CMPSC 445: Applied Machine Learning in Data Science
Spring 2023

Project-2: Face Recognition Using PCA

Gabriel Nulman – CMPSC Senior
Professor V. Elangovan – Department Chair CMPSC

February 28th, 2023

Project 2

Gabriel Nulman

Professor V. Elangovan - CMPSC 445

In [ ]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
import os
%matplotlib inline
```

Step 1, 2 and 3 where I collect the images, resize them, and them convert to grayscale

In [ ]:
```
resized_faces = []

for img in os.listdir('images'):
    file = os.path.join('images', img)
    if os.path.isfile(file):
        img = cv2.imread(file)
        resize = cv2.resize(img, dsize=(50,50))
        gray = cv2.cvtColor(resize, cv2.COLOR_BGR2GRAY)
        np.transpose(gray)
        resized_faces.append(gray)
```

Step 4: Get the pixel values for the images

In [ ]:
```
resized_faces
```

Out[ ]:
```
[array([[ 84,  85,  85, ...,  84,  84,  85],
        [ 84,  85,  86, ...,  84,  85,  85],
        [ 86,  86,  85, ...,  85,  86,  86],
        ...,
        [101,  67,  79, ...,  13,  24,  14],
        [ 85,  69,  48, ...,  16,  12,  16],
        [ 63,  50,  39, ...,  15,  17,  43]], dtype=uint8),
 array([[134, 132, 133, ..., 143, 155, 129],
        [122, 124, 124, ..., 137, 152, 151],
        [143, 135, 139, ..., 149, 139, 142],
        ...,
        [ 33, 110, 106, ...,  85,  56, 113],
        [ 86,  55,  57, ...,  79,  88,  62],
        [115,  41,  46, ...,  68, 110,  47]], dtype=uint8),
 array([[ 11,  13,   8, ...,  15,  17,   9],
        [ 12,  19,  12, ...,  17,  16,  15],
        [ 14,  11,  14, ...,  15,  18,  14],
        ...,
        [110,   0,   0, ..., 162,  59, 160],
        [ 38,   0,   0, ..., 177,  90, 155],
        [ 18,   0,   0, ..., 139, 101,  82]], dtype=uint8),
 array([[215, 215, 217, ..., 211, 210, 208],
        [216, 218, 219, ..., 214, 212, 209],
        [218, 219, 220, ..., 214, 212, 211],
        ...,
        [123, 126, 127, ..., 116, 113, 114],
        [121, 122, 125, ..., 115, 112, 111],
        [119, 118, 121, ..., 111, 109, 109]], dtype=uint8),
 array([[116, 118, 118, ...,  39,  39,  40],
        [119, 119, 122, ...,  40,  40,  39],
        [123, 125, 126, ...,  41,  41,  40],
        ...,
        [  4,   8,  12, ...,   6,   5,   4],
        [  6,  11,   4, ...,   8,   8,   3],
        [ 10,   7,   8, ...,   2,   5,   7]], dtype=uint8),
 array([[153, 154, 156, ..., 127, 122, 122],
        [156, 153, 156, ..., 127, 123, 118],
        [153, 154, 160, ..., 124, 127, 125],
        ...,
        [105,  91,  93, ...,  86,  64,  61],
        [ 72,  98,  53, ...,  90,  90,  87],
        [112,  86,  70, ...,  71,  57,  62]], dtype=uint8),
 array([[147, 148, 145, ...,  72,  71,  67],
        [149, 149, 149, ...,  73,  69,  69],
        [152, 151, 150, ...,  74,  71,  69],
        ...,
        [103, 104, 107, ...,  59,  56,  56],
        [ 99, 102, 101, ...,  57,  56,  53],
        [ 97,  97,  97, ...,  53,  53,  51]], dtype=uint8),
 array([[213, 214, 212, ..., 214, 213, 212],
        [216, 217, 217, ..., 218, 217, 218],
        [219, 219, 220, ..., 220, 220, 218],
        ...,
        [232, 227, 246, ..., 235, 234, 235],
        [235, 246, 243, ..., 231, 234, 236],
        [235, 243, 237, ..., 236, 239, 229]], dtype=uint8),
```

```
array([[ 96,  94,  92, ...,  86,  80,  88],
       [ 91,  95,  89, ..., 100,  84,  92],
       [ 85,  79,  67, ...,  97, 103, 100],
       ...,
       [183, 191, 197, ..., 101, 100,  95],
       [177, 173, 180, ...,  87,  92, 100],
       [167, 176, 179, ...,  75,  89,  96]], dtype=uint8),
array([[165, 165, 165, ..., 159, 158, 158],
       [165, 165, 167, ..., 159, 159, 158],
       [163, 166, 165, ..., 160, 159, 158],
       ...,
       [163, 164, 167, ..., 158, 157, 156],
       [164, 165,  99, ..., 156, 154, 156],
       [166,  66, 101, ..., 157, 155, 155]], dtype=uint8)]
```

Step 5: Represent every image I as a vector T. Now for each image you will have n^2x1 vector where n is 50

In [ ]:
```
n_squared = {}

for key in range(len(resized_faces)):
    vector = np.transpose(resized_faces[key]).reshape(2500, 1)
    n_squared[key] = vector

n_squared
```

Out[ ]:
```
{0: array([[84],
       [84],
       [86],
       ...,
       [14],
       [16],
       [43]], dtype=uint8),
 1: array([[134],
       [122],
       [143],
       ...,
       [113],
       [ 62],
       [ 47]], dtype=uint8),
 2: array([[ 11],
       [ 12],
       [ 14],
       ...,
       [160],
       [155],
       [ 82]], dtype=uint8),
 3: array([[215],
       [216],
       [218],
       ...,
       [114],
       [111],
       [109]], dtype=uint8),
 4: array([[116],
       [119],
       [123],
       ...,
       [  4],
       [  3],
       [  7]], dtype=uint8),
 5: array([[153],
       [156],
       [153],
       ...,
       [ 61],
       [ 87],
       [ 62]], dtype=uint8),
 6: array([[147],
       [149],
       [152],
       ...,
       [ 56],
       [ 53],
       [ 51]], dtype=uint8),
 7: array([[213],
       [216],
       [219],
       ...,
       [235],
       [236],
       [229]], dtype=uint8),
```

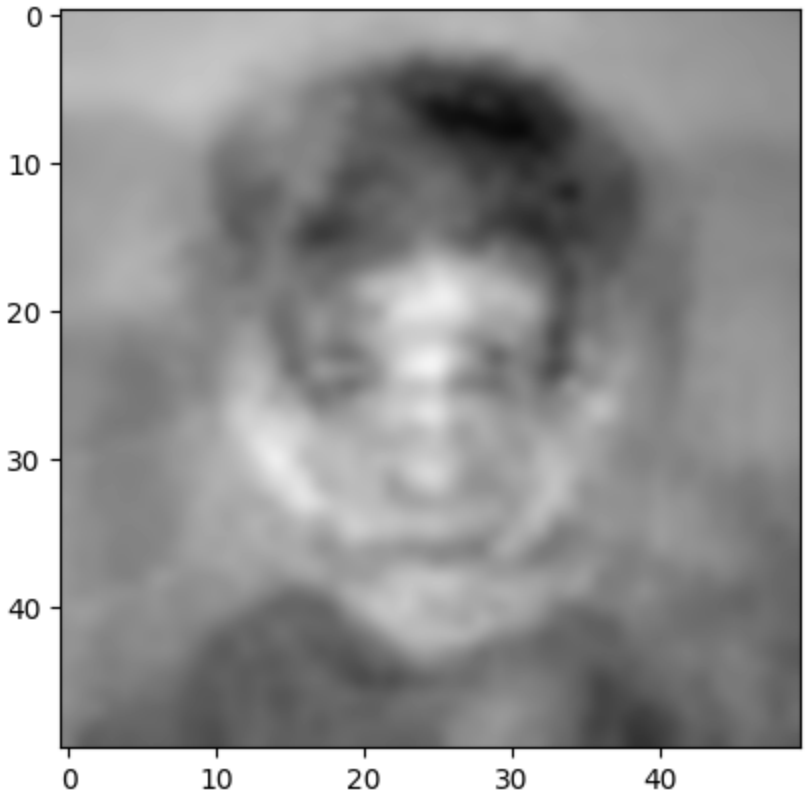
```
 8: array([[ 96],
        [ 91],
        [ 85],
        ...,
        [ 95],
        [100],
        [ 96]], dtype=uint8),
 9: array([[165],
        [165],
        [163],
        ...,
        [156],
        [156],
        [155]], dtype=uint8)}
```

In [ ]:
```
total_vector = np.concatenate((n_squared[0], n_squared[1], n_squared[2],
            n_squared[3], n_squared[4], n_squared[5],
            n_squared[6], n_squared[7], n_squared[8],
            n_squared[9]), axis=1)

total_vector.shape
```

Out[ ]:
```
(2500, 10)
```

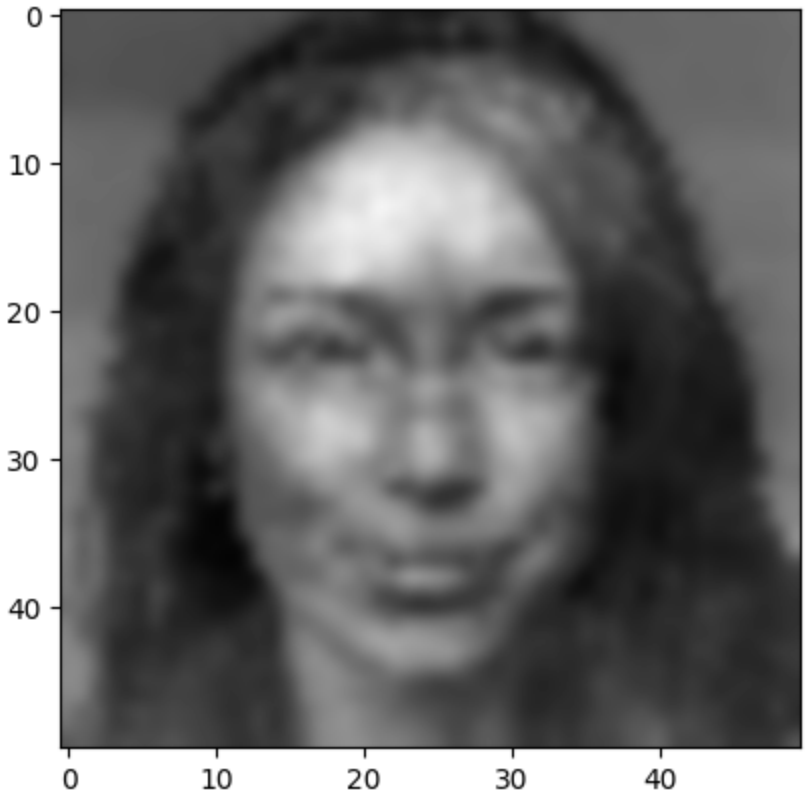
Step-6: compute the face vectors i.e. form a matrix that have each image vector in each column and compute the mean face. Display that face.

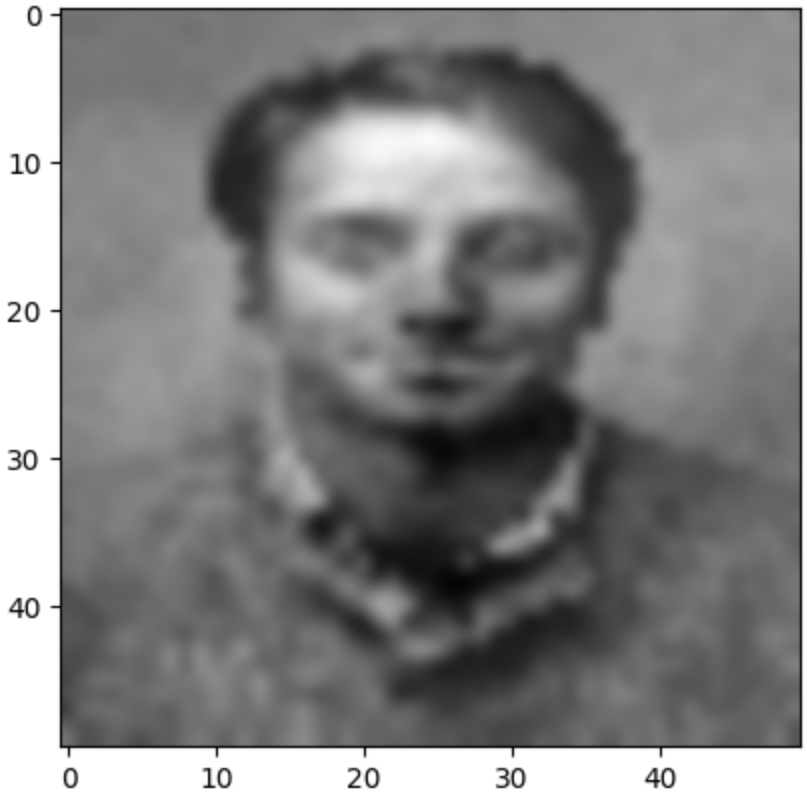
In [ ]:
```
avg_vector = np.mean(total_vector, axis=1).reshape(2500, 1)
avg_face = np.transpose(avg_vector.reshape(50, 50))
plt.imshow(avg_face, cmap='gray', interpolation='bicubic')
plt.show()
```

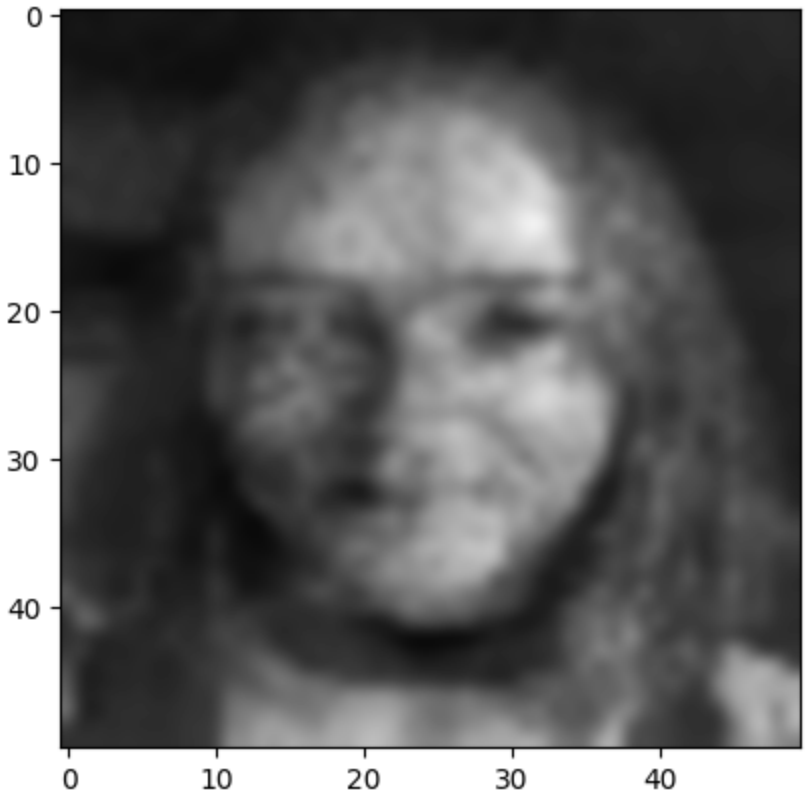


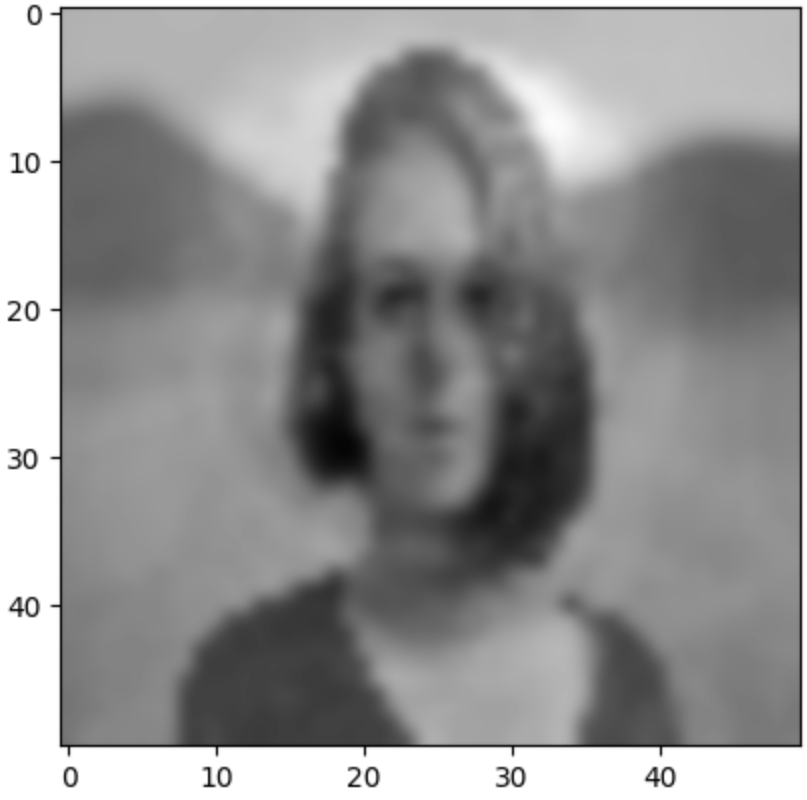
Step-7: Subtract the average face vector from the face vectors.

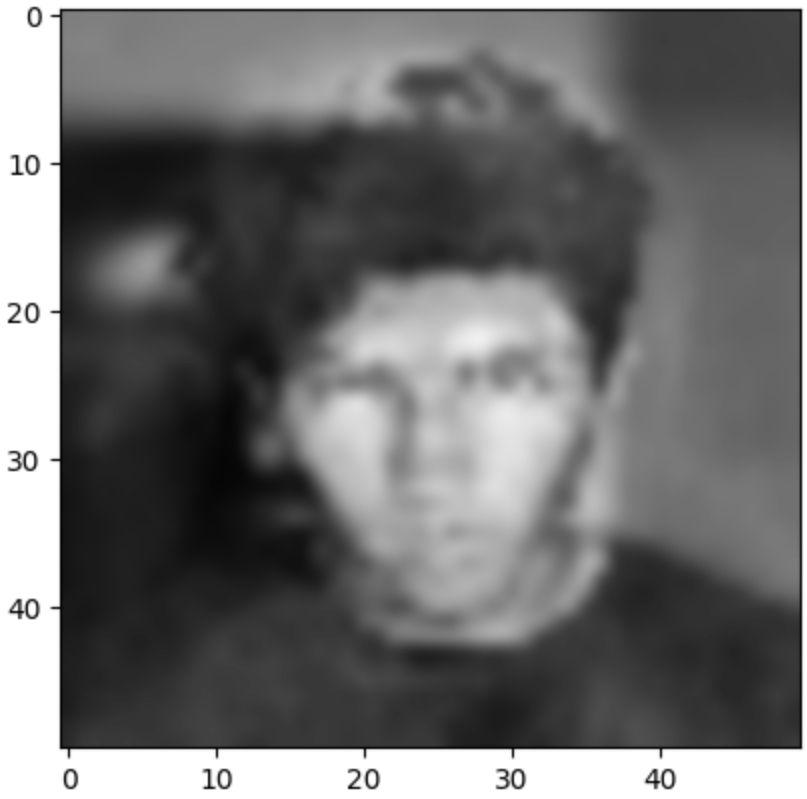
In [ ]:
```
subtracted_avg = []

for i in range(len(resized_faces)):
    subtracted_avg.append(resized_faces[i] - avg_face)

faces_diff = total_vector - avg_vector


for j, face in enumerate(subtracted_avg):
    plt.imshow(subtracted_avg[j], cmap='gray', interpolation='bicubic')
    plt.show()
```

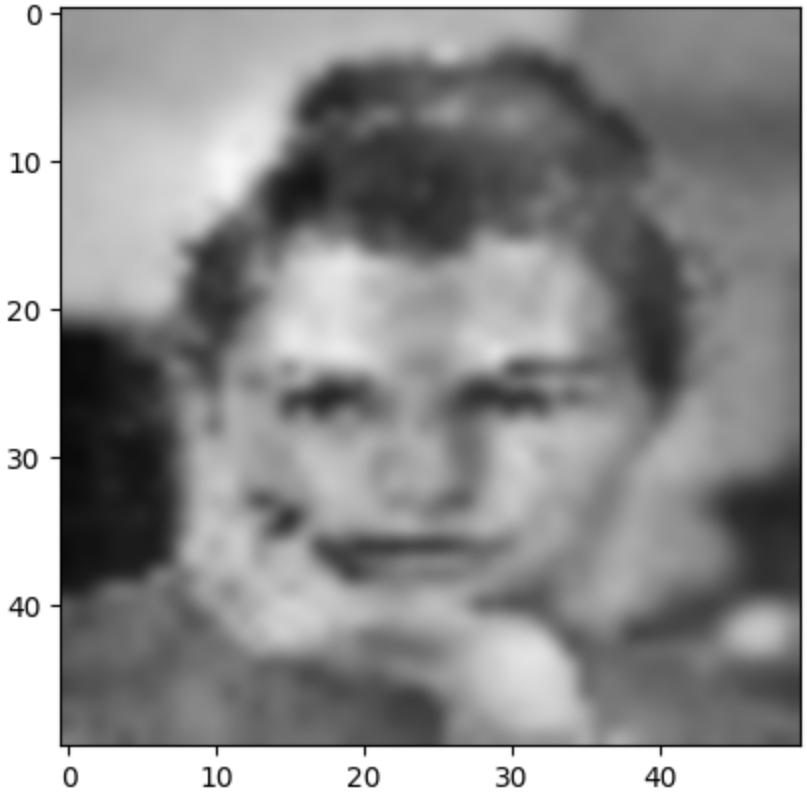

0
10
20
30
40
0
10
20
30
40

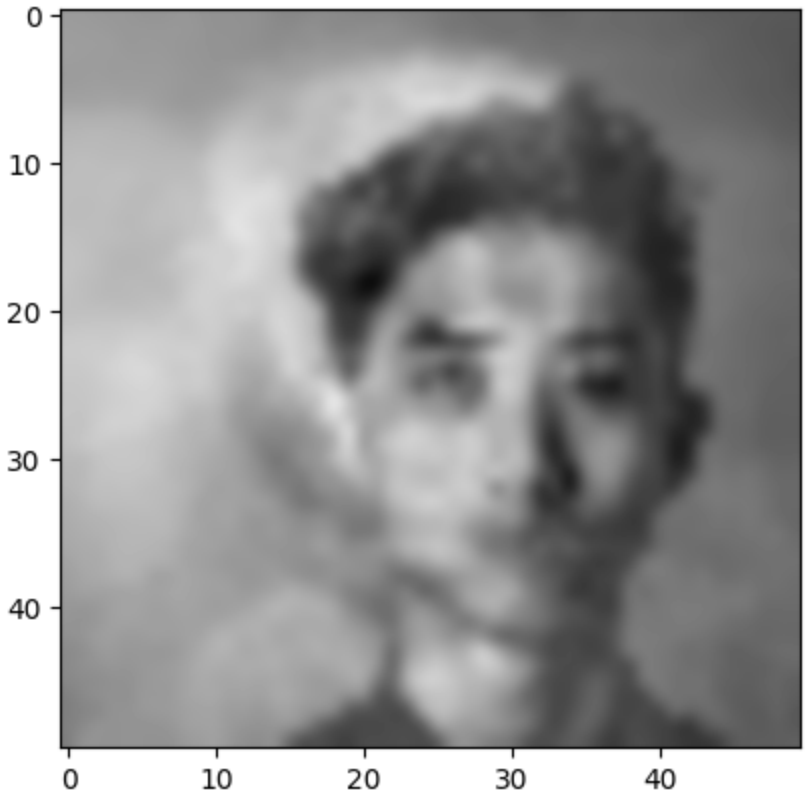

0
10
20
30
40
0
10
20
30
40

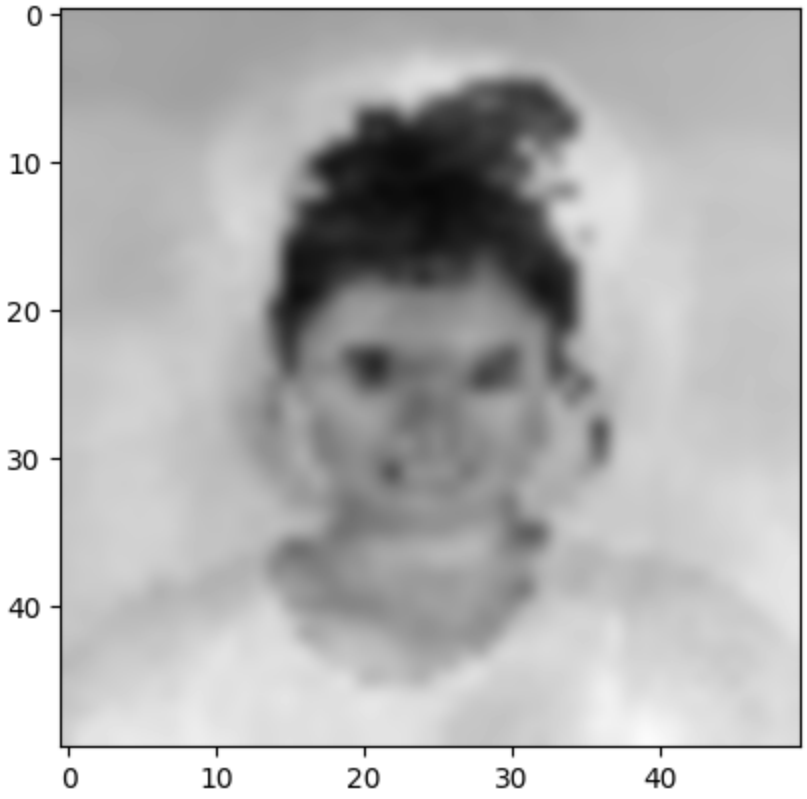

0
10
20
30
40
0
10
20
30
40

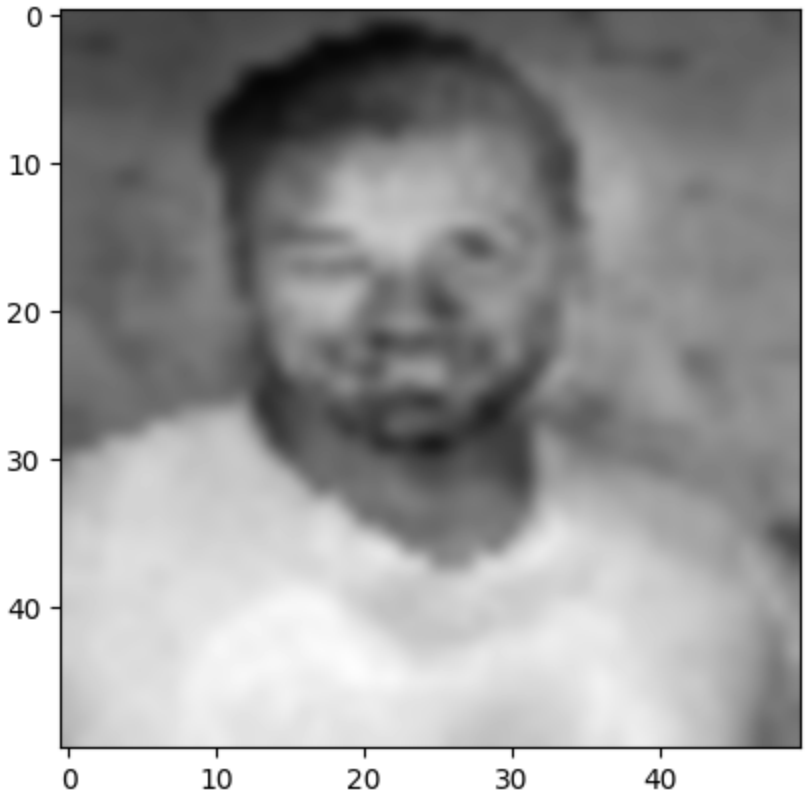

0
10
20
30
40
0
10
20
30
40

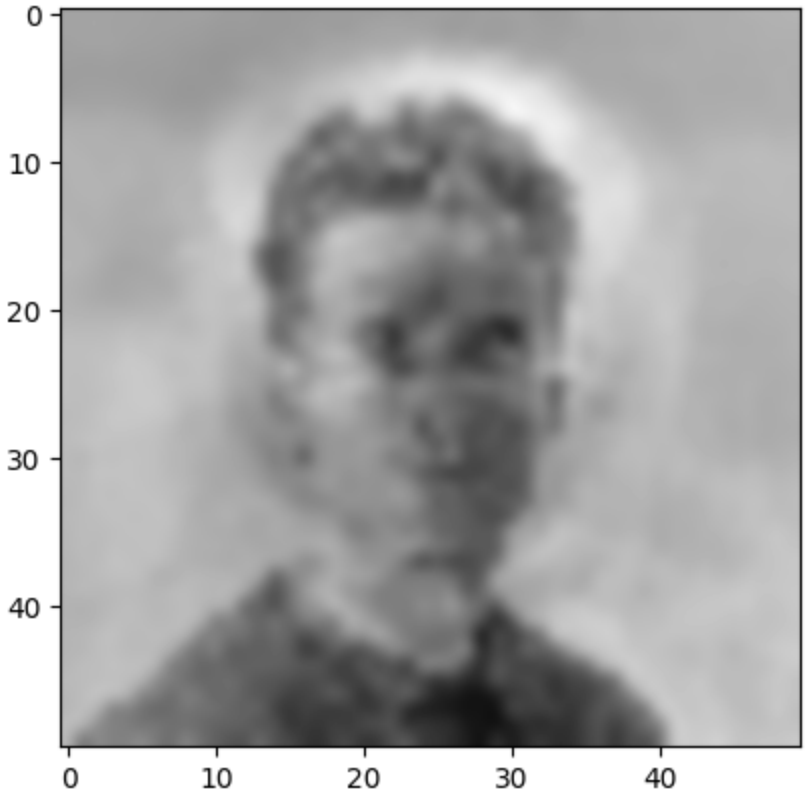

0
10
20
30
40
0
10
20
30
40


0
10
20
30
40
0
10
20
30
40


0
10
20
30
40
0
10
20
30
40


0
10
20
30
40
0
10
20
30
40





Step-8: Calculate the covariance matrix, which results in n x n matrix.

In [ ]:
```
cov = np.cov(np.transpose(faces_diff))
cov.shape
```

Out[ ]: (10, 10)

Step-9: Calculate the eigenvalues and eigenvectors from the covariance matrix.

In [ ]:
```
eValues, eVectors = np.linalg.eigh(cov)
```

Step-10: Choose the K best eigenvectors from step-9.

In [ ]:
```
best_eValues = np.array((eValues[2], eValues[3], eValues[4]))
best_eVectors = np.array((eVectors[2], eVectors[3], eVectors[4]))
```

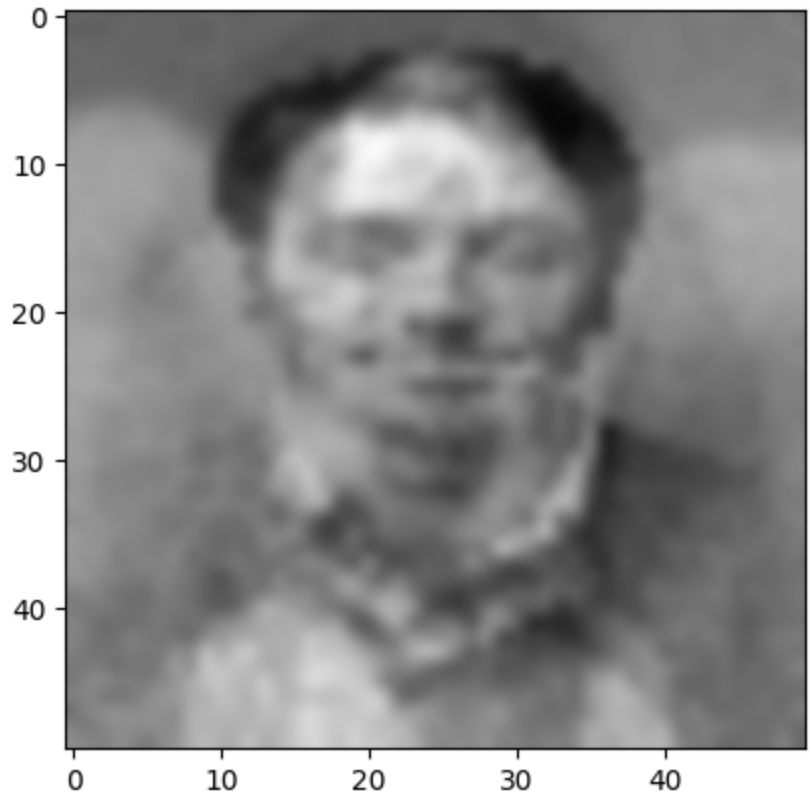
Step-11: Multiply each eigenvalues i.e. eigen vectors with the (face vector - average face vector) i.e. step-7

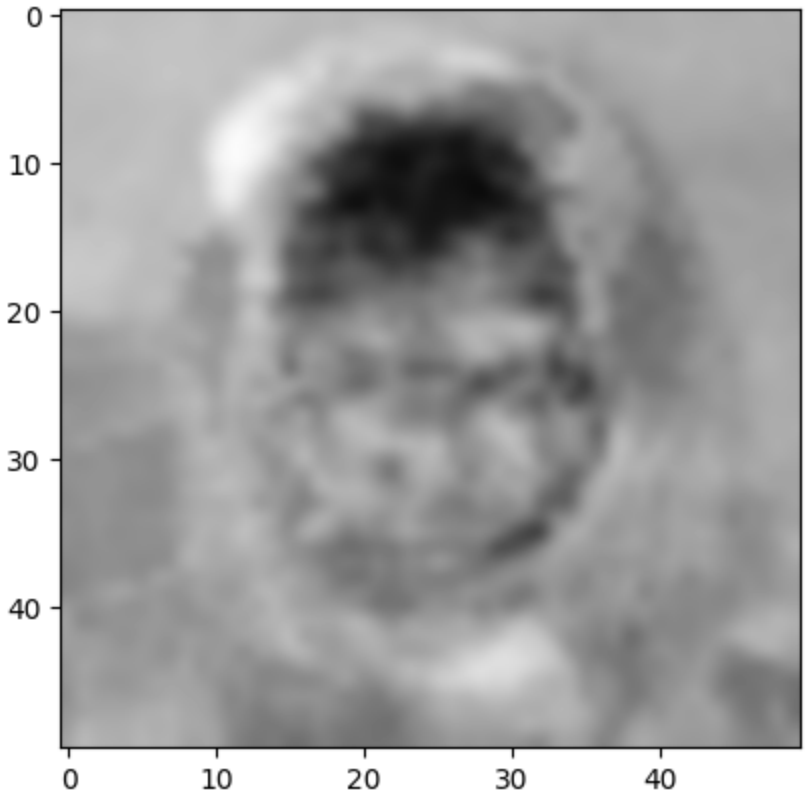
In [ ]:
```
mult_eVectors = np.transpose((eVectors).dot(np.transpose(faces_diff)))
mult_eVectors.shape
```

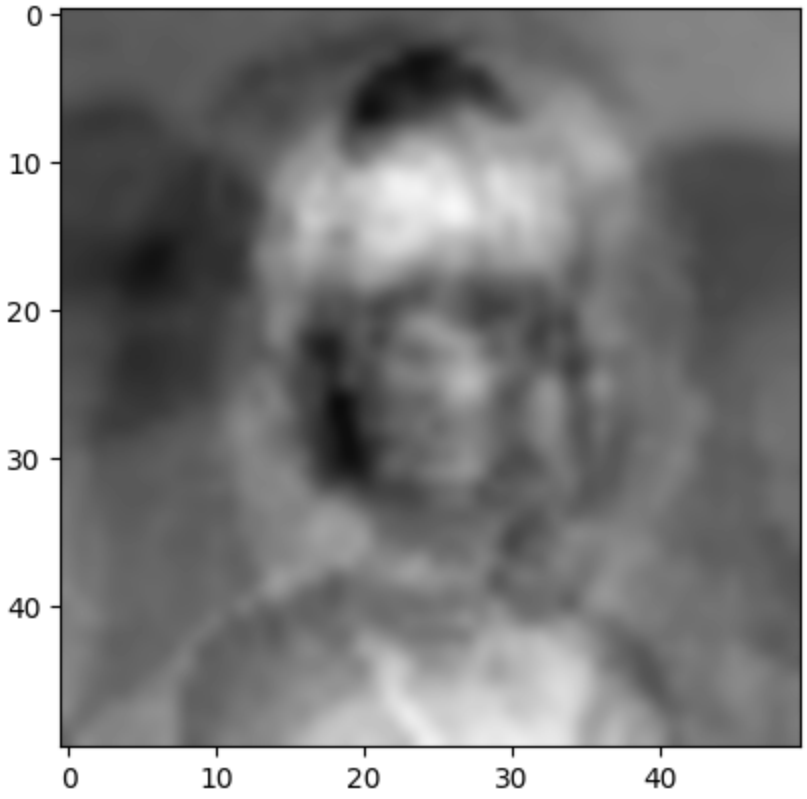
Out[ ]: (2500, 10)

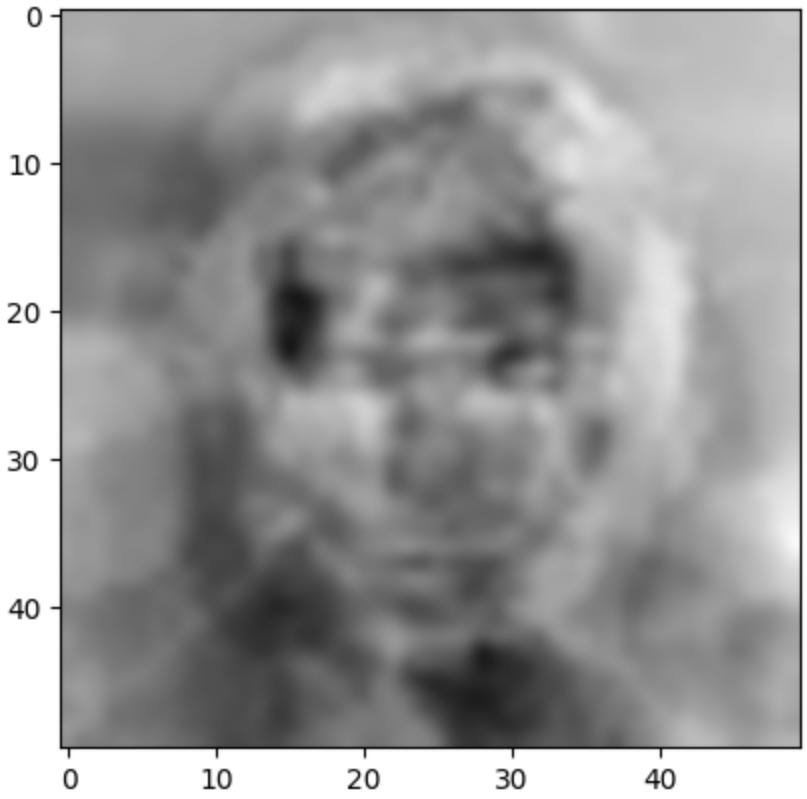
Step-12: Graphically display each face with respect to the eigenvalues.

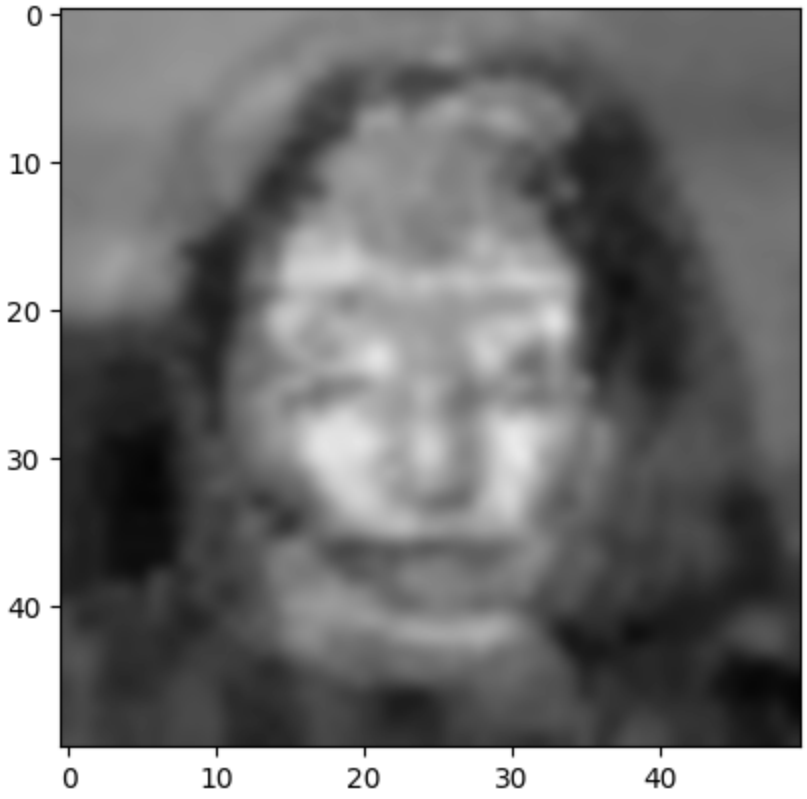
In [ ]:
```
for vector in np.transpose(mult_eVectors):
    plt.imshow(np.transpose(vector.reshape(50, 50)), cmap='gray', interpolation='bi
    plt.show()
```

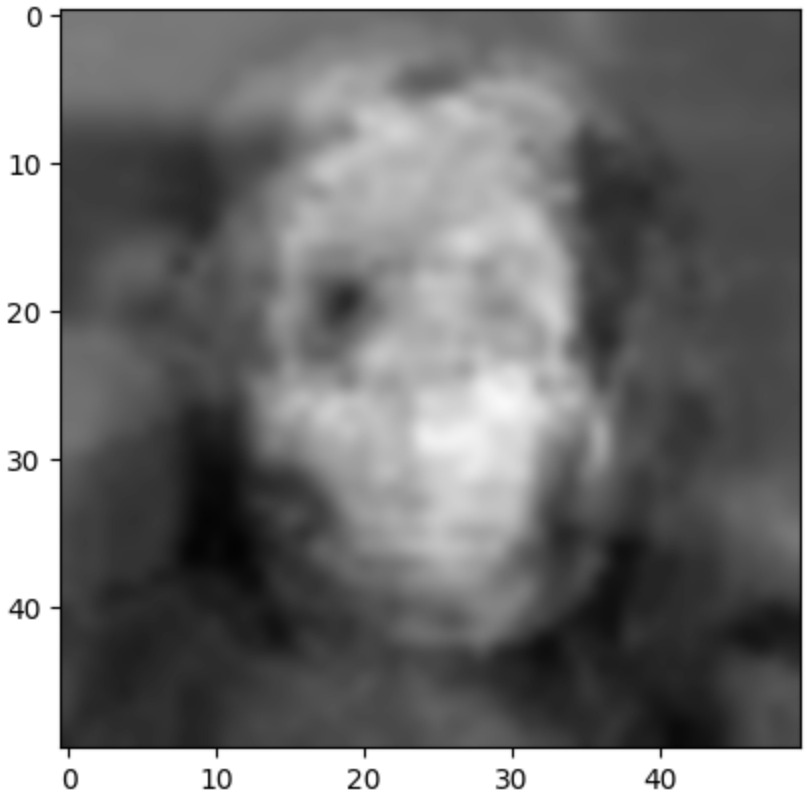


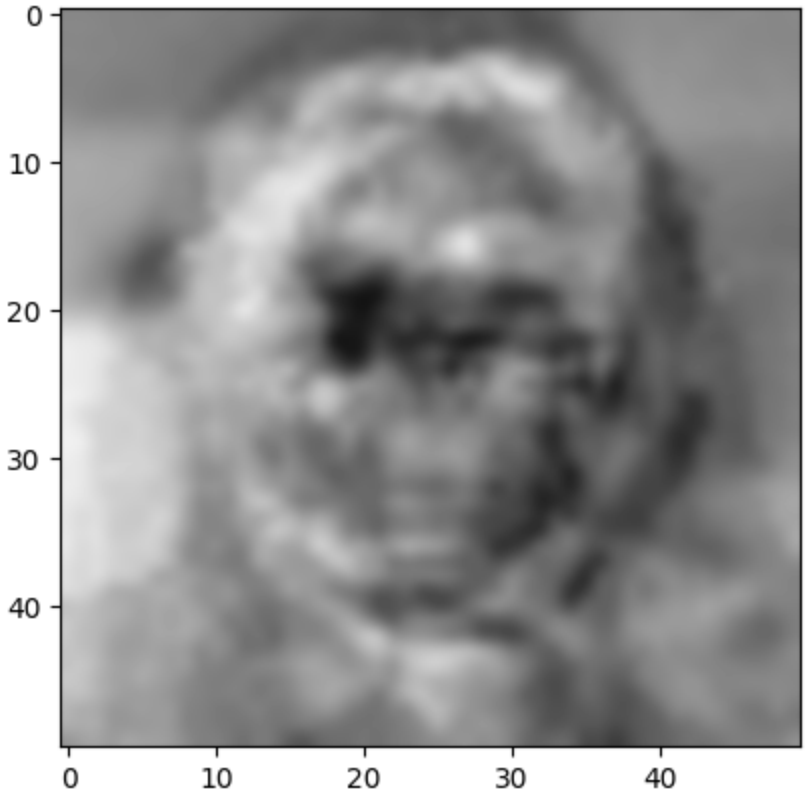

0
10
20
30
40
0
10
20
30
40

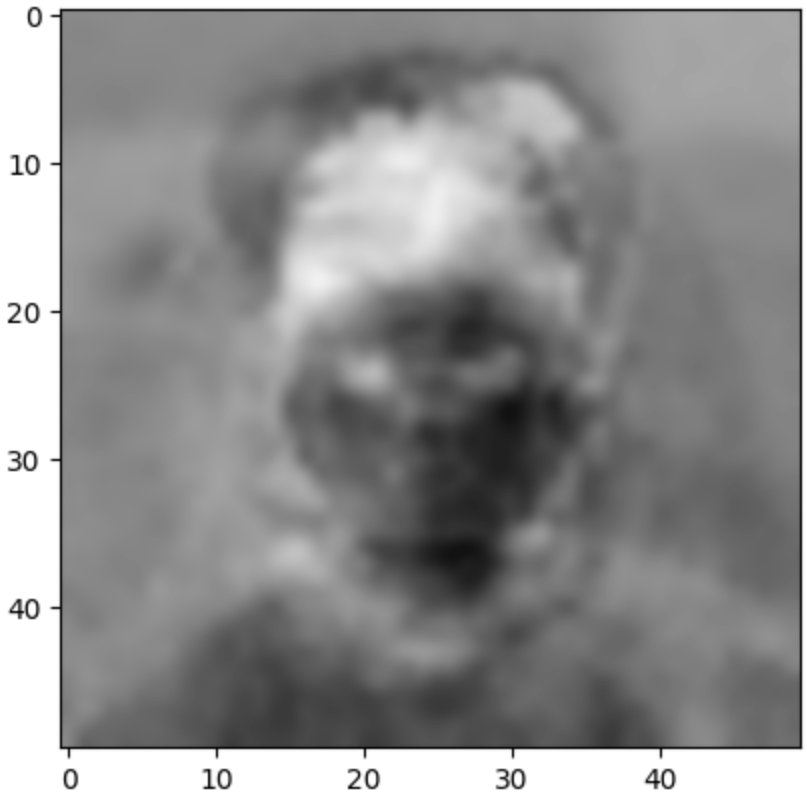

0
10
20
30
40
0
10
20
30
40

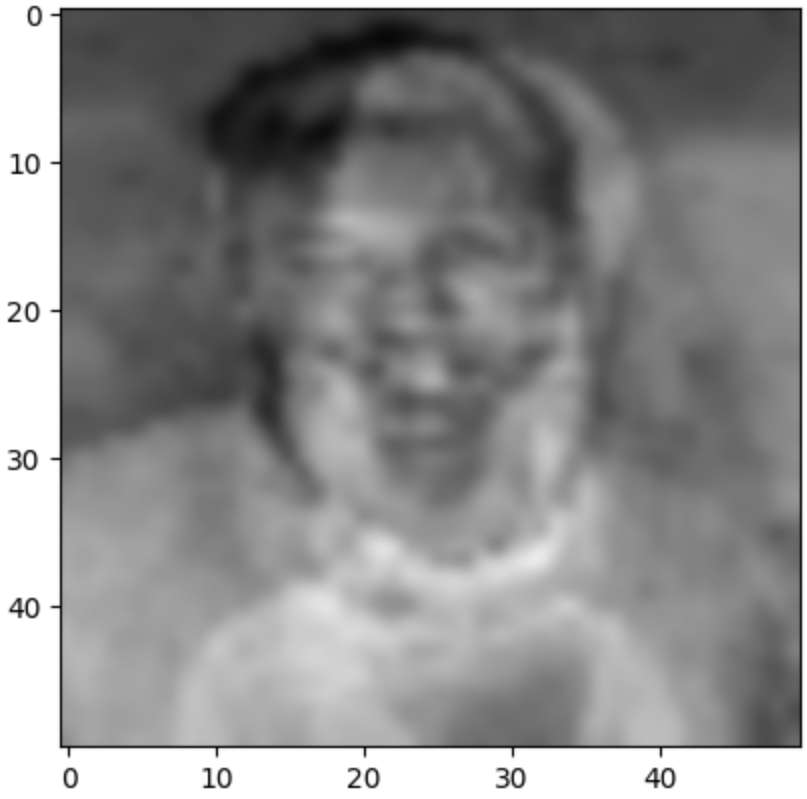

0
10
20
30
40
0
10
20
30
40

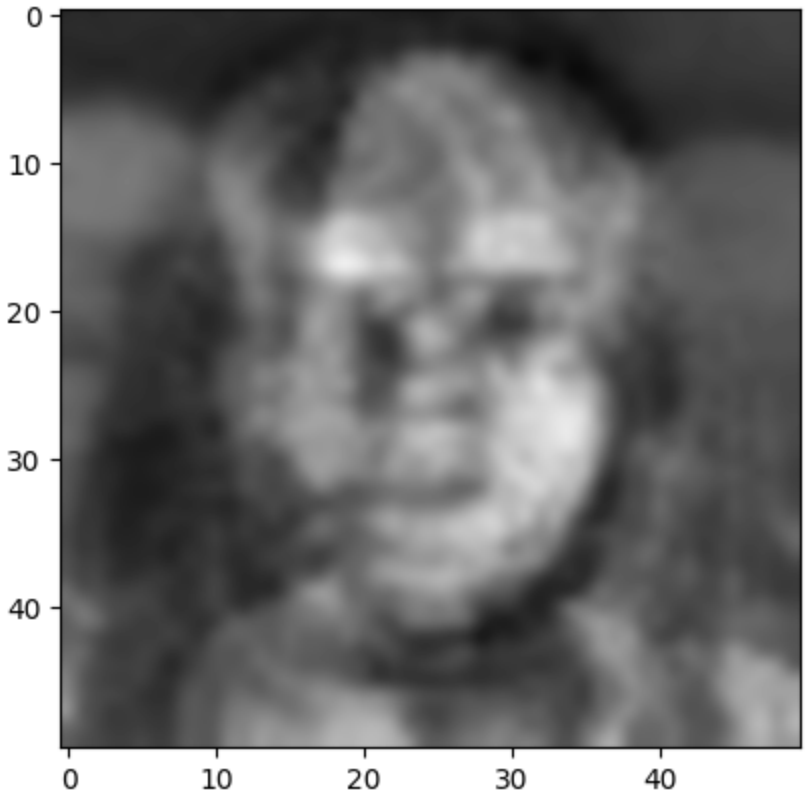

0
10
20
30
40
0
10
20
30
40


0
10
20
30
40
0
10
20
30
40


0
10
20
30
40
0
10
20
30
40


0
10
20
30
40
0
10
20
30
40


0
10
20
30
40
0
10
20
30
40



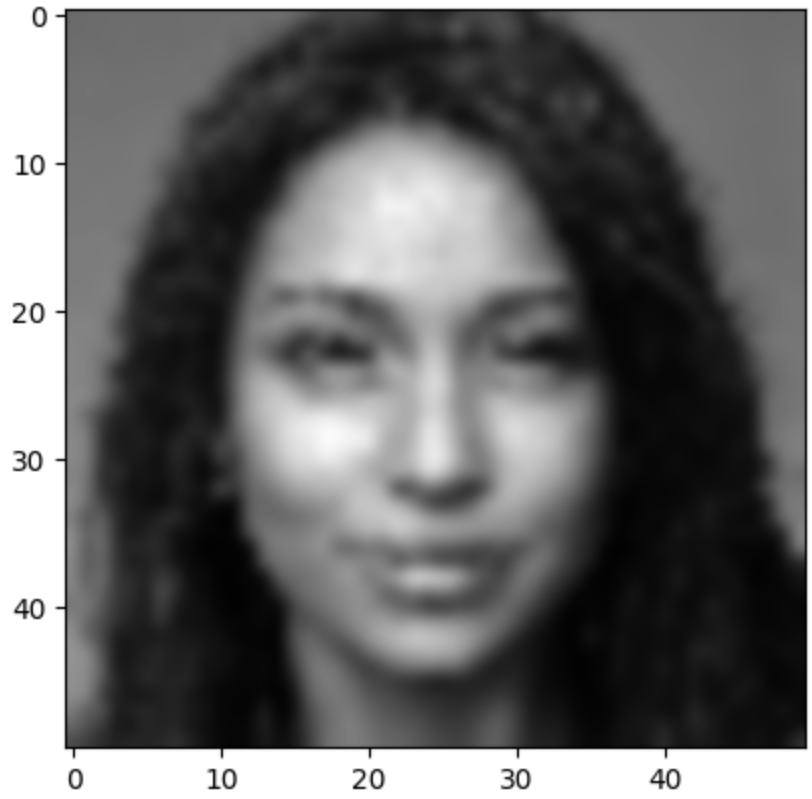
Step-13: Read the test image and separate the face from the image. If you already have a separated face image i.e. image which have a face centered and resized to 50 x 50, you can skip Step-13

This step has been skipped

Step-14: Calculate the feature vector of the test face and subtract it with the average face.

In [ ]:
```
test_face = resized_faces[0]
plt.imshow(test_face, cmap='gray', interpolation='bicubic')
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x1c4021faa70>
```



In [ ]:
```
test_face = test_face.reshape(2500,1)
test_face_avg = test_face - avg_vector
test_face_avg.shape
```

Out[ ]:
```
(2500, 1)
```

Step-15: Project the test image on the eigenspace.

In [ ]:
```
projected = np.transpose(test_face).dot(mult_eVectors)
```

Step-16: Calculate the Euclidean distance (e) it with each eigenface vectors.

In [ ]:
```
projected_transposed = np.transpose(projected)
weight = np.transpose(faces_diff).dot(mult_eVectors)
```

In [ ]:
```
e_distance = np.argmin(np.linalg.norm(projected_transposed - weight, axis=1))
e_distance
```

Out[ ]:
```
6
```

The best index is 6, or "image5" in the eigenvectors list