PySINDy: A comprehensive Python package for robust sparse system identification

Alan A. Kaptanoglu1, Brian M. de Silva2, Urban Fasel3, Kadierdan Kaheman1, Andy J. Goldschmidt1, Jared Callaham3, Charles B. Delahunt2, Zachary G. Nicolaou2, Kathleen Champion2, Jean-Christophe Loiseau4, J. Nathan Kutz2, and Steven L. Brunton3

1 Department of Physics, University of Washington 2 Department of Applied Mathematics, University of Washington 3 Department of Mechanical Engineering, University of Washington 4 Arts et Métiers Institute of Technology, CNAM, DynFluid, HESAM Université

Summary

Automated data-driven modeling, the process of directly discovering the governing equations of a system from data, is increasingly being used across the scientific community. PySINDy is a Python package that provides tools for applying the sparse identification of nonlinear dynamics (SINDy) approach to data-driven model discovery. In this major update to PySINDy, we implement several advanced features that enable the discovery of more general differential equations from noisy and limited data. The library of candidate terms is extended for the identification of actuated systems, partial differential equations (PDEs), and implicit differential equations. Robust formulations, including the integral form of SINDy and ensembling techniques, are also implemented to improve performance for real-world data. Finally, we provide a range of new optimization algorithms, including several sparse regression techniques and algorithms to enforce and promote inequality constraints and stability. Together, these updates enable entirely new SINDy model discovery capabilities that have not been reported in the literature, such as constrained PDE identification and ensembling with different sparse regression optimizers.

Statement of need

Traditionally, the governing laws and equations of nature have been derived from first principles and based on rigorous experimentation and expert intuition. In the modern era, cheap and efficient sensors have resulted in an unprecedented growth in the availability of measurement data, opening up the opportunity to perform automated model discovery using data-driven modeling. These data-driven approaches are also increasingly useful for processing and interpreting the information in these large datasets. A number of such approaches have been developed in recent years, including the dynamic mode decomposition (Kutz et al., 2016; Schmid, 2010), Koopman theory (Steven L. Brunton et al., 2021), nonlinear autoregressive algorithms (Billings, 2013), neural networks (Pathak et al., 2018; M. Raissi et al., 2019; Vlachas et al., 2018), Gaussian process regression (Maziar Raissi et al., 2017), operator inference and reduced-order modeling (Benner et al., 2015; Peherstorfer & Willcox, 2016; Qian et al., 2020), genetic programming (Bongard & Lipson, 2007; Schmidt & Lipson, 2009), and sparse regression (Steven L. Brunton et al., 2016). These approaches have seen many variants and improvements over the years, so data-driven modeling software must be regularly updated to remain useful to the scientific community. The SINDy approach has experienced

particularly rapid development, motivating this major update to aggregate these innovations into a single open-source tool that is transparent and easy to use for non-experts or scientists from other fields.

The original PySINDy code (de Silva et al., 2020) provided an implementation of the traditional SINDy method (Steven L. Brunton et al., 2016), which assumes that the dynamical evolution of a state variable \( q(t) \in \mathbb{R}^n \) follows an ODE described by a function \( f \),

\[
\frac{d}{dt} q = f(q). \tag{1}
\]

SINDy approximates the dynamical system \( f \) in Eq. (1) as a sparse combination of terms from a library of candidate basis functions \( \theta(q) = [\theta_1(q), \theta_2(q), \ldots, \theta_p(q)] \)

\[
f(q) \approx \sum_{k=1}^{p} \theta_k(q) \xi_k, \quad \text{or equivalently } \quad \frac{d}{dt} q \approx \Theta(q) \Xi, \tag{2}
\]

where \( \Xi = [\xi_1, \xi_2, \ldots, \xi_p] \) contain the sparse coefficients. In order for this strategy to be successful, a reasonably accurate approximation of \( f(q) \) should exist as a sparse expansion in the span of \( \theta \). Therefore, background scientific knowledge about expected terms in \( f(q) \) can be used to choose the library \( \theta \). To pose SINDy as a regression problem, we assume we have a set of state measurements sampled at time steps \( t_1, \ldots, t_m \) and rearrange the data into the data matrix \( Q \in \mathbb{R}^{m \times n} \),

\[
Q = \begin{bmatrix}
  q_1(t_1) & q_2(t_1) & \cdots & q_n(t_1) \\
  q_1(t_2) & q_2(t_2) & \cdots & q_n(t_2) \\
  \vdots & \vdots & \ddots & \vdots \\
  q_1(t_m) & q_2(t_m) & \cdots & q_n(t_m)
\end{bmatrix}. \tag{3}
\]

A matrix of derivatives in time, \( Q_t \), is defined similarly and can be numerically computed from \( Q \). PySINDy defaults to second order finite differences for computing derivatives, although a host of more sophisticated methods are now available, including arbitrary order finite differences, Savitzky-Golay derivatives (i.e. polynomial-filtered derivatives), spectral derivatives with optional filters, arbitrary order spline derivatives, and total variational derivatives (Ahnert & Åbel, 2007; Chartrand, 2011; Tibshirani & Taylor, 2011).

After \( Q_t \) is obtained, Eq. (2) becomes \( Q_t \approx \Theta(Q) \Xi \) and the goal of the SINDy sparse regression problem is to choose a sparse set of coefficients \( \Xi \) that accurately fits the measured data in \( Q \). We can promote sparsity in the identified coefficients via a sparse regularizer \( R(\Xi) \), such as the \( l_0 \) or \( l_1 \) norm, and use a sparse regression algorithm such as SR3 (Champion et al., 2020) to solve the resulting optimization problem,

\[
\text{argmin}_{\Xi} \| Q_t - \Theta(Q) \Xi \|^2 + R(\Xi). \tag{4}
\]

The original PySINDy package was developed to identify a particular class of systems described by Eq. (1). Recent variants of the SINDy method are available that address systems with control inputs and model predictive control (MPC) (Fasel, Kaiser, et al., 2021; Kaiser et al., 2018), systems with physical constraints (Kaptanoglu, Morgan, et al., 2021; Loiseau & Brunton, 2018), implicit ODEs (Kaheman et al., 2020; Mangan et al., 2016), PDEs (Rudy et al., 2017; Schaeffer, 2017), and weak form ODEs and PDEs (Messenger & Bortz, 2021; Reinbold et al., 2020; Schaeffer & McCalla, 2017). Other methods, such as ensembling and sub-sampling (Delahunt & Kutz, 2021; Maddu et al., 2019; Reinbold et al., 2021), are often vital for making the identification of Eq. (1) more robust. In order to incorporate these new developments and accommodate the wide variety of possible dynamical systems, we have extended PySINDy to a more general setting and added significant new functionality. Our code\(^1\) is thoroughly documented, contains extensive examples, and integrates a wide range of

\(^1\)https://github.com/dynamicslab/pysindy
functionality, some of which may be found in a number of other local SINDy implementations\(^2\). In contrast to some of these existing codes, PySINDy is completely open-source, professionally-maintained (for instance, providing unit tests and adhering to PEP8 stylistic standards), and minimally dependent on non-standard Python packages.

**New features**

Given spatiotemporal data \(Q(x, t) \in \mathbb{R}^{m \times n}\), and optional control inputs \(u \in \mathbb{R}^{m \times r}\) (note \(m\) has been redefined here to be the product of the number of spatial measurements and the number of time samples), PySINDy can now approximate algebraic systems of PDEs (and corresponding weak forms) in an arbitrary number of spatial dimensions. Assuming the system is described by a function \(g\), we have

\[
g(q, q_t, q_x, q_y, q_{xx}, \ldots, u) = 0. \tag{5}
\]

ODEs, implicit ODEs, PDEs, and other dynamical systems are subsets of Eq. (5). We can accommodate control terms and partial derivatives in the SINDy library by adding them as columns in \(\Theta(Q)\), which becomes \(\Theta(Q, Q_t, Q_x, \ldots, u)\).

In addition, we have extended PySINDy to handle more complex modeling scenarios, including trapping SINDy for provably stable ODE models for fluids (Kaptanoglu, Callaham, et al., 2021), models trained using multiple dynamic trajectories, and the generation of many models with sub-sampling and ensemble methods (Fasel, Kutz, et al., 2021) for cross-validation and probabilistic system identification. In order to solve Eq. (5), PySINDy implements several different sparse regression algorithms. Greedy sparse regression algorithms, including step-wise sparse regression (SSR) (Boninsegna et al., 2018) and forward regression orthogonal least squares (FROLS) (Billings, 2013), are now available. For maximally versatile candidate libraries, the new GeneralizedLibrary class allows for tensoring, concatenating, and otherwise combining many different candidate libraries, along with optionally specifying a subset of the inputs to use for generating each of the libraries. Figure 1 illustrates the PySINDy code structure, changes, and high-level goals for future work, and YouTube tutorials for this new functionality are available online.

PySINDy includes extensive Jupyter notebook tutorials that demonstrate the usage of various features of the package and reproduce nearly the entirety of the examples from the original SINDy paper (Steven L. Brunton et al., 2016), trapping SINDy paper (Kaptanoglu, Callaham, et al., 2021), and the PDE-FIND paper (Rudy et al., 2017). We include an extended example for the quasiperiodic shear-driven cavity flow (Callaham et al., 2021). As a simple illustration of the new functionality, we demonstrate how SINDy can be used to identify the Kuramoto-Sivashinsky (KS) PDE from data. We train the model on the first 60% of the data from Rudy et al. (Rudy et al., 2017), which in total contains 1024 spatial grid points and 251 time steps. The KS model is identified correctly and the prediction for \(\dot{q}\) on the remaining testing data indicates strong performance in Figure 2. Lastly, we provide a useful flow chart in Figure 3 so that users can make informed choices about which advanced methods are suitable for their datasets.

---


Conclusion

The goal of the PySINDy package is to enable anyone with access to measurement data to engage in scientific model discovery. The package is designed to be accessible to inexperienced users, adhere to scikit-learn standards, include most of the existing SINDy variations in the literature, and provide a large variety of functionality for more advanced users. We hope that researchers will use and contribute to the code in the future, pushing the boundaries of what is possible in system identification.

Acknowledgments

PySINDy is a fork of sparsereg (Quade, 2018). SLB, AAK, KK, and UF acknowledge support from the Army Research Office (ARO W911NF-19-1-0045). JLC acknowledges support from funding support from the Department of Defense (DoD) through the National Defense Science & Engineering Graduate (NDSEG) Fellowship Program. ZGN is a Washington Research Foundation Postdoctoral Fellow.

Figure 1: Summary of SINDy features organized by (a) PySINDy structure and (b) functionality. (a) Hierarchy from the sparse regression problem solved by SINDy, to the submodules of PySINDy, to the individual optimizers, libraries, and differentiation methods implemented in the code. (b) Flow chart for organizing the SINDy variants and functionality in the literature. Bright color boxes indicate the features that have been implemented through this work, roughly organized by functionality. Semi-transparent boxes indicate features that have not yet been implemented.
Figure 2: PySINDy can now be used for PDE identification; we illustrate this new capability by accurately capturing a set of testing data from the Kuramoto-Sivashinsky system, described by 
\[ q_t = -qq_x - q_{xx} - q_{xxxx}. \]
The identified model is 
\[ q_t = -0.98q_x - 0.99q_{xx} - 1.0q_{xxxx}. \]

Figure 3: This flow chart summarizes how PySINDy users can start with a dataset and systematically choose the proper candidate library and sparse regression optimizer that are tailored for a specific scientific task.

References


