Análisis de grafos en datos de coactivación cerebral

Este notebook contiene múltiples ejercicios que procesan una matriz de coactivación y sus coordenadas asociadas utilizando técnicas de análisis de grafos.

```
import scipy.io as sio
import numpy as np
import networkx as nx
from networkx.algorithms.components import connected_components
from networkx.algorithms.smallworld import sigma
from networkx.algorithms.centrality import closeness_centrality,
betweenness_centrality
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.animation import FuncAnimation
from IPython.display import Image
from sklearn.cluster import KMeans
```

Ejercicio 1: Visualización 3D del grafo para diferentes umbrales

```
def ejercicio1(coactivation_matrix, coordinates):
    def plot_graph_3d(graph, coords, threshold):
        fig = plt.figure(figsize=(10, 7))
        ax = fig.add_subplot(111, projection='3d')

        ax.scatter(coords[:, 0], coords[:, 1], coords[:, 2], c='blue',
s=50, label='Nodes')

        for i, j in graph.edges():
            x = [coords[i, 0], coords[j, 0]]
            y = [coords[i, 1], coords[j, 1]]
            z = [coords[i, 2], coords[j, 2]]
            ax.plot(x, y, z, c='black', alpha=0.7)

        ax.set_title(f'Graph with Threshold {threshold}')
        ax.set_xlabel('X')
        ax.set_ylabel('Y')
        ax.set_zlabel('Z')
        plt.legend()
        plt.show()
```

```
    thresholds = [0.08, 0.09, 0.1]

    for threshold in thresholds:
        graph = nx.Graph()
        size = coactivation_matrix.shape[0]

        graph.add_nodes_from(range(size))

        for i in range(size):
            for j in range(i + 1, size):
                if coactivation_matrix[i, j] > threshold:
                    graph.add_edge(i, j)

        plot_graph_3d(graph, coordinates, threshold)
```

Ejercicio 2: Animación del grafo en rotación

```
def ejercicio2(coactivation_matrix, coordinates):
    threshold = 0.09
    graph = nx.Graph()
    size = coactivation_matrix.shape[0]

    graph.add_nodes_from(range(size))

    for i in range(size):
        for j in range(i + 1, size):
            if coactivation_matrix[i, j] > threshold:
                graph.add_edge(i, j)

    def update(frame):
        ax.clear()
        ax.scatter(coordinates[:, 0], coordinates[:, 1],
coordinates[:, 2], c='blue', s=50, label='Nodes')

        for i, j in graph.edges():
            x = [coordinates[i, 0], coordinates[j, 0]]
            y = [coordinates[i, 1], coordinates[j, 1]]
            z = [coordinates[i, 2], coordinates[j, 2]]
            ax.plot(x, y, z, c='black', alpha=0.7)

        ax.view_init(30, frame)
        ax.set_title(f'Graph with Threshold {threshold}')
        ax.set_xlabel('X')
        ax.set_ylabel('Y')
        ax.set_zlabel('Z')
        plt.legend()

    fig = plt.figure(figsize=(10, 7))
    ax = fig.add_subplot(111, projection='3d')
```

```
    ani = FuncAnimation(fig, update, frames=90, interval=25)
    ani.save('rotating_graph.gif', writer='pillow', fps=30)
    Image('rotating_graph.gif')
```

Ejercicio 3: Identificación de hubs en el grafo

```
def ejercicio3(coactivation_matrix, coordinates):
    threshold = 0.09
    graph = nx.Graph()
    size = coactivation_matrix.shape[0]

    graph.add_nodes_from(range(size))

    for i in range(size):
        for j in range(i + 1, size):
            if coactivation_matrix[i, j] > threshold:
                graph.add_edge(i, j)

    node_degrees = dict(graph.degree())

    max_size = 300
    node_sizes = [max_size * (node_degrees[node] /
max(node_degrees.values())) for node in graph.nodes]

    def plot_hub_graph(graph, coords, node_sizes):
        fig = plt.figure(figsize=(10, 7))
        ax = fig.add_subplot(111, projection='3d')

        ax.scatter(coords[:, 0], coords[:, 1], coords[:, 2],
s=node_sizes, c='blue', alpha=0.8, label='Nodes')

        for i, j in graph.edges():
            x = [coords[i, 0], coords[j, 0]]
            y = [coords[i, 1], coords[j, 1]]
            z = [coords[i, 2], coords[j, 2]]
            ax.plot(x, y, z, c='black', alpha=0.6)

        ax.set_title(f'Graph with Hubs Highlighted (Threshold
{threshold})')
        ax.set_xlabel('X')
        ax.set_ylabel('Y')
        ax.set_zlabel('Z')
        plt.legend()
        plt.show()

    plot_hub_graph(graph, coordinates, node_sizes)
```

Ejercicio 4: Partición del grafo en módulos usando K-means

```
def ejercicio4(coactivation_matrix, coordinates):
    threshold = 0.09
    graph = nx.Graph()
    size = coactivation_matrix.shape[0]

    graph.add_nodes_from(range(size))

    for i in range(size):
        for j in range(i + 1, size):
            if coactivation_matrix[i, j] > threshold:
                graph.add_edge(i, j)

    adjacency_matrix = nx.adjacency_matrix(graph).todense()
    matching_matrix = np.corrcoef(adjacency_matrix)
    matching_matrix = np.nan_to_num(matching_matrix)

    n_modules = 3
    kmeans = KMeans(n_clusters=n_modules, random_state=42)
    node_labels = kmeans.fit_predict(matching_matrix)

    def plot_modules(graph, coords, node_labels, n_modules):
        fig = plt.figure(figsize=(10, 7))
        ax = fig.add_subplot(111, projection='3d')

        colors = plt.cm.tab10(np.linspace(0, 1, n_modules))

        for module in range(n_modules):
            nodes_in_module = [i for i in range(len(node_labels)) if
node_labels[i] == module]
            ax.scatter(coords[nodes_in_module, 0],
coords[nodes_in_module, 1], coords[nodes_in_module, 2],
                       c=[colors[module]], label=f'Module {module+1}',
s=50)

        for i, j in graph.edges():
            x = [coords[i, 0], coords[j, 0]]
            y = [coords[i, 1], coords[j, 1]]
            z = [coords[i, 2], coords[j, 2]]
            ax.plot(x, y, z, c='black', alpha=0.3)

        ax.set_title(f'Graph Partitioned into {n_modules} Modules')
        ax.set_xlabel('X')
        ax.set_ylabel('Y')
        ax.set_zlabel('Z')
        plt.legend()
```

```
        plt.show()

    plot_modules(graph, coordinates, node_labels, n_modules)
    return graph, coordinates
```

Ejercicio 5-7: Análisis de Rich Club, propiedades y eliminación de nodos

```
# Ejercicios adicionales omiten sus funciones aquí pero se implementan de forma similar.
# Agrega las siguientes funciones (ejercicio5, ejercicio6, ejercicio7) según necesidad.
```

Main: Carga de datos y ejecución de ejercicios

```
def main():
    mat = sio.loadmat('Coactivation_matrix (3).mat')
    coactivation_matrix = mat['Coactivation_matrix']
    coordinates = mat['Coord']

    # Ejemplo de ejecución
    # ejercicio1(coactivation_matrix, coordinates)
    graph, coordinates = ejercicio4(coactivation_matrix, coordinates)
    # ejercicio7(graph)

if __name__ == '__main__':
    main()
```