

000
001
002
003
004
005
006
007
008
009
010
011
012
013
014
015
016
017
018
019
020
021
022
023
024
025
026
027
028
029
030
031
032
033
034
035
036
037
038
039
040
041
042
043
044
045
046
047
048
049
050
051
052
053

Supplementary Material

Deep Generative Modeling for Identification of Noisy, Non-Stationary Dynamical Systems

1 METHODS

1.1 VARIATIONAL AUTOENCODERS

In this section, we elaborate on the mathematical foundation of the Variational Autoencoder (VAE) architecture (19; 6).

Like standard autoencoders, VAEs have an encoder and decoder network to process input data and generate output. However, instead of mapping inputs to fixed points in the latent space, the encoder maps them to a probability distribution. The decoder then samples from this distribution to reconstruct the input. This probabilistic framework reduces overfitting by introducing variability into the latent space. After computing the reconstruction error, the network is trained via backpropagation, with the VAE relying on the reparameterization trick to ensure gradients can be propagated through the network.

Mathematically, we aim to approximate the data distribution $p^*(X)$ of some given observations X . When direct computation is intractable, we introduce a latent variable z such that $p^*(x)$ is decomposed as:

$$p^*(x) = \int_z p^*(x|z)p^*(z)dz \quad (1)$$

where $p^*(x|z)$ is the likelihood and $p^*(z)$ is a prior, often set to a standard normal distribution. Since this integral is difficult to compute, we approximate $p_\theta(x|z)$ with a neural network parameterized by θ . To estimate the posterior distribution $p^*(z|x)$, we approximate it with another neural network $q_\phi(z|x)$, parameterized by ϕ . This is the core idea of variational inference: complex distributions are approximated by simpler, parametrized ones through optimization. We arrive at the following objective:

$$\begin{aligned} \log p_\theta(x) &= \log \int_z p_\theta(x, z) dz \\ &= \log \int_z p_\theta(x, z) \frac{q_\phi(z|x)}{q_\phi(z|x)} dz \\ &= \log E_{z \sim q_\phi(z|x)} \left[\frac{p_\theta(x, z)}{q_\phi(z|x)} \right] \geq E_z \left[\frac{\log p_\theta(x, z)}{q_\phi(z|x)} \right] \end{aligned} \quad (2)$$

by Jensen's inequality. This leads to the evidence lower bound (ELBO):

$$\mathcal{L} = E_z \frac{\log p_\theta(x, z)}{q_\phi(z|x)} \quad (3)$$

The ELBO sets a lower bound for the evidence of observations and maximizing \mathcal{L} will increase the log-likelihood of X . To find the parameters θ, ϕ so as to maximize the ELBO, it is convenient to re-write \mathcal{L} in the following way:

$$\begin{aligned}\mathcal{L} &= \int_z q_\phi(z|x) \log\left(\frac{p_\theta(x, z)}{q_\phi(z|x)}\right) \\ &= \int_z q_\phi(z|x) \log\left(\frac{p_\theta(x|z)p(z)}{q_\phi(z|x)}\right) = E_{z \sim q_\phi(z|x)} \log(p_\theta(x|z)) - D_{KL}(q_\phi(z|x)||p(z))\end{aligned}\quad (4)$$

where D_{KL} is the Kullback-Leibler divergence between the approximate posterior $q_\phi(z|x)$ and the prior $p(z)$. The ELBO comprises two terms: the expected log-likelihood of the data, and a regularization term that enforces similarity between the posterior and the prior.

In the autoencoder perspective, the encoder network maps inputs to the latent space via q_ϕ , and the decoder maps the latent variables back to the input space via p_θ . Both networks are trained jointly using stochastic gradient descent to optimize the ELBO. More details on the VAE framework can be found in Kingma et al.'s excellent review (20).

1.2 DYNAMIC VAEs

Generating time series data presents unique challenges due to the intricate temporal relationships and the distribution of features at each time point. One common approach is using generative adversarial networks (GANs), which often incorporate recurrent neural networks (RNNs) for both generation and discrimination. However, despite numerous proposed architectures, GANs have struggled to capture the complex temporal dependencies inherent in time series data. Yoon et al. (11) introduced a novel approach that blends the supervised training used in autoregressive models with the unsupervised training of GANs. While we experimented with this method for generating time series, the training proved to be time-consuming and impractical for our datasets (see Section 3.1). Further limitations are discussed by Desai et al. (5).

As a result, we shifted our focus to methods based on Variational Autoencoders (VAEs) for time series, leveraging deep learning techniques to model complex temporal patterns more effectively.

An extensive review (13) offers a unified framework for several VAE models extended to handle temporal and sequential data. These models, collectively referred to as dynamic VAEs (DVAEs), share common notation, methodology, and a standardized mathematical formalism. The review covers various approaches, including Deep Kalman Filters (22; 23), Kalman Variational Autoencoders (7), Stochastic Recurrent Networks (9), Variational Recurrent Neural Networks (10; 1), Stochastic Recurrent Neural Networks (15), Recurrent Variational Autoencoders (25), and Disentangled Sequential Autoencoders (37). In the following section, we will expand on the mathematical framework common to these methods, as outlined in (13).

Briefly, given a time-series $X_{1:T}$, and assuming latent variables $Z_{1:T}$, the goal is to specify the joint distribution of the observed and latent sequential data $p_\theta(X_{1:T}, Z_{1:T})$, where θ denotes the parameters of the true distribution's probabilistic model. DVAEs are hierarchical models in which both observed and latent variables are treated as time-ordered vectors. These models are often causal, meaning the distribution of variables at time t depends only on previous time steps. This causality imposes the following factorization:

$$p(X_{1:t}, Z_{1:t}) = \prod_{t=1}^T p(x_t, z_t | x_{1:t-1}, z_{1:t-1}) = \prod_{t=1}^T p(x_t | x_{1:t-1}, z_{1:t}) p(z_t | x_{1:t-1}, z_{1:t-1}) \quad (5)$$

The joint distribution of observed and latent variable sequences can be factorized using the chain rule. Crucially, different models proposed in the literature make different conditional assumptions to simplify the dependencies in the conditional distribution. For example, a simple model may make the following simplifications: $p(X_t | X_{1:t-1}, Z_{1:t}) = p(X_t | Z_t)$ and $p(Z_t | X_{1:t-1}, Z_{1:t-1}) = p(Z_t | Z_{t-1})$. In addition, different models may implement different network architectures to approximate p_θ and q_ϕ . A detailed account of the kind of assumptions that each model implements to simplify (5) can be found in (13).

1.2.1 TIMEVAE

TimeVAE is a variational autoencoder designed to generate multivariate time-series data (5). It extends the standard VAE framework to model both the latent space and the temporal dependencies of a sequence of data vectors. Supplementary Figure 1A illustrates the basic TimeVAE architecture, which uses dense and convolutional layers without requiring specific time-series knowledge. The decoder allows for customizable distributions by adding layers to capture time-series components like level, trends, and seasonality, though we used the base version that excludes these custom structures in our experiments.

The input to the encoder is a 3D array of size $N \times D \times T$, where N is the batch size, D is the number of feature dimensions, and T is the number of time steps. The encoder processes the data through convolutional layers with ReLU activations, flattens the output, and then applies a fully connected layer. The final encoder layer has $2d$ units, representing the mean and variance of a multivariate Gaussian distribution, where d is the dimensionality of the latent space, a key hyperparameter. The reparameterization trick is used to sample from the Gaussian distribution, parameterized by the encoder’s output.

The decoder reconstructs the data by first passing the sampled latent vector z through a fully connected layer, reshaping it into a 3D array, and processing it through a series of transposed convolutional layers with ReLU activation. The last time-distributed fully connected layer produces the final output that matches the original input dimensions.

Training TimeVAE involves optimizing the ELBO loss function (discussed in Section 1.1) with different weights on the reconstruction error and KL divergence between the approximate posterior $q_\phi(z|x)$ and the prior $p_\theta(z)$. Hyperparameters are tuned to determine the appropriate balance between reconstruction loss, KL divergence, and additional regularization terms (e.g., sparsity and total variation, for our problem set-up).

TimeVAE has been tested on four multivariate datasets (5): (1) a 5-dimensional sinusoidal dataset with varying frequencies, amplitudes, and phases; (2) a 6-dimensional stock market dataset from Yahoo Finance; (3) a 28-dimensional appliances energy prediction dataset from the UCI Machine Learning Repository; and (4) a dataset with 15 features of hourly air quality sensor readings from the UCI Machine Learning Repository. The results show that TimeVAE performs comparably to top generative models across various metrics, is computationally efficient, and outperforms existing methods in next-step prediction tasks, particularly when training data is limited (5).

1.2.2 DYNAMIC HYPERSINDY

Inspired by previous work (14; 16), we developed a hierarchical architecture to address non-autonomous systems, illustrated in Supplementary Figure 1B. The main text focuses on the dynamic SINDy framework with a timeVAE architecture, while Supplementary Figures 3, 4, 5 show that incorporating dynamic HyperSINDy results in coefficients and trajectories that closely match the ground truth.

The first level consists of a standard VAE with encoder and decoder modules (SM Sec. 1.1). The decoder generates X with a probability distribution $p_t(X)$ at each time step t . The next level introduces a hypernetwork, implemented as either a long short-term memory network (LSTM) or multi-layer perceptron (MLP), which updates the decoder’s weights to adjust the probability distribution for the following time step, allowing the system to capture temporal drift in the output:

$$W_{\text{Decoder}}(t+1) = W_{\text{Decoder}}(t) + \sum_{i=1}^M \alpha_i(t) \cdot D_i, \text{ where}$$

$$\text{LSTM}(t) = \alpha(t) = [\alpha_1(t), \dots, \alpha_n(t)] \quad (6)$$

Here, D_i are fixed basis tensors to be learned, and α_i are hypernetwork outputs. This architecture adapts to changing dynamics, adjusting the decoder based on reconstruction error and updating the output probability distribution.

We modified this architecture for our problem. Instead of a VAE generating data, our decoder produces SINDy coefficients, which, when combined with the SINDy library, replicate system dynamics. The decoder approximates the true pdf of the SINDy coefficients. This setup builds on (16) by adding a hypernetwork that updates decoder weights, forming what we call dynamic HyperSINDy. This extension allows for time-varying SINDy coefficients, processed sequentially rather than requiring the entire time series as input (as in timeVAE).

Two primary training methods are used for dynamic HyperSINDy:

- **Online Learning:** Ideal for switching systems where the network adapts as dynamics change. However, network parameters evolve, requiring tracking of parameter changes and identifying switch points after training. The hypernetwork is not needed in this setup.
- **Alternate learning:** The hypernetwork is trained first with fixed main module parameters and basis tensors D_i , followed by adjustment of the main module parameters/ D_i , while fixing the hypernetwork. This method is best for continuously varying SINDy coefficients, with LSTM as the preferred hypernetwork.

Training used the hyperparameters listed in Table 1. We processed one trajectory at a time (trial batch size of 1) and used batch sizes of 1-10 time steps. The latent dimension of the VAE was set to 25 while the starting threshold was 0.1. Every 50 epochs, we evaluated and set to zero any SINDy coefficients with a mean absolute value below a threshold. A relaxed L0 norm in the loss function encouraged sparsity in the SINDy coefficients, following (16; 4).

Several hyperparameters were increased progressively during training. The threshold rose by 0.005 every 50 epochs until it plateaued, alongside the weight λ_{kl} for the KL divergence term, which increased until it reached a maximum value λ_{max} . The threshold plateaus as well once λ_{max} is reached. We fixed the number of basis tensors D_i to 10, which combined linearly with hypernetwork outputs to form decoder weights via Eq. (6).

The encoder consisted of four fully connected layers with hidden dimensions of 64, using ELU activation and an input dimension twice that of X , as it takes X and \dot{X} as input. The decoder also had four hidden layers, with a hidden dimension of 256 and ELU activation. The hypernetwork, either an LSTM or MLP, contained two layers with an input dimension of 25. We trained using the AdamW optimizer with an initial learning rate of 0.001, weight decay of $1e - 5$, gradient clipping at 1, and Amsgrad. Additionally, an exponential learning rate scheduler with $\gamma = 0.999$ was used. Many hyperparameters match those in (16).

1.3 TRAINING DYNAMIC SINDY WITH TIMEVAE: METHODOLOGY AND HYPERPARAMETERS

Training timeVAE requires normalizing the data beforehand. While (5) normalizes by subtracting the minimum and dividing by the maximum to scale the data between 0 and 1, we normalize by dividing only by the maximum value. This normalization method affects the SINDy coefficients produced by our method, so we re-scale the resulting time-series before comparing them to the ground truth in synthetic datasets.

The loss function used to train our timeVAE architecture is:

$$\text{loss} = \lambda_{MSE} \cdot \|\hat{\dot{X}} - \dot{X}\|_2^2 + \lambda_{KL} \cdot \text{KL div} + \lambda_{sp} \cdot \langle \|\xi_{i,j}(t)\|_1 >_{i,j} \quad (7)$$

$$+ \lambda_{tv} \cdot \frac{\langle \|\xi_{i,j}(t+1) - \xi_{i,j}(t)\|_1 >_{i,j,t}}{\langle \|\xi_{i,j}(t)\|_1 >_{i,j,t} + \epsilon} \quad (8)$$

where ϵ is the machine precision limit. The hyperparameters λ_{MSE} , λ_{KL} , λ_{sp} , and λ_{tv} balance accuracy and complexity by adjusting the weights on the different loss terms: λ_{MSE} controls the mean squared error, while the others handle regularization.

- The first term represents the mean squared error between the inferred derivative $\hat{\dot{X}}$ using dynamic SINDy and the derivative from the data \dot{X} . For all synthetic datasets, the ground truth derivative is the one used to obtain the trajectories X .

Table 1: Hyperparameters for Dynamic HyperSINDy

hyperparameter	value
batch size (trials)	1
batch size (time steps)	1-10
latent variable dimension	25
threshold	0.1
threshold interval	50
threshold increment	0.005
λ_{kl}	0.01
λ_{kl} increment	$\lambda_{kl}/5$
λ_{kl} max	1
M (number of basis tensors)	10
hidden dim (decoder)	256
hidden dim (encoder)	64
input dim (LSTM)	25
gradient clip	1.0
cell dimension (LSTM).	30
optimizer	AdamW
weight decay	1e-5
amsgrad	True
learning rate	0.001
learning rate scheduler	ExponentialLR
gamma	0.999

- The second term is the Kullback-Leibler divergence (*KL div*), a standard term in variational autoencoders (discussed in SM Sec. 1.1). It measures how closely the posterior distribution of z , as computed by the encoder given X , resembles the standard normal distribution. The KL divergence has an analytic form:

$$\text{KL div} = -\frac{1}{2} \cdot \langle (1 + 2 \log(\sigma_{z_{i,j}}) - \sqrt{\mu_{z_{i,j}}} - \exp(2 \log \sigma_{z_{i,j}})) \rangle_{i,j} \quad (9)$$

where $\langle \cdot \rangle$ indicates averaging over latent dimensions i and data points j , and μ_{z_j} and σ_{z_j} represent the mean and standard deviation of z_j , with μ_{z_j} and $\log(\sigma_{z_j})$ as the encoder outputs for each input X_j .

- The third term in Eq. (8) is a sparsity penalty that encourages some SINDy coefficients to be zero.
- The fourth term is a normalized total variation penalty that prevents drastic changes in the solution over time.

1.3.1 NON-AUTONOMOUS HARMONIC OSCILLATORS

For the non-autonomous harmonic oscillators, we use the hyperparameters in Table 2 to train the timeVAE architecture. These remain constant across datasets, despite differences in the time-varying coefficients $A(t)$ and $B(t)$. However, key hyperparameters like λ_{sp} and λ_{tv} vary depending on the dataset, as shown in Table 3. Training is performed using the ADAM optimizer with a weight decay of $1e5$ and gradient clipping at 1.

1.3.2 LORENZ DYNAMICS

For the results in Sec. 4.3, involving the chaotic system with a time-varying parameter in the Lorenz dynamics, we follow the same steps as before (data normalization/post-processing, loss function, and two-stage training: first for sparsity pattern, then for coefficient recovery), but with different hyperparameters listed in Table 4. These hyperparameters remain constant, regardless of how the Lorenz parameters vary over time.

Table 2: Hyperparameters for timeVAE (non-autonomous harmonic oscillator)

hyperparameter	value
batch size	1
latent dimension	2
threshold	0.01
library size	3
λ_{MSE}	3
λ_{KL}	1000

Table 3: Hyperparameters for timeVAE at different phases of training (non-autonomous harmonic oscillator)

hyperparameters	dataset	at first training phase	at second training phase
λ_{sp}	A(t) sigmoid	50	0
	A(t) switch signal 1	500	0
	A(t) switch signal 2	200	0
	A(t) finite Fourier series	1	0
λ_{tv}	A(t) sigmoid	100	1000
	A(t) switch signal 1	100	1000
	A(t) switch signal 2	100	1000
	A(t) finite Fourier series 1	0	2

During the first training stage, when the sparsity penalty is non-zero, the batch size is set to 10 to ensure the correct sparsity pattern is learned. We use RMSProp with a weight decay of 10^{-5} and gradient clipping at 10. The threshold gradually increases from 0.05 to 0.1 in increments of 0.025 per epoch, while λ_{sp} rises from 0 to 20 in steps of 1. λ_{tv} is fixed at 1000. This gradual increase in hyperparameters follows a successful approach from a related study (16).

1.3.3 LOTKA VOLTERRA

The incomplete Lotka Volterra system has only one variable x , therefore the library has three terms: x, x^2, x^3 . For training, we use the hyperparameters listed in Tables 5 and 6.

2 LATENT VARIABLE DISCOVERY

2.1 NON-AUTONOMOUS HARMONIC OSCILLATOR

We can use the same approach with the non-autonomous harmonic oscillator as with the Lotka-Volterra system. We set $A(t) = -4$ and vary $B(t)$ sigmoidally such that $B(t) = 2 + \frac{1}{1+\exp(5+t)}$. After dynamic SINDy identifies a trajectory for B , we add it to (x, y) to form a 3D dynamical system. Using SINDy on (x, y, B) , we discover the following ODE which is almost exactly identical to the true dynamics, given that B is a sigmoid that can be described by $\dot{B} = -6 + 5B - B^2$:

$$\begin{aligned}\dot{x} &= -3.997y \\ \dot{y} &= 1 \cdot Bx \\ \dot{B} &= -5.875 + 4.903B - 0.981B^2\end{aligned}\tag{10}$$

Table 4: Hyperparameters for timeVAE (Lorenz dynamics)

hyperparameter	value
latent dimension	5
library size	3
λ_{MSE}	3
λ_{KL}	1000

Table 5: Hyperparameters for timeVAE (Lotka Volterra)

hyperparameter	value
batch size	1
latent dimension	2
library size	3
λ_{MSE}	3
λ_{KL}	1000

Table 6: Hyperparameters for timeVAE at different phases of training (Lotka Volterra)

hyperparameters	at first training phase	at second training phase
λ_{sp}	0.1	0
λ_{tv}	0	0

3 DYNAMIC SINDY FOR SYSTEM IDENTIFICATION OF NEURONAL DYNAMICS IN THE NEMATODE C. ELEGANS

3.1 RESULTS

Like in Morrison et. al., we have discovered a dynamical system model that switches between two stable fixed points. The differential equation model is expressed through a cubic function: $\dot{x} = a \cdot x^3 + b \cdot x^2 + c \cdot x + d \cdot y + u$ with distinct fixed points corresponding to the different switching states of u . More precisely, the differential model inferred has the form:

$$\begin{aligned}
 \dot{x} &= y \\
 \dot{y} &= -0.002 \cdot x^3 + 0.0087x^2 - 0.22 \cdot y + 0.05 \cdot x + u_i, \quad i = 1, 2 \\
 u_0 &\approx -0.266; u_1 \approx 0.044.
 \end{aligned} \tag{11}$$

When $u = u_0 < 0$, the dynamical system has one stable fixed point at -5.25 (the other roots of the cubic equation are complex). This fixed point corresponds to the reversal behavior. Then, when $u = u_1 > 0$, the dynamical system has two stable and one unstable fixed point: -2.32 and 7.88 stable fixed points and -1.19 unstable. Therefore, varying u can generate a bifurcation. In practice, the trajectory shifts between -5.25 and -2.32 in tandem with behavioral switches between reversal and forward states.

The two-dimensional model is a simple model that fits the first principal component and captures stable state clusters and turning trajectory variability. Once the low-dimensional coordinates are identified, dynamic SINDy effectively enables data-driven model discovery. Future work will extend this approach to multiple animals to test its generality across individuals.

Table 7: Hyperparameters for timeVAE (C. elegans data)

hyperparameter	value
batch size	1
latent dimension	2
threshold	0.01
library size	3
λ_{MSE}	3
λ_{KL}	1000

3.2 TRAINING DYNAMIC SINDY ON C. ELEGANS DATA

We apply timeVAE to infer dynamics for a single worm as a proof-of-concept, demonstrating dynamic SINDy’s capability for data-driven discovery. Since there is only one trajectory per worm, uncertainty quantification isn’t possible. As with synthetic datasets, we normalize the trajectories and train timeVAE to infer the differential equation’s sparsity pattern using the hyperparameters from Tables 7 and 8.

With a threshold of 0.01, only the terms 1, x , and x^2 are considered important. We add y and x^3 for comparison with the Morrison et al. model and retrain with a fixed sparsity pattern, omitting sparsity regularization from the loss. Our SINDy coefficients vary over time, matching behavioral transitions between forward and reversal locomotion. Increasing total variance regularization was not effective, so we averaged non-constant SINDy coefficients over time to simplify the model. This is part of the training process, where we take the average of all non-constant coefficients at the last layer of the network to yield the model output and backpropagate. The constant SINDy coefficient is not constrained, but all other coefficients do not change in time.

The resulting differential equation model, detailed in Eq. 12, includes the time series $u(t)$, which is shown in Figure 6C (main text). We interpret $u(t)$ as a switching variable and hypothesize that even a simple switching time series can qualitatively capture the neural activity data. To test this hypothesis, we post-process the $u(t)$ time series to generate a switch-like signal.

3.2.1 POST-PROCESSING THE SWITCHING SIGNAL

Starting with the $u(t)$ time series inferred using dynamic SINDy, we perform the following steps:

- Subtract the mean of $u(t)$ over time. We also note the approximate minimum and maximum values, which will be used later.
- Scale the data by a large factor (1000) and apply a pointwise sigmoid function across time, producing a time series of switches between 0 and 1.
- Finally, re-scale the time series to vary between the previously determined minimum and maximum values, and then add back the mean $\langle u(t) \rangle_t$ to obtain the final post-processed switching time series.

To evaluate the accuracy of our model, we integrate the differential equation from Eq. 12 using the post-processed switching signal $u(t)$ and compare the resulting trajectory to the real trajectory (Figure 6E-G, main text). Since the C. elegans data has low time resolution ($\Delta t = 0.35749752$), we interpolate the data using the CubicSpline function from the scipy.interpolate library. We reduce the time step to $\Delta t/100$ and perform numerical integration using the Euler method.

3.3 BACKGROUND: RELATED STUDIES AND COMPARISONS TO OUR MODEL

3.3.1 COMPARISON WITH STATE SPACE MODELS

Our findings with dynamic SINDy reveal a key similarity with the probabilistic state space model proposed by Linderman et al. (27): both models switch between different dynamical regimes.

Table 8: Hyperparameters for timeVAE at different phases of training (C. elegans data)

hyperparameters	at first training phase	at second training phase
λ_{sp}	10	0
λ_{tv}	100	1000

However, despite being nonlinear, our model is more parsimonious in several ways.

Linderman et al. propose a hierarchical recurrent state space model that switches between simple linear models, using Bayesian inference to fit the model at scale (27). This model decomposes complex nonlinear neural activity into discrete states with simple linear dynamics, which correspond to behaviorally relevant aspects of the worm’s behavior. The transition probabilities depend on both the preceding state and the position in continuous state space, with each discrete state largely tied to the activation of specific neuron clusters.

While this model provides insights into C. elegans neural dynamics, its linear state space models are local. The model switches between eight discrete states, each representing a smaller linear system fitted to the data to explain local dynamics (27). In contrast, dynamic SINDy discovers a global nonlinear ODE model that switches between only two states. Thus, we simplify the model by replacing eight local linear regimes with a more compact nonlinear system switching between two states.

A future direction is to develop a generative model for the switching behavior of $u(t)$, possibly using a probabilistic model or differential equation linked to the variables x and y . This would allow us to eliminate the dynamic SINDy network post-training, retaining a global nonlinear switching differential equation with just four parameters, compared to the many more parameters required by the hierarchical recurrent SLDS for its eight linear systems (even when considering only the continuous variable dynamics for a fair comparison with our approach in PC space).

Moreover, the hierarchical recurrent SLDS is a statistical model that doesn’t directly map onto network dynamics or account for biologically realistic state transitions. While further research is needed to validate our model’s connection to biological measurements of neural activity, nonlinear differential equations like ours are potentially more interpretable. For example, a single parameter change in a global nonlinear model similar to ours can reproduce different long-timescale behaviors observed in C. elegans (17) (see Sec. 3.1.4 below). This modulation mirrors distinct changes in state distribution and switching frequencies seen in experiments, which are linked to specific neuromodulators and neurons (31; 33).

The challenges discussed here also apply to simpler models based on Markov dynamics, such as hidden Markov models (HMMs) (32; 36; 2).

3.3.2 COMPARISON WITH A NONLINEAR GLOBAL MODEL WITH CONTROL

Our C. elegans neural activity modeling is inspired by Fieseler et al. and Morrison et al. (3; 17). Unlike state space models, Morrison et al. discovered a minimally parameterized global nonlinear model with control that mimics Hidden Markov model state transitions within a single dynamical system. This model captures key features of the C. elegans calcium imaging data, including two stable fixed points for forward and reversal behaviors, state transitions triggered by control signals, and variability in transition trajectories that match neural activity data (17).

The model is represented as:

$$\dot{\mathbf{x}} = F(\mathbf{x}, \beta) + \mathbf{u}(t) \quad (12)$$

where β governs longer timescale dynamics, and $u(t)$ is a control signal operating on faster timescales. $u(t)$ is a one-dimensional signal that may integrate multiple local and non-local processes. This separation of intrinsic dynamics and control inputs increases the model’s interpretability.

Nonlinear control has been used in other biological networks to describe switching between multiple

stable states (21; 35; 34) A significant advantage is that a nonlinear model can have multiple fixed points corresponding to different behavioral states – in the case of *C. elegans*, forward and reversal motion. A heuristic model capturing *C. elegans* behavior is:

$$\begin{aligned}\dot{\mathbf{x}} &= \mathbf{y} \\ \dot{\mathbf{y}} &= -(\mathbf{x} - 1)(\mathbf{x} - \beta)(\mathbf{x} + 1) + \lambda \mathbf{y} + \mathbf{u}(t)\end{aligned}\quad (13)$$

When $u = 0$ this cubic system has two stable fixed points at $x = \pm 1$ corresponding to forward and reversal motions, as well as an unstable point at β . The fixed points correspond to locations in the state space where $F(x, \beta) = 0$ and $u(t) = 0$. Transitions to the other stable fixed point occur when $u(t) \neq 0$, corresponding to a shift in the behavior of the animal. To capture the stochasticity of the data, additional noise terms are added to \dot{x}, \dot{y} .

The model is fit to reproduce the dominant PCA mode of the neural activity data. Importantly, the optimization is done by using the manually annotated behavioral labels to determine when the control switches values. For forward and reversal motion, $u = 0$, while each type of turn (reversal to forward and vice versa) corresponds to a different u value. These distinct models are fitted to the corresponding time series segments based on the annotations. The resulting nonlinear control model and the parameters found through optimization is fully described in (17).

Both our model and the global nonlinear model with control described above employ nonlinear terms in the dynamics. These are global models with few parameters that capture the most important qualitative features in the *C. elegans* data.

A key difference is that the control variable in Morrison et al. takes 3 values, including 0 during stable states, while dynamic SINDy’s switching variable takes 2 values that influence the fixed points and are longer-lasting than the transient controls.

A key advantage of our method is that it is entirely data-driven, requiring no behavioral annotations or manual fitting. We directly input the low-dimensional neural activity time series, allowing dynamic SINDy to automatically discover the governing equations. This reduces the effort required from the data scientist while still capturing the system’s essential dynamics.

3.3.3 COMPARISON WITH A LINEAR MODEL WITH CONTROL

A related study proposed a global linear model with control whereby a linear dynamical system is actuated by temporally sparse control signals (3). Denoting $\mathbf{x}_j = \mathbf{x}(t_j)$, neural activity across neurons at time t_j , and \mathbf{X} a matrix of neuronal data at different snapshots in time, $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_m]$, dynamic mode decomposition (DMD) provides a linear model for the dynamics of the state space:

$$\mathbf{X}' = \mathbf{A}\mathbf{X} \quad (14)$$

where $\mathbf{X}' = [\mathbf{x}_2, \mathbf{x}_3, \dots, \mathbf{x}_{m+1}]$ is offset by one time step compared to \mathbf{X} . Since a linear model alone cannot capture the neural activity data, DMD with control (DMDc, (12)) is employed to distinguish between the underlying dynamics and control signals $\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_m]$, where $\mathbf{u}_j = \mathbf{u}(t_j)$ are actuation signals at a snapshot in time. DMDc regresses to the linear control system:

$$\mathbf{X}' = \mathbf{A}\mathbf{X} + \mathbf{B}\mathbf{U} \quad (15)$$

The control signal can either be fixed using manually annotated behavioral onsets in a supervised setting or learned jointly with \mathbf{A} and \mathbf{B} (Algorithm 1, (3)). To avoid trivial solutions, the control signals are constrained to be sparse, meaning transitions between states should be infrequent. The following loss function, incorporating an l_0 regularization, is minimized using the sequential least squares thresholding algorithm:

$$\text{loss} = \min_{\mathbf{A}, \mathbf{B}, \mathbf{U}} \|\mathbf{A}\mathbf{X} + \mathbf{B}\mathbf{U} - \mathbf{X}'\|_2 + \lambda \|\mathbf{U}\|_0 \quad (16)$$

If control signals are internally generated, they are either random or encoded within the network. Sparse variable selection and time-delay embeddings test the influence of present and past data to determine which neurons predict the controls:

$$\mathbf{u}_k = \mathbf{K}_1 \mathbf{x}_k + \mathbf{K}_2 \mathbf{x}_{k-1} + \dots \quad (17)$$

Key findings from this study include that the unsupervised algorithm produces control signals somewhat correlating with manually annotated behavioral onsets, though it fails to capture forward motion onsets. This suggests that neurons involved in forward motion¹ require non-trivial nonlinearities throughout the time series for full reconstruction, not just control signals at the onset. Reversal neurons are well-modeled by the supervised control signals, implying fewer required nonlinearities other than the transition signal itself. Turns are also largely captured by the neuronal activity in specific cells, although there is much more variability.

The global linear framework, with internally generated control signals, partially explains neuronal activity but produces a weak qualitative and quantitative match with the data. For instance, the correlation between real neuronal activity and the model is at or only slightly above 0.5, even for neurons not implicated in forward motion, that the model supposedly is successful in capturing. Incorporating manual annotations significantly improves the model’s accuracy, but it shifts the approach from data-driven to supervised.

The inability to capture forward motion and the ineffectiveness of forward control signals demonstrate that nonlinearities are necessary for many interactions in the system. A linear model with control can produce only one fixed point, with all other states simply being longer-lived, conflicting with empirical evidence that both forward and reversal behaviors in *C. elegans* are stable states (3). Empirical studies have instead shown that multiple behavioral states appear to be stable (18).

These results support our approach using dynamic SINDy, which automatically identifies nonlinear systems with multiple fixed points, without requiring manual annotations. Dynamic SINDy discovers a dynamical system with switching signals as opposed to controls, while incorporating nonlinearities that could be critical in capturing complex neural dynamics. Furthermore, our framework, as described in Sec. 4.4, could discover how transition signals depend on the data to create a fully closed-loop feedback system. Future work should confirm that our low-dimensional model can explain neural activity according to cell class.

4 OTHER METHODS FOR SYSTEM IDENTIFICATION OF NON-AUTONOMOUS DYNAMIC SYSTEMS

4.1 SWITCHING LINEAR DYNAMICAL SYSTEM (SLDS)

The generative model for switching linear dynamical systems (SLDS) is as follows: for each time $t = 1, 2, \dots, T$, there is a discrete latent state $z_t \in \{1, 2, \dots, K\}$ that follows Markov dynamics:

$$z_{t+1}|z_t, \{\pi_k\}_{k=1}^K \sim \pi_{z_t},$$

where $\{\pi_k\}_{k=1}^K$ is the Markov transition matrix and $\pi_k \in [0, 1]^K$ is the k -th matrix row. In addition to z_t , there is a continuous latent state $x_t \in \mathbb{R}^M$ following linear dynamics that depend on z_t :

$$x_{t+1} = A_{z_{t+1}}x_t + b_{z_{t+1}} + v_t, v_t \sim N(0, Q_{z_t}) \quad (18)$$

where A_{z_t}, Q_{z_t} are matrices and b_{z_t} is a vector depending on the latent state $z_t \in 1, \dots, K$. In addition, we have access to observables y_t , generated from the continuous latent state x_t :

$$y_t = Cx_t + d + w_t, w_t \sim N(0, S) \quad (19)$$

where C, S, d are shared matrices and a vector across different discrete states z_t . We denote the complete set of parameters as $\theta = \{\pi_k, A_k, Q_k, b_k, C, S, d | k = 1, \dots, K\}$ and learn SLDS using Bayesian inference and a set of convenient priors as detailed in (28).

An extension of SLDS – rsLDS – allows the discrete switches to depend on the continuous latent state and exogenous inputs through a logistic regression (28). Specifically, when a discrete switch occurs whenever a continuous state enters a particular region of state space, SLDS is unable

¹This comparison between model and data at the single neuron level is only possible due to identification of neurons with stereotyped identities, as described in (24)

to learn this dependence, while rSLDS designed to address this state dependence. An important contribution of (28) is an inference algorithm leveraging Polya-gamma auxiliary variable methods to make inference fast, scalable, and, easy. We make use of this algorithm through the open-source rSLDS libraries (26).

4.1.1 TRAINING

To train the SLDS/rSLDS models, we first select the number of latent states in advance. For switch-like underlying coefficients, we choose two states to maximize SLDS's ability to identify the switching dynamics. We then perform principal component analysis (PCA) on the data, followed by fitting a simpler autoregressive hidden Markov model (AR-HMM), which lacks continuous latent states. The discrete latent variables z inferred from the AR-HMM, along with the data projected onto a small number of principal components, are used to initialize the SLDS/rSLDS algorithms. The SLDS/rSLDS training algorithms are implemented from the following github repository: <https://github.com/slinderman/recurrent-slids>.

4.2 A METHOD BASED ON GROUP SPARSITY

We focus on two studies (29; 8) that address system identification in non-autonomous systems. Both studies propose a method for identifying ODEs with time-varying SINDy coefficients by dividing the trajectory into smaller time windows and applying SINDy to each segment while maintaining the same sparsity pattern across all segments. In (29), this sparsity pattern is enforced using group sparsity regularization, while (8) introduces a novel algorithm based on sequential thresholding least squares (STLQ) from the original SINDy paper (30). This STLQ adaptation averages the SINDy coefficients across time windows and compares the average to a threshold, setting coefficients below this threshold to zero.

The group sparse penalized method for model selection and parameter estimation is used with datasets of multiple trajectories that share the same physical laws, but differ in bifurcation parameters (8). This framework is subsequently adapted to switching systems, whereby in a Lorenz system, the parameter α changes from -1 to 6.66 at some unknown time. The framework matches our problem, therefore we adapt the algorithm proposed in this analysis to the non-autonomous dynamical systems we study (Sec. 3.3).

Adapting the notation in (8) to our own, we have a total of M time windows that partition the trajectory, and we denote time windows by i . Data points from specific time windows are indexed by superscript i , while different variables of the system are denoted by subscript j , $j = 1, \dots, n$. For instance, in the Lorenz system variables x, y, z correspond to x_j , $j = 1, 2, 3$, where x_j^i corresponds to variable x_j within time window i . We can then define variable Ξ_j in terms of ξ_j^i which are SINDy coefficients corresponding to variable j within time window i :

$$\Xi_j = \begin{bmatrix} | & | & & | \\ \xi_j^1 & \xi_j^2 & \dots & \xi_j^M \\ | & | & & | \end{bmatrix}$$

Next we can define the data matrix $X^{(i)}$, the velocity matrix $\dot{X}^{(i)}$, and the dictionary matrix $\Theta^{(i)}$ as:

$$X^i = \begin{bmatrix} | & | & \dots & | \\ x_1^i & x_2^i & \dots & x_n^i \\ | & | & & | \end{bmatrix} = \begin{bmatrix} x_1(t_1; \lambda^{(i)}) & x_2(t_1; \lambda^{(i)}) & \dots & x_n(t_1; \lambda^{(i)}) \\ x_1(t_2; \lambda^{(i)}) & x_2(t_2; \lambda^{(i)}) & \dots & x_n(t_2; \lambda^{(i)}) \\ \dots & \dots & \dots & \dots \\ x_1(t_{l_i}; \lambda^{(i)}) & x_2(t_{l_i}; \lambda^{(i)}) & \dots & x_n(t_{l_i}; \lambda^{(i)}) \end{bmatrix}$$

$$\dot{X}^i = \begin{bmatrix} | & | & \dots & | \\ \dot{x}_1^i & \dot{x}_2^i & \dots & \dot{x}_n^i \\ | & | & & | \end{bmatrix} = \begin{bmatrix} \dot{x}_1(t_1; \lambda^{(i)}) & \dot{x}_2(t_1; \lambda^{(i)}) & \dots & \dot{x}_n(t_1; \lambda^{(i)}) \\ \dot{x}_1(t_2; \lambda^{(i)}) & \dot{x}_2(t_2; \lambda^{(i)}) & \dots & \dot{x}_n(t_2; \lambda^{(i)}) \\ \dots & \dots & \dots & \dots \\ \dot{x}_1(t_{l_i}; \lambda^{(i)}) & \dot{x}_2(t_{l_i}; \lambda^{(i)}) & \dots & \dot{x}_n(t_{l_i}; \lambda^{(i)}) \end{bmatrix}$$

and

$$\Theta^i = [\mathbf{1}_{l_i,1} \quad X^i \quad (X^i)^2 \quad (X^i)^3 \quad \dots]$$

Using this notation, the optimization problem can be rewritten as a least-square fitting :

$$\min_{\Xi_j} \sum_{i=1}^n \|\Theta^i \xi_j^i - \dot{X}_j^i\|_2^2 \quad (20)$$

for each $j = 1, \dots, n$.

To help prevent overfitting, we add regularization to this cost function, by including a penalty on the number of active candidate functions. The main assumption of this method is that coefficients ξ^i have the same support set (in j) for each i , but can differ in value. Thus we can group each row j together to be either zero or nonzero, therefore the number of active (nonzero) rows in Ξ_j is sparse. The cost function now implements the following group-sparse optimization problem:

$$\min_{\Xi_j} \sum_{i=1}^m \|\Theta^i \xi_j^i - \dot{X}_j^i\|_2^2 + \lambda \|\Xi_j\|_{2,0} \quad (21)$$

where the $l_{2,0}$ penalty is defined as:

$$\|A\|_{2,0} = \#\{k : (\sum_{l=1}^m |a_{kl}|^2)^{1/2} \neq 0\} \quad (22)$$

for any matrix $A = [a_{k,l}]$. Although the problem is non-convex, the authors in (8) propose to solve it numerically using an iterative hard thresholding algorithm, the *group hard-iterative thresholding algorithm for dynamical systems*:

Group Hard-Iterative Thresholding Algorithm for Dynamical Systems:

```

1: Given: initialization matrix  $\Xi^{(0)}$ , tol and parameters  $\gamma$ .
2:
3: while  $\|\Xi^{(k+1)} - \Xi^{(k)}\| > tol$  do
4:
5:   for  $i = 1$  to  $m$  do
6:
7:      $(\xi^i)^{(k+1)} = (\xi^i)^{(k)} - (\Theta^i)^T (\Theta^i (\xi^i)^{(k)} - \Xi^i)$ 
8:
9:   end for
10:
11:    $S^{(k+1)} = \text{supp}(H_{\sqrt{\gamma}}[\xi^1, \xi^2, \dots, \xi^m])$ 
12:
13:   for  $i = 1$  to  $m$  do
14:
15:      $(\xi^i)^{(k+1)} = \text{argmin}_{\xi^i} \|\Theta^i \xi^i - \dot{X}^i\|_2^2$  s.t.  $\text{supp}(\xi^i) \subset S^{k+1}$ 
16:
17:   end for
18:
19: end while
20:
```

where $\text{supp}(x)$ is the support set of x , i.e. the indices of x that correspond to the nonzero values.

While we implement this algorithm and test it on the data, we have found, surprisingly, that a simpler algorithm is more effective in many cases:

Simple sequential thresholding algorithm:

Solve $\Theta^i \Xi^i = \dot{X}^i$ for each time window i and trajectory X and stack these least squares results in a matrix $\tilde{\Xi}^{(0)}$ to be used as initial condition

```

702 Choose threshold
703 for j = 1 to 100 do %100 iterations
704     %average over time windows  $s$  and compare to threshold:
705     smallinds  $\leftarrow \{k_1, k_2 \mid < |\tilde{\Xi}^{(j-1)}[k_1, k_2, s]| >_s < \text{threshold}\}$ 
706     biginds  $\leftarrow \{k_1, k_2 \mid > |\tilde{\Xi}^{(j-1)}[k_1, k_2, s]| >_s > \text{threshold}\}$ 
707     for i = 1 to m do
708         Solve for  $\Xi^i$ :  $\Theta^i[\text{biginds}] \cdot (\Xi^i[\text{biginds}])^{(j)} = \dot{X}^i$ 
709          $(\Xi^i[\text{smallinds}])^{(j)} = 0$  %coefficients are 0 all across time windows i
710     end for
711      $\tilde{\Xi}^{(j)} = [(\Xi^1)^{(j)}, \dots, (\Xi^m)^{(j)}]$ 
712 end for

```

Comments starting with “%” are provided throughout the code to clarify its meaning. We use this algorithm throughout to showcase our results using the group sparsity method.

For training, we have varied the total time for the trajectories, the number of batches used, the time window, as well as the precise algorithm used. Throughout these experiments we have found that the algorithm was highly sensitive to whether the data was normalized or not, specifically we have found that **not** normalizing the data yielded superior results.

REFERENCES

- [1] Goyal A, Ke NR Sordoni A, Côté MA, and Bengio Y. Z-forcing: Training stochastic recurrent networks. *dvances in Neural Information Processing Systems (NeurIPS)*. Long Beach, CA, 2017.
- [2] Arous BJ, Laffont S, and Chatenay D. Molecular and sensory basis of a food related two-state behavior in *c. elegans*. *PLoS One*, 4(10):e7584, 2009.
- [3] Fieseler C, Kunert-Graf J, and Kutz JN. The control structure of the nematode *caenorhabditis elegans*: Neuro-sensory integration and proprioceptive feedback. *J. Biomech.*, 74:1–8, 2018.
- [4] Louizos C, Welling M, and Kingma DP. Learning sparse neural networks through l_0 regularization.
- [5] Abhyuday Desai, Cynthia Freeman, Zuhui Wang, and Ian Beaver. Timevae: A variational auto-encoder for multivariate time series generation, 2021.
- [6] Rezende DJ, Mohamed S, and Wierstra D. Stochastic backpropagation and approximate inference in deep generative models. *International Conference on Machine Learning*, 2014.
- [7] Paquet U Fraccaro M, Kamronn S and Winther O. A disentangled recognition and nonlinear dynamics model for unsupervised learning. *Advances in Neural Information Processing Systems (NeurIPS)*. Long Beach, CA, 2017.
- [8] Schaeffer H, Tran G, and Ward R. Learning dynamical systems and bifurcation via group sparsity, 2013.
- [9] Bayer J and Osendorfer C. Learning stochastic recurrent networks. 2014.
- [10] Chung J, Kastner K, Goel K Dinh L, Courville A, and Bengio Y. A recurrent latent variable model for sequential data. *Advances in Neural Information Processing Systems (NeurIPS)*. Montréal, Canada, 2015.
- [11] Yoon J, Jarrett D, and van der Schaar M. Time-series generative adversarial networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.
- [12] Proctor JL, Brunton SL, and Kutz JN. Dynamic mode decomposition with control. *SIAM J. Appl. Dyn. Syst.*, S 15:142–161, 2016.
- [13] Girin L, Leglaive S, Bie X, Diard J, Hueber T, and Alameda-Pineda X. *Dynamical Variational Autoencoders: A Comprehensive Review*. 2021.
- [14] Jiang LP and Rao RPN. Dynamic predictive coding: A model of hierarchical sequence learning and prediction in the neocortex. *PLoS Comput Biology*;20(2):e1011801, 2024.
- [15] Fraccaro M, Sønderby SK, Paquet U, and Winther. Sequential neural models with stochastic layers. *Advances in Neural Information Processing Systems (NeurIPS)*. Barcelona, Spain, 2016.
- [16] Jacobs M, Brunton BW, Brunton SL, Kutz JN, and Raut RV. Hypersindy: Deep generative modeling of nonlinear stochastic governing equations, 2023.
- [17] Morrison M, Fieseler C, and Kutz JN. Nonlinear control in the nematode *c. elegans*. *Frontiers in Computational Neuroscience*, 14, 2021.
- [18] Morrison M, Fieseler C, and Kutz JN. Nonlinear control in the nematode *c. elegans*. *Frontiers in Computational Neuroscience*, 2021.
- [19] Kingma PD and Welling M. Auto-encoding variational bayes, 2013.
- [20] Kingma PD and Welling M. An introduction to variational autoencoders. *Foundations and Trends in Machine Learning: Vol. 12: No. 4, pp 307-392*, 2019.
- [21] Purnick PEM and Weiss R. The second wave of synthetic biology: from modules to systems. *Nat. Rev. Mol. Cell Biol.*, 10:410–422, 2009.

810 [22] Krishnan R, Shalit U, and Sontag D. Deep kalman filters. 2015.
811
812 [23] Krishnan R, Shalit U, and Sontag D. *AAAI Conference on Artificial Intelligence, San Francisco,*
813 *CA*, 2017.
814 [24] Kato S, Kaplan H. S., Schrödel T, Skora S, Lindsay TH, Yemini E, Lockery S, and Zimmer M.
815 Global brain dynamics embed the motor command sequence of *caenorhabditis elegans*. *Cell*,
816 163(3):656–669, 2015.
817
818 [25] Leglaive S, Girin L, and Horaud R. “a variance modeling framework based on variational
819 autoencoders for speech enhancement. *IEEE International Workshop on Machine Learning for*
820 *Signal Processing (MLSP), Aalborg, Denmark*, 2018.
821 [26] Linderman S. recurrent-slds. [https://github.com/slinderman/](https://github.com/slinderman/recurrent-slds)
822 [recurrent-slds](https://github.com/slinderman/recurrent-slds), 2016.
823
824 [27] Linderman S., Nichols A., Blei D., Zimmer M, and Paninski L. Hierarchical recurrent state
825 space models reveal discrete and continuous dynamics of neural activity in *c. elegans*, 2019.
826 [28] Linderman S, Johnson M, Miller A, Adams R, Blei D, and Paninski L. Bayesian learning and
827 inference in recurrent switching linear dynamical systems. *In Proc. of the 20th Int. Conf. on*
828 *Artificial Intelligence and Statistics, vol. 54 (eds A Singh, J Zhu), Proc. of Machine Learning*
829 *Research, pp. 914–922. Fort Lauderdale, FL: JLMR: WCP.*, 2017.
830 [29] Rudy SH, Brunton SL, Proctor JL, and Kutz JN. Data-driven discovery of partial differential
831 equations. *Sci. Adv.* 3, e1602614., 2017.
832
833 [30] Brunton SL, Proctor JL, and Kutz JN. Discovering governing equations from data by sparse
834 identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*
835 *(PNAS)*, 2016.
836 [31] Flavell SW, Pokala N, Macosko EZ, DR Albrecht, Larsch J, and Bargmann CI. Serotonin
837 and the neuropeptide pdf initiate and extend opposing behavioral states in *c. elegans*. *Cell*,
838 154:1023–1035, 2013.
839
840 [32] Gallagher T, Bjorness T, Greene R, You Y-J, and Avery L. The geometry of locomotive
841 behavioral states in *c. elegans*. *PLoS ONE*, 8:e59865, 2013.
842 [33] Wakabayashi T, Kitagawa I, and Shingai R. Neurons regulating the duration of forward
843 locomotion in *caenorhabditis elegans*. *Neurosci. Res.*, 50:103–111, 2004.
844
845 [34] Kepler TB and Elston TC. Stochasticity in transcriptional regulation: origins, consequences,
846 and mathematical representations. *Biophys. J.*, 81:3116–3136, 2001.
847 [35] Gardner TS, Cantor CR, and Collins JJ. Construction of a genetic toggle switch in *escherichia*
848 *coli*. *Nature*, 403:339–342, 2000.
849
850 [36] Roberts WM, Augustine SB, Lawton KJ, Lindsay TH, Thiele TR, Izquierdo EJ, Faumont S,
851 Lindsay RA, Britton MC, Pokala N, Bargmann CI, and Lockery SR. A stochastic neuronal
852 model predicts random search behaviors at multiple spatial scales in *c. elegans*. *eLife*, 5:e12572,
853 2016.
854 [37] Li Y and Mandt S. Disentangled sequential autoencoder. *International Conference on Machine*
855 *Learning (ICML). Stockholm, Sweden*, 2018.
856
857
858
859
860
861
862
863

Supplementary Figures

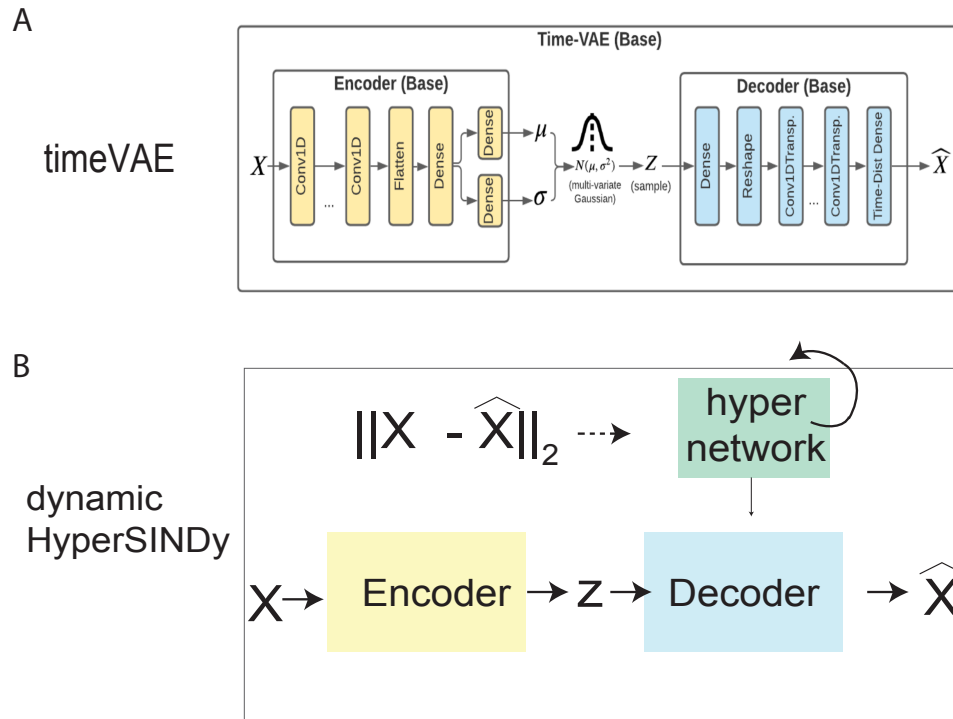


Figure 1: (A) Schematic of timeVAE architecture from (5); (B) Schematic of dynamic HyperSINDy architecture described in SM Sec. 1.2.2

Set of synthetic datasets used to test dynamic SINDy
(coefficient time series and trajectory)

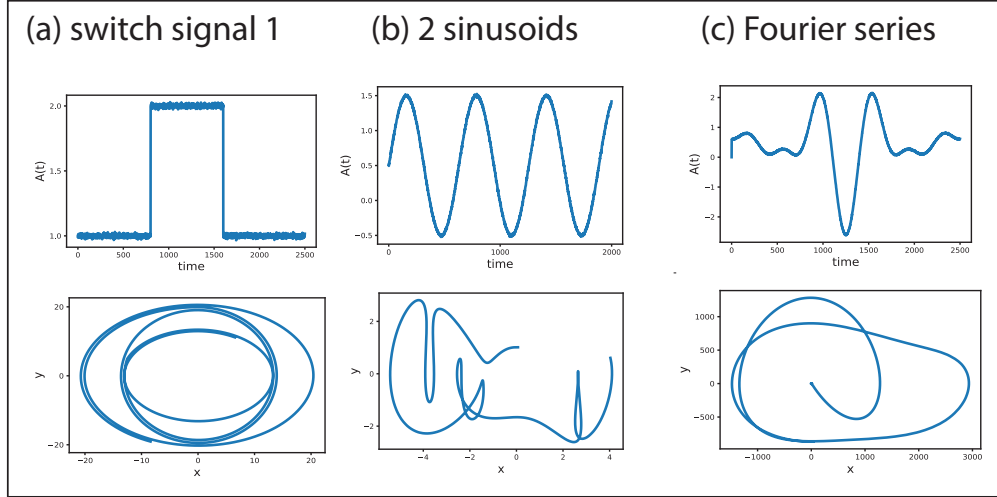


Figure 2: Time-varying coefficients (above) and corresponding dynamics (below) for the non-autonomous harmonic oscillator of Eq. (3) (main text). Complementary to Figure 1A (main text)

dynamic HyperSINDy: non-autonomous harmonic oscillators
 $A(t)$, $B(t)$ sinusoids

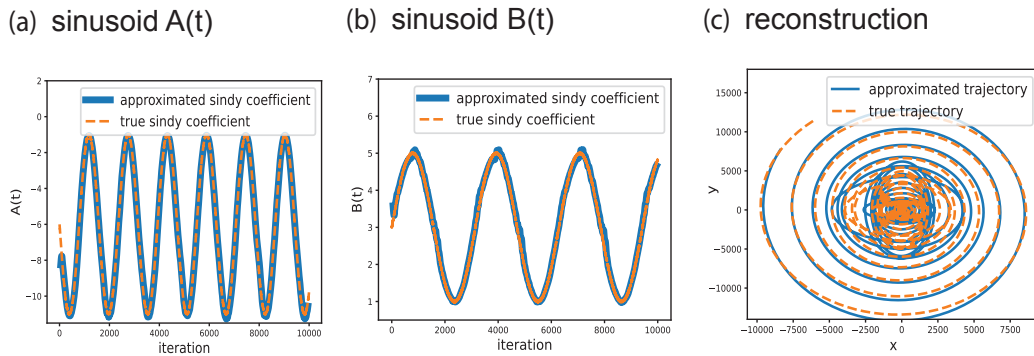


Figure 3: Data-driven discovery of sinusoid SINDy coefficients (a and b) and trajectory reconstruction (c) of a non-autonomous harmonic oscillator with dynamic HyperSINDy.

dynamic HyperSINDy: non-autonomous harmonic oscillators
 $A(t)$, $B(t)$ sigmoid and switch signals

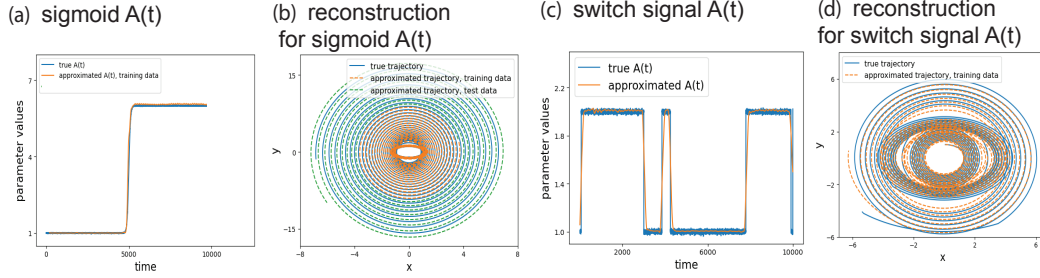


Figure 4: Data-driven discovery of sigmoid and switch signal coefficients (a and c) and the corresponding trajectory reconstruction (b and d) of a non-autonomous harmonic oscillator with dynamic HyperSINDy.

dynamic HyperSINDy: Lorenz dynamical system
 $\sigma(t)$ sigmoid and sinusoid

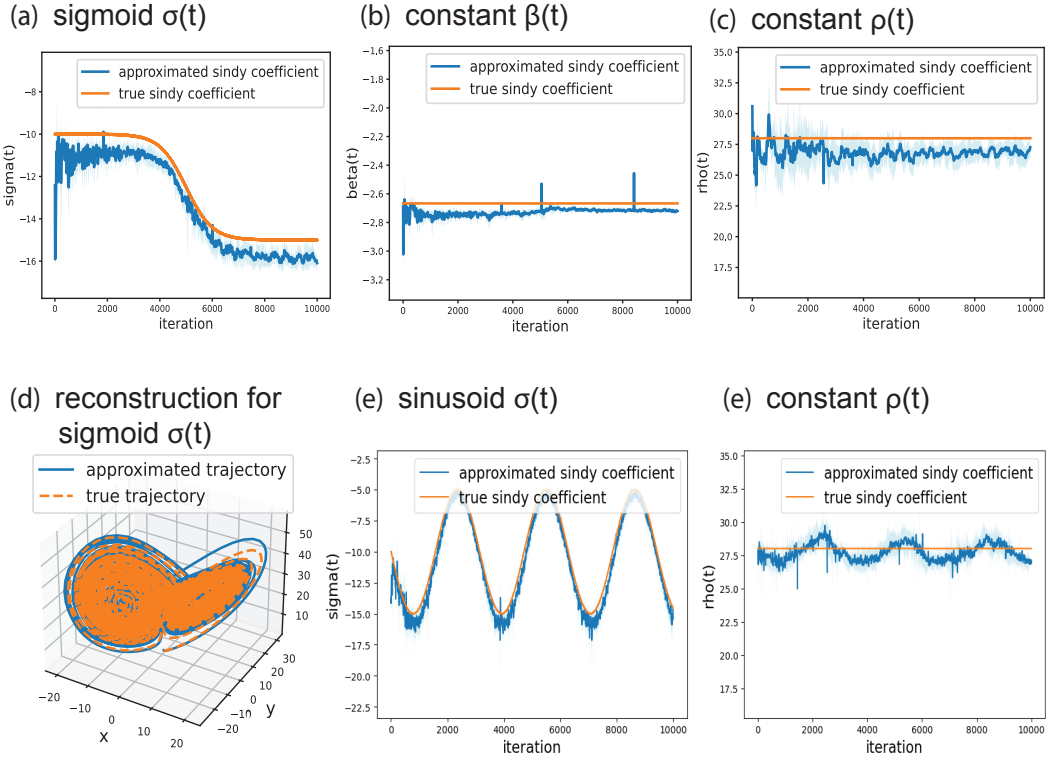


Figure 5: Data-driven discovery of sigmoid (a), constant (b, c and e) and sinusoid (e) SINDy coefficients and trajectory reconstruction (d) of the Lorenz dynamics with dynamic HyperSINDy. Time series of coefficients that correspond to constants in the real dynamics sometimes inherit frequency content from the dynamics, as in (e).

Reconstruction of \dot{X} trajectory

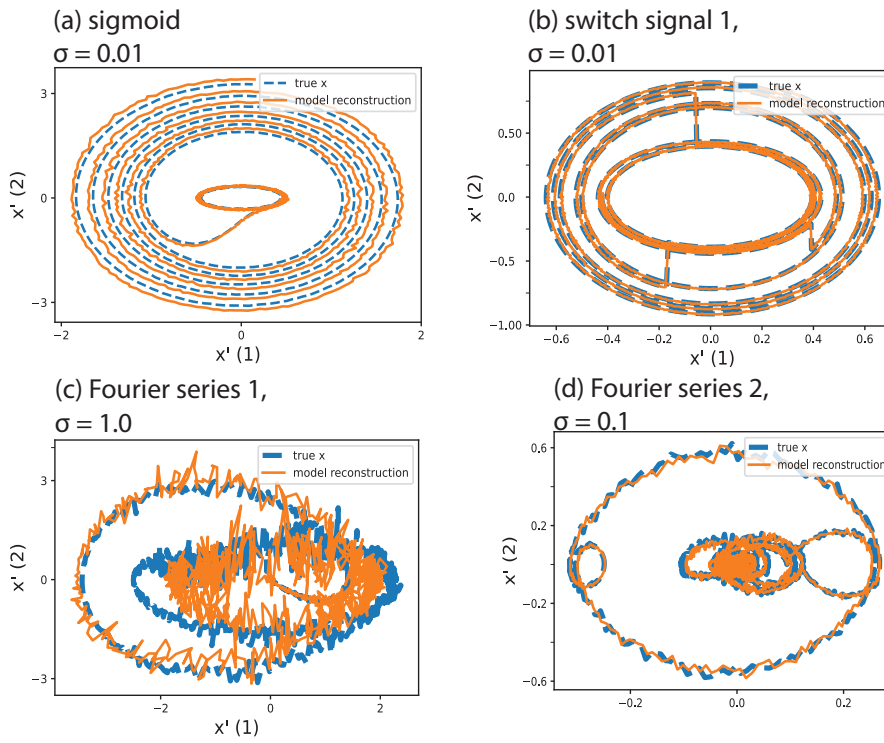


Figure 6: Trajectory reconstructions with timeVAE for a non-autonomous harmonic oscillator with different coefficients (sigmoid, switch signal, finite Fourier series) and different levels of noise in the coefficients (0.01, 0.1, 1).