Problem Set Two

Zhixian(Jason) Yu

02/27/2018

1 Theory

Problem 1

We can rewrite the optimization problem to form the Lagrangian:

$$\underset{x}{\text{maximize}} \quad u^T Av - \lambda(u^T u - 1) - \mu(v^T v - 1) \tag{1}$$

We can get a necessary condition by taking the partial derivatives over u, v and set them to zero. Thus we get:

$$\frac{\partial A}{\partial u} = Av - 2\lambda u = 0 \tag{2}$$
$$\frac{\partial A}{\partial v} = (u^T A)^T - 2\mu v = 0 \tag{3}$$

We can rewrite the above equation and get:

$$Av = 2\lambda u \tag{4}$$
$$u^T A = 2\mu v^T \tag{5}$$

Because $u^T u = 1$, we get the following if we multiply equation 4 by u^T from the left:

$$u^T Av = 2\lambda u^T u = 2\lambda \tag{6}$$

Similarly because $v^T v = 1$, we get from equation 5

$$u^T Av = 2\mu v^T v = 2\mu \tag{7}$$

From equations 6 and 7, we get $\lambda = \mu$. Let $\sigma = 2\lambda$, equations 4 and 5 become:

$$Av = \sigma u$$
$$A^T u = \sigma v$$

Because u and v are unit-length vectors, σ is a singular value for A, and u, v are the left-singular and right-singular vectors, respectively. Thus the necessary condition for the solution is given by SVD, i.e. $A = u\sigma v^T$.

Problem 2

a) Let $P = \frac{1}{n} ee^T$, so:

$$P^2 = \frac{1}{n^2} ee^T ee^T$$

Since $e^T e = n$, we have:

$$\begin{aligned} P^2 &= \frac{1}{n^2} ene^T \\ &= \frac{1}{n} ee^T \\ &= P \end{aligned}$$

Therefore $\frac{1}{n} ee^T$ is a projection.

b) The projection first calculates the mean point of the set of points, and then projects each of the point to the same mean point.

c) By applying the H matrix from the left, we have:

$$B = HA = A - \frac{1}{n}ee^T A$$

Suppose a_{ij} and b_{ij} are the ith row, jth column elements of A and B, respectively. So:

$$\begin{aligned} b_{ij} &= a_{ij} - \frac{1}{n}\sum_{k=1}^{n}(1 \cdot a_{kj}) \\ &= a_{ij} - \frac{1}{n}\sum_{k=1}^{n} a_{kj} \end{aligned}$$

Therefore it subtracts each element a_{ij} of A by the mean value of the jth column elements. In addition, each column of B has a sum of 0, showing in the following:

$$\begin{aligned} \sum_{l=1}^{n} b_{lj} &= \sum_{l=1}^{n}(a_{lj} - \frac{1}{n}\sum_{k=1}^{n} a_{kj}) \\ &= \sum_{l=1}^{n} a_{lj} - n(\frac{1}{n}\sum_{k=1}^{n} a_{kj}) \\ &= \sum_{l=1}^{n} a_{lj} - \sum_{k=1}^{n} a_{kj} \\ &= 0, \forall j \in [1, n] \end{aligned}$$

d) Similarly with last problem

$$B = AH = A - \frac{1}{n}Aee^T$$

Suppose a_{ij} and b_{ij} are the ith row, jth column elements of A and B, respectively. So:

$$\begin{aligned} b_{ij} &= a_{ij} - \frac{1}{n}\sum_{k=1}^{n}(a_{ik} \cdot 1) \\ &= a_{ij} - \frac{1}{n}\sum_{k=1}^{n} a_{ik} \end{aligned}$$

Therefore it subtracts each element a_{ij} of A by the mean value of the ith row elements. Similarly, the row sum of the result matrix B is 0.

$$\begin{aligned} \sum_{l=1}^{n} b_{il} &= \sum_{l=1}^{n}(a_{il} - \frac{1}{n}\sum_{k=1}^{n} a_{ik}) \\ &= \sum_{l=1}^{n} a_{il} - n(\frac{1}{n}\sum_{k=1}^{n} a_{ik}) \\ &= \sum_{l=1}^{n} a_{il} - \sum_{k=1}^{n} a_{ik} \\ &= 0, \forall i \in [1, n] \end{aligned}$$

e) We need to show that applying H from the right does not change the column sum of HA, i.e. 0. If we expand $B = HAH$, we get:

$$b_{ij} = (HA)_{ij} - \frac{1}{n}(HAee^T)_{ij}$$

and we want to prove

$$\sum_{l=1}^{n} b_{lj} = 0$$

As was shown in problem **c)**, $\sum_{l=1}^{n}(HA)_{lj} = 0, \forall j \in [1, n]$. therefore:

$$\begin{aligned}
\sum_{l=1}^{n} b_{lj} &= \sum_{l=1}^{n}(HA)_{lj} - \frac{1}{n}\sum_{l=1}^{n}(HAee^T)_{lj} \\
&= -\frac{1}{n}\sum_{l=1}^{n}(HAee^T)_{lj} \\
&= -\frac{1}{n}\sum_{l=1}^{n}\sum_{k=1}^{n}[(HA)_{lk} \cdot 1] \\
&= -\frac{1}{n}\sum_{l=1}^{n}\sum_{k=1}^{n}(HA)_{lk} \\
&= -\frac{1}{n}\sum_{k=1}^{n}[\sum_{l=1}^{n}(HA)_{lk}] \\
&= 0, \forall j \in [1, n]
\end{aligned}$$

Therefore, sum of any column of B will be 0.

Problem 3

We have

$$\begin{aligned}
He &= (I - ee^T/n)e \\
&= e - ee^Te/n \\
&= e - e \\
&= 0e
\end{aligned}$$

Therefore the vector of ones is an eigenvector of B, and the associated eigenvalue is 0.

Problem 4

As is shown in problem 3, $Be = 0$. Therefore

$$Be = \tilde{V}(\tilde{V}^T e) = 0 \tag{8}$$

In equation 8, $\tilde{V}(\tilde{V}^T e)$ means the linear combination of the columns of $\tilde{V}$ weighted by $(\tilde{V}^T e)$. Because the columns of $\tilde{V}$ are different eigenvectors associated with different eigenvalues (times different constants), the columns of $\tilde{V}$ are linearly independent. Therefore, equation 8 only has the trivial solution, and $\tilde{V}^T e = 0$.

Problem 5

In $\tilde{V}^T$, each column is a configuration. Let a_{ij} be the ith row, jth column element of $\tilde{V}^T$. Because $\tilde{V}^T e = 0$, we have:

$$\begin{aligned}\sum_{j=1}^{n}(a_{ij} \cdot 1) &= \sum_{j=1}^{n} a_{ij} \\ &= 0, \forall i \in [1, n]\end{aligned}$$

This means for each row, the sum across all the columns of $\tilde{V}^T$ is 0, therefore the mean of all the configurations is the origin.

2 Programming

The following code sets up the environment and import packages.

```
import numpy as np
import matplotlib.pyplot as plt
```

Problem 1

The following code defines a function to execute MDS algorithm.

```
def MDS(D):
    '''
    This function computes MDS with a distance matrix
    :param D: input distance matrix
    :return: V_tilde, whose rows are result configurations, and the
    eigenvalues
    '''
    P = D.shape[0]  # D is a symmetric distrance matrix
    A = -0.5 * D**2
    H = np.eye(P) - np.ones((P, P))/P
    B = H@A@H
    w, V = np.linalg.eig(B)
    V = V[:, np.flip(np.argsort(w), axis=0)]
    w = w[np.flip(np.argsort(w), axis=0)]
    V = V[:, w>0]
    w = w[w>0]
    V_tilde = V * np.sqrt(w[None, :])

        return V_tilde, w
```

We can compute city coordinates using MDS from distance matrix, as is shown below:

```
M = np.array([[0,244,218,284,197,312,215,469,166,212,253,270],
[0,0,350,77,167,444,221,583,242,53,325,168],
[0,0,0,369,347,94,150,251,116,298,57,284],
[0,0,0,0,242,463,263,598,257,72,340,164],
[0,0,0,0,0,441,279,598,269,170,359,277],
[0,0,0,0,0,0,245,169,210,392,143,378],
[0,0,0,0,0,0,0,380,55,168,117,143],
[0,0,0,0,0,0,0,0,349,531,264,514],
[0,0,0,0,0,0,0,0,0,190,91,173],
[0,0,0,0,0,0,0,0,0,0,273,111],
[0,0,0,0,0,0,0,0,0,0,0,256],
[0,0,0,0,0,0,0,0,0,0,0,0]])
D = M+M.T

# compute results
res, evals = MDS(D)

# plot cities
cities = ['Inverness', 'Glasgow', 'Newcastle', 'Carlisle', 'Leeds', 'Hull',
    'Norwich', 'Aberystwyth', 'London', 'Dover', 'Brighton', 'Exeter']
cities = sorted(cities)
plt.plot(res[:, 0], res[:, 1], 'r.')
for i in range(res.shape[0]):
    plt.text(res[i, 0]+1, res[i, 1]+1, cities[i])
plt.show()
```

The created map is shown in figure 1. It is similar to the actual map of UK shown in figure 2.
The mean of the configuration is verified to be 0.

```
>>>print(np.mean(res, 0))
[  7.10542736e-15  -3.43428989e-14   4.26325641e-14  -5.16623781e-14
   5.53631215e-14  -2.04281037e-14   1.05545202e-13   1.95339693e-12]
```

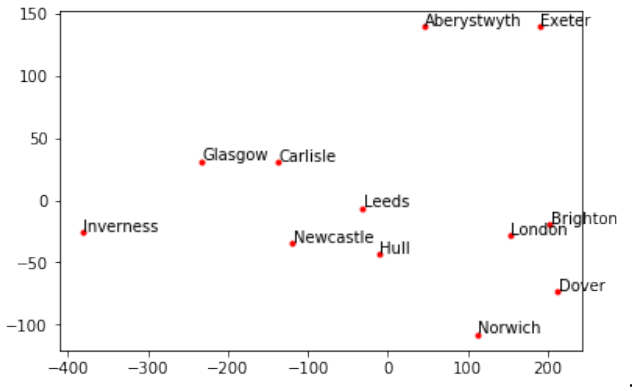

Figure 1: The city map based on generated coordinates.

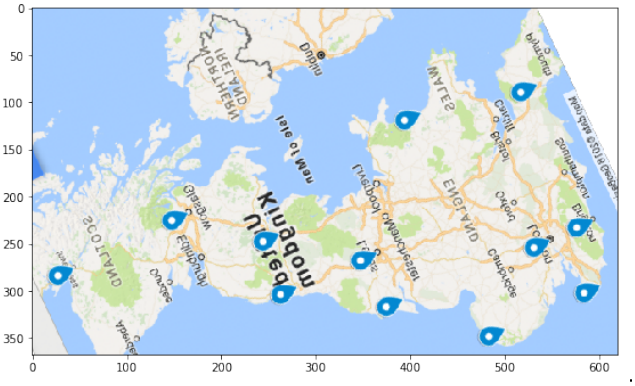

Figure 2: An actual UK map showing cities in the problem set.

We can also plot the eigenvalues of B using the following code:

```
plt.plot(evals, '.')
plt.show()
```

The eigenvalues is shown in figure 3. From it we can see that the first two dimensions are already sufficient to closely approximate the point configurations.

Problem 2

In this problem, I removed the city Aberystwyth, and used LMDS to compute the coordinate of Aberystwyth. The following function implements the LMDS algorithm.

```
def LMDS(D, dist_a, V_tilde_inv):
    '''
    :param D: the landscape marks distance matrix (not squared)
    :param dist_a: the distance vector from new point to landscape points(not squared)
    :param V_tilde_inv: the pseudo inverse of V_tilde
    :return: the configuration of the new point
    '''
    P = D.shape[0]  # dimension of data
    d_mean = (np.mean(D**2, axis= 1)).reshape(P, 1)
    d_a = (dist_a**2).reshape(P, 1)
    return -0.5*(V_tilde_inv.T@(d_a - d_mean))
```

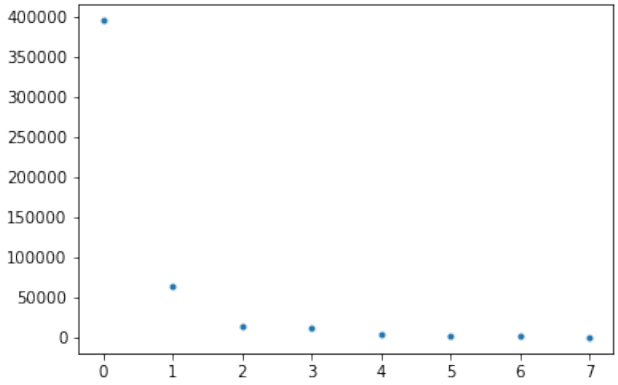

Figure 3: The eigenvalues of B.

The following code removes the city Aberystwyth and creates a new distance matrix. MDS was used to recreate the configurations based on the new distance matrix. The new map is also plotted as shown in figure 4. Note that in figure 4, blue dots were coordinates from last problem, and red dots are new coordinates computed without Aberystwyth. There are some minor differences between two sets of configurations.

```
D_new = np.delete(D, [0], 0)
D_new = np.delete(D_new, [0], 1)
V_tilde, lamb = MDS(D_new)

# plot new map
plt.plot(V_tilde[:, 0], V_tilde[:, 1], 'r.')
plt.plot(-res[:, 0], res[:, 1], 'b.')
for i in range(V_tilde.shape[0]):
    plt.text(V_tilde[i, 0]+1, V_tilde[i, 1]+1, cities[i+1])
plt.gca().invert_xaxis()
plt.show()
```

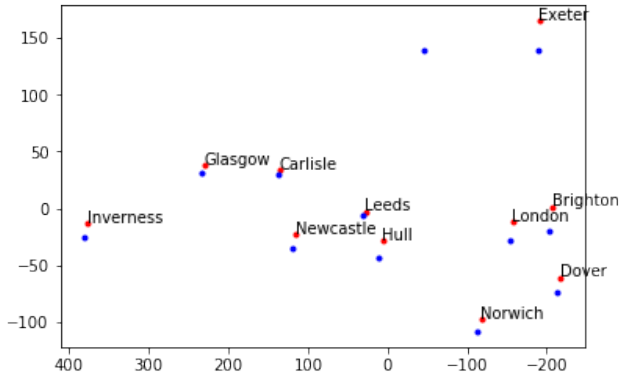

Figure 4: New map without Aberystwyth.

Now LMDS is used to compute the configuration of Aberystwyth, and plot it.

```
V_tilde_inv = V_tilde / lamb[None, :]
x_a = LMDS(D_new, D[1:, 0], V_tilde_inv)

```

```
# plot new configuration
plt.plot(V_tilde[:, 0], V_tilde[:, 1], 'r.')
plt.plot(-res[:, 0], res[:, 1], 'b.')
for i in range(V_tilde.shape[0]):
  plt.text(V_tilde[i, 0]+1, V_tilde[i, 1]+1, cities[i+1])
plt.gca().invert_xaxis()
plt.plot(x_a[0, 0], x_a[1, 0], 'c*')
plt.show()
```

As is shown in figure 5, the configuration of Aberystwyth (shown as cyan star)is successfully recreated with some minor differences.

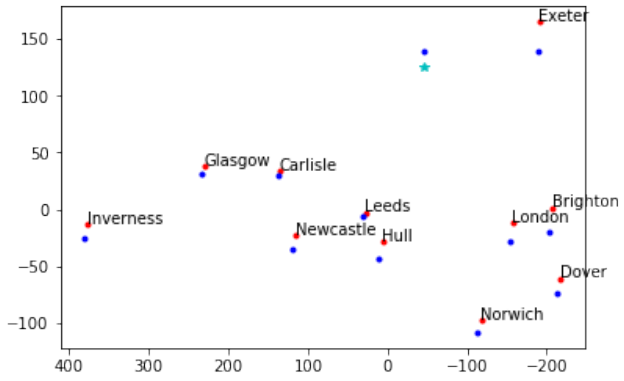

Figure 5: Map with LMDS-recreated Aberystwyth coordinate.

In addition, we can quantify the loss of reconstructing the configuration points with 2-dimensions using the following code. The loss is defined as $\frac{\|D'-D\|}{\|D\|}$, where D' is the constructed distance matrix, and D is the original matrix. MDS method has a loss of 0.051, and LMDS method has a loss of 0.061. The losses are relatively small, and MDS with all points is slightly better than LMDS which left one point out.

```
from scipy.spatial import distance_matrix
D_MDS = distance_matrix(res[:,:2], res[:,:2])
new_config = np.vstack((x_a[:2].reshape(1,2), V_tilde[:, :2]))
D_LMDS = distance_matrix(new_config, new_config)

print(np.linalg.norm(D_MDS-D)/np.linalg.norm(D)) #0.051
print(np.linalg.norm(D_LMDS-D)/np.linalg.norm(D)) #0.061
```