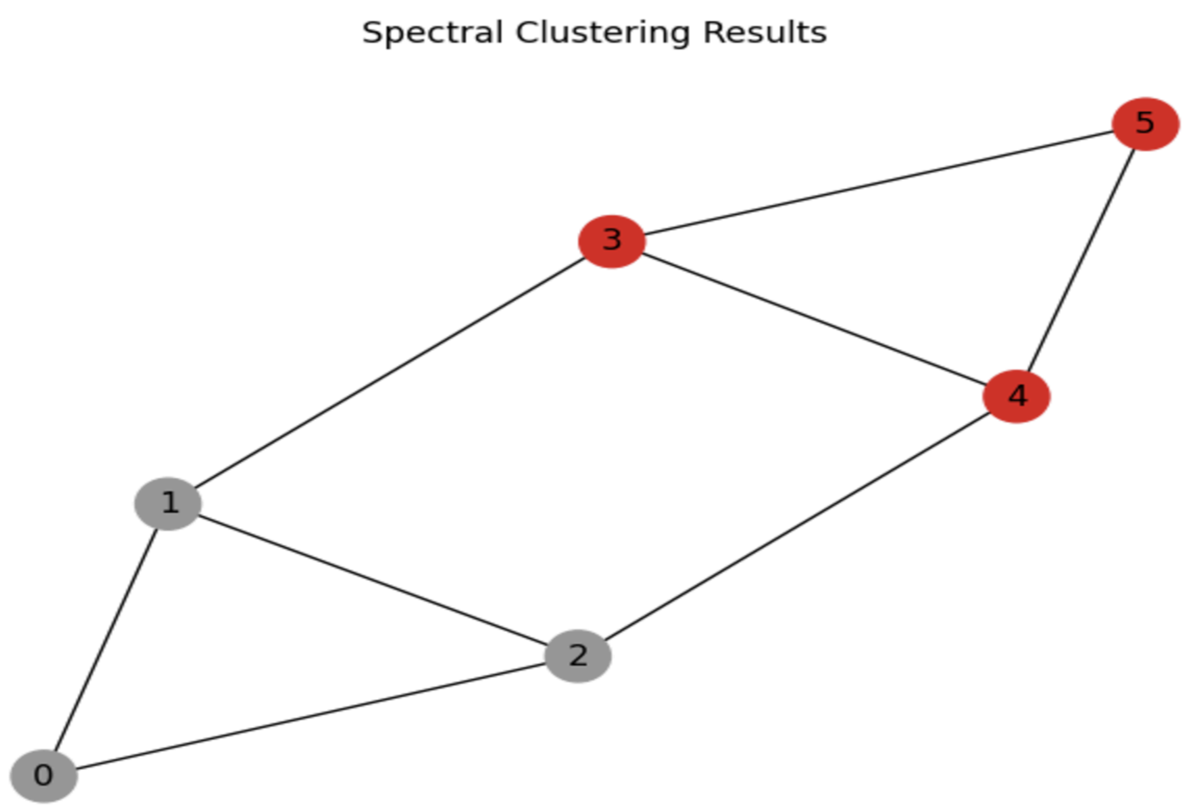
HW3

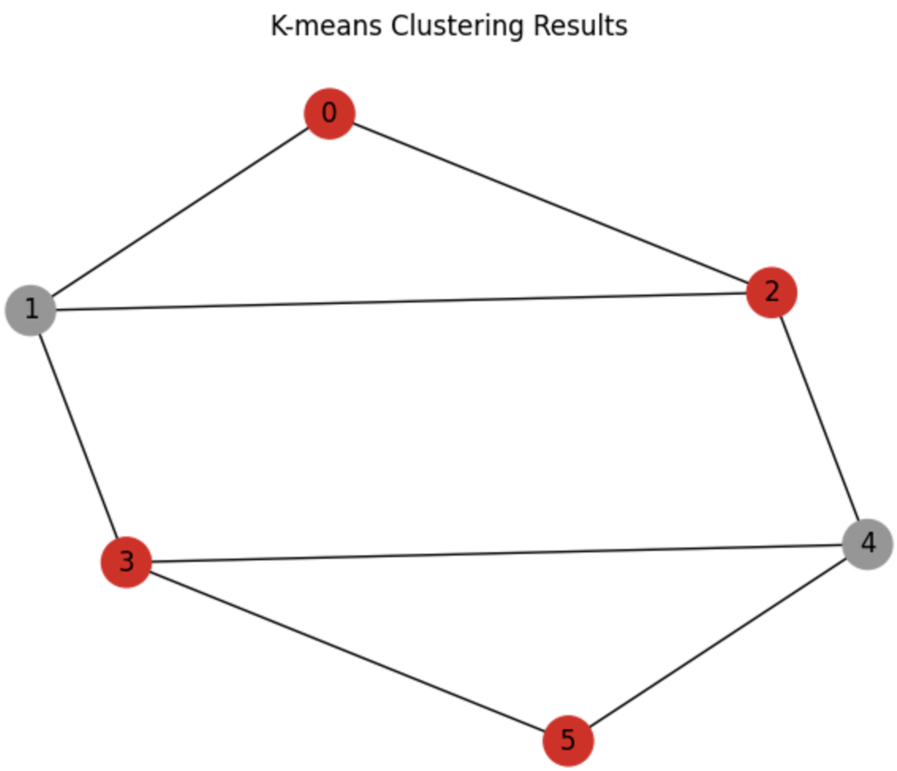
1 code :

```
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from scipy.linalg import eigh

# Define the adjacency matrix for the graph
adj_matrix = np.array([
    [0, 1, 1, 0, 0, 0],
    [1, 0, 1, 1, 0, 0],
    [1, 1, 0, 0, 1, 0],
    [0, 1, 0, 0, 1, 1],
    [0, 0, 1, 1, 0, 1],
    [0, 0, 0, 1, 1, 0]
])

# Step 1: Spectral Clustering
def spectral_clustering(adj_matrix, n_clusters=2):
    # Compute the degree matrix
    degree_matrix = np.diag(np.sum(adj_matrix, axis=1))

    # Compute the Laplacian matrix
    laplacian = degree_matrix - adj_matrix

    # Compute the eigenvalues and eigenvectors of the Laplacian
    eigenvalues, eigenvectors = eigh(laplacian)

    # Select the eigenvectors corresponding to the smallest non-zero eigenvalues
    # Exclude the first column (eigenvector for eigenvalue 0)
    features = eigenvectors[:, 1:n_clusters]

    # Apply K-means on the features
    kmeans = KMeans(n_clusters=n_clusters, random_state=0)
    labels = kmeans.fit_predict(features)
    return labels
# Step 2: K-Means Clustering
def kmeans_clustering(adj_matrix, n_clusters=2):
    # Apply K-means directly to the rows of the adjacency matrix
    kmeans = KMeans(n_clusters=n_clusters, random_state=0)
    labels = kmeans.fit_predict(adj_matrix)
    return labels
```

```
# Apply both clustering methods
spectral_labels = spectral_clustering(adj_matrix)
kmeans_labels = kmeans_clustering(adj_matrix)

# Step 3: Visualize the results using NetworkX
def visualize_clustering(adj_matrix, labels, title):
    G = nx.from_numpy_array(adj_matrix)
    pos = nx.spring_layout(G)  # Spring layout for visualization
    nx.draw(G, pos, with_labels=True, node_color=labels, cmap=plt.cm.Set1, node_size=500)
    plt.title(title)
    plt.show()

# Visualize spectral clustering results
visualize_clustering(adj_matrix, spectral_labels, "Spectral Clustering Results")

# Visualize K-means clustering results
visualize_clustering(adj_matrix, kmeans_labels, "K-means Clustering Results")
```

1 solution :





2 code :

```
import numpy as np
from scipy.linalg import svd, eigh

# Define the matrix M
M = np.array([
    [1, 2],
    [2, 1],
    [3, 4],
    [4, 3]
])

# 1. Perform SVD decomposition
U, Sigma, VT = svd(M)

print("SVD Results:")
print("U:\n", U)
print("Sigma (Singular Values):\n", Sigma)
print("V^T:\n", VT)
```

```
# 2. Compute eigenvalue decomposition of M^T M
MTM = M.T @ M
eigenvalues, eigenvectors = eigh(MTM)

# Sort eigenvalues and eigenvectors in descending order
sorted_indices = np.argsort(eigenvalues)[::-1]
eigenvalues = eigenvalues[sorted_indices]
eigenvectors = eigenvectors[:, sorted_indices]

print("\nEigenvalue Decomposition Results:")
print("Eigenvalues:\n", eigenvalues)
print("Eigenvectors (sorted by eigenvalues):\n", eigenvectors)

# 3. Comparison between V from SVD and eigenvectors of M^T M
print("\nComparison of V from SVD and Eigenvectors of M^T M:")
print("V (from SVD):\n", VT.T)
print("Eigenvectors of M^T M:\n", eigenvectors)

# 4. Relationship between singular values and eigenvalues
singular_values_squared = Sigma**2
print("\nRelationship between singular values and eigenvalues:")
print("Singular Values Squared:\n", singular_values_squared)
print("Eigenvalues of M^T M:\n", eigenvalues)
```

2 solution :

```
i) SVD Results:
U:
 [[-0.27854301  0.5        -0.75033067 -0.33078343]
 [-0.27854301 -0.5         0.12733222 -0.81006191]
 [-0.64993368  0.5         0.57233111  0.00482762]
 [-0.64993368 -0.5        -0.30533177  0.4841061 ]]
Sigma (Singular Values):
 [7.61577311 1.41421356]
V^T:
 [[-0.70710678 -0.70710678]
 [-0.70710678  0.70710678]]

ii) Eigenvalue Decomposition Results:
Eigenvalues:
 [58.  2.]
Eigenvectors (sorted by eigenvalues):
 [[ 0.70710678 -0.70710678]
 [ 0.70710678  0.70710678]]
```

iii) Comparison of V from SVD and Eigenvectors of M^T M:
V (from SVD):
[[-0.70710678 -0.70710678]
[-0.70710678 0.70710678]]
Eigenvectors of M^T M:
[[0.70710678 -0.70710678]
[0.70710678 0.70710678]]

iv) Relationship between singular values and eigenvalues:
Singular Values Squared:
[58. 2.]
Eigenvalues of M^T M:
[58. 2.]

```
Result
({'U': array([[-0.27854301,  0.5       , -0.75033067, -0.33078343],
        [-0.27854301, -0.5       ,  0.12733222, -0.81006191],
        [-0.64993368,  0.5       ,  0.57233111,  0.00482762],
        [-0.64993368, -0.5       , -0.30533177,  0.4841061 ]]),
  'Sigma (Singular Values)': array([7.61577311, 1.41421356]),
  'V^T': array([[-0.70710678, -0.70710678],
        [-0.70710678,  0.70710678]])},
 {'Eigenvalues': array([58.,  2.]),
  'Eigenvectors': array([[ 0.70710678, -0.70710678],
        [ 0.70710678,  0.70710678]])},
 {'V (from SVD)': array([[-0.70710678, -0.70710678],
        [-0.70710678,  0.70710678]]),
  'Eigenvectors of M^T M': array([[ 0.70710678, -0.70710678],
        [ 0.70710678,  0.70710678]])},
 {'Singular Values Squared': array([58.,  2.]),
  'Eigenvalues of M^T M': array([58.,  2.])})
```

SVD Results:

- **U**:

```lua
[[-0.27854301  0.5         -0.75033067 -0.33078343]
 [-0.27854301 -0.5          0.12733222 -0.81006191]
 [-0.64993368  0.5          0.57233111  0.00482762]
 [-0.64993368 -0.5         -0.30533177  0.4841061 ]]
```

- **Sigma (Singular Values)**:

```csharp
[7.61577311 1.41421356]
```

- **V^T**:

```lua
[[-0.70710678 -0.70710678]
 [-0.70710678  0.70710678]]
```

Eigenvalue Decomposition Results:

- **Eigenvalues**:

```csharp
[58.  2.]
```

- **Eigenvectors** (sorted by eigenvalues):

```lua
[[ 0.70710678 -0.70710678]
 [ 0.70710678  0.70710678]]
```

Comparison Between V from SVD and Eigenvectors of $M^T M$:

- **V (from SVD)**:

```lua
[[-0.70710678 -0.70710678]
 [-0.70710678  0.70710678]]
```

- **Eigenvectors of $M^T M$**:

```lua
[[ 0.70710678 -0.70710678]
 [ 0.70710678  0.70710678]]
```

Relationship Between Singular Values and Eigenvalues:

- **Singular Values Squared**:

```csharp
[58.  2.]
```

- **Eigenvalues of $M^T M$**:

```csharp
[58.  2.]
```