PCA and a Spring-Mass System

Yichen Shen

Feb 22, 2021

Abstract

In this paper is going to explore the practical usefulness of the Principle Component Analysis by study the motion of mass in a spring-mass system given different situations. We are going to apply singular value decomposition to our data matrix in order to apply the principle component analysis. Upon the PCA information we are able to illustrate the different aspects of the PCA and its usefulness in different situations.

1 Introduction and Overview

1.1 Background and Context

Building up a spring mass system, the experiment creates four different cases with each composed of three different angles' camera footage. We are given four different cases, among which are a small displacement in the z direction, a small displacement in the z direction with shakes, a displacement in all direction off centre, and a displacement in all direction with rotation. In this paper, we are going to apply Principle component analysis to each of the cases and analyze the result of the primary component in each case.

1.2 Topic Importance

Singular value decomposition is a very powerful tool in factorize matrix into small matrix components, by applying the principle component analysis we are able to track the path of the motion of the principle component without looking at the entire components of data. By looking at the principal component of the motion, we are able to analyze the different aspect of PCA in various situations as well as the noise on algorithm.

1.3 Objectives of Research

Each cases of data has some aspect that is worth for us to take a look and see how those noisy, motion in different direction, and rotation is going to affect the the motion of the principle component. We are going to create a graph of position in each coordinate and generate a energy capture graph to illustrate the energy captured by the principle components, lastly, we are going to plot the principle component movement and analyze the motion of the primary component. After we get our data, we can further analysis in order to compare and contrast the the different aspect of Principle component analysis.

2 Theoretical Background

2.1 Singular Value Decomposition SVD

Singular value decomposition is a very powerful tools as it can decompose a matrix in any size. In comparison, the eigenvalue decomposition can only decompose the square matrix. In real world, the data are not always in a square matrix form thus the singular value decomposition can be more robust to use.

The `Singular Value Decomposition` of a matrix A can be written as:

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

In this decomposition, matrix $U \in R^{mxm}$ matrix $V \in R^{nxn}$ are unitary matrices while $\Sigma \in R^{mxn}$ is diagonal. Matrix U is a matrix that construct with a orthonormal columns m-n to the existing $\hat{U}$. The V^* to the left is the `right singular vector` of matrix A and the diagonal value on matrix Σ returns the `singular values` of matrix A, while the U matrix compose of the `left singular vectors`.

Every matrix A has an SVD and that the singular values are always non-negative descending order real numbers along the diagonal of Σ.

2.2 Pinciple Component Analysis

2.2.1 Covariance Matrix

Given two vectors a and b, with a length of n. Assuming the mean is already been subtract, we are able to find the variance of each:

$$\sigma_a^2 = \frac{1}{n-1} aa^T, \sigma_b^2 = \frac{1}{n-1} bb^T \tag{2}$$

From this we can get the *covarience*:

$$\sigma_a b^2 = \frac{1}{n-1} ab^T \tag{3}$$

If now, we are given with a multiple row of vectors stored in a matrix X, the covarience becomes:

$$C_X = \frac{1}{n-1} XX^T \tag{4}$$

2.2.2 Covarience and SVD

Now that we know how to calculate the covarience and the principle of SVD, we are able to connect these two ideas together:

Recall the SVD, we know that matrix A is related to eiggenvalue decomposition of AA^T, with related to the covarience, now we have:

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

then,

$$C_X = \frac{1}{\sqrt{n-1}} XX^T = AA^T = U\Sigma^2 U^T \tag{6}$$

To change the basis of X, we can apply a transpose of U, which gives us:

$$Y = U^T X \tag{7}$$

Matrix Y is the matrix in the basis of principal component and we are able to retrieve the principle component information from this matrix as well as proceeds the analysis.

3 Algorithm Implementation and Development

In order to extract the max position from the video frame, first we are going to load and play the video using Matlab build-in command. Each file is consist of 4 dimensions, including the m and n size of each frame image and the rgb information as well as the number of frame. Then we need to build a loop to view the video frame by frame.

3.1 Extract Mass Position

In order to track the location of the mass (can), we are looking at the white block on the can which is the brightest point that we can use to track the movement of the can. In order to easily identify the bright spot, we change the image to gray-scale with command `rgb2gray`. To reduce the potential influence of the light spot of the background area, we cover up the unnecessary part of the image only leave out the white block movement area. The white block can be rotated away from the image, in this case, the white part of the can is going to become the light spot to be track, as long as it is the brightest pixel in the image. Notice when using the `ind2sub` command, the first element in the result matrix is the y coordinate and the second is the x coordinate.

Algorithm 1: Extract the mass position

Load the video of $cam1_N.m4a$
Load the dimension information using $size(vidFrames1_N, 4)$
for $j = 1 : numberofframes$ **do**
  Apply the `rgb2gray` and the `im2double` command in order to convert the image into gray scale integer data
  Cover up the background area where the can is not moving in.
  Find the coordinate of the location of maximum pixel using `[M,I] = max(X)` and `[y,x] = ind2sub` command
end for
Plot the position of each coordinate with respect to time frame.

3.2 Time Alignment

Note that each video is in a different time length, for us to complete the PCA procedure, we will need to align the time of each video to the same length. To do this, we set the time to the shortest video: cam1. Then for the first video, we trace the location of the first minimum point (so that we are not trim the video to too short if there is another minimum later) and we line each coordinate accordingly. For video 2 and 3, we are doing the same thing except that the length of the video is trim to be the same as the first video.

3.3 Principle Component Analysis

In order to process principle component analysis, we first need to apply singular value decomposition. Storing all the position vector into a matrix, and subtract the mean from each vector to center our data at 0, we then are ready to compute our singular value.

Algorithm 2: Apply Principal component analysis

Store the aligned vector into a matrix
Subtract the mean from each vector in the matrix by using `mean` and `repmat` command
Calculate covariance matrix C_X using $1/(n-1)XX^T$
Calculate A from the Covariance matrix using $\frac{X}{\sqrt{n-1}}$
Calculate SVD using `svd()` command, and write it in the new basis of principal component using $Y = U^T X$
Plot the Principle component graph as well as the energy/singular value/cumulative energy using $sigma.^2/sum(sigma.^2)$.

4 Computational Results

From our previous algorithm, we plot three types graph for each case, the position of mass graph, the energy graph, and the principle component graph.

4.1 Position of the Mass

After apply algorithm 1 and 2, we generate the position graph for each case, as we can see from Figure 1, the motion in the x direction is very small while we can obtain a clear motion of the can in the y direction. (In cam 3, the pattern of x and y are just reversed). In comparison, Figure 2 which is consist of the noisy case, has more motion detected in the x axis and the motion in the y axis is less smooth than that in the first case. Figure 3, which consist of a horizontal displacement, has a clear motion in both x and y direction, and for Figure 4, which consist of a horizontal displacement as well as a rotation its y motion is clearly recorded while the displacement on the x axis is damping and less clear than that in case3.

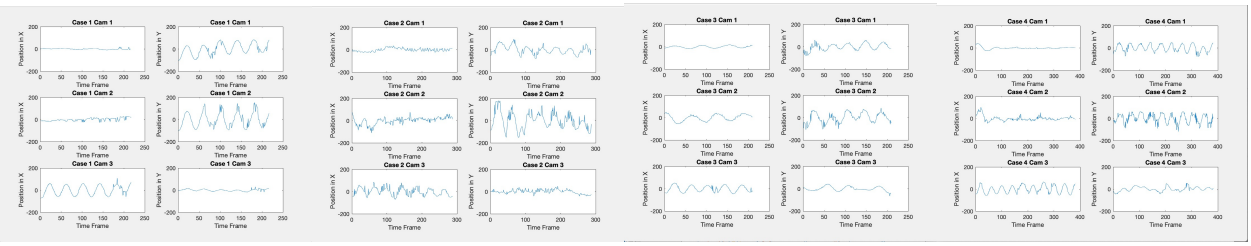

Figure 1: The position graph in Case 1: Ideal Case

Figure 2: The position graph in Case 2: Noisy Case

Figure 3: The position graph in Case 3: Horizontal Displacement

Figure 4: The position graph in Case 4: Horizontal Displacement and Rotation

4.2 Energy Capture

As we captured the energy with respect to the singular value as well as the cumulative energy in each case. We can compare the result and see the pattern from them. In Figure 5, the first singular value point of Case1 has a 87.66% of the total energy, and in Figure 6, the first singular value point of Case2 decreases to an amount of 58.82%, following by an amount of 64.04% total energy from the first point in Case3 (Figure 7) and an amount of 58.82% total energy from the first point in Case4 (Figure 8). The first singular value point in case1 capture the most energy because it is one dimensional motion; however, for a noisy case, it takes about three singular value to capture almost all the energy indicated that the noisy situation has a great impact on the ability of PCA. As we compare Case3 and Case4 which both have a motion in the x and y direction. They both takes about three singular value to capture almost all the data, and from the comparison we can see that rotation does not really affect the ability of PCA but that a motion in the x axis and noisy has the greatest impact on PCA.

4.3 Principle Components Analysis

Lastly, we are going to take a look at the principle component of movement graphs, in case 1 (Figure 9), the variance of the first principle component is pretty clear and we can see that the amplitude of variance of the second principle component is very stable around 0. Therefore, the first principle component is able to capture almost all the energy. However, as it gets to the second case (Figure 10), with the first principle component still has great amplitude of variance, the amplitude second and third principle components' variance is higher than the previous one. The second and third component takes up part of the energy, therefore, it can explain the reason why we need three components to capture almost all the energy in case2. Similarly it is for case3 (Figure11) and case4 (Figure12), we can see that the first three principle components has taken up almost all the energy. Particularly, in case4, the motion is the most complicated because of the horizontal displacement as well as the rotation.

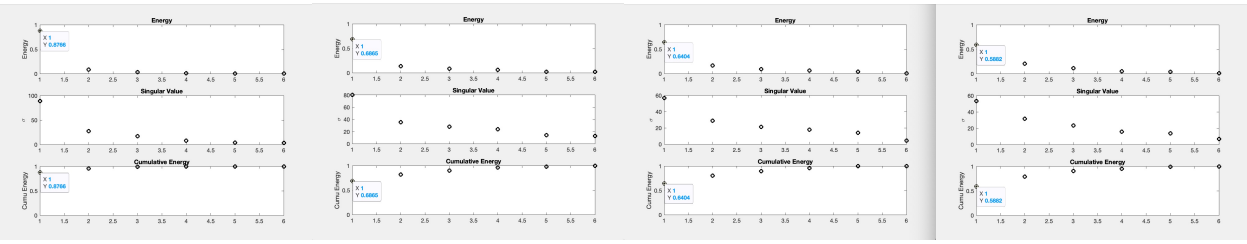

Figure 5: The energy captured in Case 1: Ideal Case

Figure 6: The energy captured in Case 2: Noisy Case

Figure 7: The energy captured in Case 3: Horizontal Displacement

Figure 8: The energy captured in Case 4: Horizontal Displacement and Rotation

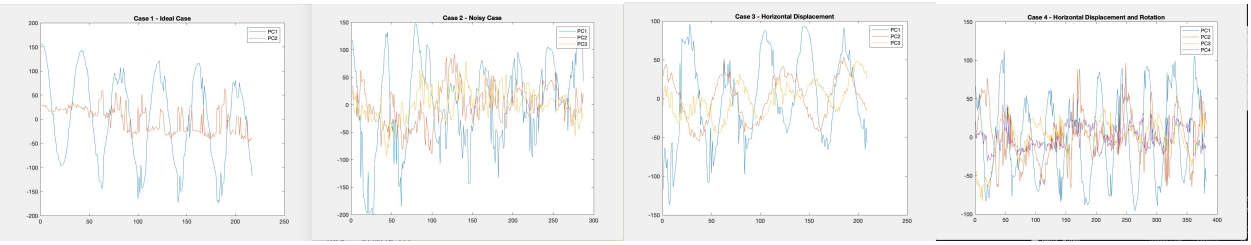

Figure 9: The Principle components of movement in Case 1: Ideal Case

Figure 10: The Principle components of movement in Case 2: Noisy Case

Figure 11: The Principle components of movement in Case 3: Horizontal Displacement

Figure 12: The Principle components of movement in Case 4: Horizontal Displacement and Rotation

5 Summary and Conclusions

This research paper studies on the topic of PCA and how it perform in difference situations. PCA is a powerful tool to find the component demonstrate the most significant component of a data set, in this case, we use PCA to find the primary motion of a spring-mass system. In order to employ this robust tool, this paper prepares the theoretical background which includes the Singular Value Decomposition and the Principle component analysis. After implement the algorithm, the paper documents the computational results which includes the motion of the mass in four cases, the energy captured by each cases, and the principle component of movement for each cases. The significance of this research is the implementation indicates that one can uses PCA to process data and spot the most significant component among the data, since data are always composed with different components, it is important to understand the background theology and algorithms to prepare for the further implementation on the principle component analysis.

Appendix A MATLAB Functions

Important MATLAB functions with a brief implementation explanation.

- `vidFrames` read the video file.
- `rgb2gray` convert the image from its true color rgb to a gray scale image.
- `[M,I] = max(A)` returns the maximum elements of an array A.

- `ind2sub` return the row and column information correspond to the index.
- `[U,S,V] = svd(A,'econ')` produces an economy sized singular value decomposition of a m by n matrix A.
- `im2double` convets the image to a double precision.
- `diag(A)` returns a column vector of the diagonal elements of matrix A.

Appendix B MATLAB Code

The following code is to perform the algorithm 1 2 and 3 on case 1. The code for the other cases are pretty similar to each other with only some modify on the area cover part in algorithm 1.

```
%% Load video
clear all; clc
load('cam1_1.mat')
load('cam2_1.mat')
load('cam3_1.mat')
implay(vidFrames1_1)
implay(vidFrames2_1)
implay(vidFrames3_1)

numFrames11 = size(vidFrames1_1,4);
% [height11 width11 rgb11 numFrames11] = size(vidFrames1_1);
numFrames21 = size(vidFrames2_1,4);
numFrames31 = size(vidFrames3_1,4);
%% Cam 1

xa = zeros(1,numFrames11);
ya = zeros(1,numFrames11);

for j = 1:numFrames11
    X = vidFrames1_1(:,:,:,j);
    X = rgb2gray(X);
    %X11(:,:,j) = im2double(X);
    X = im2double(X);

    X(:,1:300) = 0;
    X(:,400:end) = 0;
    X(1:200,:) = 0;

    %imshow(X); drawnow
    [M,I] = max(X(:));
    [y,x] = ind2sub(size(X),I);
    xa(j) = x;
    ya(j) = y;
    %xa = [xa,x];
    %ya = [ya,y];
end

%% Cam 2

xb = zeros(1,numFrames21);
yb = zeros(1,numFrames21);

for j = 1:numFrames21
    X = vidFrames2_1(:,:,:,j);
    X = rgb2gray(X);
    %X21(:,:,j) = im2double(X);
    X = im2double(X);

    X(:,1:250) = 0;
    X(:,350:end) = 0;
    X(1:80,:) = 0;
    X(400:end,:) = 0;

    %imshow(X); drawnow
    [M,I] = max(X(:));
    [y,x] = ind2sub(size(X),I);
    xb(j) = x;
    yb(j) = y;
end
```

```
%% Cam 3

xc = zeros(1,numFrames31);
yc = zeros(1,numFrames31);

for j = 1:numFrames31
    X = vidFrames3_1(:,:,:,j);
    X = rgb2gray(X);
    %X31(:,:,j) = im2double(X);
    X = im2double(X);

    X(:,1:250) = 0;
    X(:,480:end) = 0;
    X(1:240,:) = 0;
    X(330:end,:) = 0;

    %imshow(X); drawnow
    [M,I] = max(X(:));
    [y,x] = ind2sub(size(X),I);
    xc(j) = x;
    yc(j) = y;
end

%% Alignment
yamin = min(ya);
yamin_loci = find(ya == yamin);
linedxa = xa(yamin_loci(1):end);
linedya = ya(yamin_loci(1):end);

linelength = length(linedya);
ybmin = min(yb);
ybmin_loci = find(yb == ybmin);
newlength = ybmin_loci(1)+linelength-1;
linedxb = xb(ybmin_loci(1):newlength);
linedyb = yb(ybmin_loci(1):newlength);

xcmin = min(xc);
xcmin_loci = find(xc == xcmin);
newlength2 = xcmin_loci(1)+linelength-1;
linedxc = xc(xcmin_loci(1):newlength2);
linedyc = yc(xcmin_loci(1):newlength2);

%plot(1:linelength,linedya,1:linelength,linedyb,1:linelength,linedxc);

%% SVD
vec = [linedxa;linedya;linedxb;linedyb;linedxc;linedyc];
[m,n] = size(vec);
mn = mean(vec,2);
vec = vec-repmat(mn,1,n);

CX = (1/(n-1))*vec*vec';
A = vec/sqrt(n-1);
[U,S,V] = svd(A, 'econ');
Y = U'*vec;
```

Listing 2: Code to perform algorithms 123

```
%% Position Graph

figure(1)
subplot(3,2,1)
plot(vec(1,:))
%plot(1:numFrames11,vec(1,:))
%plot(xa);
ylim([-200,200])
xlabel('Time Frame')
ylabel('Position in X')
title('Case 1 Cam 1')

subplot(3,2,2)
plot(vec(2,:))
%plot(1:numFrames11,vec(2,:))
%plot(ya);
ylim([-200,200])
xlabel('Time Frame')
ylabel('Position in Y')
title('Case 1 Cam 1')

subplot(3,2,3)
plot(vec(3,:))
%plot(1:numFrames21,vec(3,:))
%plot(xb);
ylim([-200,200])
xlabel('Time Frame')
ylabel('Position in X')
title('Case 1 Cam 2')

subplot(3,2,4)
plot(vec(4,:))
ylim([-200,200])
%plot(1:numFrames21,vec(4,:))
%plot(yb);
xlabel('Time Frame')
ylabel('Position in Y')
title('Case 1 Cam 2')

subplot(3,2,5)
plot(vec(5,:))
%plot(1:numFrames31,vec(5,:))
%plot(xc);
ylim([-200,200])
xlabel('Time Frame')
ylabel('Position in X')
title('Case 1 Cam 3')

subplot(3,2,6)
plot(vec(6,:))
%plot(1:numFrames31,vec(6,:))
%plot(yc);
ylim([-200,200])
xlabel('Time Frame')
ylabel('Position in Y')
title('Case 1 Cam 3')
```


Listing 3: Code to perform algorithms 123

```
%% Energy Captured
sigma = diag(S);

energy = zeros(1,length(sigma));
for j = 1:length(sigma)
    energy(j) = sigma(j)^2/sum(sigma.^2);
end
figure(2)
subplot(3,1,1)
plot(energy,'ko','Linewidth',2)
ylabel('Energy')
ylim([0,1])
title('Energy')
subplot(3,1,2)
plot(sigma,'ko','Linewidth',2)
ylabel('\sigma')
title('Singular Value')
subplot(3,1,3)
plot(cumsum(energy),'ko','Linewidth',2)
ylabel('Cumu Energy')
ylim([0,1])
title('Cumulative Energy')

%% PCA
figure(3)
plot(Y(1,:)), hold on
plot(Y(2,:))
title('Case 1 - Ideal Case')
legend('PC1','PC2')
```

Listing 4: Code to perform algorithms 123