Hello Soft Clustering (GMM)

T1. Using 3 mixtures, initialize your Gaussian with means (3,3), (2,2), and (-3,-3), and standard Covariance, I, the identity matrix. Use equal mix- ture weights as the initial weights. Repeat three iterations of EM. Write down $w_{n,j}$, m_j, μ_j, Σ_j for each EM iteration. (You may do the calculations by hand or write code to do so)

$$w_{n,j} = \frac{p(x_n; \vec{\mu_j}, \Sigma_j) m_j}{\Sigma_j p(x_n; \vec{\mu_j}, \Sigma_j) m_j} \tag{1}$$

$w_{n,j}$ means the probability that data point n comes from Gaussian number j.

Maximization: Update the model parameters, ϕ, $\vec{\mu_j}$, Σ_j.

$$m_j = \frac{1}{N} \Sigma_n w_{n,j} \tag{2}$$

$$\vec{\mu_j} = \frac{\Sigma_n w_{n,j} \vec{x_n}}{\Sigma_n w_{n,j}} \tag{3}$$

$$\Sigma_j = \frac{\Sigma_n w_{n,j} (\vec{x_n} - \vec{\mu_j})(\vec{x_n} - \vec{\mu_j})^T}{\Sigma_n w_{n,j}} \tag{4}$$

The above equation is used for full covariance matrices. For our small toy example, we will use diagonal covariance matrices, which can be acquired by setting the off-diagonal values to zero. In other words, $\Sigma_{(i,j)} = 0$, for $i \neq j$.

TODO: Complete functions below including

- Fill relevant parameters in each function.
- Implement computation and return values.

These functions will be used in T1-4.

In [1]:
```python
import numpy as np
import matplotlib.pyplot as plt

# Hint: You can use this function to get gaussian distribution.
# https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.multivariate_normal.html
from scipy.stats import multivariate_normal
```

In [2]:
```python
multivariate_normal.pdf([[1, 2]] * 3, mean=[1, 1], cov=[[1, 0], [0, 1]])
```

Out[2]:
```
array([0.09653235, 0.09653235, 0.09653235])
```

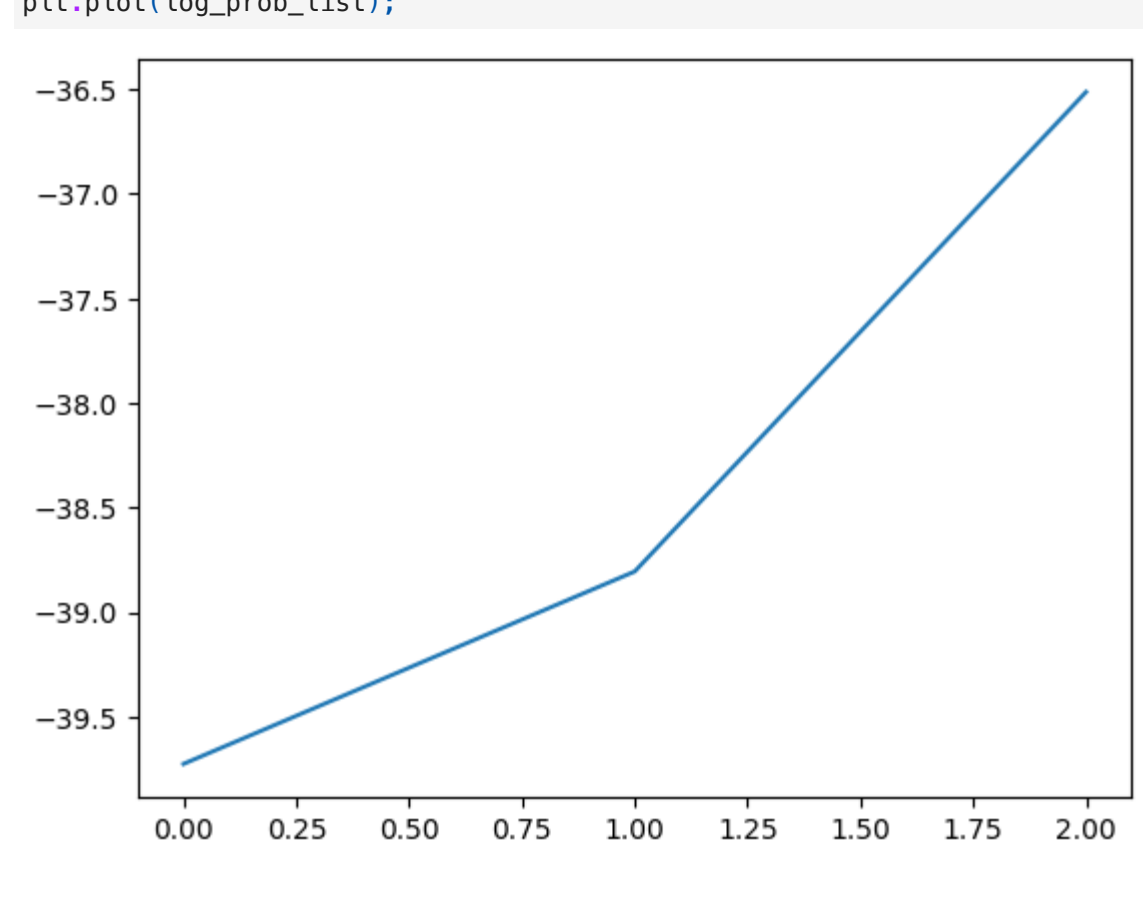
In [3]:
```python
class GMM:
    def __init__(self, mixture_weight, mean_params, cov_params):
        """
        Initialize GMM.
        """
        # Copy construction values.
        self.mixture_weight = mixture_weight
        self.mean_params = mean_params
        self.cov_params = cov_params

        # Initiailize iteration.
        self.n_iter = 0

    def estimation_step(self, data):
        """
        TODO: Perform estimation step. Then, return w_{n,j} in eq. 1)
        """

        # INSERT CODE HERE
        w = np.array([multivariate_normal.pdf(data, mean=self.mean_params[i], cov=self.cov_params[i]) * self.

        """
        print(w.shape)
        print(w)
        print(w.sum(axis=1))
        print(np.array([w.sum(axis=1)] * 3).T)
        """

        w = w / w.sum(axis=1)[:, None] # norm probs across mixtures

        """
        print(w.shape)
        print(w.sum(axis=1))
        """

        return w  # assigned soft labels (data points, mixtures)

    def maximization_step(self, data, w):
        """
        TODO: Perform maximization step.
            (Update parameters in this GMM model.)
        """
        # INSERT CODE HERE
        self.mixture_weight = w.sum(axis=0) / len(data)

        self.mean_params = np.matmul(w.T, data) / w.sum(axis=0)[:, None]

        self.cov_params = np.array([np.matmul((data - self.mean_params[i]).T, (data - self.mean_params[i]) *
        self.cov_params = np.array(
            [
                self.cov_params[i] * np.eye(self.cov_params[i].shape[0])
                for i in range(len(self.mixture_weight))
            ]
        )

    def get_log_likelihood(self, data):
        """
        TODO: Compute log likelihood.
        """
        # INSERT CODE HERE
        log_prob = np.log(np.array([multivariate_normal.pdf(data, mean=self.mean_params[i], cov=self.cov_para

        return log_prob

    def print_iteration(self):
        print("m :\n", self.mixture_weight)
        print("mu :\n", self.mean_params)
        print("covariance matrix :\n", self.cov_params)
        print("-------------------------------------------------------------")

    def perform_em_iterations(self, data, num_iterations, display=True):
        """
        Perform estimation & maximization steps with num_iterations.
        Then, return list of log_likelihood from those iterations.
        """
        log_prob_list = []

        # Display initialization.
        if display:
            print("Initialization")
            self.print_iteration()

        for n_iter in range(num_iterations):

            # TODO: Perform EM step.

            # INSERT CODE HERE
            w = self.estimation_step(data)
            self.maximization_step(data, w)

            self.print_iteration()

            # Calculate log prob.
            log_prob = self.get_log_likelihood(data)
            log_prob_list.append(log_prob)

            # Display each iteration.
            if display:
                print(f"Iteration: {n_iter}")
                self.print_iteration()

        return log_prob_list
```

In [4]:
```python
num_iterations = 3
num_mixture = 3
mixture_weight = [1] * num_mixture # m
mean_params = np.array([[3,3], [2,2], [-3,-3]], dtype = float)
cov_params = np.array([np.eye(2)] * num_mixture)

X, Y = np.array([1, 3, 2, 8, 6, 7, -3, -2, -7]), np.array([2, 3, 2, 8, 6, 7, -3, -4, -7])
data = np.vstack([X,Y]).T

gmm = GMM(mixture_weight, mean_params, cov_params)

log_prob_list = gmm.perform_em_iterations(data, num_iterations)
```

```
Initialization
m :
 [1, 1, 1]
mu :
 [[ 3.  3.]
 [ 2.  2.]
 [-3. -3.]]
covariance matrix :
 [[[1. 0.]
  [0. 1.]]

 [[1. 0.]
  [0. 1.]]

 [[1. 0.]
  [0. 1.]]]
-------------------------------------------------------------
m :
 [0.45757242 0.20909425 0.33333333]
mu :
 [[ 5.78992692  5.81887265]
 [ 1.67718211  2.14523106]
 [-4.         -4.66666666]]
covariance matrix :
 [[[4.53619412 0.        ]
  [0.         4.28700611]]

 [[0.51645579 0.        ]
  [0.         0.13152618]]

 [[4.66666668 0.        ]
  [0.         2.88888891]]]
-------------------------------------------------------------
Iteration: 0
m :
 [0.45757242 0.20909425 0.33333333]
mu :
 [[ 5.78992692  5.81887265]
 [ 1.67718211  2.14523106]
 [-4.         -4.66666666]]
covariance matrix :
 [[[4.53619412 0.        ]
  [0.         4.28700611]]

 [[0.51645579 0.        ]
  [0.         0.13152618]]

 [[4.66666668 0.        ]
  [0.         2.88888891]]]
-------------------------------------------------------------
m :
 [0.40711618 0.25954961 0.33333421]
mu :
 [[ 6.27176215  6.27262711]
 [ 1.72091544  2.14764812]
 [-3.99998589 -4.6666488 ]]
covariance matrix :
 [[[2.94672736 0.        ]
  [0.         2.93847196]]

 [[0.49649261 0.        ]
  [0.         0.12584815]]

 [[4.66673088 0.        ]
  [0.         2.88900236]]]
-------------------------------------------------------------
Iteration: 1
m :
 [0.40711618 0.25954961 0.33333421]
mu :
 [[ 6.27176215  6.27262711]
 [ 1.72091544  2.14764812]
 [-3.99998589 -4.6666488 ]]
covariance matrix :
 [[[2.94672736 0.        ]
  [0.         2.93847196]]

 [[0.49649261 0.        ]
  [0.         0.12584815]]

 [[4.66673088 0.        ]
  [0.         2.88900236]]]
-------------------------------------------------------------
m :
 [0.36070909 0.30595677 0.33333414]
mu :
 [[ 6.6962644   6.69629468]
 [ 1.91071238  2.27383436]
 [-3.99998673 -4.6666501 ]]
covariance matrix :
 [[[1.73961067 0.        ]
  [0.         1.73929602]]

 [[0.62898406 0.        ]
  [0.         0.1988491 ]]

 [[4.66672942 0.        ]
  [0.         2.88899545]]]
-------------------------------------------------------------
Iteration: 2
m :
 [0.36070909 0.30595677 0.33333414]
mu :
 [[ 6.6962644   6.69629468]
 [ 1.91071238  2.27383436]
 [-3.99998673 -4.6666501 ]]
covariance matrix :
 [[[1.73961067 0.        ]
  [0.         1.73929602]]

 [[0.62898406 0.        ]
  [0.         0.1988491 ]]

 [[4.66672942 0.        ]
  [0.         2.88899545]]]
-------------------------------------------------------------
```

T2. Plot the log likelihood of the model given the data after each EM step. In other words, plot $\log \prod_n p(\vec{x_n} | \phi, \vec{\mu}, \Sigma)$. Does it goes up every iteration just as we learned in class?

In [5]:
```python
# TODO
plt.plot(log_prob_list);
```




```
ANS : Yes
```

T3. Using 2 mixtures, initialize your Gaussian with means (3,3) and (-3,-3), and standard Covariance, I, the identity matrix. Use equal mixture weights as the initial weights. Repeat three iterations of EM. Write down $w_{n,j}$, m_j, $\vec{\mu_j}$, Σ_j for each EM iteration.

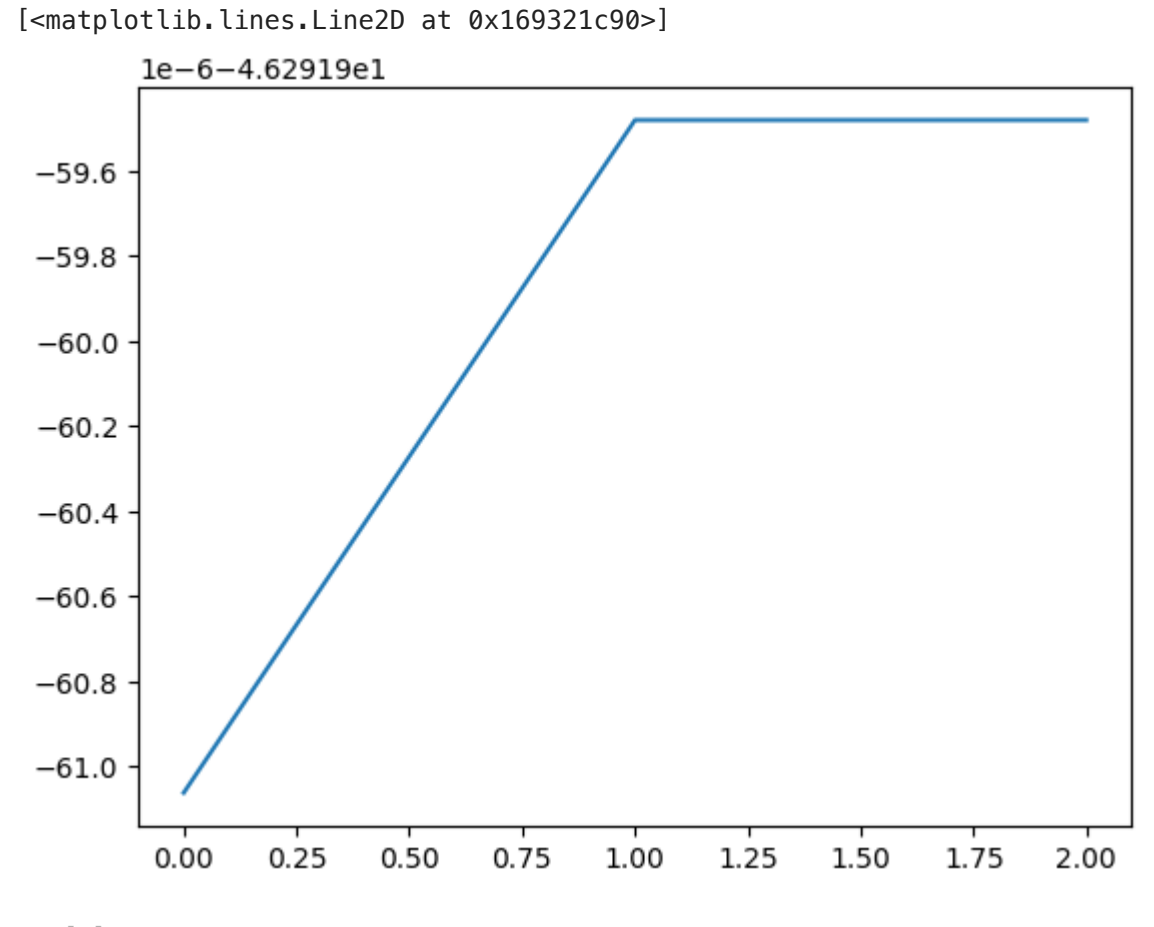
In [6]:
```python
num_mixture = 2
mixture_weight = [1] * num_mixture

mean_params = np.array([[3,3], [-3,-3]], dtype = float)
cov_params = np.array([np.eye(2)] * num_mixture)

# INSERT CODE HERE
gmm2 = GMM(mixture_weight, mean_params, cov_params)
log_prob_list2 = gmm2.perform_em_iterations(data, num_iterations)
```

```
Initialization
m :
 [1, 1]
mu :
 [[ 3.  3.]
 [-3. -3.]]
covariance matrix :
 [[[1. 0.]
  [0. 1.]]

 [[1. 0.]
  [0. 1.]]]
-------------------------------------------------------------
m :
 [0.66666666 0.33333334]
mu :
 [[ 4.50000001  4.66666667]
 [-3.99999997 -4.66666663]]
covariance matrix :
 [[[6.91666665 0.        ]
  [0.         5.88888889]]

 [[4.66666677 0.        ]
  [0.         2.8888891 ]]]
-------------------------------------------------------------
Iteration: 0
m :
 [0.66666666 0.33333334]
mu :
 [[ 4.50000001  4.66666667]
 [-3.99999997 -4.66666663]]
covariance matrix :
 [[[6.91666665 0.        ]
  [0.         5.88888889]]

 [[4.66666677 0.        ]
  [0.         2.8888891 ]]]
-------------------------------------------------------------
m :
 [0.66669436 0.33330564]
mu :
 [[ 4.49961311  4.66620178]
 [-3.99993241 -4.66651231]]
covariance matrix :
 [[[6.91944755 0.        ]
  [0.         5.89275124]]

 [[4.66806942 0.        ]
  [0.         2.89103318]]]
-------------------------------------------------------------
Iteration: 1
m :
 [0.66669436 0.33330564]
mu :
 [[ 4.49961311  4.66620178]
 [-3.99993241 -4.66651231]]
covariance matrix :
 [[[6.91944755 0.        ]
  [0.         5.89275124]]

 [[4.66806942 0.        ]
  [0.         2.89103318]]]
-------------------------------------------------------------
m :
 [0.66669453 0.33330547]
mu :
 [[ 4.49961084  4.66619903]
 [-3.99993206 -4.66651141]]
covariance matrix :
 [[[6.91946372 0.        ]
  [0.         5.8927741 ]]

 [[4.66807754 0.        ]
  [0.         2.89104566]]]
-------------------------------------------------------------
Iteration: 2
m :
 [0.66669453 0.33330547]
mu :
 [[ 4.49961084  4.66619903]
 [-3.99993206 -4.66651141]]
covariance matrix :
 [[[6.91946372 0.        ]
  [0.         5.8927741 ]]

 [[4.66807754 0.        ]
  [0.         2.89104566]]]
-------------------------------------------------------------
```

T4. Plot the log likelihood of the model given the data after each EM step. Compare the log likelihood between using two mixtures and three mixtures. Which one has the better likelihood?

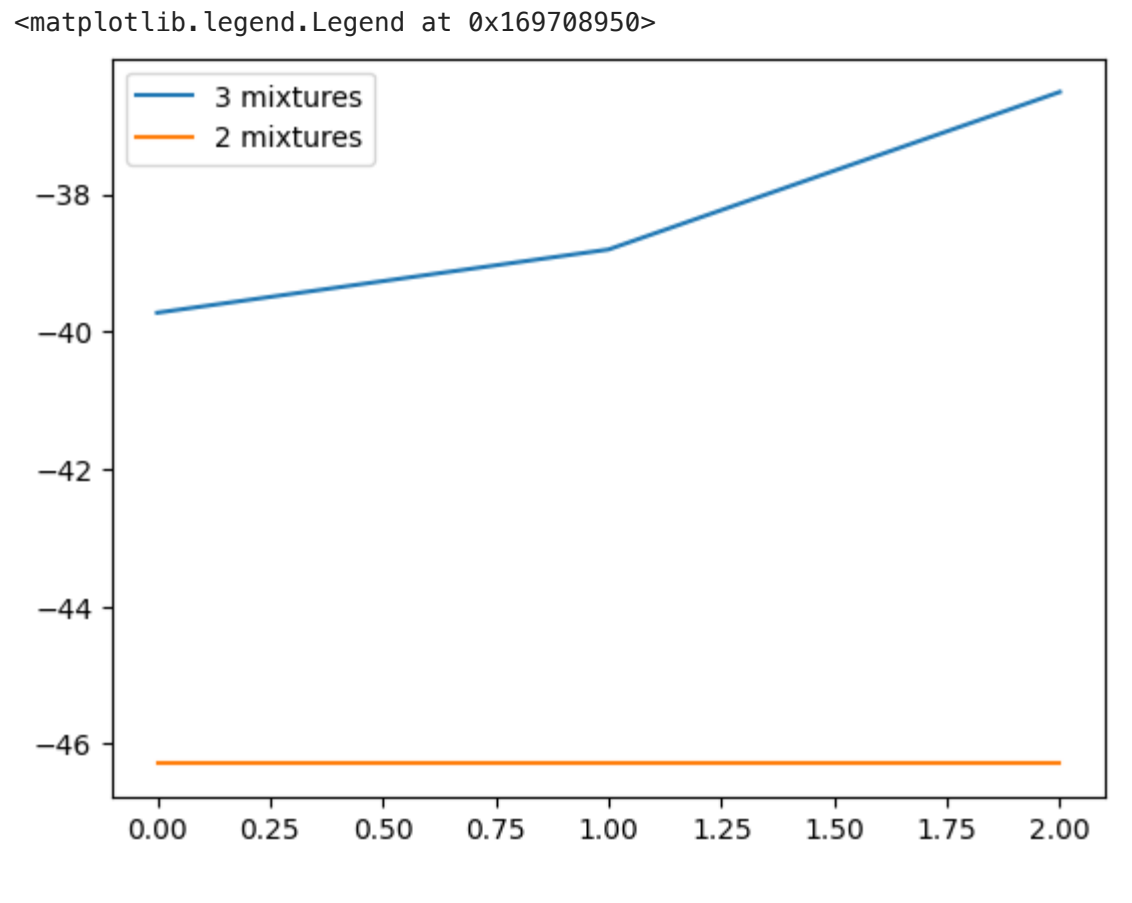
In [7]:
```python
# TODO: Plot log_likelihood from T3
plt.plot(log_prob_list2)
```

Out[7]:
```
[<matplotlib.lines.Line2D at 0x169321c90>]
```



In [8]:
```python
# TODO: Plot Comparision of log_likelihood from T1 and T3
plt.plot(log_prob_list, label="3 mixtures")
plt.plot(log_prob_list2, label="2 mixtures")
plt.legend()
```

Out[8]:
```
<matplotlib.legend.Legend at 0x169708950>
```




```
ANS : 3 mixtures has better likelihood than 2 mixtures.
```

The face database

In [9]:
```python
# Download facedata for google colab
# !wget -nc https://github.com/ekapolc/Pattern_2024/raw/main/HW/HW03/facedata_mat.zip
# !unzip facedata_mat.zip
```

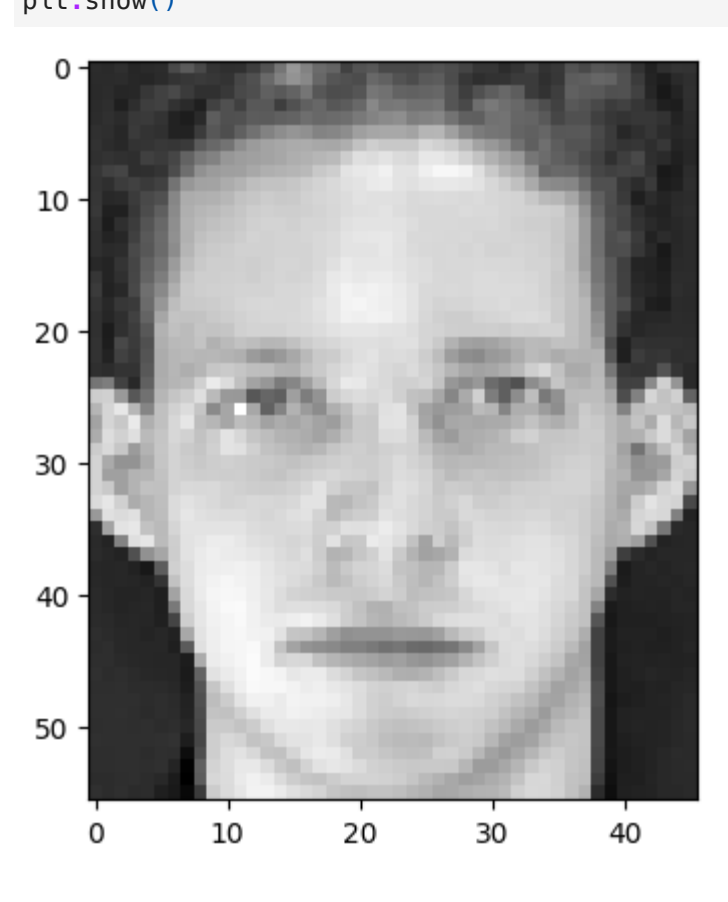
In [10]:
```python
import scipy.io
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from skimage import img_as_float

# Change path to your facedata.mat file.
facedata_path = 'facedata.mat'

data = scipy.io.loadmat(facedata_path)
data_size = data['facedata'].shape

%matplotlib inline
data_size
```

Out[10]:
```
(40, 10)
```

Preprocess xf

In [11]:
```python
xf = np.zeros((data_size[0], data_size[1], data['facedata'][0,0].shape[0], data['facedata'][0,0].shape[1]))
for i in range(data['facedata'].shape[0]):
    for j in range(data['facedata'].shape[1]):
        xf[i,j] = img_as_float(data['facedata'][i,j])
```

In [12]:
```python
# Example: Ploting face image.
plt.imshow(xf[0,0], cmap = 'gray')
plt.show()
```



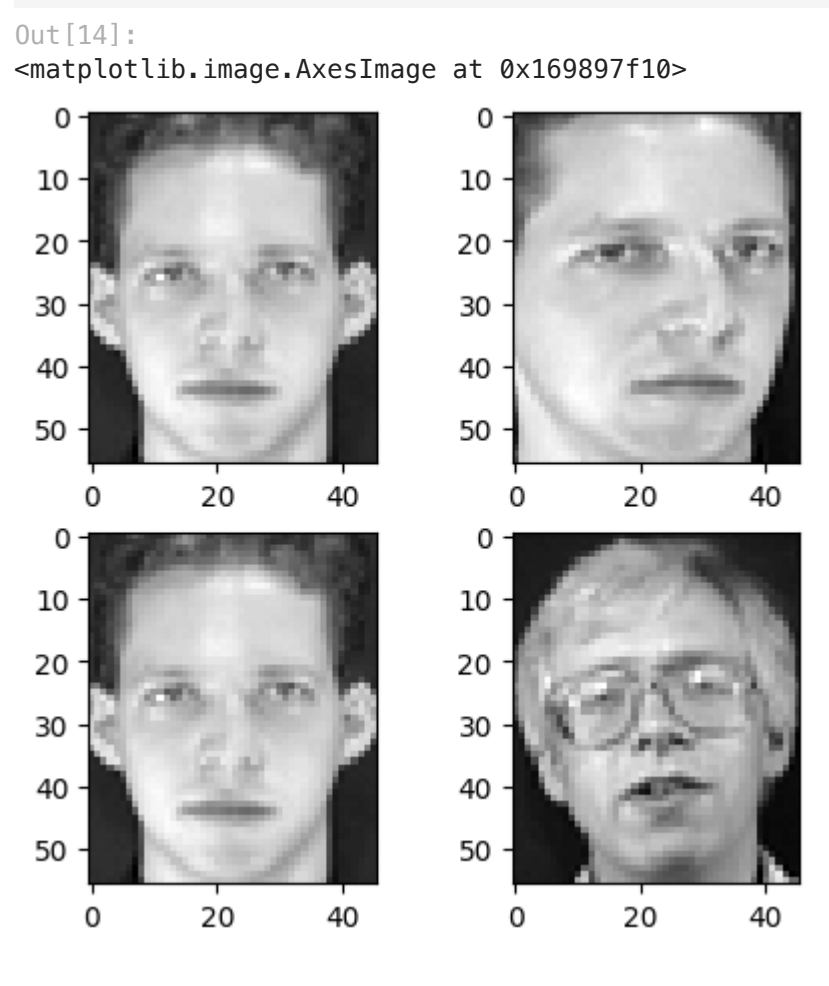
T5. What is the Euclidean distance between xf[0,0] and xf[0,1]? What is the Euclidean distance between xf[0,0] and xf[1,0]? Does the numbers make sense? Do you think these numbers will be useful for face verification?

In [13]:
```python
def L2_dist(x1, x2):
    """
    TODO: Calculate L2 distance.
    """
    return np.sqrt(((x1 - x2) ** 2).sum())

# Test L2_dist
def test_L2_dist():
    assert L2_dist(np.array([1, 2, 3]), np.array([1, 2, 3])) == 0.0
    assert  L2_dist(np.array([0, 0, 0]), np.array([1, 2, 3])) == np.sqrt(14)

test_L2_dist()

print('Euclidean distance between xf[0,0] and xf[0,1] is', L2_dist(xf[0,0], xf[0,1]))
print('Euclidean distance between xf[0,0] and xf[1,0] is', L2_dist(xf[0,0], xf[1,0]))
```

```
Euclidean distance between xf[0,0] and xf[0,1] is 10.037616294165492
Euclidean distance between xf[0,0] and xf[1,0] is 8.173295099737281
```

In [14]:
```python
# TODO: Show why does the numbers make sense
fig, axes = plt.subplots(2, 2, figsize=(5, 5))
axes[0, 0].imshow(xf[0, 0], cmap='gray')
axes[0, 1].imshow(xf[0, 1], cmap='gray')
axes[1, 0].imshow(xf[0, 0], cmap='gray')
axes[1, 1].imshow(xf[1, 0], cmap='gray')
```

Out[14]:
```
<matplotlib.image.AxesImage at 0x169897f10>
```



```
ANS : I think the numbers make sense because xf[0, 0] and xf[1, 0] is the same pose that
have the similar ratio of face and background. but xf[0, 0] and xf[0, 1] the pose is
different so there're more distance between them. But I think it's not useful for face
verification because the distance within the same person should be smaller than the
distance between different person to tell that the image shown is the same person or not.
```

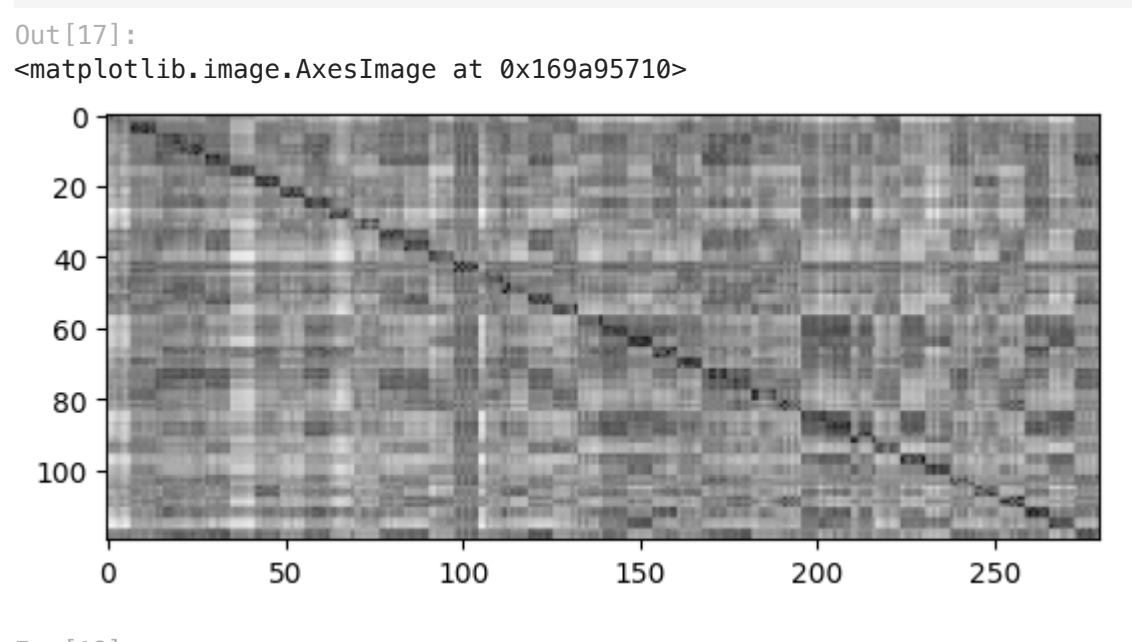
T6. Write a function that takes in a set of feature vectors T and a set of feature vectors D, and then output the similarity matrix A. Show the matrix as an image. Use the feature vectors from the first 3 images from all 40 people for list T (in order x[0, 0], x[0, 1], x[0, 2], x[1, 0], x[1, 1], ...x[39, 2]). Use the feature vectors from the remaining 7 images from all 40 people for list D (in order x[0, 3], x[0, 4], x[0, 5], x[1, 6], x[0, 7], x[0, 8], x[0, 9], x[1, 3], x[1, 4]...x[39, 9]). We will treat T as our training images and D as our testing images

In [15]:
```python
def organize_shape(matrix):
    """
    TODO (Optional): Reduce matrix dimension of 2D image to 1D and merge people and image dimension.
    This function can be useful at organizing matrix shapes.

    Example:
        Input shape: (people_index, image_index, image_shape[0], image_shape[1])
        Output shape: (people_index*image_index, image_shape[0]*image_shape[1])
    """
    return matrix.reshape(matrix.shape[0] * matrix.shape[1], matrix.shape[2] * matrix.shape[3])


def generate_similarity_matrix(A, B):
    """
    TODO: Calculate similarity matrix M,
    which M[i, j] is a distance between A[i] and B[j].
    """

    # INSERT CODE HERE
    similarity_matrix = np.array([[L2_dist(A[i], B[j]) for j in range(len(B))] for i in range(len(A))])

    return similarity_matrix

def test_generate_similarity_matrix():
    test_A = np.array([[1, 2],[3,4]])
    test_B = np.array([[1, 2], [5, 6], [7, 8]])
    expected_matrix = np.sqrt(np.array([[0, 32, 72], [8, 8, 32]]))
    assert (generate_similarity_matrix(test_A, test_B) == expected_matrix).all()

test_generate_similarity_matrix()
```

In [16]:
```python
#TODO: Show similariry matrix between T and D.

# INSERT CODE HERE
T = xf[:, :3]
D = xf[:, 3:]

similarity_matrix = generate_similarity_matrix(organize_shape(T), organize_shape(D))
```

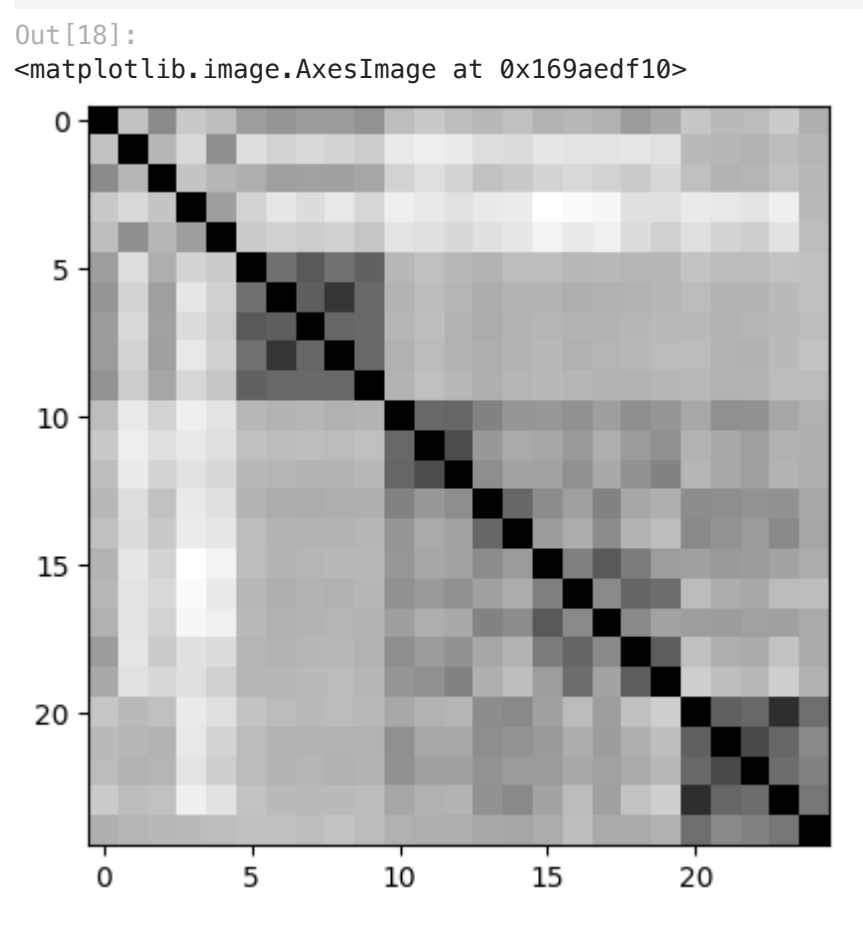
In [17]:
```python
plt.imshow(similarity_matrix, cmap="gray")
```

Out[17]:
```
<matplotlib.image.AxesImage at 0x169a95710>
```



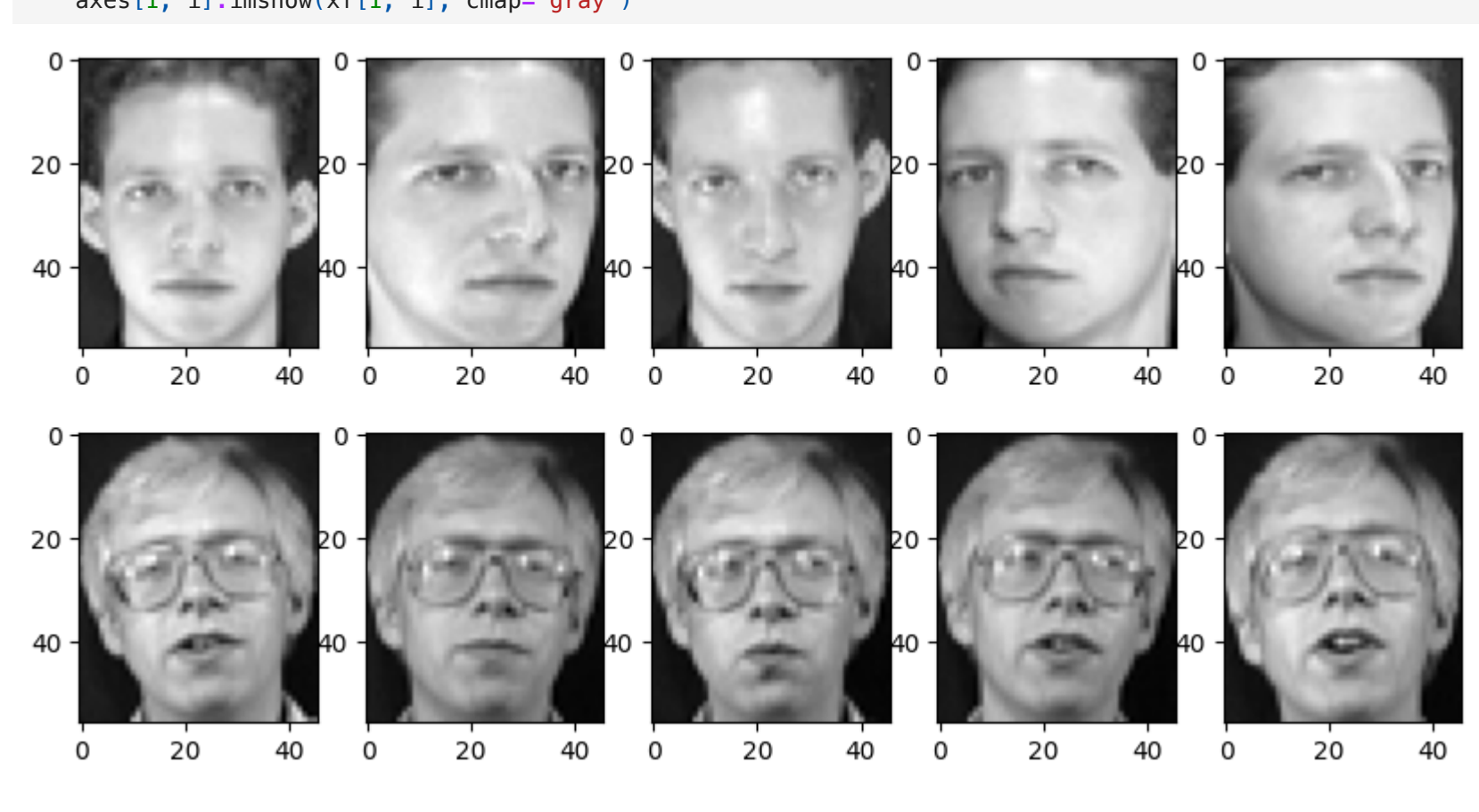
In [18]:
```python
example_similarity_matrix = generate_similarity_matrix(organize_shape(xf[:5, :5]), organize_shape(xf[:5, :5])
plt.imshow(example_similarity_matrix, cmap='gray')
```

Out[18]:
```
<matplotlib.image.AxesImage at 0x169aedf10>
```



T7. From the example similarity matrix above, what does the black square between [5:10,5:10] suggest about the pictures from person number 2? What do the patterns from person number 1 say about the images from person 1?

In [19]:
```python
# INSERT CODE HERE
fig, axes = plt.subplots(2, 5, figsize=(10, 5))
for i in range(5):
    axes[0, i].imshow(xf[0, i], cmap='gray')
    axes[1, i].imshow(xf[1, i], cmap='gray')
```



```
    ANS : It suggests that the images from person number 2 are similar to each other. The
patterns from person number 1 say that the images from person 1 are different from each
other.
```

T8. Write a function that takes in the similarity matrix created from the previous part, and a threshold t as inputs. The outputs of the function are the true positive rate and the false alarm rate of the face verification task (280 Test images, tested on 40 people, a total of 11200 testing per threshold). What is the true positive rate and the false alarm rate for t = 10?

In [64]:
```python
similarity_matrix.shape
```

Out[64]:
```
(120, 280)
```

In [66]:
```python
120/3*280
```

Out[66]:
```
11200.0
```

In [63]:
```python
40 * 7
```

Out[63]:
```
280
```

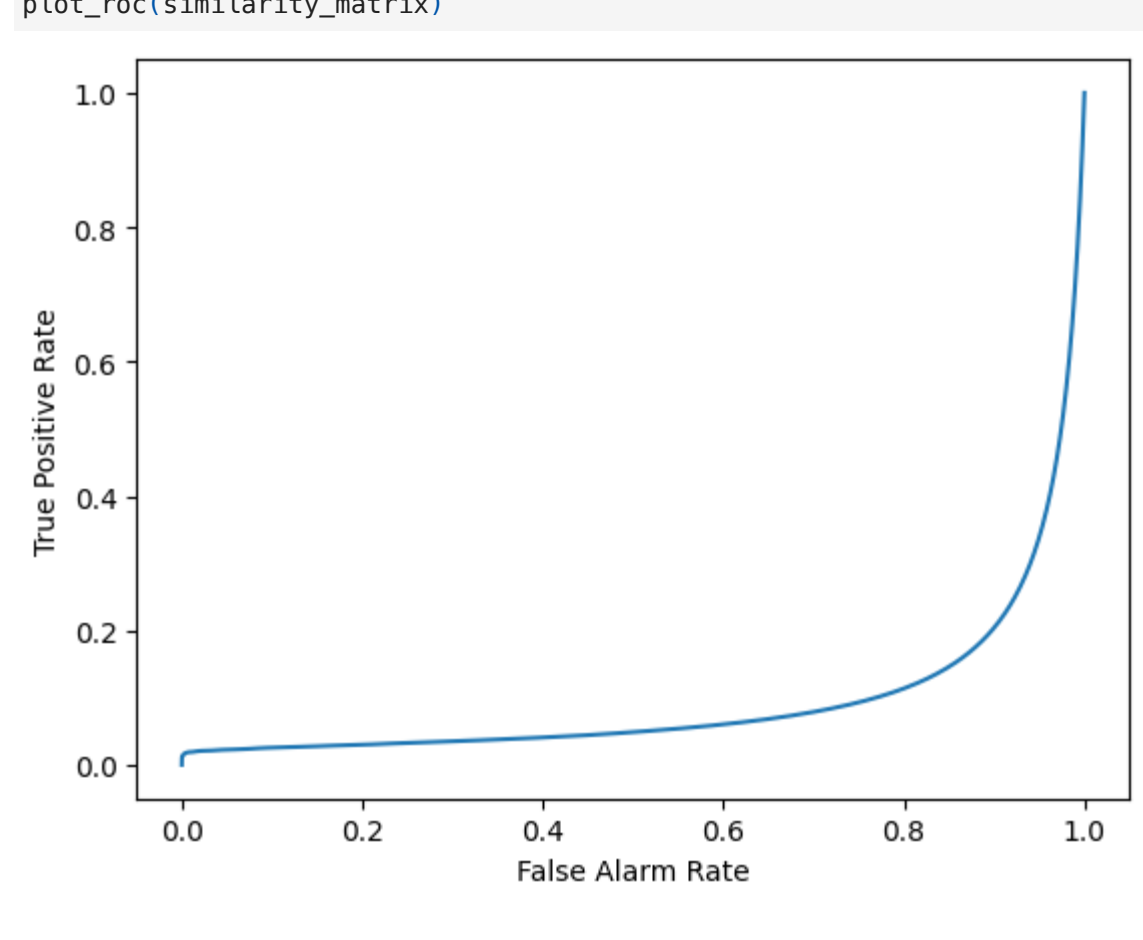
In [62]:
```python
280 * 40
```

Out[62]:
```
11200
```

In [79]:
```python
def evaluate_performance(similarity_matrix, threshold):
    """
    TODO: Calculate true positive rate and false alarm rate from given similarity_matrix and threshold.
    """

    # INSERT CODE HERE
    """
    labels = np.array([np.arange(1, 41)] * (7 * 40)).flatten()
    labels = np.arange(1, 41).repeat(7*40)
    print(labels.shape)
    print(labels)
    """

    labels = []
    predictions = []
    for i in range(0, similarity_matrix.shape[0], 3):
        for j in range(similarity_matrix.shape[1]):
            # labels.append(i // 3 + 1)
            # predictions.append(i // 3 + 1)
            labels.append(j // 7 + 1)
            # print(similarity_matrix[i : i + 3, j])
            # print(similarity_matrix[i : i + 3, j].min())
            predictions.append(np.where(similarity_matrix[i:i+3, j].min() < threshold, i // 3+1, 0))

    labels = np.array(labels)
    # print(labels.shape)
    # print(labels)
    predictions = np.array(predictions)

    # import pandas as pd
    # print(pd.DataFrame(predictions).value_counts())
    # print(predictions.shape)
    # print(predictions)
    true_pos = np.sum((predictions == labels) & (predictions != 0))
    false_pos = np.sum((predictions != labels) & (predictions != 0))
    true_neg = np.sum((predictions == 0) & (predictions != labels))
    false_neg = np.sum((predictions == 0) & (labels != 0))
    # print(true_pos, false_pos, true_neg, false_neg)
    true_pos_rate = true_pos / (true_pos + false_neg)
    false_pos_rate = false_pos / (false_pos + true_neg)
    return true_pos_rate, false_pos_rate

# Quick check
# (true_pos_rate, false_neg_rate) should be (0.9928571428571429, 0.33507326007326005)
evaluate_performance(similarity_matrix, 9.5)
```
Out[79]:
```
(0.036865137249701634, 0.3350119025819447)
```
In [80]:
```python
# INSERT CODE HERE
evaluate_performance(similarity_matrix, 10)
```
Out[80]:
```
(0.044884169884169885, 0.4563684644263346)
```

ANS: (0.044884169884169885, 0.4563684644263346)

T9. Plot the RoC curve for this simple verification system. What should be the minimum threshold to generate the RoC curve? What should be the maximum threshold? Your RoC should be generated from at least 1000 threshold levels equally spaced between the minimum and the maximum. (You should write a function for this).

In [81]:
```python
def calculate_roc(input_mat):
    """
    TODO: Calculate a list of true_pos_rate and a list of false_neg_rate from the given matrix.
    """

    # INSERT CODE HERE
    thresholds = np.linspace(input_mat.min(), input_mat.max(), 1000)

    tpr_list, far_list = [], []

    for threshold in thresholds:
        tpr, far = evaluate_performance(similarity_matrix, threshold)
        tpr_list.append(tpr)
        far_list.append(far)

    return tpr_list, far_list

def plot_roc(input_mat):
    """
    TODO: Plot RoC Curve from a given matrix.
    """
    # INSERT CODE HERE
    plt.plot(calculate_roc(input_mat)[1], calculate_roc(input_mat)[0])
    plt.xlabel("False Alarm Rate")
    plt.ylabel("True Positive Rate")
```
In [82]:
```python
# INSERT CODE HERE
plot_roc(similarity_matrix)
```


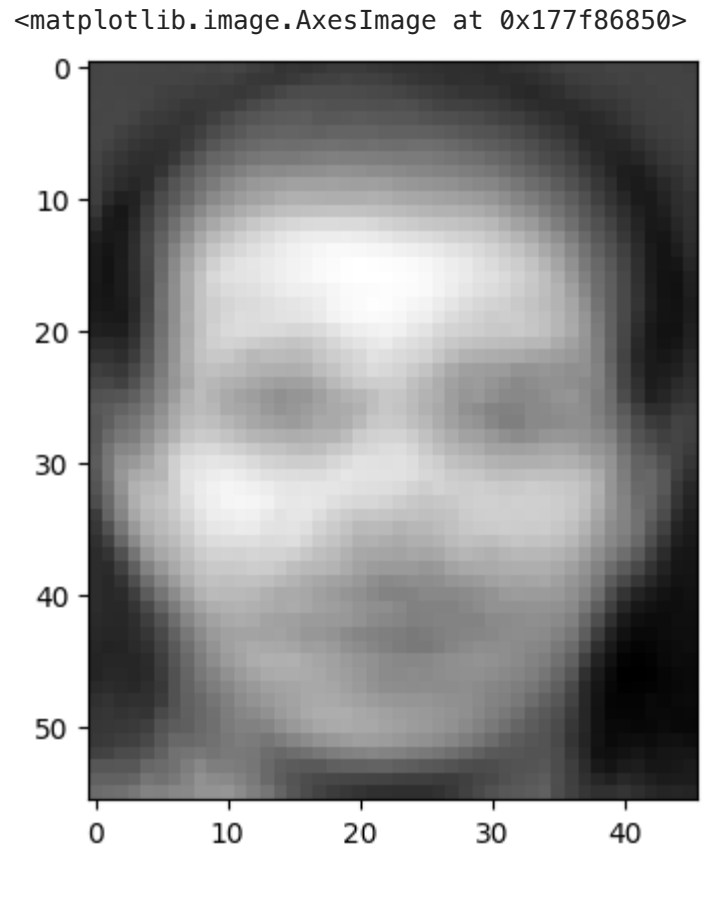
ANS: The minimum threshold should be smallest value of the similarity matrix. The maximum threshold should be the largest value of the similarity matrix.

T10. What is the EER (Equal Error Rate)? What is the recall rate at 0.1% false alarm rate? (Write this in the same function as the previous question)

In [83]:
```python
def evaluate_eer(similarity_matrix, threshold):
    pass

evaluate_eer(similarity_matrix, 9.5)

# EER should be either 0.9071428571428571 or 0.9103759398496248 depending on method.
# Recall rate at 0.1% false alarm rate should be 0.5428571428571428.
```

ANS: EER is where false alarm rate and false negative rate are equal.

T11. Compute the mean vector from the training images. Show the vector as an image (use numpy.reshape()). This is typically called the meanface (or meanvoice for speech signals). You answer should look exactly like the image shown below.

In [103]:
```python
# INSERT CODE HERE
meanface = np.mean(organize_shape(T), axis=0).reshape(56, 46)
plt.imshow(meanface, cmap="gray")
```
Out[103]:
```
<matplotlib.image.AxesImage at 0x177f86850>
```


T12. What is the size of the covariance matrix? What is the rank of the covariance matrix?

In [157]:
```python
# TODO: Find the size and the rank of the covariance matrix.
X = T.reshape(120, -1).T
X_hat = X - meanface.reshape(-1, 1)

cov_matrix = np.cov(organize_shape(T).T)
print(cov_matrix.shape)
print(np.linalg.matrix_rank(cov_matrix))
```
```
(2576, 2576)
119
```
In [ ]:

ANS: (2576, 2576), 119

T13. What is the size of the Gram matrix? What is the rank of Gram matrix? If we compute the eigenvalues from the Gram matrix, how many non- zero eigenvalues do we expect to get?

In [107]:
```python
# TODO: Compute gram matrix.
gram_matrix = np.matmul(organize_shape(T), organize_shape(T).T)
```
In [108]:
```python
# TODO: Show size and rank of Gram matrix.
print(gram_matrix.shape)
print(np.linalg.matrix_rank(gram_matrix))
```
```
(120, 120)
120
```

ANS: (120, 120), 120

T14. Is the Gram matrix also symmetric? Why?

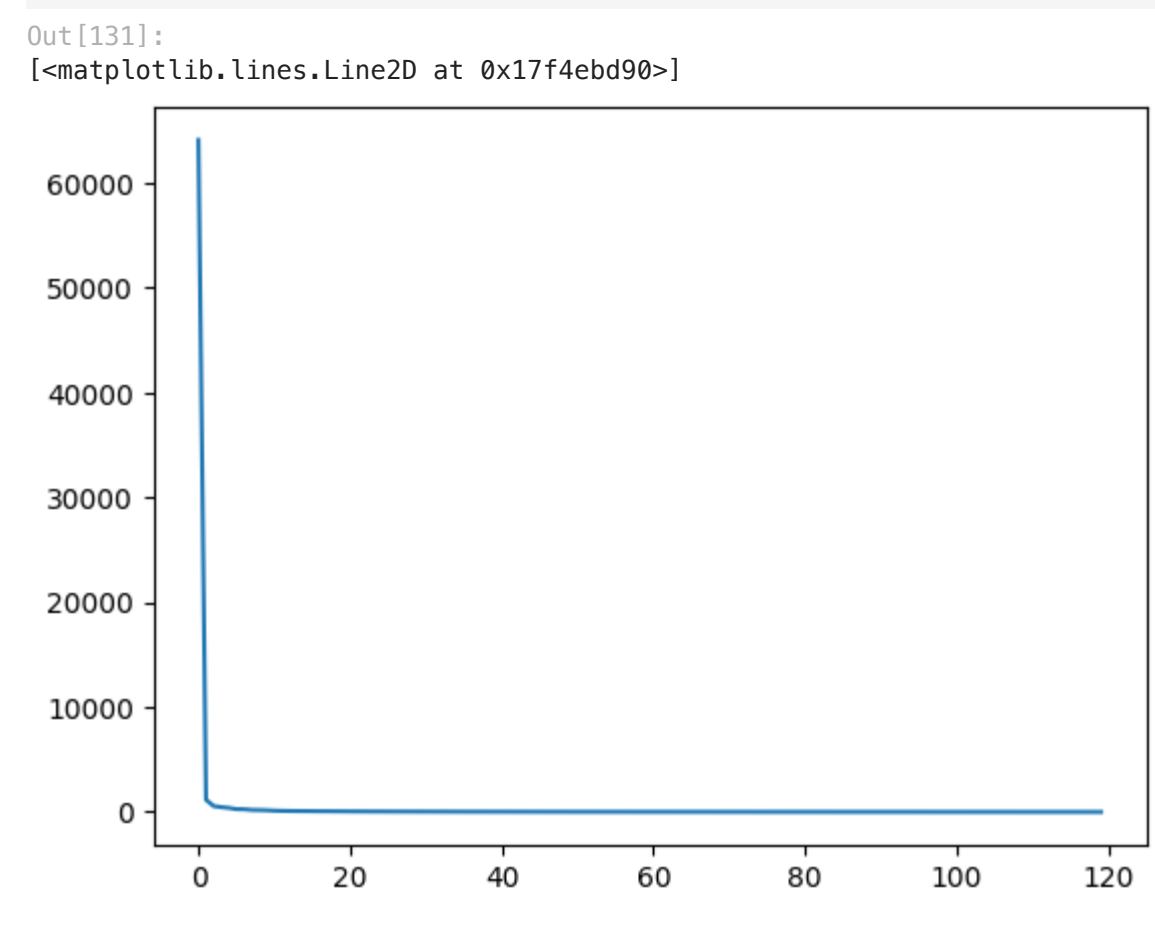
ANS: Yes, because the input is symmetric, so inner product of the input is also symmetric.

T15. Compute the eigenvectors and eigenvalues of the Gram matrix, v 0 and λ. Sort the eigenvalues and eigenvectors in descending order so that the first eigenvalue is the highest, and the first eigenvector corresponds to the best direction. How many non-zero eigenvalues are there? If you see a very small value, it is just numerical error and should be treated as zero.

In [129]:
```python
# Hint: https://numpy.org/doc/stable/reference/generated/numpy.linalg.eigh.html

def calculate_eigenvectors_and_eigenvalues(matrix):
    """
    TODO: Calculate eigenvectors and eigenvalues,
    then sort the eigenvalues and eigenvectors in descending order.

    Hint: https://numpy.org/doc/stable/reference/generated/numpy.linalg.eigh.html
    """

    # INSERT CODE HERE
    eigenvalues, eigenvectors = np.linalg.eigh(matrix)

    # sort eigenvalues and eigenvectors in descending order
    idx = eigenvalues.argsort()[::-1]
    eigenvalues = eigenvalues[idx]
    eigenvectors = eigenvectors[:, idx]

    return eigenvalues, eigenvectors

eigenvalues, eigenvectors = calculate_eigenvectors_and_eigenvalues(gram_matrix)

def test_eigenvalues_eigenvectors():
    # Dot product of an eigenvector pair should equal to zero.
    assert np.round(eigenvectors[10].dot(eigenvectors[20]), 10) == 0.0

    # Check if eigenvalues are sorted.
    assert list(eigenvalues) == sorted(eigenvalues, reverse = True)

test_eigenvalues_eigenvectors()
```
In [130]:
```python
np.count_nonzero(eigenvalues)
```
Out[130]:
```
120
```

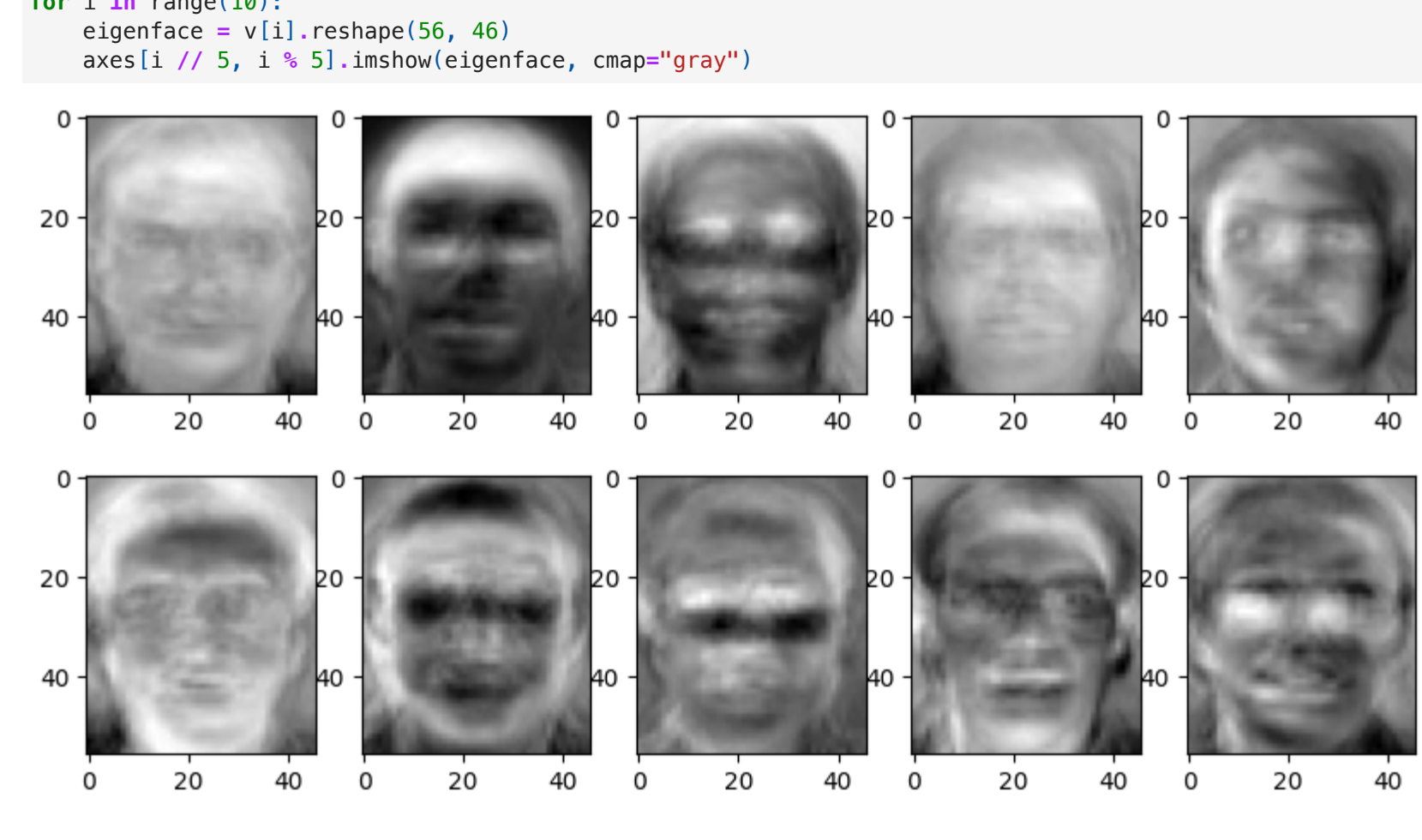
ANS: 120

T16. Plot the eigenvalues. Observe how fast the eigenvalues decrease. In class, we learned that the eigenvalues is the size of the variance for each eigenvector direction. If I want to keep 95% of the variance in the data, how many eigenvectors should I use?

In [131]:
```python
# INSERT CODE HERE
plt.plot(eigenvalues)
```
Out[131]:
```
[<matplotlib.lines.Line2D at 0x17f4ebd90>]
```


ANS: 40

T17. Compute \vec{v} . Don't forget to renormalize so that the norm of each vector is 1 (you can use numpy.linalg.norm). Show the first 10 eigenvectors as images. Two example eigenvectors are shown below. We call these images eigenfaces (or eigenvoice for speech signals).

In [153]:
```python
# TODO: Compute v, then renormalize it.

# INSERT CODE HERE

v = np.matmul(organize_shape(T).T, eigenvectors[:, 0])
v = v / np.linalg.norm(v)
```
In [154]:
```python
def test_eignevector_cov_norm(v):
    assert (np.round(np.linalg.norm(v, axis=0), 1) == 1.0).all()

test_eignevector_cov_norm(v)
```
In [155]:
```python
# TODO: Show the first 10 eigenvectors as images.

v = []
for evec in eigenvectors.T:
    v.append(np.dot(X_hat, evec))

v = np.array(v) / np.linalg.norm(v, axis=1).reshape(-1, 1)

fig, axes = plt.subplots(2, 5, figsize=(10, 5))
for i in range(10):
    eigenface = v[i].reshape(56, 46)
    axes[i // 5, i % 5].imshow(eigenface, cmap="gray")
```


T18. From the image, what do you think the first eigenvector captures? What about the second eigenvector? Look at the original images, do you think biggest variance are capture in these two eigenvectors?

ANS: I think the first eigenvector captures the brightness of the image. The second eigenvector captures the contrast between the face and the hair. I don't think the biggest variance are captured in these two eigenvectors.

T19. Find the projection values of all images. Keep the first k = 10 projection values. Repeat the simple face verification system we did earlier using these projected values. What is the EER and the recall rate at 0.1% FAR?

In [162]:
```python
def calculate_projection_vectors(matrix, meanface, v):
    """
    TODO: Find the projection vectors on v from given matrix and meanface.
    """

    # INSERT CODE HERE
    X = matrix.reshape(120, -1).T
    X_hat = X - meanface.reshape(-1, 1)
    projection_vectors = np.dot(X_hat.T, v.T)

    return projection_vectors
```
In [163]:
```python
# TODO: Get projection vectors of T and D, then Keep first k projection values.
k = 10
T_reduced = calculate_projection_vectors(T, meanface, v)[:,:k]
D_reduced = calculate_projection_vectors(D, meanface, v)[:,:k]


def test_reduce_dimension():
    assert T_reduced.shape[-1] == k
    assert D_reduced.shape[-1] == k

test_reduce_dimension()
```
```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[163], line 4
      2 k = 10
      3 T_reduced = calculate_projection_vectors(T, meanface, v)[:,:k]
----> 4 D_reduced = calculate_projection_vectors(D, meanface, v)[:,:k]
      7 def test_reduce_dimension():
      8     assert T_reduced.shape[-1] == k

Cell In[162], line 7, in calculate_projection_vectors(matrix, meanface, v)
      2 """
      3 TODO: Find the projection vectors on v from given matrix and meanface.
      4 """
      6 # INSERT CODE HERE
----> 7 X = matrix.reshape(120, -1).T
      8 X_hat = X - meanface.reshape(-1, 1)
      9 projection_vectors = np.dot(X_hat.T, v.T)

ValueError: cannot reshape array of size 721280 into shape (120,newaxis)
```
In [ ]:
```python
# TODO: Get similarity matrix of T_reduced and D_reduced
```
In [ ]:
```python
# TODO: Find EER and the recall rate at 0.1% FAR.
```

ANS:

T20. What is the k that gives the best EER? Try k = 5, 6, 7, 8, 9, 10, 11, 12, 13, 14.

In [ ]:
```python
# INSERT CODE HERE
```

ANS:

T21. In order to assure that S_W is invertible we need to make sure that S_W is full rank. How many PCA dimensions do we need to keep in order for S_W to be full rank? (Hint: How many dimensions does S_W have? In order to be of full rank, you need to have the same number of linearly independent factors)

ANS:

In [ ]:
```python
# TODO: Define dimension of PCA.
n_dim = ...

# TODO: Find PCA of T and D with n_dim dimension.
```

T22. Using the answer to the previous question, project the original in- put to the PCA subspace. Find the LDA projections. To find the inverse, use −1 numpy.linalg.inv. Is S_W S_B symmetric? Can we still use numpy.linalg.eigh? How many non-zero eigenvalues are there?

In [ ]:
```python
# TODO: Find the LDA projection.
```
In [ ]:
```python
# TODO: Find how many non-zero eigenvalues there are.
```

ANS:

T23. Plot the first 10 LDA eigenvectors as images (the 10 best projections). Note that in this setup, you need to convert back to the original image space by using the PCA projection. The LDA eigenvectors can be considered as a linear combination of eigenfaces. Compare the LDA projections with the PCA projections.

In [ ]:
```python
# INSERT CODE HERE
```

T24. The combined PCA+LDA projection procedure is called fisherface. Calculate the fisherfaces projection of all images. Do the simple face verification experiment using fisherfaces. What is the EER and recall rate at 0.1% FAR?

In [ ]:
```python
# INSERT CODE HERE
```

ANS:

T25.Plot the RoC of all three experiments (No projection, PCA, andFisher) on the same axes. Compare and contrast the three results. Submit yourwriteup and code on MyCourseVille.

In [ ]:
```python
# INSERT CODE HERE
```

ANS: