In [ ]:
```
# Imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import scipy as sp
import scipy.sparse
import scipy.linalg as alg
import scipy.sparse.linalg as spl
import skimage.data
```

In [ ]:
```
# 1 For colab
# from google.colab import drive
# drive.mount('/content/drive')
# data = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/Statistica/Hom
```

In [ ]:
```
# 1 without colab
data = pd.read_csv("data.csv")
```

In [ ]:
```
print(data.head())
print(data.shape)
```

```
   label  pixel0  pixel1  pixel2  pixel3  pixel4  pixel5  pixel6  pixel7  \
0      1       0       0       0       0       0       0       0       0
1      0       0       0       0       0       0       0       0       0
2      1       0       0       0       0       0       0       0       0
3      4       0       0       0       0       0       0       0       0
4      0       0       0       0       0       0       0       0       0

   pixel8  ...  pixel774  pixel775  pixel776  pixel777  pixel778  pixel779
\
0       0  ...         0         0         0         0         0         0
1       0  ...         0         0         0         0         0         0
2       0  ...         0         0         0         0         0         0
3       0  ...         0         0         0         0         0         0
4       0  ...         0         0         0         0         0         0

   pixel780  pixel781  pixel782  pixel783
0         0         0         0         0
1         0         0         0         0
2         0         0         0         0
3         0         0         0         0
4         0         0         0         0

[5 rows x 785 columns]
(42000, 785)
```

In [ ]:
```
# Utils

def split(X, Y, Ntrain):
    _, N = X.shape
    totalIdxs = np.arange(N)
    np.random.shuffle(totalIdxs)
    indecesTrain = totalIdxs[:Ntrain]
    indecesTest = totalIdxs[Ntrain:]
    Xtrain = X[:, indecesTrain]
    Ytrain = Y[indecesTrain]
    Xtest = X[:, indecesTest]
    Ytest = Y[indecesTest]
```

```
    return (Xtrain, Ytrain, Xtest, Ytest)

def get_centroid(X):
    return np.mean(X, axis=1)

def get_clusters(Z, Y, choosen_numbers):
  clusters = list()
  for i in choosen_numbers:
    Ii = (Y==i)
    clusters.append(Z[:,Ii])
  return clusters

def concatenation(X_set,Y_set, chosen_indexes,):
  I = []
  chosen_X = []
  chosen_Y = []

  d, N = X_set.shape
  for element in chosen_indexes:
    I.append((Y_set == element))

  for i in range(len(chosen_indexes)):
    chosen_X.append(X_set[:,I[i]])

  for i in range(len(chosen_indexes)):
    chosen_Y.append(Y_set[I[i]])

  X_conc = np.concatenate(chosen_X, axis=1)
  Y_conc = np.concatenate(chosen_Y)
  return chosen_X, chosen_Y, X_conc, Y_conc

def distances(Z,Y_conc, choosen_numbers, type):
  clusters = get_clusters(Z, Y_conc, choosen_numbers)

  centroids = []
  for cluster in clusters:
    centroids.append(get_centroid(cluster))

  print("Distances from centroids:")
  for j in range(len(choosen_numbers)):
    tot = 0
    for i in range(0,len(clusters[j])):
        tot = tot + np.linalg.norm(clusters[j][:,i] - centroids[j])
    dist = tot / len(clusters[j])
    print("Average distance between elements in cluster " + str(choosen_numl

def accuracy(X, Y, P, choosen_numbers, clusters, type):
  acc = 0
  centroids = []
  for cluster in clusters:
    centroids.append(get_centroid(cluster))
  print()
  print("Accuracy in classification:")
  for i in range(X.shape[1]):
    x = X[:, i]
    projected_x = P @ x
    min_dist = np.linalg.norm(projected_x - centroids[0])
    selected_cluster = choosen_numbers[0]
    for j,num in enumerate(choosen_numbers):
      if(np.linalg.norm(projected_x - centroids[j]) < min_dist):
        min_dist = np.linalg.norm(projected_x - centroids[j])
```

```
        selected_cluster = num
    if(selected_cluster == Y[i]):
      acc += 1
  print("Total test for accuracy: ", X.shape[1])
  print("Toatal right guess for algorithm ", type, "= ", acc)
  acc /= X.shape[1]
  print("Accuracy is: ", acc*100, "%")



def plot(Z, Y_conc, choosen_numbers, k, limit_axes=False):
  clusters = list()
  for i in choosen_numbers:
    Ii = (Y_conc==i)
    clusters.append(Z[:,Ii])
  centroids = list()
  for cluster in clusters:
    centroids.append(get_centroid(cluster))
  if(k == 2):
    plt.scatter(Z[0,:], Z[1,:], c=Y_conc)
    if (limit_axes):
      plt.xlim(-1,1)
      plt.ylim(-1,1)
    for element in centroids:
        plt.scatter(element[0], element[1], marker="x", color='r')
    plt.show()
  elif(k == 3):
    fig = plt.figure()
    ax = plt.axes(projection='3d')
    ax.scatter(Z[0,:], Z[1,:], Z[2,:], c=Y_conc)
    if (limit_axes):
      ax.set_xlim(-1,1)
      ax.set_ylim(-1,1)
      ax.set_zlim(-1,1)
    for element in centroids:
        ax.scatter(element[0], element[1],element[2], color='r')
    plt.show()
  else:
    pass
```

In [ ]:
```
# 2
data = np.array(data)
data.shape
X = data[:, 1:].T
Y = data[:, 0]

print(X.shape)
```

```
(784, 42000)
```

In [ ]:
```
# PCA (def)

def PCA(X_set, Y_set, chosen_indexes, k):
    chosen_X, chosen_Y, X_conc, Y_conc = concatenation(X_set, Y_set, chosen_
    centroid = get_centroid(X_conc)
    centroid = np.reshape(centroid, (len(centroid), 1))
    Xc = X_conc - centroid
    U, S, Vh = np.linalg.svd(Xc, full_matrices=False)
    Uk = np.resize(U, (len(U), k))
    return Uk.T@Xc, Uk.T, Y_conc
```

In [ ]:
```
# LDA (def)

def LDA(X_set,Y_set, chosen_indexes, k):
    d, N = X_set.shape
    chosen_X, chosen_Y, X_conc, Y_conc = concatenation(X_set,Y_set, chosen_

    centroids = []
    Xcs = []
    CX = get_centroid(X_conc)

    for element in chosen_X:
        centroids.append(get_centroid(element).reshape((d,1)))

    for i in range(len(chosen_indexes)):
        Xcs.append(chosen_X[i] - centroids[i])

    Xw = np.concatenate(Xcs, axis=1)
    Sw = Xw @ Xw.T

    X_s = []
    for i in range(len(centroids)):
        X_s.append(np.repeat(centroids[i], chosen_X[i].shape[1], axis=1))

    X_ = np.concatenate(X_s, axis=1)
    X_c = X_ - CX.reshape((d, 1))
    Sb = X_c @ X_c.T

    L = []
    try:
        L = np.linalg.cholesky(Sw)
    except:
        eps = 1e-6
        Sw = Sw + eps * np.eye(Sw.shape[0])
        L = np.linalg.cholesky(Sw)

    W = np.linalg.inv(L) @ Sb @ L
    _, W = spl.eigs(W,k=k)
    W = np.real(W)
    Q = np.linalg.inv(L).T @ W
    Z = Q.T @ X_conc
    return Z, Q.T, Y_conc
```

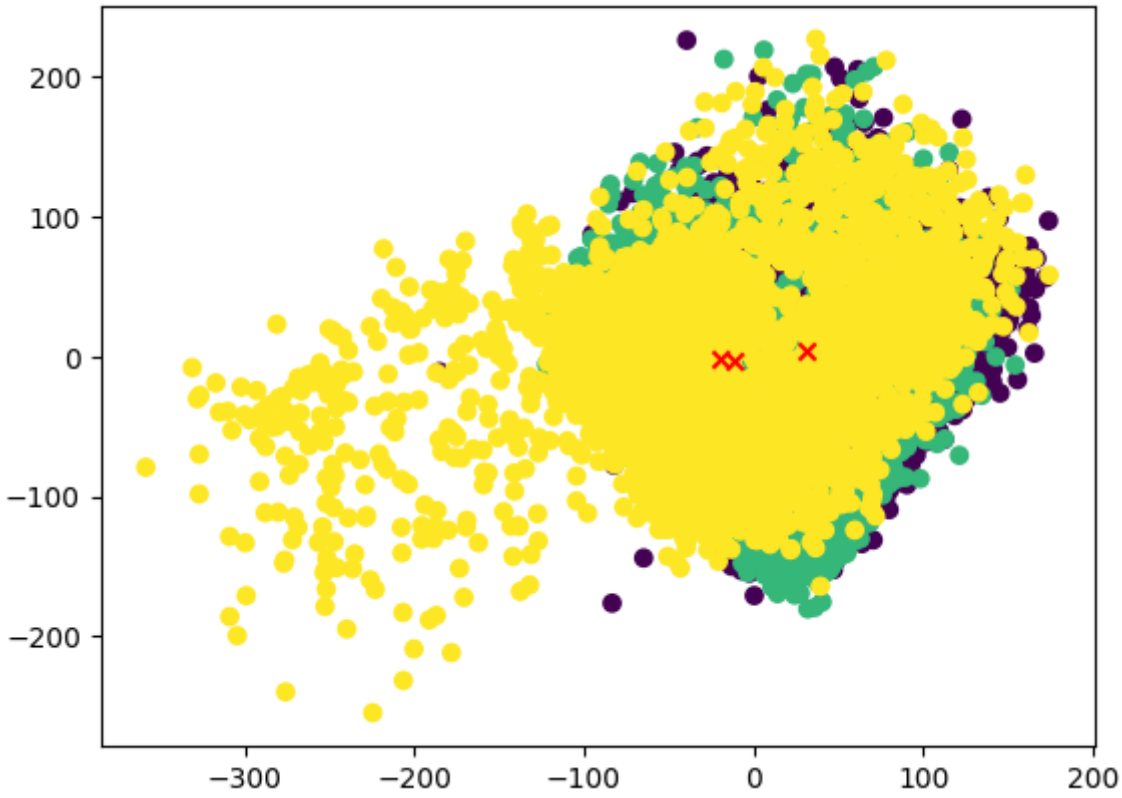
In [ ]:
```
# Definition of train-data and test-data

Ntrain = 35000
Xtrain, Ytrain, Xtest, Ytest = split(X, Y, Ntrain)
print(Xtrain.shape, Xtest.shape)
```

```
(784, 35000) (784, 7000)
```

Exercises for PCA

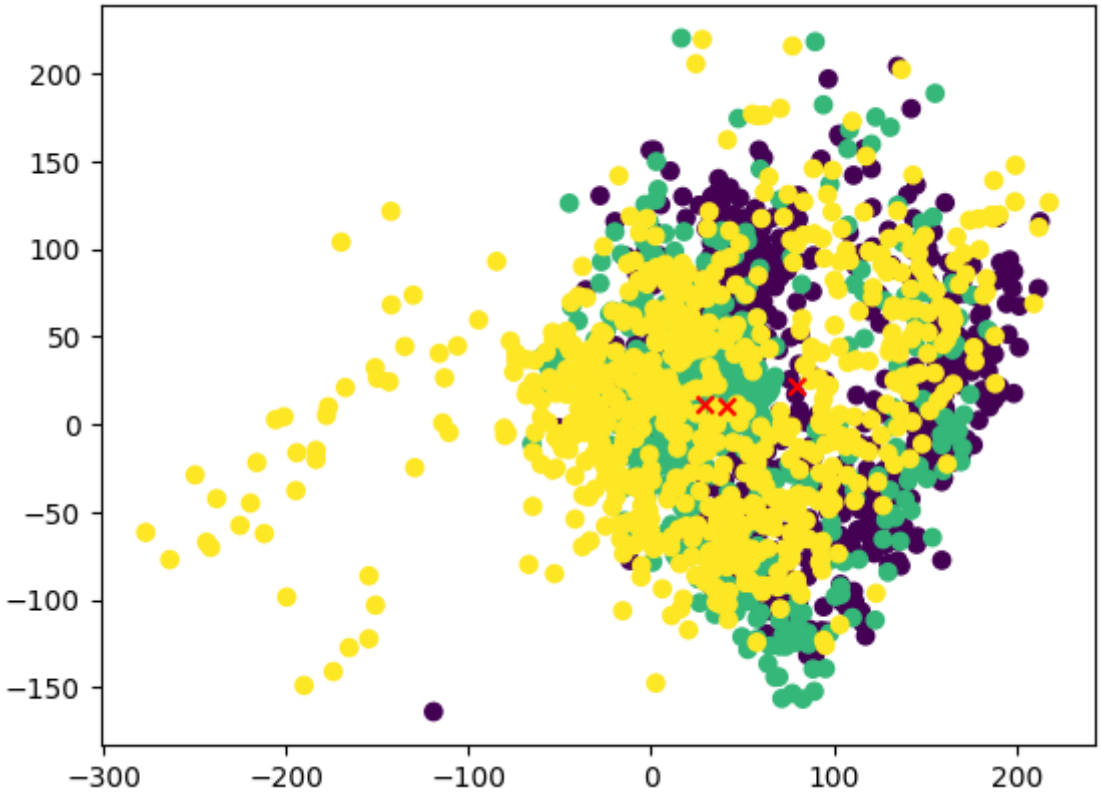
In [ ]:
```
# PCA (example)

chosen_indexes = [0,6,9]
k = 2
Z_pca, projection_pca, Y_conc = PCA(Xtrain, Ytrain, chosen_indexes, k)
clusters = get_clusters(Z_pca, Y_conc, chosen_indexes)
plot(Z_pca, Y_conc, chosen_indexes, k)
distances(Z_pca, Y_conc, chosen_indexes, "pca")
```

```
chosen_X, chosen_Y, X_conc, Y_conc = concatenation(Xtrain, Ytrain, chosen_i
accuracy(X_conc, Y_conc, projection_pca, chosen_indexes, clusters, "pca")
```



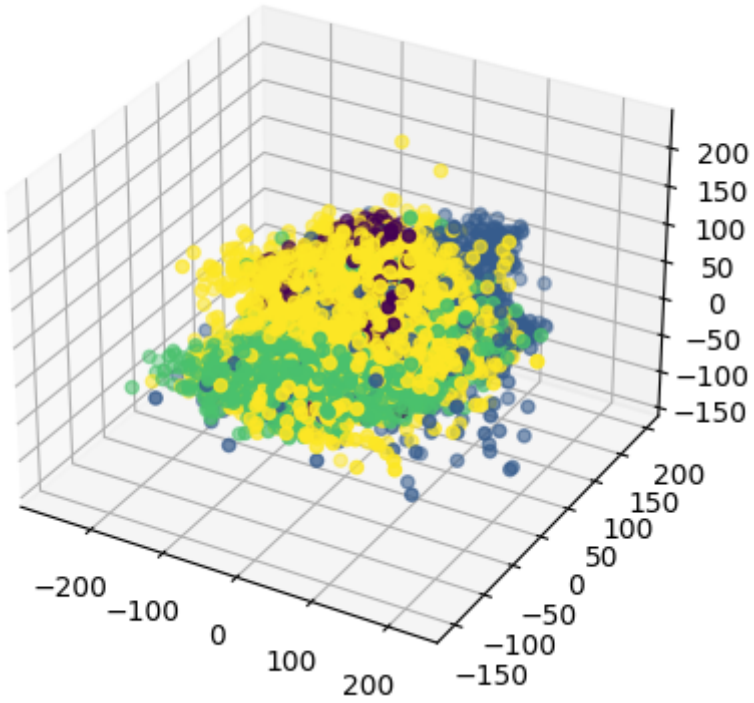

```
Distances from centroids:
Average distance between elements in cluster 0 from pca and centroid:  58.1
3125184786941
Average distance between elements in cluster 6 from pca and centroid:  49.3
9049876506543
Average distance between elements in cluster 9 from pca and centroid:  122.
27005802126033

Accuracy in classification:
Total test for accuracy:  10344
Toatal right guess for algorithm  pca =  4649
Accuracy is:  44.943928847641146 %
```

In [ ]:
```
# PCA (example on test set)

chosen_X, chosen_Y, X_conc, Y_conc = concatenation(Xtest, Ytest, chosen_inde
Ztest_pca = projection_pca @ X_conc
clusters = get_clusters(Ztest_pca, Y_conc, chosen_indexes)
plot(Ztest_pca, Y_conc, chosen_indexes, k)
distances(Ztest_pca, Y_conc, chosen_indexes, "pca")
accuracy(X_conc, Y_conc, projection_pca, chosen_indexes, clusters, "pca")
```




```
Distances from centroids:
Average distance between elements in cluster 0 from pca and centroid:  72.1
5933434157199
Average distance between elements in cluster 6 from pca and centroid:  63.9
0079322300187
Average distance between elements in cluster 9 from pca and centroid:  77.4
1219876935413

Accuracy in classification:
Total test for accuracy:  2113
Toatal right guess for algorithm  pca =  978
Accuracy is:  46.28490298154283 %
```

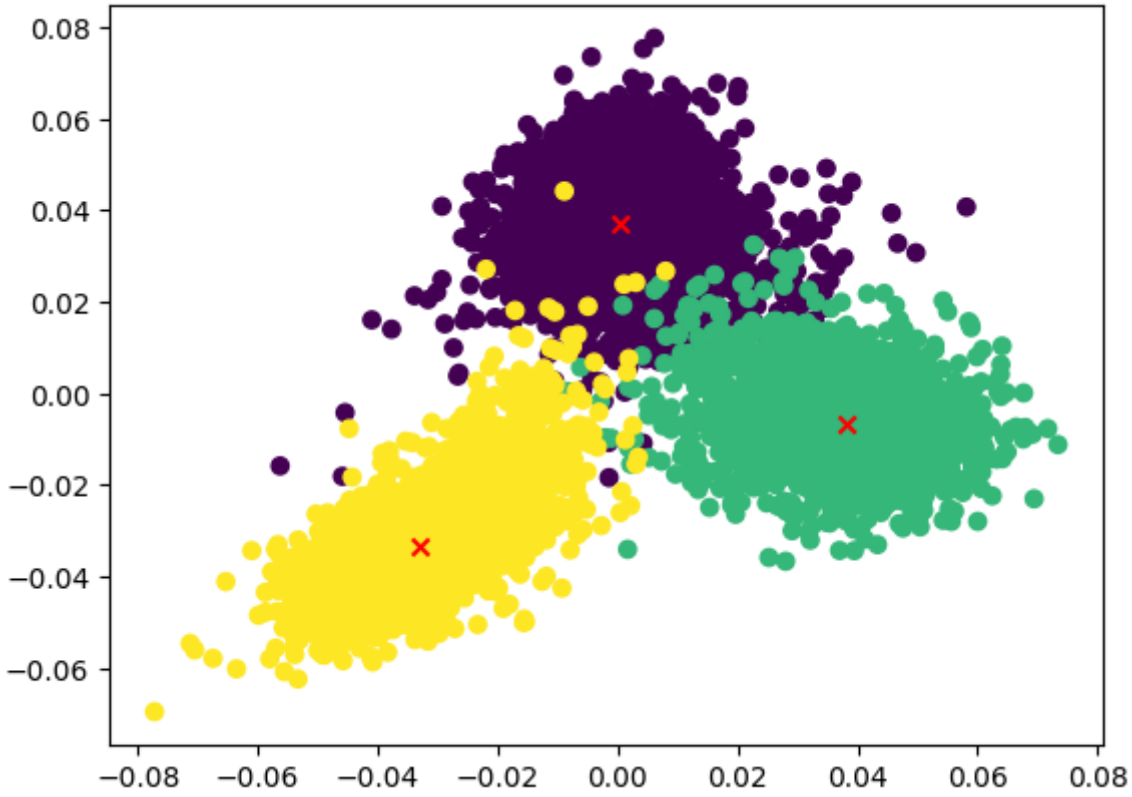
In [ ]:
```
# PCA (example with k=3 and different/more digits)

chosen_indexes = [1,3,6,8]
k = 3
Z_pca, projection_pca, Y_conc = PCA(Xtrain, Ytrain, chosen_indexes, k)
plot(Z_pca, Y_conc, chosen_indexes, k)
clusters = get_clusters(Z_pca, Y_conc, chosen_indexes)
distances(Z_pca, Y_conc, chosen_indexes, "pca")
chosen_X, chosen_Y, X_conc, Y_conc = concatenation(Xtrain, Ytrain, chosen_i
print()
accuracy(X_conc, Y_conc, projection_pca, chosen_indexes, clusters, "pca")
```




```
Distances from centroids:
Average distance between elements in cluster 1 from pca and centroid:  65.4
2728398034431
Average distance between elements in cluster 3 from pca and centroid:  110.
76408402212355
Average distance between elements in cluster 6 from pca and centroid:  55.7
53352423614395
Average distance between elements in cluster 8 from pca and centroid:  78.4
1036620443828


Accuracy in classification:
Total test for accuracy:  14325
Toatal right guess for algorithm  pca =  6401
Accuracy is:  44.68411867364747 %
```

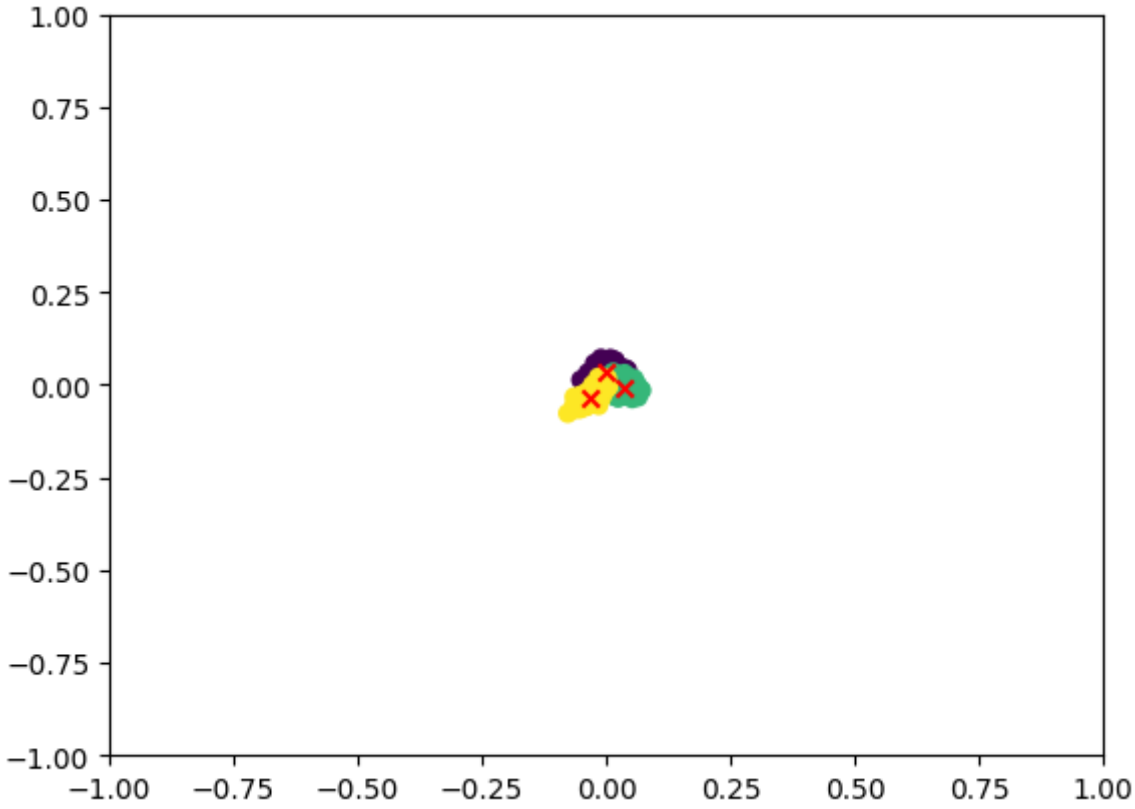
Exercises for LDA

In [ ]:
```
# LDA (example)

chosen_indexes = [0,6,9]
k = 2
Z_lda, projection_lda, Y_conc = LDA(Xtrain, Ytrain, chosen_indexes, k)
clusters = get_clusters(Z_lda, Y_conc, chosen_indexes)
plot(Z_lda, Y_conc, chosen_indexes, k)
distances(Z_lda, Y_conc, chosen_indexes, "lda")
chosen_X, chosen_Y, X_conc, Y_conc = concatenation(Xtrain, Ytrain, chosen_i
accuracy(X_conc, Y_conc, projection_lda, chosen_indexes, clusters, "lda")
```



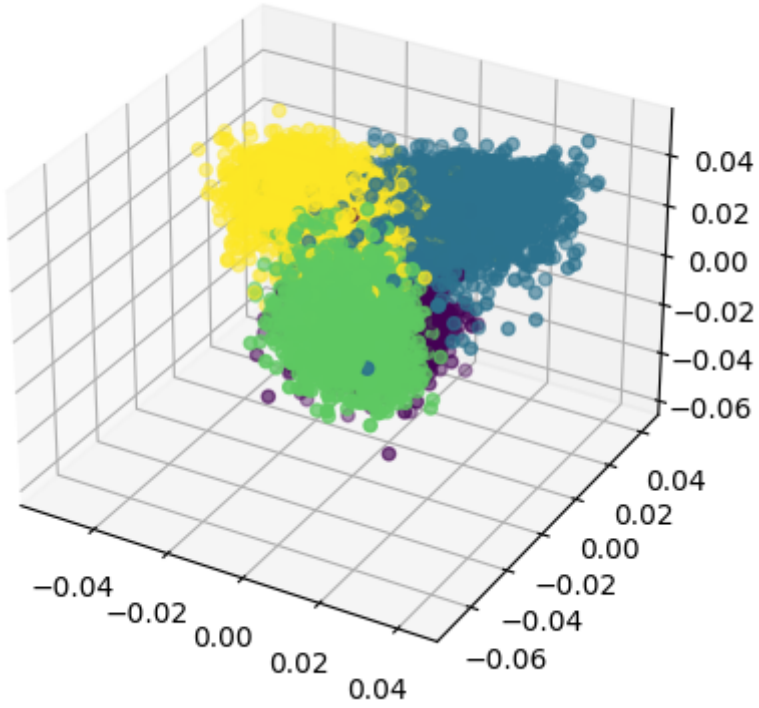

```
Distances from centroids:
Average distance between elements in cluster 0 from lda and centroid:  0.00
783816082573026
Average distance between elements in cluster 6 from lda and centroid:  0.01
4811716497001852
Average distance between elements in cluster 9 from lda and centroid:  0.01
3839982931709127

Accuracy in classification:
Total test for accuracy:  10344
Toatal right guess for algorithm  lda =  10218
Accuracy is:  98.78190255220419 %
```

In [ ]:
```
# LDA (example on test set)

chosen_X, chosen_Y, X_conc, Y_conc = concatenation(Xtest, Ytest, chosen_inde
Ztest_lda = projection_lda @ X_conc
clusters = get_clusters(Ztest_lda, Y_conc, chosen_indexes)
plot(Ztest_lda, Y_conc, chosen_indexes, k,limit_axes=True)
distances(Ztest_lda, Y_conc, chosen_indexes, "lda")
accuracy(X_conc, Y_conc, projection_lda, chosen_indexes, clusters, "lda")
```




```
Distances from centroids:
Average distance between elements in cluster 0 from lda and centroid:  0.01
4940507540051355
Average distance between elements in cluster 6 from lda and centroid:  0.01
4641162374883412
Average distance between elements in cluster 9 from lda and centroid:  0.00
6400803583000986

Accuracy in classification:
Total test for accuracy:  2113
Toatal right guess for algorithm  lda =  2076
Accuracy is:  98.24893516327496 %
```

In [ ]:
```
# LDA (example with k=3 and different/more digits)

chosen_indexes = [0,3,6,8]
k = 3
Z_lda, projection_lda, Y_conc = LDA(Xtrain, Ytrain, chosen_indexes, k)
plot(Z_lda, Y_conc, chosen_indexes, k)
clusters = get_clusters(Z_lda, Y_conc, chosen_indexes)
distances(Z_lda, Y_conc, chosen_indexes, "lda")
chosen_X, chosen_Y, X_conc, Y_conc = concatenation(Xtrain, Ytrain, chosen_i
print()
accuracy(X_conc, Y_conc, projection_lda, chosen_indexes, clusters, "lda")
```




```
Distances from centroids:
Average distance between elements in cluster 0 from lda and centroid:  0.01
4078844470334325
Average distance between elements in cluster 3 from lda and centroid:  0.00
7858464352903355
Average distance between elements in cluster 6 from lda and centroid:  0.01
3445907268672343
Average distance between elements in cluster 8 from lda and centroid:  0.00
9859714910320151


Accuracy in classification:
Total test for accuracy:  13828
Toatal right guess for algorithm  lda =  13310
Accuracy is:  96.25397743708419 %
```

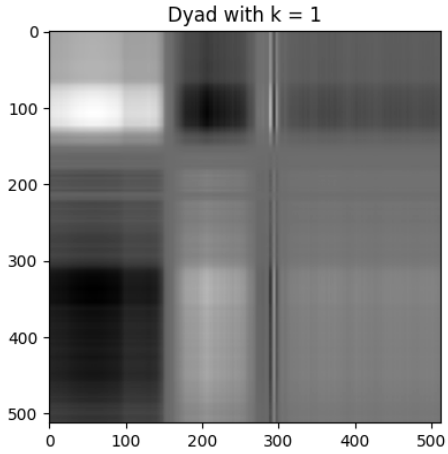
VISUALIZING DYAD

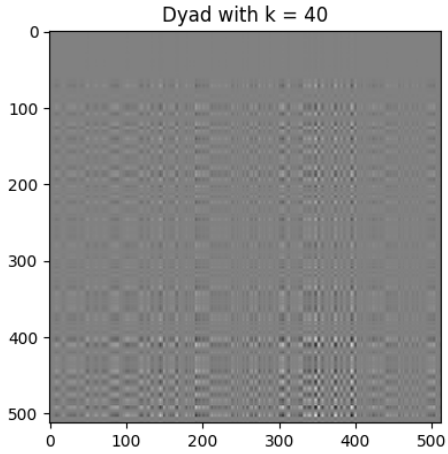
In [ ]:
```
# 1
X = skimage.data.camera()
print("Image size: ",X.shape)

U, s, Vh = alg.svd(X)
```

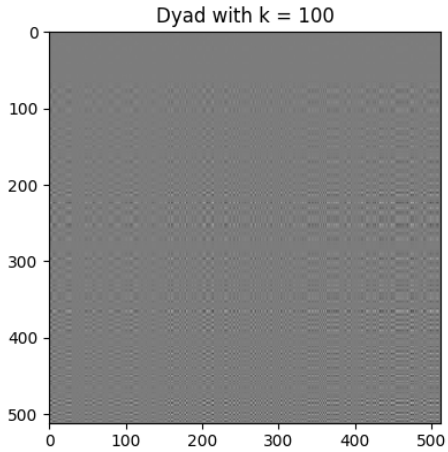
```
Image size:  (512, 512)
```

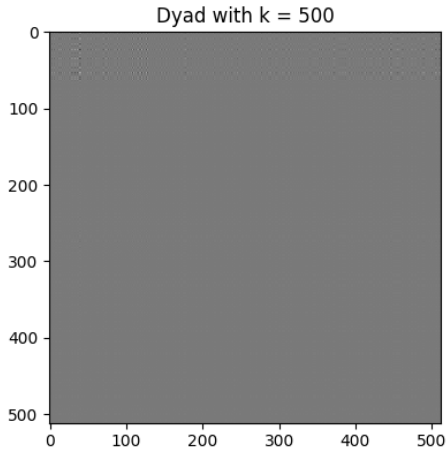
In [ ]:
```
# 2
num = [1, 40, 100, 500]
dyads = []

for i in num:
  dyads.append(np.dot(U[:,i].reshape([U.shape[0],1]), Vh[i,:].reshape([1,Vh

plt.figure(figsize=(15, 15))

fig1 = plt.subplot(3, 2, 1)
```

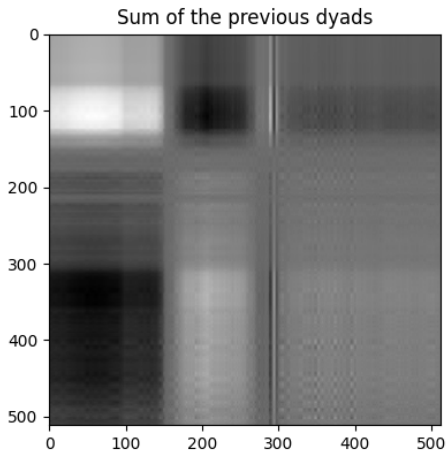
```
fig1.imshow(dyads[0], cmap='gray')
plt.title('Dyad with k = ' + str(num[0]))

fig2 = plt.subplot(3, 2, 2)
fig2.imshow(dyads[1], cmap='gray')
plt.title('Dyad with k = ' + str(num[1]))

fig3 = plt.subplot(3, 2, 3)
fig3.imshow(dyads[2], cmap='gray')
plt.title('Dyad with k = ' + str(num[2]))

fig4 = plt.subplot(3, 2, 4)
fig4.imshow(dyads[3], cmap='gray')
plt.title('Dyad with k = ' + str(num[3]))

summed_dyad = np.zeros(dyads[0].shape)
for dyad in dyads:
  summed_dyad = summed_dyad + dyad

fig5 = plt.subplot(3, 2, 5)
fig5.imshow(summed_dyad, cmap='gray')
plt.title('Sum of the previous dyads')
plt.show()
```





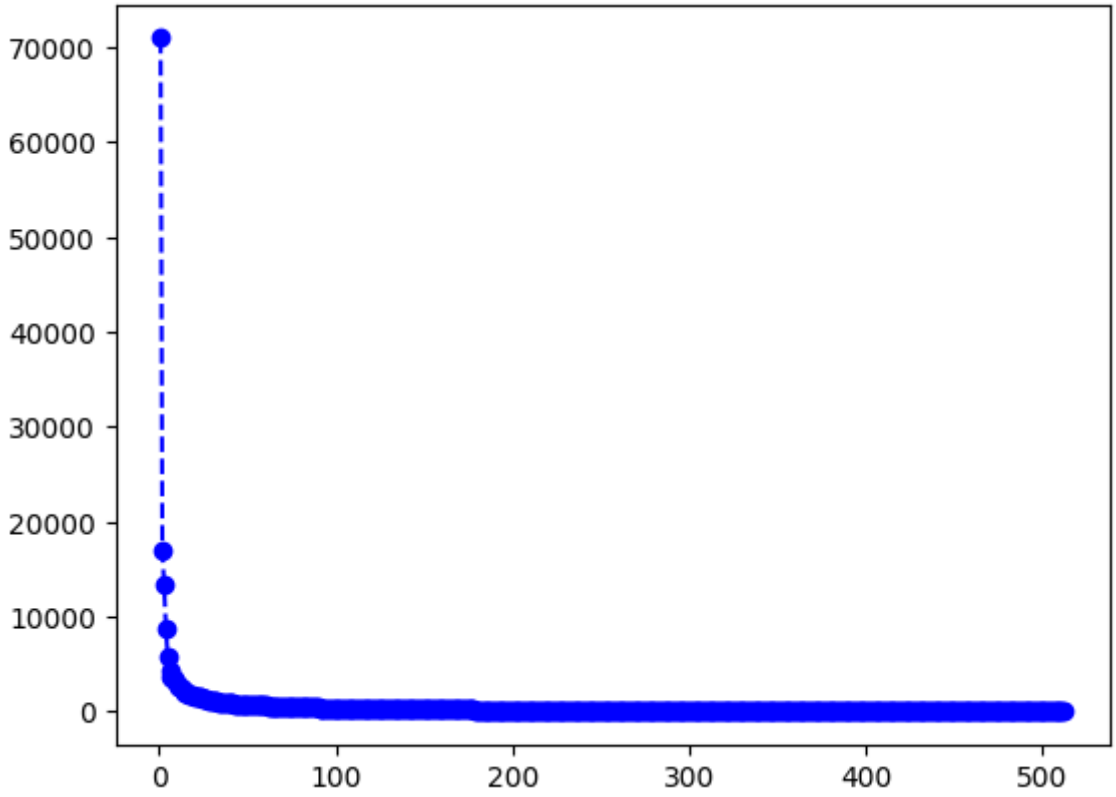






In [ ]:
```
# 3
x_plot_s = np.linspace(1,len(s),len(s))
plt.plot(x_plot_s, s, '--bo')
```

Out[ ]:
```
[<matplotlib.lines.Line2D at 0x7f3ab76e8970>]
```



In [ ]:
```
# 4,5,6

k = 100
toVisualize = [1, round(k/4), round(k/2), round(3 * k/4), k]
saved = []
errors = []
c_factors = []

X_k = np.zeros(X.shape)

for i in range(k+1):
  alph = np.dot(U[:,i].reshape([U.shape[0],1]), Vh[i,:].reshape([1,Vh.shape
  X_k = X_k + alph
  if(i in toVisualize):
    errors.append(np.linalg.norm(X-X_k))
    c_factors.append(((1/i) * min(U.shape[0], Vh.shape[0])) -1)
    saved.append(X_k)

plt.figure(figsize=(15, 15))

fig1 = plt.subplot(4, 2, 1)
fig1.imshow(saved[0], cmap='gray')
plt.title('Reconstructed image with k =' + str(toVisualize[0]))

fig2 = plt.subplot(4, 2, 2)
fig2.imshow(saved[1], cmap='gray')
plt.title('Reconstructed image with k =' + str(toVisualize[1]))

fig3 = plt.subplot(4, 2, 3)
fig3.imshow(saved[2], cmap='gray')
plt.title('Reconstructed image with k =' + str(toVisualize[2]))

fig4 = plt.subplot(4, 2, 4)
fig4.imshow(saved[3], cmap='gray')
plt.title('Reconstructed image with k =' + str(toVisualize[3]))
```

```
fig5 = plt.subplot(4, 2, 5)
fig5.imshow(saved[4], cmap='gray')
plt.title('Reconstructed image with k =' + str(toVisualize[4]))

fig6 = plt.subplot(4, 2, 6)
fig6.imshow(X, cmap='gray')
plt.title("True image")

fig7 = plt.subplot(4, 2, 7)
fig7.plot(toVisualize, errors, '--bo')
plt.grid()
plt.title("Approximation error")

fig8 = plt.subplot(4, 2, 8)
fig8.plot(toVisualize, c_factors, '--bo')
plt.grid()
plt.title("Compression factor")

plt.show()
```