DOI: 10.58895/ksp/1000174544-4 erschienen in:
**Proceedings - 34. Workshop Computational Intelligence: Berlin, 21.-22. November 2024**
DOI: 10.58895/ksp/1000174544 | https://www.ksp.kit.edu/site/books/e/10.58895/ksp/1000174544/

# An Application-oriented Review of Standard and Integral Sparse Identification of Nonlinear Dynamics

Mortimer Dockhorn, Martin Kohlhase

Institute for Data Science Solutions, Hochschule Bielefeld
Interaktion 1, 33619 Bielefeld
E-Mail: {mortimer.dockhorn, martin.kohlhase}@hsbi.de

## 1 Introduction

The aim of this article is to provide an application-oriented overview of two variants of the Sparse Identification of Nonlinear Dynamics (SINDy). The original variant introduced by Brunton et al. in [1], and a modified integral version presented by Schaeffer and McCalla in [2]. The SINDy algorithm is, in general, a method for estimating the governing equations of nonlinear dynamical systems from measurement data. It is based on sparse regression and therefore designed to find a parsimonious mathematical model with the fewest terms necessary to accurately describe the underlying dynamics of a system. SINDy leverages techniques from the field of data science to produce interpretable models that can provide additional insight into the fundamental principles that describe a system's behavior. Data-driven methods for system identification are also valuable in scenarios where more traditional approaches, such as modeling from first principles, are impractical due to the high complexity of a system or because the governing scientific laws are not fully known. These techniques can therefore accelerate industrial tasks, such as controller design, and support research by providing dynamic models that offer new insights into a process.

As presented in [1], SINDy is capable of estimating a nonlinear discrete-time model in the form of difference equations as well as a nonlinear continuous-time model in the form of ordinary differential equations (ODEs). This work,

however, focuses solely on continuous-time models, as ODEs are likely to provide a better basis for an in-depth analysis of a system. This is partly due to the fact that the form and parameter values of difference equations are influenced by factors such as the discretization method or the sample time, whereas the attributes of ODEs are determined solely by the system they describe. ODEs, therefore, provide more accessible information when governing physical laws or the physical interpretation of parameters are of interest. For more information on the estimation of discrete-time models with SINDy, the reader is referred to [1, 3].

The nonlinear ODE model generated by SINDy is of the general form

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

where $\boldsymbol{u}(t)$, $\boldsymbol{x}(t)$, $\dot{\boldsymbol{x}}(t)$, and $\boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t), t)$ are vectors whose elements represent potential inputs to the system, the system's state variables, the state variables' first derivatives with respect to time, and the corresponding state equations. This model is estimated from training data for $\boldsymbol{u}(t)$, $\boldsymbol{x}(t)$, and $\dot{\boldsymbol{x}}(t)$ by approximating the right-hand side of (1) as linear combinations of user-defined candidate functions. The construction of training datasets therefore requires data of all the inputs, state variables, and state variables' first derivatives with respect to time. Especially the last requirement, however, creates challenges since the time derivatives of a system's state variables are not always measurable, and common numerical differentiation techniques (e.g., finite differences) are likely to produce corrupted derivative approximations when applied to noisy measurements of the state variables. To address this issue, more advanced methods, such as the Total Variation Regularized Derivative [4], Spline Polynomial Interpolation [5], and an algorithm based on the Savitzky-Golay Filter [6] were proposed in [1, 7–9], [10], and [11], respectively. While these methods produce significantly more accurate derivative approximations of noise-contaminated data, some of them, like the Savitzky-Golay Filter, can corrupt the data, and therefore the derivative, by altering curvature or moving extrema. Other methods, such as the Total Variation Regularized Derivative, may be less prone to altering the data but have limitations on the size of the dataset they can be applied to. To address these issues, an algorithm for batchwise Tikhonov Regularized Differentiation (TRD) is presented in this work. The

algorithm extends the basic TRD [12], which has high accuracy compared to many of the aforementioned methods [5], but is also limited with regard to the size of the dataset it can be applied to. Batchwise TRD splits larger datasets into batches for the differentiation and merges the individual segments back into a single derivative, allowing it to be applied to datasets of all sizes. This combination of high accuracy and flexibility makes batchwise TRD a suitable tool for generating training data for SINDy when only noisy measurements of a system's state variables are available. For a more detailed description of most of the differentiation methods referenced in this work, the reader is referred to [5].

An alternative variant of SINDy, that does not require the state variables' derivatives for the construction of training datasets, was introduced in [2]. This variant will be referred to as integral SINDy (I-SINDy) within this work, as the key difference between SINDy and I-SINDy is given by the fact that I-SINDy uses the time integral of relationship (1) for the model estimation. This modified approach also translates to the training data, as the time integrals of the state variables are required for the construction of training datasets instead of their time derivatives. While these integrals are also not measurable in many cases, computing them from noisy measurement data of the state variables poses notably less challenges then it does for the derivatives. This is due to the fact that even simple numerical integration schemes resemble a high robustness against noise. Moreover, numerical integration is performed iteratively and therefore not constraint with regard to the dataset size. I-SINDy and SINDy with batchwise TRD can therefore be recognized as alternative options for the identification of nonlinear ODE models, when only noisy measurements of a system's states are available. This leads to the aim of this article, which is providing an application-oriented comparison of both methods.

The remainder of the article is structured as follows. The general SINDy framework and the modifications resulting in I-SINDy are presented together with the batchwise TRD algorithm in section 2. The robustness of both methods to noise in the training data as well as the results that were achieved with data of a real world tank system, are illustrated in section 3. The results are summarized in section 4 and future research directions are discussed.

# 2 The SINDy, I-SINDy, and TRD Methods

This section provides a detailed overview of the SINDy and I-SINDy methods, as well as the batchwise TRD algorithm. The general framework of SINDy is illustrated in 2.1, and the modifications leading to I-SINDy are summarized in 2.2. Subsequently, the batchwise TRD algorithm is be presented in 2.3.

## 2.1 The SINDy Method

Since SINDy was originally introduced in [1], the general concepts discussed here are drawn from that source unless stated otherwise.

SINDy estimates a nonlinear ODE model of the the general form

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

where $\boldsymbol{u}(t) \in \mathbb{R}^p$ and $\boldsymbol{x}(t)$, $\dot{\boldsymbol{x}}(t)$, $\boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t), t) \in \mathbb{R}^n$ are column vectors whose elements represent the inputs to the system, the system's state variables, the state variables' first derivatives with respect to time, and the corresponding state equations. The model is estimated from training data for $\boldsymbol{u}(t)$, $\boldsymbol{x}(t)$ and $\dot{\boldsymbol{x}}(t)$ by approximating the individual state equations in terms of of $k$ user-defined candidate functions $g_i : Y_i \to Z_i$, $i \in \{1, 2, \ldots, k\}$. Assuming the training data is real-valued, $g_i(\cdot)$ is a scalar and real-valued function of $w_i$ scalar and real-valued arguments. Formally, this equates to $Z_i \subseteq \mathbb{R}$, and $Y_i \coloneqq Y_{i,1} \times Y_{i,2} \cdots \times Y_{i,w_i}$ with $Y_{i,j} \subseteq \mathbb{R}$ for all $j \in \{1, 2, \ldots, w_i\}$. The model estimation process yields $n$ coefficient vectors $\boldsymbol{v}_i \coloneqq [v_{i,1} \ \ldots \ v_{i,k}]^{\mathrm{T}} \in \mathbb{R}^k$, $i \in \{1, 2, \ldots, n\}$, with each vector determining the contribution of the candidate functions to a corresponding state equation. The estimated model is therefore composed of $n$ state equations, with each being a linear combination of the candidate functions.

$$\dot{\boldsymbol{x}}(t) = \begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \\ \vdots \\ \dot{x}_n(t) \end{bmatrix} = \begin{bmatrix} v_{1,1} g_1(\cdot) + v_{1,2} g_2(\cdot) \cdots + v_{1,k} g_k(\cdot) \\ v_{2,1} g_1(\cdot) + v_{2,2} g_2(\cdot) \cdots + v_{2,k} g_k(\cdot) \\ \vdots \qquad\qquad \vdots \qquad \ddots \qquad \vdots \\ v_{n,1} g_1(\cdot) + v_{n,2} g_2(\cdot) \cdots + v_{n,k} g_k(\cdot) \end{bmatrix} \tag{2}$$

For the construction of training datasets, the data for $\boldsymbol{u}(t)$, $\boldsymbol{x}(t)$, and $\dot{\boldsymbol{x}}(t)$ are required. Since this section focuses on the general framework of SINDy, it will be assumed that these data are available in sufficiently high quality and that they are given as discrete measurements collected at the time instances $t_1, t_2, \ldots, t_m$.

A training dataset for SINDy consists of three data matrices and the vector $\boldsymbol{t} := [t_1 \ \ldots \ t_m]^{\mathrm{T}} \in \mathbb{R}^m$. To construct the matrices, the measurements are arranged into the column vectors $\mu_i$, $\chi_j$, $\dot{\chi}_j \in \mathbb{R}^m$, with $i \in \{1, 2, \ldots, p\}$ and $j \in \{1, 2, \ldots, n\}$, where each vector holds the data for one input, state variable, or time derivative, respectively.

$$\mu_i := \begin{bmatrix} u_i(t_1) \\ \vdots \\ u_i(t_m) \end{bmatrix}, \quad \chi_j := \begin{bmatrix} x_j(t_1) \\ \vdots \\ x_j(t_m) \end{bmatrix}, \quad \dot{\chi}_j := \begin{bmatrix} \dot{x}_j(t_1) \\ \vdots \\ \dot{x}_j(t_m) \end{bmatrix}$$

Subsequently, the matrices $\boldsymbol{U} \in \mathbb{R}^{m \times p}$ and $\boldsymbol{X}$, $\dot{\boldsymbol{X}} \in \mathbb{R}^{m \times n}$ are constructed from these vectors and provided to SINDy as training data along with $\boldsymbol{t}$.

$$\boldsymbol{U} := \begin{bmatrix} \mu_1 & \ldots & \mu_p \end{bmatrix}, \quad \boldsymbol{X} := \begin{bmatrix} \chi_1 & \ldots & \chi_n \end{bmatrix}, \quad \dot{\boldsymbol{X}} := \begin{bmatrix} \dot{\chi}_1 & \ldots & \dot{\chi}_n \end{bmatrix}$$

In addition to the training data, SINDy requires the user to define the candidate functions. This step is analogous to the tuning of an essential hyperparameter, as the quality of the estimated model largely depends on it. In general, any function with a domain and codomain compatible with the training data can be used as a candidate function. In practice, the selection of candidate functions should be based on the characteristics of the system being modeled. For example, if the system exhibits oscillatory behavior, trigonometric functions such as sine and cosine may be a reasonable choice. Another common choice is powers of the inputs or state variables, as they often appear in models that were derived from first principles. A constant bias, as well as functions that depend solely on time, may also be selected if the corresponding behavior is observed in the system.

Provided with the training data and candidate functions, SINDy constructs the library matrix

$$\boldsymbol{D} \coloneqq \begin{bmatrix} g_1(\cdot)\big|_{t=t_1} & g_2(\cdot)\big|_{t=t_1} & \cdots & g_k(\cdot)\big|_{t=t_1} \\ g_1(\cdot)\big|_{t=t_2} & g_2(\cdot)\big|_{t=t_2} & \cdots & g_k(\cdot)\big|_{t=t_2} \\ \vdots & \vdots & \ddots & \vdots \\ g_1(\cdot)\big|_{t=t_m} & g_2(\cdot)\big|_{t=t_m} & \cdots & g_k(\cdot)\big|_{t=t_m} \end{bmatrix} \in \mathbb{R}^{m \times k}.$$

The structure of $\mathbf{D}$ is such that each column contains the values of a candidate function evaluated at the time instances in $\mathbf{t}$.

Following the construction of the library matrix, SINDy estimates the individual state equations. Within this process, Sparse Regression Methods (SRMs) are employed for solving the optimization problems that yield the coefficient vectors. SRMs are used in in particular, as they are designed to balance model accuracy and complexity. Within the context of SINDy, this translates to the objective of finding a set of state equations, where each is composed of the fewest number of candidate functions necessary to accurately describe the dynamics of the training data. The use of SRMs, therefore, counteracts overfitting and prevents unnecessary model complexity. Moreover, the results in [1] suggest that sparse models are more likely to represent the actual physical laws that govern a system's behavior. Examples for SRMs, that have been used particularly within the SINDy framework, are the Sequential Threshold Least Squares algorithm [1], the Douglas-Rachford algorithm [2, 13], the Least Absolute Shrinkage and Selection Operator [11, 14], and Sparse Relaxed Regularized Regression [15, 16]. As the referenced literature suggests, that all of these methods are compatible with the SINDy framework, the choice of a particular SRM is not as important as, for instance, the choice of candidate functions. However, some methods may still perform better in specific scenarios than others (e.g., limited training data or high noise levels). It is therefore suggested, to perform an initial comparison of multiple models, where each is estimated with a different SRM. This approach is feasible without to much additional effort, as an open source software package for SINDy ( [3]) is available, that allows the user to choose from different pre-implemented SRMs. Since the specifics of the aforementioned sparse regression algorithms are beyond the

scope of this article, a general approach for the estimation of the state equations will be illustrated in the remainder of this section.

SINDy performs the model estimation in $n$ steps, corresponding to the estimation of $n$ state equations. In each step, the selected SRM aims to recover a sparse coefficient vector that provides the best fit to the training data. Independent of the particular choice of SRM, the overall objective of the estimation process may be expressed as

$$\underset{\boldsymbol{v}_i \in S}{\operatorname{argmin}}(\dot{\boldsymbol{\chi}}_i - \boldsymbol{D}\boldsymbol{v}_i), \quad i \in \{1,2,\ldots,n\}. \tag{3}$$

where $S$ is the set of all sparse coefficient vectors. It should be noted however, that (3) is only used for conveying the general aim of the optimization, as the particular form of the optimization problem depends on the choice of SRM.

Subsequent to the calculation of all coefficient vectors, SINDy constructs the row vector

$$\boldsymbol{\delta}^{\mathrm{T}}(t) \coloneqq \begin{bmatrix} g_1(\cdot) & g_2(\cdot) & \ldots & g_k(\cdot) \end{bmatrix} \in \mathbb{R}^k,$$

whose elements are the symbolic candidate functions, arranged in the same order as the corresponding data in $\boldsymbol{D}$. It should be noted, that $\boldsymbol{D}$ contains discrete data values that were generated by applying the candidate functions to the training data, while $\boldsymbol{\delta}^{\mathrm{T}}(t)$ contains the actual symbolic candidate functions.

In the final step, SINDy recovers the state equations via the relationship

$$\dot{x}_i(t) = \boldsymbol{\delta}^{\mathrm{T}}(t)\boldsymbol{v}_i \quad i \in \{1,2,\ldots,n\}.$$

Which is equivalent to the form (2), that was used in the introduction of this section.

## 2.2 The I-SINDy method

This subsection illustrates the modifications to the general SINDy framework that result in I-SINDy. It therefore builds strongly on the content of 2.1, where the general SINDy framework is presented. Accordingly, 2.1 is recommended as preliminary literature. As I-SINDy was originally introduced in [1], the general concepts illustrated here are drawn from that source unless stated otherwise.

I-SINDy also estimates a nonlinear ODE model of the general form (1). For the model estimation process, however, I-SINDy uses the relationship

$$\boldsymbol{x}(t) - \boldsymbol{x}(t_1) = \int_{t_1}^{t} \boldsymbol{f}(\boldsymbol{x}(\tau),\, \boldsymbol{u}(\tau),\, \tau)\, \mathrm{d}\tau, \tag{4}$$

which is the time integral of (1), assuming measurements begin at $t = t_1$. Equivalent to SINDy, I-SINDy approximates the right-hand side of (4) as linear combinations of candidate functions. However, the derivatives of the candidate functions are first defined in terms of $\boldsymbol{x}(t)$, $\boldsymbol{u}(t)$, and $t$ and then numerically integrated. Thus, the construction of training datasets requires the measurement data to be integrated rather than differentiated.

Provided with the training data and choice of candidate functions, I-SINDy constructs the library matrix $\boldsymbol{D}$. Since this process is identical for SINDy and I-SINDy, the reader is referred to section 2.1, where a detailed description is provided. As I-SINDy estimates the integrals of the state equations, the numerical integrals of the columns of $\boldsymbol{D}$ are used for the estimation. This ensures that the state equations of the final ODE model are given as linear combinations of the selected candidate functions and thus makes it easier to chose candidate functions based on knowledge of the system being modeled. With $\boldsymbol{D} = [\boldsymbol{d}_1,\, \boldsymbol{d}_2, \ldots,\, \boldsymbol{d}_k]$, I-SINDy constructs the integral library matrix

$$\boldsymbol{D}^{\mathrm{int}} := \begin{bmatrix} \boldsymbol{d}_1^{\mathrm{int}} & \boldsymbol{d}_2^{\mathrm{int}} & \ldots & \boldsymbol{d}_k^{\mathrm{int}} \end{bmatrix} \in \mathbb{R}^{m \times k},$$

where $\boldsymbol{d}_i^{\mathrm{int}} \in \mathbb{R}^m$ is the column vector of data values that was acquired by numerical cumulative integration of the column $\boldsymbol{d}_i$. Subsequently, I-SINDy performs the estimation of model (4) with the Data in $\boldsymbol{D}^{\mathrm{int}}$. As the remaining estimation

process is mostly identical to the process within SINDy, the remainder of this subsection will only provide a brief summary.

Similar to SINDy, I-SINDy aims to recover sparse coefficient vectors. The general objective of the optimization is just a slightly modified version of (3).

$$\underset{\boldsymbol{v}_i \in S}{\operatorname{argmin}}(\widetilde{\boldsymbol{\chi}}_i - \boldsymbol{D}^{\text{int}}\boldsymbol{v}_i), \quad i \in \{1, 2, \ldots, n\}.$$

Instead of derivative data, the vector $\widetilde{\boldsymbol{\chi}}_i \coloneqq \boldsymbol{\chi}_i - x_i(t_1)\mathbf{1}_m \in \mathbb{R}^m$, is used in the optimization, which modifies the data in $\boldsymbol{\chi}_i$ to represent the integral of $\dot{\boldsymbol{\chi}}_i$, with the condition $\boldsymbol{x}(t_1) = 0$. This step is required since the numerical integration causes the first value of all $\boldsymbol{d}_i^{\text{int}}$, $i \in \{1,\ 2,\ \ldots,\ k\}$ to be zero.

Equivalent to SINDy, the desired ODE model (1) can be composed from the state equations

$$\dot{x}_i(t) = \boldsymbol{\delta}^{\mathrm{T}}(t)\,\boldsymbol{v}_i, \ \ i \in \{1,\ 2,\ \ldots,\ n\}.$$

Here $\boldsymbol{\delta}^{\mathrm{T}}(t)$ is a row vector that contains the symbolic candidate functions corresponding to the columns of $\boldsymbol{D}$ (not $\boldsymbol{D}^{\text{int}}$).

## 2.3 Batchwise Tikhonov Regularized Differentiation

This subsection introduces the batchwise TRD algorithm. First, the concept of standard TRD is illustrated through an example implementation proposed in [5], and subsequently the batchwise TRD algorithm is presented.

Given a column vector $\boldsymbol{y} \coloneqq [y_1\ y_2\ \ldots\ y_n]^{\mathrm{T}} \in \mathbb{R}^n$ that contains discrete data, collected at the corresponding time instances $t_1, t_2, \ldots, t_n$, TRD defines the $L_2$-regularized optimization problem

$$\underset{\dot{\boldsymbol{y}}}{\operatorname{argmin}} \|\boldsymbol{A}\dot{\boldsymbol{y}} - \widetilde{\boldsymbol{y}}\|_2^2 + k\|\boldsymbol{B}\dot{\boldsymbol{y}}\|_2^2. \tag{5}$$

Solving (5) yields the column vector $\dot{\boldsymbol{y}} \in \mathbb{R}^n$, whose elements approximate the first time derivative of the data in $\boldsymbol{y}$ at the time instances $t_1, t_2, \ldots, t_n$. To provide

a foundation for understanding this concept, the individual elements in (5) are defined in the following passage.

The matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ is constructed such that calculating the product $\boldsymbol{A}\dot{\boldsymbol{y}}$ is equivalent to applying the cumulative trapezoidal rule to $\dot{\boldsymbol{y}}$. Assuming a uniform sample time $\Delta t \coloneqq t_{i+1} - t_i$ for all $i \in \{1, 2, \ldots, n-1\}$, this yields

$$\boldsymbol{A} \coloneqq \frac{1}{2}\Delta t \begin{bmatrix} 0 & 0 & 0 & \ldots & 0 \\ 1 & 1 & 0 & \ldots & 0 \\ 1 & 2 & 1 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 2 & 2 & \ldots & 1 \end{bmatrix}. \tag{6}$$

Therefore, if $\boldsymbol{y}^{\text{trap}} \coloneqq \boldsymbol{A}\dot{\boldsymbol{y}} = [\, y_1^{\text{trap}} \ \ldots \ y_n^{\text{trap}} \,]^{\mathrm{T}} \in \mathbb{R}^n$, then $y_i^{\text{trap}}$ approximates the definite integral from $t_1$ to $t_i$ of a function that is continuous on $[t_1, t_i]$ and interpolates the first $i$ data values in $\boldsymbol{y}$. Additionally, since (6) is defined such that $y_1^{\text{trap}} = 0$ for any given $\boldsymbol{y}$, the column vector $\widetilde{\boldsymbol{y}} \coloneqq \boldsymbol{y} - y_1 \mathbf{1}_n \in \mathbb{R}^n$ is used in (5), as $\widetilde{y}_1 = 0$ is guaranteed by definition.

The matrix $\boldsymbol{B} \in \mathbb{R}^{(n-2) \times n}$ is defined such that calculating the product $\boldsymbol{B}\dot{\boldsymbol{y}}$ is equivalent to applying the central difference method to $\dot{\boldsymbol{y}}$ while discarding the edge values. The scalar $k \in \mathbb{R}_{>0}$ is a hyperparameter that determines the amount of regularization.

$$\boldsymbol{B} \coloneqq \frac{1}{2\Delta t} \begin{bmatrix} -1 & 0 & 1 & 0 & \ldots & 0 \\ 0 & -1 & 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \ldots & -1 & 0 & 1 \end{bmatrix}$$

Given these definitions, the term $\|\boldsymbol{A}\dot{\boldsymbol{y}} - \widetilde{\boldsymbol{y}}\|_2^2$ in (5) can be interpreted as the requirement to find a $\dot{\boldsymbol{y}}$ whose approximated integral matches the data $\widetilde{\boldsymbol{y}}$ as closely as possible. The term $\|\boldsymbol{B}\dot{\boldsymbol{y}}\|_2^2$, on the other hand, can be interpreted as the requirement that the values of the approximated first derivative of $\dot{\boldsymbol{y}}$ are as small as possible. This latter requirement reduces high-frequency noise in $\dot{\boldsymbol{y}}$, as $\|\boldsymbol{B}\dot{\boldsymbol{y}}\|_2^2$ would become large when such noise is present. The combination of both requirements results in the objective of finding a derivative approximation that

is accurate while containing minimal noise. The tuning of $k$ can be interpreted as adjusting the relative weight of each requirement in the overall objective. In particular, large $k$ prioritize smoothness, whereas small $k$ emphasize accuracy.

Even though a general analytical solution to (5) is given by

$$\dot{\boldsymbol{y}} = (\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A} + k\boldsymbol{B}^{\mathrm{T}}\boldsymbol{B})^{-1}\boldsymbol{A}^{\mathrm{T}}\widetilde{\boldsymbol{y}}, \tag{7}$$

numerical solvers are preferred for solving (5) because they are more robust than the numerical matrix inversion required in (7). In practice, however, constraints on memory and computational resources make TRD unsuitable for large $\boldsymbol{y}$, as (5) is posed as a single optimization problem. To address this issue, the following batchwise TRD algorithm is proposed, which splits larger datasets into batches for the differentiation and merges the individual segments back into a single derivative.

# 3 Results

This section illustrates the results of the performed comparison between SINDy and I-SINDy. In 3.1, the sensitivity of both variants to measurement noise is compared using noise-contaminated simulation datasets from two different models. In 3.2, both methods are applied to the data of a real tank system, and the quality of the estimated models is compared.

## 3.1 Robustness to Noise

In this subsection, two models are considered. The Lotka-Volterra model, which describes the dynamics of interacting predator and prey populations, and the Duffing model, which describes a nonlinear oscillator. The governing equations of both models are listed in table 1. Simulation data for each model is generated, using scipy's solve_ivp method with a Runge-Kutta 45 solver. A time span of 10 seconds is simulated, with a uniform step size of 1 ms. This results in each simulation dataset containing $10^3$ values per state. For convenience in notation

Algorithm 1.: Batchwise TRD

**Voraussetzung:** Dataset to be differentiated: *data*, number of samples that the batches should overlap: *overlap*, uniform sample time: $\Delta t$, regularization parameter: $k$, tolerance value for merging derivative batches: *tol*

**Abschlussbedingung:** An array containing the derivative values: *reg_derivative*

**function** TRD($\boldsymbol{y}$, $k$, $\Delta t$)
  $n \leftarrow$ Number of elements in $\boldsymbol{y}$
  $\boldsymbol{A}, \boldsymbol{B} \leftarrow$ Initialize matrices of size $(n, n)$ and $(n-2, n)$
  $\widetilde{\boldsymbol{y}} \leftarrow$ Subtract first element of $\boldsymbol{y}$ from all elements of $\boldsymbol{y}$
  **for** $i = 0$ to $n-2$ **do**
    $\boldsymbol{A}[i+1, : i+2] \leftarrow \left[\frac{1}{2}, 1, \frac{1}{2}\right] \cdot \Delta t$
  Set $\boldsymbol{B}$: main diagonal $(-2\Delta t)^{-1}$, and 2nd diagonal $(2\Delta t)^{-1}$
  $\dot{\boldsymbol{y}} \leftarrow$ Solve least square: $[\boldsymbol{A}, \sqrt{k}\boldsymbol{B}]^{\mathrm{T}} \dot{\boldsymbol{y}} = [\widetilde{\boldsymbol{y}}, \boldsymbol{0}]^{\mathrm{T}}$
  **return** $\dot{\boldsymbol{y}}$
Initialize empty derivative_list, set data_len and batch_num to 0
**while** batch_num $<$ $\lceil$data_len/batchsize$\rceil$ **do**
  start_idx $\leftarrow$ batch_num $\cdot$ batchsize $-$ overlap $\cdot$ (batch_num $> 0$)
  end_idx $\leftarrow$ min(start_idx + batchsize + overlap, data_len)
  Append TRD(data[*start_idx* : *end_idx*], $k, \Delta t$) to derivative_list
  batch_num++
**for** $i = 0$ to length(derivative_list) $- 2$ **do**
  Find overlap between derivative_list[i] and derivative_list[i+1]
  Trim derivative_list[i] and append relevant part of derivative_list[i+1]
Set reg_derivative to the final merged batch

$q := 10^3$ within this section. The simulation data is then contaminated with zero-mean normally distributed noise and subsequently used as training data for both SINDy variants. This yields the general relationship $\boldsymbol{x}_i^{\text{train}} := \boldsymbol{x}_i^{\text{sim}} + \boldsymbol{d}_i$, where $\boldsymbol{x}_i^{\text{train}}$, $\boldsymbol{x}_i^{\text{sim}} \in \mathbb{R}^q$ are vectors containing the training and simulation data for the $i$-th state variable, respectively. The vector $\boldsymbol{d}_i := [d_i(t_1)\ d_i(t_2)\ \ldots d_i(t_q)] \in \mathbb{R}^q$ contains the corresponding noise values, which are drawn from a normal distribution.

$$d_i(t) \sim \mathcal{N}(0, \sigma_i^2)$$

Table 1: Models used for generating the training and test data

| Model | Governing Equations |
|---|---|
| Lotka-Volterra | $\dot{x}_1(t) = 1.5x_1(t) - x_1(t)x_2(t)$<br>$\dot{x}_2(t) = x_1(t)x_2(t) - 3x_2(t)$ |
| Duffing | $\dot{x}_1(t) = x_2(t)$<br>$\dot{x}_2(t) = -0.2x_2(t) - x_1(t) - x_1^3(t)$ |

The intensity of the noise relative to the simulation data is defined in terms of the noise level NL, which itself is defined as the ratio of the energies of the noise and the simulation data.

$$\mathrm{NL} := \frac{\|\boldsymbol{d}_i\|_2^2}{\|\boldsymbol{x}_i^{\mathrm{sim}}\|_2^2} = \frac{\sigma_i^2}{\|\boldsymbol{x}_i^{\mathrm{sim}}\|_2^2}$$

Since all noise values are drawn from a zero-mean normal distribution, the energy of the noise, that is added to the $i$-th state variable, can be approximated by the variance $\sigma_i^2$. This relationship is used for the simulation data of each state variable individually, to calculate the variance that yields a desired noise level. In particular, 4 noise levels are considered for the comparison. The intensities of these levels relative to the simulation data are illustrated in figures provided in the Appendix: Supplementary Figures for Section 3.

$$\mathrm{NL} \in \{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$$

For the comparison of both SINDy variants, 4 training datasets, corresponding to the 4 noise levels are constructed for each model in table 1. These datasets are provided to the SINDy software package [3], and the standard and weak form integral variants are used. All models are estimated with the Sequential Threshold Least Squares optimizer, and the derivatives required by SINDy are approximated with batchwise TRD. The training data are not preprocessed or smoothed in any way. In all scenarios, integer powers of the state variables up to the tenth power and products of both state variables, with one variable also

Table 2: Initial conditions used for the generation of training and test datasets.

| Model | Training data | Test data |
|---|---|---|
| Lotka-Volterra | $x_1(t_0) = 10$<br>$x_2(t_0) = 5$ | $x_1(t_0) = 1$<br>$x_2(t_0) = 10$ |
| Duffing | $x_1(t_0) = 5$<br>$x_2(t_0) = 2$ | $x_1(t_0) = 1$<br>$x_2(t_0) = 4$ |

going up to the tenth power, are used as candidate functions. A bias is also included in the library. The accuracy of each model is then measured through cross validation. Test datasets are generated by creating simulation data with different initial conditions than used for the training data. All initial conditions employed in this comparison are listed in table 2. The estimated models are simulated with identical solver settings as those used for the training and test data. The accuracy of the models is quantified using the root mean squared error $\varepsilon$ between the model trajectories and the noise free training and test data, respectively. This yields

$$\varepsilon_i^{\text{train}} := \frac{\|\boldsymbol{x}_i^{\text{sim}} - \hat{\boldsymbol{x}}_i\|_2}{\sqrt{q}}, \qquad \varepsilon_i^{\text{test}} := \frac{\|\boldsymbol{x}_i^{\text{test}} - \hat{\boldsymbol{x}}_i\|_2}{\sqrt{q}}.$$

Here $\boldsymbol{x}_i^{\text{test}}$, $\hat{\boldsymbol{x}}_i \in \mathbb{R}^q$ are vectors, containing the test and the model data for the $i$-th state variable, respectively. For each model, only the highest error value in the state variables is considered for the evaluation of the model accuracy. Since both models in table 1 have two state variables, the following errors are defined

$$\varepsilon\text{-Train} := \max(\varepsilon_1^{\text{train}}, \varepsilon_2^{\text{train}}),$$
$$\varepsilon\text{-Test} := \max(\varepsilon_1^{\text{test}}, \varepsilon_2^{\text{test}}).$$

Another metric used to quantify the accuracy of the estimated models is the Maximum Coefficient Error MCE. For a general definition of the MCE, the

absolute coefficient error $\Delta c^{\text{abs}}$ is first defined for a model with $o$ coefficients.

$$\Delta c_i^{\text{abs}} := \left| c_i^{\text{ref}} - c_i^{\text{mod}} \right|, \quad i \in \{1, 2, \ldots, o\}$$

Here $c_i^{\text{ref}}$ is the $i$-th coefficient of the reference model that was used for the generation of training and test data, and $c_i^{\text{mod}}$ is the corresponding coefficient of the estimated model. Given this definition, the MCE is then defined as

$$\text{MCE} := \frac{\max(\Delta c_1^{\text{abs}},\ \Delta c_2^{\text{abs}}, \ldots, \Delta c_o^{\text{abs}})}{\left| c_{\max}^{\text{mod}} \right|} \cdot 100\,\%. \tag{8}$$

The term $c_{\max}^{\text{mod}}$ refers to the coefficient of the reference model, which has the biggest absolute coefficient error. If, for example, the numerator of (8) yields $\Delta c_2^{\text{abs}}$, then $c_{\max}^{\text{mod}} = c_2^{\text{mod}}$.

The introduced metrics for all estimated models are listed in the tables 3 through 6, where each table holds the results for one noise level. The additional column "Structure" holds information about whether the structure of the reference model could be recovered. A "No" entry is therefore equivalent to a failed system identification. For these cases no metrics are listed, as some of the incorrect models where unstable or completely unable to match the dynamics of the reference model. In Addition, supplementary figures, which illustrate the trajectories of the training and test data as well as the achieved model accuracy are provided in the Appendix: Supplementary Figures for Section 3.

Table 3: Results with NL $= 10^{-4}$

| Model | Method | Structure | MCE | $\varepsilon$-Train | $\varepsilon$-Test |
|---|---|---|---|---|---|
| L.-V. | SINDy | Yes | 0.5 % | 0.38 | 0.57 |
| | I-SINDy | Yes | 0.2 % | $< 0.01$ | 0.03 |
| Duffing | SINDy | Yes | 10.3 % | 0.26 | 0.11 |
| | I-SINDy | Yes | 0.2 % | 0.19 | 0.02 |

Table 4: Results with NL $= 10^{-3}$

| Model | Method | Structure | MCE | $\varepsilon$-Train | $\varepsilon$-Test |
|---|---|---|---|---|---|
| L.-V. | SINDy | Yes | 2.7 % | 1.70 | 2.49 |
| | I-SINDy | Yes | 0.3 % | 0.03 | 0.04 |
| Duffing | SINDy | Yes | 23.7 % | 0.33 | 0.31 |
| | I-SINDy | Yes | 1.8 % | 0.05 | 0.02 |

Table 5: Results with NL $= 10^{-2}$

| Model | Method | Structure | MCE | $\varepsilon$-Train | $\varepsilon$-Test |
|---|---|---|---|---|---|
| L.-V. | SINDy | No | – | – | – |
| | I-SINDy | Yes | 0.2 % | 0.03 | 0.04 |
| Duffing | SINDy | Yes | 158 % | 2.86 | 1.69 |
| | I-SINDy | Yes | 33 % | 0.87 | 0.28 |

Table 6: Results with NL $= 10^{-1}$

| Model | Method | Structure | MCE | $\varepsilon$-Train | $\varepsilon$-Test |
|---|---|---|---|---|---|
| L.-V. | SINDy | No | – | – | – |
| | I-SINDy | Yes | 3.0 % | 1.75 | 2.32 |
| Duffing | SINDy | No | – | – | – |
| | I-SINDy | No | – | – | – |

## 3.2 Tank System

In this subsection, SINDy and I-SINDy are used on measurement data collected from a real-world tank system, as it is shown in figure 1. Two scenarios are considered. In the first, the drain valve connected to tank two is open, and the flow valve connecting the tanks two and three is closed. In this case the system

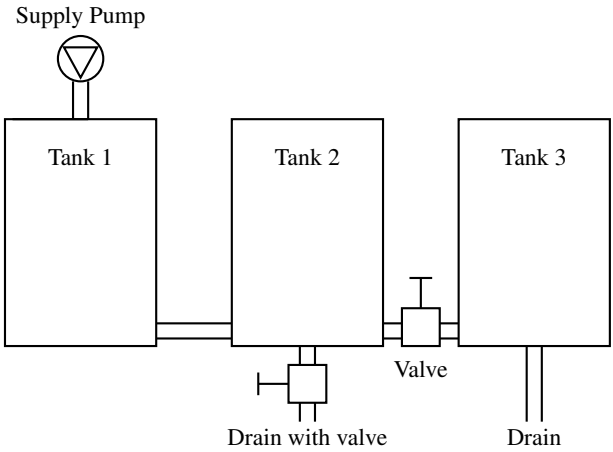



Figure 1: Schematic of the tank system that measurement data was taken from. If the drain valve from tank two is opened and the flow valve connecting tanks 2 and 3 is closed, then the third tank is inactive, yielding a two-tank system. If the drain valve from tank two is closed and the flow valve is opened, then the system operates as three tank system.

operates as a two tank system. In the second scenario, the drain valve from tank two is closed and the flow valve between tanks two and three is open. In this case, all three tanks are active. In both scenarios, the supply pump is always active, providing a continuous inflow into the the first tank. The state variables of the system are the water levels in the tanks. From measurements of the water levels, two training datasets are constructed. One for the operation as two tank system, and another for the operation as three tank system. Each dataset contains measurements sampled with a uniform step size of 1 s. Both datasets contain approximately 1,400 values for each state variable. The training datasets are provided to the SINDy software package, and the standard as well as weak form integral variant are used for estimating a model. Since a model based on Torricelli's law is commonly used for describing similar systems, terms from this model are added to the library. In particular,

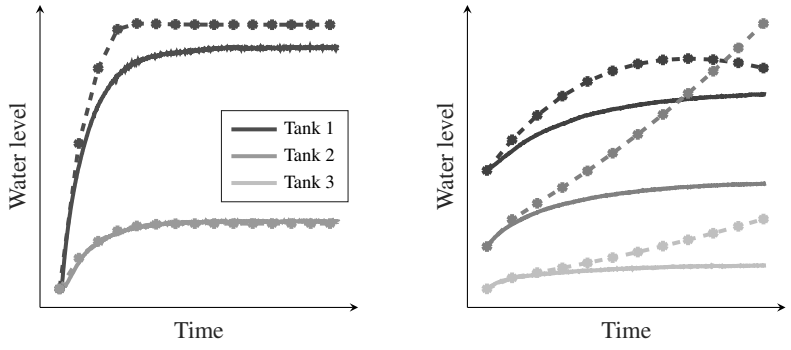
$$\begin{aligned}
&\sqrt{h_i(t)},\\
&\operatorname{sign}(h_i(t)-h_j(t))\cdot\sqrt{\left|h_i(t)-h_j(t)\right|},\\
&\text{with } i,j\in\{1,2,3\},\ i\neq j,
\end{aligned}$$

where $h_1(t), h_2(t)$, and $h_3(t)$ are water levels in the corresponding tanks. In addition, integer powers of the state variables up to the tenth power and products of both state variables, with one variable also going up to the tenth power, are used as candidate functions. A bias, as well as the sine and cosine of the state variables are also included in the library.

The training data and the achieved model quality are illustrated in figure 2. Only I-SINDy was able to estimate a model that resembles the general dynamics of the two tank system (see fig 2a). For the three tank system, I SINDy was not able to estimate a stable model, as it could not be simulated for the time span of the training data. SINDy was not able to estimate an applicable model for either configuration. The water levels simulated with SINDy's two tank model did not deviate from the initial conditions, and the three tank model shows seemingly unstable behavior (see fig. 2b). The two tank model estimated by I-SINDy is given by

$$\begin{aligned}
\dot{h}_1(t) = &-0.164\sqrt{h_1(t)} + 0.132\sqrt{h_2(t)} + 6.035\sin(h_1(t))\ldots \\
&- 160.315\sin(h_2(t)) + 0.016\cos(h_1(t)) - 5.750\,h_1(t)\ldots \\
&+ 159.828\,h_2(t), \\
\dot{h}_2(t) = &\, 0.002\sqrt{h_1(t)} - 9.322\sin(h_2(t)) + 9.300\,h_2(t).
\end{aligned}$$

Even tho this model is able to qualitatively match the dynamics in the training data, the model structure is notably more complicated than for models of equivalent tank systems, that can be found in the literature. Moreover, the presence of the sine and cosine terms in both state equations, does not seem to depict any underlying physical laws, as no oscillatory behavior is observed in the tank system. The complicated model structure as well as the presence of several trigonometric terms seems to indicate overfitting.




(a) Two-tank system (tank 3 inactive)

(b) Three-tank system

Figure 2: Training data (solid) and simulated trajectories (star-dash) for the two-tank system (a) and the three-tank system (b). The simulated trajectories in (a) were generated from the model estimated by SINDy, and in (b) from the model estimated by I-SINDy. The legend in (a) applies to both figures.

# 4 Summary and Discussion

The results acquired with the synthetic datasets have shown, that weak form integral SINDy is significantly more robust against noise in the training data than standard SINDy. Even tho batchwise TRD was used, weak form I-SINDy showed a higher estimation accuracy at high noise levels. These results were observed for the Duffing and the Lotka-Volterra model, with weak form I-SINDy being able to estimate relatively accurate models at noise levels where SINDy was not able to correctly identify the structure of the model.

While I-SINDy also yielded better results for tank system data, none of both methods was able to estimate an applicable model. This result shows, that it may not always be possible to create the conditions, which are needed for the general SINDy framework to perform reliably. Especially the choice of candidate functions poses a significant limitation, since the structure and relevant terms of a model, that is to be identified, are not always known. Contributions, that provide an alternative to manually selecting the library terms, do therefore have a great potential for making the general SINDy framework more suitable for real world applications.

## References


[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 pp. 3932-3937. 2016. https://doi.org/10.1073/pnas.1517384113

[2] Hayden Schaeffer and Scott G. McCalla. “Sparse model selection via integral terms”. In: *Phys. Rev. E* 96.2:023302. 2017. https://doi.org/10.1103/PhysRevE.96.023302

[3] Alan A. Kaptanoglu et al.. “PySINDy: A comprehensive Python package for robust sparse system identification”. In: *Journal of Open Source Software* 7.69. 2022. https://doi.org/10.21105/joss.03994

[4] Rick Chartrand. “Numerical Differentiation of Noisy, Nonsmooth Data”. In: *International Scholarly Research Notices* 2011 pp. 1-11. 2011. https://api.semanticscholar.org/CorpusID:12989196

[5] Ian Knowles and Robert J. Renka. “Methods for numerical differentiation of noisy data”. In: *Electronic Journal of Differential Equations* Conference 21, pp. 235-246. 2014. https://ejde.math.txstate.edu/conf-proc/21/k3/knowles.pdf

[6] Abraham Savitzky and M. J. E. Golay. “Smoothing and Differentiation of Data by Simplified Least Squares Procedures”. In: *Analytical Chemistry* 36.8 pp. 1627-1639. 1964. https://doi.org/10.1021/ac60214a047

[7] N. M. Mangan, S. L. Brunton, J. L. Proctor and J. N. Kutz. “Inferring Biological Networks by Sparse Identification of Nonlinear Dynamics”. In: *IEEE Transactions on Molecular, Biological, and Multi-Scale Communications* 2.1 pp. 52-63. 2016. https://doi.org/10.1109/TMBMC.2016.2633265

[8] E. Kaiser, J. N. Kutz and S. L. Brunton. “Sparse identification of nonlinear dynamics for model predictive control in the low-data limit”. In: *Proc. R. Soc. A* 474:20180335. 2018. https://doi.org/10.1098/rspa.2018.0335

[9] Kadierdan Kaheman, J. Nathan Kutz and Steven L. Brunton. “SINDy-PI: a robust algorithm for parallel implicit sparse identification of nonlinear dynamics”. In: *Proc. R. Soc. A* 476:20200279. 2020. https://doi.org/10.1098/rspa.2020.0279

[10] Samuel H. Rudy, Steven L. Brunton, Joshua L. Proctor and J. Nathan Kutz. “Data-driven discovery of partial differential equations”. In: *Science Advances* 3.4. 2017. https://doi.org/10.1126/sciadv.1602614

[11] K. Egan, W. Li and R. Carvalho. “Automatically discovering ordinary differential equations from data with sparse regression”. In: *Commun. Phys.* 7:20. 2024. https://doi.org/10.1038/s42005-023-01516-2

[12] Jane Cullum. “Numerical Differentiation and Regularization”. In: *SIAM Journal on Numerical Analysis* 8.2 pp. 254-265. 1971. https://doi.org/10.1137/0708026

[13] Patrick L. Combettes and Jean-Christophe Pesquet. “Proximal Splitting Methods in Signal Processing”. In: *Fixed-Point Algorithms for Inverse Problems in Science and Engineering* (Springer New York) pp. 185-212. 2011. https://doi.org/10.1007/978-1-4419-9569-8_10

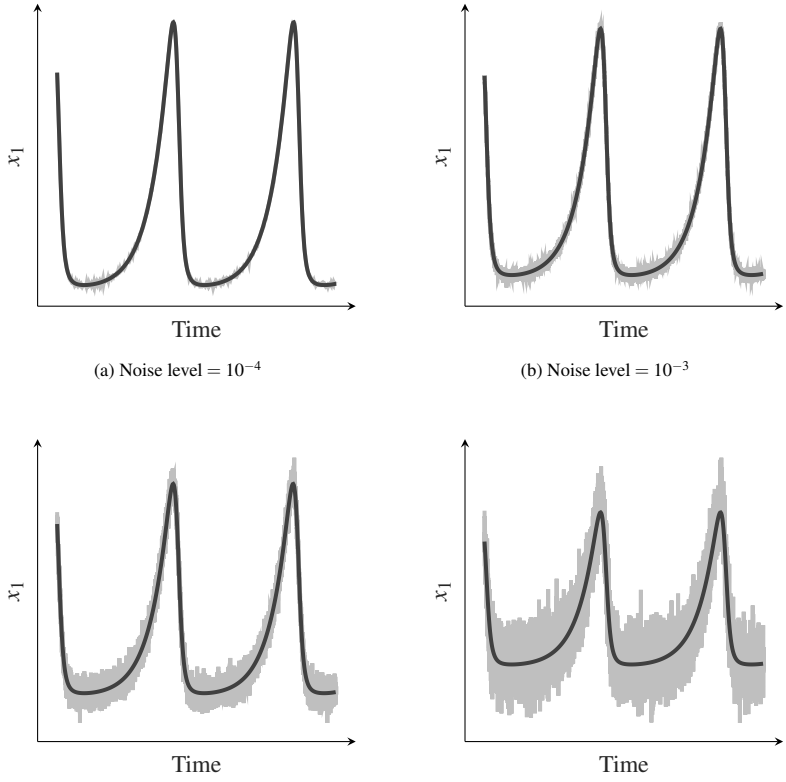
[14] Robert Tibshirani. “Regression Shrinkage and Selection via the Lasso”. In: *Journal of the Royal Statistical Society. Series B (Methodological)* 58.1 pp. 267-288. 1996. https://www.jstor.org/stable/2346178

[15] Yifei Guan, Steven L. Brunton and Igor Novosselov. “Sparse nonlinear models of chaotic electroconvection”. In: *R. Soc. Open Sci* 8:202367. 2021. https://doi.org/10.1098/rsos.202367

[16] Peng Zheng et al.. “A Unified Framework for Sparse Relaxed Regularized Regression: SR3”. In: *IEEE Access* 7 pp. 1404-1423. 2019. https://doi.org/10.1109/ACCESS.2018.2886528

# Appendix: Supplementary Figures for Section 3

Appendix



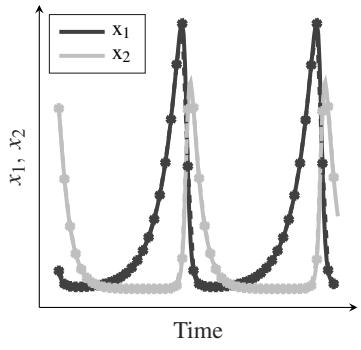

(a) Noise level $= 10^{-4}$

(b) Noise level $= 10^{-3}$

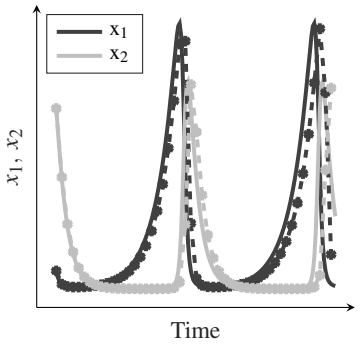
(c) Noise level $= 10^{-2}$

(d) Noise level $= 10^{-1}$

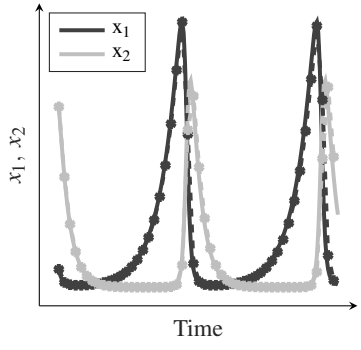
Figure 3: Data corresponding to state variable $x_1$ from the Lotka-Volterra model. The initially generated simulation data is shown in black and the added noise in light gray. The sum of both signals is used as training data.



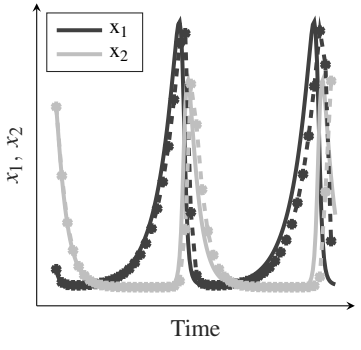

(a) I-SINDy, noise level = $10^{-2}$




(b) I-SINDy, noise level = $10^{-1}$

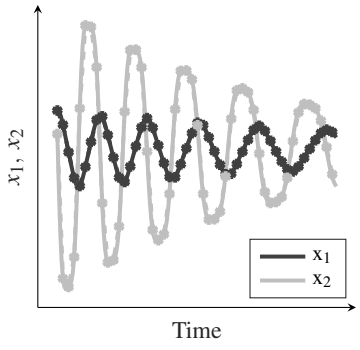



(c) SINDy, noise level = $10^{-4}$

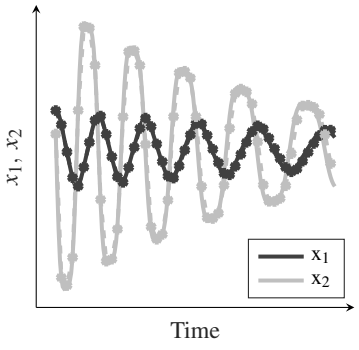



(d) SINDy, noise level = $10^{-3}$

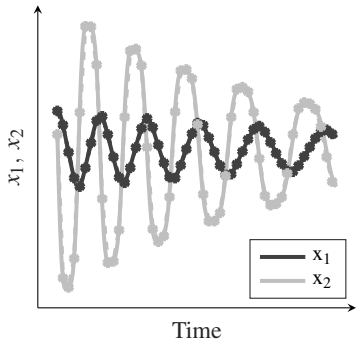
Figure 4: Test data (solid) and simulated trajectories (star-dash) for the Lotka-Volterra models, that where estimated using I-SINDy (a and b) and SINDy (c and d). Figure (a) is representative of the model accuracy with the noise levels $10^{-4}$, $10^{-3}$, and $10^{-2}$, as there was no significant visible difference between the corresponding model trajectories. Figure (b) shows the model accuracy with a noise level of $10^{-1}$. Figures (c) and (d) show the model accuracy for the noise levels $10^{-4}$ and $10^{-3}$, respectively.



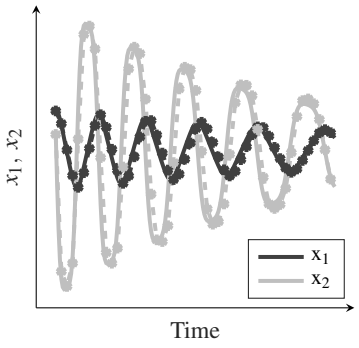

(a) I-SINDy, noise level = $10^{-4}$




(b) I-SINDy, noise level = $10^{-2}$




(c) SINDy, noise level = $10^{-3}$




(d) SINDy, noise level = $10^{-3}$

Figure 5: Training data (solid) and simulated trajectories (star-dash) for the Duffing models that where estimated using I-SINDy (a and b) and SINDy (c and d). The training data is shown because the models performed worse compared against the training data than against the test data (see tables 3 through 6). Figure (a) is representative of the model accuracy with the noise levels $10^{-4}$ and $10^{-3}$, as there was no visible difference between these model trajectories. Figure (b) shows the model accuracy with a noise level of $10^{-2}$. Figure (c) is representative of the model accuracy with the noise levels $10^{-4}$ and $10^{-3}$. Figure (d) shows the model accuracy for the noise level $10^{-2}$.