Computational Methods for Data Analysis

Homework 3

Due on March 16, 2020 at 5:00pm

Professor J. Nathan Kutz

Section A

Chandan Sharma Subedi

Abstract

The paper explores the idea Principal Component Analysis through singular value decomposition (SVD). Specifically, the paper examines how the dimension of the feature space and the noise level in the data affect the result of PCA. 4 datasets, representing the motion of mass-string system, were created with varying noise and feature space dimensions. The concept of dimensionality reduction through the diagonalization of the covariance matrix is discussed.

1 Introduction and Overview

How can we reduce the dimensions of a large dataset so as to identify the underlying low-dimensional dynamics or structure?– a fundamental question in the realm of data analysis. The process of reducing the dimensions of a data is, unsurprisingly, known as dimensionality reduction. There are multiple ways one can approach this problem; one way to do is through *feature extraction.* Principal Component Analysis (PCA) is a way to extract features from the dataset. This paper discusses the Singular Value Decomposition (SVD), a variant of PCA. In addition to that, the paper also discusses how the data is generated and cleaned for analysis.

A raw dataset contains three videos, capturing the motion of an object suspended through a thin cable, from three different angles using mobile camera. In the dataset **Test 1**, the object only moves vertically. In second dataset **Test 2**, the object moves vertically but large noise is added through camera motion. In third dataset **Test 3**, the object moves vertically along z direction as well as along x and y direction in a simple harmonic motion. In the last dataset **Test 4**, the object moves vertically along z direction, moves along x and y direction in a simple harmonic motion and also rotates about the z axis. The underlying dimensions (directions in which the object is moving) in four datasets are 1,1,3 and 4. Positional data of the object was generated from each dataset. The goal was to extract dimensions of the motion from this data using PCA.

Section 2 put forwards theoretical framework of the SVD in the context of PCA. Section 3 discusses the data analysis tools/methods used to perform the PCA. Section 4 explains the computational outcome of the analysis.

2 Theoretical Background

Principal Component Analysis is a method of extracting feature space from a high-dimensional dataset of a low-dimensional system. It does so by projecting (orthogonal projection) the data containing possibly correlated variables onto a smaller space of linearly uncorrelated variables, known as principal components. Suppose we have a matrix A such that each row represent a unique dataset and the number of columns represent the number of data-points in each dataset.

$$A = \begin{bmatrix} -\mathbf{a}- \\ -\mathbf{b}- \\ \vdots \end{bmatrix}$$

The variances of $\mathbf{a}$ and $\mathbf{b}$ are given by,

$$\sigma^2_{\mathbf{a}} = \frac{1}{n-1}\mathbf{a}\mathbf{a}^T$$
$$\sigma^2_{\mathbf{b}} = \frac{1}{n-1}\mathbf{a}\mathbf{b}^T$$

where the normalization constant of $1/(n-1)$ is for an unbiased estimator. Similarly, the covariance between $\mathbf{a}$ and $\mathbf{b}$ is given by,

$$\sigma^2_{\mathbf{ab}} = \frac{1}{n-1}\mathbf{a}\mathbf{b}^T$$

These variances and covariances measure how much two datasets are statistically dependent/independent. Thus we can construct a covariance matrix C_A whose diagonal represents the variance of particular dataset while the off-diagonal terms represent covariances between datasets.

$$C_A = \frac{1}{n-1} AA^T$$

A large off-diagonal term corresponds to redundancy while small off-diagonal term indicate that the two datasets are independent and not redundant. Thus we want to diagonalize the covariance matrix. In other words, we would like to identify an ideal *basis* in which the C_A can be written so that in this basis, all redundancies have been removed and the largest variances of particular measurements are ordered.

Singular Value Decomposition (SVD) is a method of diagonalizing a matrix by identifying a pair of orthonormal bases U and V. Furthermore, every matrix can be singularly decomposed.

$$A = U\Sigma V^*$$

By defining the tranformed variable $Y = U^*A$, we can compute the covariance of matrix Y.

$$\begin{aligned}
C_Y &= \frac{1}{n-1} YY^T \\
&= \frac{1}{n-1} (U^*A)(U^*A)^T \\
&= \frac{1}{n-1} U^*(AA^T)U \\
&= \frac{1}{n-1} U^*(U\Sigma^2 U^*)U \\
&= \frac{1}{n-1} \Sigma^2
\end{aligned}$$

This means, if we represent our data A in the form of Y using left singular matrix U, then the covariance matrix of Y will be the diagonal matrix.

3 Algorithm Implementation and Development

The analysis entailed three steps.

3.1 Extract positional data from a video

A tracking algorithm was written that would track the flashlight placed on top of the object between each successive frames. The position of the flashlight on the first frame was manually identified. Since the flashlight moves only a small distance between successive frames, a small region of interest (ROI) window, centered at the previous position, was used to search for the flashlight in the next frame. The flashlight was identified by searching for the pixel with the highest internsity in the window. If there were multiple pixels, then top-leftmost pixel was used. For videos where flashlight was not visible at all time, a small white patch of the object (paint), was used for tracking.

3.2 Data cleaning and formatting

Once the positional data was obtained, it was cleaned and formatted to be used for PCA. The process began by first synching data of all three cameras by choosing the same starting position (roughly through manual inspection). The other end of the data was trimmed to generate equal vectors of data-points. The trimming point was chosen such that it excludes any irregularities in the data near the end. Finally, the data was normalized to $[0, 1]$.

Algorithm 1: extract_position.m

```
Load the video
Manually find the position in the first frame.
for each successive frame do
    Define ROI window centered at previous position
    Search for max intensity pixel inside ROI window
    Save the position
end for
Write position data in a .mat file
```

3.3 PCA

A 6-row matrix was created by stacking all x and y data-points of each cameras. To generate zero mean and unit variance data, each element of each row is subtracted by the mean of the row and divided by the standard deviation of the row. SVD of the matrix was found using the in-built MATLAB command *svd*.

4 Computational Results

4.1 Test 1:

In this test dataset, the object was moving only along the vertical direction. The noise level in the dataset was relatively mild. Thus, we expected PCA to be able to find one dominant principal mode that reflected the motion along z direction. The position data was obtained by tracking the flashlight using the 40 x 40 ROI window, as described in the algorithm. It was checked by overlaying estimates over each image frame. The data was then cleaned and formatted to be used for PCA. Figures 1 and 2 shows the outcome of two key steps in this process.

The PCA was performed through SVD. The figure 3(a) shows the percentage of energy captured by each mode. The first three modes captured $\approx 90\%$ of the total energy. In figure figure 3(b), the evolution of each linearly uncorrelated mode, obtained by projecting each data vector onto the left singular vector, was plotted. The first mode represents the simple harmonic motion of the object along z direction. The second mode represents the change in the amplitude of oscillations. Other modes, with energy level $< 10\%$, arise due to noise that comes from camera shake and error in position estimation.

4.2 Test 2:

In this test, the object was again moving in a simple harmonic motion along z direction. However, large noise was injected into the datasets by shaking the camera. While the underlying dimension of the system

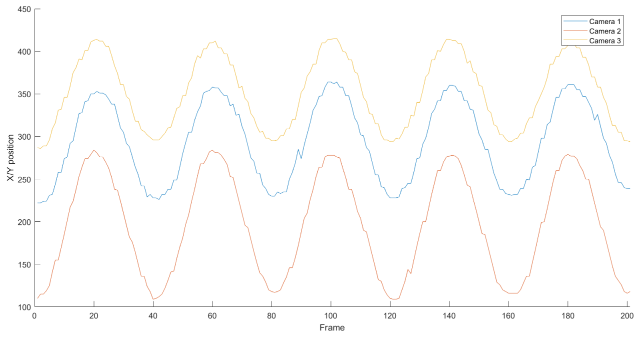


Figure 1: Plots show the position of the object in the image row-space after synchronization and trimming of the dataset.

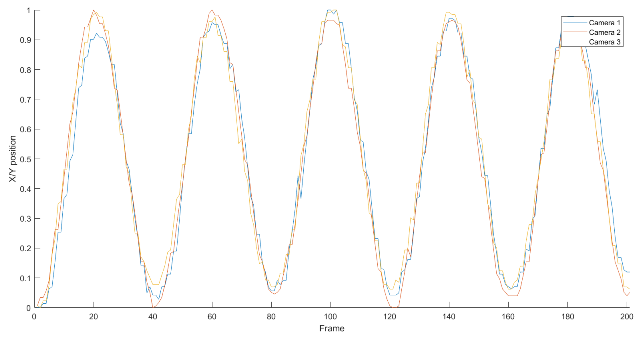
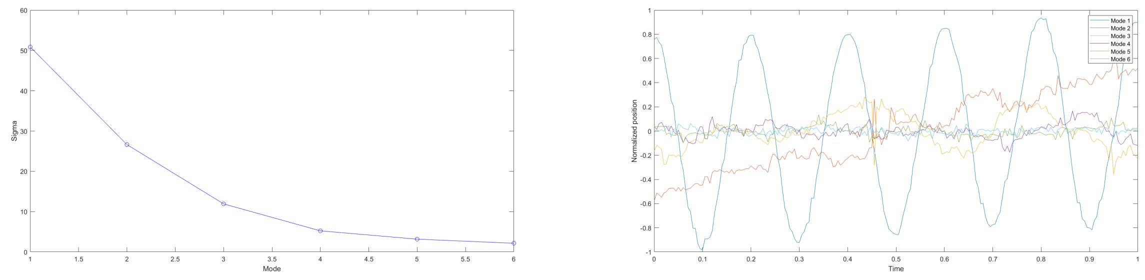


Figure 2: Plots show the normalized position in the image row-space.

was still 1, it was observed that the PCA was not able to extract any feature from the data. The noise severly hampered the efficacy of the PCA. Figure 4(b) shows no distinct motion in any of the 6 modes.

4.3 Test 3:

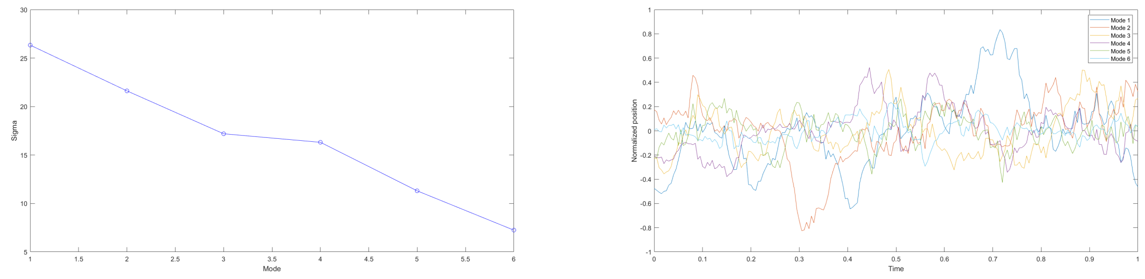
In Test 3 dataset, the object was moving in a simple harmonic motion along z direction as well as in pendulum motion along x and y direction. The noise level was relatively small (compared to Test 2). The underlying system has 3 degree of freedom. However, since we are estimation position through images, we are restricted to 2D. Thus we expect PCA to be able to find 2 principal modes to completely describe the motion in image space, given good position estimates are obtained. Unfortunately, the position estimtates were not as robust



(a) Percentage of variance captured by each mode.

(b) Evolution of proper orthogonal modes.

Figure 3: Principal Component Analysis on dataset Test 1 through singular value decomposition.



(a) Percentage of variance captured by each mode

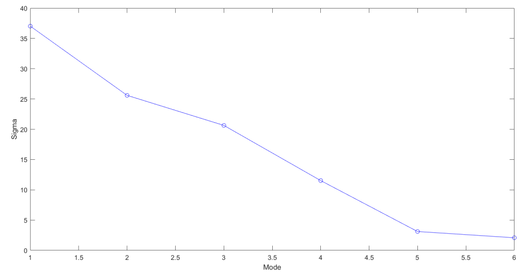
(b) Evolution of proper orthogonal modes.

Figure 4: Principal Component Analysis on dataset Test 2 through singular value decomposition.

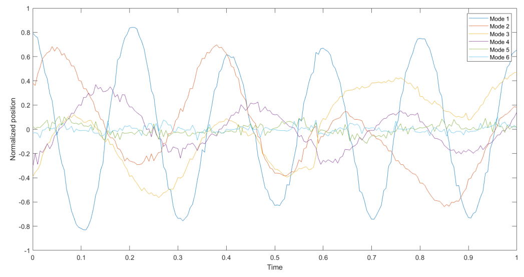
as that of Test 1 because of the flashlight occlusion. On some part of the data, the position estimates drifted away from the true value. Therefore we see in the figure 5, 4 dominant modes. Modes 1 and 2 represent the motion in the 2D image space, while modes 3 and 4 represent drift in the position estimates. Other modes correspond to the noise in the data.

4.4 Test 4:

In the last dataset, the object was moving in a simple harmonic motion along z direction, in a pendulum motion along x and y direction and in a rotating motion about the z axis. The noise level was again relatively small compared to Test 2. In this case, the underlying dimension of the system is 4. As mentioned earlier, since we are operating through images, PCA will at best give 2 dominant modes. For this case, the object was tracked using a patch of white space in the paint. The rotation of flashlight made it difficult to track it. Thus, the data only captures 3 degree of freedom motion, similar to Test 3. The rotation of flashlight about z axis is the latent variable that we do not expect to be resolved. Figure 6(b) shows two dominant modes that represents the simple harmonic motion along z axis and the pendulum motion along x and y direction. Since the object's pendulum motion discipated quickly, we see same dissipation of mode 2. Since, the noise in the position estimates was very large, we see large noisy modes.

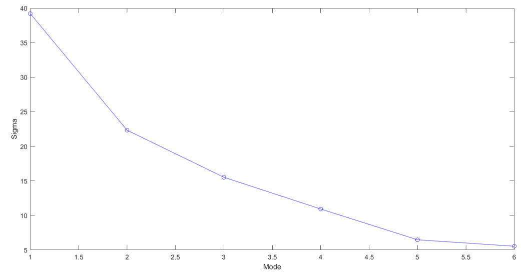


(a) Percentage of variance captured by each mode

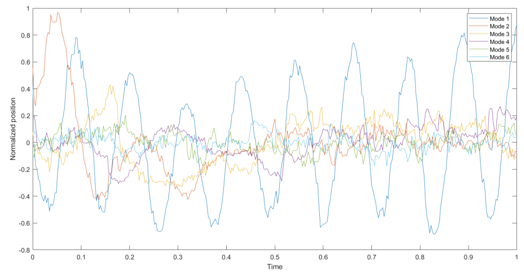


(b) Evolution of proper orthogonal modes.

Figure 5: Principal Component Analysis on dataset Test 3 through singular value decomposition.



(a) Percentage of variance captured by each mode



(b) Evolution of proper orthogonal modes.

Figure 6: Principal Component Analysis on dataset Test 4 through singular value decomposition.

5 Summary and Conclusions

The concept of reducing the dimensions of the dataset through diagonalization of the covariance matrix was introduced. PCA was used to extract the low-dimensional motion of the mass-string system from series of images. A tracking algorithm was written to estimate the position of the object in each frame. The positional data was then synchronized, trimmed and normalized for analysis. Each dataset was projected onto the left singular vectors obtained from the SVD to extract linearly uncorrelated modes (principal components). The effect of noise and the dimension of the feature space on the outcome of the PCA was also examined. It was found that large noise severely compromize the efficacy of PCA method. PCA was able to identify the simple harmonic motion of the object inTest 1, 3 and 4. In Test 3, it was able to also identify the pendulum motion of the object.

Appendix A   MATLAB Functions

Some important MATLAB functions used during the implementation.

- `y = linspace(x1,x2,n)` returns a row vector of `n` evenly spaced points between `x1` and `x2`.
- `ks = fftshift(k)` rearranges FT by shifting zero frequency component to the center of the array. Exchanges two halves of the array.
- `Y = fftn(X)` returns multidimensional Fourier transform of an N-D array using FFT algorithm.
- `[U,S,V] = svd(A)` performs a singular value decomposition of matrix A, such that A = U*S*V'.
- `B = repmat(A,r1,...,rN)` specifies a list of scalars, r1,..,rN, that describes how copies of A are arranged in each dimension. When A has N dimensions, the size of B is size(A).*[r1...rN]. For example, repmat([1 2; 3 4],2,3) returns a 4-by-6 matrix.

Appendix B MATLAB Code

The code is published on the github repository AMATH-582 under Principal Component Analysis directory.

MATLAB code for extracting position.

```
%% Clear workspace
close all; clear variables; clc

%% Set up Global variables
Test = 4;   % 1, 2, 3 or 4
Save = 1;
Filter = 0;
Verbose = 1;

%% Load Test data
load(['./Data/Test' num2str(Test) '/cam1_' num2str(Test) '.mat']);
load(['./Data/Test' num2str(Test) '/cam2_' num2str(Test) '.mat']);
load(['./Data/Test' num2str(Test) '/cam3_' num2str(Test) '.mat']);

if Test == 1
    window = 15;
    camera1 = vidFrames1_1;
    camera2 = vidFrames2_1;
    camera3 = vidFrames3_1;
    X1=321; Y1=224;
    X2=275; Y2=275;
    X3=319; Y3=272;
elseif Test == 2
    window = 20;
    camera1 = vidFrames1_2;
    camera2 = vidFrames2_2;
    camera3 = vidFrames3_2;
    X1=325; Y1=308;
    X2=313; Y2=360;
    X3=348; Y3=245;
elseif Test == 3
    window = 20;
    camera1 = vidFrames1_3;
    camera2 = vidFrames2_3;
    camera3 = vidFrames3_3;
    X1=320; Y1=286;
    X2=238; Y2=292;
    X3=352; Y3=231;
elseif Test == 4
    window = 20;
    camera1 = vidFrames1_4;
    camera2 = vidFrames2_4;
    camera3 = vidFrames3_4;
    X1=423; Y1=325;
    X2=220; Y2=321;
    X3=412; Y3=182;
end

[m1,n1,~,t1] = size(camera1);
```

```
[m2,n2,~,t2] = size(camera2);
[m3,n3,~,t3] = size(camera3);

%% Check the data
if Verbose
    for j=1:t1
        imshow(camera1(:,:,:,j))
        drawnow;
    end
    for j=1:t2
        imshow(camera2(:,:,:,j))
        drawnow;
    end
    for j=1:t3
        imshow(camera3(:,:,:,j))
        drawnow;
    end
end

%% Find position of the bucket
% Camera 1
position1 = zeros(2, t1);
position1(:,1) = [X1; Y1];

for frame = 2:t1
    img = camera1(:,:,:,frame);
    X = position1(1,frame-1); Y = position1(2, frame-1);
    roi = crop(img, X, Y, window);
    mask = (sum(roi, 3) == max(max(sum(roi,3))));
    [r, c] = find(mask);
    position1(:,frame) = [X - window + c(1); Y - window + r(1)];
end
figure(1)
plot(1:t1, position1(2,:))

% Camera 2
position2 = zeros(2, t2);
position2(:,1) = [X2; Y2];

for frame = 2:t2
    img = camera2(:,:,:,frame);
    X = position2(1,frame-1); Y = position2(2, frame-1);
    roi = crop(img, X, Y, window);
    mask = (sum(roi, 3) == max(max(sum(roi,3))));
    [r, c] = find(mask);
    position2(:,frame) = [X - window + c(1); Y - window + r(1)];
end
figure(2)
plot(1:t2, position2(2,:))

% Camera 3
position3 = zeros(2, t3);
position3(:,1) = [X3; Y3];
```

```
for frame = 2:t3
    img = camera3(:,:,:,frame);
    X = position3(1,frame-1); Y = position3(2, frame-1);
    roi = crop(img, X, Y, window);
    mask = (sum(roi, 3) == max(max(sum(roi,3))));
    [r, c] = find(mask);
    position3(:,frame) = [X - window + c(1); Y - window + r(1)];
end
figure(3)
plot(1:t3, position3(2,:))
figure(4)
plot3(position3(1,:), position3(2,:), 1:t3)

%% Plot position onto image
if Verbose
    figure()
    for frame = 1:t1
        imshow(camera1(:,:,:,frame))
        hold on
        plot(position1(1,frame), position1(2,frame), 'ro', 'Linewidth', 3)
        drawnow
    end

    figure()
    for frame = 1:t2
        imshow(camera2(:,:,:,frame))
        hold on
        plot(position2(1,frame), position2(2,frame), 'ro', 'Linewidth', 3)
        drawnow
    end

    figure()
    for frame = 1:t3
        imshow(camera3(:,:,:,frame))
        hold on
        plot(position3(1,frame), position3(2,frame), 'ro', 'Linewidth', 3)
        drawnow
    end
end

%% Shannon Filter the position data (Did not give much improvement!)
if Filter
    position1 = position1(:,10:210);
    position2 = position2(:,20:220);
    position3 = position3(:,10:210);

    xposition1_t = fft(position1(1,:));
    xposition1_t(10:end-10) = 0;
    position1(1,:) = abs(ifft(xposition1_t));

    yposition1_t = fft(position1(2,:));
    yposition1_t(10:end-10) = 0;
    position1(2,:) = abs(ifft(yposition1_t));
    xposition2_t = fft(position2(1,:));
    xposition1_t(10:end-10) = 0;
    position2(1,:) = abs(ifft(xposition2_t));

    yposition2_t = fft(position2(2,:));
    yposition1_t(10:end-10) = 0;
    position2(2,:) = abs(ifft(yposition2_t));

    xposition3_t = fft(position3(1,:));
    xposition3_t(10:end-10) = 0;
    position3(1,:) = abs(ifft(xposition3_t));

    yposition3_t = fft(position3(2,:));
    yposition3_t(10:end-10) = 0;
    position3(2,:) = abs(ifft(yposition3_t));

    if Verbose
        figure()
        for frame = 1:t1
            imshow(camera1(:,:,:,frame))
            hold on
            plot(position1(1,frame), position1(2,frame), 'ro', 'Linewidth'
                , 3)
            drawnow
        end

        figure()
        for frame = 1:t2
            imshow(camera2(:,:,:,frame))
            hold on
            plot(position2(1,frame), position2(2,frame), 'ro', 'Linewidth'
                , 3)
            drawnow
        end

        figure()
        for frame = 1:t3
            imshow(camera3(:,:,:,frame))
            hold on
            plot(position3(1,frame), position3(2,frame), 'ro', 'Linewidth'
                , 3)
            drawnow
        end
    end
    figure(); hold on
    plot(position1(2,:))
    plot(position2(2,:))
    plot(position3(1,:))
end

%% Save the position data
if Save
    save(['./Data/Test' num2str(Test) '/position.mat'], 'position1', '
       position2', 'position3');
```

```
end
```

MATLAB code for PCA.

```
%% Clear Workspace
close all; clear variables; clc

%% Global Variables
Test = 1;
Verbose = 1;

%% Load position data
data = load(['./Data/Test' num2str(Test) '/position.mat']);

if Verbose
    figure(); hold on
    plot(data.position1(2,:)),
    plot(data.position2(2,:))
    plot(data.position3(1,:))
    xlabel('Frame'), ylabel('Y position')
    legend('Camera 1', 'Camera 2', 'Camera 3')
end
%% Generate same size position vector for each camera
% Extract region of clean data.
if Test == 1
    data.position1 = data.position1(:,10:210);
    data.position2 = data.position2(:,20:220);
    data.position3 = data.position3(:,10:210);
    size=201;
elseif Test == 2
    data.position1 = data.position1(:,10:210);
    data.position2 = data.position2(:,1:201);
    data.position3 = data.position3(:,15:215);
    size=201;
elseif Test == 3
    data.position1 = data.position1(:,18:218);
    data.position2 = data.position2(:,5:205);
    data.position3 = data.position3(:,10:210);
    size=201;
elseif Test == 4
    data.position1 = data.position1(:,2:352);
    data.position2 = data.position2(:,10:360);
    data.position3 = data.position3(:,1:351);
    size=351;
end

time = linspace(0,1,size);

if Verbose
    figure(); hold on
    plot(1:size,data.position1(2,:)),
    plot(1:size,data.position2(2,:))
    plot(1:size,data.position3(1,:))
    xlim([0 size])
    xlabel('Frame'), ylabel('X/Y position')
    legend('Camera 1', 'Camera 2', 'Camera 3')
end
```

```
if Verbose
    figure(); hold on
    plot(1:size,data.position1(1,:)),
    plot(1:size,data.position2(1,:))
    plot(1:size,data.position3(2,:))
    xlim([0 size])
    xlabel('Frame'), ylabel('Y position')
    legend('Camera 1', 'Camera 2', 'Camera 3')
end

%% Generate position data normalized to [0,1]
for i=[1,2]
    data.position1(i,:) = (data.position1(i,:) - min(data.position1(i,:)))
       / ...
        (max(data.position1(i,:)) - min(data.position1(i,:)));
    data.position2(i,:) = (data.position2(i,:) - min(data.position2(i,:)))
       / ...
        (max(data.position2(i,:)) - min(data.position2(i,:)));
    data.position3(i,:) = (data.position3(i,:) - min(data.position3(i,:)))
       / ...
        (max(data.position3(i,:)) - min(data.position3(i,:)));
end

if Verbose
    figure(); hold on
    plot(1:size,data.position1(2,:))
    plot(1:size,data.position2(2,:))
    plot(1:size,data.position3(1,:))
    xlim([0 size])
    xlabel('Frame'), ylabel('X/Y position')
    legend('Camera 1', 'Camera 2', 'Camera 3')
end

%% Construct the matrix with zero-mean position data
A = [data.position1(1,:); data.position1(2,:); data.position2(1,:); ...
    data.position2(2,:); data.position3(1,:); data.position3(2,:)];
A = A-repmat(mean(A,2),1,size);

%% Perform the SVD on matrix A

[U, S, V] = svd(A);
sigma = diag(S);

% Energy captured in each POD modes
figure();
plot(sigma/(sum(sigma))*100,'-bo')
xlabel('Mode'); ylabel('Sigma')

% Projection of modes onto the left singular vectors
B = U.'* A;
figure()
plot(time,B(1,:),time,B(2,:),time,B(3,:),time,B(4,:),time,B(5,:),time,B
   (6,:))
```

```
legend('Mode 1','Mode 2','Mode 3','Mode 4','Mode 5','Mode 6')
xlabel('Time'); ylabel('Normalized position')
```