

Chaos Game

Neo Nguyen

September 8, 2020

Abstract

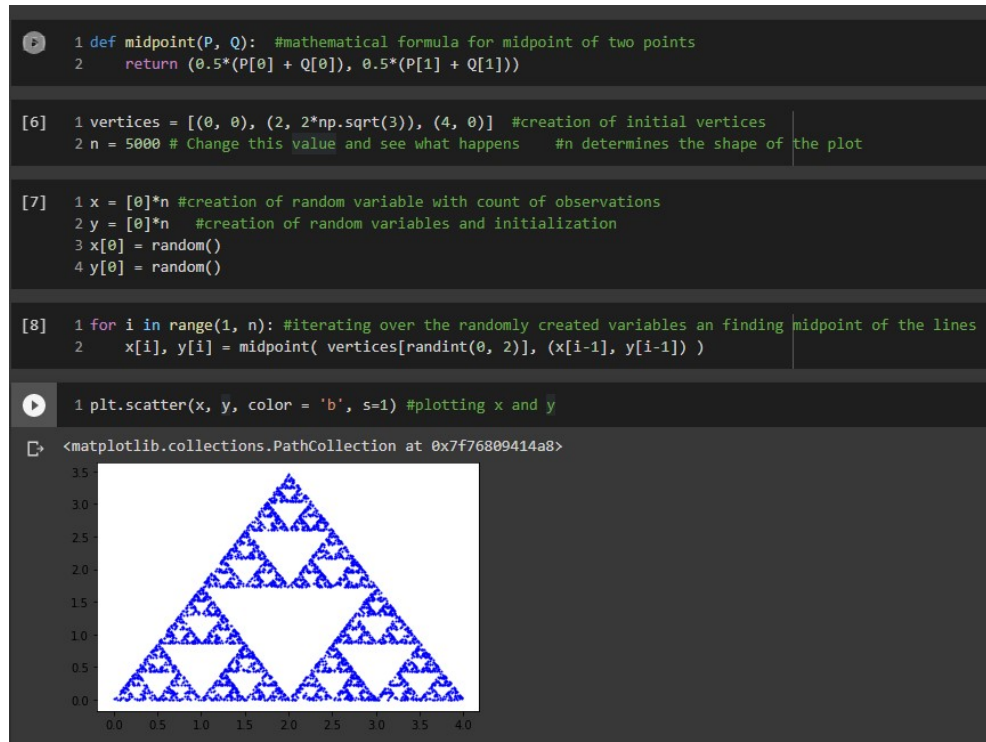
In this experiment we studied the effects of chaos, which in many cases, results in an ordered outcome. Using carefully defined mathematical operations, we arrived at ordered images such as Serpinski's triangle and Barnsley's fern, which are fractals.

1 Introduction

The application of chaos was demonstrated through the use of computer imaging and simple programming to create fractals.

2 Procedures and Graphics

The following excerpts are from the Colaboratory notebook containing code and commentary, as well as the graphics produced by editing the code in order to create new images.



```

1 def midpoint2(P, Q): #mathematical formula for midpoint of two points
2     return (0.5*(P[0] + Q[0]), 0.5*(P[1] + Q[1]))

[32] 1 vertices = [(0, 0), (2, np.sqrt(3)), (4, 0)] #creation of initial vertices
2     n2 = 5000 # Change this value and see what happens #n determines the shape of the plot

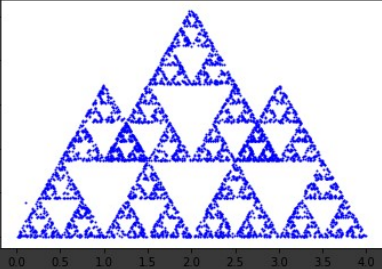
[33] 1 x2 = [0]*n2 #creation of random variable with count of observations
2     y2 = [0]*n2 #creation of random variables and initialization
3     x2[0] = random()
4     y2[0] = random()

[34] 1 for i in range(1, n2): #iterating over the randomly created variables an finding midpoint of the lines
2     x2[i], y2[i] = midpoint2( vertices[randint(0, 2)], (x[i-1], y[i-1]) )

[35] 1 plt.scatter(x2, y2, color = 'b', s=1) #plotting x and y

<matplotlib.collections.PathCollection at 0x7f7680200fd0>

```



```

[ ] 1 # Barnsley's Fern

1 # 1% of the time: x → 0, y → 0.16 y
2 # 85% of the time: x → 0.85 x + 0.04 y, y → -0.04 x + 0.05 y + 1.6
3 # 7% of the time: x → 0.2 x - 0.26 y, y → 0.23 x + 0.22 y + 1.6
4 # 7% of the time: x → -0.15 x + 0.28 y, y → 0.26 x + 0.24 y + 0.44

[36] 1 def pick(p):
2     c = np.cumsum(p) #picks a random sum from the cumulative sums of the probabilistic results
3     return bisect(c, np.random.random() * c[-1])

[37] 1 p = np.array([0.01, 0.07, 0.07, 0.85]) #array of probabilities previously defined

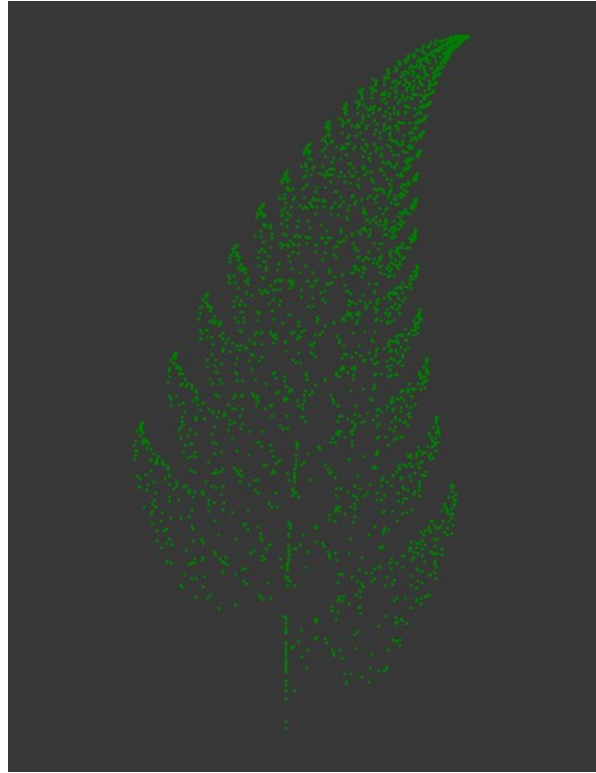
[38] 1 eq = [np.array([[0, 0, 0], [0, 0.16, 0]]), #mathematical equations to be weighted by probability
2         np.array([[0.2, -0.26, 0], [0.23, 0.22, 1.6]]), #for selection
3         np.array([[-0.15, 0.28, 0], [0.26, 0.24, 0.44]]),
4         np.array([[0.85, 0.04, 0], [-0.04, 0.85, 1.6]])]

[42] 1 n = 2500 # Change this value and see what happens
2     x = np.zeros((n, 3)) #this value determines how precise the image is
3     x[:, 2] = 1

[43] 1 for i in range(1, n): #function to find values from random probability
2     x[i, :2] = np.matmul(eq[pick(p)], x[i-1, :])

[44] 1 plt.figure(figsize=(10, 10))
2     plt.scatter(x[:, 0], x[:, 1], s=3, c="g", marker="s", linewidths=0)
3     plt.axis("equal"), plt.axis("off"); #plotting

```



3 Conclusions

Below the surface of a seemingly chaotic and random selection of mathematical operations lies a deeper complexity that can lead to very ordered things such as fractals. Experimentation of the code used yielded numerous other examples of chaos' effects, including its gradual ascent to order under higher amounts of observations.