Learning nonlinear dynamical systems from data using scientific machine learning

Romit Maulik
Assistant Computational Scientist, Argonne National Laboratory

https://github.com/Romit-Maulik/UChicago_SummerSchool_2022

August 9, 2022

Outline

1. Introduction and motivation.
2. Learning deterministic dynamical systems.
3. Learning stochastic dynamical systems.
4. Conclusions/Q&A.

Motivation

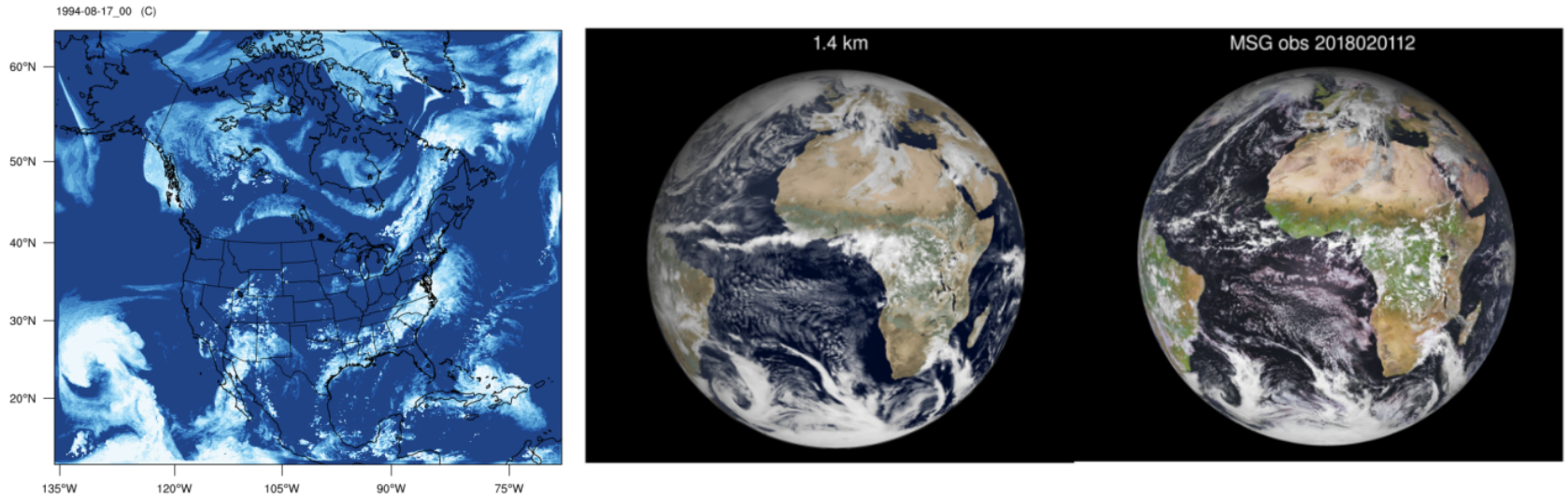


Figure: Cloud resolved weather and climate simulations are becoming a reality. 4km simulations of E3SM run over 100 forecast years require 120 million core hours (Theta-ANL) and 12 PB of storage data (250 GB/forecast day).

Image source: Jung et al., Simulations of E3SM on ANL-Theta, 2022 (top), ECMWF Simulations on ORNL-Summit, DOE E3SM All-hands meeting 2021.

Emulating dynamical systems from data

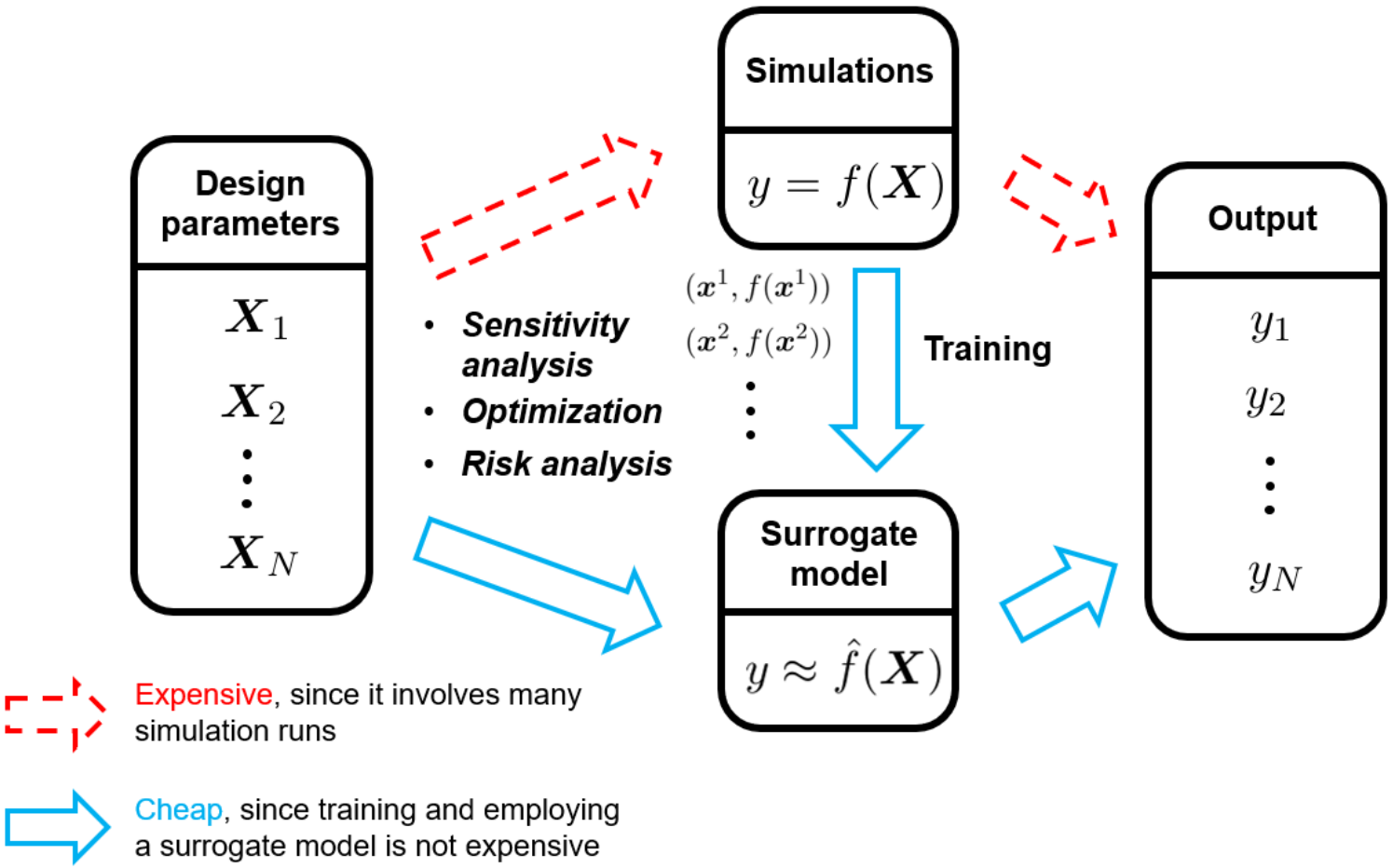


Figure: Source - "An introduction to surrogate modeling" - Shuai Guo.

Emulating dynamical systems from data

There may also be a requirement to construct 'non-intrusive' surrogate models - for example when dynamics are only partially understood/known - i.e., **No closed form governing laws available**.

This project is joint work with

- Prasanna Balaprakash (Argonne).
- Qi Tang, Joshua Burby (Los Alamos).
- Alec Linot, Mike Graham (Wisconsin).
- Varun Shankar, Vedant Puri, Venkat Vishwanathan (CMU).

Background: Neural ordinary differential equations

$$\frac{d\mathbf{a}}{dt} = f(\mathbf{a}, \theta), \quad (\theta) \in \Theta, \tag{1}$$

where $\Theta \subset \mathbb{R}^{N_w}$ is the space of trainable parameters of an arbitrary neural network. The NODE [3, 4, 5] approximates the latent-space evolution as a set of ordinary differential equations that can be trained through adjoint-based (i.e., continuous) backpropagation [3, 5], i.e.,

$$L(\tilde{\mathbf{a}}^T) = L(\mathbf{a}^0 + \int_{t=0}^{t=T} f(\mathbf{a}(\mathbf{t}), \theta) dt) \tag{2}$$

$$\frac{d\mathbf{z}}{dt} = -\mathbf{z}^T \frac{\partial f(\mathbf{a}, t, \theta)}{\partial \mathbf{a}}, \quad \mathbf{z}(t) = \frac{\partial L}{\partial \mathbf{a}(t)} \tag{3}$$

$$\frac{dL}{d\theta} = -\int_{t=T}^{t=0} \mathbf{z}(t)^T \frac{\partial f(\mathbf{a}(\mathbf{t}), \theta)}{\partial \theta} dt. \tag{4}$$

Chaotic dynamics: The Kuramoto-Sivashinsky equation

We want to address the surrogate modeling of chaotic systems. Traditionally, most data-driven time-series modeling techniques suffer with deterministic chaos.

$$\frac{\partial u}{\partial t} = -u\frac{\partial u}{\partial x} - \frac{\partial^2 u}{\partial x^2} - \frac{\partial^4 u}{\partial x^4} \tag{5}$$

$$u \in \mathbb{R}^{64}; x \in [-\pi, \pi] \subset \mathbb{R}^1 \tag{6}$$

A prototypical system to study chaotic dynamics, possesses a dissipative nature (i.e., an attractor in the long-term limit), challenging for state-of-the-art black-box forecasting methods.

A novel neural ODE for capturing chaotic attractors

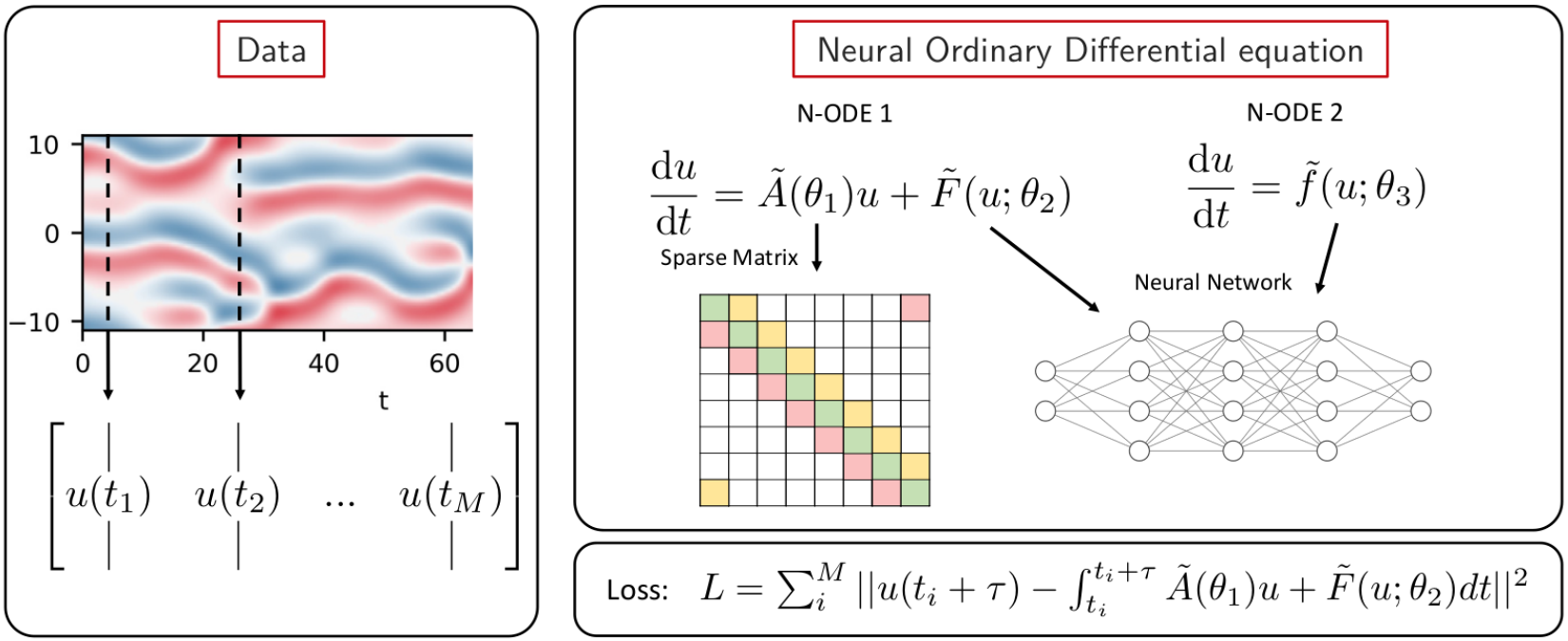


Figure: A novel neural-ODE for learning chaotic dynamics.

Linot, Burby, Tang, Balaprakash, Graham, RM. arXiv:2203.15706.

Example: The Kuramoto-Sivashinsky equation

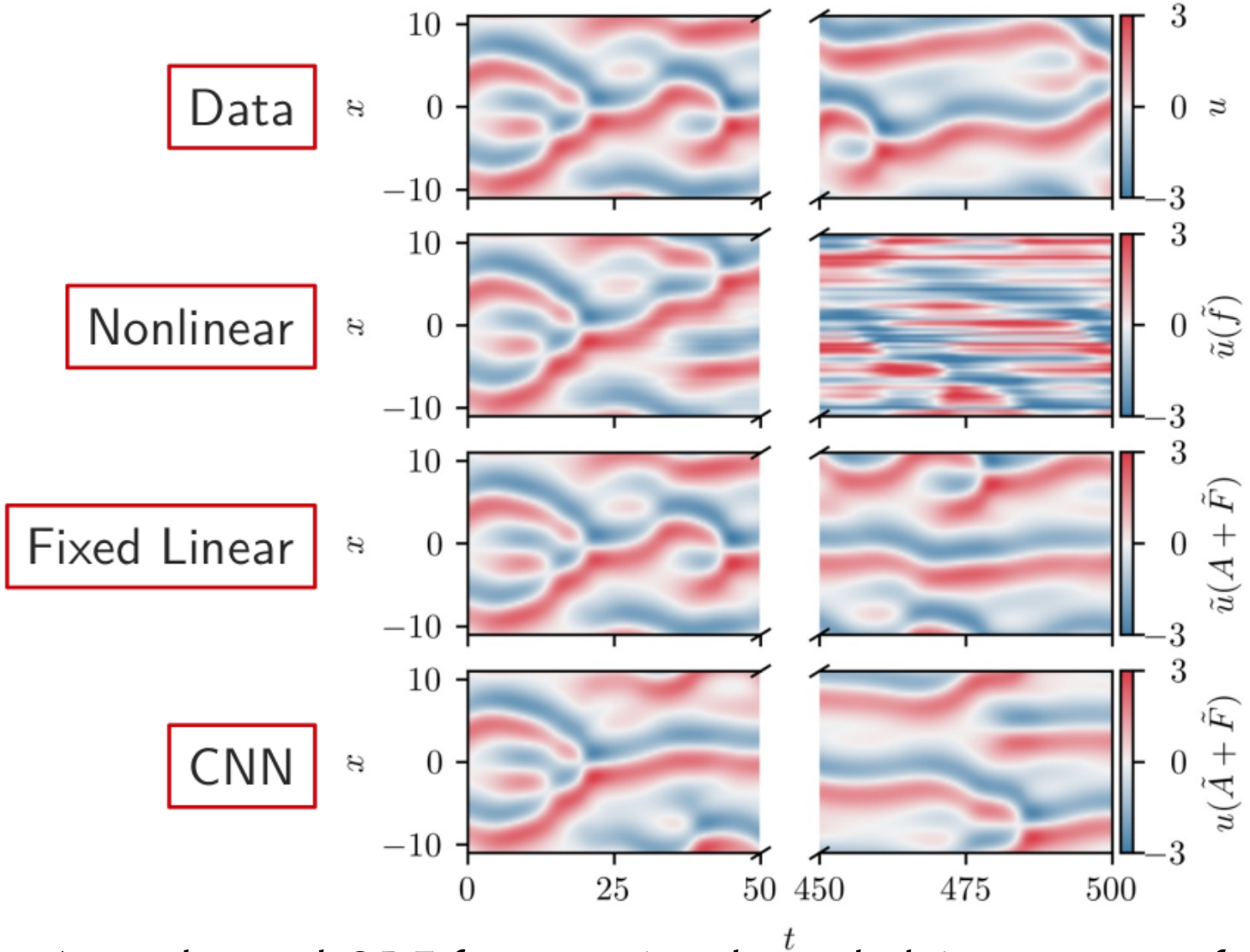


Figure: A novel neural-ODE for capturing the underlying attractors for the KS equations: Long-term stability.

Example: The Kuramoto-Sivashinsky equation

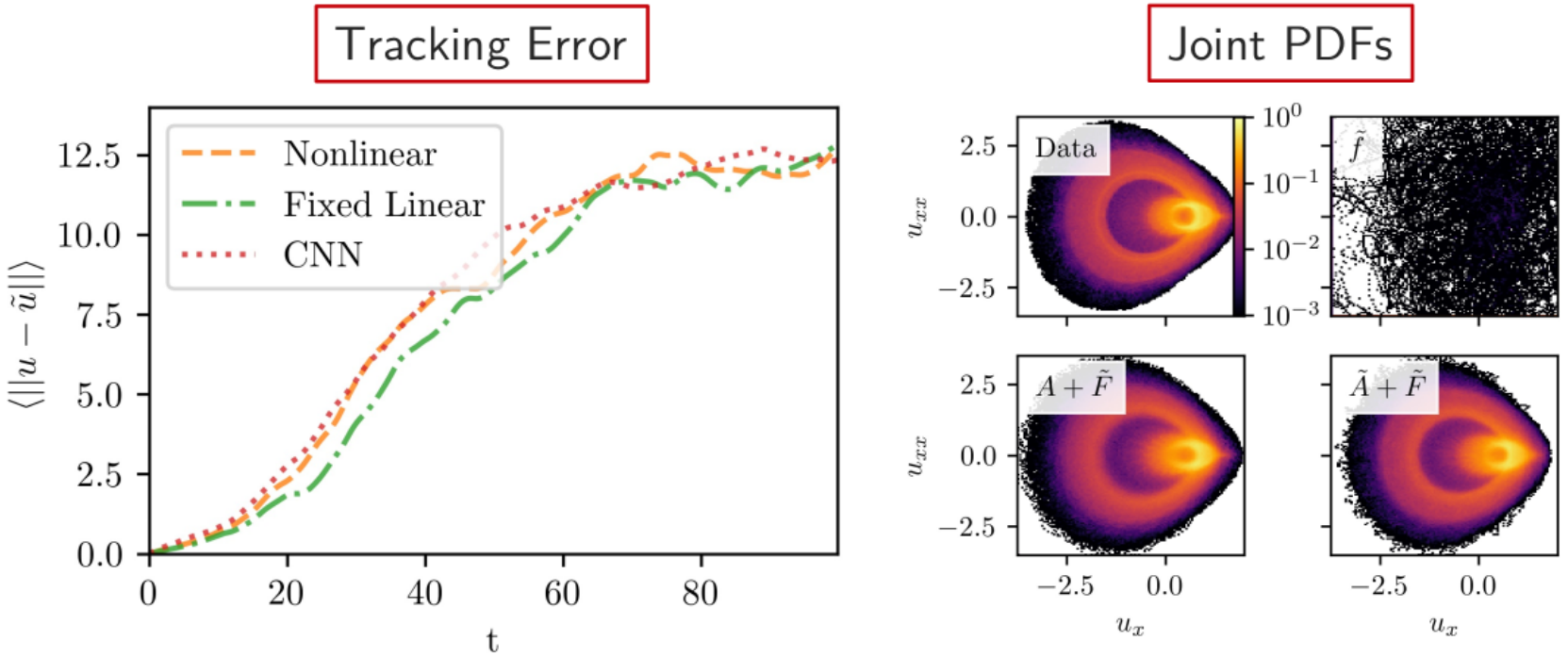


Figure: A novel neural-ODE for capturing the underlying attractors for the KS equations: Attractor captured successfully!

Approximate inertial manifold theory can then be used since we have a linear term.

Flipping the script: A reduced-order model of a surrogate

We can also reduce the order of this neural ODE a-posteriori by using the theory of approximate inertial manifolds [9]:

$$\frac{dp}{dt} = Ap + PF(p+q), \tag{7}$$

$$\frac{dq}{dt} = Aq + QF(p+q), \tag{8}$$

$$q = A^{-1}Q(p+q) \approx A^{-1}Q(p). \tag{9}$$

$$Q = I - P. \tag{10}$$

If we construct P using selected eigenvectors of the learned linear term A

$$AV = V\Lambda,$$

$$P = \tilde{V}\tilde{V}^T \tag{11}$$

where $\tilde{V}$ are a truncated subset of eigenvectors that promotes $\frac{dq}{dt} = 0$.

Ignoring the computation of q gives us the nonlinear Galerkin ROM, computing q with $A^{-1}Q(p)$ during the simulation gives us the AIM ROM and after the simulation gives us postprocessing ROMs.

Example: The Kuramoto-Sivashinsky equation

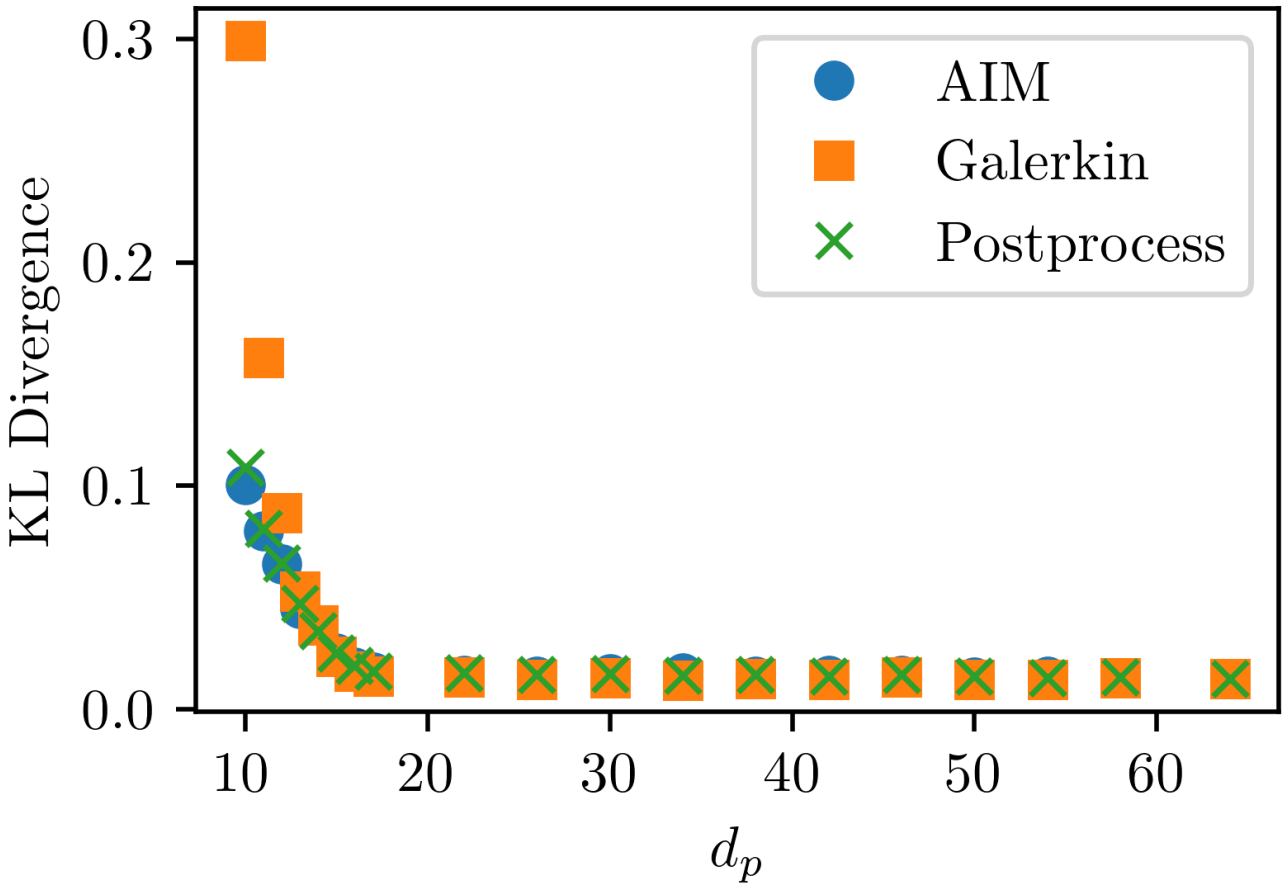


Figure: A reduced-order model from the proposed full-order neural ODE. KL-divergence of attractor statistics - model reduced to 25% of original size.

Example: Learning the viscous Burgers equations

The viscous Burgers equations are given by the following system

$$\frac{\partial u}{\partial t} = -u\frac{\partial u}{\partial x} + \nu\frac{\partial^2 u}{\partial x^2} \tag{12}$$

in a domain with length $L = 1$ and with periodic boundary conditions. Our viscosity, $\nu = 8 \times 10^{-4}$. Initial conditions sampled from superpositions of frequencies in Fourier space for the same viscosity. Solution discretized on 512 grid points.

Example: Learning the viscous Burgers equations

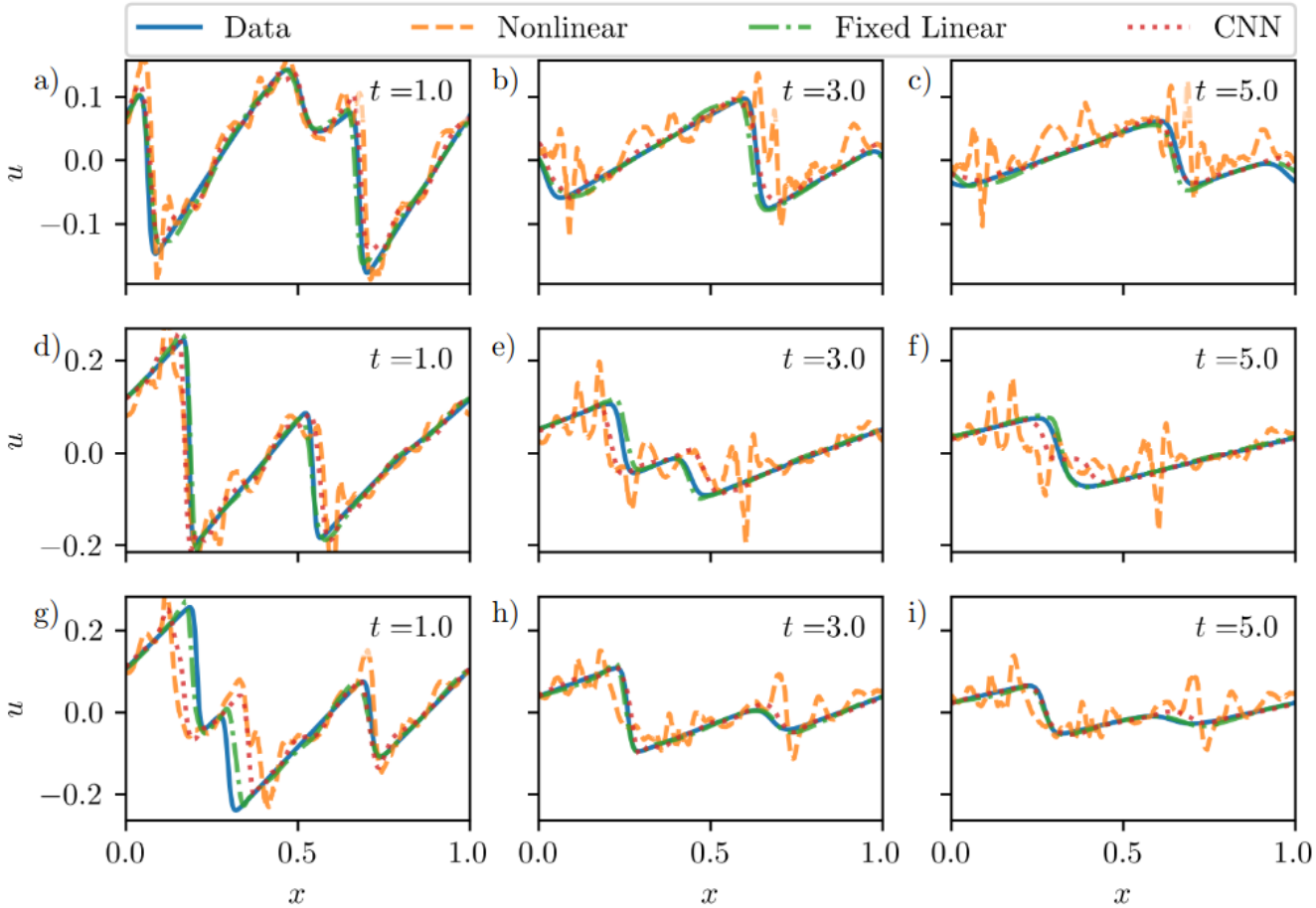


Figure: The stabilized neural ODE outperforms the standard neural ODE for learning the viscous Burgers equations.

Example: Learning the viscous Burgers equations

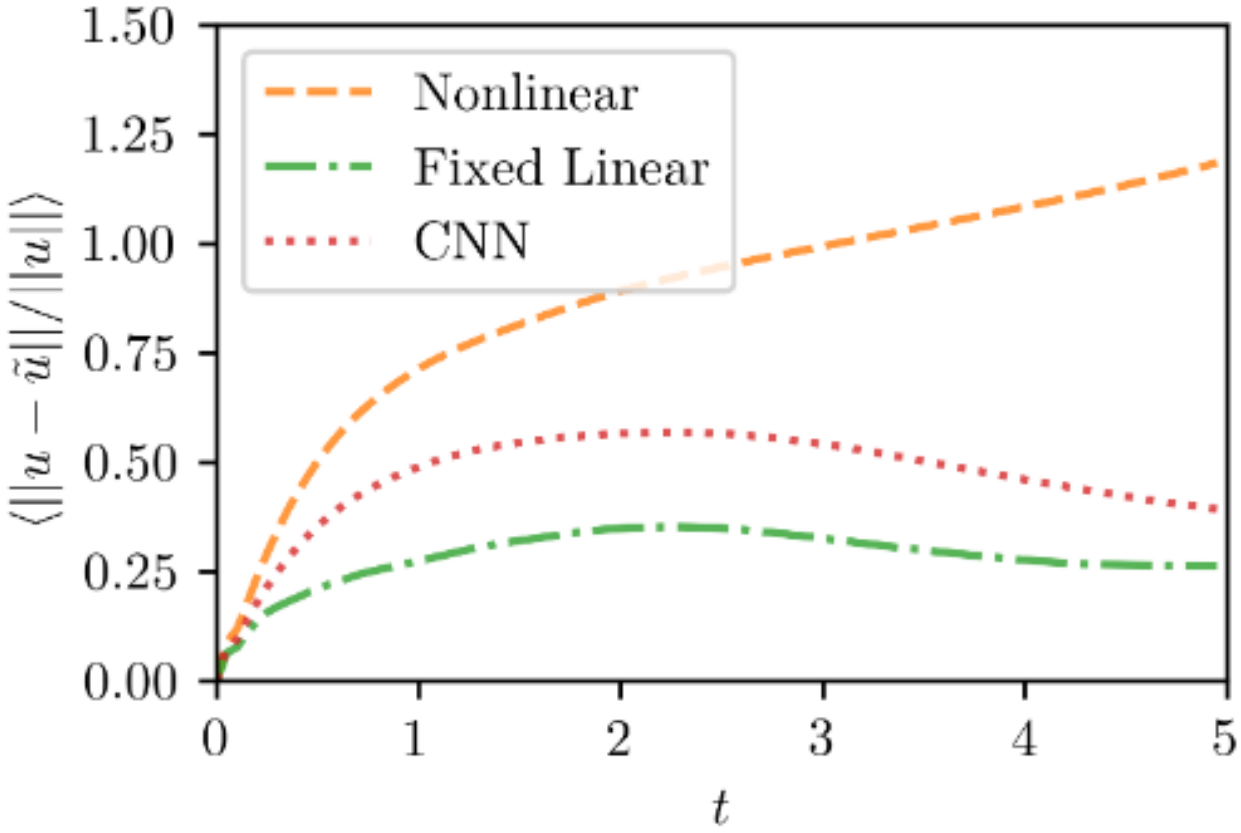


Figure: The stabilized neural ODE outperforms the standard neural ODE for learning the viscous Burgers equations - confirmed for an ensemble of test predictions.

Example: Learning the viscous Burgers equations

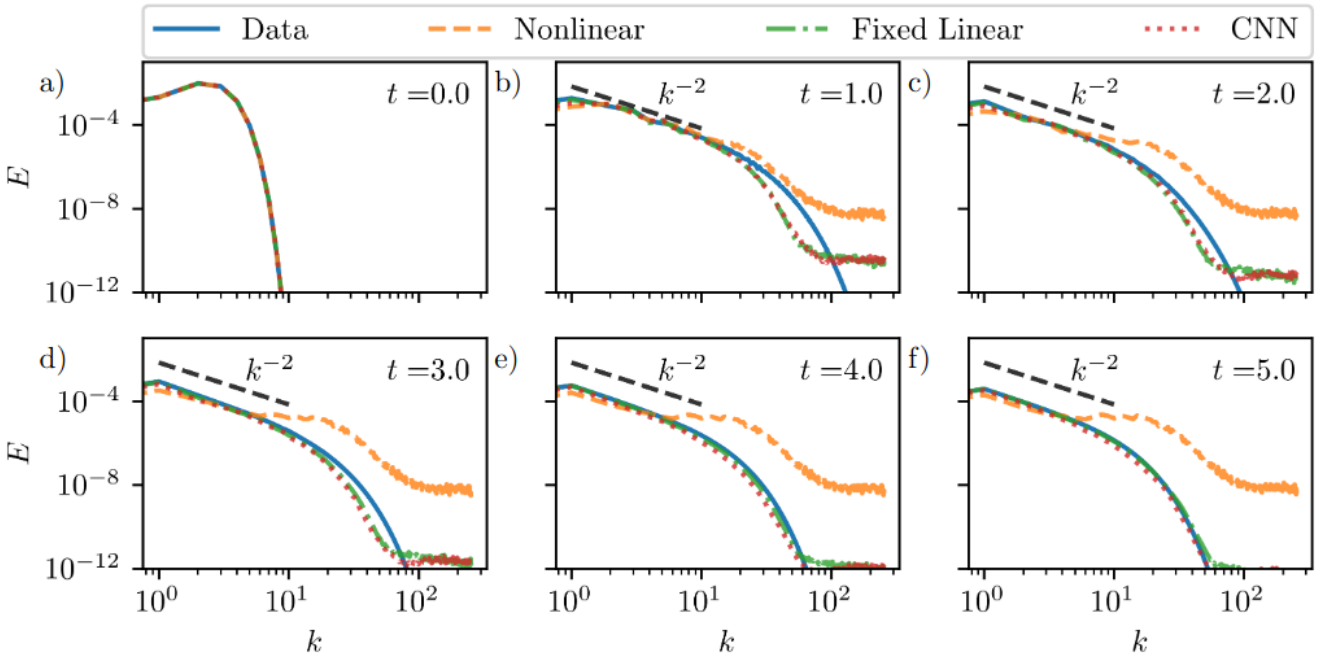


Figure: The stabilized neural ODE outperforms the standard neural ODE for learning the viscous Burgers equations - confirmed for an ensemble of test predictions.

Example: Learning the viscous Burgers equations

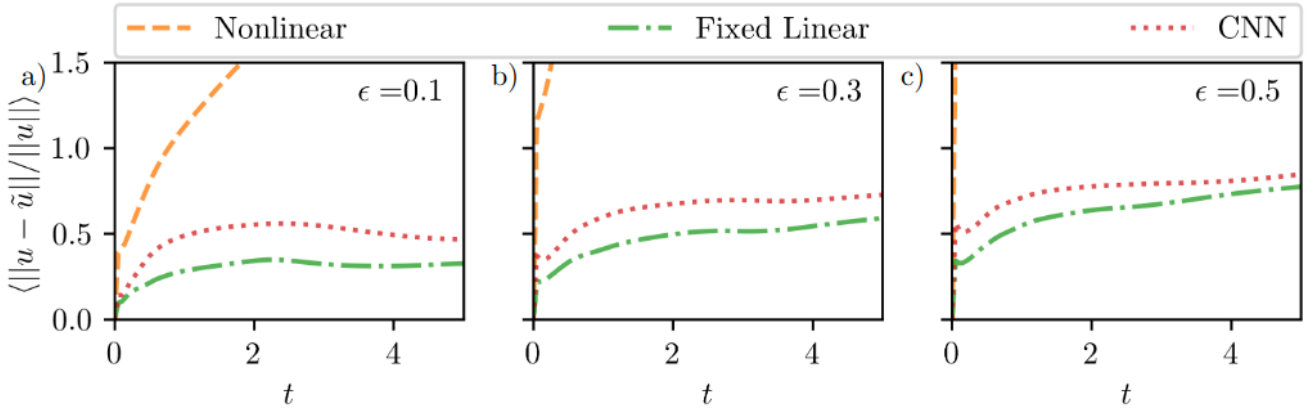


Figure: When adding noise to the initial conditions - the stabilized neural ODE performs more robustly

Example: Learning the viscous Burgers equations

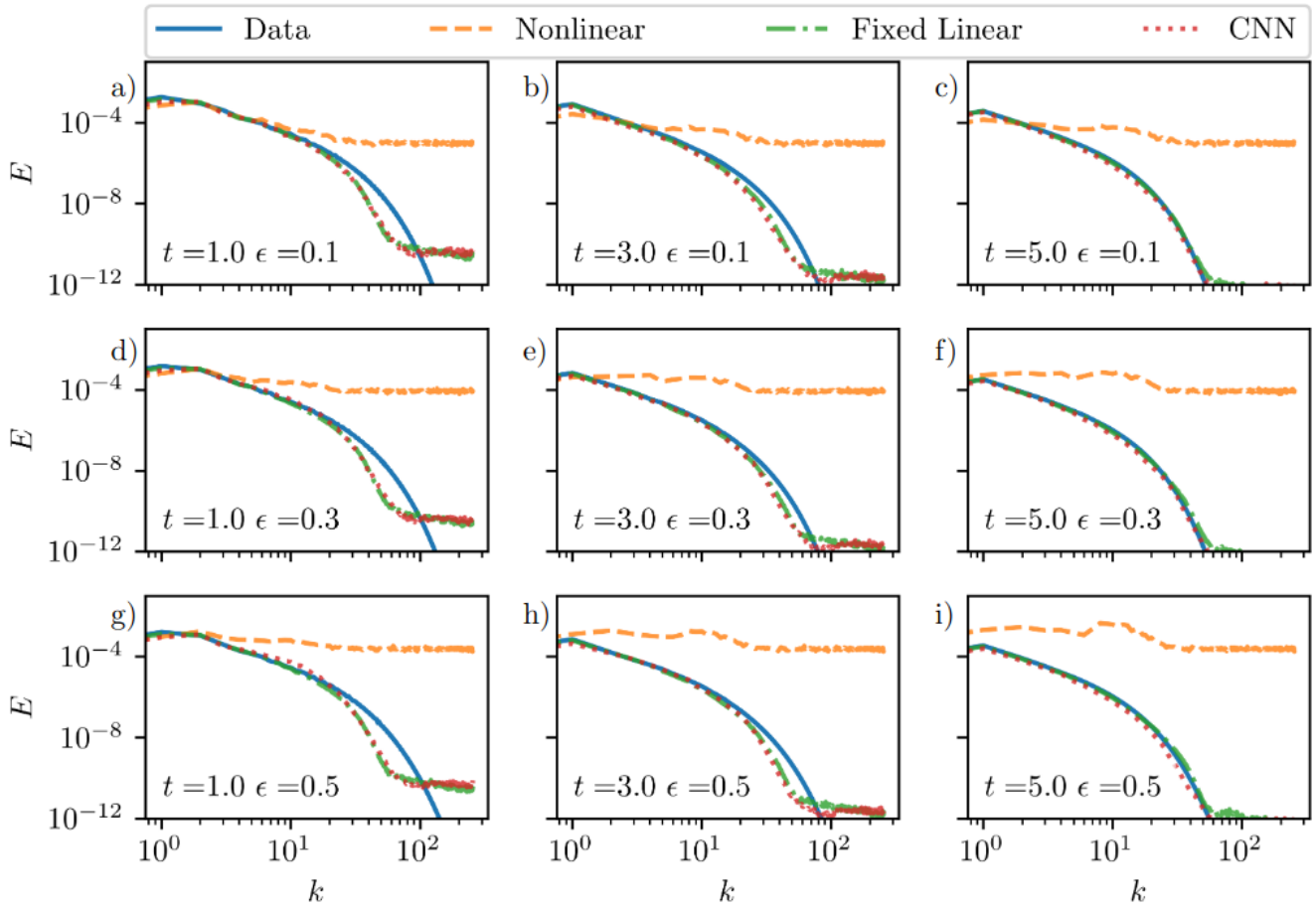


Figure: When adding noise to the initial conditions - the stabilized neural ODE performs more robustly

Under-resolved snapshot data? A preview.

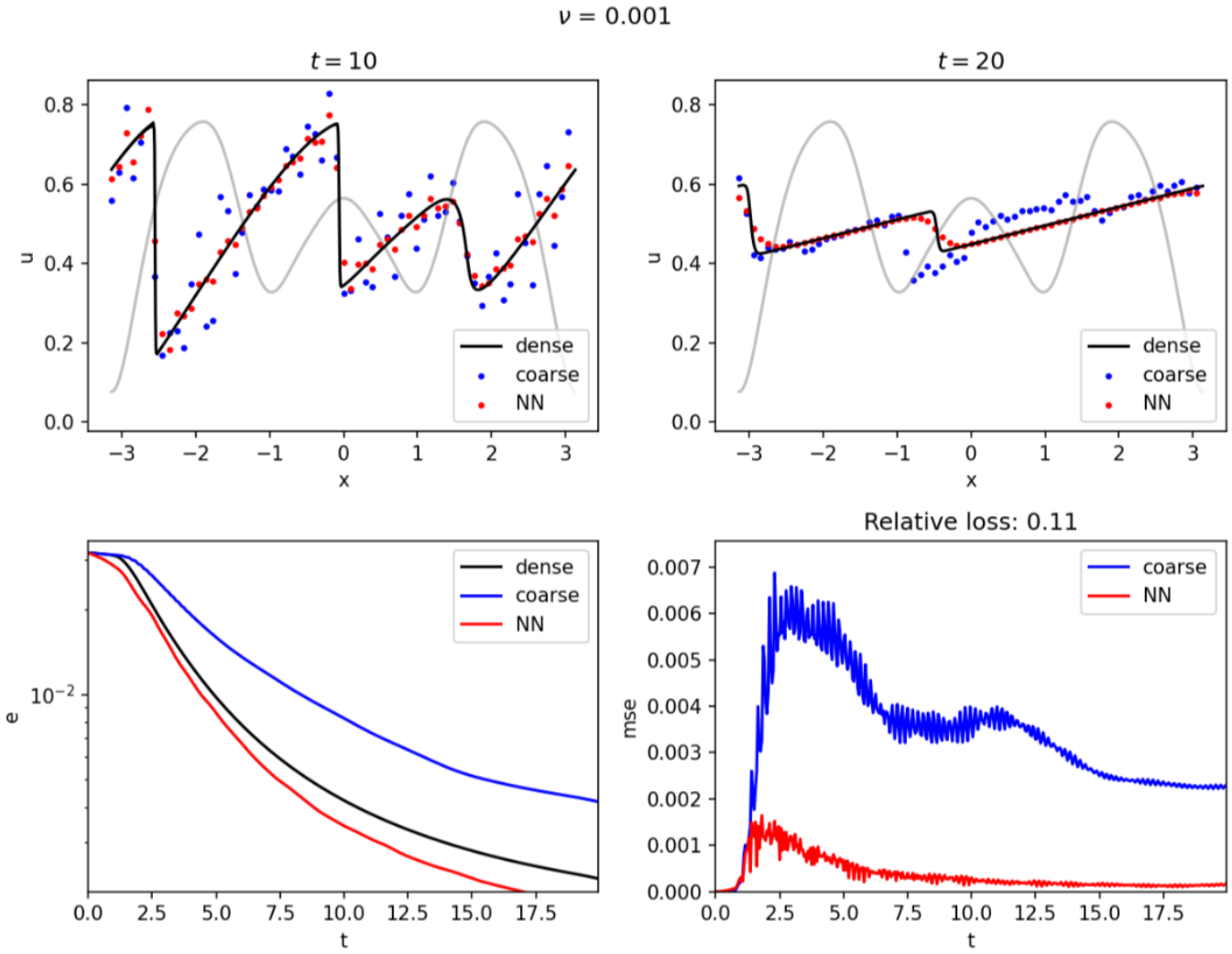


Figure: The stabilized NODE framework is also able to learn a stabilized coarse-grained evolution (i.e., if snapshot resolution is inadequate). Fine-grid 4096 DOF, coarse-grid 64 DOF.

Under-resolved snapshot data? A preview.

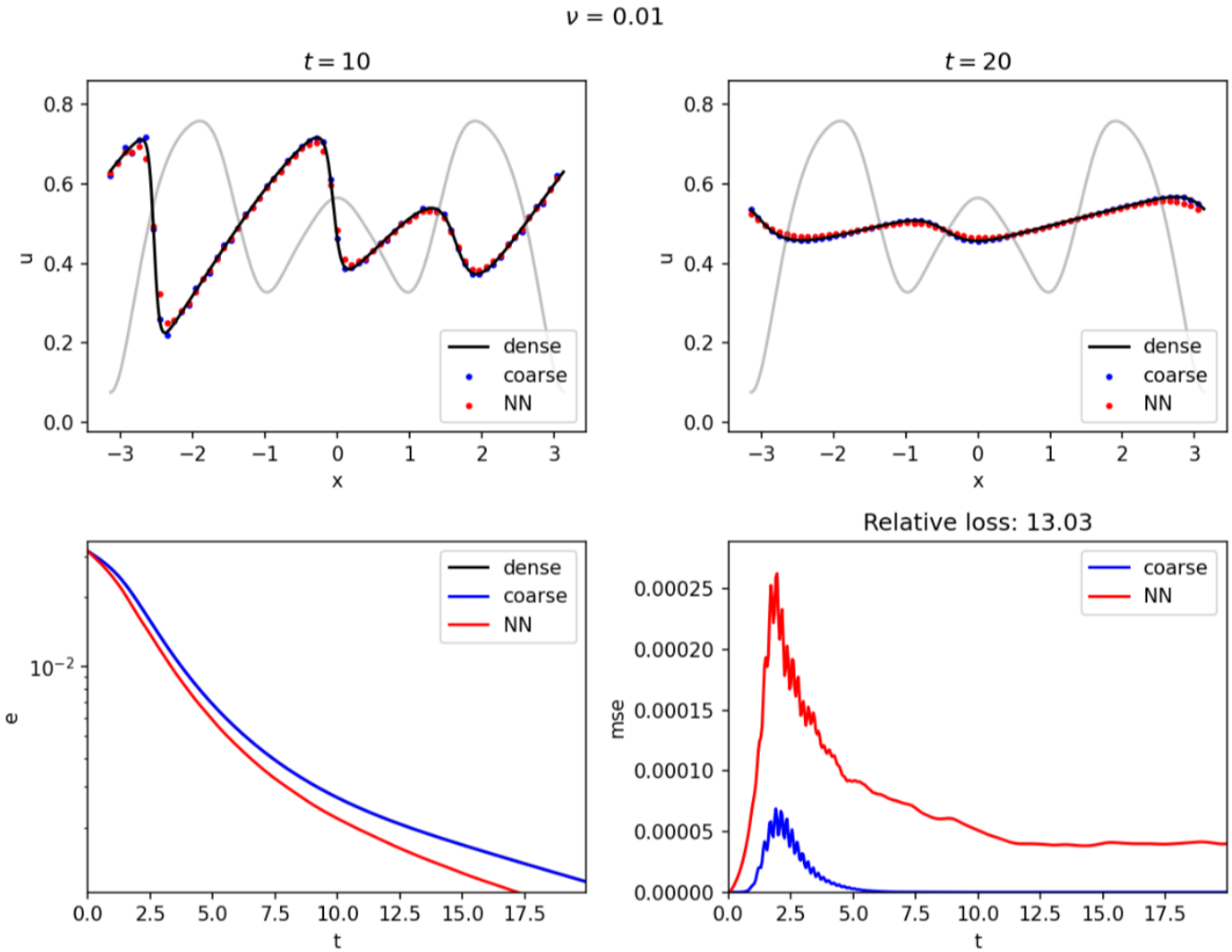


Figure: The stabilized NODE framework is also able to learn a stabilized coarse-grained evolution (i.e., if snapshot resolution is inadequate). Fine-grid 4096 DOF, coarse-grid 64 DOF.

Under-resolved snapshot data? A preview.

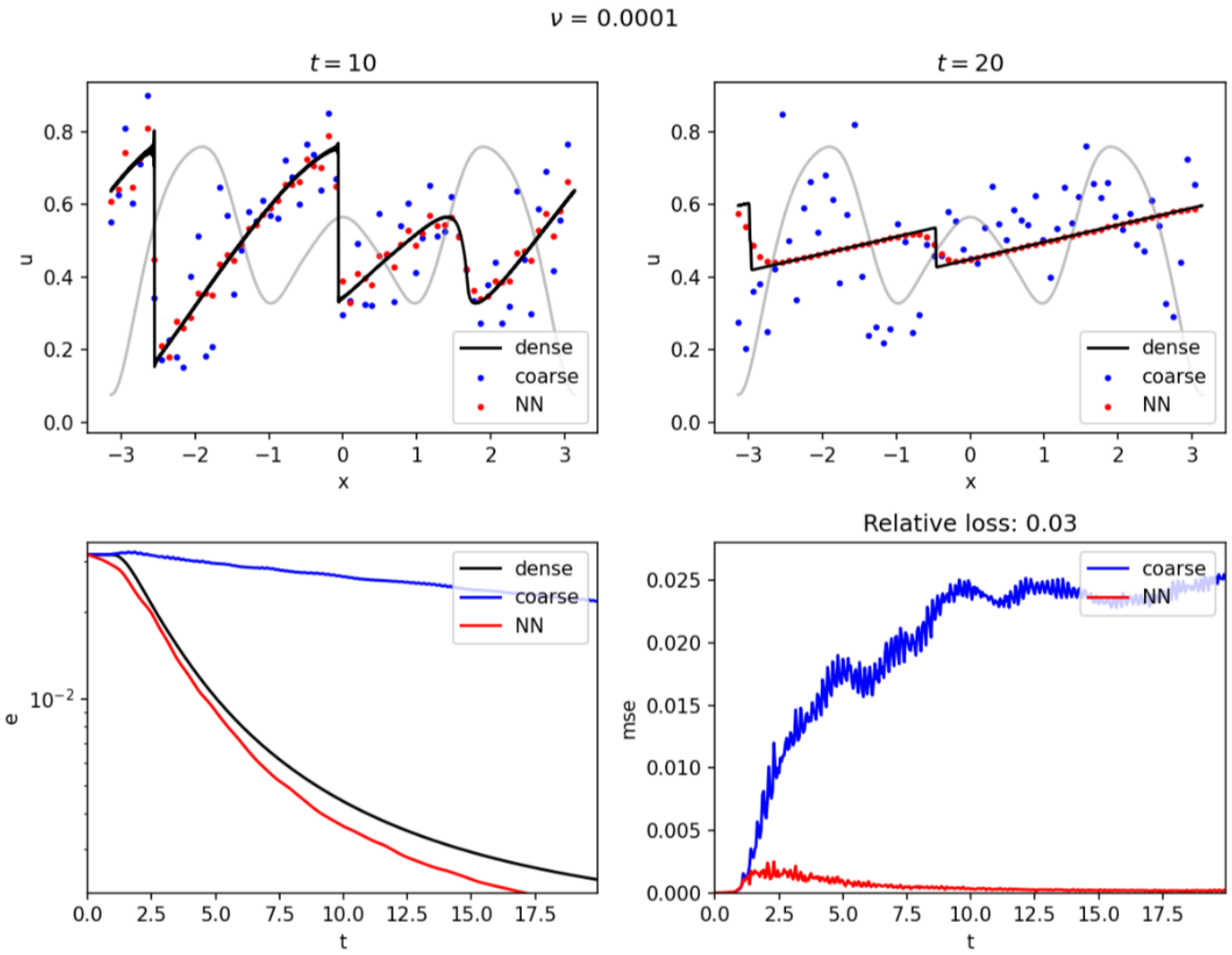


Figure: The stabilized NODE framework is also able to learn a stabilized coarse-grained evolution (i.e., if snapshot resolution is inadequate). Fine-grid 4096 DOF, coarse-grid 64 DOF.

Example: Learning the invariant manifold of the sea-surface temperature

NOAA OI SST V2 High Resolution Dataset

Data on and after 2016 is now v2.1

Brief Description:

- NOAA High-resolution Blended Analysis of Daily SST and Ice. Data is from Sep 1981 and is on a 1/4 deg global grid. *More Details...*

Temporal Coverage:

- Daily values from 1981/09 to present
- **Sea Ice Concentration data is missing for Dec 6th 1987- Jan 10th 1988.**

Spatial Coverage:

- 0.25 degree latitude x 0.25 degree longitude global grid (1440x720).
- 89.875S - 89.875N,0.125E to 359.875E.

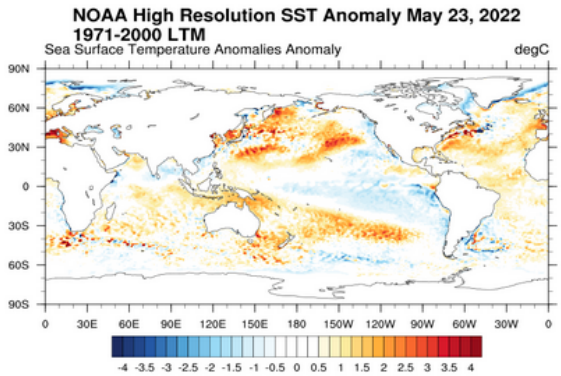


Figure: A sea-surface temperature dataset obtained from satellite and ship observations.

Example: Learning the invariant manifold of the sea-surface temperature

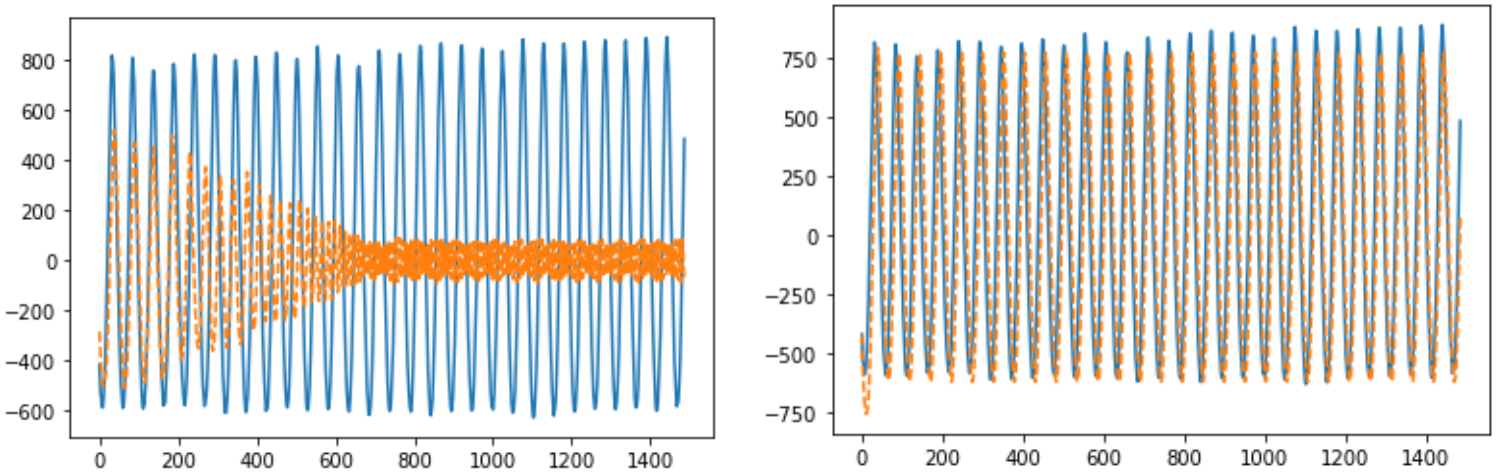


Figure: Test results for learning the POD coefficients of this dataset using regular (left) and stabilized (right) neural ODEs.

Example: Learning the invariant manifold of the sea-surface temperature

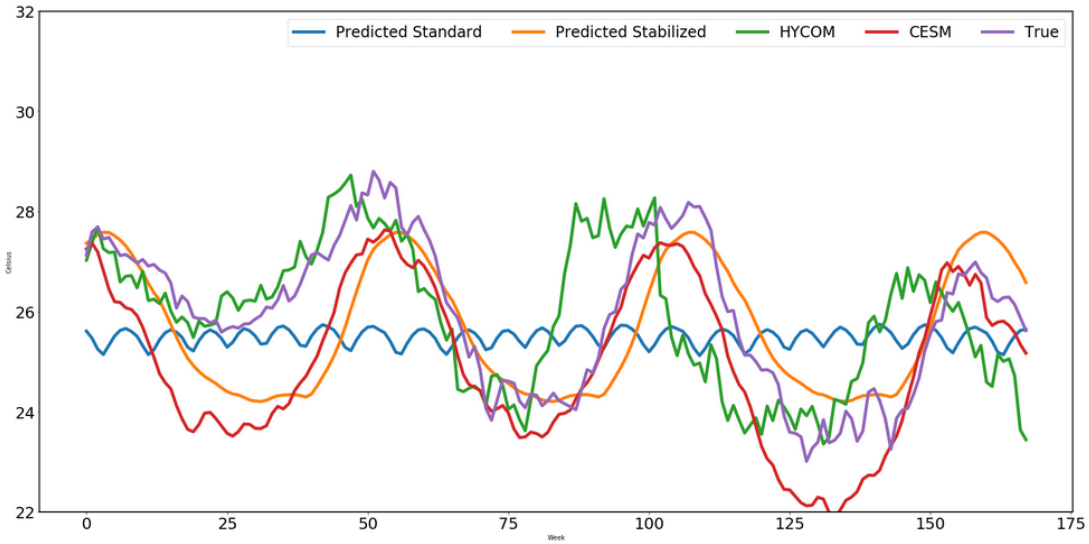


Figure: Preliminary results indicate that predictive dynamics do not decay to fixed point. Probe for solution at 95 degrees latitude and 250 degrees longitude.

Example: Learning the invariant manifold of the sea-surface temperature

RMSE from stabilized NODE (ROM) over weeks 1-8:

1.189, 1.186, 1.180, 1.171, 1.171, 1.167, 1.208, 1.223

Errors from CESM (Climate model) over weeks 1-8:

1.8779, 1.867, 1.830, 1.846, 1.861, 1.869, 1.860, 1.833

Errors from HYCOM (Weather model) over weeks 1-8:

0.998, 0.994, 1.027, 1.039, 1.023, 1.046, 1.033, 1.051

Emulating the sea-surface temperature: Worth it?

Cost to construct our NODE-ROM: 2 node hours of CPU-only laptop, cost to evaluate - negligible.

Cost to evaluate HYCOM: 44800 core hours per forecast day of Cray XC40 system.

Cost to evaluate CESM: 510 million core-hours on Yellowstone, NCAR's high-performance computing resource.

Extensions: Interfacing SST-ROMs as a 'boundary condition' to E3SM atmosphere.

Intermission

Part I - finished.

SciML for time-varying stochastic processes

Generative scientific machine learning methods can also be used to learn surrogates for stochastic dynamical systems.

This vertical is joint work with

- Jonah Botvinick-Greenhouse, Yunan Yang (Cornell University).
- Jinqiao Duan, Yubin Lu (IIT-Chicago).
- Ioannis Kevrekidis (Johns Hopkins University).

Stochastic differential equations

The most common formulation for an Itô SDE is as follows

$$d\boldsymbol{x}(t) = \boldsymbol{\mu}(\boldsymbol{x}(t), t)dt + \boldsymbol{\sigma}(\boldsymbol{x}(t), t)d\boldsymbol{B} \tag{13}$$

where $\boldsymbol{x}(t)$ is a stochastic (real) quantity, $\boldsymbol{B}(t)$ is a Brownian process, and $\boldsymbol{\sigma}(\boldsymbol{x}, t)$ stands for a diffusion coefficient. The PDF of $\boldsymbol{x}$ is then governed by the Fokker-Planck PDE

$$\frac{\partial p(\mathbf{x}, t)}{\partial t} = -\sum_{i=1}^{N} \frac{\partial}{\partial x_i} \left[\mu_i(\mathbf{x}, t)p(\mathbf{x}, t)\right] + \sum_{i=1}^{N}\sum_{j=1}^{N} \frac{\partial^2}{\partial x_i \partial x_j} \left[D_{ij}(\mathbf{x}, t)p(\mathbf{x}, t)\right] \tag{14}$$

with drift vector $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_N)$ and diffusion tensor $\mathbf{D} = \frac{1}{2}\boldsymbol{\sigma}\boldsymbol{\sigma}^\top$, i.e

$$D_{ij}(\mathbf{x}, t) = \frac{1}{2}\sum_{k=1}^{M} \sigma_{ik}(\mathbf{x}, t)\sigma_{jk}(\mathbf{x}, t) \tag{15}$$

and initial condition $p(\boldsymbol{x}, 0) = \delta\left(\boldsymbol{x} - \boldsymbol{x}_0\right)$.

Stochastic differential equations

A nonlocal Itô stochastic differential equation (SDE) may have a non-Gaussian component

$$d\boldsymbol{x}(t) = \boldsymbol{\mu}(\boldsymbol{x}(t), t)dt + \boldsymbol{\sigma}(\boldsymbol{x}(t), t)d\boldsymbol{B} + \hat{\boldsymbol{\sigma}}(\boldsymbol{x}(t), t)d\boldsymbol{L} \tag{16}$$

where $\boldsymbol{L}(t)$ is a Lévy process, and $\hat{\boldsymbol{\sigma}}(\boldsymbol{x}, t)$ is the anomalous diffusion coefficient. The PDF of $\boldsymbol{x}$ is then governed by

$$\frac{\partial p(\mathbf{x}, t)}{\partial t} = -\sum_{i=1}^{N} \frac{\partial}{\partial x_i} \left[\mu_i(\mathbf{x}, t) p(\mathbf{x}, t)\right] + \sum_{i=1}^{N}\sum_{j=1}^{N} \frac{\partial^2}{\partial x_i \partial x_j} \left[D_{ij}(\mathbf{x}, t) p(\mathbf{x}, t)\right] + \int_{\mathbb{R}^d \setminus \{0\}} \tilde{G}(x + y, x, p, \hat{\boldsymbol{\sigma}}(x)) \nu_\alpha(dy) \tag{17}$$

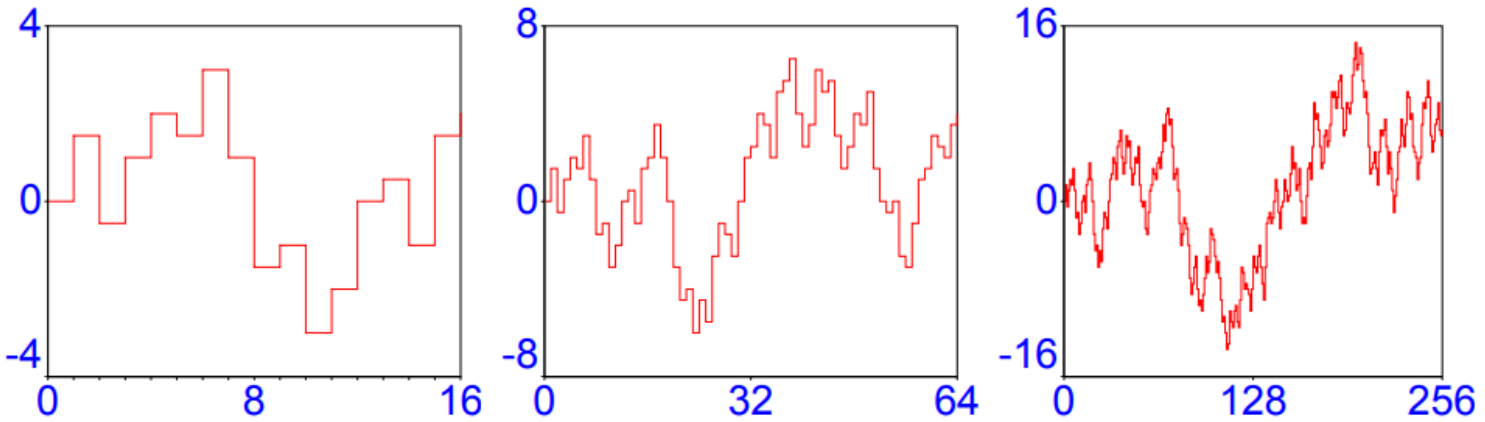
and initial condition $p(x, 0) = \delta(x - x_0)$. Here

$$\nu_\alpha(\mathrm{d}y) = \frac{C_{d,\alpha}}{|y|^{d+\alpha}} \mathrm{d}y, \qquad \text{where } C_{d,\alpha} = \frac{\alpha \Gamma((d+\alpha)/2)}{2^{1-\alpha} \pi^{d/2} \Gamma(1 - \alpha/2)} \tag{18}$$

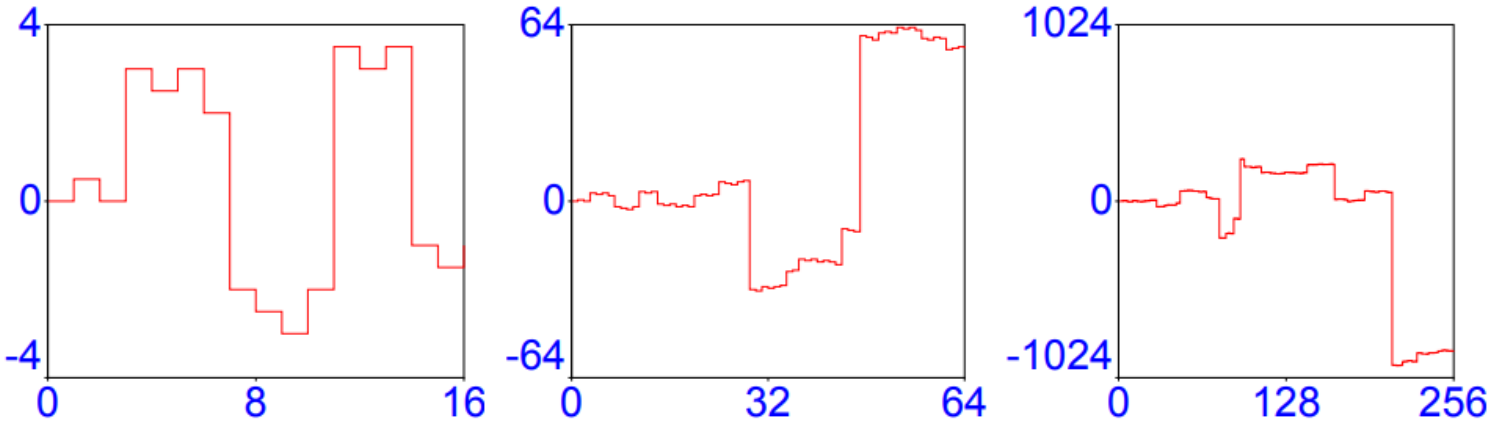
The last term in Equation 26 requires a non-local computation.

Stochastic differential equations

Brownian process



Lévy process



[7] Introduction to Lévy processes, Matthias Winkel, 2004.

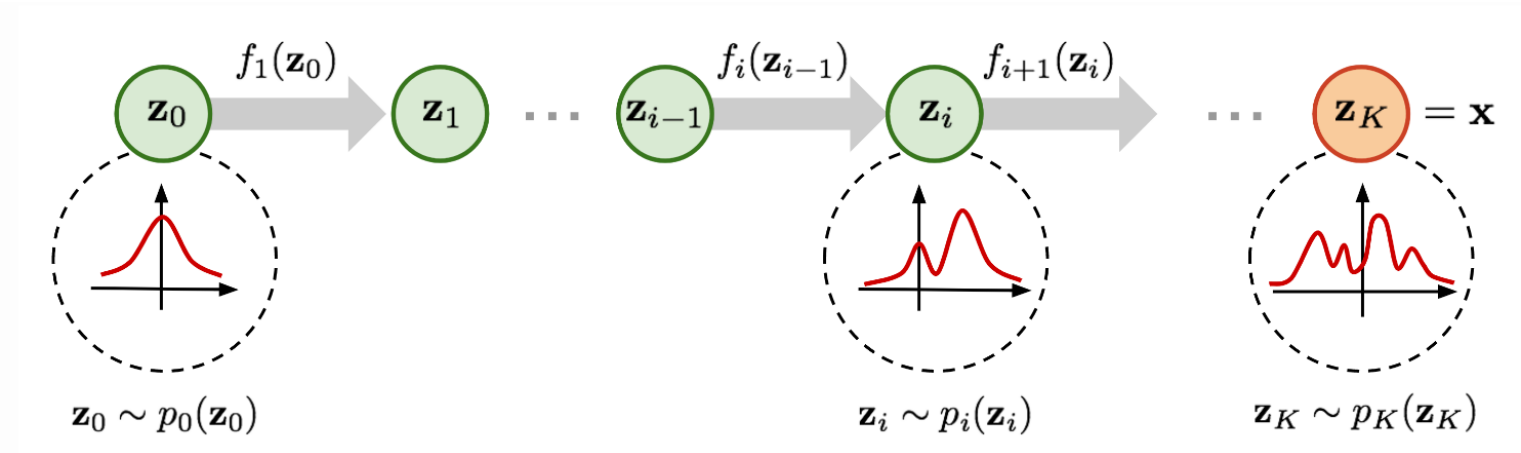
Using SciML for a 'Fokker-Planck surrogate'

Can we learn a time-varying density directly from data? Existing work broadly falls into following themes:

- The use of neural-ODE analogs for the stochastic differential equations (Duvenaud, Ruthotto, Jia).
- Use of generative adversarial networks for learning the sampler of these densities (Karniadakis).
- Direct solution of a 'Fokker-Planck' physics-informed neural network.

Normalizing flows

Normalizing flows provide a general way to express probability distributions, only requiring a base distribution and a series of bijective transformations. Given $\boldsymbol{z}$ as a $D-$ dimensional real vector sampled from a prior (or latent) density $p_{\boldsymbol{z}}(\boldsymbol{z})$.



[9] https://lilianweng.github.io/lil-log/2018/10/13/flow-based-deep-generative-models.html

Normalizing flows

Given $\boldsymbol{z}$ as a $D-$ dimensional real vector sampled from a prior (or latent) density $p_{\boldsymbol{z}}(\boldsymbol{z})$., we find a transformation T such that:

$$\boldsymbol{z} = T(\boldsymbol{x}), \quad \text{where} \quad \boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z}) \tag{19}$$

When the transformation T is invertible and both T and T^{-1} are differentiable, the density of $\boldsymbol{x}$ is well-defined and can be obtained by a change of variables:

$$p_{\boldsymbol{x}}(\boldsymbol{x}) = p_{\boldsymbol{z}}(T(\boldsymbol{x}))\left|\det J_T(\boldsymbol{x})\right|, \quad \text{where} \quad \boldsymbol{z} = T(\boldsymbol{x}) \tag{20}$$

and $J_T(\boldsymbol{z})$ is the Jacobian of T. Consider a set of samples $\{\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, x_n\}$ taken from an unknown distribution $p_{\boldsymbol{x}}(\boldsymbol{x})$, we can minimize the negative log-likelihood,

$$\mathcal{L} = -\sum_{i=1}^{n} \log p_{\boldsymbol{z}}\left(T\left(\boldsymbol{x}_i\right)\right) + \log\left|\det J_T(\boldsymbol{x})\right|_{\boldsymbol{x}=\boldsymbol{x}_i}. \tag{21}$$

Normalizing flows

Given two invertible and differentiable transformations T_1 and T_2, we have the following properties:

$$\begin{aligned} (T_2 \circ T_1)^{-1} &= T_1^{-1} \circ T_2^{-1} \\ \det J_{T_2 \circ T_1(\boldsymbol{z})} &= \det J_{T_2}\left(T_1(\boldsymbol{z})\right) \cdot \det J_{T_1}(\boldsymbol{z}). \end{aligned} \tag{22}$$

Consequently, we can build complex transformations by composing K simpler transformations, i.e., $T = T_K \circ T_{K-1} \circ \cdots \circ T_1$. Each T_K transforms $\boldsymbol{z}_{K-1}$ into $\boldsymbol{z}_K$, assuming

$$\boldsymbol{z}_0 = \boldsymbol{x} \text{ and } \boldsymbol{z}_K = \boldsymbol{z} \tag{23}$$

Consider a set of samples $\{\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, x_n\}$ taken from an unknown distribution $p_{\boldsymbol{x}}(\boldsymbol{x})$, we can minimize the negative log-likelihood,

$$\mathcal{L} = -\sum_{i=1}^{n} \log p_{\boldsymbol{z}}\left(T\left(\boldsymbol{x}_i\right)\right) + \log \left|\det J_T(\boldsymbol{x})\right|_{\boldsymbol{x}=\boldsymbol{x}_i}. \tag{24}$$

Multivariate temporal normalizing flows (TNFs)

The hard part about constructing a normalizing flow is proposing a functional form for T. In this study, we propose

$$\begin{aligned} z_{1:d} &= x_{1:d} \\ z_{d+1:D} &= x_{d+1:D} \odot e^{\mu(\boldsymbol{x}_{1:d}, t)} + \nu\left(\boldsymbol{x}_{1:d}, t\right), \end{aligned} \tag{25}$$

where the notation $\odot$ is Hadamard product, μ and ν are neural networks. Here d is a hyperparameter and $\boldsymbol{x}_{1:d} = (x_1, x_2, \ldots, x_d)$. The determinant of the Jacobian becomes

$$\det J_T = e^{\mu(\boldsymbol{x}_{1:d}, t)} \tag{26}$$

Members of the audience who are familiar with normalizing flows will recognize this as a 'time-varying' version of Dinh et als., RealNVP.

TNF Example

Consider a two-dimensional stochastic differential equation with pure jump Lévy motion, (i.e., only anomalous diffusion)

$$\begin{aligned} dX_1(t) &= \left(8X_1(t) - X_1^3(t)\right) dt + dL_1^\alpha(t) \\ dX_2(t) &= -\left(8X_2(t) - X_2^3(t)\right) dt + dL_2^\alpha(t) \\ (X_1(0), X_2(0)) &\sim \mathcal{N}\left(0, I_{2\times 2}\right) \end{aligned} \tag{27}$$

where L_1^α and L_2^α are two disjoint independent scalar real-value $\alpha-$ stable Lévy motion with triple $(0, 0, \rho)$. Sampling a dataset $\mathcal{D}$ from the solution of the SDE as our training data, where $t_1 = 0$, $t_m = 0.95$, $\Delta t = 0.05$, $m = 20$ and $n = 500$.

This example leads to a multimodal distribution evolution.

TNF Example

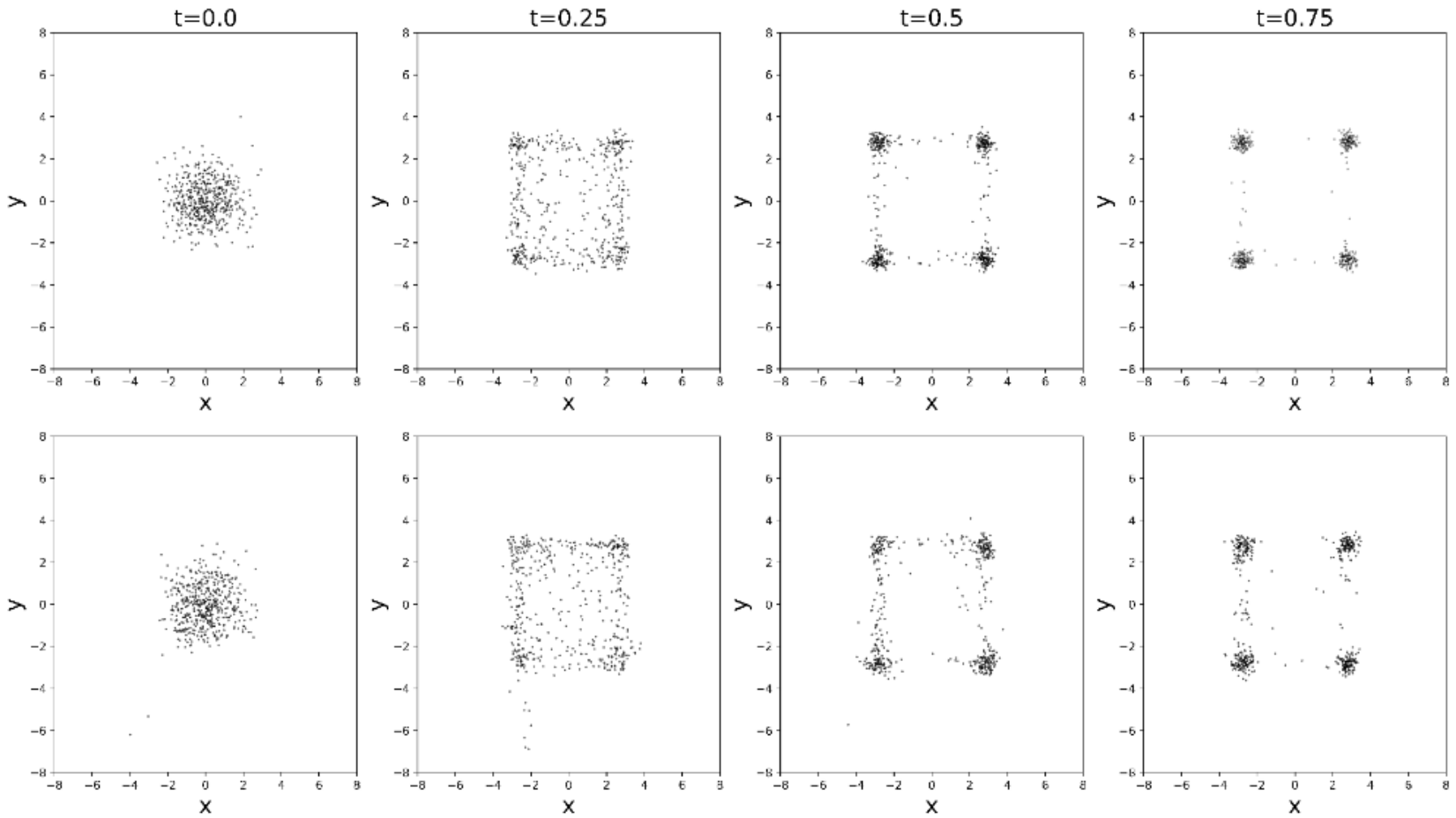


Figure: True - top and predicted - bottom. For different locations in time.

Lu, RM, Gao, Dietrich, Kevrekidis, and Duan, 2021. Chaos, 2021.

Half moons example

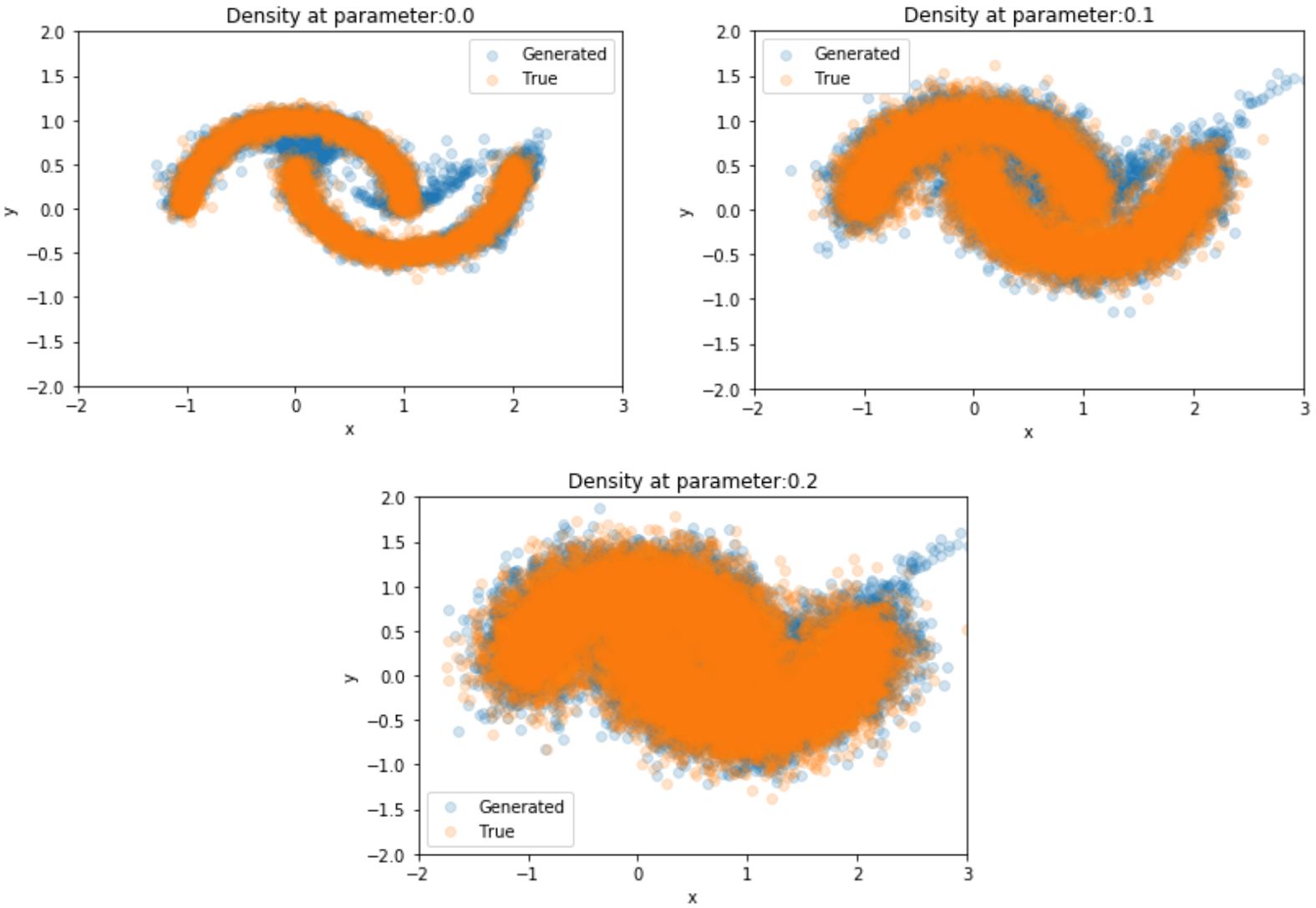


Figure: Learning the half-moon densities parameterized by noise.

Half moons example

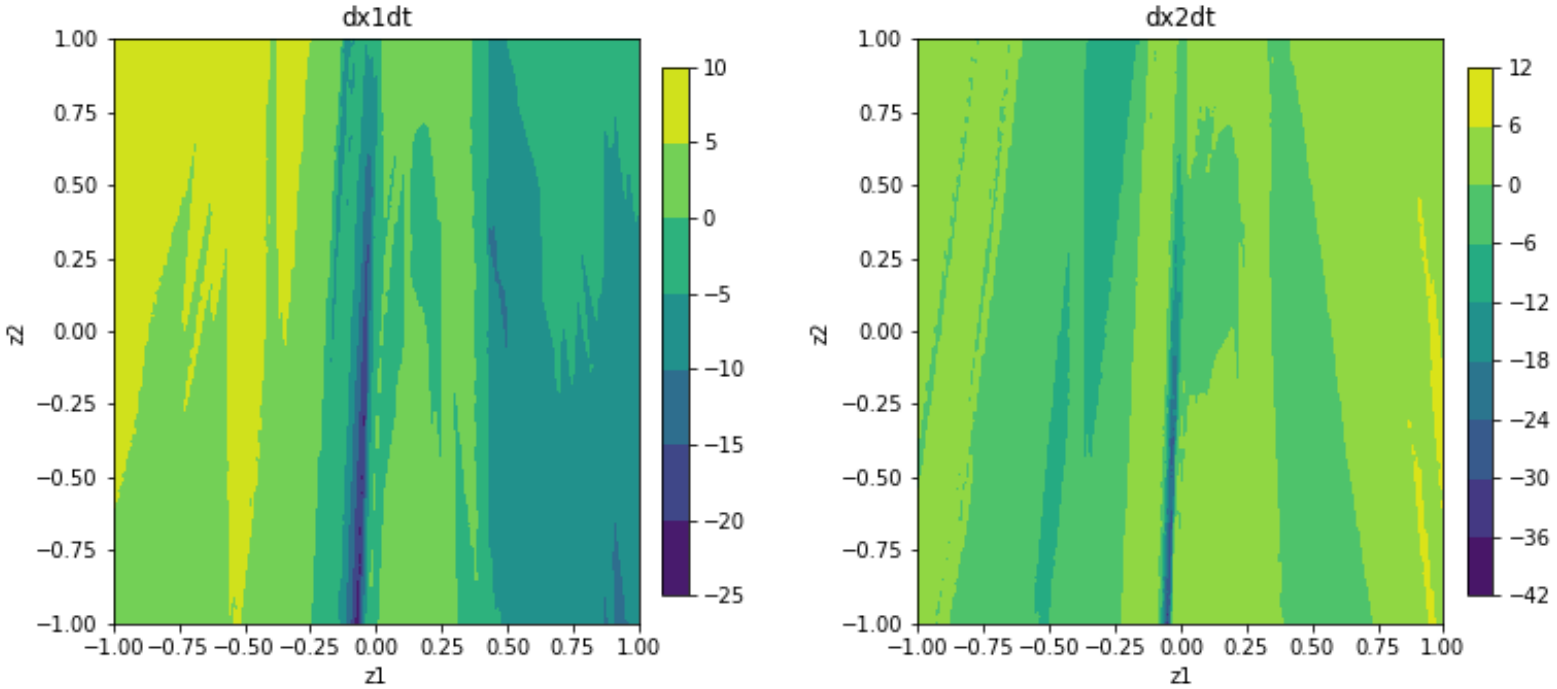


Figure: Extracting the 'learned' half-moon flow map for no noise. Connection to discovering physical systems? Work underway.

$$\frac{d\mathbf{x}}{dt} = \frac{\partial T^{-1}}{\partial t}(\mathbf{z}, t) \tag{28}$$

Use case: Attractor reconstruction

We can construct partially observed periodic systems, given training data for full state observations. Given a time delayed embedding

$$\Psi(\mathbf{x}(t); \tau, d) := \big(\psi(\mathbf{x}(t), \psi(\mathbf{x}(t-\tau)), \ldots, \psi(\mathbf{x}(t-(d-1)\tau))\big), \tag{29}$$

for a suitable embedding dimension of $d \geq 1$ and delay parameter $\tau > 0$.
By Takens' Theorem, we have that $\Psi : \mathcal{M} \to \mathcal{M}_\psi$ is a diffeomorphism - where $\mathcal{M}$ is the manifold generated by the true dynamics $\mathbf{x}(t)$ and by $\mathcal{M}_\psi$ is the manifold generated by the delayed observable. Here $\psi(\mathbf{x}_t)$ is a partial observable of the full-state. We want to learn $\Psi^{-1} : \mathcal{M}_\psi \to \mathcal{M}$.

Use case: Attractor reconstruction

Algorithm 5: Deterministic Attractor Reconstruction with Conditional Normalizing Flows

Input: Full state measurements $\{\mathbf{x}(t_i)\}_{i=1}^{K}$, where $t_i \leq T$ and the partial observations $\{\psi(\mathbf{x}(t_i))\}_{i=1}^{K+M}$

Data Preparation:

Select an embedding dimension d and a time-delay τ.
Compute the points $\Psi(\mathbf{x}(t_i); \tau, d)$.
Accumulate the noise-inflated observations $\mathcal{O}$.

Training:

For $\ell = 1, \ldots, L$
Form the r-point random sample $R := \{\mathbf{x}(t_i) + \epsilon_{ij}\}$.
Compute the log-likelihood that $\Phi(\Psi(\mathbf{x}(t_i)); \theta)^{-1}(R) \sim \mathcal{N}(0, I)$.
Update θ.
Return

Testing:

For $m = K + 1 \ldots, K + M$
Sample s points $\{p_i\}_{i=1}^{s} \sim \mathcal{N}(0, I)$
Evaluate $P := \Phi(\Psi(\mathbf{x}(t_m)); \theta) \# \{p_i\}_{i=1}^{s}$
Assign $\mathbf{x}(t_m) \leftarrow \text{mean}(P)$.
Return

Output: The reconstructed full states $\{\mathbf{x}(t_i)\}_{i=1}^{K+M}$.

Figure: Algorithm for state reconstructions using conditional normalizing flows.

Use case: Attractor reconstruction

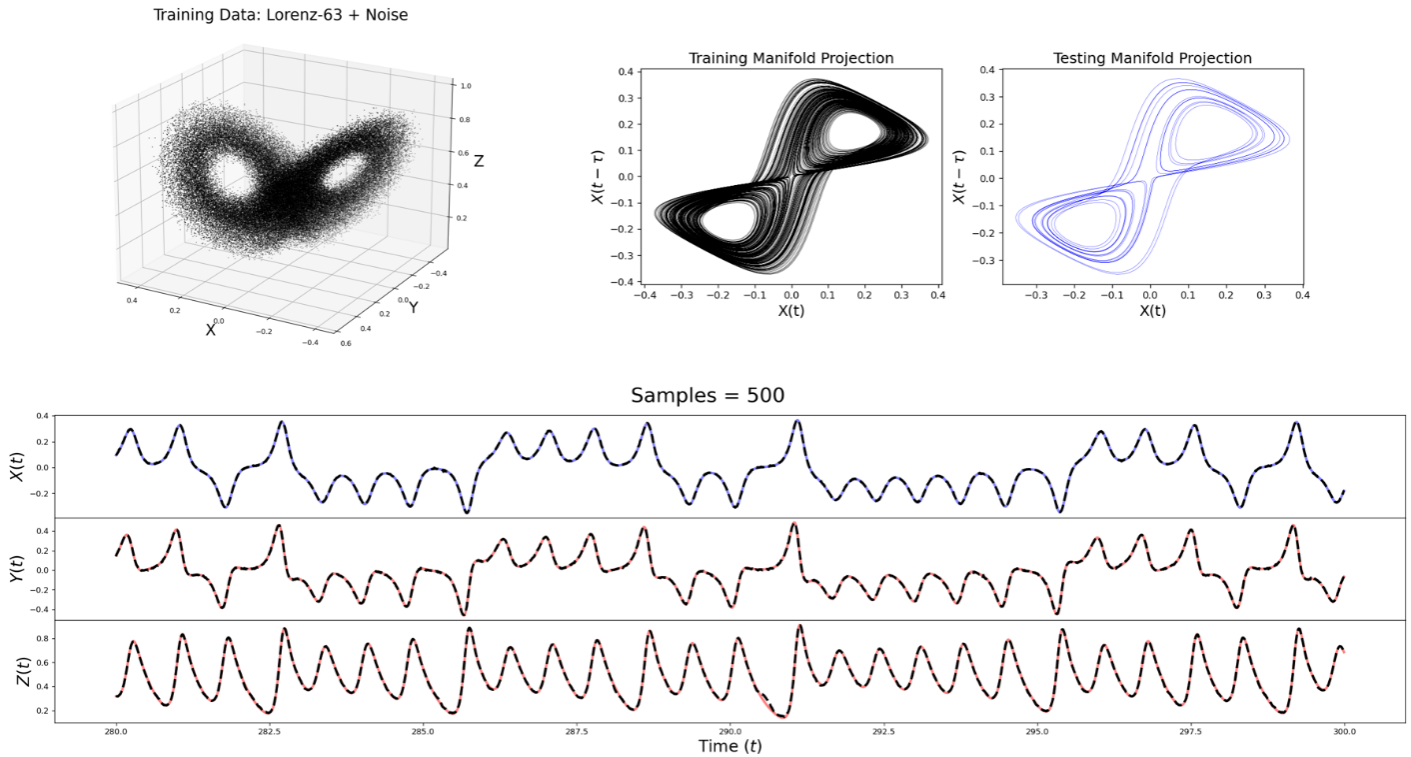


Figure: Reconstructing the Lorenz-63 attractor.

Note that the current formulation of this approach assumes partial observations of the current timestep. Extensions to a pure 'forecast' pardigm underway.

[10] Botvinick-Greenhouse, Yang, and Maulik, in-preparation.

Use case: Brownian-SDE identification

The generator of an SDE can be approximated by a normalizing flow:

$$Af(x) = \lim_{\Delta t \to 0} \frac{1}{\Delta t} \left[\mathbb{E}^x_{t+\Delta t}(f(\mathbf{x}(t)) - f(\mathbf{x}) \right] \tag{30}$$

$$\approx \frac{1}{\Delta t} \left[\sum_{i=1}^{n} f(x_i) p_{\mathcal{X}}(x_i, \Delta t) - f(\mathbf{x}) \right], \tag{31}$$

[11] Lu et al., Physica D. 417, 132830, 2021

Use case: Brownian-SDE identification

Recall, also, that the generator of the SDE is given by

$$Af(\boldsymbol{x}(t)) = \boldsymbol{\mu}(\boldsymbol{x}(t), t) \cdot \nabla f(\boldsymbol{x}(t)) + \frac{1}{2} \operatorname{Tr}\left(a(\boldsymbol{x}(t)) H_f(\boldsymbol{x}(t))\right) \tag{32}$$

where $a = \boldsymbol{\sigma}\boldsymbol{\sigma}^T$. After some algebra, the SDE extraction boils down to setting $f(\boldsymbol{x}(t)) = \boldsymbol{x}$ which gives us

$$\begin{aligned} Af(\boldsymbol{x}(t)) &= \boldsymbol{\mu}(\boldsymbol{x}(t)) \cdot \nabla f + \frac{1}{2} \operatorname{Tr}\left(a(\boldsymbol{x}(t)) H_f(\boldsymbol{x}(t))\right) \\ &= \boldsymbol{\mu}(\boldsymbol{x}(t)) \cdot \mathbf{e}_i + \frac{1}{2} \operatorname{Tr}(a(\boldsymbol{x}(t))\mathbf{0}) \\ &= \boldsymbol{\mu}_i(\boldsymbol{x}(t)) \cdot s \end{aligned} \tag{33}$$

where $\boldsymbol{\mu}_i$ obtained on some suitable discretization of $\boldsymbol{x}(t)$.

[11] Lu et al., Physica D. 417, 132830, 2021

Use case: Brownian-SDE identification

Furthermore we can set $f(\boldsymbol{x}(t)) = \boldsymbol{x}(t)^2$ to get

$$\begin{aligned} Af(\boldsymbol{x}(t)) &= \boldsymbol{\mu}(\boldsymbol{x}(t)) \cdot \nabla f(\boldsymbol{x}(t)) + \frac{1}{2} \operatorname{Tr}\left(f(\boldsymbol{x}(t)) H_f(\boldsymbol{x}(t))\right) \\ &= \boldsymbol{\mu}(\boldsymbol{x}(t)) \cdot 2\boldsymbol{x}(t)_i \mathbf{e}_i + \frac{1}{2} \operatorname{Tr}\left(a(\boldsymbol{x}(t)) H_{f(\boldsymbol{x}(t))}(\boldsymbol{x}(t))\right) \\ &= 2\boldsymbol{x}(t)_i \boldsymbol{\mu}_i(\boldsymbol{x}(t)) + a_{ii}(\boldsymbol{x}(t)) \end{aligned} \tag{34}$$

This gives us diagonal terms of $\boldsymbol{\sigma}(\boldsymbol{x}(t))$ (and the problem ends here if $\boldsymbol{\sigma}(\boldsymbol{x}(t))\boldsymbol{\sigma}(\boldsymbol{x}(t))^T$ is diagonal). In case they're not - we need to set $f(\boldsymbol{x}(t)) = x_i(t)x_j(t)$ where $i \neq j$ to obtain off-diagonal terms.

[11] Lu et al., Physica D. 417, 132830, 2021

Use case: Brownian-SDE identification

We begin with the one-dimensional double-well system (with Brownian motion)

$$\mathrm{d}X_t = \left[-X_t^3 + 4X_t\right]\mathrm{d}t + \left[0.2X_t^3 + 0.6\right]\mathrm{d}B_t \tag{35}$$

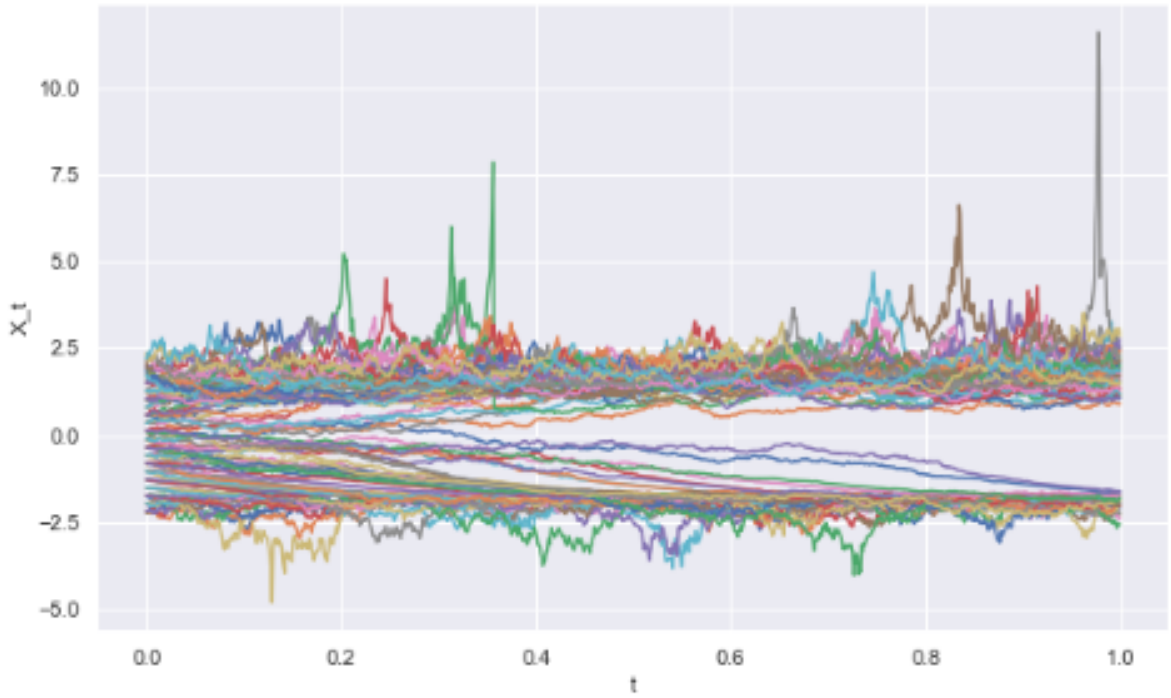


Figure: Double well system

Use case: Brownian-SDE identification

We obtain drift and diffusion approximations that are similar to the SDE terms - we may couple this to a sparse regression to obtain functional forms

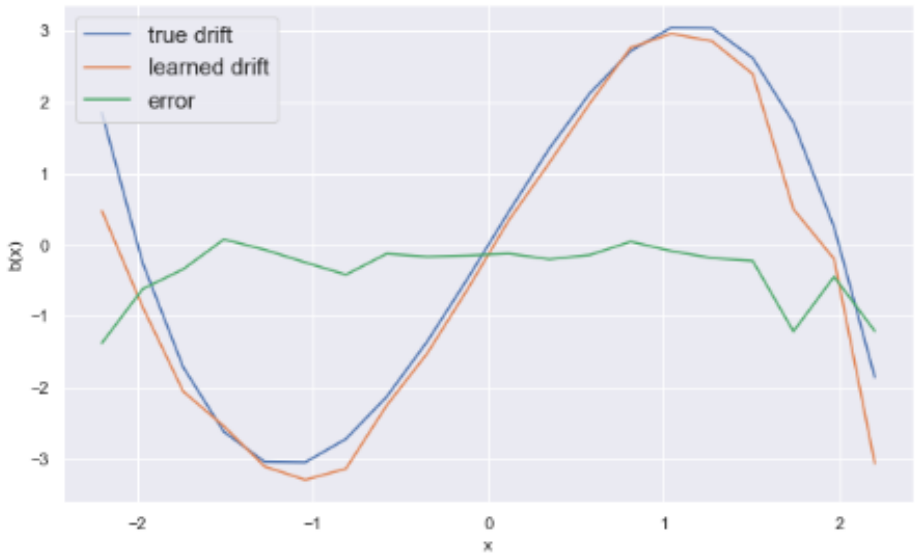


Figure: Learning the drift

Use case: Brownian-SDE identification

We obtain drift and diffusion approximations that are similar to the SDE terms - we may couple this to a sparse regression to obtain functional forms

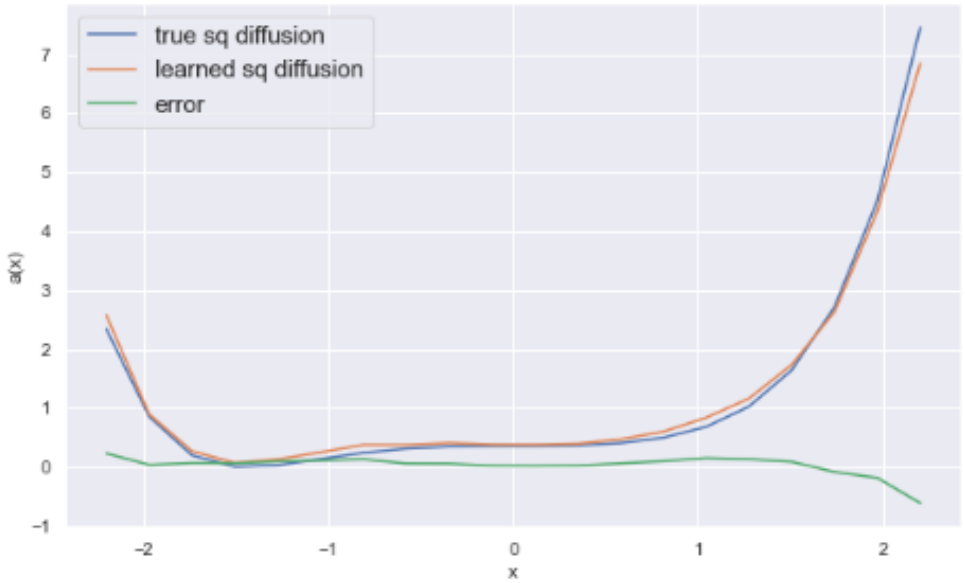


Figure: Learning the diffusion

Big picture applications

Applications:

- Elegant particle/continuum coupling for various multiscale, multiphysics problems.
- Subgrid scale modeling for complex systems.
- Synthetic boundary condition generators (inflow turbulence).

Acknowledgements

U.S. Department of Energy, Advanced Scientific Computing Research
(DOE-FOA2493: Data intensive scientific machine learning, PI-Maulik)

U.S. Department of Energy, Advanced Scientific Computing Research
(SCIDAC-RAPIDS Institute, PI-Ross)

Argonne Leadership Computing Facility
For compute resources and Margaret Butler Fellowship
(DE-AC02-06CH11357)

Thanks for listening!

`romit-maulik.github.io`

References I

Lawrence Sirovich.

Turbulence and the dynamics of coherent structures. i. coherent structures.
Quarterly of applied mathematics, 45(3):561–571, 1987.

AE Deane, IG Kevrekidis, G Em Karniadakis, and SA0746 Orszag.

Low-dimensional models for complex geometry flows: application to grooved channels and circular cylinders.
Physics of Fluids A: Fluid Dynamics, 3(10):2337–2354, 1991.

R Rico-Martinez, K Krischer, IG Kevrekidis, MC Kube, and JL Hudson.

Discrete-vs. continuous-time nonlinear signal processing of cu electrodissolution data.
Chemical Engineering Communications, 118(1):25–48, 1992.

Ramiro Rico-Martinez and Ioannis G Kevrekidis.

Continuous time modeling of nonlinear systems: A neural network-based approach.
In *IEEE International Conference on Neural Networks*, pages 1522–1525. IEEE, 1993.

Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud.

Neural ordinary differential equations.
arXiv preprint arXiv:1806.07366, 2018.

Sepp Hochreiter and Jürgen Schmidhuber.

Long short-term memory.
Neural computation, 9(8):1735–1780, 1997.

Alex Sherstinsky.

Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network.
Physica D: Nonlinear Phenomena, 404:132306, 2020.

References II

Romit Maulik, Arvind Mohan, Bethany Lusch, Sandeep Madireddy, Prasanna Balaprakash, and Daniel Livescu.

Time-series learning of latent-space dynamics for reduced-order model closure.

Physica D: Nonlinear Phenomena, 405:132368, 2020.

Michael S Jolly, IG Kevrekidis, and Edriss S Titi.

Approximate inertial manifolds for the kuramoto-sivashinsky equation: analysis and computations.

Physica D: Nonlinear Phenomena, 44(1-2):38–60, 1990.

Sebastian Mika, Bernhard Schölkopf, Alexander J Smola, Klaus-Robert Müller, Matthias Scholz, and Gunnar Rätsch.

Kernel pca and de-noising in feature spaces.

In *NIPS*, volume 11, pages 536–542, 1998.

Ronald R Coifman and Stéphane Lafon.

Diffusion maps.

Applied and computational harmonic analysis, 21(1):5–30, 2006.

Mark A Kramer.

Nonlinear principal component analysis using autoassociative neural networks.

AIChE journal, 37(2):233–243, 1991.

Romit Maulik, Romain Egele, Bethany Lusch, and Prasanna Balaprakash.

Recurrent neural network architecture search for geophysical emulation.

In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–14. IEEE, 2020.

David Salinas, Valentin Flunkert, Jan Gasthaus, and Tim Januschowski.

Deepar: Probabilistic forecasting with autoregressive recurrent networks.

International Journal of Forecasting, 36(3):1181–1191, 2020.