Principle Component Analysis of Periodic Motion

Maxwell Weil

Abstract

In this homework, we will be exploring the use of singular value decomposition (SVD) and principle component analysis (PCA) on real-life datasets. First, we will review the concepts and strategies used in this investigation, in order to establish a theoretical background. Next, we will demonstrate SVD and PCA on several videos, using MATLAB software. Finally, a discussion of the results will aim to delve into how these techniques can be practically utilized. Overall, a better understanding of SVD and PCA will be realized through this work.

I. Introduction

With ever increasing amounts of stored data in the world, one has to wonder how we can make use of these massive datasets. Traditional analytical methods will work for low dimensional data, such as 2D plots, which can allow for easy visual interpretation of the relationship between two variables. However, high dimensional data becomes increasingly hard to handle. Attempting to find patterns and relations between 20 variables is not nearly as simple as before. To combat this issue, statistical techniques like SVD and PCA have been developed to better break down large, complex datasets.

SVD is a mathematical procedure that can divide a given matrix into three basic components. These can then be used to approximate the given dataset, using a lower rank matrix than before. Effectively, SVD enables us to calculate the importance of various ranks of a given matrix. This, in turn, can be used alongside PCA, which allows us to reduce the computational complexity of our data by only considering the most important components, which are found through SVD.

In this work, we will be looking at using aspects of SVD and PCA to analyze multiple videos taken of an object in period motion. These strategies will help us visualize how the object is moving in multiple dimensions, granting a comprehensive understanding of the data. Before this can be done, SVD and PCA will be explored in more numerical detail.

II. Theoretical Background

SVD is one of the key aspects of this assignment, so understanding its innerworkings is very important. To introduce this topic, we will use an example from [1] about the rotation and scaling of a matrix in two dimensions. Equation 1 below shows the standard rotation matrix in two dimensions. When a vector, **x**, is multiplied by **A**, it is rotated counterclockwise by an angle, θ.

$$A = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \quad \text{EQ. 1}$$

Similarly, this same vector can be scaled using the matrix shown in equation 2. When multiplied by **B**, the vector will remain in the same orientation, but either increase or decrease in length along that direction by a factor of α.

$$B = \begin{bmatrix} \alpha & 0 \\ 0 & \alpha \end{bmatrix} \quad \text{EQ. 2}$$

With this, we can better explain how the previously mentioned components of the SVD of a matrix work. The reduced SVD of a matrix **C** is given in equation 3 below. The full SVD will not be covered, though it can be simplified to this reduced form.

$$C = U\Sigma V^* \qquad U \in \mathbb{C}^{m*m} \qquad \Sigma \in \mathbb{R}^{m*n} \qquad V \in \mathbb{C}^{n*n} \qquad \text{EQ. 3}$$

U and V are both unitary matrices, and Σ is a diagonal matrix. U, Σ, and V, each contain important information about **C** in this equation. Similar to the case in equations 1 and 2, these matrices all act as basic rotation or scaling matrices. If a vector **x** is multiplied by **C**, then U and V both act to rotate **x** into a new space. Likewise, Σ scales **x** by a given amount. Essentially, the transformation of **x** by **C** can be simplified into three different transformations of either rotation or scaling.

While the basic properties of the SVD of a matrix seem rather mundane, they enable important analytical techniques. An essential properties of the matrix Σ, is that the values along the diagonal occurring in descending order. These values can be used to approximate how much of the data in **C** is captured by only a few columns of data in U and V. This idea brings us to PCA and proper orthogonal decomposition (POD).

PCA and POD are essentially the same concept. Both aim to reduce the dimensionality of data, by considering only the most important pieces of information. This is done through approximating a function using the expansion shown in equation 4.

$$f(x,y) \approx \sum_{j=1}^{N} a_j(y)\,\phi_j(x) \qquad \text{EQ. 4}$$

In this equation, N is the number of modes, a_j is the scaling coefficients, and φ_j is an orthonormal basis. Briefly, this expansion converts a given function into an orthonormal basis of vectors. Note that this equation is similar to the Fourier transform, where a function is transformed onto a new basis of sines and cosines.

Combining the idea of PCA/POD and SVD gives us a simple method for capturing the primary components of a dataset. As we know, SVD produces two orthogonal bases of a matrix, U and V, as well as a set of scaling coefficients, Σ. Since the values in Σ are descending, we also know that each of these scaling coefficients descend in importance. Using this information, we can estimate a set of data stored in a matrix, **C**, by using the expansion above on only the first few modes. This will produce a set of data computed from only the first few scaling coefficients and columns of the SVD, dramatically reducing the complexity of our data.

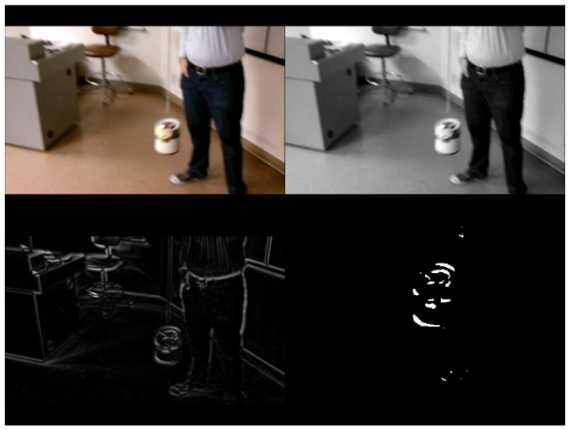


Figure 1. Montage of video processing data. Original video footage frame is shown in the top left, with grayscale version shown in the top right. The bottom left image shows the frame subtracted from the subsequent frame, where moving regions appear as white. The final image in the bottom right shows a thresholded and binarized version of this, cropped around the region of interest. Most of the white pixels here are on the object, so averaging their position gives the approximate center of mass.

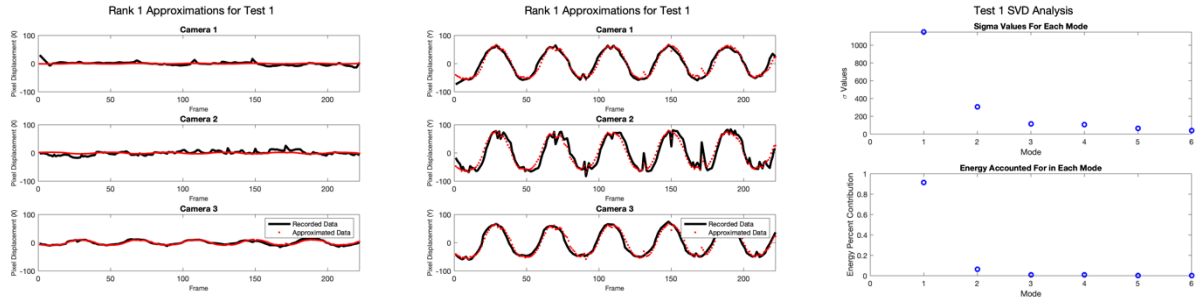


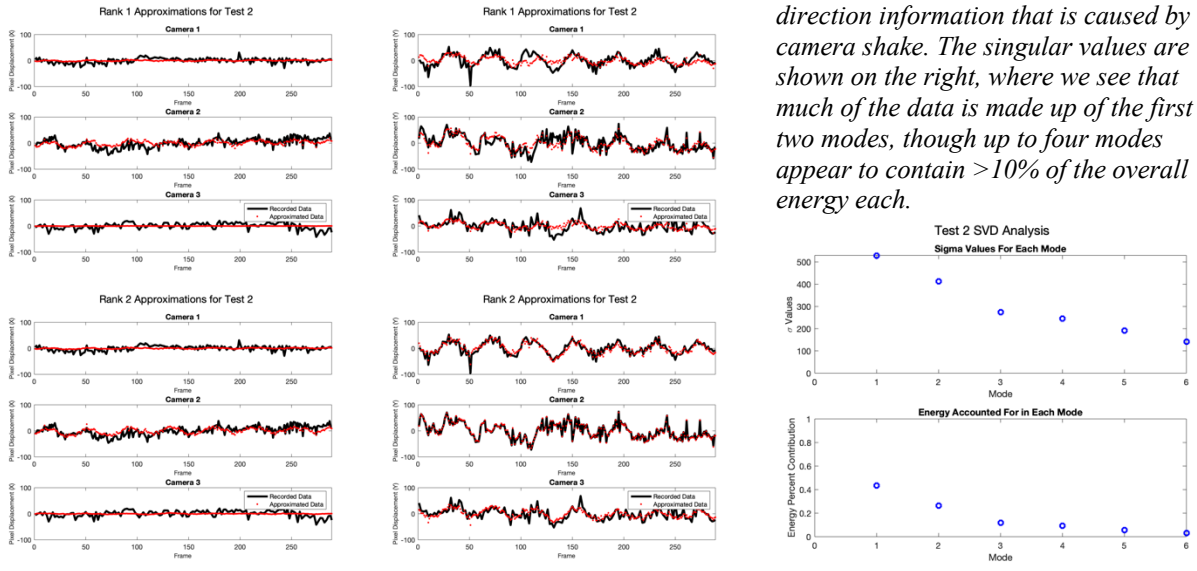
Figure 2. The rank one approximation for the first test is shown in both the x (left) and y (center) directions. Each plot shows the approximation compared to the actual data for each camera. The singular values are shown on the right, where it is clear that the first mode makes up ~90% of the total energy of all six modes.

III. Algorithm Implementation and Development

In this work, we will be looking at video footage taken from several cameras, that look at the periodic movement of an object on a spring. The concepts previously reviewed have been used to analyze this movement and explore its complexities. However, before discussing these techniques, an overview of the preprocessing of the video data is essential.

Three videos from different cameras were imported for a given test (four total tests were conducted), with each video being processed separately (line 29). First, a given video was converted from RGB to grayscale (lines 40-42). Next, the program user is asked to select a rectangular area of interest around the approximate space where the object will be moving (lines 45-47). This was done to reduce interference from any background activity while trying to track the desired object. Next, each frame is subtracted from the next. If sequential frames are identical, this would produce an empty (all zeros) frame. Any difference between the frames, such as moving objects, will be highlighted, providing for a simple tracking method. The resulting image is the thresholded and binarized, in an attempted to remove any additional small motions (lines 63-68). The primary steps of this process are illustrated in Figure 1.

Figure 3. The rank one and two approximations for the second test are shown in both the x (left) and y (center) directions. While the rank one approximation accounts for most of the variance in the y-direction, the rank two approximation adds in additional x-direction information that is caused by camera shake. The singular values are shown on the right, where we see that much of the data is made up of the first two modes, though up to four modes appear to contain >10% of the overall energy each.



The center of all pixels with value 1 (any motion found), is then determined (lines 72-76). This allows for the approximate calculation of the center of mass of the moving object. This information is then stored for further processing (lines 88-93).

Following the initial preparation, further steps are taken to clean up the data. It was found that the method of subtraction and binarizing sequential frames occasionally resulted in no motion being captured. This resulted in nearly half of the frames having no center of mass. To overcome this issue, the missing data points were interpolated (lines 104-131).

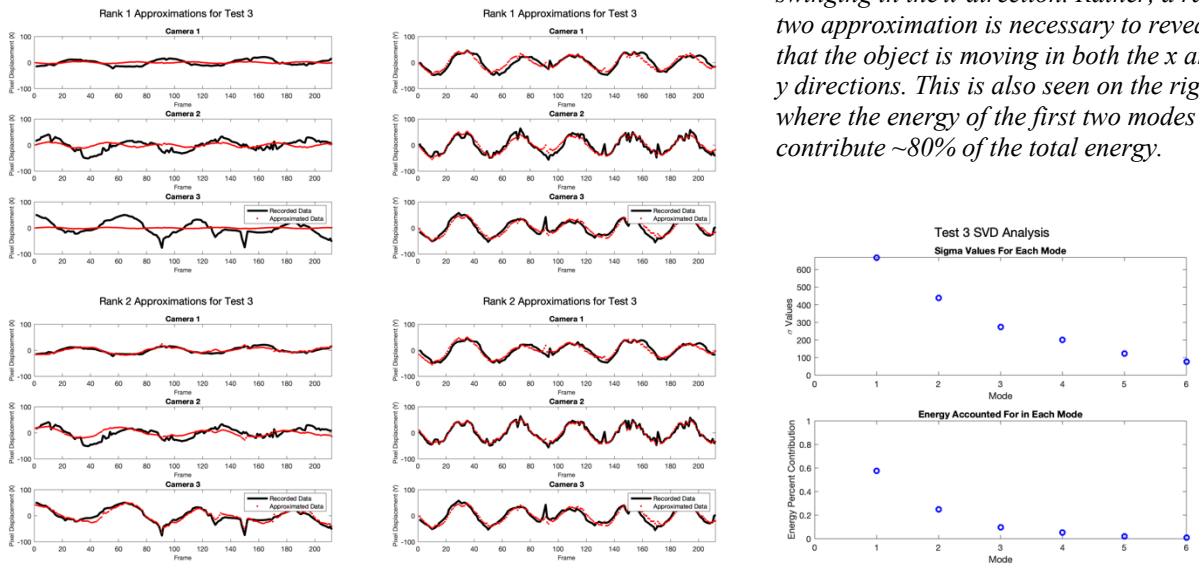
Perhaps the most challenging issue to address was that none of the videos used were properly synced together. Without synchronization, PCA/POD would tell us little about the motion of the object. Because each test had the object in periodic vertical motion, this could be used to align the data properly. However, it should be noted that this is somewhat contradictory to the purpose of this paper, as to analyze the primary motion of the object, the most important motion had to first be used. Regardless, the first handful of data points from each camera was searched for a minimum value (lines 123-127). The minimum indices were then aligned, and excess data trimmed from the ends of each video (lines 140-149). The resulting data was then arranged into a matrix for computing the SVD (line 155-175). Before this, however, the mean of each row was subtracted to give an average of 0 (line 152). This is to avoid skewing the data points to the area of the video frame at which they occurred.

Following the computation of the SVD, the various major components of the matrices were analyzed (lines 177-213). Using PCA/POD, different ranking approximations were made for each test.

IV. Computational Results

From our investigation, several interesting phenomena were observed. In the first test, shown in Figure 1, it was found that the first singular value in Σ accounted for most of the motion seen. The first singular value mode accounted for ~90% of the energy of all six modes, clearly showing that the rank one approximation was a sufficient estimate of the observed motion.

Figure 4. The rank one and two approximations for the third test are shown in both the x (left) and y (center) directions. In this test it is clear that the rank one approximation fails to capture the periodic motion caused by swinging in the x-direction. Rather, a rank two approximation is necessary to reveal that the object is moving in both the x and y directions. This is also seen on the right, where the energy of the first two modes contribute ~80% of the total energy.

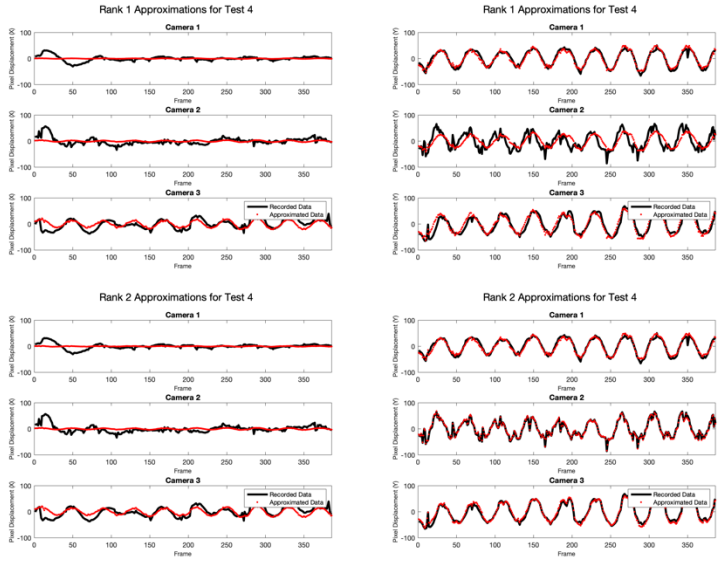


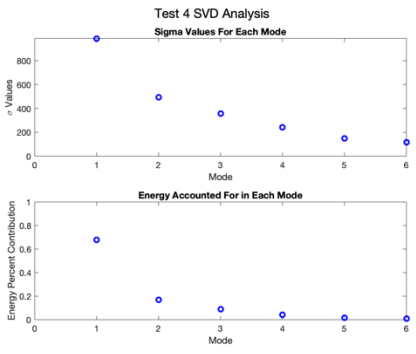
In the second test, we explored how PCA is impacted by noisy data. Figure 3 illustrates how shaking cameras tended to interfere with the major components of motion. This is because if the camera is moving significantly in a different direction than the object (periodically, as well), this motion will register as another principle component. This is why the rank 2 approximation best captured the majority of the data in this scenario, though much of the motion in both directions was highly noisy.

In test 3, we have an additional motion in the x-direction caused by the swinging of the object, as seen in Figure 4. It was found that the rank 1 approximation of this test captured the y-direction motion but failed to account for much of the movement in the x-direction. The rank 2 approximation was able to represent both directions much better. This demonstrates how different components of movement impact how well our rank approximations work. With multidimensional movement, we can't reduce this to a single rank approximation, as some dimensional motion will be lost. Essentially, at least N ranks are necessary to fully capture a N-dimensions of orthogonal motion.

While this concept is theoretically sound, test 4 proved to be somewhat troublesome. The object of interest was swinging, bouncing, and rotation, in all independent directions. However, the PCA analysis did not appear to take into account the rotating movement, as shown in Figure 5. This is most likely due to the method of video processing used. While the center of mass was used in the SVD calculation, the direction the object was facing was largely ignored. Using the algorithm previously detailed on a stationary object that is only rotating would find movement but would struggle to find a change in the center of mass. Certain details, such as moving labels, would be picked up and may help find rotation. But combined with noisy camera footage and additional motion, this information would most likely be lost.

Figure 4. The rank one and two approximations for the fourth test are shown in both the x (left) and y (center) directions. The overall results of this test appear similar to the third test, despite having additional rotational motion. However, this did not appear to be captured by our process, as the first two modes contribute ~90% of the total energy.





V. Summary and Conclusions

In this work, we have been able to explore how highly mathematical techniques like SVD and PCA can be applied to analyze real-life data. Using these methods in practical ways, we've been able to better understand how data variance can be captured and dimensionality reduced. Through this, we've seen firsthand how more effective computation can be conducted on large and daunting datasets. The methods demonstrated here can be utilized in a variety of ways, allowing for fast analysis of the most essential components of data.

References

[1] J. Nathan Kutz, *Data-Driven Modeling & Scientific Computation: Methods for Complex Systems & Big Data.* Oxford University Press, 2013.

Appendix A. MATLAB Functions Used

`eval(exp)` – Evaluates a given expression, in this assignment, this was used to read specific video files that were previously loaded into the workspace.

`rgb2gray(IM)` – Converts an RGB image to grayscale, while retaining brightness and contrast.

`getrect` – Allows the user to click and drag a rectangular selection on the current axes. The information is saved as the coordinates and size of the rectangle drawn.

`imabsdiff(IM1,IM2)` – Subtracts each corresponding pixel/element in IM2 from IM1 and outputs the absolute value of the result.

`imbinarize(IM,T)` – Sets all values of a grayscale image, IM, to 1 if they are greater than T, and 0 otherwise.

`fillmissing(X,'linear','SamplePoints',Y)` – Interpolates missing (NaN) data in X at the corresponding sample point from Y.

`[U,S,V] = svd(A, 'econ')`– Computes the reduced SVD of a matrix A, outputs the appropriate matrices.

`diag(X)` – Returns the values along the diagonal of a matrix, X.

`sgtitle('text')` – Adds a title to a grid of subplots.

Appendix B. MATLAB Code

```
% Maxwell Weil
% AMATH 482
% HW 3
clear; close all; clc;

% Setting number of camera and test number
numCams = 3;
dataSet = 3;

% Initializing name variables
camNames = strings(1,numCams);
vidNames = strings(1,numCams);

% Creating variable anmes
for i = 1:numCams
    camNames(i) = ['cam',num2str(i),'_',num2str(dataSet),'.mat'];
    vidNames(i) = ['vidFrames',num2str(i),'_',num2str(dataSet)];
end

% Initializing variables
numFrames = zeros(3,1);
camData = cell(1,3);
dataLen = zeros(1,3);
startIdx = zeros(1,3);

% Looping through each camera to find object location for each video
for i = 1:numCams

    % Loading in and setting current camera data
    load(camNames(i))
    currentVid = eval(vidNames(i));

    % Setting number of frames for current video
    numFrames(i) = size(currentVid, 4);

    % Initializing grayscale images
    grayImage = zeros(size(currentVid,1,2,4),'uint8');

    % Converting each frame of current video to grayscale
    for j = 1:numFrames(i)
        grayImage(:, :, j) = rgb2gray(currentVid(:, :, :, j));
    end

    % Selecting region of interest using rectangular area
    figure
    imshow(grayImage(:, :, 50))
    rectInfo = getrect;

    % Rounding pixel values to prevent any errors
    xCoor = round([rectInfo(1),rectInfo(1)+rectInfo(3)]);
    yCoor = round([rectInfo(2),rectInfo(2)+rectInfo(4)]);
    close

    % Initializing data and image difference matrices
    data = zeros(numFrames(i)-1,2);
    imageDiffs = zeros(size(currentVid,1,2),'uint8');

    % Looping through number of frames to find object location in each
    for k = 1:numFrames(i)-1

```

```
61            % Calculating difference between images, in order to
62            % highlights moving objects
63            imageDiffs(:, :) = imabsdiff(grayImage(:, :, k), ...
64                grayImage(:, :, k+1));
65
66            % Binarizing image to reduce movement noise
67            bw = imbinarize(imageDiffs(yCoor(1):yCoor(2), ...
68                xCoor(1):xCoor(2)),0.3);
69
70            % Find indices of nonzero values and averaging to determine
71            % location of moving object
72            [row, col] = find(bw);
73            if ~isempty(row)
74                data(k,2) = mean(row);
75                data(k,1) = mean(col);
76            end
77
78            % Plotting binary image along with calculated object location
79            imshow(bw)
80            hold on
81            plot(data(k,1),data(k,2),'rx','MarkerSize',20)
82            hold off
83            drawnow
84        end
85        close
86
87        % Storing object location and rotating data from camera 3
88        camData{i} = data;
89        if i == 3
90            camData{i}(:, [1 2]) = camData{i}(:, [2 1]);
91        end
92
93        dataLen(i) = length(camData{i});
94
95        % Plotting all extracted object locations
96        figure
97        plot(camData{i}(:,1),camData{i}(:,2), 'r.', 'MarkerSize', 20)
98        axis equal
99        drawnow
100       pause(3)
101   end
102
103   % Looping through camera to fill in any zero values and determine start
104   for i = 1:numCams
105
106       % Setting data to look at for this loop iteration
107       currentData = camData{i}(:,:);
108
109       % Making any zero values NaN
110       currentData(currentData==0) = NaN;
111
112       % Finding number of existing data points
113       nonNanValues = mean(sum(~isnan(currentData)));
114
115       % Creating sample vector with length equal to data
116       samples = linspace(1,nonNanValues,length(currentData));
117
118       % Interpolating missing points at each NaN
119       filledData = fillmissing(currentData,'linear','SamplePoints',samples);
120
121       % Finding minimum in first handful of frames
```

```
122         % And designating starting index for synchronization
123         [val, idx] = min(filledData(10:40,2));
124         startIdx(i) = idx;
125 
126         % Adjusting for new data length and saving filled data
127         dataLen(i) = dataLen(i)-idx;
128         camData{i} = filledData;
129 
130 
131     end
132 
133     % Finding minimum data length to cut all data to
134     adjLen = min(dataLen);
135 
136     % Initializing snapshot matrix
137     newData = zeros(numCams*2,adjLen-1);
138 
139     % Looping through data and adjusting length
140     for i = 1:numCams
141 
142         % Selecting current data for this loop iteration
143         currentData = camData{i}(:,:);
144 
145         % Filling in snapshot matrix with synchronized starts and lengths
146         newData((2*i-1),:) = currentData(startIdx(i):startIdx(i)+adjLen-2,1);
147         newData((2*i),:) = currentData(startIdx(i):startIdx(i)+adjLen-2,2);
148 
149     end
150 
151     % Setting mean of each row in snapshot matrix equal to zero
152     newData = newData-mean(newData,2);
153 
154     % Calculating SVD
155     [U,S,V] = svd(newData,'econ');
156 
157     % Finding sigma values from S matrix
158     sigmas = diag(S);
159 
160     % Plotting sigma values and energy
161     figure
162     subplot(2,1,1)
163     plot(sigmas,'bo','Linewidth',2)
164     axis([0 length(sigmas) 0 max(sigmas)])
165     title('Sigma Values For Each Mode')
166     ylabel('\sigma Values')
167     xlabel('Mode')
168 
169     subplot(2,1,2)
170     plot(sigmas.^2/sum(sigmas.^2),'bo','Linewidth',2)
171     axis([0 length(sigmas) 0 1])
172     title('Energy Accounted For in Each Mode')
173     ylabel('Energy Percent Contribution')
174     xlabel('Mode')
175     sgtitle(['Test ', num2str(dataSet), ' SVD Analysis'])
176     %%
177     % Calculating rank approximation for given rank number
178     rankNum = 2;
179     rankApprox = U(:,1:rankNum)*S(1:rankNum,1:rankNum)*V(:,1:rankNum)';
180 
181     % Plotting data vs rank approximation for each camera in y directions
182     figure
```

```
for i = 1:size(newData,1)/2
    subplot(3,1,i)
    plot(newData(2*i,:), 'k', 'LineWidth', 3)
    hold on
    plot(rankApprox(2*i,:),'r.')
    hold off
    axis([0 length(newData) -100 100 ])
    title(['Camera ', num2str(i)])
    ylabel('Pixel Displacement (Y)', 'FontSize', 9)
    xlabel('Frame', 'FontSize', 9)
end
sgtitle(['Rank ', num2str(rankNum),...
    ' Approximations for Test ', num2str(dataSet)])
legend('Recorded Data', 'Approximated Data')

% Plotting data vs rank approximation for each camera in x directions
figure
for i = 1:size(newData,1)/2
    subplot(3,1,i)
    plot(newData(2*i-1,:), 'k', 'LineWidth', 3)
    hold on
    plot(rankApprox(2*i-1,:),'r.')
    hold off
    axis([0 length(newData) -100 100 ])
    title(['Camera ', num2str(i)])
    ylabel('Pixel Displacement (X)', 'FontSize', 9)
    xlabel('Frame', 'FontSize', 9)
end
sgtitle(['Rank ', num2str(rankNum),...
    ' Approximations for Test ', num2str(dataSet)])
legend('Recorded Data', 'Approximated Data')
```