Homework 3: PCA of Mass Displacement Systems

Audrey Shingleton

February 21, 2020

Abstract

For this project, principal component analysis was used to analyze various videos of oscillating mass systems. The motion of the mass was determined by tracking the light at the top of the paint can. In some of the cases, there is camera shake or rotation/swinging of the paint can. I was able to perform the SVD on the coordinate data from the three cameras and view the projections of the data onto the principal components.

I. Introduction and Overview

I have been given video data of a paint can on a spring for four different oscillation situations. Each system has been recorded by three different cameras such that the motion of the bucket is oversampled. In addition, in many of the systems there is noise from the camera shaking during the recording. The goal of this project is to extract the motion of the bucket using principal component analysis (PCA).

PCA is a useful tool in data dimensionality reduction. Noisy and redundant data can be extracted to capture underlying trends. We can evaluate how much energy each principal component captures and how many dimensions are needed to accurately represent our system.

II. Theoretical Background

The key to identifying redundant data in the systems through PCA is by diagonalizing the covariance matrix C_X so that the diagonal terms are the variances of particular measurements. Larger variances are dynamics of interest.[1] C_X, a square symmetric matrix, is defined as:

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

where X, an mxn matrix contains the experimental data of m measuring positions, such x and y coordinates from 3 different cameras, and n data points.

The idea behind diagonalization is that there is a way to write the system in terms of its principal components in which redundancies have been removed. One way method of diagonalizing C_X is the singular value decomposition (SVD) method. It can be used on any matrix defined as:

$$\mathrm{X}=\mathrm{U}\Sigma\mathrm{V}^*$$

where $\Sigma^2 = \Lambda$, a diagonal matrix whose entries correspond to the eigenvalues of XX^T.[1] Thus the projection of the data onto the principal components is

$$Y = U' * X \tag{2}$$

where U is the unitary transformation associated with the SVD. In this basis, the principal components are the eigenvectors of XX^T[1].

III. Algorithm Implementation and Development

The method to determine the motion of the paint can for each case was solved in a similar manner. For each test case, I determined the number of frames in each of the three videos. Then for each frame in a video, I converted it to gray scale and multiplied the frame matrix element wise by a filter. The filter was a 480x640 matrix, the same size as the frame, containing 0's where I wanted the frame to be black and 1's where I wanted to keep the frame's pixel intensity. This blacked out areas surrounding the paint can oscillations to reduce noise. With the filtered frame, I determined the locations that had a pixel intensity above a certain value which represented the bright light attached to the can. In most cases I searched for intensities greater than 250, but sometimes lowered this cutoff depending on the video. I found the indices of these bright points and averaged their coordinates to represent the location of the paint can throughout the video.

One issue was that the videos from each of the cameras were not synced, meaning the paint can started from a different location in each. To sync the videos, I determined the frame with the lowest y-coordinate from the first 20 frames, and made each of the videos start from that frame. The next issue was that each video still had a different amount of frames. Therefore, I trimmed each video to match the length of the shortest video.

Finally, I created the matrix X, containing 6 rows of the x and y coordinates

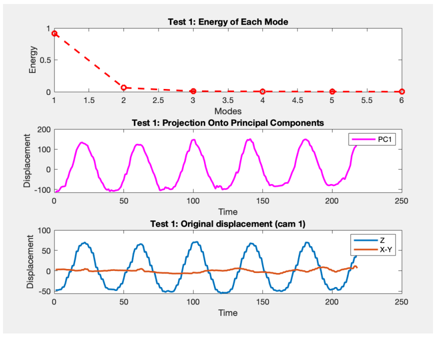

Figure 1: Plots of the energy for each principal component, the projection of the data onto the principal components, and the mass displacement recorded by camera 1 for test 1.

for each camera. X can be defined as

$$X = \begin{bmatrix} x_1 \\ y_1 \\ x_2 \\ y_2 \\ x_3 \\ y_3 \end{bmatrix} \tag{3}$$

I subtracted the mean from each row in X to center the data. Then, I performed the SVD on this matrix.

I squared the values of the diagonal of S from the SVD which gave me the variances of each principal component. I could then plot the energy captured from each mode and determine how many principal components were significant in capturing the motion of the paint can. I then projected the data X onto the principle component basis, producing the matrix Y, and plotted the most significant components.

IV. Computational Results

Test 1: Ideal Case

For test 1, I found that there was only one principal component that captured significantly more energy, approximately 90 percent. The projection of the data onto the first principal component looks nearly identical to the plot of the mass displacement recorded by camera 1, shown in Figure 1. This seems to follow what we would expect as the mass was only oscillating in one direction.

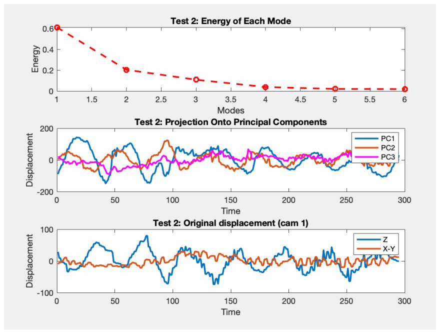

Figure 2: Plots of the energy for each principal component, the projection of the data onto the principal components, and the mass displacement recorded by camera 1 for test 2.

Test 2: Noisy Case

For test 2, it looked like three principal components were important. Even though the mass was oscillating the same way as in test 1, there was significant camera shake. I also found that the projection of the data onto the first three principal components had a similar shaky behavior as the mass displacement captured by camera 1 (Figure 2).

Test 3: Horizontal Displacement

For test 3, I found four components that were important in capturing the energy when horizontal displacement was introduced (Figure 3). It was additionally much more difficult to understand the true motion of the mass in this test using four components than it was in the clean projection produced in test 1.

Test 4: Horizontal Displacement and Rotation

For test 4, three components looked to be significant. The first component only captured about 60 percent of the energy (Figure 4). As the mass had both oscillations and rotation, it makes sense to need three components to capture a pendulum like motion along with a simple harmonic motion.

V. Summary and Conclusions

I was able to determine the motion of an oscillating paint can for different situations despite camera shake, rotations/spinning, and oversampling through using three cameras. I found that principal component analysis is a useful tool in reducing the dimension of our data to minimize redundancy.

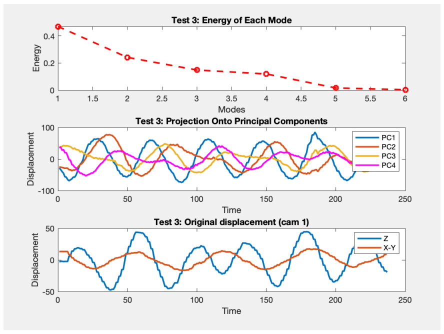

Figure 3: Plots of the energy for each principal component, the projection of the data onto the principal components, and the mass displacement recorded by camera 1 for test 3.

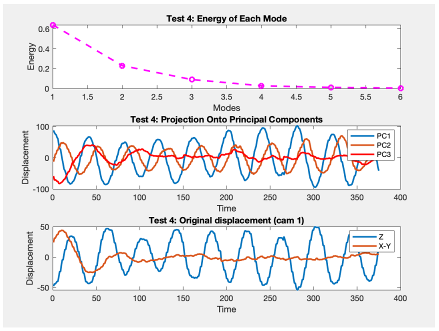

Figure 4: Plots of the energy for each principal component, the projection of the data onto the principal components, and the mass displacement recorded by camera 1 for test 4.

Appendix A. MATLAB functions used and explanations

- rgb2gray: converts a color image to a grayscale image
- diag(X): returns a vector with the diagonal values from a square matrix X
- ind2sub(size, index): returns the coordinates corresponding to a single indices
- find: returns a vector of indices
- frame2im: returns the indexed image data and colormap of a video frame

u,s,v = svd(X): performs the singular value decomposition on matrix X

Appendix B. MATLAB codes

```
clear all; close all; clc;
load('cam1_1.mat')
load('cam2_1.mat')
load('cam3_1.mat')
%%
%Play Videos
frames1 =size(vidFrames1_1);
frames2 =size(vidFrames2_1);
frames3 =size(vidFrames3_1);

%%
for k = 1:frames1(4)
    mov1(k).cdata = vidFrames1_1(:,:,:,k);
    mov1(k).colormap = [];
end

%Play video
width = 50;
filter = zeros(480,640); % size of frame
filter(300-3*width:1:300+3*width, 350-width:1:350+width) = 1; % black out around


data1 = [];
for j=1:frames1(4)
    X=frame2im(mov1(j));
    X = double(rgb2gray(X));
    Xf = X.*filter;
    thresh = Xf > 250; % pick out bright points
    coordinates = find(thresh);
    [y, x] = ind2sub(size(thresh),coordinates);

    data1 = [data1; mean(x), mean(y)];
    % play filtered video
%       subplot(1,2,1)
%       imshow(uint8((thresh * 255))); drawnow
%       subplot(1,2,2)
%       imshow(uint8(Xf)); drawnow
end

%pause(5)
close all;
```

```
%%
for k = 1:frames2(4)
    mov2(k).cdata = vidFrames2_1(:,:,:,k);
    mov2(k).colormap = [];
end

filter = zeros(480,640);
filter(250-3*width:1:250+3*width, 290-1.5*width:1:290+1.5*width) = 1;

data2 = [];
%Play video
for j=1:frames2(4)
    X=frame2im(mov2(j));
    X = double(rgb2gray(X));
    Xf = X.*filter;
    thresh = Xf > 250;
    indeces = find(thresh);
    [y, x] = ind2sub(size(thresh),indeces);
    data2 = [data2; mean(x), mean(y)];

%       subplot(1,2,1)
%       imshow(uint8((thresh * 255))); drawnow
%       subplot(1,2,2)
%       imshow(uint8(Xf)); drawnow
end

%%
for k = 1:frames3(4)
    mov3(k).cdata = vidFrames3_1(:,:,:,k);
    mov3(k).colormap = [];
end

filter = zeros(480,640);
filter(250-1*width:1:250+2*width, 360-3*width:1:360+3*width) = 1;

data3 = [];
%Play video
for j=1:frames3(4)
    X=frame2im(mov3(j));
    X = double(rgb2gray(X));
    Xf = X.*filter;
    thresh = Xf > 247;
    indeces = find(thresh);
    [y, x] = ind2sub(size(thresh),indeces);

    data3 = [data3; mean(x), mean(y)];
```

```
%       subplot(1,2,1)
%       imshow(uint8((thresh * 255))); drawnow
%       subplot(1,2,2)
%       imshow(uint8(Xf)); drawnow
end

%% Synch frames based on min y value
[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 frames to min size
min = min([length(data1), length(data2), length(data3)]);
data1 = data1(1:min, :);
data2 = data2(1:min, :);
data3 = data3(1:min, :);

%% SVD
X = [data1';data2';data3'];

[m,n]=size(X); % compute data size
mn=mean(X,2); % compute mean for each row
X=X-repmat(mn,1,n); % subtract mean

[u,s,v]=svd(X); % perform the SVD
lambda=diag(s).^2; % produce diagonal variances
Y = u'*X;
%%

figure()
subplot(3,1,1)
plot(1:6, lambda/sum(lambda), 'ro--', 'Linewidth', 2);
title("Test 1: Energy of Each Mode");
xlabel("Modes"); ylabel("Energy");
subplot(3,1,2)
plot(1:min, Y(1,:),'m','Linewidth', 2)
ylabel("Displacement"); xlabel("Time");
title("Test 1: Projection Onto Principal Components");
legend("PC1")
subplot(3,1,3)
```

```
plot(1:min, X(2,:),1:min, X(1,:), 'Linewidth', 2)
ylabel("Displacement"); xlabel("Time");
title("Test 1: Original displacement (cam 1)");
legend("Z", "X-Y")
```

References

[1] Kutz, Nathan J. "Computational Methods for Data Analysis." *In Data-Driven Modeling Scientific Computation: Methods for Complex Systems Big Data.*. Oxford University Press, 2013.