Structure-preserving learning of embedded, discrete closure models

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

7th October 2022

Woudschoten conference

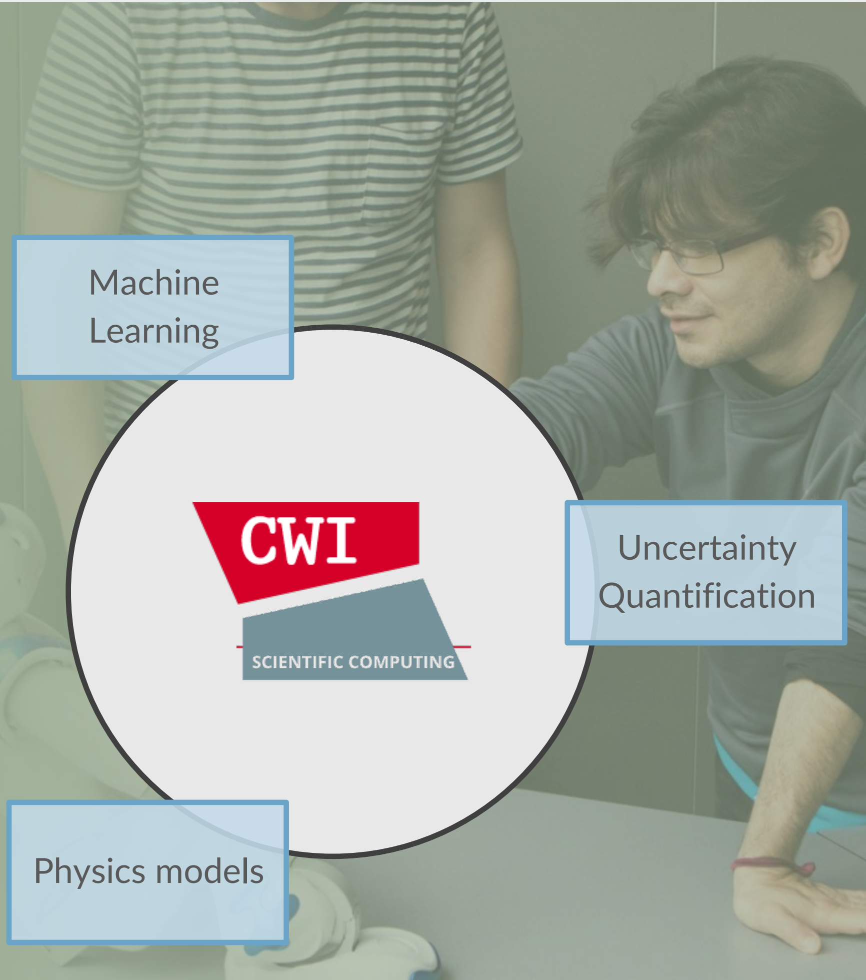
CWI

Scientific Computing group

Predictive science at the interface of ML, UQ and PDEs

Common theme: use physics knowledge to steer design of ML & UQ algorithms

- Closure models
- Reduced order models
- Bayesian inverse problems
- Neural networks



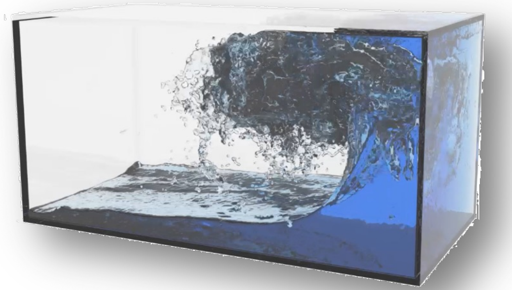
Scientific Computing and Machine Learning

- **SC for ML**
  approximation theory of neural networks; optimization theory; improve and understand NNs

- **SC by ML**
  improve existing SC methods, e.g. use NNs for matrix inversion

- **SC and ML**
  tight integration of SC and ML methods **- focus of this talk**

Typical applications: energy and climate

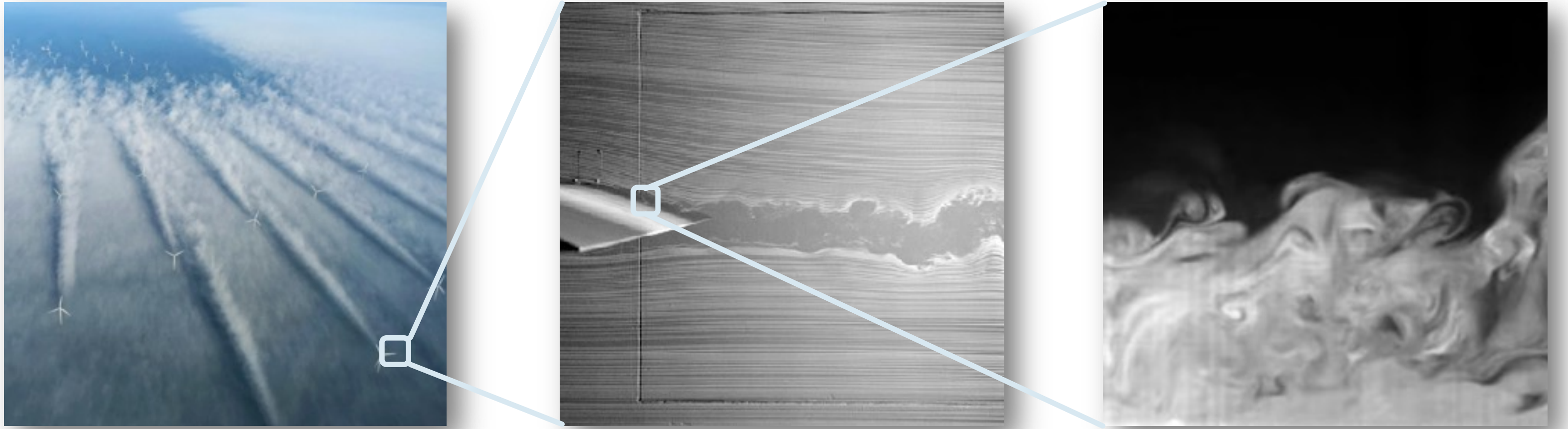


Sloshing of LNG

Offshore wind farms

Weather & climate

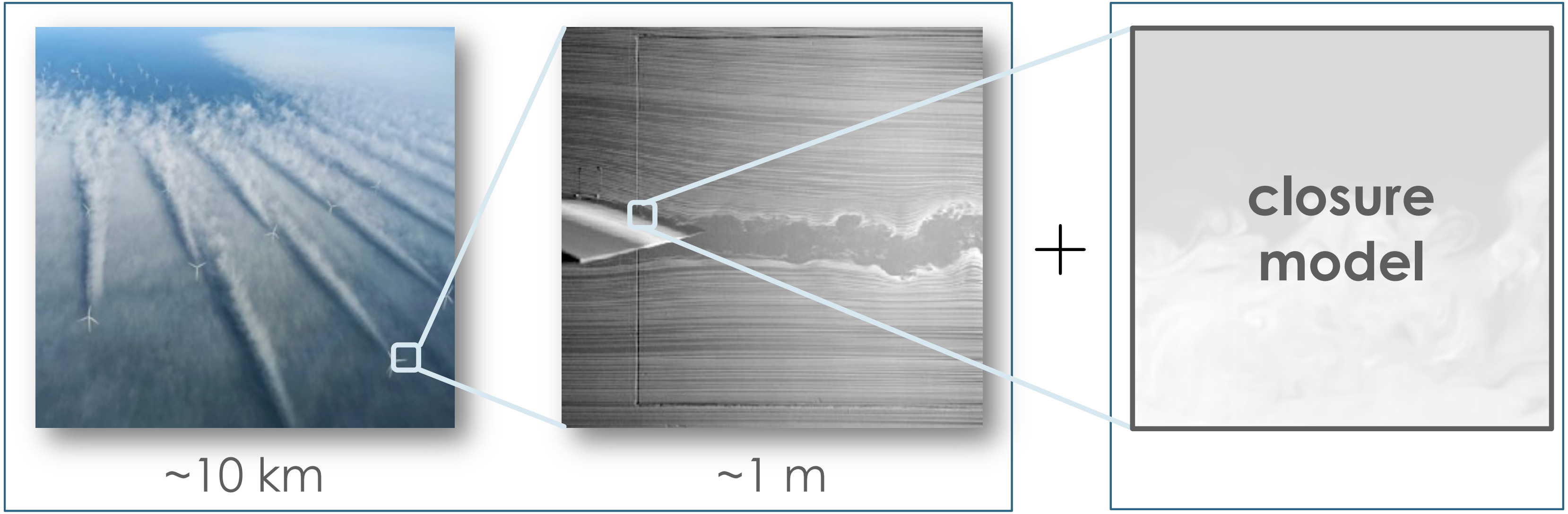
Many applications feature multiscale fluid flows



~10 km ~1 m ~1 mm

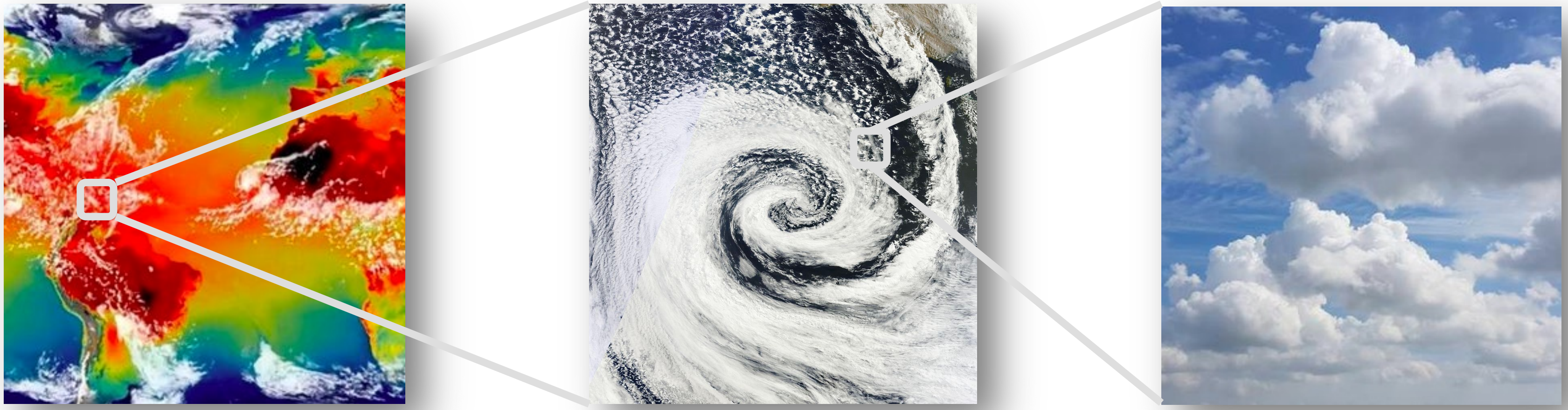
Simulating all scales with a computational model is unfeasible

Accurate and stable closure models needed



Closure model approximates effect of small scales on large scales

Closure problems occur in many fields

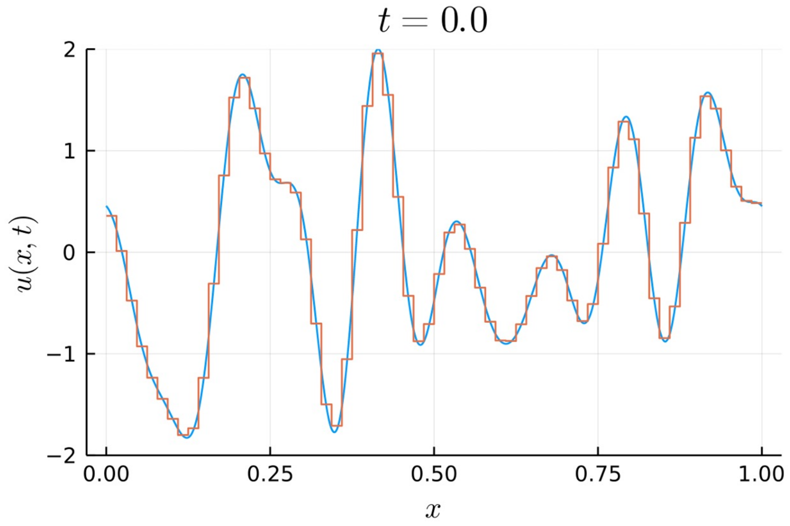


Resolving clouds in climate/weather models: “parameterization”

Example: "closure" with neural network

- Burgers' equation: $$\frac{\partial u}{\partial t} = -\frac{1}{2}\frac{\partial}{\partial x}\left(u^2\right) + \nu\frac{\partial^2 u}{\partial x^2}$$
- Small scales appear for small viscosity ν
- Aim: **accurate solutions on coarse grids**
- "Simple" machine learning approach:

$$\mathbf{u}(t + \Delta t) = \mathbf{u}(t) + \Delta t \cdot \mathrm{NN}(\mathbf{u}(t); \vartheta)$$

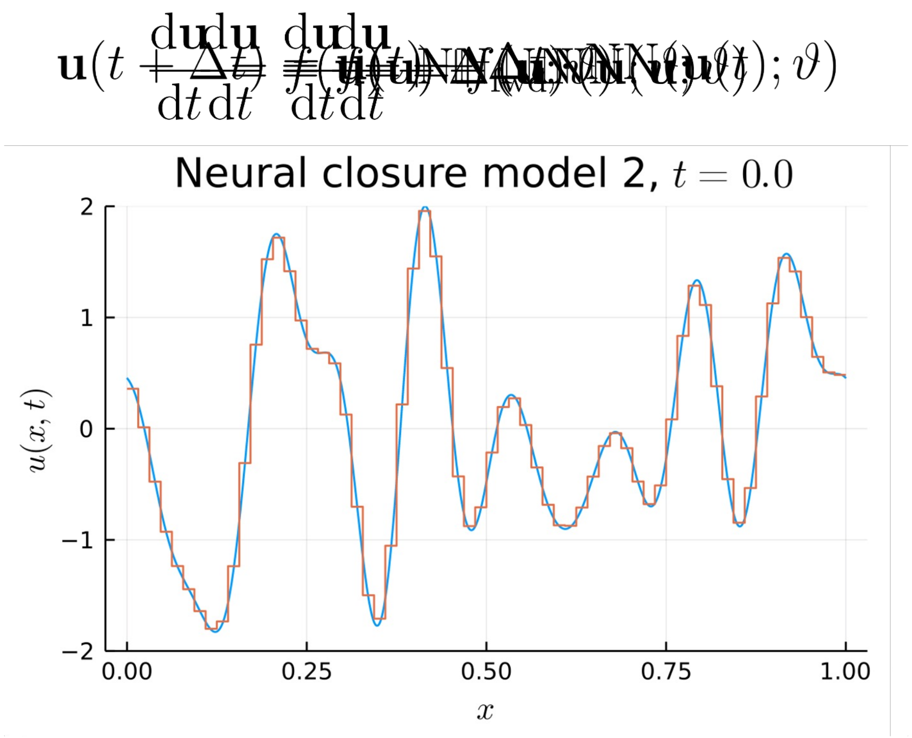


SciML approaches reduce error

increasing structure (“inductive bias”)

- Low-res model (“no closure”):
  0.10
- Basic ML model:
  0.087
- Neural ODE:
  0.041
- Neural closure model:
  0.029
- With momentum conservation:
  0.026

Including physics is most useful for small neural networks



Today’s talk

- **Structure-preserving closure models and stability**
- **Training procedures: derivative fitting vs. trajectory fitting**

“Discretize first” - “Preserve structure” – “Embedded learning”

- **Non-locality** in space and time (Mori-Zwanzig)
- **Stochastic** closure models
- **Reduced order models** and closure

Basics of closure modelling

- We consider PDEs describing many scales, e.g. the 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

- Art: find a **closure model** with parameters θ s.t.

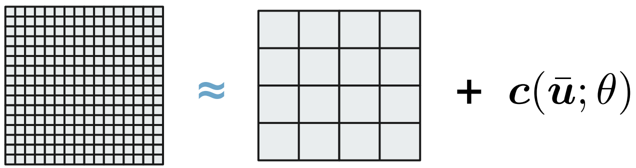
$$\boldsymbol{c}(\bar{\boldsymbol{u}};\theta) \approx \mathcal{C}[\mathcal{A},\mathcal{F}](\boldsymbol{u})$$

- Commutator error often due to nonlinearity, e.g. (Navier-Stokes):

$$\mathcal{C}[\mathcal{A},\mathcal{F}](\boldsymbol{u}) = \overline{\nabla \cdot (\boldsymbol{u} \otimes \boldsymbol{u})} - \nabla \cdot (\bar{\boldsymbol{u}} \otimes \bar{\boldsymbol{u}})$$

- Finding $\boldsymbol{c}(\bar{\boldsymbol{u}};\theta)$ is an inverse problem which can have multiple solutions

- Common form: $\dfrac{\partial \bar{\boldsymbol{u}}}{\partial t} = \boldsymbol{F}(\bar{\boldsymbol{u}}) + \boldsymbol{c}(\bar{\boldsymbol{u}};\theta)$



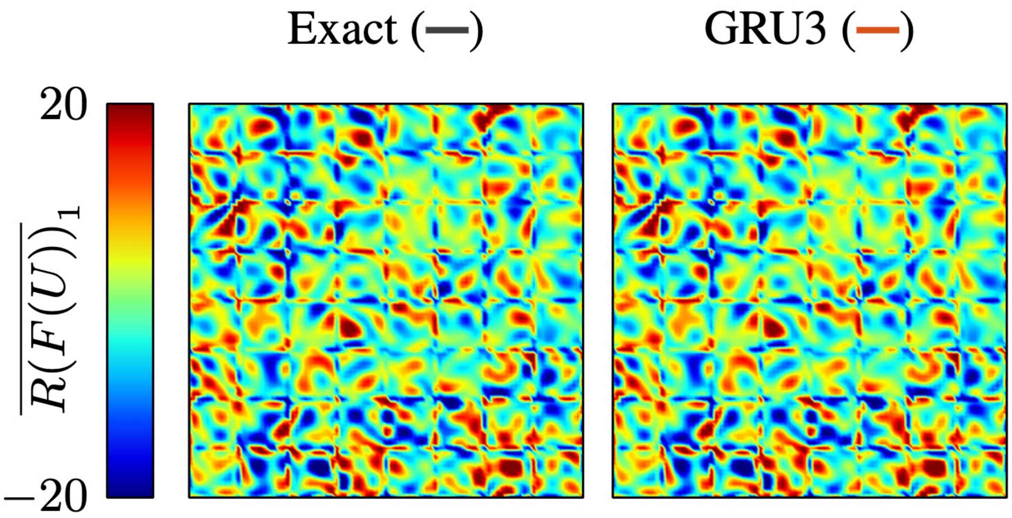
Basics of closure modelling

- Traditionally, closure model are formulated as **closed-form expressions** based on physical arguments
  - Smagorinsky model, gradient model, e.g. $\boldsymbol{c}(\bar{\boldsymbol{u}};\theta) = \nabla \cdot (\nu_T(\bar{\boldsymbol{u}}) S(\bar{\boldsymbol{u}}))$
  - Great interpretability; universal applicability highly limited
- Recent alternative:
  - Use **neural networks** to approximate the commutator error: $\boldsymbol{c}(\bar{\boldsymbol{u}};\theta) = \mathrm{NN}(\bar{\boldsymbol{u}};\theta)$

$$\theta = \mathrm{argmin}_\theta ||\mathrm{NN}(\bar{\boldsymbol{u}}_{\mathrm{ref}};\theta) - \mathcal{C}[\mathcal{A},\mathcal{F}](\boldsymbol{u}_{\mathrm{ref}})||_2^2$$

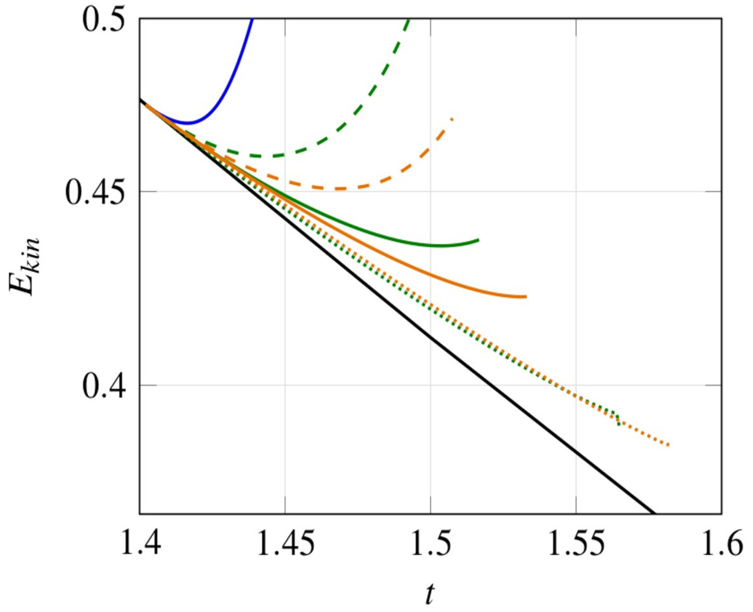
  - Issue: **difficult to get stable results**

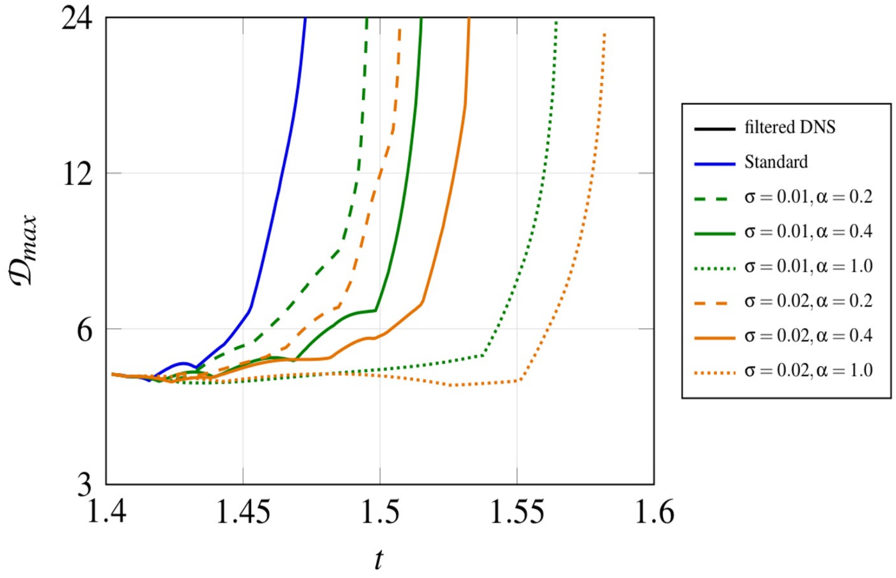
Neural-networks give great match...



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

… but give instabilities in the dynamical system





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

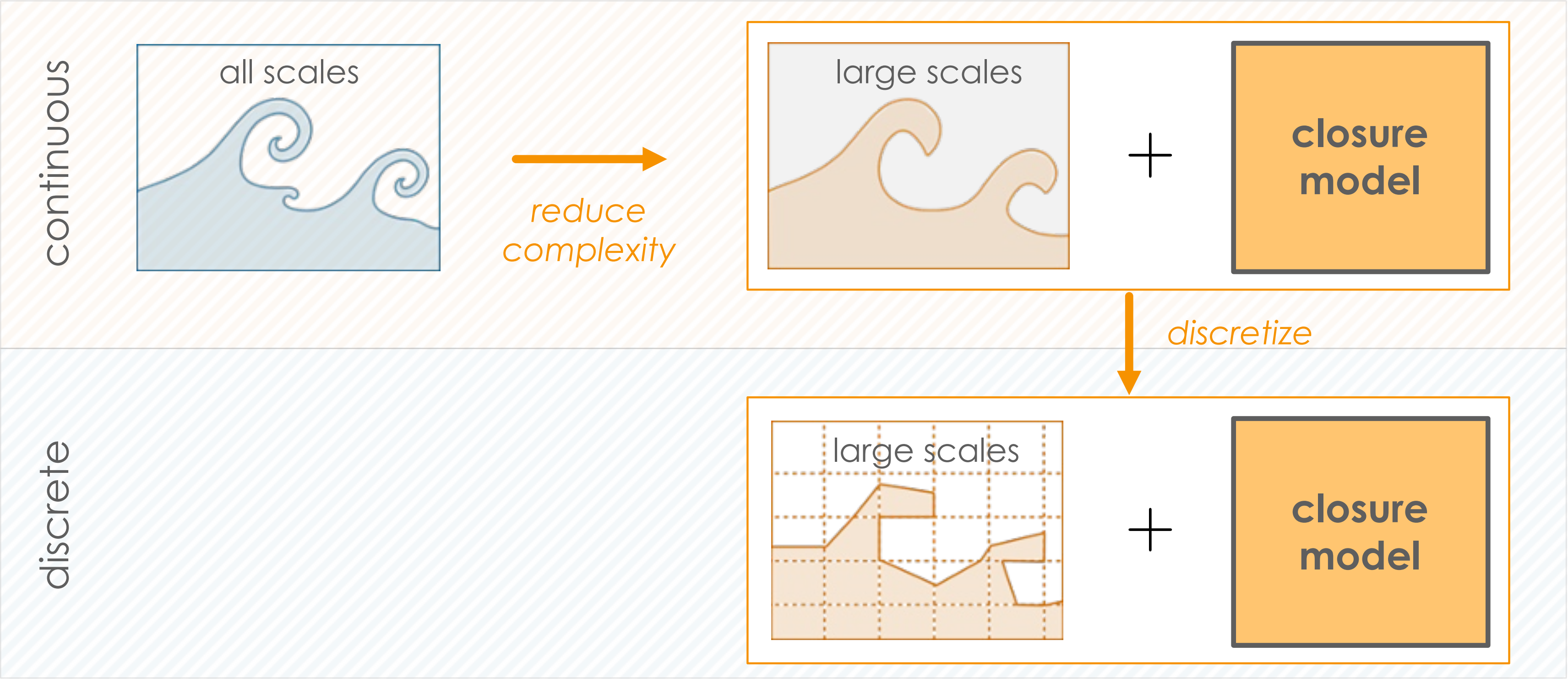
Tackling instability in dynamical systems with NNs

- “**Model-data inconsistency**” and **instability** common problem for ML-based closure models (mismatch training environment and prediction environment)

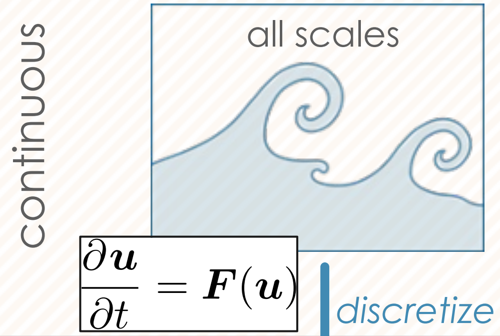
- Recent approaches:
  - Stability training on data with artificial noise (Kurz & Beck, 2021)
  - Minimizing (or eliminating) backscatter (Park & Choi, 2021)
  - Projection onto a stable basis (Beck et al., 2019)
  - Trajectory fitting (List et al., 2022; MacArt et al., 2021)
  - Reinforcement learning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

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

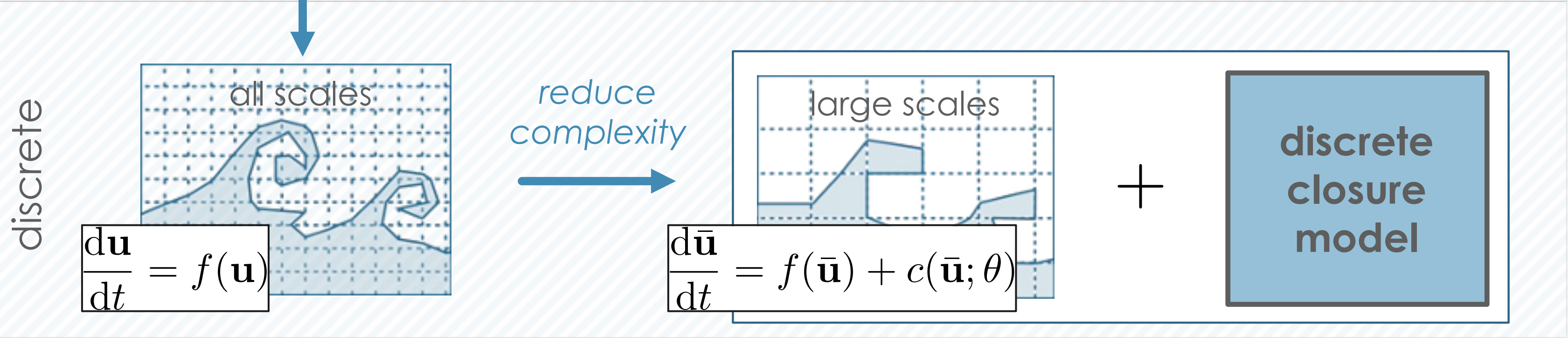
Common approach in closure modelling



New approach: discretize first



- Consistency training data and prediction environment
- Inverse problem easier on discrete level
- Closure models tailor-made to discretization
- Less regularity requirements
- Easier to handle boundary conditions



Example of “discretize first”: inferring a parameter

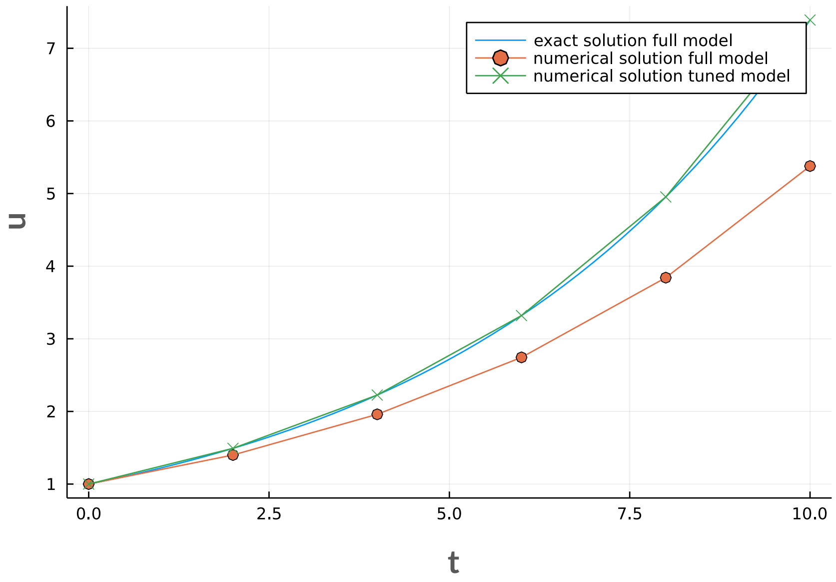
- Problem: find θ in the ODE $$\frac{\mathrm{d}u}{\mathrm{d}t} = \theta u$$
- Given: initial condition and reference solution $u_{\mathrm{ref}}(T)$
- Forward Euler discretization $$u^n = (1 + \Delta t\theta)^n u(0)$$
- Minimize loss function $$\begin{aligned}\mathcal{L}(u^n(\theta), u_{\mathrm{ref}}(T)) &= (u^n - u_{\mathrm{ref}}(T))^2 \\ &= ((1 + \Delta t\theta)^n u(0) - u_{\mathrm{ref}}(T))^2\end{aligned}$$

Example of “discretize first”: inferring a parameter

- True value: $\theta^* = 0.2$
- Forward Euler:

$$\theta_{\mathrm{FE}} = \frac{1}{\Delta t}\left(\left(\frac{u_{\mathrm{ref}}(T)}{u(0)}\right)^{1/n} - 1\right) \approx 0.245$$

- The “incorrect” parameter gives the **exact solution: it corrects the discretization error**



Example of “discretize first”: inferring a parameter

- Problem: find θ in the ODE

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = \underbrace{\begin{pmatrix} 0 & \theta \\ -\theta & 0 \end{pmatrix}}_{A(\theta)} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}$$

- Given: initial condition and reference solution $u_{\mathrm{ref}}(T)$

- RK3 discretization

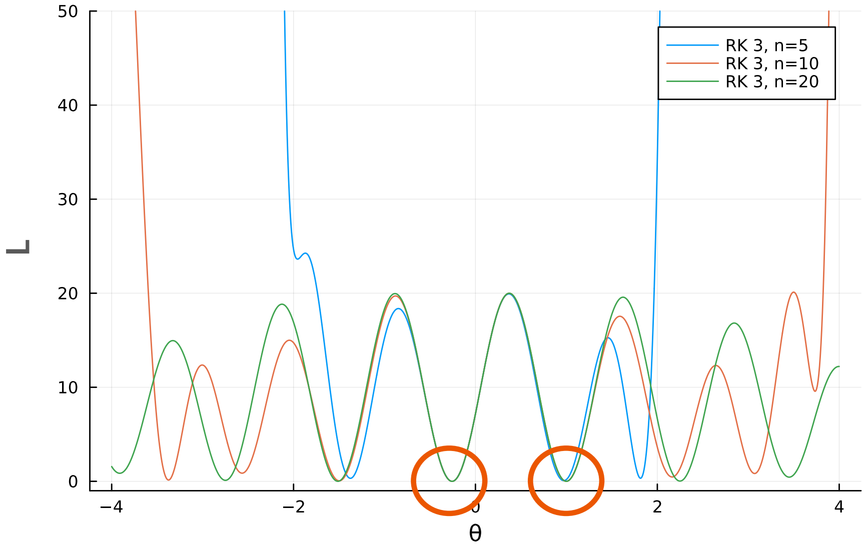
$$u^n = (I + \Delta t A(\theta) + \frac{1}{2}\Delta t^2 A(\theta)^2 + \frac{1}{6}\Delta t^3 A(\theta)^3)^n u(0)$$

- Minimize loss function

$$\mathcal{L}(u^n(\theta), u_{\mathrm{ref}}(T)) = \|u^n(\theta) - u_{\mathrm{ref}}(T)\|_2^2$$

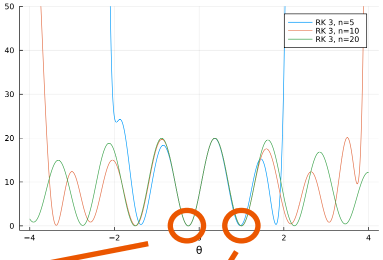
Example of "discretize first": inferring a parameter

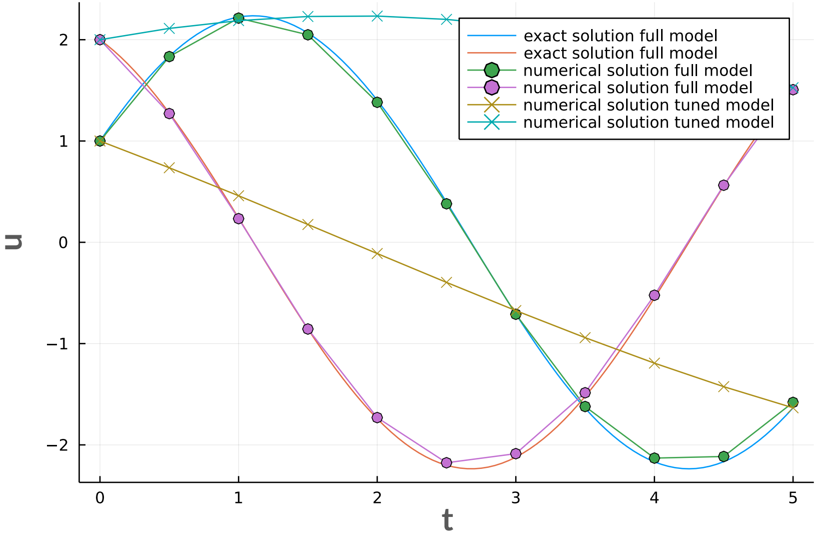
- Loss function high-order polynomial in θ
- Multiple local minima - *aliasing*
- Number of minima *increases* with number of time steps and with order of RK scheme

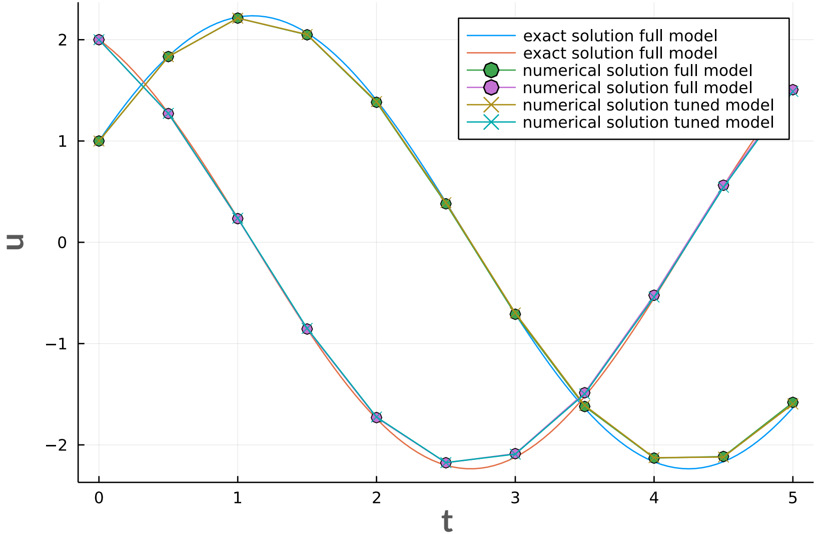


Inferring a parameter

- Loss function choice important
- Local minima can be tricky







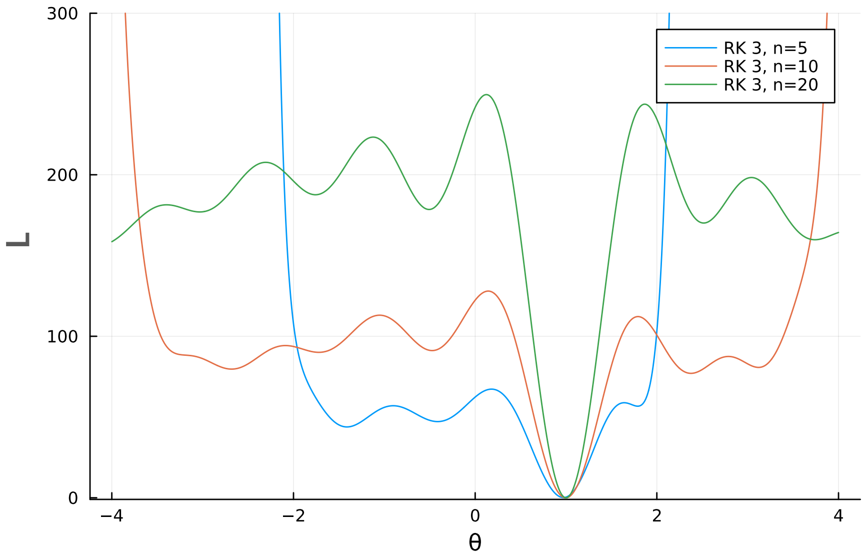
Example of "discretize first": inferring a parameter

- Adapt loss function

$$\mathcal{L}(u^n(\theta), u_{\text{ref}}) = \sum_{i=1}^{N_t} \|u^i(\theta) - u_{\text{ref}}(t_i)\|_2^2$$

- Clear global minimum

- We call this "**trajectory fitting**" – (more about this later)



Examples of preserving structure

- ODE formulation ("neural ODE")

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

- Closure model form ("neural closure model")

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

- Conservation

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

- Translation invariance

CNN architecture

- **Energy conservation**

Energy conservation implies stability

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

$$\nabla \cdot \boldsymbol{u} = 0$$

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

$$\Longrightarrow$$

$$\frac{\mathrm{d}K}{\mathrm{d}t} = -\nu \int_\Omega \nabla \boldsymbol{u} : \nabla \boldsymbol{u} \, \mathrm{d}\Omega$$

$$K := \frac{1}{2} \int \boldsymbol{u} \cdot \boldsymbol{u} \, \mathrm{d}\Omega$$

Idea: impose a similar structure on the filtered equations

Korteweg - de Vries equation

- Shallow water waves, solitons:

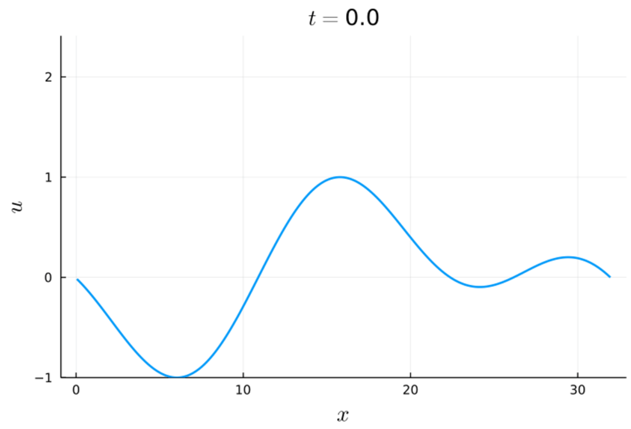
$$\frac{\partial u}{\partial t} + 3\frac{\partial u^2}{\partial x} = -\frac{\partial^3 u}{\partial x^3}$$

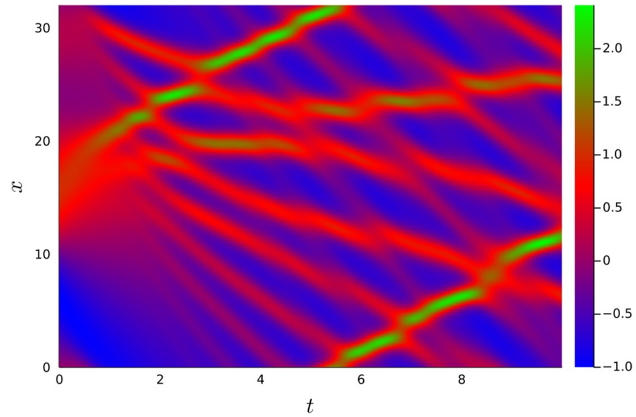
- Energy conservation (periodic BCs):

$$\frac{\mathrm{d}E}{\mathrm{d}t} = \frac{\mathrm{d}}{\mathrm{d}t}\underbrace{\frac{1}{2}\int_\Omega u^2 d\Omega}_{=:E} = 0$$

- Discretized using skew-symmetric scheme:

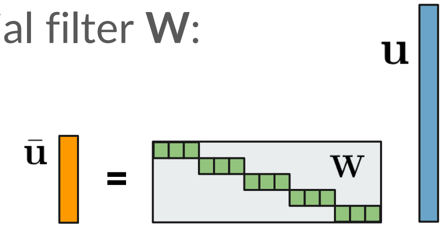
$$\frac{\mathrm{d}\mathbf{u}}{\mathrm{d}t} = -3\mathbf{G}(\mathbf{u}) - \mathbf{D}_3\mathbf{u} \qquad (\mathbf{u}, \frac{\mathrm{d}\mathbf{u}}{\mathrm{d}t}) = 0$$



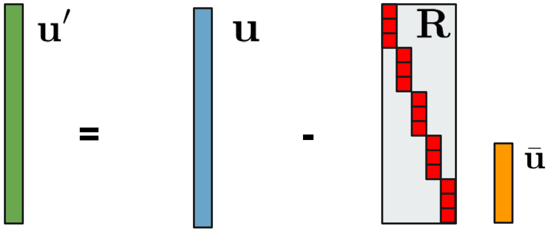


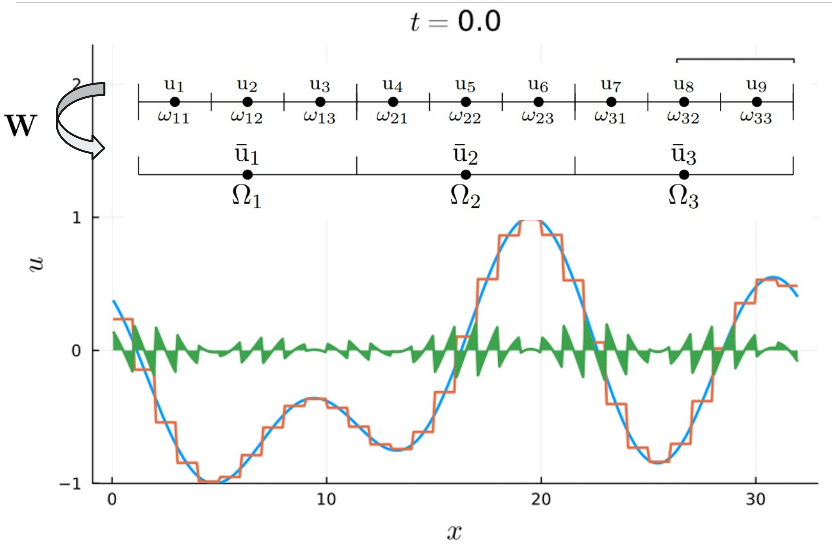
Discrete filtering and reconstruction

- Spatial filter **W**:



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





subgrid scales important near sharp gradients

Energy decomposition

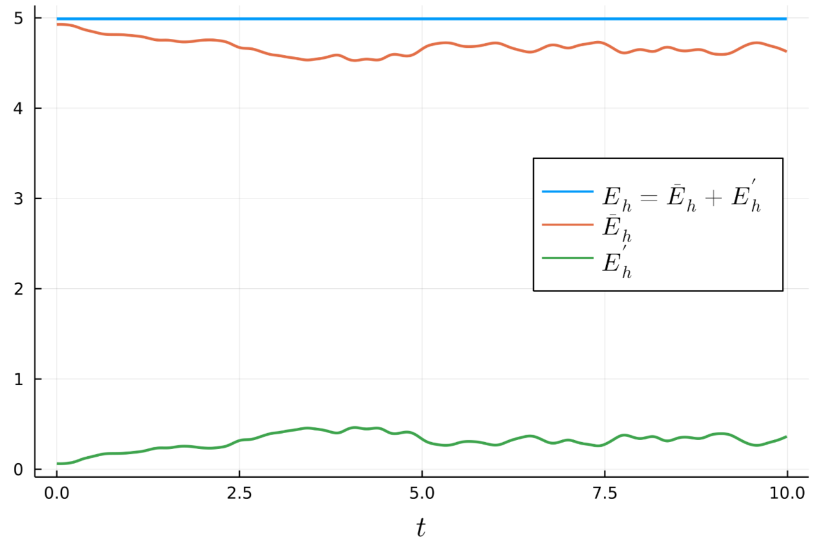
- Since **W R = I**, we can decompose the energy as:

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

- 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 stability we need information about the small scales**



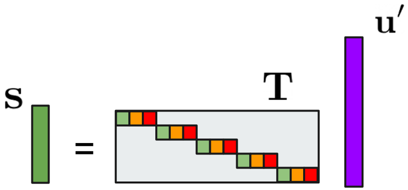
Total energy conserved, **large-scale energy not**

Subgrid compression

- Simulating **u'** is not feasible.
- Replace **u'** by **compressed (coarse-grid) variable s**

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

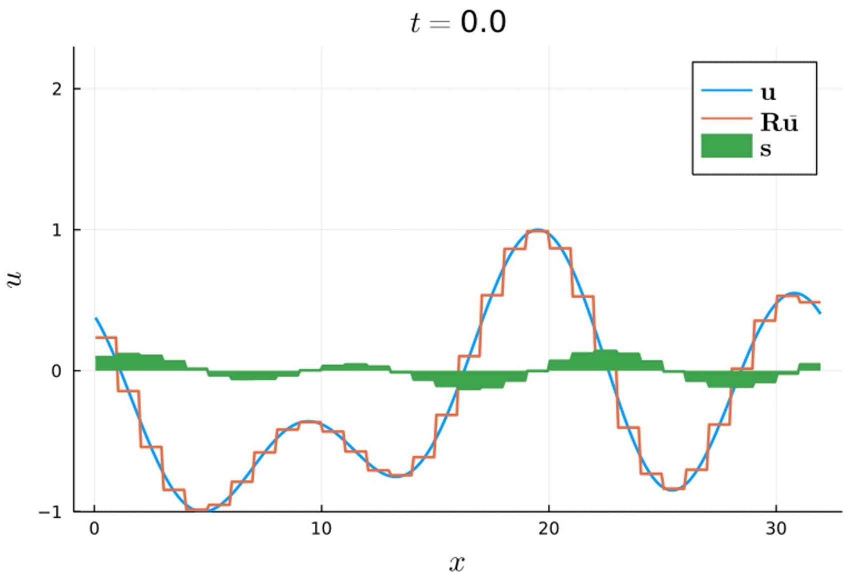
with linear compression **T**



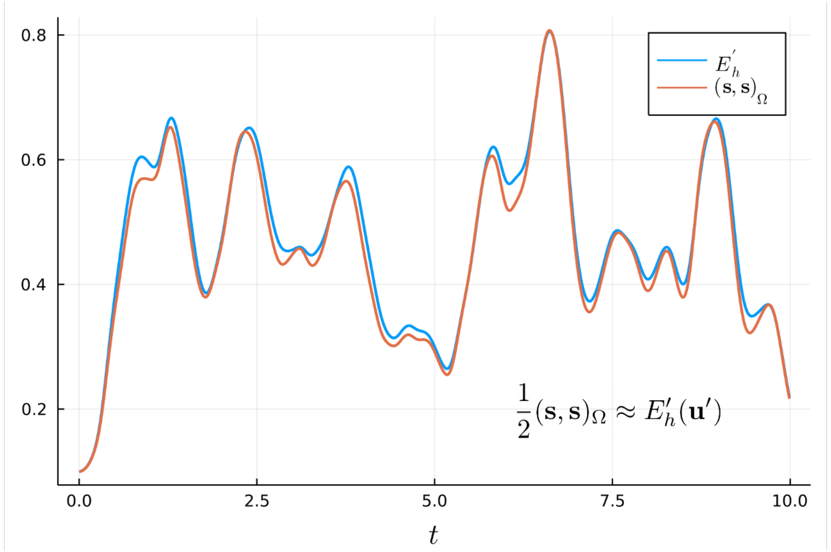
learned from

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

Compressed variables learn effective subgrid content



compressed subgrid variable identifies sharp gradients



learned compression matches small scale energy closely

Energy-conserving closure model

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

- Large scale dynamics with closure model
- Compressed small scale dynamics (latent variables)

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

"extended neural closure model"

- Energy conserving condition

$$\frac{\mathrm{d}\bar{E}_h(\bar{\mathbf{u}})}{\mathrm{d}t} + \frac{1}{2}\frac{\mathrm{d}(\mathbf{s},\mathbf{s})_\omega}{\mathrm{d}t} = 0$$

- Our proposal: learn **a skew-symmetric matrix** $\mathcal{K}$ with entries given by neural network outputs

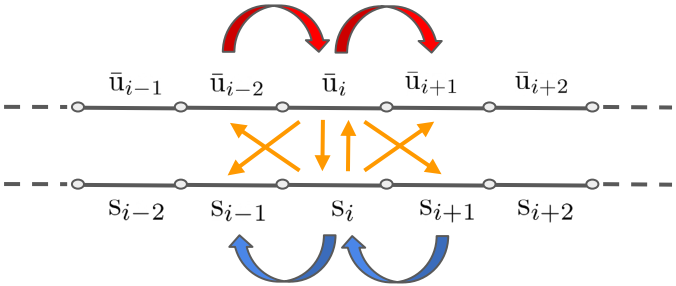
$$\begin{bmatrix}c_u(\bar{\mathbf{u}},\mathbf{s};\theta_u)\\ c_s(\bar{\mathbf{u}},\mathbf{s};\theta_s)\end{bmatrix} = \mathcal{K}(\bar{\mathbf{u}},\mathbf{s};\mathbf{\Theta})\begin{bmatrix}\bar{\mathbf{u}}\\ \mathbf{s}\end{bmatrix}$$

Skew-symmetric neural network

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

- Intuition behind skew-symmetric closure model: **local energy exchanges**

$$\mathcal{K}(\bar{\mathbf{u}}, \mathbf{s}; \mathbf{\Theta}) = \begin{bmatrix} \mathbf{K}_1 & \mathbf{K}_2 \\ -\mathbf{K}_2^T & \mathbf{K}_3 \end{bmatrix}$$



- Skew-symmetric forms obtained by

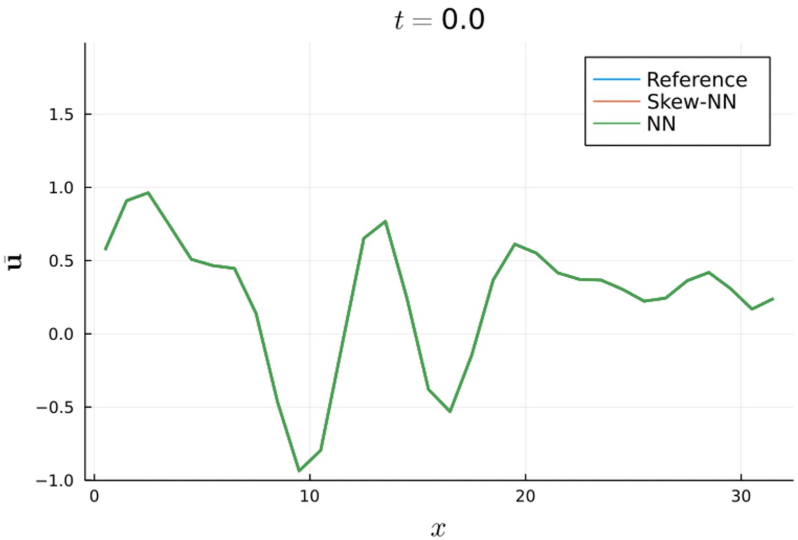
$$\mathbf{K}_1 = [\mathbf{M}_1(\theta), \mathbf{\Phi}_1(\theta), \mathbf{M}_2(\theta)]$$

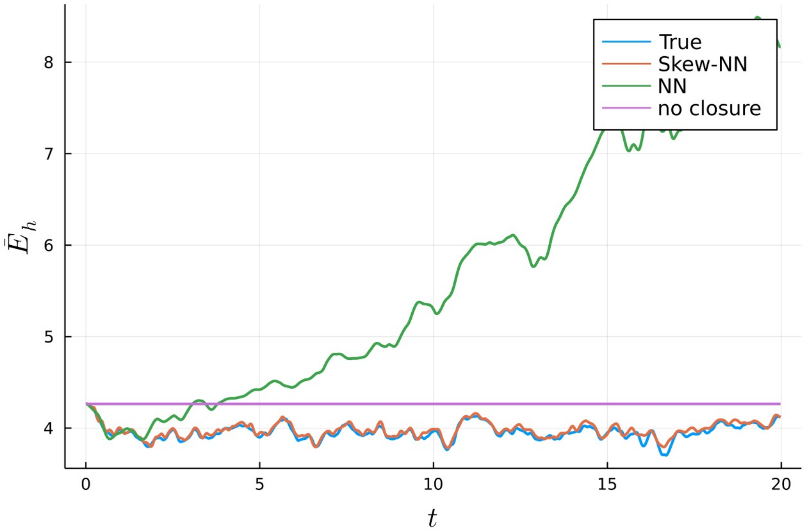
$$[\mathbf{A}, \mathbf{\Phi}, \mathbf{B}] := \mathbf{A}\mathbf{\Phi}\mathbf{B}^T - (\mathbf{A}\mathbf{\Phi}\mathbf{B}^T)^T$$

- $\mathbf{K}_2$ allows energy exchange between large and small scales

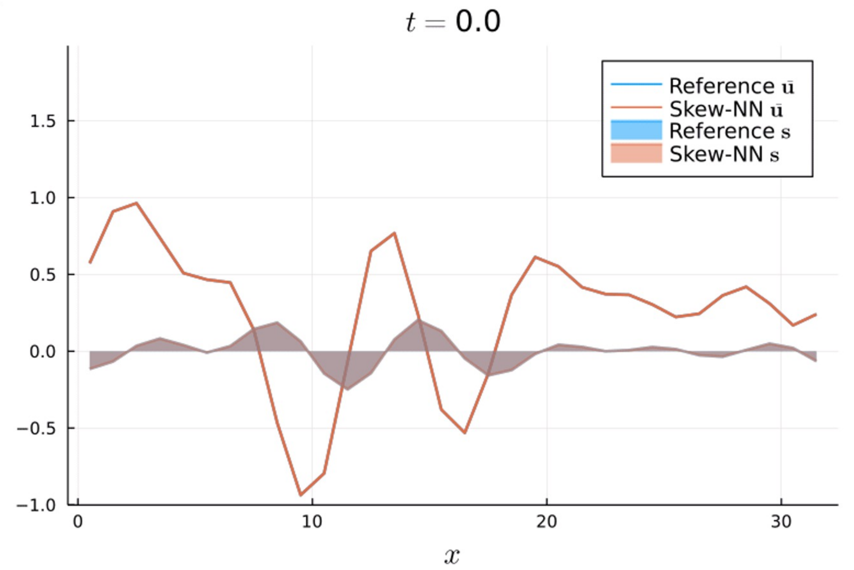
New closure model improves quality + stability

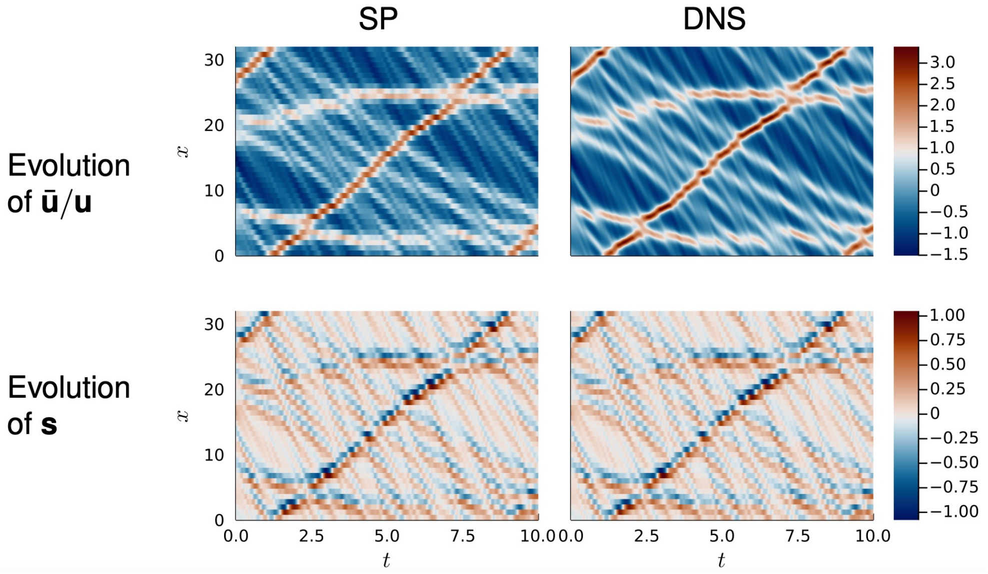
- Trained on different initial conditions, tested on unseen initial conditions
- Reduction from N = 600 to N = 30
- Compare to standard CNN





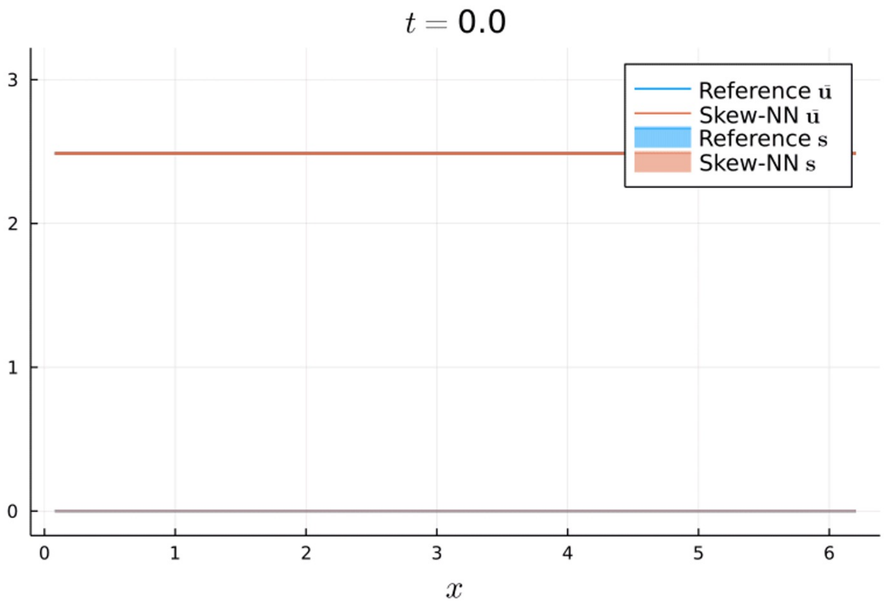
Evolution of subgrid content matches nicely





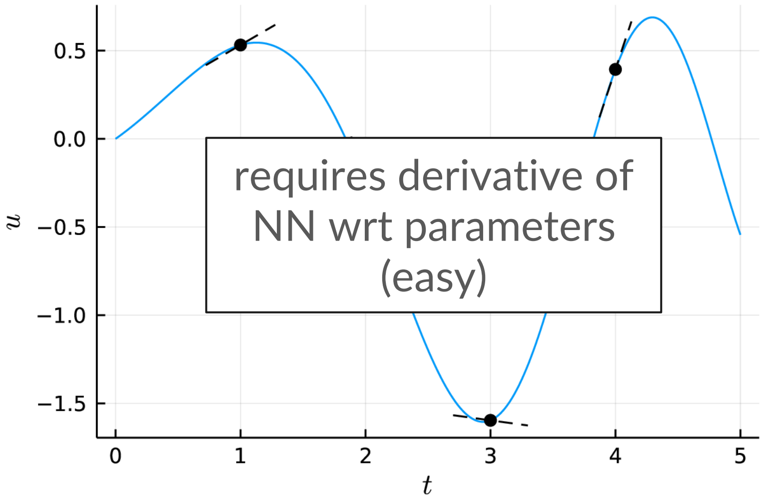
Extension to Burgers' equation

- Includes viscosity and time-dependent boundary conditions
- Reduction from N=1000 to N=40



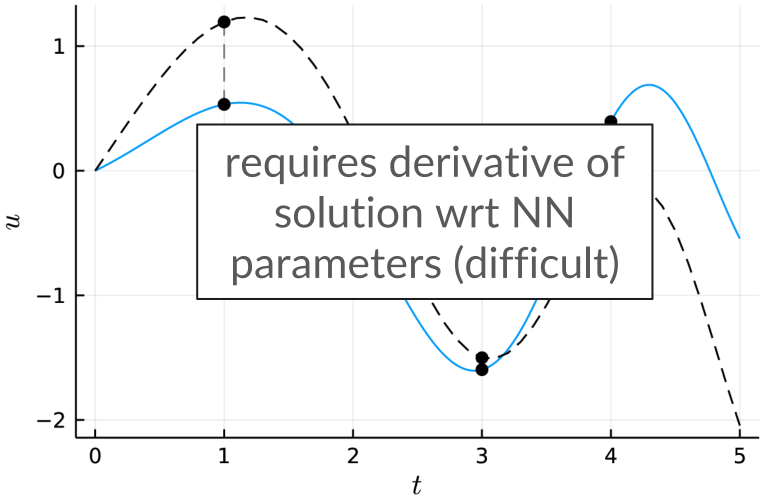
What about training neural closure models?

Training approaches for neural closure ODEs



derivative fitting

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



trajectory fitting

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

$$\text{Loss} = \left\| \left(\frac{d\mathbf{u}}{dt}\right)_{\text{ref}} - \text{NN}(\mathbf{u}_{\text{ref}}; \vartheta) \right\|^2$$

Derivative fitting can be inaccurate (and unstable)

Theorem 3.2. *Let $\mathbf{u}_{\text{ref}}(t), t \geq 0$ be given, and let $\mathbf{u}(t), t \geq 0$ be the solution of the ODE $\frac{d\mathbf{u}}{dt} = NN(\mathbf{u}; \vartheta)$. If the following holds:*

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

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

then the following error bound holds:

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

Based on the “Fundamental Lemma”, Hairer et al. (1993)

If a neural ODE:

- is given a good initial condition;
- approximates the derivative well and is Lipschitz;

Then, the resulting ODE solution may still be **inaccurate**

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

Trajectory fitting ("embedded learning")

- Trajectory fitting yields **stable results, tailor-made** to the discretization
- Derivatives of loss function computed via **sensitivity methods** $\frac{\mathrm{dLoss}}{\mathrm{d}\theta}$

1. Discretise-then-optimise:
   - Need **differentiable solver** (not always available, e.g. black box code)
2. Optimise-then-discretise
   - Solve adjoint equations (Chen et al. 2018)

$$\begin{cases} \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{y}^\top & = -\mathbf{y}^\top \frac{\partial}{\partial \mathbf{u}} \text{NN}(\mathbf{u}(t);\vartheta) \\ \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{z}^\top & = -\mathbf{y}^\top \frac{\partial}{\partial \vartheta} \text{NN}(\mathbf{u}(t);\vartheta) \end{cases}$$

$$\frac{\mathrm{dLoss}}{\mathrm{d}\theta} = \mathbf{z}(0)$$

Comparison of approaches

trajectory fitting

	Derivative fitting	Discretise-then-optimise	Optimise-then-discretise
Terms that must be differentiable	NN	NN, f, and ODE solver	NN and f
Accuracy of computed gradients of loss function	Exact	Exact	Approximate
Can learn long-term accuracy	No	Yes	Yes
Requires time-derivatives of training data	Yes	No	No
Computational cost	Low	High	High

Several issues / design choices:

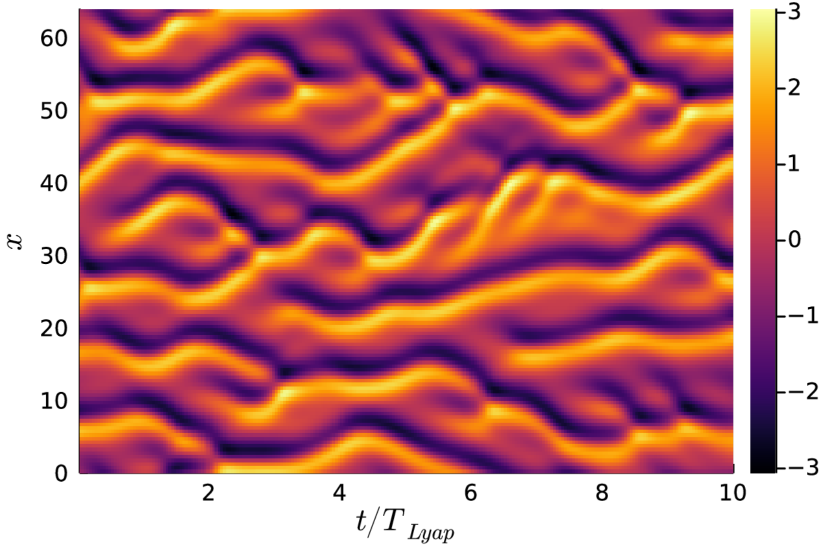
- Trajectory length / “unrolled time steps” in loss function
- Stiffness (backpropagation with implicit solvers more difficult)
- Chaotic systems
- Exploding /vanishing gradients

Kuramoto-Sivashinsky equation

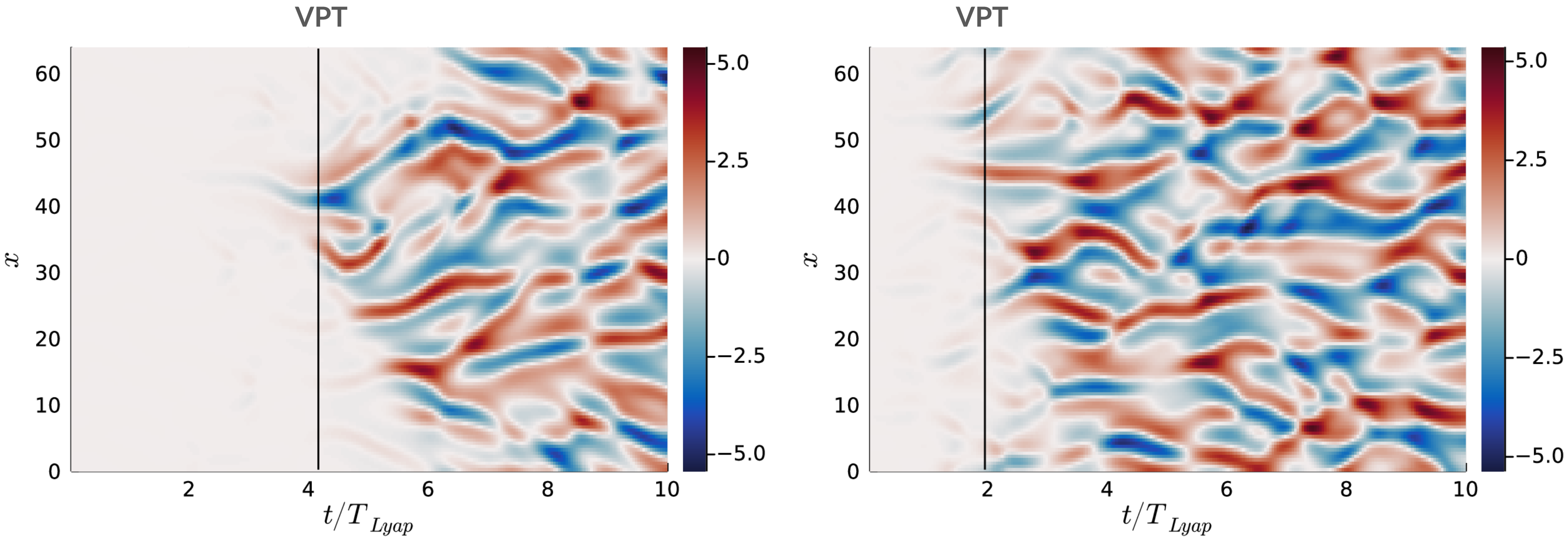
$$\frac{\partial u}{\partial t} + \frac{1}{2}\frac{\partial}{\partial x}\left(u^2\right) = -\frac{\partial^2 u}{\partial x^2} - \frac{\partial^4 u}{\partial x^4}$$

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

- **Chaotic:**
  - Use Valid Prediction Time (VPT) to assess accuracy
  - Weighting of loss function to damp exponential increase in sensitivity
- **Stiff:**
  - Opt-Disc: **implicit** ESDIRK KenCarp47
  - Disc-Opt: **explicit** ETDRK4 in Fourier domain (Kassam & Trefethen 2005)
- Reduction 1024 -> 128



Effect of trajectory length, optimise-then-discretise



short trajectories

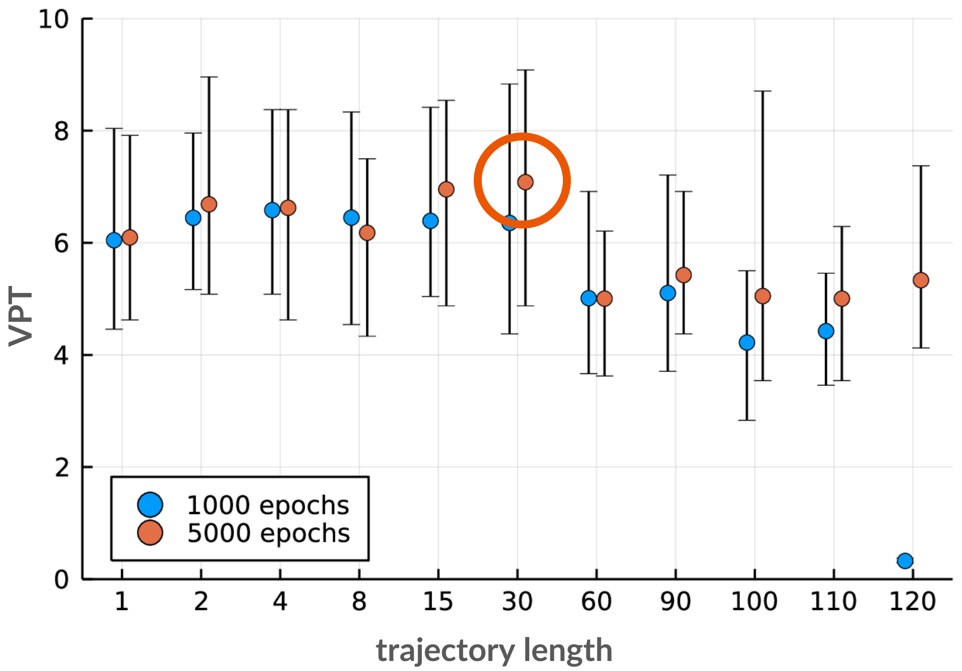
long trajectories

Valid prediction time, optimise-then-discretise

Training method		VPT Min	VPT Avg	VPT Max
Coarse ODE		1.17	1.93	3.00
Derivative fitting		4.17	5.36	7.54
Optimise-then-discretise	Short trajectories	4.08	5.84	8.29
	Long trajectories	2.38	3.38	4.67
Long trajectories, decaying error weights	$c = 0.5$	2.42	4.20	5.38
	$c = 1.0$	2.96	4.38	6.29
	$c = 1.5$	3.29	4.58	5.88
	$c = 2.0$	2.71	4.29	5.75

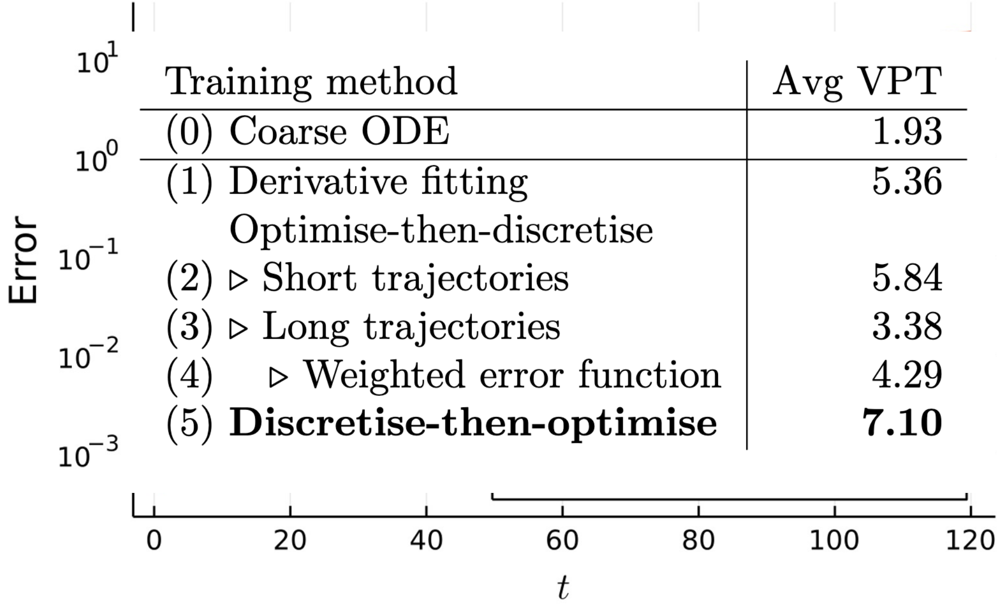
Effect of trajectory length, discretise-then-optimise

- Discretise-then-optimise higher VPT than optimise-then-discretise
- In both cases: **trajectories should not be 'too long'**



Comparison of training approaches

- **Discretise-then-optimise overall best performance**
- Optimise-then-discretise sensitive to training interval; longer interval less accurate
- Derivative fitting reasonable but diverges (for Burgers: unstable)



Training method	Avg VPT
(0) Coarse ODE	1.93
(1) Derivative fitting	5.36
Optimise-then-discretise	
(2) ▷ Short trajectories	5.84
(3) ▷ Long trajectories	3.38
(4) ▷ Weighted error function	4.29
(5) Discretise-then-optimise	**7.10**

Conclusions

- **“Discretize first”**
  - Tailor-made closure models
  - Useful framework when using neural networks, eases analysis
- **“Preserve structure”**
  - Accuracy improves by adding physics knowledge
  - Non-linear stability possible with energy conserving methods
- **“Embedded learning”** with **trajectory fitting**
  - Discretise-then-optimise with **differentiable solvers** preferred
  - Promising but with 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