

A Robust SINDy Approach by Combining Neural Networks and an Integral Form

Ali Forootani* Pawan Goyal† Peter Benner††

* *Max Planck Institute for Dynamics of Complex Technical Systems, 39106 Magdeburg, Germany.*

Email: forootani@mpi-magdeburg.mpg.de, ORCID: [0000-0001-7612-4016](https://orcid.org/0000-0001-7612-4016)

† *Max Planck Institute for Dynamics of Complex Technical Systems, 39106 Magdeburg, Germany.*

Email: goyalp@mpi-magdeburg.mpg.de, ORCID: [0000-0003-3072-7780](https://orcid.org/0000-0003-3072-7780)

†† *Max Planck Institute for Dynamics of Complex Technical Systems, 39106 Magdeburg, Germany.*

Email: benner@mpi-magdeburg.mpg.de, ORCID: [0000-0003-3362-4103](https://orcid.org/0000-0003-3362-4103)

Abstract: The discovery of governing equations from data has been an active field of research for decades. One widely used methodology for this purpose is sparse regression for nonlinear dynamics, known as SINDy. Despite several attempts, noisy and scarce data still pose a severe challenge to the success of the SINDy approach. In this work, we discuss a robust method to discover nonlinear governing equations from noisy and scarce data. To do this, we make use of neural networks to learn an implicit representation based on measurement data so that not only it produces the output in the vicinity of the measurements but also the time-evolution of output can be described by a dynamical system. Additionally, we learn such a dynamic system in the spirit of the SINDy framework. Leveraging the implicit representation using neural networks, we obtain the derivative information—required for SINDy—using an automatic differentiation tool. To enhance the robustness of our methodology, we further incorporate an integral condition on the output of the implicit networks. Furthermore, we extend our methodology to handle data collected from multiple initial conditions. We demonstrate the efficiency of the proposed methodology to discover governing equations under noisy and scarce data regimes by means of several examples and compare its performance with existing methods.

Keywords: Sparse regression, discovering governing equations, neural networks, nonlinear system identification, Runge-Kutta scheme.

Novelty statement:

- We study the problem of discovering governing equations using noisy and scarce data through the lens of SINDy.
- We utilize neural network capabilities to avoid the requirement of explicit derivative information.
- In most scenarios, we observe an improved performance of the proposed methodology.

1. Introduction

System identification is a crucial aspect of understanding and modeling the dynamics of various physical, chemical, and biological systems. Over the years, various powerful and efficient system identification techniques have been developed, and these methods have been applied in a wide range of applications, see, e.g., [1–3]. Traditionally, system identification techniques rely on prior model hypotheses. With a linear model hypothesis, several methodologies have been proposed; see, e.g., [1, 2].

However, for nonlinear system identification, defining a prior is challenging, and it is often done with the help of practitioners. Despite several earlier works [4–6], nonlinear system identification is still an active and exciting research field. Towards automatic nonlinear system identification, generic algorithms and symbolic regression have shown their effectiveness and promises in discovering governing nonlinear equations using measurement [7, 8]. However, their computational expenses remain undesirable.

Instead of building suitable functions in the spirit of symbolic regression, there has been a focus on sparsity-promoting approaches for nonlinear system identification [9–11]. They rely on the assumption that nonlinear dynamics can be defined by a few nonlinear basis functions from a dictionary with a large collection of nonlinear basis functions. Such a technique enables the discovery of interpretable, parsimonious, and generalizable models that balance precision and performance. It is nowadays widely referred to as SINDy [11]. SINDy has been employed for a handful number of challenging model discovery problems such as fluid dynamics [12], plasma dynamics [13], turbulence closures [14], mesoscale ocean closures [15], nonlinear optics [16], computational chemistry [17], and numerical integration [18]. Moreover, the results of the SINDy have been extended widely to many applications, such as nonlinear model predictive control [19], rational functions [20, 21], enforcing known conservation laws and symmetries [12], promoting stability [22], generalizations for stochastic dynamics [23], from Bayesian perspective [24].

Often, SINDy approaches require a reliable estimate of the derivative information, making them very challenging for noisy and scarce data regimes. Blending numerical methods [21, 25–28] and weak formulations of differential equations [28] avoid these requirements, but their performance still deteriorates for low signal-to-noise measurements. In addition, the method in [28] relies on the choice of basis functions that allow to write differential equations in a weak formulation. The work in [29] utilizes the concepts of an ensemble to improve the predictions, but it still requires reliable estimates of derivatives to some extent. To discover governing equations from noisy data, the authors in [30] proposed a scheme that aims to decompose the noisy signals into clean signals and the noise using a Runge-Kutta-based integration method. However, the scheme explicitly estimates the noise, making it harder to scale, and requires all the dependent variables to be available at the same time grid.

Recently, applications of deep neural networks (DNN) have received attention in sparse regression model discovery methods. For instance, in [31], a deep learning-based discovery algorithm has been employed to identify underlying (partial) differential equations. However, therein, only a single trajectory is considered to recover governing equations, but in many complex processes, we might require data for different parameters and initial conditions to explore rich dynamics, thus, the reliable discovery of governing equations. Furthermore, the work [31] discovers governing equations based on estimating derivative information using automatic differential tools. However, we know that differential equations can also be written in the integral form, whereby the numerical approaches can employed as well, see, e.g., [21, 32].

In this paper, we discuss an approach, namely, *iNeural-SINDy*, for the data-driven discovery of nonlinear dynamical systems using noisy data from the lens of SINDy. For this, we make use of DNN to learn an implicit representation based on the given data set so that the network outputs denoised data, which is later utilized for the sparse regression to discover governing equations. To solve the sparse regression problem, we make use of not only automatic differential tools but also integral forms for differential equations. As a result, we observe a robust discovery of governing equations. We note that such a concept has recently been used in the context of neural ODEs in [33] to learn black-box dynamics using noisy and scarce data. We further discuss how to incorporate the data coming from multiple initial conditions.

The rest of this paper is organized as follows. [Section 2](#) briefly recalls the SINDy approach [11]. In [Section 3](#), we propose a novel methodology for sparse regression to learn underlying governing equations by making use of DNN, automatic differential tools, and numerical methods. Furthermore, in [Section 4](#), we discuss its extension for multiple initial conditions and different parameters. In [Section 5](#), we demonstrate the proposed framework by means of various synthetic noisy measurements and present a comparison with the current state-of-the-art approaches. Finally, [Section 6](#) concludes the paper with a brief summary and future research avenues.

2. A Brief Recap of SINDy

The SINDy algorithm is a nonlinear system identification approach which is based on the hypothesis that governing equations of a nonlinear system can be given by selecting a few suitable basis functions, see, e.g., [11]. Precisely, it aims at identifying a few basis functions from a dictionary, containing a large number of candidate basis functions. In this regard, sparsity-promoting approaches can be employed to discover parsimonious nonlinear dynamical systems to have a good trade-off for model complexity and accuracy [34, 35]. Consider the problem of discovering nonlinear systems of the form:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t)), \quad (1)$$

where $\mathbf{x}(t) = [\mathbf{x}_1(t), \mathbf{x}_2(t), \dots, \mathbf{x}_n(t)]^\top \in \mathbb{R}^n$ denotes the state at time t , and $\mathbf{f}(\mathbf{x}) : \mathbb{R}^n \rightarrow \mathbb{R}^n$ is a nonlinear function of the state $\mathbf{x}(t)$.

Towards discovering the function \mathbf{f} in (1), which defines the vector field or dynamics of the underlying system, we start by collecting time-series data of state $\mathbf{x}(t)$. Let us further assume to have time derivative information of the state. If it is not readily available, we can approximate it using numerical methods, e.g., a finite difference scheme. Thus, consider that the data $\{\mathbf{x}(t_0), \dots, \mathbf{x}(t_{\mathcal{N}})\}$ and its derivative $\{\dot{\mathbf{x}}(t_0), \dots, \dot{\mathbf{x}}(t_{\mathcal{N}})\}$ are given. In the next step, we assemble the data in matrices as follows:

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}(t_1)^\top \\ \mathbf{x}(t_2)^\top \\ \vdots \\ \mathbf{x}(t_{\mathcal{N}})^\top \end{bmatrix} = \begin{bmatrix} \mathbf{x}_1(t_1) & \mathbf{x}_2(t_1) & \cdots & \mathbf{x}_n(t_1) \\ \mathbf{x}_1(t_2) & \mathbf{x}_2(t_2) & \cdots & \mathbf{x}_n(t_2) \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{x}_1(t_{\mathcal{N}}) & \mathbf{x}_2(t_{\mathcal{N}}) & \cdots & \mathbf{x}_n(t_{\mathcal{N}}) \end{bmatrix}, \quad (2)$$

where each row represents a snapshot of the state. Similarly, we can write the time derivative as follows:

$$\dot{\mathbf{X}} = \begin{bmatrix} \dot{\mathbf{x}}(t_1)^\top \\ \dot{\mathbf{x}}(t_2)^\top \\ \vdots \\ \dot{\mathbf{x}}(t_{\mathcal{N}})^\top \end{bmatrix} = \begin{bmatrix} \dot{\mathbf{x}}_1(t_1) & \dot{\mathbf{x}}_2(t_1) & \cdots & \dot{\mathbf{x}}_n(t_1) \\ \dot{\mathbf{x}}_1(t_2) & \dot{\mathbf{x}}_2(t_2) & \cdots & \dot{\mathbf{x}}_n(t_2) \\ \vdots & \vdots & \ddots & \vdots \\ \dot{\mathbf{x}}_1(t_{\mathcal{N}}) & \dot{\mathbf{x}}_2(t_{\mathcal{N}}) & \cdots & \dot{\mathbf{x}}_n(t_{\mathcal{N}}) \end{bmatrix}. \quad (3)$$

The next key building block in the SINDy algorithm is the construction of a dictionary $\Theta(\mathbf{y})$, containing candidate basis functions (e.g., constant, polynomial or trigonometric functions). For instance, our dictionary matrix can be given as follows:

$$\Theta(\mathbf{X}) = \begin{bmatrix} | & | & | & | & \cdots & | & | & | & | & | & \cdots \\ \mathbf{1} & \mathbf{X} & \mathbf{X}^{\text{P}_2} & \mathbf{X}^{\text{P}_3} & \cdots & \sin(\mathbf{X}) & \cos(\mathbf{X}) & \sin(2\mathbf{X}) & \cos(2\mathbf{X}) & \cdots \\ | & | & | & | & & | & | & | & | & | & \cdots \end{bmatrix}, \quad (4)$$

assume $\Theta(\mathbf{X}) \in \mathbb{R}^{m \times D}$, and in the above formulations polynomial terms are denoted by \mathbf{X}^{P_2} or \mathbf{X}^{P_3} ; to be more descriptive \mathbf{X}^{P_2} denotes the quadratic nonlinearities of the state \mathbf{X} as follows:

$$\mathbf{X}^{\text{P}_2} = \begin{bmatrix} \mathbf{x}_1^2(t_1) & \mathbf{x}_1(t_1)\mathbf{x}_2(t_1) & \cdots & \mathbf{x}_2^2(t_1) & \mathbf{x}_2(t_1)\mathbf{x}_3(t_1) & \cdots & \mathbf{x}_n^2(t_1) \\ \mathbf{x}_1^2(t_2) & \mathbf{x}_1(t_2)\mathbf{x}_2(t_2) & \cdots & \mathbf{x}_2^2(t_2) & \mathbf{x}_2(t_2)\mathbf{x}_3(t_2) & \cdots & \mathbf{x}_n^2(t_2) \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{x}_1^2(t_{\mathcal{N}}) & \mathbf{x}_1(t_{\mathcal{N}})\mathbf{x}_2(t_{\mathcal{N}}) & \cdots & \mathbf{x}_2^2(t_{\mathcal{N}}) & \mathbf{x}_2(t_{\mathcal{N}})\mathbf{x}_3(t_{\mathcal{N}}) & \cdots & \mathbf{x}_n^2(t_{\mathcal{N}}) \end{bmatrix}.$$

In this setting, each column of the dictionary $\Theta(\mathbf{x})$ denotes a candidate function in defining the function $\mathbf{f}(\mathbf{x})$ in (1). We are interested in identifying a few candidate functions from the dictionary Θ so that a weighted sum of these selected functions can describe the function \mathbf{f} . For this, we can set up a sparse regression formulation to achieve this goal. Precisely, we seek to identify a sparse vector $\Xi = [\xi_1, \xi_2, \dots, \xi_n]$, where $\xi_i^\top \in \mathbb{R}^m$ with m denoting the number of columns in Θ , that determines which features from the dictionary are active and their corresponding coefficients.

The SINDy algorithm formulates the sparse regression problem as an optimization problem as follows. Given a set of observed data \mathbf{X} and the corresponding time derivatives $\dot{\mathbf{X}}$, the goal is to find the sparsest matrix Ξ that fulfills the following:

$$\dot{\mathbf{X}} = \Theta(\mathbf{X})\Xi.$$

Algorithm 1 SINDy algorithm [29]

Input: Dictionary Θ , time-series data \mathbf{X} , time derivative information $\dot{\mathbf{X}}$, threshold value tol , and maximum iterations max-iter .

Output: Estimated coefficients Ξ that define governing equations for nonlinear systems.

```

1:  $\Xi = (\Theta^\top \Theta) \setminus \Theta^\top \dot{\mathbf{X}}$  ▷ For initial guess, solving a least-squares problem
2:  $k = 1$ 
3: while  $k < \text{max-iter}$  do
4:    $\text{small\_inds} = (\text{abs}(\Xi) < \text{tol})$  ▷ identifying small coefficients
5:    $\Xi(\text{small\_inds}) = 0$  ▷ excluding small coefficients
6:   Solve  $\Xi = (\Theta^\top \Theta) \setminus \Theta^\top \dot{\mathbf{X}}$  subject to  $\Xi(\text{small\_inds}) = 0$ 
7:    $k = k + 1$ 

```

However, finding such a matrix is an NP hard problem. Therefore, there is a need to come up a sparsity promoting regularization, and in this category, LASSO is a widely known approach [36,37]. Despite its success, it is unable to yield matrix which is the sparsest, or the approaches, e.g., discussed in [38,39], require a prior information about how many non-zeros elements are expected in the matrix Ξ , which is not known. On the other hand, the authors in [11] discuss a sequential thresholding approach, where simple least squares problems are solved iteratively, and at each step, coefficients below a given tolerance are pruned. Analysis of such an algorithm is discussed in [40]. We summarize the SINDy approach in Algorithm 1. Moreover, we mention that other regularization schemes or heuristics are discussed in [21, 22, 25, 41, 42], but in this work, we focus only on the sequential thresholding approach, similar to in Algorithm 1 due to its simplicity.

3. *iNeural-SINDy*: Neural Networks and Integrating Schemes Assisted SINDy Approach

A challenge in the classical SINDy approach discussed in the previous section is the availability of an accurate estimate of the derivative information. If the derivative information is inaccurate, the resulting sparse model may not accurately capture the underlying system dynamics.

In this section, we present an approach that combines SINDy framework with a numerical integration scheme and neural networks in a particular way so that a robust discovery of governing equations can be made amid poor signal-to-noise ratio and irregularities in data. The methodology is inspired by the work [33]. The main components of the methodology are as follows. For given noisy data, we aim to learn an implicit representation using a neural network based on the noisy data so that the network yields denoised data but still in the vicinity of the noisy collected data, and governing equations describing the dynamics of the denoised data can be obtained by employing SINDy. For SINDy, we utilize automatic differential tools to obtain the derivative information via the network and also make use of an integral form of the differential equations. In the following, we make these discussions more precise.

Consider noisy data $\mathbf{y}(t) \in \mathbb{R}^n$ at the time instances $\{t_0, \dots, t_{\mathcal{N}}\}$, i.e., $\{\mathbf{y}(t_0), \dots, \mathbf{y}(t_{\mathcal{N}})\}$. Moreover, $\mathbf{y}(t) = \mathbf{x}(t) + \epsilon(t)$, where $\mathbf{x}(t)$ and $\epsilon(t)$ denote clean data and noise, respectively. Under this setting, we aim to discover the structure of vector field \mathbf{f} by identifying the most active terms in the dictionary Θ so that it satisfies as follows:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}). \quad (5)$$

Note that we do not know ϵ 's. In order to learn \mathbf{f} from \mathbf{y} , we blend three ingredients together, which are discussed in the following.

- (a) **Sparse regression assumption:** In our setting, we utilize the principle of SINDy, which we discussed in the previous section. This means that the system dynamics (or vector field defining dynamics) can be represented by a few suitable terms from a dictionary of candidate functions. This allows us to obtain a parsimonious representation of dynamical systems and reduces the model complexity, leading to better generalization and interpretability of the models.
- (b) **Automatic differential to estimate derivative information:** As mentioned earlier, SINDy algorithm requires accurate derivative information for the system, which can be challenging

to obtain from experiments or to estimate using numerical methods. To cope with this issue, we make use of neural networks with its automatic differentiation (AD) feature, which is a technique used to estimate derivative information. The use of a DNN in combination with the SINDy algorithm was earlier discussed in [31], where it has been shown that the discovery of nonlinear system dynamics without explicit need of accurate derivative information [31].

We make use of a DNN to parameterize a nonlinear mapping from time t to the dependent variable $\mathbf{y}(t)$. To that end, let us denote a DNN by \mathcal{G}_θ , where θ contains DNN parameters. The input to \mathcal{G}_θ is time t , and its output is $\mathbf{y}(t)$, i.e., $\mathbf{y}(t) = \mathcal{G}_\theta(t)$. However, in the case of noisy measurement $\mathbf{y}(t)$ at time $\{t_0, \dots, t_{\mathcal{N}}\}$, we expect \mathcal{G}_θ to predict outputs in the proximity of \mathbf{y} , i.e.,

$$\mathbf{y}(t) \approx \mathbf{x}(t) = \mathcal{G}_\theta(t), \quad t = \{t_0, \dots, t_{\mathcal{N}}\}.$$

With the sparse regression hypothesis, we aim to learn a dynamical model for \mathbf{x} , as it can be seen as a denoised version of \mathbf{y} . For this, we construct a dictionary of possible candidate functions using \mathbf{x} , which we denote by $\Theta(\mathbf{X}(t))$. Next, we require the derivative information of \mathbf{x} with respect to time t . Since we have an implicit representation of $\mathbf{x}(t)$ using a DNN, we can employ AD to obtain the required information. Having dictionary and derivative information, we set up a sparse regression problem as follows:

$$\dot{\mathbf{X}}(t) = \Theta(\mathbf{X}(t))\Xi, \quad (6)$$

where Ξ is the sparsest possible matrix, which selects the most active terms from the dictionary to define dynamics. Finding the sparsest solution is computationally infeasible; we, thus, utilize the sequential thresholding approach as discussed in Algorithm 1 with minor modifications. Instead of solving least-squares problems in Steps 1 and 5 in Algorithm 1, we have a loss function as follows:

$$\mathcal{L} := \min_{\theta, \Xi} \sum_{i=0}^{\mathcal{N}} \lambda_1 \|\mathbf{y}(t_i) - \mathbf{x}(t_i)\| + \lambda_2 \|\dot{\mathbf{x}}(t) - \Theta(\mathbf{x}(t))\Xi\|, \quad (7)$$

where $\mathbf{x}(t_i) := \mathcal{G}_\theta(t_i)$, and λ_1 and λ_2 are hyperparameters.

- (c) **Numerical integration scheme:** A dynamical system is a particle or an ensemble of particles whose state varies over time and thus obeys differential equations involving time derivatives [43]. To predict the evolution of the dynamical system, it is necessary to have an analytical solution of such equations or their integration over time through computer simulations. Therefore, we aim to incorporate the information contained in the form of integration of dynamical systems while discovering governing equations via sparse regression, which is expected to make the process of discovering equations robust to the noise and scarcity of data.

When differential equations are written in an integral form, then we do not require derivative information as well; however, the resulting optimization problem involves an integral form. In this regard, one can employ the principle of Neural-ODEs [44] to solve efficiently such optimization problems. One can also approximate the integral form using suitable integrating schemes [45], and recently, fourth-order Runge-Kutta (RK4) scheme [21] and linear multi-step methods [46] are combined with SINDy. In this work, we make use of the RK4 scheme to approximate an integral.

Following [21], our goal is to predict the state of a dynamic system $\mathbf{x}(t_{k+1})$ at time $t = t_{k+1}$ from the state $\mathbf{x}(t_k)$ at time $t = t_k$, where $k \in \{0, 1, \dots, \mathcal{N} - 1\}$. By employing the RK4 scheme, $\mathbf{x}(t_{k+1})$ can be computed as a weighted sum of four components that are the product of the time-step and gradient field information $\mathbf{f}(\cdot)$ at the specific locations. These components are computed as follows:

$$\mathbf{x}(t_{k+1}) \approx \mathbf{x}(t_k) + \frac{1}{6}h_k(\mathbf{a}_1 + 2 \cdot \mathbf{a}_2 + 2 \cdot \mathbf{a}_3 + \mathbf{a}_4), \quad h_k = t_{k+1} - t_k, \quad (8)$$

where,

$$\mathbf{a}_1 = \mathbf{f}(\mathbf{x}(t_k)), \quad \mathbf{a}_2 = \mathbf{f}\left(\mathbf{x}(t_k) + h_k \frac{\mathbf{a}_1}{2}\right), \quad \mathbf{a}_3 = \mathbf{f}\left(\mathbf{x}(t_k) + h_k \frac{\mathbf{a}_2}{2}\right), \quad \mathbf{a}_4 = \mathbf{f}\left(\mathbf{x}(t_k) + h_k \mathbf{a}_3\right).$$

For the sake of simplicity with a slight abuse of a notation, the right-hand side of (8) is denoted by $\mathcal{F}_{\text{RK4}}(f, \mathbf{x}(t_k), h_k)$, i.e.,

$$\mathbf{x}(t_{k+1}) = \mathbf{x}(t_k + h_k) \approx \mathcal{F}_{\text{RK4}}(\mathbf{f}, \mathbf{x}(t_k), h_k). \quad (9)$$

Like the SINDy algorithm, we collect samples from the dynamical system at time $t = \{t_0, \dots, t_{\mathcal{N}}\}$ and define the time step as $h_k := t_{k+1} - t_k$.

With sparse regression assumption, we can write $\mathbf{f}(\mathbf{x}) = \Theta(\mathbf{x})\Xi$, where $\Theta(\mathbf{x})$ is a dictionary and Ξ is a sparse matrix. Then, we can set up a sparse regression as follows. We seek to identify the sparsest matrix Ξ so that the following is minimized:

$$\sum_k \|\mathbf{x}(t_{k+1}) - \mathcal{F}_{\text{RK4}}(\Theta(\mathbf{x})\Xi, \mathbf{x}(t_k), h_k)\|.$$

When the RK4 scheme is merged with the previously discussed DNN framework, we apply a one-time ahead prediction based on RK4-SINDy to the output of our DNN, i.e.,

$$\mathbf{x}_{\text{RK4}}(t_{k+1}) \approx \mathcal{F}_{\text{RK4}}(\mathbf{f}, \mathbf{x}(t_k), h_k).$$

Having all these ingredients, we combine them to define a loss function to train our DNN structure, as well as to discover governing equations describing underlying dynamics. To that end, we have the following loss function:

$$\mathcal{L} = \mu_1 \mathcal{L}_{\text{MSE}} + \mu_2 \mathcal{L}_{\text{deri}} + \mu_3 \mathcal{L}_{\text{RK4}}, \quad \mu_1, \mu_2, \mu_3 \in [0, 1], \quad (10)$$

where \mathcal{L}_{MSE} is the mean square error (MSE) of the output of the DNN \mathcal{G}_θ (denoted by $\hat{\mathbf{x}}$) with respect to the collected data \mathbf{y} , and $\{\mu_1, \mu_2, \mu_3\}$ are positive constants, determining the weight of different losses in the total loss function. It is given as

$$\mathcal{L}_{\text{MSE}} = \frac{1}{\mathcal{N}} \sum_{k=1}^{\mathcal{N}} \|\mathbf{y}(t_k) - \mathbf{x}(t_k)\|_2^2. \quad (11)$$

It forces the DNN to produce output in the vicinity of the measurements, and μ_1 is its weight. $\mathcal{L}_{\text{deri}}$ is inspired by the sparse regression and aims to compute the sparse coefficient matrix Ξ . It is computed as follows:

$$\mathcal{L}_{\text{deri}} = \frac{1}{\mathcal{N}} \sum_{k=1}^{\mathcal{N}} \|\dot{\mathbf{x}}(t_k) - \Theta(\mathbf{x}(t_k))\Xi\|_2^2, \quad (12)$$

The term \mathcal{L}_{RK4} encodes the capabilities of the vector field to predict the state at the next time step. This is the MSE of the output of the RK4 scheme and the output of DNN, given as follows:

$$\mathcal{L}_{\text{RK4}} = \frac{1}{\mathcal{N} - 1} \sum_{k=1}^{\mathcal{N}} \left\| \frac{1}{h_k} (\mathbf{x}(t_{k+1}) - \mathcal{F}_{\text{RK4}}(\Theta(\mathbf{x}(t_k))\Xi, \mathbf{x}(t_k), h_k)) \right\|_2^2. \quad (13)$$

It is worth highlighting that the coefficient matrix Ξ will be updated alongside the weights and biases of the DNN, and the dictionary terms are calculated by (4). Furthermore, after a certain number of epoch training, we employ sequential thresholding on Ξ to remove small coefficients as sketched in Algorithm 1, and update the remaining parameters thereafter. We summarize the procedure in Algorithm 2. Additional steps in Algorithm 2 are as follows. We train our network for initial iterations (denoted by `init-iter`) without employing sequential thresholding; this helps the DNN to learn the underlying dynamics of the dataset. Afterward, we employ sequential thresholding every q iterations. In the rest of the paper, the proposed methodology is referred to as *iNeural-SINDy*.

Algorithm 2 iNeuralSINDy: SINDy combined with neural network and integral scheme for nonlinear system identification.

Input: Data set $\{\mathbf{y}(t_0), \mathbf{y}(t_1), \dots, \mathbf{y}(t_N)\}$, \mathbf{tol} for sequential thresholding, a dictionary containing candidate functions Θ , a neural network \mathcal{G}_θ (parameterized by θ), initial iteration ($\mathbf{init-iter}$), maximum iterations $\mathbf{max-iter}$, and parameters $\{\mu_1, \mu_2, \mu_3\}$.

Output: Estimated coefficients Ξ , defining governing equations.

```

1: Initialize the DNN module parameters, and the coefficients  $\Xi$ 
2:  $k = 1$ 
3: while  $k < \mathbf{max-iter}$  do
4:   Feed time  $t_i$  as an input to the DNN ( $\mathcal{G}_\theta$ ) and predict output  $\mathbf{x}$ .
5:   Compute the derivative information  $\dot{\mathbf{x}}$  using automatic differentiation.
6:   Compute the cost function (10).
7:   Update the parameters of DNN ( $\theta$ ) and the coefficient  $\Xi$  using gradient descent.
8:   for  $k\%q == 0$  &  $k > \mathbf{init-iter}$  do                                 $\triangleright$  Employing sequential thresholding after  $q$  iterations
9:      $\mathbf{small\_inds} = (\mathbf{abs}(\Xi) < \mathbf{tol})$                                  $\triangleright$  identifying small coefficients
10:     $\Xi(\mathbf{small\_inds}) = 0$                                             $\triangleright$  excluding small coefficients
11:    Update the parameters of DNN ( $\theta$ ) and the coefficient  $\Xi$  using gradient descent,
        while ensuring  $\Xi(\mathbf{small\_inds})$  remains zero.
12:     $k = k + 1$ .
```

4. Extension to Multi-trajectories Data

Thus far, we have presented the discovery of governing equations using a single trajectory time series data set using a single initial condition. However, for complex dynamical processes, a single trajectory is not sufficient to describe underlying dynamics completely. Therefore, it is necessary to collect data using multiple trajectories; hence, we need to adopt our proposed methodology to account for multiple trajectories.

To achieve this goal, we augment the input time t with an initial condition so that a DNN can capture the nonlinear behavior of the system with respect to different initial conditions. To that end, let us consider \mathcal{M} different trajectories with initial conditions $y_0^{[j]}$, where $j \in \{1, \dots, \mathcal{M}\}$. To reflect the multi-trajectories in our framework, we modify the architecture of the DNN, which now takes t_k and $y_0^{[j]}$ as inputs, and intend to predict $y_k^{[j]}$ —that is, the state at time t_k with respect to the initial condition $y_0^{[j]}$. Then, we also adapt our loss function (10) as follows:

$$\mathcal{L} = \mu_1 \sum_{j=1}^{\mathcal{M}} \mathcal{L}_{\text{MSE}}^{[j]} + \mu_2 \sum_{j=1}^{\mathcal{M}} \mathcal{L}_{\text{deri}}^{[j]} + \mu_3 \sum_{j=1}^{\mathcal{M}} \mathcal{L}_{\text{RK4}}^{[j]}, \quad \mu_1, \mu_2, \mu_3 \in [0, 1], \quad (14)$$

where

$$\begin{aligned} \mathcal{L}_{\text{MSE}}^{[j]} &= \frac{1}{\mathcal{M} \cdot \mathcal{N}} \sum_{j=1}^{\mathcal{M}} \sum_{k=1}^{\mathcal{N}} \left\| \mathbf{y}^{[j]}(t_k) - \mathbf{x}^{[j]}(t_k) \right\|_2^2, \\ \mathcal{L}_{\text{deri}}^{[j]} &= \frac{1}{\mathcal{M} \cdot \mathcal{N}} \sum_{j=1}^{\mathcal{M}} \sum_{k=1}^{\mathcal{N}} \left\| \Theta(\mathbf{x}^{[j]}(t_k)) \hat{\Xi} - \dot{\mathbf{x}}(t_k) \right\|_2^2, \\ \mathcal{L}_{\text{RK4}}^{[j]} &= \frac{1}{h} \frac{1}{\mathcal{M} \cdot \mathcal{N}} \sum_{j=1}^{\mathcal{M}} \sum_{k=1}^{\mathcal{N}} \left\| \mathbf{x}^{[j]}(t_{k+1}) - \mathbf{x}_{\text{RK4}}^{[j]}(t_k) \right\|_2^2 \quad \text{with } h = t_{k+1} - t_k. \end{aligned}$$

We depict a schematic diagram of our proposed approach in Figure 1 for such a case.

5. Numerical Experiments

In this section, we demonstrate the proposed methodology, the so-called iNeural-SINDy, by means of several numerical examples and present a comparison with existing methodologies. For the comparison, we primarily consider two approaches, namely DeePyMoD [31], and RK4-SINDy [21]. DeePyMoD utilizes only automatic-differential tools to estimate derivative information by constructing an implicit

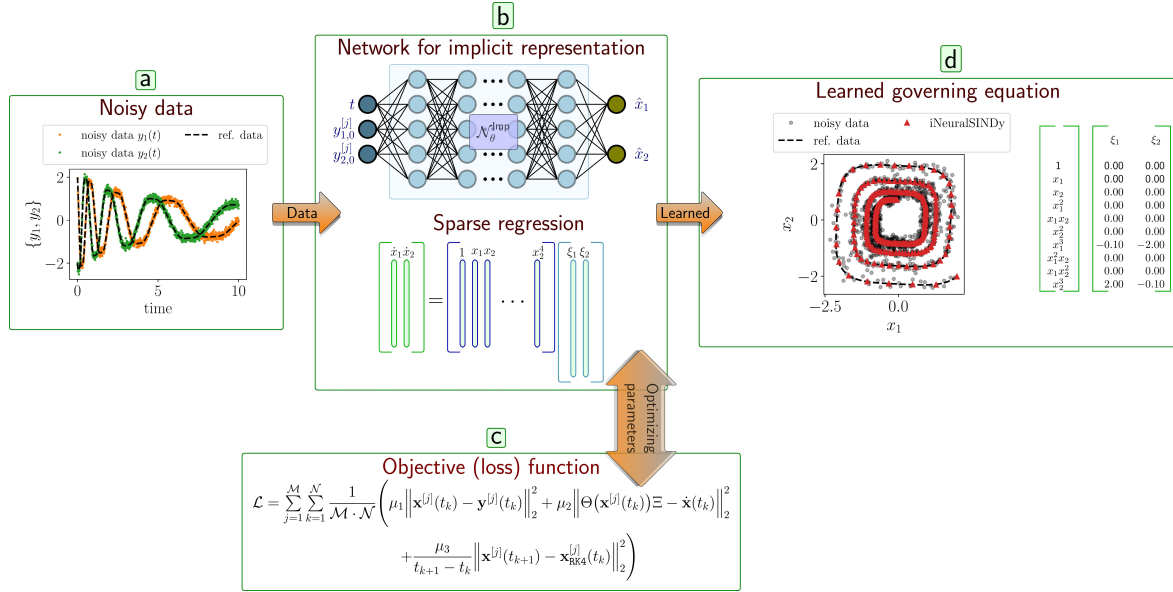


Figure 1: A schematic diagram of the approach *iNeural-SINDy*. (a) noisy measurement data, (b) feeding the initial condition $(\mathbf{y}_{1,0}, \mathbf{y}_{2,0})$ and the time t to the DNN, (c) using the output of the DNN, construct a polynomial dictionary, (d) estimating the parameters of the DNN and sparse vector Ξ by considering a loss function.

representation of the noisy data, while *RK4-SINDy* embeds a numerical integration scheme to avoid computation of derivative information. The proposed methodology *iNeural-SINDy* can be viewed as a combination of *DeePyMoD* and *RK4-SINDy*. For the chaotic Lorenz example, we also present a comparison with *Weak-SINDy* [28].

To quantify the performance of the considered methodologies, we define the following coefficient error measure for each state variable \mathbf{x}_i :

$$\mathcal{E}(\mathbf{x}_i) = \|\Xi_{\mathbf{x}_i}^{\text{truth}} - \Xi_{\mathbf{x}_i}^{\text{est}}\|_1, \quad (15)$$

where $\Xi_{\mathbf{x}_i}^{\text{truth}}$ and $\Xi_{\mathbf{x}_i}^{\text{est}}$ are, respectively, the true and estimated coefficients, corresponding to the state variable \mathbf{x}_i , and $\|\cdot\|_1$ denotes the l_1 -norm. A motivation to quantify each state variable separately is that their dynamics can be of different scales; thus, their coefficients might also be in a different order. Therefore, to better understand the quality of the discovered models, we analyze them separately. Furthermore, to observe the performance of the methodologies under the noisy data, which is often the case in real-world scenarios, we artificially generate noisy data by corrupting the clean data. For this, we use a white Gaussian noise $\mathcal{N}(\mu, \sigma^2)$ with a zero mean $\mu = 0$ and variance σ^2 , where σ denotes the standard deviation. The noise level in the data is controlled by σ , i.e., larger σ implies more noise present in the data. Additionally, since *iNeural-SINDy* and *DeePyMoD* both involve neural networks, we also compare their performance sensitivity in two scenarios as follows:

- **Scene_A:** In the first scenario, we consider having a single initial condition and a fixed number of neurons in the hidden layers but vary the amount of training data and noise levels.
- **Scene_B:** In the second one, we consider having a single initial condition and a fixed number of training data but vary the number of neurons in the hidden layers and noise levels.

In addition, in the following, we further clarify common implementation and reproducibility details that are considered for all the examples.

Data generation. We have generated the data synthetically by using `solve_ivp` function from `scipy.integrate` package to solve a given set of differential equations and produce the data set. When an identification approach terminates based on the considered methodologies (e.g., *iNeural-SINDy*, *DeePyMoD*, *RK4-SINDy*, or *Weak-SINDy*), we multiply the dictionary Θ by estimated coefficient matrix

Ξ^{est} to obtain the discovered governing equations. We then make use of the `solve_ivp` function from `scipy.integrate` to obtain time-evolution dynamics.

Moreover, we perform a data-processing step before feeding to a neural network by mapping the minimum and maximum values to -1 and 1 , respectively. The hyper-parameters μ 's in (14) are set to $\mu_1 = 1$, $\mu_2 = 0.1$ and $\mu_3 = 0.1$ for `iNeural-SINDy`. Note that we can drive `RK4-SINDy` and `DeePyMoD` approaches by setting $\mu_3 = 0$ and $\mu_2 = 0$, respectively, in (14).

Architecture. We use multi-layer perception networks with periodic activation functions, namely, `SIREN` [47], to learn an implicit representation based on measurement data. The numbers of hidden layers and neurons will be discussed for each example separately.

Hardware. For training neural networks and parameter estimations for discovering governing equations, we have used `NVIDIA`[®] `RTX A4000` GPU with 16 GB RAM, and for CPU computations (e.g., for generating data), we have used a 12th Gen `Intel`[®] `Core`[™] `i5-12600K` processor with 32 GB RAM.

Training set-up. We use the Adam optimizer [48] to update the coefficient matrix Ξ that is trained alongside the DNN parameters. The threshold value (`tol`), learning rate of the optimizer, maximum iterations (`max-iter`), initial iterations (`init-iter`), the iteration q for employing sequential thresholding for Algorithm 2 will be mentioned for each example separately.

However, we note that after each thresholding step in Algorithm 2, we reset the learning rate 5×10^{-6} for DNN parameters and 1×10^{-2} for the coefficient matrix Ξ^{est} except for the Lorenz example, which is explicitly mentioned in the Lorenz example.

5.1. Two-dimensional damped oscillators

In our first example, we consider the discovery of a two-dimensional linear oscillatory damped system using data. The dynamics of the oscillator can be given by

$$\begin{aligned}\dot{\mathbf{x}}_1(t) &= -0.1\mathbf{x}_1(t) + 2.0\mathbf{x}_2(t), \\ \dot{\mathbf{x}}_2(t) &= -2.0\mathbf{x}_1(t) - 0.1\mathbf{x}_2(t).\end{aligned}\tag{16}$$

Simulation setup: To generate the training data set, we consider three initial conditions in the range $[-2, 2]$ for \mathbf{x}_1 and \mathbf{x}_2 , and for each initial condition, we take 400 equidistant points in the time interval $t \in [0, 10]$. Our DNN architecture has three hidden layers, each having 32 neurons. We set the number of epochs `max-iter` = 15,000 and threshold value `tol` = 0.05. The initial iteration `init-iter` is set to 5,000 with the learning rate of 10^{-4} for the DNN parameters and 10^{-3} for the coefficient matrix Ξ^{est} , and after $q = 2,000$ iterations, we employ the sequential thresholding. Moreover, we construct a dictionary containing polynomials of degrees up to two.

Results: Figure 2 demonstrates the performance of different algorithms in the presence of noise. We consider additive white Gaussian noise with different standard variances $\sigma = \{0, 0.02, 0.04, 0.08\}$. It shows that as we increase the noise level, the `RK4-SINDy` fails to estimate the coefficients. However, `iNeural-SINDy` and `DeePyMoD` are robust in discovering the underlying equations accurately, even for high noise levels, and both exhibit similar performances. In Table 1 (in the appendix), we also report learned governing equations from data with various noise levels, which again illustrate that both `iNeural-SINDy` and `DeePyMoD` have similar performance, and `RK4-SINDy` fails to recover governing equations from highly noisy data. Furthermore, in Figure 3, the convergence of the non-zero coefficients for the different methods is shown as the training progresses. It can be seen that `iNeural-SINDy` has a faster convergence rate compared to `DeePyMoD` and `RK4-SINDy`. Next, we discuss the performance of `iNeural-SINDy` and `DeePyMoD` for `Scene_A` and `Scene_B`.

- **Scene_A:** We consider a DNN architecture with three hidden layers, each having 32 neurons. For comparison, we consider noise levels with standard variance $\sigma = \{0, 0.02, 0.04, 0.06\}$, and take the number of samples $\{30, 40, 50, 100, 200, 300, 400\}$ in the time interval $[0, 10]$ for a single initial condition $(\mathbf{x}_1(0), \mathbf{x}_2(0)) = (5, 2)$. The rest of the settings are the same as mentioned earlier in the simulation setup. By varying the noise levels and the number of samples, we report

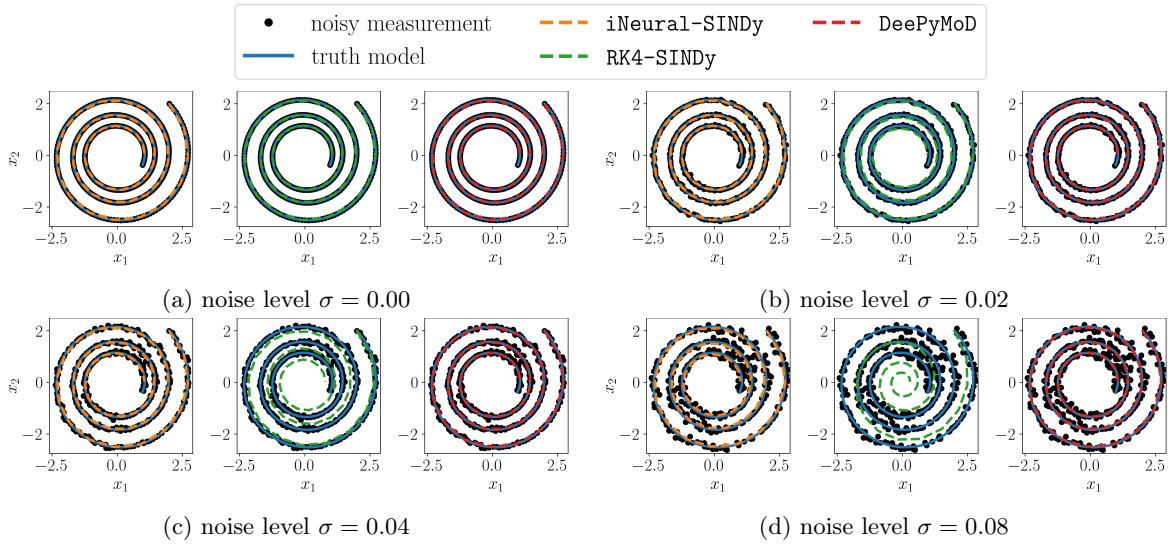


Figure 2: Linear oscillator: A comparison of the learned equations using different methods under various noise levels present in measurement with the ground truth.

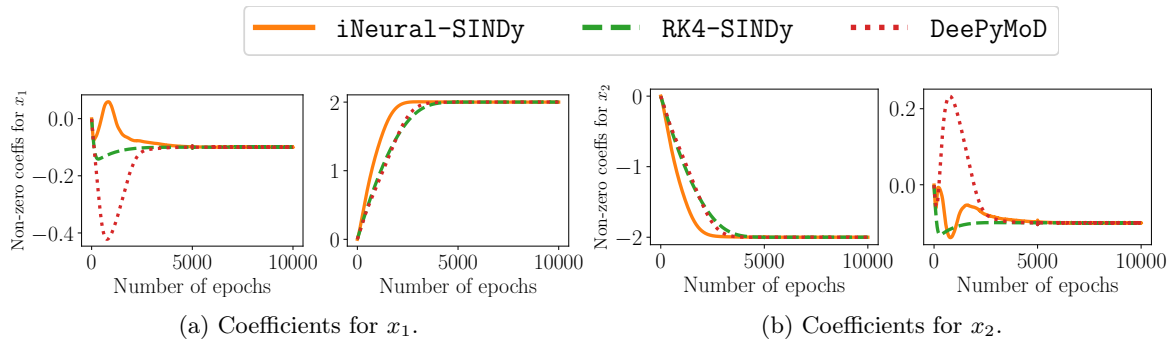


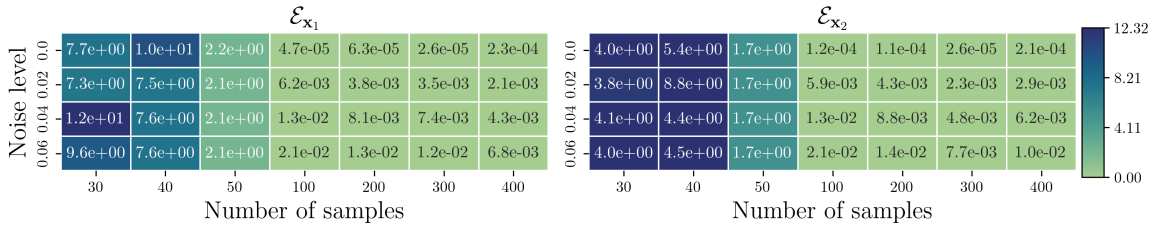
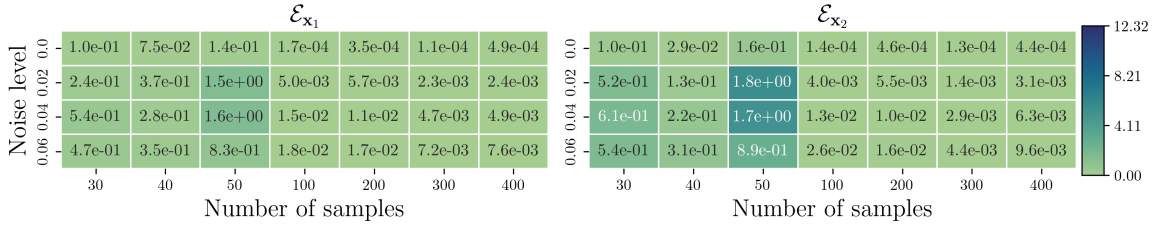
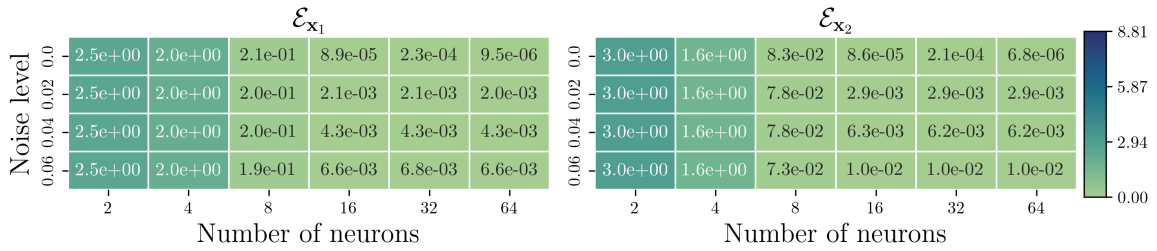
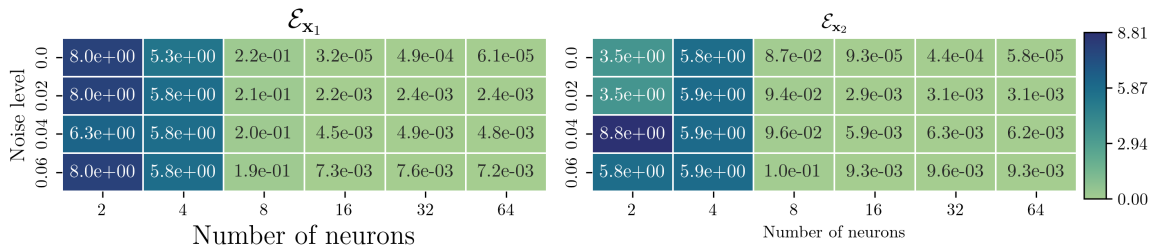
Figure 3: Linear oscillator: Estimated coefficients during the training loop for *iNeural-SINDy*, *DeePyMoD* and *RK4-SINDy*.

the quality of the learned governing equations in [Figure 4](#). Note that the error criterion defined in (15) is used. Each cell shows the error corresponding to sample sizes and noise levels. By comparing the simulation results, we notice that *DeePyMoD* performs better for low data regime, but as the number of data is increased, both *iNeural-SINDy* and *DeePyMoD* perform similarly.

- **Scene_B:** In this case, we consider a DNN architecture with three hidden layers but vary the number of neurons at each layer from 2 to 64. Again, we consider various noise levels. We take 400 samples in the time interval $[0, 10]$ for a single arbitrary initial condition $(\mathbf{x}_1(0), \mathbf{x}_2(0)) = (5, 2)$. The rest of the settings are the same as mentioned earlier in the simulation setup. By varying the noise levels and number of neurons, we report a comparison between *iNeural-SINDy* and *DeePyMoD* in [Figure 5](#), where each cell shows the error, corresponding to a specific number of neurons and noise level. These comparisons again show that both methodologies perform comparably and learn correct coefficients with similar performance for a large number of neurons as the DNN has more capacities to capture the dynamics present in the data. More interesting, we would like to highlight that both methods do not over-fit as the capacity of the DNN is increased.

5.2. Cubic damped oscillator

The cubic oscillatory system is given by the following equation:

(a) Using *iNeural-SINDy*.(b) Using *DeePyMoD*.Figure 4: Linear oscillator: A comparison of *iNeural-SINDy* and *DeePyMoD* under *Scene_A*.(a) Using *iNeural-SINDy*.(b) Using *DeePyMoD*.Figure 5: Linear oscillator: A comparison of *iNeural-SINDy* and *DeePyMoD* under *Scene_B*

$$\begin{aligned}\dot{\mathbf{x}}_1(t) &= -0.1\mathbf{x}_1^3(t) + 2.0\mathbf{x}_2^3(t), \\ \dot{\mathbf{x}}_2(t) &= -2.0\mathbf{x}_1^3(t) - 0.1\mathbf{x}_2^3(t).\end{aligned}\tag{17}$$

The system consists of two coupled, non-linear differential equations describing the time evolution of two variables, \mathbf{x}_1 and \mathbf{x}_2 . Given noisy data, we aim to recover the governing equations and perform a similar analysis as done for the previous example.

Simulation setup: To generate the training data set, we consider two initial conditions $(\mathbf{x}_1(0), \mathbf{x}_2(0)) = \{(2, 2), (-2, -2)\}$ and collect 800 points in the time interval $t \in [0, 10]$. Our DNN architecture has three hidden layers, each having 32 neurons. We set the number of epoch `max-iter` = 30,000, threshold value `tol` = 0.05. The initial training iteration (`init-iter`) is set to 15,000 with the learning rate 10^{-4} for the DNN parameters and 10^{-3} for the coefficient matrix Ξ^{est} . After the initial training, for

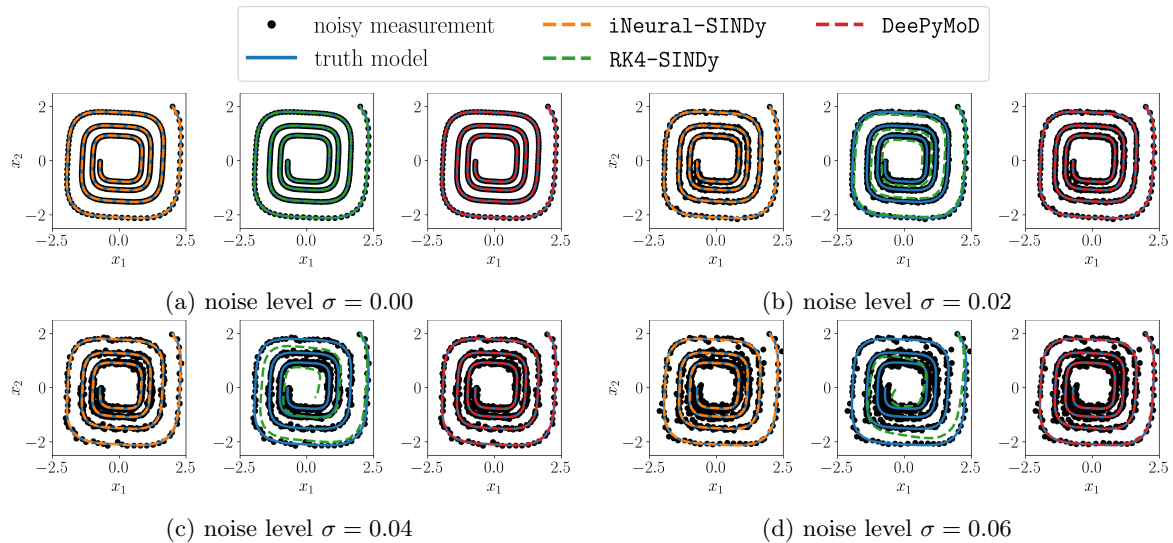


Figure 6: Cubic damped oscillator: A comparison of the estimation with different techniques and noise level

every $q = 5,000$ iterations afterward, we employ the sequential thresholding and update the DNN parameters and Ξ^{est} . The dynamical system is estimated in the space of polynomials up to order three.

Results: To see the performance of these different methodologies under the presence of noise, we consider a Gaussian noise with the standard variance $\sigma = \{0, 0.02, 0.04, 0.06\}$. We report the obtained results in Figure 6 and in Table 2 (see Appendix), and we notice that RK4-SINDy performs poorly for high noise levels, but *iNeural-SINDy* and *DeePyMoD* have competitive performance. Further, in Figure 7, we plot the convergence of the non-zero coefficients as the training progresses for the noise-free case. Here, we again observe a faster convergence for *iNeural-SINDy* as compared to the other two approaches. Next, we investigate performances of *iNeural-SINDy* and *DeePyMoD* for *Scene_A* and *Scene_B*, which are discussed in the following.

- **Scene_A:** We fix a DNN architecture with three hidden layers, each having 32 neurons. We consider a set of noise level with $\sigma = \{0, 0.02, 0.04, 0.06\}$ and a set of sample size $\{30, 40, 50, 100, 200, 300, 400\}$. The data are collected using a random initial condition in the interval $[1, 4]$ for $\{\mathbf{x}_1, \mathbf{x}_2\}$. The rest of the settings are the same as mentioned earlier in the simulation setup. The results are shown in Figure 8, where we notice that for a smaller data set, *iNeural-SINDy* performs slightly better as compared to *DeePyMoD*, whereas for larger data set, it is otherwise.
- **Scene_B:** For this case, we fix the sample size to 400 but consider a DNN architecture with three hidden layers with the number of neurons ranging from 2 to 64. Furthermore, we consider a set of noise levels with $\sigma = \{0, 0.02, 0.04, 0.06\}$. The data are generated as in *Scene_A* and the training setting is also to be as above. The results are depicted in Figure 9, where we observe that *iNeural-SINDy* performs better as compared to *DeePyMoD* for fewer neurons, and as we increase the number of neurons, both methods perform similarly.

5.3. Fitz-Hugh Nagumo system

The Fitz-Hugh Nagumo (FHN) model is a non-linear system of ordinary differential equations that is used to describe the behavior of a biological neuron, see, e.g., [25]. The FHN model is commonly used to study the behavior of biological neurons under different conditions, such as the changes in the external stimulus or variations in the intrinsic properties of the neuron. The set of differential equations that describe the underlying dynamics are as follows:

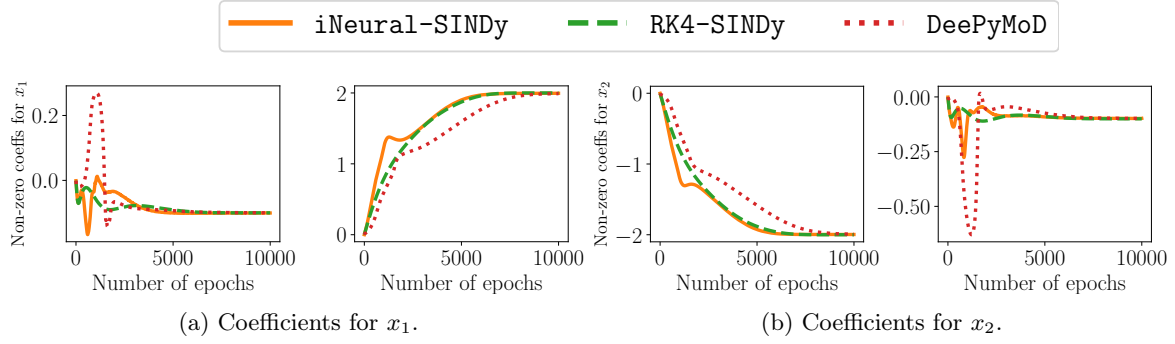


Figure 7: Cubic oscillator: Estimated coefficients during the training loop for *iNeural-SINDy*, *DeePyMoD* and *RK4-SINDy*.

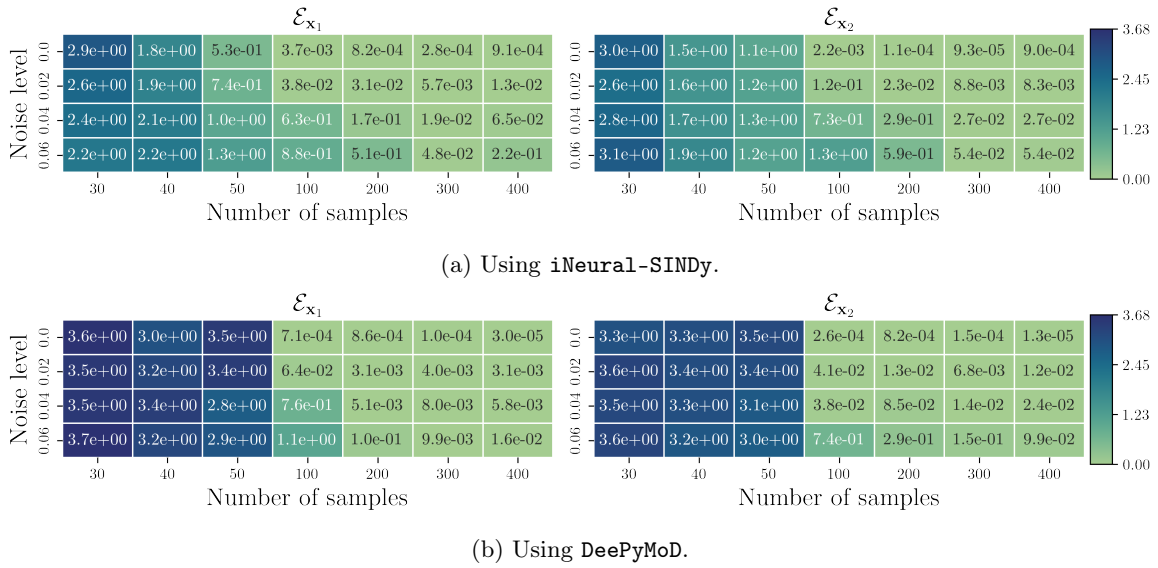


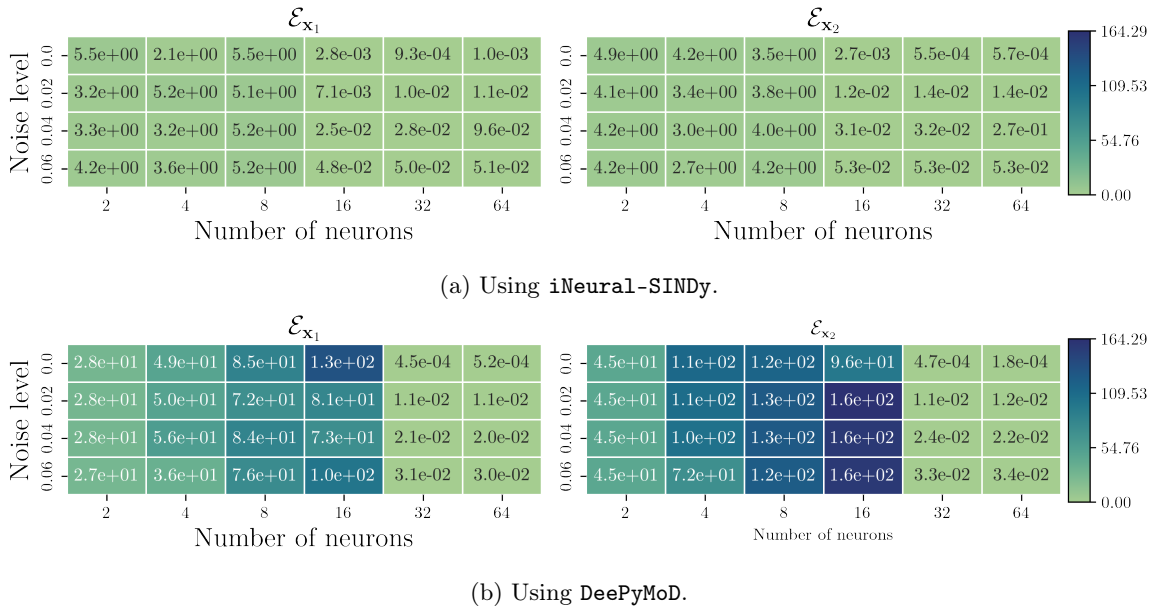
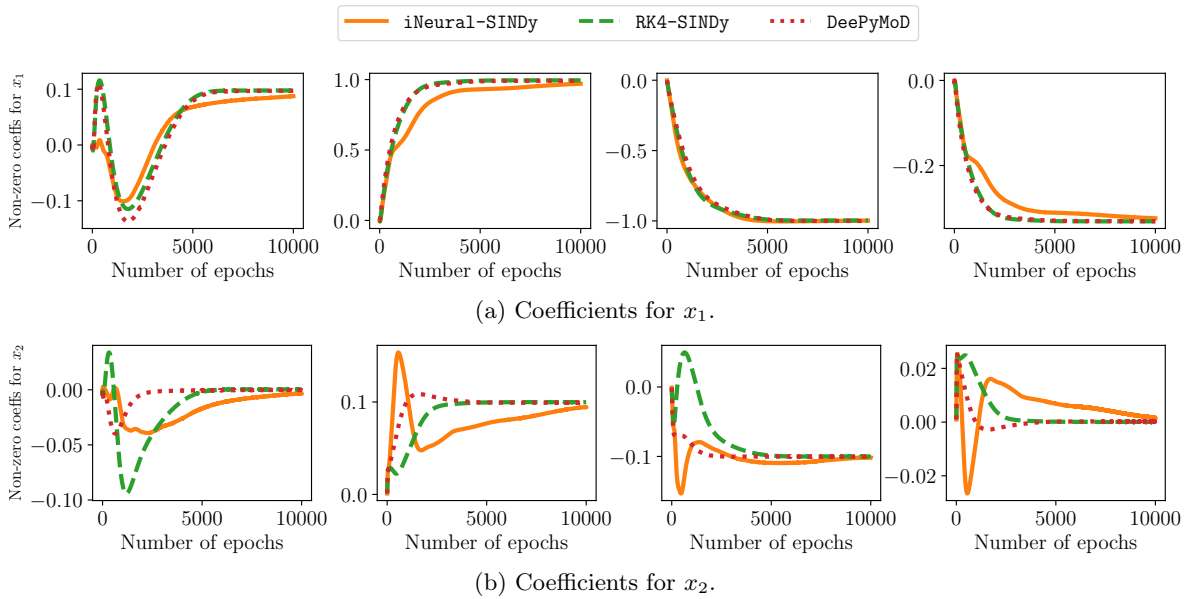
Figure 8: Cubic oscillator: A comparison of *iNeural-SINDy* and *DeePyMoD* under *Scene_A*.

$$\begin{aligned}\dot{\mathbf{x}}_1(t) &= 1.0\mathbf{x}_1(t) - 1.0\mathbf{x}_2(t) - \frac{1}{3}\mathbf{x}_1^3(t) + 0.1, \\ \dot{\mathbf{x}}_2(t) &= 0.1\mathbf{x}_1(t) - 0.1\mathbf{x}_2(t).\end{aligned}\tag{18}$$

Simulation setup: For this simulation example, we consider two initial conditions $(\mathbf{x}_1(0), \mathbf{x}_2(0)) = \{(2, 1.5), (1.5, 2)\}$ and take 400 data points in the time interval $t \in [0, 200]$. The DNN architecture has three hidden layers with 32 neurons. We set the number of epoch `max-iter` = 50,000 and threshold value `tol` = 0.05. The number of iterations for the initial training is set to 15,000 with the learning rate 10^{-4} for the DNN parameters and 10^{-3} for the coefficient matrix Ξ^{est} . After the initial training, we employ the sequential thresholding after each $q = 5,000$ iterations. We aim to learn the underlying governing equations in the space of polynomials with degrees up to order three.

Results: Converse to the results that we earned in the previous two examples, for the FHN, *iNeural-SINDy* has a slower convergence rate compared to *DeePyMoD* and *RK4-SINDy*, see [Figure 10](#).

For this example, we again make a similar observation (see [Figure 11](#), and [Table 3](#) in Appendix), where we notice that *iNeural-SINDy* and *DeePyMoD* exhibit similar performances for lower noise levels, but for the higher noise values, (see the results for $\sigma = 0.08$ [Table 3](#)), *iNeural-SINDy* tends to outperform *DeePyMoD*. Moreover, *RK4-SINDy* clearly fails for high noise levels. However, converse to the results reported in the previous two examples, for this example, we notice a slower convergence

Figure 9: Cubic oscillator: A comparison of *iNeural-SINDy* and *DeePyMoD* under *Scene_B*Figure 10: Fitz-Hugh Nagumo: Estimated coefficients during the training loop for *iNeural-SINDy*, *DeePyMoD* and *RK4-SINDy*.

of *iNeural-SINDy* compared to *DeePyMoD* and *RK4-SINDy*; see, [Figure 10](#). But we highlight that *iNeural-SINDy* can identify governing equations for highly noisy data, as stated earlier. Next, we compare the performances of *iNeural-SINDy* and *DeePyMoD* under *Scene_A* and *Scene_B*.

- **Scene_A:** In this case, we consider a fixed DNN architecture with three hidden layers, each consisting of 32 neurons. The different noise levels $\{0.0, 0.02, 0.04, 0.06\}$ are considered, while the sample size is considered in the range from 150 to 450 with an increment of 50. Here, a single initial condition is used for data collection; that is, $(\mathbf{x}_1(0), \mathbf{x}_2(0)) = (3, 2)$. The rest of the settings are the same as mentioned earlier for this example. The results are shown in [Figure 12](#), where we notice that *DeePyMoD* outperforms *iNeural-SINDy* and has a better performance.
- **Scene_B:** Here, we conduct a study where we keep the number of samples fixed at 400, obtained

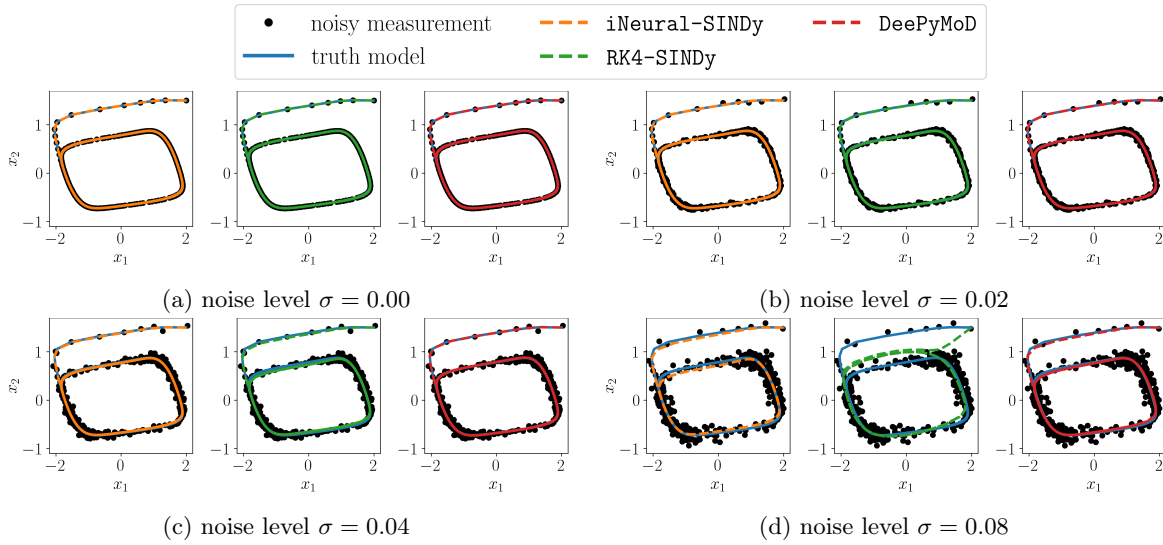
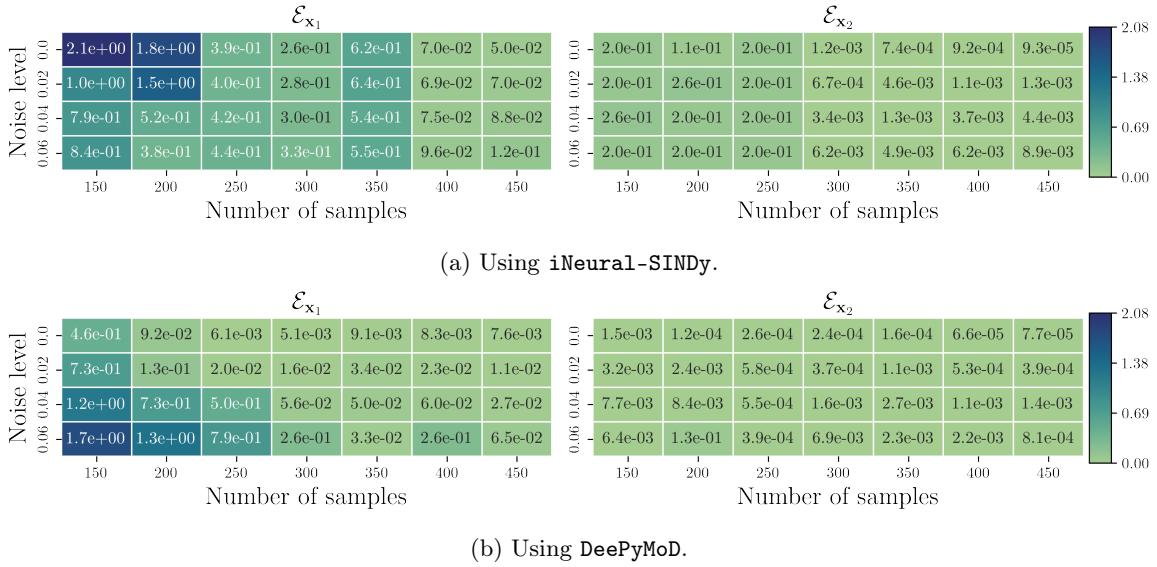


Figure 11: Fitz-Hugh Nagumo: Comparison of the estimation with different techniques and noise level


Figure 12: Fitz-Hugh Nagumo: A comparison of *iNeural-SINDy* and *DeePyMoD* under Scene_A.

using the initial condition $(\mathbf{x}_1(0), \mathbf{x}_2(0)) = (3, 2)$. The DNN architecture is designed to have three hidden layers. We aim to explore how *iNeural-SINDy* and *DeePyMoD* perform under different combinations of neurons for each layer and noise level. The training settings for each case remain the same, as mentioned earlier. The outcomes are presented in the heat-map depicted in Figure 13, where we notice that both *DeePyMoD* and *iNeural-SINDy* almost have the same performance in all the settings.

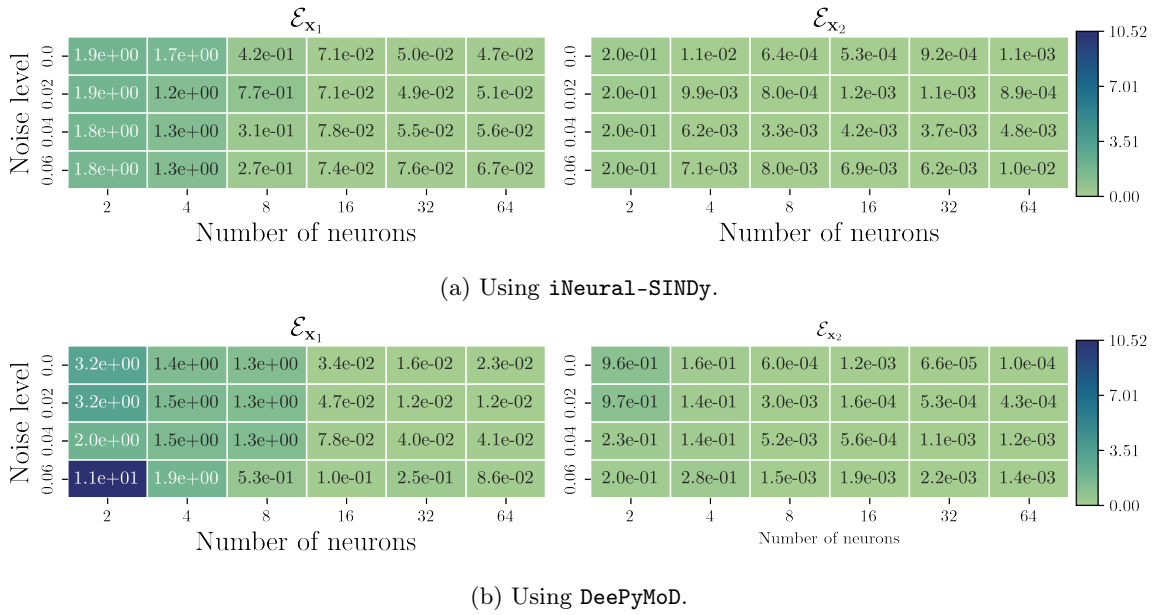
5.4. Chaotic Lorenz system

The chaotic Lorenz system is a set of three differential equations as follows [49]:

$$\dot{\mathbf{x}}_1(t) = \gamma(\mathbf{x}_2(t) - \mathbf{x}_1(t)), \quad (19a)$$

$$\dot{\mathbf{x}}_2(t) = \mathbf{x}_1(t)(\rho - \mathbf{x}_3(t)) - \mathbf{x}_2(t), \quad (19b)$$

$$\dot{\mathbf{x}}_3(t) = \mathbf{x}_1(t)\mathbf{x}_2(t) - \beta\mathbf{x}_3(t), \quad (19c)$$

Figure 13: Fitz-Hugh Nagumo: A comparison of *iNeural-SINDy* and *DeePyMoD* under *Scene_B*

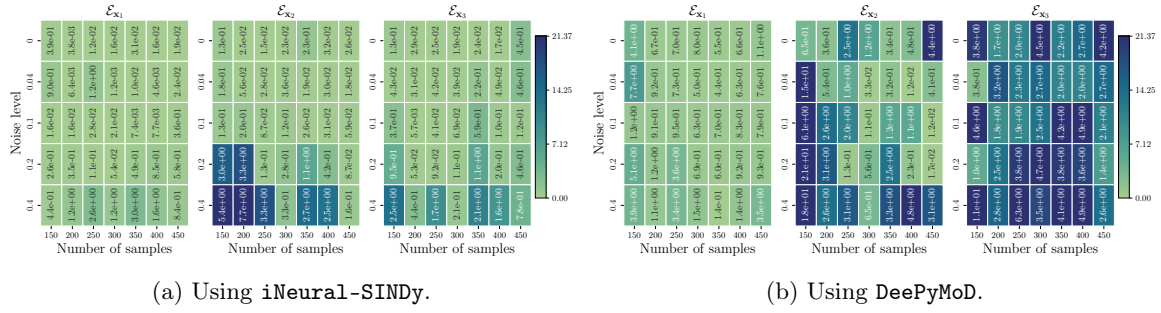
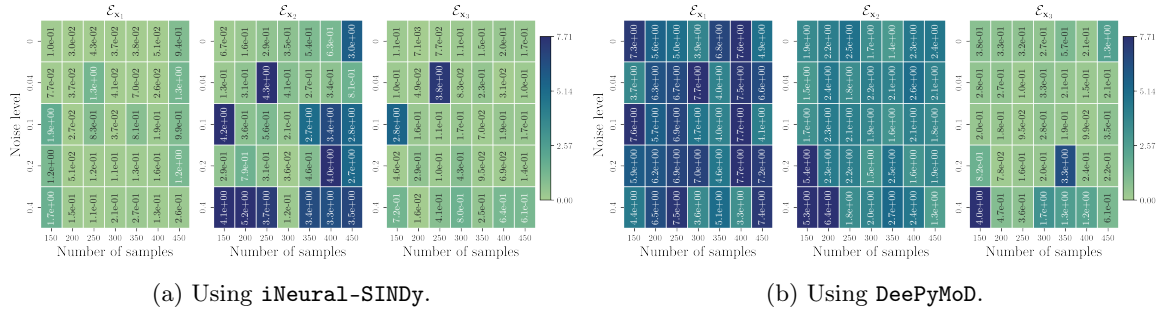
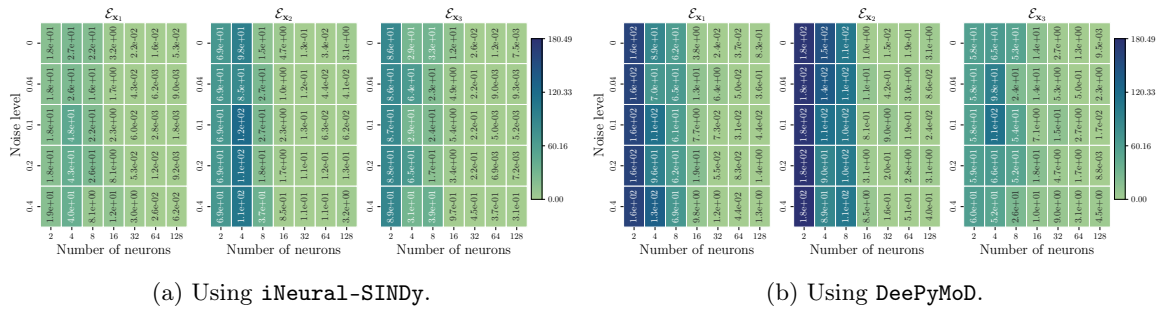
where the parameters γ , ρ , and β are positive constants with associated standard values $\gamma = 10$, $\rho = 28$, $\beta = \frac{8}{3}$. The Lorenz system is a classic example of a chaotic system, which means that small differences in the initial conditions can lead to vastly different outcomes over time. It is a widely used benchmark example for discovering governing equations [11].

Simulation setup: We collect our data in the time interval $t \in [0, 10]$ with a sample size of 200 for three different initial conditions $(\mathbf{x}_1(0), \mathbf{x}_2(0), \mathbf{x}_3(0)) = \{(-8, 7, 27), (-6, 6, 25), (-9, 8, 22)\}$. The DNN architecture has three hidden layers, each having 64 neurons. We set the number of iterations `max-iter` = 35,000, and threshold value `tol` = 0.2. We set the number of iterations for the initial training to `init-iter` = 10,000, and the learning rate $7 \cdot 10^{-4}$ for the DNN parameters and 10^{-2} for the coefficient matrix Ξ^{est} . After finishing the initial iterations, we employ sequential thresholding after every $q = 3,000$ iterations. Moreover, after each sequential thresholding, we reset the learning rate for DNN parameters to $5 \cdot 10^{-6}$ and for the coefficient matrix Ξ^{est} to 10^{-2} . The governing equations are estimated by constructing a dictionary with polynomials up to degree two.

Since the magnitude of $\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\}$ for the Lorenz example can be large, we consider scaling the \mathbf{x} 's using a scaling factor α . Note that such scaling does not affect the interaction between different \mathbf{x} 's; thus, the sparsity pattern remains the same as well. However, it is observed that improving the condition number of the dictionary matrix enhances the estimate of the coefficients and helps us to determine the right governing equations.

Results: We conduct experiments using a scaling factor $\alpha = 0.1$. Further, we aim to learn governing equations from the noisy data with noise levels of $\sigma = \{0, 0.04, 0.1, 0.2, 0.4\}$. We report the obtained results in Table 4, where we notice that *iNeural-SINDy* and *DeePyMoD* yield similar performance except for the case of higher noise level (e.g., see the results for $\sigma = 0.4$), where *iNeural-SINDy* recovered the equations better. However, *RK4-SINDy* performs poorly for the higher noise levels. Next, we conduct a performance analysis of *iNeural-SINDy* and *DeePyMoD* for *Scene_A* and *Scene_B*. We note that in both scenarios, the training data are generated using a single initial condition $(\mathbf{x}_1(0), \mathbf{x}_2(0), \mathbf{x}_3(0)) = (-8, 7, 27)$.

- **Scene_A:** We compare *iNeural-SINDy* and *DeePyMoD* under *Scene_A*. We also investigate the effect of the scaling factor α and consider two values of it, i.e., $\alpha = \{0.1, 1\}$. We fix the DNN architecture to have three hidden layers, each having 64 neurons. We consider different sample sizes and noise levels to compare the performance of *iNeural-SINDy* and *DeePyMoD*. For $\alpha = 0.1$,

Figure 14: Lorenz example: A comparison of *iNeural-SINDy* and DeePyMoD under Scene_A with the scaling factor $\alpha = 0.1$ Figure 15: Lorenz example: A comparison of *iNeural-SINDy* and DeePyMoD under Scene_A with the scaling factor $\alpha = 1$ Figure 16: Lorenz example: A comparison of *iNeural-SINDy* and DeePyMoD under Scene_B with the scaling factor $\alpha = 0.1$.

we show the results in Figure 14, where we notice that *iNeural-SINDy* outperforms DeePyMoD in most cases. A similar observation is made for $\alpha = 1$, which is reported in Figure 15. For these experiments, it is hard to conclude the effect of the scaling factors, as we notice that in some cases, the scaling improves the performance, and in some cases, it is not the case.

- **Scene_B:** In this case, we fix the sample size to 400. We also fix the number of hidden layers for the DNN architecture to three but vary the number of neurons in each layer. We also conduct experiments to see the effect of the scaling factor in this case as well. The results for $\alpha = 1$ and $\alpha = 0.1$ are shown in Figure 16 and Figure 17, respectively. These heat maps indicate the outperformance of *iNeural-SINDy* in most cases. We also observe that for a larger number of neurons, the scaling factor $\alpha = 0.1$ slightly performs better compared to scaling factor $\alpha = 1$ in both *iNeural-SINDy* as well as DeePyMoD.

A comparison of *iNeural-SINDy* with WEAK-SINDy: Beside our previous comprehensive study, we next compare *iNeural-SINDy* with Weak-SINDy, which also does not require any estimate of

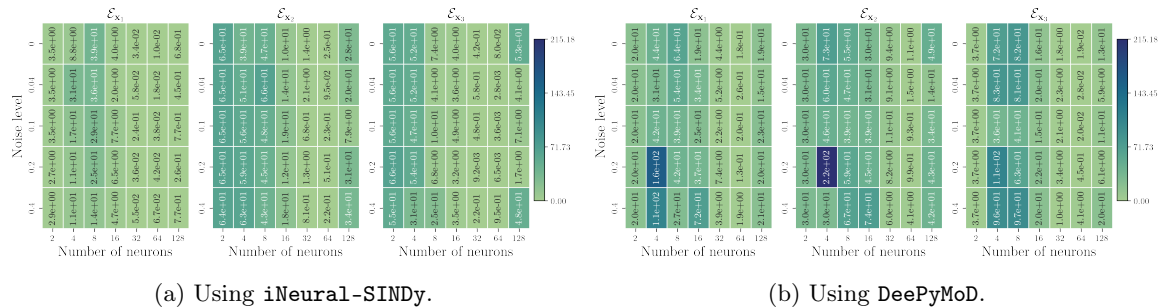


Figure 17: Lorenz example: A comparison of **iNeural-SINDy** and **DeePyMoD** under **Scene_B** with the scaling factor $\alpha = 1$

derivatives using noisy data; for more details on **Weak-SINDy**, we refer to [28].

For this study, we again consider the same initial condition as used for **Scene_A** and **Scene_B**. We take 2000 data points in the time interval $[0, 10]$, which are corrupted using different noise levels $\sigma = \{0, 0.02, 0.08, 0.1\}$. To discover the governing equations, we consider a dictionary of polynomials up to degree two. For training **iNeural-SINDy** we use the same setting as discussed in Section 5.4. For **Weak-SINDy**, we consider the code provided by the authors¹. We report the results in Table 5, which indicates that **iNeural-SINDy** outperforms **Weak-SINDy** in the presence of high noise.

6. Conclusions

In this work, we proposed a methodology, namely **iNeural-SINDy**, to discover governing equations using noisy and scarce data. It consists of three main components—these are: (a) learning an implicit representation based on given noisy data using a deep neural network, (b) setting up a sparse regression problem inspired by **SINDy** [11], discovering governing equations, and (c) utilize an integral form of differential equations. We have combined all these components innovatively to learn governing equations from noisy data. Particularly, we highlight that we leverage the implicit representation using neural networks to estimate the derivative using automatic differential to avoid any numerical derivative estimation using noisy data. We have shown how **iNeural-SINDy** can be employed when data are collected using multiple trajectories. Furthermore, we have presented an extensive comparison of the proposed methodology with **RK4-SINDy** [21] and **DeePyMoD** [31], where we noticed that **iNeural-SINDy** clearly out-performed **RK4-SINDy**, and in many cases, **iNeural-SINDy** also yielded better or comparable results as compared to **DeePyMoD**, expect for the **FHN** example. We also compared **iNeural-SINDy** with **Weak-SINDy** using the **Lorenz** example, where we noticed a better performance of **iNeural-SINDy**. In the future, we would like to extend the proposed framework to the identification of parametric and control-driven dynamical systems. We also like to combine the idea of the ensemble discussed in [29] to further improve the quality of learned governing equations.

References

- [1] L. Ljung, *System Identification: Theory for the User*. Prentice Hall, NJ, 1999.
- [2] P. Van Overschee and B. de Moor, *Subspace Identification of Linear Systems: Theory, Implementation, Applications*. Kluwer Academic Publishers, 1996.
- [3] A. K. Tangirala, *Principles of System Identification Theory and Practice*. Crc Press, 2018.
- [4] S. N. Kumpati and P. Kannan, “Identification and control of dynamical systems using neural networks,” *IEEE Trans. Neural Networks*, vol. 1, no. 1, pp. 4–27, 1990.
- [5] J. A. Suykens, J. P. Vandewalle, and B. L. de Moor, *Artificial Neural Networks for Modelling and Control of Non-Linear Systems*. Springer, 1996.

¹https://github.com/MathBioCU/WSINDy_ODE/tree/master

-
- [6] J. P. Crutchfield and B. S. McNamara, “Equations of motion from a data series,” *Complex Sys.*, vol. 1, no. 417-452, p. 121, 1987.
- [7] M. D. Schmidt, R. R. Vallabhajosyula, J. W. Jenkins, J. E. Hood, A. S. Soni, J. P. Wikswo, and H. Lipson, “Automated refinement and inference of analytical models for metabolic networks,” *Phy. Biology*, vol. 8, no. 5, p. 055011, 2011.
- [8] M. Schmidt and H. Lipson, “Distilling free-form natural laws from experimental data,” *Science*, vol. 324, no. 5923, pp. 81–85, 2009.
- [9] W.-X. Wang, R. Yang, Y.-C. Lai, V. Kovanis, and C. Grebogi, “Predicting catastrophes in nonlinear dynamical systems by compressive sensing,” *Phys. Rev. Lett.*, vol. 106, no. 15, p. 154101, 2011.
- [10] H. Schaeffer, R. Caflisch, C. D. Hauck, and S. Osher, “Sparse dynamics for partial differential equations,” *Proc. Nat. Acad. Sci. U.S.A.*, vol. 110, no. 17, pp. 6634–6639, 2013.
- [11] S. L. Brunton, J. L. Proctor, and J. N. Kutz, “Discovering governing equations from data by sparse identification of nonlinear dynamical systems,” *Proc. Nat. Acad. Sci. U.S.A.*, vol. 113, no. 15, pp. 3932–3937, 2016.
- [12] J. C. Loiseau and S. L. Brunton, “Constrained sparse Galerkin regression,” *J. Fluid Mech.*, vol. 838, pp. 42–67, 2018.
- [13] M. Dam, M. Brøns, J. Juul Rasmussen, V. Naulin, and J. S. Hesthaven, “Sparse identification of a predator-prey system from simulation data of a convection model,” *Phys. Plasmas*, vol. 24, no. 2, 2017.
- [14] S. Beetham and J. Capecelatro, “Formulating turbulence closures using sparse regression with embedded form invariance,” *Phys. Rev. Fluids*, vol. 5, no. 8, p. 084611, 2020.
- [15] L. Zanna and T. Bolton, “Data-driven equation discovery of ocean mesoscale closures,” *Geophys. Res. Lett.*, vol. 47, no. 17, p. e2020GL088376, 2020.
- [16] M. Sorokina, S. Sygletos, and S. Turitsyn, “Sparse identification for nonlinear optical communication systems: SINO method,” *Optics Express*, vol. 24, no. 26, pp. 30 433–30 443, 2016.
- [17] L. Boninsegna, F. Nüske, and C. Clementi, “Sparse learning of stochastic dynamical equations,” *J. Chem. Phys.*, vol. 148, no. 24, p. 241723, 2018.
- [18] S. Thaler, L. Paehler, and N. A. Adams, “Sparse identification of truncation errors,” *J. Comput. Phys.*, vol. 397, p. 108851, 2019.
- [19] E. Kaiser, J. N. Kutz, and S. L. Brunton, “Sparse identification of nonlinear dynamics for model predictive control in the low-data limit,” *Proc. Roy. Soc. Edinburgh Sect. A*, vol. 474, no. 2219, p. 20180335, 2018.
- [20] K. Kaheman, J. N. Kutz, and S. L. Brunton, “SINDy-PI: a robust algorithm for parallel implicit sparse identification of nonlinear dynamics,” *Proc. Roy. Soc. Edinburgh Sect. A*, vol. 476, no. 2242, p. 20200279, 2020.
- [21] P. Goyal and P. Benner, “Discovery of nonlinear dynamical systems using a Runge-Kutta inspired dictionary-based sparse regression approach,” *Philos. Trans. Roy. Soc. A*, vol. 478, no. 2262, p. 20210883, 2022.
- [22] A. A. Kaptanoglu, J. L. Callahan, A. Aravkin, C. J. Hansen, and S. L. Brunton, “Promoting global stability in data-driven models of quadratic nonlinear dynamics,” *Phys. Rev. Fluids*, vol. 6, no. 9, p. 094401, 2021.
- [23] J. L. Callahan, J.-C. Loiseau, G. Rigas, and S. L. Brunton, “Nonlinear stochastic modelling with langevin regression,” *Proc. Roy. Soc. A*, vol. 477, no. 2250, p. 20210092, 2021.

- [24] S. Zhang and G. Lin, “Robust data-driven discovery of governing physical laws with error bars,” *Proc. Roy. Soc. A: Math., Phys. Eng. Sci.*, vol. 474, no. 2217, p. 20180305, 2018.
- [25] H. Schaeffer and S. G. McCalla, “Sparse model selection via integral terms,” *Phys. Rev. E*, vol. 96, no. 2, p. 023302, 2017.
- [26] S. H. Kang, W. Liao, and Y. Liu, “Ident: Identifying differential equations with numerical time evolution,” *J. Sci. Comput.*, vol. 87, pp. 1–27, 2021.
- [27] R. T. Keller and Q. Du, “Discovery of dynamics using linear multistep methods,” *SIAM J. Numer. Anal.*, vol. 59, no. 1, pp. 429–455, 2021.
- [28] D. A. Messenger and D. M. Bortz, “Weak SINDy: Galerkin-based data-driven model selection,” *Multiscale Model. Simul.*, vol. 19, no. 3, pp. 1474–1497, 2021.
- [29] 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,” *Proc. Roy. Soc. A*, vol. 478, no. 2260, p. 20210904, 2022.
- [30] K. Kaheman, S. L. Brunton, and J. N. Kutz, “Automatic differentiation to simultaneously identify nonlinear dynamics and extract noise probability distributions from data,” *Mach. Learn.: Sci. Tech.*, vol. 3, no. 1, p. 015031, 2022.
- [31] G.-J. Both, S. Choudhury, P. Sens, and R. Kusters, “Deepmod: Deep learning for model discovery in noisy data,” *J. Comput. Phys.*, vol. 428, p. 109985, 2021.
- [32] R. T. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, “Neural ordinary differential equations,” in *Adv. Neural Inform. Process. Sys.*, 2018, pp. 6571–6583.
- [33] P. Goyal and P. Benner, “Neural ordinary differential equations with irregular and noisy data,” *Roy. Soc. Open Sci.*, vol. 10, no. 7, p. 221475, 2023.
- [34] S. L. Brunton, J. H. Tu, I. Bright, and J. N. Kutz, “Compressive sensing and low-rank libraries for classification of bifurcation regimes in nonlinear dynamical systems,” *SIAM J. Appl. Dyn. Syst.*, vol. 13, no. 4, pp. 1716–1732, 2014.
- [35] A. Mackey, H. Schaeffer, and S. Osher, “On the compressive spectral method,” *Multiscale Model. Simul.*, vol. 12, no. 4, pp. 1800–1827, 2014.
- [36] T. Hastie, R. Tibshirani, J. H. Friedman, and J. H. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2009, vol. 2.
- [37] R. Tibshirani, “Regression shrinkage and selection via the lasso,” *J. Roy. Statist. Soc.: Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, 1996.
- [38] A. Beck and Y. C. Eldar, “Sparsity constrained nonlinear optimization: Optimality conditions and algorithms,” *SIAM J. Optim.*, vol. 23, no. 3, pp. 1480–1509, 2013.
- [39] Z. Yang, Z. Wang, H. Liu, Y. Eldar, and T. Zhang, “Sparse nonlinear regression: Parameter estimation under nonconvexity,” in *Intern. Conf. on Mach. Learn.* PMLR, 2016, pp. 2472–2481.
- [40] L. Zhang and H. Schaeffer, “On the convergence of the SINDy algorithm,” *Multiscale Model. Simul.*, vol. 17, no. 3, pp. 948–972, 2019.
- [41] S. H. Rudy, S. L. Brunton, J. L. Proctor, and J. N. Kutz, “Data-driven discovery of partial differential equations,” *Sci. Adv.*, vol. 3, no. 4, p. e1602614, 2017.
- [42] P. A. Reinbold, D. R. Gurevich, and R. O. Grigoriev, “Using noisy or incomplete data to discover models of spatiotemporal dynamics,” *Phys. Rev. E*, vol. 101, no. 1, p. 010203, 2020.
- [43] J. C. Sprott and J. C. Sprott, *Chaos and time-series analysis*. Oxford university press Oxford, 2003, vol. 69.

- [44] R. T. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, “Neural ordinary differential equations,” *Adv. Neural Inform. Process. Sys.*, vol. 31, 2018.
- [45] J. C. Butcher, *Numerical methods for ordinary differential equations*. John Wiley & Sons, 2016.
- [46] E. Buckwar and R. Winkler, “Multistep methods for SDEs and their application to problems with small noise,” *SIAM J. Numer. Anal.*, vol. 44, no. 2, pp. 779–803, 2006.
- [47] V. Sitzmann, J. Martel, A. Bergman, D. Lindell, and G. Wetzstein, “Implicit neural representations with periodic activation functions,” *Adv. Neural Inform. Process. Sys.*, vol. 33, pp. 7462–7473, 2020.
- [48] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.
- [49] N. Kuznetsov and V. Reitmann, *Attractor dimension estimates for dynamical systems: theory and computation*. Springer, 2020.

A. Appendix

In this appendix, we present a comparison of learned governing equations using various methods. Moreover, the code and examples are available at the following Git repository ².

Table 1: Estimated coefficients for linear damped oscillator

Noise level	Learned equations		
	iNeural-SINDy	DeepPyMoD	RK4-SINDy [21]
0.00	$\dot{x}_1(t) = -0.100x_1(t) + 2.000x_2(t)$ $\dot{x}_2(t) = -2.000x_1(t) - 0.100x_2(t)$	$\dot{x}_1(t) = -0.100x_1(t) + 2.000x_2(t)$ $\dot{x}_2(t) = -2.000x_1(t) - 0.100x_2(t)$	$\dot{x}_1(t) = -0.100x_1(t) + 2.000x_2(t)$ $\dot{x}_2(t) = -2.000x_1(t) - 0.100x_2(t)$
0.02	$\dot{x}_1(t) = -0.100x_1(t) + 2.000x_2(t)$ $\dot{x}_2(t) = -2.000x_1(t) - 0.100x_2(t)$	$\dot{x}_1(t) = -0.100x_1(t) + 2.000x_2(t)$ $\dot{x}_2(t) = -2.000x_1(t) - 0.099x_2(t)$	$\dot{x}_1(t) = -0.103x_1(t) + 1.997x_2(t)$ $\dot{x}_2(t) = -2.003x_1(t) - 0.104x_2(t)$
0.04	$\dot{x}_1(t) = -0.105x_1(t) + 2.007x_2(t)$ $\dot{x}_2(t) = -1.994x_1(t) - 0.094x_2(t)$	$\dot{x}_1(t) = -0.098x_1(t) + 2.000x_2(t)$ $\dot{x}_2(t) = -2.000x_1(t) - 0.099x_2(t)$	$\dot{x}_1(t) = -0.109x_1(t) + 1.989x_2(t)$ $\dot{x}_2(t) = -2.009x_1(t) - 0.110x_2(t)$
0.08	$\dot{x}_1(t) = -0.097x_1(t) + 1.997x_2(t)$ $\dot{x}_2(t) = -2.003x_1(t) - 0.106x_2(t)$	$\dot{x}_1(t) = -0.1000x_1(t) + 1.996x_2(t)$ $\dot{x}_2(t) = -2.005x_1(t) - 0.102x_2(t)$	$\dot{x}_1(t) = -0.170x_1(t) + 1.916x_2(t)$ $\dot{x}_2(t) = -2.073x_1(t) - 0.177x_2(t)$

Table 2: Estimated coefficients for Cubic damped oscillator

Noise level	Estimated System		
	iNeural-SINDy	DeepPyMoD	RK4-SINDy [21]
0.00	$\dot{x}_1(t) = -0.100x_1^3(t) + 2.000x_2^3(t)$ $\dot{x}_2(t) = -2.000x_1^3(t) - 0.100x_2^3(t)$	$\dot{x}_1(t) = -0.100x_1^3(t) + 2.000x_2^3(t)$ $\dot{x}_2(t) = -2.000x_1^3(t) - 0.100x_2^3(t)$	$\dot{x}_1(t) = -0.099x_1^3(t) + 2.000x_2^3(t)$ $\dot{x}_2(t) = -2.000x_1^3(t) - 0.100x_2^3(t)$
0.02	$\dot{x}_1(t) = -0.103x_1^3(t) + 1.996x_2^3(t)$ $\dot{x}_2(t) = -1.997x_1^3(t) - 0.099x_2^3(t)$	$\dot{x}_1(t) = -0.098x_1^3(t) + 2.005x_2^3(t)$ $\dot{x}_2(t) = -1.995x_1^3(t) - 0.102x_2^3(t)$	$\dot{x}_1(t) = -0.102x_1^3(t) + 2.004x_2^3(t)$ $\dot{x}_2(t) = -1.987x_1(t) - 0.054x_1(t)x_2^2(t) - 0.119x_2^3(t)$
0.04	$\dot{x}_1(t) = -0.105x_1^3(t) + 1.994x_2^3(t)$ $\dot{x}_2(t) = -1.989x_1^3(t) - 0.0984x_2^3(t)$	$\dot{x}_1(t) = -0.0974x_1^3(t) + 2.000x_2^3(t)$ $\dot{x}_2(t) = -2.015x_1^3(t) - 0.104x_2^3(t)$	$\dot{x}_1(t) = 0.076x_2(t) - 0.147x_1^3(t)$ $+ 0.059x_1(t)x_2^2(t) + 1.907x_2^3(t)$ $\dot{x}_2(t) = -0.051x_1^2(t) - 0.059x_1(t)x_2(t)$ $- 2.042x_1^3(t) + 0.112x_1^2(t)x_2(t) - 0.167x_2^3(t)$
0.06	$\dot{x}_1(t) = -0.100x_1^3(t) + 1.947x_2^3(t)$ $\dot{x}_2(t) = -1.986x_1^3(t) - 0.112x_2^3(t)$	$\dot{x}_1(t) = -0.097x_1^3(t) + 1.972x_2^3(t)$ $\dot{x}_2(t) = -2.040x_1^3(t) - 0.105x_2^3(t)$	$\dot{x}(t) = 0.156x_1(t) + 0.167x_2(t) - 0.071x_2^2(t)$ $- 0.294x_1^3(t) - 0.165x_1(t)x_2^2(t) + 1.688x_2^3(t)$ $\dot{x}_2(t) = 0.107x_1(t) - 0.181x_2(t) + 0.084x_1(t)x_2(t)$ $- 0.084x_2^2(t) - 2.169x_1^3(t) - 0.134x_1(t)x_2^2(t)$ $- 0.355x_2^3(t)$

²https://github.com/Ali-Forootani/iNeural_SINDy_paper/tree/main

Table 3: Estimated coefficients for Fitz-Hugh Nagumo

Noise level	Estimated System		
	iNeural-SINDy	DeepPyMod	RK4-SINDy [21]
0.00	$\dot{\mathbf{x}}_1(t) = 0.989\mathbf{x}_1(t) - 0.993\mathbf{x}_2(t) - 0.329\mathbf{x}_1^3(t) + 0.100$ $\dot{\mathbf{x}}_2(t) = 0.100\mathbf{x}_1(t) - 0.099\mathbf{x}_2(t)$	$\dot{\mathbf{x}}_1(t) = 0.992\mathbf{x}_1(t) - 0.994\mathbf{x}_2(t) - 0.330\mathbf{x}_1^3(t) + 0.100$ $\dot{\mathbf{x}}_2(t) = 0.100\mathbf{x}_1(t) - 0.100\mathbf{x}_2(t)$	$\dot{\mathbf{x}}_1(t) = 0.986\mathbf{x}_1(t) - 0.996\mathbf{x}_2(t) - 0.328\mathbf{x}_1^3(t) + 0.0993$ $\dot{\mathbf{x}}_2(t) = 0.100\mathbf{x}_1(t) - 0.099\mathbf{x}_2(t)$
0.02	$\dot{\mathbf{x}}_1(t) = 0.993\mathbf{x}_1(t) - 0.997\mathbf{x}_2(t) - 0.330\mathbf{x}_1^3(t) + 0.100$ $\dot{\mathbf{x}}_2(t) = 0.100\mathbf{x}_1(t) - 0.100\mathbf{x}_2(t)$	$\dot{\mathbf{x}}_1(t) = 0.994\mathbf{x}_1(t) - 0.996\mathbf{x}_2(t) - 0.330\mathbf{x}_1^3(t) + 0.100$ $\dot{\mathbf{x}}_2(t) = 0.100\mathbf{x}_1(t) - 0.100\mathbf{x}_2(t)$	$\dot{\mathbf{x}}_1(t) = 1.000\mathbf{x}_1(t) - 1.000\mathbf{x}_2(t) - 0.333\mathbf{x}_1^3(t) + 0.100$ $\dot{\mathbf{x}}_2(t) = 0.100\mathbf{x}_1(t) - 0.101\mathbf{x}_2(t)$
0.04	$\dot{\mathbf{x}}_1(t) = 0.997\mathbf{x}_1(t) - 0.999\mathbf{x}_2(t) - 0.332\mathbf{x}_1^3(t) + 0.101$ $\dot{\mathbf{x}}_2(t) = 0.100\mathbf{x}_1(t) - 0.102\mathbf{x}_2(t)$	$\dot{\mathbf{x}}_1(t) = 0.967\mathbf{x}_1(t) - 0.973\mathbf{x}_2(t) - 0.321\mathbf{x}_1^3(t) + 0.098$ $\dot{\mathbf{x}}_2(t) = 0.100\mathbf{x}_1(t) - 0.100\mathbf{x}_2(t)$	$\dot{\mathbf{x}}_1(t) = 1.000\mathbf{x}_1(t) - 1.010\mathbf{x}_2(t) - 0.335\mathbf{x}_1^3(t) + 0.1$ $\dot{\mathbf{x}}_2(t) = 0.101\mathbf{x}_1(t) - 0.109\mathbf{x}_2(t)$
0.08	$\dot{\mathbf{x}}_1(t) = 0.978\mathbf{x}_1(t) - 0.985\mathbf{x}_2(t) - 0.324\mathbf{x}_1^3(t) + 0.097$ $\dot{\mathbf{x}}_2(t) = 0.100\mathbf{x}_1(t) - 0.112\mathbf{x}_2(t)$	$\dot{\mathbf{x}}_1(t) = 0.908\mathbf{x}_1(t) - 0.941\mathbf{x}_2(t) - 0.297\mathbf{x}_1^3(t) + 0.096$ $\dot{\mathbf{x}}_2(t) = 0.102\mathbf{x}_1(t) - 0.106\mathbf{x}_2(t)$	$\dot{\mathbf{x}}_1(t) = 0.336\mathbf{x}_1(t) - 0.081\mathbf{x}_2^2(t) - 0.076\mathbf{x}_1^3 - 0.117\mathbf{x}_1^2(t)\mathbf{x}_2(t) - 0.139\mathbf{x}_1(t)\mathbf{x}_2^2(t) - 0.113\mathbf{x}_2^3(t)$ $\dot{\mathbf{x}}_2(t) = 0.098\mathbf{x}_1(t) - 0.062\mathbf{x}_2(t)^2 - 0.220\mathbf{x}_2^3$

Table 4: Estimated coefficients for Lorenz

Noise level	Estimated System		
	iNeural-SINDy	DeepPyMod	RK4-SINDy [21]
0.00	$\dot{\mathbf{x}}_1(t) = -9.989\mathbf{x}_1(t) + 9.991\mathbf{x}_2(t)$ $\dot{\mathbf{x}}_2(t) = 28.022\mathbf{x}_1(t) - 1.004\mathbf{x}_2(t) + 10.006\mathbf{x}_1(t)\mathbf{x}_3(t)$ $\dot{\mathbf{x}}_3(t) = -2.666\mathbf{x}_3(t) + 10.013\mathbf{x}_1(t)\mathbf{x}_2(t)$	$\dot{\mathbf{x}}_1(t) = -9.992\mathbf{x}_1(t) + 9.993\mathbf{x}_2(t)$ $\dot{\mathbf{x}}_2(t) = 28.026\mathbf{x}_1(t) - 1.000\mathbf{x}_2(t) + 10.000\mathbf{x}_1(t)\mathbf{x}_3(t)$ $\dot{\mathbf{x}}_3(t) = -2.666\mathbf{x}_3(t) + 10.000\mathbf{x}_1(t)\mathbf{x}_2(t)$	$\dot{\mathbf{x}}_1(t) = -9.995\mathbf{x}_1(t) + 10.003\mathbf{x}_2(t)$ $\dot{\mathbf{x}}_2(t) = 28.032\mathbf{x}_1(t) - 1.000\mathbf{x}_2(t) + 10.014\mathbf{x}_1(t)\mathbf{x}_3(t)$ $\dot{\mathbf{x}}_3(t) = -2.665\mathbf{x}_3(t) + 10.020\mathbf{x}_1(t)\mathbf{x}_2(t)$
0.10	$\dot{\mathbf{x}}_1(t) = -9.982\mathbf{x}_1(t) + 9.985\mathbf{x}_2(t)$ $\dot{\mathbf{x}}_2(t) = 28.001\mathbf{x}_1(t) - 1.001\mathbf{x}_2(t) - 10.005\mathbf{x}_1(t)\mathbf{x}_3(t)$ $\dot{\mathbf{x}}_3(t) = -2.666\mathbf{x}_3(t) + 10.008\mathbf{x}_1(t)\mathbf{x}_2(t)$	$\dot{\mathbf{x}}_1(t) = -9.997\mathbf{x}_1(t) + 9.994\mathbf{x}_2(t)$ $\dot{\mathbf{x}}_2(t) = 28.028\mathbf{x}_1(t) - 1.005\mathbf{x}_2(t) - 10.008\mathbf{x}_1(t)\mathbf{x}_3(t)$ $\dot{\mathbf{x}}_3(t) = -2.666\mathbf{x}_3(t) + 9.999\mathbf{x}_1(t)\mathbf{x}_2(t)$	$\dot{\mathbf{x}}_1(t) = -10.004\mathbf{x}_1(t) + 10.008\mathbf{x}_2(t)$ $\dot{\mathbf{x}}_2(t) = 28.060\mathbf{x}_1(t) - 0.997\mathbf{x}_2(t) - 10.025\mathbf{x}_1(t)\mathbf{x}_3(t)$ $\dot{\mathbf{x}}_3(t) = -2.664\mathbf{x}_3(t) + 10.030\mathbf{x}_1(t)\mathbf{x}_2(t)$
0.20	$\dot{\mathbf{x}}_1(t) = -10.016\mathbf{x}_1(t) + 10.008\mathbf{x}_2(t)$ $\dot{\mathbf{x}}_2(t) = 27.980\mathbf{x}_1(t) - 0.991\mathbf{x}_2(t) - 9.998\mathbf{x}_1(t)\mathbf{x}_3(t)$ $\dot{\mathbf{x}}_3(t) = -2.670\mathbf{x}_3(t) + 10.010\mathbf{x}_1(t)\mathbf{x}_2(t)$	$\dot{\mathbf{x}}_1(t) = -10.034\mathbf{x}_1(t) + 10.032\mathbf{x}_2(t)$ $\dot{\mathbf{x}}_2(t) = 27.968\mathbf{x}_1(t) - 0.998\mathbf{x}_2(t) - 9.986\mathbf{x}_1(t)\mathbf{x}_3(t)$ $\dot{\mathbf{x}}_3(t) = -2.669\mathbf{x}_3(t) + 9.993\mathbf{x}_1(t)\mathbf{x}_2(t)$	$\dot{\mathbf{x}}_1(t) = -10.076\mathbf{x}_1(t) + 10.068\mathbf{x}_2(t)$ $\dot{\mathbf{x}}_2(t) = 27.846\mathbf{x}_1(t) - 0.928\mathbf{x}_2(t) - 9.967\mathbf{x}_1(t)\mathbf{x}_3(t)$ $\dot{\mathbf{x}}_3(t) = -2.674\mathbf{x}_3(t) + 9.995\mathbf{x}_1(t)\mathbf{x}_2(t)$
0.40	$\dot{\mathbf{x}}_1(t) = -10.122\mathbf{x}_1(t) + 10.080\mathbf{x}_2(t)$ $\dot{\mathbf{x}}_2(t) = 27.714\mathbf{x}_1(t) - 0.920\mathbf{x}_2(t) - 9.938\mathbf{x}_1(t)\mathbf{x}_3(t)$ $\dot{\mathbf{x}}_3(t) = -2.673\mathbf{x}_3(t) + 9.933\mathbf{x}_1(t)\mathbf{x}_2(t)$	$\dot{\mathbf{x}}_1(t) = -9.887\mathbf{x}_1(t) + 9.902\mathbf{x}_2(t)$ $\dot{\mathbf{x}}_2(t) = 26.627\mathbf{x}_1(t) - 9.596\mathbf{x}_1(t)\mathbf{x}_3(t) - 0.3323\mathbf{x}_2(t)\mathbf{x}_3(t)$ $\dot{\mathbf{x}}_3(t) = -2.657\mathbf{x}_3(t) + 0.035\mathbf{x}_1(t)\mathbf{x}_3(t) + 9.964\mathbf{x}_1(t)\mathbf{x}_2(t)$	$\dot{\mathbf{x}}_1(t) = -10.976\mathbf{x}_1(t) + 10.272\mathbf{x}_2(t) + 0.225\mathbf{x}_1(t)\mathbf{x}_3(t)$ $\dot{\mathbf{x}}_2(t) = 26.144\mathbf{x}_1(t) - 9.490\mathbf{x}_1(t)\mathbf{x}_3(t) - 0.270\mathbf{x}_2(t)\mathbf{x}_3(t)$ $\dot{\mathbf{x}}_3(t) = -2.683\mathbf{x}_3(t) + 10.030\mathbf{x}_1(t)\mathbf{x}_2(t)$

Table 5: Comparison of iNeural-SINDy with WEAK-SINDy

Noise level	Estimated System	
	iNeural-SINDy	Weak-SINDy
0.00	$\dot{\mathbf{x}}_1(t) = -9.990\mathbf{x}_1(t) + 9.990\mathbf{x}_2(t)$ $\dot{\mathbf{x}}_2(t) = 28.021\mathbf{x}_1(t) - 1.003\mathbf{x}_2(t) - 1.006\mathbf{x}_1(t)\mathbf{x}_3(t)$ $\dot{\mathbf{x}}_3(t) = -2.666\mathbf{x}_3(t) + 1.001\mathbf{x}_1(t)\mathbf{x}_2(t)$	$\dot{\mathbf{x}}_1(t) = 10.000\mathbf{x}_1(t) - 10.000\mathbf{x}_2(t)$ $\dot{\mathbf{x}}_2(t) = 28.026\mathbf{x}_1(t) - 1.000\mathbf{x}_2(t) - 1.000\mathbf{x}_1(t)\mathbf{x}_3(t)$ $\dot{\mathbf{x}}_3(t) = -2.670\mathbf{x}_3(t) + 1.000\mathbf{x}_1(t)\mathbf{x}_2(t)$
0.02	$\dot{\mathbf{x}}_1(t) = -9.989\mathbf{x}_1(t) + 9.991\mathbf{x}_2(t)$ $\dot{\mathbf{x}}_2(t) = 28.013\mathbf{x}_1(t) - 1.001\mathbf{x}_2(t) - 1.000\mathbf{x}_1(t)\mathbf{x}_3(t)$ $\dot{\mathbf{x}}_3(t) = -2.666\mathbf{x}_3(t) + 1.001\mathbf{x}_1(t)\mathbf{x}_2(t)$	$\dot{\mathbf{x}}_1(t) = -9.908\mathbf{x}_1(t) + 9.926\mathbf{x}_2(t)$ $\dot{\mathbf{x}}_2(t) = 27.857\mathbf{x}_1(t) - 0.991\mathbf{x}_2(t) - 0.998\mathbf{x}_1(t)\mathbf{x}_3(t)$ $\dot{\mathbf{x}}_3(t) = -2.658\mathbf{x}_3(t) + 1.002\mathbf{x}_1(t)\mathbf{x}_2(t)$
0.08	$\dot{\mathbf{x}}_1(t) = -9.989\mathbf{x}_1(t) + 9.991\mathbf{x}_2(t)$ $\dot{\mathbf{x}}_2(t) = 27.991\mathbf{x}_1(t) - 0.996\mathbf{x}_2(t) - 0.999\mathbf{x}_1(t)\mathbf{x}_3(t)$ $\dot{\mathbf{x}}_3(t) = -2.666\mathbf{x}_3(t) + 1.001\mathbf{x}_1(t)\mathbf{x}_2(t)$	$\dot{\mathbf{x}}_1(t) = -9.798\mathbf{x}_1(t) + 9.778\mathbf{x}_2(t)$ $\dot{\mathbf{x}}_2(t) = 26.699\mathbf{x}_1(t) - 0.790\mathbf{x}_2(t) - 0.969\mathbf{x}_1(t)\mathbf{x}_3(t)$ $\dot{\mathbf{x}}_3(t) = -2.640\mathbf{x}_3(t) + 1.010\mathbf{x}_1(t)\mathbf{x}_2(t)$
0.10	$\dot{\mathbf{x}}_1(t) = -9.988\mathbf{x}_1(t) + 9.991\mathbf{x}_2(t)$ $\dot{\mathbf{x}}_2(t) = 27.982\mathbf{x}_1(t) - 0.993\mathbf{x}_2(t) - 0.994\mathbf{x}_1(t)\mathbf{x}_3(t)$ $\dot{\mathbf{x}}_3(t) = -2.666\mathbf{x}_3(t) + 1.019\mathbf{x}_1(t)\mathbf{x}_2(t)$	$\dot{\mathbf{x}}_1(t) = -9.751\mathbf{x}_1(t) + 9.734\mathbf{x}_2(t)$ $\dot{\mathbf{x}}_2(t) = 26.337\mathbf{x}_1(t) - 0.716\mathbf{x}_1(t)\mathbf{x}_3(t) - 0.961\mathbf{x}_2(t)\mathbf{x}_3(t)$ $\dot{\mathbf{x}}_3(t) = -2.630\mathbf{x}_3(t) + 1.010\mathbf{x}_1(t)\mathbf{x}_2(t)$