A Direct Method to Learn States and Parameters of Ordinary Differential Equations

Ramin Raziperchikolaei
Department of Computer Science
University of California, Merced
rraziperchikolaei@ucmerced.edu

Harish S. Bhat
Department of Mathematics
University of Utah
hbhat@math.utah.edu

Abstract

Though ordinary differential equations (ODE) are used extensively in science and engineering, the task of learning ODE parameters from noisy observations still presents challenges. To address these challenges, we propose a direct method that involves alternating minimization of an objective function over the filtered states and parameters. This objective function directly measures how well the filtered states and parameters satisfy the ODE, in contrast to many existing methods that use separate objectives over the observations, filtered states, and parameters. As we show on several ODE systems, as compared to state-of-the-art methods, the direct method exhibits increased robustness (to noise, parameter initialization, and hyperparameters), decreased training times, and improved accuracy in estimating both filtered states and parameters. The direct method involves only one hyperparameter that plays the role of an inverse step size. We show how the direct method can be used with general multistep numerical discretizations, and demonstrate its performance on systems with up to $d = 40$ dimensions. The code of our algorithms can be found in the authors' web pages.

1 Introduction

Ordinary differential equations (ODEs) are widely used to describe the behavior of time-varying systems in various fields of science and engineering. When researchers derive ODEs that govern the behavior of a system, the parameter values for these ODEs are often unknown. An important task is to estimate ODE parameters given noisy experimental measurements.

ODE parameter estimation is challenging for two main reasons: (1) the data is noisy, and (2) most ODEs do not have an analytical solution. Without these two challenges, the parameters could be estimated by solving a simple regression problem.

Two popular methods to solve the ODE parameter estimation problem are Bayesian and spline-based. Spline-based methods try to fit (cubic) splines to the noisy data and then estimate the parameters [8, 9, 30, 31, 34]. Estimators other than splines, such as smoothing kernels and local polynomials, have also been tried [10, 16, 23]. These methods have many hyperparameters such as the smoothing parameter, the number of knots, the positions of the knots, etc. These methods are sensitive to the initialization of the parameters. Bayesian approaches [7, 11, 14] make assumptions about the type of the noise and distribution of the data, which are not necessarily correct. Also, these methods need to set the prior distributions, variance of the noise, kernel width, etc. very carefully to produce reasonable results. Another disadvantage of these methods is their large training time.

In this paper, we propose an entirely different approach to address the issues of previous methods. We apply alternating minimization to an objective function that directly measures how well the states and parameters satisfy the ODE system. In each step of our learning algorithm, the states and parameters move slowly away from their initializations, stepping toward the desired solution. Our method is fast

to train, easy to implement, robust with respect to parameter initializations, and works well under different magnitudes and types of noise.

We call our method a *direct method* because: (1) *it learns the states directly in the original space*, instead of learning them indirectly by fitting a smoothed function to the observations, and (2) *it learns the states and parameters jointly using one, unified objective function*, which gives better results than learning them in separate objective functions.

The rest of the paper is organized as follows. In Section 2 we define the problem mathematically. In Section 3, we explain our proposed method in detail. We show the advantages of our method with several experiments in Section 4.

1.1 Related Work

There have been several different approaches to estimate ODE parameters. Nonlinear least squares methods start with an initial guess for the parameters that is iteratively updated to bring the model's predictions close to measurements [3, 4, 18, 19]. These methods are slow and have convergence issues when the initial parameters are far from the true parameters. The main advantage of our method over these methods is that we learn states and parameters simultaneously, which makes our method robust to initialization.

In [34] it was first suggested to fit splines to the noisy observations, and then consider these splines as the clean states. Since the derivatives of the splines can be found easily, the parameters are estimated by solving a regression problem. The idea of fitting splines and other smooth functions to the observations has been used extensively in the literature [8–10, 16, 23, 30, 31].

The idea of learning the splines and parameters jointly, leading to better results, has been explored [30, 31]. Spline-based methods have many hyperparameters that require careful tuning, and are also sensitive to initialization. Our method learns parameters and states jointly, but it does not fit a smooth function to the observations. Our direct learning strategy decreases the number of hyperparameters and makes the results robust to initialization.

Bayesian approaches have also been very popular recently [7, 11, 13, 14]. Such approaches typically have large training times and depend sensitively on a large number of hyperparameters (priors, noise variances, etc.). Another disadvantage of these methods, as has been mentioned in [14], is that they cannot simultaneously learn clean states and parameters. In [14], a variational inference approach has been used to overcome this problem, but the method is not applicable to all ODEs. Often, Bayesian methods make multiple assumptions about the distribution of the data and noise.

Finally, let us mention that the problem we solve here is that of simultaneous filtering (recovering clean states from noisy observations) and parameter estimation. Many well-known nonlinear ODE filtering methods, including extended and ensemble Kalman filters as well as particle filters, are online methods that make Gaussian assumptions. Our approach is a distribution-free, batch method.

2 Problem Definition: ODE Parameter Estimation

Consider a dynamical system in $\mathbb{R}^d$ with state $\mathbf{x}(t)$ at time t. We assume the system depends on a parameter $\boldsymbol{\theta} \in \mathbb{R}^p$, in which case the time-evolution of the state is given by

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

At T distinct times $\{t_i\}_{i=1}^T$, we have noisy observations $\mathbf{y}(t_i) \in \mathbb{R}^d$:

$$\mathbf{y}(t_i) = \mathbf{x}(t_i) + \mathbf{z}(t_i), \quad i = 1, \ldots, T \tag{2}$$

where $\mathbf{z}(t_i) \in \mathbb{R}^d$ is the noise of the observation at time t_i. We represent the set of T d-dimensional states, noises, and observations by $\mathbf{X}$, $\mathbf{Z}$, and $\mathbf{Y} \in \mathbb{R}^{d \times T}$, respectively. For concision, in what follows, we write the time t_i as a subscript, i.e., $\mathbf{x}_{(t_i)}$ instead of $\mathbf{x}(t_i)$.

In this paper, we assume that the form of the vector field $\mathbf{f}(\cdot)$ is known. The goal is to use $\mathbf{Y}$ to estimate $\boldsymbol{\theta}$ and $\mathbf{X}$ (or equivalently $\mathbf{Z}$). Examples of $\mathbf{f}(\cdot)$ and $\boldsymbol{\theta}$ can be found in Section 4.

3 Our Direct Approach: Learning ODE Parameters and States

The first step of our approach is to discretize the ODE (1) in time using multistep methods. In this section, we focus on the explicit Euler method (which is a one-step method) because it is intuitive, makes our formulation simple, and helps the reader to focus purely on our novel approach. Later, we explain how higher-order multistep methods can be used in our formulation. We also compare multistep methods with different orders in our experimental results section.

The explicit Euler method discretizes the ODE (1) for the T time points as follows:

$$\mathbf{x}_{(t_{i+1})} - \mathbf{x}_{(t_i)} = \mathbf{f}(\mathbf{x}_{(t_i)}, \boldsymbol{\theta})\Delta_i, \quad i = 1, ..., T-1 \tag{3}$$

where $\Delta_i = t_{i+1} - t_i$. In (3), both states $\mathbf{X}$ and parameters $\boldsymbol{\theta}$ are unknown; we are given only the noisy observations $\mathbf{Y}$. With this discretization, we formulate our objective function:

$$E(\mathbf{X}, \boldsymbol{\theta}) = \sum_{i=1}^{T-1} \left\| \mathbf{x}_{(t_{i+1})} - \mathbf{x}_{(t_i)} - \mathbf{f}(\mathbf{x}_{(t_i)}, \boldsymbol{\theta})\Delta_i \right\|^2 \tag{4}$$

Before continuing, it is worth analyzing our objective function E. Let us define *fidelity* as the degree to which the estimated states $\mathbf{X}$ and parameters $\boldsymbol{\theta}$ actually satisfy the ODE. Our objective function directly measures time-discretized fidelity; if $E = 0$, then we have a solution to the time-discretized ODE. Note that the observations $\mathbf{Y}$ do not appear explicitly—we use $\mathbf{Y}$ to initialize Alg. (1).

In contrast, most prior work maximizes the likelihood function that stems from assuming the noise $\mathbf{Z}$ in (2) is Gaussian with mean zero and covariance matrix Σ. In such approaches, fidelity (as defined above) is treated as a secondary term, e.g., using a penalty term [31] or using a probabilistic model that accounts for temporal discretization errors [1, 2]. In the present work, we seek to show that making fidelity the primary objective yields better estimates of $\mathbf{X}$ and $\boldsymbol{\theta}$.

Theorem 1. *The objective function E in* (4) *has an infinite number of optimal solutions, each of which makes the objective value* 0*.*

Proof. Assign arbitrary real vectors to $\boldsymbol{\theta}$ and $\mathbf{x}_{(t_1)}$. Then we use the following equation sequentially with $i = 1, 2, \ldots, T-1$:

$$\mathbf{x}_{(t_{i+1})} = \mathbf{x}_{(t_i)} + \mathbf{f}(\mathbf{x}_{(t_i)}, \boldsymbol{\theta})\Delta_i. \tag{5}$$

By computing the states $\mathbf{x}_{(t_2)}, \ldots, \mathbf{x}_{(t_T)}$ in this way, we ensure that each term in the objective function $E(\mathbf{X}, \boldsymbol{\theta})$—see (4)—is zero. Since the objective function in (4) is always greater than or equal to zero, we achieve a global minimum. Because $\boldsymbol{\theta}$ and $\mathbf{x}_{(t_1)}$ are arbitrary, an infinite number of solutions exist. □

It is easy to see that Theorem 1 generalizes to the case where we replace the Euler method—in the definition of $E(\mathbf{X}, \boldsymbol{\theta})$—by a higher-order, explicit ODE solver.

Additionally, suppose we fix $\boldsymbol{\theta}$ and consider E to be a function of $\mathbf{X}$ alone. Then, if we also fix the value of $\mathbf{x}_{(t_1)}$, we see that there is a unique global minimizer. We return to this point below when we discuss minimizing over $\mathbf{X}$.

Our goal is to use the observations $\mathbf{Y}$ to find the clean (or filtered) states $\mathbf{X}$ and the parameters $\boldsymbol{\theta}$. To achieve this goal, we will not directly compute a global minimizer of (4), for reasons that we now explain. In what follows, we make a distinction between *learned* (asterisks) and *predicted* (hats) states and parameters.

We represent the *learned* states and parameters by $\mathbf{X}^*$ and $\boldsymbol{\theta}^*$, where $\mathbf{X}^*, \boldsymbol{\theta}^* = \operatorname{argmin}_{\mathbf{X}, \boldsymbol{\theta}} E(\mathbf{X}, \boldsymbol{\theta})$. We propose an iterative approach that optimizes $\mathbf{X}$ and $\boldsymbol{\theta}$ alternately for $n = 1, \ldots, N$ iterations. We represent the states and parameters *learned* at iteration n by $\mathbf{X}^{*(n)}$ and $\boldsymbol{\theta}^{*(n)}$.

We also represent the *predicted* states at iteration n by $\hat{\mathbf{X}}^{(n)}$. The predicted states are achieved by setting $\boldsymbol{\theta} = \boldsymbol{\theta}^{*(n)}$ and $\mathbf{x}_{(t_1)} = \mathbf{x}^{*(n)}_{(t_1)}$, and then stepping forward in time using an ODE solver—e.g., Euler's method in (5) or a multistep method as in (11)—to generate the states at $t = 2, \ldots, T$. By Theorem 1, this predicted state/parameter pair, $(\hat{\mathbf{X}}^{(n)}, \boldsymbol{\theta}^{*(n)})$, is a global minimizer of (4).

We initialize the states with the observations: $\mathbf{X}^{*(0)} = Y$. Then we update $\boldsymbol{\theta}$ and $\mathbf{X}$ alternately for N iterations. At each iteration n, our method has two steps: 1) fix $\mathbf{X} = \mathbf{X}^{*(n-1)}$ and optimize (4) over $\boldsymbol{\theta}$, and 2) fix $\boldsymbol{\theta} = \boldsymbol{\theta}^{*(n)}$ and optimize a *modified* version of (4) over $\mathbf{X}$.

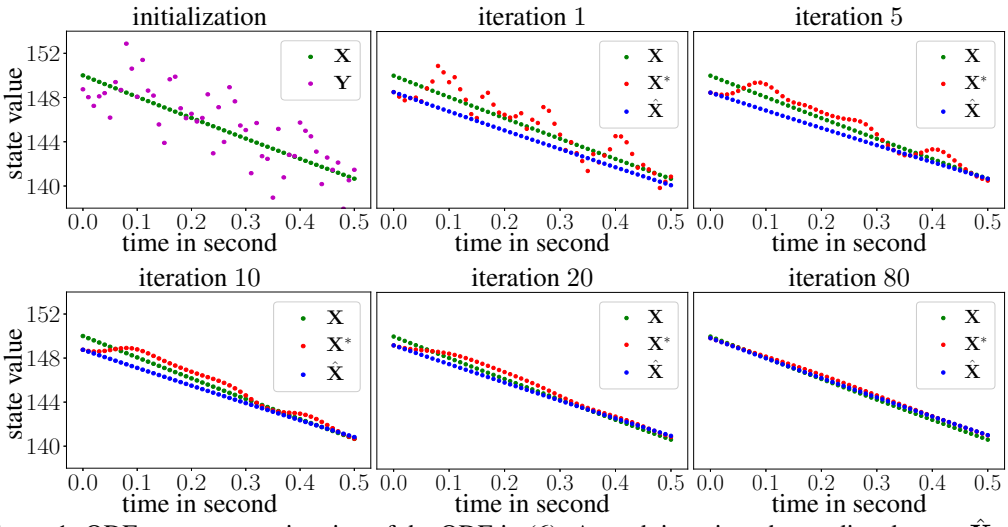

Figure 1: ODE parameter estimation of the ODE in (6). At each iteration, the predicted states $\hat{\mathbf{X}}$ and the learned states $\mathbf{X}^*$ move closer to each other and the clean states.

Before we detail the two steps, let us explain why and how we modify the objective function in the second step (optimization over $\mathbf{X}$). Assume we are in the first iteration $n = 1$, and that we have learned $\boldsymbol{\theta}^{*(1)}$ in the first step. In the second step, if we try to optimize (4) (as it is) over the states, then the predicted states give us the optimal solution. In other words, $\mathbf{X}^{*(1)} = \hat{\mathbf{X}}^{(1)}$ is the optimal solution. Since this makes the objective 0, our algorithm should stop. To see why this is undesirable, note that we initialized the states with the observations. Our hope is that the clean states are somewhat close to the observations. The predicted states are in general far away from the observations. By setting $\mathbf{X}^{*(1)} = \hat{\mathbf{X}}^{(1)}$, we lose all the information that was given to us by the observations. For this reason, we modify the objective function in (4) such that the changes in states are small from one iteration to another. We want $\mathbf{X}^{*(n)}$ to be close to $\mathbf{X}^{*(n-1)}$ and $\mathbf{X}^{*(n+1)}$.

Let us clarify with a simple 1-dimensional example. Consider the following ODE:

$$\dot{x}_{(t)} = \frac{dx_{(t)}}{dt} = 9.8 - \theta_0 x_{(t)}. \tag{6}$$

This ODE is achieved by applying Newton's second law of motion to a falling object of mass $m = 1$, where $x_{(t)}$ shows the velocity of the object at time t. Here, θ_0 is the parameter which is unknown. Fig. 1 shows how our algorithm works. To generate the clean states (shown with green circles), we solve (6) over the time interval $[0, 0.5]$ with parameters $\theta_0 = .196$ and $\mathbf{x}_{(0)} = 150$. Our algorithm does not have access to these clean states. The observations (shown with magenta circles) are achieved by adding Gaussian noise to the clean states in the initialization step. Now examine the plots for iteration 1 to 80 in Fig. 1, where we show learned and predicted states after the initialization. The predicted states (blue circles) are far from the clean states. The learned states (red circles) are not on top of the predicted states, because *we forced them to be close to their previous values (observations)*. At each iteration, the first step (learning $\boldsymbol{\theta}$), pushes the predicted states $\hat{\mathbf{X}}$ closer to the learned states $\mathbf{X}^*$. The second step pushes $\mathbf{X}^*$ close to $\hat{\mathbf{X}}$. As we can see, after about 80 iterations, both $\mathbf{X}^*$ and $\hat{\mathbf{X}}$ are very close to the clean states. The pseudocode of our method can be found in Algorithm 1. In the following, we explain the two steps in more detail.

3.1 Optimization Over the ODE Parameter $\boldsymbol{\theta}$

At iteration n, we fix $\mathbf{X}$ to equal the learned states from the previous iteration: $\mathbf{X} = \mathbf{X}^{*(n-1)}$, and then minimize the objective function (4) over $\boldsymbol{\theta}$ only:

$$\boldsymbol{\theta}^{*(n)} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, E(\mathbf{X}^{*(n-1)}, \boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \sum_{i=1}^{T-1} \left\| \mathbf{x}_{(t_{i+1})}^{*(n-1)} - \mathbf{x}_{(t_i)}^{*(n-1)} - \mathbf{f}(\mathbf{x}_{(t_i)}^{*(n-1)}, \boldsymbol{\theta})\Delta_i \right\|^2 \tag{7}$$

This is a d-dimensional regression problem where the ith input and output are $\mathbf{x}_{(t_i)}^{*(n-1)}$ and $(\mathbf{x}_{(t_{i+1})}^{*(n-1)} - \mathbf{x}_{(t_i)}^{*(n-1)})/\Delta_i$, respectively. We optimize this objective using the LBFGS algorithm, implemented in

Python via the scipy.optimize.minimize function. Throughout this work, when using LBFGS, we use automatic differentiation to supply the optimizer with gradients of the objective function.

3.2 Optimization Over the States X

At iteration n, we fix $\boldsymbol{\theta}$ to equal the parameters learned in the previous step: $\boldsymbol{\theta} = \boldsymbol{\theta}^{*(n)}$. By Theorem 1, directly minimizing $E(\mathbf{X}, \boldsymbol{\theta}^{*(n)})$ will yield a global minimizer such that $E = 0$, terminating the optimization procedure. To avoid this fate, and to learn better states and parameters, we modify the objective function in (2):

$$\mathbf{X}^{*(n)} = \underset{\mathbf{X}}{\operatorname{argmin}} \left\{ E(\mathbf{X}, \boldsymbol{\theta}^{*(n)}) + \lambda \left\| \mathbf{X} - \mathbf{X}^{*(n-1)} \right\|^2 \right\}, \tag{8}$$

where $E(\mathbf{X}, \boldsymbol{\theta})$ has been defined in (4). For $\lambda > 0$, the penalty term encourages $\mathbf{X}^{*(n)}$ to remain close to $\mathbf{X}^{*(n-1)}$. As we increase λ, we tighten this closeness. To optimize this objective function, we again use the LBFGS algorithm, as in the first step.

We can also motivate (8) using the notion of proximal operators [29]. Let us fix $\boldsymbol{\theta} = \boldsymbol{\theta}^{*(n)}$. When the Δ_i are all identically zero, the objective function E reduces to a quadratic form:

$$\sum_{i=1}^{T-1} \|\mathbf{x}_{(t_{i+1})} - \mathbf{x}_{(t_i)}\|^2.$$

If we now fix $\mathbf{x}_{(t_1)}$, the Hessian with respect to the remaining variables $\mathbf{X}_{2:} = \{\mathbf{x}_{(t_i)}\}_{i=2}^{T}$ has strictly positive eigenvalues. The eigenvalues of the Hessian of E are continuous functions of the parameters Δ_i; hence there exists $\epsilon > 0$ such that if $0 \leq \Delta_i < \epsilon$, the Hessian remains positive definite. The upshot is that, under these conditions, $\widetilde{E}(\mathbf{X}_{2:}) = E(\mathbf{x}_{(t_1)}, \mathbf{X}_{2:}, \boldsymbol{\theta}^{*(n)})$ is a strictly convex function of $\mathbf{X}_{2:}$. This is why it has a unique global minimizer $\hat{\mathbf{X}}^{(n)}$, as mentioned above.

Additionally, because $\widetilde{E}$ is convex, we can form the proximal operator

$$\operatorname{prox}_{\mu\widetilde{E}}(\mathbf{X}^{*(n-1)}) = \underset{\mathbf{X}_{2:}}{\operatorname{argmin}} \left\{ \widetilde{E}(\mathbf{X}_{2:}) + (2\mu)^{-1} \|\mathbf{X}_{2:} - \mathbf{X}_{2:}^{*(n-1)}\|^2 \right\} \tag{9}$$

We see that when $\mu \to +\infty$, the proximal step returns $\hat{\mathbf{X}}^{(n)}$. For $\mu > 0$ sufficiently small, as shown in [29], this proximal step is approximately equal to a gradient descent step:

$$\operatorname{prox}_{\mu\widetilde{E}}(\mathbf{X}^{*(n-1)}) = \mathbf{X}_{2:}^{*(n-1)} - \mu \nabla \widetilde{E}(\mathbf{X}_{2:}^{*(n-1)}) + o(\mu) \tag{10}$$

One can see that our λ in (8) plays the role of $(2\mu)^{-1}$ in (9). Hence large values of λ correspond to small gradient descent step sizes, and vice versa.

Though E in (8) is non-convex, we can view it as a set-valued proximal operator [22]; still, the analogy with convex proximal operators gives valuable intuition. When $\lambda = 0$, we see that (8) ignores $\mathbf{X}^{*(n-1)}$ and directly outputs $\hat{\mathbf{X}}^{(n)}$ with $\mathbf{x}_{(t_1)}$ arbitrary. As λ increases, (8) approximates a small step that starts from $\mathbf{X}^{*(n-1)}$ and proceeds in the direction of the minimizer $\hat{\mathbf{X}}^{(n)}$.

3.3 Higher-Order Discretization

As we mentioned before, our approach is not limited to the Euler discretization. Here, we show that it is straightforward to use higher-order discretization methods.

The idea of multistep (m-step) methods is to use the previous m states to predict the next state. Let us consider the general formulation of the explicit linear m-step method to discretize the ODE (1):

$$\mathbf{x}_{(t_{i+1})} = \sum_{j=0}^{m-1} a_j \mathbf{x}_{(t_{i-j})} + \Delta_i \sum_{j=0}^{m-1} b_j \mathbf{f}(\mathbf{x}_{(i-j)}, \boldsymbol{\theta}), \tag{11}$$

where Δ_i is the time step. There are several strategies to determine the coefficients $\{a_j\}_{j=0}^{m-1}$ and $\{b_j\}_{j=0}^{m-1}$. Each strategy leads to a specific family of multistep methods. For example, the Adams-Bashforth method approximates $\mathbf{f}(\cdot)$ with a polynomial of order m to find the coefficients and predict the next state. Further information regarding different strategies can be found in [20, 28].

Algorithm 1 Pseudo-code of our proposed method

Input: A set of T noisy observations $\mathbf{Y} = [\mathbf{y}_{(t_1)}, \ldots, \mathbf{y}_{(t_T)}] \in \mathbb{R}^{d \times T}$, the time differences $\{\Delta_i = t_{i+1} - t_i\}_{i=1}^{T-1}$, the shape of function $\mathbf{f}(\cdot)$ in (1), the value of the hyperparameter λ, and the initial guess of the parameters $\boldsymbol{\theta}^{*(0)}$.

1: $\mathbf{X}^{*(0)} = \mathbf{Y}$
2: $n = 0$
3: **repeat**
4: $\quad n = n + 1$
5: $\quad$ • Compute $\boldsymbol{\theta}^{*(n)}$ given $\mathbf{X}^{*(n-1)}$ via (7) [Euler] or (13a) [multistep].
6: $\quad$ • Compute $\mathbf{X}^{*(n)}$ given $\boldsymbol{\theta}^{*(n)}$ via (8) [Euler] or (13b) [multistep].
7: **until** convergence
8: Compute the predicted states $\hat{\mathbf{X}}$ by repeatedly applying (5) [Euler] or (11) [multistep], where $\boldsymbol{\theta} = \boldsymbol{\theta}^{*(n)}$ and $\mathbf{x}_{(t_1)} = \mathbf{x}_{(t_1)}^{*(n)}$.
9: **return** $\boldsymbol{\theta}^{*(n)}$ and $\hat{\mathbf{X}}$ as the estimated parameters and predicted states.

Note that to use m-step methods to predict the state at time i, we need its previous m states. To predict the states $\{\mathbf{x}_i\}_{i=2}^{m}$ (the first few states), the maximum order we can use is $i-1$, because there are only $i-1$ states before the state $\mathbf{x}_i$. In general, to predict $\mathbf{x}_i$ we use a multistep method of order $\min(i-1, m)$.

When using a general m-step discretization method, we define our objective function as follows:

$$E_{\text{m-step}}(\mathbf{X}, \boldsymbol{\theta}) = \sum_{i=1}^{T-1} \left\| \mathbf{x}_{(t_{i+1})} - \sum_{j=0}^{k-1} a_j \mathbf{x}_{(t_{i-j})} - \Delta_i \sum_{j=0}^{k-1} b_j \mathbf{f}(\mathbf{x}_{(i-j)}, \boldsymbol{\theta}) \right\|^2, \tag{12}$$

where $k = \min(i-1, m)$ is the order of the discretization method to predict the state $\mathbf{x}_i$. Now, we repeat the following steps until convergence:

$$\boldsymbol{\theta}^{*(n)} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, E_{\text{m-step}}(\mathbf{X}^{*(n-1)}, \boldsymbol{\theta}) \tag{13a}$$

$$\mathbf{X}^{*(n)} = \underset{\mathbf{X}}{\operatorname{argmin}} \left\{ E_{\text{m-step}}(\mathbf{X}, \boldsymbol{\theta}^{(*n)}) + \lambda \|\mathbf{X} - \mathbf{X}^{*(n-1)}\|^2 \right\} \tag{13b}$$

For both steps, we use LBFGS, initialized with $\boldsymbol{\theta}^{*(n-1)}$ and $\mathbf{X}^{*(n)}$, respectively.

Convergence. There are two notions of convergence that we will discuss briefly here: practical and theoretical.

In practice, we stop Algorithm 1 when the error E changes less than 10^{-8} from one iteration to the next. To see when this happens, we take another look at the optimization over the states $\mathbf{X}$ in (8) at iteration n. This objective function has two terms. The optimal solution of the left term is the predicted states $\hat{\mathbf{X}}^{(n)}$. The optimal solution of the right term is $\mathbf{X}^{*(n-1)}$. When we optimize this objective function to find $\mathbf{X}^{*(n)}$, there are three cases: 1) $\mathbf{X}^{*(n)} = \hat{\mathbf{X}}^{(n)}$, 2) $\mathbf{X}^{*(n)} = \mathbf{X}^{*(n-1)}$, and 3) $\mathbf{X}^{*(n)}$ is neither $\hat{\mathbf{X}}^{(n)}$ nor $\mathbf{X}^{*(n-1)}$. Our algorithm stops when we are in case 1 or 2 since further optimization over $\boldsymbol{\theta}$ and $\mathbf{X}$ does not change anything. In case 3, the algorithm continues, leading to further optimization steps to decrease error.

From a theoretical standpoint, our treatment will be elementary and serve to motivate future work. Let us first note that (7) and (8) imply, in turn:

$$E(\mathbf{X}^{*(n-1)}, \boldsymbol{\theta}^{*(n)}) \leq E(\mathbf{X}^{*(n-1)}, \boldsymbol{\theta}^{*(n-1)})$$
$$E(\mathbf{X}^{*(n)}, \boldsymbol{\theta}^{*(n)}) + \lambda \|\mathbf{X}^{*(n)} - \mathbf{X}^{*(n-1)}\|^2 \leq E(\mathbf{X}^{*(n-1)}, \boldsymbol{\theta}^{*(n)})$$

Putting these together, we obtain

$$E(\mathbf{X}^{*(n)}, \boldsymbol{\theta}^{*(n)}) \leq E(\mathbf{X}^{*(n-1)}, \boldsymbol{\theta}^{*(n-1)}). \tag{14}$$

The function E, bounded below by 0, is non-increasing along the trajectory $\{(\mathbf{X}^{*(n)}, \boldsymbol{\theta}^{*(n)})\}_{n \geq 1}$. Hence the sequence $\{E(\mathbf{X}^{*(n)}, \boldsymbol{\theta}^{*(n)})\}_{n \geq 1}$ must converge to some limit $E^* \geq 0$. Note, however, that E is not a metric on $\mathbb{R}^{d \times T + p}$. Without further assumptions, there is not much more we can say.

Let us assume that the sequence $(\mathbf{X}^{*(n)}, \boldsymbol{\theta}^{*(n)})$ itself has a limit $(\mathbf{X}^*, \boldsymbol{\theta}^*)$ in $\mathbb{R}^{d\times T+p}$. Let $F_n(\mathbf{X}, \boldsymbol{\theta}) = E(\mathbf{X}, \boldsymbol{\theta}) + \lambda\|\mathbf{X} - \mathbf{X}^{*(n-1)}\|^2$; let F_* be the same function with $\mathbf{X}^{*(n-1)}$ replaced by $\mathbf{X}^*$. Then we can reframe (7) and (8) as

$$\boldsymbol{\theta}^{*(n)} = \operatorname*{argmin}_{\boldsymbol{\theta}} F_n(\mathbf{X}^{*(n-1)}, \boldsymbol{\theta})$$
$$\mathbf{X}^{*(n)} = \operatorname*{argmin}_{\mathbf{X}} F_n(\mathbf{X}, \boldsymbol{\theta}^{*(n)})$$

Taking the limit as $n \to \infty$, we obtain

$$\boldsymbol{\theta}^* = \operatorname*{argmin}_{\boldsymbol{\theta}} F^*(\mathbf{X}^*, \boldsymbol{\theta}) = \operatorname*{argmin}_{\boldsymbol{\theta}} E(\mathbf{X}^*, \boldsymbol{\theta}) \tag{15a}$$
$$\mathbf{X}^* = \operatorname*{argmin}_{\mathbf{X}} F^*(\mathbf{X}, \boldsymbol{\theta}^*) = \operatorname*{argmin}_{\mathbf{X}} \{E(\mathbf{X}, \boldsymbol{\theta}^*) + \lambda\|\mathbf{X} - \mathbf{X}^*\|^2\}. \tag{15b}$$

From these equations, we can derive

$$\nabla_{\boldsymbol{\theta}} E(\mathbf{X}^*, \boldsymbol{\theta}^*) = 0$$
$$\nabla_{\mathbf{X}} E(\mathbf{X}^*, \boldsymbol{\theta}^*) = 0$$

Let $F^*_{\boldsymbol{\theta}\boldsymbol{\theta}}$ denote the Hessian of $F^*(\mathbf{X}^*, \boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$. Similarly, let $F^*_{\mathbf{X}\mathbf{X}}$ denote the Hessian of $F^*(\mathbf{X}, \boldsymbol{\theta}^*)$ with respect to $\mathbf{X}$. By (15a) and (15b), both $F^*_{\boldsymbol{\theta}\boldsymbol{\theta}}$ and $F^*_{\mathbf{X}\mathbf{X}}$ are positive semidefinite at $(\mathbf{X}^*, \boldsymbol{\theta}^*)$. Note that

$$F^*_{\mathbf{X}\mathbf{X}} = E_{\mathbf{X}\mathbf{X}} + 2\lambda I$$

at any point of the form $(\mathbf{X}, \boldsymbol{\theta}^*)$, including the limit point $(\mathbf{X}^*, \boldsymbol{\theta}^*)$. For λ sufficiently large, we see that $F^*_{\mathbf{X}\mathbf{X}}$ will be positive definite, implying strict convexity of $F^*(\mathbf{X}, \boldsymbol{\theta}^*)$ in a neighborhood of $\mathbf{X}^*$.

Using the above framework, together with ideas from the alternating minimization literature—e.g., [5, 6, 15, 17]—we believe we will be able to establish conditions for convergence of our algorithm's iterates, $(\mathbf{X}^{*(n)}, \boldsymbol{\theta}^{*(n)})$, to a local minimizer of E.

Advantages of our approach. Our method is robust with respect to its only hyperparameter λ. We will show in our experimental results later that for a broad range of λ values, our method works well. We fix it to $\lambda = 1$ in our later experiments. As explained before, previous methods have a large number of hyperparameters, which are difficult to set.

Our method can be trained quickly. On a standard laptop, it takes around 20 seconds for our method to learn the parameters and states jointly on ODE problems with 400 states. The spline-based methods take a few minutes and Bayesian methods take a few hours to converge on the same problem.

Because our method, unlike Bayesian methods, does not make assumptions about the type of the noise or distribution of the states, it performs well under different noise and state distributions. In particular, as the magnitude of noise in the observations increases, our method clearly outperforms the extended Kalman filter.

As our experiments confirm, both spline-based and Bayesian approaches are very sensitive to the initialization of the ODE parameters. If we initialize them far away from the true values, they do not converge. Our method is much more robust. This robustness stems from simultaneously learning states and parameters. Even if the estimated parameters are far from the true parameters at some iteration, they can improve later, as the estimated states converge to the clean states.

4 Experiments

In this section, we first introduce three benchmark datasets. Then we show how our method works under different kinds of noise and initializations on these datasets. We finally show that our method performs well compared to other state-of-the-art methods. In the following, we create the clean states using a Runge-Kutta method of order 5. In all of our experiments, unless otherwise stated, we use the three-step Adams-Bashforth method to discretize the ODE.

Lotka–Volterra model. This model is used to study the interaction between predator (variable x_0) and prey (variable x_1) in biology [25]. The model contains two nonlinear equations as follows:

$$\frac{dx_0}{dt} = \theta_0 x_0 - \theta_1 x_0 x_1 \qquad \frac{dx_1}{dt} = \theta_2 x_0 x_1 - \theta_3 x_1. \tag{16}$$

The state is two-dimensional and there are four unknown parameters. We use the same settings as in [11]. We set the parameters to $\theta_0 = 2$, $\theta_1 = 1$, $\theta_2 = 4$ and $\theta_3 = 1$. With initial condition $\mathbf{x}_{(1)} = [5, 3]$, we generate clean states in the time range of $[0, 2]$ with a spacing of $\Delta t = 0.1$.

FitzHugh–Nagumo model. This model describes spike generation in squid giant axons [12, 26]. It also has two nonlinear equations:

$$\frac{dx_0}{dt} = \theta_2(x_0 - \frac{(x_0)^3}{3} + x_1) \qquad \frac{dx_1}{dt} = -\frac{1}{\theta_2}(x_0 - \theta_0 + \theta_1 x_1), \tag{17}$$

where x_0 is the voltage across an axon and x_1 is the outward current. In this model, the states are two-dimensional and there are three unknown parameters. We use the same settings as in [31]. We set the parameters as $\theta_0 = 0.5, \theta_1 = 0.2$, and $\theta_3 = 3$. With initial condition $\mathbf{x}_{(1)} = [-1, 1]$, we generate clean states in the time range of $[0, 20]$ with a spacing of $\Delta t = 0.05$.

Rössler attractor. This three-dimensional nonlinear system has a chaotic attractor [32]:

$$\frac{dx_0}{dt} = -x_1 - x_2 \qquad \frac{dx_1}{dt} = x_0 + \theta_0 x_1 \qquad \frac{dx_2}{dt} = \theta_1 + x_2(x_0 - \theta_2). \tag{18}$$

The states are three-dimensional and there are three unknown parameters. We use the same settings as in [31]. We set the parameters as $\theta_0 = 0.2, \theta_1 = 0.2$, and $\theta_3 = 3$. With the initial condition $\mathbf{x}_{(1)} = [1.13, -1.74, 0.02]$, we generate clean states in the time range of $[0, 20]$, with a spacing of $\Delta t = 0.05$.

Lorenz-96 model. The goal of this model is to study weather predictability [24]. The kth differential equation has the following form:

$$\frac{dx_k}{dt} = (x_{k+1} - x_{k-2})(x_{k-1}) - x_k + \theta_0, \qquad k = 0, \ldots, d-1 \tag{19}$$

The model has one parameter θ_0 and d states, where d can be set by the user. This gives us the opportunity to test our method on larger ODEs. Note that to make (19) meaningful, we have $x_{-1} = x_{d-1}$, $x_{-2} = x_{d-2}$, and $x_d = x_0$. As suggested in [24], we set $d = 40$ and $\theta_0 = 8$. The clean states are generated in the time range $[0, 4]$ with a spacing of $\Delta_i = 0.01$. The initial state is generated randomly from a Gaussian distribution with mean 0 and variance 1.

Evaluation metrics. Let $\boldsymbol{\theta}$ and $\mathbf{X}$ denote the true parameters and the clean states, respectively. Let $\boldsymbol{\theta}^*$ and $\hat{\mathbf{X}}$ denote the estimated parameters and the predicted states. We report the Frobenius norm of $\mathbf{X} - \hat{\mathbf{X}}$ as the prediction error. We also consider $|\theta_l - \theta_l^*|$ as the lth parameter error. Recall that predicted states are achieved by considering $\boldsymbol{\theta}^*$ as the parameter and $\mathbf{x}^*_{(t_0)}$ as the initial state, and then repeatedly applying (5) or its multistep analogue (11).

Optimization of our objective function leads to a better estimation. We first focus only on our objective function in (4). At each iteration n of our optimization, we compute the predicted states $\hat{\mathbf{X}}^{(n)}$ and report the prediction error. Fig. 2 shows the results.

In Fig. 2, we consider two kinds of discretization: 1) one-step Euler method, and 2) three-step Adams-Bashforth method. Note that as we increase the order, we expect to see more accurate results.

We added Gaussian noise with mean 0 and variance σ^2 to each of the clean states, where $\sigma^2 = 0.5$ in the first column and $\sigma^2 = 1$ in the second column. Fig. 2 shows that at the first iteration the error is significant in all models. The error is $\sim 1\,000$ for FitzHugh–Nagumo and Rössler, and $\sim 1.5 \times 10^5$ for Lorenz-96 model.

After several iterations of our algorithm, the error decreases significantly, no matter what kind of discretization we use. But, as expected, three-step Adams-Bashforth performs better than Euler in general: it converges faster and achieves a smaller error at the end. This is more clear for the Lorenz-96 model, where the error becomes almost zero when we use the three-step Adams-Bashforth, while the error becomes around 10^4 for Euler.

The last point about Fig. 2 is that, as expected, sometimes the prediction error increases; the error does not decrease monotonically. This mainly happens at the first few iterations. The main reason for this behavior is that our objective function in (4) is different from the prediction error. We cannot directly optimize the prediction error because we do not have access to the clean states. Still, the fact that our algorithm eventually brings the prediction error close to zero suggests that minimizing the objective in (4) has the same effect as minimizing the prediction error.

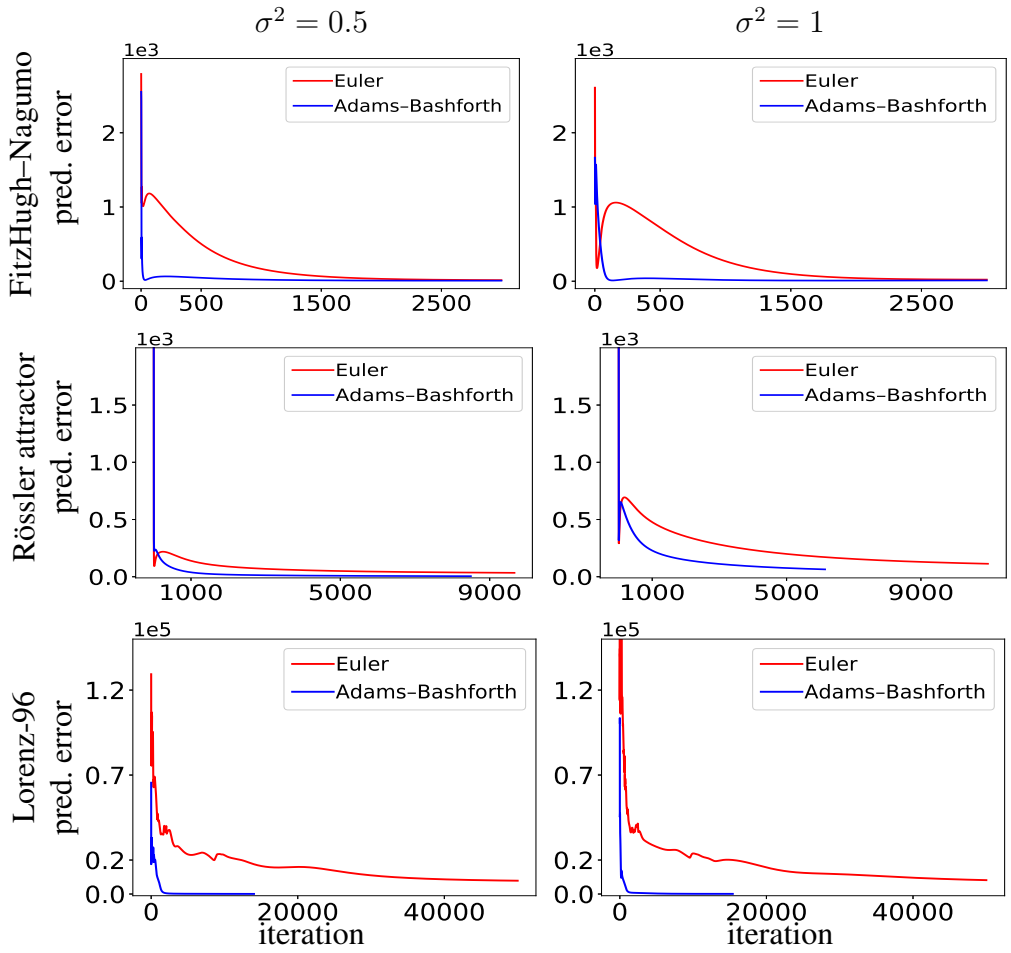

Figure 2: Prediction error at different iterations of our algorithm. Noisy observations are achieved by adding Gaussian noise with variance σ^2 to the clean observations. We consider the FitzHugh–Nagumo (first row), Rössler (second row), and Lorenz-96 (third row) models. Our learning strategy decreases the error in all cases.

Robustness to the hyperparameter λ. The only hyperparameter in our algorithm is λ. In Fig. 3, we set λ in turn to a set of values from 0 to 20, run our algorithm, and report the results after convergence. In both models, we generate observations by adding Gaussian noise with variance 0.5 to the clean states. Because of randomness included in creating noisy observations, we create 10 sets of observations, run our algorithm once for each of them, and report the mean in Fig. 3. We also show the standard deviation in prediction errors, but not in parameter values (to avoid clutter).

In Fig. 3 we report the prediction error and the estimated parameters for each value of λ. The true values for the FitzHugh–Nagumo are $\theta_0 = .5, \theta_1 = .3$, and $\theta_2 = 3$. For the Lotka–Volterra model, the true values are $\theta_0 = 2, \theta_1 = 1, \theta_2 = 4$, and $\theta_3 = 1$.

We see in Fig. 3 that for $\lambda > 0$, our method correctly finds the parameters and brings the error close to zero. Also, in the range of $\lambda = 1$ to 20, the errors and the estimated parameters remain almost the same. We actually increased λ to 1 000 and found that it works like the previous values of $\lambda > 0$. The only disadvantage of increasing λ to a large value is that training time increases—as explained above, increasing λ is analogous to decreasing the step size in a gradient descent method. Large λ implies that states can change very little from one iteration to another, forcing the algorithm to run longer for convergence. As explained in detail above, when $\lambda = 0$, the algorithm stops after a single iteration, with the predicted states far from the clean states.

Different types and amounts of noise in the observations. Our method does not assume anything about the type of noise. In reality, the noise could be from any distribution. In Fig. 4, we investigate the effect of the type of noise on the outcome of our algorithm. The red (blue) curves correspond to

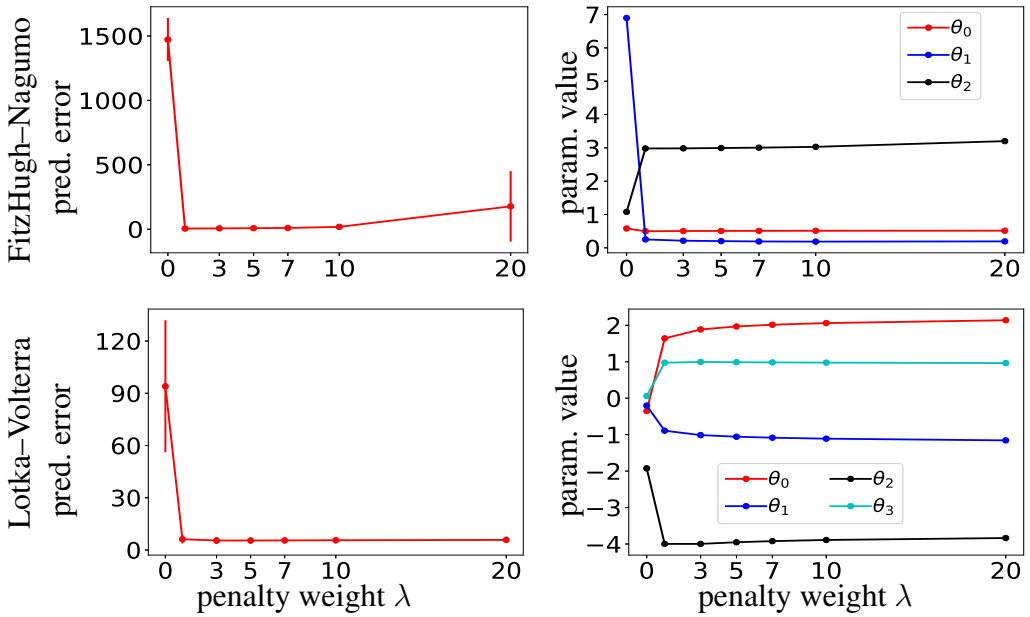

Figure 3: Robustness to the hyperparameter λ in FitzHugh–Nagumo (first row) and Lotka–Volterra (second row) models. The true parameters are $\theta_0 = .5, \theta_1 = .3$, and $\theta_2 = 3$ in the FitzHugh–Nagumo and $\theta_0 = 2, \theta_1 = 1, \theta_2 = 4$, and $\theta_3 = 1$ in the Lotka–Volterra. For each λ, we report the mean error and parameter value in 10 experiments.

the case when we add Gaussian (Laplacian) noise to the observations. We set the mean to 0, change the variance of the noise, and report the prediction and parameter errors. Note that for each noise variance, we repeat the experiment 10 times and report the mean and standard deviation of the error.

In general, increasing the noise variance increases the error. We can see this in almost all plots. In both models, the error does not change much by changing the variance from 0 to 0.5. We can also see that the method performs almost as well for observations corrupted by Laplacian noise as in the Gaussian noise case. Note that the Laplacian noise has a heavier-than-Gaussian tail.

Comparison with other methods (robustness to the initialization). As the first experiment, we compare our method with three other methods, each of them from a different category. Among the spline-based methods, we use the online MATLAB code corresponding to [31], denoted by "splines" in our experiments. Among the Bayesian approaches, we use the online R code corresponding to [11], denoted "Bayes" in our experiments. We also implement a method that uses the iterative least square approach, denoted "lsq" in our experiments. This method considers the parameters and the initial state as the unknown variables. To implement lsq, we use the Python LMFIT package [27]. We use the FitzHugh–Nagumo and Rössler models, creating noisy observations by adding Gaussian noise with mean 0 and variance 0.5 to the clean states.

All methods including ours need an initial guess for the unknown parameters. We add Gaussian noise with mean 0 and variance σ_θ^2 to the true parameter and use the result to initialize the methods. Fig. 5 shows the results, where we change the variance from $\sigma_\theta^2 = 1$ to 20. Since there is randomness in both initialization and observation, we repeat the experiment 10 times. Note that the comparisons are fair, with the same observations and initializations used across all methods.

In Fig. 5, each of the bars corresponds to the prediction or parameter error for one of the methods in one of the experiments. Hence there are 10 error bars for each of the methods in each plot. We set $\lambda = 1$ in our method for all the experiments. For the other methods, we chose the best hyperparameters we could determine after careful experimentation.

The first point regarding Fig. 5 is that our method is robust with respect to the initialization, while the other methods are not. The total number of experiments per method is 80 (on the two models). The prediction error of our method exceeds 100 in 4 experiments. The prediction error of splines (the second best method after ours) exceeds 100 in 39 experiments (nearly half the experiments). For the other methods, the error only increases.

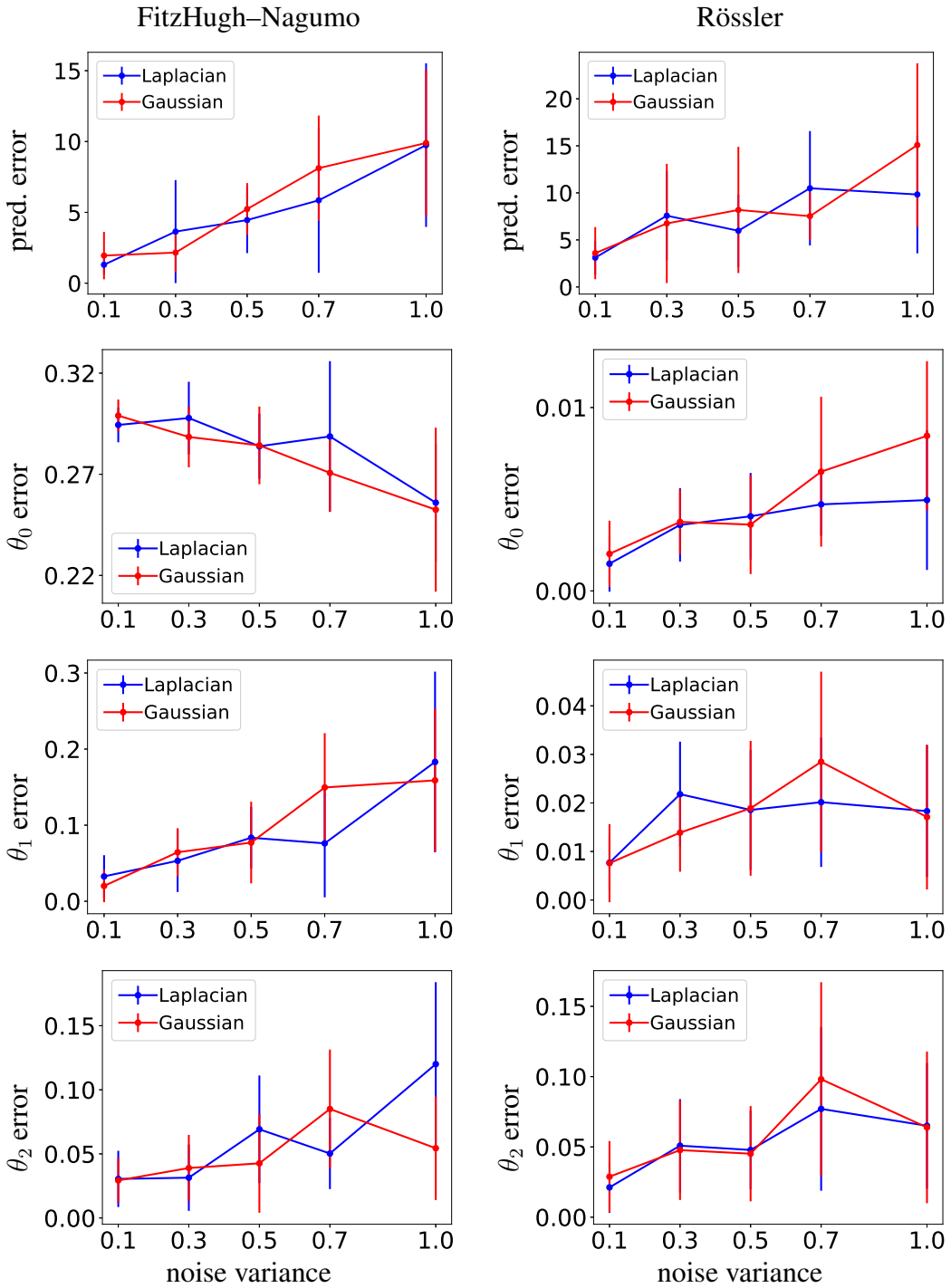

Figure 4: We change the amount and type of noise in the observations, and report the prediction and parameter errors on the FitzHugh–Nagumo (first column) and Rössler (second column) models.

We can see in Fig. 5 that almost all the methods work well when the initialization is close to the true parameters (small noise). But, in reality, we do not know what the real parameters are. So it is reasonable to say that the last column of Fig. 5 (initialization with the largest noise) determines which method performs better in real-world applications. As we can see, our method outperforms the other methods in both prediction and parameter error.

In our second experiment, we compare our method with the mean-field method (also a Bayesian method) [14] on the Lotka–Volterra model. The mean-field method is only applicable to the differ-

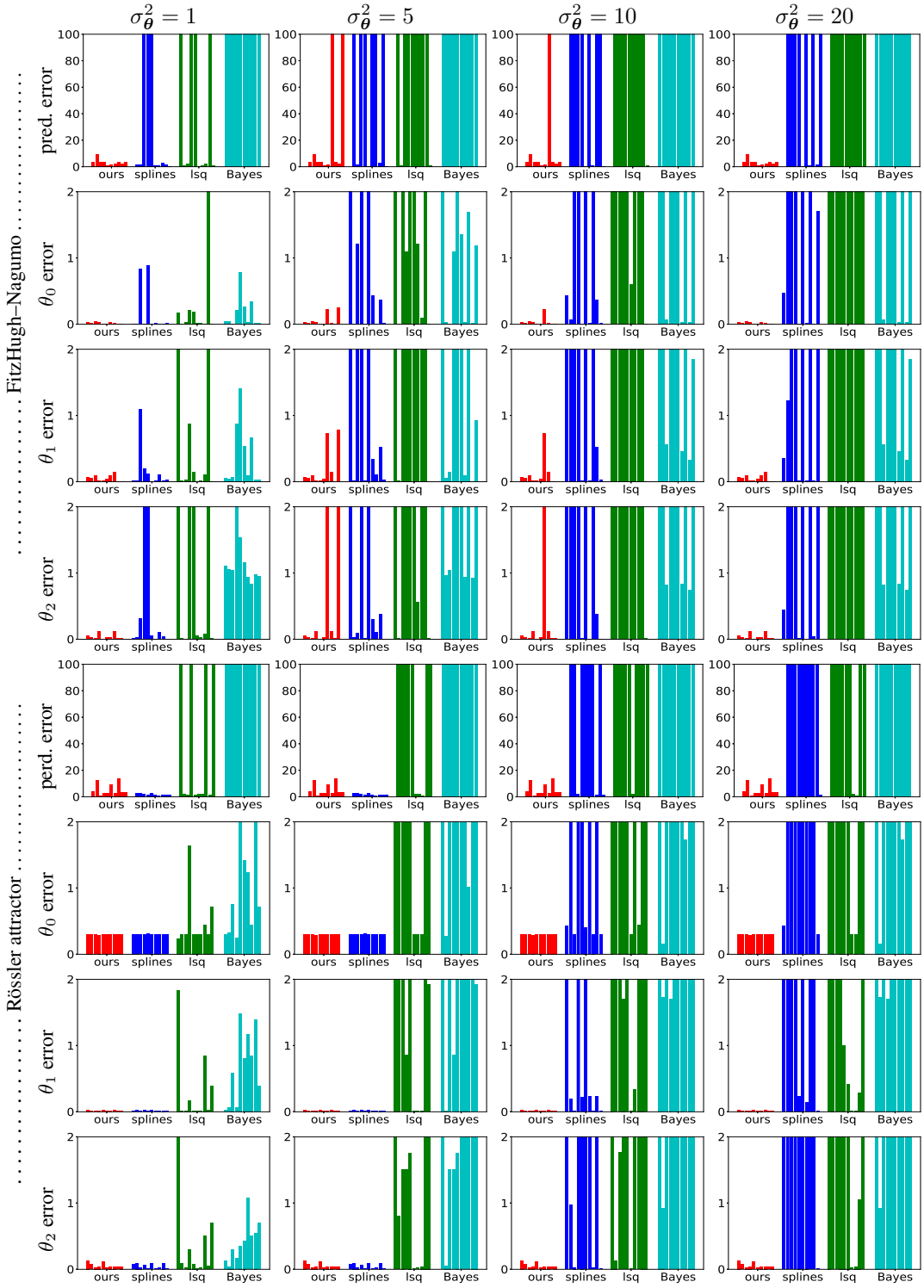

Figure 5: Comparison with other methods. We create initializations by adding Gaussian noise of variance $\sigma^2_{\boldsymbol{\theta}}$ to the true parameters. We create 10 sets of observations and initializations per each $\sigma^2_{\boldsymbol{\theta}}$ and report the errors. Each error bar corresponds to the error in one of the experiments.

ential equations with a specific form (see Eq. (10) in [14]). While we cannot apply it to FitzHugh–Nagumo and Rössler models, we can apply it to the Lotka–Volterra model. In Fig. 6, we compare the methods by reporting the prediction and the parameter errors. We create noisy observations by

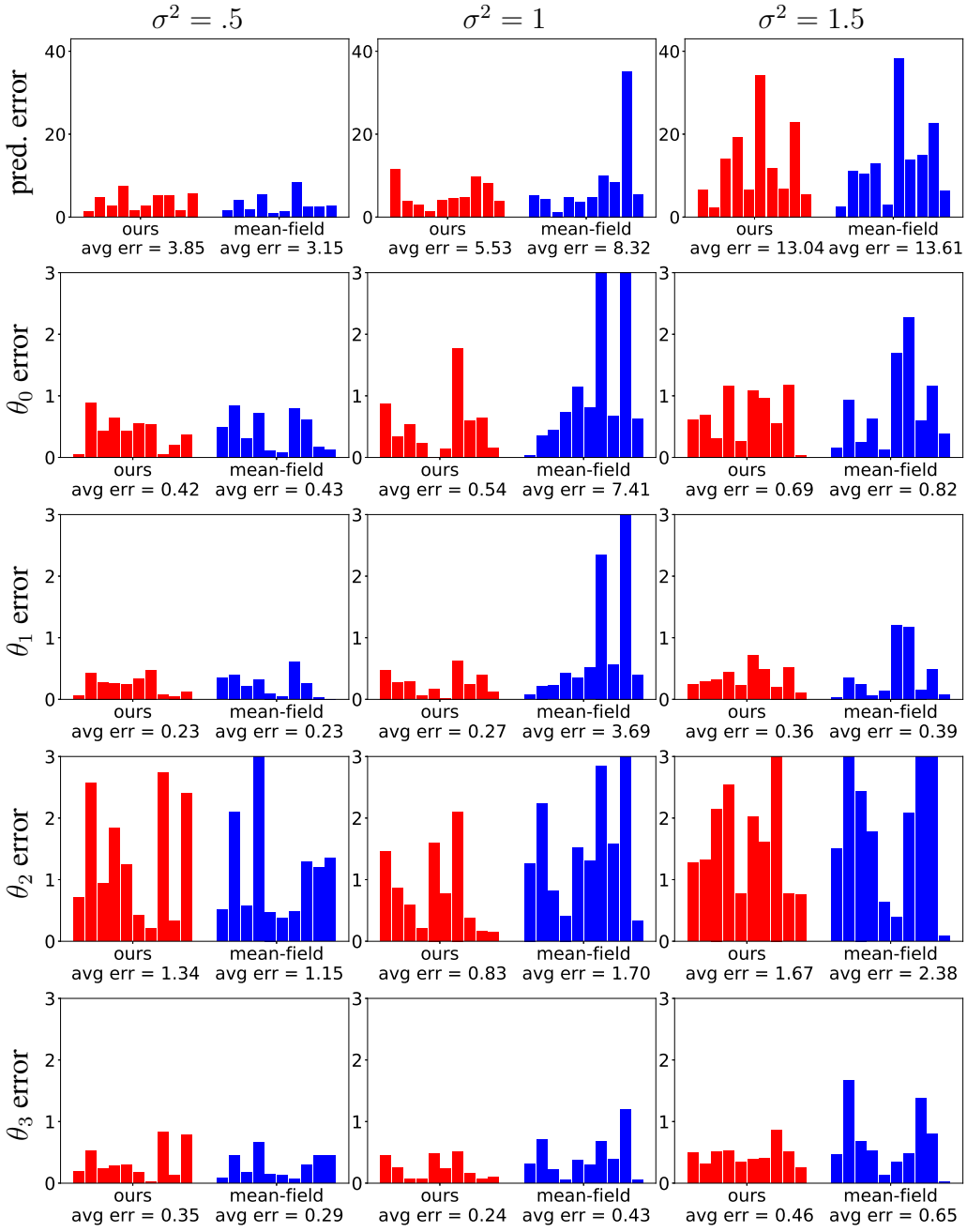

Figure 6: We compare our method with the mean-field method [14] on Lotka–Volterra model. We add Gaussian noise with variance σ^2 to the clean states to create noisy observations. For each value of σ^2, we generate 10 sets of observations and report the prediction and parameter errors. Each error bar corresponds to the error in one of the experiments. We have put the average error of each method for the 10 experiments below each plot.

starting with the clean states and adding Gaussian noise with variance σ^2. We show the results for different variances at different columns of Fig. 6. Similar to our previous experiments, we generate 10 sets of noisy observations. Each of the bars in Fig. 6 corresponds to the error for one of the methods in one of the experiments.

Fig. 6 shows that the average error of our method is less than the mean-field method in almost all cases [14]. As we increase the noise in the observations, the error of both methods increases. But, we can also see that our method is more robust than the mean-field method when it comes to observation

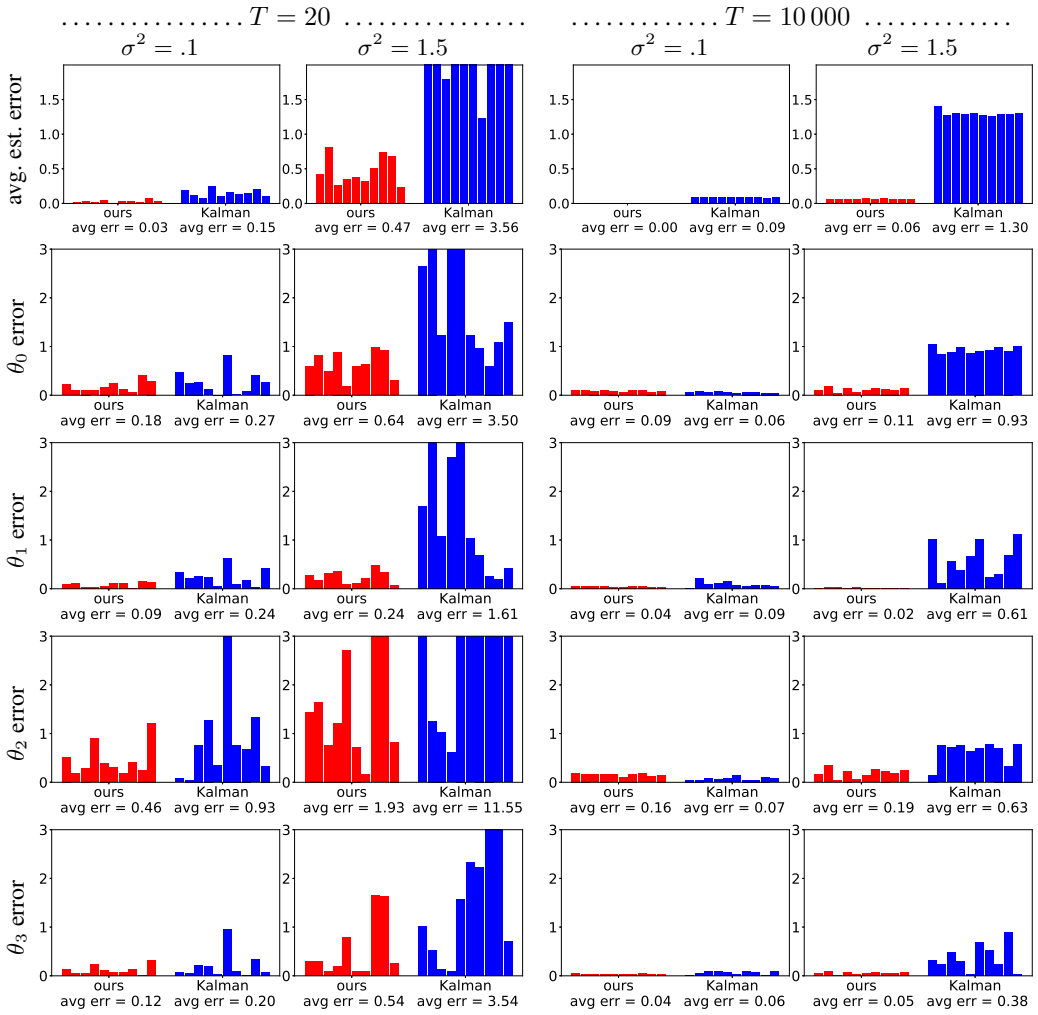

Figure 7: We compare our method with the extended Kalman filter (EKF) on the Lotka–Volterra model. We add Gaussian noise with variance $\sigma^2 = 0.1$ and $\sigma^2 = 1.5$ to the clean states to create noisy observations. We set the number of observation to $T = 20$ (left panel) and $T = 10\,000$ (right panel). For each value of T, we generate 10 sets of observations and report the average estimation and parameter errors. Each error bar corresponds to the error in one of the experiments. We have put the average error of each method for the 10 experiments below each plot.

noise. Consider the case where $\sigma^2 = 1$ (second column). In this case, the average parameter error of the mean-field method for θ_0 and θ_1 becomes around 3 and 8, respectively, but the average error of our method for both parameters remains less than 1.

Comparison with the extended Kalman filter (EKF). We follow [33] in applying the Kalman filter to our problem of estimating the parameters and states. We first need to write an equation that recursively finds the state $\mathbf{x}_{(t_{i+1})}$ in terms of $\mathbf{x}_{(t_i)}$. As suggested in [33], this can be achieved by discretizing the ODE in (1) using the Euler discretization:

$$\mathbf{x}_{(t_{i+1})} = \mathbf{x}_{(t_i)} + \mathbf{f}(\mathbf{x}_{(t_i)}, \boldsymbol{\theta})\Delta_i. \tag{20}$$

Let us define $\boldsymbol{\theta}_{(t_i)}$ as the parameter estimated at time t_i by the Kalman filter. We define a joint state variable $\boldsymbol{\xi}_{(t_i)}$, which merges the states $\mathbf{x}_{(t_i)}$ and the parameters $\boldsymbol{\theta}_{(t_i)}$ as follows:

$$\boldsymbol{\xi}_{(t_i)} = \begin{pmatrix} \mathbf{x}_{(t_i)} \\ \boldsymbol{\theta}_{(t_i)} \end{pmatrix}, \quad \boldsymbol{\xi}_{(t_i)} \in \mathbb{R}^{d+p}. \tag{21}$$

The process model to predict the next state variable can be written as:

$$\boldsymbol{\xi}_{(t_{i+1})} = \begin{pmatrix} \mathbf{x}_{(t_{i+1})} \\ \boldsymbol{\theta}_{(t_{i+1})} \end{pmatrix} = \begin{pmatrix} \mathbf{x}_{(t_i)} + \mathbf{f}(\mathbf{x}_{(t_i)}, \boldsymbol{\theta})\Delta_i \\ \boldsymbol{\theta}_{(t_i)} \end{pmatrix}. \tag{22}$$

We define the observation model as follows:

$$\mathbf{y}_{(t_i)} = \mathbf{H}\boldsymbol{\xi}_{(t_i)}, \quad \mathbf{H} = \begin{pmatrix} \mathbf{I} & \mathbf{0} \end{pmatrix}_{d\times(d+p)}, \tag{23}$$

where $\mathbf{H}$ is a $d \times (d + p)$ matrix, $\mathbf{I}$ is a $d \times d$ identity matrix, and $\mathbf{0}$ is a $d \times p$ matrix where all elements are 0.

In most cases, the function $\mathbf{f}(\cdot)$ is nonlinear, which makes the process model nonlinear. For this reason, we use the extended Kalman filter (EKF), which linearizes the model.

We compare our method with the EKF in Fig. 7 on the Lotka–Volterra model. We compare the methods in different settings by changing the amount of noise and the number of samples. To create noisy observations, we add Gaussian noise with the variances $\sigma^2 = 0.1$ and $\sigma^2 = 1.5$ to the clean states. We set the number of samples to $T = 20$ (time range $[0, 2]$) and $T = 10\,000$ (time range $[0, 1\,000]$).

We use an open-source Python code [21] to implement the EKF. We set the state covariance (noise covariance) to a diagonal matrix with elements equal to $1\,000$ (0.1). We set the process covariance using the function Q_discrete_white_noise() provided in [21], where the variance is set to 1. Note that these parameters must be carefully tuned to obtain reasonable results; changing the state or noise covariance yields significantly worse results.

In Fig. 7 we report the average estimation error instead of the prediction error. Estimation error is defined as the difference between the clean states and the estimated states $\mathbf{X}^*$. We report the estimation error because the prediction error of the EKF goes to infinity. To see why this happens, note that to obtain reasonable predictions we need good estimations of the parameters and the initial state. Since the EKF is an online method, it never updates the initial state. Given that the initial state is noisy, no matter how well parameters are estimated, the prediction error becomes very large. Our method updates the initial state, yielding small prediction error.

As we can see in Fig. 7, the only setting in which the EKF performs comparably to our method is the case of $T = 10\,000$ and $\sigma^2 = 0.1$. In other words, EKF works fine when we have long time series with low noise. In more realistic settings, our method significantly outperforms the EKF. A key difference between the two methods is that the EKF is an online method while ours is a batch method, iterating over the entire data set repeatedly. Consequently, our method updates parameters based on information in all the states, leading to more robust updates than is possible with the EKF, which updates parameters based on a single observation.

We can also see in Fig. 7 that the error of both methods becomes smaller as we increase the number of samples T. This is expected to happen because increasing T is equivalent to giving more information about the model to the methods. The average estimation error of our method becomes almost 0 for large T.

Animation to show how our method works. We consider the FitzHugh–Nagumo model, with settings as explained at the beginning of this section, except that we consider the first 10 seconds instead of 20. We add Gaussian noise with variance 0.5 to the clean states to create the noisy observations. In Fig. 8, we show how our algorithm works in the first 250 iterations. Acrobat Reader is required to play the animation. In this animation, $\mathbf{X}$ denotes clean states (green circles), $\mathbf{X}^*$ denotes estimated states, and $\hat{\mathbf{X}}$ denotes predicted states. Note that initially, $\mathbf{X}^*$ is the same as the noisy observations. Fig. 8 shows the two dimensions separately. At the top of each figure, we show the estimated parameters at each iteration. Note that the true parameters are $\theta_0 = 0.5, \theta_1 = 0.2$, and $\theta_2 = 3$. As explained before, the estimated and predicted states move closer to each other at each iteration. This helps the estimated parameters converge to the true parameters.

5 Conclusion

ODE parameter estimation is an important problem, made difficult due to noisy data and lack of analytical ODE solutions. Previous approaches make several assumptions about the states and the noise, leading to issues regarding the number of hyperparameters and robustness to the noise. In

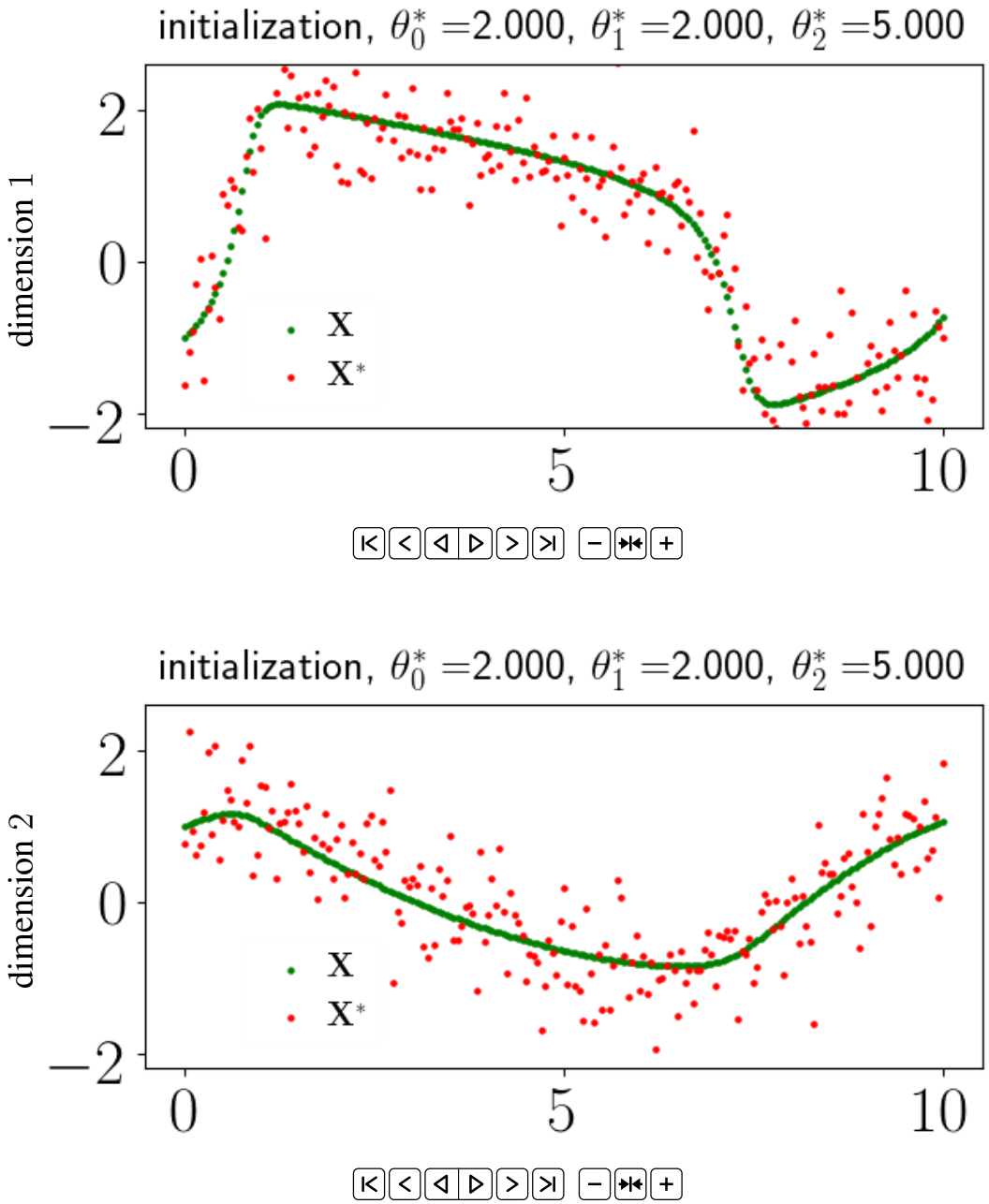

Figure 8: FitzHugh–Nagumo model, where the true parameters are $\theta_0 = 0.5, \theta_1 = 0.2$, and $\theta_2 = 3$. Noisy observations are achieved by adding Gaussian noise to the clean states. In this figure, $\mathbf{X}$ is the clean states (green circles), $\mathbf{X}^*$ is the estimated states, and $\hat{\mathbf{X}}$ is the predicted states. Note that at the initialization, $\mathbf{X}^*$ is the same as the noisy observations. The estimated parameters at each iteration of our algorithm are shown at the top of the figure.

this paper, we have shown that these assumptions are unnecessary. The key in learning the unknown parameters is to learn the clean states. Hence we proposed a direct method that learns states and parameters jointly. Our approach addresses issues of previous works, achieving fast training and robustness to noise, initialization, and hyperparameter tuning.

References

[1] A. Arnold, D. Calvetti, and E. Somersalo. Linear multistep methods, particle filtering and sequential Monte Carlo. *Inverse Problems*, 29(8):085007, 25, 2013.

[2] A. Arnold, D. Calvetti, and E. Somersalo. Parameter estimation for stiff deterministic dynamical systems via ensemble Kalman filter. *Inverse Problems*, 30(10):105008, 30, 2014.

[3] Y. Bard. *Nonlinear Parameter Estimation*. Academic Press, 1973.

[4] M. Benson. Parameter fitting in dynamic models. *Ecological Modelling*, 6(2):97–115, 1979.

[5] C. L. Byrne. Alternating minimization as sequential unconstrained minimization: A survey. *J. Optimization Theory and Applications*, 156(3):554–566, 2013.

[6] C. L. Byrne and J. S. Lee. Alternating minimization, proximal minimization and optimization transfer are equivalent. *arXiv preprint arXiv:1512.03034*, 2015.

[7] B. Calderhead, M. Girolami, and N. D. Lawrence. Accelerating Bayesian inference over nonlinear differential equations with Gaussian processes. In *Advances in Neural Information Processing Systems*, pages 217–224, 2009.

[8] J. Cao and H. Zhao. Estimating dynamic models for gene regulation networks. *Bioinformatics*, 24(14): 1619–1624, 2008.

[9] J. Cao, L. Wang, and J. Xu. Robust estimation for ordinary differential equation models. *Biometrics*, 67(4): 1305–1313, 2011.

[10] I. Dattner and C. A. J. Klaassen. Optimal rate of direct estimators in systems of ordinary differential equations linear in functions of the parameters. *Electronic Journal of Statistics*, 9(2):1939–1973, 2015.

[11] F. Dondelinger, D. Husmeier, S. Rogers, and M. Filippone. ODE parameter inference using adaptive gradient matching with Gaussian processes. In *Artificial Intelligence and Statistics*, pages 216–228, 2013.

[12] R. FitzHugh. Impulses and physiological states in theoretical models of nerve membrane. *Biophysical journal*, 1(6):445–466, 1961.

[13] M. Girolami. Bayesian inference for differential equations. *Theoretical Computer Science*, 408(1):4–16, 2008.

[14] N. S. Gorbach, S. Bauer, and J. M. Buhmann. Scalable variational inference for dynamical systems. In *Advances in Neural Information Processing Systems*, pages 4809–4818, 2017.

[15] L. Grippo and M. Sciandrone. On the convergence of the block nonlinear Gauss-Seidel method under convex constraints. *Oper. Res. Lett.*, 26(3):127–136, 2000.

[16] S. Gugushvili and C. A. Klaassen. $\sqrt{n}$-consistent parameter estimation for systems of ordinary differential equations: bypassing numerical integration via smoothing. *Bernoulli*, 18(3):1061–1098, 2012.

[17] W. Ha and R. F. Barber. Alternating minimization and alternating descent over nonconvex sets. *arXiv preprint arXiv:1709.04451*, 2017.

[18] D. M. Himmelblau, C. R. Jones, and K. B. Bischoff. Determination of rate constants for complex kinetics models. *Industrial & Engineering Chemistry Fundamentals*, 6(4):539–543, 1967.

[19] L. Hosten. A comparative study of short cut procedures for parameter estimation in differential equations. *Computers & Chemical Engineering*, 3(1-4):117–126, 1979.

[20] A. Iserles. *A First Course in the Numerical Analysis of Differential Equations*. Cambridge Texts in Applied Mathematics. Cambridge University Press, Cambridge, second edition, 2009.

[21] R. Labbe. Kalman and Bayesian filters in Python, 2014. URL `https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python`.

[22] Q. Li, Y. Zhou, Y. Liang, and P. K. Varshney. Convergence analysis of proximal gradient with momentum for nonconvex optimization. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 2111–2119, 2017.

[23] H. Liang and H. Wu. Parameter estimation for differential equation models using a framework of measurement error in regression models. *Journal of the American Statistical Association*, 103(484): 1570–1583, 2008.

[24] E. N. Lorenz and K. A. Emanuel. Optimal sites for supplementary weather observations: Simulation with a small model. *Journal of the Atmospheric Sciences*, 55(3):399–414, 1998.

[25] A. J. Lotka. The growth of mixed populations: two species competing for a common food supply. *Journal of the Washington Academy of Sciences*, 22(16/17):461–469, 1932.

[26] J. Nagumo, S. Arimoto, and S. Yoshizawa. An active pulse transmission line simulating nerve axon. *Proceedings of the IRE*, 50(10):2061–2070, 1962.

[27] M. Newville, T. Stensitzki, D. B. Allen, and A. Ingargiola. LMFIT: Non-linear least-square minimization and curve-fitting for Python, 2014.

[28] R. S. Palais and R. A. Palais. *Differential Equations, Mechanics, and Computation*. Number 51 in Student Mathematical Library. American Math. Soc. [u.a.], Providence, RI [u.a.], 2009. Literaturverz. S. 307 - 309.

[29] N. Parikh and S. P. Boyd. Proximal Algorithms. *Foundations and Trends in Optimization*, 1(3):127–239, 2014.

[30] A. Poyton, M. Varziri, K. McAuley, P. McLellan, and J. Ramsay. Parameter estimation in continuous-time dynamic models using principal differential analysis. *Computers & Chemical Engineering*, 30(4):698–708, 2006.

[31] J. O. Ramsay, G. Hooker, D. Campbell, and J. 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.

[32] O. E. Rössler. An equation for continuous chaos. *Physics Letters A*, 57(5):397–398, 1976.

[33] A. Sitz, U. Schwarz, J. Kurths, and H. U. Voss. Estimation of parameters and unobserved components for nonlinear systems from noisy time series. *Phys. Rev. E*, 66(1):016210, 2002. doi: 10.1103/PhysRevE.66.016210.

[34] 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.