# friends_sol

November 19, 2021

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

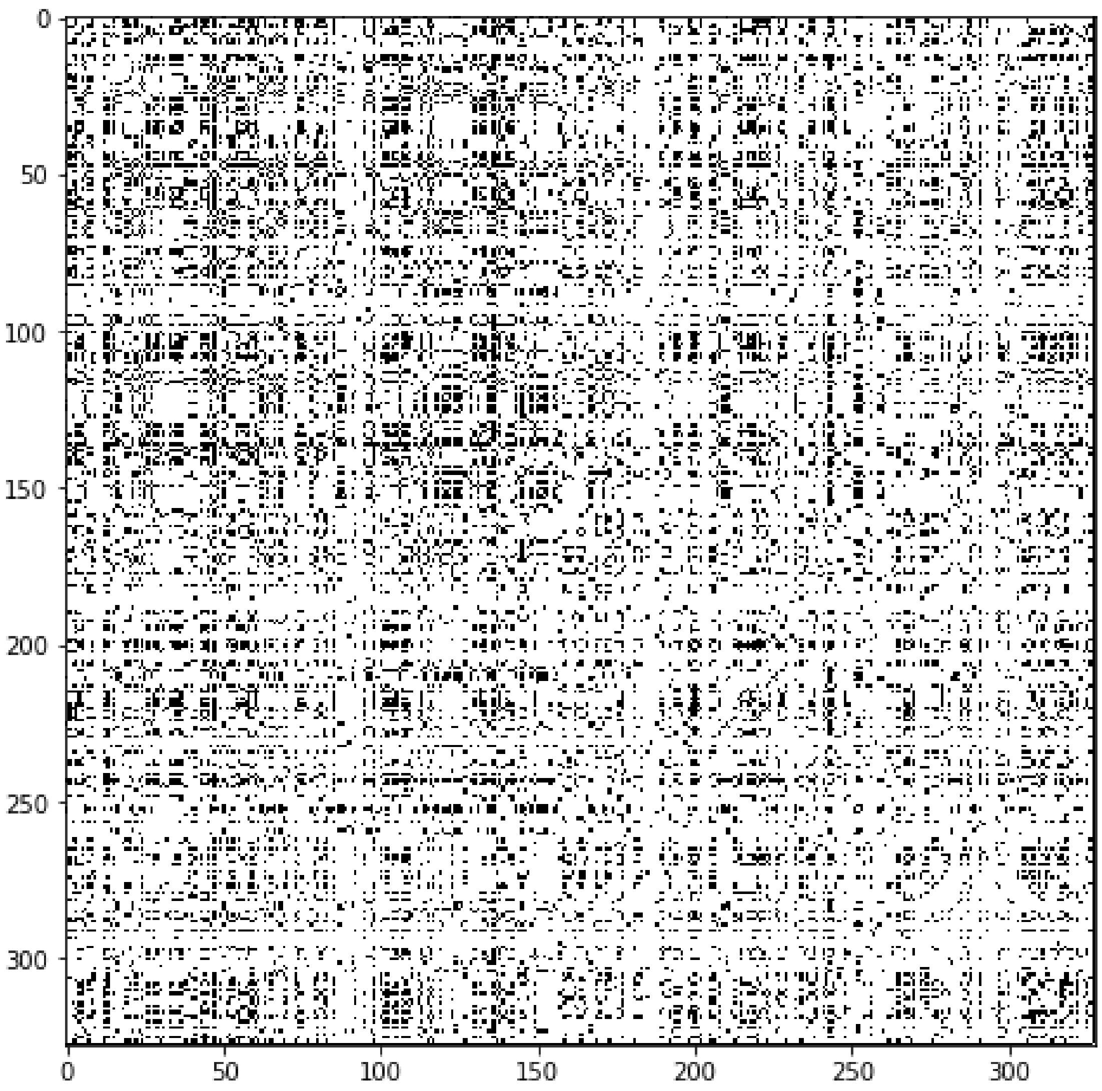
[ ]:
```
# Reads the adjacency matrix from file
A = np.loadtxt('adjacency.txt')
print(f'There are {A.shape[0]} nodes in the graph.')
```

```
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.

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

[ ]:
```
<matplotlib.image.AxesImage at 0x7fe825445f50>
```



**(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:

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

```
[ ]: # Your answer here
     D = np.diag(A.sum(0))
     D_sqrtinv = np.diag(1 / np.sqrt(A.sum(0)))
```

```
L = D - A

L_norm = D_sqrtinv @ L @ D_sqrtinv
```

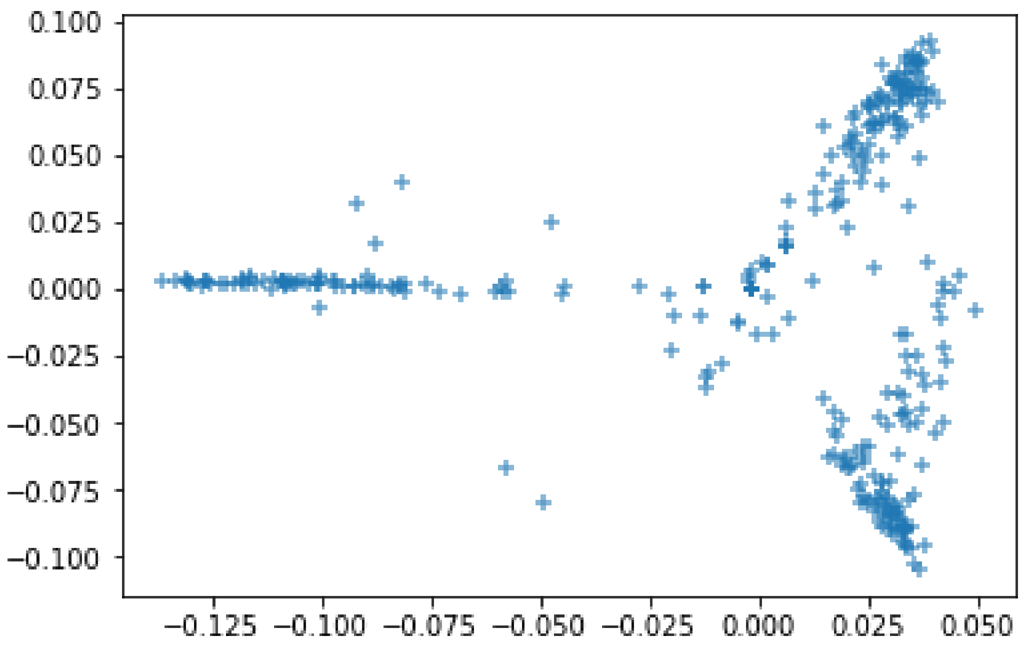
**(b)** Using the command 'linalg.eigh' from numpy, compute the eigenvalues and the eigenvectors of $L_{\text{norm}}$.

[ ]:
```
# Your answer here
val, vec = np.linalg.eigh(L_norm)
```

**(c)** We would like to cluster the nodes (i.e. the users) in 3 groups. Using the eigenvectors of $L_{\text{norm}}$, 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 $L_{\text{norm}}$. Plot these points using the 'scatter' function of matplotlib.

[ ]:
```
# Your answer here
x = vec[:, 1:3]
plt.scatter(x[:, 0], x[:, 1], alpha=0.7, marker='+')
```

[ ]:
```
<matplotlib.collections.PathCollection at 0x7fe82478d990>
```



**(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.

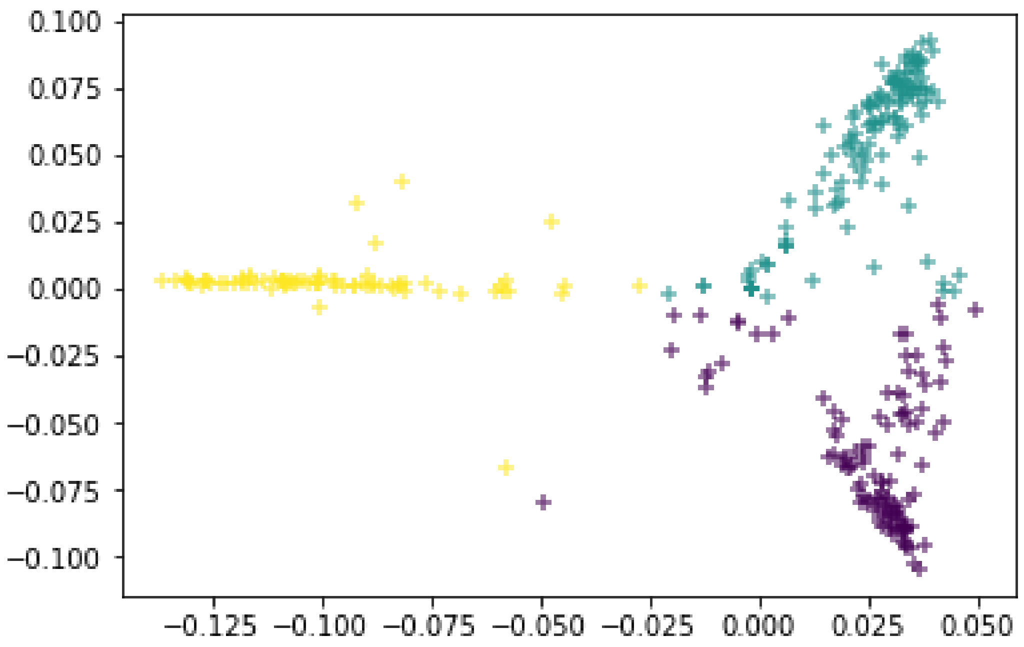
[ ]:
```
# Replace ??? by the matrix of the points computed in (c)
# Each row corresponds to a data point
# kmeans = KMeans(n_clusters=3, random_state=0).fit(???)
```

```
kmeans = KMeans(n_clusters=3, random_state=0).fit(x)
labels=kmeans.labels_
# labels contains the membership of each node 0,1 or 2

# This colors each point of R^2 according to its label
# replace "x/y coordinates" by the coordinates you computed in (c)
# plot.scatter( "x coordinate", "y coordinate", alpha=0.7, c=labels)

plot.scatter( x[:, 0], x[:, 1], alpha=0.7, marker='+', c = labels)
```

[ ]: 
```
<matplotlib.collections.PathCollection at 0x7fe825d43550>
```



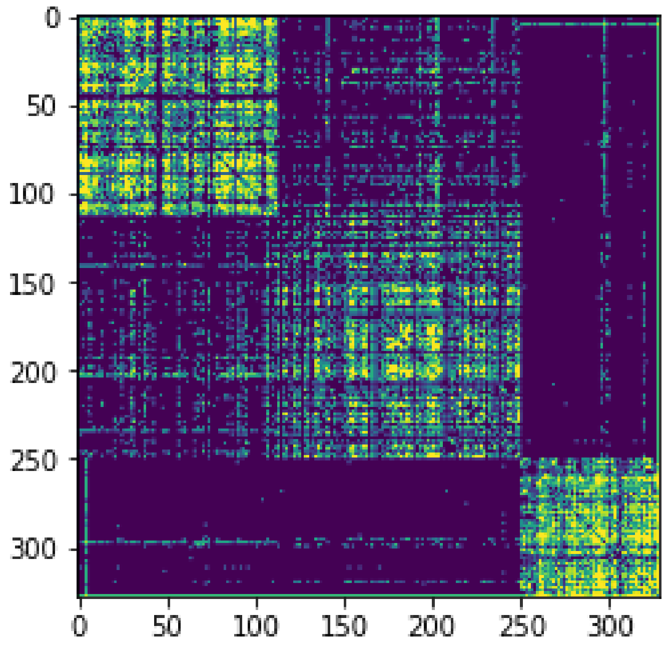
**(e)** Re-order the adjacency matrix according to the clusters computed in the previous question. That is, reorder the columns and rows of $A$ to obtain a new adjacency matrix (that represents of course the same graph) such that the $n_1$ nodes of the first cluster correspond to the first $n_1$ rows/columns, 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'.

[ ]:
```
## Your answer here
N = A.shape[0]
sorted_indx = np.argsort(labels)
rA = np.zeros_like(A)

for i in range(N):
    for j in range(N):
        rA[i, j] = A[sorted_indx[i], sorted_indx[j]]

plt.imshow(rA)
```

[ ]: <matplotlib.image.AxesImage at 0x7fe82648bd50>



```
[ ]:
```