Learning Dynamics from Video - The SINDy Method

Timur Zhanabaev

MAST 680 - Assignment 2 - March 7, 2023

Abstract

In this assignment, the SINDy (Sparse Identification of Nonlinear Dynamics) method is implemented to learn the physics of an oscillating object. First, the video is filtered to extract the bright pixels coordinates. Then the position of the bright objects in the video are tracked with a density based clustering method (DBSCAN). The cluster pertaining to the bucket is selected to create a time series for the SINDy method.

1 Introduction and Overview

As the main subject of the work, the SINDy method allows for model discovery as a sparse regression problem to time series data, [1]. The assignment then involves filtering the video to keep only bright pixels. This idea was gained from watching the video, where the bucket was identified to be bright enough to stand out from the background. Having determined the pixel coordinates of all bright pixels, a clustering method is used to aggregate pixels into clusters. The DBSCAN method is used, as it is flexible with non-uniform cluster shapes and an undetermined number of clusters. In short, filter and DBSCAN is performed to isolate the time series data of 3 camera angles. To reduce the dimension of the data, the PCA method is used to keep only 2 components, which are used to describe our dynamical system of 2 first order ODEs. The SINDy method is implemented with it's various modes; the discrete-time, the continuous-time and weak-formulation. Then. specifying a library of functions and a threshold parameter for SINDy, the method is used to learn the dynamics of the system. Finally, the EDMD method is used to test the validity of results.

2 Theoretical Background

2.1 Sparse Identification of Nonlinear Dynamics Method

Given a data set $X' \in \mathbb{R}^{M\times N}$. In the video processing scenario as in the Assignment 1, M is number of pixels and N is the number of snapshots of the video. Next, the snapshot matrix $Y \in \mathbb{R}^{M\times(N-1)}$ containing the last $N-1$ frames (each snapshot being $\triangle t > 0$ time away) is defined as,

$$Y = \begin{bmatrix} X'_2 & X'_3 & \dots & X'_N \end{bmatrix} \tag{1}$$

As well, $X \in \mathbb{R}^{M\times N}$ the snapshot matrix containing the first $N-1$ snapshots, defined as,

$$X = \begin{bmatrix} X'_1 & X'_2 & \dots & X'_{N-1} \end{bmatrix} \tag{2}$$

Compared to the DMD method, the SINDy method the uses a dictionary of user-specified functions to map the X matrix to a lower dimensional embedding with a library of functions. Such dictionary of K functions is denoted as,

$$\mathcal{D} = \{\psi_1, \psi_2, ..., \psi_K\} \tag{3}$$

Moreover, the input in each function is the snapshot of M pixels with an output of a single value. The snapshot matrix is then reduced from $M \times N$ to $K \times N$, as each snapshot is the input for each function in the dictionary. The output from the dictionary mappings then compose the observables

matrix $\Psi(X) \in \mathbb{R}^{(K\times N)}$ defined as,

$$\Psi(X) = \begin{bmatrix} \psi_1(X_1) & \psi_1(X_2) & \dots & \psi_1(X_N) \\ \vdots & \vdots & \ddots & \vdots \\ \psi_K(X_1) & \psi_K(X_2) & \dots & \psi_K(X_N) \end{bmatrix} \tag{4}$$

Following, the SINDy method seeks to solve the minimization problem,

$$\text{argmin}_A ||Y - A\Psi(X)||_F^2 + \nu||A||_0 \tag{5}$$

Where the L_0 counts the number of non-zero coefficients in the A matrix. Similar to the LASSO regression, which achieves the coefficient sparsity with the L_1 norm, the L_0 measure penalizes the model for having non-zero coefficients, with the penalty term ν increasing the weight on L_0 penalty. Although, the solution may not be computationally tractable.

2.2 Continuous-time model

For a continuous-time dynamical system, instead of the snapshots at later time (the Y matrix), we define a new $\tilde{Y}$ matrix that holds difference equations $\delta x(t_n) = (x(t_n) - x(t_{n-1}))/\triangle t$ as its elements, to model our system as,

$$\dot{x}_j = a_1\psi_1(x_j) + ... + a_K\psi_K(x_j) \tag{6}$$

With the matrix $\tilde{Y}$ replacing the Y matrix, defined as,

$$\tilde{Y} = \begin{bmatrix} \delta x_1(t_1) & ... & \delta x_1(t_N) \\ \delta x_2(t_1) & ... & \delta x_2(t_N) \\ \vdots & \ddots & \vdots \\ \delta x_M(t_1) & ... & \delta x_M(t_N) \end{bmatrix} \tag{7}$$

Furthermore, a simple finite difference equation can be very sensitive to noise, so instead we use the **weak formulation** of a differential equation to estimate it's numeric integral, being robust to noisy measurements. [1] Assuming that our library estimates the ODE, we have the equations in the weak form, where $t_1 = 0$, $\triangle t = 1/30$ sec,

$$x(t_n) - x(t_1) = \sum_{k=1}^{K} a_k \int_0^t \psi(x(s))ds \tag{8}$$

Then the integral is approximated by a numerical method over intervals $[t_n, t_{n+1}]$, defined as,

$$\mathcal{I}_k(t_n) = \triangle t \sum_{j=1}^{n} \psi(x(t_j)) \tag{9}$$

Hence the new minimization problem with SINDy methodology is now,

$$\text{argmin}_A ||\tilde{X} - A\mathcal{I}||_F^2 + \nu||A||_0 \tag{10}$$

where,

$$\tilde{X} = \begin{bmatrix} 0 & x_1(t_2) - x_1(t_1) & ... & x_1(t_N) - x_1(t_1) \\ 0 & x_2(t_2) - x_2(t_1) & ... & x_2(t_N) - x_2(t_1) \\ \vdots & \vdots & \ddots & \vdots \\ 0 & x_M(t_2) - x_M(t_1) & ... & x_M(t_N) - x_M(t_1) \end{bmatrix} \in \mathbb{R}^{M\times N} \tag{11}$$

and

$$\mathcal{I} = \begin{bmatrix} \mathcal{I}_1(t_1) & \mathcal{I}_1(t_2) & ... & \mathcal{I}_1(t_N) \\ \mathcal{I}_2(t_1) & \mathcal{I}_2(t_2) & ... & \mathcal{I}_2(t_N) \\ \vdots & \vdots & \ddots & \vdots \\ \mathcal{I}_K(t_1) & \mathcal{I}_K(t_2) & ... & \mathcal{I}_K(t_N) \end{bmatrix} \in \mathbb{R}^{K\times N} \tag{12}$$

2.3 Iterative process

Since there is no closed form solution to eq.(10), another method is employed to achieve sparsity. A **sparsity parameter** $\lambda > 0$ is defined to set all coefficients in A below λ to zero, defining the new matrix $\tilde{A}$. The procedure is as follows,

1. compute $A^{(t)} = [a_{m,k}] = \tilde{X}\mathcal{I}^{\dagger}$
2. set matrix $\tilde{A}$, where $\tilde{a}_{m,k} = 0$ if $|a_{m,k}| < \lambda$
3. re-compute $\mathcal{I}^{\dagger}$ with reduced library
4. re-compute $A^{(t+1)} = \tilde{X}\mathcal{I}^{\dagger}$, where $^{\dagger} \in \mathbb{R}^{M-K\times N}$

3 Algorithm Implementation and Development

3.1 Development

First off, the data in form a video file is to be processed. The moving bucket in the footage is then to be tracked and isolated to create a position vector over time. As each video is filmed from different angles, we then have a dynamic system over time in 3 spatial dimensions, although not an orthogonal basis such as (x, y, z) coordinates. Although the motion of interest is only the up and down oscillation in z-axis. So, all other motion is considered as xy-axis.

For the object tracking, it was identified initially that the bucket is a bright object in the video, so in order to separate the bucket first a threshold is created to remove pixels below certain values. In other words for RGB values $\in [1, 255]$, coordinates of pixels with values > 225 are kept for `cam1_1` and `cam1_2`, > 238 for `cam2_1` and `cam2_2` and > 200 for `cam3_1` and `cam3_2`. Such thresholds are chosen from trial and error while testing with DBSCAN to see if it can pick up the bucket pixels consistently. For reference, the figures showcase the filtering effect, (1).

Figure 1: (left) original `cam1_2` video and (right) video with threshold of 238, filtering out all the pixels not being as bright as the bucket.

Initially, DMD was attempted to use to separate the oscillating bucket. While it worked well for videos without camera jittering noise, but for the noisy ones it failed to pick up the bucket for DBSCAN tracking.

3.2 DBSCAN

DBSCAN is a density based clustering method, where the clusters are formed from core samples, a collection of points close to each other by user-specified distance metrics. The method allows then for flexible shapes of clusters and un-specified number of clusters unlike methods similar to K-means clustering. [2]. For the assignement, a premade solution is used from the `scikit-learn` python package. The input is the coordinates of pixels of interest, with parameters for the model being; `eps` determining the distance for points to be between to be considered within the cluster and `min_samples` determining the minimum of detected core points to create a cluster. The parameters are then tuned such that the bucket of roughly (50×50) pixels is selected as a single cluster.

3.3 PCA

The second pre-made method is PCA, using the same machine-learning package `scikit-learn`. Being a very easy to use package, PCA can be done in a single line of code with the parameter *n_components* determining the number of dimensions to keep.

example: `Xpca = PCA(n_components=2).fit_transform(X)`

3.4 Harmonic Oscillator

For the continous-time SINDy framework, we are expected to estimate first order ODEs. With a prior knowledge of the dynamics, we can find the form of the system we are estimating. The framework for the harmonic oscillator is presented here to pinpoint the choice of library.

The differential equation for a **undriven harmonic damped oscillator** is,

$$\ddot{x}(t) + 2\mathcal{D}\omega\dot{x}(t) + \omega^2 x(t) = 0 \tag{13}$$

where ω is the undamped angular frequency and $\mathcal{D} = c/(2\sqrt{mk})$ is the damping ratio. Changing to a system of 1st order ODEs for the SINDy framework, with $z_1(t) = x(t)$ and $z_2(t) = \dot{x}(t)$. Then we have 2 equations in the system,

$$\begin{cases} \dot{z}_1(t) = \dot{x}(t) = z_2(t) \\ \dot{z}_2(t) = -2\mathcal{D}\omega z_2(t) - \omega^2 z_1(t) \end{cases} \tag{14}$$

where ω is the undamped angular frequency and $\mathcal{D} = c/(2\sqrt{mk})$ is the damping ratio. Hence, we can assume that a library $\mathcal{L}' = \{z_1, z_2\}$ to be sufficient. Although for completeness and the fact that we don't have such information for unknown dynamics, the libary will be chosen to be then $\mathcal{L} = \{1, z_1, z_2, z_1^2, z_2^2, z_1 \cdot z_2\}$

Moreover, we recover 2 components with PCA from 3 camera angles, and so each component represents the 2 variables $z_1(t)$ and $z_2(t)$. Thinking in terms of dynamics, $z_1(t) = x(t)$ is the position of the object and $z_2(t) = \dot{x}(t)$ is the change in position, i.e the velocity of the object at time t.

3.5 SINDy Implementation

Using the regular `numpy` for matrix multiplications, the **discrete**, **continuous-time** with difference equations and the **continuous-time weak-formulation** methods are made for SINDy method. Although the methods do not seem to work in my implementation, hence the reasoning to try out all the methods to understand better whats going wrong.

4 Computational Results

4.1 Tracking the bucket

For the initial part, once the video frames are filtered by the cut-off and the coordinates are saved, then DBSCAN tracking is done. For the clustering method, the best parameters for `eps` = 50 and `min_samples` = 100. Such parameters consistently selected the bucket cluster, producing the motion shown in figures (2). However, since the position of the bucket is actually unknown, i.e. the actual distances traveled in meters in the class room, the unit of measure here is the average pixel coordinate (the centroid of the cluster). Due to such "less than perfect" method of tracking, the motion is choppy with extra noise added due to the cluster shape changing overtime. This can be seen more often in the `cam2_1` bucket motion (2).

Next for PCA, only the z-axis vectors are used and the 2 principle components from the videos are recovered, shown in figures (3). These 2 vectors are the needed to describe the dynamics of our system of first order ODEs shown in eq. (14). The ideal and noisy case PCA reconstructions with the pendulum/noise motions in the xy-axis are shown in the appendix figures (5). However, with or without the xy-axis motion, the reconstructions are both similar.

4.2 Discrete Time

For the ideal case, the discrete-time SINDY method is applied with the library of functions $\mathcal{L} = \{1, z_1, z_2, z_1^2, z_2^2, z_1 \cdot z_2\}$. The predicted dynamics from initial conditions are then generated, as shown in figures (4). In terms of model discovery, the model's form with $\lambda = 0.01$ was found to be,

$$\begin{cases} z_1(n+1) = -0.8216 + 0.9793 z_1(n) + 0.1613 z_2(n) \\ z_2(n+1) = 0.0974 - 0.1377 z_1(n) + 0.9811 z_2(n) \end{cases} \tag{15}$$

Increasing the sparsity to $\lambda = 0.1$, the model is then,

$$\begin{cases} z_1(n+1) = -0.8216 + 0.9793 z_1(n) + 0.1613 z_2(n) \\ z_2(n+1) = -0.1378 z_1(n) + 0.9811 z_2(n) \end{cases} \tag{16}$$

The sparser model removes the intercept coefficient for z_2 and slightly changes some values at 4th and 5th decimal places.

For the noisy case, the discrete-time SINDY method is applied with the same library of functions as before with a threshold $\lambda = 0.01$. The predicted dynamics in figures (4) as well. The discovered model was found to be,

$$\begin{cases} z_1(n+1) = 0.7324 + 0.9618 z_1(n) - 0.2319 z_2(n \\ z_2(n+1) = 0.3412 + 0.0165 z_1(n) + 0.9440 z_2(n) \end{cases} \tag{17}$$

Increasing the sparsity to $\lambda = 0.1$, the noisy case model is then,

$$\begin{cases} z_1(n+1) = 0.7324 + 0.9618 z_1(n) - 0.2319 z_2(n \\ z_2(n+1) = 0.3367 + 0.9448 z_2(n) \end{cases} \tag{18}$$

The sparser model removes the z_1 dependency for z_2, with coefficient changes 4th decimal places, where sparsification occured.

Looking at figures (4), the predicted dynamics do follow oscillatory motion of a spring system, albeit with a very high damping coefficient. The noisy case however fails at prediction. The same is done with the EDMD method for the ideal case. The results make more sense, where damping is not as visible within the video's time frame, as shown in figure (7) in the Appendix.

4.3 Continuous Time - Weak Form

Next, the SINDy method is implemented using the weak formulation for differential equations to smooth out noisy data. Skipping over details, since the method did not work either due to a faulty implementation. With $\lambda = 0.1$, the model dynamics for ideal case are,

$$\begin{cases} \dot{z}_1(t) = -28.7683 - 0.5087 z_1(t) + 5.0811 z_2(t) \\ \dot{z}_2(t) = 3.6432 - 3.2362 z_1(t) - 0.3326 z_2(t) \end{cases} \tag{19}$$

As well the model dynamics for noisy case with $\lambda = 0.1$, are,

$$\begin{cases} \dot{z}_1(t) = 14.2834 - 0.1443 z_1(t) - 2.9531 z_2(t) \\ \dot{z}_2(t) = 7.9654 + 0.1555 z_1(t) - 0.8283 z_2(t) \end{cases} \tag{20}$$

4.4 Figures

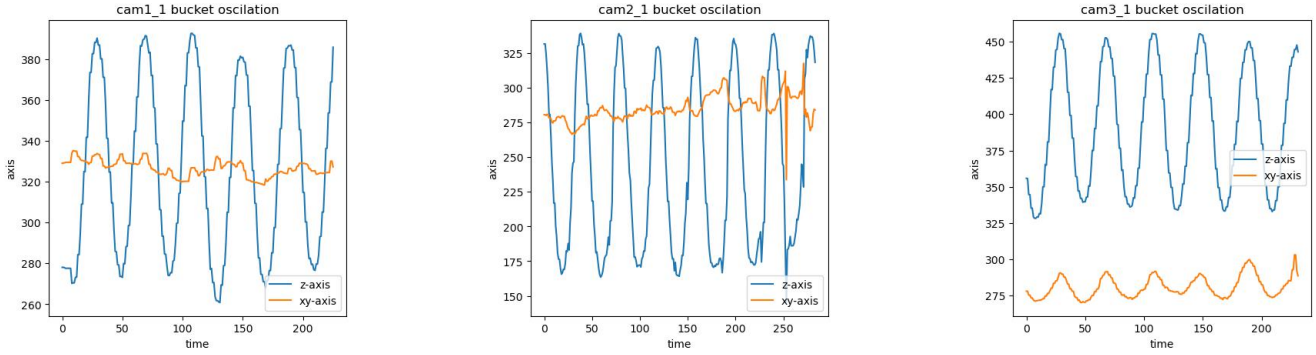

Figure 2: **Ideal Case Tracking** (left) `cam1_1` (middle) `cam2_1` and (right) `cam3_1`, oscillating bucket tracked positions on z-axis moving up and down, as well as sideways jitter on xy-plane (noise and/or pendulum motion).

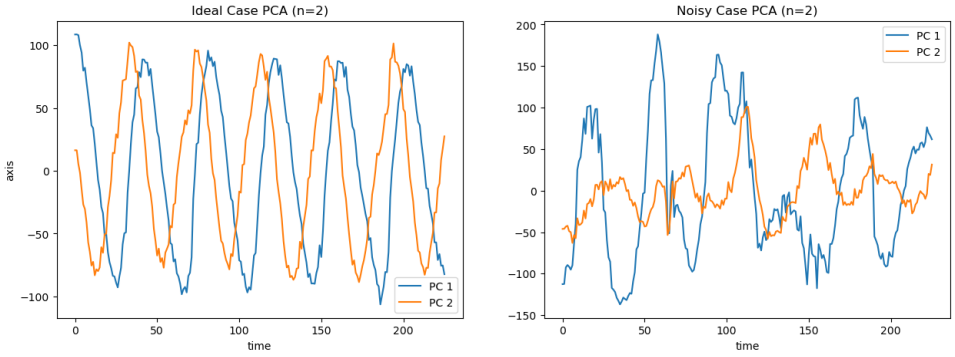

Figure 3: **PCA components** (left) ideal case PCA components, (right) noisy case PCA components

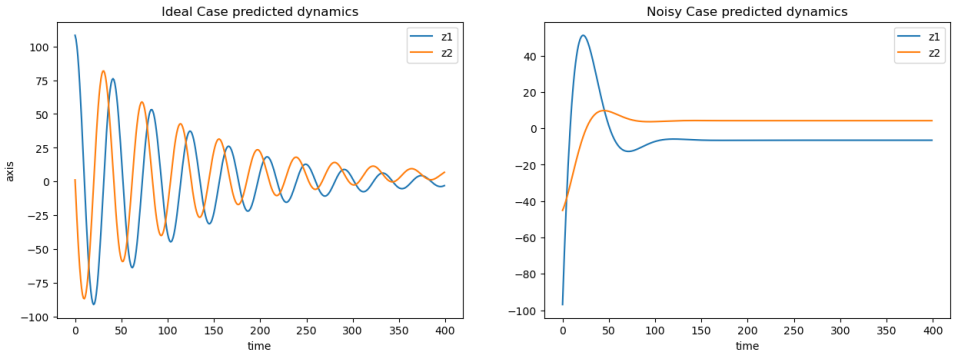

Figure 4: **discrete time SINDy method** (left) ideal case bucket dynamics prediction, (right) noisy case bucket dynamics prediction

5 Summary and Conclusions

The initial step of tracking the video worked out well. The positions were tracked and PCA recovered the 2 components. The issue begins with the implementation of the SINDy method. One of the uncertainties in the final implementation is the choice of functions and how they transform the input. Since the components themselves seem to be a pair of $sin(x)$ and $cos(x)$, it seems like the library should then be x_1 and x_2 instead of $sin(x)$ and $cos(x)$. Moreover, the second uncertainty comes from the definite way of iterating the SINDy's method, where each component is to be removed. Finally, the main difference between EDMD and SINDy observed in this trial was that SINDy produced a very damped oscillator, quickly dying out. For the continuous-time formulations, more work is required to gain any insight of the validity of the results.

References

[1] J. J. Bramburger, Data-Driven Methods for Dynamic Systems, 2023.

[2] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

Appendix A.1 Premade functions used and brief implementation explanation.

1. `pinv`: computes pseudoinverse using the svd method. Used instead of the `inv` for computational speed and for non-square matrices.
2. `PCA`: scikit-learn method to compute PCA using the svd method using user-specified number of components.
3. `DBSCAN`: scikit-learn method to cluster data points using a density based cluster method DBSCAN.

Appendix A.2 Extra Figures

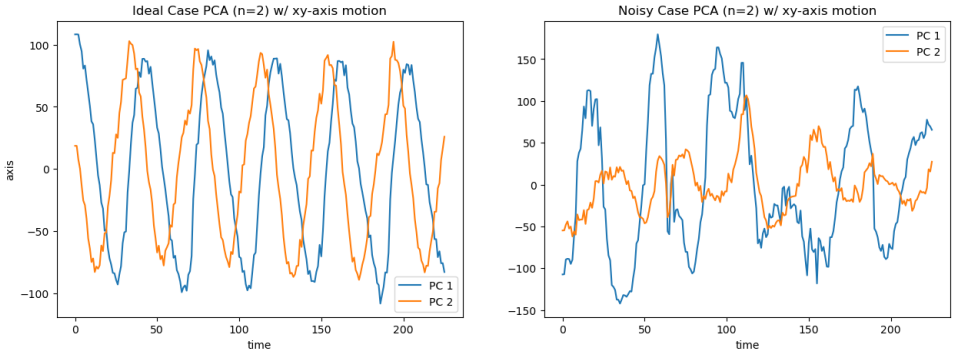

Figure 5: **PCA components** (left) ideal case PCA components, (right) noisy case PCA components where the xy-axis movement is added as dimensions before PCA computation

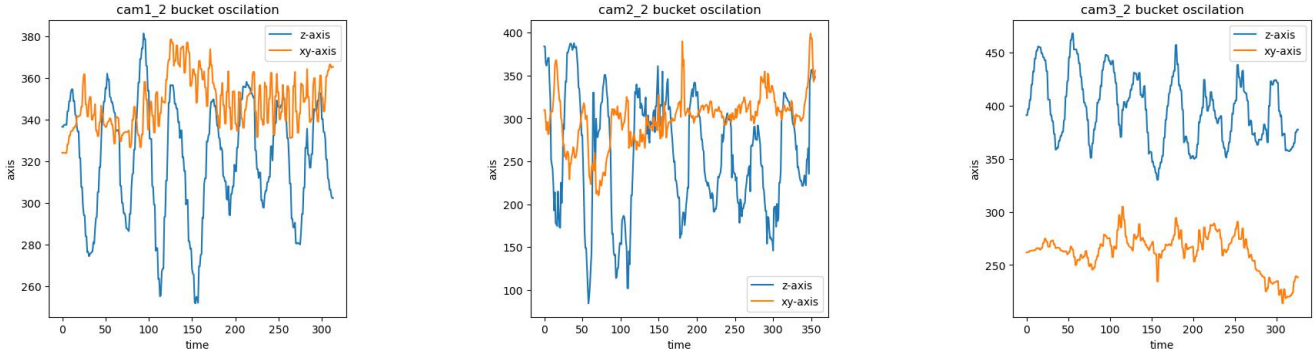

Figure 6: **Noisy Case Tracking** (left) `cam1_2` (middle) `cam2_2` and (right) `cam3_2`, oscillating bucket tracked positions on z-axis moving up and down, as well as sideways jitter on xy-plane (noise and/or pendulum motion).

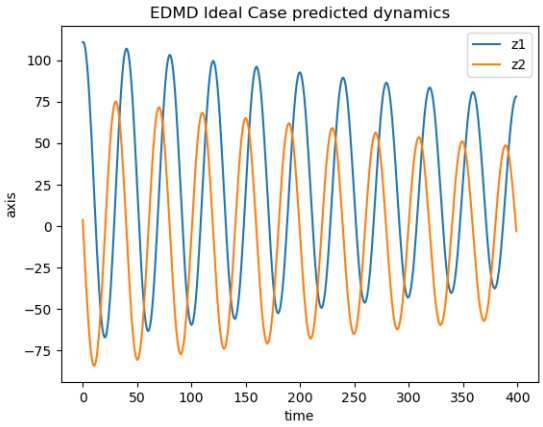

Figure 7: **discrete time E-DMD method** Ideal case bucket dynamics prediction. The difference is the pinv method is applied once to the entire Ψ matrix, generating better results.

Appendix B. Computer codes