Lab 2: Chaos Game

Lucas Rodrigues

September 8, 2020

Abstract

In this experiment we studied how changing the number of points and iterations we can vastly change how a resulting plot looks. We further experimented with repeated patterns that generate fractals, and inspected implemented code to generate the famous Sierpinski's Triangle and Barnsley's Fern.

1 Introduction

The very important physical effect has applications to astronomy, nuclear physics, condensed matter, and more...

2 Theory

We can generate Sierpinski's Triangle by drawing a triangle, connecting the midpoints, and repeating the process for each small triangle. The code uses this method, only in a random way bounded by the triangle generated from connecting the midpoints of each leg. Eventually, for large n, the system converges to Sierpinski's Triangle. Similarly, the fern is generated by randomly choosing points, applying the Barnsley's Fern algorythm while generating a bound to the system. Eventually the system approaches Barnsley's Fern.

3 Procedures

To generate the triangle, we use the following code.

```
def midpoint(P, Q):
    return (0.5*(P[0] + Q[0]), 0.5*(P[1] + Q[1])) #take midpoint by averaging
vertices = [(0, 0), (2, 2*np.sqrt(3)), (4, 0)]
n = 25000 # Change this value and see what          happens. Number of
x = [0]*n
y = [0]*n
x[0] = random()
y[0] = random()
#two lists with random n points are created
```

```
for i in range(1, n):
    x[i], y[i] = midpoint( vertices[randint(0, 2)], (x[i-1], y[i-1]) ) #Midpo
plt.scatter(x, y, color = 'b', s=1)
```

To generate the fern we use the following:

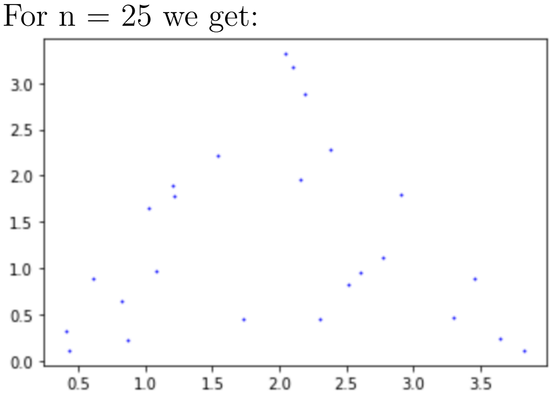
```
def pick(p):
    c = np.cumsum(p)
    return bisect(c, np.random.random() * c[-1]) #matrix is randomly bisected
p = np.array([0.01,0.07,0.07,0.85]) #percentage of the time
eq = [np.array([[0,0,0],[0,0.16,0]]),
      np.array([[0.2,-0.26,0],[0.23,0.22,1.6]]),
      np.array([[-0.15, 0.28, 0],[0.26,0.24,0.44]]),
      np.array([[0.85, 0.04, 0],[-0.04, 0.85, 1.6]])]
      #creates matrix
n = 25000 # Change this value and see what happens. Edge number
x = np.zeros((n,3))
x[:,2] = 1
#changes all 2nd column values to 1 for the nx3 matrix
for i in range(1,n):
    x[i,:2] = np.matmul(eq[pick(p)],x[i-1,:]) #values are multipled and assig
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 fern
```

4 Analysis

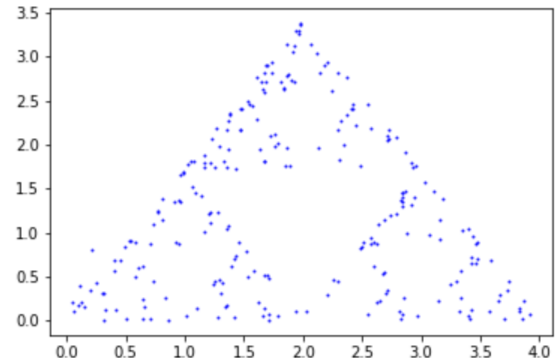
We observe that for an increasing number of n, our systems approaches the Sierpinski's Triangle and Barnsley's Fern.

Triangle

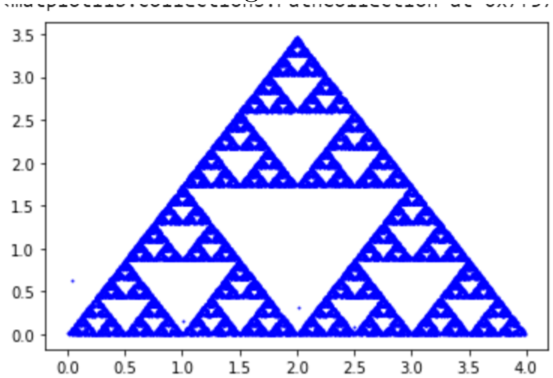
For n = 25 we get:



For n = 250 we get:



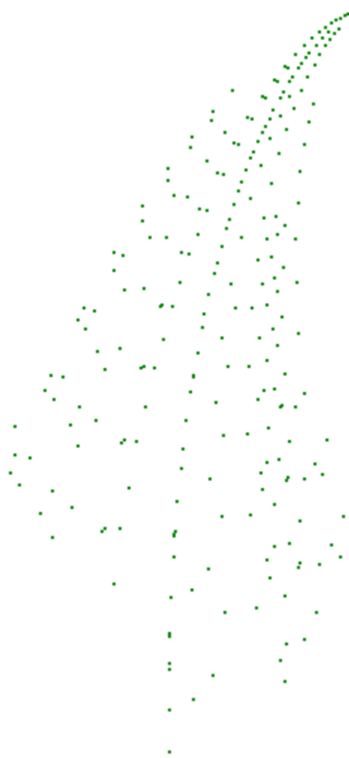
For n = 25000 we get:



Fern

For n = 25 we get:

For n = 250 we get:



For n = 25000 we get:

5 Conclusions

In conclusion, bounded randomness can generate very interesting images when iterated with itself, which changes and becomes more accurate as the number of iterations increase.