AMATH 582 Homework 3

Shannon Dow

February, 2020

Abstract

In this report we will be analyzing the motion of a paint can attached to a string by performing Principle Component Analysis (PCA) on data gathered from video recordings of the system. First, I will implement an algorithm to track the position of the paint can through each video to gather usable data. Then, using the singular value decomposition (SVD) on the matrix created from these vectors, we can analyze the gathered data. The goal of using the SVD to analyze this data will be to determine how much correlation there is in the movement of the paint can in different directions and using this to find an ideal coordinate system to represent the motion of each system. Additionally, since every matrix has an SVD, we can use it to perform analysis, even on this overdetermined system. This report will show how the SVD and PCA can be used to represent the motion of systems in their ideal basis, and how noise and imperfect data affect our analysis.

1 Introduction and Overview

For this assignment, I will be performing PCA on the movement of a paint can attached to a spring. There are four different movements of the paint can, each recorded from three cameras placed in three different locations around the paint can.

- 1. Oscillations Only: The paint can only moves up and down in the "z direction"
- 2. Noisy Oscillations: The paint can is again only moving up and down, but the cameras are shaking
- 3. Pendulum and Oscillations: This time, the paint can is released off center, so it moves side to side in the x-y plane, like a pendulum, as well as vertical oscillations. There is motion in 2 directions.
- 4. Rotations, Pendulum, and Oscillations: This is the same as the third case, but the paint can is also rotating, creating motion in all three directions.

The first step will be to track the position of the paint can through all 12 videos. My technique will be to use the colors in the paint can to track its movement in each frame. To reduce computational time and improve accuracy, I will chose an initial spot by inspecting the first frame, then only look in a small window around that spot for where that color appears most strongly in the next frame.

Once we have the data for the x and y position of the paint can in the frame, we will use the SVD or singular value decomposition to diagonalize the dynamics. In other words, we want to find an ideal basis or coordinate system to describe the motion. Using the singular values, we can determine how many modes we need to represent the data.

2 Theoretical Background

This project will make use of Principle Component Analysis (PCA). To perform this type of analysis, I will be using the Singular Value Decomposition (SVD).

The SVD is a decomposition that breaks down a matrix into three matrices: U, Σ, and V^* where U and V^* describe rotations A performs on a vector $\vec{x}$ and Σ describes how much A stretches a vector $\vec{x}$.

One of the main reasons the SVD is so powerful is that every matrix has a singular value decomposition. It also allows you to represent any matrix as a sum of rank 1 approximations to that matrix. Additionally,

it has many applications since it diagonalizes the motion into a different orthogonal coordinate system. This is possible since it uses two different bases (U and V) to represent the data. When taking the SVD of a matrix, it can be interpreted as the correlation between the data.

To calculate the SVD, while that will not be necessary in the algorithm since MATLAB has its own SVD function, you first form the correlation matrix. Understanding this leads to better interpretability of the results. The correlation matrix looks like this:

$$AA^T = \begin{bmatrix} \sigma^2_{xa,xa}, \sigma^2_{xa,ya}, \cdots \\ \sigma^2_{ya,xa}, \sigma^2_{ya,ya} \cdot \cdots \\ \ddots, \ddots \end{bmatrix}$$

To calculate the SVD, you need both A^TA and AA^T
$A^TA = (U\Sigma V^*)^T(U\Sigma V*)$
$A^TA = V\Sigma U^*U\Sigma V^*$
$A^TA = V\Sigma^2V^*$
$A^TAV = V\Sigma^2$

The columns of V are the eigenvectors of A^TA.
A similar argument can be made for the matrix U, so the columns of U are the eigenvectors of AA^T. The singular values are the squares of the eigenvalues of either AA^T or A^TA.

When interpreting, the variance of a vector X_a is $\vec{X_a} \cdot \vec{X_a}$ and the dot product of two vectors that are not the same represents the covariance of those two. A large value means that the two vectors have a lot in common, whereas a small value means they have less in common, or there is not anything going on in that direction.

With three cameras providing information about the movement of the can, we will have 6 vectors of information for each case. Since we can have at most three directions of movement (x,y, and z directions), we have an overdetermined system of equations.

$$A = \begin{bmatrix} \texttt{xvec1} & \texttt{yvec1} & \texttt{xvec2} & \texttt{yvec2} & \texttt{xvec3} & \texttt{yvec3} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix},$$

For this specific situation we can interpret the SVD as follows:

- U (number frames x 6) representing the movement of the can in time
- Σ (6x6) The six diagonal entries represent the relative importance of each of the PCA modes, or the directions of the can's motion
- V (6x6) This has 6 vectors that represent the directions of motion.

When analyzing the matrices produced by the SVD, the main thing to interpret is the relative size of the diagonal elements of the matrix Σ, or the singular values of the matrix. Each large singular value represents the importance of the corresponding directions, or the first columns of U and V. In a case where data was perfect, we might expect that the number of nonzero singular values, or the rank would correspond to the number of important directions, however, since our data, and most data, contains noise, our matrix will have full rank. Looking at the columns of U and V will also help us determine what is happening in the system.

3 Algorithm Implementation and Development

3.1 Tracking the Can

The first part of this project is developing an algorithm to track the paint can through the frames of the video, shown in more detail in Algorithm 1. Since there is flashlight strategically located atop the can, for many of the videos, it is most convenient to track the whites spot or where all of the components of RGB

are at a maximum. To reduce runtime and number of computations, as well as decrease number of tracking errors, I manually located the whitest location on the can in the first frame. For the remainder of the video, the algorithm looks for the whitest spot in a small window around the can and then uses that to center the next window. The window size and shape are adjusted as needed to get optimal tracking on each video. This can be seen in Algorithm 1. At each frame the x and y positions of the spot are added to corresponding vectors to be used in the PCA analysis.

Algorithm 1: Function for Tracking the Paint Can

```
Find the whitest spot in the first frame by inspection and label [cx,cy]
Let wx and wy be the size of the window we will use
for Frame do
  for x = cx-wx :cx+wx do
    for  y = cy-wy:cy+wy do
      Find the whitest spot in this window of the frame
    end for
  end for
  Update the x and y position vectors each frame
end for
```

For some of the videos, the paint can is spinning, so it is difficult to always see the flashlight. In a few of these cases, I modified this algorithm to look for pink instead of white, but the same structures apply.

For a few of the videos, the spot the algorithm tracks is not exactly accurate, either due to camera shaking or rotations, which makes for noisy, imperfect data. Many of these problems arise since the paint can itself is white and it is moving in front of a white wall.

3.2 Performing Principle Component Analysis

Once we have the data collected, we want to perform PCA. To get all the vectors into a matrix, they first all must be the same size. For the majority of videos, I cropped each vector to the length of the shortest video by frame. Next, I subtracted the mean of each vector from the vector. This normalizes it so the mean is zero. Finally, construct the A matrix such that:

$$A = \begin{bmatrix} \texttt{xvec1} \\ \texttt{yvec1} \\ \texttt{xvec2} \\ \texttt{yvec2} \\ \texttt{xvec3} \\ \texttt{yvec3} \end{bmatrix}, A^T = \begin{bmatrix} \texttt{xvec1}, \texttt{yvec1}, \texttt{xvec2}, \texttt{yvec2}, \texttt{xvec3}, \texttt{yvec3} \end{bmatrix}$$

For the analysis on A, I take the reduced SVD of the A^T. To determine the number of relevant directions the can moves it, I plot the 6 singular values, both on a normal scale, as well as a loglog plot. To analyze the the directions, plots are made of the first few columns of U and V, or the first few PCA modes.

4 Computational Results

4.1 Part 1: Oscillations (see Figure 1)

This is the ideal case, where the can is only moving up and down in the z direction. Since the camera captures imperfect data, there will be oscillation in the x and y directions, however it should be fairly minimal. The noise in this case is due to human error in holding the camera and releasing the can off center. As expected, in Figure 1, there is one dominant singular value. This captures 71.78% of the energy of the system. For the PCA plot, V(1,1) and V(3,1) are close to zero, since these represent the horizontal motion components from the first two videos. V(2,1) and V(4,1) are larger since it represents the vertical motion from each video. The last two components or V are not as different, as in the last video, the paint can is moving sideways.

When analyzing the plot of the components of U, you can clearly see the oscillations in the first one in time, as the frames continue. It is really only necessary to look at the first mode, as most of the directionality can be captured by movement in the z direction.

4.2 Part 2: Noisy Oscillations (see Figure 2)

In this case, we also expect one dominant singular value that represents the oscillations. However, since the camera is noticeably shaking, it makes sense that the other singular values will be higher. The dominant singular value captures 40.12% of the energy in the system. The plots of the PCA modes are similar to the first case, however less clear due to the noise on the data.

4.3 Part 3: Pendulum and Oscillations: (see Figure 3)

For this case, the motion of the can should be able to be characterized by two dominant directions, and thus two significant singular values. However, with the data, it looks like there are three dominant singular values. This is likely due to errors in tracking, as well as likely small unintentional oscillations in the y direction. As the can spins a little bit, tracking the flashlight may create false movements in other directions. The two largest singular values capture 33.26% and 26% of the energy in the system respectively. Analyzing the U vectors, it looks like there are oscillations in 3 modes, when in reality, there should only be two.

4.4 Part 4: Rotations, Pendulum, and Oscillations: (see Figure 4)

Finally, in this last experiment, the can is moving in three different directions. It is moving in a circle in the x-y plane creating oscillations, as well oscillating in the z-direction. We would expect to need 3 modes to represent this data. From the data, we get 3 larger singular values. Analyzing the U vectors as time movement, you can sort of make out three oscillations. For these particular videos, the paint can was spinning, making it very difficult to track, as the flashlight turns away from the camera. This is likely the reason the data is not quite as accurate as would be preferable.

5 Summary and Conclusions

This assignment demonstrates the power of the SVD as a computational tool. Given only three different videos that only capture two dimensional data of the motion occurring, we can get a very good idea of what is happening using Principle Component Analysis. The first challenge in this assignment was to gather good data. After being able to track the paint can's location frame by frame in each video, we get 6 vectors representing the x and y positions of the can in each of the videos. Through analysis, our goal is to determine the most ideal coordinate system to represent our data in. This can be achieved using the SVD. Since the SVD is created by finding the eigenvalues and eigenvectors of the correlation matrix, it can be interpreted in such a way. By finding the correlation of the different vectors, it collapses the relevant directions necessary to represent the motion in the system. Identifying and plotting the singular values is the most useful in determining how may modes are necessary to capture the energy in the system. In our four cases, it was shown that the number of relevant singular values corresponded to the number of directions the can was moving in space. Unfortunately, in many cases, there was noise on the data, either as a result of shaking cameras, camera timing, or imperfect tracking. Despite this, the SVD can still pull out much of the relevant data needed to describe and diagonalize the motion of the system.

Figure 1 Case 1: Oscillations

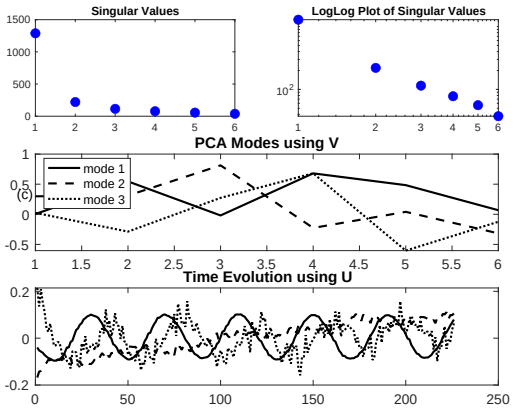


Figure 2 Case 2: Noisy Oscillations

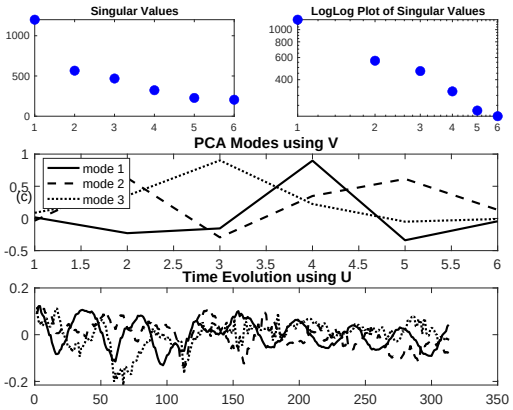


Figure 3 Case 1: Pendulum and Oscillations

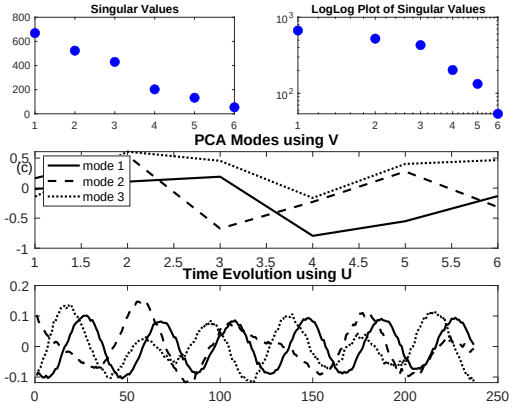
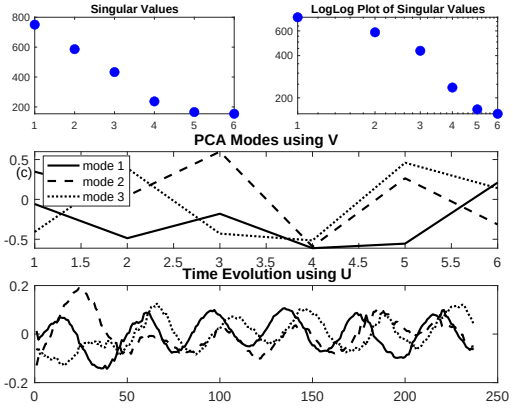


Figure 4 Case 4: Rotations, Pendulum, and Oscillations



Appendix A MATLAB Functions

- `[xvec,yvec] = paintCan(igx,igy,wx,wy,video)` This is a function I defined to help track the paint can. The algorithm used is defined above. It takes in `igx` and `igy` which are the x and y values of the initial guess of the location of the whitest spot. `wx` and `wy` are the width and height of the window where it will look for the next whitest spot. `video` is the video file to be analyzed.
- `[xvec,yvec] = paintCanRed(igx,igy,wx,wy,video)` This is a function I defined to help track the flashlight on top of the paint can. This is very similar to the one above, but looks for the reddest spot

rather than the whitest.

- `runVideo(xvec,yvec,video)` This is a short function used to play the video and plot the point tracking the can on each frame.
- `makePlotsA` This function takes in A and calculates the SVD. It uses the components of the SVD to make plots of the singular values as well as the modes for both the U and V vectors.
- `imshow` This displays an image of type uint8
- `mean()` This function calculates the average of a vector.
- `zeros(x,y)` This creates an x by y matrix of zeros.

Appendix B MATLAB Code

B.1 GITHUB LINK

https://github.com/shannondow/AMATH-582-Homeworks-2020

```
%582Homework3
%% Part 1:
clear all; clc;
%Load the first files from all three cameras:
load('cam1_1.mat')
load('cam2_1.mat')
load('cam3_1.mat')
% Camera1_1
%imshow(uint8(vidFrames1_1(:,:,:,1)))
[xvec1,yvec1] = paintCan(319,226,20,20,vidFrames1_1);
%runVideo(xvec1,yvec1,vidFrames1_1)
% Camera 2_1
%imshow(uint8(vidFrames2_1(:,:,:,1)))
[xvec2,yvec2] = paintCan(275,274,30,20,vidFrames2_1);
%runVideo(xvec2,yvec2,vidFrames2_1)
% Camera 3_1
%imshow(uint8(vidFrames3_1(:,:,:,1)))
[xvec3,yvec3] = paintCan(321,274,15,15,vidFrames3_1);
%runVideo(xvec3,yvec3,vidFrames3_1)
% SVDs
figure(1)
subplot(3,1,1)
plot(1:226,yvec1)
subplot(3,1,2)
plot(10:236,yvec2(10:236))
subplot(3,1,3)
plot(1:226,yvec3(1:226))
%Truncate so all vectors are the same length
xvec2 = xvec2(10:235);
yvec2 = yvec2(10:235);
xvec3 = xvec3(1:226);
yvec3 = yvec3(1:226);
%Subtract off the average for each vector:
xvec1 = xvec1 - mean(xvec1);
yvec1 = yvec1 - mean(yvec1);
xvec2 = xvec2 - mean(xvec2);
```

```
yvec2 = yvec2 - mean(yvec2);
xvec3 = xvec3 - mean(xvec3);
yvec3 = yvec3 - mean(yvec3);
%Put the Vectors into a matrix
A = zeros(6,226);
A(1,:) = xvec1';
A(2,:) = yvec1';
A(3,:) = xvec2';
A(4,:) = yvec2';
A(5,:) = xvec3';
A(6,:) = yvec3';
%Make Plots
figure(2)
makePlots(A)
%% Part 2:
clear all; clc;
%Load the first files from all three cameras:
load('cam1_2.mat')
load('cam2_2.mat')
load('cam3_2.mat')
% Camera1_2
%imshow(uint8(vidFrames1_2(:,:,:,1)))
[xvec1,yvec1] = paintCan(323,310,17,17,vidFrames1_2);
%runVideo(xvec1,yvec1,vidFrames1_2)
% Camera 2_2
%imshow(uint8(vidFrames2_2(:,:,:,1)))
[xvec2,yvec2] = paintCan(315,360,35,35,vidFrames2_2);
%runVideo(xvec2,yvec2,vidFrames2_2)
% Camera 3_2
%imshow(uint8(vidFrames3_2(:,:,:,1)))
[xvec3,yvec3] = paintCan(347,248,45,40,vidFrames3_2);
%runVideo(xvec3,yvec3,vidFrames3_2)
% SVDs
figure(3)
subplot(3,1,1)
plot(yvec1)
subplot(3,1,2)
plot(yvec2)
subplot(3,1,3)
plot(yvec3)
%Truncate so all vectors are the same length
xvec1 = xvec1(1:313);
yvec1 = yvec1(1:313);
xvec2 = xvec2(1:313);
yvec2 = yvec2(1:313);
xvec3 = xvec3(1:313);
yvec3 = yvec3(1:313);
%Subtract off the average for each vector:
xvec1 = xvec1 - mean(xvec1);
yvec1 = yvec1 - mean(yvec1);
xvec2 = xvec2 - mean(xvec2);
yvec2 = yvec2 - mean(yvec2);
xvec3 = xvec3 - mean(xvec3);
yvec3 = yvec3 - mean(yvec3);
```

```
%Put the Vectors into a matrix
A = zeros(6,313);
A(1,:) = xvec1';
A(2,:) = yvec1';
A(3,:) = xvec2';
A(4,:) = yvec2';
A(5,:) = xvec3';
A(6,:) = yvec3';
% Make Plots
figure(4)
makePlots(A)
%% Part 3:
clear all; clc;
%Load the first files from all three cameras:
load('cam1_3.mat')
load('cam2_3.mat')
load('cam3_3.mat')

% Camera1_3
%imshow(uint8(vidFrames1_3(:,:,:,1)))
[xvec1,yvec1] = paintCanRed(327,290,17,17,vidFrames1_3);
%runVideo(xvec1,yvec1,vidFrames1_3)
% Camera2_3
%imshow(uint8(vidFrames2_3(:,:,:,1)))
[xvec2,yvec2] = paintCanRed(237,290,17,17,vidFrames2_3);
%runVideo(xvec2,yvec2,vidFrames2_3)
% Camera3_3
%imshow(uint8(vidFrames3_3(:,:,:,1)))
[xvec3,yvec3] = paintCan(357,230,17,17,vidFrames3_3);
%runVideo(xvec3,yvec3,vidFrames3_3)
% SVDs
figure(5)
subplot(3,1,1)
plot(yvec1)
subplot(3,1,2)
plot(yvec2)
subplot(3,1,3)
plot(yvec3)
%Truncate so all vectors are the same length
xvec1 = xvec1(1:237);
yvec1 = yvec1(1:237);
xvec2 = xvec2(1:237);
yvec2 = yvec2(1:237);
xvec3 = xvec3(1:237);
yvec3 = yvec3(1:237);
%Subtract off the average for each vector:
xvec1 = xvec1 - mean(xvec1);
yvec1 = yvec1 - mean(yvec1);
xvec2 = xvec2 - mean(xvec2);
yvec2 = yvec2 - mean(yvec2);
xvec3 = xvec3 - mean(xvec3);
yvec3 = yvec3 - mean(yvec3);
%Put the Vectors into a matrix
A = zeros(6,237);
```

```
A(1,:) = xvec1';
A(2,:) = yvec1';
A(3,:) = xvec2';
A(4,:) = yvec2';
A(5,:) = xvec3';
A(6,:) = yvec3';
% Make Plots
figure(6)
makePlots(A)
%% Part 4:
clear all; clc;
%Load the first files from all three cameras:
load('cam1_4.mat')
load('cam2_4.mat')
load('cam3_4.mat')
% Camera1_3
%imshow(uint8(vidFrames1_4(:,:,:,1)))
[xvec1,yvec1] = paintCanRed(400,268,27,27,vidFrames1_4);
%runVideo(xvec1,yvec1,vidFrames1_4)
% Camera2_3
%imshow(uint8(vidFrames2_4(:,:,:,1)))
[xvec2,yvec2] = paintCanRed(241,244,17,17,vidFrames2_4);
%runVideo(xvec2,yvec2,vidFrames2_4)
% Camera3_3
%imshow(uint8(vidFrames3_4(:,:,:,1)))
[xvec3,yvec3] = paintCan(361,236,30,30,vidFrames3_4);
%runVideo(xvec3,yvec3,vidFrames3_4)
% SVDs
figure(7)
subplot(3,1,1)
plot(yvec1)
subplot(3,1,2)
plot(yvec2)
subplot(3,1,3)
plot(yvec3)
%Truncate so all vectors are the same length
xvec1 = xvec1(1:237);
yvec1 = yvec1(1:237);
xvec2 = xvec2(1:237);
yvec2 = yvec2(1:237);
xvec3 = xvec3(1:237);
yvec3 = yvec3(1:237);
%Subtract off the average for each vector:
xvec1 = xvec1 - mean(xvec1);
yvec1 = yvec1 - mean(yvec1);
xvec2 = xvec2 - mean(xvec2);
yvec2 = yvec2 - mean(yvec2);
xvec3 = xvec3 - mean(xvec3);
yvec3 = yvec3 - mean(yvec3);
%Put the Vectors into a matrix
A = zeros(6,237);
A(1,:) = xvec1';
A(2,:) = yvec1';
A(3,:) = xvec2';
```

```
A(4,:) = yvec2';
A(5,:) = xvec3';
A(6,:) = yvec3';
% Make Plots:
figure(8)
makePlots(A);
%% Functions

% paintCan:
function [xvec,yvec] = paintCan(igx,igy,wx,wy,video)
s = size(video);
maxloc = [igx,igy];
maxval = 0;
xvec = [];
yvec = [];
xvec(1) = maxloc(1);
yvec(1) = maxloc(2);
cx = maxloc(1);
cy = maxloc(2);
for frame = 2:s(4)
    for x = cx-wx:cx+wx
        for y = cy-wy:cy+wy
            find = video(y,x,:,frame);
            color = [find(1,1,1),find(1,1,2),find(1,1,3)];
            newmax = double(color(1))+double(color(2))+double(color(3));
            if newmax > maxval
                maxloc = [x,y];
                maxval = newmax;
            end
        end
    end
    %disp(maxloc);
    xvec(frame) = maxloc(1);
    yvec(frame) = maxloc(2);
    cx = maxloc(1);
    cy = maxloc(2);
    maxloc = [0,0];
    maxval = 0;
end

end

% paintCanRed
function [xvec,yvec] = paintCanRed(igx,igy,wx,wy,video)
s = size(video);
maxloc = [igx,igy];
maxval = 0;
xvec = [];
yvec = [];
xvec(1) = maxloc(1);
yvec(1) = maxloc(2);
cx = maxloc(1);
cy = maxloc(2);
for frame = 2:s(4)
```

```
253         for x = cx-wx:cx+wx
254             for y = cy-wy:cy+wy
255                 find = video(y,x,:,frame);
256                 color = [find(1,1,1),find(1,1,2),find(1,1,3)];
257                 newmax = double(color(1));
258                 if newmax > maxval && color(2) <175 && color(3) <175
259                     maxloc = [x,y];
260                     maxval = newmax;
261                 end
262             end
263         end
264         %disp(maxloc);
265         xvec(frame) = maxloc(1);
266         yvec(frame) = maxloc(2);
267         cx = maxloc(1);
268         cy = maxloc(2);
269         maxloc = [0,0];
270         maxval = 0;
271     end
272 
273     end
274 
275     %RunVideo
276     function runVideo(xvec,yvec,video)
277     s = size(video);
278     for i = 1:s(4)
279         figure(1)
280         imshow(uint8(video(:,:,:,i)))
281         hold on
282         plot(xvec(i),yvec(i),'*')
283         hold off
284     end
285 
286     end
287 
288     %makePlots
289     function makePlots(A)
290     [U,Sigma,V] = svd(A',0);
291     subplot(3,2,1)
292     plot(1:length(diag(Sigma)),diag(Sigma),'b.','MarkerSize',30)
293     title('Singular Values')
294     subplot(3,2,2)
295     loglog(1:length(diag(Sigma)),diag(Sigma),'b.','MarkerSize',30)
296     title('LogLog Plot of Singular Values')
297 
298     subplot(3,1,2)
299     plot(1:size(V),V(:,1),'k',1:size(V),V(:,2),'k--',1:size(V),V(:,3),'k:','
        Linewidth',[2]); set(gca,'Fontsize',[13]);
300     legend('mode 1','mode 2','mode 3','Location','NorthWest'); text(0.8,0.35,'(c)'
        ,'Fontsize',[13]);
301     title('PCA Modes using V');
302     subplot(3,1,3)
303     plot(1:size(U),U(:,1),'k',1:size(U),U(:,2),'k--',1:size(U),U(:,3),'k:','
        Linewidth',[2]); set(gca,'Fontsize',[13]);
```

```
304 title('Time Evolution using U');
305
306 %calculate how much each singular value captures the energy in the system
307 total = 0;
308 for i = 1:6
309     total = total + Sigma(i,i);
310 end
311
312 e1 = (Sigma(1,1)/total)*100;
313 e2 = (Sigma(2,2)/total)*100;
314 e3 = (Sigma(3,3)/total)*100;
315
316 disp('The percentage of energy captured by the first singular value is:');
317 disp(e1)
318 disp('The percentage of energy captured by the second singular value is:');
319 disp(e2)
320 disp('The percentage of energy captured by the third singular value is:');
321 disp(e3)
322 disp('The total enegry captured by the first three singular values is:');
323 disp(e1+e2+e3);
324
325 disp(V(:,1))
326
327 end
```