fon50z9qn

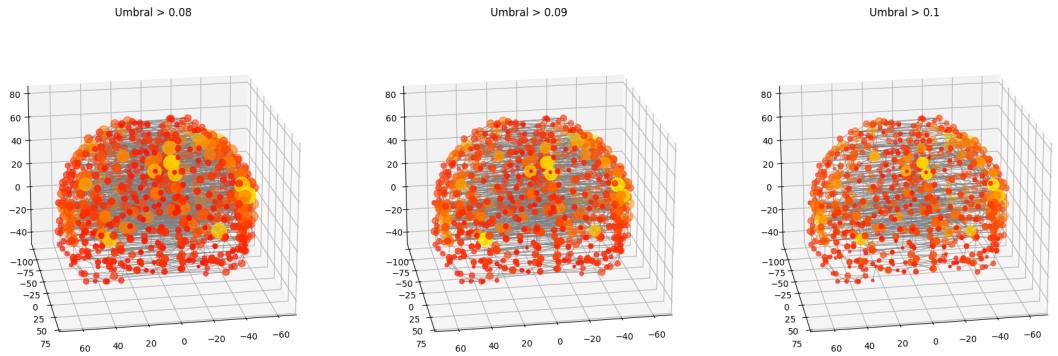
November 30, 2024

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
import networkx as nx
from mpl_toolkits.mplot3d import Axes3D
import scipy.io
from sklearn.cluster import KMeans
from sklearn.metrics import calinski_harabasz_score
from scipy.spatial.distance import pdist, squareform

# Cargar los datos
mat_path = r'C:\Users\daphn\Documents\UNAM\Neurociencias\Quinto semestre␣
 ↪1\Modelos\Git\Neurociencias-2025-1\BCT\BCT\2019_03_03_BCT\data_and_demos\Coactivation_matri
 ↪mat'
mat_data = scipy.io.loadmat(mat_path)

# Obtener la matriz de coactivación y las coordenadas
coactivation_matrix = mat_data['Coactivation_matrix']
coordinates = mat_data['Coord']
```

1 Ejercicio 1

[2]:
```
# Umbrales para los grafos
thresholds = [0.08, 0.09, 0.1]

# Crear los grafos para los tres umbrales
fig = plt.figure(figsize=(18, 6))
for i, threshold in enumerate(thresholds):
    # Crear matriz de adyacencia
    adjacency_matrix = (coactivation_matrix > threshold).astype(int)

    # Crear el grafo
    G = nx.from_numpy_array(adjacency_matrix)
    degree = dict(G.degree())

    # Coordenadas de los nodos
```

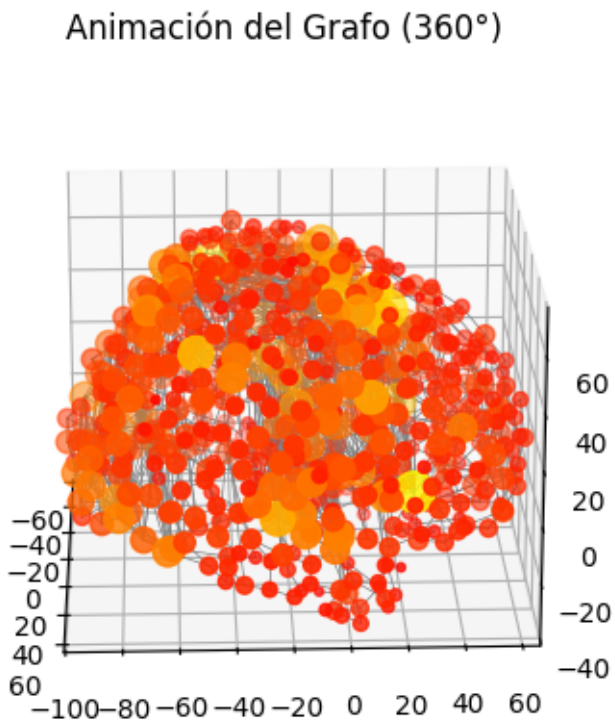
```
    pos_3d = {i: coordinates[i] for i in range(len(G))}
    x = [pos[0] for pos in pos_3d.values()]
    y = [pos[1] for pos in pos_3d.values()]
    z = [pos[2] for pos in pos_3d.values()]

    # Crear la subgráfica
    ax = fig.add_subplot(1, 3, i + 1, projection='3d')

    # Dibujar nodos
    ax.scatter(x, y, z, s=[degree[node] * 10 for node in G.nodes],
               c=list(degree.values()), cmap='autumn', label='Nodos')

    # Dibujar bordes
    for edge in G.edges:
        start, end = edge
        ax.plot([x[start], x[end]], [y[start], y[end]], [z[start], z[end]],
                color='gray', linewidth=0.5)

    ax.set_title(f'Umbral > {threshold}')
    ax.view_init(elev=20, azim=80)

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



2 Ejercicio 2

[3]:
```
# Grafo con umbral 0.09 y animación 3D
threshold = 0.09
adjacency_matrix = (coactivation_matrix > threshold).astype(int)
G = nx.from_numpy_array(adjacency_matrix)
degree = dict(G.degree())
```

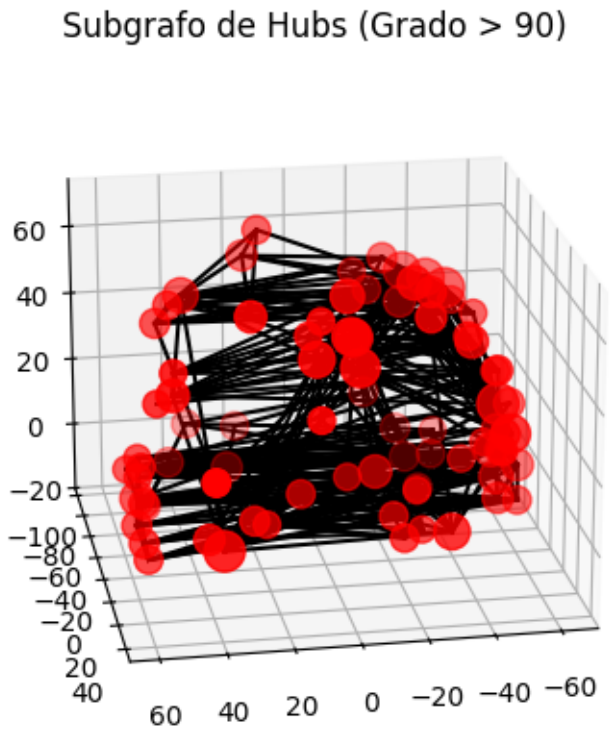
```python
# Coordenadas 3D para los nodos
pos_3d = {i: coordinates[i] for i in range(len(G))}
x = [pos[0] for pos in pos_3d.values()]
y = [pos[1] for pos in pos_3d.values()]
z = [pos[2] for pos in pos_3d.values()]

# Crear figura para animación
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')


# Inicialización de los datos en el gráfico
def init():
    """Inicializar el gráfico."""
    ax.clear()
    ax.scatter(x, y, z, s=[degree[node] * 10 for node in G.nodes],␣
 ↪c=list(degree.values()), cmap='autumn')
    for edge in G.edges:
        start, end = edge
        ax.plot([x[start], x[end]], [y[start], y[end]], [z[start], z[end]],
                color='gray', linewidth=0.5)
    ax.set_xlim(min(x), max(x))
    ax.set_ylim(min(y), max(y))
    ax.set_zlim(min(z), max(z))
    ax.set_title('Animación del Grafo (360°)')
    return ax,


# Función de actualización
def update(angle):
    """Actualizar el gráfico para la animación."""
    ax.view_init(elev=20, azim=angle)
    return ax,


# Crear la animación
anim = FuncAnimation(fig, update, init_func=init, frames=np.arange(0, 360, 2),␣
 ↪interval=100)

# Guardar la animación como GIF
anim.save('grafo_rotacion.gif', writer='pillow')

plt.show()
```



3 Ejercicio 3

[4]:
```
# Identificar hubs (nodos con grado > 9)
hub_nodes = [node for node, deg in degree.items() if deg > 9]
hub_subgraph = G.subgraph(hub_nodes)

# Crear la figura del subgrafo de hubs
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Coordenadas de los hubs
hub_pos = {i: coordinates[i] for i in hub_nodes}
hub_x = [hub_pos[node][0] for node in hub_nodes]
hub_y = [hub_pos[node][1] for node in hub_nodes]
hub_z = [hub_pos[node][2] for node in hub_nodes]

# Dibujar nodos de hubs
ax.scatter(hub_x, hub_y, hub_z, s=[degree[node] * 10 for node in hub_nodes],␣
 ↪c='red', label='Hubs')

# Dibujar bordes entre hubs
```

```
for edge in hub_subgraph.edges:
    start, end = edge
    ax.plot([hub_pos[start][0], hub_pos[end][0]],
            [hub_pos[start][1], hub_pos[end][1]],
            [hub_pos[start][2], hub_pos[end][2]], color='black', linewidth=1.5)

# Ajustar la vista
ax.set_title('Subgrafo de Hubs (Grado > 9)')
ax.view_init(elev=20, azim=80)
plt.show()
```



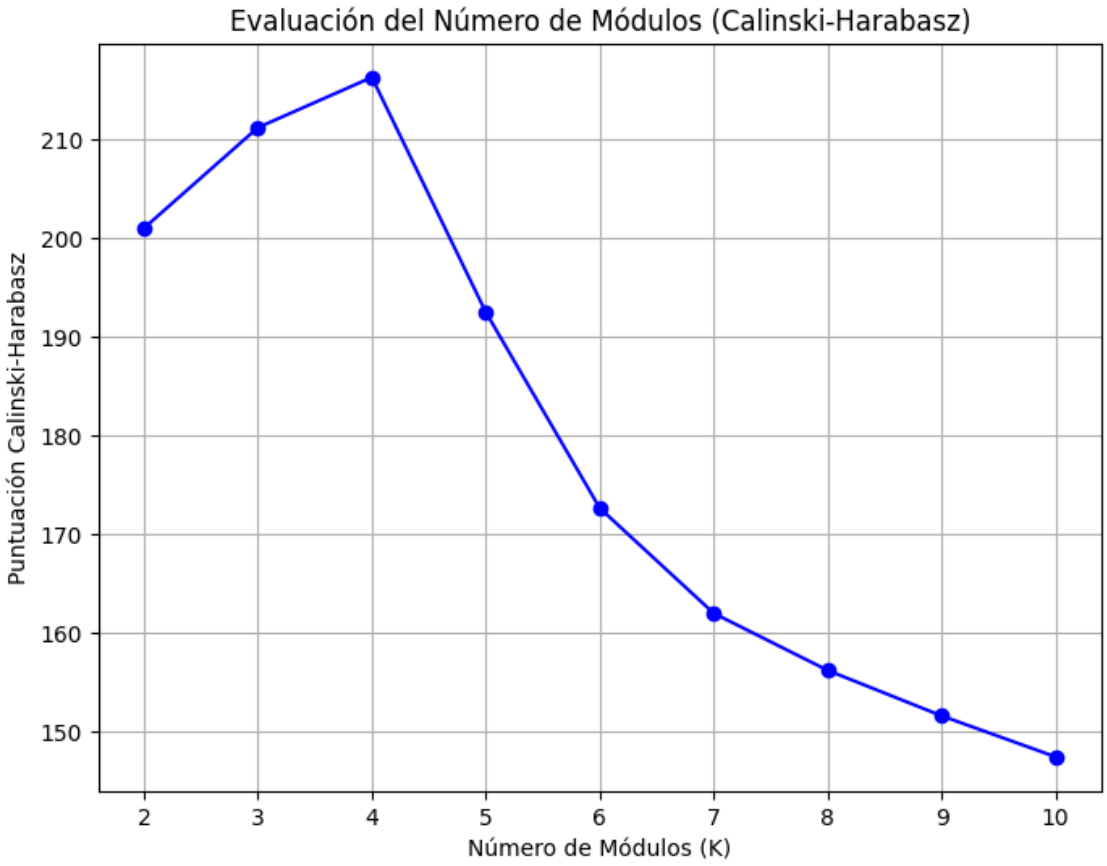
4 Ejercicio 4

[5]:
```
# Ejercicio 4

# Matriz de emparejamiento (correlación de la matriz de adyacencia)
Mat_Emp = np.corrcoef(mat_data['Coactivation_matrix'])

# Evaluar diferentes valores de K
k_range = range(2, 11)  # Rango de módulos (K) a evaluar
scores = []
```

```python
for k in k_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    labels = kmeans.fit_predict(Mat_Emp)
    score = calinski_harabasz_score(Mat_Emp, labels)
    scores.append(score)

# Encontrar el K con el mejor puntaje
optimal_k = k_range[np.argmax(scores)]

print(f'Número óptimo de módulos (K) basado en Calinski-Harabasz: {optimal_k}')

# Visualizar la mejor partición
kmeans = KMeans(n_clusters=optimal_k, random_state=42, n_init=10)
final_labels = kmeans.fit_predict(Mat_Emp)

# Mostrar los resultados de la partición
print(f'Partición de nodos en {optimal_k} módulos:', final_labels)

# Graficar la puntuación de Calinski-Harabasz para cada K
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 6))
plt.plot(k_range, scores, marker='o', linestyle='-', color='b')
plt.title('Evaluación del Número de Módulos (Calinski-Harabasz)')
plt.xlabel('Número de Módulos (K)')
plt.ylabel('Puntuación Calinski-Harabasz')
plt.grid()
plt.show()

# A partir de la métrica de Calinski-Harabasz, que mide la compacidad y␣
 ↪separación de los clústeres y el gráfico
# visualizado a partir de los cálculos, la mejor partición es en 4 módulos␣
 ↪seguida de 3.
```

```
Número óptimo de módulos (K) basado en Calinski-Harabasz: 4
Partición de nodos en 4 módulos: [0 0 0 0 0 2 2 1 0 0 2 0 2 0 0 0 1 0 2 1 2 0 0
0 0 0 0 3 0 0 0 3 0 3 0 3 0
 2 1 2 1 2 1 1 1 1 2 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1 1 0 1 1 0 1 1 1 0 0 1
 0 1 1 0 0 1 1 0 0 0 0 0 0 1 1 0 0 0 0 1 1 1 0 1 1 0 1 0 0 0 1 0 0 0 0 0 0
 0 0 0 0 0 1 0 1 1 1 1 1 0 1 1 1 0 1 1 1 1 0 0 0 1 0 0 0 0 1 0 0 1 0 0 0 0
 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
 2 1 0 0 0 0 0 1 1 3 3 0 3 0 0 0 0 3 3 3 0 3 0 3 0 0 3 3 2 2 0 0 0 0 0 0 0
 0 0 0 0 0 0 2 2 1 1 1 2 2 1 1 2 2 2 2 2 3 3 3 3 0 0 3 0 0 0 3 3 3 0 0 0 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 0 1 3 3 3 3 3 3 3 3 1 0 3 3 3 1 3 0 0 0
 1 0 1 0 3 3 3 1 3 3 0 0 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 2 1 1 1
 1 1 3 3 3 0 0 3 3 0 3 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 0 2 1 2
```

```
2 2 2 2 2 2 2 2 0 2 2 2 0 2 2 2 0 2 0 0 0 2 2 2 2 2 2 1 1 1 1 1 2 1 2 1 2
1 2 2 1 2 2 0 2 2 1 0 1 2 2 1 0 2 0 0 0 0 0 0 1 0 0 1 1 0 1 0 0 0 0 0 0 0
1 0 0 1 0 0 1 0 2 2 2 2 0 3 3 3 3 0 2 2 2 2 0 0 0 0 0 0 0 2 2 2 2 2 2 2 2 2
2 1 1 0 1 2 2 2 2 2 1 1 1 1 2 2 2 0 0 2 2 2 2 2 2 2 2 0 1 2 2 0 1 0 0 0 0
0 0 0 3 0 0 0 0 0 2 2 0 0 0 3 0 0 0 3 0 0 3 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0 3 0 0 0 0 0 0 0 0 0 2 2
0 0 0 0 0 2 2 2 2 2 2 2 2 2 2 2 2 0 2 2 2 0 2 2 2 0 0 2 2 2 2 2 2 2 0 0 2
2 2 2 0 0 0 2 0 0]
```

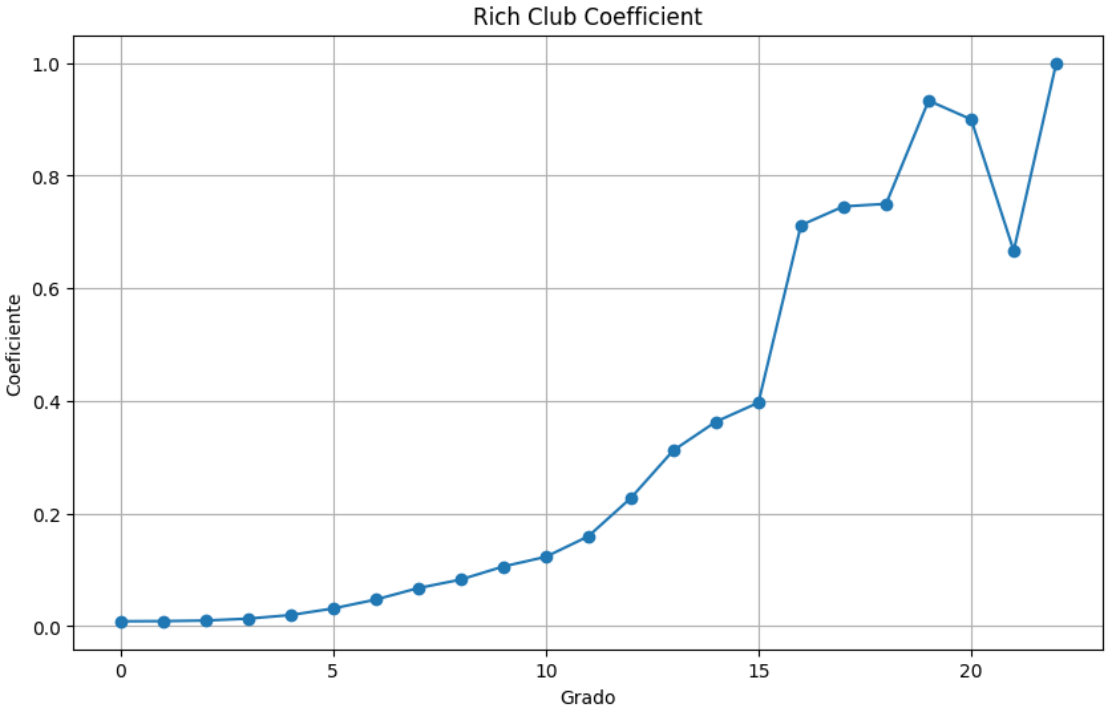


5 Ejercicio 5

Al tener nodos altamente interconectados, como aquellos que tienen aproximadamente grado de 23, 20 y 19, refleja que estos tienen una alta conexión anatómica y una buena integración funcional que puede estar relacionado con un procesamiento óptimo de procesos complejos, lo que los convertiría en áreas de alta especialización y son los necesarios para mantener una unión de la red cerebral. Por otro lado, también tenemos nodos muy aislados sin embargo, esto puede ayudar a la red a mantener un balance aumentando la eficiencia en donde las áreas cognitivas principales trabajan de manera más óptima y los nodos aislados puede que sean específicos para ciertas tareas por lo cual no necesitan estar todo el tiempo activos a comparación de los hubs.

[14]:
```
# Eliminar nodos aislados
G_filtered.remove_nodes_from(list(nx.isolates(G)))

# Comprobar que el grafo no esté vacío
if len(G_filtered.nodes) == 0:
    raise ValueError("El grafo está vacío después de eliminar nodos aislados.")

# Calcular Rich Club Coefficient con manejo de errores
try:
    rich_club = nx.rich_club_coefficient(G_filtered, normalized=True)
except ZeroDivisionError:
    print("Advertencia: Ocurrió una división entre cero. Intentando sin␣
 ↪normalización...")
    rich_club = nx.rich_club_coefficient(G_filtered, normalized=False)

# Graficar el coeficiente de Rich Club
plt.figure(figsize=(10, 6))
plt.plot(list(rich_club.keys()), list(rich_club.values()), marker='o')
plt.title("Rich Club Coefficient")
plt.xlabel("Grado")
plt.ylabel("Coeficiente")
plt.grid(True)
plt.show()
```

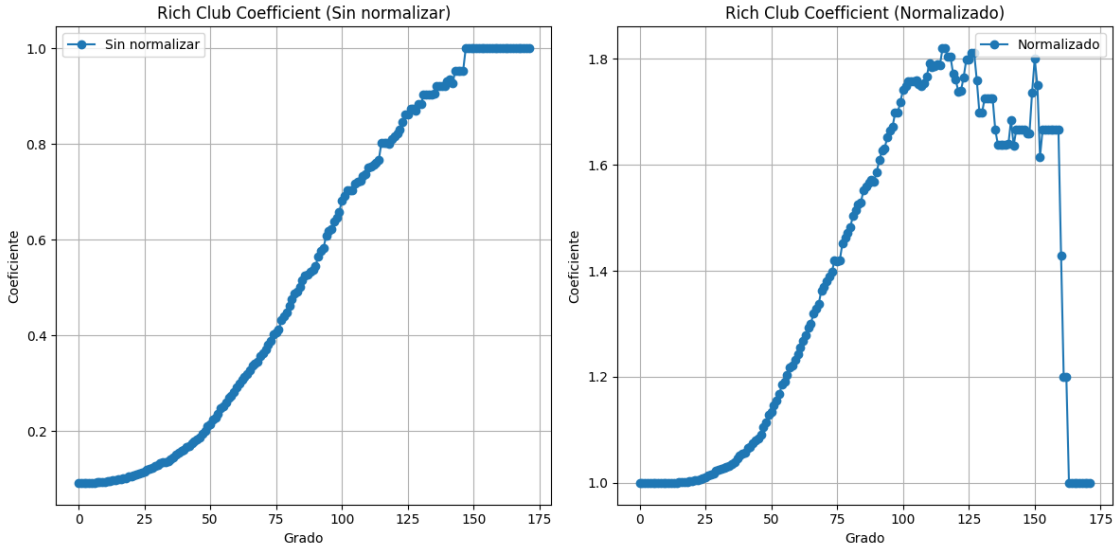
```
Advertencia: Ocurrió una división entre cero. Intentando sin normalización…
```



6 Ejercicio 6

[21]:
```
# Obtener la matriz de coactivación
coactivation_matrix = mat_data['Coactivation_matrix']

# Asegurar que la matriz sea simétrica y sin autoconexiones
coactivation_matrix = (coactivation_matrix + coactivation_matrix.T) / 2
np.fill_diagonal(coactivation_matrix, 0)

# Crear el grafo desde la matriz de coactivación
G = nx.from_numpy_array(coactivation_matrix)

# Eliminar nodos aislados para evitar errores en cálculos
G.remove_nodes_from(list(nx.isolates(G)))

# Manejo del cálculo del coeficiente de Rich Club para evitar divisiones por␣
 ↪cero
def safe_rich_club_coefficient(graph, normalized=True):
    """Calcula el coeficiente de Rich Club evitando divisiones por cero."""
    rich_club = nx.rich_club_coefficient(graph, normalized=normalized)
    # Filtrar cualquier entrada donde el coeficiente sea indefinido
    rich_club_safe = {k: v for k, v in rich_club.items() if v is not None and␣
 ↪not np.isnan(v) and not np.isinf(v)}
    return rich_club_safe

# Calcular el coeficiente de Rich Club (sin normalizar)
rich_club_unnormalized = safe_rich_club_coefficient(G, normalized=False)

# Calcular el coeficiente de Rich Club (normalizado)
rich_club_normalized = safe_rich_club_coefficient(G, normalized=True)

# Graficar los coeficientes de Rich Club
plt.figure(figsize=(12, 6))

# Gráfica sin normalizar
plt.subplot(1, 2, 1)
plt.plot(list(rich_club_unnormalized.keys()), list(rich_club_unnormalized.
 ↪values()), marker='o', label="Sin normalizar")
plt.title("Rich Club Coefficient (Sin normalizar)")
plt.xlabel("Grado")
plt.ylabel("Coeficiente")
plt.grid(True)
plt.legend()
```

```
# Gráfica normalizada
plt.subplot(1, 2, 2)
plt.plot(list(rich_club_normalized.keys()), list(rich_club_normalized.
  ↪values()), marker='o', label="Normalizado")
plt.title("Rich Club Coefficient (Normalizado)")
plt.xlabel("Grado")
plt.ylabel("Coeficiente")
plt.grid(True)
plt.legend()

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

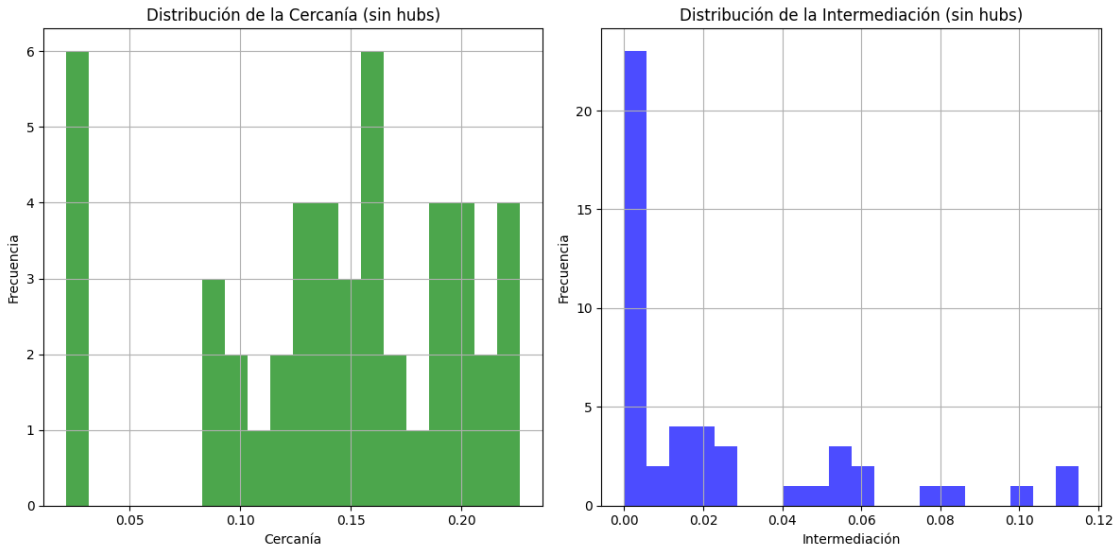


[27]:
```
# Identificar los nodos más interconectados (hubs)
threshold = 20  # Umbral de grado (ajusta este valor según la red)
hubs = [node for node, degree in G.degree() if degree >= threshold]
print(f"Nodos eliminados (hubs): {hubs}")

# Crear una copia del grafo y eliminar los hubs
G_no_hubs = G.copy()
G_no_hubs.remove_nodes_from(hubs)

# Asegurarse de que no haya nodos aislados restantes
G_no_hubs.remove_nodes_from(list(nx.isolates(G_no_hubs)))

# Recalcular las métricas para el grafo sin hubs
def calculate_graph_properties(graph):
    """Calcula las propiedades clave de un grafo."""
```

```
    # Coeficiente de mundo pequeño
    try:
        avg_shortest_path = nx.average_shortest_path_length(graph)
        clustering = nx.clustering(graph)  # Coeficiente de agrupamiento
        avg_clustering = np.mean(list(clustering.values()))  # Promedio
        small_world = avg_clustering / avg_shortest_path
    except nx.NetworkXError:
        avg_shortest_path = float('inf')
        avg_clustering = 0
        small_world = 0

    # Centralidades
    closeness = nx.closeness_centrality(graph)
    betweenness = nx.betweenness_centrality(graph)

    return small_world, avg_shortest_path, avg_clustering, closeness,␣
 ↪betweenness

# Calcular propiedades del grafo sin hubs
small_world_no_hubs, avg_shortest_path_no_hubs, avg_clustering_no_hubs,␣
 ↪closeness_no_hubs, betweenness_no_hubs =␣
 ↪calculate_graph_properties(G_no_hubs)

# Imprimir los resultados
print(f"Coeficiente de Mundo Pequeño (sin hubs): {small_world_no_hubs}")
print(f"Promedio de la distancia más corta (sin hubs):␣
 ↪{avg_shortest_path_no_hubs}")
print(f"Promedio del coeficiente de agrupamiento (sin hubs):␣
 ↪{avg_clustering_no_hubs}")

# Graficar distribuciones de centralidad en el grafo sin hubs
plt.figure(figsize=(12, 6))

# Graficar cercanía
plt.subplot(1, 2, 1)
plt.hist(list(closeness_no_hubs.values()), bins=20, color='green', alpha=0.7)
plt.title("Distribución de la Cercanía (sin hubs)")
plt.xlabel("Cercanía")
plt.ylabel("Frecuencia")
plt.grid(True)

# Graficar intermediación
plt.subplot(1, 2, 2)
plt.hist(list(betweenness_no_hubs.values()), bins=20, color='blue', alpha=0.7)
plt.title("Distribución de la Intermediación (sin hubs)")
plt.xlabel("Intermediación")
plt.ylabel("Frecuencia")
```

```
plt.grid(True)

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

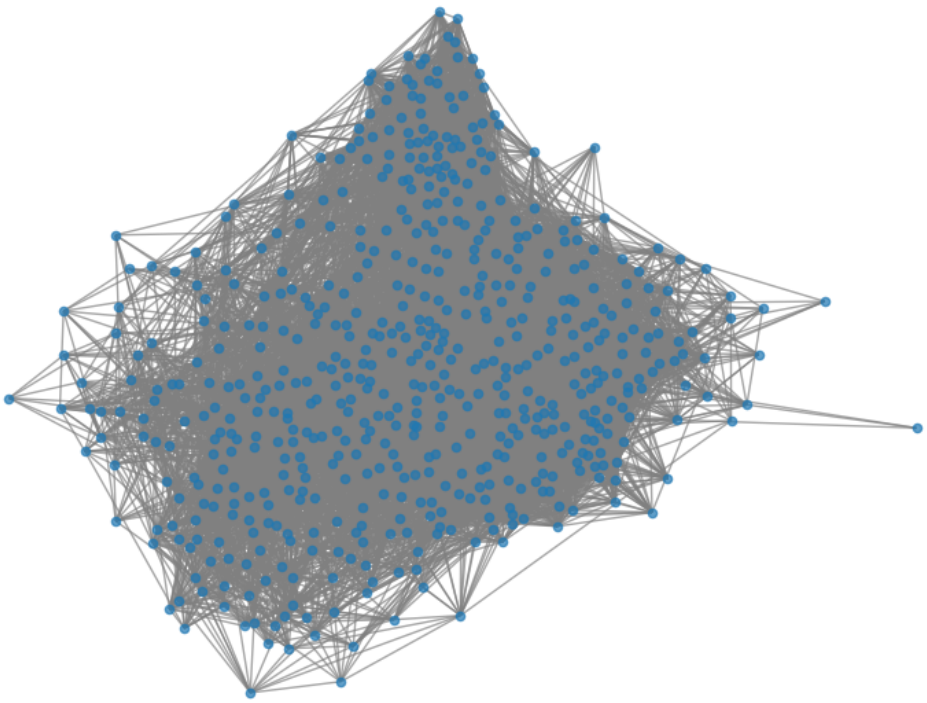
```
Nodos eliminados (hubs): [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35,
36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55,
56, 57, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76,
77, 78, 79, 80, 81, 82, 83, 84, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97,
98, 99, 100, 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, 140, 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, 173, 174, 177, 178, 180, 181, 182, 184, 185, 186,
188, 190, 191, 192, 193, 194, 195, 196, 197, 201, 202, 203, 204, 205, 206, 207,
208, 209, 210, 211, 212, 213, 214, 215, 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, 257,
258, 259, 260, 261, 262, 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, 311, 312, 313, 314, 315, 317, 318, 320, 322, 323, 324, 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, 383, 384, 385, 386, 387, 391, 392, 393, 394, 395,
396, 397, 398, 399, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411,
412, 414, 415, 416, 417, 418, 419, 420, 421, 423, 425, 426, 427, 428, 429, 430,
431, 432, 433, 434, 435, 436, 437, 438, 439, 441, 442, 443, 444, 445, 446, 447,
448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463,
464, 465, 466, 467, 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, 506, 507, 508, 509, 510, 511, 513, 514,
515, 516, 517, 519, 521, 523, 526, 527, 528, 529, 530, 531, 532, 536, 538, 539,
543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558,
559, 560, 561, 562, 563, 564, 565, 566, 567, 568, 569, 570, 571, 572, 573, 574,
575, 576, 577, 578, 579, 580, 581, 582, 584, 585, 587, 588, 589, 590, 591, 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, 628, 629, 630, 631, 632, 633, 634, 635, 637]
Coeficiente de Mundo Pequeño (sin hubs): 0
Promedio de la distancia más corta (sin hubs): inf
Promedio del coeficiente de agrupamiento (sin hubs): 0
```

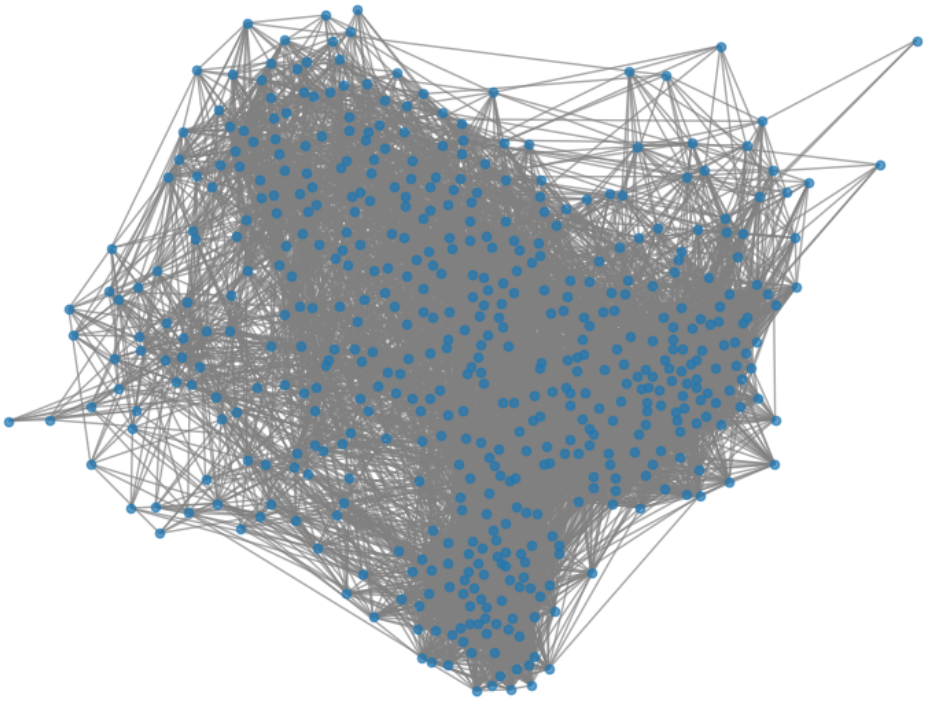


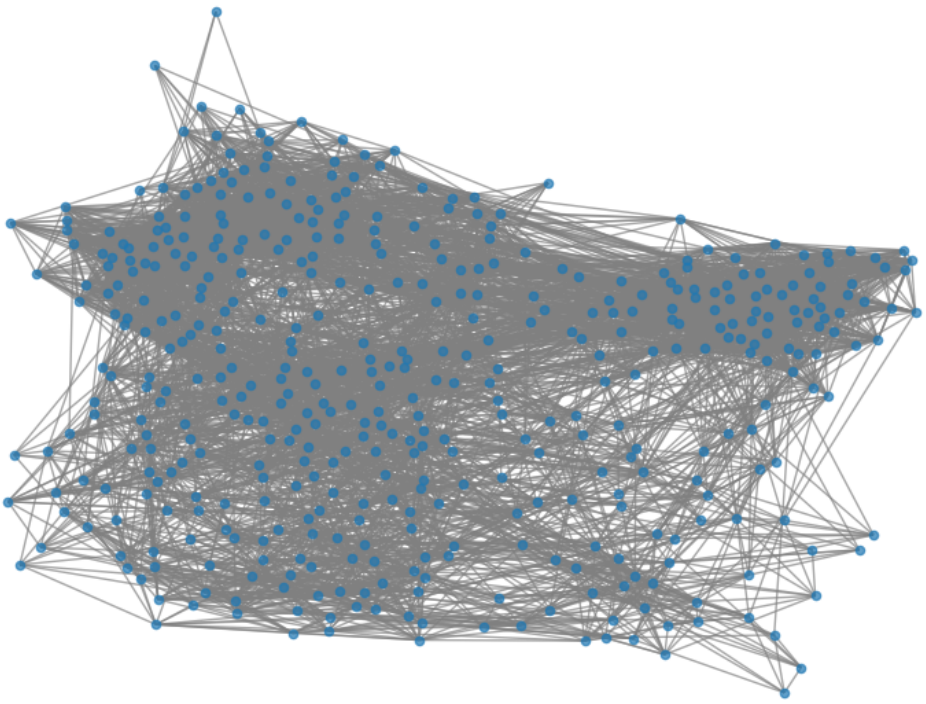
Si quitamos los hubs, hay una gran diferencia en la conectividad y por lo tanto la funcionalidad del cerebro se vería afectada si los quitamos, también lo podemos ver con el índice de mundo pequeño ya que aumentamos las distancias promedio en la red, disminuyendo la conectividad global. Se pierden la mayoría de las conexiones.

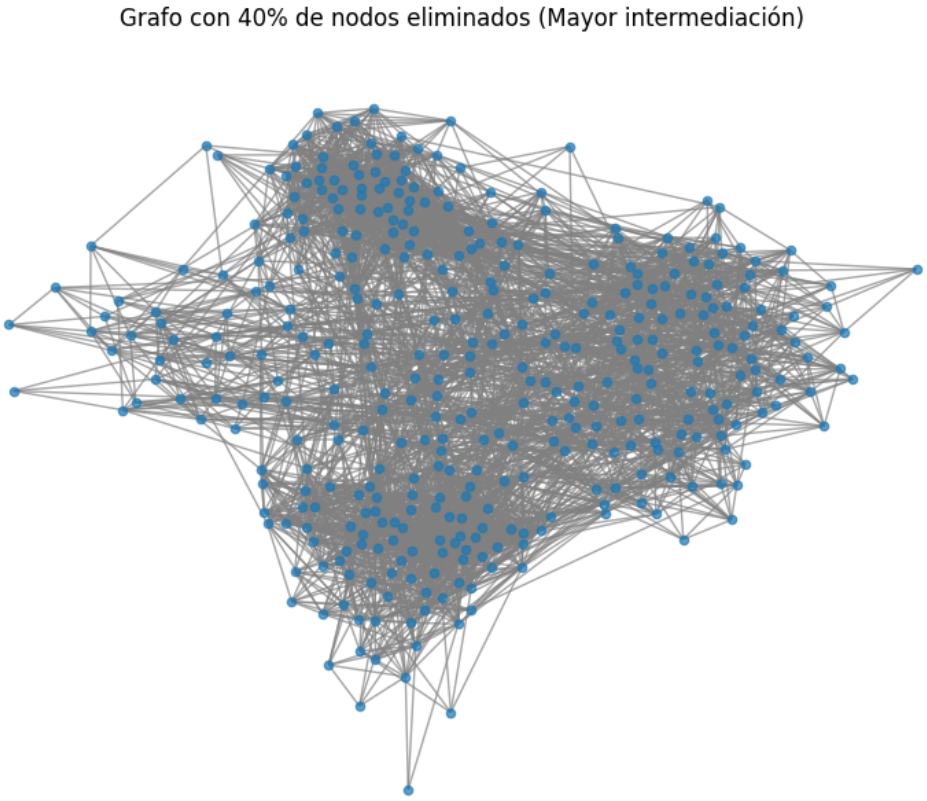
7 Ejercicio 7

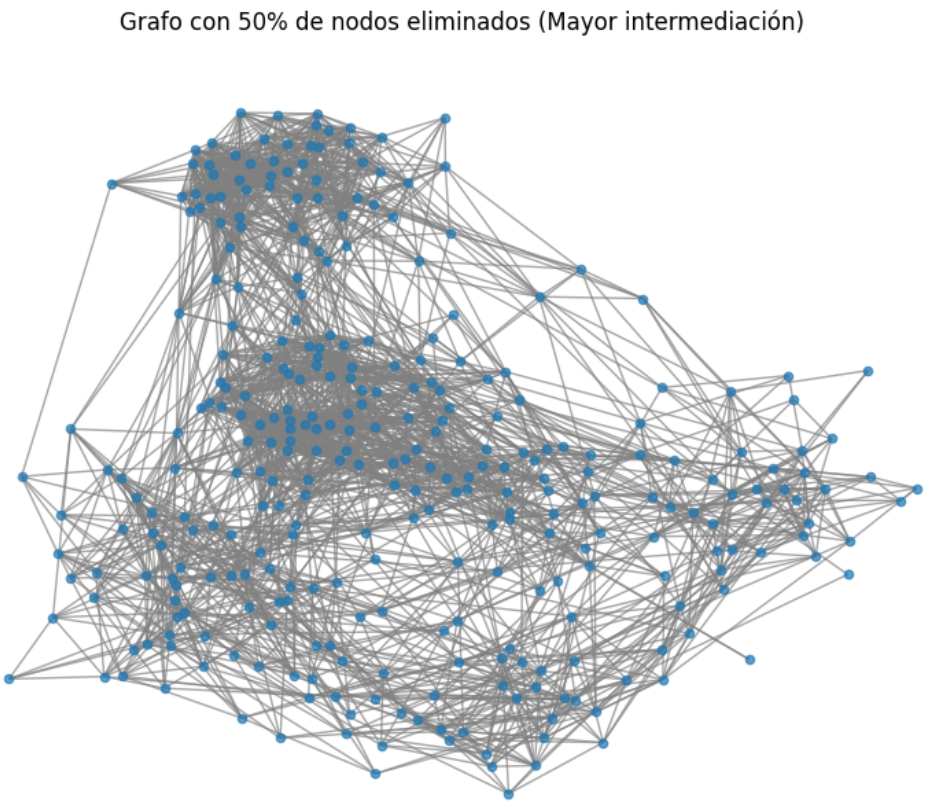
[22]:
```
percentages = [10, 20, 30, 40, 50]

for p in percentages:
    G_temp = G.copy()
    num_nodes_to_remove = int(len(G) * (p / 100))

    # Calcular intermediación y seleccionar nodos a eliminar
    betweenness = nx.betweenness_centrality(G_temp)
    nodes_to_remove = sorted(betweenness, key=betweenness.get, reverse=True)[:
 ↪num_nodes_to_remove]

    # Eliminar nodos
    G_temp.remove_nodes_from(nodes_to_remove)

    # Visualizar grafo resultante
    plt.figure(figsize=(8, 6))
    nx.draw_spring(G_temp, node_size=20, edge_color='gray', alpha=0.7)
    plt.title(f"Grafo con {p}% de nodos eliminados (Mayor intermediación)")
    plt.show()
```


Grafo con 10% de nodos eliminados (Mayor intermediación)


Grafo con 20% de nodos eliminados (Mayor intermediación)

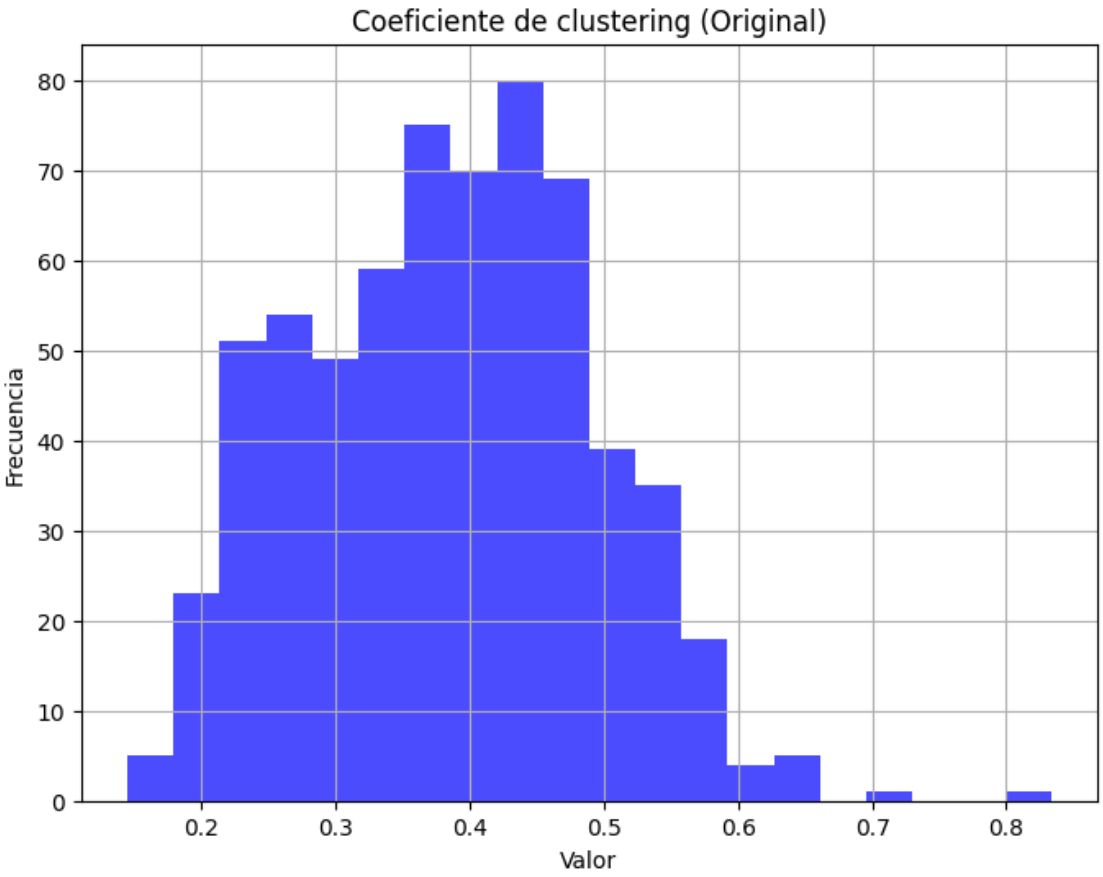

Grafo con 30% de nodos eliminados (Mayor intermediación)

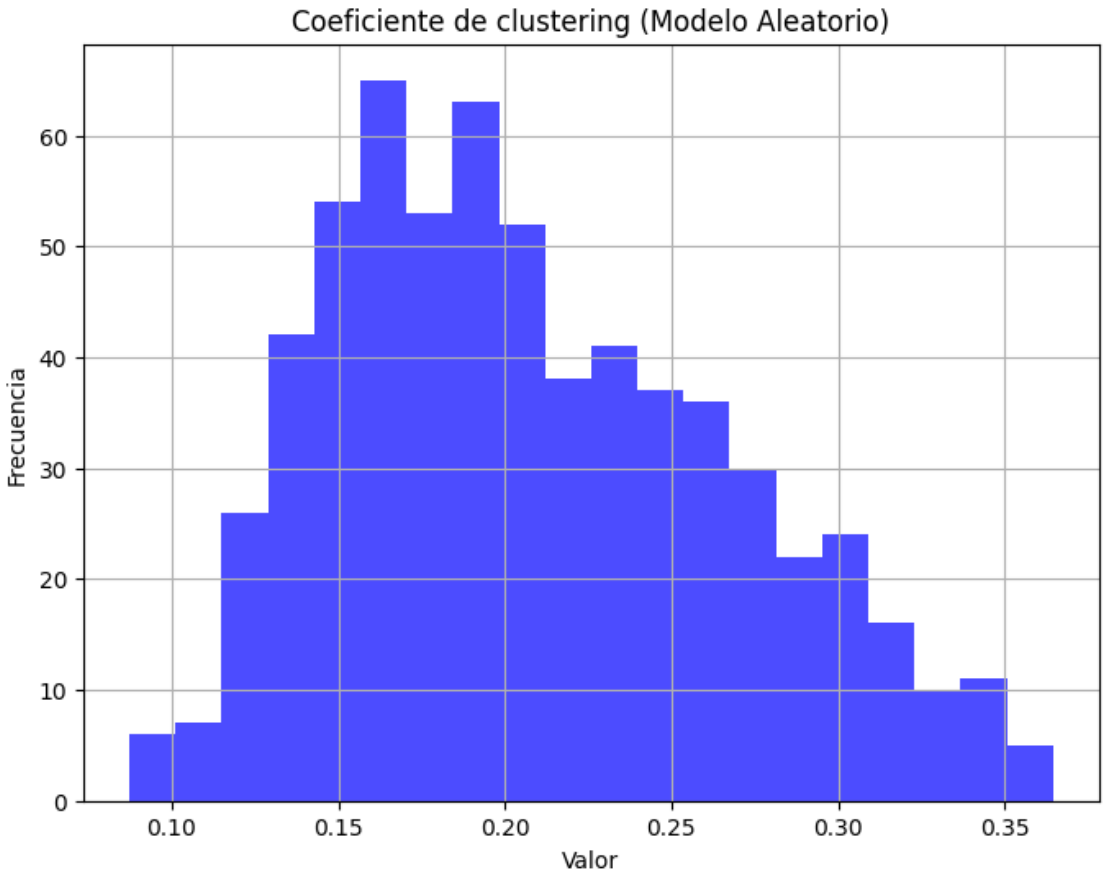

Grafo con 40% de nodos eliminados (Mayor intermediación)



8 Ejercicio 8

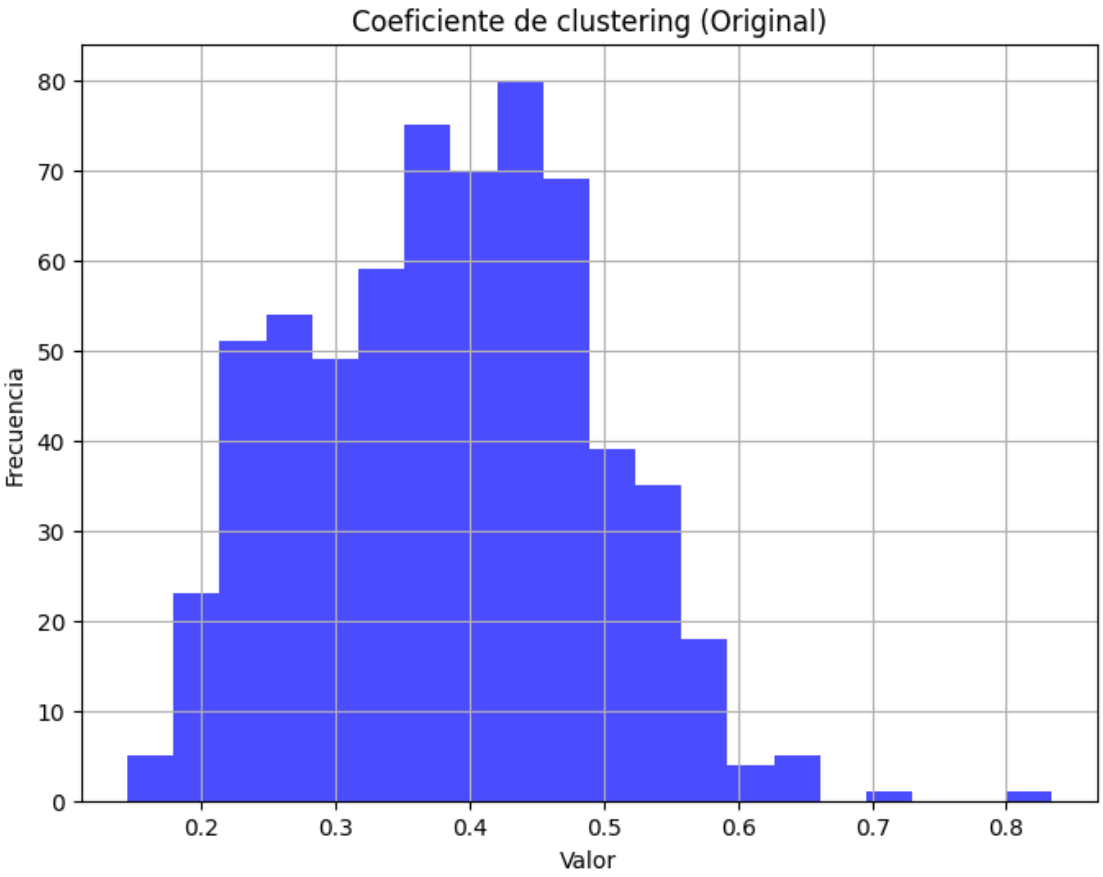
[29]:
```
random_graph = nx.from_numpy_array(np.random.permutation(coactivation_matrix))
props_random = calculate_topological_properties(random_graph)

# Comparar propiedades
plot_node_properties(props_original["clustering_coeffs"], "Coeficiente de␣
 ↪clustering (Original)")
plot_node_properties(props_random["clustering_coeffs"], "Coeficiente de␣
 ↪clustering (Modelo Aleatorio)")
```

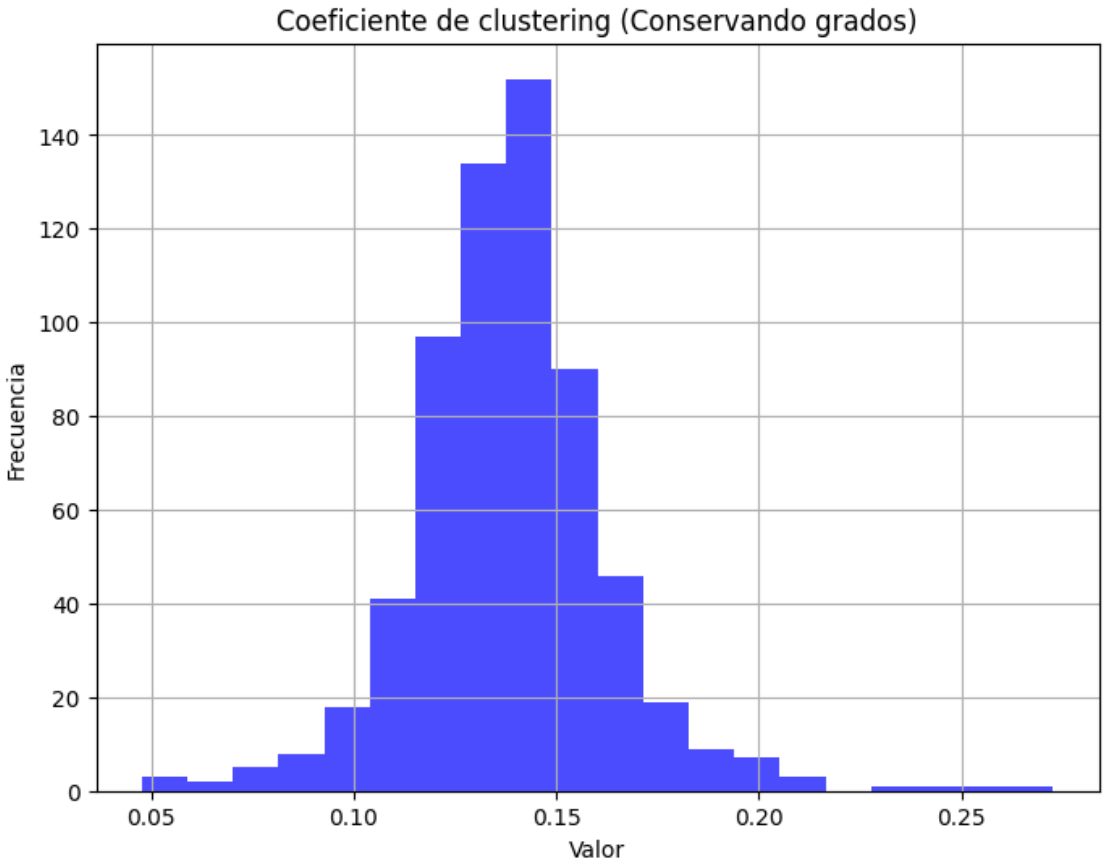

Coeficiente de clustering (Original)
Frecuencia
Valor
0
10
20
30
40
50
60
70
80
0.2
0.3
0.4
0.5
0.6
0.7
0.8



9 Ejercicio 9

[30]:
```
# Usamos la distribución de grados de la red original
degree_sequence = [d for _, d in G.degree()]
configuration_model = nx.configuration_model(degree_sequence)  # Modelo␣
 ↪conservando grados
configuration_model = nx.Graph(configuration_model)  # Convertir a grafo simple
props_configuration = calculate_topological_properties(configuration_model)

# Comparar propiedades
plot_node_properties(props_original["clustering_coeffs"], "Coeficiente de␣
 ↪clustering (Original)")
plot_node_properties(props_configuration["clustering_coeffs"], "Coeficiente de␣
 ↪clustering (Conservando grados)")
```

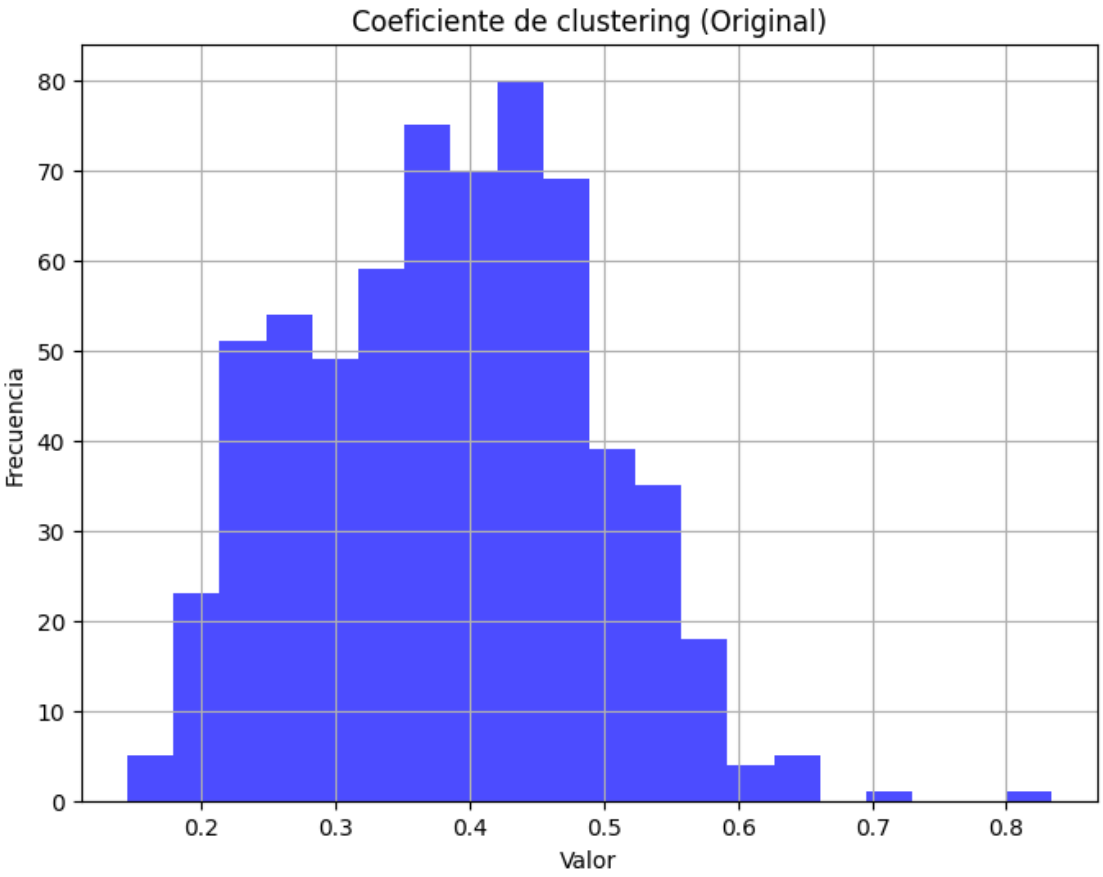

Coeficiente de clustering (Original)
Frecuencia
Valor
0
10
20
30
40
50
60
70
80
0.2
0.3
0.4
0.5
0.6
0.7
0.8

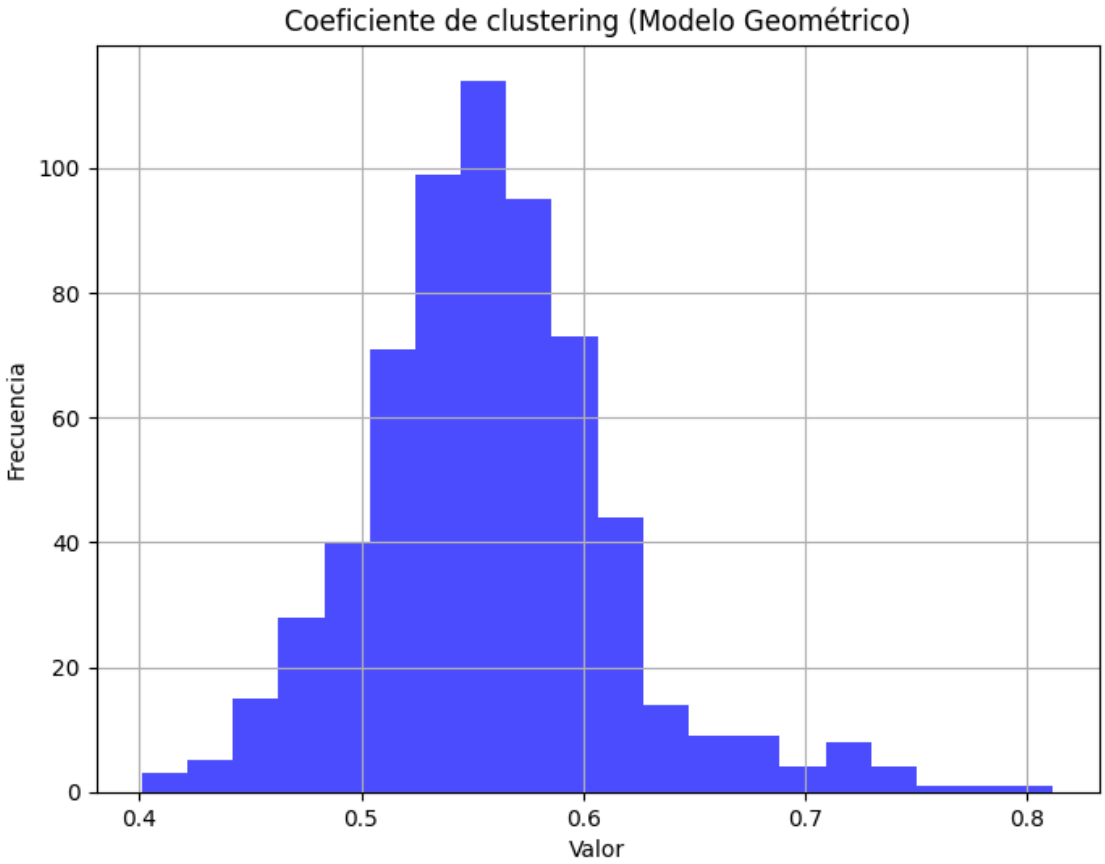


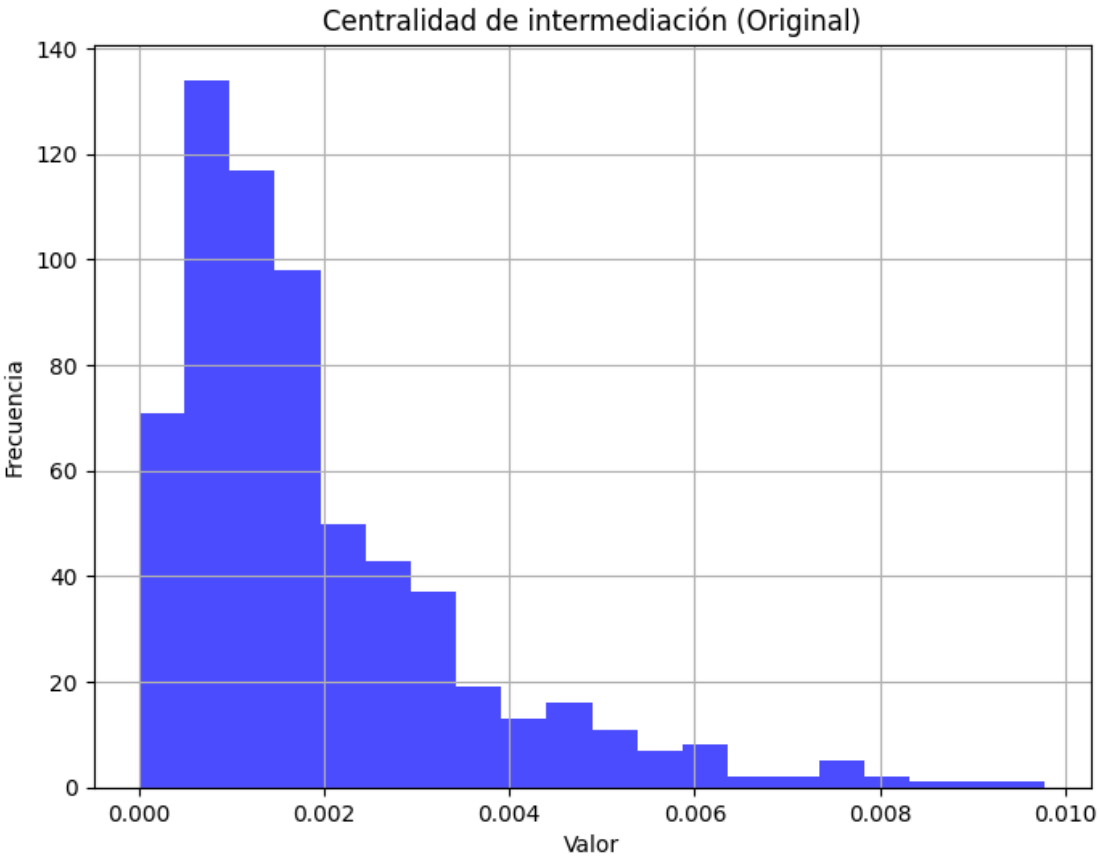
10 Ejercicio 10

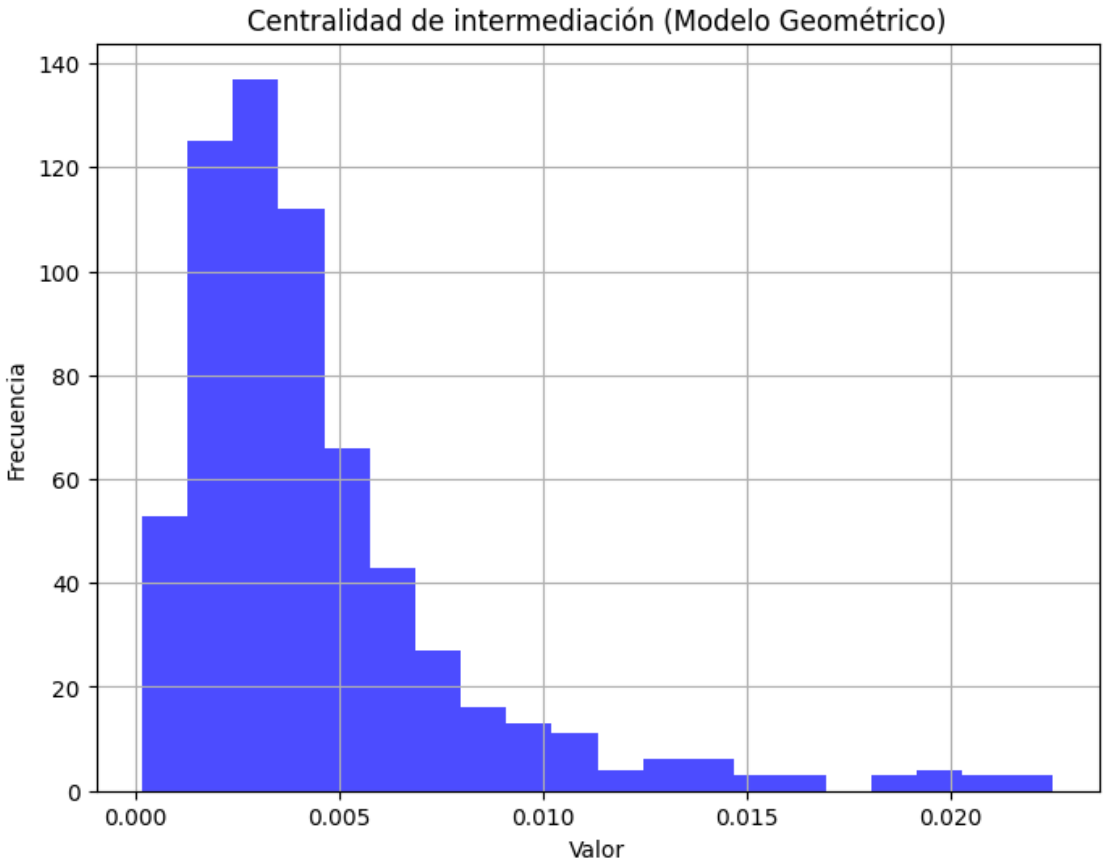
[28]:
```
coords = mat_data['Coord']

# Calcular la distancia geométrica entre los nodos
distances = squareform(pdist(coords))  # Calcula las distancias entre todas las␣
 ↪coordenadas

# Probabilidad inversamente proporcional a la distancia
probability_matrix = 1 / (distances + np.eye(len(G.nodes)))  # Evitar␣
 ↪divisiones por cero con np.eye
probability_matrix /= probability_matrix.sum(axis=1, keepdims=True)  #␣
 ↪Normalizar por filas

# Crear el modelo nulo (grafo geométrico)
threshold = np.percentile(probability_matrix, 95)  # Ajustar el umbral al␣
 ↪percentil deseado
```

```
connection_matrix = (probability_matrix > threshold).astype(int)  # Crear␣
 ↪matriz de adyacencia
np.fill_diagonal(connection_matrix, 0)  # Eliminar autoconexiones

geometric_graph = nx.from_numpy_array(connection_matrix)  # Crear el grafo␣
 ↪geométrico

# Función para calcular propiedades topológicas
def calculate_topological_properties(graph):
    clustering_coeffs = nx.clustering(graph)
    degrees = dict(graph.degree())
    betweenness = nx.betweenness_centrality(graph)
    closeness = nx.closeness_centrality(graph)

    return {
        "clustering_coeffs": clustering_coeffs,
        "degrees": degrees,
        "betweenness": betweenness,
        "closeness": closeness,
    }

# Calcular propiedades topológicas del grafo geométrico
props_geometric = calculate_topological_properties(geometric_graph)

# Comparar con las propiedades del grafo original
props_original = calculate_topological_properties(G)

# Función para graficar propiedades de nodos
def plot_node_properties(property_dict, title):
    plt.figure(figsize=(8, 6))
    plt.hist(list(property_dict.values()), bins=20, color='blue', alpha=0.7)
    plt.title(title)
    plt.xlabel("Valor")
    plt.ylabel("Frecuencia")
    plt.grid(True)
    plt.show()

# Graficar comparación de propiedades
plot_node_properties(props_original["clustering_coeffs"], "Coeficiente de␣
 ↪clustering (Original)")
plot_node_properties(props_geometric["clustering_coeffs"], "Coeficiente de␣
 ↪clustering (Modelo Geométrico)")
plot_node_properties(props_original["betweenness"], "Centralidad de␣
 ↪intermediación (Original)")
plot_node_properties(props_geometric["betweenness"], "Centralidad de␣
 ↪intermediación (Modelo Geométrico)")
```


Coeficiente de clustering (Original)
Frecuencia
Valor
0
10
20
30
40
50
60
70
80
0.2
0.3
0.4
0.5
0.6
0.7
0.8


Coeficiente de clustering (Modelo Geométrico)
Frecuencia
Valor
0
20
40
60
80
100
0.4
0.5
0.6
0.7
0.8


Centralidad de intermediación (Original)
Frecuencia
Valor
0
20
40
60
80
100
120
140
0.000
0.002
0.004
0.006
0.008
0.010



Las conexiones a larga distancia ayudan primeramente a unir dos áreas anatómicamente separadas, consigo ayudan a integrar información, coordinar actividades y ayudar a procesos cognitivos complejos. También pueden servir de sustento cuando las vías locales se ven afectadas adquiriendo características plásticas y de resiliencia contribuyendo al mantenimiento de la eficiencia de la red aún cuando haya fallos en la misma. En conclusión, las conexiones a larga distancia ayudan a un mayor funcionamiento global tanto en situaciones normales así como cuando el cerebro sufre algún daño.

11 Conclusión

Durante el curso pudimos entender, desde modelos matemáticos y su aplicación en programación (python y matlab), cómo es la organización del cerebro y sus propiedades conectivas, porqué es importante la conectividad entre áreas cercanas y lejanas, qué son los hubs y su función en la organización y procesos cerebrales. A mi parecer esto es muy importante para nuestra formación en neurociencias ya que esto nos ayudará en nuestros posibles anáisis de datos pero también a a hora de leer artículos científicos que tengan que ver con la conectividad cerebral y poderlos entender mejor y al menos tener una idea de cómo lograron llegar a esos resultados e imágenes y así tener un mayor pensamiento crítico de lo que se nos muestra en las investigaciones y sobre nuestros propios trabajos. Otra cosa a recalcar es que el entendimiento de la teoría de grafos no solo explica la conectividad cerebral sino que también se puede aplicar a muchas otras áreas de la ciencia por lo

que considero de relevante conocerlo para utilizarlo en otros posibles aspectos que me interesen en un futuro. Me parece una materia súmamente interesante pero también muy tediosa ya que se necesita de lógica y conocimientos de programación para realmente entenderla, sin embargo, aprendí los conocimientos básicos tanto de python como de matlab, los cuales sé que me servirán en un futuro. Espero poder seguir aprendiendo más de programación :)