LEARNING TURBULENCE CONTROL STRATEGIES WITH DATA-DRIVEN REDUCED-ORDER MODELS AND DEEP REINFORCEMENT LEARNING

Kevin Zeng
Department of Chemical and Biological Engineering
University of Wisconsin - Madison
1415 Engineering Dr, Madison, WI 53706
kzeng3@wisc.edu

Alec J. Linot
Department of Chemical and Biological Engineering
University of Wisconsin - Madison
1415 Engineering Dr, Madison, WI 53706
linot@wisc.edu

Michael D. Graham
Department of Chemical and Biological Engineering
University of Wisconsin - Madison
1415 Engineering Dr, Madison, WI 53706
mdgraham@wisc.edu

ABSTRACT

Design of active control strategies for turbulent drag reduction is a challenging task due to the complex dynamics and the difficulty in devising good control targets. Deep reinforcement learning (RL), an emergent machine learning method capable of learning complex control strategies for high-dimensional systems from data, is able to address broad macroscopic control objectives, such as drag minimization, making it promising for flow control application. However, the iterative RL process requires vast amounts of data to be generated from interactions with the target system. For high-dimensional and computationally demanding simulations, such as direct numerical simulations (DNS), this data generation step can be prohibitively expensive.

We mitigate this challenge in a completely data-driven fashion by combining data-driven reduced-order models (ROM) of the flow system with deep RL. We construct our ROMs by combining an undercomplete autoencoder with a neural ordinary differential equation (ODE)–both of which are trained directly from data. The autoencoder compresses the high-dimensional state representation into a low-dimensional representation while the neural ODE predicts the system dynamics as an ODE in this new representation. This ROM is substituted in place of the true system during RL training to accelerate the learning process. The ROM-based control strategy is then deployed to the flow system for control validation.

We first apply our method to the Kuramoto-Sivashinsky equation (KSE), a 1D turbulence proxy that exhibits spatiotemporal chaos, equipped with actuators. We demonstrate that we are capable of learning a ROM of the actuated dynamics and with a control goal analogous to drag reduction, we show that the ROM-based RL strategies perform well in the KSE. We highlight that the RL agent discovers and stabilizes a forced equilibrium solution. Next, we apply our method to a DNS of Couette flow with control in the form of jets that modify the wall-normal velocity at the wall. We find a ROM that captures key short-time behavior of the underlying system using drastically fewer dimensions, and we show RL forces the system to relaminarize far more frequently than the underlying system.

Data-Driven Reduced Order Modeling

The first step in our procedure is to learn a ROM surrogate of the system we aim to control. We assume we have access to time series data $[s(t_0), s(t_1), ..., s(t_M)]$ of the system and that the dynamics of the system lie (or approximately lie) on some low-dimensional manifold $\mathscr{M}$ embedded in $\mathbb{R}^d$ (i.e. $s \in \mathscr{M} \subset \mathbb{R}^d$); an assumption that has been affirmed for many dissipative systems (Temam, 1989). We can find a parameterization $h \in \mathbb{R}^{d_h}$ that is of a much smaller dimension than the ambient space $d_h \ll d$ (when the dimension is minimal we write $d_{\mathscr{M}}$ in place of d_h). We then formulate the time-evolution of h with an ODE. Once constructed, the ROM forecasts trajectories of h (which we can map to the ambient space s) and uses these trajectories to train the RL agent. Figure 1 (a)-(c) outlines the learning procedure for the ROM.

To find the ROM we first use an undercomplete autoencoder to learn the parameterization of the manifold h. An autoencoder consists of an encoding function $h = \chi(s; \theta_E)$ and a decoding function $s = \check{\chi}(h; \theta_D)$. Both of these functions are constructed by neural networks (NN) with weights $\theta = [\theta_E, \theta_D]$ that are trained simultaneously to minimize the reconstruction loss $L = 1/M \sum_i^M ||s(t_i) - \check{\chi}(\chi(s(t_i)))||^2$. We compute the gradient of this loss with respect to the NN parameters ($dL/d\theta$) and use stochastic gradient descent to update these parameters.

A challenge with this approach is determining the necessary degrees of freedom to fully parameterize the manifold on which the the data lies. Whitney's Embedding Theorem proves as long as the encoding maps to $\mathbb{R}^{2d_{\mathscr{M}}}$ then h is homeomorphic to s (Whitney, 1936). However, $d_{\mathscr{M}}$ is unknown a priori, so we must find a way to determine it. We estimate this dimension by tracking the error of the autoencoder as a function of dimension, where we have empirically observed a dramatic improvement in performance once d_h agrees with $d_{\mathscr{M}}$ (Linot & Graham, 2020, 2021).

After finding a reduced-dimensional representation h, we next find a dynamical system for h. We chose to learn an ODE for the dynamics of h such that $\dot{h} = g(h, a; \theta_{\text{ODE}})$, where g is a NN and a is the action taken by the controller. The action must be input here because the RL agent iteratively interacts

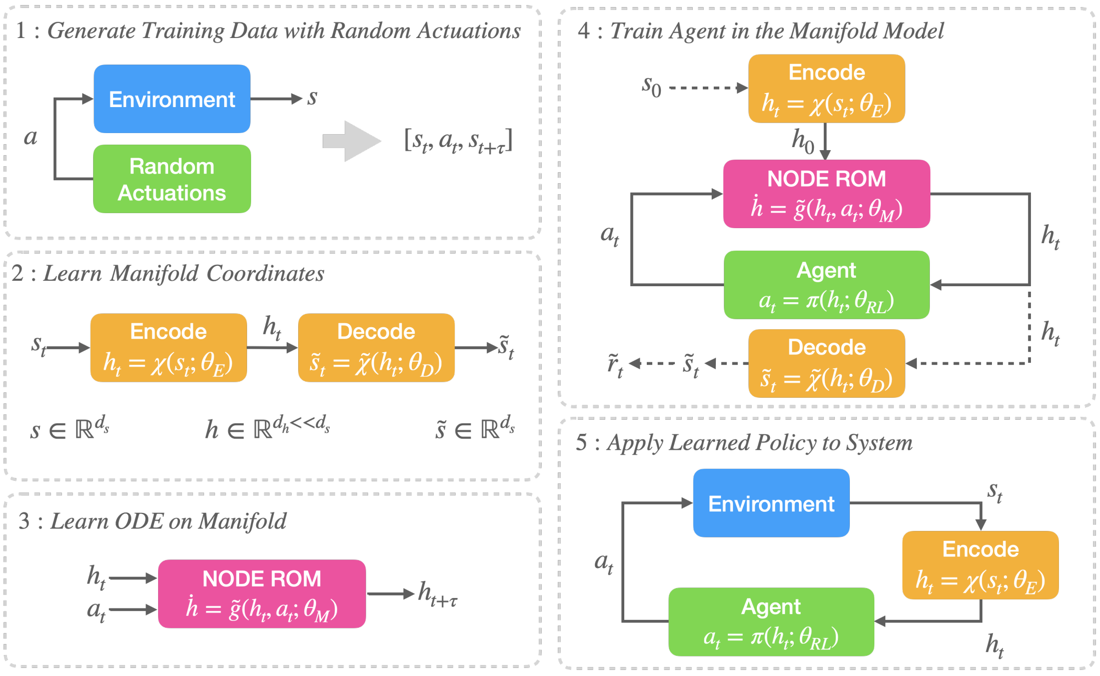
1 : Generate Training Data with Random Actuations
Environment
Random Actuations
a
s
$[s_t, a_t, s_{t+\tau}]$
2 : Learn Manifold Coordinates
Encode $h_t = \chi(s_t; \theta_E)$
Decode $\tilde{s}_t = \tilde{\chi}(h_t; \theta_D)$
s_t h_t $\tilde{s}_t$
$s \in \mathbb{R}^{d_s}$ $h \in \mathbb{R}^{d_h << d_s}$ $\tilde{s} \in \mathbb{R}^{d_s}$
3 : Learn ODE on Manifold
NODE ROM $\dot{h} = \tilde{g}(h_t, a_t; \theta_M)$
h_t a_t $h_{t+\tau}$
4 : Train Agent in the Manifold Model
s_0
Encode $h_t = \chi(s_t; \theta_E)$
h_0
NODE ROM $\dot{h} = \tilde{g}(h_t, a_t; \theta_M)$
Agent $a_t = \pi(h_t; \theta_{RL})$
Decode $\tilde{s}_t = \tilde{\chi}(h_t; \theta_D)$
a_t h_t h_t $\tilde{r}_t$ $\tilde{s}_t$
5 : Apply Learned Policy to System
Environment
Encode $h_t = \chi(s_t; \theta_E)$
Agent $a_t = \pi(h_t; \theta_{RL})$
s_t a_t h_t

Figure 1: Procedure for learning a NODE-ROM from data, combining it with RL to approximate a control policy, and deploying the approximate policy back to the true system

with the system to determine the best control policy. Using this ODE we predict new states by integrating forward in time

$$\tilde{h}(t_i+\tau) = h(t_i) + \int_{t_i}^{t_i+\tau} g(h(t), a; \theta_{\text{ODE}}) dt. \tag{1}$$

We train g by minimizing the difference between the prediction ($\tilde{h}(t_i+\tau)$) and the known state ($h(t_i+\tau)$), in the manifold coordinates, giving the loss $J = \sum_i^M ||h(t_i+\tau) - \tilde{h}(t_i+\tau)||^2$ (Chen *et al.*, 2019).

Learning a Control Strategy From Data

We use deep RL to learn control strategies from data. The general deep RL framework is a cyclic interactive learning process. During each cycle, the RL agent, the embodiment of the control policy represented by neural-network $a_t = \pi(s_t; \theta_{RL})$, outputs a control action, a_t, given a state observation of the system, s_t. The impact of this action on the system is then quantified by a scalar reward, r_t, which is defined by the control objective. During training, the agent attempts to learn the mapping between s_t and a_t that maximizes the cumulative long time reward and updates accordingly each cycle. In our ROM-based RL framework, we train the agent in the reduced space of the ROM, such that we instead learn $a_t = \pi(h_t; \theta_{RL})$. During agent deployment to the original system for control validation, we simply precede the agent with the encoder $h_t = \chi(s_t; \theta_E)$ to map state observations, s_t, to h_t as this is coordinate system the agent was trained with. Fig. 1 (d)-(e) outlines the RL training and deployment procedure. We comment that our framework can applied with any general deep RL method.

Example 1: Kuramoto-Sivashinsky Equation

We consider the periodic KSE, given by

$$\frac{\partial v}{\partial t} = -v\frac{\partial v}{\partial x} - \frac{\partial^2 v}{\partial x^2} - \frac{\partial^4 v}{\partial x^4} + f(x,t) \tag{2}$$

where f is the control term defined as in Bucci *et al.* (2019) and corresponds to 4 evenly spaced Gaussian jets,

$$f(x,t) = \sum_{i=1}^{4} \frac{a_i(t)}{\sqrt{2\pi}\sigma_s} \exp\left(-\frac{(x-X_i)^2}{2\sigma_s^2}\right). \tag{3}$$

We evolve trajectories for a domain size of $L = 22$, which exhibits spatiotemporal chaos with a Lyapunov time of $\sim$22 time units. Spatial discretization is performed with Fourier collocation on a mesh of 64 evenly spaced points and in our formulation the state vector u consists of the solution values at the collocation points. The system is time evolved with a third-order semi-implicit Runge-Kutta scheme (Bucci *et al.*, 2019). From this simulation we train autoencoders with size 500 sigmoid activated hidden layers (for encoder, decoder) and neural ODEs on 40,000 snapshots of data ($s_t = u(t)$) separated 0.25 time units apart experiencing random jet actuations. Figure 2 shows the performance of autoencoders as we vary the dimension d_h. For this system, the error is sufficiently low at $d_h = 12$, so we use this autoencoder for the mappings χ and $\check{\chi}$. Furthermore, we emphasize that we achieve orders of magnitude improvement in reconstruction performance compared to PCA while using the same number of dimensions. In this representation of the manifold coordinates, we train a neural ODE using the same dataset. In Figure 3 we show the short-time predictions of this model with random actuations match the true system well for around 1-1.5 Lyapunov times. We also comment that in the absence of actuation, i.e. $a_t = 0$, the ROM performs similarly well, indicating that the ROM is able

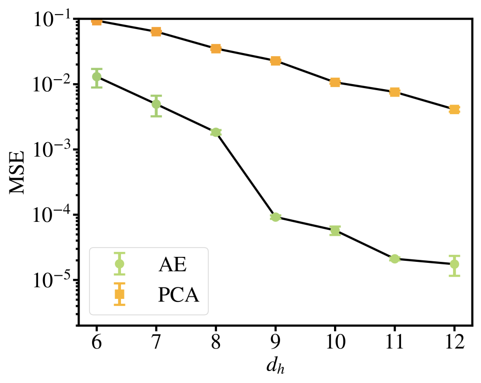

Figure 2: Mean squared error of autoencoders and PCA on test data of the KSE for various dimensions.

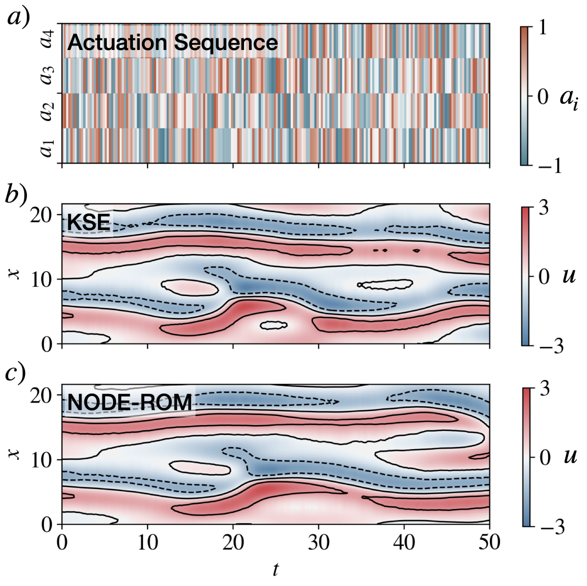

Figure 3: (a) random actuation sequences $a_i(t)$ (b) ground truth KSE trajectory starting from a random initial conditions following actuation sequences in (a), (c) the decoded NODE-ROM trajectory following actuation sequence (a) and the same initial condition in (b).

to capture the underlying natural dynamics absent of control inputs despite having been trained with no control-free data.

We now use this ROM as a surrogate model for training the RL agent. In this demonstration we utilize the Deep Deterministic Policy Gradient (DDPG) RL method (Lillicrap *et al.*, 2016). As an analogue to energy-saving in the flow control problem, we seek to minimize the dissipation and total power input of the KSE. This optimization is realized by maximizing the reward $r = -D + P_f$, where $D = \langle (\frac{\partial^2 u}{\partial x^2})^2 \rangle$ and $P_f = \langle (\frac{\partial u}{\partial x})^2 \rangle + \langle uf \rangle$, respectively. Here $\langle \cdot \rangle$ is the spatial average.

The control agent was trained with 1000 episodes of 100 time units long (i.e. 400 transitions per episode), with each episode beginning from a random on-attractor initial condition of the natural, i.e. unforced, KSE. Jet actuations implemented by the control agent, a_t, were maintained constantly from s_t to $s_{t+\tau}$, where $\tau = 0.25$. In this work the DDPG actor and critic networks utilized ReLU activated hidden layers of size 128 and 64, respectively, followed by tanh and linear activations to the outputs of size 4 and 1, respectively.

To assess the performance of our ROM-based policy, the learned control agent was applied to the NODE-ROM, with an example controlled trajectory shown in Fig. 4a. We note that after a brief control transient, the control agent navigates the NODE-ROM to an equilibrium (steady) state and stabilizes it. The quantities targeted for minimization, D and P_f, estimated from the predicted trajectory $u(t)$, are shown in Figure 4c, revealing that this equilibrium exhibits dissipation much lower than the natural unactuated dynamics. To assess how well this ROM-based control policy transfers to the original KSE (i.e. the true system), the same policy is applied to the true KSE with the same initial condition, as shown in Fig. 4b. We note that the controlled trajectory in the KSE yields not only quantitatively similar transient behavior but also the same low-dissipation equilibrium state as was targeted in the NODE-ROM. The transient behaviors between the two are structurally very similar, although the NODE-ROM displays slightly less strongly damped oscillations as it drives the trajectory to the steady state. The values of D and P_f computed from the true system, shown in Fig. 4d, are nearly identical to that of the NODE-ROM in Fig. 4c.

To demonstrate the robustness of the ROM-based policy, shown in Fig. 4e are the dissipation trajectories of the true KSE beginning from 15 randomly sampled test initial conditions that the ROM-based control agent has not seen before. We highlight that the control agent is able to consistently navigate the system to the same low-dissipation state within ~ 150 time units, with one initial condition requiring ~ 200 time units to converge. Finally we note that although the RL training horizons were only 100 time units long, the control agent is able to generalize to achieve and maintain control well beyond the the horizon it was trained in.

Here we emphasize that the ROM-based policy drives the dynamics to an equilibrium state in both the NODE-ROM and the true KSE, indicating that not only does the NODE-ROM capture this state, but it captures the dynamics leading to it accurately enough such that the RL agent could discover it during training and exploit it in a manner that still translates to the original system. We further emphasize that both the NODE-ROM and agent were never explicitly informed of this low-dissipation state's existence. Finally, we highlight that the discovered low-dissipation state is an unstable state that is stabilized by the control agent. If control is removed, the system returns to the natural chaotic dynamics. The ROM-based RL agent is comparable to that of an RL agent trained directly with original system via direct interactions in both performance and targeting strategy (Zeng & Graham, 2021).

These observations indicate that the RL policy trained on the model transfers very well to the true system. We attribute this performance to the fact that both the NODE-ROM and RL agent operate in Markovian fashion, i.e. even if the model has slight inaccuracies, so long as the modeled dynamics are reasonably accurate this does not matter once the agent makes its new state observation. Returning to the dynamical significance of the low-dissipation equilibrium state discovered and stabilized by the RL agent, a continuation in mean forcing magnitude was performed. To do so, we Newton-solved for equilibrium solutions to the KSE starting with the discovered equilibrium state while gradually decreasing the magnitude of the mean actuation profile to zero, as was done in Zeng & Graham (2021). Solutions identified by this continuation

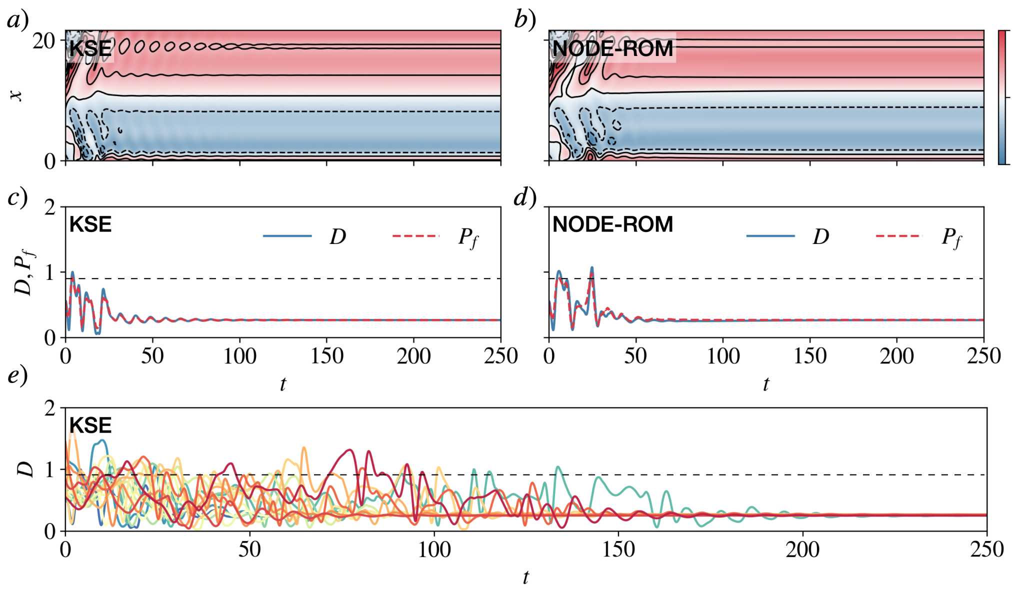

Figure 4: ROM-based RL agent applied to the same initial condition in the a) true KSE and b) data-driven reduced-order model (decoded, $d_h = 12$). The corresponding invariant quantities of dissipation and total input power for the c) true KSE and d) learned reduced-order model. The dashed black line represents the system average of the natural KSE dynamics. e) Controlled dissipation trajectories of the true KSE beginning from 15 randomly sampled test initial conditions.

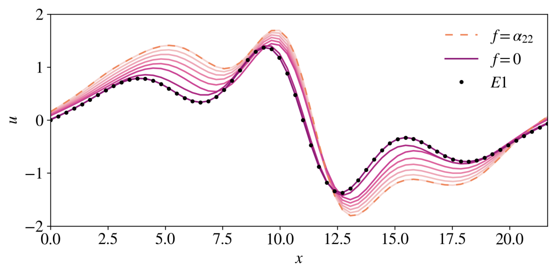

Figure 5: Forcing continuation from the forced equilibrium state (under forcing $f = \alpha_{22}$) discovered by NODE-ROM based RL policy (orange, dashed) to the unactuated KSE system (purple). The known equilibrium E1 of the KSE system is also provided (dots).

in forcing magnitude are shown in Fig. 5, which reveals that equilibrium state captured by the NODE-ROM and discovered by the RL agent is connected to a known existing solution of the KSE known as $E1$ (Cvitanović *et al.*, 2010); we obtained a similar result with an RL agent trained on interactions with the full system (Zeng & Graham, 2021). A similar observation was made for RL control of 2D bluff body flow (Li & Zhang, 2022). We speculate that in systems with complex nonlinear dynamics, the discovery and stabilization of desirable underlying equilibrium solutions (or other recurrent saddle-point solutions such as unstable periodic orbits) of the system may be a fairly general feature of RL flow control approaches. The nonlinear and exploratory nature of RL algorithms facilitates the discovery of such solutions, and since the dynamics are slow near these solutions, little control action should be required to keep trajectories near them.

Example 2: Turbulent Couette Flow

The second system we consider is Plane Couette flow in a channel with two periodic boundary conditions defined by the nondimensionalized incompressible Navier-Stokes Equations.

$$\begin{aligned}\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u}\cdot\nabla\mathbf{u} &= -\nabla P + \mathrm{Re}^{-1}\nabla^2\mathbf{u} \\ \nabla\cdot\mathbf{u} &= 0,\end{aligned} \tag{4}$$

where $\mathbf{u} = [u, v, w]$ is the vector of streamwise, wall-normal, and spanwise velocities, and P is the pressure. The boundary conditions at the wall are $u(\pm1) = \pm1$, $\partial v/\partial y(\pm1) = 0$, $w(\pm1) = 0$, $v(1) = 0$, and $v(-1) = a_i(t) f(x,z)$. This final boundary condition is the actuation, located only at the bottom wall, which is in the form of two spatially localized slot-style jets that inject or suck fluid in the wall-normal direction with zero net-flux. These slot-jets are oriented along the streamwise direction with Gaussian profiles $f(x,z)$ in the spanwise direction with $V_{\max} = \max(|f(x,z)|) = 1/20$ and $a_i(t) \in [-1,1]$. In Fig. 6 we show the wall-normal velocity in the case of $a_0 = 1$ and $a_1 = -1$.

Our code was written in Python to speed up communication with the RL code and follows the same algorithms used by Gibson *et al.* (2008). In particular, we convert the field to Fourier space in x and z and Chebyshev space in y with a resolution of $32x35x32$ Fourier-Chebyshev-Fourier modes. Then we use the multistep Adams-Bashforth/Backward-Differentiation 3 method described in Peyret (2011) for time evolution using a timestep of $\Delta t = 0.02$. This results in a set of implicit Helmholtz equations that we solve at each timestep, and through an influence matrix we compute the pressure required to satisfy $\partial v/\partial y(\pm1) = 0$. This procedure does not result in an incompressible velocity field, so we apply a Tau correction to exactly enforce this constraint. The influence matrix and Tau correction methods are described in Kleiser & Schumann (1980). Unlike the Gibson et al. code,

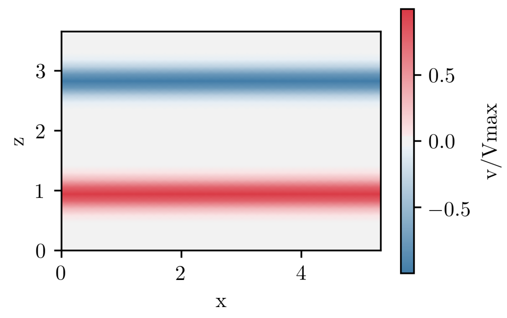

Figure 6: Visualization of the localized wall-normal velocity of the slot-style jets located at $y = -1$.

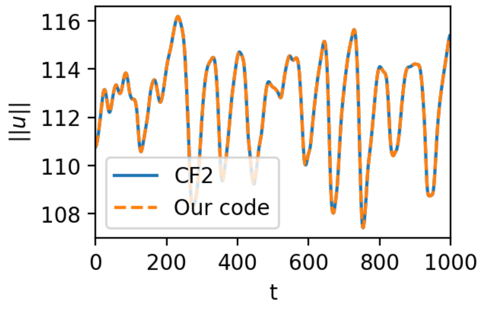

Figure 7: The $L2$ norm of the velocity field, $\mathbf{u}$, of a trajectory produced by Channel Flow 2.0 by Gibson and our python-based Couette DNS code starting from the same initial condition.

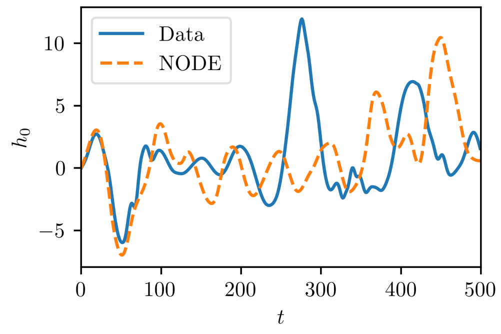

Figure 8: Trajectory of the projection of the Couette flow field onto the first POD mode, and the prediction by the neural ODE.

our code allows us to modify the boundary conditions at the wall by solving for intermediate values in the influence matrix and tau correction methods at each timestep, as opposed to once at the beginning. As illustrated in figure 7, we validated our code by showing that solutions of both codes track nearly exactly for 1000 time units.

We consider Couette flow at a Re $= 400$ and domain size of $[L_x, L_y, L_z] = [7\pi/4, 2, 6\pi/5]$, which are the parameters considered by Gibson *et al.* (2008). For this system we generated 80,000 snapshots of data ($s_t = u(t)$) separated by 1 time

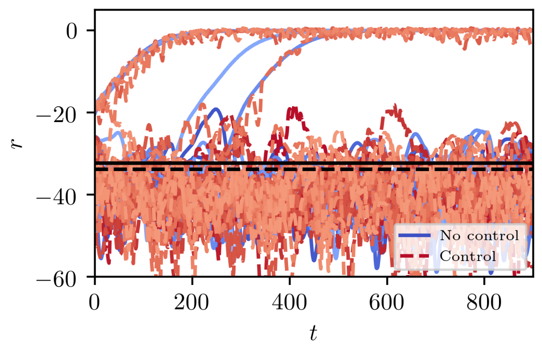

Figure 9: Trajectories beginning from 25 unseen initial conditions evolving under no control (blues) and random slot actuations (reds). The mean drag achieved across the 25 trajectories is shown in black for no control (solid) and random actuation (dashed)

unit. We performed proper orthogonal decomposition (POD) for dimension reduction. In future work an autoencoder will be used. After reducing the dimension with POD, we train neural ODEs with this lower-order representation. Figure 8 shows the trajectory of the 1st POD coefficient for the true system and the ROM where h is constructed from the first 10 POD modes. This preliminary result indicates we can achieve qualitatively similar trajectories with very few degrees of freedom using neural ODEs.

With the success of training RL agents with ROMs of the KSE, and having promising preliminary results with low-dimensional ROMs for Couette flow, our next objective is to train a RL agents on Couette flow ROMs utilizing the Soft Actor-Critic (SAC) RL algorithm (Haarnoja *et al.*, 2018). However, before this, we first learn a control policy directly from the DNS (i.e. direct interactions) to determine a benchmark control performance and timescale required for training. We train our RL agent for several hundred episodes, with each episode lasting for 300 time units and beginning from a turbulent initial condition that does not naturally laminarize in 300 time units. Actions input by the control agent persist for 5 time units, i.e. the agent makes 60 actions per training episode. The reward the agent attempts to maximize is the negative of the drag (normalized such that laminar is 0), which is given by $r = -\int_0^{L_x}\int_0^{L_z} \partial u/\partial y(-1) + \partial u/\partial y(1) - 2dxdz$. Shown in Fig. 9 is the reward vs. time of trajectories generated by our DNS evolving from 25 new unseen initial conditions under no control and with random slot actuations, i.e. the slot actuations were randomly sampled from a uniform distribution of the allowable control range every 5 time units. We observe that in the presence of random slot actuations, the mean drag across the 25 trajectories over a horizon of 900 time units is worse than in the presence of no control. Furthermore, some instances where the system would naturally laminarize are disrupted by the random slot-jets. Shown in Fig. 10 is again the total drag vs. time of Couette flow generated by our DNS evolving from the same 25 new unseen initial conditions under no control and in the presence of our trained RL agent. We observe that with the learned RL control policy, the mean drag across the 25 trajectories over a horizon of 900 time units is drastically lowered compared to no control. We highlight that the RL agent learns to direct the chaotic turbulent dynamics to the laminar state within this window of 900 time units, which

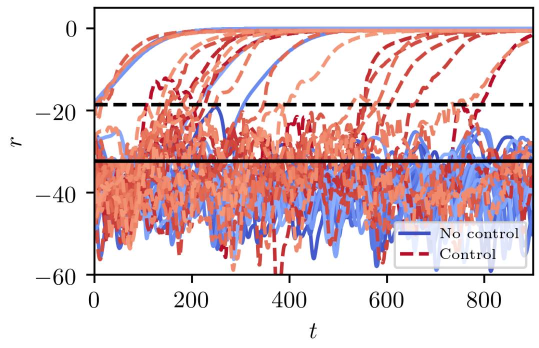

Figure 10: Trajectories beginning from 25 unseen initial conditions evolving under no control (blues) and the RL control agent (reds). The mean drag achieved across the 25 trajectories is shown in black for no control (solid) and the RL agent (dashed).

can be seen by the 19 of 25 trajectories laminarizing due to the control agent. This is a stark improvement compared to the 5 of 25 naturally laminarizing trajectories in the presence of no-control and 4 of 25 in the presence of random slot actuations.

The current accomplishment is the first, to the extent of our knowledge, application of deep RL to learn an active control strategy from a direct numerical simulation of turbulent Couette flow. To train such an agent required 3+ weeks of simulation time on 4 processors. We highlight that our preliminary NODE-ROMs is capable of simulating 500 time units of Couette flow in less than the time it takes to simulate 1 time unit by DNS. In forthcoming work we aim to demonstrate the application of training the RL agent via NODE-ROM in place of the computationally intensive DNS for efficient estimation of control strategies.

REFERENCES

Bucci, M. A., Semeraro, O., Allauzen, A., Wisniewski, G., Cordier, L. & Mathelin, L. 2019 Control of chaotic systems by deep reinforcement learning. *Proc. R. Soc. A.* **475**.

Chen, Ricky T. Q., Rubanova, Yulia, Bettencourt, Jesse & Duvenaud, David 2019 Neural ordinary differential equations. *arXiv preprint arXiv:1806.07366* .

Cvitanović, P., Davidchack, Ruslan L. & Siminos, Evangelos 2010 On the State Space Geometry of the Kuramoto–Sivashinsky Flow in a Periodic Domain. *SIAM Journal on Applied Dynamical Systems* **9** (1), 1–33.

Gibson, J. F., Halcrow, J. & Cvitanović, P. 2008 Visualizing the geometry of state space in plane Couette flow. *Journal of Fluid Mechanics* **611** (1987), 107–130.

Haarnoja, Tuomas, Zhou, Aurick, Abbeel, Pieter & Levine, Sergey 2018 Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor.

Kleiser, L. & Schumann, U. 1980 *Treatment of Incompressibility and Boundary Conditions in 3-D Numerical Spectral Simulations of Plane Channel Flows*, pp. 165–173. Wiesbaden: Vieweg+Teubner Verlag.

Li, Jichao & Zhang, Mengqi 2022 Reinforcement-learning-based control of confined cylinder wakes with stability analyses. *Journal of Fluid Mechanics* **932**, A44.

Lillicrap, Timothy P., Hunt, Jonathan J., Pritzel, Alexander, Heess, Nicolas, Erez, Tom, Tassa, Yuval, Silver, David & Wierstra, Daan 2016 Continuous control with deep reinforcement learning.

Linot, Alec J. & Graham, Michael D. 2020 Deep learning to discover and predict dynamics on an inertial manifold. *Phys. Rev. E* **101**, 062209.

Linot, Alec J. & Graham, Michael D. 2021 Data-driven reduced-order modeling of spatiotemporal chaos with neural ordinary differential equations. *arXiv preprint arXiv:2109.00060* .

Peyret, Roger 2011 *Spectral methods for incompressible viscous flow*. Springer.

Temam, R. 1989 Do inertial manifolds apply to turbulence? *Physica D: Nonlinear Phenomena* **37** (1-3), 146–152.

Whitney, Hassler 1936 Differentiable Manifolds. *Annals of Mathematics* **37** (3), 645–680.

Zeng, Kevin & Graham, Michael D. 2021 Symmetry reduction for deep reinforcement learning active control of chaotic spatiotemporal dynamics. *Phys. Rev. E* **104**, 014210.