PROYECTO FINAL - MODELOS

Estela Gil-Villegas Guevara

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import networkx as nx
import scipy.io
from matplotlib.animation import FuncAnimation, PillowWriter
from mpl_toolkits.mplot3d import Axes3D
from IPython.display import Image
```

EJERCICIO 1.

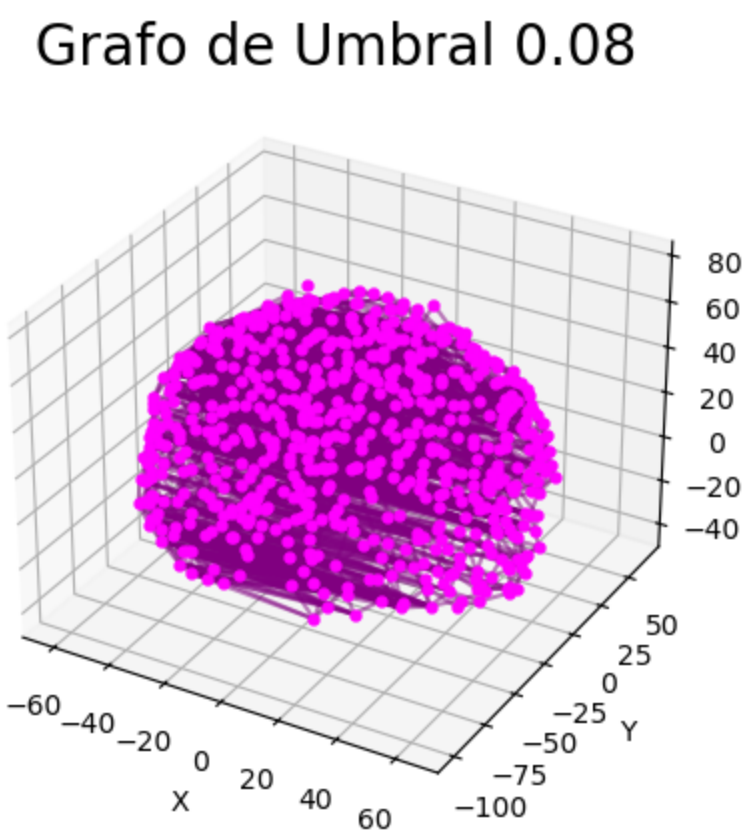
Definir grafos con la matriz estableciendo umbrales de coactivación de 0.8, 0.9 y 1 y graficar cada grafo. Añadir las coordenadas tridimensionales (incluidas en el archiv .mat.

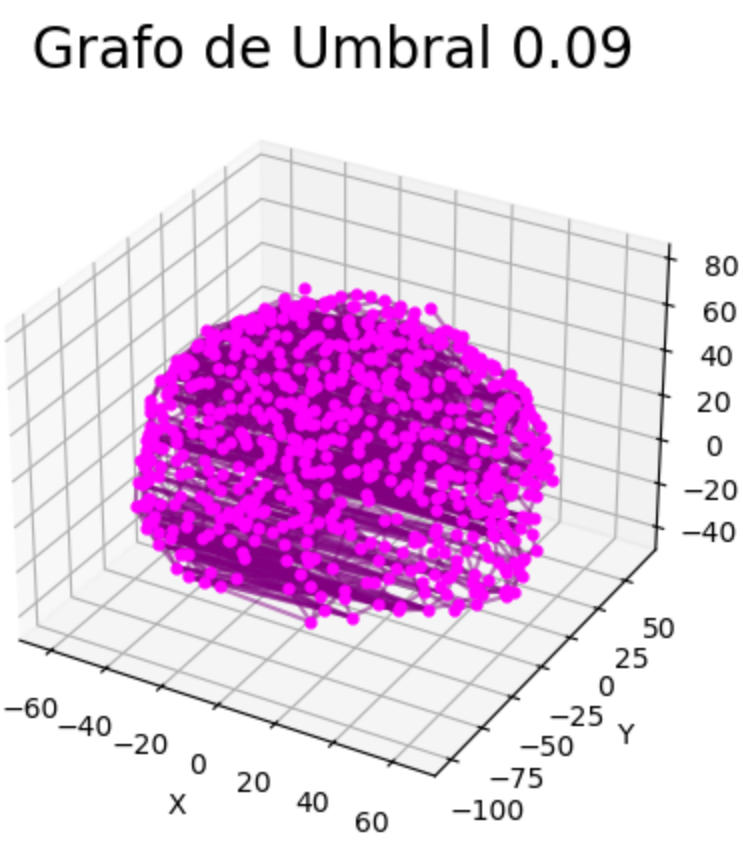
In [39]:
```
mat_data = scipy.io.loadmat('Coactivation_matrix')
mat_data
coact_mat = mat_data['Coactivation_matrix']
coords = mat_data['Coord']
x, y, z = coords[:, 0], coords[:, 1], coords[:, 2]
```

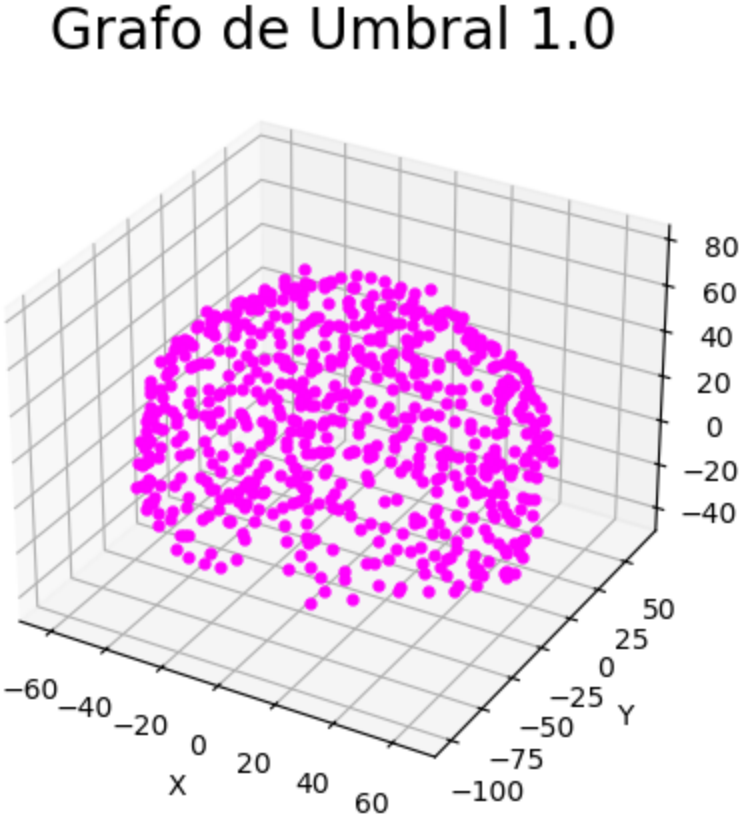
In [65]:
```
# CREAR GRAFO

def crear_graph(umbral, coact_mat, coords):
    adj_mat = (coact_mat >= umbral).astype(int)
    G = nx.Graph() # crear grafo de la matriz de adyacencia

    for i, coord in enumerate(coords):
        G.add_node(i, pos = coord) # para graficar en 3D con posiciones de nodos
    for i in range(adj_mat.shape[0]):
        for j in range(i+1, adj_mat.shape[1]): # no duplicar
            if adj_mat[i, j] == 1:
                G.add_edge(i, j)

    return G

# GRAFICAR GRAFO EN 3D

def graficar_3Dgraph(G, node_color = 'magenta', edge_color = 'purple'):
    pos = nx.get_node_attributes(G, 'pos')
    fig = plt.figure()
    ax = fig.add_subplot(111, projection = '3d')

    for node, (x, y, z) in pos.items(): # nodos
        ax.scatter(x, y, z, c ='magenta', s = 50, marker = '.')
```

```
    for edge in G.edges(): #aristas
        if edge[0] in pos and edge[1] in pos:
            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, c='purple', alpha = 0.5)

    ax.set_title(f'Grafo de Umbral {umbral}', fontsize= 20)
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')

# DEFINIR UMBRALES

umbrales = [0.08, 0.09, 1.0]
for umbral in umbrales:
    G = crear_graph(umbral, coact_mat, coords)
    graficar_3Dgraph(G)

print(f'Para el grafo {umbral}, existen las siguientes coordenadas de conexiones:\n
plt.show()
```

```
Para el grafo 1.0, existen las siguientes coordenadas de conexiones:
```




Grafo de Umbral 0.09
80
60
40
20
0
−20
−40
50
25
0
−25
−50
−75
−100
Y
−60
−40
−20
0
20
40
60
X

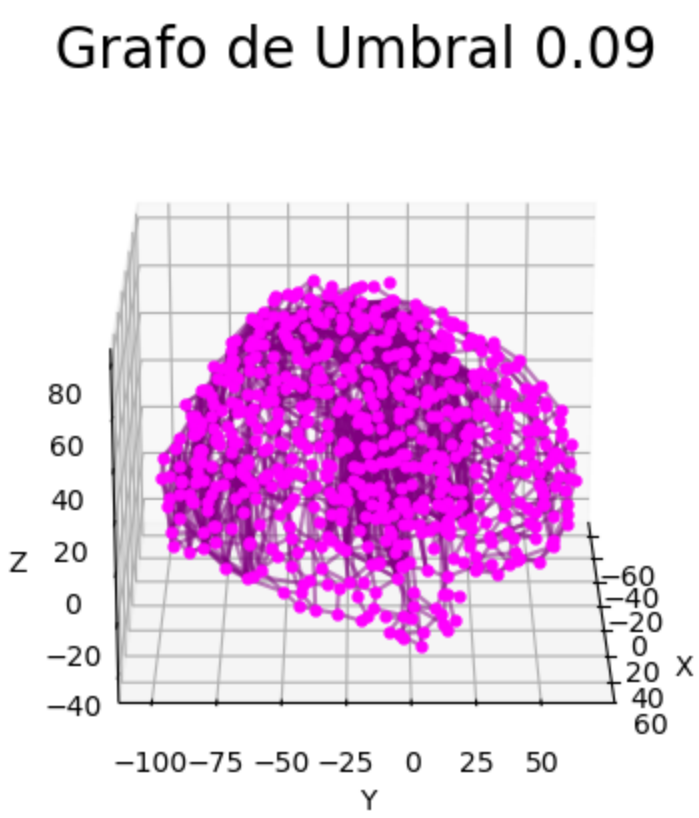

Grafo de Umbral 1.0
80
60
40
20
0
−20
−40
50
25
0
−25
−50
−75
−100
Y
−60
−40
−20
0
20
40
60
X

EJERCICIO 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.

In [64]:
```
mat_data = scipy.io.loadmat('Coactivation_matrix')
coact_mat = mat_data['Coactivation_matrix']
coords = mat_data['Coord']

umbral = 0.09
mat_filt = np.where(coact_mat >= umbral, coact_mat, 0)

G = nx.from_numpy_array(mat_filt) # grafo de matriz filtrada con umbral
pos = {i: (coords[i][0], coords[i][1], coords[i][2]) for i in range(len(coords))}
nx.set_node_attributes(G, pos, 'pos')

def graficar_3Dgraph(G, node_color = 'magenta', edge_color = 'purple'):
    pos = nx.get_node_attributes(G, 'pos')
    fig = plt.figure()
    ax = fig.add_subplot(111, projection = '3d')

    for node, (x, y, z) in pos.items():
        ax.scatter(x, y, z, c=node_color, s=50, marker='.')
    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, c=edge_color, alpha=0.5)

    ax.set_title(f'Grafo de Umbral {umbral}', fontsize = 20)
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')

    return fig, ax

fig, ax = graficar_3Dgraph(G) # graficar grafo

def init(): # iniciar visualización
    ax.view_init(elev=20, azim=0)
    return fig,
def animate(frame): # animación de rotación
    ax.view_init(elev=20, azim=frame)
    return fig,

animation = FuncAnimation(fig, animate, init_func=init, frames = 30, interval = 100
gif = (r"C:\Users\estel\OneDrive\Escritorio\NEUROCIENCIAS\QUINTO SEMESTRE\MODELOS\m
animation.save(gif, writer = PillowWriter(fps = 20))
Image(gif)
```

Out[64]: <IPython.core.display.Image object>



EJERCICIO 3.

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

In [ ]:
```
mat_data = scipy.io.loadmat('Coactivation_matrix')
coact_mat = mat_data['Coactivation_matrix']
coords = mat_data['Coord']

umbral = 0.09
mat_filt = np.where(coact_mat >= umbral, coact_mat, 0)
G = nx.from_numpy_array(mat_filt) # crear grafo
pos = {i: (coords[i][0], coords[i][1], coords[i][2]) for i in range(len(coords))}
nx.set_node_attributes(G, pos, 'pos') # establecer coordenadas de los nodos

# ENCONTRAR HUBS

def find_hubs_graphs(G): # función para encontrar hubs
    # Encontrar los hubs
    grados = dict(G.degree())  # grado de nodos
    hub_threshold = sorted(grados.values(), reverse=True)[5]  # umbral basado en lo
    hubs = [node for node, grado in grados.items() if grado >= hub_threshold]

    # Establecer proporcionalidad entre tamaño de nodos y grado
    node_sizes = [grados[node] * 100 for node in G.nodes()]  # Tamaño proporcional

    pos = nx.get_node_attributes(G, 'pos')
    figura = plt.figure(figsize=(10, 10))
    Ax = figura.add_subplot(projection='3d')
```

```
    for node, (x, y, z) in pos.items(): # graficar nodos (tamaño proporcional al gr
        Ax.scatter(x, y, z, color='orange', s=grados[node] * 10)  # tamaño ajustado
    for edge in G.edges(): # graficar aristas
        node1, node2 = edge
        x_coords = [pos[node1][0], pos[node2][0]]
        y_coords = [pos[node1][1], pos[node2][1]]
        z_coords = [pos[node1][2], pos[node2][2]]
        Ax.plot(x_coords, y_coords, z_coords, c='hotpink', alpha=0.5)

    for hub in hubs:
        x, y, z = pos[hub]
        Ax.scatter(x, y, z, color='cornflowerblue', s=grados[hub] * 100) # aumento

    Ax.set_title(f'Grafo de Nodos con Hubs y Tamaño Proporcional al Grado', fontsiz
    Ax.set_xlabel('X', fontsize=17)
    Ax.set_ylabel('Y', fontsize=17)
    Ax.set_zlabel('Z', fontsize=17)

    plt.show()
    print(f"Hay {len(hubs)} hubs: {hubs}")
    print(f'El umbral de grado para los hubs es: {hub_threshold}')

find_hubs_graphs(G)

grados = dict(G.degree())
Umbral = np.percentile(list(grados.values()), 99)
Hubs = [node for node, grado in grados.items() if grado >= Umbral]
print(f"Existen {len(Hubs)} Hubs con un umbral del 99º percentil. Son los siguiente
```

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 númer de módulos que creas conveniente y justificar por qué escogiste ese número.

In [ ]:

```
from networkx.algorithms.community import greedy_modularity_communities

threshold = 0.2
mat_adj = np.where(coact_mat >= threshold, coact_mat, 0)
corr_mat = np.corrcoef(adj_mat)

plt.figure(figsize=(15, 5))
plt.subplot(1, 2, 1)
sns.heatmap(coact_mat, cmap="inferno", cbar_kws={"label": "Intensidad"})
plt.title("Matriz Original")
plt.subplot(1, 2, 2)
sns.heatmap(corr_mat, cmap="inferno", cbar_kws={"label": "Correlación"})
plt.title("Matriz de Correlación")
plt.show()

G = nx.from_numpy_array(adj_mat)
communities = list(greedy_modularity_communities(G))
partition = {}
```

```
for i, community in enumerate(communities):
    for node in community:
        partition[node] = i

modularity = nx.algorithms.community.quality.modularity(G, communities)
node_colors = [partition[node] for node in G.nodes()]

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

scatter = ax.scatter(
    x, y, z, c=node_colors, cmap="tab10", s=100, edgecolors="black", label="Nodos"
)

for edge in G.edges():
    node1, node2 = edge
    x_edge = [x[node1], x[node2]]
    y_edge = [y[node1], y[node2]]
    z_edge = [z[node1], z[node2]]
    ax.plot(x_edge, y_edge, z_edge, c="gray", alpha=0.5)

ax.set_title(f"Grafo Particionado en {len(communities)} Módulos\nModularidad: {modu
ax.set_xlabel("X")
ax.set_ylabel("Y")
ax.set_zlabel("Z")
plt.colorbar(scatter, ax=ax, label="Módulo")
plt.show()

print(f"Se identificaron {len(communities)} módulos.")
print(f"La modularidad obtenida es {modularity:.4f}. Esto sugiere una buena divisió
```

JUSTIFICACIÓN

La moduladridad es una medida que indica la CALIDAD de una partición del grafo en módulos. Alta modularidad implica que los nodos dentro de un módulo están más conectados entre sí que con los nodos de OTROS módulos. En este caso, se identificaron 453 módulos, con una modularidad des 0.984, lo que indicaría una optimización de las conexiones internas.(dentro de los módulos). .

EJERCICIO 5.

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

In [57]:
```
G2 = nx.from_numpy_array(coact_mat)
grados = dict(G2.degree())

degree_th = np.percentile(list(grados.values()), 90)
rc = [nodo for nodo, grado in grados.items() if grado >= degree_th]
print(f"Rich Club (grado ≥ {degree_th}): {rc}")

rc_degree = {nodo: grados[nodo] for nodo in rc}  # grado de nodos del RC
```

```
rc_size = len(rc)  # tamaño del RC
print(f"Tamaño del Rich Club: {rc_size} nodos")
print(f"Grados de los nodos del Rich Club: {rc_degree}")
```

```
Rich Club (grado ≥ 108.0): [7, 16, 19, 37, 38, 41, 42, 43, 44, 70, 97, 100, 121, 12
4, 126, 130, 135, 154, 186, 193, 202, 230, 231, 235, 237, 275, 286, 327, 328, 330, 3
31, 334, 344, 345, 346, 350, 353, 356, 365, 397, 399, 400, 405, 407, 410, 416, 418,
421, 452, 465, 477, 481, 482, 485, 488, 491, 494, 495, 496, 500, 621, 622, 623, 629,
630]
Tamaño del Rich Club: 65 nodos
Grados de los nodos del Rich Club: {7: 124, 16: 124, 19: 109, 37: 108, 38: 149, 41:
114, 42: 120, 43: 123, 44: 126, 70: 115, 97: 119, 100: 120, 121: 141, 124: 112, 126:
115, 130: 115, 135: 121, 154: 108, 186: 108, 193: 115, 202: 129, 230: 164, 231: 128,
235: 160, 237: 135, 275: 124, 286: 131, 327: 119, 328: 150, 330: 179, 331: 122, 334:
121, 344: 115, 345: 111, 346: 140, 350: 151, 353: 126, 356: 153, 365: 113, 397: 136,
399: 114, 400: 151, 405: 118, 407: 117, 410: 109, 416: 163, 418: 142, 421: 142, 452:
124, 465: 124, 477: 110, 481: 143, 482: 172, 485: 161, 488: 147, 491: 150, 494: 152,
495: 115, 496: 147, 500: 123, 621: 136, 622: 140, 623: 115, 629: 143, 630: 129}
```

IMPLICACIONES ANATÓMICAS Y FUNCIONALES

El Rich Club es un conjunto de nodos altamente conectados entre sí. En redes cerebrales, estos nodos suelen ser regiones clave que tienen un papel fundamental en la integración de la información entre diferentes áreas del cerebro. Los nodos del Rich Club suelen estar relacionados con funciones cognitivas altas o de mayor orden, como el procesamiento de la memoria, la toma de decisiones, y el control motor.

Las implicaciones funcionales de estos nodos son cruciales en la dinámica de la red cerebral. Se considera que los nodos del Rich Club son los más eficientes en la transferencia de información dentro de la red, lo que puede estar relacionado con la coordinación entre distintas regiones cerebrales en tareas complejas. En términos anatómicos, estos nodos a menudo corresponden a áreas del cerebro que están altamente interconectadas y tienen una estructura densa de fibras (entre ellas, las cortezas de asociación).

EJERCICIO 6.

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

In [ ]:
```
# COMPARACIONES TOPOLÓGICAS:

grados = dict(G2.degree())
degree_th = np.percentile(list(grados.values()), 90)
rc = [nodo for nodo, grado in grados.items() if grado >= degree_th]
degree_size = {0: 0.50, 1: 20, 2: 50, 3: 100, 4: 200}
print(f"Rich Club (grado ≥ {degree_th}): {rc}")

rc_degree = {nodo: grados[nodo] for nodo in rc}  # grado de nodos del RC
rc_size = len(rc)  # tamaño del RC
```

```
print(f"Tamaño del Rich Club: {rc_size} nodos")
print(f"Grados de los nodos del Rich Club: {rc_degree}")

figura = plt.figure(figsize=(10, 10))
ax = figura.add_subplot(projection='3d')

for nodo in grados.keys():
    valor = grados[nodo]
    tamaño = grado_tamaño.get(valor, 10)
    color = 'red' if nodo in rich_club else 'orange'
    Ax.scatter(x[nodo], y[nodo], z[nodo], color=color, s=tamaño)

for edge in G2.edges():
    node1, node2 = edge
    x_coords = [x[node1], x[node2]]
    y_coords = [y[node1], y[node2]]
    z_coords = [z[node1], z[node2]]
    if node1 in rich_club and node2 in rich_club:
        ax.plot(x_coords, y_coords, z_coords, c='red', alpha=0.7, linewidth=1.5)
    else:
        ax.plot(x_coords, y_coords, z_coords, c='dimgray', alpha=0.5)

ax.set_title('Grafo con Rich Club', fontsize=30)
ax.set_xlabel('X', fontsize=17)
ax.set_ylabel('Y', fontsize=17)
ax.set_zlabel('Z', fontsize=17)
plt.show()

grados = dict(G2.degree())
degree_th = np.percentile(list(grados.values()), 90)
rc = [nodo for nodo, grado in grados.items() if grado >= grado_umbral]

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

G_wo_rc = G2.copy()
G_wo_rc.remove_nodes_from(rc)

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

figura = plt.figure(figsize=(10, 10))
ax = figura.add_subplot(projection='3d')

for nodo in G_wo_rc.nodes():
    valor = grados.get(nodo, 0)
    tamaño = grado_tamaño.get(valor, 10)
    ax.scatter(x[nodo], y[nodo], z[nodo], color='orange', s=tamaño)

for edge in G_wo_rc.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('Grafo sin Rich Club', fontsize=30)
ax.set_xlabel('X', fontsize=17)
```

```
ax.set_ylabel('Y', fontsize=17)
ax.set_zlabel('Z', fontsize=17)
plt.show()

graficar_3Dgraph(G2, f'Grafo con Rich Club (grado ≥ {degree_th})') # grafo original

# Eliminar nodos del rich club y plotear
G_wo_rc = G2.copy()
G_wo_rc.remove_nodes_from(rc)

# Grafo sin Rich Club
graficar_3Dgraph(G_wo_rc, 'Grafo SIN RC')
```

In [ ]:

```
# COMPARACIÓN DE PROPIEDADES:

grados = dict(G2.degree())
degree_th = np.percentile(list(grados.values()), 90)
rc = [nodo for nodo, grado in grados.items() if grado >= degree_th] # cálculo RC

G_wo_rc = G2.copy()
G_wo_rc.remove_nodes_from(rc)

def calculo_propiedades(G, nombre):
    propiedades = {
        "número_nodos": G.number_of_nodes(),
        "número_aristas": G.number_of_edges(),
        "grado_prom": sum(dict(G.degree()).values()) / G.number_of_nodes(),
        "coef_clustering promedio": nx.average_clustering(G),
        "centralidad de cercanía": np.mean(list(nx.closeness_centrality(G).values()),
        "centralidad de intermediación": np.mean(list(nx.betweenness_centrality(G).
        "coef_small_world": nx.transitivity(G)
    }

    print(f"Propiedades del grafo {nombre}:")
    for key, value in propiedades.items():
        print(f"  {key}: {value}")
    print()
    return propiedades

# Con RC
propiedades_original = calculo_propiedades(G2, 'Original')

# Sin RC
propiedades_sin_rich_club = calculo_propiedades(G_wo_rc, 'Sin RC')
```

DISCUSIÓN

La presencia de nodos altamente conectados afetca la estructura y funcionalidad de la red. La eliminación de nodos del Rich Club (RC) resulta en una red menos densa, lo que podría limitar la capacidad de integración global de la red. Esto sugiere que los hubs y sus conexiones desempeñan un papel crítico en mantener la conectividad general. Además, los nodos del RC no solo están altamente conectados, sino que también pueden actuar como "concentradores" de flujo de información en la red. Su eliminación disminuye la conectividad

local y global, lo que reduce las oportunidades para interacciones locales densas, lo que puede afectar la redundancia de información y la robustez a nivel local.

EJERCICIO 7.

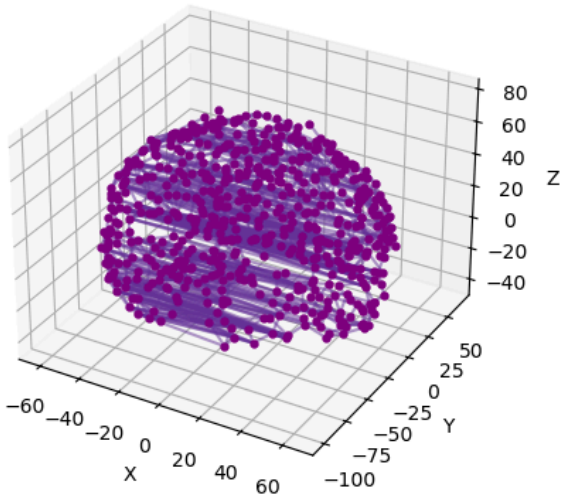
Quitar 10%-50% de los nodos con mayor medida de intermediación y describir cómo cambian las propiedades topológicas del grafo, hacer comparativas del grado coeficiente de cluster, coeficiente de mundo pequeño y las medidas de centralid d (cerca ía, intermediación)

In [81]:
```
def eliminar_nodos(G, porcentaje): # eliminar porcentaje de nodos
    intermediacion = nx.betweenness_centrality(G)
    nodos_ordenados = sorted(intermediacion, key=intermediacion.get, reverse=True)
    num_nodes_delete = int(len(nodos_ordenados) * porcentaje / 100)
    nodes_delete = nodos_ordenados[:num_nodes_delete]

    G_modificado = G.copy()
    G_modificado.remove_nodes_from(nodes_delete)
    return G_modificado, nodes_delete

# modificar función de graficar grafo en 3D

def graficar_3Dgraph(G, x = None, y = None, z = None, nodes_delete = None, node_col
    pos = nx.get_node_attributes(G, 'pos')
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')

    for node, (x, y, z) in pos.items():
        if nodes_delete and node in nodes_delete:
            ax.scatter(x, y, z, c='red', s=50, marker='o')  # Nodos eliminados en r
        else:
            ax.scatter(x, y, z, c="purple", s=50, marker='.')  # Nodos normales en

    for edge in G.edges():
        if edge[0] in pos and edge[1] in pos:
            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, c = "rebeccapurple", alpha=0.5)  # Color de las arista

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

    plt.show()

porcentajes = [10, 20, 30, 40, 50]
x, y, z = coords[:, 0], coords[:, 1], coords[:, 2]

propiedades_originales = calculo_propiedades(G, "grafo original")
print("Propiedades del grafo original:")
print(propiedades_originales)
```

```
for porcentaje in porcentajes:
    G_modificado, nodes_delete = eliminar_nodos(G, porcentaje)
    propiedades_modificadas = calculo_propiedades(G_modificado, "grafo modificado")
    print(f"\nPropiedades tras eliminar {porcentaje}% de los nodos:")
    print(propiedades_modificadas)

    graficar_3Dgraph(
        G_modificado,
        x, y, z,
        nodes_delete = nodes_delete,
        titulo = f"Grafo tras eliminar {porcentaje}% de nodos con mayor intermediac
    )
```

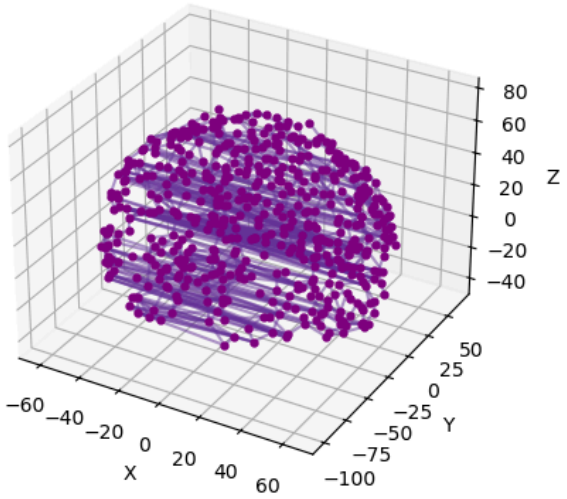
```
Propiedades del grafo grafo original:
  número_nodos: 638
  número_aristas: 1791
  grado_prom: 5.614420062695925
  coef_clustering promedio: 0.2785052819986504
  centralidad de cercanía: 0.14410899273204028
  centralidad de intermediación: 0.009627134336679834
  coef_small_world: 0.3451586655817738

Propiedades del grafo original:
{'número_nodos': 638, 'número_aristas': 1791, 'grado_prom': 5.614420062695925, 'coef
_clustering promedio': 0.2785052819986504, 'centralidad de cercanía': 0.144108992732
04028, 'centralidad de intermediación': 0.009627134336679834, 'coef_small_world': 0.
3451586655817738}
Propiedades del grafo grafo modificado:
  número_nodos: 575
  número_aristas: 1251
  grado_prom: 4.351304347826087
  coef_clustering promedio: 0.26065096256400605
  centralidad de cercanía: 0.10502884478729942
  centralidad de intermediación: 0.015025974434567928
  coef_small_world: 0.331411065056767


Propiedades tras eliminar 10% de los nodos:
{'número_nodos': 575, 'número_aristas': 1251, 'grado_prom': 4.351304347826087, 'coef
_clustering promedio': 0.26065096256400605, 'centralidad de cercanía': 0.10502884478
729942, 'centralidad de intermediación': 0.015025974434567928, 'coef_small_world':
0.331411065056767}
```

Grafo tras eliminar 10% de nodos con mayor intermediación



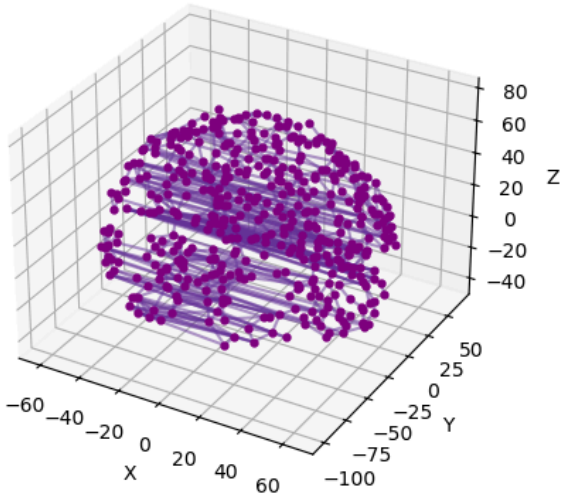

```
Propiedades del grafo grafo modificado:
  número_nodos: 511
  número_aristas: 939
  grado_prom: 3.675146771037182
  coef_clustering promedio: 0.2424004892693738
  centralidad de cercanía: 0.07423423198025483
  centralidad de intermediación: 0.01762977279616532
  coef_small_world: 0.33498976907337036


Propiedades tras eliminar 20% de los nodos:
{'número_nodos': 511, 'número_aristas': 939, 'grado_prom': 3.675146771037182, 'coef_
clustering promedio': 0.2424004892693738, 'centralidad de cercanía': 0.0742342319802
5483, 'centralidad de intermediación': 0.01762977279616532, 'coef_small_world': 0.33
498976907337036}
```

Grafo tras eliminar 20% de nodos con mayor intermediación



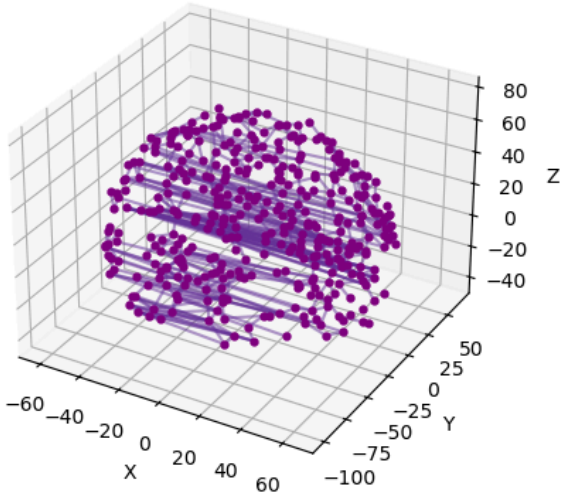
```
Propiedades del grafo grafo modificado:
  número_nodos: 447
  número_aristas: 712
  grado_prom: 3.185682326621924
  coef_clustering promedio: 0.2563371006324026
  centralidad de cercanía: 0.05099617054174734
  centralidad de intermediación: 0.02392776778147008
  coef_small_world: 0.3400174367916303


Propiedades tras eliminar 30% de los nodos:
{'número_nodos': 447, 'número_aristas': 712, 'grado_prom': 3.185682326621924, 'coef_
clustering promedio': 0.2563371006324026, 'centralidad de cercanía': 0.0509961705417
4734, 'centralidad de intermediación': 0.02392776778147008, 'coef_small_world': 0.34
00174367916303}
```

Grafo tras eliminar 30% de nodos con mayor intermediación



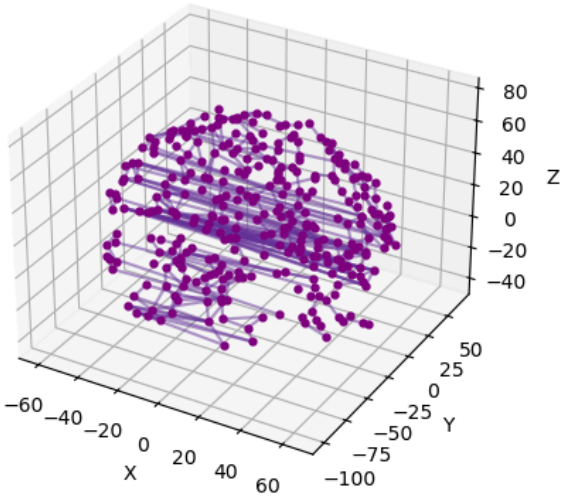

```
Propiedades del grafo grafo modificado:
  número_nodos: 383
  número_aristas: 524
  grado_prom: 2.7362924281984333
  coef_clustering promedio: 0.22687222843880805
  centralidad de cercanía: 0.030301707402235555
  centralidad de intermediación: 0.008450092358470775
  coef_small_world: 0.3543956043956044


Propiedades tras eliminar 40% de los nodos:
{'número_nodos': 383, 'número_aristas': 524, 'grado_prom': 2.7362924281984333, 'coef
_clustering promedio': 0.22687222843880805, 'centralidad de cercanía': 0.03030170740
2235555, 'centralidad de intermediación': 0.008450092358470775, 'coef_small_world':
0.3543956043956044}
```

Grafo tras eliminar 40% de nodos con mayor intermediación




```
Propiedades del grafo grafo modificado:
  número_nodos: 319
  número_aristas: 375
  grado_prom: 2.3510971786833856
  coef_clustering promedio: 0.20209732795939692
  centralidad de cercanía: 0.020176775727589722
  centralidad de intermediación: 0.0023531340530123443
  coef_small_world: 0.3497326203208556


Propiedades tras eliminar 50% de los nodos:
{'número_nodos': 319, 'número_aristas': 375, 'grado_prom': 2.3510971786833856, 'coef
_clustering promedio': 0.20209732795939692, 'centralidad de cercanía': 0.02017677572
7589722, 'centralidad de intermediación': 0.0023531340530123443, 'coef_small_world':
0.3497326203208556}
```

Grafo tras eliminar 50% de nodos con mayor intermediación



EJERCICIO 8.

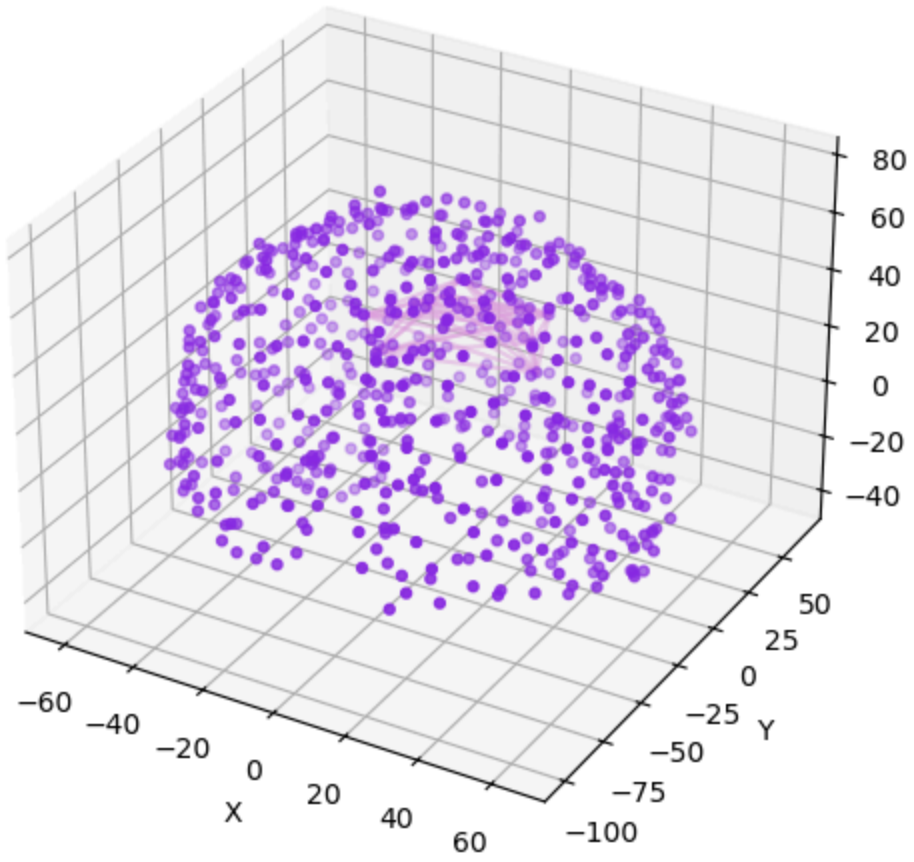
Generar un modelo nulo aleatorio donde se tenga el mismo número de nodos y el mismo número total de conexiones, y comparar sus propiedades con el grafo origina del cerebro

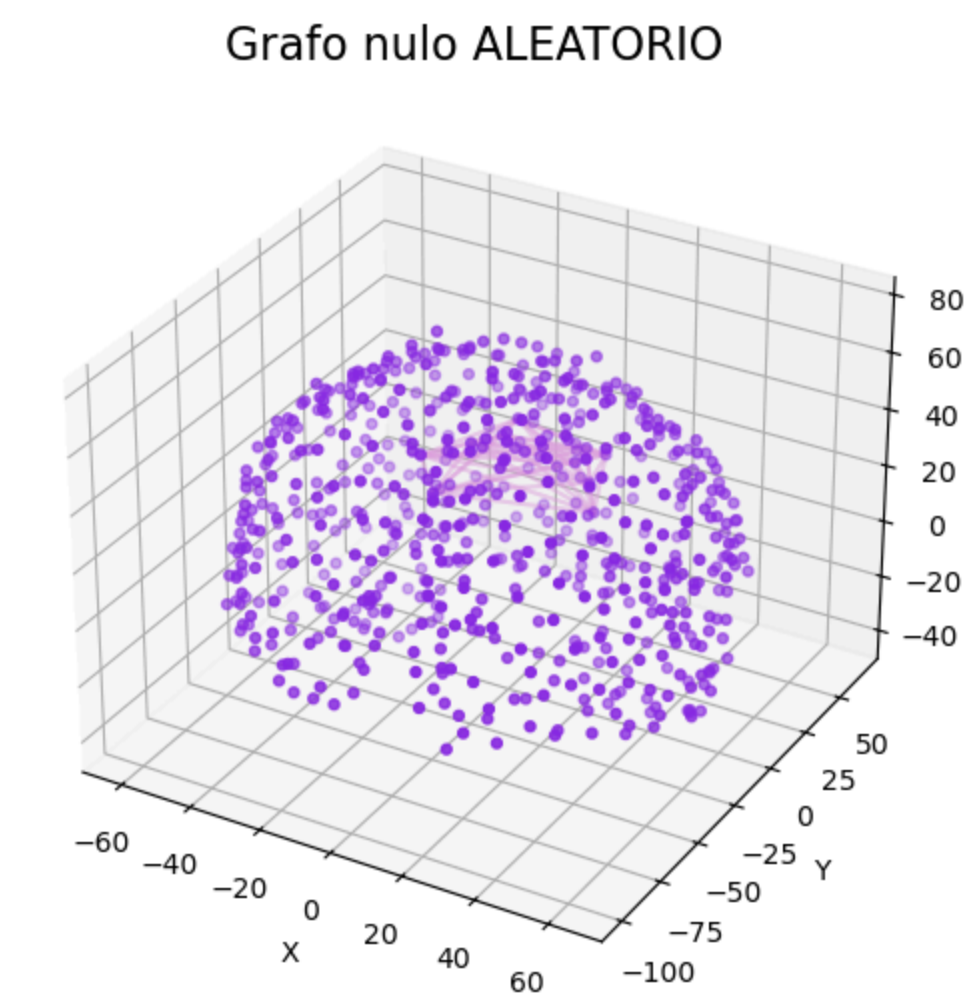
In [101…

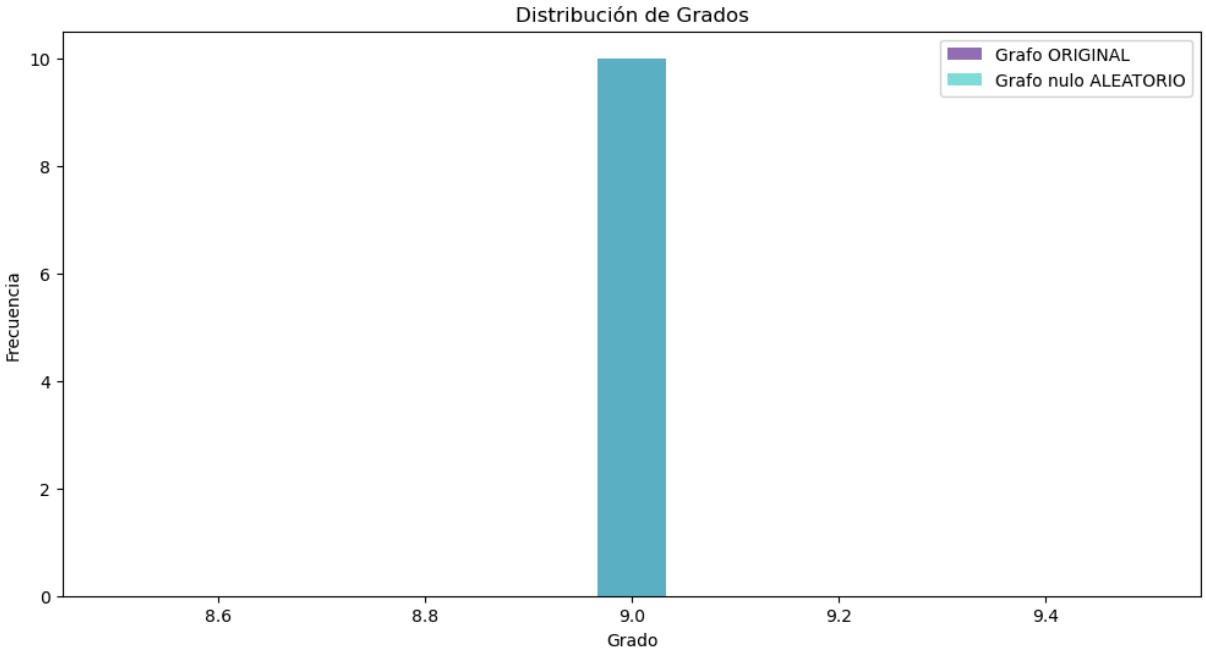
```
def grafo_nulo(num_nodos, num_edges): # grafo nulo aleatorio
    return nx.gnm_random_graph(num_nodos, num_edges)

def graficar_3Dgraph(G, x=None, y=None, z=None, titulo="Grafo 3D"):
    fig = plt.figure(figsize=(8, 6))
    ax = fig.add_subplot(111, projection='3d')

    if x is not None and y is not None and z is not None:
        ax.scatter(x, y, z, c='blueviolet', 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]]
            ax.plot(x_coords, y_coords, z_coords, c='plum', alpha=0.5)
    else:
        pos = nx.spring_layout(G, dim=3)
        x, y, z = np.array(list(pos.values())).T
        ax.scatter(x, y, z, c='mediumvioletred', 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]]
            ax.plot(x_coords, y_coords, z_coords, c='lightseagreen', alpha=0.5)

    ax.set_title(titulo, fontsize=16)
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')
    plt.show()

umbral = 0.1
mat_filt = np.where(coact_mat >= umbral, coact_mat, 0)

G = nx.from_numpy_array(mat_filt) # grafo original
num_nodos = G.number_of_nodes()
num_edges = G.number_of_edges()
G_random = grafo_nulo(num_nodos, num_edges) # grafo nulo ALEATORIO

# PROPIEDADES

propiedades_original = calculo_propiedades(G, "Propiedades del grafo ORIGINAL")
propiedades_random = calculo_propiedades(G_random, "Propiedades del grafo ALEATORIO

print("Propiedades del grafo ORIGINAL:")
print(propiedades_original)
print("\nPropiedades del grafo nulo ALEATORIO:")
print(propiedades_random)

# GRAFICAR GRAFOS
```

```
graficar_3Dgraph(G, x, y, z, titulo = "Grafo ORIGINAL")
graficar_3Dgraph(G_random, x, y, z, titulo = "Grafo nulo ALEATORIO")

# COMPARACIÓN DISTRIBUCIONES DE GRADO

grados_original = [grado for nodo, grado in G.degree()]
grados_random = [grado for nodo, grado in G_random.degree()]

plt.figure(figsize=(12, 6))
plt.hist(grados_original, bins=15, alpha=0.7, label="Grafo ORIGINAL", color = "rebe
plt.hist(grados_random, bins=15, alpha=0.7, label="Grafo nulo ALEATORIO", color = "
plt.title("Distribución de Grados")
plt.xlabel("Grado")
plt.ylabel("Frecuencia")
plt.legend()
plt.show()
```

```
Propiedades del grafo Propiedades del grafo ORIGINAL:
  número_nodos: 10
  número_aristas: 45
  grado_prom: 9.0
  coef_clustering promedio: 1.0
  centralidad de cercanía: 1.0
  centralidad de intermediación: 0.0
  coef_small_world: 1.0

Propiedades del grafo Propiedades del grafo ALEATORIO:
  número_nodos: 10
  número_aristas: 45
  grado_prom: 9.0
  coef_clustering promedio: 1.0
  centralidad de cercanía: 1.0
  centralidad de intermediación: 0.0
  coef_small_world: 1.0

Propiedades del grafo ORIGINAL:
{'número_nodos': 10, 'número_aristas': 45, 'grado_prom': 9.0, 'coef_clustering prome
dio': 1.0, 'centralidad de cercanía': 1.0, 'centralidad de intermediación': 0.0, 'co
ef_small_world': 1.0}

Propiedades del grafo nulo ALEATORIO:
{'número_nodos': 10, 'número_aristas': 45, 'grado_prom': 9.0, 'coef_clustering prome
dio': 1.0, 'centralidad de cercanía': 1.0, 'centralidad de intermediación': 0.0, 'co
ef_small_world': 1.0}
```


Grafo ORIGINAL
80
60
40
20
0
−20
−40
50
25
0
−25
−50
−75
−100
Y
−60
−40
−20
0
20
40
60
X





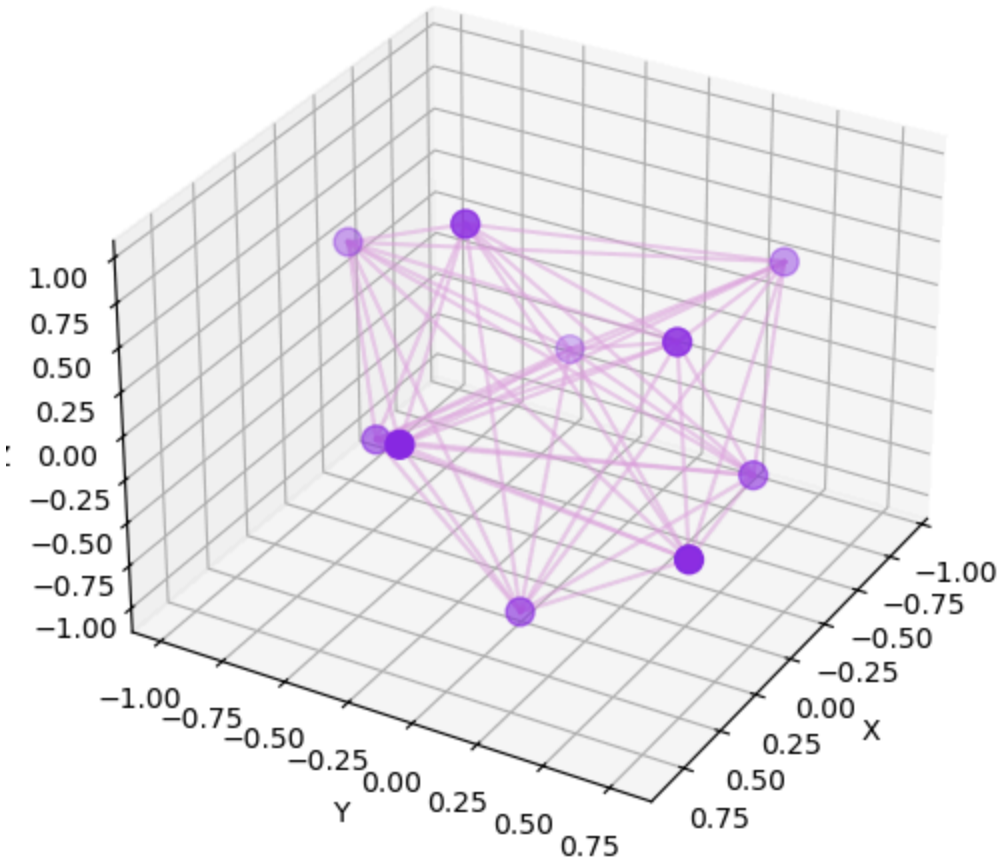
In [104…

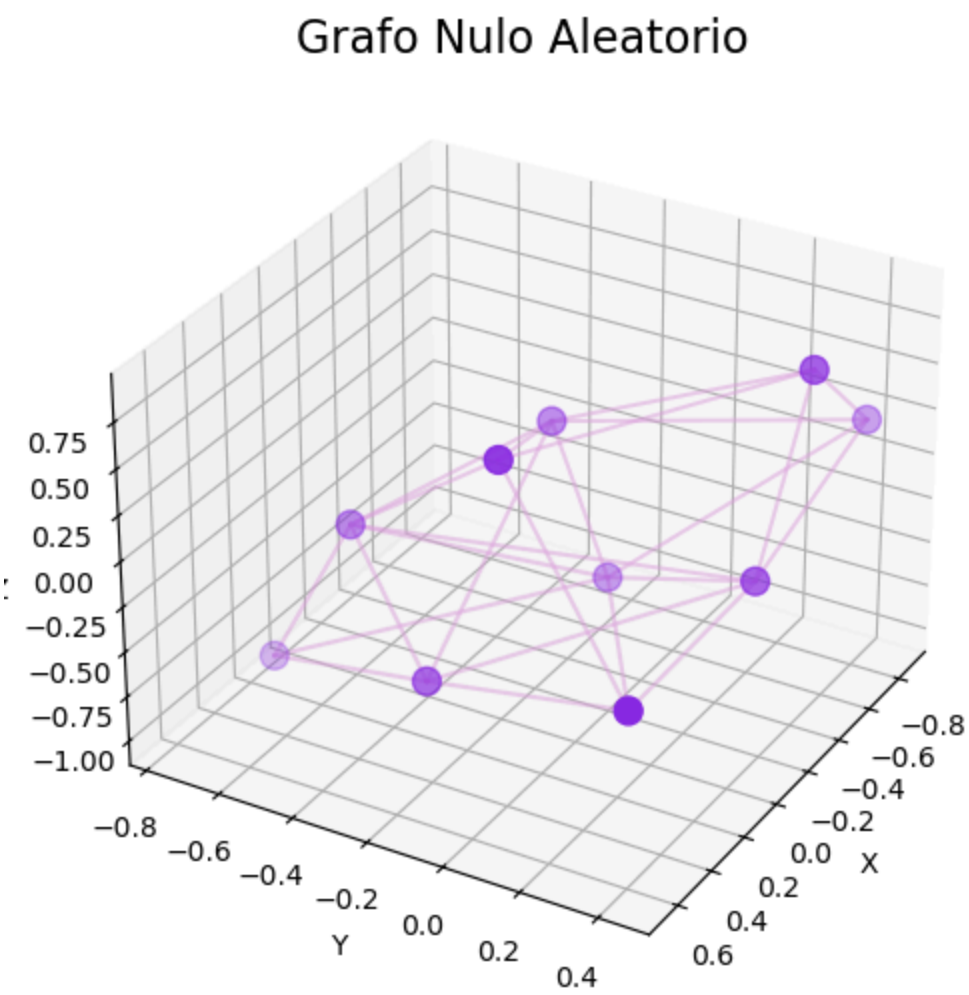
```
def graficar_3Dgraph(G, x=None, y=None, z=None, titulo="Grafo 3D"):
    fig = plt.figure(figsize=(8, 6))
    ax = fig.add_subplot(111, projection='3d')

    if x is not None and y is not None and z is not None:
```

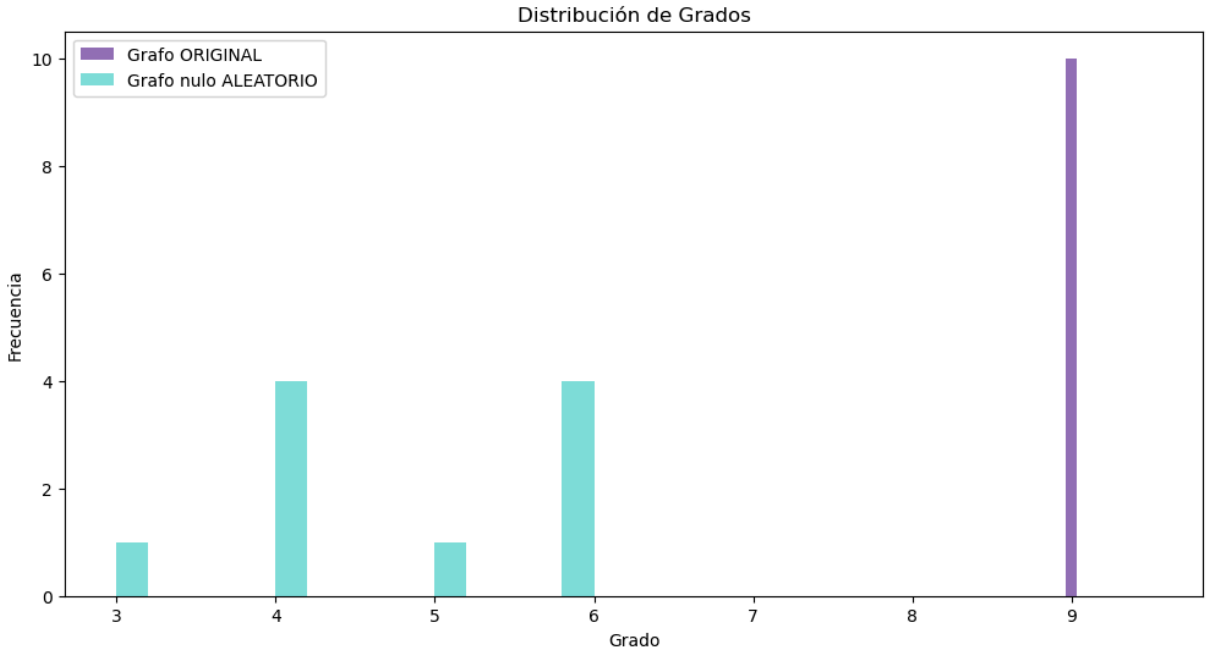
```
        ax.scatter(x, y, z, c='blueviolet', s=100, marker='o')  # Tamaño aumentado
        for edge in G.edges():
            node1, node2 = edge
            x_coords = [x[node1], x[node2]]
            y_coords = [y[node1], y[node2]]
            z_coords = [z[node1], z[node2]]
            ax.plot(x_coords, y_coords, z_coords, c='plum', alpha=0.5)
    else:
        pos = nx.spring_layout(G, dim=3, seed=42)
        x, y, z = np.array(list(pos.values())).T
        ax.scatter(x, y, z, c='mediumvioletred', s=100, marker='o')  # Tamaño aumen
        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='lightseagreen', alpha=0.5)

    ax.set_title(titulo, fontsize=16)
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')
    ax.view_init(elev=30, azim=30)  # Ajusta la vista para mejor visualización
    plt.show()

# Supón que tienes la matriz de co-activación y el umbral ya definidos
umbral = 0.1
mat_filt = np.where(coact_mat >= umbral, coact_mat, 0)

# Grafo original
G = nx.from_numpy_array(mat_filt)  # Crear el grafo original a partir de la matriz
num_nodos = G.number_of_nodes()
num_edges = G.number_of_edges()

# Grafo nulo con la misma distribución de grados que el original
G_random = grafo_nulo_con_grados(G)

# Calcular propiedades de los grafos
propiedades_original = calculo_propiedades(G, "Propiedades del grafo ORIGINAL")
propiedades_random = calculo_propiedades(G_random, "Propiedades del grafo ALEATORIO

print("Propiedades del grafo ORIGINAL:")
print(propiedades_original)
print("\nPropiedades del grafo nulo ALEATORIO:")
print(propiedades_random)

# Visualización en 3D
pos_original = nx.spring_layout(G, dim=3)  # Generar posiciones 3D para el grafo or
x, y, z = np.array(list(pos_original.values())).T  # Extraer las coordenadas x, y,
graficar_3Dgraph(G, x, y, z, titulo="Grafo ORIGINAL")

pos_random = nx.spring_layout(G_random, dim=3)  # Generar posiciones 3D para el gra
x_r, y_r, z_r = np.array(list(pos_random.values())).T  # Extraer las coordenadas x,
graficar_3Dgraph(G_random, x_r, y_r, z_r, titulo="Grafo Nulo Aleatorio")

# Comparación de distribuciones de grado
```

```
grados_original = [grado for nodo, grado in G.degree()]
grados_random = [grado for nodo, grado in G_random.degree()]

plt.figure(figsize=(12, 6))
plt.hist(grados_original, bins=15, alpha=0.7, label="Grafo ORIGINAL", color="rebecc
plt.hist(grados_random, bins=15, alpha=0.7, label="Grafo nulo ALEATORIO", color="me
plt.title("Distribución de Grados")
plt.xlabel("Grado")
plt.ylabel("Frecuencia")
plt.legend()
plt.show()
```

```
Propiedades del grafo Propiedades del grafo ORIGINAL:
  número_nodos: 10
  número_aristas: 45
  grado_prom: 9.0
  coef_clustering promedio: 1.0
  centralidad de cercanía: 1.0
  centralidad de intermediación: 0.0
  coef_small_world: 1.0

Propiedades del grafo Propiedades del grafo ALEATORIO:
  número_nodos: 10
  número_aristas: 24
  grado_prom: 4.8
  coef_clustering promedio: 0.45666666666666667
  centralidad de cercanía: 0.678337912087912
  centralidad de intermediación: 0.0611111111111111
  coef_small_world: 0.4329896907216495

Propiedades del grafo ORIGINAL:
{'número_nodos': 10, 'número_aristas': 45, 'grado_prom': 9.0, 'coef_clustering prome
dio': 1.0, 'centralidad de cercanía': 1.0, 'centralidad de intermediación': 0.0, 'co
ef_small_world': 1.0}

Propiedades del grafo nulo ALEATORIO:
{'número_nodos': 10, 'número_aristas': 24, 'grado_prom': 4.8, 'coef_clustering prome
dio': 0.45666666666666667, 'centralidad de cercanía': 0.678337912087912, 'centralida
d de intermediación': 0.0611111111111111, 'coef_small_world': 0.4329896907216495}
```


Grafo ORIGINAL
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
Y
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
X





In [ ]:
```
umbral = 0.1

# FUNCION 'VICENTE FERNANDEZ' PARA LAS PROPIEDADES
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

# FUNCION DE JOSE JOSE PARA LA GRAFICACION JAJAJAA
def grafo_nulo_aleatorio(num_nodos, num_aristas):

    G_random = nx.gnm_random_graph(num_nodos, num_aristas)
    return G_random

mat_filt = np.where(coact_mat >= umbral, coact_mat, 0)
G = nx.from_numpy_array(mat_filt)

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(f'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(f'Grafo Nulo Aleatorio', fontsize=16)
ax2.set_xlabel('X')
ax2.set_ylabel('Y')
ax2.set_zlabel('Z')

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

EJERCICIO 9.

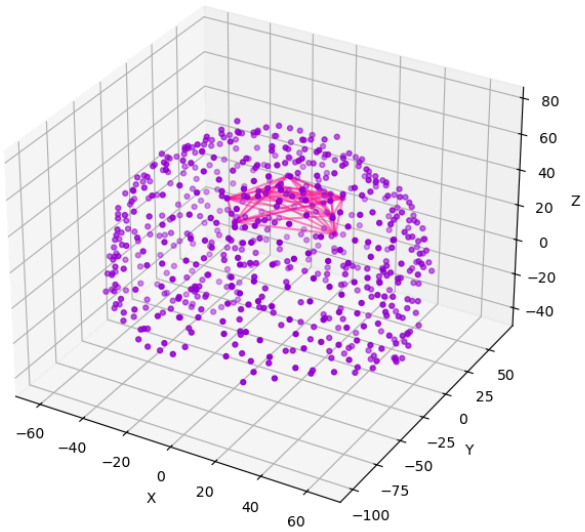
Generar un modelo nulo aleatorio donde se conserve la distribución de grado y comparar sus propiedades con el grafo original del cerebro.

In [97]:
```
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

def graficar_3Dgraph(G, x, y, z, ax, titulo):
    ax.scatter(x, y, z, c='darkviolet', 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]]
        ax.plot(x_coords, y_coords, z_coords, c='deeppink', alpha=0.5)
    ax.set_title(titulo, fontsize=16)
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')

mat_filt = np.where(coact_mat >= umbral, coact_mat, 0)
G_original = nx.from_numpy_array(mat_filt)
G_nulo = grafo_nulo_con_grados(G_original)

# Calcular propiedades de los grafos
propiedades_original = calculo_propiedades(G_original, "Propiedades de grafo ORIGIN
propiedades_nulo = calculo_propiedades(G_nulo, "Propiedades de grafo ALEATORIO")

print(f"\nPropiedades del grafo original:\n{propiedades_original}")
print(f"\nPropiedades del grafo nulo:\n{propiedades_nulo}")

# Visualización
fig = plt.figure(figsize=(12, 6))
ax1 = fig.add_subplot(121, projection='3d')
graficar_3Dgraph(G_original, x, y, z, ax1, "Grafo ORIGINAL")
```

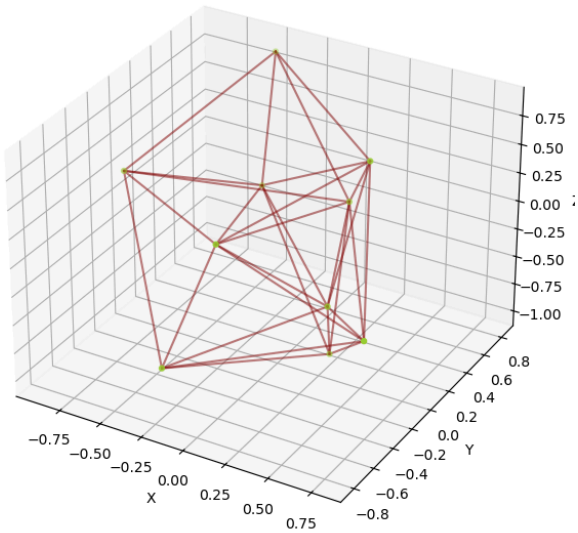
```
ax2 = fig.add_subplot(122, projection='3d')
pos_random = nx.spring_layout(G_nulo, dim=3)  # Posiciones aleatorias para el grafo
x_r, y_r, z_r = np.array(list(pos_random.values())).T
graficar_grafo_3d(G_nulo, x_r, y_r, z_r, ax2, "Grafo nulo ALEATORIO")

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

```
Propiedades del grafo Propiedades de grafo ORIGINAL:
  número_nodos: 10
  número_aristas: 45
  grado_prom: 9.0
  coef_clustering promedio: 1.0
  centralidad de cercanía: 1.0
  centralidad de intermediación: 0.0
  coef_small_world: 1.0

Propiedades del grafo Propiedades de grafo ALEATORIO:
  número_nodos: 10
  número_aristas: 29
  grado_prom: 5.8
  coef_clustering promedio: 0.6042857142857143
  centralidad de cercanía: 0.7424325674325675
  centralidad de intermediación: 0.04444444444444444
  coef_small_world: 0.6041666666666666


Propiedades del grafo original:
{'número_nodos': 10, 'número_aristas': 45, 'grado_prom': 9.0, 'coef_clustering prome
dio': 1.0, 'centralidad de cercanía': 1.0, 'centralidad de intermediación': 0.0, 'co
ef_small_world': 1.0}

Propiedades del grafo nulo:
{'número_nodos': 10, 'número_aristas': 29, 'grado_prom': 5.8, 'coef_clustering prome
dio': 0.6042857142857143, 'centralidad de cercanía': 0.7424325674325675, 'centralida
d de intermediación': 0.04444444444444444, 'coef_small_world': 0.6041666666666666}
```

Grafo Original



Grafo Nulo Aleatorio



EJERCICIO 10.

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

In [ ]:
```python
def calcular_propiedades(grafo):
    """
    Calcula propiedades relevantes del grafo.
    """
    coef_clust = nx.average_clustering(grafo)  # Coeficiente de clustering promedio
    grado = [grafo.degree(n) for n in grafo.nodes]  # Grados de los nodos
    longitud_caracteristica = nx.average_shortest_path_length(grafo) if nx.is_conne
    return coef_clust, grado, longitud_caracteristica
# Propiedades del grafo original
coef_clust_o, grados_o, long_caract_o = calcular_propiedades(G_original)

# Generador de grafo nulo
def grafo_nulo_distancia(coords, num_nodos, num_aristas, alpha=10):
    G_nulo = nx.Graph()
    G_nulo.add_nodes_from(range(num_nodos))

    # Calcular distancias entre pares de nodos
    distancias = np.zeros((num_nodos, num_nodos))
    for i in range(num_nodos):
        for j in range(num_nodos):
            if i != j:
                distancias[i, j] = np.linalg.norm(coords[i] - coords[j])

    # Probabilidad basada en distancia
    prob_conexion = np.exp(-alpha * distancias)

    # Añadir aristas basadas en probabilidad
    while G_nulo.number_of_edges() < num_aristas:
        i, j = np.random.choice(num_nodos, size=2, replace=False)
        if i != j and G_nulo.has_edge(i, j) == False:
            if np.random.rand() < prob_conexion[i, j]:
                G_nulo.add_edge(i, j)

    return G_nulo

G_nulo_distancia = grafo_nulo_distancia(coords, num_nodes, num_edges, alpha=10)
coef_clust_n, grados_n, long_caract_n = calcular_propiedades(G_nulo_distancia)

# Comparación
print("Propiedades del Grafo Original:")
print(f"Coeficiente de Clustering: {coef_clust_o:.4f}")
print(f"Longitud Característica: {long_caract_o:.4f}")
print("\nPropiedades del Grafo Nulo:")
print(f"Coeficiente de Clustering: {coef_clust_n:.4f}")
print(f"Longitud Característica: {long_caract_n:.4f}")
import matplotlib.pyplot as plt

# Distribución de grados
```

```
plt.figure(figsize=(12, 6))
plt.hist(grados_o, bins=10, alpha=0.5, label="Original", color="blue", density=True
plt.hist(grados_n, bins=10, alpha=0.5, label="Nulo", color="orange", density=True)
plt.xlabel("Grado")
plt.ylabel("Frecuencia Normalizada")
plt.title("Distribución de Grados")
plt.legend()
plt.show()

# Visualización 3D
fig = plt.figure(figsize=(12, 6))
ax1 = fig.add_subplot(121, projection='3d')
graficar_grafo_3d(G_original, coords[:, 0], coords[:, 1], coords[:, 2], ax1, "Grafo

ax2 = fig.add_subplot(122, projection='3d')
graficar_grafo_3d(G_nulo_distancia, coords[:, 0], coords[:, 1], coords[:, 2], ax2,
plt.tight_layout()
plt.show()
```

DISCUSIÓN

Las conexiones a larga distancia en el cerebro son fundamentales para garantizar la integración y coordinación de la actividad neuronal en regiones distantes, permitiendo un procesamiento eficiente y funcional. Estas conexiones juegan un papel crucial en la transmisión rápida de información entre áreas especializadas, como las redes sensoriales, motoras y cognitivas, promoviendo la integración global del cerebro.

Desde una perspectiva estructural, las fibras de materia blanca, como las asociadasRCub, *conectan nodos clave en la red cerebral. Estas conexiones no solo reducen la longitud característica de la r*ed, optimiz*ando la transmisión de señales, sino que también soportan funciones complejas como la memoria, la toma de decisiones y la percepción multisensorial. Funcionalmente, facilitan la sincronización neuronal, lo que mejora la eficiencia del procesamiento y asegura que regiones especializadas trabajen en conjunto para generar respuestas coordinadas.

Además, estas conexiones son esenciales para la plasticidad cerebral, permitiendo la reestructuración de las redes tras lesiones o durante el aprendizaje. En este sentido, actúan como un puente entre la modularidad local y la integración global, balanceando la especialización regional con la conectividad de toda la red.

La ausencia o alteración de estas conexiones a larga distancia puede impactar negativamente la organización funcional del cerebro, afectando la comunicación interregional y contribuyendo a trastornos neurológicos. Esto subraya su importancia en la conservación de la funcionalidad y la resiliencia del cerebro frente a cambios daños.

EJERCICIO 11.

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

RESEÑA

El curso de Modelos Computacionales I permitió adquirir una comprensión más profunda de cómo las herramientas computacionales y los conceptos de matemáticas se pueden aplicar a las neurociencias y el análisis de redes complejas. A través de la programación en Python y MATLAB, se enseñó el análisis de datos, usando bibliotecas como NumPy, NetworkX y Scipy, en el caso de Python. La teoría de grafos fue clave y se aplicó para comprender la conectividad en el SNC. Los grafos, que consisten en nodos y aristas, son una representación ideal de las conexiones neuronales, donde cada nodo representa una región del cerebro y cada arista indica una interacción entre dichas regiones. Conocer las propiedades de los grafos, tales como la centralidad, la robustez y la conectividad, resulta crucial para analizar cómo las diferentes áreas del cerebro se comunican entre sí, cómo se propaga la información y cómo se forman redes funcionales.

En general, comprender la conectividad cerebral a través de esta teoría permite identificar patrones de actividad y de comunicación en el cerebro, importante para estudiar trastornos neurológicos, como la esquizofrenia o el Alzheimer, así como para desarrollar interfaces cerebro-computadora.