, V) = (2700, 98516.626332)
(n, V) = (2800, 104277.447543)
(n, V) = (2900, 113870.948123)
(n, V) = (3000, 121161.716834)
(n, V) = (3100, 128380.273714)
(n, V) = (3200, 137475.698790)
(n, V) = (3300, 146118.043477)
(n, V) = (3400, 153566.585019)
(n, V) = (3500, 163336.916054)
(n, V) = (3600, 175587.274039)
(n, V) = (3700, 185338.216517)
(n, V) = (3800, 193897.593360)
(n, V) = (3900, 203004.147252)
(n, V) = (4000, 215111.560730)
(n, V) = (4100, 228582.115943)
(n, V) = (4200, 236174.801135)
(n, V) = (4300, 246420.318332)
(n, V) = (4400, 257242.978548)
(n, V) = (4500, 268404.585122)
(n, V) = (4600, 283755.666438)
(n, V) = (4700, 295126.464818)
(n, V) = (4800, 309983.957599)
(n, V) = (4900, 323376.999940)
(n, V) = (5000, 332441.936831)

y = 68.286134x + -59105.009423
y = 0.013336x^2 + 0.272923x + -160.226913
```

6.4 考察

リスト 11 の結果をプロットしたグラフを図 2、リスト 12 の結果をプロットしたグラフを図 3 に示す。

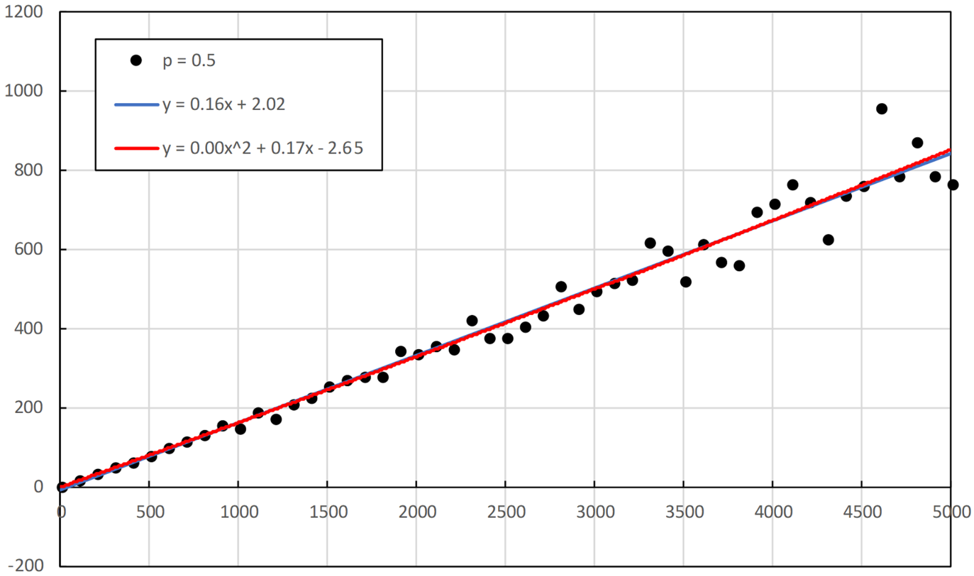

図 2　$p = 0.5$ のときのグラフ

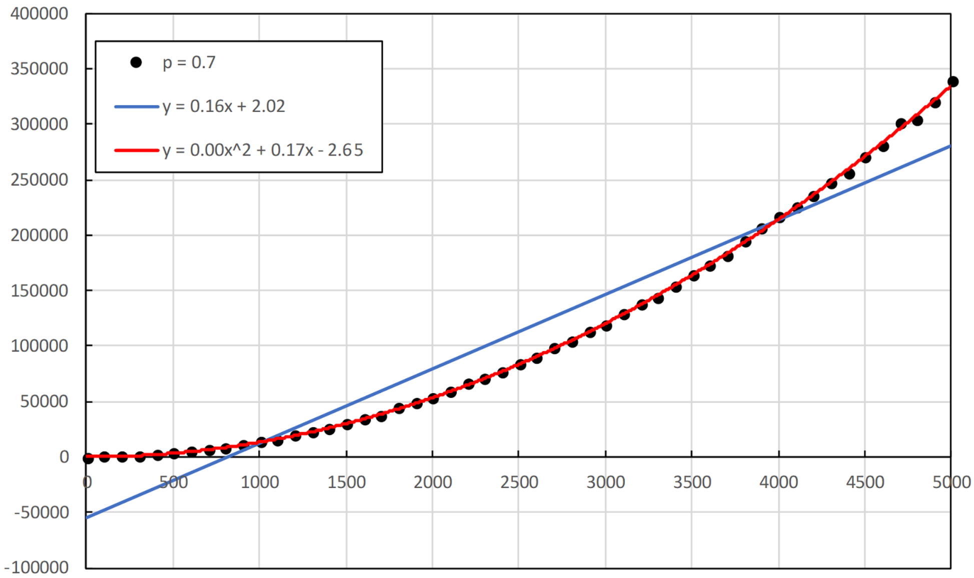

図 3　$p = 0.7$ のときのグラフ