

計算機実習 問題 14.9 拡散律速凝集

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 21 日

1 シミュレーションの目的

自然界には基本となる単位をランダムに付け加えていくことで成長するものを多く見ることができる。雪の結晶や地質学的な断層における稲妻形のひび割れの形成、バクテリアのコロニーの成長などはその例である。最近発展した多くのモデルが示す性質は、これらや他の多くの自然現象を、少数の統一原理によって理解する手がかりを与えてくれる。我々に大いなる洞察を与えてくれたモデルの 1 つは拡散律速凝集、すなわち DLA である。このモデルは、ランダムな運動がいかに美しい自己相似なクラスターを生み出すかを示す例となっている。

初めのステップは 1 つの種の粒子で 1 つの格子点を占有することである。次に、その種を中心とする大きな円の周囲から 1 つの粒子が放たれる。その粒子はランダムウォーク、つまり拡散しながら、種の 1 つの周辺の点に到達するとそこに付着する。そして、別の粒子が放たれ、クラスター内にある 2 個の粒子のどちらかの 1 つの周辺の点に到達してそこに付着するまでランダムウォークを行う。大きなクラスターが形成されるまで、このプロセスは繰り返される (典型的には数千から数 100 万回の程度)。問題 14.9 では、DLA クラスターの特性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 DLA クラスターを生成するプログラム

このプログラムは、DLA, SetParameter, Main の 3 つのクラスによって構成されている。クラス DLA は DLA クラスターを作成・描画する部分を担当している。SetParameter は、これまで使用したものと特に変化はなく、変数の値を設定するためのものである。Main は、研究課題を実際に実行するために用意したもので、課題 a,b,c のための関数と、 c でフラクタル次元を求めるための計算と描画の部分を含んでいる。

クラス DLA について述べることにすると、まず、呼び出しの時点で変数として `view` と `color` をとり、描画を行うかどうかを指定することができるようになっている。格子のサイズ L を決める際は、フラクタル次元が小さくとも $D \sim 1.3$ ほどであると仮定して、粒子数 N として $L = N^{0.78}$ で決定した。これより小さい値だと、成長したクラスターがはみ出してしまいうまくいかないことが多い。関数 `grow_cluster` によって、種の格子点の周りに N 個の粒子が付着するまでクラスターを成長させることができる。半径 $R_{\max} + 2$ の円周上の

点が、ランダムウォークする粒子の初期点として選ばれ、この点からランダムウォークを開始する。もし粒子が $2R_{\max}$ より遠くの位置に移動した場合、新たな初期点から再びランダムウォークを開始する。粒子がクラスターに接触したときに、その粒子は新たにクラスターの内部に取り込まれ、カウントを一つ追加して、新たな初期点を選ぶ。100 個ごとに格子の色を変えることもできるようにしてある。また、問題 c で取り組み、改善した箇所であるが、種の格子点からの距離 r によってランダムウォークの歩幅を変更するようにしてある。

クラス Main は問題 a,b,c のための関数と、問題 c で視野拡大法によりフラクタル次元を求めるため、view_expansion と plot の二つの関数を用意してある。view_expansion は問題 14 - 1 で使用したものである。得られた結果をグラフに表示し、関数 fitting で範囲を指定してフィッティングを行うようにした。

```
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 matplotlib.pyplot as plt
9  import random
10 import time
11
12 class DLA(object):
13
14     def __init__(self, N, view=True, color=True):
15         self.r = 3
16         self.N = N
17         self.view = view
18         self.color = color
19         self.L = int(self.N**(0.78))
20
21     if self.view:
22         self.default_size = 640 # default size of canvas
23         self.rsize = int(self.default_size/(2*self.L))
24         if self.rsize == 0:
25             self.rsize = 1
26         fig_size = 2*self.rsize*self.L
27         self.margin = 10
28         self.sub = Toplevel()
29         self.canvas = Canvas(self.sub, width=fig_size+2*self.margin,
30                               height=fig_size+2*self.margin)
31         self.c = self.canvas.create_rectangle
```

```

32         self.update = self.canvas.update
33
34         self.sub.title('DLA cluster')
35         self.c(self.margin, self.margin,
36               fig_size+self.margin, fig_size+self.margin,
37               outline='black', fill='white')
38         self.canvas.pack()
39         self.start_time = time.time()
40
41     def grow_cluster(self):
42         lattice = np.zeros([self.L*2+1, self.L*2+1], dtype=int)
43         # 種の格子点
44         self.center = self.L
45         lattice[self.center, self.center] = 1
46         if self.view:
47             c = self.c
48             rect = c((2*self.center-self.L)*self.rsize+self.margin,
49                     (2*self.center-self.L)*self.rsize+self.margin,
50                     (2*(self.center+1)-self.L)*self.rsize+self.margin-1,
51                     (2*(self.center+1)-self.L)*self.rsize+self.margin-1,
52                     outline='black', fill='black')
53
54         rn = np.random.rand
55
56     def reset():
57         """初期点の選択"""
58         theta = 2*np.pi*rn()
59         x = int((self.r+2)*np.cos(theta))+self.center
60         y = int((self.r+2)*np.sin(theta))+self.center
61         return x, y
62
63     x, y = reset()
64     n = 0
65     l = 1
66     while n < self.N:
67
68         # クラスターの周辺から遠いところでは歩幅を大きくする
69         r = np.sqrt((x-self.center)**2+(y-self.center)**2)
70         if r > self.r+2:
71             l = int(r-self.r-2)

```

```

72         if l == 0: l = 1
73     else: l = 1
74
75     # ランダムウォーク
76     p = rn()*4
77     if p < 1: x += 1
78     elif p < 2: x -= 1
79     elif p < 3: y += 1
80     else:      y -= 1
81
82     r = np.sqrt((x-self.center)**2+(y-self.center)**2)
83
84     # 中心点から離れた点で過程をやり直す
85     if r >= 2*self.r:
86         x, y = reset()
87         continue
88
89     judge = np.sum(lattice[x-1,y]+lattice[x+1,y]
90                    +lattice[x,y-1]+lattice[x,y+1])
91     # 粒子をクラスターに取り込む
92     if judge > 0:
93         lattice[x, y] = 1
94
95     # 描画
96     if self.view:
97         if self.color:
98             colors = ['#ff0000', '#ff8000', '#ffff00', '#80ff00',
99                      '#00ff00', '#00ff80', '#00ffff', '#0080ff',
100                     '#0000ff', '#8000ff', '#ff00ff', '#ff0080']
101             color = colors[int(n/100)%len(colors)]
102         else: color = "black"
103         rect = c((2*x-self.L)*self.rsize+self.margin,
104                 (2*y-self.L)*self.rsize+self.margin,
105                 (2*(x+1)-self.L)*self.rsize+self.margin-1,
106                 (2*(y+1)-self.L)*self.rsize+self.margin-1,
107                 outline=color, fill=color)
108         self.update()
109
110     # rmax の更新
111     if int(r)+1 > self.r:

```

```

112         self.r = int(r) + 1
113         x, y = reset()
114         n += 1
115     else:
116         if self.view:
117             self.end_time = time.time()
118             t = self.end_time-self.start_time
119             print "done; N = %d, time = " % self.N + str(t) + ' (s)'
120         return lattice
121
122 class SetParameter():
123
124     def show_setting_window(self, parameters, commands):
125         """ Show a parameter setting window.
126
127         parameters: A list of dictionaries {'parameter name': default_value}
128         commands: A list of dictionary {'name of button': command}
129         """
130         self.root = Tk()
131         self.root.title('Parameter')
132
133         frame1 = Frame(self.root, padx=5, pady=5)
134         frame1.pack(side='top')
135         self.entry = []
136         for i, parameter in enumerate(parameters):
137             label = Label(frame1, text=parameter.items()[0][0] + ' = ')
138             label.grid(row=i, column=0, sticky=E)
139             self.entry.append(Entry(frame1, width=10))
140             self.entry[i].grid(row=i, column=1)
141             self.entry[i].delete(0, END)
142             self.entry[i].insert(0, parameter.items()[0][1])
143         self.entry[0].focus_set()
144
145         frame2 = Frame(self.root, padx=5, pady=5)
146         frame2.pack(side='bottom')
147         self.button = []
148         for i, command in enumerate(commands):
149             self.button.append(Button(frame2, text=command.items()[0][0],
150                                     command=command.items()[0][1]))
151             self.button[i].grid(row=0, column=i)

```

```

152
153         self.root.mainloop()
154
155     class Main(object):
156
157         def __init__(self):
158             import sys
159             self.sp = SetParameter()
160             self.dla = None
161             self.b = None
162             self.sp.show_setting_window([{'N': 200}],
163                                         [{'a': self.exp_a}, {'b': self.exp_b}, {'c': self.exp_c},
164                                          {'c:fit': self.fitting}, {'save': self.pr},
165                                          {'quit': sys.exit}])
166
167         def exp_a(self):
168             self.N = int(self.sp.entry[0].get())
169             self.dla = DLA(self.N)
170             lattice = self.dla.grow_cluster()
171
172         def exp_b(self):
173             trial = 3000
174             self.dla2 = DLA(2, view=False)
175             self.dla2.L = 6
176             distribution = {'p': 0, 'q': 0, 'r': 0, 's': 0}
177             # 分類
178             for i in range(trial):
179                 lattice = self.dla2.grow_cluster()
180                 l = lattice[self.dla2.L-1:self.dla2.L+2,
181                             self.dla2.L-1:self.dla2.L+2]
182                 if np.sum(l) == 2:
183                     distribution['r'] += 1
184                 elif np.sum(l[0,1]+l[1,0]+l[1,2]+l[2,1]) == 1:
185                     distribution['p'] += 1
186                 elif max(max(np.sum(l, 0)), max(np.sum(l, 1))) == 3:
187                     distribution['s'] += 1
188                 else:
189                     distribution['q'] += 1
190             for k, v in distribution.items():
191                 distribution[k] = float(v)/trial

```

```

192         distribution['p'] = distribution['p']/2.
193         distribution['q'] = distribution['q']/2.
194         print 'trial = %d' % trial
195         print distribution
196
197     def exp_c(self):
198         self.N = int(self.sp.entry[0].get())
199         self.dla3 = DLA(self.N, view=False)
200         self.lattice = self.dla3.grow_cluster()
201         self.view_expansion()
202         self.plot()
203
204     def view_expansion(self):
205         lattice = self.lattice
206         center = self.dla3.center
207         M_b = []
208         s = np.sum
209         ave = np.average
210         append = M_b.append
211         for k in range(1, center):
212             nonzero = np.nonzero(lattice[k:-k,k:-k])
213             tmp = np.array([0])
214             for i, j in zip(nonzero[0]+k, nonzero[1]+k):
215                 tmp = np.append(tmp, s(lattice[i-k:i+k+1, j-k:j+k+1]))
216             append(ave(tmp))
217         self.b = np.array([2.*k+1 for k in range(1, center)])
218         self.M_b = np.array(M_b)
219
220     def plot(self):
221         fig = plt.figure("Fractal Dimension")
222         self.ax = fig.add_subplot(111)
223         self.ax.plot(self.b, self.M_b, '-o')
224         self.ax.set_xlabel(r'$b$', fontsize=16)
225         self.ax.set_ylabel(r'$M(b)$', fontsize=16)
226         self.ax.set_xscale('log')
227         self.ax.set_yscale('log')
228         self.ax.set_ymargin(0.05)
229         fig.tight_layout()
230         plt.show()
231

```

```

232     def fitting(self):
233         if self.b == None:
234             return
235         import scipy.optimize as optimize
236
237         def fit_func(parameter0, b, M_b):
238             log = np.log
239             c1 = parameter0[0]
240             c2 = parameter0[1]
241             residual = log(M_b) - c1 - c2*log(b)
242             return residual
243
244         def fitted(b, c1, D):
245             return np.exp(c1)*(b**D)
246
247         cut_from = int(raw_input("from ? (index) >>> "))
248         cut_to = int(raw_input("to ? (index) >>> "))
249         cut_b = np.array(list(self.b)[cut_from:cut_to])
250         cut_M_b = np.array(list(self.M_b)[cut_from:cut_to])
251         parameter0 = [0.1, 2.0]
252         result = optimize.leastsq(fit_func, parameter0, args=(cut_b, cut_M_b))
253         c1 = result[0][0]
254         D = result[0][1]
255
256         self.ax.plot(cut_b, fitted(cut_b, c1, D),
257                     lw=2, label="fit func: D = %f" % D)
258         plt.legend(loc='best')
259         plt.show()
260
261     def pr(self):
262         import tkinter as tk
263         import os
264
265         if self.dla is None:
266             print "first you should run 'a'."
267             return
268         fTyp=[('eps file','*.eps'), ('all files','*')]
269         filename = tkFileDialog.asksaveasfilename(filetypes=fTyp,
270             initialdir=os.getcwd(), initialfile="figure_1.eps")
271         if filename == None:

```



```

272         return
273         d = self.dla.canvas.postscript(file=filename)
274
275 if __name__ == '__main__':
276
277     Main()
278

```

3 実習課題

- a. 正方格子上に拡散律速凝集のクラスターを生成するプログラムを書け。 R_{\max} を原点からクラスターを構成するすべての粒子までの最大距離とすると、各粒子は半径 $r = R_{\max} + 2$ の円上のランダムな点から出発するとせよ。コンピュータの計算時間を節約するために、種からの距離が $2R_{\max}$ に到達した粒子は取り除き、新たな粒子を半径 $r = R_{\max} + 2$ の円周上にランダムに置く。格子の大きさは $L \sim 31$ とせよ。到着時間に応じてクラスター内の格子点に色をつけよ。たとえば、初めの 100 個の格子点には青色を、次の 100 個には黄色をといった具合にするとよい。クラスターのどの部分が速く成長するか。得られたクラスターがフラクタルに見えるなら、フラクタル次元をめのこ (目視) で見積もれ (専門家は数パーセントの精度で D をめのこで推定することができる！)。

作成したプログラムを用いて DLA クラスターを作成し、 $N = 1000$ とした時に得られたクラスターの様子を図 1 に示した。図から、クラスターの中で成長しやすいのは、より外側に位置する箇所であり、外側に伸びたクラスターの間に挟まれた初期の枝は、後になって成長することはあまりないことが分かる。このクラスターのフラクタル次元をめのこで見積もると、だいたい $D \sim 1.7$ ほどであると予想される。

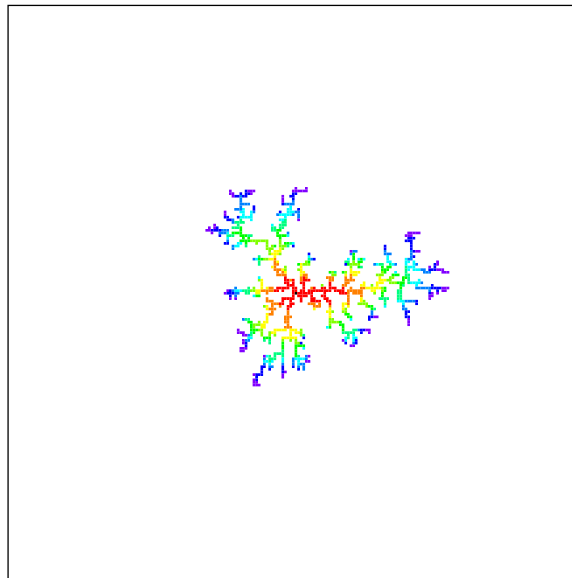


図 1 $N = 1000$ の時に得られた DLA クラスター。 N の 100 個ごとに色を変えてある。

- b. $t = 0$ で、正方格子の 4 個の周辺 (成長) 点はおおの成長確率，すなわち，クラスターの一部となる確率 $p_i = 1/4$ を持っている． $t = 1$ では，そのクラスターの質量は 2 となり，6 個の周辺の点を持つことになる．どの格子点が周辺の点になるか理解し，それらの成長確率は一定でないことを確認せよ．モンテカルロ・シミュレーションを行い，2 つの周辺の点については $p_i = 2/9 (= 0.222\cdots)$ であり，他は $p_i = 5/36 (= 0.1388\cdots)$ であることを確かめよ．問題 14.10 で成長確率を決定する更に直接な方法について議論する．

$t = 1$ では，図 2 に示したように 6 個の周辺の点をもっている．3000 回の試行によるモンテカルロ・シミュレーションで，図のそれぞれの位置に粒子が付着する確率確率 p, q, r, s を求めた．結果は $p = 0.141, q = 0.1375, r = 0.22166666666666668, s = 0.22133333333333333$ であった．すなわち，2 つの粒子に対して，クラスターの長さが長くなるような位置に，粒子が付着しやすいということがわかった．これは，得られたクラスターの形からも確かめられることであり，クラスター成長していくときに，その形を丸くするようではなく，細長く伸びていくような成長の仕方をしていることも観察できている．

ここで， p, q, r, s の判別について述べておくこととする．中心の点と周囲 1 マスの計 9 個の格子点で作られる格子を考える (図 2 の赤枠で囲まれた格子点)．まず，その格子内の粒子数が 2 であるものは r であると分かる．次に，中心の点の上下左右の 4 点の粒子数の和が 1 となるのは粒子が p で付着する場合のみである．最後に，行方向と列方向で和を計算し，その値に 3 が含まれているのは s のときのみであるので，これで s と q を区別することができる． p と q に対しては，二通りの付着の仕方が考えられるため，2 で割る必要があり，このようにしてそれぞれの付着確率 p, q, r, s を求めることができた．

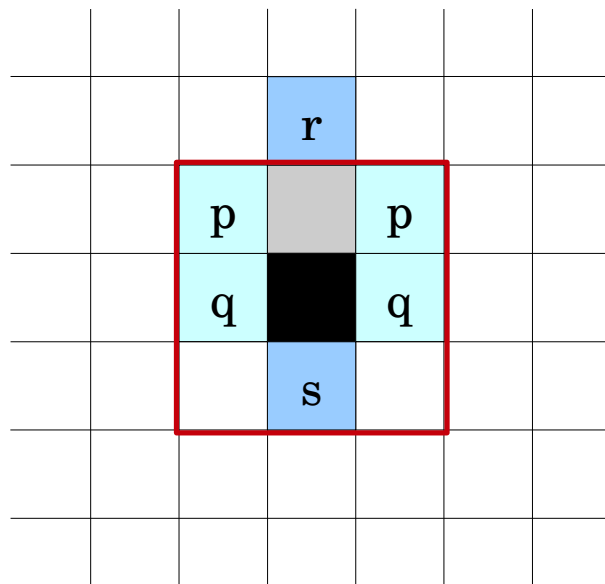


図 2 $t = 1$ における 6 つの周辺の点．

- c. クラスターの周辺の点から遠く離れたところでのランダムウォークに CPU 時間のほとんどが費やされてしまうので，DLA クラスターの生成に使用しているプログラムは能率的ではないだろう．この問題を克服するための方法がいくつかある．1 つの方法はランダムウォークの歩幅をクラスターから離れる

にしたがって大きくすることである．たとえば，粒子が距離 $R > R_{\max}$ にいて， $R - R_{\max} - 1$ が単位の格子間隔よりも大きいならば，この距離に等しいかそれ以上の歩幅を許すことにする．粒子がクラスターに非常に近い場合は，歩幅を単位の格子間隔にとる．他の可能な修正についてミーキン (Meakin) による議論がある [1]．自分のプログラム (あるいは教科書 [2] にあるプログラム die) を修正し，正方格子上の 2 次元の拡散律速クラスターのフラクタル次元を求めよ．

ランダムウォークの歩幅が，中心からの距離 r が大きくなるほど大きくなるように変更を加え，クラスターより遠い位置でのランダムウォークに費やす時間が出来るだけ小さくなるようにプログラムを変更した．得られた拡散律速クラスターのフラクタル次元を，視野拡大法によって求めた． $N = 1000$ として視野のスケール b とその範囲内の粒子数 $M(b)$ の関係を両対数グラフに表し，その傾き D を求めた．これを図 3 に示す．傾き D は $D = 1.616042$ と求められた．

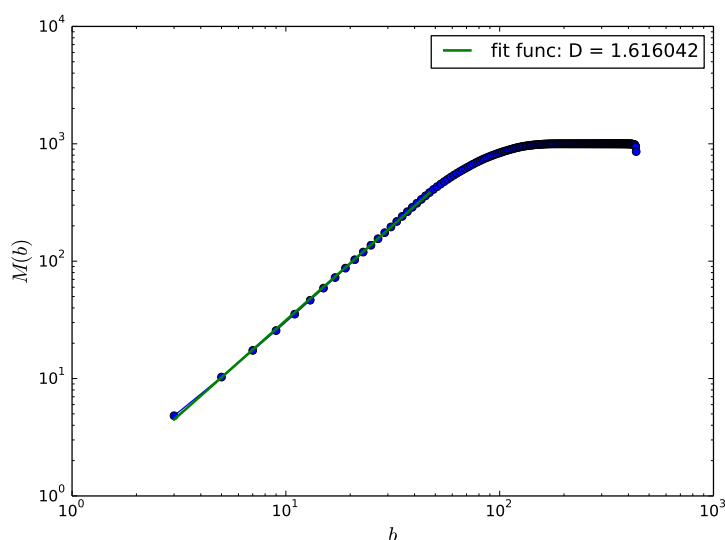


図 3 $N = 1000$ のとき，得られた DLA クラスターのフラクタル次元．

4 まとめ

自然界の様々な場所で見られるパターンの基本的なモデルである，拡散律速凝集のクラスターについて理解することができた．見ただけでフラクタル次元が正確に予想できるようになるまで精進できればと思う．

参考文献

- [1] Paul Meakin. The growth of rough surfaces and interfaces. *Physics Reports*, Vol. 235, No. 45, pp. 189 – 289, 1993.
- [2] ハーベイ・ゴールド, ジャン・トボchnik. 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.