AMATH482 HOMEWORK3, PRINCIPAL COMPONENT ANALYSIS

CIHAN MERTOZ

Abstract. In this report, we will use one of the most important tools in linear algebra called Singular Value Decomposition(SDV) to analyze the dynamics of a mass bouncing on a spring. To do that we will use different data sets generated by cameras.

1. Introduction and Overview

The main advantage of SDV is that we don't have to know the analytical equations governing the physical system we are dealing with. We can simply make data-driven analysis on the system. In this assignment we have four different cases where a mass is moving. The first one is called the ideal form where the entire motions is in the z axis. The second case is called the noisy case because we are now shaking the camera while recording the same set up as the first one Thirdly, we added a horizontal displacement to the first case. Lastly, we are also rotating the mass in addition to the third case.

2. Theoretical Background

2.1. **Singular Value Decomposition.** From Linear Algebra we know that a matrix is nothing but a rectangular array that stretches and rotates a vector. Singular Value Decomposition is the decomposition of a Matrix into three different matrices:

$$A = U\Sigma V^* \tag{1}$$

This is important because SVD decomposes a matrix into two rotating and a stretching matrix. This is very powerful because we can derive from this that the conjugate transpose of U and V are their inverses:

$$U^*U = UU^* = I \tag{2}$$

$$V^*V = VV^* = I \tag{3}$$

Σ on the other hand is a diagonal matrix which has real and positive diagonal entries. This proves very useful to eliminate something called redundancy because the number of non-zero entries are equal to the rank of the matrix. Thus we can eliminate values that has the value 0.

Date: February 2020.

2.2. **Principal Component Analysis(PCA).** PCA is basically a procedure that reduces the dimensionality of redundant data sets. Since in this problem we have 4 different data sets and they have similar dimensions, we can arrange the data into this matrix:

$$A = \begin{bmatrix} x_1 \\ y_1 \\ x_2 \\ y_2 \\ .. \\ .. \\ x_n \\ y_m \end{bmatrix} \tag{4}$$

Each row of this matrix represents one measurement. We can then find the joint probability, covariance, between the measurement pairs. Larger covariances imply redundancy between vectors. Another useful tool to statistically analyze the vectors is the variance. It shows the unique features of the matrix.

3. Algorithm Implementation and Development

We have 3 separate cameras recording the same instance. This raises the question: is there any redundancy in our data. Since we have 4 different cases with 3 camera recordings corresponding to each case, I coded for separate solutions. Knowing PCA deals with redundancy well, first I tried to implement PCA to find the dominant data points. Firstly, I uploaded the data and then converted them into arrays. Then I converted everything to gray to get rid of any possible light pollution from the environment. Since we have a moving object and it appears brighter than the rest of the environment, I found the brightest points in the data set . This tracks the position of the mass. After finding the positions of the mass for each data set generated by 3 different cameras, I put them in a new matrix. This matrix had both the X and the Y values from the vectors. After subtracting the means from each row, I diagonalized the data using the SVD command in MATLAB. This gave us the three components of the SVD. Lastly to find the energy levels we used the normalized Σ values to calculate energy levels.

4. Computational Results

4.1. **Ideal Case.** In this case the mass is only moving in the z-direction. Here are the Time vs Location and Principal Component vs Energy level graphs. Since in this case the mass is only moving in the z direction the principal components are very dominant.

4.2. **Noisy Case.** In this case the mass is again only moving in the z-direction but with a shaking camera. Here are the Time vs Location and Principal Component vs Energy level graphs. In this case, there is a lot of noise because of the shaking. Thus the principal components are less dominant and the location vs time graph is much more noisy.

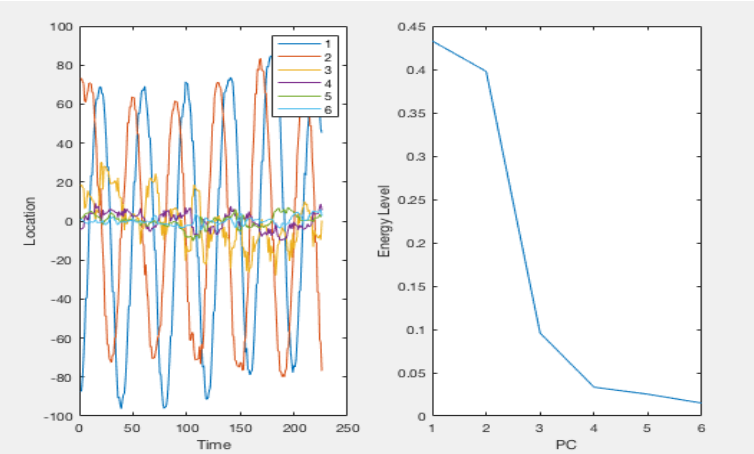

FIGURE 1. Case1, Ideal

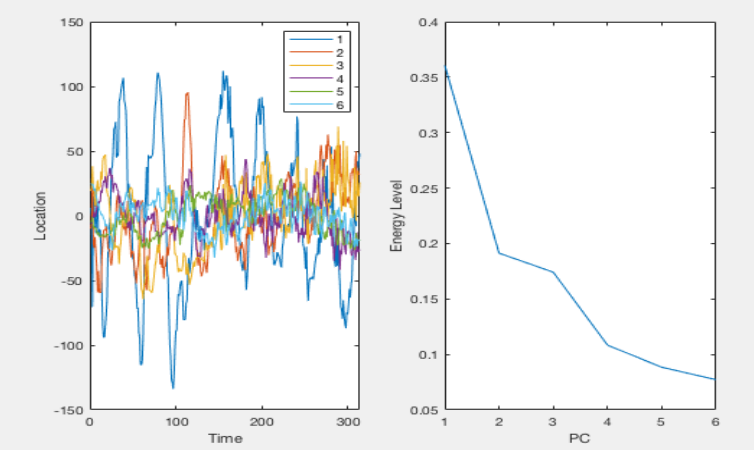

FIGURE 2. Case2, Shaking Camera

4.3. **Horizontal displacement.** In this case, the mass is released off-center so as to produce motion in the x y plane as well as the z direction. Here are the Time vs Location and Principal Component vs Energy level graphs. We can clearly see the oscillation in both horizontal and the vertical direction.

4.4. **Horizontal displacement and rotation.** In this case, the mass is released off-center and rotates so as to produce motion in the x y plane, rotation, and motion in the z direction.Here are the Time vs Location and Principal Component vs Energy level graphs. Although there is rotation in the mass, there is no clear evidence of the rotation.

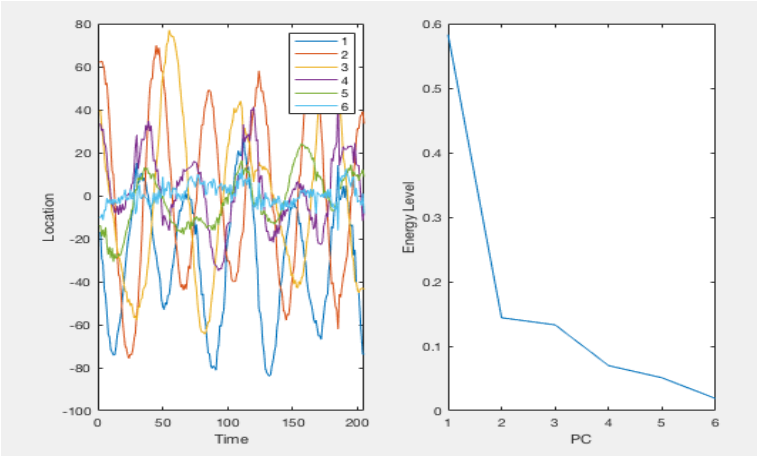

FIGURE 3. Case3, Horizontal Case

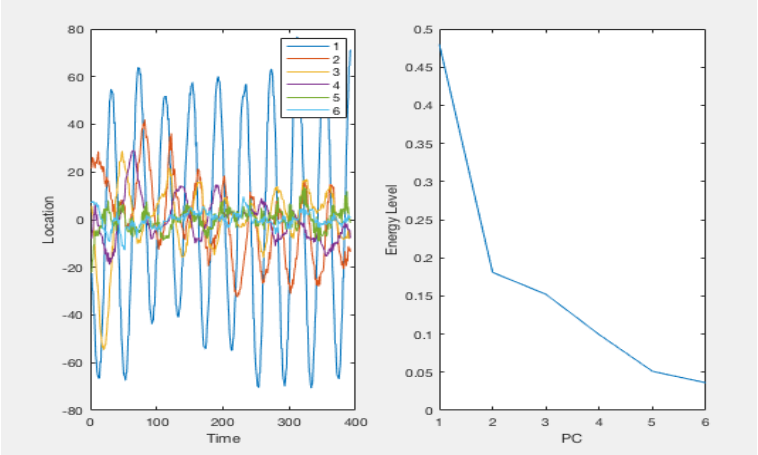

FIGURE 4. Case4, Horizontal displacement and rotation

5. CONCLUSION

In this assignment we analyzed 4 different cases with the principal component analysis. Each case presented slightly different results. As a conclusion, we can easily say that noise is a factor that needs to be handled well to get the best results out of the PCA method. Lastly, we saw that PCA does very poorly to detect rotation which is very essential information.

6. Appendix A

I = rgb2gray(): converts the truecolor image RGB to the grayscale image I. [u,s,v] = svd() produces an economy-size decomposition of m by n matrix A.

7. Appendix B

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

%load the images
load('cam1_1.mat')
load('cam2_1.mat')
load('cam3_1.mat')
% components of the video frame
[x,y,color,l] = size(vidFrames1_1);
[compx1,compy1,compx2,compy2,compx3,compy3] = deal(zeros(1,l));
filter = 0.95;



for k = 1:l
        X = double(rgb2gray(vidFrames1_1(:,:,:,k)));
        % frames
        X(1:220,:) = 0;
        X(:, 1:320) = 0;
        X(:, 390:end) = 0;
        Y = double(rgb2gray(vidFrames2_1(:,:,:,k)));
        Y(1:120,:) = 0;
        Y(:, 1:260) = 0;
        Y(:, 350:end) = 0;
        Z = double(rgb2gray(vidFrames3_1(:,:,:,k)));
        Z(1:270,:) = 0;
        Z(:, 1:290) = 0;
        Z(:, 440:end) = 0;


        % the components
        M = max(X(:));
        [maxX1,maxX2] = find(X >= M * filter);
        compx1(k) = mean(maxX1);
        compy1(k) = mean(maxX2);
        M = max(Y(:));
        [maxY1,maxY2] = find(Y >= M * filter);
        compx2(k) = mean(maxY1);
        compy2(k) = mean(maxY2);
```

```
        M = max(Z(:));
        [maxZ1,maxZ2] = find(Z >= M * filter);
        compx3(k) = mean(maxZ1);
        compy3(k) = mean(maxZ2);


end


% Adding the components to a new matrix
M = [compx1; compy1; compx2; compy2; compx3; compy3];

% singular value decomposition
[a,b] = size(M); mean = mean(M,2); M =  M - repmat(mean,1,b);

[u,s,v] = svd(M,'econ');


subplot(1,2,1)
plot(v*s)
xlabel('Time');ylabel('Location')
legend('1','2','3','4','5','6')
subplot(1,2,2)
plot(diag(s)/sum(diag(s)))
xlabel('PC');ylabel('Energy Level')


% Case 2
clear all; close all ;clc;

%load the images
load('cam1_2.mat')
load('cam2_2.mat')
load('cam3_2.mat')
% components of the video frame
[x,y,color,l] = size(vidFrames1_2);
[compx1,compy1,compx2,compy2,compx3,compy3] = deal(zeros(1,l));
filter = 0.9;

for k = 1:l
        X = double(rgb2gray(vidFrames1_2(:,:,:,k)));
        % frames
        X(1:200,:) = 0;
        X(:, 1:300) = 0;
        X(:, 401:end) = 0;
```

```
        X(:,400:end) = 0;
        Y = double(rgb2gray(vidFrames2_2(:,:,:,k)));
        Y(1:50,:) = 0;
        Y(:, 1:200) = 0;
        Y(370:end,:) = 0;
        Y(:, 401:end) = 0;
        Z = double(rgb2gray(vidFrames3_2(:,:,:,k)));
        Z(1:200,:) = 0;
        Z(320:end,:) = 0;
        Z(:, 1:280) = 0;
        Z(:, 501:end) = 0;


        % the components
        M = max(X(:));
        [maxX1,maxX2] = find(X >= M * filter);
        compx1(k) = mean(maxX1);
        compy1(k) = mean(maxX2);
        M = max(Y(:));
        [maxY1,maxY2] = find(Y >= M * filter);
        compx2(k) = mean(maxY1);
        compy2(k) = mean(maxY2);
        M = max(Z(:));
        [maxZ1,maxZ2] = find(Z >= M * filter);
        compx3(k) = mean(maxZ1);
        compy3(k) = mean(maxZ2);

end

% Adding the components to a new matrix
M = [compx1; compy1; compx2; compy2; compx3; compy3];

% singular value decomposition
[a,b] = size(M); mean = mean(M,2); M = M - repmat(mean,1,b);

[u,s,v] = svd(M,'econ');

subplot(1,2,1)
plot(v*s)
xlabel('Time');ylabel('Location')
legend('1','2','3','4','5','6')
subplot(1,2,2)
plot(diag(s)/sum(diag(s)))
xlabel('PC');ylabel('Energy Level')
```

```
% Case 3
clear all; close all ;clc;

%load the images
load('cam1_3.mat')
load('cam2_3.mat')
load('cam3_3.mat')
% components of the video frame
[x,y,color,l] = size(vidFrames1_3);
[compx1,compy1,compx2,compy2,compx3,compy3] = deal(zeros(1,l));
filter = 0.98;
l = l-10;
for k = 1:l
        X = double(rgb2gray(vidFrames1_3(:,:,:, k)));
         % frames
        X(1:200,:) = 0;
        X(:, 1:250) = 0;
        X(:, 401:end) = 0;
        Y = double(rgb2gray(vidFrames2_3(:,:,:, k)));
        Y(1:200,:) = 0;
        Y(:, 1:200) = 0;
        Y(:, 401:end) = 0;
        Z = double(rgb2gray(vidFrames3_3(:,:,:,k)));
        Z(1:200,:) = 0;
        Z(:, 1:280) = 0;
        Z(:, 501:end) = 0;


        % the components
        M = max(X(:));
         [maxX1,maxX2] = find(X >= M * filter);
         compx1(k) = mean(maxX1);
         compy1(k) = mean(maxX2);
        M = max(Y(:));
         [maxY1,maxY2] = find(Y >= M * filter);
         compx2(k) = mean(maxY1);
         compy2(k) = mean(maxY2);
        M = max(Z(:));
         [maxZ1,maxZ2] = find(Z >= M * filter);
         compx3(k) = mean(maxZ1);
         compy3(k) = mean(maxZ2);

end
```

```
% Adding the components to a new matrix
M = [compx1; compy1; compx2; compy2; compx3; compy3];

% singular value decomposition
[a,b] = size(M); mean = mean(M,2); M = M - repmat(mean,1,b);

[u,s,v] = svd(M,'econ');

subplot(1,2,1)
plot(v*s)
xlabel('Time');ylabel('Location')
legend('1','2','3','4','5','6')
xlim([0 205])
subplot(1,2,2)
plot(diag(s)/sum(diag(s)))
xlabel('PC');ylabel('Energy Level')

% Case 4
clear all; close all ;clc;

%load the images
load('cam1_4.mat')
load('cam2_4.mat')
load('cam3_4.mat')
% components of the video frame
[x,y,color,l] = size(vidFrames1_4);
[compx1,compy1,compx2,compy2,compx3,compy3] = deal(zeros(1,l));
filter = 0.9;

for k = 1:l
        X = double(rgb2gray(vidFrames1_4(:,:,:,k)));
        % frames
        X(1:200,:) = 0;
        X(:, 1:300) = 0;
        X(:, 501:end) = 0;
        Y = double(rgb2gray(vidFrames2_4(:,:,:,k)));
        Y(1:50,:) = 0;
        Y(:, 1:220) = 0;
        Y(:, 411:end) = 0;
        Z = double(rgb2gray(vidFrames3_4(:,:,:,k)));
        Z(1:150,:) = 0;
        Z(:, 1:300) = 0;
        Z(:, 511:end) = 0;
```

```
        M = max(X(:));
        [maxX1,maxX2] = find(X >= M * filter);
        compx1(k) = mean(maxX1);
        compy1(k) = mean(maxX2);
        M = max(Y(:));
        [maxY1,maxY2] = find(Y >= M * filter);
        compx2(k) = mean(maxY1);
        compy2(k) = mean(maxY2);
        M = max(Z(:));
        [maxZ1,maxZ2] = find(Z >= M * filter);
        compx3(k) = mean(maxZ1);
        compy3(k) = mean(maxZ2);

end

% Adding the components to a new matrix
M = [compx1; compy1; compx2; compy2; compx3; compy3];

% singular value decomposition
[a,b] = size(M); mean = mean(M,2); M =  M - repmat(mean,1,b);

[u,s,v] = svd(M,'econ');


subplot(1,2,1)
plot(v*s)
xlabel('Time');ylabel('Location')
legend('1','2','3','4','5','6')
subplot(1,2,2)
plot(diag(s)/sum(diag(s)))
xlabel('PC');ylabel('Energy Level')
```