Exploring the Resilience of SINDy Algorithms to Limited Noisy Data

Written By:
Elliot Trilling

Advisor:
Professor Andrea Arnold

December 2024

A Master's Capstone Project
Worcester Polytechnic Institute

Submitted to the Faculty of Worcester Polytechnic Institute in partial fulfillment of the requirements for the Degree of Master of Science in Applied Mathematics.

Abstract

We analyze the stability of the SINDy algorithm and Ensemble SINDy under various combinations of noise and limited data. Specifically, we examine the relationships between the level of noise added to the data, the time between data points, and the time span over which training data is collected. We assess the performance of different models on a suite of common test problems. Additionally, we investigate how two different derivative approximations perform when applied to E-SINDy.

Acknowledgments

I would like to thank Professor Andrea Arnold for her insight and support on this project. Her feedback and guidance made this project possible.

Contents

List of Tables

List of Figures

1 Introduction

One of the most common problems in science and engineering is determining governing equations from data collected through experiments or observations. Recently, sparse identification of nonlinear dynamics (SINDy) has shown great promise in this area [1]. SINDy seeks to determine the right-hand side (RHS) of a differential equation of the form

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t)).$$

It accomplishes this by finding a sparse representation for $\boldsymbol{f}$ in terms of a library of candidate functions, which can be chosen with nearly complete freedom. The requirement for sparsity is motivated by the observation that most physical systems can be described by differential equations containing only a few terms [1]. Moreover, sparse models are often more interpretable and potentially more generalizable than black-box models like neural networks.

While this report focuses on SINDy, it represents only the tip of the iceberg in the data-driven discovery of governing equations. Another popular method, symbolic regression via genetic algorithms, seeks to find a governing equation for the data by composing simple functions to build more complex behavior [2]. Genetic algorithms, which mimic real-world effects like natural selection and gene mutation, power these methods to discover models that fit the data. Although this approach can handle a wide range of dynamics, it often incurs significant computational costs. Despite the variety of available techniques for data-driven model discovery, SINDy remains attractive due to its conceptual simplicity, computational efficiency, and the interpretability of its results. Since its introduction, researchers have applied SINDy and its variants to a broad range of scientific and engineering problems. For example, they have used it to rediscover foundational equations in biological networks, such as the Michaelis-Menten enzyme kinetics [3]. Other examples include discovering new orbital mechanics and detecting the locations of thermal updrafts in unmanned aerial vehicles [4, 5]. The modular nature of SINDy allows researchers to integrate new library functions,

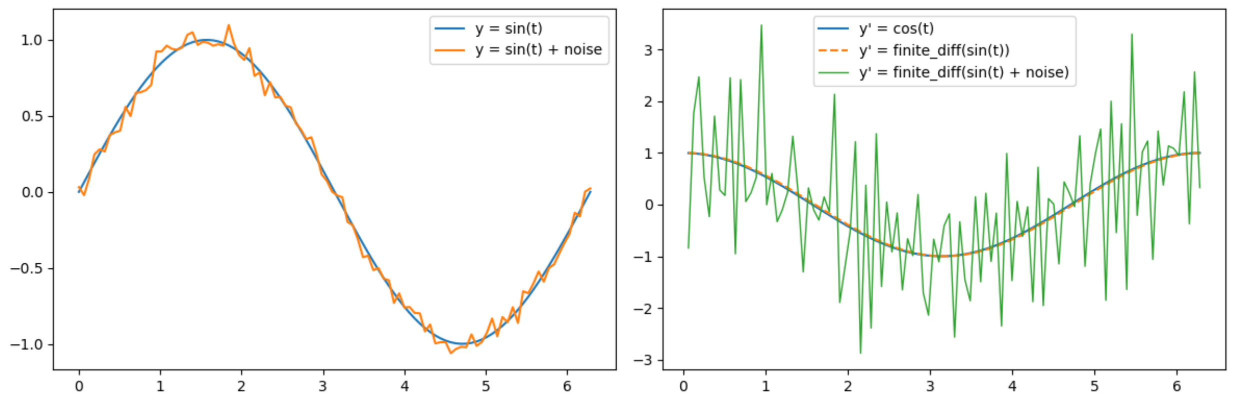

Figure 1: A toy example illustrating how even tiny amounts of noise in $\boldsymbol{x}(t)$ lead to very poor estimates of $\dot{\boldsymbol{x}}(t)$. This example uses finite differences to approximate $\dot{\boldsymbol{x}}(t)$.

regularization strategies, and optimization techniques, continually inspiring new extensions and improvements.

One of the largest limitations of SINDy is its assumption that the derivative $\dot{\boldsymbol{x}}(t)$ can be accurately computed. Sometimes researchers can measure $\dot{\boldsymbol{x}}(t)$ directly at discrete time points, but more often only discrete measurements of $\boldsymbol{x}(t)$ are recorded, requiring $\dot{\boldsymbol{x}}(t)$ to be computed numerically. For the remainder of this paper, we assume the latter case. This creates a significant challenge because even small amounts of noise in the measured values of $\boldsymbol{x}(t)$ can produce completely inaccurate estimates of $\dot{\boldsymbol{x}}(t)$. This issue is particularly pronounced when $\dot{\boldsymbol{x}}(t)$ is computed naively using finite differences, although it causes problems regardless of the chosen algorithm. Figure 1 provides a clear visual example. Many variations of the basic SINDy algorithm have been developed and documented in recent years to address this issue in various ways. A few of the more popular methods include the following:

- Ensemble SINDy (E-SINDy): Uses bootstrapping to generate an ensemble of models that are generally more tolerant to noise and limited data than vanilla SINDy [6].

- Weak SINDy: Employs a weak formulation of SINDy, using tools from functional analysis to eliminate the need for explicit derivative computation [7]. This approach is particularly useful for partial differential equations, where computing higher-order derivatives from noisy data using finite difference algorithms often yields unusable results [8].

- DeepMoD: Trains a neural network to approximate $\boldsymbol{x}(t)$ and leverages auto-differentiation to approximate $\dot{\boldsymbol{x}}(t)$ [9].
- SINDy with autoencoders: Combines SINDy with autoencoder neural networks to simultaneously learn coordinates and dynamics [10].

Other research focuses on employing different derivative approximations and/or data pre-processing techniques to produce better estimates of $\dot{\boldsymbol{x}}(t)$ that are less sensitive to noise [11, 12].

1.1 Project Goals and Contributions

One common assumption in papers discussing SINDy is the availability of a nearly unlimited amount of finely spaced data. While this assumption may be reasonable in some situations, many experimental setups do not or cannot provide this luxury. For example, in many biological experiments, gathering additional data points can be prohibitively costly in terms of both time and money. This raises important questions about how much data is truly needed to produce accurate results and how SINDy performs in data-limited scenarios.

The goal of this project is to evaluate SINDy and Ensemble SINDy (E-SINDy) for their resilience to various kinds of data limitations. We focus specifically on SINDy and E-SINDy, as they form the backbone of many other SINDy-based models. In particular, we aim to address the following questions:

- How resilient are SINDy and E-SINDy to noise as the time between data points increases?
- How resilient are SINDy and E-SINDy to noise when trained on only a portion of a trajectory?

- With a fixed level of noise, what is the relationship between the span of time over which training data is collected and the time between training data points?

We will evaluate these questions using a suite of test problems frequently referenced in the SINDy literature, providing a diverse set of behaviors to analyze [7].

2 Background

In the following section, we discuss the theoretical underpinnings of SINDy and E-SINDy, as well as our process for evaluating (E)SINDy models and quantifying noise.

2.1 The SINDy Algorithm

The SINDy algorithm was first introduced in 2016 by Brunton, Proctor, and Kutz [1]. While many variations of SINDy have since been proposed and researched, most follow the same core principles as originally described.

Suppose we have a dynamical system of the form

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t)), \quad \boldsymbol{x}(0) = \boldsymbol{x}_0 \in \mathbb{R}^n.$$

Here, $\dot{x}_i(t) = f_i(\boldsymbol{x}(t))$ for $1 \leq i \leq n$. We collect m data points from the system at times $t_1, t_2, \ldots, t_m$, assuming these data points take the form

$$\boldsymbol{y}(t_i) = \boldsymbol{x}(t_i) + \boldsymbol{\epsilon}(t_i),$$

where $\boldsymbol{x}(t_i)$ represents the true underlying state, and $\boldsymbol{\epsilon}(t_i)$ is normally distributed noise with a mean of zero and constant variance.

We further assume that $f_i(\boldsymbol{x}(t))$ can be expressed as a linear combination of library functions (also known as dictionary functions) $\theta_1(\boldsymbol{x}(t)), \theta_2(\boldsymbol{x}(t)), \ldots, \theta_p(\boldsymbol{x}(t))$. Specifically,

$$f_i(\boldsymbol{x}(t)) = \xi_{i1}\theta_1(\boldsymbol{x}(t)) + \xi_{i2}\theta_2(\boldsymbol{x}(t)) + \cdots + \xi_{ip}\theta_p(\boldsymbol{x}(t))$$

for some set of coefficients ξ_{ij}. Each library function can be a nonlinear function of the system's states. So, written out fully, $\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t))$ becomes

$$\begin{aligned}
\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \\ \vdots \\ \dot{x}_n(t) \end{bmatrix} &= \begin{bmatrix} f_1(\boldsymbol{x}(t)) \\ f_2(\boldsymbol{x}(t)) \\ \vdots \\ f_n(\boldsymbol{x}(t)) \end{bmatrix} \\
&= \begin{bmatrix} \xi_{11}\theta_1(\boldsymbol{x}(t)) + \xi_{12}\theta_2(\boldsymbol{x}(t)) + \cdots + \xi_{1p}\theta_p(\boldsymbol{x}(t)) \\ \xi_{21}\theta_1(\boldsymbol{x}(t)) + \xi_{22}\theta_2(\boldsymbol{x}(t)) + \cdots + \xi_{2p}\theta_p(\boldsymbol{x}(t)) \\ \vdots \\ \xi_{n1}\theta_1(\boldsymbol{x}(t)) + \xi_{n2}\theta_2(\boldsymbol{x}(t)) + \cdots + \xi_{np}\theta_p(\boldsymbol{x}(t)) \end{bmatrix} \\
&= \begin{bmatrix} \xi_{11} & \xi_{12} & \cdots & \xi_{1p} \\ \xi_{21} & \xi_{22} & \cdots & \xi_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ \xi_{n1} & \xi_{n2} & \cdots & \xi_{np} \end{bmatrix} \begin{bmatrix} \theta_1(\boldsymbol{x}(t)) \\ \theta_2(\boldsymbol{x}(t)) \\ \vdots \\ \theta_p(\boldsymbol{x}(t)) \end{bmatrix}.
\end{aligned}$$

By transposing each side we get

$$\begin{bmatrix} \dot{x}_1(t) & \dot{x}_2(t) & \cdots & \dot{x}_n(t) \end{bmatrix} = \begin{bmatrix} \theta_1(\boldsymbol{x}(t)) & \theta_2(\boldsymbol{x}(t)) & \cdots & \theta_p(\boldsymbol{x}(t)) \end{bmatrix} \begin{bmatrix} \xi_{11} & \xi_{21} & \cdots & \xi_{n1} \\ \xi_{12} & \xi_{22} & \cdots & \xi_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ \xi_{1p} & \xi_{2n} & \cdots & \xi_{np} \end{bmatrix}.$$

Because this relationship holds for all values of t, we can evaluate both sides at times $t_1, t_2, ..., t_m$ and concatenate together rows to get

$$\begin{bmatrix} \dot{x}_1(t_1) & \dot{x}_2(t_1) & \cdots & \dot{x}_n(t_1) \\ \dot{x}_1(t_2) & \dot{x}_2(t_2) & \cdots & \dot{x}_n(t_2) \\ \vdots & \vdots & \ddots & \vdots \\ \dot{x}_1(t_m) & \dot{x}_2(t_m) & \cdots & \dot{x}_n(t_m) \end{bmatrix} = \begin{bmatrix} \theta_1(\boldsymbol{x}(t_1)) & \theta_2(\boldsymbol{x}(t_1)) & \cdots & \theta_p(\boldsymbol{x}(t_1)) \\ \theta_1(\boldsymbol{x}(t_2)) & \theta_2(\boldsymbol{x}(t_2)) & \cdots & \theta_p(\boldsymbol{x}(t_2)) \\ \vdots & \vdots & \ddots & \vdots \\ \theta_1(\boldsymbol{x}(t_m)) & \theta_2(\boldsymbol{x}(t_m)) & \cdots & \theta_p(\boldsymbol{x}(t_m)) \end{bmatrix} \begin{bmatrix} \xi_{11} & \xi_{21} & \cdots & \xi_{n1} \\ \xi_{12} & \xi_{22} & \cdots & \xi_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ \xi_{1p} & \xi_{2n} & \cdots & \xi_{np} \end{bmatrix}.$$

Of course, in practice, we cannot measure $\boldsymbol{x}(t)$ directly and must approximate it using $\boldsymbol{y}(t)$. Furthermore, we approximate $\dot{\boldsymbol{x}}(t)$ using $\dot{\boldsymbol{y}}(t)$, which we compute from $\boldsymbol{y}(t)$. This approximation becomes very challenging, even with small amounts of noise. Using these approximations, we can rewrite the previous system as

$$\begin{bmatrix} \dot{y}_1(t_1) & \dot{y}_2(t_1) & \cdots & \dot{y}_n(t_1) \\ \dot{y}_1(t_2) & \dot{y}_2(t_2) & \cdots & \dot{y}_n(t_2) \\ \vdots & \vdots & \ddots & \vdots \\ \dot{y}_1(t_m) & \dot{y}_2(t_m) & \cdots & \dot{y}_n(t_m) \end{bmatrix} \approx \begin{bmatrix} \theta_1(\boldsymbol{y}(t_1)) & \theta_2(\boldsymbol{y}(t_1)) & \cdots & \theta_p(\boldsymbol{y}(t_1)) \\ \theta_1(\boldsymbol{y}(t_2)) & \theta_2(\boldsymbol{y}(t_2)) & \cdots & \theta_p(\boldsymbol{y}(t_2)) \\ \vdots & \vdots & \ddots & \vdots \\ \theta_1(\boldsymbol{y}(t_m)) & \theta_2(\boldsymbol{y}(t_m)) & \cdots & \theta_p(\boldsymbol{y}(t_m)) \end{bmatrix} \begin{bmatrix} \xi_{11} & \xi_{21} & \cdots & \xi_{n1} \\ \xi_{12} & \xi_{22} & \cdots & \xi_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ \xi_{1p} & \xi_{2n} & \cdots & \xi_{np} \end{bmatrix}.$$

Let's collect all our data into four matrices to simplify our problem. We write our position measurements as

$$\boldsymbol{Y} = \begin{bmatrix} \boldsymbol{y}^T(t_1) \\ \boldsymbol{y}^T(t_2) \\ \vdots \\ \boldsymbol{y}^T(t_m) \end{bmatrix} = \begin{bmatrix} y_1(t_1) & y_2(t_1) & \cdots & y_n(t_1) \\ y_1(t_2) & y_2(t_2) & \cdots & y_n(t_2) \\ \vdots & \vdots & \ddots & \vdots \\ y_1(t_m) & y_2(t_m) & \cdots & y_n(t_m) \end{bmatrix},$$

our derivative measurements as

$$\dot{\boldsymbol{Y}} = \begin{bmatrix} \dot{\boldsymbol{y}}^T(t_1) \\ \dot{\boldsymbol{y}}^T(t_2) \\ \vdots \\ \dot{\boldsymbol{y}}^T(t_m) \end{bmatrix} = \begin{bmatrix} \dot{y}_1(t_1) & \dot{y}_2(t_1) & \cdots & \dot{y}_n(t_1) \\ \dot{y}_1(t_2) & \dot{y}_2(t_2) & \cdots & \dot{y}_n(t_2) \\ \vdots & \vdots & \ddots & \vdots \\ \dot{y}_1(t_m) & \dot{y}_2(t_m) & \cdots & \dot{y}_n(t_m) \end{bmatrix},$$

our library of terms as

$$\boldsymbol{\Theta}(\boldsymbol{Y}) = \begin{bmatrix} \boldsymbol{\Theta}(\boldsymbol{y}^T(t_1)) \\ \boldsymbol{\Theta}(\boldsymbol{y}^T(t_2)) \\ \vdots \\ \boldsymbol{\Theta}(\boldsymbol{y}^T(t_m)) \end{bmatrix} = \begin{bmatrix} \theta_1(\boldsymbol{y}(t_1)) & \theta_2(\boldsymbol{y}(t_1)) & \cdots & \theta_p(\boldsymbol{y}(t_1)) \\ \theta_1(\boldsymbol{y}(t_2)) & \theta_2(\boldsymbol{y}(t_2)) & \cdots & \theta_p(\boldsymbol{y}(t_2)) \\ \vdots & \vdots & \ddots & \vdots \\ \theta_1(\boldsymbol{y}(t_m)) & \theta_2(\boldsymbol{y}(t_m)) & \cdots & \theta_p(\boldsymbol{y}(t_m)) \end{bmatrix},$$

and, lastly, our coefficients as

$$\boldsymbol{\Xi} = \begin{bmatrix} \xi_{11} & \xi_{21} & \cdots & \xi_{n1} \\ \xi_{12} & \xi_{22} & \cdots & \xi_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ \xi_{1p} & \xi_{2n} & \cdots & \xi_{np} \end{bmatrix}.$$

Thus, we can rewrite our problem succinctly as

$$\dot{\boldsymbol{Y}} \approx \boldsymbol{\Theta}(\boldsymbol{Y})\boldsymbol{\Xi}.$$

Due to measurement noise, in practice, $\dot{\boldsymbol{Y}} \approx \boldsymbol{\Theta}(\boldsymbol{Y})\boldsymbol{\Xi}$ will form an inconsistent system of equations. Our goal is to find a sparse matrix $\boldsymbol{\Xi}$ that approximately solves this system. More precisely, we aim to solve the minimization problem:

$$\boldsymbol{\Xi} = \underset{\hat{\boldsymbol{\Xi}}}{\operatorname{argmin}} \left(\left\| \dot{\boldsymbol{Y}} - \boldsymbol{\Theta}(\boldsymbol{Y})\hat{\boldsymbol{\Xi}} \right\|_2^2 + R(\hat{\boldsymbol{\Xi}}, \boldsymbol{\omega}) \right),$$

where R is a regularization function that promotes sparsity in $\boldsymbol{\Xi}$, and $\boldsymbol{\omega}$ is a vector of regularization parameters. There are multiple ways to approach this problem. The method proposed in the original SINDy paper by Brunton, Proctor, and Kutz is known as Sequentially Thresholded Least Squares (STLSQ). This method begins by using a standard least-squares approach to solve $\dot{\boldsymbol{Y}} \approx \boldsymbol{\Theta}(\boldsymbol{Y})\boldsymbol{\Xi}$ for $\boldsymbol{\Xi}$. It then applies a step to remove all coefficients below a chosen threshold, λ. By iteratively applying least squares and thresholding to the remaining non-zero coefficients, this algorithm quickly converges to a sparse solution.

One common improvement to this algorithm is replacing the standard least-squares step with ridge regression (also known as Tikhonov regularization). In this case, $R(\Xi, \boldsymbol{\omega})$ takes the form $\alpha \|\Xi\|_2^2$, where α is a regularization parameter. This modification encourages solutions in which the values of Ξ remain small [13].

2.2 Ensemble SINDy

Ensemble SINDy builds on the basic SINDy algorithm to produce results that are generally more tolerant to noise and smaller amounts of data [6]. It achieves this by bootstrapping either the data or the library variables and aggregating the resulting models. This technique, originating from the field of statistics, is colloquially known as bootstrap aggregating (bagging) [14]. For our tests, we will focus on bootstrapping the training data, as it provides the simplest approach to generating more robust and accurate results. The E-SINDy algorithm follows a straightforward set of steps [6]:

- Choose the number of bootstraps. For each bootstrap, sample the original dataset with replacement to create a new dataset of the same size.
- Train a SINDy model on each bootstrapped dataset.
- Build an aggregate SINDy model where each library function coefficient is the mean (or median) value across all bootstrapped SINDy models.
- For each library function, calculate its inclusion probability, defined as the proportion of bootstrapped SINDy models in which it appears. In the aggregate model, set any coefficients to zero if their inclusion probability is below a predefined threshold (distinct from the STLSQ threshold).

2.3 Quantifying SINDy Error

Quantifying the error in a SINDy model is essential for rigorous evaluation of model performance. The literature frequently uses three main approaches to evaluate model performance [7, 11, 6].

Firstly, we may be interested in whether the predicted model and the true model have the same set of non-zero coefficients. If they do, we can say that SINDy successfully predicted the correct form of the governing equations. However, this does not address the accuracy of the predicted coefficients.

A second clear metric of interest is the relative error of the model coefficients. Suppose the true coefficients for the RHS of an ODE are given by $\boldsymbol{\Xi}^*$. For a dynamical system governed by $\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t))$, we have $f(\boldsymbol{x}(t)) = \boldsymbol{\Theta}(\boldsymbol{x}(t))\boldsymbol{\Xi}^*$. If SINDy predicts the coefficient matrix to be $\hat{\boldsymbol{\Xi}}$, we define the relative Frobenius norm coefficient error as

$$E_{\Xi} = \frac{\left\| \hat{\boldsymbol{\Xi}} - \boldsymbol{\Xi}^* \right\|_F}{\left\| \boldsymbol{\Xi}^* \right\|_F}$$

where $\|\cdot\|_F$ is the Frobenius norm, computed as

$$\|A\|_F = \sqrt{\sum_j \sum_i A_{ij}}.$$

The third approach measures the relative error in the trajectories generated by the governing equation discovered by SINDy. In this case, the error is defined as

$$E_X = \frac{\left\| \hat{X} - X^* \right\|_F}{\left\| X^* \right\|_F}$$

where X^* is a trajectory generated by the true governing equation $\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t))$ sampled at time points $t_1, t_2, ..., t_m$

$$X^* = \begin{bmatrix} x_1(t_1) & x_2(t_1) & \cdots & x_n(t_1) \\ x_1(t_2) & x_2(t_2) & \cdots & x_n(t_2) \\ \vdots & \vdots & \ddots & \vdots \\ x_1(t_m) & x_2(t_m) & \cdots & x_n(t_m) \end{bmatrix}$$

and $\hat{X}$ is the trajectory generated by $\dot{\boldsymbol{x}}(t) = \boldsymbol{\Theta}(\boldsymbol{x}(t))\hat{\boldsymbol{\Xi}}$ sampled at the same time points.

Each method has its pros and cons. Since the goal of SINDy is to generate governing equations, one could argue that unless SINDy matches the true form of the governing equations (i.e., the same set of non-zero terms), it has failed. However, this ignores cases where SINDy achieves very low errors in the other two metrics (coefficient and trajectory error) while failing to find the exact system. This often occurs when the threshold is set too low, resulting in the selection of an extra term with a small coefficient. Alternatively, SINDy may identify a very different system that produces similar dynamics.

Comparing model coefficients makes more sense for chaotic systems, where small errors compound quickly to create large trajectory errors. Conversely, comparing trajectory errors is more appropriate for evaluating how well SINDy predicts future behavior. Naturally, all three error metrics require knowledge of the true underlying system to be computed.

2.4 Quantifying Noise

When generating training data, it is crucial to have a reliable method to quantify the amount of noise added. While there are various ways to quantify noise, we adopt a simple method here. Suppose our measured data takes the form:

$$y_i(t_j) = x_i(t_j) + \epsilon_i(t_j).$$

Here, $y_i(t_j)$ represents the noisy measurement of the i-th state variable at time t_j. We assume that the noise term ϵ_i is independently and identically distributed Gaussian noise for each sample. Specifically, ϵ_i is normally distributed with a mean of 0 and variance σ_i^2.

We quantify the noise added to the j-th state variable using the noise ratio η, which we use to define σ_i as:

$$\sigma_i = \eta \cdot \sqrt{\frac{1}{n} \sum_j x_i(t_j)^2}.$$

An important aspect to note is the assumption that noise may have different variances for each state variable. This decision reflects the possibility that state variables can vary significantly in scale, making it reasonable to expect noise to reflect these differences. In contrast, many papers assume a uniform variance across all state variables, regardless of scale.

3 Methods and Data

In the following section, we discuss the details of our training and testing procedures. All computational work was performed in Python using a collection of common numerical libraries, including `numpy` and `scipy` [15, 16]. The plots in this paper were generated using the `matplotlib` and `seaborn` Python libraries, which provide a high level of control over plot details [17, 18].

3.1 ODEs Tested

We selected a collection of ODEs that are popular in the SINDy literature [7, 11], including the following:

- Duffing Oscillator: A damped harmonic oscillator with an additional nonlinear x^3 term [19].
- Van der Pol Oscillator: A damped oscillator with a nonlinear damping term [20].
- Lotka–Volterra: Models the relationships between predator and prey species [20].
- Nonlinear Pendulum: Models the position and velocity of a pendulum [20].
- Lorenz System: A simplified model for atmospheric convection that exhibits chaotic behavior under certain parameters [20].

This collection of systems exhibits a wide range of behaviors to explore, including oscillatory, damped, chaotic, and non-chaotic dynamics. For each model, we select parameters and initial conditions that produce a representative range of behaviors. These are listed below in Table 1. In Figure 2, we display the trajectories for the state variables of each system specified in Table 1.

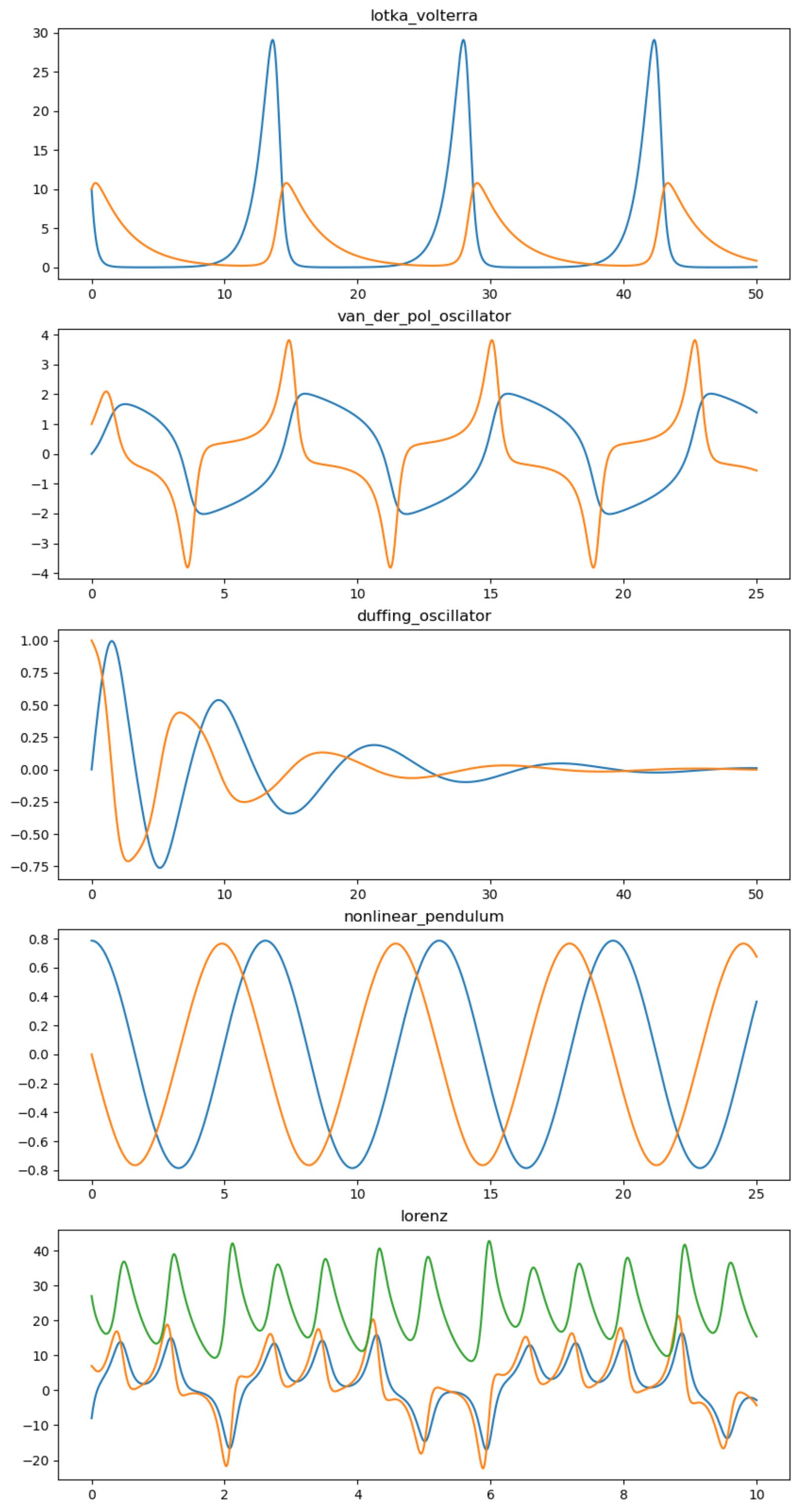

Figure 2: Trajectories for the state variables of each system, using the parameters and initial conditions specified in Table 1.

System Name	Governing Equation	Parameter Values	Initial Condition
Duffing Oscillator (no driving term)	$\dot{x}_1 = x_2$ $\dot{x}_2 = -\delta x_2 - \alpha x_1 - \beta x_1^3$	$\delta = 0.2, \alpha = 0.2$ $\beta = 1$	$(0, 1)^T$
Van der Pol Oscillator	$\dot{x}_1 = x_2$ $\dot{x}_2 = \mu(1 - x_1^2)x_2 - x_1$	$\mu = 2$	$(0, 1)^T$
Lotka–Volterra	$\dot{x}_1 = \alpha x_1 - \beta x_1 x_2$ $\dot{x}_2 = -\gamma x_2 + \delta x_1 x_2$	$\alpha = 1.1, \beta = 0.4$ $\gamma = 0.4, \delta = 0.1$	$(10, 10)^T$
Nonlinear Pendulum	$\dot{x}_1 = x_2$ $\dot{x}_2 = -\omega^2 \sin(x_1)$	$\omega^2 = 1$	$(\frac{\pi}{4}, 0)^T$
Lorenz System	$\dot{x}_1 = \sigma(x_2 - x_1)$ $\dot{x}_2 = x_1(\rho - x_3) - x_2$ $\dot{x}_3 = x_1 x_2 - \beta x_3$	$\sigma = 10, \rho = 28$ $\beta = \frac{8}{3}$	$(-8, 7, 27)^T$

Table 1: Explicit form of each dynamical system tested along with chosen parameter values and initial conditions.

3.2 Generating Ground Truth Data

To generate the ground truth data for training, we used the `solve_ivp` function in SciPy with the LSODA method [16]. LSODA employs automatic stiffness detection to switch between a non-stiff solver and a stiff solver. We also ran tests with the more common RK45 solver and found that it produced very similar results to LSODA but took significantly longer to run.

By default, `solve_ivp` returns a set of evaluation points chosen by the solver. However, we can specify our own set of time points at which the solution should be evaluated. We use this feature in all of our tests.

3.3 Approximating Derivatives

We implemented and evaluated two different methods to approximate derivatives from our data. The first was a simple finite difference method. The second used a sliding window approach to fit polynomials to local subsets of data and approximate the derivative using the polynomial's derivative. This technique, often called a Savitzky-Golay filter, is one of the most common algorithms for generating derivatives from noisy data [21].

3.3.1 Finite Difference

In the simple case where data points are equally spaced, we can use classical forward, central, and backward finite difference approximations [22]. For points that are not at the endpoints of the time range, we use the centered finite difference equation:

$$f'(x_i) \approx \frac{f(x_{i+1}) - f(x_{i-1})}{2\Delta t}$$

where Δt is the time between data points. For the left endpoint of the range, we use the forward difference formula:

$$f'(x_i) \approx \frac{f(x_{i+1}) - f(x_i)}{\Delta t}.$$

For the right endpoint of the range, we use the backward difference formula:

$$f'(x_i) \approx \frac{f(x_i) - f(x_{i-1})}{\Delta t}.$$

We implemented the finite difference method using the following Python code:

```python
def dxdt_finite_difference_1d(x, t):
    dxdt = np.zeros_like(x)

    dxdt[0] = (x[1] - x[0]) / (t[1] - t[0])

    dt1 = t[1:-1] - t[:-2]      # t_i - t_{i-1}
    dt2 = t[2:] - t[1:-1]       # t_{i+1} - t_i
    dx1 = x[1:-1] - x[:-2]      # x_i - x_{i-1}
    dx2 = x[2:] - x[1:-1]       # x_{i+1} - x_i

    denom = dt1 + dt2
    a = dt2 / denom
    b = dt1 / denom

    dxdt[1:-1] = a * (dx1 / dt1) + b * (dx2 / dt2)

    dxdt[-1] = (x[-1] - x[-2]) / (t[-1] - t[-2])

    return dxdt
```

3.3.2 Sliding Polynomial Approximation

Suppose we wish to perform a sliding polynomial approximation using polynomials with degree k and window diameter d. For points that are not near the endpoints of the time range, we use the formula:

$$f'(x_i) \approx p_i'(x_i),$$

where $p_i(x)$ is the best kth-order polynomial approximation for the set:

$$\{(t_{i-d}, \boldsymbol{x}_{i-d}), (t_{i-d+1}, \boldsymbol{x}_{i-d+1}), \ldots, (t_i, \boldsymbol{x}_i), (t_{i+1}, \boldsymbol{x}_{i+1}), \ldots, (t_{i+d}, \boldsymbol{x}_{i+d})\}.$$

For points near the endpoints of the time range, we simply remove points from the above set to ensure that the entire set is contained within the training data. We implemented the sliding window polynomial derivative approximation using the following Python code:

```
def dxdt_poly_fit_1d(x, t, poly_deg, window_diameter):
    dxdt = np.zeros_like(x)

    for i in range(len(x)):
        if i < window_diameter:
            start = 0
            end = i + window_diameter + 1
        elif i + window_diameter >= len(x):
            start = i - window_diameter
            end = len(x)
        else:
            start = i - window_diameter
            end = i + window_diameter + 1

        p = np.polyfit(t[start:end], x[start:end], poly_deg)

        dxdt[i] = np.polyval(np.polyder(p), t[i])

    return dxdt
```

3.4 Generating Training Data

To generate the training data for a given initial value problem, we can modify three types of parameters. The first is the set of time points at which we want to generate a solution. The second is the level of noise to add to the data, as described in Section 2.4. Lastly, we can select the derivative approximation algorithm to use.

The process for generating training data proceeds as follows. First, we generate ground truth data at the specified evaluation time points. Next, we add noise to the data points at the specified level. Finally, we compute derivative approximations on the noisy data using either a finite difference method or a polynomial approximation. Figure 3 illustrates examples of varying levels of noise added to a Lotka-Volterra trajectory, along with finite difference approximations and polynomial derivative approximations.

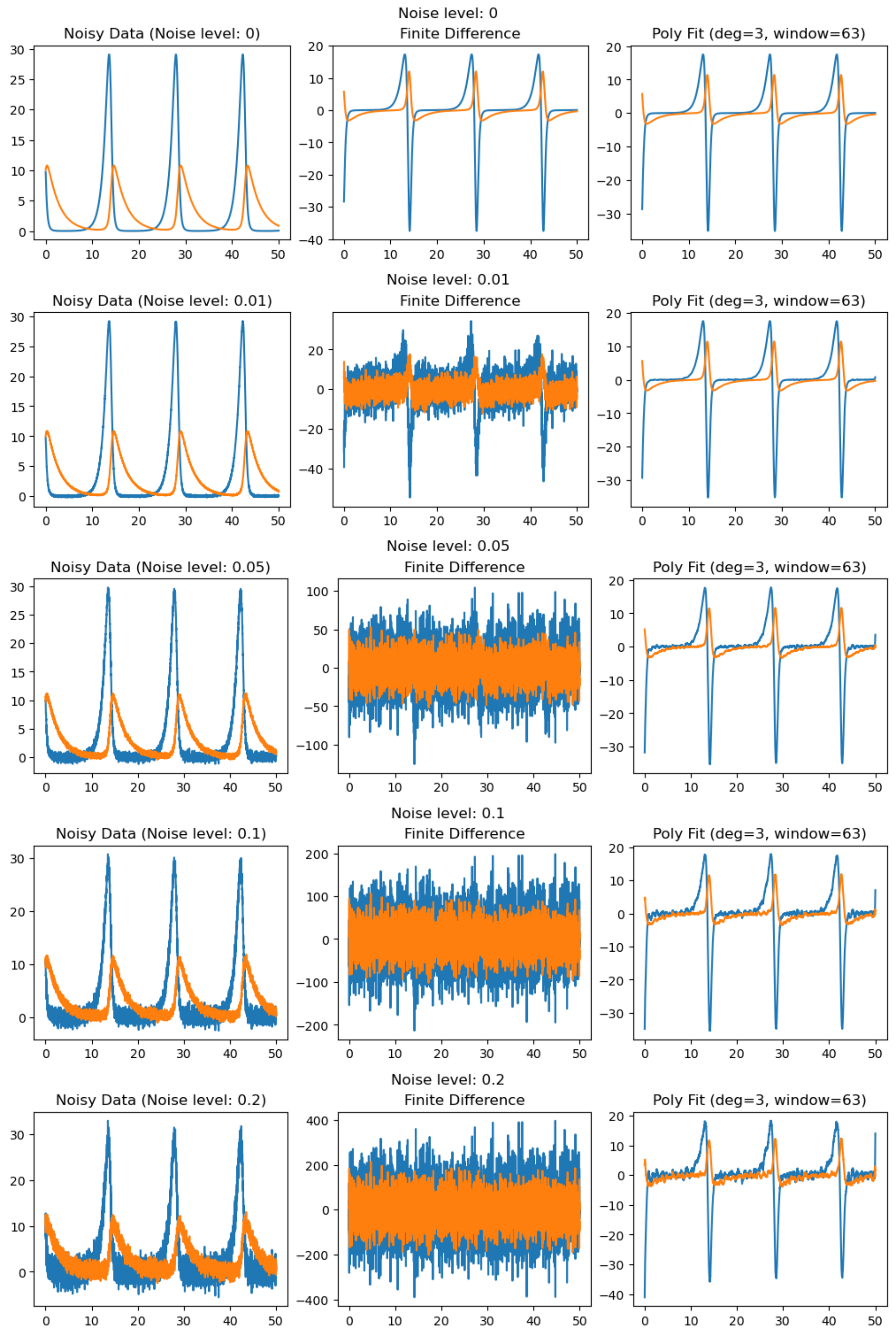

Figure 3: Trajectories of each state variable (x_1 in blue, x_2 in orange) for the Lotka-Volterra system with varying levels of noise added (left column), finite difference approximations (middle column), and polynomial approximations (right column).

3.5 Implementing SINDy and E-SINDy

While PySINDy is a popular open-source implementation of many SINDy algorithms [23, 24], we chose to implement our own versions of SINDy and E-SINDy for sake of this work. Using the NumPy Python library, we implemented the algorithms described in the original SINDy paper and the subsequent E-SINDy paper [15, 1, 6]. For our implementation of STLSQ, we utilized the `ridge_regression` function from `sklearn` [25].

We selected the default set of library functions to be polynomials of the state variables up to degree 5. This default was used in all tests except for the nonlinear pendulum, where we employed sinusoidal functions of the form $\sin(nx)$ and $\cos(nx)$ for $n = 1, 2, 3, 4, 5$. All tests used a threshold parameter value of $\lambda = 0.05$. While other thresholds were explored, this value consistently yielded the best results across all tests.

3.6 Evaluating SINDy Models

Once we fit a SINDy model to a training dataset, we need to evaluate its performance. To do this, we check whether the SINDy model identifies the correct set of non-zero coefficients and compute the relative coefficient errors and relative trajectory errors as specified in Section 2.3. Determining if the model has the correct set of non-zero coefficients and calculating the relative coefficient error is straightforward. However, more care is required when computing the relative trajectory errors.

Two factors influence the calculation of relative trajectory errors: the length of the trajectory's time interval and the spacing between data points within this interval. The spacing between data points is relatively unimportant, as long as it is small enough to capture the trajectory's details. However, the evaluation range significantly impacts the results. If the evaluation range is too short, some SINDy models may exhibit very low trajectory error even if they fail to identify the correct set of non-zero terms or have large coefficient errors. This occurs because the model approximates the trajectory well near the initial condition but performs worse as it is integrated forward in time, where small errors

System Name	Evaluation Range
Duffing Oscillator (no driving term)	$[0, 50]$
Van der Pol Oscillator	$[0, 25]$
Lotka–Volterra	$[0, 50]$
Nonlinear Pendulum	$[0, 25]$
Lorenz System	$[0, 10]$

Table 2: Evaluation ranges for each dynamical system tested.

accumulate. Conversely, if the evaluation range is too long, even correct SINDy models with low coefficient errors may diverge from the true trajectory over time. This issue is particularly pronounced in chaotic systems, where small errors can rapidly lead to significantly different trajectories.

Both cases can be mitigated by selecting an evaluation range appropriate for the specific use case of SINDy. In our tests, we chose evaluation ranges short enough not to penalize models that closely match the underlying system, but long enough to ensure that incorrect models performed poorly. The evaluation ranges used in our tests are provided in Table 2.

4 Results

Our main set of tests examines the relationships between three important variables: the level of noise added to the data (as described in Section 2.4), the time between data points Δt, and the time span over which training data are collected. We conduct three tests to explore the relationships between each pair of these variables.

In the results presented below, we primarily show graphs for the Lotka-Volterra system unless other systems exhibit behavior of particular interest. This is because, in most cases, all systems displayed similar qualitative behavior with respect to the variables being changed. Additionally, we present results only for E-SINDy, as it performs very similarly to SINDy but demonstrates slightly higher noise tolerance and greater efficacy with limited data. The same trends hold for SINDy, albeit with slightly reduced noise tolerance and a need for slightly more training data.

4.1 Basic Tests

To begin, we present a basic example of SINDy being trained on the Lotka-Volterra system with a noise level of 0.05, $\Delta t = 0.01$, and a training time span of $[0, 25]$. We use the default set of library functions which are polynomials of the state variables up to degree 5. See Figure 4 for details.

In this test, SINDy successfully identified the correct form of the governing equations. The true underlying system was

$$\dot{x}_1 = 1.1x_1 - 0.4x_1x_2$$

$$\dot{x}_2 = -0.4x_2 + 0.1x_1x_2$$

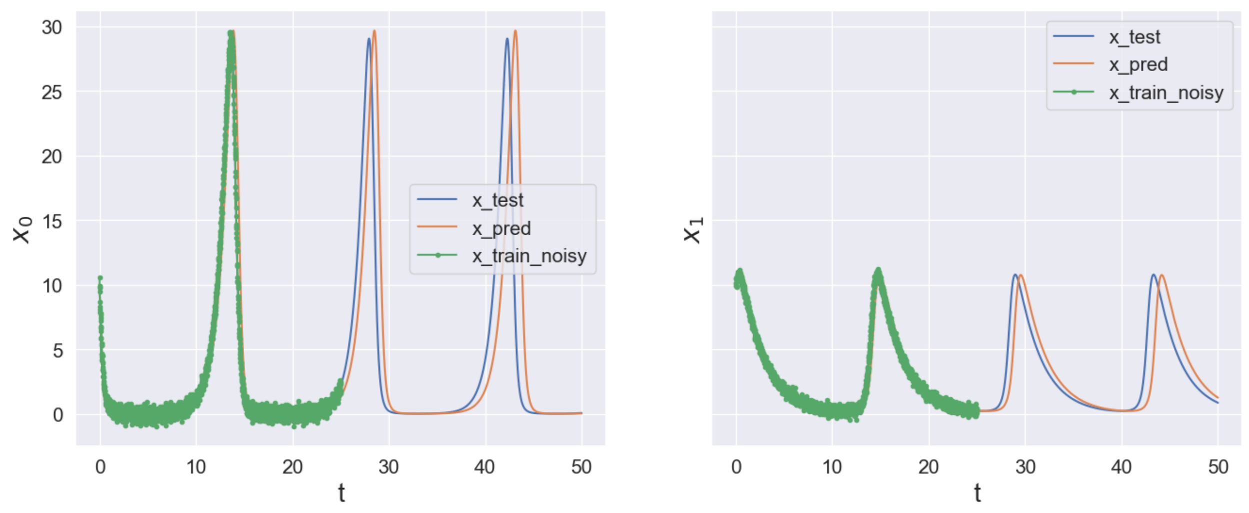

Figure 4: Example of training data (green points), prediction (orange line), and true trajectory (blue) for SINDy on the Lotka-Volterra system. The left plot shows the first state variable, and the right plot shows the second state variable.

and SINDy predicted the system

$$\dot{x}_1 = 1.09088x_1 - 0.40017x_1x_2$$

$$\dot{x}_2 = -0.39275x_2 + 0.09754x_1x_2.$$

Notated as we described SINDy in Section 2.1, the true system is

$$\begin{bmatrix} \dot{x}_1(t_1) & \dot{x}_2(t_1) \\ \dot{x}_1(t_2) & \dot{x}_2(t_2) \\ \vdots & \vdots \\ \dot{x}_1(t_m) & \dot{x}_2(t_m) \end{bmatrix} = \begin{bmatrix} 1 & x_1(t_1) & x_2(t_1) & x_1(t_1)x_2(t_1) & x_1(t_1)^2 & x_2(t_1)^2 & \cdots \\ 1 & x_1(t_2) & x_2(t_2) & x_1(t_2)x_2(t_2) & x_1(t_2)^2 & x_2(t_2)^2 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_1(t_m) & x_2(t_m) & x_1(t_m)x_2(t_m) & x_1(t_m)^2 & x_2(t_m)^2 & \cdots \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 1.1 & 0 \\ 0 & -0.4 \\ -0.4 & 0.1 \\ 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \end{bmatrix},$$

and the predicted system is

$$\begin{bmatrix} \dot{x}_1(t_1) & \dot{x}_2(t_1) \\ \dot{x}_1(t_2) & \dot{x}_2(t_2) \\ \vdots & \vdots \\ \dot{x}_1(t_m) & \dot{x}_2(t_m) \end{bmatrix} = \begin{bmatrix} 1 & x_1(t_1) & x_2(t_1) & x_1(t_1)x_2(t_1) & x_1(t_1)^2 & x_2(t_1)^2 & \cdots \\ 1 & x_1(t_2) & x_2(t_2) & x_1(t_2)x_2(t_2) & x_1(t_2)^2 & x_2(t_2)^2 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_1(t_m) & x_2(t_m) & x_1(t_m)x_2(t_m) & x_1(t_m)^2 & x_2(t_m)^2 & \cdots \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 1.09088 & 0 \\ 0 & -0.39275 \\ -0.40017 & 0.09754 \\ 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \end{bmatrix}.$$

This model resulted in a very low relative coefficient error of 0.0096. Even though the predicted trajectory closely matches the true trajectory, the peaks of the predicted trajectory begin to slightly shift away from the true trajectory over time. As a result, the relative trajectory error is relatively high at 0.51. This example highlights a potential issue with relying solely on trajectory errors for evaluation. A brief note on interpreting relative error: Recall that relative error takes the general form

$$E_X = \frac{\|X_{\text{prediction}} - X_{\text{true}}\|}{\|X_{\text{true}}\|}.$$

If $X_{\text{prediction}} = \mathbf{0}$, then $E_X = 1$. In other words, a relative error of 1 or greater indicates that the prediction is no better than simply guessing that each entry of $X_{\text{prediction}}$ is 0. Consequently, a relative error of 1 or more should render the prediction useless. That said, we have observed qualitatively that trajectories with a relative error of less than approximately 0.5 often provide a reasonable approximation of the true trajectory. To examine the effect of the STLSQ threshold on SINDy's behavior, we can lower the threshold to $\lambda = 0.01$. Now, SINDy predicts the system is

$$\dot{x}_1 = -0.03824 + 1.09295x_1 - 0.40011x_1x_2$$

$$\dot{x}_2 = -0.02100 - 0.38935x_2 + 0.09751x_1x_2.$$

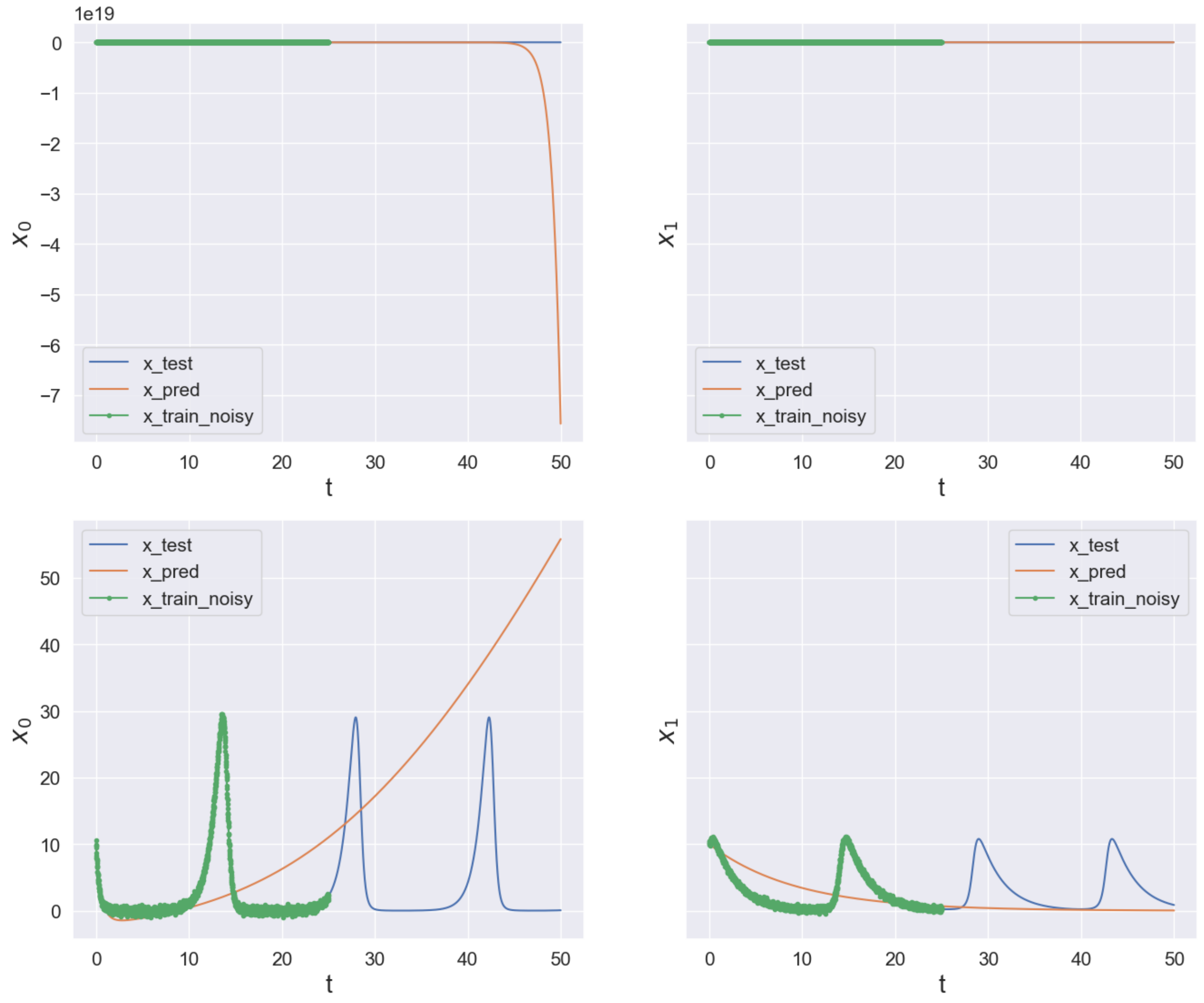

Figure 5: Examples of training data (green points), predictions (orange line), and true trajectories (blue) for SINDy on the Lotka-Volterra system. Top plots: $\lambda = 0.01$. Bottom plots: $\lambda = 0.1$. The left plots correspond to the first state variable, and the right plots correspond to the second state variable.

If on the other hand, we raise the threshold to $\lambda = 0.1$, SINDy predicts

$$\dot{x}_1 = 2.88359 + -0.64795x_2 - 0.17883x_1x_2$$

$$\dot{x}_2 = -0.10715x_2.$$

Both solutions fail to produce trajectories that closely resemble the underlying system, as shown in Figure 5.

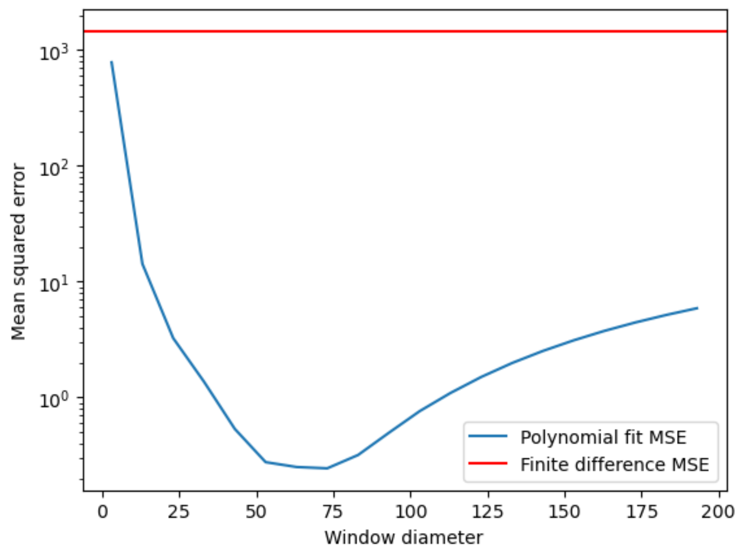

Figure 6: Mean square error of polynomial derivative approximation using third-order polynomials (y-axis) for various window diameters (x-axis). The mean square error for the finite difference method is shown as a red line.

4.2 Finite Difference vs Polynomial Approximation

In comparing the simple finite difference approximation with our polynomial derivative approximation, one surprising observation was the relatively small performance gain achieved. For most of our tests, we used third-order polynomials, as they are generally flexible enough to capture local behavior without being prone to overfitting. To determine an appropriate sliding window diameter, we performed a line search on a Lotka-Volterra trajectory with noise added at a level of 0.1. We then generated a graph of the mean square derivative error versus the window diameter, shown in Figure 6. From this graph, we can observe that for all tested window diameters, the mean square error of the polynomial approximation was lower than that of the finite difference method. For many window diameters, it was nearly four orders of magnitude better! Qualitatively, this difference is evident when we plot both the finite difference approximation and the polynomial approximation against the true derivatives, as shown in Figure 7. The polynomial approximation aligns closely with the true derivatives, while the finite difference approximation appears almost like random noise.

We anticipated that these results would allow the polynomial approximation to significantly outperform the finite difference method when used with E-SINDy. However, this was not reflected

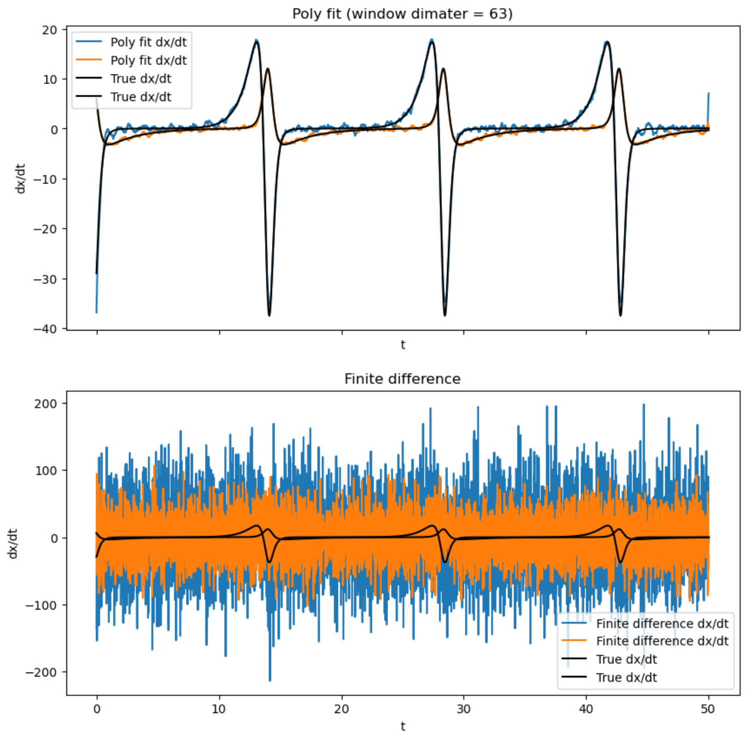

Figure 7: Top plot: Polynomial derivative approximation overlaid with the true derivatives of a Lotka-Volterra trajectory with noise added at a level of 0.1. Bottom plot: Finite difference approximation overlaid with the true derivatives for the same data.

in our tests. To compare the two methods, we trained E-SINDy models using both finite difference and polynomial approximations (with an optimal window size determined from Figure 6) across a range of noise levels on a Lotka-Volterra system. For these tests, we used $\Delta t = 0.01$ and training data spanning the full trajectory. The results are shown in Figure 8.

We observe that the finite difference method yields strictly lower coefficient errors than the polynomial approximation for noise levels up to approximately 0.1 and performs on par with it beyond this threshold. For both methods, E-SINDy fails to identify the correct models at noise levels exceeding approximately 0.13. Further research is needed to understand the reasons behind these surprising results. Given that the polynomial derivative approximation provides only marginal improvements while introducing the additional complexity of selecting a window diameter, we will use the finite difference method for the remainder of our tests.

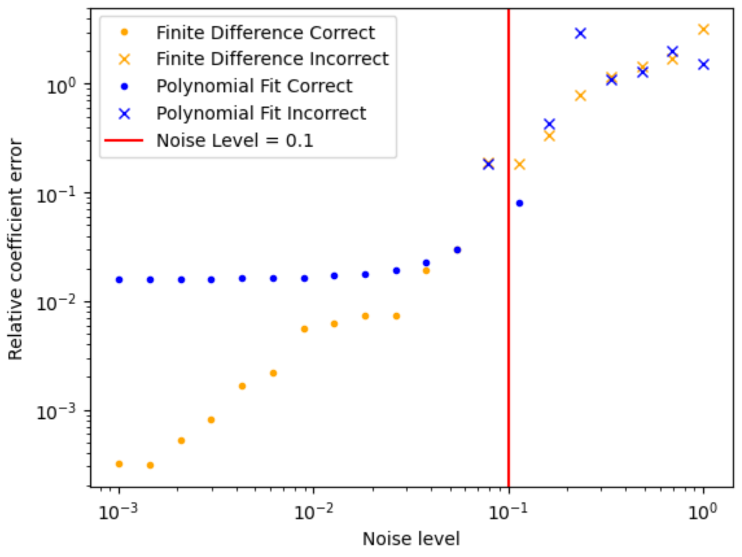

Figure 8: Relative coefficient errors (y-axis) for E-SINDy models trained on the Lotka-Volterra system across a range of noise levels (x-axis). Orange points represent models using the finite difference method, and blue points represent the polynomial derivative approximation. Dots indicate that the model identified the correct form of the governing equations, while crosses indicate failure to do so.

4.3 Noise Level vs Time Between Data Points

One of the first tests we conducted examined the behavior of E-SINDy as we varied the noise level and the time between data points, Δt. For this test, the training data was taken from the entire trajectory. The results are presented in Figure 9.

When comparing the two heat maps, we observe that the trajectory error is generally low only when the relative coefficient error is very close to zero. This suggests that unless the coefficients of the predicted system closely match the true coefficients, the predicted trajectory is likely to be a poor approximation of the true trajectory. Initially, we hypothesized that reducing the time step Δt would allow the model to tolerate higher levels of noise. However, our results contradict this assumption. Instead, we observe that each system exhibits distinct and relatively sharp cutoff thresholds for both the noise level and the time step Δt, and these thresholds appear to be largely independent of one another. If either the noise level or Δt exceeds its respective threshold, the trajectory error increases dramatically or causes the numerical computation of the trajectory to fail entirely. To better understand how trajectories differ qualitatively, we display plots for the trajectories associated with the test in cell $(0, 0)$ and the test in cell $(5, 5)$ in Figure 10. Notice

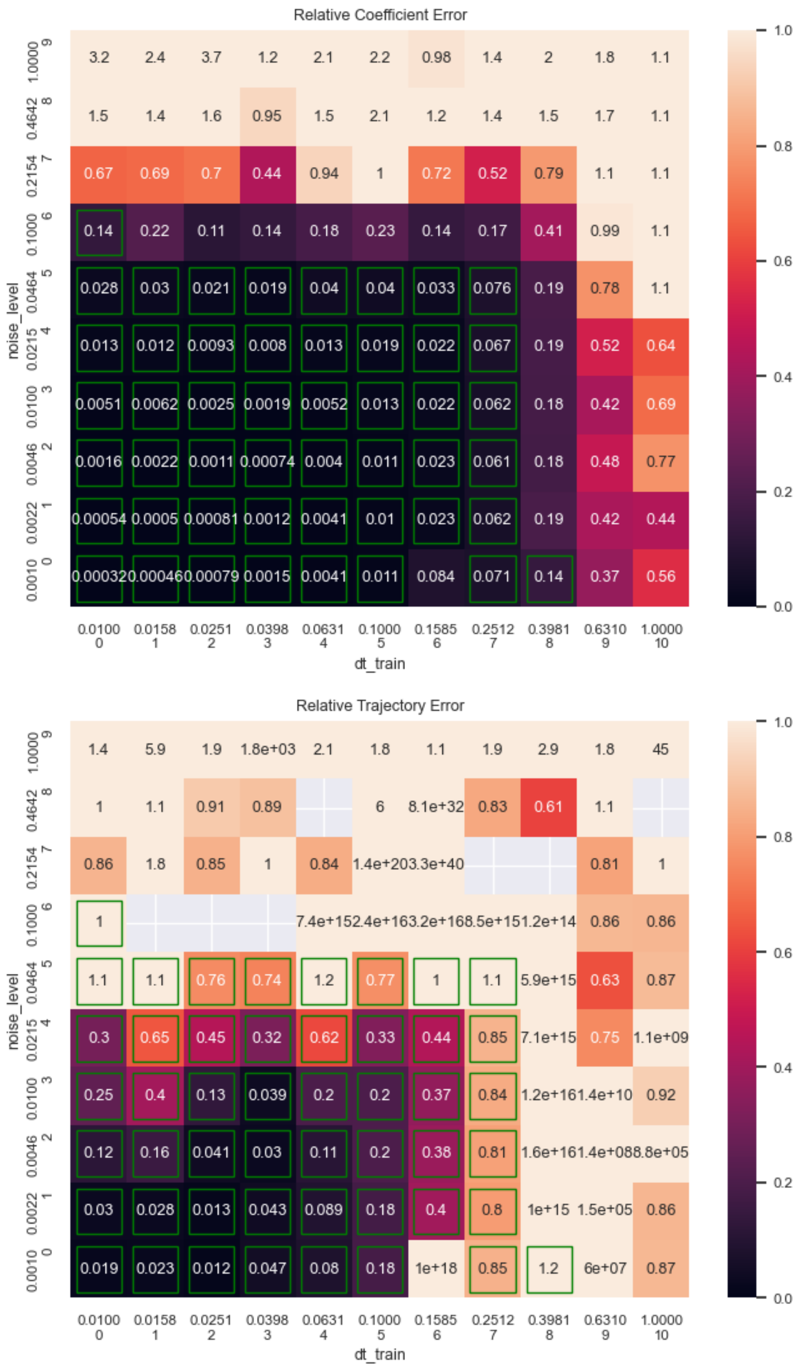

Figure 9: Heat maps of relative coefficient error (top heat map) and relative trajectory error (bottom heat map) for E-SINDy on the Lotka-Volterra system as Δt (x-axis) and noise level (y-axis) vary. Cells outlined in green indicate that E-SINDy identified the correct form of the governing equations (i.e., it produced the correct set of non-zero coefficients). Empty/gray cells in the relative trajectory error heat map indicate cases where the numerical integrator was unable to generate a complete trajectory for the predicted ODE.

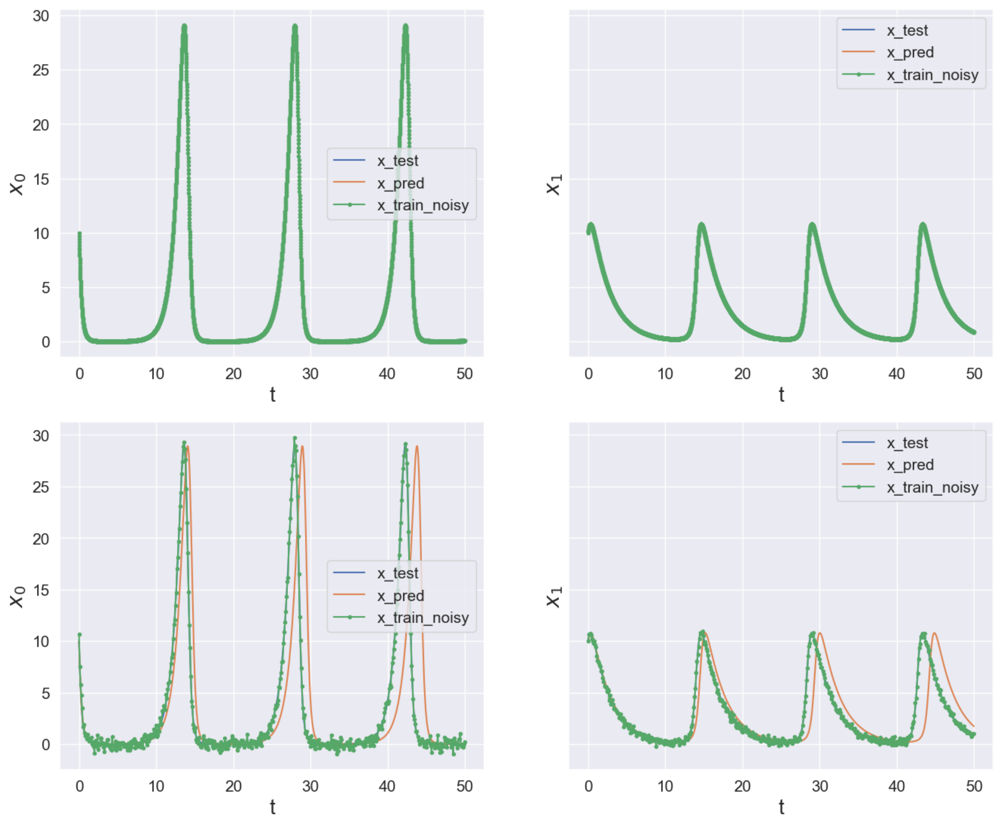

Figure 10: Comparison of two different sets of Lotka-Volterra training points and resulting E-SINDy trajectories for varying noise levels and spacing between data points. The top plot corresponds to the test in cell $(0, 0)$, and the bottom plot corresponds to the test in cell $(5, 5)$ of Figure 9.

that, even though the trajectory error is quite high (0.77) in cell $(5, 5)$, the predicted trajectory qualitatively resembles the true trajectory.

As a point of comparison, in the Duffing Oscillator system, the trajectory error can be very low even when E-SINDy fails to identify the correct form of the governing equations or when the coefficient error is high. This behavior is illustrated in Figure 11. This highlights the challenge of relying on a single metric to evaluate SINDy's performance. If the goal is strictly to identify the correct governing equations, different considerations may be necessary compared to cases where the goal is only to predict trajectories accurately.

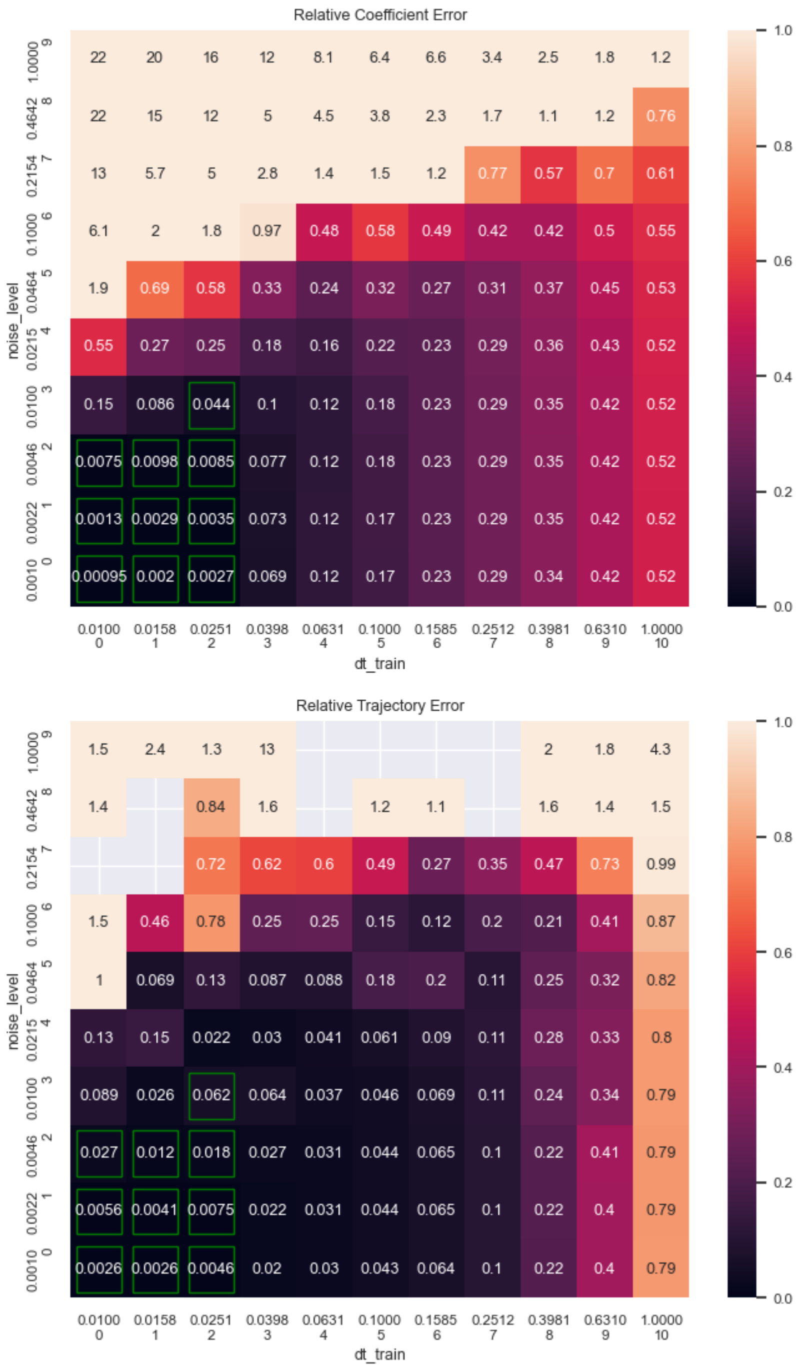

Figure 11: Heat maps of relative coefficient error (top heat map) and relative trajectory error (bottom heat map) for E-SINDy on the Duffing Oscillator system as Δt (x-axis) and noise level (y-axis) vary. Cells outlined in green indicate that E-SINDy identified the correct form of the governing equations (i.e., it produced the correct set of non-zero coefficients). Empty/gray cells in the relative trajectory error heat map indicate cases where the numerical integrator was unable to generate a complete trajectory for the predicted ODE.

4.4 Time Span vs Time Between Data Points

In the next series of tests, we examined how E-SINDy performed as we varied the time span of the training data and the time between data points, Δt. For these tests, the noise level was fixed at 0.01. Similar to the previous test, we observed sharp thresholds for both the time span and Δt. If Δt is too large or the time span is too short, E-SINDy fails to identify the correct governing equations. Interestingly, faster sampling rates do not appear to improve performance, as shown in Figure 12. Additionally, for periodic systems, collecting data from slightly less than one period appears sufficient for E-SINDy to reproduce the dynamics, as illustrated in Figure 13.

4.5 Time Span vs Noise Level

Our final series of tests examined the relationship between the time span of the training data and tolerance to noise, assuming a fixed value for Δt. In general, longer data collection windows allowed for higher noise tolerance. This is evident in Figure 14, which shows that longer data collection spans allow E-SINDy to correctly identify the system over a wider range of noise levels while maintaining low coefficient and trajectory errors. As in previous tests, a minimum of approximately one period of data was required for E-SINDy to begin producing reasonable results.

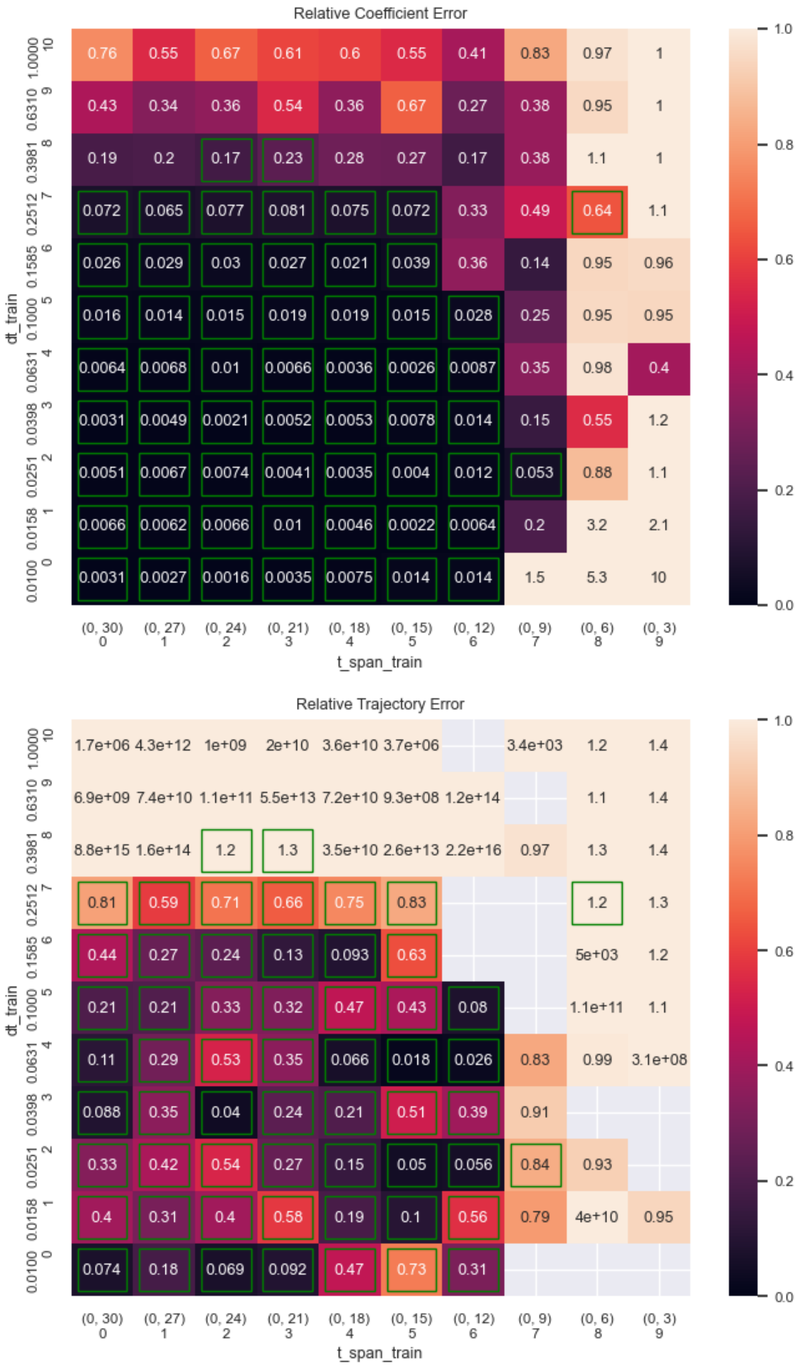

Figure 12: Heat maps of relative coefficient error (top heat map) and relative trajectory error (bottom heat map) for E-SINDy on the Lotka-Volterra system as Δt (y-axis) and the training time span (x-axis) vary. Cells outlined in green indicate that E-SINDy identified the correct form of the governing equations (i.e., it produced the correct set of non-zero coefficients). Empty/gray cells in the relative trajectory error heat map indicate cases where the numerical integrator was unable to generate a complete trajectory for the predicted ODE.

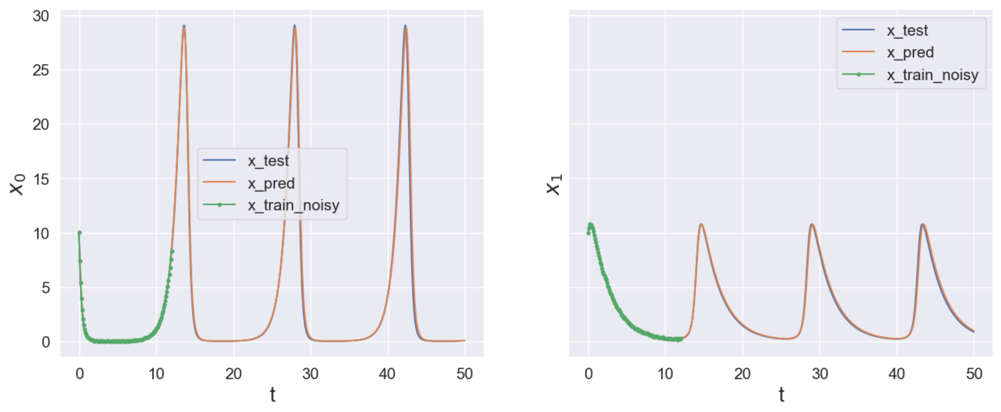

Figure 13: E-SINDy trained on a partial Lotka-Volterra trajectory, showing the resulting predicted trajectory compared to the true dynamics.

5 Discussion and Future Work

In this section, we briefly summarize our results and discuss potential areas for future exploration and testing.

5.1 Discussion

Throughout our testing, we found relatively weak correlations between the three variables tested, with each variable appearing to act largely independently of the others. The results of our experimentation can be summarized as follows:

- **Time Between Samples (Δt):** Each system exhibited a sharp cutoff for the maximum permissible value of Δt. Beyond this threshold, SINDy fails to work. However, reducing Δt below this threshold does not improve SINDy's resilience to noise.

- **Time Span vs Δt:** Having samples with a very small Δt cannot compensate for not sampling over a large enough time span of a trajectory. For systems with periodic behavior, sampling at least one complete cycle is generally necessary for SINDy to identify the governing equations accurately.

- **Time Span vs Noise Tolerance:** Sampling over a larger time span of a trajectory increases noise tolerance, but only up to a certain point. Beyond this, additional time span does not yield significant improvements.

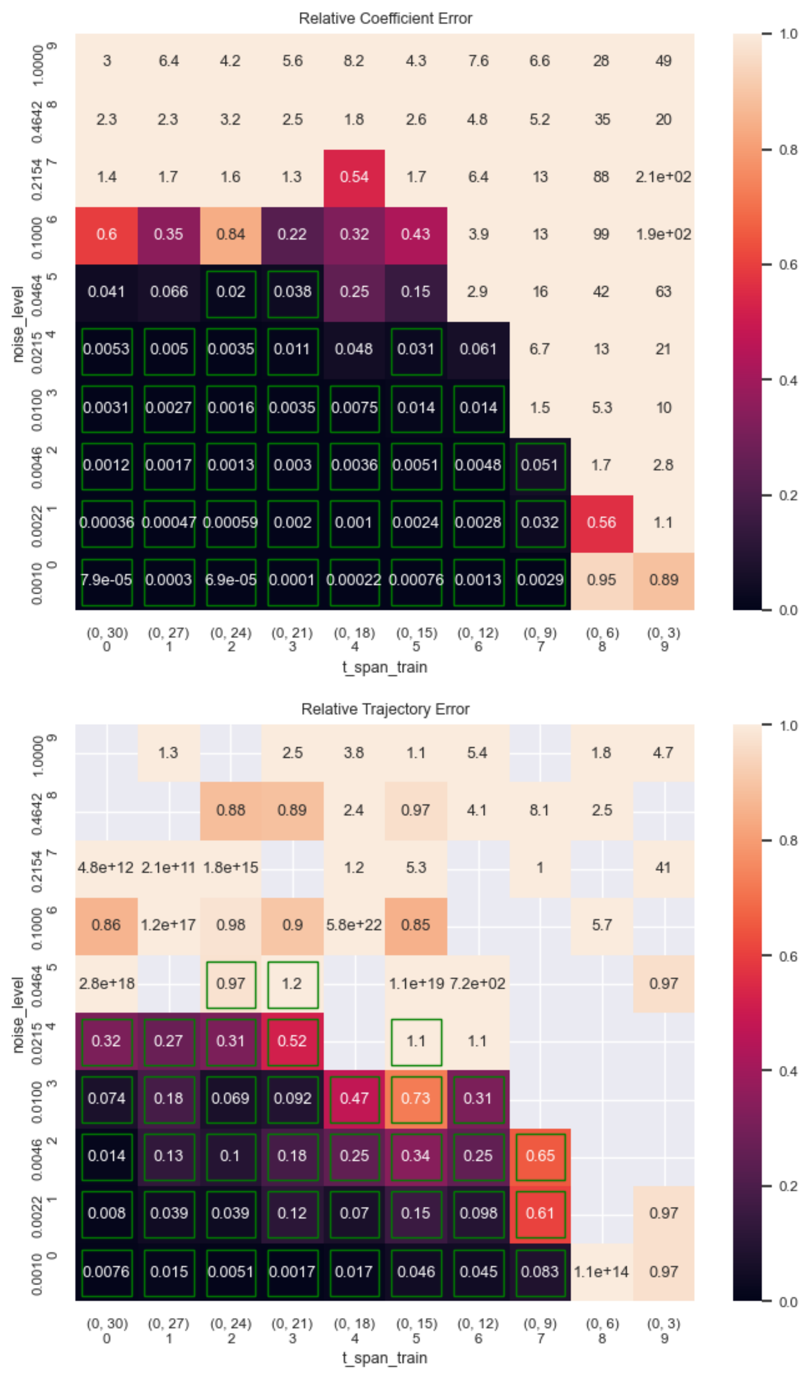

Figure 14: Heat maps of relative coefficient error (top heat map) and relative trajectory error (bottom heat map) for E-SINDy on the Lotka-Volterra system as the noise level (y-axis) and the training time span (x-axis) vary. Cells outlined in green indicate that E-SINDy identified the correct form of the governing equations (i.e., it produced the correct set of non-zero coefficients). Empty/gray cells in the relative trajectory error heat map indicate cases where the numerical integrator was unable to generate a complete trajectory for the predicted ODE.

These findings provide valuable insights into how to allocate resources when collecting data for practical applications of SINDy. Firstly, if data is not sampled frequently enough (i.e., Δt is too large), neither the quantity of data nor the trajectory length can compensate, and E-SINDy will fail. Conversely, sampling more frequently than the required threshold does not enhance noise resilience or allow for shorter trajectory sampling. Additionally, for periodic systems, it is generally necessary to collect data spanning at least one full cycle of the trajectory for (E)SINDy to identify the system accurately. Sampling over a longer time span (or potentially from multiple trajectories) improves noise resilience, but only to a certain extent.

5.2 Future Work

There are several directions we could explore in future work to build upon this project. Firstly, to complete our testing within the given time constraints, we limited our investigations to SINDy and E-SINDy. However, it would be valuable to examine how other SINDy variants perform with limited data. In particular, exploring Weak SINDy would be of interest, as it reimagines the SINDy algorithm without the need for explicit derivative computation. This characteristic may impact its tolerance to data limitations and noise.

Secondly, we were surprised by the minimal improvement observed in E-SINDy when using a polynomial derivative approximation instead of finite differences, especially given the significant quantitative and qualitative improvements in the derivative approximations themselves. Investigating the reasons behind this behavior is an important avenue for future research. One hypothesis is that polynomial approximations may introduce biases that negatively impact SINDy's performance. It would also be interesting to test whether similar behaviors are observed with other popular derivative approximation methods.

Lastly, all of our tests used a single trajectory for each system. Future work could explore how utilizing multiple trajectories might enhance results. Would E-SINDy become more tolerant to errors if samples were drawn from a collection of trajectories? Is there an optimal way to sample from multiple trajectories to maximize the effectiveness of data for training SINDy models? These directions offer promising opportunities to expand and deepen our understanding of SINDy and its variants.

6 Conclusion

In this report, we investigated the performance of SINDy E-SINDy under varying levels of noise, sampling frequency, and data availability. Our findings provide several key insights:

- There are critical thresholds for time spacing and training data duration that must be met for accurate model recovery. Sampling intervals that are too large or training data windows that are too short prevent (E)SINDy from uncovering the true governing equations, even under low noise conditions. Once these thresholds are met, further reducing sampling intervals or expanding the training data span offers minimal improvement. For periodic systems, the minimal training data span is often about one cycle.
- While polynomial derivative approximations produce more accurate numerical derivatives, these improvements did not translate into significantly better (E)SINDy performance.

Overall, our results offer practical guidance for experimental design and data collection. To maximize the utility of training data, it is crucial to select sampling rates and intervals that satisfy, but do not greatly exceed, the critical thresholds. While these findings are specific to SINDy and E-SINDy, future work should explore whether they generalize to other SINDy algorithms, such as Weak SINDy. Additionally, further research could investigate the impact of more advanced derivative approximations or the use of multiple trajectories to enhance model performance.

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*, 2016. https://www.pnas.org/doi/10.1073/pnas.1517384113.

[2] Y. Wang, N. Wagner, and J. M. Rondinelli, “Symbolic regression in materials science,” *MRS Communications*, 2019. https://doi.org/10.1557/mrc.2019.85.

[3] N. M. Mangan, S. L. Brunton, J. L. Proctor, and J. N. Kutz, “Inferring biological networks by sparse identification of nonlinear dynamics,” *arXiv [Preprint]*, 2016. http://arxiv.org/abs/1605.08368.

[4] J. Funenga, M. Guimarães, H. Costa, and C. Soares, “Finding Real-World Orbital Motion Laws from Data,” *arXiv [Preprint]*, 2023. http://arxiv.org/abs/2311.10012.

[5] Y. Lu, C. Liu, H. Hong, Y. Huang, T. Ji, and F. Xie, “An Online Data-Driven Method for Accurate Detection of Thermal Updrafts Using SINDy,” *Aerospace*, 2024. https://www.mdpi.com/2226-4310/11/10/858.

[6] U. Fasel, J. N. Kutz, B. W. Brunton, and S. L. Brunton, “Ensemble-SINDy: Robust sparse model discovery in the low-data, high-noise limit, with active learning and control,” *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 2022. https://royalsocietypublishing.org/doi/10.1098/rspa.2021.0904.

[7] D. A. Messenger and D. M. Bortz, “Weak SINDy: Galerkin-Based Data-Driven Model Selection,” *Multiscale Modeling & Simulation*, 2021. https://epubs.siam.org/doi/10.1137/20M1343166.

[8] D. A. Messenger and D. M. Bortz, “Weak SINDy for partial differential equations,” *Journal of Computational Physics*, 2021. https://www.sciencedirect.com/science/article/pii/S0021999121004204.

[9] G.-J. Both, S. Choudhury, P. Sens, and R. Kusters, “DeepMoD: Deep learning for model discovery in noisy data,” *Journal of Computational Physics*, 2021. https://www.sciencedirect.com/science/article/pii/S0021999120307592.

[10] 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*, 2019. https://www.pnas.org/doi/10.1073/pnas.1906995116.

[11] A. Cortiella, K.-C. Park, and A. Doostan, “A Priori Denoising Strategies for Sparse Identification of Nonlinear Dynamical Systems: A Comparative Study,” *arXiv.org [Preprint]*, 2022. http://arxiv.org/abs/2201.12683.

[12] F. van Breugel, J. N. Kutz, and B. W. Brunton, “Numerical differentiation of noisy data: A unifying multi-objective optimization framework,” *IEEE Access: Practical Innovations, Open Solutions*, 2020. https://pmc.ncbi.nlm.nih.gov/articles/PMC7899139/.

[13] G. C. McDonald, "Ridge regression," *WIREs Computational Statistics*, 2009. https://wires.onlinelibrary.wiley.com/doi/10.1002/wics.14.

[14] P. Bühlmann, *Bagging, Boosting and Ensemble Methods*, p. 985–1022. Springer, 2012.

[15] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, 2020. https://doi.org/10.1038/s41586-020-2649-2.

[16] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, 2020. https://www.nature.com/articles/s41592-019-0686-2.

[17] J. D. Hunter, "Matplotlib: A 2D graphics environment," *Computing in Science & Engineering*, 2007. https://ieeexplore.ieee.org/document/4160265.

[18] M. L. Waskom, "seaborn: statistical data visualization," *Journal of Open Source Software*, 2021. https://doi.org/10.21105/joss.03021.

[19] H. J. Korsch, H.-J. Jodl, and T. Hartmann, *The Duffing Oscillator*, p. 157–184. Springer, 2008.

[20] S. H. Strogatz, *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering*. CRC Press, 2024. https://books.google.com/books?id=1wrsEAAAQBAJ.

[21] A. Savitzky and M. J. E. Golay, "Smoothing and Differentiation of Data by Simplified Least Squares Procedures.," *Analytical Chemistry*, 1964. https://pubs.acs.org/doi/abs/10.1021/ac60214a047.

[22] A. Quarteroni, R. Sacco, and F. Saleri, *Numerical Mathematics (Texts in Applied Mathematics)*. Springer, 2007.

[23] B. M. de Silva, K. Champion, M. Quade, J.-C. Loiseau, J. N. Kutz, and S. L. Brunton, "PySINDy: A Python package for the sparse identification of nonlinear dynamical systems from data," *Journal of Open Source Software*, 2020. https://doi.org/10.21105/joss.02104.

[24] A. A. Kaptanoglu, B. M. de Silva, U. Fasel, K. Kaheman, A. J. Goldschmidt, J. Callaham, C. B. Delahunt, Z. G. Nicolaou, K. Champion, J.-C. Loiseau, J. N. Kutz, and

S. L. Brunton, "PySINDy: A comprehensive Python package for robust sparse system identification," *Journal of Open Source Software*, 2022. `https://doi.org/10.21105/joss.03994`.

[25] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, 2011. `http://jmlr.org/papers/v12/pedregosa11a.html`.