Practica 2.Componentes conexas y cardinales

El objetivo de esta práctica es encontrar el número de componentes conexas de un espacio, E, formado por 1000 segmentos aleatorios y la implementación y comprensión de la fórmula de inclusión-exclusión para calcular la unión e intersección de 100 conjuntos.

En cuanto al material usado se nos ha proporcionado una plantilla en la que venía un conjunto de 1000 segmentos y 100 conjuntos de cadenas de 1 o 2 caracteres. Además de eso hemos descargado la librería Shapely y hemos hecho uso de ella además de otras librerías como son:

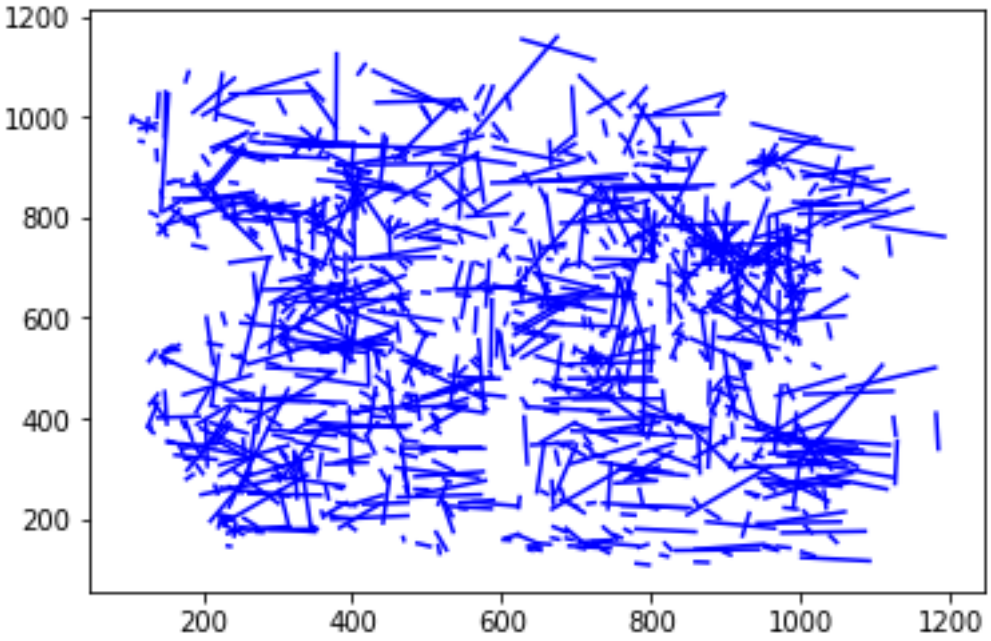
Random,
string,
matplotlib.pyplot,
NumPy,
LineString y Point(shapely.geometry),
Itertools

Resultados

Parte 1

a) Primero generamos los segmentos gracias a LineString y Point. Una vez hecho esto creamos un diccionario con todos los segmentos (keys), donde cada una tendrá de elementos a los segmentos que interseque. Creamos una lista vacía en la que mediante un proceso recursivo vamos a ir añadiendo, empezando por el segmento 0, los segmentos que no estén en esa lista. Siguiendo en el proceso recursivo, luego este recorrerá los segmentos a los que puede llegar por conexión (que estén en la llave del diccionario (que es un segmento))y si alguno de estos no está en la lista l ,que al principio estaba vacía, se volverá a llamar al proceso recursive para añadirlo a l. Cada vez que el segmento no esté en l sumaremos 1 al número de componentes conexas que luego devolveremos después de recorrer todos los segmentos (llaves).

b) Al ejecutar el código representamos gráficamente el conjunto y después nos da el número de componentes conexas que es 411.



Parte 2

a), b) Mediante las funciones intersection() y union() ya predefinidas conseguimos los resultados. La unión tendrá 702 elementos y la intersección tendrá 0.

c)Si guardáramos los conjuntos de intersecciones que sean vacíos, luego al realizar la intersección que los tenga, podremos obviar dicha intersección (conjunto vacío) porque si implementamos la formula, con itertools realizamos todas las posibles combinaciones de intersecciones de los conjuntos dados entonces a partir de más o menos 20 conjuntos dados funciona pero a partir de ahí he llegado a esperar una hora y no termina.

Conclusión

En la primera parte vemos que a partir del algoritmo las componentes conexas de conjuntos bastante complejos pueden ser calculadas, además podemos ver los elementos por los que están compuestos dichas componentes, gracias al uso de diccionarios.

En cuanto al ejercicio 2 para pocos conjuntos es factible pero para muchos el coste para implementar y ejecutar la fórmula de inclusión-exclusión es enorme y habría que hacer el cambio mencionado en c).

GCOM2021-practica2_plantilla.py | plantilla_prueba_practica2.py | Gcom_practica_2.py

```python
import random
import string
import numpy as np
import matplotlib.pyplot as plt
from shapely.geometry import LineString, Point
import itertools

# ################################## PARTE 1 ###################################

#Generamos 1000 segmentos aleatorios, pero siempre serán los mismos

#Usaremos primero el concepto de coordenadas
X = []
Y = []

#Fijamos el modo aleatorio con una versión prefijada
random.seed(a=0, version=2)

#Generamos subconjuntos cuadrados del plano R2 para determinar los rangos de X e Y
xrango1 = random.sample(range(100, 1000), 200)
xrango2 = list(np.add(xrango1, random.sample(range(10, 230), 200)))
yrango1 = random.sample(range(100, 950), 200)
yrango2 = list(np.add(yrango1, random.sample(range(10, 275), 200)))

for j in range(len(xrango1)):
    for i in range(5):
        random.seed(a=i, version=2)
        xrandomlist = random.sample(range(xrango1[j], xrango2[j]), 4)
        yrandomlist = random.sample(range(yrango1[j], yrango2[j]), 4)
        X.append(xrandomlist[0:2])
        Y.append(yrandomlist[2:4])


Segmentos=[]
for i in range(len(X)):
    Point1=Point(X[i][0], Y[i][0])
    Point2=Point(X[i][1], Y[i][1])
    Segment=LineString([Point1, Point2])
    Segmentos.append(Segment)
```

```python
def componentes_conexas(Segments):
    dicc=dict()
    for i in range(len(Segments)):
        dicc[i]=[]
        for j in range(len(Segments)):
            if (j!=i) and (Segments[i].intersects(Segments[j])):
                dicc[i].append(j)
    n=0
    l=[]
    comp=0
    while len(l)<len(dicc):
        if not(n in l):
            l=recursive(n,l,dicc)
            comp+=1
        n+=1
    return(comp)

def recursive(numero,lista, diccionario):
    lista.append(numero)
    for i in diccionario[numero]:
        if not(i in lista):
            lista=recursive(i, lista,diccionario)
    return lista



#APARTADO B
#Representamos el Espacio topológico representado por los 1000 segmentos

for i in range(len(X)):
    plt.plot(X[i], Y[i], 'b')
plt.show()




print(componentes_conexas(Segmentos))
```

GCOM2021-practica2_plantilla.py | plantilla_prueba_practica2.py | Gcom_practica_2.py | GCOM.py

```
97 alphabet = list(string.ascii_lowercase)
98 generator = alphabet
99 for i in range(len(alphabet)):
100     for j in range(len(alphabet)):
101         generator = np.append(generator, generator[i]+alphabet[j])
102
103 #Fijamos la configuración de un determinado conjunto aleatorio (¡siempre será el mismo!)
104 random.seed(a=0, version=1)
105
106 #Fijamos un número de conjuntos, y asignamos un cardinal diferentes a cada uno de ellos
107 nsets = 100
108 cardinals = random.sample(range(0, int(len(generator)/4)), nsets)
109
110 #Generamos los conjuntos "aleatorios" (prefijados), sin repeticiones
111 Sets = []
112 for i in range(nsets):
113     random.seed(a=i, version=2)
114     index_random = random.sample(range(0, len(generator)), cardinals[i])
115     newset = list(np.array(generator)[index_random])
116     Sets.append(newset)
117
118 def interseccion():
119     intersec = []
120     for i in range(len(Sets)-1):
121         inter = set(Sets[i]).intersection(Sets[i+1])
122         intersección = set(inter).intersection(intersec)
123     print("Numero de componentes de la intersección: {0}".format(len(intersec)))
124
125 def union():
126     list_union = []
127     for i in range(len(Sets)-1):
128         U = set(Sets[i]).union(Sets[i+1])
129         list_union = set(U).union(list_union)
130     print("Numero de componentes de la unión: {0}".format(len(list_union)))
131
132 def comprobar(subset, empty):
133     for element in empty:
134         if element in subset:
135             return False
136     return True
137
138
139 def B_size(sets):
140     sum = 0
141     sets = [set(elem) for elem in sets]
142     for i in range(1, len(sets) + 1):
143         for subset in itertools.combinations(sets, i):
144                 a = (-1)**(i+1) * len(set.intersection(*subset))
145                 sum += a
146     return sum
147
148 union()
149 interseccion()
150
151
```