

# 計算機実習 問題 14.7 侵略型パーコレーション

早稲田大学先進理工学部物理学科 B4 藤本将太郎

2015 年 5 月 29 日

## 1 シミュレーションの目的

侵略型パーコレーションとして知られている動的な過程は、油を含んだ多孔性の媒体に水を押しこむとき生じる油と水の界面の形をモデル化するのに用いられる。そのアイデアは可能な限り多くの油を回収するために水を用いることにある。この過程では、水のクラスターが最も抵抗の少ない経路を通して油の中に成長していく。 $L \times 2L$  の格子を考え、初め左側の端が水 (侵略者) で占有されているとする。侵略者に対する抵抗の強さは、0 と 1 間の一様乱数で格子の各点に与えられ、侵略の過程の間固定されたままである。すべての侵略者の格子点の最隣接点が周辺の点となる。各分割時間に、最小の乱数を持つ周辺の点が侵略者によって占有され、その点の油 (防衛者) は置き換えられる。侵略者のクラスターは格子の左右の端を結ぶ経路が形成されるまで成長する。境界の影響を最小にするために、周期的境界条件が上下の端に対して用いられ、すべての量は格子の中心の  $L \times L$  の領域のみについて測定される。関心のある主な量は、侵略者によって占有された格子点の割合と、 $r + dr$  の間の値の乱数を持つ格子点が占有されている確率  $P(r)$  である。問題 14.7 では、侵略型パーコレーションのモデルの特性を調べる。

## 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 侵略型パーコレーションクラスターを作成するプログラム

このプログラムでは、先に述べたアルゴリズムにしたがって、周辺の点の中で最も抵抗力の低い格子点をクラスターに取り込んでいく。取り込まれたクラスターの周囲の点を更に周辺の点に加え、これを繰り返すことで侵略型パーコレーションのクラスターを作成する。シミュレーションの終了条件は、クラスターが系の右端に到達した時である。

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

```

7  import sys
8  import numpy as np
9  import scipy.optimize as optimize
10 import matplotlib.pyplot as plt
11
12 class Percolation:
13
14     def __init__(self, Lx=40, Ly=20):
15         self.sub = None
16         self.Lx = Lx
17         self.Ly = Ly
18
19     def perc_cluster(self):
20         self.lattice = np.random.random([self.Lx+2, self.Ly])
21         Lx = self.Lx
22         Ly = self.Ly
23
24         # 左端はすべて占有サイト
25         self.lattice[:1,:] = 1.5
26
27         self.lattice[Lx+1,:] = 0
28         if self.sub is None or not self.sub.wininfo_exists():
29             lattice = self.lattice
30             ne = [(0, -1), (0, 1), (-1, 0), (1, 0)]
31
32             # 周辺の点の座標を辞書形式で保持する
33             nnsite = {(1, y): lattice[1, y] for y in range(Ly)}
34
35             percolate = False
36             while len(nnsite) != 0 and percolate == False:
37
38                 # 周辺の点で最も値の小さい格子の座標を mm に
39                 mm = min([(v, k) for k, v in nnsite.items()])[1]
40
41                 lattice[mm] = 1
42                 del nnsite[mm]
43
44                 # mm の周辺の点の座標をリスト nn に (y 方向に周期境界条件を適用)
45                 nn = [(mm[0] + nx, (mm[1] + ny)%Ly) for nx, ny in ne
46                     if lattice[mm[0] + nx, (mm[1] + ny)%Ly] < 1]

```

```

47
48         # nnの中で既に nnsite に含まれているものを除く --> nnlist
49         nnlist = list(set(nn) - set(nnsite.keys()))
50
51         # nnsite に新たな周辺の点を追加する
52         for n in nnlist:
53             nnsite[n] = lattice[n]
54
55         # 占有された格子点が右端である時、パーコレートしたと見なす
56         if mm[0] == Lx:
57             percolate = True
58
59         self.lattice = lattice[1:-1,:]
60         return self.lattice
61
62     def draw_canvas(self, rect, Lx, Ly):
63         default_size = 640 # default size of canvas
64         r = int(default_size/(2*Lx))
65         fig_size_x = 2*r*Lx
66         fig_size_y = 2*r*Ly
67         margin = 10
68         sub = Toplevel()
69
70         sub.title('invasion percolation')
71         self.canvas = Canvas(sub, width=fig_size_x+2*margin,
72                               height=fig_size_y+2*margin)
73         self.canvas.create_rectangle(margin, margin,
74                                     fig_size_x+margin, fig_size_y+margin,
75                                     outline='black', fill='white')
76         self.canvas.pack()
77
78         c = self.canvas.create_rectangle
79
80         site = np.where(rect == 1)
81         for m, n in zip(site[0], site[1]):
82             c(2*m*r+margin, 2*n*r+margin,
83               2*(m+1)*r+margin, 2*(n+1)*r+margin,
84               outline='black', fill='black')
85
86     def get_fractal_dim(self, trial=20, Lmin=20, Lmax=40, Lsample=10):

```

```

87
88     # Lmin から Lmax の間の整数値で, できるだけ log にしたとき等間隔になるように
89     L = np.array([int(i) for i
90                   in np.logspace(np.log10(Lmin), np.log10(Lmax), Lsample)])
91
92     M_L = []
93     for l in L:
94         self.Lx = l*2
95         self.Ly = l
96         m_L = 0
97         for i in range(trial):
98             lattice = self.perc_cluster()
99
100            # 中心の L x L 格子中の占有サイト数を合計
101            m_L += np.sum(lattice[int(l/2)+1:l+int(l/2),:] == 1)
102
103            M_L.append(m_L/float(trial))
104            print "L = %d, M_L = %f" % (l, M_L[-1])
105
106     M_L = np.array(M_L)
107
108     def fit_func(parameter0, L, M_L):
109         log = np.log
110         c1 = parameter0[0]
111         c2 = parameter0[1]
112         residual = log(M_L) - c1 - c2*log(L)
113         return residual
114
115     parameter0 = [0.1, 2.0]
116     result = optimize.leastsq(fit_func, parameter0, args=(L, M_L))
117     c1 = result[0][0]
118     D = result[0][1]
119     print "D = %f" % D
120
121     def fitted(L, c1, D):
122         return np.exp(c1)*(L**D)
123
124     fig = plt.figure("Fractal Dimesion")
125     ax = fig.add_subplot(111)
126     ax.plot(L, M_L, '-o', label=r"$M(L)$")

```

```

127         ax.plot(L, fitted(L, c1, D), label="fit func: D = %f" % D)
128         ax.set_xlabel(r'$\ln L$', fontsize=16)
129         ax.set_ylabel(r'$\ln M(L)$', fontsize=16)
130         ax.set_xscale('log')
131         ax.set_yscale('log')
132         ax.set_ymargin(0.05)
133         fig.tight_layout()
134         plt.legend(loc='best')
135         plt.show()
136
137     class TopWindow:
138
139         def quit(self):
140             self.root.destroy()
141             sys.exit()
142
143         def show_window(self, title="title", *args):
144             self.root = Tk()
145             self.root.title(title)
146             frames = []
147             for i, arg in enumerate(args):
148                 frames.append(Frame(self.root, padx=5, pady=5))
149                 for k, v in arg:
150                     Button(frames[i], text=k, command=v).pack(expand=YES, fill='x')
151                     frames[i].pack(fill='x')
152             f = Frame(self.root, padx=5, pady=5)
153             Button(f, text='quit', command=self.quit).pack(expand=YES, fill='x')
154             f.pack(fill='x')
155             self.root.mainloop()
156
157 if __name__ == '__main__':
158     Lx = 40
159     Ly = 20
160     top = TopWindow()
161     per = Percolation(Lx, Ly)
162     count = 1
163
164     def pr():
165         global count
166         d = per.canvas.postscript(file="figure_%d.eps" % count)

```

```

167         print "saved the figure to a eps file"
168         count += 1
169
170     def pushed():
171         per.perc_cluster()
172         per.draw_canvas(per.lattice, Lx, Ly)
173
174     def b4_pushed():
175         trial = 100
176         Lmin = 20
177         Lmax = 100
178         Lsample = 10
179         per.get_fractal_dim(trial, Lmin, Lmax, Lsample)
180
181     run = (('run', pushed), ('save canvas to sample.eps', pr))
182     run2 = (('calculate D', b4_pushed),)
183     top.show_window("Invasion Percolation", run, run2)
184

```

### 3 実習課題

- a.  $20 \times 40$  の格子上に侵略型パーコレーションのクラスターを生成し，得られたクラスターの定性的な性質について述べよ．

作成したプログラムを用いて， $20 \times 40$  の格子上に侵略型パーコレーションのクラスターを作成した．得られたクラスターの一つを図 1 に示す．この図からわかるように，左から侵略してくるクラスターは，穴の空いた複雑な構造をしており，フラクタル図形としての特徴を備えていることがわかる．問題 14.6 で議論したイーデンモデルによるクラスターはコンパクト ( $D \simeq d$ ) であったのに対し，侵略型パーコレーションではこのように複雑なクラスターを形成するのは，その成長点の選択規則の違いによる．すなわち，イーデンモデルでは周辺の点からランダムに成長点を選んでいたが，侵略型パーコレーションのモデルでは，乱数の最小のものを選択するため，乱数の大きい格子が周辺の点に含まれると，いつまでも占有されることなく残ることになり，穴だらけの構造が出来上がるというわけである．また，右端にクラスターが到達した時点でシミュレーションを終了するため，クラスターは左右対称にはならず，格子右側のクラスターの占有率は，左側のそれに比べると小さいように見える．

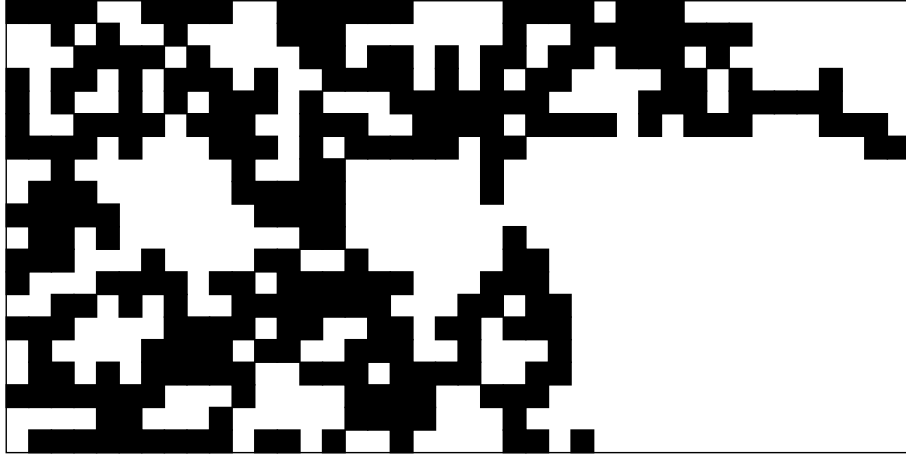


図1 20 × 40 の格子上で得られた侵略型パーコレーションのクラスター

- b. 侵略者が初めて右の端に達したとき、 $L \times 2L$  の格子の中央の  $L \times L$  の領域で、侵略者によって占有された格子点の数  $M(L)$  を少なくとも 20 回の試行について平均せよ。  $M(L) \sim L^D$  を仮定し、 $\ln M$  の  $\ln L$  に対するプロットから  $D$  の値を求めよ。得られた  $D$  の値と通常のパーコレーションのフラクタル次元とを比較せよ。（発表されている Wilkinson と Willemsen によって計算された  $M(L)$  の値は、 $L$  の 20 から 100 の範囲について各 2000 回の試行結果から得られている。）

$L$  の 20 から 100 の範囲の 10 個の  $L$  について、それぞれ 100 回の試行の平均からフラクタル次元  $D$  を計算した。横軸を  $\ln L$ 、縦軸を  $\ln M(L)$  としてプロットしたものを図 2 に示す。この傾きは  $D \simeq 1.92$  と求められた。これは通常のパーコレーションモデルによって得られたクラスターのフラクタル次元  $D \simeq 1.89$  と近い値となっている。また、侵略型パーコレーションクラスターの、大規模なシミュレーションによって得られたフラクタル次元の値も  $D \simeq 1.89$  であり、全く違う機構で生成されたクラスターのフラクタル次元が一致するということは大変興味深い。

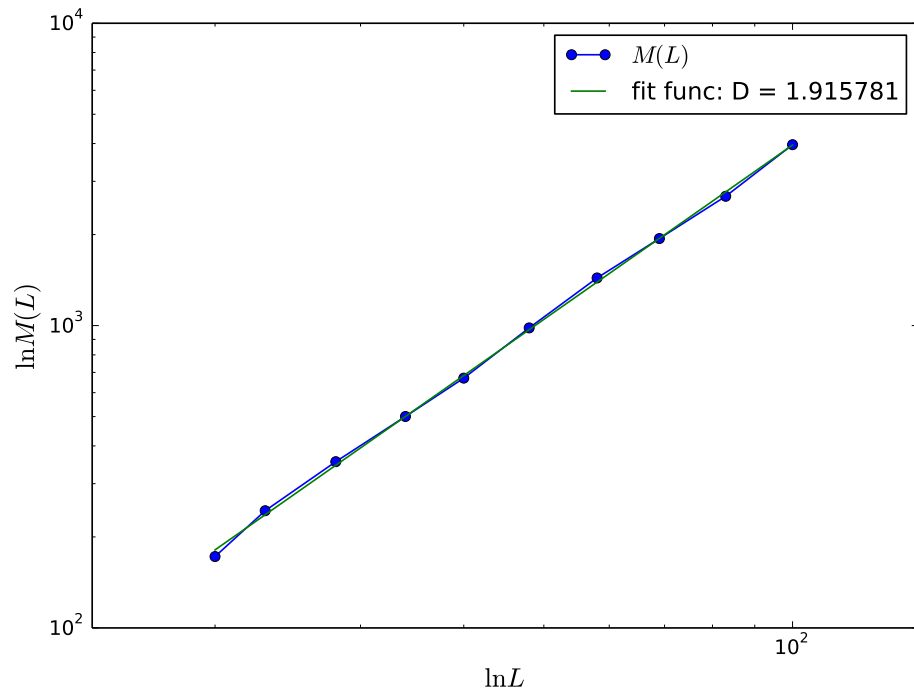


図2  $\ln M$  の  $\ln L$  に対するプロット

## 4 まとめ

侵略型パーコレーションモデルによるクラスターを作成し、そのフラクタル次元が通常のパーコレーションクラスターのフラクタル次元と等しくなることを見た。また、冒頭で侵略型パーコレーションモデルの誕生の経緯として、油を含んだ多孔性の媒体に水を押しこんで油を取り出すことを考えていたが、クラスターのフラクタル次元が  $D < d$  であることから、効率よく油を採取することはできないということもわかる。