計算機実習 問題 14.12 成長する表面

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

表面科学における問題の 1 つは粗い表面の形成を理解することにある．$t = 0$ で平らな表面があったとする．蒸着や沈着の結果として表面がどのように成長するかを考えてみよう．たとえば，初め直線上に並んだ L 個の占有された格子点があるとする．成長は垂直な方向に制限されている (図 14.13 参照)．以前と同様に，周辺の点をランダムに選んでそれを占有する．クラスターの平均の高さは

$$\bar{h} = \frac{1}{N_s}\sum_{i=1}^{N_s} h_i \tag{1}$$

で与えられる．ここで h_i は基線から i 番目の表面の点までの距離である．和はすべての表面の点 N_s 個についてとられる (イーデン・モデルにおける表面の点の正確な定義は問題 14.12 で議論されている)．

粒子 1 個が付着するたびに t を 1 だけ増加させる．ここでの主な興味は表面の”幅”が t とともにどのように変化するかにある．表面の幅を

$$\omega^2 = \frac{1}{N_s}\sum_{i=1}^{N_s}(h_i - \bar{h})^2 \tag{2}$$

で定義する．一般に，表面の幅 ω は L と t に依存し，表面の粗さの尺度を与える．初め ω は時間とともに大きくなり，

$$\omega(L, t) \sim t^{\beta} \tag{3}$$

であると予想される．指数 β は垂直方向に沿った成長の時間相関を記述する．図 14.13 はイーデン・モデルによる表面の発展を示している．ある特徴的な時間の後のゆらぎが相関している長さが L と同じ程度になり，幅は L のみに依存する定常な値に達する．つまり，

$$\omega(L, t \gg 1) \sim L^{\alpha} \tag{4}$$

である．α は粗さ指数として知られている．

式 (4) からは，定常状態では基線に垂直方向の表面の幅は L^{α} で成長することが分かる．このような幅についての定常状態の振る舞いは自己アフィンフラクタルの特徴の 1 つである．そのようなフラクタルは異方的なスケール，すなわち，異なる方向で異なる長さのスケールをもつ場合の変換のもとで (平均として) 不変である．例として，表面を水平方向に因子 b でスケールし直すとしよう．このとき，もとの表面とスケールされた表面とが相似性を保つためには表面の垂直方向を因子 b^{α} でスケールし直さなければならない．

短い長さのスケール，つまり，界面の幅よりも短い長さでは，表面は荒れていてその粗さは指数 α で特徴づけられる (表面を歩く蟻を想像せよ)．しかし，表面の幅よりもずっと長いスケールでは，表面は平らに見えるようになり，ここの例では，一次元的になる．問題 14.12 では，いくつかの成長モデルで与えられる表面の特性が調べられる．

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・モデルに基づき界面の成長を記述するプログラム

このプログラムは，3 つのクラスからなる．今回のシミュレーションのコアであるクラス Eden と，パラメータ等の設定ダイアログの表示を行うクラス TopWindow，そのダイアログでボタンを押した際に実行される関数群をまとめたものであるクラス Main である．プログラムを実行すると，まず Main が呼び出され，その初期化関数__init__の中で TopWindow が呼び出され，ダイアログが表示される．ボタンを押すと，それに対応する Main 内の関数が実行され，結果を得ることができる．Eden において，ウィンドウの縦の長さを決めるために，問題 c で確認されたスケーリングの関係と，イーデン・モデルのクラスターのフラクタル次元がおよそ 2 であることを用いている (29 行目)．また，glow_lattice の中では時間の経過ごとに最大の高さを検出して，随時格子の大きさを変化させることによって，どれだけ長い時間シミュレーションを行っても良いようにしてある（初めから大きなサイズの配列をつくろうとするとエラーとなる）．基本的な成長規則はイーデン・モデルのシミュレーションで過去に行ったものと同じである．また，描画に関しては，内容が更新された格子のみを描画更新するようにしてある．しかし，このように描画を行うと，最終的に得られた図が，保存した時に汚くなることがあったので，最後のステップの終了後，一旦全てのキャンバス内のオブジェクトを削除して初期化してから，もう一度全体を描画するようにした．アニメーションや表面サイトの強調表示はオプションで変更することができる．問題 b，c の実行は長時間の試行を繰り返すものであるので，時間がかかることを注意しておく．

```
1  #! /usr/bin/env python
2  # -*- coding:utf-8 -*-
3  #
4  # written by Shotaro Fujimoto, August 2014.
5  #
6
7  from Tkinter import *
8  import numpy as np
9  import matplotlib.pyplot as plt
10 import sys
11 import random
12 import time
13
14
```

```
class Eden:

    def __init__(self, L=32, T=1000, view=True, animation=True, surface=True):
        self.sub = None
        self.lattice = None
        self.time_delay = 0
        self.L = L  # lattice size
        self.T = T
        self.view = view
        self.animation = animation
        self.surface = surface

        if self.view:
            default_size = 630  # default size of canvas
            L = self.L
            Ly = int(self.T / L + 1.6 * self.T ** (1. / 3))
            self.rsize = int(default_size / (2 * max(L, Ly)))
            if self.rsize == 0:
                self.rsize = 1
            self.fig_size_x = 2 * self.rsize * L
            self.fig_size_y = 2 * self.rsize * Ly
            self.margin = 10
            sub = Toplevel()

            self.canvas = Canvas(sub, width=self.fig_size_x + 2 * self.margin,
                                 height=self.fig_size_y + 2 * self.margin)
            self.c = self.canvas.create_rectangle
            self.update = self.canvas.update
            if self.animation:
                self.c(self.margin - 1, self.margin,
                       self.fig_size_x + self.margin,
                       self.fig_size_y + self.margin,
                       outline='black', fill='white')

            self.canvas.pack()

    def grow_lattice(self):
        self.lattice = np.zeros([self.L, 3], dtype=int)
        self.lattice[:, 0] = 1
        self.lattice[:, 1] = -1
```

```
        if self.sub is None or not self.sub.winfo_exists():
            lattice = self.lattice
            L = self.L
            choice = random.choice
            ne = [(0, -1), (0, 1), (-1, 0), (1, 0)]
            nnsite = set([(x, 1) for x in range(L)])
            if self.view and self.animation:
                site = [(x, 0) for x in range(L)]
                self.update_canvas(site)
                self.update_canvas(list(nnsite), color='cyan')
                self.update()
            t = []  # time
            S = []  # a number of growing sites
            N = []  # a number of occupied sites
            h = np.array([1] * self.L)
            omega = []
            hmax = 0
            _t = 0
            while _t < self.T:
                t.append(_t)
                S.append(len(nnsite))
                N.append(np.sum(lattice == 1))
                omega.append(np.std(h))
                nn = choice(list(nnsite))
                nnsite.remove(nn)
                lattice[nn] = 1
                i, j = nn
                if j + 1 > hmax:
                    hmax = j + 1
                    lattice = np.append(lattice, np.zeros([L, 1]), axis=1)
                newnn = set([((i + nx) % L, j + ny) for nx, ny in ne
                             if lattice[(i + nx) % L, j + ny] == 0])
                ss = newnn - nnsite
                nnsite = nnsite | newnn
                for m in list(ss):
                    lattice[m] = -1
                    if m[1] > h[m[0]]:
                            h[m[0]] = m[1]
                if self.view and self.animation:
                    self.update_canvas([nn])
```

```
                    self.update_canvas(list(ss), color='cyan')
                    self.update()
                _t += 1
            else:
                if self.view:
                    self.canvas.delete("all")
                    occupied = np.where(lattice == 1)
                    occupied = [(m, n)
                                for m, n in zip(occupied[0], occupied[1])]
                    neighber = np.where(lattice == -1)
                    neighber = [(m, n)
                                for m, n in zip(neighber[0], neighber[1])]
                    self.c(self.margin - 1, self.margin,
                           self.fig_size_x + self.margin,
                           self.fig_size_y + self.margin,
                           outline='black', fill='white')
                    self.update_canvas(occupied, color='black')
                    self.update_canvas(neighber, color='cyan')
                    if self.surface:
                        surface = [(x, h[x]) for x in range(L)]
                        self.update_canvas(surface, color='red')
                    self.update()

                    print "done: L = %d, T = %d" % (self.L, self.T)
            self.lattice = lattice

        return t, S, N, omega

    def update_canvas(self, site, color='black'):
        for m, n in site:
            self.c(2 * m * self.rsize + self.margin,
                   self.fig_size_y + self.margin -
                   2 * (n + 1) * self.rsize,
                   2 * (m + 1) * self.rsize + self.margin - 1,
                   self.fig_size_y + self.margin - 2 * n * self.rsize - 1,
                   outline=color, fill=color)
        if self.time_delay != 0:
            time.sleep(self.time_delay)
```

```
class TopWindow:

    def show_setting_window(self, title='', parameters=None, modes=[],
                            buttons=[]):
        self.root = Tk()
        self.root.title(title)

        frame1 = Frame(self.root, padx=5, pady=5)
        frame1.pack(side='top')
        self.entry = []
        for i, parameter in enumerate(parameters):
            label = Label(frame1, text=parameter.items()[0][0] + ' = ')
            label.grid(row=i, column=0, sticky=E)
            self.entry.append(Entry(frame1, width=10))
            self.entry[i].grid(row=i, column=1)
            self.entry[i].delete(0, END)
            self.entry[i].insert(0, parameter.items()[0][1])
        self.entry[0].focus_set()

        self.v = []
        for text, default in modes:
            self.v.append(BooleanVar())
            self.v[-1].set(default)
            self.b = Checkbutton(self.root, text=text, variable=self.v[-1])
            self.b.pack(anchor=W)

        for args in buttons:
            frame = Frame(self.root, padx=5, pady=5)
            frame.pack(side='left')
            for arg in args:
                b = Button(frame, text=arg[0], command=arg[1])
                b.pack(expand=YES, fill='x')

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

        self.root.mainloop()

    def quit(self):
```

```
        self.root.destroy()
        sys.exit()


class Main(object):

    def __init__(self):
        global top
        self.eden = None
        top = TopWindow()
        title = "Growing Surface"
        parameters = [{'L': 100}, {'T': 1000}, {'time delay': 0}]
        checkbuttons = [('animation', True), ('show surface', True)]
        buttons = [(('a: run', self.pushed), ('save', self.pr)),
                   (('b: beta', self.exp_b_beta),
                    ('b: alpha', self.exp_b_alpha),
                       ('fit', self.fitting)),
                   (('c: graph', self.exp_c),)]
        top.show_setting_window(title, parameters, checkbuttons, buttons)

    def pushed(self):
        L = int(top.entry[0].get())
        T = int(top.entry[1].get())
        self.eden = Eden(L, T)
        self.eden.animation = top.v[0].get()
        self.eden.surface = top.v[1].get()
        self.eden.time_delay = float(top.entry[2].get())
        self.eden.grow_lattice()

    def exp_b_beta(self):
        T = 10000  # 100000
        self.Llist = [32, 64, 128]
        self.t = [2 ** i for i in range(int(np.log2(T) + 1)) if 2 ** i <= T]
        self.exp_b(r'$t$', r'$\omega(t)$', 15, T, target='beta')

    def exp_b_alpha(self):
        T = 200000
        self.Llist = [2 ** i for i in range(1, 11)]
        self.t = [i for i in xrange(T)]
        self.exp_b(r'$L$', r'$\omega(L)$', 15, T, target='alpha')
```

```

    def exp_b(self, xlabel, ylabel, trials, T, target=''):
        if target == 'beta':
            self.target = 'beta'
        elif target == 'alpha':
            self.target = 'alpha'
        elif target == 'c':
            self.target = 'c'
            self.data = []
        else:
            return

        fig = plt.figure("Growing Surface")
        self.ax = fig.add_subplot(111)
        self.ax.set_xscale('log')
        self.ax.set_yscale('log')
        self.ax.set_xlabel(xlabel, fontsize=16)
        self.ax.set_ylabel(ylabel, fontsize=16)
        self.ax.set_ymargin(0.05)
        self.omega = []
        for L in self.Llist:
            np_omega = np.array([])
            for trial in range(trials):
                eden = Eden(L, view=False)
                eden.T = T
                _t, S, N, omega = eden.grow_lattice()
                if self.target == 'alpha':
                    np_omega = np.append(np_omega, omega[-1])
                else:
                    omega = [omega[o] for o in self.t]
                    np_omega = np.append(np_omega, omega)
                    np_omega = np_omega.reshape(trial + 1, len(self.t))

            if self.target == 'alpha':
                self.omega.append(np.average(np_omega))
            else:
                self.omega = np.sum(np_omega, axis=0) / trials
                if self.target == 'c':
                    data = [(t / L ** 1.5, o / L ** (1. / 3))
                            for t, o in zip(self.t, self.omega)]
```

```
                    data.sort()
                    x, y = [], []
                    for d in data:
                        x.append(d[0])
                        y.append(d[1])
                    self.ax.plot(x, y, '-o', label='L = %d' % L)
                else:
                    self.ax.plot(self.t, self.omega, '-o', label='L = %d' % L)
        if self.target == 'alpha':
            self.ax.plot(self.Llist, self.omega, '-o')
        else:
            plt.legend(loc='best')

        fig.tight_layout()
        plt.show()

    def fitting(self):
        if self.target is None:
            return
        import scipy.optimize as optimize

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

        def fitted(x, c1, expo):
            return np.exp(c1) * (x ** expo)

        cut_from = int(raw_input("from ? (index) >>> "))
        cut_to = int(raw_input("to ? (index) >>> "))
        if self.target == 'beta':
            cut_x = np.array(self.t[cut_from:cut_to])
        if self.target == 'alpha':
            cut_x = np.array(self.Llist[cut_from:cut_to])
        cut_omega = np.array(self.omega[cut_from:cut_to])
        parameter0 = [0.1, 0.5]
        result = optimize.leastsq(
```

```
                fit_func, parameter0, args=(cut_x, cut_omega))
        c1 = result[0][0]
        expo = result[0][1]

        if self.target == ’beta’:
            label = r’fit func: $\beta$ = %f’ % expo
        if self.target == ’alpha’:
            label = r’fit func: $\alpha$ = %f’ % expo

        self.ax.plot(cut_x, fitted(cut_x, c1, expo), lw=2, label=label)
        plt.legend(loc=’best’)
        plt.show()

    def exp_c(self):
        T = 20000  # 100000
        self.Llist = [2 ** i for i in range(5, 15)]
        self.t = [2 ** i for i in range(int(np.log2(T) + 1)) if 2 ** i <= T]
        self.exp_b(r’$t/L^{\alpha/\beta}$’, r’$\omega(L,t)/L^{\alpha}$’,
                   15, T, target=’c’)

    def pr(self):
        import tkFileDialog
        import os

        if self.eden is None:
            print "first, you should run ’run’."
            return

        fTyp = [(’eps flle’, ’*.eps’), (’all files’, ’*’)]
        filename = tkFileDialog.asksaveasfilename(filetypes=fTyp,
                                                  initialdir=os.getcwd(),
                                                  initialfile="figure_1.eps")
        if filename is None:
            return
        try:
            self.eden.canvas.postscript(file=filename)
        except TclError:
            print """
            TclError: Cannot save the figure.
            Canvas Window must be alive for save."""
```

```
            return 1
if __name__ == ’__main__’:

    app = Main()
```

3 実習課題

a. イーデン・モデル．イーデン・モデルでは，周辺の点がランダムに選ばれ占有される．このモデルでは，”オーバーハング”が存在し得る．また，高さ h_x は基線から列 x における周辺の点までの距離の中で最大のものに対応する．水平方向に周期的境界条件を用いてすべての周辺の点を定めよ．成長の規則は通常のイーデン・モデルと同様であるが，成長は長さ L の帯の上端から始まる．$L = 100$ の正方格子を調べ，表面の成長とともに表面のようすがどのように変化していくか述べよ．表面を明確に定めることができるか．周辺の点の多くはどこにあるか．周辺の点の部分集合として表面の点が定義されている (すなわち，ある x に対して最大の h を持つ点)．もし全ての周辺の点を含めたら，結果は定性的に異なると考えるか.

$L = 100$ の正方格子で，表面の成長とともに表面の様子がどのように変化していくのかを観察した．実際に得られた図を図 1 に示した．このとき，黒で示した部分は占有された格子点を表し，水色で示した部分は周辺の点，赤色で示した部分は周辺の点のうち，ある x に対して最大の h を持つ点，すなわち表面の点を表している．

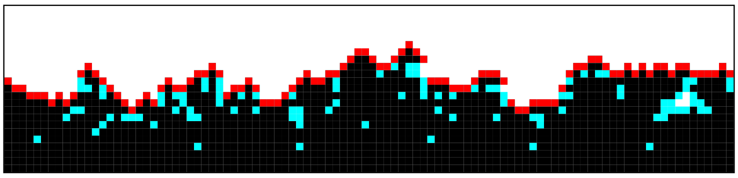

図 1　$L = 100$ のとき，成長させて得られた表面の様子

図 1 から，周辺の点の多くは表面の近くに存在していることが分かる．また，成長に応じて，成長したクラスターの表面の幅 (赤色の部分の分布の偏差) は大きくなっていくことが観察される．具体的にどのように変化しているかについては，問題 b で確かめることとする．

b. 同じグラフ上に，$L = 32, 64, 128$ について幅 $\omega(t)$ を時間の関数としてプロットし，イーデン・モデルの指数 α と β の値を求めよ．どのようにプロットするのが最も適当か．幅は初めべき乗則にしたがって成長するか．もしそうであるなら指数 β を求めよ．その時間の後に表面の幅が定常状態の値となるような，L に依存するクロスオーバ時間はあるか．どのようにして α の値を得ることができるか．数値的に得られている β と α の最も良い値は，それぞれに対して予言されている正確な値 $\beta = 1/3$，$\alpha = 1/2$ と一致している．

同じグラフ上に $L = 32, 64, 128$ について幅 $\omega(t)$ を時間の関数として，両対数グラフにしてプロットした (図 2)．このとき，これらの値は 15 回の試行の平均をとったものとなっている．

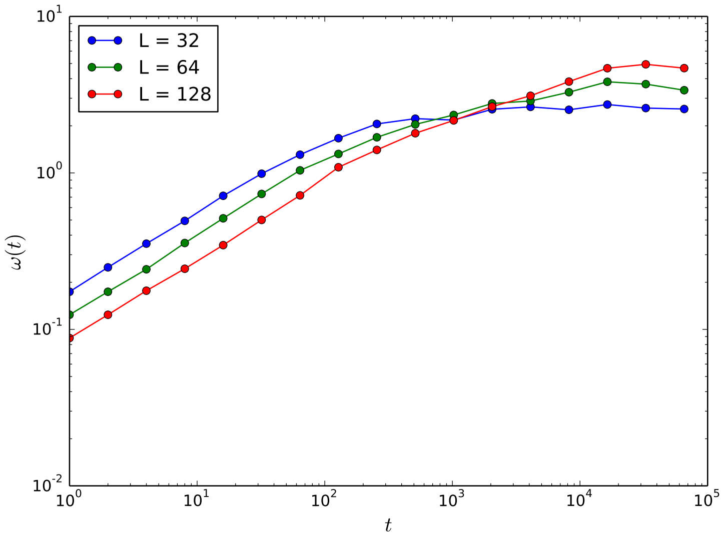

図 2　$L = 32, 64, 128$ について幅 $\omega(t)$ を時間 t の関数として両対数グラフにしたもの ($T \sim 100000$).

このようにすると，幅 ω は初め両対数グラフ上で直線に乗るので，ベキ乗則に従うことがわかる．このとき，

$$\omega(L, t) \sim t^{\beta} \tag{5}$$

として $L = 128$ の場合について β を求めると，$\beta = 0.508658$ となる．また，他の L に関しても β の値は同じであることが確かめられる．しかし，これは理論的に予測されている値 $\beta = 1/3$ とは異なっている．

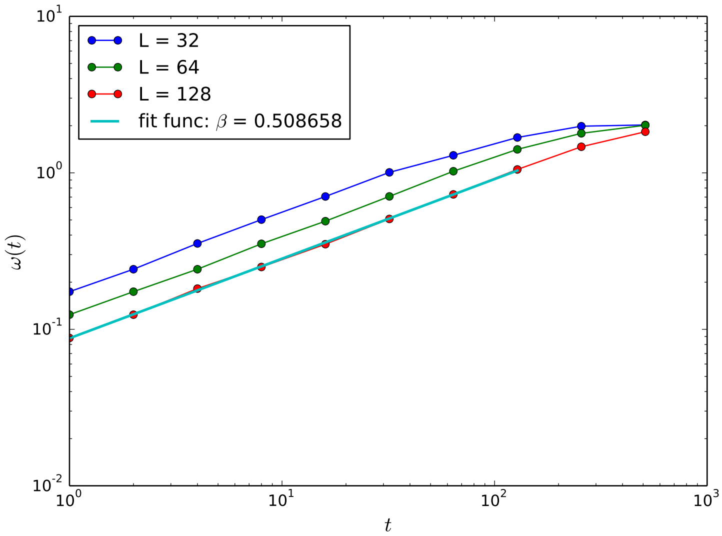

図 3　$L = 32, 64, 128$ について幅 $\omega(t)$ を時間 t の関数として両対数グラフにしたもの ($T \sim 1000$)．直線の傾きが β を表す．

また，t の十分大きいところでは，それぞれの L に対してグラフは水平になり，それ以上変化しないようになる．このような状況になったときの L に対する ω の値を比較すれば，

$$\omega(L, t \gg 1) \sim L^{\alpha} \tag{6}$$

の式から α を求めることができる．
図 2 から，L が大きいほど，表面の幅が定常状態の値となるクロスオーバー時間は長くなるので，計算量を少なくしながらも，できるだけ正確な α の値を得るためには，100 程度までの L について，それぞれの ω の定常状態の値を測定し，それらを一つのグラフに横軸 L，縦軸 ω の両対数グラフにして，その傾きを求めれば良い．このようにして得られた L に対する ω のグラフを図 4 に示す．このグラフは直線で近似することができ，その傾き α は図に示す通りであった．ただし，この値も $\alpha = 1/2$ とは異なっているように思われる．

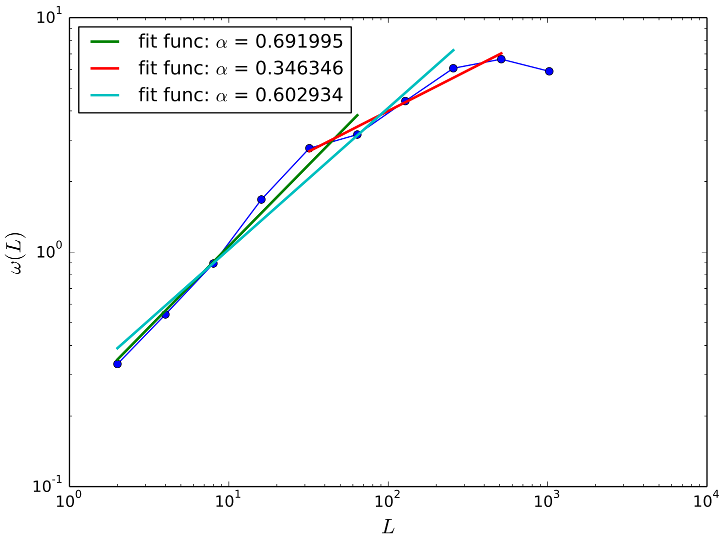

図 4　$L = 32, 64, 128$ について幅 $\omega(t)$ を時間 t の関数として両対数グラフにしたもの ($T \sim 1000$)．直線の傾きが β を表す

c. $\omega(L, t)$ および L 依存性は，スケーリング仮設

$$\omega(\varepsilon^a L, \varepsilon^b t) = \varepsilon^c \omega(L, t) \tag{7}$$

をたてることにより，$\varepsilon^a L = 1$ とおくと

$$\varepsilon = L^{-1/a} \tag{8}$$

であり，これを代入すると，

$$\omega(1, L^{-b/a} t) \equiv f(t/L^{b/a}) = L^{-c/a} \omega(L, t) \tag{9}$$

のようにスケーリング関数 f を決定することができ，$\alpha = c/a, \beta = c/b$ とおくと，

$$\omega(L, t) \approx L^{\alpha} f(t/L^{\alpha/\beta}) \tag{10}$$

で表すことができる．ここで，

$$f(x) \approx x^{\beta}, \quad x \ll 1 \text{ の場合} \tag{11}$$
$$f(x) = \text{一定}, \quad x \gg 1 \text{ の場合} \tag{12}$$

である．設問 b で考えられた L のいろいろな値について，比 $\omega(L,t)/L^{\alpha}$ を $t/L^{\alpha/\beta}$ に対してプロットすることにより，スケーリングの形 (10) の存在を確認せよ．このスケーリング式が成り立つならば，異なる L の値に対する ω の結果は普遍的な曲線上にのる．設問 b で得られた α，β の値，または正確な値を用いよ．

以下の図 5 に示すように，様々な L に対して，時間 t と表面の幅 $\omega(L,t)$ の関係を，横軸を $t/L^{\alpha/\beta}$，縦軸を $\omega(L,t)/L^{\alpha}$ として両対数グラフにすると，1 つの曲線にまとめられることが見て取れる．

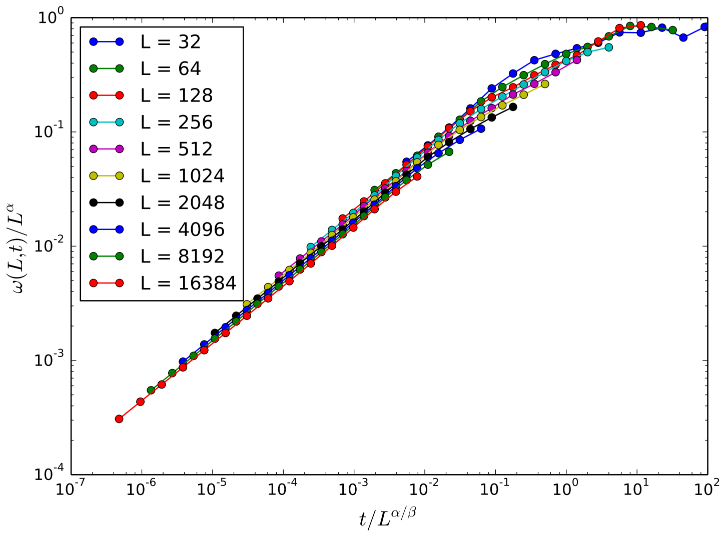

図 5　様々な L に対し，横軸を $t/L^{\alpha/\beta}$，縦軸を $\omega(L,t)/L^{\alpha}$ として両対数グラフにた．スケーリングの関係 (10) の存在を確認できる．

4　まとめ

成長する表面を，イーデンモデルによるシミュレーションで考えることができ，そのときの表面の幅 ω の性質について考えることができた．指数の値が期待している値とならないことが不思議ではあるが，現段階で間違いを見つけることができなかった．

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.

[2] 松下貢. フラクタルの物理 (I). 裳華房, 第 4 版, 2009.