AMATH 482 HW3 - Principal Component Analysis

Winnie Shao

2/24/2021

Abstract

In this report, we introduce the Principal Component Analysis and practice it on four cases of a spring-mass system. We will examine the diagonal variance content of each principal components and projections on their basis. Also, we are going to compare them with the original oscillation behavior of the system.

Section I: Introduction and Overview

Linear algebra is an important role in many application of math, physics, engineering, and also data analysis. In this report, we focus on a spring-mass example and we want to learn how we can apply some concepts to this analysis.

In the spring-mass example, a bucket is suspended with a spring and oscillates. A gray light was attached at the top of the bucket to help us track the bucket. There are three video cameras for collecting data about the mass's motion in order to ascertain its governing equations. We are given datasets produced by these three cameras in four cases. We are going to apply the PCA method on our spring-mass exmaple and explore the effects of noise on the algorithm.

Section II: Theoretical Backgroud

2.1 Eigenvalues & Eigenvectors

Let A be an n $\times$ n matrix, v is an n $\times$ 1 vector, and λ is a scalar. When we get the following equation, we can say that λ is the eigenvalue and v is the eigenvector of matrix A.

$$Av = \lambda v \tag{1}$$

By definition, an eigenvector of a linear transformation is a nonzero vector that changes at most by a scalar factor, eigenvalue λ , when linear transformation is applied to it.

2.2 Singular Value Decomposition

Singular value decomposition (SVD) is a factorization of a matrix into several specific components, as the following form:

$$A = U\Sigma V^* \tag{2}$$

with three components:

$$U \in \mathbf{C}^{m\times m} \text{ is unitary}$$
$$V \in \mathbf{C}^{n\times n} \text{ is unitary}$$
$$\Sigma \in \mathbf{R}^{m\times n} \text{ is diagonal}$$

The diagonal entries of Σ are nonnegative and ordered from largest to smallest so that $\sigma_1 \geq \sigma_2 \geq ... \geq 0$, and every matrix A has a singular value decomposition with singular values σ_j uniquely determined. The SVD of the matrix A shows that the matrix first applies a unitary transformation preserving the unit sphere via V^* . This is followed by a stretching operation that creates an ellipse with principal semi-axes given by the matrix Σ . Finally, the generated hyper-ellipse is rotated by the unitary transformation U.

2.3 Covariance Matrix

Covariance of two variables X and Y indicate the relationships, correlation, between the two which is defined as the following:

$$cov(X,Y) = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{n-1} \tag{3}$$

We know that in statistics, strongly statistically dependent variables can be considered as redundant observations of the system. Covariance matrix can be helpful to identify redundant data with high dimensions. We follow the formula:

$$\mathbf{C}_x = \frac{1}{n-1}\mathbf{X}\mathbf{X}^T \tag{4}$$

The diagonal terms represent the variance of particular measurements and the off-diagonal terms are the covariances between measurement types. Larger diagonal terms typically represent the dynamics of interest, since the large variance suggest strong fluctuations in that variable.

We can calculate $\mathbf{C}_Y$ where $Y = UX$

$$\mathbf{C}_Y = \frac{1}{n-1}\Sigma^2 \tag{5}$$

2.4 Principal Component Analysis

Principal Component Analysis (PCA) is one of the major application of SVD, which produces low-dimensional reductions of the dynamics and behavior when the governing equations are not known. We generally have three steps to implement the method. First, we need to organize the data into a matrix A $\in \mathbf{C}^{m\times n}$ where M is the number of measurement types

and n is the number of measurements. Then, for the second step, we need to subtract off the mean for each measurement type. Finally, we will compute the SVD and singular values of the covariance matrix to determine the principal components.

In this spring-mass system, we have three cameras recording the motion of a mass, each producing a 2-D representation of data. There are two thing we need to take into consideration: redundancy and SNR (signal-to-noise ratio). SNR is the ratio given as the ratio of variance of the signal and noise fields. A higher SNR suggest almost noiseless data while a low SNR suggests corrupted data by noise.

Section III: Algorithm Implementation & Development

3.1 Load in Data

The first thing is to load in data for a case ('cami 1.mat' with i = 1, 2, 3). We will get three 4-D dataset representing data points of rgb values over time. In order to only focus on the motion of the bucket, we need a filter for each frame to filter out the areas where the bucket never enters. The filter will be set to be a 480×640 matrix which is the same size of the video frame. We will fill the filter only with one and 0. The cropped area would be determined by observing each video.

By using the size comment, we obtain the number of frames in a video. For each frame, we convert the rgb value into grayscale by using rhb2gray and double comment. After that, we multiply the data in grayscale with the filter. From the video, we could see that there is a light on the top of the bucket. We can track the bucket by looking for data points with maximum value in grayscale matrix since the higher grayscale values are, the ligheter the image points.

3.2 Synchronize three cameras

Videos we have for each case are not sync. Therefore, we need to adjust the first frame in each of the three videos to be the frame with the lowest y coordinate. We identify the shortest video, and trim the other two data matrices to make the three data matrices the same frame number. We collect the three data matrices to form a single matrix, which has 6 rows incorporating the data from the three cameras over time.

3.3 SVD Inplementation

In this part, first we need to subtract the mean of each row from the data, take the transpose of it and divide by $\sqrt{n-1}$ in oreder to center the data matrix. Then we take SVD of it to get components S, V, and U. By using command lambda = $diag(s)^2$, we can get the diagonal variance. Then we use command Y = $collected_{data}. * V$ to produce the principal components projection.

Section IV: Computational Results

Test 1 (Ideal Case)

From Figure 1, we learn that only the first principal component which holds the most of the diagonal variances, or energy of interests (0.889) and all the other components have very low dynamics. The projection onto the first principal component basis forms a similar result as the original data measured which is a simple harmonic motion. These show that the system can be reproduced by the first principal component.

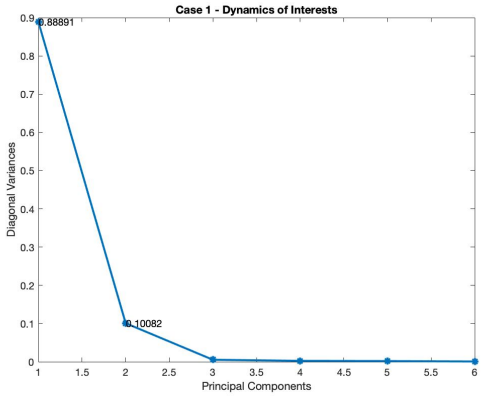

Figure 1: Plot of diagonal variance of PCs in test 1

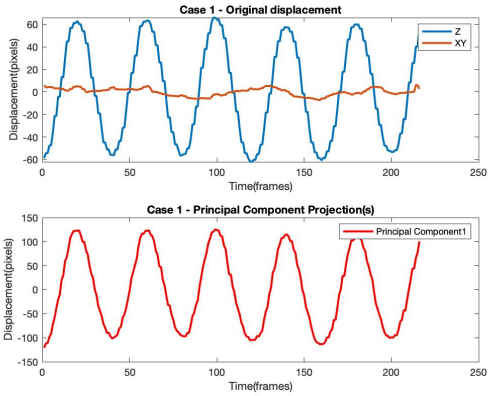

Figure 2: Plot of original displacement and in new basis, test 1

Test 2 (Noisy Case)

Based on Figure 3, we consider three principal components which hold the most of the diagonal variances: 0.643, 0.170, and 0.110. Since there exists noise in the system, the original data and the projection to the principal basis show noise as the bucket oscillations. From this, we know that PCA only reduce noise indicating as an oscillatory behavior.

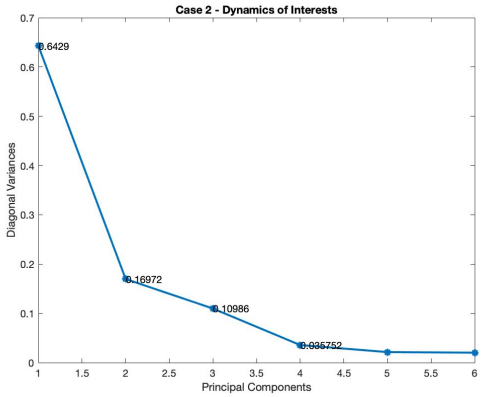

Figure 3: Plot of diagonal variance of PCs in test 2

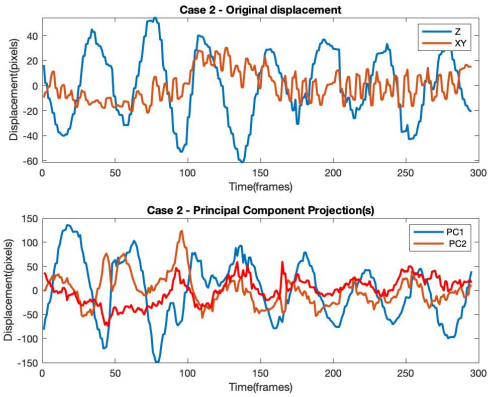

Figure 4: Plot of original displacement and in new basis, test 2

Test 3 (Horizontal Displacement)

From Figure 5, the first four principal components are consider significant with values of 0.5, 0.227, 0.163, and 0.104. The first component holds about half of the energy and the other three hold relatively important amounts of variances. The projections in Figure 6 indicates there exists a simple harmonic motion, pendulum motion, and noise caused by the camera shaking.

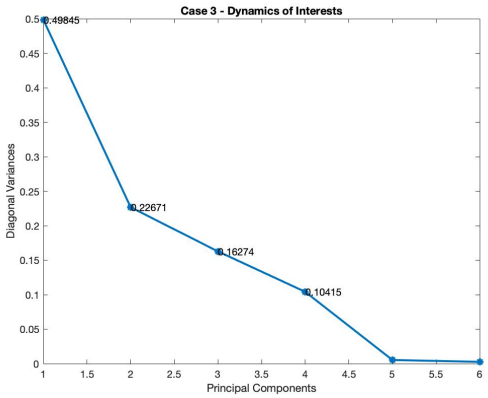

Figure 5: Plot of diagonal variance of PCs in test 3

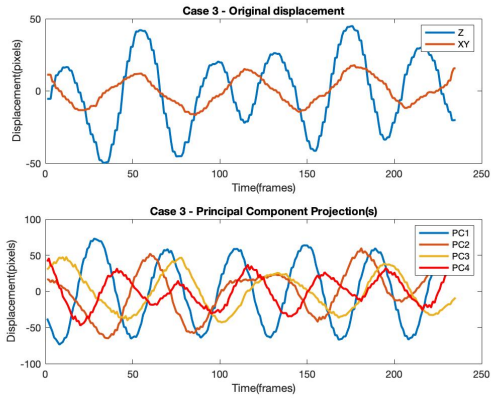

Figure 6: Plot of original displacement and in new basis, test 3

Test 4 (Horizontal Displacement and Rotation)

From Figure 7, there exist three principal components hold relatively high variances, which correspond to the values of 0.691, 0.180, 0.090. The projection in Figure 8 onto the principal component basis have relatively smooth curves and fit the case, as the system experience oscillatory motion, rotation, and pendulum motion in x-y plane. It shows that PCA has achieved the explanation of the mlti-dimensional identities of the spring-mass system.

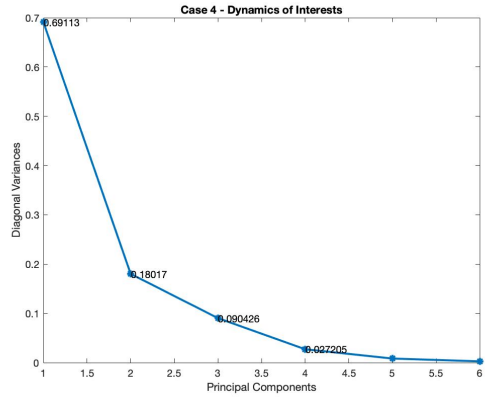

Figure 7: Plot of diagonal variance of PCs in test 4

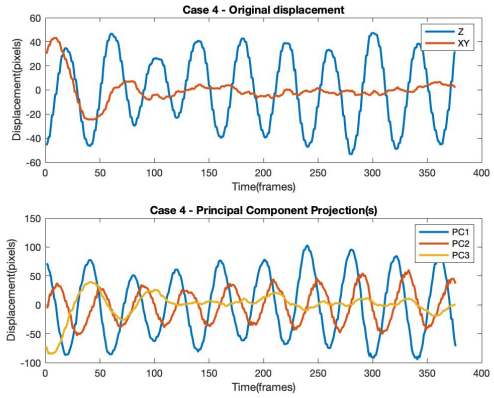

Figure 8: Plot of original displacement and in new basis, test 4

Section V: Conclusion/ Summary

In this report, we were able to track the oscillating bucket for different cases and perform PCA on the dataset to see how many significant principal components existed. We were able to plot the projection onto the various basis sets of the principal components and we were able to see that our projections are accurate to some extent. Overall, we can determine the minimum number of principal components needed to represent a system and reproduce systems into lower-order ones.

Appendix A: Matlab Function Used

Function	Syntax	Brief Explanation
min	[M, I] = min(A)	returns the linear index into A that corresponds to the min value in A
find	find(argument)	return a vector containing the linear indices of each nonzero element in array
size	size(A)	returns a row vector whose elements are the lengths of the corresponding dimension of A
repmat	repmat(arguments)	returns an array containing n copies of A in the row and column dimensions
diag	diag(A)	returns a square diagonal matrix with the elements of vector v on the main diagonal
svd	[U,S,V] = svd(A)	performs a singular value decomposition of matrix A, such that A = U*S*V'

Appendix B: Matlab Code

```
clear all; close all; clc;

%% Case 1 - Ideal Case
clear all; close all; clc;

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

filter = zeros(480, 640);
filter(200:430, 300:400) = 1;
data1 = load_cropped_data(vidFrames1_1, filter, 240);
filter = zeros(480, 640);
filter(100:390, 240:350) = 1;
data2 = load_cropped_data(vidFrames2_1, filter, 240);
```

```
filter = zeros(480, 640);
filter(230:330, 260:480) = 1;
data3 = load_cropped_data(vidFrames3_1, filter, 240);

collected_data = collect(data1, data2, data3);
[m, n] = size(collected_data);
collected_data = collected_data - repmat(mean(collected_data,...
    2),1,n); % subtract mean
[U,S,V]= svd(collected_data'/sqrt(n-1)); % perform the SVD
lambda = diag(S).^2; % produce diagonal variances
Y = collected_data' * V; % produce the principal components projection

figure(1)
plot(1:6, lambda/sum(lambda), '-*', 'Linewidth', 2);
title("Case 1 - Dynamics of Interests");
xlabel("Principal Components"); ylabel("Diagonal Variances");

y = lambda/sum(lambda);
for i=1:2
    text(i,y(i),num2str(y(i)));
end
figure(2)
subplot(2,1,1)
plot(1:216, collected_data(2,:), 1:216, collected_data(1,:),...
    'Linewidth', 2)
ylabel("Displacement(pixels)"); xlabel("Time(frames)");
title("Case 1 - Original displacement");
legend("Z", "XY")
subplot(2,1,2)
plot(1:216, Y(:,1),'r','Linewidth', 2)
ylabel("Displacement(pixels)"); xlabel("Time(frames)");
title("Case 1 - Principal Component Projection(s)");
legend("Principal Component1")

%% Case 2 - Noisy Case
clear all; close all; clc;

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

filter = zeros(480 ,640);
filter(200:400, 300:430) = 1; % Filter for Case2 - Camera1
data1 = load_cropped_data(vidFrames1_2, filter, 240);
filter = zeros(480 ,640);
filter(50:420, 180:440) = 1; % Filter for Case2 - Camera2
```

```
data2 = load_cropped_data(vidFrames2_2, filter, 240);
filter = zeros(480,640);
filter(180:330, 280:470) = 1; % Filter for Case2 - Camera3
data3 = load_cropped_data(vidFrames3_2, filter, 240);

collected_data = collect(data1, data2, data3);
[m,n]= size(collected_data);

collected_data = collected_data - repmat(mean(collected_data,...
    2),1,n); % subtract mean
[U,S,V]= svd(collected_data'/sqrt(n-1)); % perform the SVD
lambda = diag(S).^2; % produce diagonal variances
Y = collected_data' * V; % produce the principal components projection

figure(3)
plot(1:6, lambda/sum(lambda), '-*', 'Linewidth', 2);
title("Case 2 - Dynamics of Interests") ;
xlabel("Principal Components"); ylabel("Diagonal Variances");
y = lambda/sum(lambda);
for i=1:4
    text(i,y(i),num2str(y(i)));
end

figure(4)
subplot(2,1,1)
plot(1:295, collected_data(2,:), 1:295, collected_data(1,:),...
    'Linewidth', 2)
ylabel("Displacement(pixels)"); xlabel("Time(frames)");
legend("Z", "XY")
title("Case 2 - Original displacement");
subplot(2,1,2)
plot(1:295, Y(:,1), 1:295, Y(:,2), 1:295, Y(:,3), 'r', 'Linewidth', 2)
ylabel("Displacement(pixels)"); xlabel("Time(frames)");
title("Case 2 - Principal Component Projection(s)");
legend("PC1" , " PC2")

%% Case 3 - Horizontal Displacement
close all; clear all; clc;

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

filter = zeros(480,640);
filter(240:420, 280:380) = 1; % Filter for Case3 - Camera1
data1 = load_cropped_data(vidFrames1_3, filter, 240);
```

```
filter = zeros(480 ,640);
filter(180:380, 240:400) = 1; % Filter for Case3 - Camera2
data2 = load_cropped_data(vidFrames2_3, filter, 240);
filter = zeros(480 ,640);
filter(180:330, 240:480) = 1; % Filter for Case3 - Camera3
data3 = load_cropped_data(vidFrames3_3, filter, 240);

collected_data = collect(data1, data2, data3);
[m,n]= size(collected_data); % compute data size

collected_data = collected_data - repmat(mean(collected_data,...
                    2),1,n); % subtract mean
[U,S,V]= svd(collected_data'/sqrt(n-1)); % perform the SVD
lambda = diag(S).^2; % produce diagonal variances
Y = collected_data' * V; % produce the principal components projection

figure(5)
plot(1:6, lambda/sum(lambda), '-*', 'Linewidth', 2);
title("Case 3 - Dynamics of Interests");
xlabel("Principal Components"); ylabel("Diagonal Variances");
y = lambda/sum(lambda);
for i=1:4
    text(i,y(i),num2str(y(i)));
end

figure(6)
subplot(2,1,1)
plot(1:235, collected_data(2 ,:), 1:235, collected_data(1,:),...
    'Linewidth', 2)
ylabel("Displacement(pixels)"); xlabel("Time(frames)");
title("Case 3 - Original displacement");
legend("Z", "XY")
subplot(2,1,2)
plot(1:235,Y(:,1),1:235,Y(:,2),1:235,Y(:,3),1:235,Y(:,4),'r',...
    'Linewidth',2)
ylabel("Displacement(pixels)"); xlabel("Time(frames)");
title("Case 3 - Principal Component Projection(s)");
legend("PC1", "PC2", "PC3", "PC4")

%% Case 4 - Horizontal Displacement and Rotation
close all; clear all; clc;

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

```
filter = zeros(480 ,640);
filter(230:440, 330:460) = 1; % Filter for Case4 - Camera1
data1 = load_cropped_data(vidFrames1_4, filter, 240);
filter = zeros(480 ,640);
filter(100:360, 230:410) = 1; % Filter for Case4 - Camera2
data2 = load_cropped_data(vidFrames2_4, filter, 240);
filter = zeros(480 ,640);
filter(140:280, 320:500) = 1; % Filter for Case4 - Camera3
data3 = load_cropped_data(vidFrames3_4, filter, 230);

[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,:);

% Trim the data to make them a consistent length.
% For Test4, the video recorded by camera 3 is the shortest.
% Thus trim the other two as the length of video 3.
data1 = data1(1:length(data3), :);
data2 = data2(1:length(data3), :);

collected_data = [data1'; data2'; data3'];
[m,n]= size(collected_data); % compute data size


collected_data = collected_data - repmat(mean(collected_data,...
                    2),1,n); % subtract mean
[U,S,V]= svd(collected_data'/sqrt(n-1)); % perform the SVD
lambda = diag(S).^2; % produce diagonal variances
Y = collected_data' * V; % produce the principal components projection

figure(7)
plot(1:6, lambda/sum(lambda), '-*', 'Linewidth', 2);
title("Case 4 - Dynamics of Interests");
xlabel("Principal Components"); ylabel("Diagonal Variances");
y = lambda/sum(lambda);
for i=1:4
    text(i,y(i),num2str(y(i)));
end

figure(8)
subplot(2,1,1)
plot(1:376 , collected_data(2,:), 1:376, collected_data(1,:),'Linewidth', 2
ylabel("Displacement(pixels)") ; xlabel("Time(frames)") ;
```

```
title("Case 4 - Original displacement") ;
legend("Z", "XY")
subplot(2,1,2)
plot(1:376, Y(:,1), 1:376, Y(:,2), 1:376, Y(:,3), 'Linewidth', 2)
ylabel("Displacement(pixels)"); xlabel("Time(frames)") ;
title("Case 4 - Principal Component Projection(s)") ;
legend("PC1", "PC2", "PC3")

function collected_data = collect(data1, data2, data3)
    [~,I] = min(data1(1:20,2));
    data1 = data1(I:end,:);
    [~,I] = min(data2(1:20,2));
    data2 = data2(I:end,:);
    [~,I] = min(data3(1:20,2));
    data3 = data3(I:end,:);
    data2 = data2(1:length(data1), :);
    data3 = data3(1:length(data1), :);
    collected_data = [data1'; data2'; data3'];

end

function data = load_cropped_data(vidFrames, filter, scale)
    numFrames = size(vidFrames, 4);
    data = zeros(numFrames, 2);
    for j = 1:numFrames
        X = vidFrames(:,:,:,j);
        Xg = double(rgb2gray(X));
        X_cropped = Xg.* filter;
        threshold = X_cropped > scale;
        [Y, X] = find(threshold);
        data(j,1) = mean(X);
        data(j,2) = mean(Y);
    end
end
```