AMATH482 Homework 3

Vandy Zhang
February 23, 2021

Abstract

This assignment is to compare four cases of the spring-mass system in the videos provided, which were created by three cameras from three different angles. The four cases are ideal case, noisy case, horizontal displacement case, and horizontal displacement and rotation case. To do the analysis, I applied the singular value decomposition, found the principal components, and produced four graphs for each case with the energy captured by the camera measurements and the projected data onto the largest principal components included.

1 Introduction and Overview

For this homework, I was asked to apply the principal component analysis to compare the four video clips shot by three cameras respectively from three angles. For the first case, which is the ideal case, it's simply a harmonic motion in the z direction. For the second case, which is the noisy case, camera shakes were introduced so that noise could be simulated and added to the system. For the third case, which is the horizontal displacement case, the mass was initially released off-center by the experimenter so that the horizontal displacement could produce more motion both in the z direction and the x - y plane. For the fourth case, which is the horizontal displacement and rotation case, the mass was not only released off-center but also rotated so that even more motion would be added to the system. To compare the four systems, I had to first extract the mass, then find the principal components of the positions, and finally do projections of the data based on the largest principal components, which captured most of the energy. By finding the bright spot on the mass in the spring-mass system, I located the mass. Then I used the singular value decomposition and applied the principal components analysis to compare the directions of the three cameras and the noise information of the four cases.

2 Theoretical Background

2.1 The Singular Value Decomposition

While multiplying a vector by a matrix, we have $\mathbf{Av} = \sigma\mathbf{u}$, in which σ represents a stretching vector and $\mathbf{u}$ represents a rotation vector. In other words, when we trying to change a vector, we change its direction by applying a rotation vector to it and change its length by applying a stretching vector to it. In matrix form, we have $\mathbf{AV} = \mathbf{U\Sigma}$, where $\mathbf{V}$ is a unitary matrix, $\mathbf{U}$ represents a rotation matrix, and $\mathbf{\Sigma}$ represents a stretching matrix with σ on its diagonals representing singular values. For any unitary matrices, we have $\mathbf{V^{-1}} = \mathbf{V^*}$, where $\mathbf{V^*}$ means the transpose of V. Therefore, we can isolate A to get:

$$\mathrm{A} = \mathrm{U\Sigma V^*},$$

thus, giving us a decomposition of the matrix A through unitary matrices $\mathbf{U}$ and $\mathbf{V}$ and a diagonal matrix $\mathbf{\Sigma}$. [1]

In MATLAB, there is a built-in svd(X, 'econ') function that we can use to apply the reduced singular value decomposition, which is a more efficient way that has a lower order.

2.2 The Principal Components

Suppose we have

$$\mathrm{X} = \begin{bmatrix} \mathrm{a} \\ b \\ c \end{bmatrix}',$$

we can compute the covariance of the rows of X so we can get this

$$\mathrm{C_x} = \frac{1}{n-1} XX^T = \begin{bmatrix} \sigma_a^2 & \sigma_{ab}^2 & \sigma_{ac}^2 \\ \sigma_{ba}^2 & \sigma_b^2 & \sigma_{bc}^2 \\ \sigma_{ca}^2 & \sigma_{cb}^2 & \sigma_c^2 \end{bmatrix}.$$

$\mathbf{C_x}$ is a square symmetric matrix, which is called covariance matrix. Since the variables are uncorrelated, we have new information contained in each variable. The largest variable represents the direction of the largest variance of the data. Therefore, we should diagonalize the matrix to get the information:

$$\mathrm{C_x} = \mathrm{V\Lambda V^{-1}},$$

Since $\mathbf{C_x}$ is a square symmetric matrix, its eigenvalues are real and the corresponding eigenvectors are orthogonal and uncorrelated. The values of the diagonal of $\mathbf{\Lambda}$ are the eigenvalues of the $\mathbf{C_x}$ representing the variances. [2]

The eigenvalues of the covariance matrix are the squares of the singular values we calculated in 2.1.

2.3 The Spring-mass System

If we want to understand the spring-mass system from a mathematical aspect rather than in a physical way, we can use the singular value decomposition to analyze the horizontal and vertical displacement of the mass. We set up three cameras around the system and collect videos with two dimensions in each frame. We can combine the dimensions of the positions given by each camera into one matrix, and we will get [2]:

$$\mathrm{X} = \begin{bmatrix} X_a \\ Y_a \\ X_b \\ Y_b \\ X_c \\ Y_c \end{bmatrix}.$$

In our spring-mass system, since the motion should be in 1D (in the z direction), there should only be one nonzero element among the singular values (principal components). However, things aren't perfect in reality. There might be some noise in camera recordings and some movements in other directions. Therefore, after we implement the singular value decomposition, we can take the relatively large values are important ones and ignore the ones that are really small. Based on the important ones, especially the one that is significantly larger than others and having a dominance of the up-down motion of the mass, we can infer the directions that are the most important, simulate the motion of the mass, and decide whether or not we should rotate the angles of the cameras. Once we know the most important directions, we can project the data onto the direction by multiplying the data by U^{T}, since we know that $\mathrm{A} = \mathrm{U}\Sigma\mathrm{V}^{*}$, which means that $\mathrm{U}^{\mathrm{T}}\mathrm{A} = \Sigma\mathrm{V}^{*}$.

3 Algorithm Implementation and Development

To load the data provided by the three cameras, I used **load(file.mat)** and got 12 video frames in the type of unit 8. In order to track the spring and extract the location data of the mass, I created a function called **findposition**. I looped through each frame and find a range of brightest spots by implementing **find(X > 220),** since with the unit 8 data type, the range of a number is from 0 to 255. Then I got the location of the mass by calculating the mean of the indices of the brightest spots returned by the find function. Once I got my

location data given by all three cameras, I combined them in a positions matrix, and then I tried to apply the principal component analysis.

To apply the analysis, I created another function called **pcanalysis**. To do the singular value decomposition on my positions matrix, I subtracted the **mean** of each row of my positions matrix by applying the mean and the **repmat** function so that I had my positions matrix symmetric around zero. Then I divided my positions matrix by the square root of n – 1, in which minus 1 was to make the data unbiased in statistics. After applying the **svd** function to my new positions data, I squared the singular values I got to get the principal components according to the theorem that the eigenvalues are the squares of the SVs.

After computing the information above, I turned to plot the data I derived. To find the energy captured by each eigenvalue, I calculated their percentage by dividing each element by their sum. To find the projection of the data onto the directions given by the largest principal components, I multiplied the original data by the transpose of U. I plotted four graphs, which are the principal components of each measurement type, the energy captured by each principal component, the original data across the z direction and the plane XY, and the projections of the data onto the directions that have the most variances. For the other cases, I repeated what I did above.

4 Computational Results

4.1 Test 1: Ideal Case

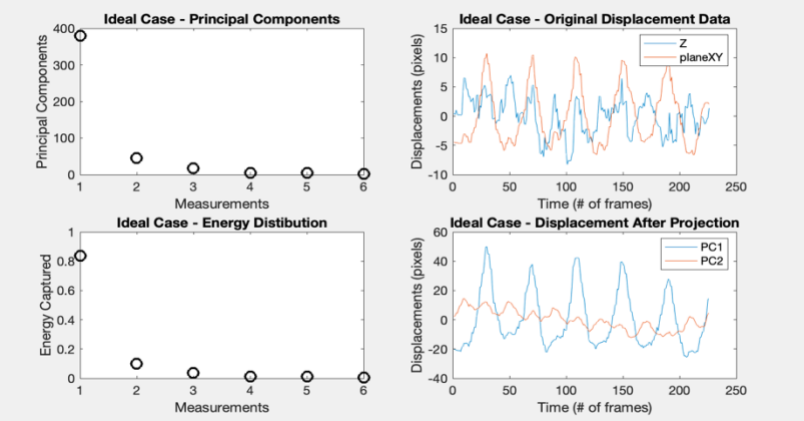

Figure 1. (test 1) (top left) plot of the principal components' distribution; (bottom left) plot of the energy captured by the components; (top right) plot of the original displacement; (bottom right) plot of the displacement over new basis

As seen in figure 1, the first two principal components represent the most important variances, with component one dominants over 80% of the energy and little noise is recognized. In this case, the projection onto the first component are highly similar to the original data.

4.2 Test 2: Noisy Case

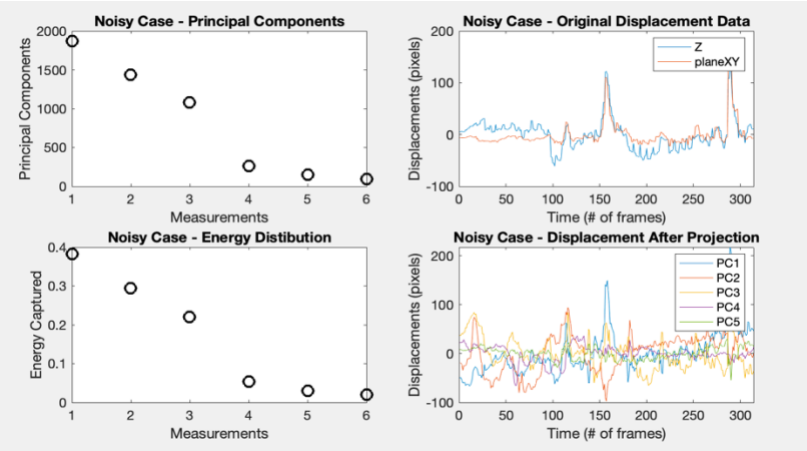

Figure 2. (test 2) (top left) plot of the principal components' distribution; (bottom left) plot of the energy captured by the components; (top right) plot of the original displacement; (bottom right) plot of the displacement over new basis

As seen in figure 2, even though the first three principal components capture most of the energy, all six components have some energy, which means that there is so much noise. This situation is also shown in the projections, but there is still oscillatory behavior.

4.3 Test 3: Horizontal Displacement Case

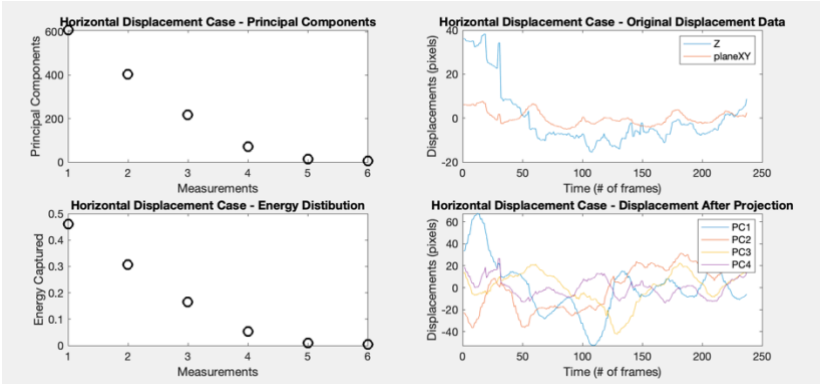

Figure 3. (test 3) (top left) plot of the principal components' distribution; (bottom left) plot of the energy captured by the components; (top right) plot of the original displacement; (bottom right) plot of the displacement over new basis

As seen in figure 3, the first four principal components represent the most important variances. From the projection graph at the bottom right corner, we notice that there are some big differences in displacements projected to the first principal component, because in this case, our mass is released off-center.

4.4 Test 4: Horizontal Displacement & Rotation Case

As seen in figure 4, the first two principal components represent the most important variances. It seems that the principal components also capture the nature of multidimensional displacement since they oscillate at different peaks.

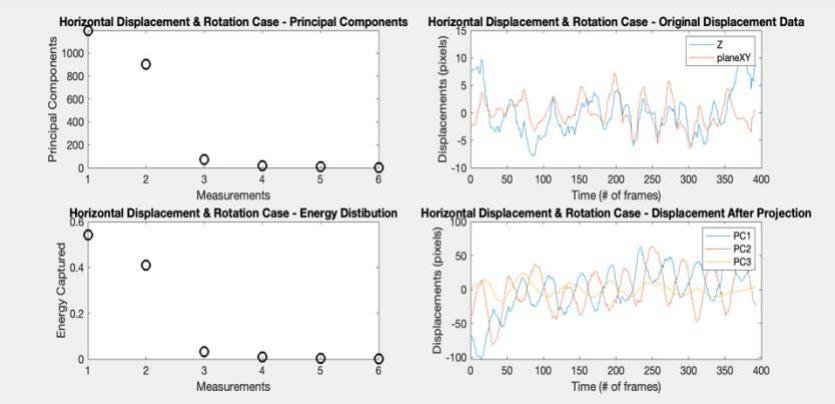

Figure 3. (test 3) (top left) plot of the principal components' distribution; (bottom left) plot of the energy captured by the components; (top right) plot of the original displacement; (bottom right) plot of the displacement over new basis

5 Summary and Conclusions

By applying the singular value decomposition and principal component analysis, I was able to analyze the position data of the spring-mass system in video frames. By comparing the principal components I derived in each case, I could recognize how the oscillatory behavior of the mass is and how it was released.

References

[1] University of Washington. AMATH482 Lecture 11 Notes. URL: https://canvas.uw.edu/courses/1433289/files/71735394?wrap=1
[2] University of Washington. AMATH482 Lecture 13 Notes. URL: https://canvas.uw.edu/courses/1433289/files/71773009?wrap=1
[3] Jose Nathan Kutz. Data-driven modeling & scientific computation: methods for complex systems & bigdata. Oxford University Press, 2013.

Appendix A. MATLAB Functions

load(filename)	loads data from *filename*
mean(A, dim)	computes the mean of elements along the dimension dim
[row, col] = find()	returns the subscripts of the elements satisfying the condition specified in find
[U, S, V] = svd(A, 'econ')	performs the singular value decomposition and produces an economy-size decomposition of matrix A
repmat(A, n)	repeat n copies of the array
plot(X1, Y1, X2, Y2, ···, Xn, Yn)	plots multiple X, Y pairs using the same axes for all lines
legend(labels)	set legend labels
set	Set properties for the graph

Appendix B. MATLAB Code

```
close all; clc; clear;
load('cam1_1.mat'), load('cam1_2.mat'), load('cam1_3.mat'),
load('cam1_4.mat'),
load('cam2_1.mat'), load('cam2_2.mat'), load('cam2_3.mat'),
load('cam2_4.mat'),
load('cam3_1.mat'), load('cam3_2.mat'), load('cam3_3.mat'),
load('cam3_4.mat')

[U1, lambda1, positions1, numFrames1] = pcanalysis(vidFrames1_1,
vidFrames2_1, vidFrames3_1);
[U2, lambda2, positions2, numFrames2] = pcanalysis(vidFrames1_2,
vidFrames2_2, vidFrames3_2);
[U3, lambda3, positions3, numFrames3] = pcanalysis(vidFrames1_3,
vidFrames2_3, vidFrames3_3);
[U4, lambda4, positions4, numFrames4] = pcanalysis(vidFrames1_4,
vidFrames2_4, vidFrames3_4);

% plot principal components
figure(1)
subplot(2, 2, 1)
plot(1:6, lambda1, 'ko', 'MarkerSize', 10, 'Linewidth', 2)
xlabel('Measurements'); ylabel('Principal Components');
title('Ideal Case - Principal Components');
set(gca,'xlim', [1 6], 'Fontsize',12)
subplot(2, 2, 2)
plot(1:numFrames1, positions1(2, :), 1:numFrames1, positions1(1,:));
legend('Z', 'planeXY')
xlabel('Time (# of frames)'); ylabel('Displacements (pixels)');
title('Ideal Case - Original Displacement Data');
set(gca,'Fontsize',12)
subplot(2, 2, 3)
plot(1:6, lambda1/sum(lambda1), 'ko', 'MarkerSize', 10, 'Linewidth', 2)
xlabel('Measurements'); ylabel('Energy Captured');
title('Ideal Case - Energy Distibution');
set(gca,'xlim', [1 6],'Fontsize',12)
subplot(2, 2, 4)
projections1 = U1' * positions1;
plot(1:numFrames1, projections1(1, :), 1:numFrames1, projections1(2, :));
legend('PC1', 'PC2');
xlabel('Time (# of frames)'); ylabel('Displacements (pixels)');
title('Ideal Case - Displacement After Projection');
set(gca,'Fontsize',12)
print('HW3IdealCase.png', '-dpng');

figure(2)
subplot(2, 2, 1)
plot(1:6, lambda2, 'ko', 'MarkerSize', 10, 'Linewidth', 2)
xlabel('Measurements'); ylabel('Principal Components');
title('Noisy Case - Principal Components');
set(gca,'xlim', [1 6],'Fontsize',12)
subplot(2, 2, 2)
plot(1:numFrames2, positions2(2, :), 1:numFrames2, positions2(1,:));
```

```
legend('Z', 'planeXY')
xlabel('Time (# of frames)'); ylabel('Displacements (pixels)');
title('Noisy Case - Original Displacement Data');
set(gca,'Fontsize',12)
subplot(2, 2, 3)
plot(1:6, lambda2/sum(lambda2), 'ko', 'MarkerSize', 10, 'Linewidth', 2)
xlabel('Measurements'); ylabel('Energy Captured');
title('Noisy Case - Energy Distibution');
set(gca,'xlim', [1 6],'Fontsize',12)
subplot(2, 2, 4)
projections2 = U2' * positions2;
plot(1:numFrames2, projections2(1, :), 1:numFrames2, projections2(2, :),
1:numFrames2, projections2(3, :), 1:numFrames2, projections2(4, :),
1:numFrames2, projections2(5, :));
legend('PC1', 'PC2', 'PC3', 'PC4', 'PC5');
xlabel('Time (# of frames)'); ylabel('Displacements (pixels)');
title('Noisy Case - Displacement After Projection');
set(gca,'Fontsize',12)
print('HW3NoisyCase.png', '-dpng');

figure(3)
subplot(2, 2, 1)
plot(1:6, lambda3, 'ko', 'MarkerSize', 10, 'Linewidth', 2)
xlabel('Measurements'); ylabel('Principal Components');
title('Horizontal Displacement Case - Principal Components');
set(gca,'xlim', [1 6],'Fontsize',12)
subplot(2, 2, 2)
plot(1:numFrames3, positions3(2, :), 1:numFrames3, positions3(1,:));
legend('Z', 'planeXY')
xlabel('Time (# of frames)'); ylabel('Displacements (pixels)');
title('Horizontal Displacement Case - Original Displacement Data');
set(gca,'Fontsize',12)
subplot(2, 2, 3)
plot(1:6, lambda3/sum(lambda3), 'ko', 'MarkerSize', 10, 'Linewidth', 2)
xlabel('Measurements'); ylabel('Energy Captured');
title('Horizontal Displacement Case - Energy Distibution');
set(gca,'xlim', [1 6],'Fontsize',12)
subplot(2, 2, 4)
projections3 = U3' * positions3;
plot(1:numFrames3, projections3(1, :), 1:numFrames3, projections3(2, :),
1:numFrames3, projections3(3, :), 1:numFrames3, projections3(4, :));
legend('PC1', 'PC2', 'PC3', 'PC4');
xlabel('Time (# of frames)'); ylabel('Displacements (pixels)');
title('Horizontal Displacement Case - Displacement After Projection');
set(gca,'Fontsize',12)
print('HW3HorizontalCase.png', '-dpng');

figure(4)
subplot(2, 2, 1)
plot(1:6, lambda4, 'ko', 'MarkerSize', 10, 'Linewidth', 2)
xlabel('Measurements'); ylabel('Principal Components');
title('Horizontal Displacement & Rotation Case - Principal Components');
set(gca,'xlim', [1 6],'Fontsize',12)
subplot(2, 2, 2)
plot(1:numFrames4, positions4(2, :), 1:numFrames4, positions4(1,:));
legend('Z', 'planeXY')
```

```
xlabel('Time (# of frames)'); ylabel('Displacements (pixels)');
title('Horizontal Displacement & Rotation Case - Original Displacement
Data');
set(gca,'Fontsize',12)
subplot(2, 2, 3)
plot(1:6, lambda4/sum(lambda4), 'ko', 'MarkerSize', 10, 'Linewidth', 2)
xlabel('Measurements'); ylabel('Energy Captured');
title('Horizontal Displacement & Rotation Case - Energy Distibution');
set(gca,'xlim', [1 6],'Fontsize',12)
subplot(2, 2, 4)
projections4 = U4' * positions4;
plot(1:numFrames4, projections4(1, :), 1:numFrames4, projections4(2, :),
1:numFrames4, projections4(3, :));
legend('PC1', 'PC2', 'PC3');
xlabel('Time (# of frames)'); ylabel('Displacements (pixels)');
title('Horizontal Displacement & Rotation Case - Displacement After
Projection');
set(gca,'Fontsize',12)
print('HW3HorizontalRotationCase.png', '-dpng');


function [u, lambda, positions, numFrames] = pcanalysis(vidFrames1,
vidFrames2, vidFrames3)
    numFrames1 = size(vidFrames1,4);
    numFrames2 = size(vidFrames2,4);
    numFrames3 = size(vidFrames3,4);
    numFrames = min([numFrames1, numFrames2, numFrames3]);
    positions = zeros(6, numFrames);

    [xvector1, yvector1] = findposition(vidFrames1, numFrames);
    positions(1, :) = xvector1;
    positions(2, :) = yvector1;
    [xvector2, yvector2] = findposition(vidFrames2, numFrames);
    positions(3, :) = xvector2;
    positions(4, :) = yvector2;
    [xvector3, yvector3] = findposition(vidFrames3, numFrames);
    positions(5, :) = xvector3;
    positions(6, :) = yvector3;

    positions = positions - repmat(mean(positions, 2), 1, numFrames);
    [u, s, ~] = svd(positions/sqrt(numFrames-1));
    lambda = diag(s).^2;
end

function [xvector, yvector] = findposition (vidFrame, numFrames)
    xvector = zeros(1, numFrames);
    yvector = zeros(1, numFrames);
    for j = 1:numFrames
        X = vidFrame(:,:,:,j);
        X = rgb2gray(X);
        [place1, place2] = find(X > 220);
        xvector(j) = mean(place1);
        yvector(j) = mean(place2);
    end
end
```