# 計算機実習 問題 12.2 2 次元のランダムウォーク

早稲田大学先進理工学部物理学科 B4 藤本將太郎

2014/05/30

## 1　シミュレーションの目的

2 次元格子状を互いに独立に動く多数の粒子を考える。初期状態として、正方格子の原点にランダムウォークをする粒子の集団を置いて、どの粒子も、各分割時間ごとに等しい確率で可能な 4 方向のいずれかにランダムに移動するとして、粒子の訪れた位置を記録する。

## 2　作成したプログラム

本シミュレーションで作成したプログラムを以下に示す。

### 2.1　ダイアログを表示するプログラム (MyDialog.py)

このプログラムでは、引数として与える文字列のリストと辞書 { 文字列:コマンド } のリストから、テキストとボタンを作成し、ボタンを押すと指定したコマンドを実行するようなダイアログを表示する。

```
1   #! /usr/bin/env python
2   # -*- coding:utf-8 -*-
3   #
4   # written by Shotaro Fujimoto, May 2014.
5
6   from Tkinter import *
7
8
9   class Dialog():
10
11      def show_window(self, texts, commands):
12          """ Show a dialog window.
13
14          texts: A list of text in a dialog
15          commands: A list of dictionary {'name of button': command}
16          """
```

```
17          self.root = Tk()
18          self.root.title('Dialog')
19
20          frame1 = Frame(self.root, padx=5, pady=5)
21          frame1.pack(side='top')
22          for text in texts:
23              label = Label(frame1, text=text)
24              label.pack()
25
26          frame2 = Frame(self.root, padx=5, pady=5)
27          frame2.pack(side='bottom')
28          self.button = []
29          for i, command in enumerate(commands):
30              self.button.append(Button(frame2, text=command.items()[0][0],
31                                        command=command.items()[0][1]
32                                        )
33                                 )
34              self.button[i].grid(row=0, column=i)
35
36          self.root.mainloop()
37
38      def quit(self):
39          self.root.destroy()
```

## 2.2 2 次元ランダムウォークのシミュレーション (12-2_random_walk_d2.py)

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。モジュール MyDialog を用いて、figure ボタンを押すと nwalkers 個の粒子の 2 次元ランダムウォークの軌跡が表示され、graph ボタンを押すと $< x(N) >$,$< y(N) >$,$< \Delta x^2(N) >$, $< \Delta y^2(N) >$,$< \Delta R^2(N) >=< x^2(N) > + < y^2(N) > - < x(N) >^2 - < y(N) >^2$ の値を、N に対してプロットする。Quit ボタンを押すと、プログラムを終了する。各関数の説明をすると、まず、random_walk_d2 は N のリストと nwalkers の値から、x と y の配列を作成し、乱数の値の大きさに応じて次の要素の値を更新していく。こうして得られた配列を元に、calc では $< x(N) >$, $< y(N) >$,$< x^2(N) >$,$< y^2(N) >$,$< \Delta x^2(N) >$, $< \Delta y^2(N) >$,$< \Delta R^2(N) >$ の値を計算する。draw_figure では、x と y の配列のデータをそのまま用いて、2 次元平面上にその軌跡をプロットする。このとき、毎回 plt.plot を呼び出すのではなく、set_data メソッドを用いて描画を行うことによって処理を軽くしている。plot_graph では calc で計算した値を用いてそれぞれの量を N に対してプロットする。

```
1   #! /usr/bin/env python
2   # -*- coding:utf-8 -*-
3   #
```

```
4  # written by Shotaro Fujimoto, May 2014.
5  """ This program is the simuration of random walk in two-dimentional space.
6  """
7  import numpy as np
8  import matplotlib.pyplot as plt
9  import sys
10 from MyDialog import Dialog
11
12
13 def random_walk_d2(l=1, x0=0, y0=0):
14     """ Random walk in two-dimentional space.
15     """
16     global x, y
17     x = np.zeros([nwalkers, max(N)], 'i')
18     y = np.zeros([nwalkers, max(N)], 'i')
19     p = np.random.random([nwalkers, max(N) - 1])
20
21     for n in range(nwalkers):
22         x[n][0] = x0
23         y[n][0] = y0
24         for i in range(1, max(N)):
25             if p[n][i - 1] < 0.25:
26                 x[n][i] = x[n][i - 1] + l
27                 y[n][i] = y[n][i - 1]
28             elif p[n][i - 1] < 0.5:
29                 x[n][i] = x[n][i - 1] - l
30                 y[n][i] = y[n][i - 1]
31             elif p[n][i - 1] < 0.75:
32                 x[n][i] = x[n][i - 1]
33                 y[n][i] = y[n][i - 1] + l
34             else:
35                 x[n][i] = x[n][i - 1]
36                 y[n][i] = y[n][i - 1] - l
37
38
39 def calc():
40     """ Caluculate the average and the variance of x(N) and y(N)
41     """
42     x_ave = np.average(x, axis=0)
43     y_ave = np.average(y, axis=0)
```

```
44      x_2_ave = np.average(x * x, axis=0)
45      y_2_ave = np.average(y * y, axis=0)
46      variance_x = x_2_ave - x_ave * x_ave
47      variance_y = y_2_ave - y_ave * y_ave
48      R_2 = x_2_ave + y_2_ave - x_ave ** 2 - y_ave ** 2
49
50      return [x_ave, y_ave, variance_x, variance_y, R_2]
51
52
53  def draw_figure():
54      """ Draw the figure of two-dimentional random walk.
55      """
56      fig = plt.figure('random walk figure')
57      ax = fig.add_subplot(111, aspect='equal')
58      ax.grid()
59      xmin, xmax = np.amin(x), np.amax(x)
60      ymin, ymax = np.amin(y), np.amax(y)
61      xmargin, ymargin = (xmax - xmin) * 0.05, (ymax - ymin) * 0.05
62      ax.set_xlim(xmin - xmargin, xmax + xmargin)
63      ax.set_ylim(ymin - ymargin, ymax + ymargin)
64      ax.set_xlabel(r'$x$')
65      ax.set_ylabel(r'$y$')
66
67      for n in range(nwalkers):
68          l, = ax.plot([], [], 'r-')
69          l.set_data(x[n], y[n])
70      plt.show().float32
71
72
73  def plot_graph(x_data, y_data, x_labels, y_labels):
74      """ Plot the graph about y_data for each x_data.
75      """
76      d = len(y_data)
77      if not len(x_data) == len(y_data) == len(x_labels) == len(y_labels):
78          raise ValueError("Arguments must have the same dimension.")
79      if d == 0:
80          raise ValueError("At least one data for plot.")
81      if d > 9:
82          raise ValueError("""So much data for plot in one figure.
83                              Please divide two or more data sets.""")
```

```
84
85      fig = plt.figure(figsize=(9, 8))
86      subplotposition = ['11', '21', '22', '22', '32', '32', '33', '33', '33']
87      axes = []
88      for n in range(d):
89          lmn = int(subplotposition[d - 1] + str(n + 1))
90          axes.append(fig.add_subplot(lmn))
91
92      for i, ax in enumerate(axes):
93          ymin, ymax = min(y_data[i]), max(y_data[i])
94          ymargin = (ymax - ymin) * 0.1
95          ax.set_ylim(ymin - ymargin, ymax + ymargin)
96          ax.plot(x_data[i], y_data[i])
97          ax.set_xlabel(x_labels[i], fontsize=16)
98          ax.set_ylabel(y_labels[i], fontsize=16)
99
100     fig.subplots_adjust(wspace=0.2, hspace=0.5)
101     fig.tight_layout()
102     plt.show()
103
104 if __name__ == '__main__':
105
106     import time
107     N = xrange(1, 501)  # caluculate when N = *
108     nwalkers = 500  # number of random walkers
109
110     def show_figure():
111         sttime = time.time()
112         random_walk_d2()
113         entime = time.time()
114         print entime - sttime
115         draw_figure()
116
117     def graph():
118         sttime = time.time()
119         random_walk_d2()
120         entime = time.time()
121         print entime - sttime
122         x_labels = [r'$N$'] * 5
123         y_labels = [r'$<x(N)>$', r'$<y(N)>$', r'$<\Delta x^{2}(N)>$',
```

```
124                     r'$<\Delta y^{2}(N)>$', r'$<\Delta R^{2}(N)>$']
125         plot_graph([N] * 5, calc(), x_labels, y_labels)
126
127     window = Dialog()
128     window.show_window(["Simulation of random walk in two-dimentional space.",
129                         "Press the button to start the simulation."],
130                        [{'figure': show_figure}, {
131                            'graph': graph}, {'Quit': sys.exit}]
132                        )
```

## 3 実習課題

a. 粒子数を nwalkers $\geq 200$、各粒子のステップ数を $N \geq 500$ にとり、プログラムを実行せよ。各粒子が一匹の蜂を表していると考えたときに、蜂の群れの形の定性的な性質について述べよ。群れの境界の定性的な性質を $N$ の関数として説明せよ。境界はギザギザしているか。それとも滑らかか。

プログラム_12-2_random_walk_d2.py を用いて 2 次元ランダムウォークのシミュレーションを行い (nwalkers $= 500$)、その結果を図 1、2、3 に示す。これらの図を比較して分かることとして、$N$ の量を増大させたとき、ランダムウォークによって形作られる境界はギザギザしている。すなわち、回転半径のようなものを考えたとき、その円周の長さに比べて境界の長さがより大きくなることが分かる。また、各粒子が一匹の蜂を表していると考えたときに、蜂の群れの形は、ある時間の間の観察の結果を重ねて書くと円形に近くなっており、観察する時間を長くすると (つまりステップ数を大きくすると)、境界はギザギザしているように見える。

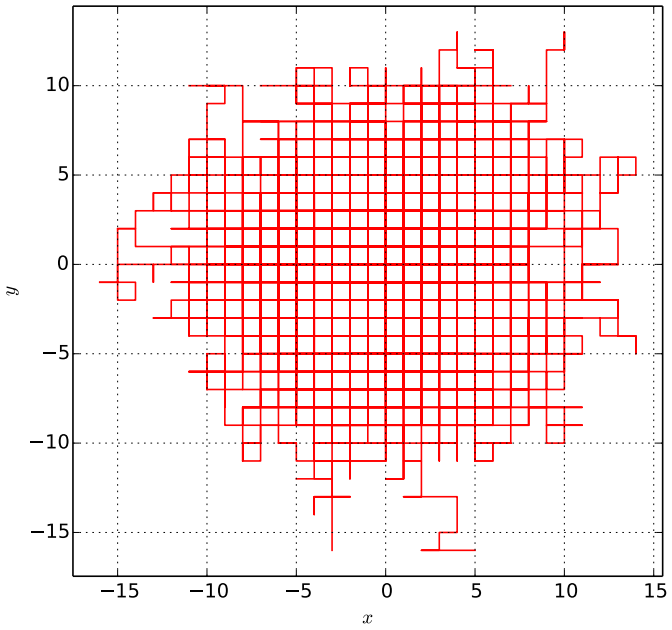


図 1　2 次元ランダムウォークのシミュレーション結果 (N=50)

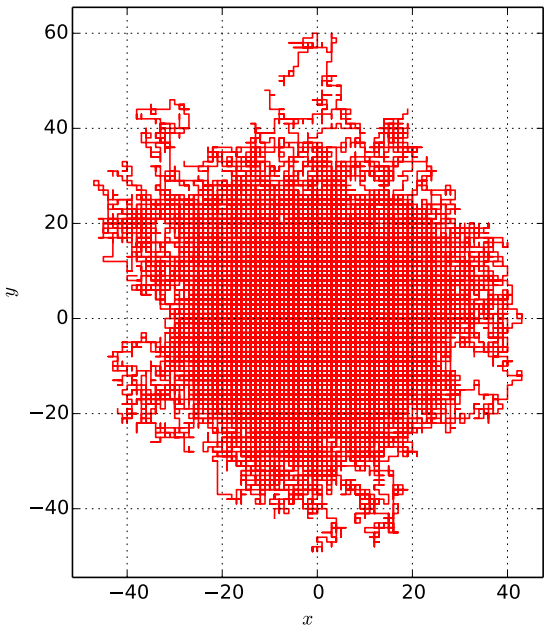


図 2　2 次元ランダムウォークのシミュレーション結果 (N=500)

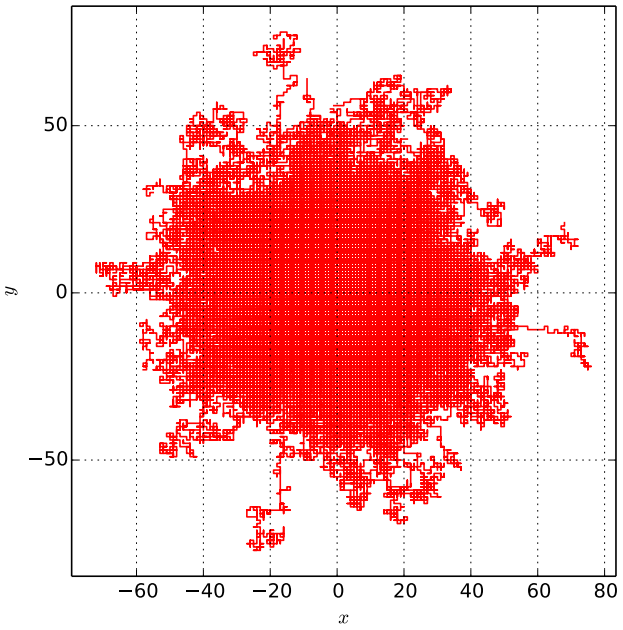


図 3　2 次元ランダムウォークのシミュレーション結果 (N=1000)

b. 量 $< x(N) >$,$< y(N) >$,$< \Delta x^2(N) >$, $< \Delta y^2(N) >$ を $N$ の関数として求めよ。平均はすべての粒子について行う。また、$< \Delta R^2(N) >=< x^2(N) > + < y^2(N) > - < x(N) >^2 - < y(N) >^2$ で与えられる平均 2 乗変位 $< \Delta R^2(N) >$ を求めよ。各量の $N$ 依存性はどうなるか。

$< x(N) >$,$< y(N) >$,$< \Delta x^2(N) >$, $< \Delta y^2(N) >$,$< \Delta R^2(N) >$ を、ステップ数 N に対してそれぞれ計算を行い (nwalkers = 500)、得られた結果をグラフにまとめたものを図 4 に示した。このグラフから読み取れるように、$< x(N) >$、$< y(N) >$ はほぼ零であり、N の依存性はない。これは 1 次元の場合と同じである。また、N が小さいところでは 0 に非常に近くなっているのに対して、N が大きくなるとばらつきが生まれるのは、問題 12.1 で考えたように、同じ精度で求めるためには N が大きいときには試行回数を増やす必要があったことを思い出せばよい。次に、$< \Delta x^2(N) >$, $< \Delta y^2(N) >$ についてであるが、これらは $N$ に比例しており、その比例係数はおよそ 1/2 であることが分かる。これは 2 次元のランダムウォークにおいて、x 方向に進む確率 $r(= 1/2)$、右に進む確率 $p(= 1/2)$、左に進む確率 $q(= 1/2)$ とすると

$$< \Delta x^2(N) >= 4pqrl^2N = 4 \times \frac{1}{2} \times \frac{1}{2} \times \frac{1}{2} \times 1^2 N = \frac{1}{2}N \tag{1}$$

となることと一致している。この $r$ は、$N$ 回のステップのうち、$N$ が非常に大きいときには、$rN$ 回が $x$ 軸方向の移動に充てられるようにみなせることと対応している。最後に $< \Delta R^2(N) >$ は、$< \Delta x^2(N) >$ と $< \Delta y^2(N) >$ の和であるので、$N$ に比例して、その傾きは 1 である。

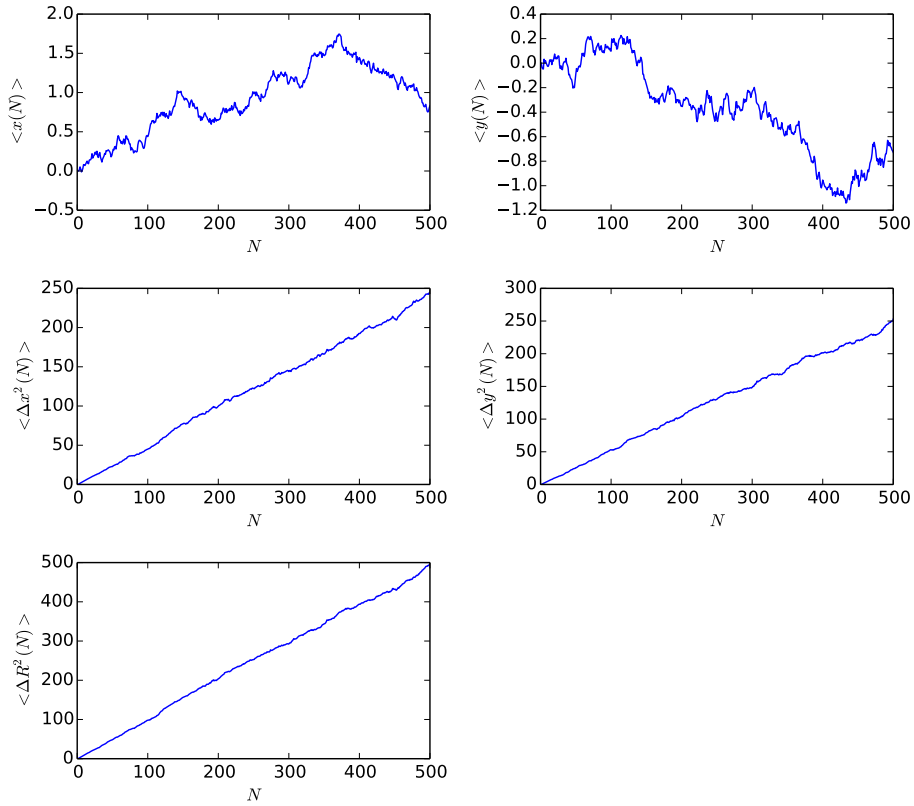


図 4　$< x(N) >$,$< y(N) >$,$< \Delta x^2(N) >$, $< \Delta y^2(N) >$,$< \Delta R^2(N) >$ の N 依存性のグラフ

## 4　まとめ

2 次元の単純なランダムウォークのシミュレーションと、そこで見られる N と平均・分散・平均 2 乗変位について成り立つ関係について調べることができた。

## 5　参考文献


- ハーベイ・ゴールド, ジャン・トボチニク, 石川正勝・宮島佐介訳『計算物理学入門』, ピアソン・エデュケーション, 2000.