HW3: PCA – Principal Component Analysis

AMATH 482 | Celeste Becker

Abstract

In this report is about using PCA on different datasets. The goal is to illustrate the usefulness and effects of noise on the PCA algorithm. A mass is attached to a spring and is bouncing up and down. There are four different tests of the mass on the spring. Given video taken from multiple different perspectives, for each test we will extract the position of the mass and then use PCA to analyze the dynamics of this system. By extracting the energies from each dataset, we should be able to tell what dimension the motion of the mass is in.

Sec. I. Introduction and Overview

Initially we are given movie files that are turned into MATLAB files. There are three different cameras simultaneously filming the spring on the mass. Each camera is at a different angle. The goal is to use PCA to extract the dynamics of the system. The mass is an off-white bucket attached to the spring. There is a flashlight, producing a bright white light taped on top of the bucket, making it easier to extract the position of the mass. There are four different tests of the mass on the spring.

Test 1: The Ideal case. Here the entire motion of the mass is in the z direction and simple harmonic motion is observed.

Test 2: The cameras are shaking, causing a noisier dataset. The entire motion of the mass is still in the z direction.

Test 3: The mass is released off-center, producing motion in the x-y plane as well as the z direction.

Test 4: Here the mass is released off-center which produces motion in the x-y plane, there is also rotation, and motion in the z direction.

Each test will create a redundant dataset with a different amount of noise.

In order to use the PCA algorithm, we first need to extract the mass positions from the videos. We do this by converting to each frame of the video to a gray scale image and saving the averaged the position of the white light on top of the bucket. Then we will use PCA to analyze each dataset and discover how noise from the different tests effects the PCA. Finally, we will use the singular values to find what dimension the motion of the mass is in for each test.

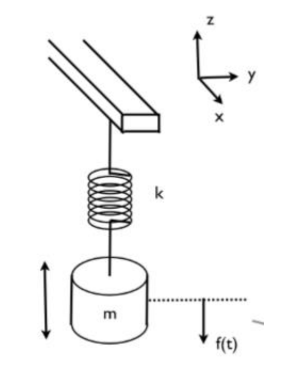


Diagram 1 - Representation of mass on spring

Sec. II. Theoretical Background

The main mathematical methods used in this program is *PCA, Principal Component Analysis*, which in turn uses *SVD, Singular Value Decomposition.* The information below is referenced from chapter 15 of *Data-Driven Modeling & Scientific Computation: Methods for Complex Systems & Big Data* by J. Nathan Kutz.

Singular Value Decomposition (SVD): The factorization of a matrix into several constitutive components all of which have a specific meaning in applications. Below is the equation for the SVD decomposition of an m x n matrix A:

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

U and V are both unitary matrixes, and Σ is a diagonal matrix, with entries that are nonnegative and ordered from largest to smallest. U contains the principal components, and Σ contains the singular values, and V contains time information. The SVD makes it possible for every matrix to be diagonal if the proper bases for the domain and range are used. This is a type of least-square fitting algorithm, which lets us project the matrix onto low-dimensional representations in a formal algorithmic way.

To compute the energy of a matrix, we use the singular values (σ) contained in Σ :

$Energies = \frac{\sigma^2}{\sum_1^n \sigma_n^2}$ example: $Energy\ 3 = \frac{\sigma_1^2+\sigma_2^2+\sigma_3^2}{\sum_1^n \sigma_n^2}$

In the above formula n is the total number of singular values. Large singular values correspond to interesting corelating data.

Principal Component Analysis (PCA): This statistical procedure converts a set of observations of possibly corelated variables into a set of linearly uncorrelated variables called principal components. It does this using an orthogonal transformation.

The way we are applying principal component analysis into this program is by storing the camera data in a matrix called x:

$$X = \begin{pmatrix} y_a \\ z_a \\ y_b \\ z_b \\ y_c \\ z_c \end{pmatrix}$$

Where the subscripts a, b, and c represent the different cameras, with x and y being the position data. We then feed this matrix X into the SVD algorithm, to find correlations in the data.

Sec. III. Algorithm Implementation and Development

Each section labeled here corresponds to the commented section with the same name in *Appendix B. Matlab Codes* at the end of this report. Code reference lines are also included. This program has been split up into four files. One for each test. Each section starting with TEST is a different file.

Tracking the mass: Each Test follows a similar method for extracting the positions of the masses. Each video is re-framed to new z and y coordinates to zoom in on the mass. This new frame size cuts out un-necessary data from each frame, making it easier to track the mass. We go through one frame at a time and convert it to a gray scale image. Then we extract the positions of all pixels that are greater than a bright white RGB value. This is because there is a white light attached to the mass. The mass is a darker shade of white. Depending on the video you will pick up the values of the flashlight, or values of the white mass. Then we take the mean of this position. There is also noise from other white objects in the frame, but we try to minimize this by cutting down the size of the video frame. Additionally, because each video has a different number of frames, we must use a common number of frames for each when storing into the X matrix. Example reference: TEST 1: Cam 1 - lines (10 - 22)

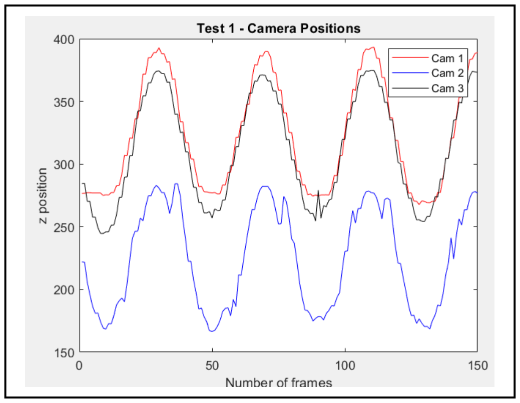
Plot positions from each camera: Using the stored Z, and Y positions, we make a plot of Z position verses the frame number to see the oscillation of the mass as the frame number increases (time goes on). Example reference: TEST 1: Cam 1 - lines (28 -29)

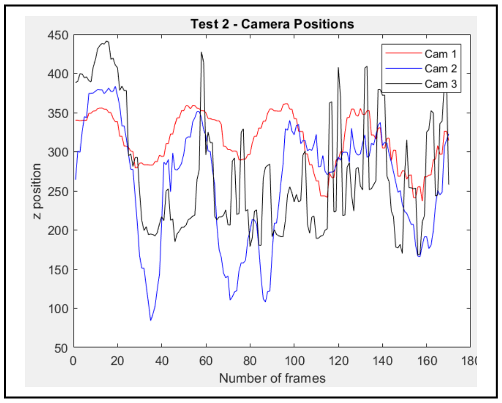
PCA: We store all positions into a matrix X as defined above. Where each subscript a, b, and c represents position data from a different camera. We then feed this X matrix into the SVD algorithm to find the singular values and principal components.

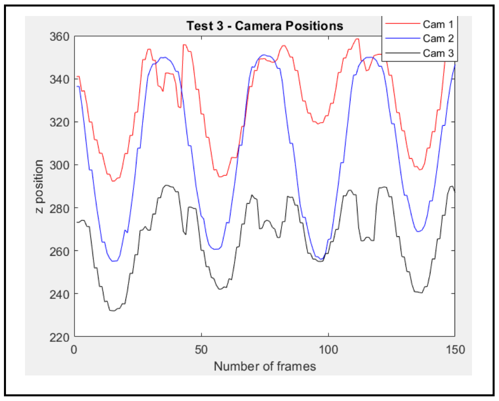
SVD – Singular Value Decomposition: We then compute the SVD on the X matrix. This gives us three different matrices, U (containing the principal components), S (containing the singular values and V (which contains time information). Example reference: TEST 1: PCA using SVD - lines (79 - 91)

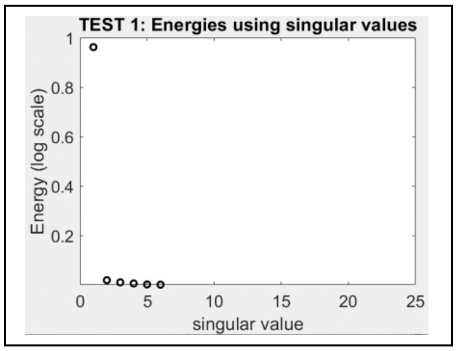
Singular Values: Taking the matrix S which holds the singular values, we take the diagonal of this matrix which contains the variances between datasets, and compute the energy of the matrix using the energy equation defined above. This allows us to see what energies, (singular values) are the largest and stand out from the rest of the dataset.

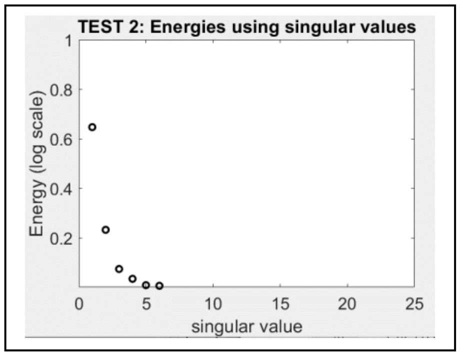
Log scale plot of the energies: By plotting the energies of each singular value on a log plot, it is easier to tell the magnitude of difference between the energies, and tell what correlations are being made. Example reference: TEST 1: Plot energies - lines (92-100).

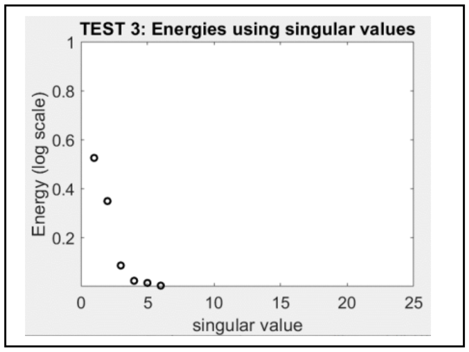












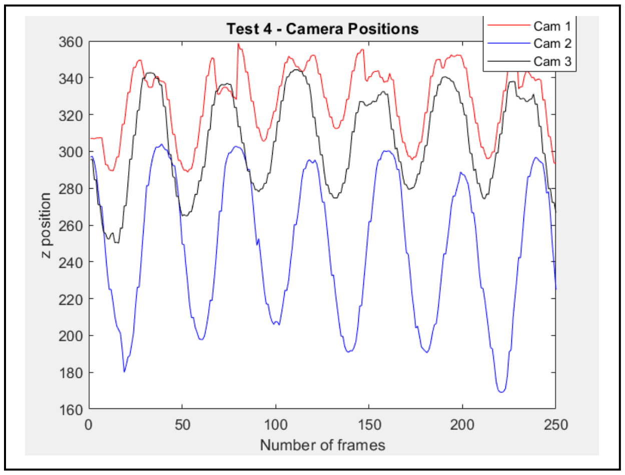
Sec. IV.
Computational Results

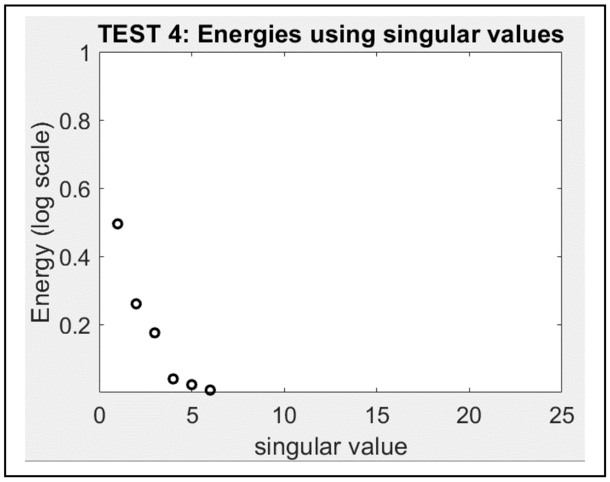
By plotting the z position verses video frame, we see the vertical position of the mass over time. Plotting the energy of the data set allows us to tell the rank by looking at the largest energies and therefore the dimension of the motion of the mass for each test.

Test 1: The far-left plots, top and bottom, are for test one. The only motion of the mass is in the z direction. This makes the top plot look very clean. This also means that only one variable should corelate between each camera video. We see this in the result for by plotting the energies. You can see that only the first energy is very large, compared to the tail of energies that represent non-corelating singular values. Therefore, the mass in test one has one dimensional motion.

Test 2: The plots in the middle, top and bottom, are for test two. The motion of the mass is the same as in Test 1, but the data is noisier because of camera shake. This makes it a little harder to tell from plot of singular value energies what is corelating, and the data point that is just above 0.2 is likely produced from the noise of the camera shake. This means that the mass in test two also has one dimensional motion, with the second largest energy being discarded because it represents camera shake, and most of the energy is contained in the first energy.

Test 3: The plots on the far right, top and bottom, are for test three. Here two variables will corelate between videos. The position in the z and y direction. See the bottom plot, the first two energy values contain most of the energy, leading to rank 2. This means the mass has two-dimensional motion for test 3. This dataset was a little noisier than in test 1, but it is still possible to determine the dimension of the motion.





Test 4: The two graphs above are for test four. The position in the z and y direction, and the rotation of the mass should be redundant data from each camera. This means there will be three singular values of interest leading to three-dimensional motion. The data is a little noisy, leading to not quite as accurate singular values, leading to not quite as accurate energy values. However, you can see that most of the energy here is represented by the first three data points. This means that the data has a rank three, leading to three-dimensional motion which is what we expected.

Sec. V. Summary and Conclusions

Using Principal Component Analysis, we were able to find the dimension of motion for the mass in each test. We were able to see that by using PCA you can find correlations between different redundant sets of data. Additionally, we explored how noise effects the PCA. Smoother data makes it easier to find correlations, and noisier data makes it harder to tell what correlations there are in the data.

Appendix A. MATLAB functions used and brief implementation explanation

rgb2gray

- I = rgb2gray (RGB)**.**
- Converts a truecolor image to a gray scale image I.
- *Implementation*: used to convert an individual video frame to a gray scale image Ex: TEST 1 line 17.

Repmat

- B = repmat(A,n)**.**
- Returns an array containing n copies of A in the row and col dimensions.

- *Implementation*: Used to subtract the mean from each row of matrix X. Ex: TEST 1 line 82.

svd

- [u,s,v] = svd(A)
- Returns the singular value decomposition of matrix A. A = u*s*v'.
- *Implementation*: used to compute the SVD on the position data. Ex: TEST 1 line 85.

Diag

- D = diag(v)
- Returns a square matrix with vector v as the main diagonal.
- *Implementation*: used to take the diagonal values from the sigma matrix Ex: TEST 1 line 89

Appendix B. MATLAB codes

Each test was programmed in a separate file:

Test 1:

```
%% -----------------TEST 1---------------

%% set up PCA data matrix X - lines (3-9)
clear; close all; clc;
%take same number of frames for each camera
%fill the X matrix with the camera position data for each cam
frameNum = 150;
X = zeros(6,frameNum);

%% Cam 1 -  lines (10 - 30)
load('cam1_1.mat')
numFrames = size(vidFrames1_1,4);
zCam1 = zeros(1,numFrames);
yCam1 = zeros(1,numFrames);
%implay(vidFrames1_1);
for j = 1:numFrames
    bnwFrame = rgb2gray(vidFrames1_1(100:420,300:400,:,j));
    [z,y,x] = find(bnwFrame >= 250);

    zCam1(j) = 100 + mean(z);
    yCam1(j) = 300 + mean(y);
```

```
end

%save cam 1 position data to X matrix
X(1,:) = yCam1(1:frameNum);
X(2,:) = zCam1(1:frameNum);

figure(1)
plot(1:frameNum,zCam1(1:frameNum), 'r');

%% Cam 2 - lines (31 - 51)
load('cam2_1.mat')
numFrames2 = size(vidFrames2_1,4);

zCam2 = zeros(numFrames2,1);
yCam2 = zeros(numFrames2,1);
%implay(vidFrames2_1(90:300,260:350,:,:));

for j = 1:numFrames2
    bnwFrame = rgb2gray(vidFrames2_1(90:300,260:350,:,j));
    [z, y, x] = find(bnwFrame >= 250);
    zCam2(j) = 90 + mean(z);
    yCam2(j) = 260 + mean(y);
end

X(3,:) = yCam2(11:frameNum+10);
X(4,:) = zCam2(11:frameNum+10);

hold on
plot(1:frameNum,zCam2(11:frameNum+10), 'b')

%% Cam 3 - lines (52 - 78)
load('cam3_1.mat')
numFrames3 = size(vidFrames3_1,4);

zCam3 = zeros(numFrames3,1);
yCam3 = zeros(1,numFrames3);

for j = 1:numFrames3
    bnwFrame = rgb2gray(vidFrames3_1(220:290,260:450,:,j));

    [z, y, x] = find(bnwFrame >= 250);
    %because the image is sideways, you need to switch the z and
y
    zCam3(j) = 220 + mean(y);
    yCam3(j) = 260 + mean(z);
end
```

```
%save cam 3 pos data to X matrix
X(5,:) = yCam3(1:frameNum);
X(6,:) = zCam3(1:frameNum);

hold on
plot(1:frameNum,zCam3(1:frameNum),'k')
title('Test 1 - Camera Positions');
legend('Cam 1','Cam 2', 'Cam 3');
ylabel('z position');
xlabel('Number of frames');

%% PCA using SVD - lines (79 - 91)
[m,n]=size(X); % compute data size
mn=mean(X,2); % compute mean for each row
X=X-repmat(mn,1,n); % subtract mean

%svd
[u,s,v]=svd(X'/sqrt(n-1)); % perform the SVD - s = singular
values


%compute energy
sig=diag(s); %this takes the diagonal values of the singular
values
energies = sig.^2/sum(sig.^2);

%% Plot energies - lines (92-100)
figure(2)
plot(energies,'ko','Linewidth',2)
axis([0 25 10^-(18) 1])
ylabel('Energy (log scale)')
xlabel('singular value');
title('TEST 1: Energies using singular values');
set(gca,'Fontsize',16,'Xtick',0:5:25)
```

Test 2:

```
%% -----------------TEST 2---------------

%% set up PCA data matrix X - lines (3-10)
clear; close all; clc;

%take same number of frames for each camera (200 frames)
%fill the X matrix with the camera position data for each cam
```

```
frameNum = 170;
X = zeros(6,frameNum);

%% Cam 1 - lines (11 - 31)
load('cam1_2.mat')
numFrames = size(vidFrames1_2,4);
zCam1 = zeros(1,numFrames);
yCam1 = zeros(1,numFrames);

for j = 1:numFrames
    bnwFrame = rgb2gray(vidFrames1_2(200:380,300:400,:,j));
    [z,y,x] = find(bnwFrame >= 250);

    zCam1(j) = 200 + mean(z);
    yCam1(j) = 300 + mean(y);
end

%save cam 1 position data to X matrix
X(1,:) = yCam1(1:frameNum);
X(2,:) = zCam1(1:frameNum);

figure(1)
plot(1:frameNum,zCam1(1:frameNum), 'r');

%% Cam 2  - lines (32 - 52)
load('cam2_2.mat')
numFrames2 = size(vidFrames2_2,4);

zCam2 = zeros(numFrames2,1);
yCam2 = zeros(numFrames2,1);
%implay(vidFrames2_1(90:300,260:350,:,:));

for j = 1:numFrames2
    bnwFrame = rgb2gray(vidFrames2_2(:,200:400,:,j));
    [z, y, x] = find(bnwFrame >= 250);
    zCam2(j) = mean(z);
    yCam2(j) = 200 + mean(y);
end

X(3,:) = yCam2(25:frameNum+24);
X(4,:) = zCam2(25:frameNum+24);

hold on
plot(1:frameNum,zCam2(25:frameNum+24), 'b')

%% Cam 3 - lines (53 - 79)
load('cam3_2.mat')
```

```
numFrames3 = size(vidFrames3_2,4);

zCam3 = zeros(numFrames3,1);
yCam3 = zeros(1,numFrames3);

for j = 1:numFrames3
    bnwFrame = rgb2gray(vidFrames3_2(:,:,:,j));

    [z, y, x] = find(bnwFrame >= 250);
    %because the image is sideways, you need to switch the z and
y
    zCam3(j) = mean(y);
    yCam3(j) = mean(z);
end

%save cam 3 pos data to X matrix
X(5,:) = yCam3(1:frameNum);
X(6,:) = zCam3(1:frameNum);

hold on
plot(1:frameNum,zCam3(1:frameNum),'k')
title('Test 2 - Camera Positions');
legend('Cam 1','Cam 2', 'Cam 3');
ylabel('z position');
xlabel('Number of frames');

%% PCA using SVD - lines (80 - 94)
[m,n]=size(X); % compute data size
mn=mean(X,2); % compute mean for each row
X=X-repmat(mn,1,n); % subtract mean

%svd
[u,s,v]=svd(X'/sqrt(n-1)); % perform the SVD - s = singular
values




%energies
sig=diag(s); %this takes the diagonal values of the singular
values
energies = sig.^2/sum(sig.^2);

%% Plot energies - lines (95-103)
figure(5)
% subplot(1,2,1)
plot(energies,'ko','Linewidth',2)
```

```
axis([0 25 10^-(18) 1])
ylabel('Energy (log scale)')
xlabel('singular value');
title('TEST 2: Energies using singular values');
set(gca,'Fontsize',16,'Xtick',0:5:25)
```

Test 3:

```
%% -----------------TEST 3----------------

%% set up PCA data matrix X - lines (3-10)
clear; close all; clc;

%take same number of frames for each camera (200 frames)
%fill the X matrix with the camera position data for each cam
frameNum = 150;
X = zeros(6,frameNum);

%% Cam 1 - lines (11 - 30)
load('cam1_3.mat')
numFrames = size(vidFrames1_3,4);
zCam1 = zeros(1,numFrames);
yCam1 = zeros(1,numFrames);
for j = 1:numFrames
    bnwFrame = rgb2gray(vidFrames1_3(230:360,250:350,:,j));
    [z,y,x] = find(bnwFrame >= 250);

    zCam1(j) = 230 + mean(z);
    yCam1(j) = 250 + mean(y);
end

%save cam 1 position data to X matrix
X(1,:) = yCam1(23:frameNum+22);
X(2,:) = zCam1(23:frameNum+22);

figure(1)
plot(1:frameNum,zCam1(23:frameNum+22), 'r');

%% Cam 2 - lines (31 - 51)
load('cam2_3.mat')
numFrames2 = size(vidFrames2_3,4);

zCam2 = zeros(numFrames2,1);
yCam2 = zeros(numFrames2,1);

for j = 1:numFrames2
```

```
    bnwFrame = rgb2gray(vidFrames2_3(150:360,200:400,:,j));
    [z, y, x] = find(bnwFrame >= 250);
    zCam2(j) = 150 + mean(z);
    yCam2(j) = 200 + mean(y);
end

%save cam 2 pos data to X matrix
X(3,:) = yCam2(11:frameNum+10);
X(4,:) = zCam2(11:frameNum+10);

hold on
plot(1:frameNum,zCam2(11:frameNum+10), 'b')

%% Cam 3  - lines (52 - 79)
load('cam3_3.mat')
numFrames3 = size(vidFrames3_3,4);

zCam3 = zeros(numFrames3,1);
yCam3 = zeros(1,numFrames3);

for j = 1:numFrames3
    bnwFrame = rgb2gray(vidFrames3_3(150:350,250:400,:,j));

    [z, y, x] = find(bnwFrame >= 235);
    %because the image is sideways, you need to switch the z and
y
    zCam3(j) = 150 + mean(y);
    yCam3(j) = 250 + mean(z);
end

%save cam 3 pos data to X matrix
X(5,:) = yCam3(15:frameNum+14);
X(6,:) = zCam3(15:frameNum+14);

hold on
plot(1:frameNum,zCam3(15:frameNum+14),'k')
title('Test 3 - Camera Positions');
legend('Cam 1','Cam 2', 'Cam 3');
ylabel('z position');
xlabel('Number of frames');

%% PCA using SVD - lines (79 - 91)
[m,n]=size(X); % compute data size
mn=mean(X,2); % compute mean for each row
X=X-repmat(mn,1,n); % subtract mean

%svd
```

```
[u,s,v]=svd(X'/sqrt(n-1)); % perform the SVD - s = singular
values


%energies
sig=diag(s); %this takes the diagonal values of the singular
values
energies = sig.^2/sum(sig.^2);

%% Plot energies - lines (92-100)
figure(5)
plot(energies,'ko','Linewidth',2)
axis([0 25 10^-(18) 1])
ylabel('Energy (log scale)')
xlabel('singular value');
title('TEST 3: Energies using singular values');
set(gca,'Fontsize',16,'Xtick',0:5:25)
```

Test 4:

```
%% -----------------TEST 4---------------

%% set up PCA data matrix X - lines (3-10)
clear; close all; clc;

%take same number of frames for each camera (200 frames)
%fill the X matrix with the camera position data for each cam
frameNum = 250;
X = zeros(6,frameNum);

%% Cam 1 - lines (11-31)
load('cam1_4.mat')
numFrames = size(vidFrames1_4,4);
zCam1 = zeros(1,numFrames);
yCam1 = zeros(1,numFrames);
for j = 1:numFrames
    bnwFrame = rgb2gray(vidFrames1_4(230:360,300:450,:,j));

    [z,y,x] = find(bnwFrame >= 240);

    zCam1(j) = 230 + mean(z);
    yCam1(j) = 300 + mean(y);
end

%save cam 1 position data to X matrix
```

```
X(1,:) = yCam1(1:frameNum);
X(2,:) = zCam1(1:frameNum);

figure(1)
plot(1:frameNum,zCam1(1:frameNum), 'r');

%% Cam 2 - lines (32 - 53)
load('cam2_4.mat')
numFrames2 = size(vidFrames2_4,4);

zCam2 = zeros(numFrames2,1);
yCam2 = zeros(numFrames2,1);

for j = 1:numFrames2
    bnwFrame = rgb2gray(vidFrames2_4(100:310,200:460,:,j));
    [z, y, x] = find(bnwFrame >= 250);
    zCam2(j) = 100 + mean(z);
    yCam2(j) = 200 + mean(y);
end

%save cam 2 pos data to X matrix
X(3,:) = yCam2(1:frameNum);
X(4,:) = zCam2(1:frameNum);

hold on
plot(1:frameNum,zCam2(1:frameNum), 'b')
%plot(1:frameNum,zCam2, 'b')

%% Cam 3 - lines (54 - 80)
load('cam3_4.mat')
numFrames3 = size(vidFrames3_4,4);

zCam3 = zeros(numFrames3,1);
yCam3 = zeros(1,numFrames3);

for j = 1:numFrames3
    bnwFrame = rgb2gray(vidFrames3_4(150:310,250:450,:,j));

    [z, y, x] = find(bnwFrame >= 230);
    %because the image is sideways, you need to switch the z and
y
    zCam3(j) = 150 + mean(y);
    yCam3(j) = 250 + mean(z);
end

%save cam 3 pos data to X matrix
X(5,:) = yCam3(1:frameNum);
```

```
X(6,:) = zCam3(1:frameNum);

hold on
plot(1:frameNum,zCam3(1:frameNum),'k')
title('Test 4 - Camera Positions');
legend('Cam 1','Cam 2', 'Cam 3');
ylabel('z position');
xlabel('Number of frames');

%% PCA using SVD - lines (81 - 95)
[m,n]=size(X); % compute data size
mn=mean(X,2); % compute mean for each row
X=X-repmat(mn,1,n); % subtract mean

%svd
[u,s,v]=svd(X'/sqrt(n-1)); % perform the SVD - s = singular
values




%energies
sig=diag(s); %this takes the diagonal values of the singular
values
energies = sig.^2/sum(sig.^2);

%% Plot energy and cumulative energy
figure(5)
plot(energies,'ko','Linewidth',2)
axis([0 25 10^-(18) 1])
ylabel('Energy (log scale)')
xlabel('singular value');
title('TEST 4: Energies using singular values');
set(gca,'Fontsize',16,'Xtick',0:5:25)
```