Parameter Inference by Neural Network-Guided Elimination of Latent Variables in Ordinary Differential Equations

Anonymous Author(s)

Abstract

We study parameter inference for ordinary differential equation (ODE) models when the available data consists of observable functions over the system states. ODE models are widely used in scientific and engineering applications to predict future behaviors from past data and use parameter values to analyze physical properties. However, parameter inference for ODE models is challenging, especially when full state observations are unavailable. We use a two-step approach based on deriving input-output (I/O) equations given the description of the ODE and the observables. The I/O equations directly involve the observables, their higher-order derivatives and the unknown parameters, but no latent states. For systems defined by polynomial or rational function ODEs, there are well-known computational approaches to computing I/O equations using ideas from differential algebra. We show that, under a constant-rank condition, an I/O equation exists locally for smooth systems and can be written in functional form, and we present an approximate approach that learns this function from samples with a neural network through an appropriate loss function. This approach can work for non-polynomial systems but provides guarantees that are approximate in nature, depending on how well the trained network represents the I/O equation. Next, we use the learned I/O equation in a least-squares optimization problem by estimating the derivatives of sampled signals with a suite of numerical differentiation methods. We evaluate our approach on 20 nonlinear ODE benchmarks with partial observation data, comparing against related approaches including solver-guided optimization and the Julia toolbox ParameterEstimation.jl. Our approach provides estimates on 19 of the 20 benchmarks, while the other approaches time out or fail on many of them, and its parameter estimates are more accurate than those of solver-guided optimization wherever both finish.

ACM Reference Format:
Anonymous Author(s). 2026. Parameter Inference by Neural Network-Guided Elimination of Latent Variables in Ordinary Differential Equations. In . ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

We study parameter inference for Ordinary Differential Equations (ODE) under partial observations. ODEs are widely used in diverse application areas ranging from weather modeling to pandemic prediction. Numerous ODE models are governed by a set of (time invariant) parameters that capture key aspects of the model such as reaction rates in chemical models, infection rates in epidemiological models or important physical constants. Although a differential equation model can involve many state variables, these are often not directly observed in a measurement framework. This leads to the challenge of having to reconstruct the missing latent states and the parameters simultaneously from the data on observables in a constrained optimization setting.

In this paper, we propose a strategy based on so-called *input-output (I/O) form* which is an equation involving just the parameters, the observables and their higher-order derivatives but not the latent/unobservable states of the system. Formally, for a ODE model $\dot{\mathbf{x}} = f(\mathbf{x}, \theta)$ over states $\mathbf{x} \in \mathbb{R}^n$ and parameters $\theta \in \mathbb{R}^k$ with an observable $\mathbf{y} = g(\mathbf{x}, \theta) \in \mathbb{R}^m$ (typically $m < n$), an I/O form is a function $\mathcal{H}(\mathbf{y}, \mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(n)}) = 0$, wherein $\mathbf{y}^{(i)}$ is the i^{th} time derivative of the observable $\mathbf{y}$. By adapting classic ideas from nonlinear observability theory, we show that I/O form exists under some conditions on the Jacobian matrix of the observables and their higher order derivatives (Theorem 3.2). In fact, the I/O form can be written in a functional form $\mathbf{y}^{(r)} = \mathcal{G}(\mathbf{y}, \mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(r-1)}, \theta)$ for some index $r \leq n$ under the assumption of a constant rank Jacobian [24, 44]. For polynomial or rational systems, well-known ideas from computational algebraic geometry can be employed to compute an I/O form [11, 14, 20, 22]. However, these involve Gröbner basis computations or differential algebraic techniques that are computationally expensive for larger benchmarks [11]. Therefore, we study an approximate approach using machine learning to compute I/O forms. Using the idea that the I/O form has a functional representation allows us to find a neural network $\mathcal{G}$ that satisfies the functional equation using a simple loss function. Once an I/O form is discovered, we can use that to find values of parameters that satisfy the I/O form given the observables by solving a least squares loss function of the form $\min_\theta \sum_{j=1}^{n} \|\mathcal{G}(\mathbf{y}(t_j), \mathbf{y}^{(1)}(t_j), \ldots, \mathbf{y}^{(r-1)}(t_j), \theta) - \mathbf{y}^{(r)}(t_j)\|_2^2$. Doing so requires us to estimate the higher order derivatives of the observed data. We achieve this through existing numerical differentiation approaches to estimate the derivatives of the given observable trajectory [29] and a combination of numerical differentiation approaches that allows us to estimate the higher order derivatives even in the presence of noise [29, 48].

Example 1.1 (Two-mass spring system)**.** Consider two unit masses connected to each other and to two fixed walls by springs with constants k_1, k_2 and k_1. With positions x_1, x_2 and velocities v_1, v_2, the model has four states and two unknown parameters:

$$\dot{x}_1 = v_1, \quad \dot{v}_1 = -(k_1 + k_2)\, x_1 + k_2\, x_2,$$
$$\dot{x}_2 = v_2, \quad \dot{v}_2 = k_2\, x_1 - (k_1 + k_2)\, x_2.$$

Only the position of the second mass is observed, $y = x_2$: we are given 500 noisy samples over $t \in [0, 15]$, generated with $k_1 = 0.5$ and $k_2 = 0.3$. Eliminating x_1, v_1 and v_2 (Section 3.1) yields the I/O

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
Conference'17, Washington, DC, USA
© 2026 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

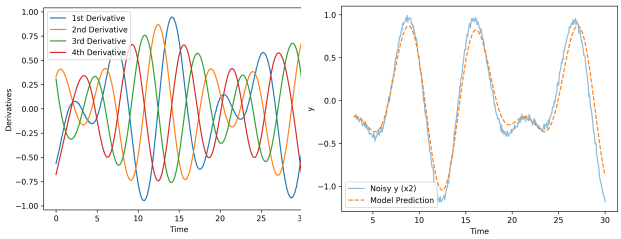


Figure 1: Two-mass spring system (Example 1.1). (Left) Derivatives $\dot{y}, y^{(2)}, y^{(3)}, y^{(4)}$ estimated from noisy samples of $y = x_2$. (Right) The noisy samples and the trajectory simulated with the estimated parameters $k_1 = 0.48$, $k_2 = 0.29$.

form

$$y^{(4)} + 2(k_1 + k_2)\, y^{(2)} + (k_1^2 + 2k_1k_2)\, y = 0,$$

which involves only the observable, its derivatives and the parameters. We estimate $y^{(2)}$ and $y^{(4)}$ from the noisy samples (Section 5) and minimize the squared residual of this equation over (k_1, k_2), obtaining $k_1 = 0.48$ and $k_2 = 0.29$. Figure 1 shows the estimated derivatives and the trajectory simulated from these estimates.

The key contributions of this paper include: (a) a proof that under a constant rank condition, the I/O form can be expressed "functionally" (Theorem 3.2); (b) a machine learning approach using the existence of a functional form that allows us to setup a loss function that produces an approximate I/O form upon minimization; and (c) the use of these ideas in parameter estimation for nonlinear/nonpolynomial ODE models from partial observation data.

We evaluate our approach on a series of 20 benchmark examples with $2-8$ state variables and $1-9$ parameters. We observe that the neural network converges appropriately in all but a few of the examples. We compare against forward solver-based optimization, symbolic elimination and ParameterEstimation.jl on these benchmarks [4]. Our approach returns estimates on 19 out of 20 benchmarks, whereas solver-based optimization times out on 9. Where both finish, our parameter estimates are more accurate, though solver-based fits are often closer. On three benchmarks, our fits remain stable as noise increases, thanks to the suite of numerical differentiation methods employed.

1.1 Related Work

The problem of parameter inference for ODEs has been widely studied [3, 30] with a wide variety of approaches. Specialized parameter estimation techniques have been evolved for classes of models such as pharmaco-kinetic/pharamaco-dynamic models used to study drug diffusion in humans and animals [23, 43]. In general, we are given a ODE $\frac{d\mathbf{x}}{dt} = f(\mathbf{x}, \theta)$ with state variables $\mathbf{x} \in \mathbb{R}^n$ and unknown parameters θ. The data provided can be full state data $\mathbf{x}(t)$ sampled at various time points t_i, or "partial" observations $\mathbf{y}(t) = g(\mathbf{x}(t), \theta) \in \mathbb{R}^m$ with $m < n$. In this paper, we study parameter inference under partial observations (typically scalar observations with $m = 1$).

The approaches may be classified into broad categories: (a) ODE solver based approaches that integrate and evaluate various parameter values against the provided data, using an optimization solver to update parameters [2, 34, 52]; or (b) a direct approach that matches the values/derivatives of state variables against the right hand sides to obtain parameters [5, 6, 13, 38, 49].

ODE solver-based approaches are generally advantageous when the available data is sparse and noisy, since estimating derivatives from data can be hard in such cases. They can also handle partial observations. However, two key difficulties arise: (a) optimization algorithms require a large number of calls to ODE solvers; (b) although derivative-based optimization is possible, calculating derivatives is quite challenging when ODEs are involved in the optimization problem as equality constraints [17, 52], requiring the solution to an adjoint ODE that can often be numerically unstable even if the original ODE is stable.

The so-called "direct approach" matches the RHS of the ODE $f(\mathbf{x}, \theta)$ against the estimated derivatives $\frac{d\mathbf{x}}{dt}$ at various time points in order to compute θ as a least-squares problem. Numerical differentiation methods can be used to compute the derivatives of $\mathbf{x}(t)$ [29]. These techniques are further discussed in Section 5. Numerical differentiation can be achieved by fitting polynomials or polynomial splines to the data and obtaining the derivatives by differentiating them [5, 49], Gaussian-process regression to estimate derivatives [9, 13] and smoothing the data to obtain derivative estimates [16, 38]. The SINDY algorithm is a popular instance of the direct approach [8] that uses the vector field data at various points in the state-space to discover the RHS of an unknown ODE as a linear combination over fixed basis functions. This approach extends to parameter estimation by setting up a least-squares problem of the form $\text{argmin}_\theta \sum_{j=1}^{N} \|f(\mathbf{x}_i, \theta) - \mathbf{v}_i\|_2^2$, wherein $\mathbf{v}_i$ is the vector field at state $\mathbf{x}_i$.

Weak SINDy/WENDy: Bortz and colleagues developed weak-form methods for model discovery (WSINDy [32**?**]) and parameter estimation (WENDy [6]). Their approach replaces the numerical differentiation by the so-called weak-form, wherein integration with test functions is used instead of numerical derivative computation. Weak-form methods such as WSINDy and WENDy avoid pointwise differentiation by integrating each state equation $\dot{x}_i = f_i(\mathbf{x}, \theta)$ to yield an equation $x_i(t) - x_i(0) = \int_0^t f_i(\mathbf{x}(s), \theta) ds$, estimated over the given data through quadrature. Rather than perform this integration directly on state variables, their approach uses a series of test functions $\tau(t)$ and integrates the product τx_i using integration by parts formula followed by an evaluation using numerical quadrature. However, because $f_i(\mathbf{x}, \theta)$ must be evaluated along the data, they require measurements of every state that appears in the RHS of the ODE, and are not directly applicable when we have partial observations. The original work assumes that f_i is linear in the parameters θ [6], whereas a recent extension by Rummel et al addresses this limitation [41].

I/O Form and Parameter Estimation. Input-Output Equations (called I/O forms in this paper) are a classic approach to expressing and analyzing control systems [24]. The use of I/O forms for parameter estimation of nonlinear ODE models has been studied for specific class of ODE models where the structure of the model and

the observation allows us to derive the I/O form by hand in terms of the unknown parameters [25, 35, 51]. The difficulty of this approach is twofold: (a) deriving the I/O forms for given nonlinear ODE and observables; and (b) computing the higher order derivatives of the observables.

Differential elimination: The use of differential algebra, first developed by Ritt [39], provides a framework similar to the classic elimination theory in polynomial systems [11] that allows us to compute I/O forms for ODEs with polynomial and rational function RHS [20]. Key algorithms that enable the computation of the I/O form include the Rosenfeld–Gröbner algorithm [7]. However, these approaches have been more widely used for identifiability analysis: can the parameters of a system be identified given data on observables [1]? One of the standard approaches for identifiability analysis is to derive an I/O form and reason about the number of solutions to the parameters over some given data. Ljung and Glad [31], Hong et al [22] and Ovchinnikov et al [36] use this machinery to perform identifiability analysis using differential algebraic techniques.

However, the use of differential algebra for elimination is often expensive since the underlying algorithms have worst-case exponential complexity. Dong et al. [14] address this with a projection-based representation of the I/O relations that handles models previously out of reach. This is implemented in the Julia package `StructuralIdentifiability.jl` [45]. The symbolic approach in Section 3.1 is an instance of this approach that uses computational algebraic geometric tools such as Grobner bases directly. However, our contribution differs in two respects: Theorem 3.2 establishes the local existence of I/O equations for smooth systems without algebraic structure, and Section 4 learns an I/O equation when no closed form can be computed. The tool `ParameterEstimation.jl` solves a problem identical to that posed in this paper by combining the differential algebraic approaches mentioned above for analyzing the number of parameters followed by rational interpolation to compute derivatives of the observable and using homotopy continuation methods to solve the resulting nonlinear equations [4]. This approach is restricted to polynomial and rational ODEs, whereas our approach is more generally applicable. A detailed comparison with this tool is provided in Section 6 and shows that the techniques proposed here can apply to more (larger) benchmarks.

Heitzman-Breen et al. extend the ideas from WENDy to partial observations for polynomial models by computing I/O equations through differential elimination methods, with the aim of assessing identifiability [21]. They assume that the I/O equations are in a form that permit weak-form integration (eg., linear weak-form). Our work differs in many ways: we learn the I/O form through machine learning approaches, extending our approach to systems beyond polynomial/rational function ODEs. We also demonstrate the use of weak-form approaches for higher-order derivative estimation (see Appendix B) in this paper. Finally, our approach does not require the linearity of the I/O form.

2 Problem Statement

An ordinary differential equation (ODE) model $\langle \mathbf{x}, \theta, f \rangle$ is of the form $\frac{d\mathbf{x}}{dt} = f(\mathbf{x}, \theta, t)$, wherein $\mathbf{x} \in \mathbb{R}^n$ is an $n-$dimensional vector of *state variables*, $\theta \in \Theta \subseteq \mathbb{R}^k$ is an $k-$dimensional vector of constant *parameters* and t represents the time. We will assume that the right hand side f is a smooth C^∞ function. Given an ODE , an *observable* $\mathbf{y} \in \mathbb{R}^m$ is a function $\mathbf{y} = g(\mathbf{x}, \theta, t)$ representing quantities that can be observed/measured. Given an initial condition $\mathbf{x}(0) = \mathbf{x}_0$, let $\mathbf{x}(t)$ represent the time trajectory of the system and $\mathbf{y}(t) = g(\mathbf{x}(t), \theta, t)$, the *observable* trajectory.

Definition 2.1 (Parameter Inference Problem)**.** The parameter inference problem inputs an ODE $\langle \mathbf{x}, \theta, f \rangle$ with observable $\mathbf{y} = g(\mathbf{x}, \theta, t)$, where the parameters θ are unknown; and data in the form of observable trajectory samples $\mathbf{y}_1, \ldots, \mathbf{y}_N$, wherein $\mathbf{y}_i = \mathbf{y}(i\delta) + \mathbf{e}_i$ for $\delta > 0$ and i.i.d Gaussian errors $\mathbf{e}_i \sim \text{Gaussian}(0, \sigma^2 I)$.

We seek values of unknown parameters $\theta \in \Theta$ that best "explain"the observable trajectory. In other words, θ must minimize the *curve fit loss* $\sum_{i=1}^N \|g(\mathbf{x}(i\delta), t, \theta) - \mathbf{y}_i\|^2$ between an observable trajectory generated by the parameter θ and the given data. This can be posed as an optimization problem involving initial state $\mathbf{x}_0$ and parameter θ:

$$\begin{aligned} \min_{\mathbf{x}_0, \theta} \quad & \sum_{j=0}^N \|g(t_0, \mathbf{x}(\delta j), \theta) - \mathbf{y}_j\|^2 \\ \text{s.t} \quad & \mathbf{x}(t) \text{ is solution of } \tfrac{d\mathbf{x}}{dt} = f(\mathbf{x}, \theta) \\ & \text{with } \mathbf{x}(0) = \mathbf{x}_0 \end{aligned} \tag{1}$$

The assumption of i.i.d. measurement noise is standard for parameter inference problems. Our approach can be modified to accommodate other measurement error models provided the distribution of errors is known.

Assumption 2.2. For simplicity of presentation, we assume: (1) The observables are scalars, i.e., $y \in \mathbb{R}$, although all the approaches in this paper can be readily extended to vector observables. (2) The function f defining the RHS of the ODE and the observable function g are time invariant, i.e., $\frac{d\mathbf{x}}{dt} = f(\mathbf{x}, \theta)$ and $y = g(\mathbf{x}, \theta)$. The extension to time varying maps is conceptually simple, requiring us to treat the time either as an observable or a latent state variable. (3) Finally, we assume that f, g are C^∞ functions.

In the absence of data about $\mathbf{x}(t)$, solving the optimization problem (1) involves formulating the adjoint equation in order to compute the gradient of the loss function or resorting to a global optimization approach [2, 34]. In this paper, we propose a novel approach that is based on elimination of the state-variables from the ODE in order to express it purely in terms of the observable. This yields a purely unconstrained least-squares optimization problem.

3 Input/Output Forms

Let $y^{(m)}$ denote the m^{th} derivative $\frac{d^m y}{dt^m}$ for $m \geq 1$. Let $X \subseteq \mathbb{R}^n$ represent the state space and $\Theta \subseteq \mathbb{R}^k$, the parameter space. The partial derivative $\frac{\partial g}{\partial x_i}$ will be written $\partial_{x_i} g$.

Definition 3.1 (Input-Ouput (I/O) Form)**.** An Input-Output (I/O) form over a domain $X \times \Theta$ is an equation $\mathcal{G}(y, y^{(1)}, \ldots, y^{(n)}, \theta) = 0$, wherein $\mathcal{G} : \mathbb{R}^{n+1} \times \mathbb{R}^k \to \mathbb{R}$, such that for each trajectory $\mathbf{x}(t) \in X$ of the ODE and parameter value $\theta \in \Theta$, the corresponding *observable trajectory* $y(t) = g(\mathbf{x}(t), \theta)$ satisfies the I/O form $\mathcal{G}(y(t), y^{(1)}(t), \ldots, y^{(n)}(t), \theta) = 0$ for all time t.

The elimination problem consists of computing the functional form $\mathcal{G}$, given a description of the ODE and the state/parameter spaces. We will now show that the I/O form exists under some conditions using simple differential geometric arguments. The higher

order derivatives of y are obtained by repeated differentiation, as follows:

$$\left.\begin{aligned} y^{(0)} &= g(\mathbf{x},\theta) \;=\; g_0(\mathbf{x},\theta) \\ y^{(1)} &= \tfrac{dy}{dt} \;= \nabla_x g \cdot f = g_1(\mathbf{x},\theta)\,. \\ y^{(2)} &= \tfrac{d^2y}{dt^2} \;= \nabla_x g_1 \cdot f = g_2(\mathbf{x},\theta)\,. \\ &\vdots \\ y^{(n)} &= \tfrac{d^ny}{dt^n} \;= \nabla_x g_{n-1} \cdot f = g_n(\mathbf{x},\theta)\,. \end{aligned}\right\} \tag{2}$$

Let $M(\mathbf{x},\theta)$ denote the $(n+1)\times n$ Jacobian matrix:

$$M(\mathbf{x},\theta) = \begin{pmatrix} \partial_{x_1} g & \cdots & \partial_{x_n} g \\ \partial_{x_1} g_1 & \cdots & \partial_{x_n} g_1 \\ & \ddots & \\ \partial_{x_1} g_{n-1} & \cdots & \partial_{x_n} g_{n-1} \\ \partial_{x_1} g_n & \cdots & \partial_{x_n} g_n \end{pmatrix}. \tag{3}$$

Theorem 3.2. *Suppose $M(\mathbf{x},\theta)$ has a constant rank $r \geq 1$ for all $(\mathbf{x},\theta) \in D\times\Theta$, where D is an open set. For every $\mathbf{x}^* \in D$ and $\theta \in \Theta$ there exists a "functional" I/O form $\mathcal{G}(y^{(i_0)},\ldots,y^{(i_{r-1})},\theta) = \mathbf{y}^{(i_r)}$ over a set of $r+1$ indices $\{i_0,\ldots,i_r\} \subseteq \{0,\ldots,n\}$ that is valid over an open neighborhood $\hat{D} \subseteq D\times\Theta$ of $(\mathbf{x}^*,\theta)$.*

Proof. Let us fix a value of $\theta \in \Theta$ and a point $\mathbf{x}^* \in D$. We note that the rank if $M(\mathbf{x},\theta)$ is $r \leq n$. The rows of M are of the form $\nabla_x g_i$ for $i \in \{0,\ldots,n\}$ with $g_0 = g$. Since $M(\mathbf{x}^*,\theta)$ has rank r, there exists a set of r distinct row indices $i_0,\ldots,i_{r-1}$ such that for any other row i_r, we have $\nabla g_{i_r} = \sum_{j=0}^{r-1} a_j \nabla g_{i_j}$.

At the same time, there is a $r\times r$ minor K_r of M consisting of rows $i_0,\ldots,i_{r-1}$ and some columns $j_0,\ldots,j_{r-1}$ such that $\det(K_r) \neq 0$ at $(\mathbf{x}^*,\theta)$. Since the determinant is a continuous function over $(\mathbf{x},\theta)$, there is a open neighborhood $\hat{D} \subseteq D\times\Theta$ such that $\det(K_r(\mathbf{x},\theta)) \neq 0$ for all $\mathbf{x} \in \hat{D}$. Thus, K_r^{-1} is a smooth function of $\mathbf{x},\theta$ in $\hat{D}$. We obtain smooth coefficient functions $a_0(\mathbf{x},\theta),\ldots,a_{r-1}(\mathbf{x},\theta)$ over $\hat{D}$ such that $\nabla g_{i_r} = \sum_{j=0}^{r-1} a_j \nabla g_{i_j}$, and $\begin{pmatrix} a_0 \\ \vdots \\ a_{r-1} \end{pmatrix} = K_r^{-1} \begin{pmatrix} \nabla g_{i_r\, j_0} \\ \vdots \\ \nabla g_{i_r\, j_{r-1}} \end{pmatrix}$.

Next, consider the map

$$h(\mathbf{x},\theta) : \hat{D} \to \mathbb{R}^r = (g_{i_0}(\mathbf{x},\theta), g_{i_1}(\mathbf{x},\theta),\ldots,g_{i_{r-1}}(\mathbf{x},\theta))\,.$$

Since the sub-matrix formed by the rows corresponding to indices $i_0,\ldots,i_{r-1}$ of M has rank r, this map is a submersion for a fixed θ, and is smooth over θ [1]. Consider for any fixed θ, a non-empty level set of the form $\{\mathbf{x} \mid h(\mathbf{x},\theta) = \mathbf{c}\}$. By the properties of a submersion, the level set is a smooth manifold. Let τ be a tangent vector to this manifold. We have that $\nabla_x g_{i_r} \cdot \tau = \sum_{j=0}^{r-1} a_j \nabla_x g_{i_j} \cdot \tau = 0$, since the tangent vector to a level set of h will need to satisfy $\nabla_x g_{i_j} \cdot \tau = 0$ for $j = 0,\ldots,r-1$. Hence, g_{i_r} is constant over all level sets of $h = (g_{i_0},\ldots,g_{i_{r-1}},\theta)$. Thus, $g_{i_r} = \varphi(g_{i_0},\ldots,g_{i_{r-1}},\theta)$ for some function φ over the domain $\hat{D}$. Thus, we obtain the required I/O form $\mathbf{y}^{(i_r)} = \varphi(y^{(i_0)},\ldots,y^{(i_{r-1})},\theta)$.

This concludes the proof of the theorem. ∎

Example 3.3. Consider the system

$$\frac{dx_1}{dt} = x_2,\ \frac{dx_2}{dt} = x_1 - \theta x_2,\ \frac{dx_3}{dt} = x_1\,, \text{with } g(\mathbf{x},\theta) = \theta x_2\,.$$

[1] A map $h : \mathbb{R}^n \to \mathbb{R}^r$ is a submersion if it is a differentiable map and its Jacobian has full row rank.

Note that $g(\mathbf{x},\theta) = \theta x_2$. We have $g_1 = \theta x_1 - \theta^2 x_2$, $g_2 = -\theta^2 x_1 + \theta(1+\theta^2)x_2$ and $g_3 = \theta(1+\theta^2)x_1 - \theta^2(2+\theta^2)x_2$.

$$M(\mathbf{x},\theta) = \begin{pmatrix} 0 & \theta & 0 \\ \theta & -\theta^2 & 0 \\ -\theta^2 & \theta(1+\theta^2) & 0 \\ \theta(1+\theta^2) & -\theta^2(2+\theta^2) & 0 \end{pmatrix}$$

For $\theta \neq 0$, $\mathrm{rank}(M) = 2$ for all $\mathbf{x}$. Also notice that the minor of M corresponding to the first two rows and columns has a non-zero determinant for all $\mathbf{x}$ when $\theta \neq 0$. We obtain an I/O form $\mathcal{G}_2 = y^{(2)} - y + \theta y^{(1)} = 0$ involving just the first and second derivatives, that is valid over all $\theta, x \in \mathbb{R}^2$ (including $\theta = 0$). However, when $\theta = 0$, the observable is $y = 0$ and this is the trivial I/O form, yielding no information on θ. The I/O form involving $y^{(2)} = y - \theta y^{(1)}$ can be seen as a "consequence" of $y = 0$ (see Lemma 3.4 below).

In general, the I/O form need not be unique. Multiple forms can be related to a single "base form".

Lemma 3.4. *Suppose $\mathcal{G}(z_0,\ldots,z_r,\theta)$ is a smooth function such that $\mathcal{G}(g_0(\mathbf{x},\theta),\ldots,g_r(\mathbf{x},\theta),\theta) = 0$ for $r < n$ and for all $(\mathbf{x},\theta) \in D$ (open set D) then we can derive an "implied" form $\mathcal{H}(g_0,\ldots,g_{r+1},\theta) = 0$ over D.*

Proof. Since $\mathcal{G}(g_0,\ldots,g_r,\theta) \equiv 0$ for any $(\mathbf{x},\theta)$, $\nabla_x \mathcal{G} = 0$. Let $\gamma = \nabla_z \mathcal{G}$ (row vector) evaluated at $z = (g_0,\ldots,g_r)$. $\nabla_x \mathcal{G} = \gamma M_r f$, wherein M_r denotes the first $r+1$ rows of M. However, by the construction of $g_1,\ldots,g_n$, $M_r \times f$ is the column vector $g_1,\ldots,g_{r+1}$. Therefore, $\gamma M_r f = \gamma \cdot (g_1,\ldots,g_{r+1}) = 0$, yielding a new implicit form $\mathcal{H}(y^{(0)},\ldots,y^{(r+1)}) = 0$.

∎

Example 3.5. Continuing with Ex. 3.3, we can infer an additional forms $\mathcal{H}_1 = -\theta y^{(2)} + y^{(1)} - y^{(3)} = 0$ and $\mathcal{H}_2 = -\theta y^{(3)} + y - (\theta^2+1)y^{(2)} = 0$. $\mathcal{H}_1$ can be obtained by differentiating $\mathcal{G}_2$, whereas $\mathcal{H}_2 = \theta\mathcal{H}_1 - \mathcal{G}_2$.

Remark 3.6. Note that the I/O form involves derivatives $i_0,\ldots,i_r \in \{0,\ldots,n\}$. The specific indices involved depend on the neighborhood $\hat{D}$, varying based on which minor of the matrix M has a non-zero determinant for given $\mathbf{x}^*,\theta$. However, since we choose i_r in the proof of Theorem 3.2 and $r \leq n$, we can always choose $i_r = 1 + \max\{i_0,\ldots,i_{r-1}\}$, provided $n \notin \{i_0,\ldots,i_{r-1}\}$. The most common case occurs when indices $\max(i_0,\ldots,i_{r-1}) \leq n-1$, which allows us in the proof to choose $i_r = n$ without loss of generality.

The ideas used here are very similar to those used to analyze nonlinear systems for observability wherein the output data $y(t)$ can be used to build observers that can reconstruct the state $\mathbf{x}(t)$ as a function of $y(t), y^{(1)}(t),\ldots,y^{(n)}(t)$ [24]. The condition for observability would require that the matrix $M(\mathbf{x})$ has rank n. In our setting, the goal is to reconstruct the parameter θ rather than the full state information $\mathbf{x}(t)$.

Section 4 will use a neural-network to represent the function $\mathcal{G}$ in the proof of Theorem 3.2 and construct a loss function to learn it through a sampling process.

3.1 I/O forms for Polynomial Systems

We present an approach to elimination when the RHS of the ODE $f(\mathbf{x},\theta)$ and the observable function $g(\mathbf{x},\theta)$ are polynomials over $\mathbf{x},\theta$.

As a result, the functions $g_1, \ldots, g_n$ in Eq. (2) are all polynomials over $\mathbf{x}, \theta$. Let $\mathbf{y} = (y, y^{(1)}, \ldots, y^{(n-1)}, y^{(n)})$ be a vector of $n+1$ variables that stand for the observable and its n successive derivatives. We will recall some key definitions from algebra. Given variables $\mathbf{z}$, let $\mathbb{R}[\mathbf{z}]$ denote all multivariate polynomials involving $\mathbf{z}$ with real-valued coefficients. We recall the concepts of a polynomial ideal and Hilbert's Nullstellensatz from standard references [11].

In order to construct an I/O form, we consider the ideal generated by the set $I = \{y - g(\mathbf{x}, \theta), y^{(1)} - g_1(\mathbf{x}, \theta), \ldots, y^{(n)} - g_n(\mathbf{x}, \theta)\}$ obtained from Eq. (2). The I/O form for the ODE $\frac{d\mathbf{x}}{dt} = f(\mathbf{x}, \theta)$ with the observable $y = g(\mathbf{x}, \theta)$ can be seen as a polynomial $\mathcal{G}(y^{(n)}, \ldots, y, \theta)$ that vanishes whenever every polynomial in the set I vanishes (or, in other words, Eq. (2) holds). Therefore, an I/O form $\mathcal{G} \in \mathbb{R}[\mathbf{y}, \theta]$ is a member of $\langle I \rangle \cap \mathbb{R}[y, \ldots, y^{(n)}, \theta]$.

Approaches to find such polynomials $\mathcal{G}$ include Macaulay resultants, Gröbner bases and the Wu-Ritt method [26]. The Gröbner basis approach can be implemented using packages such as SymPy, Singular or Mathematica, as follows:

(1) Compute a Gröbner basis G of the ideal I using an *elimination ordering*.
(2) Any polynomial in the basis G that involves only the variables in $y, \ldots, y^{(n)}, \theta$ is an I/O form.
(3) We choose the I/O form that involves the lowest order derivatives in order to avoid working with implied forms (see Lemma 3.4 and Ex. 3.5). The elimination ordering is chosen so that monomials involving lower order derivatives are ordered lower than those involving higer order derivatives.

Example 3.7. Consider a two dimensional Lotka Volterra system with 4 parameters $(w_1, \ldots, w_4)$: $\frac{dx}{dt} = w_1 x + w_2 xy$, $\frac{dy}{dt} = w_3 y + w_4 xy$, with $g(x, y) = y$. We have $y^{(1)} = w_3 y + w_4 xy$ and $y^{(2)} = (w_3 + w_4 x)(w_3 y + w_4 xy) + w_4 y(w_1 x + w_2 xy)$. The elimination ideal is generated by a single polynomial $\mathcal{G} = w_1 w_3 y^2 - w_1 y y^{(1)} + w_2 w_3 y^3 - w_2 y^2 y^{(1)} + y y^{(2)} - (y^{(1)})^2$. This polynomial $\mathcal{G}$ represents our I/O form.

Computing the I/O form in this manner requires an expensive Gröbner basis computation whose worst-case time complexity is exponential in the number of state variables and parameters with a doubly exponential dependence on the total degree in the worst case [15]. Furthermore, it is limited to polynomial systems.

4 Learning I/O Forms

We now present a general approach using machine learning, inspired by Theorem 3.2. Specifically, the I/O form expresses $y^{(n)}$ as a function of $\theta, y, \ldots, y^{(n-1)}$. Figure 2 shows the overall setup in terms of the successive derivatives $g, g_1, \ldots, g_n$ from Eq. (2).

Let $\mathcal{G}_{NN}(\mathbf{x}, \theta; \mathrm{W})$ be a neural network with parameters W. We sample states and parameters $(\mathbf{x}_i, \theta_i)$ for $i = 1, \ldots, K$ over a fixed set of states X and parameter set Θ. The derivatives $g_1, \ldots, g_n$ are evaluated using symbolic differentiation. For the sampled states and parameters, we compute $y_i^{(j)} = g_j(\mathbf{x}_i, \theta_i)$ for $i = 1, \ldots, K$ and $j = 0, \ldots, n$, with $g_0 = g$. We fix a neural network model $\mathcal{G}_{NN}$ with parameters W. The network output $\hat{y}_i^{(n)} = \mathcal{G}_{NN}(y_i, \ldots, y_i^{(n-1)}, \theta_i; \mathrm{W})$ should predict the value $y_i^{(n)}$. Therefore, the loss function to be minimized is:

$$\ell(\mathrm{W}) = \sum_{i=1}^{K} \left\| \mathcal{G}_{NN}(y_i, \ldots, y_i^{(n-1)}, \theta_i; \mathrm{W}) - y_i^{(n)} \right\|^2 .$$

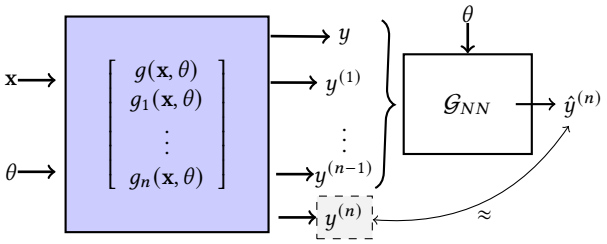


Figure 2: Overall schema for learning an I/O form using the minimization of a loss function. The functions $g_1, \ldots, g_n$ are as in Eq. (2) with the I/O form $\mathcal{G}_{NN}$ represented as a neural network. The loss function is designed to match the predicted derivative $\hat{y}^{(n)}$ with the actual one $y^{(n)}$.

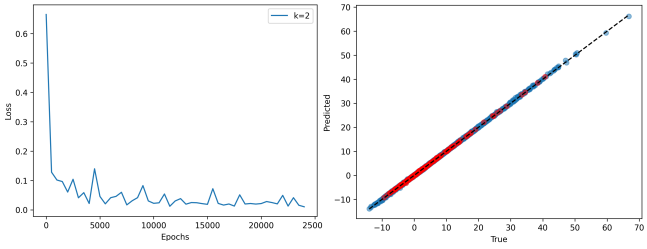


Figure 3: (Left) Training loss (averaged over each epoch) for a neural network model with 4 layers, 64 neurons/layer with ReLU activation functions to predicts $y^{(2)}$ as a function of $y^{(1)}, y^{(0)}$ and parameters w for the 2D Lotka-Volterra system in Example 3.7. (Right) Plotting the predicted versus ground truth values of $y^{(2)}$ for training samples in blue and held out test samples in red.

Example 4.1. Continuing with the system in Ex. 3.7, we trained a neural network model to predict the second derivative $y^{(2)}$ as a function of the derivative $y^{(1)}$, observable y and parameters $w_1, \ldots, w_4$. We used 10000 training samples. The final model predicts the derivative $y^{(2)}$ quite accurately over a held out test data set of 1000 samples. The final MSE training loss was 0.009 whereas the test loss was 0.01 over a held out set of 1000 test samples.

Remark 4.2. We can modify our approach to express $y^{(r+1)}$ as a function of $y, \ldots, y^{(r)}$ for $r < n-1$, starting from $r = 1$ and increasing it to $n-1$. This would requires us to train multiple networks for various values of r and a test to determine when a given network corresponds to an underlying functional relationship between the derivatives. The test itself can be quite simple: we simply calculate the loss $\ell(\mathrm{W})$ over N_t held out test set. If the average loss $\ell(\mathrm{W})/N_t$ is below some threshold (eg., 10^{-2}), we accept the resulting network as a valid I/O form. In this paper, we simply compute a single network for $r = n$ as our I/O form.

5 Parameter Inference from Data

We will now use the I/O formto infer unknown parameters θ, given a sequence of observable measurements $y_1, \ldots, y_N$ at times $t_1, \ldots, t_N$. Assume that an I/O form$y^{(n)} = \mathcal{G}(y, \ldots, y^{(n-1)}, \theta)$ has been computed either using the symbolic approach in Section 3.1 (for polynomial systems) or the machine learning-based approach in Section 4.

We will use a two step approach:

(1) Use *numerical differentiation* techniques to estimate the derivatives $y_k^{(j)}$, the j^{th} derivative at time t_k.
(2) Solve a nonlinear least squares problem:

$$\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} (y_i^{(n)} - \mathcal{G}(y_i, y_i^{(1)}, \ldots, y_i^{(n-1)}, \theta))^2 . \tag{4}$$

The loss function is designed to enforce the I/O formusing the estimated derivatives.

5.1 Numerical Differentiation

We will briefly discuss numerical differentiation techniques, including the heuristics we used to select from a large class of approaches that exist in practice. Our approach follows the recent survey on this topic [29]. The simplest approach to computing derivatives is through a finite difference formula such as center difference or a forward/backward difference formulas. The simplest such formula expresses the derivative $y_j^{(1)} = (y_{j+1} - y_j)/\delta$, assuming uniformly sampled data. Formulas can be derived for estimating higher order derivatives with error bounds of the form $O(\delta^k)$ for an order k formula [27]. However, in the presence of noisy data the error in the higher-order derivative can increase without bound. There are multiple approaches to deal with derivatives in the presence of noisy data.

Spectral approaches are based on expressing the signal as a linear combination of basis functions using a family of functions such as sines/cosines (Fourier-transform based methods), Chebyshev polynomials, radial-basis functions (gaussian kernels) [47]. To do so, we solve a least squares problem to find the coefficients of corresponding to each basis function, or use a specialized algorithm such as FFT by assuming that the data represents half a period or a full period of a periodic function.

Rather than fix a basis function, spline-based interpolation methods use splines to model the data and its derivatives. Another class of methods include smoothing techniques based on sliding window polynomial fits to the data to also estimate derivatives (the Savitzky-Golay filter) [42].

Smoothing methods compute derivatives over the smoothed data using a finite difference formula. One such approach uses total-variation smoothing of the data. Often we may combine the smoothing and derivative estimation into a single (but large) least squares problem. Appendix B presents a differentiation approach using "weak form" derivatives that has proven useful in some of our benchmarks.

Appendix A discusses our implementation of various numerical derivative approaches. Numerical derivative estimation has been implemented in python packages such as Kalmangrad [18] and PyNumDiff [48]. The implementation used for our paper outperforms Kalmangrad and overcomes some of its limitations especially for higher order derivative calculations.

Rather than apply a single numerical differentiation technique, we apply nine different techniques summarized in Table 1. After obtaining n higher order derivatives, where n is the number of state variables in the given ODE, we use an evaluation procedure to calculate a score for each technique/hyper-parameter combination. The final derivative is the one that achieves the best score. In order to evaluate performance, we adopt 5 different criteria, some derived from the recent survey by Komarov et al [29].

(1) First, we can use integration to test the accuracy of a derivative since $\int_{t_1}^{t_2} y^{(j+1)}(s)ds = y^{(j)}(t_2) - y^{(j)}(t_1)$. The integration can be approximated using numerical quadrature. Given $y^{(j+1)}$ we derive an error estimate against $y^{(j)}$.
(2) Preserving maxima and minima: we can use techniques that pin point likely positions of maxima and minima in the original signal y and check if the derivative value vanishes at/near these points. We convert the discrepancy between the critical points and the derivative values into a error.
(3) Derivative estimation techniques often suffer from "ringing" or the Gibbs phenomena at or near the start and end points of the signal. We test this by counting the frequency of local minima/maxima at the start and end of the signal (first/last 10% of the data points).
(4) Some of the estimates suffer from the so-called staircase phenomenon wherein higher-order derivative estimates are effectively piecewise constant functions over large time intervals. We measure the total width of such intervals in our derivative estimates.
(5) "Spectral roughness" analyzes the spectral power density of the derivative, taking the ratio of energy contained in the upper half to the frequency spectrum to the total frequency spectrum. This provides a mechanism to reward those estimators that filter noise.

Our approach uses a linear combination of metrics derived from each of these analyses to choose the best derivative estimate. Finally, we drop the first and last $0.1N$ data points/derivative estimates to avoid the large derivative estimates at the start/end of the signal.

6 Empirical Evaluation

We will now present the details of our implementation and an empirical evaluation over a series of 20 nonlinear ODE benchmarks of increasing size and complexity.

6.1 Evaluation Setup

We implemented the elimination approaches discussed in Sections 3.1 and 4 in Python. The Gröbner basis method (Section 3.1) uses SymPy [33] with a lexicographic order that eliminates the unobserved states; parameters are then estimated by minimizing the loss w.r.t the given observable data and its estimated derivatives according to the I/O form using the CMA-ES optimization algorithm [19]. The neural-network elimination (Section 4) uses SymPy to compute the repeated Lie derivatives and PyTorch [37] to perform the stochastic gradient descent to learn the NN weights. The I/O form is represented by a fully connected network with 6 hidden layers of 256 units and SiLU activations (about 3.3×10^5 weights). The network is architected according to the scheme shown in Figure 2. It is trained on 10^4 synthetic samples whose states and parameters

Table 1: Derivative estimators tried for each observable for computing derivatives up to order m. y: observations, Δt: sampling step, D^j: j-th forward-difference matrix.

Method	Smoothing step	Derivatives	Implementation
Smoothing spline	Spline of degree $\max(3, m)$	Analytic derivatives of the spline	SciPy `spline`
Savitzky–Golay [42]	Least-squares cubic fit in a moving window of w samples	Analytic derivatives of the local cubic	SciPy `savgol_filter`
Gaussian kernel	Convolution with a normalized truncated Gaussian	Repeated central differences	SciPy `convolve1d`
Spectral	Highest third of the FFT frequencies set to zero	Multiply by $(i\omega)^k$ and inv. FFT	NumPy `fft`
ℓ_1 trend filter [28]	$\min_x \frac{1}{2}\|y-x\|_2^2 + \alpha\|D^3 x\|_1$ (piecewise-quadratic trend)	Repeated central differences	CVXPY [12]
Total variation [40]	$\min_x \frac{1}{2}\|x-y\|_2^2 + \lambda\|D^1 x\|_1$	Repeated central differences	NumPy
Tikhonov [46]	$\min_u \|u-y\|_2^2 + \gamma\|D^2 u/\Delta t^2\|_2^2$	Repeated central differences	SciPy `spsolve`
Simpson weak form	None	See Appendix B	CVXPY

are drawn uniformly from their bounds, with derivatives computed symbolically as Lie derivatives; 20% of the samples are held out for validation. Training uses Adam (learning rate 10^{-3}, batch size 4096, at most 1000 epochs, learning rate reduced fivefold after 10 epochs without improvement, early stopping after 50 epochs of unchanged validation loss). Parameter inference minimizes the mean squared error between the decoded and the measured observable, first with AdamW (1000 steps, learning rate 0.01, starting from the midpoint of the parameter bounds) and then with L-BFGS (at most 50 iterations, strong Wolfe line search), keeping parameters within their bounds. Derivatives of the data are estimated with 16 estimators (Table 1) with various hyperparameter settings (Appendix A); each runs with a 1 s timeout, the best is selected by the metrics of Section 5, and the first and last 5% of the points along with their derivatives are discarded, as described in Section 5.

FGO Comparison: To enable comparisons, we implemented *Forward solver based Global Optimization* (FGO), which solves the full problem in Eq. 1 over the parameters and initial states. It integrates the ODE with SciPy's adaptive RK45 solver [50] (relative and absolute tolerances 10^{-6} and 10^{-9}), runs a CMA-ES global search ($\sigma = 0.5$, default population size), and refines the result with SciPy's trust-region reflective least squares method.

SI.jl and PE.jl Comparisons: We also compare against two Julia packages (Julia 1.12.7): StructuralIdentifiability.jl (SI.jl, v0.5.25) [45], whose `find_ioequations` computes input-output forms of rational ODEs, and ParameterEstimation.jl (PE.jl, v0.5.1), which combines the identifiability routines of SI.jl with a rational interpolation of the trajectory (Section 1.1). We use its default settings (AAA and Floater–Hormann interpolation, homotopy continuation) and report, among the estimates it returns, the one with the lowest fit error. Both packages apply only to rational systems, and their reported times exclude just-in-time compilation.

Benchmarks: Table 2 describes the 20 benchmarks used in our experimental evaluation. The benchmarks are commonly used ODEs across many domains ranging from oscillator systems to models of biological and epidemiological processes with $2-8$ state variables and up to 9 parameters. Appendix C describes these in detail. Synthetic data are generated by integrating each system with SciPy's `odeint` at the true parameters and adding Gaussian noise with a system-specific standard deviation. Each method has a timeout of

Table 2: Benchmark systems with n states and k parameters. Superscript: class of the vector field (^{P} polynomial, ^{R} rational, ^{N} non-rational).

ID	System	n	k	ID	System	n	k
S0^P	Van der Pol	2	1	S10^P	Hindmarsh-Rose	6	3
S1^N	Damped Pendulum	2	2	S11^P	SEIR Metapopulation	6	6
S2^R	Michaelis-Menten	2	3	S12^P	Brusselator Chain	8	8
S3^P	Lorenz 63	3	2	S13^N	Kuramoto-2	2	3
S4^P	Cubic Chua Circuit	3	4	S14^N	Pendulum Chain-2	4	3
S5^P	Cyclic CTRNN	4	3	S15^N	Torricelli-2	2	3
S6^P	FitzHugh-Nagumo	4	4	S16^N	Torricelli-3	3	4
S7^P	Hyperchaotic Rössler	4	4	S17^N	Kuramoto-3	3	4
S8^P	Lorenz 96	5	6	S18^N	Pendulum Chain-3	6	3
S9^R	Goodwin Oscillator	5	9	S19^N	tanh CTRNN	2	4

300 s per system; building the symbolic representation of a system (Lie derivatives and Gröbner basis) is subject to the same limit. For the Gröbner and neural-network methods, reported inference times exclude elimination, which is reported separately in Table 3. PMSE is the mean squared error between estimated and true parameters; Fit is the mean squared error between the data and the trajectory simulated from the estimates. Each method was run once per system. Experiments ran on an Apple M3 laptop with 24 GB of RAM under macOS 26.7.2, with Python 3.13.13, PyTorch 2.8.0 (using the Apple GPU through MPS), SymPy 1.14.0, SciPy 1.16.2 and cmaes 0.12.0. With the exception of PyTorch, all methods run on CPU.

6.2 Computing I/O Forms

Table 3 compares the various elimination approaches for computing I/O forms. Gröbner elimination timed out on several systems. We also ran the SI.jl by translating each example to Julia and running the routines available in SI.jl to compute I/O forms for the polynomial and rational ODE examples. The neural network based elimination approach converges on all but a few systems. Notably, S12 presents a challenge because of the need to compute repeated Lie derivatives in SymPy causing a timeout. As a result, the NN learning could not get started on this benchmark. Computing Lie derivatives efficiently using modern machine learning frameworks such as Jax, will form an important part of future work. Gröbner elimination timed out on most systems with 4 or more states, and

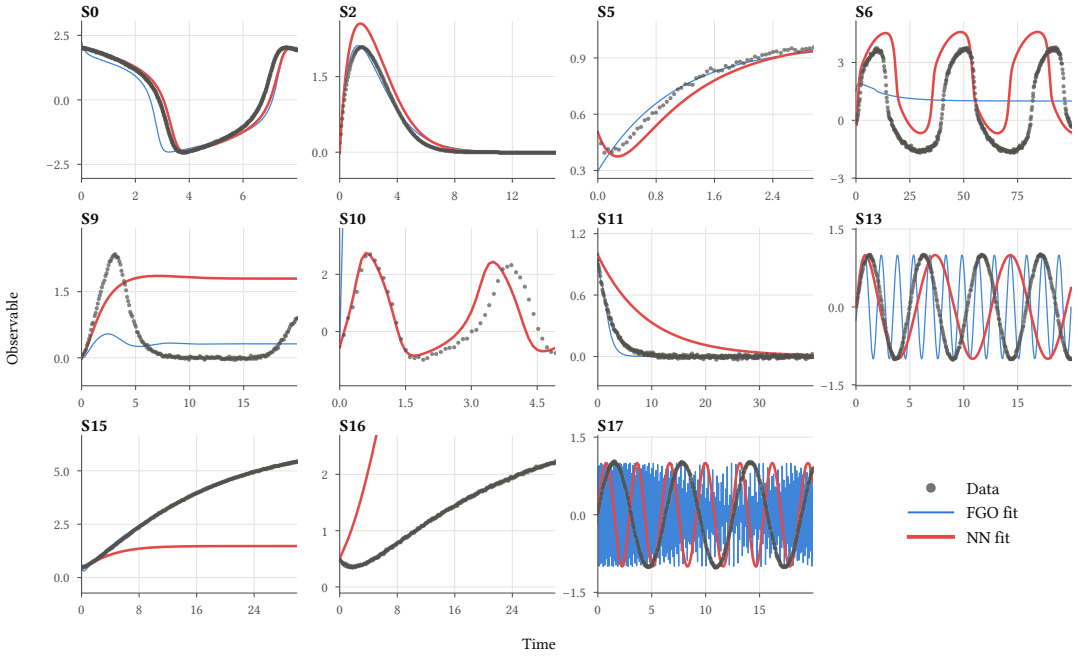


Figure 4: Fits of the direct optimizer (FGO, thin blue) and the neural-network method (NN, thick red) on the benchmarks where both finished, labeled by ID (Table 2). Gray points: observation data. The vertical range is the data range plus 25%; fits that leave it are clipped.

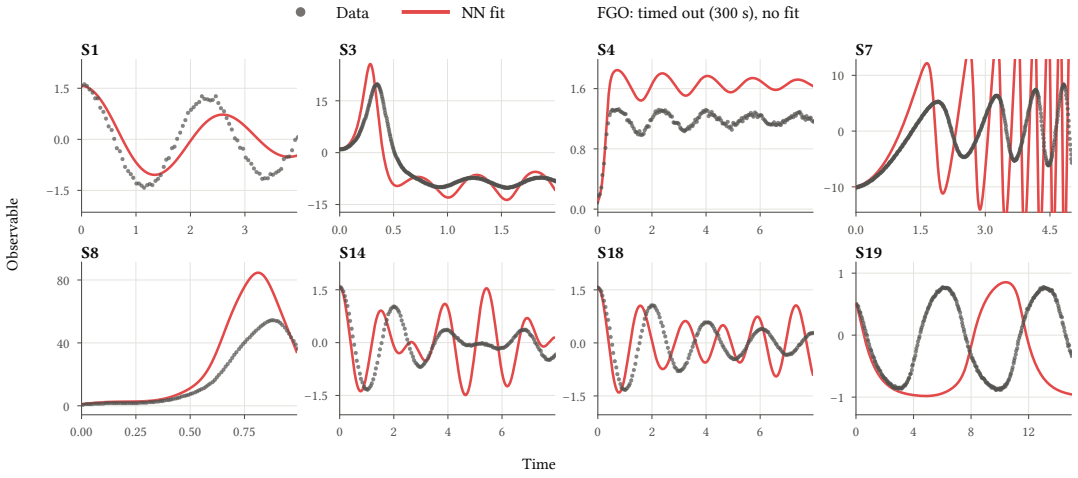


Figure 5: Neural-network (NN) fits on the benchmarks where the direct optimizer (FGO) did not finish within 300 s, labelled by ID (Table 2). Styles as in Figure 4.

SI.jl on S6 and on every system with 5 or more states. Note that the Gröbner basis approach can be improved significantly using a better implementation and handling parameters differently from the state variables.

Although the NN approach succeeds on 19 out of the 20 examples, we note that the final validation loss for the I/O form reported in Table 3 is larger than 1 for four of the examples: S6–S8 and S18.

6.3 Parameter Inference Results

Table 4 compares the performance of the various approaches to parameter inference from data, generated with low standard deviation Gaussian noise ($\sigma \in [0.02, 0.05]$). We report on the parameter MSE (PMSE) and the "Fit": the MSE distance between the original observable data and the simulated observable trajectory using the inferred parameter values. In some cases, our approach recovers

Table 3: Elimination time (s) with Gröbner bases, StructuralIdentifiability.jl (SI.jl) input-output equations, and neural-network (NN) training, with the final NN test loss. Superscripts as in Table 2. T/O: over 300 s. N/A: not applicable (Gröbner needs ^{P}, SI.jl needs ^{P} or ^{R}). †Symbolic setup timeout.

	Gröbner	SI.jl	Neural Network	
ID	Time	Time	Time	Loss
S0^P	2.28	<0.01	10.28	8.28×10^{-6}
S1^N	N/A	N/A	10.85	2.42×10^{-1}
S2^R	N/A	0.14	11.32	8.30×10^{-6}
S3^P	T/O	0.14	17.52	5.84×10^{-5}
S4^P	4.59	0.14	10.84	3.19×10^{-7}
S5^P	T/O	0.18	8.42	1.00×10^{-4}
S6^P	T/O	T/O	14.02	1.69×10^{0}
S7^P	9.48	0.14	19.55	1.47×10^{2}
S8^P	T/O	T/O	8.47	1.23×10^{1}
S9^R	N/A	T/O	9.41	9.20×10^{-3}
S10^P	T/O	T/O	12.65	5.84×10^{-4}
S11^P	T/O	T/O	12.53	5.65×10^{-2}
S12^P	T/O†	T/O	T/O†	–
S13^N	N/A	N/A	8.48	5.16×10^{-1}
S14^N	N/A	N/A	17.16	1.32×10^{-1}
S15^N	N/A	N/A	15.41	1.26×10^{-2}
S16^N	N/A	N/A	12.37	5.23×10^{-2}
S17^N	N/A	N/A	8.21	2.15×10^{-1}
S18^N	N/A	N/A	10.29	1.20×10^{0}
S19^N	N/A	N/A	17.23	1.46×10^{-4}

the ground truth parameters as evidenced by the small PMSE value. However, it is possible that parameters that are far away from the "ground truth" produce fits that are as good. More significantly, our approach achieves relatively good curve fits to the data, showing that the inferred parameters are able to explain the underlying data. Figure 4 compares curve fits for some benchmarks including the fits obtained from the FGO method. The remaining fits where the FGO method did not finish are shown in Figure 5. In general, the NN method finished within 25 seconds on all the examples with the parameter inference phase running much faster than the learning of the I/O forms. Note that all of these examples are challenging nonlinear benchmarks and the fit is performed under partial observations. As a result, the overall problem is quite challenging. Many examples yield good curve fits for the NN method including S0, S2, S5 and S10. Most examples yield fits that match the overall shape of the dynamics but exhibit features such as mismatched oscillation period or a mismatch in the peak or settling values (S1, S4, S6, S7, S8, S11, S13, S14, S18). The approach fails to fit the data in examples such as S9, S15, S16, and S19. Although timing out on 9 out of the 20 benchmarks, the FGO method performs better on 8 out of 11 benchmarks where both methods produce a result in terms of the goodness of fit metric. However, the NN method produces parameters that are closer to the ground truth for more instances when compared to FGO. A key reason could be that the NN approach does not infer the initial conditions $\mathbf{x}(0)$ and optimizes just over the parameters.

All results are from a single noise realization per benchmark; evaluating variability across noise seeds will be part of future work.

6.4 Increasing Noise

Figure 6 shows the results of running the proposed method on data with increasing noise standard deviations for three of the best performing benchmarks. Interestingly, even when we add noise that can be as large as 30% of the underlying signal in some cases, the quality of the fits are maintained and even improved in some cases. The use of a suite of numerical differentiation tools was a key factor in ensuring that the quality of the fit did not degrade even when noise was added. Under low noise, every benchmark instance had a different numerical differentiation approach that performed best. However, as the noise was increased, the total variation regularization approach proved to be the most effective in producing the best derivative estimates.

6.5 Discussion

Although our approach does not fit every benchmark well, it returns estimates on 19 of 20 benchmarks, more than any other method we compared, and its parameter estimates are closer to the ground truth than FGO's on the benchmarks where both finish. Even in the absence of perfect fits to the data, our approach often fits important features of the data across many of the benchmarks. The use of multiple numerical differentiation approaches including smoothing techniques makes our approach robust to noise in the data, as demonstrated qualitatively in Figure 6. Future work will focus on addressing some of the key limitations of our current approach: (a) the size of the data is fixed to 10^4 samples for all examples though more samples can be generated on demand; and (b) we do not adapt the size of the neural networks or other learning-time hyper parameters to the various benchmarks – we expect that doing so will yield improvements in our results. Finally, for polynomial and rational systems our approach can be applied on top of elimination: for instance, we could eliminate a few of the latent variables and then build neural network-based forms to write the remaining latent variables in terms of observables and their derivatives. Such an approach would have the advantage of being more precise and efficient at the same time.

7 Conclusion

We introduced a two-stage method for parameter inference from partial observations: we compute an I/O form of the ODE, either symbolically or approximately with a neural network, and then solve a least-squares problem on the I/O form using derivatives estimated from the data. Theorem 3.2 guarantees that such a form exists locally under a constant-rank condition, which allows the approach to extend beyond polynomial and rational models. On 20 nonlinear benchmarks, our approach returned estimates on 19, whereas solver-based optimization timed out on 9. Our main limitations are the cost of computing symbolic derivatives, which prevented training on our largest benchmark, and a fixed network size and sample budget that left a few learned I/O forms inaccurate. Future work will address these limitations, evaluate the method over multiple noise realizations, combine partial symbolic elimination with learned forms for polynomial systems, and extend the approach to inverse problems for PDEs.

Table 4: Comparison of optimization methods. FGO: direct optimization; Grb.: Gröbner elimination; PE: ParameterEstimation.jl; NN: neural-network elimination (ours). T: time (s); for Grb. and NN it excludes elimination (Table 3). PMSE: parameter mean squared error. Fit: mean squared error between the data and the fitted trajectory. Superscripts as in Table 2. N/A: not applicable. T/O: over 300 s. N/S: no solution returned. †Symbolic setup exceeded 300 s. ‡Degenerate estimate $z_i(0) = \pm 1$, so θ_i is unbounded.

	FGO			Grb.			PE			NN		
ID	T	PMSE	Fit	T	PMSE	Fit	T	PMSE	Fit	T	PMSE	Fit
$S0^P$	66.38	1.35×10^{0}	2.89×10^{-1}	0.09	2.88×10^{-2}	5.59×10^{-2}	4.36	1.34×10^{0}	4.78×10^{0}	1.00	5.70×10^{-2}	1.21×10^{-1}
$S1^N$		T/O			N/A			N/A		1.72	3.70×10^{0}	3.35×10^{-1}
$S2^R$	83.17	4.35×10^{0}	6.40×10^{-3}		N/A		5.72	3.42×10^{-18}	2.30×10^{-16}	1.04	6.08×10^{-1}	4.14×10^{-2}
$S3^P$		T/O			T/O		2.95	N/S		3.79	5.42×10^{1}	2.41×10^{1}
$S4^P$		T/O		2.22	7.40×10^{0}	2.33×10^{-2}	5.03	N/S		1.01	1.54×10^{-2}	2.18×10^{-1}
$S5^P$	8.01	3.79×10^{1}	1.70×10^{-3}		T/O		3.44	$\infty^{\ddagger}$	2.72×10^{-3}	1.26	2.87×10^{-1}	2.70×10^{-3}
$S6^P$	99.62	2.68×10^{1}	4.21×10^{0}		T/O		12.69	N/S		1.40	9.60×10^{-2}	5.39×10^{0}
$S7^P$		T/O		6.34	1.08×10^{0}	3.45×10^{1}	6.11	N/S		1.63	1.20×10^{0}	1.32×10^{2}
$S8^P$		T/O			T/O			T/O		2.27	3.12×10^{0}	2.45×10^{2}
$S9^R$	60.35	1.37×10^{1}	3.98×10^{-1}		N/A			T/O		2.14	2.18×10^{-1}	2.22×10^{0}
$S10^P$	22.87	3.13×10^{3}	1.26×10^{1}		T/O			T/O		1.29	2.56×10^{-1}	4.02×10^{-1}
$S11^P$	28.87	1.66×10^{1}	2.40×10^{-3}		T/O			T/O		2.89	3.38×10^{-1}	4.73×10^{-2}
$S12^P$		T/O†			T/O†			T/O			T/O†	
$S13^N$	19.33	1.01×10^{1}	9.45×10^{-1}		N/A			N/A		1.49	3.99×10^{-1}	1.06×10^{0}
$S14^N$		T/O			N/A			N/A		3.00	1.20×10^{1}	5.71×10^{-1}
$S15^N$	8.04	2.25×10^{2}	4.00×10^{-3}		N/A			N/A		4.26	1.95×10^{-1}	6.16×10^{0}
$S16^N$	15.45	3.65×10^{1}	3.57×10^{-4}		N/A			N/A		4.41	3.06×10^{-1}	1.34×10^{2}
$S17^N$	200.26	2.92×10^{3}	9.62×10^{-1}		N/A			N/A		1.66	4.81×10^{-1}	9.14×10^{-1}
$S18^N$		T/O			N/A			N/A		2.80	1.20×10^{1}	5.86×10^{-1}
$S19^N$		T/O			N/A			N/A		1.07	2.17×10^{-1}	1.14×10^{0}

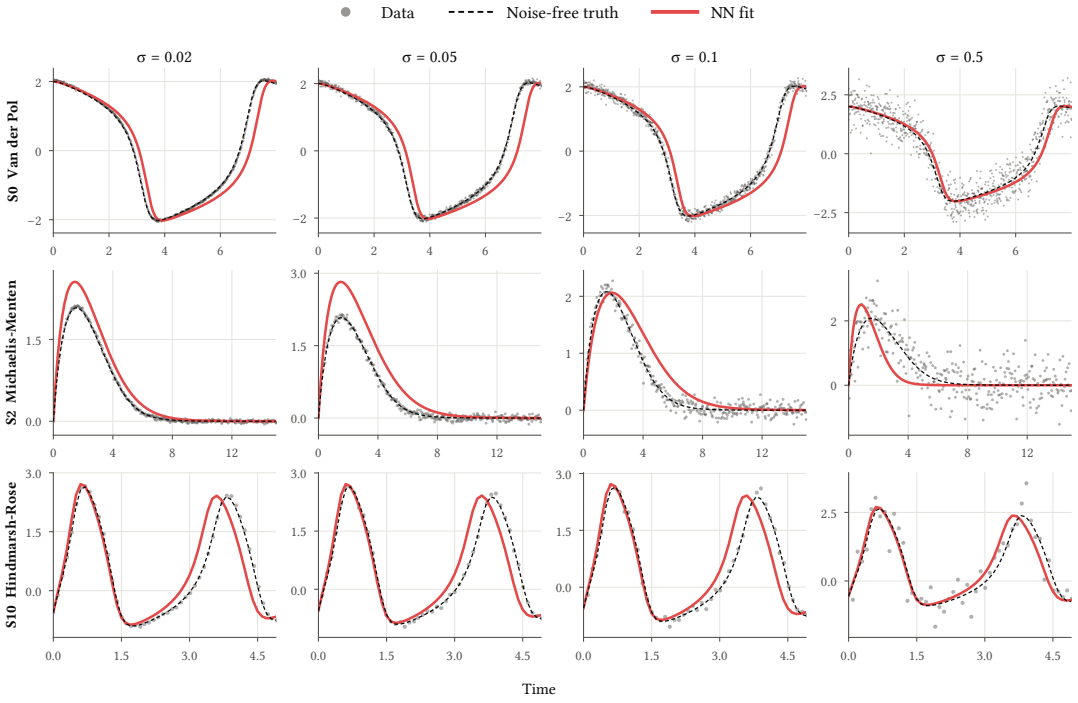


Figure 6: Neural-network (NN) fits under increasing measurement noise on three benchmarks, labelled by ID (Table 2). Columns give the noise standard deviation σ. Gray points: noisy observations (one noise realization per panel). Dashed black: the noise-free observable. Red: the NN fit.

References

[1] F. Anstett-Collin, L. Denis-Vidal, and G. Millérioux. A priori identifiability: An overview on definitions and approaches. *Annual Reviews in Control*, 50:139–149, 2020. ISSN 1367-5788.

[2] Maksat Ashyraliyev, Yacine Fomekong-Nanfack, Jaap A. Kaandorp, and Jaap G. Blom. Systems biology: Parameter estimation for biochemical models. *The FEBS Journal*, 276(4):886–902, 2009. doi: 10.1111/j.1742-4658.2008.06844.x.

[3] Yonathan Bard. *Nonlinear Parameter Estimation*. Academic Press, New York, San Francisco, London, 1974. ISBN 9780120782505.

[4] Oren Bassik, Yosef Berman, Soo Go, Hoon Hong, Ilia Ilmer, Alexey Ovchinnikov, Chris Rackauckas, Pedro Soto, and Chee Yap. Robust parameter estimation for rational ordinary differential equations, 2023.

[5] Richard Bellman. A new method for the identification of systems. *Mathematical Biosciences*, 5(1–2):201–204, 1969.

[6] David M. Bortz, Daniel A. Messenger, and Vanja Dukic. Direct estimation of parameters in ode models using WENDy: Weak-form estimation of nonlinear dynamics. *Bulletin of Mathematical Biology*, 85:110, 2023. doi: 10.1007/s11538-023-01208-6. URL https://doi.org/10.1007/s11538-023-01208-6.

[7] François Boulier, Daniel Lazard, François Ollivier, and Michel Petitot. Computing representations for radicals of finitely generated differential ideals. *Applicable Algebra in Engineering, Communication and Computing*, 20(1):73–121, 2009. doi: 10.1007/s00200-009-0091-7.

[8] Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 113(15):3932–3937, 2016. doi: 10.1073/pnas.1517384113.

[9] Ben Calderhead, Mark Girolami, and Neil Lawrence. Accelerating bayesian inference over nonlinear differential equations with gaussian processes. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems*, volume 21. Curran Associates, Inc., 2008.

[10] Antonin Chambolle. An algorithm for total variation minimization and applications. *J. Math. Imaging Vis.*, 20(1–2):89–97, January 2004. ISSN 0924-9907.

[11] David Cox, John Little, and Donald O'Shea. *Ideals, Varieties and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra*. Springer, 1991.

[12] Steven Diamond and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83): 1–5, 2016.

[13] Frank Dondelinger, Dirk Husmeier, Simon Rogers, and Maurizio Filippone. Ode parameter inference using adaptive gradient matching with gaussian processes. In Carlos M. Carvalho and Pradeep Ravikumar, editors, *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics*, volume 31 of *Proceedings of Machine Learning Research*, pages 216–228, Scottsdale, Arizona, USA, 29 Apr–01 May 2013. PMLR.

[14] Ruiwen Dong, Christian Goodbrake, Heather A. Harrington, and Gleb Pogudin. Differential elimination for dynamical models via projections with applications to structural identifiability. *SIAM Journal on Applied Algebra and Geometry*, 7(1): 194–235, 2023. doi: 10.1137/22M1469067.

[15] Thomas W. Dubé. The structure of polynomial ideals and gröbner bases. *SIAM J. Comput.*, 19(4):750–773, June 1990. ISSN 0097-5397. doi: 10.1137/0219053.

[16] Stephen P. Ellner, Yohannes Seifu, and R. H. Smith. Fitting population dynamic models to time-series data by gradient matching. *Ecology*, 83:2256–2270, 2002.

[17] Mark S. Gockenbach, David R. Reynolds, and William W. Symes. Automatic differentiation and the adjoint state method. In George Corliss, Christiane Faure, Andreas Griewank, Laurent Hascoët, and Uwe Naumann, editors, *Automatic Differentiation of Algorithms*, pages 233–245. Springer, New York, NY, 2002. doi: 10.1007/978-1-4613-0075-5_18.

[18] Hugo Hadfield. Kalmangrad: Automated, smooth n'th order derivatives of noisy, non-uniformly sampled time series data. GitHub repository, 2025. https://github.com/hugohadfield/kalmangrad/.

[19] Nikolaus Hansen. The cma evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772*, 2016. doi: 10.48550/arXiv.1604.00772. version 2, revised 2023.

[20] Heather A. Harrington and Robert A. Van Gorder. Reduction of dimension for nonlinear dynamical systems. *Nonlinear Dynamics*, 88(2):715–734, 2017. doi: 10.1007/s11071-016-3272-5.

[21] Nora Heitzman-Breen, Vanja Dukic, and David M. Bortz. A practical identifiability criterion leveraging weak-form parameter estimation, 2025.

[22] Hoon Hong, Alexey Ovchinnikov, Gleb Pogudin, and Chee K. Yap. Global identifiability of differential models. *Communications on Pure and Applied Mathematics*, 73(9):1831–1879, 2020. doi: 10.1002/cpa.21848.

[23] Roman Hovorka, Roman Hovorka, and Paolo Vicini. Parameter estimation. In Ewart Carson and Claudio Cobelli, editors, *Modeling Methodology for Physiology and Medicine*, Biomedical Engineering, pages 107–151. Academic Press, San Diego, 2001. ISBN 978-0-12-160245-1. doi: https://doi.org/10.1016/B978-012160245-1/50006-4.

[24] Alberto Isidori. *Nonlinear Control Systems*. Springer, London, 3 edition, 2002.

[25] Carine Jauberthie and Nathalie Verdière. Bounded-error parameter estimation using integro-differential equations for hindmarsh–rose model. *Algorithms*, 15(6):179, 2022. doi: 10.3390/a15060179. URL https://www.mdpi.com/1999-4893/15/6/179.

[26] Deepak Kapur and Yagiti N. Lakshman. Elimination methods: An introduction. In Bruce Randall Donald, Deepak Kapur, and Joseph L. Mundy, editors, *Symbolic and Numerical Computation for Artificial Intelligence*, pages 45–72. Academic Press, 1992.

[27] Ishtiaq Rasool Khan and Ryoji Ohba. New finite difference formulas for numerical differentiation. *Journal of Computational and Applied Mathematics*, 126(1):269–276, 2000. ISSN 0377-0427.

[28] Seung-Jean Kim, Kwangmoo Koh, Stephen Boyd, and Dimitry Gorinevsky. l_1 trend filtering. *SIAM Review*, 51(2):339–360, 2009. doi: 10.1137/070690274.

[29] Pavel Komarov, Floris van Breugel, and J. Nathan Kutz. A taxonomy of numerical differentiation methods, 2025. URL https://arxiv.org/abs/2512.09090.

[30] Lennart Ljung. System identification. In John Webster, editor, *Wiley Encyclopedia of Electrical and Electronics Engineering*, pages 1–19. Wiley, Hoboken, NJ, USA, 2017.

[31] Lennart Ljung and Torkel Glad. On global identifiability for arbitrary model parametrizations. *Automatica*, 30(2):265–276, 1994. doi: 10.1016/0005-1098(94)90029-9.

[32] Daniel A. Messenger and David M. Bortz. Weak SINDy: Galerkin-based data-driven model selection. *Multiscale Modeling & Simulation*, 19(3):1474–1497, 2021. doi: 10.1137/20M1343166.

[33] Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, Amit Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller, Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Štěpán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony Scopatz. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103, 2017. doi: 10.7717/peerj-cs.103. URL https://doi.org/10.7717/peerj-cs.103.

[34] Carmen G. Moles, Pedro Mendes, and Julio R. Banga. Parameter estimation in biochemical pathways: A comparison of global optimization methods. *Genome Research*, 13(11):2467–2474, 2003. doi: 10.1101/gr.1262503.

[35] Ryan E. Morrison. Exact reduction of the generalized lotka–volterra equations via integral and algebraic substitutions. *Computation*, 9(5):49, 2021. doi: 10.3390/computation9050049.

[36] Alexey Ovchinnikov, Gleb Pogudin, and Peter Thompson. Parameter identifiability and input–output equations. *Applicable Algebra in Engineering, Communication and Computing*, 34(2):165–182, 2023. doi: 10.1007/s00200-021-00486-8.

[37] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. *PyTorch: an imperative style, high-performance deep learning library*. Curran Associates Inc., Red Hook, NY, USA, 2019.

[38] James O. Ramsay, Giles Hooker, David Campbell, and Jiguo Cao. Parameter estimation for differential equations: A generalized smoothing approach. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 69(5):741–796, 2007.

[39] Joseph Fels Ritt. *Differential Algebra*, volume 33 of *American Mathematical Society Colloquium Publications*. American Mathematical Society, New York, 1950.

[40] Leonid I. Rudin, Stanley Osher, and Emad Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D Nonlinear Phenomena*, 60(1-4):259–268, November 1992. doi: 10.1016/0167-2789(92)90242-F.

[41] Nic Rummel, Daniel A. Messenger, Stephen Becker, Vanja Dukic, and David M. Bortz. WENDy for nonlinear-in-parameters ODEs, 2025.

[42] Abraham Savitzky and Marcel J. E. Golay. Smoothing and differentiation of data by simplified least squares procedures. *Analytical Chemistry*, 36(8):1627–1639, 1964. doi: 10.1021/ac60214a047.

[43] Lewis B. Sheiner and Jean-Louis Steimer. Pharmacokinetic/pharmacodynamic modeling in drug development. *Annual Review of Pharmacology and Toxicology*, 40:67–95, 2000. doi: 10.1146/annurev.pharmtox.40.1.67. URL https://doi.org/10.1146/annurev.pharmtox.40.1.67.

[44] Michael Spivak. *Calculus on Manifolds*. W. A. Benjamin, New York, 1965.

[45] StructuralIdentifiability.jl contributors. StructuralIdentifiability.jl: Fast and automatic structural identifiability software for ODE systems. https://github.com/SciML/StructuralIdentifiability.jl, 2026. Julia package, accessed 2026-11-04.

[46] Andrei Nikolaevich Tikhonov. On the solution of incorrectly posed problems and the method of regularization. *Doklady Akademii Nauk SSSR*, 151(3):501–504, 1963.

[47] Lloyd N. Trefethen. *Spectral Methods in MATLAB*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2000.

[48] F. van Breugel, P. Komarov, and Y. Liu. PyNumDiff: Methods for numerical differentiation of noisy data in python. GitHub repository, 2025. https://github.com/florisvb/PyNumDiff/.

[49] J. M. Varah. A spline least squares method for numerical parameter estimation in differential equations. *SIAM Journal on Scientific and Statistical Computing*, 3 (1):28–46, 1982.

[50] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

[51] S. Zhu, N. Verdière, L. Denis-Vidal, and D. Kateb. Identifiability analysis and parameter estimation of a chikungunya model in a spatially continuous domain. *Ecological Complexity*, 34:80–88, 2018.

[52] K. Łakomiec and K. Fujarewicz. Parameter estimation of non-linear models using adjoint sensitivity analysis. In J. Sobecki, V. Boonjing, and S. Chittayasothorn, editors, *Advanced Approaches to Intelligent Information and Database Systems*, volume 551 of *Studies in Computational Intelligence*. Springer, Cham, 2014. doi: 10.1007/978-3-319-05503-9_6. URL https://doi.org/10.1007/978-3-319-05503-9_6.

A Derivative Estimation Techniques

Our approach to estimating derivatives runs a bank of derivative estimation techniques on the observed signal and selects one with heuristics. Early attempts focused on finding a single ideal technique, but no single technique handled the variety of examples. Running each technique with several hyperparameter settings further improved the estimates. In total we run 22 configurations of eight techniques, summarized in Table 1.

Throughout, $y_1, \ldots, y_N$ are the observations at uniformly spaced times with step Δt, and m is the highest derivative order required, which equals the number of states. Every technique returns a smoothed signal $\hat{y}$ and its derivatives up to order m. Except where noted, the derivatives are computed by applying second-order central differences repeatedly to $\hat{y}$.

- **Spline Estimation** fits a univariate smoothing spline with `UnivariateSpline` in SciPy, which wraps FITPACK. The smoothing factor s is an upper bound on the residual sum of squares, $\sum_i (y_i - \hat{y}(t_i))^2 \le s$: FITPACK adds knots until the bound is met, so $s = 0$ interpolates the data and larger s gives a smoother spline. Because s is an absolute bound, its effect depends on the noise level and the number of samples. The spline has degree $\max(3, m)$, and its derivatives are computed analytically. FITPACK supports degrees up to 5, so this technique is only used for systems with $m \le 5$. We tested $s = 0, 10^{-4}, 0.01, 0.1, 1, 10$.
- **Savitzky-Golay** filtering fits a polynomial by least squares within a moving window and evaluates the polynomial and its derivatives at the window center [42]. Our implementation uses SciPy's `savgol_filter`. The hyperparameters are the odd window length w and the polynomial degree $d < w$. We used $d = 3$ and $w = 7, 15, 31$. Since the local polynomial is cubic, derivatives above order 3 are zero.
- **Gaussian Estimation** convolves the signal with a normalized Gaussian kernel, a linear smoother that suppresses high-frequency noise. The kernel width σ is measured in samples, the kernel is truncated at 4σ, and the signal is extended at its boundaries by repeating its first and last values. We used SciPy's `convolve1d` and tested $\sigma = 0.5, 1.0$.
- **Spectral Estimation** computes the discrete Fourier transform of the signal with NumPy's `fft`, sets the highest third of the frequencies to zero, and differentiates in the frequency domain by multiplying by $(i\omega)^k$ before transforming back. The cutoff is fixed, so this technique has no hyperparameter. The transform treats the signal as periodic, so non-periodic signals can show artifacts near the boundaries.
- **Trend Filter Estimation** applies ℓ_1 trend filtering [28], which solves
$$\min_x \tfrac{1}{2}\|y - x\|_2^2 + \alpha \|D^3 x\|_1,$$
where D^3 takes third-order differences. The ℓ_1 penalty makes the solution piecewise quadratic. We set up the problem in CVXPY [12], solved it with the Clarabel solver, and tested $\alpha = 0.01, 0.1, 1.0$.
- **Total Variation Regularization (TVR)** denoises the signal by solving the Rudin-Osher-Fatemi problem [40]
$$\min_x \tfrac{1}{2}\|x - y\|_2^2 + \lambda \|D^1 x\|_1,$$
which combines ℓ_2 data fidelity with an ℓ_1 penalty on the signal's first differences. TVR removes noise while keeping sharp transitions, but it can introduce staircase artifacts. We solve the dual problem with Chambolle's projection algorithm [10], running 50 iterations with step size 0.24. We tested $\lambda = 0.001, 0.01, 0.1$.
- **Tikhonov Regularization** [46] smooths the signal by penalizing the ℓ_2 norm of its second differences:
$$\min_u \|u - y\|_2^2 + \gamma \|Lu\|_2^2,$$
where L is the second-difference matrix divided by Δt^2. This global smoothness constraint acts as a low-pass filter that keeps low-frequency trends. The minimizer solves the sparse linear system $(I + \gamma L^\top L)\, u = y$, which we solve with SciPy's `spsolve`. Because L is scaled by Δt^{-2}, the effect of γ depends on the sampling step. We tested $\gamma = 0.1, 1.0, 10.0$.
- **Simpson Weak Form** estimates the first derivative g at every sample by solving the regularized least-squares problem $\min_g \|Ag - b\|_2^2 + \varepsilon \|g\|_2^2$ with $\varepsilon = 10^{-4}$. Each row of $Ag = b$ equates a quadrature of g over one, two or three steps, using the trapezoidal, Simpson's 1/3 or Simpson's 3/8 rule, to the corresponding difference of the data. Higher derivatives are obtained from g by central differences. This technique does not smooth the signal: it returns the data as $\hat{y}$. See Appendix B.

B Weak-Form Derivative Estimation

In this section, we briefly describe the weak-form derivative estimation procedure that forms one of the procedures used in our suite of numerical differentiation procedures.

Given data $y_1, \ldots, y_K$ at times $t_1, \ldots, t_K$ wherein $t_i = i\delta$, we wish to estimate upto n derivatives of the underlying signal $y(t)$.

Let $\tilde{y}_i^{(j)}$ for $i = 1, \ldots, N$ and $j = 1, \ldots, n$ represent the unknown j^{th} derivative of $y(t)$ at time t_i. By convention, we will take $\tilde{y}_i^{(0)}$ to be a (smoothed) estimate of the data point y_i. We can express integral constraints that represent relations between these derivatives:

$$\int_{t_a}^{t_b} \tilde{y}^{(j+1)}(s)ds = \tilde{y}^{(j)}(t_b) - \tilde{y}^{(j)}(t_a)\,. \tag{5}$$

The integral on the left can be approximated using a numerical quadrature rule such as Simpson's rule, yielding a linear expression that involves the unknowns $\tilde{y}^{(j+1)}(t_k)$ for $t_i \le t_k \le t_j$. We will express this equation as

$$I_{a,b,j}(\tilde{y}^{(j+1)}(t_a), \ldots, \tilde{y}^{(j+1)}(t_b), \tilde{y}^{(j)}(t_a), \tilde{y}^{(j)}(t_b)) = 0\,. \tag{6}$$

A more sophisticated form of the constraint uses a known "test function" $g(t)$ that is differentiable and imposes the so-called "weak form" constraint (see Bortz et al [6]):

$$\begin{aligned} &\textstyle\int_{t_a}^{t_b} g(s)\tilde{y}^{(j+1)}(s)ds \;+\; \int_{t_a}^{t_b} \dot{g}(s)\tilde{y}^{(j)}(s)ds \\ &\quad = g(t_b)\tilde{y}^{(j)}(t_b) - g(t_a)\tilde{y}^{(j)}(t_a)\,. \end{aligned} \tag{7}$$

The left-hand side is approximated using quadrature to yield a linear equality relationship. The advantage of this approach is that we may use several test functions. Further, Eq. 5 can be seen as an instance with the test function $g(t) = 1$.

We collect all the equations of the form Eq. 7 for various values of start/end times t_a, t_b, a fixed set of test functions $g \in \{g_1, \ldots, g_l\}$ and derivative indices $j \in \{0, \ldots, n-1\}$. We convert these equations into a least squares problem with a quadratic loss function ℓ over $\tilde{y}^{(j)}(t_i)$, for $i = 1, \ldots, N$, $j = 0, \ldots, n$.

The overall optimization has the following form:

$$\min \begin{pmatrix} \sum_{i=1}^{N} (y_i - \tilde{y}^{(0)}(t_i))^2 + \\ \ell\left(\tilde{y}^{(0)}(t_1), \ldots, \tilde{y}^{(n)}(t_N)\right) + \\ \epsilon \sum_{i=1}^{N-1} \sum_{j=1}^{n} \|\tilde{y}^{(j)}(t_{i+1}) - \tilde{y}^{(j)}(t_i)\|^2 \end{pmatrix}. \tag{8}$$

This is a combination of three different loss terms, enforcing the fit with the data y_i, enforcing how well the successive derivatives match up with each other through the given test functions and finally the total variation metric. Since we use the L_2 norm, the overall least-squares problem can be solved efficiently and to a global optimum as a linear least squares problem.

C Systems

We incorporated a total of 20 systems to benchmark as shown in Table 5. We consulted well known systems throughout the natural sciences and carefully designed observables, parameters, and time domains. In algebraic systems, to ensure the parameters could be found after Gröbner elimination, observables were precisely constructed in each case, and with known constants added to the equation. Initial conditions were used to generate synthetic data but were not provided to the optimization techniques. In addition to observables, time domains were constructed per system based on its dynamics. In most oscillatory systems, we made the time domain equal to one period, but found that in some cases, our method could reliably work with domains over several periods. We standardized time domains to start at $t = 0$ and end at an integer value.

Table 5: Benchmark systems, numbered as in the experiments, with n states and k fitted parameters. Ground truth lists the fitted parameters; constants are given with the equations. Observations are sampled at $t = 0, \Delta t, 2\Delta t, \ldots$ up to but not including the end of the time domain, with Gaussian noise of standard deviation σ.

ID	System	Equations	Observable	Ground Truth	State Bounds	Parameter Bounds	Time Domain	Noise (σ)
S0	**Van der Pol Oscillator** (n=2, k=1)	$\dot{x} = y$ $\dot{y} = \mu(1 - x^2)y - x$ $x(0) = 2.0$ $y(0) = 0.0$	x	$\mu = 2.0$	$x \in [-4, 4]$ $y \in [-8, 8]$	$\mu \in [0.5, 5]$	$t \in [0, 8)$ $\Delta t = 0.01$	0.01
S1	**Damped Pendulum** (n=2, k=2)	$\dot{\theta} = \omega$ $\dot{\omega} = -\frac{g}{L}\sin\theta - \frac{b}{m}\omega$ $L = 1.0,\ m = 1.0$ $\theta(0) = \pi/2$ $\omega(0) = 0.0$	θ	$g = 9.81$ $b = 0.2$	$\theta \in [-2\pi, 2\pi]$ $\omega \in [-10, 10]$	$g \in [5, 15]$ $b \in [0, 2]$	$t \in [0, 4)$ $\Delta t = 0.05$	0.05
S2	**Michaelis-Menten Kinetics** (n=2, k=3)	$\dot{D} = -k_a D$ $\dot{C} = k_a D - \frac{V_m C}{K_m + C}$ $D(0) = 5.0$ $C(0) = 0.0$	C	$k_a = 0.8$ $V_m = 2.0$ $K_m = 1.5$	$D, C \in [0, 10]$	$k_a \in [0.1, 2]$ $V_m \in [0.5, 5]$ $K_m \in [0.1, 5]$	$t \in [0, 15)$ $\Delta t = 0.05$	0
S3	**Lorenz 63** (n=3, k=2)	$\dot{x} = \sigma(y - x)$ $\dot{y} = x(\rho - z) - y$ $\dot{z} = xy - \beta z$ $\beta = 8/3$ $x(0) = y(0) = z(0) = 1.0$	x	$\sigma = 10$ $\rho = 28$	$x, y \in [-25, 25]$ $z \in [0, 60]$	$\sigma \in [1, 20]$ $\rho \in [10, 50]$	$t \in [0, 2)$ $\Delta t = 0.005$	0.02
S4	**Cubic Chua Circuit** (n=3, k=4)	$\dot{x} = \alpha\left(y - c_1 x - c_3 x^3\right)$ $\dot{y} = x - y + z$ $\dot{z} = -\beta y$ $x(0) = 0.1$ $y(0) = z(0) = 0.0$	x	$\alpha = 10.0$ $\beta = 14.87$ $c_1 = -0.7$ $c_3 = 0.5$	$x, y, z \in [-5, 5]$	$\alpha \in [5, 15]$ $\beta \in [10, 20]$ $c_1 \in [-1.5, 0.5]$ $c_3 \in [0.1, 1]$	$t \in [0, 8)$ $\Delta t = 0.05$	0.02
S5	**Cyclic CTRNN** (n=4, k=3)	$\dot{x}_i = -x_i + z_i$ $\dot{z}_i = w(1 - z_i^2)(-x_j + z_j)$ $z_i = \tanh(wx_j + \theta_i)$ $(i, j) \in \{(1, 2), (2, 1)\}$ $x_1(0) = 0.5,\ z_1(0) = -0.5$ $x_2(0) = 0.964,\ z_2(0) = -0.935$	x_1	$w = -3.0$ $\theta = (0.5, -0.2)$	$x_i \in [-2.5, 2.5]$ $z_i \in [-1, 1]$	$w \in [-5, -1]$ $\theta_i \in [-1, 1]$	$t \in [0, 3)$ $\Delta t = 0.05$	0.01
S6	**FitzHugh-Nagumo** (n=4, k=4)	$\dot{v}_i = v_i - \frac{v_i^3}{3} - w_i + I_i$ $+ D(v_j - v_i)$ $\dot{w}_i = \epsilon(v_i + a_i - b_i w_i)$ $(i, j) \in \{(1, 2), (2, 1)\}$ $D = 0.1,\ \epsilon = 0.08$ $b = (0.8, 0.7)$ $v(0) = (-0.261, 0.471)$ $w(0) = (0.066, -0.007)$	$v_1 + 2w_1$	$I = (0.5, 0.3)$ $a = (0.7, 0.75)$	$v_i \in [-2.5, 2.5]$ $w_i \in [-1, 2]$	$I_i \in [0, 1]$ $a_i \in [0.5, 1]$	$t \in [0, 100)$ $\Delta t = 0.2$	0.05

Continued on next page...

Table 5 – Continued from previous page

ID	System	Equations	Observable	Ground Truth	State Bounds	Parameter Bounds	Time Domain	Noise (σ)
S7	**Hyperchaotic Rössler** (n=4, k=4)	$\dot{x} = -y - z$ $\dot{y} = x + ay + w$ $\dot{z} = b + xz$ $\dot{w} = -cz + dw$ $x(0) = -10.0$ $y(0) = -2.0$ $z(0) = w(0) = 0.0$	x	$a = 0.25$ $b = 3$ $c = 0.5$ $d = 0.05$	$x, y, z, w \in [-30, 30]$	$a, c, d \in [0, 1]$ $b \in [1, 5]$	$t \in [0, 5)$ $\Delta t = 0.01$	0.02
S8	**Lorenz 96** (n=5, k=6)	$\dot{x}_i = (x_{i+1} - x_{i-2})x_{i-1}$ $- \nu x_i + F_i$ $i \in \{1, \ldots, 5\}$, indices mod 5 $x(0) = (1, 3, -2, 5, 0.5)$	x_1^2	$\nu = 1.0$ $F = (8.0, 8.5, 7.5,$ $8.2, 7.8)$	$x_i \in [-10, 15]$	$\nu \in [0.8, 1.2]$ $F_i \in [5, 15]$	$t \in [0, 1)$ $\Delta t = 0.01$	0.01
S9	**Goodwin Oscillator** (n=5, k=9)	$\dot{x}_1 = \frac{k_1}{1 + x_5^p} - b_1 x_1$ $\dot{x}_i = \alpha_i x_{i-1} - b_i x_i$ $i \in \{2, \ldots, 5\}$ $k_1 = 1,\ p = 4$ $x_i(0) = 0.1$	x_1^2	$b_1 = 0.5$ $b_{2\ldots5} = 0.6$ $\alpha_{2\ldots4} = 1.0$ $\alpha_5 = 0.9$	$x_i \in [0, 3]$	$b_i \in [0.1, 1]$ $\alpha_i \in [0.5, 2]$	$t \in [0, 20)$ $\Delta t = 0.1$	0.02
S10	**Coupled Hindmarsh-Rose** (n=6, k=3)	$\dot{x}_i = y_i - x_i^3 + 3x_i^2 - z_i + I$ $+ \epsilon(x_j - x_i)$ $\dot{y}_i = 1 - 5x_i^2 - y_i$ $\dot{z}_i = r(4(x_i + 1.6) - z_i)$ $(i, j) \in \{(1, 2), (2, 1)\}$ $x(0) = (-0.547, -0.530)$ $y(0) = (0.075, -0.621)$ $z(0) = (-0.937, 0.187)$	x_1	$I = 3.2$ $r = 0.006$ $\epsilon = 0.15$	$x_i \in [-2, 2.5]$ $y_i \in [-10, 5]$ $z_i \in [-1, 5]$	$I \in [2, 5]$ $r \in [0.001, 0.01]$ $\epsilon \in [0, 0.5]$	$t \in [0, 5)$ $\Delta t = 0.1$	0.05
S11	**SEIR Metapopulation** (n=6, k=6)	$\dot{S}_i = -\beta_i S_i I_i + D(S_j - S_i)$ $\dot{E}_i = \beta_i S_i I_i - \sigma_i E_i$ $+ D(E_j - E_i)$ $\dot{I}_i = \sigma_i E_i - \gamma_i I_i + D(I_j - I_i)$ $(i, j) \in \{(1, 2), (2, 1)\}$ $D = 0.01$ $S(0) = (0, 0)$ $E(0) = (0, 0.99)$ $I(0) = (0.01, 1.0)$	E_2	$\beta = (1.5, 1.2)$ $\sigma = (0.3, 0.4)$ $\gamma = (0.5, 0.6)$	$S_i, E_i, I_i \in [0, 1]$	$\beta_i \in [0.5, 3]$ $\sigma_i, \gamma_i \in [0.1, 1]$	$t \in [0, 40)$ $\Delta t = 0.1$	0.01
S12	**Brusselator Chain** (n=8, k=8)	$\dot{u}_i = A_i + u_i^2 v_i - (B_i + 1)u_i$ $+ D(u_{i-1} - 2u_i + u_{i+1})$ $\dot{v}_i = B_i u_i - u_i^2 v_i$ $+ D(v_{i-1} - 2v_i + v_{i+1})$ $i \in \{1, \ldots, 4\},\ D = 1$ $u_0 \equiv u_1,\ u_5 \equiv u_4$ (same for v) $u_i(0) = v_i(0) = 1.0$	$\sum_{i=1}^{4} i\, u_i$	$A = (1.0, 1.2,$ $1.0, 1.2)$ $B = (3.0, 2.8,$ $3.0, 2.8)$	$u_i, v_i \in [0, 6]$	$A_i \in [0.5, 2]$ $B_i \in [1, 5]$	$t \in [0, 10)$ $\Delta t = 0.05$	0.03
S13	**Kuramoto** (n=2, k=3)	$\dot{\theta}_i = \omega_i + \frac{K}{2} \sum_{j=1}^{2} \sin(\theta_j - \theta_i)$ $\theta(0) = (0, 2)$	$\sin \theta_1$	$K = 0.6$ $\omega = (1.0, 1.3)$	$\theta_i \in [-\pi, \pi]$	$K \in [0, 2]$ $\omega_i \in [0.5, 2]$	$t \in [0, 20)$ $\Delta t = 0.05$	0.01

Continued on next page...

Table 5 – Continued from previous page

ID	System	Equations	Observable	Ground Truth	State Bounds	Parameter Bounds	Time Domain	Noise (σ)
S14	**Pendulum Chain** (n=4, k=3)	$\dot{\theta}_i = \omega_i$ $\dot{\omega}_i = -g \sin\theta_i - b\,\omega_i + k(\theta_{i-1} - 2\theta_i + \theta_{i+1})$ $i \in \{1,2\},\ \theta_0 \equiv \theta_1,\ \theta_3 \equiv \theta_2$ $\theta(0) = (\pi/2, 0)$ $\omega(0) = (0,0)$	θ_1	$g = 9.81$ $b = 0.2$ $k = 2.0$	$\theta_i \in [-\pi, \pi]$ $\omega_i \in [-8, 8]$	$g \in [5, 15]$ $b \in [0, 1]$ $k \in [0.5, 5]$	$t \in [0, 8)$ $\Delta t = 0.05$	0.01
S15	**Torricelli Cascade** (n=2, k=3)	$\dot{h}_1 = q - a_1\sqrt{h_1}$ $\dot{h}_2 = a_1\sqrt{h_1} - a_2\sqrt{h_2}$ $h_i(0) = 0.5$	h_2	$q = 1.0$ $a = (0.5, 0.4)$	$h_i \in [0.01, 10]$	$q \in [0.5, 2]$ $a_i \in [0.1, 1]$	$t \in [0, 30)$ $\Delta t = 0.1$	0.01
S16	**Torricelli Cascade** (n=3, k=4)	$\dot{h}_1 = q - a_1\sqrt{h_1}$ $\dot{h}_i = a_{i-1}\sqrt{h_{i-1}} - a_i\sqrt{h_i}$ $i \in \{2,3\}$ $h_i(0) = 0.5$	h_3	$q = 1.0$ $a = (0.5, 0.4, 0.6)$	$h_i \in [0.01, 10]$	$q \in [0.5, 2]$ $a_i \in [0.1, 1]$	$t \in [0, 30)$ $\Delta t = 0.1$	0.01
S17	**Kuramoto** (n=3, k=4)	$\dot{\theta}_i = \omega_i + \frac{K}{3}\sum_{j=1}^{3}\sin(\theta_j - \theta_i)$ $\theta(0) = (0, 2, -2)$	$\sin\theta_1$	$K = 0.6$ $\omega = (1.0, 1.3, 0.7)$	$\theta_i \in [-\pi, \pi]$	$K \in [0, 2]$ $\omega_i \in [0.5, 2]$	$t \in [0, 20)$ $\Delta t = 0.05$	0.01
S18	**Pendulum Chain** (n=6, k=3)	$\dot{\theta}_i = \omega_i$ $\dot{\omega}_i = -g \sin\theta_i - b\,\omega_i + k(\theta_{i-1} - 2\theta_i + \theta_{i+1})$ $i \in \{1,2,3\},\ \theta_0 \equiv \theta_1,\ \theta_4 \equiv \theta_3$ $\theta(0) = (\pi/2, 0, 0)$ $\omega(0) = (0,0,0)$	θ_1	$g = 9.81$ $b = 0.2$ $k = 2.0$	$\theta_i \in [-\pi, \pi]$ $\omega_i \in [-8, 8]$	$g \in [5, 15]$ $b \in [0, 1]$ $k \in [0.5, 5]$	$t \in [0, 8)$ $\Delta t = 0.05$	0.01
S19	**tanh CTRNN** (n=2, k=4)	$\dot{x}_1 = -x_1 + \tanh(ax_1 + cx_2 + \theta_1)$ $\dot{x}_2 = -x_2 + \tanh(ax_2 - cx_1 + \theta_2)$ $x(0) = (0.5, -0.5)$	x_1	$a = 2.0$ $c = 2.0$ $\theta = (0.2, -0.3)$	$x_i \in [-1.5, 1.5]$	$a \in [1, 4]$ $c \in [0.5, 4]$ $\theta_i \in [-1, 1]$	$t \in [0, 15)$ $\Delta t = 0.05$	0.01