# 計算機実習 問題 14.8 迷路の蟻

早稲田大学先進理工学部物理学科 B4 藤本將太郎

2015 年 5 月 29 日

## 1　シミュレーションの目的

第 7 章と第 12 章では，完全な格子系と単純な連続系でのランダムウォークを考えた．そこでは，ランダムウォークにおける平均 2 乗変位 $\langle R^2(t)\rangle$ は十分大きな $t$ に対して $t$ に比例することを知った (単純なランダムウォークではこれはすべての $t$ に対して成り立つ)．ここでは，ランダムウォークが不規則な格子上，たとえば，パーコレーション・クラスターの占有された格子点に制限される場合を考えることにしよう．$\langle R^2(t)\rangle$ の漸近的な $t$ 依存性はこの場合どうなるだろうか．この簡単な，パーコレーション・クラスター上のランダムウォークのモデルは”迷路の蟻”の問題として知られている．

不規則格子上のランダムウォークに興味を持つのには少なくとも 2 つの理由がある．規則格子上のランダムウォークが拡散の簡単なモデルであるように，不規則格子上のランダムウォークは乱れた媒質中の拡散や輸送の一般的問題の簡単な例である．興味のある物質の多くは結晶性でなく不規則なので，迷路の蟻の運動に関係づけることができる物理現象は多い．乱れた媒質中の拡散に興味をもつもう 1 つの理由は，拡散係数が媒質の電気伝導度に比例するということである．伝導度と拡散係数の関係はアインシュタイン (Einstein) の関係として知られている．

迷路の蟻の問題における普通の定式化では，ランダムウォークをする歩行者 (蟻) を，確率 $p$ で生成されたパーコレーション・クラスターの占有された格子点の 1 つにランダムに置く．各分割時間に，蟻は 4 通り (正方格子上で) の結果が出るコインを投げる．結果が，占有された格子点での一歩に対応するならば，蟻は動く．そうでなければ蟻はその位置を動かない．どちらの場合でも時刻 $t$ は 1 単位時間だけ増加する．興味のある主な量は蟻の時刻 $t=0$ および $t$ の位置の間の距離の 2 乗 $R^2(t)$ である．蟻の平均 2 乗変位 $\langle R^2(t)\rangle$ を得るために，多くのパーコレーション・クラスターを用いるだけでなく，同じクラスターで初期位置をいろいろ変えた多くのランダムウォークを生成することができる．$\langle R^2(t)\rangle$ は $p$ と $t$ にどのように依存するか．拡散の法則はフラクタルな格子 (たとえば，$p=p_c$ でのパーコレーション・クラスター) 上でどのように変更されるだろうか．問題 14.8 ではこれらの疑問について考える．

## 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 パーコレーション・クラスター内をランダムウォークする粒子のシミュレーション

このプログラムでは，クラス Percolation でパーコレーション・クラスターを作成・描画し，クラス Ant 内の関数 rw_d2 ではそのパーコレーション・クラスター上をランダムウォークする蟻を再現する．クラス TopWindow はこれまで使用してきたものと基本的に変更はないが，クラスとしての機能をきちんと独立させ，汎用性を持たせた．クラス Common は関数 plot_graph と fitting を含んでいる．クラス Main はすべてのクラスを内部から呼び出し，ダイアログからの各操作を定義している．

クラス Percolation は，以前まで使用していたものとはほとんど変更点はないが，パーコレートしていないクラスターが得られる事のないように，周辺の点 nnsite がなくなって成長をやめたクラスターに関しては，再帰的に実行を繰り返すようにしてある．ただし，$p < p_c$ の場合があるので，この試行を 30 回繰り返してもパーコレーション・クラスターの得られなかったものについては，その時点で得られたクラスターを返すこととする．

次に今回新たに作成した関数 rw_d2 は，perc_cluster で得られたパーコレーション・クラスター上で粒子をランダムウォークさせるものである．4 方向を選ぶ確率は等しいとし，そこで決定された進行方向が占有サイトであるときのみ粒子の位置を更新して，描画モードのときは粒子の位置を再描画する．$p$ と $N$ が大きいとき，粒子が格子から飛び出すことがあるので，この場合はその過程は止まっていると見なすこととした．

クラス TopWindow では，先程述べたように関数の引数としてすべての内容を記述するようにし，独立した関数の形をなすようになった．行数も削減されている．呼び出し方は，引数として (' ボタンの表示する文字列', ボタン押下時実行する関数) のタプルを並べて代入するだけである．タプルでくくられた部分は同じフレーム内に配置され，フレームとフレームの間にはスペースが挿入される．機能毎に分けて見やすくすることを想定している．

$\langle R^2(t)\rangle$ の計算時には，描画を行うと時間がかかるため，これは行わず，したがって calculate_R_2 で rw_d2 を呼び出すときは，関係のない引数をすべて 0 にしてある．

raw_input を使用している関係で，標準入力を受け付ける環境でない環境で calculate D_s(p) を押すと，フリーズするおそれがあるので注意する．

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

from Tkinter import *
import numpy as np
import scipy.optimize as optimize
import matplotlib.pyplot as plt
import sys
import time

class Percolation:
```

```python

    def __init__(self, L=61):
        self.sub = None
        self.L = L
        self.count = 0

    def perc_cluster(self, p=0.7):
        if p > 1 or p < 0:
            raise ValueError("site occupation probability must be 0 <= p <= 1")
        self.p = p
        self.lattice = np.zeros([self.L+2, self.L+2], dtype=int)
        self.lattice[:1, :] = self.lattice[:, :1] = -1
        self.lattice[self.L+1:, :] = self.lattice[:, self.L+1:] = -1
        self.center = int(self.L/2) + 1
        self.lattice[self.center, self.center] = 1
        nextseed = [(self.center, self.center)]
        if self.sub is None or not self.sub.winfo_exists():
            lattice = self.lattice
            rn = np.random.random
            ne = [(0, -1), (0, 1), (-1, 0), (1, 0)]
            nnsite = set([(self.center+nx, self.center+ny) for nx, ny in ne])
            percolate = False
            l = set([])
            while len(nnsite) != 0 and percolate == False:
                nextseed = []
                for nn in nnsite:
                    if rn() < p:
                        lattice[nn] = 1
                        nextseed.append(nn)
                    else: lattice[nn] = -1
                nnsite = set([])
                for i, j in nextseed:
                    nnsite = nnsite | set([(i+nx, j+ny) for nx, ny in ne
                                    if lattice[i+nx, j+ny] == 0])
                    if i == 1:      l = l | set(['top'])
                    if i == self.L: l = l | set(['bottom'])
                    if j == 1:      l = l | set(['left'])
                    if j == self.L: l = l | set(['right'])

                if ('top' in l and 'bottom' in l) or \
```

```
                    ('right' in l and 'left' in l):
                     percolate = True

            if len(nnsite) == 0 and self.count < 30:
                self.count += 1
                self.perc_cluster(self.p)
            else:
                self.count = 0
                self.lattice = lattice[1:-1, 1:-1]
                return self.lattice

    def draw_canvas(self, rect, L):
        default_size = 640 # default size of canvas
        r = int(default_size/(2*L))
        fig_size = 2*r*L
        margin = 10
        self.sub = Toplevel()

        self.sub.title('invasion percolation')
        self.canvas = Canvas(self.sub, width=fig_size+2*margin,
                    height=fig_size+2*margin)
        self.canvas.create_rectangle(margin, margin,
                    fig_size+margin, fig_size+margin,
                    outline='black', fill='white')
        self.canvas.pack()

        c = self.canvas.create_rectangle

        site = np.where(rect==1)
        for m, n in zip(site[0], site[1]):
            c(2*m*r+margin, 2*n*r+margin,
                        2*(m+1)*r+margin, 2*(n+1)*r+margin,
                        outline='black', fill='black')
        self.r = r
        self.margin = margin

class Ant():

    def rw_d2(self, center, lattice, r, margin, f, N=1000, view=True):

```

```
        R_2 = []
        x, y = center, center
        per_lattice = lattice == 1
        rn = np.random.rand
        if view:
            oval = f.create_oval
            delete = f.delete
            ant = oval(2*x*r+margin, 2*x*r+margin,
                            2*(x+1)*r+margin, 2*(x+1)*r+margin,
                            outline='white', fill='red')
        for n in xrange(N):
            p = rn()*4
            if p < 1:   d = (0, 1)
            elif p < 2: d = (0, -1)
            elif p < 3: d = (1, 0)
            else:       d = (-1, 0)

            _x, _y = x+d[0], y+d[1]

            # 進行方向が占有サイトのときのみ進める
            try:
                if per_lattice[_x][_y]:
                    x, y = _x, _y
                    if view:
                        delete(ant)
                        ant = oval(2*x*r+margin, 2*y*r+margin,
                                2*(x+1)*r+margin, 2*(y+1)*r+margin,
                                outline='white', fill='red')
                        f.update()
                        time.sleep(0.02)
            except IndexError:
                _x, _y = x, y
            R_2.append((x-center)**2 + (y-center)**2)
        t = xrange(1, N+1)
        return t, R_2

class TopWindow:

    def quit(self):
        self.root.destroy()
```

```
        sys.exit()

    def show_window(self, title="title", *args):
        self.root = Tk()
        self.root.title(title)
        frames = []
        for i, arg in enumerate(args):
            frames.append(Frame(self.root, padx=5, pady=5))
            for k, v in arg:
                Button(frames[i],text=k,command=v).pack(expand=YES, fill='x')
            frames[i].pack(fill='x')
        f = Frame(self.root, padx=5, pady=5)
        Button(f,text='quit',command=self.quit).pack(expand=YES, fill='x')
        f.pack(fill='x')
        self.root.mainloop()

class Common():

    def plot_graph(self, x_data, y_data, x_labels, y_labels,
                    xscale, yscale, aspect):
        """ Plot the graph about y_data for each x_data.
        """
        d = len(y_data)
        if not len(x_data) == len(y_data) == len(x_labels) == len(y_labels)\
               == len(xscale) == len(yscale) == len(aspect):
            raise ValueError("Arguments must have the same dimension.")
        if d == 0:
            raise ValueError("At least one data for plot.")
        if d > 9:
            raise ValueError("""So much data for plot in one figure.
                                Please divide two or more data sets.""")

        fig = plt.figure(figsize=(9, 8))
        subplot_positioning = ['11','21','22','22','32','32','33','33','33']
        axes = []
        for n in range(d):
            lmn = int(subplot_positioning[d-1] + str(n+1))
            axes.append(fig.add_subplot(lmn))

        for i, ax in enumerate(axes):
```

```
            ymin, ymax = min(y_data[i]), max(y_data[i])
            ax.set_aspect(aspect[i])
            ax.set_xscale(xscale[i])
            ax.set_yscale(yscale[i])
            ax.set_xlabel(x_labels[i], fontsize=16)
            ax.set_ylabel(y_labels[i], fontsize=16)
            ax.set_ymargin(0.05)
            ax.plot(x_data[i], y_data[i], 'o-')

        fig.subplots_adjust(wspace=0.2, hspace=0.5)
        fig.tight_layout()
        plt.show()

    def fitting(self, x, y, parameter0, fit_func):

        parameter = parameter0
        result = optimize.leastsq(fit_func, parameter0, args=(x, y))
        for i in range(len(parameter)):
            parameter[i] = result[0][i]

        return parameter

class Main():

    def __init__(self):
        self.L = 61
        self.p = 0.7
        self.top = TopWindow()
        self.per = Percolation(self.L)
        self.ant = Ant()
        self.common = Common()
        self.count = 1
        run = (('percolation cluster', self.pushed),)
        run2 = (('ant_walk', self.ant_walk),
                ('calculate R_2', self.calculate_R_2),
                (r'caluculate D_s(p)', self.fit))
        save = (('save canvas to sample.eps', self.pr),)
        self.top.show_window("Ant Walk", run, run2, save)

    def pr(self):
```

```
            d = self.per.canvas.postscript(file="figure_%d.eps" % self.count)
            print "saved the figure to a eps file"
            self.count += 1

        def pushed(self):
            self.per.perc_cluster(self.p)
            self.per.draw_canvas(self.per.lattice, self.L)

        def ant_walk(self):
            if self.per.sub == None or not self.per.sub.winfo_exists():
                self.per.perc_cluster(self.p)
                self.per.draw_canvas(self.per.lattice, self.L)
            t, R_2 = self.ant.rw_d2(self.per.center, self.per.lattice,
                            self.per.r, self.per.margin, self.per.canvas)

        def calculate_R_2(self):
            trial = 1000
            N = 5000
            R_2 = []

            for i in range(trial):
                self.per.perc_cluster(self.p)
                t, R_2_ = self.ant.rw_d2(self.per.center, self.per.lattice,
                                0, 0, 0, N, view=False)
                R_2.append(R_2_)
            R_2 = np.array(R_2).reshape(trial, N)
            self.ave_R_2 = np.sum(R_2, axis=0)/float(trial)
            self.t = t
            self.common.plot_graph([self.t], [self.ave_R_2], [r'$t$'],
                            [r'$\langle R^{2}(t) \rangle$'],
                            ['linear'], ['linear'], ['auto'])

        def fit(self, view=True):

            def fit_func(parameter0, t, R_2):
                a = parameter0[0]
                b = parameter0[1]
                residual = R_2 - (a*t + b)
                return residual

```

```
        def fitted(t, a, b):
            return a*t + b

        cut_from = int(raw_input("from ? (t) >>> "))
        cut_to = int(raw_input("to ? (t) >>> "))
        parameter0 = [0.5, 1.]
        cut_t =  np.array(list(self.t)[cut_from:cut_to])
        result = self.common.fitting(cut_t,
                        np.array(self.ave_R_2[cut_from:cut_to]),
                        parameter0, fit_func)
        a = result[0]
        D = a/4.
        b = result[1]
        print D
        if view:
            fig = plt.figure('Diffusion coefficient')
            ax = fig.add_subplot(111)
            ax.plot(self.t, self.ave_R_2, '-o',
                            label=r"$\langle R^{2}(t) \rangle$")
            ax.plot(cut_t, fitted(cut_t, a, b), lw=2,
                            label=r"fit func: $D_{s}(p) = %f$" % D)
            ax.set_xlabel(r'$t$', fontsize=16)
            ax.set_ylabel(r'$\langle R^{2}(t) \rangle$', fontsize=16)
            ax.set_xscale('linear')
            ax.set_yscale('linear')
            ax.set_ymargin(0.05)
            fig.tight_layout()
            plt.legend(loc='best')
            plt.show()
        return D

if __name__ == '__main__':

    Main()
```

## 3 実習課題

a. $p = 1$ では蟻は完全格子の上を歩きまわるので，$\langle R^2(t)\rangle \propto t$ である．蟻が $p > p_c$ の 2 次元のパーコレーション・クラスター上でランダムウォークする場合を考えよう．$p > p_c$ に対して，$\langle R^2(t)\rangle \sim 4D_s(p)t$ と仮定する．ここで，端から端まで連結した (spanning) クラスター上のみを考えて，$p > p_c$ のときに存在する有限の大きさのクラスター上のランダムウォークは考えないことを強調するために，拡散係数を $D_s$ と表した．問題 14.3 で考えた成長のアルゴリズムを用いて $p = 0.7$ でのパーコレーション・クラスターを生成せよ．種の格子点として蟻の初期位置を選んで，スクリーン上で蟻の動きを観察するようにプログラムを修正せよ．蟻はどこで多くの時間を費やすか．蟻が拡散する場合，比 $D_s(p)/D(p = 1)$ は定性的にどう表せるか．

作成したプログラムを用いて，$p = 0.7$ としたときのパーコレーション・クラスター上を動く蟻を再現した (図 1)．蟻が多くの時間を費やすのは，蟻の自由度の低い箇所，すなわち三方を非占有サイトで囲まれた領域や角になった領域，細い領域であるようだ．これは蟻の行動方法からもわかることで，動ける確率が小さくなるほど蟻はその場にとどまっていることになる．したがって，蟻が拡散する場合，拡散係数の比 $D_s(p)/D(p = 1)$ が，$p$ の増加関数であることがわかる．

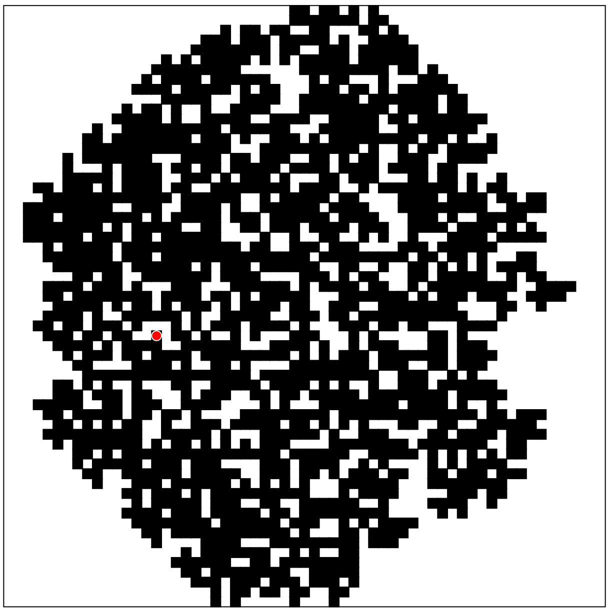

図 1　$p = 0.7$ としたときのパーコレーション・クラスター上をランダムウォークする蟻の例

b. $p = 0.4$ のときの $\langle R^2(t)\rangle$ を計算し，$p < p_c$ に対してはクラスターは有限であり，$\langle R^2(t)\rangle$ には上限があって，拡散は不可能であることを確認せよ．

$p = 0.4$ のときの $\langle R^2(t)\rangle$ を計算し，横軸 $t$，縦軸 $\langle R^2(t)\rangle$ として両対数グラフにプロットしたものを図 2 に示す．グラフからも明らかなように，$p < p_c$ ではパーコレーション・クラスターは生成されないために $\langle R^2(t)\rangle$ には系のサイズより小さい上限 (この場合 7 程度) があって，拡散は不可能であることがわかる．

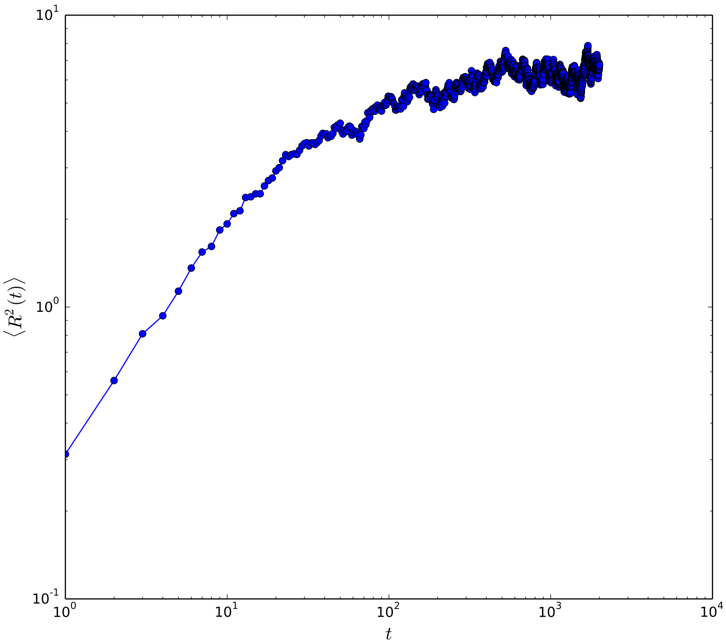


図 2　$p = 0.4$ のときの $\langle R^2(t) \rangle$.

c. 設問 a と同様に，$L = 61$ として $p = 1.0, 0.8, 0.7, 0.65, 0.62$ についての平均 2 乗変位を計算せよ．時間が許せば，いくつかのクラスターについての平均をとれ．$\langle R^2(t) \rangle$ を $t$ に対して両対数でプロットせよ．比較的短時間では，$\langle R^2(t) \rangle$ の定性的な $t$ 依存性はどのようか．長時間について $\langle R^2(t) \rangle$ が $t$ に比例するかどうか決定せよ (格子の大きさは有限なので，$\langle R^2(t) \rangle$ の最大値には上限があることを忘れないこと．)．$\langle R^2(t) \rangle \sim t$ の場合の $D_s(p)$ を求めよ．比 $D_s(p)/D(p = 1)$ を $p$ の関数としてプロットし，定性的な振る舞いについて議論せよ．

$L = 61$ として $p = 1.0, 0.8, 0.7, 0.65, 0.62$ についての平均 2 乗変位 $\langle R^2(t) \rangle$ を計算し，両対数グラフにした (図 3)．このとき，それぞれは 300 回の試行の平均となっている．

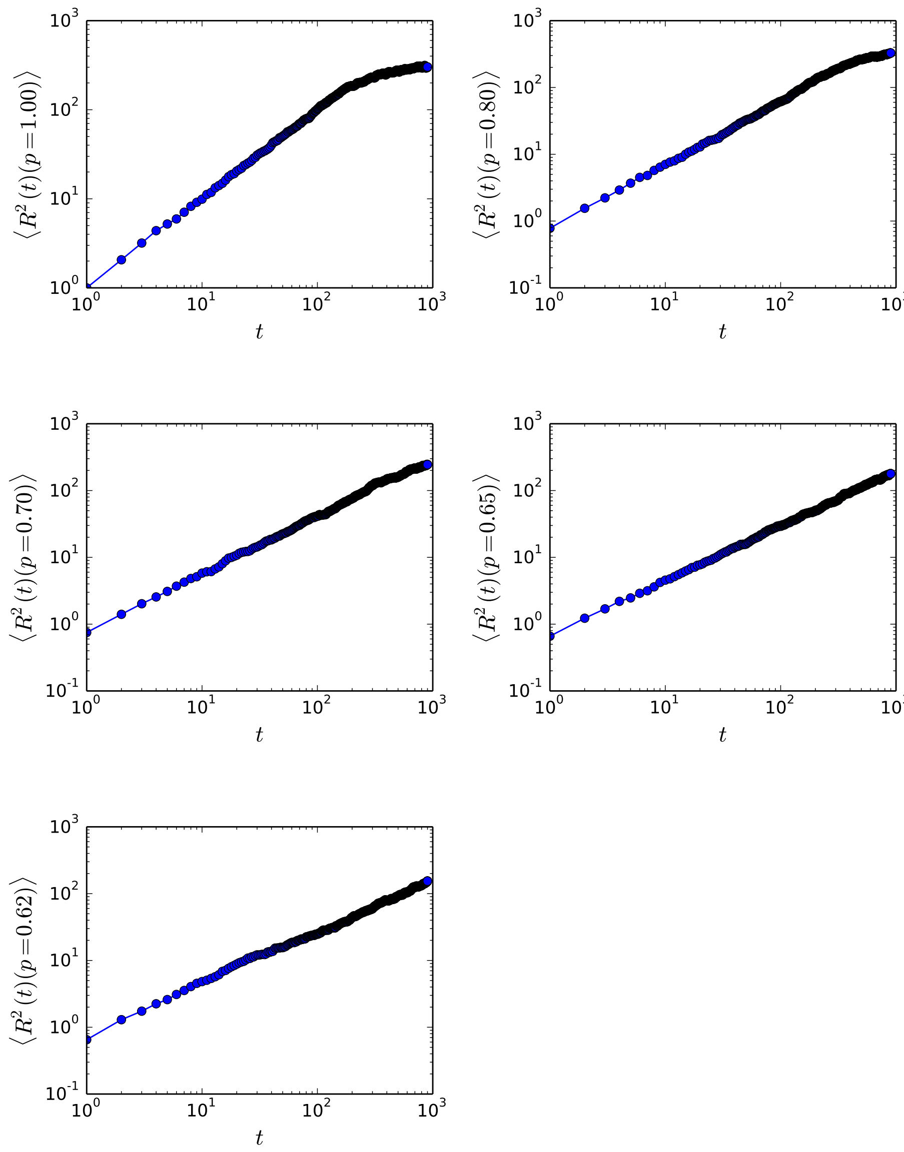


図 3　$p = 1.0, 0.8, 0.7, 0.65, 0.62$ についての平均 2 乗変位 $\langle R^2(t)\rangle$.

図 3 から，比較的短時間では $\langle R^2(t)\rangle$ は $t$ のべき乗で増加することがわかる．比較的長時間では，図 4 に示したように，$\langle R^2(t)\rangle$ は $t$ に比例するように見える．$\langle R^2(t)\rangle \sim 4D_s(p)t$ と仮定して，$D_s(p)$ を求めると，$D_s(p = 0.8) \simeq 0.0095$ となった．また，同じように $p > p_c$ の $p$ に関して $D_s(p)$ を求め，比 $D_s(p)/D(p = 1)$ を $p$ の関数としてプロットしたものを図 5 に示す．設問 d で扱われるように，$p < p_c$ では拡散はないので $D_s$ は $p \to p_c$ とともに 0 になると考えられ，図 5 で得られたグラフの形からも $D_s(p) = (p - p_c)^{\mu_s}$ であることが予想される．また，グラフの凸性から $\mu > 1$ であることもわかる．

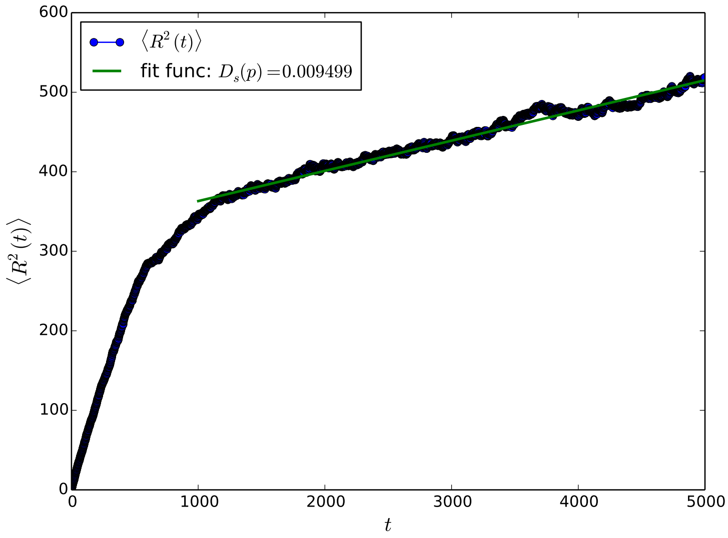


図 4　$p = 0.8$ で 1000 回の試行の平均により得られた $\langle R^2(t)\rangle$ と $t$ の関係.

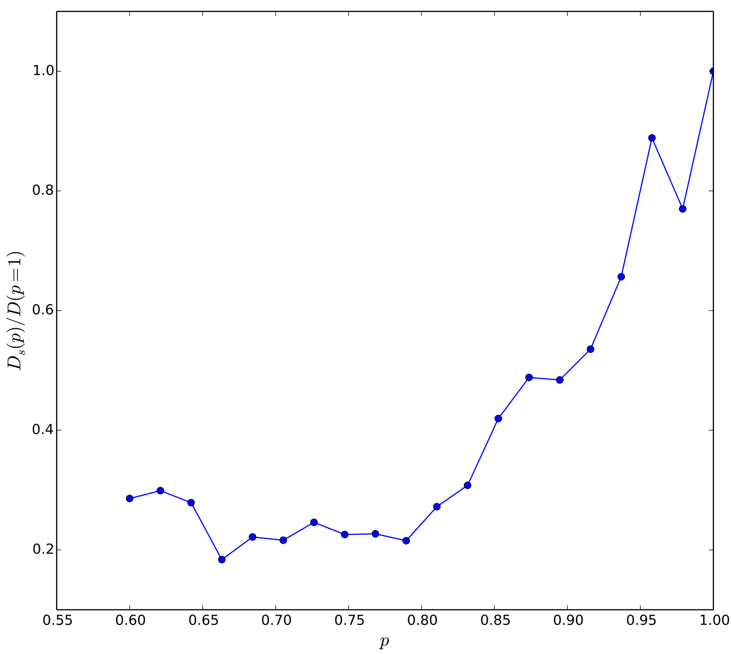


図 5　横軸 $p$，縦軸を比 $D_s(p)/D(p = 1)$ としてプロットしたグラフ.

# 4　まとめ

フラクタル図形の中を拡散する粒子の運動について，その平均 2 乗変位 $\langle R^2(t)\rangle$ と $p$，$t$ の間に成り立つ関係について，いくつかを学ぶことができた.