PySINDy: A Python package for the Sparse Identification of Nonlinear Dynamics from Data

Brian M. de Silva[1*], Kathleen Champion[1*], Markus Quade[2],
Jean-Christophe Loiseau[3], J. Nathan Kutz[1], Steven L. Brunton[4]
[1] Department of Applied Mathematics, University of Washington, Seattle, WA 98195, United States
[2] Ambrosys GmbH, Potsdam, Germany
[3] ParisTech, Paris, France
[4] Department of Mechanical Engineering, University of Washington, Seattle, WA 98195, United States

Abstract

`PySINDy` is a Python package for the discovery of governing dynamical systems models from data. In particular, `PySINDy` provides tools for applying the sparse identification of nonlinear dynamics (SINDy) [1] approach to model discovery. In this work we provide a brief description of the mathematical underpinnings of SINDy, an overview and demonstration of the features implemented in `PySINDy` (with code examples), practical advice for users, and a list of potential extensions to `PySINDy`. Software is available at `https://github.com/dynamicslab/pysindy`.

Keywords– system identification, dynamical systems, symbolic regression, open source, python

1 Introduction

Scientists have long quantified empirical observations by developing mathematical models that characterize the observations, have some measure of interpretability, and are capable of making predictions. Dynamical systems models, in particular, have been widely used to study, explain, and predict behavior in a diversity of application areas, with examples ranging from Newton's laws of classical mechanics to the Michaelis-Menten kinetics for modeling enzyme kinetics. While governing laws and equations have traditionally been derived from first principles and expert knowledge, the current growth of available measurement data and the resulting emphasis on data-driven modeling motivates algorithmic and reproducible approaches for automated model discovery.

A number of such approaches have been developed in recent years [2], including linear methods [3, 4], dynamic mode decomposition (DMD) [5, 6] and Koopman theory more generally [7–12], nonlinear autoregressive algorithms [13, 14], neural networks [15–24], Gaussian process regression [25, 26], operator inference and reduced-order modeling [27–29], nonlinear Laplacian spectral analysis [30], diffusion maps [31], genetic programming [32–34], and sparse regression [1, 35]. Maximizing the impact of these model discovery methods requires tools to make them widely accessible to scientists across domains and at various levels of mathematical expertise.

`PySINDy` is a Python package for the discovery of governing dynamical systems models from data. In particular, `PySINDy` provides tools for applying the sparse identification of nonlinear dynamics (SINDy) approach to model discovery [1]. SINDy poses model discovery as a sparse regression problem, where relevant terms in the dynamics are selected from a library of candidate functions, many of them motivated by our deep historical knowledge of diverse physics models. This approach results in interpretable models, and it has been widely applied [36–50] and extended [35, 37, 51–63] using different sparse optimization algorithms and library functions.

The `PySINDy` package is aimed at researchers and practitioners alike, enabling anyone with access to measurement data to engage in scientific model discovery. The package is designed to be accessible to inexperienced users, adhering to `scikit-learn` standards, while also including customizable options for more advanced users. A number of popular SINDy variants are implemented, but `PySINDy` is also designed to enable further extensions for research and experimentation.

* Corresponding authors (bdesilva@uw.edu and kpchamp@uw.edu).

2 Background

`PySINDy` provides an implementation of the SINDy method to discover governing dynamical systems models of the form

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

Given data in the form of state measurements $\mathbf{x}(t) \in \mathbb{R}^n$, SINDy identifies a model for the dynamics, given by the function $\mathbf{f}$, which describes how the state of the system evolves in time. In particular, SINDy sparsely approximates the dynamics in a library of candidate basis functions $\boldsymbol{\theta}(\mathbf{x}) = [\theta_1(\mathbf{x}), \theta_2(\mathbf{x}), \ldots, \theta_\ell(\mathbf{x})]$, so that

$$\mathbf{f}(\mathbf{x}) \approx \sum_{k=1}^{\ell} \theta_k(\mathbf{x})\xi_k.$$

The majority of coefficients ξ_k are zero, and nonzero entries identify active terms in the dynamics.

To pose SINDy as a regression problem, time-series measurements of $\mathbf{x}$ and their time derivatives $\dot{\mathbf{x}}$ are arranged into matrices

$$\mathbf{X} = \begin{pmatrix} x_1(t_1) & x_2(t_1) & \cdots & x_n(t_1) \\ x_1(t_2) & x_2(t_2) & \cdots & x_n(t_2) \\ \vdots & \vdots & \ddots & \vdots \\ x_1(t_m) & x_2(t_m) & \cdots & x_n(t_m) \end{pmatrix}, \quad \dot{\mathbf{X}} = \begin{pmatrix} \dot{x}_1(t_1) & \dot{x}_2(t_1) & \cdots & \dot{x}_n(t_1) \\ \dot{x}_1(t_2) & \dot{x}_2(t_2) & \cdots & \dot{x}_n(t_2) \\ \vdots & \vdots & \ddots & \vdots \\ \dot{x}_1(t_m) & \dot{x}_2(t_m) & \cdots & \dot{x}_n(t_m) \end{pmatrix}.$$

The derivatives can be approximated numerically or measured directly. The library functions are evaluated on the data, resulting in $\mathbf{\Theta}(\mathbf{X}) = [\theta_1(\mathbf{X}), \theta_2(\mathbf{X}), \ldots, \theta_\ell(\mathbf{X})]$. Sparse regression is then performed to approximately solve

$$\dot{\mathbf{X}} \approx \mathbf{\Theta}(\mathbf{X})\mathbf{\Xi}, \tag{2}$$

where $\mathbf{\Xi}$ is a set of coefficients that determines the active terms in $\mathbf{f}$. While the original SINDy formulation solves (2) using a sequentially thresholded least squares algorithm [1, 64], this problem can be solved using any sparse regression algorithm, such as lasso [65], sparse relaxed regularized regression (SR3) [66, 67], stepwise sparse regression (SSR) [58], or Bayesian methods [68–70].

SINDy has been widely applied for model identification in applications such as chemical reaction dynamics [40], nonlinear optics [36], fluid dynamics [37, 39, 41, 42, 47] and turbulence modeling [48, 50], plasma convection [38], numerical algorithms [45], and structural modeling [46], among others [43, 44, 49]. It has also been extended to handle more complex modeling scenarios such as partial differential equations [35, 52], systems with inputs or control [59], systems with implicit dynamics [51], hybrid systems [60], to enforce physical constraints [37], to incorporate information theory [53], to identify models from corrupt or limited data [54, 56] and ensembles of initial conditions [57], and extending the formulation to include integral terms [55, 63], tensor representations [61, 62], and stochastic forcing [58]. However, there is not a definitive standard implementation or package for SINDy. Versions of SINDy have been implemented within larger projects such as `sparsereg` [71], but no specific implementation has emerged as the most widely adopted and most versions implement a limited set of features. Researchers have thus typically written their own implementations, resulting in duplicated effort and a lack of standardization. This not only makes it more difficult to apply SINDy to scientific data sets, but also makes it more challenging to benchmark extensions of the method and makes such extensions less accessible to end users. This motivates the creation of a dedicated package for SINDy. The `PySINDy` package provides a central codebase where many of the basic SINDy features are implemented, allowing for easy use and standardization. In addition, it is straightforward for users to extend `PySINDy` so that new developments are available to the wider community.

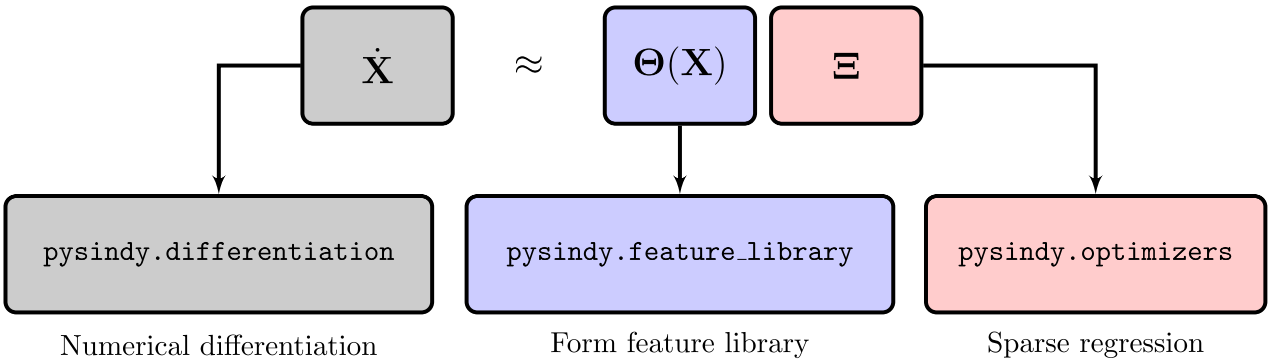

Figure 1: Correspondence between the sparse regression problem solved by SINDy and the submodules of PySINDy.

3 Features

The core object in the `PySINDy` package is the `SINDy` model class, which is implemented as a `scikit-learn` estimator. This design choice was made to ensure that the package is simple to use for a wide user base, as many potential users will be familiar with `scikit-learn`. It also expresses the `SINDy` model object at the appropriate level of abstraction so that users can embed it into more sophisticated pipelines in `scikit-learn`, such as for parameter tuning and model selection.

Our `PySINDy` implementation involves three major steps, resulting in three modeling decisions:

1. The *numerical differentiation* scheme used to approximate $\dot{\mathbf{X}}$ from $\mathbf{X}$;
2. The candidate functions constituting the *feature library* $\mathbf{\Theta}$;
3. The *sparse regression* algorithm that is applied to solve (2) to find $\mathbf{\Xi}$.

The core `SINDy` object was designed to incorporate these three components in as modular a manner as possible, having one attribute corresponding to each: `SINDy.differentiation_method` for numerical differentiation, `SINDy.feature_library` for the formation of the candidate function library, and `SINDy.optimizer` for the sparse regressor. `PySINDy` provides standard options for each step, while making it easy to replace any of these steps with more sophisticated "third-party" algorithms. In particular, at the time of writing, we have implemented the following methods:

- Numerical differentiation (for computing $\dot{\mathbf{X}}$ from $\mathbf{X}$)
  - Finite difference: `FiniteDifference`
  - Smoothed finite difference: `SmoothedFiniteDifference`
- Feature library (for constructing $\mathbf{\Theta}$)
  - Multivariate polynomials: `PolynomialLibrary`
  - Fourier modes (i.e. trigonometric functions): `FourierLibrary`
  - Custom library (defined by user-supplied functions): `CustomLibrary`
  - Identity library (in case users want to compute $\mathbf{\Theta}$ themselves): `IdentityLibrary`
- Optimizer (for performing sparse regression)
  - Sequentially thresholded least-squares [1, 64]: `STLSQ`
  - Sparse relaxed regularized regression (SR3) [66]: `SR3`

4 Examples

The `PySINDy` GitHub page[1] includes tutorials in the form of Jupyter notebooks. These tutorials demonstrate the usage of various features of the package and reproduce the examples from the original SINDy paper [1]. Throughout this section we will use the Lorenz equations (3) as the dynamical system to illustrate the `PySINDy` package:

$$\begin{cases} \dot{x} = -10x - 10y \\ \dot{y} = x(28 - z) - y \\ \dot{z} = xy - \dfrac{8}{3}z \end{cases} \tag{3}$$

In Python, the right-hand side of (3) can be expressed as follows:

```
def lorenz(x, t):
    return [
        10 * (x[1] - x[0]),
        x[0] * (28 - x[2]) - x[1],
        x[0] * x[1] - (8 / 3) * x[2]
    ]
```

To construct training data to feed into a SINDy model, we integrate (3) with:

```
import numpy as np
from scipy.integrate import odeint

dt = 0.002
t = np.arange(0, 10, dt)
x0 = [-8, 8, 27]
X = odeint(lorenz, x0, t)
```

We plot `X` in Figure 2. It is important to note that each column of `X` corresponds to a variable and each row to a point in time. All `PySINDy` objects that handle data assume the data is structured this way.

4.1 Basic usage

The `pysindy` package is built around the `SINDy` class, which encapsulates all the steps necessary to learn a dynamical system with SINDy. To create a SINDy object, fit it to the data (i.e. to infer a dynamical system from the data), and print the resulting model, we invoke the `SINDy` constructor, the `fit` method, and custom `print` functions

```
model = ps.SINDy()
model.fit(X, t=dt)
model.print()
```

which generates the following output

```
x0' = -9.999 x0 + 9.999 x1
x1' = 27.992 x0 + -0.999 x1 + -1.000 x0 x2
x2' = -2.666 x2 + 1.000 x0 x1
```

[1] `https://github.com/dynamicslab/pysindy`

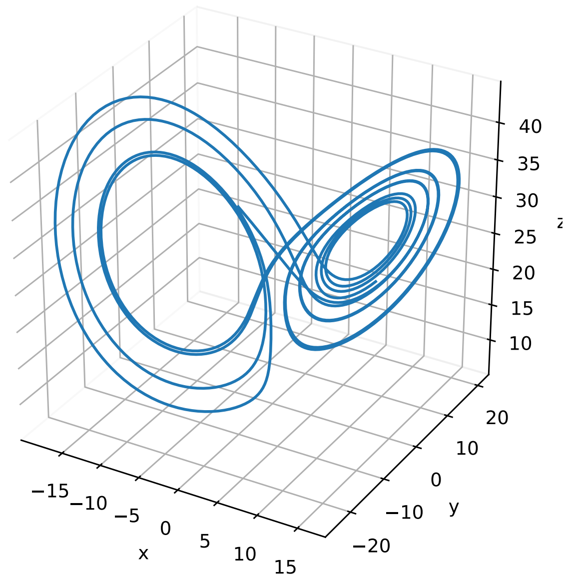

Figure 2: Measurement data simulated using the Lorenz equations (3).

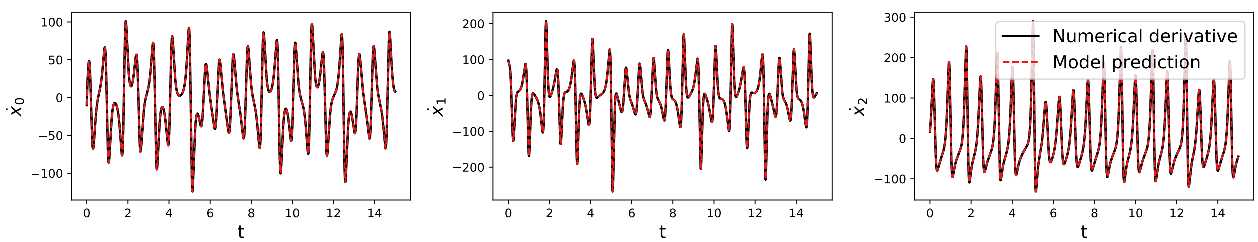

Figure 3: Derivatives of variables from the Lorenz equation via numerical differentiation and using a learned SINDy model.

Once the SINDy object has been fit we can feed in new data and use the learned model to predict the derivatives for each measurement (recall that measurements correspond to rows).

```
t_test = np.arange(0, 15, dt)
x0_test = np.array([8, 7, 15])
X_test = odeint(lorenz, x0_test, t_test)

X_dot_test_computed = model.differentiate(X_test, t=dt)
X_dot_test_predicted = model.predict(X_test)
```

The call `model.differentiate(X_test, t=dt)` applies the numerical differentiation method in the `SINDy` model to `X_test` with time steps of length `dt`. In Figure 3 we plot each dimension of `X_dot_test_computed` and `X_dot_test_predicted`.

Rather than predicting derivatives, we will often be interested in using our model to evolve initial conditions forward in time using the learned model. The `simulate` function does just that.

```
X_test_sim = model.simulate(x0_test, t_test)
```

Figure 4 shows the simulated trajectory plotted against the true trajectory `X_test`. The trajectories agree initially, but they eventually diverge due to the chaotic nature of the Lorenz equations.

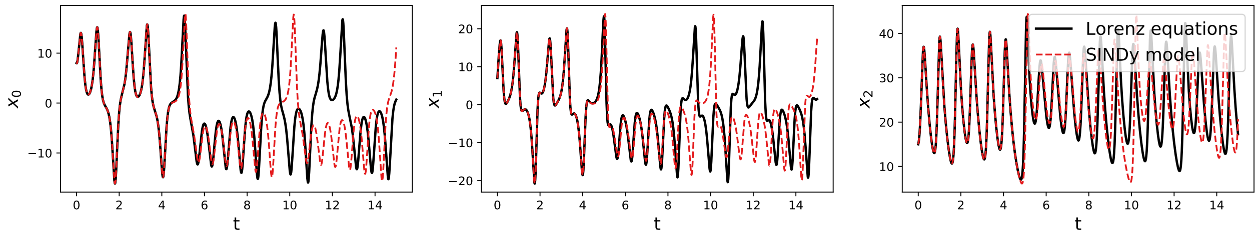

Figure 4: Two trajectories starting at the same position evolved forward in time with the exact Lorenz equations (black, solid) and the learned SINDy model (red, dashed).

4.2 Custom features

Thus far we have relied on the default options of the `SINDy` object, but `PySINDy` comes equipped with multiple alternative built-in methods for differentiation, library building, and optimization. These options are selected by passing corresponding `PySINDy` objects to the `SINDy` constructor via the `differentiation_method`, `feature_library`, and `optimizer` arguments, respectively. Parameters for the differentiation, library, and optimization algorithms are supplied to the corresponding objects' constructors rather than directly to the `SINDy` object. We demonstrate the syntax with the following example.

```
differentiation_method = ps.FiniteDifference(order=1)
feature_library = ps.PolynomialLibrary(degree=3, include_bias=False)
optimizer = ps.SR3(threshold=0.1, nu=1, tol=1e-6)

model = ps.SINDy(
    differentiation_method=differentiation_method,
    feature_library=feature_library,
    optimizer=optimizer,
    feature_names=["x", "y", "z"]
)

model.fit(X, t=dt)
model.print()
```

which prints

```
x' = -10.021 x + 9.993 y
y' = 28.431 x + -1.212 y + -1.008 x z
z' = -2.675 z + 1.000 x y.
```

A number of other built-in options are available. The official documentation[2] and examples[3] provide an exhaustive list.

[2] `https://pysindy.readthedocs.io/en/latest/index.html`
[3] `https://github.com/dynamicslab/pysindy/tree/master/example`

5 Practical tips

In this section we provide pragmatic advice for using `PySINDy` effectively. We discuss potential pitfalls and strategies for overcoming them. We also specify how to incorporate custom methods not implemented natively in `PySINDy`, where applicable. The information presented here is derived from a combination of experience and theoretical considerations.

5.1 Numerical differentiation

Numerical differentiation is one of the core components of the SINDy method. Derivatives of measurement variables provide the targets for the sparse regression problem (2). If care is not taken in computing these derivatives, the quality of the learned model is likely to suffer.

By default, a second order finite difference method is used to differentiate input data. Finite difference methods tend to amplify noise in data. If the data are smooth (at least twice differentiable), then finite difference methods give accurate derivative approximations. When the data are noisy, they give derivative estimates with *more* noise than the original data. Figure 5 visualizes the impact of noise on numerical derivatives. Note that even a small amount of noise in the data can produce noticeable degradation in the quality of the numerical derivative.

One way to mitigate the effects of noise is to smooth the measurements before computing derivatives. The `SmoothedFiniteDifference` method can be used for this purpose. A numerical differentiation scheme with total variation regularization has also been proposed [72] and recommended for use in SINDy [1].

Users wishing to employ their own numerical differentiation schemes have two ways of doing so. Derivatives of input measurements can be computed externally with the method of choice and then passed directly into the `SINDy.fit` method via the `x_dot` keyword argument. Alternatively, users can implement their own differentiation methods and pass them into the `SINDy` constructor using the `differentiation_method` argument. In this case, the supplied class need only have implemented a `__call__` method taking two arguments, `x` and `t`.

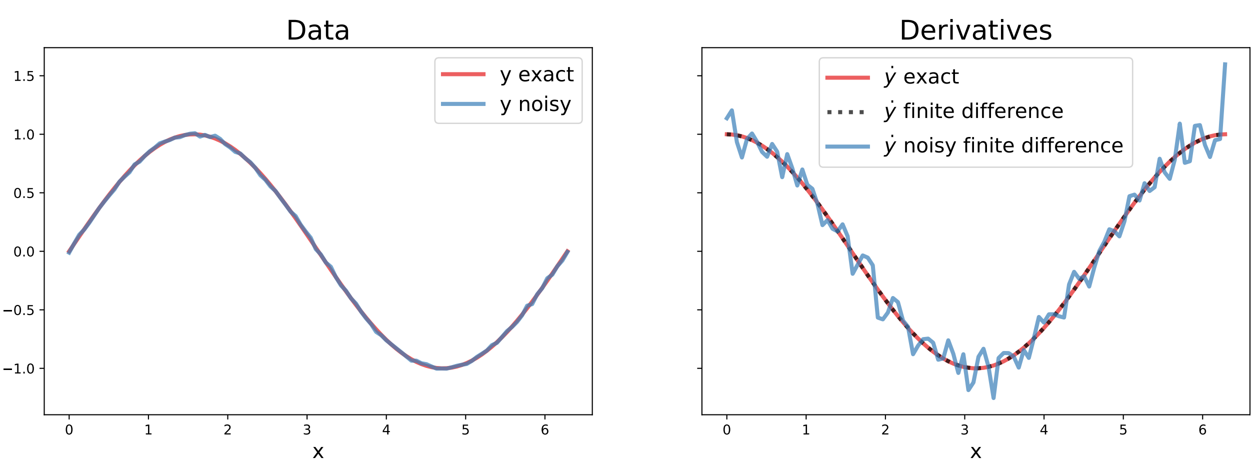

Figure 5: A toy example illustrating the effect of noise on derivatives computed with a second order finite difference method. Left: The data to be differentiated; $y = \sin(x)$ with and without a small amount of additive noise (normally distributed with mean 0 and standard deviation 0.01). Right: Derivatives of the data; the exact derivative $\cos(x)$ (blue), the finite difference derivative of the exact data (black, dashed), and the finite difference derivative of the noisy data.

5.2 Library selection

The SINDy method assumes dynamics can be represented as a *sparse* linear combination of library functions. If this assumption is violated, the method is likely to exhibit poor performance. This issue tends to manifest itself as numerous library terms being active, often with weights of vastly different magnitudes, still resulting in poor model error.

Typically, prior knowledge of the system of interest and its dynamics should be used to make a judicious choice of basis functions. When such information is unavailable, the default class of library functions, polynomials, are a good place to start, as smooth functions have rapidly converging Taylor series. Brunton et al. [1] showed that, equipped with a polynomial library, SINDy can recover the first few terms of the (zero-centered) Taylor series of the true right-hand side function $\mathbf{f}(x)$. If one has reason to believe the dynamics can be sparsely represented in terms of Chebyshev polynomials rather than monomials, then the library should include Chebyshev polynomials.

`PySINDy` includes the `CustomLibrary` and `IdentityLibrary` objects to allow for flexibility in the library functions. When the desired library consists of a set of functions that should be applied to each measurement variable (or pair, triplet, etc. of measurement variables) in turn, the `CustomLibrary` class should be used. The `IdentityLibrary` class is the most customizable, but transfers the work of computing library functions over to the user. It expects that all the features one wishes to include in the library have already been computed and are present in `X` before `SINDy.fit` is called, as it simply applies the identity map to each variable that is passed to it. It is best suited for situations in which one has very specific instructions for how to apply library functions (e.g. if some of the functions should be applied to only some of the input variables).

As terms are added to the library, the underlying sparse regression problem becomes increasingly ill-conditioned. Therefore it is recommended to start with a small library whose size is gradually expanded until the desired level of performance is achieved. For example, a user may wish to start with a library of linear terms and then add quadratic and cubic terms as necessary to improve model performance. For the best results, the strength of regularization applied should be increased in proportion to the size of the library to account for the worsening condition number of the resulting linear system.

Users may also choose to implement library classes tailored to their applications. To do so one should have the new class inherit from our `BaseFeatureLibrary` class. See the documentation for guidance on which functions the new class is expected to implement.

5.3 Optimization

`PySINDy` uses various optimizers to solve the sparse regression problem. For a fixed differentiation method, set of inputs, and candidate library, there is still some variance in the dynamical system identified by SINDY, depending on which optimizer is employed.

The default optimizer in `PySINDy` is the sequentially-thresholded least-squares algorithm (`STLSQ`). In addition to being the method originally proposed for use with SINDy, it involves a single, easily interpretable hyperparameter, and it exhibits good performance across a variety of problems.

The sparse relaxed regularized regression (`SR3`) [66, 67] algorithm can be used when the results of `STLSQ` are unsatisfactory. It involves a few more hyperparameters that can be tuned for improved accuracy. In particular, the `thresholder` parameter controls the type of regularization that is applied. For optimal results, one may find it useful to experiment with L^0, L^1, and clipped absolute deviation (CAD) regularization. The other hyperparameters can be tuned with cross-validation.

Custom or third party sparse regression methods are also supported. Simply instantiate an instance of the custom object and pass it to the `SINDy` constructor using the `optimizer` keyword. Our implementation is compatible with any of the linear models from Scikit-learn (e.g. `RidgeRegression`, `Lasso`, and `ElasticNet`). See the documentation for a list of methods and attributes a custom optimizer is expected to implement. There you will also find an example where the Scikit-learn `Lasso` object is used to perform sparse regression.

5.4 Regularization

Regularization, in this context, is a technique for improving the conditioning of ill-posed problems. Without regularization, one often obtains highly unstable results, with learned parameter values differing substantially for slightly different inputs. SINDy seeks weights that express dynamics as a *sparse* linear combination of library functions. When the columns of the measurement data or the library are statistically correlated, which is likely for large libraries, the SINDy inverse problem can quickly become ill-posed. Though the sparsity constraint is a type of regularization itself, for many problems another form of regularization is needed for SINDy to learn a robust dynamical model.

In some cases regularization can be interpreted as enforcing a prior distribution on the model parameters [73]. Applying strong regularization biases the learned weights *away* from the values that would allow them to best fit the data and *toward* the values preferred by the prior distribution (e.g. L^2 regularization corresponds to a Gaussian prior). Therefore once a sparse set of nonzero coefficients is discovered, our methods apply an extra "unbiasing" step where *unregularized* least-squares is used to find the values of the identified nonzero coefficients. All of our built-in methods use regularization by default.

Some general best practices regarding regularization follow. Most problems will benefit from some amount of regularization. Regularization strength should be increased as the size of the candidate right-hand side library grows. If warnings about ill-conditioned matrices are generated when `SINDy.fit` is called, more regularization may help. We also recommend setting `unbias` to `True` when invoking the `SINDy.fit` method, especially when large amounts of regularization are being applied. Cross-validation can be used to select appropriate regularization parameters for a given problem.

6 Extensions

In this section we list potential extensions and enhancements to our SINDy implementation. We provide references for the improvements that are inspired by previously conducted research and the rationale behind the other potential changes.

- **Partial differential equations (PDEs):** While dynamical systems given by ordinary differential equations (ODEs) provide a flexible approach to modeling physical systems, many systems are inherently described by partial differential equations (PDEs), which are not immediately discoverable using `PySINDy`. Multiple approaches for the data-driven discovery of PDEs have been proposed [35, 52] as extensions to the SINDy method, and these may be readily included within the `PySINDy` framework.

- **Identifying coordinates and latent variables:** Many complex systems, such as fluid flows, are high-dimensional, yet exhibit low-dimensional patterns that may be exploited for modeling [74–76]. Identifying effective coordinate systems on which to build models is an important aspect of data-driven discovery. Recently, SINDy has been embedded into an autoencoder framework [23] to simultaneously identify effective coordinates and sparse dynamics. Similarly, for many systems, it is impossible to measure the full state of the system, so that there are latent variables. Time-delay coordinates have been useful for identifying sparse models from limited measurements [12]. Both of these are candidates for future extensions.

- **Constraints:** SINDy has been extended to enforce *physical constraints* during the sparse regression step [37]. When working with physical systems with known conserved quantities, such as the conservation of energy in incompressible fluids [37], such a method allows one to automatically incorporate this prior information into the model discovery process.

- **Integral formulation:** We previously discussed how measurement data with too much noise can disrupt the model discovery process, and we offered smoothing as one possible solution. Another is to work with an integral version of (1), as proposed by Schaeffer and McCalla [55] and extended to PDEs by Reinbold, Gurevich, and Grigoriev [63]. Where numerical differentiation tends to amplify noise, numerical integration tends to smooth it out. This formulation has been shown to improve the robustness of SINDy to noise.

- **Ensembles:** Ensembles are a proven class of methods in machine learning for variance reduction (improved model generalizability) at the cost of extra computation. Rather than training a single model, multiple high-variance models are trained and their predictions are averaged together. We think that ideas from ensemble learning could be adapted to improve the performance of SINDy models. Recent work has hinted at possible approaches [77].

- **Extended libraries:** Choosing the appropriate basis in which to represent dynamics is of critical importance for the successful application of SINDy. Although we currently provide methods allowing users the flexibility to input their own library functions, we aim to make the library construction process even easier by providing a common suite of tools for the creation and combination of sets of candidate functions. More basis functions could be supported natively, such as nonautonomous terms (those depending explicitly on the dependent variable, time). Taking this idea a step further, specific variables (columns of $\mathbf{X}$) which should not appear on the left-hand side of (2) could be identified by the user. This would enable SINDy to include inputs and control variables [59]. Operations acting on one or more libraries could also be implemented. For example, combining libraries via union, intersection, composition, or tensor product could enable the expression of complicated nonlinear dynamics. The ability to apply libraries to only subsets of state variables could help cut down on the computational cost and improve the conditioning of the sparse regression problem solved within SINDy.

7 Acknowledgments

`PySINDy` is a fork of `sparsereg` [71]. SLB acknowledges funding support from the Army Research Office (ARO W911NF-19-1-0045) and the Air Force Office of Scientific Research (AFOSR FA9550-18-1-0200). JNK acknowledges support from the Air Force Office of Scientific Research (AFOSR FA9550-17-1-0329). This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant Number DGE- 1256082.

References

[1] S. L. Brunton, J. L. Proctor, and J. N. Kutz, "Discovering governing equations from data by sparse identification of nonlinear dynamical systems," *Proceedings of the National Academy of Sciences*, vol. 113, no. 15, pp. 3932–3937, 2016.

[2] S. L. Brunton and J. N. Kutz, *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press, 2019.

[3] O. Nelles, *Nonlinear system identification: from classical approaches to neural networks and fuzzy models*. Springer, 2013.

[4] L. Ljung, "Perspectives on system identification," *Annual Reviews in Control*, vol. 34, no. 1, pp. 1–12, 2010.

[5] P. J. Schmid, "Dynamic mode decomposition of numerical and experimental data," *Journal of fluid mechanics*, vol. 656, pp. 5–28, 2010.

[6] J. N. Kutz, S. L. Brunton, B. W. Brunton, and J. L. Proctor, *Dynamic Mode Decomposition: Data-Driven Modeling of Complex Systems*. SIAM, 2016.

[7] M. Budišić, R. Mohr, and I. Mezić, "Applied Koopmanism a)," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 22, no. 4, p. 047510, 2012.

[8] I. Mezic, "Analysis of fluid flows via spectral properties of the Koopman operator," *Annual Review of Fluid Mechanics*, vol. 45, pp. 357–378, 2013.

[9] M. O. Williams, I. G. Kevrekidis, and C. W. Rowley, "A data-driven approximation of the Koopman operator: extending dynamic mode decomposition," *Journal of Nonlinear Science*, vol. 6, pp. 1307–1346, 2015.

[10] S. Klus, F. Nüske, P. Koltai, H. Wu, I. Kevrekidis, C. Schütte, and F. Noé, "Data-driven model reduction and transfer operator approximation," *Journal of Nonlinear Science*, 2018.

[11] Q. Li, F. Dietrich, E. M. Bollt, and I. G. Kevrekidis, "Extended dynamic mode decomposition with dictionary learning: A data-driven adaptive spectral decomposition of the koopman operator," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 27, no. 10, p. 103111, 2017.

[12] S. L. Brunton, B. W. Brunton, J. L. Proctor, E. Kaiser, and J. N. Kutz, "Chaos as an intermittently forced linear system," *Nature Communications*, vol. 8, no. 19, pp. 1–9, 2017.

[13] H. Akaike, "Fitting autoregressive models for prediction," *Ann Inst Stat Math*, vol. 21, no. 1, pp. 243–247, 1969.

[14] S. A. Billings, *Nonlinear system identification: NARMAX methods in the time, frequency, and spatio-temporal domains*. John Wiley & Sons, 2013.

[15] Z. Long, Y. Lu, X. Ma, and B. Dong, "Pde-net: Learning pdes from data," *arXiv preprint arXiv:1710.09668*, 2017.

[16] L. Yang, D. Zhang, and G. E. Karniadakis, "Physics-informed generative adversarial networks for stochastic differential equations," *arXiv preprint arXiv:1811.02033*, 2018.

[17] C. Wehmeyer and F. Noé, "Time-lagged autoencoders: Deep learning of slow collective variables for molecular kinetics," *The Journal of Chemical Physics*, vol. 148, no. 241703, pp. 1–9, 2018.

[18] A. Mardt, L. Pasquali, H. Wu, and F. Noé, "VAMPnets: Deep learning of molecular kinetics," *Nature Communications*, vol. 9, no. 5, 2018.

[19] P. R. Vlachas, W. Byeon, Z. Y. Wan, T. P. Sapsis, and P. Koumoutsakos, "Data-driven forecasting of high-dimensional chaotic systems with long short-term memory networks," *Proc. R. Soc. A*, vol. 474, no. 2213, p. 20170844, 2018.

[20] J. Pathak, B. Hunt, M. Girvan, Z. Lu, and E. Ott, "Model-free prediction of large spatiotemporally chaotic systems from data: a reservoir computing approach," *Physical review letters*, vol. 120, no. 2, p. 024102, 2018.

[21] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis, "Deepxde: A deep learning library for solving differ-

ential equations," *arXiv preprint arXiv:1907.04502*, 2019.

[22] M. Raissi, P. Perdikaris, and G. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational Physics*, vol. 378, pp. 686–707, 2019.

[23] K. Champion, B. Lusch, J. N. Kutz, and S. L. Brunton, "Data-driven discovery of coordinates and governing equations," *Proceedings of the National Academy of Sciences*, vol. 116, no. 45, pp. 22 445–22 451, 2019.

[24] M. Raissi, A. Yazdani, and G. E. Karniadakis, "Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations," *Science*, vol. 367, no. 6481, pp. 1026–1030, 2020.

[25] M. Raissi and G. E. Karniadakis, "Machine learning of linear differential equations using gaussian processes," *arXiv preprint arXiv:1701.02440*, 2017.

[26] ——, "Hidden physics models: Machine learning of nonlinear partial differential equations," *Journal of Computational Physics*, vol. 357, pp. 125–141, 2018.

[27] P. Benner, S. Gugercin, and K. Willcox, "A survey of projection-based model reduction methods for parametric dynamical systems," *SIAM review*, vol. 57, no. 4, pp. 483–531, 2015.

[28] B. Peherstorfer and K. Willcox, "Data-driven operator inference for nonintrusive projection-based model reduction," *Computer Methods in Applied Mechanics and Engineering*, vol. 306, pp. 196–215, 2016.

[29] E. Qian, B. Kramer, B. Peherstorfer, and K. Willcox, "Lift & learn: Physics-informed machine learning for large-scale nonlinear dynamical systems," *Physica D: Nonlinear Phenomena*, vol. 406, p. 132401, 2020.

[30] D. Giannakis and A. J. Majda, "Nonlinear laplacian spectral analysis for time series with intermittency and low-frequency variability," *Proceedings of the National Academy of Sciences*, vol. 109, no. 7, pp. 2222–2227, 2012.

[31] O. Yair, R. Talmon, R. R. Coifman, and I. G. Kevrekidis, "Reconstruction of normal forms by learning informed observation geometries from data," *Proceedings of the National Academy of Sciences*, p. 201620045, 2017.

[32] J. Bongard and H. Lipson, "Automated reverse engineering of nonlinear dynamical systems," *Proceedings of the National Academy of Sciences*, vol. 104, no. 24, pp. 9943–9948, 2007.

[33] M. Schmidt and H. Lipson, "Distilling free-form natural laws from experimental data," *Science*, vol. 324, no. 5923, pp. 81–85, 2009.

[34] B. C. Daniels and I. Nemenman, "Automated adaptive inference of phenomenological dynamical models," *Nature communications*, vol. 6, 2015.

[35] S. H. Rudy, S. L. Brunton, J. L. Proctor, and J. N. Kutz, "Data-driven discovery of partial differential equations," *Science Advances*, vol. 3, no. e1602614, 2017.

[36] M. Sorokina, S. Sygletos, and S. Turitsyn, "Sparse identification for nonlinear optical communication systems: SINO method," *Optics express*, vol. 24, no. 26, pp. 30 433–30 443, 2016.

[37] J.-C. Loiseau and S. L. Brunton, "Constrained sparse Galerkin regression," *Journal of Fluid Mechanics*, vol. 838, pp. 42–67, 2018.

[38] M. Dam, M. Brøns, J. Juul Rasmussen, V. Naulin, and J. S. Hesthaven, "Sparse identification of a predator-prey system from simulation data of a convection model," *Physics of Plasmas*, vol. 24, no. 2, p. 022310, 2017.

[39] J.-C. Loiseau, B. R. Noack, and S. L. Brunton, "Sparse reduced-order modeling: sensor-based dynamics to full-state estimation," *Journal of Fluid Mechanics*, vol. 844, pp. 459–490, 2018.

[40] M. Hoffmann, C. Fröhner, and F. Noé, "Reactive SINDy: Discovering governing reactions from concentration data," *Journal of Chemical Physics*, vol. 150, no. 025101, 2019.

[41] J.-C. Loiseau, "Data-driven modeling of the chaotic thermal convection in an annular thermosyphon," *arXiv preprint arXiv:1911.07920*, 2019.

[42] Y. El Sayed M, R. Semaan, and R. Radespiel, "Sparse modeling of the lift gains of a high-lift configuration with periodic coanda blowing," in *2018 AIAA Aerospace Sciences Meeting*, 2018, p. 1054.

[43] A. Narasingam and J. S.-I. Kwon, "Data-driven identification of interpretable reduced-order models using sparse regression," *Computers & Chemical Engineering*, vol. 119, pp. 101–111, 2018.

[44] B. de Silva, D. M. Higdon, S. L. Brunton, and J. N. Kutz, "Discovery of physics from data: Universal laws and discrepancies," *arXiv preprint arXiv:1906.07906*, 2019.

[45] S. Thaler, L. Paehler, and N. A. Adams, "Sparse identification of truncation errors," *Journal of Computational Physics*, vol. 397, p. 108851, 2019.

[46] Z. Lai and S. Nagarajaiah, "Sparse structural system identification method for nonlinear dynamic systems with hysteresis/inelastic behavior," *Mechanical Systems and Signal Processing*, vol. 117, pp. 813–842, 2019.

[47] N. Deng, B. R. Noack, M. Morzynski, and L. R. Pastur, "Low-order model for successive bifurcations of the fluidic pinball," *Journal of fluid mechanics*, vol. 884, no. A37, 2020.

[48] M. Schmelzer, R. P. Dwight, and P. Cinnella, "Discovery of algebraic reynolds-stress models using sparse symbolic regression," *Flow, Turbulence and Combustion*, vol. 104, no. 2, pp. 579–603, 2020.

[49] S. Pan, N. Arnold-Medabalimi, and K. Duraisamy, "Sparsity-promoting algorithms for the discovery of informative koopman invariant subspaces," *arXiv preprint arXiv:2002.10637*, 2020.

[50] S. Beetham and J. Capecelatro, "Formulating turbulence closures using sparse regression with embedded form invariance," *arXiv preprint arXiv:2003.12884*, 2020.

[51] N. M. Mangan, S. L. Brunton, J. L. Proctor, and J. N. Kutz, "Inferring biological networks by sparse identification of nonlinear dynamics," *IEEE Transactions on Molecular, Biological, and Multi-Scale Communications*, vol. 2, no. 1, pp. 52–63, 2016.

[52] H. Schaeffer, "Learning partial differential equations via data discovery and sparse optimization," in *Proc. R. Soc. A*, vol. 473, no. 2197. The Royal Society, 2017, p. 20160446.

[53] N. M. Mangan, J. N. Kutz, S. L. Brunton, and J. L. Proctor, "Model selection for dynamical systems via sparse regression and information criteria," *Proceedings of the Royal Society A*, vol. 473, no. 2204, pp. 1–16, 2017.

[54] G. Tran and R. Ward, "Exact recovery of chaotic systems from highly corrupted data," *Multiscale Modeling & Simulation*, vol. 15, no. 3, pp. 1108–1129, 2017.

[55] H. Schaeffer and S. G. McCalla, "Sparse model selection via integral terms," *Physical Review E*, vol. 96, no. 2, p. 023302, 2017.

[56] H. Schaeffer, G. Tran, and R. Ward, "Extracting sparse high-dimensional dynamics from limited data," *SIAM Journal on Applied Mathematics*, vol. 78, no. 6, pp. 3279–3295, 2018.

[57] K. Wu and D. Xiu, "Numerical aspects for approximating governing equations using data," *arXiv preprint arXiv:1809.09170*, 2018.

[58] L. Boninsegna, F. Nüske, and C. Clementi, "Sparse learning of stochastic dynamical equations," *The Journal of chemical physics*, vol. 148, no. 24, p. 241723, 2018.

[59] E. Kaiser, J. N. Kutz, and S. L. Brunton, "Sparse identification of nonlinear dynamics for model predictive control in the low-data limit," *Proceedings of the Royal Society of London A*, vol. 474, no. 2219, 2018.

[60] N. M. Mangan, T. Askham, S. L. Brunton, J. N. Kutz, and J. L. Proctor, "Model selection for hybrid dynamical systems via sparse regression," *Proceedings of the Royal Society A*, vol. 475, no. 2223, p. 20180534, 2019.

[61] P. Gelß, S. Klus, J. Eisert, and C. Schütte, "Multidimensional approximation of nonlinear dynamical systems," *Journal of Computational and Nonlinear Dynamics*, vol. 14, no. 6, 2019.

[62] A. Goeßmann, M. Götte, I. Roth, R. Sweke, G. Kutyniok, and J. Eisert, "Tensor network approaches for learning non-linear dynamical laws," *arXiv preprint arXiv:2002.12388*, 2020.

[63] P. A. Reinbold, D. R. Gurevich, and R. O. Grigoriev, "Using noisy or incomplete data to discover models of spatiotemporal dynamics," *Physical Review E*, vol. 101, no. 1, p. 010203, 2020.

[64] L. Zhang and H. Schaeffer, "On the convergence of the sindy algorithm," *Multiscale Modeling & Simulation*, vol. 17, no. 3, pp. 948–972, 2019.

[65] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, 1996. [Online]. Available: http://www.jstor.org/stable/2346178

[66] P. Zheng, T. Askham, S. L. Brunton, J. N. Kutz, and A. Y. Aravkin, "A unified framework for sparse relaxed regularized regression: Sr3," *IEEE Access*, vol. 7, pp. 1404–1423, 2018.

[67] K. Champion, P. Zheng, A. Y. Aravkin, S. L. Brunton, and J. N. Kutz, "A unified sparse optimization framework to learn parsimonious physics-informed models from data," *arXiv preprint arXiv:1906.10612*, 2019.

[68] S. Zhang and G. Lin, "Robust data-driven discovery of governing physical laws with error bars," *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 474, no. 2217, p. 20180305, 2018.

[69] W. Pan, Y. Yuan, J. Gonçalves, and G. Stan, "A sparse Bayesian approach to the identification of nonlinear state-space systems," *IEEE Transactions on Automatic Control*, vol. 61, no. 1, pp. 182–187, January 2016.

[70] R. K. Niven, A. Mohammad-Djafari, L. Cordier, M. Abel, and M. Quade, "Bayesian identification of dynamical systems," *Multidisciplinary Digital Publishing Institute Proceedings*, vol. 33, no. 1, p. 33, 2020.

[71] M. Quade, "sparsereg - collection of modern sparse regression algorithms," Feb. 2018. [Online]. Available: https://github.com/ohjeah/sparsereg

[72] R. Chartrand, "Numerical differentiation of noisy, nonsmooth data," *ISRN Applied Mathematics*, vol. 2011, 2011.

[73] C. M. Bishop, *Pattern recognition and machine learning.* springer, 2006.

[74] K. Taira, S. L. Brunton, S. Dawson, C. W. Rowley, T. Colonius, B. J. McKeon, O. T. Schmidt, S. Gordeyev, V. Theofilis, and L. S. Ukeiley, "Modal analysis of fluid flows: An overview," *AIAA Journal*, vol. 55, no. 12, pp. 4013–4041, 2017.

[75] K. Taira, M. S. Hemati, S. L. Brunton, Y. Sun, K. Duraisamy, S. Bagheri, S. Dawson, and C.-A. Yeh, "Modal analysis of fluid flows: Applications and outlook," *AIAA Journal*, vol. 58, no. 3, pp. 998–1022, 2020.

[76] S. L. Brunton, B. R. Noack, and P. Koumoutsakos, "Machine learning for fluid mechanics," *Annual Review of Fluid Mechanics*, vol. 52, pp. 477–508, 2020.

[77] P. Sachdeva, J. Livezey, A. Tritt, and K. Bouchard, "Pyuoi: The union of intersections framework in python," *Journal of Open Source Software*, vol. 4, no. 44, p. 1799, 2019.