

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 藤本将太郎

2014 年 8 月 5 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 (正方向格子上で) 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步に対応するならば、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・クラスターが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。

```
1  #! /usr/bin/env python
2  # -*- coding:utf-8 -*-
3  #
4  # written by Shotaro Fujimoto, August 2014.
5
6  from Tkinter import *
7  import numpy as np
8  import scipy.optimize as optimize
9  import matplotlib.pyplot as plt
10 import sys
11 import time
12
13 class Percolation:
```

```

14
15     def __init__(self, L=61):
16         self.sub = None
17         self.L = L
18         self.count = 0
19
20     def perc_cluster(self, p=0.7):
21         if p > 1 or p < 0:
22             raise ValueError("site occupation probability must be 0 <= p <= 1")
23         self.p = p
24         self.lattice = np.zeros([self.L+2, self.L+2], dtype=int)
25         self.lattice[:, 1, :] = self.lattice[:, :1] = -1
26         self.lattice[self.L+1:, :] = self.lattice[:, self.L+1:] = -1
27         self.center = int(self.L/2) + 1
28         self.lattice[self.center, self.center] = 1
29         nextseed = [(self.center, self.center)]
30         if self.sub is None or not self.sub.winfo_exists():
31             lattice = self.lattice
32             rn = np.random.random
33             ne = [(0, -1), (0, 1), (-1, 0), (1, 0)]
34             nnsite = set([(self.center+nx, self.center+ny) for nx, ny in ne])
35             percolate = False
36             l = set([])
37             while len(nnsite) != 0 and percolate == False:
38                 nextseed = []
39                 for nn in nnsite:
40                     if rn() < p:
41                         lattice[nn] = 1
42                         nextseed.append(nn)
43                     else: lattice[nn] = -1
44             nnsite = set([])
45             for i, j in nextseed:
46                 nnsite = nnsite | set([(i+nx, j+ny) for nx, ny in ne
47                                         if lattice[i+nx, j+ny] == 0])
48                 if i == 1:      l = l | set(['top'])
49                 if i == self.L: l = l | set(['bottom'])
50                 if j == 1:      l = l | set(['left'])
51                 if j == self.L: l = l | set(['right'])
52
53             if ('top' in l and 'bottom' in l) or \

```

```

54         ('right' in l and 'left' in l):
55             percolate = True
56
57         if len(nnsite) == 0 and self.count < 30:
58             self.count += 1
59             self.perc_cluster(self.p)
60         else:
61             self.count = 0
62             self.lattice = lattice[1:-1, 1:-1]
63             return self.lattice
64
65     def draw_canvas(self, rect, L):
66         default_size = 640 # default size of canvas
67         r = int(default_size/(2*L))
68         fig_size = 2*r*L
69         margin = 10
70         self.sub = Toplevel()
71
72         self.sub.title('invasion percolation')
73         self.canvas = Canvas(self.sub, width=fig_size+2*margin,
74                               height=fig_size+2*margin)
75         self.canvas.create_rectangle(margin, margin,
76                                     fig_size+margin, fig_size+margin,
77                                     outline='black', fill='white')
78         self.canvas.pack()
79
80         c = self.canvas.create_rectangle
81
82         site = np.where(rect==1)
83         for m, n in zip(site[0], site[1]):
84             c(2*m*r+margin, 2*n*r+margin,
85               2*(m+1)*r+margin, 2*(n+1)*r+margin,
86               outline='black', fill='black')
87
88         self.r = r
89         self.margin = margin
90
91     class Ant():
92
93         def rw_d2(self, center, lattice, r, margin, f, N=1000, view=True):

```

```

94     R_2 = []
95     x, y = center, center
96     per_lattice = lattice == 1
97     rn = np.random.rand
98     if view:
99         oval = f.create_oval
100        delete = f.delete
101        ant = oval(2*x*r+margin, 2*x*r+margin,
102                  2*(x+1)*r+margin, 2*(x+1)*r+margin,
103                  outline='white', fill='red')
104    for n in xrange(N):
105        p = rn()*4
106        if p < 1: d = (0, 1)
107        elif p < 2: d = (0, -1)
108        elif p < 3: d = (1, 0)
109        else: d = (-1, 0)
110
111        _x, _y = x+d[0], y+d[1]
112
113        # 進行方向が占有サイトのときのみ進める
114        try:
115            if per_lattice[_x][_y]:
116                x, y = _x, _y
117                if view:
118                    delete(ant)
119                    ant = oval(2*x*r+margin, 2*y*r+margin,
120                              2*(x+1)*r+margin, 2*(y+1)*r+margin,
121                              outline='white', fill='red')
122                    f.update()
123                    time.sleep(0.02)
124            except IndexError:
125                _x, _y = x, y
126            R_2.append((x-center)**2 + (y-center)**2)
127    t = xrange(1, N+1)
128    return t, R_2
129
130 class TopWindow:
131
132     def quit(self):
133         self.root.destroy()

```

```

134         sys.exit()
135
136     def show_window(self, title="title", *args):
137         self.root = Tk()
138         self.root.title(title)
139         frames = []
140         for i, arg in enumerate(args):
141             frames.append(Frame(self.root, padx=5, pady=5))
142             for k, v in arg:
143                 Button(frames[i], text=k, command=v).pack(expand=YES, fill='x')
144             frames[i].pack(fill='x')
145         f = Frame(self.root, padx=5, pady=5)
146         Button(f, text='quit', command=self.quit).pack(expand=YES, fill='x')
147         f.pack(fill='x')
148         self.root.mainloop()
149
150     class Common():
151
152     def plot_graph(self, x_data, y_data, x_labels, y_labels,
153                   xscale, yscale, aspect):
154         """ Plot the graph about y_data for each x_data.
155         """
156         d = len(y_data)
157         if not len(x_data) == len(y_data) == len(x_labels) == len(y_labels)\
158             == len(xscale) == len(yscale) == len(aspect):
159             raise ValueError("Arguments must have the same dimension.")
160         if d == 0:
161             raise ValueError("At least one data for plot.")
162         if d > 9:
163             raise ValueError("""So much data for plot in one figure.
164                               Please divide two or more data sets.""")
165
166         fig = plt.figure(figsize=(9, 8))
167         subplot_positioning = ['11', '21', '22', '22', '32', '32', '33', '33', '33']
168         axes = []
169         for n in range(d):
170             lmn = int(subplot_positioning[d-1] + str(n+1))
171             axes.append(fig.add_subplot(lmn))
172
173         for i, ax in enumerate(axes):

```

```

174         ymin, ymax = min(y_data[i]), max(y_data[i])
175         ax.set_aspect(aspect[i])
176         ax.set_xscale(xscale[i])
177         ax.set_yscale(yscale[i])
178         ax.set_xlabel(x_labels[i], fontsize=16)
179         ax.set_ylabel(y_labels[i], fontsize=16)
180         ax.set_ymargin(0.05)
181         ax.plot(x_data[i], y_data[i], 'o-')
182
183     fig.subplots_adjust(wspace=0.2, hspace=0.5)
184     fig.tight_layout()
185     plt.show()
186
187     def fitting(self, x, y, parameter0, fit_func):
188
189         parameter = parameter0
190         result = optimize.leastsq(fit_func, parameter0, args=(x, y))
191         for i in range(len(parameter)):
192             parameter[i] = result[0][i]
193
194         return parameter
195
196     class Main():
197
198         def __init__(self):
199             self.L = 61
200             self.p = 0.7
201             self.top = TopWindow()
202             self.per = Percolation(self.L)
203             self.ant = Ant()
204             self.common = Common()
205             self.count = 1
206             run = (('percolation cluster', self.pushed),)
207             run2 = (('ant_walk', self.ant_walk),
208                    ('calculate R_2', self.calculate_R_2),
209                    ('caluculate D_s(p)', self.fit))
210             save = (('save canvas to sample.eps', self.pr),)
211             self.top.show_window("Ant Walk", run, run2, save)
212
213         def pr(self):

```

```

214         d = self.per.canvas.postscript(file="figure_%d.eps" % self.count)
215         print "saved the figure to a eps file"
216         self.count += 1
217
218     def pushed(self):
219         self.per.perc_cluster(self.p)
220         self.per.draw_canvas(self.per.lattice, self.L)
221
222     def ant_walk(self):
223         if self.per.sub == None or not self.per.sub.wininfo_exists():
224             self.per.perc_cluster(self.p)
225             self.per.draw_canvas(self.per.lattice, self.L)
226         t, R_2 = self.ant.rw_d2(self.per.center, self.per.lattice,
227                                self.per.r, self.per.margin, self.per.canvas)
228
229     def calculate_R_2(self):
230         trial = 1000
231         N = 5000
232         R_2 = []
233
234         for i in range(trial):
235             self.per.perc_cluster(self.p)
236             t, R_2_ = self.ant.rw_d2(self.per.center, self.per.lattice,
237                                     0, 0, 0, N, view=False)
238             R_2.append(R_2_)
239         R_2 = np.array(R_2).reshape(trial, N)
240         self.ave_R_2 = np.sum(R_2, axis=0)/float(trial)
241         self.t = t
242         self.common.plot_graph([self.t], [self.ave_R_2], [r'$t$'],
243                                [r'$\langle R^2(t) \rangle$'],
244                                ['linear'], ['linear'], ['auto'])
245
246     def fit(self, view=True):
247
248         def fit_func(parameter0, t, R_2):
249             a = parameter0[0]
250             b = parameter0[1]
251             residual = R_2 - (a*t + b)
252             return residual
253

```



```

254     def fitted(t, a, b):
255         return a*t + b
256
257     cut_from = int(raw_input("from ? (t) >>> "))
258     cut_to = int(raw_input("to ? (t) >>> "))
259     parameter0 = [0.5, 1.]
260     cut_t = np.array(list(self.t)[cut_from:cut_to])
261     result = self.common.fitting(cut_t,
262                                 np.array(self.ave_R_2[cut_from:cut_to]),
263                                 parameter0, fit_func)
264     a = result[0]
265     D = a/4.
266     b = result[1]
267     print D
268     if view:
269         fig = plt.figure('Diffusion coefficient')
270         ax = fig.add_subplot(111)
271         ax.plot(self.t, self.ave_R_2, '-o',
272                 label=r"$\langle R^2(t) \rangle$")
273         ax.plot(cut_t, fitted(cut_t, a, b), lw=2,
274                 label=r"fit func: $D_{s}(p) = %f$ % D")
275         ax.set_xlabel(r'$t$', fontsize=16)
276         ax.set_ylabel(r'$\langle R^2(t) \rangle$', fontsize=16)
277         ax.set_xscale('linear')
278         ax.set_yscale('linear')
279         ax.set_ylim(0.05)
280         fig.tight_layout()
281         plt.legend(loc='best')
282         plt.show()
283     return D
284
285 if __name__ == '__main__':
286
287     Main()
288

```

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 次元のパークレーション・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 $p = 0.7$ でのパークレーション・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、 $p = 0.7$ としたときのパークレーション・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。

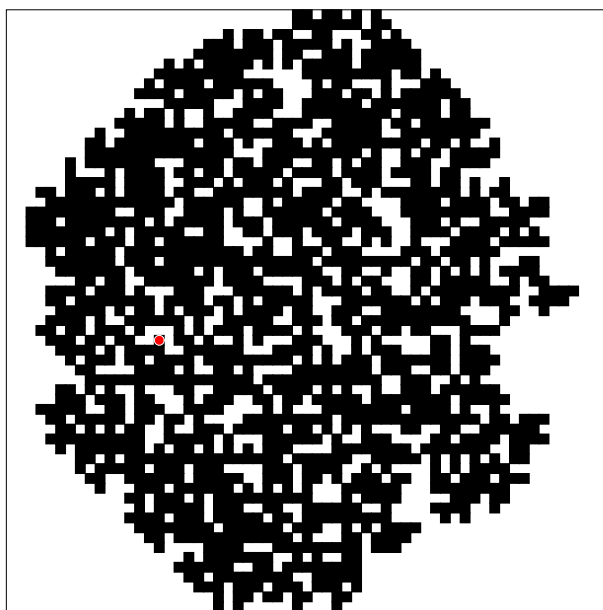


図 1 $p = 0.7$ としたときのパークレーション・クラスター上をランダムウォークする蟻の例

- 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のように、 $p < p_c$ ではパークレーション・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。

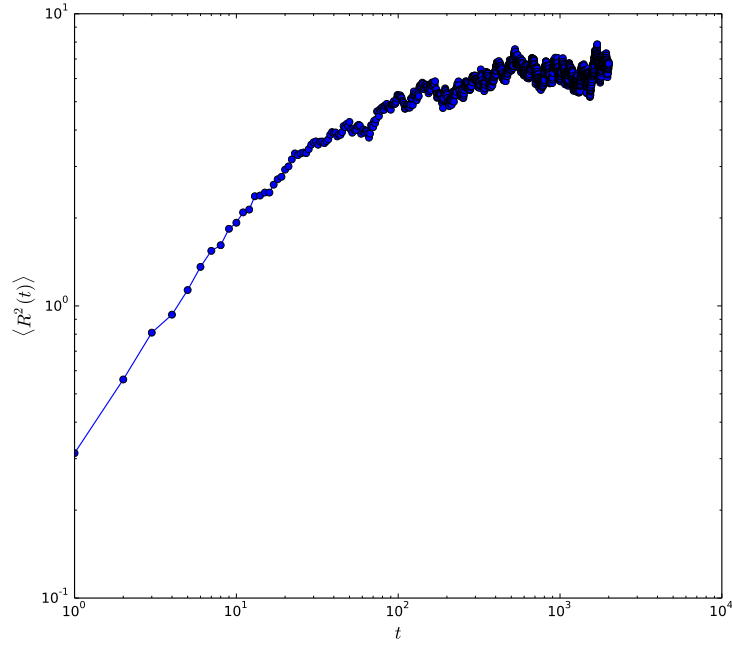


図2 $p = 0.4$ のときの $\langle R^2(t) \rangle$.

- 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 依存性はどのようなか. 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.

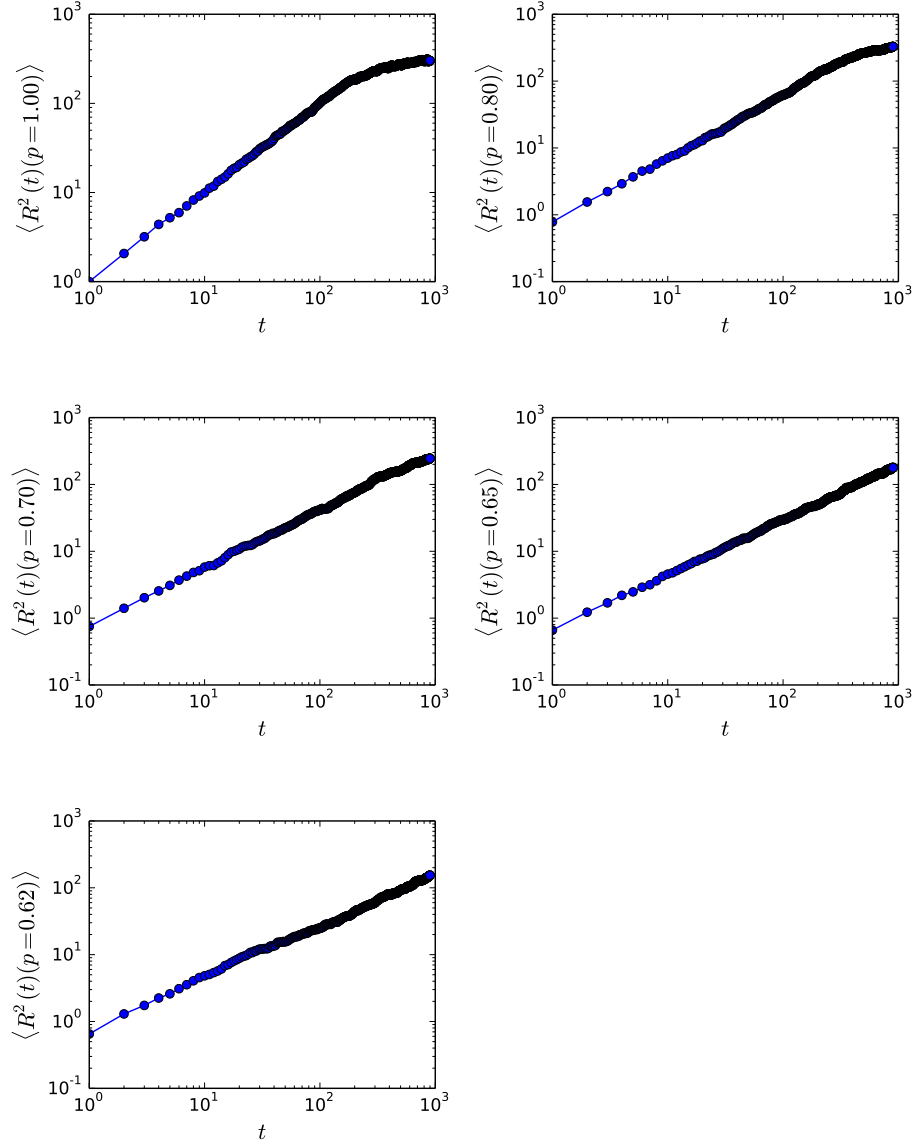


図3 $p = 1.0, 0.8, 0.7, 0.65, 0.62$ についての平均 2 乗変位 $\langle R^2(t) \rangle$.

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 $p \rightarrow p_c$ 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。

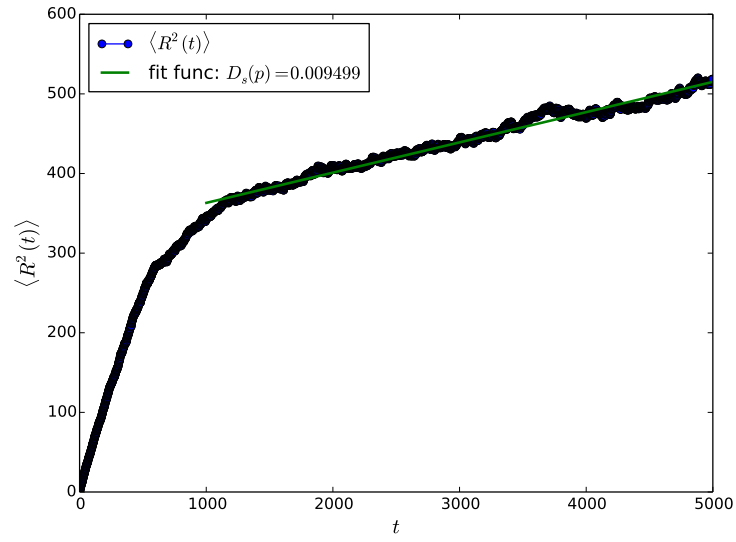


図4 $p = 0.8$ で 1000 回の試行の平均により得られた $\langle R^2(t) \rangle$ と t の関係.

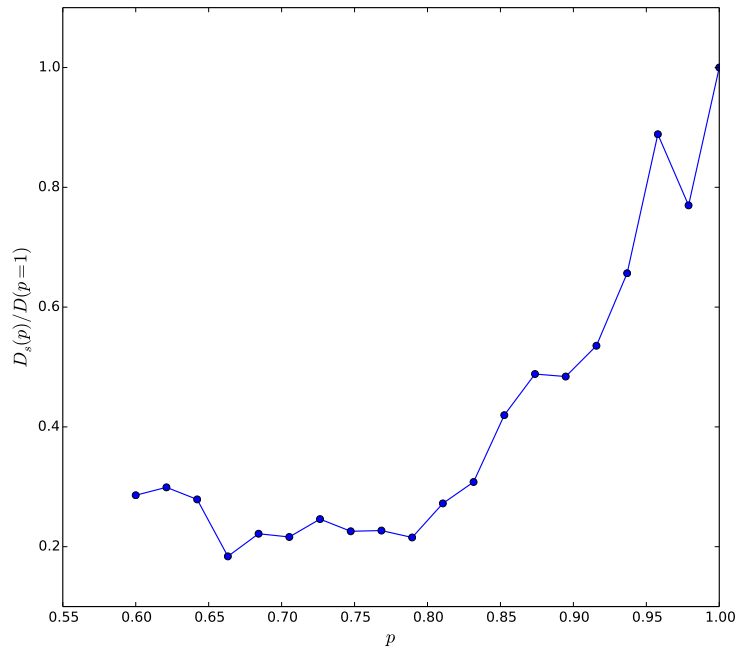


図5 横軸 p , 縦軸を比 $D_s(p)/D(p=1)$ としてプロットしたグラフ.

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.

参考文献

- [1] ハーベイ・ゴールド, ジャン・トポニック. 石川正勝・宮島佐介訳. 『計算機物理学入門』. ピアソン・エデュケーション, 2000.