

November 30, 2024

0.1 Proyecto final - Karime Jiménez Romero

Realizar las siguientes actividades usando la matriz de coactivación del conectoma del cerebro humano

```
[1]: import scipy.io
```

```
[2]: math_path = r"C:\Users\karim\OneDrive\Documentos\Python_
      ↳Scripts\Coactivation_matrix.mat"
      contents = scipy.io.loadmat(math_path)
      contents
```

```
[2]: {'__header__': b'MATLAB 5.0 MAT-file, Platform: GLNX86, Created on: Wed Dec 25
01:14:03 2013',
      '__version__': '1.0',
      '__globals__': [],
      'Coactivation_matrix': array([[0.          , 0.16071429, 0.11148649, ..., 0.
0.05045872,
      0.1011236 ],
      [0.16071429, 0.          , 0.06825939, ..., 0.          , 0.          ,
      0.06923077],
      [0.11148649, 0.06825939, 0.          , ..., 0.03412969, 0.          ,
      0.          ],
      ...,
      [0.          , 0.          , 0.03412969, ..., 0.          , 0.          ,
      0.          ],
      [0.05045872, 0.          , 0.          , ..., 0.          , 0.          ,
      0.09777778],
      [0.1011236 , 0.06923077, 0.          , ..., 0.          , 0.09777778,
      0.          ]]),
      'Coord': array([[ 7.24363636, 37.01090909,  9.42545455],
      [ 7.98653199, 46.22222222, 15.60942761],
      [ 7.55725191, 33.83206107, 23.51145038],
      ...,
      [-4.92385787, 15.31979695, 27.73604061],
      [-6.27312775, 34.70484581, -5.09251101],
      [-4.53874539, 46.53874539,  3.06273063]])})
```

1. Definir grafos con la matriz estableciendo umbrales de coactivación de 0.08, 0.09 y 0.1 y

graficar cada grafo. Añadir las coordenadas tridimensionales (incluidas en el archivo.mat)

```
[3]: coactivation_matrix = contents["Coactivation_matrix"]
      coord = contents["Coord"]
      coactivation_matrix.shape, coord.shape
```

```
[3]: ((638, 638), (638, 3))
```

```
[4]: import networkx as nx
      import matplotlib.pyplot as plt
      from mpl_toolkits.mplot3d import Axes3D
      import numpy as np

      def create_and_plot_graph(matrix, coords, threshold, ax):
          adjacency_matrix = (matrix>=threshold).astype(int)

          G = nx.from_numpy_array(adjacency_matrix)
          # Graficar en 3D
          ax.set_title(f"Threshold: {threshold}")
          ax.axis("off")
          for edge in G.edges():
              x_vals = [coords[edge[0], 0], coords[edge[1], 0]]
              y_vals = [coords[edge[0], 1], coords[edge[1], 1]]
              z_vals = [coords[edge[0], 2], coords[edge[1], 2]]
              ax.plot(x_vals, y_vals, z_vals, c='b', alpha=0.5, linewidth=0.7)

          ax.scatter(coords[:, 0], coords[:, 1], coords[:, 2], c='r', s=10)

      fig = plt.figure(figsize=(18, 6))

      thresholds = [0.08, 0.09, 0.1]

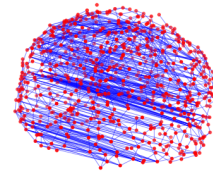
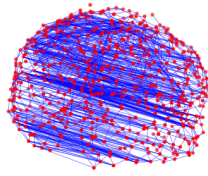
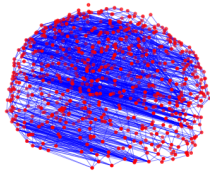
      for i, threshold in enumerate(thresholds, 1):
          ax = fig.add_subplot(1, 3, i, projection='3d')
          create_and_plot_graph(coactivation_matrix, coord, threshold, ax)

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

Threshold: 0.08

Threshold: 0.09

Threshold: 0.1



2. Con uno de los grafos en el punto uno con umbral 0.09, generar una animación donde se haga girar 360° el grafo del cerebro para visualizar las conexiones establecidas

```
[5]: from matplotlib.animation import FuncAnimation

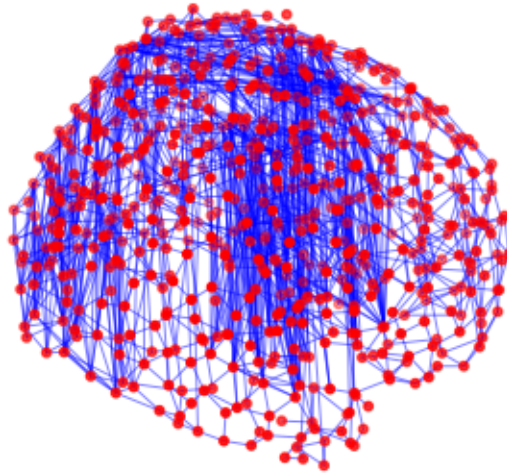
def update_rotation(frame, graph, coords, ax):
    ax.cla() # Limpiar el eje
    ax.set_title("Grafo 360°")
    ax.axis("off")
    for edge in graph.edges():
        x_vals = [coords[edge[0], 0], coords[edge[1], 0]]
        y_vals = [coords[edge[0], 1], coords[edge[1], 1]]
        z_vals = [coords[edge[0], 2], coords[edge[1], 2]]
        ax.plot(x_vals, y_vals, z_vals, c='b', alpha=0.5, linewidth=0.7)
    ax.scatter(coords[:, 0], coords[:, 1], coords[:, 2], c='r', s=10)
    ax.view_init(30, frame) # Cambiar el ángulo de vista

# Grafo para el umbral de 0.09
adjacency_matrix_09 = (coactivation_matrix >= 0.09).astype(int)
graph_09 = nx.from_numpy_array(adjacency_matrix_09)

fig = plt.figure(figsize=(6,6))
ax = fig.add_subplot(111, projection='3d')

anim = FuncAnimation(fig, update_rotation, frames=360, fargs=(graph_09, coord,
↪ax), interval=50)
```

Grafo 360°



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

```
[6]: degrees = dict(graph_09.degree())

max_degree = max(degrees.values())
node_sizes = [500 * (deg / max_degree) for deg in degrees.values()]

fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')

# Para las aristas:
for edge in graph_09.edges():
    x_vals = [coord[edge[0], 0], coord[edge[1], 0]]
    y_vals = [coord[edge[0], 1], coord[edge[1], 1]]
    z_vals = [coord[edge[0], 2], coord[edge[1], 2]]
    ax.plot(x_vals, y_vals, z_vals, c='b', alpha=0.5, linewidth=0.7)
```

```

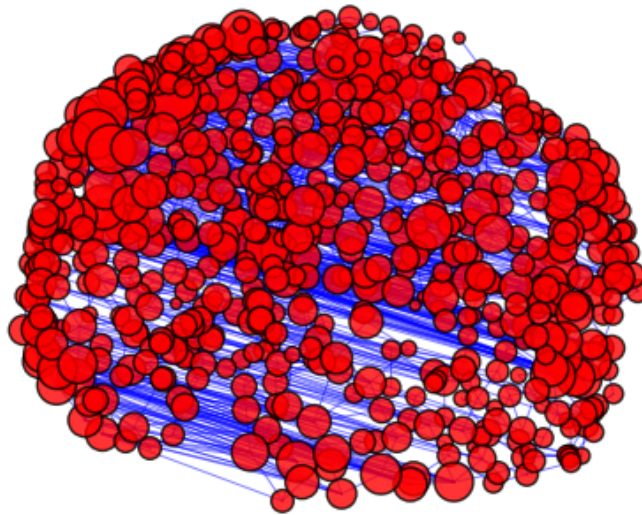
ax.scatter(
    coord[:, 0], coord[:, 1], coord[:, 2],
    c='r', s=node_sizes, alpha=0.8, edgecolors='k'
)

ax.set_title("Nodo proporcional al valor del grado (0.09)")
ax.axis("off")

plt.show()

```

Nodo proporcional al valor del grado (0.09)



4. En función de la matriz de emparejamiento (correlación de la matriz de adyacencia), establecer una participación de los nodos en módulos. Escoger el número de módulos que creas

conveniente y justificar porque escogiste ese número

```
[10]: from sklearn.cluster import SpectralClustering

graph = nx.from_numpy_array(coactivation_matrix)
# Rango de 2 a 10 módulos, esto evaluando la modularidad, para saber que tan
  ↳ bien separados están los módulos dentro de una red, estos números aseguran
  ↳ que los módulos capturan una estructura coherente en la red sin fragmentarla
range_clusters = range(2, 11)
modularity_scores = []
best_partition = None

# Calcular con diferentes números de módulos
for n_clusters in range_clusters:
    clustering = SpectralClustering(
        n_clusters=n_clusters,
        affinity='precomputed',
        random_state=42
    ).fit(coactivation_matrix)

    labels = clustering.labels_

    partition_spectral = {i: label for i, label in enumerate(labels)}
    modularity = nx.algorithms.community.quality.modularity(graph, [
        [node for node, cluster in partition_spectral.items() if cluster == c]
        for c in np.unique(labels)
    ])

    modularity_scores.append(modularity)

    if not best_partition or modularity > max(modularity_scores[:-1]):
        best_partition = partition_spectral

optimal_clusters = range_clusters[np.argmax(modularity_scores)]

optimal_clusters, max(modularity_scores)
```

```
[10]: (8, 0.4448068669431869)
```

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

```
[69]: import scipy.io as sio
import networkx as nx
import numpy as np

# Cargar los datos
```

```

mat_data = sio.loadmat(r"C:\Users\karim\OneDrive\Documentos\Python\
↳Scripts\Coactivation_matrix.mat")
coactivation_matrix = mat_data['Coactivation_matrix']

# Crear el grafo
G = nx.from_numpy_array(coactivation_matrix)

# Identificar el conjunto de Rich Club
rich_club_nodes = [node for node, degree in G.degree() if degree > np.mean([deg
↳for _, deg in G.degree()])]
print(f"Nodos en el Rich Club: {len(rich_club_nodes)}")
print(f"Algunos nodos del Rich Club: {rich_club_nodes[:10]}")

```

Nodos en el Rich Club: 255

Algunos nodos del Rich Club: [6, 7, 11, 16, 18, 19, 20, 22, 37, 38]

0.1.1 Discusión

Las áreas del cerebro con mayor numero de conexiones, conocidas como hubs, suelen tener un apepl central en la integración y transmisión de información entre distintas regiones. Entre algunas de estas estructuras relevantes se encuentran: - la corteza prefrontal medial (mPFC): que cumple su función en la toma de desiciones, planificación y regulación emocional - Precúneo: procesa imágenes mentales y se encarga de la autoconsciencia - Corteza cingulada posterior (PCC): participa en la memoria y en la regulación de las redes por defecto. - Corteza parietal superior (SPL): permite la atenci"n y la percepción espacial - Tálamo: integración sensorial y coordinación motora - Cortezas visual primaria y secundaria (V1, V2): procesamiento de estímulos visuales

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

```

[ ]: import networkx as nx
import scipy.io as sio
import numpy as np
mat_data = sio.loadmat(r"C:\Users\karim\OneDrive\Documentos\Python\
↳Scripts\Coactivation_matrix.mat")
coactivation_matrix = mat_data['Coactivation_matrix']
G = nx.from_numpy_array(coactivation_matrix)
rich_club_nodes = [node for node, degree in G.degree() if degree > np.mean([deg
↳for _, deg in G.degree()])]

# Crear un grafo sin los nodos del Rich Club

G_without_rich_club = G.copy()
G_without_rich_club.remove_nodes_from(rich_club_nodes)

# Función para calcular propiedades topológicas
def calculate_graph_properties(graph):

```

```

degree = np.mean([deg for _, deg in graph.degree()])
clustering_coeff = nx.average_clustering(graph)
small_world_coeff = nx.sigma(graph) if nx.is_connected(graph) else None
closeness_centrality = np.mean(list(nx.closeness_centrality(graph).
↪values()))
betweenness_centrality = np.mean(list(nx.betweenness_centrality(graph).
↪values()))

return {
    "degree": degree,
    "clustering_coefficient": clustering_coeff,
    "small_world_coefficient": small_world_coeff,
    "closeness_centrality": closeness_centrality,
    "betweenness_centrality": betweenness_centrality,
}

# Comparar propiedades
original_properties = calculate_graph_properties(G)
rich_club_removed_properties = calculate_graph_properties(G_without_rich_club)

print("Propiedades del grafo original:", original_properties)
print("Propiedades sin Rich Club:", rich_club_removed_properties)

```

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

```

[74]: # Identificar los nodos con mayor intermediación
betweenness = nx.betweenness_centrality(G)
sorted_nodes = sorted(betweenness, key=betweenness.get, reverse=True)

# Eliminar el 10% y 50% de los nodos con mayor intermediación
porcentajes = [0.1, 0.5]
for p in porcentajes:
    num_remove = int(len(sorted_nodes) * p)
    nodes_to_remove = sorted_nodes[:num_remove]

    G_modified = G.copy()
    G_modified.remove_nodes_from(nodes_to_remove)

    modified_properties = calculate_graph_properties(G_modified)
    print(f"Propiedades al eliminar el {int(p*100)}% de nodos con mayor_
↪intermediación:", modified_properties)

```

Propiedades al eliminar el 10% de nodos con mayor intermediación: {'degree': np.float64(1.5234782608695652), 'clustering_coefficient': 0.0028985507246376808, 'small_world_coefficient': None, 'closeness_centrality':


```

np.float64(0.013171426233179032), 'betweenness_centrality':
np.float64(0.007039707612126038)}
Propiedades al eliminar el 50% de nodos con mayor intermediación: {'degree':
np.float64(0.46394984326018807), 'clustering_coefficient': 0.0,
'small_world_coefficient': None, 'closeness_centrality':
np.float64(0.0015148229201579885), 'betweenness_centrality':
np.float64(7.463356319848853e-07)}

```

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

```

[77]: def generar_modelo_nulo_aleatorio(num_nodos, num_conexiones):
    # Crear un grafo vacío
    G = nx.gnm_random_graph(num_nodos, num_conexiones)
    return G

def plot_grafo_3d(G):
    pos = nx.spring_layout(G, dim=3) # Distribución de los nodos en 3D
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')

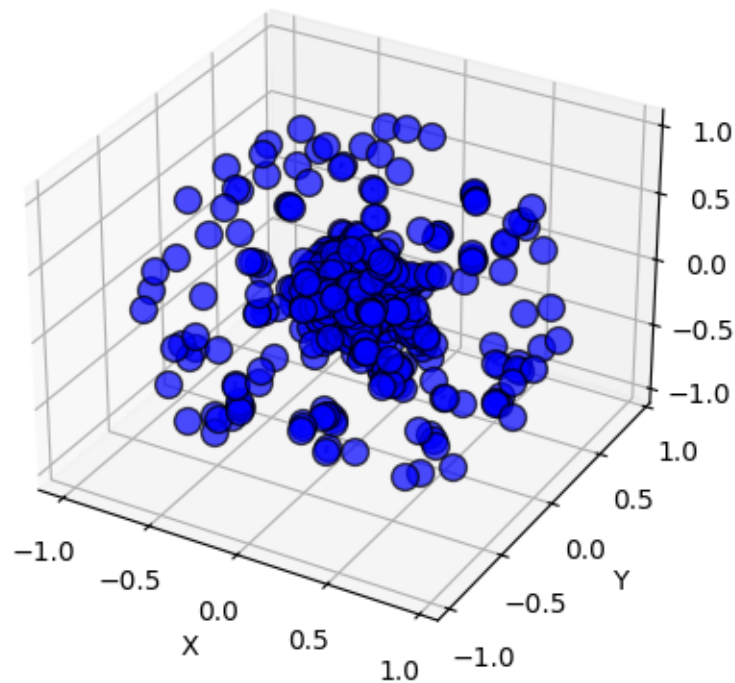
    for key, value in pos.items():
        xi, yi, zi = value
        ax.scatter(xi, yi, zi, color='b', s=100, edgecolors='k', alpha=0.7)

    for edge in G.edges():
        x = np.array((pos[edge[0]][0], pos[edge[1]][0]))
        y = np.array((pos[edge[0]][1], pos[edge[1]][1]))
        z = np.array((pos[edge[0]][2], pos[edge[1]][2]))
        ax.plot(x, y, z, color='k', alpha=0.5)

    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')
    plt.show()

num_nodos = 638
num_conexiones = 638
G_mismos = generar_modelo_nulo_aleatorio(num_nodos, num_conexiones)
plot_grafo_3d(G)

```



```
[78]: G_original = nx.from_numpy_array(adj_matrix)
def calcular_propiedades(grafo):
    propiedades = {
        "Número de nodos": grafo.number_of_nodes(),
        "Número de aristas": grafo.number_of_edges(),
        "Grado promedio": np.mean([d for n, d in grafo.degree()]),
        "Coeficiente de agrupamiento promedio": nx.average_clustering(grafo),
        "Longitud promedio de caminos más cortos": nx.
↪average_shortest_path_length(grafo) if nx.is_connected(grafo) else np.nan,
        "Eficiencia global": nx.global_efficiency(grafo),
    }
    return propiedades

# Calcular propiedades del grafo original y del modelo nulo
propiedades_original = calcular_propiedades(G_original)
propiedades_nulo = calcular_propiedades(G_mismos)

propiedades_original, propiedades_nulo
```

```
[78]: ({'Número de nodos': 638,
        'Número de aristas': 18625,
        'Grado promedio': np.float64(58.38557993730407),
        'Coeficiente de agrupamiento promedio': 0.3844533292242753,
```

```

'Longitud promedio de caminos más cortos': 2.2148737961545844,
'Eficiencia global': 0.49492420551600974},
{'Número de nodos': 638,
'Número de aristas': 638,
'Grado promedio': np.float64(2.0),
'Coeficiente de agrupamiento promedio': 0.004440961337513061,
'Longitud promedio de caminos más cortos': nan,
'Eficiencia global': 0.09115706765550627})

```

En el grafo del cerebro tenemos un grado promedio más alto, lo que indica que los nodos están muchos más conectados entre sí, a diferencia del modelo nulo donde cada nodo está conectado solo un número mínimo de nodos. Otro punto a favor que tiene el modelo cerebral es que tiene un coeficiente de agrupamiento mayor, es decir que tiene una estructura modular y el modelo nulo tiene una red más dispersa y aleatoria. En general, el modelo nulo al tener una red más aleatoria, no tiene características funcionales y estructurales si lo comparamos con el del cerebro que si tiene redes complejas con alta cohesión local y eficiencia global.

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

```

[51]: import scipy.io
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt

adj_matrix = contents['Coactivation_matrix']

# Crear el grafo original usando NetworkX
G = nx.from_numpy_array(adj_matrix)

#El modelo nulo conservando la distribución de grado
G_nulo = G.copy()
n_iteraciones = 500
nx.double_edge_swap(G_nulo, nswap=n_iteraciones, max_tries=n_iteraciones * 10)

# Para graficar un grafo en 3D
def plot_3d_graph(ax, graph, title):
    pos = nx.spring_layout(graph, dim=3, seed=42) # Posiciones en 3D

    # Coordenadas de los nodos
    x_nodes = np.array([pos[node][0] for node in graph.nodes()])
    y_nodes = np.array([pos[node][1] for node in graph.nodes()])
    z_nodes = np.array([pos[node][2] for node in graph.nodes()])

    # Dibujar nodos
    ax.scatter(x_nodes, y_nodes, z_nodes, c='r', s=10, alpha=0.8)

    # Dibujar aristas

```

```

for edge in graph.edges():
    x_coords = [pos[edge[0]][0], pos[edge[1]][0]]
    y_coords = [pos[edge[0]][1], pos[edge[1]][1]]
    z_coords = [pos[edge[0]][2], pos[edge[1]][2]]
    ax.plot(x_coords, y_coords, z_coords, c='b', alpha=0.3)

# Ajustes del gráfico
ax.set_title(title)

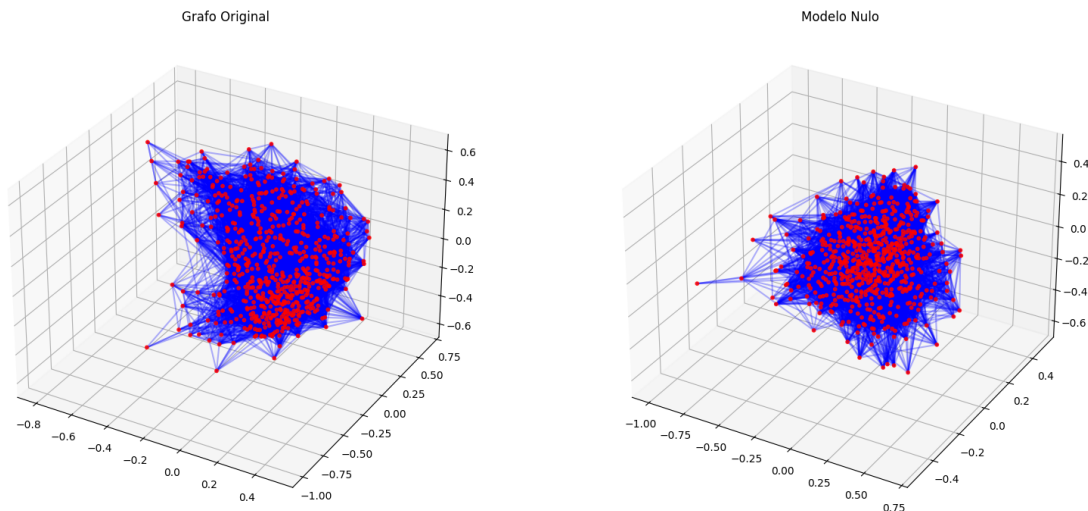
# 4. Crear una figura y subgráficos 3D para el grafo original y el modelo nulo
fig = plt.figure(figsize=(17, 7))

# Grafo original
ax1 = fig.add_subplot(121, projection='3d')
plot_3d_graph(ax1, G, 'Grafo Original')

# Grafo modelo nulo
ax2 = fig.add_subplot(122, projection='3d')
plot_3d_graph(ax2, G_nulo, 'Modelo Nulo')

# Mostrar el gráfico
plt.tight_layout()
plt.show()

```



A pesar de que ambos grafos tienen el mismo número de conexiones, nodos y grado, el coeficiente de agrupamiento del cerebro es más alto, lo que quiere decir que las conexiones en el cerebro tienden a formar más triángulos (grupos mejor conectados) que en el modelo nulo. Otro aspecto que podemos observar es que el grafo del cerebro tiene una mayor longitud promedio de caminos más cortos a comparación del modelo nulo, lo que indica que el modelo nulo es un poco más eficiente en término de conectividad global. En sí, el grafo original del cerebro tiende a formar comunidades funcionales,

pero el modelo nulo es más eficiente globalmente, lo cual es característico de redes aleatorias ya que no poseen restricciones relacionadas a principios biológicos o fisiológicos.

10. 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

```
[68]: def generar_grafo_nulo_3d(num_nodos, num_conexiones):
    # Crear un grafo vacío
    G = nx.Graph()

    # Añadir nodos con posiciones en 3D
    posiciones = {i: (np.random.random(), np.random.random(), np.random.
    random()) for i in range(num_nodos)}
    for i in range(num_nodos):
        G.add_node(i, pos=posiciones[i])

    # Calcular distancias entre todos los pares de nodos
    distancias = {(i, j): np.linalg.norm(np.array(posiciones[i]) - np.
    array(posiciones[j]))
    for i in range(num_nodos) for j in range(i+1, num_nodos)}

    # Ordenar las distancias de menor a mayor
    distancias_ordenadas = sorted(distancias.items(), key=lambda x: x[1])

    # Añadir conexiones según la distancia, hasta alcanzar el número deseado
    conexiones_agregadas = 0
    for (i, j), d in distancias_ordenadas:
        if conexiones_agregadas < num_conexiones:
            G.add_edge(i, j)
            conexiones_agregadas += 1
        else:
            break

    return G

def plot_grafo_3d(G):
    pos = nx.get_node_attributes(G, 'pos')
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')

    for key, value in pos.items():
        xi, yi, zi = value
        ax.scatter(xi, yi, zi, color='r', s=10, edgecolors='b', alpha=0.7)

    for edge in G.edges():
        x = np.array((pos[edge[0]][0], pos[edge[1]][0]))
        y = np.array((pos[edge[0]][1], pos[edge[1]][1]))
```

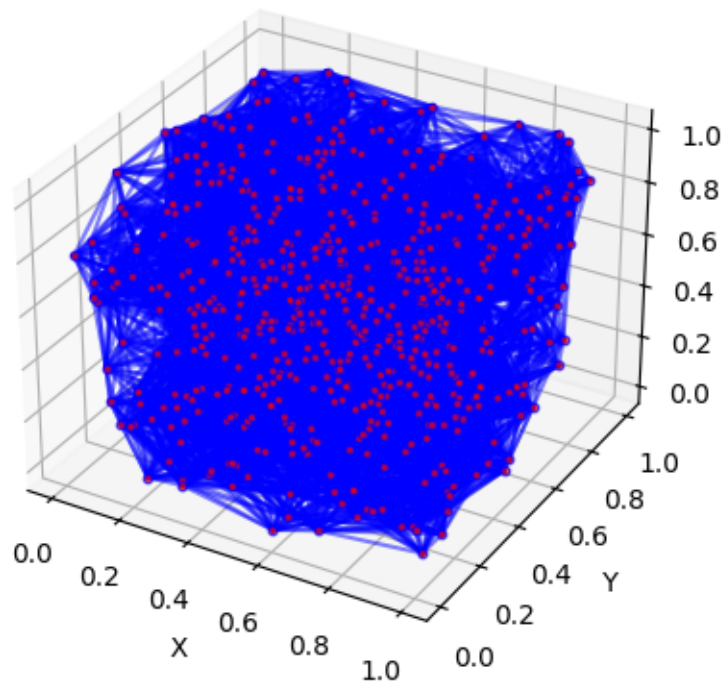
```

z = np.array((pos[edge[0]][2], pos[edge[1]][2]))
ax.plot(x, y, z, color='b', alpha=0.5)

ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
plt.show()

num_nodos= 638
num_conexiones = 18625
G = generar_grafo_nulo_3d(num_nodos, num_conexiones)
plot_grafo_3d(G)

```



La distribución es diferente a la del grafo del cerebro ya que en este grafo, a pesar de que este grafo tiene el mismo número de nodos y conexiones, las conexiones se añaden comenzando por las distancias más cortas dada su probabilidad, por eso vemos más conexiones en el centro.

Las conexiones a larga distancia en el cerebro permiten una comunicación entre diferentes regiones cerebrales que permitan la integración y coordinación de funciones complejas que requieren de diferentes funciones como la memoria, atención y toma de decisiones. Si elimináramos las conexiones de larga distancia, afectaríamos significativamente estas funciones complejas y la eficiencia de la red cerebral.

11. Escribir una reseña de lo aprendido en el curso, incluyendo la importancia de conocer her-

ramientas de teoría de grafos para comprender la conectividad del cerebro (mínimo 200 palabras)

La teoría de grafos nos ayuda a analizar y comprender un poco sobre la dinámica que tiene el cerebro de manera matemática y sobretodo visual, este enfoque visual es uno de los aspectos que más se puede subrayar del curso, representar conceptos clave como los hubs en el conectoma nos ayuda a relacionarlos con funciones cerebrales fundamentales, así como identificar patrones de conectividad para identificar alguna posible disfunción cerebral. El cálculo de distancias cortas, el coeficiente de Rich Club y establecer umbrales, nos ayuda a determinar qué nodos son esenciales para que la red sea más eficiente, cuál es su jerarquía y evaluar la robustez del cerebro frente a distintos cambios o lesiones. A pesar de que es complejo entender del todo el proceso, desde el planteamiento del problema, qué es lo que queremos analizar, cómo lo realizaríamos, hasta la visualización gráfica, la información obtenida en si es muy valiosa para adentrarnos cada vez más a como que es que funcionan las redes neuronales, si tienen algún patrón y, en caso de tenerlo, poder predecirlo. Comprender cómo funciona la conectividad cerebral en procesos cognitivos simples o complejos también significa abrir una ventana al desarrollo de técnicas de rehabilitación basadas en el arreglo de la red funcional anómala del cerebro, a pesar de que es un campo poco explorado, cualquier avance es significativo.