PCA and a Spring-Mass System

Haoran Sun

February 23, 2021

Abstract

In this assignment, a spring-mass system experiment will be analyzed. We record the motion of the mass with multiple cameras and would like to know the behavior of mass. However, the data we collected is redundant, and we introduce the method of Principal Component Analysis(PCA) to weed out the redundancy of the data. We will also compare how this method varies under different cases.

1 Introduction and Overview

In this task, we are given 4 different cases of the motion of a spring-mass system, with each case being recorded by three cameras. Each camera will capture two dimensions of the mass's motion, so we have six dimensions of the data. However, the ideal motion of a spring-mass system should only be one dimension, as in the first case, where the mass moves only up and down. It means we collected too much data with redundancy. In the second case, the cameras are not stable, which add noise into the data. The mass are released off-center in the third case, which causes the mass to move in the x-y plane besides the original z-direction, vertical movement. In the last case, A rotation in the x-y plane are produced along with the horizontal motion and the vertical motion.

We will apply the method of Principal Component Analysis(PCA) to the data of each cases, with the goal of reducing the redundancy inside the data and recognizing which directions the motion of the mass is in for the most time.

2 Theoretical Background

2.1 Singular Value Decomposition(SVD)

The **singular value decomposition** of matrix $\mathbf{A} \in \mathbb{C}^{m\times n}$ is written as:

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

Where $\mathbf{U} \in \mathbb{R}^{m\times m}$ and $\mathbf{V} \in \mathbb{R}^{n\times n}$ are both unitary matrices and their column vectors form an orthonormal basis respectively. The vectors made up the columns of $\mathbf{U}$ are called **left singular vectors**, and those made up the columns of $\mathbf{V}$ are called **right singular vectors**. The values of σ_n on the diagonal of Σ are called the **singular values** of the matrix A.
Every matrix A has such an SVD, with a unique set of non-negative singular values. The singular values are always ordered with a decreasing magnitude.

2.2 Covariance Matrix

If we have two vectors $\mathbf{a}$ and $\mathbf{b}$, both containing some data. The covariance of these two vectors is defined by the formula

$$\sigma_{ab}^2 = \frac{1}{n-1}\mathbf{a}\mathbf{b}^T \tag{2}$$

which measures how the data from one vector vary with respect to the other. Large value of covariance means the two sets of data are highly correlated. If the covariance is zero, then the two variables are **uncorrelated**. It is divided by n-1 to make an unbiased estimator.
If we have multiple row vectors of data stored in a matrix $\mathbf{X}$, we can calculate the **covariance matrix** C_X of $\mathbf{X}$:

$$\mathbf{C}_X = \frac{1}{n-1}\mathbf{X}\mathbf{X}^T \tag{3}$$

C_X is a square, m $\times$ m matrix. What does C_X tell us? First of all, it contains the correlations between all possible pairs of measurements, where the diagonal represents the variance of particular measurements and off-diagonal elements represent the covariance between different measurements. Therefore, small off-diagonal elements corresponds to little redundancy between the data, meaning the measurement quantities are close to be independent. And large diagonal term indicates large fluctuation in that variable.

2.3 Principal Component

Define

$$\mathbf{A} = \frac{1}{\sqrt{n-1}}\mathbf{X} \tag{4}$$

Then the covariance matrix C_X is

$$\mathbf{C}_X = \frac{1}{n-1}\mathbf{X}\mathbf{X}^T = \mathbf{A}\mathbf{A}^T = \mathbf{U}\Sigma^2\mathbf{U}^T \tag{5}$$

where $\mathbf{A}$ is written in its SVD decomposition (Equation 1). To applying a change of basis to let the data in the matrix be in the basis of its **principal components**, meaning the data is written(diagonalized) in the basis, where all redundancies are removed, we simply multiply the matrix by $\mathbf{U}^T$. The data written in its principal components is

$$\mathbf{Y} = \mathbf{U}^T\mathbf{X} \tag{6}$$

The covariance of $\mathbf{Y}$, therefore, is

$$\mathbf{C}_Y = \frac{1}{n-1}\mathbf{Y}\mathbf{Y}^T = \frac{1}{n-1}\mathbf{U}^T\mathbf{X}\mathbf{X}^T\mathbf{U} = U^T\mathbf{A}\mathbf{A}^T\mathbf{U} = \mathbf{U}^T\mathbf{U}\Sigma^2\mathbf{U}^T\mathbf{U} = \Sigma^2 \tag{7}$$

Since all the off-diagonal elements of Σ is 0, variables in $\mathbf{Y}$ are uncorrelated.

3 Algorithm Implementation and Development

3.1 Can Tracking

For obtaining the position information of the can from the video, we need an algorithm to track the can. Since there is a very light bulb on the can, we can locate the bulb by turning the video into gray-scale and locating the brightest pixel in each frame. However this method might fail under some accidental circumstances, such as when the brightness of the pixel at other location is larger than that of the bulb, and in some part of the video, the bulb rotates away from the screen.

Therefore, instead of searching the brightest pixel of the whole screen, we set a threshold of brightness, which varies depending on the video(but is usually between 0.90 - 0.98). We then look at all the pixels limited in a certain region(where the can move) and find all the pixels whose brightness exceed the threshold in the frame. The white can itself also has high brightness, which is beyond our threshold. So the pixels on the can can still be counted even when the bulb is not shown in some video frames. Finally, we take the maximum value of the pixel coordinates(or minimum, depending on different videos as well) to be the coordinates of the can in that frame. See Algorithm 1 for the logic. We will be using this algorithm to track the can for all the cases, where we need to change the parameters(brightness threshold, range of coordinates, etc) to fit the situation.

Algorithm 1: Spotting can

Import data from the video into the vector `y`
Set up a parameter `num`, which equals to the number of video frames
for $j = 1 : \texttt{num}$ **do**
  turn the jth frame of the video into gray-scale
  Set up the range of x and y where the can moves and a threshold of brightness
  Looking for all the pixels exceeding the threshold in the range
  Take the pixel that has maximum or minimum value of coordinates as our coordinate
  Store every coordinate information of each frame into vectors
end for
Plot the position of can with respect to time

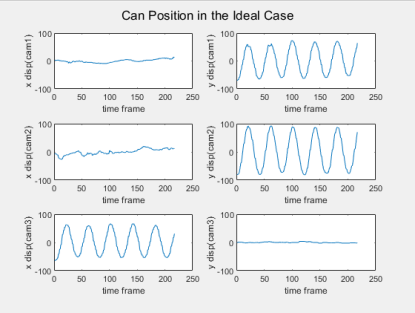
3.2 Principal Component Analysis

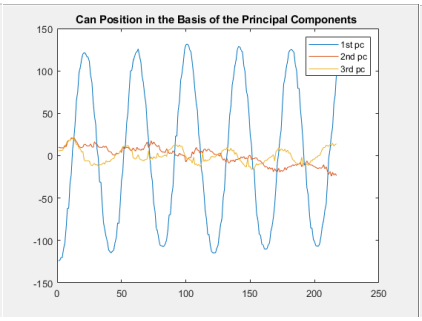
Once we obtained all the position information recorded by 3 cameras in each case, we will put them into a single Matrix and use PCA on the Matrix and see what we can find about the system. Since the videos have different length, we truncate the videos into same length and align them in time. After all the position vectors have the same length, we will do the following:

1. Put all position vectors into a matrix $\mathbf{X}$.
2. Subtract mean for each row of $\mathbf{X}$ to center the data at 0.
3. Apply SVD to the scaled matrix $\mathbf{X}$ ('scale' means multiply $\mathbf{X}$ by $1/\sqrt{n-1}$, where n is the number of frames of the truncated video).
4. Multiply the transpose of $\mathbf{U}$ obtained from SVD and $\mathbf{X}$ together, and the result matrix $\mathbf{Y}$ is the principal component projection of $\mathbf{X}$
5. The rows of $\mathbf{Y}$ are the principal components of $\mathbf{X}$. Thus we draw the rows. The plot shows the mass's bahavior in the basis of its principal components.
6. We also plot the square of each diagonal elements of matrix Σ obtained from SVD and divide it by the sum of square of all the diagonal elements, which indicates the energy of each principal components. The higher the energy of one principal components is compared to others, the more information of the system this principal component has.

4 Computational Results

4.1 Ideal Case





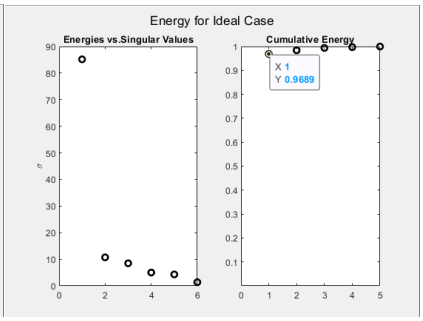


Figure 1: The x/y displacement in case 1 (Left)

Figure 2: Principal Components in case 1(Middle)

Figure 3: Energy of principal components in case 1(Right)

Using Algorithm 1, the plots of x and y displacement of the can recorded in three cameras are shown in Figure 1 (the mean of the plots are set to be 0). In each camera, the oscillation pattern is clearly observable in one direction, and there is little displacement in another direction, which meets our expectation for the ideal case. The first three principal components are shown in Figure 2. From the graph, we can tell the first principal component dominates the system, where second and third components are close to 0. The rest of the principal components are not plotted since they must be even less important than 2nd and 3rd principal components. From figure 3, we can tell the energy of only the first principal component is 0.9689, which means the first principal component almost captures all the energy of the system, telling us the data of the can is very close to be unidimensional, as we expected.

4.2 Noisy Case

The camera keeps shaking and adding a lot of random noisy movements of the mass. The position information of the mass is shown in Figure 4. Different from the first ideal case, for each camera, although the oscillatory pattern is still distinct, there are lots of random variations and spikes spotted, especially in the second camera. In Figure 5, we plotted the first 4 principal components. The first principal component still has the largest variance. However, unlike the first case(See Figure 2), the 2nd and 3rd principal components in this noisy case also have unnegligible, large variance. Looking at Figure 6, the energy of the first principal component is still larger than the rest of the principal components, but not significantly compared to case 1(see Figure 3). The 1st principal components only captures about 65% energy of the system. For obtaining over 90% energy, we need at least the first three principal components. It indicates the noise added into the system makes the data no longer one-dimension. We need at least three dimensions of data to approximate the system accurately.

4.3 Horizontal Displacement Case

With the horizontal displacement of the can added into the system besides the up and down motion, we can see how the tracking plot varied in Figure 7 from the first case. For each camera, we can see both x and y have a oscillatory motion. Figure 8 shows the first 5 principal components of the system in this case. As in previous case, the 1st component still has the largest variance. But the 2nd to 4th components also vary a lot. In previous case, the energy of first components exceeds that of the 2nd components with at least a factor of 2. In this case, the energy of the first component does not even the double of the second one, as shown in Figure 9. The energy of the first component captures only less than 60% of the total energy. We need energy of the first two components to obtain an energy close to 90%.

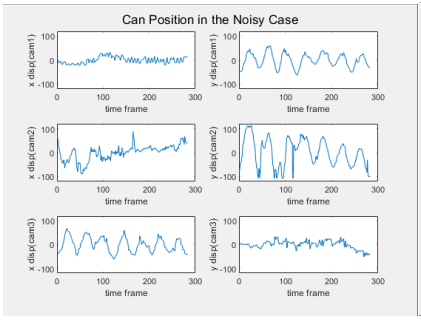


Figure 4: The x/y displacement in case 2 (Left)

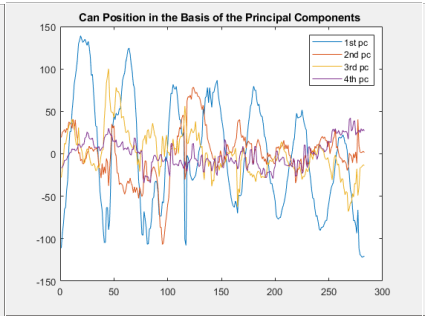


Figure 5: Principal Components in case 2(Middle)

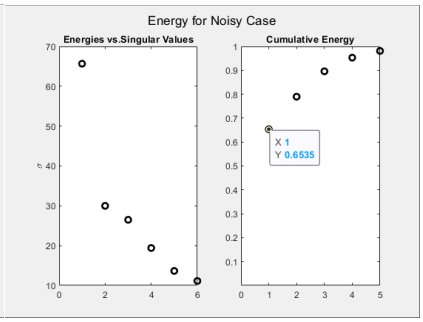


Figure 6: Energy of principal components in case 2(Right)

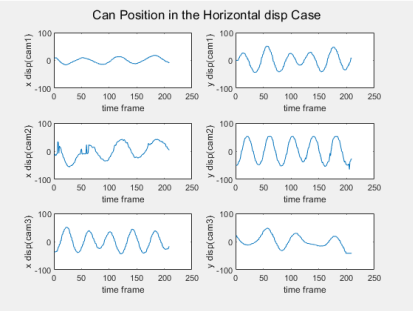


Figure 7: The x/y displacement in case 3 (Left)

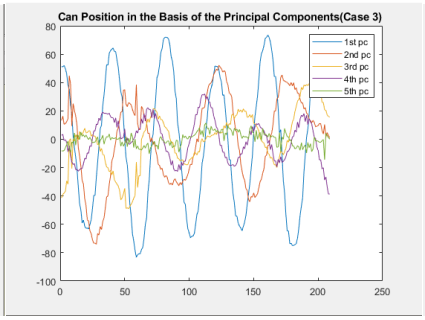


Figure 8: Principal Components in case 3(Middle)

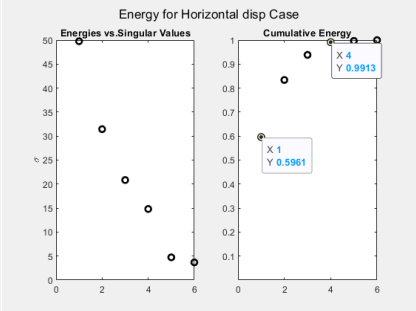


Figure 9: Energy of principal components in case 3(Right)

4.4 Horizontal Displacement with rotation

The mass in this case behaves in a strange way, as we can tell from its position tracking plot in Figure 10. As the vertical oscillation stays the same, there is a horizontal oscillation at the beginning, but it fades away and stays at around 0 afterwards. The rotation behavior is captured by the y-displacement in the third camera. The principal components figure of this case(Figure 11) indicates the first component has significantly larger variance than others. The 2nd and 3rd components also have certain noticable variance, and the rest of the components seem to be insignificant. The 1st component captures 84% of the total , as shown in Figure 12. The first three components make up about 97% of the total energy.

5 Summary and Conclusions

In this assignment, we saw how the PCA can be used to reduce redundancy in the data sets, and identify the components with maximal variance. We tracked the position information of a can in four cases. We found out the dimension of the data without redundancy in each case, and how the noise and different behaviors change the result of PCA. Most of the position information plots we have are smooth, and therefore satisfying. But some spikes and random patterns are still spotted, which may deviate the our result from accurate interpretation of the data. It can be improved by some potential algorithm that can track the position of the can more accurately.

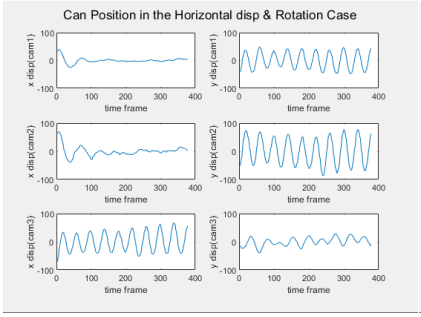


Figure 10: The x/y displacement in case 4 (Left)

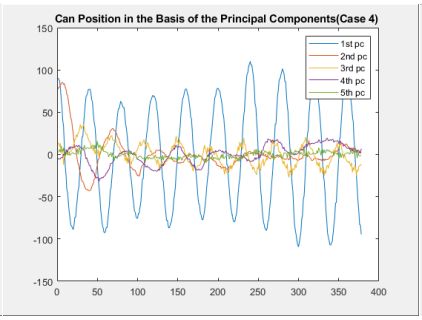


Figure 11: Principal Components in case 4(Middle)

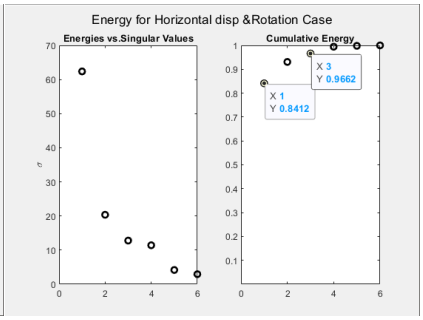


Figure 12: Energy of principal components in case 4(Right)

Appendix A MATLAB Functions

- `rgb2gray` Convert RGB image or colormap to grayscale.
- `M = max(X)` returns the largest element of vector `X`.
- `find` Find indices of nonzero elements.
- `[U,S,V] = svd(X)` produces a diagonal matrix S, of the same dimension as X and with nonnegative diagonal elements in decreasing order, and unitary matrices U and V so that X = U*S*V'.
- `diag(V)` puts V on the main diagonal

Appendix B MATLAB Code

[]

```
clear; close all; clc;



%% First camera
%Load Movie
load('cam1_1.mat');
[height,width, rgb, num_frames] = size(vidFrames1_1);
%Watch Movie
x1_coord = zeros(1,num_frames);
y1_coord = zeros(1,num_frames);
for j=1:num_frames
%Turn the fram into grayscale
X=rgb2gray(vidFrames1_1(:,:,:,j));
X=im2double(X);
brightest = 0.996;
%Returns all the positions where the frame is the brightest
[y,x] = find(X > brightest);
%Returns the positions of the birght spot;
%Limit y value to be larger than 200 and x value to be smaller than 360 ;

y_position = y>200;
y_max = min(y(y_position));

y1_coord(j) = y_max;
x_position = x<360;
x_max = max(x(x_position));
x1_coord(j) = x_max;
%imshow(X); drawnow
end
plot(1:num_frames, y1_coord);

%% Second camera

 load('cam2_1.mat');
 [height2,width2, rgb2, num_frames2] = size(vidFrames2_1);
%Watch Movie
x2_coord = zeros(1,num_frames2);
y2_coord = zeros(1,num_frames2);
for j=1:num_frames2
%Turn the fram into grayscale
X=rgb2gray(vidFrames2_1(:,:,:,j));
X=im2double(X);
brightest = 0.98;
%Returns all the positions where the frame is the brightest
[y,x] = find(X > brightest);
%Returns the positions of the birght spot;
%Limit y value to be between 100 and 380 and x value to be smaller than 330 ;
 y_low = y>100;
 y = y(y_low);
 y_high = y<380;
```

```
 y_max = min(y(y_high));
 y2_coord(j) = y_max;
 x_low = x<330;
 x = x(x_low);
 x2_coord(j) = max(x);

end
plot(1:num_frames2, y2_coord);



%% Third camera

load('cam3_1.mat');
[height2,width2, rgb2, num_frames3] = size(vidFrames3_1);
%Watch Movie
x3_coord = zeros(1,num_frames3);
y3_coord = zeros(1,num_frames3);
for j=1:num_frames3
%Turn the fram into grayscale
X=rgb2gray(vidFrames3_1(:,:,:,j));
X=im2double(X);
brightest = 0.9;
%Returns all the positions where the frame is the brightest
[y,x] = find(X > brightest);
%Returns the positions of the birght spot;
%Limit x value to be between 250 and 440 a

 x_low = x>250;
 x = x(x_low);
 x_high = x<440;
 x_max = min(x(x_high));
 x3_coord(j) = x_max;

 %Limit y value to be between 240 and 280
 y_low = y>240;
 y = y(y_low);
 y_high = y<280;
 y_max = min(y(y_high));
 y3_coord(j) = y_max;

end
plot(1:num_frames3, y3_coord);

%% Align all the data from different cameras in time

y1_min = min(y1_coord);
y1_min_position = find(y1_coord == y1_min);
y1_aligned = y1_coord(y1_min_position(1):end);
x1_aligned = x1_coord(y1_min_position(1):end);

num_newFrames = length(y1_aligned);
y2_min = min(y2_coord);
```

```
y2_min_position = find(y2_coord == y2_min);
y2_aligned = y2_coord(y2_min_position(1):y2_min_position(1)+length(y1_aligned)-1);
x2_aligned = x2_coord(y2_min_position(1):y2_min_position(1)+length(y1_aligned)-1);

x3_min = min(x3_coord);
x3_min_position = find(x3_coord == x3_min);
x3_aligned = x3_coord(x3_min_position(1):x3_min_position(1)+length(y1_aligned)-1);
y3_aligned = y3_coord(x3_min_position(1):x3_min_position(1)+length(y1_aligned)-1);

plot(1:num_newFrames,y1_aligned,1:num_newFrames,y2_aligned,1:num_newFrames,x3_aligned);

%% Reduce redundancy(PCA)
%Put all the data into a single matrix;
X = [x1_aligned;y1_aligned;x2_aligned;y2_aligned;x3_aligned;y3_aligned];
%Covariance of X

[m,n]=size(X); % compute data size
mn=mean(X,2); % compute mean for each row
X=X-repmat(mn,1,n); % subtract mean

cov_x = 1/(n-1) * X* X';


[u,s,v]=svd(X/sqrt(n-1),'econ'); % perform the SVD
lambda=diag(s).^2; % produce diagonal variances
Y=u'*X; % produce the principal components projection
cov_y = 1/(n-1) * Y* Y'; %Covariance matrix of Y;

%% Plot:Ideal case; original basis
figure(1)
sgtitle('Can Position in the Ideal Case');
subplot(3,2,1);
plot(1:num_newFrames,X(1,:));
ylim([-100,100]);
xlabel('time frame')
ylabel('x disp(cam1)');
hold on
subplot(3,2,2);
plot(1:num_newFrames,X(2,:));
ylim([-100,100]);
xlabel('time frame')
ylabel('y disp(cam1)');
subplot(3,2,3);
plot(1:num_newFrames,X(3,:));
ylim([-100,100]);
xlabel('time frame')
ylabel('x disp(cam2)');
subplot(3,2,4);
plot(1:num_newFrames,X(4,:));
ylim([-100,100]);
xlabel('time frame')
ylabel('y disp(cam2)');
subplot(3,2,5);
plot(1:num_newFrames,X(5,:));
```

```
ylim([-100,100]);
xlabel('time frame')
ylabel('x disp(cam3)');
subplot(3,2,6);
plot(1:num_newFrames,X(6,:));
ylim([-100,100]);
xlabel('time frame')
ylabel('y disp(cam3)');

%% Graph of Principle components
hold off

plot(1:num_newFrames,Y(1,:));
hold on
plot(1:num_newFrames,Y(2,:));
plot(1:num_newFrames,Y(3,:));
title('Can Position in the Basis of the Principal Components ');
legend('1st pc','2nd pc', '3rd pc');


%% Energy
sgtitle('Energy for Ideal Case');
subplot(1,2,1)
sig = diag(s);
energy1 = sig(1)^2/sum(sig.^2)
energy2 = sum(sig(1:2).^2)/sum(sig.^2)
energy3 = sum(sig(1:3).^2)/sum(sig.^2)

hold off
plot(sig,'ko','Linewidth',2)
title('Energies vs.Singular Values');
ylabel('\sigma')
subplot(1,2,2)
plot(cumsum(sig.^2)/sum(sig.^2),'ko','Linewidth',2);
axis([0 5 10^-(18) 1])
title('Cumulative Energy');
```

Code for Tracking and PCA in first case. The code is similar in all 4 cases with only some parameters varied