Estimating Vector Fields from Noisy Time Series

Harish S. Bhat
Applied Mathematics
University of California, Merced
Merced, CA USA
hbhat@ucmerced.edu

Majerle Reeves
Applied Mathematics
University of California, Merced
Merced, CA USA
mreeves3@ucmerced.edu

Ramin Raziperchikolaei
Rakuten, Inc.
San Mateo, CA USA
ramin.raziperchikola@rakuten.com

Abstract—While there has been a surge of recent interest in learning differential equation models from time series, methods in this area typically cannot cope with highly noisy data. We break this problem into two parts: (i) approximating the unknown vector field (or right-hand side) of the differential equation, and (ii) dealing with noise. To deal with (i), we describe a neural network architecture consisting of tensor products of one-dimensional neural shape functions. For (ii), we propose an alternating minimization scheme that switches between vector field training and filtering steps, together with multiple trajectories of training data. We find that the neural shape function architecture retains the approximation properties of dense neural networks, enables effective computation of vector field error, and allows for graphical interpretability, all for data/systems in any finite dimension d. We also study the combination of either our neural shape function method or existing differential equation learning methods with alternating minimization and multiple trajectories. We find that retrofitting any learning method in this way boosts the method's robustness to noise. While in their raw form the methods struggle with 1% Gaussian noise, after retrofitting, they learn accurate vector fields from data with 10% Gaussian noise.

I. Introduction

We consider the problem of learning a dynamical system from multiple, vector-valued time series. Suppose we have N time series or trajectories, each observed at T discrete times $\{t_i\}_{i=1}^T$. We assume this temporal grid is sufficiently fine to capture the dynamics of the system of interest. Let $\mathbf{y}_i^j \in \mathbb{R}^d$ denote the observation for trajectory j at time t_i. Here $d \geq 1$ is arbitrary. Given this data, we compute estimates of (i) the states $\widehat{\mathbf{x}}_i^j$ and (ii) a vector field $\widehat{\mathbf{f}} : \mathbb{R}^d \to \mathbb{R}^d$ that determines a dynamical system model for the time-evolution of the states.

Suppose the true trajectory $\mathbf{x}^j(t)$ satisfies the nonlinear system of differential equations given by

$$\dot{\mathbf{x}}^j(t) = \frac{d\mathbf{x}^j(t)}{dt} = \mathbf{f}(\mathbf{x}^j(t)). \tag{1}$$

We model the observation $\mathbf{y}_i^j$ as the true state plus noise:

$$\mathbf{y}_i^j = \mathbf{x}^j(t_i) + \epsilon^j(t_i), \quad i = 1, \ldots, T. \tag{2}$$

We seek $\widehat{\mathbf{x}}_i^j$ that approximates $\mathbf{x}^j(t_i)$, and $\widehat{\mathbf{f}}$ that approximates $\mathbf{f}$. While there has been much recent interest in learning differential equations from data, the bulk of the literature focuses on computing $\widehat{\mathbf{f}}$ from relatively clean data; [1] notes that when Gaussian noise of strength greater than 3% is present in observed states, estimation becomes unstable and inaccurate. We demonstrate below that leading methods encounter difficulty even at relatively low noise magnitudes [2], [3]. Our method yields accurate estimates even when the data is corrupted by 10% noise.

Our prior work [4] established that the block coordinate descent proximal filtering method yields more accurate and robust parameter estimates than either iPDA or the extended Kalman filter. A key element of our filtering approach is its proximal step [5]. In contrast, other techniques such *iPDA* and *soft adherence* use a penalty term that anchors the filtered states (at all iterations) to the data $\mathbf{y}$ [6], [7]. Our work shares goals with [8], who are concerned with recovering the functional form of sufficiently ergodic, chaotic dynamical systems from highly corrupted measurements. While our parameterizations of $\mathbf{f}$ differ considerably, both the present paper and [8] develop and apply alternating minimization methods.

The present paper focuses on extending [4] to the setting where the functional form of $\widehat{\mathbf{f}}$ is unknown and must be estimated. In our prior work, we assumed this vector field was known up to a finite-dimensional set of parameters. Hence our prior work contains no mention of neural networks, nor does it contain comparisons against the methods of [3] and [9].

The central finding of this paper is as follows: as the magnitude of noise increases, computing accurate $\widehat{\mathbf{x}}_i^j$ and $\widehat{\mathbf{f}}$ is still possible if one alternates model estimation steps with filtering steps, and if one trains using a larger number N of trajectories. Using tests on simulated data, we show that leading equation discovery methods can be retrofitted with our filtering approach, greatly enhancing the ability of these methods to cope with noisy data. These methods differ in the way they model the estimated vector field $\mathbf{f}$: sparse linear combinations of prescribed functions [2], neural shape functions, or dense, feedforward neural networks [3]. We conduct these tests for both the FitzHugh–Nagumo system and a nonlinear oscillator chain, showing that we can recover both the filtered states and the underlying vector field with high accuracy.

We also apply our method to power grid data recorded by

micro-phasor measurement units (μPMUs). The units record synchronized measurements of voltage and phase angle at the power distribution level [10]. Using data taken on two different days for which the system's status has been labeled, we estimate two vector fields, one for normal operation and one for anomalous operation. To estimate these vector fields, we found it essential to both filter the data and increase the number of trajectories.

II. METHODS

A. Filtering

Let $\widehat{\mathbf{f}}(\mathbf{x}, \boldsymbol{\theta})$ model the true vector field $\mathbf{f}$. For now, we are not concerned with the actual details of this model except to say that the model parameters are given by $\boldsymbol{\theta}$. Consider an explicit Euler time discretization of (1) with time step $\Delta_i = t_{i+1} - t_i$. For trajectory j, we have

$$\mathbf{x}_{i+1}^j - \mathbf{x}_i^j = \mathbf{f}(\mathbf{x}_i^j, \boldsymbol{\theta})\Delta_i, \quad i = 1, ..., T-1 \quad (3)$$

We use Euler purely for simplicity here; in practice the method can accommodate higher-order, explicit time integration schemes. Let $\widehat{\mathbf{X}}$ denote the collection $\{\widehat{\mathbf{x}}_i^j\}_{j=1,i=1}^{N,T}$ of all filtered states over all trajectories. Then define the objective function

$$E(\widehat{\mathbf{X}}, \widehat{\boldsymbol{\theta}}) = \sum_{j=1}^{N} \sum_{i=1}^{T-1} \left\| \frac{\widehat{\mathbf{x}}_{i+1}^j - \widehat{\mathbf{x}}_i^j}{\Delta_i} - \widehat{\mathbf{f}}(\widehat{\mathbf{x}}_i^j, \widehat{\boldsymbol{\theta}}) \right\|^2. \quad (4)$$

Assume that if the true states $\mathbf{X}$ were known, that $\widehat{\mathbf{f}}$ could be trained by minimizing E over $\boldsymbol{\theta}$. We thus refer to minimization over $\boldsymbol{\theta}$ as model training.

Let $\mathbf{Y}$ denote the collection $\{\mathbf{y}_i^j\}_{j=1,i=1}^{N,T}$ of all observations over all trajectories. We propose the following alternating procedure to learn $\widehat{\mathbf{X}}$ and $\widehat{\boldsymbol{\theta}}$. The states are initialized to be the data: $\widehat{\mathbf{X}}^0 = \mathbf{Y}$—superscripts denote the iteration number:

$$\text{train: } \widehat{\boldsymbol{\theta}}^{k+1} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \, E(\widehat{\mathbf{X}}^k, \boldsymbol{\theta}) \quad (5a)$$

$$\text{filter: } \widehat{\mathbf{X}}^{k+1} = \underset{\mathbf{X}}{\operatorname{argmin}} \left\{ E(\mathbf{X}, \widehat{\boldsymbol{\theta}}^{k+1}) + \lambda \|\mathbf{X} - \widehat{\mathbf{X}}^k\|^2 \right\}. \quad (5b)$$

We terminate when the change in $(\widehat{\boldsymbol{\theta}}, \widehat{\mathbf{X}})$ is sufficiently small.

B. Three Parameterizations of $\widehat{\mathbf{f}}$

1) SINDy: We first take $\widehat{\mathbf{f}}$ to be a linear combination of prescribed functions. Given a $1 \times d$ input $\mathbf{x}$ we let $\Xi(\mathbf{x})$ denote a $1 \times s$ dictionary of functions. For instance, for (x_1, x_2) we can take $\Xi(x_1, x_2) = (x_1, x_2, x_1^2, x_2^2, x_1 x_2)$, i.e., all polynomials in the components of x, up to degree two. Using this dictionary, we write

$$\widehat{\mathbf{f}}(\mathbf{x}) = \Xi(\mathbf{x})\boldsymbol{\theta} \quad (6)$$

where $\boldsymbol{\theta}$ is an $s \times d$ matrix of coefficients. Throughout this paper, to extend $\widehat{\mathbf{f}}(\mathbf{x})$ to a function $\widehat{\mathbf{f}}(\mathbf{X})$, we apply $\widehat{\mathbf{f}}$ to each $1 \times 1 \times d$ slice of $\mathbf{X}$. To solve for $\boldsymbol{\theta}$, we apply an iteratively thresholded least-squares regression procedure that promotes a sparse solution $\boldsymbol{\theta}$—for further details, consult [2], [11].
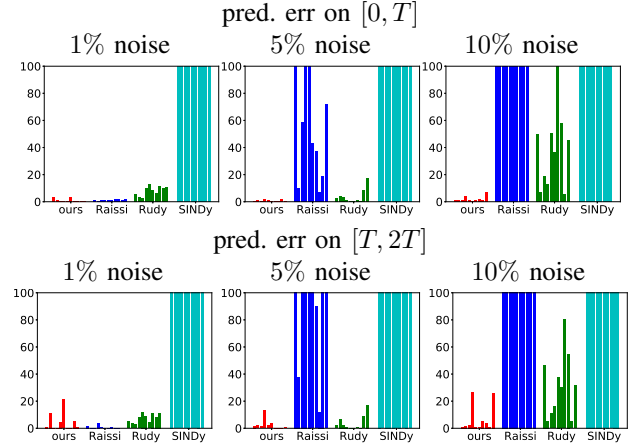

Fig. 1. We compare the neural shape method with [3], [9], and [2] on FitzHugh–Nagumo with different noise levels. For each noise level, we create 10 sets of observations, run the methods on each set separately, and report prediction errors. Each error bar shows the error on one experiment. For all plots, we train on the time interval $[0, T]$. In the upper (resp., lower) set of plots, we report the prediction error on $[0, T]$ (resp., $[T, 2T]$).

2) Neural Shape Functions: Instead of prescribing Ξ and learning only $\widehat{\boldsymbol{\theta}}$, here we learn both Ξ and $\widehat{\boldsymbol{\theta}}$. Let B be the desired number of one-dimensional shape functions; we model these using a dense neural network with one-dimensional input, $D - 1$ hidden layers, and a final layer with B outputs. Let $\mathbf{h}(x) \in \mathbb{R}^B$ denote the output corresponding to scalar input x; then h_j is the j-th one-dimensional shape function.

To create multi-dimensional shape functions, we take tensor products of one-dimensional shape functions. Let a multi-index denote a collection of integers $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_d)$ such that $0 \leq \alpha_k \leq B$. Define $h_0(x) = 1$. Then for a given multi-index $\boldsymbol{\alpha}$, define the multi-dimensional shape function

$$H_{\boldsymbol{\alpha}}(\mathbf{x}) = \prod_{k=1}^{d} h_{\alpha_k}(x_k) \in \mathbb{R}. \quad (7)$$

Let A be a collection of multi-indices and $|A|$ its cardinality. Suppose we compute $H_{\boldsymbol{\alpha}}(\mathbf{x})$ for each $\boldsymbol{\alpha} \in A$; in this way we obtain a $1 \times |A|$ vector $\mathbf{H}(\mathbf{x})$. Here $\mathbf{H}(\mathbf{x})$ plays the same role as $\Xi(\mathbf{x})$ in SINDy above; $|A|$ is analogous to s, the number of shape functions. Now let $\mathbf{B}$ denote an $|A| \times d$ matrix of weights. The neural shape function model is $\widehat{\mathbf{f}}(\mathbf{x}) = \mathbf{H}(\mathbf{x})\mathbf{B}$. Note that there are $(B+1)^d$ possible multi-dimensional shape functions. In practice, we choose A such that $|A| \ll (B+1)^d$, thus constraining $\widehat{\mathbf{f}}$ to be a small linear combination of multi-dimensional shape functions. The set of parameters $\boldsymbol{\theta}$ consists of B together with all weights and biases in the $\mathbf{h}(x)$ network.

Suppose we work in a compact subset $\mathcal{K} \subset \mathbb{R}^d$. Given direct observations of a smooth vector field $\mathbf{f}$ in $\mathcal{K}$, it is possible to achieve arbitrarily small error $\|\mathbf{f} - \widehat{\mathbf{f}}\|$ by choosing the neural shape function hyperparameters sufficiently large. To understand why, note that universal approximation theory guarantees that (even when $D = 1$) the space of our one-dimensional neural shape functions is dense in the space of continuous functions. Hence tensor products of these shape functions are dense in the space of tensor products of continuous functions.
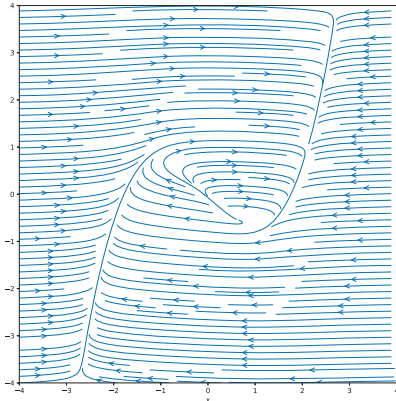
Fig. 2. Ground truth FitzHugh–Nagumo vector field.

Fig. 3. We applied algorithm (5), with neural shape function model for $\widehat{\mathbf{f}}$, to data obtained by simulating the FitzHugh–Nagumo system and then adding 5% noise. *First panel:* the clean states and predicted states over time. Here the predicted states are obtained by numerically integrating the neural vector field $\widehat{\mathbf{f}}$ forward in time starting at the estimated/filtered initial condition. *Second panel:* visualization of the learned one-dimensional shape functions $h_j(x)$.

By Stone-Weierstrass, we see that any smooth vector field can be approximated by linear combinations of tensor products of continuous functions (e.g., univariate polynomials). Putting the previous two facts together, we conclude that linear combinations of our multi-dimensional neural shape functions can be used to approximate smooth vector fields; the accuracy of this approximation can be controlled by the number of units in each h_j network together with $|A|$.

3) Dense Neural Network (DNN): We also consider a dense, feedforward neural network model $\widehat{\mathbf{f}}(\mathbf{x})$ with d inputs and d outputs, the model used by [3].

III. RESULTS

Here we present results from simulated data experiments in which the ground truth vector field $\mathbf{f}$ is known.

A. FitzHugh–Nagumo

In Figure 1, we compare neural shape functions plus filtering (our method) against three published methods [2], [3], [9] on simulated data from the FitzHugh–Nagumo system:

$$\frac{dx_0}{dt} = c\left(x_0 - \frac{x_0^3}{3} + x_1\right), \qquad \frac{dx_1}{dt} = -\frac{1}{c}(x_0 - a + bx_1).$$

To generate simulated data, we set the system parameters $a = 0.5$, $b = 0.2$, $c = 3$ and initial condition $\mathbf{x}_1 = [-1, 1]$. We then numerically solve the system on the interval $[0, T]$ with $T = 20$, recording data at a spacing of $\Delta t = 0.05$. To these clean trajectories we add mean-zero Gaussian noise at strengths of 1%, 5%, and 10%. We train using data on $[0, T]$; in Figure 1, we plot prediction errors for both the training interval and an extrapolatory test interval $[T, 2T]$.

For this initial comparison, let us define the concept of prediction error. Suppose we are at iteration k and that we have trained the model for e epochs. Define the *predicted states* $\widetilde{\mathbf{X}}_e^k$ to be the numerical solution of (1) that results from using the filtered initial conditions contained in $\widehat{\mathbf{X}}^k$ (for the j-th trajectory, $\widehat{\mathbf{x}}_1^j$) together with the current best vector field $\widehat{\mathbf{f}}(\mathbf{x}, \widehat{\boldsymbol{\theta}}^k)$. We call the distance between $\widetilde{\mathbf{X}}_e^k$ and $\mathbf{X}^k$ the *prediction error*—we claim this is a good metric to detect overfitting. If we find that the current prediction error increases as we increase the number of epochs, we halt the optimization
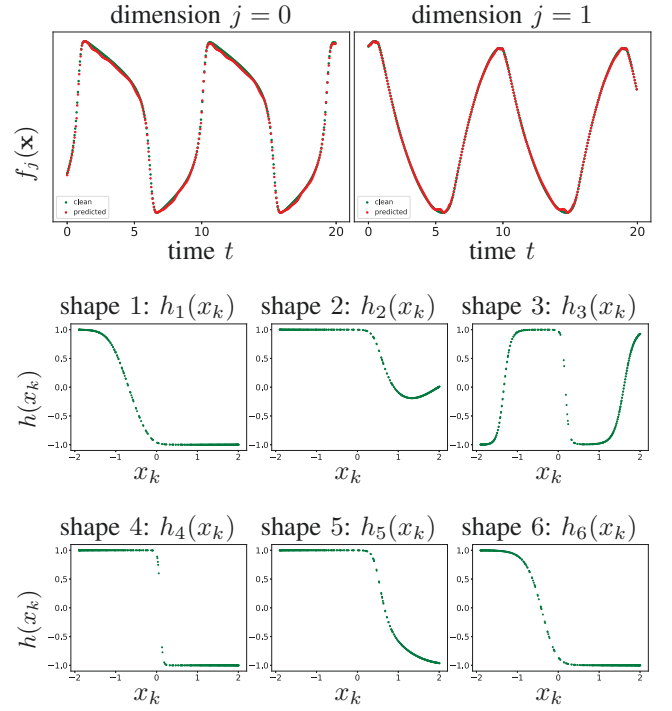
step (5a), set $\widehat{\boldsymbol{\theta}}^k$ to the weights of the network with minimum current prediction error, and proceed to the filtering step (5b).

In Figure 3, we plot the predicted and true states for the neural shape function model of $\widehat{\mathbf{f}}$ learned from data with 5% Gaussian noise. We also plot the learned one-dimensional neural shape functions $h_j(x)$. An advantage of this method is the ability to graphically interpret these shape functions, regardless of the dimension d of the vector field being modeled.

Among the methods studied in Figure 1, only our method and that of [9] attempt to deal with noise during training. Instead of using a proximal term of the form $\|\mathbf{X} - \widehat{\mathbf{X}}^k\|^2$ as in (5b), [9] use a penalty term of the form $\|\mathbf{X} - \mathbf{Y}\|^2$; filtered states are always anchored to the data. While our method outperforms this penalty-based method, both are more robust to noise than approaches that do not incorporate filtering at all. Clearly SINDy [2] has issues with even small amounts of noise, while the dense neural network approach of [3] encounters problems starting at 5% noise.

We have carried out tests similar to that of Figure 1 for nonlinear, chaotic systems such as the Lorenz, Rössler, and double pendulum systems. For such systems, if we perturb the vector field or initial conditions slightly, sensitive dependence implies that over time, trajectories will diverge exponentially. For such systems, the prediction error on $[T, 2T]$ can seem large even if we estimate the vector field and filtered states with a reasonable degree of accuracy. These tests motivate us

Noise	Trajectories	Filter SINDy	Neural Shape Functions	Filter DNN
1 %	1	1.77×10^{-4}	0.8467	0.8157
	25	3.15×10^{-4}	0.2722	0.3274
	400	3.15×10^{-4}	0.2623	0.2034
5 %	1	0.2647	0.7773	0.8667
	25	1.69×10^{-4}	0.4481	0.2580
	400	1.51×10^{-4}	0.2790	0.2186
10%	1	1.4100	0.8168	1.7879
	25	7.30×10^{-3}	0.4107	0.2276
	400	1.13×10^{-3}	0.3028	0.2128

TABLE I

FOR EACH OF THREE ODE LEARNING METHODS, ALL COMBINED WITH FILTERING, WE RECORD THE ERROR IN THE ESTIMATED VECTOR FIELD AS A FUNCTION OF BOTH NOISE STRENGTH AND THE NUMBER OF TRAJECTORIES. FOR EACH METHOD AND EACH NOISE STRENGTH, INCREASING THE NUMBER OF TRAJECTORIES DECREASES THE VECTOR FIELD ERROR. NOTE THAT RETROFITTING SINDY WITH FILTERING DRAMATICALLY IMPROVES ITS ABILITY TO ESTIMATE THE VECTOR FIELD ACCURATELY, AS COMPARED TO THE RESULTS IN FIGURE 1.

Noise	Trajectories	$\|\mathbf{f} - \widehat{\mathbf{f}}\|$	$\|\mathbf{X} - \widehat{\mathbf{X}}\|$	prediction error
1 %	10	5.43×10^{-3}	2.53×10^{-2}	1.86
	100	5.38×10^{-3}	1.70×10^{-2}	0.868
	400	8.23×10^{-3}	2.83×10^{-2}	0.975
5 %	10	8.26×10^{-3}	2.53×10^{-2}	1.86
	100	8.56×10^{-3}	2.80×10^{-2}	1.27
	400	8.20×10^{-3}	2.81×10^{-2}	0.984
10%	10	1.77×10^{-1}	6.67×10^{-2}	3.89
	100	1.78×10^{-2}	4.84×10^{-2}	1.57
	400	1.70×10^{-2}	4.69×10^{-2}	1.17

TABLE II

WE APPLY THE RETROFITTED SINDY ALGORITHM (9) TO NOISY OBSERVATIONS OF THE MASS-SPRING SYSTEM (8). RETROFITTING THE SINDY ALGORITHM ENABLES IT TO HANDLE NOISY DATA. THE RESULTS ALSO SHOW THAT, AT EACH NOISE LEVEL, INCREASING THE NUMBER OF TRAJECTORIES CLEARLY REDUCES PREDICTION ERRORS. FOR EXPLANATIONS OF THE DIFFERENT TYPES OF ERRORS, PLEASE SEE THE MAIN TEXT.

to measure the *vector field error* $\|\widehat{\mathbf{f}} - \mathbf{f}\|_2$, which provides insight into how well the estimated vector field captures the global behavior of $\mathbf{f}$.

We now retrofit both [3] (the dense neural network model for $\widehat{\mathbf{f}}$) and [2] (SINDy) with our filtering procedure. To be clear, both retrofitted methods follow (5); the only difference is in how $\widehat{\mathbf{f}}$ is modeled.

As all the results shown in Figure 1 are for one trajectory, we now move to the setting where we have multiple trajectories. We numerically integrate the FitzHugh–Nagumo system with 400 initial conditions taken from an equispaced 20×20 grid on the square $[-4, 4]$, producing a set of 400 trajectories. We then apply each retrofitted procedure to either 1, 25 (randomly chosen), or all 400 trajectories from the collection. In each case, we obtain an estimated vector field $\widehat{\mathbf{f}}(\mathbf{x})$. We compute the vector field error $\|\widehat{\mathbf{f}} - \mathbf{f}\|_2$ using elementary quadrature on the square $[-4, 4]$. Here $\mathbf{f}(\mathbf{x})$ is the right-hand side of the FitzHugh–Nagumo system; see the ground truth vector field plotted in Figure 2.

The results, shown in Figure 9 and Table I, clearly show the benefit of combining any method with filtering and increasing the number of trajectories used for training, even if those trajectories are all corrupted with 10% noise. The improvement is particularly striking for the SINDy method; in fact, after incorporating SINDy into (5), it outperforms the other methods.

B. Nonlinear Oscillator Network

Consider a ring of $\mathcal{M}$ masses connected by identical springs, each with potential energy $V(x)$ and force $F(x) = -V'(x)$. Here x denotes displacement from equilibrium. Then the equations of motion for the mass-spring system are

$$\ddot{x}_i = F(x_i - x_{i-1}) - F(x_{i+1} - x_i) \tag{8}$$

for $i = 1, \ldots, \mathcal{M}$, with the understanding that $x_0 \equiv x_{\mathcal{M}}$ and $x_1 \equiv x_{\mathcal{M}+1}$. For our tests, we choose the double-well potential $V(x) = -8x^2 + (1/4)x^4$, so that $F(x) = 16x - x^3$.

We focus on the system with $\mathcal{M} = 3$ masses; when we write the system in first-order form, the system has $2\mathcal{M} = 6$ degrees of freedom corresponding to x_i and $\dot{x}_i$ for each $i = 1, \ldots, \mathcal{M}$. To generate data for our tests, we simulate (8) using the 8th-order Dormand-Prince integrator in scipy.integrate, with absolute and relative tolerances tuned to 10^{-14}. We generate 400 trajectories, each with an initial condition sampled uniformly from the hypercube $[-1/2, 1/2]^6$. Each trajectory is saved at 401 equispaced time steps from $t = 0$ to a final time of $t = 4$, i.e., $\Delta t = 0.01$. To these clean trajectories $\mathbf{X}$, we add mean-zero Gaussian noise with strengths of 1%, 5%, and 10% as before, resulting in noisy data $\mathbf{Y}$.

We focus our attention on a retrofitted SINDy method with $\widehat{\mathbf{f}}$ modeled using a dictionary of polynomials. In this method, we use a variant of (5) theoretically analyzed in our previous work [4]. In this method, we incorporate the SINDy model (6), which has the benefit of being linear in the parameters $\boldsymbol{\theta}$. This linearity implies that the objective function (4) is convex in $\boldsymbol{\theta}$. Suppose we split the decision variables $\widehat{\mathbf{X}}$ into two halves, the first half $\widehat{\mathbf{X}}_+$ consisting of time steps $j = 1, \ldots, \lfloor \mathcal{M}/2 \rfloor$, and the second half $\widehat{\mathbf{X}}_-$ consisting of time steps $j = \lfloor \mathcal{M}/2 \rfloor + 1, \ldots, \mathcal{M}$. Then $\widehat{\mathbf{X}} = (\widehat{\mathbf{X}}_+, \widehat{\mathbf{X}}_-)$. This leads us to the following block coordinate descent algorithm:

$$\text{train: } \widehat{\boldsymbol{\theta}}^{k+1} = \underset{\boldsymbol{\theta}}{\arg\min} \, E(\widehat{\mathbf{X}}_+^k, \widehat{\mathbf{X}}_-^k, \boldsymbol{\theta}) \tag{9a}$$

$$\text{filter: } \widehat{\mathbf{X}}_-^{k+1} = \underset{\mathbf{X}_-}{\arg\min} \left\{ \widehat{\mathbf{X}}_+^k, \mathbf{X}_-, \widehat{\boldsymbol{\theta}}^{k+1}) + \lambda \|\mathbf{X}_- - \widehat{\mathbf{X}}_-^k\|^2 \right\} \tag{9b}$$

$$\text{filter: } \widehat{\mathbf{X}}_+^{k+1} = \underset{\mathbf{X}_+}{\arg\min} \left\{ \mathbf{X}_+, \widehat{\mathbf{X}}_-^{k+1}, \widehat{\boldsymbol{\theta}}^{k+1}) + \lambda \|\mathbf{X}_+ - \widehat{\mathbf{X}}_+^k\|^2 \right\}. \tag{9c}$$

Splitting $\widehat{\mathbf{X}}$ and formulating the algorithm in this way gives us block convexity. That is, when we hold two of the three variables in $\{\widehat{\mathbf{X}}_+^k, \widehat{\mathbf{X}}_-^k, \widehat{\boldsymbol{\theta}}\}$ fixed and minimize over the remaining variable, we obtain in each case a convex subproblem [4]. Note that this property would not hold if we were to instead use either of the neural network approaches to model $\widehat{\mathbf{f}}$, as in this case (9a) would be non-convex.
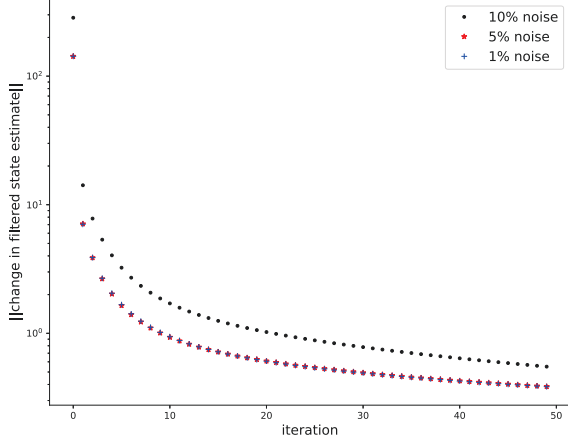
Fig. 4. We monitor and plot the magnitudes of the change in filtered states, $\|\widehat{\mathbf{X}}^{k+1} - \widehat{\mathbf{X}}^k\|$, as a function of iteration number k. We have done this for each of three training sets, each containing the indicated level of noise. In all cases, we find that this quantity decreases rapidly and monotonically.

When we train, we restrict $\mathbf{Y}$ to consist of only the first 301 steps of each trajectory. Starting with noisy data $\widehat{\mathbf{X}}^0 = \mathbf{Y}$, we run algorithm (9) with $\lambda = 10^{-8}$. Note that λ multiplies the squared Frobenius error between two large matrices; hence this value of λ is still consequential. We terminate if the norm difference between $\widehat{\mathbf{X}}^{k+1}$ and $\widehat{\mathbf{X}}^k$ is less than 10^{-4}, or if we have already completed 50 iterations.

To solve (9b) and (9c), we use 1000 steps of gradient descent with a learning rate of 3×10^{-2}. To solve (9a), we use the iteratively thresholded least squares procedure from [2], [11] with a threshold of 0.4. For the dictionary Ξ defined in (6), we include all polynomials up to degree 3 in 6 variables, not including an intercept or constant term, resulting in $s = 63$ columns. We have uploaded our code to https://github.com/hbhat4000/filtersindy/

An interesting property of algorithm (9) is the way in which $\|\widehat{\mathbf{X}}^{k+1} - \widehat{\mathbf{X}}^k\|$ behaves as a function of k. In Figure 4, we have plotted this quantity as a function of iteration k. We have recorded these magnitudes of the change in filtered states while running algorithm (9) on the full 400-trajectory training set, contaminated with either 1%, 5%, or 10% noise. In all cases, we see monotonic convergence. Note that the vertical scale on this plot is logarithmic, further indicating the rapid decrease in $\|\widehat{\mathbf{X}}^{k+1} - \widehat{\mathbf{X}}^k\|$ as a function of k. As a practical matter, this means that we do not need to run (9) for many iterations, and that the results are robust to the number of iterations.

After running algorithm (9) on either 10, 100, or 400 trajectories with either 1%, 5%, or 10% noise, we quantify errors in three ways—see Table II. The vector field error $\|\mathbf{f} - \widehat{\mathbf{f}}\|$ measures the mean absolute error between ground truth and estimated vector fields. The quantity $\|\mathbf{X} - \widehat{\mathbf{X}}\|$ is the mean absolute error between true and filtered states. After training on the first 301 steps of each trajectory, we compute predicted trajectories by numerically integrating the estimated
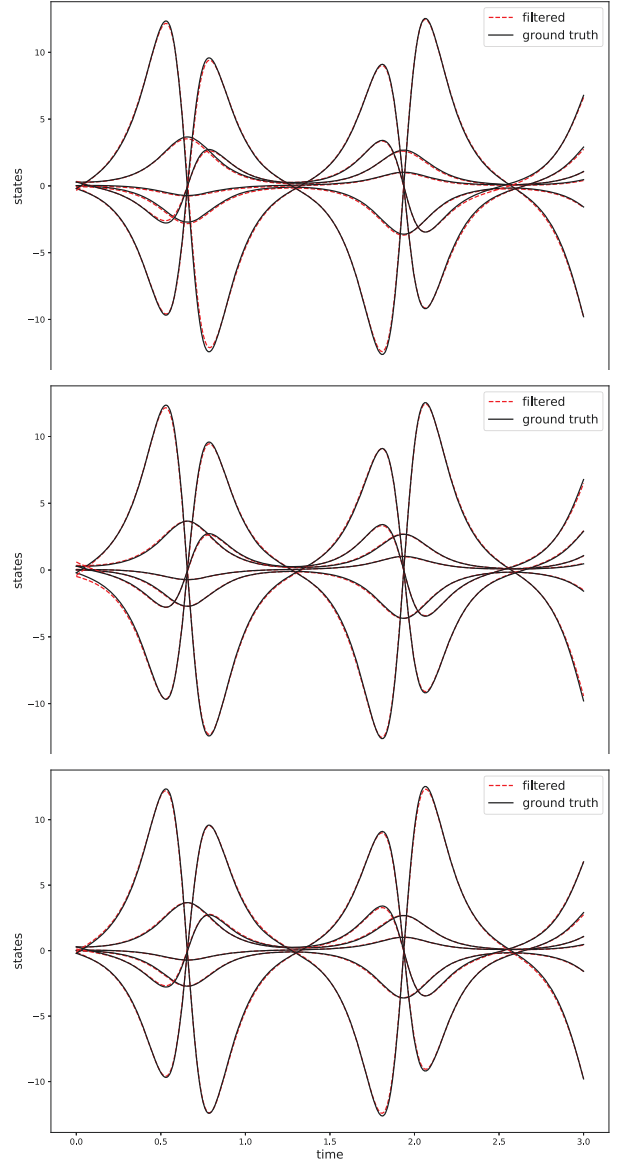

Fig. 5. We present the results of applying (9) to 10 (top), 100 (middle), or 400 (bottom) trajectories worth of data for the nonlinear mass-spring system (8), all contaminated with 10% Gaussian noise. In each of these training sets, we have singled out one common trajectory for the purposes of illustration. In each plot, for this common trajectory, we have plotted the final estimate of the filtered states $\widehat{\mathbf{X}}$ in red, and ground truth trajectories $\mathbf{X}$ in black. A close inspection of the plots reveals errors with 10 trajectories; for the purposes of plotting, these errors disappear with 400 trajectories. For more details, see the main text.

vector field forward in time, starting from the filtered initial conditions. The prediction error is the mean absolute error between these predicted trajectories and the true states $\mathbf{X}$. When we numerically integrate the estimated vector field, we use a standard fourth-order explicit Runge-Kutta method.

The first thing to notice from the results in Table II is that they confirm that combining SINDy with our alternating minimization filtering approach allows SINDy to estimate the

mass-spring system's vector field accurately. Even with 10% noise in the original data, the vector field error when we train on 400 trajectories is 1.70×10^{-2}. To get a sense for what this number means, note that the ground truth vector field is highly sparse—across the entire ground truth coefficient matrix $\boldsymbol{\theta}$, out of $63 \cdot 6$ possible entries, only 33 are nonzero. The retrofitted SINDy algorithm gets this sparsity pattern 100% correct; if we round the coefficients obtained by SINDy, we obtain the ground truth vector field.

Let us now interpret the filtering errors (or second column) of Table II. At the 10% noise level, increasing the number of trajectories appears to decrease the filtering error from 6.67×10^{-2} (with 10 trajectories) only slightly to 4.69×10^{-2}. To visualize this, we present Figure 5. From top to bottom, we present the results of training with 10 (top), 100 (middle), or 400 (bottom) trajectories worth of data, all contaminated with 10% Gaussian noise. For the purposes of illustration, we have singled out one trajectory $\{x_i(t), \dot{x}_i(t)\}_{i=1:3}$ that is a member of all three training sets. Each plot contains 6 black curves, corresponding to $\{x_i(t), \dot{x}_i(t)\}$ for $i = 1, 2, 3$, together with the filtered (or hatted) versions, for a total of 12 curves per plot. When there are only 10 trajectories (top panel), one can discern differences between red and black curves; as we go to 400 trajectories (bottom panel), these errors mostly disappear.

Overall, Figure 5 supports the notion that it is indeed possible to estimate filtered states even when the equations of motion (i.e., the vector field) for the underlying system are themselves unknown and must be simultaneously estimated. However, there is a difference between filtered trajectories and predicted trajectories of the dynamical system. One will notice from Figure 5 that the time axis ends at $t = 3$; we have used 301 steps of training data with $\Delta t = 0.01$, so there is no noisy training data to filter beyond $t = 3$.

To obtain solutions of the estimated dynamical system beyond $t = 3$, we must numerically integrate, as in the predicted trajectories whose errors are quantified in the third column of Table II. When we compute the predicted trajectories in this table, we integrate all the way up to $t = 4$ and then compare against the ground truth (clean) states of the system $\mathbf{X}$. We view predictions on $t \in (3, 4]$ as a true test set, i.e., a test of the estimated vector field's ability to extrapolate beyond the training set.

We again single out the same trajectory common to our training sets with 10, 100, or 400 total trajectories. Starting with the estimated/filtered initial conditions, we numerically integrate the estimated vector field forward in time using a standard fourth-order explicit Runge-Kutta method, from $t = 0$ to $t = 4$. In Figure 6, we compare the results of these numerical integrations (in green) against the ground truth trajectories (in black). When the number of trajectories is small (top), the predicted dynamics are qualitatively wrong. As we train on more trajectories (middle, bottom), the dynamics begin to qualitatively match the true mass-spring system's dynamics. Note that we obtain much better quantitative accuracy for $t \in [0, 3]$, the training interval.

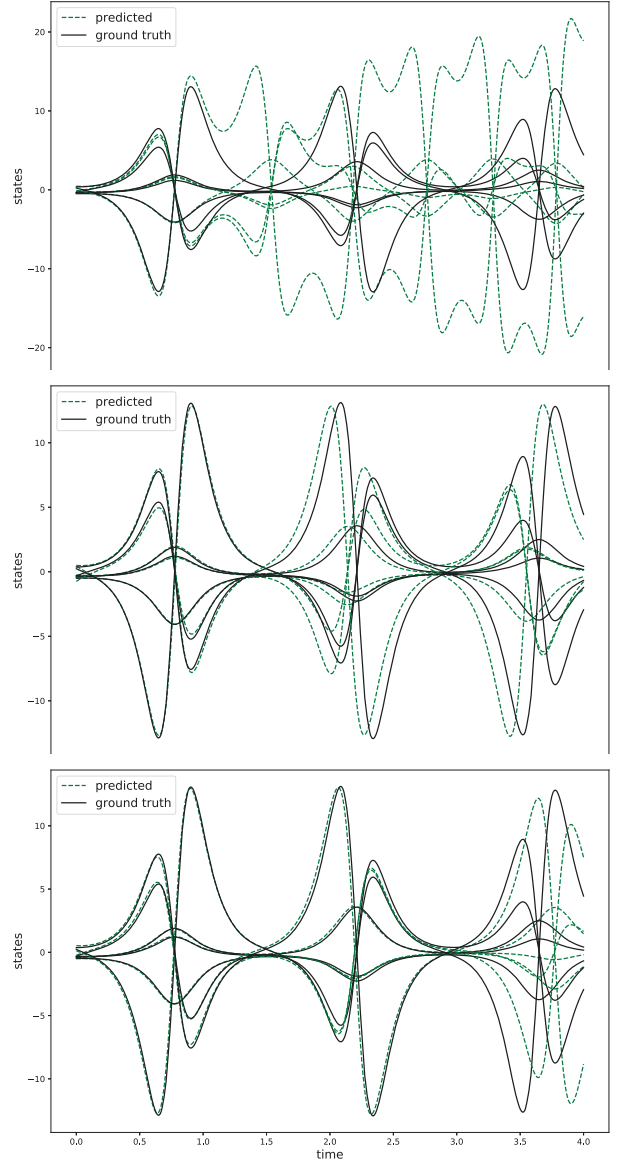For the bottom panel (trained on 400 trajectories), by the

Fig. 6. We present the results of applying (9) to 10 (top), 100 (middle), or 400 (bottom) trajectories worth of data from the nonlinear mass-spring system (8), all contaminated with 10% Gaussian noise. In each of these training sets, we have singled out one common trajectory for the purposes of illustration. In each plot, for this common trajectory, we have plotted predicted trajectories (the results of numerically integrating the estimated vector field forward in time from the estimated initial conditions) in green, and ground truth trajectories $\mathbf{X}$ in black. The estimated vector field and filtered initial conditions are inaccurate when we use only 10 trajectories, leading to qualitatively incorrect dynamics. As we increase the number of trajectories, we recover the qualitatively correct behavior of the mass-spring system. The dynamics for $t \in [0, 3]$ are more accurate because that is the interval covered by the training data; for $t \in (3, 4]$, we are seeing the results of propagating beyond the training interval.

time we reach $t = 3$, the predicted state has drifted noticeably away from the ground truth. We can improve upon this situation by *resetting* the state of the system, at $t = 3$, to equal our estimate $\hat{\mathbf{x}}(3)$ of the true state at that time and then
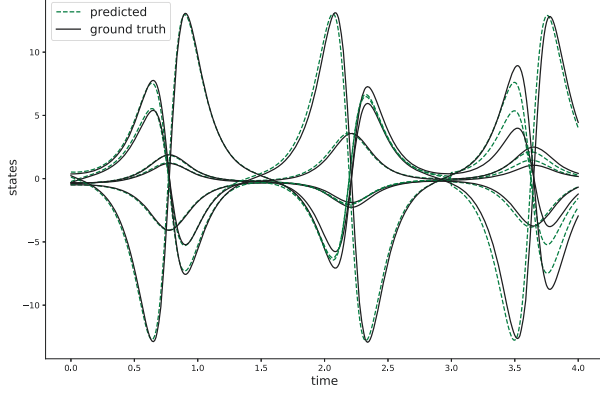
Fig. 7. We continue with results for algorithm (9) applied to noisy data from the nonlinear mass-spring system (8). We redo the numerical integration from the bottom panel of Figure 6, this time resetting the state of the system at $t = 3$ to equal our estimate $\widehat{\mathbf{x}}(3)$ of the true state of the system at that time. This improves the quantitative accuracy of our predicted trajectory on the extrapolation/test interval $t \in (3, 4]$.

Fig. 8. μPMU phase plots for normal (left) and anomalous (right) system behavior. Note that the fixed point is stable (eigenvalues $-4.5014 \times 10^{-4} \pm 1.01i$) on the left and unstable (eigenvalues $4.7166 \times 10^{-5} \pm 1.01i$) on the right.

continuing the numerical integration until $t = 4$. In Figure 7, we see that this procedure leads to improved quantitative agreement between predicted and ground truth trajectories on the test interval $t \in (3, 4]$.

C. MicroPMU Data

We apply the neural shape function plus filtering method to μPMU data taken from a Mountain View substation near Riverside, CA. We use one hour's worth of data from both Aug. 9 and Aug. 1, 2017. We use measurements aggregated at a spacing of $\Delta t = 0.01$. In prior work, these dates have been identified as corresponding to normal (Aug. 9) and anomalous (Aug. 1) system operation. Our idea is to learn two vector fields, one for normal and one for anomalous behavior.

To begin, we prefiltered the data using a wavelet low-pass filter. This was to remove high-frequency noise prevalent in the raw data. Next, we focused our modeling efforts on two phase angle variables, θ_1 and θ_2. This means that we did not use 10 components of the full 12-dimensional signal. To implement the multiple trajectory idea, we reshaped the single hour's trajectory from its original length of $T = 360000$ to $N = 3600$ trajectories each consisting of $T = 100$ points. To implement (5a) we use 100 epochs of the Adam optimizer with a learning rate of 0.002; to implement (5b), we use L-BFGS-B from scipy.optimize with default tolerances and $\lambda = 1000$. The network itself consists of $B = 3$ shape functions with a depth of $D = 2$ and $U = 16$ units per layer. The results are robust to increasing λ, B, D, or U. However, note that training on one prefiltered trajectory of length $T = 360000$, without alternating filtering, does not yield usable models.

In Figure 8, we plot the phase portraits for the learned systems. Interestingly, the main difference between the learned vector fields for the normal (left) and anomalous (right) settings has to do with stability. In short, we see that normal
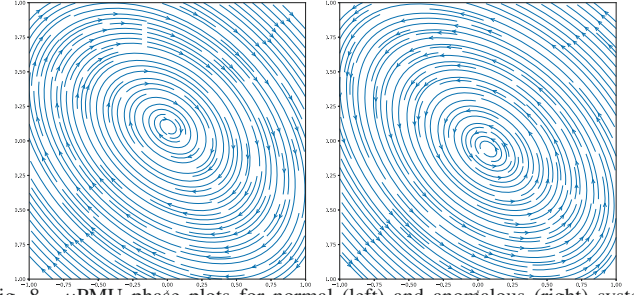
(respectively, anomalous) system operation corresponds to stable (respectively, unstable) oscillations. We have confirmed this by numerically finding the fixed points of the vector fields and checking the eigenvalues of the Jacobians at these fixed points.

IV. DISCUSSION

We find that we can compensate for noisy training data by increasing its volume (the number of trajectories) and by applying simultaneous filtering. Filtering, in the form of the proximal step (5b), reestimates the states $\widehat{\mathbf{X}}$ based on the best model at the current iteration, $\widehat{\mathbf{f}}(\mathbf{X}; \widehat{\boldsymbol{\theta}}^k)$. When $\mathbf{Y}$ is heavily contaminated by noise, several iterations of (5b) allow $\widehat{\mathbf{X}}$ to step away from $\mathbf{Y}$ as needed. While we have focused on the FitzHugh–Nagumo and nonlinear mass-spring systems in the paper, we are currently running additional tests for higher-dimensional physical systems. Preliminary results confirm the same trends we have observed for the FitzHugh–Nagumo system.

SINDy uses polynomial shape functions and the FitzHugh–Nagumo vector field consists of polynomials. Yet unless we include filtering and multiple trajectories, SINDy performs poorly on noisy data. For non-polynomial vector fields, the neural network approaches allow for greater model flexibility. We hypothesize that on non-polynomial systems, neural network-based approaches will prove superior. With the neural shape function approach, one can plot and visualize the one-dimensional shape functions $h_j : \mathbb{R} \to \mathbb{R}$, analogous to visualizing the components of Ξ in SINDy. However, due to its special form and construction, the neural shape function model is more difficult to train than a standard dense neural network.

In future work, we seek to replace these dense neural networks with sparsely connected neural networks that have optimal approximation properties [12]. We will also combine these methods with dimensionality reduction techniques to obtain reduced-order models from the full 12-dimensional μPMU time series. Ultimately, our goal is to uncover global phenomena, such as instabilities, attractors, and periodic orbits, that go beyond single trajectory forecasts.

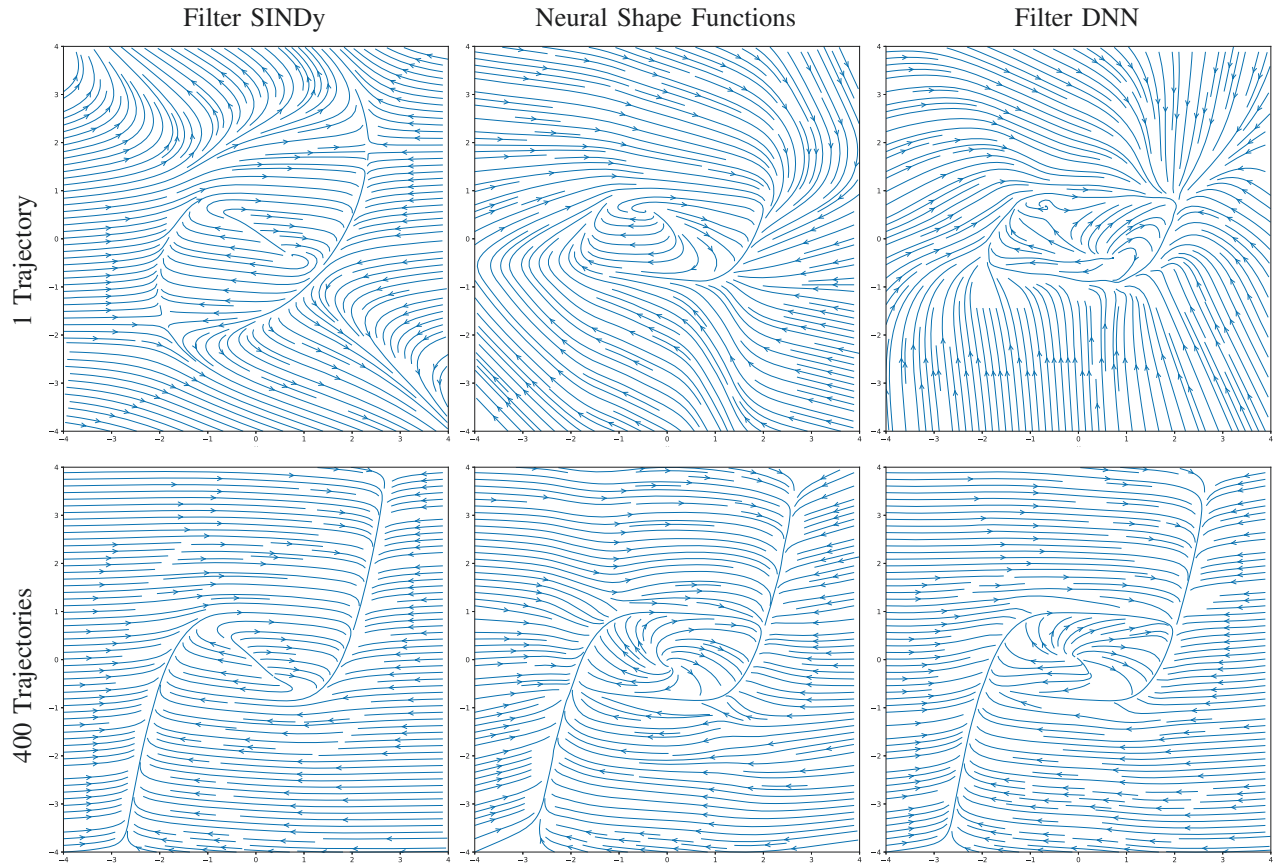|	Filter SINDy	Neural Shape Functions	Filter DNN

Fig. 9. We compare the estimated vector fields for data consisting of FitzHugh–Nagumo trajectories corrupted by 10% Gaussian noise. We combine the three ODE learning methods from Section II-B with the filtering procedure from Section II-A. Note the improvement in the quality of the phase portrait as we increase the number of trajectories used for training.

REFERENCES

[1] H. Schaeffer, "Learning partial differential equations via data discovery and sparse optimization," *Proceedings of the Royal Society A*, vol. 473, no. 2197, pp. 20 160 446, 20, 2017.

[2] S. L. Brunton, J. L. Proctor, and J. N. Kutz, "Discovering governing equations from data by sparse identification of nonlinear dynamical systems," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 113, no. 15, pp. 3932–3937, 2016.

[3] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations," *arXiv:1711.10561*, 2017.

[4] R. Raziperchikolaei and H. S. Bhat, "A block coordinate descent proximal method for simultaneous filtering and parameter estimation," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. Long Beach, California, USA: PMLR, 09–15 Jun 2019, pp. 5380–5388. [Online]. Available: http://proceedings.mlr.press/v97/raziperchikolaei19a.html

[5] N. Parikh and S. Boyd, "Proximal algorithms," *Found. Trends Optim.*, 2014. [Online]. Available: http://dx.doi.org/10.1561/2400000003

[6] J. Ramsay and G. Hooker, *Dynamic Data Analysis: Modeling Data with Differential Equations*, ser. Springer Series in Statistics. Springer New York, 2017.

[7] S. H. Rudy, S. L. Brunton, and J. N. Kutz, "Smoothing and parameter estimation by soft-adherence to governing equations," *Journal of Computational Physics*, vol. 398, p. 108860, 2019.

[8] G. Tran and R. Ward, "Exact Recovery of Chaotic Systems from Highly Corrupted Data," *Multiscale Modeling & Simulation*, vol. 15, no. 3, pp. 1108–1129, Jan. 2017.

[9] S. H. Rudy, J. N. Kutz, and S. L. Brunton, "Deep learning of dynamics and signal-noise decomposition with time-stepping constraints," *Journal of Computational Physics*, vol. 396, pp. 483–506, 2019.

[10] J. Cadena, P. Ray, and E. Stewart, "Fingerprint discovery for transformer health prognostics from micro-phasor measurements," in *ICML 2019, Time Series Workshop, Long Beach, CA*, 2019.

[11] L. Zhang and H. Schaeffer, "On the Convergence of the SINDy algorithm," *Multiscale Modeling & Simulation*, vol. 17, no. 3, pp. 948–972, 2019.

[12] H. Bölcskei, P. Grohs, G. Kutyniok, and P. Petersen, "Optimal approximation with sparsely connected deep neural networks," *SIAM Journal on Mathematics of Data Science*, vol. 1, no. 1, pp. 8–45, 2019.