```
# Project 13: Social networks, clustering, and eigenvalue problems
import numpy as np
import matplotlib.pyplot as plt
import scipy.io
from scipy.linalg import eigh
from scipy.io import loadmat


print("\nSubtask 1\n")
# 1. Simple graph: Define adjacency matrix
AdjMatrix = np.array([[0, 1, 1, 0],
[1, 0, 0, 1],
[1, 0, 0, 1],
[0, 1, 1, 0]])
print("Adjacency Matrix:")
print(AdjMatrix)
```

```
Subtask 1

Adjacency Matrix:
[[0 1 1 0]
 [1 0 0 1]
 [1 0 0 1]
 [0 1 1 0]]
```

```
print("\nSubtask 2\n")
# 2. Find the row sums of the matrix AdjMatrix
RowSums = np.sum(AdjMatrix, axis=1)
print("\nRow Sums:")
print(RowSums)
```

```
Subtask 2


Row Sums:
[2 2 2 2]
```

```
print("\nSubtask 3\n")
# 3. Compute the Laplacian of the graph
LaplaceGraph = np.diag(RowSums) - AdjMatrix
print("\nLaplacian Matrix:")
print(LaplaceGraph)
# Check if LaplaceGraph is singular
test_vector = np.ones(len(LaplaceGraph))
singularity_check = LaplaceGraph @ test_vector
print("\nSingularity Check (Laplacian * ones):")
print(singularity_check)
```

```
Subtask 3


Laplacian Matrix:
[[ 2 -1 -1  0]
 [-1  2  0 -1]
 [-1  0  2 -1]
 [ 0 -1 -1  2]]

Singularity Check (Laplacian * ones):
[0. 0. 0. 0.]
```

```
print("\nSubtask 4\n")
# 4. Find eigenvalues and eigenvectors using the eig function
D, V = np.linalg.eig(LaplaceGraph)
```

```
Subtask 4
```

```
print("\nSubtask 5\n")
d, ind = np.argsort(D), np.argsort(D)
D = np.diag(D[ind])
V = V[:, ind]
print("\nEigenvalues (sorted):")
print(np.diag(D))
print("\nEigenvectors (sorted):")
print(V)
```

```
Subtask 5


Eigenvalues (sorted):
[-2.22044605e-16  2.00000000e+00  2.00000000e+00  4.00000000e+00]

Eigenvectors (sorted):
[[ 5.00000000e-01  4.08248290e-01  7.07106781e-01 -5.00000000e-01]
 [ 5.00000000e-01 -5.77350269e-01  4.80181756e-16  5.00000000e-01]
 [ 5.00000000e-01  5.77350269e-01 -1.77321568e-16  5.00000000e-01]
 [ 5.00000000e-01 -4.08248290e-01 -7.07106781e-01 -5.00000000e-01]]
```

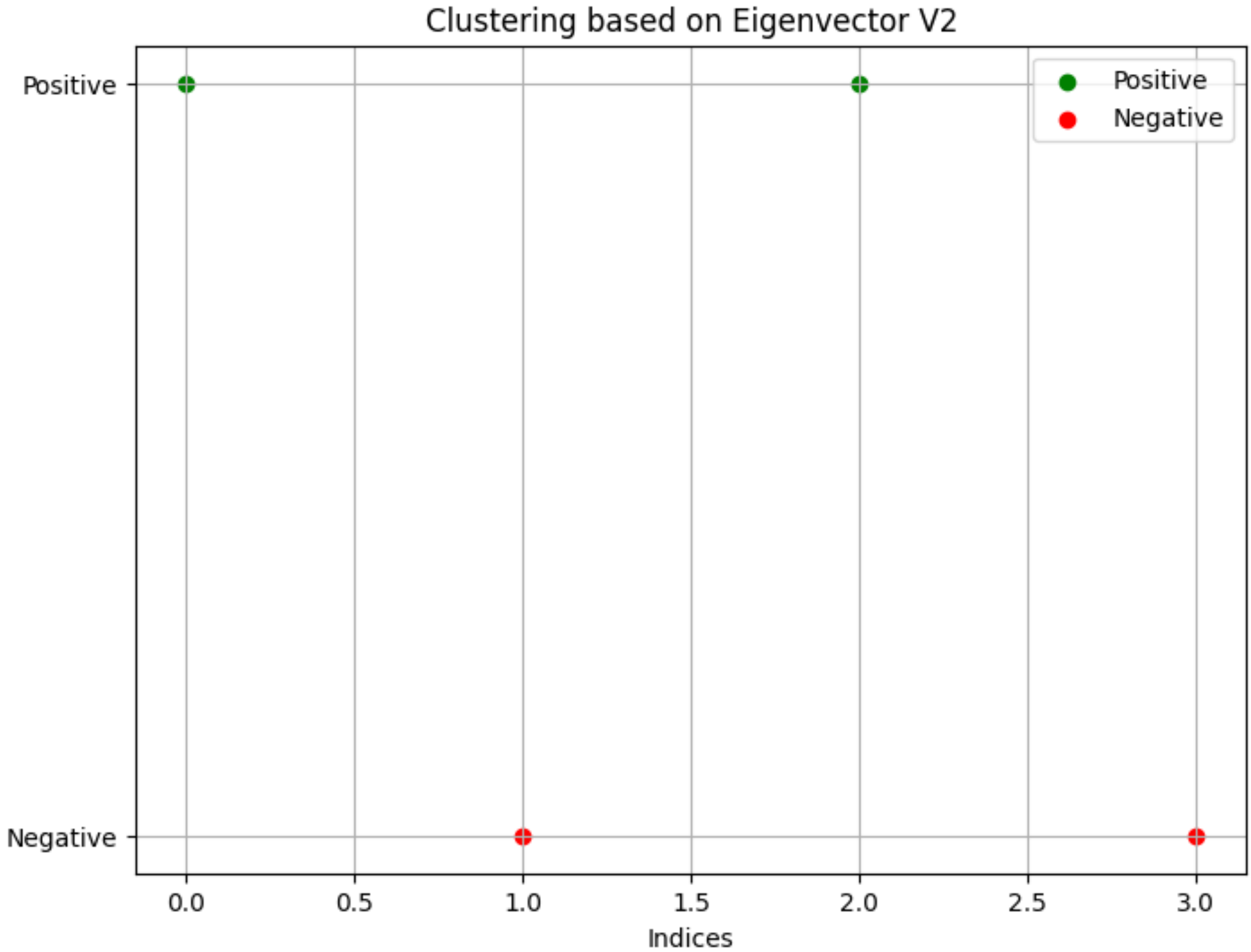
```
print("\nSubtask 6\n")
# 6. Identify the second smallest eigenvalue and its corresponding eigenvector
second_smallest_eigenvalue = D[1, 1]
V2 = V[:, 1]
# Ensure V2 has a positive first entry
if V2[0] < 0:
    V2 = -V2
print("\nSecond Smallest Eigenvalue:")
print(second_smallest_eigenvalue)
print("\nEigenvector corresponding to the second smallest eigenvalue (V2):")
print(V2)
```

```
Subtask 6


Second Smallest Eigenvalue:
1.9999999999999991

Eigenvector corresponding to the second smallest eigenvalue (V2):
[ 0.40824829 -0.57735027  0.57735027 -0.40824829]
```

```
print("\nSubtask 7\n")
# 7. Separate the elements of the eigenvector V2
pos = []
neg = []
```

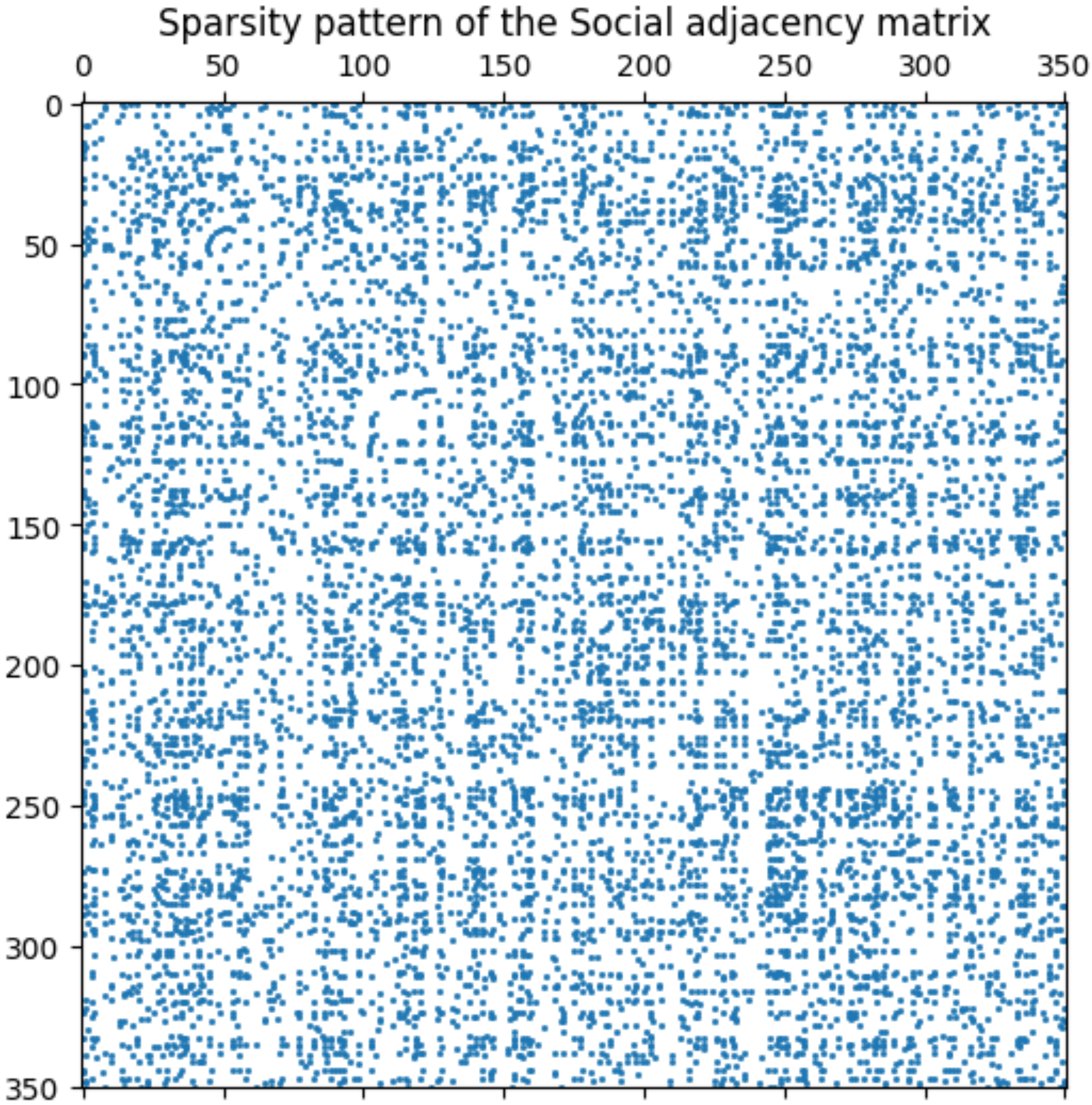
```
for j in range(len(V2)):
    if V2[j] > 0:
        pos.append(j)
    else:
        neg.append(j)
print("\nPositive Indices (V2 > 0):")
print(pos)
print("\nNegative Indices (V2 <= 0):")
print(neg)
# Optional: Visualize the clusters
plt.figure(figsize=(8, 6))
plt.scatter(pos, [1]*len(pos), color='green', label='Positive')
plt.scatter(neg, [0]*len(neg), color='red', label='Negative')
plt.yticks([0, 1], ['Negative', 'Positive'])
plt.title('Clustering based on Eigenvector V2')
plt.xlabel('Indices')
plt.legend()
plt.grid()
plt.show()
```

```
Subtask 7


Positive Indices (V2 > 0):
[0, 2]

Negative Indices (V2 <= 0):
[1, 3]
```



```
print("\nSubtask 8\n")
# 8. Load the data
data = loadmat("/content/social.mat")
Social = data['Social']
print("Loaded Social adjacency matrix with shape:", Social.shape)
# Spy plot of the Social matrix
plt.figure(figsize=(8, 6))
plt.spy(Social, markersize=1)
plt.title('Sparsity pattern of the Social adjacency matrix')
plt.show()
```

```
Subtask 8

Loaded Social adjacency matrix with shape: (351, 351)
```



```
print("\nSubtask 9\n")
# 9. Define DiagSocial and LaplaceSocial
DiagSocial = np.sum(Social, axis=1)
LaplaceSocial = np.diag(DiagSocial) - Social
print("\nDiagonal matrix DiagSocial:")
print(DiagSocial)
print("\nLaplacian matrix LaplaceSocial:")
print(LaplaceSocial)
```

```
Subtask 9


Diagonal matrix DiagSocial:
[42 19 25 36 50  5  7  2 29  5 28  5  4  8 38 17 47  4 29 49 28 17  7 23
  8 43 25 48 30 20 43 47  6 39 34 46 40 50 27 32 15 46 33 31 24 42 10  8
  6 44 26  5 24 47  6 24 50  7 56 11  4  4 15 24  9 21  7 22 23  5 40 36
  8  5  9  4  4 45  4 23  5 35 46  9  8  7 56 34 49  9 48 42  6 44 33 34
 29 24 50  4  8  4 21 43 28  6 21 26  5  8 23  3 27 51 42 48 18  7 31 50
 46 45 25  4  4  7 22 52 41  4  5 24 16  8 20  3 34  4 46 46 43 27 43 32
  4 42 42  4  5  4 26  7 20  7 51 26 32 48 42 55 48  6  6  7 28  6  7  4
 29 33 19 39  4  6  6 47 32 45 45 22 30 39 19  5 27 44 30 34 43  5  5 48
 19 26 35  7 50 31  6 31  6 34 51  7  6 29 32  5  4 18 29  4  5 38 32 11
 43 24 46 54 34 48  6  5  5 18 49 14 47 22 28 46 46 22  6  7 45  5  5  6
  6 16 23  6 41 43 49 50 40 48 41 44 23 31 27 40 46 52  5  5  5 27 25 44
 47  6  4 36 16 43  8  5  7 44 55 41  3 29 50 34 39 25 44 50  8 47 19  6
 21 46 40 34 22  6 48 28 41 29 18  3  5  6 52 44  8  6  5 18  6 42 44 45
  8  6 25 33 51 20  5 32 41  5 20  5 49 45 15 17 23  4  5  9  7 50 36 51
 46  6 47 33  6 30  5  5 48  9  8 53 21 19 22]

Laplacian matrix LaplaceSocial:
[[                  42                    0                    0 ...
                     0                    0                    0]
 [                   0                   19                    0 ...
  18446744073709551615 18446744073709551615                    0]
 [                   0                    0                   25 ...
                     0                    0                    0]
 ...
 [                   0 18446744073709551615                    0 ...
                    21                    0 18446744073709551615]
 [                   0 18446744073709551615                    0 ...
                     0                   19 18446744073709551615]
 [                   0                    0                    0 ...
  18446744073709551615 18446744073709551615                   22]]
```

```
print("\nSubtask 10\n")
# 10. Compute eigenvalues and eigenvectors
D, V = np.linalg.eig(LaplaceSocial)
print("\nEigenvalues (D):")
print(D)
```

```
print("\nEigenvectors (V):")
print(V)
# Check the shapes
print("Shape of V (eigenvectors):", V.shape)
print("Shape of D (eigenvalues):", D.shape)
d, ind = np.argsort(D), np.argsort(D)
D = np.diag(D[ind])
V = V[:, ind]
print("\nEigenvalues (sorted):")
print(np.diag(D))
print("\nEigenvectors (sorted):")
print(V)
```

```
 -7.93309180e+18 -7.30039076e+18 -6.60029818e+18 -6.19182146e+18
 -5.42242603e+18 -5.09404014e+18 -4.64429621e+18 -4.04194737e+18
 -3.42093068e+18 -2.67495781e+18 -2.22908528e+18 -1.67326547e+18
 -1.35945445e+18 -1.10252185e+18 -7.33223777e+17 -1.50655293e+16
  3.66850571e+17  8.38595118e+17  1.13289697e+18  1.34061746e+18
  2.00955967e+18  2.39555123e+18  2.74286959e+18  3.22248239e+18
  3.69741316e+18  4.26543390e+18  4.78141228e+18  5.66915645e+18
  5.96810177e+18  6.61028109e+18  6.86736139e+18  7.26178903e+18
  7.41826324e+18  8.17536984e+18  8.43627167e+18  8.60545879e+18
  9.28826591e+18  1.02872225e+19  1.14035216e+19  1.16610911e+19
  1.21049666e+19  1.27651677e+19  1.32045922e+19  1.34443840e+19
  1.42145110e+19  1.44559151e+19  1.51891359e+19  1.54114602e+19
  1.62568233e+19  1.66780272e+19  1.70721347e+19  1.76953777e+19
  1.78689735e+19  1.85117454e+19  1.88675545e+19  1.91554788e+19
  2.01182608e+19  2.10114103e+19  2.14614933e+19  2.23804992e+19
  2.31046764e+19  2.35318090e+19  2.39152487e+19  2.45434280e+19
  2.59155943e+19  2.62691177e+19  2.67046286e+19  2.69038645e+19
  2.75392242e+19  2.89546542e+19  2.90832364e+19  2.97080121e+19
  3.01780849e+19  3.04988154e+19  3.07283472e+19  3.12584236e+19
  3.19791971e+19  3.24261364e+19  3.29021359e+19  3.32991321e+19
  3.37137747e+19  3.53635056e+19  3.57735661e+19  3.60231511e+19
  3.66045702e+19  3.74247446e+19  3.76807250e+19  3.85223460e+19
  3.86551930e+19  3.96194703e+19  4.01591963e+19  4.10046768e+19
  4.13572777e+19  4.18206269e+19  4.29225349e+19  4.34763987e+19
  4.37777443e+19  4.45337445e+19  4.49655765e+19  4.61948643e+19
  4.65683594e+19  4.74903802e+19  4.81613153e+19  4.85664149e+19
  4.95902084e+19  5.08369335e+19  5.14957088e+19  5.24100864e+19
  5.36300416e+19  5.43992704e+19  5.56434325e+19  5.61031079e+19
  5.66253756e+19  5.75744520e+19  5.78253400e+19  5.81711581e+19
  5.87009950e+19  5.88330137e+19  5.99211317e+19  6.11436509e+19
  6.21840746e+19  6.27850796e+19  6.36190789e+19  6.43909357e+19
  6.48545829e+19  6.66507944e+19  6.75559282e+19  6.90258886e+19
  6.97450187e+19  6.99694147e+19  7.10174515e+19  7.23210111e+19
  7.34244853e+19  7.41688396e+19  7.51349827e+19  7.71211873e+19
  7.85607743e+19  7.93338019e+19  7.97817119e+19  8.01801141e+19
  8.23006362e+19  8.28036393e+19  8.52640489e+19  8.69930358e+19
  8.82524068e+19  8.88683300e+19  9.01311417e+19  9.18096401e+19
  9.27589905e+19  9.33279231e+19  9.49128050e+19  9.57261108e+19
  9.78420558e+19  9.88690551e+19  1.01403950e+20  1.03063596e+20
```

```
 1.04124596e+20  1.05052168e+20  1.06299208e+20  1.07829813e+20
 1.10547598e+20  1.11356675e+20  1.12100301e+20  1.14938035e+20
 1.16908138e+20  1.18005302e+20  1.22794790e+20  1.23581014e+20
 1.25564038e+20  1.30358148e+20  1.31062981e+20  1.34602091e+20
 1.38083299e+20  1.38683216e+20  1.40489284e+20  1.43213235e+20
 1.47636037e+20  1.53891311e+20  1.64450621e+20  1.65739742e+20
 1.67965098e+20  1.70182927e+20  1.80362825e+20  2.63133350e+20
 3.60858037e+20  4.69452742e+20  7.83726185e+20]

Eigenvectors (sorted):
[[ 0.04876828  0.00286912 -0.05629825 ...  0.0046746   0.01327095
  -0.08348142]
 [ 0.00317129  0.00181178  0.00862197 ... -0.02741646 -0.01983797
  -0.00337962]
 [ 0.00605985  0.00550859 -0.00155231 ... -0.19655827 -0.02129895
  -0.00717926]
 ...
 [ 0.02613452  0.01312016 -0.00454248 ... -0.00699816 -0.02059704
  -0.01086072]
 [ 0.00597751  0.02107703 -0.01764149 ... -0.00769263 -0.01119686
```

```python
print("\nSubtask 11\n")
second_smallest_eigenvalue = D[1, 1]
V2 = V[:, 1]
# Ensure V2 has a positive first entry
if V2[0] < 0:
    V2 = -V2
print("\nSecond Smallest Eigenvalue:")
print(second_smallest_eigenvalue)
print("\nEigenvector corresponding to the second smallest eigenvalue (V2):")
print(V2)
pos = []
neg = []
for j in range(len(V2)):
    if V2[j] > 0:
        pos.append(j)
    else:
        neg.append(j)
print("\nPositive Indices (V2 > 0):")
print(pos)
print("\nNegative Indices (V2 <= 0):")
print(neg)
```

```
-8.00193565e-02  1.67132254e-02  6.58769852e-03  1.90532286e-02
-8.42781828e-03  2.98076253e-02  1.38280567e-01 -2.55031076e-03
 1.43493838e-02  7.13194444e-03 -1.56960824e-02 -8.25623379e-03
-5.28984414e-03 -6.02086042e-03  1.15289001e-01 -3.82802699e-02
 4.21358191e-03 -5.04844412e-02 -6.50681138e-02  1.27276714e-01
 1.58196899e-04  1.30689236e-02 -2.76892967e-03  1.68498889e-02
-6.74437137e-03  1.43908755e-03 -2.17921627e-02 -5.33130610e-04
-1.84834730e-02  4.76846435e-02 -1.76915842e-03 -2.00026447e-01
```

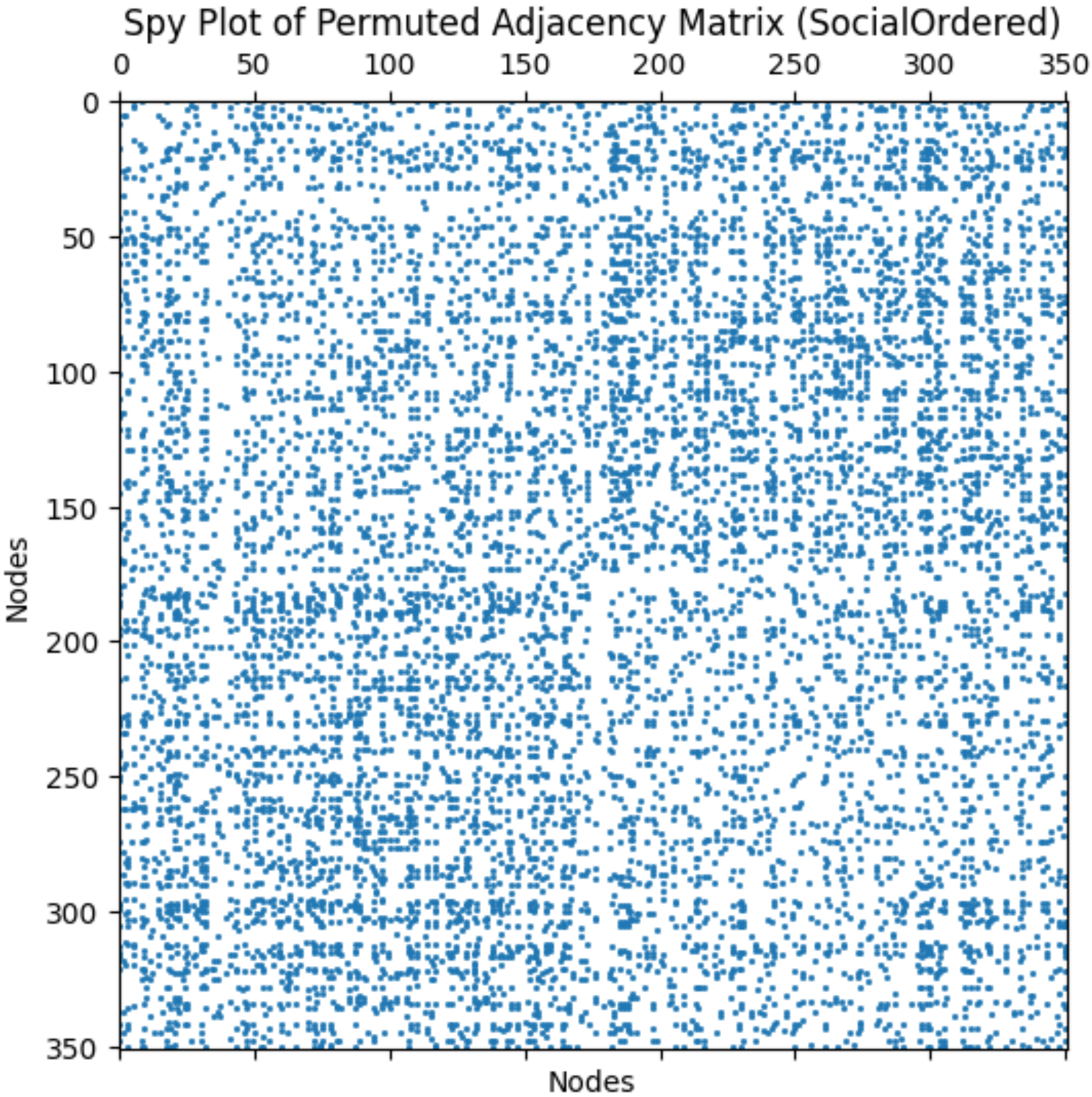
```
 7.57723128e-03 -1.04989228e-02 -2.42339790e-03  2.57064319e-01
 5.03004120e-02 -7.36586479e-02  1.11355910e-01 -5.68024422e-03
 5.33395769e-03 -1.35349559e-01  1.28771500e-02 -4.06662676e-03
-4.04452836e-02 -7.73172302e-02  9.08812695e-03 -4.54141260e-02
-2.91207970e-02 -1.46461800e-02  1.25450476e-02  2.81965722e-03
 8.00182029e-03  1.25771127e-02 -1.68009721e-03 -3.14141204e-03
 4.73560320e-02  1.47719070e-02 -1.03177489e-02 -2.29552328e-02
 1.55357218e-02  4.84842405e-03 -1.03687765e-02  7.47511522e-03
-9.98732324e-03  2.69046225e-02 -8.10422890e-03  1.86000940e-02
-5.36268616e-04 -1.56820682e-02  1.81993043e-03 -6.20376626e-03
 2.50129912e-02 -7.65756138e-02  1.88069494e-02 -1.28771303e-03
 1.41782500e-01 -8.27481775e-03 -1.64385553e-01  1.37329982e-01
 1.50010173e-02  1.08937930e-01  2.96139976e-03  1.17622786e-02
-9.82234623e-03  1.00439358e-02 -1.09461152e-01 -1.47911596e-02
-1.23072035e-01 -1.62957990e-02 -2.63807608e-04 -7.76195096e-02
 4.94955375e-02  1.65924989e-02 -1.42940293e-03  2.02409003e-03
 1.30568233e-01 -7.64818856e-03 -5.83725684e-05 -7.45797771e-03
 3.91360824e-03  4.29502672e-03 -5.97656877e-04  1.61091455e-03
-1.74934784e-01 -1.53610164e-02  3.55765208e-02 -9.99878545e-02
 1.65110621e-01 -1.78460951e-01  1.20287851e-01 -3.98525368e-02
-1.86027286e-03 -9.36484794e-03 -1.71984971e-02  6.24179773e-02
-1.48071549e-03 -1.77973811e-04 -9.73044740e-03 -5.22832816e-03
 4.14751951e-04  7.21082587e-03  4.17500617e-02  2.69802606e-02
 5.51200467e-02  1.00573239e-02 -1.80029188e-03  5.61833973e-02
-1.07784977e-04  3.93835087e-02  1.05435434e-02 -1.20862939e-03
-2.14263191e-03 -2.75859101e-02 -3.51048189e-03 -1.58736574e-02
 7.78455289e-03  7.54854038e-03 -3.30893174e-02  3.82565904e-02
 3.52091027e-02 -1.48831160e-02 -1.35660455e-01  1.65122159e-01
 2.12385721e-02  4.50022337e-02 -2.16444918e-03 -1.58638892e-03
-9.30378218e-03 -3.19200283e-02 -1.06027682e-01 -2.44161991e-02
 3.01969308e-03  1.24471287e-02  1.15639655e-02  5.49638809e-04
 3.87002943e-02 -1.23755560e-02 -1.94826448e-02  7.58025778e-05
-1.62376724e-02 -9.12963729e-03 -5.75154258e-02  2.48648417e-02
 9.55660105e-03  2.05776586e-02  3.75088223e-03  7.34015698e-03
-1.78854472e-02  2.95332799e-02  8.13913474e-02  9.16544786e-02
-1.13632801e-02 -2.19693798e-03 -3.40663822e-02 -1.53527031e-02
-2.39900672e-02  4.32027638e-03 -8.61716185e-03  2.88902614e-02
-1.25357363e-01  2.33821546e-03 -2.96881386e-03 -7.97177269e-03
 1.29967945e-01  5.22468449e-02 -4.79550478e-03  2.09966169e-02
 3.61501335e-03 -9.19280170e-04  8.46881952e-03  9.84037223e-03
-2.05589766e-02 -1.14310728e-01  5.25258325e-02  4.20871227e-02
-2.08598012e-02 -2.16309642e-02  4.29919330e-02 -1.59271404e-02
 4.61564103e-03  2.48893553e-02 -1.15637571e-02 -1.47082027e-04
-4.04863924e-02  1.24979459e-02 -2.43299556e-02 -1.54106924e-02
 1.31201612e-02  2.10770301e-02  2.15915906e-03]

Positive Indices (V2 > 0):
[0, 1, 2, 3, 6, 8, 9, 11, 14, 16, 20, 21, 22, 24, 26, 28, 29, 30, 31, 33, 3

Negative Indices (V2 <= 0):
[4, 5, 7, 10, 12, 13, 15, 17, 18, 19, 23, 25, 27, 32, 35, 37, 39, 42, 43, 4
```

```
print("\nSubtask 12\n")
# Create the order based on positive and negative indices
order = pos + neg # Combine the positive and negative indices
m, n = Social.shape # Get the shape of the Social matrix
iden = np.eye(m) # Identity matrix of size m
# Create the permutation matrix P
P = np.zeros((m, m))
for j in range(m):
    for k in range(m):
        P[j, k] = iden[order[j], k]
# Permute the adjacency matrix
SocialOrdered = P @ Social @ P.T # Using matrix multiplication
print("Shape of SocialOrdered:", SocialOrdered.shape)
```

```
Subtask 12

Shape of SocialOrdered: (351, 351)
```

```
print("\nSubtask 13\n")
# Plot the permuted adjacency matrix
plt.figure(figsize=(8, 6))
plt.spy(SocialOrdered, markersize=1) # Using a smaller marker size for better v
plt.title("Spy Plot of Permuted Adjacency Matrix (SocialOrdered)")
plt.xlabel("Nodes")
plt.ylabel("Nodes")
plt.grid(False) # Disable the grid
plt.show()
```

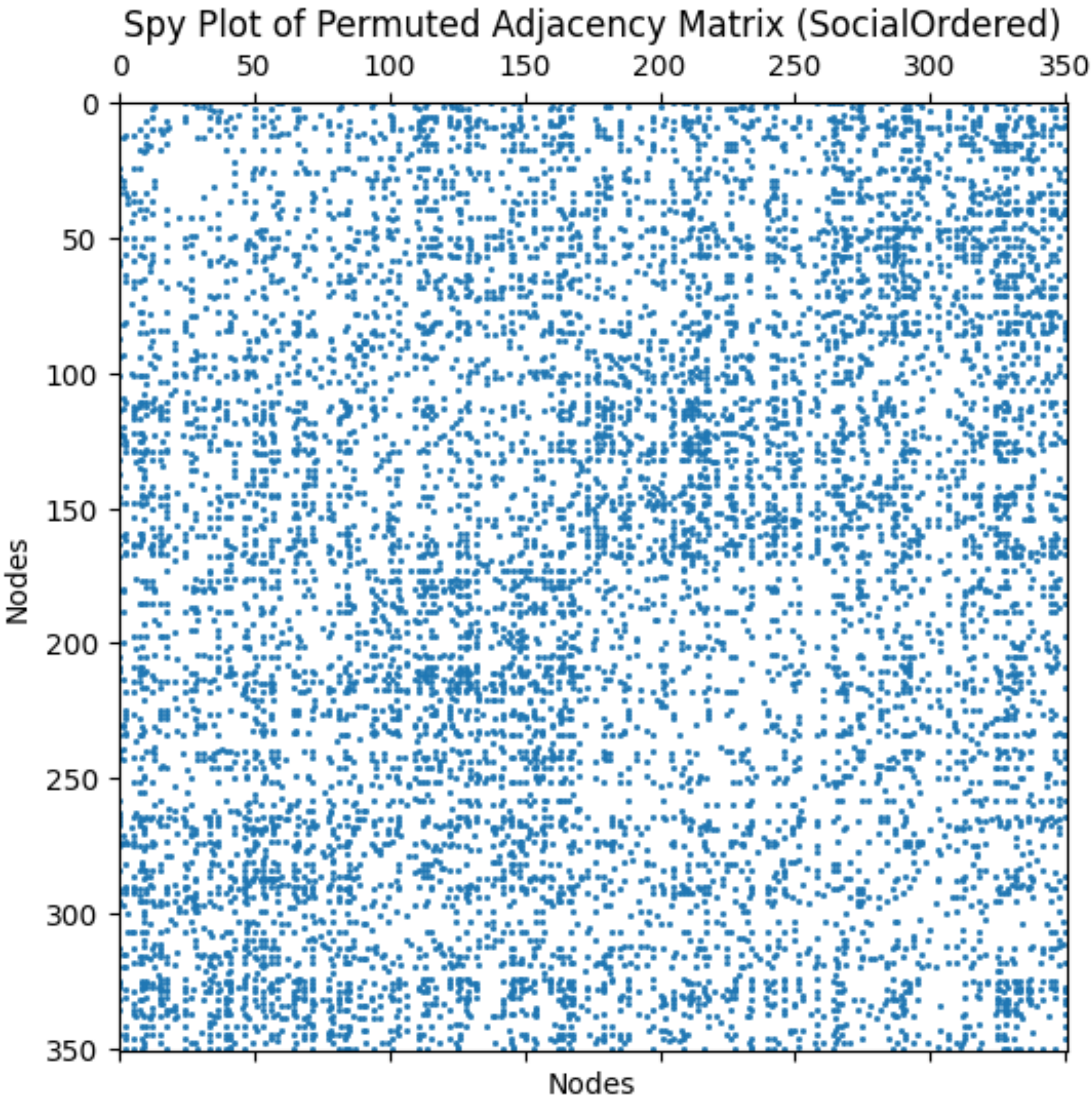
```
Subtask 13
```




```
print("\nSubtask 14\n")
# Explore the third smallest eigenvalue for clustering
V3 = V[:, 2] # Get the third eigenvector
if V3[0] < 0: # Ensure V3 has a positive first entry
```

```
    V3 = -V3
# Initialize lists for the groups
pp = [] # ++ group
pn = [] # +- group
NP = [] # -+ group
nn = [] # -- group
# Grouping based on the signs of V2 and V3
for j in range(len(V2)):
    if V2[j] > 0:
        if V3[j] > 0:
            pp.append(j)
        else:
            pn.append(j)
    else:
        if V3[j] > 0:
            NP.append(j)
        else:
            nn.append(j)
# Combine the orders of the groups
order = pp + pn + NP + nn
m = len(Social) # Get the size of Social
iden = np.eye(m) # Identity matrix of size m
P = np.zeros((m, m)) # Initialize permutation matrix
# Create the permutation matrix
for j in range(m):
    P[j, :] = iden[order[j], :]
# Permute the adjacency matrix
SocialOrdered = P @ Social @ P.T
# Plot the permuted adjacency matrix
plt.figure(figsize=(8, 6))
plt.spy(SocialOrdered, markersize=1)
plt.title("Spy Plot of Permuted Adjacency Matrix (SocialOrdered)")
plt.xlabel("Nodes")
plt.ylabel("Nodes")
plt.grid(False) # Disable the grid
plt.show()
```

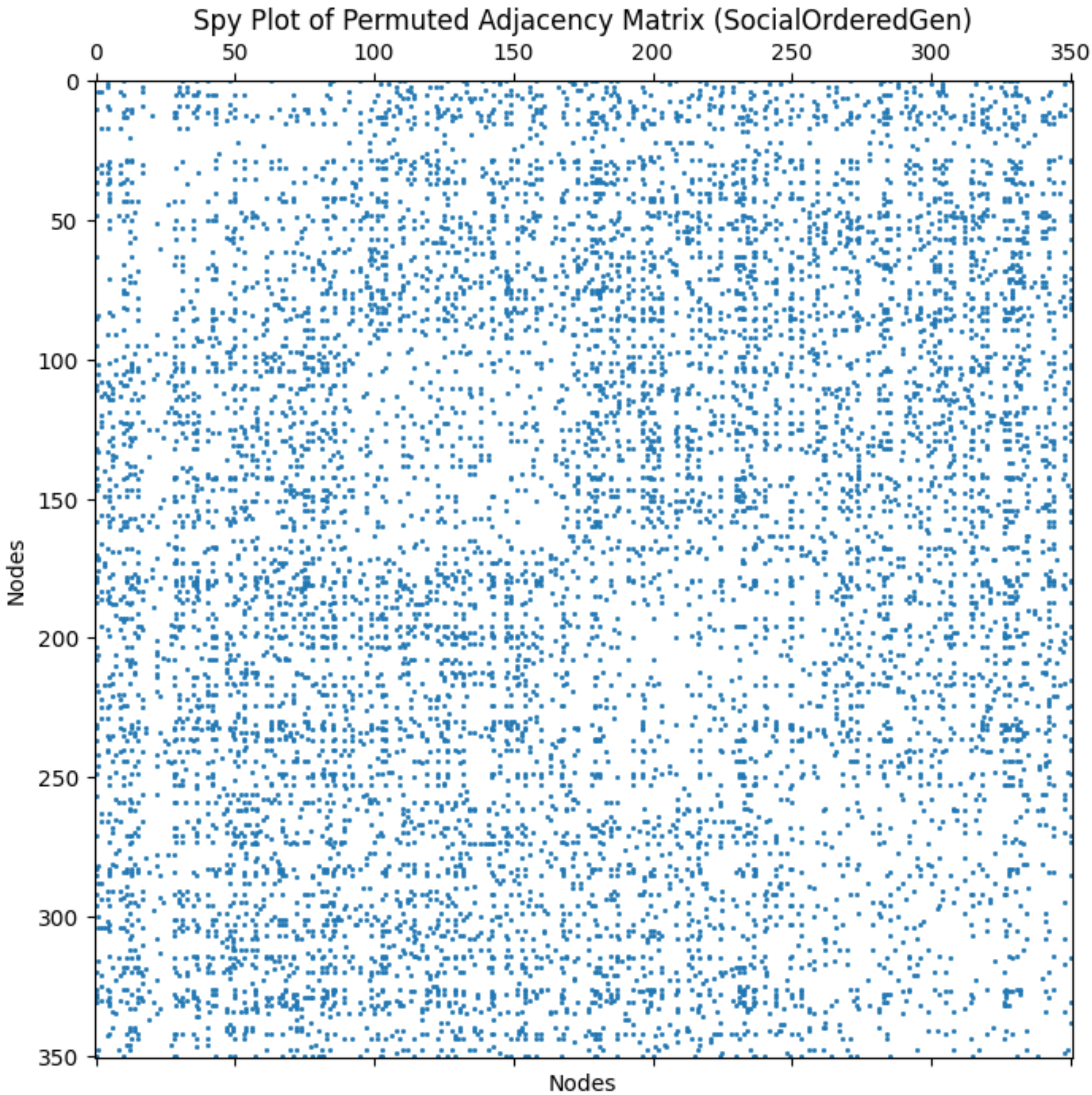
Subtask 14




```
print("\nSubtask 15\n")
# Step 15: Fiedler vector procedure iteratively for clusters
import numpy as np
import matplotlib.pyplot as plt
# Assuming 'Social' is your adjacency matrix, and 'pos' and 'neg' are your posit
# Define SocialPos and SocialNeg based on the positive and negative indices
SocialPos = Social[np.ix_(pos, pos)]
SocialNeg = Social[np.ix_(neg, neg)]
# Calculate the Laplacian for the positive group
rowsumpos = np.sum(SocialPos, axis=1)
DiagSocialPos = np.diag(rowsumpos)
LaplaceSocialPos = DiagSocialPos - SocialPos
# Eigen decomposition for positive group
DPos , VPos = np.linalg.eig(LaplaceSocialPos)
d, ind = np.argsort(DPos), np.argsort(DPos)
```

```
d, ind = np.argsort(DPos), np.argsort(DPos)
DPos = np.diag(DPos[ind])
VPos = VPos[:, ind]
V2Pos = VPos[:, 1] # Second smallest eigenvector for positive group
# Group positive nodes
posp = [] # Positive group
posn = [] # Negative group
for j in range(len(V2Pos)):
    if V2Pos[j] > 0:
        posp.append(pos[j]) # Append original index
    else:
        posn.append(pos[j]) # Append original index
# Calculate the Laplacian for the negative group
rowsumneg = np.sum(SocialNeg, axis=1)
DiagSocialNeg = np.diag(rowsumneg)
LaplaceSocialNeg = DiagSocialNeg - SocialNeg
# Eigen decomposition for negative group
DNeg , VNeg = np.linalg.eig(LaplaceSocialNeg)
d, ind = np.argsort(DNeg), np.argsort(DNeg)
DNeg = np.diag(DNeg[ind])
VNeg = VNeg[:, ind]
V2Neg = VNeg[:, 1] # Second smallest eigenvector for negative group
# Group negative nodes
negp = [] # Positive group
negn = [] # Negative group
for j in range(len(V2Neg)):
    if V2Neg[j] > 0:
        negp.append(neg[j]) # Append original index
    else:
        negn.append(neg[j]) # Append original index
# Generate the final order for the permutation
ordergen = posp + posn + negp + negn
# Create the permutation matrix
m = len(Social) # Assuming the size of Social
iden = np.eye(m) # Identity matrix of size m
P = np.zeros((m, m)) # Initialize permutation matrix
# Create the permutation matrix
for j in range(m):
    P[j, :] = iden[ordergen[j], :] # Filling the permutation matrix based on ord
# Permute the adjacency matrix
SocialOrderedGen = P @ Social @ P.T # Permutation of the Social matrix
# Plot the permuted adjacency matrix
plt.figure(figsize=(10, 8))
plt.spy(SocialOrderedGen, markersize=1)
plt.title("Spy Plot of Permuted Adjacency Matrix (SocialOrderedGen)")
plt.xlabel("Nodes")
plt.ylabel("Nodes")
plt.grid(False) # Disable grid for clarity
plt.show()
```

```
Subtask 15
```


Spy Plot of Permuted Adjacency Matrix (SocialOrderedGen)
0
50
100
150
200
250
300
350
Nodes
Nodes