**Homework 1: Singular Value Decomposition on Extended Yale Faces B Database — Eigenfaces**

Daniel Nguyen

January 17, 2018

**Abstract**

We use the singular value decomposition on a matrix of pictures and explore the picture details captured by the modes produced in the matrix decomposition. We also compare the significance of each mode, and how details vary as we increase the number of modes active in image reconstruction.

**Contents**

**1 Introduction**

**2 Theory**

**2.1 The Singular-Value Decomposition**

**2.2 Variance and Co-Variance**

**3 Using MATLAB to apply the SVD Yale Eigenfaces**

**4 Results**

**5 Summary and Conclusions**

**Appendix A MATLAB functions used**

**Appendix B MATLAB Code**

**1 Introduction**

The information age has been characterized by the tremendous increase in accessibility and capacity for data. Along with our advances in in computing power and ability to collect data comes the important questions of how to analyze and deal with all of this data. To see a powerful example as to how mathematical methods can be applied to data analysis, we will explore a linear algebra concept known as the Singular-Value Decomposition, and we will see it in action given a database of images containing faces.

**2 Theory**

**2.1 The Singular-Value Decomposition**

Consider the matrix equation , where is a matrix and both x and are vectors. What this equation represents is a vector in space being transformed into another vector by a transformation matrix . A vector can be manipulated through space in two ways: rotation and dilation (or “stretching”).

The basic 2 x 2 matrix that represents rotation is:

which will rotate any vector by theta degrees. It is a unitary matrix, which means that each column has unit length, but it is more formally defined by its property that

where the inverse of the matrix is equivalent to the complex-conjugate transpose.

For dilation, the 2 x 2 matrix is:

Keeping in mind that all a transformation can do to a matrix is either rotate or stretch it, we can begin to explore the Singular-Value Decomposition (or SVD, for short). Consider a two-dimensional sphere (though this concept can be expanded to higher dimensions) with two orthogonal bases, and . When the matrix is applied to , say that what we get is a two-dimensional ellipse with orthogonal bases and . The represent singular values, and they are responsible for the stretching function of the transformation. The vectors are unitary vectors which comprise the bases of the transformed system AS.

The transformation of an individual vector in this case can be represented by the matrix equation

But generalizing in all dimensions, if we set V equal to the set of all v vectors and U to the set of all u vectors and E to be a diagonal matrix containing all of the singular values arranged from greatest to least, we can rewrite the above system as

If we multiply both sides by the transpose of , which we will call we will get that

Where is the matrix containing the new set of orthogonal bases, is a diagonal matrix containing the singular values ordered from greatest to least, and is the original unitary set of orthogonal bases. This is the singular-value decomposition. A great thing about this decomposition is that it is guaranteed for any matrix , unlike other decompositions such as eigen-decompositions.

**2.2 Variance and Covariance**

With the equation of the SVD in mind, we will apply this concept to two vectors and which represent different streams of recorded data from some phenomenon. I will introduce two statistical concepts:

1) Variance, which refers to the spread of the data set (how far the values are in relation to the mean), which will tell us what’s going on in a matrix.

2) Covariance, which calculates the correlation between two variables and tells us how they change in relation of one another over time.

The equations for variance and covariance are given as follows for a and b:

For covariance, we take the dot product of one vector and the other’s inverse. If the two vectors are orthogonal, then they have nothing to do with each other (covariance = 0); otherwise, if they are parallel, they are completely covariant.

Let’s generalize this to more vectors. Say we have a matrix of vectors called X. We will apply the above formula on X to find the variance and covariances:

And we will get a matrix that looks like

Notice of relates to the SVD. We can relate this to the fact that , which will give us the singular values, .

What the singular values will give us is the significance of each mode in the original -dimensional data. Say we recorded a ball oscillating vertically on a spring. If we took perfect measurements without various cameras, we would only have one non-zero element in our sigma matrix. This would represent just one coordinate-axis truly at work — the -axis since all the ball is doing is moving up and down.

Our aim in using the SVD to data is to get the best coordinate system possible in order to represent the data, since the SVD will give us the guaranteed very best basis functions possible. We can think of this as compressing the n-dimensional data to low rank structures, so that we have low-dimensional representations of what we want to analyze.

This is beneficial as it filters out insignificant dimensional factors. Say for example that we have a spring-mass system, and we want to record its motion using 3 cameras. Assuming that the camera interprets the stream of data in two dimensions (horizontal and vertical), we get a total of six dimensions of data. But we only want to analyze the one-dimensional motion of the ball. The SVD will help us do that. If we took perfect measurements, the singular value matrix would give us just one non-zero singular value, and we could use that in order to get rid of the extraneous data which is basically just redundancy.

A function can be represented by a set of bases, . There are many methods to do this and the SVD is one of them. In many cases, the SVD is preferable to other methods of approximation, such as Taylor Expansions, or Cosine or Fourier transforms. Consider how Fourier transforms can only perfectly match the function we want through infinite iterations. For computational efficiency, we want to take the smallest N to approximate our function.

**3 Using MATLAB to apply the SVD Yale Eigenfaces**

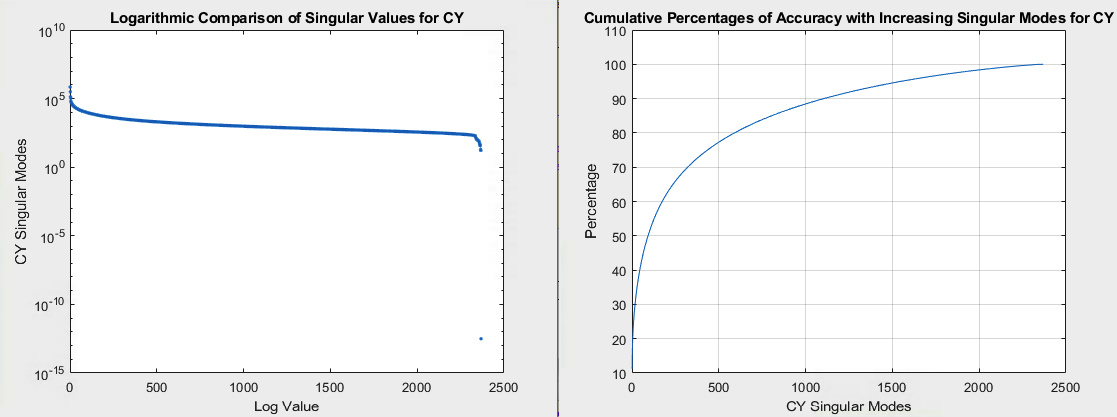
Given two sets of data containing faces of Yale students (an uncrossed data set vs. a cropped set), we will construct a matrix of images from the data and apply the SVD on them.

To construct the matrix for one set, we iterate through the folder containing the images and extract each image into our script using the imread() function in MATLAB. We then vectorize our image by using image(:), and then we concatenate it on to our main matrix as a new column. For the cropped set, the images were 192 x 168 pixels, and there were 2368 images in total. For the uncrossed set, the images were 243 x 320 pixels, and there were 165 images. We got two matrices of sizes 32256 x 2368 and 77760 x 165, respectively.

For each matrix, we now apply the SVD, using the line [U, S, V] = svd(image, ‘econ’) in MATLAB (the ‘econ’ is for computational efficiency; the full SVD tacks on extra values in order to make our U, S and V matrices square, but that is not necessary).

**4 Analysis of Results**

First, for both the cropped and uncrossed data sets, I plotted all of their singular values on a logarithmic scale in order to more accurately see the significance of each mode, and determine whether the initially “zero-looking” modes are really as close to zero as they seem.



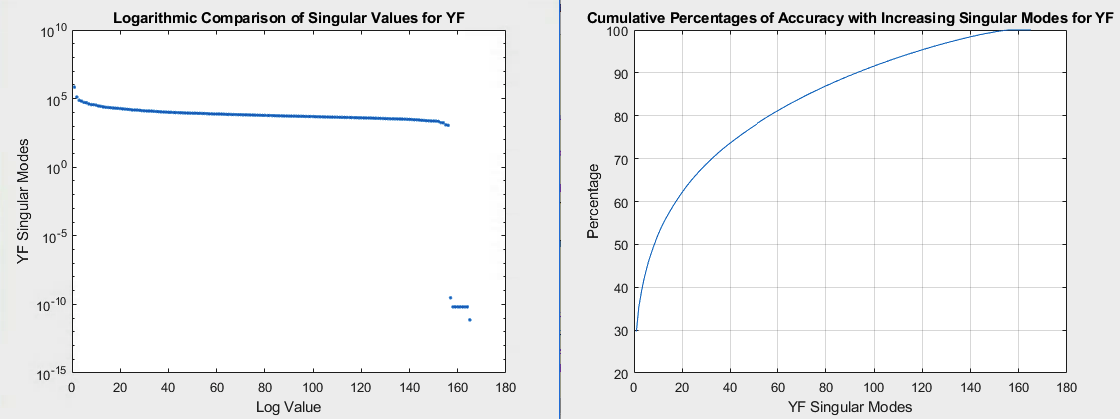


Figure 1: Logarithmic significance of singular values and increasing levels of accuracy for Cropped Yale and Uncropped Yale

For Cropped Yale, which I will call CY for short, we can see how with just 93 modes (just 4% of the total number of modes at play), we have managed to capture 50% of the full image! Though the change in accuracy gets less steep as we increase the number of modes, we manage a 90% accuracy with just 1105 modes, which is less than half of the total number of modes that we have.

For the uncrossed data set (which I will call YF)), we achieve similar results. Using just 9 modes (about 5% of the total number of modes), we get 50% of the data, and we get 90% with 93 modes (56% of the modes).

Let’s reconstruct the 64th image in our cropped data set by constructing various matrices using varying numbers of modes from our SVD matrices:

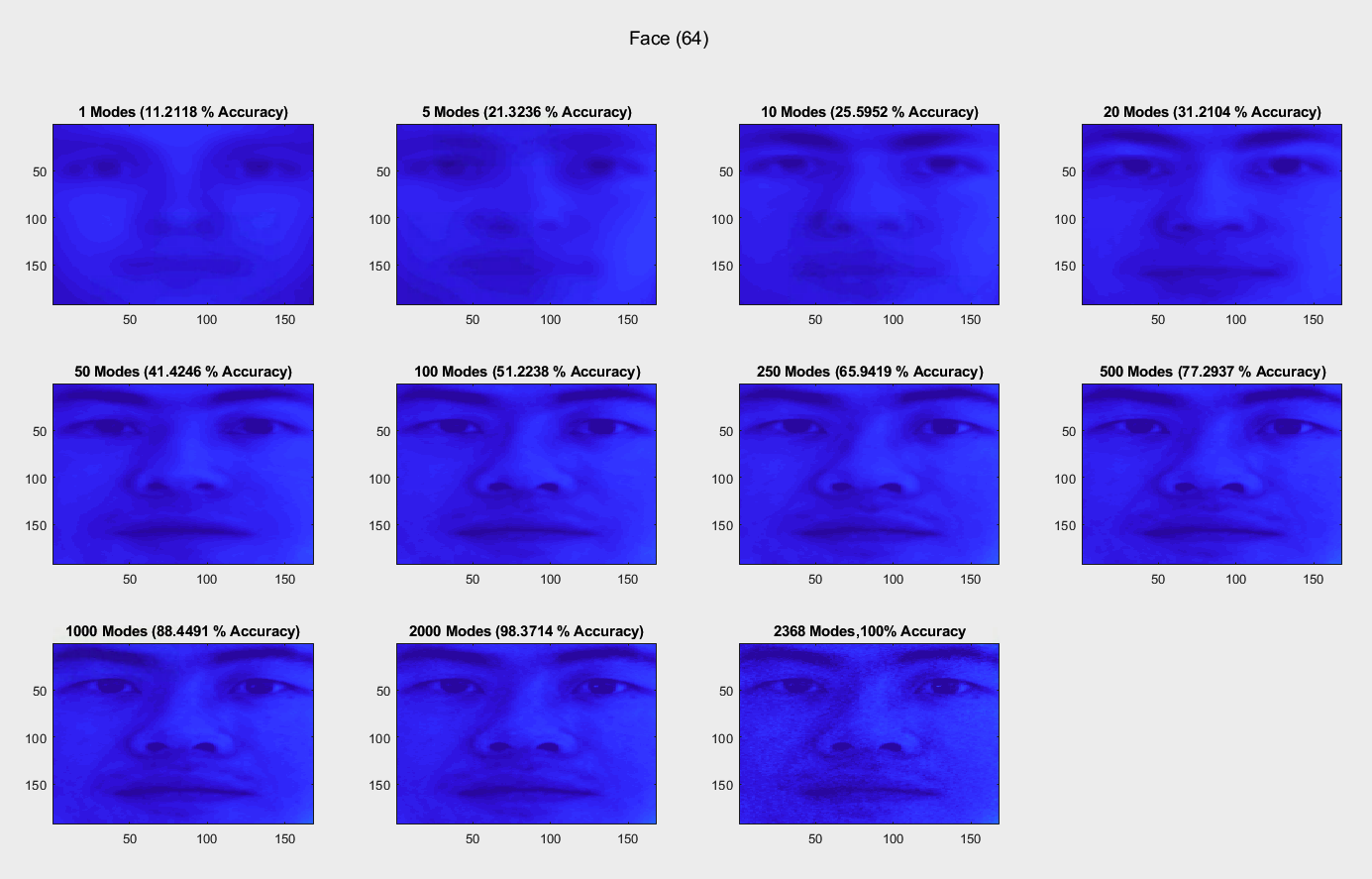


Figure 2: Reconstruction of face 64 in the Cropped Yale dataset.

Using just one mode, we can see what the SVD picked up as very common features right off the bat – eyes, nose, mouth. As we increase the number of modes, more detail is added to the already generic-looking features, and we will start to see shadows, eyebrows, etc. Notice how more dark grains are picked up when the remaining modes of the SVD are used. This tells us that those less significant modes really just capture the noise in the data. So in some cases, it can be beneficial to select an appropriate number of modes (though we are sacrificing accuracy to the original source) in order to filter out unwanted noise.

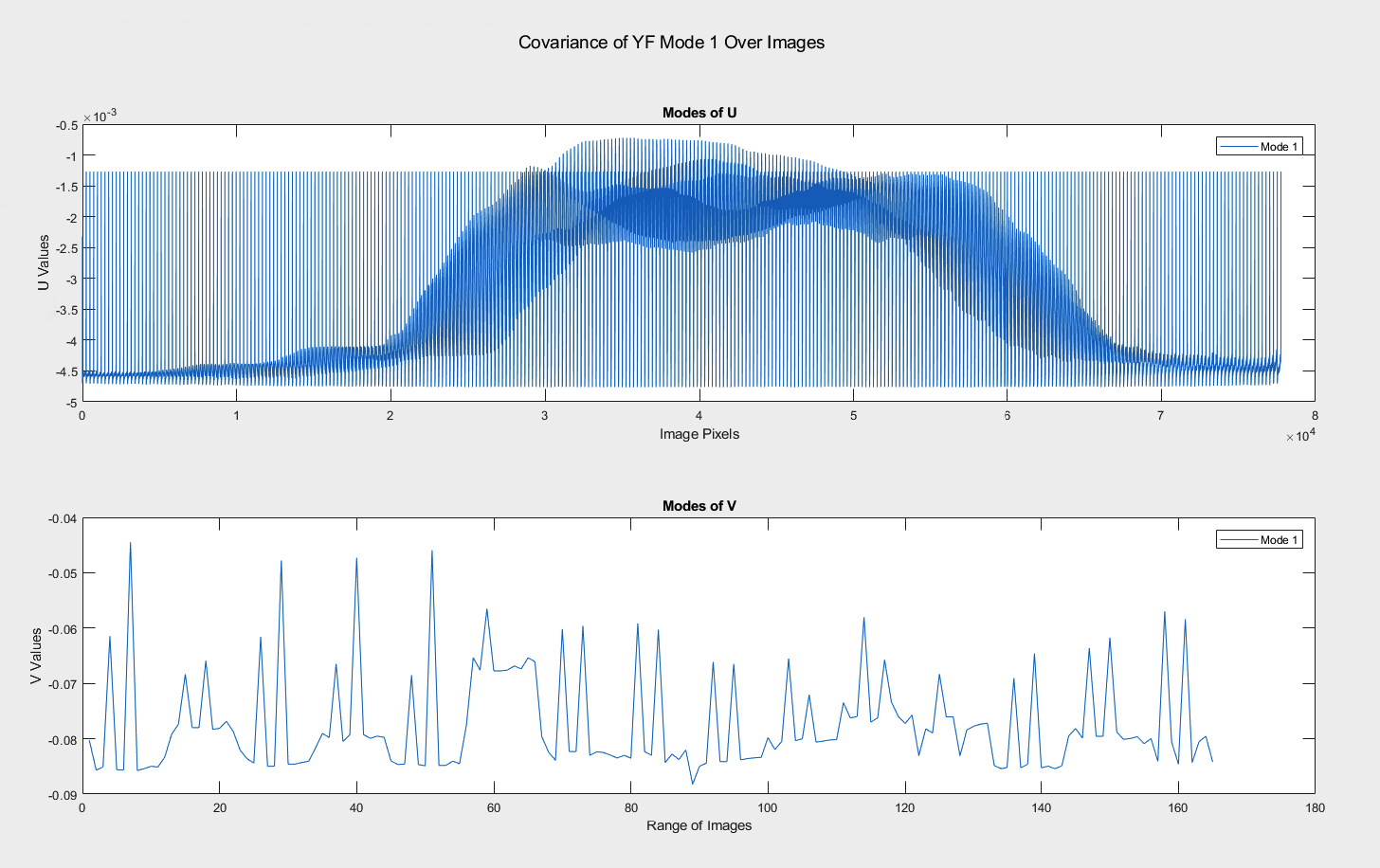
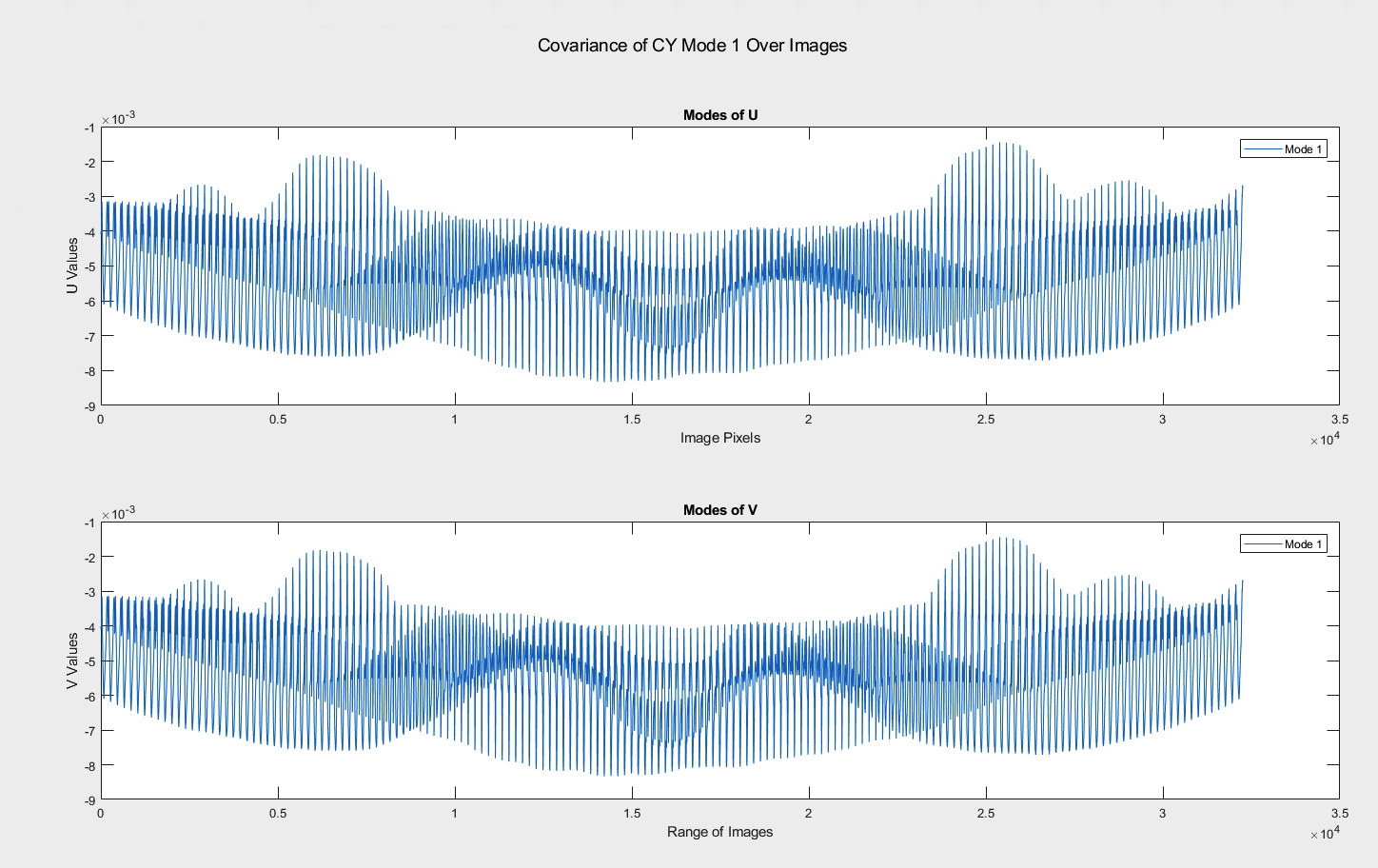
Now let’s examine what will be found when we analyze a face from the uncropped set:



Figure 3: Reconstruction of face 65 in the YF database

We see that using the first mode does not give us as clear facial features as in the cropped data set, where the faces were positioned approximately in the same position in each photograph. In the uncropped set, there is variance – the face could be more towards the right or the left, so there is no guarantee that the eyes will always be in the same position for instance. But we see from the first mode that hair, face outline, and neck shadow are very common in the pictures, and are pictured to us. But even using five moves, the eyes and the mouth are still not yet made clear to us as was the case in the CY data set.

Let us now look at what was picked up by each mode in the U matrix over time.



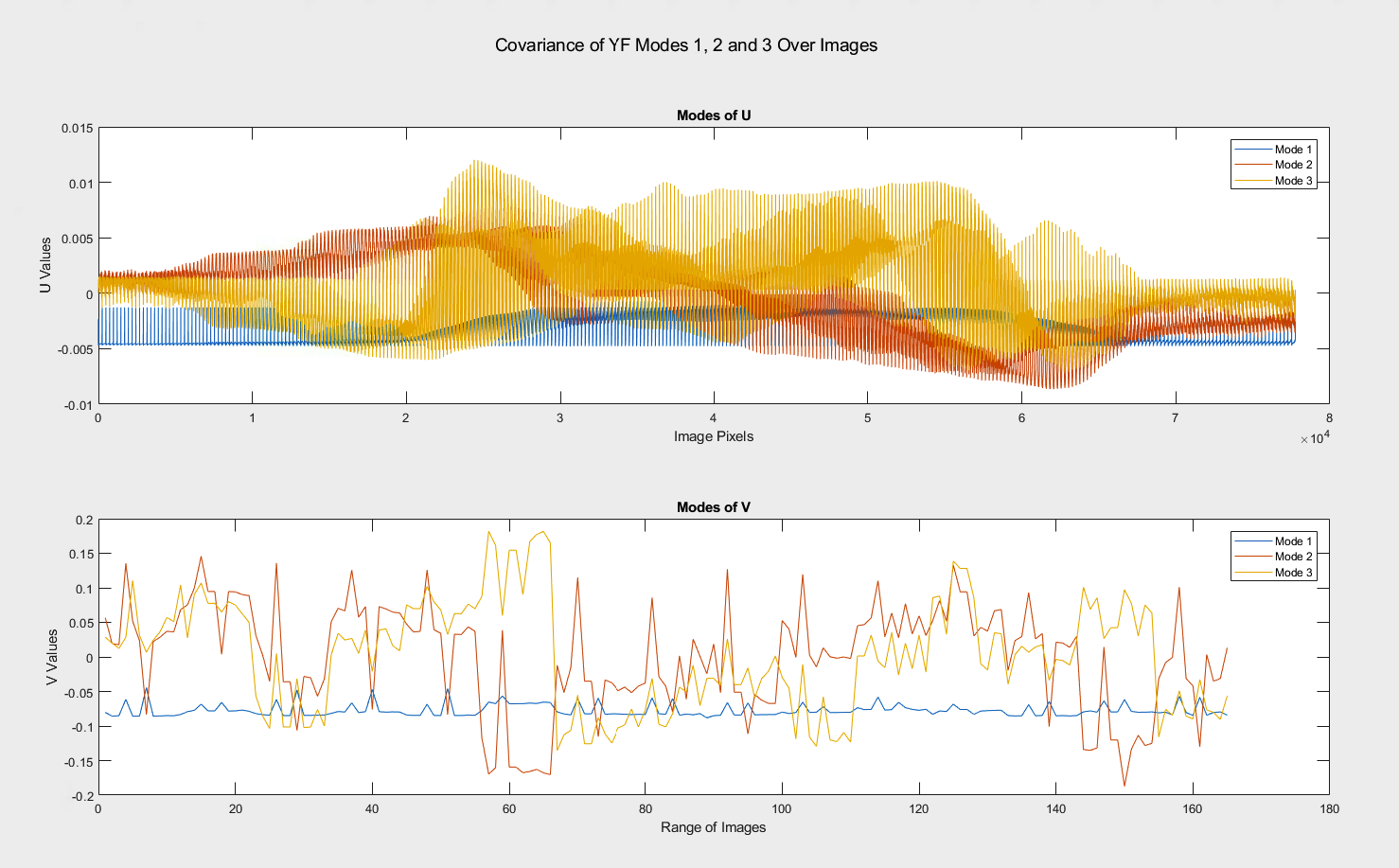
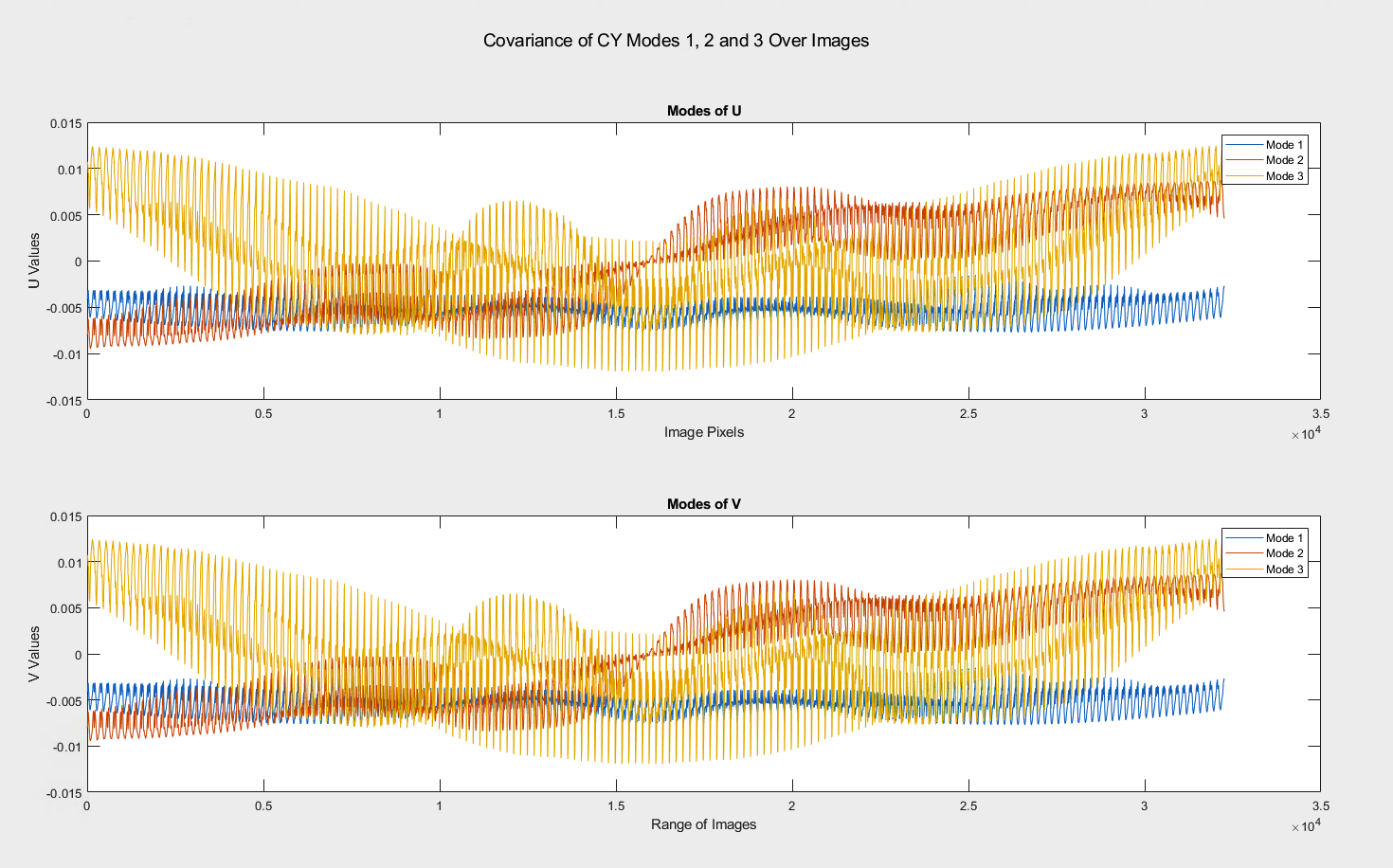


Figure 4: The first column of U plotted against each pixel and the first three columns of U plotted over one another for both CY and YF databases

We can see that as we approach the center, the values of U get higher, since it is in the center that the facial features are located in, and the beginning and end of the columns represent the blank upper-left and lower-right corners of the pictures, respectively, where nothing is going on.

**5 Summary and Conclusions**

As we have seen, the Singular-Value Decomposition is a very powerful tool when it comes to computational efficiency in data analysis. Though we were given thousands of pixels of raw data through the images, the SVD was able to pick up the bare-bones structure of a face by going through every single image vector in the full matrix of images, and it was able to give us prominent human facial features such as eyes, noses, and mouths as indicated by its largest singular values and orthogonal bases given by the SVD. But we have also seen scenarios where the SVD could have fallen short; namely when there is less covariance among our varying pieces of data, which was more pronounced in the uncropped data set. Nevertheless, with the aid of other data analysis theory and techniques, the SVD can be (and have been) applied to great ends.

**Appendix A MATLAB Functions Used**

I had to use cd to get the desired folders containing the data, and I used a for loop applying the imread() command to each image file in order to get the double matrix representing the pixels. I then vectorized those matrices and concatenated onto the main matrix I initialized to hold all of the images as column vectors.

I used the semilogy() command to create a plot of logarithmic significance of each singular value, and various plot label commands such as title(), xlabel() and ylabel() to identify my graph and its axes.

To make the graph of faces, I used the image() command, which creates an image from a matrix containing the pixel values, which I would obtain by taking the column at the index of the image that I want (column 65 of YFimages if I wanted image 65 from the uncropped dataset for example) and then I would use the reshape() command in order to get the matrix to represent the picture in its original dimension.

**Appendix B MATLAB Code**

%% Daniel Nguyen, AMATH 482, Assignment 1

%%% Cropped Yale

clear all;

close all;

cd '/Users/danielnn/Documents/MATLAB/CroppedYale'

CYfiles = dir(pwd);

CYcount = 1;

CYimages = [];

for i = 4:length(CYfiles)

  direct = strcat(pwd, '/', CYfiles(i).name);

  cd(direct);

  CYimgs = dir(direct);

  for j = 3:length(CYimgs)

  CYimg = imread(CYimgs(j).name);

  CYimages(:,CYcount) = CYimg(:);

  CYcount = CYcount + 1;

  end

  cd ..

end

%%

[CY\_U, CY\_S, CY\_V] = svd(CYimages, 'econ');

%%

CYsing\_val = diag(CY\_S);

% plot(sing\_val)

figure('Name', 'Singular Values Plotted on a Log Scale for CY', 'IntegerHandle', 'off')

semilogy(CYsing\_val, '.')

title('Logarithmic Comparison of Singular Values for CY')

xlabel('Log Value')

ylabel('CY Singular Modes')

CYsing\_val\_percent = CYsing\_val \* 100 / sum(CYsing\_val);

for i = 2:length(CYsing\_val)

  CYsing\_val\_percent(i) = CYsing\_val\_percent(i) + CYsing\_val\_percent(i-1);

end

figure('Name', 'Cumulative Accuracy for CY', 'IntegerHandle', 'off')

plot(CYsing\_val\_percent)

title('Cumulative Percentages of Accuracy with Increasing Singular Modes for CY')

xlabel('CY Singular Modes')

ylabel('Percentage')

grid on

%%

figure(3)

subplot(2,1,1), plot(CY\_U(:,2368))

legend('Mode 2368')

title('Modes of U')

xlabel('Image Pixels')

ylabel('U Values')

subplot(2,1,2), plot(CY\_U(:,2368))

legend('Mode 2368')

title('Modes of V')

xlabel('Range of Images')

ylabel('V Values')

suptitle('Covariance of CY Mode 2368 Over Images')

%%

 CY\_modes = [1 1 5 10 20 50 100 250 500 1000 2000 2368];

 CY\_pic = 1;

 figure('Name', strcat('Face ', int2str(CY\_pic)), 'IntegerHandle', 'off')

for i = 2:length(CY\_modes)

  CY\_m\_num = CY\_modes(i);

  CY\_A1 = CY\_U(:,1:CY\_m\_num)\*CY\_S(1:CY\_m\_num,1:CY\_m\_num)\*CY\_V(:,1:CY\_m\_num).';

  figure\_title = strcat(int2str(CY\_modes(i-1)), ' Modes');

  figure\_title = strcat(figure\_title, ' (', num2str(CYsing\_val\_percent(CY\_modes(i-1))), ' % Accuracy)');

  title(figure\_title)

  subplot(3,4,i-1)

  image(reshape(CY\_A1(:,CY\_pic),192,168));

end

title(strcat(int2str(2368), ' Modes', ', ', '100% Accuracy'));

suptitle(strcat('Face (', int2str(CY\_pic), ')'))

%%

cd '/Users/danielnn/Documents/MATLAB/yalefaces'

YFfiles = dir(pwd);

YFcount = 1;

YFimages = [];

YFimgs = dir();

for i = 3:length(YFfiles)

  YFimg = imread(YFimgs(i).name);

  YFimages(:,YFcount) = YFimg(:);

  YFcount = YFcount + 1;

end

%%

[YF\_U,YF\_S, YF\_V] = svd(YFimages, 'econ');

%% Yale Faces

YFsing\_val = diag(YF\_S);

% plot(sing\_val)

figure('Name', 'Singular Values Plotted on a Log Scale for YF', 'IntegerHandle', 'off')

semilogy(YFsing\_val, '.')

title('Logarithmic Comparison of Singular Values for YF')

xlabel('Log Value')

ylabel('YF Singular Modes')

YFsing\_val\_percent = YFsing\_val \* 100 / sum(YFsing\_val);

for i = 2:length(YFsing\_val)

  YFsing\_val\_percent(i) = YFsing\_val\_percent(i) + YFsing\_val\_percent(i-1);

end

figure('Name', 'Cumulative Accuracy for YF', 'IntegerHandle', 'off')

plot(YFsing\_val\_percent)

title('Cumulative Percentages of Accuracy with Increasing Singular Modes for YF')

xlabel('YF Singular Modes')

ylabel('Percentage')

grid on

%%

figure(3)

subplot(2,1,1), plot(YF\_U(:,1:3))

legend('Mode 1', 'Mode 2', 'Mode 3')

title('Modes of U')

xlabel('Image Pixels')

ylabel('U Values')

subplot(2,1,2), plot(YF\_V(:,1:3))

legend('Mode 1', 'Mode 2', 'Mode 3')

title('Modes of V')

xlabel('Range of Images')

ylabel('V Values')

suptitle('Covariance of YF Modes 1, 2 and 3 Over Images')

%%

YF\_modes = [1 1 5 10 20 50 100 150 165];

YF\_pic = 1;

figure('Name', strcat('Face ', int2str(YF\_pic)), 'IntegerHandle', 'off')

for i = 2:length(YF\_modes)

  YF\_m\_num = YF\_modes(i);

  YF\_A1 = YF\_U(:,1:YF\_m\_num)\*YF\_S(1:YF\_m\_num,1:YF\_m\_num)\*YF\_V(:,1:YF\_m\_num).';

  figure\_title = strcat(int2str(YF\_modes(i-1)), ' Modes');

  figure\_title = strcat(figure\_title, ' (', num2str(YFsing\_val\_percent(YF\_modes(i-1))), ' % Accuracy)');

  title(figure\_title)

  subplot(2, 4, i-1)

  image(reshape(YF\_A1(:,YF\_pic),243,320));

end

title(strcat(int2str(165), ' Modes', ' (', '100% Accuracy)'));

suptitle(strcat('Face (', int2str(YF\_pic), ')'))