Machine Learning Programming
Assignment 1: Principal Component Analysis

Professor: Aude Billard
Assistants: Nadia Figueroa, Laila El Hamamsy,
Hala Khodr and Lukas Huber
Contacts:
aude.billard@epfl.ch, nadia.figueroafernandez@epfl.ch
laila.elhamamsy@epfl.ch, hala.khodr@epfl.ch, lukas.huber@epfl.ch

Winter Semester 2017

Introduction

This series of practicals focus on the development of machine learning algorithms introduced in the Applied Machine Learning course. In this first practical, you will code the Principal Component Analysis (PCA) algorithm in Matlab. You will then use the code to reduce the dimensionality of a toy 2D dataset, a real-world 9d dataset and an image dataset.

Submission Instructions

Deadline: October 17, 2017 @ 6pm. Assignments must be turned in by the deadline. `1pt` will be removed for each day late (a day late starts one hour after the deadline).
Procedure: From the course Moodle webpage, the student should download and extract the .zip file named `TP1-PCA-Assignment.zip` which contains the following files:

Part 1 - Algorithm	Part 2 - Applications
`my_pca.m`	`plot_eigenvalues.m`
`project_pca.m`	`explained_variance.m`
`reconstruct_pca.m`	`test_pca_9d.m`
`reconstruction_error.m`	`plot_eigenfaces.m`
`test_pca_2d.m`	`reconstruction_eigenfaces.m`
`test_mypca.m`	`test_pca_eigenfaces.m`

As well as `TP1-PCA-Dataset.zip` which contains the datasets required to test your functions.

Assignment Instructions

The assignment consists of implementing the blue colored MATLAB functions from scratch. These functions can be tested with the `test_*.m` scripts. These testing scripts depend on `ML_toolbox`, which must be downloaded from: `https://github.com/epfl-lasa/ML_toolbox`. Before proceeding make sure that all the sub-directories of the ML_toolbox have been added to your MATLAB search path. This can be done as follows in the MATLAB command window:

```
>> addpath(genpath('path_to_ML_toolbox'))
```

Once you have tested your functions, you can submit them as a .zip file with the name: `TP1-PCA-Assignment-SCIPER.zip` on the submission link in the Moodle webpage. Your submission archive should contain ONLY the following:

```
🟥 TP1-PCA-Assignment-SCIPER
 └─🟦 Part1
 └─🟦 Part2
```

DO NOT upload `ML_toolbox`, the `check_utils` directory or the `Datasets` directory.

1 Part 1: Implementing Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a well-known dimensionality reduction technique which projects the data onto a new basis of smaller dimensions. It is used as:

- Pre-processing step for clustering or classification algorithms to reduce the dimensionality and computational costs.
- Compression for data storage and retrieval.
- Feature extraction.

Given a training dataset $\mathbf{X} \in \mathbb{R}^{N\times M}$ composed of M datapoints with N-dimensions each; i.e. $\mathbf{X} = \{\mathbf{x}^1, \mathbf{x}^2, ..., \mathbf{x}^M\}$ where $\mathbf{x}^i \in \mathbb{R}^N$ we would like to find a lower-dimensional projection $\mathbf{Y} \in \mathbb{R}^{p\times M}$ (i.e. $\mathbf{Y} = \{\mathbf{y}^1, \mathbf{y}^2, , ..., \mathbf{y}^M\}$ where $\mathbf{y}^i \in \mathbb{R}^p$) of $\mathbf{X}$ through a linear map $A : \mathbf{x} \in \mathbb{R}^N \to \mathbf{y} \in \mathbb{R}^{p\leq N}$ given by

$$\mathbf{y} = A\mathbf{x}$$

where the projection matrix $A \in \mathbb{R}^{p\times N}$ is found through Principal Component Analysis. For a thorough description of PCA, its application and derivation, refer to the PCA slides from the Applied Machine Learning Course.

PCA Algorithm

By computing the covariance matrix of the training data $C = \mathbb{E}\{\mathbf{X}\mathbf{X}^\mathbf{T}\}$ and extracting its eigenvalue decomposition $C = V\Lambda V^T$, the projection matrix A is constructed as $A = V^T$ where $V = [e^1, \dots, e^N]$. To reduce dimensionality, one then chooses a sub-set of p eigenvectors e^i from V.

1.1 Demean the Data

We will follow the steps of the PCA algorithm shown on slide 62 of the class, starting with the centering step,

$$\mathbf{x} \to \mathbf{x} - \mathbb{E}\{\mathbf{X}\} \tag{1}$$

where the expectation of a random variable $\mathbb{E}\{\mathbf{X}\}$ is in fact it's mean. The substraction of the mean is necessary for the eigenvalue decomposition of the covariance matrix to yield the expected result. Hence, the $\mathbb{E}\{\mathbf{X}\}$ of a multi-dimensional vector is the mean vector of all dimensions $\mu_\mathbf{X} = [\mu_1; \mu_2; \dots; \mu_N]$. For each j-th dimension, its mean μ_j can be computed as follows:

$$\mu_j = \frac{1}{M}\sum_{i=1}^{M} x_j^i.$$

`Implementation Hint:` `Useful functions mean(), repmat() or bsxfun().`

1.2 Covariance Matrix Computation

Once the data is demeaned, the next step in PCA is to compute the Covariance matrix $C = \mathbb{E}\{\mathbf{X}\mathbf{X}^{\mathbf{T}}\}$ of our centered (zero-mean) data, as follows:

$$C = \frac{1}{M}\sum_{i=1}^{M}(\mathbf{x}^i)(\mathbf{x}^i)^T \tag{2}$$

`Implementation Hint:  2 can be written in matrix formulation as follows:`

$$C = \frac{1}{M}XX^T$$

Although (2) is the definition of covariance matrix, in your MATLAB function, you should normalize by $M-1$; i.e. $C = \frac{1}{M-1}XX^T$. The reason C is normalized by $M-1$ instead of M is due to the fact that the true $\mathbb{E}\{\mathbf{X}\}$ is not known. According to Bessel's correction, because we use the sample mean $\mu_{\mathbf{X}}$ instead of the true mean, in order to have an unbiased estimate one must use $M-1$. Refer to Applied Multivariate Statistical Analysis book for further reading on this subject.

1.3 Eigenvalue Value Decomposition

The principal components of our dataset can then be computed by extracting the eigenvalues and eigenvectors of the covariance matrix C. This is generally done by solving the following equations:

$$\det(C - \lambda I) = 0, \qquad Ce^i = \lambda_i e^i$$

which are equivalent to, in matrix formulation,

$$C = V\Lambda V^T \tag{3}$$

where $V \in \mathbb{R}^{N\times N}$ is the matrix composed of the eigenvectors e^i on each column and $\Lambda \in \mathbb{R}^{M\times M}$ is a diagonal matrix of eigenvalue λ_i.
`Implementation Hint:  Useful functions eig() or svd().`

TASK 1: Implement my_pca.m function (3pts)
We must ensure that $V = [e^1, e^2, \dots, e^p]$ for $\lambda_1 \geq \lambda_2 \cdots \geq \lambda_p$.

```
[V, L, Mu] = function my_pca(X)

% 1.1 Data Centering
Mu = ...;
X  =  ...; % Equation 1

% 1.2 Covariance Matrix Computation
C  = ...; % Equation 2

% 1.3 EigenValue Decomposition
[V,L] = ... ; % Equation 3

end
```

Test Implementation

At this point, we can test that we have implemented `my_pca.m` function correctly, by running the **first two** code blocks in the MATLAB script `test_pca_2d.m`. This will load a 2D dataset of M=200 datapoints, drawn from a Gaussian distribution with $\mu = [1; 1]$ and $\Sigma = [1, 0.5; 0.5, 1]$ as the one in Figure 1. Moreover, it will compare the output of your function $[V, \Lambda, \mu_{\mathbf{x}}]$ to the PCA function from `ML_toolbox`.

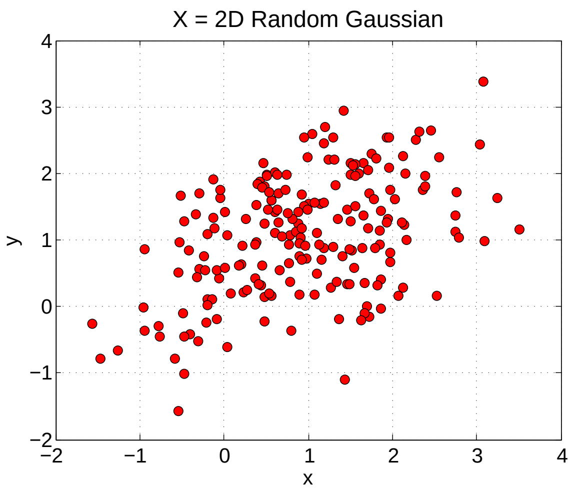

Figure 1: 2D Dataset of 200 points drawn from a Gaussian Distribution. $\mathcal{N}([1; 1], [1, 0.5; 0.5, 1])$

If your implementation is correct, you should see the following messages in your MATLAB command line window:

```
    [Test1] Error in Mean Computation: 0.00
    [Test2] Error in Eigenvalue Computation: 0.00
    [Test3] Error in Eigenvector Computation: 0.00
```

To further check, the expected results are:

$$\mu_k = \begin{bmatrix} 0.9489 \\ 1.0664 \end{bmatrix}, \qquad \Lambda = \begin{bmatrix} 1.2751 & 0 \\ 0 & 0.4780 \end{bmatrix}, \qquad V = \begin{bmatrix} 0.7802 & -0.6256 \\ 0.6256 & 0.7802 \end{bmatrix}$$

where V could have different signs, depending on how the eigenvectors were computed (`eig` or `svd`).

1.4 Projection to Lower Dimensional Space

One then chooses the number of p components/dimensions to keep and selects the first p columns of V to generate the projection matrix A_p

$$A_p = [e^1, e^2, \ldots, e^p]^T \qquad \text{for} \qquad \lambda_1 \geq \lambda_1 \cdots \geq \lambda_p \tag{4}$$

One can then project the zero-mean dataset to the lower-dimensional space as follows:

$$Y = A_p X \tag{5}$$

TASK 2: Implement project_pca.m function (2pts)
Recall that the projection matrix A_p was computed with zero-mean data, hence X in (5) must be the zero-mean dataset.

```
[A_p, Y] = function project_pca(X, Mu, V, p )

% 1.4 Projection to Lower Dimensional Space
A_p = ... ; % Equation 4

X = ... ; % Demean data if necessary
Y = ... ; % Equation 5
```

Test Implementation

To test the `project_pca.m` function, run the **third** code block in the MATLAB script `test_pca_2d.m`. The expected result of the projection code block, with $p = 2$, is shown in Figure 2 we can see how the data is distributed on each eigenvector (histograms) and how it can be decorrelated if we project on both eigenvectors (off-diagonal plots).

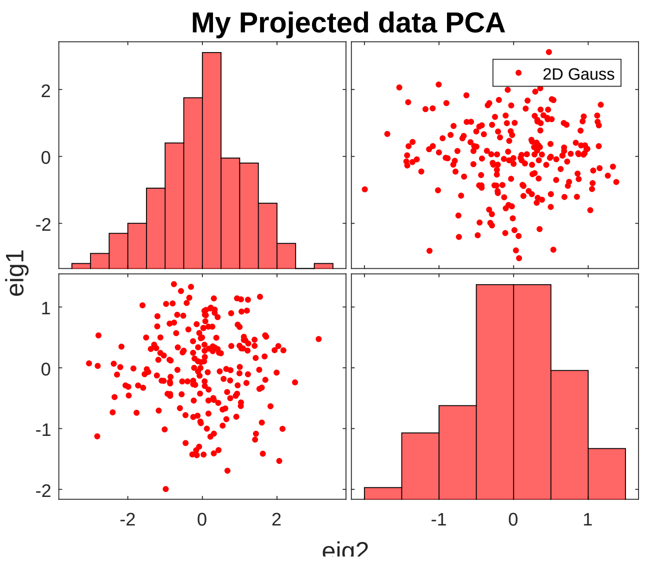

Figure 2: Scatter matrix of dataset projected to each eigenvector

If your histograms and projected data seem to be mirrored to those of Figure 2, this is a result of the opposite signs of V computed from `eig()` or `svd()`, and should yield the same reconstruction. For different MATLAB versions the color and bins of the histograms might change, this does not affect your result.

1.5 Reconstruction from Lower Dimensional Space

Now that we know that `my_pca.m` and `project_pca.m` functions are correct, we can implement the reconstruction function. This can be done by reconstructing a lossy version $\hat{\mathbf{X}}$ of the original dataset $\mathbf{X}$ through mean-square projection as follows:

$$\hat{\mathbf{X}} = A_p^{-1}\mathbf{Y} + \mathbb{E}\{\mathbf{X}\}. \tag{6}$$

TASK 3: Implement reconstruct_pca.m function (1pt)

```
[X_hat] = function reconstruct_pca(Y, A_p, Mu)
% 1.5 Reconstruct from Lower-Dimensional Space
X_hat = ...; %Equation 6
```

To estimate how much information was lost from $\mathbf{X} \rightarrow \hat{\mathbf{X}}$, we can compute the reconstruction error as follows:

$$e_{rec} = ||\mathbf{X} - \hat{\mathbf{X}}||_2 \tag{7}$$

TASK 4: Implement reconstruction_error.m function (1pt)

```
[e_rec] = function reconstruction_error(X, X_hat)
e_rec = ...; %Equation 7
```

`Implementation Hint:` `Useful function norm().`

Test Implementation

The **last** code block in `test_pca_2d.m` tests your reconstruction functions. By running, it you should get the result in Figure 3, which is computed from an A_p with $p = 1$ and yields a $e_{rec} = 9.752934$. The following message should appear in your MATLAB command line window:

```
Reconstruction Error with p=1 is 9.752934
```

If you modify $p = 2$, in the code block you will get Figure 4, which yields a much better $e_{rec} = 0.0$.

```
Reconstruction Error with p=2 is 0.000000
```

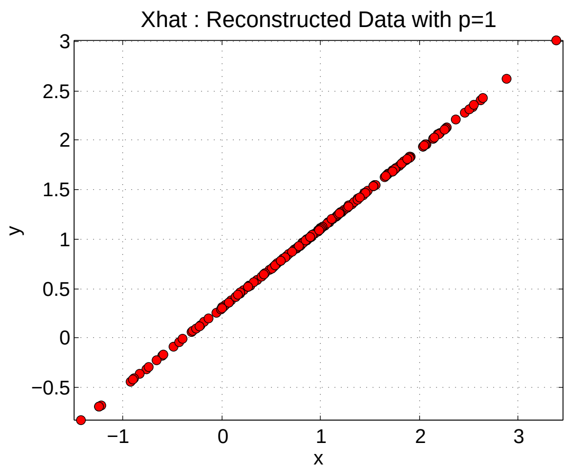

Figure 3: $\hat{\mathbf{X}}$ with $p = 1$ yields $e_{rec} = 9.752934$

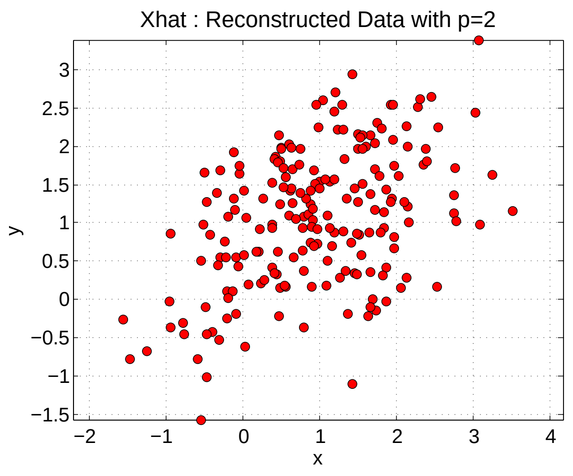

Figure 4: $\hat{\mathbf{X}}$ with $p = 2$ yields $e_{rec} = 0.0$

These results indicate that for this specific dataset it is probably not advisable to reduce dimensions, as much information is lost while projecting to $p = 1$. In the next part of the assignment, we will implement methods for selecting the best p for your dataset and application.

2 Part 2: Application of PCA to High Dimensional Data

2.1 PCA for Pre-Processing, choosing p

In this exercise we will use the Breast-Cancer-Wisconsin (Diagnostic) Dataset[1] which is composed of $M = 698$ datapoints of $N = 9$ dimensions, each corresponding to cell nucleus features in the range of $[1, 10]$. The datapoints belong to two classes $y \in \{\text{benign}, \text{malignant}\}$ (See Figure 5). With such datasets, one would like to reduce the dimensionality to possibly improve classification by finding correlations in the features. Another reason is to reduce computational costs. To achieve this, one must find the optimal p (the number of principal components to use), which can be the determined in two ways:

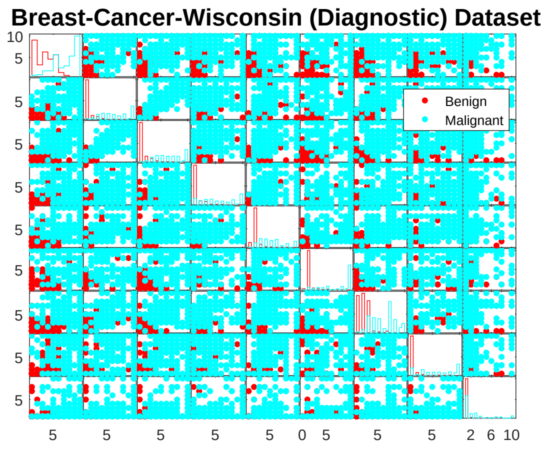

Figure 5: Scatter Matrix of Breast Cancer Wisconsin Dataset on Original Dimensions.

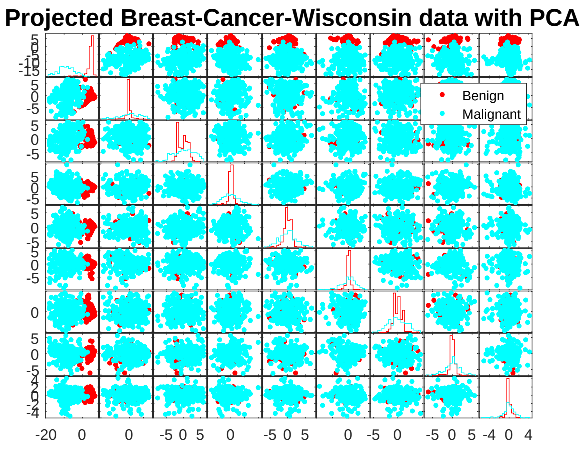

Figure 6: Scatter matrix of dataset projected to each eigenvector.

1) Eigenvalue Analysis

The behavior of the eigenvalues can be used to determine the optimal number of components. When the values corresponding to each principal component (eigenvalues) become constant and close to 0, it could mean that the remaining components are possibly contaminated with noise and can therefore be considered irrelevant. This can be analyzed by extracting the diagonal values λ_i of Λ from (3) and plotting them wrt. their corresponding eigenvector index.

TASK 5: Implement plot_eigenvalues function (1pt)

```
[lambda] = function plot_eigenvalues(L)
lambda = ...; % Column-vector of eigenvalues
```

Test Implementation

We can generate both Figure 5, corresponding to the original dataset, and Figure 6, corresponding to the projected dataset with the functions implemented in Part 1, by running the **first two** code blocks in the MATLAB script `test_pca_9d.m`. For this script to run properly, modify `ml_toolbox_path` at the beginning of the script to the root directory of `ML_toolbox`.

[1]The Breast-Cancer-Wisconsin (Diagnostic) Dataset can be found in the UCI Machine Learning Repository: `https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29`

To test your `plot_eigenvalues` function, run the **third** code block in the MATLAB script `test_pca_9d.m`. The expected output is `lambda` a column-vector filled with the eigenvalues λ_i and Figure 7. As we can see in Figure 7, from component (eigenvector) number 2/3 and up, the eigenvalues are basically constant compared to the first ones, indicating that setting p to either 2 or 3 could be enough to represent the dataset correctly. From Figure 6 one can see that when $p = 2$ the datapoint form the two classes seem to be separated nicely, with some overlapping points. This can be better visualized in Figure 8 when $p = 3$. In fact, if we analyze the projection on the first eigenvector (in Figure 6) we can see this nice separation already. This can indicate that the first eigenvector explains much of the variance of the dataset and that it also encapsulates the correlations between most of the dimensions.

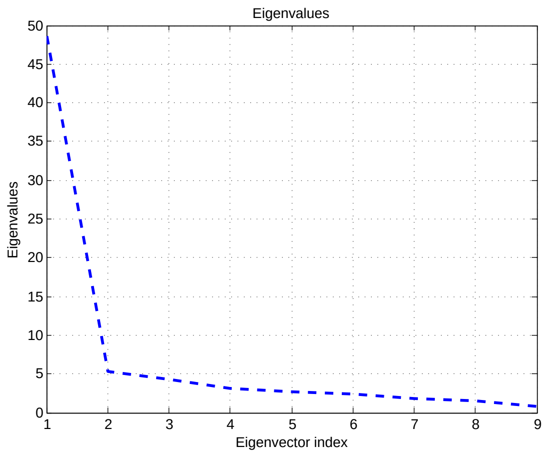

Figure 7: Plot of Eigenvalues

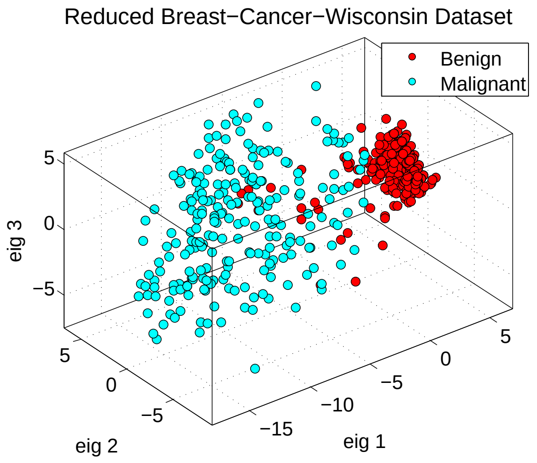

Figure 8: Projected Dataset with p=3

2) Percentage of Variance Explained

Instead of searching for the minimum number of dimensions, another approach is to search for the optimal number of dimensions to keep in order to explain a certain amount of variance of the dataset. The eigenvalues $\Lambda = [\lambda_1, \ldots, \lambda_N]$ give a measure of the variance of the distribution of X on each projection. If one wants to reduce the dimensionality of the dataset, and at the same time keep a reasonable amount of variance of the data, one can find the appropriate p that yields the desired percentage of explained variance Var, by normalizing the eigenvalues as follows:

$$\lambda_i^{var} = \frac{\lambda_i}{\sum_{j=1}^{N} \lambda_j}. \tag{8}$$

This corresponds to the percentage of the dataset covered by the i-th projection. One then computes a vector of the cumulative sum of the set of normalized eigenvalues

$$\lambda_{\mathbf{cum}} = [\lambda_1^{var}, \lambda_2^{var} + \lambda_1^{var}, \ldots, \lambda_N^{var} + \lambda_{N-1}^{var}]. \tag{9}$$

One can then use $\lambda_{\mathbf{cum}}$ to estimate the optimal p for a desired percentage of explained variance Var as

$$p \leftarrow \lambda_{\mathbf{cum}} > Var. \tag{10}$$

TASK 6: Implement explained_variance function (3pts)

```
[exp_var, cum_var, p] = function explained_variance(L, Var)
% Compute cumulative sum of explained variance
exp_var      = ; %Equation 8
cum_var = ; %Equation 9
% Choose p wrt. the Desired Explained Variance
p      = ; %Equation 10
```

`Implementation Hint:` `Useful function cumsum().`

Test Implementation

To test the `explained_variance.m` function, run the **final** code block in the MATLAB script `test_pca_9d.m`. This should generate both Figure 9, corresponding to a plot for the cumulative explained variance, and Figure 10, corresponding to the projected dataset with the function implemented in Part 1, with $p = 5$ automatically chosen from `explained_variance.m` with $Var = 0.9$.

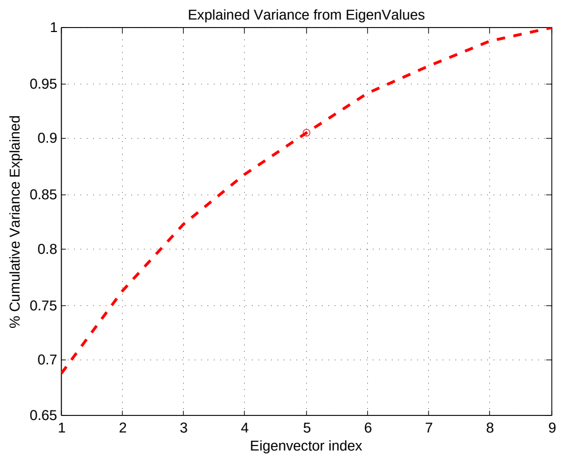

Figure 9: Plot of Cumulative Explained Variance

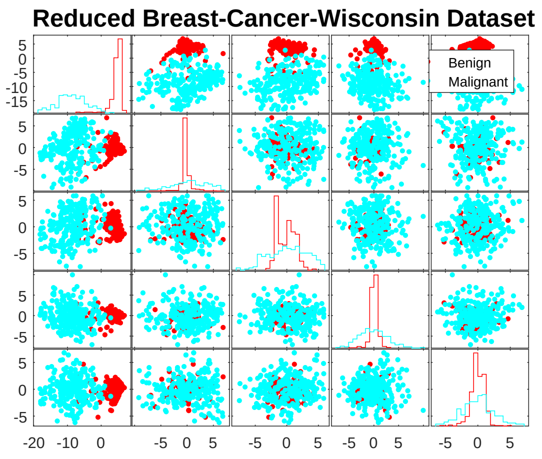

Figure 10: Projected Dataset with p=5

2.2 PCA for Feature Extraction

PCA has also been exploited in the field of facial recognition to represent a set of face images in a lower dimensional space. In this exercise, we will apply the previously coded functions to extract eigenfaces (i.e. eigenvectors applied to images of faces). We will use the Yale Face Dataset[2] which is composed of $M = 165$ face images of 32×32 pixels (N=1024). In this dataset, the faces are centered and the pictures have been taken under similar illumination conditions.

Extraction and display of eigenfaces

To compute the eigenfaces of the Yale Face Dataset, the first step is to convert each image to a vector of N features by concatenation of each row of the image. These vectors are then stored in a matrix $\mathbf{X} \in \mathbb{R}^{N \times M}$ composed of M datapoints (i.e. images) with N-dimensions (i.e. pixels) each. The resulting dataset is provided in the file **'Yale_32x32.mat'**. By running the

[2]The Yale Face Dataset can be found online:
`http://www.cad.zju.edu.cn/home/dengcai/Data/FaceData.html`

first code block in the MATLAB script `test_pca_eigenfaces.m`, $\mathbf{X} \in \mathbb{R}^{N \times M}$ is loaded and we can generate Figure 11, where we can visualize 64 samples of the dataset.

Figure 11: Samples of the Yale Face Dataset

We will then perform the mean subtraction and eigenvalue decomposition using **my_pca.m** function as developed in **TASK 1**. The resulting eigenvectors have the dimensionality of the input images and can be displayed in an image form that represents the principal components of a human face. These images are called **eigenfaces**. For that purpose, one can reshape the eigenvectors in a matrix of size 32×32 as the original images. To validate our result, we will display the first 20 eigenfaces.

TASK 7: Implement plot_eigenfaces.m function (2pts)

```
[V, L, Mu, eigenfaces] = function plot_eigenfaces(X, sizeIm)

% Extract principal components
[V, L, Mu] = ...;

% Plot first 20 eigenfaces
eigenfaces = zeros(sizeIm, sizeIm, 20);
for i=1:20
    % Construct eigenface
    % Plot eigenface
end
```

`Implementation Hint: Useful functions reshape(), imagesc(), subplot() and colormap('Gray').`

Test implementation

Run the second code block in the MATLAB script `test_pca_eigenfaces.m`. The expected result of your `plot_eigenfaces.m` function is the resulting decomposition from applying PCA on **X** and a 3D (32x32x20) array containing the 20 eigenfaces in their original image format as shown on Figure12.

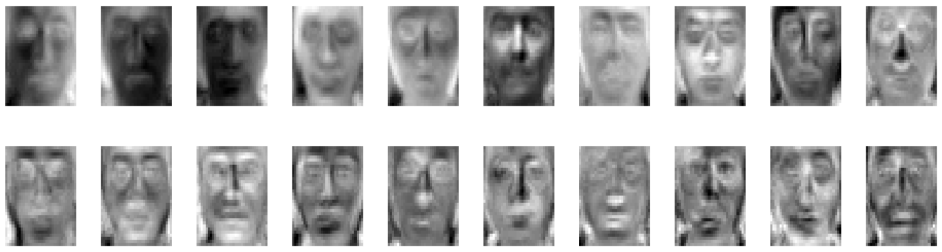

Figure 12: First 20 eigenfaces

Face Reconstruction

The computed eigenfaces encode the differences to the mean image. The eigenvalues that are associated to these eigenfaces give a measure of how much each of them is needed to reconstruct the original face from the mean face image. Here, we will use the **project_data.m** function developed in **TASK 2** to project the face images in lower dimensional space. We will then use the **reconstruct_pca.m** function developed in **TASK 3** to reconstruct a lossy version $\hat{\mathbf{X}}$ of the dataset. To validate our result, we can display the lossy version of the first face image for $p = \{1, 51, 101, 151\}$ to compare them qualitatively to the mean and original face images.

TASK 8: Implement reconstruction_eigenfaces.m function (2pts)

```
function [reconstructed_faces] = reconstruction_eigenfaces(X, V, Mu, sizeIm)
% Plot Original Face
% Plot Mean Face
% Plot Projections p={1,51,101,151}
reconstructed_eigenfaces = zeros(sizeIm,sizeIm,4);
for p = 1 : 50 : 151
    % Reconstruct image with p values
end
```

`Implementation Hint: Useful functions reshape(), imagesc(), subplot() and colormap('Gray').`

Test implementation

Run the final code block in the MATLAB script `test_pca_eigenfaces.m`. The expected result of your `reconstruction_eigenfaces.m` function is and a 3D (32x32x4) array containing the 4 reconstructed faces using the values of $p = \{1, 51, 101, 151\}$ as shown on Figure13.

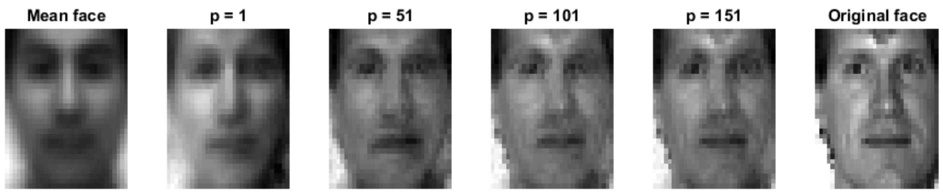

Figure 13: Mean face estimated on the dataset, reconstruction $\hat{\mathbf{X}}$ of the first image of the dataset for 4 values of p, and original first image.