Homework 3: Principal Component Analysis

Joshua Yap
AMATH 482
Winter 2020

February 22, 2020

Abstract

This paper outlines the extraction of usable data from real-world data and performs principal component analysis on it to gain insights into the usefulness of PCA. Interesting results are obtained and discussed.

1 Introduction and Overview

This homework problem requires us to use a set of videos to determine the motion of a paint can on a spring. There are 12 videos in total for 4 cases: ideal case, noisy case, horizontal displacement and horizontal displacement with rotation. We are given 3 videos for each case, each video from a different angle. The problem requires us to extract the position of the paint can in each video from and use it to explore the usefulness of principal component analysis (PCA).

2 Theoretical Background

2.1 Singular Value Decomposition

PCA is an application of singular value decomposition (SVD), a concept from linear algebra. An SVD factorizes a matrix into 3 separate matrices, such that

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

If $\mathbf{A}$ is a full-rank $m \times n$ matrix where $m > n$, then $\hat{\mathbf{U}}$ is an $m \times n$ matrix with orthonormal columns, $\hat{\mathbf{\Sigma}}$ is an $n \times n$ diagonal matrix, and $\mathbf{V}$ is an $n \times n$ unitary matrix. The decomposition highlights the action of matrix $\mathbf{A}$ on a vector: a rotation, a stretching, and a rotation.

2.2 Principal Component Analysis

PCA uses the SVD in data analysis for dimensionality reduction, noise elimination and redundancy detection, among other things. It tells us what axes would be 'best' to project our data on by identifying those along which the data varies the most. The *principal components* are vectors that point in the direction of these axes. There is an associated *singular value* for each principal component that gives the vector its length and tells us how important that principal component is. A larger singular value corresponds to a more important principal component. This means that data varies a lot in this direction and therefore there is a lot of information about the data contained in it. Identifying the most important principal components allows us to keep most of the information while discarding much of the unimportant information, making it easier to determine relationships between variables.

3 Algorithm Implementation and Development

3.1 Data extraction

The first part of the problem required us to determine the position of a paint can in a set of videos. The following outlines how I approached this problem.

Pre-process videos: The videos were converted to grayscale so that I could use a MATLAB function on them.

Initialize tracker: I used the function normxcorr2 in MATLAB, a normalized cross-correlation that compares an image with a template. I input the pixels making up the paint can as the template, and the function determined where on the image best correlated with this set of pixels. Figure 1 gives an idea of what this looks like. It was noticed throughout the video that the tracker drifted off the can, especially when it was rotating, and therefore does not give a perfect representation of the paint can's position in each frame.

Consolidation: Once all the positions were recorded, they were consolidated into appropriate matrices, one for each case described above. Each matrix is $6 \times n$, each row representing the x or y positions in a video, and each column representing a frame in the video.

4 Computational Results

Once the data was gathered and analyzed, I tried a few things with the svd function in MATLAB. First I projected the position of the paint can onto the first principal component and got the results shown in

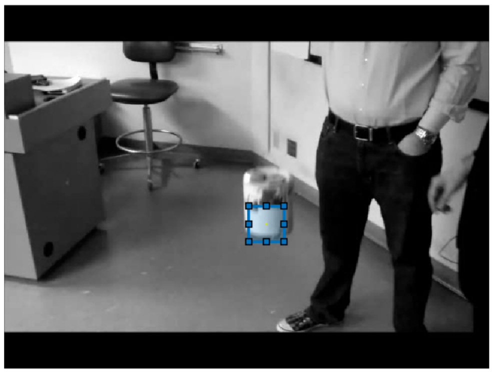


Figure 1: The blue box is superimposed on the first frame of a video, and tracks where the paint can is throughout the video. The position of the can is taken to be the center of this box.

Figure 2.

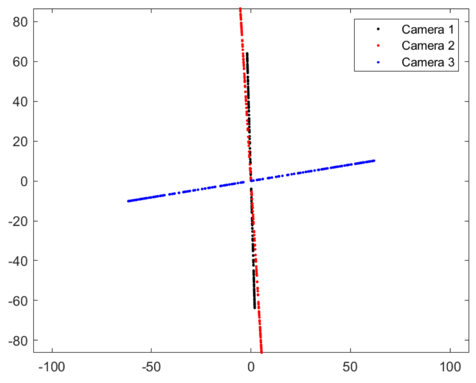
As expected, the first principal component from cameras 1 and 2 are mostly vertical while the first principal component from camera 3 is mostly horizontal. It is surprising to me how similar the ideal and noisy cases look, given how different the videos themselves looked. This highlights the effectiveness of PCA in denoising the signal.

We next look at the magnitude of the singular values. This tells us how much energy is contained in each singular value and therefore how important that corresponding principal component is. Figure 3 illustrates both the magnitudes of singular values and the cumulative fraction of energy that the singular values contain.

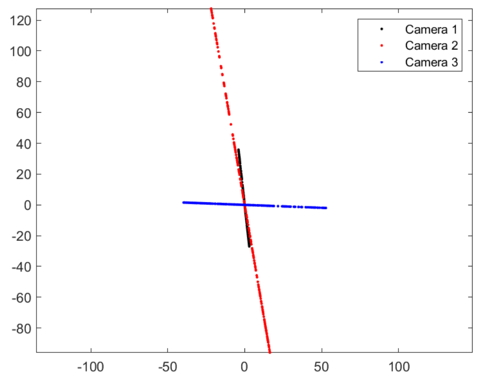
Unsurprisingly, the first singular value for the ideal case contains the most energy out of all the cases. This means that we can keep more information in the first principal component for the ideal case than for the other cases. Unlike examples in lecture, these singular values are fairly large, and none of them become negligible. This may be due to the inaccuracy of the data extraction technique or just because it is real-world data that is messy and noisy.

5 Summary and Conclusions

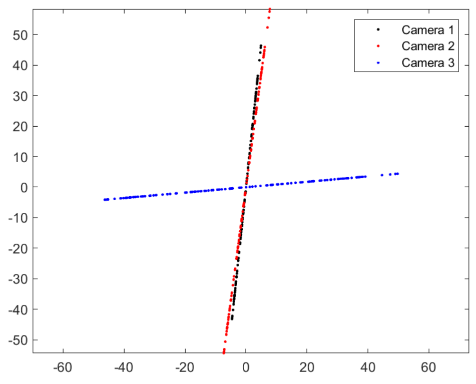
We have converted real-world data into a form that we could perform analysis on, which yielded interesting insights. The most surprising result for me was the similarity in the projection onto the first principle component for the ideal and noisy cases. There are more analyses that can be done using PCA, but the data extraction took up much of the time allocated for this project. In future, I would manually track the paint can to get more accurate results, probably in a shorter time.



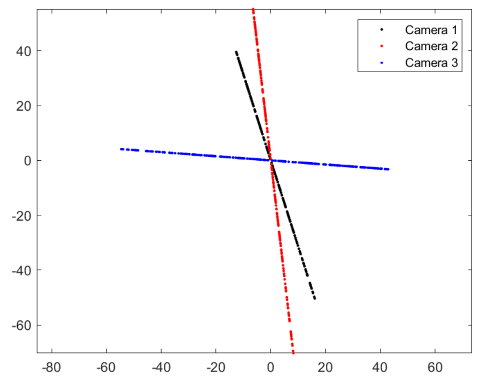
(a) Ideal case



(b) Noisy case

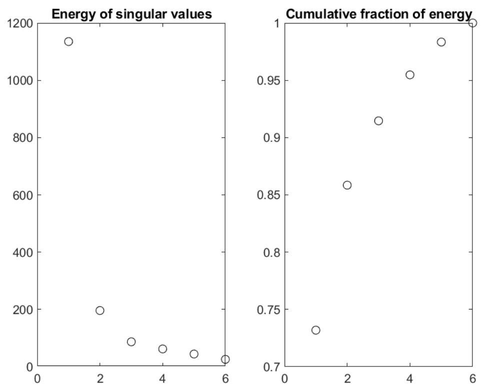


(c) Horizontal displacement

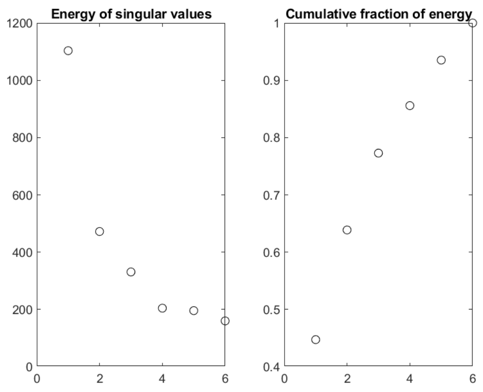


(d) Horizontal displacement and rotation

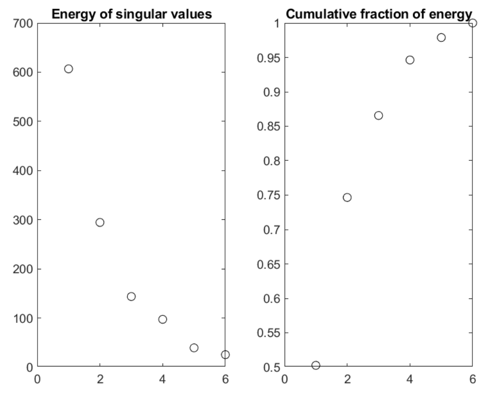
Figure 2: Positions projected onto first principal component.



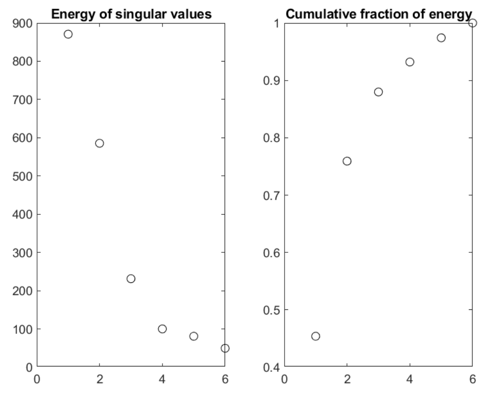
(a) Ideal case



(b) Noisy case



(c) Horizontal displacement



(d) Horizontal displacement and rotation

Figure 3: Energy of singular values for each case.

Appendices

Appendix A

Summary of MATLAB functions used

rgb2gray: Converts RGB image to grayscale.

normxcorr2: Finds a region in a given image that correlates most strongly with the pixel pattern in a given template.

svd: Performs singular value decomposition on a matrix.

Appendix B

MATLAB code

```
close all; clear; clc
myFiles = dir('vids\*.mat');
test1 = zeros(6,181);
test2 = zeros(6,241);
test3 = zeros(6,151);
test4 = zeros(6,281);
count = 1;
for j = 1:length(myFiles)
    file = strcat('vids\',myFiles(j).name);
    pos_vec = mass_position(file);
    if mod(j,4) == 1
        test1(count:count+1,:) = pos_vec';
    elseif mod(j,4) == 2
        test2(count:count+1,:) = pos_vec';
    elseif mod(j,4) == 3
        test3(count:count+1,:) = pos_vec';
    else
        test4(count:count+1,:) = pos_vec';
    end
    if mod(j,4) == 0
        count = count+2;
    end
end

save('test1.mat','test1')
save('test2.mat','test2')
save('test3.mat','test3')
save('test4.mat','test4')

% subtract mean
load('test1.mat')
load('test2.mat')
load('test3.mat')
load('test4.mat')
test = test4;
[m,n] = size(test);
mn = mean(test,2);
X1 = test - repmat(mn,1,n);

[U,S,V] = svd(X1,'econ');
% Compute and plot rank-1 approximations
X_rank1 = U(:,1)*S(1,1)*V(:,1).';
figure()
plot(X_rank1(1,:),X_rank1(2,:),'k.');
axis equal
hold on
plot(X_rank1(3,:),X_rank1(4,:),'r.');
plot(X_rank1(5,:),X_rank1(6,:),'b.');
legend('Camera 1','Camera 2','Camera 3')
hold off
```

```
% Plot energy
figure()
subplot(1,2,1)
sig = diag(S);
plot(sig,'ko')
title('Energy of singular values')
subplot(1,2,2)
sum_energy = cumsum(sig)/sum(sig);
plot(sum_energy,'ko')
title('Cumulative fraction of energy')


function [pos_vec] = mass_position(filename)
% Takes in a .mat video file and returns a matrix of the
% positions of the mass in each frame. Column 1 returns
% x-position and column 2 returns y_position.
varName = strcat('vidFrames',filename(9:11));
data = load(filename,varName);
vidFrames = data.(varName);

% load('cam2_4.mat');
% load('cam3_4.mat');
numFrames = size(vidFrames,4);
vid_gs = zeros(size(vidFrames,[1 2 4]));
% Convert to grayscale
for j = 1:numFrames
    X = vidFrames(:,:,:,j);
    X = rgb2gray(X);
    vid_gs(:,:,j) = X;
end
%imtool(vid_gs(:,:,1),[])
dl = 46;
if strcmp('vidFrames1_1',varName)
```

```
    rstart = 266;

    cstart = 323;

    frame_start = 30;

    n_frames = 180;

elseif strcmp('vidFrames1_2',varName)

    rstart = 317;

    cstart = 317;

    frame_start = 13;

    n_frames = 240;

elseif strcmp('vidFrames1_3',varName)

    rstart = 309;

    cstart = 321;

    frame_start = 59;

    n_frames = 150;

elseif strcmp('vidFrames1_4',varName)

    rstart = 287;

    cstart = 373;

    frame_start = 72;

    n_frames = 280;

elseif strcmp('vidFrames2_1',varName)

    rstart = 297;

    cstart = 267;

    frame_start = 40;

    n_frames = 180;

elseif strcmp('vidFrames2_2',varName)

    rstart = 329;

    cstart = 286;

    dl = 2*dl;

    frame_start = 38;

    n_frames = 240;

elseif strcmp('vidFrames2_3',varName)

    rstart = 300;

    cstart = 221;
    dl = 2*dl;
    frame_start = 86;
    n_frames = 150;
elseif strcmp('vidFrames2_4',varName)
    rstart = 262;
    cstart = 231;
    dl = 1.5*dl;
    frame_start = 79;
    n_frames = 280;
elseif strcmp('vidFrames3_1',varName)
    rstart = 268;
    cstart = 332;
    frame_start = 29;
    n_frames = 180;
elseif strcmp('vidFrames3_2',varName)
    rstart = 249;
    cstart = 350;
    frame_start = 18;
    n_frames = 240;
    % cut out last few frames
elseif strcmp('vidFrames3_3',varName)
    rstart = 224;
    cstart = 365;
    frame_start = 50;
    n_frames = 150;
    % cut out last few frames
elseif strcmp('vidFrames3_4',varName)
    rstart = 180;
    cstart = 345;
    dl = 2*dl;
    frame_start = 71;
    n_frames = 280;
end
```

```
can = vid_gs(rstart:rstart+dl,cstart:cstart+dl);
pos_vec = zeros(numFrames,2);

for j = 1:numFrames
    framej = vid_gs(:,:,j);
    c = normxcorr2(can,framej);
    % figure, surf(c), shading flat

    if j == 1
        [ypeak, xpeak] = find(c==max(c(:)));
    else
        c_red = c(ypeak-dl:ypeak+dl,xpeak-dl:xpeak+dl);
        max_red = max(c_red(:));
        [ypeak, xpeak] = find(c==max_red);
    end
    yoffSet = ypeak-size(can,1);
    xoffSet = xpeak-size(can,2);
%     imshow(framej,[]);
%     hold on
%     drawrectangle('Position',[xoffSet+1, yoffSet+1, dl+1, dl+1]);
%     scatter(xoffSet+1+(dl+1)/2,yoffSet+1+(dl+1)/2,'y.')
%     drawnow
%     hold off
    pos_vec(j,:) = [xoffSet+1+(dl+1)/2,yoffSet+1+(dl+1)/2];
    can = framej(yoffSet+1:yoffSet+dl+2,xoffSet+1:xoffSet+dl+2);
end

pos_vec = pos_vec(frame_start:frame_start+n_frames,:);
end
```