Principal Component Analysis on Noise Reduction and Principal Signal Extraction

Jiaqi Zhang
University of Washignton
zjq95@uw.edu

Abstract

This project applies Principal Component Analysis (PCA) on the movement of a mass-spring system. It illustrates various aspects of the PCA and its practical usefulness and the effects of noise on the PCA algorithms. This report focuses on the mathematics behind PCA, as well as how and why to apply this technique.

I. Introduction and Overview

Principle Component Analysis (PCA) is widely used in modern data analysis. The purpose of this homework is to demonstrate the usage of PCA on principal signal extraction. We use PCA to remove redundancy and to reduce the data to a lower dimension to reveal the simplified structure.

We will begin with a mathematical explanation of PCA and continue by showing how to implement the PCA algorithm to a real life example. We will discuss the computational results in detail as well as the observations behind the technique.

II. Theoretical Background

The Covariance Matrix

The main goals that we want to achieve with PCA is:

1) Removing redundancy,
2) Identify principal signals.

According to Professor Nathan Kutz's 'Introduction to PCA' [1], an easy way to identify redundant data is through the covariance between the datasets. Covariance measures the statistical dependence/independence between two variables. Strongly dependent variables are considered as redundant observations. We potentially have a several sets of data that need to be correlated and checked for redundancy, therefore, the appropriate *covariance matrix* is of the form:

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

Matrix *X* contains the experimental data. $X \in \mathbb{C}^{m \times n}$ where *m* is the number of data sets, *n* is the number of data points taken within each set. C_X is a symmetric *m x m* matrix. Its diagonal represents the variance of *X* while off-diagonals are the covariances between measurement types. Therefore C_X captures the redundancy between all possible data pairs in matrix *X*. Large diagonals corresponds to redundancy while small diagonals indicates statistical independence. Note that the large variances typically represents the principal signals because large variance suggests great changes in the variable.

Diagonalization Using Singular Value Decomposition

Singular Value Decomposition (SVD) makes it possible to say that every matrix is diagonal, under the condition that proper bases for domain and range are used[1]. The full SVD of *A* has the form:

$$A = U\Sigma V^* \tag{2}$$

where

$U \in \mathbb{C}^{m \times n}$ is unitary (3a)

$V \in \mathbb{C}^{n \times n}$ is unitary (3b)

$\Sigma \in \mathbb{R}^{m \times n}$ is diagonal (3c)

In order to find this proper bases, we define a new matrix:

$$Y = \frac{1}{\sqrt{n-1}} X^T \tag{4}$$

By analyzing $Y^T Y$, we can show that:

$$Y^T Y = C_X \tag{5}$$

$n-1$ is just a constant of normalization[2]. By construction, $Y^T Y$ equals to the covariance matrix of X. Therefore the eigenvectors of C_x are the principal components that we are looking for. If we calculate the SVD of *Y*, the eigenvectors of C_X is stored in matrix *V*. The singulr values on the diagonal of Σ is correlated to the eigenvectors. Therefore, by looking at the entries of Σ, we can find the principal components of interest. The number of large singular values of *Y* is equal to the number of principal dimensions of interest.

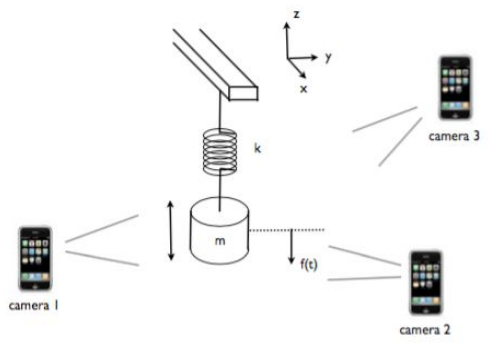


Figure 1. A Simple Mass-Spring system. Mass is suspended by a spring, whose other end is fixed [1].

III. Algorithm Implementation

The model we consider is a simple mass-spring system as illustrated in figure 1. For each case, we set up 3 cameras, each from the left, middle and right to record the movement. The cameras use different coordinate systems (one of them tilted) but record the same movement. It is our goal to extract the simple solution of the motion from the data sets.

Data Collection and Ordering

We denote the data from the 3 cameras with subscripts a, b and c. Each camera produces a two dimensional representation of the motion and the data collected is represented by:

camera 1: (x_a, y_a) (6a)

camera 2: (x_b, y_b) (6b)

camera 3: (x_c, y_c) (6c)

Where each (x_j, y_j) is the data collected with reference to the coordinate system of the camera. The length of (x_j, y_j) depends on the rate of collection and the time span when the motion is recorded. The data collected are gathered into a single matrix:

$$X = \left[x_a, y_a, x_b, y_b, x_c, y_c\right]^T \tag{7}$$

$X \in \mathbb{R}^{m \times n}$, where *m* is the number of measurement types and *n* is the number of data points. In our case, *m = 6*.

Diagonalization with SVD

After obtaining the data matrix *X*, the rest can be simply done by Matlab. First of all, in order to normalize our datasets, we subtract the mean of each row of *X* from itselves. Then we perform SVD on the normalized matrix and obtain the diagonal matrix Σ. The energy of each singular values can be calculated by:

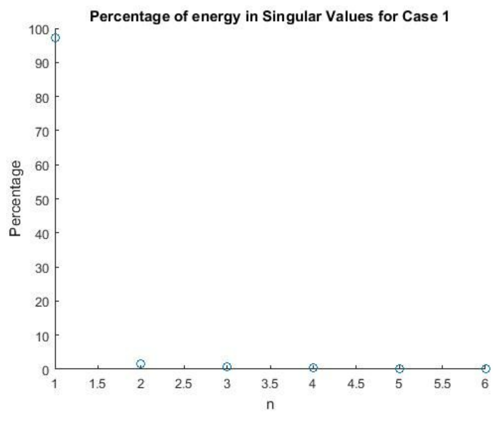
$$energy = \frac{diag(\Sigma)^2}{sum(diag(\Sigma)^2)} \tag{8}$$

IV. Computational Result

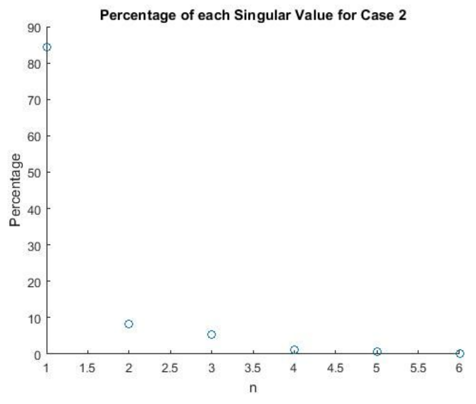
In case 1, the mass is released from the center and performs simple harmonic motion. Our observation is that the energy of the first singular value is significantly larger than the other 5 and it is above 95% (Figure 3a). This indicates that PCA has identified redundant data and the motion is only one dimensional. The fact that the energy of first singular value is not 100% one dimensional is due to artificial influence.

In case 2, the object is also released from the center but the cameras shake during recording. We can see that the majority of energy is still stored in the first singular value (figure 3b).

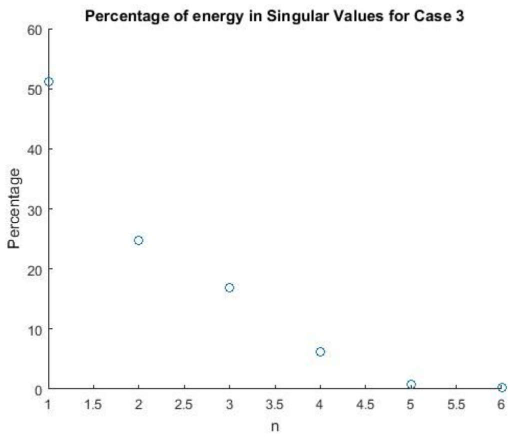
In case 3, the object is release off center. So there is both motion from top to bottom and left to right. Theoretically, the movement is constrained in the *y*-*z* plane. However PCA shows that the first 4 singular values all reserve energy (figure 3c).



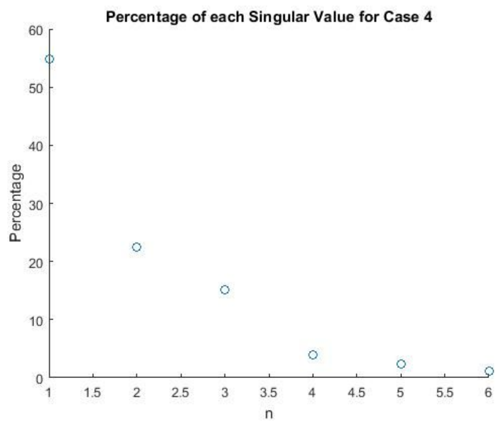
(a)



(b)

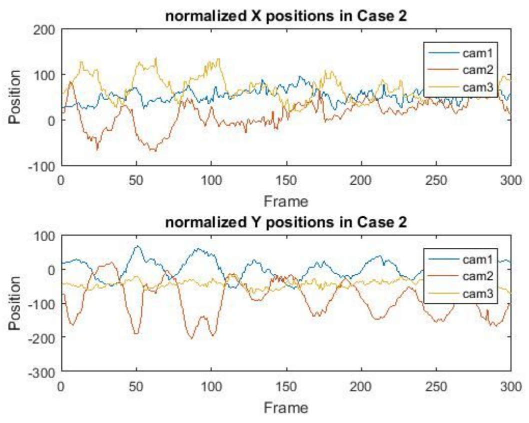


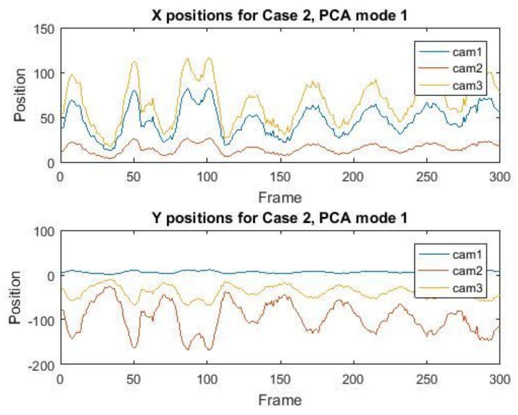
(c)



(d)

Figure 3. Percentage of energy in each cases.





(a) (b)

Figure 4. X and Y positions of the mass in Case 2. Before(a) and after(b) reconstruction using PCA.

We believe that this is because the mass we are tracking is big, making it hard to focus on exactly the same spot. Plus that possibly the initial release does produce movement on the *x-y* plane, data indicating the 3rd dimension is inevitable. Generally, PCA still indicates a 2D motion by showing that the first 2 singular value energies maintain 75% of the system.

In case 4, the mass is released off center and it is rotating. Looking at the result from PCA (figure 3d), we can see that the majority of energy is stored in the first 3 singular values while the other 3 are close to 0. This explicitly shows that the motion is 3D.

Additionally, PCA is able to reduce noise in the recorded data. We reconstruct the data matrix using the first eigenvector in case 2. Figure 4 shows that the recorded data of the motion becomes much smoother and more regular after reconstruction.

V. Summary and Conclusion

In this report, we have shown how to use PCA to examine the variances associated with the principle components. When we observe large variances between the first *k* components, we can conclude that the motion dynamics occur in the first *k* dimensions. PCA is able to extract the principal dynamics from high dimensional, redundant data. It is also able to filter out the noise and reveal the hidden motion.

VI. Bibliorgraphy

[1] N.J.Kutz. “*Data-Driven Modeling & Scientific Computation: Methods for Complex Systems & Big Data*” 1st Edition. 15 September, 2013.
[2] J. Shlens. ”*Tutorial on Principal Component Analysis”*. 10 December, 2005.

Appendix A MATLAB functions used & brief implementation explanation

1) **‘load’** `load('cam1_1.mat');`

Load the video into Matlab as a 4D matrix *M*. *M*’s first and second parameters are the x and y coordinates, the third one are the R, G, B values for color and the fourth one indicates the time spot.

2) **‘imshow’**
```
for k = 1: numFrames
    imshow(vidFrames(:,:,:,k));
end
```
Displays the image. Using it in a forloop displays the video.

3) `‘rgb2gray’  gray = rgb2gray(vidFrames(:,:,:,k));`

Converts RGB image to grayscale.

4) **‘kmeans’** `[in,C]=kmeans([row, col], 2);`

Performs *k*-means clustering to partition the observations of the *n*-by-*p* data matrix X into k clusters, and returns an *n*-by-1 vector (idx) containing cluster indices of each observation. Rows of X correspond to points and columns correspond to variables.

5) **‘svd’** `[U1,S1,V1]=svd(X1'./sqrt(n-1));`

Performs a singular value decomposition of matrix A, such that A = U*S*V'.

Appendix B MATLAB codes

Case 1

%Part 1. Constructing the X matrix for Case 1.

```
%% declare X
X1 = zeros(6, 356);
%% load image
load('cam1_1.mat');
load('cam2_1.mat');
load('cam3_1.mat');
%% Cam1_1
varName = 'vidFrames1_1';
vidFrames = evalin('base', varName);
%% play oariginal video
numFrames = size(vidFrames, 4);
for k = 1: numFrames
imshow(vidFrames(:,:,:,k));
end
%% get grayscale image
numFrames = size(vidFrames, 4);
for k = 1:numFrames
      gray = rgb2gray(vidFrames(:,:,:,k));
      Q1_1(:,:,k) = gray;
      s = size(Q1_1);
    for i = 1:s(1)
        for j = 1:s(2)
            if (Q1_1(i, j, k) < 250)
            Q1_1(i, j, k) = 0;
            else
            Q1_1(i, j, k) = 255;
            enda
        end
    end
end

%% play grayscale image
numFrames = size(vidFrames1_1, 4);
for k = 1:numFrames
      imshow(Q1_1(:,:, k));
end
%% play gray scale sub image
Q1_1sub = Q1_1(200:end,250:500,:);
numFrames = size(vidFrames1_1, 4);
for k = 1:numFrames
      imshow(Q1_1sub(:,:, k));
end
%% xa, ya
for k = 1: numFrames
      [row, col] = find(Q1_1sub(:,:,k) == 255);
      [in,C]=kmeans([row, col], 2);
    %track can top.
    if (C(1, 1) < C(2, 1)) %compare y coordinates.
      X1(1, k) = C(1, 2); %Xa at frame k
      X1(2, k) = C(1, 1); %Ya at frame k
    else
      X1(1, k) = C(2, 2);
      X1(2, k) = C(2, 1);
    end
end

%% Cam2_1
varName = 'vidFrames2_1';
vidFrames = evalin('base', varName);
%% play original video
numFrames = size(vidFrames, 4);
for k = 1: numFrames
imshow(vidFrames(:,:,:,k));
end
%% get grayscale image
numFrames = size(vidFrames, 4);
for k = 1:numFrames
      gray = rgb2gray(vidFrames(:,:,:,k));
      Q2_1(:,:,k) = gray;
      s = size(Q2_1);
    for i = 1:s(1)
        for j = 1:s(2)
            if (Q2_1(i, j, k) < 250)
            Q2_1(i, j, k) = 0;
            else
            Q2_1(i, j, k) = 255;
            end
        end
    end
end

%% play grayscale image
for k = 1:numFrames
      imshow(Q2_1(:,:, k));
end
%% play gray scale sub image
Q2_1sub = Q2_1(50:end-50, 200:450,:);
for k = 1:numFrames
      imshow(Q2_1sub(:,:, k));
end
%% xb, yb
frame = numFrames-10;
for k = 1: frame
      ind = k+10;
      [row, col] = find(Q2_1sub(:,:,ind) == 255);% camera 2 starts 10 frames
early than the other two.
      [in,C]=kmeans([row, col], 2);
    %track can top.
    if (C(1, 1) < C(2, 1)) %compare y coordinates
      X1(3, k) = C(1, 2); %Xb at frame k
      X1(4, k) = C(1, 1); %Yb at frame k
    else
      X1(3, k) = C(2, 2);
      X1(4, k) = C(2, 1);
    end
end

%% Cam3_1
varName = 'vidFrames3_1';
vidFrames = evalin('base', varName);
%% play original video
numFrames = size(vidFrames, 4);
for k = 1: numFrames
imshow(vidFrames(:,:,:,k));
end
%% get grayscale image
numFrames = size(vidFrames, 4);
for k = 1:numFrames
      gray = rgb2gray(vidFrames(:,:,:,k));
      Q3_1(:,:,k) = gray;
      s = size(Q3_1);
    for i = 1:s(1)
        for j = 1:s(2)
            if (Q3_1(i, j, k) < 240)
            Q3_1(i, j, k) = 0;
            else
            Q3_1(i, j, k) = 255;
            end
        end
    end
end

%% play grayscale image
for k = 1:numFrames
      imshow(Q3_1(:,:, k));
end
%% play gray scale sub image
Q3_1sub = Q3_1(240:330, 270:470,:);
for k = 1:numFrames
      imshow(Q3_1sub(:,:, k));
end
%% xc, yc
for k = 1: numFrames
      [row, col] = find(Q3_1sub(:,:,k) == 255);
      [in,C]=kmeans([row, col], 2);
    %track can top.
    if (C(1, 2) < C(2, 2)) %compare x coordinates
      X1(5, k) = C(1, 2); %Xc at frame k
```

```
        X1(6, k) = C(1, 1); %Yc at frame k
    else
        X1(5, k) = C(2, 2);
        X1(6, k) = C(2, 1);
    end
end

%% Save matrix X
save X1.mat X1;
```

%Part 2. PCA on X.

```
%% X1
X1 = importdata('X1.mat');
X1 = X1(:,1:150);
%%
[m,n]=size(X1); % compute data size
mn=mean(X1,2); % compute mean for each row
X1=X1-repmat(mn,1,n); % subtract mean
[U1,S1,V1]=svd(X1'./sqrt(n-1)); % perform the SVD
lambda=diag(S1).^2; % produce diagonal variances

%% Plot energy

lamda1 = diag(S1).^2;
lamda1 = 100*lamda1/sum(lamda1);
scatter(1:6,lamda1);
title('Percentage of energy in Singular Values for Case 1');
ylabel('Percentage');
xlabel('n');


%% Orginal x, y plot
subplot(2, 1, 1);
plot(X1_sub(1,:));
hold on;
plot(X1_sub(3,:));
plot(X1_sub(5,:));
legend('cam1', 'cam2', 'cam3');
title('normalized X positions in Case 1');
xlabel('Frame');
ylabel('Position');

subplot(2, 1, 2);
plot(X1_sub(2,:));
hold on;
plot(X1_sub(4,:));
plot(X1_sub(6,:));
title('normalized Y positions in Case 1');
legend('cam1', 'cam2', 'cam3');
xlabel('Frame');
```

```
ylabel('Position');

%% pca
j=1;
S_rank1 = S1;
S_rank1(j+1:end,j+1:end) = 0;
X1_new = U1*S_rank1*V1';
subplot(2, 1, 1);
plot(X1_new(1, :));
hold on;
plot(X1_new(3, :));
plot(X1_new(5, :));
title('X positions for Case 1, PCA mode 1');
legend('cam1', 'cam2', 'cam3');
xlabel('Frame');
ylabel('Position');

subplot(2, 1, 2);
plot(X1_new(2, :));
hold on;
plot(X1_new(4, :));
plot(X1_new(6, :));
title('Y positions for Case 1, PCA mode 1');
legend('cam1', 'cam2', 'cam3');
xlabel('Frame');
```