4J シミュレーション
レポート 3

4 年 電子情報工学科 9 番
岡村宙輝

2019 年　2 月　19 日 (火) 提出

1　目的

ガウスの消去法を使ったプログラムを用いて, 連立一次方程式が解けるようになる. さらに, ガウスの消去法を使って最小二乗法による直線, 曲線近似を行うプログラムを作成する. このプログラムを利用し, データを近似した 1 次多項式の傾きと切片, ,2 次多項式の係数などを求める.

2　実行環境

実行環境を表 1 に示す.

表 1: 実行環境

CPU	Intel(R)Core(TM) i7-4500U CPU @ 1.80GHz 2.4GHz
メモリ	8.00GB
OS	Ubuntu 16.04.5 LTS
言語	Python3,C
コンパイラ	gcc 5.4.0

3　課題 9:ガウスの消去法

連立方程式を解くための, ガウスの消去法と呼ばれる方法を用いる. ガウスの消去法では大きくわけて前進消去と後退代入の 2 つの手順がある [1].

$$\begin{cases} x_1 + 2x_2 = 10 \\ 3x_1 + 4x_2 = 15 \end{cases} \tag{1}$$

式 (1) を解く場合, まず式 (2) のように変形して A, b を入力とする.

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}, b = \begin{pmatrix} 10 \\ 15 \end{pmatrix} \tag{2}$$

前進消去のアルゴリズムを次に示す.n は A の次数を表しており, 式 (2) の場合は 2 である.A の i 行 k 列目の値を a_{ik} と表す.

▷ 各列 $k = 1, 2, \ldots, n-1$ について, 次の処理を反復する.
- 各行 $i = k+1, k+2, \ldots, n$ について, 次の処理を反復する.
  (1). 乗数 $m \leftarrow a_{ik}/a_{kk}$
  (2). $a_{ik} \leftarrow 0$.(この代入は無くてもよい.)
  (3). 各列 $j = k+1, k+2, \ldots, n$ について, $a_{ij} \leftarrow a_{ij} - a_{kj} \times m$ を反復する.
  (4). $b_i \leftarrow b_i - b_k \times m$

前進消去を行った後は後退代入を行う. 後退代入のアルゴリズムを次に示す.

▷ 各行 $k = n, n-1, \ldots, 1$ について, 次の処理を反復する.
(1). $x_k \leftarrow b_k$
(2). 各列 $j = k+1, k+2, \ldots, n$ について,$x_k \leftarrow x_k - a_{kj} \times x_j$ を反復する.
(3). $x_k \leftarrow x_k/a_{kk}$

3.1 課題 9-1

式 (3) の連立方程式を解く. 作成したプログラムをソースコード 1 に示す.

$$\begin{cases} 2x + 2y + 6z = 24 \\ 3x + 5y + 13z = 52 \\ 5x + 8y + 24z = 93 \end{cases} \tag{3}$$

ソースコード 1: 課題 9-1 のプログラム

```
A = [[2, 2, 6], [3, 5, 13], [5, 8, 24]]
b = [24, 52, 93]

n = len(A[0])
print('左辺:')
for i in range(n):
    print(A[i])
print('右辺:',b)

for k in range(n - 1):
    for i in range(k+1, n):
        m = A[i][k] / A[k][k]
        A[i][k] = 0
        for j in range(k+1, n):
            A[i][j] = A[i][j] - A[k][j] * m
        b[i] = b[i] - b[k] * m

print('前進消去後:')
print('左辺:', A)
print('右辺:', b)
x = [0,0,0]

for k in range(n - 1, -1, -1):
    x[k] = b[k]
    for j in range(k+1, n):
        x[k] = x[k] - A[k][j]*x[j]
    x[k] = x[k] / A[k][k]


print('後進代入後:',x)
```

実行結果を次に示す.

課題 9-1 の実行結果

```
左辺:
[2, 2, 6]
```

```
[3, 5, 13]
[5, 8, 24]
右辺: [24, 52, 93]
前進消去後: [24, 16.0, 9.0]
後進代入後: [1.0, 2.0, 3.0]
```

実行結果より, 正しく計算できていることがわかる.

3.2 課題 9-2

式 (4) の連立方程式をガウスの消去法プログラムで解き, x_1〜x_4 までの値を求める．得られた解が連立方程式を満たすことを元の方程式に代入して確認する．作成したプログラムをソースコード 2 に示す.

$$\begin{cases} x_1 + 2x_2 + x_3 + 5x_4 = 20.5 \\ 8x_1 + x_2 + 3x_3 + x_4 = 14.5 \\ x_1 + 7x_2 + x_3 + x_4 = 18.5 \\ x_1 + x_2 + 6x_3 + x_4 = 9.0 \end{cases} \tag{4}$$

ソースコード 2: 課題 9-2 のプログラム

```
1  A = [[1, 2, 1, 5], [8, 1, 3, 1], [1, 7, 1, 1], [1, 1, 6, 1]]
2  b = [20.5, 14.5, 18.5, 9.0]
3  a = [[1, 2, 1, 5], [8, 1, 3, 1], [1, 7, 1, 1], [1, 1, 6, 1]]
4
5  n = len(A[0])
6  print('左辺:')
7  for i in range(n):
8      print(A[i])
9  print('右辺:',b)
10
11
12 for k in range(n - 1):
13     for i in range(k+1, n):
14         m = A[i][k] / A[k][k]
15         for j in range(k+1, n):
16             A[i][j] = A[i][j] - A[k][j] * m
17         b[i] = b[i] - b[k] * m
18
19 print('前進消去後:',b)
20
21 x = [0]*len(A[0])
22
23 for k in range(n - 1, -1, -1):
24     x[k] = b[k]
25     for j in range(k+1, n):
26         x[k] = x[k] - A[k][j]*x[j]
27     x[k] = x[k] / A[k][k]
```

```
28
29
30 print('後進代入後:',x)
31 print('*****     検 算     *****')
32 for i in range(n):
33     print(a[i][0],'*',x[0],'+',a[i][1],'*',x[1],'+',\
34           a[i][2],'*',x[2],'+',a[i][3],'*',x[3],'=',\
35           "{0:.05f}".format(a[i][0]*x[0]+a[i][1]*x[1]+a[i][2]*x[2]+a[
                 i][3]*x[3]))
```

実行結果を次に示す.

課題 9-2 の実行結果

```
左辺:
[1, 2, 1, 5]
[8, 1, 3, 1]
[1, 7, 1, 1]
[1, 1, 6, 1]
右辺: [20.5, 14.5, 18.5, 9.0]
前進消去後: [20.5, -149.5, -51.83333333333333, -167.4]
後進代入後: [1.0000000000000036, 2.0, 0.4999999999999972, 3.0]
*****     検算     *****
1 * 1.0000000000000036 + 2 * 2.0 + 1 * 0.4999999999999972 + 5 * 3.0 = 20.50000
8 * 1.0000000000000036 + 1 * 2.0 + 3 * 0.4999999999999972 + 1 * 3.0 = 14.50000
1 * 1.0000000000000036 + 7 * 2.0 + 1 * 0.4999999999999972 + 1 * 3.0 = 18.50000
1 * 1.0000000000000036 + 1 * 2.0 + 6 * 0.4999999999999972 + 1 * 3.0 = 9.00000
```

検算を行った結果より, 正しく計算できていることがわかる.

4 課題 10:ガウスの消去法 (ピボット選択)

前述したように前進消去のときには, 乗数 m に a_{ik}/a_{kk} を代入する. このとき a_{kk} をピボットと呼び, ピボットが 0 である場合はエラーになってしまう. このエラーを避けるために, 絶対値を比較して行を入れ替えるピボット選択という処理を行う.

4.1 課題 10-1

式 (5) の連立方程式を解き, ピボット選択ありとなしで比較する. 作成したプログラムをソースコード 3 に示す.

$$\begin{cases} -2x + 2y = 0 \\ 3x - 3y + z = 0 \\ 2x + y + 6z = 0 \end{cases} \tag{5}$$

ソースコード 3: 課題 10-1 のプログラム

```
def gaussian(A,b):
    n = len(A[0])
    for k in range(n - 1):
        for i in range(k+1, n):
            m = A[i][k] / A[k][k]
            A[i][k] = 0
            for j in range(k+1, n):
                A[i][j] = A[i][j] - A[k][j] * m
            b[i] = b[i] - b[k] * m

    x = [0] * n

    for k in range(n - 1, -1, -1):
        x[k] = b[k]
        for j in range(k+1, n):
            x[k] = x[k] - A[k][j]*x[j]
        x[k] = x[k] / A[k][k]

    return b, x

def gaussian_sentaku(A,b):
    n = len(A[0])

    for k in range(n - 1):
        for j in range(n - 1, k, -1):
            if(abs(A[j][k]) > abs(A[j-1][k])):
                A[j], A[j-1] = A[j-1], A[j]
                b[j], b[j-1] = b[j-1], b[j]

        for i in range(k+1, n):
            if(A[i][k] == 0): break
            m =   A[i][k] / A[k][k]
            A[i][k] = 0
            for j in range(k+1, n):
                A[i][j] = A[i][j] - A[k][j] * m

            b[i] = b[i] - b[k] * m
    x = [0]*n
    for k in range(n - 1, -1, -1):
        x[k] = b[k]
        for j in range(k+1, n):
            x[k] = x[k] - A[k][j]*x[j]
        x[k] = x[k] / A[k][k]

    return b, x
```

```
46
47 A  = [[-2, 2, 0], [3, -3, 1], [2, 1, 6]]
48 A2 = [[-2, 2, 0], [3, -3, 1], [2, 1, 6]]
49 b = [0, 1, 9]
50
51 n = len(A2[0])
52 print('左辺:')
53 for i in range(n):
54     print(A2[i])
55 print('右辺:',b)
56
57 print('*** ピボット選択あり ***')
58 resultb, resultx = gaussian_sentaku(A2, b)
59 print('前進消去後:',resultb)
60 print('後進代入後:',resultx)
61 print('*** ピボット選択なし ***')
62 resultb, resultx = gaussian(A, b)
63 print('前進消去後:',resultb)
64 print('後進代入後:',resultx)
```

実行結果を次に示す.

課題 10-1 の実行結果

```
左辺:
[-2, 2, 0]
[3, -3, 1]
[2, 1, 6]
右辺: [0, 1, 9]
*** ピボット選択あり ***
前進消去後: [1, 8.333333333333334, 0.6666666666666666]
後進代入後: [1.0000000000000002, 1.0000000000000002, 1.0]
*** ピボット選択なし ***
Traceback (most recent call last):
  File "kadai10-1.py", line 62, in <module>
    resultb, resultx = gaussian(A, b)
  File "kadai10-1.py", line 5, in gaussian
    m = A[i][k] / A[k][k]
ZeroDivisionError: float division by zero
```

ピボット選択をしないと,0 除算が発生してしまいエラーが出てしまうことがわかる. ピボット選択をした場合は正しく計算できていることがわかる.

4.2 課題 10-2

式 (6) の連立方程式をピボット選択を使って解き, double 型と float 型で比較する. 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 の double を全て float に変えれば, float 型で求めることができる.

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

ソースコード 4: 課題 10-2 のプログラム

```
1  #include<stdio.h>
2  #include<stdlib.h>
3  #define n 4 // n次元の行列を解く
4
5  void swap(double *a, double *b)
6  {
7    double temp = *a;
8    *a = *b;
9    *b = temp;
10 }
11
12 void swap2(double a[n], double b[n])
13 {
14   double temp;
15
16   for(int i = 0; i < n;i++)
17     {
18       temp = a[i];
19       a[i] = b[i];
20       b[i] = temp;
21     }
22
23 }
24
25
26 void gaussian(double A[][n], double b[n], double x[n])
27 {
28   double m;
29   int i, j, k;
30   /* 前進消去 */
31   for(k = 0; k < n - 1; k++)
32     {
33       for(j = n -1; j > k; j--)
```

```
34    {
35          if(abs(A[j][k]) > abs(A[j-1][k]))
36      {
37        swap2(A[j], A[j-1]);
38        swap(&b[j], &b[j-1]);
39      }
40    }
41
42        for(i = k+1; i < n; i++)
43    {
44      if(A[i][k] == 0) break;
45      m = A[i][k] / A[k][k];
46      A[i][k] = 0;
47      for(j = k + 1; j < n; j++)
48        A[i][j] = A[i][j] - A[k][j] * m;
49
50      b[i] = b[i] - b[k] * m;
51    }
52      }
53
54    /* 後進代入 */
55    for(k = n - 1; k >= 0; k--)
56      {
57        x[k] = b[k];
58        for(j  = k + 1;j < n;j++)
59    x[k] = x[k] - A[k][j] * x[j];
60        x[k] = x[k] / A[k][k];
61      }
62
63  }
64
65  void disp_matrix(double A[n])
66  {
67    printf("(");
68    for(int i = 0; i < n; i++)
69      printf("%.10f  ",A[i]);
70    printf(")\n");
71  }
72
73  void disp_matrix2(double A[][n])
74  {
75    for(int i = 0; i < n; i++)
76      {
77        printf("(");
78        for(int j = 0; j < n; j++)
```

```
  printf("%.10f  ",A[i][j]);
      printf(")\n");
    }
}

int main (int argc, char *argv)
{

  double A[][n] = {{1.0, 0.96, 0.84, 0.64},
       {0.96, 0.9214, 0.4406, 0.2222},
       {0.84,0.4406, 1.0, 0.3444},
       {0.64, 0.2222, 0.3444, 1.0}};
  double b[n] = {3.44, 2.5442, 2.6250, 2.2066};
  double x[n];

  printf("A = \n");
  disp_matrix2(A);

  printf("b = \n");
  disp_matrix(b);


  gaussian(A, b, x);
  printf("x = \n");
  disp_matrix(x);

}
```

実行結果を次に示す.

課題 10-2 の結果:double 型

```
A =
(1.0000000000  0.9600000000  0.8400000000  0.6400000000  )
(0.9600000000  0.9214000000  0.4406000000  0.2222000000  )
(0.8400000000  0.4406000000  1.0000000000  0.3444000000  )
(0.6400000000  0.2222000000  0.3444000000  1.0000000000  )
b =
(3.4400000000  2.5442000000  2.6250000000  2.2066000000  )
x =
(1.0000000000  1.0000000000  1.0000000000  1.0000000000  )
```

課題 10-2 の結果:float 型

```
A =
(1.0000000000  0.9599999785  0.8399999738  0.6399999857  )
(0.9599999785  0.9214000106  0.4406000078  0.2222000062  )
```

```
(0.8399999738  0.4406000078  1.0000000000  0.3443999887  )
(0.6399999857  0.2222000062  0.3443999887  1.0000000000  )
b =
(3.4400000572  2.5441999435  2.6250000000  2.2065999508  )
x =
(0.9998441339  1.0001490116  1.0000523329  0.9999513030  )
```

実行結果より double 型は 1×10^{-10} まで精度が出ている一方で, float 型は 1×10^{-1} から誤差が出ていることがわかる. よって double 型の方が float 型より精度が高いことがわかる.

5 課題 11:最小二乗法

n 個の座標 $(x_1, y_1), (x_2, y_2), (x_3, y_3) \dots (x_n, y_n)$ を通る関数 $y = f(x)$ を近似するために, 最小二乗法という方法を使う. 最小二乗法では近似した式がすべての座標の点を通るということは考えず, n 個の座標と近似した式の違いが最小になるものを求める. n 個の座標 $(x_1, y_1), (x_2, y_2), (x_3, y_3) \dots (x_n, y_n)$ のデータを 1 次多項式 $y = a + bx$ で近似する場合は, 式 (7) で a, b を求めて近似を行う. 連立 1 次方程式を解くときには, ガウスの消去法を使用する.

$$\begin{cases} ma + \Big(\sum_{k=1}^{m} x_k\Big) b = \sum_{k=1}^{m} y_k \\ \Big(\sum_{k=1}^{m} x_k\Big) a + \Big(\sum_{k=1}^{m} {x_k}^2\Big) b = \sum_{k=1}^{m} x_k y_k \end{cases} \tag{7}$$

2 次多項式 $y = a + bx + cx^2$ で近似する場合は, 式 (8) で a, b, c を求めて近似を行う.

$$\begin{cases} ma + \Big(\sum_{k=1}^{m} x_k\Big) b + \Big(\sum_{k=1}^{m} {x_k}^2\Big) c = \sum_{k=1}^{m} y_k \\ \Big(\sum_{k=1}^{m} x_k\Big) a + \Big(\sum_{k=1}^{m} {x_k}^2\Big) b + \Big(\sum_{k=1}^{m} {x_k}^3\Big) c = \sum_{k=1}^{m} x_k y_k \\ \Big(\sum_{k=1}^{m} {x_k}^2\Big) a + \Big(\sum_{k=1}^{m} {x_k}^3\Big) b + \Big(\sum_{k=1}^{m} {x_k}^4\Big) c = \sum_{k=1}^{m} {x_k}^2 y_k \end{cases} \tag{8}$$

5.1 課題 11-1

表 2 のデータに対して最小二乗近似の 1 次多項式を求める. 作成したプログラムをソースコード 5 に示す.

表 2: 1 次多項式近似のデータ

i	1	2	3	4
x_i	1	2	3	4
y_i	0	1	2	4

ソースコード 5: 課題 11-1 のプログラム

```
1 import numpy as np
2 import matplotlib.pyplot as plt
```

```
3
4  def gaussian(A,b):
5      n = len(A[0])
6
7      for k in range(n - 1):
8          #ピポット選択
9          for j in range(n - 1, k, -1):
10             if(abs(A[j][k]) > abs(A[j-1][k])):
11                 A[j], A[j-1] = A[j-1], A[j]
12                 b[j], b[j-1] = b[j-1], b[j]
13         #前進消去
14         for i in range(k+1, n):
15             if(A[i][k] == 0): break
16             m =   A[i][k] / A[k][k]
17             A[i][k] = 0
18             for j in range(k+1, n):
19                 A[i][j] = A[i][j] - A[k][j] * m
20
21             b[i] = b[i] - b[k] * m
22     #後進代入
23     x = [0]*n
24     for k in range(n - 1, -1, -1):
25         x[k] = b[k]
26         for j in range(k+1, n):
27             x[k] = x[k] - A[k][j]*x[j]
28         x[k] = x[k] / A[k][k]
29
30     return x
31
32 def least_squares_method(x, y):
33     length = len(x)
34     sumx = sumx_2 = sumy = sumxy = 0
35     for i in range(length):
36         sumx = sumx + x[i]
37         sumx_2 = sumx_2 + x[i]**2
38         sumy = sumy + y[i]
39         sumxy = sumxy + x[i]*y[i]
40
41
42     A = [[length, sumx],[sumx, sumx_2]]
43     b = [sumy, sumxy]
44     return gaussian(A, b)
45
46 x = [1, 2, 3, 4]
47 y = [0, 1, 2, 4]
```

```
48
49 resx = least_squares_method(x, y)
50 print('y =', resx[0], '+', resx[1], 'x')
```

実行結果を次に示す. 表 2 のデータと実行結果から得られた近似式をプロットしたグラフを図 1 に示す.

```
    y = -1.5000000000000022 + 1.3000000000000007 x
```

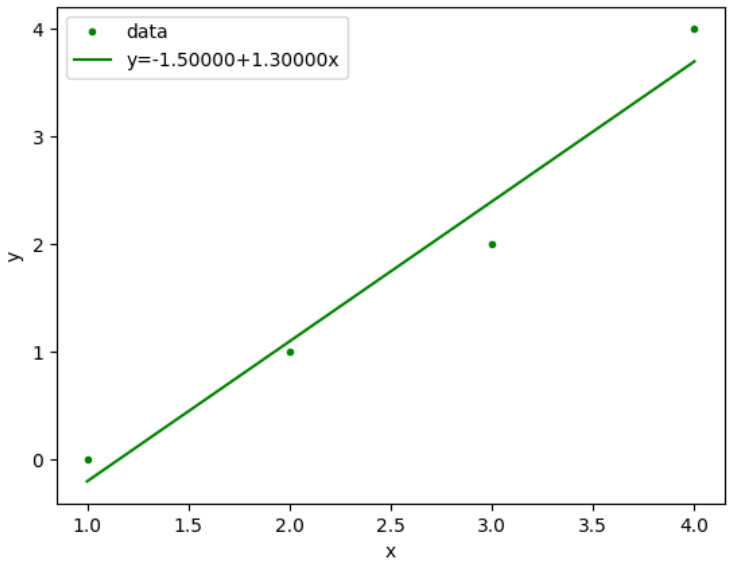

図 1: 課題 11-1 の実行結果:グラフ

グラフより正しく近似できていることがわかる.

5.2 課題 11-2

表 3 のデータに対して,2 次式 $y = a + bx + cx^2$ で近似する. 作成したプログラムをソースコード 6 に示す.

表 3: 2 次多項式近似のデータ

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

ソースコード 6: 課題 11-2 のプログラム

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 def gaussian(A,b):
5     n = len(A[0])
```

```
6
7       for k in range(n - 1):
8           #ピポット選択
9           for j in range(n - 1, k, -1):
10              if(abs(A[j][k]) > abs(A[j-1][k])):
11                  A[j], A[j-1] = A[j-1], A[j]
12                  b[j], b[j-1] = b[j-1], b[j]
13          #前進消去
14          for i in range(k+1, n):
15              if(A[i][k] == 0): break
16              m =   A[i][k] / A[k][k]
17              A[i][k] = 0
18              for j in range(k+1, n):
19                  A[i][j] = A[i][j] - A[k][j] * m
20
21              b[i] = b[i] - b[k] * m
22      #後進代入
23      x = [0]*n
24      for k in range(n - 1, -1, -1):
25          x[k] = b[k]
26          for j in range(k+1, n):
27              x[k] = x[k] - A[k][j]*x[j]
28          x[k] = x[k] / A[k][k]
29
30      return x
31
32  def least_squares_method(x, y):
33      length = len(x)
34      sumx = sumx_2 = sumx_3 = sumx_4 = sumy = sumxy = sumx2y = 0
35      for i in range(length):
36          sumx = sumx + x[i]
37          sumx_2 = sumx_2 + x[i]**2
38          sumx_3 = sumx_3 + x[i]**3
39          sumx_4 = sumx_4 + x[i]**4
40          sumy = sumy + y[i]
41          sumxy = sumxy + x[i]*y[i]
42          sumx2y = sumx2y + x[i]**2*y[i]
43
44
45      A = [[length, sumx, sumx_2],
46           [sumx, sumx_2, sumx_3],
47           [sumx_2, sumx_3, sumx_4]]
48      b = [sumy, sumxy, sumx2y]
49
50      return gaussian(A, b)
```

```
51
52 x = [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6]
53 y = [0.000, 0.034, 0.138, 0.282, 0.479, 0.724, 1.120]
54
55 resx = least_squares_method(x, y)
56
57 print(resx)
58
59 length = len(x)
60 fx = np.empty((length,))
61 fy = np.empty((length,))
62 fansx = np.empty((600,))
63 fansx =  np.arange(x[0], x[-1], (x[-1] - x[0]) / 600)
64 fans = np.empty((600,))
65 fx = x
66 fy = y
67
68 for i in range(600):
69     fans[i] = resx[0] + resx[1]*fansx[i] + resx[2]*fansx[i]**2
70
71 labelk = 'y='+format(resx[0], '.3f')+'+'+format(resx[1], '.3f')+'x+'\
72          +format(resx[2], '.3f')+'x^2'
73
74 print(fansx.size ,':', fans.size)
75
76 fig = plt.figure()
77 ax = plt.gca()
78 plt.plot(fansx, fans, linestyle='-', c = 'b',label = labelk)
79 plt.plot(fx, fy, '.', c = 'r',label = "data")
80 ax.set_xlabel("x")
81 ax.set_ylabel("y")
82 plt.legend()
83 plt.show()
```

実行結果のグラフを図 2 に示す. 図 2 より正しく近似できていることがわかる.

6 課題 12:ランダムウォーク

乱数を用いたシミュレーションの例として, ランダムウォークと呼ばれるものがある. ランダムウォークでは, ある一定の確率変数に従って直線状をランダムに動く点を考える. この点は $x = 0$ から動き始め, 確率 p で L だけ右に進み, 確率 $q = 1 - p$ で L だけ左に進むようにする.

6.1 課題 12-1

点を N 回動かした後の変位 x, 二乗変位 x^2 は式 (9) で求められる.

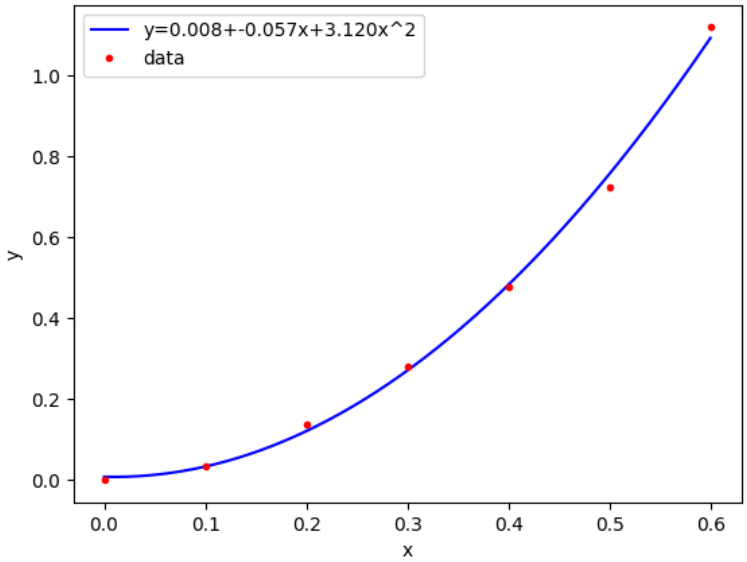

図 2: 課題 11-2 の実行結果:グラフ

$$x(n) = \sum_{i=1}^{N} S_i, x^2(n) = \Big(\sum_{i=1}^{N} S_i \Big)^2 \tag{9}$$

ここで $S_i = \pm L$ である. 今回作成したプログラムでは $L = 1$ とした. $x(n)$ と $x^2(n)$ より, 式 (10) で分散 $\Delta x^2(N)$ を求める.

$$\Delta x^2(N) = x^2(N) - x(N) \tag{10}$$

平均的なデータをとるために点を 10 個動かし, 平均した $x^2(N), x(N)$ を使って式 (11) で分散を求めるようにする.

$$\Delta x^2(N) = <\Delta x^2(N)> - <x(N)> \tag{11}$$

作成したプログラムをソースコード 7 に示す. N が 100 増えるごとに分散を求め, グラフにプロットする. さらに, プロットした点に対して最小二乗法による 1 次多項式と 2 次多項式の近似を行い, プロットした点と比較する.

ソースコード 7: 課題 12-1 のプログラム

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  from numpy.random import *
4
5  MAX = 10**6
6  STEP = 100
7  CALSTEP = int(MAX / STEP)
8  DATA = 10
9  P = 0.5
10
```

```
11 # p割の確率で +1
12 def randomwalk(p):
13     sumwalk = 0
14     for i in range(STEP):
15         random = randint(0,10)
16 
17         if random < p:
18             sumwalk += 1
19         else:
20             sumwalk -= 1
21 
22     return sumwalk
23 
24 def gaussian(A,b):
25     n = len(A[0])
26 
27     for k in range(n - 1):
28         #ピポット選択
29         for j in range(n - 1, k, -1):
30             if(abs(A[j][k]) > abs(A[j-1][k])):
31                 A[j], A[j-1] = A[j-1], A[j]
32                 b[j], b[j-1] = b[j-1], b[j]
33         #前進消去
34         for i in range(k+1, n):
35             if(A[i][k] == 0): break
36             m =   A[i][k] / A[k][k]
37             A[i][k] = 0
38             for j in range(k+1, n):
39                 A[i][j] = A[i][j] - A[k][j] * m
40 
41             b[i] = b[i] - b[k] * m
42     #後進代入
43     x = [0]*n
44     for k in range(n - 1, -1, -1):
45         x[k] = b[k]
46         for j in range(k+1, n):
47             x[k] = x[k] - A[k][j]*x[j]
48         x[k] = x[k] / A[k][k]
49 
50     return x
51 
52 def least_squares_method(x, y):
53     length = len(x)
54     sumx = sumx_2 = sumy = sumxy = 0
55     for i in range(length):
```

```
56          sumx = sumx + x[i]
57          sumx_2 = sumx_2 + x[i]**2
58          sumy = sumy + y[i]
59          sumxy = sumxy + x[i]*y[i]
60
61      A = [[length, sumx],[sumx, sumx_2]]
62      b = [sumy, sumxy]
63
64      return gaussian(A, b)
65
66  def least_squares_method_2(x, y):
67      length = len(x)
68      sumx = sumx_2 = sumx_3 = sumx_4 = sumy = sumxy = sumx2y = 0
69      for i in range(length):
70          sumx = sumx + x[i]
71          sumx_2 = sumx_2 + x[i]**2
72          sumx_3 = sumx_3 + x[i]**3
73          sumx_4 = sumx_4 + x[i]**4
74          sumy = sumy + y[i]
75          sumxy = sumxy + x[i]*y[i]
76          sumx2y = sumx2y + x[i]**2*y[i]
77
78
79      A = [[length, sumx, sumx_2],
80           [sumx, sumx_2, sumx_3],
81           [sumx_2, sumx_3, sumx_4]]
82      b = [sumy, sumxy, sumx2y]
83
84      return gaussian(A, b)
85
86
87
88  # 酔歩者が進んだ距離
89  sum = [0] * DATA
90
91  # 酔歩者が進んだ距離の分散
92  bunsan = np.empty((CALSTEP,))
93
94  # プロットするデータのx座標
95  bunsanx = np.empty((CALSTEP,))
96  lsm1 = np.empty((CALSTEP,))
97  lsm2 = np.empty((CALSTEP,))
98
99  bunsan[0] = 0
100
```

```
for i in range(1, CALSTEP):
    # STEPごとに合計を求めていく.
    # DATA個のデータについて行う.
    for j in range(DATA):
        sum[j] = sum[j] + randomwalk(P*10)

    # 全てのデータで進んだ歩数の合計を求める.
    # 合計:allsum, 二乗の合計:allsum2

    allsum = allsum2 = 0

    for j in range(DATA):
        allsum += sum[j]
        allsum2 += sum[j]**2

    # 分散を求める.
    bunsan[i] =  (1/DATA)*allsum2 - ((1/DATA)*allsum)**2

    # プロットする点のx座標
    bunsanx[i] = i * STEP

a,b = least_squares_method(bunsanx, bunsan)
a2,b2,c2 = least_squares_method_2(bunsanx, bunsan)

lsm1 = a + b * bunsanx
lsm2 = a2 + b2 * bunsanx + c2 * bunsanx**2

labelb = 'p = '+ str(P)
label1 = format(a, '.2f') + '+' + format(b, '.5f') + 'x'
label2 = format(a2, '.3f') + '+' + format(b2, '.3f') + 'x'\
          + '+' +  format(c2, '.10f') + 'x^2'

fig = plt.figure()
ax = plt.gca()
ax.plot(bunsanx, bunsan, c='g', marker='.', label=labelb, linestyle='N
      one')
ax.plot(bunsanx, lsm1, c='r', linestyle='-', label=label1)
ax.plot(bunsanx, lsm2, c='k', linestyle='--', label=label2)
ax.set_title("step = 100")
ax.set_xlabel("N")
ax.set_ylabel("dispersion")
plt.legend()
plt.show()
```

$p = 0.5$ のときの実行結果を図 3 に,$p = 0.7$ のときの実行結果を図 4 に示す.

グラフを見るとどちらも N が大きくなるにつれて, 分散が大きくなっていることがわかる. これは分散が

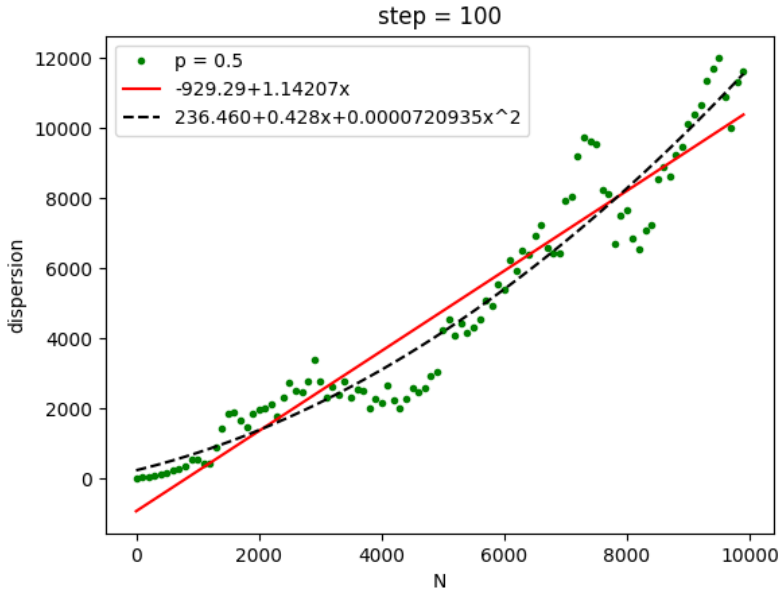

図 3: $p = 0.5$

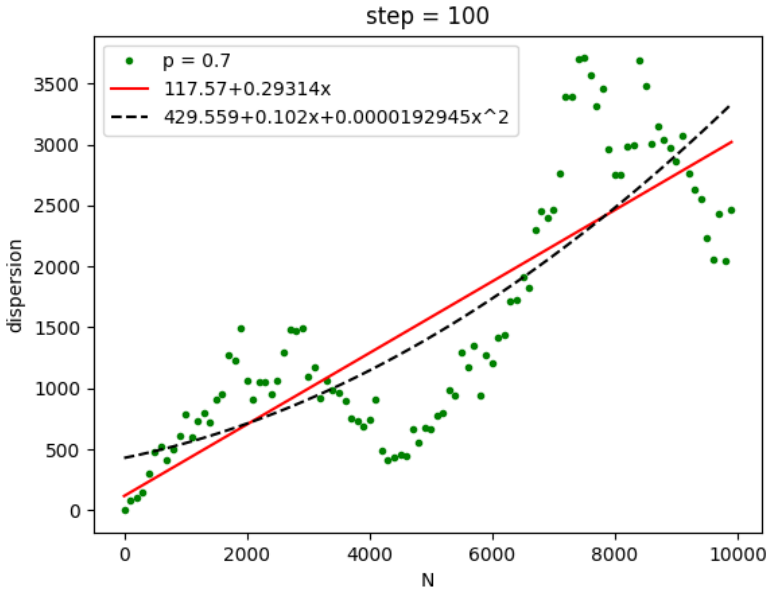

図 4: $p = 0.7$

データの散らばりの度合いを表す値であり,N が大きくなればなるほど $x(N)$ 自体が大きくなるためであると考えられる.$p = 0.5$ と $p = 0.7$ を比べると,$p = 0.7$ の方が分散が小さくなっていることがわかる. これは $p = 0.7$ の方が点が右に動く可能性が高く, データの散らばりが大きくなりにくいためであると考えられる.

6.2 課題 12-2

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 20 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. ただし, すべての粒子の初期値は,$x(0) = y(0) = 0$ とする. 作成し

たプログラムをソースコード 8 に示す.

ソースコード 8: 課題 12-2 のプログラム

```
import numpy as np
import matplotlib.pyplot as plt
from numpy.random import *

MAX = 10000
STEP = 100
CALSTEP = int(MAX / STEP)
P = 0.7
PLOTSTEP = 100

# p割の確率で +1
def randomwalk(p):
    sumwalk = 0
    for i in range(STEP):
        random = randint(0,10)

        if random < p:
            sumwalk += 1
        else:
            sumwalk -= 1

    return sumwalk

# 酔歩者が進んだ距離
sumx = [0] * 200
sumy = [0] * 200
plot_x = np.empty((200,))
plot_y = np.empty((200,))

# STEPごとのデータを格納
for i in range(1, CALSTEP+1):
    for j in range(200):
        sumx[j] = sumx[j] + randomwalk(P*10)
        sumy[j] = sumy[i] + randomwalk(P*10)
    print('step',i * STEP,' done.')
    if i * STEP == PLOTSTEP:
        break

# プロットするデータを格納
for i in range(200):
    plot_x[i] = sumx[i]
    plot_y[i] = sumy[i]

```

```
strtitle = "N = " + str(PLOTSTEP)
fig = plt.figure()
ax = plt.gca()
ax.plot(plot_x, plot_y, c='b', marker='.', linestyle='None')
ax.set_title(strtitle)
ax.set_xlabel("x")
ax.set_ylabel("y")
plt.show()
```

$N = 100, 1000, 10000$ ステップ後の粒子の位置を図 5,6,7 に示す. 粒子の座標を正の方向に動かす確率は $p = 0.5$ にした. さらに $N = 100, 1000, 10000$ ステップ後の粒子の位置を同じグラフにプロットした場合を図 8 に示す. さらに $p = 0.7$ の場合を図 9 に示す.

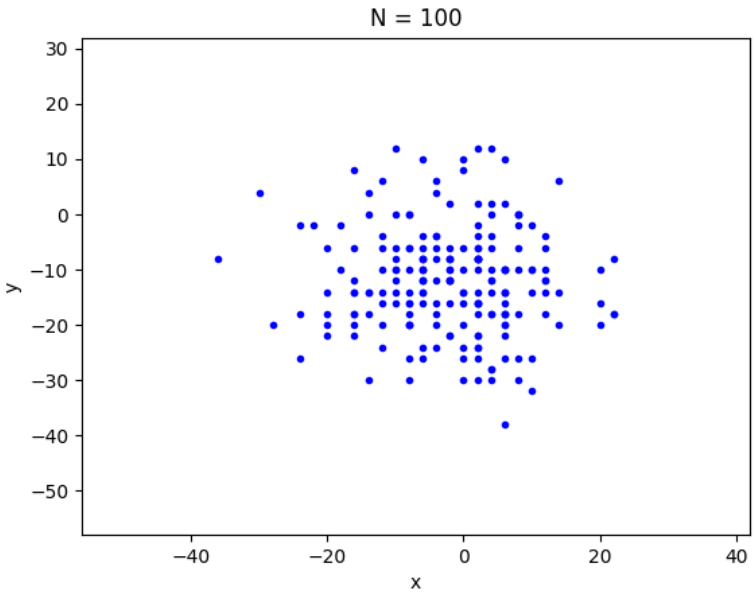

図 5: $N = 100$

図 8 では N が増えるにつれて, 粒子がちらばっていく様子がわかる. これは N が増えれば増えるほど x,y 座標が正の方向に動いた回数と, 負の方向に動いた回数が粒子ごとに異なっていくためであると考えられる. $p = 0.7$ の場合は正の方向に動きやすいため, 同じ N の粒子の集団がどんどん離れていっている. 図 9 ではわかりづらいが, 左下の点の集まりから順に $N = 100, 1000, 10000$ となっている.

参考文献

[1] 栗原正仁 (2014)『わかりやすい数値計算入門』ムイスリ出版

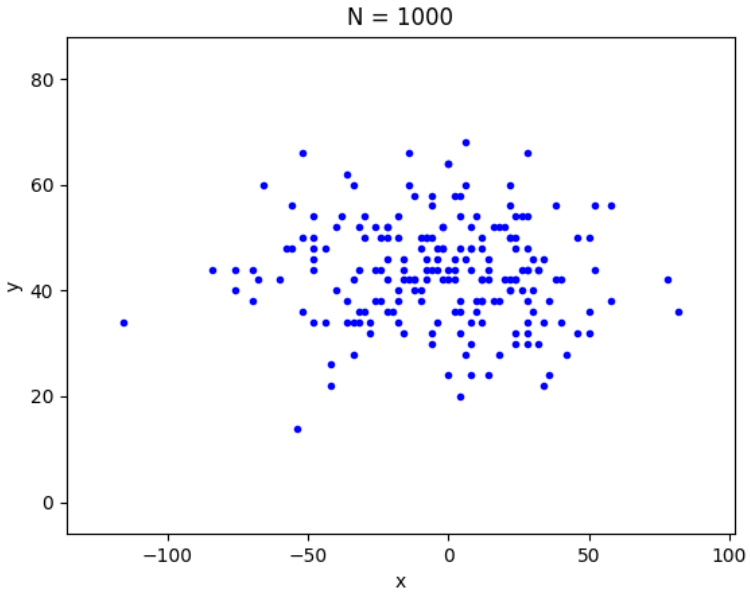

図 6: $N = 1000$

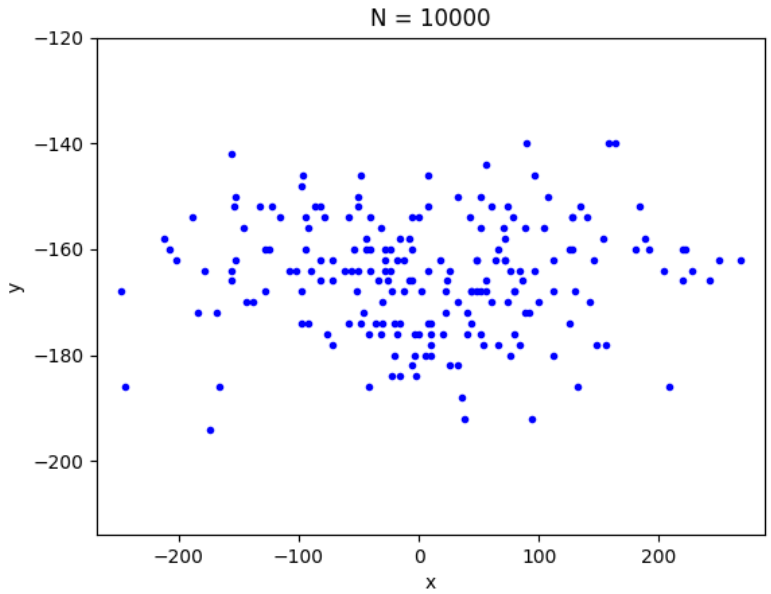

図 7: $N = 10000$

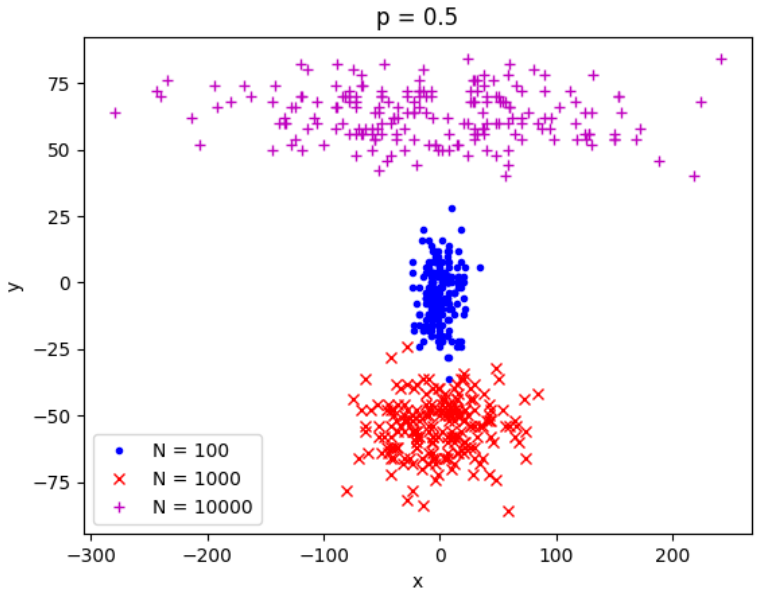

図 8: $N = 100, 1000, 10000 (p = 0.5)$

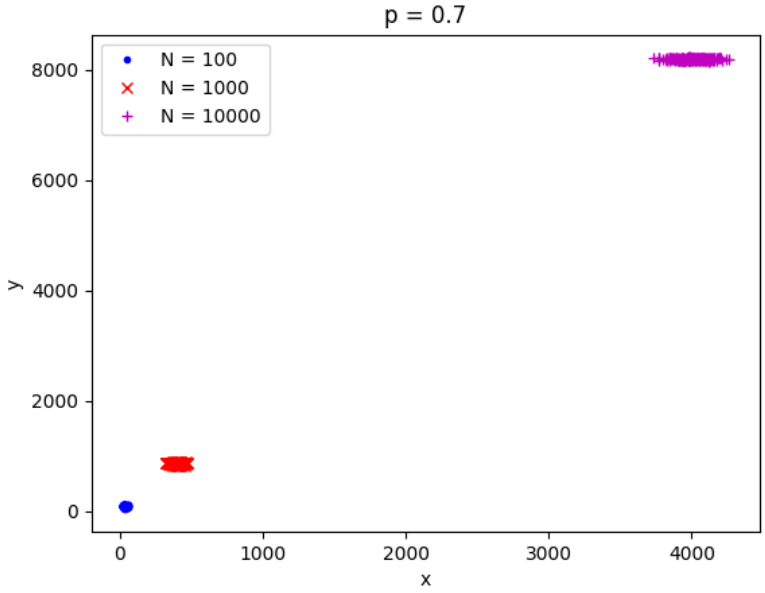

図 9: $N = 100, 1000, 10000 (p = 0.7)$