1

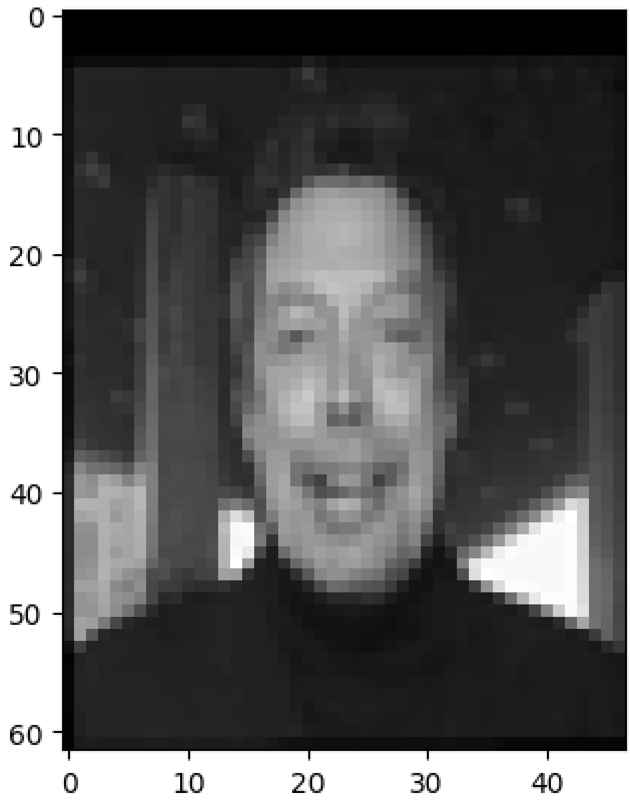
In [ ]:
```
from IPython.display import Latex
```

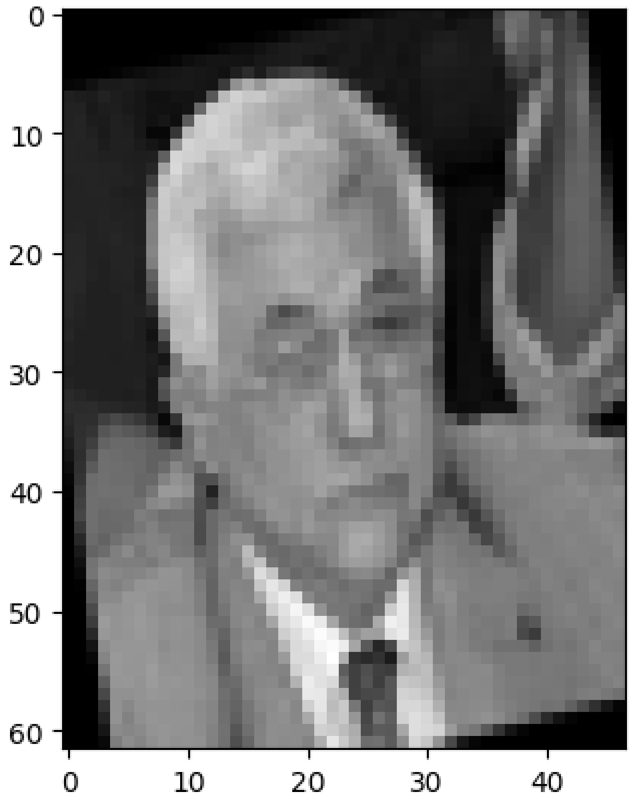
In [ ]:
```
#Downloading dataset
from sklearn import datasets
dataset = datasets.fetch_lfw_people()
X = dataset['data']
#check data
print(X.data.shape)
```

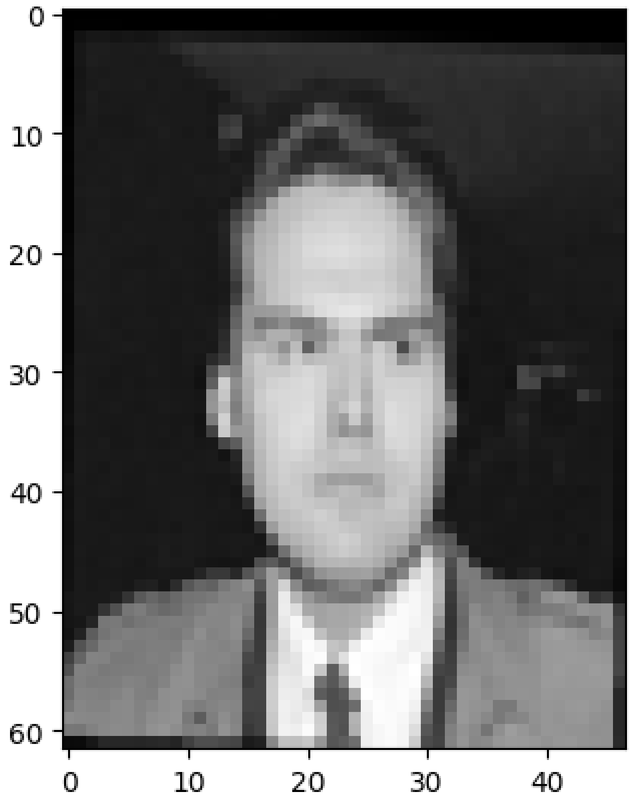
```
(13233, 2914)
```

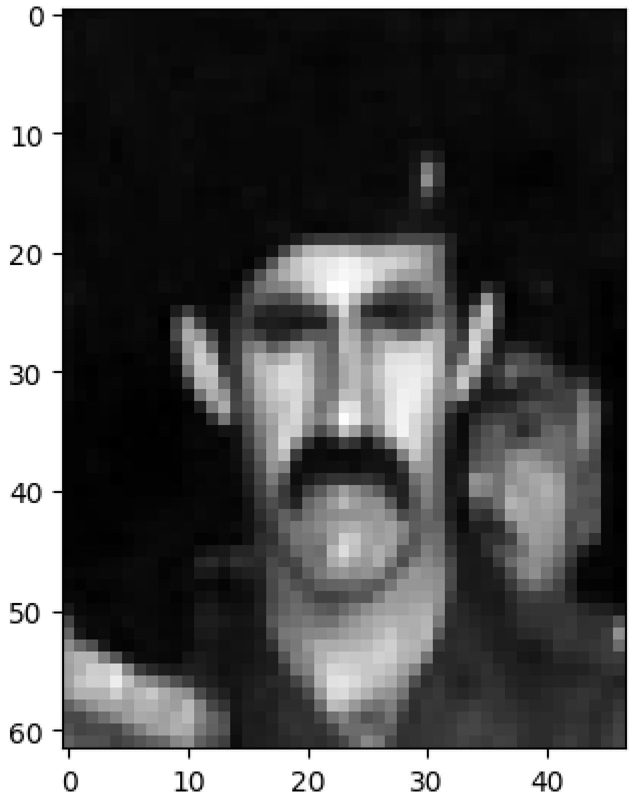
In [ ]:
```
#Relevant imports
import matplotlib
import numpy as np
from matplotlib import pyplot
from numpy import linalg
```

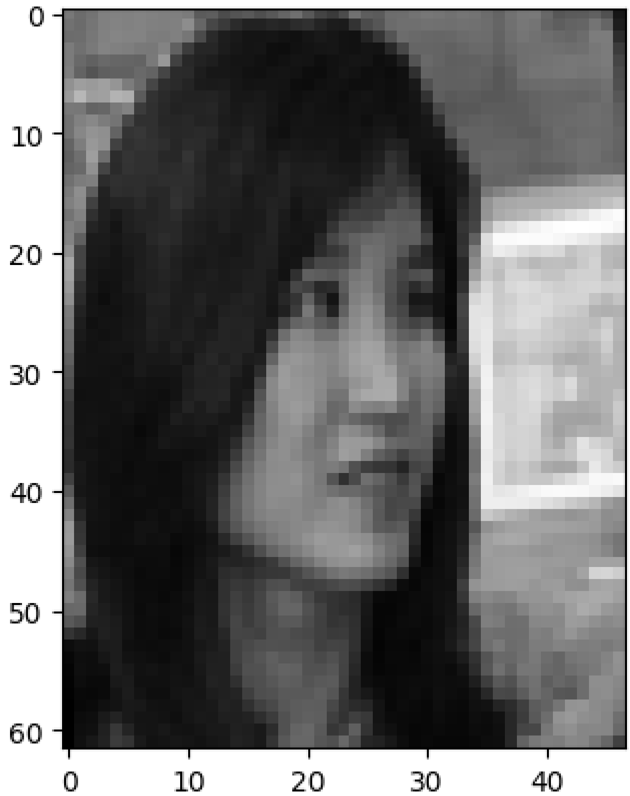
In [ ]:
```
#1.a
for i in range(20):
    matplotlib.pyplot.imshow(X[i].reshape((62,47)), cmap=matplotlib.pyplot.cm.gray
    matplotlib.pyplot.savefig(f'files/1_a={i}.png')
    matplotlib.pyplot.show()
```

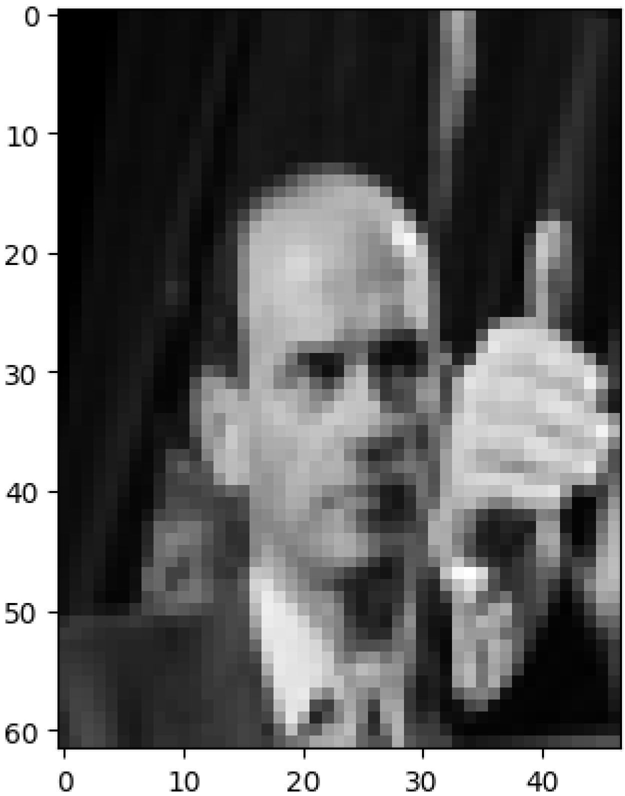


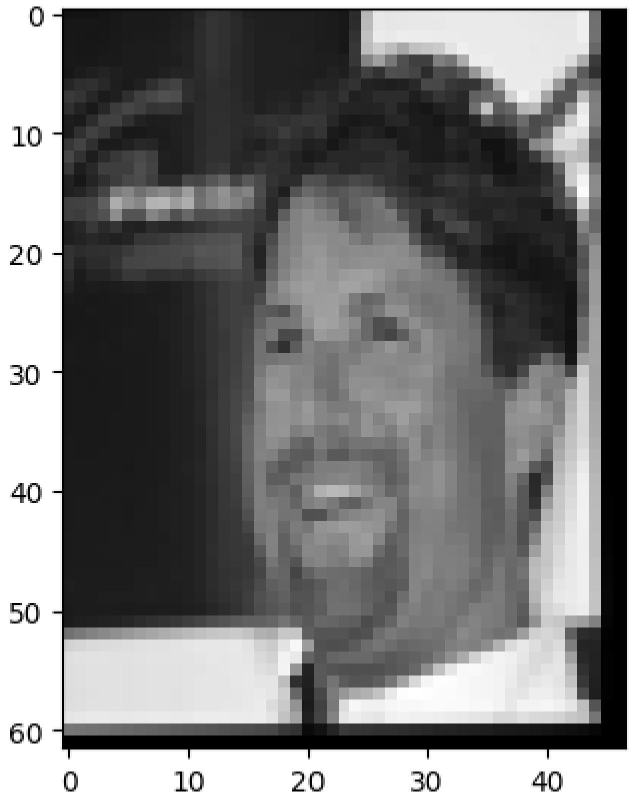

0
10
20
30
40
50
60
0
10
20
30
40

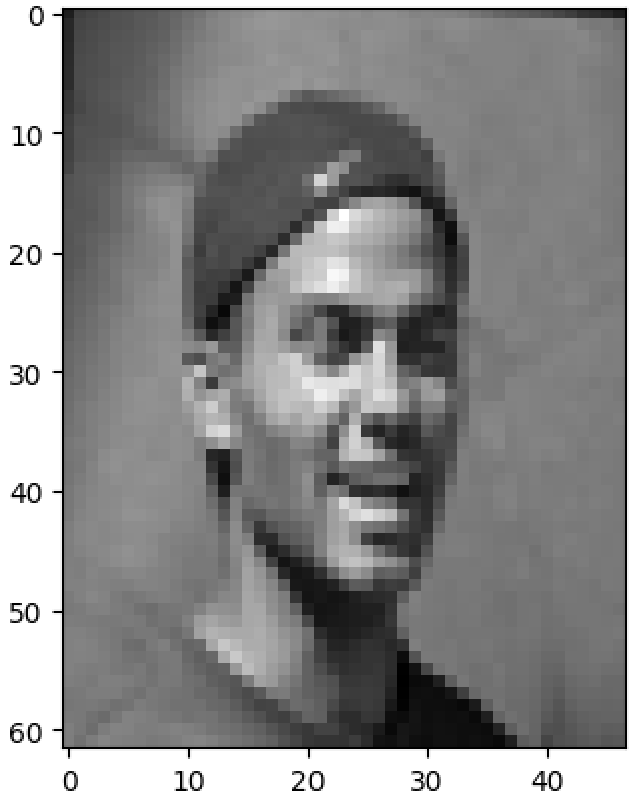

0
10
20
30
40
50
60
0
10
20
30
40

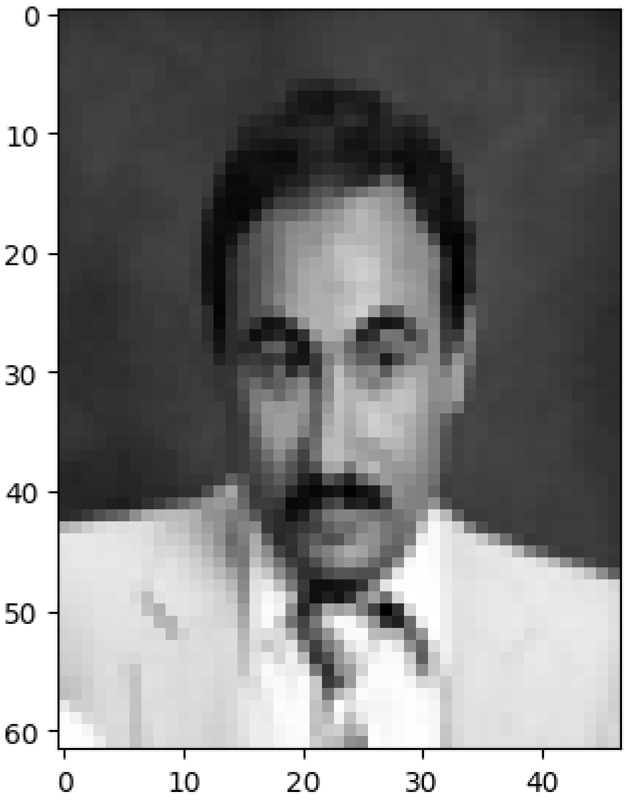

0
10
20
30
40
50
60
0
10
20
30
40

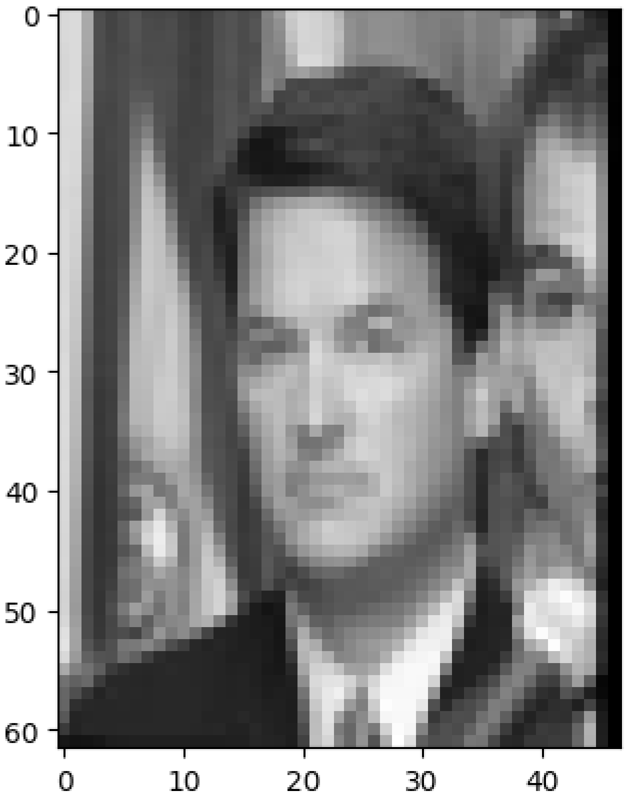

0
10
20
30
40
50
60
0
10
20
30
40

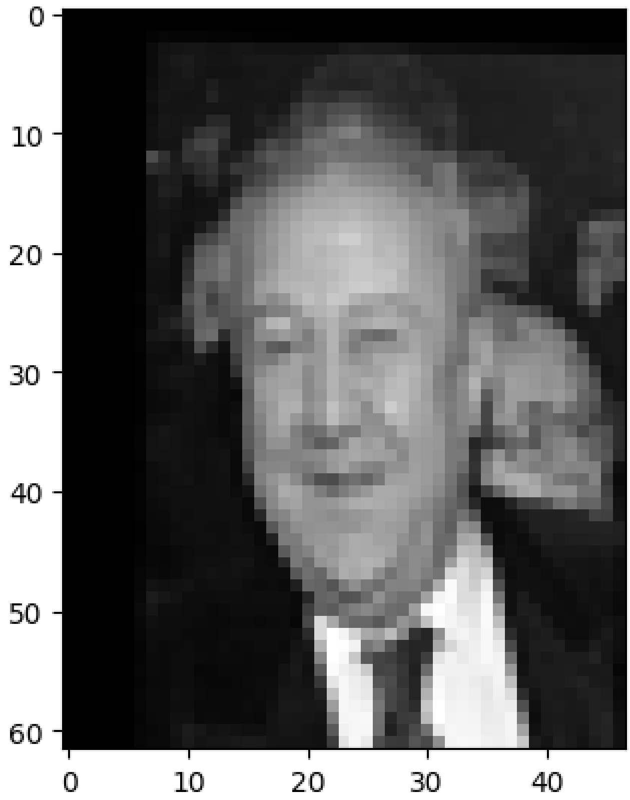


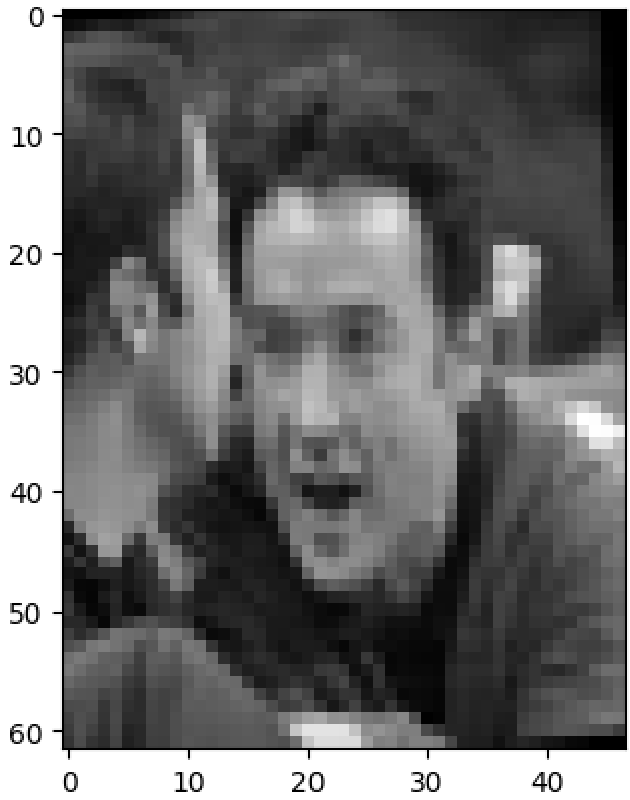

0
10
20
30
40
50
60
0
10
20
30
40

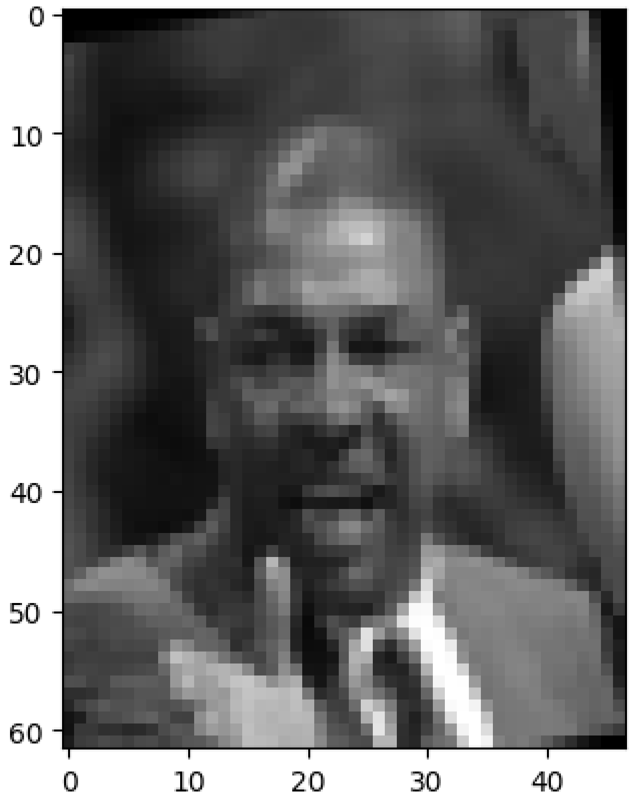

0
10
20
30
40
50
60
0
10
20
30
40

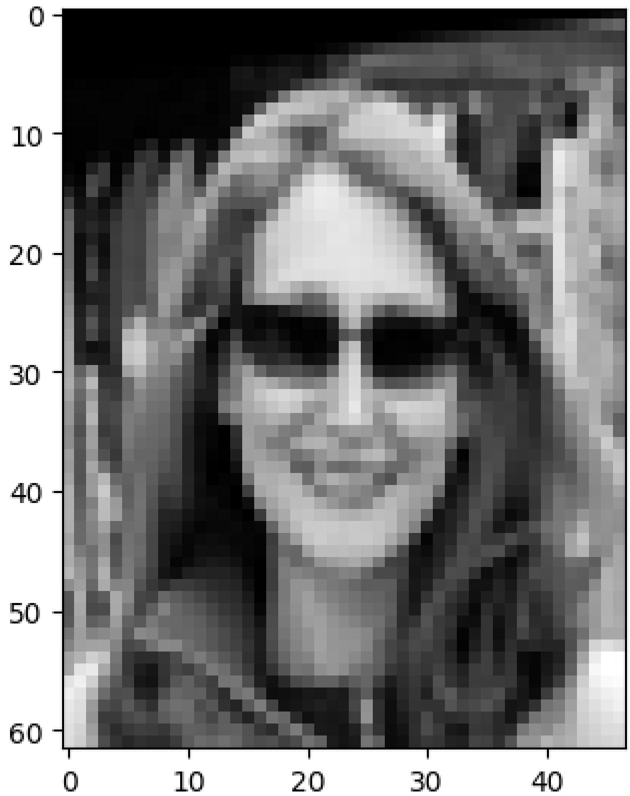


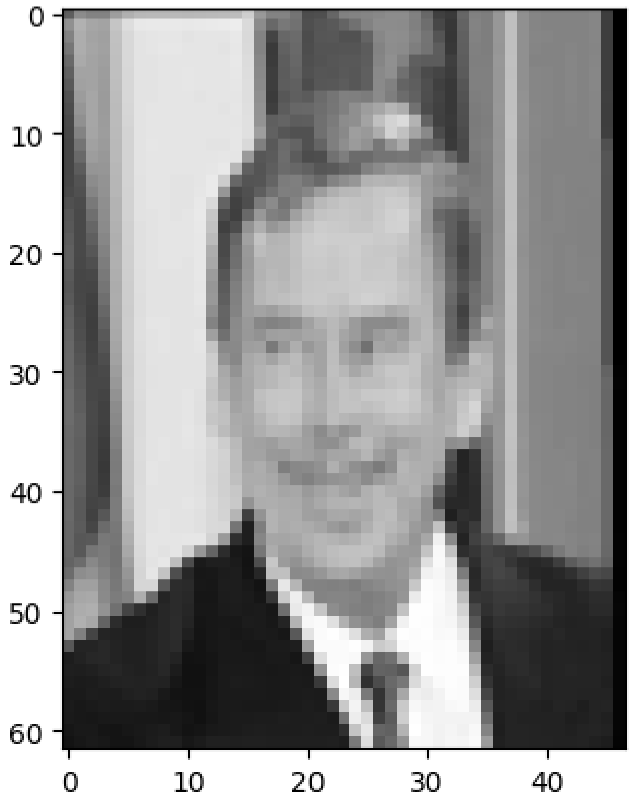



0
10
20
30
40
50
60
0
10
20
30
40

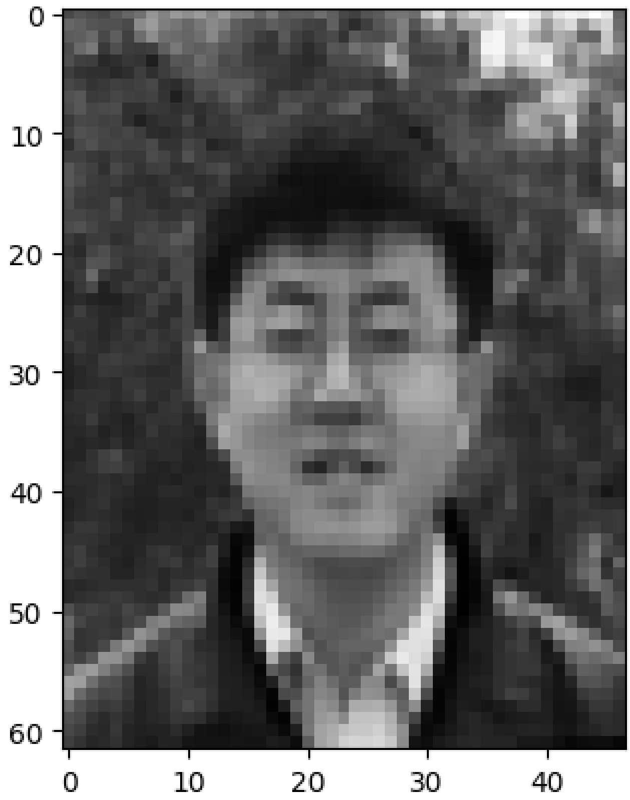

0
10
20
30
40
50
60
0
10
20
30
40

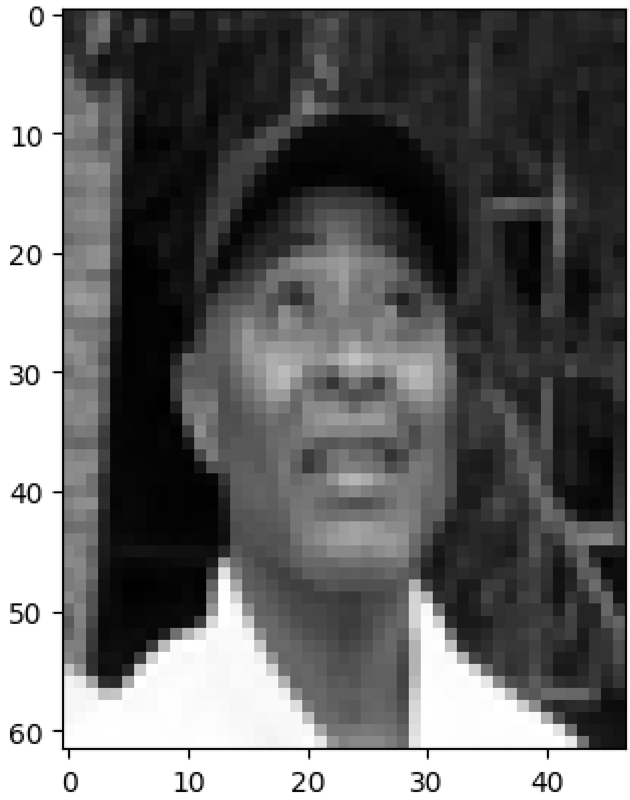

0
10
20
30
40
50
60
0 10 20 30 40

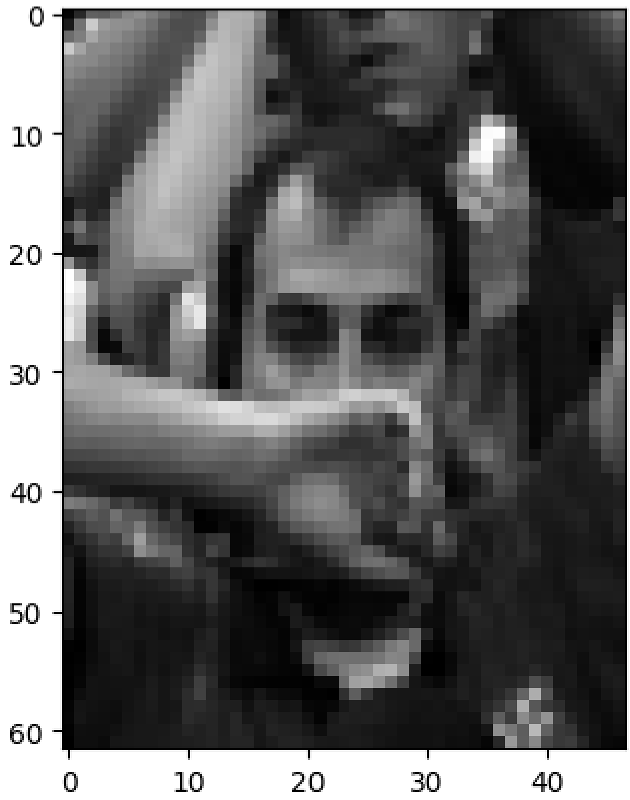

0
10
20
30
40
50
60
0 10 20 30 40


0
10
20
30
40
50
60
0
10
20
30
40


0
10
20
30
40
50
60
0
10
20
30
40


0
10
20
30
40
50
60
0
10
20
30
40


0
10
20
30
40
50
60
0
10
20
30
40

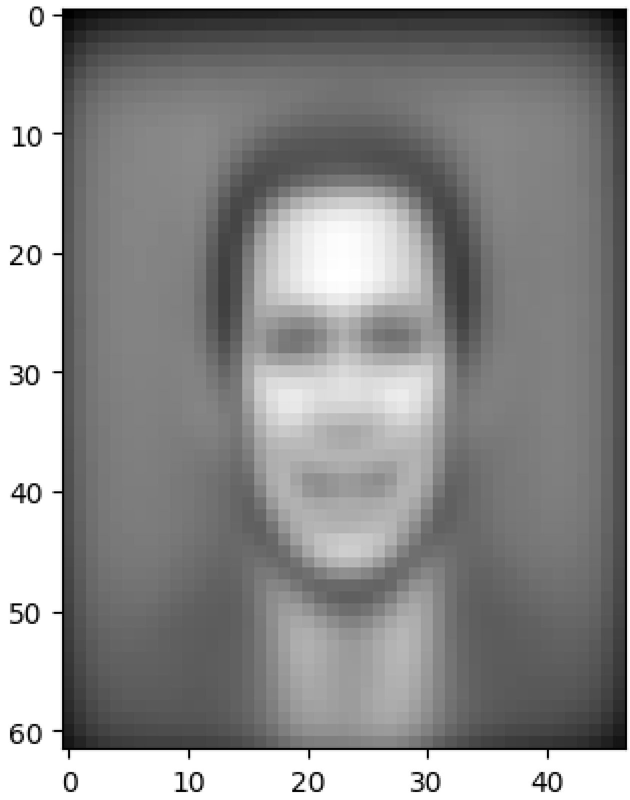


In [ ]:
```
display(Latex(r"\newpage"))
```

\newpage

In [ ]:
```
#1.b
average = X.mean(axis = 0)
print(average)
for i in range(X[0].size):
    X[i]-=average
matplotlib.pyplot.imshow(average.reshape((62,47)), cmap=matplotlib.pyplot.cm.gray )
matplotlib.pyplot.savefig(f'files/1_b=average.png')
matplotlib.pyplot.show()
```

```
[0.13771963 0.15683803 0.17031944 ... 0.2221487  0.20515166 0.18089974]
```

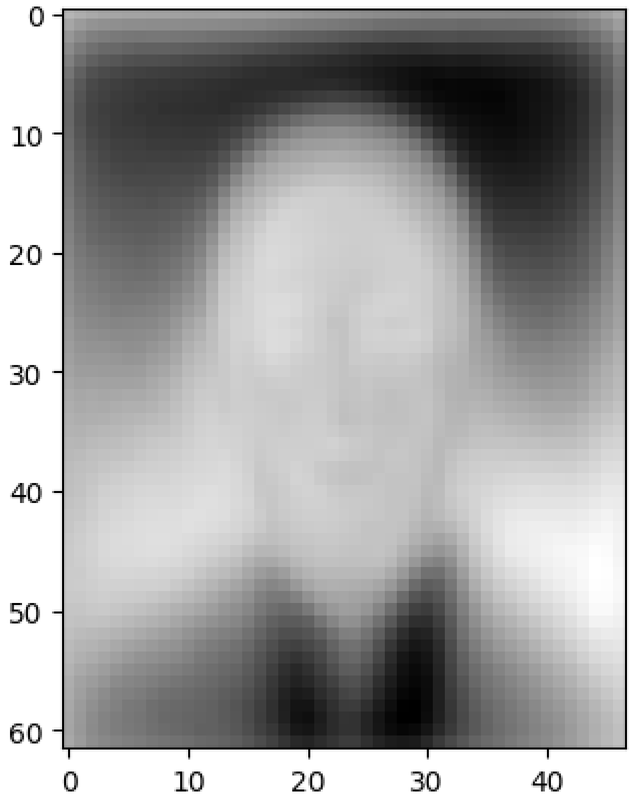


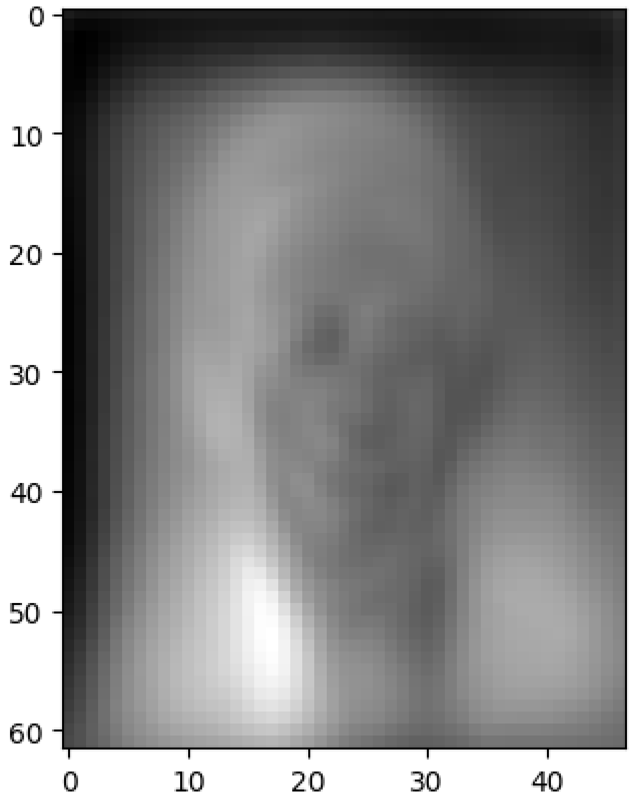
In [ ]:
```
display(Latex(r"\newpage"))
```

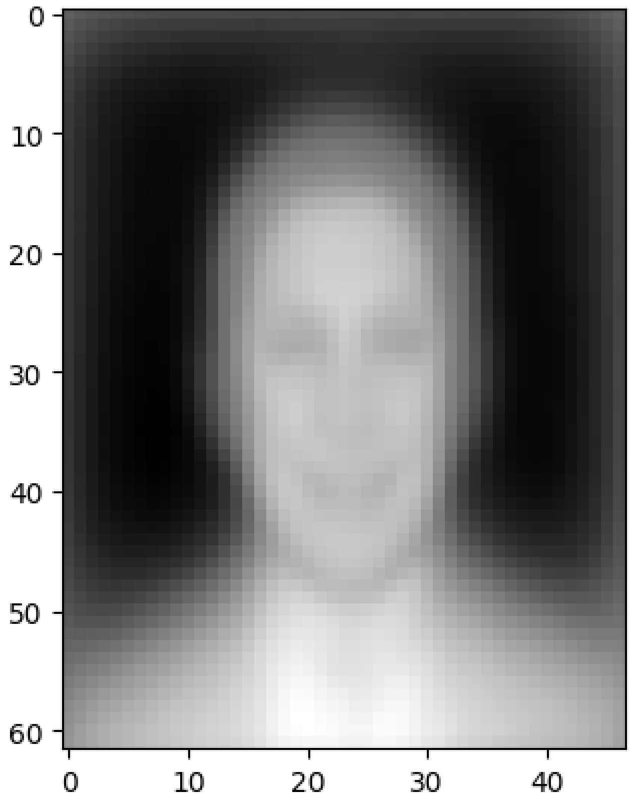
\newpage

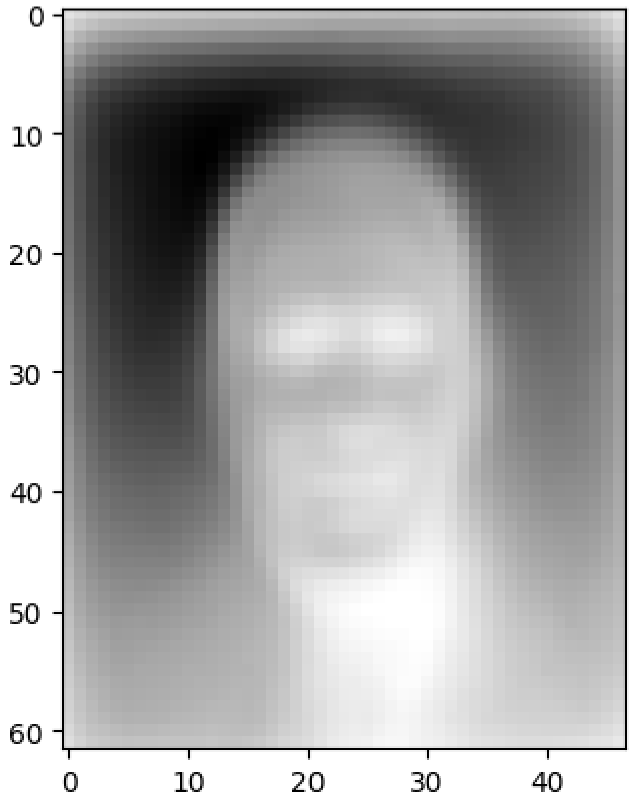
In [ ]:
```
#1.c
eig_vectors, eig_values,_  = np.linalg.svd(X.T@X)
```

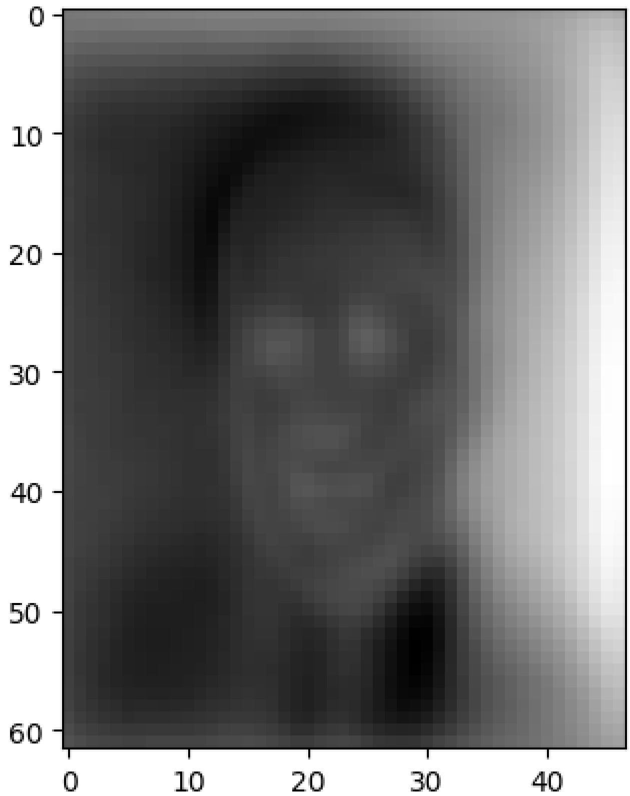
In [ ]:
```
#1.c

def projectTopKEigenVectors(X, k, eig_values, eig_vectors):
    idx = eig_values.argsort()[::-1]
    eig_values = eig_values[idx]
    eig_vectors = eig_vectors[:,idx]

    k_eig_vectors = np.zeros(eig_vectors.shape)
    k_eig_vectors[:,:k] = eig_vectors[:,:k]

    X_k = X @ k_eig_vectors
    X_recon = X_k @ k_eig_vectors.transpose()

    for i in range(20):
        matplotlib.pyplot.imshow(X_recon[i].reshape((62,47)), cmap=matplotlib.pyplo
        matplotlib.pyplot.savefig(f'files/1_c_top20_nr.{i+1}_k={k}.png')
        matplotlib.pyplot.show()

projectTopKEigenVectors(X, 10, eig_values, eig_vectors)
projectTopKEigenVectors(X, 100, eig_values, eig_vectors)
projectTopKEigenVectors(X, 1000, eig_values, eig_vectors)
```

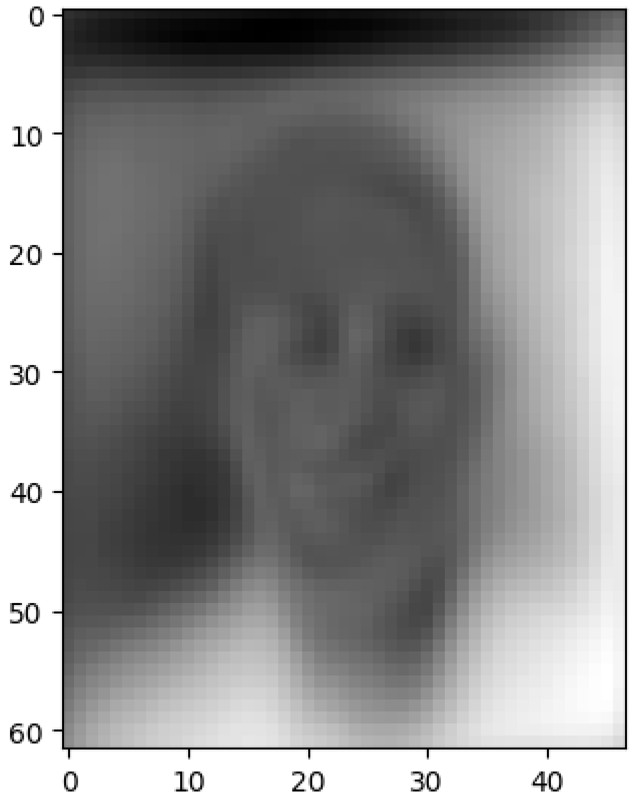

0
10
20
30
40
50
60
0
10
20
30
40

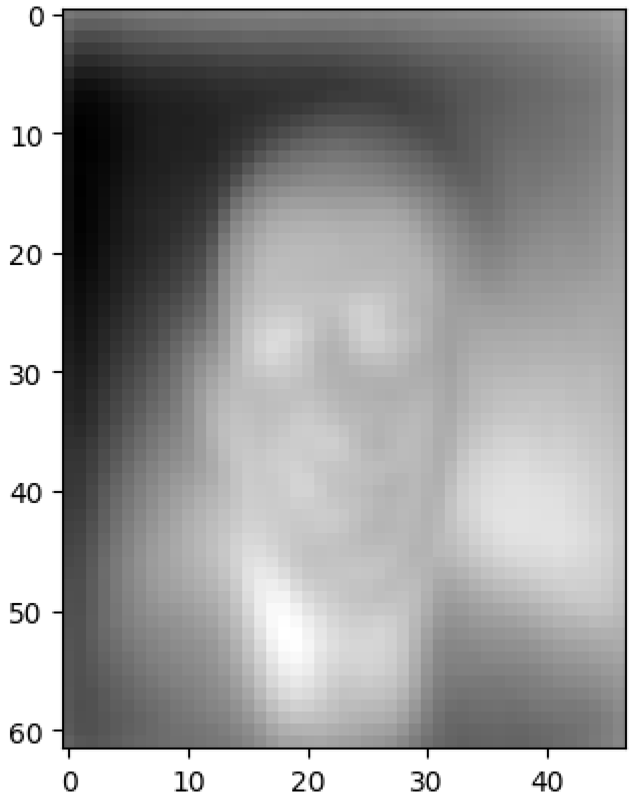

0
10
20
30
40
50
60
0
10
20
30
40

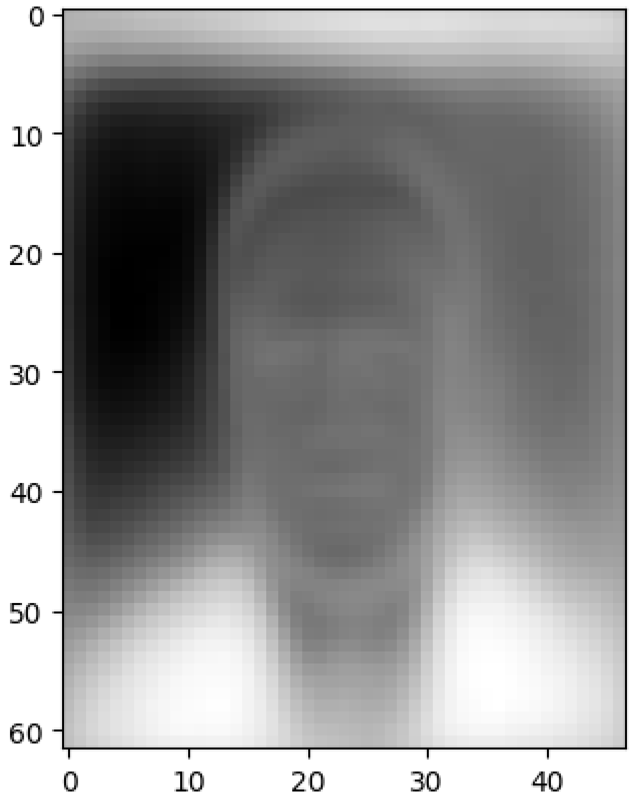

0
10
20
30
40
50
60
0
10
20
30
40

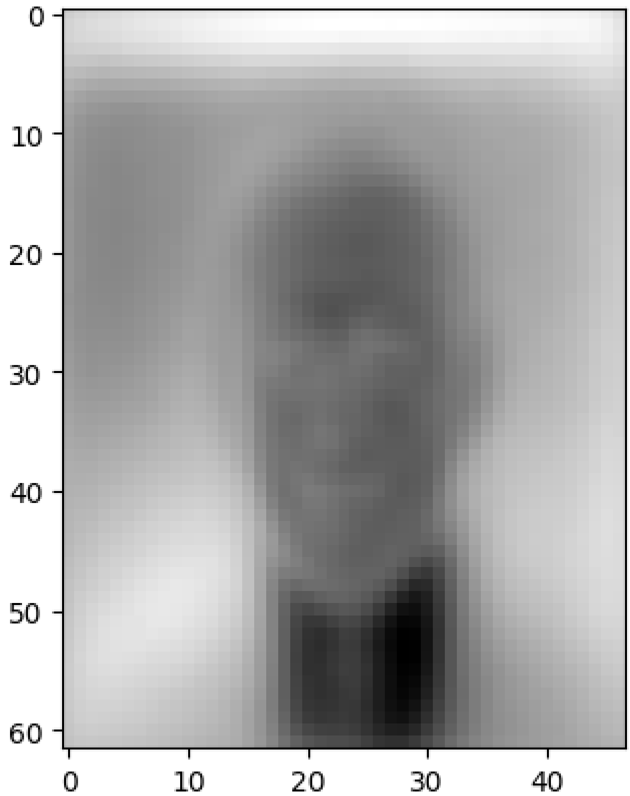

0
10
20
30
40
50
60
0
10
20
30
40

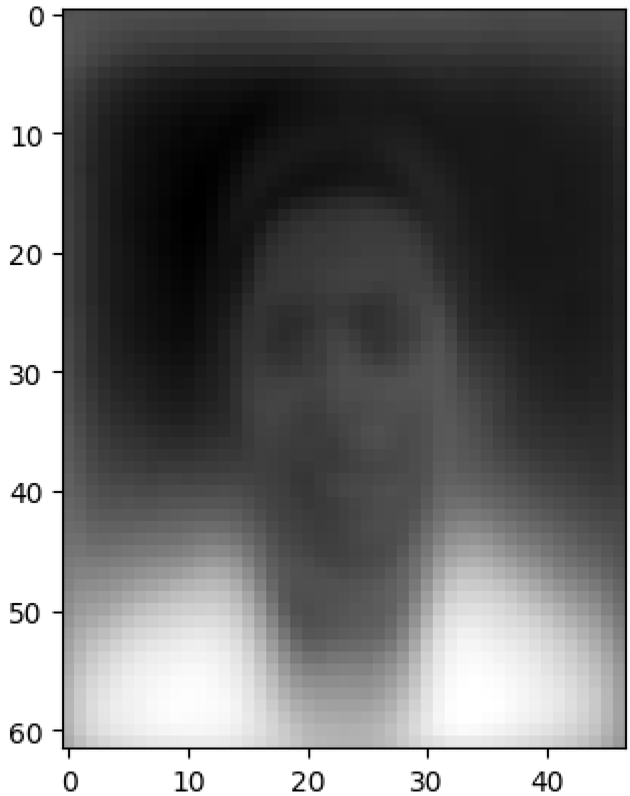

0
10
20
30
40
50
60
0
10
20
30
40

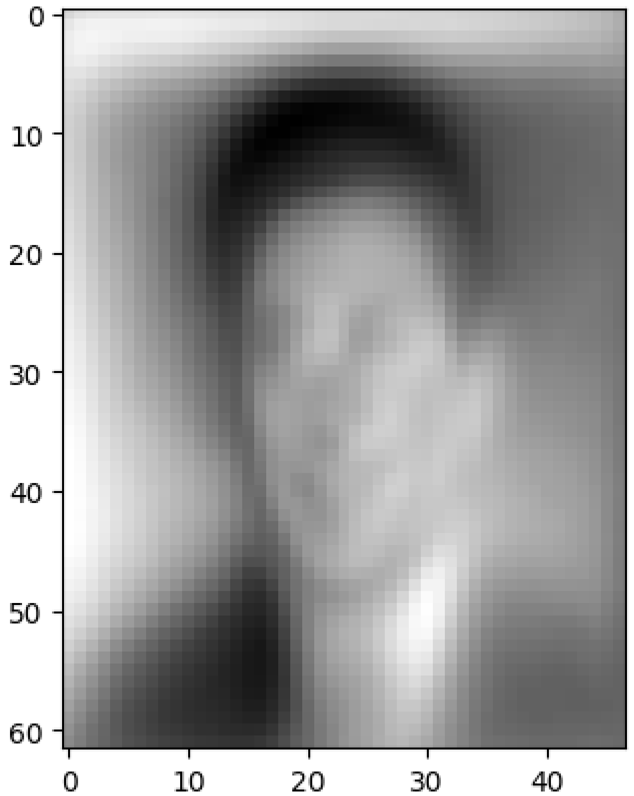

0
10
20
30
40
50
60
0
10
20
30
40

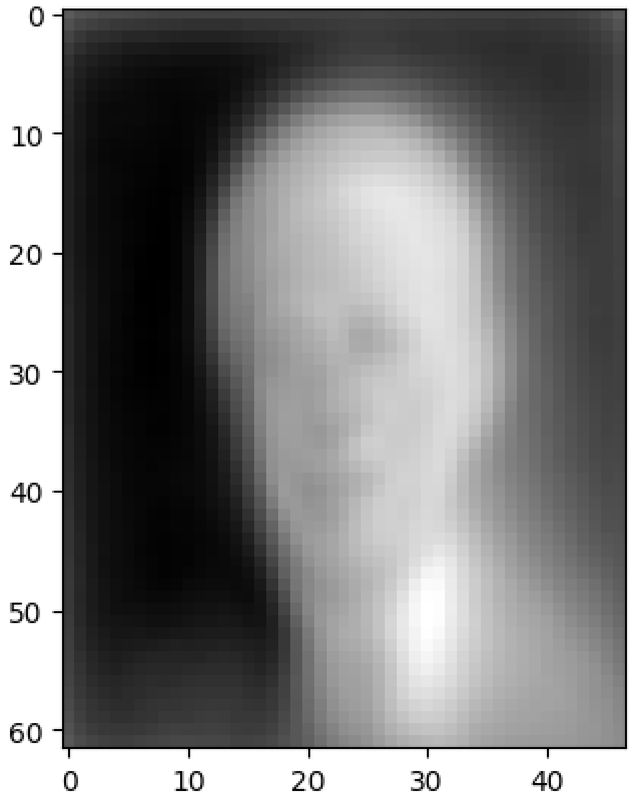

0
10
20
30
40
50
60
0
10
20
30
40

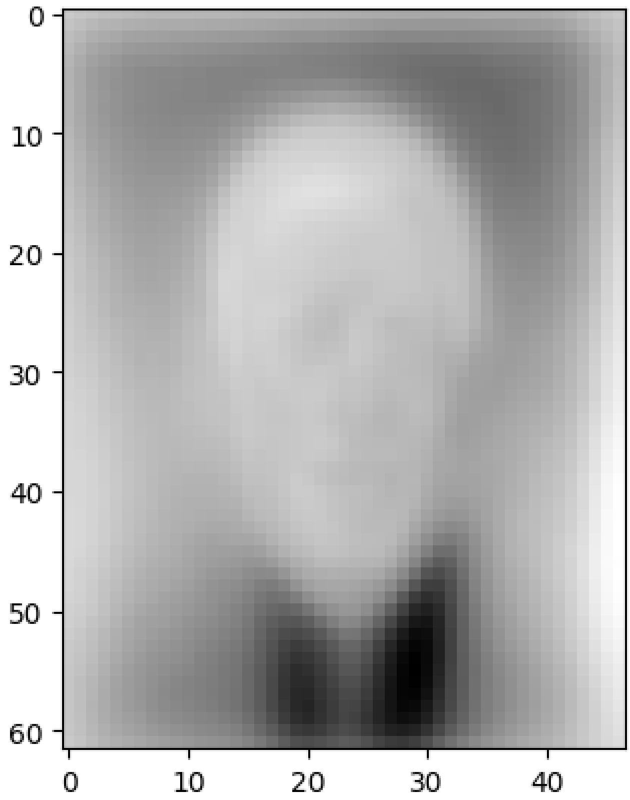

0
10
20
30
40
50
60
0
10
20
30
40

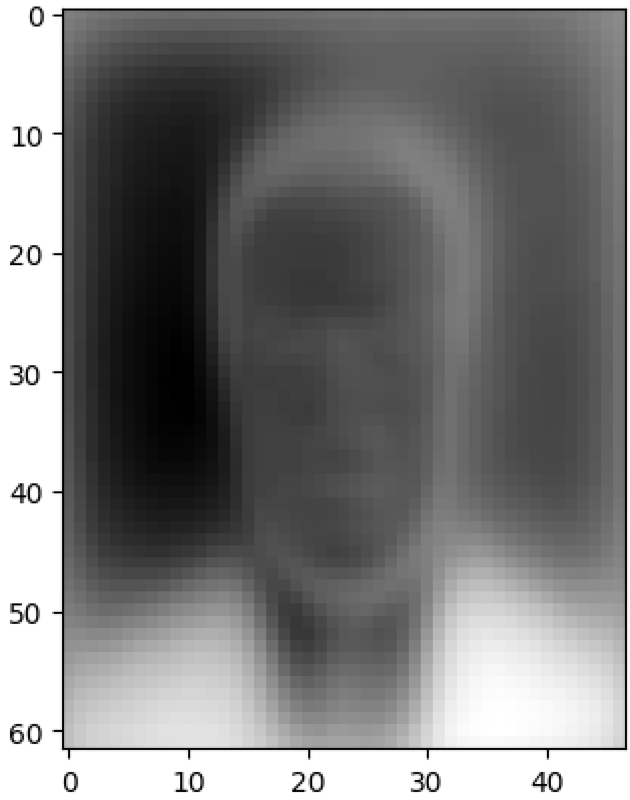

0
10
20
30
40
50
60
0
10
20
30
40

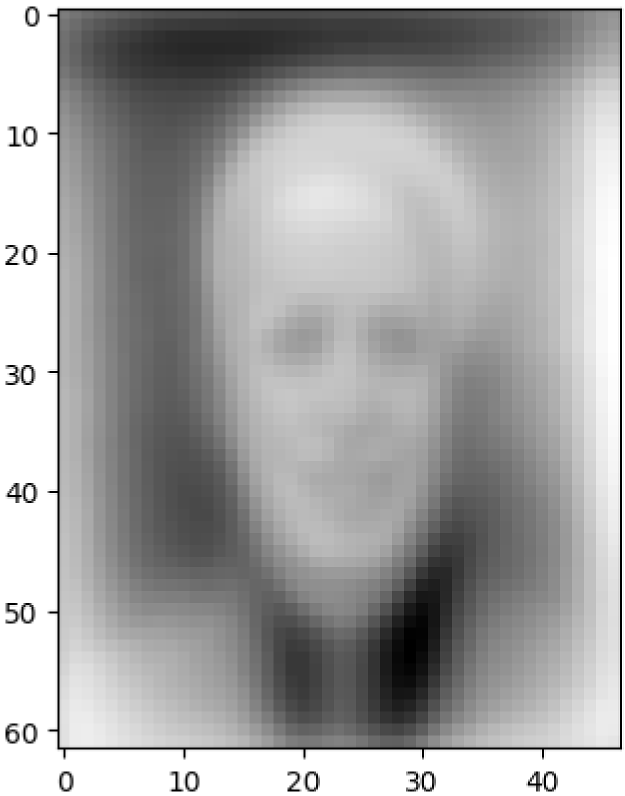

0
10
20
30
40
50
60
0
10
20
30
40

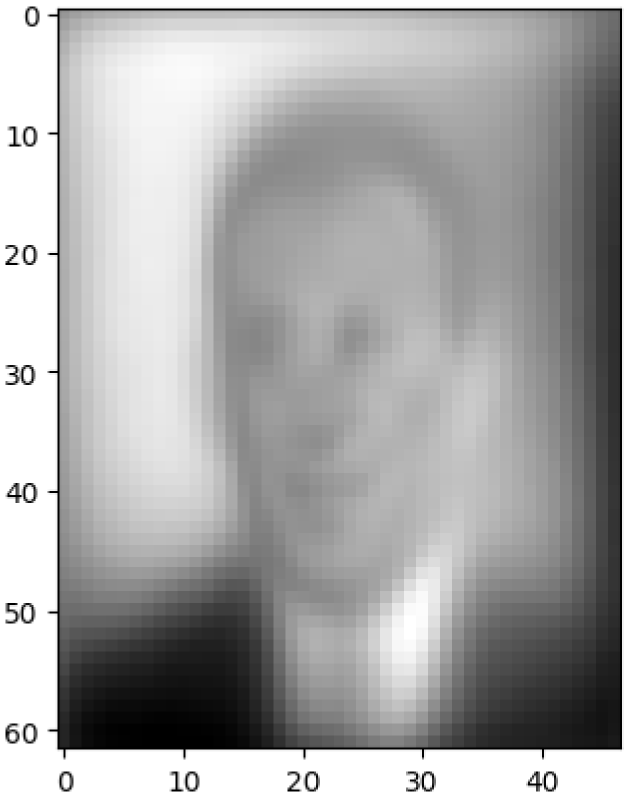


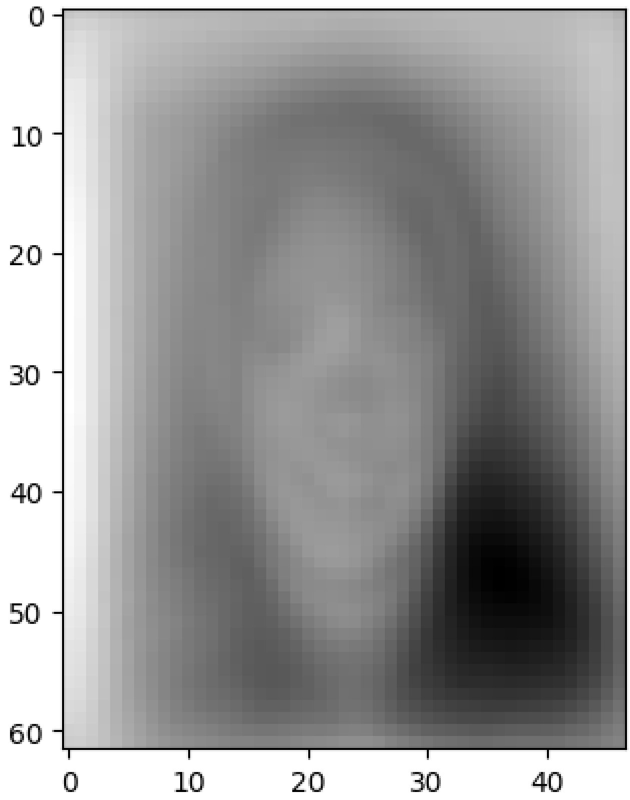


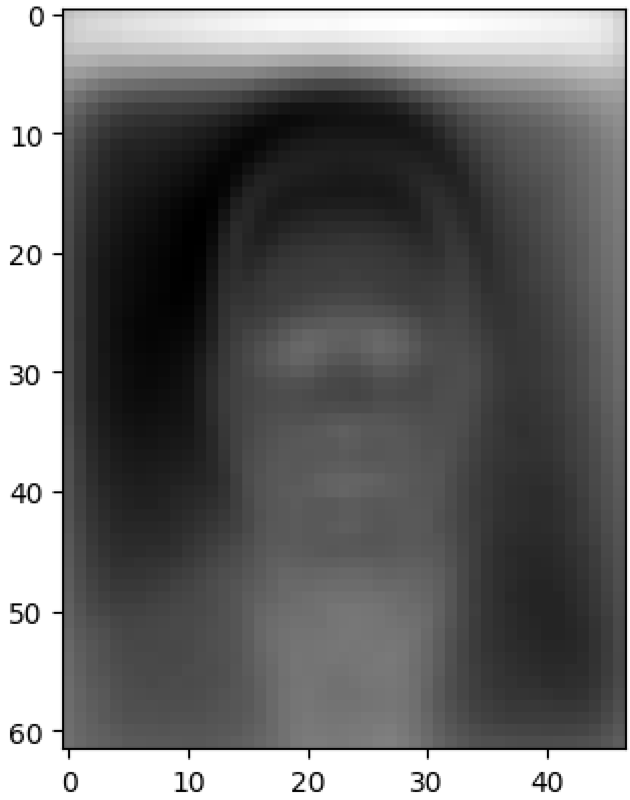

0
10
20
30
40
50
60
0
10
20
30
40

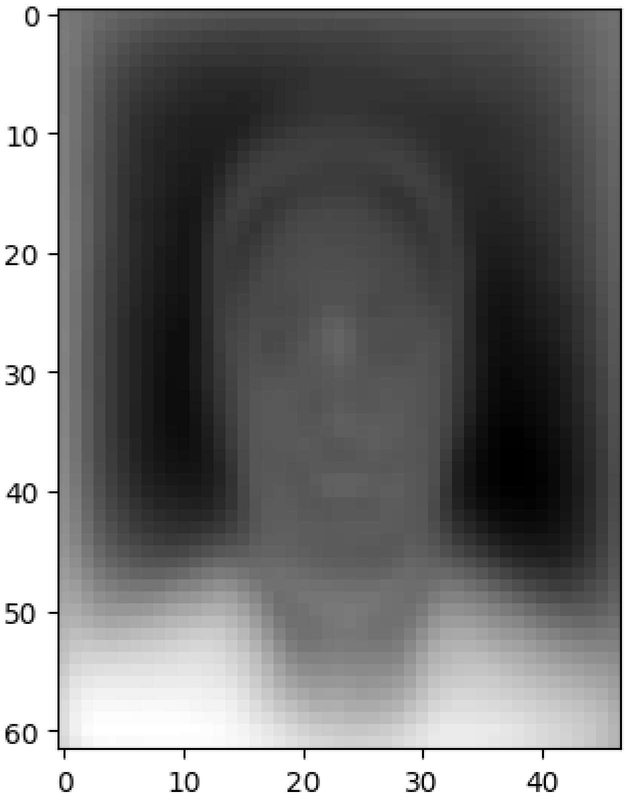

0
10
20
30
40
50
60
0
10
20
30
40

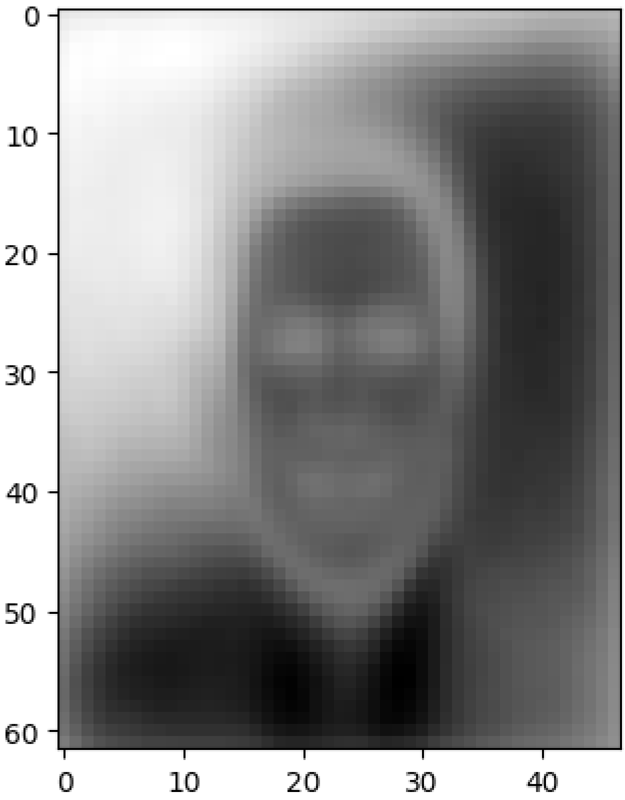

0
10
20
30
40
50
60
0
10
20
30
40

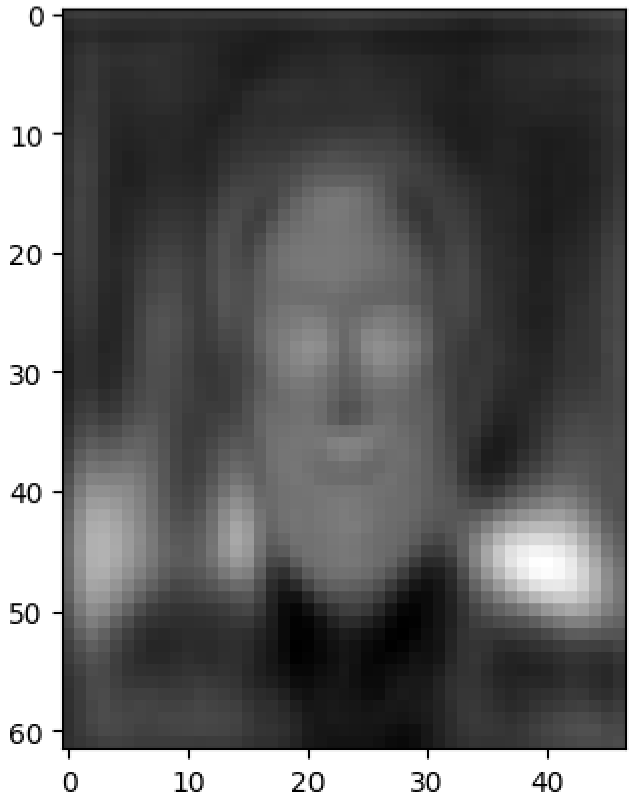

0
10
20
30
40
50
60
0
10
20
30
40

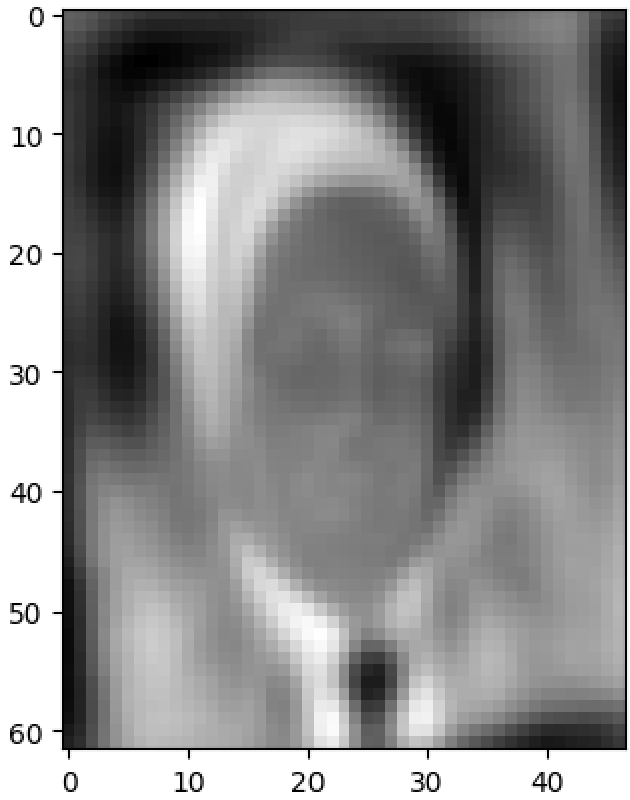

0
10
20
30
40
50
60
0
10
20
30
40

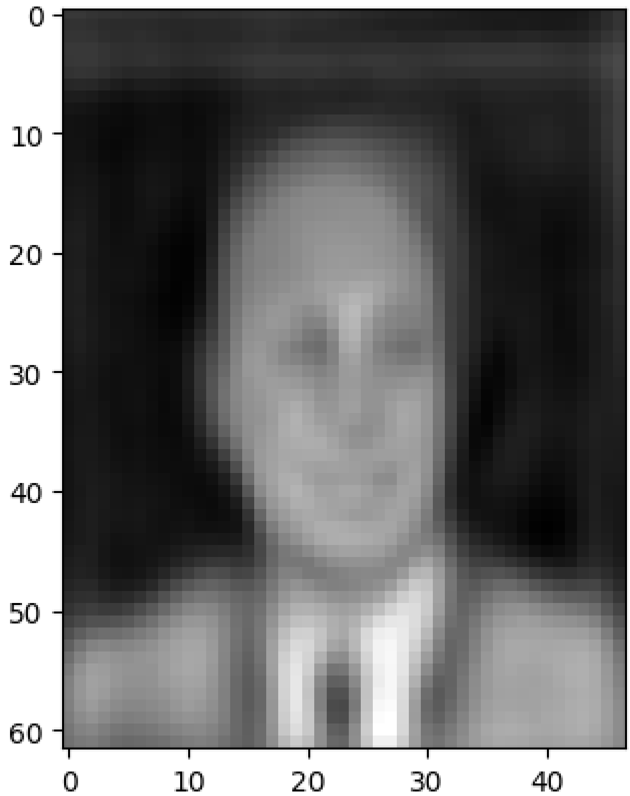

0
10
20
30
40
50
60
0
10
20
30
40

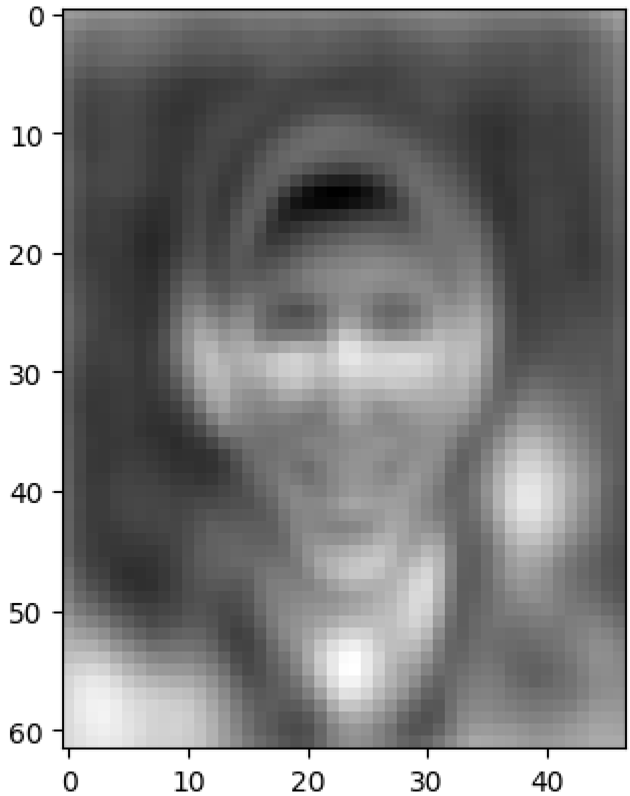


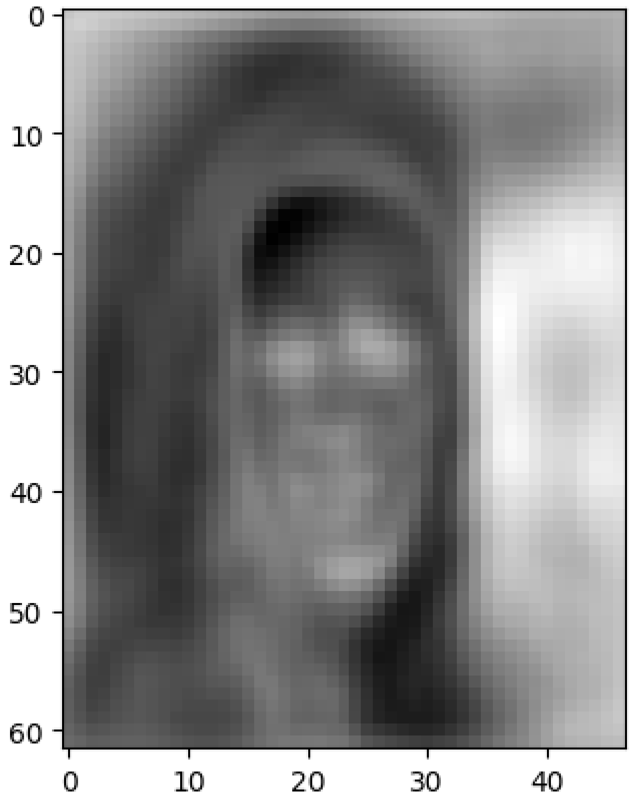


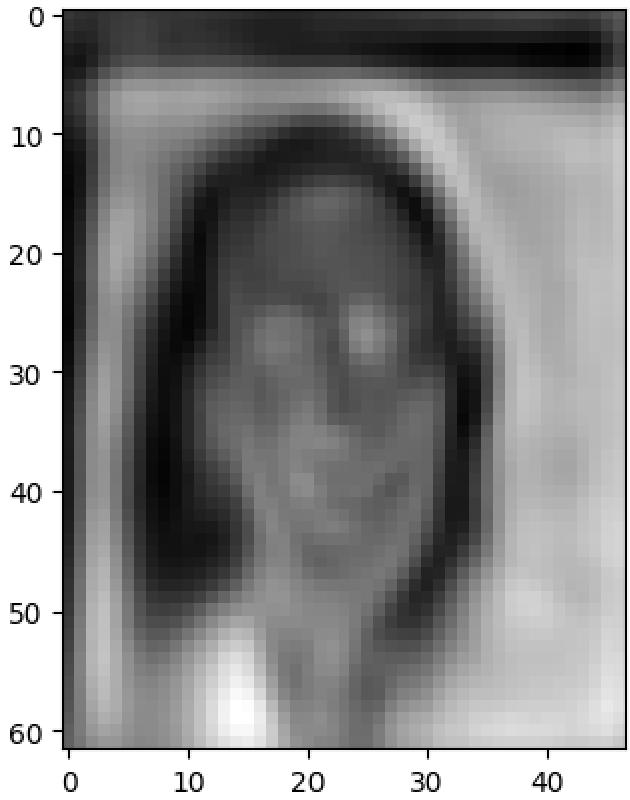

0
10
20
30
40
50
60
0
10
20
30
40

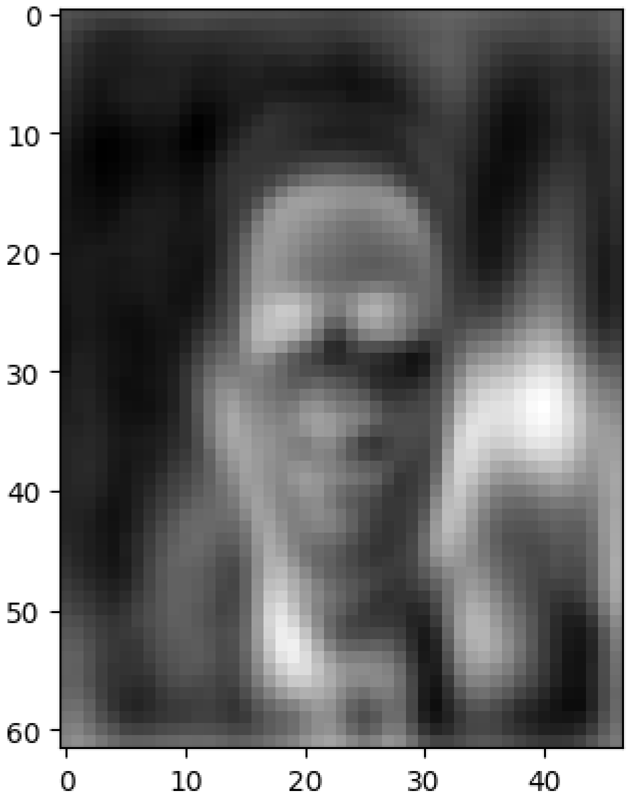

0
10
20
30
40
50
60
0
10
20
30
40

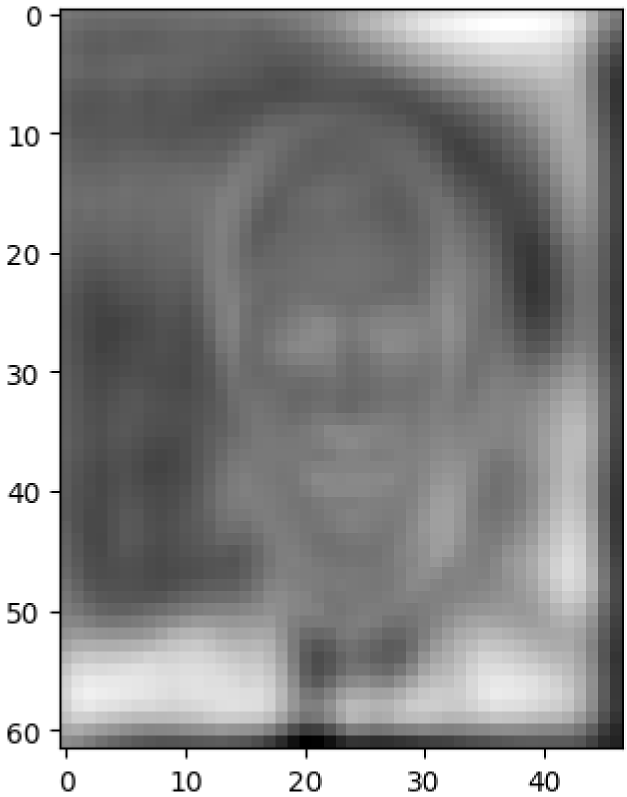

0
10
20
30
40
50
60
0
10
20
30
40

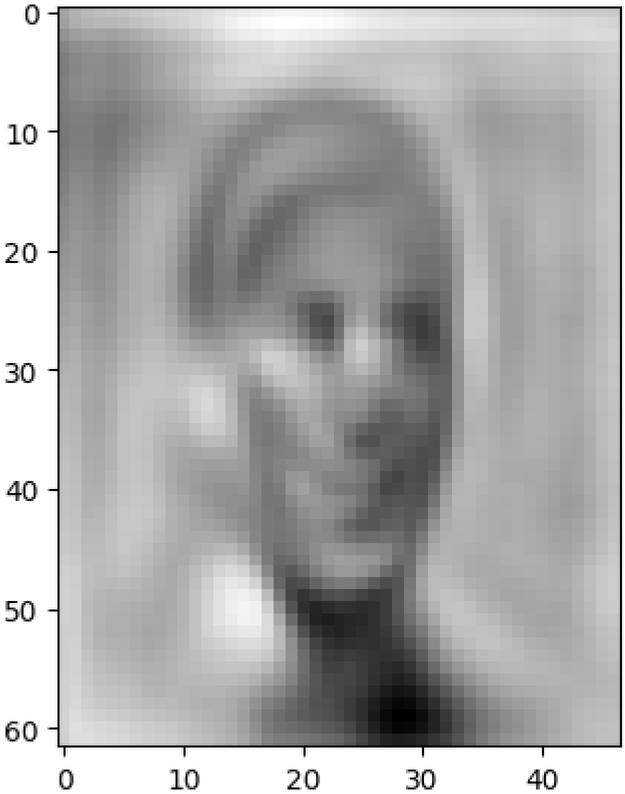

0
10
20
30
40
50
60
0
10
20
30
40

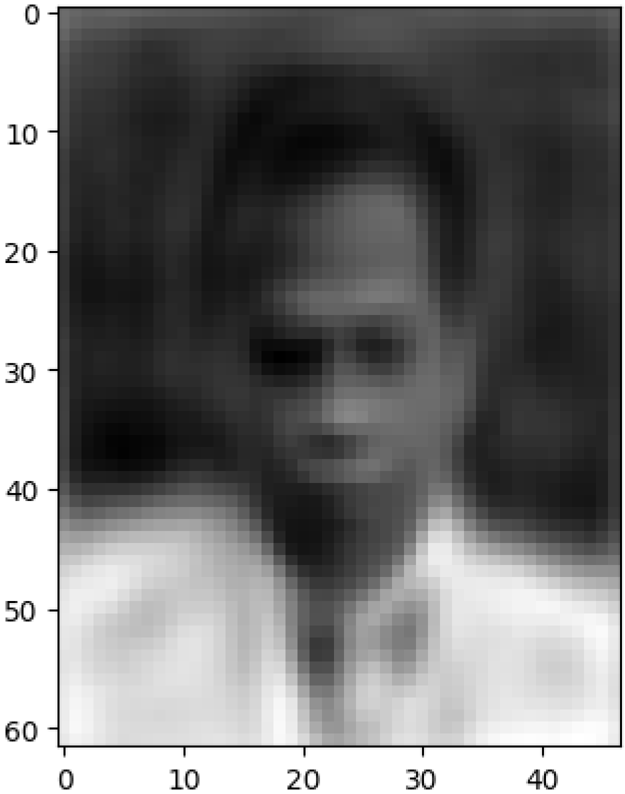

0
10
20
30
40
50
60
0
10
20
30
40

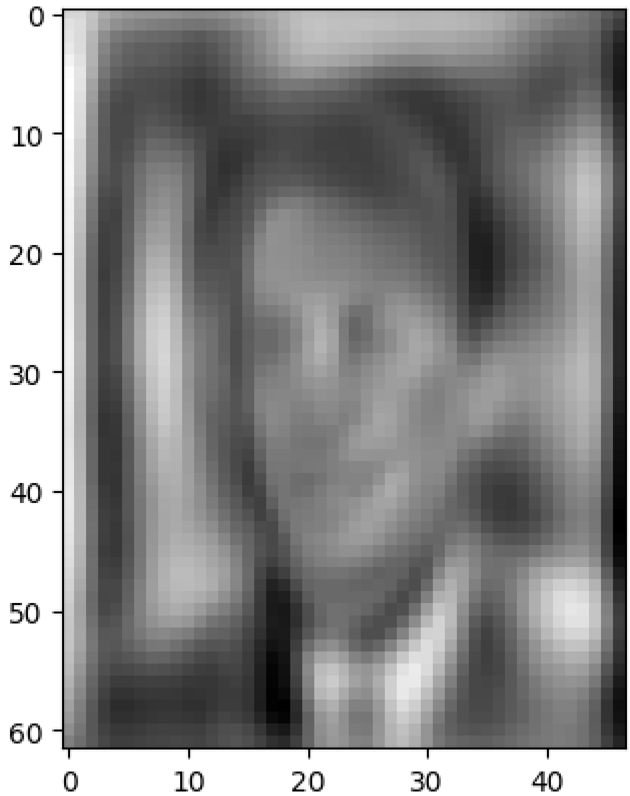

0
10
20
30
40
50
60
0
10
20
30
40

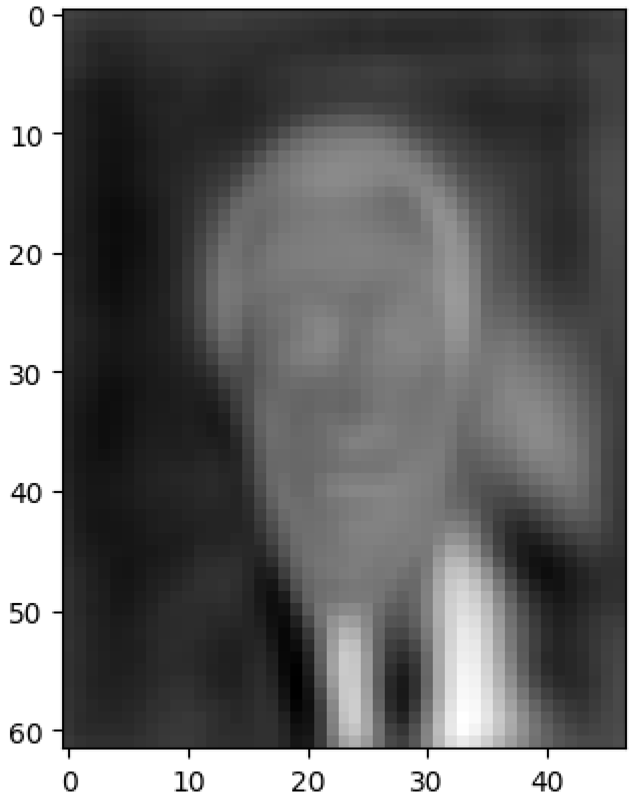


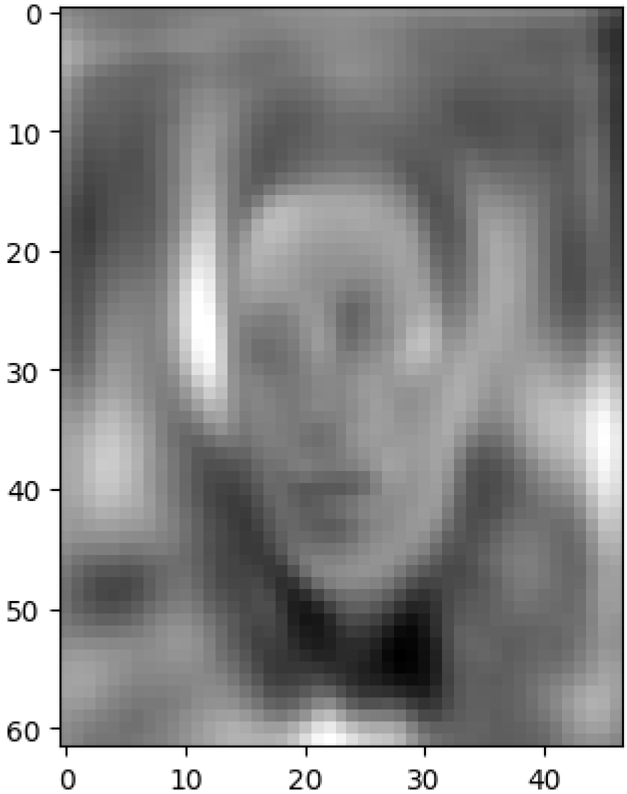


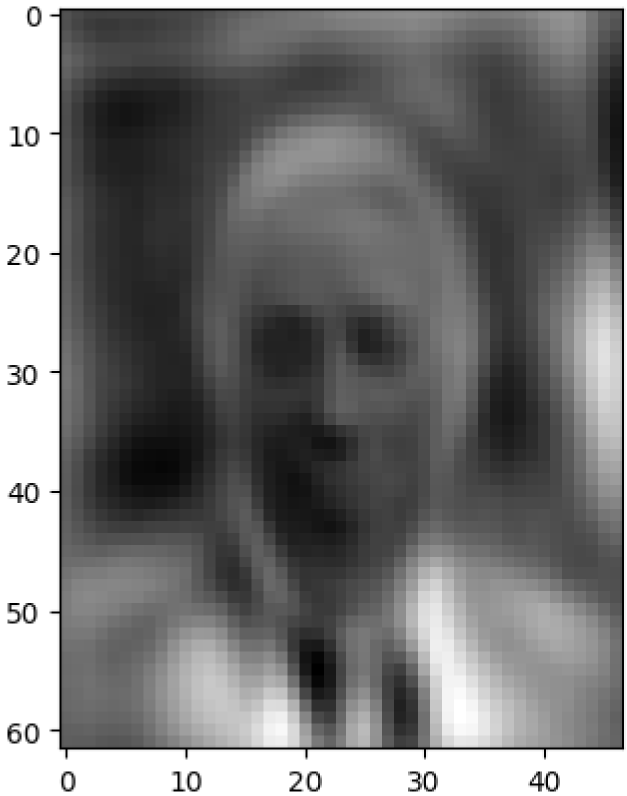

0
10
20
30
40
50
60
0
10
20
30
40

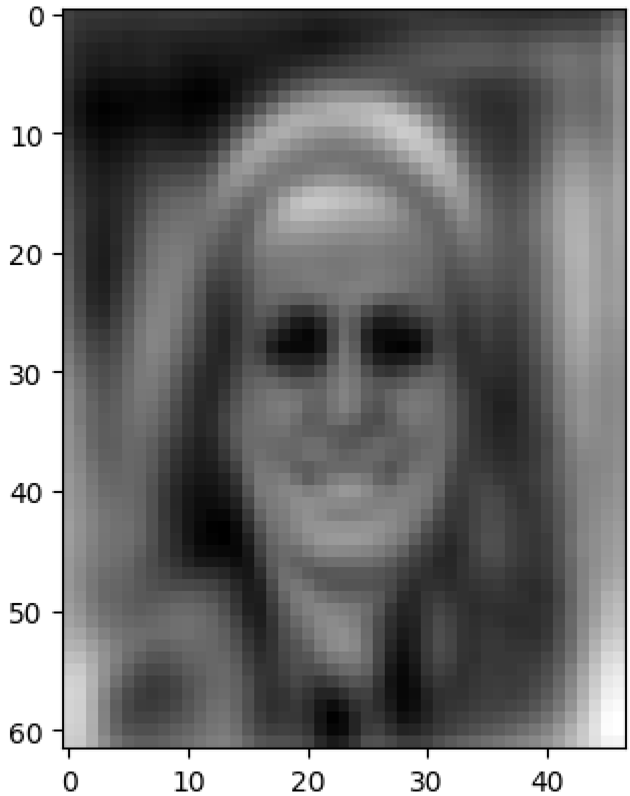

0
10
20
30
40
50
60
0
10
20
30
40

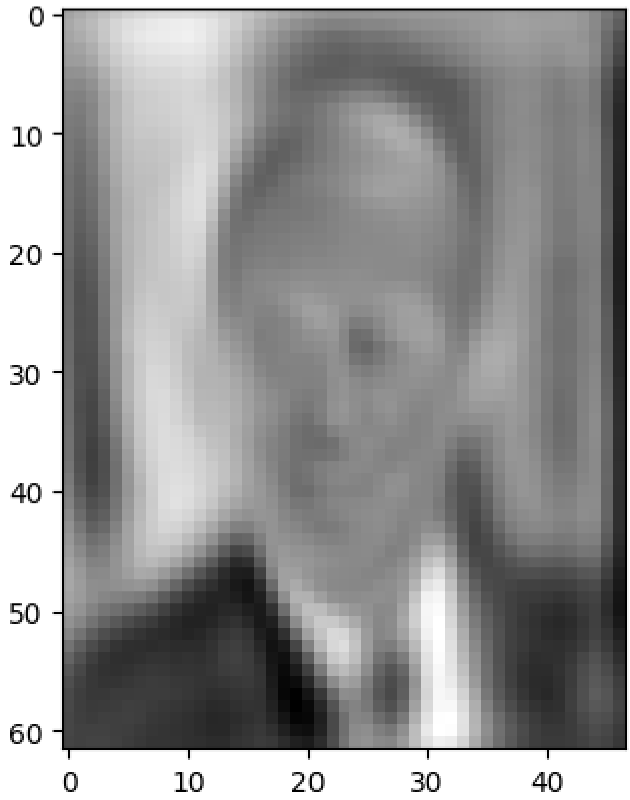

0
10
20
30
40
50
60
0
10
20
30
40

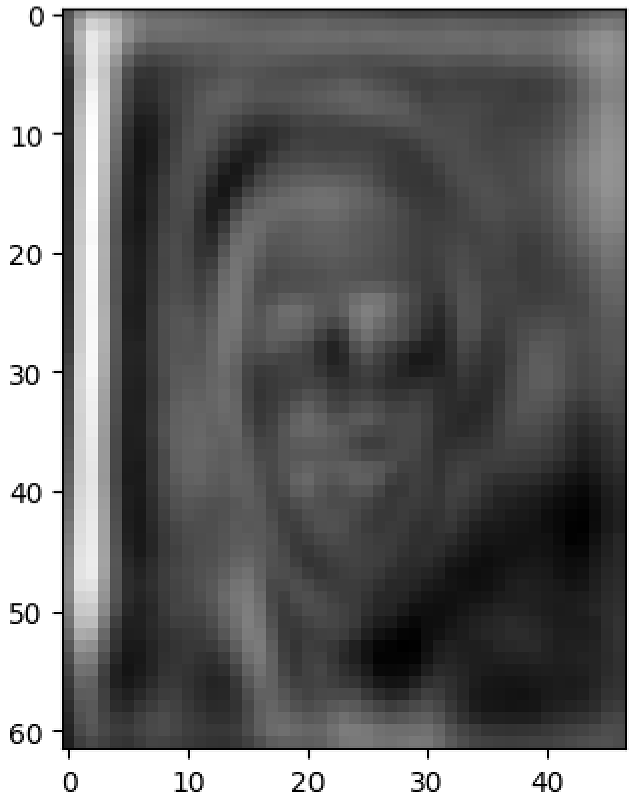

0
10
20
30
40
50
60
0
10
20
30
40

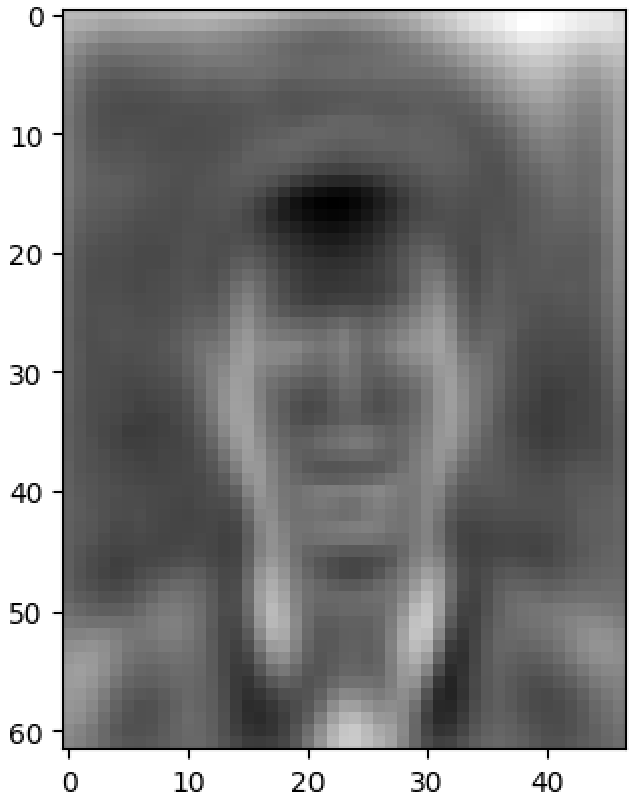

0
10
20
30
40
50
60
0
10
20
30
40

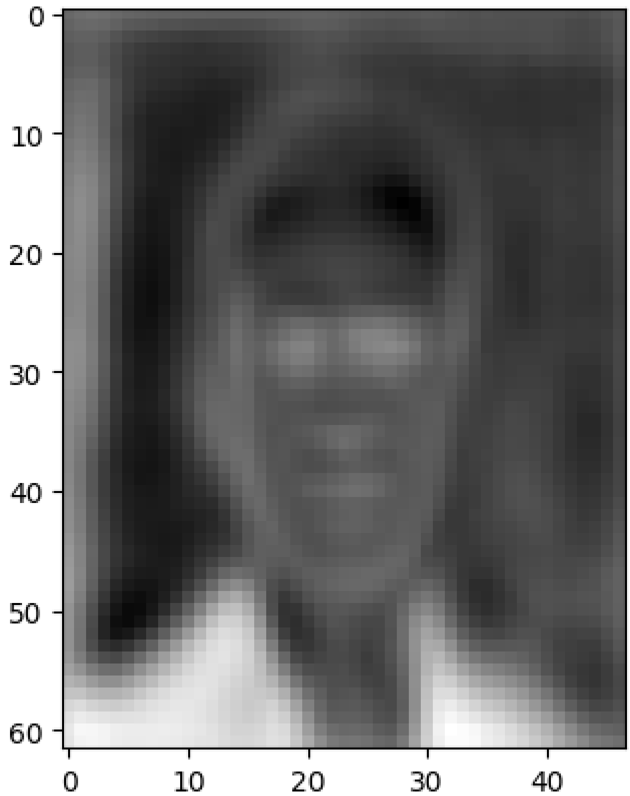

0
10
20
30
40
50
60
0
10
20
30
40

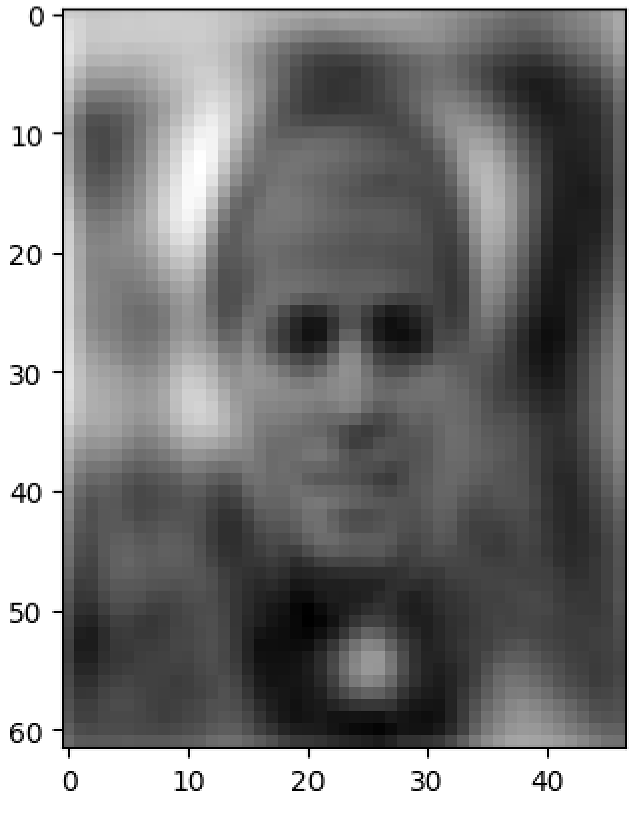

0
10
20
30
40
50
60
0
10
20
30
40

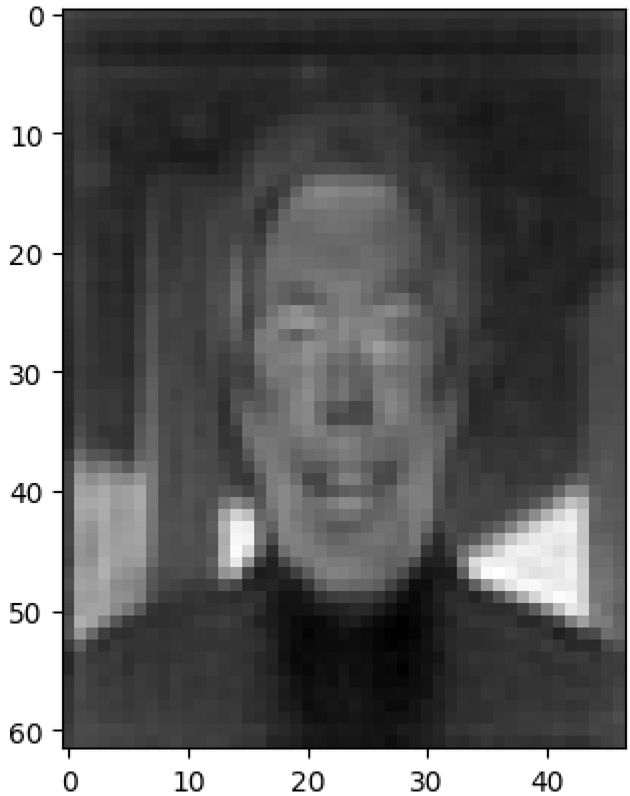

0
10
20
30
40
50
60
0
10
20
30
40

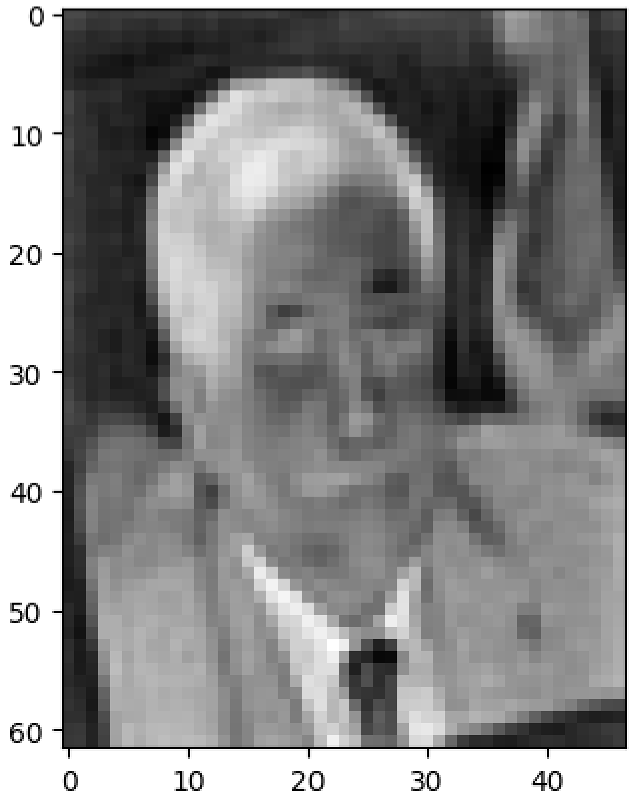

0
10
20
30
40
50
60
0
10
20
30
40

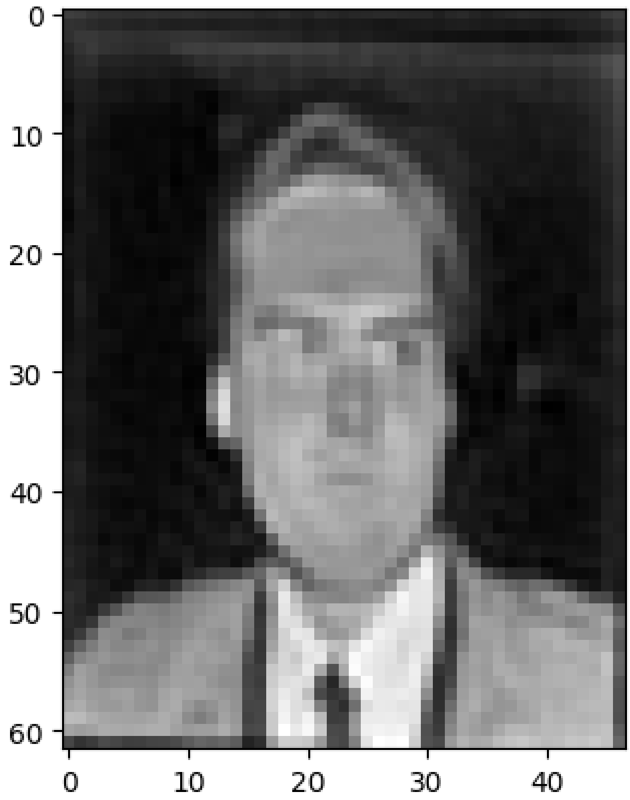

0
10
20
30
40
50
60
0
10
20
30
40

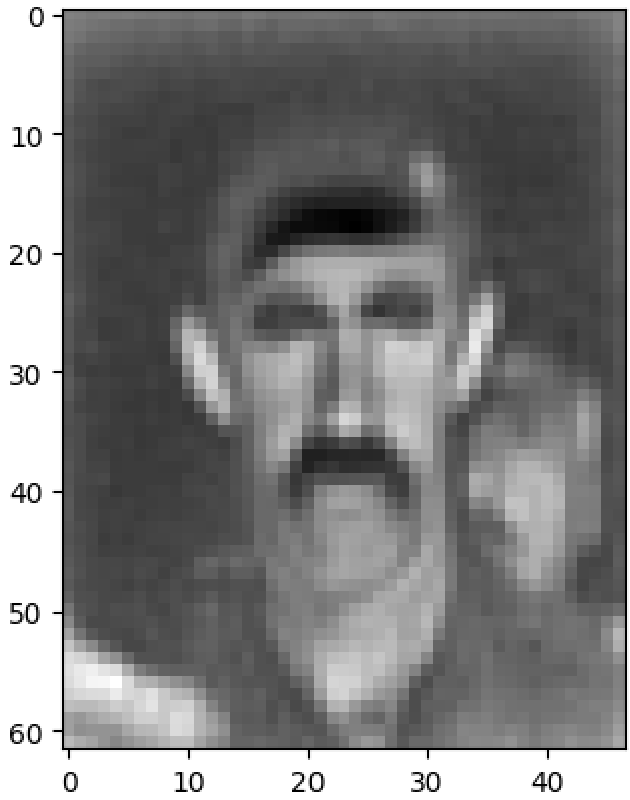

0
10
20
30
40
50
60
0
10
20
30
40

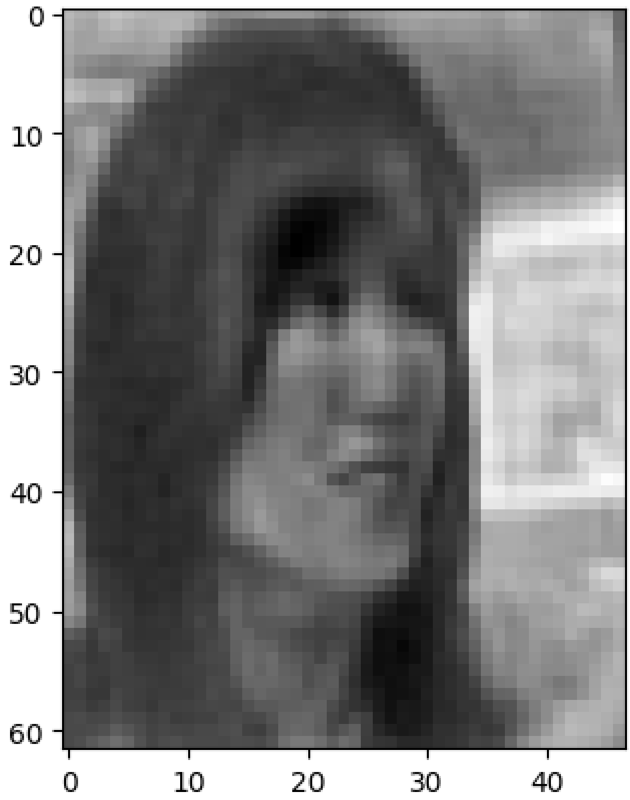

0
10
20
30
40
50
60
0
10
20
30
40

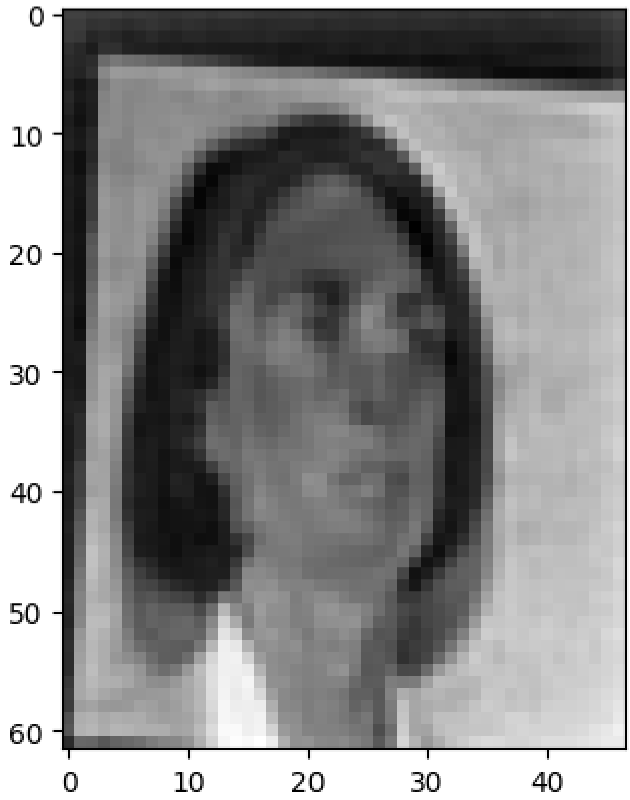


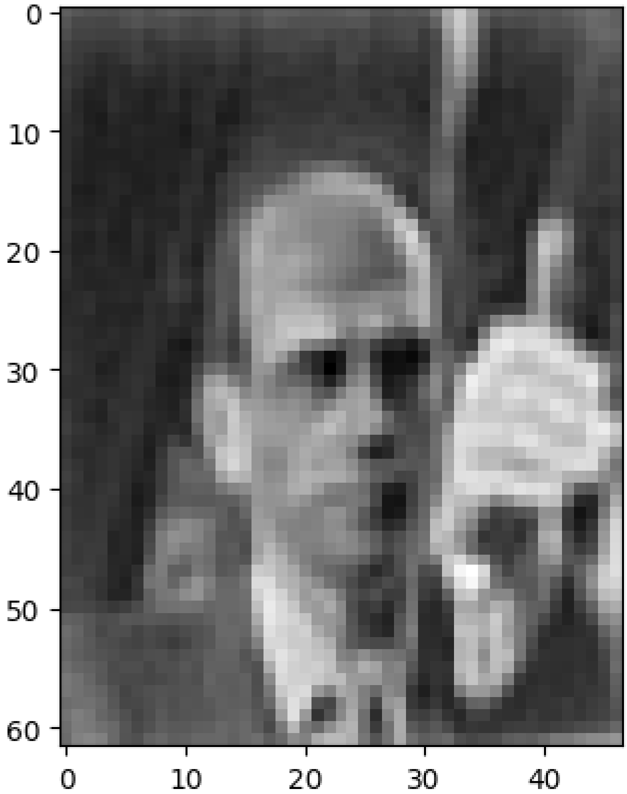


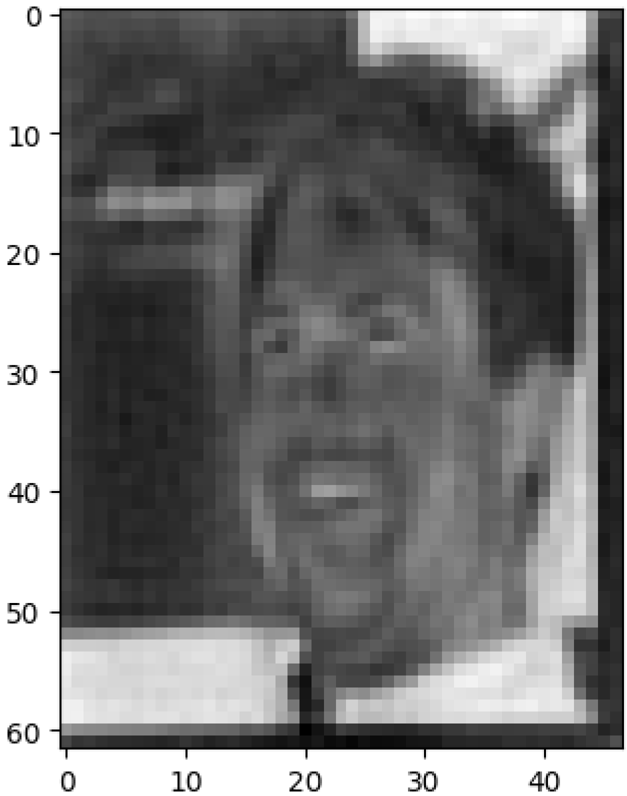

0
10
20
30
40
50
60
0
10
20
30
40

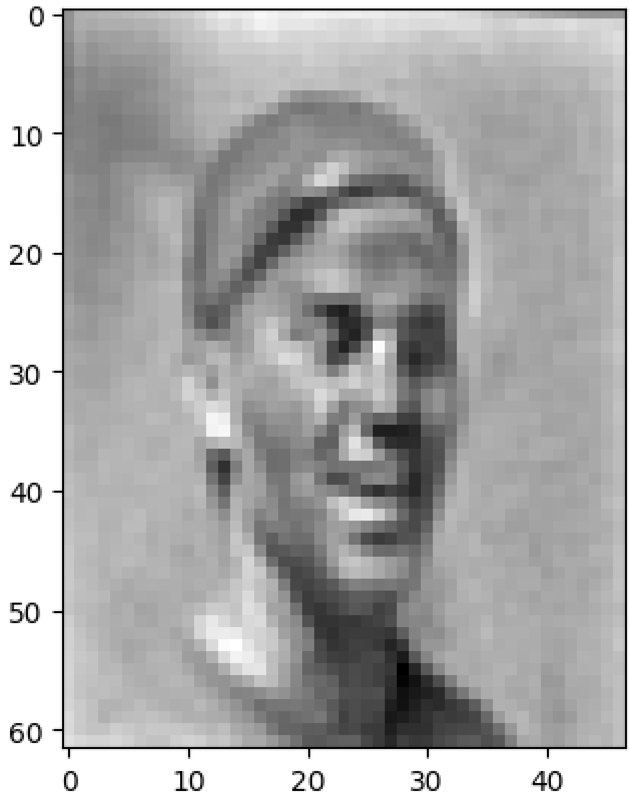

0
10
20
30
40
50
60
0
10
20
30
40

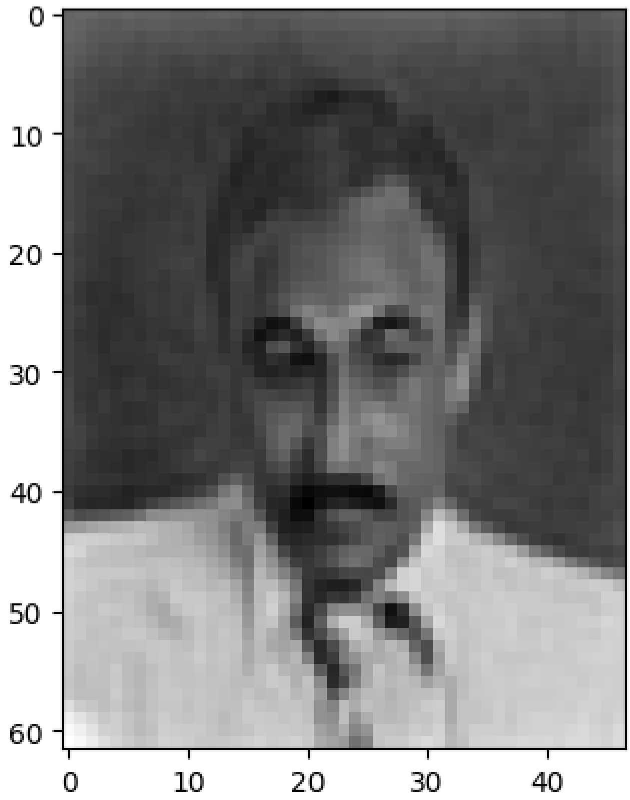

0
10
20
30
40
50
60
0
10
20
30
40

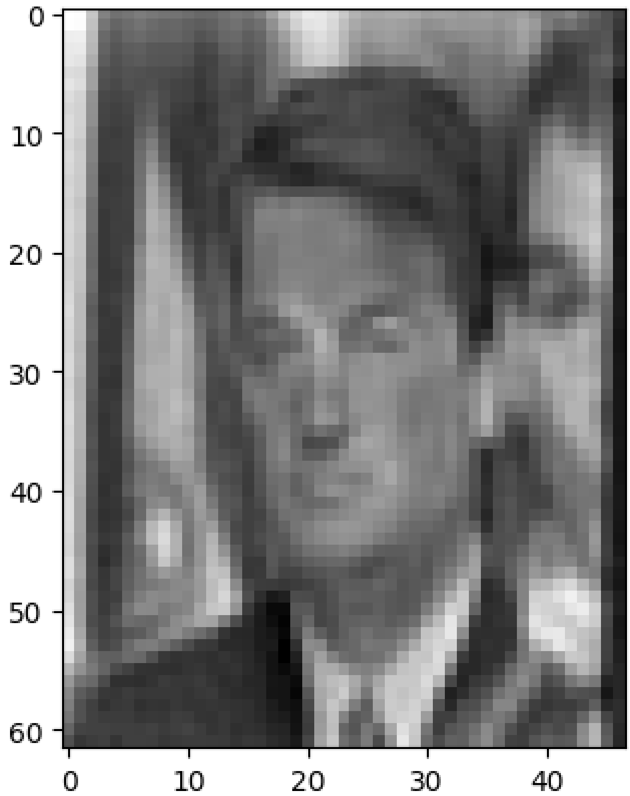

0
10
20
30
40
50
60
0
10
20
30
40

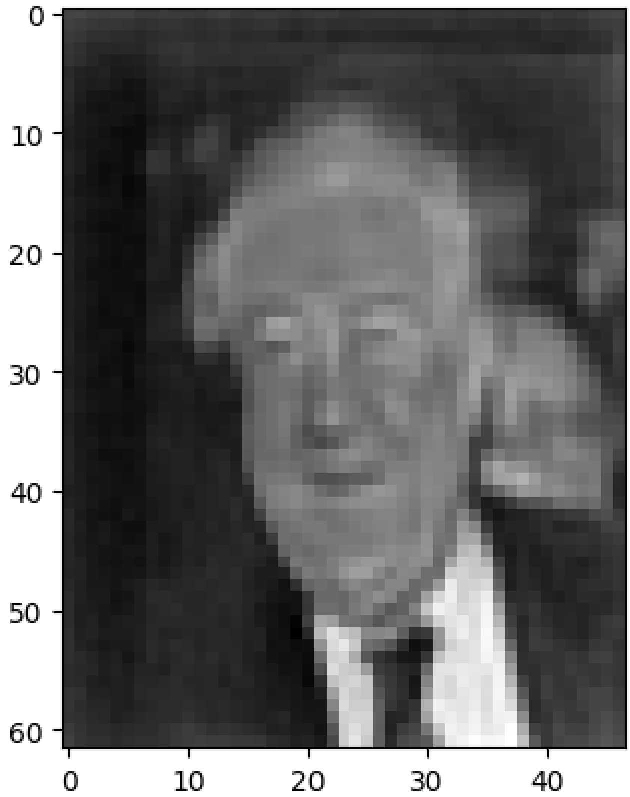

0
10
20
30
40
50
60
0
10
20
30
40

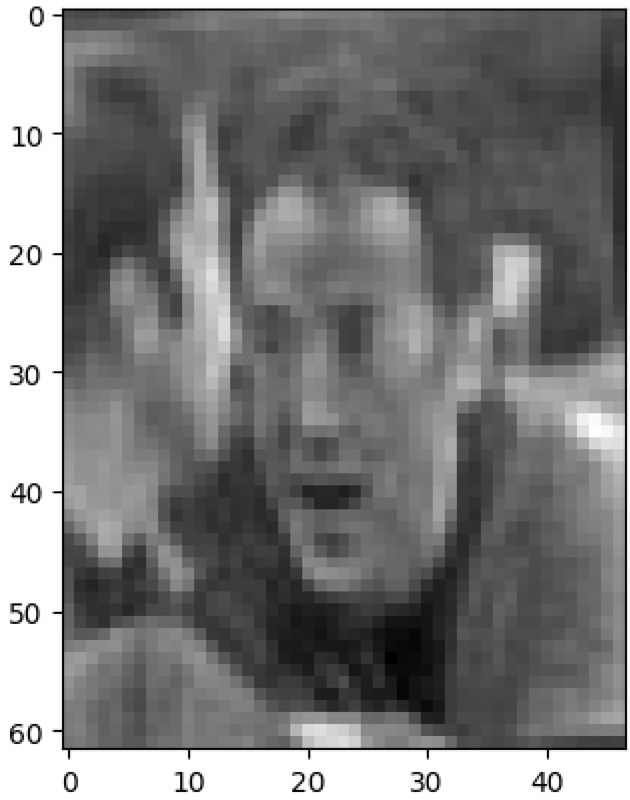

0
10
20
30
40
50
60
0
10
20
30
40

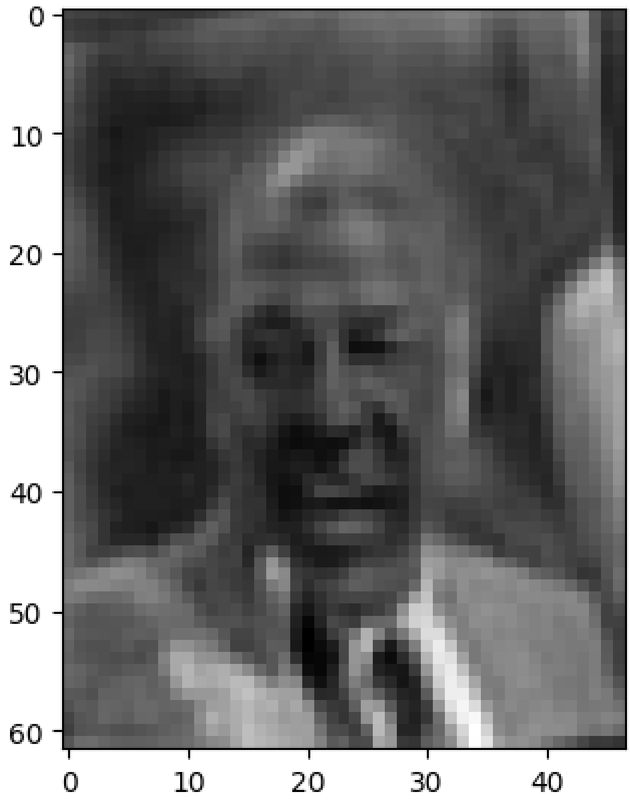

0
10
20
30
40
50
60
0
10
20
30
40

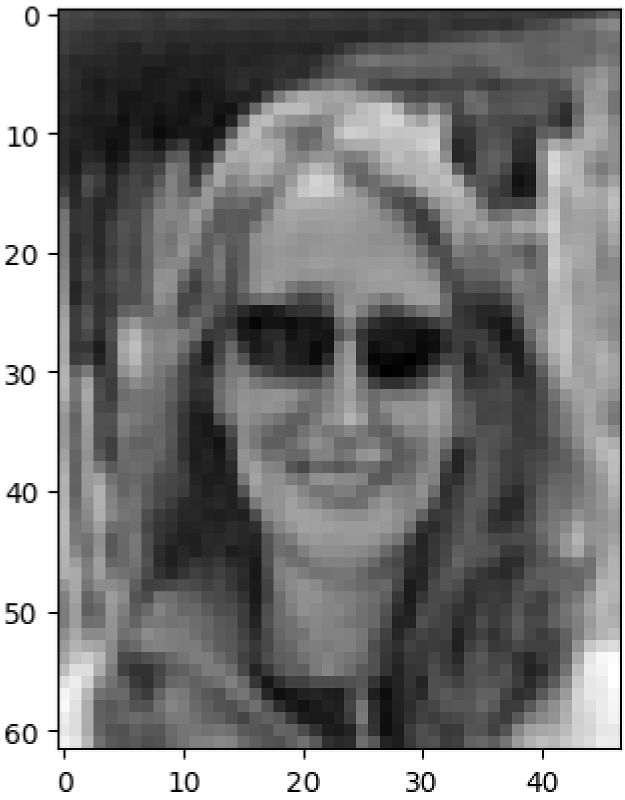

0
10
20
30
40
50
60
0
10
20
30
40

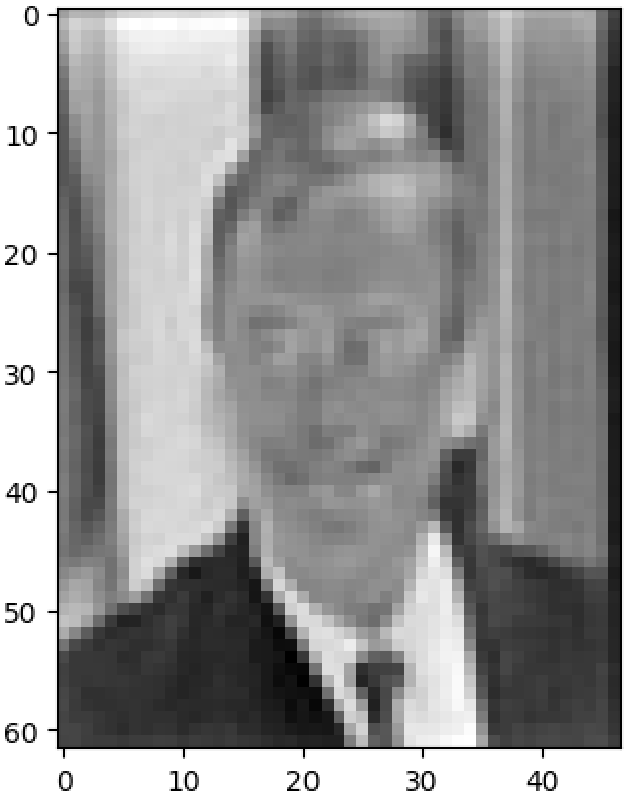


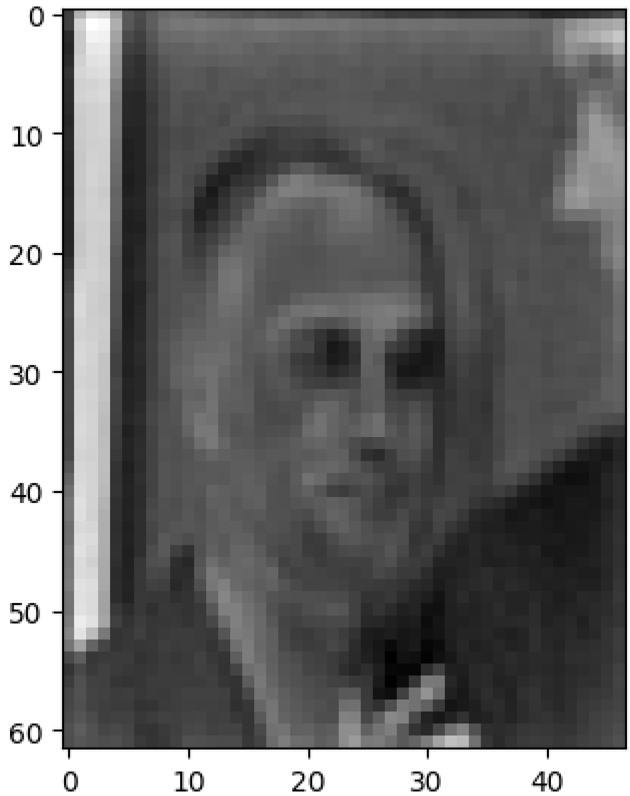


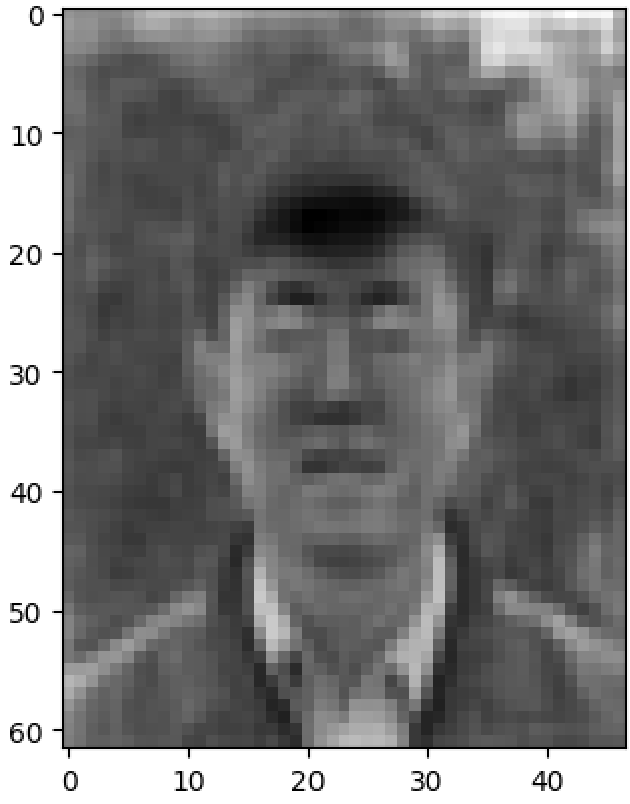

0
10
20
30
40
50
60
0
10
20
30
40

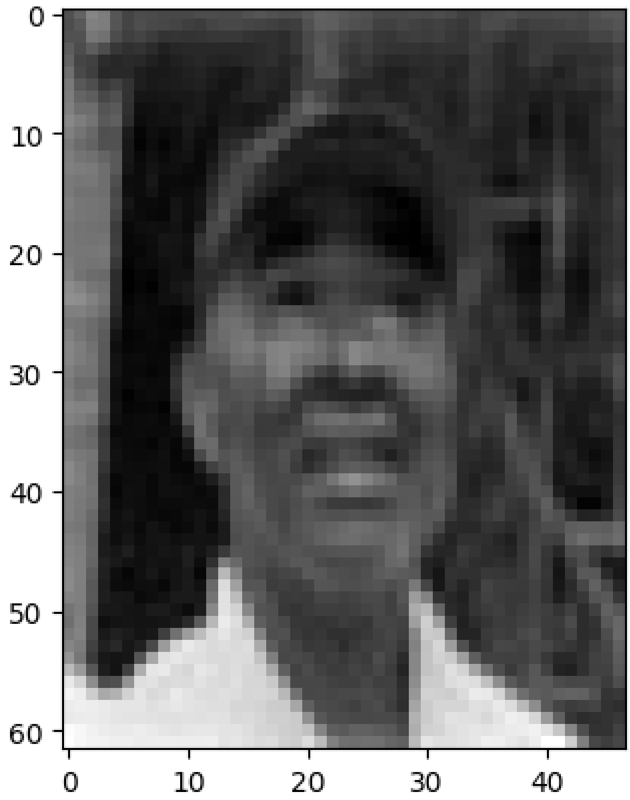

0
10
20
30
40
50
60
0
10
20
30
40

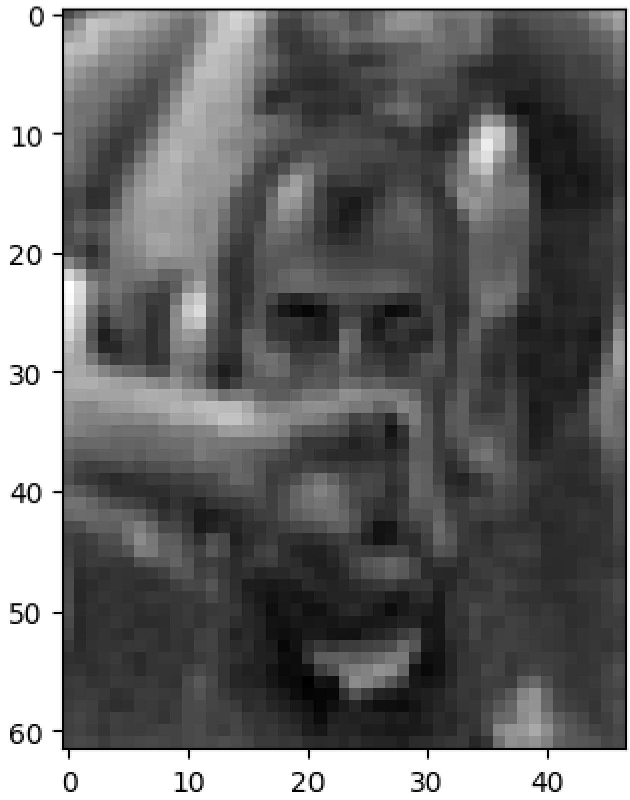

0
10
20
30
40
50
60
0
10
20
30
40


0
10
20
30
40
50
60
0
10
20
30
40


0
10
20
30
40
50
60
0
10
20
30
40


0
10
20
30
40
50
60
0
10
20
30
40

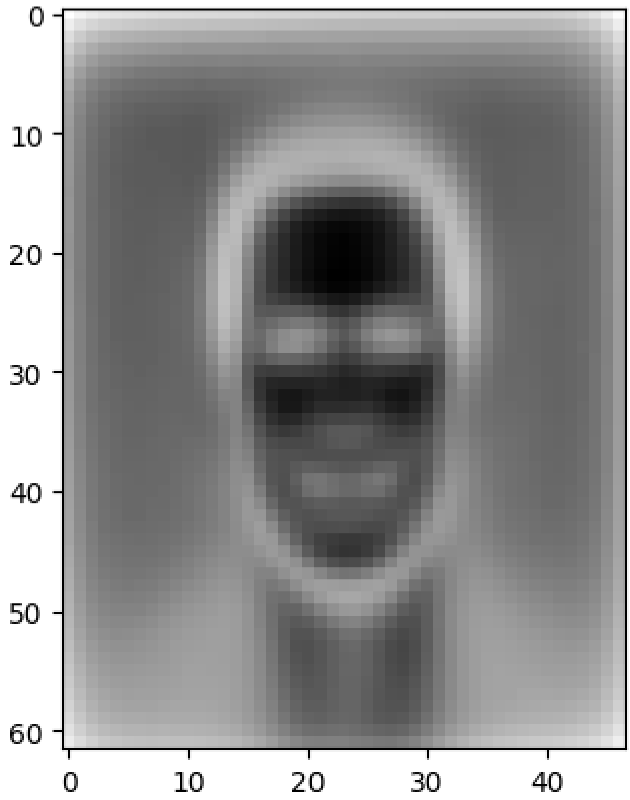


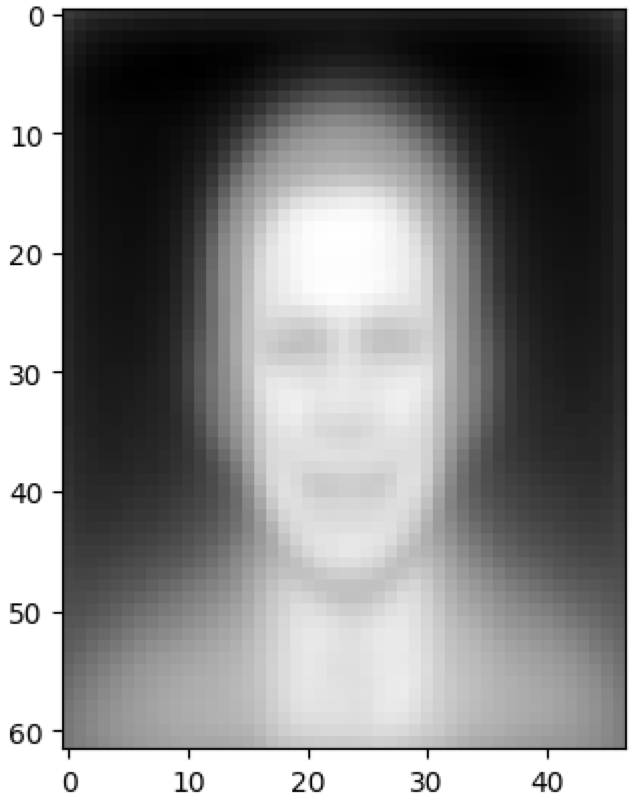


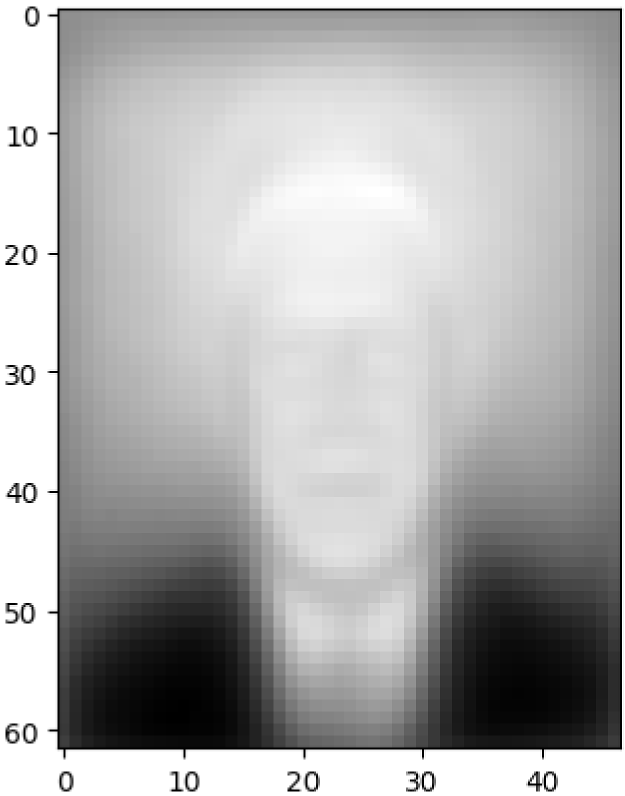
In [ ]:
```
display(Latex(r"\newpage"))
```

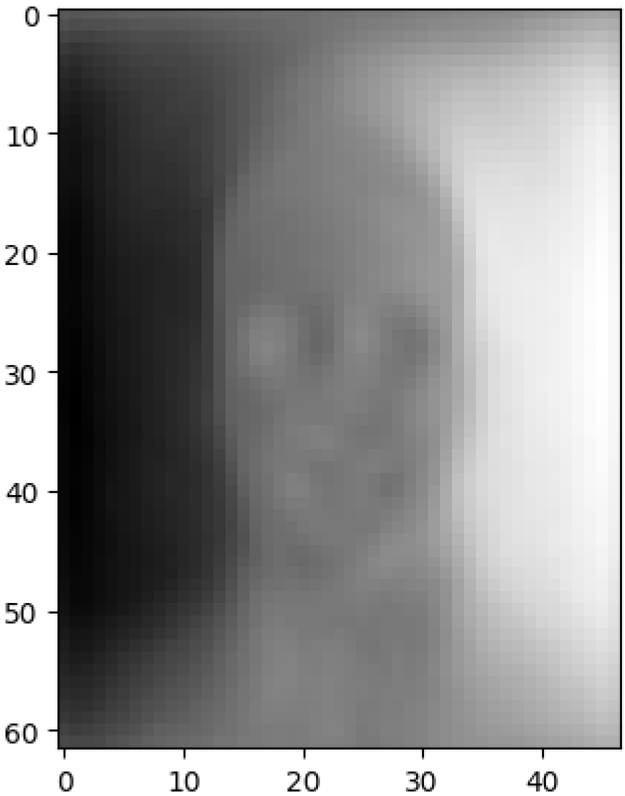
\newpage

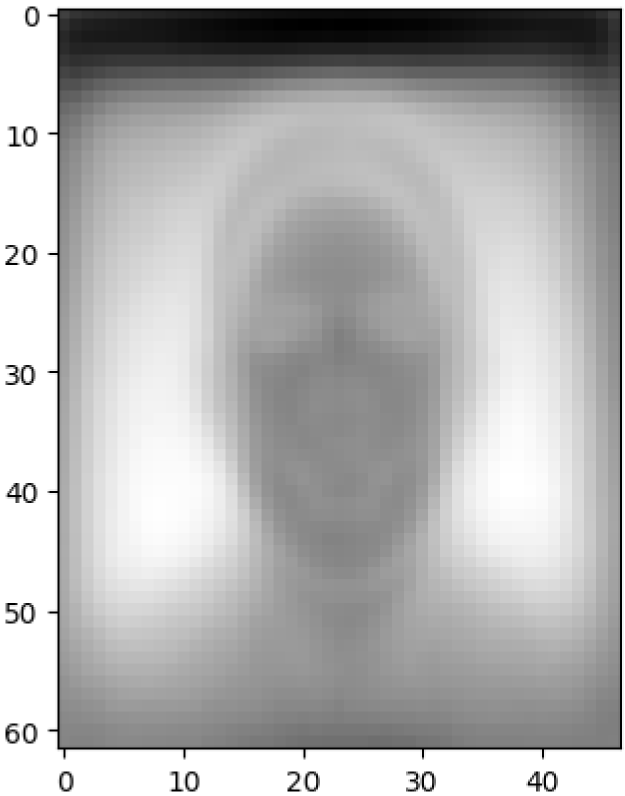
In [ ]:
```
#1.d
for i in range(20):
        matplotlib.pyplot.imshow(eig_vectors[:,i].reshape(62,47), cmap=matplotlib.p
        matplotlib.pyplot.savefig(f'files/eigenface_nr.{i+1}.png')
        matplotlib.pyplot.show()
```

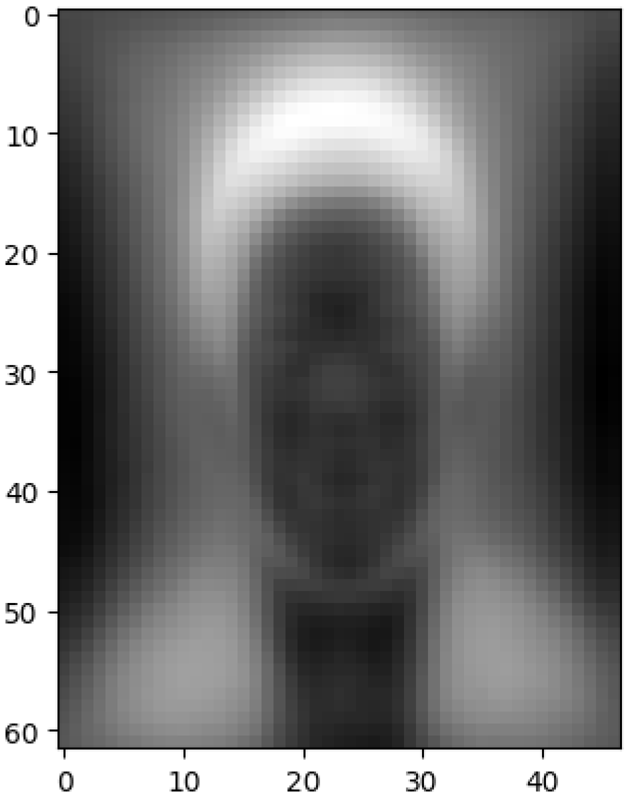

0
10
20
30
40
50
60
0 10 20 30 40

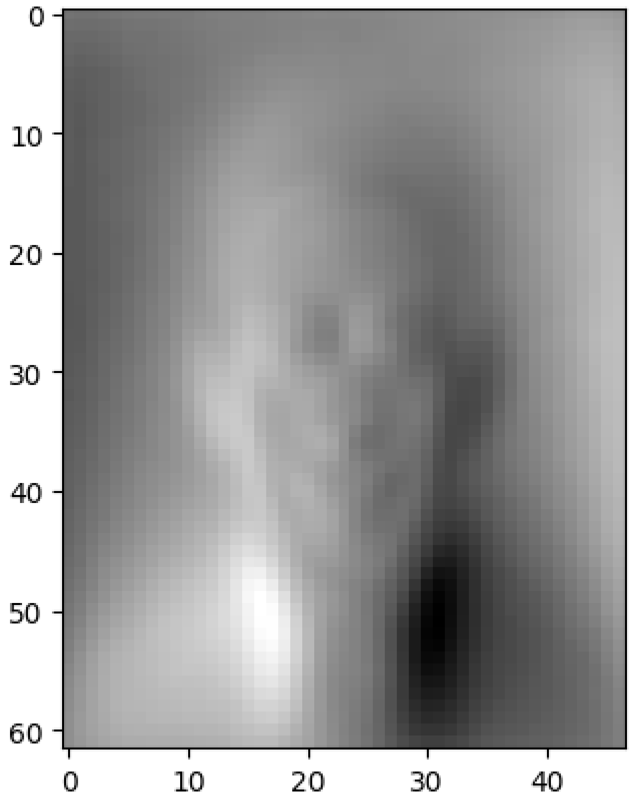

0
10
20
30
40
50
60
0 10 20 30 40

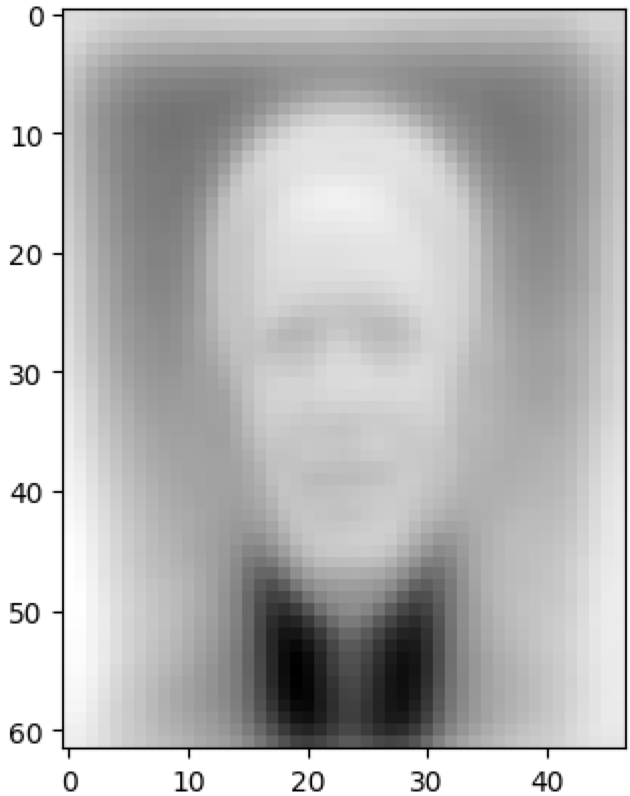

0
10
20
30
40
50
60
0
10
20
30
40

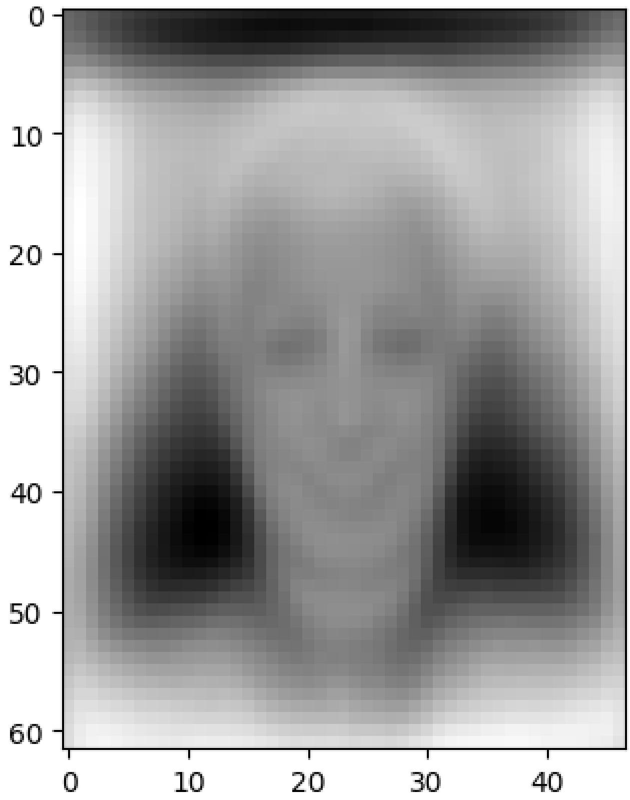

0
10
20
30
40
50
60
0
10
20
30
40

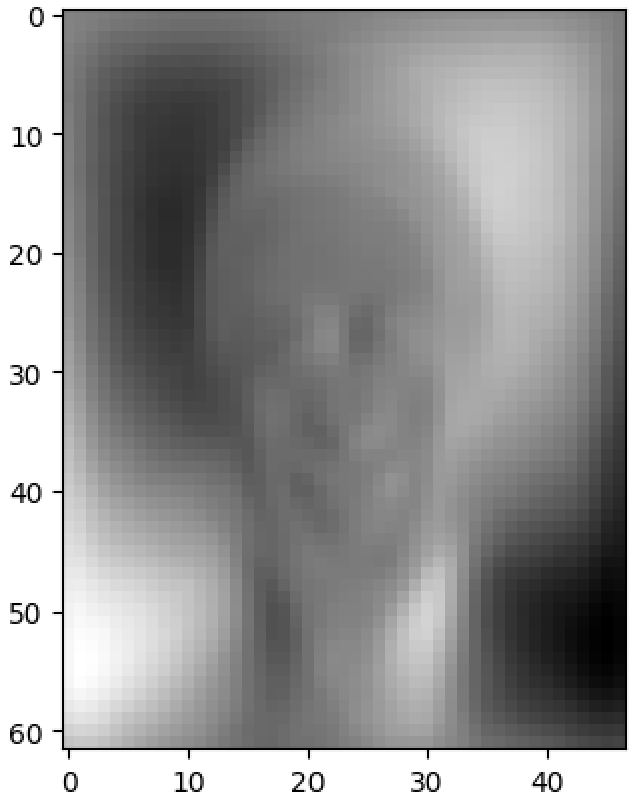

0
10
20
30
40
50
60
0
10
20
30
40

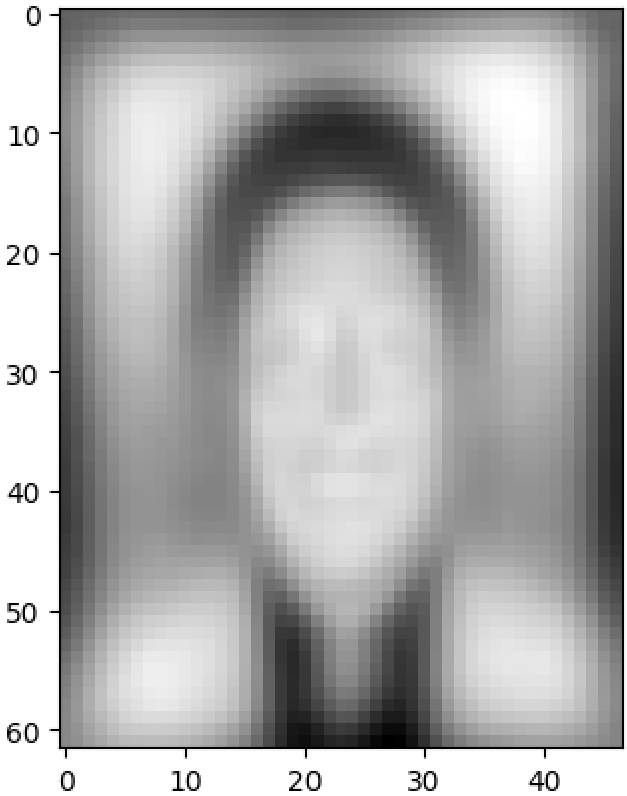

0
10
20
30
40
50
60
0
10
20
30
40

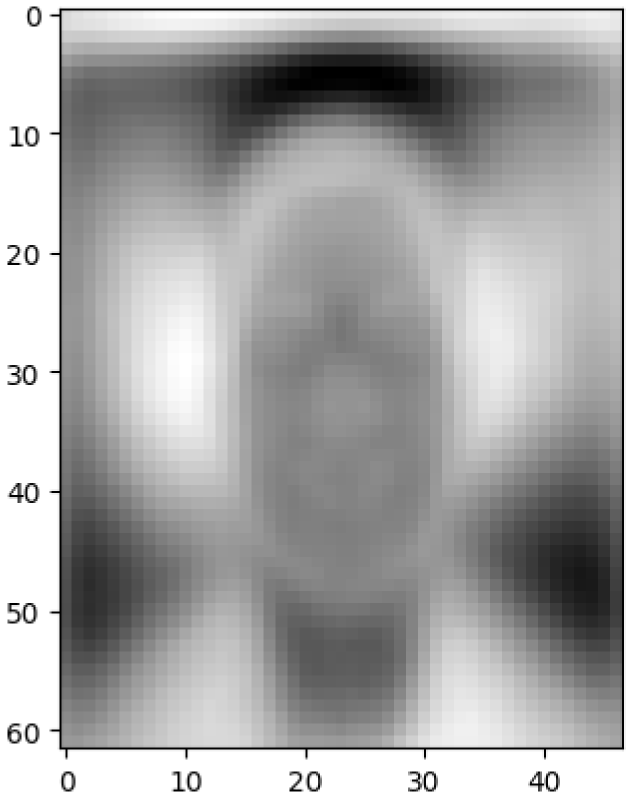

0
10
20
30
40
50
60
0
10
20
30
40

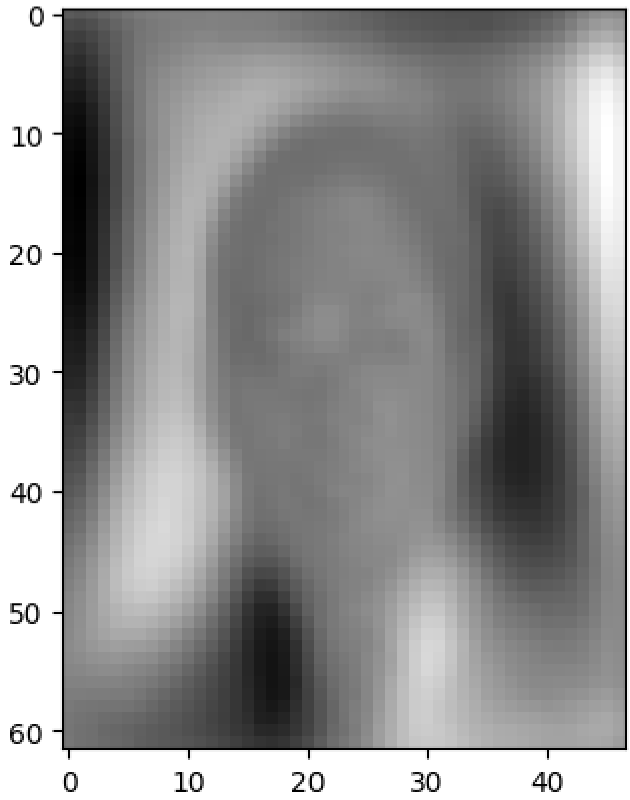

0
10
20
30
40
50
60
0
10
20
30
40

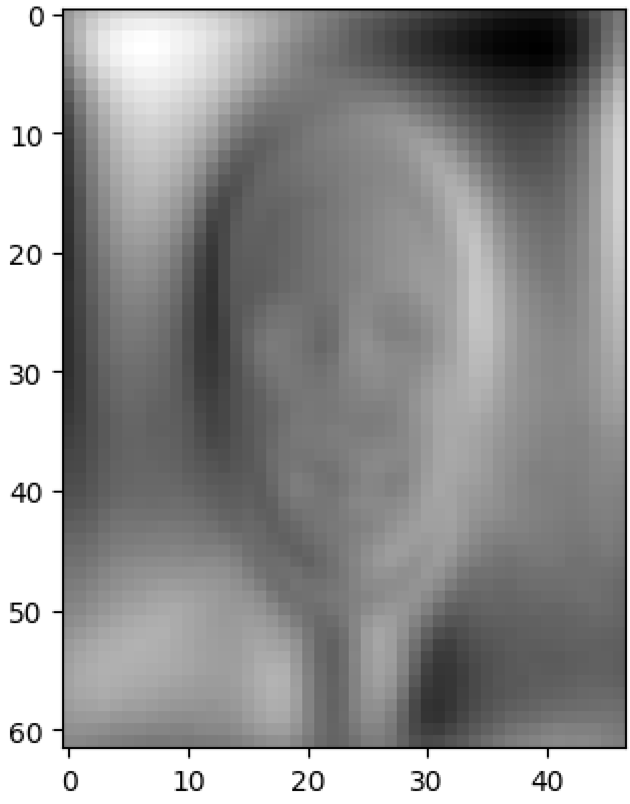

0
10
20
30
40
50
60
0
10
20
30
40

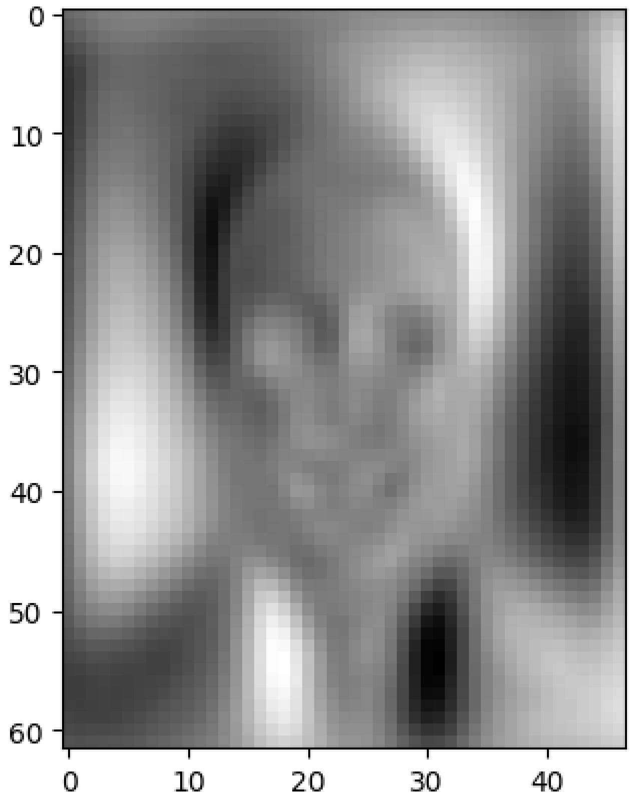

0
10
20
30
40
50
60
0
10
20
30
40

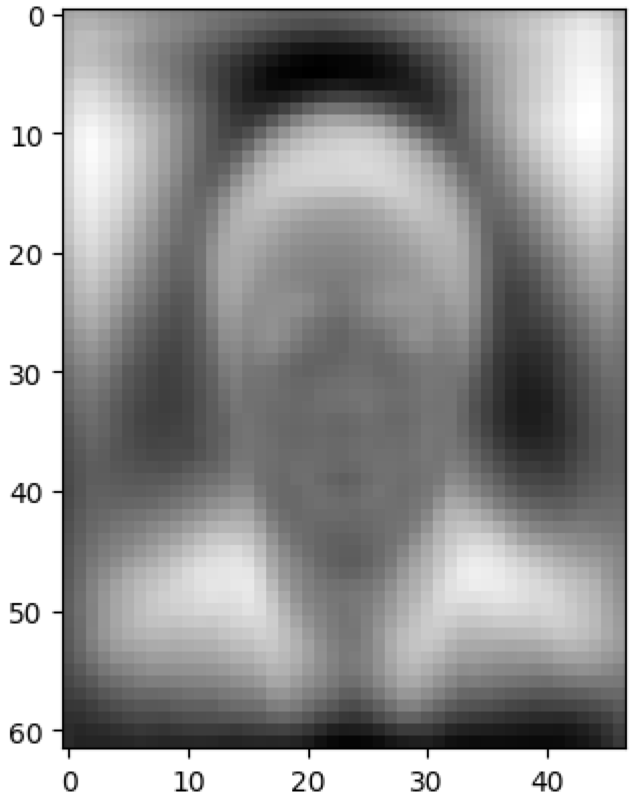

0
10
20
30
40
50
60
0
10
20
30
40

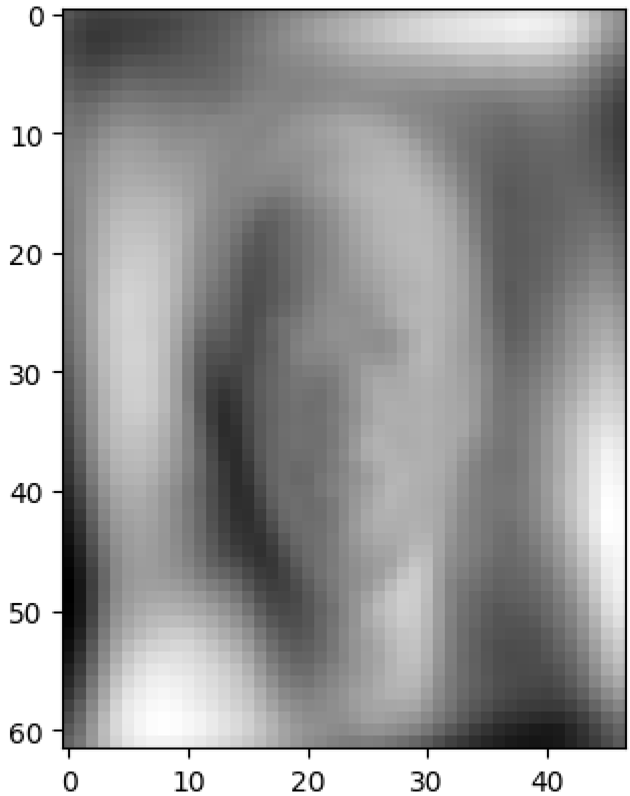

0
10
20
30
40
50
60
0
10
20
30
40

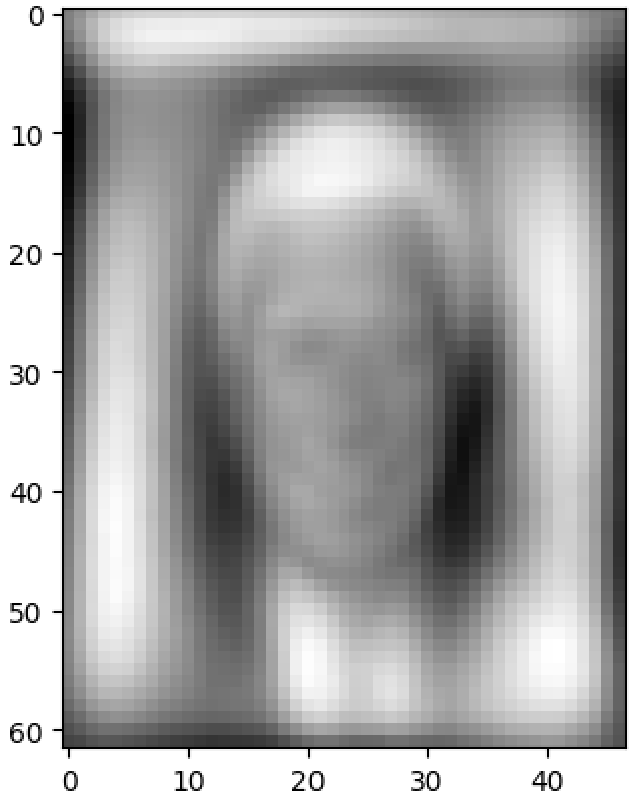

0
10
20
30
40
50
60
0
10
20
30
40

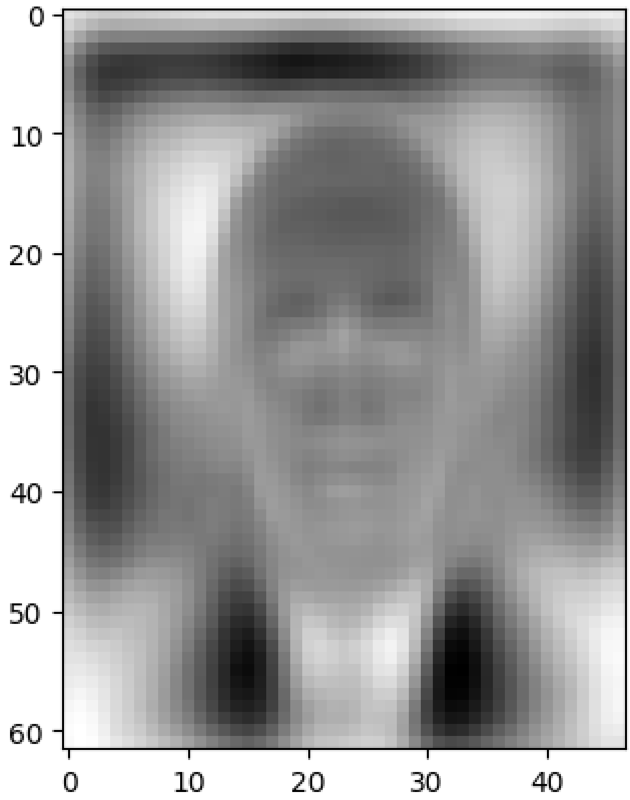

0
10
20
30
40
50
60
0
10
20
30
40

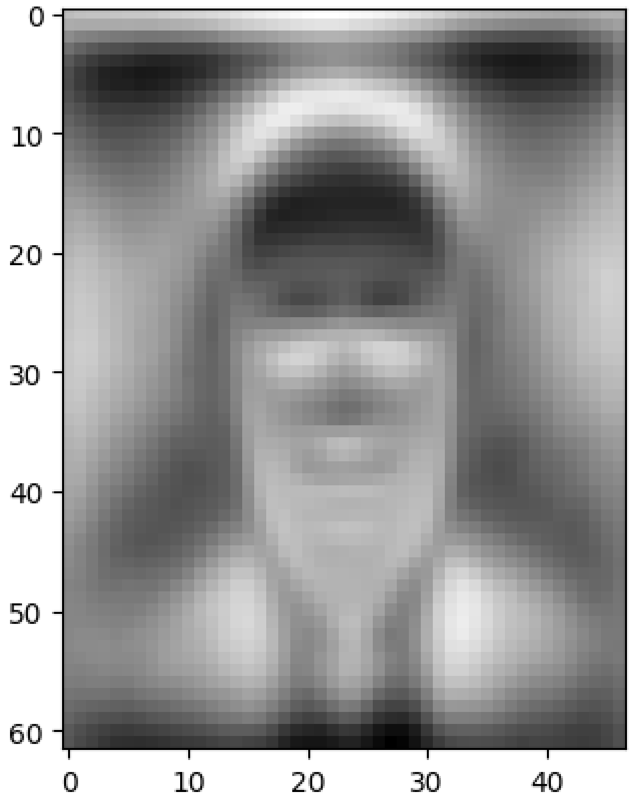

0
10
20
30
40
50
60
0
10
20
30
40


0
10
20
30
40
50
60
0
10
20
30
40


0
10
20
30
40
50
60
0
10
20
30
40


0
10
20
30
40
50
60
0
10
20
30
40





In [ ]:
```
display(Latex(r"\newpage"))
```

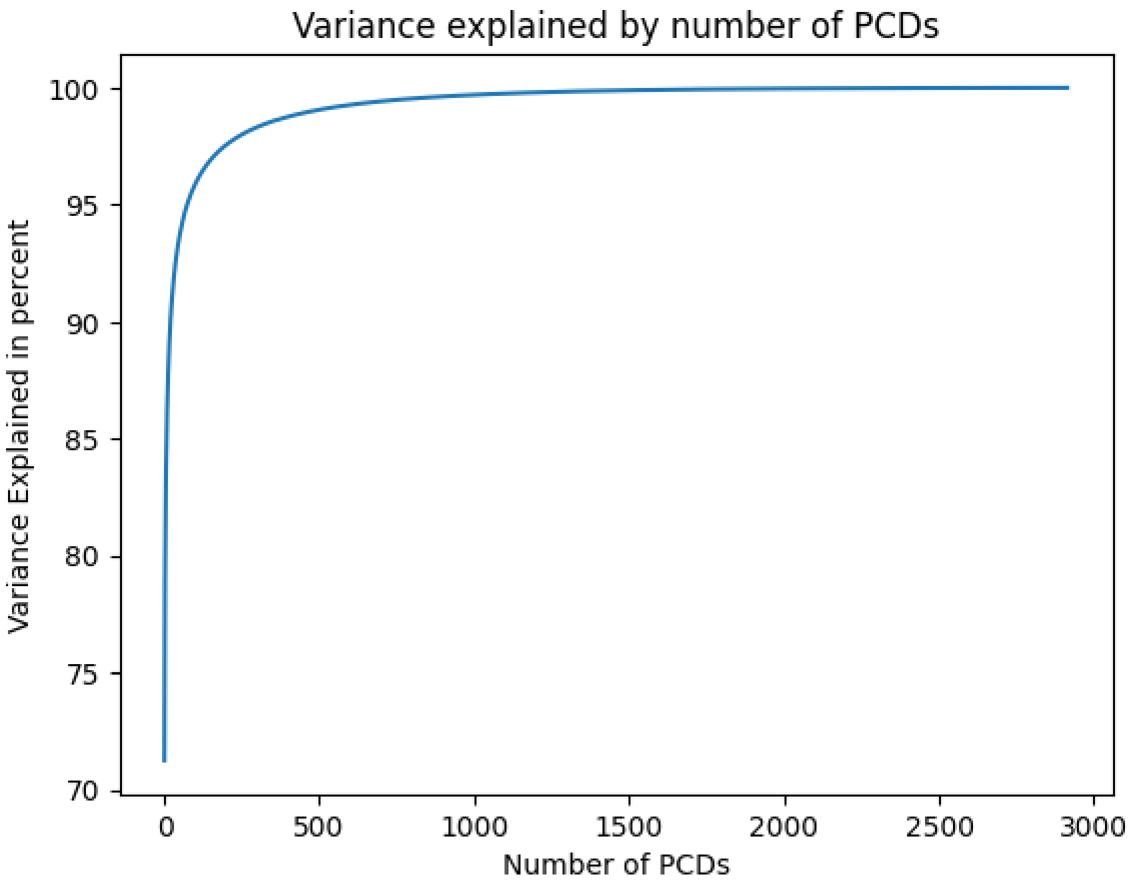
\newpage

In [ ]:
```
#1.e

explained_variances = []
for i in range(len(eig_values)):
    explained_variances.append(eig_values[i] / np.sum(eig_values)*100)
variance_explained_cum = np.cumsum(explained_variances)
```

```
k = 0
PCDs_needed=[]
for i in range(len(explained_variances)):
    k+=explained_variances[i]
    PCDs_needed.append(explained_variances[i])
    if k>95:
        break

print(f"One can see that we need {len(PCDs_needed)} PCDs to explain 95% of the vari
matplotlib.pyplot.plot(np.arange(1, X.shape[1] + 1, 1), variance_explained_cum)
matplotlib.pyplot.ylabel('Variance Explained in percent')
matplotlib.pyplot.xlabel('Number of PCDs')
matplotlib.pyplot.title('Variance explained by number of PCDs')
```

```
One can see that we need 75 PCDs to explain 95% of the variance.
```

Out[ ]:
```
Text(0.5, 1.0, 'Variance explained by number of PCDs')
```



In [ ]:
```
display(Latex(r"\newpage"))
```

\newpage

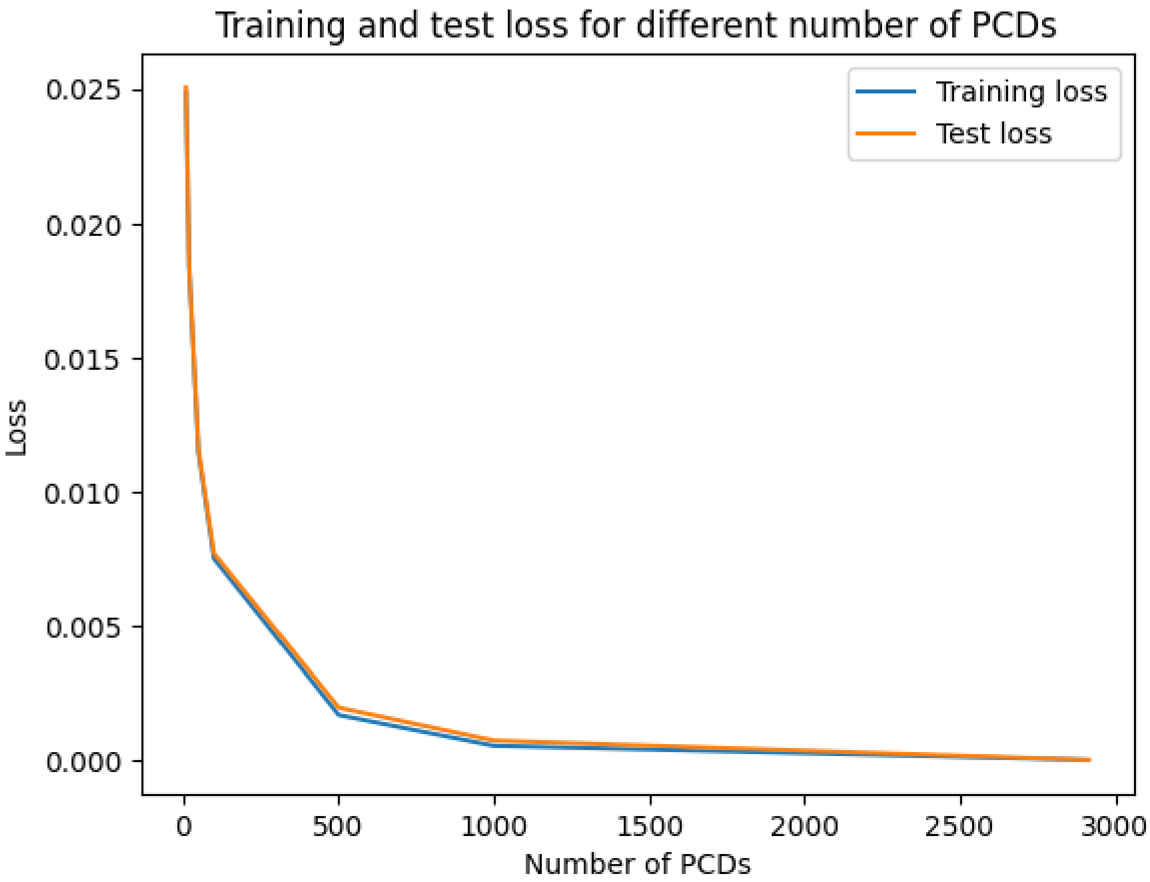
In [ ]:
```
#1.f
#load X again since our previous X was centered
X = dataset['data']

X_train = X[:int(X.shape[0]*0.8)]
X_test = X[int(X.shape[0]*0.8):]

eig_vectors, eig_values, _  = np.linalg.svd(X_train.T@X_train)

idx = eig_values.argsort()[::-1]
eig_values = eig_values[idx]
eig_vectors = eig_vectors[:, idx]
```

```
training_loss = []
test_loss = []

number_of_PCDs =  [10, 20, 50, 100, 500, 1000, 2914]

for k in number_of_PCDs:
    k_eig_vectors = np.zeros(eig_vectors.shape)
    k_eig_vectors[:,:k] = eig_vectors[:,:k]

    X_train_reconstruction = X_train @ k_eig_vectors @k_eig_vectors.T
    training_loss.append(1/(X_train.shape[0]*X_train.shape[1])*np.linalg.norm(X_tra

    X_test_reconstruction = X_test @ k_eig_vectors @k_eig_vectors.T
    test_loss.append(1/(X_test.shape[0]*X_test.shape[1])*np.linalg.norm(X_test_recc

    print(f"For {k} number of PCDs, the training loss was: {training_loss[-1]}, anc
```

```
For 10 number of PCDs, the training loss was: 0.024876154512220056, and the test l
oss was 0.025062690624474873

For 20 number of PCDs, the training loss was: 0.01832719678080922, and the test lo
ss was 0.018420726054348077

For 50 number of PCDs, the training loss was: 0.01145844974748309, and the test lo
ss was 0.011592239534264808

For 100 number of PCDs, the training loss was: 0.00749010035300871, and the test l
oss was 0.007688371896088376

For 500 number of PCDs, the training loss was: 0.0016752821280762095, and the test
loss was 0.001950320561979342

For 1000 number of PCDs, the training loss was: 0.0005270785648694966, and the tes
t loss was 0.0007230311720999202

For 2914 number of PCDs, the training loss was: 2.313802221591989e-16, and the tes
t loss was 2.8704234588938264e-16
```

In [ ]:

```
#1.f
#the plot
matplotlib.pyplot.plot(number_of_PCDs, training_loss, label = "Training loss")
matplotlib.pyplot.plot(number_of_PCDs, test_loss, label = "Test loss")
matplotlib.pyplot.title("Training and test loss for different number of PCDs")
matplotlib.pyplot.xlabel("Number of PCDs")
matplotlib.pyplot.ylabel("Loss")
matplotlib.pyplot.legend()
matplotlib.pyplot.show()
```


Training and test loss for different number of PCDs
Training loss
Test loss
0.025
0.020
0.015
0.010
0.005
0.000
Loss
0
500
1000
1500
2000
2500
3000
Number of PCDs

I worked with:

- Jens Kristoffersen
- Dherick Deravanan
- Olav Nomeland
- Sanket Behera

I certify that all solutions are entirely in my own words and that I have not looked at another student's solutioans. I have given credit to all external sources I consulted.

2.a

Building the heap: $O(k)$ where k is the size of the heap.

Remaining $n-k$ points take $O(\log k)$ time.

Finding the majority label of the k closest neighbours take $O(k)$ time.

Finding n euclidian distances take $n\,O(d)$ time.

Total: $O(k) + (n-k)\,O(\log k) + O(k) + n\,O(d) = O(nd) + O(k) + O(n \log k)$

2.b

Using the problem of placing balls in boxes, the number of monomials is $\binom{d+p}{p}$ and this is the dimension the point lives in.

d is now $\dfrac{(d+p)!}{(d+p-p)!\,p!} = \dfrac{(d+p)!}{d!\,p!}$.

New runtime:

$$O\left(n \cdot \frac{(d+p)!}{d!p!}\right) + O(k) + O(n \log k)$$

2.c

1D: left and right = 2

2D: all sides and all corners = 8

dD: There are $2 \cdot d$ adjacent cells and 2^d corners

$= 2d + 2^d$

You would need to check each one of these cells so time complexity is $O(2d + 2^d)$

3.a

I will be 0 surprised since I knew with 100% certainty what the outcome would be.

3.6.

Max surprised, my prior belief was that there was a 0% chance of picking up a white ball.

3.c

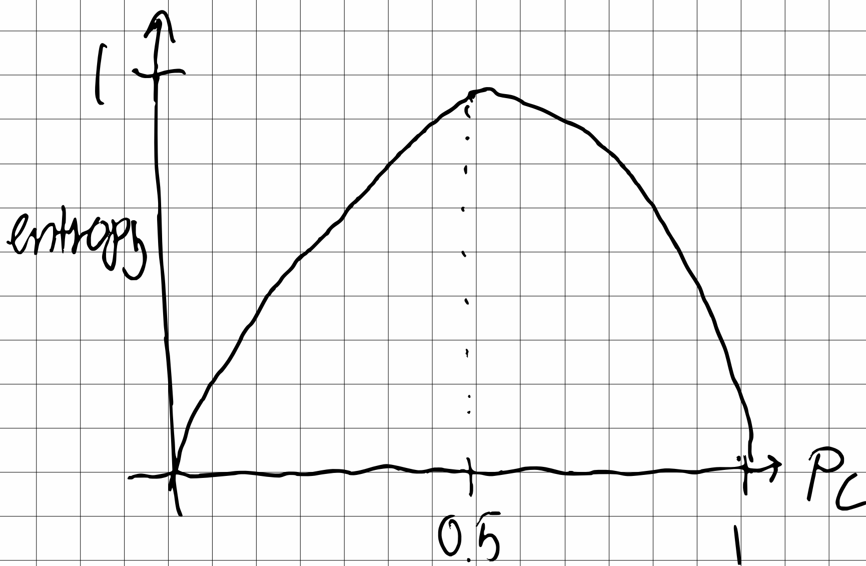
when P_B is either 0 or 1 is when the entropy is minimized.

$$H_b(0) = -0 \log 0 - 1 \log 1$$
$$= \underline{0}$$

Maximized at $P_B = 0.5$

$$Hb(0.5) = -0.5 \log 0.5 - 0.5 \log 0.5$$
$$= \underline{\underline{1}}$$

3. d



strictly concave

3. e.

$P = P(Y=1 \mid X_{j,v}=1)\,P(X_{j,v}=1)$
$\quad + P(Y=1 \mid X_{j,v}=0)\,P(X_{j,v}=0)$
$= q_1 P(X_{j,v}=1) + q_2 P(X_{j,v}=0)$
$= \lambda q_1 + (1-\lambda) q_2$ where
λ is the probability that
$X_j < v$.

Show that $H(Y) - H(Y \mid X_{j,v})$
is always positive if $q_1 \neq q_2$

$\rightarrow H_b(P(Y=1)) - \sum P(X_{jv}=i)\,H_b(P(Y=1 \mid X_{j,v}=i))$
> 0

$\rightarrow H_b(\lambda q_1 + (1-\lambda) q_2) > \lambda \cdot H_b(q_1) + (1-\lambda) H_b(q_2)$

$\rightarrow -(\lambda q_1 + (1-\lambda) q_2 \log(\lambda q_1 + (1-\lambda) q_2)$
$-(1-\lambda q_1 + (1-\lambda) q_2) \log(1-\lambda q_1 + (1-\lambda) q_2)) >$

$$\lambda \cdot (-q_1 \log q_1 - (1-q_1)\log(1-q_1))$$
$$+ (1-\lambda)\cdot(-q_2 \log q_2 - (1-q_2)\log(1-q_2))$$

→ sees that as H_b is strictly concave, multiplying with lamda on the outside of Hb function will lead to a smaller value.

I do not have a better explanation :/

3.f

Average:

$$E\left[\frac{1}{n}\sum_{i=1}^{n} Y_i\right] = \frac{1}{n}\sum_{i=1}^{n} E[Y_i]$$

$$= \frac{1}{n}\cdot\sum_{i=1}^{n} \mu = \frac{1}{n}\cdot n\mu = \mu$$

Variance:

$$Var\left(\frac{1}{n}\sum_{i=1}^{n} Y_i\right) = \frac{1}{n^2}\sum_{i=1}^{n} Var(Y_i)$$

$$= \frac{1}{n^2}\sum_{i=1}^{n} \sigma_i^2 = \frac{n\sigma^2}{n^2} = \frac{\sigma^2}{n}$$

3.9. i

Prob of not being picked:

$\left(\frac{n-1}{n}\right)^n$.

Lowest value of this function is at $n=25$ since it is monotonically increasing.

At $n=25$:

$\left(\frac{25-1}{25}\right)^{25} = 0.3604 > 0.36$

And at its limit:

$$\lim_{n\to\infty}\left(\frac{n-1}{n}\right)^n = \lim_{n\to\infty}\left(\frac{\cancel{n}\left(1-\frac{1}{n}\right)}{\cancel{n}}\right)^n$$

$$= \lim_{n\to\infty}\left(1-\frac{1}{n}\right)^n = \lim_{n\to\infty} e^{n\ln\left(1-\frac{1}{n}\right)}$$

$= \frac{1}{e}$ $\square$ $\rightarrow$ p lies in $[0.36, \frac{1}{e}]$

3.g.ii

More trees → more computation

Plot the loss at different nr. of trees in an increasing manner. and pick the number of trees when the loss levels sufficiently out towards its asymptote.

3.h

$$\mathrm{var}\left(\frac{1}{n}\sum_{i=1}^{n} z_i\right) = \frac{1}{n^2}\mathrm{var}\left[\sum_{i=1}^{n} z_i\right]$$

$$= \frac{1}{n^2}\left(\sum_{i=1}^{n}\mathrm{var}(z_i) + 2\sum_{i<j}^{n}\mathrm{cov}(z_i, z_j)\right)$$

$$= \frac{1}{n^2}\left(n\sigma^2 + 2\rho\frac{(n-1)n}{2}\right)$$

$$= \frac{\sigma^2}{n} + \frac{\rho(n-1)}{n} \quad \square$$

4

In [ ]:
```
from IPython.display import Latex
import numpy as np
import matplotlib.pyplot as plt
```

In [ ]:
```
#Import dataset
import sklearn
from sklearn import datasets
diabetes = sklearn.datasets.load_diabetes()
```

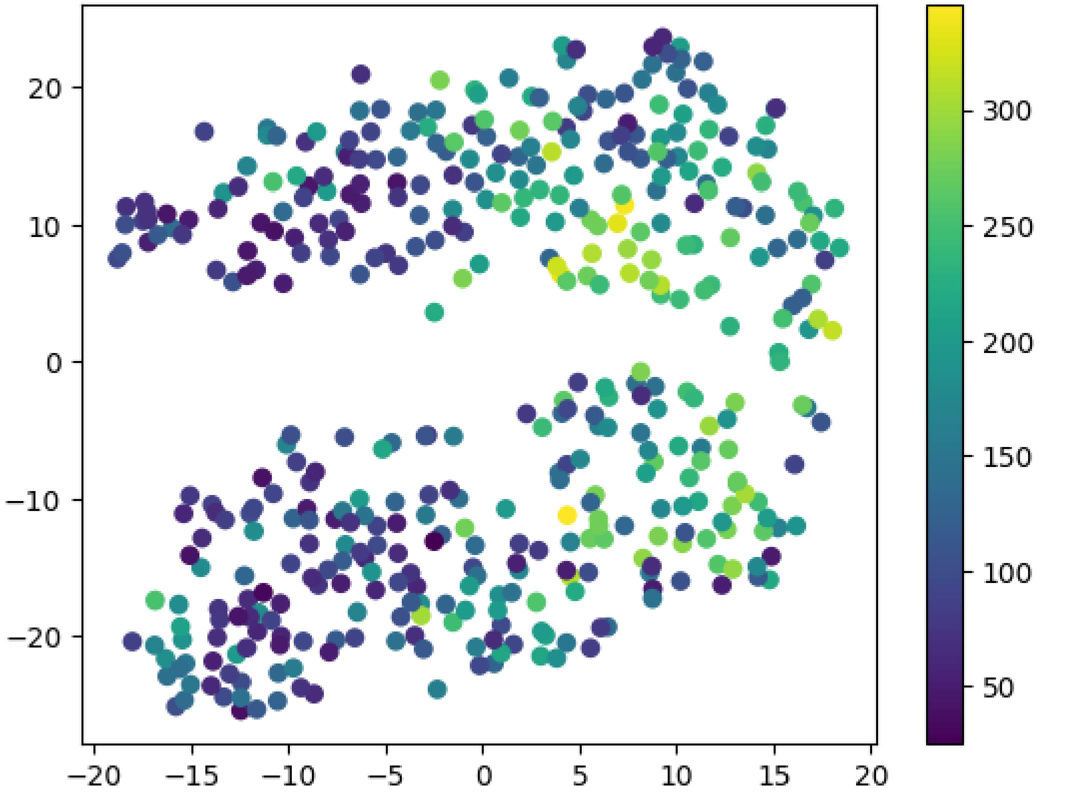
In [ ]:
```
display(Latex(r"\newpage"))
```

\newpage

In [ ]:
```
#4.a
from sklearn.manifold import TSNE
x = diabetes["data"]
y = diabetes["target"]
d = 2
tsne = TSNE(d)
tsne_result = tsne.fit_transform(x)

points = plt.scatter(tsne_result[:, 0], tsne_result[:,1], c = y, cmap = 'viridis')
plt.colorbar(points)
```

```
c:\Users\elias\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\m
anifold\_t_sne.py:800: FutureWarning: The default initialization in TSNE will chan
ge from 'random' to 'pca' in 1.2.
  warnings.warn(
c:\Users\elias\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\m
anifold\_t_sne.py:810: FutureWarning: The default learning rate in TSNE will chang
e from 200.0 to 'auto' in 1.2.
  warnings.warn(
```

Out[ ]:
```
<matplotlib.colorbar.Colorbar at 0x1cff409fca0>
```



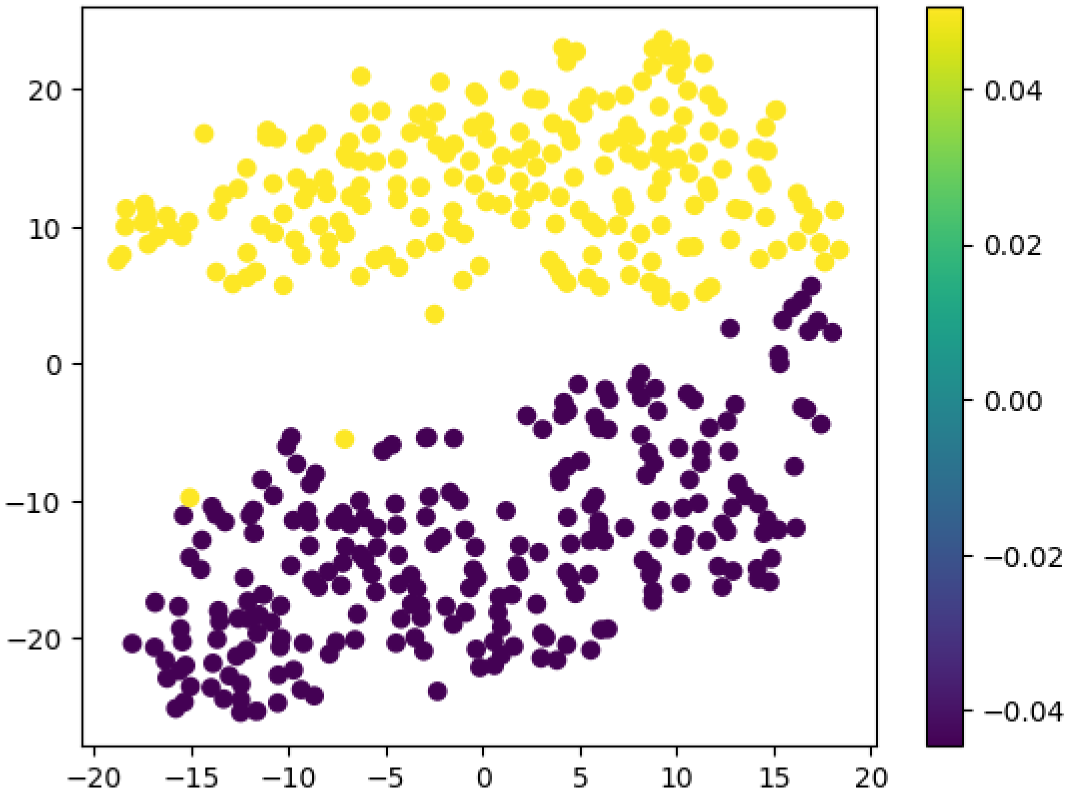
In [ ]:
```
display(Latex(r"\newpage"))
```

\newpage

4.b The feature is sex.

In [ ]:
```
points = plt.scatter(tsne_result[:, 0], tsne_result[:,1], c = x[:,1], cmap = 'virid
plt.colorbar(points)
```

Out[ ]: `<matplotlib.colorbar.Colorbar at 0x1cff3ed0460>`



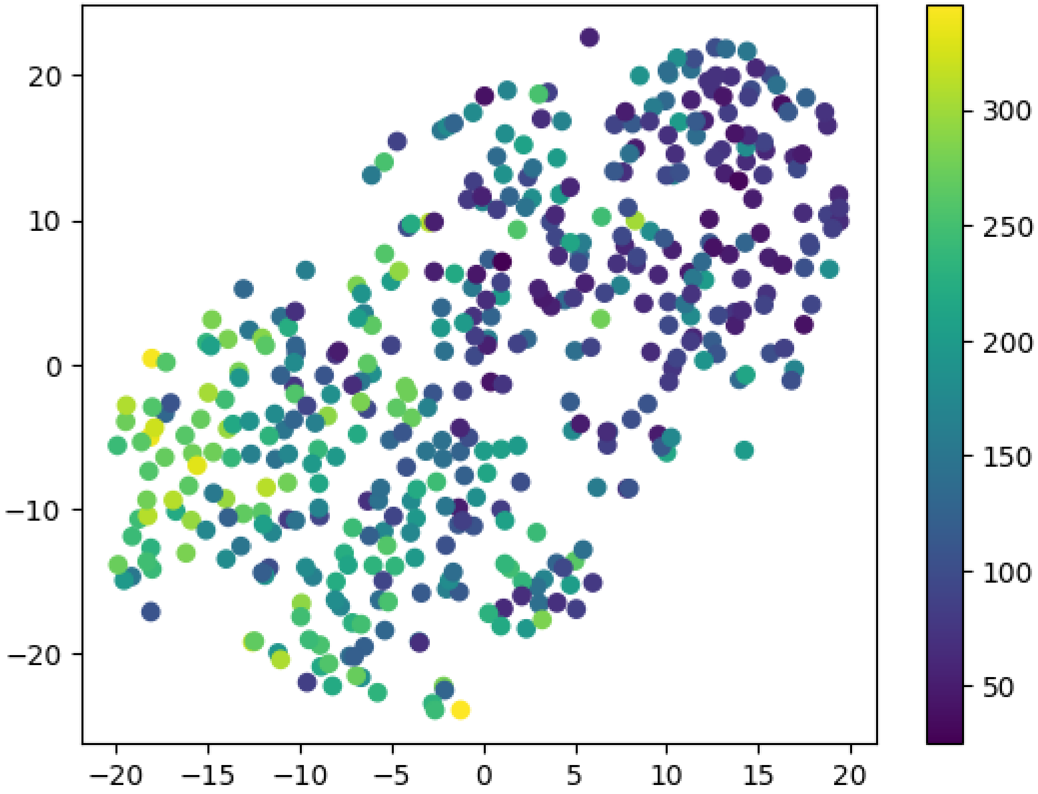
In [ ]:
```
display(Latex(r"\newpage"))
```

\newpage

In [ ]:
```
#4.c
new_x = np.delete(diabetes["data"], axis = 1, obj = 1)
tsne = TSNE(d)

new_result = tsne.fit_transform(new_x)

points = plt.scatter(new_result[:, 0], new_result[:,1], c = y, cmap = 'viridis')
plt.colorbar(points)
```

```
c:\Users\elias\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\m
anifold\_t_sne.py:800: FutureWarning: The default initialization in TSNE will chan
ge from 'random' to 'pca' in 1.2.
  warnings.warn(
c:\Users\elias\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\m
anifold\_t_sne.py:810: FutureWarning: The default learning rate in TSNE will chang
e from 200.0 to 'auto' in 1.2.
  warnings.warn(
```

Out[ ]: <matplotlib.colorbar.Colorbar at 0x1cff3f6ffd0>



We do not see any two clear clusters anymore since tsne does not differentiate on sex. In other words, most of the data points got a lot of their variance from sex meaning females and males are quite similar besides being of different genders. The fact that sex was included ended up with "pulling" the data apart and forming two clusters.

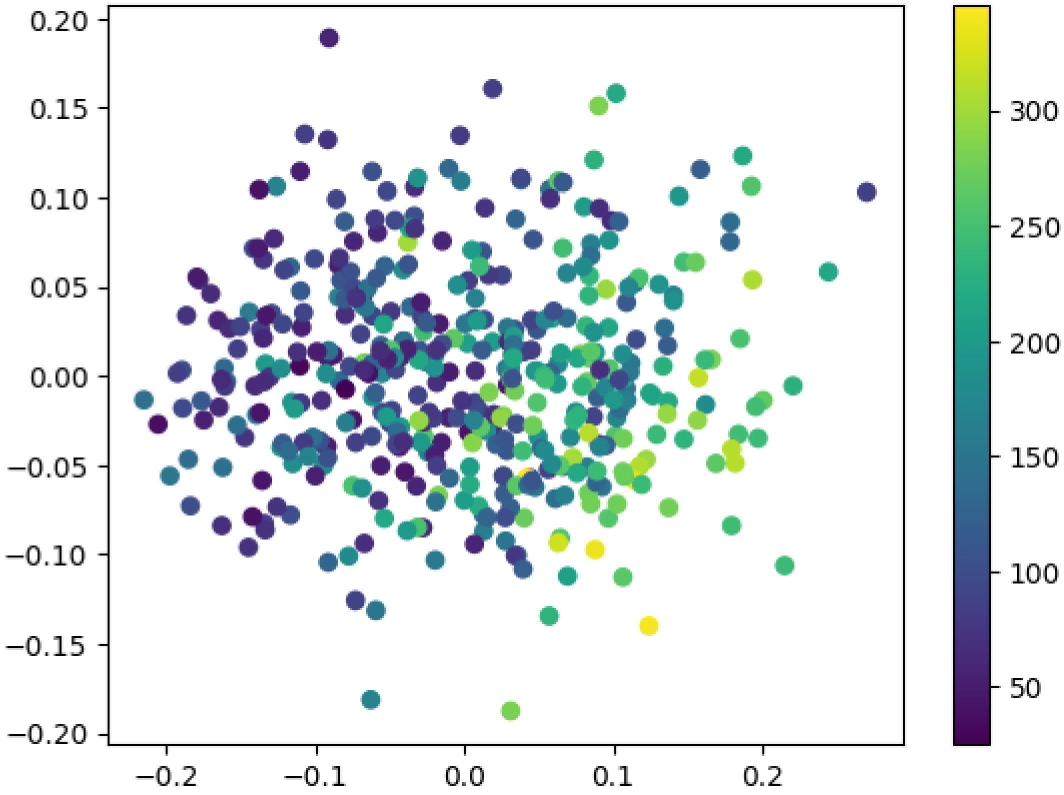
If the change did not occur and we did still see two clear clusters, it would mean that there would be another feature which was very binary in its distribution. If age was split in above 50 or below 50 this would also form two clusters.

In [ ]:
```
display(Latex(r"\newpage"))
```

\newpage

In [ ]:
```
#4.d
from sklearn.decomposition import PCA
pca = PCA(d)
pca_result = pca.fit_transform(x)
points = plt.scatter(pca_result[:, 0], pca_result[:,1], c = y, cmap = "viridis")
plt.colorbar(points)
```

Out[ ]: `<matplotlib.colorbar.Colorbar at 0x1cff42b4220>`



In [ ]:
```
display(Latex(r"\newpage"))
```

\newpage

In [ ]:
```
#4.e
def MSE(y_hat, y_true):
    return (1/y_hat.shape[0])*np.sum((y_hat - y_true)**2)

X_train = x[:100]
y_train = y[:100]
X_test = x[100:]
y_test = y[100:]

X_train_mtx = np.hstack([X_train, np.ones((100,1))])
X_test_mtx = np.hstack([X_test, np.ones((342, 1))])
w_ols = np.linalg.inv(X_train_mtx.T @ X_train_mtx) @ X_train_mtx.T @ y_train
y_hat = X_test_mtx@w_ols

test_mse = MSE(y_hat, y_test)
print(f"The test MSE is: {test_mse}")

def c_index(y_hat, y_test):
    nr_conc = 0
    nr_disc = 0
```

```
    for i in range(y_hat.shape[0]):
        for j in range(y_test.shape[0]):
            if i == j:
                continue
            else:
                y_test_i = y_test[i]
                y_test_j = y_test[j]
                y_hat_i = y_hat[i]
                y_hat_j = y_hat[j]
                if y_test_i > y_test_j and y_hat_i > y_hat_j:
                    nr_conc += 1
                elif y_test_i > y_test_j and y_hat_i < y_hat_j:
                    nr_disc += 1
    return nr_conc/(nr_conc+nr_disc)

print(f"The c index is: {c_index(y_hat, y_test)}")
```

```
The test MSE is: 3430.9233826005243
The c index is: 0.7452930850514576
```

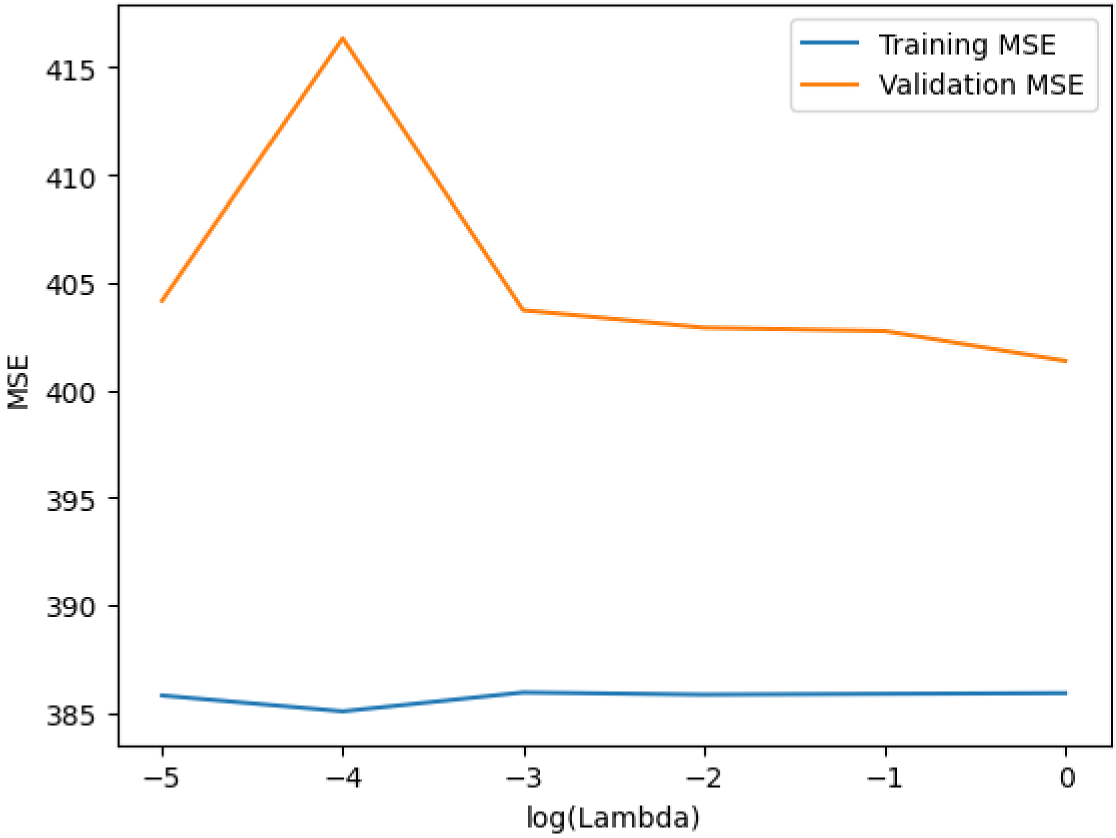
In [ ]:
```
display(Latex(r"\newpage"))
```

\newpage

In [ ]:
```
#4.f
from sklearn.linear_model import Ridge
def cross_validation(X, y, k, model, loss_calculator):
    n = X.shape[0]
    idx = np.random.permutation(n)
    X_shuffled = X[idx]
    y_shuffled = y[idx]

    size = n // k

    validation_loss = np.zeros((n,))
    training_loss = np.zeros((n,))

    for i in range(k):
        start = i*size
        end = (i+1)*size

        X_test = X_shuffled[start:end]
        y_test = y_shuffled[start:end]

        X_train = np.vstack([X_shuffled[:start], X_shuffled[end:]])
        y_train = np.hstack([y_shuffled[:start], y_shuffled[end:]])

        model.fit(X_train, y_train)
        training_loss[i] = loss_calculator(model.predict(X_train), y_train)
        validation_loss[i] = loss_calculator(model.predict(X_test), y_test)

    return np.mean(validation_loss), np.mean(training_loss)



lambdas =  [10**-5, 10**-4, 10**-3, 10**-2, 10**-1, 1]
validation_MSE = []
training_MSE = []

for l in lambdas:
```

```
    validation_MSE_current, training_MSE_current = cross_validation(X_train, y_trai
    validation_MSE.append(validation_MSE_current)
    training_MSE.append(training_MSE_current)

    print(f"Lambda={l} has validation loss={validation_MSE_current} and training lo
    print("")

plt.plot(np.log10(lambdas), training_MSE, label='Training MSE')
plt.plot(np.log10(lambdas), validation_MSE, label='Validation MSE')
plt.legend()
plt.xlabel('log(Lambda)')
plt.ylabel('MSE')

best_lambda = lambdas[np.argmin(validation_MSE)]
ridge_best_lambda = Ridge(alpha=best_lambda).fit(X_train,y_train)
ridge_y_hat = ridge_best_lambda.predict(X_test)
ridge_mse = MSE(ridge_y_hat, y_test)
print(f"The best lambda was {best_lambda}.\nThis lambda gave MSE with ridge = {ridg
```

```
Lambda=1e-05 has validation loss=404.15799243503966 and training loss=385.80102276
90261

Lambda=0.0001 has validation loss=416.3534904067575 and training loss=385.06213726
177435

Lambda=0.001 has validation loss=403.72113028383114 and training loss=385.94238469
552073

Lambda=0.01 has validation loss=402.9057175061709 and training loss=385.8414729654
401

Lambda=0.1 has validation loss=402.7489796064022 and training loss=385.87803508971
024

Lambda=1 has validation loss=401.3585709192019 and training loss=385.9097279767526

The best lambda was 1.
This lambda gave MSE with ridge = 5039.062537574326.
```


Training MSE
Validation MSE
415
410
405
400
395
390
385
MSE
−5
−4
−3
−2
−1
0
log(Lambda)

```
In [ ]:
```