Symbolic regression via neural networks

Cite as: Chaos 33, 083150 (2023); doi: 10.1063/5.0134464
Submitted: 10 November 2022 · Accepted: 20 July 2023 ·
Published Online: 23 August 2023

N. Boddupalli,[a] T. Matchen, and J. Moehlis

AFFILIATIONS

Department of Mechanical Engineering, University of California, Santa Barbara, Santa Barbara, California 93106, USA

[a]**Author to whom correspondence should be addressed:** nibodh@ucsb.edu

ABSTRACT

Identifying governing equations for a dynamical system is a topic of critical interest across an array of disciplines, from mathematics to engineering to biology. Machine learning—specifically deep learning—techniques have shown their capabilities in approximating dynamics from data, but a shortcoming of traditional deep learning is that there is little insight into the underlying mapping beyond its numerical output for a given input. This limits their utility in analysis beyond simple prediction. Simultaneously, a number of strategies exist which identify models based on a fixed dictionary of basis functions, but most either require some intuition or insight about the system, or are susceptible to overfitting or a lack of parsimony. Here, we present a novel approach that combines the flexibility and accuracy of deep learning approaches with the utility of symbolic solutions: a deep neural network that generates a symbolic expression for the governing equations. We first describe the architecture for our model and then show the accuracy of our algorithm across a range of classical dynamical systems.

The dynamics of quantities of interest are widely modeled as differential equations, often derived from first principles. However, this is not always possible, especially when the underlying mechanisms are unknown or complex. The identification of models from data has seen significant advances with the advent of machine learning. While deep neural networks have enabled sufficient accuracy in forecasting dynamic data with unprecedented versatility, the models they represent lack closed-form expressions that can be conducive to interpretation and analysis. Here, we present an algorithm that identifies parsimonious closed-form ordinary differential equations from noisy data using a novel deep learning architecture and an information criterion.

I. INTRODUCTION

Mathematical models for systems of scientific, technological, and societal interest have been pursued for centuries. Models that capture a system's dynamics are of particular importance because they allow prediction and analysis of how quantities of interest change with time and can often be used to develop control algorithms. For many systems, it is possible to derive mathematical models from first principles, such as Newton's laws, Maxwell's equations, or the Navier–Stokes equations. Such models have had far-reaching success over a broad range of length and timescales and have led to incredible advances in fields ranging from fluid dynamics to solid mechanics to robotics and beyond. Unfortunately, deriving models from first principles is not always feasible, which has led researchers to explore ways to deduce mathematical models from observed data, a process known as system identification.[1] Here, we briefly summarize several notable methods for system identification.[2]

A number of authors have approached system identification by fitting coefficients of a linear combination of basis functions, dating at least back to Crutchfield and McNamara.[3] The set of basis functions typically includes nonlinear terms, for example, terms that would arise in a Taylor series expansion about the origin of the system[3–6] or a broader class of functions.[7] The coefficients of the basis functions are determined through comparison of the original data points with points from computed solutions to the fitted models. Various sparse optimization approaches have been proposed to try to avoid an unwieldy number of terms in the resulting models.[5–8] Given its recent popularity, here, we highlight the Sparse Identification of Nonlinear Dynamics (SINDy) algorithm,[7] in which the right-hand side of the differential equation model is found as the optimal sparse linear combination of user-specified candidate functions, which may be nonlinear. The SINDy algorithm has been successfully applied to a variety of models, including the Lorenz equations and vortex shedding past a cylinder,[7] and has been extended to allow for rational function nonlinearities, which commonly appear in biological networks.[9] It was also combined with deep learning to determine which terms are needed to accurately capture the dynamics seen in data.[10] A limitation of all such approaches is that they require pre-specified basis functions whose linear combination gives the model. We call such methods "fixed dictionary" methods.

Mathematically, fixed dictionary system identification approaches find a model

$$\frac{dx}{dt} = \sum_i a_i \phi_i(x)$$

by making smart choices for the a_i's, where $\{\phi_i(x)\}$ is a pre-specified fixed dictionary of functions.

Also of recent interest is dynamic mode decomposition (DMD), which discovers coherent spatiotemporal modes from measurements of complex systems, and has connections to Koopman operator theory.[11–13] DMD was initially developed in the fluid dynamics community[14] and has since been applied to many other systems.[15] A type of DMD known as extended dynamic mode decomposition (EDMD) is a regression onto locally linear dynamics of a fixed dictionary of candidate functions and can be used to generate a nonlinear closed-form mathematical model for a system based on data.[16] However, EDMD can face "curse of dimensionality" challenges and typically requires a large amount of data to be accurate.

In another class of system identification methods, which we call "generative dictionary" methods, one only pre-specifies a set of primitive operations and functions, and the method creates the dictionary of possible model terms from combinations and compositions of these primitives. Historically, such methods have been referred to as "symbolic regression" because they search a space of mathematical expressions to find a model, as opposed to conventional regression techniques, which optimize parameters for a pre-specified model structure (i.e., a fixed dictionary). Typical symbolic regression approaches to system identification[17–19] learn models from observed data by randomly combining various terms and operations and using genetic programming to "mutate" the candidate solutions according to a fitness-weighted selection mechanism. One such approach[20] was to restrict the space of possible models by searching for invariant sets. While these invariants involved estimating higher order time and space derivatives that grow combinatorially in number with an increase in dimensions, it was also only suitable to model systems that obeyed conservation laws. Unfortunately, genetic programming strategies typically suffer from bloat,[21] excessive terms, and overly complex representations of systems, making identification of the most physically meaningful representation of the data difficult.

We emphasize that there is an important distinction between fixed dictionary and generative dictionary system identification methods: for the former, only terms that have been included in the original fixed dictionary can appear in the model. For the latter, any term that can be generated from combinations and compositions of the primitives can appear. In particular, let $S_k(x)$ be a (possibly k-dependent) subset of all possible functions of x generated by the primitive operations and functions. Mathematically, a generative dictionary system identification method seeks to find a model

$$\frac{dx}{dt} = F(S_K \circ \cdots \circ S_2 \circ S_1) \tag{1}$$

by making smart choices for which function F and subsets S_k to choose. Here, the composition $\circ$ is interpreted in an element-wise manner; for example, $S_2 \circ S_1$ can contain functions $f_2(f_1(x))$ for any $f_1 \in S_1$ and $f_2 \in S_2$. Moreover, each S_k includes the Identity operator; therefore, for example, $S_2 \circ S_1$ can contain any function in S_1. The generative dictionary is given by the set $S_K \circ \cdots \circ S_2 \circ S_1$, and both the generative dictionary and the function F are determined as part of the system identification algorithm.

An alternative approach to system identification is to use deep neural networks (DNNs) to obtain a "black box" model in which the current state of a system is mapped to the future state without knowledge of the inner workings of the transformation.[22–24] For system identification, it is notable that DNNs do not make assumptions about the terms in a model, unlike SINDy or EDMD. This comes at the cost of estimating the network weights iteratively, but fast optimization algorithms benefit from today's computational power that is higher and cheaper than before. On the other hand, generalizability of the obtained weights to unseen data-sets even in the same regime is not guaranteed.

In this paper, we present a novel neural network framework for identifying interpretable models from time series or vector-field data without *a priori* knowledge of the governing dynamics. The models are built up from primitive operations, such as addition, multiplication, exponentiation, and other user-defined functions of state variables that commonly occur in dynamical systems. Drawing inspiration from DNNs, we allow the breadth of the network to determine the number of allowable coefficients in a certain set of functions and the depth to determine compositions among them. Our approach allows the generation of interpretable functions of state variables, including linear combinations, polynomials, and non-polynomial functions, such as fractional and negative powers, logistic functions, and expressions that might arise in chemical kinetics or a conductance-based model for neural activity. The complexity of these generated functions that appear in the model is determined directly by the depth and width of the neural network rather than having to pre-specify them exactly. Our neural network architecture is novel in how the weights of the network determine the form of the terms in the generated dictionary, which allows a wider variety of model terms than other recent system identification approaches that use neural networks.[25,26] Moreover, the number of terms in the generated dictionary remains small enough to handle computationally, but large enough to capture a rich set of possibilities. Available algorithms for optimizing DNNs are used to optimize the coefficients of state variables in these expressions. To obtain parsimonious models, we use a combination of sparsity-inducing regularization and the Akaike Information Criterion (AIC),[27] which selects between candidate models, obtained through perturbations of model parameters within user-specified tolerances, to balance model simplicity and accuracy. We call this SymANNTEx (pronounced as "semantics"), for *Sym*bolic, *A*rtificial *N*eural *N*etwork-*T*rained *Ex*pressions.

A. Preliminaries

We consider dynamical systems of the form

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}), \tag{2}$$

where $\boldsymbol{x} \in \mathbb{R}^n$ and $\boldsymbol{f}: \mathbb{R}^n \to \mathbb{R}^n$. The ith state of $\boldsymbol{x}$ will be denoted as x_i. We consider data points sampled discretely—but not necessarily from a single trajectory—in time and denote the jth data point as

$\boldsymbol{x}^{[j]}$. Given m such data points $\{\boldsymbol{x}^{[j]}\}_{j=1}^{m}$, we train our neural network on data that approximates $\boldsymbol{f}(\boldsymbol{x}^{[j]}) = \dot{\boldsymbol{x}}|_{\boldsymbol{x}=\boldsymbol{x}^{[j]}} \equiv \dot{\boldsymbol{x}}^{[j]}$ at those points, as follows.

Since real data are expected to be noisy, we consider states $x_i^{[j]} + x_i^{rms}\eta_i^{\sigma_1\,[j]}$, where $\eta_i^{\sigma_1\,[j]}$ is independently drawn from the normal distribution $\mathcal{N}(0, \sigma_1^2)$ centered at zero and characterized by standard deviation σ_1, which is scaled by the Root Mean Square (RMS) of the corresponding state

$$x_i^{rms} \equiv \sqrt{\frac{1}{m}\sum_{j=1}^{m}\left(x_i^{[j]}\right)^2},$$

which is the l^2 norm of the corresponding state averaged over the m data points. We write this using the element-wise product $(\boldsymbol{x}^{rms} \odot \boldsymbol{\eta}^{\sigma_1})_i^{[j]} \equiv x_i^{rms}\eta_i^{\sigma_1\,[j]}$ and denote the state data as

$$\left\{\boldsymbol{x}^{[j]} + (\boldsymbol{x}^{rms} \odot \boldsymbol{\eta}^{\sigma_1})^{[j]}\right\}_{j=1}^{m} \equiv \left\{\boldsymbol{x}_{\sigma_1}^{[j]}\right\}_{j=1}^{m} \equiv \boldsymbol{X}. \tag{3}$$

For example, if we consider noise drawn from a distribution of standard deviation $\sigma_1 = 0.01$, we refer to it as 1% noise relative to the corresponding component of $\boldsymbol{x}^{rms}$. Similarly, we add noise to the corresponding derivative data and define the training data $\boldsymbol{Y}$ as follows:

$$\left\{\dot{\boldsymbol{x}}^{[j]} + (\dot{\boldsymbol{x}}^{rms} \odot \boldsymbol{\eta}^{\sigma_2})^{[j]}\right\}_{j=1}^{m} \equiv \left\{\dot{\boldsymbol{x}}_{\sigma_2}^{[j]}\right\}_{j=1}^{m} \equiv \boldsymbol{Y}, \tag{4}$$

where $\eta_i^{\sigma_2\,[j]}$ are each independently drawn from $\mathcal{N}(0, \sigma_2^2)$ and $\dot{\boldsymbol{x}}^{rms}$ is the vector of RMS of derivatives at points considered, which is computed as

$$\dot{x}_i^{rms} \equiv \sqrt{\frac{1}{m}\sum_{j=1}^{m}\left(f_i(\boldsymbol{x}^{[j]})\right)^2}.$$

We note that derivative data could alternatively be obtained from noisy state data using various numerical differentiation techniques.[28] The obtained derivative data may be imprecise, which is what σ_2 is intended to mimic.

Note that we scale the noise added to the states and the derivatives by the RMS of the respective corresponding states and derivatives because our examples span varying length and timescales, and we found that un-scaled additive noise of a certain level can have negligible effect on one example or state, while being detrimental to another. However, we have also verified that our approach successfully identifies models when (sufficiently weak) un-scaled additive noise is used.

Our network's objective is to map input (noisy state) data $\boldsymbol{X}$ to the training (noisy derivative) data $\boldsymbol{Y}$. We call $\boldsymbol{f}(\boldsymbol{x})$ in Eq. (2) the "ground truth model," the model represented by the network $\boldsymbol{x}_{\sigma_1}^{[j]} \mapsto \tilde{\boldsymbol{f}}\left(\boldsymbol{x}_{\sigma_1}^{[j]}\right)$ the "network model," and the model $\boldsymbol{x}_{\sigma_1}^{[j]} \mapsto \tilde{\boldsymbol{f}}_A\left(\boldsymbol{x}_{\sigma_1}^{[j]}\right)$ chosen after perturbing the network model and evaluating AIC scores as the "selected network model." This can also be directly applied to identify maps $\boldsymbol{x}(k+1) = \boldsymbol{F}(\boldsymbol{x}(k))$ with a closed-form expression. Note that even the ground truth model probably gives non-zero errors because the training data $\boldsymbol{Y}$ are not exactly the derivative evaluated at the data points $\boldsymbol{X}$ as the expected values $E(\dot{\boldsymbol{x}}_{\sigma_2}) \neq E(\boldsymbol{f}(\boldsymbol{x}_{\sigma_1}))$ for our examples even with $\sigma_1 = \sigma_2$.

II. NETWORK ARCHITECTURE

Our approach to symbolic regression fuses the adaptability of artificial neural networks with the interpretability of dictionary-based identification methods via a generative dictionary. By design, the generated candidate functions, which describe the dynamics, have global support. The width of the network allows different coefficients, e.g., $\sin(a_1x_1) + \sin(a_2x_1)$, and the depth gives compositions of increasing complexity that have closed-form expressions, e.g., $\sin(x_1^2x_2)$. Specifically, the network initially takes as input the n system states and an additional constant value of 2 to aid with scaling for $n+1$ total states; notationally, we let $x_{n+1} = 2$. If desired, time could be included as well, allowing for $n+2$ initial states. The neural network possesses two principal tiers of organization: "stacks" and "operational layers." Stacks serve as the higher-level organizational structure; they modify the data in series, with the output of each stack serving as an input to subsequent stacks. The network always possesses $K \geq 1$ stacks. An individual stack is in turn composed of a predetermined number of operational layers, which generate new terms for use both in subsequent stacks and the final expression. As shown in Fig. 1, these operational layers function in parallel, sharing a common input from the original data as well as any prior stacks. Each stack possesses $L \geq 1$ instances of each operational layer.

Each operational layer consists of four types of sublayers: (1) linear combinations, (2) polynomial combinations, (3) simple products of variables, and (4) common operators. As noted, if we have multiple stacks, the outputs from previous stacks serve as inputs to subsequent stacks. For example, if $K > 1$, then the kth stack takes as input not only the initial $n+1$ states but also the $4L\left(k-1\right)$ states generated by operations in the previous stacks, which allows for highly complex analytic expressions with minimal training parameters. Each of these processes and operations is outlined below. The total number of expressions we combine to generate the final output is $4LK + n + 1$ if the system is autonomous or $4LK + n + 2$ if the system is non-autonomous. These terms are combined to generate an output via a simple, fully-connected $n \times (4LK + n + 1)$ dense output layer. Weight vectors for trainable sublayers will be referred to as $\boldsymbol{w}$, with w_i denoting the ith component. The architecture of each operational layer is illustrated in Fig. 2.

A. Linear combinations

These sublayers are linear combinations of the pre-existing states (including $x_{n+1} = const.$),

$$y = \sum_{i=1}^{n+1} w_i x_i. \tag{5}$$

B. Polynomial combinations

The output of a polynomial combination sublayer is given by

$$y = \prod_{i=1}^{n+1} |x_i|^{w_i}. \tag{6}$$

The absolute value of each state is used to ensure that each output value y remains in the domain of real numbers; sign considerations are handled separately; for example, suppose we wish to

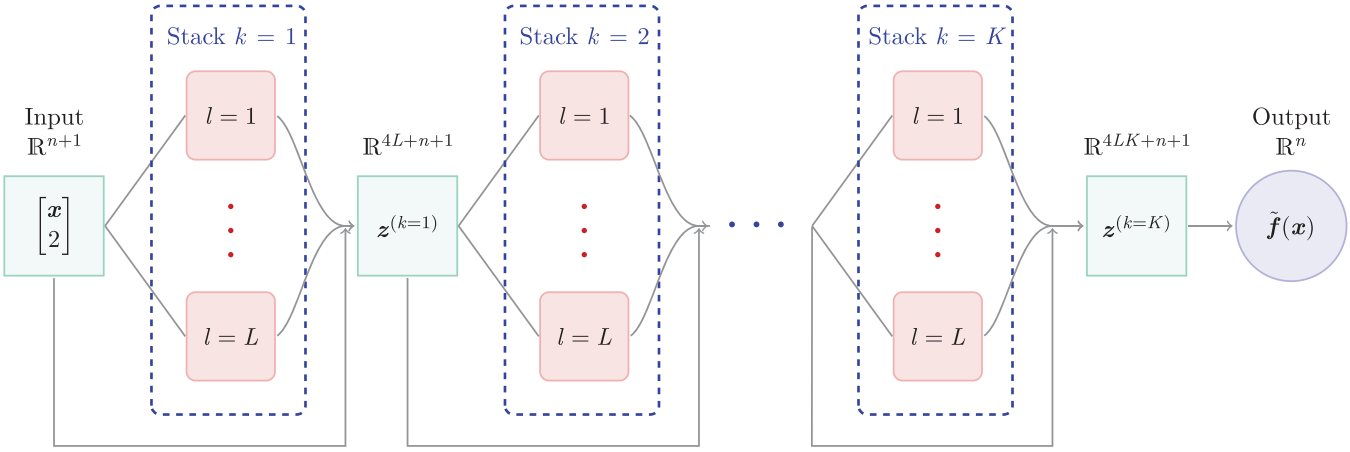

FIG. 1. Network architecture of K stacks with L operational layers within each stack. Input on the far left is the state data $\boldsymbol{x}_{\sigma_1} \in \boldsymbol{X}$ concatenated with the constant 2, making the input $\mathbb{R}^{n+1}$. Note the $4L(k-1)+n+1$ dimensional output from the $(k-1)$th stack is concatenated to the $4L$ dimensional output of the kth stack to give a $4Lk+n+1$ dimensional output. After K such stacks, the $4LK+n+1$ dimensional output of the Kth stack is reduced to $\mathbb{R}^n$ to give the network model $\tilde{\boldsymbol{f}}(\boldsymbol{x})$ on the far right. This is optimized to fit the training (noisy derivative) data $\dot{\boldsymbol{x}}_{\sigma_2} \in \boldsymbol{Y}$ to minimize the loss shown in Eq. (16). The details of each operational layer (red) are further illustrated in Fig. 2.

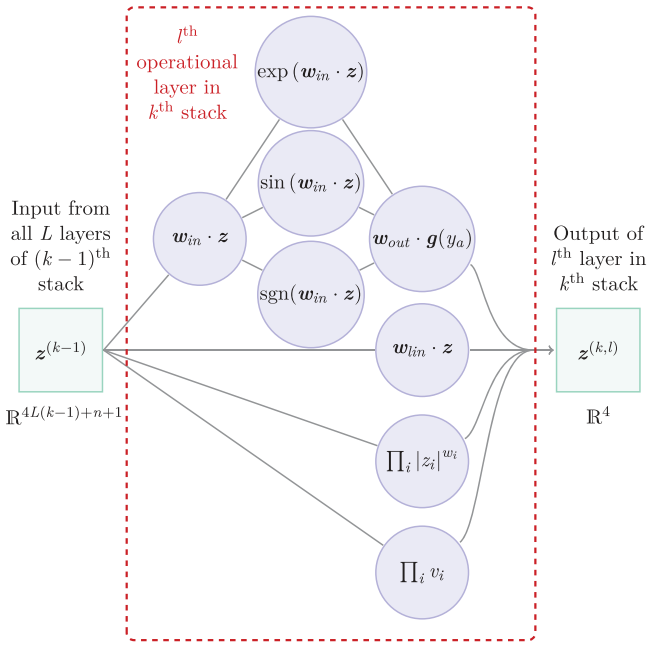

FIG. 2. Architecture of the lth operational layer in the kth stack. Input is the $4L(k-1)+n+1$ dimensional output of the $(k-1)$th stack. The output of each operational layer is four dimensional, thus giving $4L$ additional states from L layers in each stack. The sublayers (purple) denoted are $\boldsymbol{w}_{lin} \cdot \boldsymbol{z} \equiv$ linear combinations [Eq. (5)], $\prod_i |z_i|^{w_i} \equiv$ polynomial combinations [Eq. (6)], $\prod_i v_i \equiv$ simple products [Eq. (7)], $\boldsymbol{w}_{in} \cdot \boldsymbol{z}$, $\boldsymbol{w}_{out} \cdot \boldsymbol{g}(y_a) \equiv$ linear combinations [Eqs. (8) and (9)], respectively, where $\boldsymbol{g}(y_a) \equiv [\exp(\boldsymbol{w}_{in} \cdot \boldsymbol{z}),\ \sin(\boldsymbol{w}_{in} \cdot \boldsymbol{z}),\ \text{sgn}(\boldsymbol{w}_{in} \cdot \boldsymbol{z})]$ in Eq. (9). The stack and layer indices (k, l) for the weights and z denoted within the dashed red line are not shown in interest of clarity, and the summation notation is avoided to distinguish weights sublayer-wise.

represent the equation $\dot{x}_1 = {x_2}^2$ where we consider states $[x_1, x_2, 2]$ and then the corresponding learned weight vector should yield $\boldsymbol{w} = [0.0,\ 2.0,\ 0.0]$ after training.

C. Simple products

In many instances, the absolute value of a state is insufficient; the simple product sublayer allows us to combine states raised only to the first power. These operations are dictated by a product rule,

$$y = \prod_{i=1}^{n+1} [\sigma(w_i)\, x_i + (1 - \sigma(w_i))] \equiv \prod_{i=1}^{n+1} v_i. \tag{7}$$

Here, $\sigma(w_i) = \frac{1}{1+e^{-w_i}}$ represents the sigmoidal activation function so that for $w_i \ll 0$, $v_i = 1$ and for $w_i \gg 0$, $v_i = x_i$.

For example, suppose $\dot{x}_1 = x_1 x_2$ with states $[x_1, x_2, 2]$. Then, the resulting weight vector after training could, for example, be equal to $\boldsymbol{w} = [1.5,\ 2.1,\ -0.99]$; here, $\sigma(1.5) \approx \sigma(2.1) \approx 1$, while $\sigma(-0.99) \approx 0$; therefore, according to Eq. (7) yields

$$(1 \cdot x_1 + (1-1))\,(1 \cdot x_2 + (1-1))\,(0\,(2) + (1-0)) = x_1 x_2.$$

D. Common operators

These sublayers are reliant on a set of common symbolic operations. Each common operator sublayer consists of two consecutive dense sublayers; the first dense sublayer returns a linear combination of the input states,

$$y_a = \sum_{i=1}^{n+1} w_i x_i, \tag{8}$$

while the second is a linear combination of the different operations g applied to the output of the first

$$y = \sum_{j=1}^{3} g_j\left(y_a\right) W_j. \tag{9}$$

In the formulation of the model used to generate results in this paper, three possible operators g are included: the sine function, the exponential function, and the sign function. The first two were selected because of their prevalence in dynamical systems across physics, chemistry, biology, and engineering, while the third is included to accommodate the absolute-value requirement imposed on the polynomial sublayers. In principle, other commonly encountered functions, such as saturation functions and ReLU, can also be included.

We can now write down the output of each operational layer as a function. As illustrated in Fig. 2, each operational layer l in stack k takes as input a vector in $\mathbb{R}^{4L(k-1)+n+1}$ and outputs a vector in $\mathbb{R}^4$. While we showed Eqs. (5)–(9) for each operational layer in the first stack as functions of the input for simplicity, they can be generalized to every operational layer as

$$f_1^{k,l}(\boldsymbol{z}) = \boldsymbol{w}_{lin}^{k,l} \cdot \boldsymbol{z}, \tag{10a}$$

$$f_2^{k,l}(\boldsymbol{z}) = \prod_{i=1}^{d} |z_i|^{w_{pow,i}^{k,l}}, \tag{10b}$$

$$f_3^{k,l}(\boldsymbol{z}) = \prod_{i=1}^{d} \left[\sigma\left(w_{prod}^{k,l}\right) z_i + \left(1 - \sigma\left(w_{prod}^{k,l}\right)\right)\right], \tag{10c}$$

$$f_4^{k,l}(\boldsymbol{z}) = \boldsymbol{w}_{out}^{k,l} \cdot \begin{bmatrix} \exp\left(\boldsymbol{w}_{in}^{k,l} \cdot \boldsymbol{z}\right) \\ \sin\left(\boldsymbol{w}_{in}^{k,l} \cdot \boldsymbol{z}\right) \\ \operatorname{sgn}\left(\boldsymbol{w}_{in}^{k,l} \cdot \boldsymbol{z}\right) \end{bmatrix}, \tag{10d}$$

where $d = 4L(k-1) + n + 1, \boldsymbol{z} \in \mathbb{R}^{4L(k-1)+n+1}$. These four functions can be denoted as a vector $\boldsymbol{f}^{k,l} : \mathbb{R}^{4L(k-1)+n+1} \mapsto \mathbb{R}^4$ of functions

$$\boldsymbol{f}^{k,l}(\boldsymbol{z}) \equiv \begin{bmatrix} f_1^{k,1} \\ f_2^{k,1} \\ f_3^{k,1} \\ f_4^{k,1} \end{bmatrix} (\boldsymbol{z}). \tag{11}$$

Now, the neural network model $\tilde{\boldsymbol{f}}(\boldsymbol{x})$, illustrated in Fig. 1 as the output of the neural network, can be written as

$$\tilde{\boldsymbol{f}}(\boldsymbol{x}) = \boldsymbol{W}_{out} \begin{bmatrix} \boldsymbol{f}^{K,1} \\ \boldsymbol{f}^{K,2} \\ \vdots \\ \boldsymbol{f}^{K,L} \\ \mathscr{I} \end{bmatrix} \circ \cdots \circ \begin{bmatrix} \boldsymbol{f}^{2,1} \\ \boldsymbol{f}^{2,2} \\ \vdots \\ \boldsymbol{f}^{2,L} \\ \mathscr{I} \end{bmatrix} \circ \begin{bmatrix} \boldsymbol{f}^{1,1} \\ \boldsymbol{f}^{1,2} \\ \vdots \\ \boldsymbol{f}^{1,L} \\ \mathscr{I} \end{bmatrix} \left(\begin{bmatrix} \boldsymbol{x} \\ 2 \end{bmatrix}\right), \tag{12}$$

where $\mathscr{I}$ is the Identity operator. Here, we can see the generative nature of our algorithm introduced in (1), where each operational layer l in stack k, represented as $\boldsymbol{f}^{k,l}$ in (11), is made up of primitives from Eqs. (10a)–(10d), which are the sublayers elaborated earlier.

TABLE I. Examples of sublayer outputs for $n = 2$.

Sublayer type	Example output expression	Trainable parameters
Linear combination	$0.7x_1 + 1.5x_2 - 1$	$n+1$
Polynomial combination	$0.25\ \lvert x_1 \rvert^{-1.4}\ \lvert x_2 \rvert^{0.5}$	$n+1$
Simple products	$(0.3 + 0.7x_1)x_2$	$n+1$
Common operators	$0.9 \sin(0.2x_1 - 0.1\ x_2) + 0.3\exp(0.2x_1 - 0.1x_2)$	$(n+1)+3$

Collectively, the structure of these operational layers allows us to achieve tremendous flexibility in the expressions the network is capable of generating, while simultaneously tackling the challenge of overfitting. While we have a significant range of possible expressions, we are able to do so with fewer trainable parameters than conventional artificial neural networks even when the network is several stacks deep or the number of copies of the operational layers in each stack is high. Examples of the sublayer outputs when applied to a $(n+1) \times 1$ vector of states are provided in Table I, demonstrating how low the count of trainable parameters is when compared to fully connected or even convolutional networks.[29] The smaller number of parameters is a design consequence but offers significant advantages over conventional DNNs.

The fact that commonly occurring terms in models are functions of state makes them analogous to activation functions in conventional DNNs. Many functions, such as $\boldsymbol{x}$, $\prod_i |x_i|^{w_i}$, $\sin(\boldsymbol{w} \cdot \boldsymbol{x})$, and $e^{\boldsymbol{w} \cdot \boldsymbol{x}}$, have global support, unlike conventional activation functions such as $\tanh(\boldsymbol{w} \cdot \boldsymbol{x})$ and $\mathrm{ReLU}(\boldsymbol{w} \cdot \boldsymbol{x})$. While this reduces fidelity of the neural network's approximation capability, it also means that activation functions with local support need not be centered around data points as is required in k-means clustering or other activation function centering algorithms. Consequently, far fewer parameters are required $\left[O(10^2)\right]$ compared to conventional DNNs. This carries advantages in terms of generalizability, computational economy, and data requirements. The generalizability aspect is immediate from the non-local support, meaning that newer data points far away from the seen data $\boldsymbol{X}$ would not need activation functions centered around them. The computational requirements are far smaller both in terms of memory and processing requirements, a consequence of the smaller number of trainable parameters. The examples shown in this paper are run on a fairly standard laptop PC with an AMD Ryzen 4900HS processor and 16GB of RAM. The time taken from generating the training data to obtaining the equations and figures can range from a few minutes to an hour depending on the dataset and neural network sizes.

E. Regularization and initialization

We designed regularization and initialization of the neural network to achieve two primary goals: (1) promote parsimony in the

output equation and (2) prevent exploding loss/gradient values in deeper multi-stack implementations. To this end, appropriate initializers and regularizers were selected for each layer. We will briefly detail and motivate the choices made for each operational layer here.

We make two preliminary observations before proceeding. First, parsimony is generally enforced through sparsity, but in the context of our network, zero matrix weights do not directly correspond to sparsity in most layers. There is a direct connection between parsimony and sparsity in the dense output layer and other dense layers. Second, the primary cause of exploding errors or gradients during training of multi-stack networks arises from the compounding effects of variables being multiplied by themselves; therefore, in general, initialization is designed to make most terms initially nearly constant-valued.

1. Linear combinations

A core challenge across our linear combination layers is that correct terms may have relatively large coefficients, and the final mean absolute error for noiseless systems approaches 0. With this in mind, we note that we would ideally like the linear combination (and dense output) layers to function as feature selectors, which motivated us to implement L_1 regularization. We found, however, that the linear growth in error as coefficients increased was still somewhat unfavorable; therefore, we instead implemented $L_{1/2}$ regularization.[30] $L_{1/2}$ regularization allows for greater sparsity than L_1 regularization while simultaneously providing diminishing penalties for increasing weight magnitudes. The regularization loss can be calculated as

$$\alpha_1 \sum_{i=1}^{n+1} |w_i|^{1/2} \equiv \alpha_1 L_{1/2}. \tag{13}$$

Here, α_1 is a tuning parameter to modify the relative influence of this regularization. Linear combinations do not contribute significantly to the problem of exploding gradients in our application, and it is straightforward to initialize the system to small values; any standard initialization protocol with zero mean and non-constant initialization of weights is acceptable. We use hyperparameter value $\alpha_1 = 0.05$.

2. Polynomial combinations

Unlike the linear combinations considered above, the polynomial combination layer does not correspond to sparsity in the output when all weights are 0. Rather, this corresponds to a constant value. Considering parsimony more broadly, we prefer simpler polynomial terms to more complicated ones: the lower the overall degree of the expression, the more we generally prefer it; for example, we view x_1^2 as preferable to both x_1^5 and $x_1^2 x_2$; we would also prefer $x_1 x_2$ to $x_1^2 x_2$ or x_1^5. Our primary mechanism of modulating the sparsity in the output expression is the constant value of 2 we previously appended to our state to assist with scaling. If the exponent on 2 is very negative (say, -10), then that term is virtually negligible. We, therefore, designed a novel regularizer to reward highly negative values of the exponent on 2 while penalizing large magnitudes (negative or positive) of the exponents for the non-constant input values. Formally, our regularizer loss is calculated as

$$\alpha_2 1.1^{\sum_{i=1}^{n} |w_i| + w_{n+1}} \equiv \alpha_2 L_{poly}, \tag{14}$$

which approaches 0 as $w_{n+1} \to -\infty$ and approaches infinity as the magnitudes of the non-constant values' exponents increase. We note that terms in the network model $\tilde{f}(\cdot)$ may be negligible in magnitude due to regularization but are not discarded unless deemed insignificant by the information criterion in the selected network model $\tilde{f}_A(\cdot)$, as described in Sec. III.

Unlike the case of the linear combination operational layers, here, our initialization is critically important for ensuring stability of the system as the number of stacks increases. As noted previously, we generally aim for the initial value of each layer's output to be near-constant; in the case of the polynomial combination layer, this corresponds to a zero vector weight matrix; therefore, we initialize the layer with a normal distribution about a mean of 0. For deeply stacked implementations, it is important that the standard deviation of this distribution is small in magnitude to avoid numerical issues. We use the hyperparameter value $\alpha_2 = 0.01$.

3. Simple products

Regularization is less of a concern for the simple product operational layer because the output structure is already significantly constrained. All simple product layers output polynomials with degree no higher than the number of input terms, and the degree of a given variable within that expression is always either 1 or 0. As such, the enforcement of sparsity as it relates to these expressions can be safely applied at the dense output layer stage, rather than within the operational layer.

In contrast, initialization of the simple product operational layers is vital to the success of multi-stack implementations. Standard, mean-zero initialization (as was used elsewhere) will cause significant numerical issues in deeper networks because a weight of 0 does not correspond to a constant value, but rather to terms of the form $0.5 + 0.5x_1$. Instead, we center our initialization around a negative number, with the standard deviation and mean both decreasing as we wish to deepen our network. For systems with relatively few stacks, we found a mean of -1.0 to sufficiently protect against exploding errors and gradients.

4. Common operators

The common operational layers are split into two sublayers of weights: first a sublayer performing a linear combination of the input expressions and then a sublayer performing a linear combination of the outputs of the operators acting on the first linear combination. Applying $L_{1/2}$ regularization to the first of these sublayers will help with sparsity and mitigate the risks of exploding gradients, but it is insufficient to promote parsimony since some operators possess nonzero values for zero-valued inputs. As such, we apply $L_{1/2}$ regularization to both the first and second sublayers of our system. However, we found in our hyperparameter studies that a reduced weight of $\alpha_3 = 0.0375$ keeps these functions from being

penalized too much, and thus, we include it alongside Eq. (9) as

$$\alpha_3 \sum_{i=1}^{3} |w_i|^{\frac{1}{2}} \equiv \alpha_3 L_{ops}. \tag{15}$$

Initialization is important here as well, and we again make sure to initialize about mean zero. However, we take an extra precaution here of also decreasing the standard deviation of our initialization distribution as we proceed through subsequent stacks of our network to further reduce the risk of initially high gradients and losses.

F. Hyperparameters

We use certain hyperparameter values (number of stacks and layers) to identify a variety of systems, which is analogous to using the same hyperparameters (width and depth) in a deep neural network to perform regression on entirely different datasets. As with traditional machine learning, setting the hyperparameters is not an exact science. Even when set after numerical experiments for one system such that the desired metrics are at their best, using data from a different system often requires re-tuning the hyperparameters. However, we found that a set of hyperparameters such as $K = 1$ stack with $L = 10$ layers, or $K = 2$ stacks with $L = 2$ layers, works well across multiple systems, as demonstrated in Sec. IV. This is in conjunction with the same common operators of exponential, sinusoidal, and sign functions. Thus, we use the same hyperparameters to identify systems with fixed points, limit cycles, and chaotic attractors.

When using $K = 1$ stack, we found $L < 10$ layers to be sufficient for some systems, but all systems were identifiable using $K = 1$ stack and $L = 10$ layers. This is surprising as many of the systems are, in principle, sufficiently expressed with just $L = 1$ or $L = 2$ layers. Such overparameterization is common in deep neural networks with hundreds of millions of parameters, but so is the issue of overfitting to training data. In contrast, our overparameterization seems to produce equations that are generalizable beyond the training data. This phenomenon of overparameterization beyond conventionally accepted number of parameters optimal to the trade-off between bias and variance[31] has recently been reported[32] and is an active area of research.

The hyperparameters used in demonstrations of SymANNTEx presented in this work are summarized in Table II. These examples span various dynamic behaviors, such as hyperbolic (Takens–Bogdanov) and elliptic (simple pendulum) fixed points, limit cycles (FitzHugh–Nagumo, chemical kinetics) and a continuum of periodic orbits (simple pendulum), fast–slow dynamics (FitzHugh–Nagumo), and chaos (Lorenz, Rössler, Chua), governed by equations with sinusoidal (simple pendulum), exponential (chemical kinetics), and polynomial terms. We see that such a wide array of systems are identified by SymANNTEx using nearly the same hyperparameters of training.

G. Loss function and training

We use a combination of the mean absolute error (MAE) and sublayer-dependent regularization as the loss, which we seek to minimize using iterative optimizers, initialized with layer-dependent initial weights. While regression in Euclidean spaces is widely done using the mean squared error (MSE) with L_1 regularization, we found (as shown in Sec. V) this conventional approach to be (1) lacking desired parsimony and (2) giving enormous errors aggravating the optimization. Instead, we use $L_{1/2}$ regularization,[30] which is known to promote sparsity to a greater extent than L_1 does, but at

TABLE II. Hyperparameters for demonstrated examples.

Parameter	Pendulum	Chua	Takens–Bogdanov	Rössler	Lorenz	FitzHugh–Nagumo	Chemical kinetics
Stacks	1						2
Layers	10						1
Learning rate constant	0.032					0.01	
$L_{1/2}$ regularization weight	0.05						
L_{poly} regularization weight	0.01						
L_{ops} regularization weight	0.0375						
AIC tolerances	Equation (18)						
Number of data points	1000						16 000
Training: test split	800: 200						12 000: 4000
Training instances	5 (1 per split)						
Training epochs	100	6400	25	400	6400	3200	6400
Batch size	$\min\{\sqrt{trainingsplit}, 32\}$						
Training noise	$\sigma_1 = 0.01$ (up to 0.025 in some examples),			$\sigma_2 = 0.01$ (up to 0.075 in some examples)			$\sigma_1 = \sigma_2 = 0.001$

the cost of losing convexity. The loss function is given by

$$\text{Loss} = \text{MAE} + \sum_{k=1}^{K}\sum_{l=1}^{L}\left(\alpha_1 L_{1/2}^{(k,l)} + \alpha_2 L_{poly}^{(k,l)} + \alpha_3 L_{ops}^{(k,l)}\right), \tag{16}$$

where

$$\text{MAE} = \frac{1}{m}\sum_{j=1}^{m} \| \dot{\boldsymbol{x}}_{\sigma_2}^{[j]} - \tilde{\boldsymbol{f}}\left(\boldsymbol{x}_{\sigma_1}^{[j]}\right) \|_1,$$

and $L_{1/2}^{(k,l)}$, $L_{poly}^{(k,l)}$, and $L_{ops}^{(k,l)}$ are custom regularizers at each sublayer (denoted by the superscript) in the stacks $k \in \{1, \ldots, K\}$ and operational layers $l \in \{1, \ldots, L\}$. α_1, α_2, and α_3 are user-defined regularization weights. The *possibly* longer computational time taken to minimize a non-convex loss is outweighed by the greater sparsity in fewer iterations of training. Iterations of training are commonly measured in "epochs" where one epoch is when each of the data points has been used once in the optimization. Greater sparsity in fewer epochs is desirable for increased interpretability.

We use the Adam optimizer[33] to minimize the loss function. We use a 4: 1 randomized split of the data for training and testing, respectively, in a `Kfold` routine that uses different randomized initial weights for each of the five instances of the optimization. This fivefold optimization increases the likelihood of an optimal set of network weights by concealing a different fifth of the data in each of the five training runs and choosing that which has the smallest magnitude of the loss function among all five of the instances. Although it is an algorithm with an adaptive learning rate, it has a learning rate constant. We found the commonly used default learning rate constant of 0.001 to be "insufficient" even with 10^5 epochs; yet, a learning rate constant of 0.01 seems to yield convergence well within 10^4 epochs. This larger learning rate constant and the spurts of an instantaneous increase in the loss function value at some iterations suggest that we could be observing what has recently been reported as the edge of stability.[34] Our investigations with learning rate constants of {0.001, 0.004, 0.016} in our examples, each with datasets of {1000, 4000, 16 000}, indicate the lack of a linear relationship between step-size and the number of epochs taken for convergence to the expected equations. At slower learning rates, we found that the optimization seems to fall short of sparsity even when run for longer. In the interest of practical applications, we use higher learning rates that seem to remedy this over fewer training epochs. This is especially apparent with the Lorenz and FitzHugh–Nagumo models. While some models can be identified from as few as 25 epochs, we train some examples for up to 6400 epochs. Thus, we have used a learning rate constant of 0.01 in almost all of our examples. It works for all the dataset sizes used in our investigation; therefore, we show results using the smallest datasets, which have only 1000 data points. These hyperparameters are summarized in Table II.

The learning rate for the simple pendulum has been increased by a factor of three. We found that the optimizer identifies a harmonic oscillator even when trained for 12 800 epochs with a learning rate of 0.01, whereas increasing the learning rate to 0.032 identifies Eq. (23) in as few as 25 epochs. This indicates that the harmonic oscillator is a local minimum for the optimization in parameter space, which is also the classical linear approximation to the simple pendulum. Reducing the regularization weight α_3 [Eq. (15)] of the common operators in Eq. (9) alleviates this but makes the regularization sub-optimal in identifying the other examples. We also found that the higher learning rate worked better for the Chua double-scroll system.

III. INFORMATION CRITERION FOR PARSIMONIOUS MODELS

We seek parsimonious expressions for the network model for given (noisy) data because this increases interpretability. In practice, we find that the network model has coefficients that (1) do not exactly match those of the ground truth model, although they are often approximately equal, and (2) there are a large number of small but non-zero coefficients despite sparsity-promoting regularization of the empirical loss. Thus, we need a "post-processing" step that tunes the coefficients of the network model, including the possibility of setting coefficients of terms to zero. This can be posed as a problem in model selection.

We use the Akaike Information Criterion (AIC)[27] to fine-tune the network model's parameters. If an approximate model can be expressed using fewer terms in the equation, it could be viewed as carrying similar informativity but with fewer parameters. Information theoretic criteria have been used in statistics for model selection given a possible set of models, including with SINDy;[35] in particular, AIC can be viewed as quantifying the complexity of a model via the number of terms in its equation alongside the residual relative to the actual data, found from maximum likelihood estimates. As we work in a Euclidean space, we use the MSE for m data points. AIC uses the log-likelihood estimate alongside a correction term that accounts for finite data size,[36]

$$\text{AIC} = \left(2P + m\log_e(\text{MSE})\right) + 2\frac{(P+1)(P+2)}{m-P-2}, \tag{17}$$

where

$$\text{MSE} = \frac{1}{m}\sum_{j=1}^{m}\left\| \dot{\boldsymbol{x}}_{\sigma_2}^{[j]} - \tilde{\boldsymbol{f}}_S\left(\boldsymbol{x}_{\sigma_1}^{[j]}\right)\right\|_2^2,$$

and P is the number of estimated parameters. Here, $\tilde{\boldsymbol{f}}_S(\cdot)$ is a "simpler" model obtained by using the Python library SymPy's `nsimplify` that rounds parameters of the network model $\tilde{\boldsymbol{f}}(\cdot)$ to within a given tolerance. This command also searches for commonly occurring constants like π, e, etc., to within the tolerance. We compute the AIC scores for the network model over a range of user-specified tolerances and select the model $\tilde{\boldsymbol{f}}_A(\cdot)$ with the lowest AIC score. The tolerances considered are 11 logarithmically spaced numbers between 0.01 and 1 as

$$\text{Tolerances used} \equiv [0.01,\ 0.0158,\ 0.0251,\ 0.0398,\ 0.0631,\ 0.1,\ 0.1585,\ 0.2512,\ 0.3981,\ 0.6310,\ 1]. \tag{18}$$

Rounding parameters of the network model $\tilde{\boldsymbol{f}}(\cdot)$ to each of these 11 tolerances could give up to 11 models differing in parameters. The AIC score is computed on each of these models, and the one with the lowest score is the selected network model $\tilde{\boldsymbol{f}}_A(\cdot)$. The effect of the above rounding-off can sometimes be drastic when used in conjunction with sparsity-inducing regularization because many of

TABLE III. Approximate relative-RMSE for each of the examples computed using ground truth models over 100 000 data points spread across 1000 trajectories for the non-chaotic examples and over single trajectories computed for $T = 1000$ time-units for the chaotic examples.

Model	Approximate relative-RMSE (%)
Takens–Bogdanov	2.04
Simple pendulum	4.37
Rössler	2.69
Lorenz	2.67
FitzHugh–Nagumo	2.64
Chemical kinetics	0.44
Chua double-scroll	1.83

the network model parameters could be close to 0. By following the aforementioned procedure on a network model $\tilde{\boldsymbol{f}}(\cdot)$ with 322 parameters, the Lorenz system is identified exactly as $\tilde{\boldsymbol{f}}_A(\cdot)$ with five parameters or the simple pendulum is identified closely with two parameters.

IV. EXAMPLES

We have successfully identified a number of models whose characteristics span a variety of dynamic behaviors of scientific and engineering importance using noisy data. Here, we provide some of those examples to demonstrate identification of systems with stable, unstable, and saddle fixed points, a limit cycle, and chaotic behavior. Some of these systems also show timescale separation via fast initial transient dynamics onto a slow manifold. We also compare trajectories generated from a random initial condition by the selected network models to that of the ground truth model.

In this paper, we present examples trained on data with 1% noise [as defined in Eq. (3)], but we first show how that added noise translates to the metrics of performance used. Although we use the MAE in the loss function [Eq. (16)] that we optimize for, our data lie in a Euclidean space so that we look at the root mean squared error (RMSE), which is the l^2 error averaged over our dataset. We already defined MSE in Eq. (17) and the RMSE is the square-root of the same, but it can vary across different dynamical systems; therefore, we consider the relative-RMSE, which is defined as

$$\text{Relative-RMSE} = \sqrt{\frac{\sum_{j=1}^{m} \left\| \tilde{\boldsymbol{f}}_A\left(\boldsymbol{x}_{\sigma_1}^{[j]}\right) - \dot{\boldsymbol{x}}_{\sigma_2}^{[j]} \right\|_2^2}{\sum_{j=1}^{m} \left\| \dot{\boldsymbol{x}}_{\sigma_2}^{[j]} \right\|_2^2}}. \tag{19}$$

The above would be non-zero even if the ground truth model is identified exactly since $E(\dot{\boldsymbol{x}}_{\sigma_2}) \neq E(\boldsymbol{f}(\boldsymbol{x}_{\sigma_1}))$ from Sec. I A. While in principle, it is possible that some $\tilde{\boldsymbol{f}}_A(\cdot)$ could over-fit any finite dataset, we know that the least relative-RMSE of interest is obtained if $\tilde{\boldsymbol{f}}_A \equiv \boldsymbol{f}$. This can be used to lower-bound the relative-RMSE. Computing it theoretically becomes cumbersome; therefore, we instead provide numerical estimates in Table III using Eq. (19) for each example, in the limit of large data, to better approximate the errors.

A. Takens–Bogdanov normal form

We consider the Takens–Bogdanov normal form[37] to demonstrate identification of a model using data for a system that has fixed points with different stability characteristics,

$$\dot{x} = y, \tag{20a}$$

$$\dot{y} = \mu_1 + \mu_2\, y + x^2 + x\, y. \tag{20b}$$

This is the prototypical example of a dynamical system which is close to parameter values for which a fixed point has a double zero-eigenvalue, and possible special solutions can include fixed points $P_\pm \equiv (\pm\sqrt{-\mu_1}, 0)$ and limit cycles. We consider the parameter values $\mu_1 = -4.41$, $\mu_2 = 1.5$, which is a first-order approximation to values for which there exists a homoclinic orbit.[37] For the considered parameter values, there is a stable fixed point P_-, an unstable (saddle) fixed point, no limit cycles, and trajectories with initial conditions outside of the basin of attraction of P_- grow without bound.

To test our system identification algorithm, we consider as input the four randomly initialized trajectories shown in Fig. 3(a). There are a total of 1000 data points (250 for each trajectory of length $T = 1$). We find that SymANNTEx identifies the model

$$\dot{x} = y, \tag{21a}$$

$$\dot{y} = -4.333 + 1.5\, y + x^2 + x\, y, \tag{21b}$$

using the following network structures: (1) $L = 10$ layers with $K = 1$ stack with 236 trainable parameters and (2) $L = 1$ layer and $K = 2$ stacks with 68 trainable parameters. This model is estimated after 25 epochs on $\boldsymbol{Y}$ at $\boldsymbol{X}$ shown in Fig. 3 with a relative-RMSE of ≈2.10%. For reference, the ground truth model gives an error of ≈2.04% on the dataset referred to in Table III. SymANNTEx estimated similar models up to noise levels of $\sigma_1 = 0.02, \sigma_2 = 0.075$.

B. Simple pendulum

To demonstrate the ability to identify models with sinusoidal terms, we consider the simple pendulum

$$\ddot{\theta} = -\frac{g}{l}\sin\theta, \tag{22}$$

with $g = 9.81$, $l = 2$. As our algorithm is designed to approximate first-order ordinary differential equations (ODEs) as functions of state, we reformulate the above second-order ODE equation as two first-order ODEs, which is a classical technique in analyzing dynamical systems.[37] The simple pendulum has a continuum of time-periods ranging from $2\pi\sqrt{\frac{l}{g}}$ near the fixed point at $(0,0)$ to infinity near the unstable fixed points $(\pm k\pi, 0)$. We remark that this continuum of time periods is reflected in the Koopman operator perspective as a continuous spectrum,[38] which makes analysis using DMD challenging.[39]

To test our algorithm, we consider as input the four randomly initialized trajectories shown in Fig. 4. There are a total of 1000 data points (250 for each trajectory of length $T = 4$). We find that

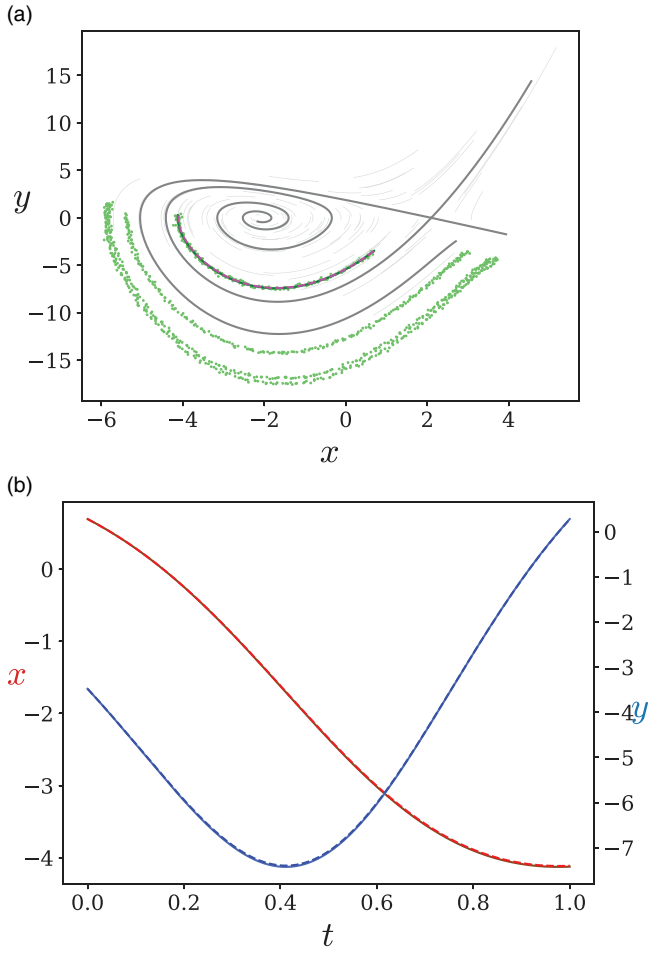

FIG. 3. (a) State-space comparison of the trajectory generated (dashed magenta) by Eqs. (21a)–(21b) estimated at $\boldsymbol{X}$ (light green) with that generated by the ground truth model [Eqs. (20a) and (20b)] (solid green), which here is visually indistinguishable. The stable and unstable manifolds of the saddle fixed point are shown as dark gray lines, and other trajectories are shown as light gray lines for better depiction of the dynamics. (b) The same trajectory (dashed) plotted as a time series compared with that generated by the ground truth model (solid), which are nearly identical. (a) Phase portrait. (b) Time series.

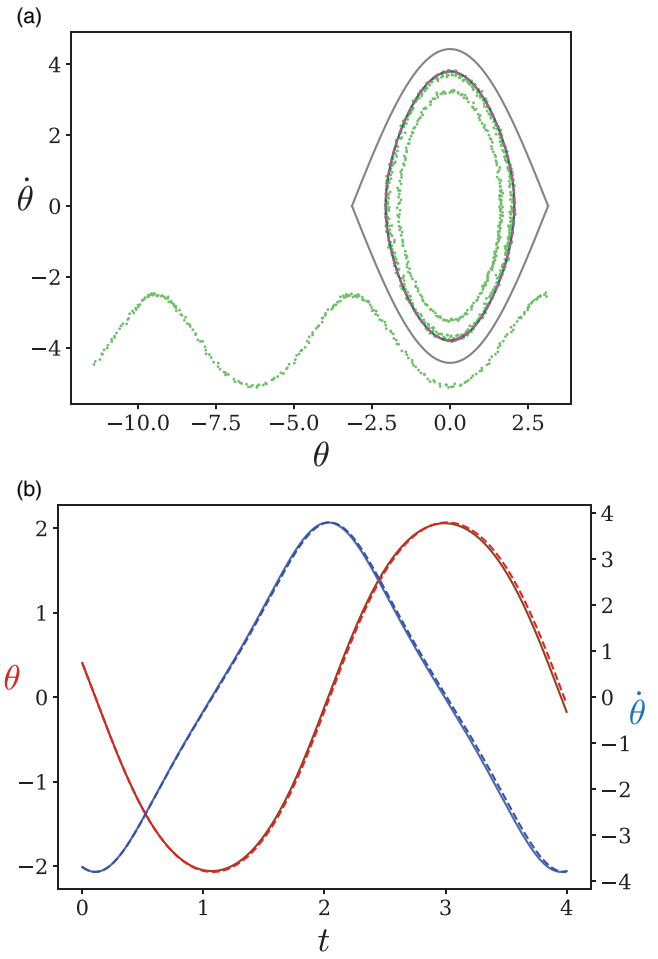

FIG. 4. (a) State-space comparison of the trajectory generated (dashed magenta) by Eq. (23) estimated at $\boldsymbol{X}$ (light green) with that generated by the ground truth model [Eq. (22)] (solid green), which here is visually indistinguishable. The separatrices between angular displacement $-\pi$ and π are shown as dark gray lines. (b) The same trajectory (dashed) plotted as a time series compared with that generated by the ground truth model (solid), which are nearly identical. (a) Phase portrait. (b) Time series.

our algorithm identifies the model in the state-space form, denoting $x \equiv \theta$ and $y \equiv \dot{\theta}$, as

$$\dot{x} = y, \tag{23a}$$

$$\dot{y} = -4.857 \sin x, \tag{23b}$$

using $L = 10$ layers and $K = 1$ stack with 236 trainable parameters after 100 epochs on training data $\boldsymbol{Y}$ at the input data $\boldsymbol{X}$ with a relative-RMSE of ≈4.45%. For reference, the ground truth model gives an error of ≈4.37% on the dataset referred to in Table III. Here, we remark that a higher learning rate of 0.032 is used to identify the above equations in as few as 25 epochs as compared to a learning rate of 0.01 only giving a linear approximation even after 12 800 epochs, indicating a large region of local minima. SymANNTEx estimated similar models up to noise levels of $\sigma_1 = 0.02, \sigma_2 = 0.075$.

C. Rössler equations

We now demonstrate the identification of chaotic dynamical systems from noisy data. Identifying the flow-map or solution of state of chaotic systems is challenging due to their sensitive dependence on initial conditions and usual lack of closed-form solutions. Their analysis using DMD methods is also challenging due to the lack of discrete modes.[12] Here, we consider the Rössler equations,

which find relevance in chaotic behavior of chemical reactions,[40]

$$\dot{x} = -y - z, \tag{24a}$$

$$\dot{y} = x + 0.5\, y, \tag{24b}$$

$$\dot{z} = 2 + z\,(x - 4). \tag{24c}$$

This model has a chaotic attractor with sensitive dependence on initial conditions. It also has a region of state-space beyond the attractor's region of attraction where trajectories grow unboundedly.

To test our algorithm, we consider 1000 data points from a single randomly initialized trajectory (equally spaced in time on the time interval $[0, 100]$), as shown in Fig. 5. Since the terms in (24a)–(24c) can be represented without stacking, we use $L = 10$ layers and $K = 1$ stack with 322 trainable parameters. This network was found to successfully identify the equations exactly, including the exact parameters, in 800 epochs on training data $\boldsymbol{Y}$ at the input data $\boldsymbol{X}$, which has a relative-RMSE of ≈2.60%. We remark that the noise levels in Table III are estimates that vary slightly with the number of data points and are, therefore, not exact. SymANNTEx estimated the exact model up to noise levels of $\sigma_1 = 0.01, \sigma_2 = 0.05$.

D. Lorenz equations

We further demonstrate the ability of SymANNTEx to identify chaotic dynamical systems from noisy data. We consider the classical Lorenz equations with standard parameters, originally used to model two-dimensional atmospheric convection,[41]

$$\dot{x} = 10\,(y - x), \tag{25a}$$

$$\dot{y} = x\,(28 - z) - y, \tag{25b}$$

$$\dot{z} = x\,y - 8z/3. \tag{25c}$$

This model has a chaotic attractor with sensitive dependence on initial conditions and a lack of closed-form expression for its solution of state, which manifests itself in forecasting time series. Reservoir computing[42] does an excellent job in predicting the state for many Lyapunov time-units but eventually diverges. Its associated Koopman operator lacks a discrete spectrum,[13] and thus, analysis using DMD is challenging with results continuously varying with the order of approximation.

To test our algorithm, we consider 1000 data points from a single randomly initialized trajectory (equally spaced in time on the time interval $[0, 25]$), as shown in Fig. 6. Since the terms in (25a)–(25c) can be represented without stacking, we use $L = 10$ layers and $K = 1$ stack with 322 trainable parameters. This network was found to successfully identify the equations exactly, including the exact parameters, in 6400 epochs on training data $\boldsymbol{Y}$ at the input data $\boldsymbol{X}$, which has a relative-RMSE of ≈2.59%. We remark that the noise levels in Table III are estimates that vary slightly with the number of data points and are, therefore, not exact. This example with coefficients as large as 28 alongside most of the other coefficients from the trainable parameters that get approximated to 0 shows the wide range of parameters that can be identified across orders of magnitude. SymANNTEx estimated the exact model up to noise levels of $\sigma_1 = 0.01, \sigma_2 = 0.05$.

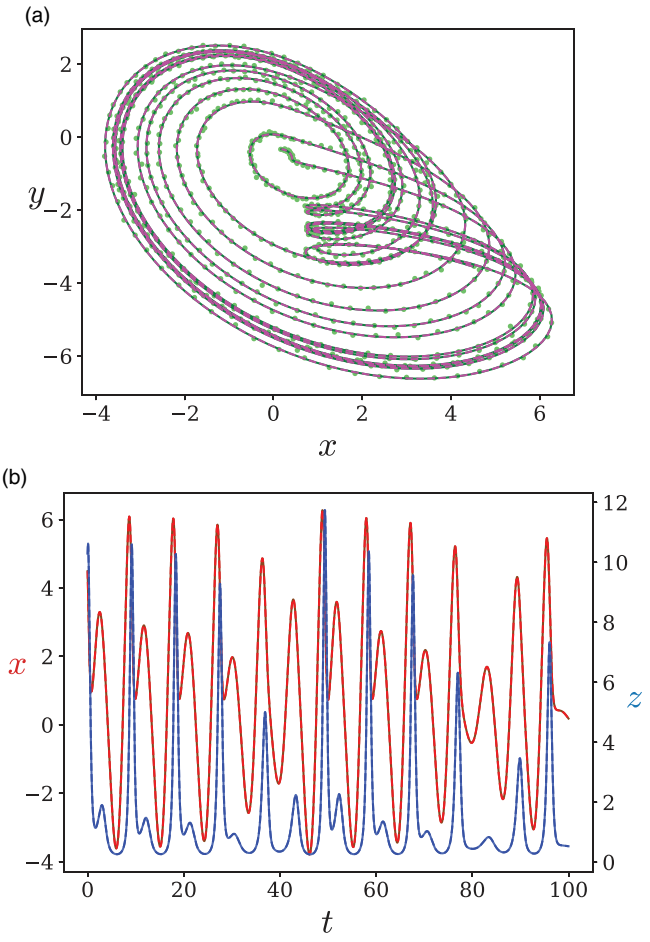

FIG. 5. (a) State-space visualization of the trajectory generated by Eqs. (24a)–(24c) (solid green). It coincides with that generated (dashed magenta) by the model identified at $\boldsymbol{X}$ (light green) as it is the same as the ground truth model. (b) The same trajectory (dashed) plotted as a time series coinciding with the ground truth. The time-evolution of the states is without any coherence due to the chaotic nature of the system. [z is not shown in (a), and y is not shown in (b) to avoid cluttering the figures.] (a) Phase portrait. (b) Time series.

E. FitzHugh–Nagumo equations

We consider the FitzHugh–Nagumo model[43]

$$\dot{v} = v - v^3/3 - w + I, \tag{26a}$$

$$\dot{w} = \varepsilon(v + aw + b) \tag{26b}$$

to demonstrate identification of a system with oscillatory behavior and a separation of timescales. This is a model for neural dynamics, which shares key features with conductance-based models, such as the Hodgkin–Huxley equations. Here, we use the parameter values $I = 0.328$, $\varepsilon = 0.08$, $a = -0.8$, $b = 0.7$, for which there is a stable limit cycle with spiking behavior.

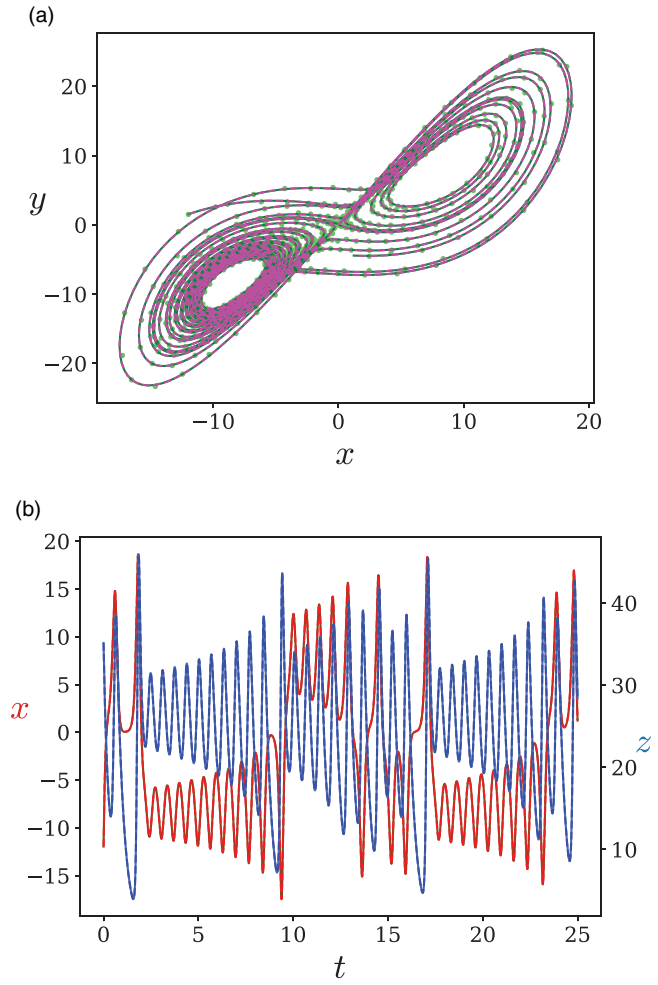

FIG. 6. (a) State-space visualization of the trajectory generated by Eqs. (25a)–(25c) (solid green). It coincides with that generated (dashed magenta) by the model identified at $\boldsymbol{X}$ (light green) as it is the same as the ground truth model. (b) The same trajectory (dashed) plotted as a time series coinciding with the ground truth. The time-evolution of the states is without any coherence due to the chaotic nature of the system. [z is not shown in (a) and y is not shown in (b) to avoid further cluttering the figures.] (a) Phase portrait. (b) Time series.

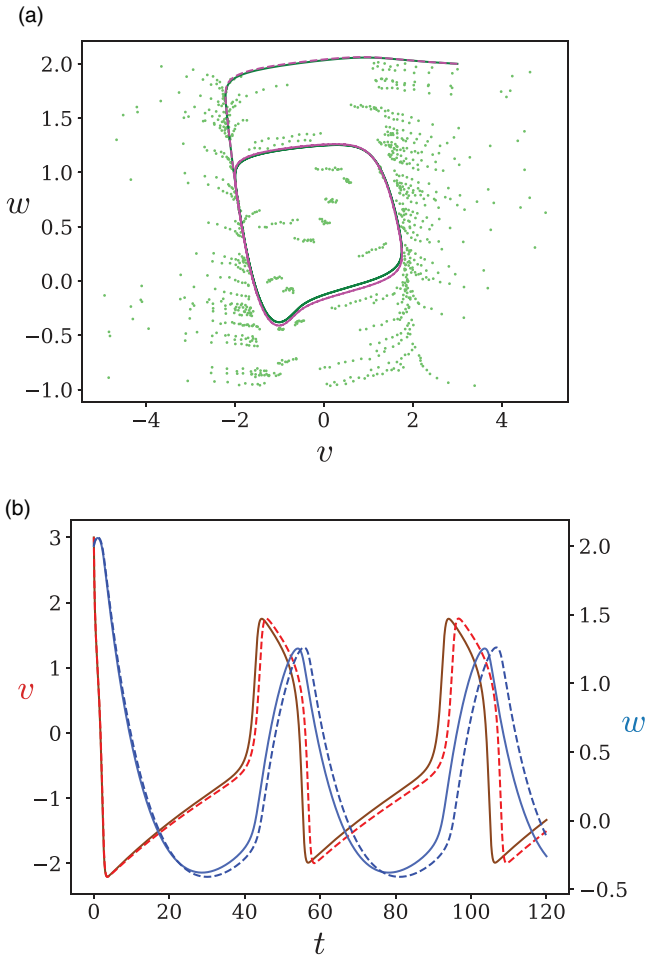

FIG. 7. (a) State-space comparison of trajectory generated (dashed magenta) by Eqs. (27a) and (27b) estimated at $\boldsymbol{X}$ (light green) with that generated by the ground truth model [Eqs. (26a) and (26b)] (solid green). The fast initial transient of the trajectory and subsequently closely following the slow manifold onto the periodic orbit are almost exact. The periodic orbit itself is also reproduced closely. (b) Evolution of the same trajectories as a time series shows how the transients are almost exact, and the time-period of the identified model (dashed) is slightly different from the ground truth model (solid). (a) Phase portrait. (b) Time series.

To test our algorithm, we consider as input 100 randomly initialized trajectories shown in Fig. 7(a). There are a total of 1000 data points (10 for each trajectory of length $T = 1$). We find that SymANNTEx identifies the model

$$\dot{v} = 0.998\, v - v^3/3 - 0.971\, w + 0.311 - 0.006\, v^2, \tag{27a}$$

$$\dot{w} = 0.078\, v - 0.06\, w + 0.054 \tag{27b}$$

using $L = 1$ layer and $K = 2$ stacks with 68 trainable parameters. We note that stacking is necessary because powers of state are taken on the absolute value; therefore, the cubic term in Eq. (26a) cannot be exactly formed using a single stack: $v^3 = |v|^2 \times v$. This model is identified in 3200 epochs on training data $\boldsymbol{Y}$ at the input data points $\boldsymbol{X}$ with a relative-RMSE of ≈2.40%. For reference, the ground truth model gives an error of ≈2.64% on the dataset referred to in Table III indicating a *slight* overfit. Yet, from Fig. 7, we remark that the phase portraits nearly coincide with the ground truth model, including the dynamic range of oscillations and the fast and slow transients. In particular, the time-period of the limit cycle behavior is also nearly the same, as seen from Fig. 7(b). SymANNTEx estimated similar models up to noise levels of $\sigma_1 = 0.025$, $\sigma_2 = 0.025$.

F. Chemical kinetics with Arrhenius rate dependence

We consider the exponential approximation to Arrhenius rate law in chemical kinetics in the context of two first-order reactions,[44] which gives the nonlinear dependence of the reaction rate on the temperature. When the rise in temperature in an exothermic reaction is small in comparison with the ambient temperature, the rate can be approximated as exponentially dependent on the rise in the temperature. After scaling the governing equations of mass and energy by relevant quantities,

$$\dot{\alpha} = -\kappa\, \alpha\, e^{\theta} + \mu, \tag{28a}$$

$$\dot{\theta} = \alpha\, e^{\theta} - \theta \tag{28b}$$

are the set of dimensionless equations, where α is the intermediate chemical concentration and θ represents a temperature rise. We consider reactant concentration measure $\mu = 0.1$ and reaction rate constant $\kappa = 0.07$ for which this system of equations has a stable limit cycle.

To test our algorithm, we consider as input 100 trajectories shown in Fig. 8(a). There are a total of 16 000 data points (160 for each trajectory of length $T = 0.001$). We find that SymANNTEx identifies

$$\dot{\alpha} = -0.07\, \alpha\, e^{0.9\,\theta}\, \frac{1.014 + 0.171\,\theta}{\theta^{0.1}\, e^{0.1\,\alpha}} + 0.1, \tag{29a}$$

$$\dot{\theta} = \alpha\, e^{0.9\,\theta}\, \frac{1.028 + 0.175\,\theta + 0.007\,\theta^2}{\theta^{0.1}\, e^{0.1\,\alpha}} - \theta \tag{29b}$$

using $L = 1$ layer and $K = 2$ stacks with 68 trainable parameters. We note from Eqs. (28a) and (28b) that stacking is necessary because the exponential term e^{θ} is multiplied with a state α, which cannot be exactly formed using a single stack. This model is identified in 6400 epochs on training data $\boldsymbol{Y}$ at the input data points $\boldsymbol{X}$ with a relative-RMSE of ≈1.56%. For reference, the ground truth model gives an error of ≈0.44% in Table III. The differences in Eqs. (29a) and (29b) compared to Eqs. (28a) and (28b) are terms with coefficients that are an order of magnitude smaller, which are even smaller when the range of $\boldsymbol{X}$ ($\alpha \in (0, 0.7]$, $\theta \in (0, 7]$) is considered. We see from Fig. 8(a) that the phase portraits are very similar to the ground truth model, including the dynamic range of oscillations. Although there is a slight mismatch in the fast transients, it is a range of state-space outside of $\boldsymbol{X}$ and on a faster timescale than $\boldsymbol{Y}$. From Fig. 8(b), we can also see that the time-period and the dynamic range of oscillations are also nearly reproduced. We remark that the fast and slow transients are also reproduced closely. This also shows our algorithm's ability to identify datasets with large separation of timescales. SymANNTEx estimated similar models up to noise levels of $\sigma_1 = 0.001, \sigma_2 = 0.01$.

Below, we also show another model identified for the same dataset but with a worse AIC score,

$$\dot{\alpha} = -0.075\, \alpha\, e^{\theta} + 0.167, \tag{30a}$$

$$\dot{\theta} = 1.016\, \alpha\, e^{\theta} - \theta. \tag{30b}$$

At first glance, these equations look more similar to the ground truth model but in comparison with Eqs. (29a) and (29b), they have larger relative-RMSE, and, as seen from Fig. 9, the trajectory they generate is not as close to that of the ground truth model.

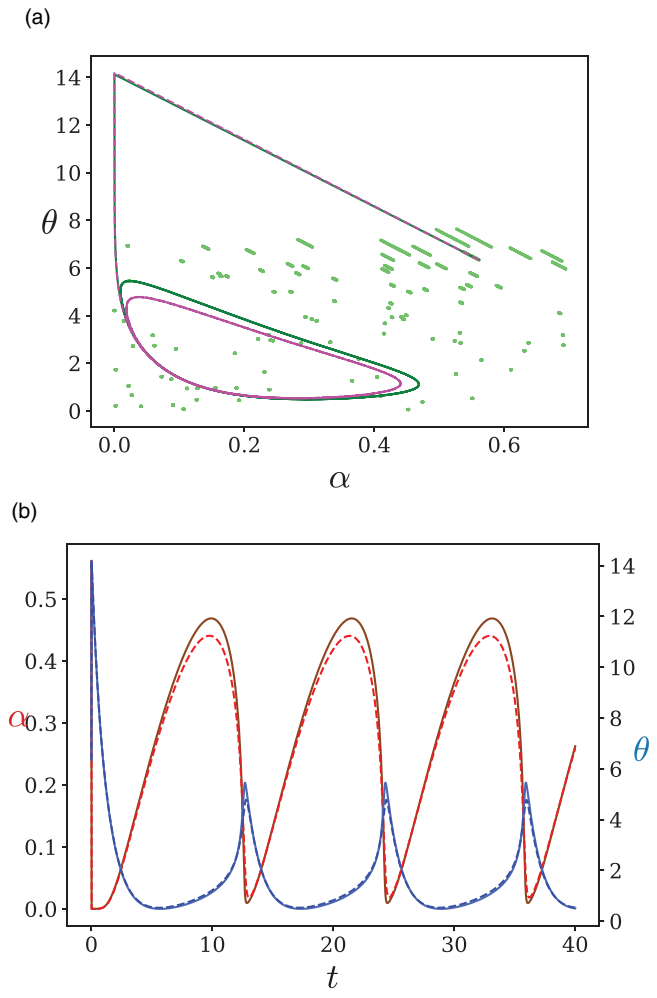

FIG. 8. (a) State-space comparison of trajectory generated (dashed magenta) by Eqs. (29a) and 29b) estimated at $\boldsymbol{X}$ (light green) with that generated by the ground truth model [Eqs. (28a) and (28b)] (solid green). The initial transient of the trajectory onto the periodic orbit happens very quickly and is reproduced almost exactly. The periodic orbit itself is similar, capturing that the dynamic range in the two states differs nearly by an order of magnitude. (b) Evolution of the same trajectories in time shows how the transient and the time-period of the identified model (dashed) is almost exactly the same as that of the ground truth model (solid). (a) Phase portrait. (b) Time series.

G. Chua double-scroll attractor

Last, we demonstrate the estimation of a model for a dynamical system that has functions which are not included in our operational layer. For this purpose, we consider the Chua double-scroll attractor,[45]

$$\dot{x} = 15.6y - \frac{31.2}{7}x + 3.343(|x+1| - |x-1|), \tag{31a}$$

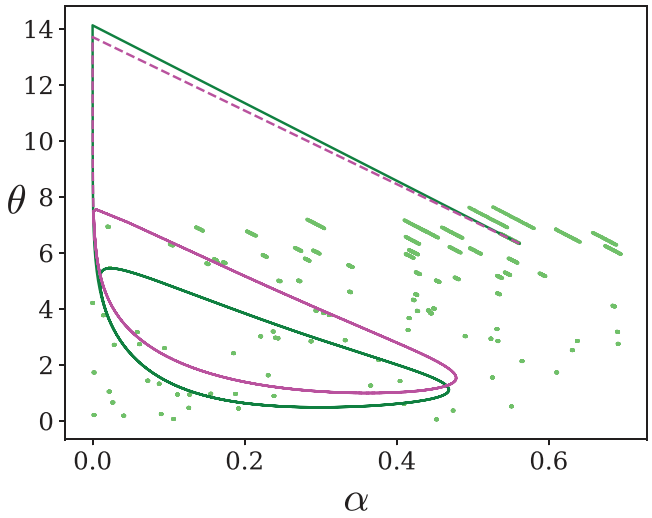

FIG. 9. State-space trajectory generated by Eqs. (30a) and (30b) (dashed magenta) compared to that generated by Eqs. (28a) and (28b) (solid green). Although the equations are visually similar, Eqs. (29a) and (29b) are a better approximation to the ground truth model in terms of the RMSE over $\boldsymbol{X}$ and the trajectories generated.

$$\dot{y} = x - y + z, \tag{31b}$$

$$\dot{z} = -28y. \tag{31c}$$

The circuit modeled by the above equations shows chaotic behavior that has been observed by an analog oscilloscope. It has one nonlinear element, which is a piecewise-linear resistor. This nonlinearity is modeled by the absolute-valued function of state as $|x+1| - |x-1|$, which is not a differentiable function of state. To test our algorithm, we again consider 1000 data points from a single randomly initialized trajectory (equally spaced in time on the time interval [0, 100]) as shown in Fig. 10(a). We use $L = 10$ layers and $K = 1$ stack with 322 trainable parameters. Without including the absolute-value function in the operational sublayers, we know that this model cannot be identified exactly. We find that most of the identified models consist of one or more sinusoidal terms, such as

$$\dot{x} = 15.857\, y - \frac{1}{4}x + 2.143\, \sin(2\,x), \tag{32a}$$

$$\dot{y} = x - y + z, \tag{32b}$$

$$\dot{z} = -28.667\, y. \tag{32c}$$

This model is identified in 6400 epochs on training data $\boldsymbol{Y}$ at the input data points $\boldsymbol{X}$ with a relative-RMSE of ≈5.41%. For reference, the ground truth model gives an error of ≈1.83% in Table III. We see from Fig. 10(c) that the selected network model gives good short-time tracking of the trajectory of the exact model, and from Fig. 10(b), they have similar long-time statistical behavior. SymANNTEx identified this model up to noise levels of $\sigma_1 = 0.01$, $\sigma_2 = 0.01$.

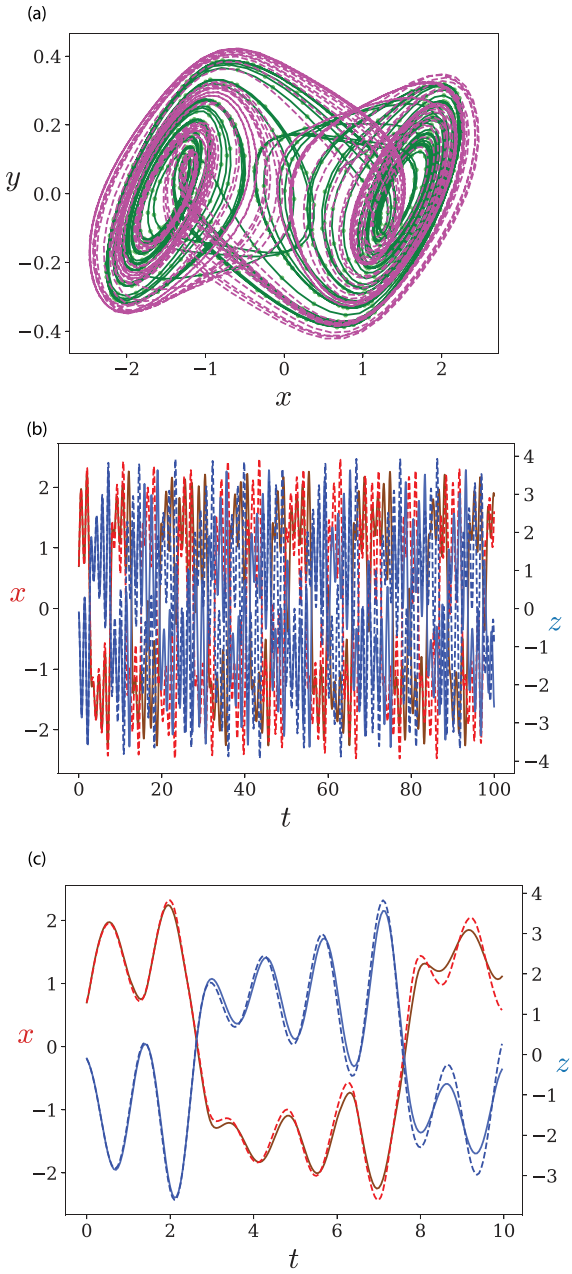

FIG. 10. (a) State-space comparison of the trajectory generated by Eqs. (32a)–(32c) (dashed magenta) estimated at $\boldsymbol{X}$ (light green) with that generated by the ground truth model [Eqs. (31a)–(31c)] (solid green). Without the absolute-valued function in our primitives, the identified model cannot give exactly the same attractor as the true model, but we can see similar dynamics and switching behavior. (b) The same trajectories plotted as a time series for x and z shows that while the solution of the identified model (dashed) diverges from that for the true model (solid), both show similar statistical behavior. Moreover, the identified model tracks the true model for the initial times as shown in (c), before gradually diverging. (a) Phase portrait. (b) Time series. (c) Zoomed-in view of (b).

V. COMPARISON WITH A L_1 REGULARIZED MEAN SQUARED ERROR AS THE LOSS

Our choice of using the mean absolute error (MAE) alongside $L_{1/2}$, L_{poly}, and L_{ops} regularization as the loss function [Eq. (16)] to be optimized differs from the common usage of L_1 regularized mean squared error (MSE) in the literature.[31] Despite working in a Euclidean space and our own metric for performance presented above being RMSE, this choice is driven by empirical observations that the loss function of MAE with custom regularization identifies models that are far more parsimonious than those identified using L_1 regularized MSE as the loss function. While this could in part be attributed to the sparsity-promoting $L_{1/2}$ regularization,[30] we found that its usage with MSE did not generate models as parsimonious as our choice of loss function. Some analysis and comparison between MAE and MSE as loss functions have been done[46] in the context of conventional DNNs without regularization. Here we summarize preliminary findings on this in Sec. V, but a detailed investigation is beyond the scope of this work. Here, we present some examples of models identified with the conventional L_1 regularized MSE. These examples are summarized in Table IV. In comparison with our custom loss function, we see that some of the identified models have far larger relative-RMSE than the training noise itself. Deep learning literature suggests that this could arise from underfitting,[47] but we can see that the models identified using SymANNTEx have a higher number of parameters than both the ground truth models and selected network models in Table IV. However, there are also models with acceptable relative-RMSE, but we can see that those models seem to have been overfitted[47] to the training data using the corresponding number of parameters, which are more than those in the ground truth models. However, we noticed that the Rössler model was consistently identified. In our numerical investigations using other combinations of error and regularization, namely, MAE with L_1 regularization, MAE with L_1, L_{poly}, L_{ops} regularizations, and MSE with L_1, L_{poly}, L_{ops} regularizations, we see trends similar with some models being identified well while others being overfitted. Thus, we avoid tabulating them in the interest of brevity and summarize only the results from L_1 regularized MSE in Table IV as it is the most commonly used one.

We remark that the AIC score, which considers the number of non-zero parameters alongside MSE, could be seen as the so-called L_0 regularization. Although we do not optimize the network parameters using this in the training process, our model selection through computing AIC scores clearly takes this into account.

VI. DISCUSSION

We presented a novel algorithm for identifying interpretable, closed-form models for a dynamical system from its time series data based on simple operations on state variables and functions. It is a generative dictionary system identification method, which only prespecifies a set of primitive operations and functions, and creates the dictionary of possible model terms from combinations and compositions of these primitives. Moreover, it is a symbolic regression algorithm, in that it searches a space of mathematical expressions to find the model, as opposed to conventional regression techniques, which optimize parameters for a fixed dictionary. These characteristics allow a much wider array of possible model terms than fixed dictionary system identification methods. Our neural network architecture is novel in its utilization of the weights of the network to determine the form of the terms in the generated dictionary, the number of which remains small enough to handle computationally, but large enough to capture a rich set of possibilities. The depth of our network, characterized by the number of stacks, represents the level of complexity obtained by composing functions. Each stack is made up of operational layers that span the breadth of the network, and which account for multiple occurrences of a function in the model but with different coefficients. The AIC score was used to select between models with parameters chosen within given tolerances. Unlike deep learning and reservoir computing approaches for forecasting a system's state,[24,48] SymANNTEx gives closed-form models for the dynamics, which could be useful for designing control algorithms. A powerful feature of our approach is that it builds upon desired primitive functions via compositions and linear combinations rather than requiring pre-specified fixed basis functions whose linear combination describes the system's dynamics. When a requisite function is absent from our chosen primitives, say, the absolute value (which is also non-differentiable), we found that SymANNTEx gives its best estimate with possible terms. Our algorithm gives accurate models even when the derivative data are noisy, but there can be a trade-off between the accuracy of the model and the number of terms that it contains. It would be interesting to

TABLE IV. Selected network models for demonstrating versatility of SymANNTEx. Note the similarity of the relative-RMSEs with those in Table III and the number of parameters in the estimated and ground truth models.

	Identified model			L_1 regularized MSE	
Example	≈ Relative-RMSE (%)	Parameters	Parameters in a ground truth model	≈ Relative-RMSE (%)	Parameters
Takens–Bogdanov	2.10	4	4	2.82	10
Simple pendulum	4.45	2	2	66.78	3
Rössler	2.60	4	4	2.62	4
Lorenz	2.59	5	5	5.09	54
FitzHugh–Nagumo	2.40	10	8	6.00	11
Chemical kinetics	1.56	11	7	0.61	48
Chua double-scroll	5.41	8	10	8.43	18

explore the use of recently proposed approaches for model selection with SymANNTEx when the data are more highly corrupted by noise.[49,50]

We found empirically that training data with a larger $\|\cdot\|_2$ norm averaged over the data points—a notion of "power" in the derivatives—leads to more accurate models with a smaller generalization error. Preliminary analysis of metrics during training indicate that data with lower power in the derivatives are prone to the occurrence of vanishing gradients.[47] Thus, as for SINDy[7] and EDMD,[16] we do not normalize the training data. This is in contrast to algorithms where normalizing training data are considered to be better practice.[47] Despite this, SymANNTEx has successfully identified the FitzHugh–Nagumo model and the model with Arrhenius terms, both of which exhibit separation of timescales.

We observed several interesting artifacts in training. We found that a loss function constituted by MAE alongside sublayer-specific regularization—some of which are non-convex, such as the $L_{1/2}$ regularization[30]—outperforms the conventional combination of MSE with L_1 regularization in estimating parsimonious models generalizable beyond seen data $\boldsymbol{X}$, even when trained for 10^5 epochs. The generalizable aspect could also be related to the recently reported double-descent phenomenon[32] where increasing the number of trainable parameters beyond the widely accepted optimal range in the bias–variance trade-off seems to promote generalizability. An example is the usage of $L = 10$ layers in identifying the Takens–Bogdanov normal form, the Lorenz, and the Rössler equations: all of these should, in principle, be identifiable with $L = 2$ layers, but we find that two layers are consistently insufficient. We also observed faster training at the edge of stability[34] with initial learning rates higher than the default in Adam.[33] For example, training using a default learning rate constant of 0.001 even for over 20 000 epochs yields residual terms in the identified Lorenz and FitzHugh–Nagumo equations that vanish in as few as 800 epochs with a learning rate constant of 0.01.

ACKNOWLEDGMENTS

This work was supported by the National Science Foundation (Grant No. NSF-2016004).

AUTHOR DECLARATIONS

Conflict of Interest

The authors have no conflicts to disclose.

Author Contributions

N. Boddupalli and T. Matchen contributed equally to this paper.

N. Boddupalli: Conceptualization (equal); Investigation (equal); Software (equal); Validation (equal); Visualization (equal); Writing – original draft (equal). **T. Matchen:** Conceptualization (equal); Investigation (equal); Software (equal); Validation (equal); Visualization (equal); Writing – review & editing (equal). **J. Moehlis:** Conceptualization (supporting); Funding acquisition (lead); Project administration (lead); Supervision (lead); Writing – review & editing (equal).

DATA AVAILABILITY

The data that support the findings of this study are available from the corresponding author upon reasonable request.

REFERENCES

[1] L. Ljung, *System Identification: Theory for the User*, 2nd ed. (Prentice Hall, Upper Saddle River, NJ, 1999).
[2] S. L. Brunton and J. N. Kutz, *Data-Driven Science and Engineering* (Cambridge University Press, Cambridge, 2019).
[3] J. P. Crutchfield and B. S. McNamara, "Equations of motion from a data series," Complex Syst. **1**, 417–452 (1987).
[4] C. Yao and E. M. Bollt, "Modeling and nonlinear parameter estimation with Kronecker product representation for coupled oscillators and spatiotemporal systems," Physica D **227**, 78–99 (2007).
[5] W.-X. Wang, R. Yang, Y.-C. Lai, V. Kovanis, and C. Grebogi, "Predicting catastrophes in nonlinear dynamical systems by compressive sensing," Phys. Rev. Lett. **106**, 154101 (2011).
[6] Y.-C. Lai, "Finding nonlinear system equations and complex network structures from data: A sparse optimization approach," Chaos **31**, 082101 (2021).
[7] S. L. Brunton, J. L. Proctor, and J. N. Kutz, "Discovering governing equations from data by sparse identification of nonlinear dynamical systems," Proc. Natl. Acad. Sci. U.S.A. **113**, 3932–3937 (2016).
[8] D. Napoletani and T. D. Sauer, "Reconstructing the topology of sparsely connected dynamical networks," Phys. Rev. E **77**, 026103 (2008).
[9] N. M. Mangan, S. L. Brunton, J. L. Proctor, and J. N. Kutz, "Inferring biological networks by sparse identification of nonlinear dynamics," IEEE Trans. Mol. Biol. Multi-Scale Commun. **2**, 52–63 (2016).
[10] K. Champion, B. Lusch, J. N. Kutz, and S. L. Brunton, "Data-driven discovery of coordinates and governing equations," Proc. Natl. Acad. Sci. U.S.A. **116**, 22445–22451 (2019).
[11] I. Mezić, "Spectral properties of dynamical systems, model reduction and decompositions," Nonlinear Dyn. **41**, 309–325 (2005).
[12] M. Budisic, R. Mohr, and I. Mezic, "Applied Koopmanism," Chaos **22**, 047510 (2012).
[13] H. Arbabi and I. Mezic, "Ergodic theory, dynamic mode decomposition, and computation of spectral properties of the Koopman operator," SIAM J. Appl. Dyn. Syst. **16**, 2096–2126 (2017).
[14] P. J. Schmid, "Dynamic mode decomposition of numerical and experimental data," J. Fluid Mech. **656**, 5–28 (2010).
[15] J. N. Kutz, S. L. Brunton, B. W. Brunton, and J. L. Proctor, *Dynamic Mode Decomposition: Data-Driven Modeling of Complex Systems* (SIAM, Philadelphia, PA, 2016).
[16] M. O. Williams, I. G. Kevrekidis, and C. W. Rowley, "A data-driven approximation of the Koopman operator: Extending dynamic mode decomposition," J. Nonlinear Sci. **25**, 1307–1346 (2015).
[17] J. Bongard and H. Lipson, "Automated reverse engineering of nonlinear dynamical systems," Proc. Natl. Acad. Sci. U.S.A. **104**, 9943 (2007).
[18] M. Quade, M. Abel, K. Shafi, R. K. Niven, and B. R. Noack, "Prediction of dynamical systems by symbolic regression," Phys. Rev. E **94**, 012214 (2016).
[19] M. Quade, J. Gout, and M. Abel, "Glyph: Symbolic regression tools," J. Open Res. Softw. **7**, 19 (2019).
[20] M. Schmidt and H. Lipson, "Distilling free-form natural laws from experimental data," Science **324**, 81–85 (2009).
[21] W. Banzhaf and W. B. Langdon, "Some considerations on the reason for bloat," Genet. Program. Evolvable Mach. **3**, 81–91 (2002).
[22] A. Lapedes and R. Farber, "How neural nets work," in *Neural Information Processing Systems*, edited by D. Z. Anderson (American Institute of Physics, 1988), pp. 442–456.
[23] R. Rico-Martinez, K. Krischer, I. G. Kevrekidis, M. C. Kube, and J. L. Hudson, "Discrete- vs continuous-time nonlinear signal processing of Cu electrodissolution data," Chem. Eng. Commun. **118**, 25–48 (1992).

[24]M. Längkvist, L. Karlsson, and A. Loutfi, "A review of unsupervised feature learning and deep learning for time-series modeling," Pattern Recognit. Lett. **42**, 11–24 (2014).
[25]S. Kim, P. Y. Lu, S. Mukherjee, M. Gilbert, L. Jing, V. Čeperić, and M. Soljačić, "Integration of neural network-based symbolic regression in deep learning for scientific discovery," IEEE Trans. Neural Netw. Learn. Syst. **32**, 4166–4177 (2020).
[26]C. Fronk and L. Petzold, "Interpretable polynomial neural ordinary differential equations," Chaos **33**, 043101 (2023).
[27]H. Akaike, "A new look at the statistical model identification," IEEE Trans. Autom. Control **19**, 716–723 (1974).
[28]F. Van Breugel, Y. Liu, B. W. Brunton, and J. N. Kutz, "PyNumDiff: A Python package for numerical differentiation of noisy time-series data," J. Open Source Softw. **7**, 4078 (2022).
[29]J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, J. Cai, and T. Chen, "Recent advances in convolutional neural networks," Pattern Recognit. **77**, 354–377 (2018).
[30]Z. Xu, H. Zhang, Y. Wang, X. Chang, and Y. Liang, "L 1/2 regularization," Sci. China Inf. Sci. **53**, 1159–1169 (2010).
[31]M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of Machine Learning* (MIT Press, Cambridge, MA, 2018).
[32]M. Belkin, D. Hsu, S. Ma, and S. Mandal, "Reconciling modern machine-learning practice and the classical bias–variance trade-off," Proc. Natl. Acad. Sci. U.S.A. **116**, 15849–15854 (2019).
[33]D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proceedings of the 3rd International Conference on Learning Representations (ICLR)* (2015).
[34]J. Cohen, S. Kaur, Y. Li, J. Z. Kolter, and A. Talwalkar, "Gradient descent on neural networks typically occurs at the edge of stability," in *Proceedings of the 9th International Conference on Learning Representations (ICLR)* (2021).
[35]N. M. Mangan, J. N. Kutz, S. L. Brunton, and J. L. Proctor, "Model selection for dynamical systems via sparse regression and information criteria," Proc. R. Soc. A **473**, 20170009 (2017).
[36]K. Burnham and D. Anderson, *Model Selection and Multimodel Inference*, 2nd ed. (Springer, New York, 2002).
[37]J. Guckenheimer and P. Holmes, *Nonlinear Oscillations, Dynamical Systems, and Bifurcations of Vector Fields* (Springer, New York, 1983).
[38]I. Mezić, "Spectrum of the Koopman operator, spectral expansions in functional spaces, and state-space geometry," J. Nonlinear Sci. **30**, 2091–2145 (2020).
[39]B. Lusch, J. N. Kutz, and S. L. Brunton, "Deep learning for universal linear embeddings of nonlinear dynamics," Nat. Commun. **9**, 4950 (2018).
[40]O. E. Rössler, "Chaotic behavior in simple reaction systems," Z. Naturforsch. A **31**, 259–264 (1976).
[41]E. N. Lorenz, "Deterministic nonperiodic flow," J. Atmos. Sci. **20**, 130–141 (1963).
[42]E. Bollt, "On explaining the surprising success of reservoir computing forecaster of chaos? The universal machine learning dynamical system with contrasts to VAR and DMD," Chaos **31**, 013108 (2021).
[43]R. FitzHugh, "Impulses and physiological states in theoretical models of nerve membrane," Biophys. J. **1**, 445–466 (1961).
[44]P. Gray and S. K. Scott, *Chemical Oscillations and Instabilities* (Oxford University Press, Oxford, 1990).
[45]L. Chua, M. Komuro, and T. Matsumoto, "The double scroll family," IEEE Trans. Circuits Syst. **33**, 1072–1118 (1986).
[46]J. Qi, J. Du, S. M. Siniscalchi, X. Ma, and C.-H. Lee, "On mean absolute error for deep neural network based vector-to-vector regression," IEEE Signal Process. Lett. **27**, 1485–1489 (2020).
[47]I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning* (MIT Press, Cambridge, MA, 2016).
[48]D. J. Gauthier, E. Bollt, A. Griffith, and W. A. Barbosa, "Next generation reservoir computing," Nat. Commun. **12**, 5564 (2021).
[49]G. Tran and R. Ward, "Exact recovery of chaotic systems from highly corrupted data," SIAM J. Multiscale Model. Simul. **15**, 1108–1129 (2017).
[50]A. A. R. AlMomani, J. Sun, and E. Bollt, "How entropic regression beats the outliers problem in nonlinear systems identification," Chaos **30**, 013107 (2020).