MATH 494A Assignment 2

Yifan Wu

March 5, 2023

Abstract

This project involves using principal component analysis (PCA) and sparse identification of nonlinear dynamics (SINDy) methods to extract dynamics from video data of a spring-mass system with 3 different camera angles under ideal and noisy conditions. The goal is to extract the time evolution from the first two dominant principal components and apply the SINDy method to learn the governing equations of the motion.

1 Introduction and Overview

In this Assignment, we aim to understand the motion of a spring-mass system without physics training. We set up three cameras at different angles to record videos of the spring mass system, then we will collect the mass position matrix of each video which is 2 dimensions of data and combine it into six dimensions of data. We will use PCA to extract the first two principal components from mass positions matrix and then apply the SINDy method to learn the equations of motion. The goal is to extract the time evolution from the first two dominant principal components and observe the effects of the sparsity parameter on the learned equations. We will be considering two different tests in this assignment: an ideal case and a noisy case. In the ideal case, we will consider a small displacement of the mass in the z direction and the ensuing oscillations. The motion will be entirely in the z direction and simple harmonic motion will be observed. In the noisy case, we will repeat the ideal case experiment, but this time introduce camera shakes into the video recording, making it more challenging to extract the simple harmonic motion.

Overall, this assignment provides an opportunity to apply various analytical techniques, such as PCA and SINDy, in a practical setting and gain a deeper understanding of the dynamics of a spring-mass system.

2 Theoretical Background

2.1: Principal component Analysis

Principal Component Analysis (PCA) is a statistical technique used to reduce the dimensionality of a dataset while retaining as much information as possible. PCA is based on the idea that it is possible to represent a high-dimensional dataset with a smaller set of variables, called principal components. PCA is performed by finding the eigenvectors and eigenvalues of the covariance matrix of the data [1] we first find

the covariance Matrix by the following equation.

$$C_X = \frac{1}{n-1} XX^T = \begin{bmatrix} \sigma_a^2 & \cdots & \sigma_{ad}^2 \\ \vdots & \ddots & \vdots \\ \sigma_{da}^2 & \cdots & \sigma_d^2 \end{bmatrix} \quad \textbf{[2].}$$

Then it would also be nice to know which variables have the largest variance because these contain the most important information about our data. Therefore, we want to diagonalize this matrix so that all off-diagonal elements (covariances) are zero: [1]

$$C_X = V \Lambda V^{-1} \quad \textbf{[3]}$$

Then the principal components are obtained by projecting the data onto the eigenvectors. As the equation above, V is a matrix containing the eigenvectors of Cov(x) in its columns, So let

$$Y = X.V \quad \textbf{[4]}$$

The matrix Y contains the principal components of X, sorted by the amount of variance they explain.

2.2: Sparse identification of nonlinear dynamics (SINDy)

SINDy is a data-driven method used to identify the governing equations of a dynamical system from time-series data. SINDy assumes that the dynamics of the system can be represented by a sparse combination of functions of the state variables. The goal of SINDy is to find the sparsest set of functions that accurately capture the dynamics of the system. The mathematical formulation of SINDy is as follows [1]

$$\frac{dx}{dt} = f(x) \quad \textbf{[5]}$$

Where x is the state vector of the system, and f is an unknow function that governs the dynamics of the system.

$$f(x) = \theta(x) X_i \quad \textbf{[6]}$$

Where $\theta(x)$ is a matrix of basis functions evaluated at the state x, and X_i is a vector of coefficients. SINDy uses sparsity-promoting techniques, such as L1 regularization, to find the sparsest set of coefficients X_i that accurately capture the dynamics of the system. The basis functions used in SINDy can be any set of functions that are nonlinear in the state variables. Examples include polynomials, trigonometric functions exponentials etc. The choice of basis functions depends on the specific system being analyzed.

3 Algorithm Implementation and Development

3.1 Data Preprocessing

We will begin by converting the three different camera angle matrices back into videos and using the KCF tracker from OpenCV to capture the mass position from each video. The resulting mass position matrix from each camera should have a size of (frame, 2). We will combine these three videos into a new matrix of size (frame, 6). Since the camera 1 video has the smallest frame size, we will truncate the frames of the other two videos to match the frame size of camera 1. This will result in our final mass position matrix of size (frame, 6).

3.2 PCA

Next, we will apply the PCA algorithm to the mass position matrix. We will first subtract the mean of the mass position matrix to obtain a new matrix B. We will then apply the SVD algorithm on B and extract the first two singular values which will be our first two principal components of the PCA. We will truncate our singular value matrix s(6,6) to s(2,2) to retain only the two largest singular values. We can apply PCA by computing T = U dot s, where U is the matrix of left singular vectors from the SVD algorithm. The resulting matrix T will have a size of (frame,2) and will contain the first two principal components of our data.

3.3 SINDy

We will then apply the Sparse Identification of Nonlinear Dynamics (SINDy) algorithm on the PCA output using PySINDy. We will choose our library as polynomial degree 2 and our feature as "x" and "y". We will adjust our sparse parameter to achieve the best approximation of our data. Finally, we will simulate some data using the initial conditions and plot it out to compare with the original data and evaluate the performance of the SINDy model.

4 Computational Results

4.1 Ideal Case

After we combine 3 matrixes into one matrix where the size will be (frame, 6) we call this matrix “C”, then we will subtract the mean of C, the resulting matrix will call it “B”. The reason we need center the data is PCA seeks to find the directions of maximum variance in the data. If the data is not centered, then the direction of maximum variance might be affected by the mean of the data. By subtracting the mean from the data, we ensure that the center of the data is at the origin and the direction of maximum variance is not affected by the mean. Figure 1 shows the mean is successfully subtracted.

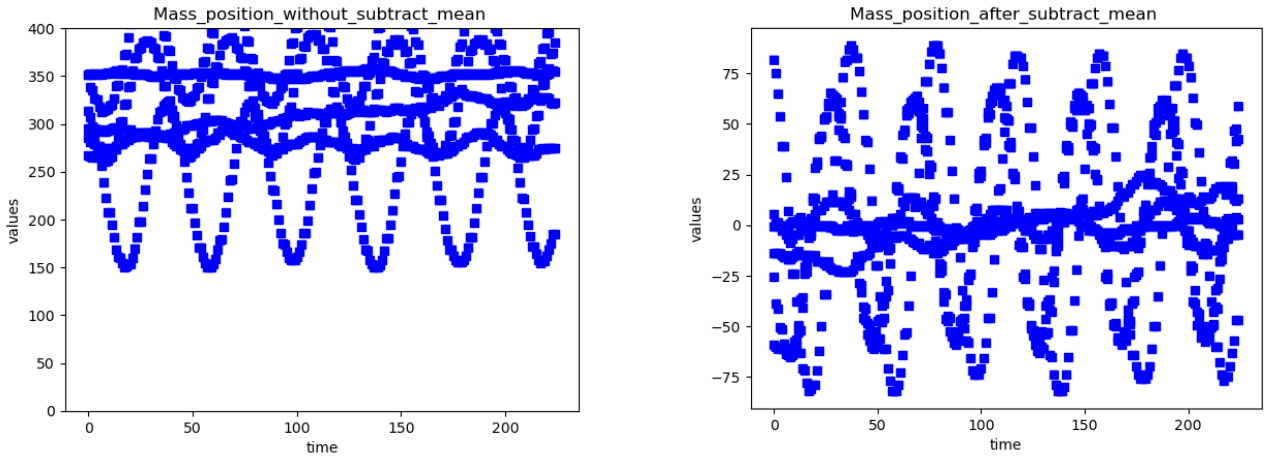


Figure 1: The plot shows the mass position matrix without subtracting mean and after subtracting mean.

Then after we apply SVD algorithms on matrix B, figure 2 shows that we have the first two principal components.

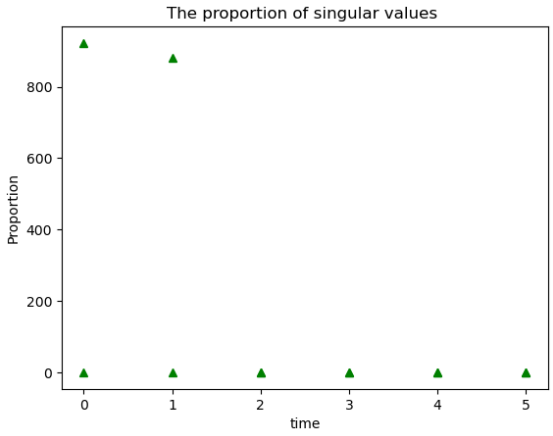
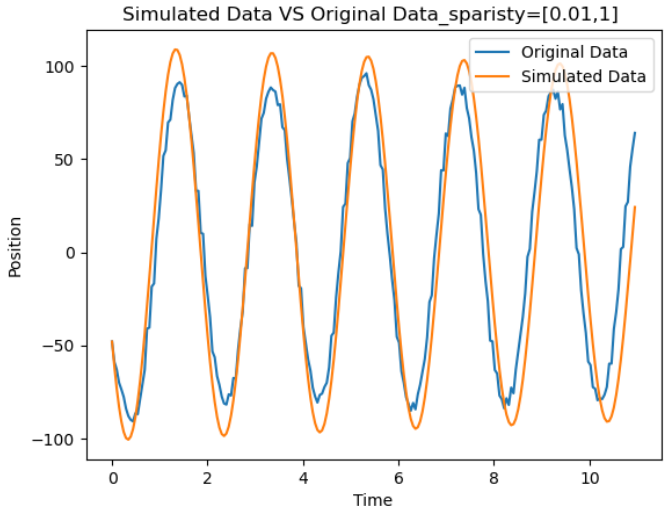


Figure 2: The plot shows the singular values in matrix B.

Then by the algorithm [3.2] we compute our PCA matrix T. we now can apply PySINDy on it. Figure 3 is the result after we apply the algorithm [3.3]. By adjusting the sparsity parameter "P", I found an interval where P = [0.01,1] provides the best approximation. Figure 3 shows the results of comparison between simulated data and original data. In addition, I will include results in appendix A section where a non-optimal sparsity parameter was selected to demonstrate the impact on the accuracy of the model.



$$\frac{dx}{dt} = 9.7371 + 0.038\,x + 3.285\,y$$

$$\frac{dy}{dt} = 13.8531 - 2.977\,x - 0.057\,y$$

Figure 3: The plot shows the result of SINDy equations and comparison with the original data.

4.3 Noisy Case

By repeating the method of [4.1] on noisy dataset

Figure 4 shows that we still can extract the first 2 principal components

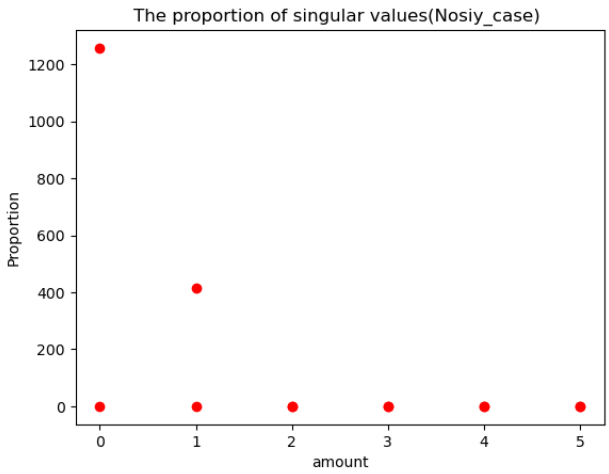


Figure 4: The plot shows the singular values of noisy dataset.

However, after we apply PCA on our dataset, figure 5 shows the dynamic of our spring-mass system is highly affected by camera shake.

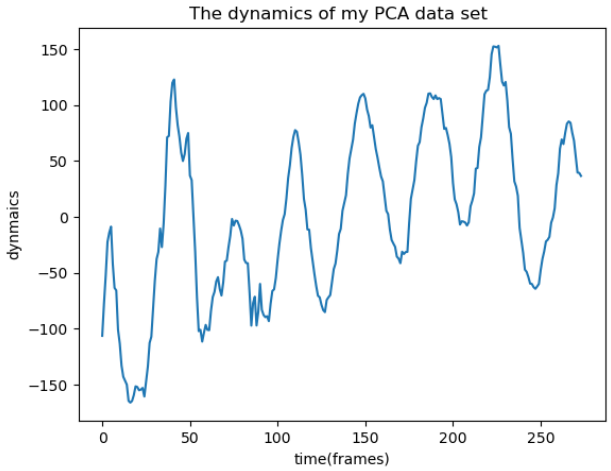


Figure 5: The plot shows the dynamics of PCA data set on noisy case.

Despite multiple attempts to adjust the sparse parameter, the SINDy model was unable to accurately capture the dynamics of the noisy case of the spring-mass system. Figure 6 is the resulting plot.

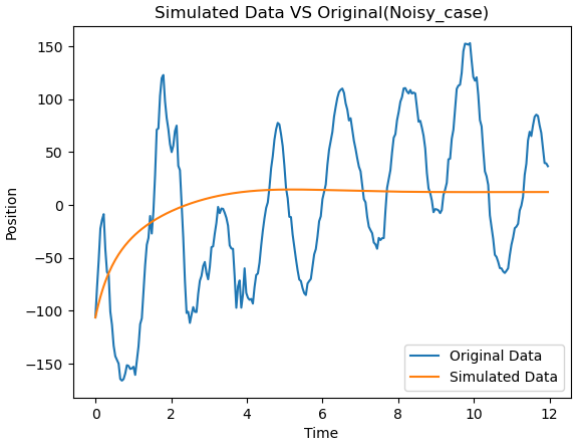


Figure 6: The plot shows the SINDy model comparison with the original data in noisy case.

5 Summary and Conclusions

In summary, the results of the ideal case showed that SINDy successfully created a model that accurately represented the spring-mass system dynamics. However, in the noisy case, SINDy failed to do so. One possible reason for this could be that when using the KCF tracker to capture the mass position from the shaky camera, the tracker itself may also be shaking, leading to a less clean dataset. As PCA requires a clean dataset to work with, this could be the reason why SINDy was not successful in creating a model in the noisy case. Additionally, the use of the first two principal components is critical because the second principal component represents the variation in the direction of motion, which is essential for understanding the dynamics of the system. Therefore, the second principal component can represent the acceleration of the mass in a given direction, which is essential for understanding the forces acting on the mass and the system's dynamics.

References

[1] Jose Nathan Kutz. *Data-Driven Modeling scientific computation: methods for complex systems and big data. Oxford University Press, 2013*

Appendix A Additional Figure

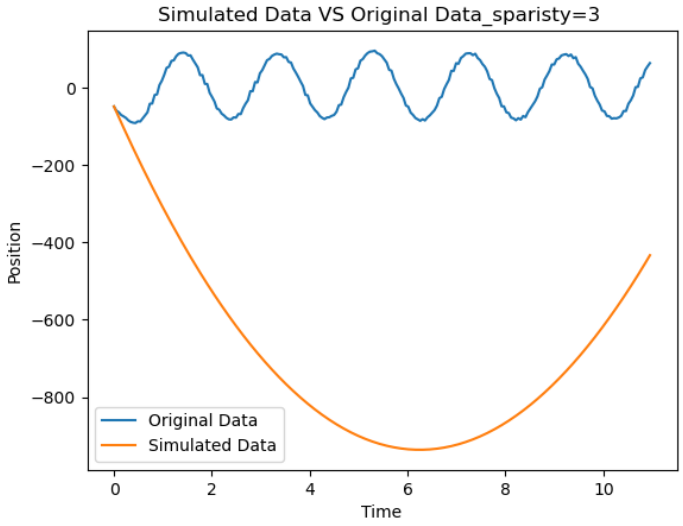


Figure 7: The plot shows the result of an ideal case when sparsity = 3 the simulated data vs original data

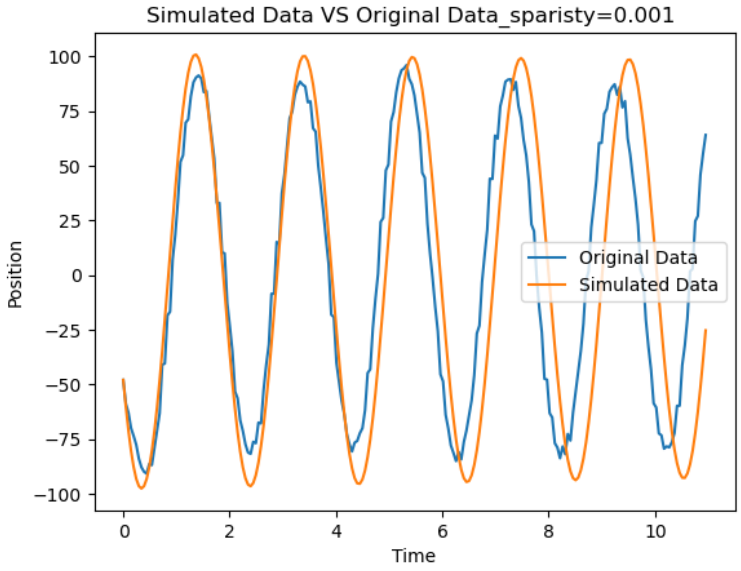


Figure 8: The plot shows the result of an ideal case when sparsity = 0.001 the simulated data vs original data

Appendix B Python Functions

1: np.load("camera3Noisy.npy"). load matrix cap = cv2.VideoCapture('camera3Noisy.mp4'). Capture video

2: tracker = cv2.TrackerKCF_create(). "Create tracker for capture the mass position in the video."

3: np.hstack((c1, c2, c3)). Combine each video into (frame,6)

4: np.mean(C, axis=0)

5: U, s, Vt = np.linalg.svd(B,full_matrices=False)

6: sindy = SINDy(
differentiation_method=FiniteDifference(),
feature_library=PolynomialLibrary(degree=2),
optimizer=STLSQ(threshold=0.01), # Set the threshold to adjust sparsity
feature_names=["x", "y"] "Create SINDy model"

7: sim_data = sindy.simulate(x0, t_sim) "simulate the data by using initial conditions

Appendix C Python Code

This code represents what I did for the ideal case, by noisy case just simply change the npy file to nosiy one

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.image import imread
import cv2
from numpy import array
from pysindy import SINDy
from pysindy.differentiation import FiniteDifference
from pysindy.feature_library import PolynomialLibrary
from pysindy.optimizers import STLSQ
from scipy.integrate import odeint

## Import the np array
my_array = np.load("camera1NoNoisy.npy")
my_array.shape

height, width,channels,num_frames = my_array.shape

# Create VideoWriter object
fourcc = cv2.VideoWriter_fourcc(*'mp4v')
out = cv2.VideoWriter('camera3Noisy.mp4', fourcc, 25, (width, height),
isColor=(channels==3))
# Iterate through frames and write to video
for i in range(num_frames):
    frame = my_array[:, :, :, i]
    out.write(frame)
# Release resources
out.release()
```

```
# Open the video file
cap = cv2.VideoCapture('camera3Noisy.mp4')

# set the frame position to 20
# cap.set(cv2.CAP_PROP_POS_FRAMES, 6)

# Read the first frame
ret, frame = cap.read()

# Select the object you want to track by drawing a bounding box around it
bbox = cv2.selectROI(frame, False)

# Initialize the KCF tracker with the bounding box coordinates
tracker = cv2.TrackerKCF_create()
tracker.init(frame, bbox)

# Initialize an array to store the mass positions
mass_positions = []

# Loop through the video frames
while True:
    # Read a frame from the video
    ret, frame = cap.read()
    if not ret:
        break

    # Update the tracker with the new frame
    success, bbox = tracker.update(frame)

    # If the tracking was successful, save the mass position
    if success:
        x, y, w, h = [int(i) for i in bbox]
        mass_positions.append([x + w // 2, y + h // 2])

# Convert the list of mass positions to a numpy array
```

```
mass_positions_array = np.array(mass_positions)
# save the array as an .npy file

np.save('camera1_No_Noisy', mass_positions_array)

## Load the Mass_Position of each of my carmera with no_noise
#c1
c1 = np.load('No_Noisy_Camera1.npy')

#c2
c2 = np.load('No_Noisy_Camera2.npy')

#c3
c3 = np.load('No_Noisy_Camera3.npy')

## reshape all in 225
c2=c2[0:225,:]
c3=c3[0:225,:]
c3.shape

# Concatenate matrices horizontally
C = np.hstack((c1, c2, c3))
C.shape

plt.title("Mass_position_without_subtract_mean ")
plt.ylim([0, 400])
plt.xlabel('time')
plt.ylabel('values')
plt.plot(C, 'bs')
plt.show()

# Subtract the mean of mass_positions_array
B = C - np.mean(C, axis=0)

# Perform singular value decomposition on B
```

```
U, s, Vt = np.linalg.svd(B,full_matrices=False)

s=np.diag(s)
plt.title("Mass_position_after_subtract_mean ")
plt.xlabel('time')
plt.ylabel('values')
plt.plot(B, 'bs')
plt.show()

plt.title("The proportion of singular values ")
plt.xlabel('time')
plt.ylabel('Proportion')
plt.plot(s[:,0:2], 'g^')
plt.show()

s=s[:,0:2]
T = U.dot(s)
plt.plot(T,'ro')
T.shape

# Create a SINDy object with the desired settings
sindy = SINDy(
    differentiation_method=FiniteDifference(),
    feature_library=PolynomialLibrary(degree=2),
    optimizer=STLSQ(threshold=0.01), # Set the threshold to adjust sparsity
    feature_names=["x", "y"]
)

##time
t_sim = np.arange(0, 11, 11/225)

# Fit the SINDy model to the principal components
sindy.fit(T,t_sim)

# Get the learned model coefficients
```

```
coef = sindy.coefficients()

# Print the learned model
print(coef)

# Get the learned model equations
eqs = sindy.equations()
print(eqs)

x0=T[0]
x0.shape

# Simulate the system forward in time using the SINDy model
sim_data = sindy.simulate(x0, t_sim)

# plt.plot(sim_data,'ro')

plt.plot(t_sim, T[:,0:1], label='Original Data')
plt.plot(t_sim, sim_data[:,0:1], label='Simulated Data')
plt.title("Simulated Data VS Original Data_sparisty=[0.01,1] ")
plt.xlabel('Time')
plt.ylabel('Position')
plt.legend()
plt.show()
```