計算機実習 問題 12.4 継続型ランダムウォーク

早稲田大学先進理工学部物理学科 B4 藤本將太郎

2014/06/06

1 シミュレーションの目的

このシミュレーションでは、継続型ランダムウォーク、すなわち一つ前の遷移の履歴が遷移の方向を確率的に定めるような場合のランダムウォークを考える。このモデルの物理的な対応としては、後の問題で言及されているように、クロマトグラフィー柱の拡散の問題などが挙げられる。

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 継続型ランダムウォークのシミュレーション-計算部 (HistoryDependentRW.py)

```
#! /usr/bin/env python
# -*- coding:utf-8 -*-
#
# written by Shotaro Fujimoto, June 2014.

import numpy as np
import matplotlib.pyplot as plt

class HistoryDependentRW():

    def __init__(self, alpha=0.75, l=(1,1), nwalkers=1000, x0=0):
        """ Initialize the values.

        alpha    : probability that a particle moves the same direction of the before one
        l        : step length
        nwalkers : number of trials
        x0       : initial position
        """
```

```
19          self.alpha = alpha
20          self.l = l
21          self.nwalkers = nwalkers
22          self.x0 = x0
23
24      def random_walk_d1(self, N):
25          """ Caluculate the displacements of each walkers.
26
27          N : A list of walk steps
28          """
29          x = np.zeros([self.nwalkers, max(N)], 'i')
30          x_direction = np.zeros([self.nwalkers, max(N)], 'i')
31          p = np.random.random([self.nwalkers, max(N)-1]) # generate random number in [0,1)
32          alpha = self.alpha
33          l = self.l
34          x0 = self.x0
35
36          for n in xrange(self.nwalkers):
37              x[n][0] = x0
38              x_direction[n][0] = +1 if n < int(self.nwalkers/2) else -1
39              for i in xrange(1,max(N)):
40                  if p[n][i-1] < alpha:
41                      x_direction[n][i] = x_direction[n][i-1]
42                  else:
43                      x_direction[n][i] = -x_direction[n][i-1]
44
45                  if x_direction[n][i] > 0:
46                      x[n][i] = x[n][i-1] + x_direction[n][i]*l[0]
47                  else:
48                      x[n][i] = x[n][i-1] + x_direction[n][i]*l[1]
49
50          self.x = x
51          self.N = N
52
53      def calc_ave(self):
54          """ Caluculate the average of displacements after max(N) steps.
55
56          You can call the results by "self.N", "self.x_ave", and "self.x_2_ave"
57          """
58          self.x_ave = np.sum(self.x, axis=0, dtype=np.float32)/self.nwalkers*1.
```

```
59          self.x_2_ave = np.sum(self.x**2, axis=0, dtype=np.float32)/self.nwalkers*1.
60          self.variance_x = self.x_2_ave - self.x_ave**2
61
62      def show(self):
63          """ Show the graph.
64          """
65          fig = plt.figure('random walk',figsize=(8,8))
66
67          x_ave = [self.x_ave[nvalue-1] for nvalue in self.N]
68          x_2_ave = [self.x_2_ave[nvalue-1] for nvalue in self.N]
69          variance_x = [self.variance_x[nvalue-1] for nvalue in self.N]
70
71          ax1 = fig.add_subplot(311)
72          ax1.plot(self.N, x_ave)
73          ax1.set_ylabel(r'$<x(N)>$', fontsize=16)
74
75          ax2 = fig.add_subplot(312)
76          ax2.plot(self.N, x_2_ave)
77          ax2.set_ylabel(r'$<x^{2}(N)>$', fontsize=16)
78
79          ax3 = fig.add_subplot(313)
80          ax3.set_ylabel(r'$<\Delta x^{2}(N)>$', fontsize=16)
81          ax3.plot(self.N, variance_x)
82          ax3.set_xlabel(r'$N$')
83
84          plt.show()
85
86      def caluculate_prob(self,_n):
87          N = max(self.N)
88          x = self.x
89          count_box = np.zeros([N, 2*N+1], 'f')
90          for n in xrange(N):
91              for walker in xrange(self.nwalkers):
92                  count_box[n][N+x[walker][n]] += 1
93          prob = count_box/self.nwalkers
94
95          def show_for_n(_n):
96
97              xmin = -N
98              xmax = N
```

```
99              for _x in xrange(2*N+1):
100                 if prob[_n][_x] != 0:
101                     xmin, xmax = _x-N, N-_x
102                     break
103
104             xmargin = xmax*0.1
105             ymax = np.amax(prob[_n])
106             ymargin = ymax*0.1
107
108             fig = plt.figure('probability')
109             ax = fig.add_subplot(111)
110             ax.grid()
111             ax.set_xlim(xmin-xmargin,xmax+xmargin)
112             ax.set_ylim(0, ymax+ymargin)
113             ax.plot(xrange(-N, N+1), prob[_n])
114             ax.set_xlabel(r'$x$', fontsize=16)
115             ax.set_ylabel(r'$P(x,N)$', fontsize=16)
116             plt.show()
117
118         show_for_n(_n)
119
120 if __name__ == '__main__':
121     rw = HistoryDependentRW()
122     N = xrange(1,512)
123     rw.random_walk_d1(N)
124     rw.caluculate_prob(_n=400)
125     rw.calc_ave()
126     rw.show()
127
```

2.2 継続型ランダムウォークのシミュレーション-実行部 (12-4_history-dependent_rw.py)

```
1  #! /usr/bin/env python
2  # -*- coding:utf-8 -*-
3  #
4  # written by Shotaro Fujimoto, June 2014.
5
6  from HistoryDependentRW import HistoryDependentRW
7
```

```
8  def ex_a(alpha=0.75, caluculate_prob=False, n=400):
9      rw = HistoryDependentRW(alpha=alpha)
10     N = xrange(1,512)
11     rw.random_walk_d1(N)
12     if caluculate_prob:
13         rw.caluculate_prob(_n=n)
14     else:
15         rw.calc_ave()
16         rw.show()
17
18 def ex_b(alpha=0.25, target='nu', N = xrange(1,1001)):
19
20     if not (target == 'nu' or 'D'):
21         print "ArgumentError: target must be 'nu' or 'D'. "
22         return 0
23
24     else:
25
26         if target == 'delta_x2':
27             N = [8,64,256,512]
28
29         rw = HistoryDependentRW(alpha=alpha)
30         rw.random_walk_d1(N)
31         rw.calc_ave()
32
33         if target == 'delta_x2':
34              for n in N:
35                 print '<\Delta x^{2}(%d)> = '%n, str(rw.variance_x[n-1])
36              return 0
37
38         parameter0 = [1.0, 1.0] # initial values for optimize
39
40         def fit_func(parameter0, x, y):
41             a = parameter0[0]
42             b = parameter0[1]
43             residual = y - (a*x+b)
44             return residual
45
46
47         import scipy.optimize as optimize
```

```
48          import numpy as np
49
50          if target == 'nu':
51              import matplotlib.pyplot as plt
52              variance_x = [rw.variance_x[nvalue-1] for nvalue in N]
53              ln_N = np.log(N)
54              ln_variance_x = np.log(variance_x)
55
56              result = optimize.leastsq(fit_func, parameter0, args=(ln_N,ln_variance_x))
57
58              print 'nu =', result[0][0]/2.
59
60              plt.xlabel(r'$N$', fontsize=16)
61              plt.ylabel(r'$<\Delta x^{2}(N)>$', fontsize=16)
62              plt.xscale('log')
63              plt.yscale('log')
64              plt.axis('equal')
65              plt.plot(N, variance_x)
66              plt.show()
67
68          elif target == 'D':
69
70              result = optimize.leastsq(fit_func, parameter0,
71                                        args=(np.arange(1,max(N)+1),rw.x_2_ave)
72                                       )
73              print 'D =', result[0][0]/4.
74
75  def ex_c():
76
77      rw = HistoryDependentRW(alpha=0.75, l=(1,0), nwalkers=1000)
78      N = xrange(1,512)
79      rw.random_walk_d1(N)
80      rw.caluculate_prob(_n=300)
81
82  if __name__ == '__main__':
83
84  #    ex_a(alpha=0.75, caluculate_prob=False, n=400)
85
86  #    ex_b(alpha=0.75,target='D', N = xrange(1,1001))
87      # taget must be 'delta_x2' or 'nu' or 'D'
```

```
88
89      ex_c()
90
```

3 実習課題

a. ”継続型”ランダムウォークでは、遷移、すなわち”ジャンプ”の確率が直前の履歴に依存する。1 次元格子上のランダムウォークを考え、すでに $N-1$ ステップ進んでいるとする。N ステップ目は確率 α で同じ方向に進み、確率 $1-\alpha$ で反対方向に進む。この 1 次元の継続型ランダムウォークのモンテカルロ・シミュレーションを行うプログラムを書け。そして、$< x(N) >$,$< x^2(N) >$,$< \Delta x^2(N) >$、$P(x, N)$ を計算せよ。粒子の初期位置と初期の方向を設定する必要がある。この継続型ランダムウォークで $\alpha = 1/2$ の極限はどうなるか。

上で示したプログラムを用いて、継続型の 1 次元ランダムウォークのシミュレーションを行った。図 1 に、N に対する $< x(N) >$ と $< x^2(N) >$、$< \Delta x^2(N) >$ のグラフを示した。このグラフから、$< x(N) >$ は、N に依存せず、零となることが分かる。また、$< x^2(N) >$ と $< \Delta x^2(N) >$ は N に比例していることが分かる。

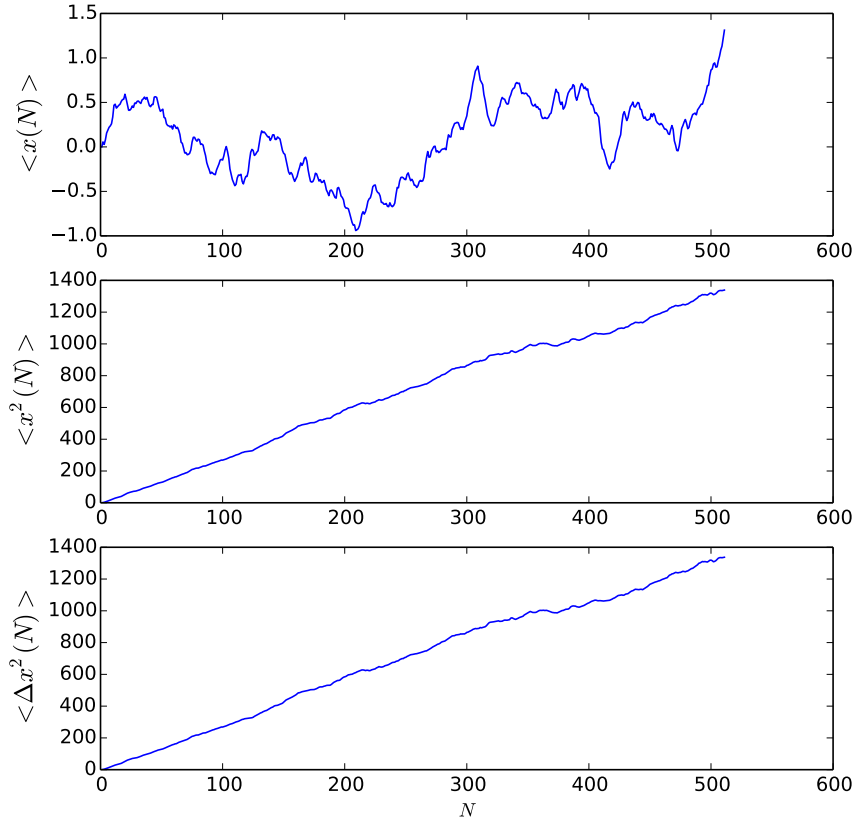

図 1　$\alpha = 0.75$ のとき、N に対する $< x(N) >$ と $< x^2(N) >$、$< \Delta x^2(N) >$ のグラフ

次に、N ステップの移動の後に粒子が位置 x に見出される確率 $P(x, N)$ の計算を行い、$N = 400$ とした時の $P(x, 400)$ のグラフを、横軸を x、縦軸を確率 P としてプロットしたものを図 2 に示した。このグラフから、確率分布はガウス分布の定数倍の連続関数に従うと予想され (問題 7.6 による)、先ほどの図 1 の結果からも、N が大きくなるほど、確率分布の裾の拡がりは $\sqrt{N}$ で大きくなることが分かる。このとき x が奇数の時の確率はいつも零であるが、これは N が偶数であることに起因するものであって、N を奇数に取ると、N ステップ後に偶数の x に到達する確率は零となる。また、この効果によって、確率分布がガウス分布を 2 倍の高さにしたものになっている、とも説明できる。

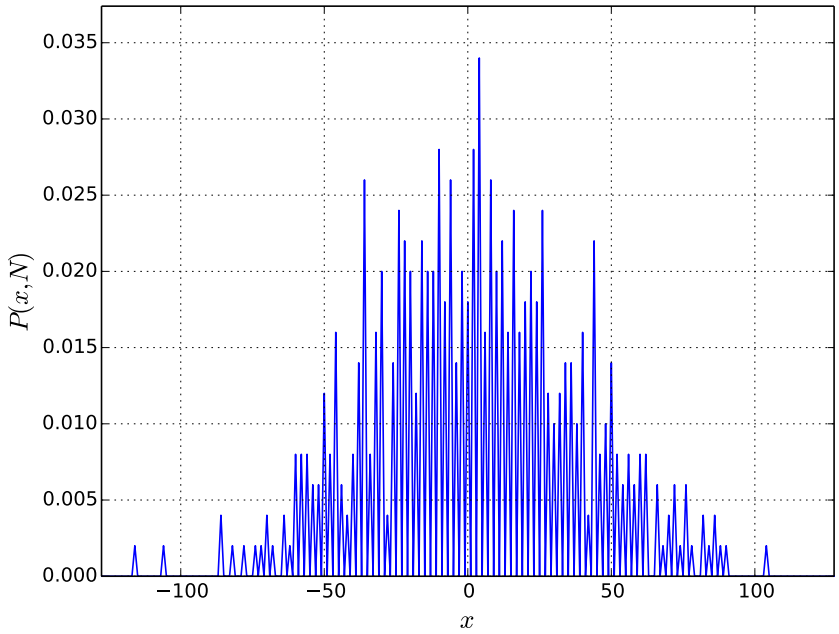

図 2　$N = 400$ のとき、x に対する $P(x, N)$ のグラフ

また、この継続型ランダムウォークの $\alpha = 0.5$ の極限を考えたとすると、これは N ステップ目の遷移の確率が $N - 1$ ステップ目の確率に依存しないことを意味するので、単純な 1 次元ランダムウォークで $p = q = 0.5$ とした時と等価になるはずである。実際に、作成したプログラムで $\alpha = 0.5$ とした時の N に対する $< x(N) >$ と $< x^2(N) >$、$< \Delta x^2(N) >$ のグラフを作成し、これを図 3 に示したが、確かに単純なランダムウォークの分散について得られた関係

$$< \Delta x^2(N) >= l^2 N \tag{1}$$

を満たしていることが分かる。

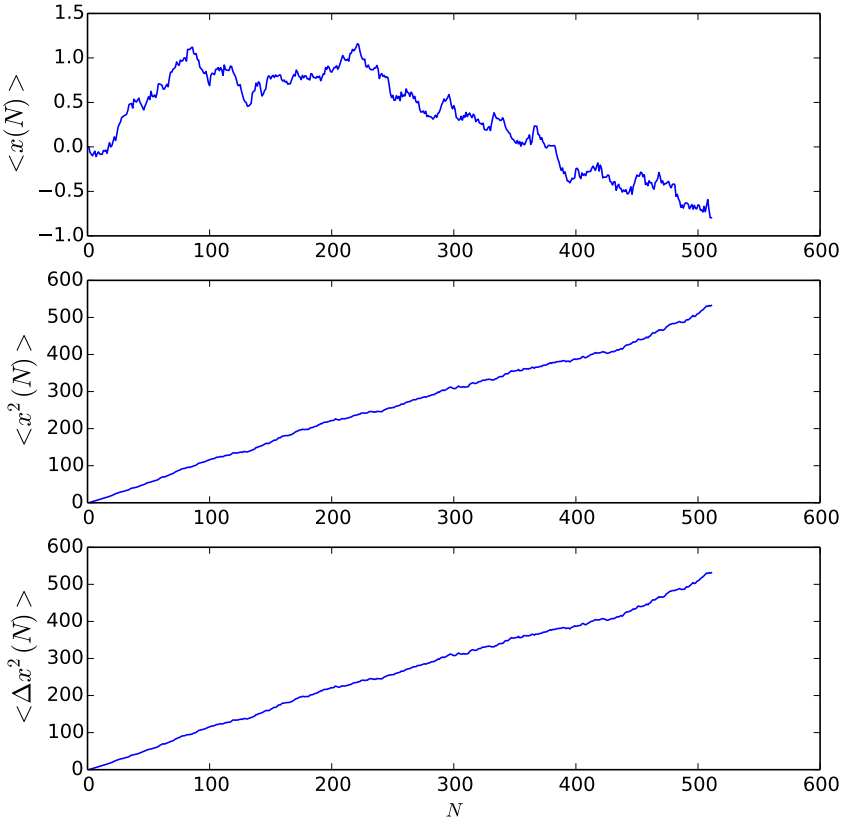

図 3　$\alpha = 0.5$ のとき、N に対する $< x(N) >$ と $< x^2(N) >$、$< \Delta x^2(N) >$ のグラフ

b. $\alpha = 0.25$ と $\alpha = 0.75$ の場合を考えて、$N = 8, 64, 256, 512$ について、$< \Delta x^2(N) >$ を求めよ。さらに大きい N についての $< \Delta x^2(N) >$ の N に対する両対数プロットから ν の値を求めよ。ν は α に依存するか。$\nu \simeq 1/2$ なら、$\alpha = 0.25$ と 0.75 について自己拡散係数 D を求めよ。$D(\alpha \neq 0.5)$ が $D(\alpha = 0.5)$ より大きくなる (または小さくなる) 物理的理由を述べよ。

$\alpha = 0.25$ と $\alpha = 0.75$ のそれぞれの場合について、$N = 8, 64, 256, 512$ に関して $< \Delta x^2(N) >$ を計算し、表 1、表 2 にまとめた。

表 1　$\alpha = 0.25$ のときの $< \Delta x^2(N) >$

N	$< \Delta x^2(N) >$
8	2.79998
64	19.6896
256	85.4902
512	186.444

表 2　$\alpha = 0.75$ のときの $< \Delta x^2(N) >$

N	$< \Delta x^2(N) >$
8	17.0156
64	178.29
256	777.731
512	1575.72

そして、さらに大きい $N(100 \leq N \leq 10000)$ について、$\alpha = 0.25$ のとき $< \Delta x^2(N) >$ の N に対する両対数プロットを図 4 に示した。また、このときの傾きの大きさの 1/2 が ν であったから、グラフの直線の傾きから ν の値を求め、その値は $\nu = 0.504529949219$ であった。また、$\alpha = 0.75$ の場合も計算を行ったが、このときの値は $\nu = 0.503435215609$ であり、他のいくつかの試行からも ν は α に依存しないことが分かる。

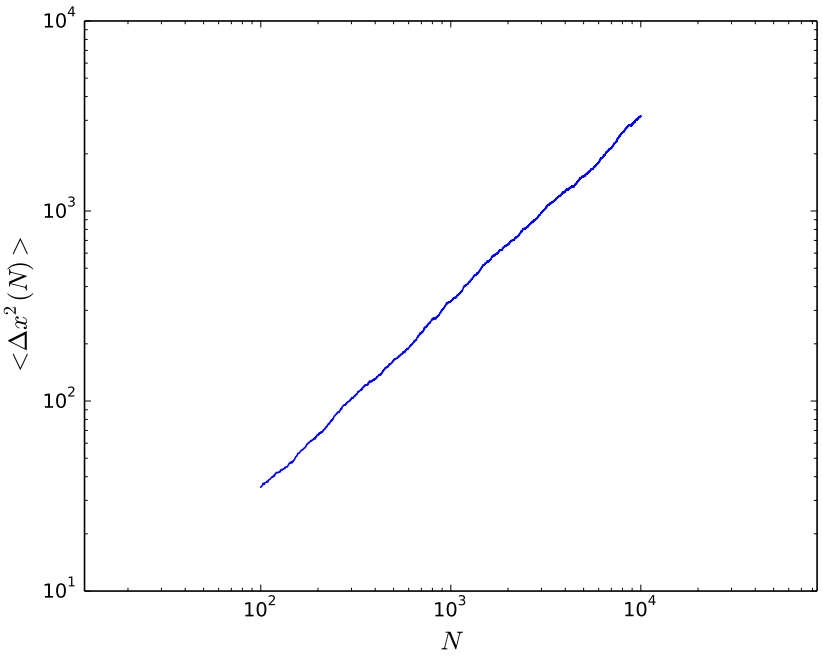

図 4　$< \Delta x^2(N) >$ の N に対する両対数プロット ($\alpha = 0.25$

次に、$\alpha = 0.25$ と $\alpha = 0.75$ について自己拡散係数 D を求めることにする。付録に示したように、d 次元空間では

$$< x^2(t) >= 2dDt \tag{2}$$

であり、また t は Δt を時間の刻み幅として $t = N\Delta t$ と書けるから、今の場合 $d = 2$ であり、$\Delta t = 1$ として

$$< x^2(t) >= 4DN \tag{3}$$

となる。この式から拡散係数 D を求めると、$\alpha = 0.25$ のとき $D = 0.0800663199903$、$\alpha = 0.75$ のとき $D = 0.787805747407$ となった。また、$\alpha = 0.5$ のときは $D = 0.258759185637$ であった。$D(\alpha = 0.75)$ は $D(\alpha = 0.5)$ より大きく、$D(\alpha = 0.25)$ は $D(\alpha = 0.5)$ より小さくなっており、これは α の定義を思い出せば理解できることである。α は 1 回前の進行方向と同じ方向に進む確率であるから、この値が小さいということは、1 回前の進行方向とは逆の方向に進む確率が高いということを意味し、原点の付近で周期 2 ステップで振動するような場合に近くなることが分かる。逆に α の値が大きいときには、初期値によって定められた方向に進みやすく、その後折り返した後も継続的に長い距離同じ方向に進むことになる。結果として粒子の移動した距離の分散は大きくなることが分かる。

c. 継続型ランダムウォークは、状態が直前の遷移によって定義される多重状態のランダムウォークの一例とみなすことができる。上の例では、粒子は 2 つの状態のうちのどちらか一方にいて、各ステップについて同じ状態にとどまる確率と遷移する確率は、それぞれ α と $1 - \alpha$ である。この 2 状態ランダムウォークの最も初期の適用例の 1 つにクロマトグラフィー柱の拡散の問題がある。クロマトグラフィー柱の中の分子は運動状態 (一定速度 v) か、捕らえられた状態 (速度 0) のどちらかにあるとする。この場合、各ステップで位置を ± 1 変化させる代わりに、各ステップで位置を $+v$ か 0 だけ変化させる。実験的に興味のある量は、分子が N ステップの後に距離 x だけ移動する確率 $P(x, N)$ である。$v = 1$, $\alpha = 0.75$ として、$P(x, N)$ の定性的な振る舞いを調べよ。分子はどちらの状態にも拡散することができないのに、分子の有効拡散係数は定義できる理由を説明せよ。

先程まで考えていた問題は、2 状態ランダムウォークの例とみなすことができて、その 2 つの状態とは右に 1 だけ進む状態と左に 1 だけ進む状態である。また、各ステップについて同じ状態にとどまる確率と遷移する確率は、それぞれ α と $1 - \alpha$ である。2 状態ランダムウォークの最も初期の適用例の 1 つにクロマトグラフィー柱の拡散の問題があり、クロマトグラフィー柱の中の分子は運動状態 (一定速度 v) か、捕らえられた状態 (速度 0) のどちらかにあるとする。この場合、各ステップで位置を $+v$ か 0 だけ変化させる。以上のようなプログラムを $v = 1$, $\alpha = 0.75$ として実行し、その結果得られた $P(x, N)$ のグラフを図 5 に示した (x 軸の方向が逆転していることに注意)。このグラフや、他の N について行ったシミュレーションによって、N ステップ後に粒子のいる位置 x の期待値は $N/2$ であり、また確率分布はガウス分布に近い形となっていることが分かる。問題 a で得られた確率 $P(x, N)$ との違いは、期待値が 0 でない値をもつことと、N の偶奇による確率 0 の位置が存在しないことである。また、この条件で分子はどちらの状態にも拡散することはできないが、問題 b で考察したように、拡散係数 D は $< x^2(N) >$ と関連付けられ、この意味において有効拡散係数を定義することができる。

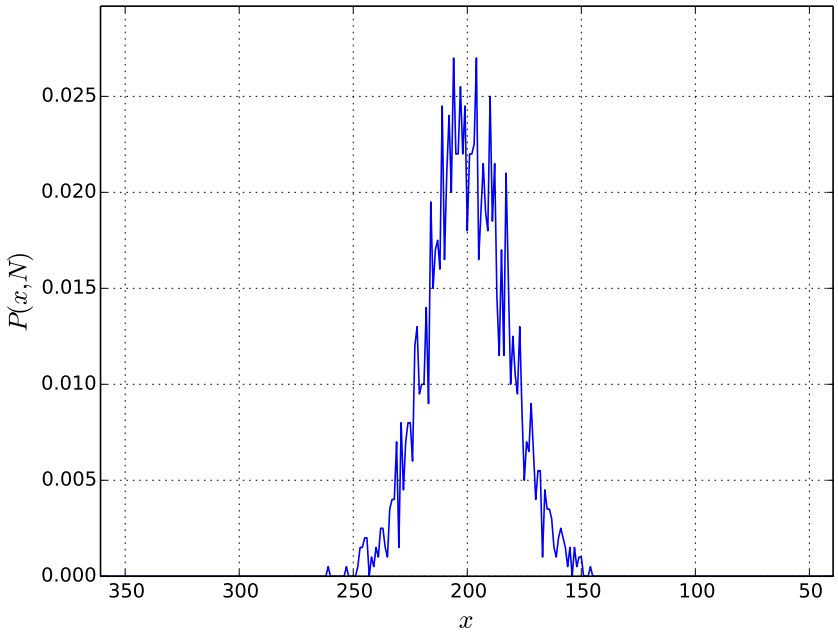

図 5　$N = 400$ のとき、x に対する $P(x, N)$ のグラフ

4　まとめ

継続型ランダムウォークと、それに関連して、2 状態ランダムウォークについて理解することができた。

5　付録-ランダムウォークと拡散方程式

ランダムウォークと拡散方程式の関係を理解するために、拡散方程式から $< x(t) >$ が 0 になることと $< x^2(t) >$ が t に比例することが導かれることを示そう。拡散方程式は

$$\frac{\partial P(x,t)}{\partial t} = D\frac{\partial^2 P(x,t)}{\partial x^2} \tag{4}$$

である。$< x(t) >$ と $< x^2(t) >$ の t 依存性を式 (4) から導くために、任意の x の関数の平均を

$$< f(x,t) >= \int_{-\infty}^{\infty} f(x)P(x,t)dx \tag{5}$$

で表す。そうすると変位の平均は

$$< x(t) >= \int_{-\infty}^{\infty} xP(x,t)dx \tag{6}$$

で与えられる。式 (6) の右辺の積分を実行するために、式 (4) の両辺に x を掛けて形式的に x について積分する。

$$\int_{-\infty}^{\infty} x\frac{\partial P(x,t)}{\partial t}dx = D\int_{-\infty}^{\infty} x\frac{\partial^2 P(x,t)}{\partial x^2}dx \tag{7}$$

左辺は

$$\int_{-\infty}^{\infty} x\frac{\partial P(x,t)}{\partial t}dx = \frac{\partial}{\partial t}\int_{-\infty}^{\infty} xP(x,t)dx = \frac{\partial}{\partial t} < x > \tag{8}$$

と表される。式 (7) の右辺は部分積分をすると、求めている形に書き換えられる。

$$D\int_{-\infty}^{\infty} x\frac{\partial^2 P(x,t)}{\partial x^2}dx = Dx\frac{\partial P(x,t)}{\partial x}\Big|_{x=-\infty}^{x=\infty} - D\int_{-\infty}^{\infty} x\frac{\partial P(x,t)}{\partial x}dx \tag{9}$$

$P(x=\pm\infty,t)=0$ であり、$x=\pm 1$ では P のすべての空間微分は 0 なので、式 (9) の右辺第 1 項は 0 になる。第 2 項は、積分すると $D[P(x=\infty,t)-P(x=\infty,t)]$ となるので、やはり 0 になる。したがって

$$\frac{\partial}{\partial x} < x >= 0 \tag{10}$$

が得られ、$< x >$ は時間によらない定数になる。$t=0$ で $x=0$ なので、結局、すべての t について $< x >= 0$ が成り立つという結論になる。$< x^2(t) >$ を計算するには 2 度の部分積分が必要であり、そうすると

$$\frac{\partial}{\partial x} < x^2(t) >= 2D \tag{11}$$

したがって

$$< x^2(t) >= 2Dt \tag{12}$$

が得られる。こうして、ランダムウォークと拡散方程式が同じ時間依存性を持つことがわかる。d 次元空間では、$2D$ は $2dD$ に置き換えられる。

6 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.