Name: Hongda Li
Class: AMATH 584

Theorem 1

"The nonzero singular values of A are the square roots of the nonzero eigenvalue of AA^H or A^HA."

Suppose that $\lambda \neq 0$ is an eigen value for the matrix A^HA, say that x is the corresponding eigen vector for that eigen value. Then we can say the following:

$$A^HAx = \lambda x$$

Now consider:

$$AA^H(Ax) = A(\lambda x) = \lambda Ax$$

And then we will know that, the value λ is also an eigen value for the matrix AA^H and the corresponding eigen vector is Ax

Therefore, The matrix A^HA and the matrix AA^H has the same set of non-zero eigen values.

Theorem 2

"$A = A^H \implies$ the singular value share the absolute values of the eigen values of A."

Proof:
The the eigen decomposition of matrix A be represented as $A = Q\Lambda Q^H$ where Λ is a diagonal matrix of real numbers. (by properties of Hermitian matrix and the matrix A is Hermitian)
Consider the product A^HA if the eigen decomposition is given:

$$A^HA = (Q\Lambda Q^H)^H(Q\Lambda Q^H) = (Q\Lambda Q^H)(Q\Lambda Q^H) = Q\Lambda^2 Q^H$$

On the other hand we also know that the Eigen decomposition of the matrix A^HA is linked to the singular values of the matrix A by the formula(By SVD of matrices):

$$A^HA = V\Sigma^2 V^H$$

And both expressions are the Eigen decomposition of the matrix A^HA, and here we will be able to obtain the following relationship:

$$\sqrt{\Lambda^2} = \Sigma = |\Lambda|$$

And hence, the statement "The Singular Value are the absolute values of the eigen values of A" is proven.

Theorem 3

"The absolute value of the determinant of a matrix is the product of all its singular values."

To proof this problem we need to use the following properties of matrices determinant

$$\det(AB) = \det(A)\det(B)$$
$$\det(U) = \pm 1 \quad \text{Where U is unitary}$$
$$\det(A^H) = \det(A)$$

Where the last one can be proved with the QR decomposition of the matrix.
Another property we are going to use is the fact that:

$$\det(A) = \prod_i \lambda_i$$

Consider:

$$\det(A^H)\det(A) = \det(A)\det(A) = (\det(A))^2$$

But by SVD, we know that $\det(A^H A) = \det(V\Sigma^2 V^H)$ which is telling us that:

$$\det(A^H A) = \det(V\Sigma^2 V^H) = \det(\Sigma)^2$$

This is true by properties of the matrix determinant. In addtion, the determinant of a diagonal matrix is the product of all of it's diagonal enties, and hence we have:

$$\det(A)^2 = \det(\Sigma)^2 = \prod_i \sigma_i^2$$

$$\det(A) = \prod_i |\sigma_i|$$

Note: Σ is a real matrix.
We have shown that the absolute value of the matrix determinant is the product of all of the absolute values of the singular values.

Yale Faces Explorations

Summary

The codes consists of 3 files, and they are printed by the end of this document for reference:

- analyze.m
  This is the main file.
- ArrayToGrayScale.m
  This function convert an array of floats into an grayscale image and plot it.
- EnergyAnalysis.m
  This use the culmuative sum on the ordered singular values to compute the best rank to reconstruct the data matrix.
- RankReduce.m
  This looks for the $\tilde{X}$, the low rank approximation for the data matrix.
- VarianceAnalysis.m
  This will use the Variance Analysis method to compute the best ranks needed to reconstruct the matrix.

In addition, here is the github repo for all the codes involved in this assignment: Link to my github repository

By replacing line 4, 5 in analyze.m with line 7, one can choose between different files to load for the script.

Question (1)

The SVD decomposition of the column data matrix minus the total average is done by the economic SVD decomposition in matlab. See lines: 24 - 44.

The total average is computed by summing up all the columns and divides by the number of image, and then take the difference on the average columns with all the columns in the matrix.

For discussion, we denotes X to be the column data matrix.

Because we do the "econ" mode of SVD decomposition, we know that $X = U\Sigma V^T$ where X is $m \times n$, U is $m \times k$, Σ is $k \times k$, V^T is $n \times k$ where $k = \min(m, n)$.

Question (2)

Matrix U:

This matrix has the same number of rows as the data matrix.

The matrix is the eigen data set, the columns of the matrix is used to reconstruct data in the training set, or data that are not in the training set.

In this case, they are referred to as the Eigen Faces, representing the common features shared by all the faces in the data set, order from most significant feature to the most non significant features.

The first 16 of these column vectors are visualized using codes on line 63 - 69, and here is the plot:

Matrix Σ:

The singular values(diagonal of the matrices) explains variance of the data set and each of the basis in U is associated with a given singular value. The first singular value is associated with the first column in U and it explains the most variance on our data set, and the second one corresponds to the second column in matrix U and the amount of variance explained by it is second to all the other singular values and so on.

Matrix V:

This matrix has the same number of columns as the number of r

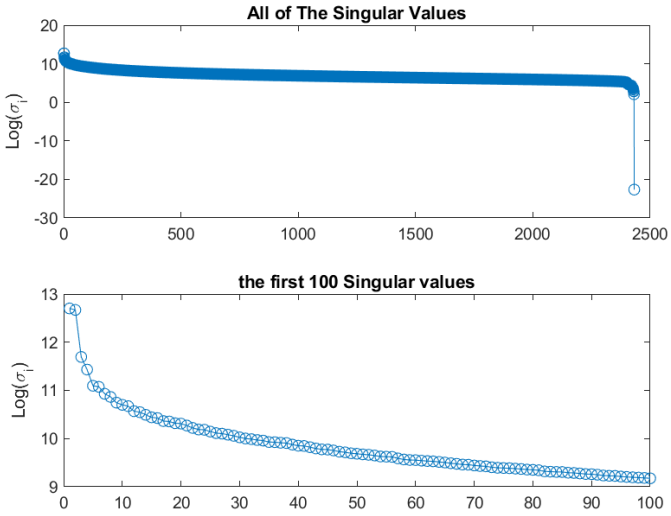
The rows of the matrix V (columns the matrix V^T) is like the finger print of that data that is in the matrix X, it's the best combinations of singular values and its corresponding columns in U such that it can be used to reconstruct the columns in the original matrix.

Here is the reasoning for this problem of the problem. Instead of focusing on the whole data matrix X, we instead select any columns (Faces in that data set) and observe it's reconstructed via components from the U, Σ, V^T matrices.

$$(X)_{col(i)} = \sum_k (U)_{col(k)} (\Sigma V^T)_{k,i}$$

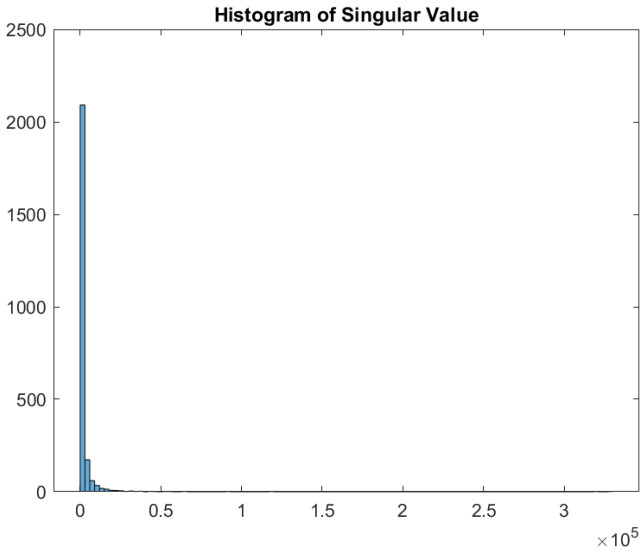
Question (3)

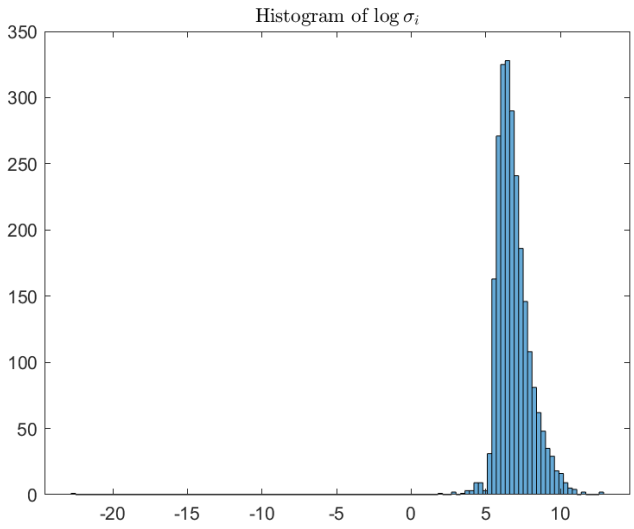
The plot of ordered singular values in descending order or their logarithm. Observe that the first few singular values are large in magnitude compare to the rest.



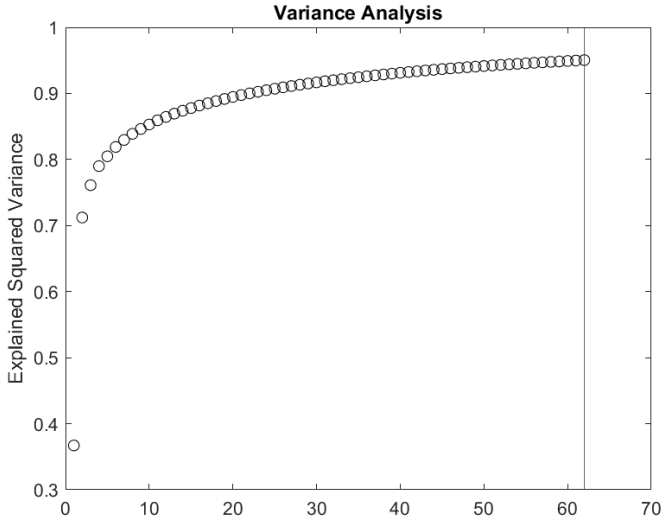
e

Here is a histogram on all the singular values:

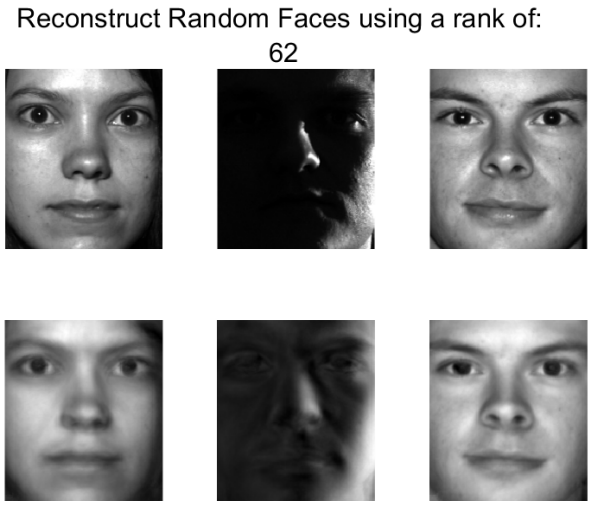




The variance is simply the squared difference between all the elements in the matrix $\tilde{X}$. With different rank, we compute the portion of variance on the matrix $\tilde{X}$ divides the total amount of variance in the matrix $\tilde{X}$. The best rank is the number of top singular values that explains a variance over 95%.

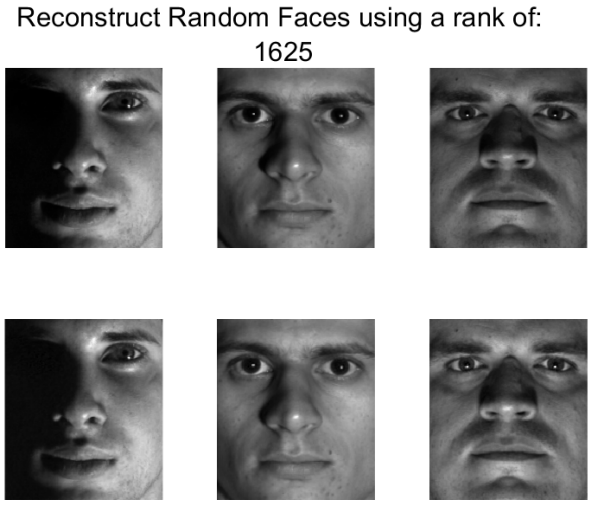


The vertical line is at rank = 62, meaning that using a 62 modes we will be able to explain over 95% of the sum of squared variance in the original matrix, and here is how the reconstruction for 62 modes looks like:

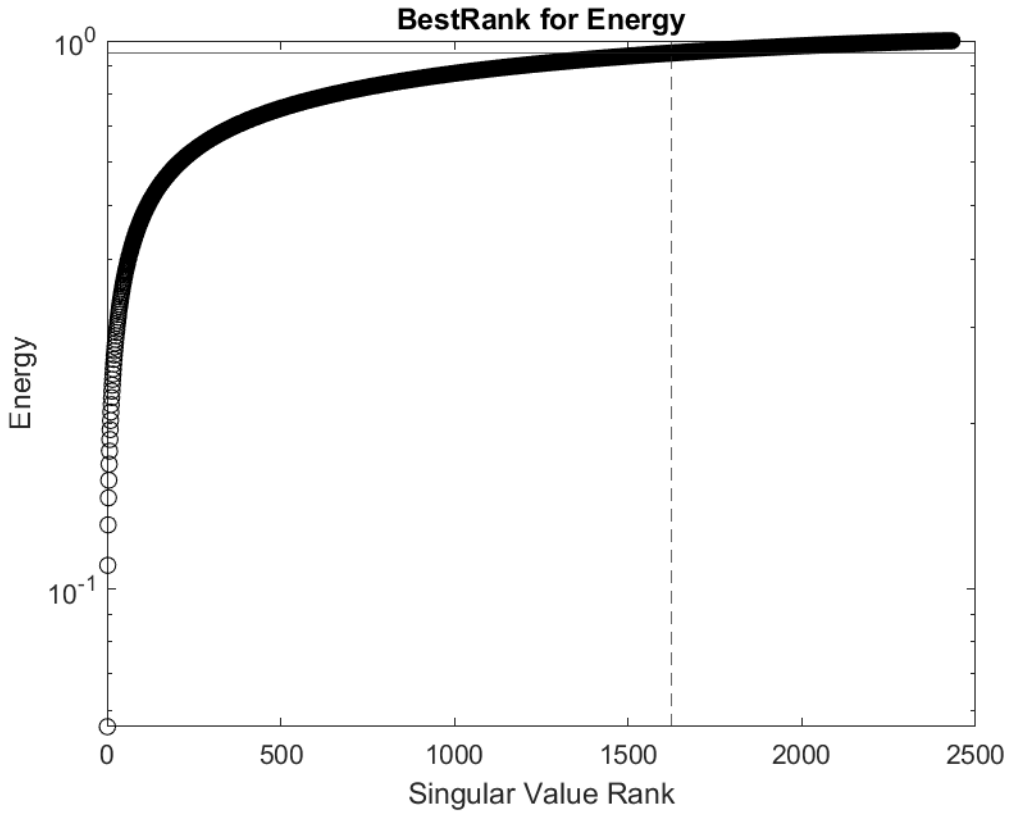


As we can see, that most of the features has been successfully reconstructed and the faces are visibly distinguishable from each other.

Here is the plot of the cumulative energy of singular value functions.



The 1525 modes are needed to reconstruct the image, as we can see that the reconstruction of the image is near perfect and it's almost indistinguishable, and here is the plot of the culmulative energy of singular values:

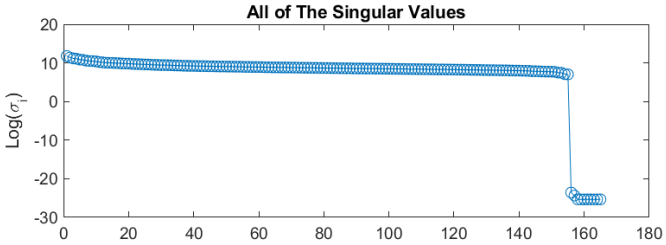


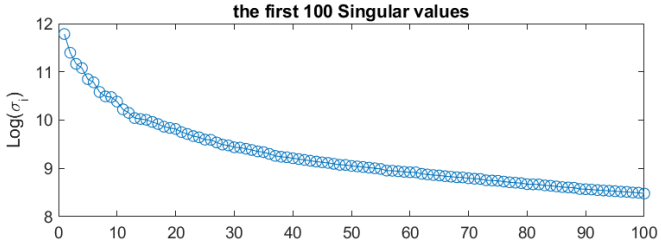
Other ranks needed for a certain threshold of energy is also determined, for a energy threshold of 90% and 99%, the number of singular values needed are 1183, 2184.

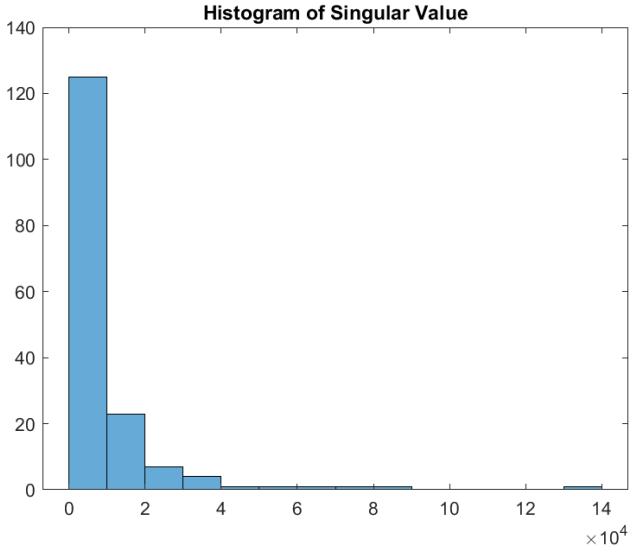
Question (4)

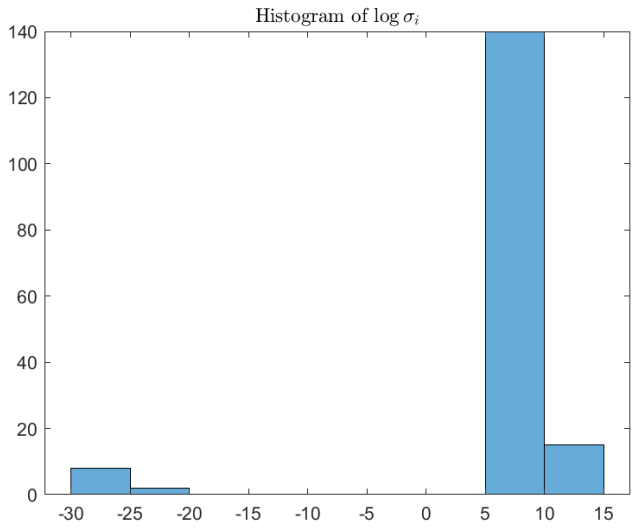
One of the limitation of the SVD decomposition is its inability to handle the unitary transformation of data object. The uncropped faces data set has more variance on the position of the head, hence it hinders the ability for SVD to reduce the number of dimensions of the data set.

Firstly, let's take a look at the distribution of the singular values:



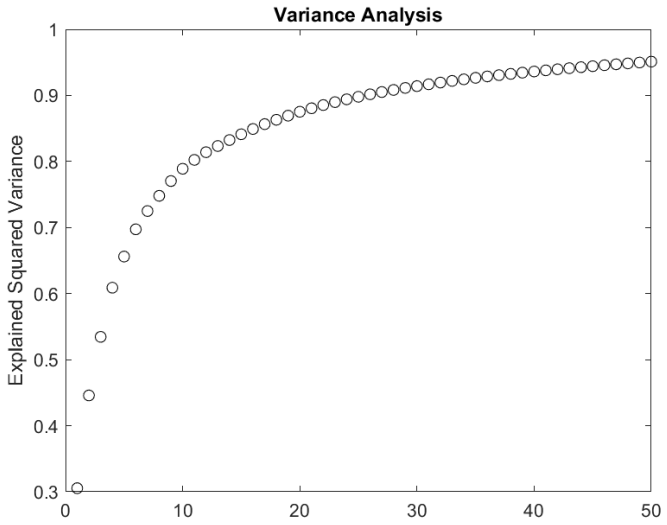


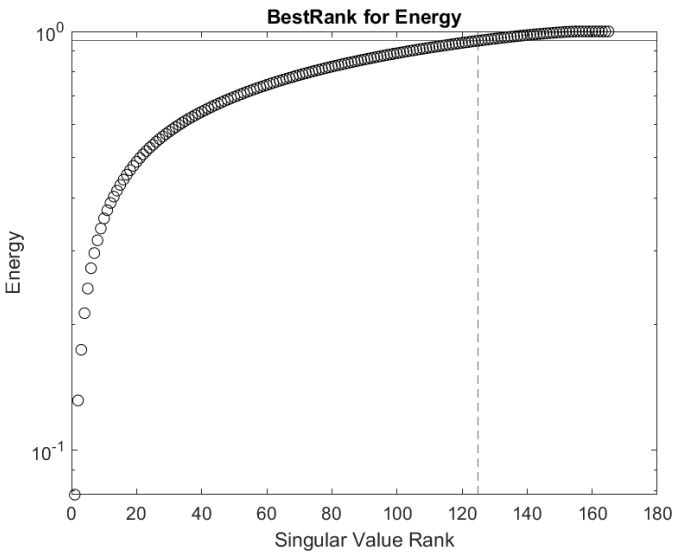




Notice that, the relative size of the first few singular values are more spread out compare to singular values from the cropped data set. Also notice that the decay of magnitudes on the singular values is less steep than the cropped data set.

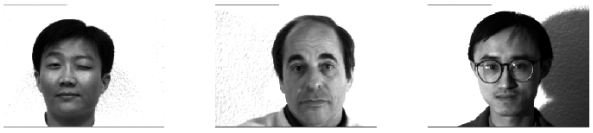
Now let's take a look at the rank determined by variance and energy analysis:





Reconstruct Random Faces using a rank of:
125

Reconstruct Random Faces using a rank of:
50



Notice that, the portion of singular values (the modes) we need to reconstruct the matrix is much larger compare to the cropped data set. The variance analysis produce a singular values of 40, and there are 165 images.

Similarly, the energy threshold is given as 95%, and 125 modes are needed to for reconstruction, which is still a much larger portion compare the the data set produced from the uncropped images.

file: analyze.m

```
clear all; close all; clc;

%% Get files
dirinfo = dir("yale-faces\yalefaces_cropped\CroppedYale\**");
dirinfo([dirinfo.isdir]) = [];

% dirinfo = dir("yale-faces\yalefaces_uncropped\subject*.*");

%% Meta Setting, create matrix
TRAINING_SET = 1: length(dirinfo);

% Get Matrices
Matrices = cell(1, length(TRAINING_SET));
for I = TRAINING_SET
    % TheImage = imread(strcat(dirinfo(I).folder, "\", dirinfo(I).name), "pgm");
    TheImage = imread(strcat(dirinfo(I).folder, "\", dirinfo(I).name));
    ImageSize = size(TheImage);
    Matrices{I} = TheImage;
end
% imshow(Matrices{100})

%%
% Put them into a big matrix
ColumnDataMatrix = zeros(size(Matrices{1}, 1)*size(Matrices{1}, 2), length(Matrices));
Column = 1;
for Matrix = Matrices
    Matrix = Matrix{1};
    ColumnDataMatrix(:, Column) = ...
        reshape(Matrix, [size(Matrix, 1)*size(Matrix, 2), 1]);
    Column = Column + 1;
end
clearvars -except ColumnDataMatrix ImageSize;

%%
% Time for Maths.

ImageTotalDataPoints = size(ColumnDataMatrix, 1);
NumberofImages = size(ColumnDataMatrix, 2);
TotalAverage = (ColumnDataMatrix*ones(NumberofImages, 1))/NumberofImages;

figure; hold on; image(reshape(TotalAverage, [ImageSize(1) ImageSize(2)]));
title("Your Average Matrix Creepy Face");

[U, S, V] = svd(ColumnDataMatrix - TotalAverage, 'econ');  % SVD on Variance Matrix!

%% PLOTING THE SINGULAR VALUES RELATED STUFF.

figure;
subplot(2, 1, 1);
plot(1:length(diag(S)), log(diag(S)), '-o');
title("All of The Singular Values");
ylabel("Log(\sigma_i)")

```

```
subplot(2, 1, 2);
SingularVals = log(diag(S));
plot(1:100, SingularVals(1:100), '-o');
title("the first 100 Singular values");
ylabel("Log(\sigma_i)")
saveas(gcf, "Singular_Value_Distribution.png");

figure;
histogram(diag(S));
title("Histogram of Singular Value");
saveas(gcf, "Singular_Value_Histogram.png");

figure;
histogram(log(diag(S)));
title("Histogram of $$\log{\sigma_i}$$", 'Interpreter','latex');
saveas(gcf, "Singular_Value_logrithm_histogram.png");

%% LOOKING A EIGEN FACES AND PLOTING IT OUT.
figure;
title("first 16 Basis in U (EigenFaces)");
for I = 1:16
   subplot(4, 4, I);
   ImgArr = U(:, I);
   imshow(ArrayToGrayScale(ImgArr, ImageSize));
end

saveas(gcf, "EigenFaces.png")

%% Variance Analysis
% COMPUTATIONALLY HEAVY
% Instead of plotting it, I am going to visualize this numerically to
% See the errors for low rank approximation.
% Technique: Variance Analysis.

[Bestrank1, CulmulativeVariance] = ...
    VarianceAnalysis(U, S, V, ColumnDataMatrix - TotalAverage, 0.95);

%% VARIANCE ANALYSIS AND PLOTING.
figure;
plot(CulmulativeVariance, "ko");
ylabel("Explained Squared Variance");
title("Variance Analysis");
xline(Bestrank1);
saveas(gcf, "Rank and Explained Squared Variance.png");



%% Energy Analysis AND PLOTING.
% Another way of determing the rank of the matrix is to use the idea of
% energy, see code 1.19 in the data book for more details.
Bestrank2 = EnergyAnalysis(S);
saveas(gcf, "Energy Analysis.png");
Bestrank90 = EnergyAnalysis(S, 0.9);
Bestrank99 = EnergyAnalysis(S, 0.99);
```

```
108
109 %% Reconstruction of Known Faces
110 % Using the new gotten ranks for reconstructing a certain column in the
111 % original matrix.
112 ReconstrctRandomFaces...
113     (ColumnDataMatrix, U, S, V, Bestrank1, ImageSize, TotalAverage);
114 saveas(gcf, "Random Faces Reconstruction using Variance Analysis.png");
115
116 ReconstrctRandomFaces...
117     (ColumnDataMatrix, U, S, V, Bestrank2, ImageSize, TotalAverage);
118 saveas(gcf, "Random Faces Reconstruction using Energy Analysis.png")
119
120 function ReconstrctRandomFaces(dataMatrix, U, S, V, rank, ImageSize, TotalAverage)
121     NUMBER_OF_RANDOM_FACES = 3;
122     RandomFaces = ...
123     randi([1 size(dataMatrix, 2)], NUMBER_OF_RANDOM_FACES, 1);
124     [A_tilde, U_tilde, S_tilde, V_tilde] = RankReduce(U, S, V, rank);
125     figure;
126     for I = 1: NUMBER_OF_RANDOM_FACES
127         FaceID  = RandomFaces(I);
128         TheFace = dataMatrix(:, FaceID);
129         subplot(2, NUMBER_OF_RANDOM_FACES, I);
130         imshow(ArrayToGrayScale(TheFace, ImageSize));
131         TheFaceReconstruct = A_tilde(:, FaceID);
132         subplot(2, NUMBER_OF_RANDOM_FACES, NUMBER_OF_RANDOM_FACES + I);
133         imshow(ArrayToGrayScale(TheFaceReconstruct + TotalAverage, ImageSize));
134     end
135     sgtitle(["Reconstruct Random Faces using a rank of: ", num2str(rank)]);
136 end
```

file: EnergyAnalysis.m

```
function BestRank = EnergyAnalysis(s, threshold)
%   Function will use the SVD decomposition of the matrix to look for the
%   number of singular values that gives 95% of the engery.
    switch nargin
        case 2
            % pass
        case 1
            threshold = 0.95
        otherwise
            error("must be 1, or 2 parameters. ")
    end

    D = diag(s);
    TotalEnergy = cumsum(D)./sum(D);
    BestRank = min(find(TotalEnergy > threshold));
    figure;
    semilogy(TotalEnergy, 'ok');
    ylabel("Energy");
    xlabel("Singular Value Rank");
    title("BestRank for Energy");
    xline(BestRank, '--');
    yline(TotalEnergy(BestRank));
end
```

file: RankReduce.m

```
function [A_tilde, U_tilde, S_tilde, V_tilde]...
    = RankReduce(U, S, V, r)
%  The function does a rank reduction on the SVD matrices of a given matrix.
%
%  U, S, V:
%     These are the matrix gotten from the SVD decomposition where diagonal
%     elements of matrix S has to be singular value ranked in decreasing
%     order of magnitudes.
%
%  r:
%     The number of top rank singular values you want to use to approximate
%     the given matrix with.
%  Return:
%     The low rank approximation of the matrix that got decomposed to
    if r > min(size(U, 1), size(V, 2))
        error("Value of r is larger than the number of possible singular values")
    end
    U_tilde = U(:, 1:r);
    S_tilde = S(1:r, 1:r);
    V_tilde = V(:, 1:r);
    A_tilde = U_tilde*S_tilde*V_tilde.';
end
```

file: ArrayToGrayScale.m

```
function Fxout = ArrayToGrayScale(arr, size)
% This function plots an array into an gray scale image.
% arr:
%   A vector that represents all the datapoint of the image.
% size:
%   2d array representing the size of the matrix.
% tittle:
%   String that is going to be the tile of the plot.
   arr = arr - min(arr);
   Scale = 255/max(arr);
   Fxout = uint8(reshape(arr*Scale, size));
end
```

file: VariandeAnalysis

```
function [Bestrank, CulmulativeVariance] = ...
    VarianceAnalysis(u, s, v, originalMatrix, threshold)
%   Using the explained variance of the reconstructed matrix to determine
%   the best number of singular values needed to reconstruct the matrix.
%   u, s, v:
%       The matrices decomposed from the SVD decomposition.
%   originalMatrix:
%       The matrix that got decomposed into u, s, v
%   threshold:
%       threshold for explained variance
    ReconstructedMatrix = zeros(size(u, 1), size(v, 1));
    CulmulativeVariance = [];
    for I = 1: size(s, 1)
        ReconstructedMatrix = ...
            ReconstructedMatrix + u(:, I)*s(I, I)*v(:, I).';
        CulmulativeVariance(I) = getVarianceFor(ReconstructedMatrix);
        CulmulativeVariance(I) = ...
            CulmulativeVariance(I)/getVarianceFor(originalMatrix);
        if CulmulativeVariance(I) > threshold
            Bestrank = I;
            break;
        end
    end

    function MatrixVariance = getVarianceFor(matrix)
        MatrixVariance = ...
            var(reshape(matrix, [1, size(matrix, 1)*size(matrix, 2)]));
    end
end
```