Jonathan Lu
Report on PCA (HW1)

This project involved analyzing various face samples from Yale's database using Principal Component Analysis. PCA has become a powerful tool in computer vision, as every person has a unique signature in a feature space. As a result, PCA can be used for face recognition software in finding distinguishing data that makes that person look like that person. It is powerful as is does not require pixel-by-pixel comparison, which would be arduous and take a long time. Instead, it reduces the whole set of images into a lower dimensional space by removing redundancy.

Sec. I: Introduction and Overview

PCA is often used in relation with data collection. Suppose that someone doesn't know the physics behind a system, such as an oscillating spring-mass system (i.e. $F = Ma$ was never invented by Newton). Furthermore, data is never clean – you will always get noise, whether it be a wobbling camera or uneven release of the spring. There also tends to be a lot of redundancy in data, making it difficult to find the relationship between the data sets. A person could collect the data using a camera and analyze it using a computer. They put the data into a matrix and organize it into a covariance matrix. This can be computed and tells the relation of the data.

Sec. II: Theoretical Background

Ideally, PCA involves collecting a lot of data and putting it all into a matrix. The SVD is essentially a transformation that stretches/compresses and rotates a given set of vectors, much like what Eigen-decomposition can do. With data, one can do the SVD to produce the U, Σ, V matrices, which tell us a lot. U and V are orthonormal, so they represent the rotations. The U matrix from SVD parallels the eigenvectors from Eigen-decomposition. The V matrix tells us how much each data projects onto the feature space. Σ is a diagonal matrix that parallels the eigenvalues in Eigen-decomposition. Furthermore, the values of σ (the singular values) in Σ tell us the principal components. These singular values are all ranked in order of importance, and they are also all positive and real.

With all the data, a covariance matrix can be constructed by

$$C_x = \frac{AA^T}{n-1}$$

where A represents the data matrix. This matrix tells us the variability in data streams (variance) and how associated the data streams are (covariance). The covariance matrix is real and symmetric, with the diagonals telling you the variance and off-diagonals telling you the covariance. A high variance means that the a lot is going on in the data. A very small covariance tells us that the data is statistically independent; while a larger value tells us that the data is statistically dependent, meaning that there is a lot of redundancy.

Diagonalization is a very important step in PCA. It is essentially making the covariance matrix into a diagonal matrix. Its goal is to find the feature space where the data is maximally coordinated. Creating the feature space is done by multiplying the data by the rotation matrix U to get

$$Y = UA$$

where Y represents the feature space. Now the covariance matrix of Y is

$$C_y = \frac{\Sigma^2}{n-1}$$

which is a covariance matrix with off-diagonals set to 0 and diagonals are ordered. This also allows us to find the direct relation between SVD and Eigen-decomposition, where $\Sigma^2 = \Lambda$. But the fact that the data matrix can be projected into a feature space tells us how many degrees of freedom matter within a system. With the principal components projection $Y = UA$, this gives each face a distinct signature.

Sec. III: Algorithm Implementation and Development

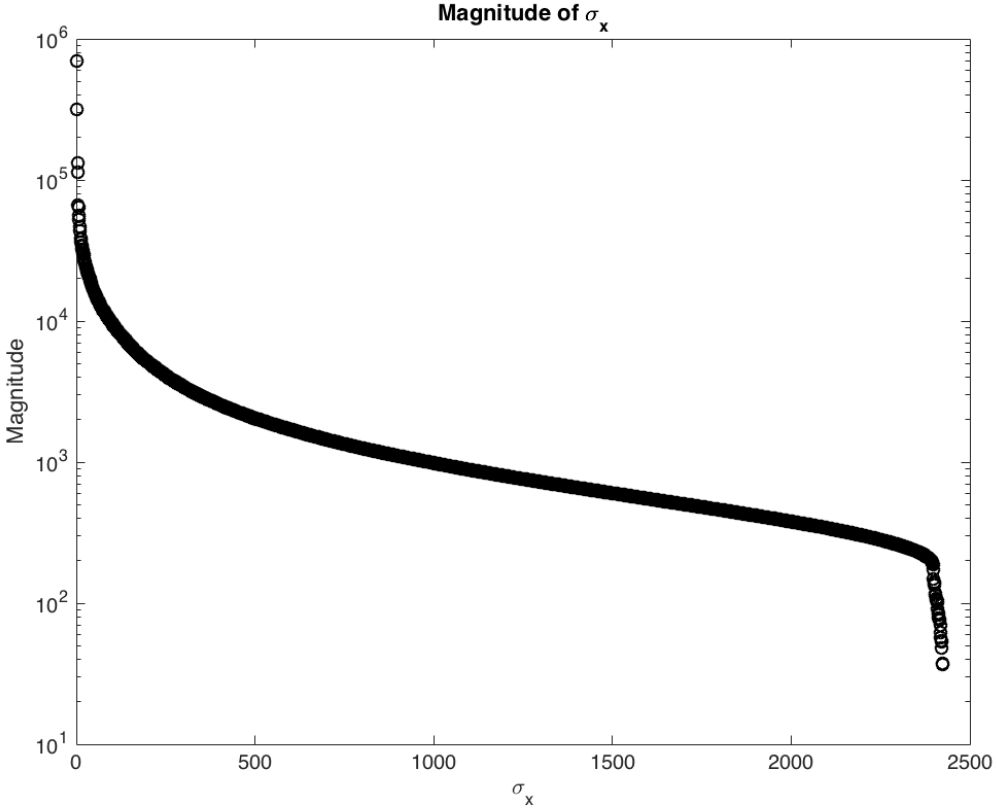
(Refer to code in Appendix B)

The first part of algorithm implementation involved loading all the files into MATLAB from the directory, which took a very long time (for the ExtendedYaleB directory). This involved parsing through the file directory through each person, processing the very obscure filenames using (*) wildcards and adding the picture data into the data matrix, then traversing back to the parent directory. The pictures were in .pgm, so there was no need to convert the image into gray-scale. There were a bunch of ambient files that did not have the correct size, causing issues when adding it to the data matrix, so I removed them. However, the data was also uint8, so I had to convert it into double so that the data would be more precise. Furthermore, the matrix data of each picture needed to be reshaped into a vector then added to the data matrix. This made each data matrix take up one row instead of having various sub-matrices in the data matrix. This was all done within a nested for loop. For some reason, there was no Yaleb14 subject. Then svd() was performed on the data matrix (in economic mode to save space/time). With the U, Σ, V matrices, many conclusions could be drawn.

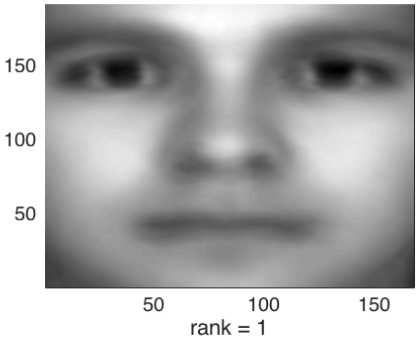
Sec. IV: Computational Results

(For some reason, setting ticks on the axes caused my PC to crash)

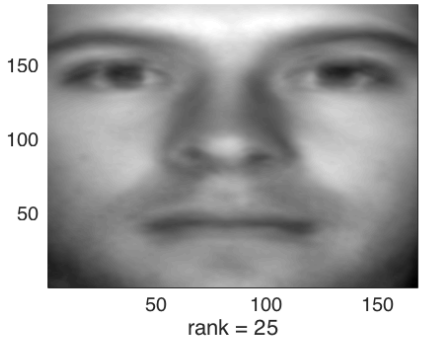
After performing the SVD, I plotted the singular values of Σ on a log plot, which were on the diagonal. From the results below, we can see that all of the singular values were sorted in decreasing order. The first singular value had a very large magnitude, on a scale of nearly 10^6, but there was a very sharp decay. Eventually the values decayed to values within 10^1. Most singular values were clustered around 10^3. This tells us that the first few singular values are the most valuable – as they contain the most data.



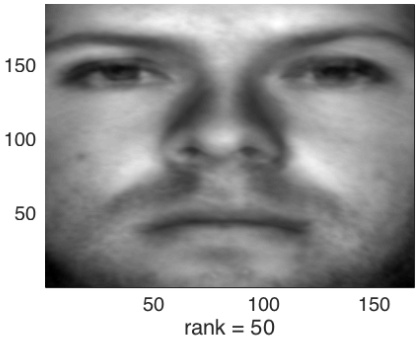
However, to visually see how much of an effect the singular values had on reconstructing the image, I did $A = U\Sigma V^*$ again but this time only chose the first few number of singular values corresponding to the rank (i.e. if I wanted to see the effect of the first 25 singular values, I set every value of Σ to 0 except for the first 25). I did this for 1, 25, 50, 100, 250, 500, 1000, and all 2414 singular values for the first test subject. With this, we could determine the distinct features of the face. σ_1 provided a base facial structure, and amazingly even showed the shadows and distinct features. σ_{1-25} already provided a decent facial reconstruction, as if it was taken from a poorly lit background from late night television. However, starting from σ_{1-50}, we could already make out the defining features of this face, such as a mole on the right cheek or several bumps. In my completely objective opinion, σ_{1-100} was the best mode necessary for rough image reconstruction, with the rest of the singular values being used to add shadows and clarity.



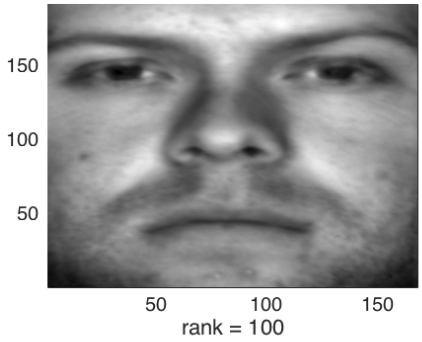
rank = 1



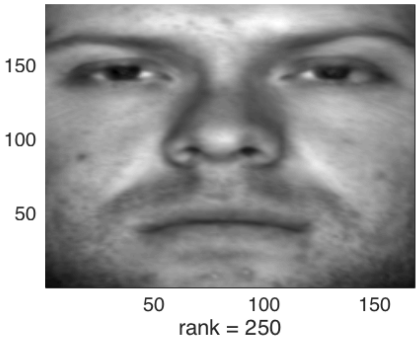
rank = 25



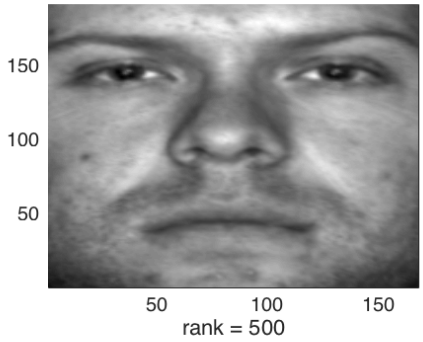
rank = 50



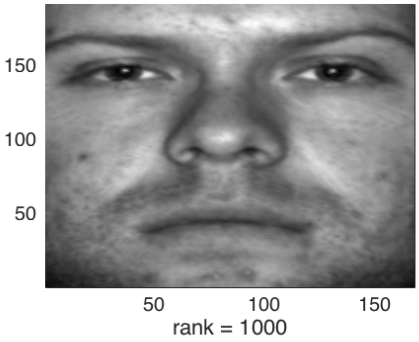
rank = 100



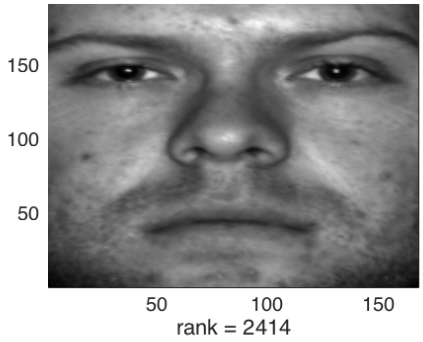
rank = 250



rank = 500



rank = 1000



rank = 2414

Sec. V: Summary and Conclusions

Based on the computational results, we see that singular values are ranked in order of magnitude, and they are all real and positive. These singular values decay, meaning that the first few have a greater impact in image reconstruction. The first singular value reconstructed already created a base face structure. Also, each face has a unique signature, given by their projection onto the feature space. This could be used in facial recognition, to see if a face can be matched with someone's signature to see who that person is.

PCA is a powerful tool to remove redundancy and noise in systems. One can measure the covariance and variance between data, which tells us a lot about the system – such as statistical dependence. What separates the SVD from Eigen-decomposition is that every matrix is guaranteed to have an SVD, but not an Eigen-decomposition. Furthermore, since Eigen-decomposition only exists for square matrices, even then it is not guaranteed. Both tell you the physics behind systems, but SVD is more robust.

Appendix A

imread() – reads the image and gives out a matrix that has a corresponding black/white value
double() – converts uint8 to double precision, giving better data
reshape() – reshapes a matrix to a vector and vice versa
svd() – do singular value decomposition, returns corresponding rotation matrices and singular values
pcolor() – adds color to structure
flipud() – flips picture from upside down to rightside up
diag() – get the diagonals of a matrix. Important for Σ

Appendix B

```
% Jonathan Lu
% AMATH 482
% HW1 - PCA for face recognition
% 1/19/17

close all; clc

N = 39; % 39 faces
A = []; % data matrix

%% Load images in %%%

cd('/Users/Jonathan/Documents/UW/AMATH:MATH/AMATH482/HW1/CroppedYale')
cd('/Users/Jonathan/Documents/UW/AMATH:MATH/AMATH482/HW1/ExtendedYaleB')
currentDir = pwd;

```

Loading in the files:

```
for num = 11:N %for num = 1:N
    if num ~= 14 % for some reason YaleB14 doesn't exist?

        % process image file
        sampleDir = strcat(currentDir, '/', strcat('yaleB', sprintf('%02d', num)));
        cd(sampleDir)
        faceDir = strcat(sampleDir, '/*.pgm'); %% read all pgm files
        fullFile = dir(faceDir); % listing of files in directory

        for j = 1:length(fullFile)
            image = fullFile(j).name;
            u = imread(image);
            %[w,h] = size(u);
            u = double(u); % convert to double precision to make data matrix
            R = reshape(u, [], 1); % from matrix to col vector

            A = [A R];
        end
        cd('..') % move up to parent directory
    end
end
```

Computing SVD:

```
%% computing the SVD %%%

cd('/Users/Jonathan/Documents/UW/AMATH:MATH/AMATH482/HW1/CroppedYale')
load('dataMat1');
[U, S, V] = svd(A, 'econ');
```

Figure 1 – ranks = [1 25 50 100]

```
figure(1)
hold on

%rank = 100
S_r(101:end,101:end) = 0;
B = U*S_r*V';
subplot(2,2,4), face1 = reshape(B(:,1), 192, 168); pcolor(flipud(face1)), shading interp, colormap(gray);
xlabel('rank = 100')
%rank = 50
S_r(51:end,51:end) = 0;
B = U*S_r*V';
subplot(2,2,3), face1 = reshape(B(:,1), 192, 168); pcolor(flipud(face1)), shading interp, colormap(gray);
xlabel('rank = 50')
%rank = 25
S_r(26:end,26:end) = 0;
B = U*S_r*V';
subplot(2,2,2), face2 = reshape(B(:,1), 192, 168); pcolor(flipud(face2)), shading interp, colormap(gray);
xlabel('rank = 25')
%rank = 1
S_r(2:end,2:end) = 0;
B = U*S_r*V';
subplot(2,2,1), face1 = reshape(B(:,1), 192, 168); pcolor(flipud(face1)), shading interp, colormap(gray);
xlabel('rank = 1')

hold off
```

Figure 2 – ranks = [250 500 1000 2414]

```
48
49 -    figure(2);
50 -    hold on
51
52      %rank = 2414
53 -    S_r = S;
54 -    B = U*S_r*V';
55 -    subplot(2,2,4), face1 = reshape(B(:,1), 192, 168); pcolor(flipud(face1)), shading interp, colormap(gray);
56 -    xlabel('rank = 2414')
57      %rank = 1000
58 -    S_r(1001:end,1001:end) = 0;
59 -    B = U*S_r*V';
60 -    subplot(2,2,3), face1 = reshape(B(:,1), 192, 168); pcolor(flipud(face1)), shading interp, colormap(gray);
61 -    xlabel('rank = 1000')
62      %rank = 500
63 -    S_r(501:end,501:end) = 0;
64 -    B = U*S_r*V';
65 -    subplot(2,2,2), face1 = reshape(B(:,1), 192, 168); pcolor(flipud(face1)), shading interp, colormap(gray);
66 -    xlabel('rank = 500')
67      %rank = 250
68 -    S_r(251:end,251:end) = 0;
69 -    B = U*S_r*V';
70 -    subplot(2,2,1), face1 = reshape(B(:,1), 192, 168); pcolor(flipud(face1)), shading interp, colormap(gray);
71 -    xlabel('rank = 250')
72 -    hold off
73
```

Figure 3 – Singular value spectrum

```
100     % The magnitudes of the singular values, where the latter sigma has greater
101     % importance
102 -   figure(3)
103 -   semilogy(diag(S), 'ko')
104 -   xlabel('\sigma_x')
105 -   ylabel('Magnitude')
106 -   title('Magnitude of \sigma_x')
```