1. Introducción

En esta práctica programaremos un código para obtener una muestra de la Alfombra de Sierpinski y después calcularemos la dimensión de *Hausdorff* de la muestra anterior.

2. Material empleado

2.1. Obtener una muestra árbitraria de la Alfombra de Sierpinski

En este primer apartado lo que hacemos primero es generar una matriz de unos de dimension $N \times N$ donde N es 3^k y k es el número de iteraciones que vamos a realizar.

Tenemos una función recursiva que, dada una matriz cuadrada y su dimensión $N \times N$ (siempre una potencia de 3) divide la matriz en 9 submatrices de dimensión $\frac{N}{3} \times \frac{N}{3}$ y "vacía" la del medio. Después se llama a si misma recursivamente con las 8 submatrices restantes. La recursión finaliza con las matrices de dimensión 1×1 en las que no se realiza ningún cambio (caso base de la recursión).

La función *pintar* simplemente genera una imagen dada la matriz anterior.

2.2. Calcular la dimensión de *Hausdorff*

Para este segundo apartado primero realizaremos varias simplificaciones: nuestra Alfombra de Sierpinski será finita, los recubrimientos serán matrices cuadradas de dimensión 3^k con $k \in \mathbb{N}$ (esto es como usar bolas con la norma infinito) y los límites para calular la dimensión de *Hausdorff* los haremos sobre conjuntos finitos.

Definimos varias funciones para organizar y simplificar el código:

- *numRecs*: dada una alfombra y el diámetro del recubridor, devuelve el número de recubridores necesarios para cubrir toda la alfombra.
- *volumenDdim*: dada una alfombra, el diámetro del recubridor y la dimensión de Haussdorf, devuelve el volumen d-dimensional de la alfombra.
- *creciente*: dada una lista devuelve si es creciente o no.

Primero generamos una alfombra de dimensión $N \times N$ con la función del apartado anterior y una lista con los diámetros que utilizaremos (los divisores de N). Después para aproximar la dimensión d de la alfombra realizaremos una búsqueda binaria (con un número fijo de iteraciones) utilizando las siguientes expresión para saber que extremo elegir:

$$d = dim_H(E) := sup\{d_0 \geq 0 : H^{d_0}(E) = \infty\} = inf\{d_0 \geq 0 : H^{d_0}(E) = 0\}$$

De este modo si el límite de los volúmenes tiende a infinito tenemos que la dimensión de la alfombra será mayor y si tiende a cero entonces será menor. Como no podemos hacer el límite con nuestra representación finita, simplemente aplicamos la función *volumenDdim* a la lista de recubridores. Si es creciente lo interpretamos como si el límite tendiese a infinito y si es decreciente como si el límite tendiese a cero.

3. Resultados

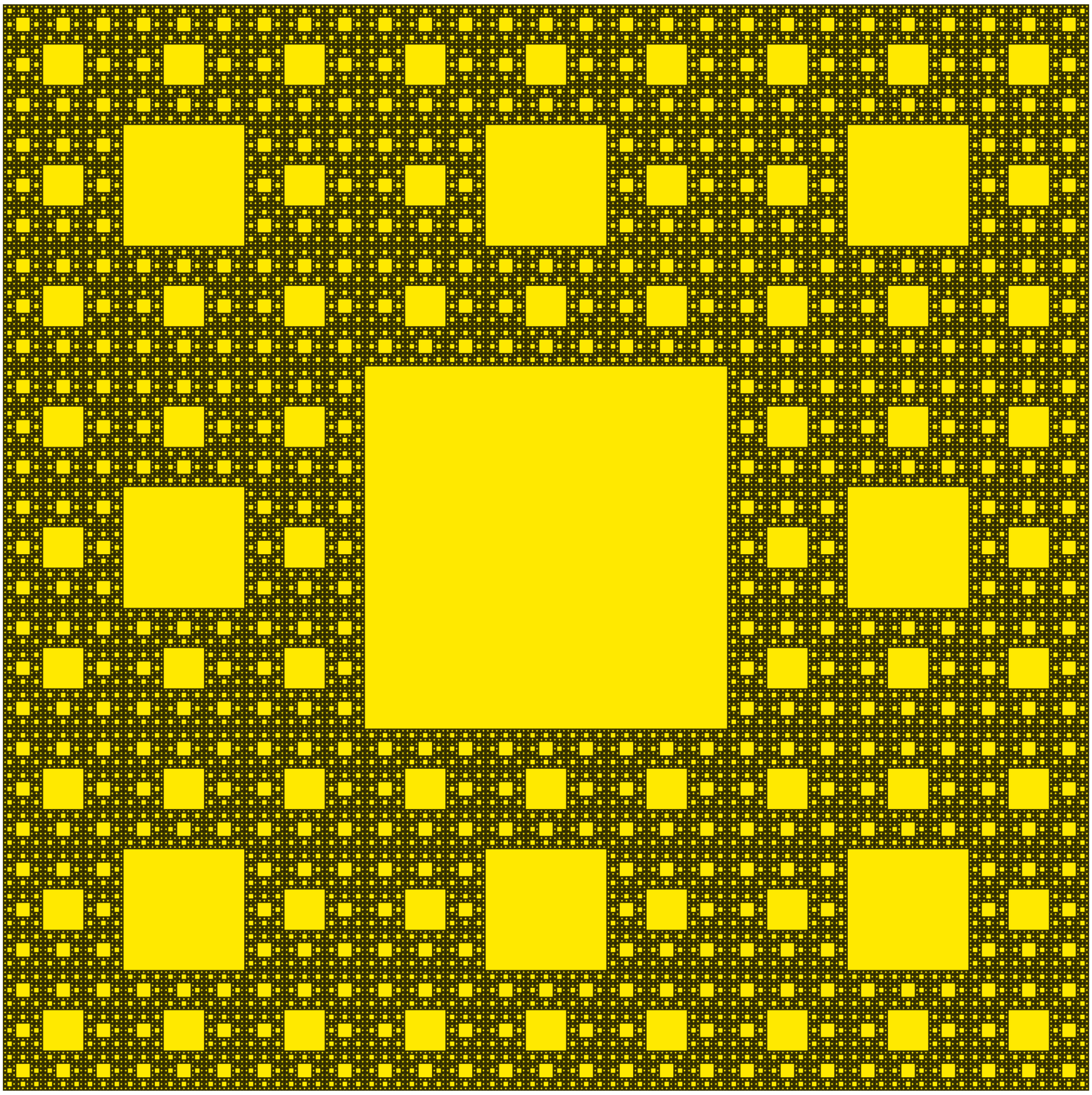

Figura 1: Alfombra de Sierpinski de 8 iteraciones

```
La dimensión de la alfombra de Sierpinski es:
1.8927888870239258
```

4. Conclusión

A pesar de que el método utiliza una Alfombra de Sierpinski y una sucesión de recubrimientos finitos, vemos que los resultados se acercan bastante a $\frac{\log 8}{\log 3} \approx 1{,}8927892607143723$.

Otro detalle a comentar es que primero intenté utilizar recubridores de cualquier diámetro, de hecho, la función *numRecs* está preparada para ello. El problema era que al calcular el volumen, utilizaba todo el diámetro de los recubridores incluso cuando solo cubrían parcialmente y las sucesiones eran siempre crecientes incluso cuando no debían.

5. Código

```
"""
Created on Wed Feb  5 13:41:00 2020

@author: Jorge Sainero
"""

import numpy as np
from PIL import Image
import math

#Recibe una matriz de 0s y 1s y la pinta
def pintar(a,dim):
    dibujo=np.empty((dim,dim,3), dtype=np.uint8)
    for i in range(dim):
        for j in range(dim):
            if a[i,j]==1:
                #negro
                dibujo[i,j]=[0,0,0]
            else:
                #amarillo
                dibujo[i,j]=[255,233,0]
    Image.fromarray(dibujo).save("Sierpinski.png")

def sierpinski(a,dim):
    if dim != 1:
        aux = dim//3
        #rellena el interior de 0s
        a[aux:2*aux,aux:2*aux] = 0
        for i in range(0,dim,aux):
            for j in range(0,dim,aux):
                #llamada recursiva menos a la del centro
                if i != aux or j != aux:
```

```
                    sierpinski(a[i:i+aux,j:j+aux],aux)

def apartado1():
    it = 8
    dim = 3**it
    alfombra=np.ones((dim,dim))
    sierpinski(alfombra,dim)
    pintar(alfombra,dim)
    return 0;

#Recibe una alfombra (o culaquier cosa a recubrir) y
#la dimensión de los recubridores (matrices cuadradas en este caso)
#devuelve cuantas son necesarias para recubrir la alfombra
def numRecs(alfombra,rec):
    total=0
    dim=alfombra.shape[0]
    ndim=math.ceil(dim/rec)*rec
    nalfombra=np.zeros((ndim,ndim))
    nalfombra[0:dim,0:dim]=alfombra
    for i in range(0,ndim,rec):
        for j in range(0,ndim,rec):
            if not np.array_equiv(nalfombra[i:i+rec,j:j+rec],np.zeros((rec,rec))):
                total+=1
    return total

#Devuelve el d-volumen dados la alfombra y el diámetro de los recubridores
def volumenDdim(alfombra,rec,d):
    return numRecs(alfombra,rec)*rec**d

#Devuelve cierto si la lista es creciente
def creciente(lista):
    for i in range(len(lista)-1):
        if lista[i]>lista[i+1]:
            return False;
    return True;

def apartado2():
    it = 5
    dim = 3**it
    alfombra=np.ones((dim,dim))
    sierpinski(alfombra,dim)
    #Generamos los diámetros de los recubridores
    recs=[3**x for x in range(it,-1,-1)]
```

```
    a=1 #Extremo izquierdo (sabemos que su d-volumen es mayor que el de
        una línea)
    b=2 #Extremo derecho (sabemos que su d-volumen es menor que el de un
        cuadrado)
    #Búsqueda binaria
    for i in range(20):
        c=(a+b)/2
        volumenes=[volumenDdim(alfombra,x,c)for x in recs]
        if creciente(volumenes):
            a=c
        else:
            b=c
    print("La dimensión de la alfombra de Sierpinski es:",c)

apartado1()
apartado2()
```