AMATH 482 Homework 3

Fiona Fu
February 2020

Abstract

Illustrating various aspects of the principal component analysis (PCA) for its practical usefulness and the effects of noise on the PCA algorithms by considering video of a spring-mass system.

1 Introduction and Overview

By exploring the PCA method on different datasets created from video of a spring-mass system recorded by three cameras, we need to first extract the mass positions to compare and contrast the different cases:

1.1 Ideal Case.

Considering the mass positions where the motion is in the z direction, and then use PCA to find the simple harmonic motion.

1.2 Noisy Case.

After repeating the ideal case experiment, the camera shakes a little bit and we need to use PCA algorithms to see how's the dynamics.

1.3 Horizontal Displacement.

The camera was shifted, and the motion was in the x – y plane as well as the z direction; try PCA to see the difference.

1.4 Horizontal Displacement and Rotation.

The camera was shifted and rotate, we also need to use PCA to see how the motion changed.

2 Theoretical Background

In this assignment, we are applying Principal Component Analysis (PCA) after we extract the mass positions from the video frames to compare and contrast the different cases.

For each of the four cases, there are three cameras for each clip that contains corresponding x – y coordinates.

Then, we want to find the covariance for the coordinate matrix, which needs to be square and symmetric, and we have the formula for an unbiased estimator with the division on (n-1):

$$\mathbf{C_X} = \frac{1}{n-1}\mathbf{XX}^T$$

Also, we can connect this to the Singular Value Decomposition (SVD) that account for the 1/(n – 1) factor, Then,

$$\mathbf{C_X} = \frac{1}{n-1}\mathbf{XX}^T = \mathbf{AA}^T = \mathbf{U}\Sigma^2\,\mathbf{U^T}$$

We used a change of basis to work in the basis of the principal components by multiplying by $\mathbf{U^{-1} = U^T}$ and the data in new coordinates is

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

3 Algorithm Implementation and Development

All cases had a similar code and method to extract the mass positions from the video frames and then use PCA to see the overall dynamics. Take the Test 1: ideal case as an example:

At first, we need to load the movie and get the number of frames for each video clip by:

```
load ('camN_1.mat');
numFrames = size(vidFramesN_1,4);
```

Then we use a for loop to check each frames in the video:

```
for j = 1:numFrames
   X = vidFramesN_1(:,:,:,j);
   imshow(X); drawnow
end
```

By adding the code within the for loop, we can clean out the unrelated stuff in the background by blackening the area of pixels. Take the clip Cam1_1 as an example, X here is 480*640*3, and the spring mass is oscillating around pixels at (:, 240:480) and(120:460, :), then our code will be:

```
  X(:,1:240) = 0;
  X(:,480:end) = 0;
```

We need to find the pixels at the max magnitude to trace the motion of mass, and then we can plug in the index to the video and plot the positions for both x and y.

```
  [Max Ind] = max(X(:));
  [yN1 xN1] = ind2sub(size(X), Ind);
  XN1 = [XN1 xN1];
  YN1 = [YN1 yN1];
  plot(XN1,'b','Linewidth',[1]);
  hold on
  plot(YN1,'r','Linewidth',[1]);
```

Continue this process for N = 1, 2, 3 and then use PCA to plot the dynamics for each case:
To begin with PCA, we need to make the number of frames to be the same when the lengths are different for each clip. Therefore, we need o find the minimum length and made all length of XN1 and YN1 to be the same by using the code:

```
Xmin = min([length(X11) length(X21) length(X31)]);
XN1 = XN1(1:Xmin);
YN1 = YN1(1:Xmin);
matrix = [X11;Y11;X21;Y21;X31;Y31];
```

Then we have our matrix for further calculation. Then we need to find the mean value and subtract mean from all entries in the matrixes we have and get a new matrix for the calculation of covariance:

```
[m,n] = size(matrix);
% Find the mean
miu = mean(matrix,2);
new_matrix = zeros(m,n);
```

```
% Subtract the mean from all entries in matrix we have
for i = 1:n
    for j = 1:m
      new_matrix(j,i) = matrix(j,i) - miu(j,1);
    end
end
```

Finally, we can find the covariance by the formula introduced in theoretical background and plot the final motion of dynamics.

```
Cx = (1/(n-1)) * new_matrix * new_matrix';
[V,D] = eig(Cx);
Y = V'*new_matrix;
plot(Y(1,:),'Linewidth',[1])
```

We will repeat the very same code with the same method and idea for other cases in Test 2, 3, 4 and finally compare the final graphs.

4 Computational Results

Test 1: Ideal Case

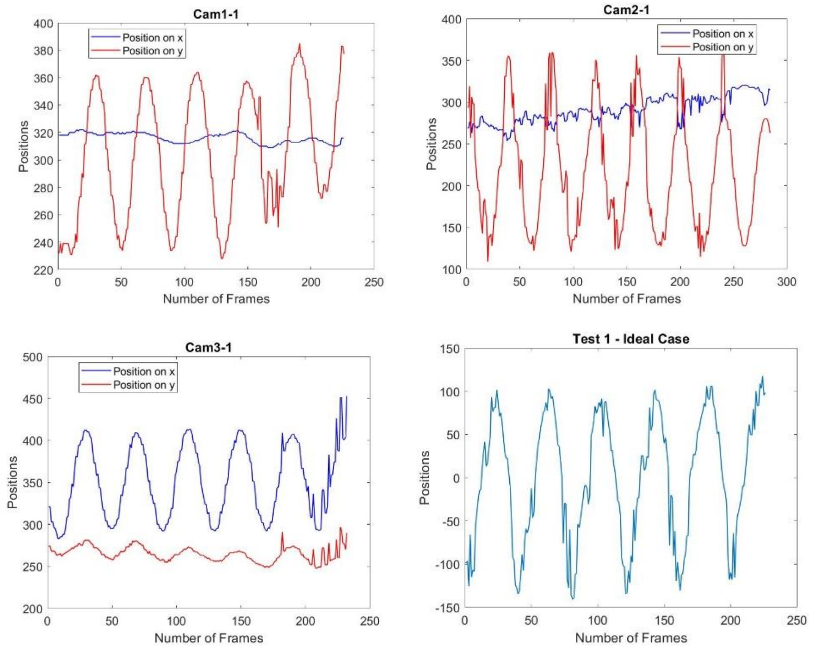

Figure 1: Ideal Case

Figure 1 contains four subplots where the first three are the mass positions in x-y coordinates for camera 1, 2, 3 separately. The first two for camera 1 and 2 has a very stable position on x, which means a little motion on x direction, when y oscillate stronger. In plot of camera 3, x oscillated more obvious when y became stable, because the camera was shifted for the x-y axis. The final subplot is the look of principle component.

Test 2: Noisy Case

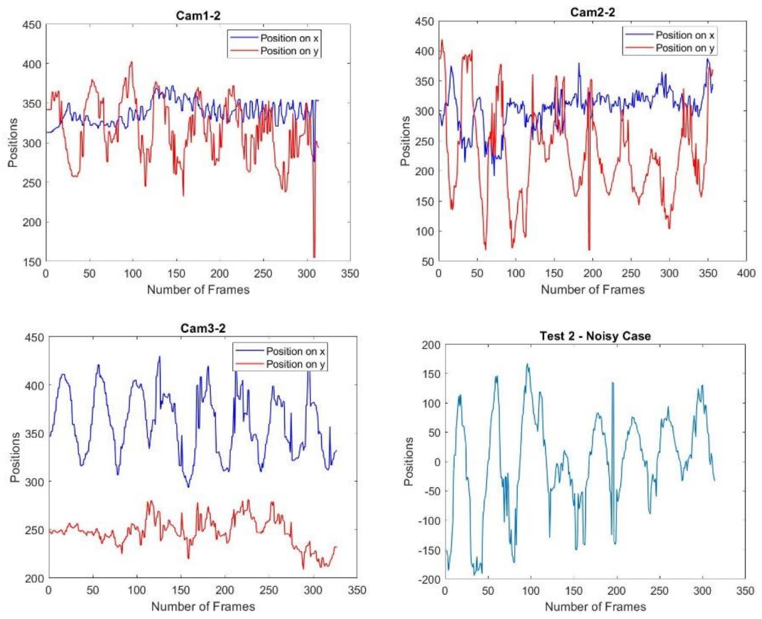

Figure 2: Noisy Case

Like Figure 1, Figure 2 also contains four subplots with the first three to be the x – y positions from three different camera and the last to be the movement of principle component. The overall motion is stronger than what is in Figure 1 and there are more oscillations with less regularity.

Test 3: Horizontal Displacement

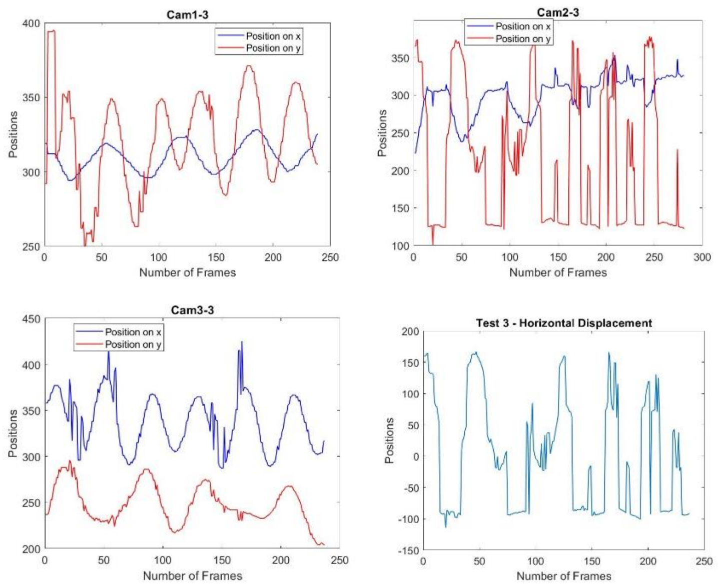

Figure 3: Horizontal Displacement

In Figure 3, there are even more unregular oscillations in both x and y direction. The motion in x direction is not as the first two figures (Figure 1, 2) that tend to be flatter in camera 1 and 2, which means there are more movements on the x-direction for the horizontal displacement.

Test 4: Horizontal Displacement and Rotation

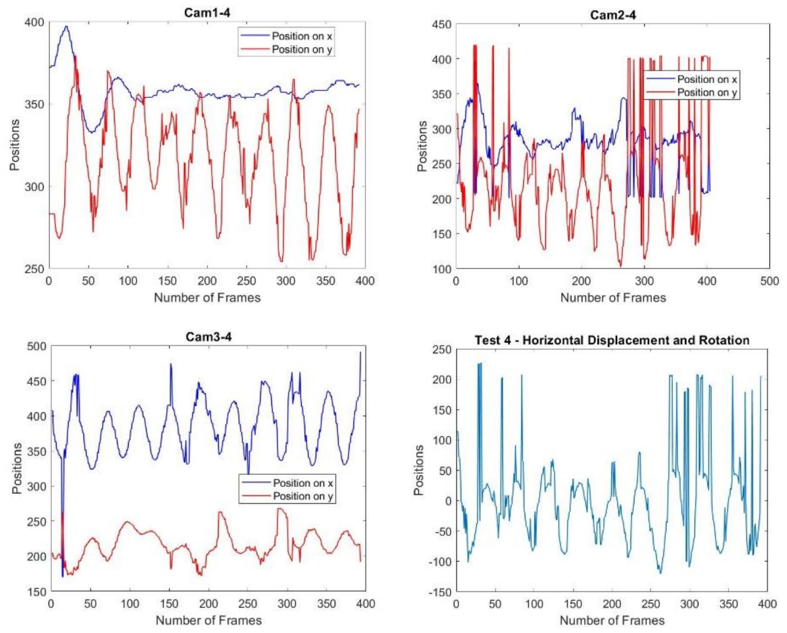

Figure 4: Horizontal Displacement and Rotation

In Figure 4, the dynamics are stronger and quicker when the movements contain not only displacement but also rotations as shown in the first three subplots depicting the mass positions in x – y direction. The fourth subplots of principle component have a denser change in each frame, which is reasonable for the situation of various movements.

3 Summary and Conclusions

Principal component analysis (PCA) can be a very useful way of factoring matrices on different datasets. When there are more elements, including noises, movements, and rotations, in the spring-mass system, the oscillations of the mass positions are identifiable by the strongness and frequencies in the change of positions. For example, in the ideal case in Figure 1, the oscillations are stable and regular in all of the three cameras; however, in Figure 4 that had horizontal displacement and rotation on the mass, the plots provide a hush. If we compare Figure 1 of ideal case and Figure 2 with noisy cases, we can find the graph of principal component has a fewer regular oscillation than the ideal case. Also, for each of the camera 1, 2, 3, we can find that the camera 3 was recorded from a different angle because the x – y positions shifted compared to camera 1 and 2. This is the usefulness of PCA.

References

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

[2] MathWork. https://www.mathworks.com/help/matlab/ref/max.html

[3] Jason J. Bramburger, Lecture Notes for AMATH482, Winter, 2021

[4] Prof. Bramburger, Slack, assignment-3,
https://app.slack.com/client/T01GMTJJGN5/C01M6HYNF18/thread/C01M6HYNF18-1613155658.015100

Appendix A MATLAB Functions

- [row, col] = ind2sub(size, index) returns the arrays row and col containing the equivalent row and column subscripts corresponding to the linear indices ind for a matrix of size sz. Here sz is a vector with two elements, where sz(1) specifies the number of rows and sz(2) specifies the number of columns.
- M = mean(A,dim) returns the mean along dimension dim. For example, if A is a matrix, then mean(A,2) is a column vector containing the mean of each row.
- legend(label1,...,labelN) sets the legend labels. Specify the labels as a list of character vectors or strings, such as legend('Jan','Feb','Mar').
- [M,I] = max(___) also returns the index into the operating dimension that corresponds to the maximum value of A for any of the previous syntaxes.

Appendix B MATLAB Code

(Excerpt of the entire code; Sample code of Ideal Case when others were about the same idea)

```
close all; clc; clear
% Two methods to view the video
% Method 1: use MATLAB'd built-in video player
% load('cam1_1.mat')
% implay(vidFrames1_1)

% Method 2: create a loop and display each frame
% numFrames = size(vidFrames1_1, 4);
% for j = 1:numFrames
%     X = vidFrames1_1(:,:,:,j);
%     imshow(X); drawnow
% end

% Test 1
% Cam1-1
% -- Load Movie
% load ('cam1_1.mat');
% [height width rgb nf1_1] = size(vidFrames1_1);
% -- Watch Movie
% X11 = [];
% Y11 = [];
% for j = 1:nf1_1
%     X = vidFrames1_1(:,:,:,j);
% %     imshow(X); drawnow
% % clean out the noises by just blackening
% % the background, surrounding images
% % Take the cam1_1 as an example, X here is 480*640*3,
% % and the spring-mass is oscilating around pixels around
% % (:,240:480) and (120:460,:)
%     X(:,1:240) = 0;
%     X(:,480:end) = 0;
%     X(1:40,:) = 0;
%     X(460:end, :) = 0;
%     [Max Ind] = max(X(:));
%     [y11 x11] = ind2sub(size(X), Ind);
%     X11 = [X11 x11];
%     Y11 = [Y11 y11];
%     plot(X11,'b','Linewidth',[1]);
%     hold on
%     plot(Y11,'r','Linewidth',[1]);
%     legend('Position on x', 'Position on y','Location','Best')
%     title('Cam1-1')
%     xlabel('Number of Frames')
%     ylabel('Positions')
%     set(gca, 'Fontsize', [12])
% end

% Cam2-1
% -- Load Movie
% load ('cam2_1.mat');
% [height width rgb nf2_1] = size(vidFrames2_1);
% % -- Watch Movie
% X21 = [];
% Y21 = [];
% for j = 1:nf2_1
%     X = vidFrames2_1(:,:,:,j);
% %     imshow(X); drawnow
%     X(:,1:240) = 0;
%     X(:,400:end) = 0;
% %     imshow(X)
%     [Max Ind] = max(X(:));
%     [y21 x21] = ind2sub(size(X), Ind);
%     X21 = [X21 x21];
%     Y21 = [Y21 y21];
%     plot(X21,'b','Linewidth',[1]);
%     hold on
%     plot(Y21,'r','Linewidth',[1]);
%     legend('Position on x', 'Position on y','Location','Best')
%     title('Cam2-1')
%     xlabel('Number of Frames')
%     ylabel('Positions')
%     set(gca, 'Fontsize', [12])
% end

% Cam3-1
% -- Load Movie
% load ('cam3_1.mat');
% % height, width, rgb, num_frames
% [height width rgb nf3_1] = size(vidFrames3_1);
% % -- Watch Movie
% X31 = [];
% Y31 = [];
% for j = 1:nf3_1
%     X = vidFrames3_1(:,:,:,j);
% %     imshow(X); drawnow
% %     keep around(200:360,:) and (:,1:510)
%     X(1:230,:) = 0;
%     X(340:end,:) = 0;
%     X(:,1:60) = 0;
%     X(:,500:end) = 0;
% %     imshow(X)
%     [Max Ind] = max(X(:));
%     [y31 x31] = ind2sub(size(X), Ind);
%     X31 = [X31 x31];
%     Y31 = [Y31 y31];
%     plot(X31,'b','Linewidth',[1]);
%     hold on
%     plot(Y31,'r','Linewidth',[1]);
%     legend('Position on x', 'Position on y','Location','Best')
%     title('Cam3-1')
%     xlabel('Number of Frames')
%     ylabel('Positions')
%     set(gca, 'Fontsize', [12])
% end

% % To make the num_frames to be the same that equals to the shortest,
% % we need to find the minimum length by using:
% Xmin = min([length(X11) length(X21) length(X31)]);
% X11 = X11(1:Xmin);
% X21 = X21(1:Xmin);
% X31 = X31(1:Xmin);
% Y11 = Y11(1:Xmin);
% Y21 = Y21(1:Xmin);
% Y31 = Y31(1:Xmin);
% matrix = [X11;Y11;X21;Y21;X31;Y31];
% [m,n] = size(matrix);
% % Find the mean
% miu = mean(matrix,2);
% % miu = sum(matrix)/(h*w);
% new_matrix = zeros(m,n);
% % Subtract the mean from all entries in matrix we have
% for i = 1:n
%     for j = 1:m
%         new_matrix(j,i) = matrix(j,i) - miu(j,1);
%     end
% end
% % Find covariance by the formula
% Cx = (1/(n-1)) * new_matrix * new_matrix';
% [V,D] = eig(Cx);
% lambda = diag(D);
% Y = V'*new_matrix;
% plot(Y(1,:),'Linewidth',[1])
% title('Test 1 - Ideal Case')
% xlabel('Number of Frames')
% ylabel('Positions')
% set(gca, 'Fontsize', [12])
```

Listing 1: Excerpt Code from MATLAB