PCA and a Spring-Mass System
AMATH 482 HW3

Xuanhui Chen

Abstract

This report aims to analyze four cases of three movie clips taken by three different cameras using principle component analysis (PCA). The experiments will develop the way of using PCA in practical problems from different aspects. The four cases are: the ideal case, the noisy case, the case involving horizontal displacement, and the case involving horizontal displacement as well as rotation, respectively. The latter three cases will be compared to the original case (ideal case).

I. Introduction and Overview

We are given four sets of three movie files of spring-mass oscillation for each of the case we want to analyze. Each movie file contained in different sets are recorded by three different cameras. We will discuss four situations adjusted with noise and rotation. The first one is the ideal case, which the mass has a small displacement in the z direction and the ensuing oscillations. This case only includes simple harmonic motion in the z direction. The second one is the noisy case, which includes camera shake that fluctuate the simple harmonic motion in addition to the ideal case. The PCA algorithms still work if the shake is acceptable. For the next two cases, the mass is released off-center. The difference between these two cases is: one produce motion horizontally in the x-y plane as well as the z-direction, the other one adds rotation to the horizontal movement along the z-direction.

II. Theoretical Background

Singular Value Decomposition(SVD)

The Singular Value Decomposition is in the form of:

$$\mathbf{A} = \mathbf{U\Sigma V}^*, \quad (1)$$

where $\mathbf{U}$ and $\mathbf{V}$ are unitary matrices, and $\mathbf{\Sigma}$ is a diagonal matrix with σ_j on its diagonals.

One important property of SVD is: Every matrix $\mathbf{A} \in \mathbb{C}^{m\times n}$ has an SVD.

Computing the SVD

We can compute the SVD in the following way:

$$\begin{aligned}\mathbf{A}^*\mathbf{A} &= (\mathbf{U\Sigma V}^*)^*(\mathbf{U\Sigma V}^*)\\ &= \mathbf{V\Sigma U}^*\mathbf{U\Sigma V}^*\\ &= \mathbf{V\Sigma^2 V}^*. \quad (2)\end{aligned}$$

Multiply both sides by $\mathbf{V}$ gives the following equation:

$$\mathbf{A}^*\mathbf{AV} = \mathbf{V\Sigma^2}, \quad (3)$$

where the columns of $\mathbf{V}$ are eigenvectors of $\mathbf{A^*A}$, the eigenvectors are given by the square of the singular values, and it can be used to solve $\mathbf{V}$.
Similarly, we can compute the SVD and obtain the following equations:

$$\mathbf{A^*A = U\Sigma^2U^*}, \qquad (4)$$

$$\mathbf{A^*AU = U\Sigma^2}. \qquad (5)$$

where it can be used to solve $\mathbf{U}$.

Principle Component Analysis (PCA)

The Principle Component Analysis (PCA) is a method that analyzes the low-rank approximations geometrically, while the SVD can be used to produce low-rank approximations of a dataset. Here we can see the connections between these two. The covariance of a specific matrix $\mathbf{X}$ can be obtained as:

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

where $\mathbf{U}$ is the orthogonal matrix of left-singular vectors, and $\mathbf{\Sigma}$ is the diagonal matrix of singular values. And we can also obtain the data in the new coordinates:

$$\mathbf{Y = U^TX},$$

where the covariance of $\mathbf{Y}$ is:

$$\mathbf{C_Y} = \frac{1}{\text{n-1}}\mathbf{YY^T} = \mathbf{\Sigma^2},$$

and the variables in $\mathbf{Y}$ are uncorrelated.

III. Algorithm Implementation and Development

We will repeat the following processes for all cases with details changing:

- Loading the given movie files and forming the initial conditions:
    - The video file is saved to a 4-D tensor, and we can obtain the number of frames by find the size of the 4-D tensor. (See **Appendix B** Test 1.)
- Introduce the filter to isolate the irrelevant portions and extract the motion:
    - For each camera, we set up a filter as a matrix of zeroes and ones and filter out the irrelevant pixels by setting them to zero. In this way, we can find out where a movement occurs. To extract the motion of the object we are focusing on, we search for the maximum value of the data. Built-in functions *rgb2gray()* is used to convert the RBG file to grayscale. (See **Appendix A** and **Appendix B** Test 1-4 camera part.)
- Trim the videos to produce the same number of time frames:
    - We trim the longer videos to the same length as the shortest one, and combine the information of these three videos. (See **Appendix B** Test 1-4 "Organize the data into 1 matrix".)
- Obtain the results and make the graphs:

- Perform the SVD to obtain the **U**, **V**, and **Σ** matrices. (See **Appendix B** Test Results.)
- Make the plots of variance vs. diagonal energy, and the displacement before and after the PCA. (See **Appendix B** Test 1-4 Graph Parts.)

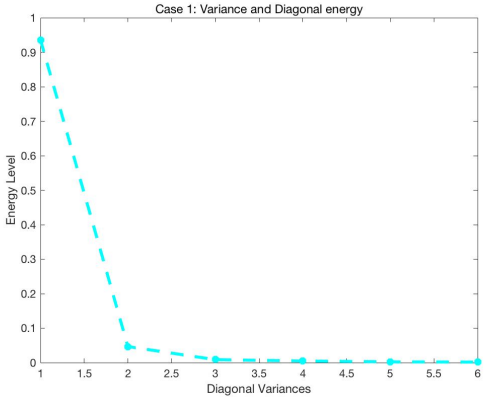
IV. Computational Results

4.1 Test 1: Ideal Case

As we can see in the Figure 1 on the left side, we only had one principle component corresponding to an approximate of above 90% energy capturing. And then the energy drastically decreases to a very low level after that and never bounces back. In this way, the mass oscillates in one direction, which needs only one principle component as expected. The Figure 2 shown on the right side tells that our transformation of the principle component results in accurate oscillation. The two graphs represent similar results, indicating that the motion is accurately reproduced with only one principle component.

4.2 Test 2: Noisy Case

In this case, we can see two principle components as there are two significant energy level points shown in Figure 3. The first principle component corresponds to an approximate of above 60% energy capturing, which the second one corresponds to energy close to 20%. So the noise generated by shaking has affected and weakened the PCA calculations. In addition, we can still see clear oscillations even when the noise is involved in Figure 4. And the two graphs also represent similar results, meaning we can still use two principle components to reproduce the motion accurately.



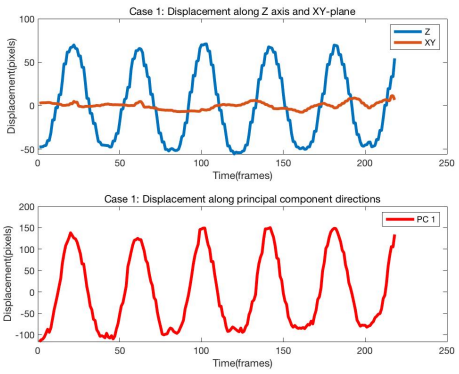
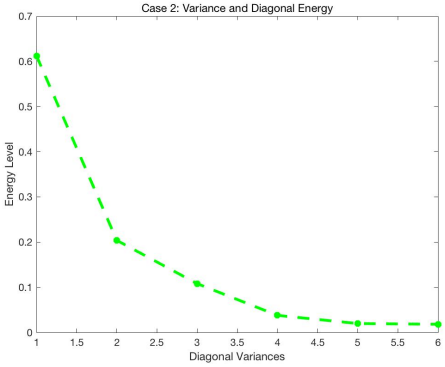


Figure 1(left): Variances vs. Energy Level for the first case

Figure 2(right): Comparison of the displacement track before and after the PCA for the first case.



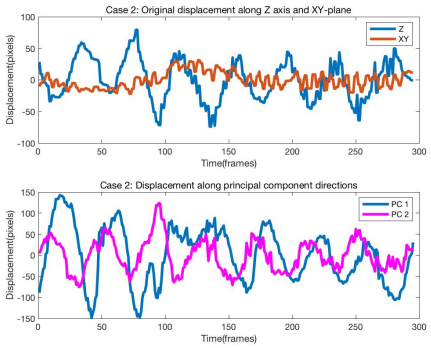


Figure 3 (left): Variances vs. Energy Level for the second case

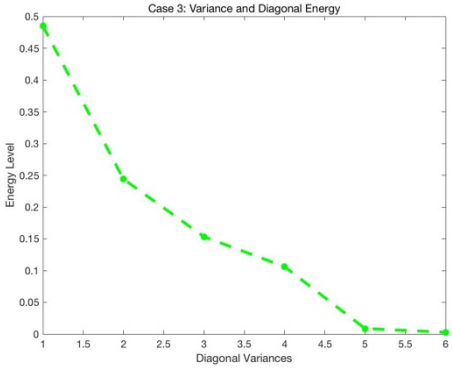
Figure 4 (right): Comparison of the displacement track before and after the PCA for the second case.

4.3 Test 3: Horizontal Displacement:

In this case, we can see four principle components that can capture significant amount of energy shown in Figure 5, and they are quite close to each other. In Figure 6, we can see significant oscillations.

4.4 Test 4: Horizontal Displacement and Rotation:

In Figure 7, we can see three significant principle components of energy capturing. In Figure 8, we can see pendulum nature as well as simple harmonic motion, indicating that the PCA captures the multi-dimensional nature of the horizontal displacement and rotation.



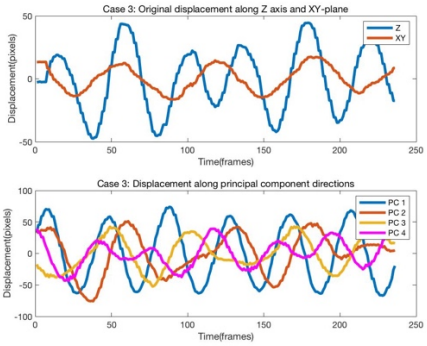
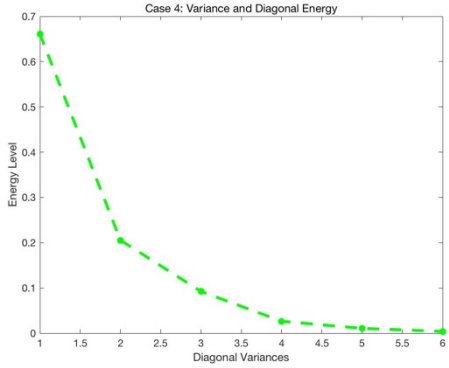


Figure 5 (left): Variances vs. Energy Level for the third case.

Figure 6 (right): Comparison of the displacement track before and after the PCA for the third case.



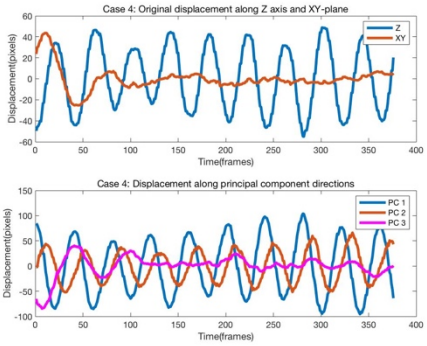


Figure 7 (left): Variances vs. Energy Level for the fourth case.

Figure 8 (right): Comparison of the displacement track before and after the PCA for the fourth case.

V. Summary and Conclusion

In conclusion, the principle component analysis is helpful for analyzing the practical problems without actually involves the physical analysis. We have pretty accurate projections overall, and we can find the number of principle components existed. With the PCA, we are able to remove the irrelevant information and recover the motion of the system.

Appendix A: MATLAB functions used and brief implementation explanation

- **diag(v)**: Create diagonal matrix or get diagonal elements of matrix
- **rgb2gray**(RGB): Convert RGB image or colormap to grayscale
- **size(A):** returns a row vector whose elements contain the length of the corresponding dimension of A.
- **mean():** returns the mean of the elements of A along the first array dimension whose size does not equal 1.

Appendix A: MATLAB Codes


```
% Test 1
clear all; close all; clc;

load('cam1_1.mat')
load('cam2_1.mat')
load('cam3_1.mat')

frame1 = size(vidFrames1_1,4);
frame2 = size(vidFrames2_1,4);
frame3 = size(vidFrames3_1,4);

%for j = 1:frame1
    %X = vidFrames1_1(:,:,:,j);
    %imshow(X); drawnow
%end

%% Test 1: Camera 1
filter = zeros(480, 640);
filter(170:430, 300:400) = 1;

data1 = zeros(frame1, 2);
for j = 1:frame1
    X = vidFrames1_1(:,:,:,j);
    Xg = double(rgb2gray(X));
    Xf = Xg.*filter;
    thres = Xf > 250;

    [Y, X] = find(thres);
    data1(j,1) = mean(X);
    data1(j,2) = mean(Y);
end

%% Test 1: Camera 2
filter = zeros(480, 640);
filter(100:400, 230:360) = 1;

data2 = zeros(frame2, 2);
for j = 1:frame2
    X = vidFrames2_1(:,:,:,j);
    Xg = double(rgb2gray(X));
    Xf = Xg.*filter;
    thres = Xf > 250;

    [Y, X] = find(thres);
    data2(j,1) = mean(X);
    data2(j,2) = mean(Y);
end

%% Test 1: Camera 3
filter = zeros(480, 640);
filter(220:350, 250:500) = 1;

data3 = zeros(frame3, 2);
for j = 1:frame3
    X = vidFrames3_1(:,:,:,j);
    Xg = double(rgb2gray(X));
    Xf = Xg.*filter;
    thres = Xf > 245;
```

```
    [Y, X] = find(thres);
    data3(j,1) = mean(X);
    data3(j,2) = mean(Y);
end

%% Organize the data into 1 matrix
[M,I] = min(data1(1:20,2));
data1 = data1(I:end,:);
[M,I] = min(data2(1:20,2));
data2 = data2(I:end,:);
[M,I] = min(data3(1:20,2));
data3 = data3(I:end,:);

% data1 has the least values
data2 = data2(1:length(data1), :);
data3 = data3(1:length(data1), :);

data = [data1'; data2'; data3'];

%% Test1 Results
[m,n]= size(data);

data = data-repmat(mean(data,2),1,n);
[U,S,V]= svd(data'/sqrt(n-1));
lambda = diag(S).^2;
Y = data' * V;

% Test1: graphs
figure(1)
plot(1:6, lambda/sum(lambda), 'c*--', 'Linewidth', 3);
title("Case 1: Variance and Diagonal energy");
xlabel("Diagonal Variances"); ylabel("Energy Level");

figure(2)
subplot(2,1,1)
plot(1:length(data), data(2,:), 1:length(data), data(1,:), 'Linewidth', 3)
ylabel("Displacement(pixels)"); xlabel("Time(frames)");
title("Case 1: Displacement along Z axis and XY-plane");
legend("Z", "XY")
subplot(2,1,2)
plot(1:length(data), Y(:,1),'r','Linewidth', 3)
ylabel("Displacement(pixels)"); xlabel("Time(frames)");
title("Case 1: Displacement along principal component directions");
legend("PC 1")

%% Test 2
clear all; close all; clc;

load('cam1_2.mat')
load('cam2_2.mat')
load('cam3_2.mat')

frame1 = size(vidFrames1_2,4);
frame2 = size(vidFrames2_2,4);
frame3 = size(vidFrames3_2,4);

%for j = 1:frame1
```

```
    %X = vidFrames1_1(:,:,:,j);
    %imshow(X); drawnow
%end

%% Test 2: Camera 1
filter = zeros(480, 640);
filter(170:430, 300:450) = 1;

data1 = zeros(frame1, 2);
for j = 1:frame1
    X = vidFrames1_2(:,:,:,j);
    Xg = double(rgb2gray(X));
    Xf = Xg.*filter;
    thres = Xf > 250;

    [Y, X] = find(thres);
    data1(j,1) = mean(X);
    data1(j,2) = mean(Y);
end

%% Test 2: Camera 2
filter = zeros(480, 640);
filter(50:470, 170:440) = 1;

data2 = zeros(frame2, 2);
for j = 1:frame2
    X = vidFrames2_2(:,:,:,j);
    Xg = double(rgb2gray(X));
    Xf = Xg.*filter;
    thres = Xf > 248;

    [Y, X] = find(thres);
    data2(j,1) = mean(X);
    data2(j,2) = mean(Y);
end

%% Test 2: Camera 3
filter = zeros(480, 640);
filter(150:350, 250:500) = 1;

data3 = zeros(frame3, 2);
for j = 1:frame3
    X = vidFrames3_2(:,:,:,j);
    Xg = double(rgb2gray(X));
    Xf = Xg.*filter;
    thres = Xf > 245;

    [Y, X] = find(thres);
    data3(j,1) = mean(X);
    data3(j,2) = mean(Y);
end

%% Organize the data into 1 matrix
[M,I] = min(data1(1:20,2));
data1 = data1(I:end,:);
[M,I] = min(data2(1:20,2));
data2 = data2(I:end,:);
[M,I] = min(data3(1:20,2));
```

```
data3 = data3(I:end,:);

% data1 has the least values
data2 = data2(1:length(data1), :);
data3 = data3(1:length(data1), :);

data = [data1'; data2'; data3'];

%% Test2 Results
[m,n]= size(data);

data = data-repmat(mean(data,2),1,n);
[U,S,V]= svd(data'/sqrt(n-1));
lambda = diag(S).^2;
Y = data' * V;

% Test2: graphs
figure(3)
plot(1:6, lambda/sum(lambda), 'g*--', 'Linewidth', 3);
title("Case 2: Variance and Diagonal Energy");
xlabel("Diagonal Variances"); ylabel("Energy Level");

figure(4)
subplot(2,1,1)
plot(1:length(data), data(2,:), 1:length(data), data(1,:), 'Linewidth', 3)
ylabel("Displacement(pixels)"); xlabel("Time(frames)");
title("Case 2: Original displacement along Z axis and XY-plane");
legend("Z", "XY")
subplot(2,1,2)
plot(1:length(data), Y(:,1), 1:length(data), Y(:,2),'m','Linewidth', 3)
ylabel("Displacement(pixels)"); xlabel("Time(frames)");
title("Case 2: Displacement along principal component directions");
legend("PC 1", "PC 2")

%% Test 3
clear all; close all; clc;

load('cam1_3.mat')
load('cam2_3.mat')
load('cam3_3.mat')

frame1 = size(vidFrames1_3,4);
frame2 = size(vidFrames2_3,4);
frame3 = size(vidFrames3_3,4);

%for j = 1:frame1
    %X = vidFrames1_1(:,:,:,j);
    %imshow(X); drawnow
%end

%% Test 3: Camera 1
filter = zeros(480, 640);
filter(220:450, 270:400) = 1;

data1 = zeros(frame1, 2);
for j = 1:frame1
    X = vidFrames1_3(:,:,:,j);
    Xg = double(rgb2gray(X));
```

```
    Xf = Xg.*filter;
    thres = Xf > 250;

    [Y, X] = find(thres);
    data1(j,1) = mean(X);
    data1(j,2) = mean(Y);
end

%% Test 3: Camera 2
filter = zeros(480, 640);
filter(150:400, 240:420) = 1;

data2 = zeros(frame2, 2);
for j = 1:frame2
    X = vidFrames2_3(:,:,:,j);
    Xg = double(rgb2gray(X));
    Xf = Xg.*filter;
    thres = Xf > 248;

    [Y, X] = find(thres);
    data2(j,1) = mean(X);
    data2(j,2) = mean(Y);
end

%% Test 3: Camera 3
filter = zeros(480, 640);
filter(150:350, 250:500) = 1;

data3 = zeros(frame3, 2);
for j = 1:frame3
    X = vidFrames3_3(:,:,:,j);
    Xg = double(rgb2gray(X));
    Xf = Xg.*filter;
    thres = Xf > 245;

    [Y, X] = find(thres);
    data3(j,1) = mean(X);
    data3(j,2) = mean(Y);
end

%% Organize the data into 1 matrix
[M,I] = min(data1(1:20,2));
data1 = data1(I:end,:);
[M,I] = min(data2(1:20,2));
data2 = data2(I:end,:);
[M,I] = min(data3(1:20,2));
data3 = data3(I:end,:);

% data1 has the least values
data1 = data1(1:length(data3), :);
data2 = data2(1:length(data3), :);

data = [data1'; data2'; data3'];

%% Test3 Results
[m,n]= size(data);

data = data-repmat(mean(data,2),1,n);
```

```
[U,S,V]= svd(data'/sqrt(n-1));
lambda = diag(S).^2;
Y = data' * V;

% Test3: graphs
figure(5)
plot(1:6, lambda/sum(lambda), 'g*--', 'Linewidth', 3);
title("Case 3: Variance and Diagonal Energy");
xlabel("Diagonal Variances"); ylabel("Energy Level");

figure(6)
subplot(2,1,1)
plot(1:length(data), data(2,:), 1:length(data), data(1,:), 'Linewidth', 3)
ylabel("Displacement(pixels)"); xlabel("Time(frames)");
title("Case 3: Original displacement along Z axis and XY-plane");
legend("Z", "XY")
subplot(2,1,2)
plot(1:length(data), Y(:,1), 1:length(data), Y(:,2), 1:length(data), ...
    Y(:,3), 1:length(data), Y(:,4), 'm','Linewidth', 3)
ylabel("Displacement(pixels)"); xlabel("Time(frames)");
title("Case 3: Displacement along principal component directions");
legend("PC 1", "PC 2", "PC 3", "PC 4")

%% Test 4
clear all; close all; clc;

load('cam1_4.mat')
load('cam2_4.mat')
load('cam3_4.mat')

frame1 = size(vidFrames1_4,4);
frame2 = size(vidFrames2_4,4);
frame3 = size(vidFrames3_4,4);

%for j = 1:frame1
    %X = vidFrames1_1(:,:,:,j);
    %imshow(X); drawnow
%end

%% Test 4: Camera 1
filter = zeros(480, 640);
filter(200:450, 300:480) = 1;

data1 = zeros(frame1, 2);
for j = 1:frame1
    X = vidFrames1_4(:,:,:,j);
    Xg = double(rgb2gray(X));
    Xf = Xg.*filter;
    thres = Xf > 245;

    [Y, X] = find(thres);
    data1(j,1) = mean(X);
    data1(j,2) = mean(Y);
end

%% Test 4: Camera 2
filter = zeros(480, 640);
filter(100:400, 220:420) = 1;
```

```
data2 = zeros(frame2, 2);
for j = 1:frame2
    X = vidFrames2_4(:,:,:,j);
    Xg = double(rgb2gray(X));
    Xf = Xg.*filter;
    thres = Xf > 248;

    [Y, X] = find(thres);
    data2(j,1) = mean(X);
    data2(j,2) = mean(Y);
end

%% Test 4: Camera 3
filter = zeros(480, 640);
filter(130:300, 300:500) = 1;

data3 = zeros(frame3, 2);
for j = 1:frame3
    X = vidFrames3_4(:,:,:,j);
    Xg = double(rgb2gray(X));
    Xf = Xg.*filter;
    thres = Xf > 230;

    [Y, X] = find(thres);
    data3(j,1) = mean(X);
    data3(j,2) = mean(Y);
end

%% Organize the data into 1 matrix
[M,I] = min(data1(1:20,2));
data1 = data1(I:end,:);
[M,I] = min(data2(1:20,2));
data2 = data2(I:end,:);
[M,I] = min(data3(1:20,2));
data3 = data3(I:end,:);

% data1 has the least values
data1 = data1(1:length(data3), :);
data2 = data2(1:length(data3), :);

data = [data1'; data2'; data3'];

%% Test4 Results
[m,n]= size(data);

data = data-repmat(mean(data,2),1,n);
[U,S,V]= svd(data'/sqrt(n-1));
lambda = diag(S).^2;
Y = data' * V;

% Test4: graphs
figure(7)
plot(1:6, lambda/sum(lambda), 'g*--', 'Linewidth', 3);
title("Case 4: Variance and Diagonal Energy");
xlabel("Diagonal Variances"); ylabel("Energy Level");

figure(8)
```



```
subplot(2,1,1)
plot(1:length(data), data(2,:), 1:length(data), data(1,:), 'Linewidth', 3)
ylabel("Displacement(pixels)"); xlabel("Time(frames)");
title("Case 4: Original displacement along Z axis and XY-plane");
legend("Z", "XY")
subplot(2,1,2)
plot(1:length(data), Y(:,1), 1:length(data), Y(:,2), 1:length(data), ...
    Y(:,3), 'm','Linewidth', 3)
ylabel("Displacement(pixels)"); xlabel("Time(frames)");
title("Case 4: Displacement along principal component directions");
legend("PC 1", "PC 2", "PC 3")
```

Reference

1. Course note.
2. MATLAB help section.