From data to dynamics: discovering governing equations from data

Kathleen Champion

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2019

Reading Committee:

J. Nathan Kutz, Chair

Steven L. Brunton

Eric Shea-Brown

Program Authorized to Offer Degree:
Applied Mathematics

University of Washington

Abstract

From data to dynamics: discovering governing equations from data

Kathleen Champion

Chair of the Supervisory Committee:
Professor J. Nathan Kutz
Applied Mathematics

Governing laws and equations, such as Newton's second law for classical mechanics and the Navier-Stokes equations thence derived, have been responsible throughout history for numerous scientific breakthroughs in the physical and engineering sciences. There are many systems of interest for which large quantities of measurement data have been collected, but the underlying governing equations remain unknown. While machine learning approaches such as sparse regression and deep neural networks have been successful at discovering governing laws and reduced models from data, many challenges still remain. In this work, we focus on the discovery of nonlinear dynamical systems models from data. We present several methods based on the sparse identification of nonlinear dynamics (SINDy) algorithm. These approaches address a number of challenges that occur when dealing with scientific data sets, including unknown coordinates, multiscale dynamics, parametric dependencies, and outliers. Our methods focus on discovering parsimonious models, as parsimony is key for obtaining models that have physical interpretations and can generalize to predict previously unobserved behaviors.

TABLE OF CONTENTS

LIST OF FIGURES

Figure Number Page

LIST OF TABLES

ACKNOWLEDGMENTS

I am deeply thankful to Nathan Kutz and Steve Brunton. In addition to their endlessly helpful research guidance, the personal and professional support and encouragement they provide truly sets them apart as advisors. Working with them has been a blast, and I'm forever grateful to them for making my graduate experience so fun and enriching.

A big thanks also goes out to the other members of my committee, Eric Shea-Brown and Andy Stewart. I'm very appreciative of the thought-provoking questions and discussion at my general and final exams. I'm particularly thankful to Eric for being there throughout my time in graduate school with an enthusiastic show of support. I'd also like to thank my collaborators, particularly Bethany Lusch, Peng Zheng, and Sasha Aravkin. The work we've done together has become an integral part of my thesis, and I had a great time collaborating with all of them.

My graduate school experience would not have been the same without the amazing friends I met. I'm so grateful to have had such a great cohort, particularly my wonderful friends and officemates Brian de Silva, Andreas Freund, and Jeremy Upsal. I owe so much to Emily Dinan and Ellie Levey. They are always there for me, and it's been great having friends I can relate to in so many ways, including being women in math. I'm forever thankful to Will Lowrey for being the most enthusiastic cheerleader I could ever ask for.

Finally I'd like to thank my parents, Terrence and Michelle Champion, for all their love and support. They've always been there to cheer me on through my successes and to listen patiently when I'm stressed. I'm grateful to them for supporting my education through so many phases and not giving up on me in elementary school when I expressed skepticism about the order of operations.

My graduate research was supported by the National Science Foundation Graduate Research Fellowship Program (Grant No. DGE-1256082), an NIH Computational Neuroscience Training Grant, a Seattle ARCS Foundation Fellowship (sponsored by the Washington Research Foundation), and a Boeing Fellowship.

DEDICATION

to mom and dad

Chapter 1

INTRODUCTION

As human knowledge advances, we become ever better in our ability to model and predict complex phenomena observed in the world around us. From the understanding of astronomical phenomena to the engineering of advanced technologies such as rockets and airplanes, mathematical modeling has vastly improved our ability to understand and adapt to the world around us. However, there are still myriad phenomena we would like to better understand: how the brain gives rise to consciousness, how biological factors such as genetics and protein folding are linked to disease, and how we can improve predictions of weather and climate. Physical and biological systems are exceedingly complex, and while significant progress has been made we still lack unified frameworks for modeling and predicting the behaviors of many systems of interest.

Advances in sensor technology have led to a massive influx of measurement data for systems across the physical, biological, and engineering sciences. The collection of measurement data has outpaced our ability to parse and interpret such data. This has led to an explosive interest in the study of data science, focused on methods for handling and drawing conclusions from large amounts of data. A large focus of research is on methods that can pull out mathematical models and understanding from this abundance of data. Many early mathematical frameworks, such as Newtonian mechanics and Kepler's laws of planetary motion, came from collecting measurements and developing a mathematical model that fit the collected data. These data collections, however, were still at scales that enabled human comprehension. As the amount of data and complexity of the systems we seek to understand increase, we need to develop more advanced methods for learning models from data.

Many complex systems of interest can be viewed from a dynamical systems perspective.

Specifically, many systems are *dynamic*: that is, the state of the system is constantly changing, and we would like to develop an understanding of how this state changes. A *dynamical systems* model represents the state of a system as a vector of numbers and defines evolution rules for how the state evolves in time. The state could be, for example, the electrical potentials of neurons in the brain or ocean temperatures at a set of locations around the globe. The evolution rules might describe how neurons interact or how temperature changes propagate around the ocean. Even data such as social networks or user movie preferences can be modeled as dynamical systems, as users' social connections and tastes evolve over time. Dynamical systems is a field with a rich history of study, and the perspective gained from this study can help us understand systems in a wide range of application areas.

The combination of abundant dynamic data and well-developed theory for dynamical systems motivates methods for identifying dynamical systems models from data. Specifically, the goal is to take measurements of the states of a system at various points in time and find a set of evolution rules, or governing equations, that explains how the states change. There are several motivations for applying this type of approach depending on the field or system of study. For some systems, such as the brain, we do not know a set of governing equations and would like to gain scientific insight by discovering these equations. In other cases, we may have only partial knowledge of governing equations and want to fill in missing equations or identify unknown parameter values. For many problems in engineering applications, on the other hand, a full set of governing equations is known but we would like to identify reduced models that allow for efficient simulation and future state prediction. These applications may all benefit from methods for identifying dynamical systems models from data.

Machine learning (ML) is a broad class of approaches for turning data into models. ML methods can be used to learn models of many different forms and have been applied widely across applications. To create a machine learning model, we define a parameterized model form and set up an optimization problem to learn the parameters that best fit a set of training data. The specificity of the model form ranges widely depending on the approach: for example regression may be used to fit simple interpretable models of a prescribed form

(e.g. polynomial fitting), whereas deep neural networks can parameterize more complex and general functions. The problem of discovering governing equations can also be viewed from a machine learning perspective, using ML-inspired approaches to learn dynamical systems models.

Unlike many applications in machine learning, which are focused solely on prediction, science and engineering aim for *interpretation*, *generalization*, and *extrapolation*. A key feature of many scientific governing models is that they have only a few active terms, and the terms in the model have physical interpretations. In addition, these models can be generalized to new parameter regimes and extrapolate to predict previously unobserved behaviors, beyond where data was collected. Parsimony is key for discovering models that are interpretable, as we can keep track of the terms in the model, and generalizable, as the models are encouraged to avoid overfitting. Parsimonious models strike the balance between model efficiency and descriptive capabilities, having the fewest terms required to capture essential interactions.

In this work we introduce a number of approaches for discovering dynamical systems models from data. We place a strong emphasis on the discovery of models that are nonlinear and *parsimonious*, having only a few active terms in the dynamics. Machine learning for model discovery in physics, biology, and engineering is of growing importance for characterizing complex systems. Our approaches address several challenges associated with modeling scientific data: unknown coordinates, multiscale dynamics, parametric dependencies, and outliers.

1.1 Organization

This thesis introduces several approaches that address challenges associated with identifying governing dynamical systems models from data. In Chapter 2 we provide background on data-driven methods for modeling dynamical systems. We first cover standard approaches for building linear models of dynamics. We then discuss the sparse identification of nonlinear dynamical systems (SINDy) method, which we will extend and build upon in successive chapters. Chapters 3 and 4 address challenges associated with multiscale systems. Chap-

ter 3 provides an efficient sampling method for SINDy that enables it to scale for studying multiscale systems. Chapter 4 discusses strategies for applying Hankel alternative view of Koopman (HAVOK), a related method for identifying dynamics from data, to multiscale systems. In Chapter 5, we address systems with unknown coordinates. In particular, we introduce an approach that extracts parsimonious nonlinear governing equations from high-dimensional data by performing a simultaneous discovery of reduced coordinates and associated dynamical models. In Chapter 6, we propose a unified optimization framework for SINDy. This framework, based on the sparse relaxed regularized regression (SR3) algorithm, allows the use of non-convex sparsity promoting regularization functions and can be adapted to address key challenges in scientific problems and data sets, including outliers, parametric dependencies, and physical constraints.

Chapter 2

BACKGROUND

Complex systems in many application areas can be studied and modeled through the lens of dynamical systems. In this work, we consider dynamical systems of the form

$$\frac{d}{dt}\mathbf{x}(t) = \mathbf{f}(\mathbf{x}(t)). \tag{2.1}$$

Here $\mathbf{x}(t) \in \mathbb{R}^n$ is the state of the system at time t and $\mathbf{f}$ defines the evolution rules for the system, or how the states $\mathbf{x}$ evolve in time. Depending on the application, the state variables could represent any number of properties of the system: for example, membrane potentials of a set of neurons, velocities of a fluid flow, or concentrations of chemical substances. The function $\mathbf{f}$ describes how these states interact and change in time.

As a simple example of a dynamical system we can consider the nonlinear pendulum. The behavior of a nonlinear pendulum is described by the second order differential equation

$$\ddot{x} + \frac{g}{\ell}\sin x = 0,$$

which can be derived using classical mechanics. Here $x(t)$ is the angular displacement of the pendulum at time t, g is the gravitational constant, and ℓ is the length of the pendulum. The pendulum equation can be rewritten as the dynamical system

$$\begin{aligned} \dot{x} &= y \\ \dot{y} &= -\frac{g}{\ell}\sin x \end{aligned}$$

where $y(t)$ is the angular velocity of the pendulum. Given an initial position $x(0)$ and velocity $y(0)$, this system describes the movement of the pendulum in time. Using dynamical systems analysis, one can study how the initial states $x(0), y(0)$ and/or the parameters (e.g. the length of the pendulum) affect the behavior.

Standard approaches for studying dynamical systems methods assume we know the form of the governing equations $\mathbf{f}$ and want to predict the evolution of the states $\mathbf{x}$. However, it is often the case that measurements of the states $\mathbf{x}$ are available but the form of $\mathbf{f}$ is unknown. This provides the motivation for methods that seek to discover the form of $\mathbf{f}$ from measurements of the states. Learning the dynamical equations enables studying and predicting the behavior of the system using standard dynamical systems approaches. Discovering governing equations also has the potential to further scientific understanding through the development of new laws and models of behavior. In addition, knowing $\mathbf{f}$ enables predicting the state evolution for previously unobserved parameter regimes or initial conditions and can improve our ability to control a system. This work will address this key challenge in the study of dynamical systems: discovering governing equations, or finding the form of $\mathbf{f}$, given measurements of $\mathbf{x}(t)$.

2.1 Machine learning for discovering models from data

Broadly defined, machine learning (ML) can be viewed as a means of learning models from data. While the process of fitting models to data has been used throughout history, advances in data collection, computing resources, and machine learning theory have allowed ML approaches to fit more and more general classes of models. This has lead to its broad success in tasks such as image classification [42] and speech recognition [29]. As a result of this success, ML approaches are increasingly being adapted to applications in data-intensive fields of science and engineering. In addition to standard techniques in clustering and classification, ML is now being used to discover models that characterize and predict the behavior of physical systems. While extremely promising, applying ML to problems in the physical sciences comes with a unique set of challenges: scientists want physically interpretable models that can (i) generalize to predict previously unobserved behaviors, (ii) provide effective forecasting predictions (extrapolation), and (iii) be certifiable.

The problem of identifying dynamical systems from data can be framed as an ML problem: essentially, the goal is to find a model that predicts $\dot{\mathbf{x}}$ from $\mathbf{x}$. To set this up as an ML

problem, one would define a general parameterized model form h_θ, where θ is an associated set of parameters (θ could be, for example, the weights and biases of a neural network). Given data in the form of samples $(\mathbf{x}, \dot{\mathbf{x}})$, one would then try to learn the best choice of parameters $\hat{\theta}$ to approximately solve $\dot{\mathbf{x}} = h_\theta(\mathbf{x})$.

Clearly the choice of the general model form h_θ has a significant impact on the predictive ability of the resulting models and the conclusions thence drawn. Given the success of deep neural networks (DNNs), an obvious choice might be to fit a DNN to predict $\dot{\mathbf{x}}$ from $\mathbf{x}$. While neural networks have been studied for decades in dynamical systems [26, 58] and have had a recent resurgence for modeling time-series data in general [55, 84, 85, 94, 79, 52, 67, 68, 5, 62], they typically struggle with extrapolation and are highly parameterized, making them difficult to interpret. Many data-driven approaches for modeling dynamical systems have instead sought more generalizable and interpretable models by using dynamical systems theory to inform the choice of model form. In the following sections we discuss two classes of such strategies. The first focuses on building linear dynamical models, which are well studied. The second relies on parsimony to build interpretable nonlinear models.

2.2 Koopman theory and dynamic mode decomposition

A large body of research on modeling dynamical systems from data focuses on building linear dynamical models. While physical systems are typically nonlinear, linear models provide many advantages, as linear systems are well understood: their behavior can be predicted analytically and there is a comprehensive set of control techniques framed around linear systems. Much of the theory for building linear models of nonlinear systems is based on Koopman analysis, an emerging data-driven modeling tool for dynamical systems. First proposed in 1931 [41], Koopman analysis has experienced a recent resurgence in interest and development [56, 13, 57]. The basic idea behind Koopman analysis is to trade finite dimensional nonlinear dynamics for linear dynamics in an infinite dimensional space. In practical applications, one seeks finite dimensional approximations to the infinite dimensional linear operator known as the Koopman operator.

To introduce Koopman analysis, we assume we are working with a discrete-time version of the dynamical system in (2.1):

$$\mathbf{x}_{k+1} = \mathbf{F}(\mathbf{x}_k) = \mathbf{x}_k + \int_{k\Delta t}^{(k+1)\Delta t} \mathbf{f}(\mathbf{x}(\tau))\mathrm{d}\tau. \tag{2.2}$$

The Koopman operator $\mathcal{K}$ is an infinite-dimensional linear operator on the Hilbert space of measurement functions of the states $\mathbf{x}_k$. Given measurements $g(\mathbf{x}_k)$, the Koopman operator is defined by

$$\mathcal{K}g \triangleq g \circ \mathbf{F} \quad \Rightarrow \quad \mathcal{K}g(\mathbf{x}_k) = g(\mathbf{x}_{k+1}). \tag{2.3}$$

Thus the Koopman operator maps the system forward in time in the space of measurements.

While having a linear representation of the dynamical system is advantageous, the Koopman operator is infinite-dimensional and obtaining finite-dimensional approximations is difficult in practice. In order to obtain a good model, one seeks a set of measurements that form a Koopman invariant subspace [11]. Dynamic mode decomposition (DMD)[75] is one well-known method for approximating the Koopman operator [70, 83, 44]. DMD constructs a linear mapping satisfying

$$\mathbf{x}_{k+1} \approx \mathbf{A}\mathbf{x}_k.$$

However as might be expected, DMD does not perform well for strongly nonlinear systems. Extended DMD and kernel DMD are two methods that seek to resolve this issue by constructing a library of nonlinear measurements of $\mathbf{x}_k$ and finding a linear operator that works on these measurements [88, 89]. However these methods can be computationally expensive, and it is not guaranteed that the selected measurements will form a Koopman invariant subspace [11] unless results are rigorously cross-validated, as in the equivalent variational approach of conformation dynamics (VAC) approach [59, 60]. Alternatively, one can find judicious choices of the nonlinear measurements that transform the underlying dynamics from a strongly nonlinear system to a weakly nonlinear system [45]. A review of the DMD algorithm and its many applications can be found in Ref. [44]

2.3 Sparse identification of nonlinear dynamics (SINDy)

The methods discussed in Section 2.2 are based on the discovery of *linear* dynamical models. While linear dynamical systems have several nice properties, there are many rich and interesting dynamical behaviors that cannot be completely captured by a linear model. This motivates approaches for discovering *nonlinear* dynamical systems models.

A breakthrough approach used symbolic regression to identify the form of nonlinear dynamical systems and governing laws from data [7, 77, 16]. This works remarkably well for discovering interpretable physical models, balancing accuracy with model complexity. However, symbolic regression is computationally expensive and can be difficult to scale to large problems.

A related approach, which will be the focus throughout this work, is the *sparse identification of nonlinear dynamics* (SINDy) [12] algorithm. SINDy is a regression technique for extracting parsimonious dynamics from time-series data. The method takes snapshot data $\mathbf{x}(t) \in \mathbb{R}^n$ and attempts to discover a best-fit dynamical system of the form (2.1). SINDy seeks a parsimonious model for the dynamics, resulting in a function $\mathbf{f}$ that contains only a few active terms: it is sparse in a basis of possible functions. This is consistent with extensive knowledge of a diverse set of evolution equations used throughout the physical, engineering, and biological sciences. Thus, the types of functions that comprise $\mathbf{f}$ are typically known from modeling experience.

SINDy frames model discovery as a sparse regression problem. If snapshot derivatives are

available, or can be calculated from data, the snapshots are stacked to form data matrices

$$\mathbf{X} = \begin{pmatrix} x_1(t_1) & x_2(t_1) & \cdots & x_n(t_1) \\ x_1(t_2) & x_2(t_2) & \cdots & x_n(t_2) \\ \vdots & \vdots & \ddots & \vdots \\ x_1(t_m) & x_2(t_m) & \cdots & x_n(t_m) \end{pmatrix}, \tag{2.4}$$

$$\dot{\mathbf{X}} = \begin{pmatrix} \dot{x}_1(t_1) & \dot{x}_2(t_1) & \cdots & \dot{x}_n(t_1) \\ \dot{x}_1(t_2) & \dot{x}_2(t_2) & \cdots & \dot{x}_n(t_2) \\ \vdots & \vdots & \ddots & \vdots \\ \dot{x}_1(t_m) & \dot{x}_2(t_m) & \cdots & \dot{x}_n(t_m) \end{pmatrix}.$$

with $\mathbf{X}, \dot{\mathbf{X}} \in \mathbb{R}^{m \times n}$. Although $\mathbf{f}$ is unknown, we can construct an extensive library of p candidate functions $\boldsymbol{\Theta}(\mathbf{X}) = [\boldsymbol{\theta}_1(\mathbf{X}) \cdots \boldsymbol{\theta}_p(\mathbf{X})] \in \mathbb{R}^{m \times p}$, where each $\boldsymbol{\theta}_j$ is a candidate model term. We assume $m \gg p$ so the number of data snapshots is larger than the number of library functions; it may be necessary to sample transients and multiple initial conditions to improve the condition number of $\boldsymbol{\Theta}$. The choice of basis functions typically reflects some knowledge about the system of interest: a common choice is polynomials in $\mathbf{x}$ as these are elements of many canonical models. Another interpretation is that the SINDy algorithm discovers the dominant balance dynamics of the measured system, which is often of a polynomial form due to a Taylor expansion of a complicated nonlinear function. The library is used to formulate an overdetermined linear system

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

where the unknown matrix $\boldsymbol{\Xi} = (\boldsymbol{\xi}_1\ \boldsymbol{\xi}_2\ \cdots\ \boldsymbol{\xi}_n) \in \mathbb{R}^{p \times n}$ is the set of coefficients that determine the active terms from $\boldsymbol{\Theta}(\mathbf{X})$ in the dynamics $\mathbf{f}$. Sparsity-promoting regression is used to solve for $\boldsymbol{\Xi}$ that result in parsimonious models, ensuring that $\boldsymbol{\Xi}$, or more precisely each $\boldsymbol{\xi}_j$, is sparse and only a few columns of $\boldsymbol{\Theta}(\mathbf{X})$ are selected. For high-dimensional systems, the goal is to identify a low-dimensional state $\mathbf{z} = \varphi(\mathbf{x})$ with dynamics $\dot{\mathbf{z}} = \mathbf{g}(\mathbf{z})$, as in (5.1). The standard SINDy approach uses a sequentially thresholded least squares algorithm to find the coefficients [12], which is a proxy for ℓ_0 optimization [96] and has convergence

guarantees [95]. Yao and Bollt [93] previously formulated system identification as a similar linear inverse problem without including sparsity, resulting in models that included all terms in $\boldsymbol{\Theta}$. In either case, an appealing aspect of this model discovery formulation is that it results in an overdetermined linear system for which many regularized solution techniques exist. Thus, it provides a computationally efficient counterpart to other model discovery frameworks [77]. Optimization approaches for SINDy are discussed further in Chapter 6.

Chapter 3

MULTISCALE SYSTEMS: SAMPLING STRATEGIES FOR SINDY

Many complex systems of interest exhibit behavior across multiple time scales, which poses unique challenges for modeling and predicting their behavior. It is often the case that while we are primarily interested in macroscale phenomena, the microscale dynamics must also be modeled and understood, as they play a role in driving the macroscale behavior. The macroscale dynamics in turn drive the microscale dynamics, thus producing a coupling whereby the dynamics at different time scales feedback into each other. This can make dealing with multiscale systems particularly difficult unless the time scales are disambiguated in a principled way. There is a significant body of research focused on modeling multiscale systems: notably the heterogeneous multiscale modeling (HMM) framework and equation-free methods for linking scales [39, 87, 86]. Additional work has focused on testing for the presence of multiscale dynamics so that analyzing and simulating multiscale systems is more computationally efficient [21, 22]. Many of the same issues that make modeling multiscale systems difficult can also present challenges for model discovery and system identification. This motivates the development of specialized methods for performing model discovery on problems with multiple time scales, taking into account the unique properties of multiscale systems.

In this chapter we investigate the performance of the SINDy algorithm (introduced in Chapter 2) on multiscale systems. In Section 3.1 we establish a baseline understanding of the performance of SINDy on systems with a single timescale. We show in Section 3.2 that for a system with fast and slow time scales, if data is sampled uniformly in time the amount of data required to identify the system scales with the frequency separation of time scales.

This motivates a more efficient sampling method for applying SINDy to multiscale systems. We introduce a sampling strategy for multiscale systems that involves collecting short bursts of samples at a high sampling rate, which means that the data captures elements of both the slow and fast time scales. This method allows SINDy to scale efficiently for systems with multiple time scales.

3.1 Sampling requirements for uniscale dynamical systems

Before considering multiscale systems, we establish a baseline understanding of the data requirements of the SINDy algorithm. While previous results have assessed the performance of SINDy in various settings [53, 37], so far none have looked explicitly at how much data is necessary to correctly identify a system. We determine the data requirements of SINDy on four example systems: the periodic Duffing and Van der Pol oscillators, and the chaotic Lorenz and Rossler systems. To assess how quickly SINDy can correctly identify a given system, we look at its performance on measurement data from a single trajectory. Each example system has dynamics that evolve on an attractor. We choose data from time points after the system has converged to the attractor and quantify the sampling rate and duration in relation to the typical time T it takes the system to make a trip around some portion of the attractor. We refer to this as a "period" of oscillation. This gives us a standard with which to compare the data requirements among multiple systems. We show that in all four models considered, SINDy can very rapidly discover the underlying dynamics of the system, even if sampling only a fraction of a period of the attractor or oscillation.

Our procedure is as follows. For all systems, we simulate a single trajectory at a sampling rate $r = 2^{18}$ samples/period, giving us a time step $\Delta t = T/r$ (note T is defined differently for each system). We then subsample the data at several rates $r_{\text{sample}} = 2^5, 2^6, \ldots, 2^{18}$ and durations on different portions of the attractor. For each set of subsampled data, we train a SINDy model and determine if the identified model has the correct set of nonzero coefficients, meaning SINDy has identified the proper form of the dynamics. We are specifically looking for the correct structural identification of the coefficient matrix $\boldsymbol{\Xi}$, rather than precise co-

efficient values. In general we find that if SINDy correctly identifies the active coefficients in the model, the coefficients are reasonably close to the true values. Our library of SINDy candidate functions includes polynomial terms of up to order 3. In this section, we choose the coefficient threshold for the iterative least squares algorithm to be $\lambda = 0.1$. The results indicate the sampling rate and length of time one must sample for SINDy to correctly discover a dynamical system.

When assessing the performance of SINDy, it is important to consider how measurements of the derivative are obtained. While in some cases we may have measurements of the derivatives, most often these must be estimated from measurements of $\mathbf{x}$. In this work, we consider the low noise case and are thus able to use a standard center difference method to estimate derivatives. With noisy data, more sophisticated methods such as the total variation regularized derivative may be necessary to obtain more accurate estimates of the derivative [15, 12]. Indeed, accurate computations of the derivative are critical to the success of SINDy.

To assess performance, we consider the duration of sampling required to identify the system at each chosen sampling rate. The exact duration required depends on which portion of the attractor the trajectory is sampled from; thus we take data from different portions of the attractor and compute an average required duration for each sampling rate. This provides insight into how long and at what rate we need to sample in order to correctly identify the form of the dynamics. Surprisingly, in all four example systems, *data from less than a full trip around the attractor is typically sufficient for SINDy to identify the correct form of the dynamical system.*

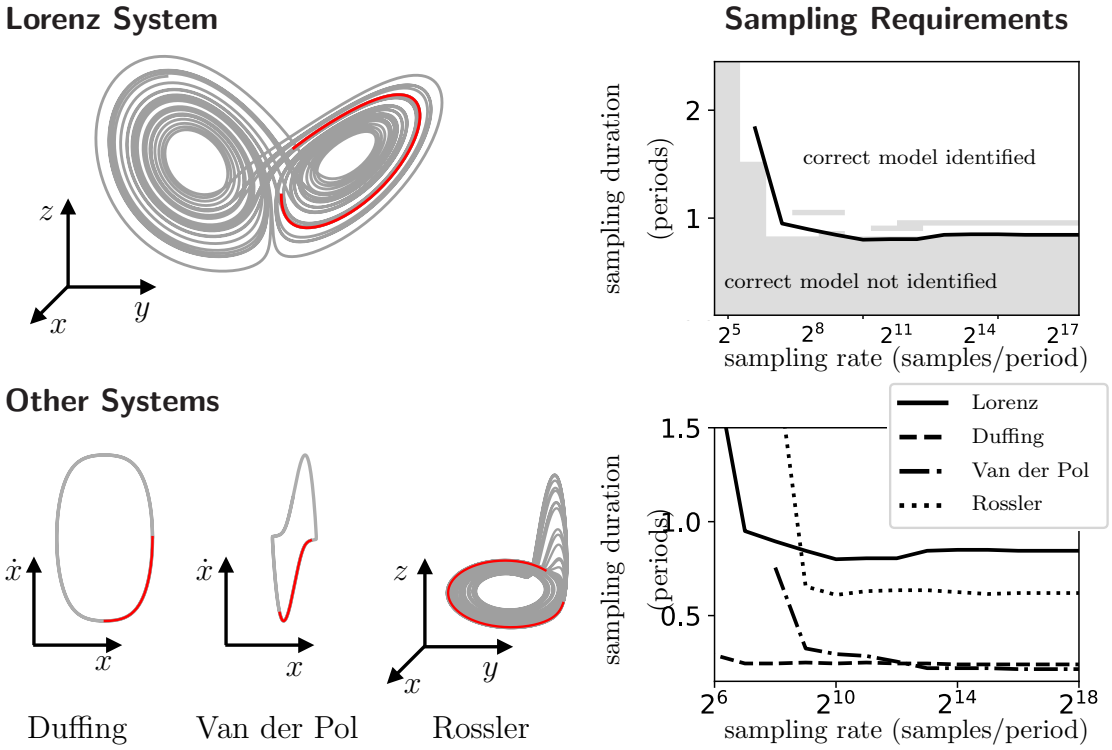

Figure 3.1: Data requirements of SINDy for four example systems: the Lorenz system, Duffing oscillator, Van der Pol oscillator, and Rossler system. We plot the attractor of each system in gray. The portion of the attractor highlighted in red indicates the average portion we must sample from to discover the system using SINDy. For all systems, we see that we do not need to sample from the full attractor. Plots on the right indicate how the sampling rate affects the duration we must sample to obtain the correct model. In the bottom plot, we show for each system the average sampling duration necessary to identify the correct model at various sampling rates. The top plot provides a detailed look at SINDy's performance on the Lorenz system. (top right) Gray shading indicates regions where the correct model was not identified. The black curve indicates on average the sampling duration necessary at each sampling rate (and is the same curve plotted for the Lorenz system in the bottom plot).

3.1.1 Lorenz system

As a first example, consider the chaotic Lorenz system:

$$\dot{x} = \sigma(y - x)$$
$$\dot{y} = x(\rho - z) - y$$
$$\dot{z} = xy - \beta z,$$

with parameters $\sigma = 10$, $\rho = 28$, and $\beta = 8/3$. The dynamics of this system evolve on an attractor shown in Figure 3.1. At the chosen parameter values the Lorenz attractor has two lobes, and the system can make anywhere from one to many cycles around the current lobe before switching to the other lobe. For the Lorenz system, we define the period T to be the typical time it takes the system to travel around one lobe. At the chosen parameter values, we determine that $T \approx 0.759$ (calculated by averaging over many trips around the attractor); however, because of the chaotic nature of the system, the time it takes to travel around one lobe is highly variable.

In the top right panel of Figure 3.1, we plot the sampling rates and durations at which the system is correctly identified. The black curve shows the average duration of recorded data necessary to discover the correct model at each sampling rate. As the sampling rate increases, SINDy is able to correctly identify the dynamics with a shorter recording duration. In our analysis, we find that the system has both a baseline required sampling rate and a baseline required duration: if the sampling rate is below the baseline, increasing the duration further does not help discover the model. Similarly, if the duration is below the baseline, increasing the sampling rate does not help discover the model. For our chosen parameter values, we find the average baseline sampling duration to be 85% of a period. Depending on the portion of the attractor sampled from, this duration ranges from 70-110% of a period. The portion of the attractor covered by this average baseline duration is shown highlighted in red on the Lorenz attractor in Figure 3.1.

3.1.2 Duffing oscillator

Next we consider the Duffing oscillator, which can be written as a two-dimensional dynamical system:

$$\dot{x} = y$$
$$\dot{y} = -\delta y - \alpha x - \beta x^3.$$

We consider the undamped equation with $\delta = 0$, $\alpha = 1$, and $\beta = 4$. The undamped Duffing oscillator exhibits regular periodic oscillations. The parameter $\beta > 0$ controls how nonlinear the oscillations are. At the selected parameter values, the system has period $T \approx 3.179$. We simulate the Duffing system, then apply the SINDy algorithm to batches of data subsampled at various rates and durations on different portions of the attractor.

The phase portrait and sampling requirements for the Duffing oscillator are shown in Figure 3.1. With a sufficiently high sampling rate, SINDy requires approximately 25% of a period on average in order to correctly identify the dynamics. This portion of the attractor is highlighted in red on the phase portrait in Figure 3.1. Depending on where on the attractor we sample from, the required duration ranges from about 15-35% of a period.

3.1.3 Van der Pol oscillator

We next look at the Van der Pol oscillator, given by

$$\dot{x}_1 = x_2 \tag{3.1a}$$
$$\dot{x}_2 = \mu(1 - x_1^2)x_2 - x_1. \tag{3.1b}$$

The parameter $\mu > 0$ controls the degree of nonlinearity; we use $\mu = 5$. At this parameter value we have period $T \approx 11.45$. We simulate the Van der Pol oscillator and again apply the SINDy method to sets of subsampled data.

The phase portrait and average sampling requirements for the Van der Pol oscillator are shown in Figure 3.1. The average baseline duration is around 20% of a period. For this

system in particular, the baseline duration is highly dependent on where the data is sampled from. If samples are taken during the fast part of the oscillation, the system can be identified with as little as 5% of a period; sampling during the slower part of the oscillation requires as much as 35% a period.

3.1.4 Rossler system

As a final example, consider the Rossler system

$$\tau\dot{x} = -y - z \tag{3.2a}$$

$$\tau\dot{y} = x + ay \tag{3.2b}$$

$$\tau\dot{z} = b + z(x - c), \tag{3.2c}$$

with parameters $a = 0.1$, $b = 0.1$, and $c = 14$. Note we include a time constant τ, with value $\tau = 0.1$. The dynamics of this system evolve on the chaotic attractor shown in Figure 3.1. We define the period of oscillation in this system as the typical time it takes to make one trip around the portion of the attractor in the $x - y$ plane, which at these parameter values (with $\tau = 0.1$) is $T \approx 6.14$. We simulate the system and apply the SINDy method as in previous examples.

The sampling requirements for the Rossler system are shown in Figure 3.1. Depending on the portion of the attractor we sample from, the baseline duration ranges from 35-95% of a period of oscillation. Remarkably, SINDy can identify the system without any data from when the system leaves the $x - y$ plane. The average baseline sampling duration (65% of a period) is highlighted in red on the Rossler attractor in Figure 3.1.

3.2 Sampling strategies for multiscale systems

We have established that SINDy can identify uniscale dynamical systems with relatively little data. However, many systems of interest contain coupled dynamics at multiple time scales. Identifying the dynamics across scales would require sampling at a sufficiently fast rate to capture the fast dynamics while also sampling for a sufficient duration to observe

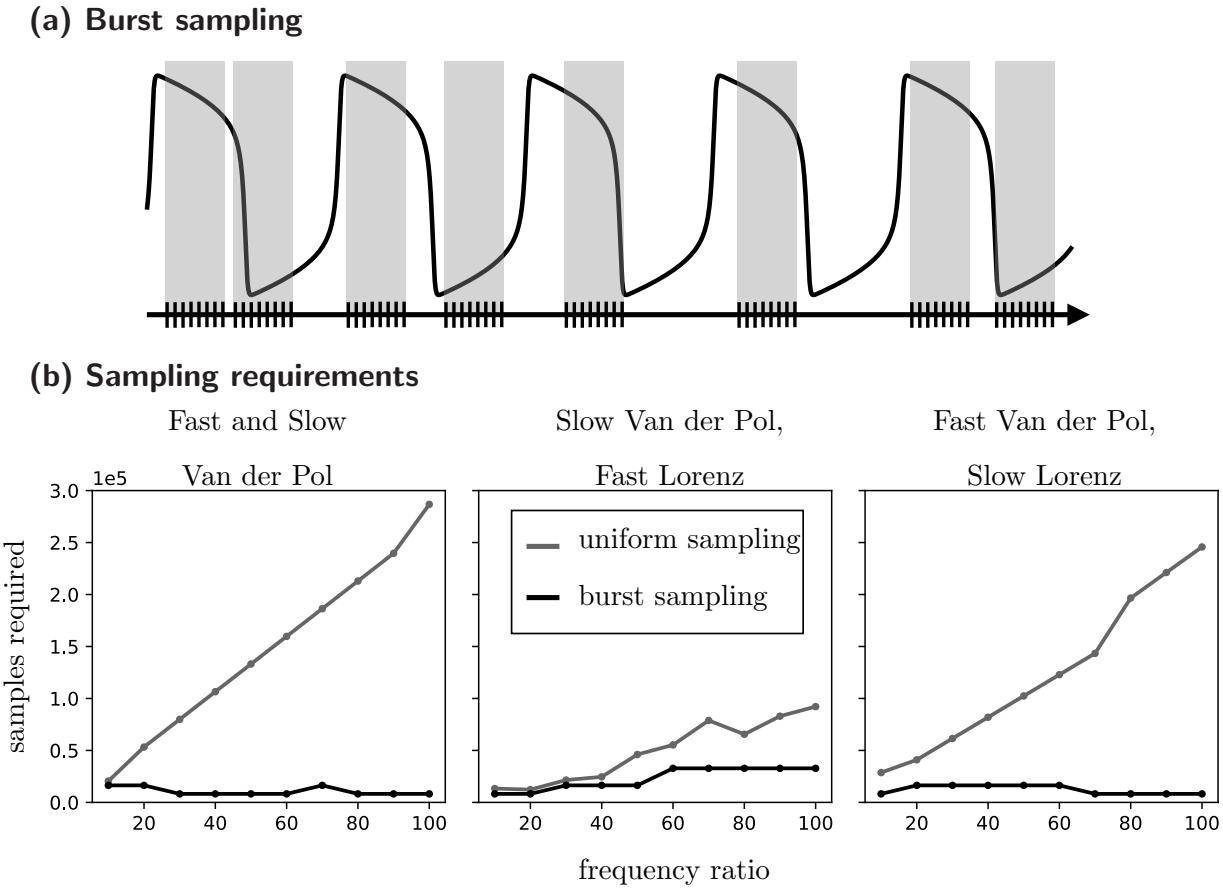

Figure 3.2: Schematic and results of our burst sampling method for applying SINDy to multiscale systems. (a) Illustration of burst sampling. Samples are collected in short bursts at a fine resolution, spaced out randomly over a long duration. This allows us to sample for a long time with a reduced effective sampling rate (as compared with naive uniform sampling) without increasing the time step. (b) Comparison of uniform sampling and burst sampling on systems with multiple time scales. For each method we look at how many samples are required for SINDy to identify the correct model at different frequency ratios between the fast and slow time scales. With uniform sampling, the total sampling requirement increases significantly as the frequency ratio between the fast and slow systems increases. Burst sampling significantly reduces this scaling effect, allowing SINDy to scale efficiently to multiscale systems.

the slow dynamics. Assuming we collect samples at a uniform rate, this leads to an increase in the amount of data required as the time scales of the coupled system separate. In this section, we introduce a sampling method that overcomes this issue, allowing SINDy to scale efficiently to multiscale problems.

We consider coupled systems with two distinct time scales. In particular, we look at nonlinear systems with linear coupling:

$$\tau_{\text{fast}}\dot{\mathbf{u}} = \mathbf{f}(\mathbf{u}) + \mathbf{C}\mathbf{v}$$
$$\tau_{\text{slow}}\dot{\mathbf{v}} = \mathbf{g}(\mathbf{v}) + \mathbf{D}\mathbf{u}.$$

In this system $\mathbf{u}(t) \in \mathbb{R}^n$ is the set of variables that comprise the fast dynamics, and $\mathbf{v}(t) \in \mathbb{R}^l$ represents the slow dynamics. The linear coupling is determined by our choice of $\mathbf{C} \in \mathbb{R}^{n\times l}, \mathbf{D} \in \mathbb{R}^{l\times n}$, and time constants $\tau_{\text{fast}}, \tau_{\text{slow}}$, which determine the frequency of the fast and slow dynamics. We consider three example systems: two coupled Van der Pol oscillators, a slow Van der Pol oscillator coupled with a fast Lorenz system, and a fast Van der Pol oscillator coupled with a slow Lorenz system. In order to understand the effect of time scale separation, we consider the frequency ratio $F = T_{\text{slow}}/T_{\text{fast}}$ between the coupled systems; $T_{\text{slow}}, T_{\text{fast}}$ are the approximate "periods" of the slow and fast systems, defined for Lorenz and Van der Pol in Sections 3.1.1, 3.1.3 respectively. We assess how much data is required to discover the system as F increases.

For each of the three example multiscale systems, we assess the performance of SINDy using the same process outlined in Section 3.1. Using naive uniform sampling, the data requirement increases approximately linearly with the frequency ratio F for all three systems (see Figure 3.2). This means that the sampling requirement is extremely high for systems where the frequency scales are highly separated (large F). Because the tasks of data collection and of fitting models to large data sets can be computationally expensive, reducing the data required by SINDy is advantageous.

We introduce a sampling strategy to address this issue, which we refer to as burst sampling. By using burst sampling, the data requirement for SINDy stays approximately con-

stant as time scales separate (F increases). The top panel of Figure 3.2 shows an illustration of our burst sampling method. The idea is to maintain a small step size but collect samples in short bursts spread out over a long duration. This reduces the effective sampling rate, which is particularly useful when we are limited by bandwidth in the number of samples we can collect. By maintaining a small step size, we still observe the fine detail of the dynamics, and we can get a more accurate estimate of the derivative. However by spacing out the samples in bursts, our data also captures more of the slow dynamics than would be observed with the same amount of data collected at a uniform sampling rate. We thus reduce the effective sampling rate without degrading our estimates of the derivative or averaging out the fast time scale dynamics.

Employing burst sampling requires the specification of a few parameters. We must select a step size Δt, burst size (number of samples in a burst), duration over which to sample, and total number of bursts to collect. We fix a single step size and consider the effect of burst size, duration, and number of bursts. In general, we find that smaller burst sizes give better performance. We also find that while it is important to have a sufficiently long duration (on the order of 1-2 periods of the slow dynamics), increasing duration beyond this does not have a significant effect on performance. Therefore, for the rest of our analysis we fix a burst size of 8 and a sampling duration of $2T_{\text{slow}}$. We then adjust the total number of bursts collected in order to control the effective sampling rate.

Another important consideration is how to space out the bursts of samples. In particular we must consider the potential effects of aliasing, which could reduce the effectiveness of this method if bursts are spaced to cover the same portions of the attractor. To address this issue, we introduce randomness into our decision of where to place bursts. In streaming data applications, this can be handled in a straightforward manner by selecting burst collection times as Poisson arrival times, with the rate of the Poisson process chosen so that the expected number of samples matches our desired overall sampling rate. For the purpose of testing our burst sampling method, we do something slightly different. In order to understand the performance of burst sampling, we need to perform repeated trials with different choices of

burst locations. We also need to ensure that for each choice of parameters, the training data for each trial has a constant number of samples and covers the same duration. If burst locations are selected completely randomly, it is likely that some will overlap which would reduce our number of overall samples. To address this, we start with evenly spaced bursts and sample offsets for each, selected from a uniform distribution that allows each burst to shift left or right while ensuring that none of the bursts overlap. In this manner we obtain the desired number of samples at each trial, but we introduce enough randomness into the selection of burst locations to account for aliasing.

In Figure 3.2 we show the results of burst sampling for our three example systems. For all three systems, the number of samples required by SINDy remains approximately constant as the frequency ratio F increases. To determine the number of samples required at a given frequency ratio, we fix an effective sampling rate and run repeated trials with burst locations selected randomly as described above. For each trial, we apply SINDy to the sampled data and determine if the correct nonzero coefficients were identified, as in Section 3.1. We run 100 trials at each effective sampling rate and find the minimum rate for which the system was correctly identified in all trials; this determines the required effective sampling rate. In Figure 3.2, we plot the total number of samples collected with this effective sampling rate.

Chapter 4

MULTISCALE SYSTEMS WITH INCOMPLETE MEASUREMENTS: SAMPLING STRATEGIES FOR HAVOK

The results in Chapter 3 assessed the performance of the SINDy algorithm on coupled multiscale systems with limited full-state measurements in time. One limitation of SINDy is that it requires knowledge of the full underlying state space variables that govern the behavior of the system of interest. In many real-world applications, some governing variables may be completely unobserved or combined into mixed observations. With multiscale systems in particular, a single observation variable may contain dynamics from multiple time scales. We therefore need methods for understanding multiscale dynamics that do not require full state measurements.

In this chapter, we consider systems for which we do not have full state measurements. In Section 4.1 we provide background on time-delay embedding and the Hankel alternative view of Koopman (HAVOK) method [10]. We then show in Section 4.2 that for quasiperiodic systems, HAVOK provides a closed linear model using time-delay embeddings that allows for highly accurate reconstruction and long-term prediction of the dynamics on the attractor. These models do not need the nonlinear forcing term that was necessary for the chaotic systems studied in [10]. We then provide two strategies for using HAVOK to model multiscale systems. In Section 4.3.1 we introduce a strategy for constructing our data matrices so that we can create an accurate model for the system with less data. Section 4.3.2 introduces an iterative method for separating fast and slow time scales.

4.1 Background: time-delay embedding and HAVOK

The recent Hankel alternative view of Koopman (HAVOK) method constructs an approximation to the Koopman operator (2.3) by relying on the relationship between the Koopman operator and the Takens embedding [80, 10]. Delay coordinates were previously used to augment the rank in DMD [83, 9], and the connection to Koopman theory has been strengthened [2, 18] following the original HAVOK paper [10]. Measurements of the system are formed into a Hankel matrix, which is created by stacking delayed measurements of the system:

$$\mathbf{H} = \begin{pmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_p \\ \mathbf{x}_2 & \mathbf{x}_3 & \cdots & \mathbf{x}_{p+1} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{x}_q & \mathbf{x}_{q+1} & \cdots & \mathbf{x}_m \end{pmatrix}. \tag{4.1}$$

A number of other algorithms make use of the Hankel matrix, including the eigensystem realization algorithm (ERA) [36, 8, 47]. By taking a singular value decomposition (SVD) of the Hankel matrix, we are able to obtain dominant time-delay coordinates that are approximately invariant to the Koopman operator [10]. Thus the time-delay embedding provides a new coordinate system in which the dynamics are linearized.

In [10] the focus is on chaotic systems, and an additional nonlinear forcing term is included in the HAVOK model to account for chaotic switching or bursting phenomena. Here we focus on quasiperiodic systems and show that the linear model found by HAVOK is sufficient for reconstruction and long-term prediction, with no need for a forcing term. HAVOK has the advantage that the discovered models are linear and require no prior knowledge of the true governing variables. By eliminating the need for the nonlinear forcing term, we obtain deterministic, closed-form models. In this section we assess the performance of HAVOK on uniscale dynamical systems and introduce two strategies for scaling the method to problems with multiple time scales.

4.2 Uniscale dynamical systems

We start by focusing on quasiperiodic systems with a single time scale. To apply the HAVOK method, we form two shift-stacked matrices that are analogous to the snapshot matrices typically formed in DMD [44]:

$$\mathbf{H} = \begin{pmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_{m-q} \\ \mathbf{x}_2 & \mathbf{x}_3 & \cdots & \mathbf{x}_{m-q+1} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{x}_q & \mathbf{x}_{q+1} & \cdots & \mathbf{x}_{m-1} \end{pmatrix}, \quad \mathbf{H}' = \begin{pmatrix} \mathbf{x}_2 & \mathbf{x}_3 & \cdots & \mathbf{x}_{m-q+1} \\ \mathbf{x}_3 & \mathbf{x}_4 & \cdots & \mathbf{x}_{m-q+2} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{x}_{q+1} & \mathbf{x}_{q+2} & \cdots & \mathbf{x}_m \end{pmatrix}. \tag{4.2}$$

We then perform the standard DMD algorithm on the shift-stacked matrices. A rank-r DMD model gives us a set of r eigenvalues λ_k, modes $\boldsymbol{\phi}_k \in \mathbb{C}^n$, and amplitudes b_k. By rewriting the discrete-time eigenvalues λ_k as $\omega_k = \ln(\lambda_k)/\Delta t$, we obtain a linear model

$$\mathbf{x}(t) = \sum_{k=1}^{r} \boldsymbol{\phi}_k \exp(\omega_k t) b_k. \tag{4.3}$$

Note that this linear model provides a prediction of the behavior at any time t, without requiring the system to be fully simulated up to that time.

In Figure 4.2 we show HAVOK models for the periodic Van der Pol, Rossler, and Lorenz systems. In each example we take observations of a single variable of the system and time delay embed to build a linear HAVOK model. At the selected parameter values, each of these systems exhibits regular periodic behavior. We also show the power spectrum of each HAVOK model by plotting the model frequencies $\mathrm{Im}(\omega_k)$ and their respective amplitudes $|b_k|$.

When applying HAVOK to data, we must consider the sampling rate, number of delays q, and rank of the model r. In order to obtain a good model, it is imperative that (1) there are enough delay coordinates to provide a model of sufficient rank and (2) the delay duration $D = (q-1)\Delta t$ must be large enough to capture a sufficient duration of the oscillation. To obtain a good model, we need the modes found by taking an SVD of $\mathbf{H}$ to resemble Fourier modes. This is due to the fact that HAVOK builds a linear dynamical model: linear

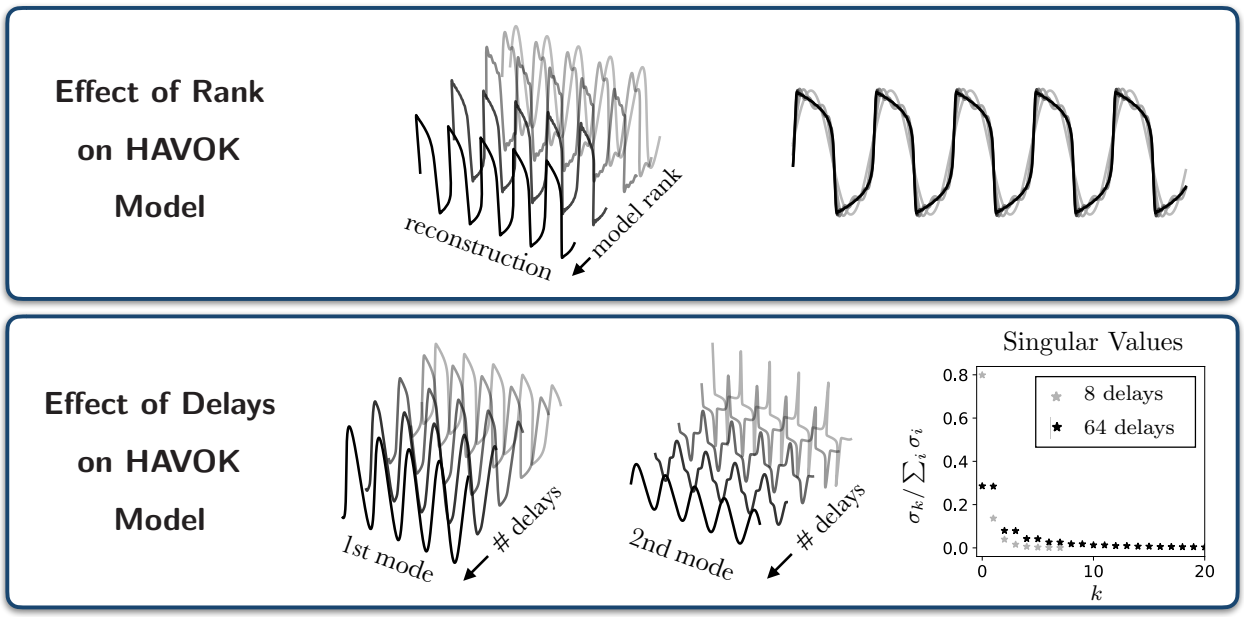

Figure 4.1: Effects of rank and delays on HAVOK model of a Van der Pol oscillator. In the top panel, we show how the choice of model rank affects the HAVOK model. As the rank of the model increases, the model goes from being a sinusoidal linear oscillator to closely matching the Van der Pol dynamics. In the bottom panel, we show how the number and duration of the delay coordinates affects the HAVOK modes and singular values. HAVOK models are linear, and linear dynamics consist of the superposition of sinusoids; thus to obtain a good model the modes found by taking an SVD of $\mathbf{H}$ should resemble Fourier modes (in time). With few delays the modes are highly nonlinear, meaning that our (linear) HAVOK model cannot accurately capture the dynamics. As the number of delays increases the modes appear more like Fourier modes, allowing us to construct a good linear model. The right panel shows the singular values for two models, one with 8 delays and one with 64 delays (only the first 20 singular values are shown). Using 8 delays admits a low-rank model with nonlinear modes; most of the energy is captured in the first mode. Using 64 delays admits linear modes, and the energy is more evenly shared among the first several modes; therefore several modes must be included to get an accurate model.

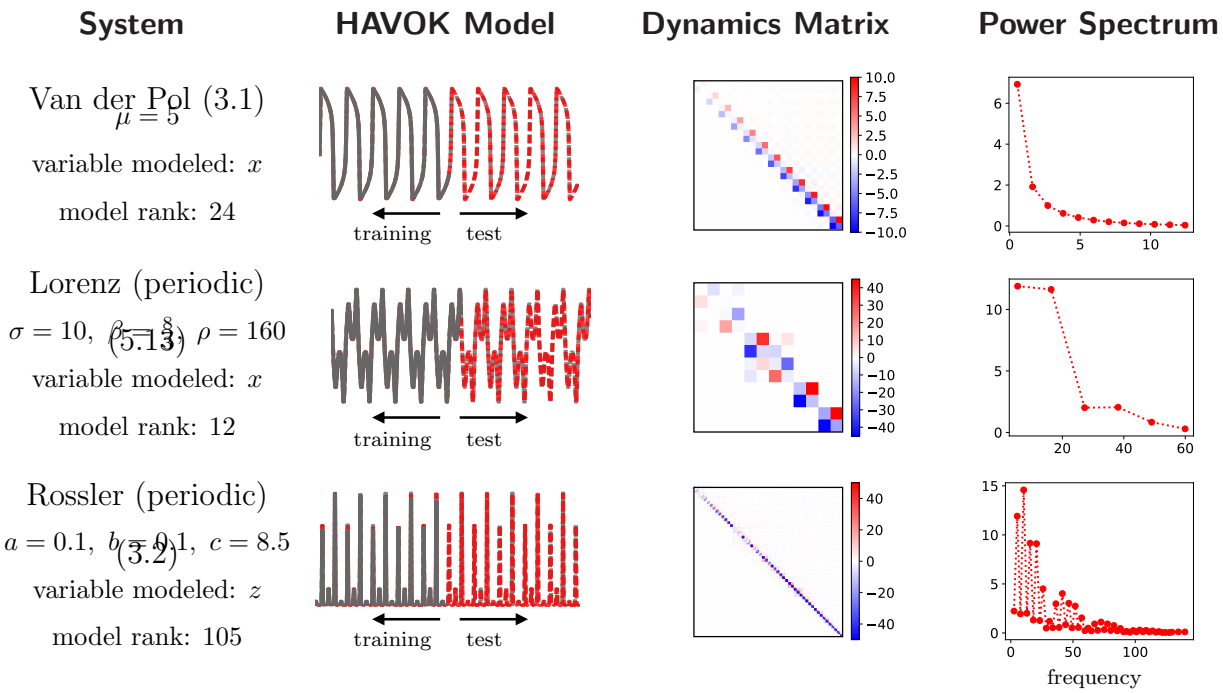

Figure 4.2: Hankel alternative view of Koopman (HAVOK) models for three example systems. We train HAVOK models for a single variable of each system. For each example we show the model reconstruction of the dynamics, the matrix defining the linear model of the dynamics, and the power spectrum of the DMD modes.

dynamics without damping consists of the superposition of pure tone sinusoids in time. Highly nonlinear modes will not be well captured by a linear HAVOK model. In Figure 4.1 we show how the modes for a Van der Pol oscillator go from nonlinear to Fourier modes as we increase the number of time delays (which simultaneously increases the delay duration D). As a rule of thumb, for a system with period T we choose our delays such that $D = T$. Note that with a high sampling rate, this could necessitate using a significant number of delay coordinates, making the snapshot matrices impractically large. Two options for dealing with this are (1) downsampling the data and (2) spacing out the delays; this is discussed further in Section 4.3.1. For all examples in Figure 4.2, we use $q = 128$ delays and take $\Delta t = T/(q-1)$. We fit the model using training data from a single trajectory covering 5 periods of oscillation ($m\Delta t = 5T$).

Choosing the proper rank of the HAVOK model is another important consideration. Increasing the rank of the model adds additional frequencies, which in general can lead to a more accurate reconstruction. The top panel of Figure 4.1 shows how increasing rank affects a HAVOK model of a Van der Pol oscillator. A rank-2 model looks like a linear oscillator with the same dominant frequency as the Van der Pol oscillator. As we increase the rank, the model becomes more like a Van der Pol oscillator, but we observe Gibbs phenomenon near the sharp transitions in the dynamics. With a rank-32 model we get a very close reconstruction of the Van der Pol behavior. In order to obtain a model of rank r, we must choose the number of delays q such that $q > r$. Assuming the delay embedding is constructed adequately for capturing the dynamics (m, D sufficiently large and Δt sufficiently small), the rank of the model could be selected using traditional Pareto front analysis.

Our HAVOK model is constructed by taking an SVD of the delay embedding matrix, which consists of samples from a finite sampling period. This introduces some bias into the calculation of the eigenvalues of the systems. In particular, for periodic and quasiperiodic systems where we are trying to capture the behavior on the attractor, we do not want our model to have blowup or decay. Thus the continuous eigenvalues ω_k should have zero real part. In general, when we apply HAVOK we obtain eigenvalues with a small but nonzero real

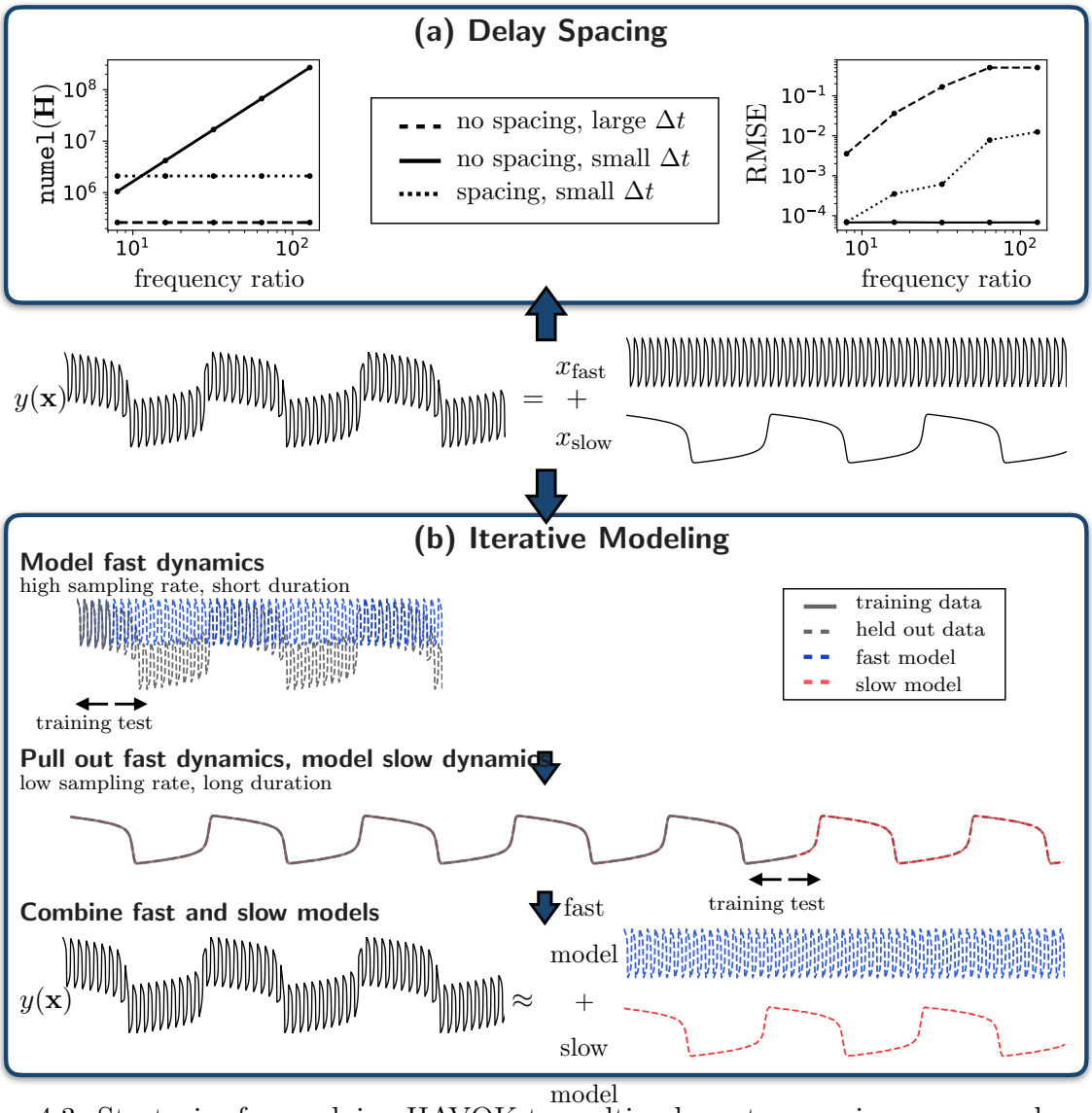

Figure 4.3: Strategies for applying HAVOK to multiscale systems, using an example system of a fast and a slow Van der Pol oscillator. (a) Spacing out the rows and columns of the HAVOK snapshot matrices while maintaining a small time step allows us to make a flexible trade-off between computational cost and model error. We show how the choice of time step, number of delay coordinates, and spacing affects the size of the embedding matrices (left) and model error (right). (b) Schematic of our iterative method for applying HAVOK to multiscale systems. First, HAVOK is applied to training data from a very short recording period to capture just the fast dynamics. Next, the data is sampled over a longer duration at a much lower rate. The known fast dynamics are subtracted out and the resulting data is used to create a HAVOK model for the slow dynamics. The fast and slow models can be combined to model the full system.

part. To deal with this issue, we simply set the real part of all eigenvalues to zero after fitting the HAVOK model. This approach is also taken in [47]. Forcing all eigenvalues to be purely imaginary allows the model to predict long term behavior of the system without blowup or decay. While we find this sufficient to produce good models for the example systems studied, an alternative solution would be to use a method such as optimized DMD with an imposed constraint on the eigenvalues [3].

4.3 Multiscale dynamical systems

It is still feasible to apply HAVOK as described above to many systems with multiple time scales. As with SINDy, the time scale separation requires that the amount of data acquired by the algorithm increase to account for both the fast and slow dynamics so that the problem becomes more computationally expensive. The use of delay coordinates compounds this issue beyond just a blowup in the number of samples required: not only do we need a sufficiently small time step to capture the fast dynamics, we also need a sufficient number of delays q so that our delay duration $D = (q-1)\Delta t$ covers the dynamics of the slow time scale. Thus the size of our snapshot matrices scales as F^2 with the frequency ratio F between fast and slow time scales. For large frequency separations and systems with high-dimensional state variables, this results in an extremely large delay embedding matrix that could make the problem computationally intractable. In this section, we discuss two strategies for applying HAVOK to problems with multiple time scales.

4.3.1 Method 1: Delay spacing

Our first approach for applying HAVOK to multiscale problems is a simple modification to the form of the delay embedding matrices. Rather than using the standard embedding matrices given in (4.2), we introduce row and column spacings d, c and create the following snapshot matrices:

$$\mathbf{H} = \begin{pmatrix} \mathbf{x}_1 & \mathbf{x}_{c+1} & \cdots & \mathbf{x}_{(p-1)c+1} \\ \mathbf{x}_{d+1} & \mathbf{x}_{d+c+1} & \cdots & \mathbf{x}_{d+(p-1)c+1} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{x}_{(q-1)d+1} & \mathbf{x}_{(q-1)d+c+1} & \cdots & \mathbf{x}_{(q-1)d+(p-1)c+1} \end{pmatrix},$$

$$\mathbf{H}' = \begin{pmatrix} \mathbf{x}_2 & \mathbf{x}_{c+2} & \cdots & \mathbf{x}_{(p-1)c+2} \\ \mathbf{x}_{d+2} & \mathbf{x}_{d+c+2} & \cdots & \mathbf{x}_{d+(p-1)c+2} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{x}_{(q-1)d+2} & \mathbf{x}_{(q-1)d+c+2} & \cdots & \mathbf{x}_{(q-1)d+(p-1)c+2} \end{pmatrix}.$$

The number of columns p is determined by the total number of samples. This form of delay embedding matrix separates out both the rows (delay coordinates) and columns (samples) without increasing the time step used in creating the HAVOK model. Each shift-stacked copy of the data advances in time by $d\Delta t$, which means the delay duration D is now multiplied by a factor of d, giving $D = (q-1)d\Delta t$. If we keep q fixed and increase d, our delay embedding captures more of the dynamics without increasing the number of rows in the embedding matrices. Similarly, we construct $\mathbf{H}$ and $\mathbf{H}'$ so that each column advances in time by $c\Delta t$, meaning we reduce the number of columns in the snapshot matrices by a factor of c without reducing the total sampling duration. The key point is that we keep the time step small by constructing $\mathbf{H}'$ so that it only advances each data point by time step Δt from the snapshots in $\mathbf{H}$. This allows the HAVOK model to account for the fast time scale.

We consider multiscale systems with two time scales and assess performance as the frequency ratio between the time scales grows. In particular, as the frequency ratio increases we are concerned with both the size of the snapshot matrices and the prediction error of the respective model. We compare three different models: a standard HAVOK model with a large time step, a standard HAVOK model with a small time step, and a HAVOK model built with row and column spacing in the snapshot matrices. As discussed in Section 4.2, the most important considerations in building a HAVOK model are selecting the model rank

r and using a sufficiently large delay duration D. In our comparison we keep $r = 100$ fixed and set D equal to the period of the slow dynamics.

As an example we consider a system of two Van der Pol oscillators with parameter $\mu = 5$ and periods $T_{\text{fast}}, T_{\text{slow}}$. We sum the activity of the two oscillators together to produce a multiscale time series of one variable. We adjust the frequency ratio $F = T_{\text{slow}}/T_{\text{fast}}$ and compare the performance of the three different HAVOK models discussed above. When adjusting the frequency ratio, we keep the period of the fast oscillator fixed and adjust the period of the slow oscillator. In all models we train using time series data covering five periods of the slow oscillation ($m\Delta t = 5T_{\text{slow}}$). As F increases, we scale our models in three different ways. For the HAVOK model with a large time step, we fix q and increase Δt such that the delay duration $D = (q-1)\Delta t = T_{\text{slow}}$. For the HAVOK model with a small time step, we fix Δt and increase the number of delays q to obtain delay duration $D = (q-1)\Delta t = T_{\text{slow}}$. Finally, for the HAVOK model with spacing, we fix $q, \Delta t$ and increase the row spacing d such that $D = (q-1)d\Delta t = T_{\text{slow}}$. We also increase column spacing c such that we have the same number of samples for each system. This allows us to maintain a constant size of the snapshot matrix as F increases.

In the top panel of Figure 4.3 we compare both the size of the snapshot matrices and the prediction error for our three different models. By increasing the time step in the standard HAVOK model as the frequency ratio increases, we are able to avoid increasing the size of the snapshot matrix but are penalized with a large increase in error (dashed lines). In contrast, building a standard HAVOK model with a small time step allows us to maintain a small error but causes F^2 growth in the size of the snapshot matrices (solid lines). By introducing row and column spacing, we can significantly reduce the error while also avoiding growth in the size of the snapshot matrices (dotted lines). While we choose our row and column spacings d, c to avoid any growth in the size of the snapshot matrices as the frequency ratio grows, these can be adjusted to give more flexibility in the size/error trade off. This allows the method to be very flexible based on computational resources and accuracy requirements.

4.3.2 Method 2: iterative modeling

Our second strategy is an iterative method for modeling multiscale systems using HAVOK. A key enabling assumption is that with sufficient frequency separation between time scales, the slow dynamics will appear approximately constant in the relevant time scale of the fast dynamics. This allows us to build a model for the fast dynamics using data from a short time period without capturing for the slow dynamics. We then use this model to predict the fast behavior over a longer duration, subtract it out, and model the slow dynamics using downsampled data.

The algorithm proceeds in three steps, illustrated in Figure 4.3. As training data we use the same time series of two Van der Pol oscillators used in Section 4.3.1. In Figure 4.3 we show frequency ratio $F = 20$. In the first step, we sample for a duration equivalent to five periods of the fast oscillation at a rate of 128 samples/period. We use this data to build a HAVOK model with rank $r = 50$ and $q = 128$, using the procedure described in Section 4.2. The slow dynamics are relatively constant during this recording period, meaning the model only captures the fast dynamics. Setting the real part of all DMD eigenvalues to zero and subtracting any constants (modes with zero eigenvalues), we obtain a model for the fast dynamics. Such subtraction of *background* modes has been used effectively for foreground-background separation in video streams [28, 20]

The next step is to approximate the slow dynamics, which will be used to build the slow model. We sample the system over a longer duration at a reduced sampling rate; in the example we sample for five periods of the slow oscillation and reduce the sampling rate by a factor of the frequency ratio F. We use our fast model to determine the contribution of the fast dynamics at the sampled time points and subtract this out to get an approximation of the slow dynamics. Note that because we have a linear model of the form in (4.3), we can predict the contribution of the fast oscillator at these sample points without needing to run a simulation on the time scale of the fast dynamics.

Finally we use this subsampled data as training data to construct a HAVOK model of

the slow dynamics. In the example in Figure 4.3, we again build a rank 50 model using 128 delay coordinates. This procedure gives us two models, one for the fast dynamics and one for the slow dynamics. These models can be combined to give a model of the full system or used separately to analyze the individual time scales.

This method provides an additional strategy for applying HAVOK to multiscale systems. As it relies on the assumption that the slow dynamics are relatively constant in the relevant time scale of the fast dynamics, it is best suited to problems where there is large separation between time scales. In practice, the application of this method requires a choice of how long to sample to capture the fast dynamics. The method is not sensitive to the exact sampling duration used provided it captures at least a few periods of the fast oscillation and the slow dynamics remain approximately constant throughout this duration. One strategy would be to use the frequency spectrum of the data to inform this decision.

Chapter 5

SIMULTANEOUS DISCOVERY OF COORDINATES AND DYNAMICS

Obtaining parsimonious models is fundamentally linked to the coordinate system in which the dynamics are measured. As introduced in Chapter 4, without knowledge of the proper coordinates, approaches such as SINDy may fail to discover simple dynamical models. The joint discovery of models and coordinates is critical for understanding many modern complex systems. In particular, successful model identification relies on the assumption that the dynamics are measured in a coordinate system in which the dynamics may be sparsely represented. While simple models may exist in one coordinate system, a different coordinate system may obscure these parsimonious representations. For modern applications of data-driven discovery, there is no reason to believe that our sensors are measuring the correct variables to admit a parsimonious representation of the dynamics. This motivates the systematic and automated discovery of coordinate transformations to facilitate this sparse representation.

The challenge of discovering an effective coordinate system is as fundamental and important as model discovery. Many key historical scientific breakthroughs were enabled by the discovery of appropriate coordinate systems. Celestial mechanics, for instance, was revolutionized by the heliocentric coordinate system of Copernicus, Galileo, and Kepler, thus displacing Ptolemy's *doctrine of the perfect circle*, which was dogma for more than a millennium. Fourier introduced his famous transform to simplify the representation of the heat equation, resulting in a *sparse*, diagonal, decoupled linear system. Eigen-coordinates have been used more broadly to enable simple and sparse decompositions, for example in quantum mechanics and electrodynamics, to characterize energy levels in atoms and propagating

modes in waveguides, respectively. Principal component analysis (PCA) is one of the most prolific modern coordinate discovery methods, representing high-dimensional data in a low-dimensional linear subspace [63, 33]. Nonlinear extensions of PCA have been enabled by a neural network architecture, called an autoencoder [4, 58, 27]. However, PCA coordinates and autoencoders generally do not take dynamics into account and, thus, may not provide the right basis for parsimonious dynamical models. In a related vein, Koopman analysis seeks to discover coordinates that linearize nonlinear dynamics [41, 56, 13, 57]; while linear models are useful for prediction and control, they cannot capture the full behavior of many nonlinear systems. Thus, it is important to develop methods that combine simplifying coordinate transformations and nonlinear dynamical models. We advocate for a balance between these approaches, identifying coordinate transformations where only a few nonlinear terms are present, as in the classic theory of near-identity transformations and normal forms [34, 90].

In this chapter we present a method for discovery of nonlinear coordinate transformations that enable associated parsimonious dynamics. Our method combines a custom autoencoder network with a SINDy model for parsimonious nonlinear dynamics. The autoencoder architecture enables the discovery of reduced coordinates from high-dimensional input data that can be used to reconstruct the full system. The reduced coordinates are found along with nonlinear governing equations for the dynamics in a joint optimization. We demonstrate the ability of our method to discover parsimonious dynamics on three examples: a high-dimensional spatial data set with dynamics governed by the chaotic Lorenz system, a spiral wave resulting from the reaction-diffusion equation, and the nonlinear pendulum. These results demonstrate how to focus neural networks to discover interpretable dynamical models. The proposed method is the first to provide a mathematical framework that places the discovery of coordinates and models on equal footing.

5.1 Background: neural networks for dynamical systems

The success of neural networks (NNs) on problems such as image classification [42] and speech recognition [29] has led to the use of NNs to perform a wide range of tasks in science

and engineering. One recent area of focus has been the use of NNs for studying dynamical systems, which has a surprisingly rich history [26]. In addition to improving solution techniques for systems with known equations [65, 66, 67, 68, 5], deep learning has been used for understanding and predicting dynamics for complex systems with potentially unknown equations [55, 84, 85, 94, 79, 52, 61]. Several recent methods have trained NNs to predict dynamics, including a time-lagged autoencoder which takes the state at time t as input data and uses an autoencoder-like structure to predict the state at time $t + \tau$ [85]. Other approaches use a recurrent NN architecture, particularly long short-term memory (LSTM) networks, for applications involving sequential data [30, 31, 50]. LSTMs have recently been used to perform forecasting on chaotic dynamical systems [84]. Reservoir computing has also enabled impressive predictions [62]. Autoencoders are increasingly being leveraged for dynamical systems because of their close relationship to other dimensionality reduction techniques [58, 14, 25, 48].

Another class of NNs use deep learning to discover coordinates for Koopman analysis. Koopman theory seeks to discover coordinates that linearize nonlinear dynamics [41, 56, 13, 57]. Methods such as dynamic mode decomposition (DMD) [75, 70, 76, 43], extended DMD [88], kernel DMD [89], and time-delay DMD [10, 2] build linear models for dynamics, but these methods rely on a proper set of coordinates for linearization. Several recent works have focused on the use of deep learning methods to discover the proper coordinates for DMD and extended DMD [49, 94, 79]. Other methods seek to learn Koopman eigenfunctions and the associated linear dynamics directly using autoencoders [61, 52].

Despite their widespread use, NNs face three major challenges: generalization, extrapolation, and interpretation. The hallmark success stories of NNs (computer vision and speech, for instance) have been on data sets that are fundamentally interpolatory in nature. The ability to extrapolate, and as a consequence generalize, is known to be an underlying weakness of NNs. This is especially relevant for dynamical systems and forecasting, which is typically an extrapolatory problem by nature. Thus models trained on historical data will generally fail to predict future events that are not represented in the training set. An additional limitation

of deep learning is the lack of interpretability of the resulting models. While attempts have been made to interpret NN weights, network architectures are typically complicated with the number of parameters (or weights) far exceeding the original dimension of the dynamical system. The lack of interpretability also makes it difficult to generalize models to new data sets and parameter regimes. However, NN methods still have the potential to learn general, interpretable dynamical models if properly constrained or regularized. In addition to methods for discovering linear embeddings [52, 61], deep learning has also been used for parameter estimation of PDEs [67, 68].

5.2 SINDy autoencoders

We present a method for the simultaneous discovery of sparse dynamical models and coordinates that enable these simple representations. Our aim is to leverage the parsimony and interpretability of SINDy with the universal approximation capabilities of deep neural networks [35] in order to produce interpretable and generalizable models capable of extrapolation and forecasting. Our approach combines a SINDy model and a deep autoencoder network to perform a joint optimization that discovers intrinsic coordinates which have an associated parsimonious nonlinear dynamical model. The architecture is shown in Figure 5.1. We again consider dynamical systems of the form

$$\frac{d}{dt}\mathbf{x}(t) = \mathbf{f}(\mathbf{x}(t))$$

where $\mathbf{x}(t) \in \mathbb{R}^n$ is the state of the system at time t. While this dynamical model may be dense in terms of functions of the original measurement coordinates $\mathbf{x}$, our method seeks a set of reduced coordinates $\mathbf{z}(t) = \varphi(\mathbf{x}(t)) \in \mathbb{R}^d$ ($d \ll n$) with an associated dynamical model

$$\frac{d}{dt}\mathbf{z}(t) = \mathbf{g}(\mathbf{z}(t)) \tag{5.1}$$

that provides a parsimonious description of the dynamics. This means that $\mathbf{g}$ contains only a few active terms. Along with the dynamical model, the method provides coordinate transforms φ, ψ that map the measurement coordinates to intrinsic coordinates via $\mathbf{z} = \varphi(\mathbf{x})$ (encoder) and back via $\mathbf{x} \approx \psi(\mathbf{z})$ (decoder).

The coordinate transformation is achieved using an autoencoder network architecture. The autoencoder is a feedforward neural network with a hidden layer that represents the intrinsic coordinates. Rather than performing a task such as prediction or classification, the network is trained to output an approximate reconstruction of its input, and the restrictions placed on the network architecture (e.g. the type, number, and size of the hidden layers) determine the properties of the intrinsic coordinates [27]; these networks are known to produce nonlinear generalizations of PCA [4]. A common choice is that the dimensionality of the intrinsic coordinates $\mathbf{z}$, determined by the number of units in the corresponding hidden layer, is much lower than that of the input data $\mathbf{x}$: in this case, the autoencoder learns a *nonlinear* embedding into a reduced latent space. Our network takes measurement data $\mathbf{x}(t) \in \mathbb{R}^n$ from a dynamical system as input and learns intrinsic coordinates $\mathbf{z}(t) \in \mathbb{R}^d$, where $d \ll n$ is chosen as a hyperparameter prior to training the network.

While autoencoders can be trained in isolation to discover useful coordinate transformations and dimensionality reductions, there is no guarantee that the intrinsic coordinates learned will have associated sparse dynamical models. We require the network to learn coordinates associated with parsimonious dynamics by simultaneously learning a SINDy model for the dynamics of the intrinsic coordinates $\mathbf{z}$. This regularization is achieved by constructing a library $\boldsymbol{\Theta}(\mathbf{z}) = [\boldsymbol{\theta}_1(\mathbf{z}), \boldsymbol{\theta}_2(\mathbf{z}), \ldots, \boldsymbol{\theta}_p(\mathbf{z})]$ of candidate basis functions, e.g. polynomials, and learning a sparse set of coefficients $\boldsymbol{\Xi} = [\boldsymbol{\xi}_1, \ldots, \boldsymbol{\xi}_d]$ that defines the dynamical system

$$\frac{d}{dt}\mathbf{z}(t) = \mathbf{g}(\mathbf{z}(t)) = \boldsymbol{\Theta}(\mathbf{z}(t))\boldsymbol{\Xi}. \tag{5.2}$$

While the functions in the library must be specified prior to training, the coefficients $\boldsymbol{\Xi}$ are learned along with the NN parameters as part of the training procedure. Assuming derivatives $\dot{\mathbf{x}}(t)$ of the original states are available or can be computed, one can calculate the derivative of the encoder variables as $\dot{\mathbf{z}}(t) = \nabla_{\mathbf{x}}\varphi(\mathbf{x}(t))\dot{\mathbf{x}}(t)$ and enforce accurate prediction of the dynamics by incorporating the following term into the loss function:

$$\mathcal{L}_{d\mathbf{z}/dt} = \left\|\nabla_{\mathbf{x}}\varphi(\mathbf{x})\dot{\mathbf{x}} - \boldsymbol{\Theta}(\varphi(\mathbf{x})^T)\boldsymbol{\Xi}\right\|_2^2. \tag{5.3}$$

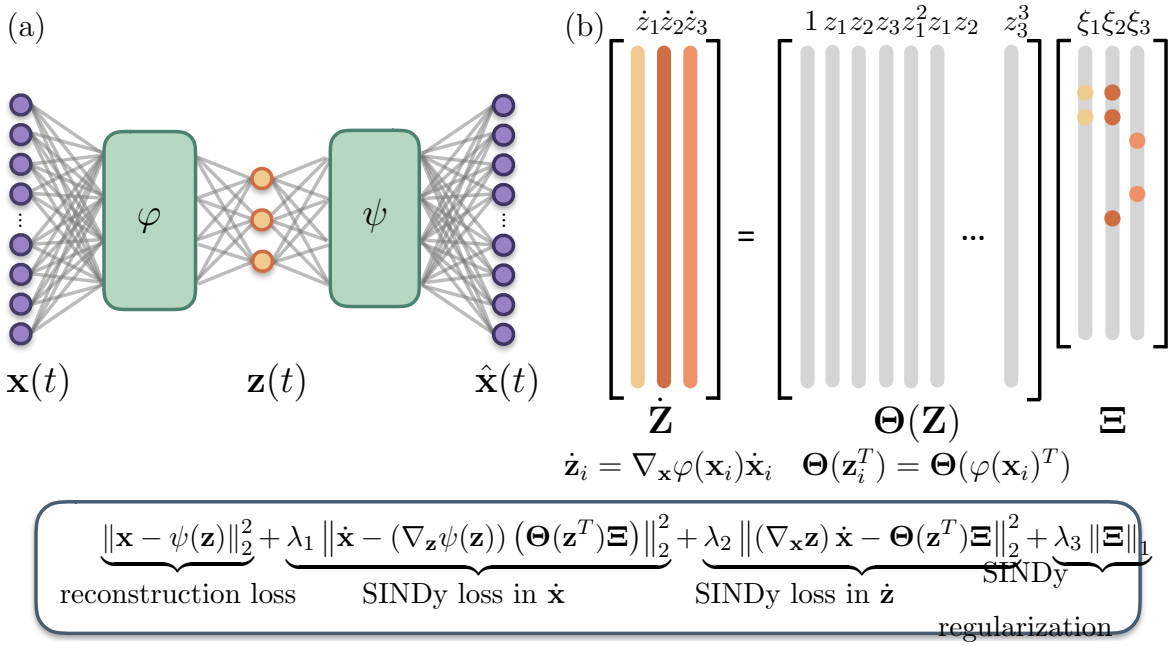

Figure 5.1: Schematic of the SINDy autoencoder method for simultaneous discovery of coordinates and parsimonious dynamics. (a) An autoencoder architecture is used to discover intrinsic coordinates $\mathbf{z}$ from high-dimensional input data $\mathbf{x}$. The network consists of two components: an encoder $\varphi(\mathbf{x})$, which maps the input data to the intrinsic coordinates $\mathbf{z}$, and a decoder $\psi(\mathbf{z})$, which reconstructs $\mathbf{x}$ from the intrinsic coordinates. (b) A SINDy model captures the dynamics of the intrinsic coordinates. The active terms in the dynamics are identified by the nonzero elements in $\boldsymbol{\Xi}$, which are learned as part of the NN training. The time derivatives of $\mathbf{z}$ are calculated using the derivatives of $\mathbf{x}$ and the gradient of the encoder φ. The inset shows the pointwise loss function used to train the network. The loss function encourages the network to minimize both the autoencoder reconstruction error and the SINDy loss in $\mathbf{z}$ and $\mathbf{x}$. L_1 regularization on $\boldsymbol{\Xi}$ is also included to encourage parsimonious dynamics.

This term uses the SINDy model along with the gradient of the encoder to encourage the learned dynamical model to accurately predict the time derivatives of the encoder variables. We include an additional term in the loss function that ensures SINDy predictions can be used to reconstruct the time derivatives of the original data:

$$\mathcal{L}_{d\mathbf{x}/dt} = \left\| \dot{\mathbf{x}} - (\nabla_{\mathbf{z}} \psi(\varphi(\mathbf{x}))) \left(\boldsymbol{\Theta}(\varphi(\mathbf{x})^T) \boldsymbol{\Xi} \right) \right\|_2^2 . \tag{5.4}$$

We combine (5.3) and (5.4) with the standard autoencoder loss function

$$\mathcal{L}_{\text{recon}} = \|\mathbf{x} - \psi(\varphi(\mathbf{x}))\|_2^2 , \tag{5.5}$$

which ensures that the autoencoder can accurately reconstruct the input data. We also include an L_1 regularization on the SINDy coefficients $\boldsymbol{\Xi}$, which promotes sparsity of the coefficients and therefore encourages a parsimonious model for the dynamics. The combination of the above four terms gives the following overall loss function:

$$\mathcal{L}_{\text{recon}} + \lambda_1 \mathcal{L}_{d\mathbf{x}/dt} + \lambda_2 \mathcal{L}_{d\mathbf{z}/dt} + \lambda_3 \mathcal{L}_{\text{reg}}, \tag{5.6}$$

where the scalars $\lambda_1, \lambda_2, \lambda_3$ are hyperparameters that determine the relative weighting of the three terms in the loss function.

In addition to the L_1 regularization, to obtain a model with only a few active terms we also incorporate sequential thresholding into the training procedure as a proxy for L_0 sparsity [96]. This technique is inspired by the original algorithm used for SINDy [12], which combined least squares fitting with sequential thresholding to obtain a sparse dynamical model. To apply sequential thresholding during training, we specify a threshold that determines the minimum magnitude for coefficients in the SINDy model. At fixed intervals throughout the training, all coefficients below the threshold are set to zero and training resumes using only the terms left in the model. We train the network using the Adam optimizer [40]. In addition to the loss function weightings and SINDy coefficient threshold, training requires the choice of several other hyperparameters including learning rate, number of intrinsic coordinates d, network size, and activation functions. Full details of the training procedure are discussed in Section 5.3.4.

5.3 Network Architecture and Training

5.3.1 Network architecture

The autoencoder network consists of a series of fully-connected layers. Each layer has an associated weight matrix $\mathbf{W}$ and bias vector $\mathbf{b}$. We use sigmoid activation functions $f(x) = 1/(1+\exp(-x))$, which are applied at all layers of the network, except for the last layer of the encoder and the last layer of the decoder. Other choices of activation function, such as rectified linear units and exponential linear units, may also be used and appear to achieve similar results.

5.3.2 Loss function

The loss function used in training is a weighted sum of four terms: autoencoder reconstruction $\mathcal{L}_{\text{recon}}$, SINDy prediction on the input variables $\mathcal{L}_{d\mathbf{x}/dt}$, SINDy prediction on the encoder variables $\mathcal{L}_{d\mathbf{z}/dt}$, and SINDy coefficient regularization $\mathcal{L}_{\text{reg}}$. For a data set with m input samples, each loss is explicitly defined as follows:

$$\begin{aligned}
\mathcal{L}_{\text{recon}} &= \frac{1}{m}\sum_{i=1}^{m} \|\mathbf{x}_i - \psi(\varphi(\mathbf{x}_i))\|_2^2 \\
\mathcal{L}_{d\mathbf{x}/dt} &= \frac{1}{m}\sum_{i=1}^{m} \left\|\dot{\mathbf{x}}_i - (\nabla_{\mathbf{z}}\psi(\varphi(\mathbf{x}_i)))\left(\boldsymbol{\Theta}(\varphi(\mathbf{x}_i)^T)\boldsymbol{\Xi}\right)\right\|_2^2 \\
\mathcal{L}_{d\mathbf{z}/dt} &= \frac{1}{m}\sum_{i=1}^{m} \left\|\nabla_{\mathbf{x}}\varphi(\mathbf{x}_i)\dot{\mathbf{x}}_i - \boldsymbol{\Theta}(\varphi(\mathbf{x}_i)^T)\boldsymbol{\Xi}\right\|_2^2 \\
\mathcal{L}_{\text{reg}} &= \frac{1}{pd}\|\boldsymbol{\Xi}\|_1 .
\end{aligned}$$

The total loss function is

$$\mathcal{L}_{\text{recon}} + \lambda_1\mathcal{L}_{d\mathbf{x}/dt} + \lambda_2\mathcal{L}_{d\mathbf{z}/dt} + \lambda_3\mathcal{L}_{\text{reg}}. \tag{5.7}$$

$\mathcal{L}_{\text{recon}}$ ensures that the autoencoder can accurately reconstruct the data from the intrinsic coordinates. $\mathcal{L}_{d\mathbf{x}/dt}$ and $\mathcal{L}_{d\mathbf{z}/dt}$ ensure that the discovered SINDy model captures the dynamics of the system by ensuring that the model can predict the derivatives from the data. $\mathcal{L}_{\text{reg}}$ promotes sparsity of the coefficients in the SINDy model.

5.3.3 Computing derivatives

Computing the derivatives of the encoder variables requires propagating derivatives through the network. Our network makes use of an activation function $f(\cdot)$ that operates elementwise. Given an input $\mathbf{x}$, we define the pre-activation values at the jth encoder layer as

$$\mathbf{l}_j = f(\mathbf{l}_{j-1})\mathbf{W}_j + \mathbf{b}_j. \tag{5.8}$$

The first layer applies the weights and biases directly to the input so that

$$\mathbf{l}_0 = \mathbf{x}\mathbf{W}_0 + \mathbf{b}_0. \tag{5.9}$$

The activation function is not applied to the last layer, so for an encoder with L hidden layers the autoencoder variables are defined as

$$\mathbf{z} = f(\mathbf{l}_{L-1})\mathbf{W}_L + \mathbf{b}_L. \tag{5.10}$$

Assuming that derivatives $d\mathbf{x}/dt$ are available or can be computed, derivatives $d\mathbf{z}/dt$ can also be computed:

$$\frac{d\mathbf{z}}{dt} = \left(f'(\mathbf{l}_{L-1}) \circ \frac{d\mathbf{l}_{L-1}}{dt}\right)\mathbf{W}_L$$

with

$$\begin{aligned}
\frac{d\mathbf{l}_j}{dt} &= \left(f'(\mathbf{l}_{j-1}) \circ \frac{d\mathbf{l}_{j-1}}{dt}\right)\mathbf{W}_j \\
\frac{d\mathbf{l}_0}{dt} &= \frac{d\mathbf{x}}{dt}\mathbf{W}_0.
\end{aligned}$$

For the nonlinear pendulum example, we use a second order SINDy model that requires the calculation of second derivatives. Second derivatives can be computed using the following:

$$\begin{aligned}
\frac{d^2\mathbf{z}}{dt^2} &= \left(f''(\mathbf{l}_{L-1}) \circ \frac{d\mathbf{l}_{L-1}}{dt} \circ \frac{d\mathbf{l}_{L-1}}{dt} + f'(\mathbf{l}_{L-1}) \circ \frac{d^2\mathbf{l}_{L-1}}{dt^2}\right)\mathbf{W}_L \\
\frac{d^2\mathbf{l}_j}{dt^2} &= \left(f''(\mathbf{l}_{j-1}) \circ \frac{d\mathbf{l}_{j-1}}{dt} \circ \frac{d\mathbf{l}_{j-1}}{dt} + f'(\mathbf{l}_{j-1}) \circ \frac{d^2\mathbf{l}_{j-1}}{dt^2}\right)\mathbf{W}_j \\
\frac{d\mathbf{l}_0}{dt} &= \frac{d^2\mathbf{x}}{dt^2}\mathbf{W}_0.
\end{aligned}$$

5.3.4 Training procedure

We train multiple models for each of the example systems. Each instance of training has a different random initialization of the network weights. The weight matrices $\mathbf{W}_j$ are initialized using the Xavier initialization: the entries are chosen from a random uniform distribution over $[-\sqrt{6/\alpha}, \sqrt{6/\alpha}]$ where α is the dimension of the input plus the dimension of the output [23]. The bias vectors $\mathbf{b}_j$ are initialized to 0 and the SINDy model coefficients $\boldsymbol{\Xi}$ are initialized so that every entry is 1. We train each model using the Adam optimizer for a fixed number of epochs [40]. The learning rate and number of training epochs for each example are specified in Section 5.4.

To obtain parsimonious dynamical models, we use a sequential thresholding procedure that promotes sparsity on the coefficients in $\boldsymbol{\Xi}$, which represent the dynamics on the latent variables $\mathbf{z}$. Every 500 epochs, we set all coefficients in $\boldsymbol{\Xi}$ with a magnitude of less than 0.1 to 0, effectively removing these terms from the SINDy model. This is achieved by using a mask $\boldsymbol{\Upsilon}$, consisting of 1s and 0s, that determines which terms remain in the SINDy model. Thus the true SINDy terms in the loss function are given by

$$\lambda_1 \frac{1}{m}\sum_{i=1}^{m} \left\|\dot{\mathbf{x}}_i - \left(\nabla_{\mathbf{z}}\psi(\varphi(\mathbf{x}_i))\right)\left(\boldsymbol{\Theta}(\varphi(\mathbf{x}_i)^T)(\boldsymbol{\Upsilon}\circ\boldsymbol{\Xi})\right)\right\|_2^2 + \lambda_2 \frac{1}{m}\sum_{i=1}^{m} \left\|\nabla_{\mathbf{x}}\varphi(\mathbf{x}_i)\dot{\mathbf{x}}_i - \boldsymbol{\Theta}(\varphi(\mathbf{x}_i)^T)(\boldsymbol{\Upsilon}\circ\boldsymbol{\Xi})\right\|_2^2 \tag{5.11}$$

where $\boldsymbol{\Upsilon}$ is passed in separately and not updated by the optimization algorithm. Once a term has been thresholded out during training, it is permanently removed from the SINDy model. Therefore the number of active terms in the SINDy model can only be decreased as training continues. The L_1 regularization on $\boldsymbol{\Xi}$ encourages the model coefficients to decrease in magnitude, which combined with the sequential thresholding produces a parsimonious dynamical model.

While the L_1 regularization penalty on $\boldsymbol{\Xi}$ promotes sparsity in the resulting SINDy model, it also encourages nonzero terms to have smaller magnitudes. This results in a trade-off between accurately reconstructing the dynamics of the system and reducing the magnitude of the SINDy coefficients, where the trade-off is determined by the relative magnitudes of

the loss weight penalties λ_1, λ_2 and the regularization penalty λ_3. The specified training procedure therefore typically results in models with coefficients that are slightly smaller in magnitude than those which would best reproduce the dynamics. To account for this, we add an additional coefficient refinement period to the training procedure. To perform this refinement, we lock in the sparsity pattern in the dynamics by fixing the coefficient mask $\boldsymbol{\Upsilon}$ and continue training for 1000 epochs without the L_1 regularization on $\boldsymbol{\Xi}$. This ensures that the best coefficients are found for the resulting SINDy model and also allows the training procedure to refine the encoder and decoder parameters. This procedure is analagous to running a debiased regression following the use of LASSO to select model terms [81].

Choice of hyperparameters

The training procedure described requires the choice of several hyperparameters: chiefly the number of intrinsic coordinates d and the loss weight penalties $\lambda_1, \lambda_2, \lambda_3$. The choice of these parameters greatly impacts the success of the training procedure. Here we outline some guidelines for choosing these parameters.

The most important hyperparameter choice is the number of intrinsic coordinates d, as this impacts the interpretation of the reduced space and the associated dynamical model. In the examples discussed here we had a good sense of how many coordinates were necessary, but for many systems the choice may not be obvious. To choose d, we suggest first training a standard autoencoder without the associated SINDy model to determine the minimum number of coordinates necessary to reproduce the input data. As simpler models (lower d) are typically easier to interpret, one should start by using the smallest d possible. Once this minimum has been found, the full SINDy autoencoder can be trained. It is possible that more coordinates may be needed to capture the dynamics, in which case the method may accurately reproduce the input but fail to find a good model for the dynamics. In this case d could be increased from the minimum until a suitable dynamical model is found.

The choice of loss weight penalties $\lambda_1, \lambda_2, \lambda_3$ also has a significant impact on the success of the training procedure. The parameter λ_1 determines the importance of the SINDy pre-

diction in the original input space. We generally choose λ_1 to be slightly less than the ratio $\sum_{i=1}^{m} \|\mathbf{x}_i\|_2^2 / \sum_{i=1}^{m} \|\dot{\mathbf{x}}_i\|_2^2$. This slightly prioritizes the reconstruction of $\mathbf{x}$ over prediction of $\dot{\mathbf{x}}$ in the training. This is important to ensure that the autoencoder weights are being trained to reproduce $\dot{\mathbf{x}}$ and that it is the SINDy model that gives an accurate prediction of $\dot{\mathbf{x}}$. We choose λ_2 to be one or two orders of magnitude less than λ_1. If λ_2 is too large, it encourages shrinking the magnitude of $\dot{\mathbf{z}}$ to minimize $\mathcal{L}_{d\mathbf{z}/dt}$; however, having λ_2 nonzero encourages a good prediction by the SINDy model in the $\dot{\mathbf{z}}$ space. The third parameter λ_3 determines the strength of the regularization of the SINDy model coefficients $\boldsymbol{\Xi}$ and thus affects the sparsity of the resulting models. If λ_3 is too large, the model will be too simple and achieve poor prediction; if it is too small, the models will be nonsparse. This loss weight requires the most tuning and should typically be chosen last by trying a range of values and assessing the level of sparsity in the resulting model.

In addition to the hyperparameters used in the neural network training, this method requires a choice of library functions for the SINDy model. The examples shown here use polynomial library terms, along with sine terms in the last two examples. For general systems, the best library functions to use may be unknown and choosing the wrong library functions can obscure the simplest model. A best practice is typically to start with polynomial models, as many common physical models are polynomials representing dominant balance terms. Polynomials can also represent Taylor series approximations for a broad class of smooth functions. For some systems, the choice of library functions can also be informed by application specific knowledge.

5.3.5 Model selection

Random initialization of the NN weights is standard practice for deep learning approaches. This results in the discovery of different models for different instances of training, which necessitates comparison among multiple models. In this work, when considering the success of a resulting model, one must consider the parsimony of the SINDy model, how well the decoder reconstructs the input, and how well the SINDy model captures the dynamics.

To assess model performance, we calculate the relative L_2 error of both the input data $\mathbf{x}$ and its derivative $\dot{\mathbf{x}}$:

$$e_{\mathbf{x}} = \frac{\sum_{i=1}^{m} \|\mathbf{x}_i - \psi(\varphi(\mathbf{x}_i))\|_2^2}{\sum_{i=1}^{m} \|\mathbf{x}_i\|_2^2}, \quad e_{\dot{\mathbf{x}}} = \frac{\sum_{i=1}^{m} \left\|\dot{\mathbf{x}}_i - (\nabla_{\mathbf{z}}\psi(\varphi(\mathbf{x}_i)))\left(\boldsymbol{\Theta}(\varphi(\mathbf{x}_i)^T)\boldsymbol{\Xi}\right)\right\|_2^2}{\sum_{i=1}^{m} \|\dot{\mathbf{x}}_i\|_2^2}. \quad (5.12)$$

These error values capture the decoder reconstruction and the fit of the dynamics, respectively. When considering parsimony, we consider the number of active terms in the resulting SINDy model. While parsimonious models are desirable for ease of analysis and interpretability, a model that is too parsimonious may be unable to fully capture the dynamics. In general, for the examples explored, we find that models with fewer active terms perform better on validation data (lower relative L_2 error $e_{\dot{\mathbf{x}}}$) whereas models with more active terms tend to over-fit the training data. In reporting our results, we also calculate the relative L_2 error in predicting $\dot{\mathbf{z}}$:

$$e_{\dot{\mathbf{z}}} = \frac{\sum_{i=1}^{m} \left\|\nabla_{\mathbf{x}}\varphi(\mathbf{x}_i)\dot{\mathbf{x}}_i - \boldsymbol{\Theta}(\varphi(\mathbf{x}_i)^T)(\boldsymbol{\Upsilon} \circ \boldsymbol{\Xi})\right\|_2^2}{\sum_{i=1}^{m} \|\nabla_{\mathbf{x}}\varphi(\mathbf{x}_i)\dot{\mathbf{x}}_i\|_2^2}.$$

For each example system, we apply the training procedure to ten different initializations of the network and compare the resulting models. For the purpose of demonstration, for each example we show results for a chosen "best" model, which is taken to be the model with the lowest relative L_2 error on validation data among models with the fewest active coefficients. While every instance of training does not result in the exact same SINDy sparsity pattern, the network tends to discover a few different closely related forms of the dynamics. We discuss the comparison among models for each particular example in Section 5.4.

5.4 Results

We demonstrate the success of the proposed method on three example systems: a high-dimensional system with the underlying dynamics generated from the canonical chaotic Lorenz system, a two-dimensional reaction-diffusion system, and a two-dimensional spatial representation (synthetic video) of the nonlinear pendulum. Results are shown in Figure 5.2.

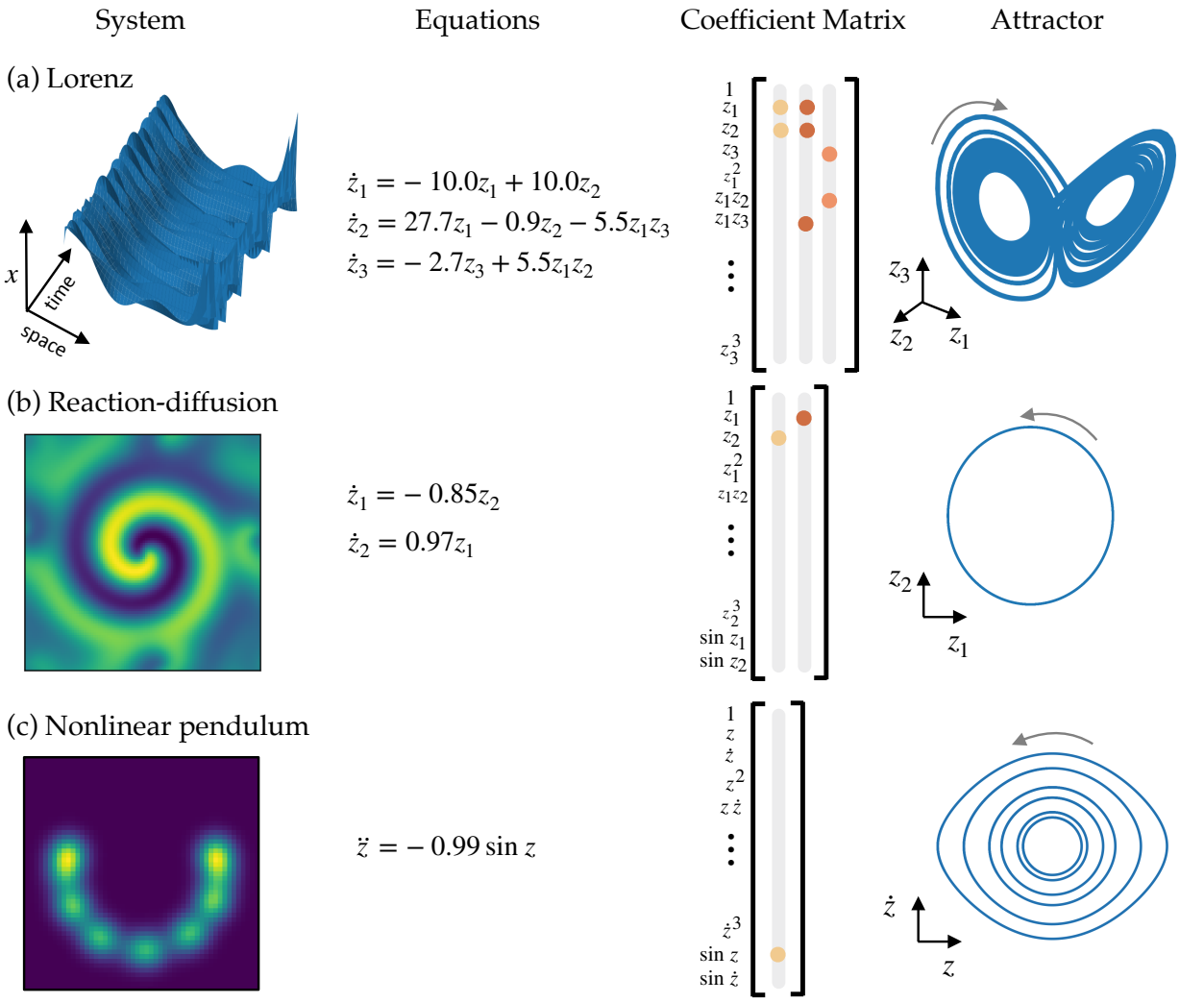

Figure 5.2: Discovered dynamical models for example systems. (a,b,c) Equations, SINDy coefficients $\boldsymbol{\Xi}$, and attractors for the Lorenz system, reaction-diffusion system, and nonlinear pendulum.

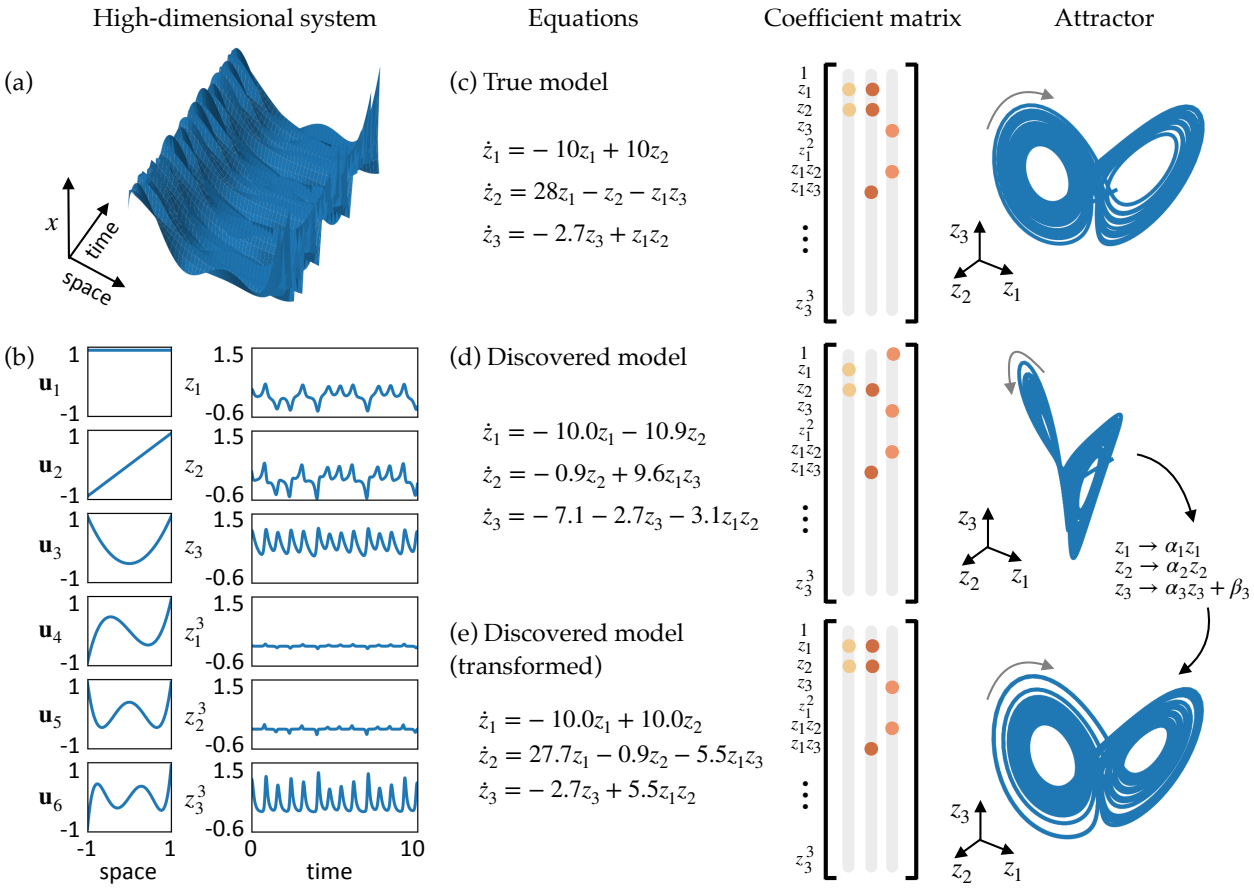

Figure 5.3: Model results on the high-dimensional Lorenz example. (a) Trajectories of the chaotic Lorenz system ($\mathbf{z}(t) \in \mathbb{R}^3$) are used to create a high-dimensional data set ($\mathbf{x}(t) \in \mathbb{R}^{128}$). (b) The spatial modes are created from the first six Legendre polynomials and the temporal modes are the variables in the Lorenz system and their cubes. The spatial and temporal modes are combined to create the high-dimensional data set via (5.14). (c,d) The equations, SINDy coefficients $\boldsymbol{\Xi}$, and attractors for the original Lorenz system and a dynamical system discovered by the SINDy autoencoder. The attractors are constructed by simulating the dynamical system forward in time from a single initial condition. (e) Applying a suitable variable transformation to the system in (d) reveals a model with the same sparsity pattern as the original Lorenz system. The parameters are close in value to the original system, with the exception of an arbitrary scaling, and the attractor has a similar structure to the original system.

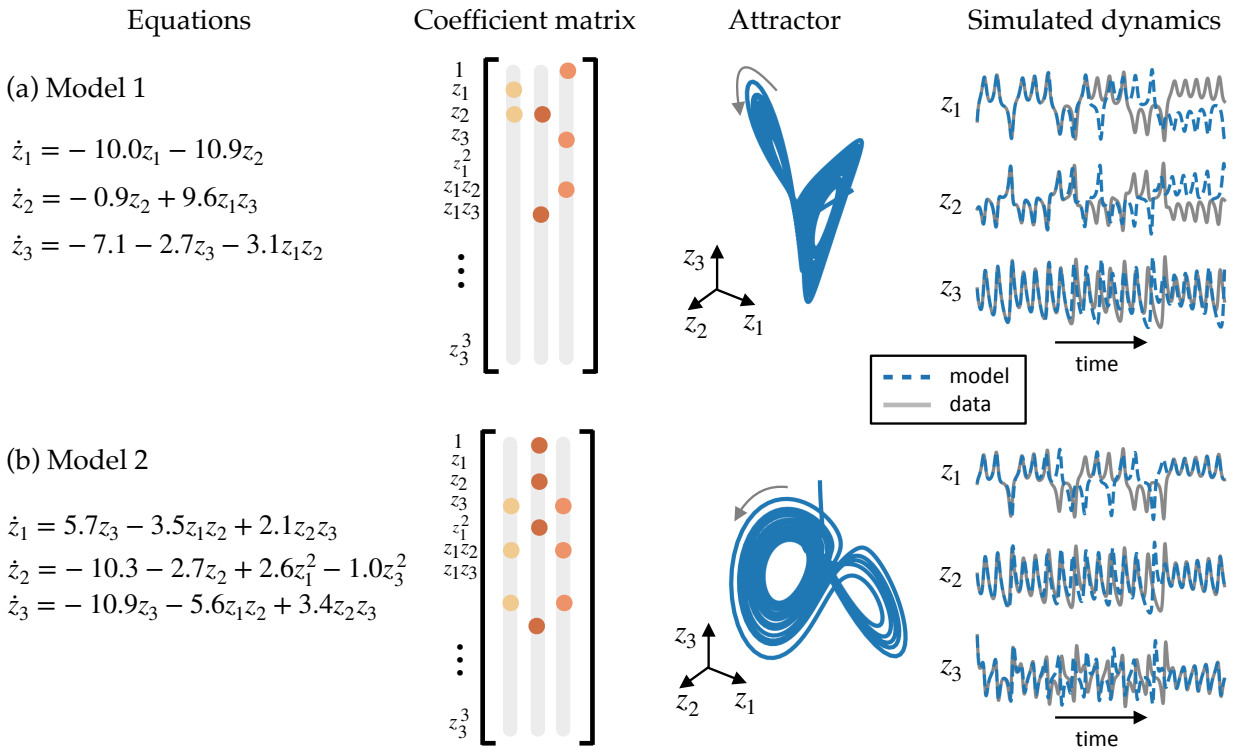

Figure 5.4: Comparison of two discovered models for the Lorenz example system. For both models we show the equations, SINDy coefficients Ξ, attractors, and simulated dynamics for two models discovered by the SINDy autoencoder. (a) A model with 7 active terms. This model can be rewritten in the same form as the original Lorenz system using the variable transformation described in Section 5.4.1. Simulation of the model produces an attractor with a two lobe structure and is able to reproduce the true trajectories of the dynamics for some time before eventually diverging due to the chaotic nature of the system. (b) A model with 10 active terms. The model has more terms than the true Lorenz system, but has a slightly lower relative L_2 error of $\mathbf{x}, \dot{\mathbf{x}}$ than the model in (a). Simulation shows that the dynamics also lie on an attractor with two lobes. The model can accurately predict the true dynamics over a similar duration as (a).

Table 5.1: Hyperparameter values for the Lorenz example

Parameter	**Value**
n	128
d	3
training samples	5.12×10^5
batch size	8000
activation function	sigmoid
encoder layer widths	$64, 32$
decoder layer widths	$32, 64$
learning rate	10^{-3}
SINDy model order	1
SINDy library polynomial order	3
SINDy library includes sine	no
SINDy loss weight $\dot{\mathbf{x}}$, λ_1	10^{-4}
SINDy loss weight $\dot{\mathbf{z}}$, λ_2	0
SINDy regularization loss weight, λ_3	10^{-5}

5.4.1 Chaotic Lorenz system

We first construct a high-dimensional example problem with dynamics based on the chaotic Lorenz system. The Lorenz system is a canonical model used as a test case for many dynamical systems methods, with dynamics given by the following equations

$$\dot{z}_1 = \sigma(z_2 - z_1) \tag{5.13a}$$

$$\dot{z}_2 = z_1(\rho - z_3) - z_2 \tag{5.13b}$$

$$\dot{z}_3 = z_1 z_2 - \beta z_3. \tag{5.13c}$$

The dynamics of the Lorenz system are chaotic and highly nonlinear, making it an ideal test problem for model discovery. We use the standard parameter values of $\sigma = 10, \rho = 28, \beta = 8/3$. To create a high-dimensional data set based on this system, we choose six fixed spatial modes $\mathbf{u}_1, \ldots, \mathbf{u}_6 \in \mathbb{R}^{128}$ and define

$$\mathbf{x}(t) = \mathbf{u}_1 z_1(t) + \mathbf{u}_2 z_2(t) + \mathbf{u}_3 z_3(t) + \mathbf{u}_4 z_1(t)^3 + \mathbf{u}_5 z_2(t)^3 + \mathbf{u}_6 z_3(t)^3. \tag{5.14}$$

We choose our spatial modes $\mathbf{u}_1, \ldots, \mathbf{u}_6$ to be the first six Legendre polynomials defined at 128 grid points on a 1D spatial domain $[-1, 1]$. To generate our data set, we simulate the system from 2048 initial conditions for the training set, 20 for the validation set, and 100 for the test set. For each initial condition we integrate the system forward in time from $t = 0$ to $t = 5$ with a spacing of $\Delta t = 0.02$ to obtain 250 samples. Initial conditions are chosen randomly from a uniform distribution over $z_1 \in [-36, 36]$, $z_2 \in [-48, 48]$, $z_3 \in [-16, 66]$. This results in a training set with 512,000 total samples. The data set is a nonlinear combination of the true Lorenz variables and is shown in Figure 5.3a. The spatial and temporal modes that combine to give the full dynamics are shown in Figure 5.3b.

Following the training procedure described in Section 5.3.4, we learn ten models using the single set of training data (variability among the models comes from the initialization of the network weights). The hyperparameters used for training are shown in Table 5.1. For each model we run the training procedure for 10^4 epochs, followed by a refinement period

of 10^3 epochs. Of the ten models, two have 7 active terms, two have 10 active terms, one has 11 active terms, and five have 15 or more active terms. While all models have less than 1% relative L_2 error for both $\mathbf{x}$ and $\dot{\mathbf{x}}$, the three models with 20 or more active terms have the worst performance predicting $\dot{\mathbf{x}}$. The two models with 10 active terms have the lowest overall error, followed by models with 7, 15, and 18 active terms. While the models with 10 active terms have a lower overall error than the models with 7 terms, both have a very low error and thus we choose to highlight the model with the fewest active terms in Figure 5.3. A model with 10 active terms is shown in Figure 5.4 for comparison.

For analysis, we highlight the model with the lowest error among the models with the fewest active terms. Figure 5.3d shows the dynamical system discovered by the SINDy autoencoder. The discovered model has equations

$$\begin{aligned}
\dot{z}_1 &= -10.0z_1 - 10.9z_2 \\
\dot{z}_2 &= -0.9z_2 + 9.6z_1z_3 \\
\dot{z}_3 &= -7.1 - 2.7z_3 - 3.1z_1z_2.
\end{aligned}$$

While the resulting model does not appear to match the original Lorenz dynamics, the model is parsimonious with only 7 active terms, and with dynamics that live on an attractor which has a two lobe structure similar to that of the original Lorenz attractor. Additionally, we can define an affine transformation that gives it the same structure as the original Lorenz system, demonstrating that the SINDy autoencoder is able to recover the correct sparsity pattern of the dynamics. The variable transformation $z_1 = \alpha_1\tilde{z}_1$, $z_2 = \alpha_2\tilde{z}_2$, $z_3 = \alpha_3\tilde{z}_3 + \beta_3$

gives the following transformed system of equations:

$$\begin{aligned}
\dot{\tilde{z}}_1 &= \frac{1}{\alpha_1}\left(-10.0\alpha_1\tilde{z}_1 - 10.9\alpha_2\tilde{z}_2\right) \\
&= -10.0\tilde{z}_1 - 10.9\frac{\alpha_2}{\alpha_1}\tilde{z}_2 \\
\dot{\tilde{z}}_2 &= \frac{1}{\alpha_2}\left(-0.9\alpha_2\tilde{z}_2 + 9.6\alpha_1\tilde{z}_1(\alpha_3\tilde{z}_3 + \beta_3)\right) \\
&= 9.6\frac{\alpha_1}{\alpha_2}\beta_3\tilde{z}_1 - 0.9\tilde{z}_2 + 9.6\frac{\alpha_1\alpha_3}{\alpha_2}\tilde{z}_1\tilde{z}_3 \\
\dot{\tilde{z}}_3 &= \frac{1}{\alpha_3}\left(-7.1 - 2.7(\alpha_3\tilde{z}_3 + \beta_3) - 3.1\alpha_1\alpha_2\tilde{z}_1\tilde{z}_2\right) \\
&= \frac{1}{\alpha_3}(-7.1 - 2.7\beta_3) - 2.7\tilde{z}_3 - 3.1\frac{\alpha_1\alpha_2}{\alpha_3}\tilde{z}_1\tilde{z}_2.
\end{aligned}$$

By choosing $\alpha_1 = 1$, $\alpha_2 = -0.917$, $\alpha_3 = 0.524$, $\beta_3 = -2.665$, the system becomes

$$\dot{\tilde{z}}_1 = -10.0\tilde{z}_1 + 10.0\tilde{z}_2 \tag{5.15a}$$

$$\dot{\tilde{z}}_2 = 27.7\tilde{z}_1 - 0.9\tilde{z}_2 - 5.5\tilde{z}_1\tilde{z}_3 \tag{5.15b}$$

$$\dot{\tilde{z}}_3 = -2.7\tilde{z}_3 + 5.5\tilde{z}_1\tilde{z}_2. \tag{5.15c}$$

This has the same form as the original Lorenz equations with parameters that are close in value, apart from an arbitrary scaling that affects the magnitude of the coefficients of $\tilde{z}_1\tilde{z}_3$ in (5.15b) and $\tilde{z}_1\tilde{z}_2$ in (5.15c). The attractor dynamics for this system are very similar to the original Lorenz attractor and are shown in Figure 5.3c.

On a test data set of trajectories from 100 randomly chosen initial conditions, the relative L_2 error the decoder reconstruction is less than 3×10^{-5}. The relative L_2 error in predicting the derivatives $\dot{\mathbf{x}}$ and $\dot{\mathbf{z}}$ are 2×10^{-4} and 7×10^{-4}, respectively. For initial conditions within the training distribution, simulations of the resulting SINDy model provide a good prediction of the dynamics over the duration of trajectories in the training data. Over longer durations, the trajectories start to diverge from the true trajectories. This result is not surprising due to the chaotic nature of the Lorenz system and its sensitivity to initial conditions. However, the dynamics of the discovered system match the sparsity pattern of the Lorenz system and the form of the attractor. Improved prediction over a longer duration could be achieved by increased parameter refinement or training the system with longer trajectories.

The learning procedure discovers a dynamical model by fitting coefficients that predict the continuous-time derivatives of the variables in a dynamical system. Thus it is possible for the training procedure to discover a model with unstable dynamics or which is unable to predict the true dynamics through simulation. We assess the validity of the discovered models by simulating the dynamics of the discovered low-dimensional dynamical system. Simulation of the system shows that the system is stable with trajectories existing on an attractor very similar to the original Lorenz attractor. Additionally, the discovered system is able to predict the dynamics of the original system. The fourth panel in Figure 5.4a shows the trajectories found by stepping the discovered model forward in time as compared with the values of $\mathbf{z}$ obtained by mapping samples of the high-dimensional data through the encoder. Although this is done on a new initial condition, the trajectories match very closely up to $t = 5$, which is the duration of trajectories contained in the training set. After that the trajectories diverge, but the predicted trajectories remain on an attractor. The Lorenz dynamics are chaotic, and thus slight differences in coefficients or initial conditions cause trajectories to diverge quickly.

For comparison, in Figure 5.4b we show a second model discovered by the training procedure. This model has 10 active terms, as compared with 7 in the true Lorenz system. While the model contains additional terms not present in the original system, the dynamics lie on an attractor with a similar two lobe structure. Additionally, the system is able to predict the dynamics through simulation. This model has a lower error on test data than the original 7 term model, with relative L_2 errors $e_{\mathbf{x}} \approx 2 \times 10^{-6}$, $e_{\dot{\mathbf{x}}} \approx 6 \times 10^{-5}$, and $e_{\dot{\mathbf{z}}} \approx 3 \times 10^{-4}$.

5.4.2 Reaction-diffusion

In practice, many high-dimensional data sets of interest come from dynamics governed by partial differential equations (PDEs) with more complicated interactions between spatial and temporal dynamics. To test the method on data generated by a PDE, we consider a

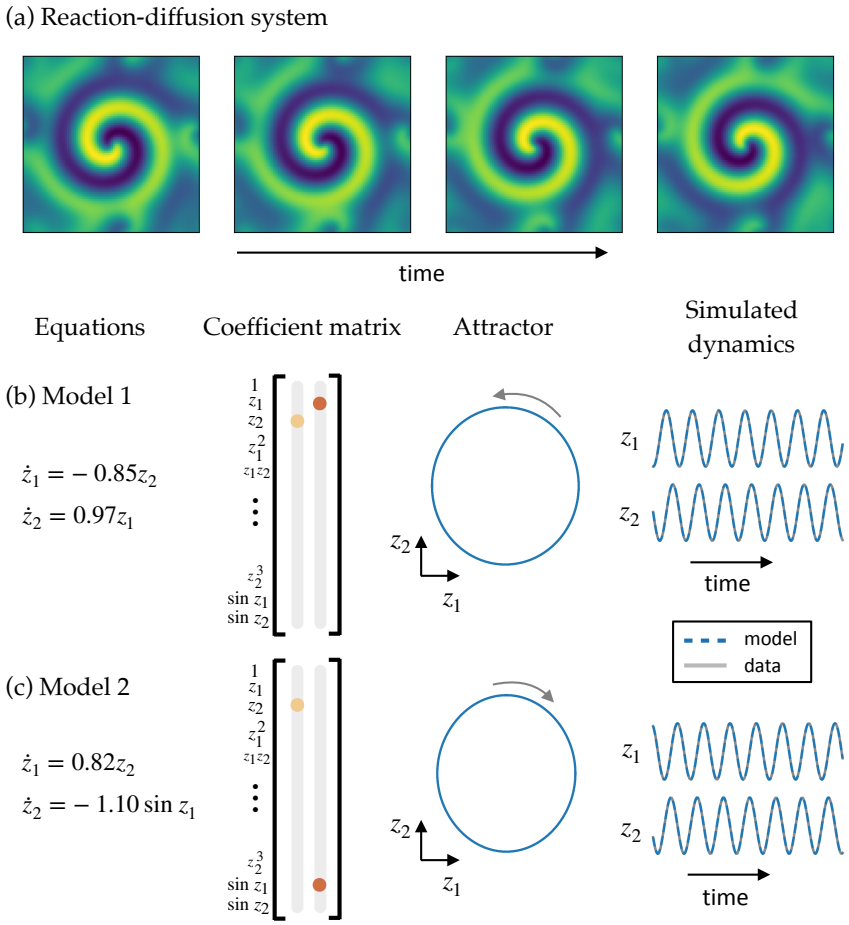

Figure 5.5: Resulting models for the reaction-diffusion system. (a) Snapshots of the high-dimensional system show a spiral wave formation. (b,c) Equations, SINDy coefficients $\boldsymbol{\Xi}$, attractors, and simulated dynamics for two models discovered by the SINDy autoencoder. The model in (b) is a linear oscillation, whereas the model in (c) is a nonlinear oscillation. Both models achieve similar error levels and can predict the dynamics in the test set via simulation of the low-dimensional dynamical system.

Table 5.2: Hyperparameter values for the reaction-diffusion example

Parameter	**Value**
n	10^4
d	2
training samples	8000
batch size	1024
activation function	sigmoid
encoder layer widths	256
decoder layer widths	256
learning rate	10^{-3}
SINDy model order	1
SINDy library polynomial order	3
SINDy library includes sine	yes
SINDy loss weight $\dot{\mathbf{x}}$, λ_1	0.5
SINDy loss weight $\dot{\mathbf{z}}$, λ_2	0.01
SINDy regularization loss weight, λ_3	0.1

lambda-omega reaction-diffusion system governed by

$$u_t = (1 - (u^2 + v^2))u + \beta(u^2 + v^2)v + d_1(u_{xx} + u_{yy})$$
$$v_t = -\beta(u^2 + v^2)u + (1 - (u^2 + v^2))v + d_2(v_{xx} + v_{yy})$$

with $d_1, d_2 = 0.1$ and $\beta = 1$. To generate data, the system is simulated from a single initial condition from $t = 0$ to $t = 10$ with a spacing of $\Delta t = 0.05$ for a total of 10,000 samples. The initial condition is defined as

$$u(y_1, y_2, 0) = \tanh\left(\sqrt{y_1^2 + y_2^2}\cos\left(\angle(y_1 + iy_2) - \sqrt{y_1^2 + y_2^2}\right)\right)$$
$$v(y_1, y_2, 0) = \tanh\left(\sqrt{y_1^2 + y_2^2}\sin\left(\angle(y_1 + iy_2) - \sqrt{y_1^2 + y_2^2}\right)\right)$$

over a spatial domain of $y_1 \in [-10, 10],\ y_2 \in [-10, 10]$ discretized on a grid with 100 points on each spatial axis. This results in a high-dimensional input data set with $n = 10^4$. The solution of these equations produces a spiral wave formation, whose behavior can be approximately captured by two oscillating spatial modes. We apply our method to snapshots of $u(y_1, y_2, t)$ generated by the above equations, multiplied by a Gaussian $f(y_1, y_2) = \exp(-0.1(y_1^2 + y_2^2))$ centered at the origin to localize the spiral wave in the center of the domain. Our input data is thus defined as $\mathbf{x}(t) = f(:, :) \circ u(:, :, t) \in \mathbb{R}^{10^4}$. We also add Gaussian noise with a standard deviation of 10^{-6} to both $\mathbf{x}$ and $\dot{\mathbf{x}}$. Four time snapshots of the input data are shown in Figure 5.5a.

We divide the total number of samples into training, validation, and test sets: the last 1000 samples are taken as the test set, 1000 samples are chosen randomly from the first 9000 samples as a validation set, and the remaining 8000 samples are taken as the training set. Using $d = 2$, we train ten models using the procedure outlined in Section 5.3.4 for 3×10^3 epochs followed by a refinement period of 10^3 epochs. Hyperparameters used for training are shown in Table 5.2. Nine of the ten resulting dynamical systems models have two active terms and one has three active terms. The dynamical equations, SINDy coefficient matrix, attractors, and simulated dynamics for two example models are shown in Figure 5.5b,c. The models with two active terms all have one of the two forms shown in the figure: three models

Table 5.3: Hyperparameter values for the nonlinear pendulum example

Parameter	**Value**
n	2601
d	1
training samples	5×10^4
batch size	1024
activation function	sigmoid
encoder layer widths	$128, 64, 32$
decoder layer widths	$32, 64, 128$
learning rate	10^{-4}
SINDy model order	2
SINDy library polynomial order	3
SINDy library includes sine	yes
SINDy loss weight $\dot{x}$, λ_1	5×10^{-4}
SINDy loss weight $\dot{z}$, λ_2	5×10^{-5}
SINDy regularization loss weight, λ_3	10^{-5}

have a linear oscillation and six models have a nonlinear oscillation. Both model forms have similar levels of error on the test set and are able to predict the dynamics in the test set from simulation, as shown in the fourth panel of Figure 5.5b,c.

For the discovered model shown in Figure 5.2b, the relative L_2 error for the input data $\mathbf{x}$ and the input derivatives $\dot{\mathbf{x}}$ is 0.016. The relative L_2 error for $\dot{\mathbf{z}}$ is 0.002. Simulation of the dynamical model accurately captures the low dimensional dynamics, with relative L_2 error of $\mathbf{z}$ totaling 1×10^{-4}..

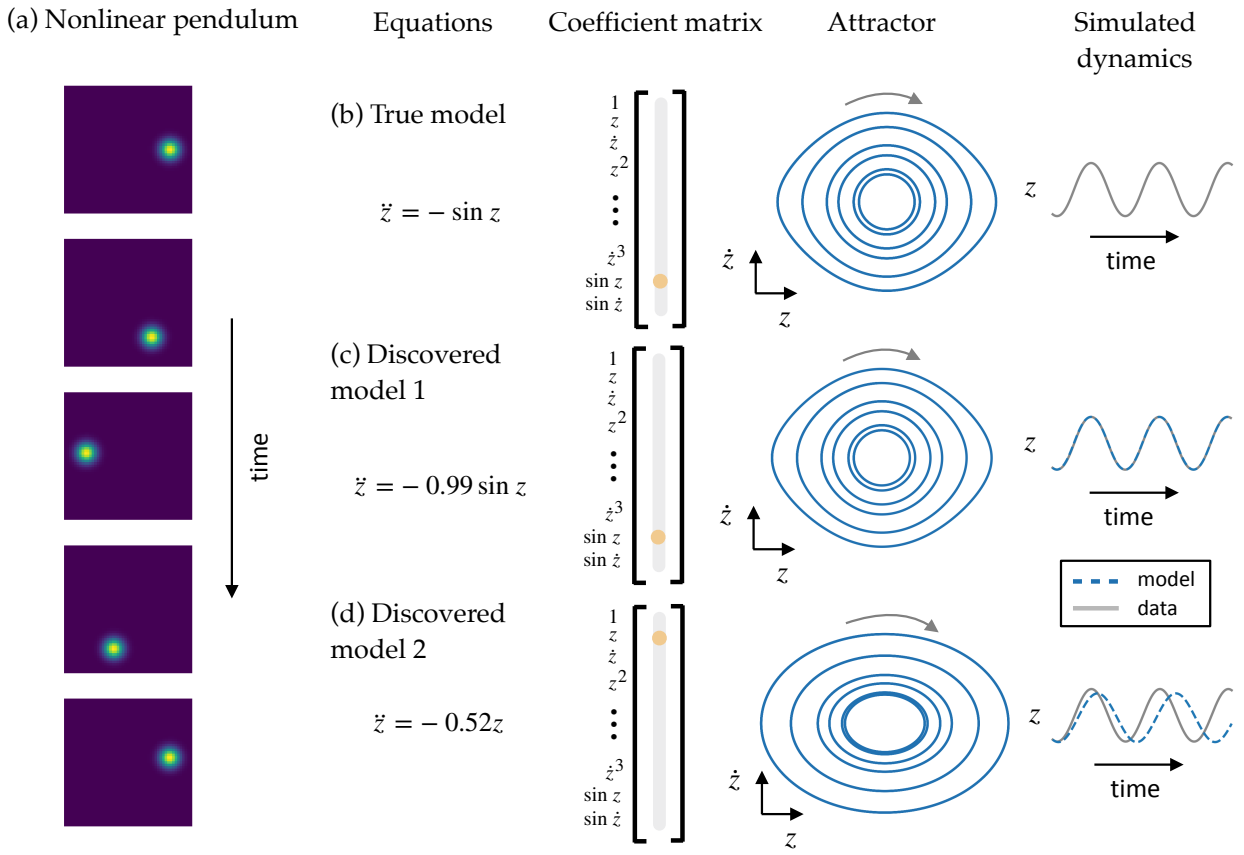

Figure 5.6: Resulting models for the nonlinear pendulum. (a) Snapshots of the high-dimensional system are images representing the position of the pendulum in time. (b,c,d) Comparison of two discovered models with the true pendulum dynamics. Equations, SINDy coefficients $\boldsymbol{\Xi}$, attractors, and simulated dynamics for the true pendulum equation are shown in (b). The model in (c) correctly discovered the true form of the pendulum dynamics. Both the image of the attractor and simulations match the true dynamics. (d) In one instance of training, the SINDy autoencoder discovered a linear oscillation for the dynamics. This model achieves a worse error than the model in (c).

5.4.3 Nonlinear pendulum

As a final example, we consider simulated video of a nonlinear pendulum in two spatial dimensions. The nonlinear pendulum is governed by the following second order differential equation:

$$\ddot{z} = -\sin z. \tag{5.16}$$

We generate synthetic video of the pendulum in two spatial dimensions by creating a series of high-dimensional snapshot images with a two-dimensional Gaussian centered at the center of mass, determined by the pendulum's angle z. The images are given by

$$x(y_1, y_2, t) = \exp\left(-20\big((y_1 - \cos(z(t) - \pi/2))^2 + (y_2 - \sin(z(t) - \pi/2))^2\big)\right) \tag{5.17}$$

at a discretization of $y_1, y_2 \in [-1.5, 1.5]$. We use 51 grid points in each dimension resulting in snapshots $\mathbf{x}(t) \in \mathbb{R}^{2601}$. To generate a training set, we simulate (5.16) from 100 randomly chosen initial conditions with $z(0) \in [-\pi, \pi]$ and $\dot{z}(0) \in [-2.1, 2.1]$. The initial conditions are selected from a uniform distribution in the specified range but are restricted to conditions for which the pendulum does not have enough energy to do a full loop. This condition is determined by checking that $|\dot{z}(0)^2/2 - \cos z(0)| \leq 0.99$.

This series of images is the high-dimensional data input to the autoencoder. Despite the fact that the position of the pendulum can be represented by a simple one-dimensional variable, methods such as PCA are unable to obtain a low-dimensional representation of this data set. A nonlinear autoencoder, however, is able to discover a one-dimensional representation of the data set.

For this example, we use a second-order SINDy model: that is, we use a library of functions including the first derivatives $\dot{\mathbf{z}}$ to predict the second derivative $\ddot{\mathbf{z}}$. This approach is the same as with a first order SINDy model but requires estimates of the second derivatives in addition to estimates of the first derivatives. Second order gradients of the encoder and decoder are therefore also required. Computation of the derivatives is discussed in Section 5.3.3.

Following the training procedure outlined in Section 5.3.4 and using $d = 1$, we train ten models for 5×10^3 epochs followed by a refinement period of 10^3 epochs. Hyperparameters used for this example are shown in Table 5.3. Of the ten training instances, five correctly identify the nonlinear pendulum equation. These five models have the best performance of the ten models. The attractor and simulated dynamics for the best of these five models are shown in Figure 5.6. We calculate test error on trajectories from 50 randomly chosen initial conditions. The best model has a relative L_2 error of 8×10^{-4} for the decoder reconstruction of the input $\mathbf{x}$. The relative L_2 error of the SINDy model predictions for $\ddot{\mathbf{x}}$ and $\ddot{\mathbf{z}}$ are 3×10^{-4} and 2×10^{-2}, respectively.

One model, also shown in Figure 5.6, recovers a linear oscillator. This model is able to achieve a reasonably low prediction error for $\ddot{\mathbf{x}}, \ddot{\mathbf{z}}$ but the simulated dynamics, while still oscillatory, appear qualitatively different from the true pendulum dynamics. The four remaining models all have two active terms in the dynamics and have a worse performance than the models with one active term.

5.5 Conclusions

We have presented a data-driven method for discovering interpretable, low-dimensional dynamical models and their associated coordinates for high-dimensional dynamical systems. The simultaneous discovery of both is critical for generating dynamical models that are sparse, and hence interpretable and generalizable. Our approach takes advantage of the power of NNs by using a flexible autoencoder architecture to discover nonlinear coordinate transformations that enable the discovery of parsimonious, nonlinear governing equations. This work addresses a major limitation of prior approaches for the discovery of governing equations, which is that the proper choice of measurement coordinates is often unknown. We demonstrate this method on three example systems, showing that it is able to identify coordinates associated with parsimonious dynamical equations. For the examples studied, the identified models are interpretable and can be used for forecasting (extrapolation) applications (see Figure 5.4).

A current limitation of our approach is the requirement for clean measurement data that is approximately noise-free. Fitting a continuous-time dynamical system with SINDy requires reasonable estimates of the derivatives, which may be difficult to obtain from noisy data. While this represents a challenge, approaches for estimating derivatives from noisy data such as the total variation regularized derivative can prove useful in providing derivative estimates [15]. Moreover, there are emerging NN architectures explicitly constructed for separating signals from noise [72], which can be used as a pre-processing step in the data-driven discovery process advocated here. Alternatively our method can be used to fit a discrete-time dynamical system, in which case derivative estimates are not required. Many methods for modeling dynamical systems work in discrete time rather than continuous time, making this a reasonable alternative. It is also possible to use the integral formulation of SINDy to abate noise sensitivity [74].

A major problem with deep learning approaches is that models are typically neither interpretable nor generalizable. Specifically, NNs trained solely for prediction may fail to generalize to classes of behaviors not seen in the training set. We have demonstrated an approach for using NNs to obtain classically interpretable models through the discovery of low-dimensional dynamical systems, which are well-studied and often have physical interpretations. Once the proper terms in the governing equations are identified, the discovered model can be generalized to study other parameter regimes of the dynamics. While the coordinate transformation learned by the autoencoder may not generalize to data regimes far from the original training set, if the dynamics are known, the network can be retrained on new data with fixed terms in the latent dynamics space. The problem of relearning a coordinate transformation for a system with known dynamics is greatly simplified from the original challenge of learning the correct form of the underlying dynamics without knowledge of the proper coordinate transformation.

The challenge of utilizing NNs to answer scientific questions requires careful consideration of their strengths and limitations. While advances in deep learning and computing power present a tremendous opportunity for new scientific breakthroughs, care must be taken to

ensure that valid conclusions are drawn from the results. One promising strategy is to combine machine learning approaches with well-established domain knowledge: for instance physics-informed learning leverages physical assumptions into NN architectures and training methods. Methods that provide interpretable models have the potential to enable new discoveries in data-rich fields. This work introduced a flexible framework for using NNs to discover models that are interpretable from a standard dynamical systems perspective. In the future, this approach could be adapted using domain knowledge to discover new models in specific fields.

Chapter 6

A UNIFIED OPTIMIZATION FRAMEWORK FOR SINDY

The SINDy approach has been shown to produce interpretable and generalizable dynamical systems models from limited data, having been applied broadly to identify models for optical systems [78], fluid flows [51], chemical reaction dynamics [32], plasma convection [17], structural modeling [46], and for model predictive control [38]. It is also possible to extend SINDy to identify partial differential equations [71, 73], to trim corrupt data [82], and to incorporate partially known physics and constraints [51]. These diverse mathematical developments provide a mature framework for broadening the applicability of the model discovery method.

While the SINDy modeling framework is remarkably flexible, the approach has typically used a sequentially thresholded least squares algorithm to discover the resulting models. Although this algorithm works remarkably well, it is customized to the least squares formulation and does not readily accommodate extensions such as incorporation of additional constraints, robust formulations, or nonlinear parameter estimation. A number of the extensions to SINDy cited above have required adaptations to the algorithm [82, 71, 73, 51, 38]. However, because the SINDy framework is fundamentally based on a sparsity-regularized regression, there is an opportunity to unify these innovations via the sparse relaxed regularized regression (SR3) [96], resulting in a unified sparse model discovery framework. The SR3 framework relaxes the initial regularized optimization problem, allowing for better performance. In this chapter, we develop a unified sparse optimization framework for dynamical system discovery. This puts the SINDy problem into an optimization framework that enables straightforward extensions to the original problem.

In this chapter, we introduce the SINDy SR3 framework and compare its performance

against both the standard sequential thresholded least squares (STLSQ) algorithm and the widely used LASSO approach for sparse regression. We show that SR3 has similar performance to STLSQ, with the advantage that it formulates SINDy model fitting as an optimization problem. The performance of both these approaches is superior to LASSO for coefficient selection. We also demonstrate the flexibility of this approach by implementing two extensions to the SINDy problem within the SR3 framework: trimming of outliers and incorporating parametric dependencies.

6.1 Optimization framework: SINDy SR3

We extend the SINDy problem to include additional structure, robustness to outliers, and nonlinear parameter estimation using the sparse relaxed regularized regression (SR3) approach that uses relaxation and partial minimization [96]. As introduced in Chapter 2, the sparse identification of nonlinear dynamics (SINDy) method [12] seeks a solution of

$$\dot{\mathbf{X}} = \boldsymbol{\Theta}(\mathbf{X})\boldsymbol{\Xi}. \tag{6.1}$$

where $\boldsymbol{\Xi} = (\boldsymbol{\xi}_1\ \boldsymbol{\xi}_2\ \cdots\ \boldsymbol{\xi}_n)$ are sparse coefficient (loading) vectors. A natural optimization is given by

$$\min_{\boldsymbol{\Xi}} \frac{1}{2}\|\dot{\mathbf{X}} - \boldsymbol{\Theta}(\mathbf{X})\boldsymbol{\Xi}\|^2 + \lambda R(\boldsymbol{\Xi}) \tag{6.2}$$

where $R(\cdot)$ is a regularizer that promotes sparsity. When R is convex, a range of well-known algorithms for (6.2) are available. The standard approach is to choose R to be the sparsity-promoting ℓ_1 norm, which is the convex relaxation of ℓ_0 norm. In this case, SINDy is solved via LASSO [81]. In practice, LASSO does not perform well at coefficient selection (see Section 6.1.1 for details). In the context of dynamics discovery we would like to use non-convex R, specifically the ℓ_0 norm.

Rather than solving the optimization problem (6.2), the standard SINDy algorithm performs sequential thresholded least squares (STLSQ). The process is as follows: given a parameter η that specifies the minimum magnitude for a coefficient in $\boldsymbol{\Xi}$, do a least squares fit and then zero out all coefficients with magnitude below the threshold; repeat the process

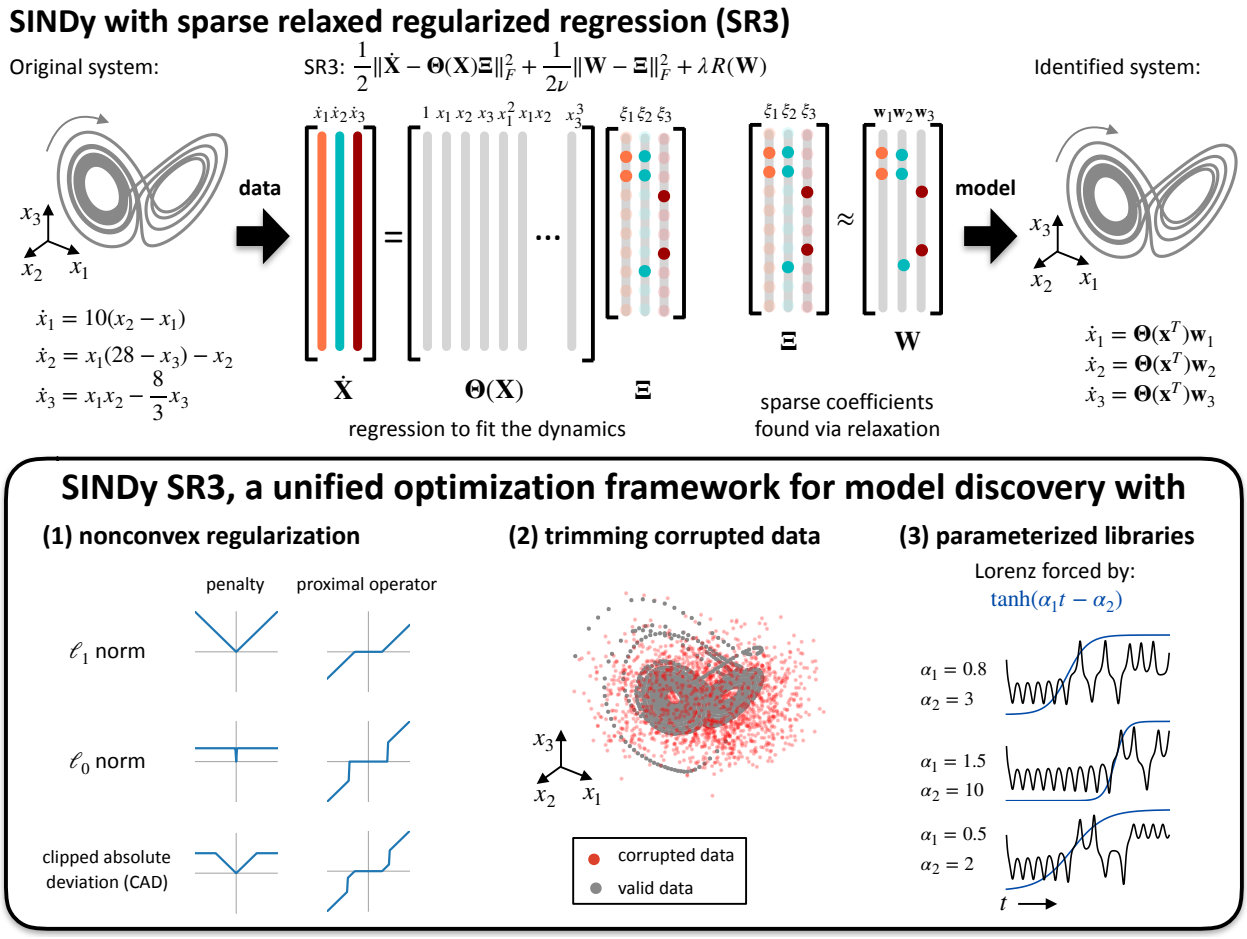

Figure 6.1: Overview of the SINDy method with SR3 for identifying nonlinear dynamical systems. SINDy sets up the system identification problem as a sparse regression problem, selecting a set of active governing terms from a library. Sparse relaxed regularized regression (SR3) provides a flexible, unified framework that can be adapted to address a number of challenges that might occur with data from physical systems, including outlier identification, parameterized library functions, and forcing.

Algorithm 1 Basic SR3 Algorithm

Input ϵ, $\mathbf{W}^0$

Initialize $k = 0$, err$= 2\epsilon$.

while err$> \epsilon$ **do**

$k \leftarrow k + 1$

$\boldsymbol{\Xi}^k = \underset{\boldsymbol{\Xi}}{\operatorname{argmin}} \frac{1}{2}\|\dot{\mathbf{X}} - \boldsymbol{\Theta}(\mathbf{X})\boldsymbol{\Xi}\|^2 + \frac{1}{2\nu}\|\boldsymbol{\Xi} - \mathbf{W}^{k-1}\|^2$

$\mathbf{W}^k = \operatorname{prox}_{\lambda\nu R}(\boldsymbol{\Xi}^k)$

err $= \|\mathbf{W}^k - \mathbf{W}^{k-1}\|/\nu$

end while

of fitting and thresholding until convergence. The STLSQ algorithm performs better than LASSO at coefficient selection, but it is an *algorithm* rather than an optimization problem and thus has limited flexibility.

SR3 for (6.2) introduces the auxiliary variable $\mathbf{W}$ and relaxes the optimization to

$$\min_{\boldsymbol{\Xi},\mathbf{W}} \frac{1}{2}\|\dot{\mathbf{X}} - \boldsymbol{\Theta}(\mathbf{X})\boldsymbol{\Xi}\|^2 + \lambda R(\mathbf{W}) + \frac{1}{2\nu}\|\boldsymbol{\Xi} - \mathbf{W}\|^2. \tag{6.3}$$

We can solve (6.3) using the alternating update rule in Algorithm 1. This requires only least squares solves and prox operators [96]. The resulting solution approximates the original problem (6.2) as $\nu \downarrow 0$. When R is taken to be the ℓ_0 penalty, the prox operator is hard thresholding, and Algorithm 1 is similar, but not equivalent, to thresholded least squares, and performs similarly in practice. However, unlike thresholded least squares, the SR3 approach easily generalizes to new problems and features. Figure 6.1 show a visual depiction of the SR3 approach for SINDy.

6.1.1 Performance of SR3 for SINDy

SR3 for SINDy provides an optimization framework that both (1) enables the identification of truly sparse models and (2) can be adapted to include additional features. We first compare SR3 to both STLSQ and the LASSO algorithm. While STLSQ works well for identifying

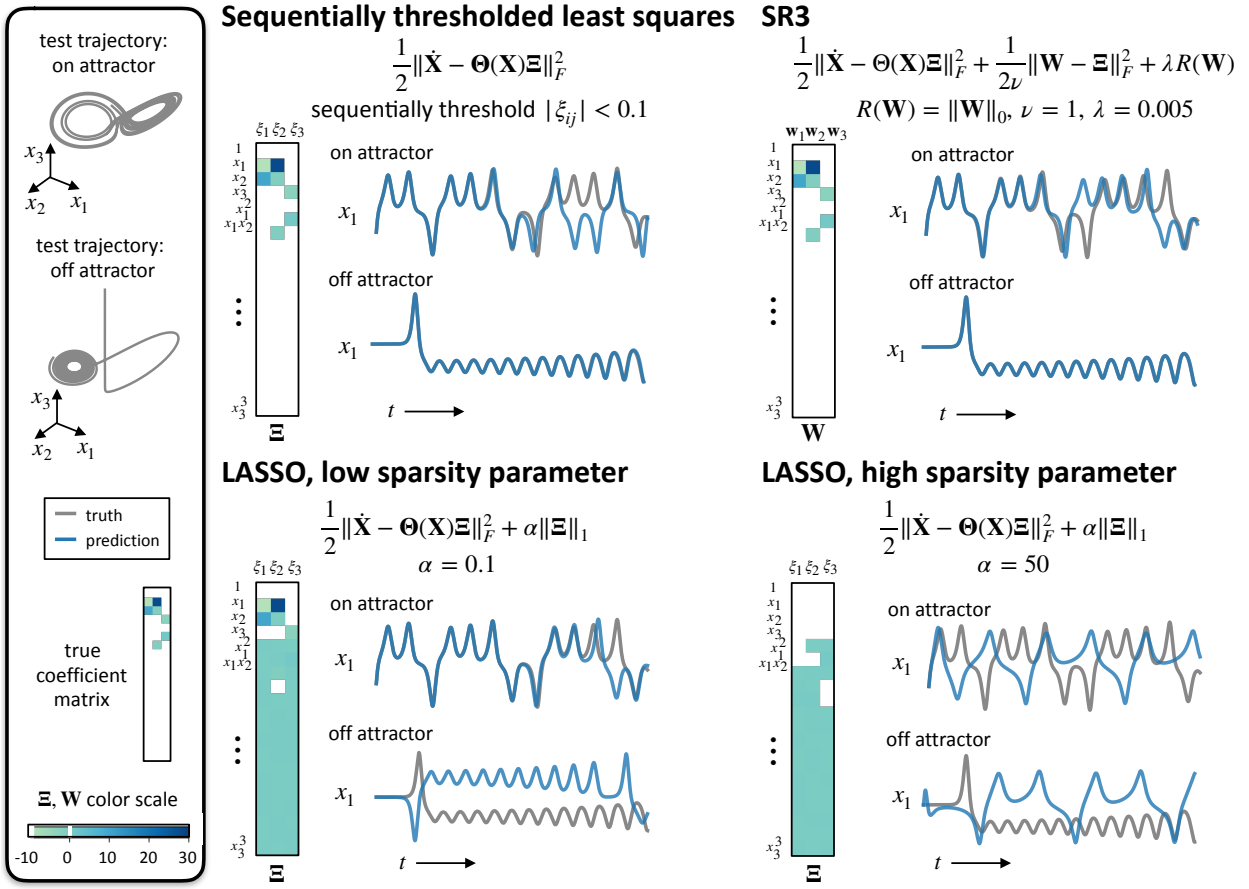

Figure 6.2: Comparison of optimization methods for identifying the active coefficients in a SINDy model. The standard approach has been STLSQ, which is able to identify a sparse model that fits the data well. However, this approach lacks flexibility and is not easily adapted to incorporate other optimization challenges. LASSO is a standard approach for performing a sparse regression, but does not do well at performing coefficient selection: many of the terms in the coefficient matrix are small but nonzero. Increasing the regularization strength leads to a model that is still not sparse and has a poor fit of the data. SR3 relaxes the regression problem in a way that enables the use of nonconvex regularization functions such as the ℓ_0 norm or hard thresholding. This results in a truly sparse model, and provides a flexible framework that can easily incorporate additional optimizations such as trimming outliers and fitting parameterized library functions.

sparse models that capture the behavior of a system, it is a standalone method without a true optimization cost function, meaning the algorithm must be reformulated to work with other adaptations to the SINDy problem [54]. LASSO provides a standard optimization approach but does not successfully identify sparse models. Even with clean data, LASSO models for SINDy typically have many coefficients that are small in magnitude but nonzero. Obtaining a sparse set of coefficients is key for interpretability. SR3 works with nonconvex regularization functions such as the ℓ_0 norm, enabling the identification of truly sparse models.

In Fig. 6.2 we compare these algorithms using data from the canonical chaotic Lorenz system:

$$\dot{x}_1 = 10(x_2 - x_1), \quad \dot{x}_2 = x_1(28 - x_3) - x_2, \quad \dot{x}_3 = x_1 x_2 - (8/3)x_3. \tag{6.4}$$

We simulate the system from 20 initial conditions and fit a SINDy model with polynomials up to order 3 using the following optimization approaches: STLSQ with threshold 0.1, SR3 with ℓ_0 regularization, LASSO with a regularization weight of 0.1, and LASSO with a regularization weight of 50. For each model we analyze (1) the sparsity pattern of the coefficient matrix and (2) simulations of the resulting model on test trajectories. As shown in Figure 6.2, STLSQ and SR3 yield the same correct sparsity pattern. In simulation, both track a Lorenz test trajectory for several trips around the attractor before eventually falling off. The eventual deviation is expected due to the chaotic nature of the Lorenz system, as a slight difference in coefficient values or initial conditions can lead to vastly different trajectories (although the trajectories continue to fill in the Lorenz attractor). These models also track the behavior well for a trajectory that starts off the attractor. The LASSO models both have many terms that are small in magnitude but still nonzero. As the regularization penalty is increased, rather than removing the unimportant terms in the dynamics the method removes many of the true coefficients in the Lorenz model. The LASSO model with heavy regularization has a very poor fit for the dynamics, as seen via simulation. While the LASSO model with less regularization provides a good fit for the dynamics on the attractor, it does not generalize off the attractor.

6.1.2 Choice of parameters for SR3

The SR3 algorithm requires the specification of two parameters, ν and λ. The parameter ν controls how closely the relaxed coefficient matrix $\mathbf{W}$ matches $\boldsymbol{\Xi}$: small values of ν encourage $\mathbf{W}$ to be a close match for $\boldsymbol{\Xi}$, whereas larger values will allow $\mathbf{W}$ to be farther from $\boldsymbol{\Xi}$.

The parameter λ determines the strength of the regularization. If the regularization function is the ℓ_0 norm, the parameter λ can be chosen to correspond to the coefficient threshold used in the sequentially thresholded least squares algorithm (which determines the lowest magnitude value in the coefficient matrix). This is because the prox function for the ℓ_0 norm will threshold out coefficients below a value determined by ν and λ. In particular, if the desired coefficient threshold is η, we can take

$$\lambda = \frac{\eta^2}{2\nu}$$

and the prox update will threshold out values below η. In the examples shown here, we determine λ in this manner based on the desired values for ν, η. If the desired coefficient threshold is known (which is the case for the examples studied here, but may not be the case for unknown systems), this gives us a single parameter to adjust: ν. With λ defined in this manner, decreasing ν provides more weight to the regularization, whereas increasing ν provides more weight to the least squares model fit.

6.2 Simultaneous Sparse Inference and Data Trimming

Many real world data sets contain corrupted data and/or outliers, which is problematic for model identification methods. For SINDy, outliers can be especially problematic, as derivative computations are corrupted. Many data modeling methods have been adapted to deal with corrupted data, resulting in "robust" versions of the methods (such as robust PCA). The SR3 algorithm for SINDy can be adapted to incorporate trimming of outliers, providing a robust optimization algorithm for SINDy. Starting with least trimmed squares [69], extended formulations that simultaneously fit models and trim outliers are widely used in statistical

learning. Trimming has proven particularly useful in the high-dimensional setting when used with the LASSO approach and its extensions [91, 92].

The high-dimensional trimming extension applied to (6.2) takes the form

$$\min_{\boldsymbol{\Xi},\mathbf{v}} \sum_{i=1}^{m} \frac{1}{2} v_i \|(\dot{\mathbf{X}} - \boldsymbol{\Theta}(\mathbf{X})\boldsymbol{\Xi})_i\|^2 + \lambda R(\boldsymbol{\Xi}) \quad \text{s.t.} \quad 0 \leq v_i \leq 1, \quad \mathbf{1}^T\mathbf{v} = h, \tag{6.5}$$

where h is an estimate of the number of 'inliers' out of the potential m rows of the system. The set $\Delta_h := \{\mathbf{v} : 0 \leq v_i \leq 1, \quad \mathbf{1}^T\mathbf{v} = h\}$ is known as the capped simplex. Current algorithms for (6.5), such as those of [92], rely on LASSO formulations and thus have significant limitations (see previous section). Here, we use the SR3 strategy (6.3) to extend to the trimmed SINDy problem (6.5):

$$\min_{\boldsymbol{\Xi},\mathbf{W},\mathbf{v}\in\Delta_h} \sum_{i=1}^{m} \frac{1}{2} v_i \|(\dot{\mathbf{X}} - \boldsymbol{\Theta}(\mathbf{X})\boldsymbol{\Xi})_i\|^2 + \lambda R(\mathbf{W}) + \frac{1}{2\nu}\|\boldsymbol{\Xi} - \mathbf{W}\|^2. \tag{6.6}$$

We then use the alternating Algorithm 2 to solve the problem. The step size β is completely up to the user, as discussed in the convergence theory (see Section 6.5). The trimming algorithm requires specifying how many samples should be trimmed, which can be chosen by estimating the level of corruption in the data. Estimating derivatives using central differences, for instance, makes derivative estimates on either side of the original corrupted sample corrupt as well, meaning that three times as many samples as were originally corrupted will be bad. Thus trimming will need to be more than the initial estimate of how many samples were corrupted. Trimming ultimately can help identify and remove points with bad derivative estimates, leading to a better SINDy model fit.

Example: Lorenz. We demonstrate the use of SINDy SR3 for trimming outliers on data from the Lorenz system (5.13). We randomly select a subset of samples to corrupt, adding a high level of noise to these samples to create outliers. We apply the SINDy SR3 algorithm with trimming to simultaneously remove the corrupted samples and fit a SINDy model. Figure 6.3 shows the results of trimming on a dataset with 10% of the samples corrupted. The valid data points are shown in gray and the corrupt data points are highlighted in red. As derivatives are calculated directly from the data using central differences, this results in

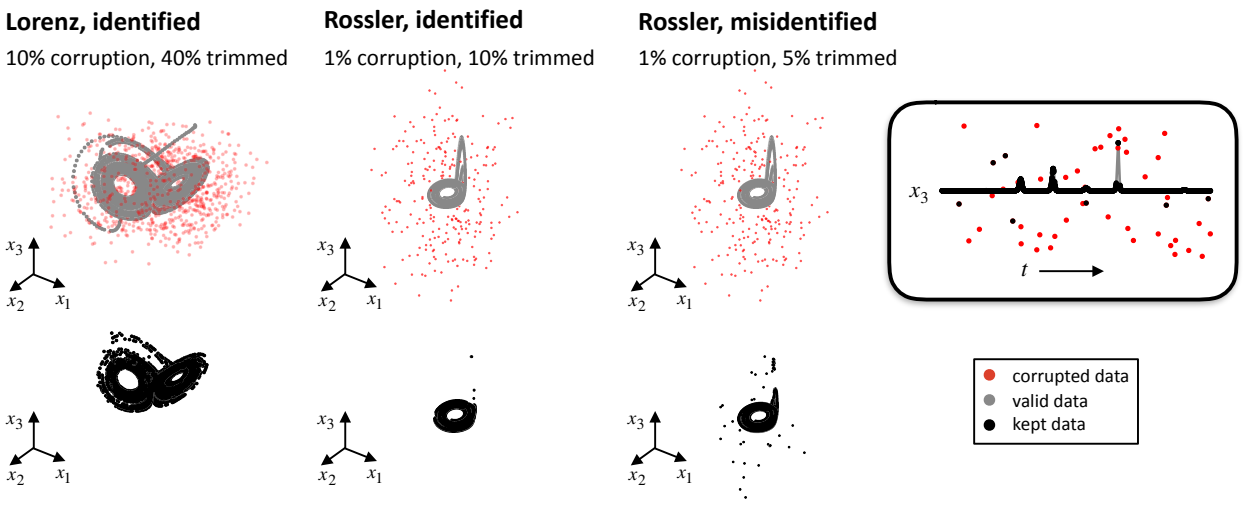

Figure 6.3: Demonstration of the trimming problem for the Lorenz and Rossler systems. For each system, we corrupt some subset of the data (corrupted values shown in red, valid data values shown in gray). We then apply SINDy SR3 with trimming. The black data points show the data that is left after trimming. For the Lorenz system, only data that is on the attractor remains and the system is correctly identified. For the Rossler system, the trimming algorithm also trims points from the portion of the attractor in the x_3 plane. The system is still correctly identified, but more data must be trimmed.

Algorithm 2 SR3-Trimming Algorithm

Input ϵ, β, $\mathbf{W}^0$, $\mathbf{v}^0$

Initialize $k = 0$, err$= 2\epsilon$.

while err$> \epsilon$ **do**

$k \leftarrow k + 1$

$$\boldsymbol{\Xi}^k = \underset{\boldsymbol{\Xi}}{\operatorname{argmin}} \sum_{i=1}^{m} \frac{1}{2} v_i \|(\dot{\mathbf{X}} - \boldsymbol{\Theta}(\mathbf{X})\boldsymbol{\Xi})_i\|^2 + \frac{1}{2\nu}\|\boldsymbol{\Xi} - \mathbf{W}^{k-1}\|^2$$

$$\mathbf{W}^k = \operatorname{prox}_{\lambda\nu R}(\boldsymbol{\Xi}^k)$$

$$\mathbf{v}^k = \operatorname{proj}_{\Delta_h}\left(\mathbf{v}^{k-1} - \beta \mathbf{g}_{\mathbf{v}}\right), \quad (\mathbf{g}_{\mathbf{v}})_i = \|(\dot{\mathbf{X}} - \boldsymbol{\Theta}(\mathbf{X})\boldsymbol{\Xi}^k)_i\|^2$$

$$\text{err} = \|\mathbf{W}^k - \mathbf{W}^{k-1}\|/\nu + \|\mathbf{v}^k - \mathbf{v}^{k-1}\|/\beta$$

end while

closer to 30% corruption (as derivative estimates on either side of each corrupt sample will also be corrupted). We find that the algorithm converges on the correct solution more often when a higher level of trimming is specified: in other words, it is better to remove some clean data along with all of the outliers than to risk leaving some outliers in the data set. Accordingly, we set our algorithm to trim 40% of the data. Despite the large fraction of corrupted samples, the method is consistently able to identify the Lorenz model (or a model with only 1-2 extra coefficients) from the remaining data in repeated simulations.

Example: Rossler The Rossler system

$$\dot{x}_1 = -x_2 - x_3, \quad \dot{x}_2 = x_1 + 0.1x_2, \quad \dot{x}_3 = 0.1 + x_3(x_1 - 14). \tag{6.7}$$

exhibits chaotic behavior characterized by regular orbits around an attractor in the x_1, x_2 plane combined with occasional excursions into the x_3 plane. The Rossler attractor is plotted in Fig. 6.3 with 1% of samples corrupted (highlighted in red). While the excursions into the x_3 dimension occur consistently as a part of the Rossler dynamics, the fact that the majority of the attractor lies in the x_1, x_2 plane means that these excursions can be seen as outliers in the dynamics. The algorithm trims these events along with the corrupted samples, and therefore a higher percentage of the data must be trimmed to ensure outliers are not missed.

Figure 6.3 shows the results of trimming when the outliers are all removed and the system is correctly identified (center panel, 10% trimmed) and when there is not enough trimming and the system is misidentified (right panel, 5% trimmed). We see that in the under-trimmed case, a significant portion of the attractor in the x_3 plane is removed whereas many of the corrupted samples are missed. In the case where the system is properly identified, the x_3 portion of the attractor is mostly removed but the system is still correctly identified.

6.3 Parameterized library functions

In standard examples of SINDy, the library is chosen to contain polynomials, which make a natural basis for many models in the physical sciences. However, many systems of interest may include more complicated terms in the dynamics, such as exponentials or trigonometric functions, that include parameters that contribute nonlinearly to the fitting problem. In addition to parameterized basis functions, systems may be subject to parameterized external forcing: for example, periodic forcing where the exact frequency of the forcing is unknown. SINDy with unknown parameters is given by

$$\min_{\boldsymbol{\Xi},\boldsymbol{\alpha}} \frac{1}{2}\|\dot{\mathbf{X}} - \boldsymbol{\Theta}(\mathbf{X},\boldsymbol{\alpha})\boldsymbol{\Xi}\|^2 + \lambda R(\boldsymbol{\Xi}). \tag{6.8}$$

This is a regularized nonlinear least squares problem. The SR3 approach makes it possible to devise an efficient algorithm for this problem as well. The relaxed formulation is given by

$$\min_{\boldsymbol{\Xi},\mathbf{W},\boldsymbol{\alpha}} \frac{1}{2}\|\dot{\mathbf{X}} - \boldsymbol{\Theta}(\mathbf{X},\boldsymbol{\alpha})\boldsymbol{\Xi}\|^2 + \lambda R(\mathbf{W}) + \frac{1}{2\nu}\|\boldsymbol{\Xi} - \mathbf{W}\|^2. \tag{6.9}$$

We solve (6.9) using Algorithm 3. The $\boldsymbol{\alpha}$ variable is updated using a true Newton step, where the gradient and Hessian are computed using algorithmic differentiation.

The joint optimization for the parameterized case yields a nonconvex problem with potentially many local minima depending on the initial choice of the parameter(s) $\boldsymbol{\alpha}$. This makes it essential to assess the fit of the discovered model through model selection criteria. While the best choice of model may be clear (see Section 6.3 and Figure 6.5 for further details), this means parameterized SINDy works best for models with only a small number of

Algorithm 3 SR3-Parameter Estimation

Input ϵ, $\mathbf{W}^0$, $\boldsymbol{\alpha}^0$

Initialize $k = 0$, err$= 2\epsilon$.

while err$> \epsilon$ **do**

$k \leftarrow k + 1$

$\boldsymbol{\Xi}^k = \underset{\boldsymbol{\Xi}}{\operatorname{argmin}} \frac{1}{2}\|\dot{\mathbf{X}} - \boldsymbol{\Theta}(\mathbf{X}, \boldsymbol{\alpha}^{k-1})\boldsymbol{\Xi}\|^2 + \frac{1}{2\nu}\|\boldsymbol{\Xi} - \mathbf{W}^{k-1}\|^2$

$\mathbf{W}^k = \operatorname{prox}_{\lambda\nu R}(\boldsymbol{\Xi}^k)$

$\mathbf{g}^k_{\boldsymbol{\alpha}} = \nabla_{\boldsymbol{\alpha}} \left(\frac{1}{2}\|\dot{\mathbf{X}} - \boldsymbol{\Theta}(\mathbf{X}, \boldsymbol{\alpha})\boldsymbol{\Xi}^k\|^2\right), \quad \mathbf{H}^k_{\boldsymbol{\alpha}} = \nabla^2_{\boldsymbol{\alpha}} \left(\frac{1}{2}\|\dot{\mathbf{X}} - \boldsymbol{\Theta}(\mathbf{X}, \boldsymbol{\alpha})\boldsymbol{\Xi}^k\|^2\right),$

$\boldsymbol{\alpha}^k = \boldsymbol{\alpha}^{k-1} - (\mathbf{H}^k_{\boldsymbol{\alpha}})^{-1}\mathbf{g}^k_{\boldsymbol{\alpha}}$

err $= \|\mathbf{W}^k - \mathbf{W}^{k-1}\|/\nu + \|\boldsymbol{\alpha}^k - \boldsymbol{\alpha}^{k-1}\|$

end while

parameters in the library, as scanning through different initializations scales combinatorially with added parameters.

Lorenz with parameterized forcing. We consider (5.13) with x_1 forced by a parameterized hyperbolic tangent function $\tanh(\alpha_1 t - \alpha_2)$. The parameters α_1, α_2 determine the steepness and location of the sigmoidal curve in the forcing function. We simulate the system with forcing parameters $\alpha_1 = 0.8, \alpha_2 = 3$. Figure 6.4 shows the results of fitting the SINDy model with and without the parameterized forcing term in the library. In the case without forcing, the equation for x_1 is loaded up with several active terms in an attempt to properly fit the dynamics. The model is not able to reproduce the correct system behavior through simulation. In the case with forcing, we start with an initial guess of $\alpha_1 = 5, \alpha_2 = 10$ and perform the joint optimization to fit both the parameters and the coefficient matrix. The algorithm correctly identifies the forcing and finds the correct coefficient matrix. The resulting system matches the true dynamics for several trips around the attractor.

Exponential integrate and fire: trimming and parameterized library. We also consider the exponential integrate and fire (EIF) neuron model. The EIF model is a spiking

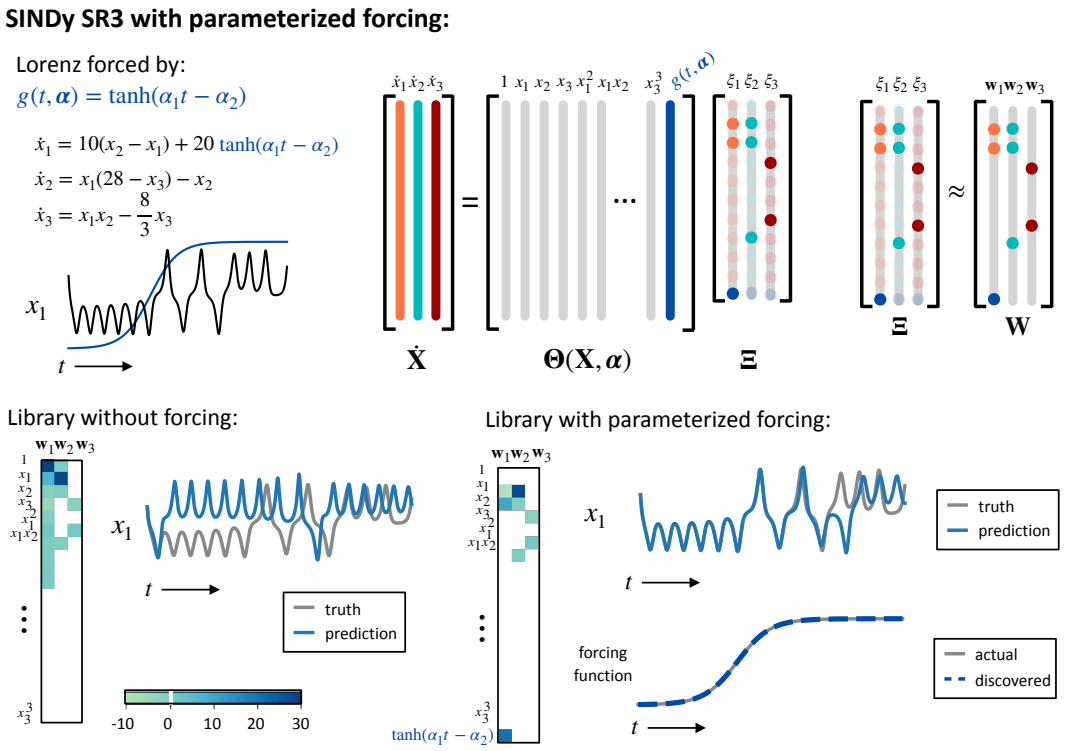

Figure 6.4: Depiction of SINDy SR3 with parameterized library terms, using the example of the Lorenz system forced by a hyperbolic tangent. The library includes a parameterized forcing term and a joint optimization is performed to find the parameter $\boldsymbol{\alpha}$ along with the SINDy model. Without the forcing term, a sparse model is not identified and the resulting model does not reproduce the behavior in simulation. With parameterized forcing in the library, both the forcing parameters and the library can be correctly identified given a sufficiently close initialization of the parameters $\boldsymbol{\alpha}$.

neural model where the membrane potential x of a neuron is governed by

$$\dot{x} = -(x - x_{\text{rest}}) + \Delta_T \exp((x - x_c)/\Delta_T) + I \tag{6.10}$$

and a set of rules that determine when the neuron spikes. In modeling the system, when the value of the potential reaches a threshold potential $x > x_{\text{threshold}}$, the neuron is considered to have fired a spike and the potential is reset to $x = x_{\text{reset}}$. While the EIF model is a simplified model that does not capture the rich dynamics of real neurons, it serves as an ideal example for illustrating issues that may arise in scientific data. This model has sharp discontinuities at

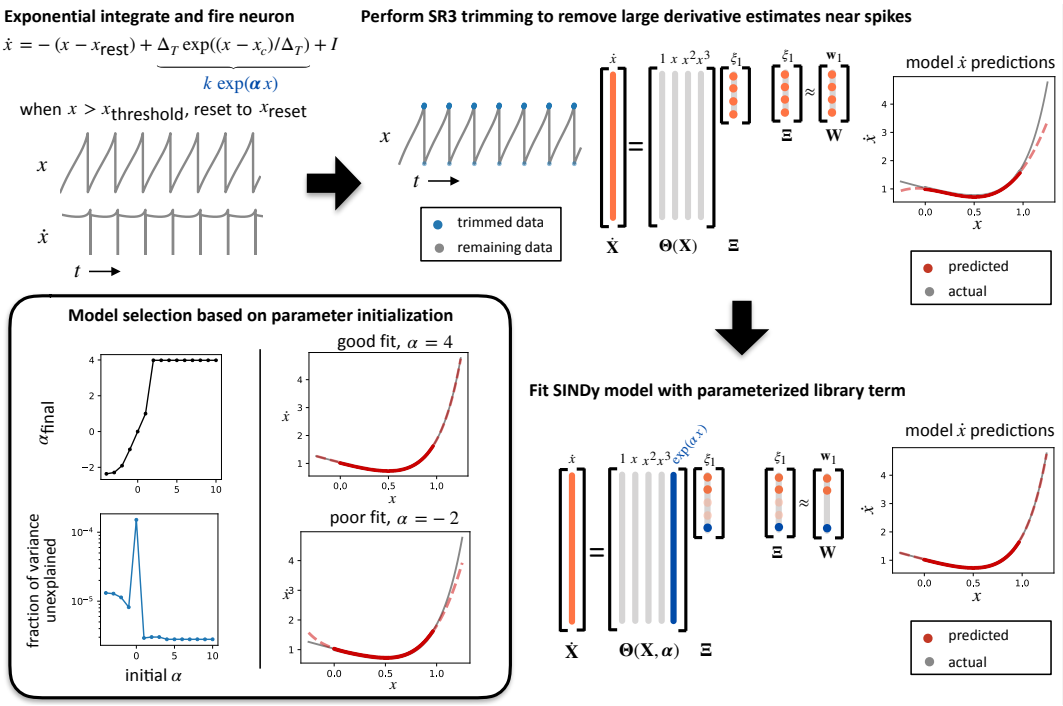

Figure 6.5: Workflow for identifying the exponential integrate and fire neuron, a spiking neuron model. First SINDy SR3 with trimming is performed, which removes the data points near the spikes but does not capture the exponential term. However, the SINDy algorithm with a parameterized exponential term is able to identify the correct model given proper initialization of the parameter. The inset shows how initialization affects the discovered parameter value: some initializations do not recover the correct parameter, and in this case the SINDy model error is higher (error shown on log scale). Model selection should therefore be used to identify the correct parameter value.

the spikes. While these discontinuities are artificial, true neural data also typically contains sharp peaks at the spikes, leading to inaccurate derivative estimates in these regions. It can therefore be useful to trim this data when derivative estimates are likely to be inaccurate. Additionally, the model has parameterized terms in the dynamics: there is an exponential term in the governing equation determined by a parameter that cannot be fit using a simple linear least squares regression.

We simulated the EIF model with a constant input current at a level that results in 7 spikes over the course of the simulation. The discontinuities at the spikes lead to bad deriva-

tive estimates around these points. We therefore run the trimming algorithm introduced in Sec. 6.2, which removes these points and fits a SINDy model. In this case, a sparse SINDy model is not identified as the regression uses all polynomial terms to try to approximate the exponential term present in the dynamics. Therefore, following the trimming we run the algorithm introduced in Sec. 6.3 to fit a SINDy model with a parameterized exponential term. The model predictions of the derivatives $\dot{x}$ are shown in Fig. 6.5. The parameterized model results in an optimization that may have multiple local minima, thus the initial guess for the parameters influences the optimization results. Figure 6.5 shows an analysis of how initialization of the parameter α affects the discovered model. For some initializations the optimization does not find the correct value for α. However, in these cases the model error is higher and looking at the resulting model predictions shows that the discovered model is incorrect.

6.4 Simulation details

6.4.1 Performance of SR3 for SINDy

We illustrate a comparison of three algorithms for SINDy using data from the canonical example of the chaotic Lorenz system (5.13). To generate training data, we simulate the system from $t = 0$ to 10 with a time step of $\Delta t = 0.005$ for 20 initial conditions sampled from a random uniform distribution in a box around the attractor. This results in a data set with 40×10^4 samples. We add random Gaussian noise with a standard deviation of 10^{-2} and compute the derivatives of the data using the central difference method. The SINDy library matrix $\boldsymbol{\Theta}(\mathbf{X})$ is constructed using polynomial terms through order 3.

We find the SINDy model coefficient matrix using the following optimization approaches: sequentially thresholded least squares (STLSQ) with threshold 0.1, SR3 with ℓ_0 regularization, LASSO with a regularization weight of 0.1, and LASSO with a regularization weight of 50. The STLSQ algorithm is performed by doing 10 iterations of the following procedure: (1) perform a least squares fitting on remaining coefficients, (2) remove all coefficients with

magnitude less than 0.1. The LASSO models are fit using the scikit-learn package [64]. LASSO models are fit without an intercept, and for both LASSO and SR3 we initialize the coefficient matrix using least squares. For the SR3 algorithm, we use parameters $\nu = 1$ and $\lambda = 0.005$ (which corresponds to a coefficient threshold of 0.1, see Section 6.1.2).

For each of the four resulting models we analyze (1) the sparsity pattern of the coefficient matrix and (2) the simulation of the resulting dynamical systems model. We compare the sparsity pattern of the coefficient matrix against the true sparsity pattern for the Lorenz system: SR3 and STLSQ identify the correct sparsity pattern, where as the LASSO models do not. For all models, we simulate the identified system on test trajectories using initial conditions not found in the training set. The initial conditions are $(-8, 7, 27)$ (on attractor) and $(0.01, 0.01, 80)$ (off attractor), and the systems are simulated for the same time duration used in the training set. These results are shown in Figure 6.2 in the main text.

6.4.2 Simultaneous sparse inference and data trimming

Example: Lorenz

We demonstrate the use of the SR3-trimming algorithm 2 on data from the Lorenz system (5.13). We simulate the system over the same time as in Section 6.4.1 from 5 randomly sampled initial conditions. This results in a data set with 10^4 samples. We add Gaussian noise to the data with standard deviation 10^{-3}. We then randomly choose 10% of the samples to corrupt (1000 total samples). For each state variable of each corrupted sample, noise chosen from a random uniform distribution over $[-50, 50]$ is added. Derivatives are calculated from the data using central difference after the corruption is applied. We then apply the SR3-trimming algorithm, specifying that around 40% of the data points will be trimmed. We use SR3 parameters $\nu = 20$ and $\lambda = 0.00025$ (corresponding to a coefficient threshold of 0.1), and the step size is taken to be the default value $\beta = 1$. With repeated testing we find that the algorithm is consistently able to correctly remove the outliers from the data set and identify the Lorenz system.

Example: Rossler

As an additional example, we test the trimming algorithm on data from the Rossler system (6.7). We generate sample data from 5 randomly sampled initial conditions around the portion of the attractor in the x_1, x_2 plane, simulating trajectories from $t = 0$ to 50 with a time step of $\Delta t = 0.01$. Our data set consists of 25000 samples. We add Gaussian noise with a standard deviation of 10^{-3} and add outliers to 1% of the data in the same manner as in Section 6.4.2, with the noise level chosen from a random uniform distribution over $[-100, 100]$. Derivatives are calculated from the corrupted data using central difference.

We apply the SR3-trimming algorithm with two different levels of trimming. In both cases we use SR3 parameters $\nu = 20$ and $\lambda = 6.25 \times 10^{-5}$ (corresponding to a coefficient threshold of 0.05) and default step size $\beta = 1$. On repeated trials we find that if we trim only 5% of the data, many of the outliers are typically missed and the system is not correctly identified (instead, the algorithm trims part of the attractor in the x_3 plane). However, if we trim 10% of the data the system is correctly identified in most cases (or only 1 or 2 coefficients are misidentified).

6.4.3 Parameterized library functions

Lorenz with parameterized forcing

To demonstrate the use of SR3 for SINDy with parameter estimation, we look at an example of the Lorenz system (5.13) forced by a parameterized hyperbolic tangent function $\tanh(\alpha_1 t - \alpha_2)$. The full set of equations for the system is

$$\begin{aligned}
\dot{x}_1 &= 10(x_2 - x_1) + 20\tanh(\alpha_1 t - \alpha_2) \\
\dot{x}_2 &= x_1(28 - x_3) - x_2 \\
\dot{x}_3 &= x_1 x_2 - (8/3)x_3.
\end{aligned}$$

The parameters α_1, α_2 determine the steepness and location of the sigmoidal curve in the forcing function. We simulate the system as in Section 6.4.1 for a single initial condition

$(8, -7, 27)$ with forcing parameters $\alpha_1 = 0.8, \alpha_2 = 3$. We add Gaussian noise of standard deviation 10^{-3} and compute the derivatives via central difference.

We apply Algorithm 2 to perform a joint discovery of both the coefficients $\mathbf{W}$ and forcing parameters $\boldsymbol{\alpha}$. We use parameters $\nu = 0.1$ and $\lambda = 0.05$ (corresponding to coefficient threshold 0.1). $\mathbf{W}$ is initialized using least squares, and as an initial guess for $\boldsymbol{\alpha}$ we use $\boldsymbol{\alpha}^0 = (5, 10)$. The algorithm discovers the correct parameters $\boldsymbol{\alpha}$ as well as the correct sparsity pattern in the coefficient matrix. We simulate the system and see that the discovered system tracks the behavior for several trips around the attractor. Results are shown in Figure 6.4.

For comparison, we apply the SR3 algorithm for SINDy with no forcing term in the library, using the same SR3 parameters as in the forcing case. The resulting model has many active terms in the equation for $\dot{x}_1$, as it attempts to capture the forcing behavior with polynomials of x_1, x_2, x_3. This model does not perform well in simulation, even from the same initial condition used in the training set. Figure 6.4 shows the coefficient matrix and model simulation for the discovered system.

Exponential integrate and fire neuron: trimming and parameterized library

To demonstrate a work flow with both trimming and parameterized library functions, we perform systems identification on simulation of an exponential integrate and fire (EIF) neuron model

$$\dot{x} = -(x - x_{\text{rest}}) + \Delta_T \exp((x - x_c)/\Delta_T) + I$$

with parameters $x_{\text{rest}} = 0, x_c = 0.5, \Delta_T = 0.25$. The input current I is set to a constant value of 1. In the EIF model, the differential equation above is combined with a mechanism for spiking: when the potential x reaches a threshold $x_{\text{threshold}}$, its value at that time point is reset to a reset potential x_{reset} and the neuron is said to have fired a spike at that time point. We use $x_{\text{threshold}} = 1$ and $x_{\text{reset}} = 0$. We simulate the EIF model from $t = 0$ to 8 with a time step of $\Delta t = 10^{-3}$, using a forward Euler time stepping method and the spiking mechanism described above. At the given parameter values, there are a total of 7 spikes

over the course of the simulation. This example is particularly sensitive to noise, and thus we demonstrate the results without added noise. Derivatives are computed using the central difference method.

Due to the discontinuities at the spikes, the derivative estimates for this data have sharp peaks near the spikes. We first apply Algorithm 2, which removes data points near the spikes. We apply the algorithm with parameters $\nu = 10$, $\lambda = 5 \times 10^{-6}$ (corresponding to a coefficient threshold of 0.01), telling the algorithm to trim 2% of the data. The result is that several data points near the spikes are trimmed. The resulting SINDy model is not sparse, as the coefficient library does not have an exponential term and the algorithm instead tries to approximate the exponential using polynomial terms.

To capture the true model for the neuron, we next apply Algorithm 3 to the trimmed data. Rather than including a forcing term in the library as in Section 6.4.3, we include a parameterized function of x in the form of an exponential: $g(\alpha, x) = \exp(\alpha x)$. The parameterized library is $\boldsymbol{\Theta}(\mathbf{x}, \alpha) = [1, \mathbf{x}, \mathbf{x}^2, \mathbf{x}^3, \exp(\alpha \mathbf{x})]$. We apply Algorithm 3 with the same parameters $\nu = 10$, $\lambda = 5 \times 10^{-6}$. Because the parameterized model results in an optimization with potentially many local minima, the initial guess α^0 significantly impacts the discovered parameter value α and coefficients $\mathbf{W}$. In Figure 6.5, we show the discovered parameter α and the prediction error of $\dot{x}$ for several initial values α^0. The correct value in this example is $\hat{\alpha} = 4$, and we use initial values ranging from $\alpha^0 = -4$ to 10. The prediction error shown is the fraction of variance of $\dot{x}$ unexplained by the resulting model (defined by the discovered parameters $\alpha, \mathbf{W}$), with error plotted on a log scale. Initializations close enough to the true value $\hat{\alpha}$ discover the right value and have a low error compared to models where the incorrect value is discovered. At these values, the correct sparsity pattern in $\mathbf{W}$ is also discovered. This motivates the use of model selection to select among models with different initializations.

It should be possible to combine both the trimming and parameter search into a single optimization problem within the SR3 framework, but we leave this to future work.

6.5 Convergence results

Here we state convergence results for Algorithm 1 and Algorithm 2. These algorithms fall under the framework of two classical methods, proximal gradient descent and the proximal alternating linearized minimization algorithm (PALM) [6]. While we demonstrate the use of Algorithm 3 on two example problems, this algorithm is much harder algorithm to analyze due to the complication from the Newton's step. We leave obtaining theoretical guarantees of Algorithm 3 as future work.

6.5.1 Convergence of Algorithm 1

Using the variable projection framework [24], we partially optimize out $\mathbf{\Xi}$ and then treat Algorithm 1 as the classical proximal gradient method on $\mathbf{W}$. The convergence result for Algorithm 1 is provided in [96, Theorem 2] and is restated here:

Theorem 1 *Define the value function as,*

$$p(\mathbf{W}) = \min_{\mathbf{\Xi}} \frac{1}{2}\|\dot{\mathbf{X}} - \mathbf{\Theta}(\mathbf{X})\mathbf{\Xi}\|^2 + \lambda R(\mathbf{W}) + \frac{1}{2\nu}\|\mathbf{\Xi} - \mathbf{W}\|^2$$

When p is bounded below, we know that the iterators from Algorithm 1 satisfy,

$$\frac{1}{N}\sum_{k=1}^{N} \|g^k\|^2 \leq \frac{1}{\nu N}(p(\mathbf{W}^0) - p^*),$$

where $g^k \in \partial p(\mathbf{W}^k)$ and $p^* = \min_{\mathbf{W}} p(\mathbf{W})$.

We obtain a sub-linear convergence rate for all prox-bounded regularizers R.

6.5.2 Convergence of Algorithm 2

Following the same idea provided by the variable projection framework, the iterations from Algorithm 2 are equivalent with an alternating proximal gradient step between $\mathbf{W}$ and $\mathbf{v}$. This is the PALM algorithm, which is thoroughly analyzed in the context of trimming in [1] and [19]. We restate the convergence result here:

Theorem 2 *Consider the value function,*

$$p(\mathbf{W}, \mathbf{v}) = \min_{\boldsymbol{\Xi}} \sum_{i=1}^{m} \frac{1}{2} v_i \|(\dot{\mathbf{X}} - \boldsymbol{\Theta}(\mathbf{X})\boldsymbol{\Xi})_i\|^2 + \lambda R(\mathbf{W}) + \frac{1}{2\nu}\|\boldsymbol{\Xi} - \mathbf{W}\|^2$$

And we know that the iterators (W^k, v^k) *converge to the stationary point of* p*, with the rate,*

$$\min_{k=0,\ldots,N} \operatorname{dist}(0, \partial p(\mathbf{W}^k, \mathbf{v}^k)) = o\left(\frac{1}{k+1}\right).$$

Algorithm 2 also requires the specification of a step size β for the proximal gradient step for $\mathbf{v}$. Because the objective is linear with respect to $\mathbf{v}$, the step size will not influence the convergence result in the above theorem. However, because the objective is non-convex, β will have an impact on where the solution lands. In this work we use a default step size of $\beta = 1$ for all examples.

6.6 Discussion

Machine learning for model discovery in physics, biology and engineering is of growing importance for characterizing complex systems for the purpose of control and technological applications. Critical for the design and implementation in new and emerging technologies is the ability to interpret and generalize the discovered models, thus requiring that parsimonious models be discovered which are minimally parametrized. Moreover, model discovery architectures must be able to incorporate the effects of constraints, provide robust models, and/or give accurate nonlinear parameter estimates. We here propose the SINDy-SR3 method which integrates a sparse regression framework for parsimonious model discovery with a unified optimization algorithm capable of incorporating many of the critical features necessary for real-life applications. We demonstrate its accuracy and efficiency on a number of example problems, showing that SINDy-SR3 is a viable framework for the engineering sciences.

BIBLIOGRAPHY

[1] Aleksandr Aravkin and Damek Davis. A smart stochastic algorithm for nonconvex optimization with applications to robust machine learning. *arXiv preprint arXiv:1610.01101*, 2016.

[2] Hassan Arbabi and Igor Mezić. Ergodic theory, dynamic mode decomposition and computation of spectral properties of the Koopman operator. *SIAM J. Appl. Dyn. Syst.*, 16(4):2096–2126, 2017.

[3] Travis Askham and J Nathan Kutz. Variable projection methods for an optimized dynamic mode decomposition. *SIAM Journal on Applied Dynamical Systems*, 17(1):380–416, 2018.

[4] Pierre Baldi and Kurt Hornik. Neural networks and principal component analysis: Learning from examples without local minima. *Neural Netw.*, 2(1):53–58, 1989.

[5] Yohai Bar-Sinai, Stephan Hoyer, Jason Hickey, and Michael P Brenner. Data-driven discretization: a method for systematic coarse graining of partial differential equations. *arXiv preprint arXiv:1808.04930*, 2018.

[6] Jérôme Bolte, Shoham Sabach, and Marc Teboulle. Proximal alternating linearized minimization for nonconvex and nonsmooth problems. *Mathematical Programming*, 146(1-2):459–494, 2014.

[7] Josh Bongard and Hod Lipson. Automated reverse engineering of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 104(24):9943–9948, 2007.

[8] David S. Broomhead and Roger Jones. Time-series analysis. *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 423(1864):103–121, 1989.

[9] B. W. Brunton, L. A. Johnson, J. G. Ojemann, and J. N. Kutz. Extracting spatial–temporal coherent patterns in large-scale neural recordings using dynamic mode decomposition. *Journal of Neuroscience Methods*, 258:1–15, 2016.

[10] Steven L. Brunton, Bingni W. Brunton, Joshua L. Proctor, Eurika Kaiser, and J. Nathan Kutz. Chaos as an intermittently forced linear system. *Nature Communications*, 8(1), December 2017.

[11] Steven L. Brunton, Bingni W. Brunton, Joshua L. Proctor, and J. Nathan Kutz. Koopman invariant subspaces and finite linear representations of nonlinear dynamical systems for control. *PLOS ONE*, 11(2):1–19, 2016.

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

[13] Marko Budisic, Ryan Mohr, and Igor Mezic. Applied Koopmanism. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 22(4):047510, 2012.

[14] Kevin T Carlberg, Antony Jameson, Mykel J Kochenderfer, Jeremy Morton, Liqian Peng, and Freddie D Witherden. Recovering missing cfd data for high-order discretizations using deep neural networks and dynamics learning. *arXiv preprint arXiv:1812.01177*, 2018.

[15] Rick Chartrand. Numerical differentiation of noisy, nonsmooth data. *ISRN Applied Mathematics*, 2011, 2011.

[16] Theodore Cornforth and Hod Lipson. Symbolic regression of multiple-time-scale dynamical systems. In *Proceedings of the 14th annual conference on Genetic and evolutionary computation*, pages 735–742. ACM, 2012.

[17] Magnus Dam, Morten Brøns, Jens Juul Rasmussen, Volker Naulin, and Jan S Hesthaven. Sparse identification of a predator-prey system from simulation data of a convection model. *Physics of Plasmas*, 24(2):022310, 2017.

[18] Suddhasattwa Das and Dimitrios Giannakis. Delay-coordinate maps and the spectra of Koopman operators. *arXiv preprint arXiv:1706.08544*, 2017.

[19] Damek Davis. The asynchronous palm algorithm for nonsmooth nonconvex problems. *arXiv preprint arXiv:1604.00526*, 2016.

[20] N. B. Erichson, S. L. Brunton, and J. N. Kutz. Compressed dynamic mode decomposition for real-time object detection. *J. Real Time Processing*, 2017.

[21] Gary Froyland, Georg A Gottwald, and Andy Hammerlindl. A computational method to extract macroscopic variables and their dynamics in multiscale systems. *SIAM Journal on Applied Dynamical Systems*, 13(4):1816–1846, 2014.

[22] Gary Froyland, Georg A Gottwald, and Andy Hammerlindl. A trajectory-free framework for analysing multiscale systems. *Physica D: Nonlinear Phenomena*, 328:34–43, 2016.

[23] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[24] Gene H Golub and Victor Pereyra. The differentiation of pseudo-inverses and nonlinear least squares problems whose variables separate. *SIAM Journal on numerical analysis*, 10(2):413–432, 1973.

[25] Francisco J Gonzalez and Maciej Balajewicz. Learning low-dimensional feature dynamics using deep convolutional recurrent autoencoders. *arXiv preprint arXiv:1808.01346*, 2018.

[26] R Gonzalez-Garcia, R Rico-Martinez, and IG Kevrekidis. Identification of distributed parameter systems: A neural net based approach. *Computers & chemical engineering*, 22:S965–S968, 1998.

[27] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[28] J. Grosek and J. N. Kutz. *Dynamic Mode Decomposition for Real-Time Background/Foreground Separation in Video. arXiv preprint, arXiv:1404.7592*, 2014.

[29] Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Brian Kingsbury, et al. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal processing magazine*, 29, 2012.

[30] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[31] Sepp Hochreiter and Jürgen Schmidhuber. LSTM can solve hard long time lag problems. In *Advances in neural information processing systems*, pages 473–479, 1997.

[32] Moritz Hoffmann, Christoph Fröhner, and Frank Noé. Reactive SINDy: Discovering governing reactions from concentration data. *Journal of Chemical Physics*, 150(025101), 2019.

[33] P. J. Holmes, J. L. Lumley, G. Berkooz, and C. W. Rowley. *Turbulence, coherent structures, dynamical systems and symmetry.* Cambridge Monographs in Mechanics. Cambridge University Press, Cambridge, England, 2nd edition, 2012.

[34] Philip Holmes and John Guckenheimer. *Nonlinear oscillations, dynamical systems, and bifurcations of vector fields*, volume 42 of *Applied Mathematical Sciences*. Springer-Verlag, Berlin, Heidelberg, 1983.

[35] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Netw.*, 2(5):359–366, 1989.

[36] J. N. Juang and R. S. Pappa. An eigensystem realization algorithm for modal parameter identification and model reduction. *Journal of Guidance, Control, and Dynamics*, 8(5):620–627, September 1985.

[37] Eurika Kaiser, J Nathan Kutz, and Steven L Brunton. Sparse identification of nonlinear dynamics for model predictive control in the low-data limit. *arXiv preprint arXiv:1711.05501*, 2017.

[38] Eurika Kaiser, J Nathan Kutz, and Steven L Brunton. Sparse identification of nonlinear dynamics for model predictive control in the low-data limit. *Proc. R. Soc. A*, 474(2219), 2018.

[39] Ioannis G Kevrekidis, C William Gear, James M Hyman, Panagiotis G Kevrekidid, Olof Runborg, Constantinos Theodoropoulos, and others. Equation-free, coarse-grained multiscale computation: Enabling mocroscopic simulators to perform system-level analysis. *Communications in Mathematical Sciences*, 1(4):715–762, 2003.

[40] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[41] B. O. Koopman. Hamiltonian systems and transformation in Hilbert space. *Proceedings of the National Academy of Sciences*, 17(5):315–318, 1931.

[42] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[43] J. N. Kutz, S. L. Brunton, B. W. Brunton, and J. L. Proctor. *Dynamic Mode Decomposition: Data-Driven Modeling of Complex Systems*. SIAM, 2016.

[44] J Nathan Kutz, Steven L Brunton, Bingni W Brunton, and Joshua L Proctor. *Dynamic mode decomposition: data-driven modeling of complex systems*, volume 149. SIAM, 2016.

[45] J Nathan Kutz, Joshua L Proctor, and Steven L Brunton. Koopman theory for partial differential equations. *arXiv preprint arXiv:1607.07076*, 2016.

[46] Zhilu Lai and Satish Nagarajaiah. Sparse structural system identification method for nonlinear dynamic systems with hysteresis/inelastic behavior. *Mech. Sys. & Sig. Proc.*, 117:813–842, 2019.

[47] Soledad Le Clainche and José M. Vega. Higher order dynamic mode decomposition. *SIAM Journal on Applied Dynamical Systems*, 16(2):882–925, January 2017.

[48] Kookjin Lee and Kevin Carlberg. Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders. *arXiv preprint arXiv:1812.08373*, 2018.

[49] Qianxiao Li, Felix Dietrich, Erik M. Bollt, and Ioannis G. Kevrekidis. Extended dynamic mode decomposition with dictionary learning: A data-driven adaptive spectral decomposition of the Koopman operator. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 27(10):103111, 2017.

[50] Zachary C Lipton, John Berkowitz, and Charles Elkan. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*, 2015.

[51] J.-C. Loiseau and S. L. Brunton. Constrained sparse Galerkin regression. *Journal of Fluid Mechanics*, 838:42–67, 2018.

[52] Bethany Lusch, J Nathan Kutz, and Steven L Brunton. Deep learning for universal linear embeddings of nonlinear dynamics. *Nature communications*, 9(1):4950, 2018.

[53] N. M. Mangan, J. N. Kutz, S. L. Brunton, and J. L. Proctor. Model selection for dynamical systems via sparse regression and information criteria. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Science*, 473(2204):20170009, August 2017.

[54] Niall M Mangan, Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. Inferring biological networks by sparse identification of nonlinear dynamics. *IEEE Transactions on Molecular, Biological, and Multi-Scale Communications*, 2(1):52–63, 2016.

[55] Andreas Mardt, Luca Pasquali, Hao Wu, and Frank Noé. VAMPnets: Deep learning of molecular kinetics. *Nature Communications*, 9(5), 2018.

[56] Igor Mezic. Spectral properties of dynamical systems, model reduction and decompositions. *Nonlinear Dynamics*, 41(1):309–325, August 2005.

[57] Igor Mezic. Analysis of fluid flows via spectral properties of the Koopman operator. *Annual Review of Fluid Mechanics*, 45(1):357–378, 2013.

[58] Michele Milano and Petros Koumoutsakos. Neural network modeling for near wall turbulent flow. *Journal of Computational Physics*, 182(1):1–26, 2002.

[59] Frank Noé and Feliks Nuske. A variational approach to modeling slow processes in stochastic dynamical systems. *Multiscale Modeling & Simulation*, 11(2):635–655, 2013.

[60] Feliks Nüske, Bettina G Keller, Guillermo Pérez-Hernández, Antonia SJS Mey, and Frank Noé. Variational approach to molecular kinetics. *Journal of chemical theory and computation*, 10(4):1739–1752, 2014.

[61] Samuel E Otto and Clarence W Rowley. Linearly-recurrent autoencoder networks for learning dynamics. *arXiv preprint arXiv:1712.01378*, 2017.

[62] Jaideep Pathak, Brian Hunt, Michelle Girvan, Zhixin Lu, and Edward Ott. Model-free prediction of large spatiotemporally chaotic systems from data: a reservoir computing approach. *Physical review letters*, 120(2):024102, 2018.

[63] K. Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2(7–12):559–572, 1901.

[64] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python . *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[65] Maziar Raissi and George Em Karniadakis. Hidden physics models: Machine learning of nonlinear partial differential equations. *arXiv preprint arXiv:1708.00588*, 2017.

[66] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations. *arXiv preprint arXiv:1711.10561*, 2017.

[67] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics informed deep learning (part ii): Data-driven discovery of nonlinear partial differential equations. *arXiv preprint arXiv:1711.10566*, 2017.

[68] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Multistep neural networks for data-driven discovery of nonlinear dynamical systems. *arXiv preprint arXiv:1801.01236*, 2018.

[69] Peter J Rousseeuw. Least median of squares regression. *Journal of the American statistical association*, 79(388):871–880, 1984.

[70] C. W. Rowley, I. Mezić, S. Bagheri, P. Schlatter, and D. S. Henningson. Spectral analysis of nonlinear flows. *Journal of Fluid Mechanics*, 645:115–127, 2009.

[71] S. H. Rudy, S. L. Brunton, J. L. Proctor, and J. N. Kutz. Data-driven discovery of partial differential equations. *Science Advances*, 3(e1602614), 2017.

[72] Samuel H Rudy, J Nathan Kutz, and Steven L Brunton. Deep learning of dynamics and signal-noise decomposition with time-stepping constraints. *arXiv preprint arXiv:1808.02578*, 2018.

[73] Hayden Schaeffer. Learning partial differential equations via data discovery and sparse optimization. In *Proc. R. Soc. A*, volume 473, page 20160446. The Royal Society, 2017.

[74] Hayden Schaeffer and Scott G McCalla. Sparse model selection via integral terms. *Physical Review E*, 96(2):023302, 2017.

[75] Peter J. Schmid. Dynamic mode decomposition of numerical and experimental data. *Journal of Fluid Mechanics*, 656:5–28, 2010.

[76] PJ Schmid, L Li, MP Juniper, and O Pust. Applications of the dynamic mode decomposition. *Theoretical and Computational Fluid Dynamics*, 25(1-4):249–259, 2011.

[77] Michael Schmidt and Hod Lipson. Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85, 2009.

[78] Mariia Sorokina, Stylianos Sygletos, and Sergei Turitsyn. Sparse identification for nonlinear optical communication systems: SINO method. *Optics express*, 24(26):30433–30443, 2016.

[79] Naoya Takeishi, Yoshinobu Kawahara, and Takehisa Yairi. Learning Koopman invariant subspaces for dynamic mode decomposition. In *Advances in Neural Information Processing Systems*, pages 1130–1140, 2017.

[80] Floris Takens. Detecting strange attractors in turbulence. In *Dynamical systems and turbulence, Warwick 1980*, pages 366–381. Springer, 1981.

[81] Robert Tibshirani, Martin Wainwright, and Trevor Hastie. *Statistical learning with sparsity: the lasso and generalizations*. Chapman and Hall/CRC, 2015.

[82] Giang Tran and Rachel Ward. Exact recovery of chaotic systems from highly corrupted data. *Multiscale Modeling & Simulation*, 15(3):1108–1129, 2017.

[83] J. H. Tu, C. W. Rowley, D. M. Luchtenburg, S. L. Brunton, and J. N. Kutz. On dynamic mode decomposition: theory and applications. *Journal of Computational Dynamics*, 1(2):391–421, 2014.

[84] Pantelis R. Vlachas, Wonmin Byeon, Zhong Y. Wan, Themistoklis P. Sapsis, and Petros Koumoutsakos. Data-driven forecasting of high-dimensional chaotic systems with long-short term memory networks. *arXiv preprint arXiv:1802.07486*, 2018.

[85] Christoph Wehmeyer and Frank Noé. Time-lagged autoencoders: Deep learning of slow collective variables for molecular kinetics. *The Journal of chemical physics*, 148(24):241703, 2018.

[86] E Weinan. *Principles of multiscale modeling*. Cambridge University Press, 2011.

[87] E Weinan, Bjorn Engquist, and others. The heterogeneous multiscale methods. *Communications in Mathematical Sciences*, 1(1):87–132, 2003.

[88] Matthew O. Williams, Ioannis G. Kevrekidis, and Clarence W. Rowley. A data–driven approximation of the Koopman operator: Extending dynamic mode decomposition. *Journal of Nonlinear Science*, 25(6):1307–1346, December 2015.

[89] Matthew O Williams, Clarence W Rowley, and Ioannis G Kevrekidis. A kernel-based method for data-driven Koopman spectral analysis. *Journal of Computational Dynamics*, 2(2):247–265, 2015.

[90] Or Yair, Ronen Talmon, Ronald R Coifman, and Ioannis G Kevrekidis. Reconstruction of normal forms by learning informed observation geometries from data. *Proceedings of the National Academy of Sciences*, page 201620045, 2017.

[91] Eunho Yang and Aurélie C Lozano. Robust Gaussian graphical modeling with the trimmed graphical lasso. In *Advances in Neural Information Processing Systems*, pages 2602–2610, 2015.

[92] Eunho Yang, Aurélie C Lozano, Aleksandr Aravkin, et al. A general family of trimmed estimators for robust high-dimensional data analysis. *Electronic Journal of Statistics*, 12(2):3519–3553, 2018.

[93] Chen Yao and Erik M Bollt. Modeling and nonlinear parameter estimation with Kronecker product representation for coupled oscillators and spatiotemporal systems. *Physica D*, 227(1):78–99, 2007.

[94] Enoch Yeung, Soumya Kundu, and Nathan Hodas. Learning deep neural network representations for Koopman operators of nonlinear dynamical systems. *arXiv preprint arXiv:1708.06850*, 2017.

[95] Linan Zhang and Hayden Schaeffer. On the convergence of the SINDy algorithm. *arXiv preprint arXiv:1805.06445*, 2018.

[96] Peng Zheng, Travis Askham, Steven L Brunton, J Nathan Kutz, and Aleksandr Y Aravkin. A unified framework for sparse relaxed regularized regression: Sr3. *IEEE Access*, 7:1404–1423, 2019.