Principal Component Analysis on a Spring Mass System

Johnny Van

February 21, 2021

Abstract

Principal Component Analysis is a tool that allows us to condense large datasets into an approximation matrix with significantly fewer dimensions. This project covers the use of principal component analysis on the typical spring mass system for a variety of scenarios in which we attempt to find the principal components that describe the direction of our data with the most variance and project our data onto these principal components.

1 Introduction and Overview

Let's consider a typical spring mass system and we want to understand it's motion without physics training. So we set up three cameras at different angles to record videos of the spring mass system and collection two dimensions of data from each camera for a total of six dimensions of data. This may seem like overkill, but the reason is because we don't know how many dimensions our data actually has. When presented with an abundance of data, the singular value decomposition(SVD) will weed out redundancies such that we can create an approximation of the original data which contains significantly less dimensions. This project covers four different cases of the spring mass system in which we use the SVD and principal component analysis to find approximations of each case.

2 Theoretical Background

2.1 Singular Value Decomposition

Suppose we have a matrix A, due to the properties of matrix operations, we are able to decompose a matrix A into components, where U and V^* are unitary matrices and Σ is a diagonal matrix. By the textbook,[1], we are introduced to the singular value decomposition(SVD) in equation (1).

$$A = U\Sigma V^* \tag{1}$$

Where $U \in \mathbb{R}^{mxm}$,$V \in \mathbb{R}^{nxn}$ and $V \in \mathbb{R}^{mxn}$. The values on the diagonal of Σ are called the **singular values** of A, the column vectors of U are called the **left singular values** of A, and similarly the column vectors of V are the **right singular values** of A.

The reason why the SVD is so useful is due to how we can create low-dimensional approximations of data. If A is a matrix with rank r, then A is the sum of r rank 1 matrices:

$$A = \sum_{j=1}^{r} \sigma_j u_j v_j^* \tag{2}$$

The product of $u_j v_j^*$ is an "outer product", and it has the same size as A and has a rank of 1. It turns out that the sum of rank 1 matrices gives an excellent approximation of the matrix A. If rank r is rather large, we don't have to evaluate the summation all the way up to r, we could simply sum the first N terms and that will give us a decent approximation. This method is known as the best rank N approximation of A. If we have N = 1, then we just sum the first term, and this sum will be the rank-1 approximation of A.

This has many applications. For example, if we have an image, storing the entire image in a matrix takes a lot of storage, however if we take a low-rank approximation using SVD, we can represent our image without having to store every single pixel value.

Basically, SVD allows us to retain as much information possible while keeping the fewest number of dimensions to our data. It essentially captures the most important trends in our data and uses that as a approximation of the data.

2.2 Background of Principal Component Analysis

Imagine that we have 4 row vectors each containing data about students' grades. Let's put these row vectors into a matrix, like so.

$$X = \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

We first subtract each row of X by its respective mean, and lets still define this as the matrix X. We compute all the variances and covariances between the rows of X using the equation.

$$C_X = \frac{1}{n-1} X X^T = \begin{bmatrix} \sigma_a^2 & \sigma_{ab}^2 & \sigma_{ac}^2 & \sigma_{ad}^2 \\ \sigma_{ba}^2 & \sigma_b^2 & \sigma_{bc}^2 & \sigma_{bd}^2 \\ \sigma_{ca}^2 & \sigma_{cb}^2 & \sigma_c^2 & \sigma_{cd}^2 \\ \sigma_{da}^2 & \sigma_{db}^2 & \sigma_{dc}^2 & \sigma_d^2 \end{bmatrix}. \tag{3}$$

The matrix, C_X, is called the covariance matrix which as you can see is square and symmetric. The purpose of principal component analysis is to find new set of coordinates (change of basis) such that all the variables are uncorrelated. This will mean that each variable will contain new information which none of the information is redundant. Next, we want to know which variable has the largest variance because it will contain the majority of the information about our data. To do this, we diagonalize our covariance matrix so that all of the off-diagonal elements in our matrix is zero.

$$C_X = V \Lambda V^{-1} \tag{4}$$

The basis of the eigenvectors contained in matrix V are the **principal components** which are orthogonal, indicating that they are uncorrelated. This is because C_X is a symmetric matrix, which causes its eigenvalues to be real and thus their eigenvectors will be orthogonal. Then the diagonal of the matrix, Λ, basically the eigenvalues of C_X, represents the magnitude of the variances of these newly formed variables.

2.3 Connection to Singular Value Decomposition

The prinicpal component analysis has strong connections to the singular value decompsition. The SVD of a matrix A is connected to the eigenvalue decomposition of AA^T. Consider,

$$A = \frac{1}{\sqrt{n-1}} X \tag{5}$$

Then it is known that,

$$C_X = \frac{1}{n-1} X X^T = A A^T \tag{6}$$

And from our previous work in the course,

$$C_X = A A^T = U \Sigma^2 U^T \tag{7}$$

So you can see that Σ^2 contain the eigenvalues of the covariance matrix, and that the basis of eigenvectorrs in matrix U are the principal components. To project the data into a new basis of the principal components, you would use the equation:

$$Y = U^T X \tag{8}$$

The main message of this section is that if we have a matrix X, containing a few row vectors of data, we first have to **standardize** X by subtracting each row of X by its mean. Then we define A with equation (5), and perform the SVD on matrix A to be able to find the matrices U, Σ, and V. From these matrices we can then find the eigenvectors, principal components, of the covariance matrix, C_X since they are the columns of U, and we can find the singular values of each principal components by looking at the diagonal of σ^2. And the matrix V contains the time-series information of our approximation. Then to plot our approximation, we project our data onto a certain number of princicpal components, we use equation (8).

3 Algorithm Implementation and Development

For a single test case, we are given video footage of 3 different cameras recording the same spring-mass system. Using Matlab, we convert these video files into 4-D matrices, width by height by RGB values by frame. It would be complicated to analyze a video, so it is simpler to analyze the video frame by frame.

3.1 Algorithm 1: Tracking the position of the mass

In this project, we are tracking the mass in this system by taking advantage of the fact that there is a very bright light on top of the mass. Therefore, if we convert each frame of the video from RGB to grayscale, then the pixel with the largest value refers to the brightest spot in the grayscale image. We use this technique to track the x and y position of the mass.

1. First convert the frame into grayscale image.
2. Examine the frame and filter out all parts of the image that does not contain the movement of the mass.
   (a) In this project, I created a small box that contained the flashlight on the mass and the position of this small box updates based on the coordinates of the flashlight in the previous frame, essentially "following" the position of the flashlight as mass moves throughout the video.
   (b) The goal is to make sure to track only one point of the mass at all times. If tracked coordinates keeps jumping from the bright flashlight to the white of the bucket, your data will not be accurate since it would not reflect the actual motion of the mass.
3. Use the max() function to find the brightest spot in the filtered frame. Place the two coordinates into their respective vectors.
4. Plot the two coordinates onto the frame itself so you will be able to see in real time what coordinates your vectors are storing and whether or not they are accurate.
   (a) If you see that you are not tracking the coordinates of the mass properly, then you need to adjust what you did in step 2. For example, changing the size of the small frame tracking the flashlight would usually fix this issue.
5. Repeat steps 1-4 for all frames of the video.
6. Repeat steps 1-5 for the other two videos of the test case.

3.2 Algorithm 2: Aligning the Videos

Aligning the videos is very important to the accuracy of your PCA approximation. If they are not aligned, then your principal component analysis will believe there are additional sources of variance in the data which causes you to need more principal components in your approximation than you actually need. The curves of the approximations will also not be very smooth unless you align the videos properly.

1. Play each video frame by frame and find the earliest frame for each video such that the starting position of the mass is roughly the same.
   (a) Usually, the simple way to do this is to align the videos such that the mass is at its peak or at the bottom of its oscillation.
   (b) Also, if you plot the raw data, such as the figures in Appendix A, aligning the peaks of the sinusodial curves ensures that the videos are aligned properly.
2. For each video, cut out all frames that comes before this certain frame found in step 2, and cut out the ends of the video such that all 3 videos have the same number of frames.

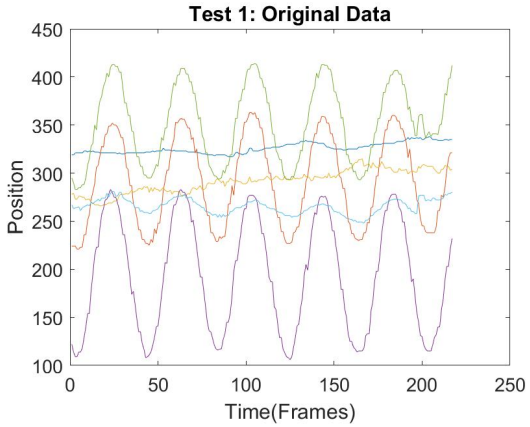

Figure 1: A plot containing the coordinates of the tracked mass from each of the 6 cameras in Test Case 1.

3.3 Algorithm 3: Principal Component Analysis

This algorithm refers heavily to Connection to Singular Value Decomposition section in the theoretical background. We have our matrix X formed from our 6 row vectors from the 3 cameras one of our cases. We first have to standardize the matrix X by subtracting each row of X by its mean and then define A by equation (5). The full PCA process is below:

1. Taking all of the vectors we found in Algorithm 1, stack all of the row vectors into a matrix called X.
2. We must **standardize** X, by finding the mean of each row of X, and subtract each row of X by its mean.
3. Find the matrix A using equation (5). Use the svd() function on matrix A to find the matrices of U, S, and V.
4. Square the matrix S, and plot diagonal of S^2 divided by the sum of S^2 to get percentage of energy contained by each value in the diagonal. Observe the percentage of energy each principal components contains and find how many components you need to contain at least 95 percent of the variance of the data
5. Plot the PCA approximation by projecting our data onto the necessary principal component using equation (8). You do this by multiplying the standardized X matrix by only a certain number of columns of U, the number determined by how many principal components you need for 95% of the variance.

4 Computational Results

In my figures below, the term "PCA mode 1" refers the projection of the original data onto the first principal component and so on. We can compare our projections to the original data, most of which is in Appendix A, of each case and see how well we approximated the general trend of the data with our approximation that uses fewer dimensions of data than the raw data.

4.1 Ideal Case - Test Case 1

My results for the ideal case in Figure 2 were almost perfect. You can see we need two principal components to describe at least 95% of the variance of the data and the first principal component contains a significant amount of energy which makes sense because the system is only moving in the up and down motion. So "PCA Mode 1" should represent the vertical motion of the mass and the "PCA Mode 2" should represent the very minimal side to side motion. We can compare our projection to Figure 1, and see that our projection describes the general trend of the mass motion very well with only two curves.

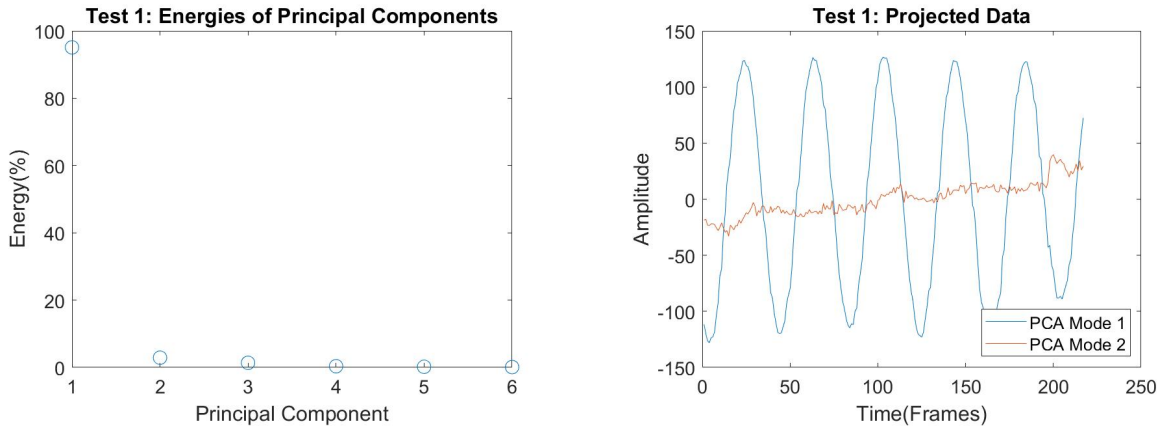

Figure 2: The plots of the energies of each principal component and the projection of original data onto the essential principal components for Test Case 1.

4.2 Noisy Case - Test Case 2

In this case, the camera is shaking in each video, we can expect three sources of variance in the data. The up and down movement of the mass, the minimal side to side motion, and the unpredictable, yet significant movements of the camera due to noise. So in Figure 3, we can see in the energy plot, that we require 3 principal components and that the "Projected Data" plot seems very noisy. Thus, it is very difficult to interpret what each PCA Mode represents in terms of the movements of the mass.

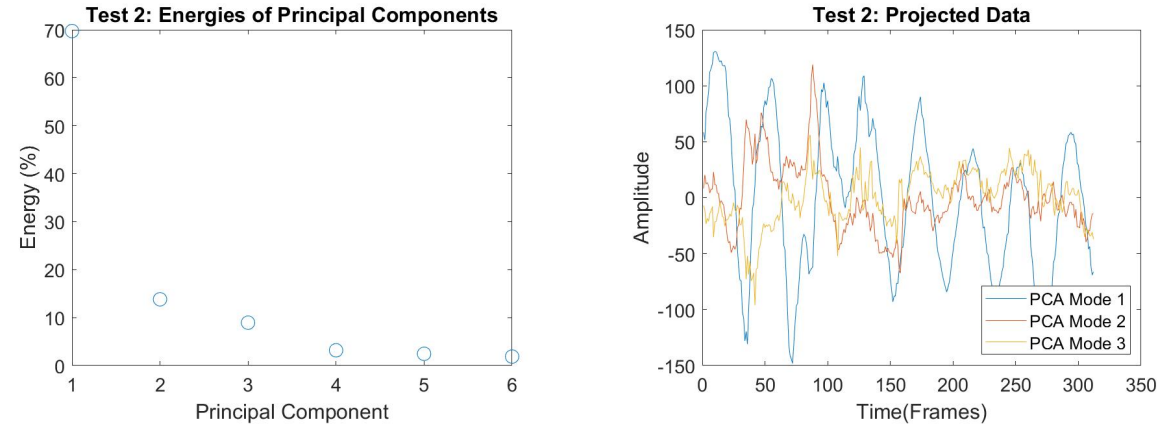

Figure 3: The plots of the energies of each principal component and the projection of original data onto the essential principal components for Test Case 2.

4.3 Ideal Case with Horizontal Displacement: Test Case 3

In this case, the camera is not shaking, but there is more horizontal displacement of the mass compared to case 1. So, we can expect two sources of variance in the data. The up and down movement of the mass, and the horizontal displacement of the mass. So in figure 4, we can see in the energies plot, that we require 2 principal components for 95% of the variance, and if we compare the projection to Figure 2, this result makes sense. We get the same vertical displacement, but instead of having "PCA Mode 2" be a sinusodial curve with low amplitude in Figure 1, "PCA Mode 2" has a much higher magnitude in Figure 4, reflecting the increased horizontal motion.

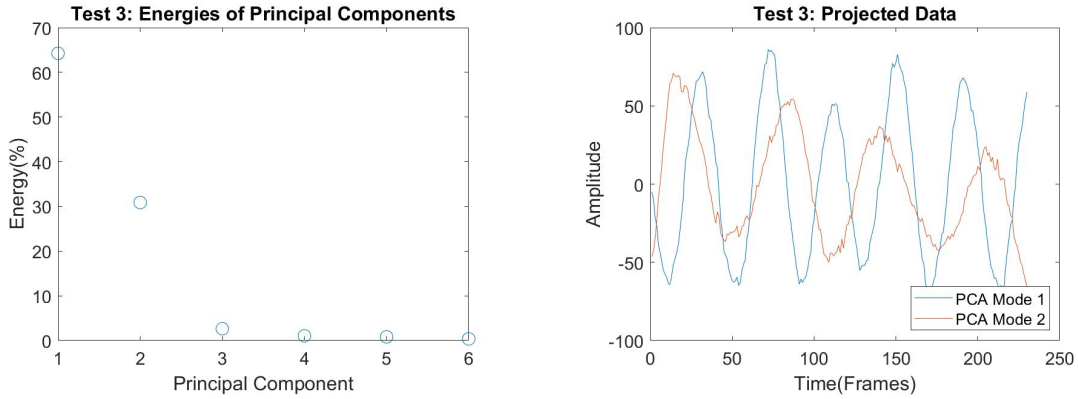

Figure 4: The plots of the energies of each principal component and the projection of original data onto the essential principal components for Test Case 3.

4.4 Horizontal Displacement and Rotation: Test Case 4

There is now vertical, horizontal and rotational movement of the mass in the system which corresponds to three main sources of variance. If we look at Figure 5, we can see that we need 3 principal components for our approximation. Our projected data isn't as clean as we had hoped, but we could make some conclusions. In the video, the mass has much more vertical motion compared to horizontal or rotational motion, so it is safe to predict that PCA Mode 1 represents the vertical motion of the mass. In the video, it is difficult to know whether the horizontal or rotational motion was slightly greater than the other, so we cannot make any conclusions about the other two PCA modes.

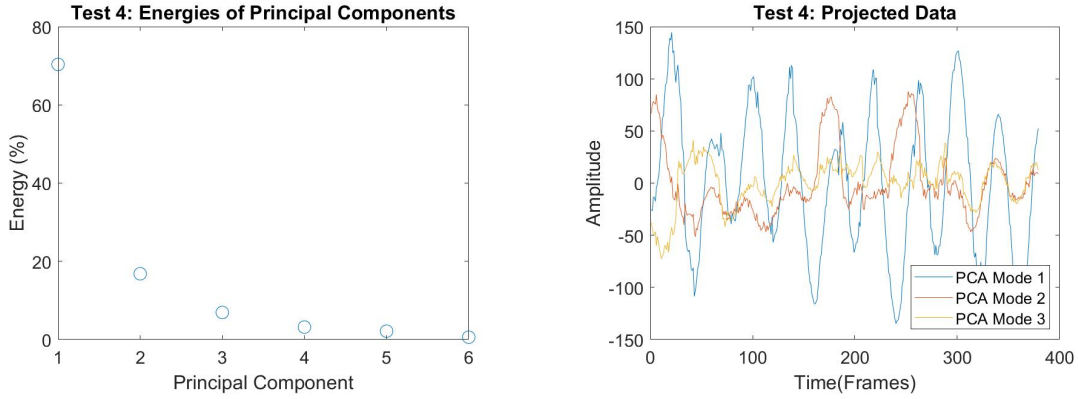

Figure 5: The plots of the energies of each principal component and the projection of original data onto the essential principal components for Test Case 4.

5 Summary and Conclusions

With Principal Component Analysis, we were able to analyze four different scenarios of the spring mass system and in each scenario, we are somewhat able to determine what the PCA modes represent in terms of the spring-mass system. This type of analysis is useful for almost every data science field because we are able to condense large data sets into a much smaller data set that approximates the original data with a very high accuracy. This allows us to find the general trends in data quite easily without much effort.

References

[1] Jose Nathan Kutz. *Data-driven modeling & scientific computation: methods for complex systems & big data*. Oxford University Press, 2013.

Appendix A Additional Images

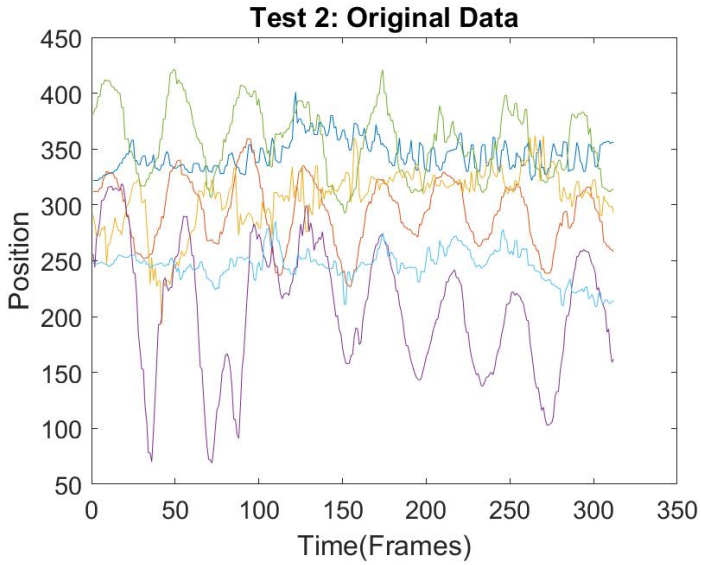

Figure 6: Plot of the coordinates tracked by each of the 3 cameras in Test Case 2

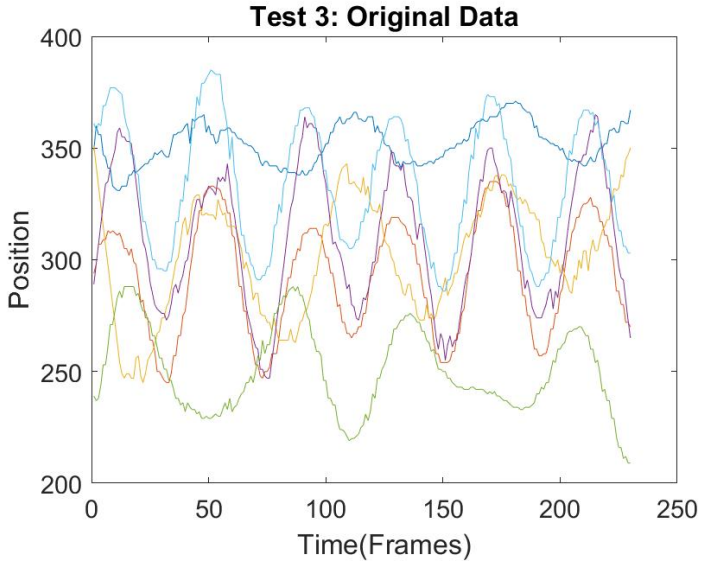

Figure 7: Plot of the coordinates tracked by each of the 3 cameras in Test Case 3

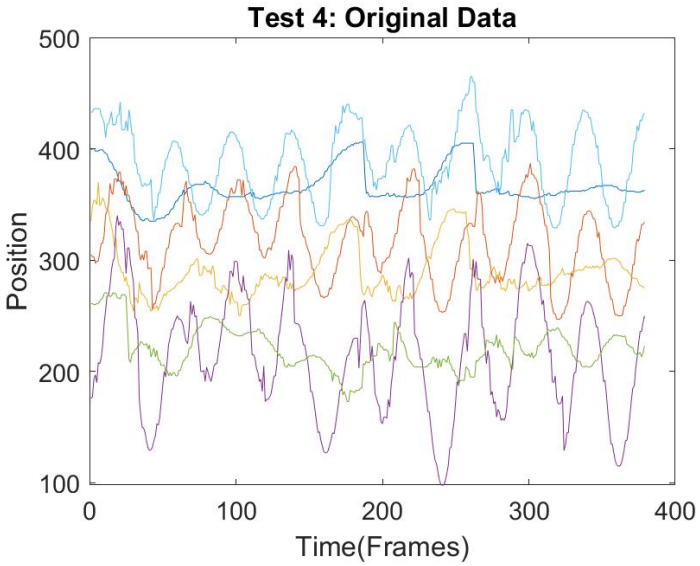

Figure 8: Plot of the coordinates tracked by each of the 3 cameras in Test Case 4

Appendix B MATLAB Functions

- `load('X.mat')` returns a matrix representing video file X, where the matrix is 4-D, width by height by RGB values by frame.
- `rgb2gray(X)` converts the RGB image, X, into a grayscale image where each pixel has a value indicating how bright that pixel is.
- `[`I_1, I_2`] = ind2sub(sz, linearIndex)` returns I_1, I_2 which represents the equivalent multidimensional subscripts corresponding to the linearIndex for a multidimensional array of size sz.
- `[Max, Index] = max(x(:))` returns the max of an array of values and gives its index.
- `imshow(X)` returns the frame of the video, X, as a figure.
- `[m,n] = size(X)` stores the number of rows of X as m and the number of columns of X as n.
- `repmat(mean, 1, n)` returns an array containing n copies of mean in the row dimension specified by the 1.
- `[U,S,V] = svd(A, 'econ')` performs the singular value decomposition of matrix A and stores the decompsition as the three matrices U,S,V such that A = U*S*V'. The 'econ' means that the command returns an economy-size decomposition of A.

Appendix C MATLAB Code

This code represents what I did for Test Case 1, for the other three test cases, you simply have to change the parameters in order to align the three videos properly and track the coordinates of the mass properly as well. Basically, the code for Principal Component Analysis of each of the four scenarios is very similar. However, it takes a lot of precise adjusting to get accurate results. My entire code can be found in my Github.

```
%% Tracking Mass in Camera 1 of Case 1: Ideal Case

load('cam1_1.mat'); %% Load in video in width by height by rgb by frame matrix

x_pos = 319; %% Initial x location of flashlight/area I am tracking
y_pos = 225; %% Initial y location of flashlight/area I am tracking

vidFrames1_1 = vidFrames1_1(:,:,:,7:223); %% Trims the video to contain only
                                          % frames 7 to 223 of the video.
numFrames = size(vidFrames1_1,4); %% Number of frames in video.
for j = 1:numFrames
    X = rgb2gray(vidFrames1_1(:,:,:,j)); %% Converts the j-th frame to grayscale

    X(:, 1:x_pos-20) = 0; %% Makes the frame zero outside 20 pixels region
    X(:, x_pos+20:640) = 0; %% from the location of the tracked area.

    X(1:y_pos-20, :) = 0;
    X(y_pos+20:480, :) = 0;

    [Max, Index] = max(X(:)); %% Finds the pixel with highest value

    [y_pos, x_pos] = ind2sub(size(X), Index); %% Finds the indexes of the pixel
                                              %with highest value

    x_1(j) = x_pos; %% Stores the x-coordinate of mass in j-th frame in vector
    y_1(j) = y_pos; %% Stores the x-coordinate of mass in j-th frame in vector

     imshow(X);  %% Plots the filtered grayscale j-th frame of the video.
     hold on
     plot(x_1(j), y_1(j), 'o', 'MarkerSize', 10); %% Plots the coordinates being stored
                                                  % on the j-th frame.
     drawnow

end

%% Tracking Mass in Camera 2 of Case 1: Ideal Case

load('cam2_1.mat');

vidFrames2_1 = vidFrames2_1(:,:,:,17:233);

numFrames2 = size(vidFrames2_1,4);

x_pos = 278;
y_pos = 122;
```

```
for j = 1:numFrames2
    X2 = rgb2gray(vidFrames2_1(:,:,:,j));

     X2(:, 1:x_pos-20) = 0;
     X2(:, x_pos+20:640) = 0;

     X2(1:y_pos-30,:) = 0;
     X2(y_pos+30:480, :) = 0;

    [Max, Index] = max(X2(:));

    [y_pos, x_pos] = ind2sub(size(X2), Index);
    x_2(j) = x_pos;
    y_2(j) = y_pos;

     imshow(X2);
     hold on
     plot(x_2(j), y_2(j), 'o', 'MarkerSize', 10);
     drawnow

end

%% Tracking Mass in Camera 3 of Case 1: Ideal Case

load('cam3_1.mat');

vidFrames3_1 = vidFrames3_1(:,:,:,6:222);

numFrames2 = size(vidFrames3_1,4);

x_pos = 295;
y_pos = 267;

for j = 1:numFrames2
    X3 = rgb2gray(vidFrames3_1(:,:,:,j));

    X3(1:y_pos-20, :) = 0;
    X3(y_pos+20:480, :) = 0;

    X3(:, 1:x_pos-30) = 0;
    X3(:, x_pos+30:640) = 0;

    [Max, Index] = max(X3(:));

    [y_pos, x_pos] = ind2sub(size(X3), Index);
    x_3(j) = x_pos;
    y_3(j) = y_pos;

     imshow(X3);
     hold on
     plot(x_3(j), y_3(j), 'o', 'MarkerSize', 10);
     drawnow

end
```

```
%% Performing Principal Component Analysis

X = [x_1; y_1; x_2; y_2; x_3; y_3]; %% Stacks all row vectors into single matrix

[m,n] = size(X); %% Stores the number of rows as m and columns as n.

mn = mean(X,2); %% Creates a 6x1 matrix containing the mean of each row of X
X = X - repmat(mn, 1, n); %% Subtracts the mean of each row from X.
A = X/sqrt(n-1); %% Rescales X so we can use SVD on it.

[U,S,V] = svd(A, 'econ'); %% Performs SVD on A.

figure(2);
plot(1:numFrames, X); %% Plots raw data, the coordinates of the mass from each camera.
set(gca,'Fontsize',14)
title("Test 1: Original Data")
xlabel("Time(Frames)")
ylabel("Position");


figure(3);
plot(V(:, 1:2)); %% Plots the time-series data
title("Test 1: Time-Series Data")
set(gca,'Fontsize',14)
xlabel("Time(Frames)")
ylabel("Position");

figure(4);
plot((diag(S).^2/sum(diag(S).^2)*100), 'o', 'Markersize', 10); %% Plots percentage
set(gca,'Fontsize',14)                                          %values of each sigma
title("Test 1: Energies of Singular Values")
xlabel("Singular Value Index")
ylabel("Energy of Singular Value(%)");

U_star = U(:,1:2); %% A 6x2 matrix, each column is a principal component.
projection = U_star'*(X); %% Projection of data using 2 principal components

figure(5);
plot(1:n,projection); %% Plots the projection data using 2 principal components.
set(gca,'Fontsize',14)
title("Test 1: Projected Data")
xlabel("Time(Frames)")
ylabel("Position");
```