0　レポート課題

1. DLA のプログラムを作成し、得られた DLA パターンのフラクタル次元を求めよ。
2. 1 次元ブラウン曲線でできた時系列を PC で作成し、得られた時系列のハースト指数を求めよ。

1　DLA パターンのフラクタル次元

DLA(Diffusion Limited Aggregation:拡散律速凝集) によって作られるパターンのフラクタル次元を，回転半径法によって求める。

DLA パターンの作成について，まず，正方格子状の格子点を考え，その原点を 1 つの粒子が占有している状態をスタートとする。次に中心から距離 `R` の位置に一つの粒子を (ランダムに) 配置し，この粒子は単位時間ステップごとに上下左右の 4 方向に等確率で移動するようなランダムウォークを行うとする。

プログラムの実行時間を短縮するために，粒子が `R + 2` より遠い位置に存在するときにはランダムウォークの歩幅 `l` を大きくし，`l = r - (R + 2)`(`r` は DLA の中心から粒子までの距離) のようにする。また，粒子が中心からあまりにも遠く (`r` $>$ `2R`) 離れてしまった際には，その粒子を取り除き，再度中心から距離 `R` の円周上から新しい粒子を出発させる。

粒子が DLA クラスターのどれかと 4 辺のうちどこかで接触した際には，その時点でその粒子を DLA クラスターに取り込むことにする。また，初期出発円 (半径 `R`) の半径は，中心から最も遠い点までの距離 $r_{\max}$ として $R = r_{\max} + 2$ とすることにする。さらに，粒子が付着するごとに，回転半径 R_g を求め，記録しておく。回転半径は粒子の重さ 1，クラスターを形成する粒子数 N として

$$R_g = \sqrt{\frac{1}{N}\sum_{i=1}^{N} r_i^2}$$

のようにして求めることができる。

DLA パターンを作成するのには付録に示す Python スクリプト (`DLA.py`[3.3]) を使用した。

実際に粒子数 $N = 16384$ 個として作成した DLA パターンを図 1 に示す。ここで粒子の色の違いは，何番目に DLA パターンに取り込まれたかものかを表すものとなっている。

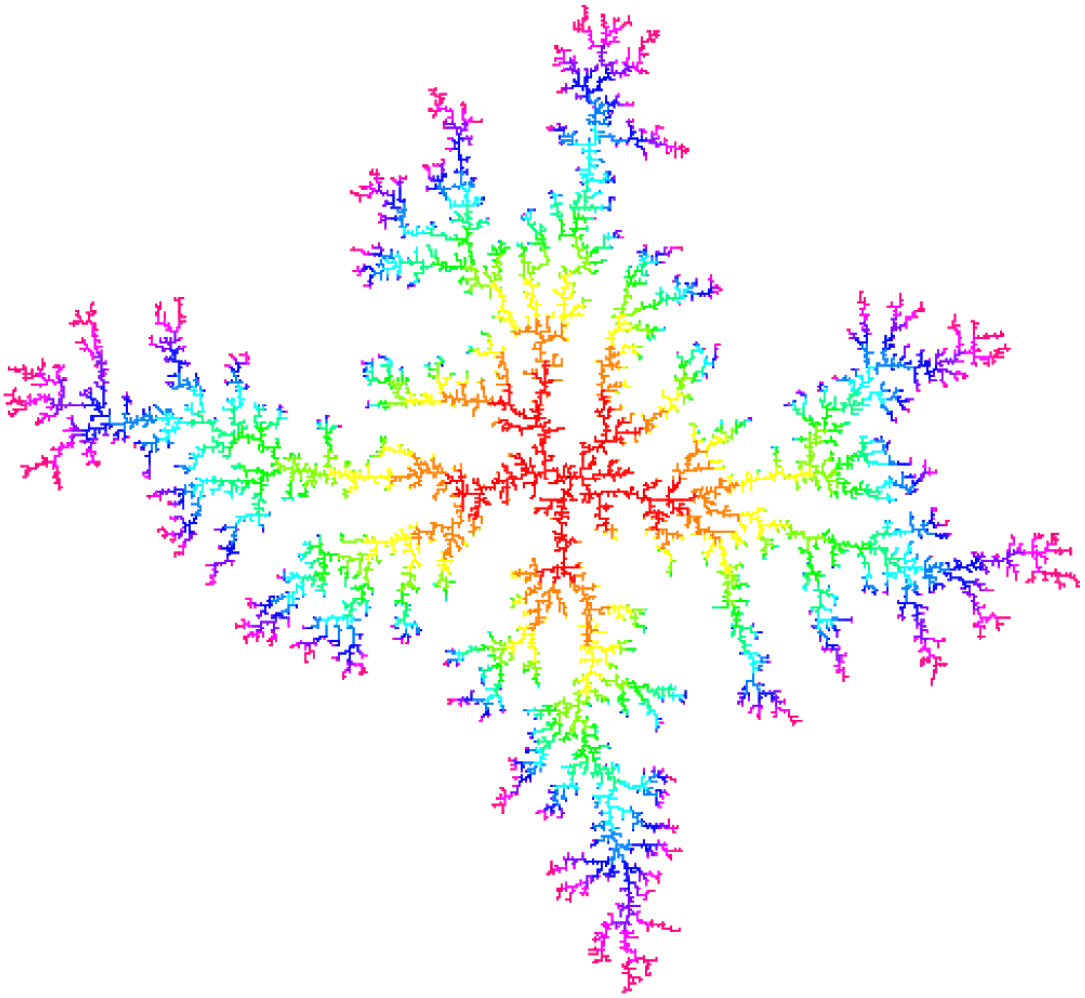

図 1　$N = 16384$ のときの DLA パターン

さて，図 1 からも DLA パターンは自己相似的に見えるが，回転半径 R_g と粒子数 N の間に以下のようなスケール則が成り立つということが知られている。

$$N \sim R_g^D \tag{1}$$

ここで D はフラクタル次元である。

したがって R_g と N を両対数グラフにプロットし，グラフ上で直線で近似すると，式 (1) からこの直線の傾きがフラクタル次元 D に対応することが分かる (使用したプログラム: `SetParameter.py`[3.1], `fitting.py`[3.2], `DLA.py`[3.3], `fractal_dimension_of_DLA.py`[3.4])。

実際に $N = 16384$ として作成した DLA パターンについて R_g と N を両対数グラフにプロットしたものを図 2 に示す。$N \geq 8$ の範囲でフィッティングを行ったものがグラフ中の緑色で表した線分であり，この傾きは約 1.73 と求めることができた。より大規模なシミュレーションによれば $D \simeq 1.71$ となることが知られて

いるが，今回のシミュレーションで得られたフラクタル次元と比較しても近い値であるので，本シミュレーションの結果もある程度の精度があると考えても良いだろう。

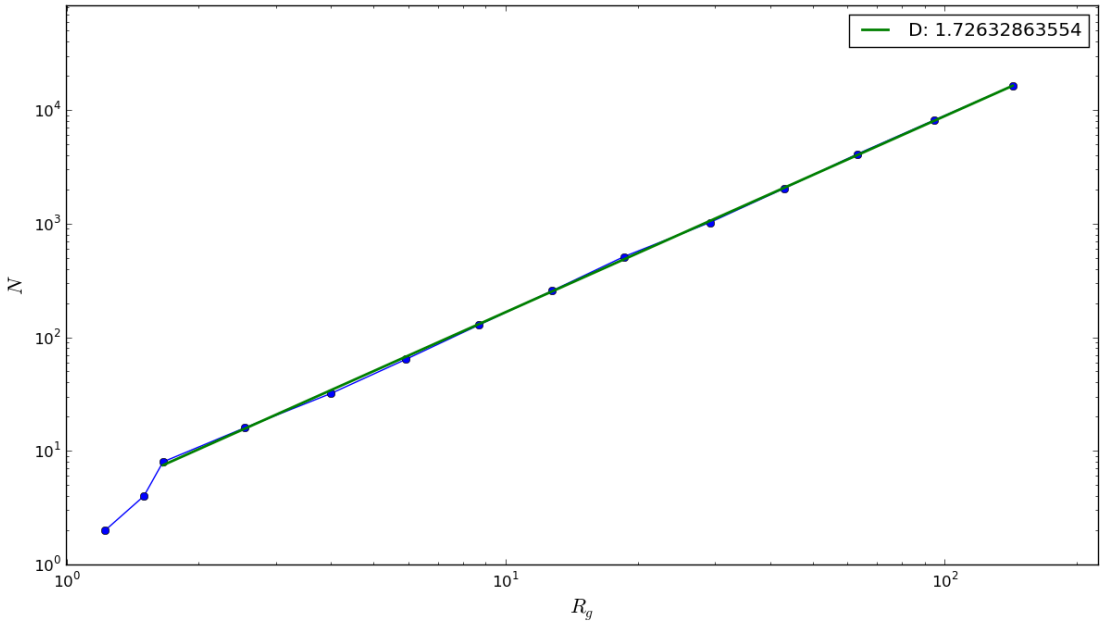

図 2　回転半径 R_g と N の間の関係

2　1 次元ブラウン曲線のハースト指数

1 次元ブラウン曲線 $x(t)$ は以下のように生成される時系列である:

$$x(t+1) = x(t) + \xi(t)$$

$$\xi(t) = \begin{cases} +1 & (\text{with probability} \quad \frac{1}{2}) \\ -1 & (\text{with probability} \quad \frac{1}{2}) \end{cases}$$

これをプログラム上で生成するために付録に示す `hurst.py`[3.5] 内の関数 `brownian_curve_1d` を作成した。この関数によって `N` ステップの時系列 `X` が作成される。

実際にプログラムによって作成した 1 次元ブラウン曲線を図 3 に示す。

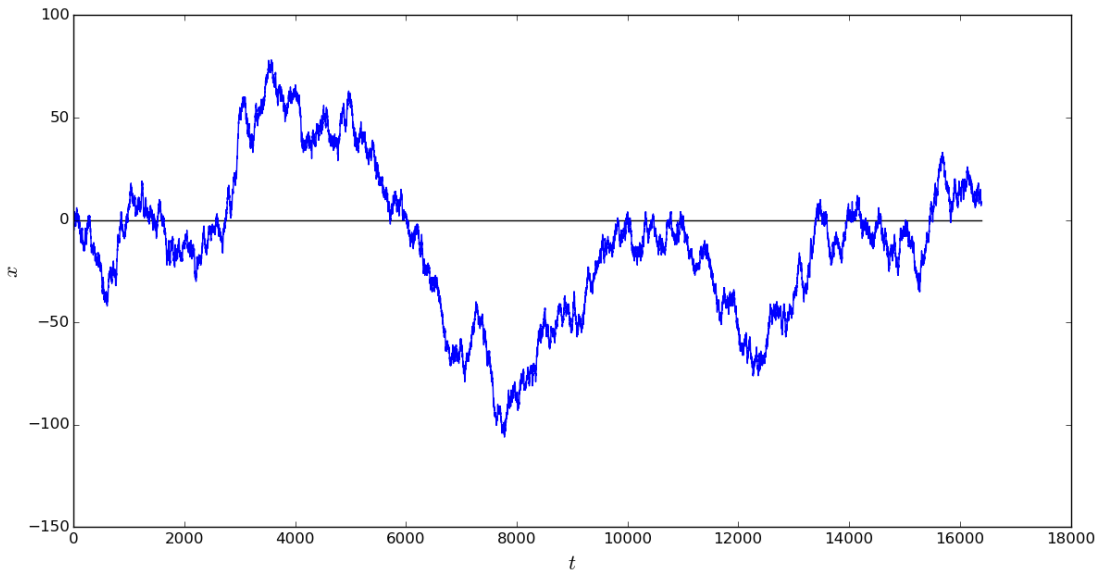

図 3　プログラムによって作成した 1 次元ブラウン曲線 ($T_{\max} = 16383$)

ハースト指数はある時系列があったときに，その時間スケールと偏差の間で

$$t \to \varepsilon; \quad \sigma \to (\varepsilon\sigma)^H$$

のように縦と横で異なるスケール変換を行うことで σ の値が統計的に元の値と等しくなるときのこの H のことを意味する。

また，同じ意味であるが統計を取る時間 T を決めてその範囲での偏差 $\sigma_x(T)$ を求め，これが

$$\sigma_x(T) \sim T^H$$

のような関係を満たすときの H をハースト指数と定義できる。

即ち，ある時系列について，統計を取る範囲 T を変化させ，その時々に計算して得られる偏差 $\sigma_x(T)$ を記録し，横軸 T，縦軸 σ_x として両対数グラフにプロットし，直線で近似した時の傾きが H として求められるということが分かる。

実際に時系列についてハースト指数を計算するプログラムを作成した [3.5]。関数 `calc_hurst` によって，異なる T それぞれについて時系列全体を覆うようにサンプル範囲をスライドさせ，その時々に得られた偏差 σ を平均することで，時系列のどの位置をサンプルとして選ぶかの依存性をなくした値を得ることができる。このとき T が大きくなると当然取りうるサンプルの個数も少なくなることに注意が必要である。

このようにして各 T について偏差 σ の平均値を求め，これを両対数グラフにプロットしたものが図 4 である。

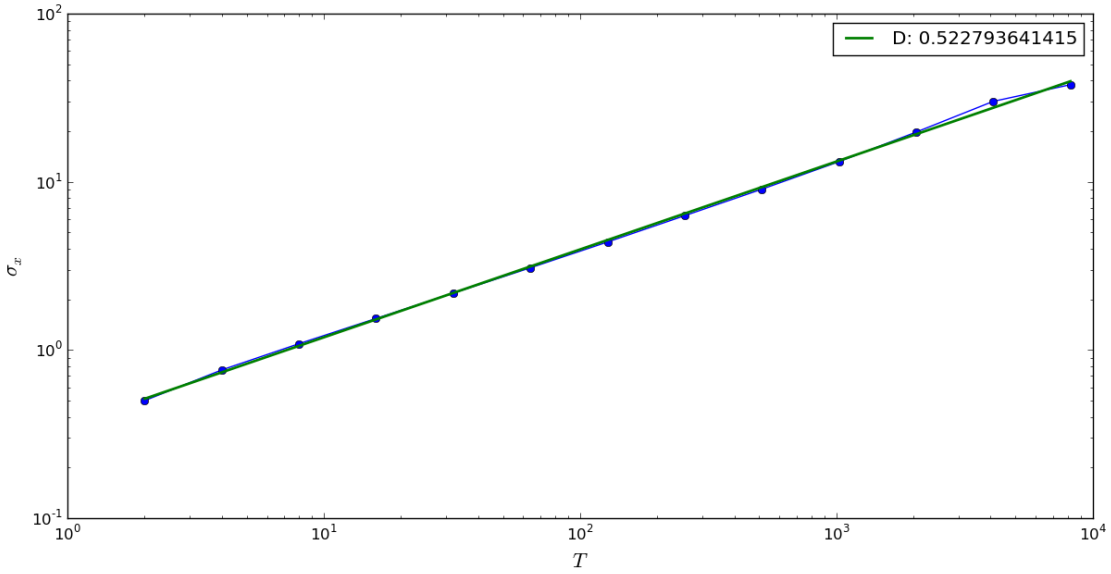

図 4　T と σ_x の間の関係

この図からも直線で近似しても良さそうなことがわかり，実際にこれをフィッティングすると，緑色の線分のようになり，またその傾きは約 0.523 となった。よく知られているように，1 次元ブラウン曲線について $H = 0.5$ であるので，これと比べても適切な値が得られていることが分かる。

3 付録

3.1 プログラムの実行用ダイアログを表示するスクリプト

SetParameter.py

```
1  #!/usr/bin/env python
2  # -*- coding:utf-8 -*-
3  #
4  # written by Shotaro Fujimoto
5
6  from Tkinter import Tk, Frame, Label, Button, Entry, E, END
7
8
9  class SetParameter():
10
11     def show_setting_window(self, parameters, commands):
12         """ Show a parameter setting window.
13
14         parameters: A list of dictionaries {'parameter name': default_value}
15         commands: A list of dictionary {'name of button': command}
16         """
17         self.root = Tk()
18         self.root.title('SetParameter')
19
20         frame1 = Frame(self.root, padx=5, pady=5)
21         frame1.pack(side='top')
22
23         self.entry = []
24         for i, parameter in enumerate(parameters):
25             label = Label(frame1, text=parameter.items()[0][0] + ' = ')
26             label.grid(row=i, column=0, sticky=E)
27             self.entry.append(Entry(frame1, width=10))
28             self.entry[i].grid(row=i, column=1)
29             self.entry[i].delete(0, END)
30             self.entry[i].insert(0, parameter.items()[0][1])
31         self.entry[0].focus_set()
32
33         frame2 = Frame(self.root, padx=5, pady=5)
34         frame2.pack(side='bottom')
```

```
35
36          self.button = []
37          for i, command in enumerate(commands):
38              self.button.append(Button(frame2, text=command.items()[0][0],
39                                        command=command.items()[0][1]))
40              self.button[i].grid(row=0, column=i)
41
42          self.root.mainloop()
43
44      def quit(self):
45          self.root.destroy()
```

3.2 フィッティングを行うためのラッパー関数

fitting.py

```
1   #!/usr/bin/env python
2   # -*- coding:utf-8 -*-
3   #
4   # written by Shotaro Fujimoto
5
6   import numpy as np
7   import scipy.optimize as optimize
8   import matplotlib.pyplot as plt
9
10
11  def fitting(X, Y, fit_func, parameters, fitted, xscale='log', yscale='log',
12              xlabel=r'$X$', ylabel=r'$Y$', param_to_show={'param0': 0}):
13      """Fitting method to calcurate the hurst exponent."""
14
15      # set the data set to be fitted by user input.
16      # User should specify the data length by its index(not the true value).
17      cut_from = int(raw_input("fit from ? (index) >>> "))
18      cut_to = int(raw_input("fit to ? (index) >>> "))
19      cut_X = np.array(X[cut_from:cut_to])
20      cut_Y = np.array(Y[cut_from:cut_to])
21
22      # fitting by least aquare method
23      # fit_func(parameters, X, Y)
24      result = optimize.leastsq(fit_func, parameters, args=(cut_X, cut_Y))
```

```
25      # it returns fitted parameters
26      fitted_params = result[0]
27      print "fitted parameters: "
28      print fitted_params
29
30      # Plot the result and fitting func with fitted parameters
31      fig = plt.figure("Fitting")
32      ax = fig.add_subplot(111)
33      ax.plot(X, Y, '-o')
34      ax.set_xlabel(xlabel, fontsize=16)
35      ax.set_ylabel(ylabel, fontsize=16)
36      ax.set_xscale(xscale)
37      ax.set_yscale(yscale)
38      ax.set_ymargin(0.05)
39      labels = '\n'.join([s + ': ' + str(fitted_params[i])
40                          for s, i in param_to_show.iteritems()])
41      ax.plot(cut_X, fitted(cut_X, *fitted_params), lw=2, label=labels)
42      plt.legend(loc='best')
43      fig.tight_layout()
44      plt.show()
```

3.3 DLA パターンを作成するスクリプト

DLA.py

```
1   #!/usr/bin/env python
2   # -*- coding:utf-8 -*-
3   #
4   # written by Shotaro Fujimoto, January 2016.
5
6   from Tkinter import Toplevel, Canvas
7   import numpy as np
8   import time
9
10
11  class DLA(object):
12
13      def __init__(self, N, view=True, color=True):
14          self.R = 3
15          self.sum_rxr = 0
```

```
16          # radius of gyration
17          self.R_g = []
18          self.N = N
19          self.view = view
20          self.color = color
21          self.L = int(self.N**(0.76)) + 2
22
23          if self.view:
24              self.default_size = 640  # default size of canvas
25              self.rsize = int(self.default_size/(2 * self.L)) or 1
26              fig_size = 2 * self.rsize * self.L
27              self.margin = 10
28              self.sub = Toplevel()
29              canvas_w = fig_size + 2 * self.margin
30              canvas_h = fig_size + 2 * self.margin
31              self.canvas = Canvas(self.sub, width=canvas_w, height=canvas_h)
32              self.c = self.canvas.create_rectangle
33              self.update = self.canvas.update
34
35              self.sub.title('DLA cluster')
36
37              self.c(
38                  self.margin,
39                  self.margin,
40                  fig_size + self.margin,
41                  fig_size + self.margin,
42                  outline='black',
43                  fill='white'
44              )
45
46              self.canvas.pack()
47              self.start_time = time.time()
48
49      def grow_cluster(self):
50          """Glow the DLA cluster by random walking particles.  """
51          rn = np.random.rand
52
53          # Set the lattice size
54          lattice = np.zeros([self.L * 2 + 1, self.L * 2 + 1], dtype=int)
55          # center of the lattice
```

```
56          self.center = self.L
57          # the center of lattice is occupied by a particle from start.
58          lattice[self.center, self.center] = 1
59
60          # visualization
61          if self.view:
62              self.c(
63                  (2 * self.center - self.L)*self.rsize + self.margin,
64                  (2 * self.center - self.L)*self.rsize + self.margin,
65                  (2 * (self.center + 1) - self.L)*self.rsize + self.margin - 1,
66                  (2 * (self.center + 1) - self.L)*self.rsize + self.margin - 1,
67                  outline='black',
68                  fill='black'
69              )
70              self.update()
71
72          def reset_particle_postion():
73              """Initialise the postion of the particle."""
74              theta = 2 * np.pi * rn()
75              x = int((self.R + 2) * np.cos(theta)) + self.center
76              y = int((self.R + 2) * np.sin(theta)) + self.center
77              return x, y
78
79          def diffusion(x, y):
80              """Set a partcle at outer circle and move it as random walk.
81              Then, if it contacts the existing cluster, the cluster grows.
82              """
83
84              def get_distance_from_center(x, y):
85                  """Get the distance from the center to the particle position"""
86                  return np.sqrt((x - self.center)**2 + (y - self.center)**2)
87
88              # increase the step size of RW when it is far from the center.
89              #     r: distance from the center to the particle
90              r = get_distance_from_center(x, y)
91
92              #     l: step size of the random walk of the particle
93              l = int(r - self.R - 2) if int(r - self.R - 2) > 0 else 1
94
95              # Random walk
```

```
96              p = rn() * 4
97              if p < 1:
98                  x += l
99              elif p < 2:
100                 x -= l
101             elif p < 3:
102                 y += l
103             else:
104                 y -= l
105
106             # if the particle is far from the center, reset the possition.
107             r = get_distance_from_center(x, y)
108             if r >= 2 * self.R:
109                 return 2
110
111             # if there is no occupied site near the particle, continue.
112             # if judge == 0:
113             if not (lattice[x-1, y] == 1 or lattice[x+1, y] == 1 or
114                     lattice[x, y-1] == 1 or lattice[x, y+1] == 1):
115                 return x, y
116
117             # else, the particle is occupied to the DLA cluster.
118             lattice[x, y] = 1
119
120             # visualise
121             if self.view:
122                 if self.color:
123                     colors = ['#ff0000', '#ff8000', '#ffff00', '#80ff00',
124                               '#00ff00', '#00ff80', '#00ffff', '#0080ff',
125                               '#0000ff', '#8000ff', '#ff00ff', '#ff0080']
126                     len_colors = 12
127                     n_samecolor = (self.N / len_colors) + 1
128                     color = colors[n / n_samecolor]
129                 else:
130                     color = "black"
131
132                 self.c(
133                     (2 * x - self.L) * self.rsize + self.margin,
134                     (2 * y - self.L) * self.rsize + self.margin,
135                     (2 * (x + 1) - self.L) * self.rsize + self.margin - 1,
```

```
                    (2 * (y + 1) - self.L) * self.rsize + self.margin - 1,
                    outline=color,
                    fill=color
                )
                self.update()

            # Update R
            self.R = int(r) + 1 if int(r) + 1 > self.R else self.R
            # Update sum_rxr
            self.sum_rxr += r*r
            # Update R_g
            self.R_g.append(np.sqrt(self.sum_rxr/(float(len(self.R_g))+1.)))
            # Finish the random walk of the particle
            return 0

        n = 0
        while n < self.N:
            x, y = reset_particle_postion()
            while True:
                res = diffusion(x, y)
                # 0: process successfully done
                # 2: restart process
                if res == 0:
                    # increment n
                    n += 1
                    break
                elif res == 2:
                    x, y = reset_particle_postion()
                else:
                    x, y = res
        else:
            if self.view:
                # Save the canvas image
                filename = "img/" + str(time.time()) + ".eps"
                self.canvas.postscript(file=filename)
                print "Save the figure to " + filename

                # Print the time
                self.end_time = time.time()
                t = self.end_time - self.start_time
```

```
176                 print "done; N = %d, time = " % self.N + str(t) + ' (s)'
177
178         self.lattice = lattice
179         return self.lattice
```

3.4 DLA のフラクタル次元を計算するための実行スクリプト

fractal_dimension_of_DLA.py

```
1  #!/usr/bin/env python
2  # -*- coding:utf-8 -*-
3  #
4  # written by Shotaro Fujimoto
5
6  import matplotlib.pyplot as plt
7  import numpy as np
8  from SetParameter import SetParameter
9  from DLA import DLA
10 from fitting import fitting
11
12
13 class Main(object):
14
15     def __init__(self):
16         import sys
17         self.sp = SetParameter()
18         self.N = None
19         self.dla = None
20
21         self.sp.show_setting_window(
22             [
23                 {'N': 200}
24             ],
25             [
26                 {'start': self.grow_cluster},
27                 {'plot graph': self.plot__N_R},
28                 {'calcurate D': self.fit_to_powerlow},
29                 {'save': self.save_to_file},
30                 {'quit': sys.exit}
31             ]
```

```
32          )
33
34      def grow_cluster(self):
35          """Create a DLA cluster with N particles by dla.grow_cluster method."""
36          self.N = int(self.sp.entry[0].get())
37          self.dla = DLA(self.N)
38          self.lattice = self.dla.grow_cluster()
39          self.center = self.dla.center
40
41      def plot__N_R(self):
42          """Plot a N-R_g graph to calcurate DLA cluster's fractal dimension."""
43          self.N = int(self.sp.entry[0].get())
44          self.dla = DLA(self.N)
45          self.lattice = self.dla.grow_cluster()
46          self.center = self.dla.center
47          self.Narr = np.array([2**x for x in range(1, int(np.log2(self.N))+1)])
48          self.R_g = np.array([self.dla.R_g[n-1] for n in self.Narr])
49
50          # plot
51          fig = plt.figure("Fractal Dimension")
52          self.ax = fig.add_subplot(111)
53          self.ax.plot(self.R_g, self.Narr, '-o')
54          self.ax.set_xlabel(r'$R_{g}$', fontsize=16)
55          self.ax.set_ylabel(r'$N$', fontsize=16)
56          self.ax.set_xscale('log')
57          self.ax.set_yscale('log')
58          self.ax.set_ymargin(0.05)
59          fig.tight_layout()
60          plt.show()
61
62      def fit_to_powerlow(self):
63          """Fitting method to calcurate the fractal dimension of DLA cluster."""
64
65          def fit_func(parameter0, R_g, Narr):
66              """Fitting function: Narr ~ R_{g}^{D}"""
67              log = np.log
68              c1 = parameter0[0]
69              c2 = parameter0[1]
70              residual = log(Narr) - c1 - c2*log(R_g)
71              return residual
```

```
72
73          def fitted(R, c1, D):
74              return np.exp(c1)*(R**D)
75
76          fitting(self.R_g, self.Narr,
77                  fit_func, [0.1, 1.7], fitted,
78                  xlabel=r'$R_{g}$', ylabel=r'$N$',
79                  param_to_show={'D': 1}
80                  )
81
82      def save_to_file(self):
83          """Save the figure of the DLA cluster with eps format."""
84          import tkFileDialog
85          import os
86
87          if self.dla is None:
88              print "No figure exists."
89              return
90
91          ftype = [('eps flle', '*.eps'), ('all files', '*')]
92          filename = tkFileDialog.asksaveasfilename(
93              filetypes=ftype,
94              initialdir=os.getcwd(),
95              initialfile="figure_1.eps"
96          )
97          if filename is None:
98              return
99          self.dla.canvas.postscript(file=filename)
100
101
102 if __name__ == '__main__':
103     Main()
```

3.5 1 次元ブラウン曲線のハースト指数を求めるための実行スクリプト

hurst.py

```
1  #!/usr/bin/env python
2  # -*- coding:utf-8 -*-
3  #
```

```
4  # written by Shotaro Fujimoto
5
6  import numpy as np
7  import matplotlib.pyplot as plt
8  from fitting import fitting
9
10
11 def brownian_curve_1d(N, plot=True):
12     """Create basic N step brownian motion"""
13     p = 0.5
14     l = 1
15     x0 = 0
16
17     xi = np.random.random(N)
18     xi[xi > p] = l
19     xi[xi <= p] = -l
20
21     X = [x0, ]
22     for i in range(N):
23         X.append(X[i] + xi[i])
24     # X = [x0, x1, ... , xN]
25     # len(X) == N+1
26     X = np.array(X)
27
28     if plot:
29         # plot
30         fig = plt.figure("Brownian motion")
31         ax = fig.add_subplot(111)
32         ax.plot([0, len(X)-1], [x0, x0], '-', color='black')
33         ax.plot(range(len(X)), X, '-')
34         ax.set_xlabel(r'$t$', fontsize=16)
35         ax.set_ylabel(r'$x$', fontsize=16)
36         ax.set_xscale('linear')
37         ax.set_yscale('linear')
38         ax.set_ymargin(0.05)
39         fig.tight_layout()
40         plt.show()
41
42     return X
43
```

```
44
45  def calc_hurst(X, plot=True):
46      """Calucurate hurst exponent"""
47
48      def std(start, T):
49          return np.std(X[start:start+T])
50
51      # Tarr = [ 1, 2, 4, ... , 2^{int(log2(N))} ]
52      Tarr = np.array([2**x for x in range(1, int(np.log2(len(X)-1))+1)])
53      # sigma_T: list of the average of the deviation in a time-width T.
54      sigma_T = [np.average([std(i, T) for i in range(N+3-T)]) for T in Tarr]
55
56      if plot:
57          # plot
58          fig = plt.figure("Hurst exponent")
59          ax = fig.add_subplot(111)
60          ax.plot(sigma_T, Tarr, '-o')
61          ax.set_xlabel(r'$T$', fontsize=16)
62          ax.set_ylabel(r'$\sigma_{x}$', fontsize=16)
63          ax.set_xscale('log')
64          ax.set_yscale('log')
65          ax.set_ymargin(0.05)
66          fig.tight_layout()
67          plt.show()
68
69      return (Tarr, sigma_T)
70
71
72  def main(N):
73
74      def fit_func(parameter, Tarr, sigma_T):
75          """Fitting function: sigma ~ T^{H}"""
76          log = np.log
77          c1 = parameter[0]
78          c2 = parameter[1]
79          residual = log(sigma_T) - c1 - c2*log(Tarr)
80          return residual
81
82      def fitted(T, c1, H):
83          return np.exp(c1)*(T**H)
```

```
84
85      # Create brownian motion and calcurate deviations for each T
86      Tarr, sigma_T = calc_hurst(brownian_curve_1d(N, plot=True), plot=False)
87
88      # Fitting
89      fitting(Tarr, sigma_T,
90              fit_func, [0.1, 0.5], fitted,
91              xlabel=r'$T$', ylabel=r'$\sigma_{x}$',
92              param_to_show={'D': 1}
93              )
94
95
96  if __name__ == '__main__':
97      N = 16383  # = 16384(=2**14) - 1
98      main(N)
```