計算機実習 問題 14.5 簡単な伝染病モデル

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

自然界には，さまざまな場面でフラクタルな構造を持つものと出会うことがある．その一つの身近な例として，伝染病の簡単なモデルにおいて現れる性質について考えることにする．伝染病の広がりについて考えるとき，流行の条件を知りたいと思うのが普通である．伝染病の広がりの簡単な格子モデルをつぎのように定式化することができる．占有された格子点は病気に感染した人に対応する．初め，1 つの感染した格子点があり，最隣接の 4 個の周辺の点 (正方格子上で) に感染する可能性がある．つぎの分割時間で，それら 4 個の感染可能な格子点のそれぞれは確率 p で占有される (感染する)．もし感染可能な格子点が占有されなければ，その格子点は免疫があるとし，再び調べることはしない．つぎに，新しい感染可能な格子点を見出し，その病気の伝染が抑制されるか格子の境界に達するかまで続ける．この病気の広がりに関する成長モデルは，確率 p のパーコレーション・クラスターと等価な，感染した格子点のクラスター形成することを確認せよ．唯一の違いはモデルに離散的な分割時間を導入したことである．問題 14.5 では，このモデルの性質のいくつかについて調べることにする．

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　簡単な伝染病モデルを再現するプログラム

パーコレーション・クラスター 1 つを効率的に生成する 1 つの方法としてよく知られたアルゴリズムに，リースのアルゴリズムがある．これは次の手順を実行することと同等である．

1. 種として格子点の 1 つが占有される．種の最隣接格子点 (正方格子では 4) を周辺の点と呼ぶ．
2. 各周辺の点について，単位区間の乱数 r を生成する．$r \leq p$ の場合にはその格子点は占有され，クラスターに加えられる．そうでない場合には占有されない．格子点が占有されない確率を $1-p$ とするために，占有されなかった格子点について再び占有されるかどうか試すことはしない．
3. 占有された各格子点について，新しい周囲の点，つまり試されていない隣接格子点が存在するかどうか調べる．周囲の点の表に新しい周辺の点を加える．
4. 占有されるかどうかが試されていない周辺の点がなくなるまで，手順 2 と 3 を続ける．

このリースのアルゴリズムの一部を変更し，伝染病に関する簡単なシミュレーションを行う．確率 p によって占有された (感染した) 格子点は値 1 を持つとし，調べられたが占有されなかった格子点 (免疫をもつ格子点) には値-1 を代入していく．種を格子の中心の点とし，考えている格子の外側の領域はすべて-1 の値を持たせておく．次にこのステップで占有された格子点に関して，その周囲でまだ試されていない格子点 (値 0 を持つ) を探し，これを配列 nnsite に格納する．また，その格子点自身が系の端に位置するとき，集合 l にその情報を記録しておき，端と端が連結したとき，この系はパーコレートしたとする．この過程を繰り返し，nnsite に要素が含まれなくなったとき．または系がパーコレートしたときに過程を終了する (41〜67 行目)．関数 perc_cluster がこれらのアルゴリズムを実行する部分であり，各時刻 t に関して，その時刻での周囲の点の数 S と，占有された格子点の数 N を記録している．関数を実行し終わった後には，これらの時系列でのデータが返され，関数内部で格子の占有状態が記録されている．以前に使用した draw_canvas や TopWindow クラス，plot_graph 関数を用いて，占有された格子点を描画したり，S，N についてのグラフを表示したりすることができるようにしてある．

```
1   #! /usr/bin/env python
2   # -*- coding:utf-8 -*-
3   #
4   # written by Shotaro Fujimoto, June 2014.
5
6   from Tkinter import *
7   import numpy as np
8   import sys
9   import matplotlib.pyplot as plt
10
11
12  class Percolation:
13
14      def __init__(self, L=61, p=0.5927):
15          if L % 2 == 0:
16              raise ValueError("lattice size L must be odd number.")
17          self.sub = None
18          self.L = L # lattice size
19          self.p = p
20
21      def perc_cluster(self, p):
22          if p > 1 or p < 0:
23              raise ValueError("site occupation probability must be 0 <= p <= 1")
24          self.p = p
25          self.lattice = np.zeros([self.L+2, self.L+2], dtype=int)
26          self.lattice[:1,:] = self.lattice[:, :1] = -1
27          self.lattice[self.L+1:,:] = self.lattice[:, self.L+1:] = -1
```

```
28          center = (L/2) + 1
29          self.lattice[center, center] = 1
30          nextseed = [(center, center)]
31          if self.sub is None or not self.sub.winfo_exists():
32              lattice = self.lattice
33              rn = np.random.random
34              ne = [(0, -1), (0, 1), (-1, 0), (1, 0)]
35              nnsite = set([(center+nx, center+ny) for nx, ny in ne])
36              t = [0] # time
37              S = [4] # a number of sites can be infected
38              N = [1] # a number of infected sites
39              percolate = False
40              l = set([])
41              while len(nnsite) != 0 and percolate == False:
42                  nextseed = []
43                  for nn in nnsite:
44                      if rn() < p:
45                          lattice[nn] = 1
46                          nextseed.append(nn)
47                      else:
48                          lattice[nn] = -1
49                  nnsite = set([])
50                  for i, j in nextseed:
51                      nnsite = nnsite | set([(i+nx, j+ny) for nx, ny in ne
52                                             if lattice[i+nx, j+ny] == 0])
53                      if i == 1:
54                          l = l | set(['top'])
55                      if i == self.L:
56                          l = l | set(['bottom'])
57                      if j == 1:
58                          l = l | set(['left'])
59                      if j == self.L:
60                          l = l | set(['right'])
61
62                  if ('top' in l and 'bottom' in l) or \
63                     ('right' in l and 'left' in l):
64                      percolate = True
65
66                  t.append(t[-1]+1)
67                  S.append(len(nnsite))
```

```
68                  N.append(np.sum(lattice == 1))
69              self.lattice = lattice[1:-1, 1:-1]
70
71          return t, S, N
72
73      def draw_canvas(self, rect, L):
74          default_size = 640 # default size of canvas
75          r = int(default_size/(2*L))
76          fig_size = 2*r*L
77          margin = 10
78          sub = Toplevel()
79
80          sub.title('figure  '+'(p=%s)' % str(self.p))
81          self.canvas = Canvas(sub, width=fig_size+2*margin,
82                               height=fig_size+2*margin)
83          self.canvas.create_rectangle(margin, margin,
84                                       fig_size+margin, fig_size+margin,
85                                       outline='black', fill='white')
86          self.canvas.pack()
87
88          c = self.canvas.create_rectangle
89
90          site = np.where(rect == 1)
91          for m, n in zip(site[0], site[1]):
92              c(2*m*r+margin, 2*n*r+margin,
93                2*(m+1)*r+margin, 2*(n+1)*r+margin,
94                outline='', fill='black')
95
96
97  class TopWindow:
98
99      def quit(self):
100         self.root.destroy()
101         sys.exit()
102
103     def show_window(self, pr, pushed, b4_pushed, auto):
104         self.root = Tk()
105         self.root.title('Percolation')
106         f = Frame(self.root)
107         self.label = Label(f, text='p =')
```

```
108           self.label.pack(side='left')
109           self.entry = Entry(f, width=20)
110           self.entry.pack(side='left')
111           self.entry.delete(0, END)
112           self.entry.insert(0, 0.5927)
113           self.entry.focus_set()
114
115           b5 = Button(f, text='auto', command=auto)
116           b5.pack(side='left', expand=YES, fill='x')
117
118           b1 = Button(f, text='run', command=pushed)
119           b1.pack(side='left', expand=YES, fill='x')
120
121           b4 = Button(f, text='plot graph', command=b4_pushed)
122           b4.pack(side='left', expand=YES, fill='x')
123
124           b2 = Button(f, text='write canvas to sample.eps', command=pr)
125           b2.pack(side='left', expand=YES, fill='x')
126
127           b3 = Button(f, text='quit', command=self.quit)
128           b3.pack(side='right', expand=YES, fill='x')
129
130           f.pack(fill='x')
131
132           self.root.mainloop()
133
134   def plot_graph(x_data, y_data, x_labels, y_labels,
135                  xscale, yscale, aspect):
136       """ Plot the graph about y_data for each x_data.
137       """
138       d = len(y_data)
139       if not len(x_data) == len(y_data) == len(x_labels) == len(y_labels)\
140              == len(xscale) == len(yscale) == len(aspect):
141           raise ValueError("Arguments must have the same dimension.")
142       if d == 0:
143           raise ValueError("At least one data for plot.")
144       if d > 9:
145           raise ValueError("""So much data for plot in one figure.
146                               Please divide two or more data sets.""")
147
```

```
148       fig = plt.figure(figsize=(9, 8))
149       subplot_positioning = ['11', '21', '22', '22', '32', '32', '33', '33', '33']
150       axes = []
151       for n in range(d):
152           lmn = int(subplot_positioning[d-1] + str(n+1))
153           axes.append(fig.add_subplot(lmn))
154
155       for i, ax in enumerate(axes):
156           ymin, ymax = min(y_data[i]), max(y_data[i])
157           ax.set_aspect(aspect[i])
158           ax.set_xscale(xscale[i])
159           ax.set_yscale(yscale[i])
160           ax.set_xlabel(x_labels[i], fontsize=16)
161           ax.set_ylabel(y_labels[i], fontsize=16)
162           ax.set_ymargin(0.05)
163           ax.plot(x_data[i], y_data[i], 'o-')
164
165       fig.subplots_adjust(wspace=0.2, hspace=0.5)
166       fig.tight_layout()
167       plt.show()
168
169   if __name__ == '__main__':
170       L = 61
171       top = TopWindow()
172       per = Percolation(L=L)
173       count = 1
174
175       def pr():
176           global count
177           p = float(top.entry.get())
178           d = per.canvas.postscript(file="figure_%d(p=%s).eps" % (count, str(p)))
179           print "saved the figure to a eps file"
180           count += 1
181
182       def pushed():
183           global t, S, N
184           p = float(top.entry.get())
185           t, S, N = per.perc_cluster(p)
186           per.draw_canvas(per.lattice, L)
187
```

```
188     def b4_pushed():
189         x_data = [t[1:-1]]*2
190         y_data = [N[1:-1], S[1:-1]]
191         x_labels = [r'$t$', r'$t$']
192         y_labels = [r'$N$', r'$S$']
193         plot_graph(x_data, y_data, x_labels, y_labels,
194                     ['log', 'linear'], ['log', 'linear'], ['auto']*2)
195
196     def auto():
197         trial = 200
198         p = np.linspace(0.3, 1.0, 50)
199         N_p = []
200         S_p = []
201         perc_rate = []
202         for _p in p:
203             N_p_ = []
204             S_p_ = []
205             perc = 0
206             for i in range(trial):
207                 t, S, N = per.perc_cluster(_p)
208                 if S[-1] != 0:
209                     perc += 1
210                 N_p_.append(N[-1])
211                 S_p_.append(S[-1])
212             N_p.append(np.average(np.array(N_p_)))
213             S_p.append(np.average(np.array(S_p_)))
214             perc_rate.append(float(perc)/trial)
215         plot_graph([p], [perc_rate], [r'$p$'], [r'$P(p)$'],
216                     ['linear'], ['linear'], ['auto'])
217
218     top.show_window(pr, pushed, b4_pushed, auto)
219
```

3 実習課題

a. 感染可能な格子点が感染する確率を p とすると，本文で議論された簡単な伝染病のモデルが，リースのアルゴリズムと同じクラスターを生じる理由を説明せよ．広く伝染するために必要な p の最小の値はどれほどか．分割時間ごとに，すべての感染可能な格子点が同時に調べられ，確率 p で感染することを忘れないこと．感染した格子点の数 N がいろいろな p の値に対してどのように時刻 t(分割時間の数) に

依存するかを定めよ．これらを実行する直接的な方法は，プログラム perc_cluster を修正して，新しい周辺の点を見出す前にすべての周辺の点を調べて確率 p で占有されるようにすることである．第 15 章ではこのモデルがセルラーオートマトンの 1 つの例であることを学ぶ．

2 で示したようなアルゴリズムを用いたとき，このモデルはリースのアルゴリズムと同じクラスターを生じる．これは，この 2 つのモデルの間の相違というのが，離散的な時間間隔を導入しているかどうかの違いだけにある，ということに起因する．すなわち，リースのアルゴリズムにおいては格子の成長は一回にひとつずつであったが，伝染病モデルにおいては，分割時間ごとに，そのとき感染可能な格子点すべてに対して，占有するかどうかの操作が行われている．しかしながらこれは，調べられたが占有されなかった点を，まだ調べられていない格子点と区別していることによって，本質的な違いを生まない．実際，伝染病のモデルにおいても，結局は調べる格子 1 個 1 個に対して操作を順番に行っているのであり，複数の感染した格子点に囲まれていようが感染確率は p で変わらないことから，このアルゴリズムで生成された図形は，リースのアルゴリズムによって形成されたものと同じものであるということができる．

次に，感染した格子点の数 N が時間 t に対してどのように変動するかということを，さまざまな p の値に対して観察した．この結果を図 1，2，3，4(各上段) に示す．図から分かるように，どの p に対しても，$\ln N$ はほぼ $\ln t$ に比例しているように見える．また，$p = 0.3$ から $p = 1.0$ の間の p について，パーコレーション確率 $P(p)$ を試行回数 200 回のうち何回パーコレートしたかで求め，これを図 5 に示す．図から，パーコレーション閾値 p_c はおよそ $p = 0.6$ であることが分かる．これは，先ほどの考察で伝染病のモデルがリースのアルゴリズムと等価であることを示したが，このことから要求されるとおりに，通常のサイト・パーコレーションにおけるパーコレーション閾値と同じ値を持つことを表している．

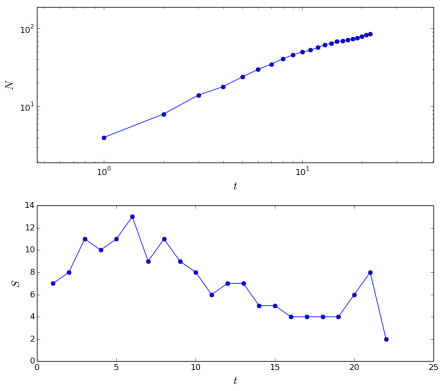

図 1　$p = 0.5$ のとき，時刻 t と N，S の関係

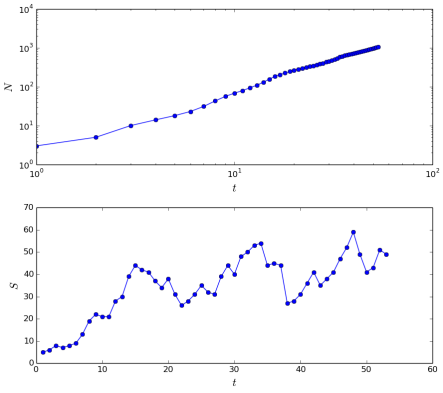

図 2　$p = 0.6$ のとき，時刻 t と N，S の関係

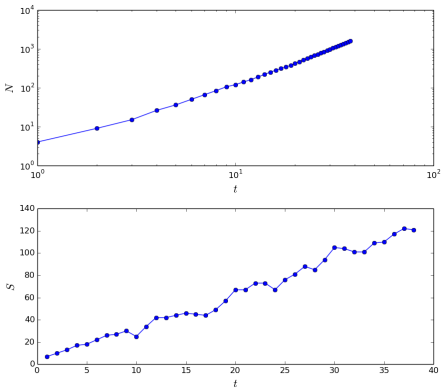

図 3　$p = 0.7$ のとき，時刻 t と N，S の関係

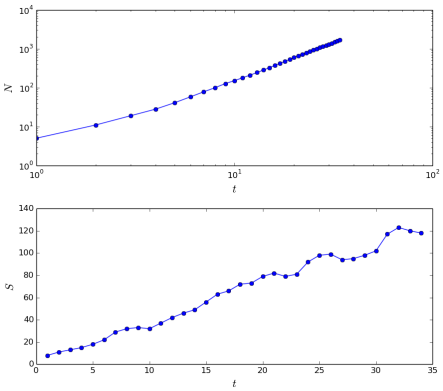

図 4　$p = 0.8$ のとき，時刻 t と N，S の関係

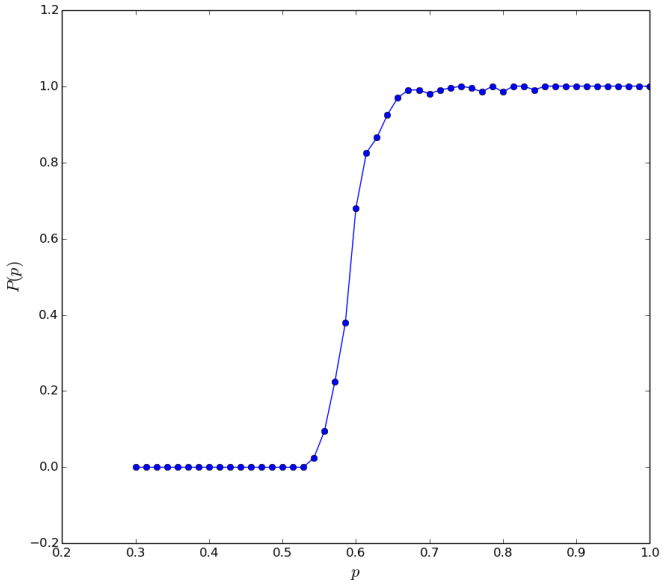

図 5　$p = 0.3$ から $p = 1.0$ までの p における，パーコレーション確率 $P(p)$

4　まとめ

簡単な伝染病のモデルについて，その性質のフラクタル性について考えることができた．

参考文献

[1] 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.