Fractals

Jordan Moore, Kyle Wedgwood

June 3, 2019

1 Measuring the Coastline of The UK

If you were asked to measure the length of the coastline of the UK, and the land area it contained, how would you do it?

It turns out that this is a problem that mathematicians have only really begun working on in the last 50 years. Imagine you have at your disposal a set of measuring sticks and laid them out end to end as illustrated below in figure 1.

(a) Coastal measurement at a 200 km resolution. Approx. length = $2,400\ km$

(b) Coastal measurement at a 100 km resolution. Approx. length = $2,800\ km$

(c) Coastal measurement at a 50 km resolution. Approx. length = $3,400\ km$

Figure 1: Measurements of the coastline of the UK at increasing resolution (source: Wikimedia Commons).

It turns out that whilst the area of the UK converges (to approx $242,495\ km^2$), the length of coastline does not converge, and is dependent on the length of the measuring stick used to approximate it. Benoit Mandelbrot, in 1967, uncovered a branch of mathematics, called fractals, to describe this phenomena.

2 What is a Fractal

Although a relatively new branch of mathematics, fractals have already found uses in highly diverse numbers of fields, from fashion design, to seismology. There is no concise definition of what a fractal is, so we will define it with a set of properties, by way of example of a fractal called the Cantor set, shown in Figure 2. The set can be defined by the following algorithm:

1. We begin by drawing a line between zero and one on a number line (shown at line zero below).

2. For each subsequent row, we copy the row above, whilst deleting the middle third from each line segment.

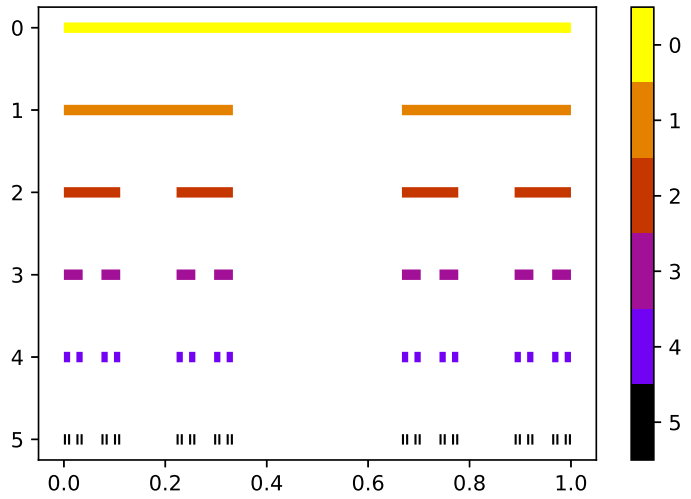


Figure 2: This is the most basic fractal, the Cantor set. Each subsequent line set is generated by removing the middle third from the set of lines above. Created in Python3.

The Cantor set exhibits the following features, common to many fractals:

1. Each element in every row of the Cantor set is self-similar to other elements within the set. Simply put, if you zoom in on a particular area of the Cantor set, you get another Cantor set.

2. The set has a 'fine structure'; that is, it contains detail at arbitrarily small scales. The more we enlarge the picture of the Cantor set, the more gaps become apparent to the eye.

3. Although the set has a detailed structure, defining it is a simple process.

4. The Cantor set is recursively defined. That is, the construction of every row relies on the previous row (aside from the initial row).

5. The global geometry of the Cantor set is not easy to define in traditional manners; it is neither the locus of points that satisfy some simple geometric condition nor is it the set of solutions of any simple equation.

6. It is awkward to describe the local geometry of the set. Near each of the lines are a large number of other lines, separated by gaps of varying lengths.

The above has been adapted from Fractal Geometry by Kenneth Falconer (ISBN-10: 0471957240). Fractals can get intricate and highly aesthetically pleasing, such as the following example of a Julia fractal:

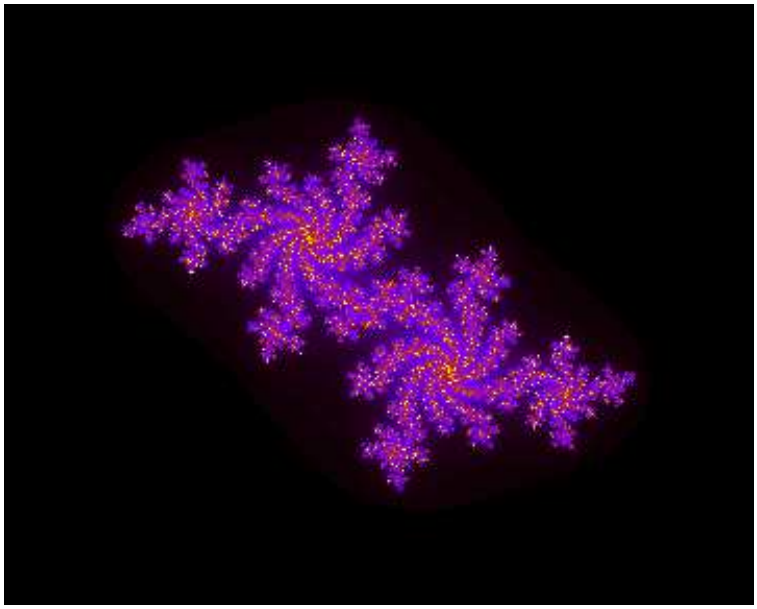
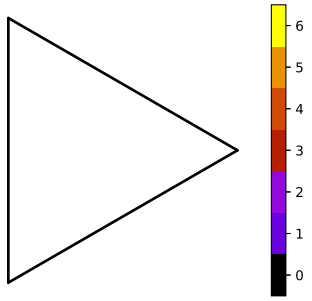


Figure 3: This is an example of a Julia fractal, generated using Python3. Note how this satisfies the properties above.

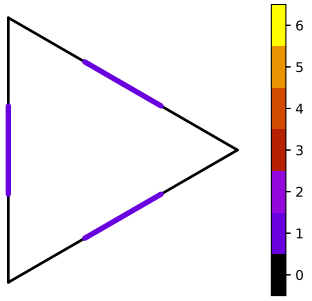
3 The Koch Snowflake

Discovered in 1904 by Swedish mathematician Helge von Koch, The Koch snowflake was one of the earliest fractals to be discovered. The algorithm to generate it is as follows:

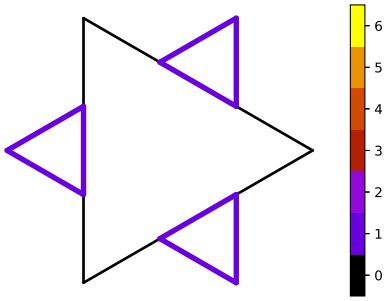
1. Draw an equilateral triangle



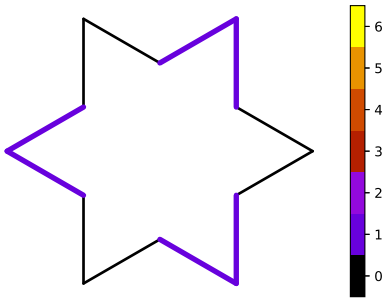
2. Split every line into three equal components.



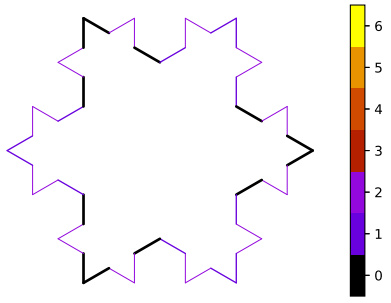
3. Draw an equilateral triangle on every line, using the middle third of line from step 2 as a base, pointing outwards



4. Remove the line segment that is the base of the triangle in step 3



5. Recursively repeat steps 2-4 to generate Koch snowflakes of increasing order.



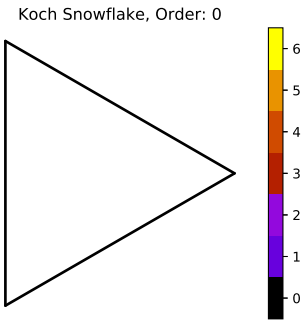
A graphical output of the first 7 Koch snowflakes is provided below in figure 5, and the Python code that was used to generate this is provided at the end of this document.

3.1 Complexity of a Fractal

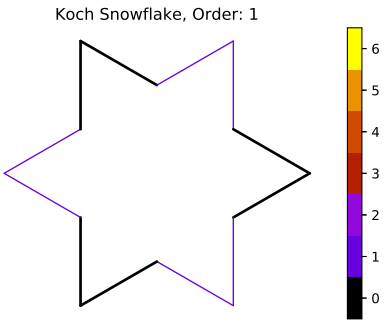
One way that we can measure a fractal is to consider the rate at which it becomes more intricate. We do this via a measure called the fractal dimension,

$$D = -\frac{\ln N}{\ln s}. \tag{1}$$

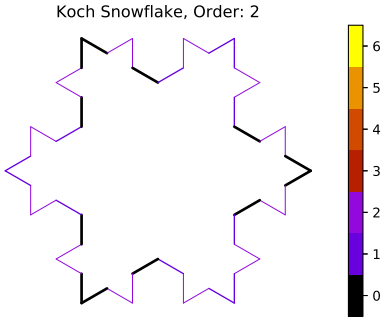
Here, ln is the natural log, N is the factor by which the number of lines increases and s is the factor by which the line length is scaled down. An informal way of thinking about a fractal dimension is that the closer to one, the more one can describe the object in terms of a 1 dimensional shape.



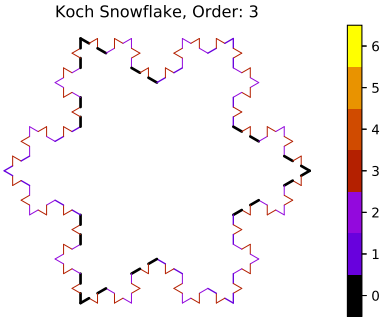
(a) 0^{th} Order Koch snowflake



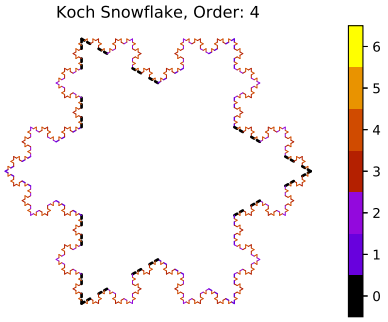
(b) 1^{st} Order Koch snowflake



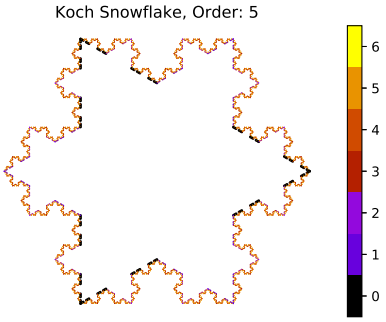
(c) 2^{nd} Order Koch snowflake



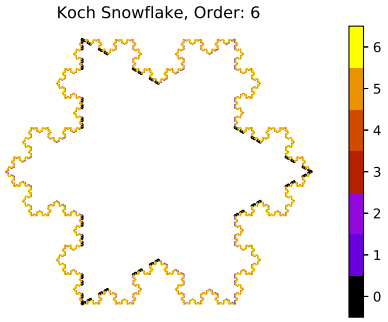
(d) 3^{rd} Order Koch snowflake



(e) 4^{th} Order Koch snowflake



(f) 5^{th} Order Koch snowflake



(g) 6^{th} Order Koch snowflake

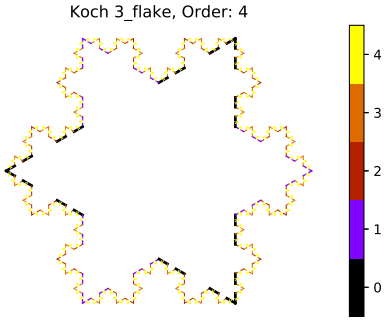
Figure 5: The above images shows Koch snowflakes in incremental order

4 Koch M-Flake

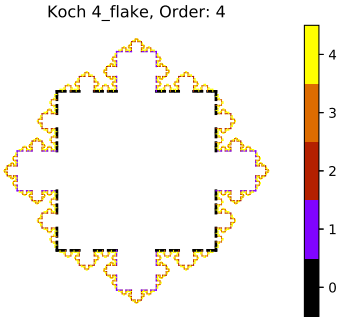
We now extend upon the Koch snowflake. For $M \geq 3 \in \mathbb{Z}$, we can define the Koch M-flake as follows.

1. Draw a regular M sided polygon
2. Upon each line, identify the segment between $\frac{M-1}{2M}$ and $\frac{M+1}{2M}$
3. Draw a regular M sided polygon on every line, using the middle M^{th} of line, identified in step 2 as a base, pointing outwards
4. Remove the line segment that is the base of the polygon in step 3
5. Recursively repeat steps 2-4 to generate Koch M-flake of increasing order.

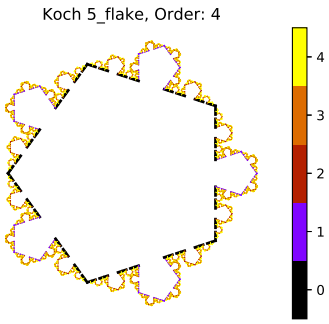
Below are provided the Koch M-flakes for M between 3 and 7 at order 4.



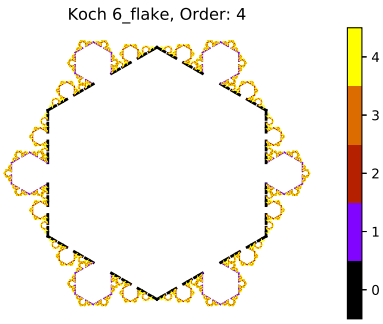
(a) 4th Order Koch 3-flake



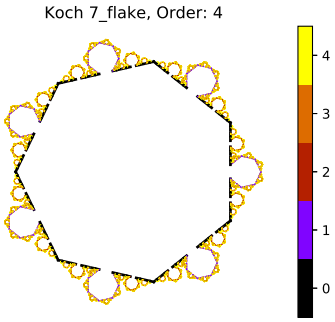
(b) 4th Order Koch 4-flake



(c) 4th Order Koch 5-flake



(d) 4th Order Koch 6-flake



(e) 4th Order Koch 7-flake

Figure 6: The above images shows Koch m-flakes in incremental order of m

5 Questions

5.1 Closed Questions

1. What is the fractal dimension of the Koch snowflake
2. As we increase the order of the Koch snowflake towards infinite, what happens to the perimeter and area of the snowflake?
3. Same as 1 and 2, but for the Koch M-flake
4. What would the final area of the Koch ∞-flake be?

For an example on how to construct a proof, here is an example of one to show that $\sqrt{2}$ is an irrational number: `https://www.math.utah.edu/~pa/math/q1.html`. Here is another to show that there are infinitely many prime numbers: `https://www.math.utah.edu/~pa/math/q2.html`. Feel free to use these as guides/templates on how you should contruct your own proofs.

5.2 Open Ended Extensions

Feel free to extend these questions in whatever way you see fit. Below are a couple of suggestions, if you're struggling for ideas.

- Using the code provided below for the Koch Snowflake, or otherwise, can you write your own piece of code to generate a Koch M-flake?
- Investigate the Sierpinski triangle and Sierpinkiski carpet in a similar fashion to the Koch snowflake and produce images of them. See e.g. `http://mathworld.wolfram.com/SierpinskiCarpet.html` and `http://mathworld.wolfram.com/SierpinskiSieve.html`.
- Do some reading and review Mandelbrot and/or Julia fractals and produce a short review of them.
- Can you write some code to produce images for the Julia/Mandelbrot set, such as that produced in the opening example.
- See if you can produce some fractal images in 3D. For artistic inspiration, see `http://www.mandelbulb.com/`

6 Koch Snowflake Python Code

```
#!/usr/bin/env python3

##import matplotlib for plotting
import matplotlib as mpl
import matplotlib.pyplot as plt
from matplotlib import rc
##import numpy for linear algebra
import numpy as np
import math
ext='.eps'



##Couple of property constants. prop1 is the constant about which the triangles are inserted, prop2 is the perpendicular distance from the line for the peak of the triangle
prop1=1/3
prop2=math.sqrt(3)/6

##Set a max iteration for the snowflake
Maxiterations=7


##Get a plotting colourmap
cmap = plt.get_cmap('gnuplot',Maxiterations)


##Create a class object for each line, storing within the points about which to add new triangles
class Line:
    ##Class only needs to create a constructor as that will generate all the points
    def __init__(self,Startpoint,Endpoint,Depth):
        ##Find X and Y distances
        Lx=Endpoint[0]-Startpoint[0]
        Ly=Endpoint[1]-Startpoint[1]
        ##Calculate vector, corresponding to a third of the way up the line
        Dist=[(Lx)*prop1,(Ly)*prop1]
        ##Calculate transpose vector, corresponding to the vector where the point in the iterated fractal lies
        DistT=[-(Ly)*prop2,(Lx)*prop2]
        ##Calculate and store the 5 points as class members for the next iteration of the fractal
        self.stp=Startpoint
        self.edp=Endpoint
        self.p1=[x + y for x, y in zip(Startpoint, Dist)]
        self.p3=[x - y for x, y in zip(Endpoint, Dist)]
        self.p2=[(x + y)*0.5 + z for x, y, z in zip(Startpoint, Endpoint, DistT)]
        ##Depth corresponds to the iteration that the fractal was generated in
        self.colour=Depth

##Function to plot lines out from the list of l
def plot(LineList,ax):
    for Li in LineList:
        ax.plot([Li.stp[0],Li.edp[0]],[Li.stp[1],Li.edp[1]],c=cmap(Li.colour),linewidth=2/(Li.colour+1),solid_capstyle='round')



##Function to update list at each iteration.
def ListUpdate(LSti,i):
    LSt=[]
    for j in LSti:
        LSt.append(Line(j.stp,j.p1,j.colour))
        LSt.append(Line(j.p1,j.p2,i))
        LSt.append(Line(j.p2,j.p3,i))
        LSt.append(Line(j.p3,j.edp,j.colour))
    return LSt


##Create empty list
LSt=[]
##Create the 0th iteration corresponding to an equilateral triangle
LSt.append(Line([0,0],[0,10],0))
LSt.append(Line([0,10],[5*math.sqrt(3),5],0))
```

```
LSt.append(Line([5*math.sqrt(3),5],[0,0],0))


##Set some plotting variables in matplotlib, for the colourmap
norm = mpl.colors.Normalize(vmin=0,vmax=Maxiterations)
sm = plt.cm.ScalarMappable(cmap=cmap, norm=norm)
sm.set_array([])
##Loop through the iterations
for i in range(0,Maxiterations):
    ##Upadate the linelist if not on the 0th iterartion
    if i !=0:
        LSt=ListUpdate(LSt,i)
    ##Plot figure
    fig = plt.figure()
    ax1 = fig.add_subplot(1,1,1, adjustable='box', aspect=1)
    ##Call out plot function for the linelist at a particular iteration
    plot(LSt,ax1)
    ax1.set_title(r'Koch Snowflake, Order: %s' %(str(i)))
    ##Turn off axis as an image
    ax1.set_axis_off()
    ##Create colourbar
    cax=plt.colorbar(sm, ticks=range(0,Maxiterations),boundaries=np.arange(-0.5,Maxiterations
    +0.5,1))
    ##Save figure and close
    plt.savefig('Image_Out/Snowflake_Ord_'+str(i)+ext, interpolation='none')
    plt.close()
```