Structure-preserving learning of embedded closure models

Benjamin Sanderse, Syver Agdestein, Toby van Gastelen, Henrik Rosenberger, Hugo Melchers

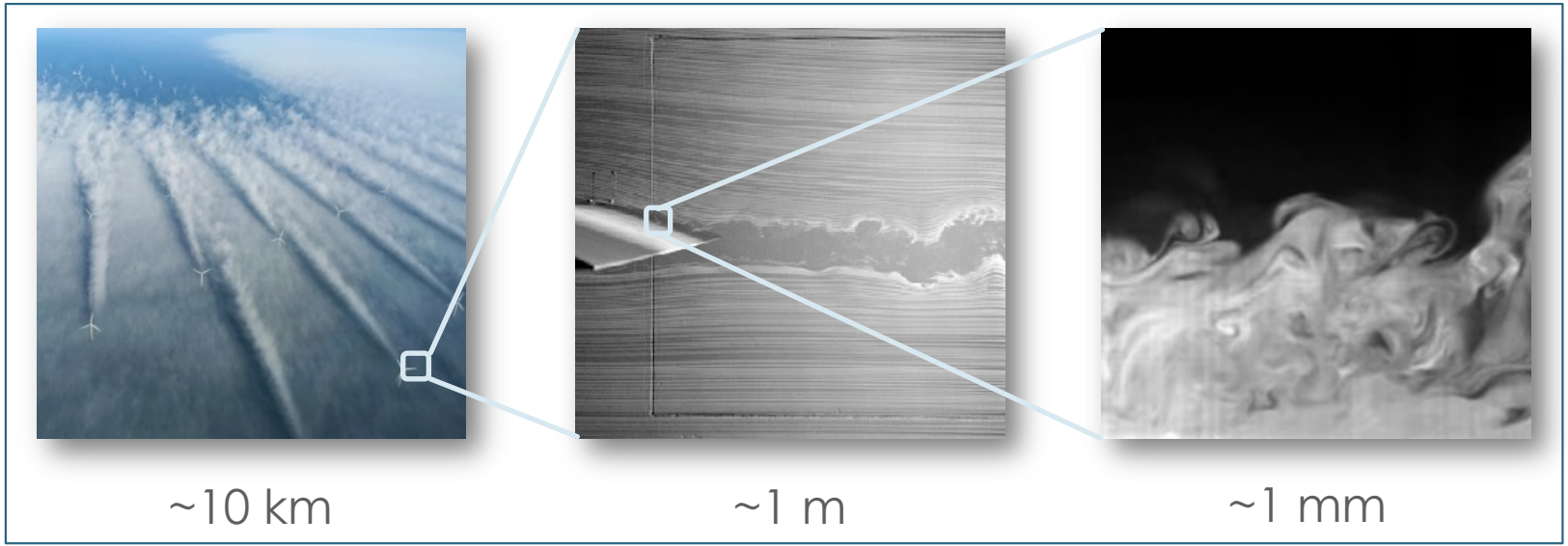
20th June 2023

BIRS Workshop on Scientific Machine Learning

CWI

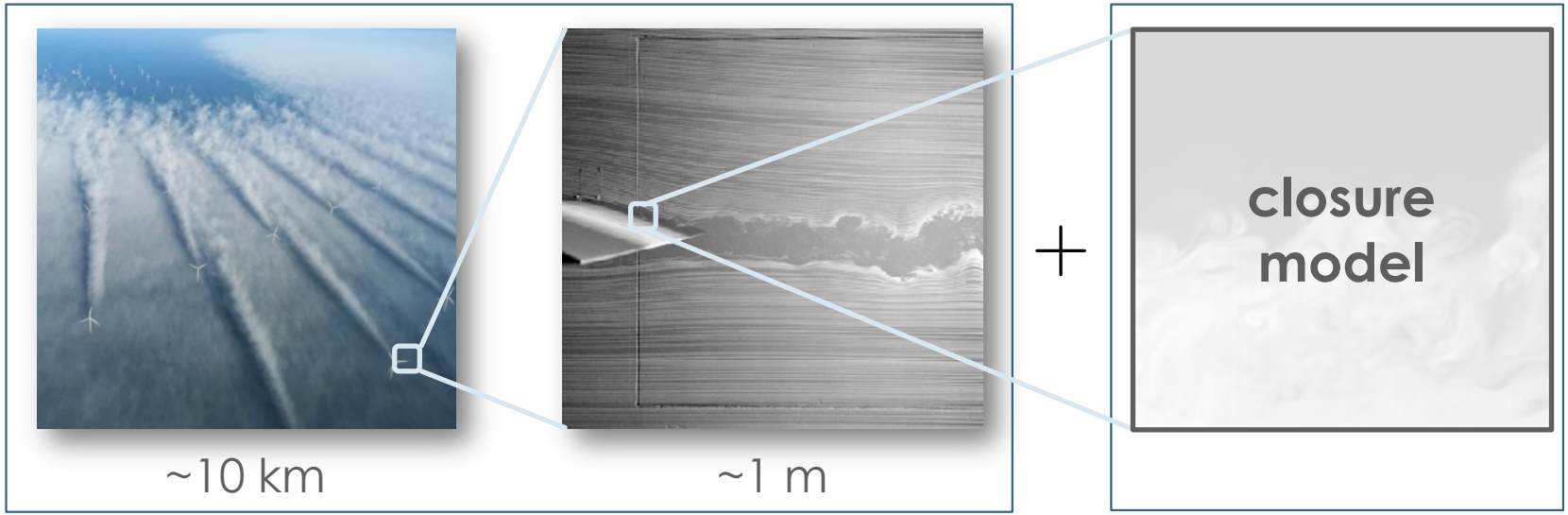
Scientific Machine Learning semester programme

Motivation: multiscale problems



Simulating all scales with a computational model is unfeasible

Accurate and stable closure models needed



Closure model approximates effect of small scales on large scales

Basics of closure modelling

- Multiscale fluid flow: Navier-Stokes equations

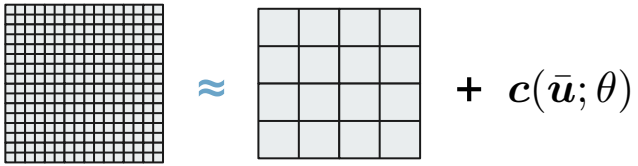
$$\frac{\partial \boldsymbol{u}}{\partial t} = \boldsymbol{f}(\boldsymbol{u}) \qquad \boldsymbol{f}(\boldsymbol{u}) := -\nabla \cdot (\boldsymbol{u} \otimes \boldsymbol{u}) - \nabla p + \nu \nabla^2 \boldsymbol{u}$$

- NS describes (too) many scales of motion for small viscosity ν
- Reduce range of scales by a **filtering** operation:

$$\bar{\boldsymbol{u}} = \mathcal{A}(\boldsymbol{u}) \qquad \mathcal{A}(\boldsymbol{u}) = \int \boldsymbol{u}(\xi, t) G(x, \xi) \mathrm{d}\xi \qquad \boldsymbol{u}' = \boldsymbol{u} - \bar{\boldsymbol{u}}$$

- **Aim**: use coarser meshes and larger time steps when solving for $\bar{\boldsymbol{u}}$

Basics of closure modelling



- Problem: filter and PDE operator **do not commute**
- Art: find a **closure model** with parameters θ s.t.

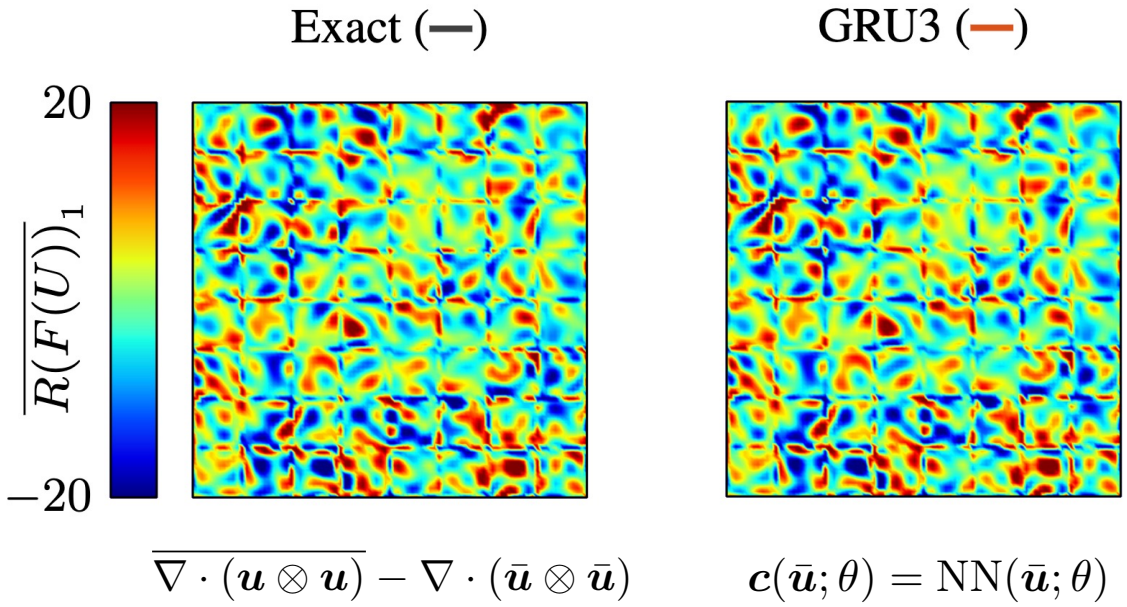
$$\boldsymbol{c}(\bar{\boldsymbol{u}};\theta) \approx \mathcal{C}[\mathcal{A},\boldsymbol{f}](\boldsymbol{u}) = \overline{\nabla\cdot(\boldsymbol{u}\otimes\boldsymbol{u})} - \nabla\cdot(\bar{\boldsymbol{u}}\otimes\bar{\boldsymbol{u}})$$

- Finding $\boldsymbol{c}(\bar{\boldsymbol{u}};\theta)$ is an inverse problem (model discovery)
- Common form: $\dfrac{\partial\bar{\boldsymbol{u}}}{\partial t} = \boldsymbol{f}(\bar{\boldsymbol{u}}) + \boldsymbol{c}(\bar{\boldsymbol{u}};\theta)$

Neural networks as closure model

$$\boldsymbol{c}(\bar{\boldsymbol{u}};\theta) = \mathrm{NN}(\bar{\boldsymbol{u}};\theta)$$

$$\frac{\mathrm{d}\bar{\mathbf{u}}}{\mathrm{d}t} = f(\bar{\mathbf{u}}) + \mathrm{NN}(\bar{\mathbf{u}};\theta)$$

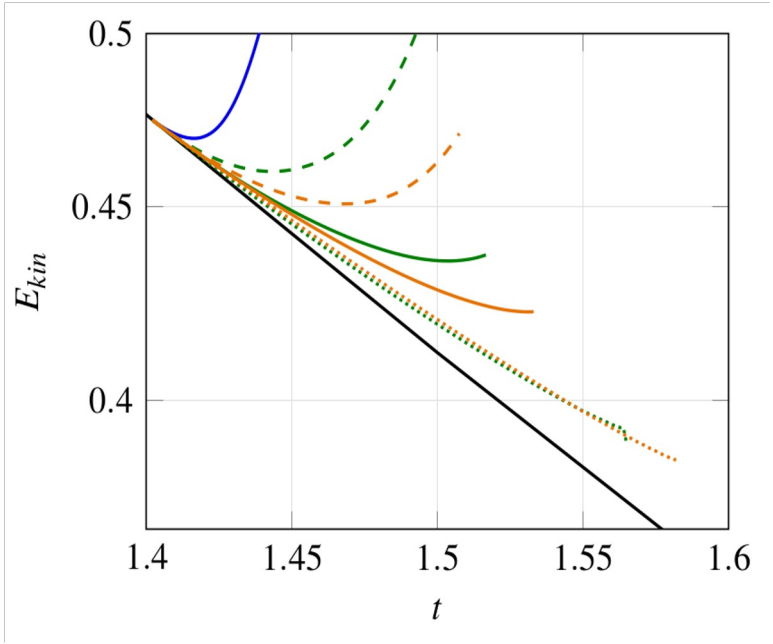


Kurz & Beck, “A machine learning framework for LES closure terms”, 2021

Issue: NNs destabilize the dynamical system

$$\frac{\mathrm{d}\bar{\mathbf{u}}}{\mathrm{d}t} = f(\bar{\mathbf{u}}) + \mathrm{NN}(\bar{\mathbf{u}};\theta)$$

- NN accurately matches closure term (operator fit)
- But: **solution is wrong**



Kurz & Beck, Investigating Model-Data Inconsistency in Data-Informed Turbulence Closure Terms, 2020

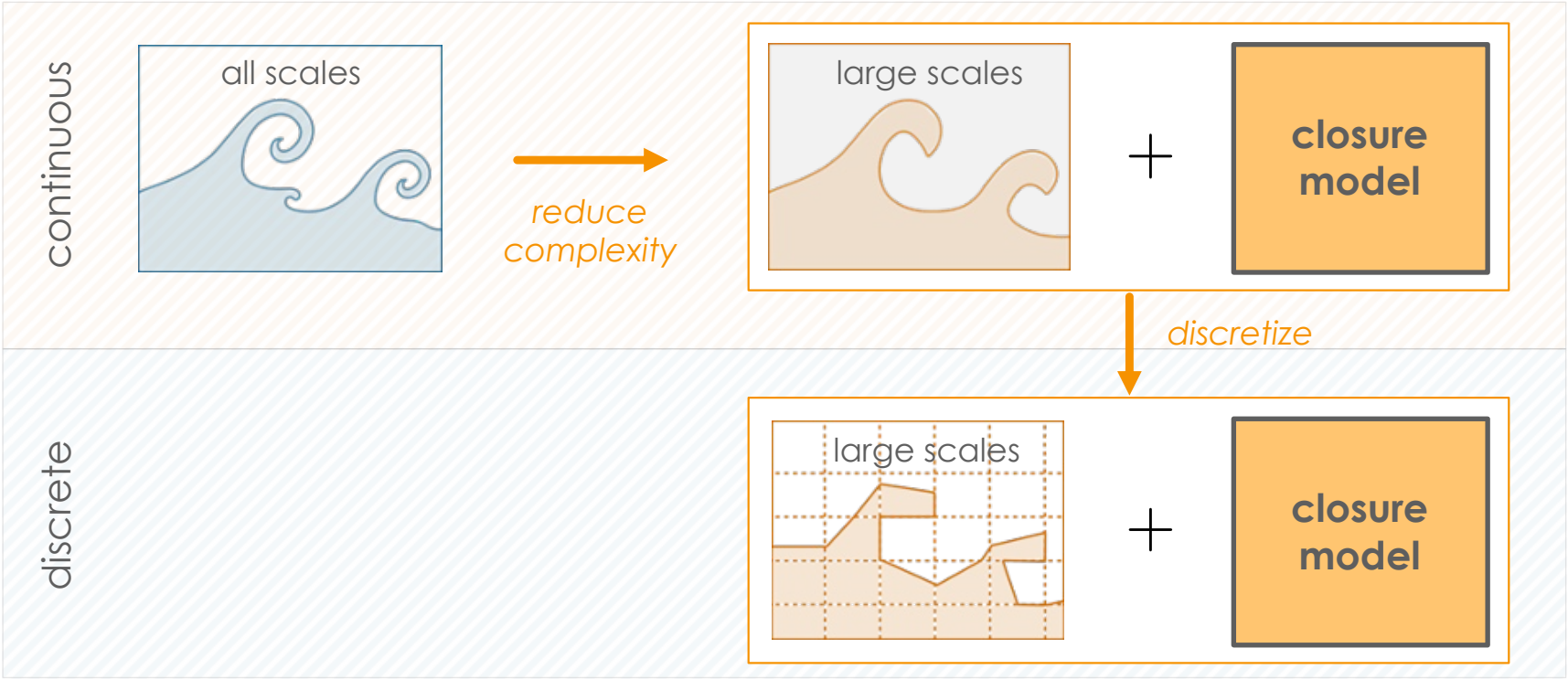
Tackling instability in dynamical systems with NNs

- **Instability** common problem for ML-based closure models (mismatch training environment and prediction environment)

- Recent approaches:
  - Stability training on data with artificial noise (Kurz & Beck, 2021)
  - Minimizing (or eliminating) backscatter (Park & Choi, 2021)
  - Projection onto a stable basis (Beck et al., 2019)
  - Trajectory fitting (List et al., 2022; MacArt et al., 2021)
  - Reinforcement learning (Bae & Koumoutsakos, 2022; Kurz et al. 2022)

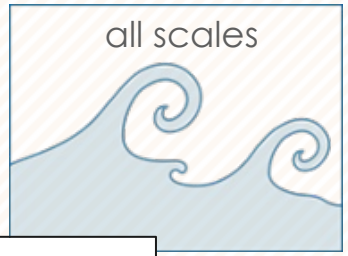
Our approach: “discretize first” + “preserve structure” + “embedded learning”

Common approach in closure modelling



Alternative: discretize first

continuous

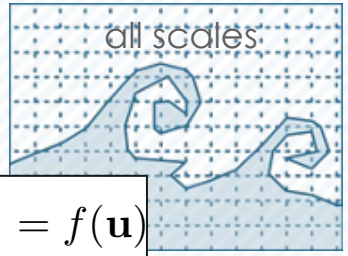


$$\frac{\partial \boldsymbol{u}}{\partial t} = \boldsymbol{F}(\boldsymbol{u})$$

discretize

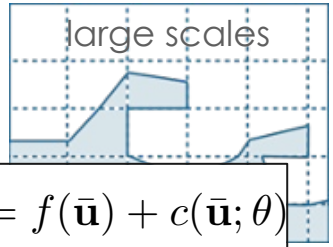
- Consistency training data and prediction environment
- Inverse problem easier on discrete level
- Less regularity requirements
- Easier to handle boundary conditions
- Observation data is always discrete

discrete



$$\frac{\mathrm{d}\mathbf{u}}{\mathrm{d}t} = f(\mathbf{u})$$

reduce complexity



$$\frac{\mathrm{d}\bar{\mathbf{u}}}{\mathrm{d}t} = f(\bar{\mathbf{u}}) + c(\bar{\mathbf{u}};\theta)$$

+

discrete closure model

Examples of preserving structure

- ODE formulation (“neural ODE”)
- Closure model form (“neural closure model”)
- Conservation form
- Translation invariance
- **Energy conservation**

$$\frac{\mathrm{d}\bar{\mathbf{u}}}{\mathrm{d}t} = \mathrm{NN}(\bar{\mathbf{u}};\theta)$$

$$\frac{\mathrm{d}\bar{\mathbf{u}}}{\mathrm{d}t} = f(\bar{\mathbf{u}}) + \mathrm{NN}(\bar{\mathbf{u}};\theta)$$

$$\frac{\mathrm{d}\bar{\mathbf{u}}}{\mathrm{d}t} = f(\bar{\mathbf{u}}) + \nabla \cdot \mathrm{NN}(\bar{\mathbf{u}};\theta)$$

CNN architecture

Energy conservation implies stability

- Many PDEs possess **secondary conservation laws**, such as energy or entropy, which give a **stability bound**

- **Example: Korteweg – de Vries**

$$\frac{\partial u}{\partial t} + 3\frac{\partial u^2}{\partial x} = -\frac{\partial^3 u}{\partial x^3} \quad \Longrightarrow \quad \frac{\mathrm{d}E}{\mathrm{d}t} = \frac{\mathrm{d}}{\mathrm{d}t}\underbrace{\frac{1}{2}\int_\Omega u^2 d\Omega}_{=:E} = 0$$

$$E := \frac{1}{2}\int u^2 \,\mathrm{d}\Omega$$

Idea: impose a similar structure on the filtered equations

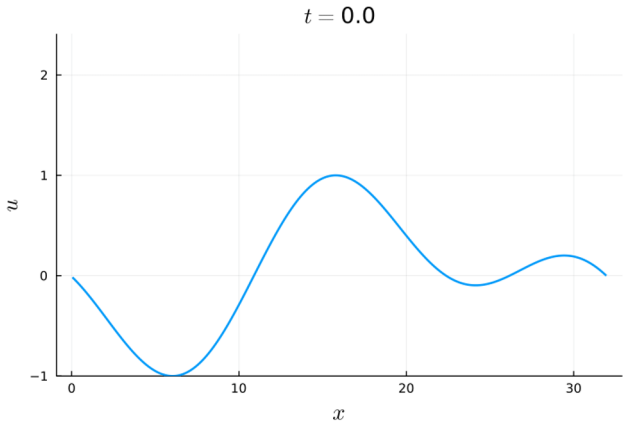
Korteweg - de Vries equation

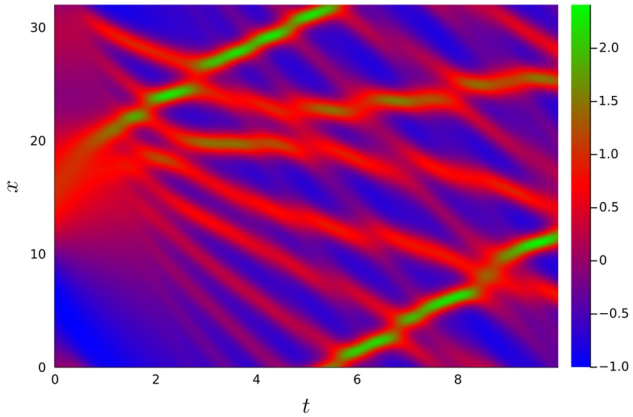
- KdV discretized using skew-symmetric scheme:

$$\frac{\partial u}{\partial t} + 3\frac{\partial u^2}{\partial x} = -\frac{\partial^3 u}{\partial x^3} \quad \Longrightarrow \quad \frac{\mathrm{d}\mathbf{u}}{\mathrm{d}t} = -3\mathbf{G}(\mathbf{u}) - \mathbf{D}_3\mathbf{u}$$

- Energy conservation (periodic BCs):

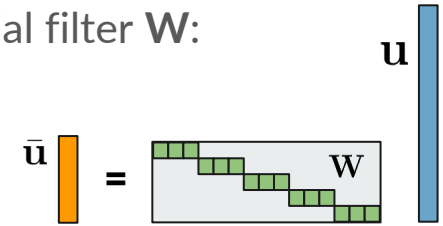
$$\frac{\mathrm{d}E}{\mathrm{d}t} = \frac{\mathrm{d}}{\mathrm{d}t}\underbrace{\frac{1}{2}\int_\Omega u^2 d\Omega}_{=:E} = 0 \quad \Longrightarrow \quad (\mathbf{u}, \frac{\mathrm{d}\mathbf{u}}{\mathrm{d}t}) = 0$$



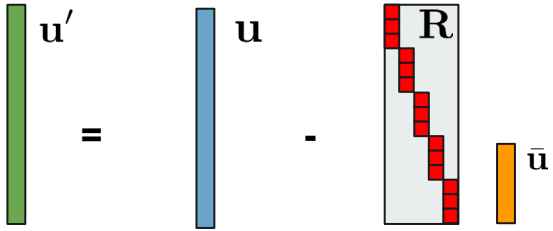


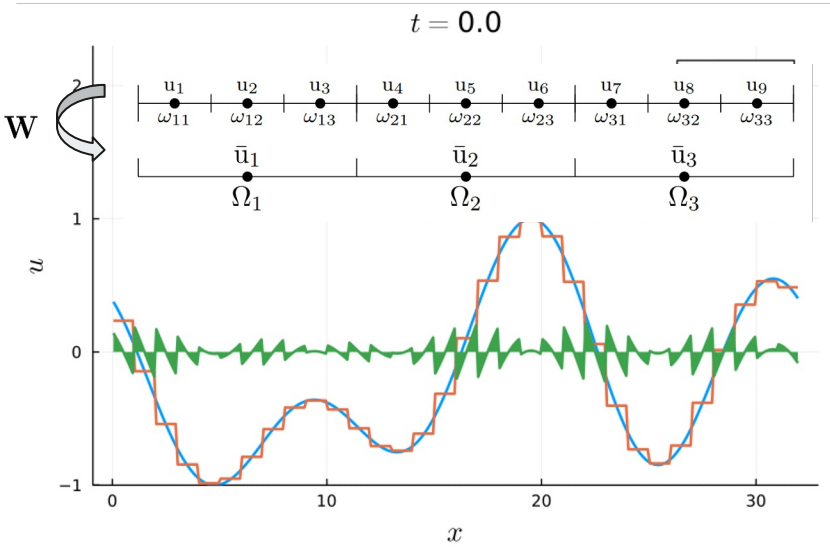
Discrete filtering and reconstruction

- Spatial filter **W**:



- Subgrid-scales defined via reconstruction operator **R**:





subgrid scales important near sharp gradients

Energy decomposition

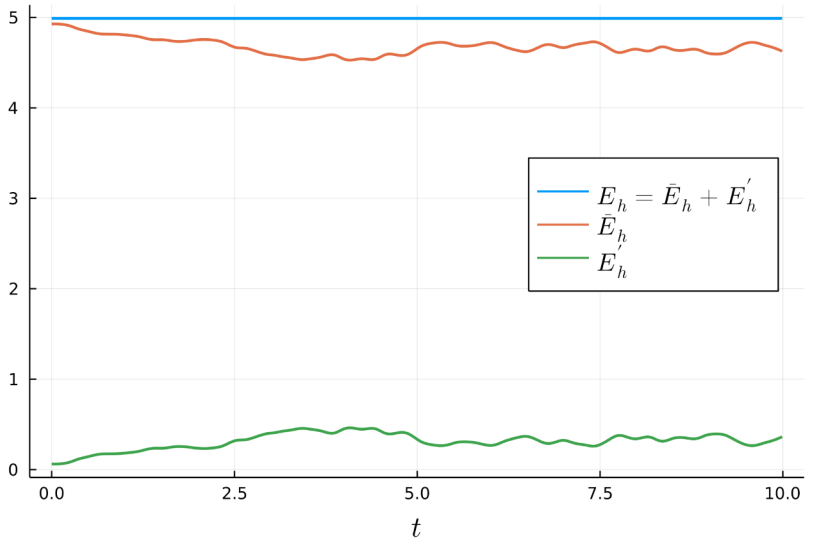
- Decompose the energy as:

$$E_h = \underbrace{\frac{1}{2}(\bar{\mathbf{u}}, \bar{\mathbf{u}})_\Omega}_{=:\bar{E}_h} + \underbrace{\frac{1}{2}(\mathbf{u}', \mathbf{u}')_\omega}_{=:E'_h}$$

- Time evolution:

$$\frac{\mathrm{d}E_h}{\mathrm{d}t} = \boxed{\frac{\mathrm{d}\bar{E}_h(\bar{\mathbf{u}})}{\mathrm{d}t}} + \boxed{\frac{\mathrm{d}E'_h(\mathbf{u}')}{\mathrm{d}t}} = 0$$

- **To use energy conservation we need information about the small scales**

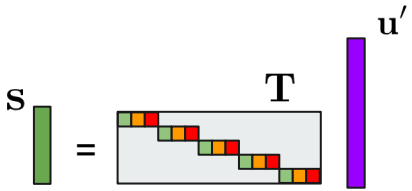


Total energy is conserved

Energy of filtered solution is not conserved

Subgrid compression

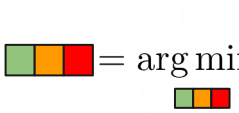
- Introduce (linear) compressed subgrid variables **s:**



- Require **s** to have same energy as **u'**:

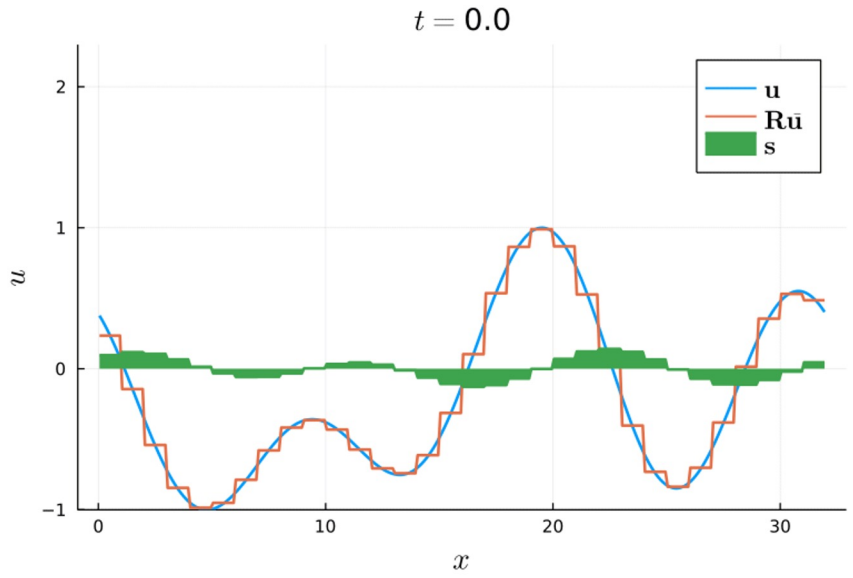
$$\frac{1}{2}(\mathbf{s},\mathbf{s})_\Omega \approx E'_h(\mathbf{u}')$$

- Solve minimization problem ("local POD"):

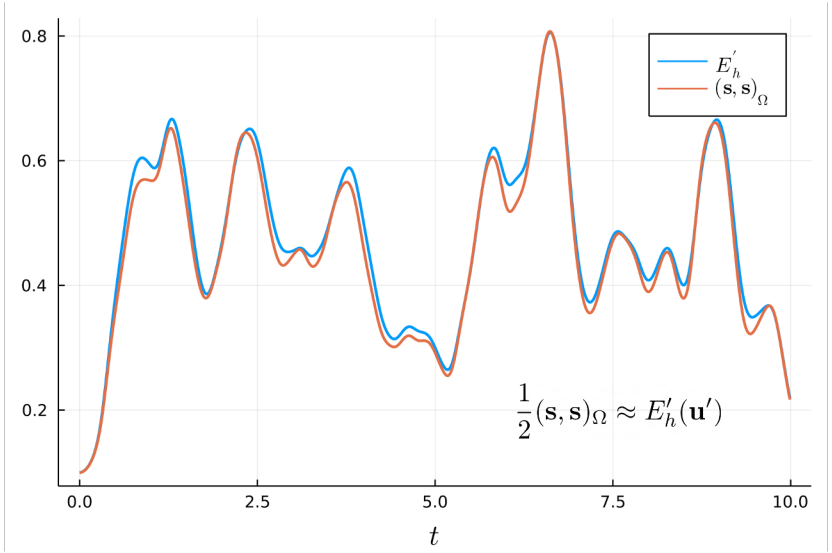


$$= \arg\min \sum_{d=1}^{\mathcal{D}} || \frac{1}{2}\mathbf{s}_d^2 - \frac{1}{2}\mathbf{W}(\mathbf{u}'_d)^2 ||_2^2$$

Compressed variables learn effective subgrid content



compressed subgrid variable identifies sharp gradients



learned compression matches subgrid energy closely

$$\frac{\mathrm{d}\bar{\mathbf{u}}}{\mathrm{d}t} = f(\bar{\mathbf{u}}) + \underbrace{\overline{f(\mathbf{u})} - f(\bar{\mathbf{u}})}_{\approx c(\bar{\mathbf{u}};\theta)}$$

Energy-conserving closure model

- **Extended neural closure model:**

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix}\bar{\mathbf{u}}\\ \mathbf{s}\end{bmatrix} = \begin{bmatrix}f(\bar{\mathbf{u}})\\ \mathbf{0}\end{bmatrix} + \begin{bmatrix}c_u(\bar{\mathbf{u}},\mathbf{s};\theta_u)\\ c_s(\bar{\mathbf{u}},\mathbf{s};\theta_s)\end{bmatrix}$$

Large scale dynamics

Compressed small scale dynamics

- Idea: learn **a skew-symmetric matrix** $\mathcal{K}$ whose entries are NN outputs

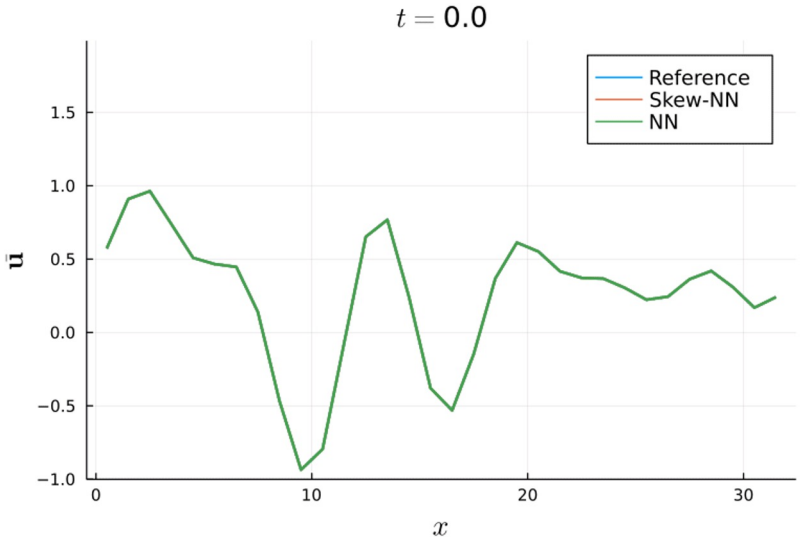
$$\begin{bmatrix}c_u(\bar{\mathbf{u}},\mathbf{s};\theta_u)\\ c_s(\bar{\mathbf{u}},\mathbf{s};\theta_s)\end{bmatrix} = \mathcal{K}(\bar{\mathbf{u}},\mathbf{s};\boldsymbol{\Theta})\begin{bmatrix}\bar{\mathbf{u}}\\ \mathbf{s}\end{bmatrix}$$

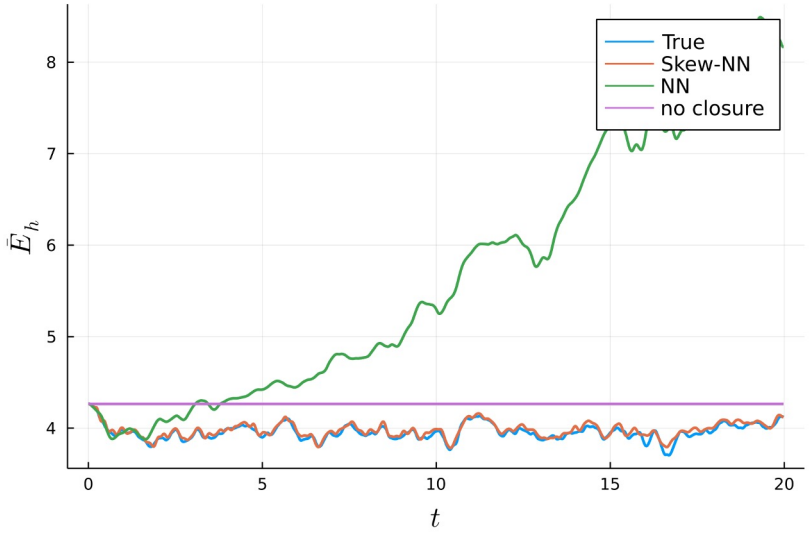
- Energy conservation:

$$\begin{bmatrix}\bar{\mathbf{u}}\\ \mathbf{s}\end{bmatrix}^T \mathcal{K}(\cdot)\begin{bmatrix}\bar{\mathbf{u}}\\ \mathbf{s}\end{bmatrix} = 0 \quad \Longrightarrow \quad \frac{\mathrm{d}\bar{E}_h(\bar{\mathbf{u}})}{\mathrm{d}t} + \frac{1}{2}\frac{\mathrm{d}(\mathbf{s},\mathbf{s})_\omega}{\mathrm{d}t} = 0$$

New closure model improves quality + stability

- Trained on different initial conditions, tested on unseen initial conditions
- Reduction from N = 600 to N = 30
- Compare to standard CNN

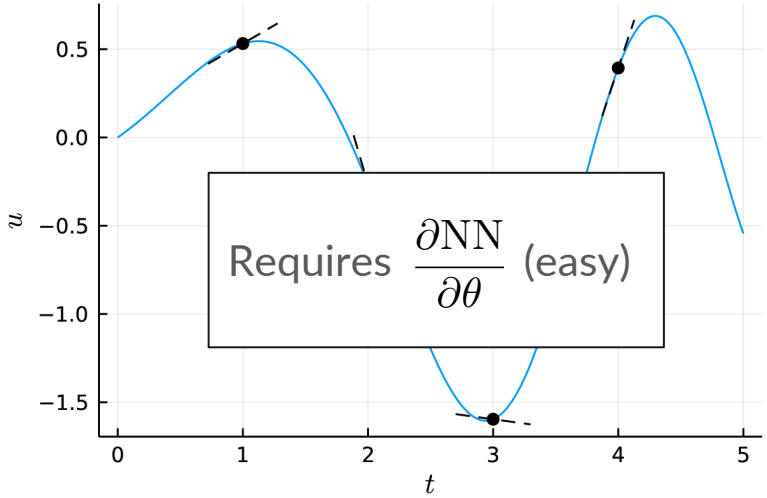




What about training neural closure models?

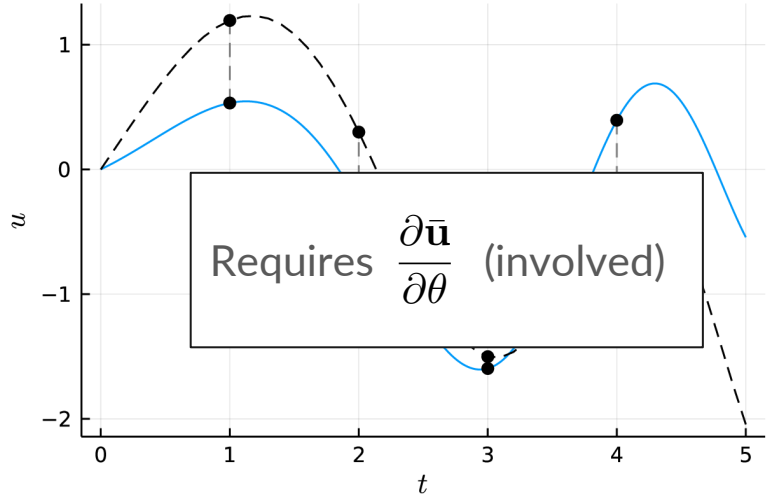
$$\frac{\mathrm{d}\bar{\mathbf{u}}}{\mathrm{d}t} = f(\bar{\mathbf{u}}) + \mathrm{NN}(\bar{\mathbf{u}};\theta)$$

Derivative fitting



$$\mathrm{Loss} = \left\| \left(\frac{\mathrm{d}\bar{\mathbf{u}}}{\mathrm{d}t} \right)_{\mathrm{ref}} - f(\bar{\mathbf{u}}_{\mathrm{ref}}) - \mathrm{NN}(\bar{\mathbf{u}}_{\mathrm{ref}};\theta) \right\|^2$$

Trajectory fitting



$$\mathrm{Loss} = \sum_{i=1}^{N_t} \left\| \bar{\mathbf{u}}_{\mathrm{ref}}(t_i) - \bar{\mathbf{u}}(t_i) \right\|^2, \text{ where } \frac{\mathrm{d}\bar{\mathbf{u}}}{\mathrm{d}t} = f(\bar{\mathbf{u}}) + \mathrm{NN}(\bar{\mathbf{u}};\theta)$$

$$\text{Loss} = \left\| \left(\frac{\mathrm{d}\bar{\mathbf{u}}}{\mathrm{d}t} \right)_{\text{ref}} - f(\bar{\mathbf{u}}_{\text{ref}}) - \text{NN}(\bar{\mathbf{u}}_{\text{ref}}; \theta) \right\|^2$$

Derivative fitting: theoretical results

Theorem 3.1 (Hairer et al. [11]). *Let $\mathbf{u}_{\text{ref}}(t) \in \mathbb{R}^{N_x}, t \geq 0$ be given, let $\mathbf{u}(t) \in \mathbb{R}^{N_x}, t \geq 0$ be the solution of the ODE $\frac{\mathrm{d}\mathbf{u}}{\mathrm{d}t} = g(\mathbf{u}; \vartheta)$ with initial condition $\mathbf{u}(0) = \mathbf{u}_{\text{ref}}(0)$, and let $\|\cdot\|$ be a norm on $\mathbb{R}^{N_x}$. If the following holds:*

$$a) \quad \left\| \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{u}_{\text{ref}}(t) - g(\mathbf{u}_{\text{ref}}(t); \vartheta) \right\| \leq \varepsilon,$$

$$b) \quad \|g(\mathbf{a}; \vartheta) - g(\mathbf{b}; \vartheta)\| \leq C \|\mathbf{a} - \mathbf{b}\|,$$

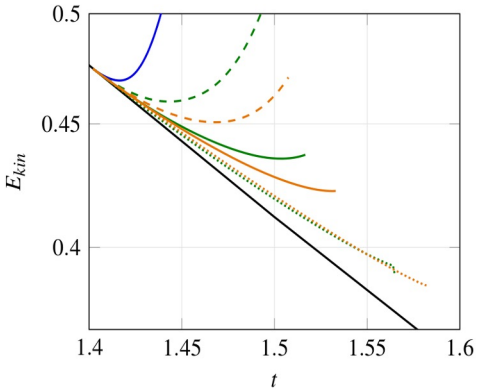
for fixed Lipschitz constant $C > 0$ and fixed $\varepsilon > 0$. Then the following error bound holds:

$$\|\mathbf{u}_{\text{ref}}(t) - \mathbf{u}(t)\| \leq \frac{\varepsilon}{C}\left(e^{Ct} - 1\right).$$

If a neural ODE:

- approximates the derivative well
- is Lipschitz

Then, the resulting solution **may be inaccurate**.



$$\text{Loss} = \sum_{i=1}^{N_t} \|\bar{\mathbf{u}}(t_i) - \bar{\mathbf{u}}(t_i)\|^2, \text{ where } \frac{\mathrm{d}\bar{\mathbf{u}}}{\mathrm{d}t} = f(\bar{\mathbf{u}}) + \mathrm{NN}(\bar{\mathbf{u}}; \theta)$$

Trajectory fitting: theoretical results

Theorem 3.2. *Let* $\mathbf{u}_{\text{ref}}(t_i), i = 0, 1, \ldots, \ldots$ *be a sequence of vectors in* $\mathbb{R}^{N_x}$ *where* $t_i = i\Delta t$*, let* $\|\cdot\|$ *be a norm on* $\mathbb{R}^{N_x}$*, and let* $G(\cdot; \vartheta) : \mathbb{R}^{N_x} \to \mathbb{R}^{N_x}$ *be a function such that:*

$$a) \quad \|\mathbf{u}_{\text{ref}}(t_{i+1}) - G(\mathbf{u}_{\text{ref}}(t_i); \vartheta)\| \leq \varepsilon \text{ for all } i = 1, 2, \ldots, N_t,$$
$$b) \quad \|G(\mathbf{a}; \vartheta) - G(\mathbf{b}; \vartheta)\| \leq C \|\mathbf{a} - \mathbf{b}\| \text{ for all } \mathbf{a}, \mathbf{b} \in \mathbb{R}^{N_x},$$

for fixed Lipschitz constant $C > 0, C \neq 1$ *and fixed* $\varepsilon > 0$*. Define the sequence* $\mathbf{u}(t_{i+1}) = G(\mathbf{u}(t_i); \vartheta)$ *with* $\mathbf{u}(0) = \mathbf{u}_{\text{ref}}(0)$*. Then the following error bound holds:*

$$\|\mathbf{u}(t_k) - \mathbf{u}_{\text{ref}}(t_k)\| \leq \varepsilon \frac{C^k - 1}{C - 1} \text{ for } k = 0, 1, 2, \ldots.$$

If a discretised neural ODE:

- accurately represents single time steps
- is Lipschitz

Then the resulting ODE solution **may be inaccurate.**

Trajectory length should be **sufficiently large**

$$\text{Loss} = \sum_{i=1}^{N_t} \|\bar{\mathbf{u}}(t_i) - \bar{\mathbf{u}}(t_i)\|^2, \text{ where } \frac{\mathrm{d}\bar{\mathbf{u}}}{\mathrm{d}t} = f(\bar{\mathbf{u}}) + \mathrm{NN}(\bar{\mathbf{u}}; \theta)$$

Trajectory fitting: computing $\frac{\mathrm{dLoss}}{\mathrm{d}\theta}$

1. **Discretise-then-optimise (DtO):**
   - Need differentiable solver
   - Preferably explicit

2. **Optimise-then-discretise (OtD):**
   - Solve adjoint equations[1]
   - Here: interpolating adjoint
   - Need dense output

$$\begin{cases} \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{y}^T = -\mathbf{y}^T \frac{\partial}{\partial \bar{\mathbf{u}}}(f(\bar{\mathbf{u}}) + \mathrm{NN}(\bar{\mathbf{u}}; \theta)) \\ \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{z}^T = -\mathbf{y}^T \frac{\partial}{\partial \theta}(f(\bar{\mathbf{u}}) + \mathrm{NN}(\bar{\mathbf{u}}; \theta)) \\ \frac{\mathrm{dLoss}}{\mathrm{d}\theta} = \mathbf{z}(0) \end{cases}$$

[1]Chen et al, Neural ordinary differential equations, NeurIPS 2018

Derivative fitting vs. trajectory fitting

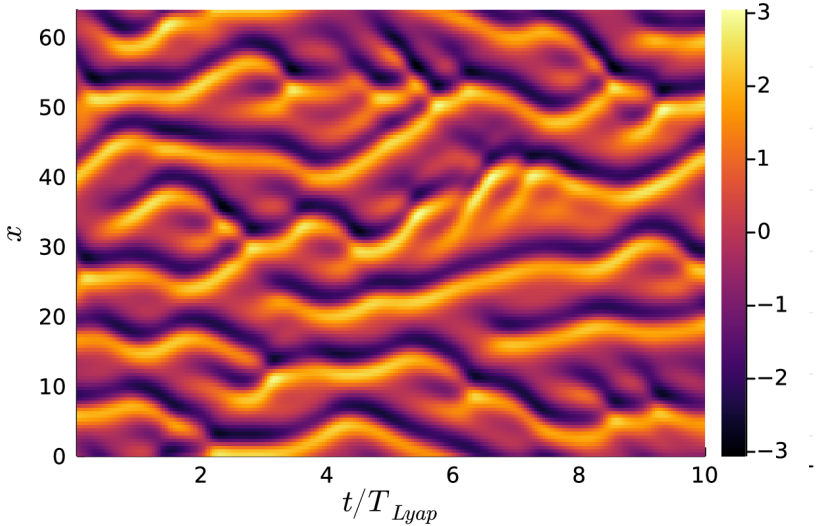
	Derivative fitting	Trajectory fitting	
		DtO	OtD
Differentiability required	NN	NN, f, ODE solver	NN, f
Accuracy of loss function gradients	Exact	Exact	Approximate
Learns long-term accuracy	No	Yes	Yes
Requires time-derivatives of training data	Yes	No	No
Computational cost	Low	High	High

Kuramoto-Sivashinsky equation

$$\frac{\partial u}{\partial t} = -\frac{1}{2}\frac{\partial}{\partial x}\left(u^2\right) - \frac{\partial^2 u}{\partial x^2} - \frac{\partial^4 u}{\partial x^4}$$

$$\frac{\mathrm{d}\bar{\mathbf{u}}}{\mathrm{d}t} = f(\bar{\mathbf{u}}) + \Delta_{\mathrm{fwd}}\mathrm{NN}(\bar{\mathbf{u}};\theta)$$

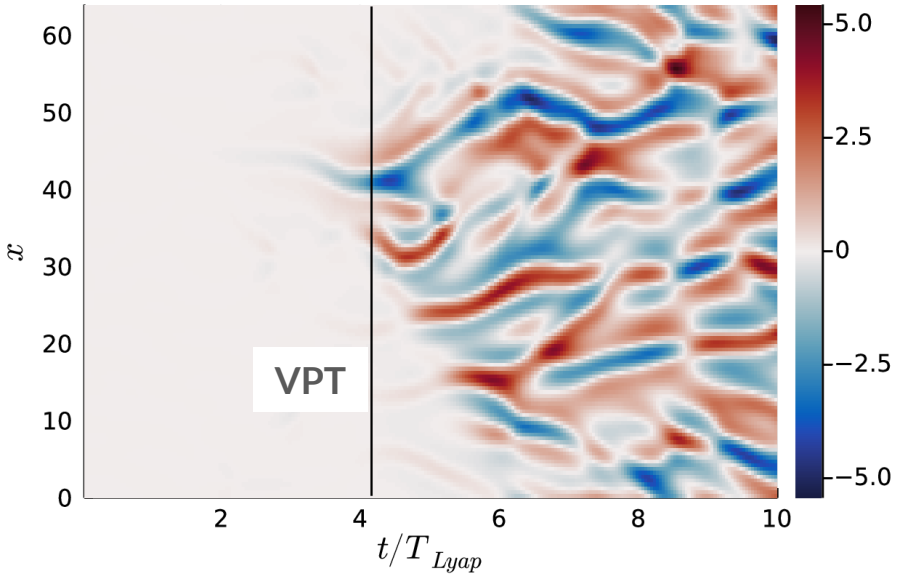
- **Chaotic:**
  - Trajectory lengths not too large
- **Stiff**:
  - OtD: impl/expl RK, KenCarp47[1]
  - DtO: expl ETDRK4 in Fourier domain[2]
- Filter **W**:
  - downsampling 1024 => 128



[1]Kennedy and Carpenter, Higher-order additive Runge-Kutta schemes for ODEs, Applied Numerical Mathematics, 2019.
[2]Kassam & Trefethen, Fourth-order time-stepping for stiff PDEs. SIAM Journal on Scientific Computing, 2005

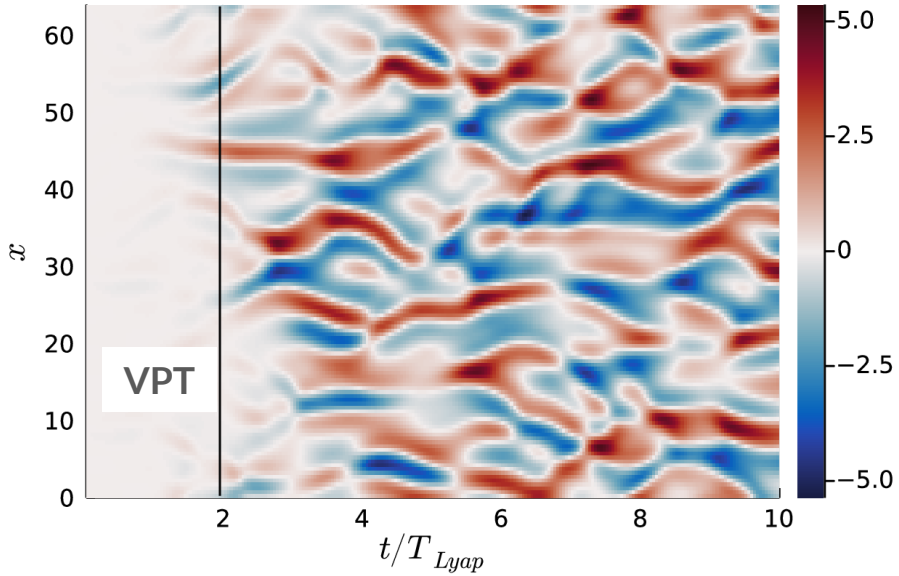
Effect of trajectory length, OtD

prediction errors



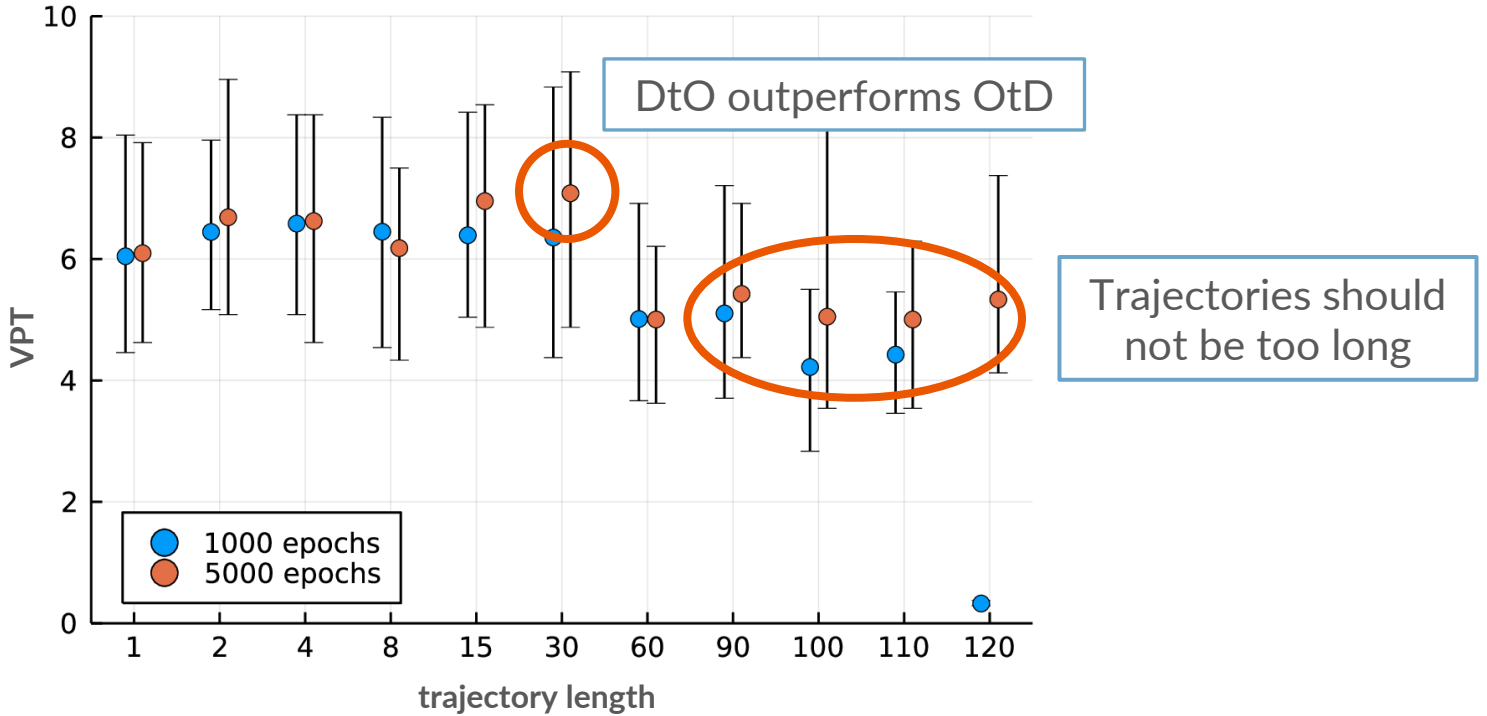
short trajectories (24 steps)

prediction errors



long trajectories (144 steps)

Effect of trajectory length, DtO

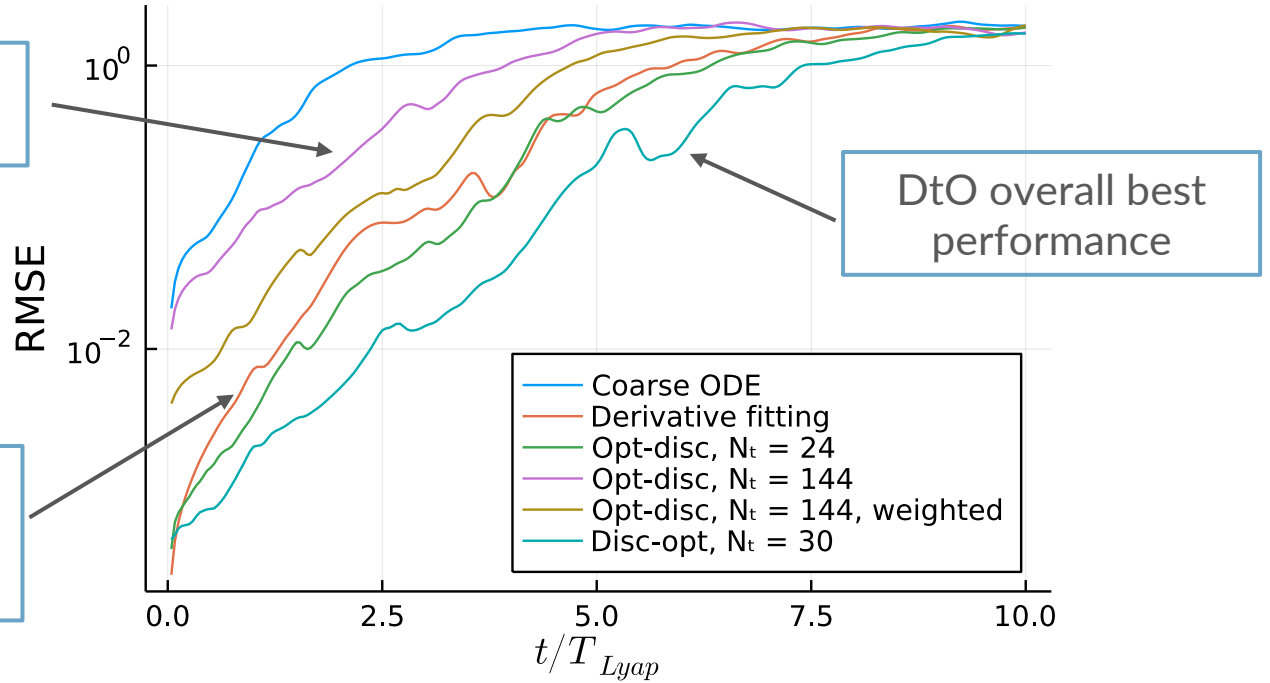


DtO outperforms OtD and derivative fitting

OtD sensitive to longer intervals

Derivative fitting OK, but testcase dependent

DtO overall best performance

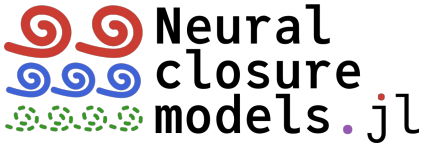


Conclusions

- **“Discretize first”**
  - Tailor-made closure models
  - Useful framework for NNs, eases analysis
- **“Preserve structure”**
  - Accuracy improvement by adding physics knowledge
  - Non-linear stability with energy-conserving methods
- **“Embedded learning”** with **trajectory fitting**
  - Discretise-then-optimise with **differentiable solvers** preferred
  - Promising, but “strings attached”: problem-dependent, comparison not easy

Julia is great for differentiable programming

- Neural closure models
  - https://github.com/HugoMelchers/neural-closure-models



- Incompressible, energy-conserving Navier-Stokes code
  - https://github.com/agdestein/IncompressibleNavierStokes.jl

- DifferentialEquations.jl by Rackauckas et al.
  - https://sciml.ai

References

Energy-Conserving Neural Network for Turbulence Closure Modeling

T. van Gastelen[a], W. Edeling[a], B. Sanderse[a]

[a]*Centrum Wiskunde & Informatica, Science Park 123, Amsterdam, The Netherlands*

Abstract

In turbulence modeling, and more particularly in the Large-Eddy Simulation (LES) framework, we are concerned with finding closure models that represent the effect of the unresolved subgrid scales on the resolved scales. Recent approaches gravitate towards machine learning techniques to construct such models. However, the stability of machine-learned closure models and their abidance by physical structure (e.g. symmetries, conservation laws) are still open problems. To tackle both issues, we take the 'discretize first, filter next' approach, in which we apply a spatial averaging filter to existing energy-conserving (fine-grid) discretizations. The main novelty is that we extend the system of equations describing the filtered solution with a set of equations that describe the evolution of (a compressed version of) the energy of the subgrid scales. Having an estimate of the energy of the subgrid scales, we can use the concept of energy conservation and derive stability of the discrete representation. The compressed variables are determined via a data-driven technique in such a way that the energy of the subgrid scales is matched. For the extended system, the closure model should be energy-conserving, and a new skew-symmetric convolutional neural network architecture is proposed that has this property. Stability is thus guaranteed, independent of the actual weights and biases of the network. Importantly, our framework allows energy exchange between resolved scales and compressed subgrid scales and thus enables backscatter. To model dissipative systems (e.g. viscous flows), the framework is extended with a diffusive component. The introduced neural network architecture is constructed such that it also satisfies momentum conservation. We apply the new methodology to both the viscous Burgers' equation and the Korteweg-De Vries equation in 1D and show superior stability properties when compared to a vanilla convolutional neural network.

v3 [math.NA] 28 Feb 2023

Computers and Mathematics with Applications 143 (2023) 94–107

Contents lists available at ScienceDirect

Computers and Mathematics with Applications

journal homepage: www.elsevier.com/locate/camwa

Comparison of neural closure models for discretised PDEs

Hugo Melchers [a,c,1], Daan Crommelin [a,b], Barry Koren [c], Vlado Menkovski [c], Benjamin Sanderse [a,*]

[a] *Centrum Wiskunde & Informatica, Science Park 123, 1098 XG, Amsterdam, the Netherlands*
[b] *Korteweg-de Vries Institute for Mathematics, University of Amsterdam, Science Park 105-107, 1098 XG, Amsterdam, the Netherlands*
[c] *Eindhoven University of Technology, De Zaale, 5600 MB, Eindhoven, the Netherlands*

ARTICLE INFO

Dataset link: https://github.com/HugoMelchers/neural-closure-models

Keywords:
Ordinary differential equations
Neural networks
Neural ODE
Partial differential equations
Multiscale modelling
Closure model

ABSTRACT

Neural closure models have recently been proposed as a method for efficiently approximating small scales in multiscale systems with neural networks. The choice of loss function and associated training procedure has a large effect on the accuracy and stability of the resulting neural closure model. In this work, we systematically compare three distinct procedures: "derivative fitting", "trajectory fitting" with discretise-then-optimise, and "trajectory fitting" with optimise-then-discretise. Derivative fitting is conceptually the simplest and computationally the most efficient approach and is found to perform reasonably well on one of the test problems (Kuramoto-Sivashinsky) but poorly on the other (Burgers). Trajectory fitting is computationally more expensive but is more robust and is therefore the preferred approach. Of the two trajectory fitting procedures, the discretise-then-optimise approach produces more accurate models than the optimise-then-discretise approach. While the optimise-then-discretise approach can still produce accurate models, care must be taken in choosing the length of the trajectories used for training, in order to train the models on long-term behaviour while still producing reasonably accurate gradients during training. Two existing theorems are interpreted in a novel way that gives insight into the long-term accuracy of a neural closure model based on how accurate it is in the short term.