# proyecto-melannie

November 29, 2024

[138]:
```
import scipy.io
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import networkx as nx
import pandas as pd
import networkx.algorithms.community as nx_comm
from matplotlib.animation import FuncAnimation
from IPython.display import HTML
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
```

[140]:
```
datos = r"C:\Users\melan\Downloads\Coactivation_matrix.mat"
```

[142]:
```
m_jason = scipy.io.loadmat(datos)
```

[144]:
```
m_dict = {k:v for k, v in m_jason.items() if k[0] != '_'}
m_dict = {k:v for k, v in m_jason.items() if k[0] != '_'}
m_dict
```

[144]:
```
{'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]])}
```

[146]:
```
m_dict.keys()
```

[146]:
```
dict_keys(['Coactivation_matrix', 'Coord'])
```

[148]:
```
# DINENSIONES
comat= m_jason['Coactivation_matrix']
coordenadas = m_jason['Coord']
comat_shape = comat.shape
coord_shape= coordenadas.shape

comat_shape, coord_shape
```

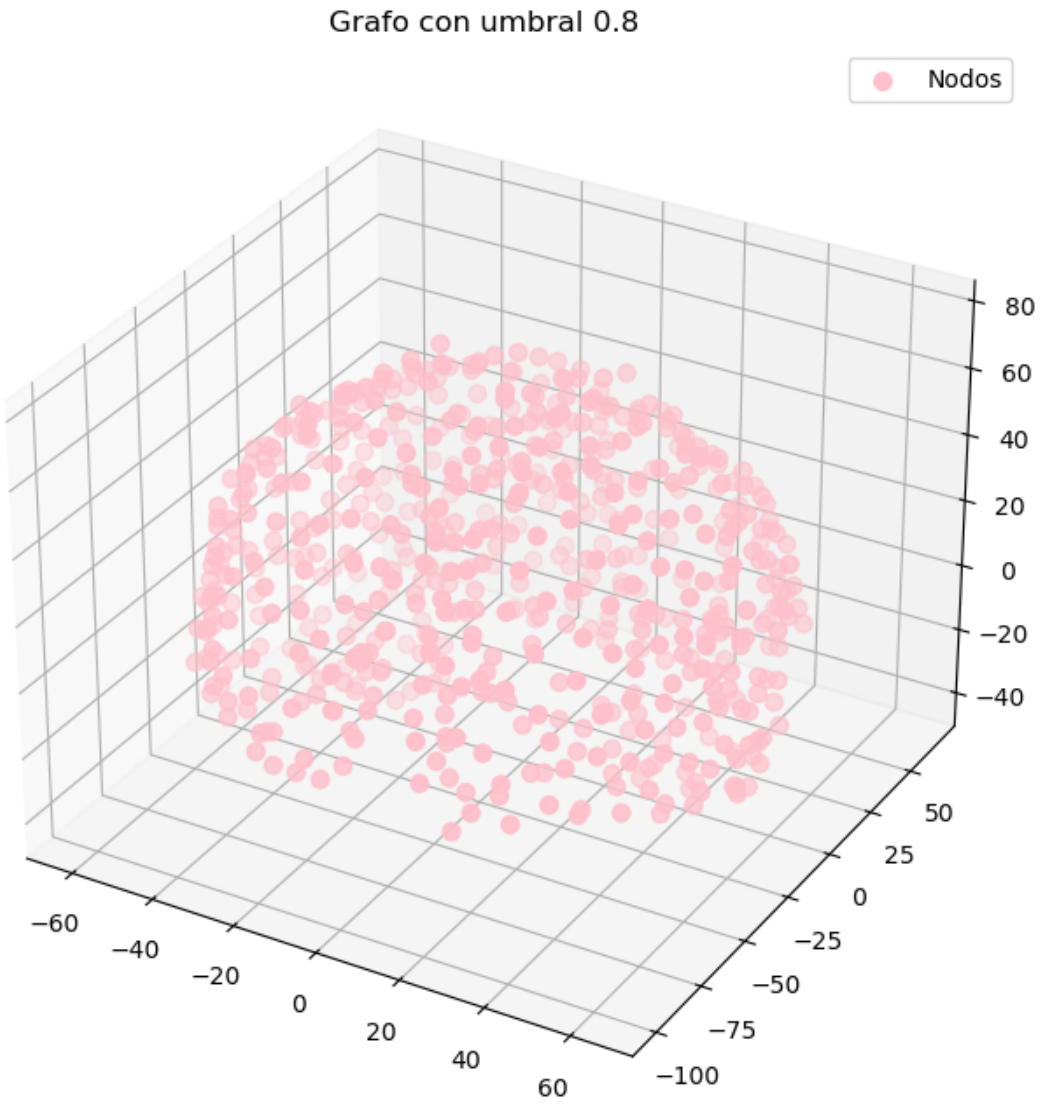
[148]:
```
((638, 638), (638, 3))
```

[150]:
```
# EJERCICIO 1
# Definir grafos con la matriz estableciendo umbrales de coactivación de 0.8, 0.
 ↪9 y 1 y graficar cada grafo.
# Añadir las coordenadas tridimensionales (incluidas en el archivo.mat).
```

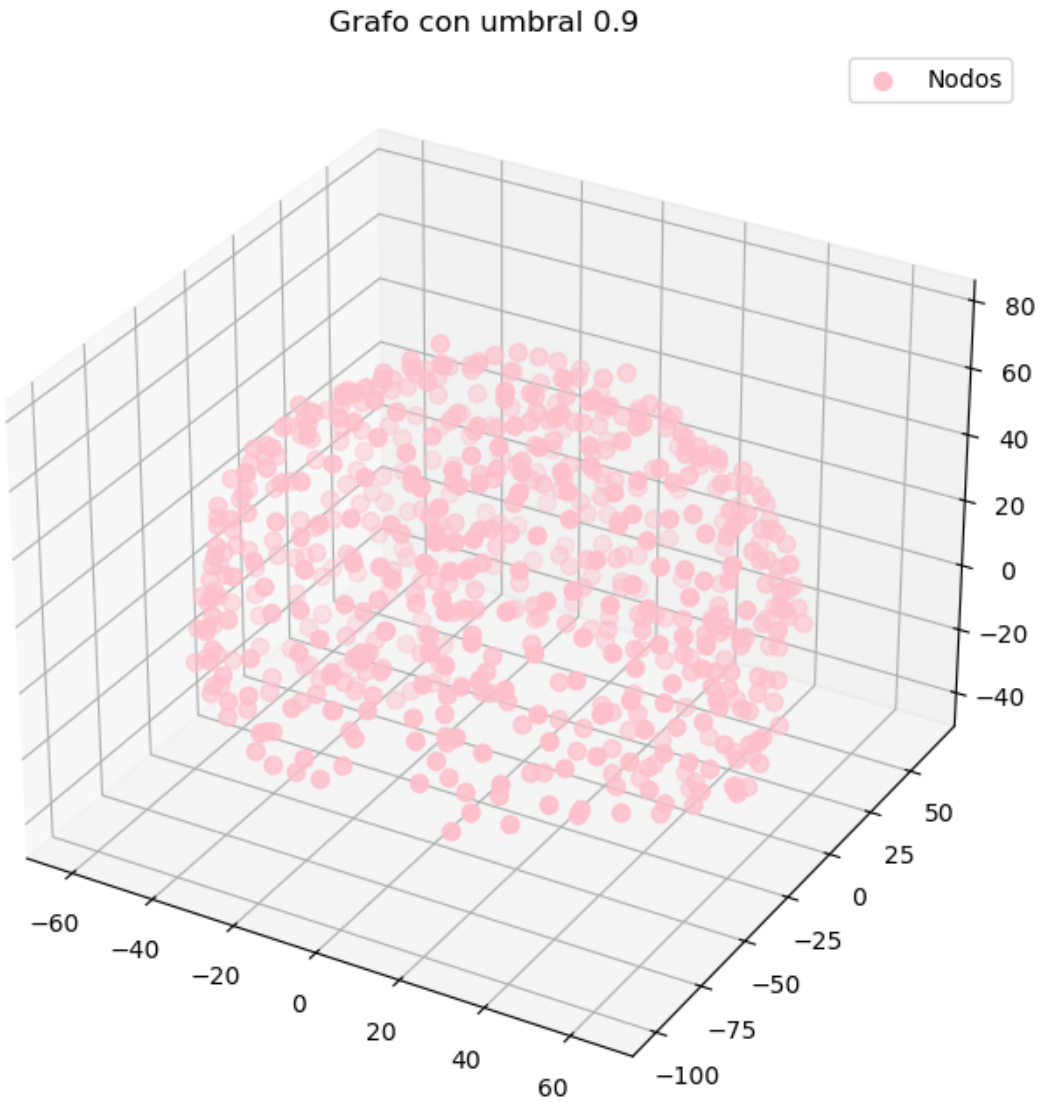
[152]:
```
# Función -> grafos
def generate_graph(mat, umbral, coordenadas):
    # Mat. adj. binaria
    matadj = (mat >= umbral).astype(int)
    np.fill_diagonal(matadj, 0)

    G = nx.from_numpy_array(matadj)

    # Coordenadas 3D -> nodos
    for i, coord in enumerate(coordenadas):
        G.nodes[i]['pos'] = coord

    return G

# Umbrales 0.8, 0.9 y 1.0
umbrales = [0.8, 0.9, 1.0]
graph = {umbral: generate_graph(comat, umbral, coordenadas) for umbral in␣
 ↪umbrales}

# Grafo 3D
def plot_graph_3d(G, title):
    fig = plt.figure(figsize=(10, 8))
    ax = fig.add_subplot(111, projection='3d')
```

```
    # Posiciones
    positions = nx.get_node_attributes(G, 'pos')
    if not positions:
        print("Las posiciones 3D en los nodos no están definidas.")
        return

    x, y, z = np.array(list(positions.values())).T

    #GRAFICAR
    # Nodos
    ax.scatter(x, y, z, c='pink', s=50, label='Nodos')

    # Aristas
    for edge in G.edges():
        x_edge, y_edge, z_edge = zip(pos[edge[0]], pos[edge[1]])
        ax.plot(x_edge, y_edge, z_edge, c='b', alpha=0.5)

    ax.set_title(title)
    plt.legend()
    plt.show()

# Graficar los grafos generados
for umbral, graph in graphs.items():
    print(f"Grafo con umbral {umbral}: {graph.number_of_edges()} aristas y␣
 ↪{graph.number_of_nodes()} nodos.")
    plot_graph_3d(graph, f"Grafo con umbral {umbral}")
```

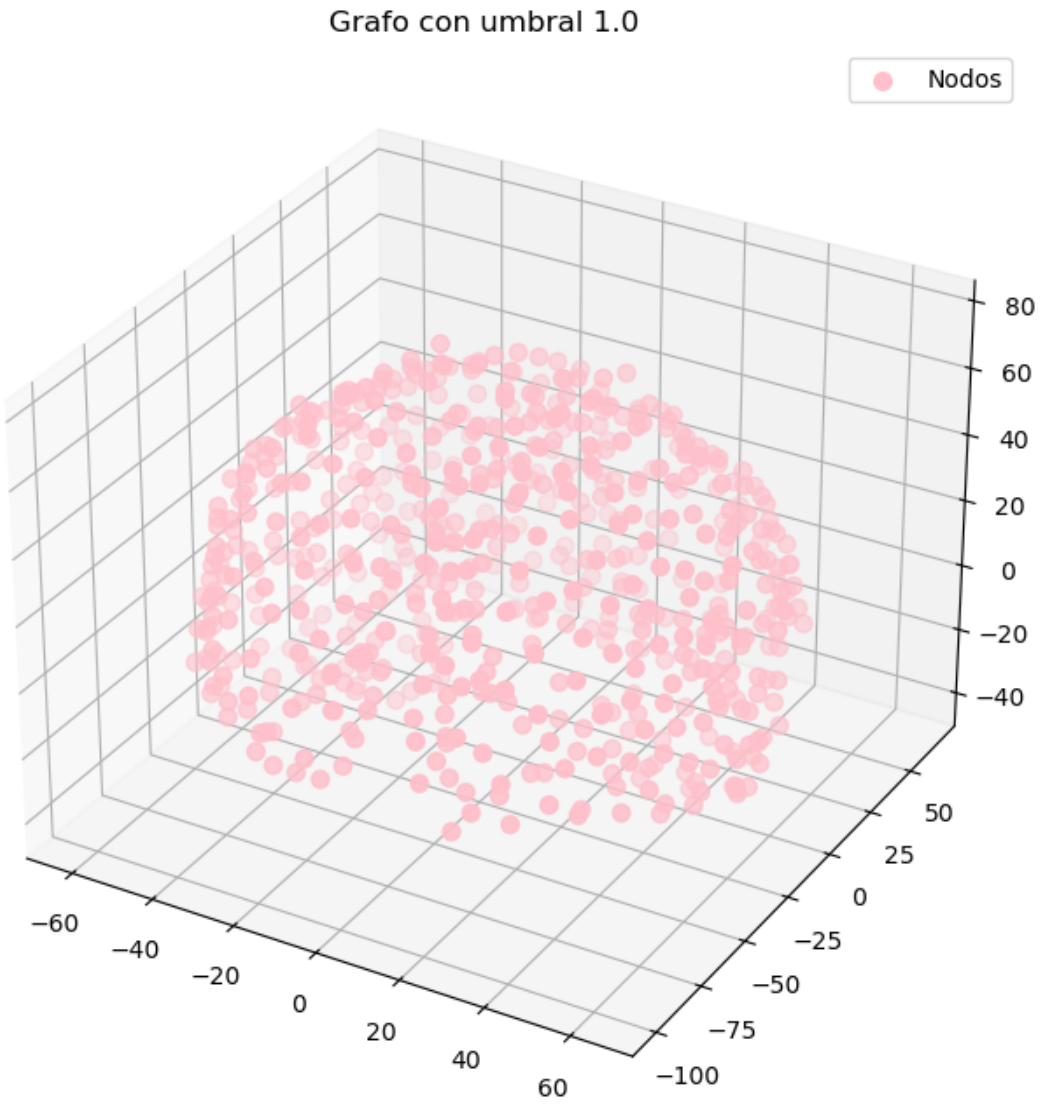
```
Grafo con umbral 0.8: 0 aristas y 638 nodos.
```




Grafo con umbral 0.9: 0 aristas y 638 nodos.




Grafo con umbral 1.0: 0 aristas y 638 nodos.




[153]:
```
# EJERCICIO 2
# Con uno de los grafos en el punto uno con umbral 0.9, generar una animación␣
 ↪donde se haga girar 360° el grafo del cerebro para visualizar
# las conexiones establecidas
```

[ ]:
```
```

[155]:
```
from matplotlib.animation import FuncAnimation
def animacion_grafo(G, ej2):
    fig = plt.figure()
    ax= fig.add_subplot(111, projection ='3d')
    pos= nx.get_node_attributes(G, 'pos')
```

```
    x, y, z= np.array(list(pos.values())).T

    #nodos y aristas
    scatter= ax.scatter(x,y,z, c='purple', s=20, label='Nodos')
    lines = []
    for edge in G.edges():
        x_edge, y_edge, z_edge = zip(pos[edge[0]],pos[edge[1]])
        lines.append(ax.plot( x_edge, y_edge, z_edge, c= 'b', alpha= 0.5)[0])

    ax.set_title("Grafo con umbral 0.9")

    # rotación, animación y guardar imagen
    def update(frame):
        ax.view_init(elev=10, azim=frame)
        return scatter, *lines

    animation = FuncAnimation(fig, update, frames= np.arange(0, 360, 2),␣
 ↪interval= 50, blit= False)

    animation.save(ej2, writer='pillow', fps=20)
    plt.close(fig)

r_archivo = r"C:\Users\melan\OneDrive\Documentos\Escuela\grafo09.gif"
grafo09= graphs[0.9]
animacion_grafo(grafo09, r_archivo)
```

[156]:
```
from IPython.display import Image
Image(r"C:\Users\melan\OneDrive\Documentos\Escuela\grafo09.gif")
```

[156]:
```
<IPython.core.display.Image object>
```

[217]:
```
#  EJERCICIO 3. Encontrar los hubs del grafo, y establecer el tamaño del nodo␣
 ↪proporcional al valor del grado
```

[261]:
```
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np

def plot_hubs(G, titulo):
    #Posiciones de los nodos (obtenerlas)
    pos = nx.get_node_attributes(G, 'pos')
    x, y, z = np.array(list(pos.values())).T

    # Calcular grados de los nodos
    grados = dict(G.degree())
    print("Grados de los nodos:", grados)
```

```
    # Evitar división por cero => me estaba dando este error y para evitarlo
    gradomax = max(grados.values()) if grados and max(grados.values()) > 0 else␣
↪1
    print(f"Grado máximo: {gradomax}")

    # Calcular tamaño de nodos proporcional al grado
    tamano_min, tamano_max = 6, 200
    tamano_nodo = [
        tamano_min + (grado / gradomax) * (tamano_max - tamano_min) for grado␣
↪in grados.values()
    ]

    # Verificar los tamaños calculados
    print(f"Tamaño de nodos calculado: {tamano_nodo}")

    # Calcular el percentil 90 para el umbral
    umbral = np.percentile(list(grados.values()), 90) if grados else 0
    print(f"Umbral (percentil 90): {umbral}")

    # Encontrar hubs
    nodos_hubs = [node for node, grado in grados.items() if grado >= umbral]
    print(f"Nodos hubs: {nodos_hubs}")

    # Asignar colores -> rosa para hubs
    nodecol = ['pink' if node in nodos_hubs else 'purple' for node in G.nodes()]
    print("Colores de los nodos:", nodecol)

    # Figura 3D
    fig = plt.figure(figsize=(12, 8))
    ax = fig.add_subplot(111, projection='3d')

    # Nodos
    scatter = ax.scatter(x, y, z, c=nodecol, s=tamano_nodo, label='Nodos')

    # Aristas
    for edge in G.edges():
        x_edge, y_edge, z_edge = zip(pos[edge[0]], pos[edge[1]])
        ax.plot(x_edge, y_edge, z_edge, c='b', alpha=0.5, linewidth=0.2)

    # Especificaciones gráfica
    ax.set_title(titulo)
    ax.legend(["Hubs rosa, morado el resto"])
    plt.show()

# Llamar función
grafo02 = generate_graph(comat, 0.2, coordenadas)
plot_hubs(grafo02, "Hubs del grafo con umbral 0.2")
```

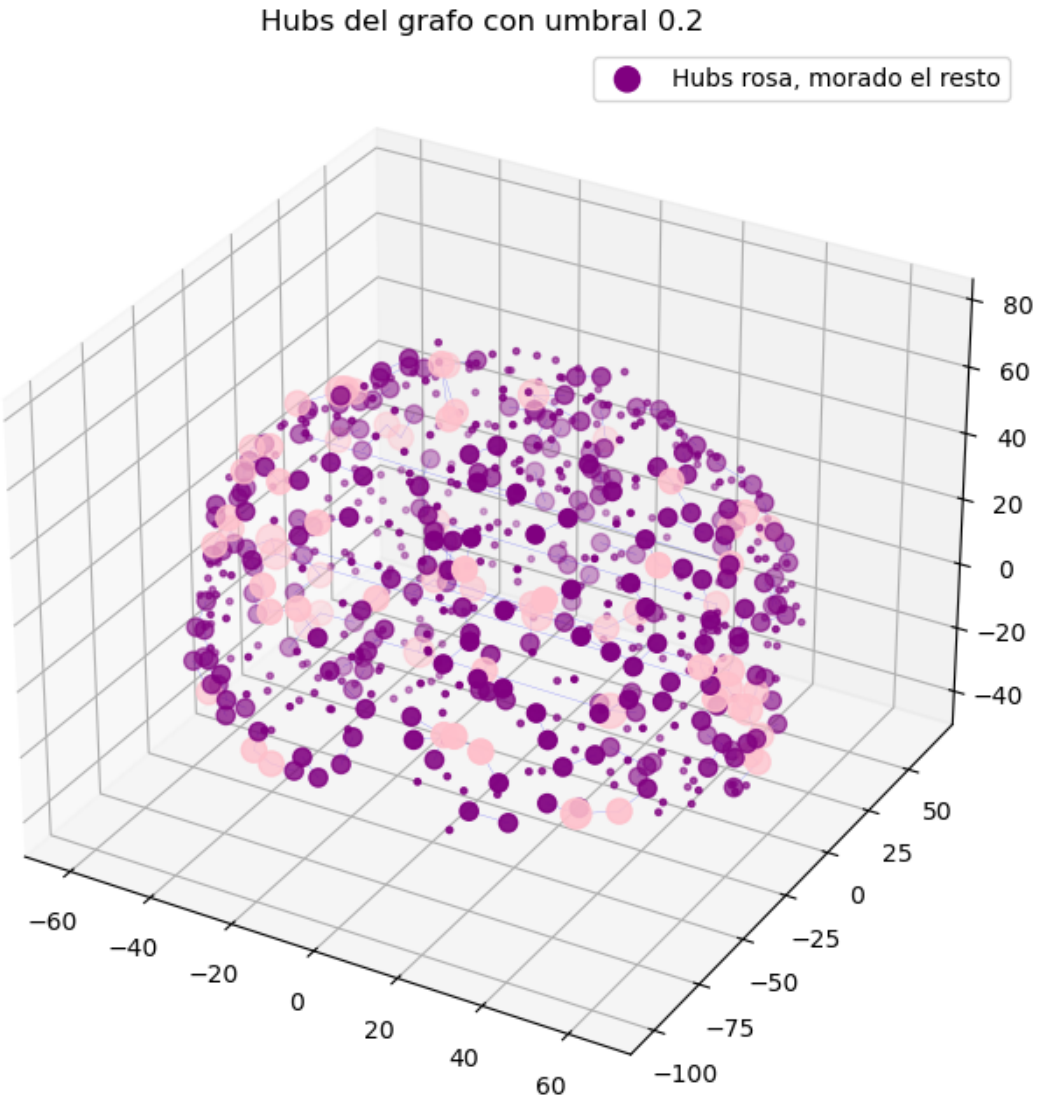
Grados de los nodos: {0: 0, 1: 0, 2: 1, 3: 1, 4: 0, 5: 1, 6: 0, 7: 1, 8: 1, 9: 1, 10: 0, 11: 1, 12: 1, 13: 0, 14: 0, 15: 0, 16: 1, 17: 0, 18: 0, 19: 1, 20: 0, 21: 0, 22: 0, 23: 1, 24: 0, 25: 1, 26: 0, 27: 1, 28: 0, 29: 0, 30: 0, 31: 0, 32: 0, 33: 0, 34: 0, 35: 1, 36: 0, 37: 0, 38: 2, 39: 0, 40: 0, 41: 1, 42: 2, 43: 1, 44: 0, 45: 1, 46: 1, 47: 0, 48: 0, 49: 1, 50: 0, 51: 0, 52: 0, 53: 1, 54: 1, 55: 0, 56: 0, 57: 0, 58: 0, 59: 0, 60: 0, 61: 0, 62: 2, 63: 1, 64: 0, 65: 0, 66: 1, 67: 0, 68: 0, 69: 2, 70: 1, 71: 1, 72: 0, 73: 1, 74: 0, 75: 0, 76: 0, 77: 0, 78: 1, 79: 0, 80: 0, 81: 1, 82: 0, 83: 1, 84: 0, 85: 0, 86: 1, 87: 0, 88: 0, 89: 1, 90: 0, 91: 0, 92: 2, 93: 0, 94: 1, 95: 0, 96: 0, 97: 1, 98: 0, 99: 0, 100: 0, 101: 1, 102: 0, 103: 0, 104: 0, 105: 1, 106: 0, 107: 0, 108: 0, 109: 0, 110: 0, 111: 1, 112: 0, 113: 1, 114: 0, 115: 0, 116: 0, 117: 1, 118: 0, 119: 1, 120: 0, 121: 1, 122: 0, 123: 1, 124: 2, 125: 0, 126: 1, 127: 1, 128: 1, 129: 0, 130: 1, 131: 0, 132: 0, 133: 0, 134: 0, 135: 1, 136: 1, 137: 1, 138: 0, 139: 0, 140: 0, 141: 0, 142: 0, 143: 0, 144: 0, 145: 1, 146: 1, 147: 1, 148: 1, 149: 0, 150: 2, 151: 0, 152: 0, 153: 1, 154: 1, 155: 1, 156: 0, 157: 1, 158: 1, 159: 0, 160: 0, 161: 0, 162: 0, 163: 1, 164: 1, 165: 0, 166: 0, 167: 0, 168: 0, 169: 0, 170: 0, 171: 0, 172: 0, 173: 1, 174: 1, 175: 0, 176: 0, 177: 0, 178: 1, 179: 0, 180: 1, 181: 0, 182: 0, 183: 0, 184: 0, 185: 0, 186: 0, 187: 0, 188: 0, 189: 0, 190: 0, 191: 0, 192: 0, 193: 0, 194: 0, 195: 0, 196: 0, 197: 0, 198: 0, 199: 1, 200: 0, 201: 0, 202: 1, 203: 0, 204: 0, 205: 1, 206: 0, 207: 0, 208: 0, 209: 1, 210: 1, 211: 1, 212: 0, 213: 0, 214: 0, 215: 0, 216: 0, 217: 3, 218: 1, 219: 1, 220: 1, 221: 2, 222: 0, 223: 2, 224: 0, 225: 1, 226: 4, 227: 0, 228: 0, 229: 0, 230: 2, 231: 1, 232: 0, 233: 0, 234: 0, 235: 2, 236: 0, 237: 1, 238: 0, 239: 0, 240: 0, 241: 0, 242: 0, 243: 1, 244: 0, 245: 2, 246: 0, 247: 0, 248: 0, 249: 0, 250: 1, 251: 0, 252: 0, 253: 0, 254: 0, 255: 0, 256: 1, 257: 0, 258: 1, 259: 2, 260: 2, 261: 1, 262: 1, 263: 3, 264: 1, 265: 1, 266: 1, 267: 0, 268: 0, 269: 1, 270: 0, 271: 1, 272: 0, 273: 0, 274: 1, 275: 2, 276: 0, 277: 1, 278: 0, 279: 1, 280: 1, 281: 2, 282: 1, 283: 1, 284: 1, 285: 1, 286: 0, 287: 0, 288: 0, 289: 0, 290: 2, 291: 0, 292: 2, 293: 0, 294: 1, 295: 1, 296: 2, 297: 1, 298: 1, 299: 1, 300: 0, 301: 1, 302: 1, 303: 1, 304: 2, 305: 1, 306: 0, 307: 0, 308: 1, 309: 2, 310: 0, 311: 0, 312: 0, 313: 1, 314: 0, 315: 0, 316: 0, 317: 1, 318: 1, 319: 2, 320: 2, 321: 2, 322: 0, 323: 0, 324: 0, 325: 0, 326: 0, 327: 2, 328: 1, 329: 1, 330: 2, 331: 2, 332: 1, 333: 1, 334: 0, 335: 0, 336: 1, 337: 1, 338: 0, 339: 1, 340: 1, 341: 1, 342: 1, 343: 1, 344: 2, 345: 1, 346: 0, 347: 1, 348: 1, 349: 0, 350: 1, 351: 1, 352: 2, 353: 1, 354: 0, 355: 0, 356: 2, 357: 0, 358: 2, 359: 1, 360: 0, 361: 1, 362: 1, 363: 0, 364: 1, 365: 1, 366: 0, 367: 0, 368: 0, 369: 2, 370: 0, 371: 0, 372: 1, 373: 1, 374: 1, 375: 0, 376: 0, 377: 0, 378: 0, 379: 0, 380: 1, 381: 0, 382: 0, 383: 1, 384: 0, 385: 1, 386: 1, 387: 1, 388: 1, 389: 0, 390: 0, 391: 0, 392: 0, 393: 2, 394: 1, 395: 1, 396: 0, 397: 0, 398: 1, 399: 0, 400: 1, 401: 0, 402: 1, 403: 1, 404: 3, 405: 0, 406: 1, 407: 2, 408: 2, 409: 1, 410: 1, 411: 0, 412: 1, 413: 0, 414: 1, 415: 0, 416: 2, 417: 0, 418: 1, 419: 2, 420: 1, 421: 0, 422: 0, 423: 0, 424: 0, 425: 0, 426: 0, 427: 1, 428: 2, 429: 0, 430: 1, 431: 1, 432: 0, 433: 0, 434: 0, 435: 0, 436: 1, 437: 0, 438: 0, 439: 0, 440: 0, 441: 1, 442: 0, 443: 1, 444: 0, 445: 1, 446: 1, 447: 0, 448: 1, 449: 0, 450: 1, 451: 1, 452: 0, 453: 0, 454: 0, 455: 0, 456: 2, 457: 0, 458: 1, 459: 0, 460: 2, 461: 1, 462: 0, 463: 1, 464: 0, 465: 2, 466: 1, 467: 1, 468: 0, 469: 0, 470: 0, 471: 0, 472: 1, 473: 1, 474: 0, 475: 0, 476: 0, 477: 1, 478: 0, 479: 0, 480: 0, 481: 0, 482: 3, 483: 0, 484: 0, 485: 1, 486: 0, 487: 0, 488: 0, 489: 0, 490: 0,

```
491: 1, 492: 1, 493: 1, 494: 1, 495: 0, 496: 0, 497: 0, 498: 0, 499: 1, 500: 1,
501: 1, 502: 1, 503: 0, 504: 1, 505: 1, 506: 1, 507: 1, 508: 0, 509: 0, 510: 1,
511: 1, 512: 1, 513: 1, 514: 1, 515: 0, 516: 0, 517: 0, 518: 0, 519: 1, 520: 0,
521: 1, 522: 1, 523: 0, 524: 0, 525: 0, 526: 0, 527: 0, 528: 0, 529: 1, 530: 0,
531: 0, 532: 2, 533: 0, 534: 0, 535: 0, 536: 1, 537: 0, 538: 0, 539: 0, 540: 0,
541: 0, 542: 0, 543: 0, 544: 0, 545: 0, 546: 1, 547: 2, 548: 1, 549: 0, 550: 0,
551: 1, 552: 1, 553: 0, 554: 0, 555: 0, 556: 0, 557: 1, 558: 0, 559: 1, 560: 0,
561: 2, 562: 1, 563: 1, 564: 1, 565: 2, 566: 1, 567: 1, 568: 1, 569: 1, 570: 0,
571: 1, 572: 0, 573: 1, 574: 2, 575: 0, 576: 0, 577: 0, 578: 1, 579: 2, 580: 1,
581: 1, 582: 0, 583: 0, 584: 0, 585: 1, 586: 0, 587: 1, 588: 1, 589: 0, 590: 0,
591: 0, 592: 0, 593: 0, 594: 0, 595: 0, 596: 1, 597: 1, 598: 0, 599: 3, 600: 1,
601: 1, 602: 1, 603: 3, 604: 2, 605: 2, 606: 0, 607: 1, 608: 1, 609: 1, 610: 1,
611: 2, 612: 0, 613: 2, 614: 2, 615: 3, 616: 3, 617: 2, 618: 3, 619: 1, 620: 1,
621: 0, 622: 2, 623: 2, 624: 0, 625: 0, 626: 0, 627: 0, 628: 2, 629: 3, 630: 1,
631: 1, 632: 1, 633: 0, 634: 0, 635: 0, 636: 1, 637: 1}
Grado máximo: 4
Tamaño de nodos calculado: [6.0, 6.0, 54.5, 54.5, 6.0, 54.5, 6.0, 54.5, 54.5,
54.5, 6.0, 54.5, 54.5, 6.0, 6.0, 6.0, 54.5, 6.0, 6.0, 54.5, 6.0, 6.0, 6.0, 54.5,
6.0, 54.5, 6.0, 54.5, 6.0, 6.0, 6.0, 6.0, 6.0, 6.0, 6.0, 54.5, 6.0, 6.0, 103.0,
6.0, 6.0, 54.5, 103.0, 54.5, 6.0, 54.5, 54.5, 6.0, 6.0, 54.5, 6.0, 6.0, 6.0,
54.5, 54.5, 6.0, 6.0, 6.0, 6.0, 6.0, 6.0, 6.0, 103.0, 54.5, 6.0, 6.0, 54.5, 6.0,
6.0, 103.0, 54.5, 54.5, 6.0, 54.5, 6.0, 6.0, 6.0, 6.0, 54.5, 6.0, 6.0, 54.5,
6.0, 54.5, 6.0, 6.0, 54.5, 6.0, 6.0, 54.5, 6.0, 6.0, 103.0, 6.0, 54.5, 6.0, 6.0,
54.5, 6.0, 6.0, 6.0, 54.5, 6.0, 6.0, 6.0, 54.5, 6.0, 6.0, 6.0, 6.0, 6.0, 54.5,
6.0, 54.5, 6.0, 6.0, 6.0, 54.5, 6.0, 54.5, 6.0, 54.5, 6.0, 54.5, 103.0, 6.0,
54.5, 54.5, 54.5, 6.0, 54.5, 6.0, 6.0, 6.0, 6.0, 54.5, 54.5, 54.5, 6.0, 6.0,
6.0, 6.0, 6.0, 6.0, 6.0, 54.5, 54.5, 54.5, 54.5, 6.0, 103.0, 6.0, 6.0, 54.5,
54.5, 54.5, 6.0, 54.5, 54.5, 6.0, 6.0, 6.0, 6.0, 54.5, 54.5, 6.0, 6.0, 6.0, 6.0,
6.0, 6.0, 6.0, 54.5, 54.5, 6.0, 6.0, 6.0, 54.5, 6.0, 54.5, 6.0, 6.0, 6.0,
6.0, 6.0, 6.0, 6.0, 6.0, 6.0, 6.0, 6.0, 6.0, 6.0, 6.0, 6.0, 6.0, 6.0, 6.0, 54.5,
6.0, 6.0, 54.5, 6.0, 6.0, 54.5, 6.0, 6.0, 6.0, 54.5, 54.5, 54.5, 6.0, 6.0, 6.0,
6.0, 6.0, 151.5, 54.5, 54.5, 54.5, 103.0, 6.0, 103.0, 6.0, 54.5, 200.0, 6.0,
6.0, 6.0, 103.0, 54.5, 6.0, 6.0, 6.0, 103.0, 6.0, 54.5, 6.0, 6.0, 6.0, 6.0, 6.0,
54.5, 6.0, 103.0, 6.0, 6.0, 6.0, 6.0, 54.5, 6.0, 6.0, 6.0, 6.0, 6.0, 54.5, 6.0,
54.5, 103.0, 103.0, 54.5, 54.5, 151.5, 54.5, 54.5, 54.5, 6.0, 6.0, 54.5, 6.0,
54.5, 6.0, 6.0, 54.5, 103.0, 6.0, 54.5, 6.0, 54.5, 54.5, 103.0, 54.5, 54.5,
54.5, 54.5, 6.0, 6.0, 6.0, 6.0, 103.0, 6.0, 103.0, 6.0, 54.5, 54.5, 103.0, 54.5,
54.5, 54.5, 6.0, 54.5, 54.5, 54.5, 103.0, 54.5, 6.0, 6.0, 54.5, 103.0, 6.0, 6.0,
6.0, 54.5, 6.0, 6.0, 6.0, 54.5, 54.5, 103.0, 103.0, 103.0, 6.0, 6.0, 6.0, 6.0,
6.0, 103.0, 54.5, 54.5, 103.0, 103.0, 54.5, 54.5, 6.0, 6.0, 54.5, 54.5, 6.0,
54.5, 54.5, 54.5, 54.5, 54.5, 103.0, 54.5, 6.0, 54.5, 54.5, 6.0, 54.5, 54.5,
103.0, 54.5, 6.0, 6.0, 103.0, 6.0, 103.0, 54.5, 6.0, 54.5, 54.5, 6.0, 54.5,
54.5, 6.0, 6.0, 6.0, 103.0, 6.0, 6.0, 54.5, 54.5, 54.5, 6.0, 6.0, 6.0, 6.0, 6.0,
54.5, 6.0, 6.0, 54.5, 6.0, 54.5, 54.5, 54.5, 54.5, 6.0, 6.0, 6.0, 6.0, 103.0,
54.5, 54.5, 6.0, 6.0, 54.5, 6.0, 54.5, 6.0, 54.5, 54.5, 151.5, 6.0, 54.5, 103.0,
103.0, 54.5, 54.5, 6.0, 54.5, 6.0, 54.5, 6.0, 103.0, 6.0, 54.5, 103.0, 54.5,
6.0, 6.0, 6.0, 6.0, 6.0, 6.0, 54.5, 103.0, 6.0, 54.5, 54.5, 6.0, 6.0, 6.0, 6.0,
54.5, 6.0, 6.0, 6.0, 6.0, 54.5, 6.0, 54.5, 6.0, 54.5, 54.5, 6.0, 54.5, 6.0,
```

```
54.5, 54.5, 6.0, 6.0, 6.0, 6.0, 103.0, 6.0, 54.5, 6.0, 103.0, 54.5, 6.0, 54.5,
6.0, 103.0, 54.5, 54.5, 6.0, 6.0, 6.0, 6.0, 54.5, 54.5, 6.0, 6.0, 6.0, 54.5,
6.0, 6.0, 6.0, 6.0, 151.5, 6.0, 6.0, 54.5, 6.0, 6.0, 6.0, 6.0, 6.0, 54.5, 54.5,
54.5, 54.5, 6.0, 6.0, 6.0, 6.0, 54.5, 54.5, 54.5, 54.5, 6.0, 54.5, 54.5, 54.5,
54.5, 6.0, 6.0, 54.5, 54.5, 54.5, 54.5, 54.5, 6.0, 6.0, 6.0, 6.0, 54.5, 6.0,
54.5, 54.5, 6.0, 6.0, 6.0, 6.0, 6.0, 6.0, 54.5, 6.0, 6.0, 103.0, 6.0, 6.0, 6.0,
54.5, 6.0, 6.0, 6.0, 6.0, 6.0, 6.0, 6.0, 6.0, 6.0, 54.5, 103.0, 54.5, 6.0, 6.0,
54.5, 54.5, 6.0, 6.0, 6.0, 6.0, 54.5, 6.0, 54.5, 6.0, 103.0, 54.5, 54.5, 54.5,
103.0, 54.5, 54.5, 54.5, 54.5, 6.0, 54.5, 6.0, 54.5, 103.0, 6.0, 6.0, 6.0, 54.5,
103.0, 54.5, 54.5, 6.0, 6.0, 6.0, 54.5, 6.0, 54.5, 54.5, 6.0, 6.0, 6.0, 6.0,
6.0, 6.0, 6.0, 54.5, 54.5, 6.0, 151.5, 54.5, 54.5, 54.5, 151.5, 103.0, 103.0,
6.0, 54.5, 54.5, 54.5, 54.5, 103.0, 6.0, 103.0, 103.0, 151.5, 151.5, 103.0,
151.5, 54.5, 54.5, 6.0, 103.0, 103.0, 6.0, 6.0, 6.0, 6.0, 103.0, 151.5, 54.5,
54.5, 54.5, 6.0, 6.0, 6.0, 54.5, 54.5]
Umbral (percentil 90): 2.0
Nodos hubs: [38, 42, 62, 69, 92, 124, 150, 217, 221, 223, 226, 230, 235, 245,
259, 260, 263, 275, 281, 290, 292, 296, 304, 309, 319, 320, 321, 327, 330, 331,
344, 352, 356, 358, 369, 393, 404, 407, 408, 416, 419, 428, 456, 460, 465, 482,
532, 547, 561, 565, 574, 579, 599, 603, 604, 605, 611, 613, 614, 615, 616, 617,
618, 622, 623, 628, 629]
Colores de los nodos: ['purple', 'purple', 'purple', 'purple', 'purple',
'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple',
'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple',
'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple',
'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple',
'purple', 'pink', 'purple', 'purple', 'purple', 'pink', 'purple', 'purple',
'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple',
'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple',
'purple', 'pink', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple',
'pink', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple',
'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple',
'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'pink',
'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple',
'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple',
'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple',
'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'pink',
'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple',
'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple',
'purple', 'pink', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple',
'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple',
'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple',
'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple',
'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple',
'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple',
'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple',
'purple', 'purple', 'purple', 'purple', 'pink', 'purple', 'purple', 'purple',
```

'pink', 'purple', 'pink', 'purple', 'purple', 'pink', 'purple', 'purple',
'purple', 'pink', 'purple', 'purple', 'purple', 'purple', 'pink', 'purple',
'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple',
'pink', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple',
'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'pink', 'pink',
'purple', 'purple', 'pink', 'purple', 'purple', 'purple', 'purple', 'purple',
'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'pink', 'purple',
'purple', 'purple', 'purple', 'purple', 'pink', 'purple', 'purple', 'purple',
'purple', 'purple', 'purple', 'purple', 'purple', 'pink', 'purple', 'pink',
'purple', 'purple', 'purple', 'pink', 'purple', 'purple', 'purple', 'purple',
'purple', 'purple', 'purple', 'pink', 'purple', 'purple', 'purple', 'purple',
'pink', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple',
'purple', 'purple', 'pink', 'pink', 'pink', 'purple', 'purple', 'purple',
'purple', 'purple', 'pink', 'purple', 'purple', 'pink', 'pink', 'purple',
'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple',
'purple', 'purple', 'purple', 'pink', 'purple', 'purple', 'purple', 'purple',
'purple', 'purple', 'purple', 'pink', 'purple', 'purple', 'purple', 'pink',
'purple', 'pink', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple',
'purple', 'purple', 'purple', 'purple', 'pink', 'purple', 'purple', 'purple',
'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple',
'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple',
'purple', 'purple', 'purple', 'purple', 'pink', 'purple', 'purple', 'purple',
'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'pink',
'purple', 'purple', 'pink', 'pink', 'purple', 'purple', 'purple', 'purple',
'purple', 'purple', 'purple', 'pink', 'purple', 'purple', 'pink', 'purple',
'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'pink',
'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple',
'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple',
'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple',
'purple', 'purple', 'purple', 'pink', 'purple', 'purple', 'purple', 'pink',
'purple', 'purple', 'purple', 'purple', 'pink', 'purple', 'purple', 'purple',
'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple',
'purple', 'purple', 'purple', 'purple', 'purple', 'pink', 'purple', 'purple',
'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple',
'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple',
'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple',
'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple',
'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'pink',
'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple',
'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'pink', 'purple',
'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple',
'purple', 'purple', 'purple', 'purple', 'pink', 'purple', 'purple', 'purple',
'pink', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple',
'purple', 'pink', 'purple', 'purple', 'purple', 'purple', 'pink', 'purple',
'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple',
'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple',
'purple', 'purple', 'pink', 'purple', 'purple', 'purple', 'pink', 'pink',

```
'pink', 'purple', 'purple', 'purple', 'purple', 'purple', 'pink', 'purple',
'pink', 'pink', 'pink', 'pink', 'pink', 'pink', 'purple', 'purple', 'purple',
'pink', 'pink', 'purple', 'purple', 'purple', 'purple', 'pink', 'pink',
'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple', 'purple']
```




[265]:
```
# EJERCICIO 4. En función de la matriz de emparejamiento (correlación de la
 ↪matriz de adyacencia),
# establecer una partición de los nodos en módulos. Escoger el número de
 ↪módulos que creas conveniente
# y justificar por qué escogiste ese número.
```

[269]:
```
!pip install python-louvain
```

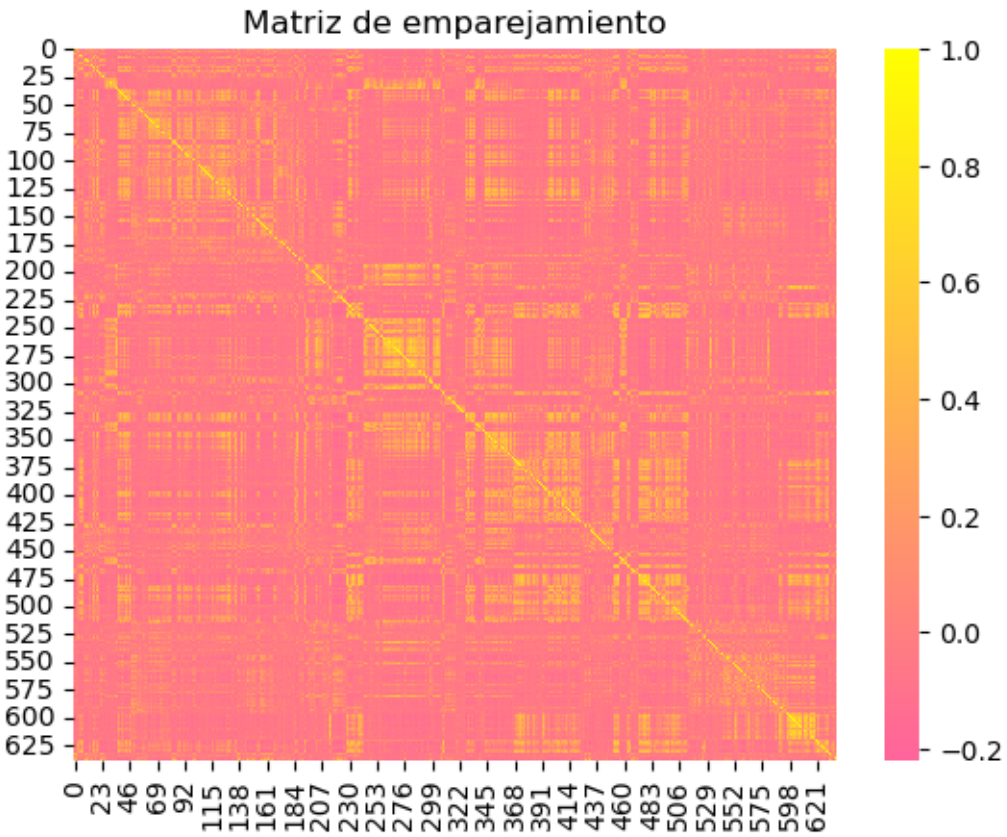
```
Collecting python-louvain
  Downloading python-louvain-0.16.tar.gz (204 kB)
     ---------------------------------------- 0.0/204.6 kB ? eta -:--:--
     ------ -------------------------------- 30.7/204.6 kB ? eta -:--:--
     ------ -------------------------------- 30.7/204.6 kB ? eta -:--:--
     ---------- ----------------------- 61.4/204.6 kB 409.6 kB/s eta 0:00:01
     ---------- ----------------------- 61.4/204.6 kB 409.6 kB/s eta 0:00:01
     ---------- ----------------------- 61.4/204.6 kB 409.6 kB/s eta 0:00:01
     ------------------------ ---------- 143.4/204.6 kB 568.9 kB/s eta 0:00:01
     ------------------------ ---------- 143.4/204.6 kB 568.9 kB/s eta 0:00:01
     ---------------------------------- - 194.6/204.6 kB 537.4 kB/s eta 0:00:01
     ---------------------------------- 204.6/204.6 kB 541.5 kB/s eta 0:00:00
  Preparing metadata (setup.py): started
  Preparing metadata (setup.py): finished with status 'done'
Requirement already satisfied: networkx in c:\users\melan\anaconda3\lib\site-
packages (from python-louvain) (3.2.1)
Requirement already satisfied: numpy in c:\users\melan\anaconda3\lib\site-
packages (from python-louvain) (1.26.4)
Building wheels for collected packages: python-louvain
  Building wheel for python-louvain (setup.py): started
  Building wheel for python-louvain (setup.py): finished with status 'done'
  Created wheel for python-louvain: filename=python_louvain-0.16-py3-none-
any.whl size=9403
sha256=a8639d153b6388d73ae189367923131a7b1fb30c82d85d9b1a79c16eaada9b04
  Stored in directory: c:\users\melan\appdata\local\pip\cache\wheels\40\f1\e3\48
5b698c520fa0baee1d07897abc7b8d6479b7d199ce96f4af
Successfully built python-louvain
Installing collected packages: python-louvain
Successfully installed python-louvain-0.16
```

[271]:
```python
import community as community_louvain
import seaborn as sns
```

[300]:
```python
def modulos(G, ej4):
    # Matriz de adyacencia
    matadj = nx.to_numpy_array(G)
    print("Matriz de adyacencia calculada")

    # Matriz de emparejamiento
    corrmat = np.corrcoef(matadj)
    print("Matriz de emparejamiento calculada")

    # Visualizar
    sns.heatmap(corrmat, cmap="spring", center=0)
    plt.title("Matriz de emparejamiento")
    plt.show()
```

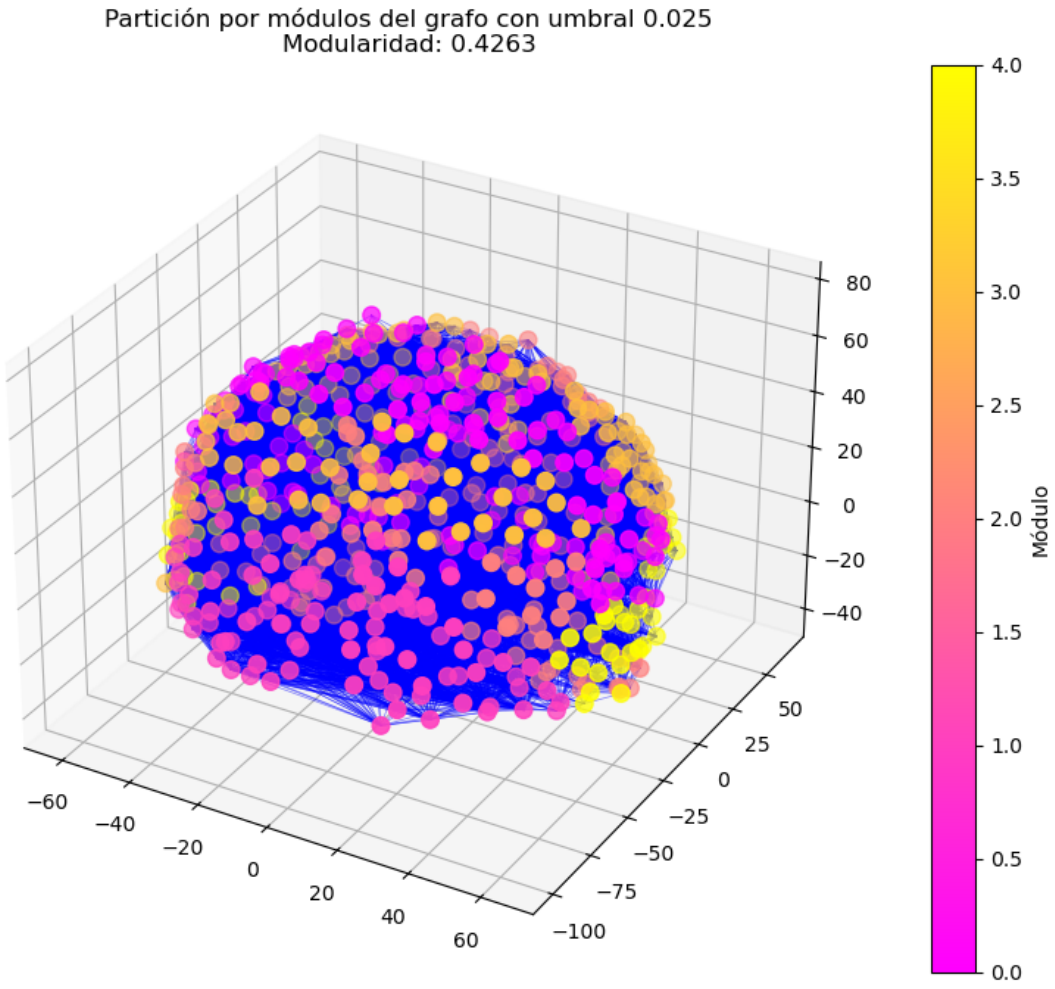
```
    # Partición por módulos usando el algoritmo de Louvain
    print("Calculando la partición de los nodos en módulos ")
    particion = community_louvain.best_partition(G)
    print("Partición calculada:", particion)

    # Color de nodos según su módulo
    nodcol = list(particion.values())

    # Cálculo de modularidad
    modularidad = community_louvain.modularity(particion, G)
    print(f"Modularidad: {modularidad:.4f}")

    # Figura en 3D
    fig = plt.figure(figsize=(12, 8))
    ax = fig.add_subplot(111, projection='3d')

    # Obtener posiciones de los nodos
    pos = nx.get_node_attributes(G, 'pos')
    if not pos:
        raise ValueError("El grafo no tiene posiciones 3D asignadas a los nodos.
 ↪")

    x, y, z = np.array(list(pos.values())).T

    # Graficar los nodos
    scatter = ax.scatter(x, y, z, c=nodcol, cmap="spring", s=70, label='Nodos')

    # Graficar las aristas
    for edge in G.edges():
        x_edge, y_edge, z_edge = zip(pos[edge[0]], pos[edge[1]])
        ax.plot(x_edge, y_edge, z_edge, c='b', alpha=0.5, linewidth=0.5)

    # Título y leyenda
    ax.set_title(f"{ej4}\nModularidad: {modularidad:.4f}")
    plt.colorbar(scatter, ax=ax, label="Módulo")
    plt.show()

# Grafo con umbral 0.1
grafo_025 = generate_graph(comat, 0.025, coordenadas)

# Partición por módulos y graficar
modulos(grafo_025, "Partición por módulos del grafo con umbral 0.025")
```

```
Matriz de adyacencia calculada
Matriz de emparejamiento calculada
```




```
Calculando la partición de los nodos en módulos
Partición calculada: {0: 2, 1: 2, 2: 3, 3: 3, 4: 2, 5: 0, 6: 0, 7: 3, 8: 2, 9:
2, 10: 0, 11: 2, 12: 0, 13: 2, 14: 2, 15: 0, 16: 3, 17: 2, 18: 0, 19: 3, 20: 0,
21: 2, 22: 2, 23: 2, 24: 2, 25: 1, 26: 1, 27: 1, 28: 1, 29: 1, 30: 1, 31: 1, 32:
1, 33: 1, 34: 1, 35: 1, 36: 1, 37: 0, 38: 3, 39: 0, 40: 4, 41: 0, 42: 3, 43: 0,
44: 3, 45: 3, 46: 0, 47: 4, 48: 4, 49: 4, 50: 4, 51: 4, 52: 2, 53: 2, 54: 2, 55:
4, 56: 2, 57: 3, 58: 2, 59: 4, 60: 4, 61: 4, 62: 3, 63: 4, 64: 4, 65: 4, 66: 3,
67: 4, 68: 4, 69: 3, 70: 3, 71: 4, 72: 4, 73: 3, 74: 4, 75: 3, 76: 3, 77: 3, 78:
4, 79: 3, 80: 3, 81: 2, 82: 2, 83: 2, 84: 2, 85: 2, 86: 2, 87: 3, 88: 3, 89: 3,
90: 2, 91: 3, 92: 3, 93: 3, 94: 3, 95: 3, 96: 3, 97: 3, 98: 3, 99: 2, 100: 3,
101: 2, 102: 3, 103: 3, 104: 3, 105: 2, 106: 3, 107: 3, 108: 2, 109: 3, 110: 3,
111: 3, 112: 3, 113: 3, 114: 3, 115: 3, 116: 3, 117: 2, 118: 3, 119: 3, 120: 3,
121: 3, 122: 3, 123: 3, 124: 3, 125: 3, 126: 3, 127: 3, 128: 3, 129: 3, 130: 3,
131: 3, 132: 2, 133: 3, 134: 2, 135: 3, 136: 2, 137: 2, 138: 2, 139: 0, 140: 3,
141: 2, 142: 2, 143: 3, 144: 2, 145: 3, 146: 2, 147: 2, 148: 2, 149: 2, 150: 2,
151: 2, 152: 3, 153: 2, 154: 3, 155: 2, 156: 2, 157: 2, 158: 2, 159: 2, 160: 3,
161: 2, 162: 2, 163: 2, 164: 2, 165: 2, 166: 2, 167: 3, 168: 2, 169: 3, 170: 2,
171: 2, 172: 2, 173: 2, 174: 3, 175: 2, 176: 2, 177: 2, 178: 3, 179: 2, 180: 2,
181: 3, 182: 2, 183: 2, 184: 3, 185: 0, 186: 3, 187: 2, 188: 2, 189: 2, 190: 2,
191: 2, 192: 3, 193: 0, 194: 1, 195: 1, 196: 1, 197: 1, 198: 2, 199: 2, 200: 2,
201: 1, 202: 1, 203: 1, 204: 1, 205: 2, 206: 1, 207: 1, 208: 1, 209: 2, 210: 2,
```

211: 1, 212: 1, 213: 0, 214: 0, 215: 1, 216: 2, 217: 2, 218: 2, 219: 3, 220: 2,
221: 2, 222: 2, 223: 2, 224: 2, 225: 2, 226: 2, 227: 2, 228: 0, 229: 0, 230: 3,
231: 3, 232: 3, 233: 0, 234: 0, 235: 3, 236: 3, 237: 0, 238: 0, 239: 0, 240: 0,
241: 0, 242: 1, 243: 1, 244: 1, 245: 1, 246: 1, 247: 1, 248: 1, 249: 1, 250: 1,
251: 1, 252: 1, 253: 1, 254: 1, 255: 1, 256: 1, 257: 1, 258: 1, 259: 1, 260: 1,
261: 1, 262: 1, 263: 1, 264: 1, 265: 1, 266: 1, 267: 1, 268: 1, 269: 1, 270: 1,
271: 1, 272: 1, 273: 1, 274: 1, 275: 1, 276: 2, 277: 1, 278: 1, 279: 1, 280: 1,
281: 1, 282: 1, 283: 1, 284: 1, 285: 1, 286: 1, 287: 2, 288: 1, 289: 1, 290: 1,
291: 3, 292: 1, 293: 3, 294: 2, 295: 2, 296: 3, 297: 2, 298: 3, 299: 2, 300: 1,
301: 1, 302: 1, 303: 3, 304: 1, 305: 1, 306: 2, 307: 2, 308: 0, 309: 0, 310: 2,
311: 2, 312: 2, 313: 2, 314: 1, 315: 2, 316: 2, 317: 2, 318: 2, 319: 0, 320: 0,
321: 0, 322: 0, 323: 0, 324: 0, 325: 0, 326: 0, 327: 3, 328: 3, 329: 0, 330: 3,
331: 3, 332: 3, 333: 3, 334: 3, 335: 1, 336: 1, 337: 1, 338: 1, 339: 1, 340: 1,
341: 1, 342: 1, 343: 1, 344: 3, 345: 3, 346: 3, 347: 3, 348: 3, 349: 3, 350: 3,
351: 3, 352: 3, 353: 1, 354: 0, 355: 0, 356: 3, 357: 3, 358: 3, 359: 3, 360: 3,
361: 3, 362: 3, 363: 0, 364: 3, 365: 3, 366: 0, 367: 0, 368: 0, 369: 0, 370: 0,
371: 0, 372: 0, 373: 0, 374: 0, 375: 0, 376: 0, 377: 0, 378: 0, 379: 0, 380: 0,
381: 0, 382: 0, 383: 0, 384: 0, 385: 0, 386: 0, 387: 0, 388: 0, 389: 0, 390: 0,
391: 0, 392: 0, 393: 0, 394: 0, 395: 0, 396: 0, 397: 3, 398: 3, 399: 0, 400: 3,
401: 3, 402: 0, 403: 3, 404: 0, 405: 0, 406: 0, 407: 3, 408: 0, 409: 0, 410: 0,
411: 0, 412: 0, 413: 0, 414: 0, 415: 0, 416: 3, 417: 0, 418: 3, 419: 0, 420: 0,
421: 3, 422: 0, 423: 0, 424: 0, 425: 2, 426: 0, 427: 2, 428: 2, 429: 0, 430: 3,
431: 2, 432: 3, 433: 3, 434: 3, 435: 2, 436: 3, 437: 0, 438: 2, 439: 0, 440: 3,
441: 0, 442: 2, 443: 2, 444: 3, 445: 2, 446: 2, 447: 3, 448: 2, 449: 2, 450: 3,
451: 2, 452: 0, 453: 0, 454: 0, 455: 0, 456: 1, 457: 1, 458: 1, 459: 1, 460: 1,
461: 1, 462: 0, 463: 0, 464: 0, 465: 0, 466: 2, 467: 2, 468: 2, 469: 2, 470: 2,
471: 2, 472: 0, 473: 0, 474: 0, 475: 0, 476: 0, 477: 0, 478: 0, 479: 0, 480: 0,
481: 0, 482: 3, 483: 3, 484: 3, 485: 3, 486: 0, 487: 0, 488: 0, 489: 0, 490: 0,
491: 3, 492: 3, 493: 3, 494: 3, 495: 0, 496: 0, 497: 0, 498: 0, 499: 2, 500: 0,
501: 0, 502: 0, 503: 0, 504: 0, 505: 0, 506: 0, 507: 0, 508: 2, 509: 3, 510: 0,
511: 0, 512: 2, 513: 3, 514: 4, 515: 4, 516: 2, 517: 4, 518: 2, 519: 2, 520: 4,
521: 4, 522: 2, 523: 4, 524: 2, 525: 0, 526: 2, 527: 0, 528: 0, 529: 2, 530: 2,
531: 2, 532: 1, 533: 2, 534: 4, 535: 2, 536: 1, 537: 1, 538: 4, 539: 1, 540: 2,
541: 4, 542: 2, 543: 4, 544: 4, 545: 2, 546: 4, 547: 2, 548: 4, 549: 4, 550: 4,
551: 1, 552: 4, 553: 4, 554: 4, 555: 4, 556: 2, 557: 4, 558: 4, 559: 2, 560: 2,
561: 2, 562: 4, 563: 4, 564: 4, 565: 4, 566: 2, 567: 2, 568: 4, 569: 2, 570: 4,
571: 4, 572: 4, 573: 4, 574: 2, 575: 2, 576: 1, 577: 4, 578: 4, 579: 2, 580: 1,
581: 4, 582: 2, 583: 2, 584: 4, 585: 2, 586: 2, 587: 4, 588: 2, 589: 2, 590: 0,
591: 4, 592: 2, 593: 4, 594: 4, 595: 2, 596: 4, 597: 4, 598: 0, 599: 0, 600: 4,
601: 4, 602: 4, 603: 4, 604: 0, 605: 4, 606: 4, 607: 0, 608: 0, 609: 2, 610: 0,
611: 4, 612: 0, 613: 4, 614: 0, 615: 0, 616: 4, 617: 4, 618: 4, 619: 4, 620: 0,
621: 0, 622: 0, 623: 0, 624: 0, 625: 0, 626: 0, 627: 0, 628: 0, 629: 0, 630: 0,
631: 0, 632: 2, 633: 3, 634: 2, 635: 0, 636: 2, 637: 2}
Modularidad: 0.4263



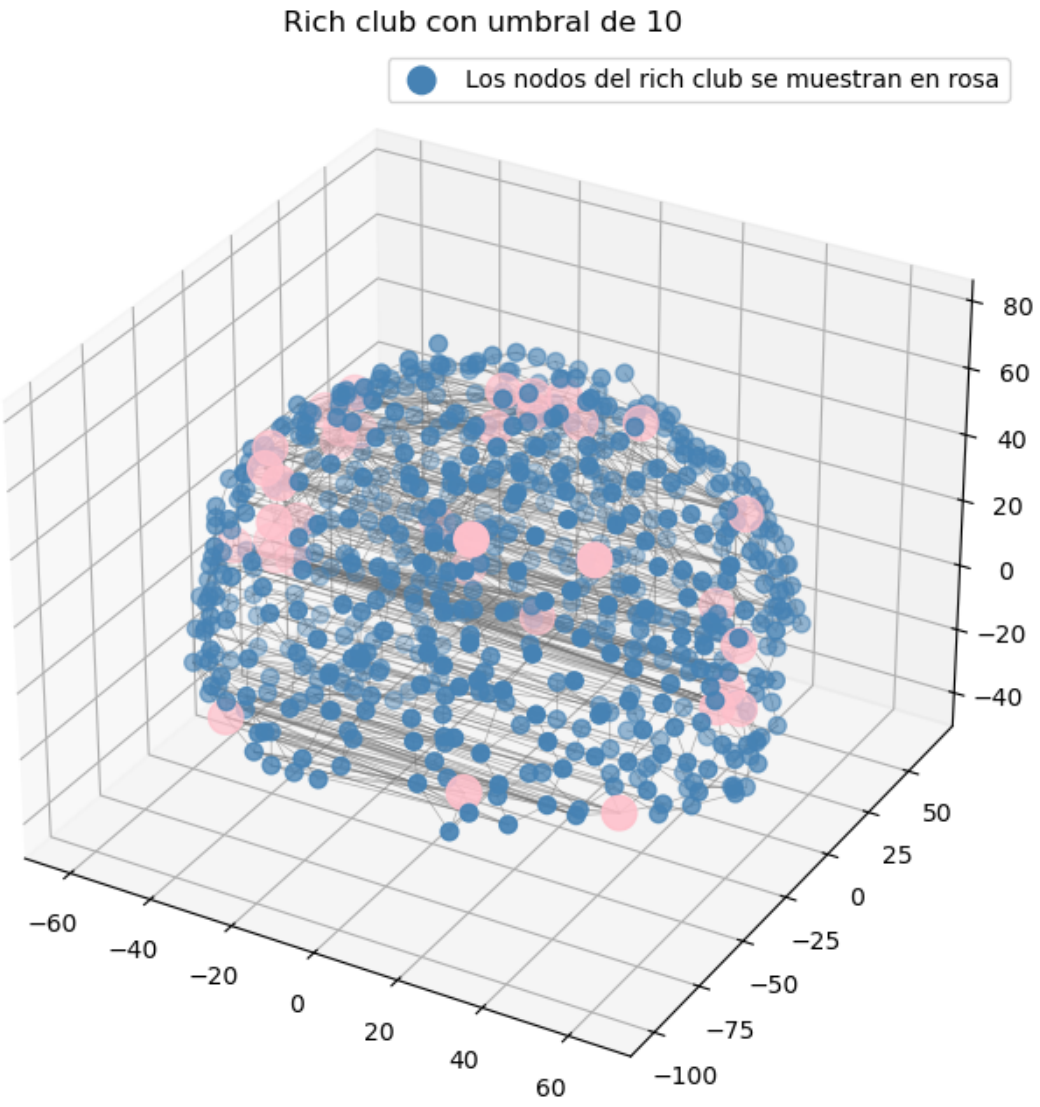

[296]:
```
#Mi elección se basa en el cálculo de la modularidad para checar agrupación de␣
↪los nodos dentro de cada módulo.
#Si aumentaba los módulos, la modularidad también aumentaba y no salía bien.
#Principalmente me basé en qué tamn bien se veía el grafo.
```

[302]:
```
# EJERCICIO 5
# Determinar el conjunto del Rich Club y discutir las implicaciones anatómicas y
# funcionales de este grupo de nodos (mínimo 100 palabras).
```

[324]:
```
def rich_club(G, ej5, min_degree=10):

    # grados de los nodos
    grados= dict(G.degree())

    #selección de nodos con grado >= min_degree
    rc_nodes= [node for node, degree in grados.items() if degree >= min_degree]
```

```
    print(f"Nodos con grado  {min_degree}): {rc_nodes}")


    if not rc_nodes:
        print("No hay nodos  ", min_degree)

    # posiciones de los nodos
    pos= nx.get_node_attributes(G, 'pos')
    x,y,z = np.array(list(pos.values())).T

    # colores y tamaños
    node_color = ['pink' if node in rc_nodes else 'steelblue' for node in G.
↪nodes()]
    node_sizes = [200 if node in rc_nodes else 50 for node in G.nodes()]

    # crear figura en 3d
    fig= plt.figure(figsize=(12,8))
    ax= fig.add_subplot(111, projection='3d')

    # graficar nodos
    ax.scatter(x, y, z, c=node_color, s=node_sizes, label='Nodos')

    # Graficar aristas
    for edge in G.edges():
        if edge[0] in pos and edge[1] in pos:
            x_edge, y_edge, z_edge = zip(pos[edge[0]], pos[edge[1]])
            ax.plot(x_edge, y_edge, z_edge, c='dimgrey', alpha=0.5, linewidth=0.
↪5)

    # Configurar título
    ax.set_title(ej5)
    plt.legend(["Los nodos del rich club se muestran en rosa"])
    plt.show()

# grafo
grafo01= generate_graph(comat, 0.1, coordenadas)

rich_club(grafo01, "Rich club con umbral de 10", min_degree=10)
#%%
```

```
Nodos con grado  10): [38, 70, 186, 230, 235, 253, 262, 328, 330, 334, 346,
356, 362, 397, 400, 407, 416, 454, 473, 477, 481, 482, 485, 488, 491, 494, 532,
599, 603, 606, 607, 616, 617, 619, 629]
```




[402]:
```
# Discusión
# El Rich Club es básicamente un grupo de nodos súper conectados, muy␣
 ↪importantes porque ayudan al cerebro a integrar toda la información que␣
 ↪recibe.
# en el cererbo son como lugarescomo la corteza prefrontal y otras áreas␣
 ↪importantes que están súper activas.
# Estos nodos facilitan la comunicación entre los diferentes módulos del␣
 ↪cerebro, como centros de control que conectan todo.
# Hablando de la parte funcional, el Rich Club es importante para cosas␣
 ↪complejas cotidianas:
# tomar decisiones, recordar cosas importantes o concentrarnos en algo.
```

```
# Sin estos nodos súper conectados, nuestro cerebro no sería tan bueno␣
 ↪manejando tareas complicadas.
```

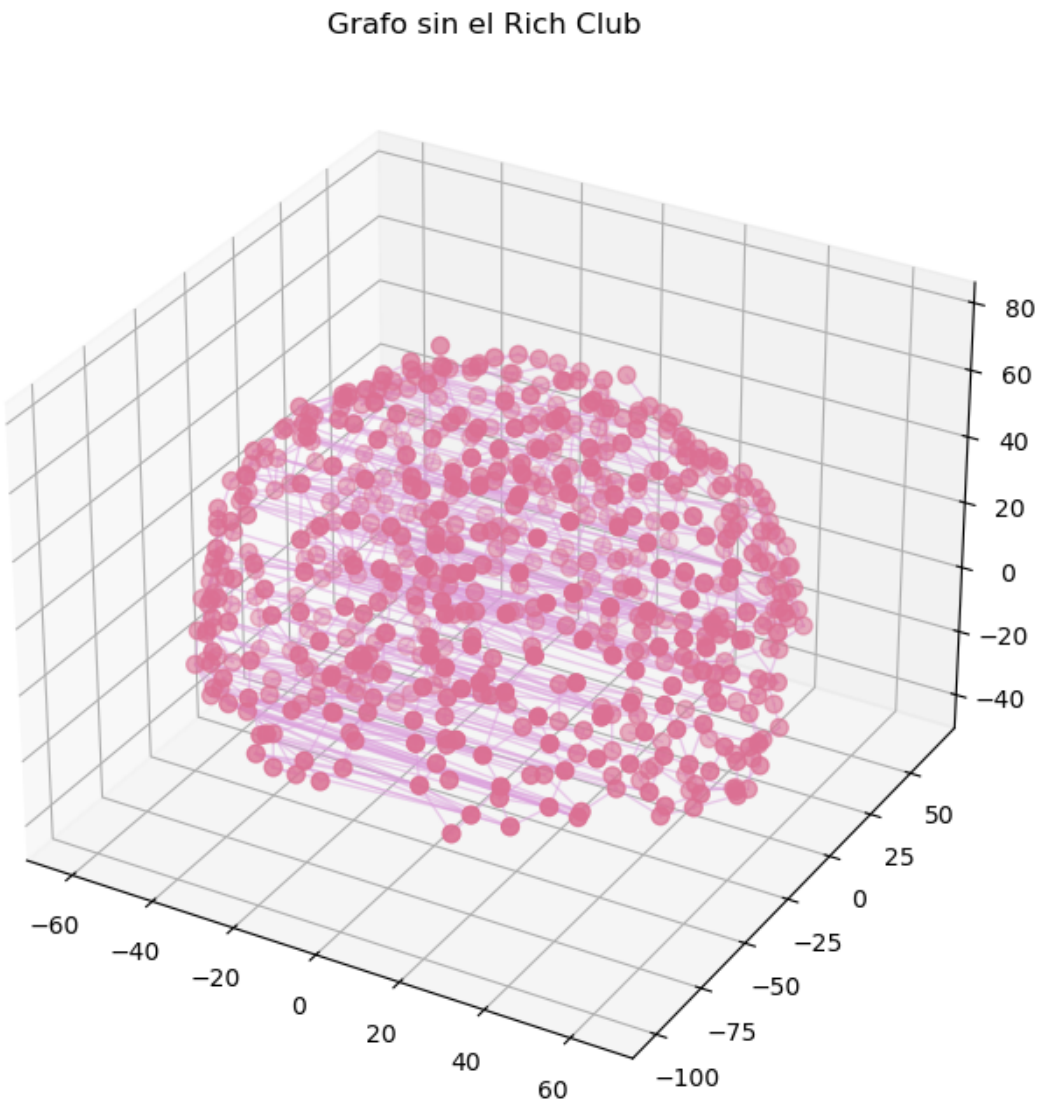
[ ]:

[405]:
```
# EJERCICIO 6
# Supongamos que eliminamos los nodos del RichClub, describir cómo cambian las
# propiedades topológicas del grafo, hacer comparativas del grado, coeficiente␣
 ↪de
# cluster, coeficiente de mundo pequeño y las medidas de centralidad
# (cercanía,intermediación)
```

[407]:
```
def propiedades_grafo(G):
    propiedades = {}

    # Average grade
    avg_grad = sum(dict(G.degree()).values()) / G.number_of_nodes()
    propiedades['grado_medio'] = avg_grad

    # Coeficiente de cluster
    coef_cluster = nx.average_clustering(G)
    propiedades['coeficiente_de_agrupamiento'] = coef_cluster

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

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

    return propiedades

def comparacion_propiedades(datos_originales, datos_modificados):
    """
    Compara las propiedades del grafo original y el modificado.
    """
    print("\nComparación de propiedades:")
    for prop in datos_originales.keys():
        original = datos_originales[prop]
        modificado = datos_modificados[prop]
        cambio = ((modificado - original) / original) * 100 if original != 0␣
 ↪else "N/A"
        print(f"{prop.capitalize()}:")
        print(f"  Original: {original:.4f}")
        print(f"  Modificado: {modificado:.4f}")
```

```
        print(f"  Cambio: {cambio if cambio == 'N/A' else cambio:.2f}%\n")

# hacer grafo original
grafo_original = generate_graph(comat, 0.1, coordenadas)
grados = dict(grafo_original.degree())

# calcular las propiedades iniciales
print("Calcular las propiedades del grafo original:")
datos_originales = propiedades_grafo(grafo_original)
print(datos_originales)

# selección de nodos del Rich Club
min_degree = 10
rc_nodes = [node for node, degree in grados.items() if degree >= min_degree]
print(f"Nodos del Rich Club (grado  {min_degree}): {rc_nodes}")

# copiar grafo y eliminar nodos del Rich Club
grafomodificado = grafo_original.copy()
grafomodificado.remove_nodes_from(rc_nodes)

# calcular las propiedades del grafo modificado
print("\nCalculando las propiedades del grafo sin el Rich Club:")
datos_modificados = propiedades_grafo(grafomodificado)
print(datos_modificados)

# comparar propiedades
comparacion_propiedades(datos_originales, datos_modificados)

# grafo modificado sin el Rich Club
def plot_grafo(G, titulo):
    pos = nx.get_node_attributes(G, 'pos')
    x, y, z = np.array(list(pos.values())).T

    fig = plt.figure(figsize=(12, 8))
    ax = fig.add_subplot(111, projection='3d')

    # graficar nodos y aristas
    scatter = ax.scatter(x, y, z, c='palevioletred', s=50)
    for edge in G.edges():
        x_edge, y_edge, z_edge = zip(pos[edge[0]], pos[edge[1]])
        ax.plot(x_edge, y_edge, z_edge, c='plum', alpha=0.5, linewidth=1)

    ax.set_title(titulo)
    plt.show()

plot_grafo(grafomodificado, "Grafo sin el Rich Club")
#%%
```

```
Calcular las propiedades del grafo original:
{'grado_medio': 4.529780564263323, 'coeficiente_de_agrupamiento':
0.24364464673564803, 'centralidad_cercania': 0.1194031264903796,
'centralidad_intermediacion': 0.01147712930871217}
Nodos del Rich Club (grado  10): [38, 70, 186, 230, 235, 253, 262, 328, 330,
334, 346, 356, 362, 397, 400, 407, 416, 454, 473, 477, 481, 482, 485, 488, 491,
494, 532, 599, 603, 606, 607, 616, 617, 619, 629]

Calculando las propiedades del grafo sin el Rich Club:
{'grado_medio': 3.691542288557214, 'coeficiente_de_agrupamiento':
0.22183658427439523, 'centralidad_cercania': 0.09214115900724197,
'centralidad_intermediacion': 0.016110568269096143}

Comparación de propiedades:
Grado_medio:
  Original: 4.5298
  Modificado: 3.6915
  Cambio: -18.51%

Coeficiente_de_agrupamiento:
  Original: 0.2436
  Modificado: 0.2218
  Cambio: -8.95%

Centralidad_cercania:
  Original: 0.1194
  Modificado: 0.0921
  Cambio: -22.83%

Centralidad_intermediacion:
  Original: 0.0115
  Modificado: 0.0161
  Cambio: 40.37%
```




[408]:
```
#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 centralidad
# (cercanía, intermediación)
```

[409]:
```
import networkx as nx
import numpy as np
import matplotlib.pyplot as plt

def propiedades_grafo(G):
```

```
    propiedades = {}

    # Average grade
    avg_grade = sum(dict(G.degree()).values()) / G.number_of_nodes()
    propiedades['grado_medio'] = avg_grade

    # Coeficiente de agrupamiento
    coef_cluster = nx.average_clustering(G)
    propiedades['coeficiente_de_agrupamiento'] = coef_cluster

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

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

    return propiedades

def comparacion_propiedades(datos_originales, datos_modificados):
    """
    Compara las propiedades del grafo original y el modificado.
    """
    print("\nComparación de propiedades:")
    for prop in datos_originales.keys():
        original = datos_originales[prop]
        modificado = datos_modificados[prop]
        cambio = ((modificado - original) / original) * 100 if original != 0␣
 ↪else "N/A"
        print(f"{prop.capitalize()}:")
        print(f"  Original: {original:.4f}")
        print(f"  Modificado: {modificado:.4f}")
        print(f"  Cambio: {cambio if cambio == 'N/A' else cambio:.2f}%\n")

# Crear función que elimine un porcentaje de nodos
def eliminar_porcentaje(G, percentaje=0.1):
    """
    Elimina un porcentaje de los nodos con mayor centralidad de intermediación.

    Args:
        G (networkx.Graph): Grafo original.
        percentaje (float): Porcentaje de nodos a eliminar (entre 0 y 1).

    Returns:
        grafo_modificado (networkx.Graph): Grafo con los nodos eliminados.
```

```
    """
    # Centralidad de intermediación
    betweenness = nx.betweenness_centrality(G)

    # Ordenar los nodos por centralidad de intermediación de mayor a menor
    sorted_nodes = sorted(betweenness.items(), key=lambda x: x[1], reverse=True)

    # Calcular el número de nodos a eliminar
    num_nodos_eliminar = int(len(sorted_nodes) * percentaje)
    nodos_quitar = [node for node, _ in sorted_nodes[:num_nodos_eliminar]]

    # Crear una copia del grafo y eliminar los nodos
    grafo_modificado = G.copy()
    grafo_modificado.remove_nodes_from(nodos_quitar)

    print(f"Eliminando el {percentaje*100}% de los nodos con mayor␣
 ↪intermediación: {nodos_quitar}")

    return grafo_modificado

# grafo original
grafo_original = generate_graph(comat, 0.1, coordenadas)

# calcular propiedades del grafo original
print("Calculo de las propiedades del grafo original: ")
datos_originales = propiedades_grafo(grafo_original)
print(datos_originales)

# eliminar diferentes porcentajes de nodos con más centralidad
percentajes = [0.1, 0.2, 0.3, 0.4, 0.5]
grafo_modificado1 = eliminar_porcentaje(grafo_original,␣
 ↪percentaje=percentajes[0])
grafo_modificado2 = eliminar_porcentaje(grafo_original,␣
 ↪percentaje=percentajes[1])
grafo_modificado3 = eliminar_porcentaje(grafo_original,␣
 ↪percentaje=percentajes[2])
grafo_modificado4 = eliminar_porcentaje(grafo_original,␣
 ↪percentaje=percentajes[3])
grafo_modificado5 = eliminar_porcentaje(grafo_original,␣
 ↪percentaje=percentajes[4])

# calcular las propiedades de los nuevos grafos
print("\nCalculando propiedades del grafo sin el 10%:")
datos_modificados_10 = propiedades_grafo(grafo_modificado1)
print(datos_modificados_10)
```

```python
print("\nCalculando propiedades del grafo sin el 20%:")
datos_modificados_20 = propiedades_grafo(grafo_modificado2)
print(datos_modificados_20)

print("\nCalculando propiedades del grafo sin el 30%:")
datos_modificados_30 = propiedades_grafo(grafo_modificado3)
print(datos_modificados_30)

print("\nCalculando propiedades del grafo sin el 40%:")
datos_modificados_40 = propiedades_grafo(grafo_modificado4)
print(datos_modificados_40)

print("\nCalculando propiedades del grafo sin el 50%:")
datos_modificados_50 = propiedades_grafo(grafo_modificado5)
print(datos_modificados_50)

# comparación de propiedades
comparacion_propiedades(datos_originales, datos_modificados_10)
comparacion_propiedades(datos_originales, datos_modificados_20)
comparacion_propiedades(datos_originales, datos_modificados_30)
comparacion_propiedades(datos_originales, datos_modificados_40)
comparacion_propiedades(datos_originales, datos_modificados_50)

# visualización de los grafos modificados
def plot_grafo(G, titulo):
    pos = nx.get_node_attributes(G, 'pos')
    x, y, z = np.array(list(pos.values())).T

    fig = plt.figure(figsize=(12, 8))
    ax = fig.add_subplot(111, projection='3d')

    # nodos y aristas
    scatter = ax.scatter(x, y, z, c='maroon', s=50)
    for edge in G.edges():
        x_edge, y_edge, z_edge = zip(pos[edge[0]], pos[edge[1]])
        ax.plot(x_edge, y_edge, z_edge, c='firebrick', alpha=0.5, linewidth=0.5)

    ax.set_title(titulo)
    plt.show()

# grafos
plot_grafo(grafo_modificado1, "Sin 10% de nodos")
plot_grafo(grafo_modificado2, "Sin 20% de nodos")
plot_grafo(grafo_modificado3, "Sin el 30% de nodos")
plot_grafo(grafo_modificado4, "Sin el 40% de nodos")
plot_grafo(grafo_modificado5, "Sin el 50% de nodos")
```

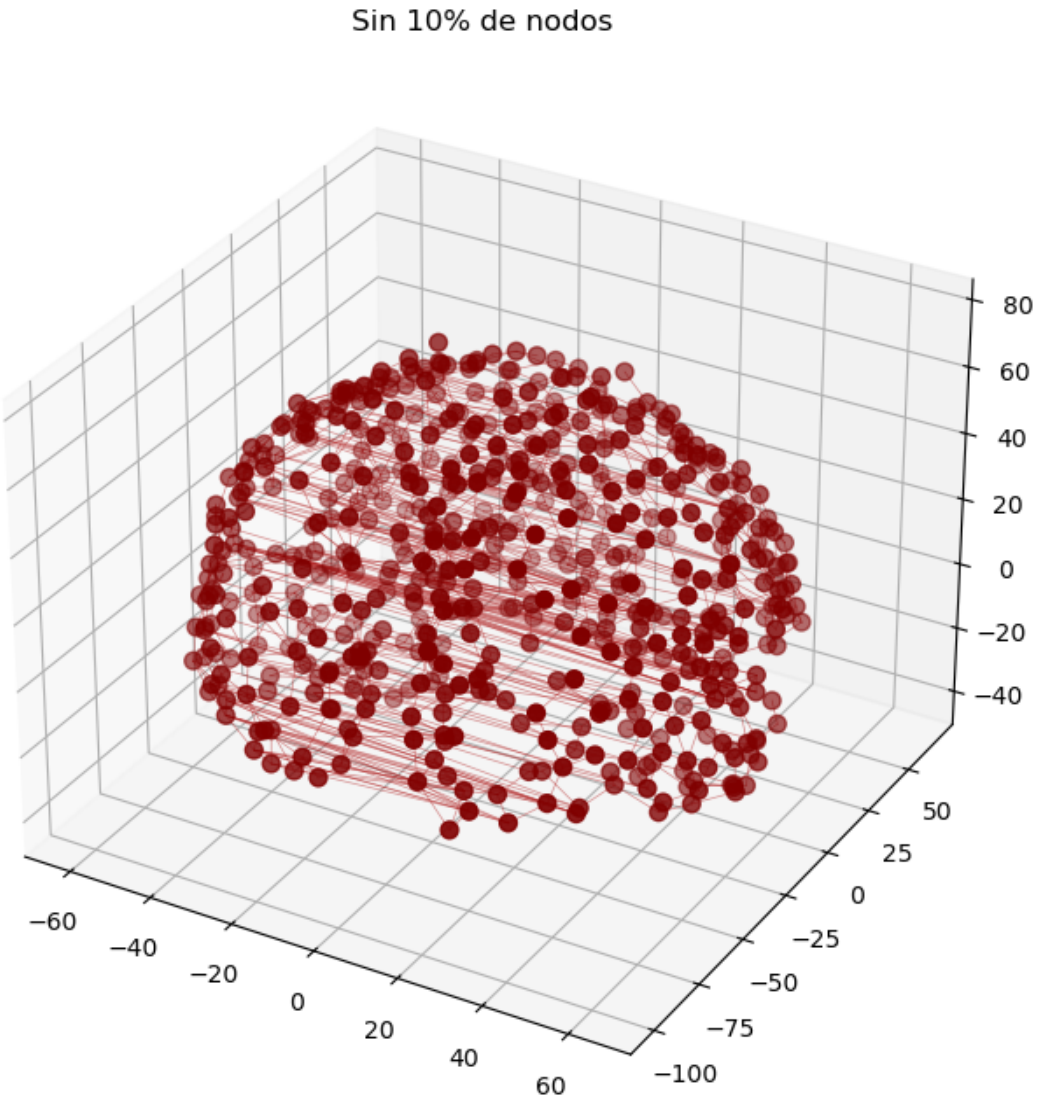
```
Calculo de las propiedades del grafo original:
{'grado_medio': 4.529780564263323, 'coeficiente_de_agrupamiento':
0.24364464673564803, 'centralidad_cercania': 0.1194031264903796,
'centralidad_intermediacion': 0.01147712930871217}
Eliminando el 10.0% de los nodos con mayor intermediación: [334, 277, 276, 235,
400, 272, 353, 121, 38, 482, 230, 154, 284, 160, 41, 196, 532, 286, 629, 473,
159, 202, 598, 485, 330, 356, 195, 367, 271, 480, 494, 50, 454, 281, 488, 312,
220, 619, 344, 128, 427, 618, 565, 16, 153, 315, 405, 2, 186, 285, 303, 599, 3,
237, 0, 318, 346, 481, 431, 597, 434, 432, 302]
Eliminando el 20.0% de los nodos con mayor intermediación: [334, 277, 276, 235,
400, 272, 353, 121, 38, 482, 230, 154, 284, 160, 41, 196, 532, 286, 629, 473,
159, 202, 598, 485, 330, 356, 195, 367, 271, 480, 494, 50, 454, 281, 488, 312,
220, 619, 344, 128, 427, 618, 565, 16, 153, 315, 405, 2, 186, 285, 303, 599, 3,
237, 0, 318, 346, 481, 431, 597, 434, 432, 302, 496, 584, 97, 491, 69, 275, 580,
591, 65, 223, 428, 292, 345, 549, 262, 605, 232, 93, 55, 226, 57, 100, 261, 546,
104, 553, 268, 289, 339, 579, 562, 67, 328, 362, 105, 501, 375, 331, 607, 613,
7, 120, 430, 582, 596, 513, 522, 540, 116, 70, 211, 361, 351, 587, 149, 329,
516, 512, 204, 36, 416, 18, 48, 132]
Eliminando el 30.0% de los nodos con mayor intermediación: [334, 277, 276, 235,
400, 272, 353, 121, 38, 482, 230, 154, 284, 160, 41, 196, 532, 286, 629, 473,
159, 202, 598, 485, 330, 356, 195, 367, 271, 480, 494, 50, 454, 281, 488, 312,
220, 619, 344, 128, 427, 618, 565, 16, 153, 315, 405, 2, 186, 285, 303, 599, 3,
237, 0, 318, 346, 481, 431, 597, 434, 432, 302, 496, 584, 97, 491, 69, 275, 580,
591, 65, 223, 428, 292, 345, 549, 262, 605, 232, 93, 55, 226, 57, 100, 261, 546,
104, 553, 268, 289, 339, 579, 562, 67, 328, 362, 105, 501, 375, 331, 607, 613,
7, 120, 430, 582, 596, 513, 522, 540, 116, 70, 211, 361, 351, 587, 149, 329,
516, 512, 204, 36, 416, 18, 48, 132, 603, 250, 327, 612, 365, 531, 407, 28, 4,
62, 418, 199, 526, 503, 5, 34, 500, 61, 398, 423, 359, 436, 410, 135, 414, 539,
87, 257, 563, 560, 348, 425, 368, 299, 600, 267, 99, 556, 514, 123, 193, 297,
201, 604, 442, 495, 113, 263, 98, 111, 17, 306, 557, 564, 542, 611, 122, 350,
219, 433, 304, 459, 134, 60]
Eliminando el 40.0% de los nodos con mayor intermediación: [334, 277, 276, 235,
400, 272, 353, 121, 38, 482, 230, 154, 284, 160, 41, 196, 532, 286, 629, 473,
159, 202, 598, 485, 330, 356, 195, 367, 271, 480, 494, 50, 454, 281, 488, 312,
220, 619, 344, 128, 427, 618, 565, 16, 153, 315, 405, 2, 186, 285, 303, 599, 3,
237, 0, 318, 346, 481, 431, 597, 434, 432, 302, 496, 584, 97, 491, 69, 275, 580,
591, 65, 223, 428, 292, 345, 549, 262, 605, 232, 93, 55, 226, 57, 100, 261, 546,
104, 553, 268, 289, 339, 579, 562, 67, 328, 362, 105, 501, 375, 331, 607, 613,
7, 120, 430, 582, 596, 513, 522, 540, 116, 70, 211, 361, 351, 587, 149, 329,
516, 512, 204, 36, 416, 18, 48, 132, 603, 250, 327, 612, 365, 531, 407, 28, 4,
62, 418, 199, 526, 503, 5, 34, 500, 61, 398, 423, 359, 436, 410, 135, 414, 539,
87, 257, 563, 560, 348, 425, 368, 299, 600, 267, 99, 556, 514, 123, 193, 297,
201, 604, 442, 495, 113, 263, 98, 111, 17, 306, 557, 564, 542, 611, 122, 350,
219, 433, 304, 459, 134, 60, 439, 487, 165, 602, 417, 517, 412, 363, 253, 538,
369, 551, 561, 550, 158, 461, 589, 37, 187, 238, 381, 110, 221, 566, 637, 206,
397, 468, 352, 366, 84, 71, 15, 118, 371, 541, 66, 126, 6, 91, 574, 534, 222,
544, 585, 548, 590, 499, 457, 581, 320, 570, 22, 537, 373, 383, 103, 475, 171,
212, 124, 44, 606, 31]
```

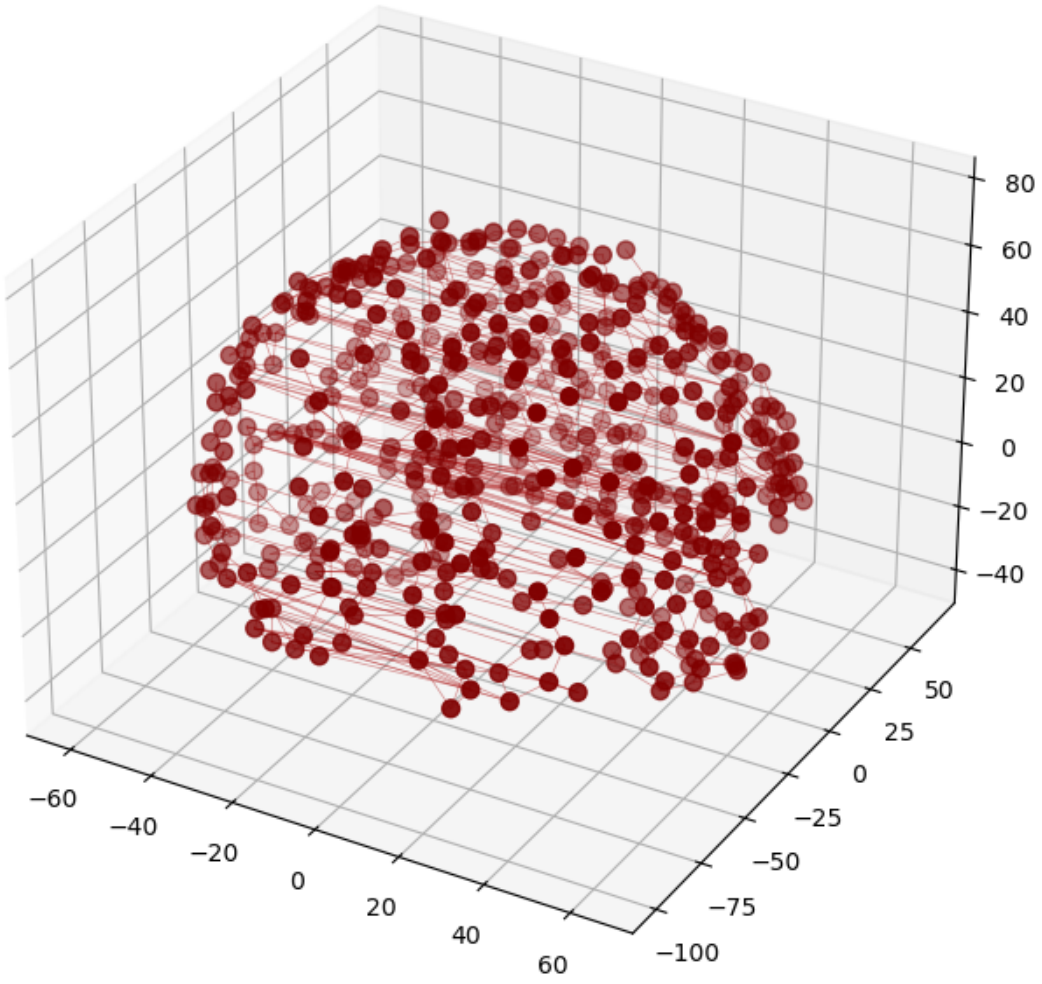
```
Eliminando el 50.0% de los nodos con mayor intermediación: [334, 277, 276, 235,
400, 272, 353, 121, 38, 482, 230, 154, 284, 160, 41, 196, 532, 286, 629, 473,
159, 202, 598, 485, 330, 356, 195, 367, 271, 480, 494, 50, 454, 281, 488, 312,
220, 619, 344, 128, 427, 618, 565, 16, 153, 315, 405, 2, 186, 285, 303, 599, 3,
237, 0, 318, 346, 481, 431, 597, 434, 432, 302, 496, 584, 97, 491, 69, 275, 580,
591, 65, 223, 428, 292, 345, 549, 262, 605, 232, 93, 55, 226, 57, 100, 261, 546,
104, 553, 268, 289, 339, 579, 562, 67, 328, 362, 105, 501, 375, 331, 607, 613,
7, 120, 430, 582, 596, 513, 522, 540, 116, 70, 211, 361, 351, 587, 149, 329,
516, 512, 204, 36, 416, 18, 48, 132, 603, 250, 327, 612, 365, 531, 407, 28, 4,
62, 418, 199, 526, 503, 5, 34, 500, 61, 398, 423, 359, 436, 410, 135, 414, 539,
87, 257, 563, 560, 348, 425, 368, 299, 600, 267, 99, 556, 514, 123, 193, 297,
201, 604, 442, 495, 113, 263, 98, 111, 17, 306, 557, 564, 542, 611, 122, 350,
219, 433, 304, 459, 134, 60, 439, 487, 165, 602, 417, 517, 412, 363, 253, 538,
369, 551, 561, 550, 158, 461, 589, 37, 187, 238, 381, 110, 221, 566, 637, 206,
397, 468, 352, 366, 84, 71, 15, 118, 371, 541, 66, 126, 6, 91, 574, 534, 222,
544, 585, 548, 590, 499, 457, 581, 320, 570, 22, 537, 373, 383, 103, 475, 171,
212, 124, 44, 606, 31, 478, 136, 622, 536, 323, 529, 78, 456, 357, 502, 279,
380, 32, 9, 81, 552, 23, 11, 528, 627, 52, 8, 445, 403, 354, 470, 29, 441, 573,
194, 296, 477, 21, 452, 378, 213, 107, 40, 112, 117, 137, 214, 244, 387, 133,
314, 571, 506, 35, 190, 493, 88, 394, 188, 558, 358, 176, 448, 588, 625, 472,
399, 489, 614]

Calculando propiedades del grafo sin el 10%:
{'grado_medio': 3.6278260869565218, 'coeficiente_de_agrupamiento':
0.23032754684928597, 'centralidad_cercania': 0.08004376356795163,
'centralidad_intermediacion': 0.018409194439575366}

Calculando propiedades del grafo sin el 20%:
{'grado_medio': 3.095890410958904, 'coeficiente_de_agrupamiento':
0.20173950858882367, 'centralidad_cercania': 0.04998781383427804,
'centralidad_intermediacion': 0.01913867035093501}

Calculando propiedades del grafo sin el 30%:
{'grado_medio': 2.7024608501118568, 'coeficiente_de_agrupamiento':
0.1928305102801747, 'centralidad_cercania': 0.033719447568480526,
'centralidad_intermediacion': 0.025198856261586827}

Calculando propiedades del grafo sin el 40%:
{'grado_medio': 2.2715404699738904, 'coeficiente_de_agrupamiento':
0.17272162128558996, 'centralidad_cercania': 0.016589724250986537,
'centralidad_intermediacion': 0.002754698176363759}

Calculando propiedades del grafo sin el 50%:
{'grado_medio': 1.786833855799373, 'coeficiente_de_agrupamiento':
0.13396029258098224, 'centralidad_cercania': 0.010798088556210773,
'centralidad_intermediacion': 0.000696766507093889}

Comparación de propiedades:
```

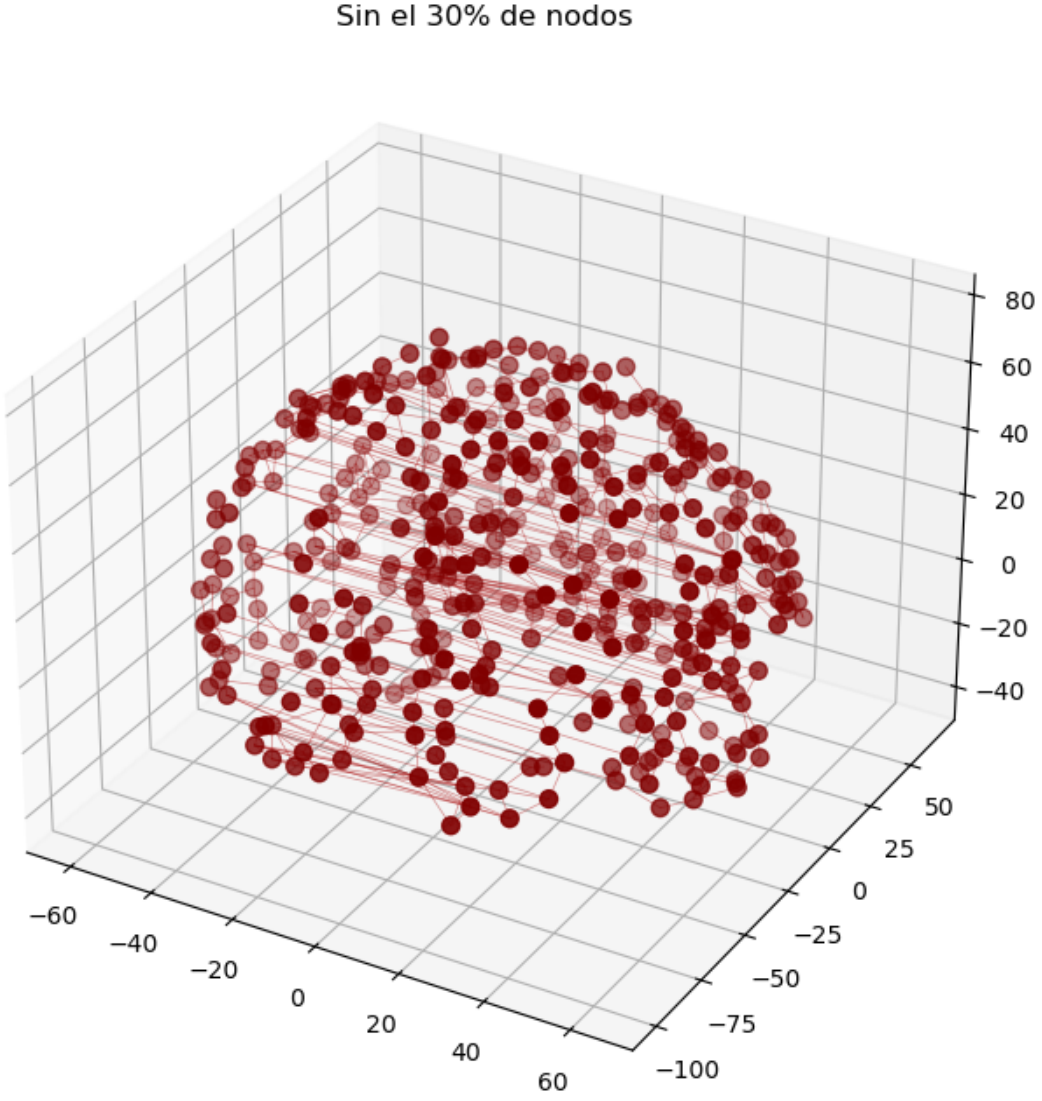
```
Grado_medio:
  Original: 4.5298
  Modificado: 3.6278
  Cambio: -19.91%

Coeficiente_de_agrupamiento:
  Original: 0.2436
  Modificado: 0.2303
  Cambio: -5.47%

Centralidad_cercania:
  Original: 0.1194
  Modificado: 0.0800
  Cambio: -32.96%

Centralidad_intermediacion:
  Original: 0.0115
  Modificado: 0.0184
  Cambio: 60.40%


Comparación de propiedades:
Grado_medio:
  Original: 4.5298
  Modificado: 3.0959
  Cambio: -31.65%

Coeficiente_de_agrupamiento:
  Original: 0.2436
  Modificado: 0.2017
  Cambio: -17.20%

Centralidad_cercania:
  Original: 0.1194
  Modificado: 0.0500
  Cambio: -58.14%

Centralidad_intermediacion:
  Original: 0.0115
  Modificado: 0.0191
  Cambio: 66.75%


Comparación de propiedades:
Grado_medio:
  Original: 4.5298
  Modificado: 2.7025
  Cambio: -40.34%
```

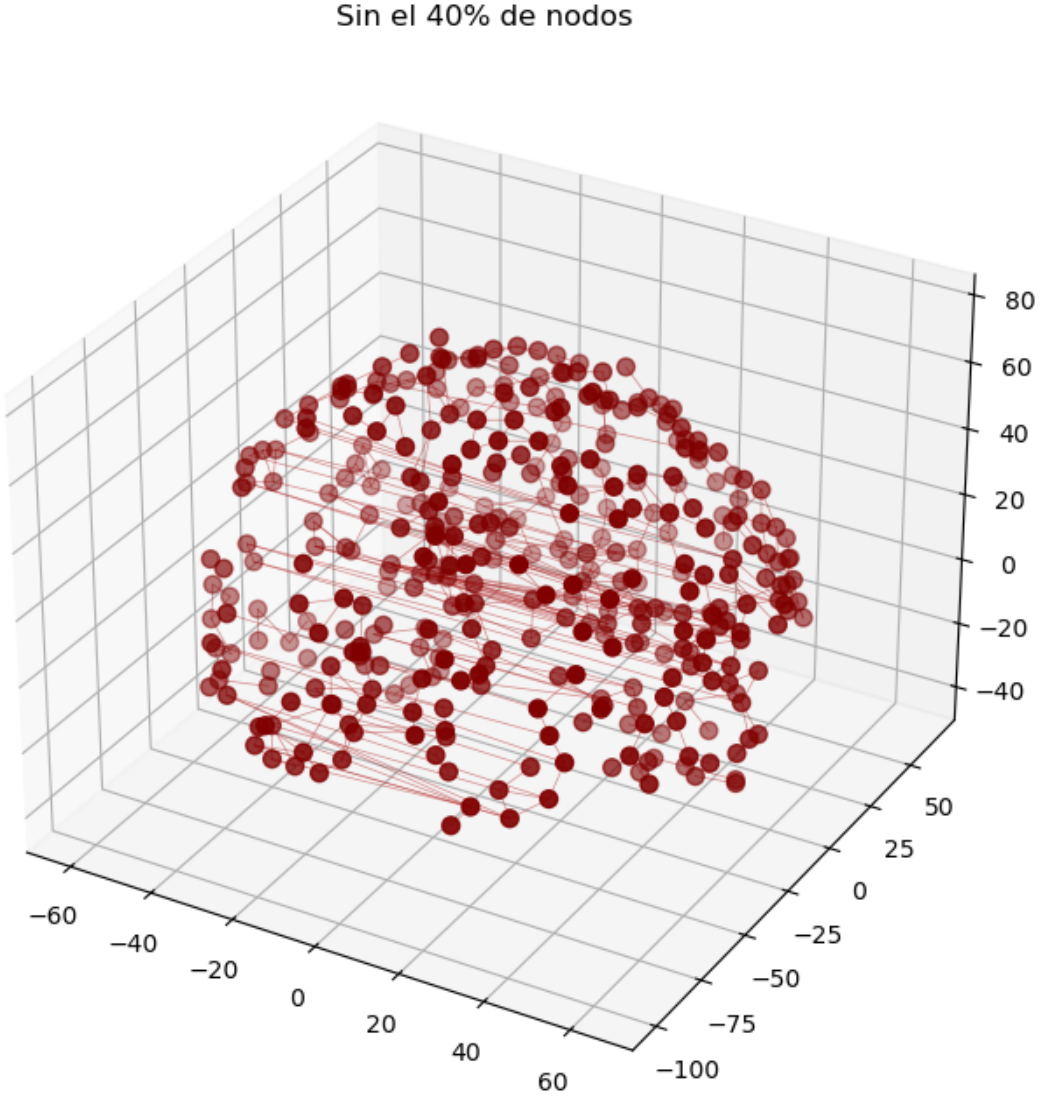
```
Coeficiente_de_agrupamiento:
  Original: 0.2436
  Modificado: 0.1928
  Cambio: -20.86%

Centralidad_cercania:
  Original: 0.1194
  Modificado: 0.0337
  Cambio: -71.76%

Centralidad_intermediacion:
  Original: 0.0115
  Modificado: 0.0252
  Cambio: 119.56%


Comparación de propiedades:
Grado_medio:
  Original: 4.5298
  Modificado: 2.2715
  Cambio: -49.85%

Coeficiente_de_agrupamiento:
  Original: 0.2436
  Modificado: 0.1727
  Cambio: -29.11%

Centralidad_cercania:
  Original: 0.1194
  Modificado: 0.0166
  Cambio: -86.11%

Centralidad_intermediacion:
  Original: 0.0115
  Modificado: 0.0028
  Cambio: -76.00%


Comparación de propiedades:
Grado_medio:
  Original: 4.5298
  Modificado: 1.7868
  Cambio: -60.55%

Coeficiente_de_agrupamiento:
  Original: 0.2436
  Modificado: 0.1340
```

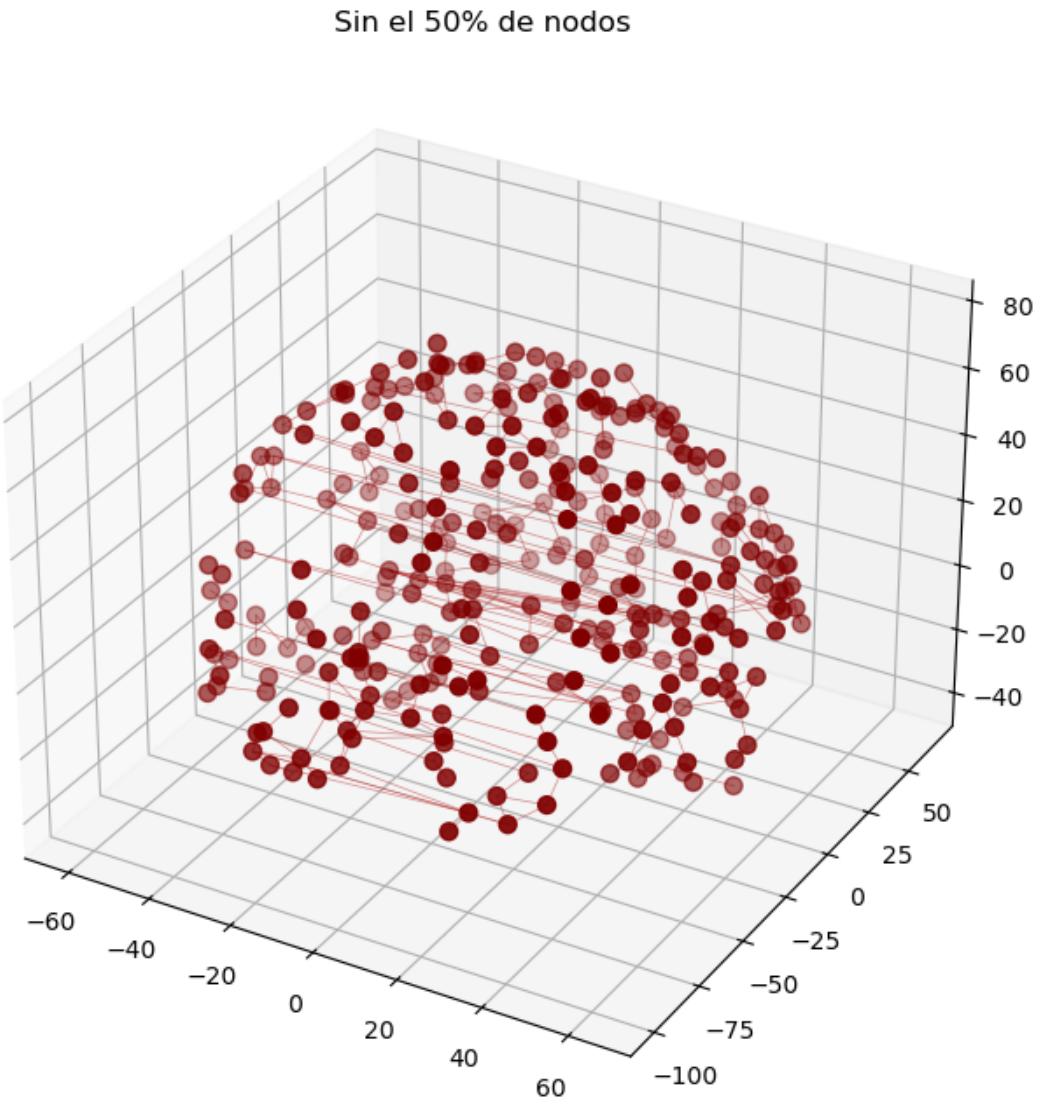
```
  Cambio: -45.02%

Centralidad_cercania:
  Original: 0.1194
  Modificado: 0.0108
  Cambio: -90.96%

Centralidad_intermediacion:
  Original: 0.0115
  Modificado: 0.0007
  Cambio: -93.93%
```




Sin 20% de nodos
80
60
40
20
0
−20
−40
50
25
0
−25
−50
−75
−100
−60
−40
−20
0
20
40
60


Sin el 30% de nodos
80
60
40
20
0
−20
−40
50
25
0
−25
−50
−75
−100
−60
−40
−20
0
20
40
60


Sin el 40% de nodos
80
60
40
20
0
−20
−40
50
25
0
−25
−50
−75
−100
−60
−40
−20
0
20
40
60



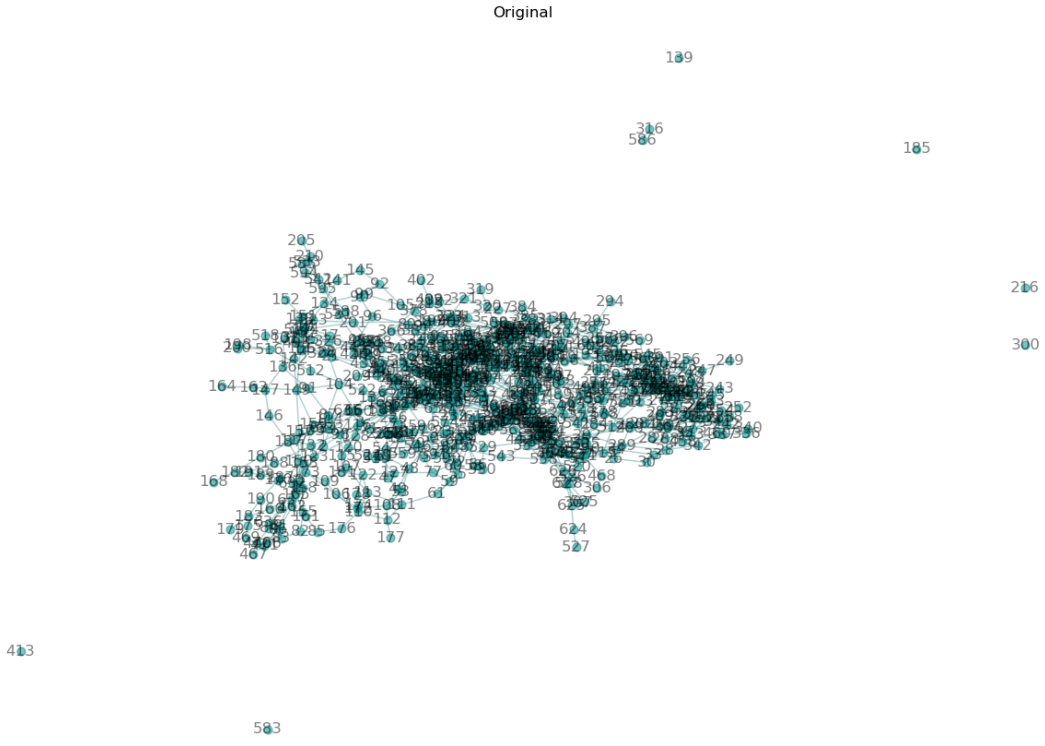

[410]:
```
# EJERCICIO 8. Generar un modelo nulo aleatorio donde se tenga el mismo número␣
 ↪de nodos y el
# mismo número total de conexiones, y comparar sus propiedades con el grafo␣
 ↪original del cerebro.
```

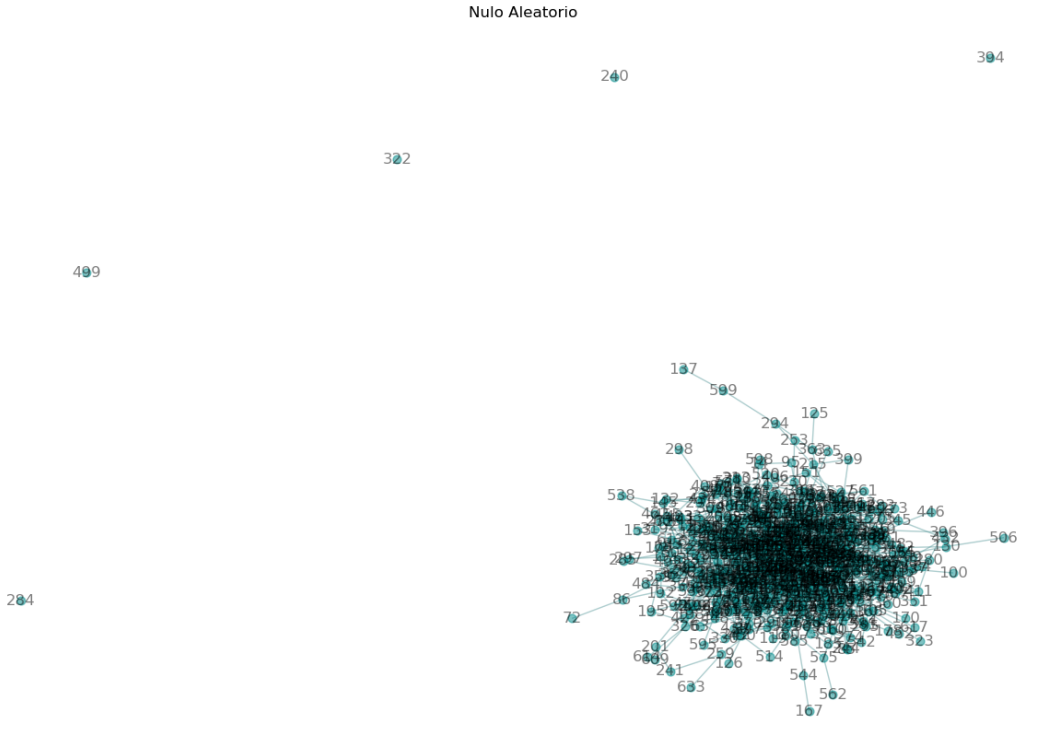
[411]:
```
def propiedades_grafo(G):
    propiedades = {}

    # average grade
    avg_grade = sum(dict(G.degree()).values()) / G.number_of_nodes()
    propiedades['grado_medio'] = avg_grade
```

```
    # coeficiente de agrupamiento
    coef_cluster = nx.average_clustering(G)
    propiedades['coeficiente_de_agrupamiento'] = coef_cluster

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

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

    return propiedades

def comparacion_propiedades(datos_originales, datos_modificados):
    """
    Compara las propiedades del grafo original y el modificado.
    """
    print("\nComparación de propiedades:")
    for prop in datos_originales.keys():
        original = datos_originales[prop]
        modificado = datos_modificados[prop]
        cambio = ((modificado - original) / original) * 100 if original != 0␣
 ↪else "N/A"
        print(f"{prop.capitalize()}:")
        print(f"  Original: {original:.4f}")
        print(f"  Modificado: {modificado:.4f}")
        print(f"  Cambio: {cambio if cambio == 'N/A' else cambio:.2f}%\n")


grafo_original = generate_graph(comat, 0.1, coordenadas)

# propiedades del original
print("Calculando las propiedades del grafo original:")
datos_originales = propiedades_grafo(grafo_original)
print(datos_originales)

# para hacer un modelo nulo aleatorio
numero_nodos = grafo_original.number_of_nodes()
numero_aristas = grafo_original.number_of_edges()

# con el mismo número de aristas que el original
modelo_nulo = nx.gnm_random_graph(numero_nodos, numero_aristas)

# propiedades del modelo aleatorio
print("Propiedades del modelo nulo aleatorio:")
```

```python
datos_nulo = propiedades_grafo(modelo_nulo)
print(datos_nulo)

# comparación de propiedades
comparacion_propiedades(datos_originales, datos_nulo)

# grafo original y modelo nulo aleatorio
def plot_grafo(G, titulo):
    pos = nx.spring_layout(G)  # Layout para visualizar en 2D
    plt.figure(figsize=(12, 8))
    nx.draw(G, pos, with_labels=True, node_color='darkcyan', node_size=40,␣
 ↪edge_color='cadetblue', alpha=0.5)
    plt.title(titulo)
    plt.show()
plot_grafo(grafo_original, "Original")
plot_grafo(modelo_nulo, "Nulo Aleatorio")
#%%
```

```
Calculando las propiedades del grafo original:
{'grado_medio': 4.529780564263323, 'coeficiente_de_agrupamiento':
0.24364464673564803, 'centralidad_cercania': 0.1194031264903796,
'centralidad_intermediacion': 0.01147712930871217}
Propiedades del modelo nulo aleatorio:
{'grado_medio': 4.529780564263323, 'coeficiente_de_agrupamiento':
0.005694958124425209, 'centralidad_cercania': 0.22315646535893222,
'centralidad_intermediacion': 0.005344780695649735}

Comparación de propiedades:
Grado_medio:
  Original: 4.5298
  Modificado: 4.5298
  Cambio: 0.00%

Coeficiente_de_agrupamiento:
  Original: 0.2436
  Modificado: 0.0057
  Cambio: -97.66%

Centralidad_cercania:
  Original: 0.1194
  Modificado: 0.2232
  Cambio: 86.89%

Centralidad_intermediacion:
  Original: 0.0115
  Modificado: 0.0053
  Cambio: -53.43%
```


Original


Nulo Aleatorio

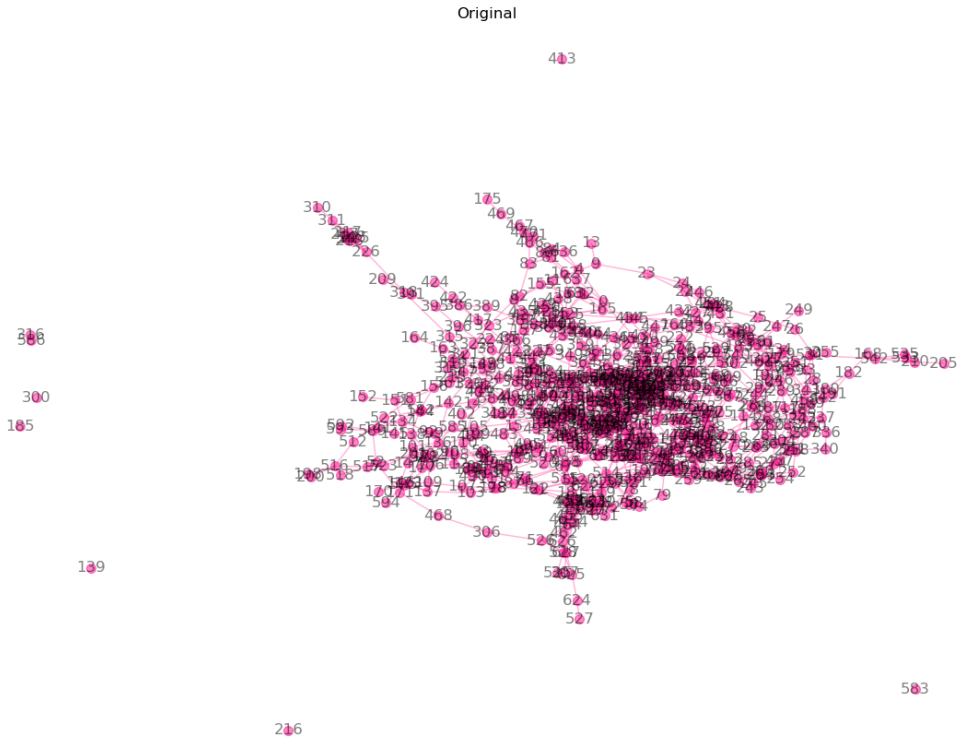
[412]:
```
# EJERCICIO 9. Generar un modelo nulo aleatorio donde se conserve la␣
 ↪distribución de grado y
# comparar sus propiedades con el grafo original del cerebro.
```

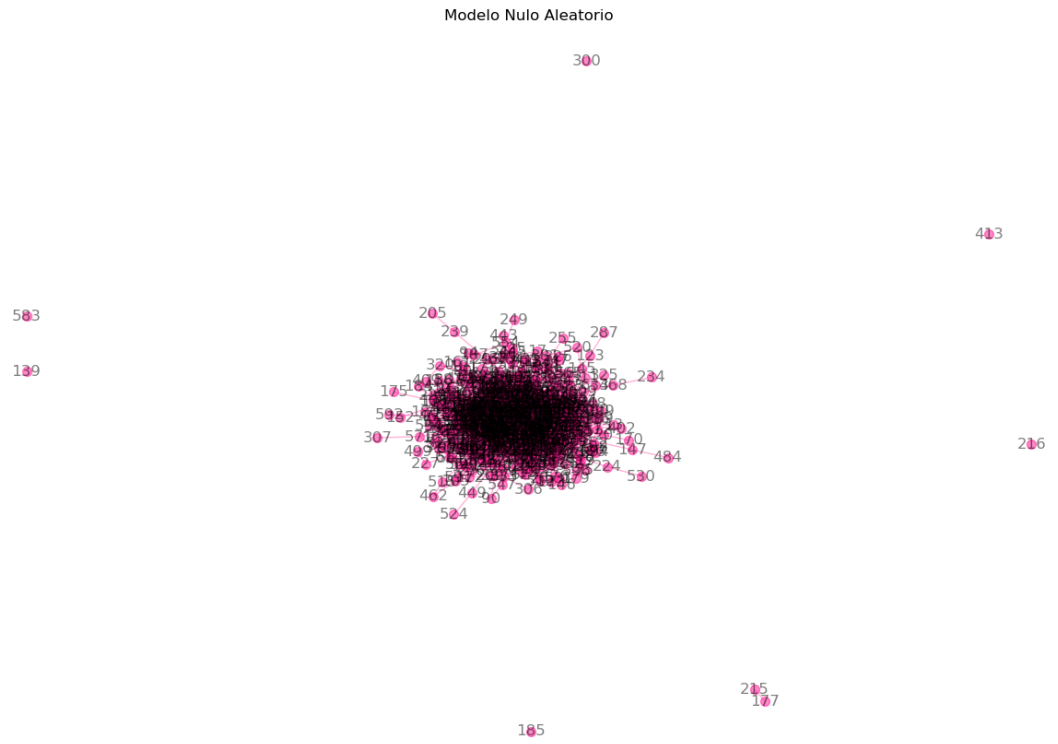
[413]:
```
import networkx as nx
import numpy as np
import matplotlib.pyplot as plt

def propiedades_grafo(G):
    propiedades = {}

    # Average grade
    avg_grade = sum(dict(G.degree()).values()) / G.number_of_nodes()
    propiedades['grado_medio'] = avg_grade

    # Coeficiente de cluster
    coef_cluster = nx.average_clustering(G)
    propiedades['coeficiente_de_agrupamiento'] = coef_cluster

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

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

    return propiedades

def comparacion_propiedades(datos_originales, datos_modificados):

    print("Comparación de propiedades:")
    for prop in datos_originales.keys():
        original = datos_originales[prop]
        modificado = datos_modificados[prop]
        cambio = ((modificado - original) / original) * 100 if original != 0␣
 ↪else "N/A"
        print(f"{prop.capitalize()}:")
        print(f"  Original: {original:.4f}")
        print(f"  Modificado: {modificado:.4f}")
        print(f"  Cambio: {cambio if cambio == 'N/A' else cambio:.2f}%\n")

# grafo original
```

```python
grafo_original = generate_graph(comat, 0.1, coordenadas)

#propiedades grafo original
print("Cálculo de las propiedades del grafo original:")
datos_originales = propiedades_grafo(grafo_original)
print(datos_originales)

# crear modelo nulo aleatorio con la misma distribución de grados
modelo_nulo = nx.configuration_model([grafo_original.degree(n) for n in␣
 ↪grafo_original.nodes()])

# modelo nulo a grafo simple
modelo_nulo_multigrafo = nx.configuration_model([grafo_original.degree(n) for n␣
 ↪in grafo_original.nodes()])

#eliminar self-loops
modelo_nulo_multigrafo.remove_edges_from(nx.
 ↪selfloop_edges(modelo_nulo_multigrafo))

# modelo nulo a grafo simple
modelo_nulo = nx.Graph(modelo_nulo_multigrafo)

# propiedades del modelo nulo aleatorio
print("Calculamos las propiedades del modelo nulo aleatorio:")
datos_nulo = propiedades_grafo(modelo_nulo)
print(datos_nulo)

# comparación de propiedades
comparacion_propiedades(datos_originales, datos_nulo)

# grafo original y modelo nulo aleatorio
def plot_grafo(G, titulo):
    pos = nx.spring_layout(G)
    plt.figure(figsize=(12, 8))
    nx.draw(G, pos, with_labels=True, node_color='deeppink', node_size=50,␣
 ↪edge_color='hotpink', alpha=0.5)
    plt.title(titulo)
    plt.show()
plot_grafo(grafo_original, "Original")
plot_grafo(modelo_nulo, "Modelo Nulo Aleatorio")
```

```
Cálculo de las propiedades del grafo original:
{'grado_medio': 4.529780564263323, 'coeficiente_de_agrupamiento':
0.24364464673564803, 'centralidad_cercania': 0.1194031264903796,
'centralidad_intermediacion': 0.01147712930871217}
Calculamos las propiedades del modelo nulo aleatorio:
{'grado_medio': 4.492163009404389, 'coeficiente_de_agrupamiento':
```

```
0.005780935877284761, 'centralidad_cercania': 0.2270149367324883,
'centralidad_intermediacion': 0.005130639413565337}
Comparación de propiedades:
Grado_medio:
  Original: 4.5298
  Modificado: 4.4922
  Cambio: -0.83%

Coeficiente_de_agrupamiento:
  Original: 0.2436
  Modificado: 0.0058
  Cambio: -97.63%

Centralidad_cercania:
  Original: 0.1194
  Modificado: 0.2270
  Cambio: 90.12%

Centralidad_intermediacion:
  Original: 0.0115
  Modificado: 0.0051
  Cambio: -55.30%
```






[414]:
```
# EJERCICIO 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.
```

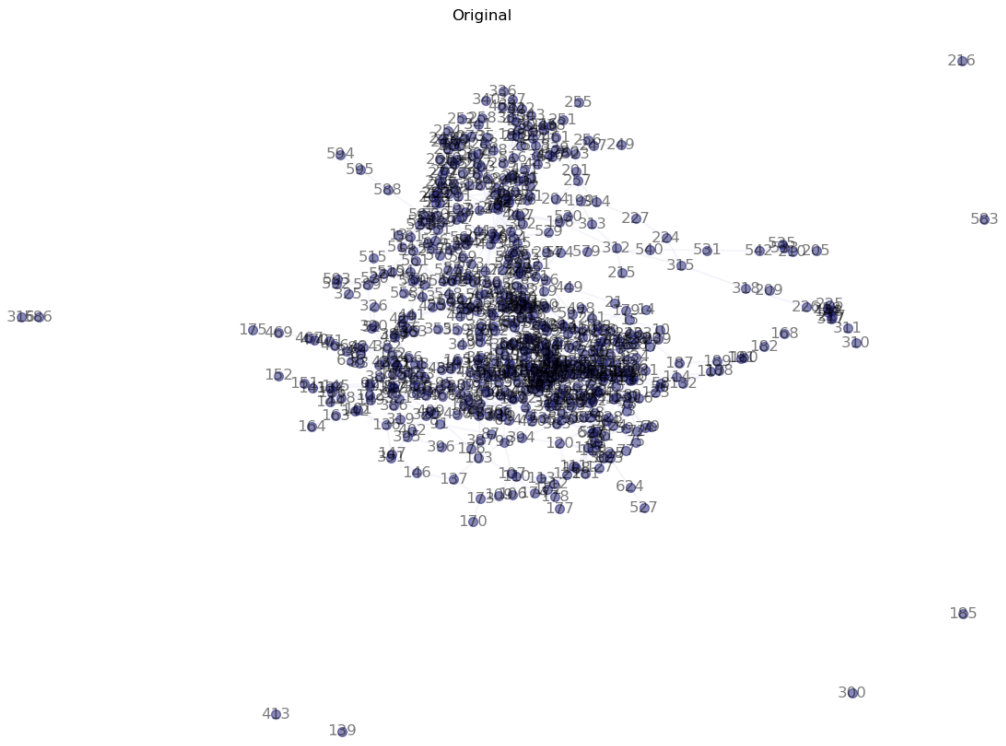
[417]:
```
import networkx as nx
import numpy as np
import matplotlib.pyplot as plt

def propiedades_grafo(G):
    propiedades = {}

    # average grade
    avg_grade = sum(dict(G.degree()).values()) / G.number_of_nodes()
    propiedades['grado_medio'] = avg_grade

    # coeficiente de cluster
    coef_cluster = nx.average_clustering(G)
    propiedades['coeficiente_de_agrupamiento'] = coef_cluster

    # centralidad de cercanía
```

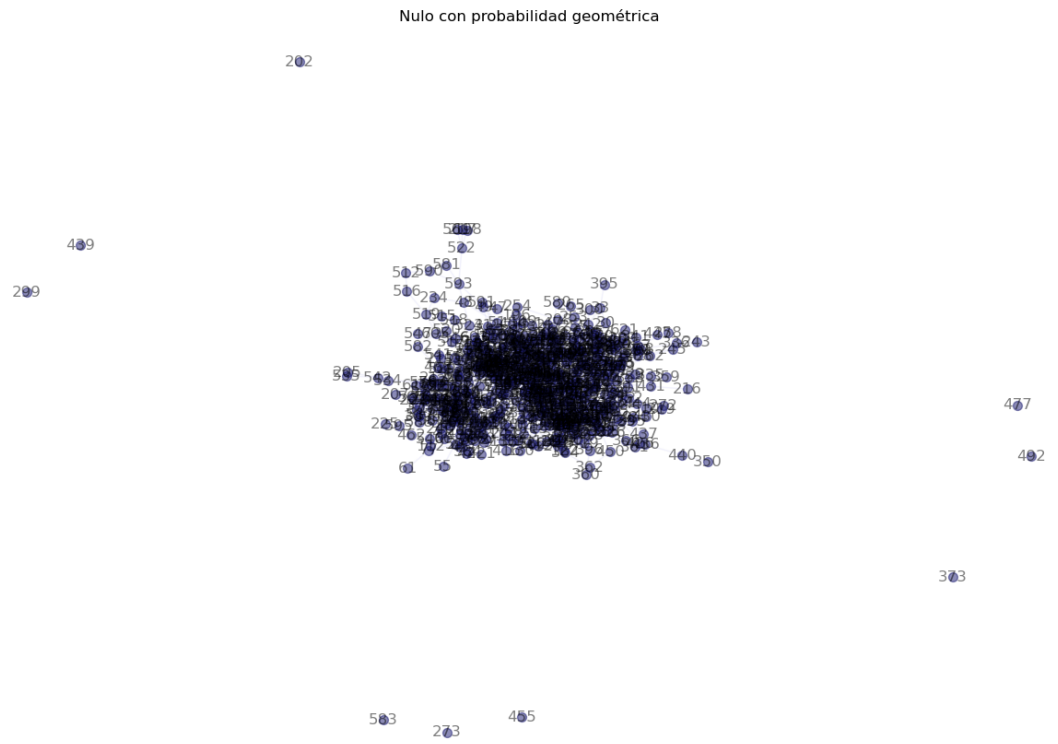
```
    cercania = np.mean(list(nx.closeness_centrality(G).values()))
    propiedades['centralidad_cercania'] = cercania

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

    return propiedades

def comparacion_propiedades(datos_originales, datos_modificados):

    print("\nComparación de propiedades:")
    for prop in datos_originales.keys():
        original = datos_originales[prop]
        modificado = datos_modificados[prop]
        cambio = ((modificado - original) / original) * 100 if original != 0␣
 ↪else "N/A"
        print(f"{prop.capitalize()}:")
        print(f"  Original: {original:.4f}")
        print(f"  Modificado: {modificado:.4f}")
        print(f"  Cambio: {cambio if cambio == 'N/A' else cambio:.2f}%\n")

# original
grafo_original = generate_graph(comat, 0.1, coordenadas)

# propiedades del grafo original
print("Calculando las propiedades del grafo original:")
datos_originales = propiedades_grafo(grafo_original)
print(datos_originales)

# coordenadas de los nodos
nodos = list(grafo_original.nodes())
pos = nx.get_node_attributes(grafo_original, 'pos')

# matriz de distancias entre nodos
distancias = np.zeros((len(nodos), len(nodos)))
for i in range(len(nodos)):
    for j in range(len(nodos)):
        if i != j:
            distancias[i, j] = np.linalg.norm(np.array(pos[i]) - np.
 ↪array(pos[j]))

# parámetro de decaimiento para la probabilidad de conexión
alpha = 0.1

# matriz de probabilidades según distancia
```

```
probabilidades = np.exp(-alpha * distancias)

# crear modelo nudo en probabilidades de conexión
numero_aristas = grafo_original.number_of_edges()
modelo_nulo = nx.Graph()
modelo_nulo.add_nodes_from(nodos)

# conexiones aleatorias basadas en las probabilidades
aristas = []
while len(aristas) < numero_aristas:
    i, j = np.random.choice(len(nodos), size=2, replace=False)
    if (i, j) not in aristas and (j, i) not in aristas:
        if np.random.rand() < probabilidades[i, j]:
            aristas.append((i, j))

modelo_nulo.add_edges_from(aristas)

# calcular propiedades del modelo nulo
print("Calcular propiedades del modelo nulo con probabilidad geométrica:")
datos_modelonulo = propiedades_grafo(modelo_nulo)
print(datos_nulo)

# comparación de propiedades entre grafo original y el modelo nulo
comparacion_propiedades(datos_originales, datos_modelonulo)

# grafo original y modelo nulo
def plot_grafo(G, titulo):
    pos = nx.spring_layout(G)  # Layout para visualizar en 2D
    plt.figure(figsize=(12, 8))
    nx.draw(G, pos, with_labels=True, node_color='midnightblue', node_size=50,␣
 ↪edge_color='lavender', alpha=0.5)
    plt.title(titulo)
    plt.show()
plot_grafo(grafo_original, "Original")
plot_grafo(modelo_nulo, "Nulo con probabilidad geométrica")
```

```
Calculando las propiedades del grafo original:
{'grado_medio': 4.529780564263323, 'coeficiente_de_agrupamiento':
0.24364464673564803, 'centralidad_cercania': 0.1194031264903796,
'centralidad_intermediacion': 0.01147712930871217}
Calcular propiedades del modelo nulo con probabilidad geométrica:
{'grado_medio': 4.492163009404389, 'coeficiente_de_agrupamiento':
0.005780935877284761, 'centralidad_cercania': 0.2270149367324883,
'centralidad_intermediacion': 0.005130639413565337}

Comparación de propiedades:
Grado_medio:
```

```
  Original: 4.5298
  Modificado: 4.5298
  Cambio: 0.00%

Coeficiente_de_agrupamiento:
  Original: 0.2436
  Modificado: 0.0562
  Cambio: -76.91%

Centralidad_cercania:
  Original: 0.1194
  Modificado: 0.1876
  Cambio: 57.15%

Centralidad_intermediacion:
  Original: 0.0115
  Modificado: 0.0065
  Cambio: -43.55%
```






[418]:
```
#11. RESEÑA DEL CURSO
```

[419]:
```
# Me hubiese gustado que la clase fuera diferente, ya que al menos a mí, me␣
 ↪pareció poco efectiva la forma de enseñar,
# aunque también creo que lo que más nos afectó en el aprendizaje fue el␣
 ↪horario, al menos para las de Python, al ser
# nuestra última clase, llegabamos ya cansados y saturados de lo que teníamos␣
 ↪antes, además nos atrasamos mucho debido a las personas
# que usaban MacBook y en un punto dejé de comprender lo que estábamos haciendo.
 ↪ Honestamente, tuve que recurrir a IA para que me explicara
# ejercicios y justo por eso siento que no puedo hacer completamente sin ayuda␣
 ↪los ejercicios, necesito practicar, incluso para esto Anel tuvo
# que explicarme muchas cosas y ayudarme cuando me trababa.
```

[420]:
```
# Respecto a la importancia: La teoría de grafos es una herramienta muy útil␣
 ↪para entender cómo se conecta el cerebro.
# Se representa el cerebro como una red, los nodos son las neuronas o regiones␣
 ↪cerebrales y las líneas representan las conexiones entre ellas.
# Así podemos analizar el funcionamiento y comportamiento de redes neuronales.
# Se pueden identificar hubs (nodos que conectan muchas regiones). Algunas␣
 ↪áreas pueden ser representadas como módulos.
```

```
# Tiene muchas aplicaciones que nos permiten estudiar cómo cambia el cerebro en␣
 ↪diferentes condiciones o eventos, por ejemplo, meuroplasticidad.
```

[ ]: