計算機実習 問題 14.6 イーデン・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

成長モデルのさらに簡単な例が 1958 年にイーデンによって，細胞のコロニーの成長のシミュレーションを行うために提案された．結果の質量分布はフラクタルではないことがわかるが，イーデン成長のアルゴリズムの記述は，フラクタルな成長モデルの一般的な性質を示している．問題 14.6 ではイーデンクラスターのいくつかの性質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　イーデンモデルにより正方格子上でクラスターを生成するプログラム

イーデンモデルのアルゴリズムは，以下のようにまとめられる．

1. 種を原点に置き，占有されていないすべての隣接格子点 (成長点と呼ぶ) を探す．
2. 1 つの成長点がランダムに選ばれ，占有される．このとき選ばれた成長点は必ず占有されることに注意する．すなわちこれまでのモデルにおいて $p = 1$ としたことに対応する．
3. 新しく占有された格子点を成長点のリストから除外し，新しい成長点が加えられる．
4. 2,3 の過程を，クラスターが考えている格子サイズの端から端まで連結するまで繰り返すとする．

このモデルと前のモデルとの基本的な相違は，調べられたすべての格子点が占有されることであり，問題 14.5 の伝染病のモデルとの対応で言えばどの格子点も永久に”免疫”を持つことがない．

以下に，実際に作成したプログラムの内容を示すが，ダイアログの表示，ウィンドウへの格子の描画など，ほとんどの部分は問題 14.5 で使用したものをそのまま利用できる．したがって，変更した点のみを述べることにすると，関数 perc_cluster の内部での挙動は，上で説明したアルゴリズムを実現するように変更されている．1 回に 1 つずつ成長させていくので，リストとして扱っていた nextseed を廃止し，for 文を使うことを避けた．成長点のリスト nnsite からランダムに要素を取り出すために random モジュールの choice メソッドを用いた．これは与えられたリストの要素数と，生成された乱数の値によって，返す値を決定するものである．

次に，関数 b4_pushed について説明すると，これは種を中心とした半径 r の円の中に含まれる，占有された格子点の数を計算し，それを両対数グラフにして表示するものである．rlattice が実際の格子における種か

らの距離を表現するものとなっており，rlattice のなかで距離 r(プログラム上では_r) より小さい値をもつところの座標が s に記録される．この座標 s は格子点より右下の 1/4 の部分のものであり，格子の中心を (0,0) としているので，そのまま使うには都合の悪い形をしている．したがって，次の set_in_r で実際の格子上での座標に置き換えられた (x, y) の組にし，座標の重複は除かれる．その 2 行後で (x の列，y の列) の形に整形されて，M_r でその座標において値が 1 であるものの総数を計算している．半径 r は 2^i $(i = 1, 2, \cdots)$ ととり，r は格子の中心から端までの距離を超えないようにしている．

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

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

class Percolation:

    def __init__(self, L=61):
        if L % 2 == 0:
            raise ValueError("lattice size L must be odd number.")
        self.sub = None
        self.lattice = None
        self.L = L # lattice size

    def perc_cluster(self):
        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
        center = (self.L//2) + 1
        self.lattice[center, center] = 1
        nextseed = [(center, center)]
        if self.sub is None or not self.sub.winfo_exists():
            lattice = self.lattice
            L = self.L
            rn = np.random.random
            choice = random.choice
```

```
            ne = [(0, -1), (0, 1), (-1, 0), (1, 0)]
            nnsite = set([(center+nx, center+ny) for nx, ny in ne])
            t = [0] # time
            S = [4] # a number of growing sites
            N = [1] # a number of occupied sites
            percolate = False
            l = set([])
            while percolate == False:
                nn = choice(list(nnsite))
                nnsite.remove(nn)
                lattice[nn] = 1
                i, j = nn
                nnsite = nnsite | set([(i+nx, j+ny) for nx, ny in ne
                                        if lattice[i+nx, j+ny] == 0])
                if i == 1:
                    l.add(’top’)
                if i == L:
                    l.add(’bottom’)
                if j == 1:
                    l.add(’left’)
                if j == L:
                    l.add(’right’)

                glsite = (np.array([a[0] for a in list(nnsite)]),
                          np.array([a[1] for a in list(nnsite)]))
                lattice[glsite] = -1
                if (’top’ in l and ’bottom’ in l) or \
                   (’right’ in l and ’left’ in l):
                    percolate = True

                t.append(t[-1]+1)
                S.append(len(nnsite))
                N.append(np.sum(lattice == 1))
            self.lattice = lattice[1:-1, 1:-1]

        return t, S, N

    def draw_canvas(self, rect, L, show=1):
        default_size = 640 # default size of canvas
        r = int(default_size/(2*L))
```

```
        if r == 0:
            r = 1
        fig_size = 2*r*L
        margin = 10
        sub = Toplevel()

        self.canvas = Canvas(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 == show) # 1: occupied site, -1: growing site
        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='', fill='black')

class TopWindow:

    def quit(self):
        self.root.destroy()
        sys.exit()

    def show_window(self, pr, pushed, b4_pushed, b5_pushed):
        self.root = Tk()
        self.root.title('Percolation')
        f1 = Frame(self.root, padx=5, pady=5)
        b1 = Button(f1, text='run (and show figure)', command=pushed)
        b1.pack(side='top', expand=YES, fill='x')
        f1.pack(fill='x')

        f2 = Frame(self.root, padx=5, pady=5)
        b5 = Button(f2, text='show growing site', command=b5_pushed)
        b5.pack(side='top', expand=YES, fill='x')

        b4 = Button(f2, text='plot graph', command=b4_pushed)
```

```
        b4.pack(expand=YES, fill='x')

        b2 = Button(f2, text='save canvas to sample.eps', command=pr)
        b2.pack(expand=YES, fill='x')
        f2.pack(fill='x')

        f3 = Frame(self.root, padx=5, pady=5)
        b3 = Button(f3, text='quit', command=self.quit)
        b3.pack(expand=YES, fill='x')
        f3.pack(fill='x')

        self.root.mainloop()


if __name__ == '__main__':
    L = 201
    top = TopWindow()
    per = Percolation(L=L)
    count = 1

    def pr():
        global count
        d = per.canvas.postscript(file="figure_%d.eps" % count)
        print "saved the figure to a eps file"
        count += 1

    def pushed():
        global t, S, N
        t, S, N = per.perc_cluster()
        per.draw_canvas(per.lattice, L)

    def b5_pushed():
        if per.lattice == None:
            per.perc_cluster()
        per.draw_canvas(per.lattice, L, show=-1)

    def b4_pushed():
        if per.lattice == None:
            per.perc_cluster()
        sqrt = np.sqrt
```

```
        c = (L//2)
        rlattice = np.array([sqrt(x**2 + y**2) for x in range(c)
                             for y in range(c)]).reshape(c, c)
        i = 1
        _r = 2
        r, M = [], []
        while _r <= c:
            s = np.where(rlattice <= _r)
            set_in_r = set(zip(np.append(s[0], [s[0], -s[0], -s[0]])+c,
                                   np.append(s[1], [-s[1], s[1], -s[1]])+c))
            l = list(set_in_r)
            site = (np.array([p[0] for p in l]), np.array([p[1] for p in l]))
            M_r = np.sum(per.lattice[site] == 1)
            r.append(_r)
            M.append(M_r)
            i += 1
            _r = 2**i

        log = np.log
        def fit_func(parameter0, r, M_r):
            c1 = parameter0[0]
            c2 = parameter0[1]
            residual = log(M_r) - c1 - c2*log(r)
            return residual

        r = np.array(r)
        M = np.array(M)
        x = np.logspace(np.log10(np.min(r))-0.1, np.log10(np.max(r))+0.1, 10)
        parameter0 = [0.1, 2.0]
        result = optimize.leastsq(fit_func, parameter0, args=(r, M))
        c1 = result[0][0]
        D = result[0][1]

        def fitted(r, c1, D):
            return np.exp(c1)*(r**D)

        fx = fitted(x, c1, D)

        fig = plt.figure()
        ax = fig.add_subplot(111)
```

```
        plt.subplots_adjust(bottom=0.14)
        plt.subplots_adjust(right=0.68)
        ax.set_xscale('log')
        ax.set_yscale('log')
        ax.set_xlabel(r'$\ln r$', fontsize=16)
        ax.set_ylabel(r'$\ln M$', fontsize=16)
        ax.set_ymargin(0.05)
        ax.plot(r, M, 'o')
        ax.plot(x, fx, '-', color='black',
                label='\n$\mathrm{fitted\ curve}$\n$(D= %1.2f)$' % D)
        ax.legend(bbox_to_anchor=(1.05, 1.0), loc='upper left', borderaxespad=0)
        plt.show()

    top.show_window(pr, pushed, b4_pushed, b5_pushed)

```

3 実習課題

a. イーデンモデルにしたがって正方格子上でクラスターを形成せよ．周辺の点を無制限に占有し続けるならば，何が起こるだろうか．問題 14.3 の手順に従い，種の格子点から距離 r の範囲内の占有された格子点の数 $M(r)$ を求めよ．十分に大きな r に対して $M(r) \sim r^D$ を仮定し，r に対する M の両対数プロットの傾きから D を求めよ．得られたデータからイーデンクラスターはコンパクトであると結論できるか．

作成したプログラムを用いて，イーデンモデルにより正方格子上でクラスターを形成した．$L = 201$ としたときのクラスターの様子を図 1 に示す．この図から，イーデンモデルによるクラスターは，これまで生成したクラスターとは異なり，クラスターの内部の点はほとんどが占有されていて，穴の多いフラクタル的な構造とはなっていないことが分かる．

次に，種の格子点から距離 r の範囲内の占有された格子点の数 $M(r)$ を求め，r に対する M の両対数プロットを図 2 に示した．このグラフより，$M(r) \sim r^D$ と表すことができて，その傾き D は $D \approx 2.00$ となることが分かる．すなわち，フラクタル次元 D が 2 でユークリッド次元と等しく，したがってイーデンクラスターはコンパクトであると言える．

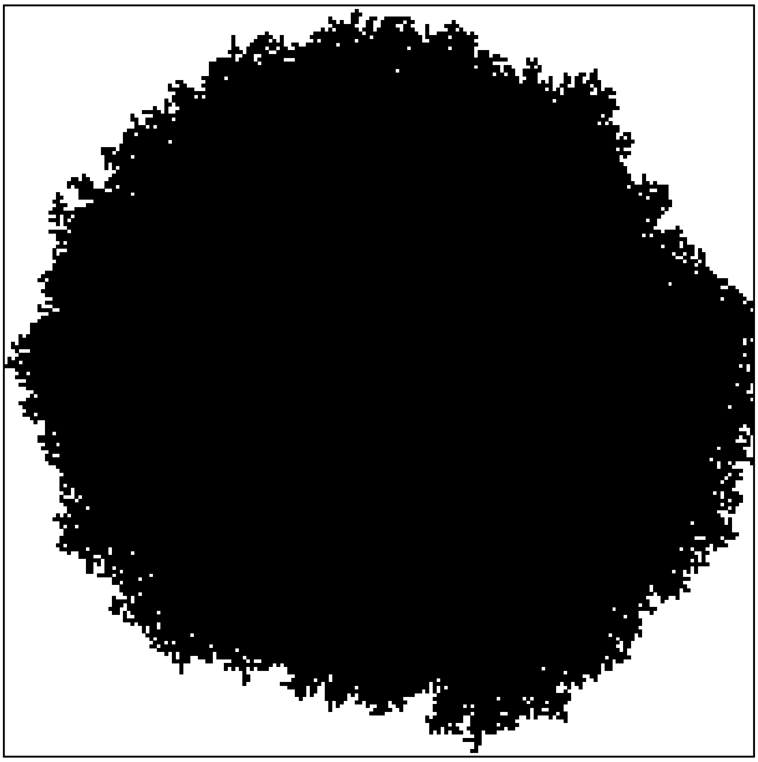

図 1　格子サイズ $L = 201$ のとき，生成されたイーデンクラスター

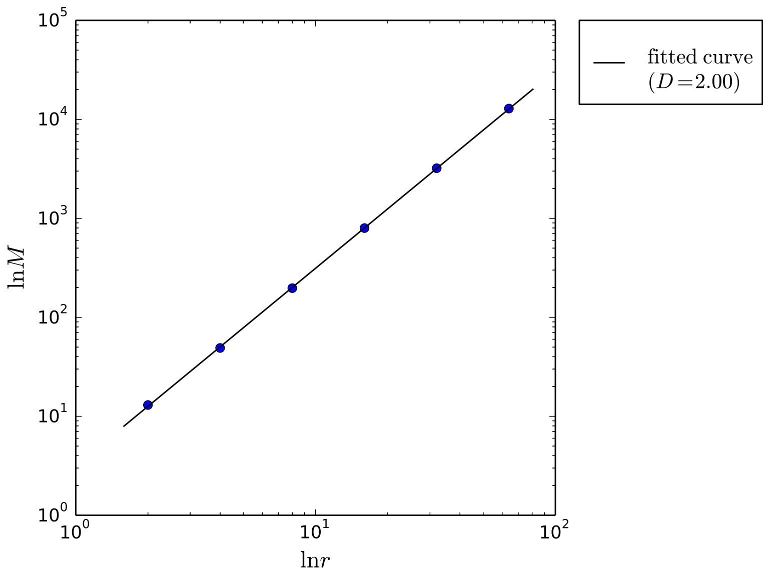

図 2　種の格子点からの距離 r とその内部の占有された格子点数 $M(r)$ の関係 ($L = 201$)

b. 周辺の点つまり成長点だけが示されるようにプログラムを修正せよ．大部分の周辺の点はクラスターの中心から見てどこにあるか．計算時間と忍耐の許す限り大きなクラスターを成長させせよ．

$L = 257$ としたとき，もとのイーデンクラスターと，その成長点のみを表示したものとを，図 3，4

に示す．この図から，成長点の大部分はクラスターの外縁にあることが分かる．

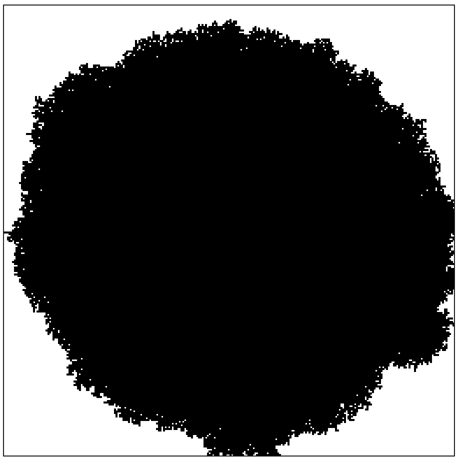

図 3　格子サイズ $L = 257$ のとき，生成されたイーデンクラスター

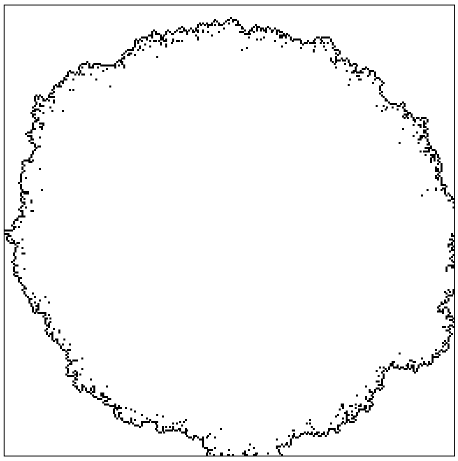

図 4　格子サイズ $L = 257$ のとき，イーデンクラスターの成長点

また，$L = 513$ としたときのクラスターの成長点を描画したものを図 5 に示した．

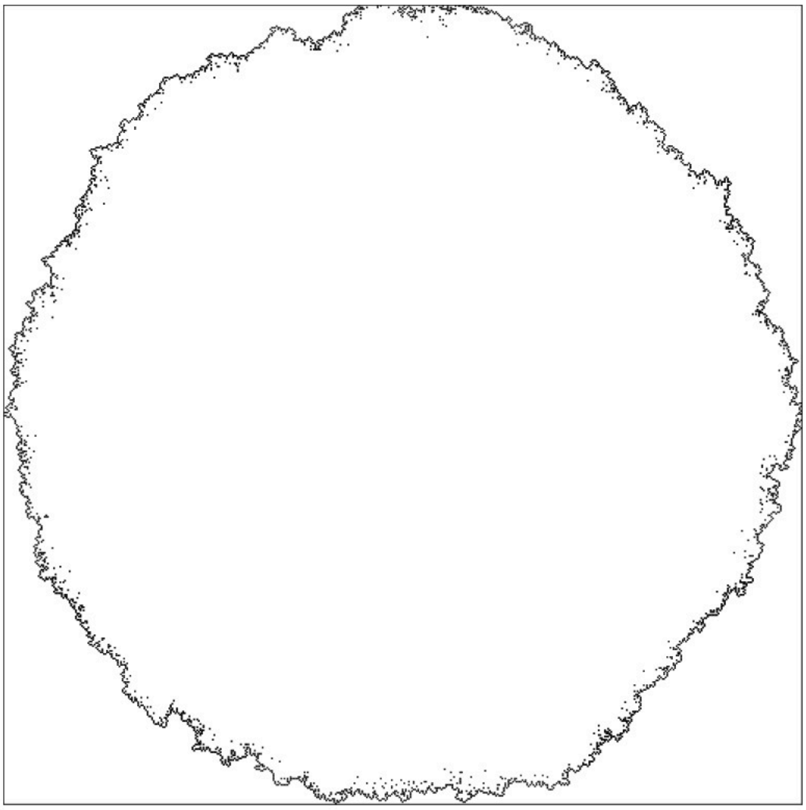

図 5　格子サイズ $L = 513$ としたとき，イーデンクラスターの成長点

4　まとめ

クラスターの生成方法としてよく知られたイーデンモデルについて学び，またその生成されたクラスターはフラクタル図形ではなく，コンパクトであることを確認できた．

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. 『計算機物理学入門』. ピアソン・エデュケーション, 2000.