AMATH 582 Homework 3: PCA

Emma Nuss, github: emmashie

Abstract

Singular value decomposition (SVD) is a powerful tool that decomposes a matrix into three components. This tool can be applied to data to understand the principle components that drive the data's variation. This project applies SVD to video data of a spring-mass system for 4 different test cases: an ideal case, a case with noise, a case with horizontal movement of the mass, and a case with horizontal and rotational movement of the mass. SVD generally proves to be a useful tool in detecting the anticipated dynamics of the spring-mass system; however, noise and complicated movement limit its ability.

1. Introduction

Four test cases of a spring-mass system were captured with 3 videos of different camera angles. Each test case introduces different dynamics into the spring-mass system, from camera noise to additional mass movement. Singular value decomposition is applied to this dataset to better understand the dynamics of the mass's movement. A theoretical background on singular value decomposition and its application to this problem is outlined below.

2. Theoretical Background

2.1. Singular Value Decomposition

In linear algebra, singular value decomposition (SVD) factors any matrix into three matrices which specific properties that can be useful in data analysis. If you have a $m \times n$ matrix $\mathbf{A}$, you can decompose this matrix into:

$$\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^T \tag{1}$$

where, $\mathbf{U}$ and $\mathbf{V}$ are unitary matrices and Σ is a diagonal matrix with singular values (σ_i) along the diagonal, which are uniquely determined and non-negative real numbers. The singular values (σ_i) in Σ are ordered largest to smallest. The columns of $\mathbf{U}$ and $\mathbf{V}$ are known as the left-singular and right-singular vectors, respectively, and these vectors each form orthonormal bases.

SVD is a powerful tool in linear algebra and data analysis because you can use SVD to construct principle components of your matrix $\mathbf{A}$ and the associated variance. If $\mathbf{A}$ is a matrix of data, you can use $\mathbf{U}$ to produce the principal components projection of your data by taking $\mathbf{U}^T\mathbf{A}$, where the columns of this new matrix correspond to each mode. These modes each explain some amount of the original data. In addition, you can also can determine what proportion of the total variance each mode explains by using the singular values, σ_i. To compute the variance in each mode you can take σ_i^2. Since by the definition of the SVD, σ_i are ordered from largest to smallest, the first column of $\mathbf{U}^T\mathbf{A}$ corresponds to the largest proportion of the total variance.

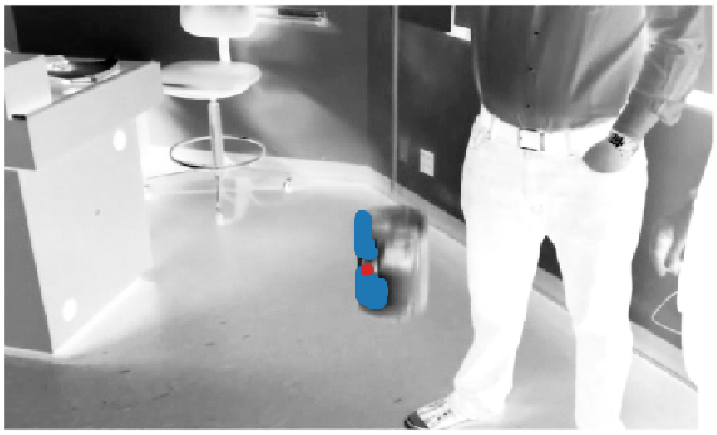

Figure 1: An example video frame from test case 1 (ideal case), camera angle 1 of the mass identifying method. Blue dots denote all the pixels above the threshold 0.95 within the specified region and the red dot denotes the center of mass calculated from the indices.

3. Algorithm Implementation and Development

3.1. Processing Video Data

Our dataset includes 3 video files, from 3 different camera angles, for 4 different cases, so a total of 12 video files. Each video depicts a mass (a large paint can) suspended on a spring and moving through time. The first step of our analysis is to determine the x and y position of the mass at each time step. First, the video files were converted to greyscale and normalized. Since the paint can has very dark spots on it, the mass can be tracked by finding the indices in the photo where the the normalized greyscale values are close to 1. The indices of pixels that had values above 0.95 were found at each video frame (Fig. 1). Due to dark objects in the background, the x-range was restricted to limit identifying pixels not associated with the mass. This restricted x-range was chosen by visual inspection to be centered around the mass. When a limited x-range was not a sufficient, a y-range was also added to restrict the region. The x and y indices within the restricted region that were above the 0.95 threshold were averaged at each time frame to find the center of mass (Fig. 1). The center of mass coordinates were compiled through time for each video and test case (Fig. 2). All the indices identified and the center of mass computed were plotted overlaying the video and animations were created and watched to confirm that the mass's position was reasonably identified.

After the x and y positions were found for each video, the positions for each video in a test case were plotted and inspected for obvious time syncing issues. If an offset was clearly discernible, the appropriate video's mass positions were shifted to match the other videos. In addition, the minimum number of frames between the three videos for a test case was found and position vectors were restricted to match and be equal to the minimum length. For each test case,

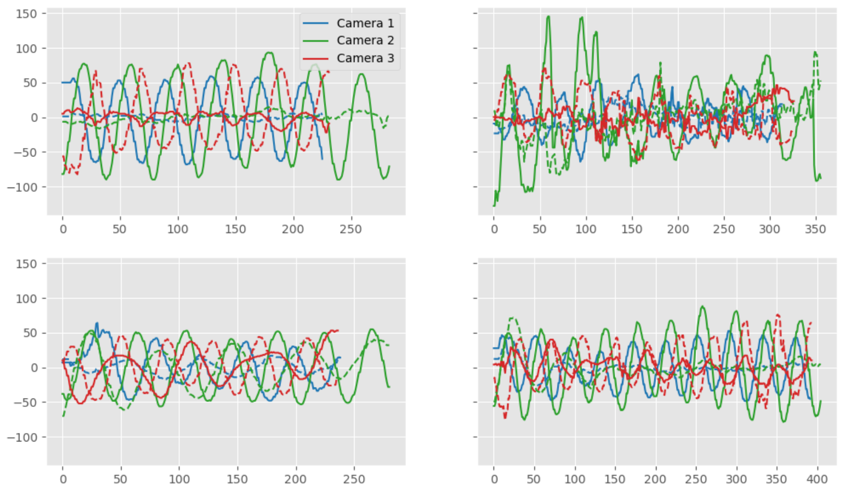

Figure 2: The raw positions of the mass identified from each camera angle for the ideal case, the noisy case, the horizontal displacement case, and the horizontal displacement and rotation case.

the can's x and y positions were compiled into a data matrix as:

$$\mathbf{X} = \begin{bmatrix} x_1(0) & \dots & x_1(n) \\ y_1(0) & \dots & y_1(n) \\ x_2(0) & \dots & x_2(n) \\ y_2(0) & \dots & y_2(n) \\ x_3(0) & \dots & x_3(n) \\ y_3(0) & \dots & y_3(n) \end{bmatrix} \tag{2}$$

where n is the minimum number of frames between the three videos. Lastly, SVD was computed for $\mathbf{X}$ and the principle components were computed from $\mathbf{U}^T\mathbf{X}$ and variance was calculated from Σ.

4. Computational Results

4.1. Ideal Case

In this case, all videos provide relatively clean data with clear x and y movement with little camera shaking or moving. The SVD of this data shows that the first mode clearly represents the harmonic nature of the moving mass in the video, with higher modes showing little motion (Fig. 3). While it is clear from visual inspection of the modes that the first mode explains a high proportion of the variance, from the computed variance we see that the first mode explains 95% of the variance and modes 2 through 6 each making up less than 5% of the variance (Fig. 4).

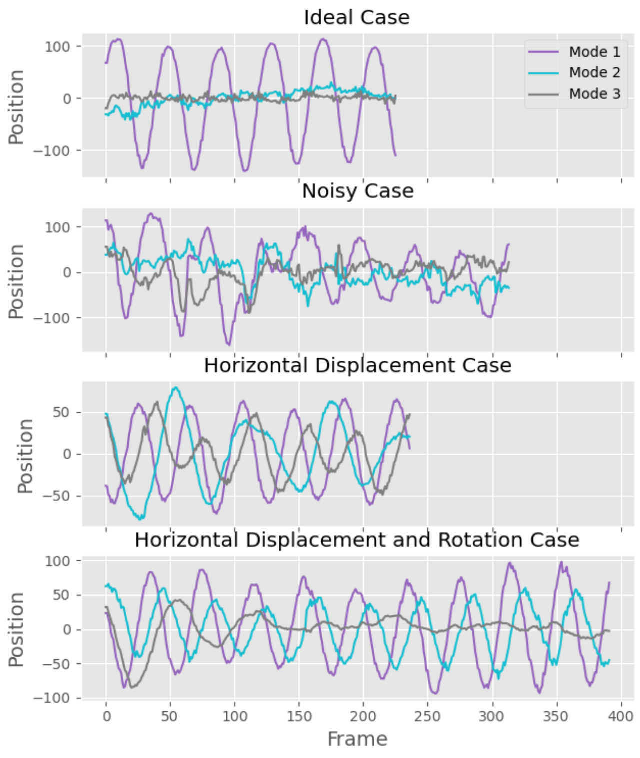

Figure 3: Modes 1, 2, and 3 reconstructed from SVD of the x and y positions of the mass in each video for all 4 test cases (Ideal, Noisy, Horizontal Displacement, Horizontal Displacement and Rotation).

4.2. Noisy Case

From the identified positions alone, it can be seen that the mass's harmonic movement is more difficult to identify from the raw data due to the noise of the data in this case; however, the SVD is still able to identify the mass's motion reasonably well, with 65% of the variance explained by the first mode (Fig. 4). The phase of the mass's movement is captured well and matches the ideal case, but the amplitude of the mass's movement varies through time and is not consistent with the ideal case (Fig. 3). This discrepancy is due to the movement of the camera recording the mass's movement by introducing noise to our data.

4.3. Horizontal Displacement Case

The horizontal displacement case is the worst case for applying SVD of the four test cases. The first mode only identifies 41% of the variance with the second mode explaining 35% of the variance (Fig. 4). The phase of the mass's movement does not appear to be consistent with the ideal case for either mode 1 or mode 2 (Fig. 3). The horizontal motion that is added to the vertical harmonic motion makes it much harder to clearly identify the mass's motion.

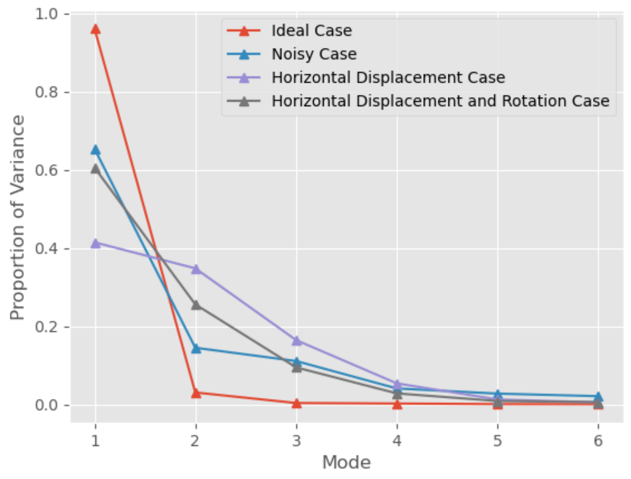

Figure 4:

4.4. Horizontal Displacement and Rotation Case

In the horizontal displacement and rotation case the mass moves horizontally and rotates which complicates the mass's movement. Through applying SVD 60% of the variance is captured in mode 1 and 25% of the variance is captured in mode 2 (Fig. 4). Initially as the mass moves horizontally and rotates, mode 3 is much higher in amplitude and is of the same scale as mode 1; however, when the mass slows its horizontal motion and mostly only rotates around its axis while moving up and down, mode 3 goes to zero and mode 1 appears very similar to our ideal case (Fig. 3).

5. Summary and Conclusions

SVD's ability to decompose a matrix into principle components and a measure of variance is useful in analyzing data. its ability to quantify the amount of variance explained by each component is useful as well as its ability to reconstruct each principle component. SVD works very well for the ideal case; however, camera movement induced noise makes identifying a clean signal difficult, but the phase is captured. Additional horizontal and rotational movement are much more difficult to clearly identify the mass's movement and multiple modes are needed to more fully explain the mass's movement. Despite its limitation, SVD is still a powerful tool and helpful in identifying patterns in a dataset and quantifying how important those variations might be.

Appendix A Python Functions

Relevant Python functions:

- `[U, Sdiag, VH] = np.linalg.svd(X)`: `Computes the SVD of matrix X where Sdiag is the diagonal components of` Σ `and VH is the transpose of` $\mathbf{V}$
- `np.intersect1d(x,y)`: Returns the intersect between two 1D arrays
- `x = np.arange(0, n)`: Creates a 1D array of integers starting at 0 up to `n-1`
- `z = np.append(x, y)`: Appends array y to the end of array x and stores it in variable z
- `ind = np.argwhere(D == condition) or np.where(D == condition)`: Finds the indices of an array where a logical condition is satisfied
- `np.asarray(D)`: Takes a list or some non-array and converts it to an array type
- `L = [i for i in range(n)]`: (List comprehension) Creates a list of `0` to `n-1`, can be used for high dimensions of lists and with if conditional statements
- `os.path.join(path, filename)`: Creates a path object based on the `path` and `filename` given
- `dat = sio.loadmat(filename)`: Loads a Matlab .mat file into Python

Appendix B Python Code

Code can be found at: https://github.com/emmashie/amath-582

```
import numpy as np
import matplotlib.pyplot as plt
import os
import scipy.io as sio
import hw3_functions as funct
plt.ion()
plt.style.use('ggplot')

# define data path
datapath = '../data/'

########### TEST 1 #############
# define filenames
cam11_file = 'cam1_1.mat'
cam21_file = 'cam2_1.mat'
cam31_file = 'cam3_1.mat'

# load mat files into python
cam11 = sio.loadmat(os.path.join(datapath, cam11_file))
cam21 = sio.loadmat(os.path.join(datapath, cam21_file))
cam31 = sio.loadmat(os.path.join(datapath, cam31_file))

# pull camera data from mat files
vid11 = cam11['vidFrames1_1']
vid21 = cam21['vidFrames2_1']
vid31 = cam31['vidFrames3_1']

# put camera data into greyscale
vid11_grey = np.asarray([funct.rgb2grey(vid11[:,:,:,i]) for i in range(len(vid11[0,0,0,:]))])
vid21_grey = np.asarray([funct.rgb2grey(vid21[:,:,:,i]) for i in range(len(vid21[0,0,0,:]))])
vid31_grey = np.asarray([funct.rgb2grey(vid31[:,:,:,i]) for i in range(len(vid31[0,0,0,:]))])

# find indices of mass from each frame
mindx11, mindy11 = funct.find_ind_and_plot(vid11_grey, 'plots/vid11_animation/fig', xmin=250, xmax=400, plot=False)
mindx21, mindy21 = funct.find_ind_and_plot(vid21_grey, 'plots/vid21_animation/fig', xmin=200, xmax=400, plot=False)
mindx31, mindy31 = funct.find_ind_and_plot(vid31_grey, 'plots/vid31_animation/fig', xmin=200, xmax=600, plot=False)

# plot
funct.plot_positions(mindx11, mindy11, mindx21, mindy21, mindx31, mindy31, '1')
funct.plot_positions(mindx11, mindy11, mindx21[10:], mindy21[10:], mindx31, mindy31, '1_shifted')

minlen = np.min([len(mindx11), len(mindx21[10:]), len(mindx31)])

X = np.zeros((6, minlen))
X[0,:] = mindx11[:minlen]
X[1,:] = mindy11[:minlen]
X[2,:] = mindx21[10:10+minlen]
X[3,:] = mindy21[10:10+minlen]
X[4,:] = mindx31[:minlen]
X[5,:] = mindy31[:minlen]

X = X - np.expand_dims(np.mean(X, axis=-1), axis=-1)

## python vs. matlab differences in svd function: https://stackoverflow.com/questions/50930899/svd-command-in-python-
[m, n] = X.shape
U, Sdiag, VH = np.linalg.svd(X)
```

```
V = VH.T

pca1 = np.matmul(U.T, X)
percentage1 = Sdiag**2/np.sum(Sdiag**2)

########### TEST 2 #############
# define filenames
cam12_file = 'cam1_2.mat'
cam22_file = 'cam2_2.mat'
cam32_file = 'cam3_2.mat'

# load mat files into python
cam12 = sio.loadmat(os.path.join(datapath, cam12_file))
cam22 = sio.loadmat(os.path.join(datapath, cam22_file))
cam32 = sio.loadmat(os.path.join(datapath, cam32_file))

# pull camera data from mat files
vid12 = cam12['vidFrames1_2']
vid22 = cam22['vidFrames2_2']
vid32 = cam32['vidFrames3_2']

# put camera data into greyscale
vid12_grey = np.asarray([funct.rgb2grey(vid12[:,:,:,i]) for i in range(len(vid12[0,0,0,:]))])
vid22_grey = np.asarray([funct.rgb2grey(vid22[:,:,:,i]) for i in range(len(vid22[0,0,0,:]))])
vid32_grey = np.asarray([funct.rgb2grey(vid32[:,:,:,i]) for i in range(len(vid32[0,0,0,:]))])

# find indices of mass from each frame
mindx12, mindy12 = funct.find_ind_and_plot(vid12_grey, 'plots/vid12_animation/fig', xmin=250, xmax=500, plot=False)
mindx22, mindy22 = funct.find_ind_and_plot(vid22_grey, 'plots/vid22_animation/fig', xmin=200, xmax=450, plot=False)
mindx32, mindy32 = funct.find_ind_and_plot(vid32_grey, 'plots/vid32_animation/fig', xmin=225, xmax=600, plot=False)

# plot
funct.plot_positions(mindx12, mindy12, mindx22, mindy22, mindx32, mindy32, '2')

# svd calculation
minlen = np.min([len(mindx12), len(mindx22), len(mindx32)])

X = np.zeros((6, minlen))
X[0,:] = mindx12[:minlen]
X[1,:] = mindy12[:minlen]
X[2,:] = mindx22[:minlen]
X[3,:] = mindy22[:minlen]
X[4,:] = mindx32[:minlen]
X[5,:] = mindy32[:minlen]

X = X - np.expand_dims(np.mean(X, axis=-1), axis=-1)

[m, n] = X.shape
U, Sdiag, VH = np.linalg.svd(X)
V = VH.T
#Xrank1_2 = np.matmul(np.expand_dims(U[:,0]*Sdiag[0],axis=-1), np.expand_dims(V[:,0], axis=-1).T)

pca2 = np.matmul(U.T, X)
percentage2 = Sdiag**2/np.sum(Sdiag**2)

########### TEST 3 #############
# define filenames
cam13_file = 'cam1_3.mat'
cam23_file = 'cam2_3.mat'
```

```
cam33_file = 'cam3_3.mat'

# load mat files into python
cam13 = sio.loadmat(os.path.join(datapath, cam13_file))
cam23 = sio.loadmat(os.path.join(datapath, cam23_file))
cam33 = sio.loadmat(os.path.join(datapath, cam33_file))

# pull camera data from mat files
vid13 = cam13['vidFrames1_3']
vid23 = cam23['vidFrames2_3']
vid33 = cam33['vidFrames3_3']

# put camera data into greyscale
vid13_grey = np.asarray([funct.rgb2grey(vid13[:,:,:,i]) for i in range(len(vid13[0,0,0,:]))])
vid23_grey = np.asarray([funct.rgb2grey(vid23[:,:,:,i]) for i in range(len(vid23[0,0,0,:]))])
vid33_grey = np.asarray([funct.rgb2grey(vid33[:,:,:,i]) for i in range(len(vid33[0,0,0,:]))])

# find indices of mass from each frame
mindx13, mindy13 = funct.find_ind_and_plot(vid13_grey, 'plots/vid13_animation/fig', xmin=250, xmax=450, plot=False)
mindx23, mindy23 = funct.find_ind_and_plot(vid23_grey, 'plots/vid23_animation/fig', xmin=200, xmax=450, ymin=200, ym
mindx33, mindy33 = funct.find_ind_and_plot(vid33_grey, 'plots/vid33_animation/fig', xmin=225, xmax=600, plot=False)

# plot
funct.plot_positions(mindx13, mindy13, mindx23, mindy23, mindx33, mindy33, '3')

# svd calculation
minlen = np.min([len(mindx13), len(mindx23), len(mindx33)])

X = np.zeros((6, minlen))
X[0,:] = mindx13[:minlen]
X[1,:] = mindy13[:minlen]
X[2,:] = mindx23[:minlen]
X[3,:] = mindy23[:minlen]
X[4,:] = mindx33[:minlen]
X[5,:] = mindy33[:minlen]

X = X - np.expand_dims(np.mean(X, axis=-1), axis=-1)

[m, n] = X.shape
U, Sdiag, VH = np.linalg.svd(X)
V = VH.T
pca3 = np.matmul(U.T, X)
percentage3 = Sdiag**2/np.sum(Sdiag**2)


########### TEST 4 #############
# define filenames
cam14_file = 'cam1_4.mat'
cam24_file = 'cam2_4.mat'
cam34_file = 'cam3_4.mat'

# load mat files into python
cam14 = sio.loadmat(os.path.join(datapath, cam14_file))
cam24 = sio.loadmat(os.path.join(datapath, cam24_file))
cam34 = sio.loadmat(os.path.join(datapath, cam34_file))

# pull camera data from mat files
vid14 = cam14['vidFrames1_4']
vid24 = cam24['vidFrames2_4']
```

```
vid34 = cam34['vidFrames3_4']

# put camera data into greyscale
vid14_grey = np.asarray([funct.rgb2grey(vid14[:,:,:,i]) for i in range(len(vid14[0,0,0,:]))])
vid24_grey = np.asarray([funct.rgb2grey(vid24[:,:,:,i]) for i in range(len(vid24[0,0,0,:]))])
vid34_grey = np.asarray([funct.rgb2grey(vid34[:,:,:,i]) for i in range(len(vid34[0,0,0,:]))])

# find indices of mass from each frame
mindx14, mindy14 = funct.find_ind_and_plot(vid14_grey, 'plots/vid14_animation/fig', xmin=300, xmax=500, plot=False)
mindx24, mindy24 = funct.find_ind_and_plot(vid24_grey, 'plots/vid24_animation/fig', xmin=215, xmax=400, plot=False)
mindx34, mindy34 = funct.find_ind_and_plot(vid34_grey, 'plots/vid34_animation/fig', xmin=200, xmax=600, plot=False)

# plot
funct.plot_positions(mindx14, mindy14, mindx24, mindy24, mindx34, mindy34, '4')

# svd calculation
minlen = np.min([len(mindx14), len(mindx24), len(mindx34)])

X = np.zeros((6, minlen))
X[0,:] = mindx14[:minlen]
X[1,:] = mindy14[:minlen]
X[2,:] = mindx24[:minlen]
X[3,:] = mindy24[:minlen]
X[4,:] = mindx34[:minlen]
X[5,:] = mindy34[:minlen]

X = X - np.expand_dims(np.mean(X, axis=-1), axis=-1)

[m, n] = X.shape
U, Sdiag, VH = np.linalg.svd(X)
V = VH.T
pca4 = np.matmul(U.T, X)
percentage4 = Sdiag**2/np.sum(Sdiag**2)


### analysis figures ###
fig, ax = plt.subplots(figsize=(12,7), nrows=2, ncols=2, sharex=False, sharey=True)
ax[0,0].plot(mindx11-np.mean(mindx11), '--',  color='tab:blue')
ax[0,0].plot(mindy11-np.mean(mindy11),label='Camera 1', color='tab:blue')
ax[0,0].plot(mindx21-np.mean(mindx21), '--', color='tab:green')
ax[0,0].plot(mindy21-np.mean(mindy21), label='Camera 2', color='tab:green')
ax[0,0].plot(mindx31-np.mean(mindx31), '--', color='tab:red')
ax[0,0].plot(mindy31-np.mean(mindy31), label='Camera 3', color='tab:red')
ax[0,0].legend(loc='best')
ax[0,0].set_title('Ideal Case')

ax[0,1].plot(mindx12-np.mean(mindx12), '--',  color='tab:blue')
ax[0,1].plot(mindy12-np.mean(mindy12),label='Camera 1', color='tab:blue')
ax[0,1].plot(mindx22-np.mean(mindx22), '--', color='tab:green')
ax[0,1].plot(mindy22-np.mean(mindy22), label='Camera 2', color='tab:green')
ax[0,1].plot(mindx32-np.mean(mindx32), '--', color='tab:red')
ax[0,1].plot(mindy32-np.mean(mindy32), label='Camera 3', color='tab:red')
ax[0,1].set_title('Noisy Case')

ax[1,0].plot(mindx13-np.mean(mindx13), '--',  color='tab:blue')
ax[1,0].plot(mindy13-np.mean(mindy13),label='Camera 1', color='tab:blue')
ax[1,0].plot(mindx23-np.mean(mindx23), '--', color='tab:green')
ax[1,0].plot(mindy23-np.mean(mindy23), label='Camera 2', color='tab:green')
ax[1,0].plot(mindx33-np.mean(mindx33), '--', color='tab:red')
```

```
ax[1,0].plot(mindy33-np.mean(mindy33), label='Camera 3', color='tab:red')
ax[1,0].set_title('Horizontal Displacement Case')

ax[1,1].plot(mindx14-np.mean(mindx14), '--',  color='tab:blue')
ax[1,1].plot(mindy14-np.mean(mindy14),label='Camera 1', color='tab:blue')
ax[1,1].plot(mindx24-np.mean(mindx24), '--', color='tab:green')
ax[1,1].plot(mindy24-np.mean(mindy24), label='Camera 2', color='tab:green')
ax[1,1].plot(mindx34-np.mean(mindx34), '--', color='tab:red')
ax[1,1].plot(mindy34-np.mean(mindy34), label='Camera 3', color='tab:red')
ax[1,1].set_title('Horizontal Displacement and Rotation Case')
fig.savefig('plots/positions.png')


fig, ax = plt.subplots(figsize=(7,8.5), nrows=4, sharex=True)
ax[0].plot(pca1[0,:], label='Mode 1', color='tab:purple')
ax[0].plot(pca1[1,:], label='Mode 2', color='tab:cyan')
ax[0].plot(pca1[2,:], label='Mode 3', color='tab:gray')
ax[0].set_title('Ideal Case')
ax[0].legend(loc='best')
ax[0].set_ylabel('Position', fontsize=14)

ax[1].plot(pca2[0,:], label='Mode 1', color='tab:purple')
ax[1].plot(pca2[1,:], label='Mode 2', color='tab:cyan')
ax[1].plot(pca2[2,:], label='Mode 3', color='tab:gray')
ax[1].set_title('Noisy Case')
#ax[1].legend(loc='best')
ax[1].set_ylabel('Position', fontsize=14)

ax[2].plot(pca3[0,:], label='Mode 1', color='tab:purple')
ax[2].plot(pca3[1,:], label='Mode 2', color='tab:cyan')
ax[2].plot(pca3[2,:], label='Mode 3', color='tab:gray')
ax[2].set_title('Horizontal Displacement Case')
#ax[2].legend(loc='best')
ax[2].set_ylabel('Position', fontsize=14)

ax[3].plot(pca4[0,:], label='Mode 1', color='tab:purple')
ax[3].plot(pca4[1,:], label='Mode 2', color='tab:cyan')
ax[3].plot(pca4[2,:], label='Mode 3', color='tab:gray')
ax[3].set_title('Horizontal Displacement and Rotation Case')
ax[3].set_xlabel('Frame', fontsize=14)
ax[3].set_ylabel('Position', fontsize=14)
#ax[3].legend(loc='best')
fig.savefig('plots/PCA_comparison_modes.png')


fig, ax = plt.subplots()
ax.plot(np.arange(1,7,1), percentage1, '-^', label='Ideal Case')
ax.plot(np.arange(1,7,1), percentage2, '-^', label='Noisy Case')
ax.plot(np.arange(1,7,1), percentage3, '-^', label='Horizontal Displacement Case')
ax.plot(np.arange(1,7,1), percentage4, '-^', label='Horizontal Displacement and Rotation Case')
ax.legend(loc='best')
ax.set_xlabel('Mode')
ax.set_ylabel('Proportion of Variance')
fig.savefig('plots/variance.png')

###### python class for hw3 #####
#!/usr/bin/env python
""" functions used for AMATH 582 hw3
"""
```

```
import numpy as np
import matplotlib.pyplot as plt

def norm(x):
        return x/np.nanmax(x)

def rgb2grey(rgb):
    """ convert rgb data to greyscale
    """
    grey = 0.2989 * rgb[:,:,0] + 0.5870 * rgb[:,:,1] + 0.1140 * rgb[:,:,2]
    return norm(grey)

def plotim_w_max(vid, indx, indy, avg_indx, avg_indy, path):
        fig, ax = plt.subplots()
        p = ax.pcolormesh(vid[::-1,:], cmap='Greys')
        #fig.colorbar(p, ax=ax)
        ax.scatter(indx, indy, color='tab:blue')
        ax.scatter(avg_indx, avg_indy, color='tab:red')
        ax.get_xaxis().set_visible(False)
        ax.get_yaxis().set_visible(False)
        fig.savefig(path)
        plt.close()

def find_ind_and_plot(vid, path, xmin=250, xmax=400, ymin=0, ymax=500, plot=True, restricty=False):
        sindx = []
        sindy = []
        for i in range(len(vid)):
                ind = np.where(vid[i,:,:]>0.95)
                subindx = np.where((ind[-1]<xmax)&(ind[-1]>xmin))
                if restricty==True:
                        subindy = np.where((ind[0]<ymax)&(ind[0]>ymin))
                        subind = np.intersect1d(subindx, subindy)
                if restricty==False:
                        subind=subindx
                indx = ind[-1][subind]
                indy = len(vid[i,:,:]) - ind[0][subind]
                avg_indx = int(np.mean(indx))
                avg_indy = int(np.mean(indy))
                sindx.append(avg_indx)
                sindy.append(avg_indy)
                if plot==True:
                        plotim_w_max(vid[i,:,:], indx, indy, avg_indx, avg_indy, path+'%05d' % i)
        return sindx, sindy

def plot_positions(mindx11, mindy11, mindx21, mindy21, mindx31, mindy31, test_num):
        fig, ax = plt.subplots(figsize=(8,5))
        ax.plot(mindx11, '--',  color='tab:blue')
        ax.plot(mindy11,label='Camera 1', color='tab:blue')
        ax.plot(mindx21, '--', color='tab:green')
        ax.plot(mindy21, label='Camera 2', color='tab:green')
        ax.plot(mindx31, '--', color='tab:red')
        ax.plot(mindy31, label='Camera 3', color='tab:red')
        ax.legend(loc='best')
        ax.set_ylabel('X & Y position', fontsize=16)
        ax.set_xlabel('Frame', fontsize=16)
        fig.savefig('plots/test%s_positions.png' % test_num)
```