Project 1

April 20, 2020

1 Project 1. Dimension Reduction Using SVD and PCA

1.0.1 Student ID: 915942842

1.0.2 Student Name: Xuecheng Zhang

[33]:
```
# All Import Statements Defined Here
# Note: Do not change anything

import numpy as np
import math
import matplotlib.pyplot as plt
from matplotlib.image import imread

from numpy.linalg import svd
from sklearn.decomposition import PCA
import os
from sklearn.datasets import load_digits

%matplotlib inline
```

1.1 Part 1. SVD Image Compression

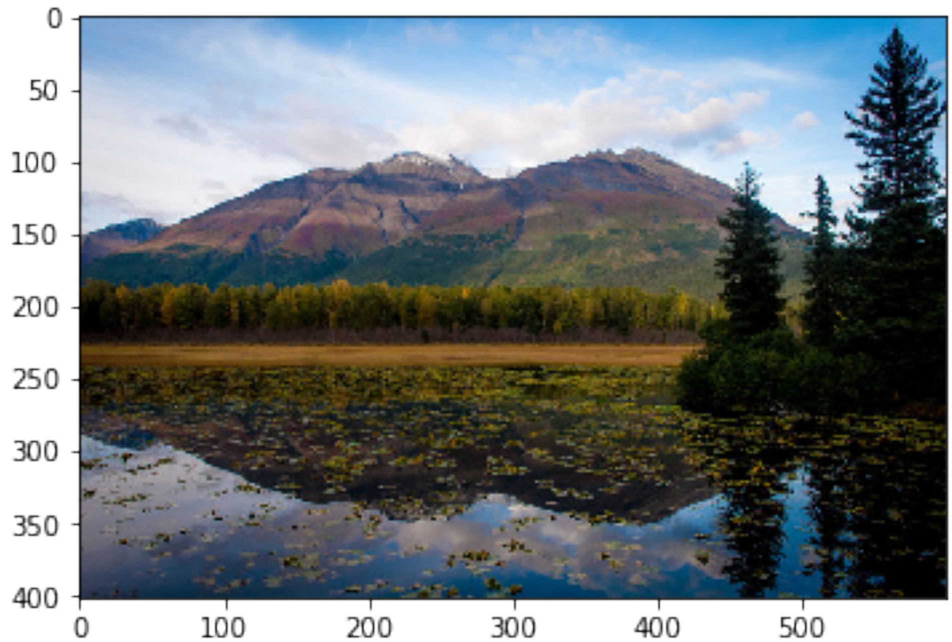
1.1.1 Load Data (do not change the code !)

[34]:
```
## image1 should be data.png, which is contained in the project zip file
## image2 should be your own colored face photo in png format named as photo.
 ↪png with a size no larger than 600x600

image1 = imread(os.getcwd()+'/data.png')
image2 =  imread(os.getcwd()+'/photo.png')

image1 = image1[:,:,:3]
image2 = image2[:,:,:3]

## store the images in image_dict with name lake for image1 and name photo for␣
 ↪image2
```

```
image_dict = {'lake':image1,'photo':image2}
```

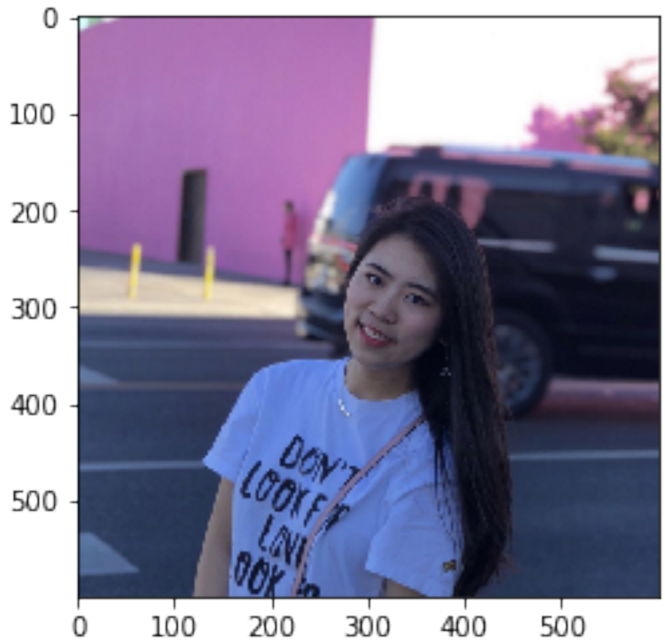
[35]:
```
plt.imshow(image_dict['lake'])
```

[35]: <matplotlib.image.AxesImage at 0x1a229f7590>



[36]:
```
plt.imshow(image_dict['photo'])
```

[36]: <matplotlib.image.AxesImage at 0x1a243917d0>




```
[37]: image_dict['photo'].shape
```

```
[37]: (600, 600, 3)
```

Add a bit of introduction to yourself here, like your name, major, year, background, interets etc.

My name is Xuecheng Zhang, I am a senior of this spring quarters, puring a bechelors's degree in Statistical Data Science major. I love telling stories with data especially with data visualization and using data to help bring that data to life. I plan to become a data analyst or product manager after graduation

1.1.2 Write a function to get the compressed matrix

```
[38]: def compress_svd(image,k):
          """
          use svd decomposition to perform image compression
          use the svd function from numpy.linalg to perfrom the svd decomposition
          use the first k singular values to reconstruct the compressed matrix
          """

          ## write your code here

          U,s,V=svd(image,full_matrices=False)
          compressed_image=np.dot(U[:,:k],np.dot(np.diag(s[:k]),V[:k,:]))
```

```
    ##end of your code

    return compressed_image
```

1.1.3 Write the following functions to compress the images differently

[39]:
```
def compress_show_images_reshape(image_name,k,show_image=True):
    """
    image_name (string): image name in the image_dict
    k (int): number of singular value for image compression
    show_image (boolean): whether to plot the compressed functions

    Concatenate the first three layers of the image tensor into one wide matrix
    Use compress_svd function to perform svd compression
    Reshape the wide compressed matrix into an image tensor of three layers
    if show_image is true, plot the compressed image
    put the number of singular values and reconstruction error in the title

    return reconst_error (float), which is the mean squared error of the␣
 ↪compressed image

    """

    ## your code starts here

    image=image_dict[image_name]
    size=image.shape
    img=image.copy()

    X=np.concatenate((image[:,:,0],image[:,:,1],image[:,:,2]),axis=1)#column␣
 ↪conbine RGB
    image_compressed=compress_svd(X,k)#image compressed
    A=np.hsplit(image_compressed,3)#reshape back to three layers
    for i in (0,1,2):
        img[:,:,i]=A[i]

    reconst_error=np.square(np.subtract(image,img)).mean()

    if show_image == True:
        plt.imshow(img)
        plt.title('Singular value is '+str(k)+' and the reconst error is␣
 ↪'+str(reconst_error))


    ## end of your code
```

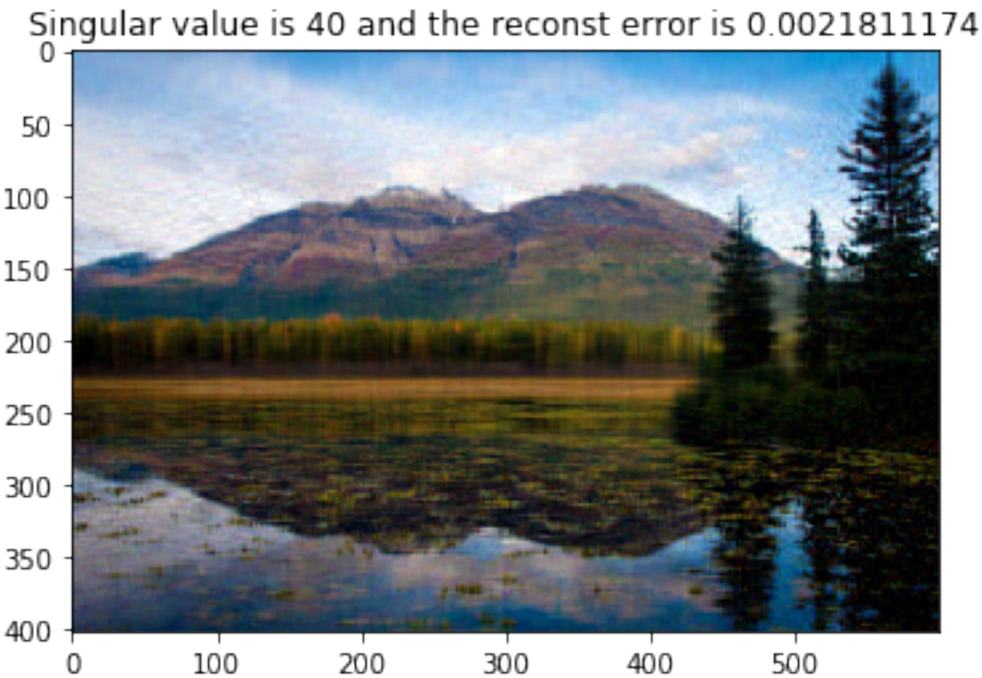
```
    return reconst_error
```

[40]:
```
def compress_show_images_separate(image_name,k,show_image=True):
    """
    image_name (string): image name in the image_dict
    k (int): number of singular value for image compression
    show_image (boolean): whether to plot the compressed functions

    Use compress_svd function to perform svd compression for each of the three␣
 ↪layers of the image tensor
    if show_image is true, plot the compressed image
    put the number of singular values and reconstruction error in the title

    return reconst_error (float), which is the mean squared error of the␣
 ↪compressed image

    """

    ## your code starts here
    image=image_dict[image_name]
    image1=image.copy()

    image1[:,:,1]=compress_svd(image[:,:,1],k)
    image1[:,:,0]=compress_svd(image[:,:,0],k)
    image1[:,:,2]=compress_svd(image[:,:,2],k)

    reconst_error=np.square(np.subtract(image,image1)).mean()

    if show_image == True:
        plt.imshow(image1)
        plt.title('Singular value is '+str(k)+' and the reconst error is␣
 ↪'+str(reconst_error))

    ## end of your code

    return reconst_error
```

[41]:
```
compress_show_images_separate('lake',40)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
```

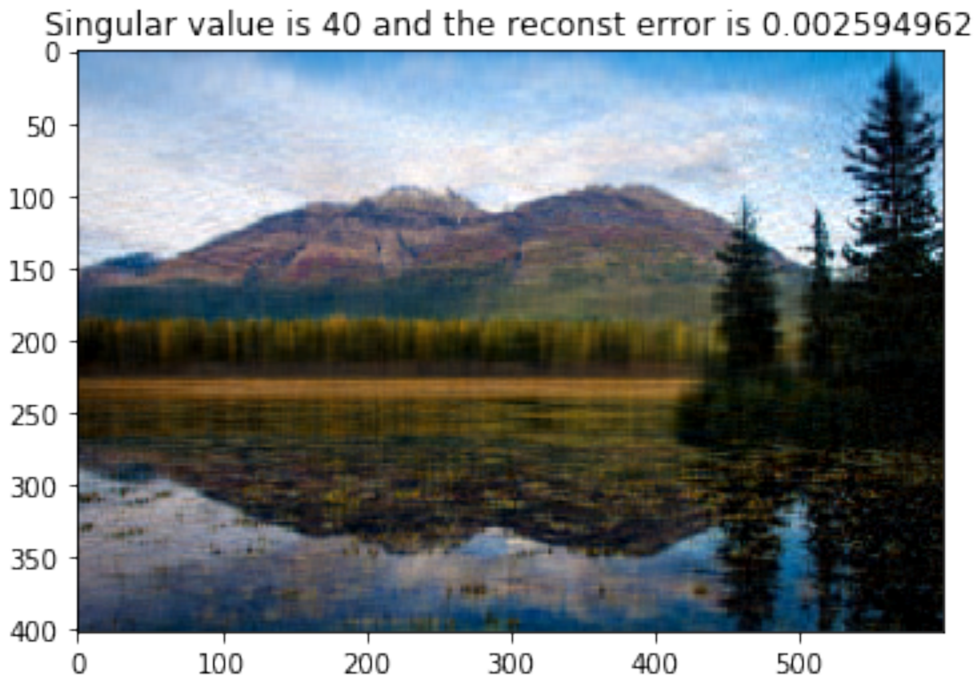
[41]: 0.0021811174



[42]:
```
compress_show_images_reshape('lake',40)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
```

[42]: 0.002594962



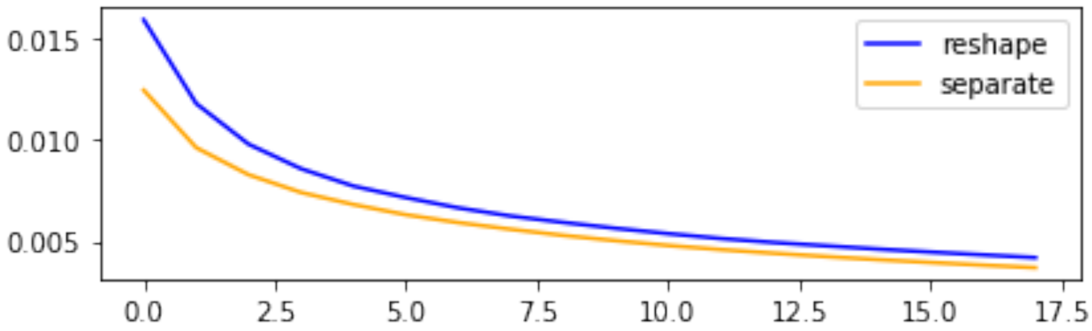
1.1.4 Write a function to plot the reconstruction errors and their differences

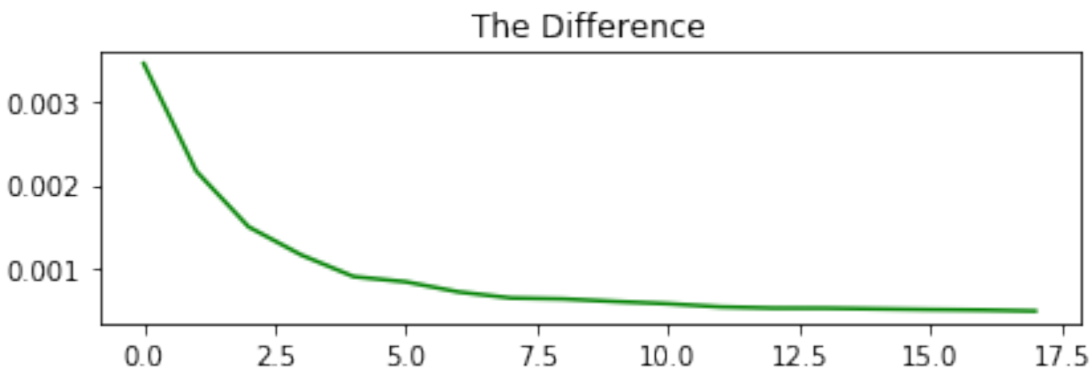
```
[43]: def plot_error(image_name,k_min,k_max):
          """
          image_name (string): image name in the image_dict
          k_min (int): minimum number of singular values used for image compression
          k_max (int): maximum number of singular values used for image compression

          plot the reconstruction errors using k between k_min and k_max
          use a blue line to indicate the reconstruction error using␣
      ↪compress_show_images_reshape function
          use a orange line to indicate the reconstruction error using␣
      ↪compress_show_images_reshape function
          in a separate subfigure, plot the difference of the reconstruction errors␣
      ↪for each k between k_min and k_max

          describe briefly what you see (is one error always smaller than the other?␣
      ↪is the difference monotonic?)
          """

          ## your code starts here
          reshape_error=[]
          separate_error=[]
          for k in range (k_min,k_max):
```

```
        re = compress_show_images_reshape(image_name,k,show_image=False)
        se = compress_show_images_separate(image_name,k,show_image=False)
        reshape_error.append(re)
        separate_error.append(se)


    plt.subplot(2,1,1)
    plt.plot(reshape_error, color='blue',label="reshape")
    plt.plot(separate_error, color='orange',label="separate")
    plt.legend(loc='upper right')




    plt.subplot(2,1,2)
    dif=[]
    zip_object=zip(reshape_error,separate_error)
    for reshape_error,separate_error in zip_object:
        dif.append(reshape_error-separate_error)
    plt.plot(dif, color='green',label="reshape")
    plt.title('The Difference')
    plt.tight_layout()



    ## end of your code
```

1.2 Run the following code. (do not change the code!)

[44]:
```
plot_error('lake',2,20)
```





In range k between 2 and 20, the separate error always smaller than reshape error. The the error is decrease but not monotonic.

[45]:
```
[compress_show_images_separate('lake',k,False) for k in [1,5,10,20,30,40,20000]]
```

[45]:
```
[0.018221032,
 0.0073899403,
 0.005279084,
 0.0035794626,
 0.0027349214,
 0.0021811174,
 2.2496302e-14]
```

[46]:
```
[compress_show_images_reshape('lake',k,False) for k in [1,5,10,20,30,40,20000]]
```

[46]:
```
[0.021424817,
 0.0085557,
 0.005917929,
 0.004069459,
 0.003191525,
 0.002594962,
 2.2788656e-14]
```

[47]:
```
from ipywidgets import interact
```

```
[48]: interact(compress_show_images_reshape,image_name=['lake','photo'], k=(10,70))
```

```
interactive(children=(Dropdown(description='image_name', options=('lake', 'photo'), value='lake
```

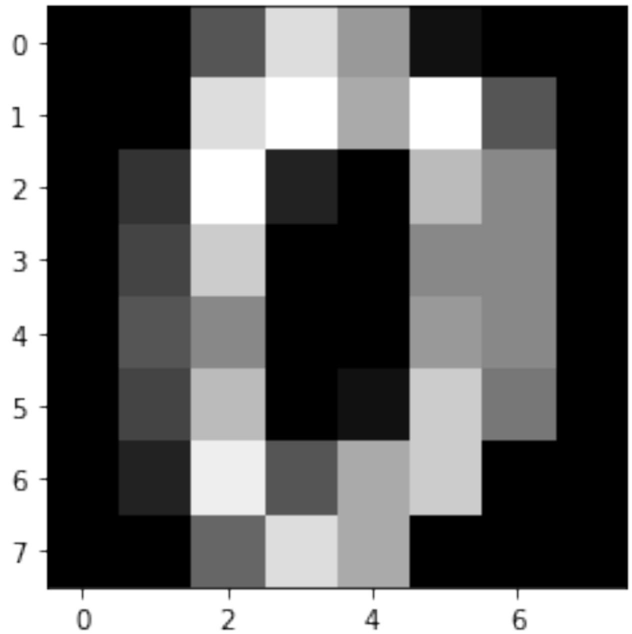
```
[48]: <function __main__.compress_show_images_reshape(image_name, k, show_image=True)>
```

1.3 Part 2. PCA of hand-written digits

1.3.1 Load data

```
[49]: digits = load_digits()
      plt.imshow(digits.images[0],cmap='gray')
```

```
[49]: <matplotlib.image.AxesImage at 0x1a2286c8d0>
```



1.3.2 Check the covariance matrix

```
[50]: mu=digits.data.mean(axis=0)
      X=digits.data
      X_bar=np.repeat(mu,X.shape[0]).reshape(len(mu),-1).T
      cov=np.dot( (X-X_bar).T, X-X_bar )

      ## What do you find? Not full rank!

      np.linalg.matrix_rank(cov)
```

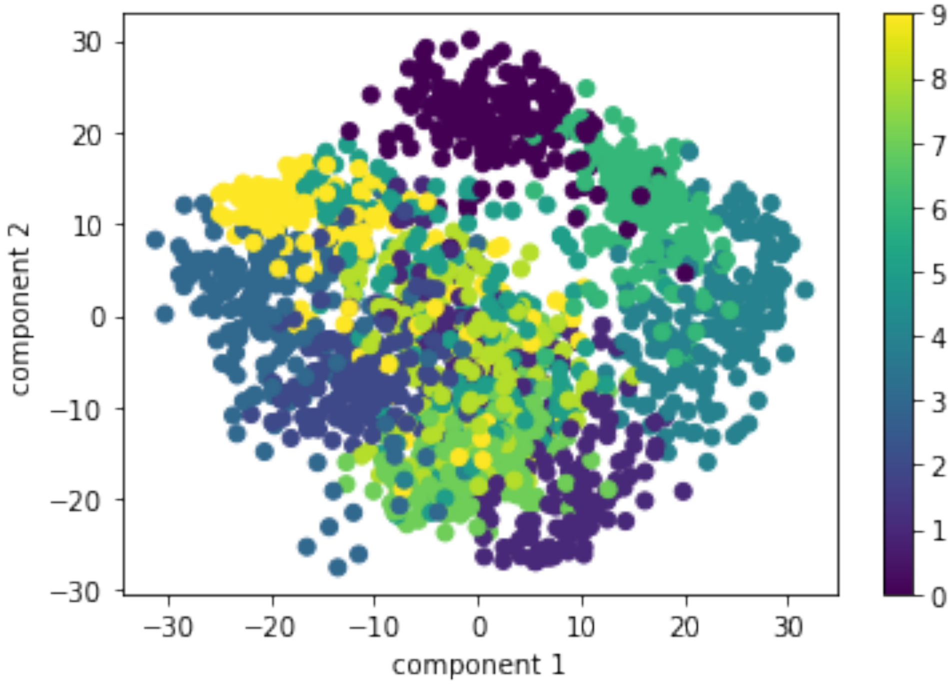
[50]: 61

1.3.3 Now use PCA function to use a 2-dim subspace to reconstruct the digits

[51]:
```
## X_new is a list of lists which contains the first two PC for each digit
## X_inv is the reconstructed digit image with the same shape as X
## set the random_state = 12 for the PCA function ! this is important for your␣
↪grade.

## your code starts here

pca=PCA(n_components=2,random_state = 12)
X_new=pca.fit_transform(X)
X_inv=pca.inverse_transform(X_new)

## end here
```

1.3.4 Plot each digits (elements in X_new) with different color labels

[52]:
```
## your code starts here

plt.scatter(X_new[:, 0], X_new[:, 1],c=digits.target)
plt.xlabel('component 1')
plt.ylabel('component 2')
plt.colorbar();

## end here
```



1.3.5 Write a function to do a side-by-side plot of the original digit and the reconstructed digit

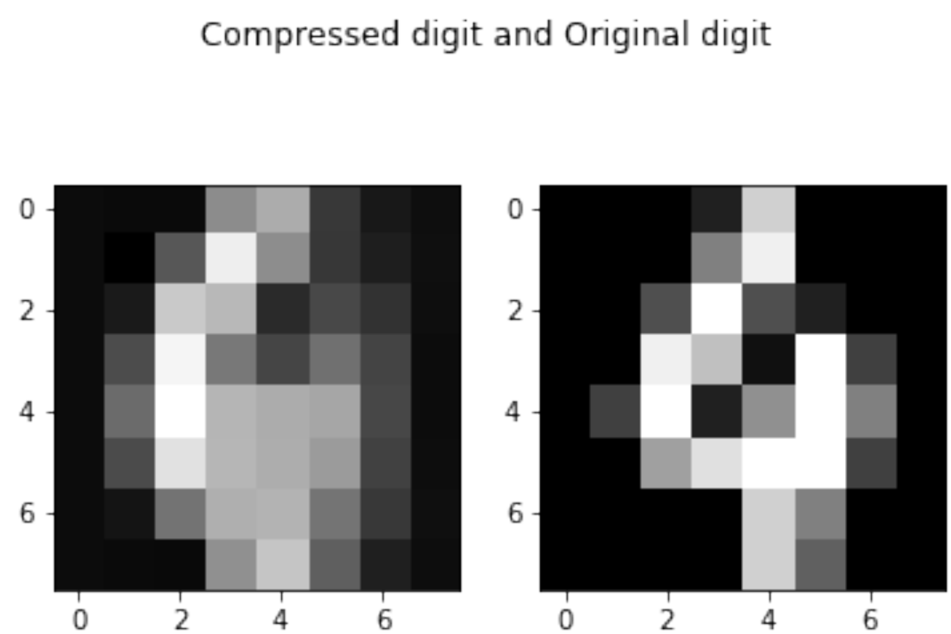
[53]:
```
def plot_digits(k):
    """
    the left part is the compressed digit, the right part is the original digit
    """
    ## your code starts here
    #ax1= plt.figure.subplots(1,2)
    #fig.suptitle('Horizontally stacked subplots')
    #ax1.plot(X_inv[k].reshape(8,8),camp='gray')
    #ax2.plot(X[k].reshape(8,8),camp='gray')
    #plt.imshow()
    fig, axes = plt.subplots(nrows=1,ncols=2)
    fig.suptitle('Compressed digit and Original digit')
    axes[0].imshow(X_inv[k].reshape(8,8),cmap='gray')
    axes[1].imshow(X[k].reshape(8,8),cmap='gray')




    ## end here
```

1.4 Run the following code. (do not change the code!)

[54]:
```
X_inv[0]
```

[54]:
```
array([ 8.07635321e-16,  1.10626914e-01,  4.44191193e+00,  1.18063215e+01,
        1.07493158e+01,  3.39833625e+00,  5.09920504e-02, -4.20291639e-02,
        2.49610241e-03,  1.69701669e+00,  1.19740914e+01,  1.16788043e+01,
        8.38141400e+00,  7.34547162e+00,  6.11223386e-01,  3.14124548e-02,
        1.94499279e-03,  3.47286485e+00,  1.40160400e+01,  5.68604617e+00,
        2.51564997e+00,  7.91668154e+00,  2.32198152e+00,  4.03119253e-02,
       -2.15810066e-04,  4.02956705e+00,  1.27804312e+01,  5.93137976e+00,
        4.63725756e+00,  8.66603267e+00,  3.63836316e+00,  2.81493730e-03,
        0.00000000e+00,  4.01350990e+00,  9.05748081e+00,  3.10220333e+00,
        4.12475708e+00,  1.22384928e+01,  5.60232331e+00,  0.00000000e+00,
        1.34385175e-02,  2.53046924e+00,  9.27133942e+00,  1.19509350e+00,
        9.92216788e-01,  1.33325823e+01,  8.13131651e+00,  5.34711306e-02,
        1.24618644e-02,  9.39578740e-01,  1.08402535e+01,  6.87893829e+00,
        7.21199881e+00,  1.50137994e+01,  6.97000803e+00,  2.16781211e-01,
       -1.66186424e-04,  8.26542693e-02,  4.35938527e+00,  1.26309359e+01,
        1.59542946e+01,  1.06358814e+01,  2.52788152e+00,  2.36464021e-01])
```

[55]:
```
plot_digits(100)
```



2 Submission Instructions

1. Click the Save button at the top of the Jupyter Notebook.

2. Please make sure to have entered your Student ID above.
3. Select Cell -> All Output -> Clear. This will clear all the outputs from all cells (but will keep the content of ll cells).
4. Select Cell -> Run All. This will run all the cells in order, and will take several minutes. You will not get any grade if you don't follow this step strictly.
5. Once you've rerun everything, select File -> Download as -> PDF via LaTeX
6. Look at the PDF file and make sure all your solutions are there, displayed correctly. The PDF is the only thing your graders will see!
7. Submit your PDF on Canvas.

Part B:

1a show that for a given m-dimensional vector a, the map

$f_a(x) = \langle a, x\rangle$ is linear

$f_a(x) = \langle a\ x\rangle = \vec{a}^T \cdot x = (a_1\ a_2\ \cdots\ a_m) \cdot \begin{pmatrix} x_1 \\ \vdots \\ x_m \end{pmatrix} = \boxed{a_1 \cdot x_1 + a_2 \cdot x_2 + \cdots a_m \cdot x_m}$

So the map $f_a(x)$ is linear.

Then function linear means that:

$f_a(x+y) = \langle a, x+y\rangle = a_1(x_1+y_1) + a_2(x_2+y_2) + \cdots a_m(x_m+y_m) = a_1 x_1 + \cdots + a_m x_m + a_1 y_1 + \cdots + a_m y_m = f_a(x) + f_a(y)$

$f_a(\lambda x) = \langle a, \lambda x\rangle = a_1 \lambda x_1 + a_2 \lambda x_2 + \cdots a_m \lambda x_m = \lambda(a_1 x_1 + a_2 x_2 + \cdots a_m x_m) = \lambda f_a(x)$ #

1b let A be a $(n \times m)$-matrix, show that the map $f_A(x) = A \cdot x$ is linear.

$f_A(x) = A \cdot x = \begin{pmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & & \vdots \\ a_{n1} & \cdots & a_{nm} \end{pmatrix} \cdot x = \begin{pmatrix} a_{1\cdot} \\ \vdots \\ a_{n\cdot} \end{pmatrix} \cdot x = (a_{\cdot 1}\ \cdots\ a_{\cdot m}) \cdot \begin{pmatrix} x_1 \\ \vdots \\ x_m \end{pmatrix}$

$= a_{\cdot 1} \cdot x_1 + a_{\cdot 2} \cdot x_2 + \cdots a_{\cdot m} \cdot x_m$

$\Rightarrow f_A(x)$ is a linear combination of the columns of A, the map is linear.

$f_A(x+y) = A(x+y) = a_{\cdot 1}(x_1+y_1) + \cdots a_{\cdot m}(x_m+y_m)$

$= a_{\cdot 1} x_1 + \cdots a_{\cdot m} x_m + a_{\cdot 1} y_1 + \cdots a_{\cdot m} \cdot y_m = Ax + Ay = f_A(x) + f_A(y)$

$f_A(\lambda x) = A(\lambda x) = a_{\cdot 1} x_1 \cdot \lambda + a_{\cdot 2} x_2 \cdot \lambda + \cdots a_{\cdot m} x_m \cdot \lambda = \lambda(a_{\cdot 1} x_1 + a_{\cdot 2} x_2 + \cdots a_{\cdot m} x_m)$

$= \lambda A x = \lambda \cdot f_A(x)$ #

$\boxed{2.}$ $A = \begin{pmatrix} \frac{3}{2} & -\frac{1}{2} \\ -\frac{1}{2} & \frac{3}{2} \end{pmatrix}$

a) for any $\vec{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$, we have

$$\vec{x}^T A \cdot \vec{x} = \frac{3}{2} x_1^2 + \frac{3}{2} x_2^2 - x_1 x_2 \geq 0$$

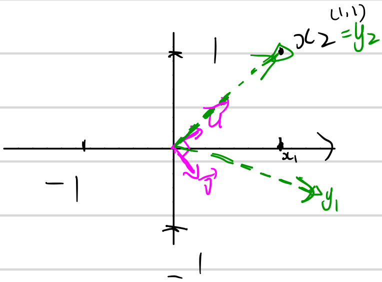
Therefore, we know A is positive semidefinite, moreover, if we let

$$\vec{x}^T \cdot A \cdot \vec{x} = \frac{3}{2} x_1^2 + \frac{3}{2} x_2^2 - x_1 x_2 = 0$$

then the only solution is $x_1 = x_2 = 0$, This implies that A is positive definite

b)

bi



$y_1 = A \cdot x_1 = \begin{pmatrix} \frac{3}{2} \\ -\frac{1}{2} \end{pmatrix}$, $y_2 = A \cdot x_2 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$

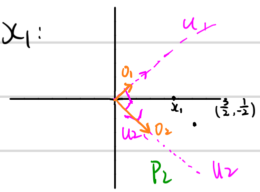
b ii) $\det(A - \lambda I) \Rightarrow (\frac{3}{2} - \lambda)^2 - \frac{1}{4} = 0 \Rightarrow \lambda_1 = 1$ and $\lambda_2 = 2$

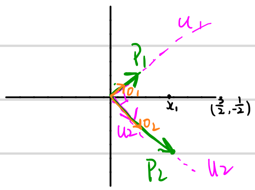
when $\lambda = 1$ $\Rightarrow \begin{pmatrix} \frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{2} & \frac{1}{2} \end{pmatrix} \cdot \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = 0 \Rightarrow \vec{u} = \begin{bmatrix} \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \end{bmatrix}$

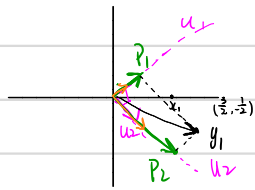
when $\lambda = 2$ $\Rightarrow \begin{pmatrix} -\frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{2} & -\frac{1}{2} \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = 0 \Rightarrow \vec{v} = \begin{bmatrix} \frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} \end{bmatrix}$

biii

For x_1:

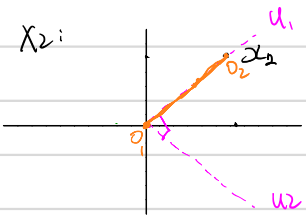


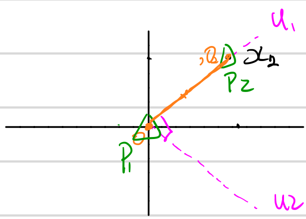


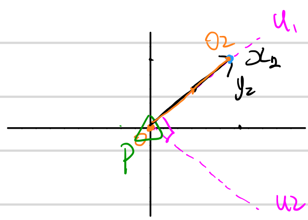


step I : projection $O = u_i^T x_i \cdot u_i$ step2 : Scaling : $P = \lambda_i (u_i^T x_i) u_i$ Step3 : adding

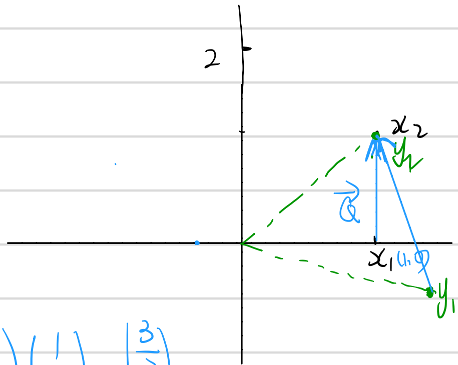
For X_2:







biv



$$T(x_1) = \begin{pmatrix} \frac{3}{2} & -\frac{1}{2} \\ -\frac{1}{2} & \frac{3}{2} \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{3}{2} \\ -\frac{1}{2} \end{pmatrix}$$

$$T(x_2) = \begin{pmatrix} \frac{3}{2} & -\frac{1}{2} \\ -\frac{1}{2} & \frac{3}{2} \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$