The barnsley fern

is a pattern generated mathematically that can be created by iterating over a large number of times on four mathematical equations, introduced by Michael Barnsley, known as Iterated Function System (IFS). Being an IFS fractal pattern, it exhibits the characteristics of self-similarity, i.e. reproducible at any magnification or reduction. It is also infinitely complex in nature, as well as a chaotic fractal.

The Fractal Dimension of the Barnsley Fern cannot be calculated by conventional means, and is estimated to be about 1.45. Barnsley's fern uses four affine transformations. The formula for one transformation is the following

$$f(x) = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix}$$

where, the letters have the following value :

a	b	c	d	e	f	p	PART
0	0	0	0.16	0	0	0.01	Stem
0.85	0.04	-0.04	0.85	0	1.60	0.85	Small Leaflet
0.20	-0.26	0.23	0.22	0	1.60	0.07	Large Leaflet(Left)
-0.15	0.28	0.26	0.24	0	0.44	0.07	Large Leaflet(Right)

In the table above, the columns "a" through "f" are the coefficients of the equation, whereas "p" represents the proba bility factor. Hence, the four equations are:

$$f_1 :$$
$$x_{n+1} = 0$$
$$y_{n+1} = 0.16y_n$$
$$f_2 :$$
$$x_{n+1} = 0.85x_n + 0.04y_n$$
$$y_{n+1} = -0.04x_n + 0.85y_n + 1.6$$
$$f_3 :$$
$$x_{n+1} = 0.2x_n - 0.26y_n$$
$$y_{n+1} = 0.23x_n + 0.22y_n + 1.6$$
$$f_4 :$$
$$x_{n+1} = -0.15x_n + 0.28y_n$$
$$y_{n+1} = 0.26x_n + 0.24y_n + 0.44$$

With the help of the above equations, the fern fractal is generated as shown bellow:

The python code for this is the following:

```
import matplotlib.pyplot as plt
from random import randint

x = [0]
y = [0]

current = 0

for i in range(1, 58000):
    z = randint(1, 100)
    if z == 1:
        x.append(0)
        y.append(0.16 * y[current])
    elif 2 <= z <= 86:
        x.append(0.2 * x[current] - 0.26 * y[current] )
        y.append(0.23 * x[current] + 0.22 * y[current] + 1.6)
    elif 87 <= z <= 93:
        x.append(-0.15 * x[current] + 0.28 * y[current] )
        y.append(0.26 * x[current] + 0.24 * y[current] + 0.44)
    elif 94 <= z <= 100:
        x.append(0.85 * x[current] + 0.04 * y[current])
        y.append(-0.04 * x[current] + 0.85 * y[current] + 1.6)
    current += 1
```

```
plt.scatter(x, y, s=0.2, edgecolor='green')
plt.show()
plt.savefig("output.jpg")
```

Fractal trees

A recursive symmetric binary branching where a trunk of unit length splits into two branches of length R and make a Q angle with the direction of their parent. This process keeps on repeating a set certain number of times in order to give the final output.

A fractal tree, also known as a Symmetric Binary Tree, exhibits the concept of self-similarity. Each branch is a smaller version of the main trunk of the tree. The primary idea in constructing fractal trees is to have a base object and to then create smaller, similar objects protruding from that initial object. The angle, length and other features of these children can be randomized for a more naturalistic look. The method used is a recursive symmetric binary branching where a trunk of unit length splits into two branches of length R and make a θ angle with the direction of their parent. This process keeps on repeating a set certain number of times in order to give the final output.

The two branches are obtained from the trunk by scaling by a factor of r, rotating counterclockwise by θ (left) and θ (right), then translating to the top of the trunk. The third function needed is the identity. This keeps the branches already drawn in their current locations while the first two functions add the new branches:

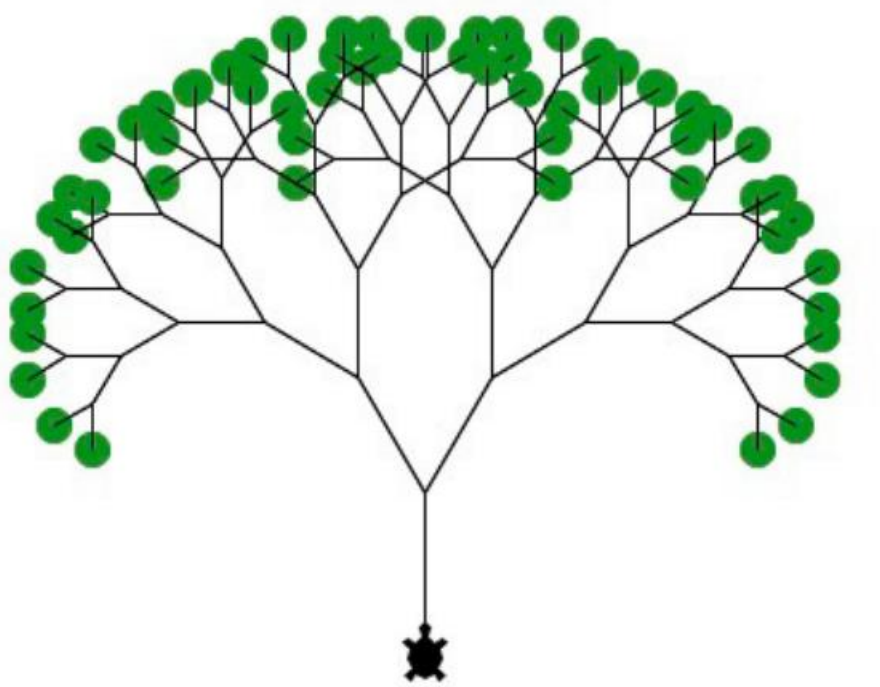
$$f_1(x) = \begin{bmatrix} r\cos(\theta) & -r\sin(\theta) \\ r\sin(\theta) & r\cos(\theta) \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$f_2(x) = \begin{bmatrix} r\cos(\theta) & r\sin(\theta) \\ -r\sin(\theta) & r\cos(\theta) \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$f_3(x) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \boldsymbol{x}$$

If the scaling factor r is too small, the branches of the tree will be self-avoiding, while if r is too large the branches will overlap.

If the function runs it will produce the following output



The code for this function is the following

```
from turtle import *

shape("turtle")
speed(0)

def tree(size, levels, angle):
    if levels <= 0:
        color("green")
        dot(size)
        color("black")
        return
    forward(size)
    right(angle)
    tree(size * 0.8, levels - 1, angle)
    left(angle * 2)
    tree(size * 0.8, levels - 1, angle)
    right(angle)
```

```
    backward(size)

    if levels == 7:  # Optional: Draws a trunk after the last level
        left(90)
        forward(size)
        right(90)
        color("brown")
        width(10)
        forward(20)
        width(1)
        color("black")
        backward(20)
        right(90)
        backward(size)
        left(90)

tree(70, 7, 30)
```

reference

1. Losa, Gabriele & Ristanovic, Dusan & Ristanovic, Dejan & Zaletel, Ivan & Beltraminelli, Stefano. (2016). From Fractal Geometry to Fractal Analysis. Applied Mathematics.
2. Neff, Cassondra, ”Fractal Ferns” (2021). Academic Excellence Showcase Proceedings. 281. https://digitalcommons.wou.edu/aes/281
3. Bovill, Carl. (2000). Fractal geometry as design aid. Journal for Geometry and Graphics Volume. 4. 71-78
4. Adam K. Glaser, Ye Chen, and Jonathan T. C. Liu, ”Fractal propagation method enables realistic optical microscopy simulations in biological tissues,”