Fractal Diagram

2021 年 2 月 21 日

S.Matoike

目次

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

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

において、$n \to \infty$ で z_n が発散しないような複数数 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、順次計算を売り返していく
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 ∞ 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

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

```
def Calc(self, re, im): # re, im: Real number
    x = 0.0
    y = 0.0
    for n in range(self.maxItr):
        zx2 = x**2
        zy2 = y**2
        if zx2 + zy2 > self.M:
            return n
        x_ = zx2 - zy2 + re
        y_ = 2*x*y + im
        x, y = x_, y_
    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

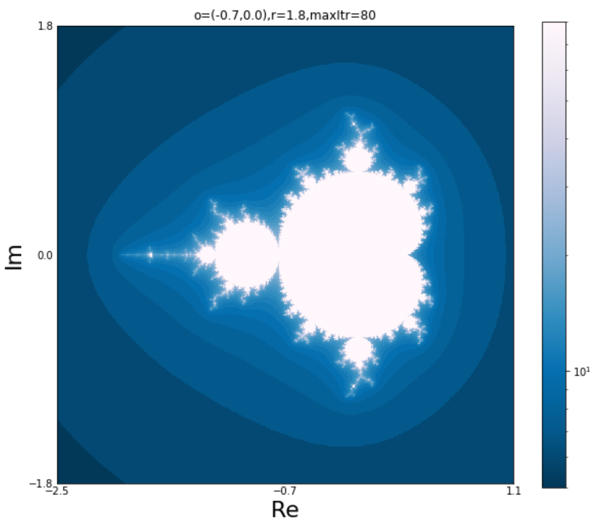
ここでは numpy.meshgrid を使ってヒートマップを描いていく

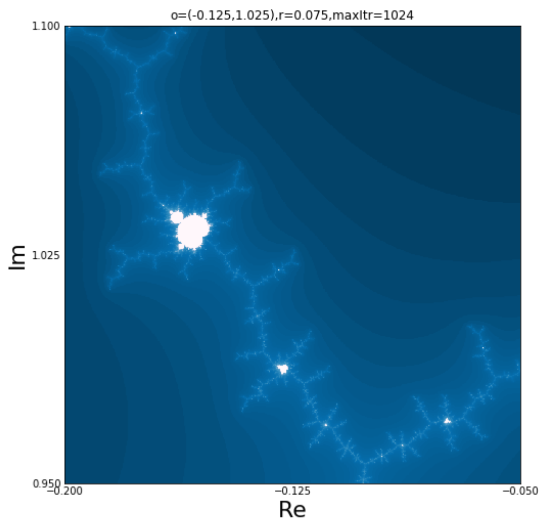
ソースコード 2.3　マンデルブロ集合

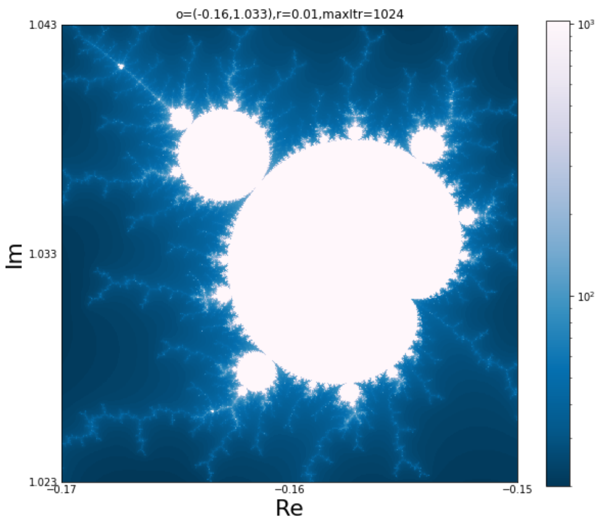
```
import numpy as np
from numba import jit
from tqdm import tqdm
import matplotlib.pyplot as plt
from matplotlib import colors
import csv

class Mandelbrot:
    def __init__(self, M, maxItr):
        self.M = M
        self.maxItr = maxItr

    def Calc(self, re, im): # re, im: 2d Array
        x = 0.0
        y = 0.0
        for n in range(self.maxItr):
            zx2 = x**2
            zy2 = y**2
            if zx2 + zy2 > self.M:
                return n
            x_ = zx2 - zy2 + re
            y_ = 2*x*y + im
            x, y = x_, y_
        return self.maxItr

    def Calculate(self, c): # c: complex
        # z0 = 0.0 + 1j*0.0
        z = c # z = z0**2 + c
        for n in range(self.maxItr):
            az = abs(z)
            if az > self.M:
                return n
            z = z*z + c
        return self.maxItr
```

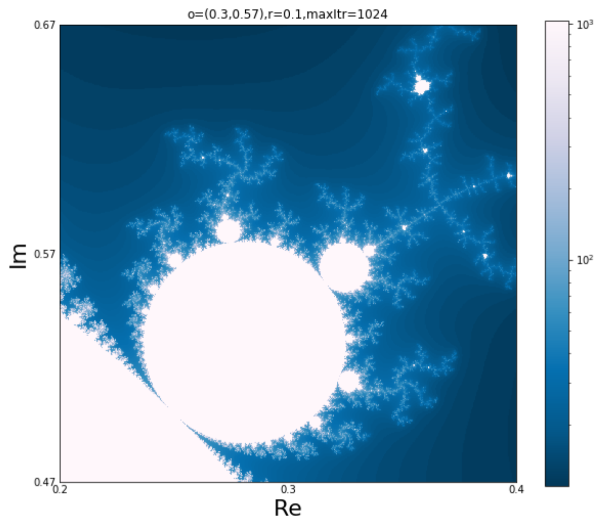
```python

@jit
def mandel_set(xmin,xmax,ymin,ymax,xpix,ypix,maxItr):
    M = 2 ** 40
    Mandel = Mandelbrot(M, maxItr)
    re = np.linspace(xmin, xmax, xpix)
    im = np.linspace(ymin, ymax, ypix)
    z  = np.empty((xpix, ypix))
    for i in tqdm(range(xpix)):
        for j in range(ypix):
            z[j][i] = Mandel.Calc(re[i], im[j])
            #z[j][i] = Mandel.Calculate(re[i] + 1j*im[j])
    return re, im, z

def mandel_image(x,y,radius,width=10,height=10,dpi=72,maxItr=80):
    xpix, ypix = dpi * width, dpi * height
    xmin, xmax = x - radius, x + radius
    ymin, ymax = y - radius, y + radius
    re, im, z = mandel_set(xmin,xmax,ymin,ymax,xpix,ypix,maxItr)
    return re, im, np.array(z)

def mandel_plot(data,geom,gamma,cmap,normz,title,figfile):
    x,y,Z = data[0], data[1], data[2]
    xmin,xmax,ymin,ymax,width,height,dpi = geom[0],geom[1],geom[2],geom[3],geom[4],geom[5],geom[6]
    X, Y = np.meshgrid(x, y)
    plt.rcParams['figure.figsize'] = width,height
    plt.rcParams["figure.dpi"] = dpi
    plt.gca().set_aspect('equal', adjustable='box')
    if normz=='Power':
        norm = colors.PowerNorm(gamma)
    elif normz=='Log':
        norm = colors.LogNorm(vmin=Z.min(), vmax=Z.max())
    pcm = plt.pcolor(X, Y, Z,norm=norm,cmap=cmap)
    plt.colorbar(pcm,shrink=0.82)
    plt.title(title)
    plt.xlabel('Re', fontsize=22)
    plt.ylabel('Im', fontsize=22)
    plt.xticks(np.linspace(xmin,xmax,3), color="black")
    plt.yticks(np.linspace(ymin,ymax,3), color="black")
    plt.tick_params(length=0)
    plt.savefig(figfile)
    plt.show()

if __name__ == '__main__':
    project = 'mandel010'
    csvfile,figfile,txtfile = project+'.csv', project+'.png', project+'.txt'
    u = input('(1)Calculate,(2)Display : ')
    if u=='1':
        radius = 0.0005
        x_o = -0.75 + radius
        y_o =  0.04 + radius
        maxItr = 1024
        width,height,dpi = 10,10,72
        gamma = 0.8
        x, y, Z = mandel_image(x_o,y_o,radius,width,height,dpi,maxItr)
        np.savetxt(csvfile, Z)
        with open( txtfile, 'w', encoding='utf-8') as f:
            f.write('radius,o_x,o_y,maxItr,width,height,dpi,gamma,normalize\n')
            f.write('{:f},{:f},{:f},{:d},{:d},{:d},{:d},{:f},{:s}'.\
                    format(radius,x_o,y_o,maxItr,width,height,dpi,gamma,'Power'))
    else:
        Z = np.loadtxt(csvfile)
        with open( txtfile, 'r', encoding='utf-8') as f:
            reader = csv.reader(f)
```

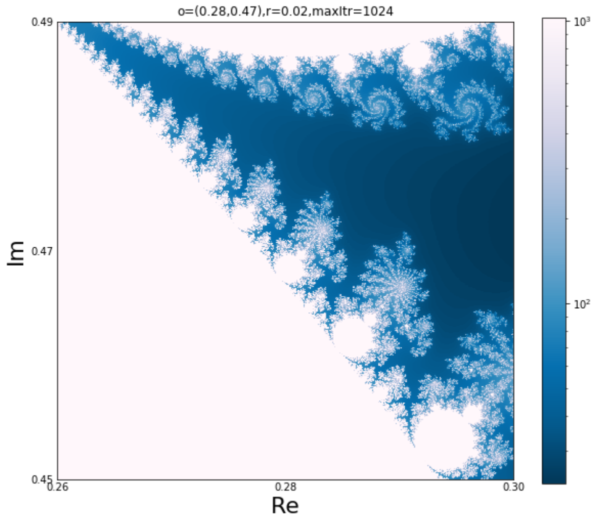
```python
            for row in reader:
                print(row)
            radius = float(row[0])
            x_o, y_o = float(row[1]), float(row[2])
            maxItr = int(row[3])
            width, height = int(row[4]), int(row[5])
            dpi = int(row[6])
            gamma = float(row[7])
            normz = row[8]
        xmin, xmax = x_o - radius, x_o + radius
        ymin, ymax = y_o - radius, y_o + radius
        x = np.linspace(xmin, xmax, dpi*width)
        y = np.linspace(ymin, ymax, dpi*height)
        cmap = 'PuBu_r'
        title = 'o=({0},{1}),r={2},maxItr={3}'.format(x_o,y_o,radius,maxItr)
        normz = 'Log'
        data = (x,y,Z)
        geom = (xmin,xmax,ymin,ymax,width,height,dpi)
        mandel_plot(data,geom,gamma,cmap,normz,title,figfile)
```

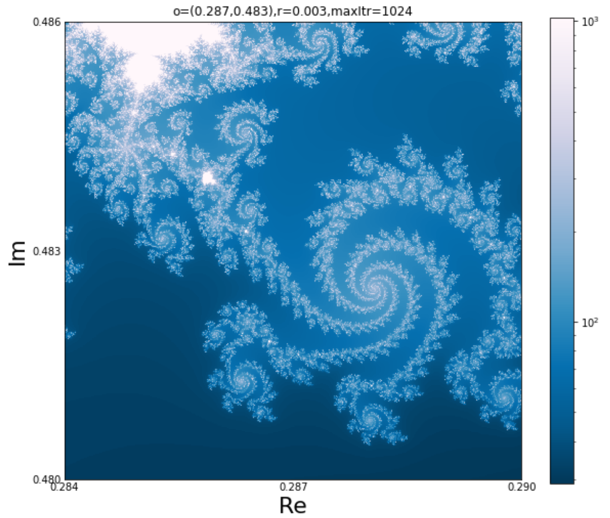


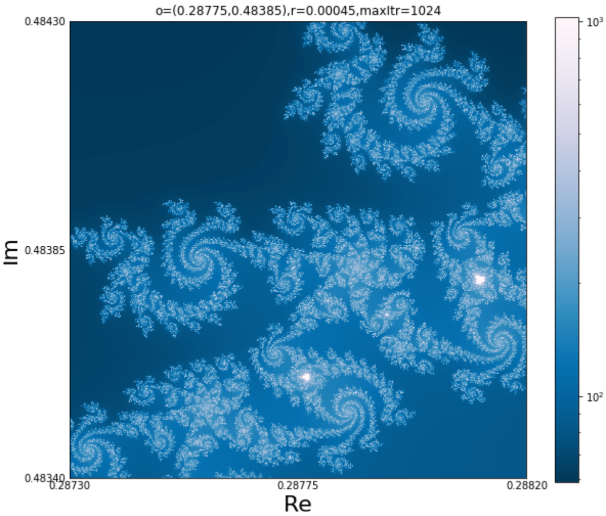


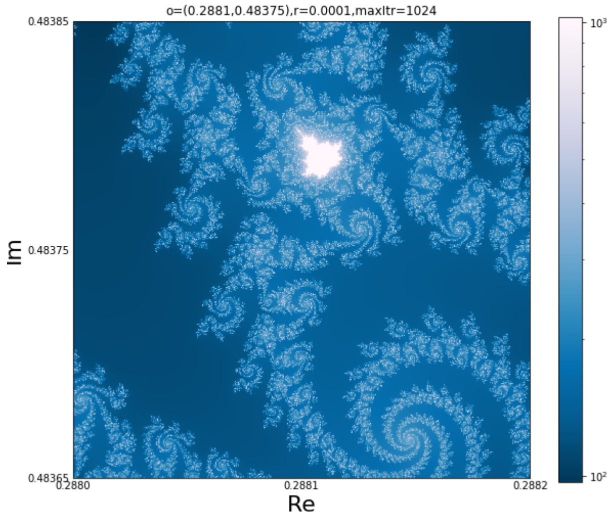


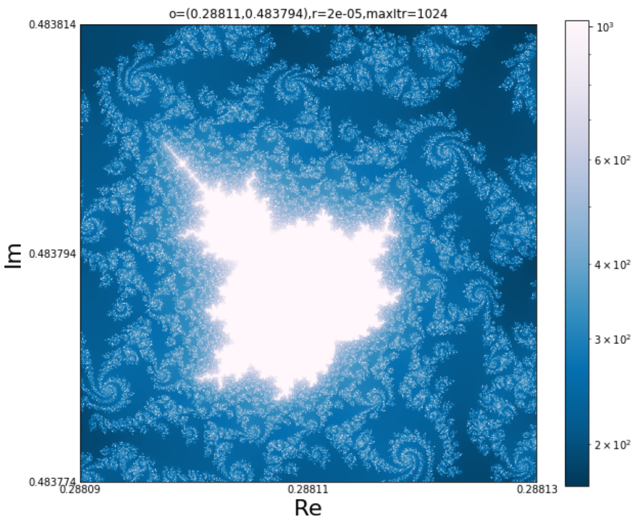


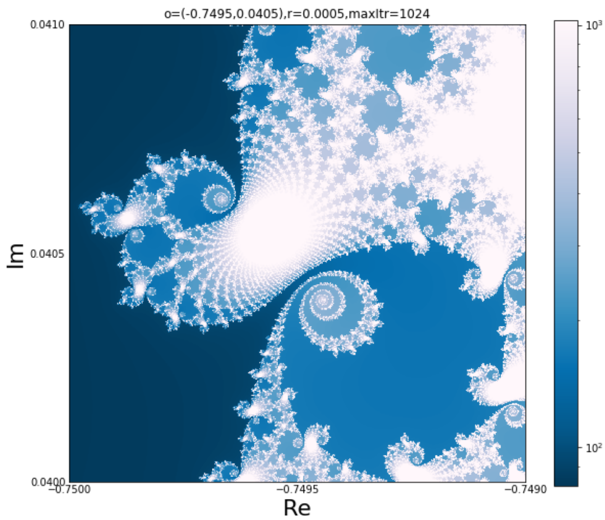

o=(0.28,0.47),r=0.02,maxItr=1024
Im
Re
0.49
0.47
0.45
0.26
0.28
0.30
10^3
10^2


o=(0.287,0.483),r=0.003,maxItr=1024
Im
Re
0.486
0.483
0.480
0.284
0.287
0.290
10^3
10^2


o=(0.28775,0.48385),r=0.00045,maxItr=1024
Im
Re
0.48430
0.48385
0.48340
0.28730
0.28775
0.28820
10^3
10^2


o=(0.2881,0.48375),r=0.0001,maxItr=1024
Im
Re
0.48385
0.48375
0.48365
0.2880
0.2881
0.2882
10^3
10^2


o=(0.28811,0.483794),r=2e-05,maxItr=1024
Im
Re
0.483814
0.483794
0.483774
0.28809
0.28811
0.28813
10^3
6×10^2
4×10^2
3×10^2
2×10^2


o=(-0.7495,0.0405),r=0.0005,maxItr=1024
Im
Re
0.0410
0.0405
0.0400
−0.7500
−0.7495
−0.7490
10^3
10^2

【別解】

ソースコード 2.4　マンデルブロー集合

```
'''
mandelbrotset.py
Draw a Mandelbrot set
Using "Escape time algorithm" from:
http://en.wikipedia.org/wiki/Mandelbrot_set#Computer_drawings
'''
from numba import jit
from tqdm import tqdm
import matplotlib.pyplot as plt

# Subset of the complex plane we are considering
x0, x1 = -2.5, 1  # x0, x1 = -0.75, -0.749
y0, y1 = -1, 1    # y0, y1 = 0.040, 0.041

@jit
def mandelbrot_set(n, max_iterations):
    image = initialize_image(n, n)
    # Generate a set of equally spaced points
    #  in the region above
    dx = (x1-x0)/(n-1)
    dy = (y1-y0)/(n-1)
    x_coords = [x0 + i*dx for i in range(n)]
    y_coords = [y0 + i*dy for i in range(n)]
    #
    for i, x in tqdm( enumerate(x_coords) ):
        for k, y in enumerate(y_coords):
            z1 = complex(0, 0)
            iteration = 0
            c = complex(x, y)
            while (abs(z1) < 2  and iteration < max_iterations):
                z1 = z1**2 + c
                iteration += 1
            image[k][i] = iteration
    return image

def initialize_image(x_p, y_p):
    image = []
    for i in range(y_p):
        x_colors = []
        for j in range(x_p):
            x_colors.append(0)
        image.append(x_colors)
    return image

def draw_mandelbrot(n, max_iterations):
    image = mandelbrot_set(n, max_iterations)
    clmap = 'hot'     # clmap = 'Greys_r' or 'PuBu_r' or 'hot' etc
    # 画素の補間 (Nearest neighbor,Bilinear,Bicubic)
    plt.imshow(image, origin='lower', extent=(x0, x1, y0, y1),
               cmap=clmap, interpolation='nearest')
    plt.savefig('mandelbrotset00.png')
    plt.show()

if __name__ == '__main__':
    n = int( input('Enter the image size nxn points (100 to 1000) : ') )
    max_iter = int( input('Enter the max number of iteration (100 to 10000) : ') )
    draw_mandelbrot(n, max_iter)
    # 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

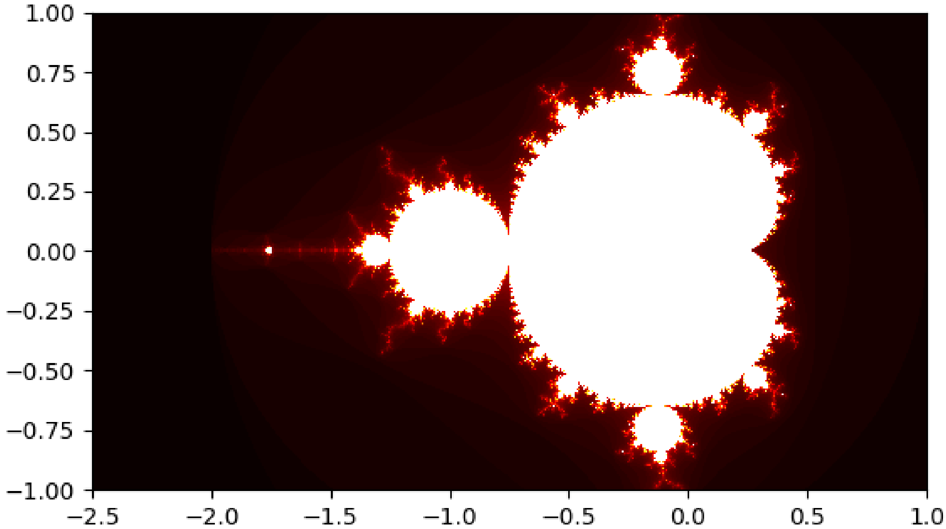


図 2.1　n=1000, max_iter=100, 02:05 elapsed

max_iter を大きくしないと、白い部分の詳細が描かれない

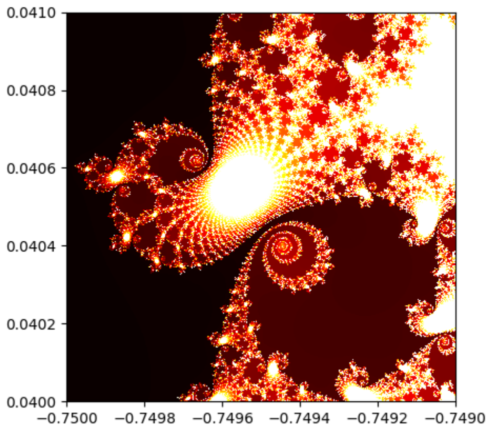


図 2.2　n=1000, max_iter=1000,19:18 elapsed

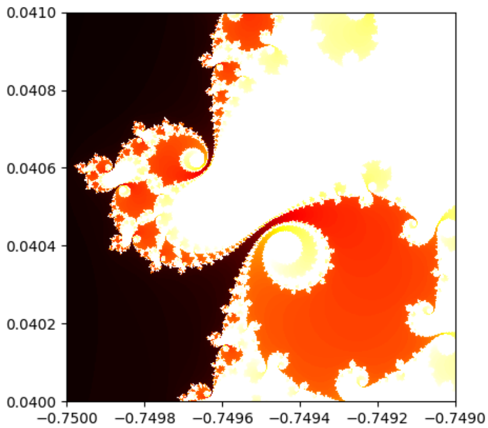


図 2.3　n=1000, max_iter=250, 15:36 elapsed

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、$c \quad (\in \mathbb{Z})$ 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

```
def Calculate(self, z0): # z0: Complex number
    z = z0
    for n in range(self.maxItr):
        az = abs(z)
        if az > self.M:
            return n
        z = z*z + self.c
    return self.maxItr
```

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

```
def Calc(self, re, im): # re, im: Real number
    z_r = re
    z_i = im
    for n in range(self.maxItr):
        zr2 = z_r**2
        zi2 = z_i**2
        if zr2 + zi2 > self.M:
            return n
        z_r_ = zr2 - zi2 + self.c_r
        z_i_ = 2*z_r*z_i + self.c_i
        z_r, z_i = z_r_, z_i_
    return self.maxItr
```

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

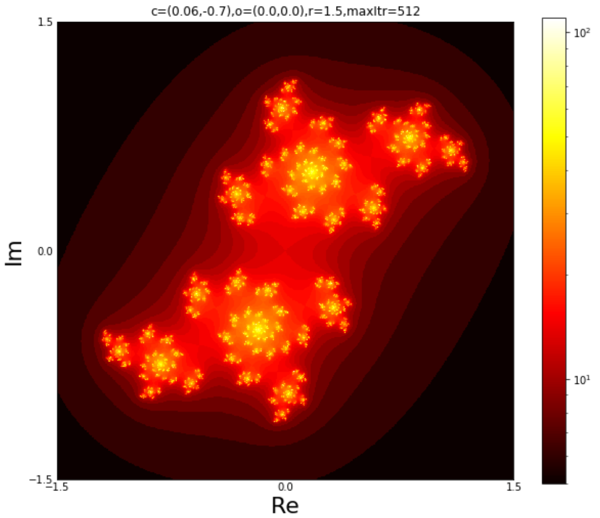
複素定数の c(プログラム中の c_r と c_i) を変えると、色々なグラフが得られる

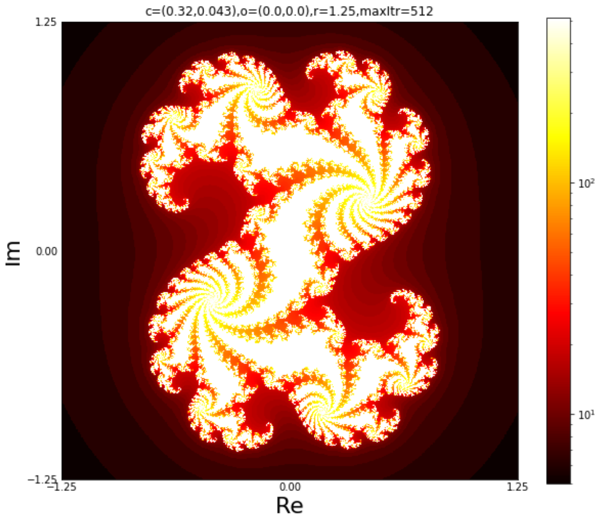
ソースコード 3.3　ジュリア集合

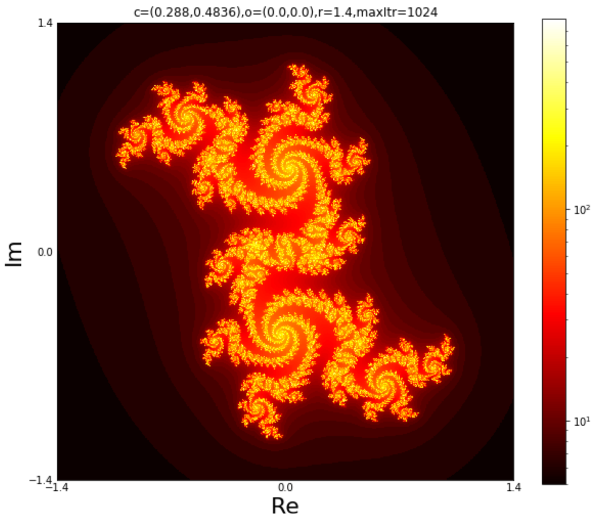
```
import numpy as np
from numba import jit
from tqdm import tqdm
import matplotlib.pyplot as plt
from matplotlib import colors
import csv
```

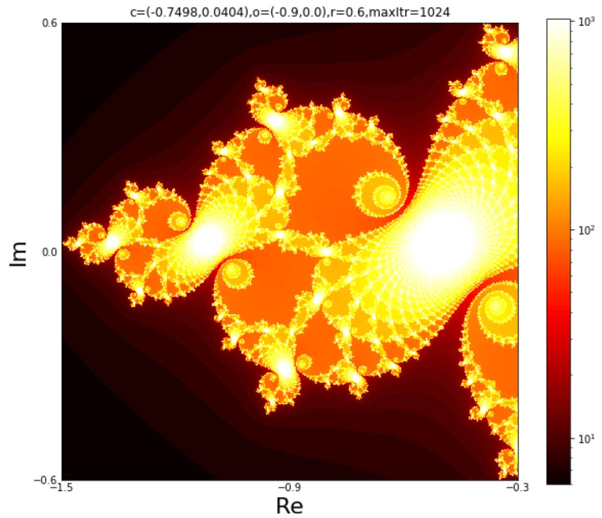
```python

class Julia:
    def __init__(self, M, maxItr, c_r, c_i):
        self.M = M
        self.maxItr = maxItr
        self.c_r = c_r
        self.c_i = c_i
        self.c = c_r + 1j*c_i

    def Calc(self, re, im): # re, im: Real number
        z_r = re
        z_i = im
        for n in range(self.maxItr):
            zr2 = z_r**2
            zi2 = z_i**2
            if zr2 + zi2 > self.M:
                return n
            z_r_ = zr2 - zi2 + self.c_r
            z_i_ = 2*z_r*z_i + self.c_i
            z_r, z_i = z_r_, z_i_
        return self.maxItr

    def Calculate(self, z0): # z0: Complex number
        z = z0
        for n in range(self.maxItr):
            az = abs(z)
            if az > self.M:
                return n
            z = z*z + self.c
        return self.maxItr

@jit
def julia_set(c_r,c_i,xmin,xmax,ymin,ymax,xpix,ypix,maxItr):
    M = 2 ** 40
    J = Julia(M, maxItr, c_r, c_i)
    re = np.linspace(xmin, xmax, xpix)
    im = np.linspace(ymin, ymax, ypix)
    z  = np.empty((xpix,ypix))
    for i in tqdm(range(xpix)):
        for j in range(ypix):
            z[j][i] = J.Calc(re[i], im[j])
            #z[j][i] = J.Calculate(re[i] + 1j*im[j])
    return re, im, z

def julia_image(c_r,c_i,x,y,radius,width=10,height=10,dpi=72,maxItr=80):
    xpix, ypix = dpi * width, dpi * height
    xmin, xmax = x - radius, x + radius
    ymin, ymax = y - radius, y + radius
    re,im,z = julia_set(c_r,c_i,xmin,xmax,ymin,ymax,xpix,ypix,maxItr)
    return re, im, np.array(z)

def julia_plot(data,geom,gamma,cmap,normz,title,figfile):
    x,y,Z = data[0], data[1], data[2]
    xmin,xmax,ymin,ymax,width,height,dpi = geom[0],geom[1],geom[2],geom[3],geom[4],
      geom[5],geom[6]
    X, Y = np.meshgrid(x, y)
    plt.rcParams['figure.figsize'] = width,height
    plt.rcParams["figure.dpi"] = dpi
    plt.gca().set_aspect('equal', adjustable='box')
    if normz=='Power':
        norm = colors.PowerNorm(gamma)
    elif normz=='Log':
        norm = colors.LogNorm(vmin=Z.min(), vmax=Z.max())
    pcm = plt.pcolor(X, Y, Z,norm=norm,cmap=cmap)
    plt.colorbar(pcm,shrink=0.82)
```

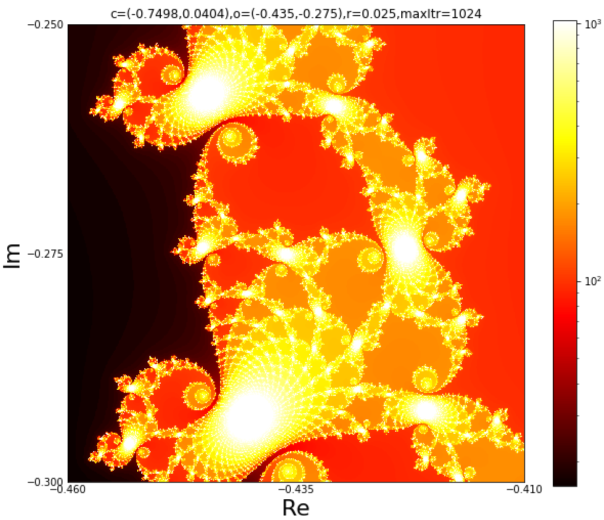
```
    plt.title(title)
    plt.xlabel('Re', fontsize=22)
    plt.ylabel('Im', fontsize=22)
    plt.xticks(np.linspace(xmin,xmax,3), color="black")
    plt.yticks(np.linspace(ymin,ymax,3), color="black")
    plt.tick_params(length=0)
    plt.savefig(figfile)
    plt.show()

if __name__ == '__main__':
    project = 'julia00A'
    csvfile, figfile,txtfile = project+'.csv', project+'.png', project+'.txt'
    u = input('(1)Calculate,(2)Display : ')
    if u=='1':
        radius = 1.4
        x_o = 0
        y_o = 0
        c_r =  0.288
        c_i =  0.4836
        maxItr = 1024
        width = 10
        height = 10
        dpi = 220
        gamma = 0.6
        x,y,Z = julia_image(c_r,c_i,x_o,y_o,radius,width,height,dpi,maxItr)
        np.savetxt(csvfile, Z)
        with open( txtfile, 'w', encoding='utf-8') as f:
            f.write('c_real,c_imag,radius,o_x,o_y,maxItr,width,height,dpi,gamma,normalize\n')
            f.write('{:f},{:f},{:f},{:f},{:f},{:d},{:d},{:d},{:d},{:f},{:s}'.\
                    format(c_r,c_i,radius,x_o,y_o,maxItr,width,height,dpi,gamma,'Power'))
    else:
        Z = np.loadtxt(csvfile)
        with open( txtfile, 'r', encoding='utf-8') as f:
            reader = csv.reader(f)
            for row in reader:
                print(row)
            c_r, c_i = float(row[0]), float(row[1])
            radius = float(row[2])
            x_o, y_o = float(row[3]), float(row[4])
            maxItr = int(row[5])
            width, height = int(row[6]), int(row[7])
            dpi = int(row[8])
            gamma = float(row[9])
            normz = row[10]
        xmin, xmax = x_o - radius, x_o + radius
        ymin, ymax = y_o - radius, y_o + radius
        x = np.linspace(xmin, xmax, dpi*width)
        y = np.linspace(ymin, ymax, dpi*height)
        cmap = 'hot'
        title = 'c=({},{}),o=({},{}),r={},maxItr={}'.format(c_r,c_i,x_o,y_o,radius,maxItr)
        normz = 'Log'
        data = (x,y,Z)
        geom = (xmin,xmax,ymin,ymax,width,height,dpi)
        julia_plot(data,geom,gamma,cmap,normz,title,figfile)
```

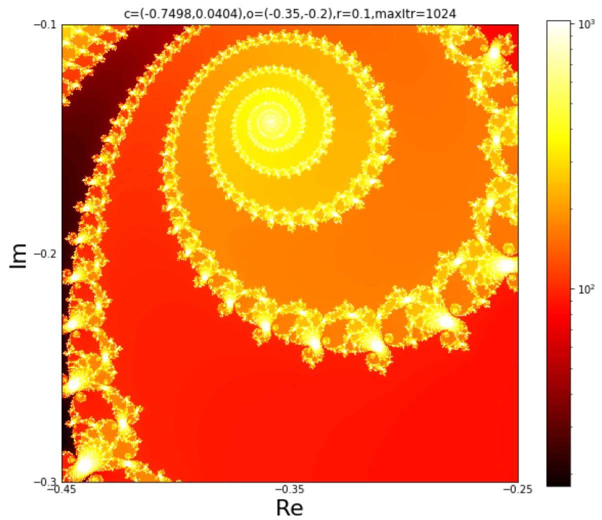

c=(0.06,-0.7),o=(0.0,0.0),r=1.5,maxItr=512
Im
Re
1.5
0.0
−1.5
1.5
10^2
10^1


c=(0.32,0.043),o=(0.0,0.0),r=1.25,maxItr=512
Im
Re
1.25
0.00
−1.25
1.25
10^2
10^1


c=(0.288,0.4836),o=(0.0,0.0),r=1.4,maxItr=1024
Im
Re
1.4
0.0
−1.4
1.4
10^2
10^1


c=(-0.7498,0.0404),o=(-0.9,0.0),r=0.6,maxItr=1024
Im
Re
0.6
0.0
−0.6
−1.5
−0.9
−0.3
10^3
10^2
10^1


c=(-0.7498,0.0404),o=(-0.435,-0.275),r=0.025,maxItr=1024
Im
Re
−0.250
−0.275
−0.300
−0.460
−0.435
−0.410
10^3
10^2

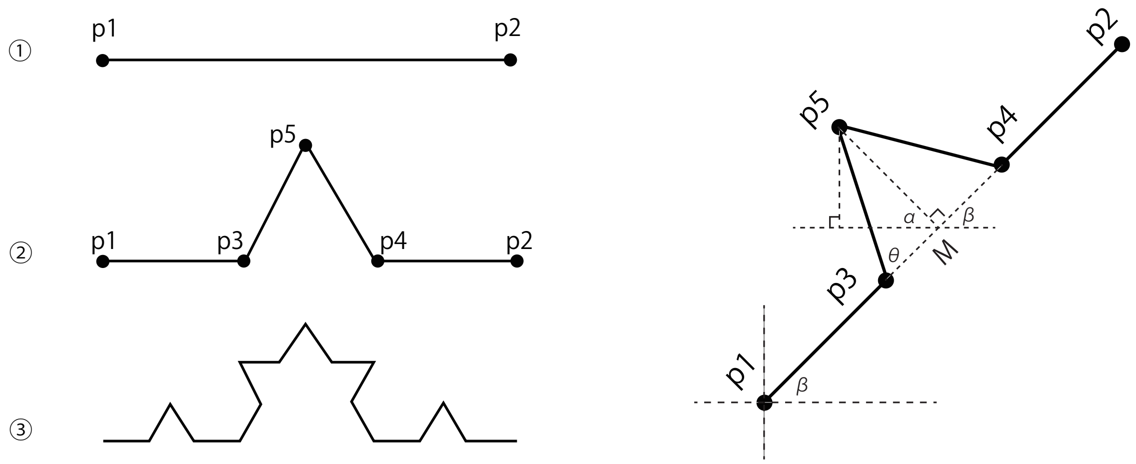

c=(-0.7498,0.0404),o=(-0.35,-0.2),r=0.1,maxItr=1024
Im
Re
−0.1
−0.2
−0.3
−0.45
−0.35
−0.25
10^3
10^2

第 4 章

再帰的処理による図形

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

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

$p_4(x_4, y_4)$ も同様に

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

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

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

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 $\theta = \pi/3$[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

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

ここで、角度 β 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 β 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、

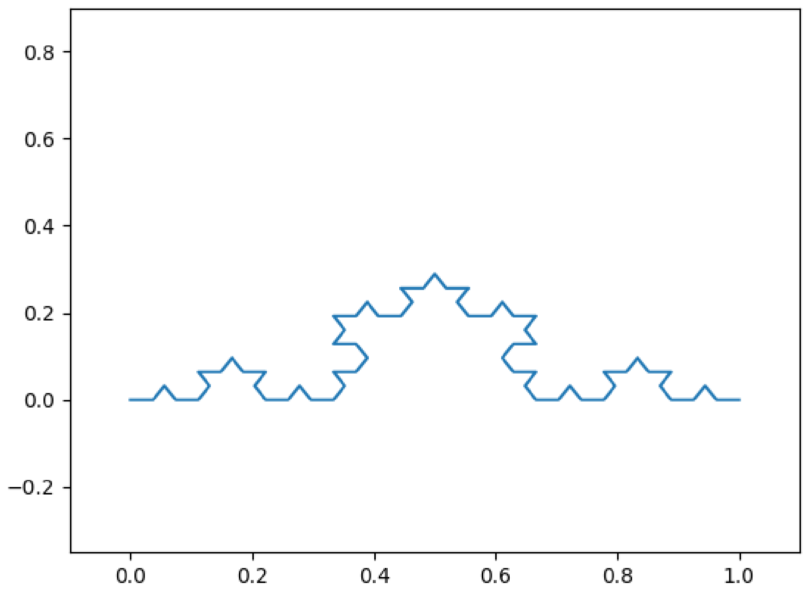
$$x_5 = x_M - h, \qquad y_5 = y_M, \qquad (\text{但し、} h = l\sin\frac{\pi}{3} = l\frac{\sqrt{3}}{2})$$

$p1$ の y 座標が $p2$ の y 座標より小さいとき、$p5$ は左へ飛び出しているが、$p1$ の y 座標が $p2$ の y 座標より大きいとき、$p5$ 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

```
import math
import matplotlib.pyplot as plt

class koch():
    def __init__( self,p1, p2, gene=0 ):
        self.result=[]
        self.gene = gene
        self.p1 = p1
        self.p2 = p2
        self.p3 = [(2*self.p1[0] + self.p2[0]) / 3,(2*self.p1[1] + self.p2[1]) / 3]
        self.p4 = [(self.p1[0] + 2*self.p2[0]) / 3,(self.p1[1] + 2*self.p2[1]) / 3]
        xm = [0,0]
        xm[0] = (self.p2[0] + self.p1[0]) / 2
        xm[1] = (self.p2[1] + self.p1[1]) / 2
        l = math.sqrt((self.p2[0]-self.p1[0])**2+(self.p2[1]-self.p1[1])**2)/3
        h = l*math.sqrt(3)/2
        if self.p2[0]==self.p1[0]:
            if self.p1[1] < self.p2[1]:
                h=-h
            self.p5 = [xm[0]-h,xm[1]]
        else:
            beta = math.atan((self.p2[1]-self.p1[1])/(self.p2[0]-self.p1[0]))
            alpha = math.pi/2-beta
            if self.p1[0] > self.p2[0]:
                self.p5 = [xm[0]+h*math.cos(alpha),xm[1]-h*math.sin(alpha)]
            else:
                self.p5 = [xm[0]-h*math.cos(alpha),xm[1]+h*math.sin(alpha)]
        self.generate()

    def generate( self ):
        if self.gene > 0:
            k1 = koch( self.p1, self.p3, self.gene-1 )
            k2 = koch( self.p3, self.p5, self.gene-1 )
            k3 = koch( self.p5, self.p4, self.gene-1 )
            k4 = koch( self.p4, self.p2, self.gene-1 )
            self.result.extend(k1.getResult())
            self.result.extend(k2.getResult())
            self.result.extend(k3.getResult())
            self.result.extend(k4.getResult())
```

```
        else:
            #self.getPrint()
            return self.result.extend( self.getCoordinates() )

    def getCoordinates( self ):
        return [self.p1, self.p3, self.p5, self.p4, self.p2]

    def getResult( self ):
        return self.result

    def getPrint(self):
        print( 'x:',self.p1[0],',y:',self.p1[1] )
        print( 'x:',self.p3[0],',y:',self.p3[1] )
        print( 'x:',self.p5[0],',y:',self.p5[1] )
        print( 'x:',self.p4[0],',y:',self.p4[1] )
        print( 'x:',self.p2[0],',y:',self.p2[1] )

if __name__ == '__main__':
    fig, ax = plt.subplots()
    ax.set_xlim(-0.1, 1.1)
    ax.set_ylim(-0.35, 0.9)

    gene=2
    x,y=[],[]

    p1, p2 = [0,0], [1,0]
    k = koch(p1, p2, gene)
    k0 = k.getResult()
    for j in range( len(k0) ):
        x.append(k0[j][0])
        y.append(k0[j][1])

    plt.plot(x,y)
    fig.savefig('figkoch1.png')
    fig.show()
```

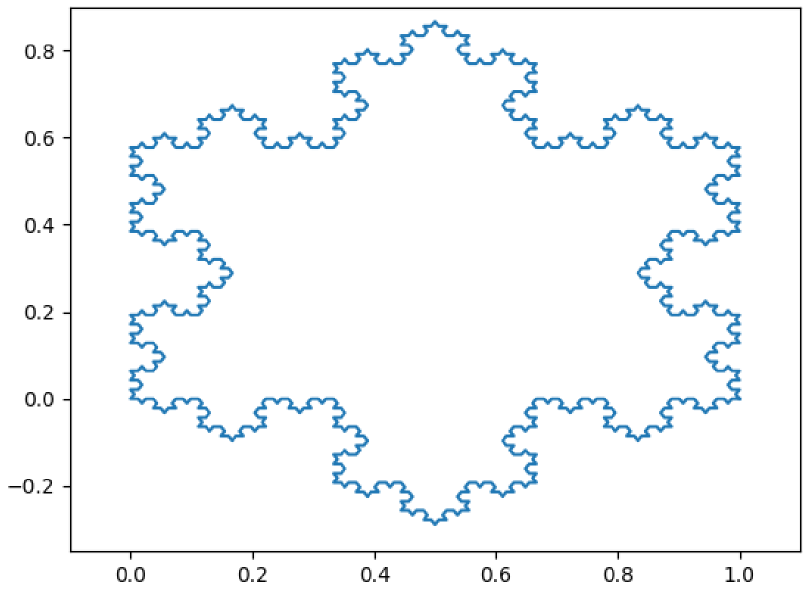


4.1.1 コッホの雪片曲線

ソースコード 4.2　コッホの雪片曲線

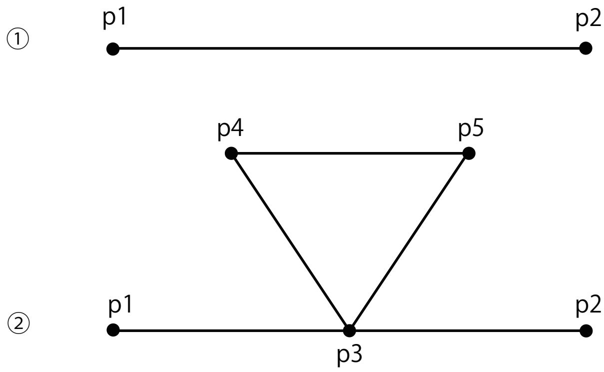
```
import math
import matplotlib.pyplot as plt

class koch():
    def __init__( self,p1, p2, gene=0 ):
        self.result=[]
        self.gene = gene
        self.p1 = p1
        self.p2 = p2
        self.p3 = [(2*self.p1[0] + self.p2[0]) / 3,(2*self.p1[1] + self.p2[1]) / 3]
        self.p4 = [(self.p1[0] + 2*self.p2[0]) / 3,(self.p1[1] + 2*self.p2[1]) / 3]
        xm = [0,0]
        xm[0] = (self.p2[0] + self.p1[0]) / 2
        xm[1] = (self.p2[1] + self.p1[1]) / 2
        l = math.sqrt((self.p2[0]-self.p1[0])**2+(self.p2[1]-self.p1[1])**2)/3
        h = l*math.sqrt(3)/2
        if self.p2[0]==self.p1[0]:
            if self.p1[1] < self.p2[1]:
                h=-h
            self.p5 = [xm[0]-h,xm[1]]
        else:
            beta = math.atan((self.p2[1]-self.p1[1])/(self.p2[0]-self.p1[0]))
            alpha = math.pi/2-beta
            if self.p1[0] > self.p2[0]:
                self.p5 = [xm[0]+h*math.cos(alpha),xm[1]-h*math.sin(alpha)]
            else:
                self.p5 = [xm[0]-h*math.cos(alpha),xm[1]+h*math.sin(alpha)]
        self.generate()

    def generate( self ):
        if self.gene > 0:
            k1 = koch( self.p1, self.p3, self.gene-1 )
            k2 = koch( self.p3, self.p5, self.gene-1 )
            k3 = koch( self.p5, self.p4, self.gene-1 )
            k4 = koch( self.p4, self.p2, self.gene-1 )
            self.result.extend(k1.getResult())
            self.result.extend(k2.getResult())
            self.result.extend(k3.getResult())
            self.result.extend(k4.getResult())
        else:
            #self.getPrint()
            return self.result.extend( self.getCoordinates() )

    def getCoordinates( self ):
        return [self.p1, self.p3, self.p5, self.p4, self.p2]

    def getResult( self ):
        return self.result

    def getPrint(self):
        print( 'x:',self.p1[0],',y:',self.p1[1] )
        print( 'x:',self.p3[0],',y:',self.p3[1] )
        print( 'x:',self.p5[0],',y:',self.p5[1] )
        print( 'x:',self.p4[0],',y:',self.p4[1] )
        print( 'x:',self.p2[0],',y:',self.p2[1] )

if __name__ == '__main__':
    fig, ax = plt.subplots()
```

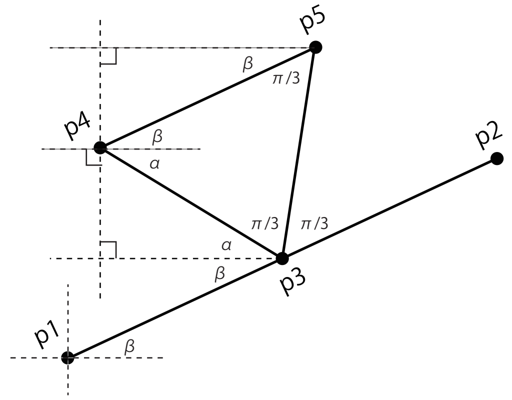
```
    ax.set_xlim(-0.1, 1.1)
    ax.set_ylim(-0.35, 0.9)

    gene=3
    x,y=[],[]

    p1, p2 = [1,0], [0,0]
    k = koch(p1, p2, gene)
    k0 = k.getResult()
    for j in range( len(k0) ):
        x.append(k0[j][0])
        y.append(k0[j][1])

    p1, p2 = [0,0], [1/2,math.sqrt(3)/2]
    k = koch(p1, p2, gene)
    k0 = k.getResult()
    for j in range( len(k0) ):
        x.append(k0[j][0])
        y.append(k0[j][1])

    p1, p2 = [1/2,math.sqrt(3)/2],[1,0]
    k = koch(p1, p2, gene)
    k0 = k.getResult()
    for j in range( len(k0) ):
        x.append(k0[j][0])
        y.append(k0[j][1])

    plt.plot(x,y)
    fig.savefig('figkoch2.png')
    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

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

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 α と辺の長さ l を使って、

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

ここで、β 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 β 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、β が $\pi/2$[rad] あるいは $3\pi/4$[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

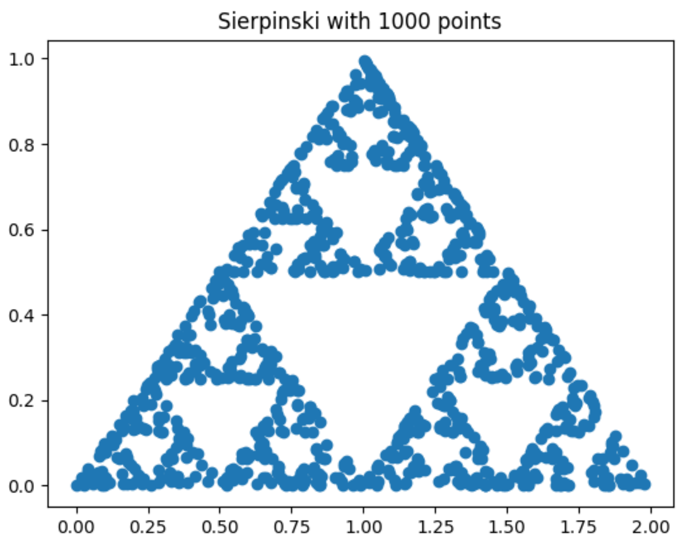
$$x_5 = x_4 + l\cos\beta, \qquad y_5 = y_4 + l\sin\beta \tag{4.6}$$

【別解】

ソースコード 4.3　シェルピンスキーの三角形

```python
'''
sierpinski.py
Draw Sierpinski Triangle
'''
import random
import matplotlib.pyplot as plt

def transformation_1(p):
    x = p[0]
    y = p[1]
    x1 = 0.5*x
    y1 = 0.5*y
    return x1, y1

def transformation_2(p):
    x = p[0]
    y = p[1]
    x1 = 0.5*x + 0.5
    y1 = 0.5*y + 0.5
    return x1, y1

def transformation_3(p):
    x = p[0]
    y = p[1]
    x1 = 0.5*x + 1
    y1 = 0.5*y
    return x1, y1

def get_index(probability):
    r = random.random()
    c_probability = 0
    sum_probability = []
    for p in probability:
        c_probability += p
        sum_probability.append(c_probability)
    for item, sp in enumerate(sum_probability):
        if r <= sp:
            return item
    return len(probability)-1

def transform(p):
    # list of transformation functions
    transformations = [transformation_1, transformation_2, transformation_3]
    probability = [1/3, 1/3, 1/3]
    # pick a random transformation function and call it
    tindex = get_index(probability)
    t = transformations[tindex]
    x, y = t(p)
    return x, y

def draw_sierpinski(n):
    # We start with (0, 0)
    x = [0]
    y = [0]
    x1, y1 = 0, 0
    for i in range(n):
       x1, y1 = transform((x1, y1))
       x.append(x1)
       y.append(y1)
    return x, y

```

```python
if __name__ == '__main__':
    n = int( input('Enter␣the␣number␣of␣points␣in␣the␣Sierpinski:␣') )
    x, y = draw_sierpinski( n )
    # Plot the points
    plt.plot(x, y, 'o')
    plt.title('Sierpinski␣with␣{0}␣points'.format(n))
    plt.savefig('sierpinski.png')
    plt.show()
```



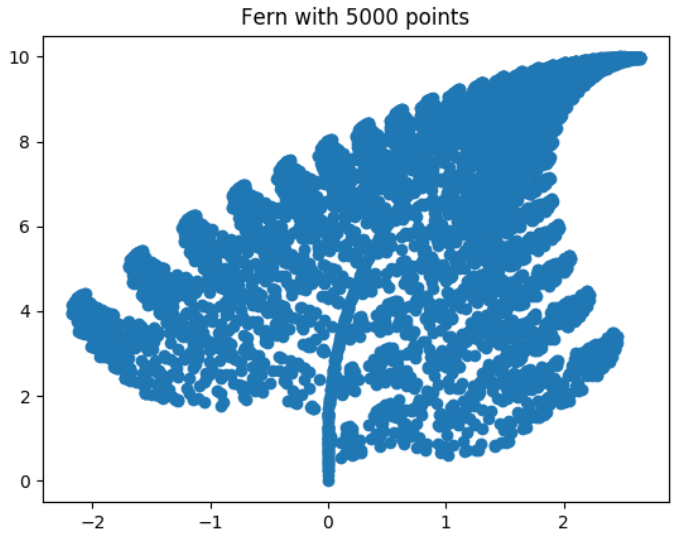
4.3　バーンスレイのシダ

Michael Barnsley

ソースコード 4.4　バーンスレイのシダ

```python
import random
import matplotlib.pyplot as plt

def transformation_1( p ):
    x = p[0]
    y = p[1]
    x1 = 0.85*x + 0.04*y
    y1 = -0.04*x + 0.85*y + 1.6
    return x1, y1

def transformation_2( p ):
    x = p[0]
    y = p[1]
    x1 = 0.2*x - 0.26*y
    y1 = 0.23*x + 0.22*y + 1.6
    return x1, y1

def transformation_3( p ):
    x = p[0]
    y = p[1]
    x1 = -0.15*x + 0.28*y
    y1 = 0.26*x + 0.24*y + 0.44
    return x1, y1
```

```

def transformation_4( p ):
    x = p[0]
    y = p[1]
    x1 = 0
    y1 = 0.16*y
    return x1, y1

def get_index(probability):
    r = random.random()
    c_probability = 0
    sum_probability = []
    for p in probability:
        c_probability += p
        sum_probability.append( c_probability )
    for item, sp in enumerate( sum_probability ):
        if r <= sp:
            return item
    return len( probability ) - 1

def transform( p ):
    # list of transformation function 変換関数のリスト
    transformations = [ transformation_1, transformation_2, transformation_3, transformation_4 ]
    probability = [0.85, 0.07, 0.07, 0.01]
    # pick a random transformation function and call it ランダム変換関数を呼ぶ
    tindex = get_index( probability )
    t = transformations[tindex]
    x,y = t( p )
    return x, y

def draw_fern( n ):
    # We start with (0, 0) (0,0)から始める
    x = [0]
    y = [0]
    x1, y1 = 0, 0
    for i in range( n ):
        x1, y1 = transform( (x1, y1) )
        x.append( x1 )
        y.append( y1 )
    return x, y

if __name__ == '__main__':
    n = int( input('Enter the number of points in the Fern: ') )
    x, y = draw_fern( n )
    # Plot the points
    plt.plot( x, y, 'o' )
    plt.title('Fern with {0} points'.format(n))
    plt.savefig('fern.png')
    plt.show()
```


Fern with 5000 points
10
8
6
4
2
0
−2
−1
0
1
2

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

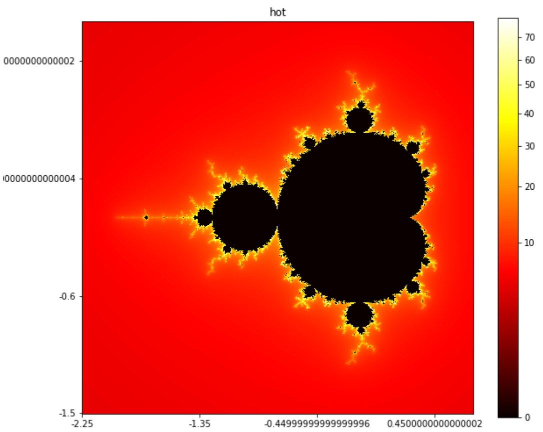
ソースコード 5.1　参考記事

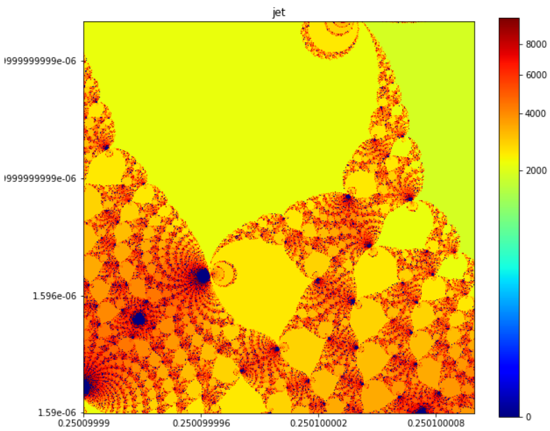
```
# https://nepia01.blogspot.com/2017/07/python.html
import numpy as np
from numba import jit
from tqdm import tqdm

@jit
def mandelbrot(z,maxiter,horizon,log_horizon):
    c = z
    for n in range(maxiter):
        az = abs(z)
        if az > horizon:
            return n - np.log(np.log(az))/np.log(2) + log_horizon
        z = z*z + c
    return 0

@jit
def mandelbrot_set(xmin,xmax,ymin,ymax,width,height,maxiter):
    horizon = 2 ** 40
    log_horizon = np.log(np.log(horizon))/np.log(2)
    r1 = np.linspace(xmin, xmax, width)
    r2 = np.linspace(ymin, ymax, height)
    n3 = np.empty((width,height))
    for i in tqdm(range(width)):
        for j in range(height):
            n3[i,j] = mandelbrot(r1[i] + 1j*r2[j],maxiter,horizon, log_horizon)
    return (r1,r2,n3)

from matplotlib import pyplot as plt
from matplotlib import colors

def mandelbrot_image(x,y,radius,width=10,height=10,maxiter=80,cmap='jet',gamma=0.3):
    dpi = 72
    img_width = dpi * width
    img_height = dpi * height
    xmin = x - radius
    xmax = x + radius
    ymin = y - radius
    ymax = y + radius
    x,y,z = mandelbrot_set(xmin,xmax,ymin,ymax,img_width,img_height,maxiter)

    fig, ax = plt.subplots(figsize=(width, height),dpi=72)
    ticks = np.arange(0,img_width,3*dpi)
```

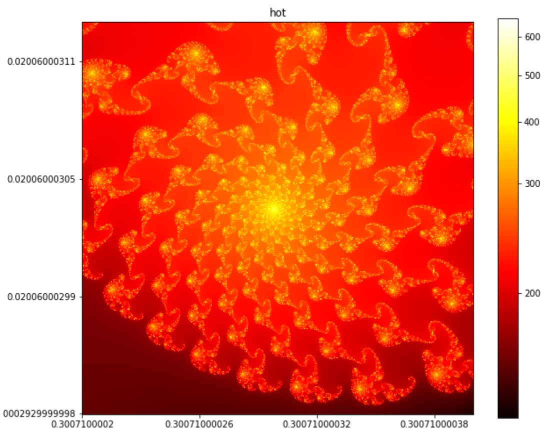
```
    x_ticks = xmin + (xmax-xmin)*ticks/img_width
    plt.xticks(ticks, x_ticks)
    y_ticks = ymin + (ymax-ymin)*ticks/img_width
    plt.yticks(ticks, y_ticks)
    ax.set_title(cmap)

    norm = colors.PowerNorm(gamma)
    cax = ax.imshow(z.T,cmap=cmap,origin='lower',norm=norm)
    fig.colorbar(cax, shrink=0.82)
    plt.savefig('fig000.png')
    plt.show()

mandelbrot_image(-0.75, 0, 1.5, cmap='hot', gamma=0.4)
#mandelbrot_image(0.2501,1.6e-6,1e-8,maxiter=10000)
#mandelbrot_image(0.3007100003,0.02006000303,1e-10,maxiter=2048,cmap='hot',gamma=0.4)
#mandelbrot_image(-0.74958245,0.0300396,1e-6,maxiter=30000,cmap='copper')
#mandelbrot_image(-0.74997501,-0.00999551,1e-8,maxiter=100000,cmap='flag',gamma=0.98)
```

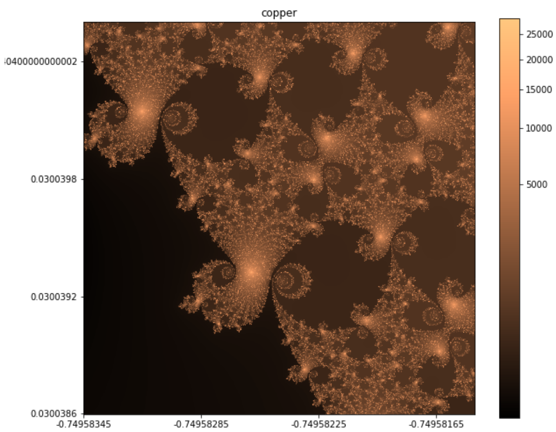
cmap='hogehoge' で指定する hogehoge は、以下に一覧がある

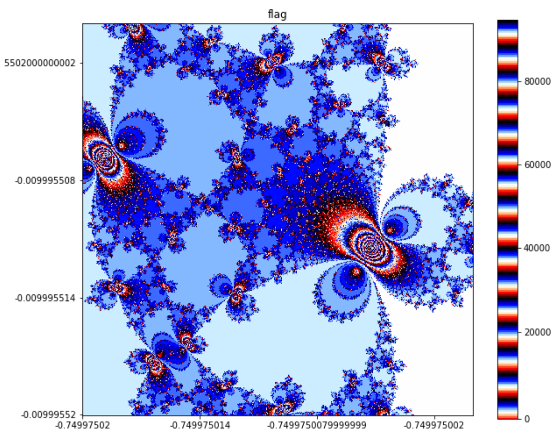
https://matplotlib.org/examples/color/colormaps_reference.html


hot
0000000000002
0000000000004
-0.6
-1.5
-2.25
-1.35
-0.44999999999999996
0.4500000000000002
70
60
50
40
30
20
10
0


jet
9999999999e-06
9999999999e-06
1.596e-06
1.59e-06
0.25009999
0.250099996
0.250100002
0.250100008
8000
6000
4000
2000
0


hot
0.02006000311
0.02006000305
0.02006000299
0002929999998
0.3007100002
0.30071000026
0.30071000032
0.30071000038
600
500
400
300
200


copper
0400000000002
0.0300398
0.0300392
0.0300386
-0.74958345
-0.74958285
-0.74958225
-0.74958165
25000
20000
15000
10000
5000


flag
5502000000002
-0.009995508
-0.009995514
-0.00999552
-0.74997502
-0.749975014
-0.7499750079999999
-0.749975002
80000
60000
40000
20000
0

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 からはじめる数学入門』初版（オライリー・ジャパン 2016）
[7] 奥村晴彦，黒木裕介『LaTeX 2_ε 美文書作成入門』第 7 版（技術評論社，2017）