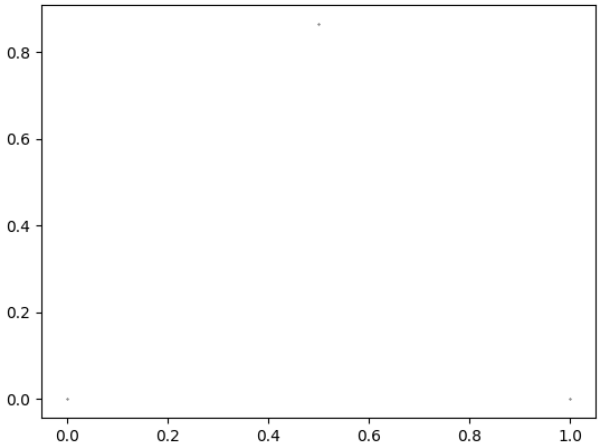
Práctica 3 voluntaria - GCOMP

Celia Rubio Madrigal

17 de mayo de 2022

Índice



*Para ver este gráfico en movimiento, se recomienda abrir este PDF en Adobe Reader u Okular.

1. Introducción

En esta práctica vamos a dibujar un fractal, y a calcular su medida de Hausdorff.

En este caso, el fractal escogido es el triángulo de Sierpinski, que es una generalización del conjunto de Cantor para triángulos equiláteros.

Sabemos que su dimensión de Hausdorff real es $\frac{\log 3}{\log 2} \approx 1.585 \pm 0.001$. Queremos observar cuánto nos acercamos a este valor.

2. Material usado y metodología

2.1. Apartado *i)*

En el primer apartado, vamos a construir el fractal y a representarlo. Tendremos una familia de subconjuntos del fractal en función de un nivel ℓ, cuyos puntos estarán contenidos en $\ell + 1$.

Para $\ell = 0$, el conjunto de puntos son los tres vértices del triángulo equilátero de lado 1. Para ℓ arbitrario, tomamos todos los triángulos formados en el nivel $\ell - 1$. Por cada uno de ellos (de coordenadas (p_1, p_2, p_3)), se añadirán estos triángulos a la capa siguiente, donde m es el punto medio:

$$\Delta_0 = (p_1, m(p_1, p_2)), m(p_1, p_3))$$

$$\Delta_1 = (m(p_1, p_2)), p2, m(p_2, p_3))$$

$$\Delta_2 = (m(p_1, p_3)), m(p_2, p_3), p_3)$$

2.2. Apartado *ii)*

En el segundo apartado, calcularemos su dimensión de Hausdorff. Tomando su d_0-medida de Hausdorff:

$$\mathcal{H}^{d_0} = \lim_{\delta \to 0} \inf \sum_{i \in I} \delta^{d_0}$$

Donde $\{U_i\}_{i \in I}$ es un recubrimiento del conjunto de diámetro δ embebido en $\mathbb{R}^2$.

La d que buscamos es la d ínfima en d_0 tal que $\mathcal{H}^{d_0}$ es nula, que es equivalente a la d suprema en d_0 tal que $\mathcal{H}^{d_0}$ es infinita.

Sea $\varepsilon = 0.01$. Tomaremos un recubrimiento de cuadrados de lado $\frac{1}{n}$, con n entre 1 y $\frac{1}{\varepsilon} = 100$. Recorreremos el conjunto de posibles dimensiones, desde 0 (dimensión de los puntos) a 2 (dimensión ambiente).

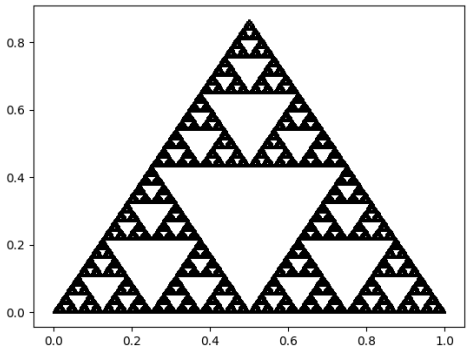
Por cada nivel de construcción ℓ del triángulo de Sierpinski, por cada n y por cada d, contaremos cuántos cuadrados del recubrimiento tienen puntos del fractal. Cuando la medida de Hausdorff (esa cantidad por el volumen) sea menor que ε, entonces pasaremos al siguiente n.

Por cada ℓ y cada d, haremos una gráfica de los valores de esa medida frente a n. El resultado que devolveremos será el máximo d para el máximo ℓ que hayamos considerado, tal que la medida de Hausdorff del máximo n sea menor que ε. Con el resto de gráficas, podremos comprobar las tendencias de cada medida tomada.

3. Resultados y conclusiones

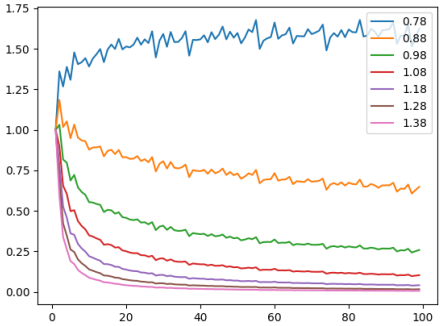
3.1. Apartado *i*)

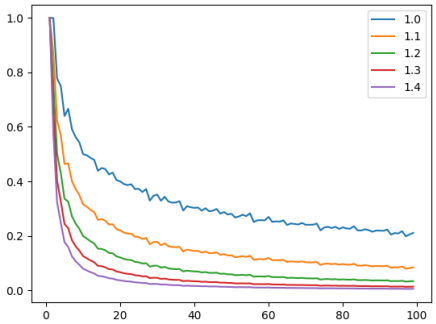
En la portada de este trabajo se encuentra una animación de todas las figuras para los niveles de 0 a 8. Para $\ell = 8$, el triángulo de Sierpinski resultante es el siguiente:



3.2. Apartado *ii*)

Por ejemplo, la gráfica del nivel $\ell = 6$ llega a $d = 1.38 \pm 0.01$, y las gráficas, desde $d = 0.78$, son las siguientes (izquierda):





A la derecha se encuentran las gráficas para el valor máximo $\ell = 8$. El d para el que, en el último n, la medida de Hausdorff fue menor que ε, fue $d = 1.40 \pm 0.01$. Ese es el valor que proporciona nuestro algoritmo. Con mayor capacidad de cómputo, haría tender al valor real de $\frac{\log 3}{\log 2} \approx 1.59 \pm 0.01$.

Con este algoritmo, no solo hemos calculado aproximadamente la dimensión de Hausdorff del triángulo de Sierpinski, sino que también podemos calcular la dimensión de otros fractales, incluso de aquellos para los que no tenemos una respuesta teórica.

4. Código

```
from cvxpy import norm
import matplotlib.pyplot as plt
import numpy as np

def mid(p1,p2):
    return np.array([(p1[0]+p2[0])/2,(p1[1]+p2[1])/2])

def norm(v1):
    return np.linalg.norm(v1)

def perp(v1):
    return np.array([-v1[1],v1[0]]) / norm(v1)

def tri(p1,p2):
    # third point of an equilateral triangle
    return mid(p1,p2) + perp(p2-p1)*norm(p2-p1)*np.sqrt(3)/2

""" Apartado i) """

max_level = 9

triangles = dict()
p01,p02 = np.array([0.,0.]),np.array([1.,0.])
p03 = tri(p01,p02)
triangles[0] = [[p01,p02,tri(p01,p02)]]
# plot new figure
fig = plt.figure()
plt.scatter([p01[0],p02[0],p03[0]],[p01[1],p02[1],p03[1]],color='k',s=0.1)
plt.savefig('img/triangle%d'%0)
plt.close()
points = dict()
points[0] = [p01,p02,p03]
for level in range(1,max_level):
    fig = plt.figure()
    triangles[level] = []
    points[level] = points[level-1]
    for [p1,p2,p3] in triangles[level-1]:
        tri_a = [p1,mid(p1,p2),mid(p1,p3)]
        tri_b = [mid(p1,p2),p2,mid(p2,p3)]
        tri_c = [mid(p1,p3),mid(p2,p3),p3]
        triangles[level] += [tri_a,tri_b,tri_c]
        # scatter points
        points[level] += [mid(p1,p2),mid(p1,p3),mid(p2,p3)]
        x,y = zip(*points[level])
        plt.scatter(x,y,color='k',s=0.1)

    plt.savefig('img/triangle%s'%str(level))
```

```
    plt.close()

    print(level,len(points[len(points)-1]))

""" Apartado ii) """

for level in range(2,max_level):
    ds = np.arange(0+(level+1)/max_level,1+(level+1)/max_level,0.1)
    fig = plt.figure()
    for d in ds:
        ms = []
        ns = range(1,100)
        for n in ns:
            total = dict()
            vertical_stripes = {i:[] for i in range(n)}
            for l in range(level):
                for px, py in points[l]:
                    i = int(px*n)
                    if i == n:
                        i -= 1
                    vertical_stripes[i].append(py)

                for i in range(n):
                    for py in vertical_stripes[i]:
                        j = int(py*n)
                        total[i*n+j] = 1

                total_sum = len(total)
                ms.append(total_sum * (1/n) ** (2*d))
        # plot
        plt.plot(ns,ms,label=str(round(d,2)))
        if(ms[-1]<0.01):
            print(level,d)
            break
    plt.legend()
    plt.savefig('img/'+str(level)+'.png')
    plt.close()

print(np.log(3)/np.log(2))
```