Sparse Identification of Nonlinear Dynamical Systems and the Sequential Threshold Least Squares Algorithm

Maurice Constantin Matar

CMA CGM

August 1, 2024

Outline

1. Introduction
2. The SINDy Algorithm
3. Theoretical Background
4. Some Applications
5. Bibliography

INTRODUCTION

Introduction

First, a bit of history...

Kepler's Laws

1609: Kepler formulates the three laws of planetary motion.

=> Kepler's law were empirical relations retrieved from a set of collected data that are a mere description of the observations of planetary orbits as they did not really clarify what happens on a fundamental level so that these orbits are formed.

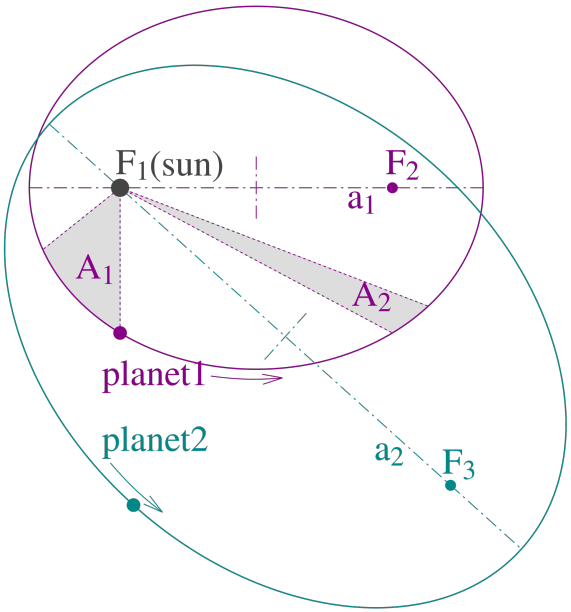
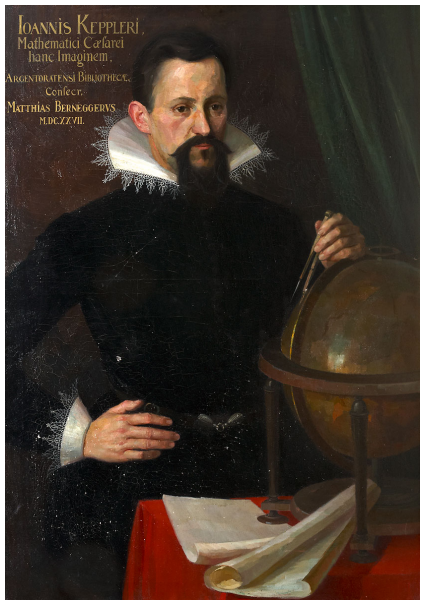


Illustration of Kepler's laws [3]

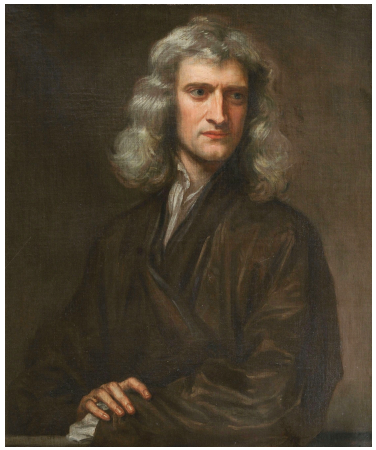


Johannes Kepler [2]

Newton's Laws

1666: Newton comes up with his laws of motion.

$=>$ Unlike Kepler's laws, Newton's laws described what happens fundamentally in order for Kepler's laws to occur. In fact, the latter can be derived from the former (but not vise versa).
$=>$ Moreover, they provided a framework that could be applied to other systems in order to predict their behaviour without having to collect data on them. For example, thanks to Newton's laws, we are able to launch a satellite into space. This would not have been possible with Kepler's laws alone.



Portrait of Sir Isaac Newton [4]

Goal

- With the advent of computer science and machine learning, the way we do science is rapidly changing.
- Aware of the large availability of data on multiple subjects, we are interested in finding a way to not only describe a **dynamical system**, but to identify the underlying laws that govern it by involving new technologies.

Defining A Dynamical System

Dynamical system: A mathematical model that describes how the state of a system evolves over time following a certain function.

The SINDy Algorithm

The SINDy Algorithm

Let us talk about thw Sparse Identification of Nonlinear Dynamics (SINDy) algorithm.

First introduced by Steven L. Brunton, Joshua L. Proctor and J. Nathan Kutz in their 2016 paper [1], the SINDy algorithm is a computational technique used to identify the governing equation(s) of a system from time series data.

The general idea

Essentially, think of it as a machine operating in the following way:

Data $\rightarrow$ **Equation(s)**

	Time (s)	Height (m)	Velocity (m/s)	Acceleration (m/s²
0	0.000000	100.000000	0.000000	-9.8
1	0.102041	99.948928	-1.001020	-9.8
2	0.204082	99.795710	-2.002041	-9.8
3	0.306122	99.540348	-3.003061	-9.8
4	0.408163	99.182840	-4.004082	-9.8
5	0.510204	98.723188	-5.005102	-9.8
6	0.612245	98.161391	-6.006122	-9.8
7	0.714286	97.497449	-7.007143	-9.8
8	0.816327	96.731362	-8.008163	-9.8
9	0.918367	95.863130	-9.009184	-9.8
10	1.020408	94.892753	-10.010204	-9.8
11	1.122449	93.820231	-11.011224	-9.8
12	1.224490	92.645564	-12.012245	-9.8
13	1.326531	91.368753	-13.013265	-9.8
14	1.428571	89.989796	-14.014286	-9.8
15	1.530612	88.508694	-15.015306	-9.8
16	1.632653	86.925448	-16.016327	-9.8
17	1.734694	85.240056	-17.017347	-9.8
18	1.836735	83.452520	-18.018367	-9.8
19	1.938776	81.562838	-19.019388	-9.8
20	2.040816	79.571012	-20.020408	-9.8
21	2.142857	77.477041	-21.021429	-9.8
22	2.244898	75.280925	-22.022449	-9.8
23	2.346939	72.982663	-23.023469	-9.8

$\rightarrow$ $\vec{F_{ext}} = m\vec{a}$

How is this possible?

At first, such a thing a may seem impossible... but with some close examination into how the algorithm works, we can understand how everything is done.

Theoretical Background

Theoretical Background

We consider a dynamical system of n points of the form:

$$\frac{d}{dt}\boldsymbol{x}(t) = \boldsymbol{f}(\boldsymbol{x}(t)) \tag{1}$$

where $\boldsymbol{x}(t) \in \mathbb{R}^n$ is the state of the system at time t and $\boldsymbol{f}(\boldsymbol{x}(t))$ is the equation governing the points system's behavior (such as Newton's second law).

The state vector

Considering that the state of the system is measured at specific points in time $t_1, t_2, \ldots, t_m$ (let's say in given dataset), we can define $\boldsymbol{X}$ as:

$$\boldsymbol{X} = \begin{bmatrix} {}^t\boldsymbol{x}(t_1) \\ {}^t\boldsymbol{x}(t_2) \\ \vdots \\ {}^t\boldsymbol{x}(t_m) \end{bmatrix} = \begin{bmatrix} x_1(t_1) & x_2(t_1) & \ldots & x_n(t_1) \\ x_1(t_2) & x_2(t_2) & \ldots & x_n(t_2) \\ \vdots & \vdots & \ddots & \vdots \\ x_1(t_m) & x_2(t_m) & \ldots & x_n(t_m) \end{bmatrix}$$

$\dot{\boldsymbol{X}}$ is thus:

$$\dot{\boldsymbol{X}} = \begin{bmatrix} \dot{x_1}(t_1) & \dot{x_2}(t_1) & \ldots & \dot{x_n}(t_1) \\ \dot{x_1}(t_2) & \dot{x_2}(t_2) & \ldots & \dot{x_n}(t_2) \\ \vdots & \vdots & \ddots & \vdots \\ \dot{x_1}(t_m) & \dot{x_2}(t_m) & \ldots & \dot{x_n}(t_m) \end{bmatrix}$$

The candidate function matrix

We would now like to construct an $n \times m$ candidate function matrix $\Theta(\boldsymbol{X})$ whose entries are the linear and non-linear terms that $\boldsymbol{f}(\boldsymbol{x}(t))$ might be a function of.

There is a fair amount of flexibility in choosing the terms we want to include in the matrix. The functions that play no role in the actual underlying equation(s) that describe the dynamical system will be removed thanks to an algorithm that we will explore later on.

All of this is very important, as it will allow us to include a lot of terms in the $\Theta(\boldsymbol{X})$, which are all essentially guesses of what the actual parameters of $\boldsymbol{f}(\boldsymbol{x}(t))$ are.

The form of the candidate function matrix

As mentioned before, candidate function matrix $\Theta(\boldsymbol{X})$ may contain linear and non-linear terms, such as polynomials and trigonometric functions, as well as the spatial derivatives of the vector $\boldsymbol{X}$. Here is an example of what a candidate function matrix looks like:

$$\Theta(\boldsymbol{X}) = \begin{bmatrix} 1 & \boldsymbol{X} & \boldsymbol{X}^2 & \dots & \cos(\boldsymbol{X}) & \sin(\boldsymbol{X}) & \dots & \frac{d}{dx}\boldsymbol{X} & \dots \end{bmatrix}$$

The sparsity matrix

Now, we would like to write $\boldsymbol{f}(\boldsymbol{x})$ as the product of $\Theta(\boldsymbol{X})$ and Ξ. Ξ is a matrix made up of sparse columns ξ_i (columns with 0s and 1s), whose job is to select the active terms in $\Theta(\boldsymbol{X})$.
The initial equation can be thus rewritten as:

$$\frac{d}{dt}\boldsymbol{x}(t) = \boldsymbol{f}(\boldsymbol{x}(t)) \iff \dot{\boldsymbol{X}} = \Theta\Xi \tag{2}$$

Finding the concrete expression of the sparsity matrix

Here comes the interesting part.

Our goal is to find the concrete expression of Ξ in order to get know what is the function $\boldsymbol{f}$.

The main idea

The main idea: Perform a linear regression in order to determine ξ_i (i.e. the columns of the sparsity matrix Ξ) for $i \in [\![1, n]\!]$.

Linear Regression

Linear regression is a statistical method that allows us to determine a linear relationship between two variables: one depending on the other (target) and one independent (predictor).

Aim: Determine the concrete expression of the equation of the line $y = ax + b$ that best fits the data. If x is a vector (i.e. we have multiple predictors), we want to determine the vector $\boldsymbol{\beta}$ that fits best the data:

$$y = \sum_{i=1}^{n} X_i \beta_i + \beta_0 = \begin{bmatrix} 1 & X_1 & X_2 & \dots & X_n \end{bmatrix} \cdot \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_n \end{bmatrix} = \boldsymbol{X}\boldsymbol{\beta} \qquad (3)$$

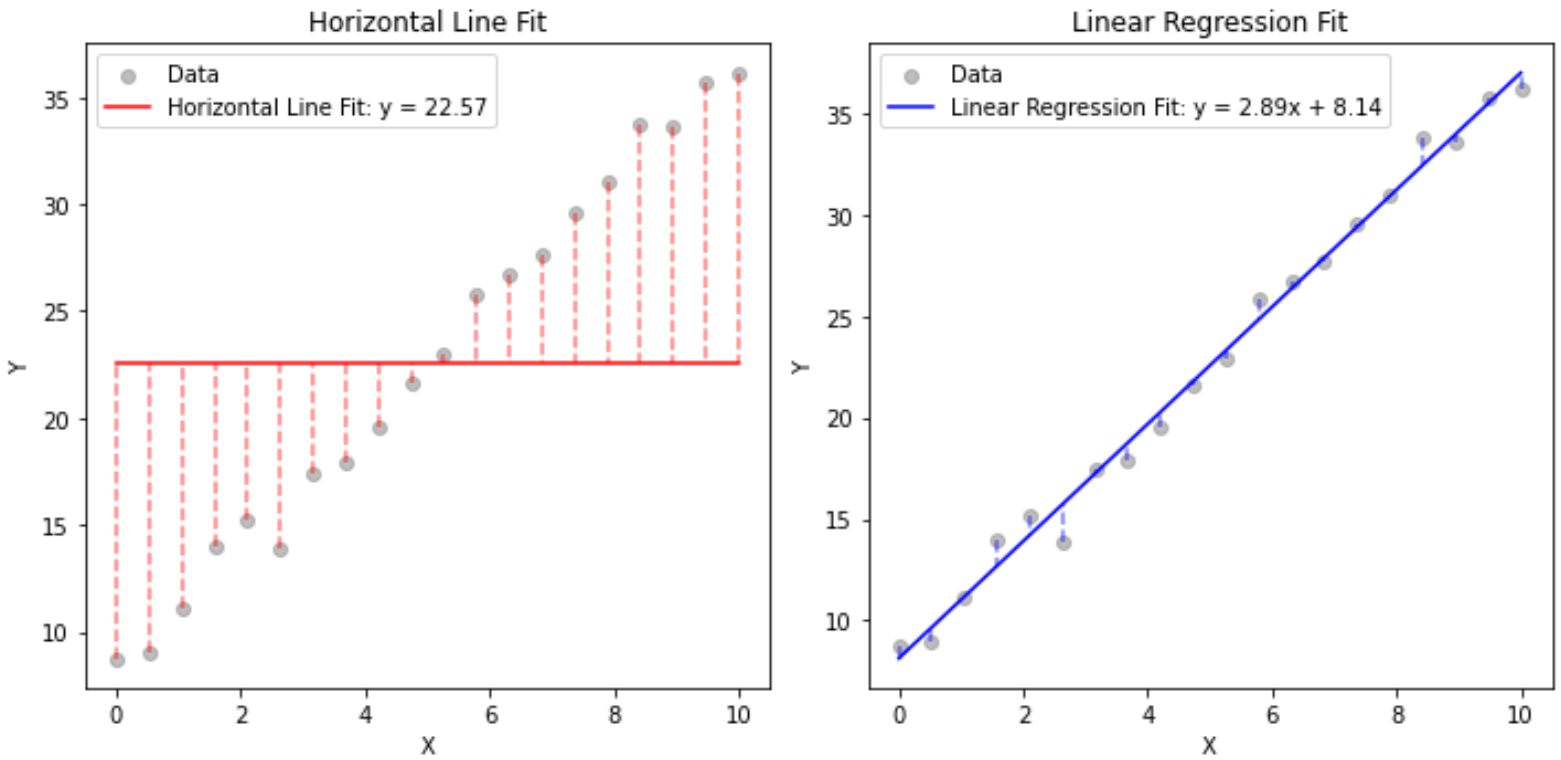
Multiple targets and predictor case

If y and x are both vectors (i.e. multiple targets and multiple predictors), the previous equation becomes:

$$\boldsymbol{Y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & X_{11} & X_{12} & \dots & X_{1n} \\ 1 & X_{21} & X_{22} & \dots & X_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & X_{n1} & X_{n2} & \dots & X_{nn} \end{bmatrix} \cdot \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_n \end{bmatrix} = \boldsymbol{X}\boldsymbol{\beta} \tag{4}$$

Least Squares

The linear regression coefficients can be estimated using the Least Squares method. Essentially, we look for the coefficients that minimize the distance between the constructed line and all the data points' targets.



Comparison between a horizontal line fit and a linear regression

Least Squares formula

As the difference between some components can be sometimes positive and sometimes negative (due to some data points being above or below the line). This could make the fitted line look better than it actually is.

$=>$ To fix the issue, we thus need a way to ensure that all the terms we are working with are positive. We could take the absolute value, but this will make the math a bit tricky. We therefore resort to taking the square of the differences, hence the name Least Squares. We thus look to find the vector $\boldsymbol{\beta}$ by:

$$\boldsymbol{\beta} = \underset{\boldsymbol{\beta'}}{\operatorname{argmin}} \| \boldsymbol{Y} - \boldsymbol{X}\boldsymbol{\beta'} \|_2^2 \tag{5}$$

The problem

The problem: The Least Squares method might be prone to overfitting and can yield undesirable results when performed on a large number of predictors.

$=>$ We thus need to find a middle ground with a model that predicts accurately enough the data but with a somewhat interpretable result.

The solution

The solution: Sparse regression.

$=>$ We would like to "select" the predictors that have the most impact when predicting our targets and discard the rest.

The STLSQ algorithm

One algorithm that does exactly that is the Sequential Threshold Least Squares (STLSQ) algorithm.

The main idea is to perform a Least Square regression multiple times with a pre-defined threshold each time. The coefficients below this threshold get nullified, and we should observe some sort of convergence.

Description of the STLSQ Algorithm

$\text{STLSQ}(\boldsymbol{X}, \boldsymbol{Y}, \lambda, N)$

Apply the Least Square Method
$\boldsymbol{\beta} = \text{argmin}_{\boldsymbol{\beta'}} \|\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{\beta'}\|_2^2$

Keep the coefficients of $\boldsymbol{\beta}$ that are smaller than a given threshold λ
$\text{lowcoefficients} = \{\text{i} \mid |\beta_i| \leq \lambda\}$
$\boldsymbol{\beta}[\text{lowcoefficients}] = 0$

Repeat the steps above with the resultant vector $N - 1$ times
$\boldsymbol{\beta}[\text{lowcoeffcients}] = \text{STLSQ}(\boldsymbol{X}, \boldsymbol{Y}, \lambda, N - 1)$
return $\boldsymbol{\beta}$

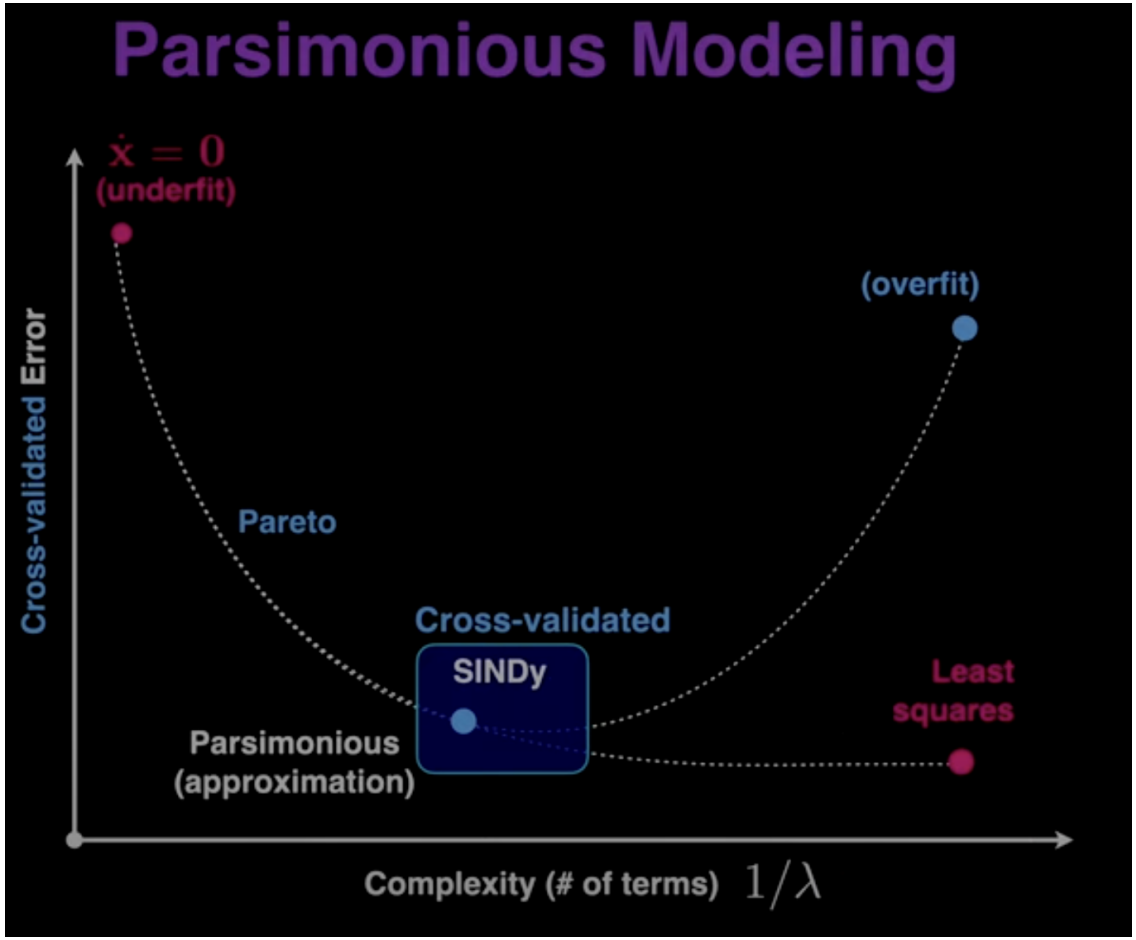
Summary of the STLSQ algorithm

Back to our case

- In our case, the target vector is the vector whose components are the time derivatives of the n data points of our system: $\dot{\boldsymbol{X}}$.
- The predictors are the terms in the candidate function matrix $\Theta(\boldsymbol{X})$.
- We are looking to approximate Ξ.

Retrieving the equations

$=>$ By applying the STLSQ algorithm for different values of λ, we select the value of λ that minimizes the error after testing.



Source: https://youtu.be/pY2iJnngk4g?si=CQ9GexZQNHQUaZZS

Resisting noise

One of the strenghts of STLSQ compared to other sparse regression methods such as LASSO is that STLSQ is quite good in resisting noise. The proof of that statement will not be covered in this presentation.

Constructing a model

After retrieving the important terms from the candidate function matrix, SINDy then builds a model using them in order to describe the system.

Summary

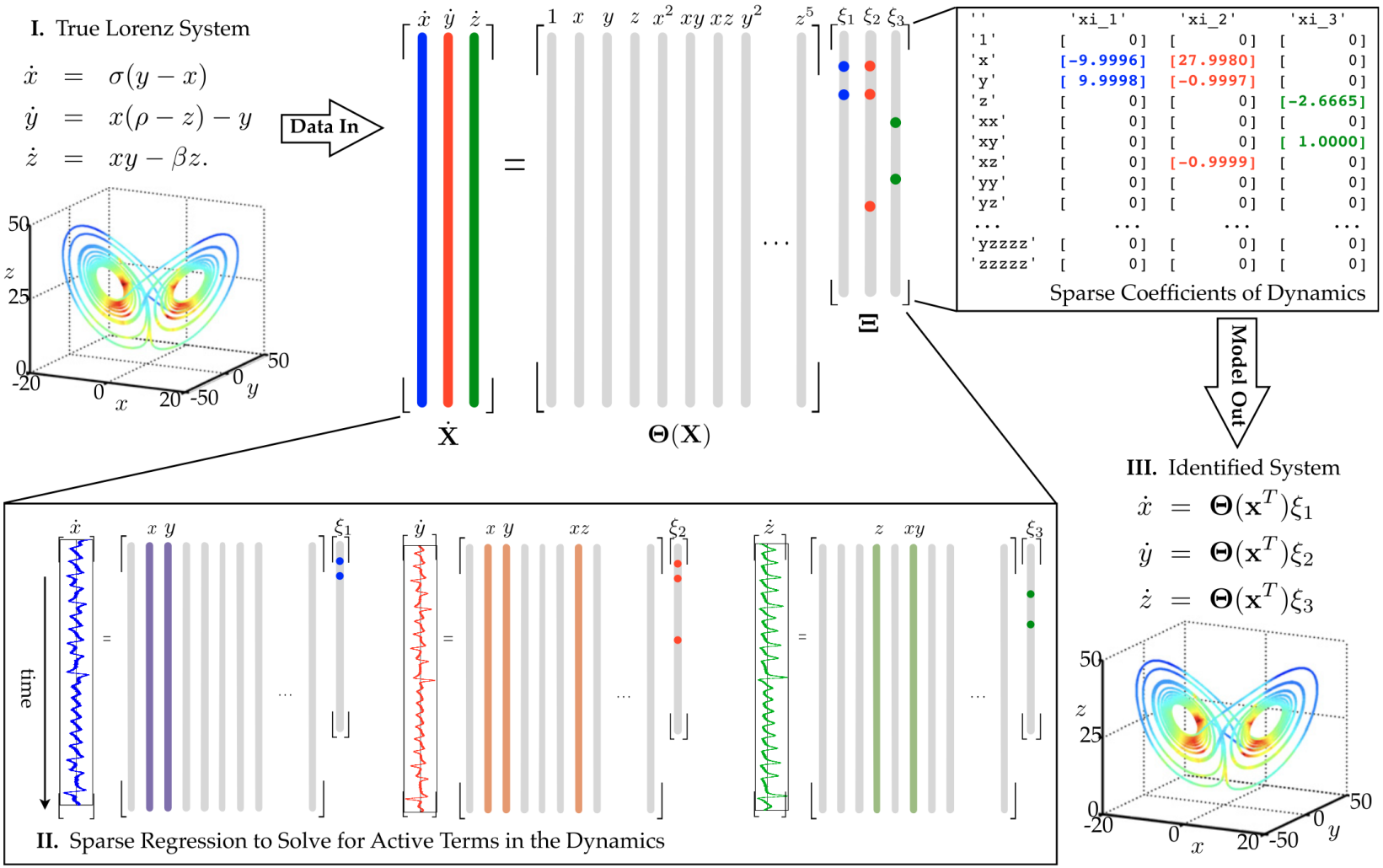


Figure: Schematic of the SINDy algorithm, demonstrated on the Lorentz equations [1]

Some Applications

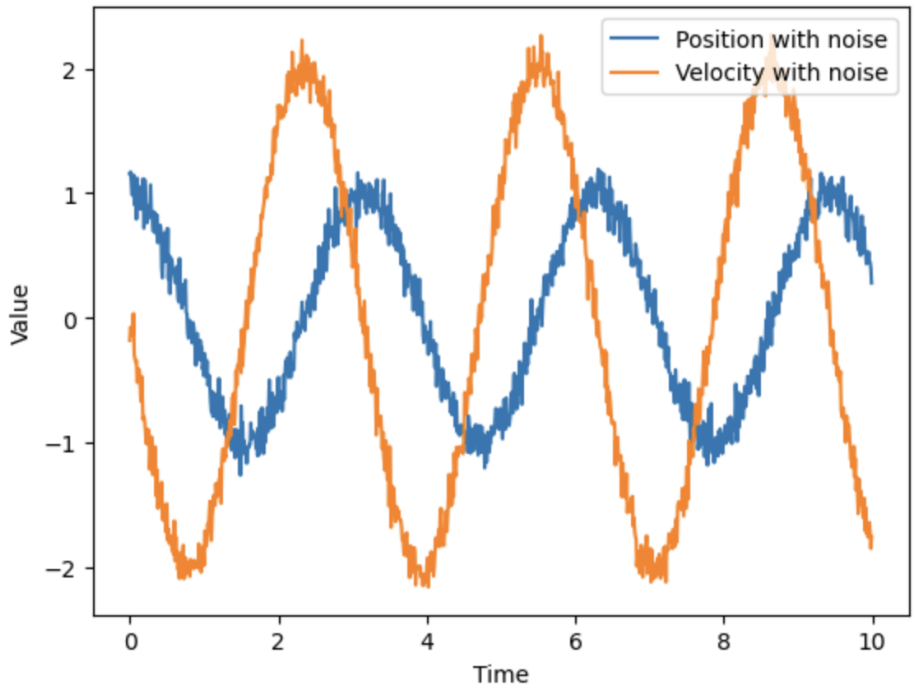
Application I: Simple Harmonic Oscillator

Simple Harmonic oscillator

We consider a simple harmonic oscillator of $\omega_0 = 2$ described by:

$$\ddot{x} + \omega_0^2 x = 0 \tag{6}$$

We generate data for such a harmonic oscillator and we add a bit of noise to it (to emulate real life data acquisition):



Position and velocity of a harmonic oscillator

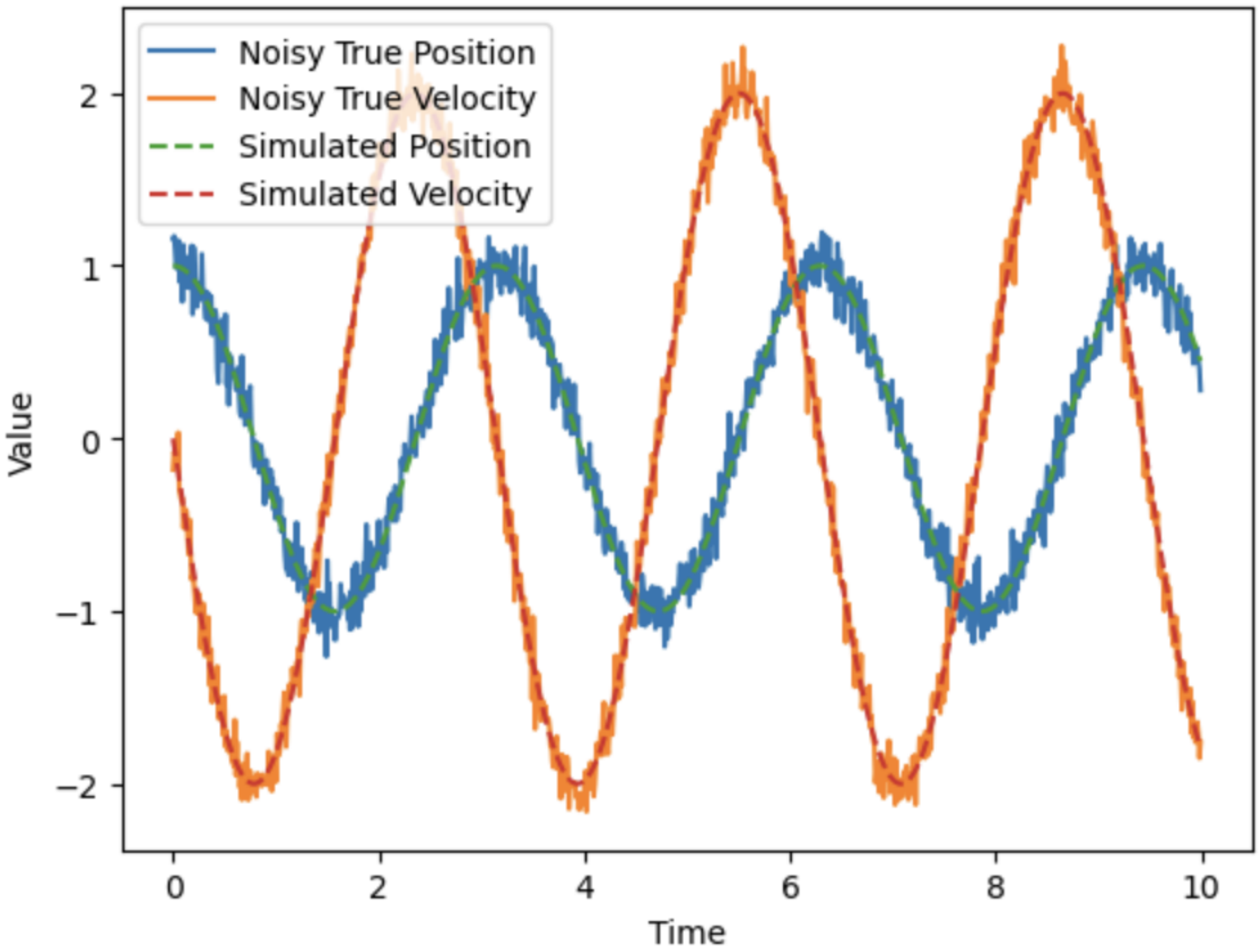
Applying SINDy to a simple harmonic oscillator in Python

[16]:
```
# Stack the position and velocity data
X = np.vstack((x, v)).T

# Create a SINDy model
model = ps.SINDy()

# Fit the model to the data
model.fit(X, t=dt)

# Print the identified equations
model.print()
```

```
(x0)' = 1.000 x1
(x1)' = -3.990 x0
```

Output of pySINDy (SINDy's Python library) when applied to the simple harmonic oscillator we considered previously

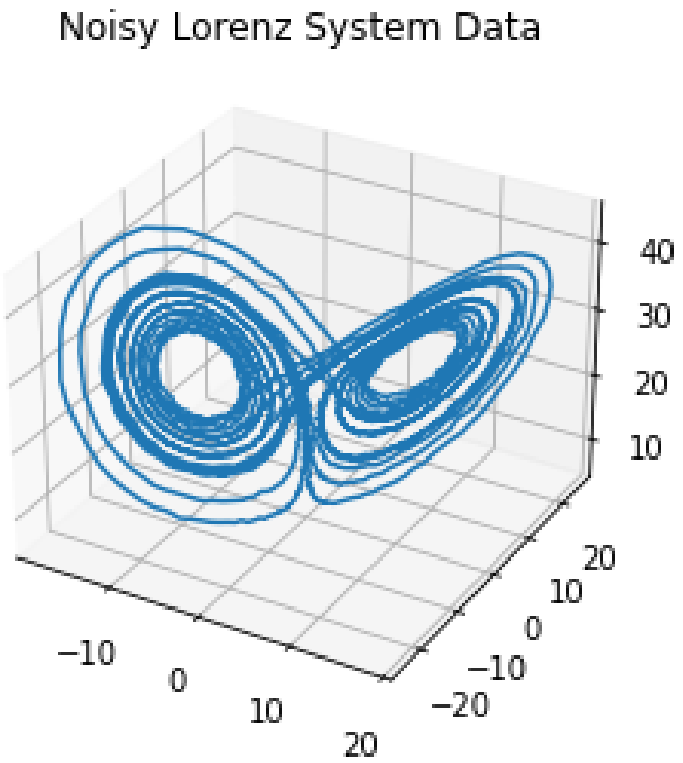
Comparison between the models

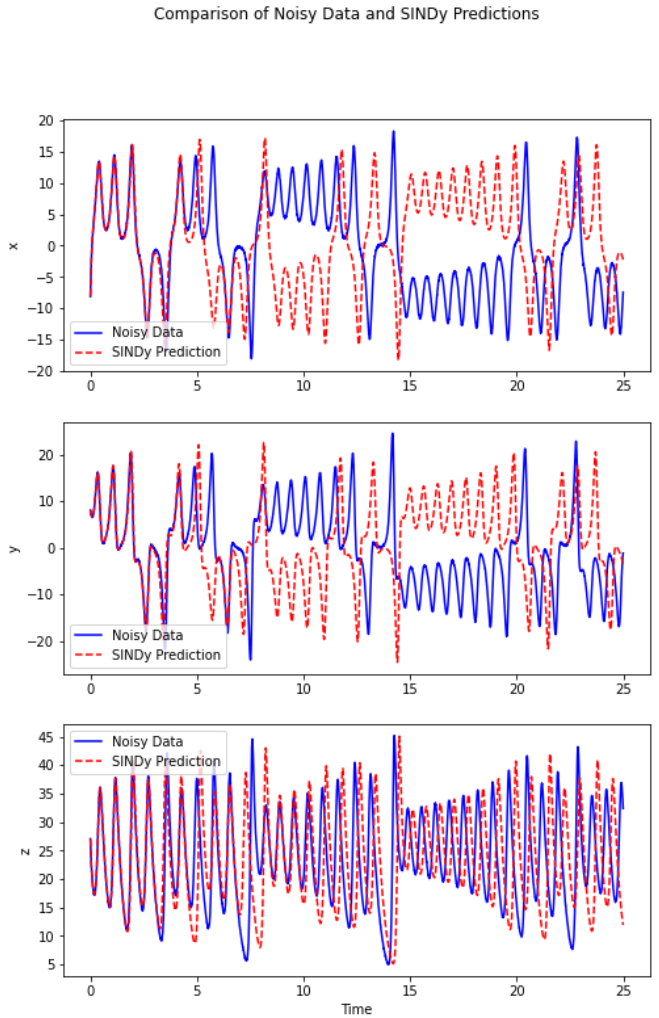


Plot showing the data and the predictions

Application II: Lorenz System

Applying SINDy to a Lorenz system in Python





Results of applying SINDy to a Lorenz system

Bibliography

[1] Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. "Discovering governing equations from data by sparse identification of nonlinear dynamical systems". In: *Proceedings of the national academy of sciences* 113.15 (2016), pp. 3932–3937.

[2] Wikimedia Commons. *File:JKepler.jpg — Wikimedia Commons, the free media repository*. [Online; accessed 31-July-2024]. 2024. URL: %5Curl%7Bhttps://commons.wikimedia.org/w/index.php?title=File:JKepler.jpg&oldid=874824950%7D.

[3] Wikimedia Commons. *File:Kepler laws diagram.svg — Wikimedia Commons, the free media repository*. [Online; accessed 31-July-2024]. 2023. URL: %5Curl%7Bhttps://commons.wikimedia.org/w/index.php?title=File:Kepler_laws_diagram.svg&oldid=727156686%7D.

[4] Wikimedia Commons. *File:Portrait of Sir Isaac Newton, 1689 (brightened).jpg — Wikimedia Commons, the free media repository*. [Online; accessed 31-July-2024]. 2024. URL: %5Curl%7Bhttps: