arXiv:2404.18572v2 [cs.LG] 7 May 2024

Learning Governing Equations of Unobserved States in Dynamical Systems

Gevik Grigorian[a,*], Sandip V. George[b,c], Simon Arridge[c]

[a]*Department of Mechanical Engineering, University College London, London, WC1E 6BT, UK*
[b]*Department of Physics, University of Aberdeen, Aberdeen, AB24 3FX, UK*
[c]*Department of Computer Science, University College London, London, WC1E 6BT, UK*

Abstract

Data-driven modelling and scientific machine learning have been responsible for significant advances in determining suitable models to describe data. Within dynamical systems, neural ordinary differential equations (ODEs), where the system equations are set to be governed by a neural network, have become a popular tool for this challenge in recent years. However, less emphasis has been placed on systems that are only partially-observed. In this work, we employ a hybrid neural ODE structure, where the system equations are governed by a combination of a neural network and domain-specific knowledge, together with symbolic regression (SR), to learn governing equations of partially-observed dynamical systems. We test this approach on two case studies: A 3-dimensional model of the Lotka-Volterra system and a 5-dimensional model of the Lorenz system. We demonstrate that the method is capable of successfully learning the true underlying governing equations of unobserved states within these systems, with robustness to measurement noise.

Keywords: Scientific machine learning, Hybrid neural ODE, Dynamical systems, Symbolic regression, Time series

1. Introduction

Models based on dynamical systems are useful in describing time-dependent phenomena in the real world. Continuous time processes are described using dynamical models based on differential equations. Discovering the underlying governing equations from observed time series data is a problem that has generated considerable interest, particularly with the recent thrust towards data-driven modelling. Such approaches use input from data, instead of *a priori* knowledge, to determine a useful model for the system. An alternate approach is to use data as well as system knowledge to determine appropriate ordinary differential equation (ODE) models. A specific area of interest in the context of the latter is to determine unknown terms of a system of ODEs using observed time series data of the system. Such questions appear in various different areas of application including biology[1], chemistry[2, 3], engineering[4] and the geosciences[5]

A number of approaches have been developed recently in scientific machine learning to determine unknown terms in a system of partially-known differential equations[6]. The framework of universal differential equations (UDEs) is one such approach, where unknown terms in a system of differential equations are replaced by neural networks[7]. These neural networks can be trained on time series data derived from the system, to fully capture its dynamics. This approach is often referred to as a hybrid neural ODE. It is also possible to determine the terms missing from the system through regression of the trained neural network, using techniques such as the sparse identification of nonlinear dynamics (SINDy) or symbolic regression (SR) [6, 8].

While considerable focus has been placed on using hybrid neural ODE approaches to determine unknown terms in partially-known but fully observed systems of differential equations, a more challenging problem is to study systems of differential equations that are only partially-observed. A possible example of this could be the population dynamics of a system of interacting species, where the population of one is difficult to measure. In accordance with Taken's embedding theorem, information about the dynamics of the unobserved variable must be fully captured in the dynamics of the other variables[9].

*Corresponding author.
Email address: `gevik.grigorian.18@ucl.ac.uk` (Gevik Grigorian)

In the recent past, there has been a focus on addressing this question. In [10], the authors attempted to discover the terms involved in an equation corresponding to a single variable when only time series data from that variable was available, using higher-order time derivatives and Lasso regression. The latter is a regression technique that accounts for model simplicity by promoting sparse solutions. In [11] autoencoding and the sparse identification of nonlinear dynamics (SINDy) algorithm were used to discover alternate equations for the Lorenz system when only one of the variables of the system was observed.

In this paper, we use the framework of hybrid neural ODEs and SR to tackle the problem of partially-observed systems. We start by simulating time series data from the 3-dimensional Lotka-Volterra and 5-dimensional Lorenz systems.

The systems are chosen to demonstrate the applicability of the method over a differing number of variables and dynamical behaviours. Next, at least one of the variables is considered to be unobserved. The equation(s) for the unobserved variables are also assumed to be unknown and hence replaced by neural networks, resulting in a hybrid neural ODE. The hybrid neural ODE is then trained using the observed variables, using the mean squared error of the predicted values from the observed values as the loss function. Finally, SR is employed to determine the unknown equations in the model.

2. Methodology

The Lotka-Volterra equations, defined in section 2.1, are a system of nonlinear coupled ODEs which describe predator-prey interactions, a key phenomenon in ecology. The Lorenz equations, defined in section 2.2, are a system of nonlinear coupled ODEs which exhibit chaotic behaviour and are used to model thermal convection. The details of both the training process of the Lotka-Volterra/Lorenz hybrid neural ODEs and the SR stage are presented in section 2.3. Training was carried out in Julia [12], where the original systems and the hybrid neural ODEs were simulated using the Tsit5 ODE solver, with both relative and absolute tolerances of 10^{-6}. SR was implemented using the Python package (with Julia back-end) PySR [13]. Experiments were conducted both without and in the presence of measurement noise. In this work, Gaussian noise at various percentages of the standard deviation of the synthetic data were added to the data.

2.1. 3D Lotka-Volterra System

Many variants of the Lotka-Volterra equations have been proposed, but we employ a simple 3-dimensional version, describing the interactions between 3 species in a system. The model equations are given by

$$\frac{dx}{dt} = ax - bxy, \tag{1.1}$$

$$\frac{dy}{dt} = -cy + dxy - eyz, \tag{1.2}$$

$$\frac{dz}{dt} = -fz + gyz, \tag{1.3}$$

where x, y and z are the 3 state variables representing the 3 species. The parameters a, c and f represent the growth rates of species x, y and z, respectively, and the parameters b, d, e and g represent the effects of predation or competition on their growth rates. In this work, we use a value of 1.0 for all parameters a, b, c, d, e, f, g, as well as an initial condition of $x_0 = 0.5, y_0 = 1.0, z = 2.0$. The dynamics of this system are periodic and therefore do not depend significantly on parameter values and initial conditions. This system is simulated for 20 units of time, at a sampling rate of 0.05 (frequency of 20 Hz), generating a synthetic data set of 1200 data points (400×3). Given the periodic behaviour of the system, 20 units of time was selected to capture a number of full periods. The dynamics of this simulation are shown in Figure 1.

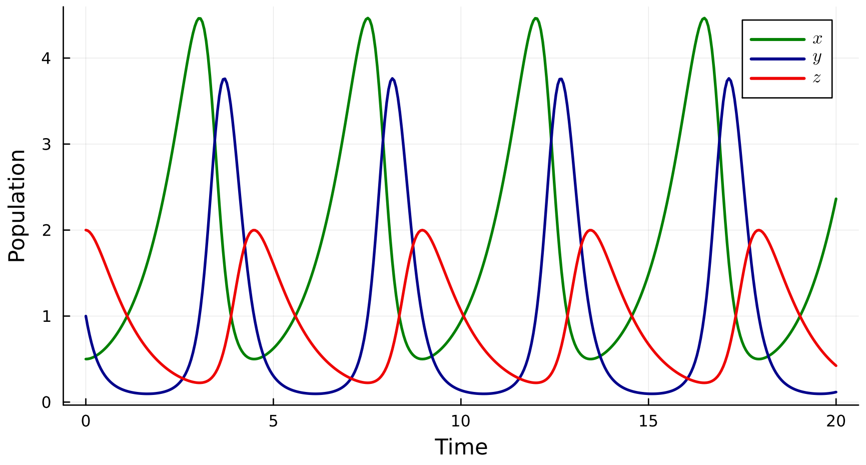


Figure 1: Temporal evolution of the 3D Lotka-Volterra system with initial condition $[x_0, y_0, z_0] = [0.5, 1.0, 2.0]$. This synthetic data is used for training and validation.

We now assume that equation 1.2 is unknown and that measurement data for y are not available. We therefore construct a hybrid neural ODE by replacing equation 1.2 with $\frac{dy}{dt} = N_\theta(x, y, z)$, where N_θ is a neural network parameterised by the set of weights and biases θ. The architecture of the network is a fully connected network with 3 inputs (x, y, z), 2 hidden layers of 40 neurons each with Gaussian error linear unit (gelu) activation functions, and a single output. The activation gelu is defined as $\text{gelu}(x) = x\Phi(x)$, where $\Phi(x)$ is the standard Gaussian cumulative distribution function. It can also be approximated by $\text{gelu}(x) \approx 0.5x\left(1 + \tanh\left[\sqrt{\frac{2}{\pi}}\left(x + 0.044715x^3\right)\right]\right)$, which is the form used in this work. The single output of the network represents the dynamics of $\frac{dy}{dt}$. The synthetic data is considered as ground truth and a window of this data is selected for training. Given that this system generates periodic dynamics, a training range of less than one period of the data is selected, namely $0 - 2$ units of time. This means the training set consists of 80 data points (40 points in x and z each, while y is unobserved). The experiment is repeated at 0%, 2% and 5% noise added to the training data.

2.2. 5D Lorenz System

We employ the 5-dimensional version of the Lorenz system, defined as

$$\frac{dx}{dt} = \sigma(y - x), \tag{1.4}$$

$$\frac{dy}{dt} = x(\rho - z) - y, \tag{1.5}$$

$$\frac{dz}{dt} = xy - \beta z - xv, \tag{1.6}$$

$$\frac{dv}{dt} = xz - 2xw - (1 + 2\beta)v, \tag{1.7}$$

$$\frac{dw}{dt} = 2xv - 4\beta w, \tag{1.8}$$

where the state variables x, y, z, v, w collectively represent the spatial distribution and dynamics of temperature deviations within a convective cell, including horizontal and vertical variations, as well as additional aspects of the system's behavior. σ is the Prandtl number, ρ is the Rayleigh number and β is the ratio of the width to the height of the convective cell. We use the parameter values $\sigma = 10, \rho = 35, \beta = \frac{8}{3}$ and the initial conditions $x_0 = -8, y_0 = 8, z_0 = 27, v_0 = 0.4, w_0 = 0.5$. These values are common choices in the literature and are selected here as they correspond to a state of the system that lies close to the attractor, minimising any initial transients. A simulation of 6 units of time is carried out, using a sampling rate of 0.01 (frequency of 100 Hz), resulting in a data set of 3000 data points (600×5). Since the Lorenz system exhibits chaotic behaviour, 6 units of time is selected in order to capture a significant portion of the dynamics while also concentrating on the range where predictions may deviate from the ground truth. The dynamics of this system are shown in Figure 2.

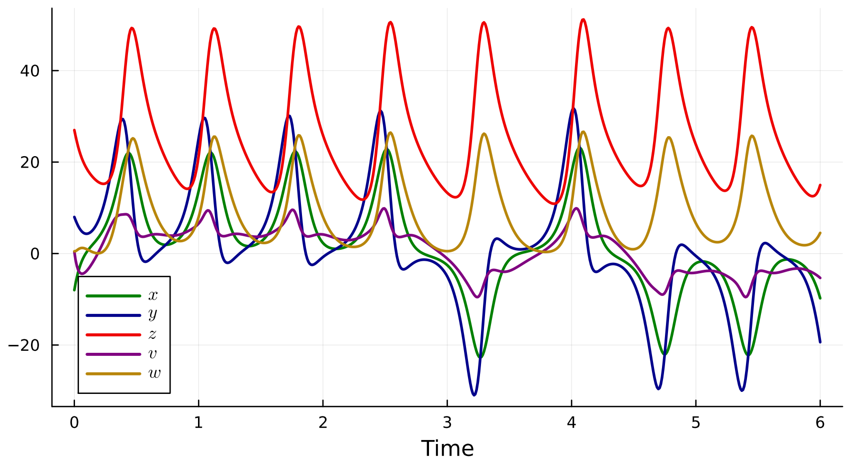


Figure 2: Temporal evolution of the 5D Lorenz system with initial condition $[x_0, y_0, z_0, v_0, w_0] = [-8.0, 8.0, 27.0, 0.4, 0.5]$. This synthetic data is used for training and validation.

We now assume that equations 1.4 and 1.5 are unknown and that measurement data for x and y are unknown. We construct a hybrid neural ODE by replacing equations 1.4 and 1.5 with $\frac{dx}{dt} = N^1_\theta(x, y, z, v, w)$ and $\frac{dy}{dt} = N^2_\theta(x, y, z, v, w)$, respectively. Here, N_θ is a network parameterised by the set of weights and biases θ. The architecture of the network is a fully connected network with 5 inputs (x, y, z, v, w), 2 hidden layers of 40 neurons with gelu activation functions (defined in section 2.1), and 2 outputs. The two outputs are specified by the superscripts 1 and 2. Again, the synthetic data is considered as ground truth and the training range was selected to be the data between 0 - 0.25 units of time, resulting in a training data set of 75 points (25 points in z, v and w each, with x and y being unobserved). The chaotic nature of this system together with the task of learning two governing equations, poses a significantly more challenging problem than that which is outlined in section 2.1. Given this added difficulty, this experiment is only repeated at 0% and 0.3% noise added to the training data.

2.3. Methodology Details

For the problems outlined in sections 2.1 and 2.2, the training process, the implementation of SR and further analysis on the lengths of the training ranges are detailed in this section.

2.3.1. Training

The hybrid neural ODEs have the form

$$\dot{\mathbf{U}} = \begin{pmatrix} F_{\mathbf{C}} \\ N_\theta \end{pmatrix} (\mathbf{U}), \tag{1}$$

where $\mathbf{U}$ is the vector of state variables ($[x, y, z]$ for the Lotka-Volterra case and $[x, y, z, v, w]$ for the Lorenz case), F_C is a set of prescribed mathematical operations with parameters C (equations 1.1 and 1.3 for the Lotka-Volterra case, and equations 1.6, 1.7 and 1.8 for the Lorenz case) and N_θ is a neural network, parameterised by θ. A normal ODE corresponds to the case where the vector of the states is related to its derivative through mathematical operations (F_C) only. A neural ODE corresponds to the case where the state vector is relative to its derivative through a neural network (N_θ) only. Hence, in this case, a hybrid neural ODE refers to the relationship between the state vector and its derivative being governed by some combination of mathematical operations and a neural network.

The training process involves simulating the hybrid neural ODE for the length of the training range (using the same solver, tolerances, and sampling rate) in order to generate a prediction. During training, the unobserved states are removed from both the ground truth data and the prediction. The remaining values (only the observed states) from the ground truth and the prediction are then compared in order to generate a loss value. The loss function used in this work is mean squared error (MSE), defined as

$$L = \frac{1}{D}\sum_{i=1}^{D}(u_i - \hat{u}_i)^2, \tag{2}$$

where L is the loss value, D is the number of data points, u_i is the ith element of the ground truth (after removing the unobserved states) and $\hat{u}_i$ is ith element of the prediction (after removing the unobserved states).

Training is then the process of minimising this loss function by optimising the weights and biases of the network. The optimisation is done in two stages, as in [7]. First, Adam [14] is used to move the network parameters into a more favourable starting position for the L-BFGS optimiser [15], which is used subsequently. Adam is a common choice as it is easy to implement and works efficiently with little hyper-parameter tuning. At this stage, the Adam optimiser is used with a learning rate of 0.01 for 1000 iterations. However, Adam may sometimes converge slowly and therefore the L-BFGS optimiser (a quasi-Newton algorithm) is used subsequently for faster convergence (and sometimes towards a better optimum), as it uses second-order information about the loss function (the Hessian matrix) to make more informed updates to the network parameters. For this second stage, the L-BFGS optimiser is used for 1000 iterations.

In this work, an ensemble of ten hybrid neural ODEs is trained. This means that for each of the cases outlined in sections 2.1 and 2.2, the training process is carried out 10 times. Due to the random initialisation of the network parameters, the training process results in a different set of learned parameters each time and therefore different predictions. Averaging over these predictions is a means of improving accuracy and obtaining desired results more consistently. In this work, a mean of the outputs of the ten networks in the ensemble is taken. This training process is repeated in the same way at each level of noise.

2.3.2. Symbolic regression

Upon completing the training process, any of the ten hybrid neural ODEs in the ensemble can be used to make extrapolations (predictions beyond the range of training data). While these predictions are expected to be more accurate than that of a black box model (due to the physical knowledge encoded in the system equations), they can still lack some predictive accuracy since neural networks are known to be poor at extrapolating. This would especially be true if the training data captures only a small portion of the system dynamics. This motivates the use of a sparse regression technique, where the neural network in the hybrid neural ODE is converted to symbolic expressions. This step results in a partially-learned model with greater interpretability than the hybrid neural ODE, but can also improve the extrapolation capabilities.

In this work, SR is used for this step. SR is a machine learning technique that fits analytic expressions to data. It requires a set of unary operators (e.g. sin, cos, exp, etc) and a set of binary operators (e.g addition, subtraction, multiplication, division, etc) to be defined by the user. The function space defined by these operators is then searched in a 'brute force' manner via genetic programming. Processes such as mutations, crossovers and tournaments encourage a 'survival of the fittest' environment among different candidate expressions. For a more detailed description of SR, see [16] and [17].

SR requires a set of inputs and a target set (the data to fit an expression to). First, each of the 10 hybrid neural ODEs in the ensemble are simulated for the length of the training range. The mean of the inputs to the networks ($[x, y, z]$ for the Lotka-Volterra problem and $[x, y, z, v, w]$ for the Lorenz problem) is taken. This averaged input is treated as the input to SR. The mean of the outputs of the networks ($[x, y, z]$ for the Lotka-Volterra problem and $[x, y, z, v, w]$ for the Lorenz problem) is also taken. This averaged output is treated as the target data for SR. Table B.3 in Appendix Appendix B shows the hyper-parameters used in the PySR implementation.

2.3.3. Varying the lengths of the training range

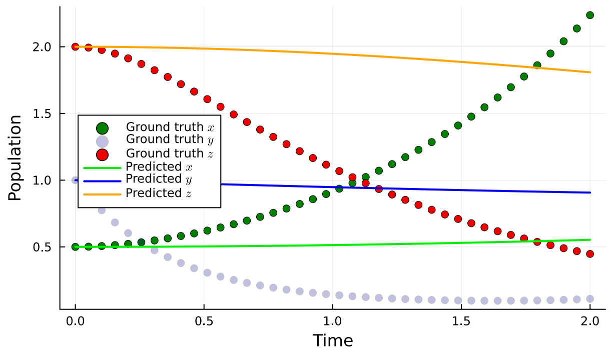
For both cases, further analysis is also done to investigate the effects of varying the length of the training range on the accuracy of the extrapolations. To this end, a hybrid neural ODE is trained using three different training ranges for each case study. For the Lotka-Volterra case, the training ranges are 2.0, 3.25 and 4.5 units of time, while for the Lorenz case, the training ranges are 0.4, 0.8 and 1.2 units of time. For each training range (and each level of noise), a sliding window approach is used, where the root MSE (RMSE) between the prediction and the ground truth is calculated within a window of 20 data points. The errors accumulate as the window shifts along to the end of the simulation, and these errors are compared.

3. Results

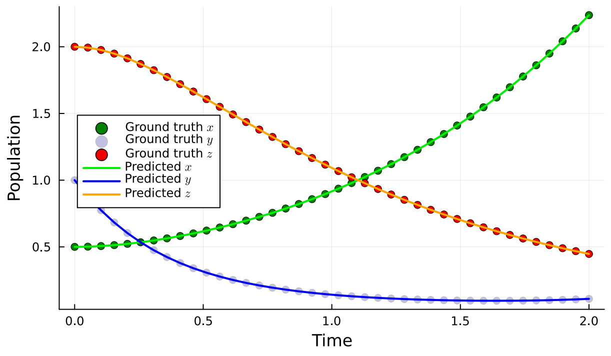
Once each of the ten hybrid neural ODE models in the ensemble are trained, the untrained and trained predictions of a randomly selected model from the ensemble is shown, as well as an extrapolation to examine the model's ability to generalise beyond the training range. Corresponding extrapolations of the partially-learned models trained on noisy data are given in Appendix Appendix A.

3.1. 3D Lotka-Volterra System

Figures 3a and 3b show the predictions of a hybrid neural ODE from the ensemble before and after the training process, respectively. In both of these plots, the ground truth of the y state variable is faint to emphasise that this data is not available to the model during training.



(a) Untrained hybrid neural ODE prediction



(b) Trained hybrid neural ODE prediction

Figure 3: Prediction of the hybrid neural ODE (shown by the solid curves) against the ground truth data (shown by the scatter points), both before (Figure 3a) and after (Figure 3b) the training process. The ground truth data for the state variable y is faint as this data is not available during training.

Figure 3 shows that the predictions after the training process are very accurate, which is expected given that neural networks are universal function approximators and are therefore powerful at interpolating (making predictions within the training range). An extrapolation of this hybrid neural ODE is subsequently carried out up to 20 units of time, as shown in Figure 4, to examine its ability to generalise beyond the range of training data.

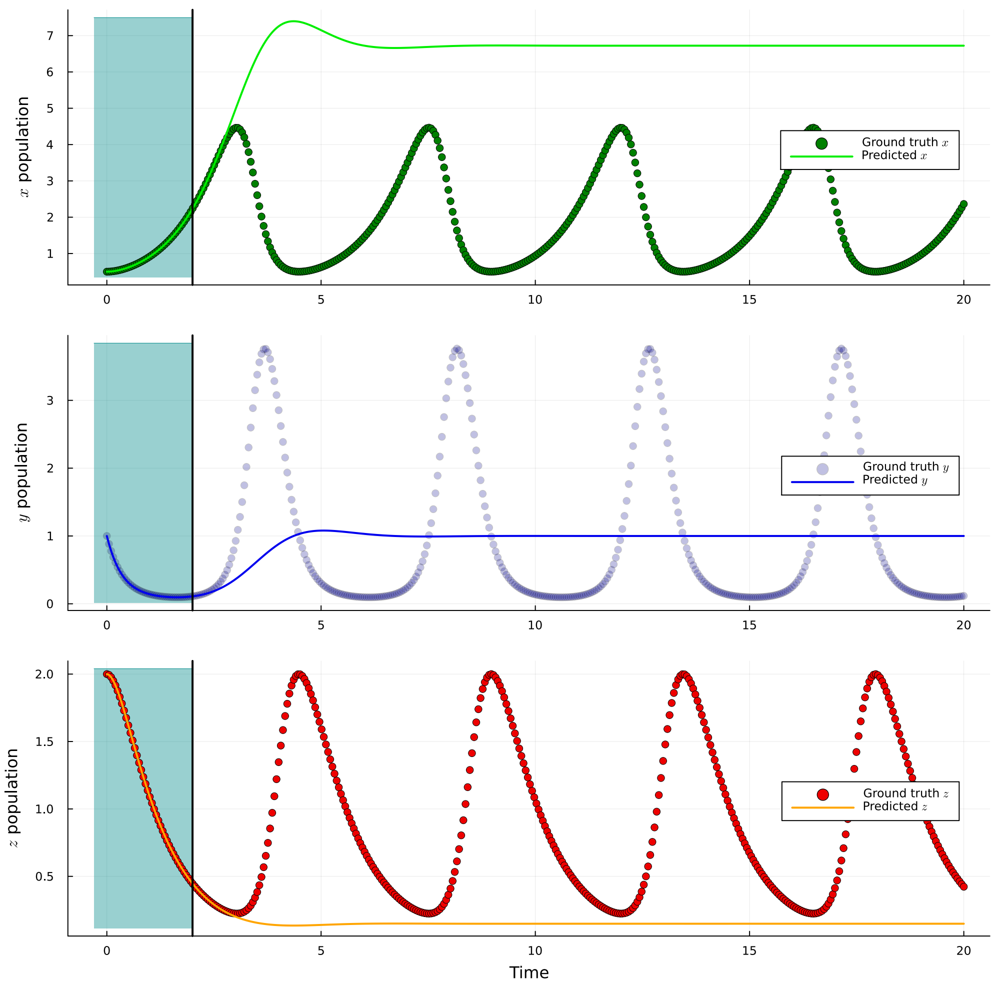
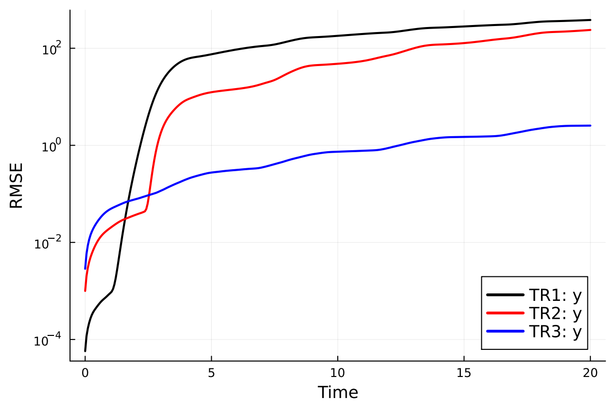
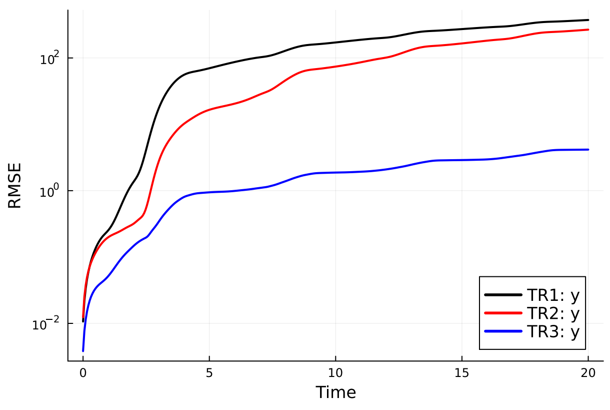


Figure 4: Extrapolation of trained hybrid neural ODE. The predictions are shown by the solid curves and the ground truth is shown by the scatter points. The ground truth for the state variable y is faint as this data is assumed to be unavailable. The blue region to the left of each plot represents the training range.

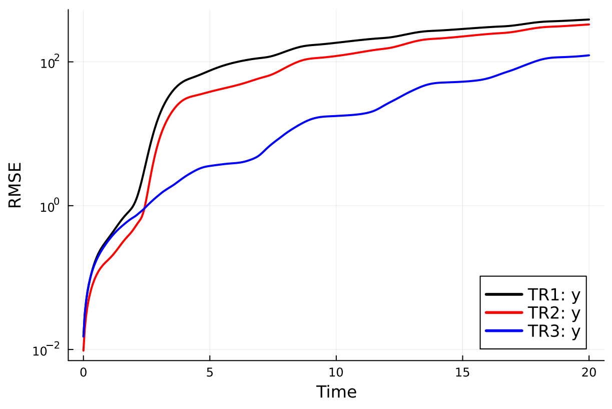
From Figure 4, it can be seen that the extrapolation of the hybrid neural ODE is poor. While neural networks are generally weak at extrapolating beyond their training range, it is expected that the added domain knowledge present in the hybrid neural ODE (equations 1.1 and 1.3) may guide the neural network to produce more accurate extrapolations. However, in this case, the poor extrapolation capabilities of the neural network prevails due to the short range of dynamics captured within the training data. Figure 5 shows the errors accumulated via the sliding window approach on the extrapolations of the hybrid neural ODE using three different training ranges. The accuracy of the predictions increases as the training range increases. This trend is expected as a longer training range provides the neural network with more information about the system. The periodic nature of the dynamics also explains this behaviour, since the greater the proportion of a full period of the data that is captured within the training range, the more likely it is to achieve accurate extrapolations. In this work, the longest training range (4.5 units of time) corresponds to roughly one full cycle of the data.



(a) 0% noise



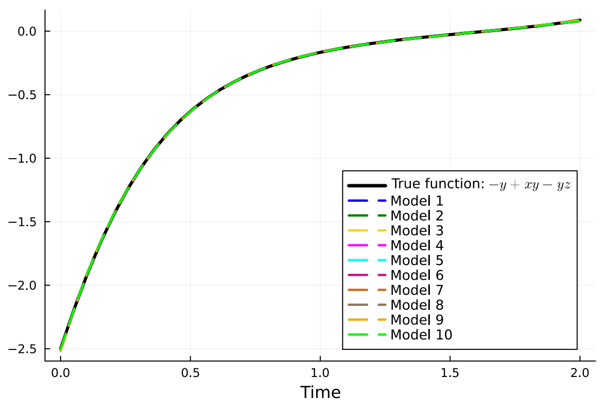
(b) 2% noise



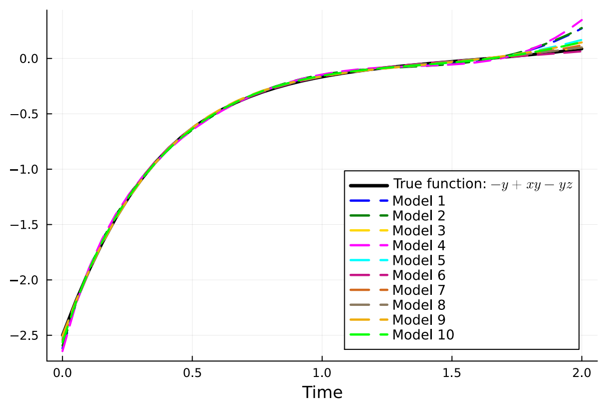
(c) 5% noise

Figure 5: Sliding window errors of hybrid neural ODE extrapolations, using three different training ranges. The training ranges of 2.0, 3.25 and 4.5 units of time are abbreviated to TR1, TR2 and TR3, respectively. Only the unobserved state variable y is shown. This comparison is done at each level of noise.

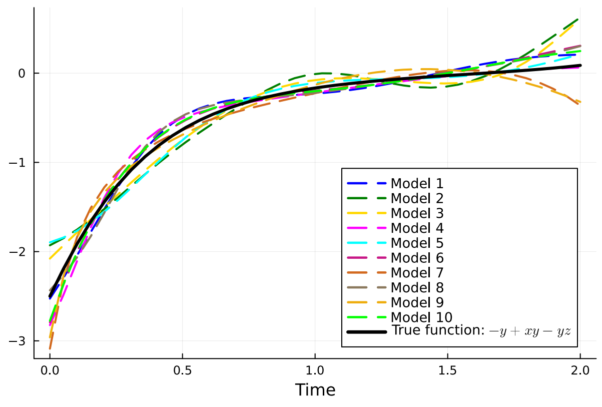
Returning to the case of a training range of 2.0 units of time, the use of SR is motivated, which is applied to the averaged output of the ten neural networks in the ensemble. To this end, each of the trained hybrid neural ODEs are simulated for the length of the training range to generate the temporal evolution of the states $[x, y, z]$, which the neural networks take as input to generate an output representing the learned dynamics for the $\frac{dy}{dt}$ equation. This is done at each level of noise, as shown in Figure 6.



(a) 0% noise



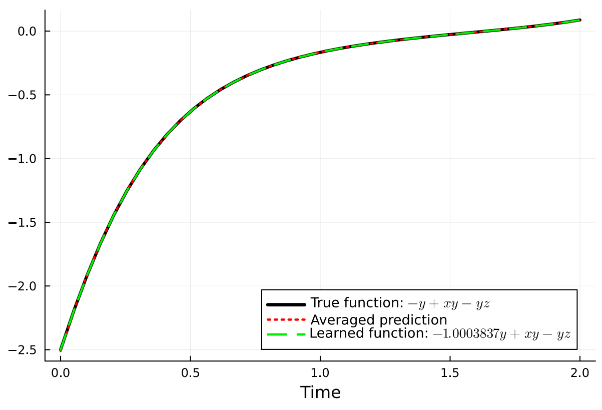
(b) 2% noise



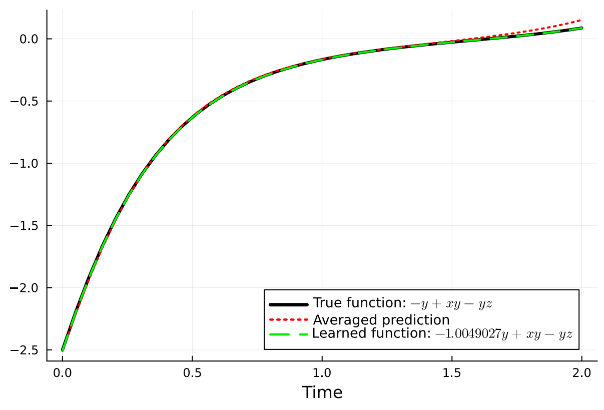
(c) 5% noise

Figure 6: The predicted dynamics of the $\frac{dy}{dt}$ equation generated by each network in the ensemble, at each level of noise added to the ground truth data.

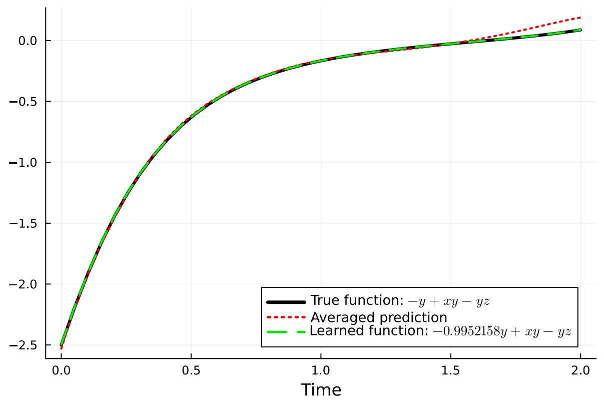
Figure 6a shows that each network generates a very similar and accurate prediction. Therefore, the method of training an ensemble of models and applying SR to the averaged prediction is likely not necessary in the absence of measurement noise, but is still used for consistency. Figures 6b and 6c show increased variability in the predictions, as expected. Figure 7 shows the averaged predictions of the networks (red dotted curves) and the learned functions through SR (green dashed curves), at each level of noise.



(a) 0% noise



(b) 2% noise



(c) 5% noise

Figure 7: The averaged predicted dynamics of the $\frac{dy}{dt}$ equation generated by each of the networks in the ensemble (red) and the corresponding learned function (green), at each level of noise added to the ground truth data.

As expected, the averaged prediction in the absence of noise (Figure 7a) follows the true dynamics very closely. For 2% and 5% noise (Figures 7b and 7c, respectively), there is a similar deviation of the averaged prediction from the true dynamics towards the end of the training range. Despite this, SR is able to recover the equation with the correct symbolic form due to the remainder of the dynamics being accurately approximated by the averaged prediction. Table 1 shows the equations discovered by SR at each level of noise.

Noise	**Learned Equation**	**True Equation**
0%	$\dot{y} = -1.0003837y + xy - yz$	
2%	$\dot{y} = -1.0049027y + xy - yz$	$\dot{y} = -y + xy - yz$
5%	$\dot{y} = -0.9952158y + xy - yz$	

Table 1: Learned equations for $\frac{dy}{dt}$ at each level of noise added to the ground truth data. The target data for SR was the averaged prediction of the ensemble (red curves in Figure 7).

Table 1 shows that at each level of noise, the correct equation structure is discovered, with the only difference being the coefficient of the y term. The learned equation in the absence of noise is then substituted for the neural network in the hybrid neural ODE, resulting in a partially-learned system, where equation 1.2 is set to be $\frac{dy}{dt} = -1.0003837y + xy - yz$.

To examine this partially-learned model's ability to generalise, another extrapolation up to 20 units of time is made. The resulting predictions are shown in Figure 8, which accurately track the dynamics of the true system.

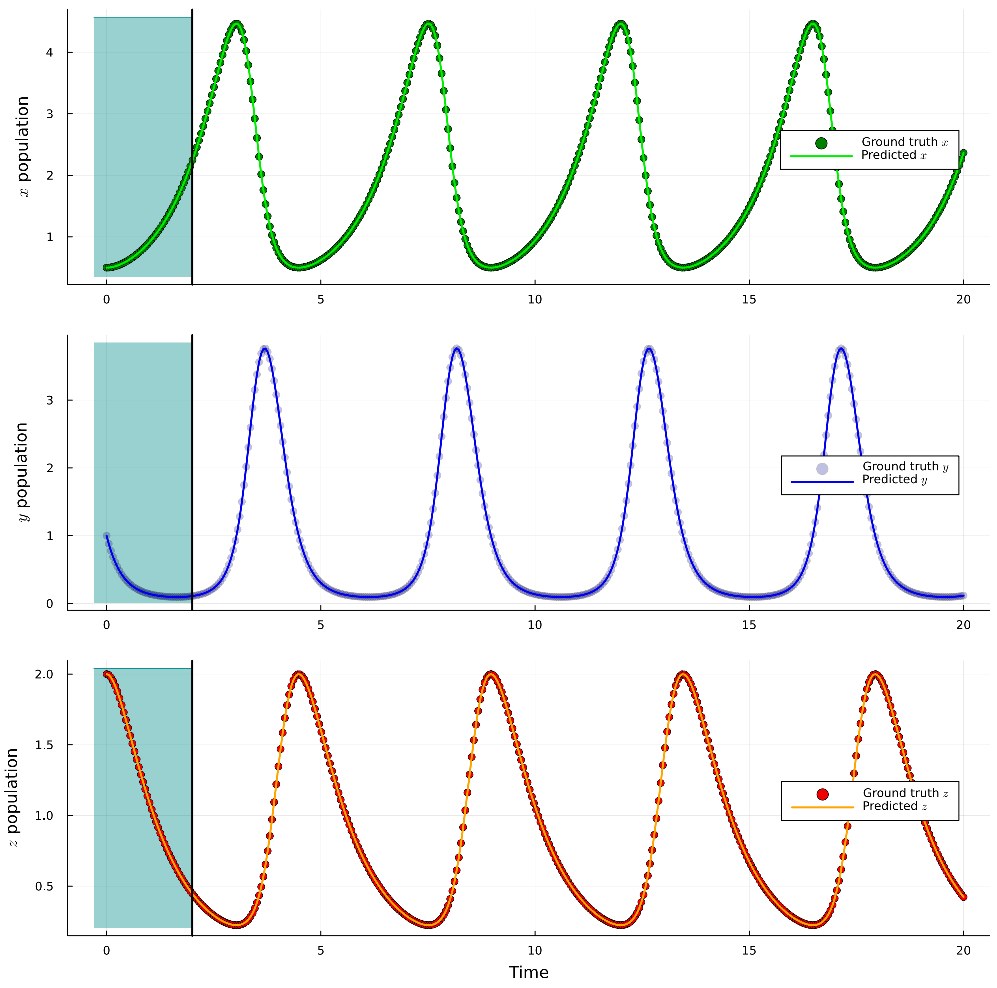
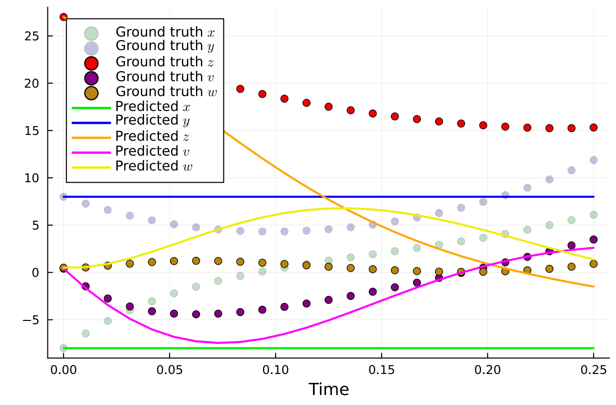


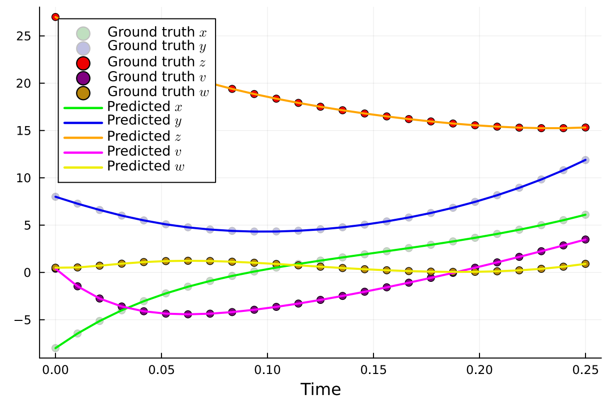
Figure 8: Extrapolation of the partially-learned model, where $\frac{dy}{dt} = -1.0003837y + xy - yz$. The predictions are shown by the solid curves and the ground truth is shown by the scatter points. The ground truth for the state variable y is faint as this data is assumed to be unavailable. The blue region to the left of each plot represents the training range.

3.2. 5D Lorenz System

Figures show the predictions of one of the ten hybrid neural ODEs from the ensemble before and after the training process, respectively. In this case, the x and y state variables are faint as these data are not provided to the model in training.



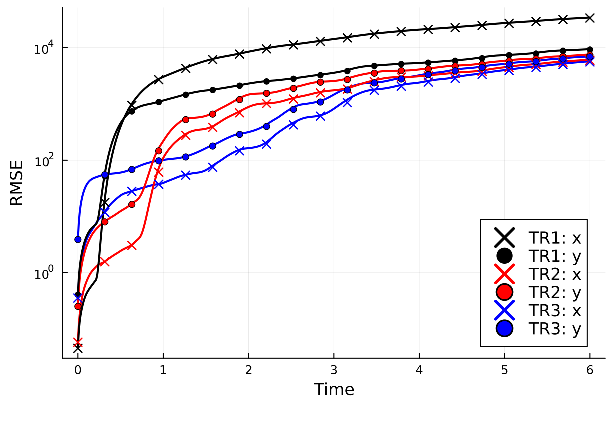
(a) Untrained hybrid neural ODE prediction



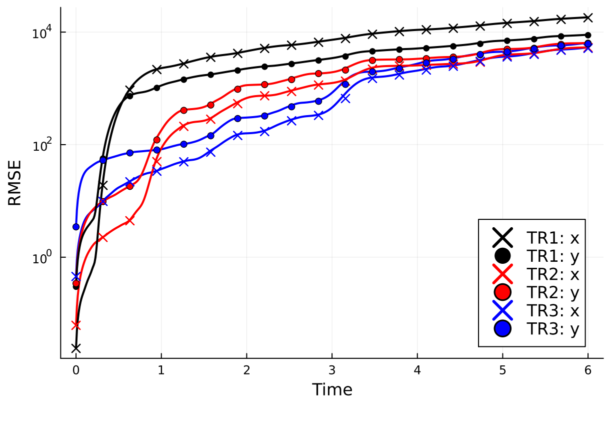
(b) Trained hybrid neural ODE prediction

Figure 9: Prediction of the hybrid neural ODE (shown by the solid curves) against the ground truth data (shown by the scatter points), both before (Figure 9a) and after (Figure 9b) the training process. The ground truth data for the state variables x and y are faint as this data is not available during training.

Figure 9 shows that the hybrid neural ODE is able to accurately capture the dynamics of the system within the training range, for both the observed and unobserved states. To investigate the model's ability to generalise beyond the training data, an extrapolation up to 6 units of time is made. However, the short training range means that even with the physical knowledge available through the $\frac{dz}{dt}$, $\frac{dv}{dt}$ and $\frac{dw}{dt}$ equations, there is insufficient information available for the network to produce accurate extrapolations. This, together with chaotic nature of the Lorenz attractor, results in an unstable prediction and is hence not shown. It is for this reason that the training range of 0.25 units of time is omitted from the comparative analysis of the effects of varying the length of the training range on the accuracy of the extrapolations (0.4, 0.8 and 1.2 units of time are used for this comparison). Figure 10 shows the results of this analysis. Increasing the training range improves the accuracy of the extrapolation, however the chaotic nature of the system ensures a divergence from the ground truth relatively quickly, even in the case of the longest training range. As a result, the errors for each training range converge towards the end of the simulation.



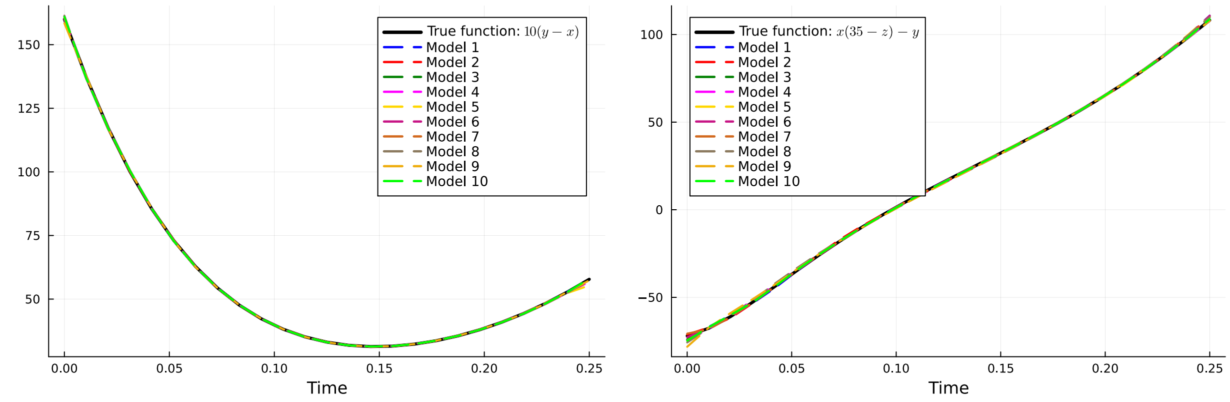
(a) 0% noise



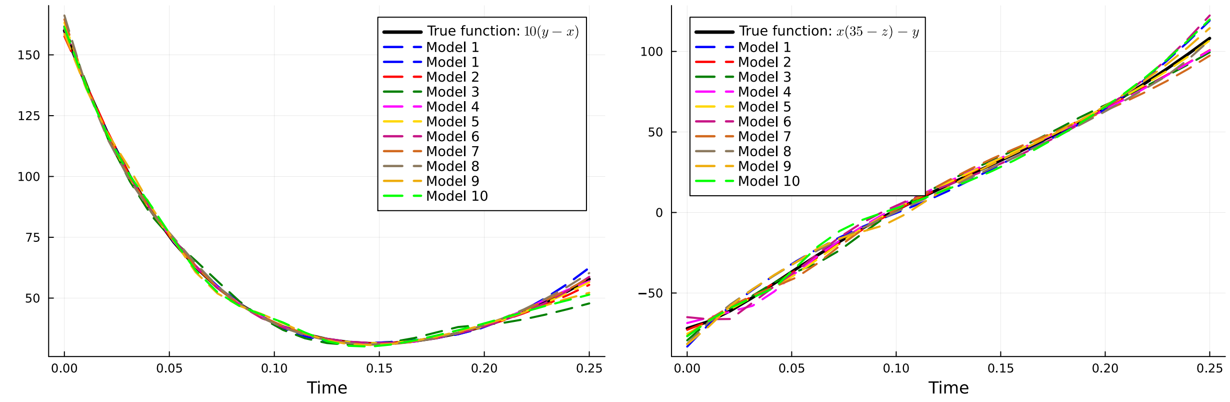
(b) 0.3% noise

Figure 10: Sliding window errors of hybrid neural ODE extrapolations, using three different training ranges. The training ranges of 0.4, 0.8 and 1.2 units of time are abbreviated to TR1, TR2 and TR3, respectively. Only the unobserved states are shown, with x and y being represented by the crosses and circles, respectively. This comparison is done at each level of noise.

The issue of a relatively quick divergence from the ground truth can be ameliorated if the correct equations are discovered by SR. Returning to the case of a training range of 0.25 units of time, the ten hybrid neural ODEs in the ensemble are all simulated to generate the temporal evolution of the states $[x, y, z, v, w]$, which are then passed as input to each of the ten neural networks. The outputs of each network then correspond to the learned dynamics of the $\frac{dx}{dt}$ and $\frac{dy}{dt}$ equations. This is shown in Figure 11.



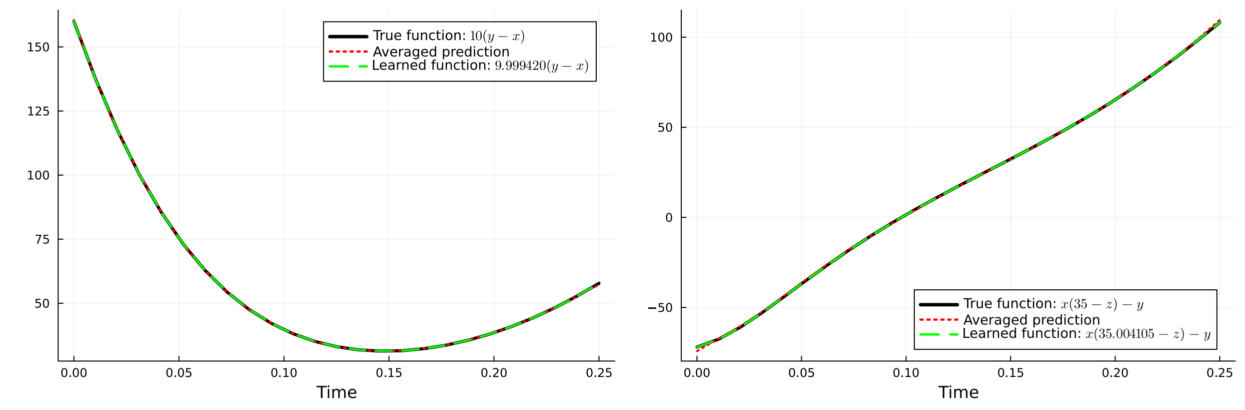
(a) 0% noise



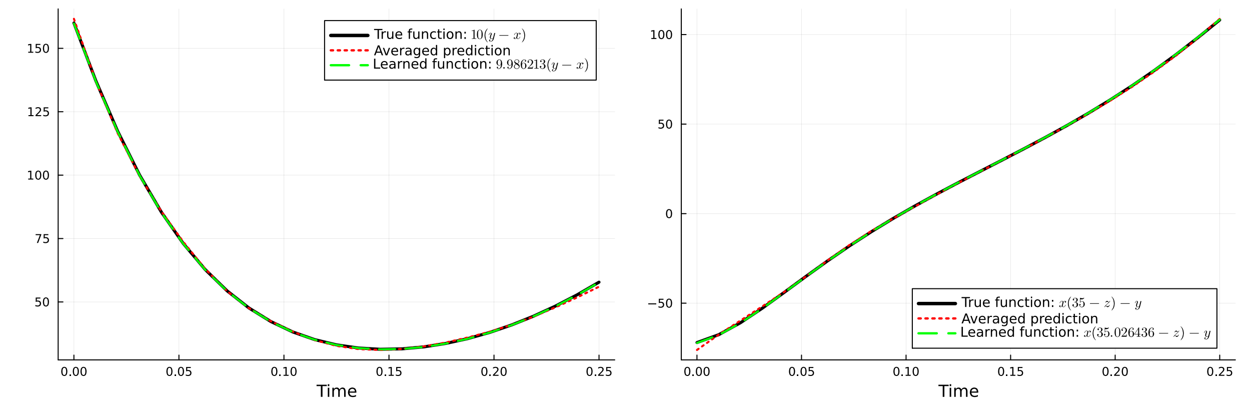
(b) 0.3% noise

Figure 11: The predicted dynamics of the $\frac{dx}{dt}$ and $\frac{dy}{dt}$ equations generated by each network in the ensemble, at each level of noise added to the ground truth data.

Figure 11a shows little variability among the predictions of the networks in the ensemble for both the $\frac{dx}{dt}$ and $\frac{dy}{dt}$ equations. This variability increases significantly with a small amount of measurement noise added to the training data, as in Figure 11b. SR is then applied to the average of these predictions. This is displayed in Figure 12, which shows the averaged prediction (red dotted curve) along with the corresponding learned function through SR (green dashed curve).



(a) 0% noise



(b) 0.3% noise

Figure 12: The averaged predicted dynamics of the $\frac{dx}{dt}$ and $\frac{dy}{dt}$ equations generated by each of the networks in the ensemble (red) and the corresponding learned function (green), at each level of noise added to the ground truth data.

Figure 12 highlights the benefit of training an ensemble of models and averaging the network predictions, as the errors cancel out, producing a curve which more accurately approximates the dynamics of the true equations 1.4 and 1.5. Despite small deviations of the averaged prediction from the true dynamics, SR is able to recover true equation structure. The learned equations are given in Table 2.

Noise	Learned Equations	True Equations
0%	$\dot{x} = 9.999420(y-x)$ $\dot{y} = x(35.004105-z)-y$	$\dot{x} = 10(y-x)$ $\dot{y} = x(35-z)-y$
0.3%	$\dot{x} = 9.986213(y-x)$ $\dot{y} = x(35.026436-z)-y$	

Table 2: Learned equations for $\frac{dx}{dt}$ and $\frac{dy}{dt}$ at each level of noise added to the ground truth data. The target data for SR was the averaged prediction of the ensemble (red curves in Figure 12).

The learned equations in Table 2 are of the correct form, with the only differences from the true equations being the coefficient of the y term for the $\frac{dx}{dt}$ equation and the coefficient of the x term in the $\frac{dy}{dt}$ equation. The learned equations in the absence of noise are then substituted back into the model, resulting in a partially-learned system, where equations 1.4 and 1.5 are set to be $\frac{dx}{dt} = 9.999420(y-x)$ and $\frac{dy}{dt} = x(35.004105-z)-y$, respectively.

To assess this partially-learned model's ability to generalise beyond the training data, an extrapolation up to 6 units of time is made, generating the temporal evolution shown in Figure 13.

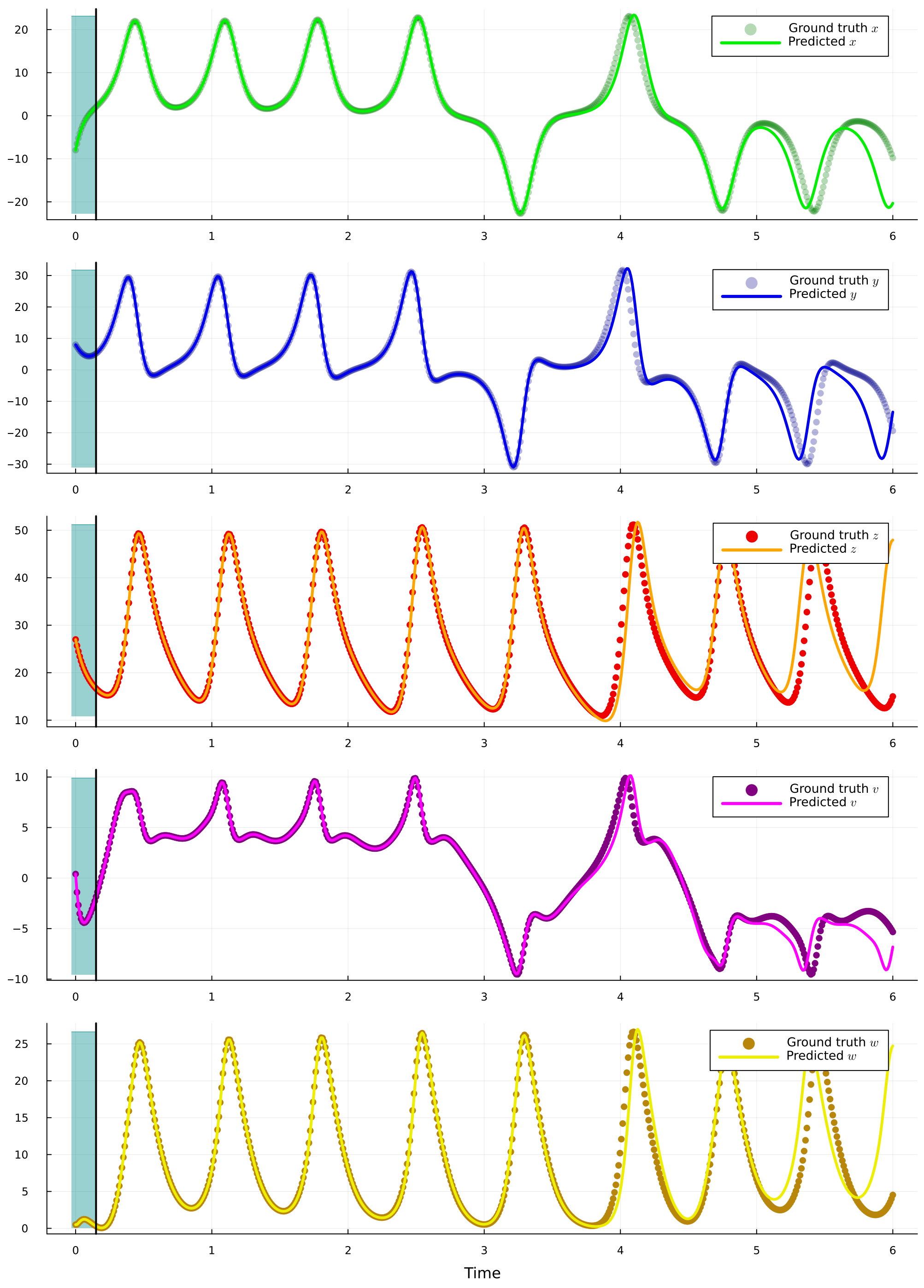


Figure 13: Extrapolation of the partially-learned model with the equations discovered in the absence of noise. Here, $\dot{x} = 9.999420(y - x)$ and $\dot{y} = x(35.004105 - z) - y$. The predictions are shown by the solid curves and the ground truth is shown by the scatter points. The ground truth for the x and y state variables are faint as this data are assumed to be unavailable. The blue region to the left of each plot represents the training range.

The predictions of the partially-learned model are accurate up to approximately 5 units of time. The deviation of the predictions from the ground truth beyond 5 units of time is expected given the chaotic behaviour of the Lorenz system in that small perturbations to the parameters of the system have a significant effect.

4. Discussion and Conclusion

In this work, we demonstrate that it is possible to recover the underlying governing equations of unobserved states in dynamical systems prescribed by ODEs. The ensemble technique of training multiple hybrid neural ODEs and taking the averaged output of the networks as the target data for SR proved to be an effective means of ensuring that the true governing equations were recovered in the presence of measurement noise.

If data are available on the function to be learned, then SR can be applied directly to the data. However, the cases where data are unavailable motivates the use of the neural network, where the missing dynamics can first be learned by training on the available data, before applying SR.

In the PySR implementation, a list of the most suitable candidate functions is generated. Generally, the most complex function (often with many nested terms) has the lowest error. However, this can be an over-fit and can lack interpretability. Therefore, PySR assigns each function a 'score', based on a trade-off between accuracy and complexity [13]. In all experiments conducted in this work, the learned equations presented were the ones with the highest score from the list of generated functions.

For both the Lotka-Volterra and Lorenz systems addressed in this work, the training range was selected to be relatively short, capturing only a small region of the dynamics. While this can often highlight the advantage of incorporating physical knowledge within the model, in that less data is needed to train the embedded neural network, extrapolations of the trained model can be poor if the range of training data is too short. This was the case for both examples in this work. Despite this, a short training range was used to allow the neural network to approximate the missing dynamics more closely, in order to increase the likelihood of learning the true underlying equations via SR.

For the hybrid neural ODEs that were trained on noisy data, the extrapolations of the corresponding partially-learned models are shown in Appendix Appendix A. For the Lotka-Volterra system, the partially-learned models constructed from 2% and 5% measurement noise generate predictions (Figures A.14 and A.15, respectively) very similar to the partially-learned model constructed in the absence of noise (Figure 8). This is because the dynamics of the system are periodic and the learned $\frac{dy}{dt}$ equations at each level of noise (Table 1) are all similar. For this system, the method is generally not robust to 10% measurement noise. For the 5D Lorenz system, the learned $\frac{dx}{dt}$ and $\frac{dy}{dt}$ equations in the presence of 0.3% noise (Table 2) are still accurate, but the slight additional error causes a noticeable difference in the extrapolation of the partially-learned model (Figure A.16), where the predictions are now only accurate up to 3.5 units of time. This is again due to the chaotic nature of the Lorenz system, in that small changes to parameters of the system have large effects at a later stage.

Appendix Appendix C details a slightly alternative method, whereby instead of applying SR to the averaged prediction of networks in the ensemble, SR is applied to each network prediction and an average of the learned models is taken. This highlights the effects of the added measurement noise, since more noise generally causes increased variability among the predictions of the networks in the ensemble, which in turn results in increased variability among the learned functions via SR. This approach shows that for the learned terms spanning all learned functions from the ensemble, the true coefficient values generally lie within 1 standard deviation of the mean learned coefficient values.

A possible avenue of future work is to use a Bayesian neural network within the hybrid neural ODE. This way, the training process would only be carried out once (instead of training an ensemble of 10 models) and multiple predictions of the hybrid model could be generated by sampling from the posterior distribution of the network parameters. As a result of this, SR can be carried out for each sample, and a distribution of partially-learned models can also be generated.

CRediT authorship contribution statement

Gevik Grigorian: Conceptualization, Formal analysis, Investigation, Methodology, Software, Validation, Visualization, Writing - original draft, Writing - review & editing. **Sandip V. George:** Conceptualization, Formal analysis, Investigation, Methodology, Software, Supervision, Validation, Visualization, Writing - original draft, Writing - review & editing. **Simon Arridge:1** Conceptualization, Methodology, Software, Supervision, Validation, Visualization, Writing - original draft, Writing - review & editing.

Data availability

No data was used for the research described in this article.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work was undertaken as part of the EPSRC-Funded CHIMERA (**C**ollaborative **H**ealthcare **I**nnovation through **M**athematics, **E**nginee**R**ing and **A**I) Mathematical Sciences in Healthcare Hub (grant no. EP/T017791/1), which aims to improve outcomes of critically unwell intensive care unit (ICU) patients through combinations of mechanistic modelling, statistical and machine learning.

Appendix A. Appendix: Learned Extrapolations

The extrapolations of the learned models which were trained on noisy data are given here.

Appendix A.1. 3D Lotka-Volterra System

Figure A.14 shows the extrapolation of the model including the equations learned in the presence of 2% noise, from Table 1.

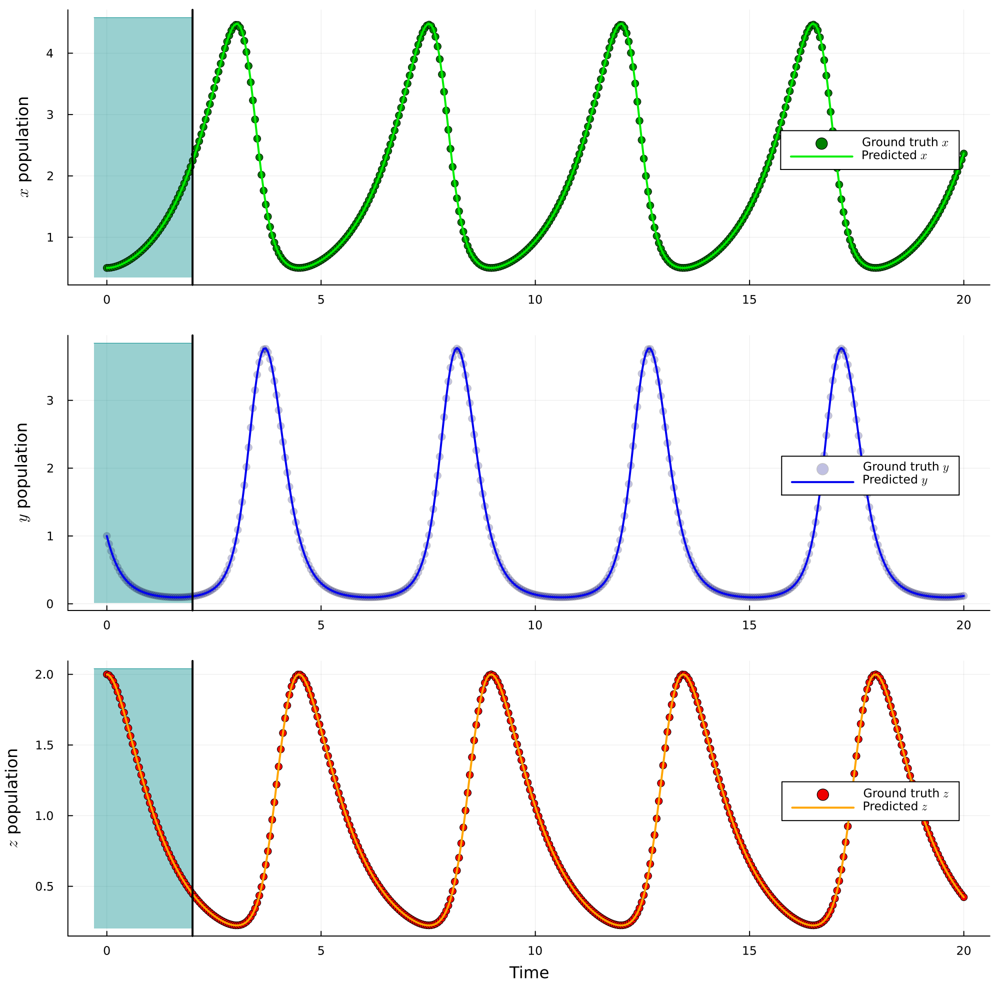


Figure A.14: Extrapolation of the partially-learned model with the equations discovered in the presence of 2% noise. Here, $\dot{y} = -1.0049027y + xy - yz$. The predictions are shown by the solid curves and the ground truth is shown by the scatter points. The ground truth for the state variable y is faint as this data is assumed to be unavailable. The blue region to the left of each plot represents the training range.

Figure A.15 shows the extrapolation of the model including the equations learned in the presence of 5% noise, from Table 1.

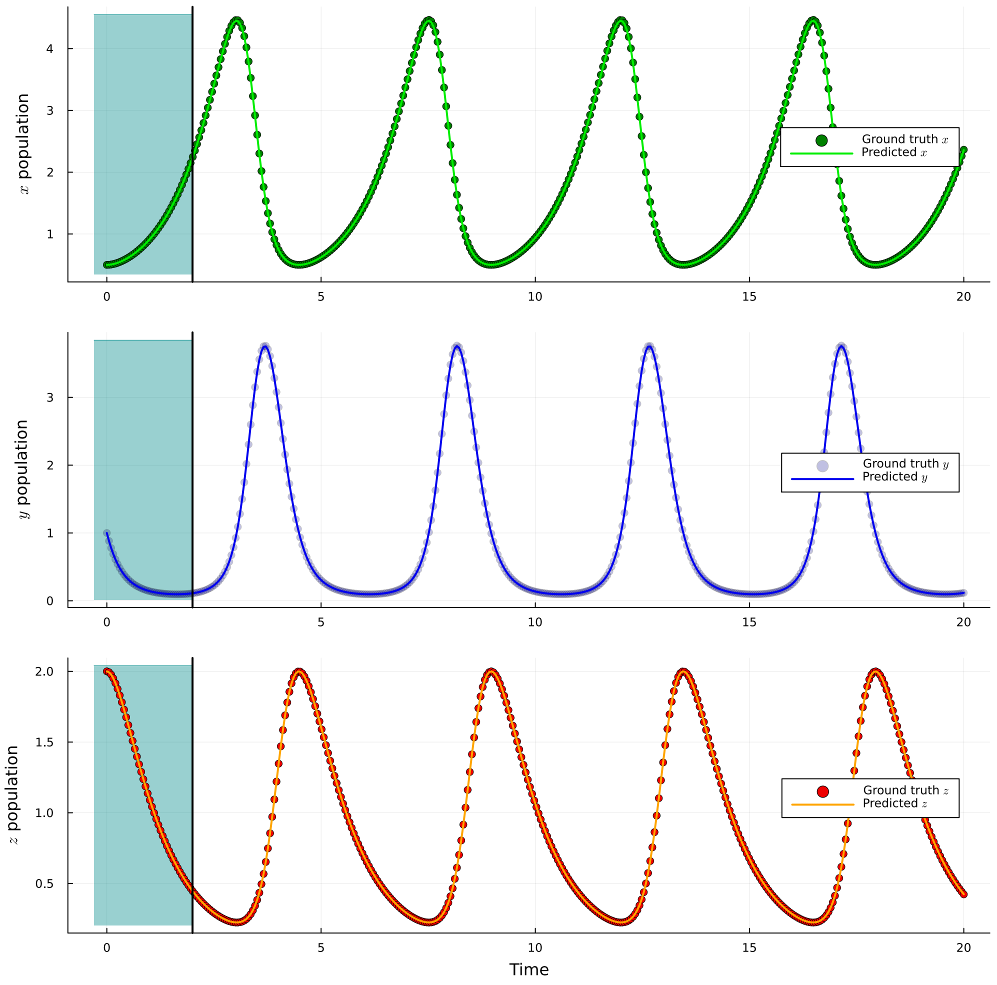


Figure A.15: Extrapolation of the partially-learned model with the equations discovered in the presence of 5% noise. Here, $\dot{y} = -0.9952158y + xy - yz$. The predictions are shown by the solid curves and the ground truth is shown by the scatter points. The ground truth for the state variable y is faint as this data is assumed to be unavailable. The blue region to the left of each plot represents the training range.

Appendix A.2. 5D Lorenz System

Figure A.16 shows the extrapolation of the model including the equations learned in the presence of 0.3% noise, from Table 2.

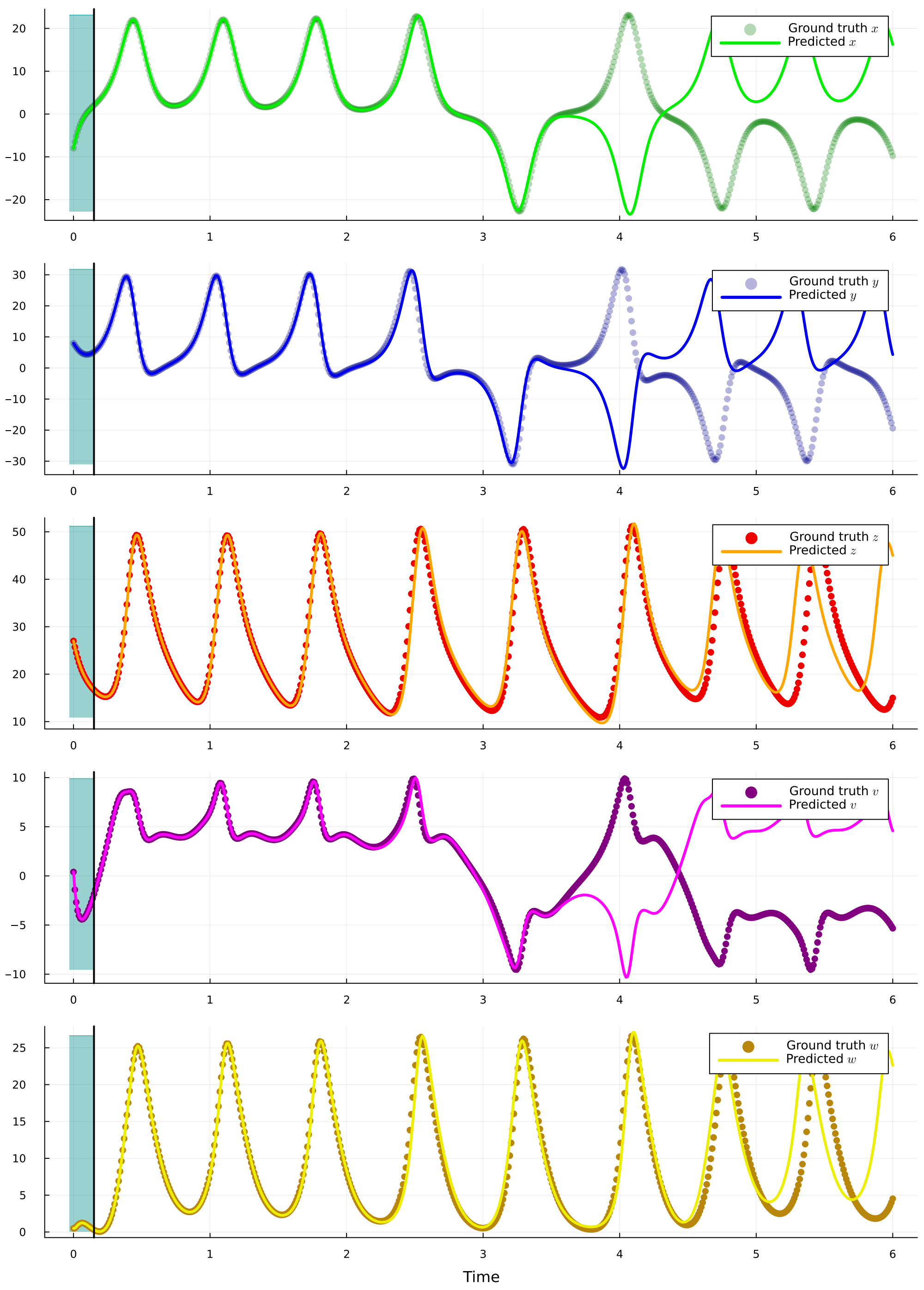


Figure A.16: Extrapolation of the partially-learned model with the equations discovered in the presence of 0.3% noise. Here, $\dot{x} = 9.986213(y - x)$ and $\dot{y} = x(35.026436 - z) - y$. The predictions are shown by the solid curves and the ground truth is shown by the scatter points. The ground truth for the x and y state variables are faint as this data are assumed to be unavailable. The blue region to the left of each plot represents the training range.

Appendix B. Appendix: PySR Details

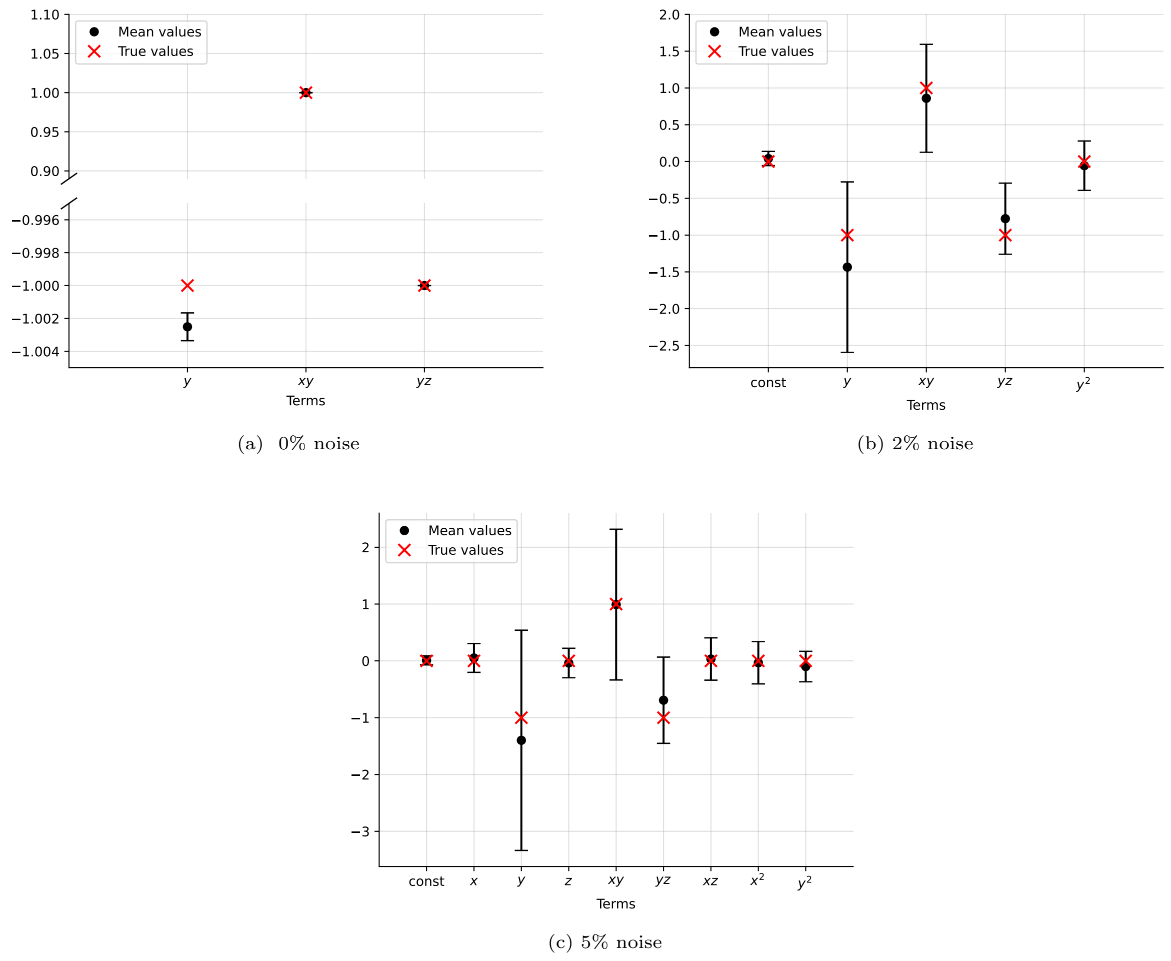
The user defined hyper-parameters for the PySR implementation are given in Table B.3. The process begins with 1000 sub-populations containing 33 randomly initialised trees each. Via the processes mentioned in section 2.3.2, new trees are iteratively generated. MSE is selected as the fitness metric to evaluate how well a candidate function fits the target data. Though we select 200 iterations as the stopping criteria for the algorithm, alternative stopping criteria such as an amount of time elapsed or a desired fitness reached, can be used.

Unary operators	$\{e\}$
Binary operators	$\{+, -, \div, \times\}$
Functions per population	33
Populations	1000
Iterations	200
Performance metric	MSE

Table B.3: Hyper-parameters used in the PySR implementation.

Appendix C. Appendix: Alternative Method

For this approach, the same case studies are examined, but SR is applied to each model in the ensemble, i.e. prior to averaging. The mean and standard deviations of the coefficients of the terms that appear in any of the learned models are then taken. Here, the ensembles consist of 20 hybrid neural ODEs to obtain more accurate estimates of the means and variances. The results of these analyses are shown in Figures C.17 and C.18, while also being summarised in Tables C.4 and C.5. In each case, the error bars correspond to one standard deviation.



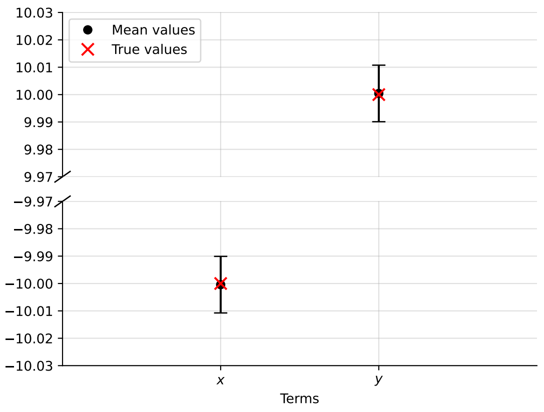
(a) 0% noise (b) 2% noise (c) 5% noise

Figure C.17: Means and standard deviations of the learned coefficients for the Lotka-Volterra hybrid neural ODE.

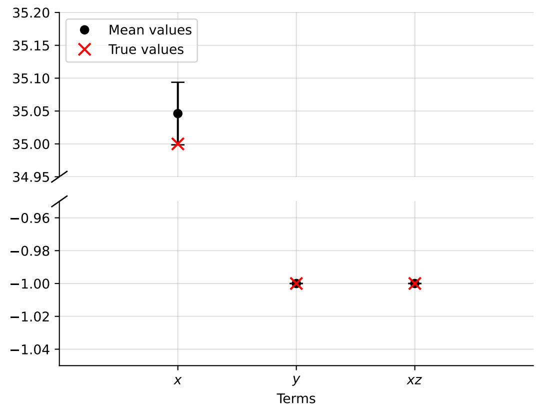
Noise	Mean Learned Equation	True Equation
0%	$\dot{y} = -1.00250887y + xy - yz$	$\dot{y} = -y + xy - yz$
2%	$\dot{y} = 0.0409 - 1.435y + 0.859xy - 0.743yz - 0.056y^2$	
5%	$\dot{y} = -0.0089 + 0.0517x - 1.398y - 0.037z + 0.991xy - 0.692yz + 0.033xz - 0.033x^2 - 0.1y^2$	

Table C.4: Lotka-Volterra: mean coefficients of learned equations across each model in the ensemble, at each level of noise.

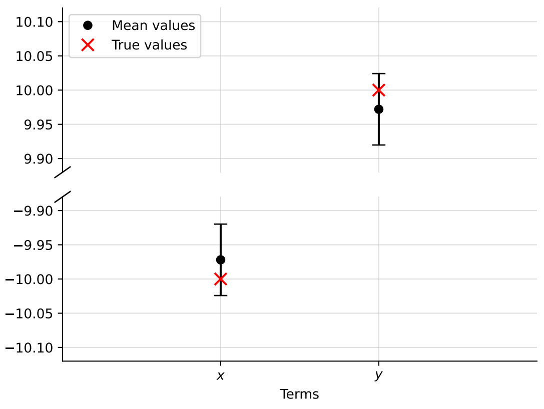
Figure C.17a shows that without measurement noise, the correct equation structure for $\frac{dy}{dt}$ is consistently found, while Figures C.17b and C.17c show that increasing measurement noise increases the number of incorrect terms learned by SR. The mean coefficient values of these incorrect terms are, however, close to 0. Apart from the y term in Figure C.17a, all the true values of the coefficients lie within 1 standard deviation of the mean learned values. The means of these learned equations are given in Table C.4.



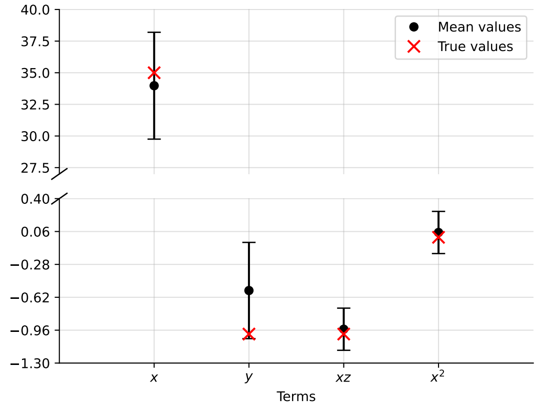
(a) Eq1, 0% noise



(b) Eq2, 0% noise



(c) Eq1, 0.3% noise



(d) Eq2, 0.3% noise

Figure C.18: Means and standard deviations of the learned coefficients for the Lorenz hybrid neural ODE.

Noise	Mean Learned Equations	True Equations
0%	$\dot{x} = 10.00042325(y - x)$ $\dot{y} = x(35.04614365 - z) - y$	$\dot{x} = 10(y - x)$ $\dot{y} = x(35 - z) - y$
0.3%	$\dot{x} = 9.9719798(y - x)$ $\dot{y} = x(33.97495465 - 0.95z) - 0.55y + 0.05x^2$	

Table C.5: Lorenz: mean coefficients of learned equations across each model in the ensemble, at each level of noise

Figures C.18a and C.18c show that the correct form of the $\frac{dx}{dt}$ equation is consistently learned, both with and without noise (and with accurate coefficient values), since no incorrect terms are recovered. Without noise, the $\frac{dy}{dt}$ is also correctly learned consistently (Figure C.18b), whereas in the presence of noise, an incorrect xz term is also recovered. Again, the true coefficient values all lie within 1 standard deviation of the mean learned values. The means of these learned equations are given Table C.5.

References

[1] B. Prokop, L. Gelens, From biological data to oscillator models using sindy, iScience (2024).

[2] F. Sorourifar, Y. Peng, I. Castillo, L. Bui, J. Venegas, J. A. Paulson, Physics-enhanced neural ordinary differential equations: application to industrial chemical reaction systems, Industrial & Engineering Chemistry Research 62 (2023) 15563–15577.

[3] F. A. R. Lima, C. M. Rebello, E. A. Costa, V. V. Santana, M. G. de Moares, A. G. Barreto Jr, A. R. Secchi, M. B. de Souza Jr, I. B. Nogueira, Improved modeling of crystallization processes by universal differential equations, Chemical Engineering Research and Design 200 (2023) 538–549.

[4] C. T. Mackay, D. Nowell, Informed machine learning methods for application in engineering: A review, Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science (2023) 09544062231164575.

[5] C. Shen, A. P. Appling, P. Gentine, T. Bandai, H. Gupta, A. Tartakovsky, M. Baity-Jesi, F. Fenicia, D. Kifer, L. Li, et al., Differentiable modelling to unify machine learning and physical models for geosciences, Nature Reviews Earth & Environment 4 (2023) 552–567.

[6] S. L. Brunton, J. L. Proctor, J. N. Kutz, Discovering governing equations from data by sparse identification of nonlinear dynamical systems, Proceedings of the national academy of sciences 113 (2016) 3932–3937.

[7] C. Rackauckas, Y. Ma, J. Martensen, C. Warner, K. Zubov, R. Supekar, D. Skinner, A. Ramadhan, A. Edelman, Universal differential equations for scientific machine learning, arXiv preprint arXiv:2001.04385 (2020).

[8] J. Bongard, H. Lipson, Automated reverse engineering of nonlinear dynamical systems, Proceedings of the National Academy of Sciences 104 (2007) 9943–9948.

[9] F. Takens, Detecting strange attractors in turbulence, in: Dynamical Systems and Turbulence, Warwick 1980: proceedings of a symposium held at the University of Warwick 1979/80, Springer, 2006, pp. 366–381.

[10] A. Somacal, Y. Barrera, L. Boechi, M. Jonckheere, V. Lefieux, D. Picard, E. Smucler, Uncovering differential equations from data with hidden variables, Physical Review E 105 (2022) 054209.

[11] J. Bakarji, K. Champion, J. Nathan Kutz, S. L. Brunton, Discovering governing equations from partial measurements with deep delay autoencoders, Proceedings of the Royal Society A 479 (2023) 20230422.

[12] J. Bezanson, A. Edelman, S. Karpinski, V. B. Shah, Julia: A fresh approach to numerical computing, SIAM review 59 (2017) 65–98.

[13] M. Cranmer, Interpretable machine learning for science with PySR and symbolic regression. jl, arXiv preprint arXiv:2305.01582 (2023).

[14] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980 (2014).

[15] D. C. Liu, J. Nocedal, On the limited memory BFGS method for large scale optimization, Mathematical programming 45 (1989) 503–528.

[16] D. A. Augusto, H. J. Barbosa, Symbolic regression via genetic programming, in: Proceedings. Vol. 1. Sixth Brazilian symposium on neural networks, IEEE, 2000, pp. 173–178.

[17] Y. Wang, N. Wagner, J. M. Rondinelli, Symbolic regression in materials science, MRS Communications 9 (2019) 793–805.