Homework 3: Principal Component Analysis

AMATH 482: Computational Methods for Data Analysis

Rosemichelle Marzan

February 21, 2020

Abstract **– The positional coordinates of a moving object filmed using three cameras in four different experiments were obtained with a computational algorithm. Singular value decomposition was then performed on these data sets, the results of which were then interpreted with principal component analysis. This report demonstrates the practical usefulness of principal component analysis for examining real-world data which may contain redundancies in information and noise, and how these factors influence the accuracy of the algorithm.**

1 Introduction and Overview

In this study, the position of a moving object was tracked in four different experiments. The recordings are of a paint can attached to a string and is made to move in a simple harmonic motion (cases 1 and 2, with and without camera shake, respectively), a pendulum motion (case 3), and a combination of oscillatory and pendulum motion and rotation (case 4). Three cameras were used to film the motion in all four cases, two of which were filmed upright but at different locations within the testing room, and the third filmed at an unknown angle. The recordings were not all taken at the same time or for the same duration within an experiment. Afterwards, a principal component analysis was applied to the acquired data.

2 Theoretical Background

Principal component analysis (PCA) is a powerful statistical tool that reduces large data sets to statistically independent components. In a way, it filters the signal from the noise caused by redundancies in information, errors in data acquisition, interference, etc. Singular value decomposition (SVD) is a computational method for PCA implementation. SVD transforms a set of vectors by stretching/compressing and rotating them. A matrix M containing the data to be analyzed can be represented by a hyperellipse in $\mathbb{R}^m$. If we start off with a unit hypersphere, performing an SVD essentially yields the set of instructions for converting the unit hypersphere into the hyperellipse. These instructions are in the form of three matrices which indicate how much to stretch/compress and rotate the unit hypersphere. The product of these matrices is $\mathbf{M}$ (Eqn. 1):

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

$\mathbf{U}$ contains the principal components, or the left singular vectors of $\mathbf{M}$, and can be thought of as the rotation instructions in our analogy. The diagonal values of $\mathbf{\Sigma}$ are the singular values of $\mathbf{M}$, which scale the principal component vectors. And $\mathbf{V}$ are the right singular vectors of $\mathbf{M}$, which perform an intermediate rotation step. The vectors in $\mathbf{U}$ and $\mathbf{V}$ are orthonormal to each other; hence, they form the basis of the given matrix. For a visual representation, see Figure 1.

The SVD values can be interpreted in the context of the desired application by measuring the energy contained in each mode (as described by the singular value). In this context, a "mode" can be thought of as a matrix "basis". The energy can be thought of as the significance factor of that mode. Examining all of the energies within a matrix allows us to consider how much redundancy is present in the data. In the case of the data points obtained in the ideal case, for example, I expected to see the first mode containing >95% of the energy, or statistically significant information, of the system. This is because two out of the three cameras demonstrate most movement occurring in the y-direction (in 2D; in 3D reality, this would be the z-direction). Any minor motion along the x-direction will be regarded as noise by the PCA algorithm. The third camera also shows movement in a single direction, but this direction is at an angle to both the x- and y-axes, therefore I also expected the second mode to contain most of the remaining 5% energy. Essentially, the more energy that is present in the first few modes, the less new or meaningful information provided by additional camera recordings of the same experiment. Furthermore, examining the energies can tell us how noise within a data source (such as camera shake) can necessitate the use of additional data sources to provide meaningful information.

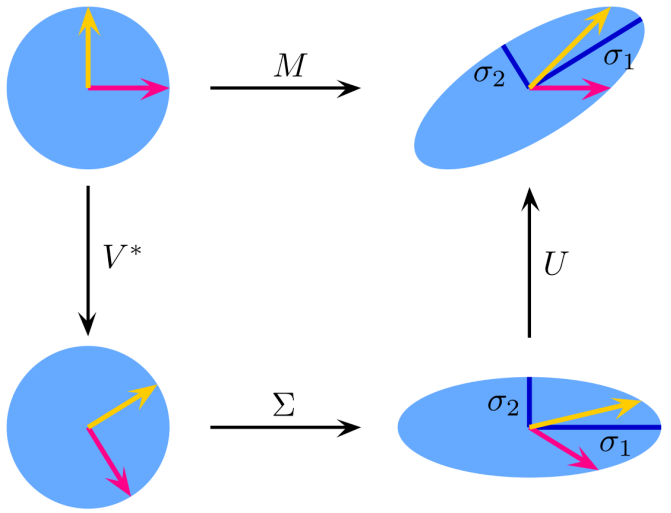

Figure 1: The singular value decomposition, depicted as a physical transformation of a 2D matrix.

3 Algorithm Implementation and Development

3.1 Data Acquisition

The experimental setup is depicted in Figure 2 (left). I implemented a pseudo-cropping approach that relies on prior knowledge about the object of interest, the paint can. Since the can is primarily white, each frame could be converted to a grayscale image then searched for whiter regions (ie. pixels above a "white-ness" threshold). Since most of the objects in each frame are not white, the majority of the pixels within the frame could be disregarded. The second piece of information the algorithm requires is a general region of interest. For most of the videos, the paint can only move within a certain region in the whole frame. Thus, if the identified "white" pixel is not within the given region, it is likely not part of the paint can (such as in the case of a piece of white paper usually near the top left-hand corner of some videos such as in Figure 2, right). These values are then saved temporarily for each frame and processed to remove outliers using the `rmoutliers` function, because we assume that most of the white values should be close together in space since they constitute a part of the white paint can. The remaining values are then averaged in the x- and y-dimensions to return a theoretical "midpoint" that sort of acts like the paint can's center of mass as a point in space. This single coordinate value representing the paint can's location in the current frame is then saved to a matrix called `points` which saves the paint can's location in all the frames within the video.

All video files in `.mat` format were loaded into MATLAB. Each video was examined frame by frame using a `for` loop. Each frame was converted to grayscale, then each pixel within a frame was examined to determine if it met both of the following criteria: (1) if the pixel is within the region of interest (ie. its x- and y-coordinates fall within the given range of values), and (2) if the grayscale value was at or above the threshold given for the object to be considered "white".

Even though this `for` loop structure remained constant for all the videos, the parameters for the aforementioned criteria had to be adjusted for each video. The process for determining the range of interest also

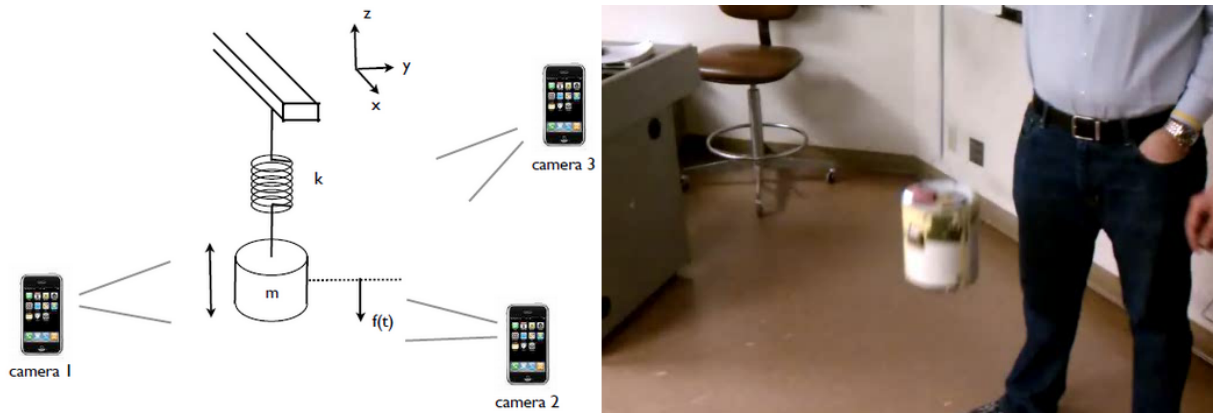

Figure 2: Left: Experimental setup. Three cameras record a moving paint can attached to a string, represented as the spring-mass system. Right: The first frame of camera 1 in case 1.

varied slightly for each video. In the ideal case and for cameras 1 and 2, we know that the paint can only moves in the y-direction (of the image). Therefore, I examined the first frame for these videos and obtained the x-coordinates of the lateral edges of the paint can and added an average of 20 pixels each as a buffer. For most of the other videos, I had to play through all of the frames several times and made note of which frames the paint can may have swung furthest to the left (`xmin`), furthest to the right (`xmax`), and furthest up (`ymin`) and down (`ymax`). I then examined those frames of interest individually using the `imshow` command to obtain the limit coordinates for the `if` statement in the `for` loop. As for determining the grayscale threshold value for what can be considered "white", I would first start with 255 (true white), which often led to the `for` loop terminating quickly because the paint can is rarely ever true white, especially for the entire duration of the recording. This is because the light and shadows cast on the paint can can shift from frame to frame. Whenever this would happen, I would decrease the value by 5 points until the entire loop is able to run successfully.

3.2 Principal Component Analysis

The raw data were processed in Excel in order to generate four snapshot matrices, $\mathbf{M_1}$, $\mathbf{M_2}$, $\mathbf{M_3}$, and $\mathbf{M_4}$. It is termed a "snapshot matrix" because each column of $\mathbf{M}$ is of a video frame, or a snapshot in time. Each matrix is composed of six rows, representing the x- and y-coordinates of the paint can obtained from each of the three cameras, normalized around the mean. The coordinate pairs of the paint can from the three recordings per case were compared so as to sync them in time. Since some recordings are longer than others, these excess points were removed, instead of adding zeros to the shorter vectors. In MATLAB, the `svd` function was used on each of the snapshot matrices, which produced the $\mathbf{U}$, $\mathbf{\Sigma}$ and $\mathbf{V}$ matrices.

The energies of each mode were computed according to Eqn. 2, where F is the Frobenius norm. The principal components, or left singular vectors, in the $\mathbf{U}$ matrix were also plotted to observe how the displacements in each mode varied in relation to each other.

$$energy_N = \frac{\sigma_1^2 + \sigma_2^2 + ... + \sigma_N^2}{\sigma_1^2 + \sigma_2^2 + ... + \sigma_r^2} = \frac{||X_N||_F^2}{||X||_F^2} \tag{2}$$

4 Computational Results

Figure 3 (right) shows the principal component (ie. mode) energy breakdown for each snapshot matrix created from the positional data collected in each case experiment. Out of all first energy mode approximations, case 1 had the highest value, which was expected. Essentially, this means that for case 1, the first principal component alone provides a decent approximation of the movement of the paint can by producing approximately

89% of the total energy in the system. Since we know that the paint can is primarily moving in along a single axis, we can also see that reflected in the clear dominance of one principal component. However, as depicted in Figure 3 (left), the second principal component in case 1 also depicts some oscillatory behavior. This is most likely due to the data obtained in the third camera, which films at an unknown angle. It appears as though the paint can is moving in both the x- and y-direction, with much of the oscillatory motion occurring in the x-direction. Since the SVD algorithm is unaware that each snapshot matrix essentially contains the same information, it considers movement in the x- and y-directions contributed by camera 3 to be significant information.

The case 2 data set, aka the noisy case, shows an oscillatory pattern with very jagged edges. However, the SVD tells us that despite this noise, the algorithm can still identify a relatively strong general movement of the paint can in one axis. This could explain why despite producing the noisiest data sets, the case 2 first mode approximation captures slightly more energy from its system than the first mode in case 3 does. The case 2 energy plot follows a similar shape to the case 1 energy plot in that there appears to be a significant difference in energies between the first and second modes, because of the paint can motion primarily limited to one axis. However, it is clear that retaining the noise does affect the PCA results in that the algorithm thinks that there is significant movement made by the paint can, as opposed to the camera.

In case 3, wherein the paint can moves in a pendulum motion, we observe significant changes in displacement in all three directions. Thus, each recording in case 3 proffers meaningful (ie. less redundant) information, which is reflected in the first four modes containing the bulk of the energy, without one particularly dominating the next.

In case 4, the paint can is made to move in a pendulum motion with rotation in the z-axis. As seen in Figure 3, the first mode energy in this system is the second largest of all the cases. This first mode also appears to dominate the system much more than the second mode, indicating that the motion of the paint can could be favoring a single dimension. The recordings for case 4 do indeed support this observation; the paint can starts off moving in a pendulum-like fashion like in case 3 while rotating in the z-direction, but after a few seconds, we observe a dampening of the pendulum motion, as the paint can proceeds to continue rotating while simply bobbing up and down the vertical axis, similar to cases 1 and 2. Indeed, the energies

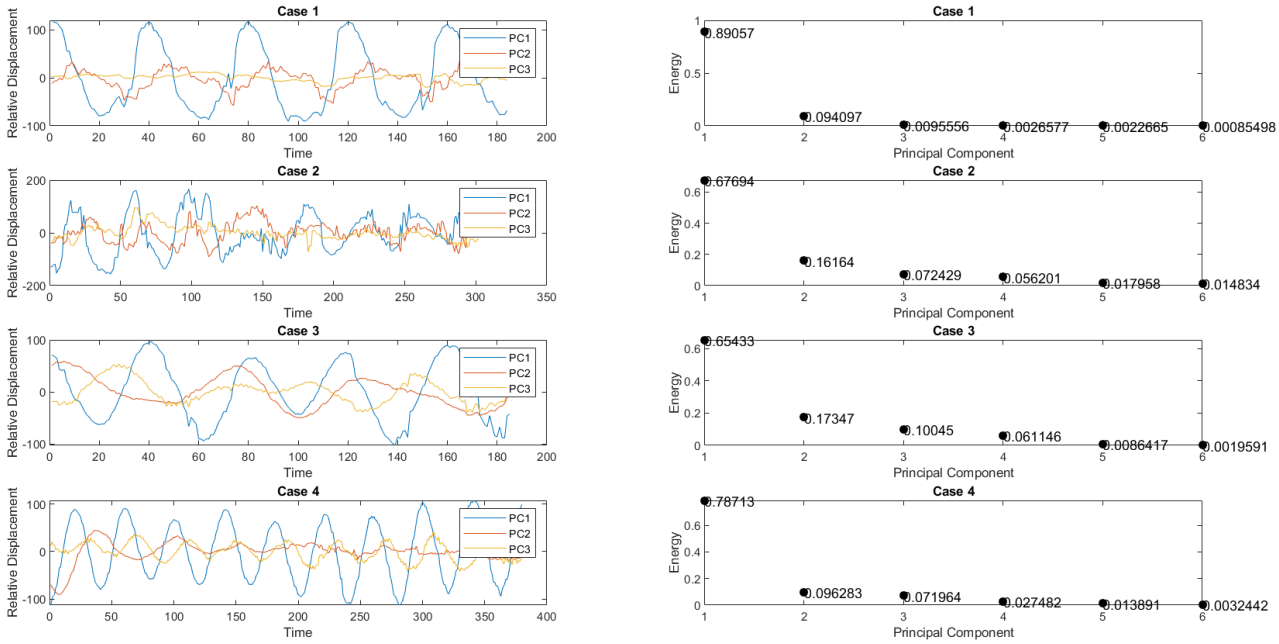

Figure 3: Left column: Paint can displacement shown as the first three principal components of each case. Right column: Energies captured in each principal component for each case system.

in case 4 shows the perfect combination of motions observed in the first three cases. Since the cameras are only able to capture movement in 2D, and the algorithm only identifies whiter areas in a frame, movement in the z-direction does not contribute much to the energy in the system.

Performing PCA on these data sets also allowed us to assess how well our data acquisition algorithm was. Ideally, the algorithm is able to correctly pinpoint the center of the paint can in each frame. This was the aim of the algorithm developed, but it was only programmed to detect "white-ish" pixels, which is subject to bias introduced by changing shadow intensities cast onto the moving paint can. Thus, the center of mass detected by the algorithm may not actually be where the true center of mass is for a given frame. A better method would have been to implement an edge detection method that could identify the perimeter of the paint can, which should not change significantly from frame to frame.

5 Summary and Conclusions

In this report, we explored how PCA can be used to analyze large data sets with lots of variance as well as redundancies. The power of PCA lies in its ability to reduce a multi-dimensional data set to its core components, or principal components. We witnessed how a specific method for performing PCA, the SVD algorithm, is able to extract the dominant motion of the moving paint can, the object of interest, even when noise from camera shake is introduced. Yet, even under ideal filming conditions, the SVD algorithm was also able to capture the multi-dimensional motion of the paint can in cases 3 and 4, depicted by higher energies attributed to the second and third principal components. Furthermore, we were able to gauge the amount of redundancy present in our data sets, considering that in each experimental case, three cameras were filming the same subject. This redundancy presents itself as significantly lower energies captured in typically the fourth, fifth, and sixth modes.

6 Appendix A. MATLAB functions used and brief implementation explanation

- **readmatrix**: Converts values from a .csv file to a double-precision matrix.
- **svd**: Decomposes the given matrix to three matrices: a unitary matrix of left singular vectors (principal components), a diagonal matrix of singular values, and a unitary matrix of right singular vectors.
- **diag**: Extracts the values along the diagonal of a square matrix.
- **num2str**: Converts a numerical value to a string; useful for titles of plots with varying parameters.
- **text**: Used to label points on a plot.
- **rgb2gray**: Converts the given RGB image to grayscale.
- **ind2sub**: Used to convert the index of a given vector to a 2D coordinate pair.
- **rmoutliers**: Removes outlier points in the given data set.

7 Appendix B. MATLAB code

```
clear all; close all; clc;

%%% Name: ROSEMICHELLE MARZAN
%%% Course: AMATH 482
%%% Homework 3, Due 2/21/2019

%%  Data Acquisition

% import all recordings
load('cam1_1.mat'); load('cam2_1.mat'); load('cam3_1.mat')
load('cam1_2.mat'); load('cam2_2.mat'); load('cam3_2.mat')
load('cam1_3.mat'); load('cam2_3.mat'); load('cam3_3.mat')
load('cam1_4.mat'); load('cam2_4.mat'); load('cam3_4.mat')

% % inspect one video at a time
% implay(vidFrames1_4)
% % inspect one frame to get limits for cropping area
% BW = rgb2gray(vidFrames1_4(:,:,:,392));
% imshow(BW)

cam1_1 = trackCan(vidFrames1_1, 250, 300, 400, 0, 480);
cam2_1 = trackCan(vidFrames2_1, 250, 240, 340, 0, 480);
cam3_1 = trackCan(vidFrames3_1, 250, 0, 640, 290, 460);
cam1_2 = trackCan(vidFrames1_2, 250, 290, 640, 0, 480);
cam2_2 = trackCan(vidFrames2_2, 250, 0, 640, 0, 480);
cam3_2 = trackCan(vidFrames3_2, 245, 0, 640, 200, 340);
cam1_3 = trackCan(vidFrames1_3, 250, 300, 400, 0, 480);
cam2_3 = trackCan(vidFrames2_3, 250, 220, 430, 0, 480);
cam3_3 = trackCan(vidFrames3_3, 250, 0, 640, 140, 350);
cam1_4 = trackCan(vidFrames1_4, 245, 330, 430, 215, 480);
cam2_4 = trackCan(vidFrames2_4, 250, 200, 430, 0, 480);
cam3_4 = trackCan(vidFrames3_4, 235, 250, 640, 145, 300);


%% Principal Component Analysis

case1 = readmatrix('case1_A.csv');
[u1,s1,v1] = svd(case1);
case2 = readmatrix('case2_A.csv');
[u2,s2,v2] = svd(case2);
case3 = readmatrix('case3_A.csv');
[u3,s3,v3] = svd(case3);
case4 = readmatrix('case4_A.csv');
[u4,s4,v4] = svd(case4);
sig1 = diag(s1); sig2 = diag(s2); sig3 = diag(s3); sig4 = diag(s4);

figure(1)
% plot the principal components
subplot(4,2,1)
svd_proj_1 = u1'*case1;
plot(1:length(case1(1,:)),svd_proj_1(1:3,:))
legend('PC1', 'PC2', 'PC3')
xlabel('Time')
ylabel('Relative Displacement')
title('Case 1')

subplot(4,2,3)
svd_proj_2 = u2'*case2;
plot(1:length(case2(1,:)),svd_proj_2(1:3,:))
legend('PC1', 'PC2', 'PC3')
xlabel('Time')
ylabel('Relative Displacement')
title('Case 2')
```

```
64
65 subplot(4,2,5)
66 svd_proj_3 = u3'*case3;
67 plot(1:length(case3(1,:)),svd_proj_3(1:3,:))
68 legend('PC1', 'PC2', 'PC3')
69 xlabel('Time')
70 ylabel('Relative Displacement')
71 title('Case 3')
72
73 subplot(4,2,7)
74 svd_proj_4 = u4'*case4;
75 plot(1:length(case4(1,:)),svd_proj_4(1:3,:))
76 legend('PC1', 'PC2', 'PC3')
77 xlabel('Time')
78 ylabel('Relative Displacement')
79 title('Case 4')
80
81 % plot the energies of each mode for each case
82 cases = [sig1, sig2, sig3, sig4];
83 for i = 1:4
84     subplot(4,2,2*i)
85     plot(cases(:,i).^2/sum(cases(:,i).^2),'k*','Linewidth',2)
86     ylabel('Energy')
87     set(gca,'Xtick', [1:6])
88     xlabel('Principal Component')
89     title(['Case ', int2str(i)])
90     for j = 1:6
91         val = cases(j,i)^2/sum(cases(:,i).^2);
92         text(j,val,num2str(val));
93     end
94 end
95
96
97 %% FUNCTIONS
98
99 function points = trackCan(video, threshold, xmin, xmax, ymin, ymax)
100     nFrames = size(video,4);      % total # of frames
101     points = zeros(2,nFrames);    % paint can coordinates for all frames
102     for i = 1:nFrames
103         keep = [];                % collects white areas in frame
104         % convert the frame to grayscale
105         BW = rgb2gray(video(:,:,:,i));
106         for j = 1:length(BW(:))  % inspect each pixel
107             % convert pixel # to (X,Y) coordinate
108             [Y,X] = ind2sub([480,640],j);
109             % is pixel in ROI?
110             if X > xmin && X < xmax && Y > ymin && Y < ymax
111                 % is pixel at or above white-ness threshold?
112                 if BW(Y,X) >= threshold
113                     keep = [keep; X, Y];
114                 end
115             end
116         end
117         B = rmoutliers(keep);      % remove white pixels not part of paint can
118         % get coordinate of approximate center-of-mass
119         x = mean(B(:,1));
120         y = mean(B(:,2));
121         points(:,i) = [x;y];       % save coordinate
122     end
123 end
```