Chaos Game

Nickolas Arustamyan

September 8, 2020

Abstract

In this experiment, we studied how out of chaos can come order. Using simple mathematical operations and random numbers we were able to generate complex and fractal-like images. Two of these included Serpenki's Triangle and a Fern

1 Introduction

This system has applications within computer graphics and image generation as a method to create shapes and images through relatively simple code.

2 Theory

Here give a brief summary of the physical effect of interest and provide necessary equations. Here is how you insert an equation. According to references [?,?,?] the dependence of interest is given by

$$x_{n+1} = rx_n(1 - x_n) \tag{1}$$

Don't forget to explain what each variable means the first time that you introduce it.

3 Procedures

The code used to generate the triangle and the fern are shown below.

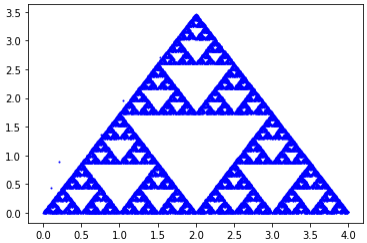
```
def midpoint(P, Q):
    return (0.5*(P[0] + Q[0]), 0.5*(P[1] + Q[1]))
#avg of the inputs and the outpus

vertices = [(0, 0), (2,2*np.sqrt(3)), (4, 0)]
n = 25000 # Change this value and see what happens
#creates a triangle w the given verticies

x = [0]*n #creates two lists with n slots and fills the first slots with random numbers
y = [0]*n
x[0] = random()
y[0] = random()

for i in range(1, n):
    x[i], y[i] = midpoint( vertices[randint(0, 2)], (x[i-1], y[i-1])) #fills the rest of the list with the values obtained from the midpoints
plt.scatter(x, y, color = 'b', s=1) #plots
```

This code uses the midpoint formula and random numbers to plot points. These points eventually end up displaying the triangle below when the number of points is increased.



Part 2

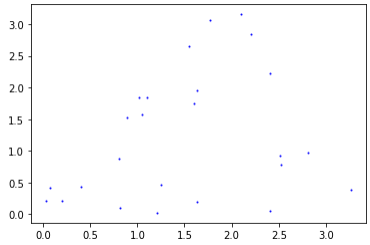
```
[88] # Barnsley's Fern
     # 1% of the time:x → 0,y → 0.16 y
     # 85% of the time:x → 0.85 x + 0.04 y, y → -0.04 x + 0.05 y + 1.6
     # 7% of the time:x → 0.2 x - 0.26 y, y → 0.23 x + 0.22 y + 1.6
     # 7% of the time:x → -0.15 x + 0.28 y, y → 0.26 x + 0.24 y + 0.44
     def pick(p):
         c = np.cumsum(p)
         return bisect(c, np.random.random() * c[-1]) #randomly bisects the matrix
     p = np.array([0.01,0.07,0.07,0.85])
     eq = [np.array([[0,0,0],[0,0.16,0]]),
           np.array([[0.2,-0.26,0],[0.23,0.22,1.6]]),
           np.array([[-0.15, 0.28, 0],[0.26,0.24,0.44]]), #defines matrix
           np.array([[0.85, 0.04, 0],[-0.04, 0.85, 1.6]])]
     n = 25000 # Change this value and see what happens
     x = np.zeros((n,3))
     x[:,2] = 1  #makes an n by 3 matrix of 0's and then changes the last column to 1's
     for i in range(1,n):
         x[i,:2] = np.matmul(eq[pick(p)],x[i-1,:]) #multiplies the two matricies and make the last column of x the resulting values
     plt.figure(figsize=(10,10))
     plt.scatter(x[:,0], x[:, 1], s=3, c="g", marker="s", linewidths=0)
     plt.axis("equal"),plt.axis("off"); #plots the values
     #Woah its a fern!
```

This code bisects a somewhat random matrix with certain cut offs and multiplies them together. After plotting the resulting values, one can see a figure that resembles a fern more and more when the number of points is increased.

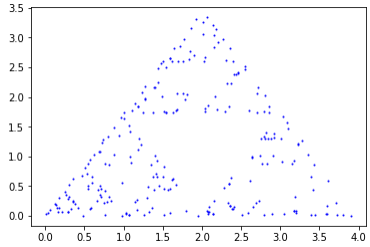
4 Results

With a growing n, the amount that the figure resembles the desired result grows. This is demonstrated below.

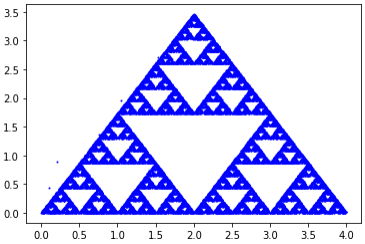
n = **25**



n = **250**

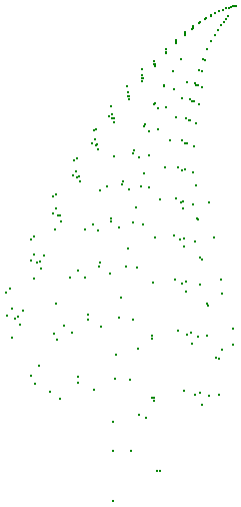


n = **25000**



n= **25**

n = **250**



n = **25000**

5 Analysis

One can see from the results that with an increasing n, the figure looks more and more orderly and less chaotic and random. This might be attributed to the fact that with only a few points, there is normally not enough data to actually visualize and understand the pattern.

6 Conclusions

In conclusion, it is clear that even within randomness there exists some order that can be studied and utilized. Tweaking the parameters of the code can results in massive changes

into the output, which can be very useful for image generation.