Can we PCA it all Together: Testing Principal Component Analysis in Various Circumstances

Timothy Lu

Abstract

We examine the outcome from applying Principal Component Analysis (PCA) to video footage of an oscillating mass system. PCA is applied in various situations, such that we observe the effects of noise, additional horizontal displacement, and rotation on the output. This will be accomplished by tracking the position of a can in x and y coordinates and taking the Singular Value Decomposition (SVD) of the data and comparing the results across various situations.

1. Introduction and Overview

Principal Component Analysis (PCA) is a useful tool to analyze data. PCA reduces dimensionality by picking a dimension or a viewpoint where projected data onto a given direction (used as a basis vector) can be easily analyzed. For the purposes of this report, PCA is used to analyze video data of a can attached to a spring undergoing simple harmonic motion. The aim is to use this action of a mass-spring system to test how PCA responds to variations of an oscillating mass (the can) attached to a spring. The following is tested:

1) *An ideal case:* where there is a small displacement in the z direction, and the can is moving in simple harmonic motion.
2) *A noisy case:* where the mass is moving the same way as case 1, but the video of the mass moving is shaking resulting in noisy data.
3) *Horizontal displacement:* where the mass is released off center, so that it moves back and forth like a pendulum in the $x-y$ plane while oscillating in the z direction as well
4) *Horizontal displacement and rotation:* where the mass is released off center much like case 3, but also in a way that it rotates in addition to oscillating in the z direction.

2. Theoretical Background

The foundation of our analysis stems from *Singular Value Decomposition* (*SVD*). SVD is "a factorization of (a) matrix into a number of constitutive components all of which have a specific meaning in applications" (Kutz, 102). Moreover, applying SVD to a matrix means transforming a matrix into the product of three separate matrices. This is illustrated in the following equation (1).

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

where A is a given $m \times n$ matrix to perform the SVD, $\hat{U}$ is an $m \times n$ matrix with orthonormal columns, $\hat{\Sigma}$ is an $n \times n$ diagonal matrix with positive elements (given matrix A is full rank) and V^* is the conjugate transpose of an $n \times n$ unitary matrix.

SVD can be visualized as a transformation that stretches by a matrix $\hat{E}$ and rotates by a matrix $\hat{U}$. It is also worth noting that every matrix $A \in \mathbb{C}^{m\times n}$ has a SVD and the singular values $\{\sigma_j\}$ are uniquely determined per a theorem in Kutz notes page 104.

There are several use cases of the SVD that prove essential to linear algebra such as for diagonalizing matrices (and getting eigenvalues), however they will not be discussed here.

Principal Component Analysis is a key application of SVD. To understand why SVD is so crucial to the task at hand, we must first break down the "goals" of our task at hand. That is, given frame by frame data, we are trying to track the movement of the can, and draw conclusions from what we find. Our goals then are to: (i) remove redundancy given our three data sources of the same event and (ii) to identify the signal with maximal variance (Kutz, 166). To gauge redundancy, we use *covariance,* or more specifically, a *covariance matrix* which is generally of the form as shown in equation 2.

$$C_x = \frac{1}{n-1} X X^T \tag{2}$$

where C_x is a square, symmetric $m \times m$ matrix whose diagonal represents the variance of particular measurements and off-diagonal values are zero and X is a matrix of all data from various sources gathered together in the form

$$X = \begin{bmatrix} x_a \\ y_a \\ x_b \\ y_b \\ x_c \\ y_c \end{bmatrix} \tag{3}$$

where x_a, y_a ... are the plotted x and y positions from a given data source (camera angle).

Thus, we are looking to find a covariance matrix of our data sources with values only in the diagonals and with values ordered largest to smallest. SVD accomplishes that exactly while diagonalizing as well as capturing as much energy as possible as measured by the singular values σ_j (Kutz, 116).

3. Algorithm Implementation and Development

The algorithm is divided up into the following parts:

1) Import the frame data and cut down the size to improve computational speed and performance (cropping to relevant areas and converting to grayscale).
2) Go through every frame of the video looking for high light intensity pixels that are over a given threshold (245 in this case).
3) Store the location of the mean of the high intensity pixels as the x and y position of the paint can (where x and y are equivalent to the $x-y$ plane and z respectively as given in part 1).
4) Store data gathered from all three cameras into one matrix in order to perform SVD.
5) Examine energies found from the SVD to get principal components.

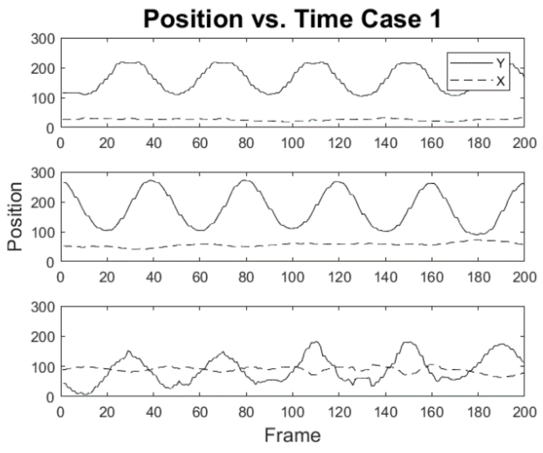

Figure 1 - Cam1 (top) Cam2 (mid) and Cam3 (bottom)

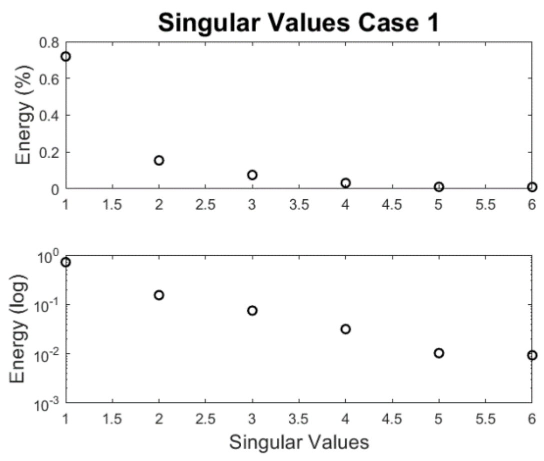

Figure 2 - Singular values and associated energies in case 1

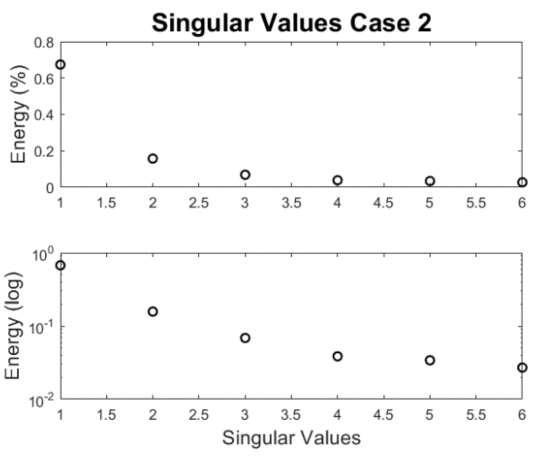

Figure 4 - Singular values and associated energies in case 2

4. Computational Results

4.1 Case 1

Being the ideal case, this proved relatively simple to track. Although there was some noise in camera 2 and 3, the oscillations are very even and the simple harmonic motion is very clearly tracked and plotted. The motion of the can as tracked by each camera is plotted in figure 1. Given this plot, we see that the majority of the motion is in the "y" direction (or z) while the "x" direction tends to not vary.

The SVD (as plotted in figure 2) applied to the matrix conatining data from all three cameras reveals that most of the energy is associated with a single singular value. This means that we see the majority of the variation or motion of the system is in one direction, as to be expected given this case.

4.2 Case 2

When noise is added to the system, we see that first of all, the tracking of the can becomes noisy. As depicted in figure 3, although one can make out waves displaying oscillation, there are large variations in the data that make it unsmooth. That being said, the oscillations are still roughly visible, and it is still clear that the y component is the source of motion.

This becomes abundantly clear when reviewing the results of SVD in figure 4.

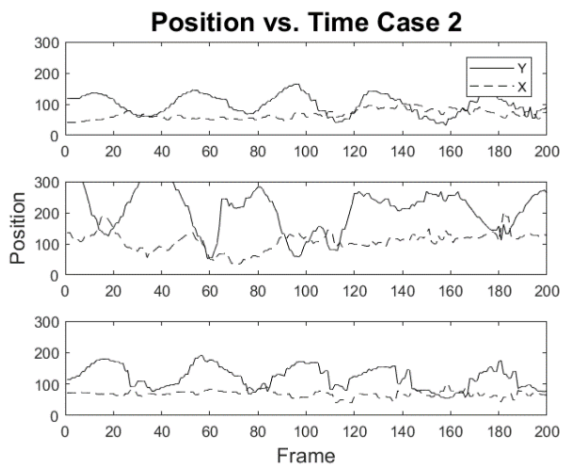

Figure 3 - Cam1 (top) Cam2 (mid) and Cam3 (bottom)

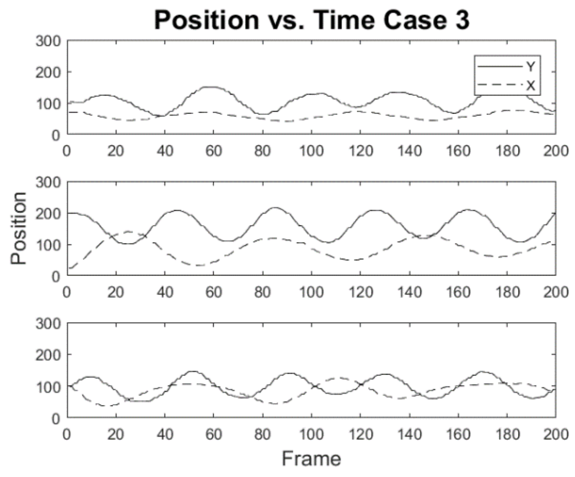

Figure 5 - Cam1 (top) Cam2 (mid) and Cam3 (bottom)

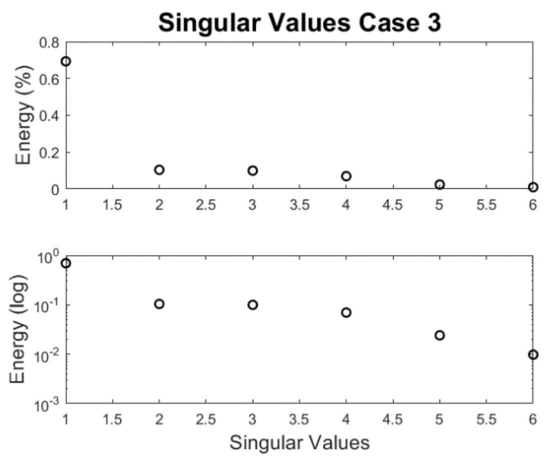

Figure 6 - Singular values and associated energies in case 3

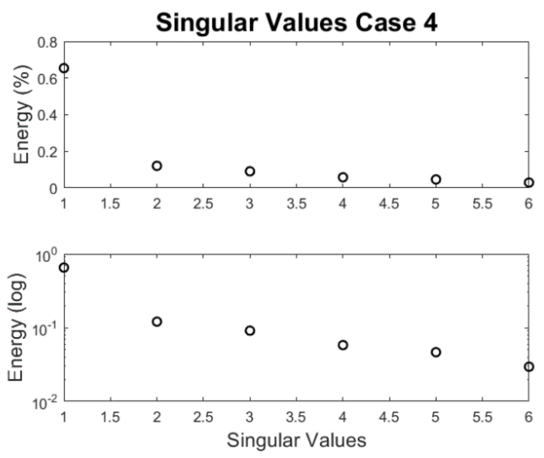

Figure 8 - Singular values and associated energies in case 4

Figure 4 appears to be very similar to figure 2 in that the majority of energy is focus in one mode (singular value). This means that the motion is still only really in one component. Thus, the principal component is still quite pronounced. It is noteworthy, however, that the percent of energy stored in the first node is marginally less than that of case 1 implying that the principal component may be slightly less pronounced.

4.3 Case 3

Here we see motion in both the "x" ($x - y$ plane) and the "y" (z direction) positions as illustrated in figure 5. The motion here is very clearly simple harmonic motion in both directions as the system oscillates in both directions.

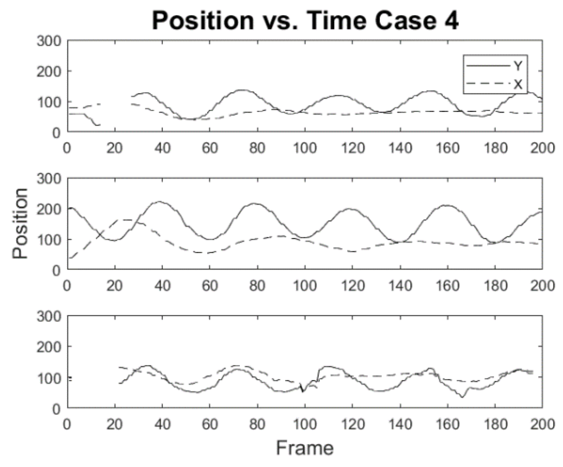

Figure 7 - Cam1 (top) Cam2 (mid) and Cam3 (bottom)

Although movement in both directions are pronounced in figure 5, the SVD of the data for case 3 tells a slightly different story as shown in figure 6. Much like case 1 and 2, there is one singular value with the majority of the energy with other modes not even coming close. This suggests that although there was motion in a different direction, the principal component would be in the same direction as in case 1 and 2.

4.4 Case 4

With added rotation, there is rotational motion in the $x - y$ plane (or the "x" direction). For the purposes of figure 7, there is little changed as the added motion was added rotational motion around the z axis. From the perspective given in these plots, this motion is relatively unaccounted for.

The SVD here shows nothing much different meaning that the principal component found would be the same as case 1, 2, and 3.

5. Summary and Conclusions

In our tests of noisy data, added horizontal movement and rotation, we see that the PCA, although energy may be slightly affected, still gives the same principal component. Because the aim of applying PCA to a set of data is to reduce dimensionality and redundancy, we see that the goals of performing PCA on data is still accomplished regardless of noise or added redundancies. Case 2 tested the affect of noise on our PCA output, while case 3 and 4 tested the effects of redundancy on PCA. Thus, we see that PCA is a very powerful tool to reduce dimensionality and find the optimal dimension to examine a given amount of data.

Appendix A

Command	Implementation
size	Returns the size of a given matrix. Used to get the number of frames in a given video.
imcrop	Used to crop an image (a frame of data) to a given vector of dimensions.
rgb2gray	Used to convert a color photo in RGB to grayscale.
zeros	Creates an empty matrix of values of a given size. Used to pre-allocate space for data such that the script is more efficient.
find	Finds all values that fit a description. Used to find all bright values over a given threshold.
mean	Gives the arithmetic average of given data. Used to locate the center of mass of bright points to find the can.
double	Used to turn frame data into parse-able double values.
svd	Performs the SVD on a given matrix. Used to single out singular values for principal component analysis.
isnan	Checks for NaN values. Used to avoid errors.
diag	Returns a vector representation of a diagonal of a matrix. Used to get the sigma values from SVD.

Appendix B

1. *Setup*

```
clear all; close all; clc

cam_num = 4; % for convenience of making plots

cropsize11 = [300 160 110 240];
cropsize12 = [230 70 150 310];
cropsize13 = [120 280 130 220];
cropsize21 = [280 220 150 200];
cropsize22 = [180 60 220 340];
cropsize23 = [150 280 150 220];
cropsize31 = [260 230 140 170];
cropsize32 = [220 150 280 300];
cropsize33 = [150 280 170 220];
cropsize41 = [300 250 90 200];
cropsize42 = [200 100 270 260];
cropsize43 = [180 320 170 200];

% load data
if cam_num == 1
    load('cam1_1.mat');
    load('cam2_1.mat');
    load('cam3_1.mat');
    v1 = vidFrames1_1; v2 = vidFrames2_1; v3 = vidFrames3_1;
    clear vidFrames1_1 vidFrames2_1 vidFrames3_1;
    cropsize1 = cropsize11; cropsize2 = cropsize12; cropsize3 = cropsize13;
elseif cam_num == 2
    load('cam1_2.mat');
    load('cam2_2.mat');
    load('cam3_2.mat');
    v1 = vidFrames1_2; v2 = vidFrames2_2; v3 = vidFrames3_2;
    clear vidFrames1_2 vidFrames2_2 vidFrames3_2;
    cropsize1 = cropsize21; cropsize2 = cropsize22; cropsize3 = cropsize23;
elseif cam_num == 3
    load('cam1_3.mat');
    load('cam2_3.mat');
    load('cam3_3.mat');
    v1 = vidFrames1_3; v2 = vidFrames2_3; v3 = vidFrames3_3;
    clear vidFrames1_3 vidFrames2_3 vidFrames3_3;
    cropsize1 = cropsize31; cropsize2 = cropsize32; cropsize3 = cropsize33;
else
    load('cam1_4.mat');
    load('cam2_4.mat');
    load('cam3_4.mat');
    v1 = vidFrames1_4; v2 = vidFrames2_4; v3 = vidFrames3_4;
    clear vidFrames1_4 vidFrames2_4 vidFrames3_4;
    cropsize1 = cropsize41; cropsize2 = cropsize42; cropsize3 = cropsize43;
end
```

2. *Analysis on frame data*

```
%% Analysis on frame data

% number of frames
nf1 = size(v1,4);

% crop video and convert to RGB
for j=nf1:-1:1
    cropped(:,:,:,j) = imcrop(v1(:,:,:,j), cropsize1);
    grayscale(:,:,:,j) = rgb2gray(cropped(:,:,:,j));
end
Rave1 = zeros(nf1); Cave1 = zeros(nf1);

% track the can
for j = 1:nf1
    j_frame = double(grayscale(:,:,j));
    [row_1,col_1] = find(j_frame >= 245);
    Rave1(j)=mean(row_1);
    Cave1(j)=mean(col_1);
end
clear cropped grayscale;

% number of frames
nf2 = size(v2,4);
for j=nf2:-1:1
    cropped(:,:,:,j) = imcrop(v2(:,:,:,j), cropsize2);
    grayscale(:,:,:,j) = rgb2gray(cropped(:,:,:,j));
end
Rave2 = zeros(nf2); Cave2 = zeros(nf2);

% track the can
for j = 1:nf2
    j_frame = double(grayscale(:,:,j));
    [row_2,col_2] = find(j_frame >= 245);
    Rave2(j)=mean(row_2);
    Cave2(j)=mean(col_2);
end
clear cropped grayscale;

% number of frames
nf3 = size(v3,4);
for j=nf3:-1:1
    rot_vid(:,:,:,j) = imrotate(v3(:,:,:,j), -90);
    cropped(:,:,:,j) = imcrop(rot_vid(:,:,:,j), cropsize3);
    grayscale(:,:,:,j) = rgb2gray(cropped(:,:,:,j));
end
Rave3 = zeros(nf3); Cave3 = zeros(nf3);

% track the can
for j = 1:nf3
    j_frame = double(grayscale(:,:,j));
    [row_3,col_3] = find(j_frame >= 245);
    Rave3(j)=mean(row_3);
    Cave3(j)=mean(col_3);
end
clear cropped grayscale;
```

3. *Plot the tracked positions*

```
%% plot positions

axi = [0 200 0 300];

figure(1);
subplot(3,1,1)
min_frames1=min(size(Rave1,2));
t1=1:min_frames1;
plot(t1, Rave1(1:min_frames1), 'k', t1 ,Cave1(1:min_frames1), 'k--')
axis(axi);
title('Position vs. Time Case 4', 'FontSize', 18)
legend('Y','X');

subplot(3,1,2)
min_frames2=min(size(Rave2,2));
t2=1:min_frames2;
plot(t2,Rave2(1:min_frames2),'k',t2,Cave2(1:min_frames2),'k--')
axis(axi);
ylabel('Position', 'FontSize', 14)

subplot(3,1,3)
min_frames3=min(size(Rave3,2));
t3=1:min_frames3;
plot(t3,Rave3(1:min_frames3),'k',t3,Cave3(1:min_frames3),'k--')
axis(axi);
xlabel('Frame', 'FontSize', 14)
```

4. *SVD*

```
%% SVD
data=[Rave1(1:200); Cave1(1:200); Rave2(1:200);
Cave2(1:200); Rave3(1:200); Cave3(1:200)];


% handle NaN values
data(isnan(data))=0;
[u, s, v] = svd(data, 'econ');
sigma = diag(s);
energy = sigma/sum(sigma);
```

5. *Plot singular values*

```
%% plot singular values
figure(2)
subplot(2,1,1)
plot(energy, 'ko','LineWidth', [1.4])
title('Singular Values Case 4', 'FontSize', 18)
ylabel('Energy (%)', 'FontSize', 14)
subplot(2,1,2)

semilogy(energy, 'ko','LineWidth', [1.4])
ylabel('Energy (log)', 'FontSize', 14)
xlabel('Singular Values', 'FontSize', 14)
```