PCA and a Spring-Mass System

Jiasui Qin

February 23, 2020

Abstract

Consider a situation when we want to build a model using data with a number of different variables, training the model with all features would be inefficient and may lead to the problem of overfitting. Therefore, it's a good idea to extract and use only the important features that best represent data. PCA is one of the simplest and most robust ways of doing dimensionality reduction. This paper will illustrate various aspects of Principal Component Analysis(PCA), its practical usefulness, and the effects of noise on the PCA algorithms.

1 Introduction and Overview

We have 12 movie files of a spring-mass system created from three different cameras. There are four different situations, each situation has three movie files taken from three cameras. The mass is hanging from a spring and moves up and down. Our goal is to understand the motion of the mass-spring system by using the Principal Component Analysis(PCA).

Four situations are:

1. Ideal case: the entire motion is in the z direction with simple harmonic motion being observed.
2. Noisy Case: repeat the ideal case experiment with camera shakes in the video recording.
3. Horizontal Displacement: the mass is released off-centre so as to produce motion in the x-y plane as well as the z direction.
4. Horizontal Displacement and Rotation: the mass is released off-centre and rotates so as to produce rotation in the x-y plane as well as the z direction.

2 Theoretical Background

2.1 Covariance Matrix

The covariance matrix measures the statistical dependence/independence between all possible pairs of measurements. It's a square, symmetric $m \times m$ matrix where m is the dimension of the original data. The diagonal terms of the matrix represents the variance of a particular variable, and the off-diagonal terms shows the covariances between each pair of variables. This covariance matrix helps us find redundancy as well as identify variables with large variance. Consider two data sets a and b, if the covariance of a and b is equal to the variance of a and is also equal to the variance of b, we can conclude that these two variables are dependent. Therefore, a large off-diagonal term in the matrix represents strong dependency. Dependent variables are considered redundant, which can be removed to reduce the dimensionality. Moreover, a large variance suggests strong fluctuations in that variable, which should be considered as an important feature of the data. [3]

2.2 Diagonalization

We assume that there exists an ideal basis in which the covariance matrix can be diagonalized. In this basis, all redundancies of the original data are removed, and variables with the largest variances are ordered. Since the covariance matrix is square and symmetric, its eigenvalues are real and distinct, and the eigenvectors are also orthogonal. Therefore, it's beneficial to apply the standard eigenvalue/eigenvector expansion technique. After computing the eigenvalues and the eigenvectors, we can build the eigenvector matrix with columns representing each eigenvetor, and the eigenvalue matrix with each diagonal value representing each eigenvalue. The jth diagonal value in the eigenvalue matrix is the variance of the jth column of the eigenvector. These eigenvectors are then called principal components. After getting the principal component basis, we can then project the original data onto the new basis with fewer dimensions. [3]

2.3 Principal Component Analysis

Principal Component Analysis(PCA) is a dimensionality-reduction method that helps reduce the dimensionality of large data sets. It transforms the original data into a new basis with fewer dimensions that still contain most of the information. The algorithm involves the previous two sections, which are building the covariance matrix from the original data and diagonalizing the covariance matrix to extract the principal components.

By applying the PCA algorithm, we can essentially get three types of information about the data:

1. A measure of how each variable is associated with one another. (Covariance matrix)
2. The directions in which our data spreads. (Eigenvectors)
3. The relative importance of these different directions. (Eigenvalues)

3 Algorithm Implementation and Development

3.1 Movement Tracking

In the frame of each video, there are two dimensions. For each instance in time, we need to find the (x, y) position of the mass to track how the mass moves. We can take advantage of the bright spot on the mass. Since a large pixel value in a grayscale image represents a brighter area, we can eliminate the background by only keeping areas with large pixel values. After examining the data, I found that the pixel value of the bright spot is normally larger than 253. Therefore, we can reduce all other pixels that have values less than 253 to 0, which helps us extract only the movement of the mass. Then, we can track the point that has the largest change in pixel value between adjacent frames, since the mass is the only object that is moving. In addition, we can trim the image to only include the mass-spring system to avoid other bright areas in the image that are uncorrelated to the movement of the mass.

General steps:

1. Trim the image to only include the mass-spring system.
2. Change the image to grayscale by using the Matlab function rgb2gray()[4].
3. Reduce pixel values that are less than 253 to 0.
4. Collect the data point with the largest difference in the pixel value from frame to frame.
5. Take the average position of these points if there are more than one.

3.2 PCA

After extracting the (x, y) position of the mass at each instance in time, we can then make vectors of all the positions extracted from each camera and get the following matrix:

$$X = \begin{bmatrix} x_a \\ y_a \\ x_b \\ y_b \\ x_c \\ y_c \end{bmatrix}$$

We can then apply the PCA algorithm with the following steps:

1. For each vector, find the mean and subtract it from the original value.
2. Compute its covariance matrix by using the Matlab function cov()[1]. (Note: remember to transpose the original data since the function assumes that columns represent different features.)
3. Compute the eigenvectors and eigenvalues of the covariance matrix by using the Matlab function eig()[2]. Then, find the largest eigenvalue and its associated eigenvector.
4. Deriving the new one-dimensinal data set by projecting the original data onto that eigenvector.

Algorithm 1: PCA

```
for i = 1:6 do
    original data(i,:) = original data(i,:)-mean(original data(i,:))
end for
covariance matrix = cov(transpose of the original data)
[V,D] = eig(covariance matrix)
maxV = the eigenvector associated with the largest eigenvalue
new data = maxV×original data
```

4 Computational Results

4.1 Case 1

Eigenvalues of the covariance matrix are: 25.9546537638645, 50.5302593685451, 135.011965309866, 463.591320552771, 2483.31514124448, 4257.54714102044.

Let's look at the extracted points for each camera and the PCA result:

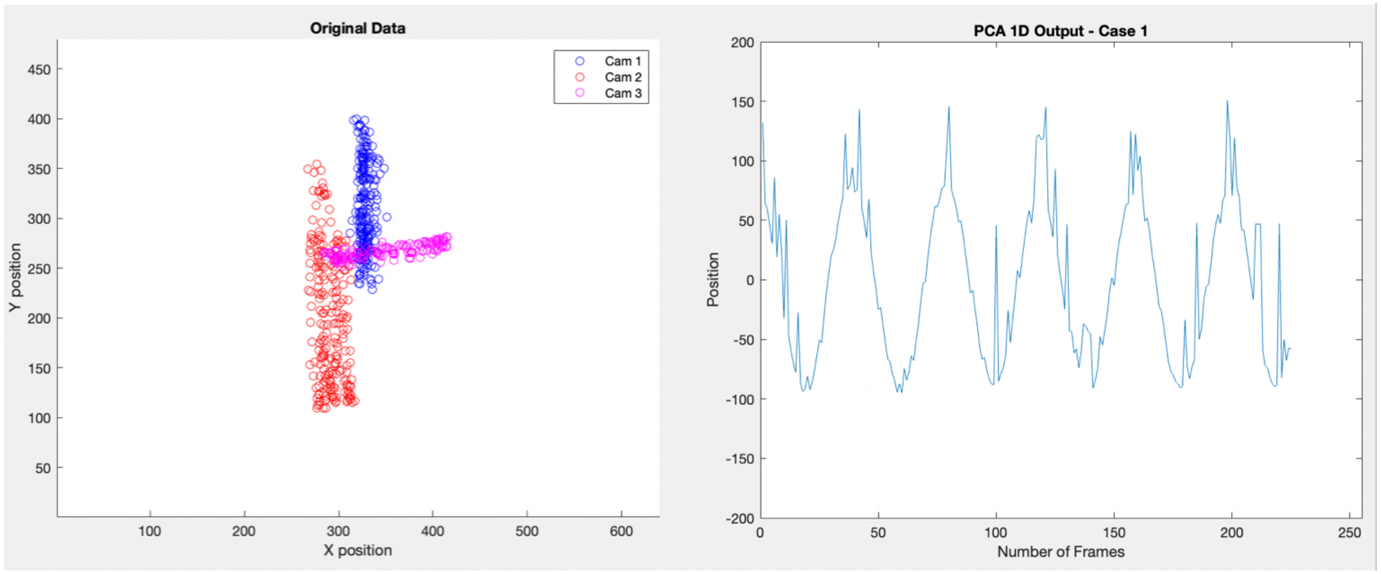

Figure 1: Original data and PCA result of Case 1

From the extracted points, we can conclude that the mass is moving directly up and down in a straight line. It's not moving in any other directions and there is no other noises. After applying PCA, the original data is reduced to one-dimensional data. We can see that it captures the up-and-down movement pretty well, and it tells us the movement of the mass is only one-dimensional in the z direction.

4.2 Case 2

Eigenvalues of the computed covariance matrix are: 51.5136937324474, 158.757584152153, 381.869954047640, 530.998960581641, 1038.66441191930, 3816.63917989821.

The following graph shows the extracted points for each camera and the PCA result:

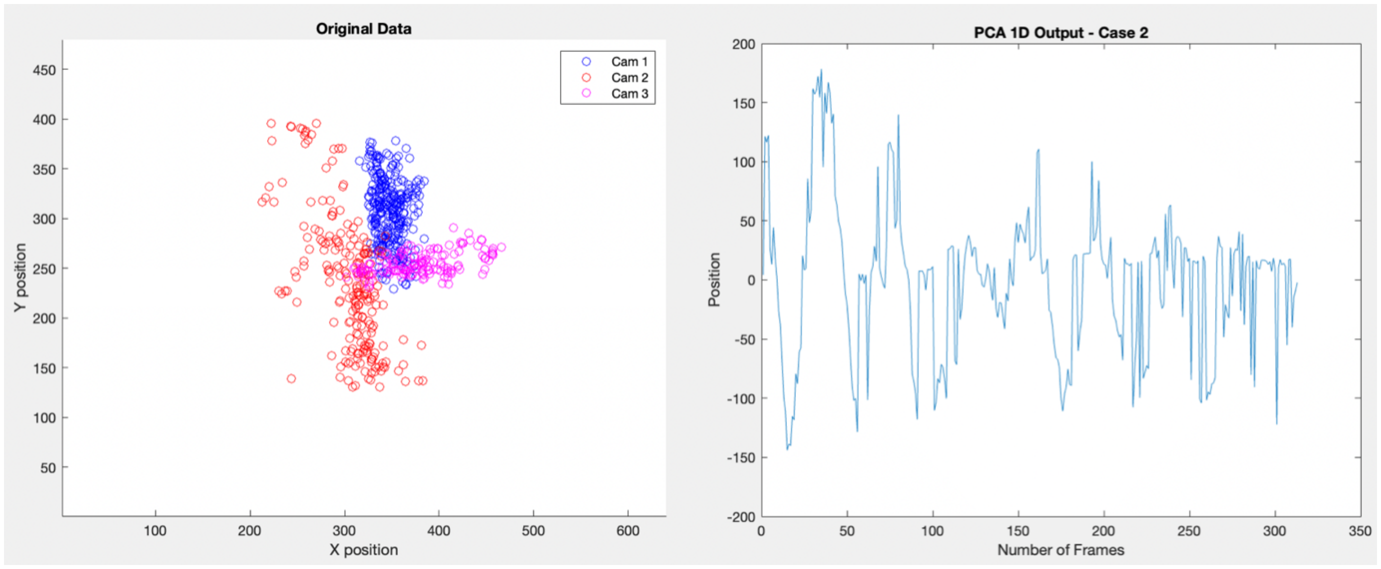

Figure 2: Original data and PCA result of Case 2

Now, we can see from the extracted points that the movement is not stable due to the shaking of the camera. The largest eigenvalue is also lower than the largest eigenvalue in the previous case. By looking at the projected result in one dimension, since the data points we collected show that the mass is moving in other directions as well, it's harder for PCA to capture the exact up-and-down movement.

4.3 Case 3

Eigenvalues of the computed covariance matrix are: 56.0457163928876, 119.776533306731, 325.426506319198, 742.588131369629, 1003.20124537878, 2215.73644946638.

The following graph shows the extracted points for each camera and the PCA result:

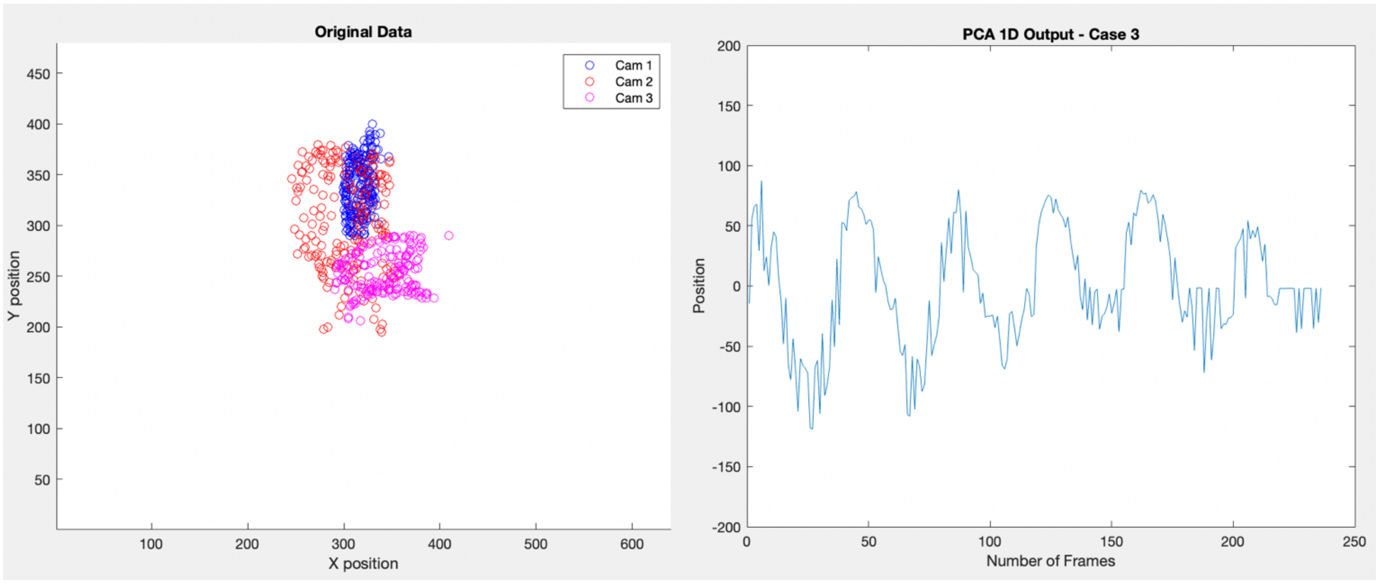

Figure 3: Original data and PCA result of Case 3

From the extracted points, we can see that the points are moving in the shape of an ellipse, showing that the mass is also moving between left and right while moving up and down. In this case, the one-dimensional output cannot capture the entire movement since the movement also happens in the x-y plane. The first three eigenvalues are now closer to each other compared to eigenvalues of the first two cases. This shows us that not only the first principle component is important, we also need to consider the second or the third principle component. However, the first principal component still captures the most important feature.

4.4 Case 4

Eigenvalues of the computed covariance matrix are: 145.803515883300, 387.117035872304, 579.206211565304, 753.965469425261, 1744.66311663745, 2528.94482207138.

The following graph shows the extracted points for each camera and the PCA result:

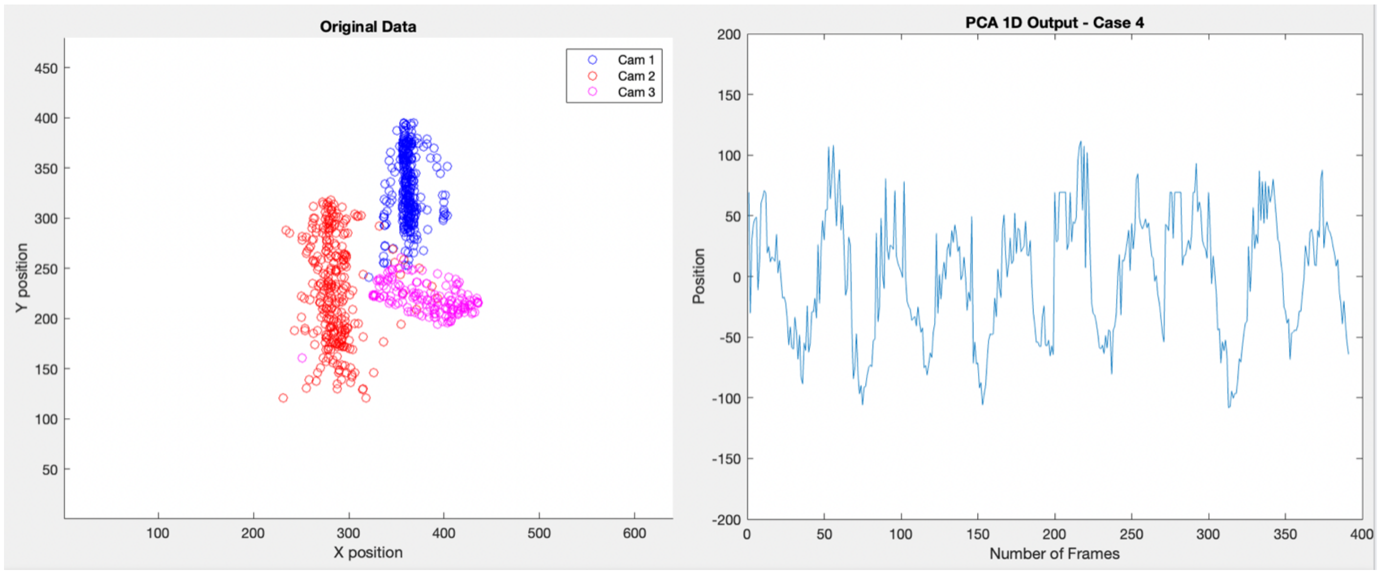

Figure 4: Original data and PCA result of Case 4

In this case, the first two eigenvalues are even closer to each other compared to case 3, even the last three eigenvalues are larger compared to all previous cases. This shows us that the original data must be represented with more dimensions now, since it involves the vertical movement, the horizontal movement, and the rotation of the mass itself. However, the projected one-dimensional result is about the same as the result from case 3, which tells us that PCA successfully extracts the most important feature that represents the vertical movement.

5 Summary and Conclusions

Case 1 sets a benchmark showing how PCA helps us capture the one-dimensional movement correctly when our spring-mass system is only moving in the z direction and the data points collected are clean. After adding noises to the data like case 2, we see that the performance of PCA decreases. When points collected do not show a clear pattern in the change of the movement, it's harder to find the exact correlation between features. For case 3 and case 4, when the movement of the mass is not one-dimensional anymore, we can see that the difference between the first few principal components decreases. This shows that representing data with only one dimension may not be sufficient, other principle components are also important and need to be considered. However, the most important feature that PCA tells us is about the same in all cases, which is the vertical movement in the z direction. The one-dimensional results prove the efficiency and accuracy of PCA for dimensionality reduction.

References

[1] *cov()*. URL: `https://au.mathworks.com/help/matlab/ref/cov.html`.

[2] *eig()*. URL: `https://au.mathworks.com/help/matlab/ref/eig.html`.

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

[4] *rgb2gray()*. URL: `https://au.mathworks.com/help/matlab/ref/rgb2gray.html`.

Appendix A MATLAB Functions

- `zeros(m,n)` returns an m-by-n matrix of zeros.
- `rgb2gray(frame)` changes the frame in RGB to grayscale.
- `[row,col] = find(diff == max(diff(:)))` returns the row number and the column number of the position associated with the maximum value in 'diff' matrix.
- `mean(A)` computes the mean value. If A is a vector, then mean(A) returns the mean of the elements. If A is a matrix, then mean(A) returns a row vector containing the mean of each column.
- `cov(A)` returns the covariance of A, where A is a matrix whose columns represent random variables and whose rows represent observations.
- `[V,D] = eig(A)` returns diagonal matrix D of eigenvalues and matrix V whose columns are the corresponding right eigenvectors, so that A*V = V*D.
- `diag(V)` returns a square diagonal matrix with the elements of vector V on the main diagonal.

Appendix B MATLAB Code

```
clear all; clc; close all
load('cam1_1.mat')
load('cam2_1.mat')
load('cam3_1.mat')

numFrames_1 = size(vidFrames1_1,4)
X_1_1 = zeros(2, numFrames_1-1)
for j = 2:numFrames_1
    frame1 = rgb2gray(vidFrames1_1(200:400,300:400,:,j))
    frame2 = rgb2gray(vidFrames1_1(200:400,300:400,:,j-1))
    threshold1 = frame1
    threshold2 = frame2
    threshold1(frame1<253) = 0
    threshold2(frame2<253) = 0
    diff = abs(threshold1-threshold2)
    [row,col] = find(diff == max(diff(:)))
    X_1_1(1,j-1) = mean(col,'all')+300
    X_1_1(2,j-1) = mean(row,'all')+200
end

numFrames_2 = size(vidFrames2_1,4);
X_2_1 = zeros(2, numFrames_2-1)
for j = 2:numFrames_2
    frame1 = rgb2gray(vidFrames2_1(100:400,200:350,:,j))
    frame2 = rgb2gray(vidFrames2_1(100:400,200:350,:,j-1))
    threshold1 = frame1
    threshold2 = frame2
    threshold1(frame1<253) = 0
    threshold2(frame2<253) = 0
    diff = abs(threshold1-threshold2)
    [row,col] = find(diff == max(diff(:)))
    X_2_1(1,j-1) = mean(col,'all')+200
    X_2_1(2,j-1) = mean(row,'all')+100
end

numFrames_3 = size(vidFrames3_1,4);
X_3_1 = zeros(2, numFrames_3-1)
for j = 2:numFrames_3
    frame1 = rgb2gray(vidFrames3_1(250:300,260:430,:,j))
    frame2 = rgb2gray(vidFrames3_1(250:300,260:430,:,j-1))
    threshold1 = frame1
    threshold2 = frame2
    threshold1(frame1<253) = 0
    threshold2(frame2<253) = 0
    diff = abs(threshold1-threshold2)
    [row,col] = find(diff == max(diff(:)))
    X_3_1(1,j-1) = mean(col,'all')+260
    X_3_1(2,j-1) = mean(row,'all')+250
end
```

Listing 1: Case 1 example

```
dim = min([numFrames_1, numFrames_2, numFrames_3])-1
ori = [X_1_1(:,1:dim)' X_2_1(:,1:dim)' X_3_1(:,1:dim)']
M = mean(ori)
sub_mean = []
for i = 1:6
    sub = ori(:,i)-M(i)*ones(dim,1)
    sub_mean(:,i) = sub
end

scatter(ori(:,1),ori(:,2),'b'),hold on
scatter(ori(:,3),ori(:,4),'r'),hold on
scatter(ori(:,5),ori(:,6),'m')
xlim([1 640])
ylim([1 480])
xlabel('X position')
ylabel('Y position')
title('Original Data')
legend('Cam 1','Cam 2','Cam 3')

cov_matrix = cov(sub_mean)
[V,D] = eig(cov_matrix);
D=diag(D);
maxD = max(D)
maxV=V(:,find(D==max(D)));
finaldata=maxV'*sub_mean';

plot((1:dim),finaldata,'-')
ylim([-200 200])
xlabel('Number of Frames')
ylabel('Position')
title('PCA 1D Output - Case 1')
```

Listing 2: Case 1 example