Sierpinski's Carpet

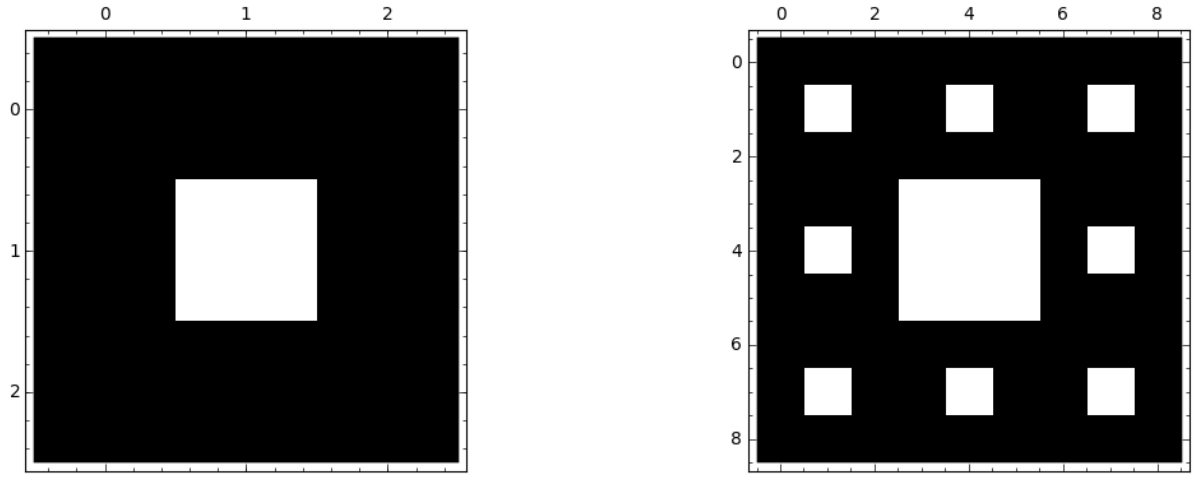
James Hedge

December 10, 2015

1 Introduction

Waclaw Sierpinski was a Polish mathematician born in 1882 [1] who first described the fractal I will be working with.

Sierpinski's Carpet is a fractal defined by taking a square and dividing it into a 3x3 grid of congruent squares, and then removing the central square. This process is then applied to the 8 remaining squares.



(a) Step 1

(b) Step 2

Figure 1: These show what the carpet looks like after 1 and 2 steps

This process is then repeated on the 8 smaller squares within each of the 8 larger ones, and if applied an infinite number of times it gives you Sierpinski's Carpet.

2 Idea behind code

The code used to generate the various stages was made by representing the carpet as a matrix containing 1's and 0's, with 1 representing a removed square (referred to as a pixel from now on) and 0 one that remains.

The resulting matrix looks complex but can be broken down into 3x3 matrices, of which only 2 types exist in the carpet:

$$\mathbf{A} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \mathbf{B} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

We can now define the process of getting the next step of Sierpinski's Carpet as replacing every 0 in the previous step by a type **A** matrix and every 1 by a type **B**. For this definition to work we need a base case, step 1, which clearly can be represented by a type **A** matrix

This definition allows you to code the process of Sierpinski's Carpet, although I did not code it to follow exactly the wording above (as doing so causes problems when trying to plot the matrix), my code uses this idea to find the required step.

3 Coding Sierpinski's Carpet

The full code used can be found at:

`https://cloud.sagemath.com/projects/2d50b04b-89d0-4bdb-a809-8bd701e9ef7d/files/Sierpinski.sagews`

3.1 Filled or not?

Bellow is the code used to establish if a given pixel is filled or not. Note that in the code the pixel (1,1) is in the upper left and 1 denotes unfilled not, as one may expect, filled (The reasoning for this can be found in Section 3.2)

```
class Pixel():
    def __init__(self, x, y):
        self.x = x
        self.y = y
        self.filled = 0
    def filled_check(self, n):
        if n >= 1:
            if (self.x % 3 == 2 and self.y % 3 == 2):
                self.filled = 1
            else:
                self.x = ceil(self.x / 3)
                self.y = ceil(self.y / 3)
                self.filled_check(n - 1)
```

In the code, the __init__ method gives the pixel its initial attributes. Whether the pixel is filled can then be checked by passing it through the filled_check method. This method works by applying the idea in Section 2 backwards. It figures out which pixel in the (n - 1) th step the 3x3 matrix the pixel in the nth step is a part of corresponds to, and does this until it gets to step 1 or a point where the pixel is not filled.

3.2 Generating and graphing the matrix

In order to probably visualize each step some form of graph must be made. Thankfully sage has an inbuilt matrix_plot function [2] which is perfect for what we need.

First however we must generate all the pixels and put them into a matrix form, which was done with the following code:

```
def sierpinski_graph(n):
    p = 3 ^ n
    line = 1
    matrix_s = []
    for i in range(1, p + 1):
        matrix_row = []
        for j in range(1, p + 1):
            pixel = Pixel(j, i)
            pixel.filled_check(n)
            matrix_row.append(pixel.filled)
        matrix_s.append(matrix_row)
    return matrix_s
```

It generates each pixel, starting in the upper left, it then generates the row, and adds the values of the filled attribute to a list, which is then added to the matrix. It then does this for each remaining row, giving the full matrix of 1's and 0's corresponding to the pixels.

```
matrix_plot(sierpinski_graph(5))
```

With this line, a plot is generated which represents all the 1's with a white (or blank) square, and 0's with a black (or filled) square, the result is shown in Figure 2

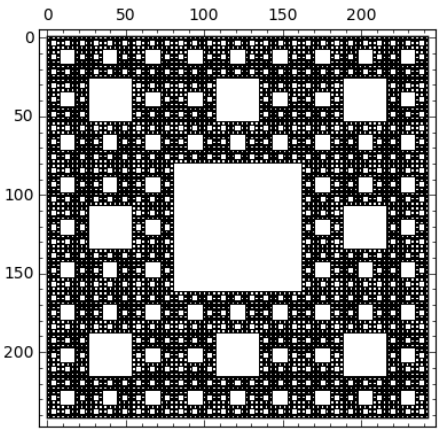


Figure 2: 5^{th} step of Sierpinski's Carpet

4 Conclusion

I have shown how to generate a given stage in Sierpinski's Carpet in python using a matrix interpretation of it. You could branch out into other fractals, including the more well known Sierpinski's Triangle.

My code could be built upon to be more efficient, allowing for quicker generation of higher steps. You could do this by taking advantage of the symmetrical nature of the Carpet, which would allow you to only need to calculate if the pixels are filled for 1/4 of the Carpet.

References

[1] E F Robertson J J O'Connor. Waclaw sierpinski. `http://www-history.mcs.st-andrews.ac.uk/Biographies/Sierpinski.html`, 1997.

[2] The Sage Development Team. Matrix plots. `http://doc.sagemath.org/html/en/reference/plotting/sage/plot/matrix_plot.html#sage.plot.matrix_plot.matrix_plot`.