In [ ]:
```
%matplotlib inline
import matplotlib
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import pandas as pd
import seaborn as sns
```

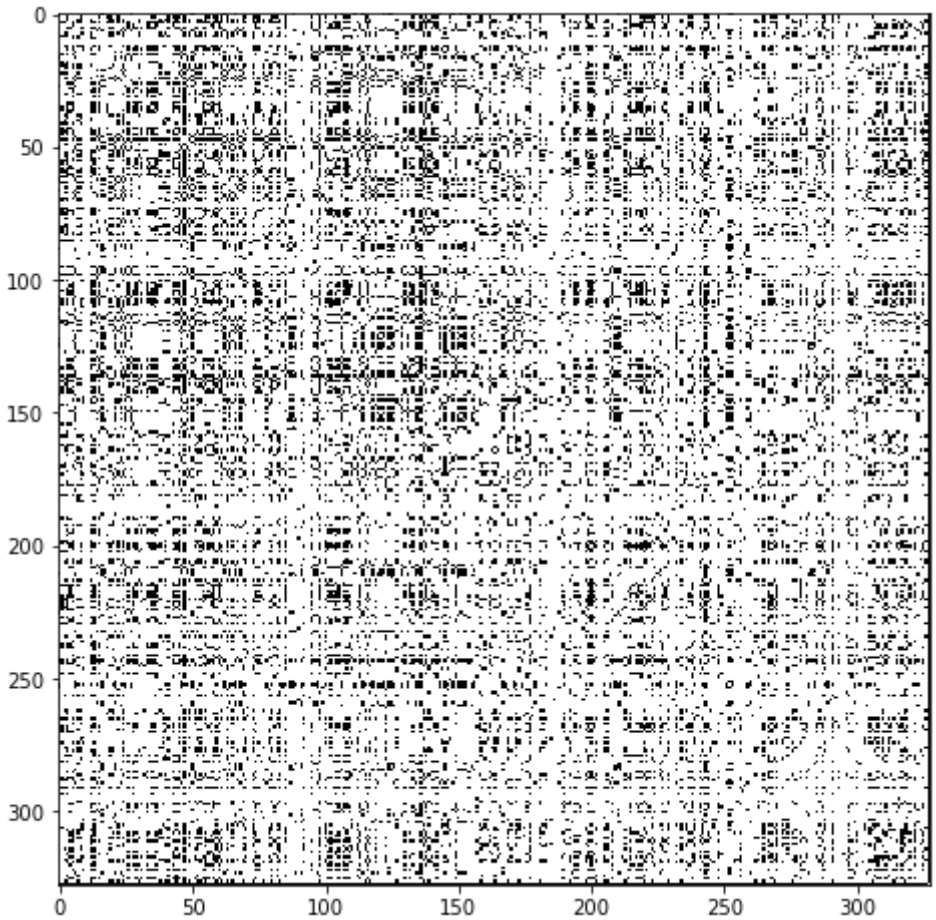
In [ ]:
```
# Reads the adjacency matrix from file
A = np.loadtxt('adjacency.txt')
print(f'There are {A.shape[0]} nodes in the graph.')
```

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

Out[ ]:
```
<matplotlib.image.AxesImage at 0x7f91b90bc8b0>
```



(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:

$$L_{\text{norm}} = D^{-1/2} L D^{-1/2}.$$

In [ ]:
```
# Your answer here
from scipy.linalg import fractional_matrix_power

## Get sum of all rows --> turn row sums into diagonal matrix
row_sums = np.sum(A,axis=1).tolist()
D = np.diag(row_sums)
## Compute L
L = D - A
## Take D ^ -0.5
D_norm = fractional_matrix_power(D, -0.5)
## Compute L_norm
L_norm = D_norm @ L @ D_norm
L_norm
```

Out[ ]:
```
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 L_{norm}.

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

```
[-7.80703533e-17  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]
[[ 3.73659565e-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]]
```

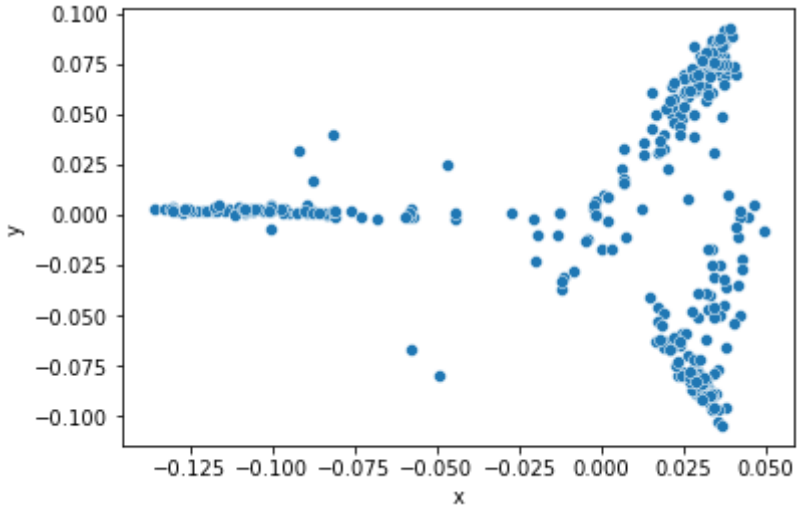
(c) We would like to cluster the nodes (i.e. the users) in 3 groups. Using the eigenvectors of L_{norm}, assign to each node a point in $\mathbb{R}^2$, exactly as explained in last lecture (also in 'Algorithm 1' of the notes) where you replace L by L_{norm}. Plot these points using the 'scatter' function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

In [ ]:

```
# Your answer here

## Get first 3 eigenvectors
k_eigenv = eigenvectors[:,:3]
k_eigenv.shape

## Associate nodes with vectors
k_minus_1_eigenv = k_eigenv[:,1:]
k_minus_1_eigenv

## Format Data for plotting
data = pd.DataFrame(k_minus_1_eigenv)
data.columns = ['x', 'y']

sns.scatterplot(data=data, x="x", y="y")
```

Out[ ]:

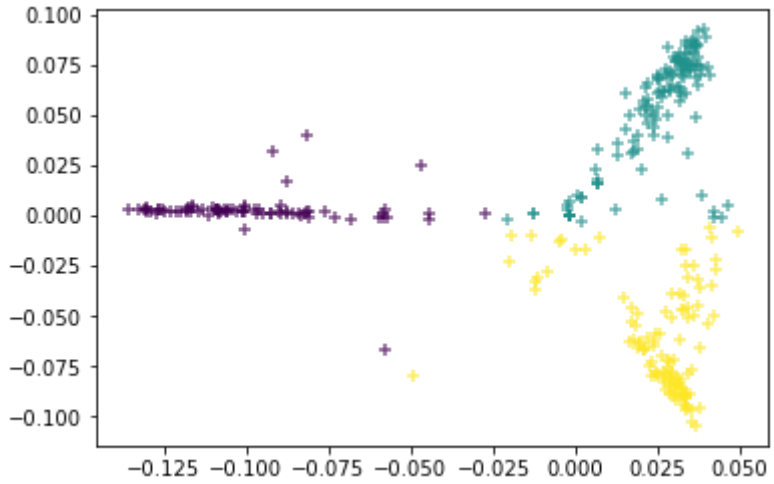
```
<AxesSubplot:xlabel='x', ylabel='y'>
```



(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 [ ]:
```
# 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(k_minus_1_eigenv)
labels=kmeans.labels_
# labels contains the membership of each node 0,1 or 2

data['label'] = 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( data['x'], data['y'], alpha=0.7, marker='+', c = labels)
```

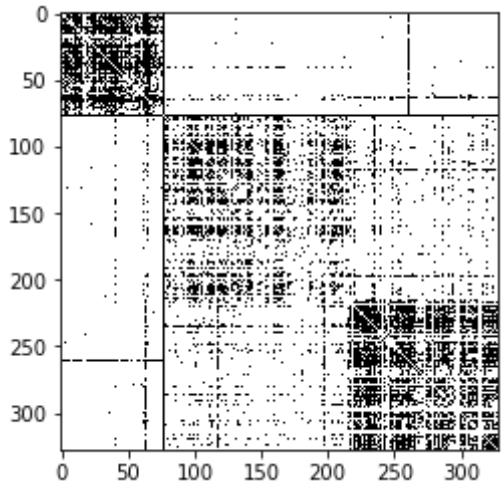
Out[ ]:
```
<matplotlib.collections.PathCollection at 0x7f91a967d6a0>
```



(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 [ ]:
```
## Your answer here
zero_index = data.index[data['label'] == 0].tolist()
one_index =  data.index[data['label'] == 1].tolist()
two_index =  data.index[data['label'] == 2].tolist()

index_list = zero_index + one_index + two_index

reorder_A = A[index_list]
reorder_A = reorder_A.transpose()
reorder_A = reorder_A[index_list]

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

Out[ ]:
```
<matplotlib.image.AxesImage at 0x7f91db8d1070>
```



In [ ]:

In [ ]: