問題 12.5 制限されたランダムウォーク

早稲田大学先進理工学部物理学科 B4 藤本將太郎

2014/06/10

1　シミュレーションの目的

いろいろなタイプの制限や境界がある場合に、それらがランダムウォークに与える効果をより一般的に議論する。

2　作成したプログラム

本シミュレーションを行うにあたり作成したプログラムを以下に示す。

2.1　制限要素を含めた 1 次元のランダムウォーク (RestrictedRW.py)

このプログラムでは、1 次元の格子上に、捕獲格子点や反射点を定義し、その中に初期値を設定した際のランダムウォークの性質を調べることができる。例えば、restricted_rw_erase を用いると、捕獲格子点 (粒子がその点に接触すると粒子が消滅する) を $x = 0, a$ の位置に定義することができ、また restricted_rw_reflect を用いると、$x = -a, a$ の位置に置いた場合をシミュレーションできる。前者に関しては、全ての粒子が消滅するまで計算を続けるように設定してあり、粒子の消滅した時間 (ステップ数) は tau に記録されるようになっている。また、消滅した点に関しては、それ以後計算を行わない。後者ではステップ回数 N を明示的に与えて、その回数だけループが続くようにしてある。caluculate_prob は問題 12.4 で用いたものとほぼ同様である。

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

import numpy as np
import matplotlib.pyplot as plt


class RestrictedRW:

```

```
    def __init__(self, walker=1000, alpha=0.5):

        self.walker = walker  # walker
        self.alpha = alpha

    def restricted_rw_erase(self, a=5, x0=2):

        self.a = a
        self.x0 = x0  # integer between (0,a)
        self.x = np.tile([self.x0], self.walker).reshape((self.walker, 1))
        x = self.x
        self.tau = np.zeros(self.walker, 'int32')

        while min(self.tau) == 0:
            # random walk
            x = np.insert(x, len(x[0]), 0, axis=1)

            for m in xrange(self.walker):
                if not x[m][-2] == 0:
                    p = np.random.rand()
                    if p < self.alpha:
                        x[m][-1] = x[m][-2] + 1
                    else:
                        x[m][-1] = x[m][-2] - 1

                    if x[m][-1] == 0 or x[m][-1] == self.a:
                        x[m][-1] = 0
                        self.tau[m] = len(x[m]) - 1
        else:
            # caluculate
            self.ave_tau = np.average(self.tau)
            self.std_tau = np.std(self.tau)

    def restricted_rw_reflect(self, N=10, a=5):

        self.a = a
        self.x0 = 0
        self.N = N
        self.x = np.tile([self.x0], self.walker).reshape((self.walker, 1))
        x = self.x
```

```

        for n in xrange(1, N):
            x = np.insert(x, len(x[0]), 0, axis=1)
            for m in xrange(self.walker):
                if x[m][-2] == -self.a:
                    x[m][-1] = -self.a + 1
                elif x[m][-2] == self.a:
                    x[m][-1] = self.a - 1
                else:
                    p = np.random.rand()
                    if p < self.alpha:
                        x[m][-1] = x[m][-2] + 1
                    else:
                        x[m][-1] = x[m][-2] - 1
        self.x = x

    def random_walk_d1(self, N=10):

        x = np.zeros([self.walker, N], 'i')

        # generate random number in [0,1)
        p = np.random.random([self.walker, N - 1])
        prob = self.alpha
        l = 1
        x0 = 0

        for n in xrange(self.walker):
            x[n][0] = x0
            for i in xrange(1, N):
                d = +l if p[n][i - 1] < prob else -l
                x[n][i] = x[n][i - 1] + d
        self.x = x
        self.N = N
        self.a = N

    def caluculate_prob(self, _n=6):

        x = self.x

        count_box = np.zeros([self.N, 2 * self.a + 1], 'f')
```

```
        for n in xrange(self.N):
            for m in xrange(self.walker):
                count_box[n][self.a + x[m][n]] += 1
        prob = count_box / self.walker

        def show_for_n(_n):

            xmin = -self.a
            xmax = self.a

            for _x in xrange(2 * self.a + 1):
                if prob[_n][_x] != 0:
                    xmin, xmax = _x - self.a, self.a - _x
                    break

            xmargin = xmax * 0.1
            ymax = np.amax(prob[_n])
            ymargin = ymax * 0.1

            fig = plt.figure('probability')
            ax = fig.add_subplot(111)
            ax.grid()
            ax.set_xlim(xmin - xmargin, xmax + xmargin)
            ax.set_ylim(0, ymax + ymargin)
            ax.plot(xrange(-self.a, self.a + 1), prob[_n])
            ax.set_xlabel(r'$x$', fontsize=16)
            ax.set_ylabel(r'$P(x,N)$', fontsize=16)
            plt.show()

        show_for_n(_n)

if __name__ == '__main__':

    def test(target):
        rw = RestrictedRW(walker=1000)
        if target == 'erase':
            from math import sqrt
            rw.restricted_rw_erase(a=5, x0=2)
            print rw.ave_tau, '+-', rw.std_tau / sqrt(rw.walker)
        elif target == 'reflect':
```

```
                rw.restricted_rw_reflect(N=1000, a=100)
                rw.caluculate_prob(_n=900)
            else:
                pass

    test('erase')
```

2.2 制限要素を含めた 1 次元のランダムウォーク (12-5_restricted_rw.py)

先の RestrictedRW を利用して実際に実行するプログラム。3D グラフの描画や、条件を変えた際の振る舞いはどうなるかを調べることができる。

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

import numpy as np
import matplotlib.pyplot as plt
from math import sqrt
from RestrictedRW import RestrictedRW as RRW


def plot3d_a_x_tau(amin=0, amax=10):

    list_a = np.repeat(
        np.array(range(amin, amax + 1))[:, np.newaxis], amax + 1, axis=1)
    list_x = np.repeat(np.array([range(0, amax + 1)]), amax + 1, axis=0)
    list_tau = np.zeros([amax + 1, amax + 1], 'f')

    from mpl_toolkits.mplot3d import Axes3D
    for _a in range(amin + 2, amax + 1):
        for _x in range(amin + 1, _a - 1):
            rw.restricted_rw_erase(a=_a, x0=_x)
            list_tau[_a][_x] = rw.ave_tau

    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    ax.plot_wireframe(list_a, list_x, list_tau, rstride=1, cstride=1)
    ax.set_xlabel(r'$a$', fontsize=16)
```

```
    ax.set_ylabel(r'$x_{0}$', fontsize=16)
    ax.set_zlabel(r'$\tau$', fontsize=16)
    plt.show()

if __name__ == '__main__':

    rw = RRW(walker=1000)
# comment out to use
#
#    rw.restricted_rw_erase(a=5, x0=2)
#    print rw.ave_tau, '+-', rw.std_tau/sqrt(rw.walker)
#
    plot3d_a_x_tau(amax=15)
#
#    rw.restricted_rw_reflect(N=1000, a=100)
#    rw.caluculate_prob(_n=900)
#
#    rw.random_walk_d1(N=1000)
#    rw.caluculate_prob(_n=900)
#
```

3 実習課題

a. $x = 0$ と $x = a(> 0)$ に”捕獲”格子点がある 1 次元格子を考える。粒子は位置 $x_0(0 < x_0 < a)$ から動き始め、隣接した左右の格子点に当確率で進むとする。粒子が捕獲格子点に到達すると粒子は消滅する。モンテカルロ・シミュレーションを行い、粒子が捕まえられるまでの平均ステップ数 τ(第 1 経過時間) が

$$\tau = (2D)^{-1} x_0 (a - x_0) \tag{1}$$

で与えられることを確かめよ。ここで D は捕獲格子点がない場合の自己拡散係数であり、平均はすべてのランダムウォークについて行う。

$x = 0, a$ に捕獲格子点のある 1 次元格子上で、粒子が捕らえられるまでの平均時間を算出する。粒子が左右に動く確率は等しく ($\alpha = 0.5$)、捕獲格子点に接触した粒子は消滅する。粒子の数 (試行回数と見ても良い) を walker $= 1000$ としたとき、捕獲格子点の位置 a と出発地点 $x_0(0 < x_0 < a)$ に対する平均ステップ数 τ を計算した。この計算をまとめて 3 次元空間にプロットした (図 (1))。横から見た図 (2) を見るとわかりやすいが、τ は x_0 の 2 次で変化し、また a の大きさには比例している。これらの結果から、捕獲格子点のないとときの自己拡散係数 $D(D = 0.5)$ を用いて、τ は

$$\tau = (2D)^{-1} x_0 (a - x_0) \tag{2}$$

で与えられることが確かめられる。

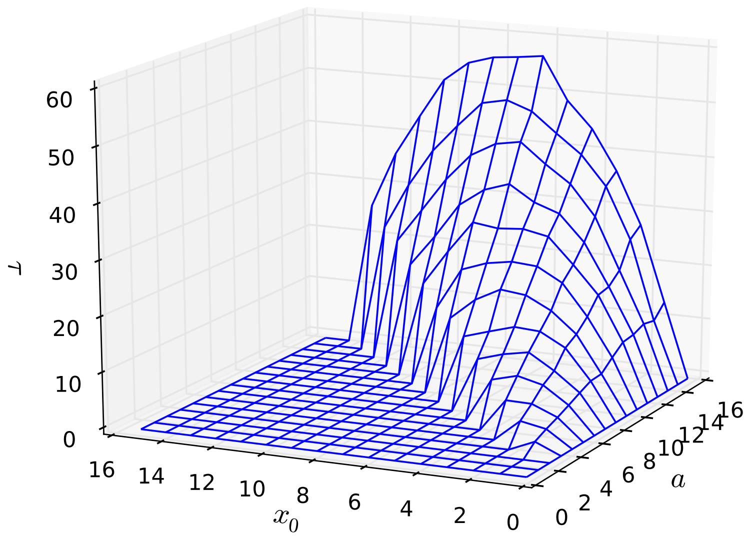

図 1　捕獲格子点の位置 a と初期位置 $x_0(0 < x_0 < a)$ に対する平均ステップ数 τ

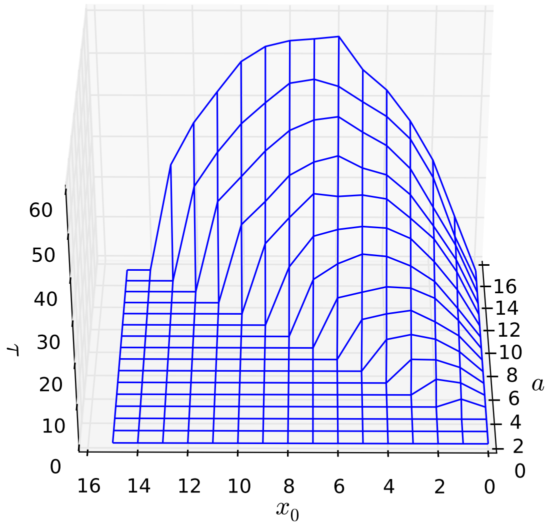

図 2　図 1 を a 軸と平行に $a = 0$ から見た場合

b. 捕獲格子点のあるランダムウォークのモデルは、物性物理で重要な役割を担っている。例として固体中のエネルギー輸送についての次の理想化されたモデルを考える。固体を、 母体の格子点と捕獲格子点の 2 種類からなる格子とみなす。入射した光子は母体の格子点で吸収され、その母体分子を励起する。その時の励起エネルギー、すなわち励起子は、母体のどれか 1 つの隣接格子点にランダムに移り、最初の励起分子は基底状態に戻る。そのようにして励起子は捕獲格子点に達するまで格子中を動き回る。そこで励起子は捕らえられ、化学反応を起こす。

このようなエネルギー輸送のモデルの簡単なものの 1 つに、格子上に捕獲格子点が周期的に配置された 1 次元格子モデルがある。捕獲格子点は規則的な間隔で配置されているので、無限に長い格子上のランダムウォークの問題はリング状の格子でのランダムウォークの問題に置き換えることができる。N 個の母体の格子点、すなわち N 個の非捕獲格子点と、1 個の捕獲格子点からなるリングを考える。粒子がどの母体格子点から出発する確率も等しく、等しい確率で隣接格子点に移動する場合、平均生存時間 τ(捕獲格子点に達するまでの平均のステップ数) の N-依存性はどうなるか。追加のシミュレーションを行うのではなく、設問 a の結果を使え。

設問 a の結果をこの場合に当てはめて考えると、捕獲格子点の位置を $x = 0$ と $x = a = N + 1$ とすればよいので、平均生存時間 $\tau(x_0, N)$ は

$$
\begin{aligned}
\tau(x_0, N) &= \frac{1}{2D} x_0 (a - x_0) \\
&= (N+1)x_0 - x_0^2
\end{aligned}
$$

である。粒子がどの母体格子点から出発する確率も等しいので、$x_i = i (1 \le i \le N)$ とすると

$$
\begin{aligned}
\tau(N) &= \frac{1}{N}\sum_{i=1}^{N}\left((N+1)x_0 - x_i^2\right) \\
&= \frac{1}{N}\left[(N+1)\sum_{i=1}^{N}x_i - \sum_{i=1}^{N}x_i^2\right] \\
&= \frac{1}{N}\left[(N+1)\frac{1}{2}N(N+1) - \frac{1}{6}N(N+1)(2N+1)\right] \\
&= \frac{1}{6}(N+1)\left[3N+3-(2N+1)\right] \\
&= \frac{1}{6}(N+1)(N+2)
\end{aligned}
$$

と計算することができる。

c. $x=-a$ と $x=a$ に反射格子点がある 1 次元格子を考える。たとえば、粒子が $x=a$ の反射点にくると、次のステップで $x=a-1$ に反射されるとする。粒子は $t=0$ に $x=0$ を出発し、等確率で隣接格子点に進むとする。モンテカルロ法のプログラムを書いて、N ステップの後に粒子が x の位置にいる確率 $P(x,N)$ を求めよ。また、得られた $P(x,N)$ の形を、反射する”壁”のない場合と比較せよ。N と a が同程度の大きさのとき、これら 2 つの確率分布は区別できるか。どの N の値で初めて、2 つの分布を区別できるようになるか。

$x=-a=-100$ と $x=a=100$ に反射格子点があり、粒子が $t=0$ に $x=0$ を出発して、等確率で隣接格子点に進むとき、N ステップの後に粒子が x の位置にいる確率 $P(x,N)$ を求めて、これを図 3 に示した ($N=900$)。また、反射格子点のない場合にも同じようにして確率 $P(x,N)$ を求めた (図 4)。これらの図を比較したとき、反射格子点の有無を見分けることはできない。というのは、反射格子点のないときの確率 P の x に対する広がりの大きさが 100 程度であり、これより大きい a を設定したところで、そもそも到達できないためにグラフの形に差異は生まれない。初めて 2 つの分布を区別できるようになる N の値は、正規分布において 99 %信頼区間を算出するときのように、標準偏差 (今の場合 $\sigma=\sqrt{N}$) に 2.56 を掛けた値が 100 となるような N であると考えられる。したがってこの方法で計算を行うと、およそ $N=1600$ で、$P(x,N)$ の面積の 99 %が $-a$ から a の間に含まれることになり、このとき残りの 1 %が境界での反射の影響を受けることになる。実際に、$N=1600$ のときに反射格子点のある場合とない場合の 2 つの場合についてシミュレーションを行い、図 5、6 に示した。この 2 つの確率分布は区別することができると言えるだろう。また、これよりもさらに N が大きいときには、反射格子点の位置で折り返して重ねたものとなっていることも確認できる (図 7)。

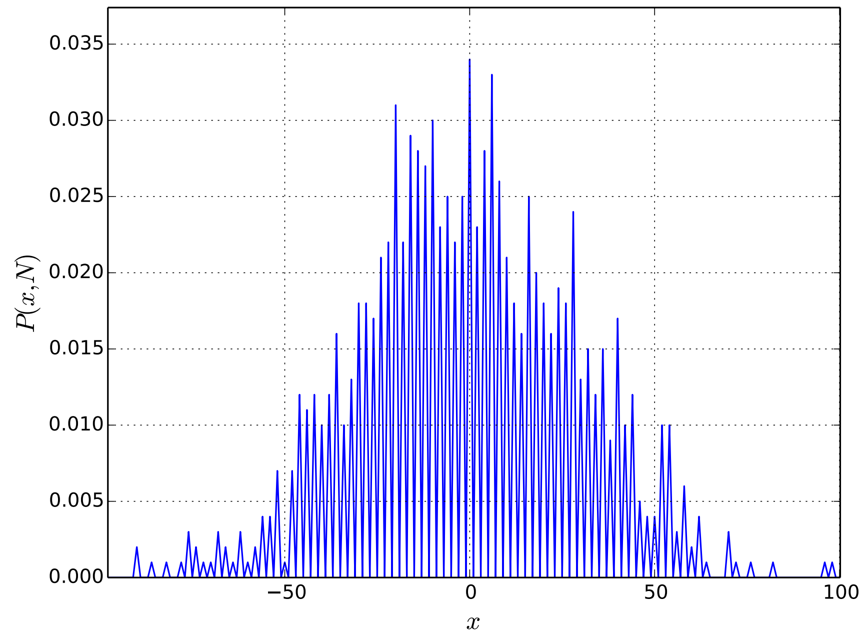

図 3　$N = 900$ のとき、$x \sim$ に対する $P(x,N)$ のグラフ (反射格子点有:$a = 100$)

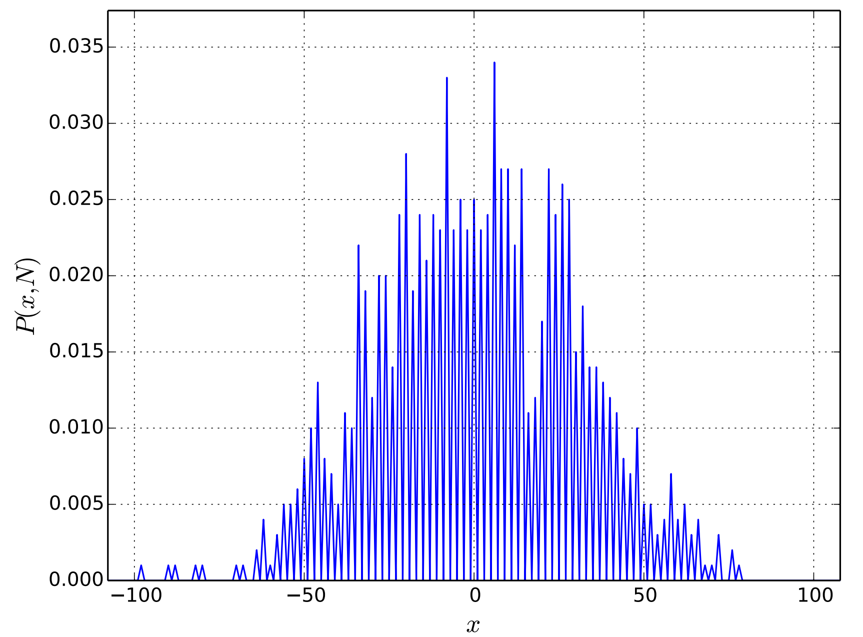

図 4　$N = 900$ のとき、x に対する $P(x,N)$ のグラフ (反射格子点無)

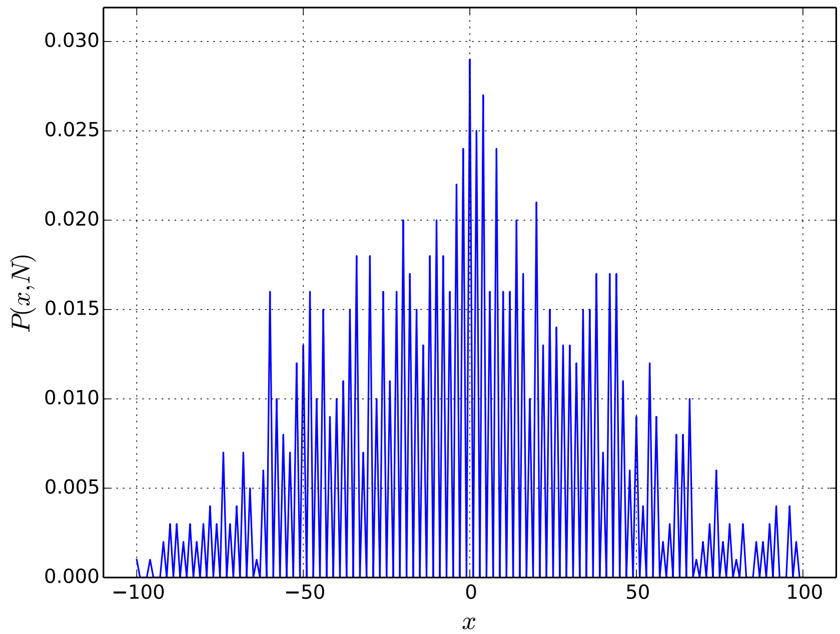

図 5　$N = 1600$ のとき、x に対する $P(x, N)$ のグラフ (反射格子点有:$a = 100$)

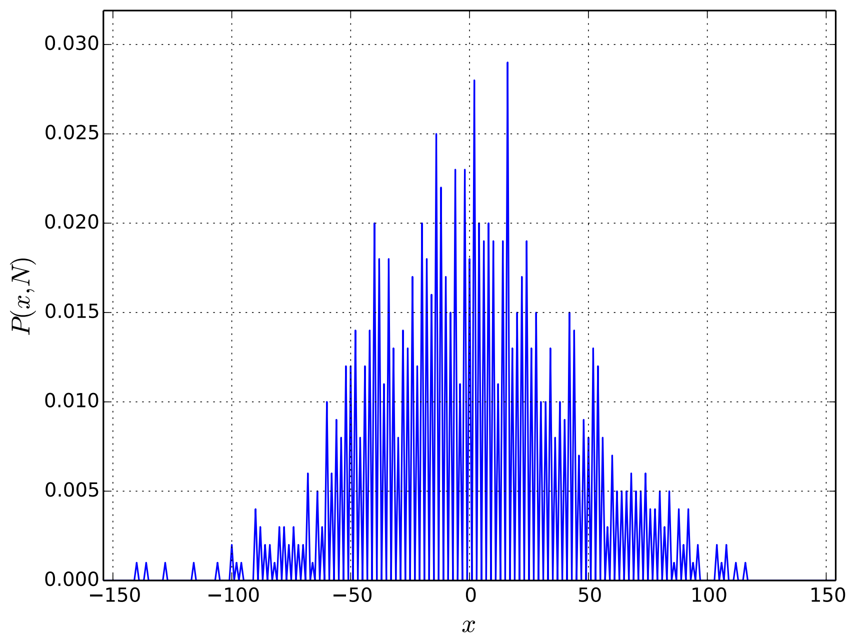

図 6　$N = 1600$ のとき、x に対する $P(x, N)$ のグラフ (反射格子点無)

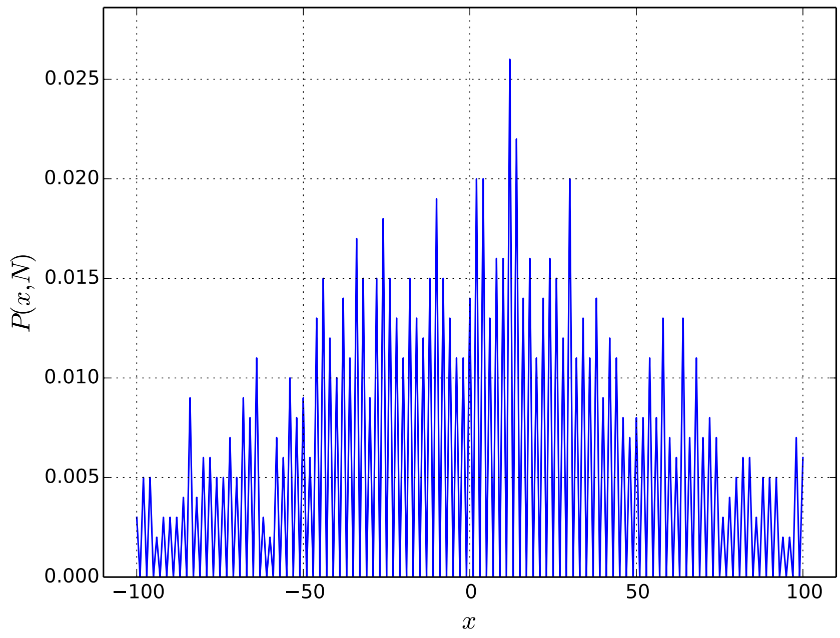

図 7　$N = 2500$ のとき、x に対する $P(x,N)$ のグラフ (反射格子点有:$a = 100$)

4　まとめ

制限を設けた場合のランダムウォークについて調べることができた。プログラムの内容に関しては、3D プロットを試し、また数値計算に用いる行列の操作についていくつかの方法を学んだ。

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.