Lab 10: Principal Component Analysis

In [ ]:
```
import numpy as np
from PIL import Image as im
from matplotlib import pyplot as plt
import os
from IPython.display import display
import scipy.sparse.linalg
```

Load the faces dataset.

In [ ]:
```
DATASET_PATH = "Lab 10.zip" # TO DO
# DATASET_PATH = "Lab 10.zip" # TO DO
archive = np.load(DATASET_PATH)
faces = archive["faces.npy"]
mu = archive["mu.npy"]
sigma = archive["sigma.npy"]
```

In [ ]:
```
SAVE_PATH = "./outputs"
```

Build useful classes.

In [ ]:
```
import scipy.sparse.linalg

class K_eigenvectors():

    def __init__(self, X, K):

        self.X = X
        self.K = K
        self.V = None

    def solve(self):

        L, V = scipy.sparse.linalg.eigs(self.X, k=self.K)
        self.V = V[:, :self.K]
        return self.V
    def is_unitary(self):
        if self.V is None:
            raise ValueError("Run solve() first to compute eigenvectors.")
        identity = np.eye(self.K)
        product = np.conj(self.V.T) @ self.V
        return np.allclose(product, identity)

class Eigendecomposition():

    def __init__(self, face, mu, eigenfaces):

        self.x = np.ravel(face, 'F')
        self.mu = np.ravel(mu, 'F')
        self.V = eigenfaces
```

```
    def solve(self):

        return np.matmul(np.conj(np.transpose(self.V)),self.x-self.mu)

class Reconstruction():

    def __init__(self, mu, eigenfaces, coefficients):

        self.mu = np.ravel(mu, 'F')
        self.K = eigenfaces.shape[1]
        self.V = eigenfaces
        self.w = coefficients

    def solve(self):

        return np.transpose((np.abs(np.matmul(self.V,self.w) + self.mu)).reshape
```

1. PCA: Decomposition

1.1 Decomposition in principal components

In [ ]:
```
# check that matrix is unitary
K = 10
solver = K_eigenvectors(sigma, K)
eigenfaces = solver.solve()

print("Is unitary:", solver.is_unitary())
```

```
Is unitary: True
```

1.2 Decomposition of a face in the space of eigenfaces

In [ ]:
```
face = faces[:, :, 0]
decomposer = Eigendecomposition(face, mu, eigenfaces)
coefficients = decomposer.solve()

# === Print results ===
print(f"Coefficients for top {K} principal components:\n", coefficients)

# Optional: visualize mean face and first eigenface
plt.figure(figsize=(8, 4))
plt.subplot(1, 2, 1)
plt.title("Mean Face")
plt.imshow(mu, cmap='gray')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.title("First Eigenface")
plt.imshow(np.real(eigenfaces[:, 0]).reshape(112, 92, order='F'), cmap='gray')
plt.axis('off')

plt.tight_layout()
plt.show()
```

```
Coefficients for top 10 principal components:
 [-1646.55201641+0.j  1307.35365512+0.j  1888.54048426+0.j
  -548.41445491+0.j   600.7535633 +0.j  -377.13628784+0.j
  -578.06968564+0.j   821.48541641+0.j  -263.13028016+0.j
 -1170.30836116+0.j]
```

Mean Face

First Eigenface

2. Reconstruction

2.1 Reconstruction of a face using K principal components & 2.2 Reconstruction error

In [ ]:

```
K_list = [1, 2, 5, 10, 25, 50] # number of principal components
face_idx = 20

error_reduction_list = []
for K in K_list:

    eig_obj = K_eigenvectors(sigma, K) # TO DO
    K_eig = eig_obj.solve()

    eigdec_obj = Eigendecomposition(faces[:,:,face_idx], mu, K_eig) # TO DO
    coefficients = eigdec_obj.solve()
    print('The coefficients of the ', K, ' principal components are:', coefficie

    recon_obj = Reconstruction(mu, K_eig, coefficients) # TO DO
    recon_face = recon_obj.solve()

    og_face = faces[:,:,face_idx]

    image = im.fromarray(og_face.astype(np.uint8), mode='L')
    display(image)
    #image.save(f"{SAVE_PATH}/original.png")

    recon_image = im.fromarray(recon_face.astype(np.uint8), mode='L')
    display(recon_image)
    #recon_image.save(f"/content/{SAVE_PATH}/recon_K_{K}.png")
```

```
    error_energy = np.linalg.norm(og_face - recon_face) ** 2 # TO DO: Find the e
    error_K_0 = np.linalg.norm(og_face - mu) ** 2 # TO DO: Find the energy of th
    print('Error energy: ', error_energy)
    print('Error energy for K = 0: ', error_K_0)
    print('Error reduction: ', (error_K_0-error_energy)/error_K_0)

    error_reduction_percent = 100 * (error_K_0 - error_energy) / error_K_0 # TO
    error_reduction_list.append(error_reduction_percent)

fig, axs = plt.subplots()
axs.plot(K_list, error_reduction_list, '-s')
axs.set_xlabel('K')
axs.set_ylabel('Error reduction (%)')
axs.hlines(y = 60, xmin=min(K_list), xmax=max(K_list), linestyle = '--', color =
axs.grid(True)
#plt.savefig(f"{SAVE_PATH}/error_reduction.png", dpi = 300)
plt.show()
plt.close(fig)
```

```
The coefficients of the  1  principal components are: [-1917.20199986+0.j]
```

```
Error energy:  6347121.581103278
Error energy for K = 0:  10022785.089375002
Error reduction:  0.36673075153215023
The coefficients of the  2  principal components are: [-1917.20199986+0.j  -106.1
4949791+0.j]
```

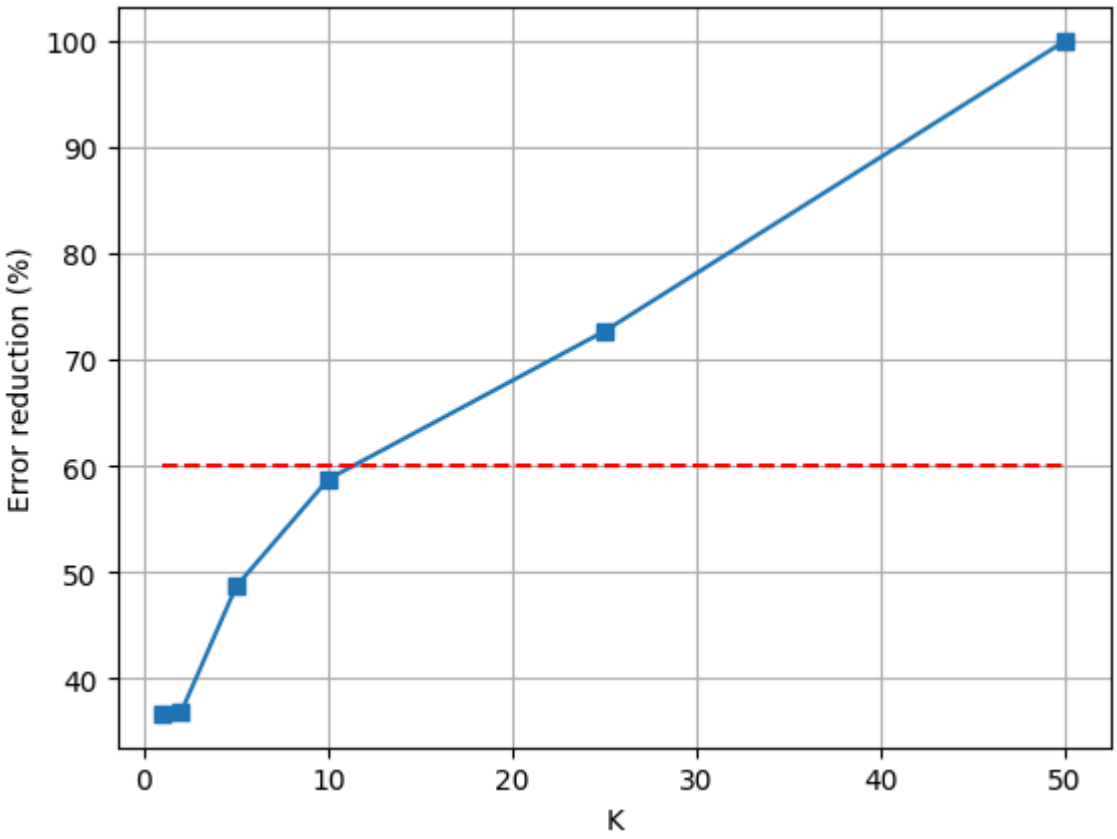
```
Error energy:  6335853.86519642
Error energy for K = 0:  10022785.089375002
Error reduction:  0.3678549616001485
The coefficients of the  5  principal components are: [1917.20199986+0.j -106.149
49791+0.j  291.25158972+0.j 1034.52646364+0.j
 -185.66734993+0.j]
```


```
Error energy:  5146309.007875408
Error energy for K = 0:  10022785.089375002
Error reduction:  0.4865390246338885
The coefficients of the  10  principal components are: [-1917.20199986+0.j   106.
14949791+0.j  -291.25158972+0.j
 -1034.52646364+0.j   185.66734993+0.j    -2.94147084+0.j
   648.8307638 +0.j   299.87449754+0.j  -490.77873374+0.j
   510.2305514 +0.j]
```


```
Error energy:  4134195.3002255377
Error energy for K = 0:  10022785.089375002
Error reduction:  0.5875203086407457
The coefficients of the  25  principal components are: [1917.20199986+0.j -106.14
949791+0.j -291.25158972+0.j 1034.52646364+0.j
  185.66734993+0.j    2.94147084+0.j -648.8307638 +0.j  490.77873374+0.j
  299.87449754+0.j -510.2305514 +0.j -467.52796913+0.j  -20.46434852+0.j
  444.77503753+0.j -264.55116393+0.j  191.8209542 +0.j -245.24374973+0.j
 -465.1702159 +0.j  258.76446606+0.j -503.40083191+0.j  -36.73798832+0.j
 -317.10670166+0.j  -47.06444165+0.j   -7.92176534+0.j -375.6914697 +0.j
 -174.72380386+0.j]
```


```
Error energy:  2737830.7608277733
Error energy for K = 0:  10022785.089375002
Error reduction:  0.7268393229612291
The coefficients of the  50  principal components are: [ 1.91720200e+03+0.j -1.06
149498e+02+0.j -2.91251590e+02+0.j
  1.03452646e+03+0.j  1.85667350e+02+0.j -2.94147084e+00+0.j
 -6.48830764e+02+0.j -4.90778734e+02+0.j -2.99874498e+02+0.j
  5.10230551e+02+0.j  4.67527969e+02+0.j -2.04643485e+01+0.j
  4.44775038e+02+0.j  2.64551164e+02+0.j -1.91820954e+02+0.j
  2.45243750e+02+0.j -4.65170216e+02+0.j -2.58764466e+02+0.j
 -5.03400832e+02+0.j  3.67379883e+01+0.j -3.17106702e+02+0.j
  4.70644416e+01+0.j -7.92176534e+00+0.j  3.75691470e+02+0.j
 -1.74723804e+02+0.j  4.49655897e+02+0.j  1.92084608e+02+0.j
 -1.12563911e+02+0.j  8.72369676e+02+0.j -2.73801266e+02+0.j
 -7.15289240e+02+0.j  7.85791844e+01+0.j  5.73930240e+01+0.j
  1.55352792e+02+0.j  4.52196020e+02+0.j -3.23389767e+01+0.j
  8.75287954e+02+0.j -3.55108167e+02+0.j  8.41280155e+01+0.j
  1.75193193e-13+0.j  2.35367281e-14+0.j -3.09752224e-14+0.j
 -2.22155627e-13+0.j -1.96287431e-13+0.j -3.83304499e-14+0.j
 -1.72084569e-13+0.j -6.10622664e-15+0.j  6.21724894e-14+0.j
 -1.55209179e-13+0.j -3.99680289e-14+0.j]
```

```
Error energy:  7.036358313871423e-22
Error energy for K = 0:  10022785.089375002
Error reduction:  1.0
```



Question: How many principal components are needed to obtain a 60% reduction in the reconstruction error (as compared to K = 0)?

Answer: With 25 principal components, you can pretty much obtain a 60% reduction in the reconstruction error