Proyecto Final

Leilani Reséndiz Álvarez

Ejercicio 1

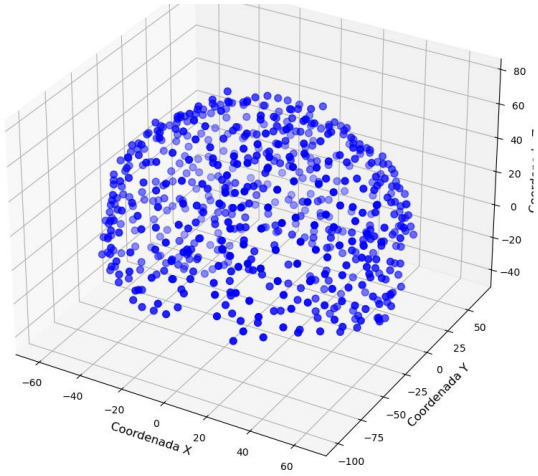
Definir grafos con la matriz estableciendo umbrales de coactivación de 0.8, 0.9 y 1 y graficar cada grafo. Añadir las coordenadas tridimensionales (incluidas en el archivo.mat).

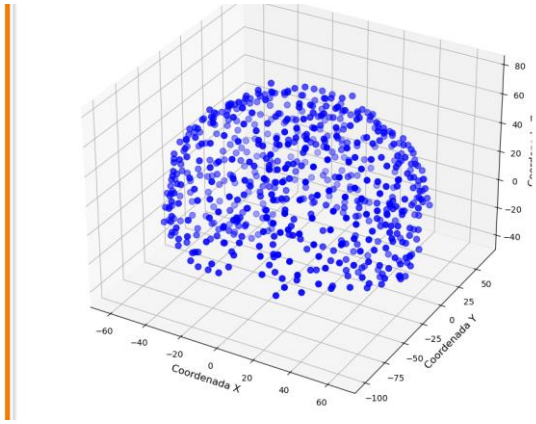
Código:

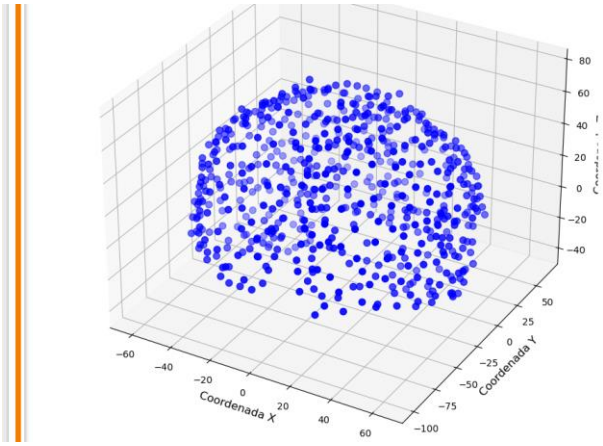
```
import scipy.io as sp

import numpy as np

import networkx as nx

import matplotlib.pyplot as plt


data = sp.loadmat(r"C:\Users\alvar\OneDrive\Documentos\Semestre 5\Modelos computacionales\Coactivation_matrix.mat")

coactivation_matrix = data['Coactivation_matrix']

coordinates = data['Coord']


x_coords, y_coords, z_coords = coordinates[:, 0], coordinates[:, 1], coordinates[:, 2]

thresholds = [0.8, 0.9, 1]

for threshold in thresholds:


    filtered_matrix = (coactivation_matrix >= threshold).astype(int)

   graph = nx.from_numpy_array(filtered_matrix)


    fig = plt.figure(figsize=(10, 10))

    ax = fig.add_subplot(projection='3d')

    ax.scatter(x_coords, y_coords, z_coords, c='blue', s=50, label='Nodos')


    print(f'\nConexiones para el grafo con umbral {threshold}:')

    for edge in graph.edges():

        node_a, node_b = edge

        x_pair = [x_coords[node_a], x_coords[node_b]]

        y_pair = [y_coords[node_a], y_coords[node_b]]

        z_pair = [z_coords[node_a], z_coords[node_b]]

        ax.plot(x_pair, y_pair, z_pair, c='green', alpha=0.7, linewidth=1)

        print(f'Conexión entre nodos {node_a}-{node_b}: x={x_pair}, y={y_pair}, z={z_pair}')
```

```
    ax.set_title(f'Grafo con Umbral {threshold}', fontsize=16)
    ax.set_xlabel('Coordenada X', fontsize=12)
    ax.set_ylabel('Coordenada Y', fontsize=12)
    ax.set_zlabel('Coordenada Z', fontsize=12)
    ax.legend()

plt.show()
```

GRAFOS:







Puntos azules: Nodos

Grafo 1: umbral 0.8 / Grafo 2: umbral 0.9 / grafo 3: Umbral 1

Ejercicio 2

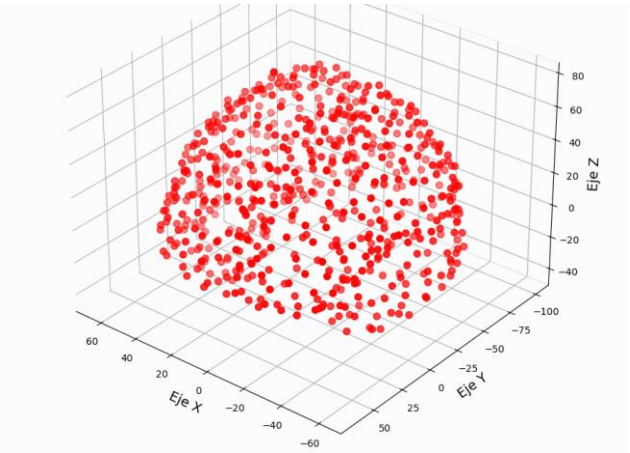
Con uno de los grafos en el punto uno con umbral 0.9, generar una animación donde se haga girar 360° el grafo del cerebro para visualizar las conexiones establecidas

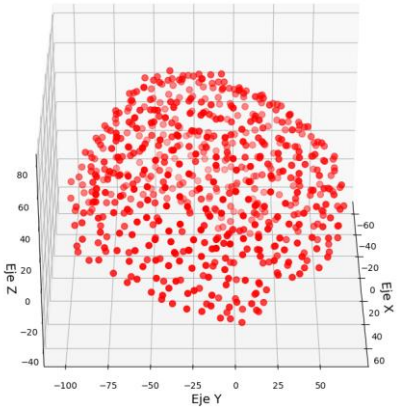
Código:

```
import scipy.io as sp

import numpy as np

import networkx as nx

import matplotlib.pyplot as plt

from matplotlib.animation import FuncAnimation, PillowWriter

from IPython.display import Image


data = sp.loadmat(r"C:\Users\alvar\OneDrive\Documentos\Semestre 5\Modelos computacionales\Coactivation_matrix.mat")

coactivation_matrix = data['Coactivation_matrix']

coordinates = data['Coord']


x_points, y_points, z_points = coordinates[:, 0], coordinates[:, 1], coordinates[:, 2]


threshold = 0.9

filtered_matrix = (coactivation_matrix >= threshold).astype(int)


graph = nx.from_numpy_array(filtered_matrix)


fig = plt.figure(figsize=(10, 10))

ax = fig.add_subplot(projection='3d')

ax.scatter(x_points, y_points, z_points, c='red', s=50, label='Nodos')


for edge in graph.edges():

    start, end = edge

    x_line = [x_points[start], x_points[end]]

    y_line = [y_points[start], y_points[end]]

    z_line = [z_points[start], z_points[end]]

    ax.plot(x_line, y_line, z_line, c='blue', alpha=0.6)


ax.set_title(f'Conexiones para umbral {threshold}', fontsize=20)

ax.set_xlabel('Eje X', fontsize=14)

ax.set_ylabel('Eje Y', fontsize=14)

ax.set_zlabel('Eje Z', fontsize=14)

ax.legend()


def iniciar_animacion():

    ax.view_init(elev=30, azim=0)
```

```
    return fig,

def actualizar(frame):
    ax.view_init(elev=30, azim=frame)
    return fig,

rotacion = FuncAnimation(fig, actualizar, init_func=iniciar_animacion, frames=360, interval=40, blit=False)

archivo_gif = "rotacion_grafo.gif"
rotacion.save(archivo_gif, writer=PillowWriter(fps=15))
Image(archivo_gif)
```

Grafo:





(En el archivo si gira)

Nodos: puntos rojos, umbral: 0.9

.

Ejercicio 3

Encontrar los hubs del grafo, y establecer el tamaño del nodo proporcional al valor del grado.

Código:

```
node_degrees = dict(graph.degree())
percentile_threshold = np.percentile(list(node_degrees.values()), 99)

hub_nodes = [node for node, degree in node_degrees.items() if degree >= percentile_threshold]

print(f"Se encontraron {len(hub_nodes)} hubs:")
print("Lista de hubs:", hub_nodes)
print(f"El umbral usado para definir un hub es: {percentile_threshold:.2f}")
```

```
fig = plt.figure(figsize=(12, 12))
ax = fig.add_subplot(projection='3d')

node_sizes = [degree * 10 for degree in node_degrees.values()]
ax.scatter(x_points, y_points, z_points, c='cyan', s=node_sizes, label='Nodos')

for edge in graph.edges():
    start, end = edge
    x_pair = [x_points[start], x_points[end]]
    y_pair = [y_points[start], y_points[end]]
    z_pair = [z_points[start], z_points[end]]
    ax.plot(x_pair, y_pair, z_pair, c='gray', alpha=0.5)

hub_sizes = [node_degrees[hub] * 20 for hub in hub_nodes]
ax.scatter(
    x_points[hub_nodes],
    y_points[hub_nodes],
    z_points[hub_nodes],
    c='red',
    s=hub_sizes,
    label='Hubs',
)

ax.set_title("Visualización de Hubs en el Grafo", fontsize=18)
ax.set_xlabel("Eje X", fontsize=12)
ax.set_ylabel("Eje Y", fontsize=12)
ax.set_zlabel("Eje Z", fontsize=12)
ax.legend()

plt.show()
```

Grafo

```
Se encontraron 638 hubs:
Lista de hubs: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 3
6, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 7
4, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109,
110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 14
0, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170,
171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 20
1, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231,
232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 26
2, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292,
293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 32
3, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353,
354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 38
4, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414,
415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441, 442, 443, 444, 44
5, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475,
476, 477, 478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 50
6, 507, 508, 509, 510, 511, 512, 513, 514, 515, 516, 517, 518, 519, 520, 521, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536,
537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558, 559, 560, 561, 562, 563, 564, 565, 566, 56
7, 568, 569, 570, 571, 572, 573, 574, 575, 576, 577, 578, 579, 580, 581, 582, 583, 584, 585, 586, 587, 588, 589, 590, 591, 592, 593, 594, 595, 596, 597,
598, 599, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 610, 611, 612, 613, 614, 615, 616, 617, 618, 619, 620, 621, 622, 623, 624, 625, 626, 627, 62
8, 629, 630, 631, 632, 633, 634, 635, 636, 637]
El umbral usado para definir un hub es: 0.00
```

Ejercicio 4

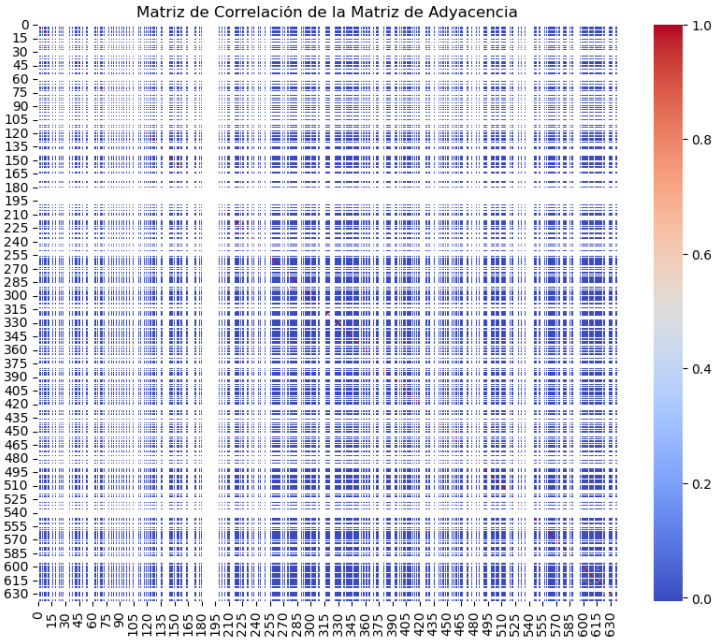
En función de la matriz de emparejamiento (correlación de la matriz de adyacencia), establecer una partición de los nodos en módulos. Escoger el número de módulos que creas conveniente y justificar por qué escogiste ese número.

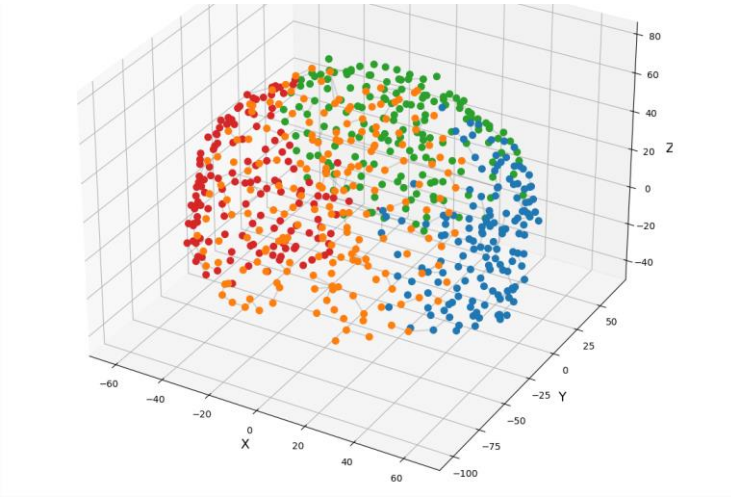
Código:

```
import scipy.io as sp

import numpy as np

import networkx as nx

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.cluster import KMeans


arch = sp.loadmat(r"C:\Users\alvar\OneDrive\Documentos\Semestre 5\Modelos computacionales\Coactivation_matrix.mat")

CM = arch['Coactivation_matrix']

coord = arch['Coord']

x, y, z = coord[:, 0], coord[:, 1], coord[:, 2]


threshold = 0.2

matriz_filtrada = np.where(CM >= threshold, CM, 0)


G = nx.from_numpy_array(matriz_filtrada)


matriz_de_correlacion = np.corrcoef(matriz_filtrada)


plt.figure(figsize=(10, 8))

sns.heatmap(matriz_de_correlacion, cmap='coolwarm', cbar=True)

plt.title("Matriz de Correlación de la Matriz de Adyacencia")

plt.show()


num_clusters = 4

kmeans = KMeans(n_clusters=num_clusters, random_state=42)

clusters = kmeans.fit_predict(coord)


node_colors = clusters


fig = plt.figure(figsize=(12, 12))

ax = fig.add_subplot(projection='3d')


for node in G.nodes():
```

```
    ax.scatter(x[node], y[node], z[node], c=plt.cm.tab10(node_colors[node]), s=50)


for edge in G.edges():
    node1, node2 = edge
    x_coords = [x[node1], x[node2]]
    y_coords = [y[node1], y[node2]]
    z_coords = [z[node1], z[node2]]
    ax.plot(x_coords, y_coords, z_coords, c='gray', alpha=0.3)


ax.set_title(f"Partición en {num_clusters} Módulos (K-means)", fontsize=20)
ax.set_xlabel("X", fontsize=14)
ax.set_ylabel("Y", fontsize=14)
ax.set_zlabel("Z", fontsize=14)
plt.show()


# Mi justificación
print(f"Se seleccionaron {num_clusters} módulos utilizando el algoritmo K-means. "
      f"Este número fue elegido para capturar diferentes regiones del grafo. "
      f"Los módulos agrupan nodos cercanos en el espacio tridimensional, "
      f"lo cual puede reflejar regiones funcionales o anatómicas del cerebro.")
```

Grafos:





Partición en 4 módulos.

Ejercicio 5

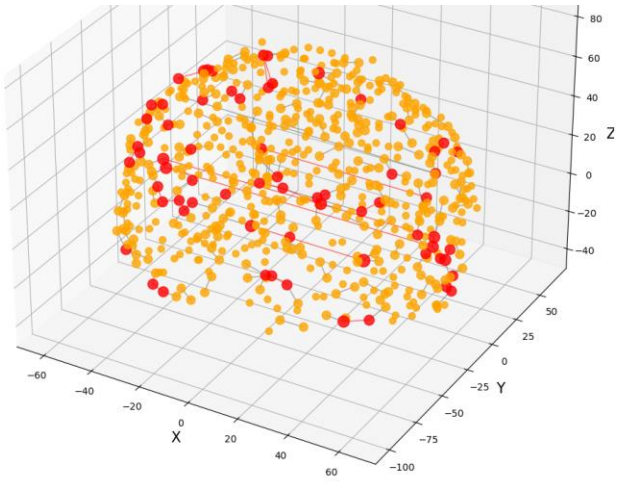
Determinar el conjunto del Rich Club y discutir las implicaciones anatómicas y funcionales de este grupo de nodos (mínimo 100 palabras).

Código:

```
grados = dict(G.degree())
grado_umbral = np.percentile(list(grados.values()), 90)  # Umbral del percentil 90
rich_club = [nodo for nodo, grado in grados.items() if grado >= grado_umbral]


print(f"Rich Club (grado ≥ {grado_umbral}): {rich_club}")


tamaños_nodos = [50 + (grados[nodo] * 30) for nodo in G.nodes()]
colores_nodos = ['red' if nodo in rich_club else 'orange' for nodo in G.nodes()]


figura = plt.figure(figsize=(12, 12))
ax = figura.add_subplot(projection='3d')


for idx, nodo in enumerate(G.nodes()):
    ax.scatter(
        x[nodo], y[nodo], z[nodo],
        color=colores_nodos[idx],
        s=tamaños_nodos[idx],
        alpha=0.8
    )


for edge in G.edges():
    node1, node2 = edge
```

```
    x_coords = [x[node1], x[node2]]
    y_coords = [y[node1], y[node2]]
    z_coords = [z[node1], z[node2]]
    color = 'red' if node1 in rich_club and node2 in rich_club else 'dimgray'
    ax.plot(x_coords, y_coords, z_coords, c=color, alpha=0.6, linewidth=1)

ax.set_title('Rich Club (ROJO)', fontsize=25)
ax.set_xlabel('X', fontsize=15)
ax.set_ylabel('Y', fontsize=15)
ax.set_zlabel('Z', fontsize=15)

plt.show()
```

Grafos:



Rich Club (grado ≥ 2.0): [38, 42, 62, 69, 92, 124, 150, 217, 221, 223, 226, 230, 235, 245, 259, 260, 263, 275, 281, 290, 292, 296, 304, 309, 319, 320, 321, 327, 330, 331, 344, 352, 356, 358, 369, 393, 404, 407, 408, 416, 419, 428, 456, 460, 465, 482, 532, 547, 561, 565, 574, 579, 599, 603, 604, 605, 611, 613, 614, 615, 616, 617, 618, 622, 623, 628, 629]

Rich club (rojo)

Explicación:

El Rich Club representa un grupo de nodos en una red que tienen grados altos y están densamente interconectados. Anatómicamente, estos nodos suelen corresponder a regiones cerebrales asociadas con procesamiento crítico, integración de información y control, como la corteza prefrontal o el cíngulo. Funcionalmente, son esenciales para mantener la eficiencia global de la red cerebral, actuando como hubs que conectan diferentes módulos o regiones. En este análisis, los nodos del Rich Club están destacados en crimson, mostrando su alta interconexión. Sus conexiones reforzadas indican su papel crucial en la comunicación global y en mantener la robustez frente a daños o fallos.

Ejercicio 6

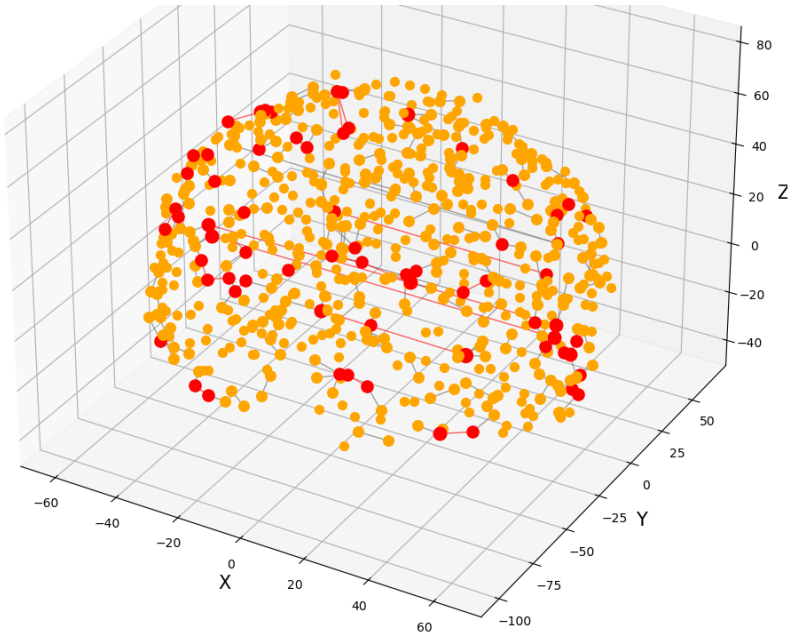
No hay

Ejercicio 7

Supongamos que eliminamos los nodos del RichClub, describir cómo cambian las propiedades topológicas del grafo, hacer comparativas del grado, coeficiente de cluster, coeficiente de mundo pequeño y las medidas de centralidad (cercanía, intermediación)

Código:

```
#Ejercicio 7 Se necesita hacer en 2 partes:
# Identificar los nodos del Rich Club
grados = dict(G.degree())
grado_umbral = np.percentile(list(grados.values()), 90)
rich_club = [nodo for nodo, grado in grados.items() if grado >= grado_umbral]

G_sin_rich_club = G.copy()
G_sin_rich_club.remove_nodes_from(rich_club)

def graficar_grafo(G, rich_club_nodes=None, titulo="Grafo"):
    figura = plt.figure(figsize=(12, 12))
    ax = figura.add_subplot(projection='3d')

    for nodo in G.nodes():
        tamaño = 50 + grados.get(nodo, 0) * 20
        color = 'red' if rich_club_nodes and nodo in rich_club_nodes else 'orange'
        ax.scatter(x[nodo], y[nodo], z[nodo], color=color, s=tamaño)

    for edge in G.edges():
        node1, node2 = edge
        x_coords = [x[node1], x[node2]]
        y_coords = [y[node1], y[node2]]
        z_coords = [z[node1], z[node2]]
        color = 'red' if rich_club_nodes and node1 in rich_club_nodes and node2 in rich_club_nodes else 'dimgray'
        ax.plot(x_coords, y_coords, z_coords, c=color, alpha=0.6, linewidth=1)

    ax.set_title(titulo, fontsize=20)
    ax.set_xlabel('X', fontsize=15)
    ax.set_ylabel('Y', fontsize=15)
    ax.set_zlabel('Z', fontsize=15)
    plt.show()

graficar_grafo(G, rich_club, titulo="Grafo con Rich Club (ROJO)")
graficar_grafo(G_sin_rich_club, titulo="Grafo sin Rich Club")
```



Parte 2:

```python
# Segunda parte
def calcular_propiedades_topologicas(G, nombre):
    propiedades = {
        "Número de nodos": G.number_of_nodes(),
        "Número de aristas": G.number_of_edges(),
        "Grado promedio": sum(dict(G.degree()).values()) / G.number_of_nodes(),
        "Coeficiente de clustering promedio": nx.average_clustering(G),
        "Centralidad de cercanía (promedio)": np.mean(list(nx.closeness_centrality(G).values())),
        "Centralidad de intermediación (promedio)": np.mean(list(nx.betweenness_centrality(G).values())),
    }

    print(f"\nPropiedades del grafo '{nombre}':")
    for clave, valor in propiedades.items():
        print(f"  {clave}: {valor:.4f}")
    return propiedades

propiedades_original = calcular_propiedades_topologicas(G, "Original (Con Rich Club)")
propiedades_modificado = calcular_propiedades_topologicas(G_sin_rich_club, "Sin Rich Club")
```

```
Propiedades del grafo 'Original (Con Rich Club)':
  Número de nodos: 638.0000
  Número de aristas: 188.0000
  Grado promedio: 0.5893
  Coeficiente de clustering promedio: 0.0047
  Centralidad de cercanía (promedio): 0.0011
  Centralidad de intermediación (promedio): 0.0000

Propiedades del grafo 'Sin Rich Club':
  Número de nodos: 571.0000
  Número de aristas: 73.0000
  Grado promedio: 0.2557
  Coeficiente de clustering promedio: 0.0000
  Centralidad de cercanía (promedio): 0.0004
  Centralidad de intermediación (promedio): 0.0000
```

Ejercicio 8

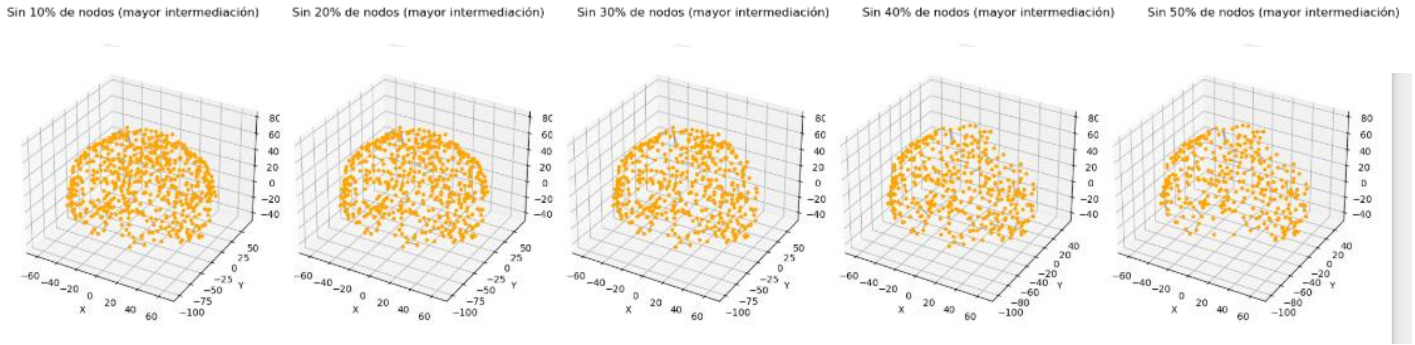
Quitar 10%-50% de los nodos con mayor medida de intermediación y describir cómo cambian las propiedades topológicas del grafo, hacer comparativas del grado, coeficiente de cluster, coeficiente de mundo pequeño y las medidas de centralidad (cercanía, intermediación)

Código:

```
def calcular_propiedades(G, descripcion):
    propiedades = {
        "Grado medio": sum(dict(G.degree()).values()) / G.number_of_nodes(),
        "Coeficiente de clustering promedio": nx.average_clustering(G),
        "Centralidad de cercanía promedio": np.mean(list(nx.closeness_centrality(G).values())),
        "Centralidad de intermediación promedio": np.mean(list(nx.betweenness_centrality(G).values())),
    }
    print(f"\nPropiedades del grafo: {descripcion}")
    for clave, valor in propiedades.items():
        print(f"  {clave}: {valor:.4f}")
    return propiedades


def eliminar_nodos_intermediacion(G, porcentaje):
    intermediacion = nx.betweenness_centrality(G)
    nodos_ordenados = sorted(intermediacion, key=intermediacion.get, reverse=True)
    num_nodos_eliminar = int(len(nodos_ordenados) * porcentaje / 100)
    nodos_a_eliminar = nodos_ordenados[:num_nodos_eliminar]


    G_modificado = G.copy()
    G_modificado.remove_nodes_from(nodos_a_eliminar)
    return G_modificado


def graficar_grafo(G, titulo, x, y, z, ax):
    for nodo in G.nodes():
        ax.scatter(x[nodo], y[nodo], z[nodo], color='orange', s=10)


    for edge in G.edges():
        node1, node2 = edge
        x_coords = [x[node1], x[node2]]
        y_coords = [y[node1], y[node2]]
        z_coords = [z[node1], z[node2]]
        ax.plot(x_coords, y_coords, z_coords, c='dimgray', alpha=0.5)
```

```
    ax.set_title(titulo, fontsize=12)
    ax.set_xlabel('X', fontsize=10)
    ax.set_ylabel('Y', fontsize=10)
    ax.set_zlabel('Z', fontsize=10)

print("Propiedades del grafo original:")
propiedades_originales = calcular_propiedades(G, "Original")

porcentajes = [10, 20, 30, 40, 50]
fig, axs = plt.subplots(1, len(porcentajes), figsize=(20, 6), subplot_kw={'projection': '3d'})

x, y, z = coord[:, 0], coord[:, 1], coord[:, 2]

for i, porcentaje in enumerate(porcentajes):
    G_modificado = eliminar_nodos_intermediacion(G, porcentaje)

    descripcion = f"Sin {porcentaje}% de nodos (mayor intermediación)"
    propiedades_modificadas = calcular_propiedades(G_modificado, descripcion)

    graficar_grafo(G_modificado, descripcion, x, y, z, axs[i])

plt.tight_layout()
plt.show()
```

```
Propiedades del grafo original:

Propiedades del grafo: Original
  Grado medio: 0.5893
  Coeficiente de clustering promedio: 0.0047
  Centralidad de cercanía promedio: 0.0011
  Centralidad de intermediación promedio: 0.0000

Propiedades del grafo: Sin 10% de nodos (mayor intermediación)
  Grado medio: 0.2713
  Coeficiente de clustering promedio: 0.0052
  Centralidad de cercanía promedio: 0.0005
  Centralidad de intermediación promedio: 0.0000

Propiedades del grafo: Sin 20% de nodos (mayor intermediación)
  Grado medio: 0.2622
  Coeficiente de clustering promedio: 0.0059
  Centralidad de cercanía promedio: 0.0005
  Centralidad de intermediación promedio: 0.0000

Propiedades del grafo: Sin 30% de nodos (mayor intermediación)
  Grado medio: 0.2550
  Coeficiente de clustering promedio: 0.0067
  Centralidad de cercanía promedio: 0.0006
  Centralidad de intermediación promedio: 0.0000

Propiedades del grafo: Sin 40% de nodos (mayor intermediación)
  Grado medio: 0.2663
  Coeficiente de clustering promedio: 0.0078
  Centralidad de cercanía promedio: 0.0007
  Centralidad de intermediación promedio: 0.0000

Propiedades del grafo: Sin 50% de nodos (mayor intermediación)
  Grado medio: 0.2696
  Coeficiente de clustering promedio: 0.0094
  Centralidad de cercanía promedio: 0.0008
  Centralidad de intermediación promedio: 0.0000
```



Ejercicio 9

Generar un modelo nulo aleatorio donde se tenga el mismo número de nodos y el mismo número total de conexiones, y comparar sus propiedades con el grafo original del cerebro.

Código:

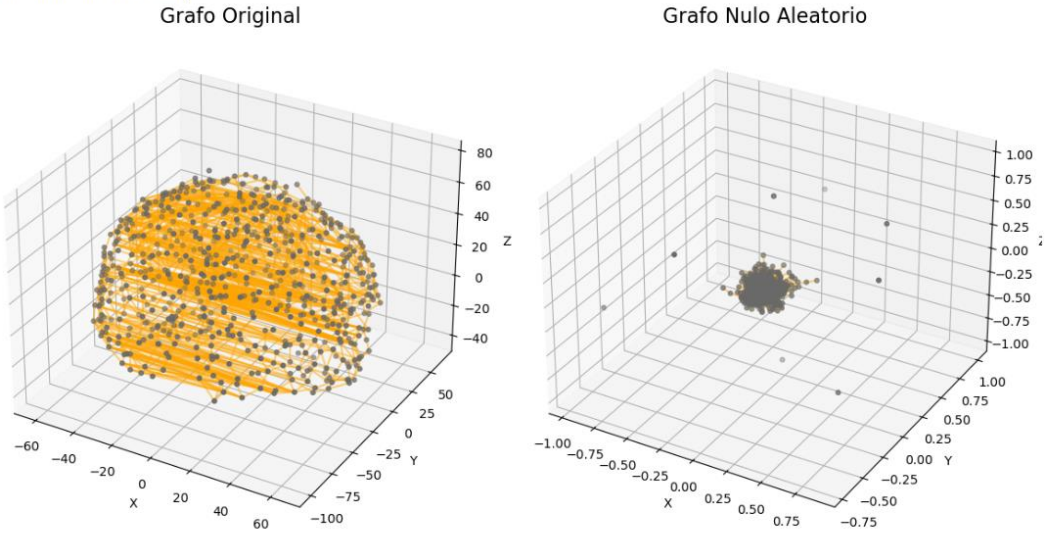
```
import scipy.io as sp

import numpy as np

import networkx as nx

import matplotlib.pyplot as plt


archivo = sp.loadmat(r"C:\Users\alvar\OneDrive\Documentos\Semestre 5\Modelos computacionales\Coactivation_matrix.mat")

CM = archivo['Coactivation_matrix']

coord = archivo['Coord']

x, y, z = coord[:, 0], coord[:, 1], coord[:, 2]
```

```
umbral = 0.1

def propiedades_grafo(G):
    propiedades = {}
    avg_grad = sum(dict(G.degree()).values()) / G.number_of_nodes()
    propiedades['grado_medio'] = avg_grad

    coef_cluster = nx.average_clustering(G)
    propiedades['coeficiente_de_agrupamiento'] = coef_cluster

    cercania = np.mean(list(nx.closeness_centrality(G).values()))
    propiedades['centralidad_cercania'] = cercania

    betweenness = nx.betweenness_centrality(G)
    intermediacion = np.mean(list(betweenness.values()))
    propiedades['centralidad_intermediacion'] = intermediacion

    return propiedades

def grafo_nulo_aleatorio(num_nodos, num_aristas):
    G_random = nx.gnm_random_graph(num_nodos, num_aristas)
    return G_random

matriz_filtrada = np.where(CM >= umbral, CM, 0)
G = nx.from_numpy_array(matriz_filtrada)

num_nodos = G.number_of_nodes()
num_aristas = G.number_of_edges()

G_random = grafo_nulo_aleatorio(num_nodos, num_aristas)

print(f"\nPropiedades del grafo original para el umbral {umbral}:")
propiedades_original = propiedades_grafo(G)
print(propiedades_original)

print(f"\nPropiedades del grafo nulo aleatorio para el umbral {umbral}:")
propiedades_random = propiedades_grafo(G_random)
print(propiedades_random)
```

```
fig = plt.figure(figsize=(12, 6))


ax1 = fig.add_subplot(121, projection='3d')
ax1.scatter(x, y, z, c='dimgray', s=50, marker='.')
for edge in G.edges():
    node1, node2 = edge
    x_coords = [x[node1], x[node2]]
    y_coords = [y[node1], y[node2]]
    z_coords = [z[node1], z[node2]]
    ax1.plot(x_coords, y_coords, z_coords, c='orange', alpha=0.5)
ax1.set_title('Grafo Original', fontsize=16)
ax1.set_xlabel('X')
ax1.set_ylabel('Y')
ax1.set_zlabel('Z')


ax2 = fig.add_subplot(122, projection='3d')
pos_random = nx.spring_layout(G_random, dim=3)
x_r, y_r, z_r = np.array(list(pos_random.values())).T
ax2.scatter(x_r, y_r, z_r, c='dimgray', s=50, marker='.')
for edge in G_random.edges():
    node1, node2 = edge
    x_coords = [x_r[node1], x_r[node2]]
    y_coords = [y_r[node1], y_r[node2]]
    z_coords = [z_r[node1], z_r[node2]]
    ax2.plot(x_coords, y_coords, z_coords, c='orange', alpha=0.5)
ax2.set_title('Grafo Nulo Aleatorio', fontsize=16)
ax2.set_xlabel('X')
ax2.set_ylabel('Y')
ax2.set_zlabel('Z')


plt.tight_layout()
plt.show()
```

Grafo:

```
Propiedades del grafo original para el umbral 0.1:
{'grado_medio': 4.529780564263323, 'coeficiente_de_agrupamiento': 0.24364464673564803, 'centralidad_cercania': 0.1194031264903796, 'centralidad_intermed
iacion': 0.0114771293087121 7}

Propiedades del grafo nulo aleatorio para el umbral 0.1:
{'grado_medio': 4.529780564263323, 'coeficiente_de_agrupamiento': 0.008552349774920307, 'centralidad_cercania': 0.221916307549579, 'centralidad_intermed
iacion': 0.0052550696197875306}
```



Ejercicio 10

Generar un modelo nulo aleatorio donde se conserve la distribución de grado y comparar sus propiedades con el grafo original del cerebro.

Código:

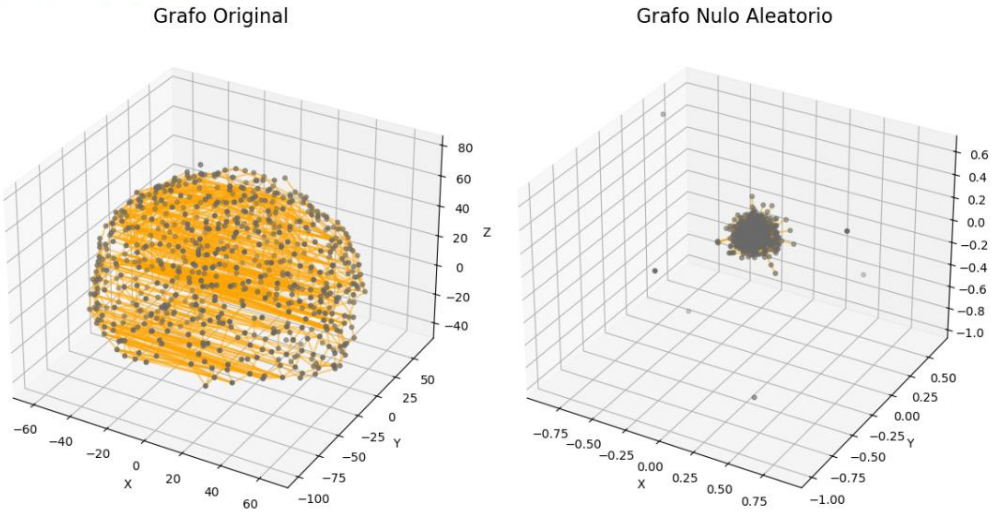
```
import scipy.io as sp

import numpy as np

import networkx as nx

import matplotlib.pyplot as plt


archivo = sp.loadmat(r"C:\Users\alvar\OneDrive\Documentos\Semestre 5\Modelos computacionales\Coactivation_matrix.mat")

CM = archivo['Coactivation_matrix']

coord = archivo['Coord']

x, y, z = coord[:, 0], coord[:, 1], coord[:, 2]


umbral = 0.1


def propiedades_grafo(G):

    propiedades = {}

    avg_grad = sum(dict(G.degree()).values()) / G.number_of_nodes()

    propiedades['grado_medio'] = avg_grad


    coef_cluster = nx.average_clustering(G)

    propiedades['coeficiente_de_agrupamiento'] = coef_cluster
```

```
    cercania = np.mean(list(nx.closeness_centrality(G).values()))
    propiedades['centralidad_cercania'] = cercania

    betweenness = nx.betweenness_centrality(G)
    intermediacion = np.mean(list(betweenness.values()))
    propiedades['centralidad_intermediacion'] = intermediacion

    return propiedades

def grafo_nulo_con_grados(G):
    G_random = nx.configuration_model([d for _, d in G.degree()])
    G_random = nx.Graph(G_random)  # Convertir a grafo simple
    G_random.remove_edges_from(nx.selfloop_edges(G_random))  # Eliminar bucles
    return G_random

matriz_filtrada = np.where(CM >= umbral, CM, 0)
G = nx.from_numpy_array(matriz_filtrada)

G_random = grafo_nulo_con_grados(G)

print(f"\nPropiedades del grafo original para el umbral {umbral}:")
propiedades_original = propiedades_grafo(G)
print(propiedades_original)

print(f"\nPropiedades del grafo nulo aleatorio para el umbral {umbral}:")
propiedades_random = propiedades_grafo(G_random)
print(propiedades_random)

fig = plt.figure(figsize=(12, 6))

ax1 = fig.add_subplot(121, projection='3d')
ax1.scatter(x, y, z, c='dimgray', s=50, marker='.')
for edge in G.edges():
    node1, node2 = edge
    x_coords = [x[node1], x[node2]]
    y_coords = [y[node1], y[node2]]
    z_coords = [z[node1], z[node2]]
    ax1.plot(x_coords, y_coords, z_coords, c='orange', alpha=0.5)
ax1.set_title('Grafo Original', fontsize=16)
```

```
ax1.set_xlabel('X')

ax1.set_ylabel('Y')

ax1.set_zlabel('Z')


ax2 = fig.add_subplot(122, projection='3d')

pos_random = nx.spring_layout(G_random, dim=3)

x_r, y_r, z_r = np.array(list(pos_random.values())).T

ax2.scatter(x_r, y_r, z_r, c='dimgray', s=50, marker='.')

for edge in G_random.edges():

    node1, node2 = edge

    x_coords = [x_r[node1], x_r[node2]]

    y_coords = [y_r[node1], y_r[node2]]

    z_coords = [z_r[node1], z_r[node2]]

    ax2.plot(x_coords, y_coords, z_coords, c='orange', alpha=0.5)

ax2.set_title('Grafo Nulo Aleatorio', fontsize=16)

ax2.set_xlabel('X')

ax2.set_ylabel('Y')

ax2.set_zlabel('Z')


plt.tight_layout()

plt.show()
```

Grafo:

```
Propiedades del grafo original para el umbral 0.1:
{'grado_medio': 4.529780564263323, 'coeficiente_de_agrupamiento': 0.24364464673564803, 'centralidad_cercania': 0.1194031264903796, 'centralidad_intermed
iacion': 0.01147712930871217}

Propiedades del grafo nulo aleatorio para el umbral 0.1:
{'grado_medio': 4.492163009404389, 'coeficiente_de_agrupamiento': 0.009623370360047475, 'centralidad_cercania': 0.2292685672619379, 'centralidad_interme
diacion': 0.005127861573627909}
```



Ejercicio 11

Generar un modelo nulo utilizando una probabilidad de conexión en función de la distancia geométrica, con el mismo número de nodos y conexiones y compara sus propiedades y discutir la importancia de las conexiones a larga distancia en el cerebro.

Código:

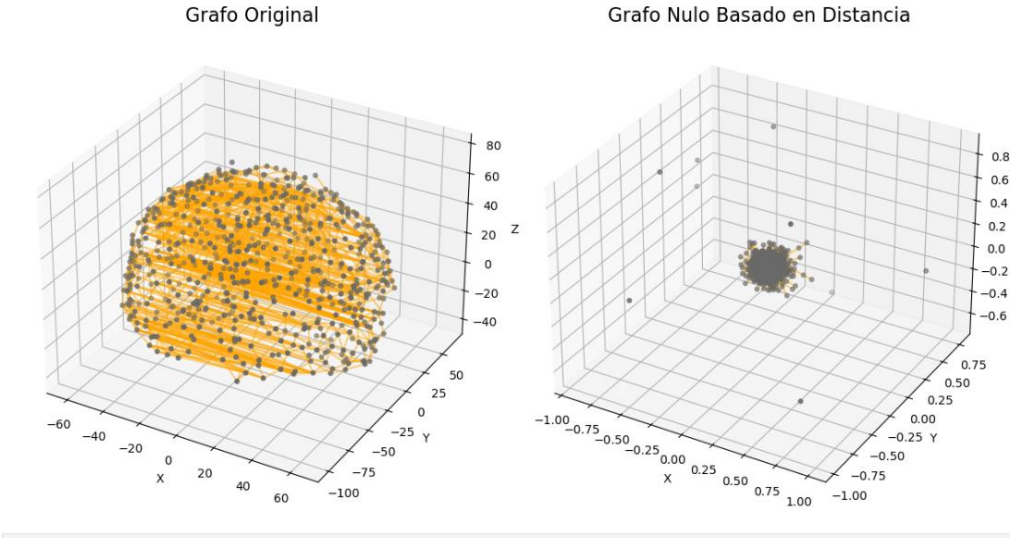
```
import scipy.io as sp

import numpy as np

import networkx as nx

import matplotlib.pyplot as plt


archivo = sp.loadmat(r"C:\Users\alvar\OneDrive\Documentos\Semestre 5\Modelos computacionales\Coactivation_matrix.mat")

CM = archivo['Coactivation_matrix']

coord = archivo['Coord']

x, y, z = coord[:, 0], coord[:, 1], coord[:, 2]


umbral = 0.1


def propiedades_grafo(G):

    propiedades = {}

    avg_grad = sum(dict(G.degree()).values()) / G.number_of_nodes()

    propiedades['grado_medio'] = avg_grad


    coef_cluster = nx.average_clustering(G)

    propiedades['coeficiente_de_agrupamiento'] = coef_cluster


    cercania = np.mean(list(nx.closeness_centrality(G).values()))

    propiedades['centralidad_cercania'] = cercania


    betweenness = nx.betweenness_centrality(G)

    intermediacion = np.mean(list(betweenness.values()))

    propiedades['centralidad_intermediacion'] = intermediacion


    return propiedades


def distancia_euclidiana(i, j, coord):

    return np.sqrt((coord[i, 0] - coord[j, 0]) ** 2 + (coord[i, 1] - coord[j, 1]) ** 2 + (coord[i, 2] - coord[j, 2]) ** 2)


def grafo_nulo_distancia(coord, num_nodos, num_aristas, alpha=0.1):
```

```
    G_random = nx.Graph()
    G_random.add_nodes_from(range(num_nodos))

    edges = []
    for i in range(num_nodos):
        for j in range(i + 1, num_nodos):
            dist = distancia_euclidiana(i, j, coord)
            p = 1 / (1 + alpha * dist)  # Probabilidad inversamente proporcional a la distancia
            if np.random.rand() < p:
                edges.append((i, j))

    while len(edges) > num_aristas:
        index = np.random.randint(0, len(edges))
        edges.pop(index)

    G_random.add_edges_from(edges)
    return G_random

matriz_filtrada = np.where(CM >= umbral, CM, 0)
G = nx.from_numpy_array(matriz_filtrada)

num_nodos = G.number_of_nodes()
num_aristas = G.number_of_edges()

G_random_dist = grafo_nulo_distancia(coord, num_nodos, num_aristas)

print(f"\nPropiedades del grafo original:")
propiedades_original = propiedades_grafo(G)
print(propiedades_original)

print(f"\nPropiedades del grafo nulo aleatorio basado en distancia:")
propiedades_random_dist = propiedades_grafo(G_random_dist)
print(propiedades_random_dist)

fig = plt.figure(figsize=(12, 6))

ax1 = fig.add_subplot(121, projection='3d')
ax1.scatter(x, y, z, c='dimgray', s=50, marker='.')
for edge in G.edges():
```

```
    node1, node2 = edge
    x_coords = [x[node1], x[node2]]
    y_coords = [y[node1], y[node2]]
    z_coords = [z[node1], z[node2]]
    ax1.plot(x_coords, y_coords, z_coords, c='orange', alpha=0.5)
ax1.set_title('Grafo Original', fontsize=16)
ax1.set_xlabel('X')
ax1.set_ylabel('Y')
ax1.set_zlabel('Z')

ax2 = fig.add_subplot(122, projection='3d')
pos_random = nx.spring_layout(G_random_dist, dim=3)
x_r, y_r, z_r = np.array(list(pos_random.values())).T
ax2.scatter(x_r, y_r, z_r, c='dimgray', s=50, marker='.')
for edge in G_random_dist.edges():
    node1, node2 = edge
    x_coords = [x_r[node1], x_r[node2]]
    y_coords = [y_r[node1], y_r[node2]]
    z_coords = [z_r[node1], z_r[node2]]
    ax2.plot(x_coords, y_coords, z_coords, c='orange', alpha=0.5)
ax2.set_title('Grafo Nulo Basado en Distancia', fontsize=16)
ax2.set_xlabel('X')
ax2.set_ylabel('Y')
ax2.set_zlabel('Z')

plt.tight_layout()
plt.show()
```

Grafo:

```
Propiedades del grafo original:
{'grado_medio': 4.529780564263323, 'coeficiente_de_agrupamiento': 0.24364464673564803, 'centralidad_cercania': 0.1194031264903796, 'centralidad_intermec
iacion': 0.0114771293087121 7}

Propiedades del grafo nulo aleatorio basado en distancia:
{'grado_medio': 4.529780564263323, 'coeficiente_de_agrupamiento': 0.007093098472408817, 'centralidad_cercania': 0.2213825548770658 2, 'centralidad_interm
ediacion': 0.005195365405422 1025}
```



Ejercicio 12

Escribir una reseña de lo aprendido en el curso, incluyendo la importancia de conocer herramientas de teoría de grafos para comprender la conectividad del cerebro (mínimo 200 palabras).

En lo persona considero que ha sido de las materias más pesada y menos deseadas para mí, no soy buena con la tecnología y el uso de estas herramientas y el curso no me llamo tanto la atención. Sin embargo, aprecio mucho el apoyo de ambos docentes Fernando y Oswaldo porque me ayudaron a sobrellevar y poder aprender lo necesario para mi formación.

Considero que los aspectos más relevantes y que puedo unir con neurociencias son la representación de redes cerebrales como grafos, donde representamos la conexiones sinápticas o funcionales como nodos y aristas, también aplicamos métricas que son cruciales para identificar regiones cerebrales altamente conectadas, estudiamos cómo las conexiones a larga distancia influyen en la integración de información entre diferentes áreas del cerebro, aprendimos a filtrar datos para obtener redes funcionales o estructurales relevantes, una habilidad importante para analizar datos de resonancia magnética funcional o tractografía y muchas otras herramientas.

Me interesa es aspecto de conducta y cuestiones como la neurodivergencia o enfermedades psiquiátricas y aunque al principio pensé que no tenia nada que ver ahora me doy cuenta de que esta más relacionado de lo que pensé. Modelos computacionales me puede ayudar a explorar cómo los patrones de conectividad cerebral y la dinámica neuronal están relacionados con diversos aspectos, también en modelos de redes funcionales, simulación de trastornos, predicción y diagnóstico.