Untitled10

October 27, 2019

1 Problem 2

[143]:

```
import numpy as np
from scipy import linalg
import scipy

def GaussianKernel(x,y,sigma):
    G = np.exp((-np.linalg.norm(x-y)**2)/(2*sigma**2))
    return G

def PolynomialKernel(x,y,a):
    P = (x.dot(y))**a
    return P

def KappaMatrix(X,kernel,tunPara):
    row = X.shape[0]
    col = X.shape[1]
    a = np.ones(col).reshape(col,1)
    H = np.identity(col) - np.matmul(a,a.T)/col

    if kernel == "Polynomial":
        X = X.dot(H)

    K = np.zeros((col,col))

    for i in range(col):
        xx = X[:,i]
        for j in range(col):
            yy = X[:,j]
            if kernel == "GaussianKernel":
                K[i,j] = GaussianKernel(xx,yy,tunPara)
            elif kernel == "Polynomial":
                K[i,j] = PolynomialKernel(xx,yy,tunPara)
            else :
                print ("Not valid kernel")
                return
                K[j,i] = K[i,j]
```

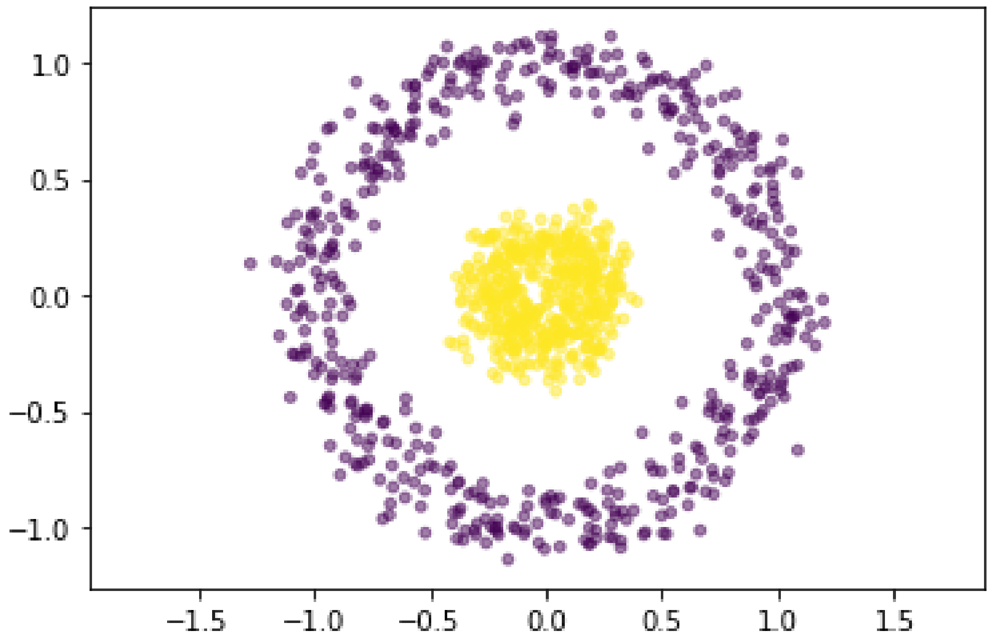
```
    K = H.dot(K.dot(H))
    return K

def KernelPCA(kappa,d):
    u,s,v = np.linalg.svd(kappa)

    Vtmp = v.T[:,0:d]
    s = np.diag(s)
    Sigma = scipy.linalg.sqrtm(s[0:d,0:d])
    Y = np.matmul(Sigma,Vtmp.T)
    return Sigma, Vtmp, Y
```

2 Problem 3

2.0.1 (1)

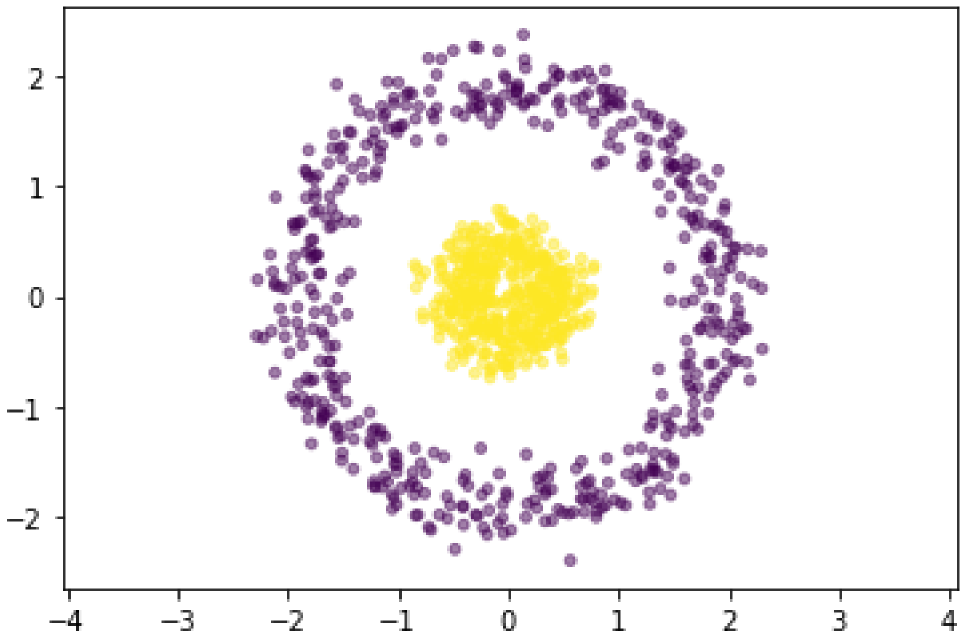
[47]:
```
import csv
import pandas as pd
import matplotlib.pyplot as plt

X = pd.read_csv("HW6_dat.csv").as_matrix()
X12= X[:,0:2]
X1 = X[:,0]
X2 = X[:,1]
y  = X[:,2]

from sklearn import preprocessing

X_scaled = preprocessing.scale(X12)
X1 = X12[:,0]
X2 = X12[:,1]

fig = plt.figure()

plt.scatter(X1,X2,alpha=0.5,c=y,s=15)
plt.axis('equal')
plt.show()
```

```
/Users/boyaozhu/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:5:
FutureWarning: Method .as_matrix will be removed in a future version. Use
.values instead.
  """
```



2.0.2 (2)

[159]:
```
eigval, eigvec = np.linalg.eig(np.cov(X_scaled.T))

Z12 = X_scaled.dot(eigvec)
Z1 = Z12[:,0]
Z2 = Z12[:,1]

fig = plt.figure()

plt.scatter(Z1,Z2,alpha=0.5,c=y,s=15)
plt.axis('equal')
plt.show()
```

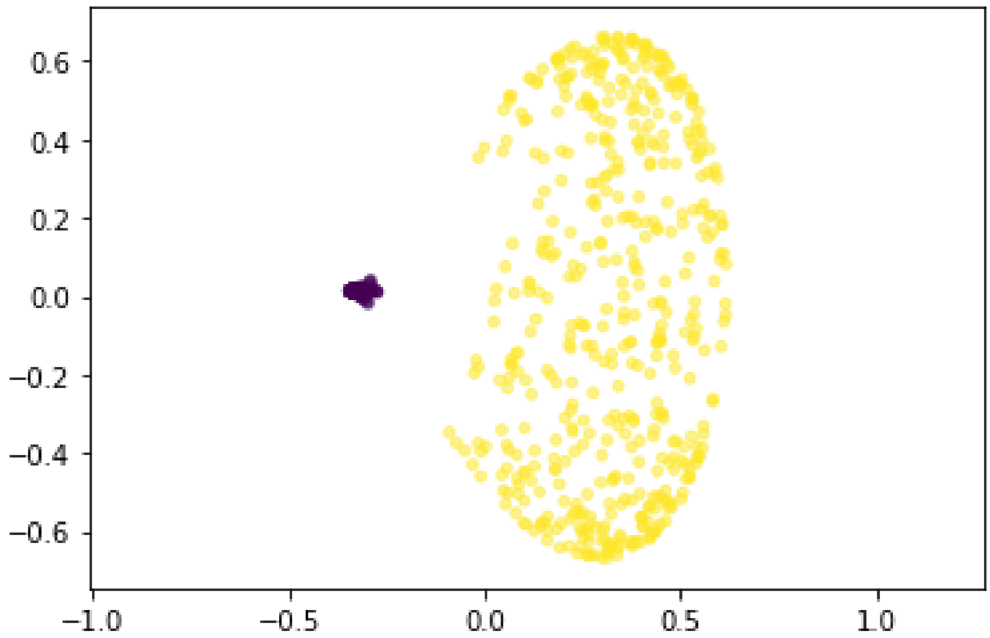


The PCA plot amounts to simply a rotation of the original data plot, and the data points are not linearly separable.

2.0.3 (3)

Gaussian Kernel

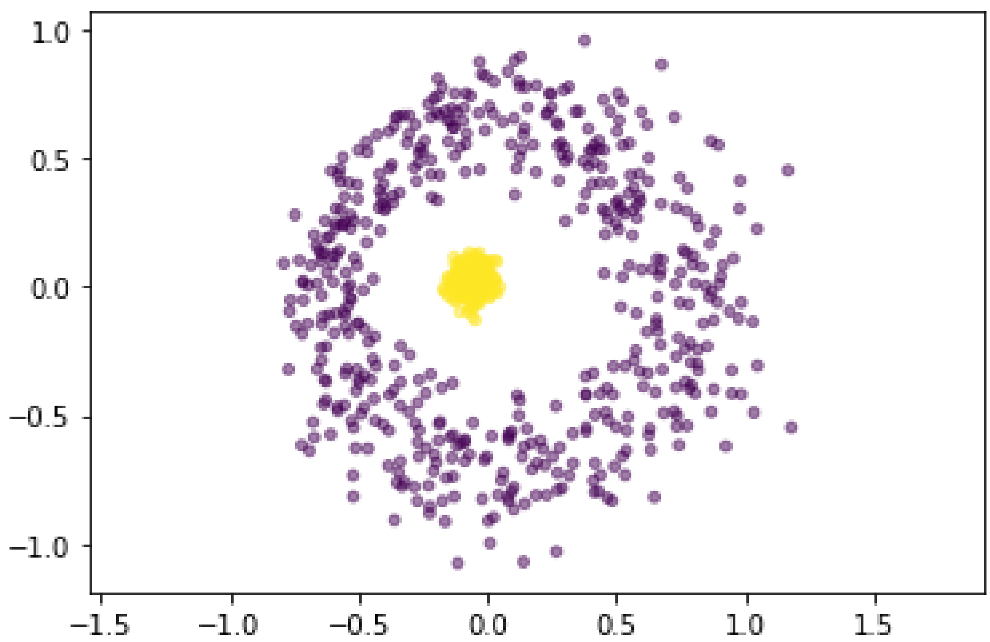
[147]:
```
K = KappaMatrix(X12.T,"GaussianKernel",0.2)

S,V,Y = KernelPCA(K,2)

fig = plt.figure()

Z1 = Y[0,:]
Z2 = Y[1,:]
plt.scatter(Z1,Z2,alpha=0.5,c=y,s=15)
plt.axis('equal')
plt.show()
```



This is the plot of KPCA with Gaussian kernal using the 1st and 2nd principal components. The data points with different colors have distinctly different first principal components. It is possible to separate the two groups by drawing a straight line on the plot.

Polynomial Kernel

[148]:
```
K = KappaMatrix(X12.T,"Polynomial",2)

S,V,Y = KernelPCA(K,2)

fig = plt.figure()

Z1 = Y[0,:]
Z2 = Y[1,:]
plt.scatter(Z1,Z2,alpha=0.5,c=y,s=15)
plt.axis('equal')
plt.show()
```



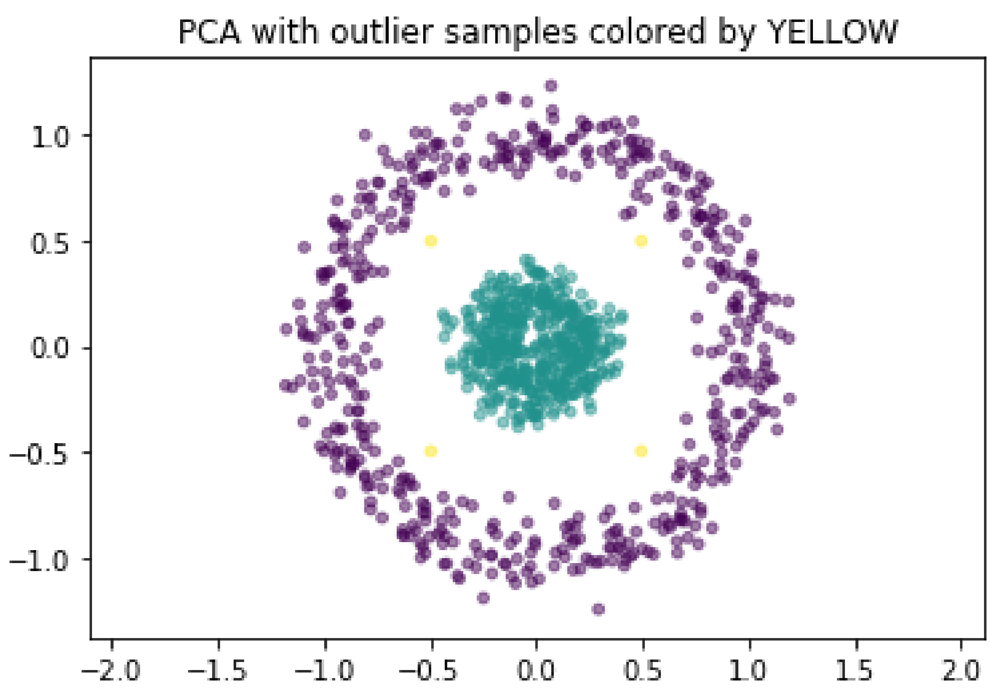
This is the plot of KPCA with polynomial kernel using the 1st and 2nd principal components. The first two principal components are not enough to separate the two groups of data points linearly.

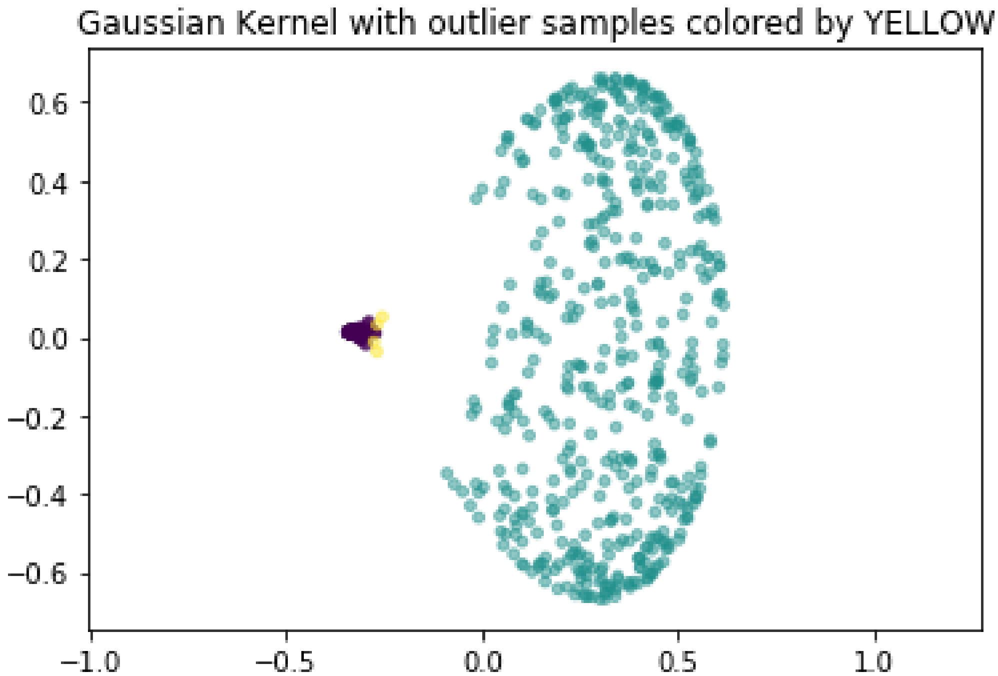
2.0.4 (4)

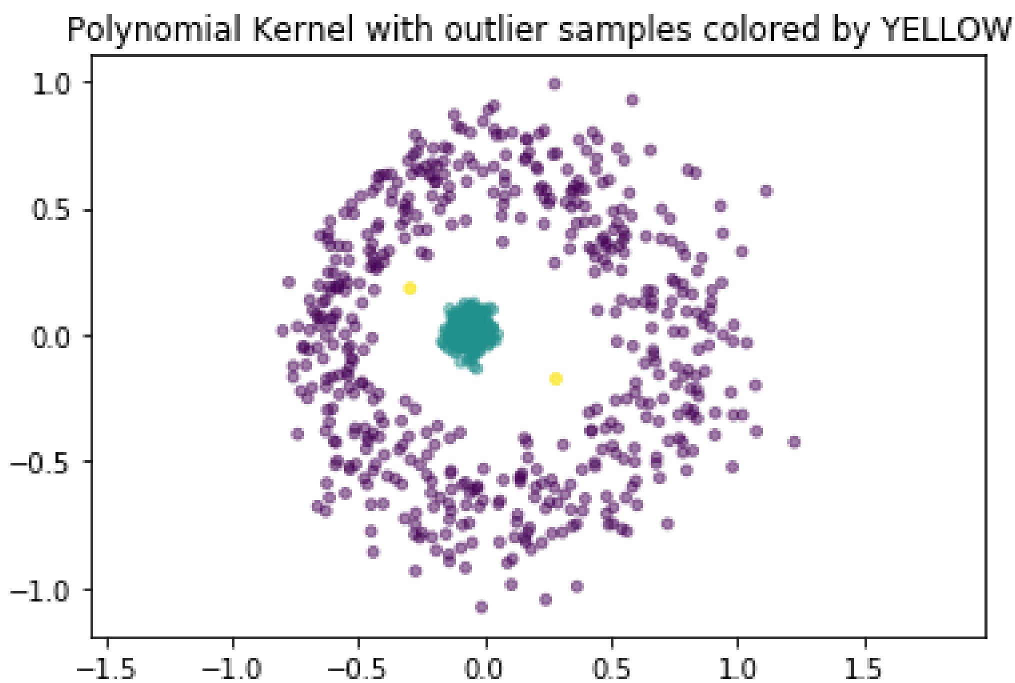
[178]:
```
X = pd.read_csv("HW6_dat.csv").as_matrix()
outPoint = np.array([[0,-0.7,2],[0,0.7,2],[0.7,0,2],[-0.7,0,2]])
yy = outPoint[:,2]
X = np.vstack((X,outPoint))

X12 = X[:,0:2]
y   = X[:,2]
# project first principal component learned from PCA
Comp12 = X12.dot(eigvec)
Comp1 = Comp12[:,0]
Comp2 = Comp12[:,1]

#PCA with outlier sample points colored by yellow
fig = plt.figure()
plt.scatter(Comp1,Comp2,alpha=0.5,c=y,s=15)
plt.axis('equal')
plt.title("PCA with outlier samples colored by YELLOW")
plt.show()
```

```python
# Project first two PC learned from Kernel
K = KappaMatrix(X12.T,"GaussianKernel",0.2)
S,V,Y = KernelPCA(K,2)
fig = plt.figure()
Comp1 = Y[0,:]
Comp2 = Y[1,:]
plt.scatter(Comp1,Comp2,alpha=0.5,c=y,s=15)
plt.axis('equal')
plt.title("Gaussian Kernel with outlier samples colored by YELLOW")
plt.show()


# Project first two PC learned from Kernel
K = KappaMatrix(X12.T,"Polynomial",2)
S,V,Y = KernelPCA(K,2)
fig = plt.figure()
Comp1 = Y[0,:]
Comp2 = Y[1,:]
plt.scatter(Comp1,Comp2,alpha=0.5,c=y,s=15)
plt.title("Polynomial Kernel with outlier samples colored by YELLOW")
plt.axis('equal')
plt.show()
```

```
/Users/boyaozhu/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:1:
FutureWarning: Method .as_matrix will be removed in a future version. Use
.values instead.
  """Entry point for launching an IPython kernel.
```




Gaussian Kernel with outlier samples colored by YELLOW
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−1.0
−0.5
0.0
0.5
1.0


Polynomial Kernel with outlier samples colored by YELLOW
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

3 Problem 4

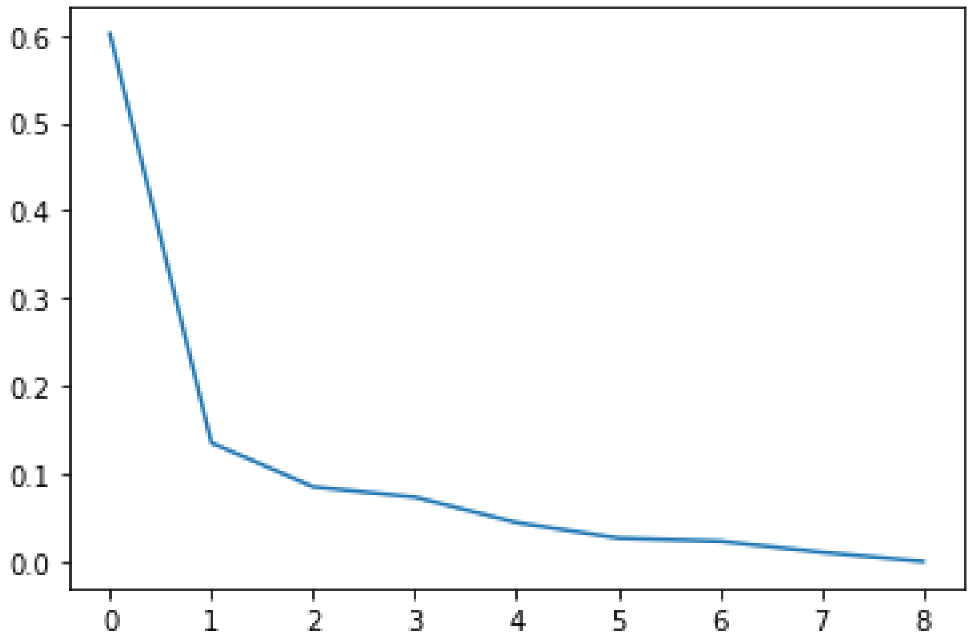
3.0.1 (1)

[210]:
```
D1 = np.array([[0     ,850.7 ,393.4 ,1765.9,2593  ,1259.7,187.4 ,2485.5,2694.1],
               [850.7 ,0     ,595.9 ,916.6 ,1742.3,1189.5,713.8 ,1731.2,1853.7],
               [393.4 ,595.9 ,0     ,1490.2,2296.5,927.1 ,206   ,2321.7,2435.7],
               [1765.9,916.6 ,1490.2,0     ,831.9 ,1723.5,1628.9,1016.7,946.6 ],
               [2593  ,1742.3,2296.5,831.9 ,0     ,2333.1,2448.8,959.9 ,342   ],
               [1259.7,1189.5,927.1 ,1723.5,2333.1,0     ,1092.3,2729.9,2588.9],
               [187.4 ,713.8 ,206   ,1628.9,2448.8,1092.3,0     ,2401.7,2567.4],
               [2485.5,1731.2,2321.7,1016.7,959.9 ,2729.9,2401.7,0     ,684.1 ],
               [2694.1,1853.7,2435.7,946.6 ,342   ,2588.9,2567.4,684.1 ,0     ␣
 ↪]])

print (D1)
```

```
[[   0.    850.7  393.4 1765.9 2593.  1259.7  187.4 2485.5 2694.1]
 [ 850.7    0.    595.9  916.6 1742.3 1189.5  713.8 1731.2 1853.7]
 [ 393.4  595.9    0.   1490.2 2296.5  927.1  206.  2321.7 2435.7]
 [1765.9  916.6 1490.2    0.    831.9 1723.5 1628.9 1016.7  946.6]
 [2593.  1742.3 2296.5  831.9    0.   2333.1 2448.8  959.9  342. ]
 [1259.7 1189.5  927.1 1723.5 2333.1    0.   1092.3 2729.9 2588.9]
 [ 187.4  713.8  206.  1628.9 2448.8 1092.3    0.   2401.7 2567.4]
 [2485.5 1731.2 2321.7 1016.7  959.9 2729.9 2401.7    0.    684.1]
 [2694.1 1853.7 2435.7  946.6  342.  2588.9 2567.4  684.1    0. ]]
```

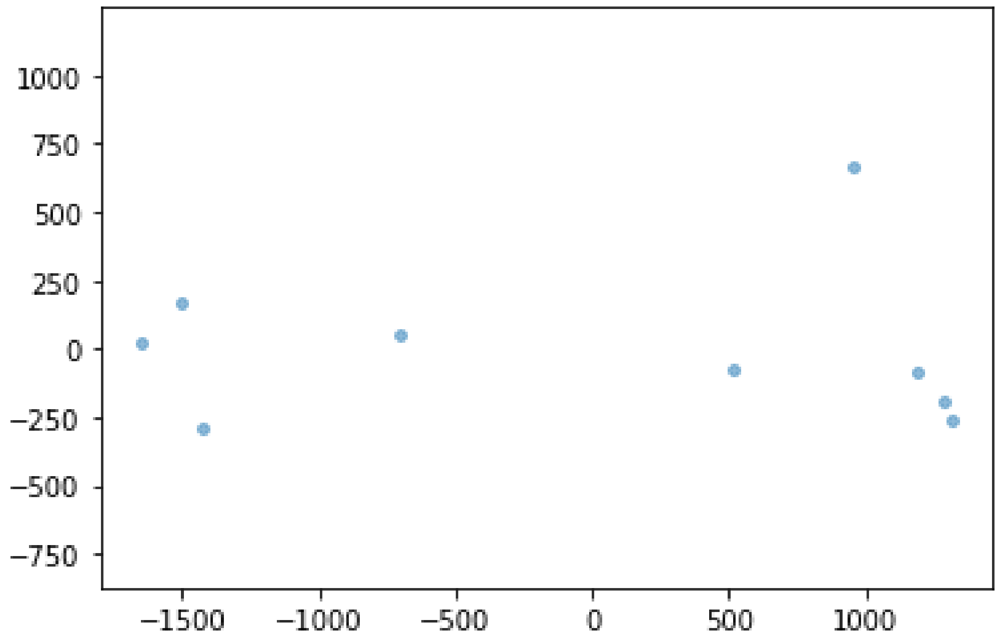
3.0.2 (2)

[234]:
```
identity = np.identity(D1.shape[0])
ones = np.ones(D1.shape[0])
h = identity-np.outer(ones,ones)/D1.shape[0]
b = -1/2*h.dot((D1).dot(h))



eigval, eigvec = np.linalg.eig(b)

eig_scale = np.flip(np.sort(eigval/np.sum(eigval)))

plt.plot(eig_scale)
plt.show()
```



3.0.3 (3)

[235]:
```
PC12 = b.dot(eigvec[:,0:2])
fig = plt.figure()
plt.scatter(PC12[:,0],PC12[:,1],alpha=0.5,s=15)
plt.axis('equal')
plt.show()
```

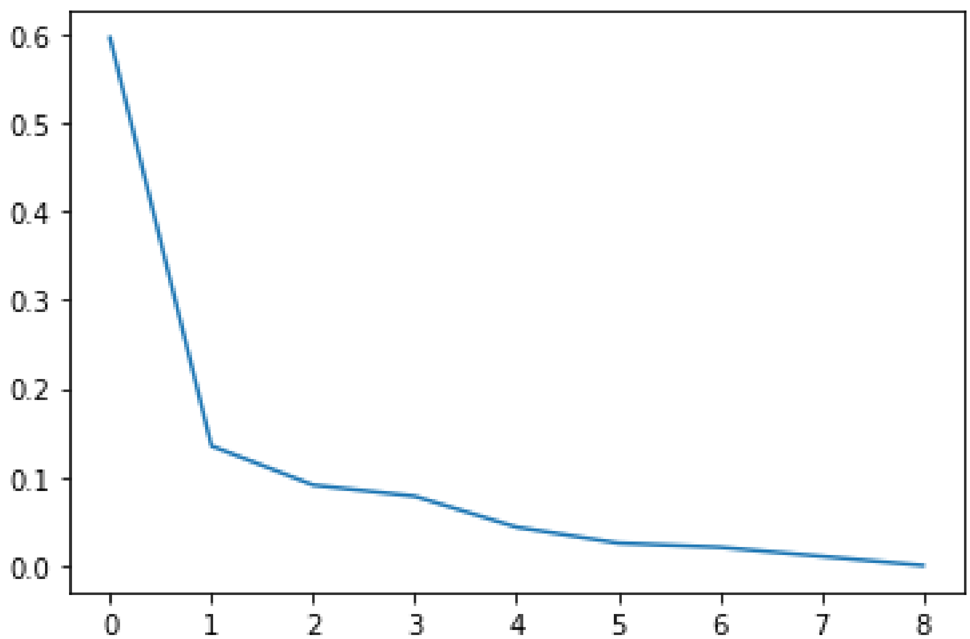


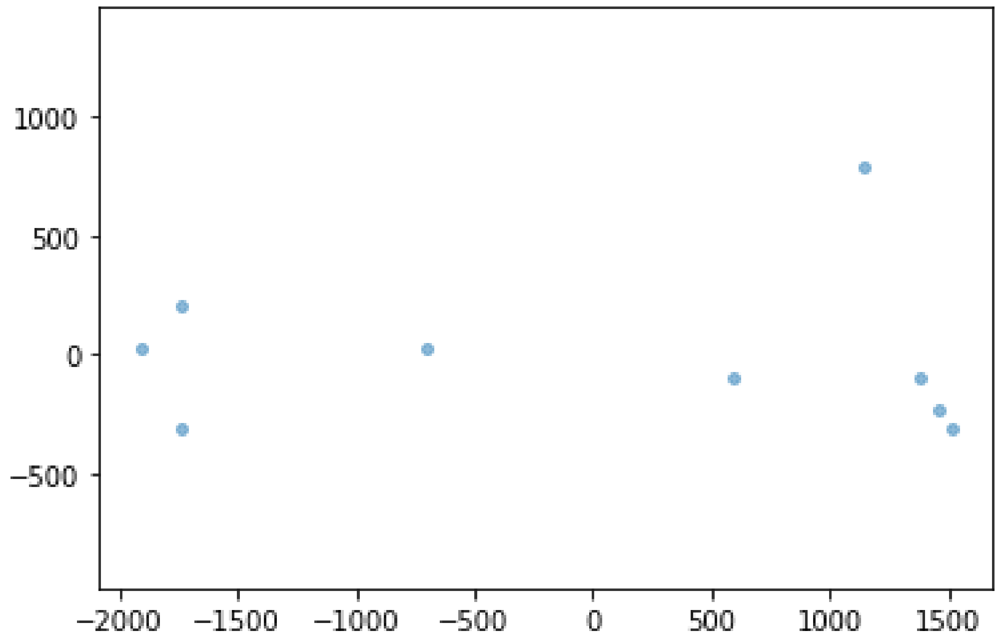
The relative positions and distances of the cities are well recovered.

3.0.4 (4)

[237]:
```
D1 = np.array([[0   ,983 ,441 ,1970,2983,1500,220 ,3054,3095],
               [983 ,0   ,699 ,1002,2015,1377,789 ,2064,2127],
               [441 ,699 ,0   ,1672,2669,1052,225 ,2767,2811],
               [1970,1002,1672,0   ,1017,2065,1779,1371,1255],
               [2983,2015,2669,1017,0   ,2734,2790,1135,382 ],
               [1500,1377,1052,2065,2734,0   ,1278,3300,3075],
               [220 ,789 ,225 ,1779,2790,1278,0   ,2861,2902],
               [3054,2064,2767,1371,1135,3300,2861,0   ,808 ],
               [3095,2127,2811,1255,382 ,3075,2902,808 ,0   ]])


identity = np.identity(D1.shape[0])
ones = np.ones(D1.shape[0])
h = identity-np.outer(ones,ones)/D1.shape[0]
b = -1/2*h.dot((D1).dot(h))


eigval, eigvec = np.linalg.eig(b)

eig_scale = np.flip(np.sort(eigval/np.sum(eigval)))
```

```
plt.plot(eig_scale)
plt.show()


PC12 = b.dot(eigvec[:,0:2])
fig = plt.figure()
plt.scatter(PC12[:,0],PC12[:,1],alpha=0.5,s=15)
plt.axis('equal')
plt.show()
```



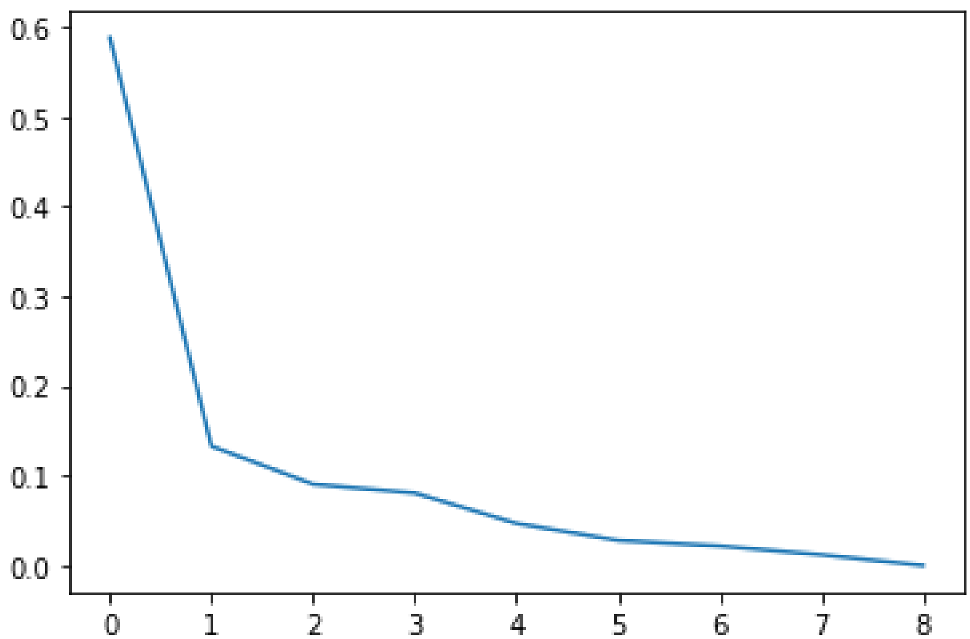


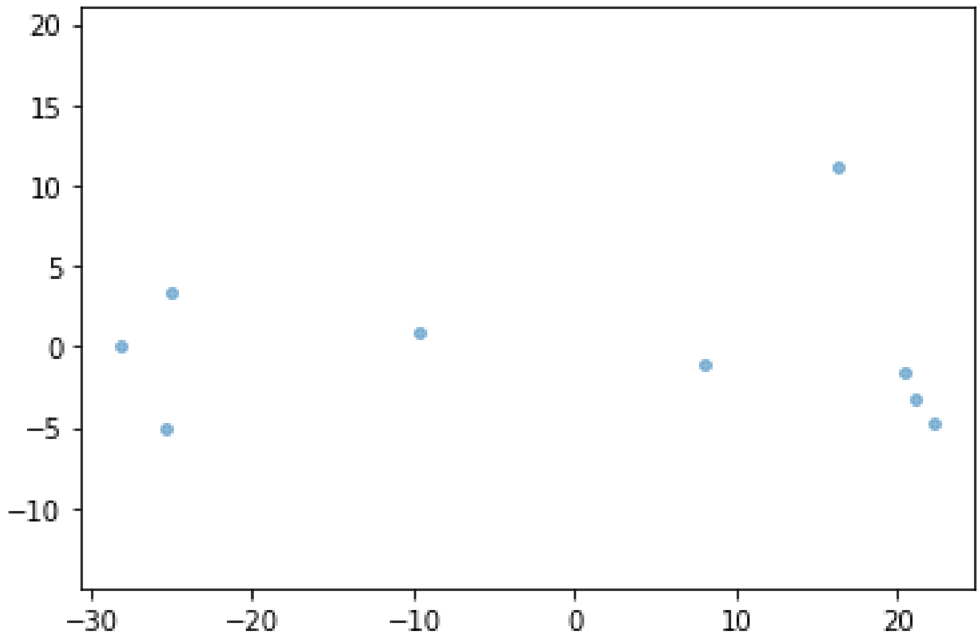
The relative positions and distances of the cities are well recovered. And we can see this is same as that of the air.

3.0.5 (5)

[238]:
```
D1 = np.array([[0   ,15   ,7.07 ,29   ,44   ,21.9 ,3.78 ,45   ,46   ],
               [15  ,0    ,10.9 ,14.87,29   ,19.82,12.6 ,30   ,31   ],
               [7.07,10.9 ,0    ,25   ,40   ,15.3 ,3.88 ,41   ,42   ],
               [29  ,14.87,25   ,0    ,15.83,29   ,26   ,20.4 ,19.18],
               [44  ,29   ,40   ,15.83,0    ,39   ,40   ,17.63,5.95 ],
               [21.9,19.82,15.3 ,29   ,39   ,0    ,18.6 ,48   ,45   ],
               [3.78,12.6 ,3.88 ,26   ,40   ,18.6 ,0    ,43   ,43   ],
               [45  ,30   ,41   ,20.4 ,17.63,48   ,43   ,0    ,12.28],
               [46  ,31   ,42   ,19.18,5.95 ,45   ,43   ,12.28,0    ]])

identity = np.identity(D1.shape[0])
ones = np.ones(D1.shape[0])
h = identity-np.outer(ones,ones)/D1.shape[0]
b = -1/2*h.dot((D1).dot(h))


eigval, eigvec = np.linalg.eig(b)

eig_scale = np.flip(np.sort(eigval/np.sum(eigval)))
```

```
plt.plot(eig_scale)
plt.show()


PC12 = b.dot(eigvec[:,0:2])
fig = plt.figure()
plt.scatter(PC12[:,0],PC12[:,1],alpha=0.5,s=15)
plt.axis('equal')
plt.show()
```





The relative positions and distances of the cities are well-recovered.

The three scatter plots of cities using classical MDS with different kinds of distances are very similar to one another, and the results agree very well with the actual geographical distances and positions. This is because all the distances used here are very close to the Euclidean distance on a 2D map.

[ ]: