Szeregi czasowe - Eigenfaces

Sprawozdanie z laboratorium

Wykonali:
Paweł Suchanicz
Elżbieta Dziedzic

In [438]:
```
from matplotlib import pyplot as plt
from matplotlib.image import imread
import numpy as np
import os
import math
```

In [439]:
```
images_path = 'images/'
images_files = os.listdir(images_path)
```

In [440]:
```
# zamiana na rgb
def rgb_to_grayscale(rgb):
    return np.dot(rgb[...,:3], [0.2989, 0.5870, 0.1140])
```

In [441]:
```
# wyświetlenie obrazków
def display_images(images):
    fig=plt.figure()
    for i in range(images.shape[0]):
        img = images[i].reshape(height,width)
        plt.subplot(math.ceil(images_count / 5), 5, i+1)
        plt.imshow(img, cmap='gray')
        plt.subplots_adjust(right=1.2, top=1.2)
    plt.show()
```

Wczytanie obrazków i zamian na skalę szarości

In [442]:
```
width  = 250
height = 250
images_count = len(images_files)
images = np.zeros((images_count, height*width))
for i in range(images_count):
    img = rgb_to_grayscale(imread(images_path + images_files[i]))
    images[i] = np.array(img.flatten('C'), dtype='float64').flatten()

display_images(images)
```

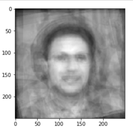
Wyliczenie średniego obrazu

In [443]:
```
mean_face = np.zeros((1, height * width))

for image in images:
    mean_face = mean_face + image

mean_face = mean_face / images.shape[0]

plt.imshow(mean_face.reshape(height, width), cmap='gray')
plt.show()
```



Znormalizowanie pozostałych obrazków (odjęcie średniego obrazu)

In [444]:
```
normalised_images = np.ndarray(shape=images.shape)

for i in range(normalised_images.shape[0]):
    normalised_images[i] = images[i] - mean_face

display_images(normalised_images)
```

In [445]:
```
covariance_matrix = np.dot(normalised_images.transpose(), normalised_images)
```

```
---------------------------------------------------------------------------
MemoryError                               Traceback (most recent call last)
<ipython-input-445-ca0165397739> in <module>
----> 1 covariance_matrix = np.dot(normalised_images.transpose(), normalised_images)

MemoryError:
```

Nie można obliczyć macierzy kowariancji z powodu zbyt dużej ilości danych (brak pamięci) W związku z tym stosujemy trik - liczymy macierz kowariancji pomiędzy kolumnami a nie wierszami

In [446]:
```
covariance_matrix = np.dot(normalised_images, normalised_images.transpose())
covariance_matrix = np.divide(covariance_matrix, normalised_images.shape[0])
```

In [447]:
```
eigenvalues, transposed_matrix_eigenvectors, = np.linalg.eig(covariance_matrix)
print('Eigenvectors of transposed matrix:\n%s' %eigenvectors)
print('\nEigenvalues: \n%s' %eigenvalues)
```

```
Eigenvectors of transposed matrix:
[[-3.11.97511285  -32.94120319   49.1109235  ...   15.49109135
   -29.33949667   25.93421632]
 [-112.30797553  -32.08365329   48.96574392 ...   15.71772419
   -28.49034217   26.29935402]
 [-113.11193858  -34.08226125   50.10978551 ...   17.07642644
   -29.23084893   25.25334525]
 ...
 [ -15.67313926  -29.51179215  -35.5372352  ...  -77.8644001
   -46.09169502   65.17356417]
 [ -13.18100279  -31.8447276   -33.89418943 ...  -68.24820984
   -30.59360824   63.27346216]
 [ -12.7600897   -36.32212797  -29.03618538 ...  -57.99612537
   -33.60516971   62.53093087]]

Eigenvalues:
[ 6.70593608e+07  3.54027347e+07  2.34230326e+07  1.79676495e+07
  1.35607321e+07  1.16049661e+07 -2.54437061e-09  1.05940529e+07
  9.56094452e+06  8.79125965e+06  8.21789014e+06  6.42831180e+06
  6.03446185e+06  2.45460441e+06  2.91945277e+06  4.71424283e+06
  4.41057312e+06  4.06930032e+06  3.61560786e+06  3.80893246e+06]
```

Wartości własne są takie same dla macierzy korelacji *po zastosowaniu* triku i przed. Różnica jest w wektorach własnych. Aby uzyskać wektory własne dla właściwej macierzy należy pomnożyć macierz z obrazkami przez macierz z wyliczonymi wektorami własnymi dla zastępczej macierzy.

In [448]:
```
eigenvectors = np.dot(normalised_images.transpose(), transposed_matrix_eigenvectors)
```

Posortowanie wartości własnych (oraz stowarzyszonych z nimi wektorów własnych) od najwiekszej wartości własnej do najmniejszej

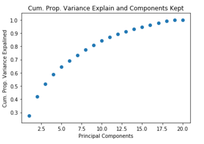
In [449]:
```
eig_pairs = [(eigenvalues[index], eigenvectors[:,index]) for index in range(len(eigenvalues))]
eig_pairs.sort(reverse=True)
eigvalues_sort  = [eig_pairs[index][0] for index in range(len(eigenvalues))]
eigvectors_sort = [eig_pairs[index][1] for index in range(len(eigenvalues))]
```

Zatrzymanie tylko tych wektorów własnych dla których suma wartości sięga 90% sumy wszystkich wartości własnych.

In [450]:
```
var_comp_sum = np.cumsum(eigvalues_sort)/sum(eigvalues_sort)

num_comp = range(1,len(eigvalues_sort)+1)
plt.title('Cum. Prop. Variance Explain and Components Kept')
plt.xlabel('Principal Components')
plt.ylabel('Cum. Prop. Variance Explained')

plt.scatter(num_comp, var_comp_sum)
plt.show()
```

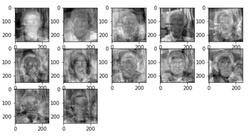


In [451]:
```
def reduce_data(eigvalues_sorted, eigvectors_sorted):
    for index, cumsum in enumerate(np.cumsum(eigvalues_sorted)/sum(eigvalues_sorted)):
        if(cumsum) >= 0.9:
            return np.array(eigvalues_sorted[:index]), np.array(eigvectors_sorted[:index])
```

In [452]:
```
reduced_eigenvalues, reduced_eigenvectors = reduce_data(eigvalues_sort, eigvectors_sort)
```

Transformacja wektorów własnych na macierz oraz jej wizualizacja

In [453]:
```
display_images(reduced_eigenvectors)
```



Zrzutowanie każdego obrazu na wektory własne

In [454]:
```
proj_data = np.dot(normalised_images, reduced_eigenvectors.transpose())
imgs_weights = np.zeros((normalised_images.shape[0], reduced_eigenvectors.shape[0]))
for i in range(normalised_images.shape[0]):
    reduced_eigenvectors = reduced_eigenvectors / np.linalg.norm(reduced_eigenvectors, axis=1)[:, np.newaxis]
    proj = normalised_images[i] * reduced_eigenvectors
    coef = [sum(x) for x in proj]
    imgs_weights[i] = coef
```

Rekonstrukcja obrazu przy pomocy nowego układu współrzędnych: obraz zrekonstruowany = rzut na dany wektor własny * wektor własny.

In [455]:
```
def reconstruct(eigenvectors, weights):
    img = np.zeros((1, height * width))
    for i in range(len(eigenvectors)):
        img += eigenvectors[i] * weights[i]
    return img

def reconstruct_images(eigenvectors, imgs_weights):
    return np.array([reconstruct(reduced_eigenvectors, weight) for weight in imgs_weights])
```

Wykonanie rekonstrukcji i wyświetlenie wynikowych obrazów

In [456]:
```
reconstructed_images = reconstruct_images(reduced_eigenvectors, imgs_weights)
display_images(reconstructed_images)
```