AMATH 482 Homework 3

Jiatian Xie

Feb 28, 2020

Abstract

This paper aims to explore the traces of a spring-mass system. We are going to use the PCA(Principal Component Analysis) method on the different data sets that were from the experiment. Also, this paper showed how to use PCA method to reduce the dimensionality of a system.

1 Introduction and Overview

When collecting the data from a system, it is likely that the data is redundant and contains various noises. In order to cut back on the redundancy, we are supposed to reduce the dimensionality of the system and make it easier for us to track the dynamics of the system. One way I would like to show in this paper is the Singular Value Decomposition(SVD) and Principal Component Analysis(PCA).

We have several data sets from a spring-mass system which record the four cases of a paint can's oscillating from a spring. It also used three different camera angles for each case. Comparing the track from each camera of each case provided the path of the paint can. The four cases are as follows:

1. Ideal Case
2. Noisy Case
3. Horizontal Displacement
4. Horizontal Displacement and Rotation

2 Theoretical Background

2.1 Singular Value Decomposition(SVD)

The first concept used in this problem is the Singular Value Decomposition(SVD). The SVD is a matrix factorization of a given matrix A and it comes from the eigenvalue decomposition. If A is an $n \times m$ matrix, then we may write A as a product of three factors:

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

where U is an orthogonal $n \times n$ matrix, V is an orthogonal $m \times m$ matrix, V^* is the transpose of V, and Σ is an $n \times m$ matrix that has all zeros except for its diagonal entries, which are

non-negative real numbers. If σ_{ij} is the i, j entry of Σ, then $\sigma_{ij} = 0$ unless $i = j$ and $\sigma_{ii} = \sigma_i \geq 0$. The σ_i are the "singular values" and the columns of u and v are respectively the right and left singular vectors. some properties of singular value are as followed:

- The singular values are uniquely determined and are always non-negative real numbers.
- The singular values are always ordered from largest to smallest along the diagonal of Σ.
- The number of nonzero singular values in the rank of A.

2.2 Principal Component Analysis(PCA)

The Principal Component Analysis(PCA) provides us a new way to SVD application. The SVD tells us the dimension and the direction of the principal components of th matrix A. We can get information in each dimension through the each singular value.

Given some matrix X whose rows are different measurements, we can compute all the variances and covariances between the rows of X with one matrix multiplication:

$$C_X = \frac{1}{n-1} X X^T = A A^T \tag{2}$$

From the SVD, we have that

$$C_X = A A^T = U \Sigma^2 U^T \tag{3}$$

Where U is the matrix of left-singular vectors and Σ is the diagonal matrix of singular values. So the eigenvalues of the covariance matrix are the squares of singular values. By using a change of basis, we multiply by $U^-1 = U^T$. The data in the new coordinates is

$$Y = U^T X \tag{4}$$

The covariance of Y is

$$C_Y = \Sigma^2$$

Since the off-diagonal elements of Σ are zero, it follows that the variables in Y are uncorrelated.

3 Algorithm Implementation and Development

To begin with this problem, I firstly watched all twelve videos and wanted to figure out a way to ensure the position of bright spot that helped to track the system. I considered a number of methods to get the position of the can over time. Some issues were aroused because of rotation or the change of angles.

Therefore, I choose to find the maximum value of brightness and then search for the xy coordinates with such brightness. However, since there are some noises like other 'bright' thing, I need to determine the approximate xy range of the paint can. The way I used was:

1. `X = rgb2gray(vidFrames1(:,:,:,j));`

2. determine the range of x and y by looking at each frame.

3. find the largest value of brightness and use `ind2sub` to find all possible xy coordinates.

4. filter out xy value that are out of the range.

5. record the mean of the remaining xy values as the final xy coordinates for each frame.

I did meet some troubles in this process. One thing I wanted to mention here was I could not find one xy coordinates lying in the range, so I just used `mode(x)` to record the most frequent xy values.

After doing the same process for all the videos taken by three cameras for the ideal case, I found that the time is different. I need to put them together in order to do PCA later. Therefore, I just find the shortest one and cut the other videos, to keep them the same length.

Then, I combine all the X and Y coordinates that we found before into a matrix and the matrix has the dimension 6 by n where n is the length of the coordinates. Then, I computed the mean of each row and subtract the mean to get the matrix. Now, I can find the SVD of the matrix. After determining the singular values, we can extract the principal components by `Y = U*X` and get the projection of the matrix into the principal component basis.

We repeat this whole process for all four test cases - ideal, noisy, horizontal displacement, and horizontal displacement and rotation and adjust the parameters like the range of x,y values to find what works best.

4 Computational Results

Figure 1 is the ideal case of the movement of paint can. It shows the principal component corresponding to the dominant singular value, which represents the oscillation of the can. Moreover, we can see the clear trajectory of the position through each frame.

Figure 2 is the noisy case which contained some noise and becomes more ambiguous. The noise was caused by the camera shake and our system is still one-dimensional. As we seen in Figure 2, the principal component capture only a little data of the system. Comparing to the ideal case, it is hard to tell us most information about the system.

Figure 3 is the horizontal displacement case, giving the can some pendulum motion and some simple oscillation. Therefore, our system is two-dimensional. As we see, the figure illustrates the movement of the paint can. However, comparing to the ideal case, PCA shows that the movement is mainly in x direction with horizontal displacement.

Figure 4 is the horizontal displacement and rotation. Based on the figure, we can get that the rotation is in x direction. Comparing with case 3, we may also know that there is a rotation in this movement. Also, comparing to the ideal case, PCA seems not to extract the information about the rotation, but the figure does track the most clear principal component.

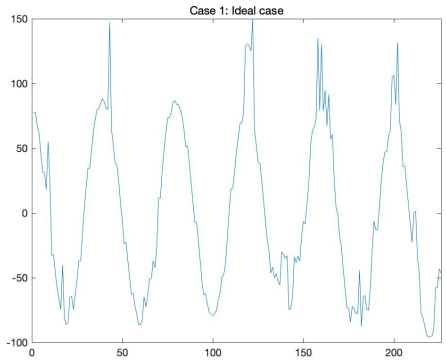

Figure 1: Here is a figure of the principle component of movement for Case 1: Ideal Case

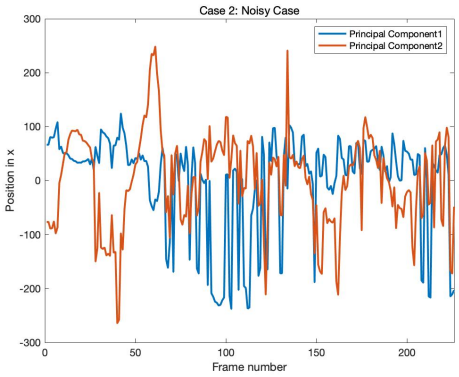

Figure 2: Here is a figure of the principle component of movement for Case 2: Noisy Case

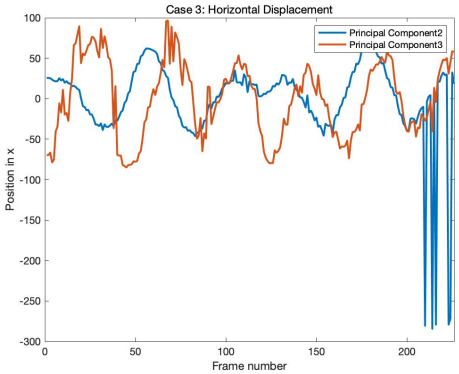

Figure 3: Here is a figure of the principle component of movement for Case 3: Horizontal Displacement

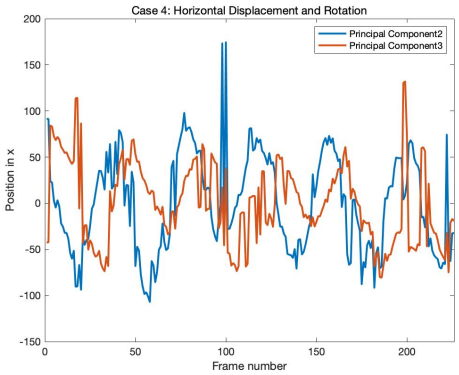

Figure 4: Here is a figure of the principle component of movement for Case 4: Horizontal Displacement and Rotation

5 Summary and Conclusions

PCA can do a meaningful job of reducing the dimension of a system and find out the most significant components from several data sets recording the same thing. Through this project, we collected the positions of a paint can in different videos and performed PCA in four cases. We found that we were able to see the clearest data in Case 1 - Ideal Case, which also illustrated that we collected the most correct data in this case. And for Case2 - Noisy Case, we also gained some basic data but much less than the first case. Case 4 is complex movement so that my PCA could not capture the rotation. It failed may be because of our data extraction method. It was hard for me to find the position of can by using a comparably perfect method.

This project is amazing for me since I learned how to apply PCA to a complex system and extract the important data from it.

Appendix A MATLAB Functions

unordered list:

- `rgb2gray` converts the truecolor image RGB to the grayscale image.
- `[U,S,V] = svd(A)` performs a singular value decomposition of matrix A, such that `A = U*S*V'`.
- `k = find(X)` returns a vector containing the linear indices of each nonzero element in array X.
- `[row,col] = ind2sub(sz,ind)` returns the arrays row and col containing the equivalent row and column subscripts corresponding to the linear indices ind for a matrix of size sz. Here sz is a vector with two elements, where sz(1) specifies the number of rows and sz(2) specifies the number of columns.

Appendix B MATLAB Code for Ideal Case(1)

I only included my code dealing with the ideal case. The remaining three cases are very similar to this.

Appendix C MATLAB Code for Ideal Case(2)

```
%% Ideal Case
[height11, width11, rgb11, num_frames1_1] = size(vidFrames1_1);
X11 = []; Y11 = [];
for j = 1:num_frames1_1
    X = rgb2gray(vidFrames1_1(:,:,:,j));
    yrange = [300,400];
    xrange = [200,450];
    bright = max(X,[],'all');
    [x,y] = ind2sub(size(X), find(X==bright));
    filter_x = x(find(x>xrange(1) & x<xrange(2)));
    filter_y = y(find(y>yrange(1) & y<yrange(2)));
    X11 = [X11 mean(round((filter_x)))];
    Y11 = [Y11 mean(round((filter_y)))];
end
[height12, width12, rgb12, num_frames1_2] = size(vidFrames2_1);
X12 = [];Y12 = [];
for j = 1:num_frames1_2
    X = rgb2gray(vidFrames2_1(:,:,:,j));
    xrange = [100,375];
    yrange = [250,350];
    bright = max(X,[],'all');
    [x,y] = ind2sub(size(X), find(X==bright));
    filter_x = x(find(x>xrange(1) & x<xrange(2)));
    filter_y = y(find(y>yrange(1) & y<yrange(2)));
    X12 = [X12 mean(round((filter_x)))];
    Y12 = [Y12 mean(round((filter_y)))];
end
[height13, width13, rgb13, num_frames1_3] = size(vidFrames3_1);
X13 = [];Y13 = [];
for j = 1:num_frames1_3
    X = rgb2gray(vidFrames3_1(:,:,:,j));
    xrange = [200,350];
    yrange = [250,500];
    bright = max(X,[],'all');
    [x,y] = ind2sub(size(X), find(X==bright));
    filter_x = x(find(x>xrange(1) & x<xrange(2)));
    filter_y = y(find(y>yrange(1) & y<yrange(2)));
    if isempty(filter_x)
        x_last = mode(x);
    else
        x_last = mean(round((filter_x)));
    end
    if isempty(filter_y)
        y_last = mode(x);
    else
        y_last = mean(round((filter_y)));
    end
    X13 = [X13 x_last];
    Y13 = [Y13 y_last];
end
```

```
% combine the data from three cameras
X_min1 = min([length(X11) length(X12) length(X13)]);
X1 = [X11(1:X_min1);Y11(1:X_min1);X12(1:X_min1);Y12(1:X_min1);X13(1:X_min1);Y13(1:X_min1)];
[row,col] = size(X1);
means = mean(X1.').';
X1 = X1-means;
[U1,S1,V1] = svd(X1,'econ');
Y = U1'*X1;
principal_component = Y(1:2,:);
figure(1)
names = [];
for k = 1:2
    plot(principal_component(k,:),'linewidth',2),hold on;
    names = [names;strcat('Principal Component',num2str(k))];
    xlabel('Frame number')
    ylabel('Position in x')
end
xlim([0,length(principal_component)])
title('Case 1: Ideal case')
legend(names);
```