ISB SU22 Undergraduate Intern Project: pySINDy for System Modeling

Ainsley Lai

July 18, 2023

Abstract

A nifty little overview of learning to use SINDy. The purpose of writing this is to record how things went and why certain actions were taken so that it'd be easier to understand the project in the future for anyone who wants to read it. Not supposed to be a formal document.

1 Introduction

1.1 Inverse dynamics problems in biology and medicine

The overarching idea here is "getting equations from data" to use them in dynamic/mechanistic modeling (using more specific equations rather than statistical models). As per the book, Dynamical Models in Biology by Ellner [EG06], having a dynamic model allows us to see the relations between variables/states in the system, giving greater insight into the system as a whole rather than just describing a system as a whole under a certain context, like the law of mass action or enzyme kinetics. This *might* allow us to find underlying rules for a system we model (think Keppler modeling orbits using tons of data and Newton using his laws of motion which can be used to describe the orbits and more).

Having a dynamic model equation allows us to, in theory, use less data than predictive statistical models as well since those models need a lot of data to verify probabilities (I think). Since it's an equation, we can also see how a system changes far out in time. For that though, the accuracy is up to debate... That aside, using dynamic models may serve a purpose in acting as a proxy for testing when we don't want to use humans.

1.2 SINDy: Sparse Identification of Nonlinear Dynamics

For this, we'll be using the SINDy method [BPK16]. The general driving mathematical idea is below:

$$\dot{\mathbf{X}} = \boldsymbol{\Theta}(\mathbf{X})\,\boldsymbol{\Xi} \tag{1}$$

where $\mathbf{X}$ is our data matrix in shape (time $\times$ features) and $\boldsymbol{\Theta}$ are different chosen functions for each variable and $\boldsymbol{\Xi}$ are the sparse vectors that determine which functions of our feature data are relevant. $\dot{\mathbf{x}}$ can be found using finite differences. In other words, all we need is the data and a defined function library (and setting hyper-parameters like how we want to find the sparse vectors).

1.3 Tech overview

Main python packages/stuff in use:

- pySINDy
- Jupyter Notebook
- Numpy
- Scipy

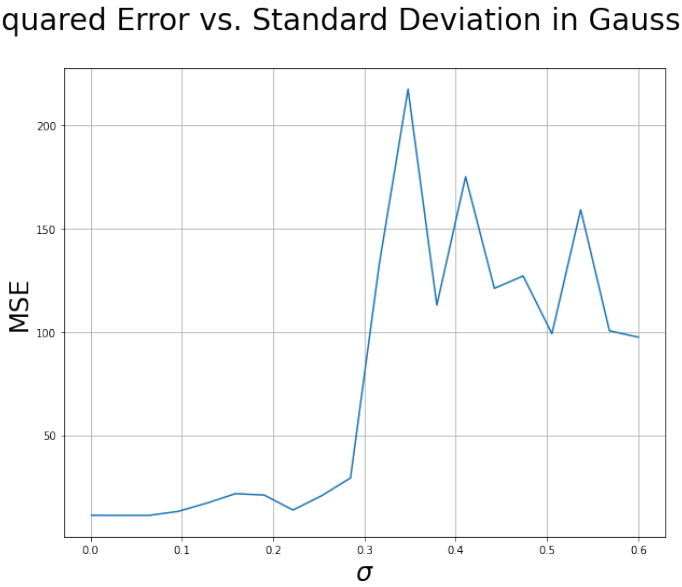

Figure 1: MSE of vanilla pySINDy on Gene Switch system w/ noise

- SymPy

Google Cloud computing was also used to work with the INCOV set since it was already stored on there and didn't need Citi training. At the end of the project, the incov-pysindy notebook had a machine type of "Efficient Instance, 32 vCPUs, 32 GB Ram".

2 pySINDy

GitHub: https://github.com/dynamicslab/pysindy
pySINDy is a Python implementation of the SINDy algorithm and a few spin-offs extensions like PDE functionality and SINDy-PI (SINDy multiverse?!).

In terms of the code, it requires us to set up a function library (with names), set up an optimizer for finding the sparse vector (or I think that's what it's for) with a threshold controlling the sensitivity and using ps.SINDy(parameters here) to create a skeleton model. From there, we fit it to our training data and a uniform time interval (our data has to be a uniform time series). There's also an option to test on multiple trajectories/samples as well to improve fit. For instance, fitting the model on a dataset with multiple initial conditions (different IC for each sample).

2.1 Vanilla pySINDy

Our project and problems begin with using the basic implementation of pySINDy. In short, basic SINDy doesn't do so well with rational functions. Consider:

$$\frac{1}{1+x}, \frac{1}{2+x}, \frac{1}{3+x}, \dots \tag{2}$$

and related functions. SINDy has a hard time differentiating between them all thus reducing sparsity and accuracy of the generated model. It also handled noise horribly. Figure 1 shows this. Specific error values are not the same across each run but it should be noted that they can get pretty high.

2.2 SINDy-PI

In comes SINDy-PI [KKB20], which is supposed to fix the issue by taking an implicit equation perspective and parallelizing the work (which pySINDy doesn't actually do) for different functions with

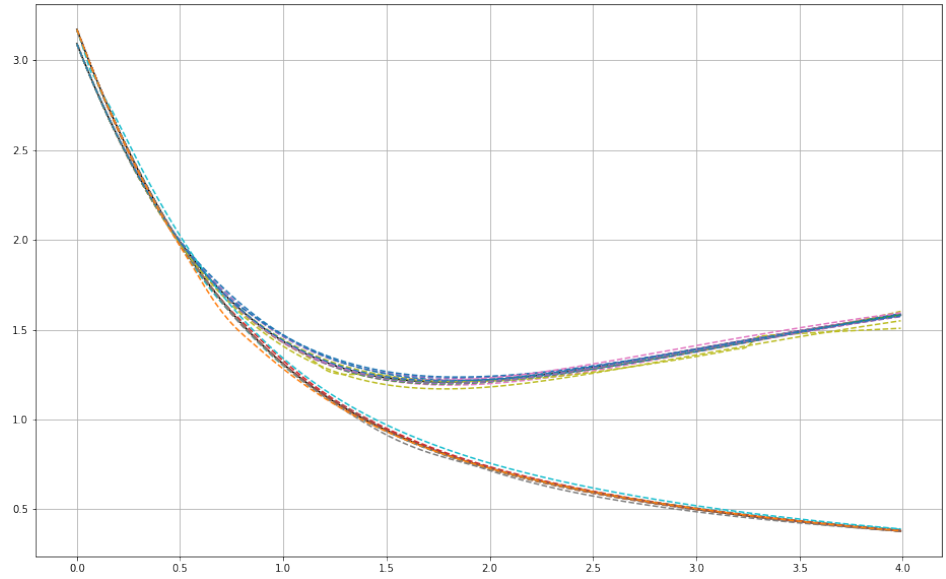

Figure 2: A really good set of models from SINDy-PI for the Network Switch system

this driving idea/algorithm:

$$\underbrace{C(\mathbf{X}, \dot{\mathbf{X}})}_{\text{LHS}} = \underbrace{\boldsymbol{\Theta}(\mathbf{X}, \dot{\mathbf{X}})}_{\text{RHS}} \underbrace{\boldsymbol{\Xi}}_{\text{Sparse Vector}} \tag{3}$$

LHS would be some function that we think can be in the true system equation. Refer to hte paper for a better illustration. In the code, this is implemented via the SINDyPI Optimizer for the SINDy object but for our work, we used the SINDyPI function library as well so that we can isolate to just using a single $\dot{x}_i$ in our equations.

Long story short, models it produces are a whole lot better (but not great) but the equations still aren't correct, even without noise. More specifically, they aren't sparse. Sometimes the models fit the true solution pretty well, sometimes not and there would be some models that, while generally fitting the derivative shape, have trajectories that are obviously off the mark.

Regarding implementation, they use CVXPY and SymPy. The latter is just for putting a model equation back into explicit form for an ODE solver (solve_ivp) to handle and simulate while the former is to minimize the cost funciton and find the coefficients in the model equations. On the toy example, there seemed to be issues where `NaN` coefficients were generated for some model equations when the sample size was high (>100).

3 INCOV

3.1 Data Reduction

There were 140 samples with 453 samples so we did a PCA to reduce it down to 3 samples, which was where the elbow in the scree plot was. pySINDy's implementation of the SINDy-PI algorithm also supposedly has issues with going up to 4 or more features anyways so lucky us. After projecting it to a 3 feature PC space, we can now process the data.

3.2 Data Processing

To work on the INCOV data set with SINDy, we had to do a few tweaks since it only had 3 non-uniform time points for each patient (sample). To fix that, tSMOTE was used to fit them onto a uniform time

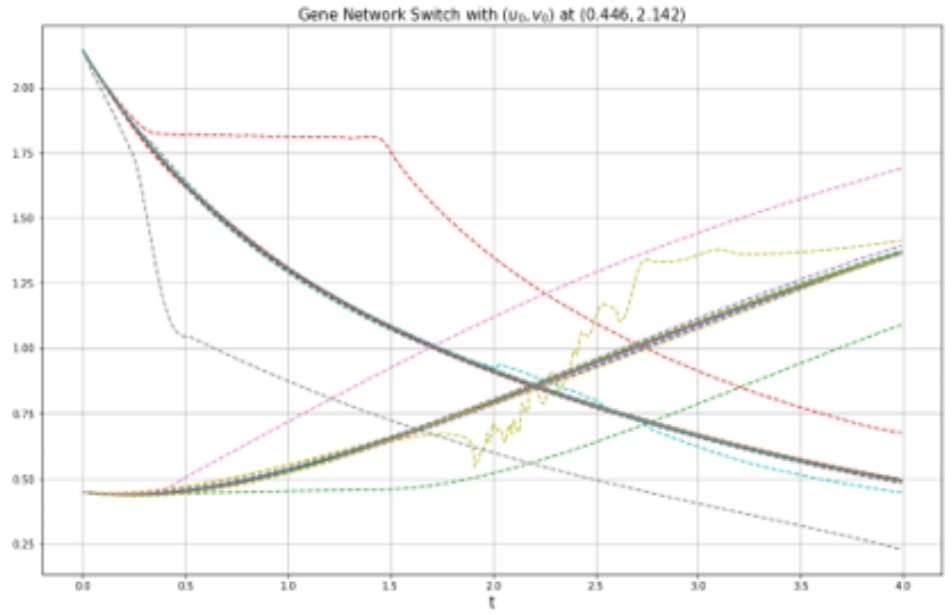

Figure 3: A 'less' good set of models from SINDy-PI for the Network Switch system with img compression from saving from ppt

point of 30 times. Since the data was more dense in the beginning of the time frame, we cut down our time frame to the first 20 time points to avoid any problems that interpolating too far out might cause. Then a Savitsky-Golay filter was applied to smooth out the data and hopefully iron out any issues After that, we ran cubic spline interpolation between all the time points to generate more data for SINDy to use.

100 samples were used to train and 2 were set aside to test (predicted computation for remaining 40 would've taken too long; other issues came up as well).

3.3 pySINDy Modeling

Used a custom defined generalized library containing 3 different libraries: a custom library that just had a constant term, another custom library that had the polynomial terms (SymPy had trouble working with the default SINDy polynomial library), and a SINDyPI library that contained just the $\dot{x}$ term so that we can isolate it to a specific $\dot{x}_i$ in the generalized library initialization. Optimizer was the SINDyPI optimizer.

Idea behind this was to generate a four-nested list that allows us to store each equation, rounded or simplified, for each feature. This was done by encapsulation all the SymPy formatting stuff into a function that returns the list of equations for a certain model (for a certain feature). Then, we make a loop for the number of features we are going through and generate a model for each one and format the equation and add it to a list. This basically just gives us the equations for a model for a certain $\dot{x}_i$ that we can use to simulate the system.

A small note: generating a new SINDy model overwrites the previous one used so you can't call past models. Might be some weird object stuff going on.

3.4 Issues

3.4.1 NaN Coefficients

Generated model equations would sometimes have `NaN` coefficients for a couple models, which prevented the equations from being formatted. This can be fixed by replacing those coefficients with 0

instead.

This issue came up in the toy example when I used a large amount of samples (≥ 150). This probably has to do with the CVXPY package.

3.4.2 Numerical Solving

When attempting to simulate the system via numerical integration with default options, it got stuck and couldn't finish the 4th out of 20 models for the first feature. Switching to LSODA didn't help but Radau and BDF integration methods let us finish the models but it failed the integration and exited the integration interval early, meaning that we can't compute errors since we have incomplete simulation sets. Beefing up the cloud instance didn't help.

4 Possible Future Directions

- Compare SINDy model to known biological system models to assess accuracy
- Scale examples up to higher dimensions like the 6-D repressilator gene clock system in Ellner's book [EG06] to see how well the system scales. SINDyPI algorithm might not do so well with 4-D and up systems (see Github; they mention scaling issues somewhere).
- Contact related SINDy authors. SINDyPI original implementation may be different from python implementation. As of writing, SymINDy (symbolic regression spin-off of SINDy) dropped recently so that may be relevant.
- Investigate how to fix the numerical solving issue

References

[BPK16] Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 113(15):3932–3937, 2016.

[EG06] Stephen P. Ellner and John Guckenheimer. *Dynamic Models in Biology*. Princeton University Press, Princeton, NJ, 1. edition, 2006.

[KKB20] Kadierdan Kaheman, J. Nathan Kutz, and Steven L. Brunton. Sindy–pi: A robust algorithm for parallel implicit sparse identification of nonlinear dynamics. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 476(2242):20200279, Oct 2020.