Evaluating Linearized Echo State Networks, SINDy Framework, and Direct Differentiation Methods for Dynamical System Modeling

Atticus Rex
Instructor: Serkan Gugercin, PhD

Abstract

This paper develops a framework for improving upon traditional regression methods to model discrete nonlinear dynamical systems. This framework is derived from a branch of recurrent neural-networks called Echo State Networks (ESNs). This report compares this methodology to the famous Sparse Identification of Nonlinear Dynamics (SINDy) framework developed by Brunton et al. This investigation specifically analyzes the Lorenz System as an example nonlinear and chaotic system. In the models investigated within this report, the Linear ESN model outperforms the SINDy algorithm when noise in the system is sufficiently low. The SINDy algorithm is also far more computationally intensive, requiring much more data to yield comparable results. In addition, SINDy requires sampling of a numerical derivative of a signal. Using finite difference methods amplifies any noise present in the signal, and thus, a direct, discrete method for calculating a regularized derivative is proposed, based on the Total Variation Regularized Derivative (TVD).

A report presented as the term project for
MATH 5564 - System-Theory Model Reduction

Dept. of Mathematics
Blacksburg, Virginia
May 10, 2023

Atticus Rex
MATH 5564
Term Project
Dr. Serkan Gugercin
May 10, 2023

1 Introduction

Dynamical systems are a fundamental area of mathematical study, with deep applications in disciplines such as physics, engineering, biology, and numerical economics [1]. These systems are characterized by a set of underlying laws that determine behavior over time. The modeling of dynamical systems has become increasingly important in recent years due to the growing complexity of many real-world phenomena, as well as the exponential rise in computational speed and parallelism to process them.

One of the primary challenges in modeling dynamical systems is developing methods to accurately capture the evolution of these systems over time. This involves constructing mathematical frameworks that capture the essential features of a system, developing numerical methods to solve these models efficiently, and producing models that are resistant to disturbances, such as noise. In this paper, we will provide a framework for a new method of simulating discrete time dynamical systems and compare it to existing methods such as linear regression, nonlinear regression and the SINDy framework developed by Brunton, Proctor and Kutz et al. [2]. This new method was developed from a type of time-series machine learning algorithm called Echo State Neural Networks (ESNs) [3].

This paper seeks to determine whether established algorithms like SINDy that rely on computing a continuous-time model based on a numerical derivative of a system is more effective than the discrete temporal information preserved within ESNs.

1.1 Echo State Networks

Echo State Networks (ESNs) are a type of recurrent neural network that have gained increasing attention in recent years due to their ability to efficiently model and predict complex time-series data [3]. ESNs are characterized by a fixed, randomly-initialized recurrent layer that is trained using only a linear readout layer, making them computationally efficient and easy to train [4]. This simplicity and efficiency make ESNs particularly useful for modeling time-series data with high-dimensional inputs, such as speech or image data.

The key innovation of ESNs is the use of a randomly-initialized recurrent layer, known as the "echo state". The echo state has fixed connectivity and weights, and is not trained during the learning process. Instead, a readout layer is trained to produce the desired output based on the current state of the echo state and the input at each time step. This approach allows ESNs to model complex time-series data with minimal training, as the echo state acts as a reservoir of information that can capture the temporal dynamics of the input signal.

ESNs have been successfully applied to a wide range of problems in speech recognition, image classification, and time-series prediction, and have been shown to outperform traditional recurrent neural networks and other machine learning approaches in many cases [5]. Despite their success, however, ESNs remain relatively underexplored compared to other types of neural networks, and there is still much to be learned about their behavior and capabilities.

This paper proposes a novel framework to lose the nonlinearity of Echo State Networks and expand their dynamics to be modeled in a system-theory context. In the world of controls engineering and continuous-time systems, it is useful to model systems in state-space notation, denoted by the following:

$$\mathbf{x}'(\mathbf{t}) = \mathbf{A}\mathbf{x}(\mathbf{t}) + \mathbf{B}\mathbf{u}(\mathbf{t})$$
$$\mathbf{y}(\mathbf{t}) = \mathbf{C}\mathbf{x}(\mathbf{t}) + \mathbf{D}\mathbf{u}(\mathbf{t})$$

where $\mathbf{x}(\mathbf{t})$ denotes a vector of the state of a system and $\mathbf{y}(\mathbf{t})$ denotes a vector of observations of $\mathbf{x}(\mathbf{t})$. Unfortunately, traditional ESNs are inherently nonlinear and cannot be adapted to fit this linear state-space continuous-time representation [6].

1.2 Sparse Identification of Nonlinear Dynamics (SINDy)

Sparse Identification of Nonlinear Dynamics (SINDy) is a data-driven algorithm for discovering the underlying dynamics of a system from measured time series data [2]. It is particularly useful for identifying sparse and interpretable models of complex nonlinear systems, where the number of variables and interactions is large, but only a few dominant variables and interactions are significant.

The SINDy algorithm starts by representing the system's dynamics as a set of candidate functions, typically chosen from a library of known nonlinear functions. The algorithm then uses sparse regression techniques, such as Lasso or Least Absolute Shrinkage and Selection Operator (LASSO), to identify the most important functions and their corresponding coefficients.

The resulting sparse model provides a compact representation of the system's dynamics, which can be used for prediction, control, and analysis. SINDy has been successfully applied to a wide range of systems, including fluid dynamics, chemical reaction networks, and neural networks.

The SINDy algorithm has several advantages over traditional model-based approaches, including its ability to handle high-dimensional systems with limited data, its interpretability, and its flexibility to incorporate domain knowledge into the model. However, it also has some limitations, including the need for a carefully chosen library of candidate functions, the computational cost, and the potential for overfitting when the data is noisy or insufficient. SINDy is a powerful tool for discovering and understanding the underlying dynamics of complex nonlinear systems, and it is likely to have many applications in mathematics, physics, engineering, and biology.

1.3 Direct Method for Differentiating Noisy Signals

Total Variation (TV) regularization is a popular technique used in various fields, including image processing and signal analysis, for denoising and smoothing noisy data while preserving edges and details [7]. Specifically, this is the method that SINDy uses to differentiate noisy data. In this context, the TV norm measures the total variation of the signal, which is defined as the sum of the absolute differences between neighboring data points [2].

Total Variation Regularized Derivative (TVD) is a derivative-based regularization technique that utilizes the TV norm to estimate the derivative of a function from noisy measurements. This technique is particularly useful in cases where the measured data is noisy and the derivative is ill-defined or unstable. The TVD method seeks to minimize the TV norm of the derivative of the function subject to the constraint that the estimated function matches the measured data. The resulting optimization problem is typically solved using numerical methods, such as gradient descent or proximal algorithms. TVD has been applied successfully in various fields, including image processing, signal analysis, and machine learning, for tasks such as image denoising, deblurring, and super-resolution. TVD has also been used in medical imaging for reconstructing images from noisy and incomplete data.

Compared to other derivative estimation techniques, TVD has several advantages, including its ability to handle noisy data, its robustness to outliers, and its ability to preserve sharp edges and details. However, TVD also has some serious limitations, including the need for carefully chosen regularization parameters, the potential for introducing artifacts in the estimated derivative, and often exceedingly slow gradient descent convergence.

This paper proposes a technique for developing new norms for differentiating noisy signals, and doing so directly instead of iteratively. This method focuses on penalizing the l_2-norm of the second derivative while also enforcing the error between the antiderivative of the computed derivative and the original function.

2 Echo State Neural Networks

2.1 Traditional ESNs - Theory and Structure

Traditional Echo State Networks are inherently nonlinear systems that function with a group of hidden neurons preserving temporal information and acting as a sort of memory for a dynamical system [3]. Figure 1 depicts the structure of a traditional ESN.

Consider at state u_k that we are trying to model. This u_k can be observed with some observation function to extract pertinent details, like quadratic or sinusoidal nonlinearity to form $\tilde{u}_k$. This vector is then fed into a reservoir of interconnected neurons, whose states are described as x_k, with weights $\mathbf{W_{in}}$. Once the input has been added to the state x, a transition matrix, $\mathbf{W}$, maps x_k to itself. The dimension of x is independent of the system and can be chosen to be as large as is computationally feasible. A nonlinear activation function, usually $\tanh(x)$, is applied to the state x_k. Finally, a proportion of the previous state of x_k and an inverse proportion the updated state after the input, transition and activation is applied to form x_{k+1}. Lastly, x_{k+1} is combined with u_k to solve a least squares regression problem to find u_{k+1}. This is formalized in the following:

Definition 1. *u_k is mapped to u_{k+1} with the following algorithm.*

$$\tilde{u}_k = f(u_k)$$

$$x_{k+1} = \alpha x_k + (1-\alpha)\tanh(\mathbf{W}x_k + \mathbf{W_{in}}\tilde{u}_k)$$

$$u_{k+1} = \begin{bmatrix}\mathbf{W_x} & \mathbf{W_u}\end{bmatrix}\begin{bmatrix}x_{k+1}\\ \tilde{u}_k\end{bmatrix}$$

α is a term used to quantify the leaking rate, or the extent that previous states of x are incorporated into the next state.

In the literature, $\mathbf{W}$ and $\mathbf{W_{in}}$ are chosen randomly, satisfying that the spectral radius, ρ, of $\mathbf{W}$ is less than 1 [3]. This tends to be the large advantage of Traditional ESNs because these weights between the echo state neurons do not have to be trained. This paper is focused on improving upon this framework and optimizing these weights to accurately model dynamical systems. A limitation of this model in the world of controls and understanding the dynamics is its inability to be represented as a single transition matrix because of the nonlinear tanh term. Further, the inputs to x cannot vary significantly in scale because of this nonlinearity; if these values are too large or small, tanh will simply scale them to 1 or 0, making the flexibility of this method to model diverse initial conditions low.

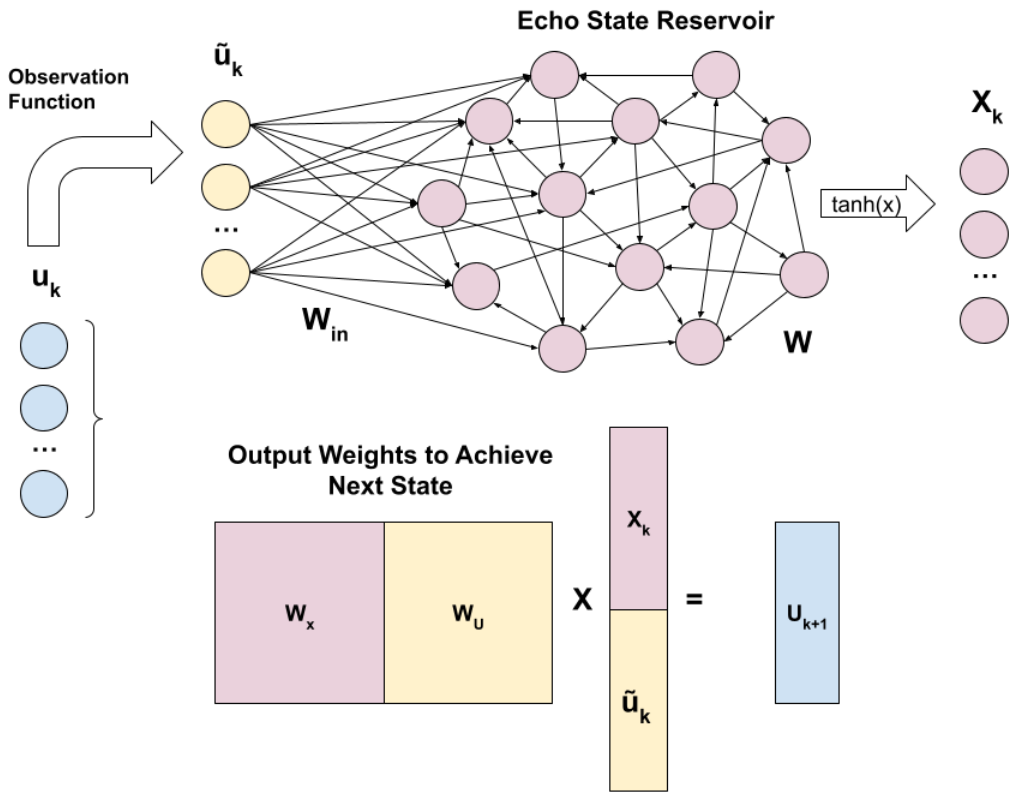

Figure 1: Structure of a Traditional ESN.

2.2 Linear Adaptation of ESNs

To remove some of these problems with nonlinearity and scale, we remove the nonlinearity of the algorithm.

Definition 2. *Consider the following linear version of the Traditional ESNs described in Definition 1.*

$$\tilde{u}_k = f(u_k)$$
$$x_{k+1} = \mathbf{W}x_k + \mathbf{W_{in}}\tilde{u}_k$$
$$\tilde{u}_{k+1} = \begin{bmatrix}\mathbf{W_x} & \mathbf{W_u}\end{bmatrix}\begin{bmatrix}x_{k+1}\\ \tilde{u}_k\end{bmatrix}$$

With this, we can combine the states of x and $\tilde{u}$ to form one dynamical system.

Theorem 1. *A new state, Z_k, can be formed combining states x_k and $\tilde{u}_k$:*

$$Z_k = \begin{bmatrix} x_k \\ u_k \end{bmatrix}$$

We can write a full transition matrix to map Z_k to Z_{k+1}:

$$Z_{k+1} = \mathbf{B} Z_k$$
$$\mathbf{B} = \begin{bmatrix} \mathbf{W} & \mathbf{W_{in}} \\ \mathbf{W_x W} & \mathbf{W_x W_{in}} + \mathbf{W_u} \end{bmatrix}$$

Proof.

$$u_{k+1} = \begin{bmatrix} \mathbf{W_x} & \mathbf{W_u} \end{bmatrix} \begin{bmatrix} x_{k+1} \\ \tilde{u}_k \end{bmatrix} = \mathbf{W_x} x_{k+1} + \mathbf{W_u} \tilde{u}_k$$
$$x_{k+1} = \mathbf{W} x_k + \mathbf{W_{in}} \tilde{u}_k$$
$$u_{k+1} = \mathbf{W_x}(\mathbf{W} x_k + \mathbf{W_{in}} \tilde{u}_k) + \mathbf{W_u} \tilde{u}_k$$
$$= \mathbf{W_x W} x_k + \mathbf{W_x W_{in}} \tilde{u}_k + \mathbf{W_u} \tilde{u}_k$$
$$= \mathbf{W_x W} x_k + (\mathbf{W_x W_{in}} + \mathbf{W_u}) \tilde{u}_k$$
$$\begin{bmatrix} x_{k+1} \\ \tilde{u}_{k+1} \end{bmatrix} = \begin{bmatrix} \mathbf{W} & \mathbf{W_{in}} \\ \mathbf{W_x W} & \mathbf{W_x W_{in}} + \mathbf{W_u} \end{bmatrix} \begin{bmatrix} x_k \\ \tilde{u}_k \end{bmatrix}$$

□

2.3 Differences between Linear and Traditional Echo State Networks

It is important to identify the differences between the Linear ESNs developed in this paper and what is described in the existing literature. The following is a summary of the major differences:

- **Traditional ESNs are nonlinear**. The Echo State neurons outlined in the literature are updated with the following formula:

  $$x_{k+1} = \alpha \cdot x_k + (1 - \alpha) \cdot \tanh(\mathbf{W} x_k + \mathbf{W_{in}} u_k)$$

  In this methodology the α term is expressed in the $\mathbf{W}$ matrix and there is no nonlinearity in updating the Echo State, x_k. Recall how the Echo State neurons are updated in this paper:

  $$x_{k+1} = \mathbf{W} x_k + \mathbf{W_{in}} \tilde{u}_k$$

- **Traditional ESNs do not use observables.** The observed state variable $\tilde{u}_k$ is not explicitly used in the literature.
- **Traditional ESNs require binary input scaling.** In order for the tanh() term to vary consistently and not stay at 1 or -1, the inputs to the Echo State need to be of the same magnitude. The networks developed in this paper do not depend on this condition, making them more flexible.

2.4 Iterative Optimization Algorithm for Linear ESNs

With the framework shown in Theorem 1, if we have enough snapshots of x_k and $\tilde{u}_k$, we can solve for this matrix $\mathbf{B}$ with regularized linear regression. This is tricky, however, because x is entirely dependent on u and is tricky to form. This forms the fixed-point iteration that is used within this paper to optimize the weights $\mathbf{W}$ and $\mathbf{W_{in}}$ so that the system is optimized to match whatever dynamics are present. Consider we have two matrices $\mathbf{Z_0}$ and $\mathbf{Z_1}$ that are formed with snapshots of the system as follows:

Theorem 2. *Consider the following fixed-point iteration. The algorithm is initialized by solving a simple least squares regression with a Moore-Penrose Pseudoinverse. Consider two matrices $\mathbf{\tilde{U}_1}$ and $\mathbf{\tilde{U}_0}$ which form N snapshots of the state, u, after being passed through the observation function:*

$$\mathbf{\tilde{U}_0} = \begin{bmatrix} | & | & & | \\ \tilde{u}_0 & \tilde{u}_1 & ... & \tilde{u}_{N-1} \\ | & | & & | \end{bmatrix}, \mathbf{\tilde{U}_1} = \begin{bmatrix} | & | & & | \\ \tilde{u}_1 & \tilde{u}_2 & ... & \tilde{u}_{N} \\ | & | & & | \end{bmatrix}$$

We can solve the system for some transition matrix, $\mathbf{A}$ that maps $\mathbf{\tilde{U}_0}$ to $\mathbf{\tilde{U}_1}$:

$$\mathbf{A^T} = \mathbf{\tilde{U}_1^T}(\mathbf{\tilde{U}_0^T})^\dagger$$

Where $(\tilde{U}_0^T)^\dagger$ represents the Pseudoinverse of $\tilde{U}_0^T$. From this result, we can initialize our matrix of hidden states, $\mathbf{X_1}$ as the error of this linear system:

$$\mathbf{X_1} = \mathbf{\tilde{U}_1} - \mathbf{A}\mathbf{\tilde{U}_0}$$

We can form $\mathbf{X}_0$ by adding on a column of zeros to $\mathbf{X}_1$ and removing the last column of $\mathbf{X}_1$. With these matrices formed, we can form the larger matrices $\mathbf{Z_0}$ and $\mathbf{Z_1}$:

$$\mathbf{Z_0} = \begin{bmatrix} \mathbf{X_0} \\ \mathbf{\tilde{U}_0} \end{bmatrix}, \mathbf{Z_1} = \begin{bmatrix} \mathbf{X_1} \\ \mathbf{\tilde{U}_1} \end{bmatrix}$$

From this, we can solve for $\mathbf{B}$. It is a good idea to use some form of ridge regression to prevent overfitting. So find a $\mathbf{B}$ such that:

$$||\mathbf{Z_1} - \mathbf{B}\mathbf{Z_0}||_2^2 + \beta||\mathbf{B}||_2^2$$

is minimized. With this new $\mathbf{B}$ *matrix formed, we can simulate the system again, using the following:*

$$Z_k = \mathbf{B}^k Z_0$$

Simulating the system again, we obtain new values for x_k *which are used to form new* $\mathbf{X_0}$ *and* $\mathbf{X_1}$ *matrices. These new* $\mathbf{X_0}$ *and* $\mathbf{X_1}$ *matrices are used along with* $\tilde{\mathbf{U}}_\mathbf{1}$ *and* $\tilde{\mathbf{U}}_\mathbf{0}$ *to form new* $\mathbf{Z_0}$ *and* $\mathbf{Z_1}$ *which can be used to solve for a new* $\mathbf{B}$*. This process can be repeated until some convergence criterion is met.*

This method will be used to optimize the weights to get the best fit possible to nonlinear dynamics. It should be noted that as a fixed-point iteration, sometimes the Jacobian of the system around the fixed point becomes larger than 1; this means that the system jumps around the fixed point but does not converge. I was not able compute the Jacobian by hand, but an area of further research would be perhaps using autodifferentiation software found in many neural network packages to compute either a Jacobian for Newton's method or some gradient descent scheme. Other optimization strategies such as genetic algorithms may prove helpful as well.

2.5 Application to the Lorenz System

In the field of nonlinear modeling, the Lorenz system is an influential mathematical system [8]. Introduced by Edward N. Lorenz in 1963, this system exhibits highly chaotic behavior and serves as a canonical example of a deterministic system that is exceedingly sensitive to initial conditions. The Lorenz system's intrinsic complexity has motivated mathematicians, physicists, and researchers across a wide range of disciplines, oftentimes offering profound insights into the study of chaos theory and the broader understanding of nonlinear dynamical systems.The Lorenz System is characterized by the following system of differential equations:

$$\frac{dx}{dt} = \sigma(y - x)$$
$$\frac{dy}{dt} = x(\rho - z) - y$$
$$\frac{dz}{dt} = xy - \beta z$$

This system is of quadratic nonlinearity which means it is impossible to put the dynamics into purely linear State-Space Notation without some sort of modification. Hence, it is crucial to observe the state, $u = \begin{bmatrix} x & y & z \end{bmatrix}^T$, in such a way as to include these nonlinear terms.

This is performed with some candidate functions that try to guess what type of behavior the underlying dynamics exhibit. Candidate functions might be polynomial functions, exponential functions, logarithmic functions, or sinusoidal functions. The bottom line is that we are trying to capture the essence of the nonlinearity in these candidate functions.

Definition 3. *The Lorenz System is of quadratic nonlinearity, so we can use a set of candidate functions of order 2 multiplying every combination of x, y, and z. This is accomplished with the MATLAB Code in the Appendix, which can produce polynomial combinations of any order given any input. This observed $\tilde{u}$ is as follows:*

$$\tilde{u} = \begin{bmatrix} x & y & z & x^2 & xy & xz & y^2 & yz & z^2 \end{bmatrix}^T$$

Moving into initial conditions so others can replicate this demonstration, we used constants $\sigma = 10$, $\rho = 28$, and $\beta = 8/3$, a timestep $h = 0.005$ and initial conditions $u_0 = \begin{bmatrix} -8 & 7 & 27 \end{bmatrix}^T$. First, we simulate a set of training data using the underlying differential equation and add normally distributed noise to the training data with mean 0 and standard deviation 0.05. We then apply the observable function from Definition 3 to each state in this dataset. Next, let us examine what happens when we solve the first step of the iteration in Theorem 2, that is solving:

$$\mathbf{A}\tilde{\mathbf{U}}_{\mathbf{0}} = \tilde{\mathbf{U}}_{\mathbf{1}}$$

And, subsequently, simulate the dynamical system using the following:

$$\tilde{u}_{k+1} = \mathbf{A}\tilde{u}_k$$

Note that all nonlinear terms are accounted for, and this should be able to approximate the dynamics for at least some period of time. The approximation of the least squares regression with the nonlinear observables is shown in Figure 2.

As shown by the figure, the least squares regression does not accurately capture the dynamics. However, after a few tries, running the iteration described in Theorem 2 converges on the blue line shown in Figure 3.

As shown in Figure 3, the Linear ESN with the iterative method described in Theorem 2 has the ability to match the unstable dynamics with near-perfect accuracy. As we turn up the noise level, the dynamics become more difficult to match, however the algorithm can still match noisy samples for an initial period of time. If we modify the noise term to have a standard deviation of 0.5 (10x the current value), Figure 4 illustrates the predicted dynamics by the Linear ESN model with a noisy input signal. Notice how the dynamics do not match nearly as well, indicating how the x state may be sensitive to perturbations from noise.

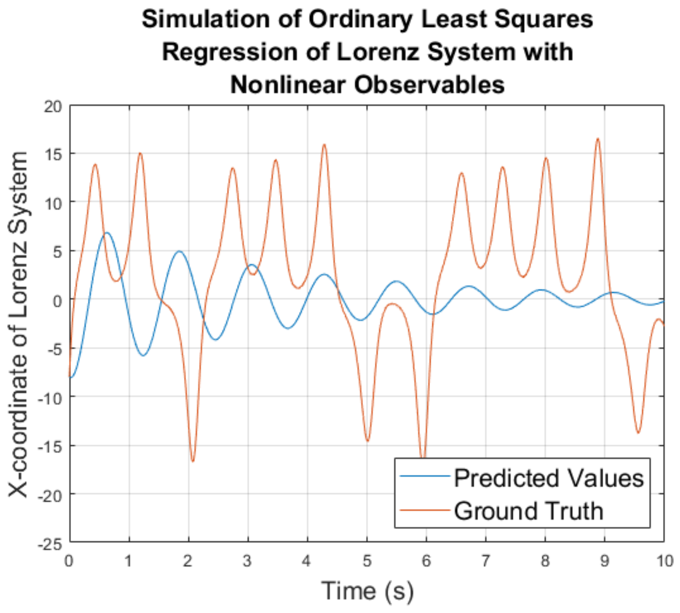

Figure 2: Least Squares Regression model of Lorenz System with nonlinear observable terms.

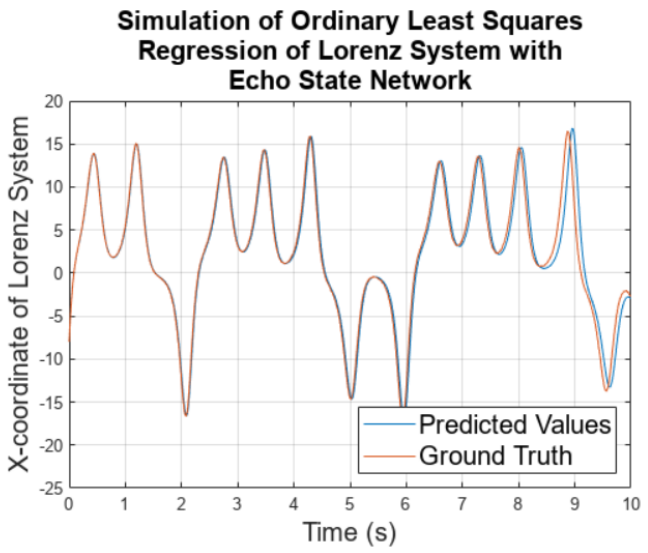

Figure 3: Linear ESN model of Lorenz System with nonlinear observable terms.

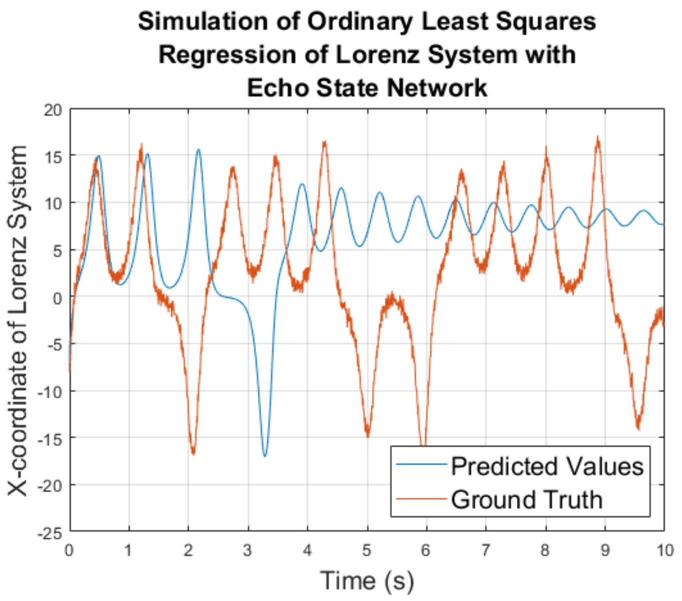

Figure 4: Linear ESN model of Lorenz System with nonlinear observable terms (Noise standard deviation=0.5).

3 Sparse Identification of Nonlinear Dynamics (SINDy) Algorithm

3.1 Introduction

The Sparse Identification of Nonlinear Dynamics (SINDy) algorithm was developed by Stephen Brunton, Joshua Proctor, and J. Nathan Kutz in 2016 as a method to recover governing equations from data [2]. In the pursuit to understand complex dynamical systems, traditional modeling approaches often rely on physical laws or assumptions that may not capture the full complexity of the underlying dynamics. SINDy, on the other hand, offers a data-driven framework that enables the discovery of governing equations directly from observational data, providing a powerful tool for system identification and model discovery.

The main idea behind SINDy is to leverage the inherent sparsity of the governing equations in many real-world systems. By assuming that the true governing equations can be represented by a small number of significant terms, SINDy formulates the problem as a sparse regression task. Given a set of measurements from the system, the algorithm systematically constructs a series of candidate functions comprising potential terms in the governing equations. These functions can include nonlinear terms, spatial derivatives, and other relevant mathematical expressions. Through a process of sparse regression, SINDy then identifies the most parsimonious set of terms that best describe the system's dynamics.

Since its introduction, the SINDy algorithm has found applications in fields such as physics, engineering, biology, and finance. It has proven particularly advantageous in scenarios where data is abundant, but the underlying equations are unknown or difficult to obtain. By providing interpretable and sparse models, SINDy offers valuable insights into the underlying dynamics, uncovers hidden relationships, and facilitates system prediction and control.

This section aims to provide a comprehensive overview of the SINDy algorithm, its theoretical foundations, implementation techniques, and limitations. We explore the algorithm's effectiveness in recovering governing equations from noisy and limited data, discuss the considerations for model selection and hyperparameter tuning, and highlight the strengths and weaknesses of the SINDy framework.

The SINDy algorithm has emerged as a groundbreaking approach for uncovering governing equations directly from data. Its ability to exploit the sparsity inherent in complex systems has revolutionized the field of data-driven modeling. As we delve deeper into the intricacies of complex dynamical systems, SINDy offers an invaluable tool for understanding, predicting, and controlling the behavior of diverse systems.

3.2 Mathematical Framework

The SINDy framework assumes that all dynamical systems can be modeled in terms of their derivatives in the following general form:

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

Similarly to the ESN framework, we use a matrix of snapshots of x over a given time, the only difference being we also need to sample the derivative of x. This can be illustrated as:

$$\mathbf{X} = \begin{bmatrix} | & | & & | \\ x_1 & x_2 & ... & x_N \\ | & | & & | \end{bmatrix}^T, \dot{\mathbf{X}} = \begin{bmatrix} | & | & & | \\ \dot{x}_1 & \dot{x}_2 & ... & \dot{x}_N \\ | & | & & | \end{bmatrix}^T$$

And then the library of nonlinear candidate functions can be modeled by $\mathbf{\Theta}(\mathbf{X})$. These, like the ESN candidate functions are truly up to the user. The more functions, the more computational cost, but the higher probability that the model will find the correct governing equations. An example of $\mathbf{\Theta}(\mathbf{X})$ might be:

$$\mathbf{\Theta}(\mathbf{X}) = \begin{bmatrix} \mathbf{1} & \mathbf{X} & \mathbf{X}^{P_2} & \mathbf{X}^{P_3} & ... & \sin(\mathbf{X}) & ... & \ln(\mathbf{X}) & ... & e^{\mathbf{X}} \end{bmatrix}$$

In this case, like the ESN observables, P_2, P_3 and so on represent polynomial combinations of a certain order as described in Definition 3. Once this matrix of candidate functions evaluated at every single datapoint is formed, the following sparse regression equation can be solved:

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

where Ξ is a matrix of weights that shows how many variables of $\mathbf{\Theta}(\mathbf{X})$ are present in the dynamics. The sparse linear regression involves some sparsity parameter, λ, to cut off terms that are not very present. The idea is that the least squares regression is solved and an initial Ξ is obtained. Then, every term in Ξ that is less than λ is set to zero. And the least squares regression is repeated, but *only* onto the terms that have not been set to zero. This process is repeated until only a handful of terms are remaining and the rest are zero.

Possibly the most overlooked component of this algorithm is the ability to sample the derivative. The original SINDy paper introduced noisy signals into their analysis and illustrated how the model still held up reasonably well with various noise-levels. However, to smooth the noise, the authors used the TVD algorithm described in the introduction. This is an *incredibly* computationally costly algorithm to run at the scale that it was used in this paper, having hundreds of gradient descent steps involving matrices of size 100,000 x 100,000 individually for the x, y, and z coordinates. More on this in **Section 3.4**.

3.3 Rediscovering Dynamics of the Lorenz System

The original SINDy paper simulates the Lorenz System with the same initial conditions and constant values as were used in the ESN section of this paper. However, the original paper uses quite a bit more data. This original paper uses a time-step, $h = 0.001$ and simulates until $t = 100$. With these initial conditions and the same initial level of noise (normally distributed with mean 0 and standard deviation 0.05), the SINDy algorithm produces the following Ξ matrix:

Table 1: Computed Ξ Values and Corresponding Functions

	x	y	z	x^2	xy	xz	x^2	y^2	z^2
$\dot{x}$	**-9.9593**	**9.9599**	0	0	0	0	0	0	0
$\dot{y}$	**-27.6647**	**-0.9337**	0	0	0	**-0.9906**	0	0	0
$\dot{z}$	0	0	**-2.6534**	0	**0.9951**	0	0	0	0

Table 2: Theoretical Ξ Values and Corresponding Functions

	x	y	z	x^2	xy	xz	x^2	y^2	z^2
$\dot{x}$	**-10**	**10**	0	0	0	0	0	0	0
$\dot{y}$	**-28**	**-1**	0	0	0	**-1**	0	0	0
$\dot{z}$	0	0	**-8/3**	0	**1**	0	0	0	0

As shown in Tables 1 and 2, the SINDy algorithm recovers quite close approximations to the underlying dynamics. The question, now, is how do these underlying dynamics compare to the real thing when simulated? The newly simulated system is depicted in Figure 5. Interestingly, it was only able to follow the dynamics until about $t = 5$ before it diverged.

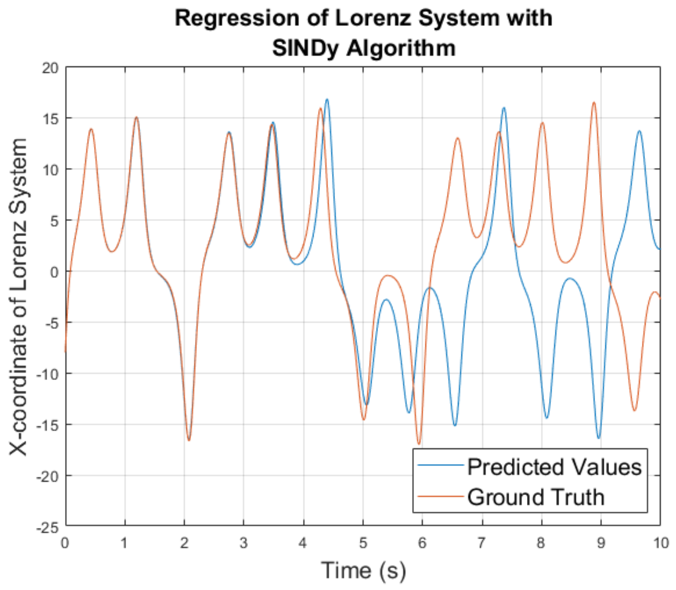

Figure 5: Computed SINDy model of Lorenz System (Noise standard deviation=0.05).

3.4 Limitations of the SINDy Framework

As alluded to in the previous section, the data and its derivative are both 3x100,000 double-precision matrices. Further, the candidate function matrix, $\mathbf{\Theta}(\mathbf{X})$, will scale even higher than this. Let's say one wanted to examine polynomials up to degree five, as is described in the original SINDy paper. This would produce a $\mathbf{\Theta}$ matrix of size 55 x 100,000. If the state had dimension 4 instead of dimension 3, this matrix would be 125 x 100,000. This matrix would take up 100 gigabytes of memory. So without a proper scaled computing system, this method has a significant barrier to entry for recovering system dynamics.

Indeed the 16-core 4.2 GHz machine used for this project with 32gb of RAM had quite a bit of trouble just simulating a ground-truth reference of the system when it already had the underlying dynamics! As mentioned previously, the TVD algorithm via gradient descent proved to be so costly for one machine, that I instead formulated a direct method to solve for the derivative in chunks instead of the gradient descent computation (discussed in **Section 4**).

Further, the SINDy algorithm had nearly 17x more data points than the Echo State Network was given, a timestep that was 5x smaller and still produced a less accurate result than the Echo State Networks. Now, much of the utility of the SINDy framework comes in recovering semantic information about governing dynamics, but this is evidence of the power of Linear ESNs to model complex dynamical systems with a much lower training cost.

4 Direct Method for Differentiating Noisy Signals

As described earlier, the differentiation of noisy signals is crucial to the accuracy of the SINDy algorithm. In this paper, we will consider existing methods and a novel method to compute these derivatives efficiently and accurately. Consider a function $f(x) = |x| + \epsilon(x)$ as an example with a simple, but discontinuous derivative, where $\epsilon(x)$ represents the noise in the function. In this example, it will be normally distributed with mean 0 and standard deviation 0.02. This produces the following reference plot shown in Figure 6.

If we wish to differentiate this function, Figure 7 illustrates just how useless a finite difference approximation of the signal is.

The absolute value function is an example where convolutional smoothing struggles because of the discontinuous derivative jumping from -1 to 1. Kernel-smoothing algorithms take weighted averages of the function across a moving window of points. This makes such a discontinuity difficult to achieve. A popular method for smoothing noisy signals is something known as Savitzky-Golay (SG) filtering [9]. This method is useful when the function has a continuous derivative, but Figure 8 illustrates how the function loses the sharpness of the jump discontinuity of the derivative of the absolute value function, while it does provide a better approximation. This is important to preserve in functions that have sharp changes in slope, even if they are continuous, like the Lorenz system studied in this paper. Another

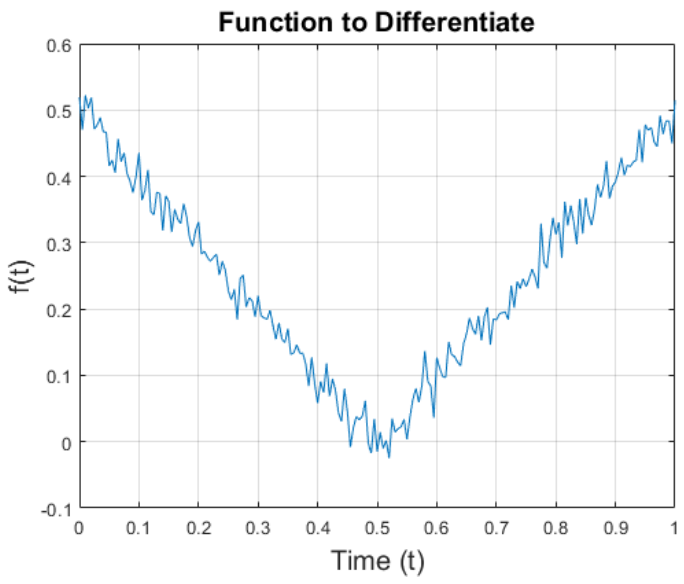

Figure 6: Plot of reference function, $f(t) = |t| + \epsilon(t)$

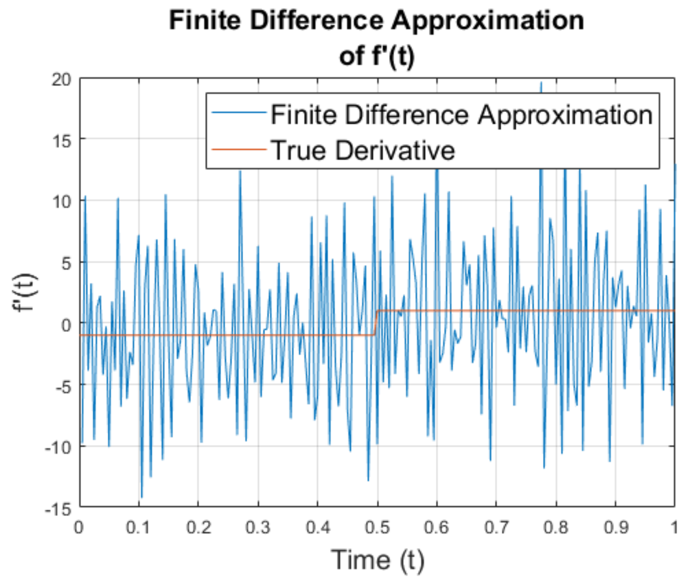

Figure 7: Finite Difference Approximation of $f'(t)$.

setback is the dimensionality loss of convolution. To achieve sufficiently smooth results, Savitsky-Golay filtering has to reduce the dimensionality by the window-size, leading to a reduction in useful derivative data. In Figure 8, the algorithm simply pads the computed derivative with duplicates of the first and last value.

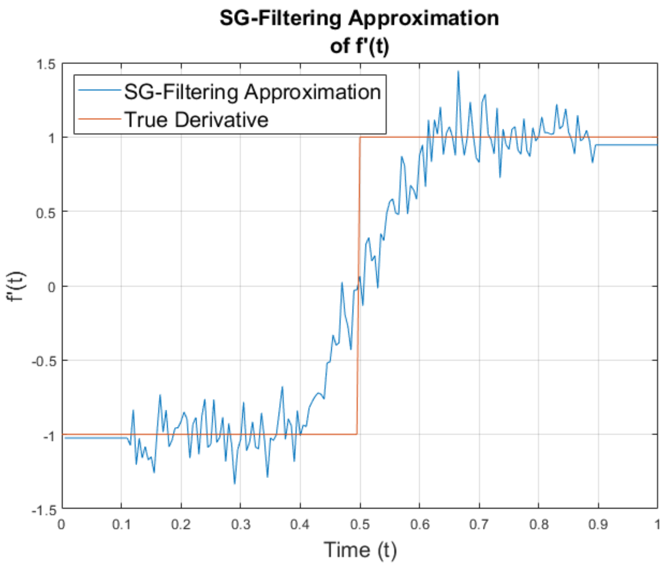

Figure 8: Savitsky-Golay Filtering Approximation of $f'(t)$.

4.1 Modifying the TV-Norm

The current TVD algorithm [7] proposes the following cost-function in terms of the derivative, u, of a function f on some interval $[0, L]$:

$$R(u) = \alpha \int_0^L |u'(t)|dt + \int_0^L |f(t) - \int_0^t u(s)ds|dt$$

This function penalizes the absolute value of the second derivative, $u'(t)$, in the first term and the difference between the function and the antiderivative of the derivative in the second term.

However, a discrete-time implication was not directly described in the paper describing this method, the literature does not give direct implementation for the discrete time case, but simply mentions that gradient descent is the most effective way to target a good approximation of the derivative. Not only this, but the paper cites having to perform thousands of gradient descent steps to reach convergence, which is not only computationally costly, but also severely limits the ability to determine an optimal regularization parameter. This is where the work in this paper begins. Consider the following modified cost function:

$$R(u) = \alpha \int_0^L u'(t)^2 dt + \int_0^L (f(t) - \int_0^t u(s)ds)^2 dt$$

This new cost function that uses a sort of ridge regularization instead of a lasso allows us to make some generalizations. Let us consider a discrete case of dimension 4. In this case, we wish to solve for some vector derivative $u \in \mathbb{R}^4$. Next, it is important to use some linear operator to approximate the derivatives and antiderivatives of this vector u. We can calculate a finite difference derivative using the following matrix, $\mathbf{D}$:

$$u' \approx \mathbf{D}u$$

$$\begin{bmatrix} u_1' \\ u_2' \\ u_3' \end{bmatrix} = \frac{1}{h}\begin{bmatrix} -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{bmatrix}\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix}$$

Notice how with the act of differentiation the dimension of the finite difference derivative decreases by 1. Similarly, we can form a linear operator, $\mathbf{L}$, to aproximate the antiderivative using the trapezoidal integration rule.

$$\int_0^L u\ dt \approx \mathbf{L}u$$

$$\mathbf{L} = \frac{h}{2}\begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 2 & 1 & 0 \\ 1 & 2 & 2 & 1 \end{bmatrix}$$

Now that we have formed these matrices, we can rewrite our cost function R in terms of these discrete operators. For both terms we can simply write them as an inner product of the second derivative to calculate magnitude of the second derivative and the difference between the underlying function and antiderivative.

4.1.1 A note on conditioning f

Because the antiderivative and finite difference operators, $\mathbf{L}$ and $\mathbf{D}$, lower the dimensionality of u by 1, in order for the dimensions to match up in the following computations, f must also have its dimension reduced by 1. This was accomplished by linear interpolation halfway between the values of f. It is also important to ensure that f begins at zero. Implementing a nonzero f_0 value into the algorithm is far messier and, in practice, it is simpler to just force f to begin at zero and then add $f(0)$ onto the computed antiderivative. In the case where $f \in \mathbb{R}^4$, this can be written as:

$$f = \frac{1}{2}\begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}\begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix} - f(0)$$

If the timestep, h, is sufficiently small, this barely alters the function values. Now for the main algorithm in this paper.

Definition 4. *We can define the following cost function for the discrete-time case.*

$$R(u) = \alpha(\mathbf{D}u)^T(\mathbf{L}u) + (f - \mathbf{L}u)^T(f - \mathbf{L}u) \tag{1}$$

Theorem 3. *The regularized derivative can be directly computed by solving the following linear system:*

$$(\alpha\mathbf{D}^T\mathbf{D} + \mathbf{L}^T\mathbf{L})^{-1}\mathbf{L}^T f = u \tag{2}$$

Proof. We can expand (1) to yield the following:

$$R(u) = \alpha u^T\mathbf{D}^T\mathbf{D}u + f^Tf - 2u^T\mathbf{L}^Tf + u^T\mathbf{L}^T\mathbf{L}u$$

We can then compute the gradient with respect to u of the cost function:

$$\nabla R_u = 2\alpha\mathbf{D}^T\mathbf{D}u - 2\mathbf{L}^Tf + 2\mathbf{L}^T\mathbf{L}u$$

We wish to minimize $R(u)$, so we set the gradient equal to zero. This is a linearly solvable system:

$$0 = 2\alpha\mathbf{D}^T\mathbf{D}u - 2\mathbf{L}^Tf + 2\mathbf{L}^T\mathbf{L}u$$

$$\mathbf{L}^Tf = (\alpha\mathbf{D}^T\mathbf{D} + \mathbf{L}^T\mathbf{L})u$$

$$(\alpha\mathbf{D}^T\mathbf{D} + \mathbf{L}^T\mathbf{L})^{-1}\mathbf{L}^Tf = u$$

We can enforce a local minimum with our choice of α. The Hessian of the R can be computed as:

$$\nabla^2 R_u = 2\alpha\mathbf{D}^T\mathbf{D} + 2\mathbf{L}^T\mathbf{L}$$

The $\mathbf{D}^T\mathbf{D}$ term is a tridiagonal matrix with negative values on the super and sub diagonals. As long as α is sufficiently small to lower the magnitude of these negative values enough to be cancelled by $\mathbf{L}^T\mathbf{L}$, which is an entirely positive matrix, then the solution to (2) will be a minimum. Further, the matrix $(\alpha\mathbf{D}^T\mathbf{D} + \mathbf{L}^T\mathbf{L})$ is full rank which means u has to be unique.

□

If $u \in \mathbb{R}^N$, solving this linear system will take something like $O(N^3)$ steps to solve; hence, if N is large, the signal can be split into intervals to reduce computational cost. A plot of the approximated derivative with $\alpha = 5 \cdot 10^{-8}$ is shown in Figure 9. This illustrates a significant improvement in both the smoothness of the line and the preservation of sharp change in continuity.

Not only this but choosing an optimal value of α can be done by creating an L-Plot. On the y-axis, is the error between the original function and the antiderivative of the computed derivative and on the x-axis is the magnitude of the computed second derivative. In other words, $y = (f - \mathbf{L}u)^T(f - \mathbf{L}u)$ and $x = u^T\mathbf{D}^T\mathbf{D}u$. This produces a distinct L-Shape and the optimal α can be chosen at the elbow of this shape. This is depicted in Figure 10. It should be mentioned that computing the derivative of the absolute value function (with its discontinuity, zero concavity, and noise) is difficult; we saw earlier how poorly SG-filtering performed on the data and the performance of this direct method should be shown on a function with a continuous derivative that has some concavity to it as well.

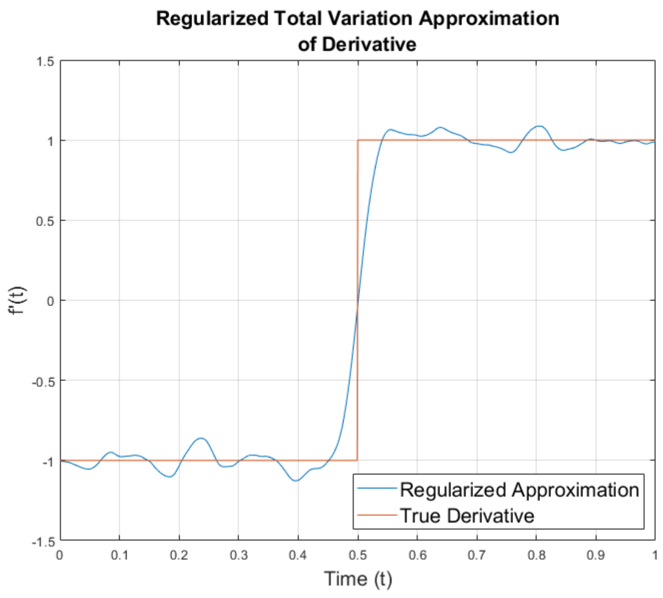

Figure 9: Derivative approximation penalizing the magnitude of the second derivative of the function.

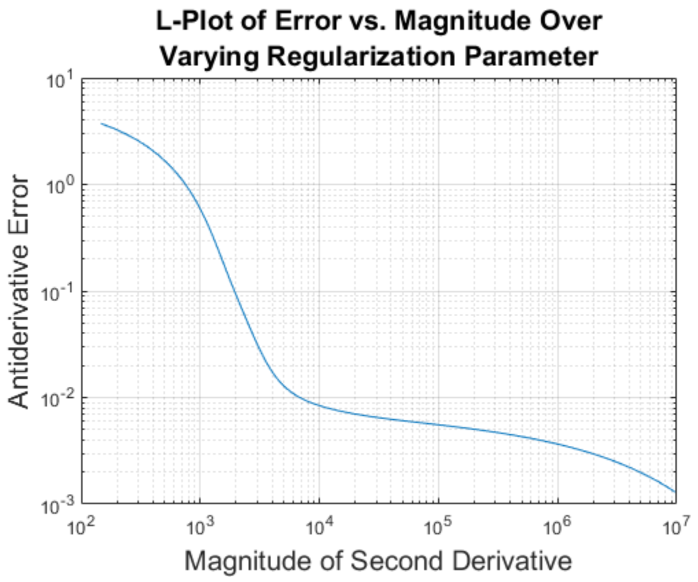

Figure 10: L-Plot of error vs. magnitude of second derivative term.

4.2 Sinusoidal Example

To illustrate the versatility of this method, consider one more example with a sinusoidal wave instead of a simple absolute value function. Consider the following function and corresponding derivative:

$$f(t) = \sin(10t) + \epsilon(t)$$
$$f'(t) = 10\cos(10t)$$

This produces the line in Figure 11. Once again, Figure 12 illustrates the amplified noise in the finite difference approximation. Next, Figure 13 illustrates the direct method for the regularized derivative. In this case, when the derivative is smooth, the Lastly, the L-Plot to choose the optimal α is shown in Figure 14.

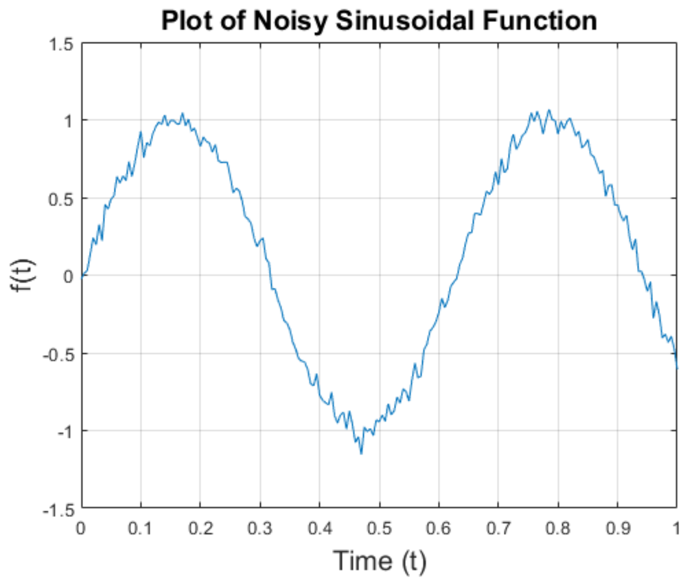

Figure 11: Noisy Sine Function.

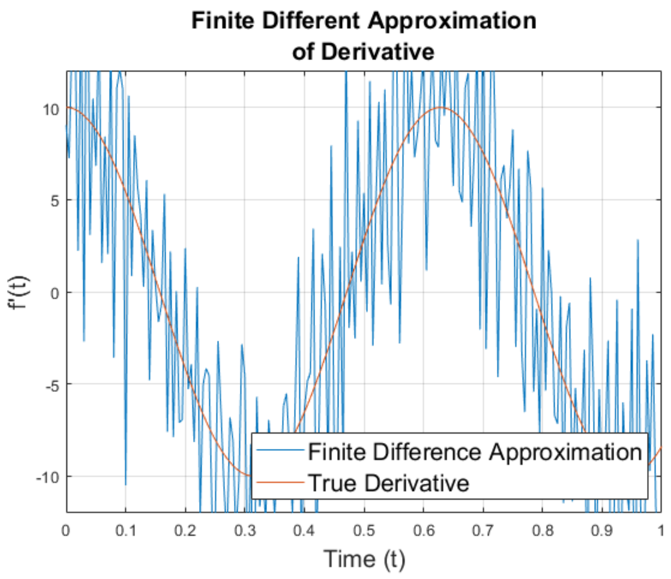

Figure 12: Noise in finite difference approximation.

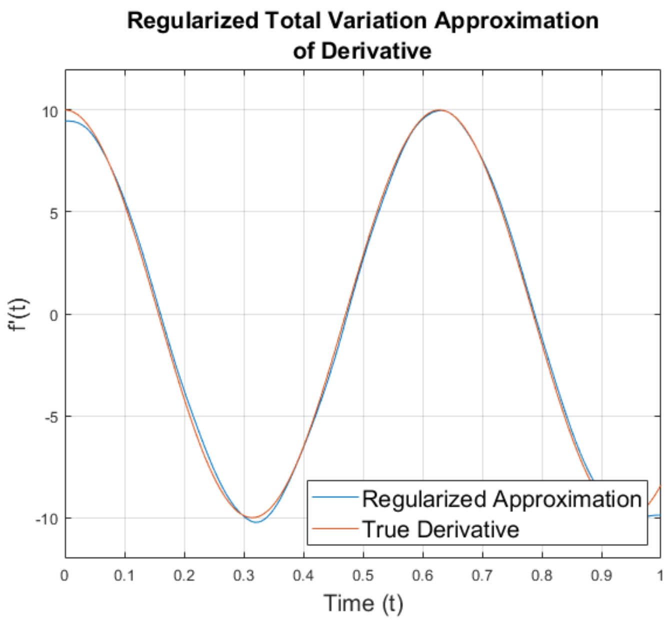

Figure 13: Regularization method implemented with $\alpha = 5 \cdot 10^{-7}$.

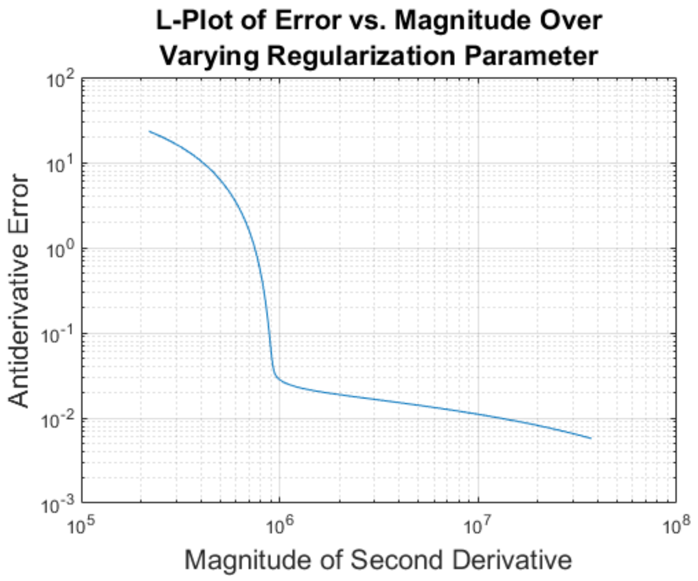

Figure 14: L-Plot of error vs. magnitude of second derivative.

5 Conslusion and Discussion of Results

This paper investigates a novel method for modeling dynamical systems that originated from a recurrent neural network technique, known as Echo State Networks. This technique functions by augmenting information about the state of a dynamical system with a series of memory states that capture information about the current state and also retain information about previous states. This idea was adapted into a linear discrete-time dynamical system and an iterative algorithm was produced to optimize the weights that map the input into this hidden layer of nodes. This algorithm still needs fine-tuning, and possible areas of further research include autodifferentiation techniques to compute a Jacobian or gradient of some cost function, or other optimization strategies. However, this paper illustrates the power that ESNs have to accurately model chaotic systems, like the Lorenz Attractor System, even when they have some noies. When the systems have too much noise, the model does a worse job at accurately reproducing the dynamics.

The SINDy algorithm was used to recover the Lorenz System from data. While this did produce decent approximations of the active nonlinarities in the system, it did not model the system nearly as well as the Linear ESN did, even when it had exceedingly more training data. Not only this, but at the resolution necessary for generating good estimates of coefficients, the SINDy algorithm is extremely computationally costly and the library of candidate functions scales exponentially when analyzing more complex functions. In addition, the need to compute the derivative is a significant challenge to the SINDy algorithm. This calls for a much more efficient method to compute derivatives from data which brings us to the last section of this project.

The Total Variation Regularized Derivative (TVD) algorithm is effective for computing the derivative of noisy signals while preserving sharp edges and detail. That said, it is extremely computationally costly. At the scale required to smooth the derivatives in the SINDy required, it was taking multiple minutes to complete one step of gradient descent, and the literature claimed that it needed on the order of 10,000 steps to converge. Further, this computational cost makes it nearly extremely time-consuming to compare a wide range of regularization parameters. The method proposed in the last section of this paper modifies the TV-Norm discussed in Rudin et al. and develops a discrete optimization framework where the derivative becomes directly solvable as a linear system. Not only does this outperform many popular modern filtering methods, such as Savitsky-Golay Filtering, it is also much faster to compute than the gradient descent method. For functions with strange concavities and jump discontinuities in their derivatives, such as the absolute value function, the algorithm gives a respectable approximation of the derivative. However, this algorithm really shines when the derivative is smooth and continuous. As illustrated in Figure 13, the algorithm nails the derivative of a sine wave, even with a significant amount of noise. Not only this, but because the algorithm is so much faster, we can compute approximations for many different regularization parameters and produce L-Plots to determine the optimal value. This is a huge advantage over the existing TVD algorithm.

6 References

[1] J. Moser, Dynamical Systems, Theory and Applications: Battelle Seattle 1974 Rencontres. Springer, 1975.

[2] S. L. Brunton, J. L. Proctor, and J. N. Kutz, "Discovering governing equations from data by sparse identification of nonlinear dynamical systems," Proceedings of the National Academy of Sciences, vol. 113, no. 15, pp. 3932–3937, 2016. doi:10.1073/pnas.1517384113

[3] M. Lukoševičius, "A practical guide to applying Echo State Networks," Lecture Notes in Computer Science, pp. 659–686, 2012. doi:10.1007/978-3-642-35289-8-36

[4] A. Rodan and P. Tino, "Minimum Complexity Echo State Network," IEEE Transactions on Neural Networks, vol. 22, no. 1, pp. 131–144, 2011.

[5] X. Yao, Z. Wang, and H. Zhang, "Prediction and identification of discrete-time dynamic nonlinear systems based on Adaptive Echo State Network," Neural Networks, vol. 113, pp. 11–19, 2019. doi:10.1016/j.neunet.2019.01.003

[6] S. S. Rao, Mechanical Vibrations. Vancouver, B.C.: Langara College, 2022.

[7] L. I. Rudin, S. Osher, and E. Fatemi, "Nonlinear total variation based noise removal algorithms," Physica D: Nonlinear Phenomena, vol. 60, no. 1–4, pp. 259–268, 1992. doi:10.1016/0167-2789(92)90242-f

[8] X. Wang and M. Wang, "A hyperchaos generated from Lorenz System," Physica A:

Statistical Mechanics and its Applications, vol. 387, no. 14, pp. 3751–3758, 2008.

[9] R. Schafer, "What is a Savitzky-Golay filter? [lecture notes]," IEEE Signal Processing Magazine, vol. 28, no. 4, pp. 111–117, 2011. doi:10.1109/msp.2011.941097

7 Appendix

7.1 Lorenz System Ground Truth Function

```
function dsdt = lorenz(S)

% Constants
sigma = 10;
rho = 28;
beta = 8/3;

% Computing Derivative
dsdt = [sigma*(S(2,:)-S(1,:));...
    S(1,:).*(rho - S(3,:))-S(2,:);...
    S(1,:).*S(2,:)-beta*S(3,:)];

end
```

7.2 Polynomial Combination Function

```
function out = polycomb(vec, order)
n = length(vec);
out = vec;
for i = 2:order
    combs = unique(nchoosek(repelem(1:n,i),i), 'rows');
    newVec = zeros(size(combs,1),1);
    for j = 1:size(combs,1)
        newVec(j) = prod(vec(combs(j,:)));
    end
    out = [out; newVec];
end
end
```

7.3 Computing Lorenz System Ground Truth

```
clear; clf();
%warning('off');
h = 0.005;
tspan = 0:h:30;
y0 = [-8; 7; 27];
func = @(t, S) lorenz(S);
options = odeset('RelTol',1e-10);
[t, Utrue] = ode45(func,tspan,y0,options);
Utrue = Utrue';

eps = 5e-1;
Noise = random('normal',0,1,size(Utrue));
UNoise = Utrue + eps*Noise;
UNoise(:,1) = Utrue(:,1);

figure()
plot(t', UNoise(1,:)); hold on;
plot(t', Utrue(1,:)); hold on;
title("Ground Truth X Coord. vs. Time Plot for Lorenz System")
xlabel("Time");
ylabel("X-Coordinate");
grid on; legend("Noisy U", "True U");
```

7.4 Least Squares Approximation with Observables

```
order = 2;

Uvec = zeros(size(polycomb(UNoise(:,1), order),1), size(UNoise,2));

for i = 1:size(UNoise,2)
    Uvec(:,i) = polycomb(UNoise(:,i), order);
end

% Initializing our Steps
U0 = Uvec(:,1:end-1);
U1 = UNoise(:,2:end);

% Solving for Transition Matrix

C = (U0') \ (U1'); C = C';

lambda = 0.05;
for k=1:25
    smallinds = (abs(C)<lambda); % find small coefficients
    C(smallinds)=0; % and threshold
    for ind = 1:3 % n is state dimension
        biginds = ¬smallinds(:,ind);
        % Regress dynamics onto remaining terms to find sparse Xi
        C(biginds,ind) = ((U0(biginds,:)')\(U1(ind,:)'))';
    end
end

% Seeing how good the initial fit is
Upred = zeros(size(U1));
Upred = [y0 Upred];
for i = 1:size(U0,2)
    Upred(:,i+1) = C*polycomb(Upred(:,i),order);
end
time_cutoff = 2000;
figure();
plot(t(1:time_cutoff), Upred(1,1:time_cutoff));
hold on;
plot(t(1:time_cutoff), UNoise(1,1:time_cutoff));
grid on;
xlabel("Time (s)", "FontSize",15);
ylabel("X-coordinate of Lorenz System", "FontSize", 15);
title(["Simulation of Ordinary Least Squares", "Regression of Lorenz ...
   System with", "Nonlinear Observables"], "FontSize", 15);
lgd = legend("Predicted Values", "Ground Truth");
lgd.FontSize = 15;
lgd.Location = "southeast";
ylim([-25 20])
```

7.5 Computing ESN Fixed-Point Iteration

```
figure();
alpha = 1e-3;
lambda = 0e-8;
beta = 5;
warning('all');
for k = 1:100
    % Making Z Matrices

    Z0 = [X(:,1:end-1); U0];
    Z1 = [X(:,2:end); Uvec(:,2:end)];

    % Making L Matrices

    Z0 = [Z0'; lambda*eye(size(Z0,1))];
    Z1 = [Z1'; zeros(size(Z1,1))];
    B = (Z0) \ (Z1);

   % B = B';

    % Extracting sub-matrices
    n = size(X,1);
    p = size(Uvec,1);

    %alpha = alpha * 0.95; %*norm(linspace(5,0,800).*(UNoise(:,1:800) - ...
        Upred(:,1:800)), 'fro');

    [U, S, V] = svd(B(1:n, 1:n));

    S(S > beta) = beta;
    if (S(1,1) < beta)
        S(1,1) = beta;
    end

    W = alpha * (U*S*(V')) + (1-alpha)*W;
    Win = alpha * B(1:n, n+1:end) + (1 - alpha)*Win;

    for i = 1:size(U0,2)
        X(:,i+1) = W*X(:,i) + Win*polycomb(UNoise(:,i),order);
    end

    Wout = ([X(:,2:end); U0]') \ (U1'); Wout = Wout';
    Wx = Wout(:,1:n); Wu = Wout(:,n+1:end);

    % Re-running system dynamics

    X = zeros(size(X));
    for i = 1:size(U0,2)
        X(:,i+1) = W*X(:,i) + Win*polycomb(Upred(:,i),order);
        Upred(:,i+1) = Wx(1:3,:) * X(:,i+1) + Wu(1:3,:) * ...
            polycomb(Upred(:,i), order);
    end

    clf()
    %fprintf("Norm of error: %f\n", norm(U1-Upred(:,2:end)));
    plot(t(1:time_cutoff), Upred(1,1:time_cutoff)); hold on;
    plot(t(1:time_cutoff), UNoise(1,1:time_cutoff)); drawnow;
    grid on;
    xlabel("Time");
    ylabel("X(t)");
end
```

7.6 Running SINDy Algorithm

```
order = 2;
U = UNoise;
Uvec = zeros(size(polycomb(U(:,1), order),1), size(U,2));

Ytrue = zeros(size(U));

for i= 1:size(U,2)
    Ytrue(:,i) = lorenz(Utrue(:,i));
end

for i = 1:size(U,2)
    Uvec(:,i) = polycomb(U(:,i), order);
end

Y = guessDeriv;

for i = 1:3
    Y(i,:) = sgolayfilt(Y(i,:),1,25);
end

Theta = Uvec'; dXdt = Y';
Xi = Theta\dXdt;

lambda = 0.18;

% lambda is our sparsification knob.
for k=1:25
    smallinds = (abs(Xi)<lambda); % find small coefficients
    Xi(smallinds)=0; % and threshold
    for ind = 1:3 % n is state dimension
```

```
        biginds = ¬smallinds(:,ind);
        % Regress dynamics onto remaining terms to find sparse Xi
        Xi(biginds,ind) = Theta(:,biginds)\dXdt(:,ind);
    end
end

Xi = Xi'

norm(Xi*Uvec - Ytrue, 'fro')

approxFunc = @(t, x) Xi*polycomb(x,order);

options = odeset('RelTol',1e-10);
tspan2 = 0:0.01:20;
[t2, Uapprox] = ode45(approxFunc,tspan2,UNoise(:,1), options);

figure()

plot(t2', Uapprox(:,1));
hold on; grid on;
plot(t(1:10000)', Utrue(1,1:10000));
xlim([0 10])
xlabel("Time (s)", "FontSize",15);
ylabel("X-coordinate of Lorenz System", "FontSize", 15);
title(["Regression of Lorenz System with", "SINDy Algorithm"], "FontSize", ...
   15);
lgd = legend("Predicted Values", "Ground Truth");
lgd.FontSize = 15;
lgd.Location = "southeast";
ylim([-25 20])
```

7.7 Direct Method for Computing Noisy Derivative

```
h = 0.005;
t = (0:h:1)';
n = length(t);
eps = 0.01;
rng(14);
f = abs(t-0.5) + eps*randn(size(t));
%f = sin(10*t) + eps*randn(size(t));
ut = zeros(size(t));
ut(t < 0.5) = -1;
ut(t ≥ 0.5) = 1;
%ut = 10*cos(10*t);

alpha = 1.4e-7;
beta = 0;
```

```
L = tril(ones(n,n)*2);
L = L + diag(ones(n-1,1),1);
L(:,1) = ones(n,1);
L = L(1:end-1,:) * h/2;

D = -diag(ones(n,1)) + diag(ones(n-1,1),1);
D = D(1:end-1,:) / h;

C = 0.5*(diag(ones(n,1)) + diag(ones(n-1,1),1));
C = C(1:end-1,:);
fh = C*f - f(1);

ufd = diff(f) / h; ufd = [ufd; ufd(end)];

u = (alpha * (D'*D) + L'*L + beta*eye(n))\(L'*fh + beta*ufd);

grad = 2*alpha*(D'*D)*u+2*L'*fh+2*(L'*L)*u;

figure()
plot(t, u);
hold on;
plot(t, ut);
ylim([-1.5, 1.5]);
title(["Regularized Total Variation Approximation" "of Derivative"], ...
   "FontSize", 15)
xlabel("Time (t)", "FontSize",15);
ylabel("f'(t)", "FontSize",15);
lgd = legend("Regularized Approximation", "True Derivative");
lgd.FontSize = 15;
lgd.Location = "southeast";
grid on;
```

7.8 L-Plot Formation

```
alphas = logspace(-10, -2, 100);
costs = zeros(size(alphas));
mags = zeros(size(alphas));

for i = 1:length(alphas)
    alpha = alphas(i);
    u = (alpha * (D'*D) + L'*L + beta*eye(n))\(L'*fh + beta*ufd);
    costs(i) = (fh - L*u)'*(fh - L*u);
    mags(i) = u'*(D'*D)*u;
end

figure()
```

```
loglog(mags, costs);
grid on;
xlabel("Magnitude of Second Derivative", "FontSize",15);
ylabel("Antiderivative Error", "FontSize",15);
title(["L-Plot of Error vs. Magnitude Over", "Varying Regularization ...
   Parameter"], "FontSize",15);
```