

CS224 - Winter 2023

PROGRAMMING ASSIGNMENT 1 - Principal Component Analysis (PCA)

Due: February 13, 2023 @ 11:59pm PDT

Submission Method: Export the Jupyter notebook as **PDF** and submit the PDF file on **Gradescope**. Please **LINK** each question from the assignment in Gradescope to the location of its answer in your PDF. (For more details, see the Assignment Guidelines.)

Maximum points: 10

Enter your information below:

(full) Name: Yash Aggarwal

Student ID Number: 862333037

By submitting this notebook, I assert that the work below is my own work, completed for this course. Except where explicitly cited, none of the portions of this notebook are duplicated from anyone else's work or my own previous work.

Academic Integrity

Each assignment should be done individually. You may discuss general approaches with other students in the class, and ask questions to the TA, but you must only submit work that is yours. If you receive help by any external sources (other than the TA and the instructor), you must properly credit those sources. The UCR Academic Integrity policies are available at <http://conduct.ucr.edu/policies/academicintegrity.html>.

Overview

In this assignment, We will implement PCA, apply it to the **MNIST** dataset, and observe how the reconstruction changes as we change the number of principal components used.

For this assignment we will use the functionality of **Numpy**, and **Matplotlib**.

- Before you start, make sure you have installed all those packages in your local Jupyter instance.
- If you are asked to implement a particular functionality, you should **not** use an existing implementation from the libraries above (or some other library that you may find). When in doubt, **please just ASK**.
- It's okay to use functions in `numpy.linalg` to calculate matrix decomposition (e.g., `la.eig()`, `la.svd()`), but using built-in functions like `sklearn.decomposition.PCA()` will **not** get you any points.

Please read **all** cells carefully and answer **all** parts (both text and missing code). You will need to complete all the code marked `TODO` and answer descriptive/derivation questions.

```
In [1]: %matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
matplotlib.interactive(False)
from sklearn.datasets import load_digits
# DO NOT REMOVE THE CODE ABOVE
```

Question 1 [10 points]

Preliminaries

The **MNIST** database (Modified National Institute of Standards and Technology database) is a large database of handwritten digits that is commonly used for training various image processing systems.

First, Let's import the images and vectorize each image in the dataset.

```
In [2]: mnist = load_digits()
imgs = mnist.data
```

Compute the mean of the images and subtract out the mean.

```
In [3]: mean = np.mean(imgs, axis=0)
imgs -= mean
```

Let us call each zero mean vectorized image to be X . For N images, you have X_1, X_2, \dots, X_N . Compute $[X^\top X] = R$.

(a) [2 points] Perform a principal components analysis of $R = V\Lambda V^\top$.

```
In [4]: #TODO
covar_matrix = np.matmul(imgs.T, imgs)
values, vectors = np.linalg.eig(covar_matrix)
vectors = vectors.T
```

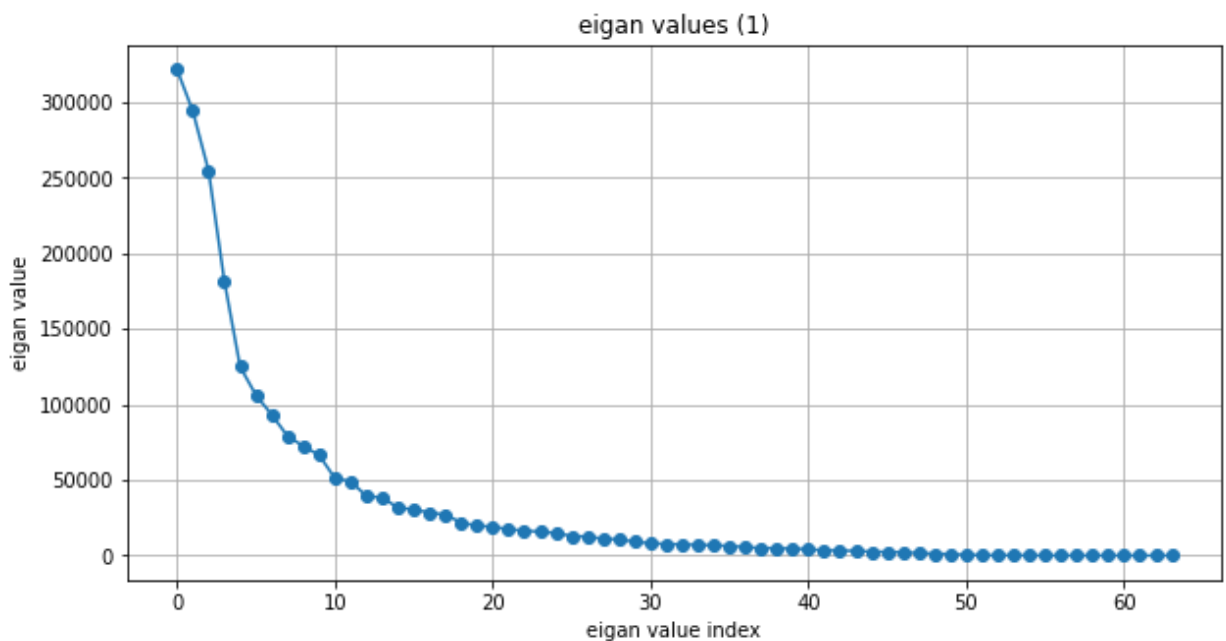
(b) [2 points] Analyze the eigenvalues in Λ and decide which eigenvalues to retain and which can be set to zero.

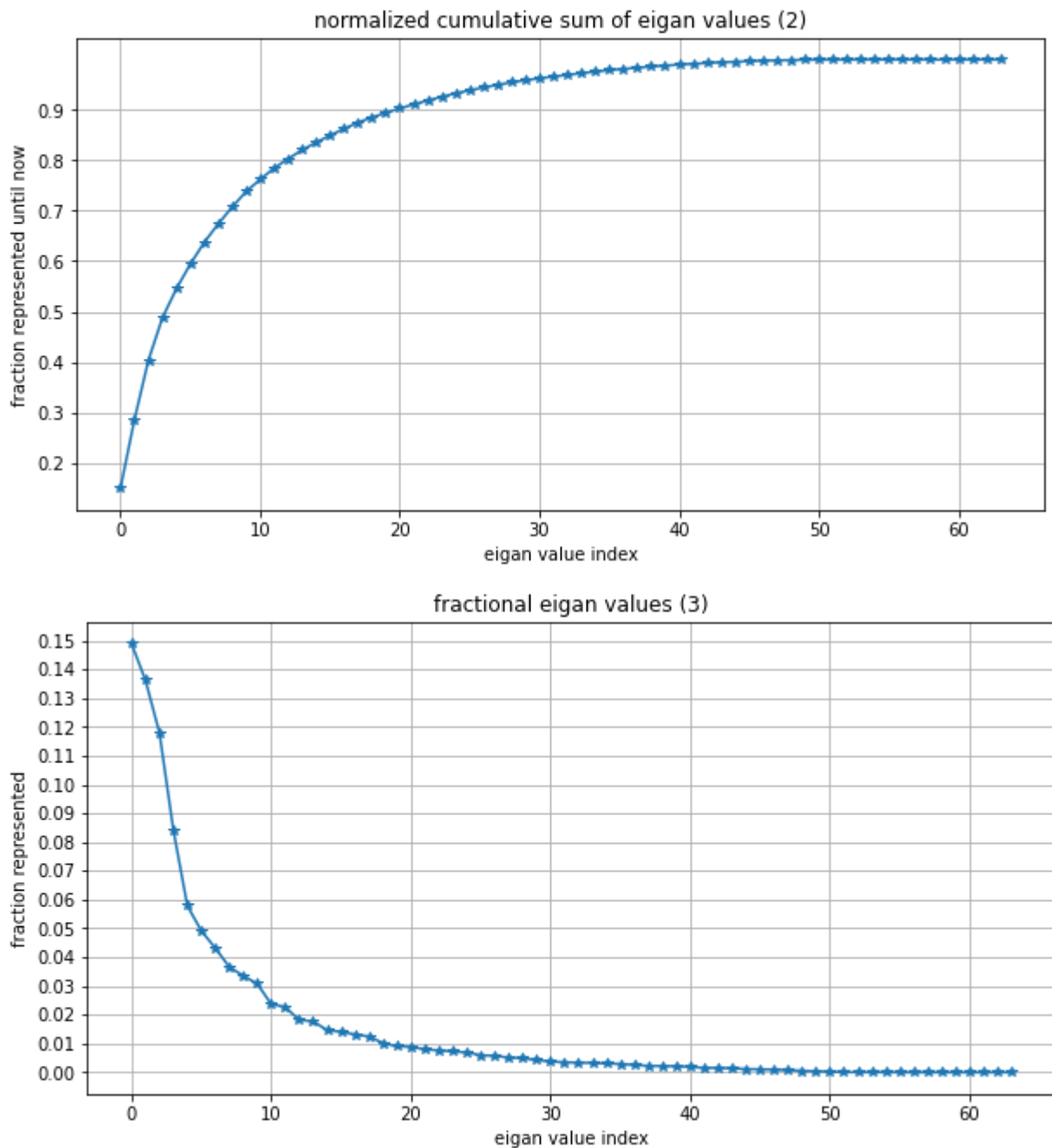
- You may want to plot the eigenvalues, the fraction of variance explained, AIC, or BIC, and set a threshold.

```
In [5]: # TODO
plt.figure(1,figsize=(10,5))
plt.grid()
plt.title('eigan values (1)')
plt.xlabel('eigan value index')
plt.ylabel('eigan value')
plt.plot(values, marker = 'o')
plt.show(block=True)

plt.figure(2,figsize=(10,5))
plt.grid()
plt.title('normalized cumulative sum of eigan values (2)')
plt.xlabel('eigan value index')
plt.ylabel('fraction represented until now')
plt.yticks(np.arange(0,1,0.1))
plt.plot(np.cumsum(values)/np.sum(values), marker = '*')
plt.show(block=True)

plt.figure(3,figsize=(10,5))
plt.grid()
plt.title('fractional eigan values (3)')
plt.xlabel('eigan value index')
plt.ylabel('fraction represented')
plt.yticks(np.arange(0,0.2,0.01))
plt.plot(values/np.sum(values), marker = '*')
plt.show(block=True)
```





Observation

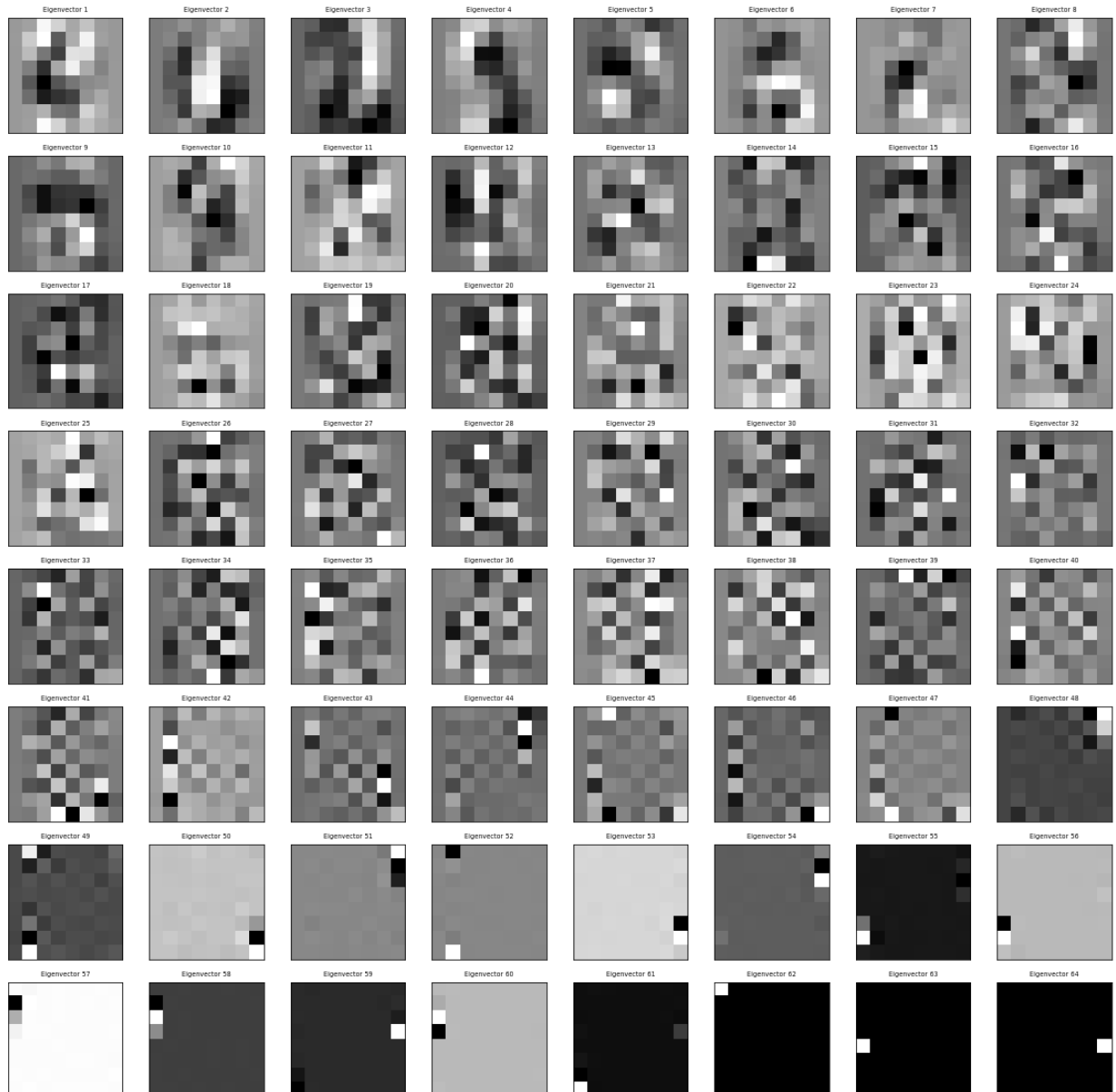
From graph (2) we can see that first 40 eigen values represent 90% of data or from graph (3) we can see that after first 40 eigen values, the data representation is less than 0.5% per value, so we will use first 40 values to approximate the data this means using ~63% for original data

```
In [6]: # Plot eigenVectors
n_row = 8
n_col = 8
plt.figure(figsize=(20,20))
plt.title('Eigen V')
for i in list(range(n_row * n_col)):
    plt.subplot(n_row, n_col, i + 1)
    plt.imshow(vectors[i].reshape(8,8), cmap='gray')
    title_text = 'Eigenvector ' + str(i + 1)
```

```
plt.title(title_text, size=6.5)
plt.xticks(())
plt.yticks(())
plt.show()
```

C:\Users\yg375\AppData\Local\Temp\ipykernel_20064\2416994209.py:7: MatplotlibDeprecationWarning: Auto-removal of overlapping axes is deprecated since 3.6 and will be removed two minor releases later; explicitly call ax.remove() as needed.

```
plt.subplot(n_row, n_col, i + 1)
```



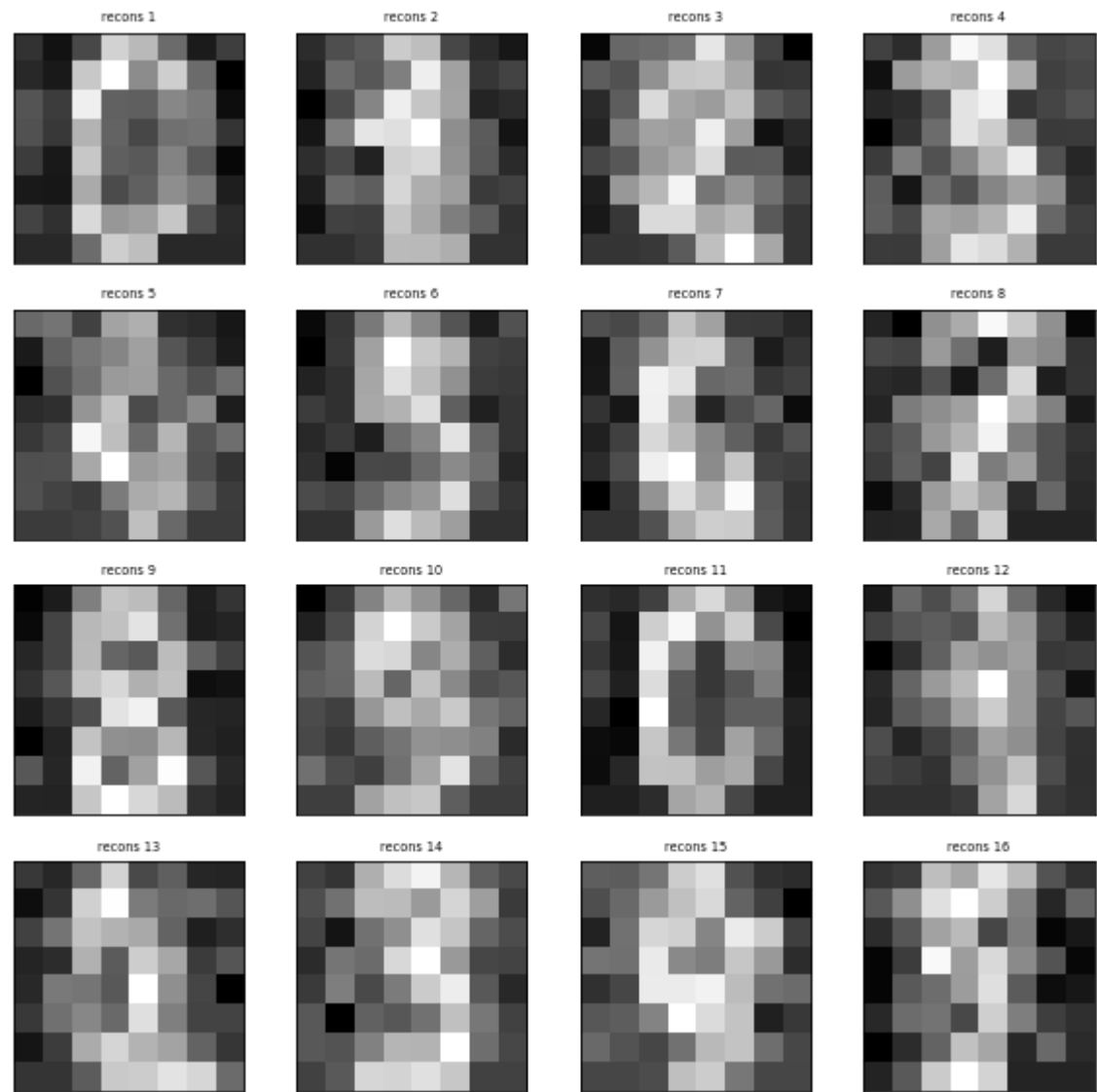
(c) [2 points] Reconstruct an approximation of each X after removing some of the small eigenvalues. (Display only a couple of the reconstructed **images**, and you will need to add the mean back.)

```
In [7]: # TODO
# taking only first 40 vectors
topVectors = vectors[:, :40]
new_coordinates = imgs @ topVectors
recons = new_coordinates @ topVectors.T
recons_with_mean = recons + mean
```

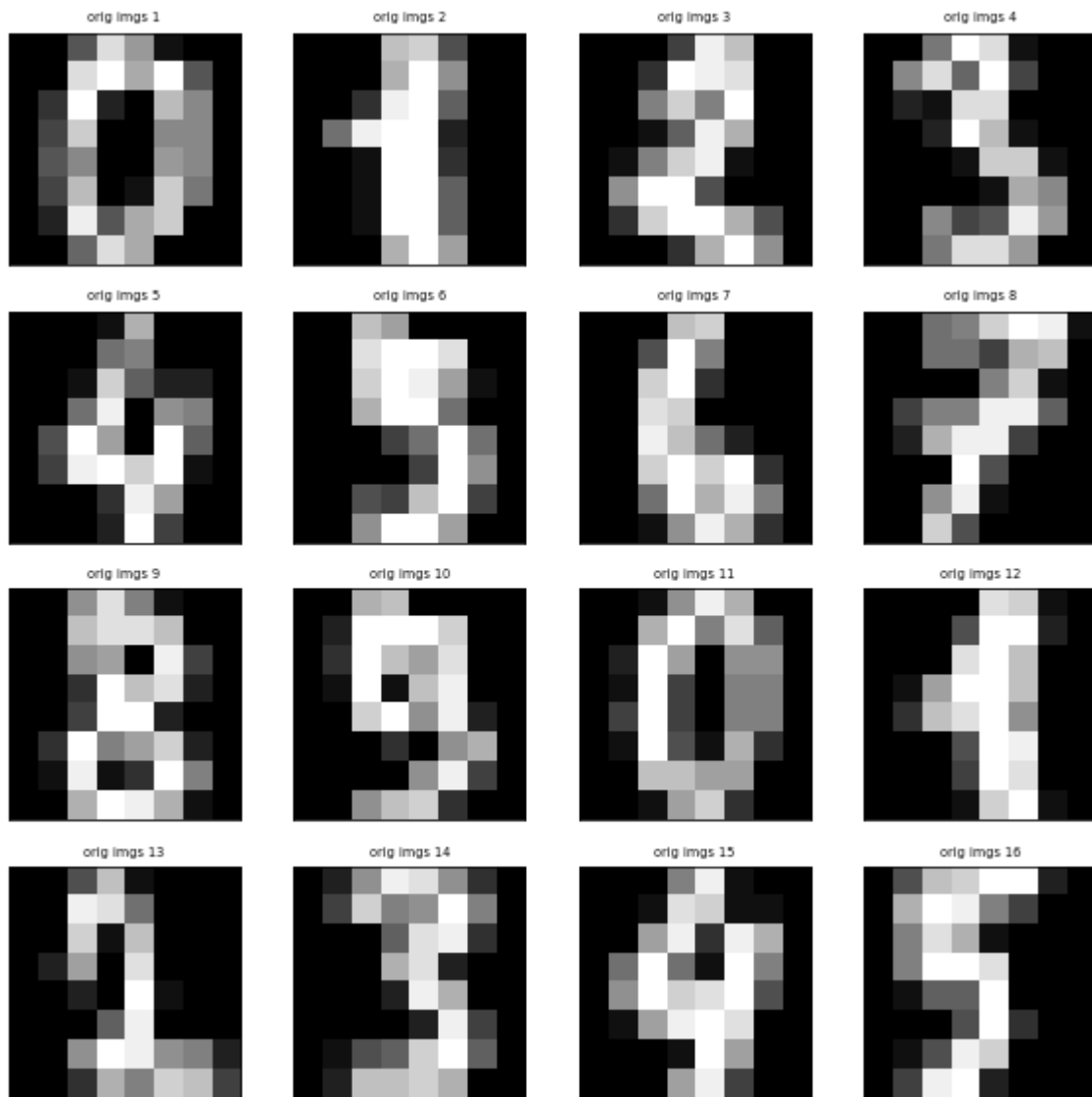
```
print ('reconstructed')
n_row = 4
n_col = 4
plt.figure(figsize=(10,10))
for i in list(range(n_row * n_col)):
    plt.subplot(n_row, n_col, i + 1)
    plt.imshow(recons_with_mean[i].reshape(8,8), cmap='gray')
    title_text = 'recons ' + str(i + 1)
    plt.title(title_text, size=6.5)
    plt.xticks(())
    plt.yticks(())
plt.show(block=True)

print ('\noriginal')
n_row = 4
n_col = 4
origimgs_with_mean = imgs+mean
plt.figure(figsize=(10,10))
for i in list(range(n_row * n_col)):
    plt.subplot(n_row, n_col, i + 1)
    plt.imshow(origimgs_with_mean[i].reshape(8,8), cmap='gray')
    title_text = 'orig imgs ' + str(i + 1)
    plt.title(title_text, size=6.5)
    plt.xticks(())
    plt.yticks(())
plt.show(block=True)
```

reconstructed



original



(d) [2 points] Compute the error between the reconstructed X and original image. (The mean of the original data should **not** be included in the error.)

```
In [8]: # TODO
error = np.sum((recons - imgs)**2)/len(imgs)
print (error)
```

510.2892496919737

(e) [2 points] Analyze by choosing different numbers of eigenvalues to be zeroed out. Provide a short summary of your conclusions based on this analysis.

```
In [9]: # Plot of any image for different eigen vectors

img_num = 9
n_row = 8
n_col = 8
plt.figure(figsize=(10,10))
for i in range(64):
    topVectors = vectors[:,i]
    new_coordinates = imgs @ topVectors
    recons = new_coordinates @ topVectors.T
```



```

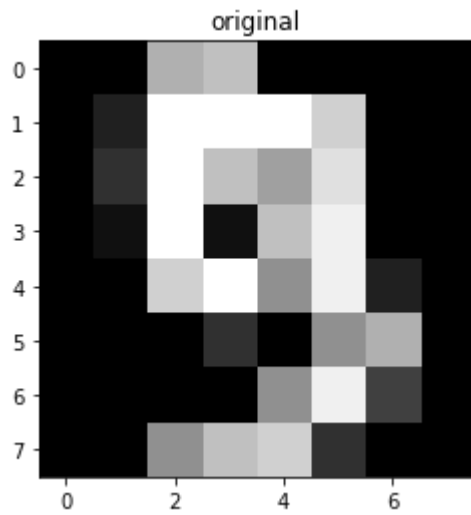
recons_with_mean = recons + mean

plt.subplot(n_row, n_col, i + 1)
plt.imshow(recons_with_mean[img_num].reshape(8,8), cmap='gray')
title_text = 'top ' + str(i + 1) + ' eigen vectors'
plt.title(title_text, size=6.5)
plt.xticks(())
plt.yticks(())

plt.figure(5)
plt.title('original')
plt.imshow(origimgs_with_mean[img_num].reshape(8,8), cmap='gray')
plt.show(block=True)

```



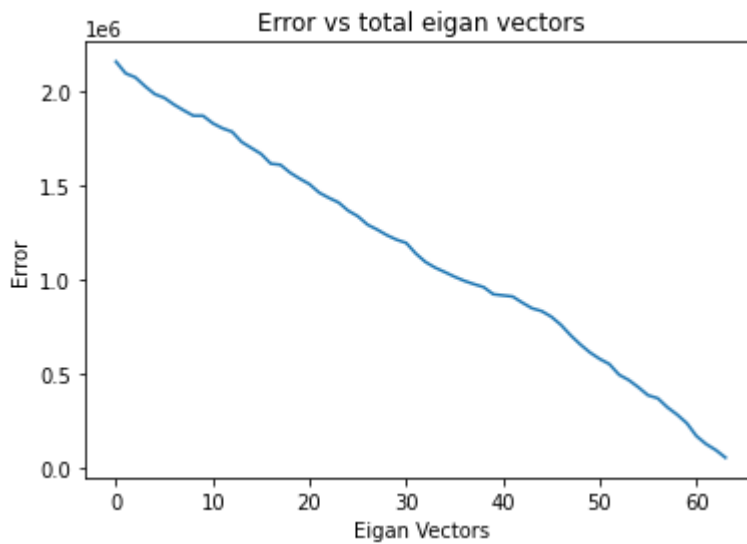


```
In [10]: # TODO
errorVals = []

for i in range(64):
    topVectors = vectors[:,i]
    new_coordinates = imgs @ topVectors
    recons = new_coordinates @ topVectors.T
    recons_with_mean = recons + mean

    error = np.sum((recons - imgs)**2)
    errorVals.append(error)

plt.title('Error vs total eigan vectors')
plt.xlabel('Eigan Vectors')
plt.ylabel('Error')
plt.plot(errorVals)
plt.show(block=True)
```



Conclusion

From the Error Graph and the plot with multiple eigen vectors, we can see that when we increase the number of eigen vectors, we get an image closer to the original and if we pick fewer eigen vectors, we get a worse approximation but that does mean that we will use less space to store the data.

In []: