Khrisna Kamarga 2/26/2019

AMATH 482 Homework 2

Section 1. Introduction and Overview

In this homework assignment, we were given three videos of the oscillatory motion of a paint on a spring shot from three different locations and orientations. The goal of this analysis is to characterize the oscillatory motion of the paint bucket assuming that we do not know the best coordinate system and the governing equation of the motion. The purpose of this assignment is to learn Principal Component Analysis which is used to identify the dominant coordinate system that represents the motion the best.

We can find the principal components by using the Singular Value Decomposition, which arranges the covariance matrix of each input coordinates into diagonal matrix which corresponds to each of the principal direction. Once the diagonal matrix (S) is found, we can analyze its energy so that we can compare which components are the most dominant, which essentially conveys how much each component is distinct from each other.

There are 4 sections of the given videos: undisturbed vertical motion, noisy 1-D motion, both lateral and vertical motion, and lateral, vertical, and rotational motion. Performing PCA on each video type provides more information about how PCA works in distinguishing the dominant motions.

Section 2. Theoretical Background

The system is governed by a simple harmonic motion of a mass spring system. The expected solution of the system is a sinusoidal motion in the coordinate system that is aligned with the direction of gravity. We can use this knowledge to make sense of the result of each x-coordinate and y-coordinate motion.

We are interested to obtain the X and Y coordinate movement of the paint bucket in each video, which requires tracking the bucket. I used the previously implemented Frequency Domain Filtering technique to track the position of the bucket at each frame. Before the Fourier transform process, I cropped the video such that only the images around the bucket is recovered to avoid noise. This is done by making a filter that is 1 around those pixel values and 0 elsewere. Next, I Fourier transformed the whole video, obtaining the average frequency, of which the dominant frequency is retrieved to be the center of the filter. Then, each frame is Fourier transformed and multiplied by the filter. In the end, the coordinate of the paint bucket is recovered. Since the result is still noisy, I filtered it once more to smoothen out any wiggles in the motion.

After each of the motion in x and y is recovered, we can create a matrix consisting all the **x** and **y** data: **X** = [x1 y1; x2 y2; x3 y3]. We can then perform Singular Value Decomposition to this matrix which will return a pair of bases **U** and **V**. We can then obtain the principal components **(Y)** by projecting the basis **U** on **X** **(U'*X).** The energy plot can be obtained by squaring the components in the diagonal matrix **S** that contains the standard deviation of each row in the basis of principal components.

Section 3. Algorithm Implementation and Development

The MATLAB code for this assignment can be divided into 2 major parts: tracking the paint bucket and performing PCA to find the principal components.

I first loaded the video and assigned its size and timeframe to its own variables. To visualize the video, a for loop that uses pcolor to display each frame (with flipud) is used. Then, the crop filter, which is a matrix consisting 1s across a patch of pixel coordinates and 0 elsewhere is implemented. The pixel starting condition and ending condition where it encloses the paint bucket, which is manually found, is implemented by using a for loop that assigns 1 to the areas in between those pixels. Because each video has its unique crop filter, the filter start and end pixels are divided into sections so that the appropriate filter can be prepared with each video.

The next step is to accumulate all the frequency domain representation of each image to find the strongest frequency character that corresponds to a spot in the moving bucket. The algorithm implementation for this section is very similar with that of the previous homeworks. After the dominant frequency is found, we can apply a filter to only allow the dominant frequency component to be shown at each frame. In addition to frequency filtering, I also have eliminated the background of the video to reduce noise, by making a matrix that consists of the summation of all the frames in the time domain, then averaged it by dividing it with the maximum time frame. In the end, this matrix is subtracted from each frame to eliminate the static and repeating background of each frame. Ultimately, by finding the strongest component of the frequency, we can track the movement of the paint bucket. One thing to keep in mind in this frequency domain filtering is that all the coordinate systems need to be transposed in order to get an accurate result. The explanation for this extra step is because of the flipped coordinate system of images in MATLAB.

The resulting movement however, is still very noisy. Therefore, I took another step to refilter the resulting movement around the low frequency component (center of filter at 0 Hz). This will eliminate the high frequency wiggles in the result, and will smoothen the movement to give more acceptable result.

For each of the camera angle, the resulting workspace is saved in a '.mat' file to be used later.

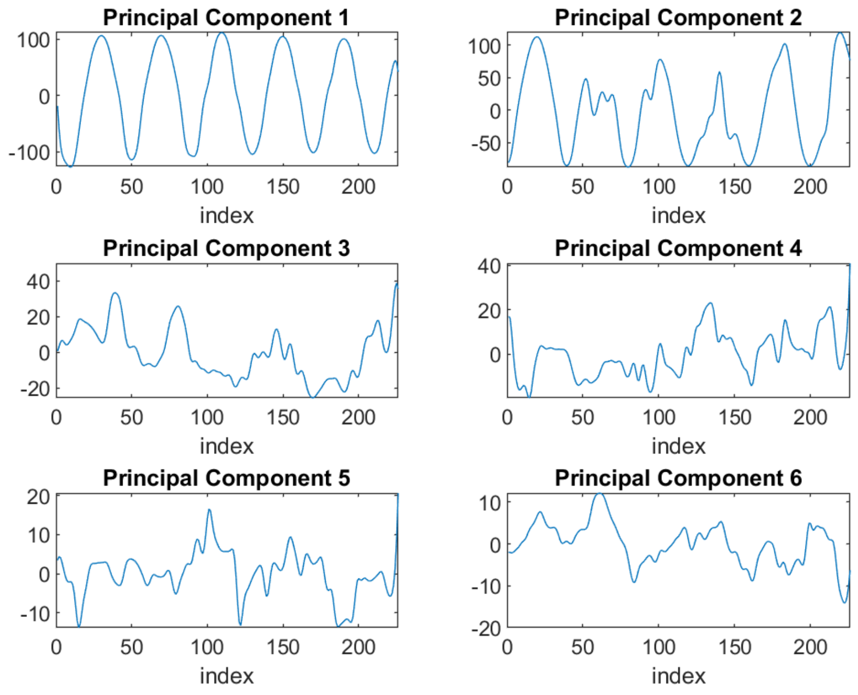
After each of the camera angle for one video type has been recovered we are ready to proceed to the PCA section. Upon loading the workspaces and only recovering x1, y1 through x3,y3, we need to ensure that each vectors are all column vectors to avoid dimension errors. We are now ready to make the X matrix containing all the movements: [x1 y1; x2 y2; x3 y3]. This is done by preparing a 6X226 matrix kernel. The columns are limited to 226 since each videos have different lengths, and this serves as a way to concatenate them into smaller and more consistent data points. We can then assign each rows of the matrix to its corresponding data.

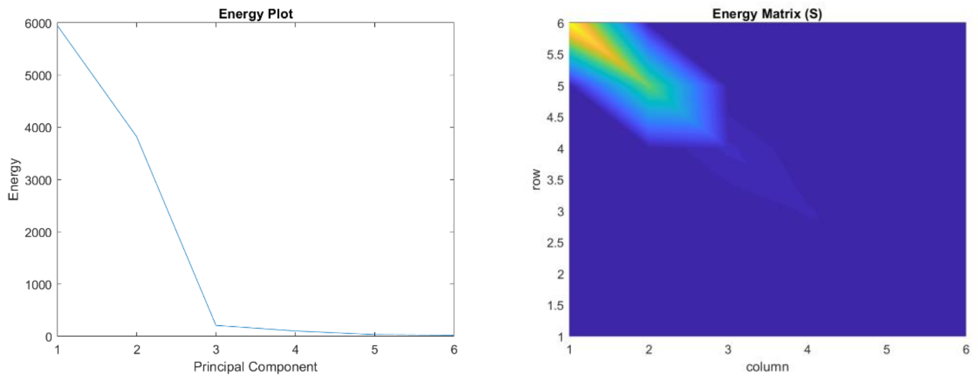
After recovering matrix X, I computed the mean, and reassigned X to itself subtracted by the mean to ensure that the coordinates oscillate around 0. We can then perform SVD function to (X/sqrt(n-1)). The components matrix is found by performing u'.*X to rearrange our data matrix into the principal components. Each of the principle components is then plotted to show its features. In addition, we can take the values of the diagonals in the S matrix corresponding to the standard deviation, and squaring each value to compute the covariance. I obtained the energy plot by plotting each of the values, and this plot can be used to determine the most unique components, determined by its larger energy magnitude. We can then store the final workspace containing the plots to each of the video type, which makes it easier to obtain the plots.

Section 4. Computational Results

(test 1) ideal case:

The principal components of test 1, and its energy plots are shown in the figure below.

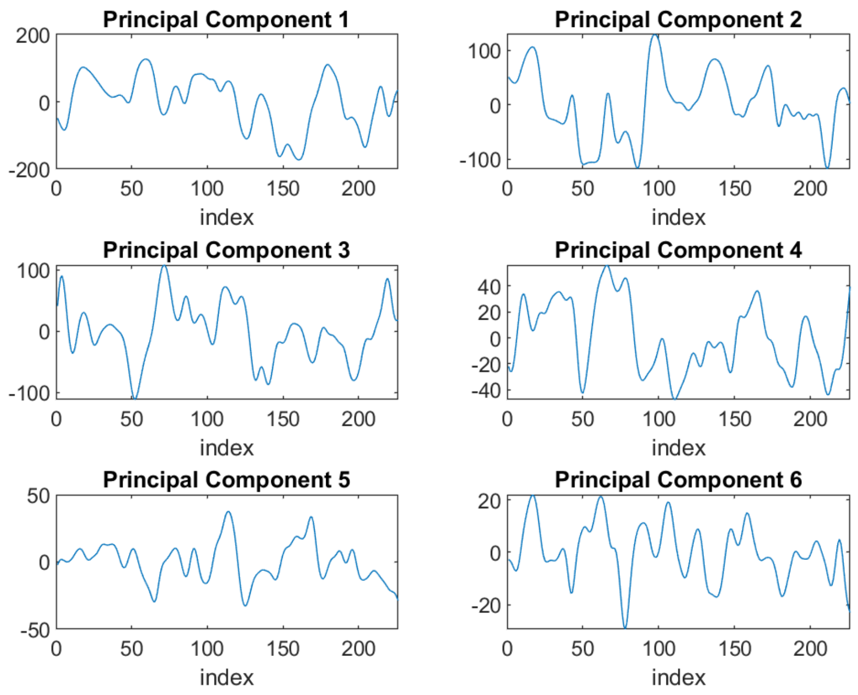


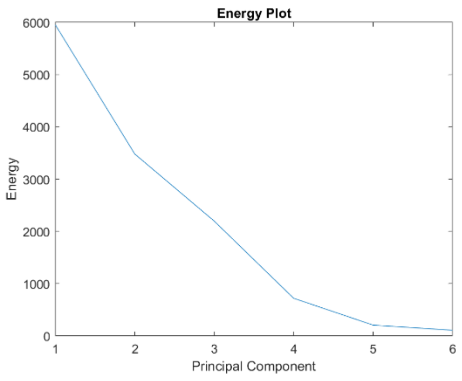


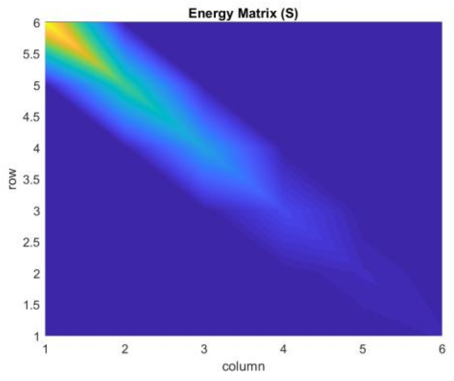
Principal Component 1 and Principal Component 2 describes the characteristic motion of the paint bucket, which is a sinusoidal motion with an amplitude of 100 pixels. According to the energy plot, Pricipal Component 1 (PC1) and PC2 have significantly higher energy than the rest of the PCs. This shows that the bucket's motion is uniquely characterized by PC1 and PC2. However, as expected from the solution from the equation of motion, there should only be one principal component, while we currently have 2. The explanation for this case is because there are two videos that are shot at a time delay between each other, resulting in similar motion which are phase shifted. As a result, PC1 and PC2 have the same shape, only that they are shifted. The rest of the PCs are not correlated too much from the actual motion of the paint bucket; they might pertain to the noise of the filter that is used to track the paint bucket.

(test 2) noisy case:

The principal components of test2, and its energy plots are shown in the figure below.



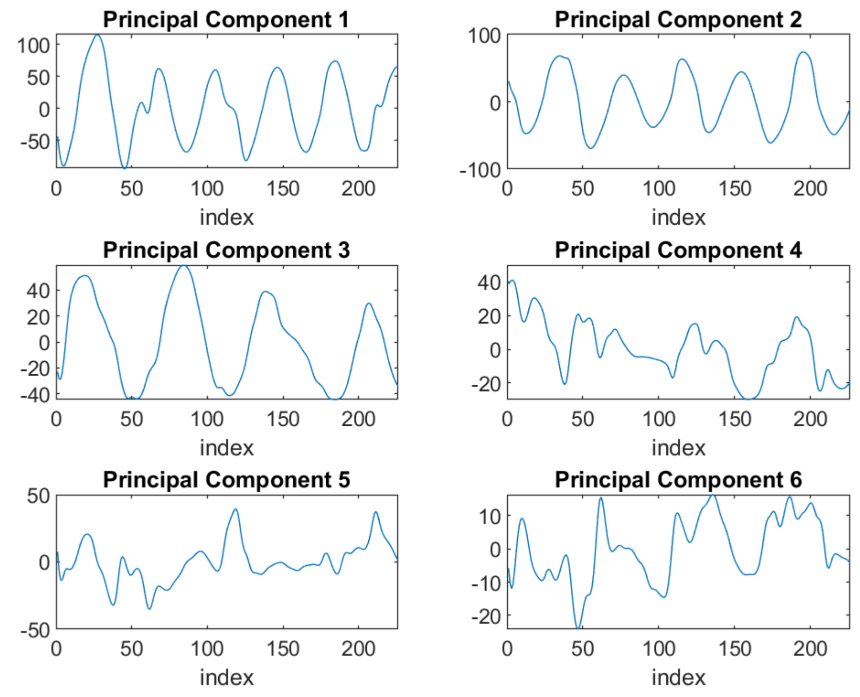


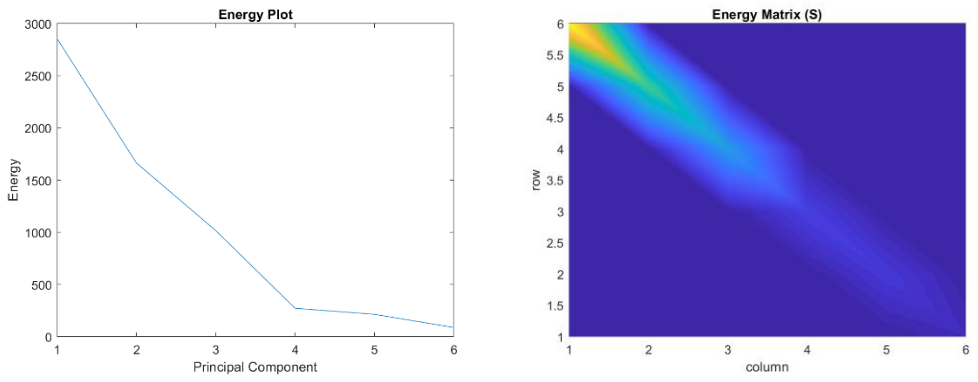


Test 2 by far has the least expected PC shapes from all the other tests. All the PCs are not really uniformly sinusoidal; each peaks seems to be at different magnitudes, which is not what the paint bucket undergoes. Looking at the energy graph, we can see clearer that there is no clear cutoff between the energies. These results are caused by the fact that the camera is shaky when they shot the videos. Since the shakiness or the noise of each camera is not uniform or similar to each other, SVD is not able to separate the noise into its own principal component. Instead, the noise will be embedded in each principal component, which renders the principal components less descriptive of the bucket movement. As we can see from the spectrogram, the diagonal is bright almost throughout the whole matrix which corresponds to the high covariance caused by the noise.

(test 3) horizontal displacement:

The principal components of test3, and its energy plots are shown in the figure below.

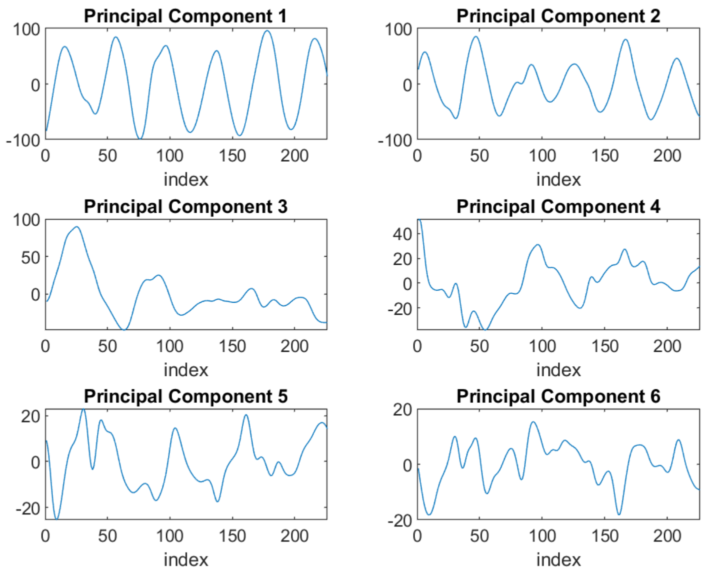


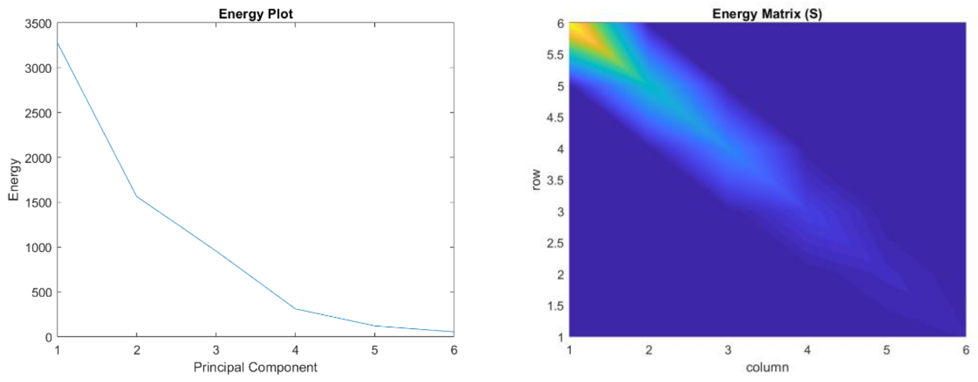


Test 3 has two dominant principal components with high energies. PC1 and PC2 do have similar maximum magnitude and they look correlated with each other. This means that PC2 is the phase shifted version of PC1 due to the delay in camera recording. In the energy plot, the energy of PC3 is still high, this is due to the distinct lateral/planar movement of the paint bucket. PC4 and onwards, not only have relatively lower energy that PC1 & PC2, but they also have lower maximum amplitude and more random motion, which are probably uncorrelated data pertaining to the disturbance of the motion.

(test 4) horizontal displacement and location)

The principal components of test4, and its energy plots are shown in the figure below.





Test 4 should have similar PCA results as that of test 3. We can see that the two main principal components are the first and the second components, which just like test 3, corresponds to the vertical and its phase shifted motion respectively. The energy correspoding to PC3 is high just like that of test 3. Upon closer inspection, PC3 looks like a decaying sinusoid, and after comparing with the video data, it is clear that PC3 describes lateral/planar motion since the lateral motion dies out towards the end of the video. From PC4 onwards, the energy is low enough that we can associate them to be contributed by noise.

Section 5. Summary and Conclusion

By performing PCA, we are able to find the projection of the data matrix we inputted to the principal components to find the unique components of the data that is meaningful for us. This is useful to represent a large data set that contains noise and has elements that might be redundant or highly correlated to each other with its principal components.

This assignment can be improved by having a better method to recover the position of the paint bucket at each frame. This assignment also provides an understanding of the principal components that corresponds to useful data extracted from a large redundant dataset, which is accomplished by analyzing the energy plot and comparing the principal components with the expected value.

Appendix A: MATLAB Functions Implementation Explanation

load <filename>

Loads the file to the workspace.

save <filename>

Saves the current workspace to a '.m' file

rgb2gray(image)

Converts RGB image into black and white image

flipud(image)

Flips the Y coordinate so that the data retains its orientation when plotted by pcolor.

pcolor(data), shading interp, colormap hot

Plots a m x n matrix and maps its elements into a color specified in the colormap with interpolated shading

repmat(mn, 1, n)

Makes an m x n matrix containing the number mn for each row

mean(X, 2)

Computes the mean for each row of the matrix.

svd(X)

Returns U, S, and V matrix that corresponds to the PCA matrices.

diag(S)

Takes the diagonal component of a matrix and returns a vector pertaining to the diagonal elements.

Appendix B: MATLAB Code

```
% AMATH 482 - HW3
% Khrisna Kamarga

% Setup - Load the Video and Prepare Variables
clear all; close all; clc;
load cam1_4.mat;
video = (vidFrames1_4);
[m, n, rgb, t] = size(video);

%% visualize the motion for each dataset
for i = 140:t
%     bwVideo = rgb2gray(video(:,:,:,i));
%     pcolor((bwVideo)), shading interp, colormap(gray);
    imshow(video(:,:,:,i)); drawnow; pause(0.05);
%     drawnow
end

%% crop location for 1_1 & 1_2 & 1_3 & 1_4
isolate = zeros(m,n);
jStart = 250;
iStart = 150;
jEnd = 450;
iEnd = 450;
%% 2_1 & 2_2 & 2_3 & 2_4
isolate = zeros(m,n);
jStart = 150;
iStart = 100;
jEnd = 400;
iEnd = 400;
%% 3_1 & 3_2 & 3_3
isolate = zeros(m,n);
jStart = 200;
iStart = 200;
jEnd = 500;
iEnd = 350;

%% 3_4
isolate = zeros(m,n);
jStart = 300;
iStart = 150;
jEnd = 500;
iEnd = 300;

%% create the crop filter
for i = iStart:iEnd
    for j = jStart:jEnd
        isolate(i,j) = 1;
    end
end
% crop filter preview
% clc;
% for i = 1:50
%     bwVideo = rgb2gray(video(:,:,:,i));
%     pcolor(flipud(double(bwVideo).*isolate)), shading interp, colormap(gray);
%     drawnow
% end

% find characteristic frequency
% set up the fourier coefficients
```

```
xaxis2 = linspace(0,m,m+1); x = xaxis2(1:m);
yaxis2 = linspace(0,n,n+1); y = yaxis2(1:n);
kx = (2*pi/(2*m))*[0:(m/2-1) -m/2:-1]; ksx = fftshift(kx);
ky = (2*pi/(2*n))*[0:(n/2-1) -n/2:-1]; ksy = fftshift(ky);
% set up the 3D coordinate points
[X,Y] = meshgrid(x,y); % spatial coordinates
[Kx,Ky] = meshgrid(ksx,ksy); % wave numbers
% transposing the coordinates since the pictures cause weird flips
X = X';
Y = Y';
Kx = Kx';
Ky = Ky';

UtnAve = zeros(m,n); % kernel for the averaged frequency domain signal
background = zeros(m,n); % background kernel
for i = 1:t
    bwVideo = double(rgb2gray(video(:,:,:,i)));
    Un = bwVideo; % gets the 2D coordinate representation of the sample
    background = background + double(Un);
    Utn = fftn(Un); % fourier transform of the data
    UtnAve = UtnAve + Utn; % cummulative sum of the frequency domain signal
end

%average background
background = background / t;

% %% plot the spectrogram
% % plot the resulting normalized averaged data in the frequency domain
% pcolor(abs(fftshift(UtnAve))/max(abs(UtnAve), [], 'all')), shading interp, colormap(gray);
% xlabel("Kx"); ylabel("Ky");
% title("Averaged Data in the Frequency Domain");

% find the indices of the max magnitude in the frequency domain
[ind1 ind2] = ind2sub([m,n], find(fftshift(UtnAve) == max(fftshift(UtnAve), [], 'all')));
% look up the frequency domain coordinate of the strongest signal
Kc = [Kx(ind1, ind2), Ky(ind1, ind2)];

% applying the filter and recovering x and y
close all; clc;
% 2D gaussian filter
tau = 100; % bandwith of the filter (good: 0.2)
[kux, kuy] = meshgrid(kx,ky); % unshifted wave numbers
kux = kux';
kuy = kuy';
filter = exp(-tau*((kux - Kc(1)).^2+(kuy - Kc(2)).^2));

bucket = zeros(t, 2); % kernel for the coordinates of the bucket
for i = 1:t
    bwVideo = double(rgb2gray(video(:,:,:,i)));
    Un = (bwVideo - background).*isolate; % gets the 2D coordinate representation of the
sample
    Utn = fftn(Un); %Utn = fftshift(Utn);
    UtnFilter = Utn.*filter; % filtered frequency domain signal
    UnFilter = real(ifftn(UtnFilter)); % obtain the spatial filtered data

%     % draw the resulting spatial filtered data
%     pcolor(flipud(abs(UnFilter)/max(abs(UnFilter), [], 'all')))
%     shading interp, colormap(gray);
%     grid on

    % find the coordinate of the center of the bucket
    [ind1 ind2] = ind2sub([m,n], find(abs(UnFilter) == max(abs(UnFilter), [], 'all')));
    bucket(i,:) = [X(ind1, ind2), Y(ind1, ind2)];
```

```
%     hold on
%     plot(bucket(i,2), -(bucket(i,1)-m), 'm.-', 'MarkerSize', 20);
%     hold off
%     drawnow
end
% %% plot the trajectory of the bucket
% plot(bucket(:,2), bucket(:,1), 'm.-', 'MarkerSize', 20);
% xlim([0 m]); ylim([0 n]);
% title("Trajectory of the bucket");
% xlabel("X"); ylabel("Y");
% grid on
% %% plot the decoupled x and y data
% subplot(2,1,1)
% plot(1:length(bucket(:,1)), bucket(:,1));
% title("X");
% subplot(2,1,2)
% plot(1:length(bucket(:,2)), bucket(:,2));
% title("Y");

% filter the noisy x and y
% set up the fourier coefficients
xaxis2 = linspace(0,t,t+1); xaxis = xaxis2(1:t);
k = [0:(t/2-1) -t/2:-1]; ks = fftshift(k);
% 3D gaussian filter
tau = 0.001; % bandwith of the filter (good: 0.2)
filter = exp(-tau*(k).^2);
filter = filter';

x = real(ifft(fft(bucket(1:length(k),2)).*filter));
y = real(ifft(fft(bucket(1:length(k),1)).*filter));

subplot(2,1,2)
plot(1:length(x), x)
title("X-axis displacement")
subplot(2,1,1)
plot(1:length(y), y)
title("Y-axis displacement")

clearvars -except x y

%% save variables
x1=x;
y1=y;
save('vid1')
%%
x2=x;
y2=y;
save('vid2')
%%
x3=x;
y3=y;
save('vid3')
%% PCA
clear all; clc; close all;
load('vid1.mat')
load('vid2.mat')
load('vid3.mat')
clearvars -except x1 x2 x3 y1 y2 y3
x1 = x1';
x2 = x2';
y2 = y2';
```

```
% maxSize = max([length(x1) length(x2) length(x3) length(y1) length(y2) length(y3)]);
maxSize = 226;
X = zeros(6,maxSize);
X(1,:) = x1(1:226);
X(2,:) = y1(1:226);
X(3,:) = x2(1:226);
X(4,:) = y2(1:226);
X(5,:) = x3(1:226);
X(6,:) = y3(1:226);

%%
clear all; clc;
load '4.mat'
[m,n]=size(X); % compute data size
mn=mean(X,2); % compute mean for each row
X=X-repmat(mn,1,n); % subtract mean
Cx=(1/(n-1))*X*X'; % covariance
[u,s,v]=svd(X/sqrt(n-1)); % perform the SVD
lambda=diag(s).^2; % produce diagonal variances
Y=u'*X; % produce the principal components projection

% figure(1)
% for j = 1:6
%     subplot(3,2,j);
%     plot(1:226,-Y(j,1:226));
%     title("Principal Component " + j);
% end

%%
hold on
figure(2)
plot(1:length(lambda), lambda, 'bo','LineWidth',3)

%%
% save('4')

%% Results
clear all; close all; clc;
load '3.mat'

%%
close all; clc;
figure(1)
for j = 1:6
    subplot(3,2,j);
    plot(1:226,Y(j,1:226));
    title("Principal Component " + j);
    xlabel("index");
end

figure(2)
plot(1:length(lambda), lambda);
title("Energy Plot")
xlabel("Principal Component")
ylabel("Energy")

figure(3)
Cy = cov((u'*X)');
pcolor(flipud(Cy)), shading interp, colormap
title("Energy Matrix (S)")
ylabel("row");
xlabel("column");
```