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 sare 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 formuta(By SVD of matrices):

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

And both expression 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:

- analyze.m: This is the script the load data, do the SVD and do all the plotting.
- ArrayToGrayScale.m: This function convert a row/column vector into a matrix of uint8 of a given size, it's for visulizing the eigen faces.
- RankReduce.m: Give the SVD matrices of a matrix, and a rank, it returns the best approximated matrix with the given rank.

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:

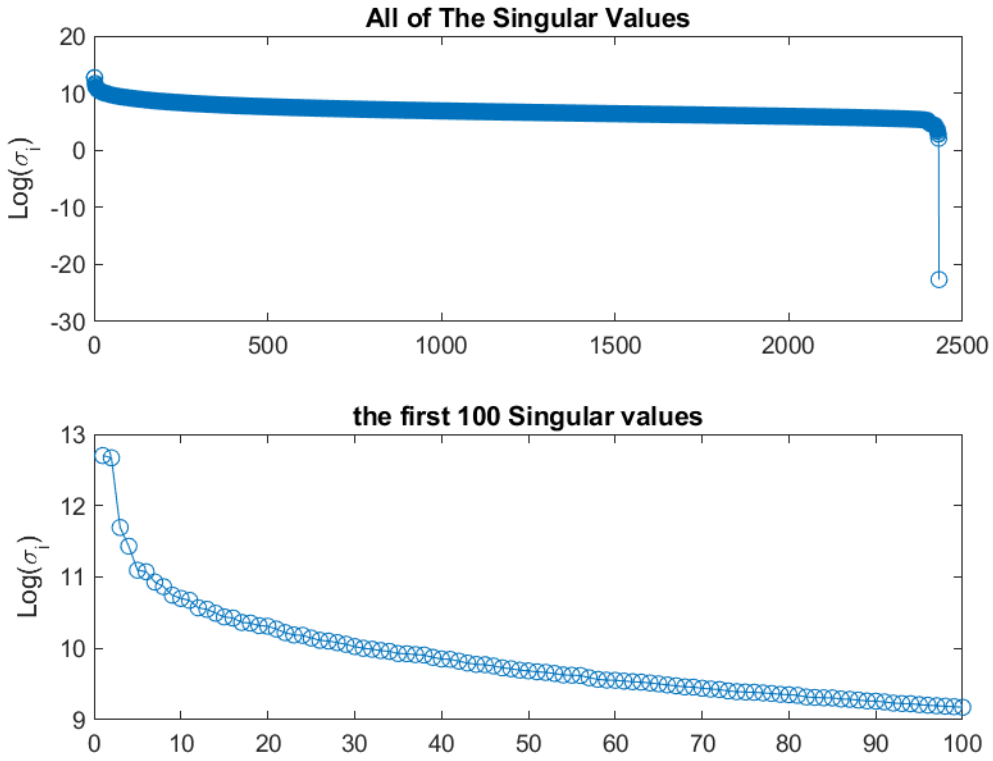
This matrix has the same number of columns as the number of r

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.

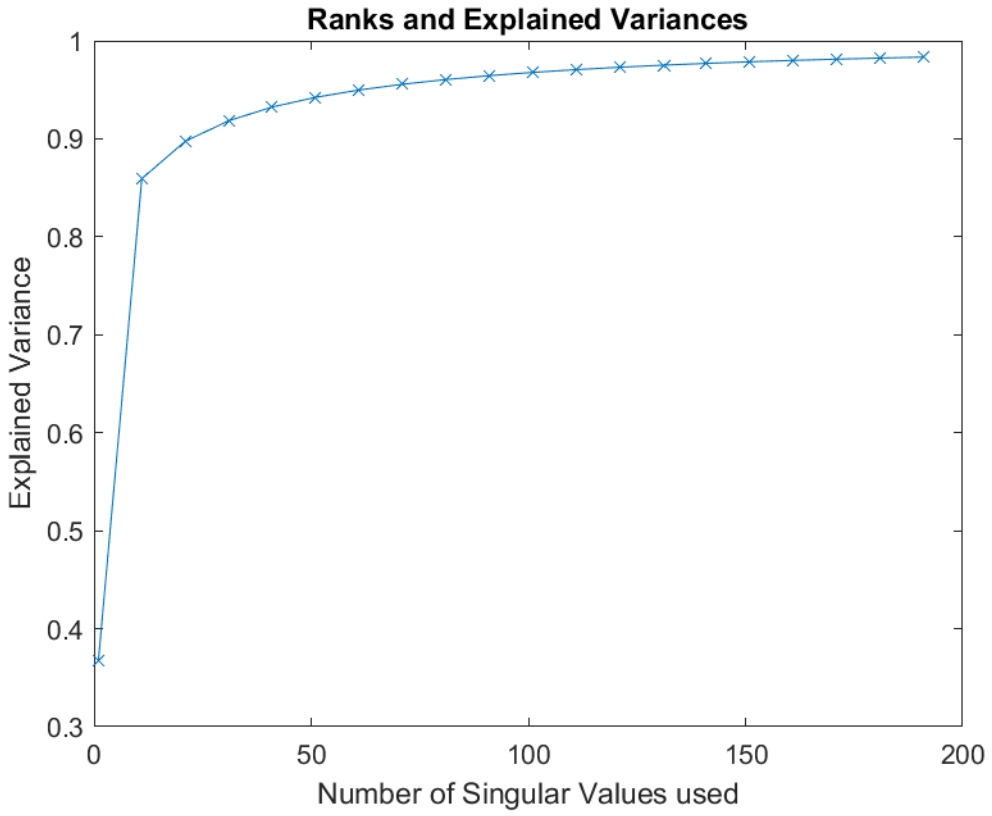
$$(X)_{col(i)} = \sum_k (U)_{col(k)} (\Sigma V^T)_{k,i}$$

Question (3)



This is the plot of the singular values, where the x-axis is the index of the singular value and the y-axis is the logarithm of the singular value.

The number of modes needed for a good approximation of the original data matrix is determined by variance analysis. For each number of modes use, we try to reconstruct the approximated matrix $\tilde{X}$, and then compute the variance after all the elements of $\tilde{X}$ and divides it by the total variance on all the entries of the matrix X to obtain the explained variance for the given number of modes. See line 106-123. Here is the plot of number of modes and variance explained:

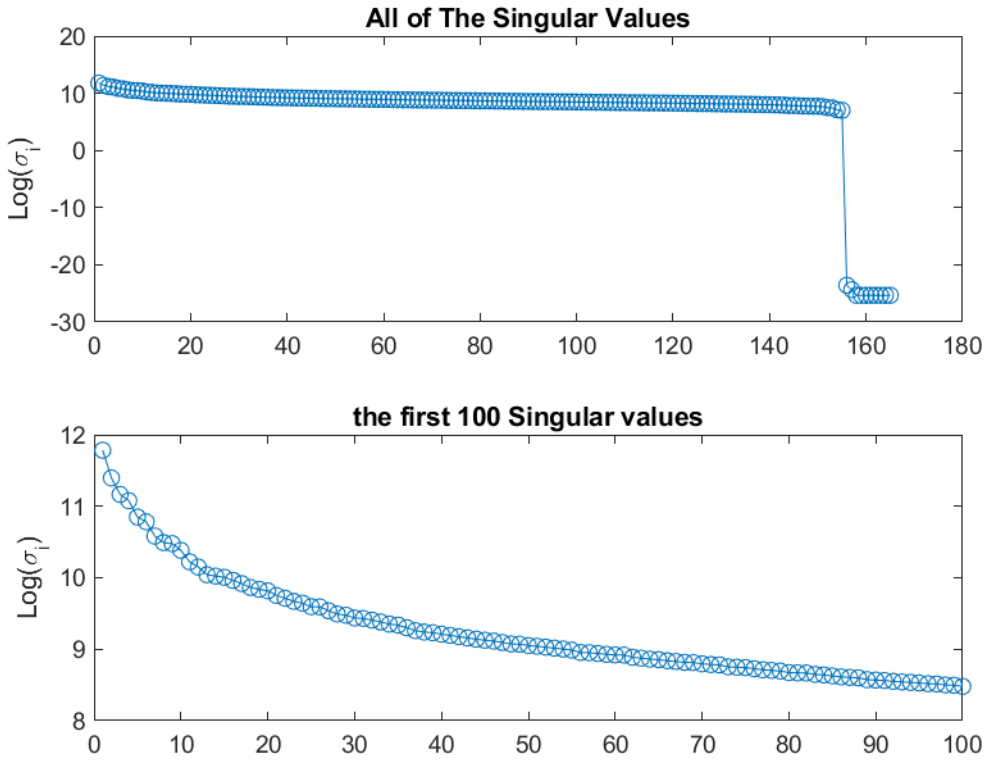


As we can see that, over 89% of the variance are explained by the first 10 singular values, and the least number of modes needed to explain over a 90 of variance is approximately 71 modes.

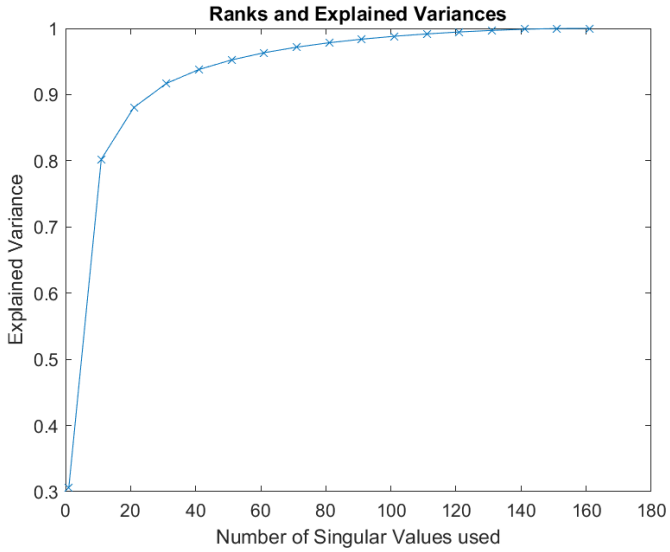
Question (4)

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 dimentions of the data set. The first significant difference is the distribution of the singular values and the portions of modes needed to explain over 95% of the variance.

Here is the plot of the singular value distribution of the uncropped faces:



The value of the singular value decays slower compare the uncropped faces data set. And this is the plot of number of modes and the amount of explained variance of the matrix:



The number of modes need to explain over 95% of the variance is approximately 51, which is about **one third** of all the singular values of the matrix. Compare to 71/2432, which is a much much smaller portion of singular values.

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!

%% Look at the singular values

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");

%% Look at the Basis in U
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")

%% reconstructions For Known Faces
% define good constants:
NUMBER_OF_RANDOM_FACES = 3;
RECONSTRUCTION_RANK = 200;

% Reconstruct for faces inside of the known data set.
U_tild = U;
S_tild = S;
S_tild(RECONSTRUCTION_RANK + 1: end, :) = 0;
V_tild = V;  % Careful about here, because USV^T
A_tild = U_tild*S_tild*V_tild.';
A_tild = A_tild + TotalAverage;

RandomFaces = ...
    randi([1 size(ColumnDataMatrix, 2)], NUMBER_OF_RANDOM_FACES, 1);
figure;
for I = 1: NUMBER_OF_RANDOM_FACES
    FaceID  = RandomFaces(I);
    TheFace = ColumnDataMatrix(:, FaceID);
    subplot(2, NUMBER_OF_RANDOM_FACES, I);
    imshow(ArrayToGrayScale(TheFace, ImageSize));
    TheFaceReconstruct = A_tild(:, FaceID);
    subplot(2, NUMBER_OF_RANDOM_FACES, NUMBER_OF_RANDOM_FACES + I);
    imshow(ArrayToGrayScale(TheFaceReconstruct, ImageSize));
end

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

RANKS = 1:10:min(size(ColumnDataMatrix, 2), 200);
VarianceUnexplained = zeros(length(RANKS));
TotalVariance = (ColumnDataMatrix - TotalAverage);
TotalVariance = ...
```

```
    reshape(TotalVariance, [1, size(TotalVariance, 1)*size(TotalVariance, 2)]);
TotalVariance = var(TotalVariance);

Variances = [];
for R = RANKS  % This part can be made faster with dynamic programming, but whatever.
    A_tilde = RankReduce(U, S, V, R);
    A_tilde = reshape(A_tilde, [1, size(A_tilde, 1)*size(A_tilde, 2)]);
    VarianceExplained = var(A_tilde);
    Variances(end + 1) = VarianceExplained/TotalVariance;
end
figure;
plot(RANKS, Variances, "-x");
title("Ranks and Explained Variances");
ylabel("Explained Variance");
xlabel("Number of Singular Values used");
saveas(gcf, "ExplainedVariance.png")
```

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: RankReduce.m

```
function [A_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
```