AMATH 482 Homework 3
Brad Hong
February 24, 2021

Abstract
This report will consist of analyzing different movie files and extracting a pattern from the files to perform a Principal Component Analysis algorithm.

Section I. Introduction and Overview

The main objective of this experiment is to apply the Principal Component Analysis algorithm on multiple movie files to determine its usefulness. There will be four tests, three of which will introduce a different type of noise to the problem and one of which is the ideal case to compare with. Each of the movies will contain a mass in simple harmonic motion moving in the z direction. In the ideal case, the video will have a clear motion and minimal noise to easily extract the simple harmonic motion. In the second test, there will be camera shake to make extracting the motion of the mass harder. In the third test, the mass will sway from side to side. Finally, in the final test, the mass will sway from side to side as well as be rotated. Each test has 3 movie files, each of which are the same test taken in different points of view.

Section II. Theoretical Background

Explain PCA and Single Value Decomp

The main concepts that will be observed are, of course, the Principal Component Analysis (PCA) and with that comes the Single Value Decomposition (SVD).

The SVD is a method of deconstructing a matrix into different parts much like diagonalizing a matrix. It deconstructs the matrix A below in the following way

$$A = U\Sigma V^* \quad (1)$$

This equation, while seems unimpressive at first, is actually quite powerful. The V* matrix is used as a change of basis matrix to align the vectors of x to the correct axis. Then multiplying by sigma stretches/shrinks the vector to size and finally the U vector properly orients the vectors. These three matrices explain the process that is undergone when Ax is performed.

SVD is a useful tool in creating low dimensional models. These models, when created from geometric matrices of data, are called the PCA. The PCA consists of the principal components which are represented by this equation:

$$A = \sum_{j=1}^{r} \sigma_j u_j v_j^* \quad (2)$$

This equation represents the sum of all principal components of A. This means that each iteration of the summation represents a principal component of A. These values are all derived from the SVD of A where σ_j is the j-th diagonal of Sigma, u_j is the j-th column of U, and v_j^* is the j-th column of V*. Since the values in Sigma are ordered in descending order, the first principal component represents the component with the largest amount of information. The takeaway here is that, if is σ_j is near zero or small in comparison to other σ_i, then the j-th component can be ignored. This means we are able to create extremely accurate models of data using a handful of components.

Section III. Algorithm Implementation and Development

The major challenge in this experiment is actually from gathering data from the movies themselves. In order to perform a PCA we first need a matrix to perform it on. In order to do this we must extract the position of the mass from the three video angles for each test. The idea was to use the light on the top of the mass as a method of determining the simple harmonic motion. To do this I noticed that the light source was the brightest on the screen and assumed that the pixel with the largest rgb value would be where the mass would be. Another method I used was to determine the largest difference in pixel color after 4 frames. However, it turned out that the latter was not so great in finding the location of the mass as there were other things in the video that would move. So, I went with the former for my experiment.

Once the x,y coordinates of the brightest pixels were extracted from the video for each angle into xi and yi where i = 1,2,3, the six variables were trimmed so that they would all be the same length. However, this was not enough as the videos also needed to line up. I looked into the values of y's and determined where there were peaks in the values to sync up the six variables. I also subtracted the mean of all of the variables to center the oscillation around 0 as this would synchronize all of the oscillations no matter where they were on the video. Once this was done we had a 6 by N matrix to perform the SVD on.

Similar processes were taken for tests 2,3,4 to compare the results. However, I do understand that the light source might not have always been present in some of the videos for the whole duration. The data extraction was not something I was too familiar with going into this experiment and would like to explore that more if given the time.

Section IV. Computational Results

Test 1:

For this test, after completing the procedure above, I got very strange sigma values. I expected there to be a lot of data gathered around the first principal component since the mass is moving in one axis. However, from the results I got, all of the sigma values were: 181, 120, 102, 90, 71,

45, meaning there was still a lot of noise. With this information I decided to get the POD mode graph of time to see any trends.

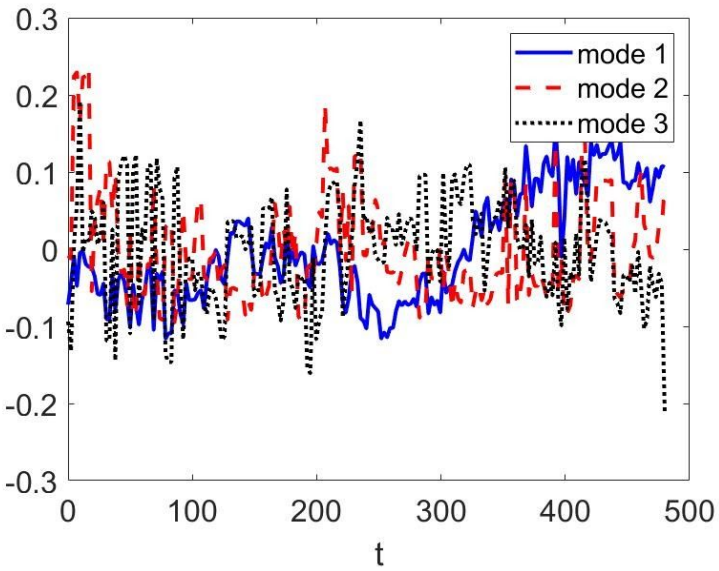


Figure 1: The POD Modal Graph of Time for Test 1 (3 modes are shown)

As seen above, the blue line shows a little bit of a sinusoidal curve whereas the black and red just seem like garbage.

Test 2:

In test two, I actually got better sigma values as σ_1 was ~4000 and σ_2 was ~2000 and σ_3 was ~1000. This meant that there was 6 times as much data in σ_1 and σ_2 than in σ_3. This is somewhat what I expected as the noise from the camera shake should be explained by the x and y coordinate or in other words through the second and third principal components. The V* vs time graph looked something like this:

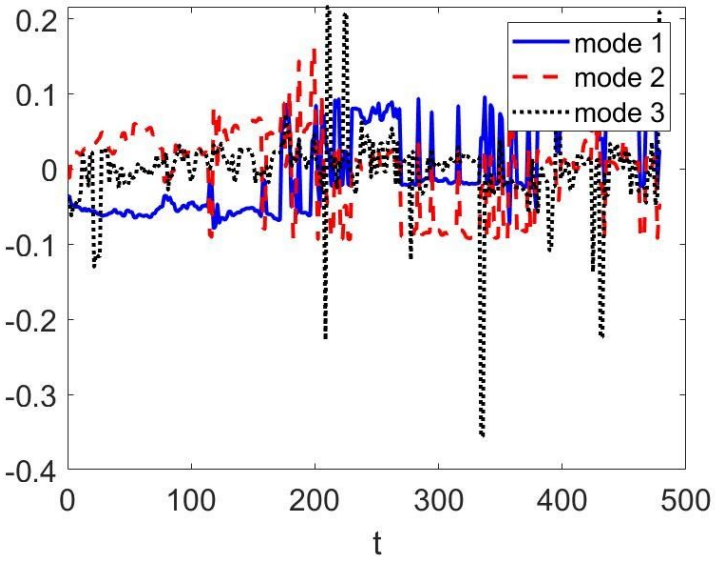
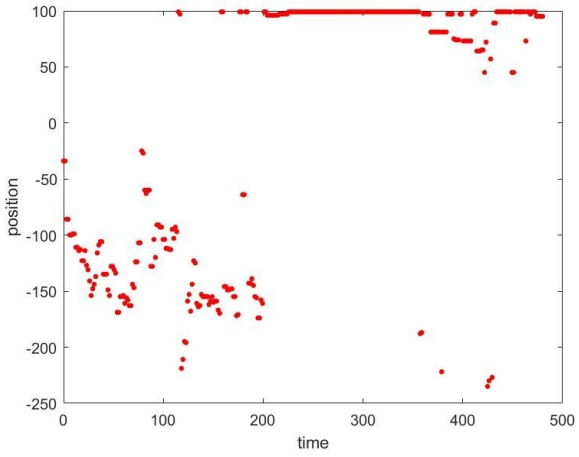


Figure 2: The POD Modal Graph of Time for Test 2 (3 modes are shown)

It was after looking at this test that I realized that a lot of the data I was getting didn't really make much sense. So I decided to explore what was going wrong. I plotted the graph of time vs position for different x and y values. Below are some of the graphs I saw:



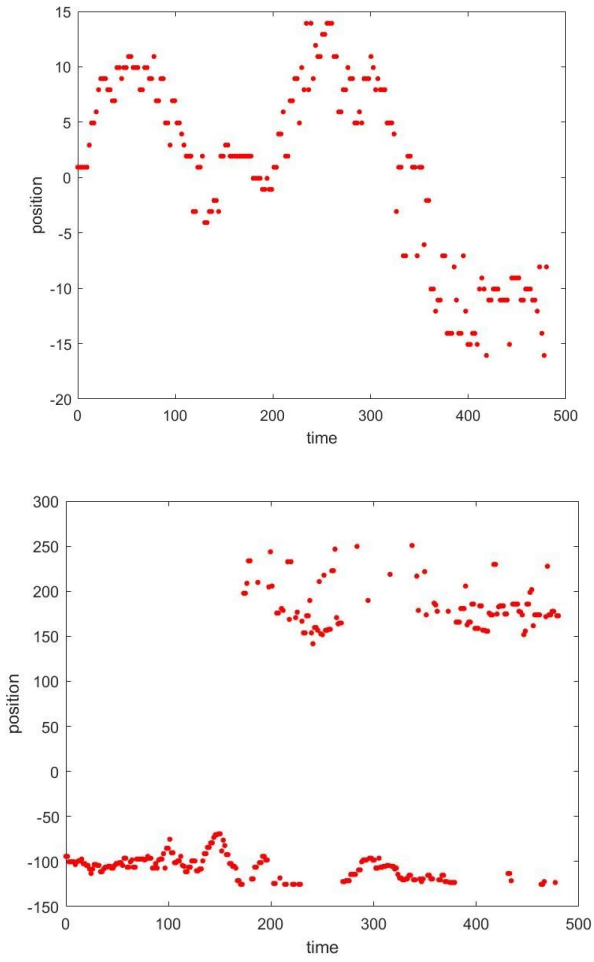


Image 3: The position vs time graphs for y1 in test 1 and y1, y3 in test 2

As seen above, the graphs of position vs time were not as clean as I had hoped. This realization was unfortunately too late into the experiment to fix. I realized that the algorithm I used to extract the data was horrible thus, the data I received from the SVD was nonsensical.

I planned on doing tests 3 and 4 in a similar manner, but due to the amount of work I needed to redo and the amount of time I had left, I decided it would be best to turn in what I had already done instead of completely scrapping everything that I had.

Section V. Summary and Conclusions

I firmly believe this report made me get a better understanding of the SVD and PCA, however, due to the lack of understanding on extracting data from a video I was unable to get substantial data to back this claim up. Although the data might not show it, I know that the PCA is a powerful tool that allows us to make powerful models of data using very little information. Given more time, I would like to revisit this experiment with a better understanding of how to extract data from a movie. With much better values for x and y positions, I feel like I would be able to

better understand the power of the PCA. With that said, the data I received was extremely incorrect due to the error in the x,y positions of the mass extracted. From what I understand, regardless of the noise, the PCA should have been able to properly model the motion of the mass. From the ideal case, the majority of the model should be explained through σ_1 however this was not the case. In test 2, since the camera shake wasn't extremely bad, the majority of the data should still be explained by σ_1 and some of the noise from the horizontal shake in σ_2. In test 3, I believe the swaying would be similar to test 2's noise in that the majority of the data would be explained by σ_1 however x and y swaying would be explained in σ_2 and σ_3. Finally, test 4 would be one the hardest to decipher mainly due to the difficulty in getting x and y. I imagine this would be even harder than simplifying following the light as it would sometimes be rotated away from the camera.

Appendix A. MATLAB functions used and brief implementation explanation

Image Processing Toolbox:
The toolbox was mainly used to watch the videos for each test.

svd()
Used to get the Single Value Decomposition of a matrix. A second parameter ‘econ’ is used to use the simplified version of the SVD to save time.
This function return values U, S, and V each of which correspond to U, Sigma, and V from the SVD algorithm. Note that V should be transposed to get V*.

Appendix B. MATLAB codes

```
[height1, width1, rgb1, num_frames1] = size(vidFrames1_1);
[height2, width2, rgb2, num_frames2] = size(vidFrames2_1);
[height3, width3, rgb3, num_frames3] = size(vidFrames3_1);
x1 = zeros(1, num_frames1);
y1 = zeros(1, num_frames1);
x2 = zeros(1, num_frames1);
y2 = zeros(1, num_frames1);
x3 = zeros(1, num_frames1);
y3 = zeros(1, num_frames1);
for j=1:num_frames1-23
    pixels1 = zeros(height1, width1);
    for i=1:rgb1
        pixels1 = pixels1 + im2double(vidFrames1_1(:,:,i,j));
    end
    [pixdiffmax, indices] = max(pixels1, [], 'linear');
    x1(j) = indices(1);
    y1(j) = indices(2);
end
for j=1:num_frames1
    pixels1 = zeros(height2, width2);
    for i=1:rgb2
        pixels1 = pixels1 + im2double(vidFrames2_1(:,:,i,j));
    end
    [pixdiffmax, indices] = max(pixels1, [], 'linear');
    x2(j) = indices(1);
    y2(j) = indices(2);
end
for j=1:num_frames1
```

```
    pixels1 = zeros(height3, width3);
    for i=1:rgb3
        pixels1 = pixels1 + im2double(vidFrames3_1(:,:,i,j));
    end
    [pixdiffmax, indices] = max(pixels1, [], 'linear');
    y3(j) = indices(1);
    x3(j) = indices(2);
end
x1 = x1(1:length(x1)-23);
y1 = y1(1:length(y1)-23);
x1 = x1 - mean(x1);
y1 = y1 - mean(y1);
x2 = x2(21:length(x2)-3);
y2 = y2(21:length(y2)-3);
x2 = x2 - mean(x2);
y2 = y2 - mean(y2);
x3 = x3(23:length(x3)-1);
y3 = y3(23:length(y3)-1);
x3 = x3 - mean(x3);
y3 = y3 - mean(y3);

% code 2
M = [x1;y1;x2;y2;x3;y3];
X = linspace(0,width1,length(x1));
t = linspace(0,height1,length(x1));
plot(t, x3, 'r.','Markersize',10);
xlabel('time')
ylabel('position')

% code 3
[U,S,V] = svd(M,'econ');
X_rank1 = S(1,1)*U(:,1)*V(:,1)';
plot(X_rank1(1,:), X_rank1(2,:),'r.','MarkerSize',10)

legend('mode 1','mode 2','mode 3','Location','best')
plot(t,V(:,1),'b',t,V(:,2),'--r',t,V(:,3),':k','Linewidth',2)
xlabel('t')
set(gca,'Fontsize',16)
```