

# Fractal Diagram

2021 年 2 月 23 日

S.Matoi

# 目次

第 1 章	はじめに	2
第 2 章	マンデルブロ集合	3
第 3 章	ジュリア集合	10
第 4 章	再帰的处理による図形	14
4.1	コッホ曲線 . . . . .	14
4.1.1	コッホの雪片曲線 . . . . .	17
4.2	シェルピンスキーの三角形 . . . . .	19
4.3	バーンスレイのシダ . . . . .	21
第 5 章	おわりに	24
	謝辞	27
	参考文献	28

## 第 1 章

# はじめに

フラクタル図形は、図形の一部を拡大すると、再び同じ形状の図形が現れるという自己相似性と呼ばれる性質を持っている

## 第2章

# マンデルブロ集合

漸化式

$$\begin{cases} z_{n+1} = z_n^2 + c & (c \in \mathbb{Z}) \\ x_0 = 0 \end{cases}$$

において、 $n \rightarrow \infty$  で  $z_n$  が発散しないような複素数  $c$  の集合がマンデルブロ集合。

1980 年、フラクタルの名付け親であった、ベノワ・マンデルブロによって発見された。

まず、 $z = 0$  から計算を始めるので、 $z^2$  に  $c$  を加えた結果は  $c$  である

これを元の  $z$  に代入すると、 $c^2 + c$  という結果が得られる

これを再び元の  $z$  に代入し、 $(c^2 + c)^2 + c$  を計算する

このように、1 つ前の計算で得られた結果をもう一度  $z$  に代入して、順次計算を繰り返していく

定数  $c$  が特別の値だと、計算を何度繰り返しても、 $z^2 + c$  の大きさは、ある値を超えることはない

このような複素数  $c$  からなる集合をマンデルブロ集合という

複素平面上の  $c$  の位置毎に、繰り返し回数  $n$  をプロットしていけばマンデルブロ集合を作図できる

数値計算なので、厳密な収束や発散の判定はできない

つまり、コンピュータは  $\infty$  回まで反復を続けられないので、反復回数の上限  $maxItr$  を定めておく

そこまで繰り返しても発散の判定に至らなかった時は、 $maxItr$  (或いは 0) をその時の反復回数とする

発散の判定においても、適当に大きな数字  $M$  を定めておき、 $|z_n|$  が  $M$  を超えたら発散と判定していく

これをプログラムした部分は、次の通り

ソースコード 2.1 複素数で計算

```
1 def Calculate(self, c): # c: Complex number
2     # z0 = 0.0 + 1j*0.0
3     z = c # z = z0**2 + c
4     for n in range(self.maxItr):
5         az = abs(z)
6         if az > self.M:
7             return n
8         z = z*z + c
9     return self.maxItr
```

上記反復計算は複素平面上の値  $c$  毎に呼び出され、その  $c$  の時の反復回数  $n$  を発散の速さを表す値として配列に格納していき、計算終了後に、配列に納められた  $n$  に応じた色を  $c$  の位置に描画する

複素平面上のグリッドに対応する2次元配列に格納された計算結果  $n$  に小さな値が入っているということは、それだけ速く発散したことに相当し、逆に大きな値、特に上限値  $maxItr$  と等しい値（或いは0）が入っている場合は、実際には発散しなかったとみなせる

発散しなかったときの複素平面上の値  $c$  は、マンデルブロ集合に属する要素であるということ

Python は複素数型の変数を持っているので、上記のようなプログラムの記述が可能だが、そうでない場合は次の様にして、実数の領域での計算を行うことができる

ソースコード 2.2 実数で計算

```

1 def Calc(self, re, im): # re, im: Real number
2     x = 0.0
3     y = 0.0
4     for n in range(self.maxItr):
5         zx2 = x**2
6         zy2 = y**2
7         if zx2 + zy2 > self.M:
8             return n
9         x_ = zx2 - zy2 + re
10        y_ = 2*x*y + im
11        x, y = x_, y_
12    return self.maxItr

```

結果を格納した2次元配列を、ヒートマップとして matplotlib で描画する

ここでは numpy.meshgrid を使ってヒートマップを描いていく

ソースコード 2.3 マンデルブロ集合

```

1 import numpy as np
2 from numba import jit
3 from tqdm import tqdm
4 import matplotlib.pyplot as plt
5 from matplotlib import colors
6 import csv
7
8 class Mandelbrot:
9     def __init__(self, M, maxItr):
10         self.M = M
11         self.maxItr = maxItr
12
13     def Calc(self, re, im): # re, im: 2d Array
14         x = 0.0
15         y = 0.0
16         for n in range(self.maxItr):
17             zx2 = x**2
18             zy2 = y**2
19             if zx2 + zy2 > self.M:
20                 return n
21             x_ = zx2 - zy2 + re
22             y_ = 2*x*y + im
23             x, y = x_, y_
24         return self.maxItr
25
26     def Calculate(self, c): # c: complex
27         # z0 = 0.0 + 1j*0.0
28         z = c # z = z0**2 + c
29         for n in range(self.maxItr):
30             az = abs(z)
31             if az > self.M:
32                 return n
33             z = z*z + c
34         return self.maxItr

```

```

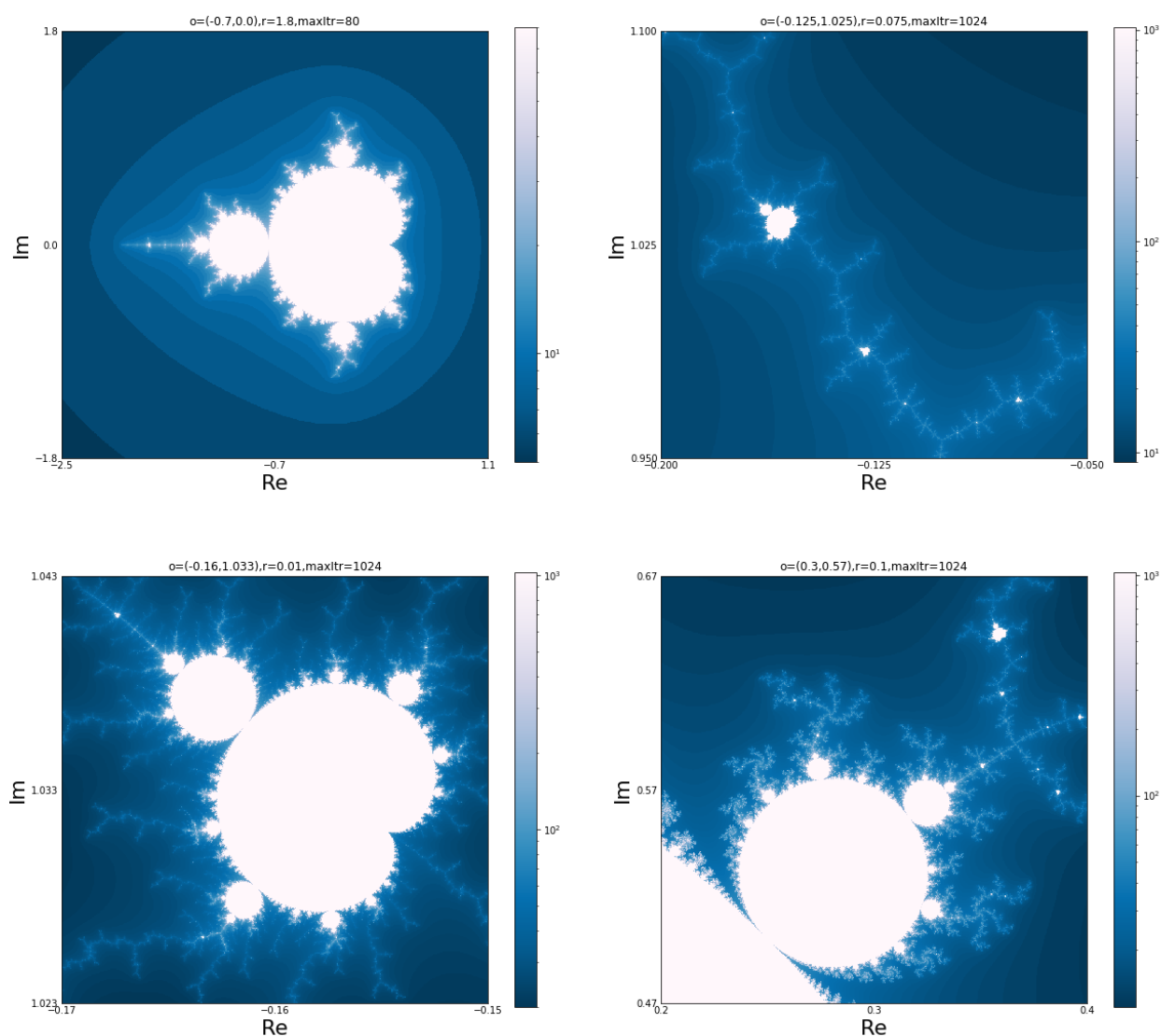
35
36 @jit
37 def mandel_set(xmin,xmax,ymin,ymax,ypix,ypix,maxItr):
38     M = 2 ** 40
39     Mandel = Mandelbrot(M, maxItr)
40     re = np.linspace(xmin, xmax, xpix)
41     im = np.linspace(ymin, ymax, ypix)
42     z = np.empty((xpix, ypix))
43     for i in tqdm(range(xpix)):
44         for j in range(ypix):
45             z[j][i] = Mandel.Calc(re[i], im[j])
46             #z[j][i] = Mandel.Calculate(re[i] + 1j*im[j])
47     return re, im, z
48
49 def mandel_image(x,y,radius,width=10,height=10,dpi=72,maxItr=80):
50     xpix, ypix = dpi * width, dpi * height
51     xmin, xmax = x - radius, x + radius
52     ymin, ymax = y - radius, y + radius
53     re, im, z = mandel_set(xmin,xmax,ymin,ymax,xpix,ypix,maxItr)
54     return re, im, np.array(z)
55
56 def mandel_plot(data,geom,gamma,cmap,normz,title,figfile):
57     x,y,Z = data[0], data[1], data[2]
58     xmin,xmax,ymin,ymax,width,height,dpi = geom[0],geom[1],geom[2],geom[3],geom[4],
59         geom[5],geom[6]
59     X, Y = np.meshgrid(x, y)
60     plt.rcParams['figure.figsize'] = width,height
61     plt.rcParams["figure.dpi"] = dpi
62     plt.gca().set_aspect('equal', adjustable='box')
63     if normz=='Power':
64         norm = colors.PowerNorm(gamma)
65     elif normz=='Log':
66         norm = colors.LogNorm(vmin=Z.min(), vmax=Z.max())
67     pcm = plt.pcolor(X, Y, Z,norm=norm,cmap=cmap)
68     plt.colorbar(pcm,shrink=0.82)
69     plt.title(title)
70     plt.xlabel('Re', fontsize=22)
71     plt.ylabel('Im', fontsize=22)
72     plt.xticks(np.linspace(xmin,xmax,3), color="black")
73     plt.yticks(np.linspace(ymin,ymax,3), color="black")
74     plt.tick_params(length=0)
75     plt.savefig(figfile)
76     plt.show()
77
78 if __name__ == '__main__':
79     project = 'mandel010'
80     csvfile,figfile,txtfile = project+'.csv', project+'.png', project+'.txt'
81     u = input('(1)Calculate,(2)Display: ')
82     if u=='1':
83         radius = 0.0005
84         x_o = -0.75 + radius
85         y_o = 0.04 + radius
86         maxItr = 1024
87         width,height,dpi = 10,10,72
88         gamma = 0.8
89         x, y, Z = mandel_image(x_o,y_o,radius,width,height,dpi,maxItr)
90         np.savetxt(csvfile, Z)
91         with open( txtfile, 'w', encoding='utf-8') as f:
92             f.write('radius,o_x,o_y,maxItr,width,height,dpi,gamma,normalize\n')
93             f.write('{:f},{:f},{:f},{:d},{:d},{:d},{:d},{:f},{:s}'.\
94                 format(radius,x_o,y_o,maxItr,width,height,dpi,gamma,'Power'))
95     else:
96         Z = np.loadtxt(csvfile)
97         with open( txtfile, 'r', encoding='utf-8') as f:
98             reader = csv.reader(f)

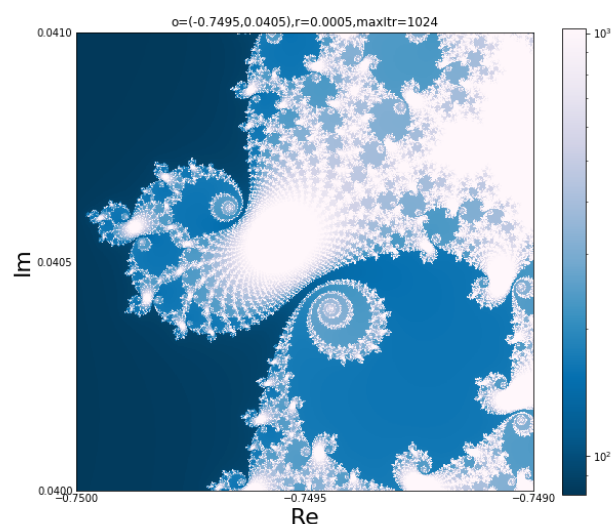
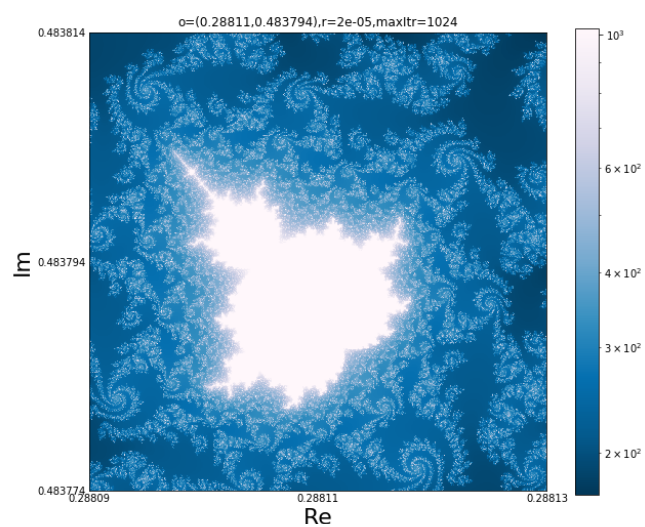
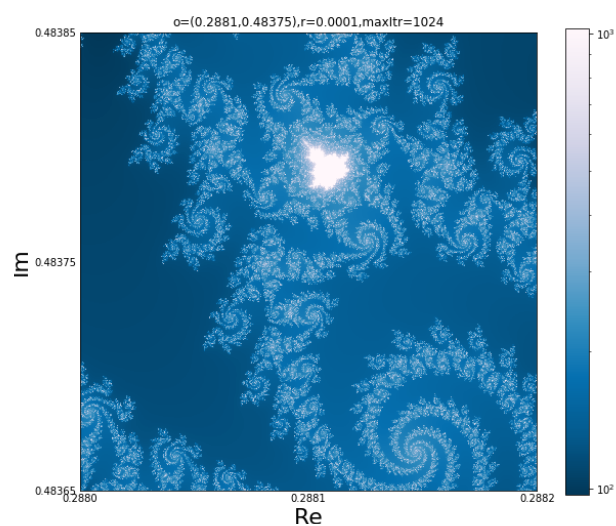
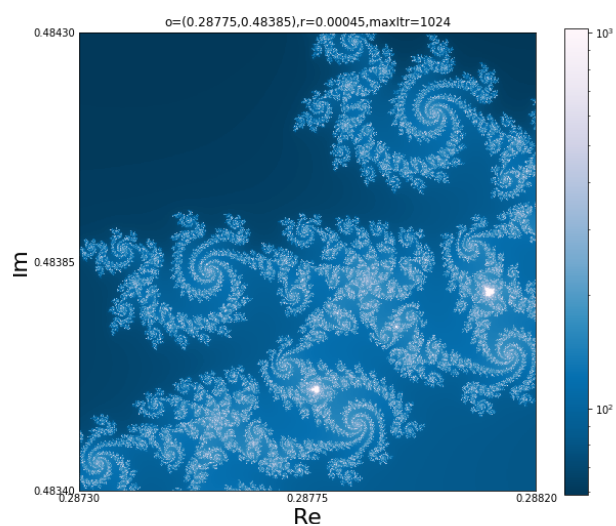
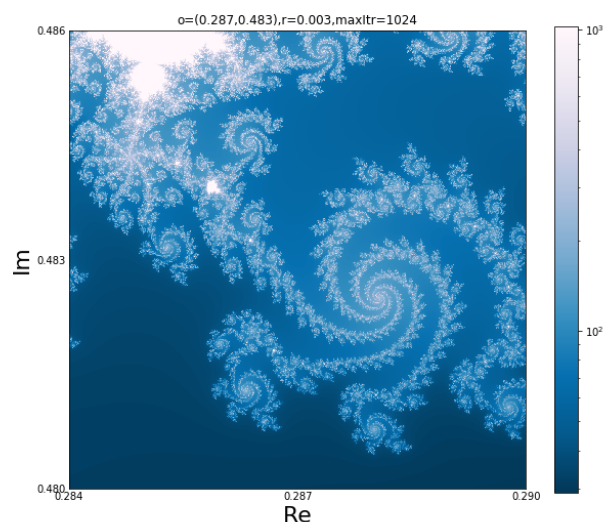
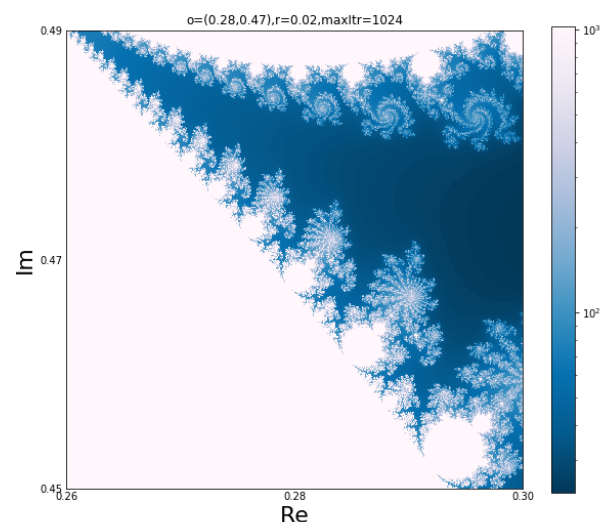
```

```

99     for row in reader:
100         print(row)
101         radius = float(row[0])
102         x_o, y_o = float(row[1]), float(row[2])
103         maxItr = int(row[3])
104         width, height = int(row[4]), int(row[5])
105         dpi = int(row[6])
106         gamma = float(row[7])
107         normz = row[8]
108         xmin, xmax = x_o - radius, x_o + radius
109         ymin, ymax = y_o - radius, y_o + radius
110         x = np.linspace(xmin, xmax, dpi*width)
111         y = np.linspace(ymin, ymax, dpi*height)
112         cmap = 'PuBu_r'
113         title = 'o={0},{1},r={2},maxItr={3}'.format(x_o,y_o,radius,maxItr)
114         normz = 'Log'
115         data = (x,y,Z)
116         geom = (xmin,xmax,ymin,ymax,width,height,dpi)
117         mandel_plot(data,geom,gamma,cmap,normz,title,figfile)

```







## 【別解】

## ソースコード 2.4 マンデルブロー集合

```

1  '''
2  mandelbrotset.py
3  Draw a Mandelbrot set
4  Using "Escape time algorithm" from:
5  http://en.wikipedia.org/wiki/Mandelbrot_set#Computer_drawings
6  '''
7  from numba import jit
8  from tqdm import tqdm
9  import matplotlib.pyplot as plt
10
11  # Subset of the complex plane we are considering
12  x0, x1 = -2.5, 1 # x0, x1 = -0.75, -0.749
13  y0, y1 = -1, 1 # y0, y1 = 0.040, 0.041
14
15  @jit
16  def mandelbrot_set(n, max_iterations):
17      image = initialize_image(n, n)
18      # Generate a set of equally spaced points
19      # in the region above
20      dx = (x1-x0)/(n-1)
21      dy = (y1-y0)/(n-1)
22      x_coords = [x0 + i*dx for i in range(n)]
23      y_coords = [y0 + i*dy for i in range(n)]
24      #
25      for i, x in tqdm(enumerate(x_coords)):
26          for k, y in enumerate(y_coords):
27              z1 = complex(0, 0)
28              iteration = 0
29              c = complex(x, y)
30              while (abs(z1) < 2 and iteration < max_iterations):
31                  z1 = z1**2 + c
32                  iteration += 1
33              image[k][i] = iteration
34      return image
35
36  def initialize_image(x_p, y_p):
37      image = []
38      for i in range(y_p):
39          x_colors = []
40          for j in range(x_p):
41              x_colors.append(0)
42          image.append(x_colors)
43      return image
44
45  def draw_mandelbrot(n, max_iterations):
46      image = mandelbrot_set(n, max_iterations)
47      cmap = 'hot' # cmap = 'Greys_r' or 'PuBu_r' or 'hot' etc
48      # 画素の補間 (Nearest neighbor, Bilinear, Bicubic)
49      plt.imshow(image, origin='lower', extent=(x0, x1, y0, y1),
50                  cmap=cmap, interpolation='nearest')
51      plt.savefig('mandelbrotset00.png')
52      plt.show()
53
54  if __name__ == '__main__':
55      n = int( input('Enter the image size nxn points (100 to 1000) : ') )
56      max_iter = int( input('Enter the max number of iteration (100 to 10000) : ') )
57      draw_mandelbrot(n, max_iter)
58      # draw_mandelbrot(n=1000, max_iteration=100)

```

`max_iter` が大きいと、この程度の画素 ( $n=1000$ ) では境界周辺が塗りつぶされて描かれない

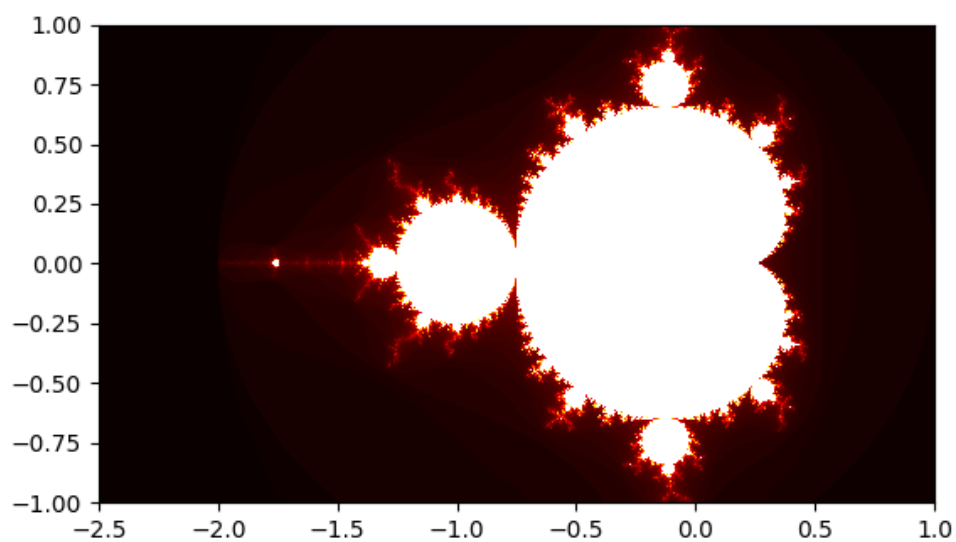


図 2.1  $n=1000$ , `max_iter=100`, 02:05 elapsed

`max_iter` を大きくしないと、白い部分の詳細が描かれない

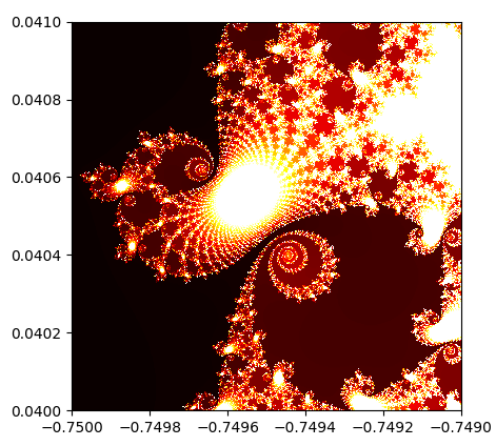


図 2.2  $n=1000$ , `max_iter=1000`, 19:18 elapsed

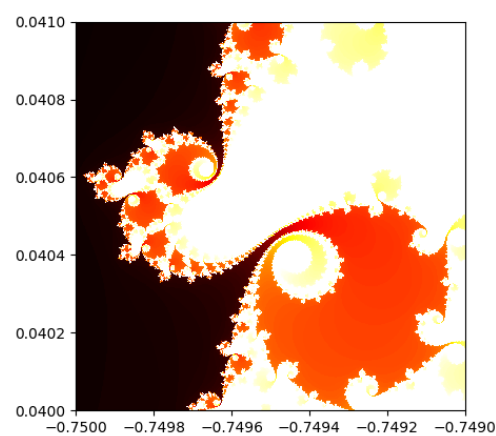


図 2.3  $n=1000$ , `max_iter=250`, 15:36 elapsed

## 第3章

# ジュリア集合

漸化式はマンデルブロ集合の場合と同じままで、 $c$  ( $\in \mathbb{Z}$ ) は予め定数として固定しておき、 $z_0$  を変数として、複素平面上の点を順に与えて計算した時に、漸化式が発散しないような  $z_0$  を要素とする集合は、Julia(ジュリア) 集合と呼ばれる

実際のプログラムは次の通り

ソースコード 3.1 複素数で計算

```
1 def Calculate(self, z0): # z0: Complex number
2     z = z0
3     for n in range(self.maxIter):
4         az = abs(z)
5         if az > self.M:
6             return n
7         z = z*z + self.c
8     return self.maxIter
```

なお、実数の世界で計算するならば、

ソースコード 3.2 実数で計算

```
1 def Calc(self, re, im): # re, im: Real number
2     z_r = re
3     z_i = im
4     for n in range(self.maxIter):
5         zr2 = z_r**2
6         zi2 = z_i**2
7         if zr2 + zi2 > self.M:
8             return n
9         z_r_ = zr2 - zi2 + self.c_r
10        z_i_ = 2*z_r*z_i + self.c_i
11        z_r, z_i = z_r_, z_i_
12    return self.maxIter
```

ここでも発散の速度をヒートマップにすることで、グラフを可視化することができる  
複素定数の  $c$  (プログラム中の  $c_r$  と  $c_i$ ) を変えると、色々なグラフが得られる

ソースコード 3.3 ジュリア集合

```
1 import numpy as np
2 from numba import jit
3 from tqdm import tqdm
4 import matplotlib.pyplot as plt
5 from matplotlib import colors
6 import csv
```

```

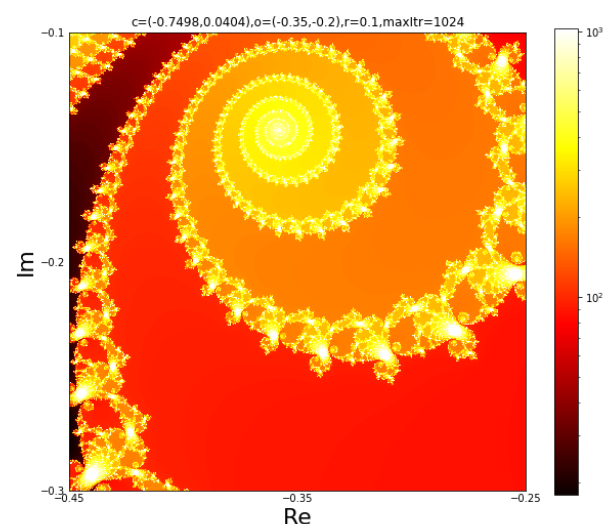
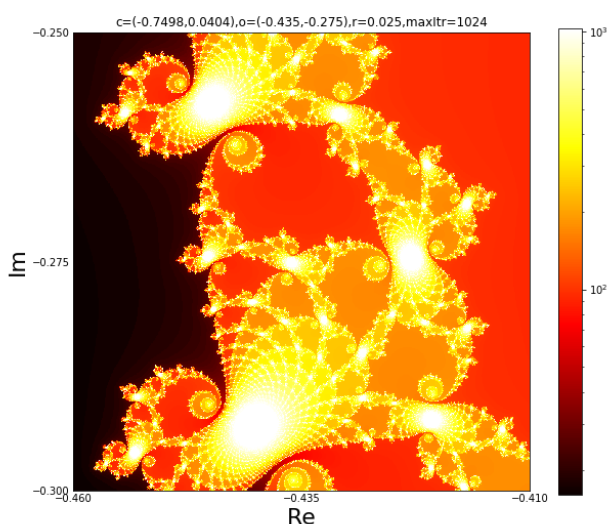
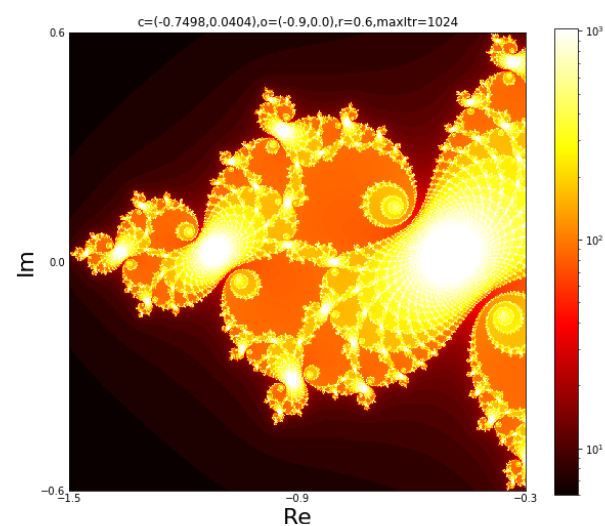
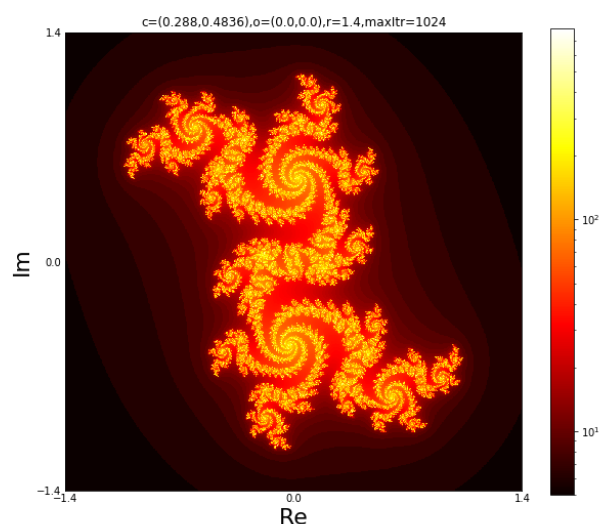
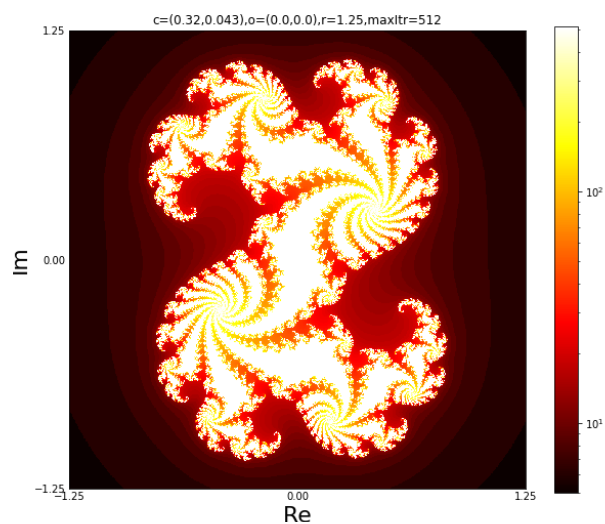
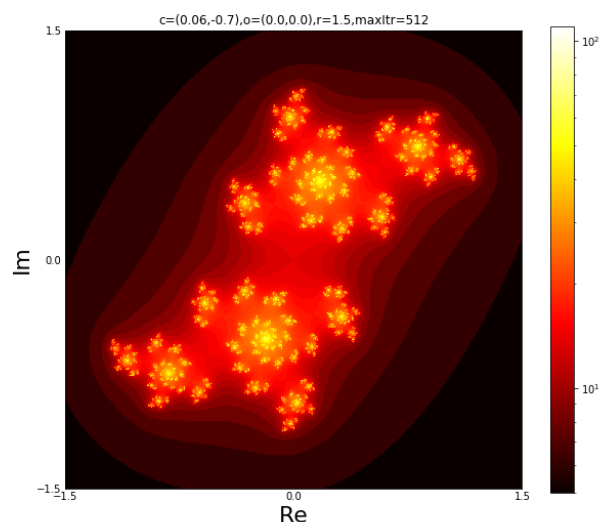
7
8 class Julia:
9     def __init__(self, M, maxItr, c_r, c_i):
10         self.M = M
11         self.maxItr = maxItr
12         self.c_r = c_r
13         self.c_i = c_i
14         self.c = c_r + 1j*c_i
15
16     def Calc(self, re, im): # re, im: Real number
17         z_r = re
18         z_i = im
19         for n in range(self.maxItr):
20             zr2 = z_r**2
21             zi2 = z_i**2
22             if zr2 + zi2 > self.M:
23                 return n
24             z_r_ = zr2 - zi2 + self.c_r
25             z_i_ = 2*z_r*z_i + self.c_i
26             z_r, z_i = z_r_, z_i_
27         return self.maxItr
28
29     def Calculate(self, z0): # z0: Complex number
30         z = z0
31         for n in range(self.maxItr):
32             az = abs(z)
33             if az > self.M:
34                 return n
35             z = z*z + self.c
36         return self.maxItr
37
38 @jit
39 def julia_set(c_r,c_i,xmin,xmax,ymin,ymax,ypix,ypix,maxItr):
40     M = 2 ** 40
41     J = Julia(M, maxItr, c_r, c_i)
42     re = np.linspace(xmin, xmax, xpix)
43     im = np.linspace(ymin, ymax, ypix)
44     z = np.empty((xpix,ypix))
45     for i in tqdm(range(xpix)):
46         for j in range(ypix):
47             z[j][i] = J.Calc(re[i], im[j])
48             #z[j][i] = J.Calculate(re[i] + 1j*im[j])
49     return re, im, z
50
51 def julia_image(c_r,c_i,x,y,radius,width=10,height=10,dpi=72,maxItr=80):
52     xpix, ypix = dpi * width, dpi * height
53     xmin, xmax = x - radius, x + radius
54     ymin, ymax = y - radius, y + radius
55     re,im,z = julia_set(c_r,c_i,xmin,xmax,ymin,ymax,xpix,ypix,maxItr)
56     return re, im, np.array(z)
57
58 def julia_plot(data,geom,gamma,cmap,normz,title,figfile):
59     x,y,Z = data[0], data[1], data[2]
60     xmin,xmax,ymin,ymax,width,height,dpi = geom[0],geom[1],geom[2],geom[3],geom[4],
61         geom[5],geom[6]
62     X, Y = np.meshgrid(x, y)
63     plt.rcParams['figure.figsize'] = width,height
64     plt.rcParams["figure.dpi"] = dpi
65     plt.gca().set_aspect('equal', adjustable='box')
66     if normz=='Power':
67         norm = colors.PowerNorm(gamma)
68     elif normz=='Log':
69         norm = colors.LogNorm(vmin=Z.min(), vmax=Z.max())
70     pcm = plt.pcolor(X, Y, Z,norm=norm,cmap=cmap)
71     plt.colorbar(pcm,shrink=0.82)

```

```

71 plt.title(title)
72 plt.xlabel('Re', fontsize=22)
73 plt.ylabel('Im', fontsize=22)
74 plt.xticks(np.linspace(xmin,xmax,3), color="black")
75 plt.yticks(np.linspace(ymin,ymax,3), color="black")
76 plt.tick_params(length=0)
77 plt.savefig(figfile)
78 plt.show()
79
80 if __name__ == '__main__':
81     project = 'julia00A'
82     csvfile, figfile, txtfile = project+'.csv', project+'.png', project+'.txt'
83     u = input('(1) Calculate, (2) Display: ')
84     if u=='1':
85         radius = 1.4
86         x_o = 0
87         y_o = 0
88         c_r = 0.288
89         c_i = 0.4836
90         maxItr = 1024
91         width = 10
92         height = 10
93         dpi = 220
94         gamma = 0.6
95         x,y,Z = julia_image(c_r,c_i,x_o,y_o,radius,width,height,dpi,maxItr)
96         np.savetxt(csvfile, Z)
97         with open( txtfile, 'w', encoding='utf-8') as f:
98             f.write('c_real,c_imag,radius,o_x,o_y,maxItr,width,height,dpi,gamma,
99                     normalize\n')
100             f.write('{:f},{:f},{:f},{:f},{:f},{:d},{:d},{:d},{:d},{:f},{:s}'.\
101                     format(c_r,c_i,radius,x_o,y_o,maxItr,width,height,dpi,gamma,'
102                             Power'))
103
104     else:
105         Z = np.loadtxt(csvfile)
106         with open( txtfile, 'r', encoding='utf-8') as f:
107             reader = csv.reader(f)
108             for row in reader:
109                 print(row)
110                 c_r, c_i = float(row[0]), float(row[1])
111                 radius = float(row[2])
112                 x_o, y_o = float(row[3]), float(row[4])
113                 maxItr = int(row[5])
114                 width, height = int(row[6]), int(row[7])
115                 dpi = int(row[8])
116                 gamma = float(row[9])
117                 normz = row[10]
118
119                 xmin, xmax = x_o - radius, x_o + radius
120                 ymin, ymax = y_o - radius, y_o + radius
121                 x = np.linspace(xmin, xmax, dpi*width)
122                 y = np.linspace(ymin, ymax, dpi*height)
123                 cmap = 'hot'
124                 title = 'c=({},{}),o=({},{}),r={},maxItr={}'.format(c_r,c_i,x_o,y_o,radius,
125                             maxItr)
126                 normz = 'Log'
127                 data = (x,y,Z)
128                 geom = (xmin,xmax,ymin,ymax,width,height,dpi)
129                 julia_plot(data,geom,gamma,cmap,normz,title,figfile)

```

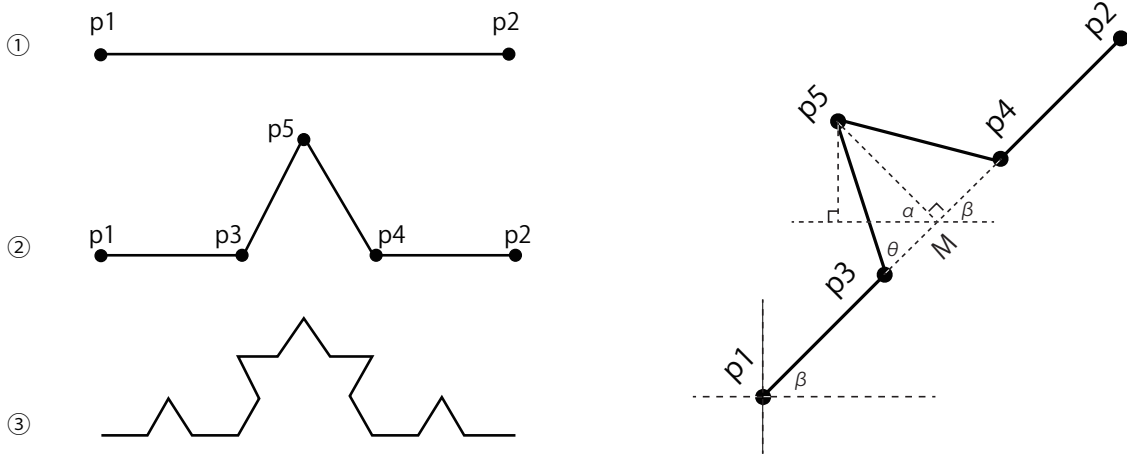


## 第4章

# 再帰的处理による図形

### 4.1 コッホ曲線

Helge von Koch が考案したフラクタル図形



①の線分  $p_1(x_1, y_1)$ 、 $p_2(x_2, y_2)$  が与えられて、 $p_3, p_4, p_5$  の座標を求める問題を考える  
例えば、点  $p_3(x_3, y_3)$  ならば次の通り

$$x_3 = x_1 + \frac{x_2 - x_1}{3} = \frac{1}{3}(2x_1 + x_2), \quad y_3 = y_1 + \frac{y_2 - y_1}{3} = \frac{1}{3}(2y_1 + y_2) \quad (4.1)$$

$p_4(x_4, y_4)$  も同様に

$$x_4 = x_1 + \frac{2(x_2 - x_1)}{3} = \frac{1}{3}(x_1 + 2x_2), \quad y_4 = y_1 + \frac{2(y_2 - y_1)}{3} = \frac{1}{3}(y_1 + 2y_2) \quad (4.2)$$

点  $p_5$  の座標は、点  $p_1$  と点  $p_2$  の中間点  $M(x_M, y_M)$  を基準に計算を進める

$$x_M = x_1 + \frac{1}{2}(x_2 - x_1) = \frac{1}{2}(x_1 + x_2), \quad y_M = y_1 + \frac{1}{2}(y_2 - y_1) = \frac{1}{2}(y_1 + y_2)$$

辺  $p_1p_3$ 、辺  $p_3p_4$ 、辺  $p_4p_2$ 、そして辺  $p_3p_5$ 、辺  $p_5p_4$  の長さ  $l$  は等しくて、

$$l = \sqrt{(x_4 - x_3)^2 + (y_4 - y_3)^2} = \frac{1}{3}\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

また、正三角形なので  $\theta = \pi/3[\text{rad}]$  より、その正三角形の高さは  $h = l \sin \theta = l\sqrt{3}/2$  で求まる

$$x_5 = x_M - h \cos \alpha, \quad y_5 = y_M + h \sin \alpha \quad (\text{但し } \alpha = \frac{\pi}{2} - \beta) \quad (4.3)$$

ここで、角度  $\beta$  は、

$$\beta = \tan^{-1} \left( \frac{y_2 - y_1}{x_2 - x_1} \right)$$

となるのだが、 $x_2 - x_1$  が 0 になる場合（即ち  $x_1 = x_2$  の場合、この時  $y_5 = y_M$ 、あるいは  $\beta$  が  $\pi/2$  や  $3\pi/4$  になる）、 $\tan$  や  $\arctan$  は計算できないので、プログラム上では特別扱いが必要になる

即ち、 $x_1 = x_2$  の時、

$$x_5 = x_M - h, \quad y_5 = y_M, \quad (\text{但し、} h = l \sin \frac{\pi}{3} = l \frac{\sqrt{3}}{2})$$

$p_1$  の  $y$  座標が  $p_2$  の  $y$  座標より小さいとき、 $p_5$  は左へ飛び出しているが、 $p_1$  の  $y$  座標が  $p_2$  の  $y$  座標より大きいとき、 $p_5$  は右へ飛び出しているので、上の式の  $x_5 = x_M - h$  で  $h$  の符号を反転させる必要がある

ソースコード 4.1 コッホの曲線

```

1  import math
2  import matplotlib.pyplot as plt
3
4  class koch():
5      def __init__( self, p1, p2, gene=0 ):
6          self.result=[]
7          self.gene = gene
8          self.p1 = p1
9          self.p2 = p2
10         self.p3 = [(2*self.p1[0] + self.p2[0]) / 3, (2*self.p1[1] + self.p2[1]) / 3]
11         self.p4 = [(self.p1[0] + 2*self.p2[0]) / 3, (self.p1[1] + 2*self.p2[1]) / 3]
12         xm = [0,0]
13         xm[0] = (self.p2[0] + self.p1[0]) / 2
14         xm[1] = (self.p2[1] + self.p1[1]) / 2
15         l = math.sqrt((self.p2[0]-self.p1[0])**2+(self.p2[1]-self.p1[1])**2)/3
16         h = l*math.sqrt(3)/2
17         if self.p2[0]==self.p1[0]:
18             if self.p1[1] < self.p2[1]:
19                 h=-h
20             self.p5 = [xm[0]-h,xm[1]]
21         else:
22             beta = math.atan((self.p2[1]-self.p1[1])/(self.p2[0]-self.p1[0]))
23             alpha = math.pi/2-beta
24             if self.p1[0] > self.p2[0]:
25                 self.p5 = [xm[0]+h*math.cos(alpha),xm[1]-h*math.sin(alpha)]
26             else:
27                 self.p5 = [xm[0]-h*math.cos(alpha),xm[1]+h*math.sin(alpha)]
28         self.generate()
29
30     def generate( self ):
31         if self.gene > 0:
32             k1 = koch( self.p1, self.p3, self.gene-1 )
33             k2 = koch( self.p3, self.p5, self.gene-1 )
34             k3 = koch( self.p5, self.p4, self.gene-1 )
35             k4 = koch( self.p4, self.p2, self.gene-1 )
36             self.result.extend(k1.getResult())
37             self.result.extend(k2.getResult())
38             self.result.extend(k3.getResult())
39             self.result.extend(k4.getResult())

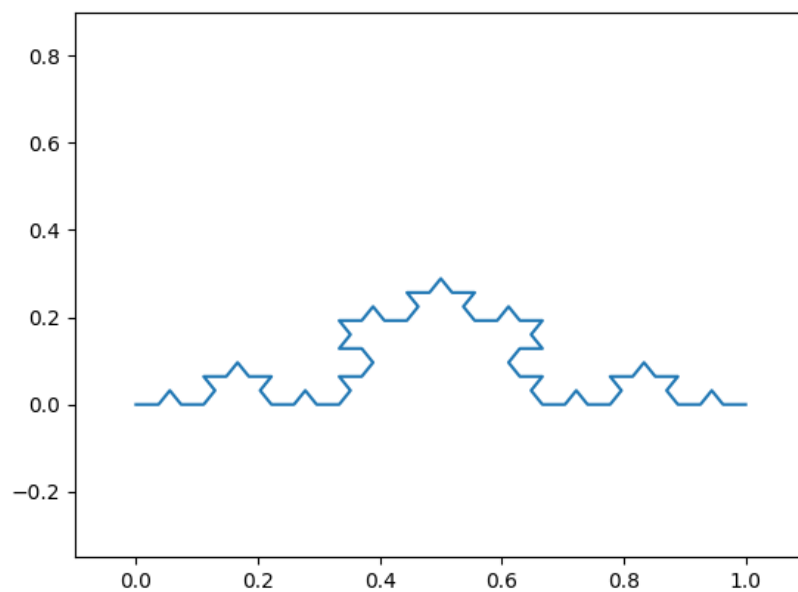
```



```

40         else:
41             #self.getPrint()
42             return self.result.extend( self.getCoordinates() )
43
44     def getCoordinates( self ):
45         return [self.p1, self.p3, self.p5, self.p4, self.p2]
46
47     def getResult( self ):
48         return self.result
49
50     def getPrint(self):
51         print( 'x:',self.p1[0],',y:',self.p1[1] )
52         print( 'x:',self.p3[0],',y:',self.p3[1] )
53         print( 'x:',self.p5[0],',y:',self.p5[1] )
54         print( 'x:',self.p4[0],',y:',self.p4[1] )
55         print( 'x:',self.p2[0],',y:',self.p2[1] )
56
57 if __name__ == '__main__':
58     fig, ax = plt.subplots()
59     ax.set_xlim(-0.1, 1.1)
60     ax.set_ylim(-0.35, 0.9)
61
62     gene=2
63     x,y=[],[]
64
65     p1, p2 = [0,0], [1,0]
66     k = koch(p1, p2, gene)
67     k0 = k.getResult()
68     for j in range( len(k0) ):
69         x.append(k0[j][0])
70         y.append(k0[j][1])
71
72     plt.plot(x,y)
73     fig.savefig('figkoch1.png')
74     fig.show()

```



## 4.1.1 コッホの雪片曲線

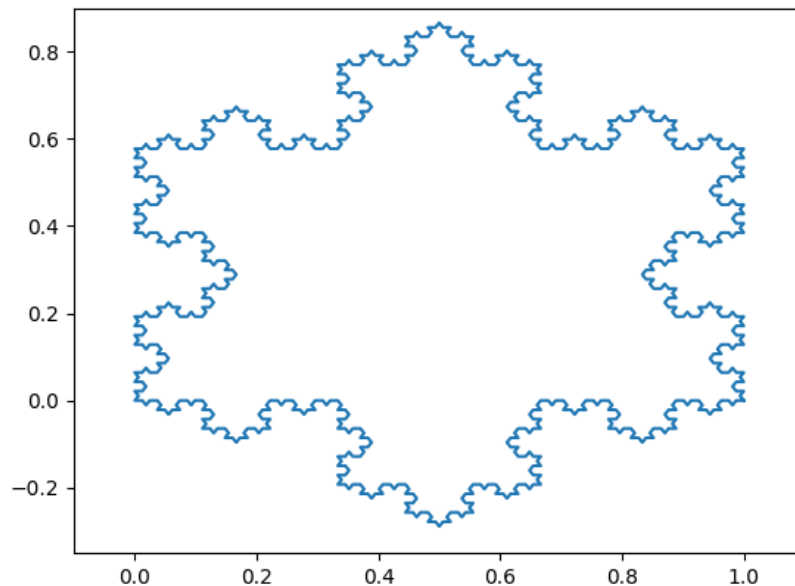
ソースコード 4.2 コッホの雪片曲線

```

1 import math
2 import matplotlib.pyplot as plt
3
4 class koch():
5     def __init__( self,p1, p2, gene=0 ):
6         self.result=[]
7         self.gene = gene
8         self.p1 = p1
9         self.p2 = p2
10        self.p3 = [(2*self.p1[0] + self.p2[0]) / 3,(2*self.p1[1] + self.p2[1]) / 3]
11        self.p4 = [(self.p1[0] + 2*self.p2[0]) / 3,(self.p1[1] + 2*self.p2[1]) / 3]
12        xm = [0,0]
13        xm[0] = (self.p2[0] + self.p1[0]) / 2
14        xm[1] = (self.p2[1] + self.p1[1]) / 2
15        l = math.sqrt((self.p2[0]-self.p1[0])**2+(self.p2[1]-self.p1[1])**2)/3
16        h = l*math.sqrt(3)/2
17        if self.p2[0]==self.p1[0]:
18            if self.p1[1] < self.p2[1]:
19                h=-h
20            self.p5 = [xm[0]-h,xm[1]]
21        else:
22            beta = math.atan((self.p2[1]-self.p1[1])/(self.p2[0]-self.p1[0]))
23            alpha = math.pi/2-beta
24            if self.p1[0] > self.p2[0]:
25                self.p5 = [xm[0]+h*math.cos(alpha),xm[1]-h*math.sin(alpha)]
26            else:
27                self.p5 = [xm[0]-h*math.cos(alpha),xm[1]+h*math.sin(alpha)]
28        self.generate()
29
30    def generate( self ):
31        if self.gene > 0:
32            k1 = koch( self.p1, self.p3, self.gene-1 )
33            k2 = koch( self.p3, self.p5, self.gene-1 )
34            k3 = koch( self.p5, self.p4, self.gene-1 )
35            k4 = koch( self.p4, self.p2, self.gene-1 )
36            self.result.extend(k1.getResult())
37            self.result.extend(k2.getResult())
38            self.result.extend(k3.getResult())
39            self.result.extend(k4.getResult())
40        else:
41            #self.getPrint()
42            return self.result.extend( self.getCoordinates() )
43
44    def getCoordinates( self ):
45        return [self.p1, self.p3, self.p5, self.p4, self.p2]
46
47    def getResult( self ):
48        return self.result
49
50    def getPrint(self):
51        print( 'x:',self.p1[0],',y:',self.p1[1] )
52        print( 'x:',self.p3[0],',y:',self.p3[1] )
53        print( 'x:',self.p5[0],',y:',self.p5[1] )
54        print( 'x:',self.p4[0],',y:',self.p4[1] )
55        print( 'x:',self.p2[0],',y:',self.p2[1] )
56
57 if __name__ == '__main__':
58     fig, ax = plt.subplots()

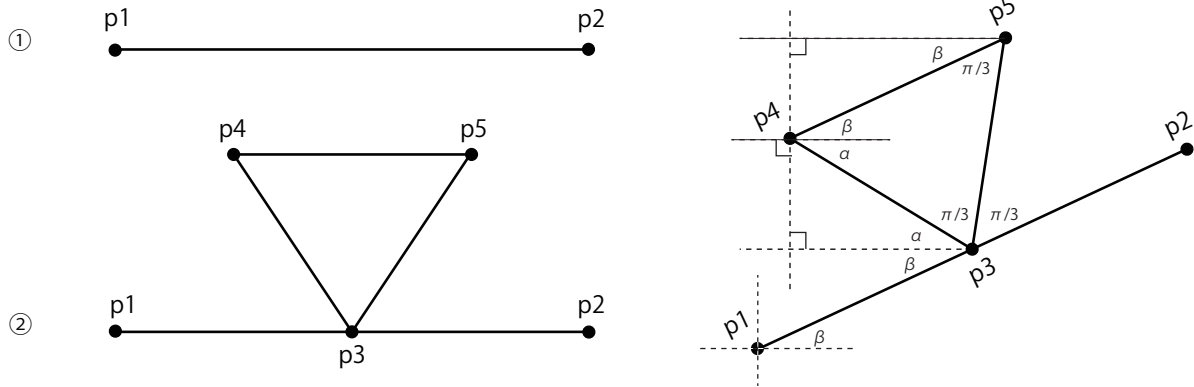
```

```
59 ax.set_xlim(-0.1, 1.1)
60 ax.set_ylim(-0.35, 0.9)
61
62 gene=3
63 x,y=[],[]
64
65 p1, p2 = [1,0], [0,0]
66 k = koch(p1, p2, gene)
67 k0 = k.getResult()
68 for j in range( len(k0) ):
69     x.append(k0[j][0])
70     y.append(k0[j][1])
71
72 p1, p2 = [0,0], [1/2,math.sqrt(3)/2]
73 k = koch(p1, p2, gene)
74 k0 = k.getResult()
75 for j in range( len(k0) ):
76     x.append(k0[j][0])
77     y.append(k0[j][1])
78
79 p1, p2 = [1/2,math.sqrt(3)/2],[1,0]
80 k = koch(p1, p2, gene)
81 k0 = k.getResult()
82 for j in range( len(k0) ):
83     x.append(k0[j][0])
84     y.append(k0[j][1])
85
86 plt.plot(x,y)
87 fig.savefig('figkoch2.png')
88 fig.show()
```



## 4.2 シェルピンスキーの三角形

Waclaw Sierpinski's gasket



①の線分が2つの点  $p_1, p_2$  によって与えられた時に、3つの点  $p_3, p_4, p_5$  の座標を求める問題を考える  
点  $p_3(x_3, y_3)$  は、2つの点  $p_1$  と  $p_2$  の中点である

$$x_3 = x_1 + \frac{x_2 - x_1}{2} = \frac{1}{2}(x_1 + x_2), \quad y_3 = y_1 + \frac{y_2 - y_1}{2} = \frac{1}{2}(y_1 + y_2) \quad (4.4)$$

辺  $p_1p_3$ 、辺  $p_3p_4$ 、辺  $p_4p_5$ 、辺  $p_5p_3$ 、及び辺  $p_3p_2$  の長さ  $l$  は等しくて

$$l = \frac{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}}{2}$$

従って、三角形  $p_3 p_4 p_5$  は正三角形である

点  $p_4(x_4, y_4)$  は、角度  $\alpha$  と辺の長さ  $l$  を使って、

$$x_4 = x_3 - l \cos \alpha, \quad y_4 = y_3 + l \sin \alpha \quad \text{但し、} \alpha = \frac{\pi}{3} - \beta \quad (4.5)$$

ここで、 $\beta$  は次の様にして予め計算しておく必要があります

$$\beta = \tan^{-1} \left( \frac{y_2 - y_1}{x_2 - x_1} \right)$$

この  $\beta$  は  $x_1 = x_2$  の時に  $\arctan$  や  $\tan$  は計算できないので、プログラム上の特別扱いが必要です  
即ち  $x_1 = x_2$  の時というのは、 $\beta$  が  $\pi/2[\text{rad}]$  あるいは  $3\pi/4[\text{rad}]$  の場合ですが、

$$y_4 = y_1 + \frac{l}{2}, \quad y_5 = y_3 + \frac{l}{2}, \quad x_4 = x_5 = x_1 - h, \quad (\text{但し、} h = l \sin \frac{\pi}{3} = l \frac{\sqrt{3}}{2})$$

ここまでに計算した  $p_4(x_4, y_4)$  を元にして  $p_5(x_5, y_5)$  を求めます

$$x_5 = x_4 + l \cos \beta, \quad y_5 = y_4 + l \sin \beta \quad (4.6)$$

## 【別解】

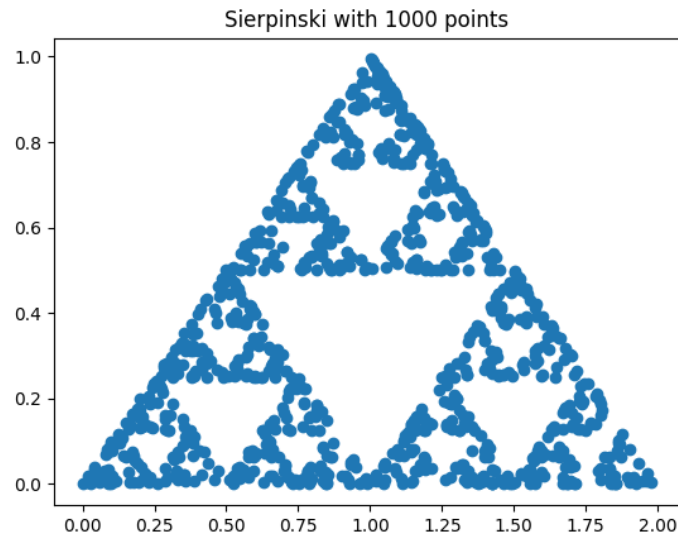
ソースコード 4.3 シェルピンスキーの三角形

```
1  '''
2  sierpinski.py
3  Draw Sierpinski Triangle
4  '''
5  import random
6  import matplotlib.pyplot as plt
7
8  def transformation_1(p):
9      x = p[0]
10     y = p[1]
11     x1 = 0.5*x
12     y1 = 0.5*y
13     return x1, y1
14
15  def transformation_2(p):
16     x = p[0]
17     y = p[1]
18     x1 = 0.5*x + 0.5
19     y1 = 0.5*y + 0.5
20     return x1, y1
21
22  def transformation_3(p):
23     x = p[0]
24     y = p[1]
25     x1 = 0.5*x + 1
26     y1 = 0.5*y
27     return x1, y1
28
29  def get_index(probability):
30     r = random.random()
31     c_probability = 0
32     sum_probability = []
33     for p in probability:
34         c_probability += p
35         sum_probability.append(c_probability)
36     for item, sp in enumerate(sum_probability):
37         if r <= sp:
38             return item
39     return len(probability)-1
40
41  def transform(p):
42     # list of transformation functions
43     transformations = [transformation_1, transformation_2, transformation_3]
44     probability = [1/3, 1/3, 1/3]
45     # pick a random transformation function and call it
46     tindex = get_index(probability)
47     t = transformations[tindex]
48     x, y = t(p)
49     return x, y
50
51  def draw_sierpinski(n):
52     # We start with (0, 0)
53     x = [0]
54     y = [0]
55     x1, y1 = 0, 0
56     for i in range(n):
57         x1, y1 = transform((x1, y1))
58         x.append(x1)
59         y.append(y1)
60     return x, y
61
```

```

62 if __name__ == '__main__':
63     n = int( input('Enter the number of points in the Sierpinski:') )
64     x, y = draw_sierpinski( n )
65     # Plot the points
66     plt.plot(x, y, 'o')
67     plt.title('Sierpinski with {0} points'.format(n))
68     plt.savefig('sierpinski.png')
69     plt.show()

```



### 4.3 バーンスレイのシダ

Michael Barnsley

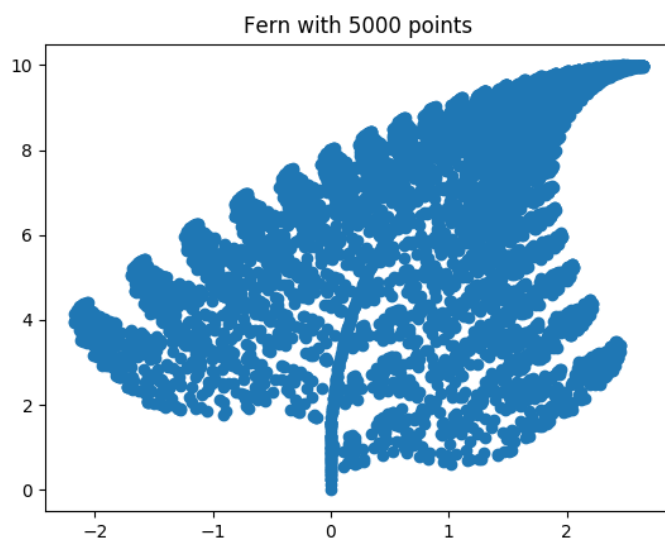
ソースコード 4.4 バーンスレイのシダ

```

1  import random
2  import matplotlib.pyplot as plt
3
4  def transformation_1( p ):
5      x = p[0]
6      y = p[1]
7      x1 = 0.85*x + 0.04*y
8      y1 = -0.04*x + 0.85*y + 1.6
9      return x1, y1
10
11 def transformation_2( p ):
12     x = p[0]
13     y = p[1]
14     x1 = 0.2*x - 0.26*y
15     y1 = 0.23*x + 0.22*y + 1.6
16     return x1, y1
17
18 def transformation_3( p ):
19     x = p[0]
20     y = p[1]
21     x1 = -0.15*x + 0.28*y
22     y1 = 0.26*x + 0.24*y + 0.44
23     return x1, y1

```

```
24
25 def transformation_4( p ):
26     x = p[0]
27     y = p[1]
28     x1 = 0
29     y1 = 0.16*y
30     return x1, y1
31
32 def get_index(probability):
33     r = random.random()
34     c_probability = 0
35     sum_probability = []
36     for p in probability:
37         c_probability += p
38         sum_probability.append( c_probability )
39     for item, sp in enumerate( sum_probability ):
40         if r <= sp:
41             return item
42     return len( probability ) - 1
43
44 def transform( p ):
45     # list of transformation function 変換関数のリスト
46     transformations = [ transformation_1, transformation_2, transformation_3,
47                         transformation_4 ]
48     probability = [0.85, 0.07, 0.07, 0.01]
49     # pick a random transformation function and call it ランダム変換関数を呼ぶ
50     tindex = get_index( probability )
51     t = transformations[tindex]
52     x,y = t( p )
53     return x, y
54
55 def draw_fern( n ):
56     # We start with (0, 0) (0,0)から始める
57     x = [0]
58     y = [0]
59     x1, y1 = 0, 0
60     for i in range( n ):
61         x1, y1 = transform( (x1, y1) )
62         x.append( x1 )
63         y.append( y1 )
64     return x, y
65
66 if __name__ == '__main__':
67     n = int( input('Enter the number of points in the Fern: ') )
68     x, y = draw_fern( n )
69     # Plot the points
70     plt.plot( x, y, 'o' )
71     plt.title('Fern with {0} points'.format(n))
72     plt.savefig('fern.png')
73     plt.show()
```





## 第5章

# おわりに

参考にした記事より（おまけ）

<https://nepia01.blogspot.com/2017/07/python.html>

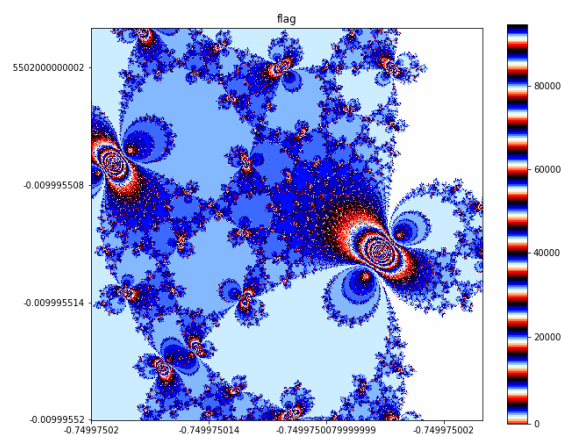
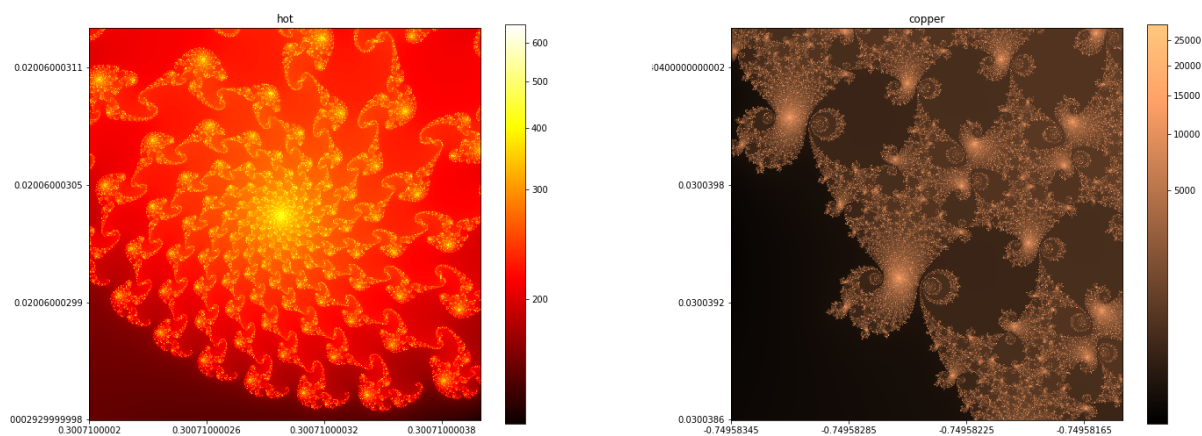
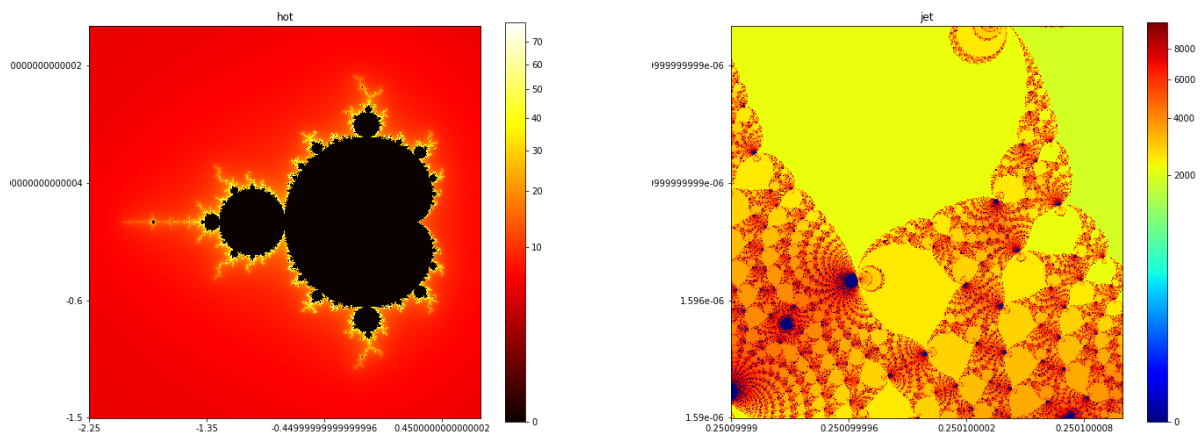
ソースコード 5.1 参考記事

```
1 # https://nepia01.blogspot.com/2017/07/python.html
2 import numpy as np
3 from numba import jit
4 from tqdm import tqdm
5
6 @jit
7 def mandelbrot(z,maxiter,horizon,log_horizon):
8     c = z
9     for n in range(maxiter):
10         az = abs(z)
11         if az > horizon:
12             return n - np.log(np.log(az))/np.log(2) + log_horizon
13         z = z*z + c
14     return 0
15
16 @jit
17 def mandelbrot_set(xmin,xmax,ymin,ymax,width,height,maxiter):
18     horizon = 2 ** 40
19     log_horizon = np.log(np.log(horizon))/np.log(2)
20     r1 = np.linspace(xmin, xmax, width)
21     r2 = np.linspace(ymin, ymax, height)
22     n3 = np.empty((width,height))
23     for i in tqdm(range(width)):
24         for j in range(height):
25             n3[i,j] = mandelbrot(r1[i] + 1j*r2[j],maxiter,horizon, log_horizon)
26     return (r1,r2,n3)
27
28 from matplotlib import pyplot as plt
29 from matplotlib import colors
30
31 def mandelbrot_image(x,y,radius,width=10,height=10,maxiter=80,cmap='jet',gamma=0.3):
32     dpi = 72
33     img_width = dpi * width
34     img_height = dpi * height
35     xmin = x - radius
36     xmax = x + radius
37     ymin = y - radius
38     ymax = y + radius
39     x,y,z = mandelbrot_set(xmin,xmax,ymin,ymax,img_width,img_height,maxiter)
40
41     fig, ax = plt.subplots(figsize=(width, height),dpi=72)
42     ticks = np.arange(0,img_width,3*dpi)
```

```
43     x_ticks = xmin + (xmax-xmin)*ticks/img_width
44     plt.xticks(ticks, x_ticks)
45     y_ticks = ymin + (ymax-ymin)*ticks/img_width
46     plt.yticks(ticks, y_ticks)
47     ax.set_title(cmap)
48
49     norm = colors.PowerNorm(gamma)
50     cax = ax.imshow(z.T, cmap=cmap, origin='lower', norm=norm)
51     fig.colorbar(cax, shrink=0.82)
52     plt.savefig('fig000.png')
53     plt.show()
54
55 mandelbrot_image(-0.75, 0, 1.5, cmap='hot', gamma=0.4)
56 #mandelbrot_image(0.2501, 1.6e-6, 1e-8, maxiter=10000)
57 #mandelbrot_image(0.3007100003, 0.02006000303, 1e-10, maxiter=2048, cmap='hot', gamma=0.4)
58 #mandelbrot_image(-0.74958245, 0.0300396, 1e-6, maxiter=30000, cmap='copper')
59 #mandelbrot_image(-0.74997501, -0.00999551, 1e-8, maxiter=100000, cmap='flag', gamma=0.98)
```

cmap='hogehoge' で指定する hogehoge は、以下に一覧がある

[https://matplotlib.org/examples/color/colormaps\\_reference.html](https://matplotlib.org/examples/color/colormaps_reference.html)



## 謝辞

## 参考文献

- [1] B.Mandelbrot 著、広中平祐 監訳 『フラクタル幾何学』第 1 版（日経サイエンス社 1986）
- [2] 宇敷重広 『フラクタルの世界』第 1 版（日本評論社 1987）
- [3] 渕上季代絵 『フラクタル CG コレクション』第 2 版（サイエンス社 1987）
- [4] 佃勉 『フラクタルの世界』第 1 版（山海堂 1987）
- [5] 佐藤幸悦 『フラクタル・グラフィックス』（ラッセル社 1989）
- [6] Amit Saha 著、黒川利明 訳 『Python から始める数学入門』初版（オライリー・ジャパン 2016）
- [7] 奥村晴彦、黒木裕介 『 $\text{\LaTeX}$  2 $\epsilon$  美文書作成入門』第 7 版（技術評論社, 2017）