SINDY for Systems with Actuation: Modelling and Control

ESAM479 Individual Project

Jonathan Bosnich

March 2022

1 Introduction

In the second half of this course, we've investigated two different data-driven methods for modelling complex systems: sparse identification of nonlinear dynamics (SINDY) and dynamic mode decomposition (DMD). We've seen that both of these methods can identify accurate models of complex systems by way of analyzing measured data in lieu of a first-principles approach that requires full knowledge of the governing system equations. We've also seen that SINDY and DMD can fail to accurately generate models when the measured data is not sampled at a sufficient rate or is contaminated with noise. However, there is another class of systems for which SINDY and DMD (in their original formulations) will be sure to fail—actuated systems with control inputs.

Because modelling and controlling actuated systems is ubiquitous in applied sciences, it would be quite impactful for SINDY and DMD to be extended to include systems with actuation. Unsurprisingly, that's exactly what the architects of the original formulations of SINDY and DMD have done. In [2] Proctor et al introduce DMDc (Dynamic Mode Decomposition with Control) which extends the idea of dynamic mode decomposition to include controlled systems. In [1] Kaiser et al introduce a SINDY variant with control SINDYc and SINDY-MPC (model predictive control using SINDY) and demonstrate SINDY-MPC on a number of dynamical systems. Additionally, Kaiser et al use DMDc and neural networks within the MPC framework and compare all three data-driven MPC strategies.

For the purposes of this report, I will only give an exposition of SINDYc and SINDY-MPC. First, I will highlight the key difference between SINDY and SINDYc, namely that the library of candidate nonlinear functions includes cross terms with the actuation variable u. Second, I will briefly discuss model predictive control (MPC) and how SINDYc can can be incorporated into this control scheme.

2 SINDY with Control

The general idea of SINDY is to use time series data and sparse regression to extract a parsimonious system of nonlinear functions from a library of candidate functions that will well-approximate the true dynamical system. While this method is dependent on the assumption that the true system dynamics is governed by only a few nonlinear terms, Brunton et al point out that this is usually the case for physical systems [3]. However, the original formulation of SINDY is restricted to the case of a general unforced dynamical system

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t)), \tag{1}$$

where $\boldsymbol{x} \in \mathbb{R}^n$ is the state vector and the smooth nonlinear function $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^n$ define the system equations of motion. We are often concerned with modelling and/or controlling systems that are forced by a control input, i.e. any robotic system, which is of the form

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t)), \tag{2}$$

where $\boldsymbol{u} \in \mathbb{R}^q$ is the control input and the smooth nonlinear function becomes $\boldsymbol{f} : \mathbb{R}^n \times \mathbb{R}^q \to \mathbb{R}^n$. Not surprisingly, there is a natural and straightforward way to extend the SINDY algorithm to include modelling of such actuated systems.

First, we must again assume that the right hand side of (2) has only a few terms. But since we've already made this assumption for the right hand side of (1), it seems safe to assume that adding a control input won't considerably complicate things. For example, proportional control only changes the coefficients of the RHS terms; it doesn't add any additional nonlinear functions to the RHS. Then, assuming that $\boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t))$ is dependent on only a few terms, the original algorithm for SINDY (which I'm assuming my reader is quite familiar with) needs only to be altered in the following ways:

(a) For every snapshot in time t_i, measure both the state $\boldsymbol{x}(t_i)$ and the control input $\boldsymbol{u}(t_i)$

(b) Expand the library of candidate functions $\boldsymbol{\Theta}(\boldsymbol{x})$ to now be $\boldsymbol{\Theta}(\boldsymbol{x}, \boldsymbol{u})$, which will include nonlinear cross terms in $\boldsymbol{x}$ and $\boldsymbol{u}$

A general example of the library $\boldsymbol{\Theta}(\boldsymbol{x}, \boldsymbol{u})$ including polynomial and sine terms is

$$\boldsymbol{\Theta}(\boldsymbol{x}, \boldsymbol{u}) = \begin{bmatrix}\mathbf{1} & \boldsymbol{x} & \boldsymbol{u} & (\boldsymbol{x} \otimes \boldsymbol{x}) & (\boldsymbol{x} \otimes \boldsymbol{u}) & \cdots & \sin(\boldsymbol{x}) & \sin(\boldsymbol{u}) & \sin(\boldsymbol{x} \otimes \boldsymbol{u}) & \cdots\end{bmatrix}, \tag{3}$$

where $\boldsymbol{x} \otimes \boldsymbol{u}$ is the vector of all product combinations of the components of $\boldsymbol{x}$ and $\boldsymbol{u}$.

The full algorithm for SINDYc is as follows.

Algorithm 1 SINDY with Control (SINDYc)

```
1: Measure $m$ snapshots of the $n$-dimensional state $\boldsymbol{x}$ and the $q$-dimensional control input $\boldsymbol{u}$
2: If possible, also measure $m$ snapshots of the state derivative $\dot{\boldsymbol{x}}$. Otherwise, estimate $\dot{\boldsymbol{x}}$ from the measured $\boldsymbol{x}$ using numerical differentiation
3: Construct the data matrices
     $\boldsymbol{X} = \begin{bmatrix} | & | & & | \\ \boldsymbol{x}_1 & \boldsymbol{x}_2 & \cdots & \boldsymbol{x}_m \\ | & | & & | \end{bmatrix}, \quad \dot{\boldsymbol{X}} = \begin{bmatrix} | & | & & | \\ \dot{\boldsymbol{x}}_1 & \dot{\boldsymbol{x}}_2 & \cdots & \dot{\boldsymbol{x}}_m \\ | & | & & | \end{bmatrix}$
   and
     $\boldsymbol{U} = \begin{bmatrix} | & | & & | \\ \boldsymbol{u}_1 & \boldsymbol{u}_2 & \cdots & \boldsymbol{u}_m \\ | & | & & | \end{bmatrix}$
4: Choose a library of candidate nonlinear functions $\boldsymbol{\Theta}(\boldsymbol{x}, \boldsymbol{u})$, for example (3)
5: Evaluate $\boldsymbol{\Theta}(\boldsymbol{x}, \boldsymbol{u})$ using $\boldsymbol{X}$ and $\boldsymbol{U}$. Using (3) as an example
     $\boldsymbol{\Theta}(\boldsymbol{X}, \boldsymbol{U}) = \begin{bmatrix}\mathbf{1}^T & \boldsymbol{X}^T & \boldsymbol{U}^T & (\boldsymbol{X} \otimes \boldsymbol{X})^T & (\boldsymbol{X} \otimes \boldsymbol{U})^T & \cdots & \sin(\boldsymbol{X})^T & \sin(\boldsymbol{U})^T & \sin(\boldsymbol{X} \otimes \boldsymbol{U})^T & \cdots\end{bmatrix}$
6: Compute the sparse matrix $\boldsymbol{\Xi}$ by solving the following sparse regression problem using a sequentially thresholded least squares approach (see Algorithm 2):
     $\boldsymbol{\xi}_k = \arg\min_{\hat{\boldsymbol{\xi}}_k} \frac{1}{2} \|\dot{\boldsymbol{X}}_k - \hat{\boldsymbol{\xi}}_k \boldsymbol{\Theta}^T(\boldsymbol{X}, \boldsymbol{U})\|_2^2 + \lambda \|\hat{\boldsymbol{\xi}}_k\|_1$
   where $\dot{\boldsymbol{X}}_k$ is the $k$th row of $\dot{\boldsymbol{X}}$ and $\boldsymbol{\xi}_k$ is the $k$th row of $\boldsymbol{\Xi}$.
7: Note: One can also use other optimization techniques to solve this problem, such as LASSO.
8: The identified dynamical system is
     $\dot{\boldsymbol{x}} = \boldsymbol{\Xi}\boldsymbol{\Theta}^T(\boldsymbol{x}, \boldsymbol{u})$,
   which means that the dynamics governing the $k$th state variable is
     $\dot{x}_k = \boldsymbol{\Theta}(\boldsymbol{x}, \boldsymbol{u})\boldsymbol{\xi}_k$
```

Since a key component of the SINDYc algorithm is solving the sparse regression problem using a sequentially thresholded least squares (STLS) approach, I have also included the the STLS algorithm outlined by Kaiser et al in [1].

Algorithm 2 Sequentially Thresholded Least Squares (STLS)

Input: Time derivative $\dot{\boldsymbol{X}}$, library of candidate functions $\boldsymbol{\Theta}^T(\boldsymbol{X}, \boldsymbol{U})$, thresholding parameter ϵ
Output: Matrix of sparse coefficient vectors $\boldsymbol{\Xi}$

```
 1: function STLS_REGRESSION($\dot{X}$, $\Theta^T(X, U)$, $\epsilon$, $N$)
 2:     $\hat{\Xi}^0 \leftarrow (\Theta^T)^+ \dot{X}$                              # Initialize least squares guess
 3:     while not converged do
 4:         $k \leftarrow k + 1$
 5:         $I_{small} \leftarrow (\text{abs}(\hat{\Xi}) < \epsilon)$                       # Find small entries ...
 6:         $\hat{\Xi}^k(I_{small}) \leftarrow 0$                                      # ... and threshold
 7:         for all variables do
 8:             $I_{big} \leftarrow \sim I_{small}(:, ii)$                          # Find big entries ...
 9:             $\hat{\Xi}^k(I_{big}, ii) \leftarrow (\Theta^T(:, I_{big}))^+ \dot{X}(:, ii)$   # ... and regress onto those terms
10:         end for
11:     end while
12: end function
```

3 Example: Naive SINDY vs. SINDYc

In this illustrative example, we will see that naively using SINDY does not recover the dynamics of a system with control inputs; whereas, SINDYc will accurately capture the full dynamics. While I was tempted to return to our beloved Lorenz system, I will resist and instead look at another interesting example from [1]—the Lotka-Volterra system. This weakly nonlinear system describes the interactions between two competing populations (i.e. prey and predator populations), which are represented by state variables x_1 and x_2, respectively. The Lotka-Volterra system describing the prey and predator population dynamics is given by

$$\dot{x}_1 = ax_1 - bx_1x_2 \tag{4}$$
$$\dot{x}_2 = -cx_2 + dx_1x_2 + u, \tag{5}$$

where u is the control input and the constant parameters a, b, c and d represent the growth/death rates, the effect of predation on the prey population, and the growth of predators based on the size of the prey population, respectively. In order to be consistent with [1], let these parameters be $a = 0.5$, $b = 0.025$, $c = 0.5$, and $d = 0.005$.

First, we will simulate the system with a periodic control input to obtain training data. Again keeping consistent with [1], I simulated the system using MATLAB's ode45 function with initial conditions $(x_1, x_2)_0 = (60, 50)$, a time span of $t \in [0, 100]$ with time step $dt = 0.01$, and a tolerance of 1e-10, along with the control input defined as $u(t) = (2\sin(t)\sin(t/10))^2$. Next, in order to compare naive SINDY and SINDYc, consider the following two libraries of polynomials up to order 3.

$$\boldsymbol{\Theta}(\boldsymbol{x}) = \begin{bmatrix} 1 & x_1 & x_2 & x_1^2 & x_1x_2 & x_2^2 & x_1^3 & \cdots & x_2^3 \end{bmatrix} \tag{6}$$
$$\boldsymbol{\Theta}(\boldsymbol{x}, u) = \begin{bmatrix} 1 & x_1 & x_2 & u & x_1^2 & x_1x_2 & x_1u & \cdots & u^3 \end{bmatrix} \tag{7}$$

The library defined by (6) corresponds to naive SINDY since it doesn't include any cross terms with u; whereas, the library defined by (7) corresponds to SINDYc and does take into account cross terms with u. Using the naive SINDY library and the SINDYc library with the same training data and the same sparsification parameter $\lambda = 0.001$, the original SINDY algorithm and Algorithm 1 yield a naive SINDY identified model and a SINDYc identified model. I then simulated both of these identified models along with the true model for another 100 time units with the same time step and control input $u(t)$, which is shown in the following plot.

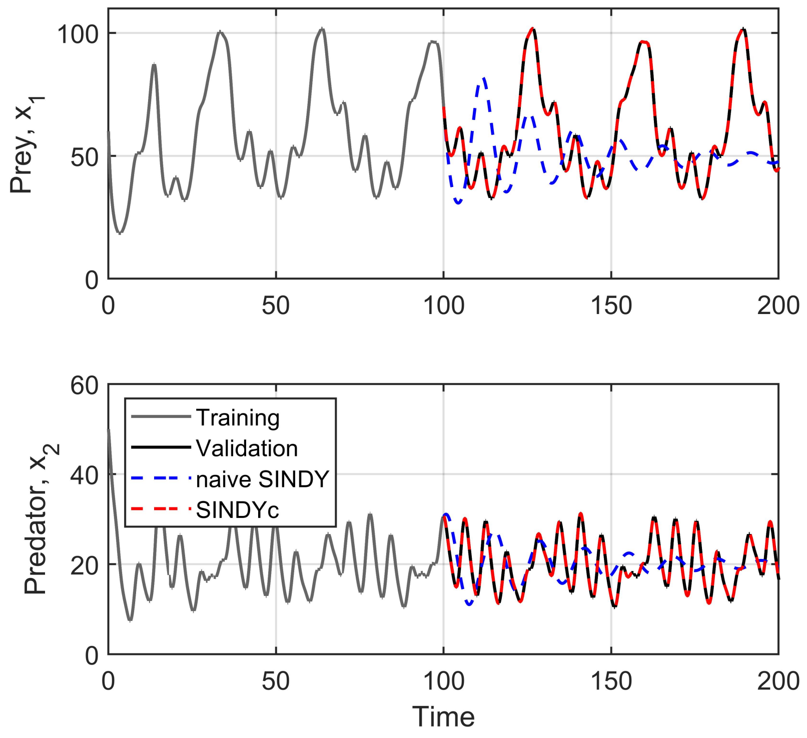

Figure 1: Naive SINDY and SINDYc identified models, which are trained on the interval [0,100] and used for prediction on the interval [100,200]

Clearly, the naive SINDY identified model does not track with the true model well beyond the training period, while the SINDYc model exactly tracks the true model over the entire time span.

4 Model Predictive Control using SINDYc

Model predictive control (MPC) is a popular control technique that utilizes the system model for real-time control by way of solving a constrained optimization problem at each time step to generate an optimal control input u_j. However, a complex system model can make solving the optimization problem at each time step computationally infeasible. Further, all of the familiar issues with finding an appropriate model arise, such as not being able to use first-principles to find an analytic model, needing a huge amount of data to train a neural net, etc. Therefore, the fact that SINDYc is purely data-driven and still yields simple, parsimonious, interpretable models makes it an ideal modelling strategy to use with MPC, which Kaiser et al call SINDY-MPC in [1].

Unfortunately, for the sake of brevity (and since I'm running out of time to turn this in), I won't give a detailed description of MPC or the algorithm for SINDY-MPC in this report. Instead, I direct my reader to Section 2b in [1] for a great overview of MPC and a clear description of how SINDYc is used in conjunction to create SINDY-MPC (that is, if you haven't already read [1]). With that being said, I wouldn't be doing SINDYc justice without at least showing a plot of SINDY-MPC at work. Thus, again consider the prey-predator dynamics described by (6) and (7). The unforced system exhibits cyclical behavior and has a critical point at $(x_1, x_2)_{crit} = (g/d, a/b) = (100, 20)$, which it won't be able to converge to unless there is a stabilizing control input. Using the SINDY-MPC code from Kaiser's github repository that is associated with [1], we can see that SINDY-MPC quickly drives the system to the critical point and stabilizes.

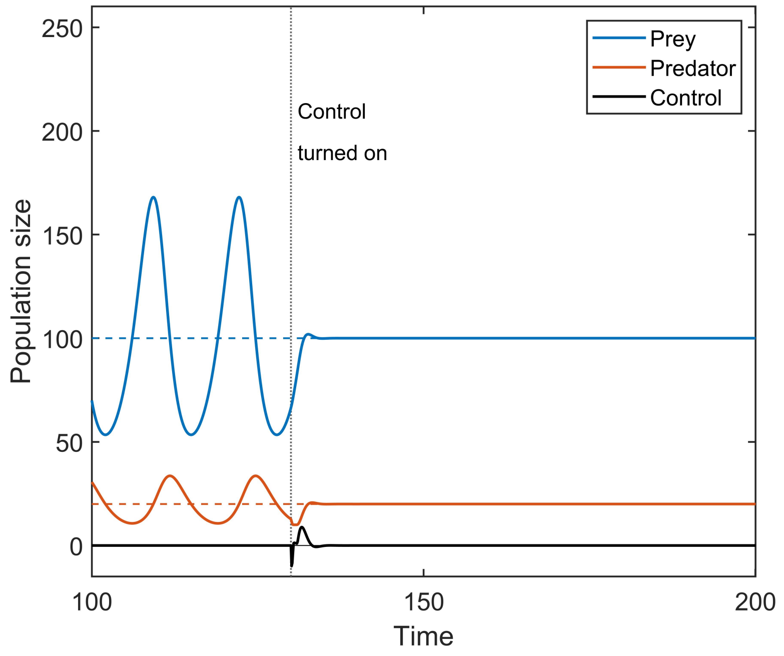

Figure 2: Control inputs generated by SINDY-MPC to drive the system to its critical point of (100,20) and stabilize it there.

5 Conclusion

Although the original SINDY formulation was restricted to unforced dynamical systems, we have seen that it is quite easy and fruitful to extend SINDY to included actuated systems with control inputs. Ultimately, the overarching idea of sparse system identification presents many of the sought after benefits in data-driven methods. To reiterate a few, SINDY (and/or SINDYc) doesn't need knowledge of governing equations, doesn't need much data to train on, yields a simple and interpretable identified model (not a black box), and is relatively robust over a short time period. Also, one of my personal favorite things about SINDY is how easy the general idea is to understand. In a class where we've seen methods that can be quite hard to understand (for me it was transfer entropy), it's refreshing that such an intuitive idea as only needing a few nonlinear functions to approximate a dynamical system works as well as you'd hope it would. I'm looking forward to seeing what new variations and extensions of SINDY and DMD Steve and his crew have next!

6 References

[1] Kaiser, Eurika, J. Nathan Kutz, and Steven L. Brunton. "Sparse identification of nonlinear dynamics for model predictive control in the low-data limit." *Proceedings of the Royal Society A* 474.2219 (2018): 20180335.

[2] Proctor, Joshua L., Steven L. Brunton, and J. Nathan Kutz. "Dynamic mode decomposition with control." *SIAM Journal on Applied Dynamical Systems* 15.1 (2016): 142-161.

[3] Brunton, Steven L., 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 (2016): 3932-3937.