In [6]:
```
%matplotlib inline
import matplotlib
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
```

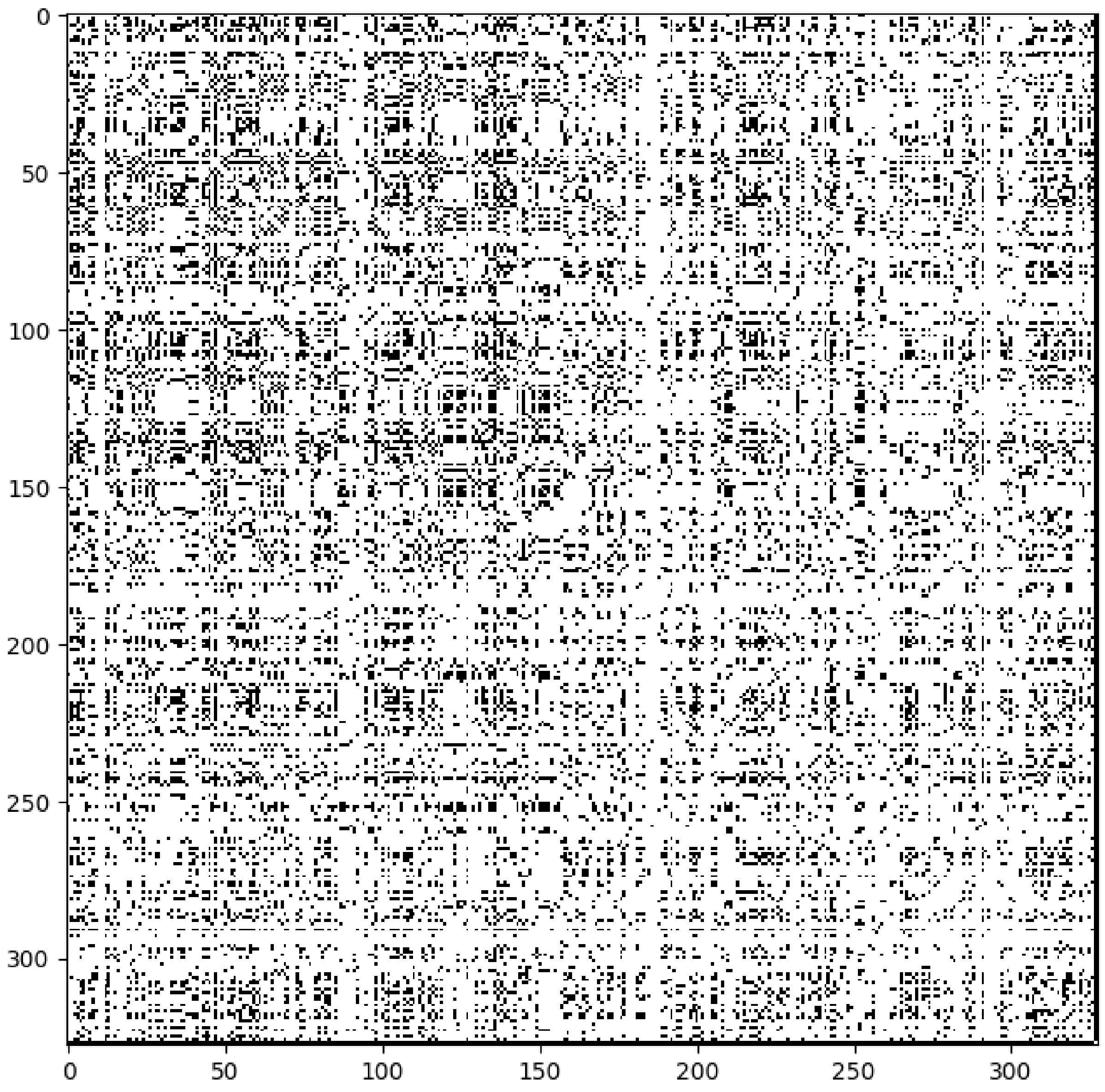
In [7]:
```
# Reads the adjacency matrix from file
A = np.loadtxt('adjacency.txt')
print(f'There are {A.shape[0]} nodes in the graph.')
```

```
There are 328 nodes in the graph.
```

As you can see above, the adjacency matrix is relatively large (328x328): there are 328 persons in the graph. In order to visualize this adjacency matrix, it is convenient to use the 'imshow' function. This plots the 328x328 image where the pixel (i,j) is black if and only if A[i,j]=1.

In [8]:
```
plt.figure(figsize=(8,8))
plt.imshow(A,aspect='equal',cmap='Greys',  interpolation='none')
```

Out[8]:
```
<matplotlib.image.AxesImage at 0x17ae385c220>
```



(a) Construct in the cell below the degree matrix:

$$D_{i,i} = \deg(i) \qquad \text{and} \qquad D_{i,j} = 0 \;\text{ if } i \neq j,$$

the Laplacian matrix:

$$L = D - A$$

and the normalized Laplacian matrix:

$$\tilde{L} = D^{-1/2} L D^{-1/2}.$$

In [25]:
```
# Your answer here
D = np.diag(A.sum(axis = 1))
L = D - A
D_sqrt = np.diag(np.power(A.sum(axis=1),-0.5))
L_norm = D_sqrt @ L @ D_sqrt
L_norm
```

Out[25]:
```
array([[ 1.        ,  0.        ,  0.        , ...,  0.        ,
         0.        , -0.01064251],
       [ 0.        ,  1.        ,  0.        , ...,  0.        ,
         0.        , -0.00606998],
       [ 0.        ,  0.        ,  1.        , ...,  0.        ,
        -0.01628656, -0.00685914],
       ...,
       [ 0.        ,  0.        ,  0.        , ...,  1.        ,
         0.        , -0.00680698],
       [ 0.        ,  0.        , -0.01628656, ...,  0.        ,
         1.        , -0.00726126],
       [-0.01064251, -0.00606998, -0.00685914, ..., -0.00680698,
        -0.00726126,  1.        ]])
```

(b) Using the command 'linalg.eigh' from numpy, compute the eigenvalues and the eigenvectors of $\tilde{L}$.

In [26]:
```
# Your answer here
eigenvalues, eigenvectors = np.linalg.eigh(L_norm)
print(eigenvalues, eigenvectors)
```

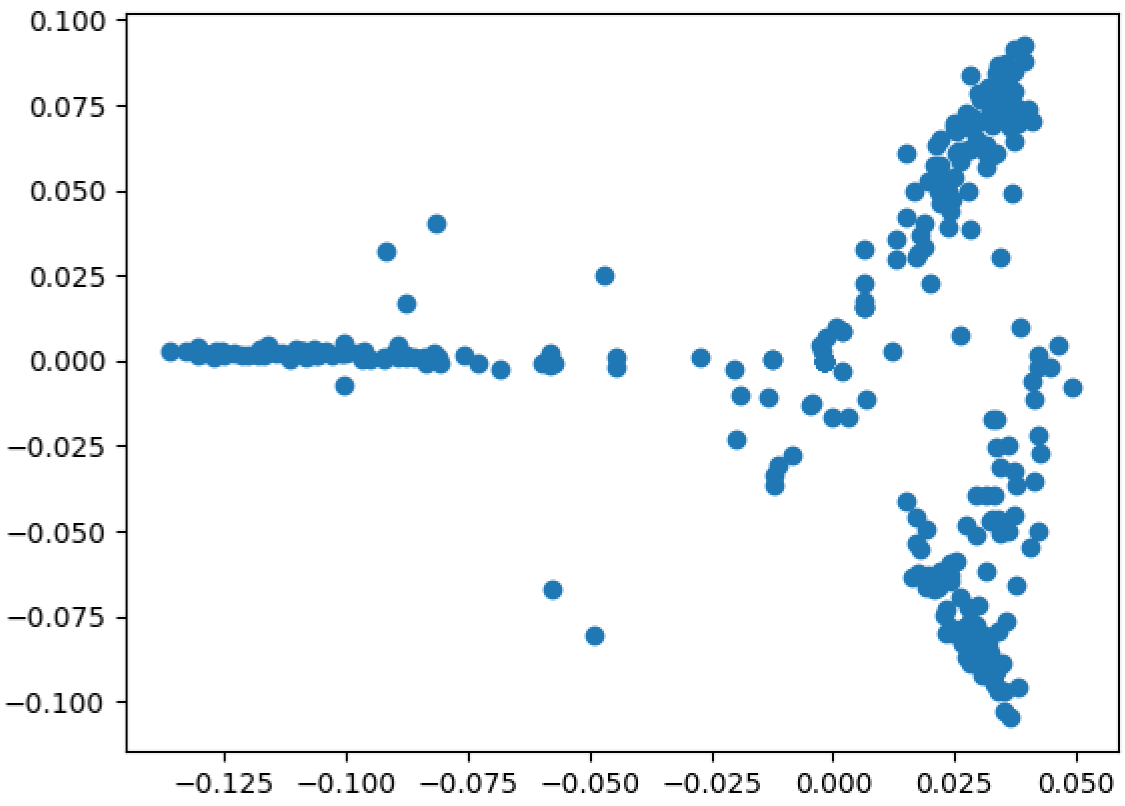
```
[-8.60335003e-16  8.22971080e-02  2.73920284e-01  2.86756239e-01
  4.12340841e-01  4.41921035e-01  6.16054279e-01  6.70249866e-01
  7.06406288e-01  7.25463028e-01  7.35075504e-01  7.52268234e-01
  7.71169767e-01  7.73015222e-01  7.88819638e-01  7.93830094e-01
  8.03472018e-01  8.14921509e-01  8.21827484e-01  8.27206976e-01
  8.35085854e-01  8.38589162e-01  8.46760685e-01  8.56016240e-01
  8.58888980e-01  8.61157975e-01  8.65395654e-01  8.68032309e-01
  8.71228049e-01  8.75612490e-01  8.79472847e-01  8.81071746e-01
  8.83567703e-01  8.85609635e-01  8.87491226e-01  8.90039358e-01
  8.93125892e-01  8.96071979e-01  8.97872008e-01  9.00108767e-01
  9.02215754e-01  9.04086375e-01  9.06058925e-01  9.07454646e-01
  9.09056960e-01  9.09610504e-01  9.13082816e-01  9.13504100e-01
  9.15522027e-01  9.16340436e-01  9.17403135e-01  9.19270844e-01
  9.22130905e-01  9.23161207e-01  9.23511054e-01  9.24832666e-01
  9.28088113e-01  9.29143852e-01  9.30923431e-01  9.32402646e-01
  9.33870511e-01  9.35555401e-01  9.36692116e-01  9.38080974e-01
  9.39859181e-01  9.41188489e-01  9.42451551e-01  9.44524080e-01
  9.45495347e-01  9.46814646e-01  9.47076227e-01  9.48401324e-01
  9.48702506e-01  9.50156125e-01  9.50672500e-01  9.51089565e-01
  9.52229499e-01  9.54700979e-01  9.55973606e-01  9.57012037e-01
  9.59007875e-01  9.59340528e-01  9.60095483e-01  9.61813213e-01
  9.62496471e-01  9.63666669e-01  9.63774402e-01  9.64619656e-01
  9.65020720e-01  9.65381367e-01  9.66932735e-01  9.67219909e-01
  9.69151813e-01  9.69662636e-01  9.70255098e-01  9.71038681e-01
  9.71844822e-01  9.72518729e-01  9.73098475e-01  9.74397971e-01
  9.74765109e-01  9.76393231e-01  9.77111815e-01  9.77513934e-01
  9.78096613e-01  9.79256225e-01  9.79893301e-01  9.80091360e-01
  9.81342758e-01  9.81868729e-01  9.82715996e-01  9.83755570e-01
  9.84490267e-01  9.85156265e-01  9.85601087e-01  9.86227000e-01
  9.87272162e-01  9.87546526e-01  9.88666227e-01  9.88890536e-01
  9.89439372e-01  9.89939160e-01  9.90497496e-01  9.91449441e-01
  9.92825820e-01  9.93252781e-01  9.94124653e-01  9.94843424e-01
  9.95395267e-01  9.96278082e-01  9.97150885e-01  9.98352354e-01
  9.98574988e-01  9.98971618e-01  1.00000000e+00  1.00000000e+00
  1.00000000e+00  1.00000000e+00  1.00000000e+00  1.00000000e+00
  1.00000000e+00  1.00000000e+00  1.00000000e+00  1.00000000e+00
  1.00000000e+00  1.00029329e+00  1.00060897e+00  1.00123040e+00
  1.00217855e+00  1.00264326e+00  1.00318027e+00  1.00360296e+00
  1.00394648e+00  1.00452451e+00  1.00493913e+00  1.00585384e+00
  1.00682744e+00  1.00754310e+00  1.00846734e+00  1.00910240e+00
  1.00984873e+00  1.01066596e+00  1.01081159e+00  1.01131858e+00
  1.01173508e+00  1.01252273e+00  1.01264331e+00  1.01317383e+00
  1.01406340e+00  1.01495706e+00  1.01529273e+00  1.01598070e+00
  1.01700814e+00  1.01735886e+00  1.01789071e+00  1.01846635e+00
  1.01865757e+00  1.01941545e+00  1.02005446e+00  1.02114281e+00
  1.02136189e+00  1.02232889e+00  1.02256850e+00  1.02303408e+00
  1.02348028e+00  1.02382565e+00  1.02407066e+00  1.02531368e+00
  1.02605812e+00  1.02650922e+00  1.02730736e+00  1.02792683e+00
  1.02874064e+00  1.02939545e+00  1.02974518e+00  1.03062856e+00
  1.03145498e+00  1.03241489e+00  1.03278909e+00  1.03328519e+00
  1.03474330e+00  1.03574221e+00  1.03671091e+00  1.03693224e+00
  1.03748780e+00  1.03774168e+00  1.03846617e+00  1.03882506e+00
  1.03914011e+00  1.04164351e+00  1.04213483e+00  1.04249023e+00
  1.04303488e+00  1.04390826e+00  1.04485040e+00  1.04495977e+00
  1.04512557e+00  1.04659637e+00  1.04815093e+00  1.04847570e+00
  1.04879178e+00  1.05002517e+00  1.05073620e+00  1.05111870e+00
```

```
 1.05182571e+00  1.05278393e+00  1.05335259e+00  1.05422111e+00
 1.05520493e+00  1.05628715e+00  1.05670284e+00  1.05759106e+00
 1.05874514e+00  1.05907374e+00  1.05967862e+00  1.06110234e+00
 1.06149099e+00  1.06207963e+00  1.06310380e+00  1.06331023e+00
 1.06488980e+00  1.06609391e+00  1.06761350e+00  1.06873409e+00
 1.06901424e+00  1.06983455e+00  1.07030620e+00  1.07189652e+00
 1.07242659e+00  1.07412998e+00  1.07453284e+00  1.07514340e+00
 1.07554058e+00  1.07696311e+00  1.07767367e+00  1.07821995e+00
 1.08019425e+00  1.08054105e+00  1.08161381e+00  1.08312387e+00
 1.08375073e+00  1.08409038e+00  1.08556125e+00  1.08664060e+00
 1.08761192e+00  1.08912356e+00  1.08937828e+00  1.09082270e+00
 1.09233753e+00  1.09283179e+00  1.09383110e+00  1.09502089e+00
 1.09630653e+00  1.09827775e+00  1.09876234e+00  1.09958891e+00
 1.10096741e+00  1.10140510e+00  1.10351237e+00  1.10384381e+00
 1.10602352e+00  1.10798429e+00  1.10845139e+00  1.10954290e+00
 1.11080279e+00  1.11159317e+00  1.11306899e+00  1.11441063e+00
 1.11561448e+00  1.11638074e+00  1.11844457e+00  1.11906802e+00
 1.12265873e+00  1.12388268e+00  1.12517735e+00  1.12569225e+00
 1.12793035e+00  1.13018630e+00  1.13179435e+00  1.13239995e+00
 1.13358140e+00  1.13734581e+00  1.13883209e+00  1.14147333e+00
 1.14285714e+00  1.14285714e+00  1.14387864e+00  1.14516077e+00
 1.14874599e+00  1.15097199e+00  1.15137460e+00  1.15407050e+00
 1.16007708e+00  1.16168594e+00  1.16181259e+00  1.16613952e+00
 1.17101907e+00  1.17472079e+00  1.17527485e+00  1.19182376e+00
 1.19796968e+00  1.20891304e+00  1.23304226e+00  1.27349456e+00
 1.28324671e+00  1.39234807e+00  1.41442916e+00  1.57937217e+00] [[ 3.73659
565e-02 -8.21492875e-02  9.47994044e-04 ... -5.44469096e-03
   5.32314013e-03 -1.17415024e-04]
 [ 6.55138721e-02  3.34978246e-02 -1.72780496e-02 ...  3.81467899e-05
   1.39720393e-03  1.95839484e-03]
 [ 5.79763540e-02  3.25034504e-02  7.28978656e-02 ... -1.37581831e-04
   9.68810202e-04 -1.28229038e-04]
 ...
 [ 5.84206238e-02  2.87958439e-02 -8.34034886e-02 ... -6.56493088e-04
  -8.72425941e-05 -5.93146577e-05]
 [ 5.47656465e-02  3.06424242e-02  7.66912615e-02 ... -2.46122914e-04
   1.28299897e-03 -1.21988865e-04]
 [ 1.30037346e-01 -2.74479658e-02  8.36567360e-04 ...  2.21492906e-02
  -1.04092103e-01  7.72424456e-03]]
```

(c) We would like to cluster the nodes (i.e. the users) in 3 groups. Using the eigenvectors of $\tilde{L}$, assign to each node a point in $\mathbb{R}^2$, exactly as explained in last lecture (also in 'Algorithm 1' of the notes) where you replace L by $\tilde{L}$. Plot these points using the 'scatter' function of matplotlib.

In [32]:
```
# Your answer here
# k = 3, we only need the first three eigenvectors
eigenvectors_3 = eigenvectors[:, 1:3]
points = eigenvectors_3
plt.scatter(points[:,0],points[:,1])
```

Out[32]:
```
<matplotlib.collections.PathCollection at 0x17aed9587c0>
```

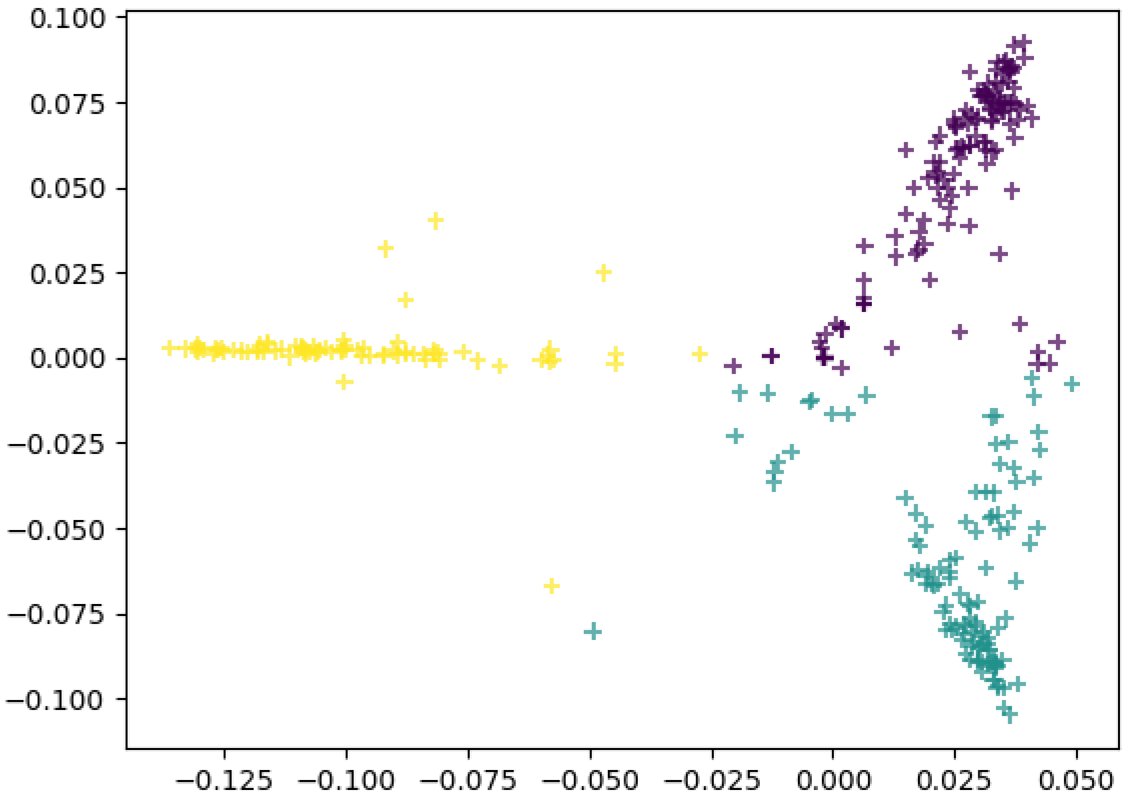


(d) Using the K-means algorithm (use the built-in function from scikit-learn), cluster the embeddings in $\mathbb{R}^2$ of the nodes in 3 groups.

In [41]:
```
# Replace ??? by the matrix of the points computed in (c)
# Each row corresponds to a data point
kmeans = KMeans(n_clusters=3, random_state=0).fit(points)
labels=kmeans.labels_
# labels contains the membership of each node 0,1 or 2
print(labels)
# This colors each point of R^2 according to its label
# replace "x/y coordinates" by the coordinates you computed in (c)
plt.scatter( points[:,0], points[:,1], alpha=0.7, marker='+', c = labels)
```

```
D:\Program_Files\Python\Python310\Lib\site-packages\sklearn\cluster\_kmeans.
py:1412: FutureWarning: The default value of `n_init` will change from 10 to
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
[2 1 0 0 1 1 2 0 0 0 0 1 1 1 2 1 2 2 1 0 0 2 0 2 0 2 1 2 1 1 0 1 0 1 1 1 1
 1 0 0 0 0 0 1 1 1 2 1 2 0 2 0 0 0 1 0 1 1 1 0 1 0 2 0 2 0 2 0 2 0 0 0 1 0
 1 1 1 0 2 0 0 1 0 0 2 0 2 2 2 2 1 2 0 0 0 1 2 2 0 2 0 1 1 1 1 1 2 1 1 1 0
 2 0 1 2 0 1 0 2 0 2 2 2 0 2 2 2 0 0 2 1 2 1 1 2 2 1 0 1 0 1 1 1 2 0 2 0 0
 2 1 2 2 2 2 1 2 2 1 0 0 0 0 1 0 0 0 1 2 2 0 0 2 0 0 2 0 1 1 0 2 0 0 0 0 0
 0 1 0 2 1 1 0 0 2 1 1 1 1 0 1 1 1 0 0 0 0 0 2 1 2 2 2 1 0 1 1 1 1 1 1 0 1
 1 0 1 2 2 1 1 0 0 1 2 1 0 1 0 1 1 0 0 1 0 2 1 0 0 0 1 1 0 2 2 2 0 0 0 0 1
 2 2 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 1 0 1 2 2 2 1 0 1 0 2 1 0 0 1 0 1
 0 1 1 2 1 2 0 0 2 0 0 0 1 1 0 1 0 0 0 1 1 1 0 0 1 0 1 2 0 1 0 2]
```

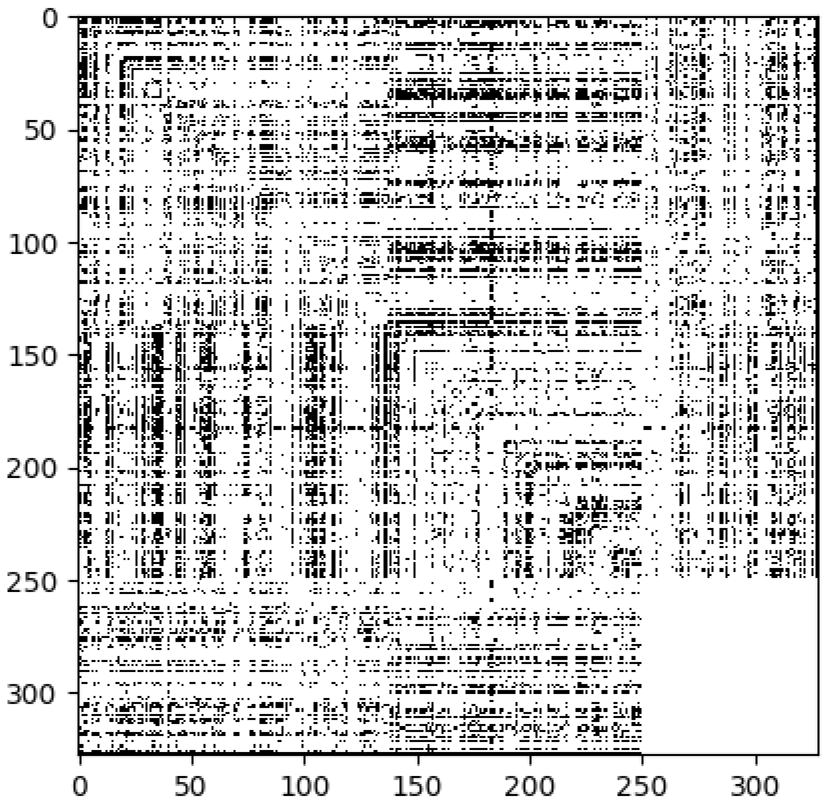
Out[41]:
```
<matplotlib.collections.PathCollection at 0x17aedee8e80>
```



(e) Re-order the adjacency matrix according to the clusters computed in the previous question. That is, reorder the columns and rows of A to obtain a new adjacency matrix (that represents of course the same graph) such that the n_1 nodes of the first cluster correspond to the first n_1 rows/columns, the n_2 nodes of the second cluster correspond to the next n_2 rows/columns, and the n_3 nodes of the third cluster correspond to the last n_3 rows/columns. Plot the reordered adjacency matrix using 'imshow'.

In [85]:
```
## Your answer here
import pandas as pd
from functools import reduce
point_label_df = pd.DataFrame(np.hstack((points,labels.reshape(-1,1))))
concatenated = reduce(lambda accu, curr: accu + curr,[list(point_label_df[pc
reordered_adj = np.zeros_like(A)

counter = 0
for i in concatenated:
    reordered_adj[counter,:] = A[i,:]
    reordered_adj[:,counter] = A[:,i]
    counter += 1

plt.imshow(reordered_adj, aspect='equal',cmap='Greys',  interpolation='none'
```

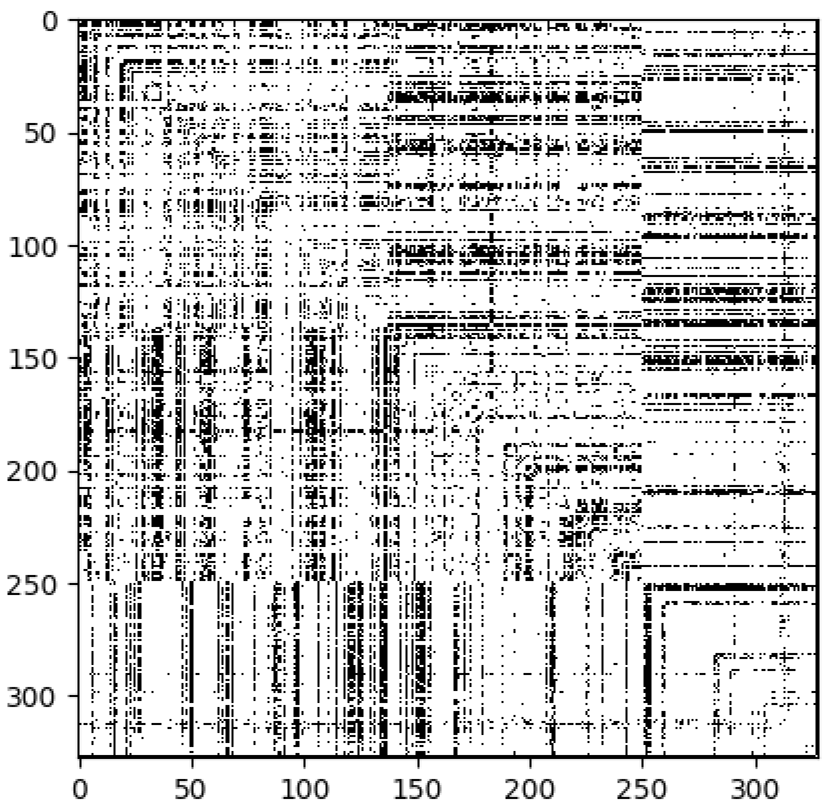
Out[85]:
```
<matplotlib.image.AxesImage at 0x17af2291810>
```



In [110…

```
# Full Algorithm Packages
class Spectral_Graph_Clustering_Algorithm():

    def __init__(self, filename):
        self.filepath = filename
        self.A = self.load_data(filename)


    def load_data(self, filepath):
        A = np.loadtxt(filepath)
        return A

    def compute_L_A_D(self, A, normalized = True):
        D = np.diag(A.sum(axis = 1))
        L = D - A
        D_sqrt = np.diag(np.power(A.sum(axis = 1),- 0.5))
        L_norm = D_sqrt @ L @ D_sqrt
        L_norm
        return L_norm if normalized else L

    def compute_data_for_KMeans(self,L,k):
        eigenvalues, eigenvectors = np.linalg.eigh(L)

        data_matrix = eigenvectors[:, 1:k]

        self.eigenmatrix = data_matrix

        return data_matrix
    def perform_KMeans(self,data, k):
        import pandas as pd
        from functools import reduce
        kmeans = KMeans(n_clusters=k, random_state=0).fit(data)
        labels = kmeans.labels_

        return labels


    def reorder_adj_matrix(self,A,k):
        point_label_df = pd.DataFrame(np.hstack((points,labels.reshape(-1,1)
        concatenated = reduce(lambda accu, curr: accu + curr,[list(point_lab
        reordered_adj = np.zeros_like(A)
        counter = 0
        for i in concatenated:
            reordered_adj[counter,:] = A[i,:]
            reordered_adj[:,counter] = A[:,i]
            counter += 1

        return reordered_adj

    def fit(self, k, normalized = True):
        A = self.load_data(self.filepath)
        L = self.compute_L_A_D(A)
        data_matrix = self.compute_data_for_KMeans(L, k)
        labels = self.perform_KMeans(data_matrix, k)
        reordered_adj = self.reorder_adj_matrix(A,k)
        self.reordered_adj = reordered_adj
        self.L = L
        return reordered_adj


    def plot_adj_matrix(self,):
        plt.imshow(self.reordered_adj, aspect='equal',cmap='Greys',  interpo

    def plot_clusters(self,):
        plt.scatter(self.eigenmatrix[:,0],self.eigenmatrix[:,1])
        plt.show()



adj_filepath = "./adjacency.txt"
SGCA = Spectral_Graph_Clustering_Algorithm(adj_filepath)


SGCA.load_data(adj_filepath)
reordered_adj = SGCA.fit(k = 6)
SGCA.plot_adj_matrix()
# SGCA.plot_clusters()
plt.show()
```

```
D:\Program_Files\Python\Python310\Lib\site-packages\sklearn\cluster\_kmeans.
py:1412: FutureWarning: The default value of `n_init` will change from 10 to
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```



```
In [ ]:
```