Assignment 2 Math 494

Alexander Vakkas

March 7, 2023

Abstract

The goal of this assignment is to use video data from three different cameras to extract mass positions from video frames in order to perform the Principal Component Analysis (PCA) method. After applying PCA to the mass positions, extract the time evolution from only the first two dominant principal components. Then, using the Sparse Identification of Nonlinear Dynamics (SINDy) method, try to find the equations of motion that govern the dynamics of this system.

1 Introduction and Overview

Despite the fact that Principal Component Analysis (PCA) is one of the most well-known and oldest multivariate analysis techniques. It, like many other multivariate methods, was not widely used until the introduction of modern computers, but it is now well embedded in virtually every statistical computer package available today. In this assignment, we are attempting to generate equations that describe the motion of a mass attached to a string. Our data consists of three videos that show the same movement of a bucket attached to a string. We extract the displacement of the bucket in the x and y directions based on the position of each camera from these videos. Using PCA, we identify two principal components and use them as a new foundation for our data, attempting to recover the second-order differential equation that characterizes such a motion. We then use the Sparse Identification of Nonlinear Dynamics (SINDy) method to try and learn equations of motion that govern the dynamics of the system, which is comprised of video recording data, using data from PCA. We perform these maneuvers twice, once with data with minimal noise and again with noise intentionally added [2].

2 Theoretical Background

2.1 Principal Component Analysis

The central idea behind the Principal Component Analysis (PCA) is to reduce the dimensionality of a data set with a large number of interrelated variables while retaining as much variation as possible. This is accomplished by transforming to a new set of variables, the principal components, which are uncorrelated and ordered in such a way that the first few retain the majority of the variation present in all of the original variables. The solution of an Eigenvalue decomposition problem for a positive symmetric matrix is required for computing the principal components.

Now imagine we have multiple vectors $x_1, x_2, x_3, andx_4$. Each vector is made up of a single row of length n and is then put into a data matrix X.

$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \tag{1}$$

With this matrix X, we can compute all its variances and covariances between its rows;

$$C_x = \begin{bmatrix} \sigma^2_{x_1} & \sigma^2_{x_1x_2} & \sigma^2_{x_1x_3} & \sigma^2_{x_1x_4} \\ \sigma^2_{x_2x_1} & \sigma^2_{x_2x_2} & \sigma^2_{x_2x_3} & \sigma^2_{x_2x_4} \\ \sigma^2_{x_3x_1} & \sigma^2_{x_3x_2} & \sigma^2_{x_3x_3} & \sigma^2_{x_3x_4} \\ \sigma^2_{x_4x_1} & \sigma^2_{x_4x_2} & \sigma^2_{x_4x_3} & \sigma^2_{x_4} \end{bmatrix}. \tag{2}$$

The variance of the data is a measure of its spread, and the covariance of the variables tells us how they relate to one another. If the covariance is high this indicates that the variables are identical, whereas a covariance of zero indicates statistical independence [1]. Now we wish to find a new set of coordinates that will yield unrelated variables. This means that each variable now contains entirely new information that can be used to find the highest variance. The reason for this is that they contain the most critical information about our data . Then we will need to diagonalize Cx to get rid of all values above and below the diagonal entries. We can use two methods at this step: Eigenvalue Decomposition or Singular Value Decomposition (SVD) [1]. With this, our data can now be used.

2.2 Sparse Identification of Nonlinear Dynamics

The SINDy method is a sparse identification approach for nonlinear dynamical systems that combines the dynamical system discovery problem with a statistical regression problem. The SINDy method attempts to find the best dynamic system f for the data $\dot{X} = f(x)$. This function approximation problem is expressed as linear regression $\dot{X} = \Theta(X)\Xi$, with coefficients Ξ and a regression term library $\Theta(X)$. To begin, we generate data from X, a dynamical system, and then compute its derivatives $\dot{X}$. $\Theta(X)$ is then defined as a row of column matrices composed of theta functions applied to the data. Finally, we use the previous data to calculate the coefficients Ξ. In short, we used data to build a model that we can now use and get a sparse and accurate solution.

$$INSERTFORMULAS \tag{3}$$

3 Algorithm Implementation and Development

We begin by downloading the video array files and converting them from Mathlab files to python. After we are done loading the video data matrices, we reconstruct the videos so that we may begin to observe their motion. We notice that each video was shot from a different angle and depicts a bucket hopping up and down. To track the motion in these videos we will use the legacy motion tracker package from openCV. After going through each of the trackers we had available (or the ones we could use; some did not work), we were able to get the best results using the "MIL" tracker. The "MIL" tracker is also the best to use when dealing with high amounts of noise. We continue to use openCV to display the first frame of each video and draw a rectangle around the bucket. These rectangles that we draw manually will be tracking the motion of the bucket and give us an array of tracked data. This will be done three separate times, one for each video. The data gathered from the motion tracking rectangle are saved as row variables, which are then used to make the X data matrix. We then subtract the mean from each row of X and create a reduced matrix X. We then calculate the SVD using numpy, one of Python's many libraries of functions, to find the two principal components that we will be using. The reason we take only the first two is that they have most of the data we need. Now we use pysindy which is Python's library for the SINDy method. We take the transpose of our reduced matrix X, then we use the library of commands as seen in the code to find the models. The models are derivatives and so we integrate and find our results.

4 Computational Results

The results were very fruitful for this assignment. We were successful in tracking the buckets at all the three different angles, as seen in figure 1. The PCA results, shown in figure 2, track the position of the bucket and its velocity. This data then was manipulated by the SINDy method to produce the models shown on the graph in figure 3.

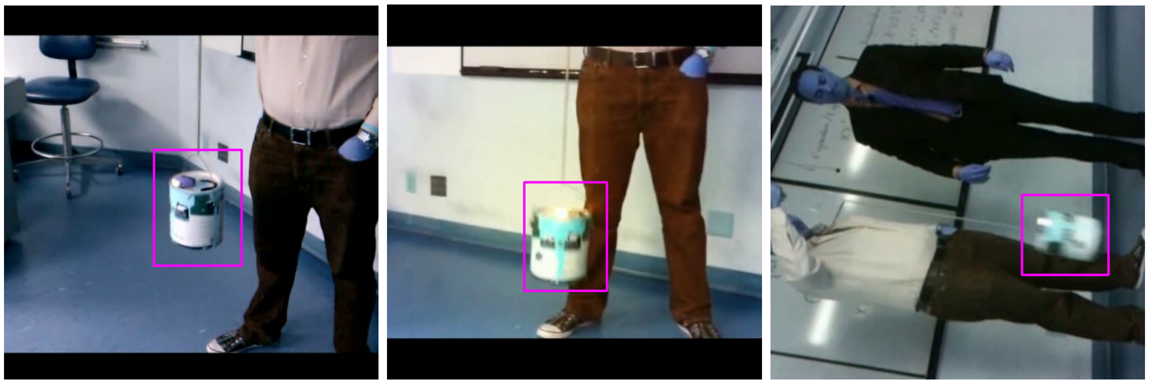

Figure 1: Tracking of the bucket

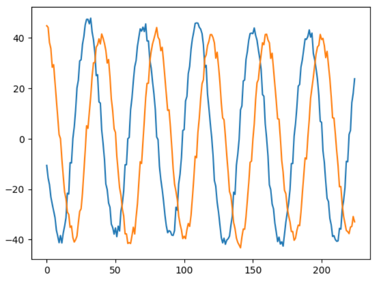

Figure 2: PCA Graph

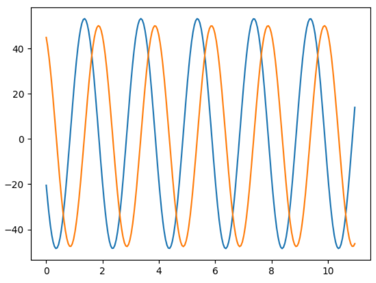

Figure 3: SINDy Results

5 Summary and Conclusions

We utilized an arsenal of different trackers and chose the best one "MIL". We calculated the Principal Component Analysis using Singular Value Decomposition. Lastly, with the models given from our PCA we were able to calculate the models we wanted using the Sparse Identification of Nonlinear Dynamics.

References

[1] Jason Bramburger. *Data-Driven Methods for Dynamical Systems: A new perspective on old problems.* Concordia University, 2023.

[2] I.T. Jolliffe. *Principal Component Analysis, Second Edition.* Springer, 2002.

Appendix A Python Functions

- `u,s,vt = np.linalg.svd(X)`: calculates the SVD of a matrix array.
- `model = ps.SINDy(featurenames = featureNames, optimizer = opt)`: calculates the derivative equations from the data of the PCA using the pysindy package.
- `z = odeint(function, z0, t)`: calculates ordinary differential equations.

Appendix B Python Code

```
#INSTALLS

!pip install numpy
!pip install scipy
!pip install matplotlib
!pip install opencv-python

import numpy as np
import scipy.io
import matplotlib.pyplot as plt
from PIL import Image as im
import cv2 as cv

#DOWNLOAD VIDEO

#For each camera recording, we extract the videos from mat files
from scipy.io import loadmat
matrix1 = loadmat('cam1_1.mat')
matrix2 = loadmat('cam2_1.mat')
matrix3 = loadmat('cam3_1.mat')

#We convert the data into python arrays
datam1 = matrix1["vidFrames1_1"]
datam2 = matrix2["vidFrames2_1"]
datam3 = matrix3["vidFrames3_1"]

#We determine the dimensions of each of the cameras to see what dimensions our data matrix have
print(datam1.shape)
print(datam2.shape)
print(datam3.shape)

#MOTION TRACKING

#We write the videos necessary to capture the motion from each video

#ANGLE 1:

height, width,_,nbFrames = datam1.shape
codec_id = "mp4v" # ID for a video codec.
fourcc = cv.VideoWriter_fourcc(*codec_id)
filename = "video1.mp4"
out = cv.VideoWriter(filename, fourcc=fourcc, fps=20, frameSize=(width, height))


for i in range(nbFrames):
    out.write(datam1[:,:,:,i])


#ANGLE 2:

    height, width,_,nbFrames = datam2.shape
codec_id = "mp4v" # ID for a video codec.
fourcc = cv.VideoWriter_fourcc(*codec_id)
filename = "video2.mp4"
```

```
out = cv.VideoWriter(filename, fourcc=fourcc, fps=20, frameSize=(width, height))


for i in range(nbFrames):
    out.write(datam2[:,:,:,i])


#ANGLE 3:

height, width,_,nbFrames = datam3.shape
codec_id = "mp4v" # ID for a video codec.
fourcc = cv.VideoWriter_fourcc(*codec_id)
filename = "video3.mp4"
out = cv.VideoWriter(filename, fourcc=fourcc, fps=20, frameSize=(width, height))


for i in range(nbFrames):
    out.write(datam3[:,:,:,i])


#Here we declare the matrix that will store our x and y coordinates
x_y_coordinateMatrix1 = np.zeros((0,225))
x_y_coordinateMatrix2 = np.zeros((0,225))
x_y_coordinateMatrix3 = np.zeros((0,225))

#Now we can set up the motion tracking
#We set up a dictionary of legacy motion trackers in opencv

tracker_types = ['BOOSTING', 'MIL','KCF', 'TLD', 'MEDIANFLOW', 'CSRT']
tracker_type = tracker_types[1]

if tracker_type == 'BOOSTING':
    tracker = cv.legacy.TrackerBoosting_create()
if tracker_type == 'MIL':
    tracker = cv.TrackerMIL_create()
if tracker_type == 'KCF':
    tracker = cv.TrackerKCF_create()
if tracker_type == 'TLD':
    tracker = cv.legacy.TrackerTLD_create()
if tracker_type == 'MEDIANFLOW':
    tracker = cv.legacy.TrackerMedianFlow_create()
#if tracker_type == "CSRT":
    #tracker = cv.TrackerCSRT_create()

#VIDEO 1

#The code below allows us to manually draw a rectangle that will track the motion of the bucket

v1 = cv.VideoCapture('video1.mp4')
ret, frame = v1.read()
cv.imshow('frame',frame)
bb = cv.selectROI(frame)
tracker.init(frame,bb)


#Displaying and storing the traking data into arrays

x_coordinates1 = np.zeros((226,1))
y_coordinates1 = np.zeros((226,1))
index = 0
while True:
    ret,frame = v1.read()
    if not ret:
        break
    (success,box) = tracker.update(frame)
    if success:
        (x,y,w,h) = [int(a) for a in box]
        cv.rectangle(frame,(x,y),(x+w,y+h),(250,0,250), 2)
        x_centerCoordinate = (x+w)/2
        y_centerCoordinate = (y+h)/2
        print(x_centerCoordinate,y_centerCoordinate)
        x_coordinates1[index] = x_centerCoordinate
        y_coordinates1[index] = y_centerCoordinate
    cv.imshow('Frame',frame)
    key = cv.waitKey(30)
    index = index+1
    if key == ord('q'):
        break

v1.release()
cv.destroyAllWindows()


#Here we will resize our coordinate arrays so that they are an appropriate shape
#since the camera take with the fewest number of frames generates 225 data points

x_coordinates1 = np.resize(x_coordinates1,(225,1))
y_coordinates1 = np.resize(y_coordinates1,(225,1))


#Now we can reshape the coordinates to make them one long row

x_coordinates1 = np.reshape(x_coordinates1,(1,225))
y_coordinates1 = np.reshape(y_coordinates1,(1,225))


x_y_coordinateMatrix1 = np.append(x_y_coordinateMatrix1,x_coordinates1,axis=0)


x_y_coordinateMatrix1 = np.append(x_y_coordinateMatrix1,y_coordinates1,axis=0)


np.shape(x_y_coordinateMatrix1)


x_y_coordinateMatrix1

#Now do the prevoius steps for the other two videos
#VIDEO 2

#The code below allows us to manually draw a rectangle that will track the motion of the bucket


v2 = cv.VideoCapture('video2.mp4')
ret, frame = v2.read()
cv.imshow('frame',frame)
bb = cv.selectROI(frame)
tracker.init(frame,bb)

#Displaying and storing the tracking data into arrays

x_coordinates2 = np.zeros((284,1))
y_coordinates2 = np.zeros((284,1))
index = 0
while True:
    ret,frame = v2.read()
    if not ret:
        break
    (success,box) = tracker.update(frame)
    if success:
        (x,y,w,h) = [int(a) for a in box]
        cv.rectangle(frame,(x,y),(x+w,y+h),(250,0,250), 2)
        x_centerCoordinate = (x+w)/2
        y_centerCoordinate = (y+h)/2
        print(x_centerCoordinate,y_centerCoordinate)
        x_coordinates2[index] = x_centerCoordinate
        y_coordinates2[index] = y_centerCoordinate
    cv.imshow('Frame',frame)
    key = cv.waitKey(30)
    index = index+1
    if key == ord('q'):
        break

v2.release()
cv.destroyAllWindows()

#Here we will resize our coordinate arrays so that they are an appropriate shape
#Since the camera take with the fewest number of frames generates 225 data points,

x_coordinates2 = np.resize(x_coordinates2,(225,1))
y_coordinates2 = np.resize(y_coordinates2,(225,1))

#Now we can reshape the coordinates to make them one long row

x_coordinates2 = np.reshape(x_coordinates2,(1,225))
y_coordinates2 = np.reshape(y_coordinates2,(1,225))

x_y_coordinateMatrix2 = np.append(x_y_coordinateMatrix2,x_coordinates2,axis=0)

x_y_coordinateMatrix2 = np.append(x_y_coordinateMatrix2,y_coordinates2,axis=0)

np.shape(x_y_coordinateMatrix2)

x_y_coordinateMatrix2

#VIDEO 3

#The code below allows us to manually draw a rectangle that will track the motion of the bucket


v3 = cv.VideoCapture('video3.mp4')
ret, frame = v3.read()
cv.imshow('frame',frame)
bb = cv.selectROI(frame)
tracker.init(frame,bb)

#Displaying and storing the tracking data into arrays

x_coordinates3 = np.zeros((232,1))
y_coordinates3 = np.zeros((232,1))
index = 0
while True:
    ret,frame = v3.read()
    if not ret:
        break
    (success,box) = tracker.update(frame)
    if success:
        (x,y,w,h) = [int(a) for a in box]
        cv.rectangle(frame,(x,y),(x+w,y+h),(250,0,250), 2)
        x_centerCoordinate = (x+w)/2
        y_centerCoordinate = (y+h)/2
        print(x_centerCoordinate,y_centerCoordinate)
        x_coordinates3[index] = x_centerCoordinate
        y_coordinates3[index] = y_centerCoordinate
    cv.imshow('Frame',frame)
    key = cv.waitKey(30)
    index = index+1
    if key == ord('q'):
        break

v3.release()
cv.destroyAllWindows()

#Here we will resize our coordinate arrays so that they are an appropriate shape
```

```
#Since the camera take with the fewest number of frames generates 225 data points

x_coordinates3 = np.resize(x_coordinates3,(225,1))
y_coordinates3 = np.resize(y_coordinates3,(225,1))

#Now we can reshape the coordinates to make them one long row

x_coordinates3 = np.reshape(x_coordinates3,(1,225))
y_coordinates3 = np.reshape(y_coordinates3,(1,225))

x_y_coordinateMatrix3 = np.append(x_y_coordinateMatrix3,x_coordinates3,axis=0)

x_y_coordinateMatrix3 = np.append(x_y_coordinateMatrix3,y_coordinates3,axis=0)

np.shape(x_y_coordinateMatrix3)

x_y_coordinateMatrix3

#Now stack all the videos together
x_y_coordinateMatrix = np.zeros((0,225))
x_y_coordinateMatrix = np.append(x_y_coordinateMatrix,x_y_coordinateMatrix1,axis=0)
x_y_coordinateMatrix = np.append(x_y_coordinateMatrix,x_y_coordinateMatrix2,axis=0)
x_y_coordinateMatrix = np.append(x_y_coordinateMatrix,x_y_coordinateMatrix3,axis=0)
x_y_coordinateMatrix
np.shape(x_y_coordinateMatrix)
np.save('x_y_coordinateMatrix',x_y_coordinateMatrix)

#PCA
x_y_coordinateMat = np.load('x_y_coordinateMatrix.npy')
x_y_coordinateMat.shape
plt.plot(x_y_coordinateMat[1,:])

#We must compute the mean
#we want the mean row to form the mean matrix so that we can subtract it
#from our data matrix to ensure that we work with a zero mean, gaussian matrix

row1_x = x_y_coordinateMat[0,:]
row1_y = x_y_coordinateMat[1,:]
row2_x = x_y_coordinateMat[2,:]
row2_y = x_y_coordinateMat[3,:]
row3_x = x_y_coordinateMat[4,:]
row3_y = x_y_coordinateMat[5,:]

mean1_x = np.mean(row1_x)
mean1_y = np.mean(row1_y)
mean2_x = np.mean(row2_x)
mean2_y = np.mean(row2_y)
mean3_x = np.mean(row3_x)
mean3_y = np.mean(row3_y)

x_y_coordinateMat[0,:] = x_y_coordinateMat[0,:] - mean1_x
x_y_coordinateMat[1,:] = x_y_coordinateMat[1,:] - mean1_y
x_y_coordinateMat[2,:] = x_y_coordinateMat[2,:] - mean2_x
x_y_coordinateMat[3,:] = x_y_coordinateMat[3,:] - mean2_y
x_y_coordinateMat[4,:] = x_y_coordinateMat[4,:] - mean3_x
x_y_coordinateMat[5,:] = x_y_coordinateMat[5,:] - mean3_y

NewcoordMat = x_y_coordinateMat

NewcoordMat

plt.plot(NewcoordMat[1,:])


#Now we proceed to construct the covariance matrix, which allows us to figure out
#which varable are mosst related to each other

covariance = np.cov(NewcoordMat)
```

```python
covariance

#We need to diagonalize our covariance matrix to remove redundancies.
#A diagonalized covarance matrix implies that the off diagonals are zeros(it means our variables are
↪  independent of each other)

#To this w e will use the SVD method

u,s,vt = np.linalg.svd(NewcoordMat)

vt

#These are our Principle Components (PC's)
ut = u.transpose()

#We can switch to the basis of thw PC's by doing the following:
Y = np.matmul(ut,NewcoordMat)
covarianceY = np.cov(Y)
covarianceY
s

#The blue curve is position and the orange curve is its derivative
plt.plot(s[0]*vt[0,:])
plt.plot(s[1]*vt[1,:])

#SINDy METHOD

!pip install pysindy

import pysindy as ps

NewcoordMat

coordRedux = (u[:,0:2].transpose())@NewcoordMat

coordRedux.shape

coordReduxT = coordRedux.transpose()

x = coordReduxT[:,0]
y = coordReduxT[:,1]

featureNames = ['x','y']

opt = ps.STLSQ(threshold = 0.1)

model = ps.SINDy(feature_names =featureNames, optimizer = opt)

model.fit(coordReduxT, t = 11/226)

model.print()

#We must integrate to find  (x) and (y)
from scipy.integrate import odeint
def function(z,t):
    a=3.176
    b=-3.302
    c=-7.267
    d=2.998
    dxdt = a + b*z[1]
    dydt = c + d*z[0]

    dzdt = [dxdt,dydt]
    return dzdt

#Define your initial condition (its the first row of your projected data)
z0 = [coordReduxT[0,0],coordReduxT[0,1]]
```

```
#Time points
t = np.arange(0,11,11/226)

#This will solve the ODE (it will integrate it)
z = odeint(function, z0, t)

#Now z will be a matrix with 2 columns
plt.plot(t,z[:,0]) #this is x(t)
plt.plot(t,z[:,1]) #this is y(t)
```