Proyecto

November 29, 2024

[1]:
```
pip install scipy networkx matplotlib numpy
```

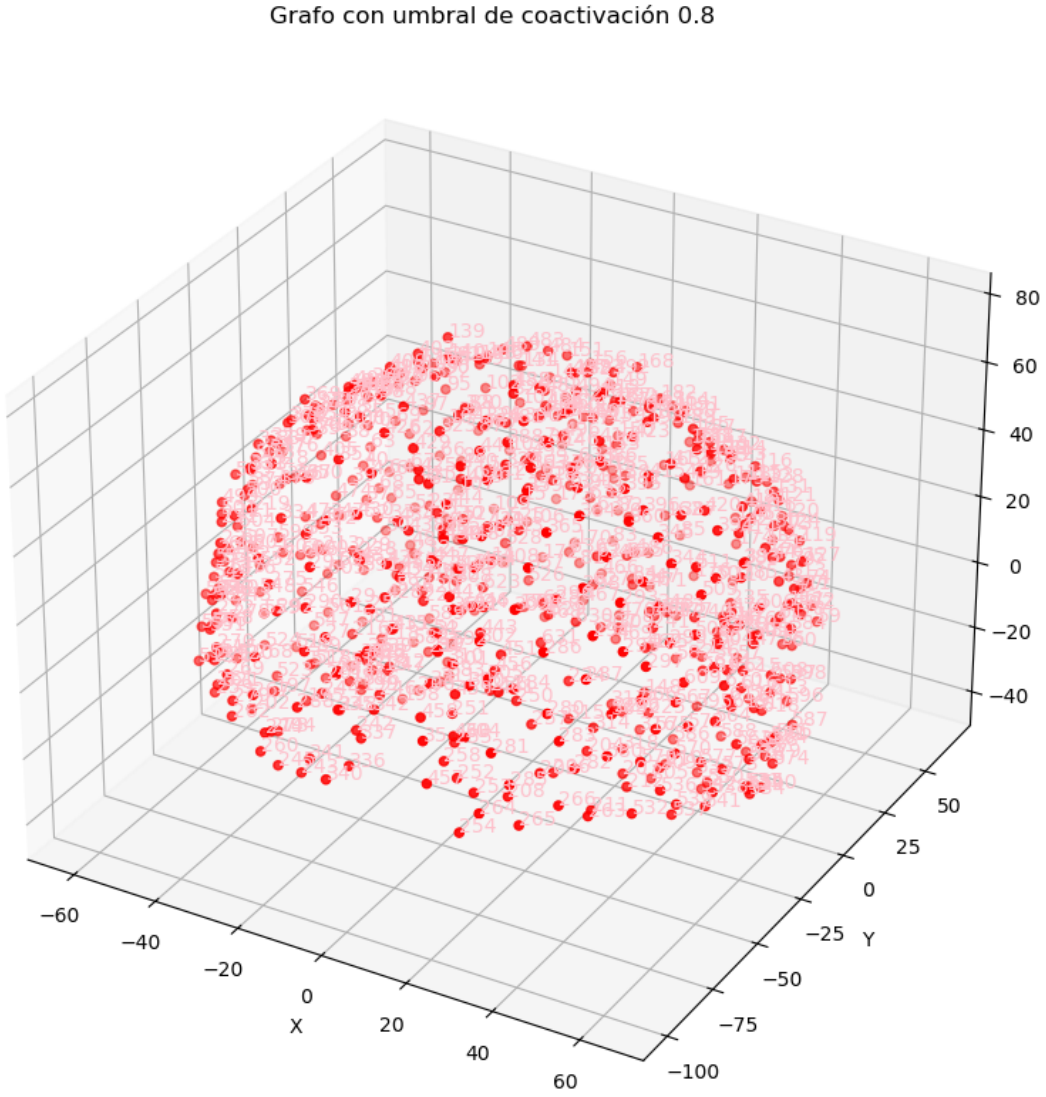
```
Defaulting to user installation because normal site-packages is not writeable
Looking in links: /usr/share/pip-wheels
Requirement already satisfied: scipy in /opt/conda/envs/anaconda-
ai-2024.04-py310/lib/python3.10/site-packages (1.12.0)
Requirement already satisfied: networkx in ./.local/lib/python3.10/site-packages
(3.4.2)
Requirement already satisfied: matplotlib in /opt/conda/envs/anaconda-
ai-2024.04-py310/lib/python3.10/site-packages (3.8.0)
Requirement already satisfied: numpy in /opt/conda/envs/anaconda-
ai-2024.04-py310/lib/python3.10/site-packages (1.26.4)
Requirement already satisfied: contourpy>=1.0.1 in /opt/conda/envs/anaconda-
ai-2024.04-py310/lib/python3.10/site-packages (from matplotlib) (1.2.0)
Requirement already satisfied: cycler>=0.10 in /opt/conda/envs/anaconda-
ai-2024.04-py310/lib/python3.10/site-packages (from matplotlib) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in /opt/conda/envs/anaconda-
ai-2024.04-py310/lib/python3.10/site-packages (from matplotlib) (4.25.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /opt/conda/envs/anaconda-
ai-2024.04-py310/lib/python3.10/site-packages (from matplotlib) (1.4.4)
Requirement already satisfied: packaging>=20.0 in /opt/conda/envs/anaconda-
ai-2024.04-py310/lib/python3.10/site-packages (from matplotlib) (23.2)
Requirement already satisfied: pillow>=6.2.0 in /opt/conda/envs/anaconda-
ai-2024.04-py310/lib/python3.10/site-packages (from matplotlib) (10.2.0)
Requirement already satisfied: pyparsing>=2.3.1 in /opt/conda/envs/anaconda-
ai-2024.04-py310/lib/python3.10/site-packages (from matplotlib) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in /opt/conda/envs/anaconda-
ai-2024.04-py310/lib/python3.10/site-packages (from matplotlib) (2.8.2)
Requirement already satisfied: six>=1.5 in /opt/conda/envs/anaconda-
ai-2024.04-py310/lib/python3.10/site-packages (from python-
dateutil>=2.7->matplotlib) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

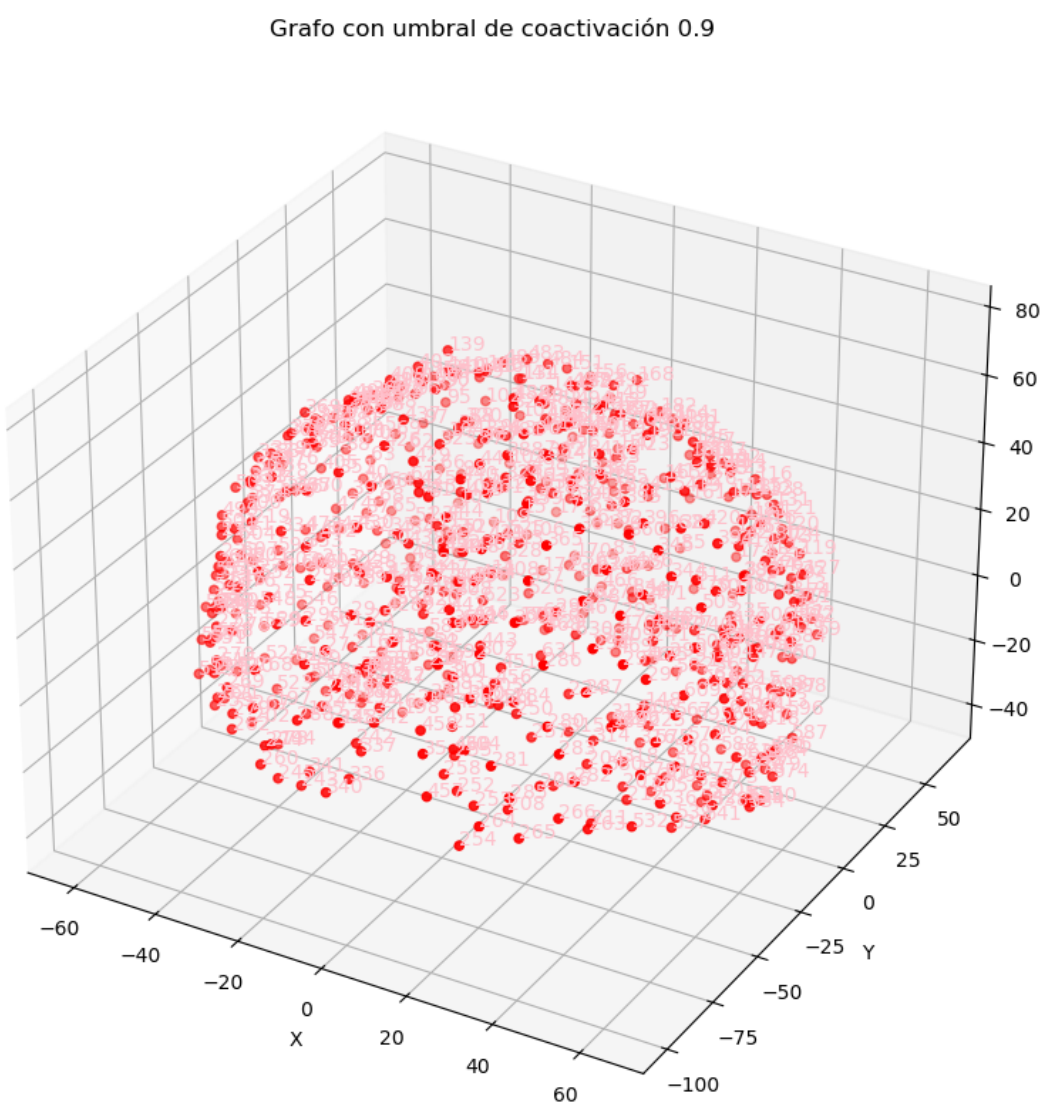
[19]:
```
import numpy as np
import scipy.io as sio
import networkx as nx
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

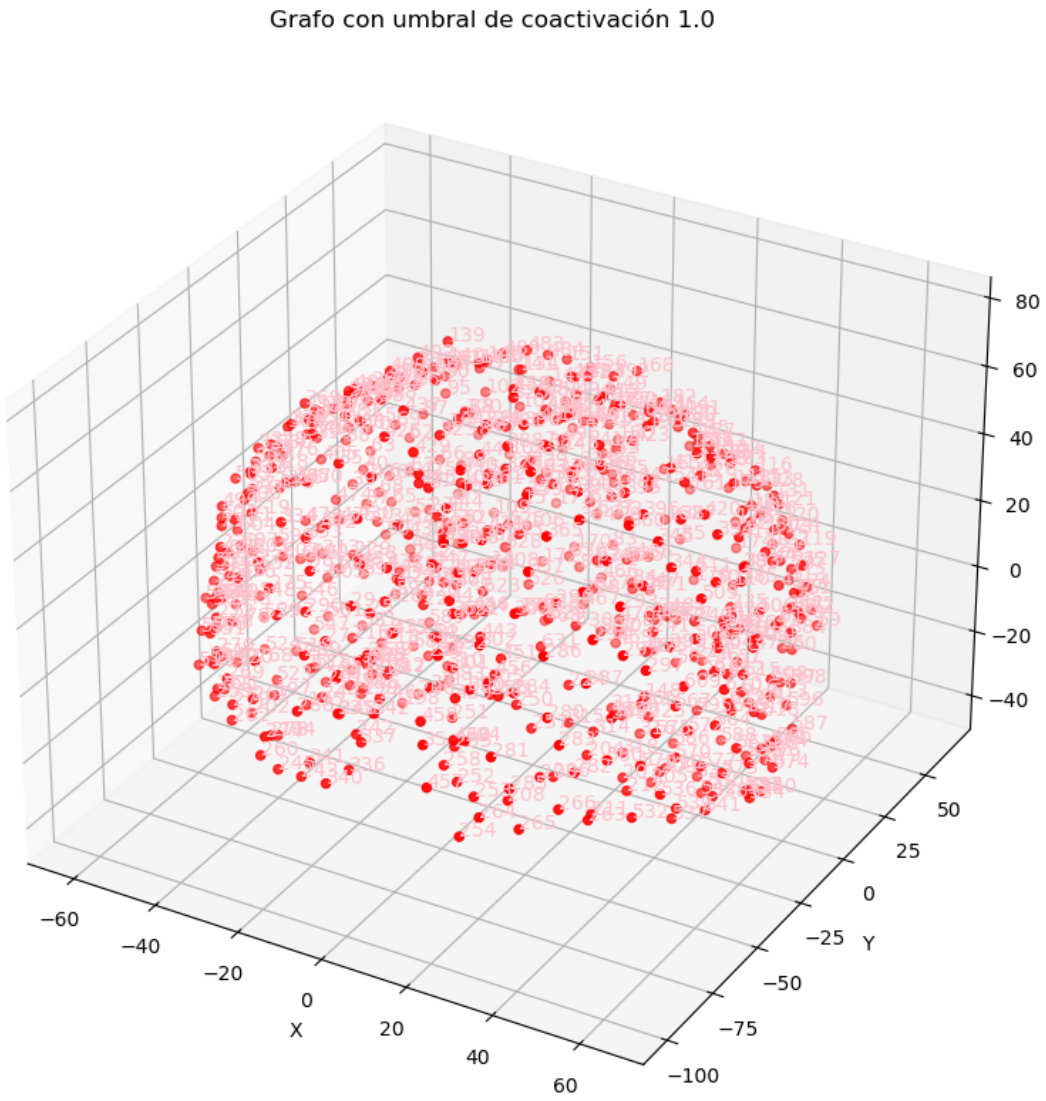
```
mat_data = sio.loadmat('Coactivation_matrix.mat')

print(mat_data.keys())

coactivation_matrix = mat_data['Coactivation_matrix']
coordinates = mat_data['Coord']

coactivation_matrix = np.array(coactivation_matrix)

umbral_1 = 0.8
umbral_2 = 0.9
umbral_3 = 1.0

def graficar_grafo(coactivation_matrix, umbral, coordenadas):
    G = nx.Graph()

    for i in range(coactivation_matrix.shape[0]):
        G.add_node(i, pos=coordenadas[i])

    for i in range(coactivation_matrix.shape[0]):
        for j in range(i+1, coactivation_matrix.shape[1]):
            if coactivation_matrix[i, j] >= umbral:
                G.add_edge(i, j, weight=coactivation_matrix[i, j])

    pos = nx.get_node_attributes(G, 'pos')

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

    x_vals = [pos[node][0] for node in pos]
    y_vals = [pos[node][1] for node in pos]
    z_vals = [pos[node][2] for node in pos]

    for (i, j) in G.edges():
        x_vals_edge = [pos[i][0], pos[j][0]]
        y_vals_edge = [pos[i][1], pos[j][1]]
        z_vals_edge = [pos[i][2], pos[j][2]]
        ax.plot(x_vals_edge, y_vals_edge, z_vals_edge, color='b', alpha=0.5)

    ax.scatter(x_vals, y_vals, z_vals, c='r', marker='o')

    for i, (x, y, z) in enumerate(zip(x_vals, y_vals, z_vals)):
        ax.text(x, y, z, str(i), color='pink', fontsize=10)

    ax.set_title(f'Grafo con umbral de coactivación {umbral}')
    ax.set_xlabel('X')
```

```
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')

    plt.show()

graficar_grafo(coactivation_matrix, umbral_1, coordinates)
graficar_grafo(coactivation_matrix, umbral_2, coordinates)
graficar_grafo(coactivation_matrix, umbral_3, coordinates)
```

```
dict_keys(['__header__', '__version__', '__globals__', 'Coactivation_matrix',
'Coord'])
```



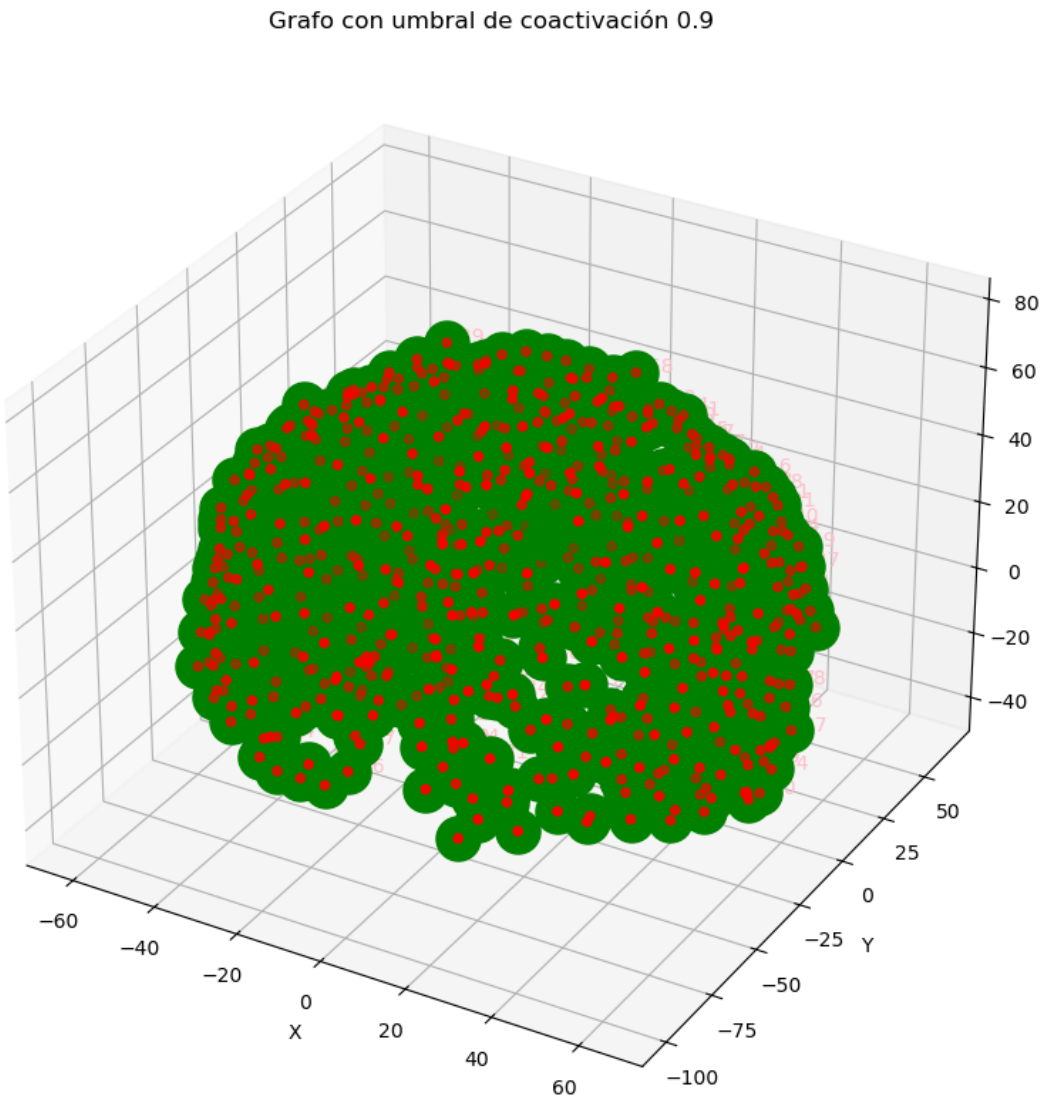

Grafo con umbral de coactivación 0.9
X
Y
−60
−40
−20
0
20
40
60
−100
−75
−50
−25
0
25
50
−40
−20
0
20
40
60
80



[31]:
```
from matplotlib.animation import FuncAnimation

mat_data = sio.loadmat('Coactivation_matrix.mat')
coactivation_matrix = mat_data['Coactivation_matrix']
coordinates = mat_data['Coord']

coactivation_matrix = np.array(coactivation_matrix)

umbral = 0.9

def graficar_grafo_animado(coactivation_matrix, umbral, coordenadas):
    G = nx.Graph()
```

```
for i in range(coactivation_matrix.shape[0]):
    G.add_node(i, pos=coordenadas[i])

for i in range(coactivation_matrix.shape[0]):
    for j in range(i+1, coactivation_matrix.shape[1]):
        if coactivation_matrix[i, j] >= umbral:
            G.add_edge(i, j, weight=coactivation_matrix[i, j])

pos = nx.get_node_attributes(G, 'pos')

degrees = dict(G.degree())

max_degree = max(degrees.values())
hubs = [node for node, degree in degrees.items() if degree == max_degree]

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

x_vals = [pos[node][0] for node in pos]
y_vals = [pos[node][1] for node in pos]
z_vals = [pos[node][2] for node in pos]

for (i, j) in G.edges():
    x_vals_edge = [pos[i][0], pos[j][0]]
    y_vals_edge = [pos[i][1], pos[j][1]]
    z_vals_edge = [pos[i][2], pos[j][2]]
    ax.plot(x_vals_edge, y_vals_edge, z_vals_edge, color='b', alpha=0.5)

node_sizes = [degrees[node] * 50 for node in pos]

scatter = ax.scatter(x_vals, y_vals, z_vals, c='r', marker='o')

for hub in hubs:
    x, y, z = pos[hub]
    ax.scatter(x, y, z, c='g', marker='o', s=500)

for i, (x, y, z) in enumerate(zip(x_vals, y_vals, z_vals)):
    ax.text(x, y, z, str(i), color='pink', fontsize=10)

ax.set_title(f'Grafo con umbral de coactivación {umbral}')
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')

def update(num):
    ax.view_init(elev=20, azim=num)
```

```
        return scatter,

    ani = FuncAnimation(fig, update, frames=np.arange(0, 360, 1), interval=50,␣
 ↪blit=False)

    plt.show()

graficar_grafo_animado(coactivation_matrix, umbral, coordinates)
```



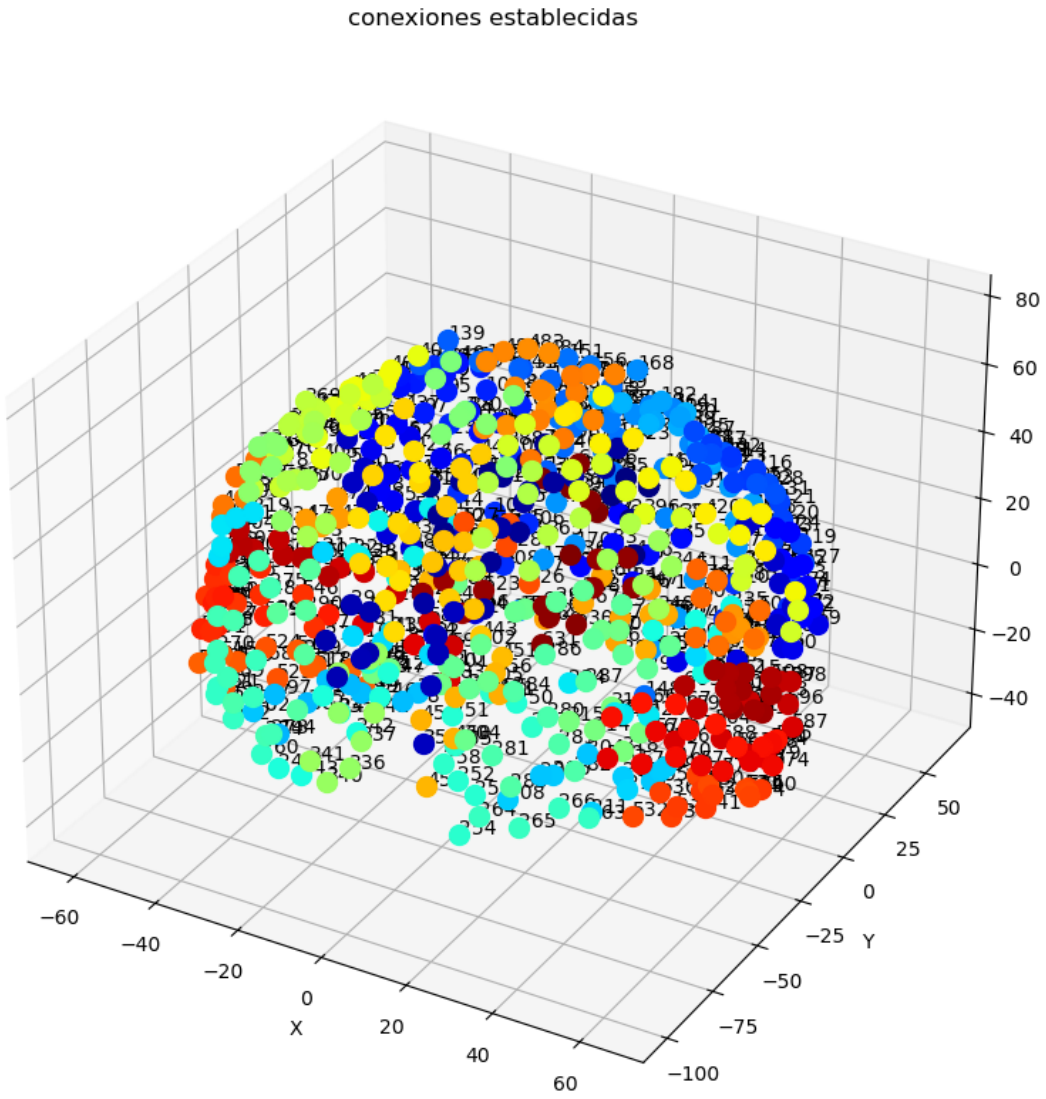
[39]:
```
pip install python-louvain
```

```
Defaulting to user installation because normal site-packages is not writeable
```

```
Looking in links: /usr/share/pip-wheels
Collecting python-louvain
  Downloading python-louvain-0.16.tar.gz (204 kB)
                             204.6/204.6
kB 4.4 MB/s eta 0:00:005 MB/s eta 0:00:01
  Preparing metadata (setup.py) … done
Requirement already satisfied: networkx in ./.local/lib/python3.10/site-
packages (from python-louvain) (3.4.2)
Requirement already satisfied: numpy in /opt/conda/envs/anaconda-
ai-2024.04-py310/lib/python3.10/site-packages (from python-louvain) (1.26.4)
Building wheels for collected packages: python-louvain
  Building wheel for python-louvain (setup.py) … done
  Created wheel for python-louvain: filename=python_louvain-0.16-py3-none-
any.whl size=9389
sha256=64334165196372a655f6eac8205a1e5b33fc09dde965b4cf5e5f6eb11641c1e7
  Stored in directory: /home/bb488668-48f4-4d48-bc2d-
0bcce24566c1/.cache/pip/wheels/d0/b0/d7/6dd26c3817810fa379088eaeb755a01d9a2a411c
37632079d1
Successfully built python-louvain
Installing collected packages: python-louvain
Successfully installed python-louvain-0.16
Note: you may need to restart the kernel to use updated packages.
```

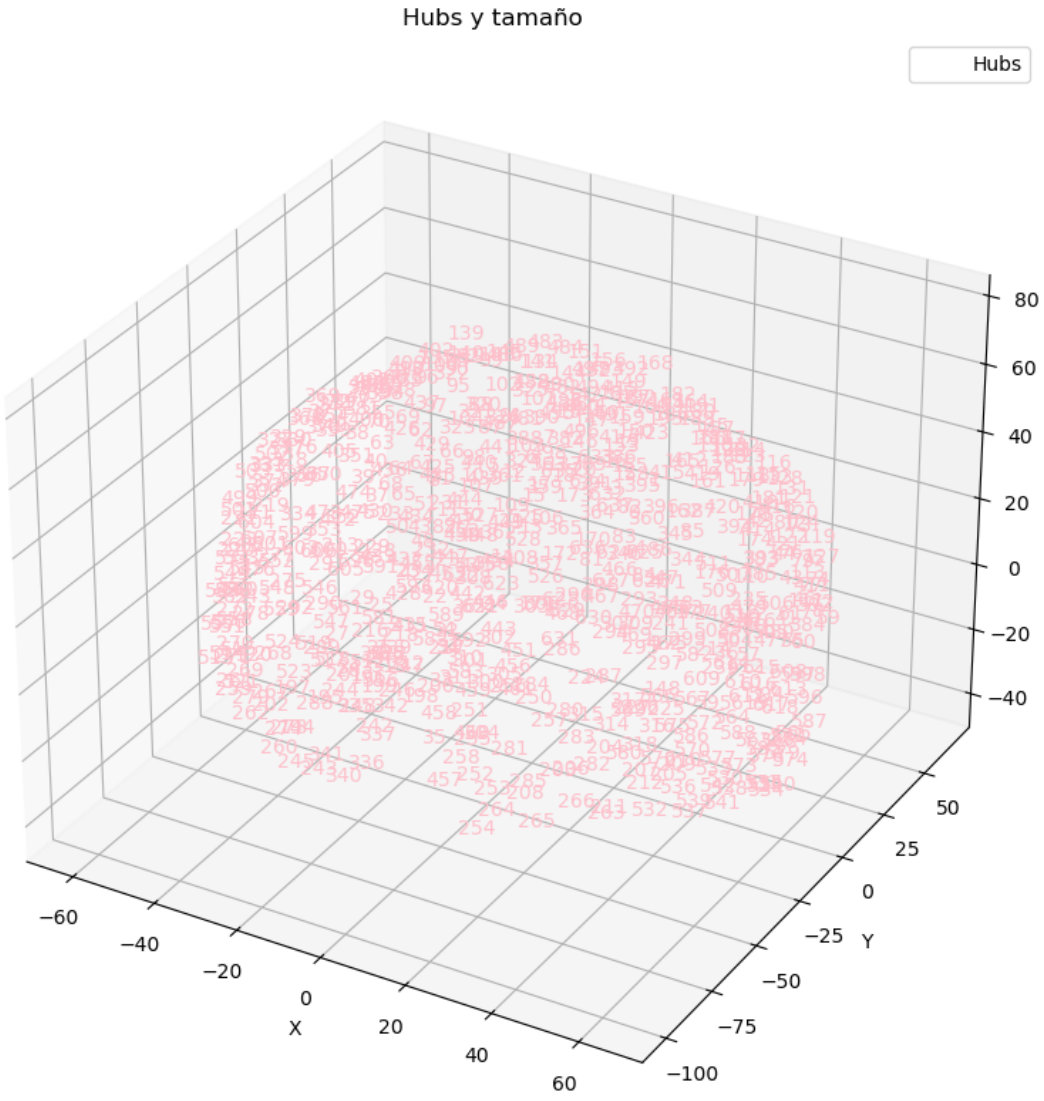
[58]:
```python
from community import community_louvain

coactivation_matrix = np.array(coactivation_matrix)

def graficar_grafo_animado(coactivation_matrix, umbral, coordenadas):
    G = nx.Graph()

    for i in range(coactivation_matrix.shape[0]):
        G.add_node(i, pos=coordenadas[i])

    partition = community_louvain.best_partition(G)

    comunidades = {}
    for node, comm in partition.items():
        if comm not in comunidades:
            comunidades[comm] = []
        comunidades[comm].append(node)

    pos = nx.get_node_attributes(G, 'pos')
    fig = plt.figure(figsize=(10, 10))
    ax = fig.add_subplot(111, projection='3d')

    x_vals = [pos[node][0] for node in pos]
    y_vals = [pos[node][1] for node in pos]
```

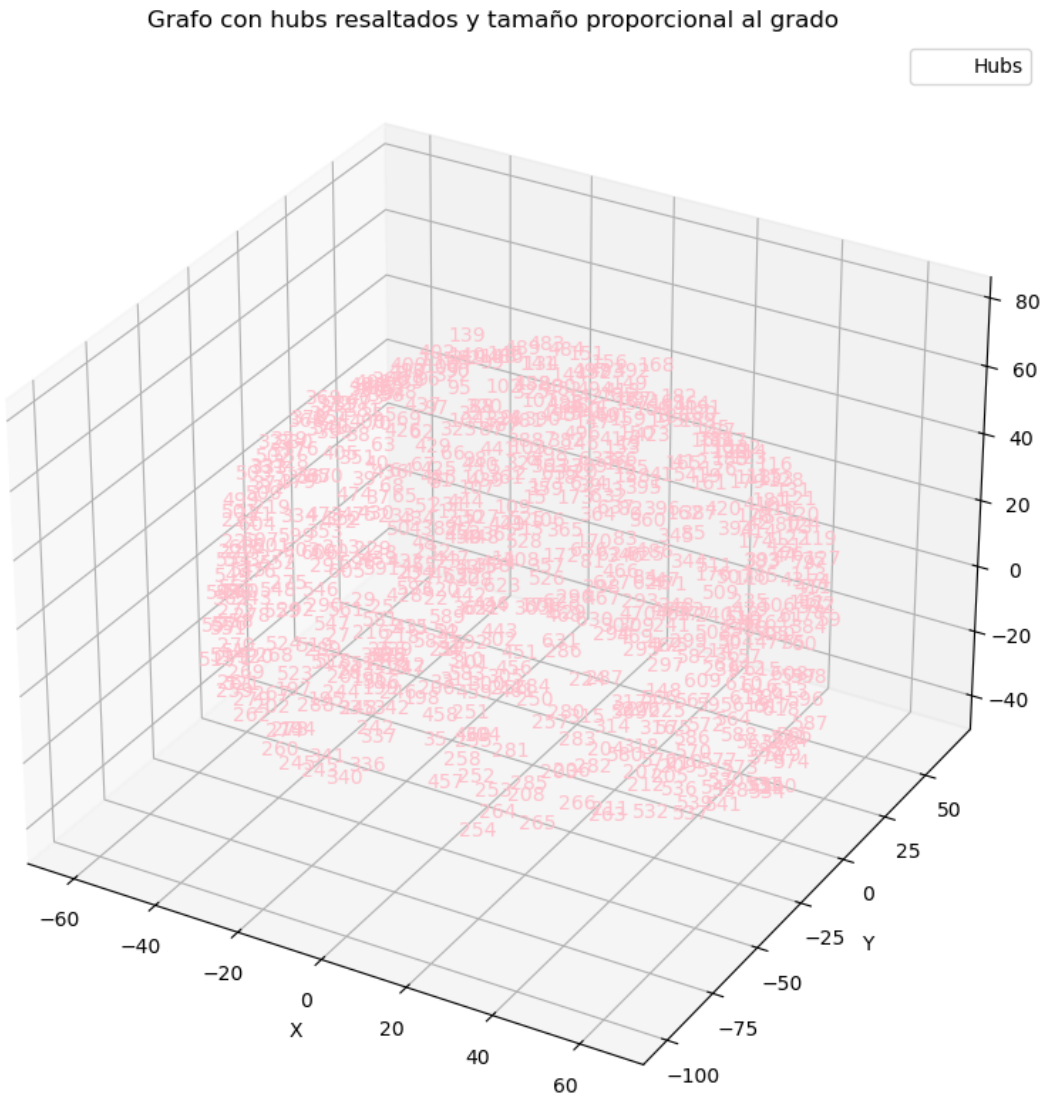
```
    z_vals = [pos[node][2] for node in pos]

    for (i, j) in G.edges():
        x_vals_edge = [pos[i][0], pos[j][0]]
        y_vals_edge = [pos[i][1], pos[j][1]]
        z_vals_edge = [pos[i][2], pos[j][2]]
        ax.plot(x_vals_edge, y_vals_edge, z_vals_edge, color='b', alpha=0.5)

    colores = plt.cm.jet(np.linspace(0, 1, len(comunidades)))
    for idx, comm in enumerate(comunidades.values()):
        color = colores[idx]
        x_vals_comm = [pos[node][0] for node in comm]
        y_vals_comm = [pos[node][1] for node in comm]
        z_vals_comm = [pos[node][2] for node in comm]
        ax.scatter(x_vals_comm, y_vals_comm, z_vals_comm, c=[color],␣
↪marker='o', s=100)

    for i, (x, y, z) in enumerate(zip(x_vals, y_vals, z_vals)):
        ax.text(x, y, z, str(i), color='black', fontsize=10)

    ax.set_title(f'conexiones establecidas')
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')

    plt.show()

graficar_comunidades(coactivation_matrix, umbral, coordinates)
```



[60]:
```
grado = dict(G.degree())

max_grado = max(grado.values())
hubs = [node for node, deg in grado.items() if deg == max_grado]

pos = nx.get_node_attributes(G, 'pos')
fig = plt.figure(figsize=(10, 10))
ax = fig.add_subplot(111, projection='3d')

x_vals = [pos[node][0] for node in pos]
y_vals = [pos[node][1] for node in pos]
z_vals = [pos[node][2] for node in pos]
```

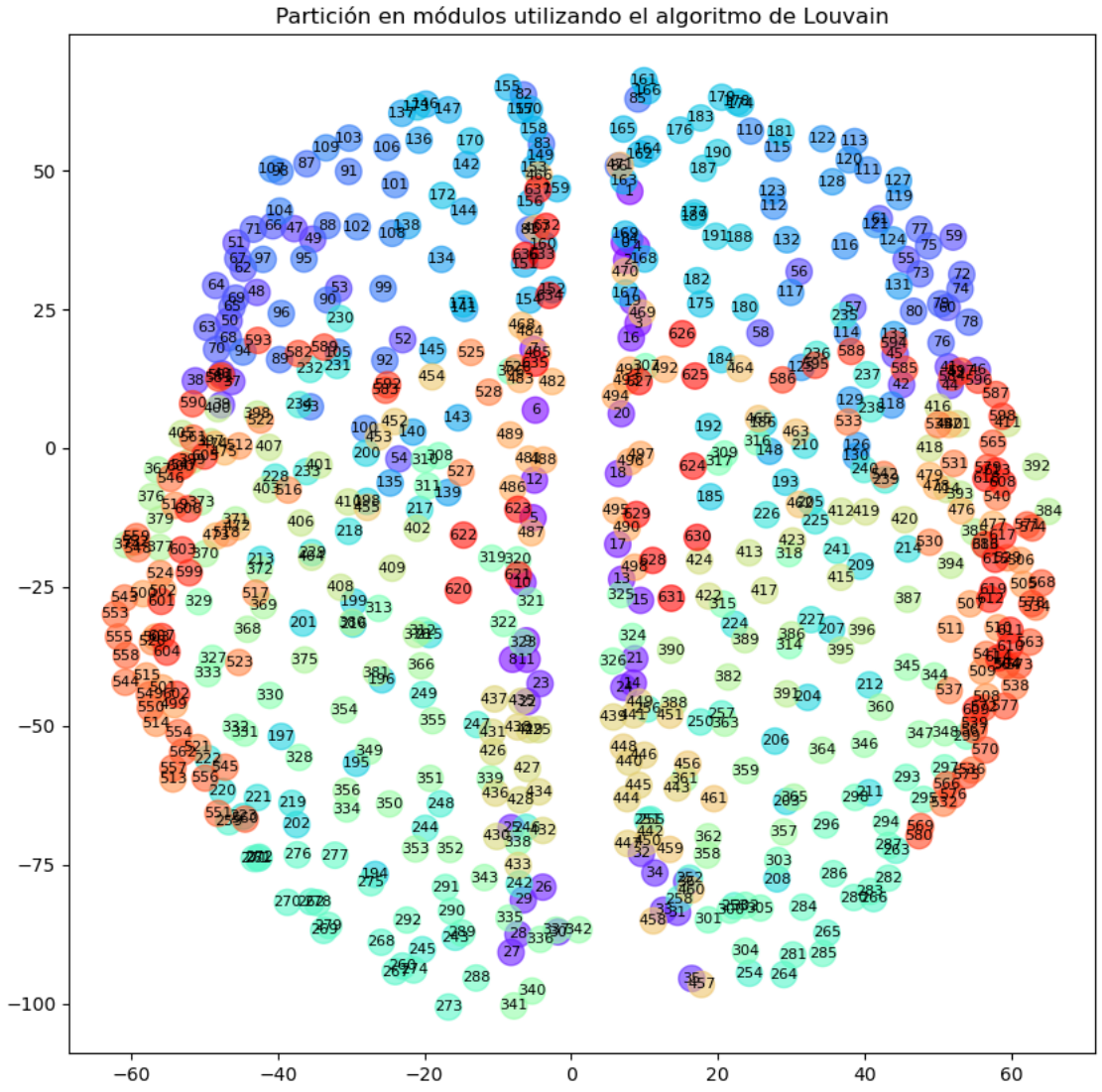
```
    for (i, j) in G.edges():
        x_vals_edge = [pos[i][0], pos[j][0]]
        y_vals_edge = [pos[i][1], pos[j][1]]
        z_vals_edge = [pos[i][2], pos[j][2]]
        ax.plot(x_vals_edge, y_vals_edge, z_vals_edge, color='b', alpha=0.5)

    node_sizes = [grado[node] * 100 for node in G.nodes()]

    ax.scatter(x_vals, y_vals, z_vals, c='r', marker='o', s=node_sizes)

    x_vals_hub = [pos[node][0] for node in hubs]
    y_vals_hub = [pos[node][1] for node in hubs]
    z_vals_hub = [pos[node][2] for node in hubs]
    ax.scatter(x_vals_hub, y_vals_hub, z_vals_hub, c='g', marker='o',␣
 ↪s=[grado[node] * 150 for node in hubs], label='Hubs')

    for i, (x, y, z) in enumerate(zip(x_vals, y_vals, z_vals)):
        ax.text(x, y, z, str(i), color='pink', fontsize=10)

    ax.set_title(f'Hubs y tamaño')
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')

    ax.legend()

    plt.show()

graficar_hubs(coactivation_matrix, umbral, coordinates)
```


Hubs y tamaño
Hubs
X
Y
−60
−40
−20
0
20
40
60
80
60
40
20
0
−20
−40
50
25
0
−25
−50
−75
−100



[68]:
```
partition = community_louvain.best_partition(G)

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

pos = nx.get_node_attributes(G, 'pos')
plt.title(f'Partición en módulos utilizando el algoritmo de Louvain')

community_colors = list(partition.values())
plt.scatter([pos[node][0] for node in G.nodes()],
            [pos[node][1] for node in G.nodes()],
            c=community_colors, cmap=plt.cm.rainbow, s=200, alpha=0.6)
```

```
for i, (x, y) in enumerate(zip([pos[node][0] for node in G.nodes()],
                               [pos[node][1] for node in G.nodes()])):
    plt.text(x, y, str(i), color='black', fontsize=8, ha='center', va='center')

plt.show()

num_communities = len(set(partition.values()))
print(f"Se detectaron {num_communities} comunidades.")
print("Partición de nodos:", partition)
```




```
Se detectaron 638 comunidades.
Partición de nodos: {0: 0, 1: 1, 2: 2, 3: 3, 4: 4, 5: 5, 6: 6, 7: 7, 8: 8, 9: 9,
10: 10, 11: 11, 12: 12, 13: 13, 14: 14, 15: 15, 16: 16, 17: 17, 18: 18, 19: 19,
```

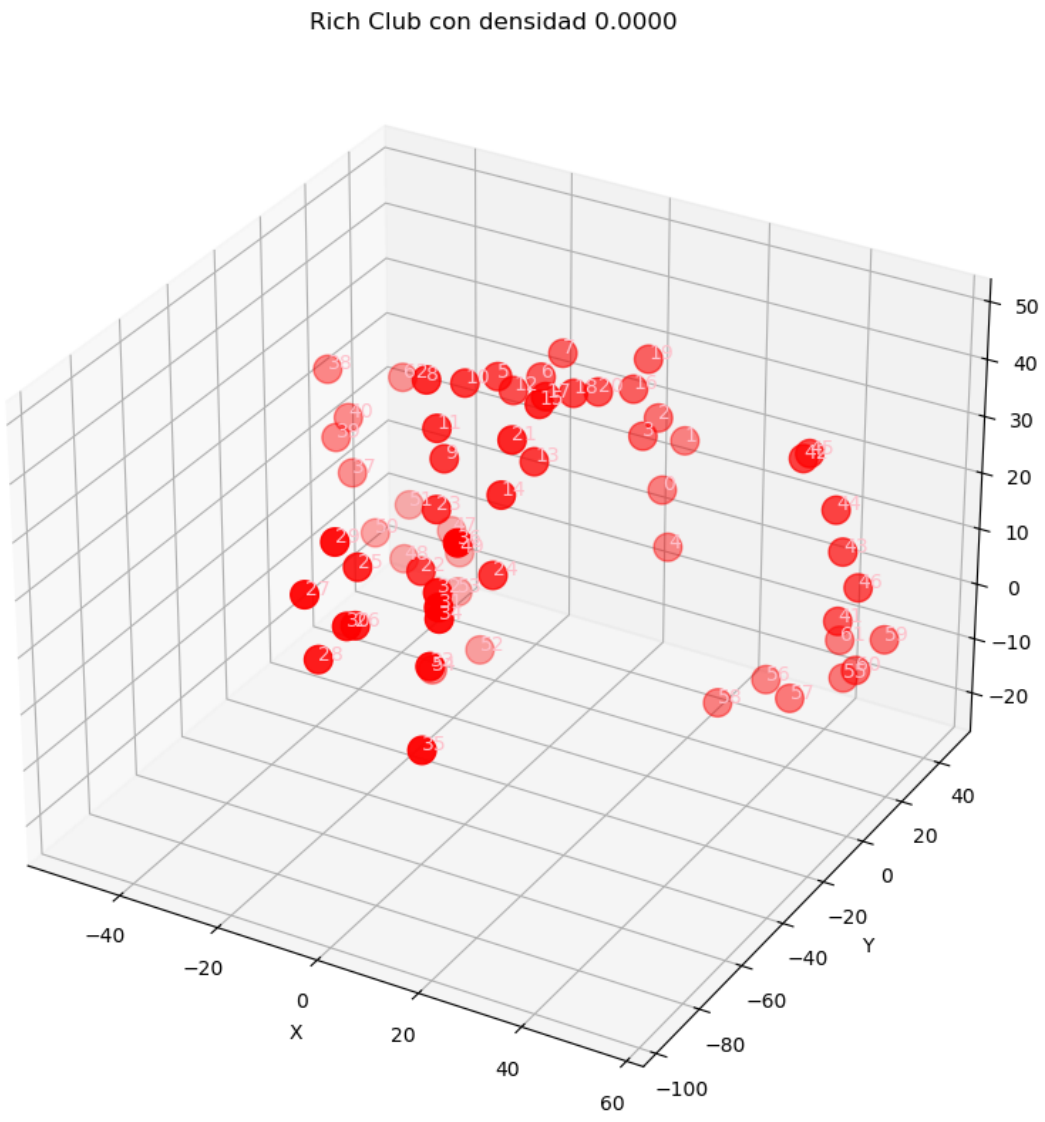
20: 20, 21: 21, 22: 22, 23: 23, 24: 24, 25: 25, 26: 26, 27: 27, 28: 28, 29: 29,
30: 30, 31: 31, 32: 32, 33: 33, 34: 34, 35: 35, 36: 36, 37: 37, 38: 38, 39: 39,
40: 40, 41: 41, 42: 42, 43: 43, 44: 44, 45: 45, 46: 46, 47: 47, 48: 48, 49: 49,
50: 50, 51: 51, 52: 52, 53: 53, 54: 54, 55: 55, 56: 56, 57: 57,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

```
420: 420, 421: 421, 422: 422, 423: 423, 424: 424, 425: 425, 426: 426, 427: 427,
428: 428, 429: 429, 430: 430, 431: 431, 432: 432, 433: 433, 434: 434, 435: 435,
436: 436, 437: 437, 438: 438, 439: 439, 440: 440, 441: 441, 442: 442, 443: 443,
444: 444, 445: 445, 446: 446, 447: 447, 448: 448, 449: 449, 450: 450, 451: 451,
452: 452, 453: 453, 454: 454, 455: 455, 456: 456, 457: 457, 458: 458, 459: 459,
460: 460, 461: 461, 462: 462, 463: 463, 464: 464, 465: 465, 466: 466, 467: 467,
468: 468, 469: 469, 470: 470, 471: 471, 472: 472, 473: 473, 474: 474, 475: 475,
476: 476, 477: 477, 478: 478, 479: 479, 480: 480, 481: 481, 482: 482, 483: 483,
484: 484, 485: 485, 486: 486, 487: 487, 488: 488, 489: 489, 490: 490, 491: 491,
492: 492, 493: 493, 494: 494, 495: 495, 496: 496, 497: 497, 498: 498, 499: 499,
500: 500, 501: 501, 502: 502, 503: 503, 504: 504, 505: 505, 506: 506, 507: 507,
508: 508, 509: 509, 510: 510, 511: 511, 512: 512, 513: 513, 514: 514, 515: 515,
516: 516, 517: 517, 518: 518, 519: 519, 520: 520, 521: 521, 522: 522, 523: 523,
524: 524, 525: 525, 526: 526, 527: 527, 528: 528, 529: 529, 530: 530, 531: 531,
532: 532, 533: 533, 534: 534, 535: 535, 536: 536, 537: 537, 538: 538, 539: 539,
540: 540, 541: 541, 542: 542, 543: 543, 544: 544, 545: 545, 546: 546, 547: 547,
548: 548, 549: 549, 550: 550, 551: 551, 552: 552, 553: 553, 554: 554, 555: 555,
556: 556, 557: 557, 558: 558, 559: 559, 560: 560, 561: 561, 562: 562, 563: 563,
564: 564, 565: 565, 566: 566, 567: 567, 568: 568, 569: 569, 570: 570, 571: 571,
572: 572, 573: 573, 574: 574, 575: 575, 576: 576, 577: 577, 578: 578, 579: 579,
580: 580, 581: 581, 582: 582, 583: 583, 584: 584, 585: 585, 586: 586, 587: 587,
588: 588, 589: 589, 590: 590, 591: 591, 592: 592, 593: 593, 594: 594, 595: 595,
596: 596, 597: 597, 598: 598, 599: 599, 600: 600, 601: 601, 602: 602, 603: 603,
604: 604, 605: 605, 606: 606, 607: 607, 608: 608, 609: 609, 610: 610, 611: 611,
612: 612, 613: 613, 614: 614, 615: 615, 616: 616, 617: 617, 618: 618, 619: 619,
620: 620, 621: 621, 622: 622, 623: 623, 624: 624, 625: 625, 626: 626, 627: 627,
628: 628, 629: 629, 630: 630, 631: 631, 632: 632, 633: 633, 634: 634, 635: 635,
636: 636, 637: 637}
```

[70]:
```python
grado = dict(G.degree())

sorted_nodes = sorted(grado, key=grado.get, reverse=True)

rich_club_threshold = int(0.1 * len(sorted_nodes))

rich_club_nodes = sorted_nodes[:rich_club_threshold]

rich_club_subgraph = G.subgraph(rich_club_nodes)

density_rich_club = nx.density(rich_club_subgraph)

print(f"Nodos del Rich Club: {rich_club_nodes}")
print(f"Densidad del Rich Club: {density_rich_club:.4f}")

pos = nx.get_node_attributes(G, 'pos')
x_vals = [pos[node][0] for node in rich_club_nodes]
y_vals = [pos[node][1] for node in rich_club_nodes]
```

```python
z_vals = [pos[node][2] for node in rich_club_nodes]

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

for (i, j) in rich_club_subgraph.edges():
    x_vals_edge = [pos[i][0], pos[j][0]]
    y_vals_edge = [pos[i][1], pos[j][1]]
    z_vals_edge = [pos[i][2], pos[j][2]]
    ax.plot(x_vals_edge, y_vals_edge, z_vals_edge, color='b', alpha=0.5)

ax.scatter(x_vals, y_vals, z_vals, c='r', marker='o', s=200)

for i, (x, y, z) in enumerate(zip(x_vals, y_vals, z_vals)):
    ax.text(x, y, z, str(rich_club_nodes[i]), color='pink', fontsize=10)

ax.set_title(f'Rich Club con densidad {density_rich_club:.4f}')
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')

plt.show()

# Explicación implicaciones anatómicas y funcionales.
# El Rich Club en el cerebro humano se refiere a un conjunto de nodos altamente␣
 ↪interconectados, generalmente áreas cerebrales clave para la integración de␣
 ↪información.
# Anatómicamente, estos nodos suelen estar ubicados en regiones centrales, como␣
 ↪la corteza prefrontal, que permiten una comunicación eficiente y rápida␣
 ↪entre diferentes áreas del cerebro.
# Funcionalmente, el Rich Club está relacionado con funciones cognitivas␣
 ↪complejas como la toma de decisiones, el lenguaje y la memoria.
# Además, su alta conectividad proporciona resiliencia ante lesiones o␣
 ↪patologías neurológicas, ya que sus redundantes conexiones permiten mantener␣
 ↪la funcionalidad cerebral incluso cuando algunas áreas se ven afectadas.
```

```
Nodos del Rich Club: [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, 58, 59, 60, 61, 62]
Densidad del Rich Club: 0.0000
```

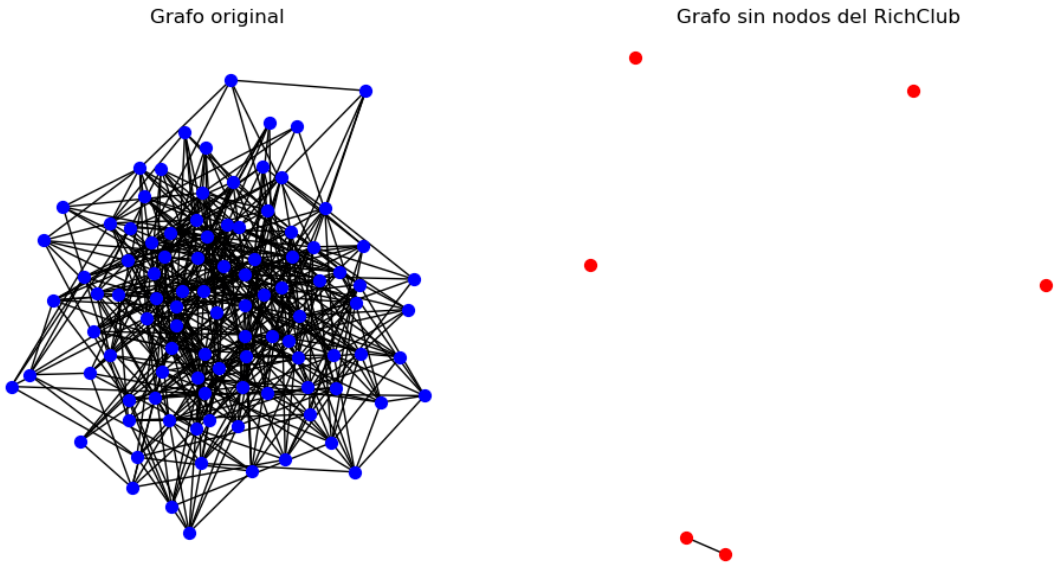


[3]:
```python
G = nx.erdos_renyi_graph(100, 0.1)

degrees = dict(G.degree())

rich_club_nodes = [node for node, degree in degrees.items() if degree > 5]

G_copy = G.copy()

G_copy.remove_nodes_from(rich_club_nodes)

def calculate_topological_properties(G):
    degree = np.mean([d for _, d in G.degree()])
```

```
    clustering_coefficient = nx.average_clustering(G)

    if nx.is_connected(G):
        small_world_coefficient = nx.average_shortest_path_length(G)
    else:
        small_world_coefficient = np.mean(
            [nx.average_shortest_path_length(G.subgraph(c)) for c in nx.
 ↪connected_components(G)]
        )

    centrality_closeness = np.mean(list(nx.closeness_centrality(G).values()))
    centrality_betweenness = np.mean(list(nx.betweenness_centrality(G).
 ↪values()))

    return degree, clustering_coefficient, small_world_coefficient,␣
 ↪centrality_closeness, centrality_betweenness

degree_before, clustering_before, small_world_before, closeness_before,␣
 ↪betweenness_before = calculate_topological_properties(G)

degree_after, clustering_after, small_world_after, closeness_after,␣
 ↪betweenness_after = calculate_topological_properties(G_copy)

print("Propiedades antes de eliminar el RichClub:")
print(f"Grado promedio: {degree_before}")
print(f"Coeficiente de agrupamiento: {clustering_before}")
print(f"Coeficiente de mundo pequeño: {small_world_before}")
print(f"Centralidad de cercanía: {closeness_before}")
print(f"Centralidad de intermediación: {betweenness_before}")

print("\nPropiedades después de eliminar el RichClub:")
print(f"Grado promedio: {degree_after}")
print(f"Coeficiente de agrupamiento: {clustering_after}")
print(f"Coeficiente de mundo pequeño: {small_world_after}")
print(f"Centralidad de cercanía: {closeness_after}")
print(f"Centralidad de intermediación: {betweenness_after}")

# Visualizar los grafos antes y después de la eliminación
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
nx.draw(G, node_size=50, with_labels=False, node_color='blue')
plt.title("Grafo original")

plt.subplot(1, 2, 2)
nx.draw(G_copy, node_size=50, with_labels=False, node_color='red')
plt.title("Grafo sin nodos del RichClub")
```

```
plt.show()
```

```
Propiedades antes de eliminar el RichClub:
Grado promedio: 10.3
Coeficiente de agrupamiento: 0.1068504714240008
Coeficiente de mundo pequeño: 2.2022222222222223
Centralidad de cercanía: 0.45606373737400135
Centralidad de intermediación: 0.012267573696145125

Propiedades después de eliminar el RichClub:
Grado promedio: 0.3333333333333333
Coeficiente de agrupamiento: 0.0
Coeficiente de mundo pequeño: 0.2
Centralidad de cercanía: 0.06666666666666667
Centralidad de intermediación: 0.0
```

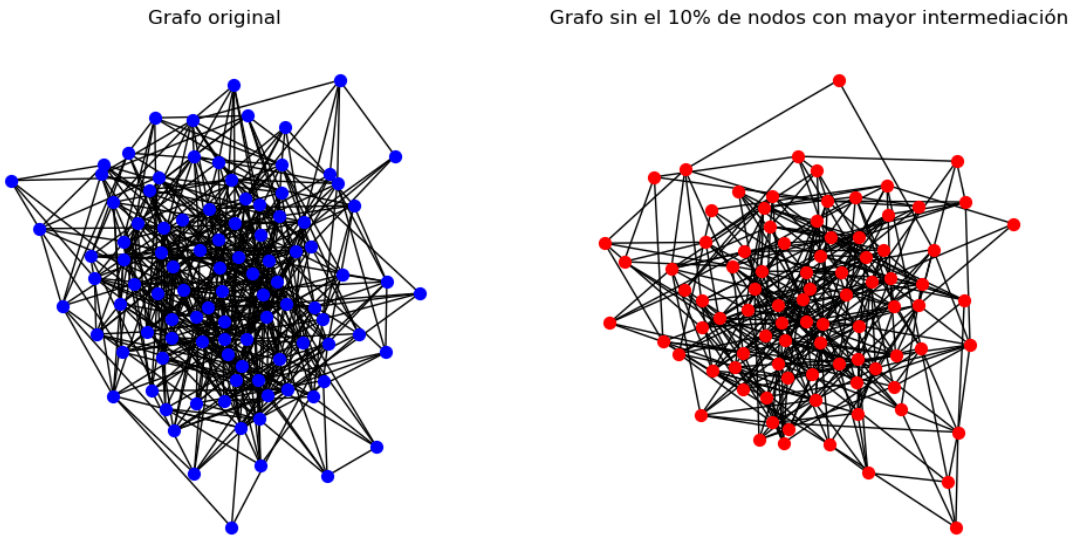


[5]:
```
G = nx.erdos_renyi_graph(100, 0.1)

def calculate_topological_properties(G):
    degree = np.mean([d for _, d in G.degree()])
    clustering_coefficient = nx.average_clustering(G)

    if nx.is_connected(G):
        small_world_coefficient = nx.average_shortest_path_length(G)
    else:
        small_world_coefficient = np.mean(
```

```
            [nx.average_shortest_path_length(G.subgraph(c)) for c in nx.
 ↪connected_components(G)]
        )

    centrality_closeness = np.mean(list(nx.closeness_centrality(G).values()))
    centrality_betweenness = np.mean(list(nx.betweenness_centrality(G).
 ↪values()))

    return degree, clustering_coefficient, small_world_coefficient,␣
 ↪centrality_closeness, centrality_betweenness

betweenness_centrality = nx.betweenness_centrality(G)

sorted_nodes_by_betweenness = sorted(betweenness_centrality.items(), key=lambda␣
 ↪item: item[1], reverse=True)

percentage_to_remove = 0.1

num_nodes_to_remove = int(len(sorted_nodes_by_betweenness) *␣
 ↪percentage_to_remove)

nodes_to_remove = [node for node, _ in sorted_nodes_by_betweenness[:
 ↪num_nodes_to_remove]]

G_copy = G.copy()

G_copy.remove_nodes_from(nodes_to_remove)

degree_before, clustering_before, small_world_before, closeness_before,␣
 ↪betweenness_before = calculate_topological_properties(G)

degree_after, clustering_after, small_world_after, closeness_after,␣
 ↪betweenness_after = calculate_topological_properties(G_copy)

print("Propiedades antes de eliminar los nodos con mayor intermediación:")
print(f"Grado promedio: {degree_before}")
print(f"Coeficiente de agrupamiento: {clustering_before}")
print(f"Coeficiente de mundo pequeño: {small_world_before}")
print(f"Centralidad de cercanía: {closeness_before}")
print(f"Centralidad de intermediación: {betweenness_before}")

print("\nPropiedades después de eliminar los nodos con mayor intermediación:")
print(f"Grado promedio: {degree_after}")
print(f"Coeficiente de agrupamiento: {clustering_after}")
print(f"Coeficiente de mundo pequeño: {small_world_after}")
print(f"Centralidad de cercanía: {closeness_after}")
```

```
print(f"Centralidad de intermediación: {betweenness_after}")

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

plt.subplot(1, 2, 1)
nx.draw(G, node_size=50, with_labels=False, node_color='blue')
plt.title("Grafo original")

plt.subplot(1, 2, 2)
nx.draw(G_copy, node_size=50, with_labels=False, node_color='red')
plt.title(f"Grafo sin el {int(percentage_to_remove * 100)}% de nodos con mayor␣
 ↪intermediación")

plt.show()
```

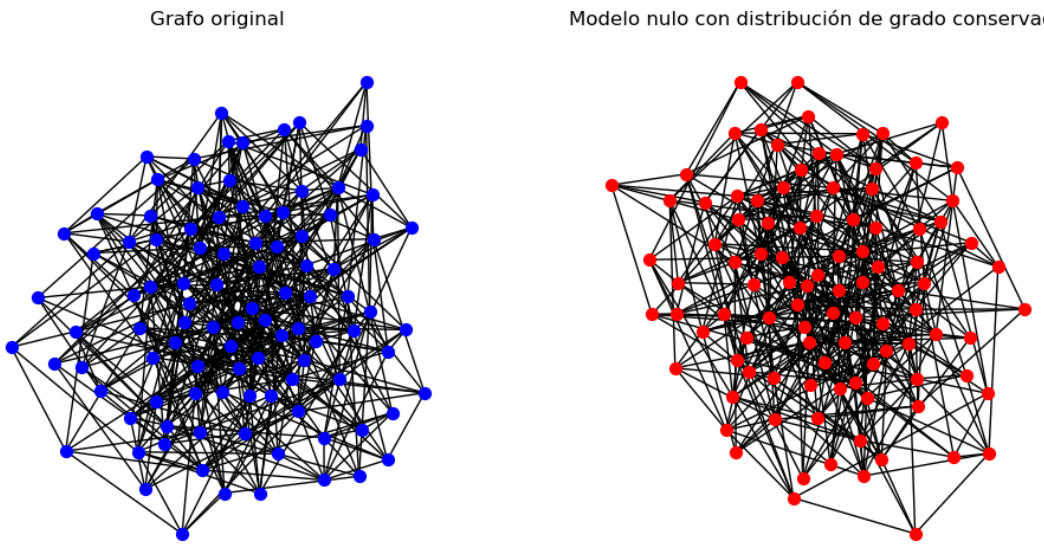
```
Propiedades antes de eliminar los nodos con mayor intermediación:
Grado promedio: 10.28
Coeficiente de agrupamiento: 0.09826262789498079
Coeficiente de mundo pequeño: 2.2155555555555555
Centralidad de cercanía: 0.45343957790206135
Centralidad de intermediación: 0.012403628117913831

Propiedades después de eliminar los nodos con mayor intermediación:
Grado promedio: 8.2
Coeficiente de agrupamiento: 0.09378516545183206
Coeficiente de mundo pequeño: 2.366791510611735
Centralidad de cercanía: 0.42489011344166705
Centralidad de intermediación: 0.01553172171149699
```



[11]:
```
import networkx as nx
import numpy as np
import matplotlib.pyplot as plt

G = nx.erdos_renyi_graph(100, 0.1)

num_edges_original = G.number_of_edges()

degree_sequence = [d for _, d in G.degree()]
G_null = nx.configuration_model(degree_sequence)

G_null = nx.Graph(G_null)

G_null.remove_edges_from([(u, v) for u, v in G_null.edges() if u == v])

def calculate_topological_properties(G):
    degree = np.mean([d for _, d in G.degree()])
    clustering_coefficient = nx.average_clustering(G)

    if nx.is_connected(G):
        small_world_coefficient = nx.average_shortest_path_length(G)
    else:
        small_world_coefficient = np.mean(
            [nx.average_shortest_path_length(G.subgraph(c)) for c in nx.
 ↪connected_components(G)]
        )

    centrality_closeness = np.mean(list(nx.closeness_centrality(G).values()))
    centrality_betweenness = np.mean(list(nx.betweenness_centrality(G).
 ↪values()))

    return degree, clustering_coefficient, small_world_coefficient,␣
 ↪centrality_closeness, centrality_betweenness

degree_original, clustering_original, small_world_original, closeness_original,␣
 ↪betweenness_original = calculate_topological_properties(G)

degree_null, clustering_null, small_world_null, closeness_null,␣
 ↪betweenness_null = calculate_topological_properties(G_null)

print("Propiedades del grafo original:")
print(f"Grado promedio: {degree_original}")
print(f"Coeficiente de agrupamiento: {clustering_original}")
print(f"Coeficiente de mundo pequeño: {small_world_original}")
print(f"Centralidad de cercanía: {closeness_original}")
print(f"Centralidad de intermediación: {betweenness_original}")
```

```python
print("\nPropiedades del modelo nulo con distribución de grado conservada:")
print(f"Grado promedio: {degree_null}")
print(f"Coeficiente de agrupamiento: {clustering_null}")
print(f"Coeficiente de mundo pequeño: {small_world_null}")
print(f"Centralidad de cercanía: {closeness_null}")
print(f"Centralidad de intermediación: {betweenness_null}")

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

plt.subplot(1, 2, 1)
nx.draw(G, node_size=50, with_labels=False, node_color='blue')
plt.title("Grafo original")

plt.subplot(1, 2, 2)
nx.draw(G_null, node_size=50, with_labels=False, node_color='red')
plt.title("Modelo nulo con distribución de grado conservada")

plt.show()
```

```
Propiedades del grafo original:
Grado promedio: 10.14
Coeficiente de agrupamiento: 0.11128045724485344
Coeficiente de mundo pequeño: 2.2185858585858584
Centralidad de cercanía: 0.45252837082962627
Centralidad de intermediación: 0.01243454957740672

Propiedades del modelo nulo con distribución de grado conservada:
Grado promedio: 9.56
Coeficiente de agrupamiento: 0.10541835941835938
Coeficiente de mundo pequeño: 2.2707070707070707
Centralidad de cercanía: 0.4420108633305969
Centralidad de intermediación: 0.012966398680684393
```

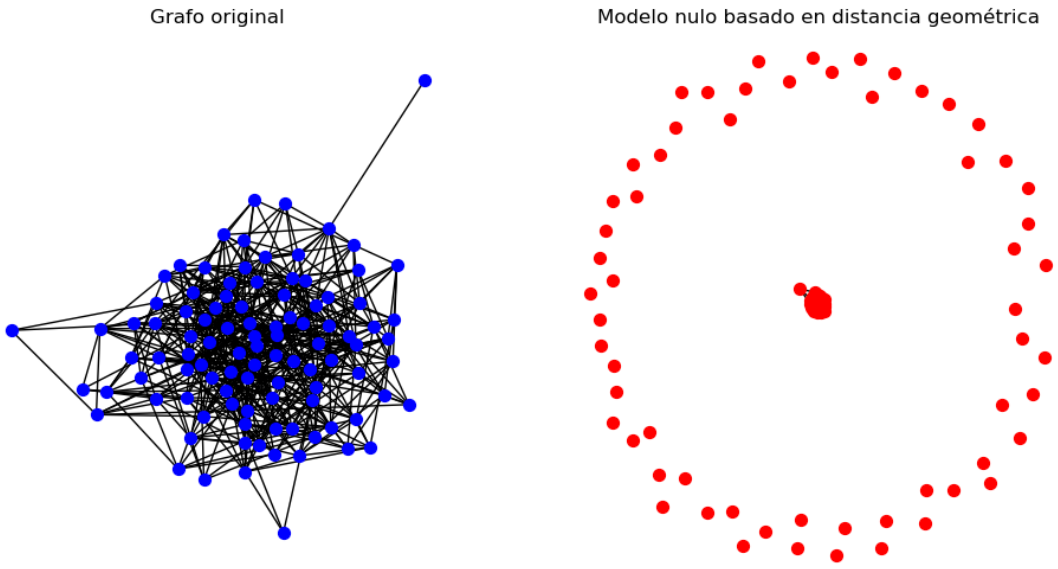


[13]:
```
G = nx.erdos_renyi_graph(100, 0.1)

num_edges_original = G.number_of_edges()

positions = np.random.rand(len(G.nodes()), 2)

def calculate_distances(positions):
    distances = np.linalg.norm(positions[:, np.newaxis] - positions, axis=2)
    np.fill_diagonal(distances, 0)
    return distances

distances = calculate_distances(positions)

def connection_probability(distance, alpha=1):
    return 1 / (1 + alpha * distance)

edges_null = []
for i in range(len(G.nodes())):
    for j in range(i+1, len(G.nodes())):
        p = connection_probability(distances[i, j])
        if np.random.rand() < p:
            edges_null.append((i, j))

G_null = nx.Graph()
G_null.add_nodes_from(G.nodes())
G_null.add_edges_from(edges_null)
```

```
while len(G_null.edges()) > num_edges_original:
    G_null.remove_edge(*list(G_null.edges())[0])

def calculate_topological_properties(G):
    degree = np.mean([d for _, d in G.degree()])
    clustering_coefficient = nx.average_clustering(G)

    if nx.is_connected(G):
        small_world_coefficient = nx.average_shortest_path_length(G)
    else:
        small_world_coefficient = np.mean(
            [nx.average_shortest_path_length(G.subgraph(c)) for c in nx.
 ↪connected_components(G)]
        )

    centrality_closeness = np.mean(list(nx.closeness_centrality(G).values()))
    centrality_betweenness = np.mean(list(nx.betweenness_centrality(G).
 ↪values()))

    return degree, clustering_coefficient, small_world_coefficient,␣
 ↪centrality_closeness, centrality_betweenness

degree_original, clustering_original, small_world_original, closeness_original,␣
 ↪betweenness_original = calculate_topological_properties(G)

degree_null, clustering_null, small_world_null, closeness_null,␣
 ↪betweenness_null = calculate_topological_properties(G_null)

print("Propiedades del grafo original:")
print(f"Grado promedio: {degree_original}")
print(f"Coeficiente de agrupamiento: {clustering_original}")
print(f"Coeficiente de mundo pequeño: {small_world_original}")
print(f"Centralidad de cercanía: {closeness_original}")
print(f"Centralidad de intermediación: {betweenness_original}")

print("\nPropiedades del modelo nulo basado en la distancia geométrica:")
print(f"Grado promedio: {degree_null}")
print(f"Coeficiente de agrupamiento: {clustering_null}")
print(f"Coeficiente de mundo pequeño: {small_world_null}")
print(f"Centralidad de cercanía: {closeness_null}")
print(f"Centralidad de intermediación: {betweenness_null}")

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

plt.subplot(1, 2, 1)
nx.draw(G, node_size=50, with_labels=False, node_color='blue')
plt.title("Grafo original")
```

```
plt.subplot(1, 2, 2)
nx.draw(G_null, node_size=50, with_labels=False, node_color='red')
plt.title("Modelo nulo basado en distancia geométrica")

plt.show()
```

```
Propiedades del grafo original:
Grado promedio: 10.12
Coeficiente de agrupamiento: 0.10226380525219528
Coeficiente de mundo pequeño: 2.2284848484848485
Centralidad de cercanía: 0.4511285528619114
Centralidad de intermediación: 0.012535559678416823

Propiedades del modelo nulo basado en la distancia geométrica:
Grado promedio: 10.12
Coeficiente de agrupamiento: 0.2662006935486566
Coeficiente de mundo pequeño: 0.022173182009247585
Centralidad de cercanía: 0.11730885288549693
Centralidad de intermediación: 0.0005668934240362812
```



[ ]:
```
# Este curso fue bastante interesante en el sentido de que mayoritariamente yo
 ↪desconocía
# como era realizar programación, hacer gráficas fuera de excel o cuestiones
 ↪más sencillas,
# aquí pudimos observar conexiones entre nodos, distribución de estos,
 ↪coeficiente de cluster, incluso como estan relacionados
```

```
# de forma dinámica, hacer comparaciones de gráficos y nodos, en fnción de␣
 ↪neurociencias aplica para las conexiones entre
# neuronas, glía, arborizaciones dendriticas, etc.
```