Low-dimensional data-driven models for forecasting and control of chaotic dynamical systems

by

Alec J. Linot

A dissertation submitted in partial fulfillment of

the requirements for the degree of

Doctor of Philosophy

(Chemical and Biological Engineering)

at the

UNIVERSITY OF WISCONSIN - MADISON

2023

Date of final oral examination: February 15, 2023

This dissertation is approved by the following members of the Final Oral Committee:

Michael D. Graham, Professor, Chemical and Biological Engineering

Reid C. Van Lehn, Associate Professor, Chemical and Biological Engineering

Victor M. Zavala, Professor, Chemical and Biological Engineering

Jean-Luc Thiffeault, Professor, Mathematics

Dedication

First, I would like to say I love you to my family and thank them for all they have done in helping me get to this point. My parents Joe and Becky have always been extremely supportive and loving. Their support played a major role in giving me the confidence needed to overcome the challenges that come with grad school. To my brother Reid, you've always reminded me to be a little less serious, and I am very excited about your upcoming wedding with Natalie. To my Grandpa Jim Rhatigan, I always appreciated the calls we had over the years about my progress and the guidance you gave me. Lastly, Suzie, I appreciate that you always listen to me going on about math and science, and I am especially grateful for your encouragement in the last few stressful months.

My friends have also been a huge support throughout this process. To Trina and Fernando, thank you for being there since AP Chem. I will never forget our travels through Brazil and hope we can go somewhere that none of us know the language – so Fernando can be as helpless as Trina and me. To my friends back in Kansas, especially Jordan, Alex, Bryce, and Jared, it was always a pleasure coming home from Wisconsin over the holidays to see you. To my friends from K-State, especially Quentin, Kail, Lincoln, and Brodie. Kail and Lincoln, I always look back fondly upon our time living together. Quentin and Lincoln, I was very happy you both moved to Minneapolis, and periodically seeing the two of you has been a wonderful excuse to drive over and forget about research for a weekend. And, to Brodie, I miss our times in the dorms, and always cherish hearing from you and appreciate your willingness to meet up when I give you an hour's notice that I will be passing by. I

would also like to thank Sigi Castro for being instrumental in my transition to grad school. He played a key role in inspiring me to pursue this degree in uncertain times.

While at Wisconsin I have also made some amazing friendships. To Mike Jindra, I appreciate all of the years we lived together. You taught me many things including if I want a sake bomb at a fancy sushi restaurant I should ask because that boldness may just pay off. To Jonathan, I will never know how you had two kids in grad school and still found time to play hockey with me every week, along with the many other activities we got up to with Mike. To Frank and Hector, I appreciate all you guys did in passing on your grad school experience to me. To my current labmates, Kevin, Eric, Carlos, RJ, Xiaopo, Alex, Dell, Charlie, Andrew, Ricardo, and Manish, I enjoy coming into the office thanks to all of you. Some fond memories I have include: going to the Bad Bunny concert with Carlos, my periodic dinners with Kevin and Eric, getting drinks with RJ and Jonathan at AIChE, and all my volleyball outings with Ricardo and Mike Jindra. To my labmates who have graduated, Ashwin, Xiao, Frank, and Daniel, thank you for all of your help over the years.

Finally, I would like to thank my defense committee of Prof. Van Lehn, Zavala, and Thiffeault. And, to my advisor Mike Graham, I have greatly appreciated your mentorship. You have been a huge influence on how I approach research, and all of the conferences, presentations, and scholarships you have helped me with over the years have been invaluable for my future career.

Abstract

Modeling high-dimensional and chaotic dynamics remains a challenging problem with a wide range of applications from controlling turbulent flows, to weather forecasting, to predicting cardiac arrhythmias – to name a few. Two major challenge in modeling these systems is that sometimes the equations are unknown and when they are known solving them can be prohibitively expensive. Due to these issues, only recently have experimental databases become mature enough and computational resources fast enough for there to exist large datasets of high-dimensional chaotic dynamical systems. The existence of these large datasets and advances in machine learning techniques opens the possibility for drastic improvements in the modeling and interpretability of chaotic dynamical systems through data-driven low-dimensional models. Here, we generate extremely low-dimensional "exact" models of chaotic dynamics in dissipative systems. In Ch. 1 we motivate finding low-dimensional models, provide background on manifolds, review existing methods on dimension reduction, and review methods on modeling dynamics. Following this, we describe the general framework for data-driven modeling that we take in the subsequent chapters. This model consists of an autoencoder – a neural network that reduces and expands dimension – to reduce the dimension of the problem and a neural ordinary differential equation (ODE) to evolve this low-dimensional system.

In Ch. 2 we investigate the application of this model on one of the simplest partial differential equations that can exhibit chaotic dynamics, the Kuramoto-Sivashinsky equation (KSE). We find that despite the high-dimensional nature of the state when simulating the

KSE, using an autoencoder we can drastically reduce the dimension of the problem without losing any of the underlying physics. Interestingly, the minimum number of dimensions we find to accurately recreate the dynamics, in this case, coincides with the true dimension of the manifold on which the data lies. This indicates that not only did we discover a parameterization of this low-dimensional manifold, but that the autoencoder approach we take may allow for an estimate of the manifold dimension in other systems. Furthermore, the minimal dimension needed to capture the physics exhibits a linear scaling as we increase the complexity of the KSE in agreement with the literature.

The KSE results are extended into Ch. 3, where we use a neural ordinary differential equation (ODE) for the dynamics. In a neural ODE, we learn a time evolution by replacing the right-hand side of an ODE with a neural network (NN). This representation allows us to evolve trajectories to arbitrary times and empirically we find this allows us to train our models with data that is spaced far apart in time. In addition, we find that neural ODE models at dimensions higher than the expected manifold dimension are unstable, further supporting the usefulness of representing the state in the low-dimensional manifold coordinates system from the autoencoder.

Then, in Ch. 4 we use this low-dimensional model (with slight modifications) to learn a control strategy for the KSE. Four "jets" control the system, and we use reinforcement learning (RL) to minimize the total power consumption. We first use RL to find a control strategy on the model and then deploy this strategy on the high-resolution simulation of the KSE. This overcomes the large computational burden of applying RL to the high-resolution simulation. When applying the control strategy (learned from the model) to the high-resolution simulation, we find the controller consistently pushes the system to an unstable low-"drag" fixed point, around which it oscillates. We then perform continuations of the controlled fixed point to find it connects to an unstable fixed point of the unactuated KSE. This reveals that RL can take advantage of underlying invariant solutions for control without any prior knowledge of their existence.

We continue to investigate the issue of stability in neural ODEs in Ch. 5, however here we do not perform dimension reduction before training the neural ODEs. In this chapter, we find long-time instability in neural ODEs of dissipative systems appears when they lack the dissipative structure of the underlying equations. We address this in Ch. 5 by training neural ODEs with a linear and a nonlinear term. This simple addition to the structure results in stable long-time dynamics. Furthermore, knowing the structure of the linear term alone can be useful. For example, we use the eigenvectors of the linear term for reduced-order modeling with classic Galerkin and nonlinear Galerkin methods. This "flips" the approach from before where instead of reducing the dimension and then finding dynamics, we find dynamics and then reduce the dimension.

In Ch. 6, we combine the stabilized neural ODEs with the dimension reduction techniques to model turbulent plane Couette flow. We find that we can reduce a fully resolved simulation requiring $\mathcal{O}(10^5)$ degrees of freedom down to fewer than 20 dimensions and still accurately reproduce short- and long-time statistics of this flow. Notably, the model exhibits the underlying "self-sustaining process" that drives wall-bounded turbulence. In addition, we use this low-dimensional model to discover 9 previously undiscovered invariant solutions of the model – a difficult and expensive task in the full simulation, but simple and cheap in our low-dimensional model.

As with the KSE, we then apply control to this low-dimensional model of Couette flow in Ch. 7 with slight modifications. In this case, we apply control in the form of two zero-flux slot jets on one wall. We first train a model from data with randomly chosen actuations, then we train an RL agent to control this model. Upon applying this control strategy to the full system we find that over 80% of the trajectories laminarize within 900 time units. This outperforms the standard opposition control approach, which has much more control authority as it involves controlling the wall-normal velocity at all locations of the wall – not just two slot jets. Furthermore, we find that we can achieve these results from just wall shear observations, making this control policy much more viable to realize in a real-world

experiment.

Finally, we conclude with a discussion of some exploratory works and future directions in Ch. 8. The first of these explorations is that of distributed modeling. Specifically, we show that we can build a set of low-dimensional models that we couple together for the prediction of the KSE in large domains. This opens the possibility for the extension of this work to many practical systems. Next, we explore incorporating both discrete and continuous symmetries into the modeling approach. By accounting for both of these types of symmetries we significantly improve the autoencoder performance and guarantee equivariance of trajectories under symmetry operations. We end this chapter by discussing how to incorporate the equations of motion into the model and various extensions of the Couette control system.

The results highlighted in this work show that the apparent high dimensionality of many chaotic dynamical systems is a facade. The chaotic dynamics exhibited by dissipative PDEs tend towards a region of state space that lies on a finite-dimensional invariant manifold. By parameterizing this manifold we can represent the high-dimensional state information in a low-dimensional manifold coordinate system. Once data lies on the manifold, all forecasting can be done in this manifold coordinate system, drastically improving computational efficiency. This drop in computational cost then allowed us to perform the extremely computationally expensive tasks of finding invariant solutions and RL control strategies.

Contents

1 General introduction **1**

1.1 Dynamical systems on manifolds 3

1.2 Dimension reduction 6

1.2.1 Manifolds 6

1.2.2 Approximating manifold coordinates 8

1.2.3 Linear dimension reduction 9

1.2.4 Nonlinear dimension reduction 11

1.3 Time evolution 17

1.3.1 Approximating discrete-time maps 18

1.3.2 Approximating ODEs 20

1.4 Framework 22

1.4.1 Modeling 22

1.4.2 Model-based control 24

1.5 Outline of this work 25

2 Parameterizing an Inertial Manifold **27**

2.1 Introduction 27

2.2 Formulation 30

2.3 Methodology and results 31

2.4 Summary 43

3 Dynamics with neural ODEs **44**

3.1 Introduction . . . 45

3.1.1 Dimension reduction methods . . . 46

3.1.2 Time evolution methods . . . 46

3.2 Framework . . . 49

3.3 Results . . . 52

3.3.1 Dimension reduction with autoencoders . . . 53

3.3.2 Effect of dimension reduction on time evolution . . . 56

3.3.3 Effect of temporal data spacing on time evolution prediction . . . 60

3.3.4 Effect of model dimension on time evolution predictions . . . 65

3.4 Summary . . . 68

4 Reinforcement learning with model-based control **70**

4.1 Introduction . . . 71

4.2 Formulation . . . 78

4.2.1 Kuramoto-Sivashinsky equation . . . 78

4.2.2 Background and method formulation . . . 79

4.3 Results . . . 85

4.3.1 Neural ODE model evaluation . . . 85

4.3.2 Model-based control performance . . . 89

4.4 Summary . . . 94

5 Stabilized neural ODEs **99**

5.1 Introduction . . . 100

5.2 Stabilized Neural ODE Framework . . . 102

5.3 Results . . . 107

5.3.1 Viscous Burgers equation . . . 108

5.3.2 Kuramoto-Sivashinsky equation . . . 113

5.4 Summary 122

6 Low-dimensional model of Couette flow 124

6.1 Introduction 125

6.2 Framework 129

6.3 Results 134

6.3.1 Description of plane Couette flow data 134

6.3.2 Dimension reduction and dynamic model construction 137

6.3.3 Short-time tracking 142

6.3.4 Long-time statistics 146

6.3.5 Finding ECS in the model 151

6.4 Summary 155

7 Model-based control of Couette flow 157

7.1 Introduction 158

7.2 Framework 161

7.2.1 Navier-Stokes equation with slot jets 161

7.2.2 Data-driven framework 164

7.2.3 DManD modeling framework 165

7.2.4 Reinforcement learning using DManD 170

7.3 Results 172

7.3.1 Description of data 172

7.3.2 Manifold coordinate system 174

7.3.3 DManD performance 174

7.3.4 DManD-RL performance 179

7.4 Summary 186

8 Conclusions 188

8.1 General summary 188

8.2 Future directions for modeling 189
8.2.1 Distributed models 189
8.2.2 Symmetry charting 193
8.2.3 Incorporating the underlying equations 199
8.3 Future directions for control 200
8.3.1 Model predictive control 200
8.3.2 DManD-RL Re = 500 202
8.3.3 Application to different flow systems 204

References **205**

List of Figures

1.1 Black lines indicate arbitrary initial conditions in $\mathbb{R}^d$ attract towards $\mathcal{M}$. . . 5

1.2 (a) One-dimensional manifold (described in the text) embedded in two dimensions where PCA to one dimension is a diffeomorphism. (b) One-dimensional Archimedean spiral embedded in two dimensions where PCA to one dimension is not a diffeomorphism. The blue lines indicate the mapping from PCA to the manifold. 10

1.3 (a) Learning the three-dimensional manifold coordinates of the four-dimensional "spiral" Lorenz (color is the fourth dimension). (b) Learning the vector field in the manifold coordinates. (c) Transforming new initial conditions (black dots) into the manifold coordinates, evolving them with the vector field g (black curves), and transforming back to the ambient space. 23

1.4 Left is the standard RL procedure of a state being fed to an RL agent that acts on an environment, resulting in a state and a reward. Right is the same procedure on the less expensive DManD model. u is the full state, h is the state in the manifold coordinates, a is the action, and r is the reward. 25

2.1 Block diagram for the hybrid autoencoder and time-evolution scheme. NNs are pink (light gray). 31

2.2 MSE of test data for various d_h for $L = 22$. The legend is described in the text. 34

2.3 $\mathcal{P}$ vs. ε state-space projection for the data at $L = 22$ and HNN ROM prediction with $d_h = 7$. 36

2.4 Time-correlation function for the data at $L = 22$, and HNN ROM prediction with $d_h = 7$. 37

2.5 (a) and (c) trajectories (color contours of $u(x,t)$ with solid lines at $u = 1$ and dashed lines at $u = -1$) evolving from different initial conditions according to the HNN ROM; $L = 22$. (c) and (d) true trajectories corresponding to (b) and (d). 38

2.6 Spatial correlation function for the data and HNN ROM prediction with $d_h = 7$, $L = 22$. 39

2.7 (a) joint PDF of data, $L = 22$, and (b) joint PDF of HNN ROM prediction, both plotted on a logarithmic scale. (c) Relative error in PDF vs dimension. The PDF from data is denoted f, and that from the model prediction is $\tilde{f}$. . 41

2.8 (a) MSE vs the difference between d_h and the predicted $d_{\mathcal{M}}$. (b) Dimension $d_{\mathcal{M}} + 1$ of the unreduced IM for different domain sizes: "HNN" denotes the current results, "PM" the number of physical modes from [237], and "PM Fit" a linear fit of the PM curve. 42

2.9 (a) and (d) joint PDF of data for $L = 44, 66$. (b), (c), (e), and (f) joint PDF of HNN ROM prediction at $d_{\mathcal{M}} - 1$ and $d_{\mathcal{M}}$ for $L = 44, 66$. 42

3.1 MSE of test data as a function of dimension for domain sizes $L = 22$, 44, and 66. The expected manifold dimensions are $d_{\mathcal{M}} = 8$, 18, and 28 for these domain sizes, respectively. 54

3.2 Examples of trajectory predictions for different models, $L = 22, \tau = 0.25$. The left shows predictions up to ~ 2.3 Lyapunov times, and the right shows predictions after ~ 23 Lyapunov times. (a) true trajectory, (b) predicted trajectory when approximating g with $d_{\mathcal{M}} = 8$. (c) predicted trajectory when approximating f. (d) predicted trajectory when approximating $\mathcal{F}(f)$. . . . 58

3.3 Magnitude of Fourier modes over time when evolving an initial condition with a neural ODE approximation of (a) the full-space dynamics in real space (f) and of (b) the full-space dynamics in Fourier space ($\hat{f}$), $L = 22, \tau = 0.25$. The inset in (b) is the evolution of the real part of $\hat{f}_{30}$. 59

3.4 Example of trajectory and predictions for data spaced $\tau = 10$ apart, $L = 22$: (a) true trajectory; (b) predicted trajectory; (c) absolute error between the two trajectories; (d) norm of the error. The norm of the error is shown for the raw data, and for the spatial shift that minimizes this error at each time. 60

3.5 Normalized difference between trajectory and prediction as a function of time for different training data spacing, $L = 22, d_h = 8$. 61

3.6 Temporal autocorrelation for models learned with different data spacing, $L = 22, d_h = 8$. 62

3.7 (a) joint PDF at $L = 22, d_h = 8$. (b) and (c) joint PDFs when approximating the ODE and the discrete-time map, respectively, at $\tau = 10$. (d)-(f) joint PDFs when approximating the ODE for $\tau = 15$, 16, and 20. 63

3.8 KL divergence of the models trained with different data spacing. Here time is scaled with Lyapunov time for the corresponding domain size. For $L = 22, 44, 66$ we use $d_h = 8, 18, 28$. 64

3.9 (a) KL divergence of the best autoencoder and neural ODE pair at multiple dimensions for $L = 22$. The nonlinear encoder corresponds to an autoencoder with a NN for encoding, and the linear encoder corresponds to an autoencoder with the projection onto the first d_h Fourier modes for encoding. Both cases use a NN for decoding. (b) mean squared error of reconstruction for the nonlinear autoencoder at each dimension. 66

3.10 Short-time error at various dimensions. Domain sizes are $L = 22, 44, 66$ for (a)-(c). 67

3.11 KL divergence for all trained models at different dimensions. Domain sizes are $L = 22, 44, 66$ for (a)-(c). The dotted lines on each plot correspond to the D_{KL} computed by comparing two joint PDFs of the true system. 68

3.12 (a)-(c) True joint PDFs for $L = 22, 44, 66$. (d)-(f) joint PDFs from the best models (lowest D_{KL}) for $L = 22, 44, 66$ at $d_h = 8, 18, 28$. 69

4.1 Typical reinforcement learning cycle. 84

5.1 Snapshots of VBE trajectories of true system and model predictions for different initial conditions. The rows indicate different initial conditions and the columns indicate solution fields at $t = 1$, 3, and 5, respectively. 107

5.2 Ensemble averaged error of predictions of all models with noisy initial conditions of $\epsilon = 0.0$, 0.1, and 0.5 for (a)-(c). 109

5.3 Ensemble-averaged energy spectrum of the true system and the model predictions at different times and noise on the initial conditions. (a)-(c),(d)-(f), and (g)-(i) are with noise $\epsilon = 0.0$, 0.1, and 0.5, at times $t = 1$, 3, and 5, respectively.112

5.4 Comparison of training initial conditions to noisy initial conditions. (a) compares example initial conditions and (b) compares the ensemble-averaged energy spectrum. 112

5.5 Predictions of trajectories using different models. (a) is the true trajectory. (b) is the predicted trajectory with the nonlinear model. (c) is the predicted trajectory with the fixed linear model. (d) is the predicted trajectory with the CNN model. 115

5.6 (a) is the true joint PDF at $L = 22$. (b)-(d) are the joint PDFs for the nonlinear, fixed linear, and CNN models, respectively. 117

5.7 (a) and (c) are trajectories of the CNN model for initial conditions with $\epsilon = 0.5$ and 10. (b) and (d) are trajectories of the standard neural ODE model for initial conditions with $\epsilon = 0.5$ and 10. 117

5.8 (a) shows three example snapshots of the training data with different levels of noise (ϵ). (b)-(d) are joint PDFs of the CNN model for $\epsilon = 0.1, 0.3$ and 0.5, respectively. 119

5.9 (a) is the variance of the true ODE projected onto each eigenvector of the true linear term sorted from largest to smallest eigenvalue. (b) and (c) are the projection operators from the eigenvectors of the true linear term sorted by the eigenvalues and the variance in (a), respectively. (d) is the variance of the CNN ODE projected onto each eigenvector of the CNN linear term sorted from largest to smallest eigenvalue. (e) and (f) are the projection operators from the eigenvectors of the CNN linear term sorted by the eigenvalues and the variance in (c), respectively. The projection operators are for 10 eigenvectors. 120

5.10 KL divergence of the joint PDFs generated with either standard Galerkin "G" or nonlinear Galerkin "NG" when using the CNN neural ODE "CNN" or the true equations "True". The inset is the CNN Galerkin ROM at a dimension $d_p = 18$. 123

6.1 (a) Eigenvalues of POD modes sorted in descending order. (b) Components of the Reynolds stress for data generated by the DNS and this data projected onto 256 POD modes. In (a) the curves are, from top to bottom, $\langle u_x'^2 \rangle$, $\langle u_z'^2 \rangle$, $\langle u_y'^2 \rangle$, and $\langle u_x' u_y' \rangle$. 139

6.2 Mean squared error on test data for POD, standard autoencoders, and hybrid autoencoders at various dimensions d_h. At each dimension there are four standard and four hybrid autoencoders. 141

6.3 Snapshots of the streamwise velocity at $y = 0$ from the DNS and from the DManD model at $d_h = 18$. 143

6.4 Modal RMS velocity from the DNS (M) and from the DManD model at $d_h = 18$ ($\tilde{M}$). The markers correspond to the times in Fig. 6.3. 143

6.5 Normalized tracking error for 10 random initial conditions (different shades) using DManD with $d_h = 18$. 144

6.6 (a) ensemble averaged short-time tracking and (b) temporal autocorrelation of the kinetic energy for DmanD models of increasing dimension. In (b) odd numbers above $d_h = 5$ are omitted for clarity. 145

6.7 (a) an example of the phase evolution and (b) the MSD of the phase evolution for the DNS and the DManD model at $d_h = 18$. 146

6.8 Fraction of unstable DManD models with standard neural ODEs and with stabilized neural ODEs at various dimensions. 147

6.9 Comparison of $\langle ||a_n||^2 \rangle$ (mean-squared POD coefficient amplitudes) from the DNS to (a) $\langle ||a_n||^2 \rangle$ from the DManD model at $d_h = 18$ and (b) $\langle ||a_n||^2 \rangle$ from POD-Galerkin with N POD modes (reproduced with permission from Gibson [66]). In (b), the quantity λ is equivalent to $\langle ||a_n||^2 \rangle$ from the DNS. 148

6.10 Components of the Reynolds stress with increasing dimension for DManD models at various dimensions. Odd numbers above $d_h = 5$ are omitted for clarity. 149

6.11 (a)-(c): examples of joint PDFs for the true system, the DManD model at $d_h = 6$, and the DManD model at $d_h = 18$. (d): earth movers distance between the PDF from the DNS and the PDFs predicted by the DManD model at various dimensions. "DNS" is the error between two PDFs generated from DNS trajectories of the same length with different initial conditions. (e) and (f): the error associated with the DManD model PDFs at $d_h = 6$ and $d_h = 18$. 150

6.12 Power input vs. dissipation of known periodic orbits (period reported in the bottom right) from the DNS and periodic orbits found in the DManD model at $d_h = 18$. The blue line is a long trajectory of the DNS for comparison. . . 154

6.13 (a) Periodic orbits found in DManD models at $d_h = 18$ that converged to the (b) periodic orbits found in the DNS. Each of the colors corresponds to one of the new solutions in Table 6.2. The blue curve at the back is a long trajectory of the DNS for comparison. 155

7.1 Schematic of the Couette flow domain with two slot jets on one wall. 161

7.2 Schematic of the DManD-RL framework. After step 3, $\tilde{\cdot}$ is omitted for clarity. 166

7.3 (a) Eigenvalues from the POD. (b) and (c) Snapshots of the centerline streamwise velocity ($u_x(y = 0)$) from the DNS at representative low-drag (left) and high-drag (right) instants. (d) and (e) POD reconstruction with 500 modes of the DNS results in (b) and (c). (f) and (g) Autoencoder reconstruction of (d) and (e). Solid contour lines are positive and dotted contour lines are negative. 173

7.4 Comparison of snapshots from a randomly actuated trajectory from the DNS (top) with the DManD reconstruction (bottom). Times are given at the top right of each image. 176

7.5 (a) Ensemble averaged tracking error and (b) temporal autocorrelation of kinetic energy for the DNS, the DManD model, and the autoencoder. "Auto" represents putting the DNS trajectories through the autoencoder without any time prediction: i.e. this represents the error of just reducing, then expanding the dimension. 177

7.6 Four components of the Reynolds stress for the DNS, the DManD model, and the autoencoder. 178

7.7 (a) Joint PDF of the true (D) and predicted drag ($\tilde{D}$) from the reward network, Eq. 7.20. (b) Joint PDF of the true (h) and predicted ($\tilde{h}$) low-dimensional state from the observation network, Eq. 7.30. Note the logarithmic scales. The cyan line indicates perfect reconstruction, and the MSE is for D and h with the mean subtracted and divided by the standard deviation. . 179

7.8 Trajectories beginning from test initial conditions with (a) no control, (b) random actuations, (c) DManD-RL control, (d) DNS-based RL control, and (e) DManD-RL control using wall observations. Each figure shows 25 test trajectories. 181

7.9 (a) Time series of drag (red) and actuation signal of the left jet (black) for an example DNS trajectory controlled by the DManD-RL agent. (b)-(e) snapshots of the trajectory at times marked in (a). These snapshots include an isosurface of $u_x = -0.35$ and the jet actutations (red is fluid injection, blue is fluid suction). 183

7.10 Joint PDF of max jet velocity and average wall-normal velocity at (a) $y^+ \approx 10$ and (b) $y^+ = 33$ (the cyan dotted line is $v_J = -\langle u_y \rangle_J$). 185

8.1 A snapshot of the KSE for domain size $L = 220$. The red dotted lines separate the domain for each of the 10 chunks. 190

8.2 (a) and (b) are trajectories of the KSE at $L = 220$ for the true data and the distributed DManD model. (c) is the error between these trajectories. 193

8.3 Joint PDFs of u_x vs u_{xx} for (a) the true data and (b) the distributed DManD model. 194

8.4 (a) shows a trajectory of Kolmogorov flow for $5,000$ time units projected onto $\hat{\omega}_R(0,1)$, $\hat{\omega}_I(0,1)$, $\hat{\omega}_I(0,2)$. The color corresponds to the indicator that identifies the symmetry group. (b) shows this data mapped to the fundamental space. 197

8.5 Autoencoder MSE vs training data size without accounting for symmetry (Full), account for discrete symmetry (Discrete), accounting for continuous (Continuous), and accounting for all symmetries (Fundamental). 198

8.6 Trajectories of enstrophy for the true system and the symmetry aware DManD model starting with different discrete symmetries in (a) and (b). The color indicates the current discrete symmetry indicator. 199

8.7 Trajectories of drag for 25 initial conditions without control and with control applied to a DNS at Re = 400. The control policy comes from applying MPC to the DManD model. 201

8.8 Four components of the Reynolds stress for the DNS, the DManD model, and the autoencoder at Re = 500. 203

8.9 Trajectories of drag for 25 initial conditions without control and with control from the DManD-RL agent on a DNS at Re = 500. 204

Chapter 1

General introduction

In this thesis, we are predominantly concerned with the study of the chaotic dynamics of turbulent fluid flows. The study of turbulent flows is of huge practical importance due to the massive amounts of energy lost due to turbulent drag in industrial and commercial processes like air flowing over a plane or oil pumped through a pipe. Turbulent drag accounts for 25% of the energy used in industry and commerce, which results in 5% of all man-made CO_2 emissions [104]. Small reductions in turbulence could lead to massive savings in energy. However, a major challenge is directly solving the Navier-Stokes equations is extremely computationally expensive [165] and with the ending of Moore's law [187] modeling more turbulent flows may remain a challenge with current approaches. This motivates the goal of this thesis, which is the development of low-dimensional data-driven models of chaotic dynamical systems with a focus on turbulent flows.

Chaotic dynamical systems are ubiquitous in physics and nature from systems as simple as two pendulums attached together to those as complex as global weather patterns. These types of systems are extremely hard to predict and, despite the fact their dynamics are governed by deterministic systems, the dynamics often appear random. This characteristic of deterministic chaotic dynamical systems comes from two traits inherent to these types of systems: high sensitivity to initial conditions and aperiodic long-time dynamics [201]. Sensitivity to initial conditions makes predicting and controlling these systems difficult because

small changes cause solutions to diverge exponentially fast. Additionally, the aperiodic long-time behavior makes the dynamics of the system appear "random" hindering our ability to find the physical mechanisms that describe the underlying behavior.

We predominantly study two chaotic dynamical systems in this thesis. First, we will develop our modeling techniques on a one-dimensional proxy for turbulence called the Kuramoto-Sivashinsky equation (KSE), which was originally developed for flame propagation [191]. This partial differential equation (PDE) is a useful testbed as it is the "simplest" PDE that can exhibit chaotic dynamics [14]. The second system we consider is three-dimensional direct numerical simulations of the Navier-Stokes equations. In particular, we will study turbulent plane Couette flow in the smallest domain that sustains turbulence [83]. In what follows we will develop methods for finding low-dimensional models of these systems and then applying control to them.

For finding the low-dimensional models for these systems we attempt to connect approaches from chaotic dynamical systems, topology, and machine learning. In particular, these low-dimensional models will rely on the fact that the long-time dynamics of the systems we are interested in lie on strange attractors [49]. Strange attractors are characterized by fractal (noninteger) dimensions. Despite this complexity, these attractors still lie on manifolds, which are of an integer dimension, so we can consider describing the dynamics of the attractor in terms of the topology of these manifolds. We approach the parameterization of these manifolds and the description of dynamics on them using modern data-driven tools developed in the machine learning community.

The objective of the following sections is to provide a detailed background on what is needed for "exact" low-dimensional modeling on manifolds, and a review of the many possible approaches we considered taking in this pursuit. As such, the following sections contain details on dynamical systems on manifolds, data-driven dimension reduction techniques, and data-driven time-evolution techniques. To orient the reader, we provide the main conclusions from each section. In Sec. 1.1 we discuss the equations that describe dynamical systems that

collapse onto finite invariant manifolds. This leads to the conclusion that if we can find three functions – an encoding function, a decoding function, and a time-evolution function – then we can exactly model a high-dimensional dynamical system in a low-dimensional manifold coordinate system. As these functions are typically unknown, we want to approximate them from data.

After this, in Sec. 1.2 we first determine that generally, we want the encoder and the decoder to be *nonlinear* functions. Then we use this result to motivate an extensive review of data-driven techniques for dimension reduction. We determine that autoencoders constitute a good choice for finding an encoding and a decoding function. This leads to a discussion on how to approximate the time-evolution function in Sec. 1.3. We first review methods involving both the true equations of motion, and data-driven techniques, and chose to use neural ordinary differential equations (ODE). Finally, we present how these methods combine to form a general framework for low-dimensional modeling in Sec. 1.4, and we outline the chapters in Sec.1.5

1.1 Dynamical systems on manifolds

Throughout this thesis, the objective is to approximate (from data) the dynamics of some partial differential equation

$$\frac{\partial v}{\partial t} = \mathcal{F}(v), \tag{1.1}$$

that is a function of space and time $v(x,t)$. In order to approximate these dynamics we first need to convert this formally "infinite"-dimensional problem [178] into a finite-dimensional problem. We do this by approximating v with a sum of time-dependent modal coefficients $(u_i(t))$

$$v(x,t) = \sum_{i=1}^{d} u_i(t)\varphi_i(x), \tag{1.2}$$

with the orthonormal basis $\varphi_i(x)$ [92]. Then, by substituting Eq. 1.2 into Eq. 1.1 and performing a Galerkin projection onto $\varphi_j(x)$ we transform Eq. 1.1 into

$$\frac{du_j}{dt} = \left(\mathcal{F}\left(\sum_{i=1}^{d} u_i(t)\varphi_i(x) \right), \varphi_j(x) \right), \tag{1.3}$$

which we write generally as

$$\frac{du}{dt} = f(u). \tag{1.4}$$

Now, this ordinary differential equation is the full-order model (FOM) [121], which we often refer to as a high-resolution simulation. Instead of considering recreating the dynamics of Eq. 1.1 we can instead consider recreating the dynamics of Eq. 1.4, with full state $u \in \mathbb{R}^d$ in the ambient space $\mathbb{R}^d$.

In general, evolving the dynamics of the FOM requires d degrees of freedom, however, we either know, or evidence suggests, that the dynamics of many dissipative dynamical systems collapse onto low-dimensional manifolds $\mathcal{M} \subset \mathbb{R}^d$ at long times [44, 59, 60, 205, 241]. Figure 1.1 shows a caricature of this behavior. If we start anywhere in the ambient space $\mathbb{R}^d$, at long times we expect trajectories of these systems to lie on $\mathcal{M}$. Dissipative systems, in this thesis, refer to ones where energy dissipates at high Fourier modes, these energy losses are then offset by energy injection at low Fourier modes, and energy flows from the lower to higher modes [241]. Once the dynamics of these systems collapse onto a manifold, we only need $\mathbb{R}^{d_\mathcal{M}}$ degrees of freedom to describe the dynamics, where $d_\mathcal{M}$ is the manifold dimension (the details of which we discuss in Sec. 1.2).

On the manifold $\mathcal{M}$ we can define a mapping to a manifold coordinate system (we also refer to this as encoding)

$$h = \chi(u), \tag{1.5}$$

where $h \in \mathbb{R}^{d_\mathcal{M}}$, and a mapping back to the full state (we also refer to this as decoding)

$$u = \chi^{-1}(h). \tag{1.6}$$

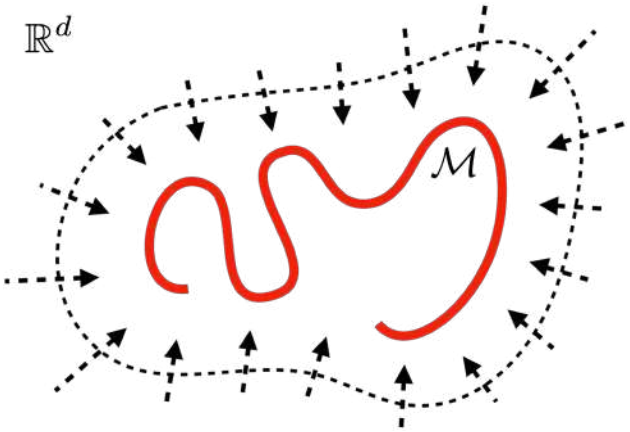


Figure 1.1: Black lines indicate arbitrary initial conditions in $\mathbb{R}^d$ attract towards $\mathcal{M}$.

Then, by simply taking the time derivative of Eq. 1.5, performing chain rule, and substituting Eq. 1.6 for u, we can write the ODE in this manifold coordinate system:

$$\frac{dh}{dt} = \frac{\partial \chi(u)}{\partial u}\bigg|_{\chi^{-1}(h)} f(\chi^{-1}(h)). \tag{1.7}$$

Alternatively, you can perform the same process on Eq. 1.6, which is the *manifold Galerkin* approach discussed in Lee and Carlberg [121], resulting in a similar ODE. In Eq. 1.7 we see that the ODE in the manifold coordinates only depends upon the manifold coordinate representation of the state, so we can also write the ODE generally as

$$\frac{dh}{dt} = g(h). \tag{1.8}$$

This shows that two equivalent methods exist for evolving trajectories of the FOM (Eq. 1.4) after dynamics have collapsed onto a manifold: 1) Take an initial condition u_0 and integrate Eq. 1.4 forward in time, or 2) Take an initial condition u_0, map u_0 to the manifold coordinate system (Eq. 1.5) to get h_0, then integrate Eq. 1.8 forward in time, and, finally, map the trajectory back to the full state with Eq. 1.6. As we show above, both cases result in the same solution, however for many cases $d_{\mathcal{M}} \ll d$. This low dimensionality makes the second approach far faster to solve numerically. For example, in Ch. 6 and Ch. 7 by taking the second approach we will turn a problem that takes days into a problem that takes

minutes. Furthermore, the description of the dynamics in the manifold coordinate system may be more interpretable than the description in the full state.

Unfortunately, in most cases χ and χ^{-1} are unknown and even if f is known (it is not in all cases), directly evaluating $f(\chi^{-1}(h))$ in Eq. 1.7 may be computationally expensive. These challenges motivate the key aim of this work: can we generate "exact" reduced-order models of dissipative dynamical systems by approximate χ, χ^{-1}, and g directly from data? To answer that question we review possible methods for approximating χ and χ^{-1} in Sec. 1.2, followed by possible methods of approximating g in Sec. 1.3.

In what follows, we discuss many methods for approximating functions using data, so first let us describe the form in which this data exists. Typically, we will assume we have data of the full state in form of M snapshots $\{u_1, u_2, ..., u_M\}$ from a trajectory (or trajectories) that has collapsed onto the manifold (i.e. $u_i \in \mathcal{M} \subset \mathbb{R}^d$). In the case of dimension reduction, the organization of these snapshots does not matter, but in the case of learning the time evolution typically this data is a time sequence. Often full-state data is unavailable making this assumption seem restrictive, however, most of what we discuss works for any observations $o \in \mathbb{R}^{d_o}$ diffeomorphic to the full state (e.g. time delays [203, 239]). This is because a diffeomorphism implies that a function $u = O(o)$ exists (and is continuous and smooth). Then we can simply insert $u = O(o)$ into Eq. 1.5 and Eq. 1.6, which allows us to approximate the composition of those functions.

1.2 Dimension reduction

1.2.1 Manifolds

We are predominantly interested in parameterizing manifolds with as few degrees of freedom as possible, which is effectively a problem of dimension reduction. In order to understand what is needed to parameterize a manifold, we first define a topological $d_{\mathcal{M}}$-manifold $\mathcal{M}$. For completeness, we provide the full set of properties, but in our case, the third property

is the one of particular importance. We refer the reader to Lee [120] for a more complete description. A topological $d_{\mathcal{M}}$-manifold $\mathcal{M}$ posseses the following properties [120]:

1. $\mathcal{M}$ is a **Hausdorff space**: for every pair of distinct points $p, q \in \mathcal{M}$ there are disjoint open subsets $U, V \subseteq \mathcal{M}$ such that $p \in U$ and $q \in V$.

2. $\mathcal{M}$ is **second countable**: there exists a countable basis for the topology of $\mathcal{M}$.

3. $\mathcal{M}$ is **locally Euclidean of dimension** $d_{\mathcal{M}}$: each point of $\mathcal{M}$ has a neighborhood that is homeomorphic to an open subset of $\mathbb{R}^{d_{\mathcal{M}}}$. This means for each $p \in \mathcal{M}$ we can find

   (a) an open subset $U \subseteq \mathcal{M}$, containing p,

   (b) an open subset $\check{U} \subseteq \mathbb{R}^{d_{\mathcal{M}}}$, and

   (c) a homeomorphism $\varphi : U \to \check{U}$.

Of these three properties defining a manifold, the third property is notable because it says that a homeomorphism between $\mathcal{M}$ and $\mathbb{R}^{d_{\mathcal{M}}}$ only has to exist on a local open subset. This means for a $d_{\mathcal{M}}$-manifold a single global map between $\mathcal{M}$ and $\mathbb{R}^{d_{\mathcal{M}}}$ will not necessarily exist, although it can. When no global map exists, the χ we defined in Eq. 1.5 is an "atlas" that consists of a set of "charts" on the manifold, where these charts map local portions of the manifold $\mathcal{M}$ to $\mathbb{R}^{d_{\mathcal{M}}}$. For more details on how to perform this charting procedure, we refer the reader to Floryan and Graham [57].

To avoid this added complexity, in what follows we describe χ as a single global function. As it turns out, using a global map for χ accurately predicts the correct manifold dimension for a number of systems in this thesis [128, 129]. Additionally, even in cases where charting is required a global map will exist, it just may require up to $2d_{\mathcal{M}}$ degrees of freedom [233]. Lastly, we note that this global description of χ can simply be applied to each of the charts when applying the charting procedure [57].

1.2.2 Approximating manifold coordinates

So far our description of a manifold assumes some function χ exists but provides no means of finding this function. Fortunately, a vast literature exists on techniques for data-driven dimension reduction. In our case, we want a dimension-reduction technique that can approximate a diffeomorphism $\chi : \mathcal{M} \to \mathbb{R}^{d_\mathcal{M}}$, the inverse of this function χ^{-1}, and the minimal dimension $d_\mathcal{M}$. These properties are all vital to our modeling approach.

We want a diffeomorphism χ because a diffeomorphism is a bijection that is smooth and has a smooth inverse [120], where, in this context, smooth means the function and all of its derivatives are continuous. It is important for χ to be a diffeomorphism because this implies that χ^{-1} exists and both functions are continuous and smooth. The data-driven techniques we consider will tend to be continuous smooth functions, so trying to approximate discontinuous functions would be challenging. Smoothness is also needed for the ODE in Eq. 1.7.

In addition to χ, we want the dimension-reduction technique to approximate χ^{-1}, which allows us to map back to the ambient space. We will denote data-driven approximations of χ^{-1} as $\check{\chi}$. We only denote the approximation for χ^{-1} and not χ because χ is nonunique, but upon selecting χ there is a unique χ^{-1}, which we try to approximate. The last property we want from a dimension-reduction technique is for it to provide an estimate of the dimension of the manifold $d_\mathcal{M}$ because this is generally unknown.

Before discussing specific dimension reduction techniques it is first worth exploring when we might expect these techniques to provide a diffeomorphism. Whitney [232] showed that a generic smooth map from $\mathcal{M}$ to $\mathbb{R}^{2d_\mathcal{M}+1}$ is a diffeomorphism. Then, Sauer et al. [185] later refined this result, showing *almost every* smooth map from a $d_\mathcal{M}$-manifold to $\mathbb{R}^n$ for $n > 2d_b$ is a diffeomorphism, where d_b is the box-counting dimension. "Almost every" means that the set of functions that are not diffeomorphisms have measure zero [185]. The box-counting dimension is one of a family of fractal dimensions that are a defining characteristic of chaotic attractors [88], and it must be less than the manifold dimension. These results are

surprising as it suggests that as long as the dimension technique we consider maps to $\mathbb{R}^{2d_{\mathcal{M}}}$ then it is a diffeomorphism. Notably, these results form the backbone of why time-delays representations of the state are possible [203].

However, just because almost every map is a diffeomorphism with sufficiently many dimensions does not mean all encodings χ are equivalent. Even with the above result, the proper choice of χ allows the manifold to be parameterized with $d_{\mathcal{M}}$ degrees of freedom, while an improper choice may require $2d_{\mathcal{M}}$ degrees of freedom. Additionally, we want to approximate χ^{-1}, which may be more or less difficult depending upon the selection of χ. And, finally, we want to model the dynamics in the manifold coordinate system h, so a poor selection of χ may make this prediction more difficult.

1.2.3 Linear dimension reduction

A natural first approach to dimension reduction is with linear techniques. These techniques all involve projecting the manifold $\mathcal{M}$ in the ambient space $\mathbb{R}^d$ onto some linear subspace $\mathbb{R}^{d_h}$. Thus, all of these techniques simplify to determining the best basis for spanning this subspace. An overview of linear dimension reduction techniques from a machine learning perspective is provided in Cunningham and Ghahramani [38]. Many of these techniques result in the widely used principal component analysis (PCA), also known as the proper orthogonal decomposition (POD) or the Karhunen-Loève decomposition [92].

In PCA, we find an orthonormal basis onto which we project the data. The first vector in this basis maximizes the variance in the projection of the data along this direction, then the second vector maximizes the variance orthogonal to the first, and the subsequent vectors continue this process [38]. By selecting a basis in this manner, the projection onto any d_h modes results in the lowest reconstruction error of any linear projection. The PCA basis comes from computing the leading left singular vectors $U \in \mathbb{R}^{d \times d_h}$ with the singular value decomposition (SVD) of the centered snapshot matrix $X = [u_0 - \langle u \rangle, ..., u_M - \langle u \rangle]$ (here $\langle . \rangle$ denotes ensemble average). A similar method to the SVD is to compute the eigenvectors

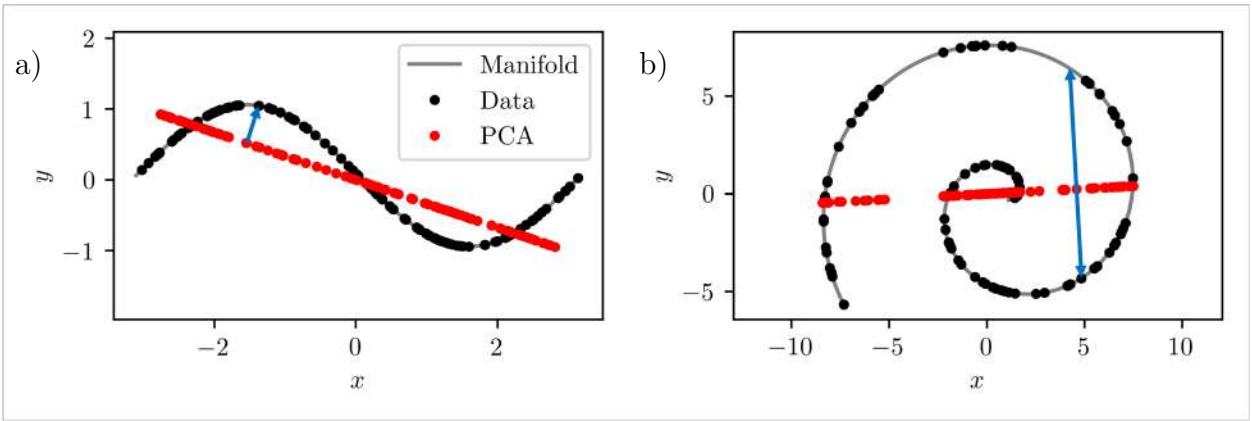


Figure 1.2: (a) One-dimensional manifold (described in the text) embedded in two dimensions where PCA to one dimension is a diffeomorphism. (b) One-dimensional Archimedean spiral embedded in two dimensions where PCA to one dimension is not a diffeomorphism. The blue lines indicate the mapping from PCA to the manifold.

of the covariance matrix $X^T X$. With these left singular vectors, we define the encoding as $h = \chi(u) = U^T u$ and the decoding as $\tilde{u} = \check{\chi}(h) = Uh$. Here we write $\check{\chi}$ instead of χ^{-1} to emphasize this is an approximation. An important subtlety worth noting here is that although Whitney's embedding theorem indicates that a linear mapping χ is guaranteed to be a diffeomorphism when $d_h = 2d_{\mathcal{M}}$ the linear mapping back $\check{\chi}(h) = Uh$ is not necessarily χ^{-1}.

To illustrate this point, in Fig. 1.2a we show data that has been sampled randomly from $x \in [0, 2\pi]$ with $y = \sin(x)$, centered, and reconstructed with one PCA mode. We compute the PCA projection of this data $u = [x, y]^T$ onto one PCA mode with $\tilde{u} = U_1 U_1^T u$. Despite the underlying manifold being one-dimensional (i.e. only a function of x), PCA alone does not accurately reconstruct the data on this manifold with one dimension. However, as illustrated by the blue line, there is a one-to-one correspondence between points on the manifold and those projected onto the PCA mode. This indicates (omitting some details) that the encoding $h = U_1^T u$ is a diffeomorphism and some χ^{-1} exists to map between the two representations. However, we emphasize, even in this simple case where the encoding may be linear, the decoding must be *nonlinear* for perfect reconstruction.

In addition to a nonlinear decoder, it is also often advantageous to use a nonlinear function

for encoding because it can require fewer than the $2d_{\mathcal{M}}$ degrees of freedom that a linear method may need. In Fig. 1.2b we show data that has been randomly sampled from $r \in [0, 10]$ and embedded in two dimensions via the Archimedean spiral $[x, y] = [r\cos(r/a), r\sin(r/a)]$ (with $a = 1$) [87]. Then we centered this data and again projected it onto one PCA mode with $\tilde{u} = U_1 U_1^T u$, where $u = [x, y]^T$. Like before, projecting onto the PCA mode results in poor reconstruction. The interesting difference, in this case, is that mapping from the PCA projection onto the manifold is no longer one-to-one. This fact is highlighted by the blue arrows in the figure. There is no way to tell whether or not data projected from the spiral onto the line came from the top or bottom of the spiral. This is a clear case where a minimal embedding requires a *nonlinear* encoding χ, for example, $h = x^2 + y^2 = r$.

1.2.4 Nonlinear dimension reduction

Now that we have emphasized the value of finding both a nonlinear mapping to the manifold coordinate system (encoder) and a nonlinear mapping back (decoder), we next review the multitude of nonlinear techniques that exist. One reason so many techniques exist is that the encoding is nonunique, so many of the methods attempt to preserve some desirable property of the dataset in the ambient space. Some popular methods include kernel PCA [186], diffusion maps [35], local linear embedding (LLE) [180], isometric feature mapping (Isomap) [208], Sammon mapping [183], t-distributed stochastic neighbor embedding (tSNE) [211], Uniform Manifold Approximation and Projection (UMAP) [141], and undercomplete autoencoders [74, 90]. We note that many of these techniques only approximate χ and require special extensions to approximate χ^{-1}, which we discuss when appropriate. In what follows we first describe all of these techniques and try to explain possible reasons for selecting one over another. In Sec. 1.4 we choose to use autoencoders, but, with some modifications, any of these methods could work in the framework we describe.

Kernel methods

First, we discuss kernel PCA, as many of these methods are similar to this approach. In kernel PCA the covariance matrix from PCA (X^TX) is replaced with a kernel matrix $K(u_i, u_j)$. We then compute the eigenvectors of this kernel matrix (v_{ik} is the ith element of the kth eigenvector). Finally, we compute the encoding using the eigenvectors with $h = [\sum_i^M v_{i1}K(u_i, u), ..., \sum_i^M v_{id_h}K(u_i, u)]$. As a reminder, M is the size of our dataset. This procedure is an application of the "kernel trick" on the covariance matrix between snapshots of data that has been mapped into a higher-dimensional (often infinite-dimensional) "feature space" [212]. A common example of a kernel is the polynomial kernel $K(u_i, u_j) = ((u_i, u_j) + R)^d$, with scalar parameters R and d [188]. The usefulness of kernel PCA as an encoder will depend upon the selection of the kernel, which is problem dependent.

The methods of diffusion maps, LLE, and Isomap can all be viewed as using special kernels in kernel PCA [212]. First, diffusion maps use a Gaussian kernel to generate the matrix $A_{ij} = \exp(-||u_i - u_j||/2\epsilon)$, in which ϵ is a tuning parameter corresponds to the local connectivity between data points [56]. As with kernel PCA, we again use the eigenvectors of this kernel matrix A. Diffusion maps are slightly different in that we first normalize A such that the columns add to one, and then define the encoding to be $h_i = [v_{i2}, ..., v_{i(d_h+1)}]$. We remove the first eigenvector because the normalization of A makes it the trivial all-ones vector. Diffusion maps are frequently used in tasks like reducing the dimension of molecular simulations because configurational microstates that are close in the ambient space remain close in the encoded space [56].

In the next method, LLE, a set of weights ($W_{i,j}$) are found to minimize $\sum_i ||u_i - \sum_j W_{ij}u_j||^2$, where $W_{ij} = 0$ if u_j is not a neighbor of u_i (i.e. the weights minimize the reconstruction error with a linear combination of the k nearest neighbors). Then, the objective is to find an encoding that minimizes $\sum_i ||h_i - \sum_j W_{i,j}h_j||^2$ using the W calculated in the first step [180]. The relationship between this method and kernel PCA is that this encoding can be found by computing the eigenvectors of the kernel $(I - W)^T(I - W)$ (W is

a function of u_i and u_j in the first step). Then, we compute the encoding using the eigenvectors $h_i = [v_{i,1}, ..., v_{i,d_h}]$. LLE may be a useful method when the local structure is important [180].

The last kernel-based method we discuss is the Isomap, which is interesting because it uses a geodesic distance instead of a Euclidean distance. The kernel matrix in this approach uses the double-centered geodesic distance [27, 208] $K(u_i, u_j) = -1/2HD(u_i, u_j)^2H$. In this kernel, $D(u_i, u_j)$ is the geodesic distance. To compute geodesic distance, first, a graph of the k-nearest neighbors between points is constructed, weighted by their Euclidean distances. Using this graph, the geodesic distance between points is computed to be the shortest pathway between any point along the graph. Then D is centered by the matrix $H = 1/NI - \mathbb{1} \cdot \mathbb{1}^T$ ($\mathbb{1} = [1, ..., 1]^T$), and, using the eigenvector decomposition of K, we compute the encoding $h_i = [\sqrt{\lambda_1} v_{i1}, ..., \sqrt{\lambda_{d_h}} v_{id_h}]$. As a geodesic distance is the shortest path between two points on a curve, this approach may be beneficial for highly curved surfaces [65].

A few major drawbacks of these kernel-based methods are they do not provide the decoding $\check{\chi}$ (the estimate of χ^{-1}), they do not provide a method for adding out-of-sample data, and they do not scale well for large amounts of data [212]. Ad hoc methods must be added in order to solve the decoding and out-of-sample problems. Out-of-sample (i.e not used in the kernel) data can be added using the Nyström extension [9]. However, using the Nyström extension results in different functions for χ depending on whether the data is in-sample or out-of-sample. A method to solve the decoding problem is to use a neural network, as we will discuss when describing autoencoders.

All of these kernel-based methods require solving an eigenvalue problem. This is a beneficial trait in that we have robust means to find unique solutions to this problem [210]. However, the computational cost of solving this problem scales poorly with large datasets (as much as $\mathcal{O}(M^3)$ operations) [158]. Another barrier is we must compute kernels with a memory cost that scales as $\mathcal{O}(M^2)$, which becomes infeasible with the large number of snapshots often needed. The next set of algorithms are not subject to these computational

and memory costs. In the next set of algorithms, the minimization problems are solved using some form of gradient descent or quasi-Newton method. This means these methods are not guaranteed to converge to a global minimum [74].

Nonconvex methods

Sammon mapping has been characterized as the first nonlinear method for manifold learning [65]. In this method, the objective is to find h that minimizes

$$\min_{h_i} \frac{1}{a} \sum_{i=1,j<i}^{M} \left(d_u\left(u_i, u_j\right) - d_h\left(h_i, h_j\right)\right)^2 / d_u\left(u_i, u_j\right), \tag{1.9}$$

where a is a scalar normalization and d_u and d_h are metrics in their respective spaces [65, 183]. In the original work, these metrics were just taken to be the Euclidean distance. This approach is interesting because h is initialized with some value and then optimized using a gradient-based method. As mentioned, this means the algorithm can result in different encodings depending upon the initialization of h. Also, the encoding function χ corresponds to whatever optimization algorithm is used to solve eq. 1.9, which could be expensive to compute and again does not allow for out-of-sample data. However, other work has shown neural networks can be used to approximate this encoding function [42].

Similar to Sammon mapping, tSNE also varies the low-dimensional state representation h to solve an optimization problem [211]. In tSNE, the objective is to match the distribution of the data in the high-dimensional space to the data in the low-dimensional space. This is done by finding h that minimizes the Kullback-Leibler (KL) divergence $\sum_i \sum_j p_{ij} \log p_{ij}/q_{ij}$, where p and q are the distributions of the high- and low-dimensional data respectively. Numerically, these distributions are approximated in the high-dimensional space with $p = \exp(-||u_i - u_j||^2/2\sigma^2)/\sum_{k\neq l}\exp(-||u_k - u_l||^2/2\sigma^2)$ and in the low-dimensional space with $q = (1 + ||h_i - h_j||^2)^{-1}/\sum_{k\neq l}(1 + ||h_k - h_l||^2)^{-1}$. If the t-distribution for q is replaced with the distribution for p this constitutes the original stochastic nearest neighbor embedding

(SNE) [89]. Typically, tSNE is used for visualization because, like all the previous methods, it does not provide a decoder. In visualization tasks, tSNE has seen wide use because it often separates complex data sets out into visually distinct clusters [211].

The last method we mentioned in this class is UMAPs. Like tSNE, the underlying idea is to first construct a topological representation of the high-dimensional data u, then optimize the layout of the data in the low-dimensional space to minimize the error between the two representations [141]. We do not delve into the details of UMAPs, but they differ from tSNE in their approximation of the high-dimensional and low-dimensional densities. Specifically, the densities in UMAPs do not have to use Euclidean distances, do not require measuring the distance between all points, and factor in the distance between neighbors. These modifications, among others, result in a faster algorithm that does not have restrictions on embedding dimension and, arguably, provides a better embedding [141]. This has led to a recent increase in the use UMAPs for visualizing high-dimensional datasets, for example in Becht et al. [7].

Although many of the techniques described above are often used for visualization, these are all methods that deserve further exploration in our modeling framework. The properties like weighting local structure, accounting for geodesics, and matching distributions could all be useful for parameterizing a manifold. In particular, they could result in parameterizations that are more physically significant or parameterizations with simpler dynamics. Unfortunately, as mentioned, these methods do not approximate a decoding $\check{\chi}$ or provide an estimate of dimension $d_{\mathcal{M}}$.

Undercomplete Autoencoders

For a dimension reduction technique that addresses these issues, we turn to undercomplete autoencoders [74, 90]. An undercomplete autoencoder is an hour-glass shaped neural network (NN) that compresses high-dimensional data with an encoder $h = \chi(u; \theta_E)$ and then expands this "latent" representation back to the full space $\tilde{u} = \check{\chi}(h; \theta_D)$, with NN weights θ_E and θ_D.

This structure is appealing because it matches Eq. 1.5 and Eq. 1.6, and illuminates why we refer to the dimension reduction as encoding and the expansion as decoding. To highlight the benefits of an autoencoder, we first describe a deep feedforward NN based on Goodfellow et al. [74] (typically we just refer to this as a NN).

In a NN we have an input $x \in \mathbb{R}^{n_x}$, an output $y \in \mathbb{R}^{n_y}$, and "hidden units" $z^{(i)} \in \mathbb{R}^{n_i}$. A NN consists of a chain of functions mapping $x \to z^{(1)} \to ... \to z^{(N-1)} \to y$. The function at each step in this chain is

$$z^{(i)} = \sigma^{(i)} \left(W^{(i)} z^{(i-1)} + b^{(i)} \right), \tag{1.10}$$

which consists of a nonlinear activation function $\sigma^{(i)}$, weight matrix $W^{(i)}$, and bias vector $b^{(i)}$, and we consider $x = z^{(0)}$ and $y = z^{(N)}$. Instead of writing this chain of equations every time we use a NN, we will instead simply write this as $y = f_{\text{NN}}(x; \theta)$, where θ are the parameters $W^{(i)}$ and $b^{(i)}$. Given a set of data, x_j^* and y_j^* our goal is to find the parameters θ that minimize

$$\min_{\theta} \sum_{j=0}^{K} L(y_j^*, \tilde{y}_j), \tag{1.11}$$

where $\tilde{y} = f_{\text{NN}}(x^*; \theta)$ and K is the size of a batch of data. Often, this loss will be the mean-squared error between the true data and the prediction $L = ||y^* - \tilde{y}||^2$ averaged over a batch of data. After finding these parameters θ, we now have an approximate function to map x to y.

The power of NNs comes from the surprising conclusion that, with sufficiently many hidden units, a NN can approximate any function with any desired nonzero amount of error [74]. This was proven in the case where the NN consists of a single layer ($x \to z^{(1)} \to y$) with a "squashing" activation function (e.g. sigmoid) and a linear output [97]. This is known as the universal approximation theorem. Unfortunately, there is no way to use the universal approximation theorem to find the proper number of hidden units or the NN parameters. In practice, we vary network architectures and optimization schemes until we achieve acceptably low errors. We emphasize that this universal approximation theorem motivates the continued

use of NNs throughout this thesis.

Now, that we have described NNs, we come back to our description of an undercomplete autoencoder. As mentioned, it consists of an encoding NN and a decoding NN. We link these together to approximate the full state with $\tilde{u} = \check{\chi}(\chi(u;\theta_E);\theta_D)$. Then the parameters θ_E and θ_D are typically found by performing a method like stochastic gradient descent to minimize a loss $L = ||u - \tilde{u}||^2$ averaged over a batch of data. This procedure then provides us with our two desired mappings χ and $\check{\chi}$. For approximating the correct dimension to use $d_{\mathcal{M}}$ we vary the size of h until we see a sharp drop in the error indicating we selected the correct dimension [128]. Due to the universal function approximation, we know that this procedure must work if χ and χ^{-1} exist because the error can be made arbitrarily small. However, as we will see in later chapters, challenges around the amount of data, the NN architecture, and the optimization procedure will prevent this from always working. We also note that different autoencoder structures exist for preserving different properties [53, 114]. However, we do not discuss these approaches here, as it is currently unclear if these approaches offer any advantages for our systems of interest.

1.3 Time evolution

So far we have described the first step in our low-dimensional modeling framework – using an autoencoder to approximate a manifold coordinate system from data. The next step is approximating the dynamics of the system in this manifold coordinate system. In this section, we do not differentiate between models with and without dimension reduction. All of the data-driven methods we discuss work for both the full-state u and the manifold coordinates h. For the these time-evolution models, we consider training data that is evenly sampled ever τ times $\{u(0), u(\tau), ..., u(M\tau)\}$, unless otherwise specified. Although this data comes from solutions to an ODE, we can write this evolution in the form of a discrete-time map

$$u(t+\tau) = F(u(t)), \tag{1.12}$$

where the right-hand side of the discrete-time map is simply the solution of integrating Eq. 1.4 forward τ. It can be useful to consider the discrete-time problem because this matches the format of our data. We first describe methods that approximate discrete-time maps, and then describe methods that approximate an ODE.

1.3.1 Approximating discrete-time maps

The simplest way to model the discrete-time system is in a linear fashion $u(t+\tau) = Au(t)$, which can be done using the dynamic mode decomposition (DMD) [117]. This method can be used for forecasting when the system decays to a fixed point or exhibits quasiperiodic dynamics with discrete frequencies. For more complicated systems the state u can be augmented with a set of nonlinear observables in a method called extended DMD [234]. For some nonlinear systems, the correct choice of observables (i.e. change of basis) can result in a linear dynamical system (e.g. Cole-Hopf transformation of the Burgers' equation [36, 95]). However, when the observables are unknown Lusch et al. [137] showed an undercomplete autoencoder could be used for a few simple nonlinear systems to learn a change of basis under which the dynamics are linear.

Surprisingly, for any nonlinear system – even chaotic ones – there always exists a linear *Koopman operator* [20]. For finite-dimensional dynamical systems, the Koopman operator is a linear map of the infinite-dimensional space of scalar functions on the state that evolves these functions forward in time. In essence, to use the Koopman operator we trade nonlinear finite-dimensional dynamics for linear infinite-dimensional dynamics. This creates the problem of how to determine observables and approximate the Koopman operator, which is essentially the goal of extended DMD. So-called "dictionary learning" methods have been used in the extended DMD framework to map the state to the best finite-dimensional set of observables with NNs when approximating the Koopman operator [125, 238]. Our goal is to reduce dimension, so we do not take any of these linear approaches.

Now we turn to nonlinear time-evolution methods. Two approaches that have shown

excellent performance in modeling chaotic systems include reservoir networks [162] and recurrent neural networks (RNN) [220, 222]. We group these methods together because they are discrete-time maps that are non-Markovian (i.e. the future state depends on a history of past states). In reservoir computing, this non-Markovian nature comes from the reservoir state that contains "memory" of the system. Specifically, reservoir computing involves finding a high-dimensional reservoir state $r(t) \in \mathbb{R}^{d_r}$, and finding a mapping from r to u. This reservoir state is evolved forward in time according to $r(t+\tau) = G(r(t), W_{\text{in}}u(t))$, with fixed functions G and W_{in} chosen a priori. Then, the reservoir is mapped back to the state with $\tilde{u}(t+\tau) = W_{\text{out}}(r(t+\tau); p)$. An optimization procedure is used to find the parameters of this function p that minimize $\langle ||u(t+\tau) - \tilde{u}(t+\tau)||^2 \rangle$. Pathak et al. [162] used a reservoir network for modeling chaotic dynamics in the KSE. Doing this involved a reservoir state of dimension $d_r = 5000$ for a state space dimension of $d = 64$ – a significant increase in the dimension of the problem.

In RNNs there is a "hidden" state $h_r \in \mathbb{R}^{d_{h_r}}$ that is evolved forward in time by $h_r(t+\tau) = \sigma_h(u(t), h_r(t); W_h)$. Then the full state is estimated from the hidden state by $\tilde{u}(t+\tau) = \sigma_u(h_r(t+\tau); W_u)$ [220]. This is similar to reservoir networks in that some term with memory of the system is evolved instead of the state directly, and then a mapping is found to recover the state. In the case of RNNs, the functions σ_h and σ_u depend on the type of RNN but include some set of NNs with weights W_h and W_u. Again, these parameters come from an optimization procedure that seeks to minimize $\langle ||u(t+\tau) - \tilde{u}(t+\tau)||^2 \rangle$. Some popular examples of RNNs include the long short-term memory (LSTM) [91] and the gated recurrent unit [26].

A detailed comparison of these two methods can be found in Vlachas et al. [220]. In that work, Vlachas et al. found both methods can predict chaotic dynamics over multiple Lyapunov times (the relevant timescale for chaotic systems). As mentioned the key drawback of these methods, in our case, is the non-Markovian nature results in a system that is not minimum dimensional, and may be even higher than the original state space dimension [162].

Furthermore, we know our systems are Markovian, so it is unappealing to select a method that is not. These methods also have some technical challenges in that they typically require evenly spaced data for training, and start-up often requires multiple states instead of a single initial condition.

A simpler method is to train a dense NN to approximate the discrete-time map $F(u)$. We take this approach in Ch. 2. This approach has been used by many others, for example, Pérez De Jesús and Graham [163] used a discrete-time map for a two-dimensional turbulent flow, and Srinivasan et al. [198] compared using dense NN (they refer to it as a multi-layer perceptron) and an LSTM on a low-dimensional model of wall-bounded turbulence. A drawback of these methods is, after training, the model can only be used to predict the state at discrete times forward, which is especially undesirable if our data is sparse in time. To avoid this, we instead prefer to discover an ODE instead of a discrete-time map.

1.3.2 Approximating ODEs

Before discussing the strictly data-driven methods, we highlight some methods that use the equations of motion Eq. 1.4 with dimension reduction. When using linear dimension reduction a classical approach to modeling is projecting onto a low number of modes with the Galerkin projection [93]. These modes can either be prescribed a priori or come from data (e.g. PCA) [66, 193]. The accuracy of the Galerkin approach can be improved by estimating the coefficients of the truncated modes using nonlinear Galerkin [77, 103, 138] or post-processing Galerkin [63]. Alternatively, the estimation of the coefficients of these truncated modes can be approximated with LSTMs [228]. When a nonlinear dimension reduction technique is used, as opposed to these linear methods, Lee and Carlberg [121] showed how to use the equations in the manifold Galerkin technique, as described in Sec. 1.1. Finally, without dimension reduction, it is popular to use the equations of motion in "physics-informed neural networks" [173]. We choose to find models strictly from data in this thesis, so we do not take any of these approaches.

In regards to data-driven methods, we first discuss methods where the known "data" is the right-hand side of the ODE du/dt. Assuming that this is known is somewhat restrictive, but if data is spaced closely it can be estimated with finite differencing or interpolation. One of the first works to take this approach was in Gonzalez-Garcia et al. [73], who approximated $f(u)$ for a regime of the KSE where the dynamics are periodic. They used a dense NN for this approximation that took the state and spatial derivatives as the input. Similarly, Raissi et al. [172] used snapshots of closely spaced data in a multistep time-integration scheme to estimate the right-hand side, which they then trained a NN to approximate.

A drawback of using NNs is the functional form of the right-hand side is difficult to interpret. A method that attempts to overcome this issue is "Sparse Identification of Nonlinear Dynamics" (SINDy) [15]. In SINDy, sparse regression is used to identify a set of dominant terms from a predefined dictionary of candidate functions. For low-dimensional systems (three and below), Champion et al. [22] showed SINDy could be used with dimension reduction via autoencoders. Also, if spatial derivatives are included in the dictionary of candidate functions, this method is referred to as "PDE functional identification of nonlinear dynamics" (PDE-FIND) [182]. Rudy et al. [182] showed PDE-FIND could successfully predict the dynamics of the viscous Burgers equation and the KSE. We choose not to use these SINDy-based methods because it requires the time derivative, requires the selection of a dictionary of nonlinear functions (which must include the "correct" terms), and the interpretation of the resulting equation still remains a difficulty for systems that exhibit chaotic dynamics.

To overcome the issue of needing the right-hand side of the ODE (or closely spaced data), we use "neural ODEs" [24]. In neural ODEs, we approximate an ODE with a NN (e.g. $du/dt = f_{\text{NN}}(u; \theta_f)$). An ODE solver integrates this neural ODE forward in time to predict the state at some time τ in the future $\tilde{u}(t+\tau)$. This prediction is then used for computing a loss, which is often the MSE given by $||u(t_i+\tau) - \tilde{u}(t_i+\tau)||^2$ averaged over a batch of data. By computing the loss directly from snapshots of data, this method overcomes the difficulty of needing the right-hand side for training.

The framework for optimizing the parameters of the neural ODE was presented in Chen et al. [24]. This work addressed how to compute the gradient of the loss with respect to the parameters of the NN using an adjoint method. However, we note that the standard automatic differentiate tools used to typically train NNs can be used for neural ODEs at a higher memory cost than the adjoint method. In many situations, neural ODEs are used simply as function approximations where some arbitrary time is set to integrate the neural ODE [48]. However, as should be clear, this method is also powerful for predicting dynamics. Some relevant uses of neural ODEs for time prediction include applications to Burgers equation [140], decaying isotropic turbulence [167], and flow around a cylinder in a time-periodic regime [179]. For a majority of this thesis, neural ODEs will be our method of choice because they allow for solutions to be evolved to arbitrary points in time – an especially useful trait for varying the actuation time in control problems.

1.4 Framework

In the following two sections, we outline the general approach to finding low-dimensional models of chaotic dynamical systems on manifolds that we refer to as "data-driven manifold dynamics" (DManD), and then the general approach we take to incorporating this method with control (in particular reinforcement learning (RL)). Here we note that in Ch. 2 and Ch. 3 we do not refer to the modeling approach as DManD, but it is the same approach. Also, the nomenclature will vary slightly for Ch. 4 and Ch. 7 because the RL literature uses different nomenclature than the dynamical systems literature. Notably, it is common in the RL literature to refer to the state as s instead of u.

1.4.1 Modeling

As mentioned in Sec. 1.1, when the dynamics of a system lie on a manifold we can exactly describe their evolution in a coordinate system parameterizing this manifold. The three

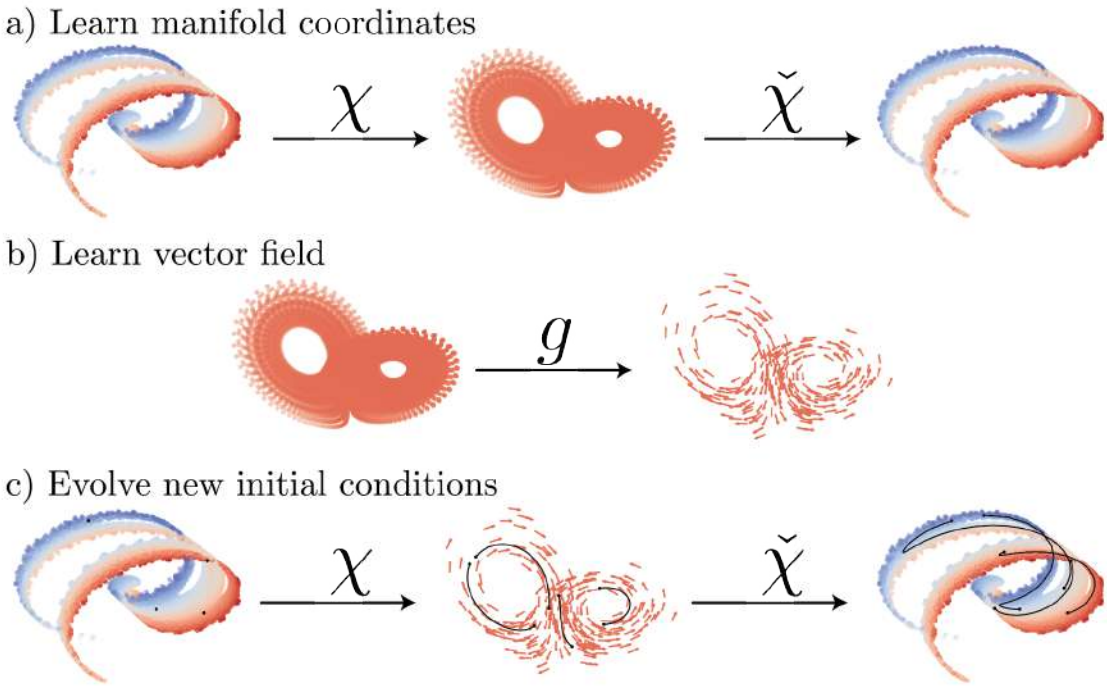


Figure 1.3: (a) Learning the three-dimensional manifold coordinates of the four-dimensional "spiral" Lorenz (color is the fourth dimension). (b) Learning the vector field in the manifold coordinates. (c) Transforming new initial conditions (black dots) into the manifold coordinates, evolving them with the vector field g (black curves), and transforming back to the ambient space.

functions needed for this procedure are a mapping to a manifold coordinate system $h = \chi(u)$, a mapping back to the full space $u = \chi^{-1}(h)$ (the approximation of which is $\tilde{u} = \check{\chi}(h)$), and a description of the dynamics in this coordinate system $dh/dt = g(h)$. Although we describe this process for continuous-time methods, in Ch. 2 we treat the dynamics as a discrete-time problem $h(t + \tau) = G(h(t))$. The DManD modeling framework consists of three steps. 1) train an undercomplete autoencoder at various dimensions, and select the dimension that shows a steep drop in the error (in the case of no steep drop multiple dimensions may need to be considered). 2) use the encoder to map all of the data to the manifold coordinate system h and train a neural ODE for time evolution of h. 3) evolve new initial conditions by mapping them to the manifold coordinate system, evolving them with the neural ODE, and then mapping them back to the full state when desired.

We illustrate this procedure in Fig. 1.3 using data from a solution of the Lorenz equation

[135] embedded in four dimensions by transforming the z coordinate with the Archimedean spiral. This mapping is given by $[u_1, u_2, u_3, u_4] = [x, y, \alpha z \cos \alpha z, \alpha z \sin \alpha z]$ where x, y, z are from the standard Lorenz system and $\alpha = 0.2$. We chose this example because three dimensions are required to exhibit chaotic dynamics (for an ODE) [201] and we want a manifold represented in an ambient dimension higher than the manifold dimension.

We plot the "spiral" Lorenz with three spatial coordinates u_1, u_3, and u_4 and color as the fourth coordinate u_2. In Fig. 1.3a we highlight that this system can be reduced to the three-dimensional Lorenz system and back without any loss of information. As mentioned, these two maps can be found with an autoencoder. Then, in Fig. 1.3b, we show how neural ODE training provides us with a vector field. Finally, Fig. 1.3c, illustrates how to combine these maps together for predicting the evolution of new initial conditions (shown in black).

1.4.2 Model-based control

Another interesting use case for these low-dimensional models, outside of simply forecasting, is for finding control policies. We leave the details for Ch. 4 and Ch. 7, but highlight the main themes in this section. The specific form of control we consider in this thesis comes from reinforcement learning (RL). In RL, a NN control agent trains through iterative interactions with an environment to maximize the discounted future reward (r) of that system [202]. The reward is some user-defined control objective for the RL agent (e.g. maximizing negative drag), and the agent interacts with the environment through actions a. One of the appealing traits of RL is that it does not require online optimization during deployment. After training, the actions are simply the output of the RL agent NN that takes the state u as an input. On the left side of Fig. 1.4 we show the basic iterative process in RL of an agent acting on an environment (in our case the FOM $du/dt = f(u)$), resulting in a new state and a reward, and then the process repeats.

The challenge with RL training is that finding near-optimal policies requires a tremendous number of interactions with the environment [47]. This could mean many interactions with

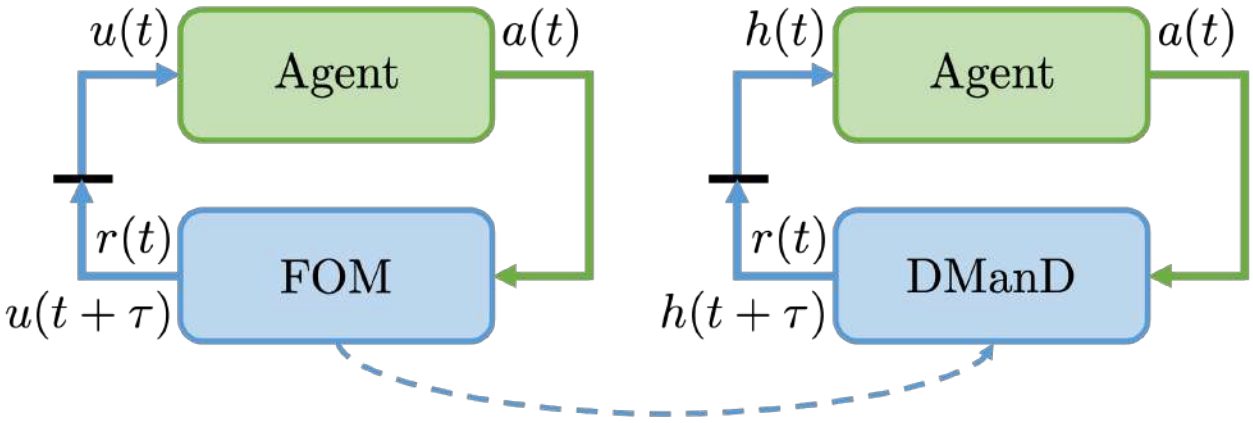


Figure 1.4: Left is the standard RL procedure of a state being fed to an RL agent that acts on an environment, resulting in a state and a reward. Right is the same procedure on the less expensive DManD model. u is the full state, h is the state in the manifold coordinates, a is the action, and r is the reward.

a high-resolution simulation or slow experiments. However, if we simply replace the FOM with the DManD model, we can drastically speed up these computations, overcoming this bottleneck. On the right side of Fig. 1.4 we show the same RL framework with the DManD model. The only major modification required for this replacement is we now make the ODE a function of the h and a:

$$\frac{dh}{dt} = g(h, a). \tag{1.13}$$

After making this change to the model, we simply follow the same steps for generating this model, use it in the RL training, and then deploy the agent learned with the DManD model on the true environment.

1.5 Outline of this work

Now that we have discussed the general framework of our DManD modeling approach we outline how the following chapters build this framework and how we use it for forecasting and control of chaotic systems. In Ch. 2 we explore how to find a manifold coordinate system using undercomplete autoencoders. Also in this chapter, we show how to predict dynamics with a NN for discrete-time mapping and how to account for some physical properties of

the KSE. In Ch. 3, we replace the discrete-time map for a neural ODE, again for the KSE. Following this, in Ch. 4 we use the KSE DManD model in conjunction with RL. Then, in Ch. 5 we develop a method for making the neural ODEs stable for long-time forecasting. We found this stabilizing approach was an invaluable step towards applying the DManD method to more complex systems. In Ch. 6 we develop DManD models for turbulent three-dimensional Couette flow, which we again use for finding RL control policies in Ch. 7. Finally, in Ch. 8 we summarize the key results of this work and discuss future directions.

Chapter 2

Parameterizing an inertial manifold [1]

In this chapter, we present a data-driven framework to represent and evolve chaotic dynamics on an inertial manifold (IM) and apply it to solutions of the Kuramoto-Sivashinsky equation. The key result of this chapter is the method we take to parameterize this IM. A hybrid method combining linear and nonlinear (neural network) dimension reduction transforms between coordinates in the full state space and a coordinate system parameterizing the IM. Additional neural networks then predict time evolution on the IM. The formalism accounts for translation invariance and energy conservation, and substantially outperforms linear dimension reduction, reproducing very well key dynamic and statistical features of the attractor.

2.1 Introduction

Partial differential equations are formally infinite-dimensional, but the presence of dissipation (through viscosity or diffusion, for example) leads to the expectation that the long-time dynamics collapse onto a finite-dimensional invariant manifold [94]. Specifically, for some systems, including the Kuramoto-Sivashinsky equation (KSE) that we consider here, it can be proven that all initial conditions exponentially approach an *inertial manifold* $\mathcal{M}$ of finite

[1] The text of this chapter is adapted from the publication by A. J. Linot and M. D. Graham *Physical Review E.*, 101, 2020

dimension $d_{\mathcal{M}}$ [60], on which the long time dynamics evolve. For states on the IM, $v \in \mathcal{M}$, one can in principle find a coordinate transformation $h = \chi(v)$ to coordinates h on the inertial manifold, a change of coordinates $v = \check{\chi}(h)$ back to the full space, and a dynamical system $h(t+\tau) = F(h(t))$ on $\mathcal{M}$. (Alternately, one could represent the dynamical system in differential form $dh/dt = G(h)$.) This dynamical system is an exact reduced-order model (ROM). Such a model can be practically useful, for computationally efficient simulations of a complex process, and may also be fundamentally important, since the coordinates h represent the key dynamical variables for the phenomenon of interest.

In the present work, we use "data" in the form of chaotic solutions to the KSE with periodic boundary conditions, to find neural-network (NN) representations of the functions $\chi, \check{\chi}$, and F. Many prior studies of inertial manifolds, and approximations thereof, take the inertial manifold to be the graph of a function Φ such that $\mathcal{M} := \{v_+ + \Phi(v_+)\}$, where $v_+ = Pv$ is a projection onto the $d_{\mathcal{M}}$ leading eigenfunctions of the linear operator for the PDE [59, 78, 241]. The present work is not subject to this restriction. Furthermore, our formalism explicitly accounts for the important physical features of translation invariance and energy conservation found in this system, and reproduces, very well, with a minimal number of degrees of freedom, key dynamic and statistical features of the attractor.

A standard machine learning method for nonlinear dimension reduction is the undercomplete autoencoder [74, 90]. This is a pair of neural networks, one mapping from a high-dimensional space to a low-dimensional one, and the second doing the reverse. The networks take data u as input, compute an output $\tilde{u}$ and are trained to minimize a loss function $L = ||u - \tilde{u}||^2$ summed over a batch of data vectors u. Autoencoders have been used for nonlinear dimension reduction in many applications [90], including turbulent flow fields [143, 154]. For dynamical systems, autoencoders have been used to explicitly yield coordinate transformations on which the dynamics are linear [137, 156] (e.g. to determine eigenmodes of the Koopman operator) or have a sparse representation [22]. Gonzalez and Balajewicz [72] have combined autoencoders with nonlinear time-evolution models for the

reconstruction of dynamics of isotropic turbulence and lid-driven cavity flow. Lee and Carlberg [121] found equations that describe the dynamics of systems that used NN to reduce the dimension, and applied the method to the transient dynamics of Burgers equation. The physical interpretation of NN representations of physical phenomena has been explored in Iten et al. [101].

Other studies have focused on developing NNs for evolving equations with chaotic dynamics without nonlinear dimension reduction. An early example of this is González-García et al. [73], who used a NN to predict the right-hand side of the discretized KSE. Specifically, they input the full state of the KSE and its derivatives into a NN to predict the parameters of a Runge-Kutta method for time integration. More recently larger NN have been used for prediction. For example, Pathak et al. [162] showed that a reservoir network trained with time-evolution "data" from the KSE was capable of making excellent predictions of future time-evolution. No explicit model reduction was performed. In [136], proper orthogonal decomposition (POD) and a spectral version (SPOD), both linear dimension reduction techniques, were used to reduce the dimension of fluid flow data that were then used to train a NN for time-evolution. Vlachas et al. [222] combined various linear dimension reduction approaches with a long-short term memory NN and mean stochastic modeling to keep trajectories on the attractor. Similarly, in [228] a long-short term memory NN was used in a nonlinear Galerkin approach to estimate the nonlinearity, a task often achieved by assuming lower modes evolve slowly, and iteratively solving for the higher modes [58, 78, 106, 139, 209].

Although methods exist for modeling dynamics on an IM, estimating $d_{\mathcal{M}}$ remains a difficult problem. In [116], an autoencoder is used to estimate $d_{\mathcal{M}}$ from data for the dynamics of the complex Ginzburg-Landau (CGL) equation; however, the dynamics on $\mathcal{M}$ are not modeled. A dynamical approach to determining $d_{\mathcal{M}}$ was taken in [237], where the covariant Lyapunov vectors of trajectories the KSE and the CGL were found to decompose into "physical" and "isolated" modes. Physical modes are entangled, in the sense that tangencies between them result in perturbations of a single mode affecting other modes, whereas

isolated modes lack tangencies with physical or isolated modes. This suggests the number of physical modes corresponds to $d_{\mathcal{M}}$. Expanding on that work, Ding et al. [43] found the dimension for the KSE in similar ways using Floquet vectors from an ensemble of unstable periodic orbits that are close to the chaotic attractor. The present work combines data-driven dimension reduction and time evolution using an efficient autoencoder structure that incorporates translation symmetry and energy conservation.

2.2 Formulation

Our testbed for this approach is the KSE,

$$\partial_t v = -v\partial_x v - \partial_{xx} v - \partial_{xxxx} v, \tag{2.1}$$

with periodic boundary conditions in the domain $x \in [0, L]$. We select $L = 22$, 44, and 66 because these domain sizes yield increasingly chaotic dynamics, and $d_{\mathcal{M}}$ is known at $L = 22$ [43]. Solutions to this equation are only unique to within a translation that we will represent with a phase variable $\phi \in \mathbb{R}$. This equation has an energy conservation principle: when time-averaged, the energy production rate $\mathcal{P} = \langle \partial_x v \partial_x v \rangle$ balances the dissipation rate $\varepsilon = \langle \partial_{xx} v \partial_{xx} v \rangle$. Here $\langle \cdot \rangle$ represents averaging over x. These properties are incorporated into the dimension reduction formulation as detailed below. Trajectories of (2.1) were generated using a Fourier spectral method in space and a fourth-order time integration scheme [108] with the code available from Cvitanović et al. [41]. The solution $v(x)$ is represented on a uniformly-spaced mesh of $d = 64$ points; we denote the solution on this mesh as $u \in \mathbb{R}^d$, so d is the dimension of the full state space in the present system.

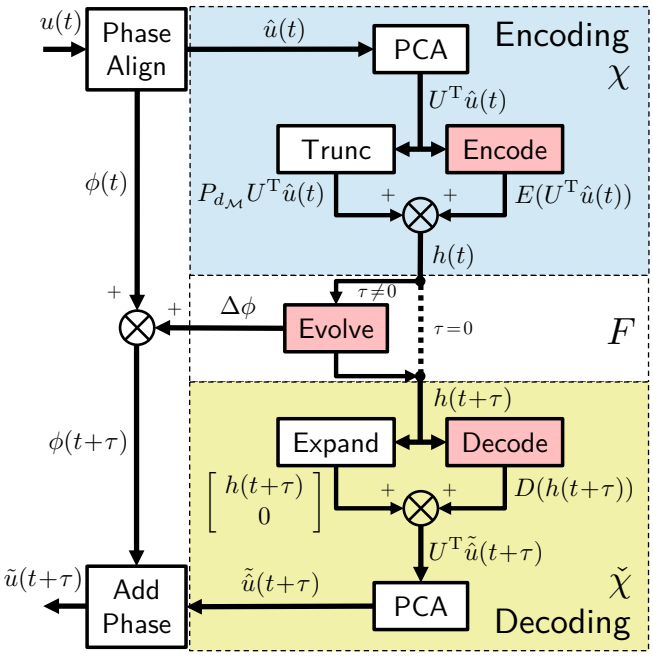


Figure 2.1: Block diagram for the hybrid autoencoder and time-evolution scheme. NNs are pink (light gray).

2.3 Methodology and results

Figure 2.1 illustrates our framework for finding the inertial manifold and the dynamical system on it. The first step of the process exploits translation invariance: the solution u at every time instant is transformed into a pattern $\hat{u} \in \mathbb{R}^d$ and a phase ϕ using an approach called the "method of slices" [18, 19]. Factoring out the phase leads to a more compact representation of the data by eliminating the need of training redundant weights for translated signals. (E.g., the representation will not need to separately represent $\sin 2\pi x/L$ and all of its translations.) Furthermore, such "symmetry reduction" methods have been found to help elucidate the state space structure in fluid mechanics problems such as pipe flow [235].

The method of slices involves taking the discrete Fourier transform $\mathcal{F}$ of the data in x to yield $a(t) = \mathcal{F}\{u(t)\}$. With the data in Fourier space, the phase of the first Fourier mode $a_1(t)$ is found using $\phi(t) = \mathrm{atan2}(\mathrm{Im}(a_1(t)), \mathrm{Re}(a_1(t)))$. Now we can construct a phase-aligned solution $\hat{u}(t)$ so that its first Fourier mode is a pure cosine: $\hat{u}(t) = \mathcal{F}^{-1}\left\{a(t)e^{-ik\phi(t)}\right\}$. Storage of ϕ for each time instant allows conversion of $\hat{u}$ back to u (i.e. $u(t)$ contains both $\phi(t)$

and $\hat{u}(t)$). Times when $a_1(t)$ approaches zero require special treatment, as was recognized by Budanur et al. [19]. We use the solution they proposed, which is to stretch time according to $\Delta\hat{t} = \Delta t/|a_1(t)|$. The rescaled time $\hat{t}$ is called "in-slice" time. Given any $u(t)$ we can always find $\hat{u}(t)$, $\phi(t)$ and $a_1(t)$, so it is always possible to move back and forth between the original and phase-aligned solutions and between real and in-slice time. The data used for training the NNs were $\sim 4 - 5 \cdot 10^5$ solutions u separated by $\Delta\hat{t} = 0.2$ "in-slice" time units, which corresponds to $\Delta t \approx 0.023$ time units on average for $L = 22$. Data was gathered after the dynamics had settled onto the attractor.

Given the phase-aligned pattern data $\hat{u}$, the first machine learning task is to find the manifold $\mathcal{M}$, of dimension $d_{\mathcal{M}}$, on which this data lives, or equivalently the coordinate transformations $h = \chi(\hat{u})$ and $\hat{u} = \check{\chi}(h)$. Because the phase of any given data vector u is arbitrary, the phase information is not needed for this step. The coordinates $h(t)$ and the phase $\phi(t)$ completely describe the state of the system so the dimension of the attractor in the unreduced state space will be $d_{\mathcal{M}} + 1$. (For further discussion of invariant manifolds in translation-symmetric systems, see Ref. [19].) Indeed, phase alignment allows a more efficient representation of the data, because phase information need not be encoded – it is captured separately as noted above.

To find χ and $\check{\chi}$, we use a variant of a standard undercomplete autoencoder, shown as the "$\tau = 0$" branch of architecture shown in Figure 2.1. This variant uses a NN to represent the *difference* between the data and its projection onto the basis arising from principal components analysis (PCA) of the data set [199]. PCA is widely used for linear dimension reduction because it yields the projection of dimension d_h that minimizes the mean squared deviation from the original data. Let U be a square orthogonal matrix whose columns are the PCA basis vectors, and $P_{d_h}U^{\mathrm{T}}$ and $P_{d-d_h}U^{\mathrm{T}}$ the projection onto the first d_h and last $d-d_h$ such vectors, respectively. The encoding step learns the function $E(U^{\mathrm{T}}\hat{u}(t))$ such that

$$E(U^{\mathrm{T}}\hat{u}(t)) = h(t) - P_{d_h}U^{\mathrm{T}}\hat{u}(t). \tag{2.2}$$

Table 2.1: Architectures of the NNs. "Shape" indicates the dimension of each layer, and "activation" the corresponding activation functions (sig is the sigmoid activation) [74]. Decoder1 refers to domains $L = 22$ and 44. Decoder2 refers to $L = 66$.

	Function	Shape	Activation
Encoder	E	$d/500/d_h$	sig/tanh
Decoder1	D	$d_h/500/d$	sig/linear
Decoder2	D	$d_h/500/500/d$	sig/linear
Evolution	F_h	$d_h/200/200/d_h$	sig/sig/linear
Evolution	F_ϕ	$d_h/500/50/500/1$	sig/sig/sig/linear

This structure is shown inside the blue (upper) box in Figure 2.1. It must be emphasized that there is no approximation in choosing this representation. Furthermore, U need not come from PCA; for example, U could be the discrete Fourier transform operator or simply the identity, the latter corresponding to using the solution values on the mesh points.

The decoding step takes the data $h(t) \in \mathbb{R}^{d_h}$ in the inertial manifold coordinates and transforms it back to the full space, as shown in the yellow (lower) box in Figure 2.1. Again one can think of learning a difference: the decoder learns a function $D(h(t))$ such that

$$D(h(t)) = U^{\mathrm{T}}\tilde{\hat{u}}(t) - \begin{bmatrix} h(t) \\ 0 \end{bmatrix}. \tag{2.3}$$

Taking $E(U^{\mathrm{T}}\hat{u}(t)) = 0$ recovers the original IM formulation, but precludes the representation of curved manifolds that do not have a one-to-one mapping from a linear projection. An example of such a manifold is the Archimedean spiral, whose Cartesian representation is $(x, y) = (\phi\cos\phi, \phi\sin\phi)$. The autoencoder architecture used here is able to represent this manifold with $d_h = 1$.

Finally, inserting Eq. 2.2 into Eq. 2.3, solving for $\tilde{\hat{u}}_t$, and noting that this can be written $\tilde{\hat{u}}(t) = U\left[P_{d_h}U^{\mathrm{T}}\hat{u}(t), P_{d-d_h}U^{\mathrm{T}}\hat{u}(t)\right]^{\mathrm{T}}$, shows that the exact solution satisfies $E(U^{\mathrm{T}}\hat{u}(t)) + D_{d_h}(h(t)) = 0$, where D_{d_h} contains the first d_h components of D. This constraint can be satisfied approximately by adding a penalty term to the autoencoder loss function so it

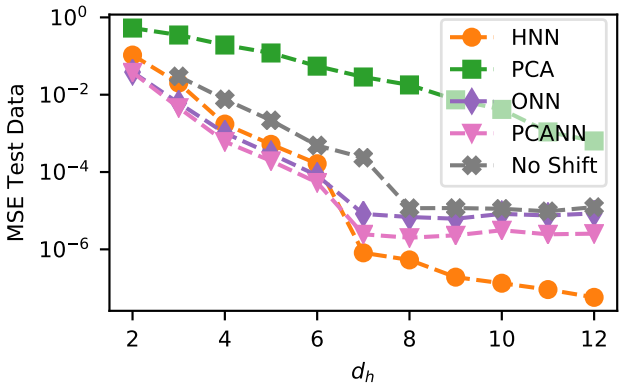


Figure 2.2: MSE of test data for various d_h for $L = 22$. The legend is described in the text.

becomes

$$L = ||\hat{u}(t) - \tilde{\hat{u}}(t)||^2 + \alpha||E(\hat{u}(t)) + D_{d_h}\left(h(t)\right)||^2. \tag{2.4}$$

With this structure, we can in principle achieve an exact representation (within the approximation error of the functions E and D) of data on a manifold of dimension $d_{\mathcal{M}}$ for all $d_h \geq d_{\mathcal{M}}$. In general, the functions E and D, or more generally χ and $\breve{\chi}$, need not come from NNs. Other approaches to nonlinear dimension reduction and function approximation (e.g. tSNE, diffusion maps, kernel regression [211, 212]) might be useful as well. The overall structure of our approach would be the same.

Autoencoders of the above structure, which we denote hybrid neural networks (HNN) were trained (i.e. the functions E and D were determined) using the phase-aligned data. At a given value of d_h, twenty HNNs (each initialized with different initial guesses for the weights), with $\alpha = 1$ were trained for 1000 epochs with an Adam optimizer using Keras [31]. This process was repeated for a range of d_h. Results are reported for the model with the lowest MSE at each value of d_h.

For comparison, we trained three variations on the HNN to evaluate the effect of the linear projection (P_{d_h}), the PCA change of basis (U^{T}), and phase alignment steps. In the first variation, denoted PCANN, we built a NN without the "Trunc" and "Expand" blocks in Fig. 2.1, which corresponds to using the PCA basis, but using E and D to learn the whole nonlinear coordinate transformation rather than just the difference from PCA. The next

variation builds upon the previous and removes the "PCA" block in Figure 2.1, which leaves it in the original basis, so we denote it ONN. Then, the last variation is to remove the phase shift (No Shift). Both the PCANN and the ONN are trained with the loss $L = ||\hat{u} - \tilde{\hat{u}}||^2$, while the unsifted variation is trained with $L = ||u - \tilde{u}||^2$. Hyperparameter tuning of the NN architectures was performed manually by varying width, depth, and activation functions. All of these variations used the same architecture, shown in Table 2.1, for functions E and D.

Figure 2.2 shows the mean squared error (MSE) on a separate test dataset for the NN methods described above and PCA for $L = 22$. At low d_h, the HNN, PCANN, and ONN all perform similarly, and in all three cases, the MSE drops significantly at $d_h = 7$. For the case of no shifting, the drop appears at $d_h = 8$ because the continuous translation symmetry has not been factored out. All NNs perform orders of magnitude better than PCA. On continuing to increase d_h, the MSE for the HNN continues to improve while the others stagnate, because the HNN only needs correct coefficients of the less relevant higher PCA modes, while the other methods modify all of them. Notably, the abrupt drop in MSE at $d_h = 7$ coincides with the true dimension $d_{\mathcal{M}}$ of the attractor as found in [43]. The remaining error for $d_h \geq 7$ for the HNN is small, at $O(10^{-7})$, which follows from the fact that at this dimension an exact coordinate transformation exists, so the remaining error is approximation error.

Having in hand the coordinate representation for points on $\mathcal{M}$, we now use NNs to learn the dynamical system ("exact" reduced-order model) on the manifold, corresponding to the "$\tau \neq 0$" branch in Figure 2.1. This approach will be denoted "HNN ROM". We construct discrete time mappings

$$h(t + \hat{\tau}) = F_h(h(t)), \quad \Delta\phi \equiv \phi(t + \hat{\tau}) - \phi(t) = F_\phi(h(t)), \tag{2.5}$$

where F_h and F_ϕ have the architectures shown in Table 2.1. We use the symbol $\hat{\tau}$ instead of τ to emphasize that the discrete-time mappings use in-slice time. We chose $\hat{\tau} = 2$, which

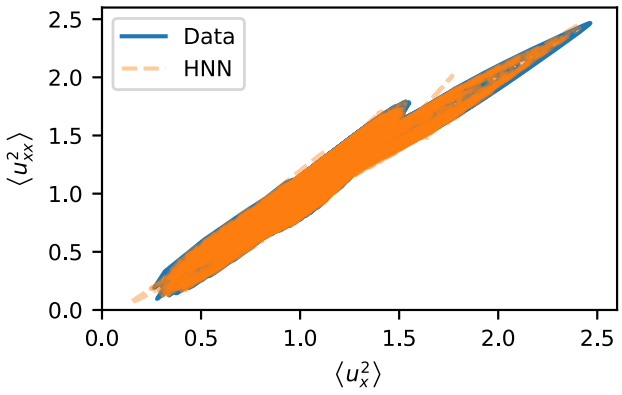


Figure 2.3: $\mathcal{P}$ vs. ε state-space projection for the data at $L = 22$ and HNN ROM prediction with $d_h = 7$.

reproduces trajectories well, by allowing for the signal to change an appreciable amount, but not too much, in one time interval. Setting $\hat{\tau}$ much smaller or larger results in poor model predictions.

Recall that the energy balance for the KSE requires that the production and dissipation rates $\mathcal{P}$ and ε must balance on average. We incorporate this fact in the training of the dynamic models as follows. We compute the projection of the data onto $\mathcal{P}$ and ε, as shown in Figure 2.3. The relation between $\mathcal{P}$ and ε is narrowly distributed around the line $\mathcal{P} = \varepsilon$, with a sharp boundary, and we can find maximum and minimum dissipation rates $\varepsilon_{\max}$ and $\varepsilon_{\min}$ associated with a given value of $\mathcal{P}$. We then add a penalty for crossing this boundary to the loss function L_F for F_h and F_ϕ, as follows:

$$L_F = ||u(t+\hat{\tau}) - \tilde{u}(t+\hat{\tau})||^2 + \beta \max(\max(0, \tilde{\varepsilon} - \varepsilon_{\max}(\tilde{\mathcal{P}})), \varepsilon_{\min}(\tilde{\mathcal{P}}) - \tilde{\varepsilon}), \tag{2.6}$$

where $\tilde{\mathcal{P}}$ and $\tilde{\varepsilon}$ are calculated from $\tilde{u}$. We selected $\beta = 0.1$ so the second term contributed the same order of error to the loss as the first term. For each d_h, the best dimension reduction model was chosen, and fifty time-evolution models were trained for 200 epochs. Results are reported for the best models, as determined at a given d_h based on producing low errors in both short and long-time statistics.

To illustrate the performance of this approach, which we denote HNN ROM, on predicting

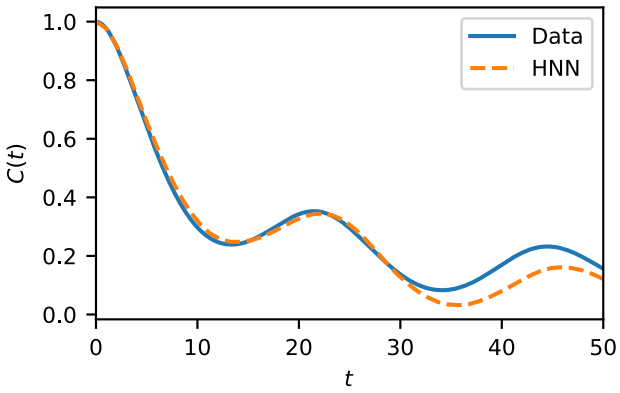


Figure 2.4: Time-correlation function for the data at $L = 22$, and HNN ROM prediction with $d_h = 7$.

dynamics, we first present short-time tracking results and then long-time statistics. All trajectories are evolved from a given initial condition $u(0)$ on the manifold, from which we find $\hat{u}(0)$ and $\phi(0)$ by phase alignment and then set $h(0) = \chi(\hat{u}(0))$. This initial condition in the manifold coordinates is evolved forward in in-slice time with Eq. 2.5. For validating the performance of the short-time tracking we need a timescale for comparison. Here we consider the integral timescale $T_I = \int_0^\infty C(t)\, dt \approx 19$, where

$$C(t) = \frac{\langle u(0)u(t)\rangle}{\langle u(0)^2\rangle}$$

is the temporal autocorrelation, and the Lyapunov time $T_L \approx 21$ [43]. Figure 2.4 shows the temporal autocorrelation of the data and the HNN ROM at $d_h = d_{\mathcal{M}}$ are in good agreement for $t \lesssim 30$. Likewise, typical trajectories show close tracking for 30 or more real time units. This comparison appears in Fig. 2.5, where Fig. 2.5a and 2.5c show the evolution of two initial conditions of test data using the dynamical system found with $d_h = d_{\mathcal{M}} = 7$, and the "exact" results are shown in Fig. 2.5c and 2.5d obtained from solving the KSE. These results indicate predictive capability for time scales longer than T_I or T_L, and thus represent very good performance for prediction of chaotic dynamics.

Next, we evaluate the ability of our dynamic model to reproduce key long-time statistical properties of the attractor, focusing on the quantities u_x and u_{xx} that determine the energy

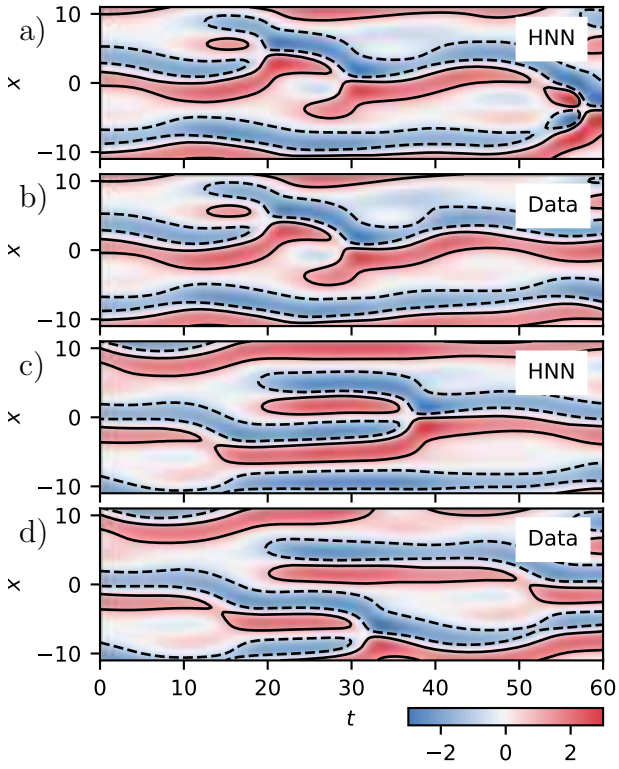


Figure 2.5: (a) and (c) trajectories (color contours of $u(x,t)$ with solid lines at $u=1$ and dashed lines at $u=-1$) evolving from different initial conditions according to the HNN ROM; $L=22$. (c) and (d) true trajectories corresponding to (b) and (d).

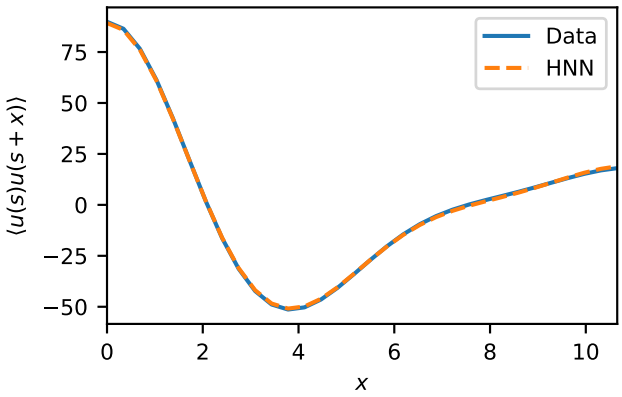


Figure 2.6: Spatial correlation function for the data and HNN ROM prediction with $d_h = 7$, $L = 22$.

production and dissipation in the KSE. We examine predictions both for $d_h = d_{\mathcal{M}}$ and for values of d_h either larger or smaller than $d_{\mathcal{M}}$. The trajectories considered here cover approximately $9 \cdot 10^3$ real time units.

Figures 2.6 and 2.3, respectively, show the spatial autocorrelation function (averaged over space and time) and the energy balance ($\mathcal{P}$ vs. ε) of the HNN ROM for $d_h = d_{\mathcal{M}}$ and for data, illustrating close agreement of these quantities. These statistics show that long-time trajectories do not diverge from the attractor and that the HNN ROM prediction stays within the envelope of the energy balance, which was the intent of the penalty in the loss for the time-evolution training, Eq. 2.6. These predictions deteriorate when $d_h < d_{\mathcal{M}}$.

A more detailed representation of the attractor is the joint probability density function (PDF) of the pointwise values of u_x and u_{xx}. Figure 2.7a shows this PDF, on a log scale, as determined from the data. At $d_h = d_{\mathcal{M}} = 7$, the HNN ROM prediction, Fig. 2.7b, is very close to the exact PDF. To highlight the effect of the nonlinear autoencoder, we also consider predictions with linear dimension reduction from PCA (i.e. $E = D = 0$), and NNs for the dynamics; we denote this approach PCA ROM. At the same dimension, Fig. 2.7d shows that the PCA ROM prediction yields much poorer results. Figure 2.7c shows how the relative L_2 difference between the true PDF and the model predictions varies with d_h. For $d_h = 5$, the predictions are poor for both cases, but for $d_h \geq 7$,the error is small and nearly unchanging for the HNN ROM case, which is unsurprising since $d_{\mathcal{M}} = 7$. On the

other hand, it takes $d_h = 14$ for the PCA ROM to yield a comparable model to the HNN ROM at $d_h = 7$. This result might be expected based on Whitney's embedding theorem, which states that any manifold of dimension k can be embedded in $\mathbb{R}^{2k}$ [80].

To investigate the generality of this method, we examine its performance for larger domains, $L = 44$ and $L = 66$. For $L = 22$ there is one positive Lyapunov exponent [43, 50], whereas the dynamics at $L = 44$ and $L = 66$ are more chaotic; at these values, Edson et al. [50] report four and seven positive Lyapunov exponents, respectively. Data gathering and NN training were performed in the same way as for $L = 22$. Initial trials for the $L = 66$ showed poor results, so the capacity of the decoder was increased, as noted in Table 2.1. Figure 2.8a shows the MSE on a test data set of HNNs with the lowest MSE at various d_h for $L = 44$ and 66. The horizontal axis is centered around the $d_{\mathcal{M}}$ of each domain size inferred from the drop in the MSE. This corresponds to $d_{\mathcal{M}} = 17$ and $d_{\mathcal{M}} = 27$ for $L = 44$ and $L = 66$. Increasing dimension still shows a distinct drop in MSE, however, it becomes less substantial with increased domain size. The $L = 66$ autoencoder performs better and has lower MSE than $L = 44$ because of the increased capacity.

The results at these domain sizes suggests that there is a linear scaling of $d_{\mathcal{M}}$ with L. This observation agrees well with the results of Yang et al. [237] where they show the number of physical modes (PM) scales linearly with domain size. Figure 2.8b shows $d_{\mathcal{M}} + 1$ (the dimension in the unreduced state space) against the domain size for both our results and their results, along with the extrapolation of their results to smaller domains. The excellent agreement between these results provides additional computational evidence that the dimension of the IM for the KSE scales linearly with L.

Finally, we show the attractor recreation with the joint PDFs of u_x and u_{xx}. Figure 2.9a and 2.9d show these for $L = 44$ and $L = 66$ for trajectories of approximately $8 \cdot 10^3$ and $6 \cdot 10^3$ real time units, respectively. Model predictions for $d_h = d_{\mathcal{M}} - 1$, Fig. 2.9b and 2.9e, and $d_H d_{\mathcal{M}}$ degrees of freedom, Fig. 2.9c and 2.9f, are shown for comparison. Unlike with $L = 22$, where models with too few dimensions tended to land on periodic orbits, here, with more

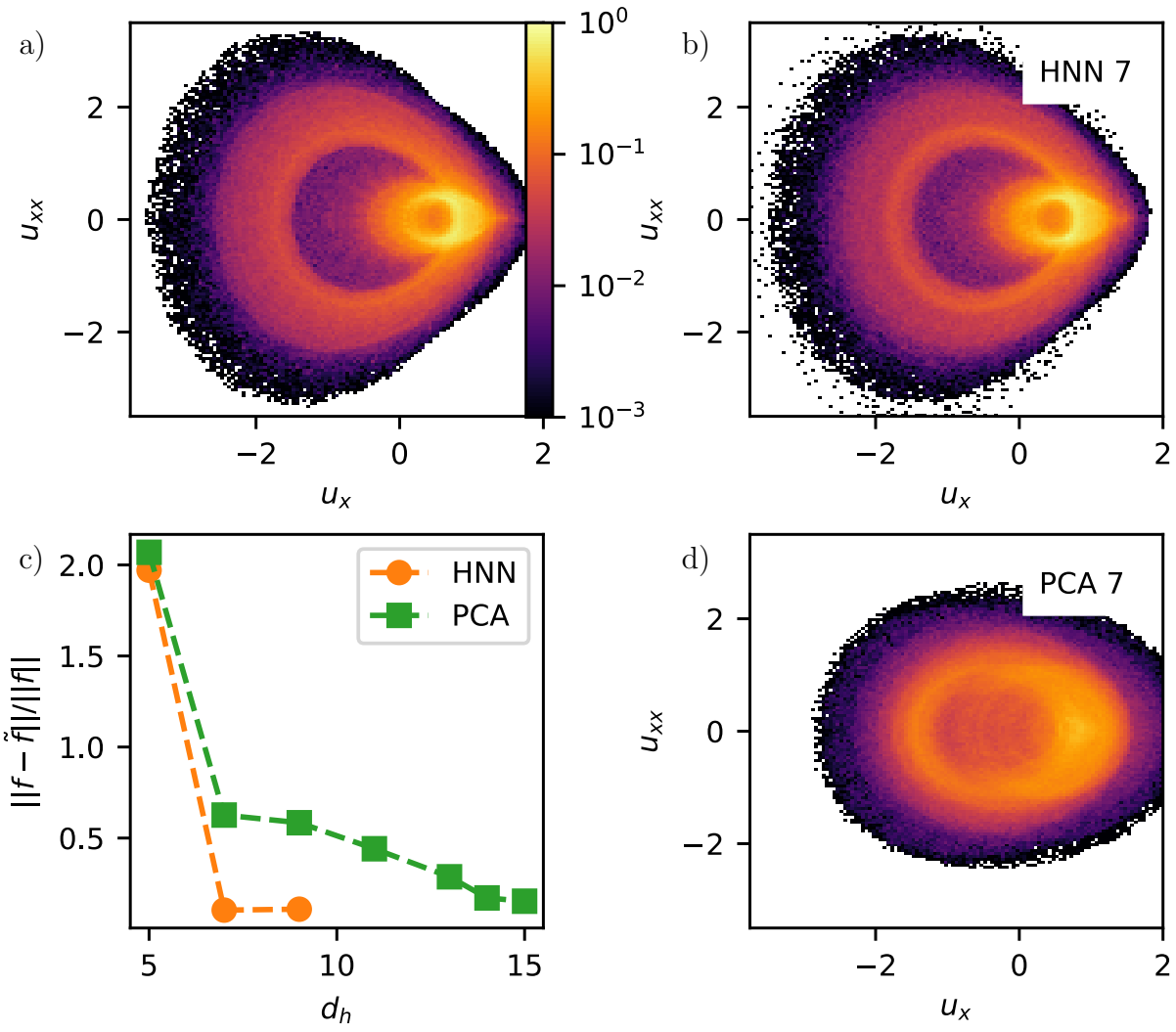


Figure 2.7: (a) joint PDF of data, $L = 22$, and (b) joint PDF of HNN ROM prediction, both plotted on a logarithmic scale. (c) Relative error in PDF vs dimension. The PDF from data is denoted f, and that from the model prediction is $\tilde{f}$.

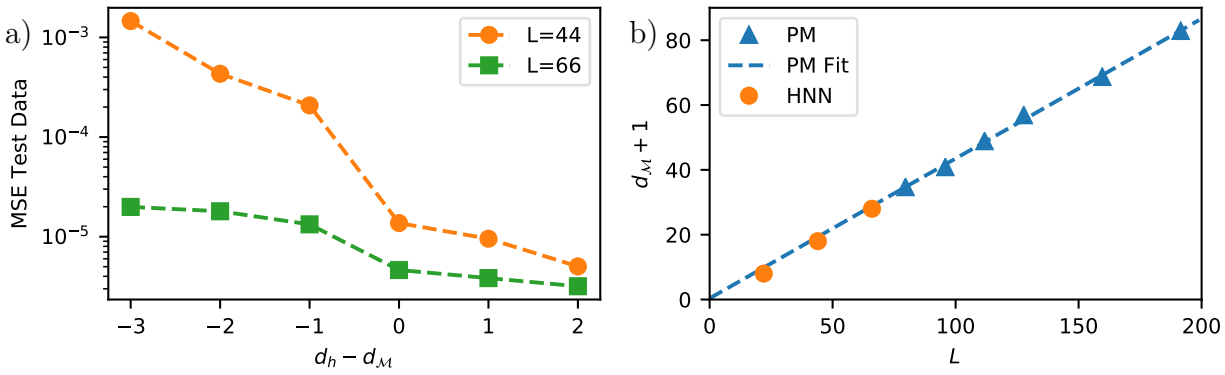


Figure 2.8: (a) MSE vs the difference between d_h and the predicted $d_{\mathcal{M}}$. (b) Dimension $d_{\mathcal{M}} + 1$ of the unreduced IM for different domain sizes: "HNN" denotes the current results, "PM" the number of physical modes from [237], and "PM Fit" a linear fit of the PM curve.

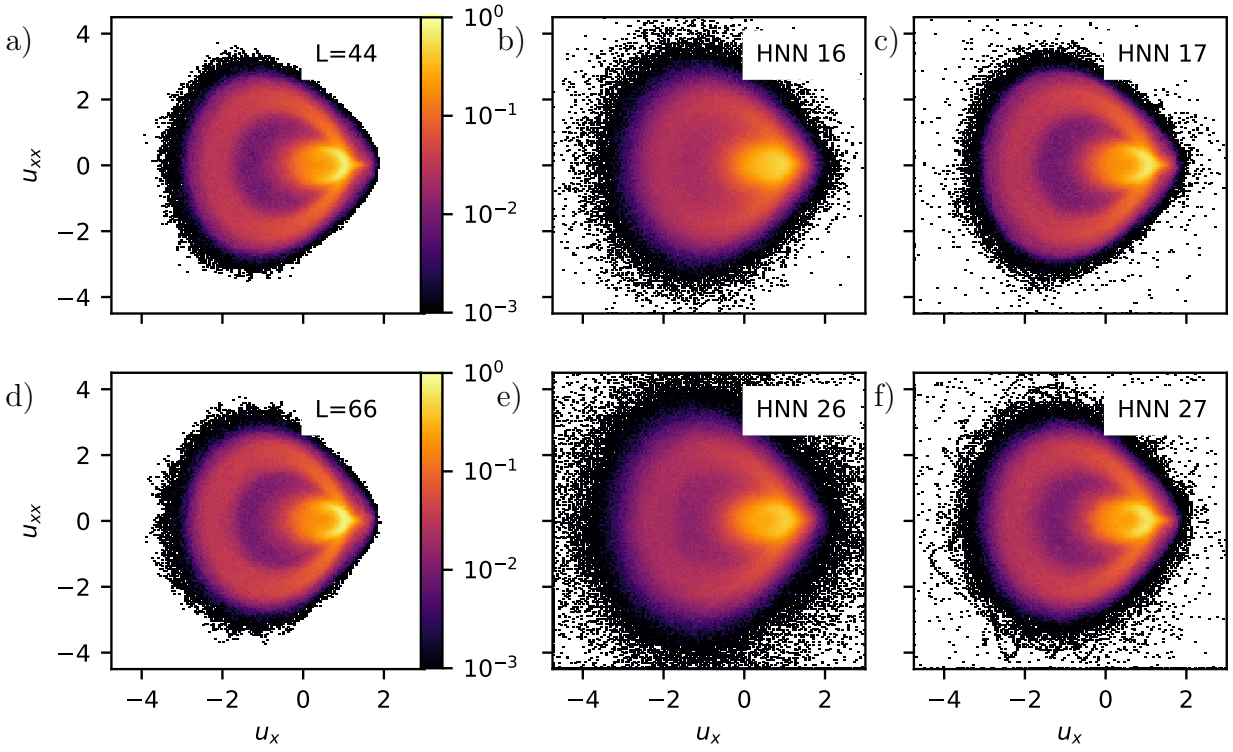


Figure 2.9: (a) and (d) joint PDF of data for $L = 44, 66$. (b), (c), (e), and (f) joint PDF of HNN ROM prediction at $d_{\mathcal{M}} - 1$ and $d_{\mathcal{M}}$ for $L = 44, 66$.

degrees of freedom, models with too few dimensions maintain chaos. However, the PDFs for $d_{\mathcal{M}} - 1$ are more diffuse, less accurately recreating the data. In both cases, when the models retain $d_{\mathcal{M}}$ dimensions the joint PDF agrees well with the data, the main discrepancy being a broader tail of the model PDF, i.e. a higher, but still very low, probability of large excursions.

2.4 Summary

In this chapter, we have shown a framework for data-driven "exact" reduction of a dynamical system onto a low dimensional invariant manifold and time-evolution on that manifold. Translation symmetry and energy conservation, two important features of many systems of interest, are incorporated naturally into the framework. By observing the model reduction error as a function of dimension, the dimension $d_{\mathcal{M}}$ of the invariant manifold can be determined, and once $d_{\mathcal{M}}$ is known, highly accurate model predictions can be obtained. In particular, key statistical quantities in a chaotic system can be well-approximated, indicating that the model dynamics capture the shape of the attractor. In this work, the NNs can be trained on a single processor in a couple days, and time-evolution over 10^4 time units takes only minutes. At present, it is difficult to predict how network size and training time will scale for more complex problems, especially given that it is not even known in general how $d_{\mathcal{M}}$ scales. Extensions to systems with higher-dimensional dynamics are underway. Systematizing this method could provide a straightforward, data-driven means of approximating the dimension of manifolds and constructing reduced order models, a difficult task for high-dimensional chaotic systems like turbulence.

Chapter 3

Dynamics with neural ODEs [1]

In this chapter, we improve upon the modeling setup of the previous chapter by treating the dynamics as an ODE instead of a discrete-time map. Again, the manifold coordinates are discovered using an undercomplete autoencoder – a neural network (NN) that reduces then expands dimension. Then the ODE, in these coordinates, is determined by a NN using the neural ODE framework. Both of these steps only require snapshots of data to learn a model, and the data can be widely and/or unevenly spaced. Time-derivative information is not needed. We apply this framework to the Kuramoto-Sivashinsky equation for domain sizes that exhibit chaotic dynamics with estimated manifold dimensions ranging from 8 to 28. With this system, we find that dimension reduction improves performance relative to predictions in the ambient space, where artifacts arise. Then, with the low-dimensional model, we vary the training data spacing and find excellent short- and long-time statistical recreation of the true dynamics for widely spaced data (spacing of $\sim$0.7 Lyapunov times). We end by comparing performance with various degrees of dimension reduction and find a "sweet spot" in terms of performance vs. dimension.

[1] The text of this chapter is adapted from the publication by A. J. Linot and M. D. Graham *Chaos*, 32, 2022

3.1 Introduction

A common question in many applications is: given a time series of measurements on a system how can a predictive model be generated to estimate future states of the system? For problems like weather forecasting, just knowing the future state of the system is useful. In other problems, like minimizing turbulent drag on an aircraft, models are desirable for creating control policies. These models can sometimes be generated from first principles, but insufficient information on the system often limits the ability to write them explicitly. Even when a model can be written out explicitly, it may be very high-dimensional and computationally expensive to simulate. Thus it is often desirable to generate a low-dimensional model from data.

We consider data sets $\{u(t_1), u(t_2), \ldots, u(t_N)\}$, where $u(t_i) \in \mathbb{R}^d$ comes from measuring the state of a system at a given time t_i. The full state at a given time can be represented by direct measurements (e.g. position and velocity of a pendulum) or by a representation that is diffeomorphic to the state. One such representation is time delay measurements of the system, as shown by Takens [203]. In the case of the pendulum, for example, this could correspond to writing the state as $u(t) = [\theta(t), \theta(t-\tau), \theta(t-2\tau), \theta(t-3\tau)]$, where θ is the angle of the pendulum and τ is the delay time. If u lies on a d-manifold then a time delay representation in $\mathbb{R}^{2d+1}$ is diffeomorphic to u [203].

For many systems of interest the state space is high-dimensional and the dynamics are chaotic. Despite this complex behavior, when these systems are dissipative the long-time dynamics often lie on a smooth invariant finite-dimensional inertial manifold $\mathcal{M} \subset \mathbb{R}^d$ of a much lower dimension ($d_{\mathcal{M}}$) than that of the state space [206]. Two such systems are the Kuramoto-Sivashinsky equation (KSE) [60, 107, 207, 241] and the complex Ginzburg-Landau equation [44]. Similarly, for the Navier-Stokes in two and three spatial dimensions, it has been shown that there is an *approximate* inertial manifold [59, 205]. It is approximate in the sense that there are small variations in many dimensions that can be assumed to be zero and still provide an accurate approximation of the state.

With a mapping to the manifold coordinate system the state can be represented (at least locally) in the manifold coordinates $h(t) \in \mathbb{R}^{d_{\mathcal{M}}}$. That is, mappings χ and $\check{\chi}$ exist such that $h = \chi(u)$ and $u = \check{\chi}(h)$. In the machine learning literature, χ and $\check{\chi}$ correspond to the the encoder and decoder, respectively, of a so-called undercomplete autoencoder structure [128], as we further discuss below. It should be noted that there is no guarantee that $\mathcal{M}$ can be *globally* represented with a cartesian representation in $d_{\mathcal{M}}$ dimensions. Indeed in general this cannot be done, and an "atlas" of overlapping local representations, or "charts", must be used [120]. The application considered here will not require this more general formalism, but for related work using charts, see [57]. Our aim here is to use data from a spatiotemporally chaotic system to learn the mappings $\chi(u)$ and $\check{\chi}(h)$ back and forth between $\mathbb{R}^d$ and a coordinate system on $\mathcal{M}$ and then to learn the evolution equation for the dynamics in this coordinate system. This will be a minimal-dimensional representation for the dynamics of the system in question. We first introduce some background for the dimension reduction problem and then for the time evolution problem.

3.1.1 Dimension reduction methods

Dimension reduction is a challenging and widely-studied problem, and many approaches have been considered. In Ch. 1 we overview a wide range of dimension reduction techniques and determine *nonlinear* techniques provide the most versatility to approximate a manifold coordinate system. In this chapter, we again use the hybrid autoencoders from Ch. 2. This approach is advantageous because it approximating the mappings to the manifold coordinates χ and back $\check{\chi}$ from data. We describe the specifics of the autoencoders used in this chapter in more detail in Section 3.2.

3.1.2 Time evolution methods

We now turn to developing, from data, dynamical models for the evolution of a state $u(t)$. We consider systems that display deterministic, Markovian dynamics, so if $u(t)$ is the full

state (either directly or diffeomorphically through embedding of a sufficient number of time delays), then the dynamics can either be represented by a discrete-time map

$$u(t+\tau) = F(u(t)), \tag{3.1}$$

or an ordinary differential equation (ODE)

$$\frac{du}{dt} = f(u). \tag{3.2}$$

The learning goal is an accurate representation of F or f, or their lower-dimensional analogs in the case that we apply dimension reduction.

Again, in Ch. 1 we explored many different techniques for performing this time evolution. Of the many discrete-time maps, we are particularly interested in generating low-dimensional data-driven Markovian models. In the previous chapter, we showed this is possible using a dense NN to approximate $F(u)$, and that we can drastically reduce the state dimension in a domain size of $L = 22$ from $u \in \mathbb{R}^{64}$ to $h \in \mathbb{R}^8$. We found this coordinate system using an undercomplete autoencoder that corrected on PCA, as described in more detail in Section 3.3.1. Representing the state in this minimal fashion allows for faster prediction, and may provide insight by limiting the system to only the essential degrees of freedom. Iten et al. [101] illustrated the latter point by recovering physical variables from data in some low-dimensional example settings.

Rather than approximating $F(u)$ in Eq. 3.1, the discrete-time representation, one can, in principle, approximate $f(u)$ in Eq. 3.2, the continuous-time representation. This is more challenging because one does not generally have access to data for du/dt, which eliminates many of the techniques we mentioned in Sec. 1.3. The task of learning $f(u)$ is closely related to the problem of *data assimilation* [4]. In that problem, the general functional form of f is given and the task is to learn a relatively small number of parameters, whereas in the present case we wish to represent $f(u)$ as an essentially arbitrary function. A framework

for doing this, with f given as a neural network, was presented by Chen et al. [24], who dubbed the approach "neural ODEs". To determine the neural network parameters of f, the difference between data and predictions is minimized. To determine the derivatives of f with respect to the parameters, either an adjoint problem can be solved, or an automatic differentiation method used. With a given f, the state evolution at arbitrary points in time can be computed as the solution to Eq. 3.2. We further describe and apply, this approach in Section 3.2.

Neural ODEs have been applied to a number of time series problems. Maulik et al. [140] used neural ODEs and LSTMs for Burgers equation, and showed that both outperform a Galerkin projection approach. Portwood et al. [167] used the neural ODE approach to find an evolution equation for dissipation in decaying isotropic turbulence, showing that it outperformed a heuristic model. Neural ODEs have also been applied to flow around a cylinder in the time-periodic regime, where velocity field simulation data were projected onto 8 PCA modes, and the time evolution of these modes was determined [179].

The present work combines nonlinear dimension reduction using autoencoders with the neural ODE method to model the dynamics of a system displaying spatiotemporal chaos, the Kuramoto-Sivashinsky equation, over a range of parameter values. In Section 3.2, we introduce the framework of our methodology. Section 3.3.1 briefly describes the results for the dimension reduction problem alone, then Section 3.3.2 uses the reduced-dimensional descriptions to illustrate the performance of the neural ODE approximation for closely spaced data. An important conclusion here is that dimension reduction can improve neural ODE performance relative to predictions in the ambient space, where artifacts arise. Section 3.3.3 examines the role of data spacing on neural ODE performance, showing that even for chaotic systems, good performance on both short-time predictions and long-time statistics can be obtained even from widely spaced data, up to a fairly well-defined limit. Finally, Section 3.3.4 shows comparisons of neural ODE performance with various degrees of dimension reduction, finding a "sweet spot" in terms of performance vs. dimension. We summarize in Section 3.4.

3.2 Framework

Here we recapitulate and extend the framework presented in Sec. 1.4. We consider data $u \in \mathbb{R}^d$ that lies on a $d_{\mathcal{M}}$-dimensional manifold $\mathcal{M}$ that is embedded in the ambient space $\mathbb{R}^d$ of the data. With the data, we find a coordinate transformation $\chi : \mathbb{R}^d \to \mathbb{R}^{d_{\mathcal{M}}}$ giving the state in the manifold coordinates, $h \in \mathbb{R}^{d_{\mathcal{M}}}$. We also find the inverse $\check{\chi} : \mathbb{R}^{d_{\mathcal{M}}} \to \mathbb{R}^d$ to reconstruct u from h. Then we describe the dynamics on $\mathcal{M}$ with the differential equation

$$\frac{dh}{dt} = g(h). \tag{3.3}$$

If we do not do any dimension reduction then Eq. 3.3 is the same as Eq. 3.2. Of course, for an arbitrary data set, we do not know $d_{\mathcal{M}}$ a priori, so in Section 3.3.4 we present results with various choices for d_h, where $h \in \mathbb{R}^{d_h}$. Using the mapping to the manifold coordinates, the evolution in the manifold coordinates, and the mapping back, we evolve new initial conditions forward in time. So, our task is to approximate χ, $\check{\chi}$, and g. We also consider the case with no dimension reduction, where we determine $f(u)$, the right-hand side (RHS) of the ODE in the ambient space. In general, there can be no expectation that any of these functions have a simple (e.g. polynomial) form, so here we approximate them with NNs, as detailed below.

Figure 1.3, in Sec. 1.4, illustrates this framework on data from a solution of the Lorenz equation [135] that we embedded in four dimensions by mapping the z coordinate of the Lorenz equation to the Archimedean spiral. The change of coordinates for this embedding is given by $[u_1, u_2, u_3, u_4] = [x, y, \alpha z \cos \alpha z, \alpha z \sin \alpha z]$ where x, y, z are the standard variables in the Lorenz equation and $\alpha = 0.2$. In Fig. 1.3a we show an example of learning the manifold coordinates for this system. The three spatial dimensions for the embedded data are u_1, u_3, and u_4 and the color is u_2. For a trajectory to be chaotic it must lie on at least a three-dimensional manifold, so the minimum embedding dimension for a chaotic trajectory that requires all the steps in this framework is four dimensions. Figure 1.3b illustrates learning

the vector field g in the manifold coordinates. In Fig. 1.3c we show how, after learning these functions, new initial conditions can be mapped to the manifold coordinates, evolved forward in time, and then mapped back to the ambient space.

To find χ and $\check{\chi}$, we represent them as NNs and find parameters that minimize a reconstruction loss averaged over a batch of data given by

$$L = \left\langle ||u - \check{\chi}(\chi(u;\theta_1);\theta_2)||^2 \right\rangle, \tag{3.4}$$

where θ_1 and θ_2 are the weights of χ and $\check{\chi}$, respectively. Here, and elsewhere, we use stochastic gradient descent methods to determine the parameters. Further details are given in Section 3.3.1. This architecture is known as an undercomplete autoencoder, where χ is an encoder and $\check{\chi}$ is a decoder [98]. Autoencoders are widely used for dimension reduction; in the fluid mechanics context they have been used for many examples including flow around an airfoil [154], flow around a flat plate [151], Kolmogorov flow [157], and channel flow [62, 143].

As noted above, to approximate g in Eq. 3.3, we use the neural ODE approach of Chen et al. [24]. In the neural ODE framework we use g to estimate $\tilde{h}(t_i + \tau)$, an approximation of the reduced state $h(t_i + \tau)$, at some time $t_i + \tau$ by integrating

$$\tilde{h}(t_i + \tau) = h(t_i) + \int_{t_i}^{t_i+\tau} g(h(t);\theta_3)dt, \tag{3.5}$$

where θ_3 is the set of weights of the NN. Then g is learned by minimizing the difference between the prediction of the state ($\tilde{h}(t_i + \tau)$) and the known state ($h(t_i + \tau)$), in the manifold coordinates, at that time. Specifically, the loss we minimize is

$$J = \left\langle ||h(t_i + \tau) - \tilde{h}(t_i + \tau)||_1 \right\rangle. \tag{3.6}$$

Here $||.||_1$ denotes the L_1-norm – of course, other norms can be used. By taking this approach we can estimate g from data spaced further apart in time than when g (or more precisely,

dh/dt) is estimated directly from finite differences. The difficulty comes in determining the gradient of the loss with respect to the weights of the NN, $\partial J/\partial\theta_3$.

One approach to the computation of $\partial J/\partial\theta_3$ is to use automatic differentiation to back-propagate through all of the steps of the time integration scheme used to solve Eq. 3.5. Another approach involves solving an adjoint problem backwards in time, which is the primary method used in [24], and indeed is the classical approach to optimization problems involving differential equation constraints [4]. A drawback of back-propagating through the solver is that to calculate the gradient, the entire state must be stored at each time step, which can lead to memory issues [24]. However, we consider a chaotic system which puts an implicit constraint on how far apart data can feasibly be sampled (roughly one Lyapunov time) and still yield a good estimate of g. In the present work, we reach this limit before the memory limit of back-propagation becomes an issue. In our trials, the adjoint method yielded results in good agreement with back-propagation, but required longer computation times for training, so we chose to use the back-propagation approach.

Note that we have separated the training problems for determining the manifold coordinates (i.e. the dimension reduction problem) from that for learning the dynamics in those coordinates. In principle, one could have put those problems together, minimizing a single loss that is a linear combination of Eq. 3.4 and Eq. 3.6. Indeed, Champion et al. [22] did something similar, though it must be noted that their approach requires data for time derivatives, while ours does not, and also that they knew in advance the dimensions of the manifolds where their data lives, which we do not. And in any case, the invariant manifold dimensions in their examples were three or fewer. There are two reasons why we treated the two problems separately. The first is conceptual. The question of finding the manifold on which a data set lies is a self-contained problem. It seems natural to address this problem first, and only afterward address the question of how the dynamics evolve on that manifold. The second reason is more practical. Even if one knows a priori the dimension of the manifold, solving simultaneously for the autoencoder and time evolution means simultaneously

solving for three neural networks with a very complicated loss landscape. Indeed, when we attempted to train both models together the predictive capabilities were extremely poor. Furthermore, in the problems we address, the manifold dimension is *not* known a priori, so trying to learn a vector field (RHS of an ODE) when one does not even have an idea of how many dimensions that vector field should have may lead to much-wasted computer time. Therefore, we take the view that it makes more sense to first get an estimate of how many dimensions are required to represent the data, and only then to estimate the vector field that determines the dynamics.

The data sets we consider are numerical solutions of the 1D Kuramoto-Sivashinsky equation (KSE),

$$\frac{\partial v}{\partial t} = -v\frac{\partial v}{\partial x} - \frac{\partial^2 v}{\partial x^2} - \frac{\partial^4 v}{\partial x^4}, \tag{3.7}$$

in a domain of length L with periodic boundary conditions. The domain size determines the types of dynamics this system exhibits. For the domain sizes we consider, trajectories exhibit sustained chaos. The dimension of the manifold that contains the global attractor has been computationally approximated using several approaches [43, 128, 237]. We generate high-dimensional state representations ($u \in \mathbb{R}^d$) via a Galerkin projection of v onto Fourier modes. Then, we use an exponential time differencing method [108] to integrate the resulting system of ordinary differential equations forward in time. The code used is available from Cvitanović et al. [41]. The data vectors u that we use are solutions v of the KSE on $d = 64$ equally-spaced grid points in the domain. We performed resolution checks on a grid of $d = 128$ and found, for the domain sizes we consider, trajectories from the same initial condition track one another for multiple Lyapunov times.

3.3 Results

Section 3.3.1 briefly describes the dimension reduction (manifold representation results). Section 3.3.2 considers neural ODE predictions, with and without dimension reduction, for

closely spaced data. Section 3.3.3 shows the effect of data spacing on the results, and Section 3.3.4 illustrates the effect of the degree of dimension reduction. We summarize in Section 3.4.

We consider datasets for three domain sizes, $L = 22$, 44, and 66, where we estimate the manifold dimension to be $d_{\mathcal{M}} = 8$, 18, and 28, as explained in the following section. The relevant timescale for each of these system is the Lyapunov timescale (inverse of the largest Lyapunov exponent), which we previously found to be be very close to the integral time for KSE data [128]. For these domain sizes, the Lyapunov times are $\tau_L = 22.2$, 12.3, and 11.6, respectively [43, 50]. We use 10^5 time units of data for training at each domain size, and vary how frequently we sample this dataset in time. For training the NNs we use Keras [31] for the autoencoders and PyTorch [161] for the neural ODEs. The neural ODE code is a modification on the code used in [24], which includes methods for computing the gradient both with the adjoint method and back-propagation.

3.3.1 Dimension reduction with autoencoders

In each section we use autoencoders to approximate the map to the manifold coordinates χ and back $\check{\chi}$. We found in [128] that a useful way to represent the map to the manifold coordinates ($h = \chi(u)$) is as a difference from a linear projection of the data onto the leading d_h PCA modes:

$$h = E(U^T u; \theta_1) + P_{d_h} U^T u. \tag{3.8}$$

Here E is a NN, U is the full PCA matrix, and P_{d_h} is the projection onto the leading d_h modes. Similarly, we can learn a difference for the decoder ($\tilde{u} = \check{\chi}(h)$)

$$\tilde{u} = UD(h; \theta_2) + U \begin{bmatrix} h \\ 0 \end{bmatrix}, \tag{3.9}$$

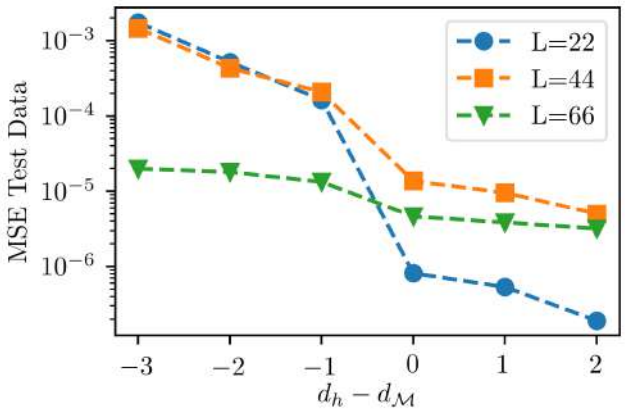


Figure 3.1: MSE of test data as a function of dimension for domain sizes $L = 22$, 44, and 66. The expected manifold dimensions are $d_{\mathcal{M}} = 8$, 18, and 28 for these domain sizes, respectively.

where D is a NN. By taking this approach we simplify the problem such that the NN only needs to correct upon PCA. We refer to this as a hybrid NN (HNN).

In [128] we also took advantage of the translational invariance of the problem. We used the first Fourier mode method-of-slices [19] to phase shift the data. In this method, the phase is calculated as $\phi(t) = \text{atan2}\{\text{Im}[a_1(t)], \text{Re}[a_1(t)]\}$, where a_1 is the projection of the state onto the first Fourier mode. Then, the state is phase shifted at each time using the Fourier shift theorem $\hat{u}_a(t) = a(t)e^{-ik\phi(t)}$. This approach removes the continuous symmetry, which reduces the dimension of the problem by one. More precisely, it decouples the evolution of the phase ϕ from the evolution of the pattern $\hat{u}_a$ – the phase evolution depends on $\hat{u}_a$ but not vice versa. This reduction in dimension improves the performance of the autoencoder but introduces near-discontinuities in the flow that arise when a_1 approaches zero that must be accounted for by warping time with "in-slice" time [19]. These near-discontinuities add additional difficulties in training the time evolution models, so we did not take this approach here.

In [128] we trained these HNN using an Adam optimizer with a learning rate of 0.001, for multiple dimensions d_h. Figure 3.1 shows the mean squared reconstruction error (MSE) of a test data set not used in training at various dimensions for each of the domain sizes. When using this hybrid approach we see a significant drop in the MSE at dimensions of $d_h = 8$, 18,

and 28 for the domain sizes $L = 22$, 44, and 66, respectively. We take these values of d_h to be the "correct" dimension, $d_{\mathcal{M}}$, in Sections 3.3.2 and 3.3.3. It is worth noting that finding this drop requires careful hyperparameter tuning. For example, we found using rectified linear units rather than sigmoids resulted in higher MSE for $d_h \gtrsim d_{\mathcal{M}}$ while having little effect on $d_h < d_{\mathcal{M}}$, making the drop in MSE less evident. Due to difficulties in autoencoder training, and due to the increased complexity of the system at larger domain sizes it can be difficult to determine the dimension from the drop in the autoencoder MSE alone. In Section 3.3.4, we further examine the issue of estimating manifold dimension by considering the performance of neural ODE models as a function of dimension – if the estimate of $d_{\mathcal{M}}$ is poor (especially on the low side) then the dynamic model is poor.

The estimated dimension at $L = 22$ of $d_{\mathcal{M}} = 8$ is in agreement with estimates by Ding et al. [43], Vlachas et al. [221], Yang and Radons [236] and Cvitanovic et al [39]. Ding et al. reached this conclusion by considering a finite number of unstable periodic orbits embedded in the long-time attractor and performing Floquet analysis on these solutions. Vlachas et al. also used an autoencoder approach of varying dimension and considering the MSE at different dimensions. In that work, they used different autoencoder architectures (e.g. convolutional autoencoder), with different activation functions (continuously differentiable exponential linear unit), and still found the same drop at $d_h = 8$. Yang and Radons [236] showed that the Lyapunov vectors can be split into two categories by looking for a drop in the Lyapunov exponents. The positive and weakly negative Lyapunov exponents give Lyapunov vectors called "physical modes" that were shown to span a linear space that locally approximates the inertial manifold [236]. This drop in Lyapunov exponents was found to be at $d_h = 8$ in [39] and at $d_h = 9$ in [236].

For larger domains, the dimension estimates reported here are consistent with Yang et al. [237]. In that work, they tracked the number of physical modes for larger domain sizes of the KSE. We showed in [128] that our estimates of $d_{\mathcal{M}}$ agree with the predicted dimensions and linear scaling of dimensions shown in Yang et al. [237]. Linear scaling is also seen in

Sondak and Protopapas [195], though their estimate of the manifold dimension is higher than ours or any of the other works mentioned here. They approximated dimension by training autoencoders with a high-dimensional latent space, and then considered the variance of each latent variable. Then $d_{\mathcal{M}}$ was taken to be the number of latent variables with a variance above some threshold.

These results guide our selection of dimension in the next two sections. Furthermore, we find that once $d_h \gtrsim d_{\mathcal{M}}$, the same MSE is achieved whether we use the full HNN or simply project on to $d_{\mathcal{M}}$ PCA modes (by setting $E(U^{\mathrm{T}}u) = 0$ in Eq. 3.8). This observation indicates, for the KSE with periodic boundary conditions, that the leading PCA amplitudes fully parameterize the state. Note that this parametrization is *nonlinear*, because a nonlinear decoder is still needed to reconstruct the full state. So, in the next two sections, χ is the projection onto $d_{\mathcal{M}}$ PCA modes and $\check{\chi}$ is a NN that approximates the remaining PCA coefficients from the leading ones. Table 3.1 shows the architectures for the autoencoders and for the NNs used for time evolution in Sections 3.3.2 and 3.3.3.

3.3.2 Effect of dimension reduction on time evolution

Now that we have a map to the manifold coordinates and back, the next step is approximating the ODE. Here we train three different types of ODEs to evaluate the impact of mapping the data to the manifold coordinates. We learn the RHS of the ODE in the manifold coordinates: $\dot{h} = g(h)$, in physical space: $\dot{u} = f(u)$, and in the space of the spatial Fourier coefficients: $\dot{\hat{u}} = \hat{f}(\hat{u})$, where $\hat{u} = \mathcal{F}(u)$ and $\mathcal{F}$ is the discrete Fourier transform. For the last two cases, there is no dimension reduction, so χ is the identity in the first case and the discrete Fourier transform in the second, with $\check{\chi}$ being, again, the identity in the first case and the discrete inverse Fourier transform in the second.

We train each of the NN with data spaced 0.25 time units in these trials. This timescale is substantially shorter than the Lyapunov time, which decouples the issue of dimension from that of time resolution. Each NN is trained until the loss levels off using an Adam

Table 3.1: Architectures of NNs and matrices used in Sections 3.3.1-3.3.3. "Shape" indicates the dimension of each layer, and "activation" the corresponding activation functions ("sig" is the sigmoid activation) [98].

	Function	Shape	Activation
Encoder $L = 22, 44, 66$	χ	$d/d_{\mathcal{M}}$	linear
Decoder $L = 22, 44$	$\check{\chi}$	$d_{\mathcal{M}}/500/d$	sig/linear
Decoder $L = 66$	$\check{\chi}$	$d_{\mathcal{M}}/500/500/d$	sig/sig/linear
ODE	f, g	$d, d_{\mathcal{M}}/200/200/200/d, d_{\mathcal{M}}$	sig/sig/sig/linear
Discrete Map	G	$d_{\mathcal{M}}/200/200/200/d_{\mathcal{M}}$	sig/sig/sig/linear

optimizer with a learning rate of 10^{-3} that we drop to 10^{-4} halfway through training. First, the autoencoder is trained, then the neural ODE. To avoid spurious results, we train 5 autoencoders then 10 ODEs and select the combination of autoencoder and ODE that provides the best short-time tracking. Due to random weight initialization and the stochastic nature of the optimization, we train multiple models and use the one that performs best. This is a common practice in neural network training, as noted by Bishop in Section 9.6 of [11]. As mentioned earlier, the models are trained to minimize the loss in Eq. 3.6. In previous work [128], we added an additional contribution to the loss for the time evolution problem that penalizes predicted states that fall outside the envelope of the joint PDF for energy production and dissipation. In the present work, we found excellent performance of the best models without the need for this additional term, because we did not phase-shift the data and thus did not have to work in "in-slice" time.

In Fig. 3.2 we compare the short- and long-time trajectories between the data at $L = 22$ (Fig. 3.2a) and the NN models (Fig. 3.2b-3.2d). For this domain size, the Lyapunov time is $\tau_L = 22.2$. The same initial condition, which is on the attractor, is used for all cases. The first model prediction, shown in Fig. 3.2b, comes from g (the low-dimensional ODE, with $d_h = 8$), the second model prediction, shown in Fig. 3.2c, comes from f (the full physical space ODE), and the third model prediction, shown in Fig. 3.2d, comes from $\hat{f}$ (the full Fourier space ODE). For each of these figures, the first 50 time units (about 2.5 Lyapunov times) are shown in the left column and 450-500 time units (i.e. after the solution has evolved

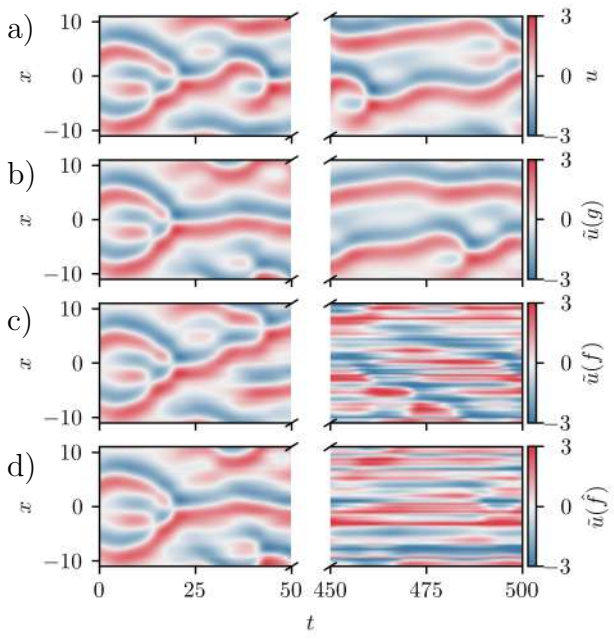


Figure 3.2: Examples of trajectory predictions for different models, $L = 22, \tau = 0.25$. The left shows predictions up to ~ 2.3 Lyapunov times, and the right shows predictions after ~ 23 Lyapunov times. (a) true trajectory, (b) predicted trajectory when approximating g with $d_{\mathcal{M}} = 8$. (c) predicted trajectory when approximating f. (d) predicted trajectory when approximating $\mathcal{F}(f)$.

for more than 20 Lyapunov times) are shown on the right.

All three of the models are able to generate good predictions for times up to about $t = 30$; this is a reasonable quantitative prediction horizon for a data-driven model of chaotic dynamics. After many Lyapunov times, a good model should still yield behavior consistent with the data. Indeed, at long times the reduced model no longer matches the true solution, but continues to exhibit realistic dynamics, as shown in Fig. 3.2b. In the next section, we discuss the long-time behavior of the reduced model in more detail. In contrast, the full state model predictions at long times, shown in Figs. 3.2c and 3.2d, develop high wavenumber striations , which eventually pollute the lower wavenumbers, and are thus not faithful to the dynamics. This same behavior appears when modeling the full state evolution for the larger domain sizes $L = 44$ and 66. Training high-dimensional neural ODEs is computationally more expensive, and the predictions from these models are worse.

To diagnose the origin of the spurious high wavenumber behavior, we plot the magnitude

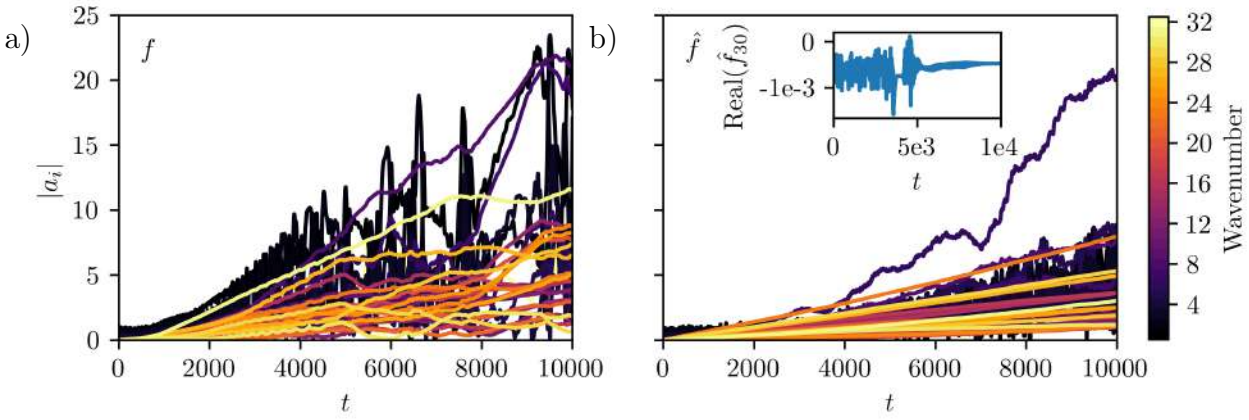


Figure 3.3: Magnitude of Fourier modes over time when evolving an initial condition with a neural ODE approximation of (a) the full-space dynamics in real space (f) and of (b) the full-space dynamics in Fourier space ($\hat{f}$), $L = 22, \tau = 0.25$. The inset in (b) is the evolution of the real part of $\hat{f}_{30}$.

of the Fourier modes as a function of time in Fig. 3.3 over a very long time interval – hundreds of Lyapunov times. In both cases, we observe a very slow linear growth in the high-wavenumber modes. The inset in Fig. 3.3b shows the time evolution of the RHS of the ODE for the real part of one of the high-wavenumber modes $\hat{f}_{30}$. At long-times $\hat{f}_{30}$ settles to a constant – effectively a bias term. The time-integration of this constant results in the linear growth seen in Fig. 3.3. We tested multiple activation functions in the NN architecture, and we see growth in the high wavenumbers regardless of whether sigmoids, hyperbolic tangents, or rectified linear units are used.

To further explain the origin of the bias that leads to the linear growth, we first note that in the true system the high wavenumbers are strongly damped due to hyperdiffusivity in the KSE, so data sampled from the attractor has negligible content in these wavenumbers. In particular, with the data given, the model has no information with which to learn that these modes are strongly damped. Furthermore, the prediction horizon in the training of the neural ODE is short (0.25 here), so small errors in high wavenumbers have a negligible effect on the loss. Thus the model does not learn that there should be a strong damping term that would overwhelm any small bias that arises in the RHS due to the stochastic nature of the initialization and training of neural net representations. Indeed, experiments that explicitly

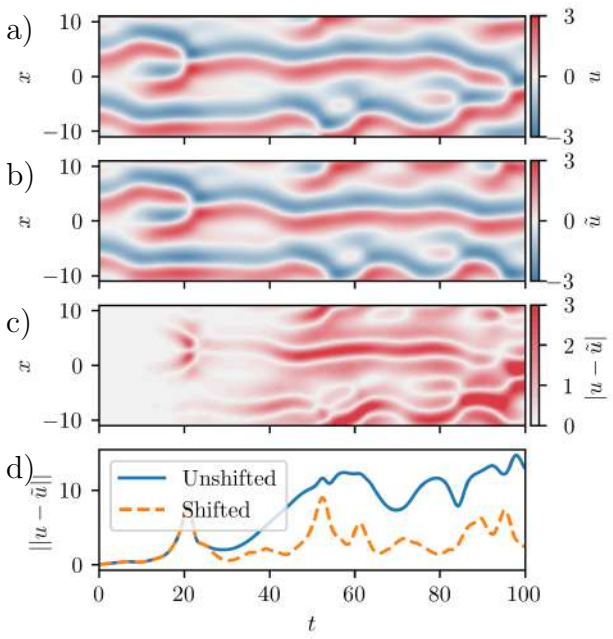


Figure 3.4: Example of trajectory and predictions for data spaced $\tau = 10$ apart, $L = 22$: (a) true trajectory; (b) predicted trajectory; (c) absolute error between the two trajectories; (d) norm of the error. The norm of the error is shown for the raw data, and for the spatial shift that minimizes this error at each time.

introduce a damping term, modifying Eq. 3.3 to read $dh/dt = g(h;\theta_3) - ah$ where $a > 0$, is sufficient to stabilize models against this slow linear growth. In [132] we show this is the case when the damping is chosen to match the linear term in the KSE. The combination of these factors is why the trajectories of the high-dimensional models leave the attractor, while the trajectories of the reduced-dimensional models, where degrees of freedom that are strongly damped and thus have negligible amplitude on the attractor are absent, do not.

3.3.3 Effect of temporal data spacing on time evolution prediction

Now we consider only reduced-dimension ODE models, and address the dependence of model quality on the temporal spacing τ between data points. For $L = 22$, 44, and 66, we select the manifold dimensions mentioned in Section 3.3.1 – $d_{\mathcal{M}} = 8$, 18, and 28, respectively. We judge the model performance based on, first, short-time tracking for $L = 22$ and then long-time statistical agreement for all domain sizes. In these cases autoencoders and ODEs were both trained with τ much larger than the value of 0.25 used in Section 3.3.2, while keeping all other

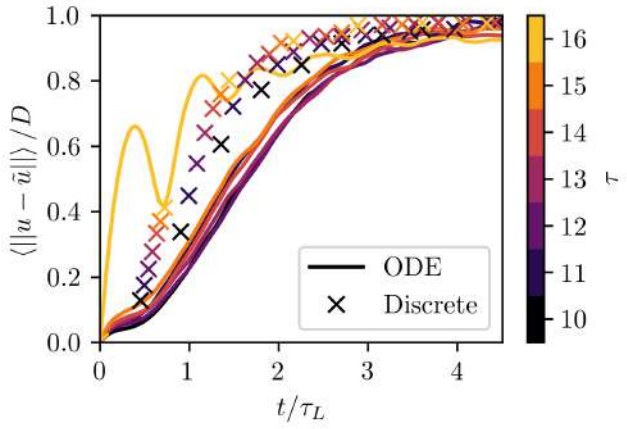


Figure 3.5: Normalized difference between trajectory and prediction as a function of time for different training data spacing, $L = 22, d_h = 8$.

training parameters the same (e.g. architecture, optimizer, dimension). Figure 3.4 compares a true short-time trajectory (3.4a) to the model prediction (with $\tau = 10$) of this trajectory (3.4b) starting from the same initial condition. Qualitatively the space-time plots exhibit similar behavior over around 80 time units. The magnitude of the difference is shown in 3.4c. Here we see the difference growing at around 40 time units due primarily to a growing spatial phase difference between the true and predicted results. Figure 3.4d shows the norm of this difference and the minimum value of the norm when shifting $\tilde{u}$ to any position in the domain. The minimum translated difference, in 3.4d, remains much smaller than the standard Euclidean distance indicating that, in this case, the phase difference causes most of the error.

We now consider ensemble average quantities to better understand the reconstruction quality of the models as a function of τ. In addition to the ODE models, we also consider discrete-time maps $h(t_i + \tau) = G(h(t_i))$. Details of the discrete-time map NN architecture is given in Table 3.1.

Figure 3.5 shows the root mean squared difference between the exact trajectory and ODE/discrete-time map models as a function of time, averaged over 1000 initial conditions and normalized by the root mean squared difference between random training data, which is denoted with D. In this figure, the ODE models and discrete time maps use data spaced

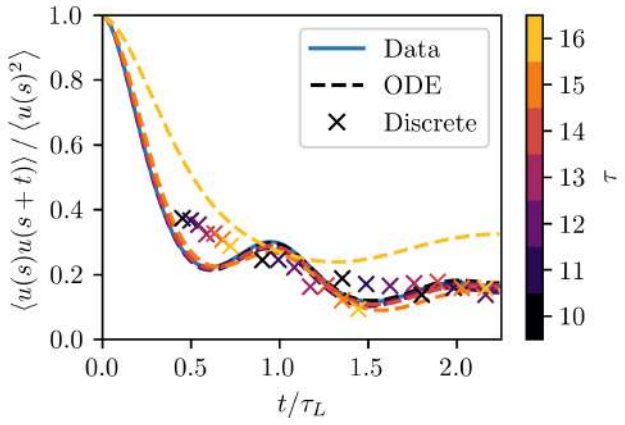


Figure 3.6: Temporal autocorrelation for models learned with different data spacing, $L = 22, d_h = 8$.

$\tau = 10 - 16$. For $\tau < 10$, there is little improvement in the ODE models' performance. ODE predictions at and below $\tau = 15$ all track well for short times, with the error being halfway to random at around $t \sim 1.5\tau_L$, and diverge at long times, with the error leveling off at around $t \sim 3\tau_L$. Then, performance degrades sharply at $\tau = 16$ (yellow curve). In all cases, the discrete-time map performs worse than the ODE with the same data spacing. Furthermore, although we do not show it here, the performance for discrete-time maps becomes even worse when trying to interpolate between prediction times. This is a significant drawback not seen when using ODEs.

A similar trend appears in the temporal autocorrelation. Figure 3.6 shows the temporal autocorrelation of u at a given gridpoint averaged over space and 1000 initial conditions. The temporal autocorrelation of the neural ODE matches the exact temporal autocorrelation well for data spaced up to $\tau = 15$. Then, there is an abrupt deviation from the true solution at $\tau = 16$. Also, in Fig. 3.6 we show the temporal autocorrelation for discrete-time maps. At $\tau = 10$ discrete-time maps perform worse than all ODEs below $\tau = 15$, and predictions worsen when increasing τ.

The other half of evaluating a model is determining if trajectories stay on the attractor at long times. For this purpose, we consider the long-time joint PDF of the first (u_x) and second (u_{xx}) spatial derivatives of the solution. In the KSE, $\overline{u_x^2}$ is the energy production and

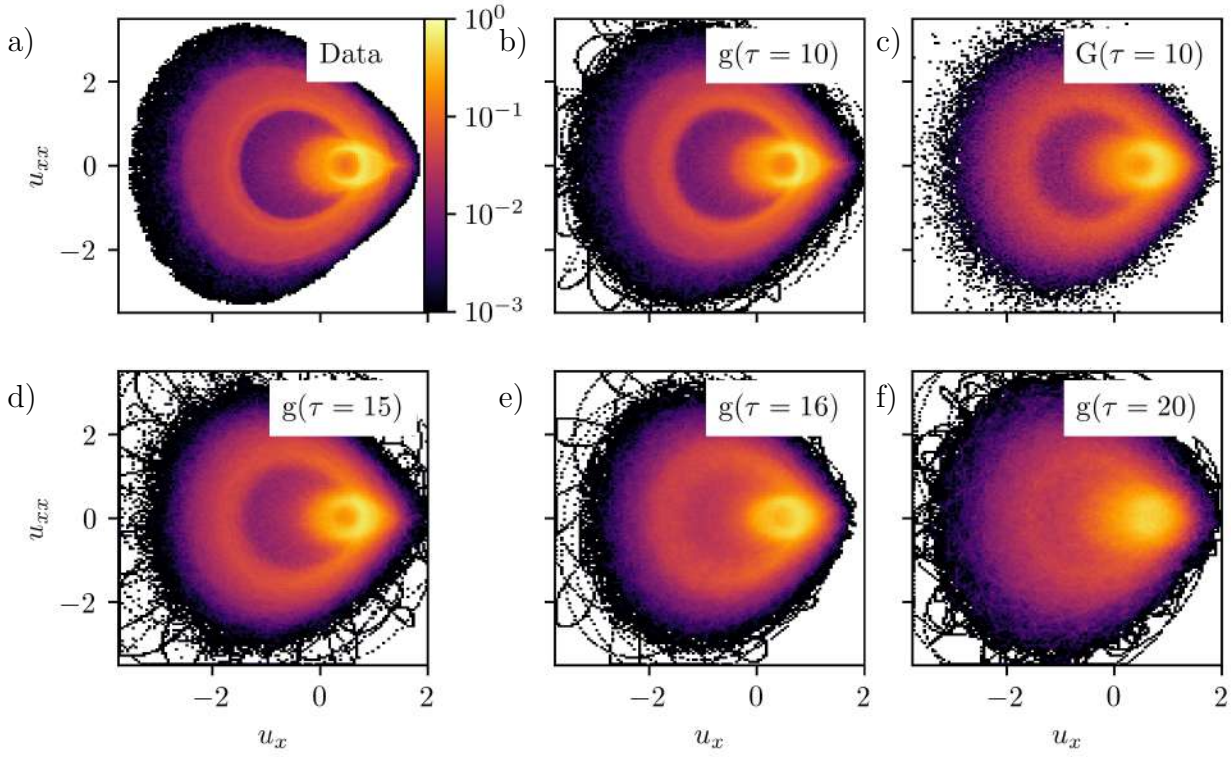


Figure 3.7: (a) joint PDF at $L = 22, d_h = 8$. (b) and (c) joint PDFs when approximating the ODE and the discrete-time map, respectively, at $\tau = 10$. (d)-(f) joint PDFs when approximating the ODE for $\tau = 15$, 16, and 20.

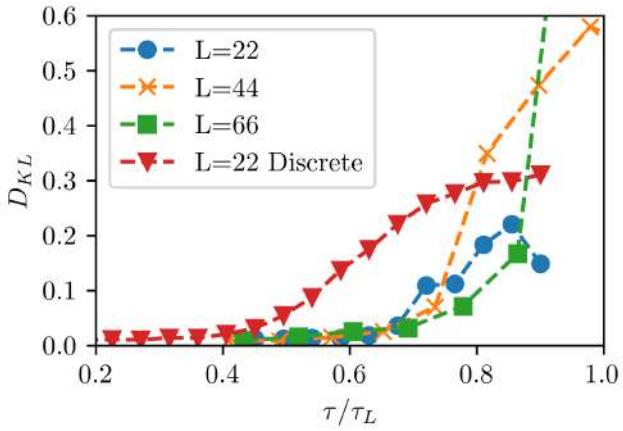


Figure 3.8: KL divergence of the models trained with different data spacing. Here time is scaled with Lyapunov time for the corresponding domain size. For $L = 22, 44, 66$ we use $d_h = 8, 18, 28$.

$\overline{u_{xx}^2}$ is the dissipation [39], where $\bar{\cdot}$ is the spatial average. We select these quantities because of their relevance to the energy balance, and because two-dimensional joint PDFs are more challenging and important to reconstruct than one-dimensional PDFs. In Fig. 3.7 the joint PDF for the data is compared to various models. The colormap is a log scale, with low probability excursions in black, and no data in white regions. When $\tau = 10$, the ODE model matches well, with differences primarily in the low probability excursions, while at $\tau = 10$ the discrete-time mapping appears more diffuse in the high probability region and matches poorly. At $\tau = 15$, the joint PDF for the ODE prediction still matches well, degrading once $\tau \geq 16$. This deterioration becomes more evident at $\tau = 20$.

We quantify the difference between true and predicted PDFs by computing the KL divergence

$$D_{KL}(\tilde{P}||P) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \tilde{P}(u_x, u_{xx}) \ln \frac{\tilde{P}(u_x, u_{xx})}{P(u_x, u_{xx})} du_x du_{xx} \tag{3.10}$$

between the true joint PDF, P, and the predicted PDF, $\tilde{P}$. We take the convention of setting the quantity under the integral to 0 when $P = 0$ or $\tilde{P} = 0$ as was done in [21]. The KL divergence appears in Fig. 3.8 for various τ, where we normalize by the Lyapunov time τ_L. For all domain sizes, the joint PDFs for neural ODE models match the true PDF very well when data is spaced below $\sim$0.7 Lyapunov times. In contrast, good reconstruction of the

Table 3.2: Architectures of NNs used in Section 3.3.4. Labels are the same as in Table 3.1.

	Function	Shape	Activation
Encoder $L = 22$	χ	$d/500/d_h$	sig/linear
Encoder $L = 44, 66$	χ	$d/500/500/d_h$	sig/sig/linear
Decoder $L = 22$	$\check{\chi}$	$d_h/500/d$	sig/linear
Decoder $L = 44, 66$	$\check{\chi}$	$d_h/500/500/d$	sig/sig/linear
ODE	g	$d_h/200/200/200/d_h$	sig/sig/sig/linear

joint PDF with the discrete-time model, for $L = 22$, requires $\tau/\tau_L \lesssim 0.4$.

3.3.4 Effect of model dimension on time evolution predictions

In the previous sections, the model dimension was fixed at values assumed to be the correct inertial manifold dimensions based on other studies [43, 128, 237]. Here we vary the dimension and consider the dynamical model performance, which provides more information on the necessary number of dimension. For training we again use 10^5 time units of data spaced apart 0.25 time units and train 5 autoencoders and 15 neural ODEs. The data is normalized by subtracting the mean and dividing by the standard deviation before training and projected onto the full set of PCA modes before going into the autoencoder ($U^T u$). Training was done with an Adam optimizer with learning rate scheduling from 10^{-3} to 10^{-4} halfway through training. For $L = 22$ we trained for 5000 epochs and for $L = 44, 66$ we trained for 500 epochs, due to the computational cost of the larger networks. In all these trials, the loss plateaued at the end of training, but further training in the $L = 44, 66$ cases could lead to mild improvement. The code used for training these autoencoders is available on GitHub [130].

Our first result here is the observation that, for models with a lower dimension than the "correct" one, reasonable approximations of the joint PDF of u_x and u_{xx} can be obtained, but only if a nonlinear encoder is used for dimension reduction. This point is illustrated in Figure 3.9a, for the case $L = 22$, where we see good reconstruction of the joint PDF when using a

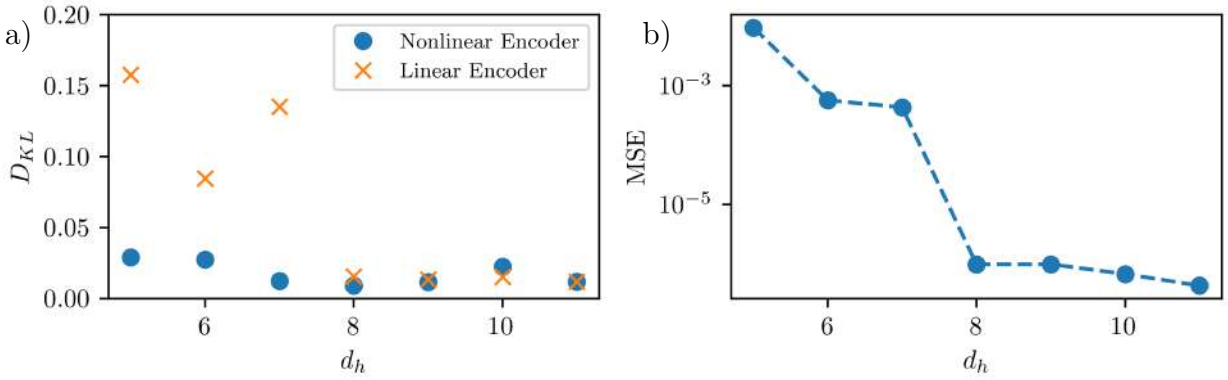


Figure 3.9: (a) KL divergence of the best autoencoder and neural ODE pair at multiple dimensions for $L = 22$. The nonlinear encoder corresponds to an autoencoder with a NN for encoding, and the linear encoder corresponds to an autoencoder with the projection onto the first d_h Fourier modes for encoding. Both cases use a NN for decoding. (b) mean squared error of reconstruction for the nonlinear autoencoder at each dimension.

nonlinear encoder. Accordingly, the results below all use a nonlinear encoder; architectures are reported in Table 3.2. However, even though this statistic is reconstructed well at low dimensions, Fig. 3.9b shows the MSE for the autoencoder, with a nonlinear encoder, still drops significantly at $d_h = 8$. This MSE is on test data normalized using the training data mean and standard deviation. As with the autoencoders used in Sections 3.3.2 and 3.3.3, we again performed manual hyperparameter tuning. Changing the activation functions from sigmoids to rectified linear units and removing the learning rate scheduler both resulted in higher MSE for $d_h \gtrsim d_{\mathcal{M}}$.

Figure 3.10 shows the ensemble-averaged short-time tracking error of the models with the best tracking as the dimension varies. For all domain sizes the recreation gradually improves and then becomes dimension-independent as dimension increases. This happens because the short-time tracking is directly related to the loss minimized in training the neural ODEs, and h contains the same information as u if d_h is large enough. As mentioned before, for $L = 22$, 44, and 66, respectively, the "correct" manifold dimensions are 8, 18, and 28, and Fig. 3.10 shows the tracking error becoming dimension-independent near these values.

Now we turn to long-time statistical recreation upon varying dimension. Figure 3.11 shows the KL divergence from the true joint PDF $P(u_x, u_{xx})$ for all the autoencoder and

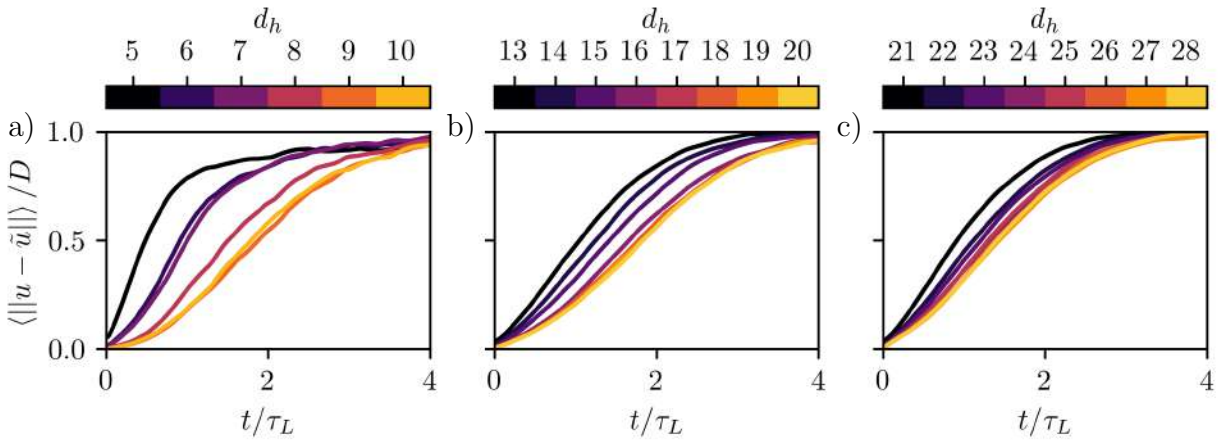


Figure 3.10: Short-time error at various dimensions. Domain sizes are $L = 22, 44, 66$ for (a)-(c).

neural ODE pairs trained for each dimension and domain size. For reference, the dashed lines show D_{KL} when comparing the true joint PDF, used to compute D_{KL} for the models, to a separate joint PDF from the true system, and evolved forward for the same amount of time as the models but from a different initial condition. This gives a baseline for how two empirical joint PDFs from the true system compare.

In all the cases, when the dimension is low, the models do a poor job of reconstructing the joint PDF. When we are near the expected manifold dimension i.e. $d_h \approx 8$, 18, and 28, for $L = 22, 44$, and 66, respectively, the typical model performance dramatically improves and the variation in performance between the models decreases, with $D_{KL} \approx 10^{-2}$ for the best-performing models and only a factor of two or three larger for the worst. Furthermore, D_{KL} between true and best-model predictions is only slightly larger than that between the two time intervals for the true system, with a particularly close agreement for $L = 22$ and 44. In Fig. 3.12, we plot the true PDFs and the best model PDFs at $L = 22, 44, 66$ and $d_h = 8, 18, 28$, illustrating the excellent agreement. However, with further increase of dimension, the variation in performance again becomes large, although the best models still perform well. Like the short-time prediction, the joint PDFs of the best models become dimension-independent, but here we see that errors arise, particularly the slow linear growth

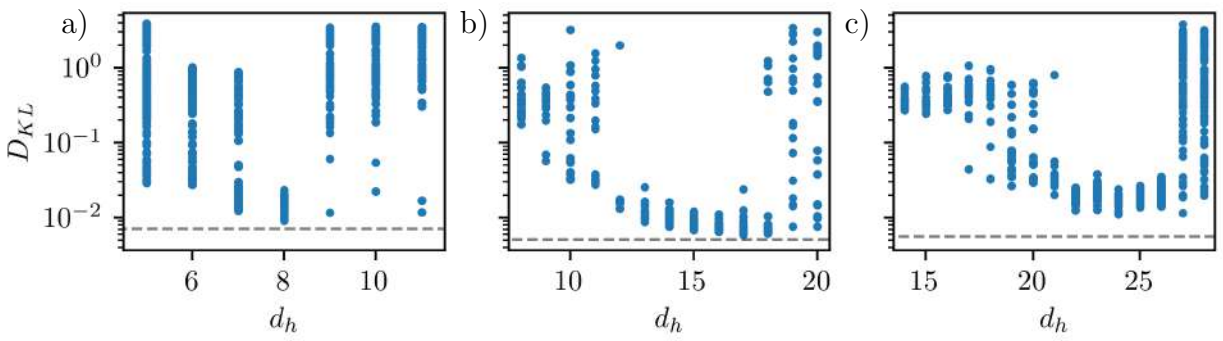


Figure 3.11: KL divergence for all trained models at different dimensions. Domain sizes are $L = 22, 44, 66$ for (a)-(c). The dotted lines on each plot correspond to the D_{KL} computed by comparing two joint PDFs of the true system.

in high wavenumbers, when training the neural ODE with unnecessary dimensions. Finally, recall that the error at high d_h arises from spurious slow linear growth at high wavenumbers, which, as we saw in Section 3.3.2, arises in the absence of any autoencoder at all.

In summary, Figure 3.11 highlights that there is a "sweet spot", $d_h \approx d_{\mathcal{M}}$, where we see robust model performance. Below this, too few dimensions are retained so model performance is poor, and above it, spurious slow growth of high wavenumbers pollutes the long-time predictions. So lack of robustness in model performance is an indication that the dimension has been chosen incorrectly – either too low or too high. As noted previously, we also find that stabilizing long-time dynamics is possible with the addition of a damping term.

3.4 Summary

Neural ODEs and undercomplete autoencoders can be used to create accurate data-driven models for time evolution of spatiotemporal chaotic dynamical systems, with dimensions near the expected manifold dimension, which for the present examples is in the approximate range $8 - 28$. Dimension reduction is a vital step in the process. Both the training process and the prediction of trajectories are more expensive with more dimensions, and additional dimensions lead to computational difficulties with the introduction of small spurious bias that leads to slow linear growth of high-wavenumber modes. With dimension reduction, we find that

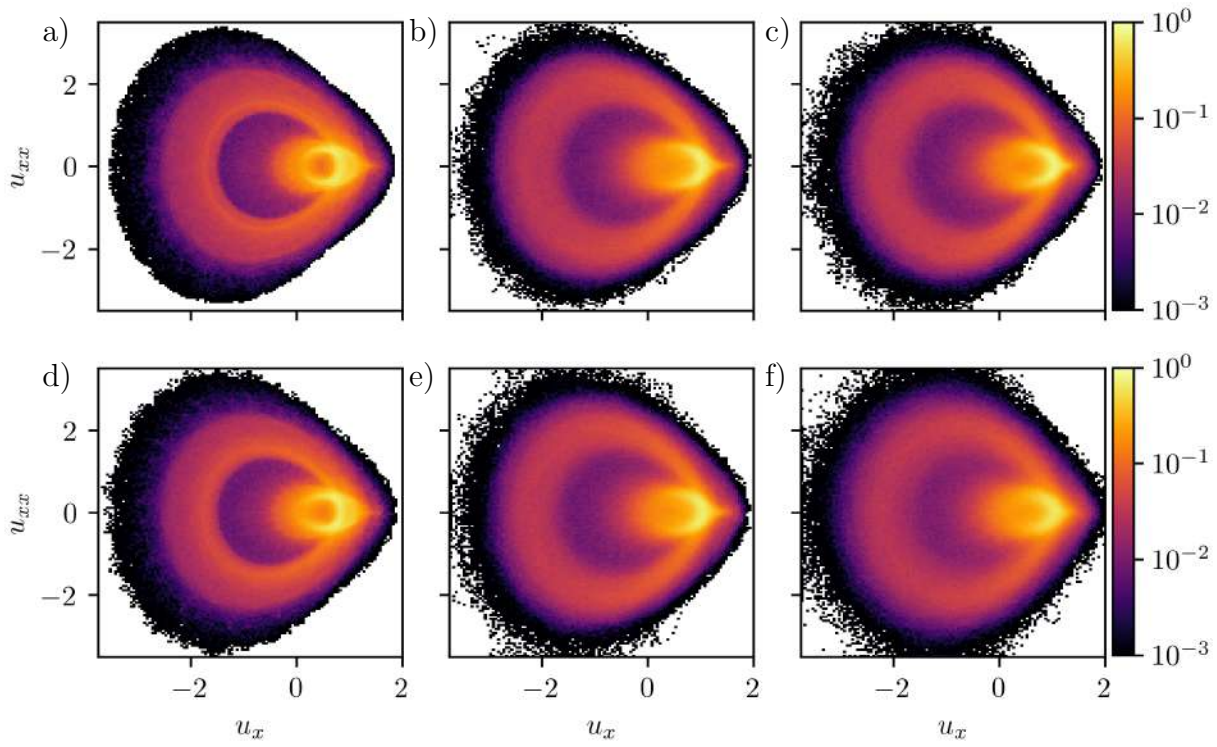


Figure 3.12: (a)-(c) True joint PDFs for $L = 22, 44, 66$. (d)-(f) joint PDFs from the best models (lowest D_{KL}) for $L = 22, 44, 66$ at $d_h = 8, 18, 28$.

training with data spaced up to ~ 0.7 Lyapunov times gives accurate short-time tracking and long-time statistical reconstruction. Finally, we investigate the impact of varying the dimension and find that model performance improves with dimension, and stops improving once the manifold dimension is reached. However, because of the spurious excitation of high wavenumbers, keeping too many dimensions hurts training, resulting in many poor models. While we have used the Kuramoto-Sivashinsky equation as a model system, this computational approach using data mediated neural ODEs might also be useful in the modeling of PDE systems like atmospheric phenomena [135], in the modeling of infinite-dimensional discrete-time maps [153], or in the modeling of other complex spatially distributed systems [111]. In forthcoming work we will show how this reduced order modeling approach can be integrated into a reinforcement learning framework for control of spatiotemporally chaotic dynamics.

Chapter 4

Reinforcement learning with model-based control[1]

In this chapter, we modify the model we used in Ch. 3 to model the KSE with control and apply deep reinforcement learning (RL) to the model. RL is a data-driven method capable of discovering complex control strategies for high-dimensional systems, making it promising for flow control applications. In particular, here the goal of the RL is to reduce energy dissipation in turbulent flows, and the example considered is the spatiotemporally chaotic dynamics of the KSE. A major challenge associated with RL is that substantial training data must be generated by repeatedly interacting with the target system, making it costly when the system is computationally or experimentally expensive. We mitigate this challenge in a data-driven manner by combining dimensionality reduction via an autoencoder with a neural ODE framework to obtain a low-dimensional dynamical model from just a limited data set. We substitute this data-driven reduced-order model (ROM) in place of the true system during RL training to efficiently estimate the optimal policy, which can then be deployed on the true system. For the KSE actuated with localized forcing ("jets") at four locations, we demonstrate that we are able to learn a ROM that accurately captures the

[1]The text of this chapter is adapted from the publication by K. Zeng, A. J. Linot and M. D. Graham *Proc. Royal Soc. A.*, 478, 2022

actuated dynamics as well as the underlying natural dynamics just from snapshots of the KSE experiencing random actuations. Using this ROM and a control objective of minimizing dissipation and power cost, we extract a control policy from it using deep RL. We show that the ROM-based control strategy translates well to the true KSE and highlight that the RL agent discovers and stabilizes an underlying forced equilibrium solution of the KSE system. We show that this forced equilibrium captured in the ROM and discovered through RL is related to an existing known equilibrium solution of the natural KSE.

4.1 Introduction

In recent years, deep reinforcement learning (RL), a data-driven model-free control method, has achieved recognition for its ability to learn and discover complex control strategies for high-dimensional systems. Many of its milestone achievements involve defeating the best professional players in complex games such as GO [189], DOTA II [155], and Starcraft II [215], as well as the leading GO, Chess, and Shogi engines [190]. The successes of RL in these complex and nontrivial systems has made its application towards control of high-dimensional chaotic dynamical systems, such as turbulent flows, extremely promising. Deep RL has recently been demonstrated to be able to discover strategies in flow systems that exhibit nonchaotic dynamics such as reducing the drag in flow over bluff bodies [54, 124, 159, 171, 176] and decreasing convection amplitude in Rayleigh-Benard systems [8]. Using the Kuramoto-Sivashinsky equation (KSE) as an example, Bucci et al. [16] demonstrated that RL is able to direct a system with natural chaotic dynamics to a given (i.e. previously known) equilibrium solution of the system. In another study of the KSE in a chaotic parameter regime, the present authors [242] demonstrated that an RL policy whose aim is to minimize dissipation is able to discover and stabilize a nontrivial underlying equilibrium solution with low dissipation, even when no such information of its existence is given. The aim of the present work is to combine RL with data-driven reduced-order dynamical models for the

purpose of control of spatiotemporal dynamics, as a step toward the goal of reducing energy dissipation in turbulent flows.

Despite all of the successes of deep RL, a major challenge is that it suffers from poor sampling complexity. For example, recent work by Du et al. [46] showed that even if the optimal control policy could be perfectly represented by a linear function, the reinforcement learning agent still requires an exponential number of trajectories to find a near-optimal policy. Furthermore, the overall practical problem is further exacerbated by the temporal credit assignment problem, which is the difficulty in parsing out what actions in a long series of actions actually contributed to the end result–often bottlenecking RL algorithms attempting to train large models with millions of weights in application [81].

In practice, this issue is often addressed by simply brute-force generating an extraordinary amount of interaction data from the target system for training. For example, to achieve the previously mentioned accolades, over 11,000+ years of DOTA II gameplay [155], 4.9 million self-play games of GO [189], and 200 years of Starcraft II gameplay [215] was required. In the realm of fluid control, even learning a two-parameter control scheme for reducing drag in an LES bluff-body simulation required over 3 weeks of training and simulation time [54]. The default solution of simply generating more data can quickly make deep RL a prohibitively costly solution when the target flow system is computationally (e.g. via direct numerical simulations) or experimentally (e.g. via wind tunnel tests) expensive to realize.

Despite these challenges, there are growing efforts toward developing more data-efficient learning algorithms and work-around techniques. These efforts can be broadly categorized into three groups: modified RL algorithms with added complexity, true off-policy RL, and model-based RL. The first group, modified RL algorithms, is a catch-all group comprising of methods that build upon existing established algorithms with advanced architectures and formulations and/or incorporate domain knowledge or hierarchy to improve training stability and data efficiency at the cost of greater implementation and training complexity. For example, [242] reduced the data-sampling complexity of RL by exploiting the natural

symmetries of the system and [148] found complex behavior tasks could be learned efficiently via a hierarchical RL framework. A comprehensive survey of RL combined with attention mechanisms, transfer learning, and hierarchy can be found in [125]. The second group, true off-policy or Offline RL, is comprised of methods that reformulate or modify the typical semisupervised RL problem to extract control strategies from a pre-existing limited data set of transitions [1, 123]. However, these methods are still in the early stages of development and it is still unclear how well they perform and generalize in our complex systems of interest.

The last group, model-based RL, are methods that utilize a model of the target environment. These methods are advantageous in that they can leverage the existing vast classical modeling principles as well as the newly emerging deep learning techniques. As a result, model construction, implementation, and degree of data-drivenness vary widely from method to method. For example, Nagabandi et al. [149] introduced the Model-Based+Model-Free method, which utilizes a system model and an MPC controller to initialize the RL agent's weights. Towards more data-driven approaches, Wahlstrom et al. [225] introduced Deep Dynamical Models, which uses an autoencoder to learn a reduced-order representation of the observation data and a feedforward neural network to learn a discrete time stepping model in said reduced-order representation to control an inverted pendulum from pixel inputs. Similarly, Watter et al. [231] also utilized autoencoders and feedforward neural networks to learn a local linear dynamical representation of the pendulum problem. Feinberg et al. [55] utilized models to generate short "imagined" trajectories at each training step to better estimate the value functions. This so-called "Dyna"-style of model-based RL [202] has been recently applied in control of spatiotemporally chaotic systems by Liu et al. [134] who demonstrated with autoencoded recurrent network models containing hardcoded boundary conditions and soft conservation constraints that the number of real interactions needed with the target system to achieve control could be greatly reduced. Alternatively, Chua et al. [32] demonstrated another form of model-based RL wherein a data-driven dynamics model is iteratively learned online with model predictive control (MPC) serving as the controller. The

authors demonstrated that this method was able to reach similar asymptotic performance to state-of-the-art model-free RL algorithms on benchmark tasks while using much fewer data samples, although at the expense of requiring real-time model trajectory optimization at every step during implementation. Finally, Ha and Schmidhuber [81] introduced recurrent world models, which model the control system of interest with mixture density networks and recurrent neural networks. In this method, the authors use a data-driven model in place of the real environment during RL training to estimate a control policy for driving a 2D car and playing DOOM. We will refer to this style of utilizing a data-driven model as a surrogate training ground as "model-based RL".

The learning process of the above-defined model-based RL parallels the classic control design philosophy [142] which is: 1) create a model for control, 2) design a control policy based on the model, and 3) simulate based on a high-fidelity model and repeat if needed. In parallel to these principles, we seek to: 1) *learn* a model – specifically a reduced-order, but still highly accurate model – for control from data, 2) *learn* a control policy based on the model with RL, and 3) validate based on a high-fidelity model.

A key aspect of this approach is the development of an efficient reduced-order model of the process, which we will consider to be completely governed by a state space representation

$$\frac{ds}{dt} = f(s, a), \tag{4.1}$$

or its discrete-time version

$$s_{t+\tau} = F(s_t, a_t), \tag{4.2}$$

where $s \in \mathbb{R}^d$ is the state of the system and $a \in \mathbb{R}^{d_a}$ a time-dependent actuation. Our focus will be the continuous-time case Eq. 4.1, where we will seek a reduced-order representation

$$\frac{dh}{dt} = f(h, a), \tag{4.3}$$

where $h \in \mathbb{R}^{d_h}$ and $d_h \ll d$. Before describing our specific approach, we briefly review other methods for developing dynamic models from data. A more detailed discussion is given in [129].

If the governing equations of the process are known, one approach is simply to use a direct simulation without any model reduction. Here of course none of the efficiency gains enabled by reducing the dimension of the model are realized. A traditional approach to reducing the dimension of these problems is the Galerkin projection method [93] and other variants like nonlinear Galerkin [77, 103, 138] or post-processing Galerkin [63], which have higher accuracy than the traditional Galerkin approach for given number of unknowns. Additionally, such approaches can be improved by adding data-driven information. For example, Wan et al. [228] approximated the unresolved dynamics in nonlinear Galerkin with an LSTM. The concept of combining first-principles and data-driven models is a widely studied one (see e.g. "physics-informed neural networks" [173]).

In the present work, however, our focus is data-driven models. The classical approach is to seek a linear discrete-time model; a specific example of this approach, which has seen wide use recently even for nonlinear systems, is dynamic mode decomposition (DMD)[117]. We describe and implement an adaption of DMD to closed-loop control, known as DMDc, in Section IIIB. As a linear state space model, at long times, trajectories predicted by DMD always evolve toward a fixed point or a quasiperiodic orbit with a discrete set of frequencies, so it cannot faithfully characterize chaotic dynamics, which is our interest here. In principle, for all nonlinear systems there does exist a linear, albeit infinite-dimensional, operator called the *Koopman* operator that describes the evolution of arbitrary observables [20, 118]. We choose not to take this approach because our aim is to reduce the dimension of our model. In any case, effective methods for approximation of chaotic long-time dynamics with data-driven Koopman operator approximations remain elusive.

With the increasing power of neural network representations, nonlinear data-driven modeling methods have received substantial recent attention. Two popular methods include re-

current NN (RNN) [91, 96] and reservoir computing [102, 122]. These approaches are similar in that a hidden or reservoir state r_t is used along with the current state s_t to predict a future hidden or reservoir state $r_{t+\tau}$. Then a function is used to predict the new state $s_{t+\tau}$ from the hidden state. The two approaches are different in how these functions are approximated. In reservoir computing the functions for evolving the reservoir are chosen a priori, and the mapping back to $s_{t+\tau}$ is learned. In RNNs, specific neural networks are trained to learn all of the functions – two examples are LSTMs [91] and gated recurrent units [33]. Vlachas et al. [220] shows that both these methods perform well at predicting chaotic dynamics. However, because they involve hidden variables, these approaches do not reduce the dimension of the system, but rather increase it: i.e. they do not determine $s_{t+\tau}$ from s_t alone, but rather also use past history stored in r_t to make predictions.

A more direct approach to learning dynamical models is to learn the right-hand side (RHS) of Eq. 4.1 or Eq. 4.2 from data. One way to do this, at least for low-dimensional systems, is using the "Sparse Identification of Nonlinear Dynamics " (SINDy) approach [15]. In this method a dictionary of candidate nonlinear functions are selected to approximate f in Eq. 4.1. Then, by using sparse regression, the dominant terms are identified and kept. Alternately, one could simply represent f with a neural network [73]. If one does not have time-derivative data or estimates, then the neural ODE approach [24, 129] can be used.

Now we turn to the issue of dimension reduction. The dynamics of formally infinite-dimensional partial differential equations describing dissipative flow systems are known to collapse onto a finite-dimensional invariant manifold [204]–a so-called "inertial manifold". In past works, it was shown that one can capture a high-dimensional system's dynamics on this lower-dimensional manifold by using a combination of autoencoders, to identify the manifold coordinates, and Neural ODEs (NODE) [24], to model the dynamics on the manifold in discrete time [128] or continuous time [129]. Our aim is to adapt this data-driven, low-dimensional model framework as a data-efficient surrogate training grounds for model-based RL. Once training is complete, this ROM-based policy is then deployed to the

real system for assessment or further fine-tuning. We refer to our method as *Data-driven Manifold Dynamics for Reinforcement Learning*, or DManD-RL for short.

Unlike the previous implementations of data-driven models in model-based RL [81, 225, 231], which model the evolution of the controlled system in fixed discrete time steps, our method models the vector field of the dynamics on the manifold in the presence of control inputs using neural ODEs [24]. The choice of using neural ODEs has several benefits: 1) it allows one to utilize the vast and already present array of integration schemes for time-stepping, 2) it allows one to train the model from data collected from variable time step sizes as well as make variable time step predictions, which grants one the ability to adjust the transition time step in RL without resorting to recollecting new data and training a new dynamics model, 3) it is a natural formulation for the modeling of the dynamics of our physical flow systems. We highlight that this framework is not limited to fixed time intervals, can utilize unevenly and/or widely spaced data in time, respects the Markovian nature of the dynamics of the systems we are interested in capturing, and respects the Markov environment assumption that underlies RL theory [202], unlike other common data-driven approaches such as recurrent neural networks [91, 96] and reservoir computing [102, 122]. Finally, we do not impose that the underlying relationship between the control input and the dynamics is affine in control (i.e. that $ds/dt = f(s) + g(s)a$ where s is the state and a is the control input), as some past works have done [231]. This is advantageous for application towards complex flow control systems of practical interest as many, such as aircraft [13] and underwater vehicles [64], do not have relationships with this affine structure.

We highlight here an important interplay regarding the RL problem and our dissipative systems of interest. In RL the general algorithm seeks to learn a strategy by exploring and optimizing the state-action space. At first glance, this may seem hopeless for flow systems exhibiting high-dimensional ambient spaces. However, an important distinction here is that although our formally infinite-dimensional flow systems may require $\mathcal{O}(10^{6+})$ degrees of freedom to resolve in the ambient space, the underlying dynamics can be much lower

dimensional. This indicates that the underlying complexity of the problem is independent of mesh resolution (assuming the system is already resolved). Assuming we can find a high-fidelity reduced-order representation of the data, the state-action space that we seek to optimize and capture in our method is actually only of complexity *manifold dimension* $\times$ *action dimension*, which leaves an optimistic outlook for our method toward application to larger and more complex flow systems.

The remainder of this paper is structured as follows: In Section 4.2 we introduce the Kuramoto-Sivashinsky equation, a proxy system for turbulent flows that displays rich spatiotemporal chaos, and the control objective which serves as a drag reduction analog. We conclude this section with an outline of our ROM-based RL framework. In Section 4.3 we examine the performance of our learned reduced-order model, the control strategy extracted from the model using RL, and the dynamical systems relevancy of the strategy. In Section 4.4 we summarize our findings and discuss applications to more complex systems.

4.2 Formulation

4.2.1 Kuramoto-Sivashinsky equation

The Kuramoto Sivashinsky Equation (KSE) is given by,

$$\frac{\partial v}{\partial t} = -v\frac{\partial v}{\partial x} - \frac{\partial^2 v}{\partial x^2} - \frac{\partial^4 v}{\partial x^4} + \mathcal{F}(x,t). \tag{4.4}$$

Here f is a spatiotemporal forcing term that will be used for control actuation. We consider the KSE in a domain of length $L = 22$ with periodic boundary conditions as this is a dynamically well-characterized system [39]. The uncontrolled KSE, $\mathcal{F} = 0$, exhibits rich dynamics and spatiotemporal chaos, which has made it a common toy problem and proxy system for the Navier-Stokes Equations. Spatially localized control is implemented in the KSE with $N = 4$ equally spaced Gaussian "jets" located at $X \in \{0, L/4, 2L/4, 3L/4\}$ as done

by Bucci et al. [16] and Zeng and Graham [242], where $a_i(t) \in [-1, 1]$ is the control signal output by the RL control agent,

$$\mathcal{F}(x,t) = \sum_{i=1}^{4} \frac{a_i(t)}{\sqrt{2\pi\sigma_s}} \exp\left(-\frac{(x - X_i)^2}{2\sigma_s^2}\right). \tag{4.5}$$

The system is time evolved with a time step of $\Delta t = 0.05$ using the same numerical method and code as Bucci et al. [16] with a third-order semi-implicit Runge-Kutta scheme, which evolves the linear second and fourth order terms with an implicit scheme and the nonlinear convective and forcing terms with an explicit scheme. 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. To serve as an analog to energy-saving flow control problems, we are interested in the minimization of the dissipation, D, and total power input, P_f required to power the system and jets of the KSE system, which are described by $D = \langle(\frac{\partial^2 u}{\partial x^2})^2\rangle$ and $P_f = \langle(\frac{\partial u}{\partial x})^2\rangle + \langle u\mathcal{F}\rangle$, respectively. Here $\langle\cdot\rangle$ is the spatial average. These arise from writing an evolution equation of the "kinetic energy" $\frac{1}{2}\langle u^2\rangle$ by multiplying (4.4) by u and integrating over the domain to yield the equation

$$\frac{d}{dt}\frac{1}{2}\langle u^2\rangle = P_f - D. \tag{4.6}$$

4.2.2 Background and method formulation

At its core, deep RL is a cyclic learning process with two main components: the RL agent and the environment. The agent is the embodiment of the control policy and generally a neural network, while the environment is the target system for control. During each cycle, the agent makes a state observation of the system, s_t, and outputs an action, a_t. The impact of this action on the environment is then quantified by a scalar quantity, 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. This learning cycle is shown in Fig. 4.1.

Deep RL's poor sampling complexity makes this cyclic process costly or even intractable for learning control strategies when simulations or experiments of the environment are expensive to realize. We aim to circumvent this issue in a completely data-driven manner by training the RL agent with a surrogate reduced-order model (ROM) of the target system. Our method can be divided into five steps: 1) obtain ROM training data 2) learn a reduced-order embedding coordinate transformation using an autoencoder 3) learn the RHS of the controlled system's dynamics in the reduced-order coordinates using Neural ODEs, 4) extract a control policy from this ROM using deep RL, 5) deploy and assess the control strategy in the real system. An outline of this process is presented in Fig. 4.2.

Step 1: Training Data Collection: We obtain our training data for our data-driven model by observing actuated trajectories of the target system. As we assume there is neither a model nor control strategy available at this stage, the actuations applied are randomly sampled from a uniform distribution of the available range of control inputs. We comment here that although the ambient space dimensionality for many flow systems can be extremely high, the underlying dynamics are generally much lower dimensional. And equally importantly, we only need to sample the responses of the system to the available actuations, not to random perturbations in every dimension of the ambient state space. Our approach determines the total number of degrees of freedom – 12 in the present case – needed to model the actuated dynamics. This means that when collecting data to sample the state-action space, we are not concerned with the dimensionality of the ambient input configurations and action configurations, but rather we need to sufficiently sample the combinations of manifold states and actions. In other words, although our formally infinite-dimensional system may be resolved indefinitely by adding more grid points, the underlying dynamics do not change. Thus data sampling complexity is independent of mesh resolution (assuming the system is already resolved and we have a decent reduced-order representation) and the state-action space complexity is of size *manifold dimension* $\times$ *action dimension*.

The observed series of states, s_t, applied actions, a_t, at time t, and the resulting state after τ time units, $s_{t+\tau}$ are saved as transition snapshots $[s_t, a_t, s_{t+\tau}]$ for ROM training. In our demonstration with the KSE, we take the state observable to be the solution of the KSE at time t, $s_t = u(t)$, and the coinciding control signals to the jet actuations to be the action, $a_t = a(t)$ (where $a_i(t)$ is the signal to the ith jet).

Step 2: Learning the Manifold Coordinate System: We utilize undercomplete neural network autoencoders [74] to learn the coordinate transformation to and from the lower-dimensional manifold of the actuated dynamics of our system. Undercomplete autoencoders are hourglass-shaped neural networks that are composed of an encoder that restricts the number of degrees of freedom and the decoder that expands back to the full space. The encoder, $h_t = \chi(s_t; \theta_E)$, is tasked with learning the mapping from the full-state representation, $s_t \in \mathbb{R}^{d_s}$, to the manifold representation, $h_t \in \mathbb{R}^{d_h}$, and is parameterized by weights θ_E. Importantly, $d_h \ll d_s$ as our objective is to learn a reduced-order manifold representation. The decoder, $\hat{s}_t = \tilde{\chi}(h_t; \theta_D)$, is tasked with learning the mapping from the manifold representation to the full-state representation and is parameterized by weights θ_D. The autoencoder is trained to minimize the mean squared reconstruction loss (MSE) $\mathcal{L}_{AE} = \langle ||s_t - \hat{s}_t||^2 \rangle$ where $\hat{s}_t = \tilde{\chi}(\chi(s_t))$ using the snapshots of state data obtained from the previous step and $\langle \cdot \rangle$ is the average over a training batch. In our example with the KSE, we also perform an intermediary change of basis from the observed state s_t to its projection onto the Principal Component Analysis basis (computed from the training data) prior to input to $\chi(s_t; \theta_E)$ and a return to the full space post output from $\tilde{\chi}(h_t; \theta_D)$. This change of basis does not reduce dimension and although NNs are universal function approximators, we found empirically in [128] that this change of basis results in a lower reconstruction error indicating that this change of basis eases the training procedure.

We identify the dimension of the finite-dimensional manifold, $d_{\mathcal{M}}$, by tracking the MSE performance of the autoencoders as we vary d_h, [128, 129]. We comment that we use a standard autoencoder here as we and others [129, 223] have found empirically that it performs

well and yields a manifold representation that is conducive to forecasting tasks. [223] found that variational autoencoders and convolutional autoencoders exhibited no clear advantage over standard feedforward autoencoders for the KSE system as well as the FitzHugh-Nagumo equation. However, for systems with more complex underlying dynamics (e.g. intermittent bursting), consideration may be warranted for different AE frameworks and latent space regularization mechanisms. For example, in [57] the authors found that systems with multiple time-scales could be better captured via a charts and atlases approach of splitting the manifold dynamics into a collection of autoencoders and time-stepping models. [230] found improvement in forecasting the Lorenz system by promoting orthogonality and isotropy in the latent space while [53] promoted orthogonality with β-VAEs. However, what properties a latent space should possess for improved forecasting is still an open and active research question, we opt for simplicity.

Step 3: Learning the Actuated Dynamics along the Manifold: We next develop a neural-ODE (NODE) model of the dynamics in the manifold coordinates h by learning the RHS (vector field) of the ODE $\dot{h} = g(h_t, a_t; \theta_M)$, where θ_M denotes the neural-network parameters that are to be determined [24]. To make dynamical forecasts given a manifold state, h_t, and input control action, a_t, the evolution of the manifold state τ time units forward, $h_{t+\tau}$, can be computed for a given $g(h_t, a_t; \theta_M)$ with a standard time-integration algorithm. To train the NODE network, the same dataset generated in step 1, $[s_t, a_t, s_{t+\tau}]$ can be used. Because the NODE network's purpose is to model the vector field of the dynamics in the manifold coordinates, we must first convert the training data to $[h_t, a_t, h_{t+\tau}]$ using $\chi(s_t; \theta_E)$. This data set can then be used to train the NODE network to minimize the forecasting loss: $\mathcal{L}_{NODE} = \langle ||h_{t+\tau} - \hat{h}_{t+\tau}||^2 \rangle$ where $\hat{h}_{t+\tau}$ is the forecast made by integrating the NODE with a numerical integrator forward τ time units. In this work we use the Dormand-Prince 45 numerical scheme [45]. The gradient of the loss with respect to the network parameters, θ_M, can be obtained by either performing back-propagation via automatic differentiation through all the steps of the time integration scheme (which can be memory intensive for very large

forecasts) or by the adjoint method described by Chen et al. [24]. In this work, we opt for the former as we did not encounter memory issues and past works have demonstrated good agreement between both options [129]. We highlight that once training is complete, NODE forecasted trajectories, which are in the manifold coordinates, can be recovered back to the original ambient space, s, at any point using the decoder, $\tilde{\chi}(h_t; \theta_D)$, obtained in step 2.

Step 4: Learning a control strategy from the NODE-ROM with deep RL: We differentiate our DManD-RL method from typical RL with two distinctions. First, the RL agent learns by interacting with the learned NODE-ROM, not the true environment. Second, during training, the RL agent learns in the manifold coordinate, h, not the observable space, s. We point out that the usual RL nomenclatures for state transitions, s_{t+1} or s', are written here as $h_{t+\tau}$ in the manifold representation and $s_{t+\tau}$ in the ambient representation, to make explicit the fact that the time interval τ is, in fact, a parameter of the system. To train the RL agent, the NODE-ROM is first initialized with an encoded initial condition. The agent learns by interacting only with the NODE-ROM: given a state in the manifold representation, h_t, the agent attempts to map it to the optimal control action. This action is then passed back to the NODE-ROM and the evolution in h subject to the prescribed actions is obtained by integrating the NODE-ROM forward in time. In this work we aim to lower the total power consumption $D + P_f$ of the KSE dynamics, so we define the immediate reward for the RL algorithm as $r_t = -(D(t) + P_f(t))$. The algorithm seeks to maximize the long-time discounted cumulative reward $R_t = \sum_{l=0}^{\infty} \gamma^l r_{t+l\tau}$ where $\gamma = 0.99$. The reward return for this manifold state-action pair ($[h_t, a_t]$) can be estimated by decoding the resulting manifold trajectory using $\tilde{\chi}(h_t; \theta_D)$ and estimating r_t, which is used to update the agent. The learning cycle then repeats. In this work, we use the Deep Deterministic Policy Gradient (DDPG) RL method [126], but we emphasize that any general RL method can be used with our framework.

Step 5: Deploying and Validating the DManD-RL Control Strategy: Once RL training within the NODE-ROM is complete, the learned DManD-RL policy can be

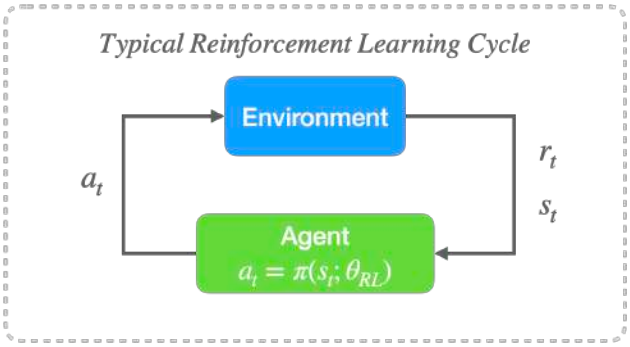


Figure 4.1: Typical reinforcement learning cycle.

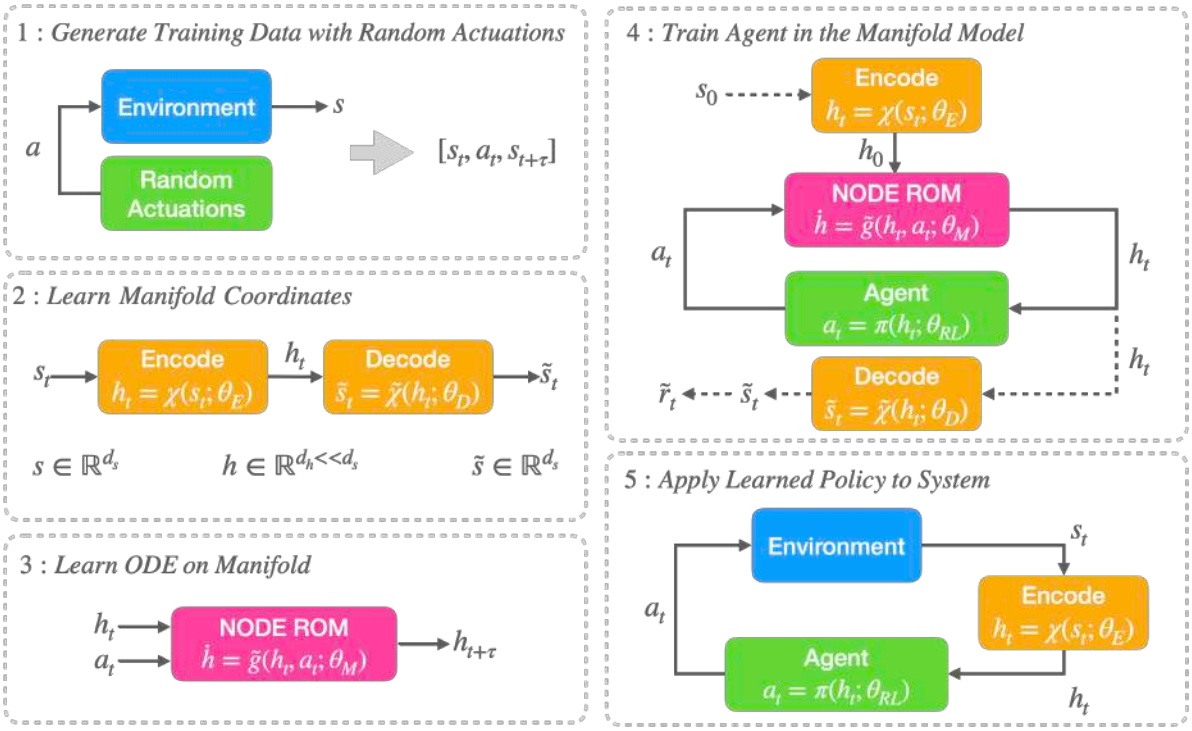


Figure 4.2: Procedure for learning a NODE-ROM from data, combining it with RL to approximate a control policy, and deploying the approximate policy back to the true system.

applied to the true system. As the agent was trained in the reduced manifold space, the encoder obtained in step 2 must be inserted between the environment and agent to map state observations to the manifold representation prior to input to the agent. The DManD-RL policy can then be applied in the typical closed-loop fashion. If desired, new additional on-policy data can be collected to further improve the model/agent in an iterative fashion or the agent can even be simply fine-tuned with real-system training.

4.3 Results

4.3.1 Neural ODE model evaluation

Training data was collected by capturing snapshots of the KSE experiencing jet actuations whose magnitude and direction are randomly sampled from a uniform distribution of the allowable action range described in Section 4.2. We collect 40,000 of these transition snapshots spanning 10,000 time units for our NODE-ROM training, where $\tau = 0.25$ time units. We emphasize here that in this study τ is kept constant for simplicity but in practice τ can vary from snapshot to snapshot. Actuation signals applied during these trajectories and during control are held constant for the duration of τ (i.e. zero-order hold; see Fig. 4.5a for typical actuation trajectories).

As the autoencoder serves as the mapping function to and from the manifold representation h_t, we need to estimate the number of degrees of freedom in the manifold, d_h, to accurately capture the dynamics of the system. As d_h is generally not known a priori for complex systems, we empirically estimate it by tracking the reconstruction error as a function of d_h as done by Linot and Graham [128, 129]. In this work our autoencoder networks utilize encoders and decoders each with a hidden layer of size 500 activated by sigmoid functions.

Before considering actuated data, we first apply the autoencoder analysis to the natural unactuated KSE. For the domain size $L = 22$ considered here, the manifold dimension has been determined to be $d_{\mathcal{M}} = 8$ in past studies [43, 128]. This will aid in validating our procedure and give insight into the results with the actuated system. To validate our procedure with the natural KSE, we collect 40,000 snapshots of the KSE where $a_t = 0$, to capture the unactuated KSE dynamics. Shown in Fig. 4.3a are the mean-squared errors (MSE) of reconstruction for undercomplete autoencoder networks trained on natural, unactuated KSE snapshots for bottleneck layers of varying dimension d_h. We note that as d_h increases from 7 to 8 the MSE exhibits a sharp drop. Here, $d_h = 8$ corresponds to the minimum number of degrees of freedom required to fully represent the manifold in which the natural KSE

data lives, which agrees with previous studies [43, 128]. We denote this manifold dimension of $d_h = 8$ as $d_{\mathcal{M}}$. We note that increasing d_h beyond $d_{\mathcal{M}}$ does not significantly improve reconstruction error, indicating that the data can be effectively represented in $d_h = 8$ and additional dimensions are superfluous [128].

Repeating this analysis with the snapshots collected from the randomly actuated KSE to estimate the dimensionality of the actuated dynamics, we observe a similar phenomenon. Shown in Fig. 4.3b are the mean-squared errors of reconstruction as d_h is changed for undercomplete autoencoders trained with KSE snapshots obtained from trajectories perturbed randomly by the four jet actuators. We note that compared to the Fig. 4.3a, the sharp drop in MSE is delayed to $d_h = 9$, with a lesser drop at $d_h = 11$.

To dynamically explain this, shown in Fig. 4.3c is a cartoon of the $d_{\mathcal{M}} = 8$ embedded manifold $\mathcal{M}$ on which the long time dynamics of the unactuated KSE lives, where states that begin off of the manifold are attracted exponentially to it due to dissipation. When actuations are applied, the KSE produces states and trajectories that live off of the natural attractor, effectively giving the manifold "thickness" in additional dimensions, as shown schematically in Fig. 4.3d. This "thickness" require additional degrees of freedom, i.e. increased $d_{\mathcal{M}}$ from the original system, to accurately capture the dynamics.

To further support this view, shown in Fig. 4.4 is the power spectral density of the unactuated and actuated data sets $u(t)$ used to train the autoencoders shown in Fig. 4.3. We note that in the presence of actuations the data exhibits a broadening of the high-wavenumber tail compared to the unactuated data. Physically, this indicates that there are additional high-wavenumber spatial features in the actuated data set, which require additional degrees of freedom to accurately capture. Finally, we highlight in Fig. 4.3a and Fig. 4.3b that our autoencoders outperform dimensionality reduction using only Principal Component Analysis in reconstruction error by several orders of magnitude once $d_{\mathcal{M}}$ has been reached.

The autoencoder MSE landscape of the actuated KSE data suggests that a manifold

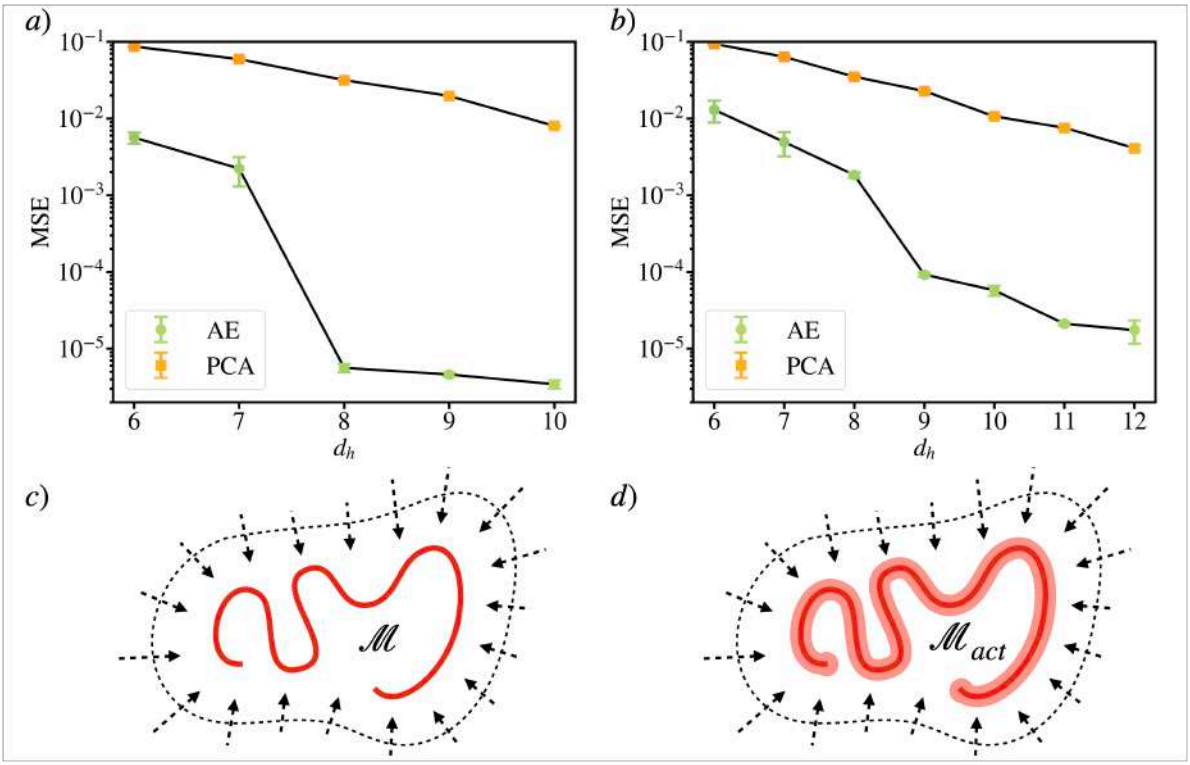


Figure 4.3: Mean squared reconstruction error for autoencoders trained on a) unactuated and b) actuated KSE data vs. manifold representation dimension, d_h. Cartoon illustrations of the inertial manifolds in which the data lives on for the c) unactuated and d) actuated system.

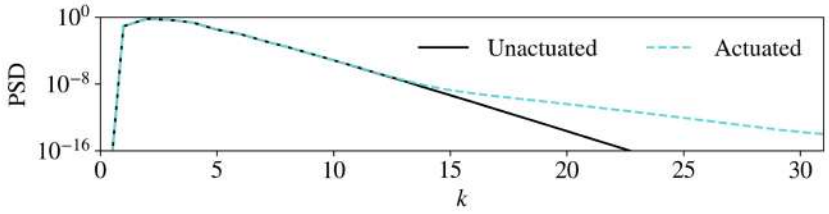


Figure 4.4: Power spectral density vs. wavenumber of unactuated and actuated data.

representation of $d_h = 12$ should be sufficient to capture the dynamics of the controlled KSE. Using the $\chi, \tilde{\chi}$ mappings learned by the $d_h = 12$ autoencoder, a NODE network with two hidden layers of size 200, 200 with sigmoid activation was trained to model the actuated dynamics in the $d_h = 12$ manifold. This NODE was trained using the same data used to train the autoencoder by simply converting the collected data of $[s_t, a_t, a_{t+\tau}]$ to $[h_t, a_t, h_{t+\tau}]$ with $h_t = \chi(s_t; \theta_E)$.

To demonstrate the learned NODE-ROM's predictive capability, shown in Fig. 4.5c and Fig. 4.5d are two example KSE trajectories experiencing random jet actuation signal

sequences visualized in Fig. 4.5a, Fig. 4.5b, respectively. We also show in Fig. 4.5e and Fig. 4.5f the decoded $d_h = 12$ trajectory forecast by the NODE-ROM beginning from the same initial condition and following the same actuation signal sequence as its ground truth trajectory, Fig. 4.5c, Fig. 4.5d, respectively. We note that the forecasted trajectories agree qualitatively with the ground truth for about 20-30 times units or 1-1.5 Lyapunov times of the natural system, shown in Fig. 4.5g, Fig. 4.5h. Finally, we show in Fig. 4.5i, Fig. 4.5j the 12 dimensional manifold representation of these trajectories, h. We highlight that the learned representation evolves smoothly in time.

To quantitively compare the ensemble performance of the NODE-ROM and the actuated KSE, an ensemble of 50 actuated forecast/ground truth pairs of trajectories (where the random actuation sequence and initial conditions between each pair are the same) was used to compute the spatial and temporal autocorrelation, shown in Fig. 4.6a and Fig. 4.6b, respectively. We note that the NODE-ROM accurately captures the spatial autocorrelation of the actuated KSE, while the temporal autocorrelation exhibits good agreement with a slight temporal dilation. These results indicate that the NODE-ROM is accurately capturing the distribution of features of the actuated KSE in both space and time.

As the NODE-ROM was trained with only transition snapshots experiencing random actuations, a natural question is: how well does our model capture the underlying natural dynamics, i.e. with $a_t = 0$? Shown in Fig.4.7a and Fig.4.7b are two example KSE trajectories where $a_t = 0$. Accompanying the unactuated ground truth trajectories shown in Fig.4.7a and Fig.4.7b are the decoded $d_h = 12$ trajectories forecasted by the same NODE-ROM that was trained on actuated data beginning from the same initial conditions and with zero actuation signal input, shown in Fig.4.7c and 4.7d, respectively. We note that the forecasted trajectories agree qualitatively with the ground truth for about 1-1.5 Lyapunov times. We again assess the spatial and temporal autocorrelations between an ensemble of 50 pairs of ground truth trajectories and NODE-ROM forecasts, shown in Fig. 4.6c and 4.6d, respectively. We note that the ROM matches the spatial autocorrelation of the KSE very

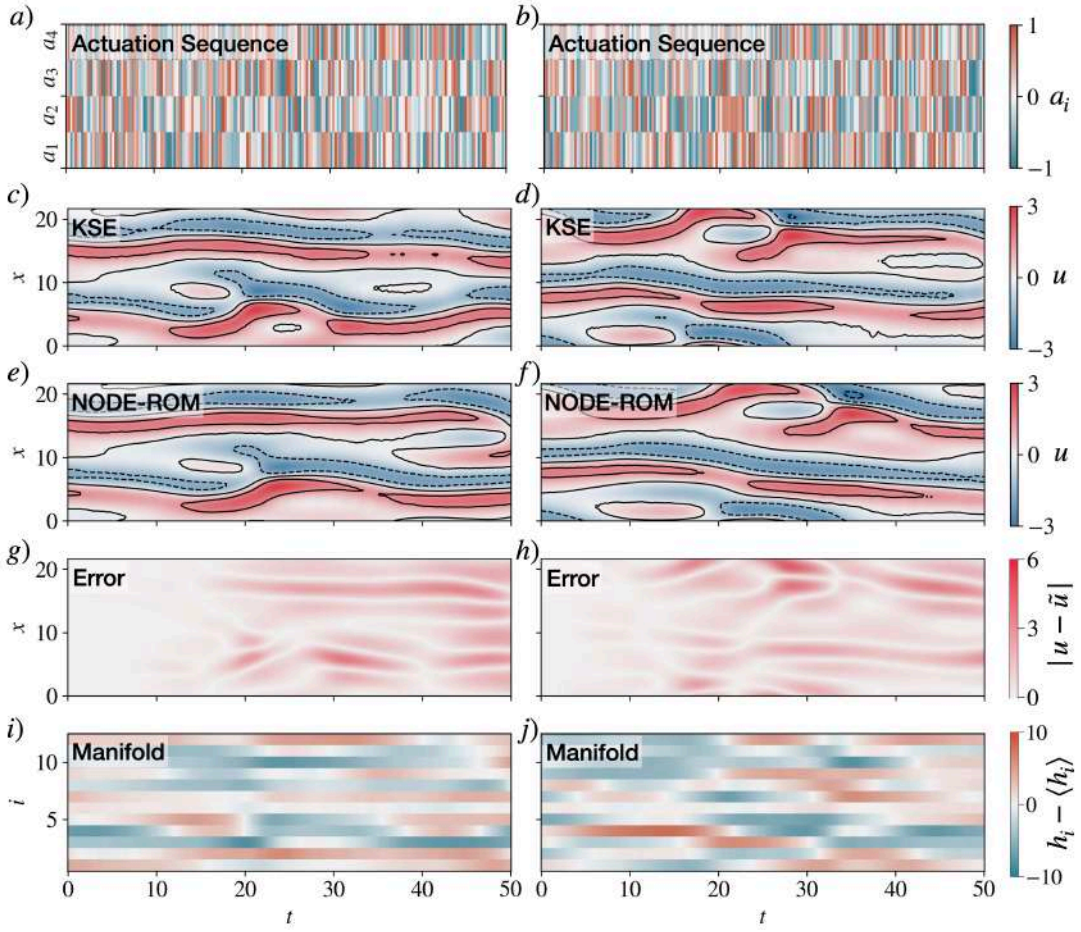


Figure 4.5: Left column: (a) random actuation sequences $a_i(t)$ (c) ground truth KSE trajectory starting from a random initial condition following actuation sequences in (a), (e) the decoded NODE-ROM trajectory following actuation sequence (a) and the same initial condition in (c). The absolute error difference is shown in (g) while the manifold representation, h, is shown in (i). A second example is shown in the right column: (b), (d), (f), (h), and (j), respectively.

well while the temporal autocorrelation reveals that the ROM exhibits a more pronounced but still quantitatively small temporal dilation. We emphasize that despite the training data for the ROMs being obtained from randomly actuated trajectories and that there is no substantial unactuated data, the ROMs still recover the unactuated dynamics well.

4.3.2 Model-based control performance

With this data-driven NODE-ROM, a control agent was trained by interacting only with the NODE-ROM in the manifold space with r_t estimated from the $s = \tilde{\chi}(h; \theta_D)$ decoded

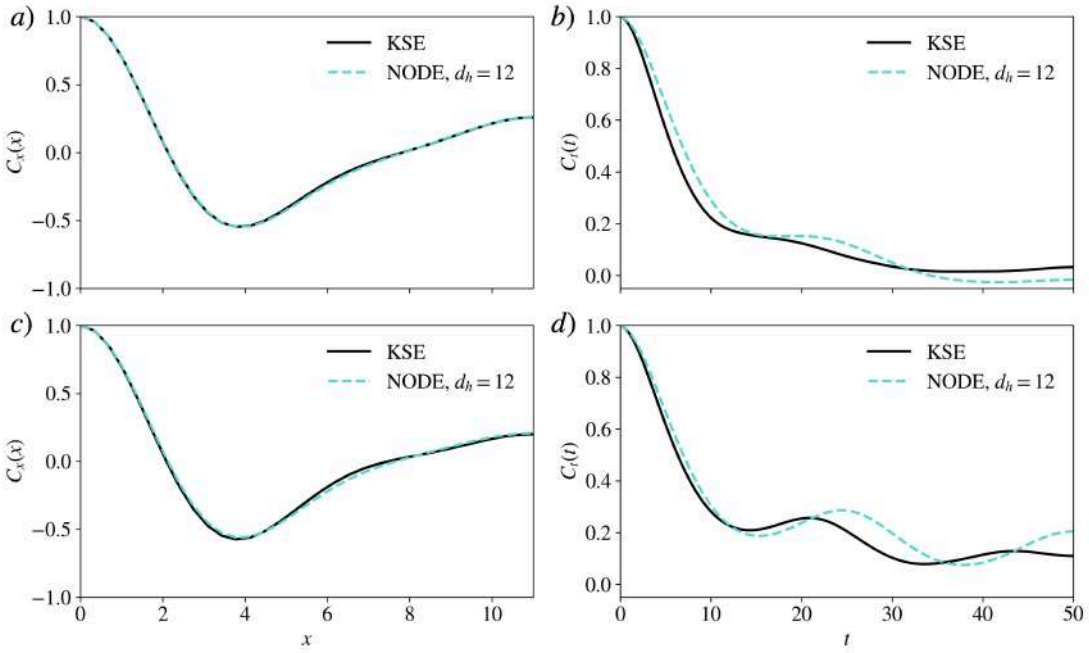


Figure 4.6: a) Spatial and b) temporal autocorrelation computed from an ensemble of trajectories experiencing random jet actuations. NODE-ROM ($d_h = 12$) forecasts were produced using the same initial conditions and jet actuation sequences as each respective trajectory produced from the true KSE. c) Spatial and d) temporal autocorrelation computed from an ensemble of trajectories with actuations set to zero. NODE-ROM ($d_h = 12$) forecasts were produced using the same initial conditions as each respective trajectory produced from the true KSE.

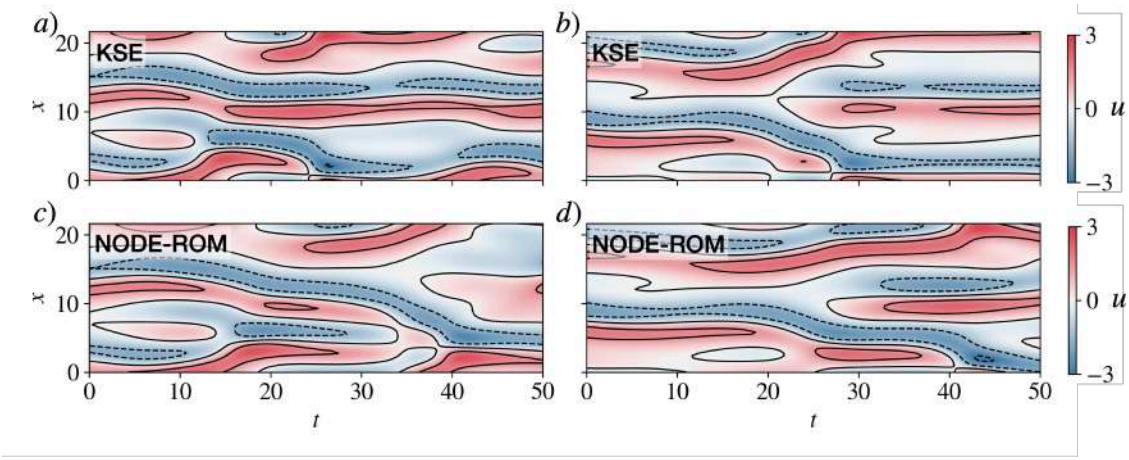


Figure 4.7: Example unactuated trajectories of the KSE in a) and b) and their corresponding decoded NODE-ROM ($d_h = 12$) forecasts starting from the same initial conditions in c) and d), respectively.

state. We set the NODE-ROM model transition time to be $\tau = 0.25$ for all experiments. 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 $s_{t+\tau}$. In this work the DDPG actor and critic networks use 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 DManD-RL policy, the learned control agent was applied to the NODE-ROM, with an example controlled trajectory shown in Fig. 4.8a. 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 4.8c, revealing that this equilibrium exhibits dissipation much lower than the natural unactuated dynamics. To assess how well this DManD-RL 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. 4.8b. 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. 4.8d, are nearly identical to that of the NODE-ROM in Fig. 4.8c.

To demonstrate the robustness of the DManD-RL policy, shown in Fig. 4.8e are the dissipation trajectories of the true KSE beginning from 15 randomly sampled test initial conditions that the DManD-RL 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 horizon it was trained in.

Here we emphasize that the DManD-RL 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.

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. The agent is not forced to act out a series of actions based on a forecasted trajectory but rather it is reassessing at every action point in the true system. 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. An example of the policy still succeeding far from the training horizon can be seen in Fig. 4.8e.

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 [242]. Solutions identified by this continuation in forcing magnitude are shown in Fig. 4.10, which reveals that the 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 $E1$ [39]; we obtained a similar result with an RL agent trained on interactions with the full system [242]. A similar observation was made for RL control of 2D bluff body flow [124].

We speculate that in systems with complex 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 that aim to minimize dissipation while penalizing control action; as was shown in [124, 242]. 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. This is a promising outlook for RL as the dynamics of even more complex chaotic systems, such as turbulent flows governed by the Navier-Stokes equations, are also known to be structured around such solutions [76].

To highlight the importance of the type of data-driven model, we compare to Dynamic Mode Decomposition (DMD), a common data-driven method that has been applied to fluid flows and dynamical systems. Recently, Qin et al. [169] demonstrated active flow control of 2D cylinder flow with a reward signal targeting the minimization of DMD mode amplitudes. Here we specifically investigate the performance of using DMDc [168], the extension of DMD to account for control input, as an alternate data-driven model of the KSE dynamics for RL policy learning. The DMDc model takes the following form,

$$s_{t+\tau} = \mathbf{A}s_t + \mathbf{B}a_t, \tag{4.7}$$

where $\mathbf{A}$ is the state transition matrix and $\mathbf{B}$ is the control input matrix. Using the same training data as our NODE-ROM, $[s_t, a_t, s_{t+\tau}]$, , we obtained a full state DMDc model by simultaneously fitting the two matrices with the $[s_t, a_t, s_{t+\tau}]$ data as described in [168]. This model was then used as the environment for the RL policy training.

Shown in Fig. 4.9a is the DMDc-based RL control agent applied to the DMDc model it was trained in. We show in Fig. 4.9c that the agent is able to rapidly minimize the DMDc model values of D, P_f. We highlight that the DMDc-based strategy accomplishes this by exploiting the unphysical dynamics of the DMDc model. To elaborate, shown in Fig. 4.9b

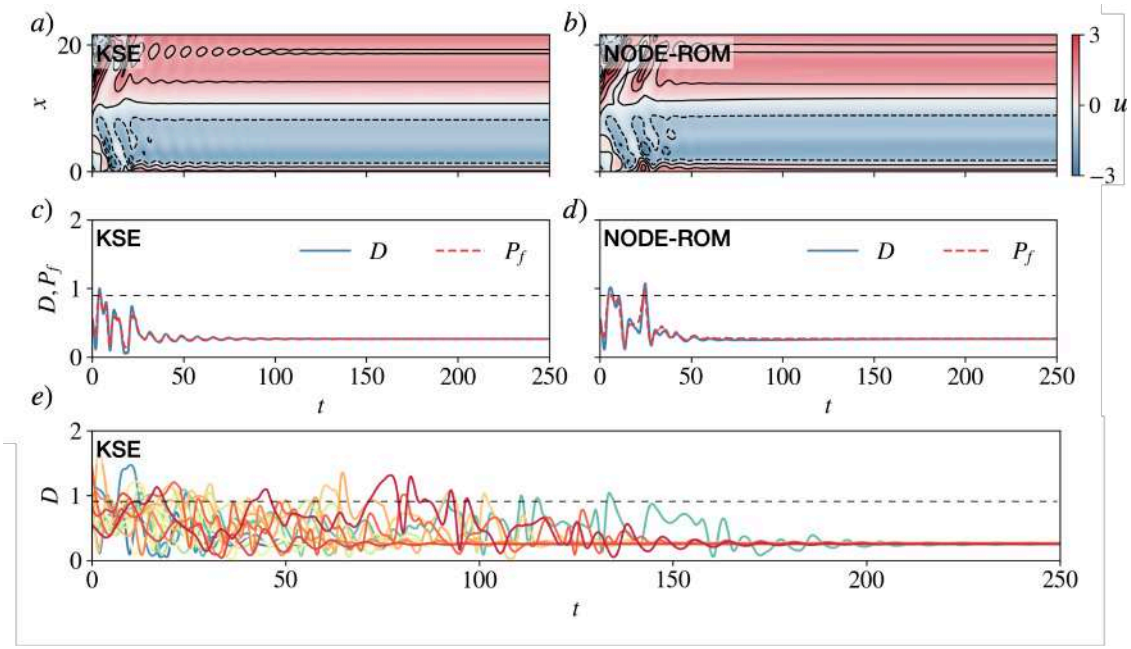


Figure 4.8: ROM-based RL agent applied to the same initial condition in the a) true KSE and b) data-driven reduced-order model (decoded, $d_h = 12$). The corresponding invariant quantities of dissipation and total input power for the c) true KSE and d) learned reduced-order model. The dashed black line 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 of the KSE.

is the same strategy applied to the true KSE with the same initial condition as the DMDc trajectory in Fig. 4.9a. The DMDc-based RL strategy fails to minimize the control objective when applied to the true KSE. Furthermore, the strategy even results in performance that is even worse than the time-averaged uncontrolled system, which can be seen in the above average D, P_f values at long times in Fig. 4.9d, where the controller appears to get trapped in a high energy region of the KSE. This mismatch in performance is due to the inability of the linear model to describe the nonlinear dynamics of the KSE.

4.4 Summary

In this chapter, we introduce our NODE-ROM-Based RL method, DManD-RL, that maintains an end-to-end data-driven method of learning control policies from a limited data set. The governing equations and control term do not need to be known or specified–we only

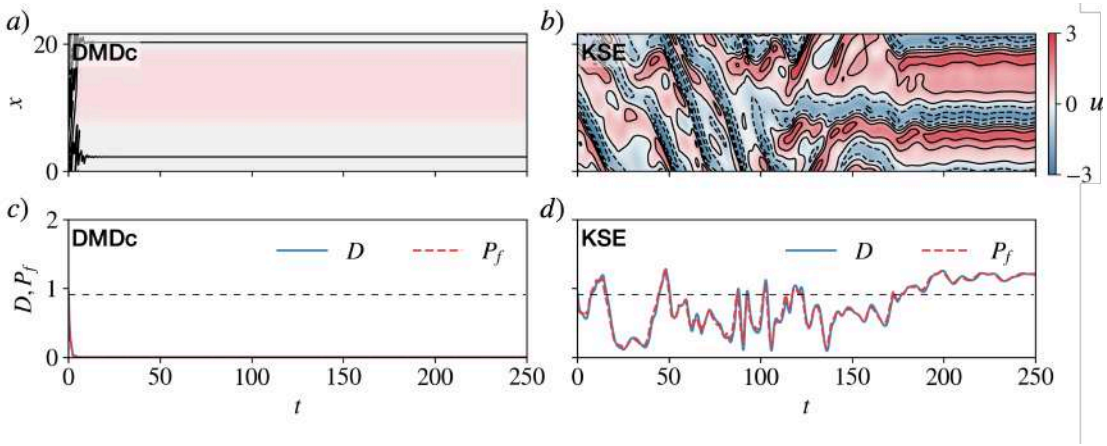


Figure 4.9: DMDc control policy applied to the same initial condition in the (a) DMDc model and (b) True KSE. Corresponding dissipation and total input power for the (c) DMDc model and (d) KSE. The dashed black line represents the average of the natural KSE dynamics.

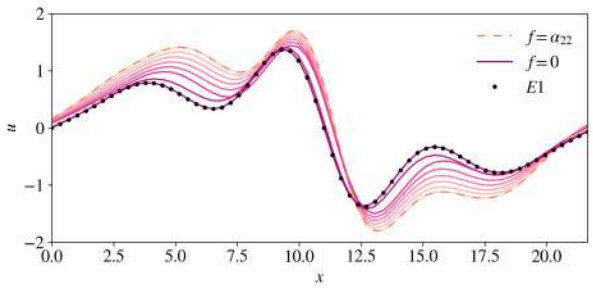


Figure 4.10: Forcing continuation from the forced equilibrium state (under forcing $f = \alpha_{22}$) discovered by NODE-ROM based RL policy (orange, dashed) to the unactuated KSE system (purple). The known equilibrium E1 of the KSE system is also provided (dots).

assume that the system's actuated dynamics can be represented by some governing system of ODEs in the manifold coordinate. Furthermore, we do not impose any structural relationship between the control input and the dynamics of the systems as previous methods have done in the past. This is advantageous for many complex flow control systems as the relationship between the dynamics and the control inputs are not simple linear relationships [13, 64]. We exploit the notion that many nominally high-dimensional systems actually display low-dimensional dynamics and aim to capture and model these actuated dynamics from data using a combination of autoencoders and Neural ODEs. Then, with deep RL, we extract control policies from our data-driven ROMs. Importantly, our method does not require RL to directly interact with the target system nor does it require great modification of standard RL algorithms.

Our method identifies the lower-dimensional manifold the actuated dynamics live on with autoencoders and models the dynamics with neural ODEs, which are capable of making predictions for arbitrary time intervals [24] and have been demonstrated to produce good predictions of spatiotemporal chaotic systems [129]. Neural ODEs are also a natural formulation and deep architecture for the systems we are attempting to model and control. Our method also respects the Markovian nature underlying many RL frameworks as well as those of our target dynamical systems, unlike other common data-driven methods such as reservoir computing and recurrent neural networks. The use of high-fidelity low-dimensional models allows for faster interaction simulations (or even running multiple surrogate environments in parallel) compared to costly simulations or experiments.

By using a more informative and compact state space representation than a high-dimensional sensed state, the RL policy training load is lessened. In naive applications of RL, a portion of the agent's network capacity is exhausted in learning to transform the raw inputs to more useful internal representations [81]. Similar to Ha and Schmidhuber [81], in our method, we also explicitly separate the agent's learning of the control policy and the task of learning a useful state representation into two discrete tasks.

We apply our DManD-RL RL method to the KSE, a proxy system for turbulent flows that exhibits rich spatiotemporal chaotic dynamics. From a limited data set generated from trajectories of the KSE under random actuations, we are able to find a reduced-order mapping between the full state and the reduced low-order dynamics of the actuated KSE, as well as to successfully model the dynamics in this reduced space. We find that our NODE-ROM has good forecasting ability and captures ensemble characteristics of the KSE well for not only actuated predictions but also unactuated predictions, for which it was not given training data. In this work, it was found that NODE-ROM training data obtained from snapshots of the KSE experiencing random jet actuations was sufficient. With even more complex systems, it may be possible that this sampling approach is insufficient. If necessary, the NODE-ROM can be iteratively improved by applying the learned control strategy to the true system, collecting additional data, and fine-tuning the NODE-ROM with the new data. Using the updated NODE-ROM, the RL agent can then be updated. Together the control agent and model can be improved in a cyclic training fashion. We highlight that our model could be alternatively trained and used in the "Dyna"-style [202] for improved online learning. We comment that, in the scope of this article, the manifold coordinate is learned from the full state. However, in many systems of interest, this information may not be readily available. In the absence of full-state observations, a combination of sensor measurements and time-delays should suffice so long as they satisfy Whitney's Embedding Theorem. While we demonstrate in this article that this framework can achieve similar performance compared to past studies that involved direct applications of RL [242], other systems of interest may exhibit much higher dimensional action spaces or greater dynamic complexity, which may require smarter data collection and successive model updates.

Finally, we acknowledge here that although we do not obtain substantial speed up in per-step computation time between the semi-implicit KSE solver and the NODE-ROM, we emphasize that for flow systems of increasing complexity and Re, the time to solve $\mathcal{O}(10^6)$ or more degrees of freedom in a direct numerical simulation is much greater than that of solving

a system of ODEs $\mathcal{O}(10^1 - 10^2)$. In Ch. 7 this speed-up will be imperative for finding control policies for Couette flow. Aside from the computational speed up, we also emphasize that our method greatly reduces the data generation burden, as we successfully obtained a control strategy for the KSE using only 40,000 transition snapshots, while a naive application of RL directly on the KSE required generating approximately 1.6 million transition snapshots [242].

Applying RL to the NODE-ROM environment resulted in an interesting control policy that discovered a low-dissipation equilibrium state and exploited this state to minimize D, P_f. When we deployed this DManD-RL control policy to the true KSE, we observe that not only are the same equilibrium states targeted, but the performance is nearly indistinguishable from our learned NODE-ROM. This indicates that not only does the NODE-ROM capture the existence of the forced equilibrium, but it also captures the dynamics sufficiently well such that the agent could find it during training and exploit it in a manner that still translates in the original system. We emphasize that we accomplished this with a 12-dimensional NODE-ROM while the full state is 64-dimensional. A continuation in the magnitude of the forcing profile reveals that the RL discovered equilibrium state is connected to an existing equilibrium solution of natural KSE. The naturally occurring RL optimization about underlying solutions of the system has been observed in the KSE [242] as well as bluff-body flows [124], which is promising, as more complex dissipative systems, such as the turbulent dynamics of the Navier-Stokes equations, are also known to be organized about various types of invariant solutions [76].

Chapter 5

Stabilized neural ODEs [1]

After discovering we could accurately model and control the KSE, our next goal was to apply our modeling approach to turbulent flows. However, we discovered, for higher-dimensional systems, that the dynamics would become unstable at long-times. This chapter explores how to better structure the neural ODEs for stability, and interesting additional uses of this added structure. We do not use autoencoders to reduce dimension in this chapter.

In data-driven modeling of spatiotemporal phenomena careful consideration is needed in capturing the dynamics of the high wavenumbers. This problem becomes especially challenging when the system of interest exhibits shocks or chaotic dynamics. We present a data-driven modeling method that accurately captures shocks and chaotic dynamics by proposing a new architecture, *stabilized neural ordinary differential equation* (ODE). In our proposed architecture, we learn the right-hand-side (RHS) of an ODE by adding the outputs of two NN together where one learns a linear term and the other a nonlinear term. Specifically, we implement this by training a sparse linear convolutional NN to learn the linear term and a dense fully-connected nonlinear NN to learn the nonlinear term. This contrasts with the standard neural ODE which involves training a single NN for the RHS. We apply this setup to the viscous Burgers equation, which exhibits shocked behavior, and show that stabilized neural ODEs provide better short-time tracking, prediction of the energy spectrum, and ro-

[1] The text of this chapter is adapted from the publication by A. J. Linot et al. *Journal of Comp. Physics*, 474, 2023

bustness to noisy initial conditions than standard neural ODEs. We also apply this method to chaotic trajectories of the Kuramoto-Sivashinsky equation. In this case, stabilized neural ODEs keep long-time trajectories on the attractor, and are highly robust to noisy initial conditions, while standard neural ODEs fail at achieving either of these results. We conclude by demonstrating how stabilizing neural ODEs provide a natural extension for use in reduced-order modeling by projecting the dynamics onto the eigenvectors of the learned linear term.

5.1 Introduction

With the ever-increasing amount of data available from experiments or high-resolution simulations it has become common to build models directly and strictly from data. This tactic is powerful in that no *a priori* knowledge of the system is needed and it forces the data to guide the solution. Unfortunately, models built without any knowledge of the system often lack robustness to perturbations in the data and do a poor job of generalizing outside of the training data. In particular, we are interested in the problem of modeling dynamical systems from time-series data. For example, this could be predicting the future location of a pendulum given angle and velocity data, predicting the weather, or predicting chemical concentrations in a reactor. Time-series predictions are especially susceptible to issues of robustness because states are repeatedly put through the model and error that pushes states away from the training data becomes worse as the state moves further and further away from the training data.

Here we seek to overcome the issue of robustness that can appear in data-driven models, by using the fact that often in dissipative partial differential equations (PDEs), stability comes from a linear operator that dissipates energy [37]. In particular, we are concerned with the *nonlinear* stability of attractors. By nonlinear stability, we mean finite perturbations to states on the attractor are pushed back to the attractor. This type of stability can

be shown using the energy method [200], with Lyapunov functions [112], or for dynamical systems which possess an inertial manifold [37]. The energy method in particular highlights the importance of the linear term in stabilizing many dissipative PDEs. For example, when computing the energy balance for the Viscous Burgers Equation (VBE) and the Navier-Stokes Equation dissipation comes from the diffusive term [166] and for the Kuramoto-Sivashinky equation (KSE) hyperdiffusivity dissipates energy [40]. Below we make this notion of stability more precise.

When modeling time-series data of autonomous systems (i.e. the right-hand side does not explicitly depend on time) we can represent the dynamics as an ODE

$$\frac{du}{dt} = f(u), \tag{5.1}$$

where u is the state of the system at a given time. If we want to model this system from snapshots of data $\{u(t_1), \ldots, u(t_N)\}$ we can take the neural ODE approach outlined in the previous two chapters. Some advantages this approach has over the other methods include data can be spaced further in time, trajectories can be evolved to arbitrary points in time, and adaptive time stepping can be used.

In Ch. 3 we showed that the long-time dynamics of the KSE could be captured using neural ODEs. However, this only works when the dimension of the problem is reduced. Without any dimension reduction high wavenumbers are amplified resulting in trajectories leaving the attractor. In this chapter, we seek to overcome this problem of high-wavenumbers amplifying not by dimension reduction, but by explicitly adding a linear term into the neural ODE approach. The properties of this linear term are described below.

A similar idea to learn a linear term of an ODE is "Linear and Nonlinear Disambiguation Optimization" (LANDO) [5]. In this approach, Eq. 5.1 is approximated with a dictionary-based kernel model. Then, the linear term on the RHS of the ODE is approximated by calculating the Jacobian of f around the zero base state. There are two clear practical

differences between that approach and the one we present. First, LANDO requires either time derivative data, or data closely spaced in time, whereas neural ODEs can use data widely spaced in time. Second, the Jacobian around the base state is only accurate if the function approximation is accurate around the base state. In our case, the base state is the zero solution, and the data we use for training never approaches the base state, so the Jacobian around this state may be poorly estimated.

In what follows, we show that by adding an explicit linear term into the neural ODE we can improve long-time predictive capabilities, we can improve robustness to "noisy" initial conditions outside of the range of the training data, and we can create a reduced-order model (ROM) from the basis given by the linear term. Section 5.2 describes the framework we take for constructing and training our models. Then, in Section 5.3 we present the performance of the different models on the VBE (Section 5.3.1) and on the KSE (Section 5.3.2). The VBE is presented to show how the model can handle shock behavior. The KSE is presented to show how the model handles the prediction of chaotic dynamics on an attractor and how the learned linear term provides a basis for a ROM. A key result in both situations is for standard neural ODEs, that lack an explicit linear term, high wavenumbers grow in time, but by learning a linear term this erroneous behavior is avoidable and the models become much more robust to noise. Finally, we conclude that the proposed models perform better at capturing statistics, are more robust to noise, and provide a method for selecting a natural basis for ROMs in Section 5.4.

5.2 Stabilized Neural ODE Framework

We first introduce our proposed *stablized neural ODE* framework, which builds a dissipative linear mapping into the conventional neural ODE framework. The approach is motivated by the stabilizing effect the linear term has in many dynamical systems. Then, we further discuss a natural and interesting extension of the proposed approach for ROMs.

For our analysis we consider snapshots of data $\{u(t_1), u(t_2), \ldots, u(t_N)\}$ that come from solutions to an autonomous ODE with the form

$$\frac{du}{dt} = f(u) = Au + F(u). \tag{5.2}$$

Here $u(t_i) \in \mathbb{R}^d$ is the state, $t \in \mathbb{R}$ is time, $A \in \mathbb{R}^{d\times d}$ is a matrix, and $f : \mathbb{R}^d \to \mathbb{R}^d$ and $F : \mathbb{R}^d \to \mathbb{R}^d$ are nonlinear functions. In particular we investigate systems where A is symmetric and F contains a quadratic nonlinearity, but the ideas should hold for any system with a dissipative linear term. For the two equations we consider, the VBE and the KSE in one spatial dimension, we know, not only that the long-time solutions lie on attractors, but that the long-time dynamics also collapse onto inertial manifolds [60, 224]. An inertial manifold is a finite-dimensional invariant attractive manifold that attracts trajectories exponentially fast [58].

With this data, we seek to train a NN to approximate the RHS of this ODE using the neural ODE framework. Typically only f is represented by a NN, but here we show that a better approach is training two NNs *simultaneously*, a linear NN – in our case this will be a convolutional NN (CNN) – for A and a nonlinear NN for F. When we approximate one of these terms with a NN, or make a prediction with these approximations, we will denote the approximation with a tilde (e.g. $\tilde{f}$).

To train these NNs, we numerically integrate an ODE forward in time to estimate the state $\tilde{u}(t_i + \tau)$

$$\tilde{u}(t_i + \tau) = u(t_i) + \int_{t_i}^{t_i+\tau} \tilde{h}(u(t); \theta)dt. \tag{5.3}$$

In what follows we consider three variations on this ODE $\tilde{h}$: 1) a nonlinear NN, $\tilde{h}(u) = \tilde{f}(u; \theta_1)$, 2) a nonlinear NN with a fixed linear term, $\tilde{h}(u) = Au + \tilde{F}(u; \theta_2)$, and 3) a nonlinear NN with a linear CNN, $\tilde{h}(u) = \tilde{A}(\theta_3)u + \tilde{F}(u; \theta_2)$. Variations 2 and 3 are what we refer to as stabilized neural ODEs. In all of these cases, θ_i refers to the parameters that are trained. These parameters are trained to minimize the norm of the difference between the predicted

states, $\tilde{u}(t_i + \tau_j)$, and the known states, $u(t_i + \tau_j)$

$$J = \left\langle \sum_{j=1}^{K} ||u(t_i + \tau_j) - \tilde{u}(t_i + \tau_j)|| \right\rangle, \tag{5.4}$$

where the predicted states come from integrating $\tilde{h}$ forward in time and K are the samples of the state along the trajectory. We only predict forward to a single τ ($K = 1$) unless noted otherwise. Here we use the L_1-norm. Other norms are applicable, but in our experience, the choice between L_1 and L_2 does not appear to substantially change results.

The selection of the linear term – whether in selecting A beforehand or in how to optimize $\tilde{A}(\theta_3)$ – is an important choice when using stabilized neural ODEs. In some cases, like the trials shown here, it may be known that the linear term is diffusive or hyper-diffusive, in which case the suspected linear term can be used directly. Even if there is a suspected set of possible linear terms, each could be applied and the most accurate model selected. Alternatively, if the structure of the linear term is unimportant, but long-time model stability is important, then A can simply be a scalar damping term. However, often it may be desirable to discover the form of the linear term directly from the data, which involves training a linear term $\tilde{A}(\theta_3)$.

When training the stabilized neural ODEs there are many ways to parameterize $\tilde{A}(\theta_3)$. The simplest approach is to allow all parameters of the matrix to be trainable. For some simple cases, this may work, but we found training with this many degrees of freedom led to a poor approximation of the linear term in the examples studied here. Instead, we chose to train linear CNNs with a single filter, a stride of one, and different filter widths N. This is equivalent to learning a sparse N-diagonal matrix whose filter width determines the number of trainable parameters. We know the equations in the cases investigated here, so we set the appropriate filter widths. For cases where the linear term is unknown, but expected to act locally, then the filter width becomes another hyperparameter to search over in training. No *a priori* information is necessary to train stabilized neural ODEs, but training improves as

additional structure is applied to $\tilde{A}(\theta_3)$.

In addition to using CNNs, we constrain $\tilde{A}(\theta_3)$ to be symmetric for the VBE, and apply no constraints for the KSE. Although this symmetry exists in both systems we found that enforcing the symmetry improved results in the VBE case, but was less important for the KSE case. These constraints do not need to be applied $\tilde{A}(\theta_3)$ to use this method. We chose to limit the constraints applied to the stabilized neural ODEs because the underlying systems possess nonlinear stability in the form of an inertial manifold. We took this approach to show that the model can learn this property from the data when a linear term is trained in the neural ODE. An alternative future direction is explicitly enforcing the existence of an inertial manifold. This could be done by enforcing a spectral gap condition, which guarantees the existence of an inertial manifold [241]. The spectral gap condition is

$$\lambda_{N+1} - \lambda_N > 2L \tag{5.5}$$

where λ_i is the i^{th} eigenvalue of A, sorted in increasing order, and L is the Lipschitz constant of F. In order to satisfy this condition, a recently developed class of NNs called Recurrent Equilibrium Networks could be used, which guarantee an upper bound on L [177].

For learning the parameters of these NNs, the gradient of this loss with respect to all the parameters, θ_i, must be calculated. This gradient can either be calculated by backpropagating through the solver with automatic differentiation or by solving an adjoint problem backwards in time [24]. When backpropagating all of the data must be stored at each time-step, which becomes memory intensive when the prediction horizon becomes large. For our trials we do not consider training data spaced apart far enough in time where this becomes a problem, and empirically we found using the adjoint method took longer, so we use backpropagation for training.

In addition to stabilizing the dynamics, which we show in the following section, the addition of a linear term leads to an interesting ROM framework, i.e., the stabilized neural ODE framework implies a natural basis for ROM based on the eigenvectors of the linear

term. This "flips" the traditional ROM approach in that we first discover dynamics and then use this dynamical system with classical methods for ROM. In particular, we apply standard Galerkin and nonlinear Galerkin approaches for ROM. These methods require that the ODE in Eq. 5.2 is known, but because we learn this equation from data we can directly apply these methods without knowing the equation *a priori*.

Specifically, we perform an eigendecomposition on the linear term, which we learn from a CNN, to find $\tilde{A}\mathrm{v}_i = \lambda_i \mathrm{v}_i$ and then we project the model approximation of Eq. 5.2 onto the leading eigenvectors, giving the *resolved* dynamics

$$\frac{\mathrm{d}p}{\mathrm{d}t} = \tilde{A}p + P\tilde{F}(p+q), \tag{5.6}$$

and the trailing eigenvectors, giving the *unresolved* dynamics

$$\frac{\mathrm{d}q}{\mathrm{d}t} = \tilde{A}q + Q\tilde{F}(p+q), \tag{5.7}$$

Here $p = Pu$ is the projection onto the leading eigenvectors (resolved dynamics) and $q = Qu$ is the projection onto the trailing eigenvectors (unresolved dynamics). For the systems we consider, $\tilde{A}$ is symmetric, which results in real eigenvalues and orthogonal eigenvectors. This means the projections can be written $P = V_p V_p^T$ and $Q = V_q V_q^T$, where $V_p = [\mathrm{v}_0, \ldots, \mathrm{v}_p]$ and $V_q = [\mathrm{v}_{p+1}, \ldots, \mathrm{v}_d]$. The classic way to sort the eigenvectors is in the order of decreasing real part of the eigenvalues [138]. In section 5.3.2 we study the best approach to organizing the eigenvectors.

With Eq. 5.6 and Eq. 5.7 ROM can be performed by either assuming $q = 0$, which is the standard Galerkin approach, or by assuming $\mathrm{d}q/\mathrm{d}t = 0$, which is the nonlinear Galerkin approach [103, 138]. With the standard Galerkin approach Eq. 5.6 can be solved forward in time without any additional equations. With the nonlinear Galerkin approach q is approximated iteratively by

$$q_{i+1} = \tilde{A}^{-1} Q\tilde{F}(p+q_i), \tag{5.8}$$

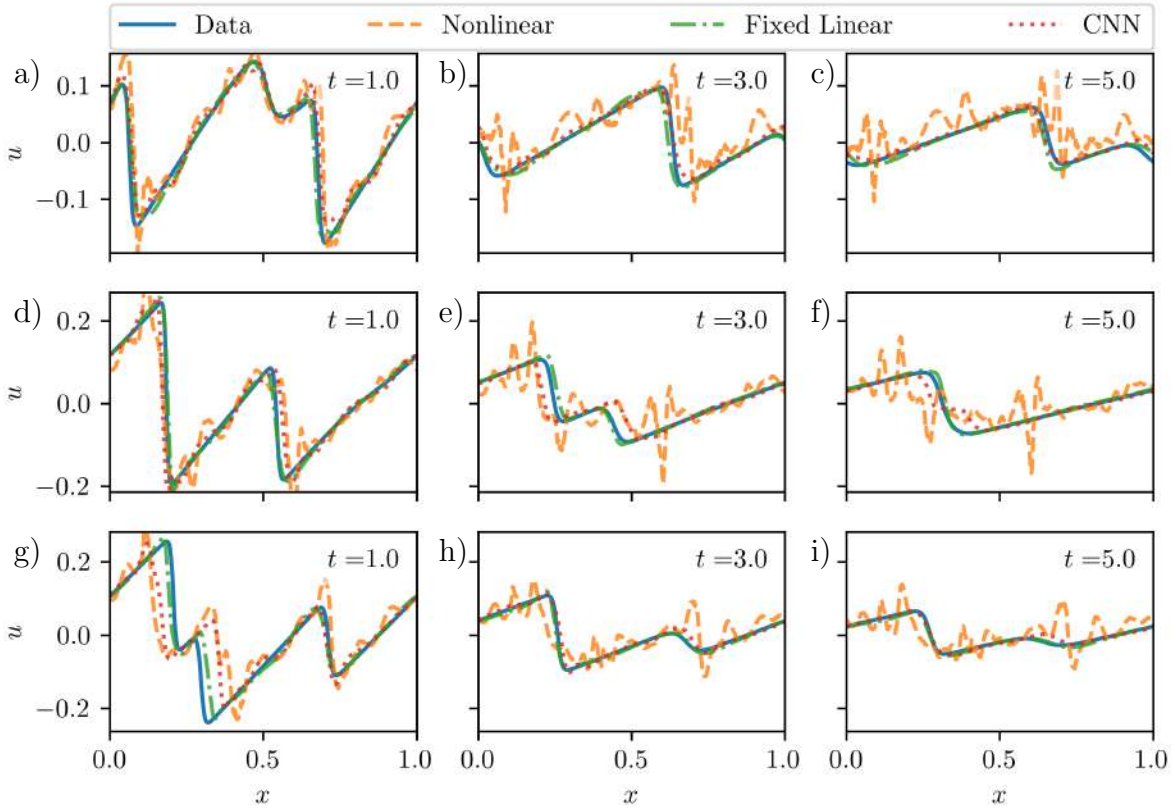


Figure 5.1: Snapshots of VBE trajectories of true system and model predictions for different initial conditions. The rows indicate different initial conditions and the columns indicate solution fields at $t = 1$, 3, and 5, respectively.

with an initial guess of $q_0 = 0$. In our trials, we only use one iteration to calculate q_1, because additional iterations had little effect on the results. Nonlinear Galerkin is more accurate than Galerkin, but is computationally more expensive due to solving for q at every timestep. Another alternative is postprocessing Galerkin where p is solved forward in time using standard Galerkin and q is approximated for all p at the end with Eq. 5.8 [63].

5.3 Results

In the following sections, we demonstrate the importance of constructing neural ODEs with a linear and a nonlinear term on the VBE and the KSE. We examine the VBE to highlight the ability of this method to accurately capture the shock behavior seen at low viscosity and to accurately capture the energy spectrum. Then, we consider the KSE to show how our

approach is needed to keep the long-time trajectories of a chaotic dynamical system on the attractor. This then leads to a demonstration of reduced order modeling via Galerkin and nonlinear Galerkin using the eigenvectors of the linear term we discover. In all cases, results are shown for test data that was not used for training any of the models. Neural ODEs are trained using PyTorch [161] with code modified from Chen et al. [24].

5.3.1 Viscous Burgers equation

In this section, the data comes from numerical solutions of the 1D viscous Burgers equation

$$\frac{\partial v}{\partial t} = -v\frac{\partial v}{\partial x} + \nu\frac{\partial^2 v}{\partial x^2}, \tag{5.9}$$

in a domain of unit length ($L = 1$) with periodic boundary conditions, with viscosity $\nu = 8 \cdot 10^{-4}$. This parameter set results in what is known as Burgers turbulence or *Burgulence* [140]. We solve this equation using a pseudospectral Runge-Kutta Crank-Nicolson scheme as described in [184]. The state u is then represented in $\mathbb{R}^{512}$ by v sampled at equidistant points in the domain. We construct 1000 random initial conditions for training the models and 100 random initial conditions for testing model performance. We then evolve each initial condition over the interval $[0, 5]$ sampling every 0.05 time units. This results in 10^4 total snapshots of data used for training. All figures compare test data to model reconstruction of the test data.

When generating training and test datasets, the space of feasible initial conditions is infinite, so we must first limit this region. Here, we chose to limit the random initial conditions such that the ensemble-averaged energy spectrum

$$E(k,t) = \left\langle \frac{1}{2}|\hat{u}(k,t)|^2 \right\rangle \tag{5.10}$$

at $t = 0$ satisfies $\int E(k,0)\,dk = 0.5L/(2\pi)$. In Eq. 5.10 k is the wavenumber and the Fourier transform is $\hat{u} = \mathcal{F}(u)$. This energy condition is satisfied by selecting the initial conditions

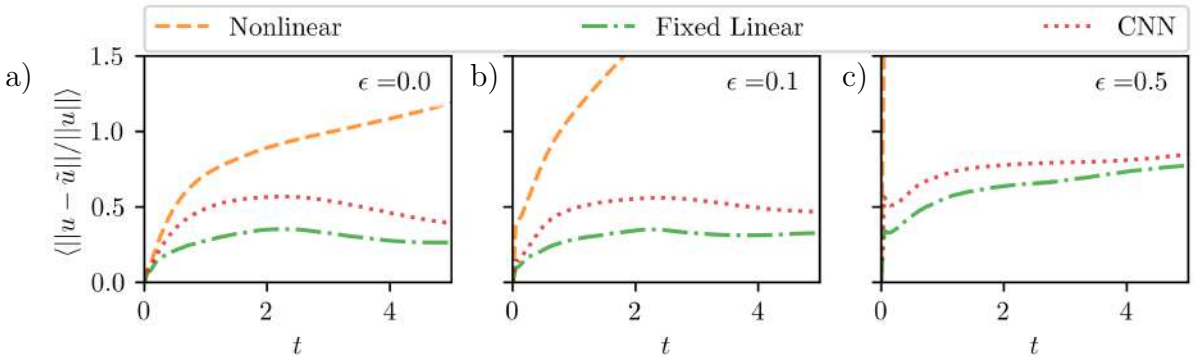


Figure 5.2: Ensemble averaged error of predictions of all models with noisy initial conditions of $\epsilon = 0.0$, 0.1, and 0.5 for (a)-(c).

Table 5.1: Architectures of NNs and matrices used in Sections 2A. "Shape" indicates the dimension of each layer, "Activation" the corresponding activation functions, "Learning Rate" is the learning rate the NN switches to evenly over the training period, and "Weight Init" is the distribution used to initialize the weights of the NNs. "relu" is the rectified linear unit. $\tilde{F}_1$ is the NN trained with the true A and $\tilde{F}_2$ is the NN trained with $\tilde{A}$.

Function	Shape	Activation	Learning Rate	Weight Init
$\tilde{f}$	512/200/200/200/512	relu/relu/relu/linear	$[10^{-3}, 10^{-4}, 10^{-5}]$	$\mathcal{N}(0, 10^{-2})$
$\tilde{F}_1$	512/200/200/200/512	relu/relu/relu/linear	$[10^{-3}, 10^{-4}, 10^{-5}]$	$\mathcal{N}(0, 10^{-2})$
$\tilde{F}_2$	512/200/200/200/512	relu/relu/relu/linear	$[10^{-3}, 10^{-4}]$	$\mathcal{N}(0, 10^{-2})$
$\tilde{A}$	512/512	linear	$[10^{0}, 10^{-1}, 10^{-2}]$	$\mathcal{N}(0, 10^{4})$

according to

$$\hat{u}(k,0) = \sqrt{2E_0(k)}\left(\cos(2\pi\Psi(k)) - \sin(2\pi\Psi(k))\right), \tag{5.11}$$

with $E_0(k) = Ak^4 \exp\left(-(k/k_0)^2\right)$, and $\Psi(k)$ coming from a uniform distribution between 0 and 1 for each wavenumber ($\Psi(k) = \mathcal{U}(0,1)$). The constants A and k_0 are the same as in [140]. Selecting the initial conditions in this manner results in training and testing datasets contained within a region of different initial energies. By selecting a region of initial conditions with similar energies we densely sample trajectories allowing the neural ODEs to learn similar dynamics. We later show that the stabilized neural ODEs can successfully generalize to "noisy" initial conditions outside of the range of the training data.

With this data, we train neural ODEs for the three cases listed above: 1) $\tilde{h}(u) = \tilde{f}(u;\theta_1)$, 2) $\tilde{h}(u) = Au + \tilde{F}_1(u;\theta_2)$, and 3) $\tilde{h}(u) = \tilde{A}(\theta_3)u + \tilde{F}_2(u;\theta_2)$, which we will denote as nonlinear,

fixed linear, and CNN (we replace $\tilde{F}$ with $\tilde{F}_i$ to denote the nonlinear term for the different approaches). We train these models for 10^4 epochs at which point the error in the loss (Eq. 5.4) stops decreasing. The architectures and training parameters for each of these NNs are presented in Table 5.1. Hyperparameter tuning was performed manually by training many NNs and selecting the NNs with the best short-time tracking performance. First, hyperparameter tuning was done in the fixed linear case by varying the width of the layers, the number of layers, and the activation functions for $\tilde{F}$. Then, these hyperparameters were used for $\tilde{F}$ in the CNN case, and the learning rate and weight initialization were tuned for the linear term. The linear term in particular showed sensitivity to hyperparameter selection. When the initialization or the learning rates were too small the weights of the linear term would remain small and the predictions were poor. Finding a systematic way to avoid this should be investigated further in future work.

We construct $\tilde{f}$ and $\tilde{F}$ from fully connected NNs, and $\tilde{A}$ from a linear CNN with stride one, one filter, and a filter width of 3. This is equivalent to learning a sparse tridiagonal matrix. In the case of the VBE, we enforce symmetry in $\tilde{A}$ by training a CNN $B(\theta_3)$ and letting $\tilde{A} = B(\theta_3) + B(\theta_3)^T$. Without applying the symmetry we found the results were much less accurate in terms of the approximation of the linear term and the tracking error. With this tridiagonal structure, the optimal filter for approximating the derivative is $[210, -420, 210]$ (this is the approximation of the second derivative with central differencing), and the filter learned after training is $[45, -90, 45]$. Although the scale of the filter is off, we will see the results with this filter and the true filter match well.

In Fig. 5.1 we show the performance of the three neural ODE approaches on different initial conditions of the VBE. For all three initial conditions, we see that stabilized neural ODE approaches (fixed linear and CNN) result in good agreement between the true solution and the model prediction. When, instead we take the nonlinear approach high-wavenumber behavior appears in the solution and persists at long times resulting in a poor prediction.

The behavior just described becomes even more pronounced when considering the relative

error averaged over initial conditions, shown in Fig. 5.2a. Here, all methods match at very short times. The fixed linear approach performs the best followed by the CNN approach, and then by the nonlinear approach. For the two stabilized neural ODE methods the errors level off, or slightly go down, at long times indicating that trajectories tend to separate somewhat and then decay in a similar fashion causing them to stay the same relative distance apart. However, this is not the case for the nonlinear method where the high-wavenumber behavior at long times continues to push the trajectory further away increasing the relative error.

We also consider the ensemble-averaged energy spectrum defined in Eq. 5.10. Figures 5.3a-5.3c show the energy spectrum at various times. At early times, when the shock develops, there is an increase in the tail of the energy spectrum, and we see the development of a k^{-2} scaling. At $t = 1$ all methods capture most of the spectrum with errors appearing at high-wavenumbers. Then, at $t \gtrsim 2$, both the stabilized methods match the true energy spectrum over full range of wavenumbers, while the high-wavenumber behavior of the nonlinear model deviates drastically as time increases. Capturing the dynamics properly at all scales requires a linear term to damp out high wavenumbers.

The explicit linear term in our approach also contributes to the robustness and generalizability of the neural ODE setup by accurately predicting the evolution of noisy initial conditions outside of the range of the training data. The purpose of considering these new initial conditions is to investigate the ability of the stabilized neural ODEs to handle perturbations away from the manifold that the training trajectories lie on. We show this by comparing the evolution of noisy initial conditions between the true system and the models. The initial conditions are generated by adding random Gaussian noise of the form $\mathcal{N}(0, \epsilon^2)$ to each grid location for the 100 original test initial conditions. Then those noisy initial conditions are evolved forward with the true equations and the learned models for comparison. Fig. 5.4a shows examples of an initial condition with different levels of noise, and Fig. 5.4b shows ensemble-averaged energy spectrum for the different datasets. The initial conditions with noise added contain much more energy in the higher wavenumbers than the original

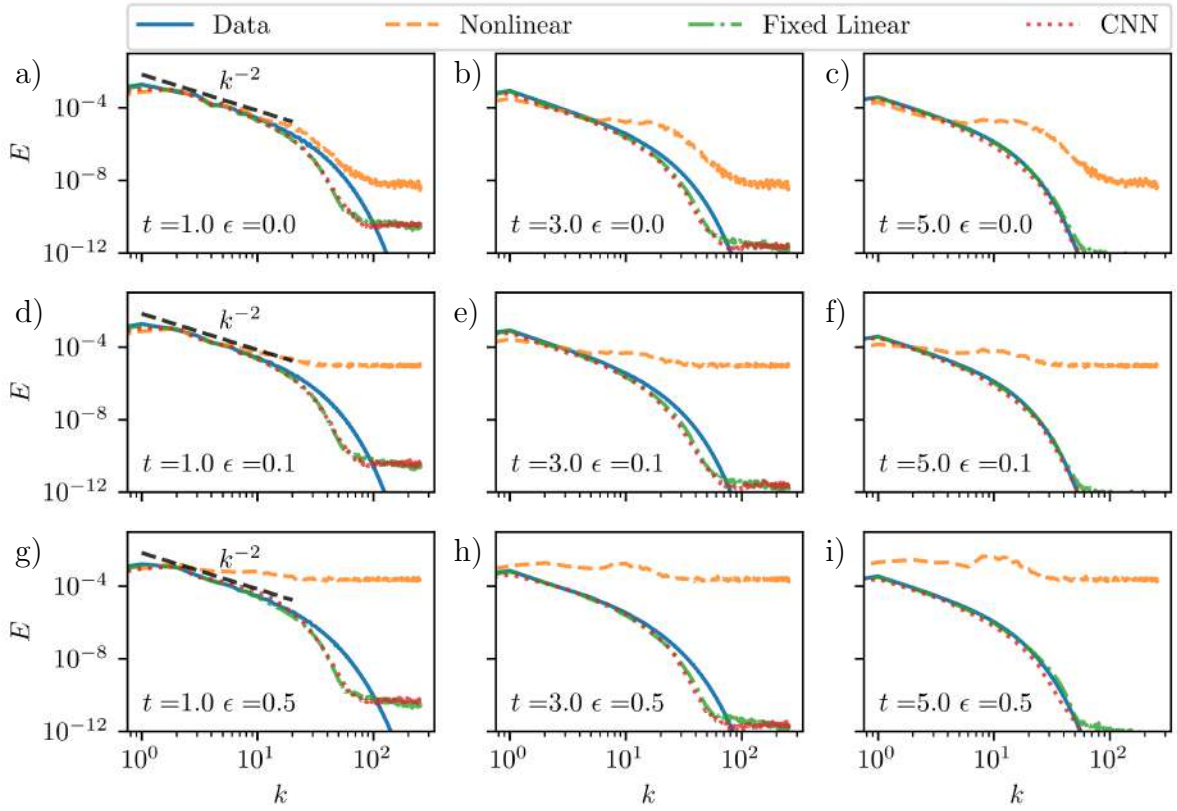


Figure 5.3: Ensemble-averaged energy spectrum of the true system and the model predictions at different times and noise on the initial conditions. (a)-(c),(d)-(f), and (g)-(i) are with noise $\epsilon = 0.0$, 0.1, and 0.5, at times $t = 1$, 3, and 5, respectively.

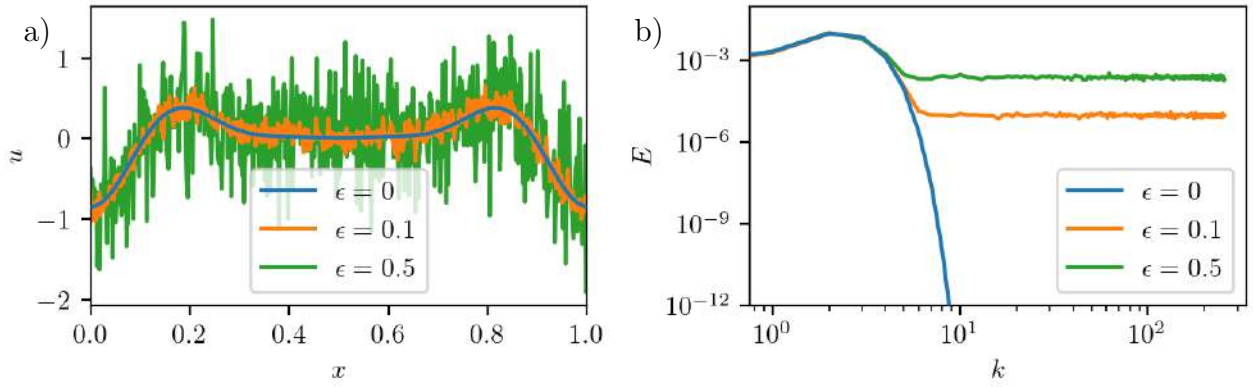


Figure 5.4: Comparison of training initial conditions to noisy initial conditions. (a) compares example initial conditions and (b) compares the ensemble-averaged energy spectrum.

data.

Figures 5.2b and 5.2c show the relative error in predictions for various levels of noise added to the initial condition. In all these cases the error in the nonlinear cases rapidly increases, while the stabilized neural ODE cases show an initial rise in the error and then a leveling off in error as the trajectories decay at a similar rate. This result matches the trend of the trials with no noise, with slightly larger error. We also show the ensemble-averaged energy spectrum at different times for the different levels of noise in Figs. 5.3d-5.3f and Figs. 5.3g-5.3i. The strictly nonlinear model does a poor job of capturing the spectrum at any wavenumber, while the stabilized neural ODEs accurately capture the spectrum as they did for the initial conditions with no noise.

These results highlight that the stabilized neural ODEs accurately track and reproduce statistics of trajectories from noisy initial conditions that are much different from the training data. The ability of the stabilized neural ODEs to generalize outside of the training dataset is due to the fact that the linear term is independent of the state and it is the damping due to this linear term that dominates the initial dynamics of the true trajectories. So even though there is likely a poor approximation of the nonlinear term ($F(u)$) at short times, the dynamics are dominated by the linear term at these times. Then, after this decay, the state is pushed back towards a region of space similar to the training data, and $F(u)$ is accurately approximated.

5.3.2 Kuramoto-Sivashinsky equation

In the previous section, we showed that our setup captures the shock dynamics and transient decay exhibited by the VBE. In that case, all solutions decay to zero at long times. Now we examine the KSE, where the dynamics are chaotic and collapse onto a low-dimensional inertial manifold at long times [60, 107, 207, 241]. A useful model for the KSE should keep long-time trajectories on the attractor and provide a means for reduced-order modeling to capitalize on the low dimensionality of the manifold on which the data lies.

Table 5.2: Neural ODE details for 5.3.2. Labels are the same as in Table 5.1. "Sig" is the sigmoid activation

Function	Shape	Activation	Learning Rate	Weight Init
$\tilde{f}$	64/200/200/200/64	sig/sig/sig/linear	$[10^{-3}, 10^{-4}]$	$\mathcal{N}(0, 10^{-2})$
$\tilde{F}_1$	64/200/200/200/64	sig/sig/sig/linear	$[10^{-3}, 10^{-4}]$	$\mathcal{N}(0, 10^{-2})$
$\tilde{F}_2$	64/200/200/200/64	sig/sig/sig/linear	$[10^{-3}, 10^{-4}]$	$\mathcal{N}(0, 10^{-2})$
$\tilde{A}$	64/64	linear	$[10^{0}, 10^{-1}, 10^{-2}]$	$\mathcal{U}(-\sqrt{1/3}, \sqrt{1/3})$

We numerically solve the 1D KSE,

$$\frac{\partial v}{\partial t} = -v\frac{\partial v}{\partial x} - \frac{\partial^2 v}{\partial x^2} - \frac{\partial^4 v}{\partial x^4}, \tag{5.12}$$

with periodic boundary conditions on a domain of size $L = 22$. Solutions are found by performing a Galerkin projection onto Fourier modes and using exponential time differencing to evolve the ODE forward in time [108]. After solving in Fourier space, we then transform the data back to physical space for training the neural ODEs. Code for solving the KSE is available in Cvitanović et al. [41]. The data used is a single trajectory that has collapsed onto the inertial manifold evolved over $t = [0, 10^5]$ sampled every 0.25 time units resulting in $4{\cdot}10^5$ snapshots of data. The first 80% of this data is used for training and the remaining 20% is used as test data. For comparison, in [220] 10^5 snapshots of data were used for training reservoir computers and RNNs. We chose a large enough amount of data so that the data would not hinder model performance. However, we also ran a sweep over the amount of data used in the CNN stabilized neural ODE models and found that statistics were captured well with as little as 10^4 snapshots of training data.

We train neural ODEs for $4 \cdot 10^4$ epochs, and consider the same three neural ODE structures as before – nonlinear, fixed linear, and CNN. Table 5.2 contains details on the architectures and training parameters used for each of these NNs. Small modifications were made in the architectures due to the different dynamics of the KSE. Hyperparameters were tuned manually using the same approach as with the VBE. As before, $\tilde{f}$ and $\tilde{F}$ are fully connected NN and $\tilde{A}$ is a linear CNN. For $\tilde{A}$ we selected a wider filter of width 5 because

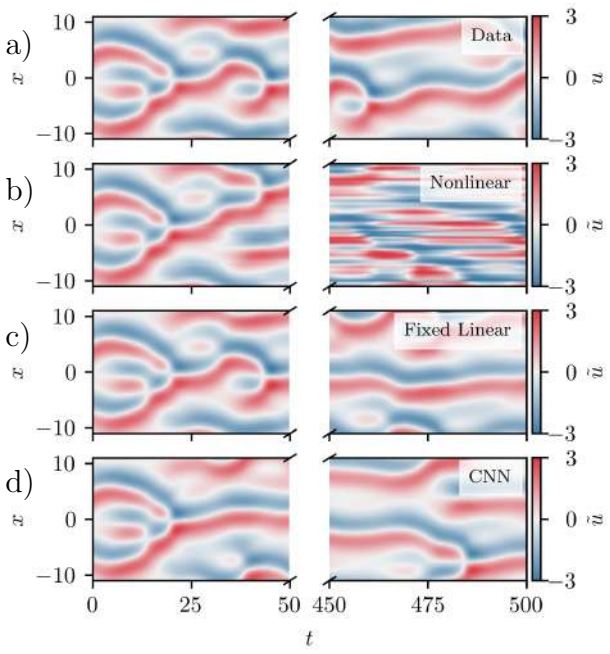


Figure 5.5: Predictions of trajectories using different models. (a) is the true trajectory. (b) is the predicted trajectory with the nonlinear model. (c) is the predicted trajectory with the fixed linear model. (d) is the predicted trajectory with the CNN model.

higher-order gradients need to be approximated. In general, the linear term is unknown, so the filter width can be viewed as an additional tuning parameter that is similar to selecting the stencil size for a finite difference approximation. For our system, and filter width, the optimal filter parameters are $[-72, 278, -413, 278, -72]$ (this is central differencing of the diffusive and hyperdiffusive terms), while the parameters learned for the CNN were $[-0.7, 1.8, -2.7, 1.8, -0.7]$. Here we also ran trials where we forced symmetry in the filter, but we found the training procedure resulted in the symmetry without the need to enforce it. Despite the magnitude of the parameters being much smaller, we will show the CNN successfully damps out high wavenumbers.

Figure 5.5 compares a true trajectory to predictions from the three methods. The left side shows short-times, where model predictions should match the true trajectory. The right side shows long times, where the model predictions should have statistics that agree with the true trajectory. In Fig. 5.5b the nonlinear method prediction is shown. At early times the predicted trajectory matches the true trajectory, but at long-times erroneous high-

wavenumber behavior appears and the trajectory has moved far away from the attractor. This result agrees with what was found in [129] and shows for chaotic systems even model error can push trajectories off the manifold with the standard neural ODE approach resulting in incorrect long-time predictions. Predictions from the stabilized neural ODEs appear in Figs. 5.5c and 5.5d for the fixed linear and CNN methods, respectively. In the linear cases, there is good agreement at short times, with the fixed linear method tracking slightly better, and, more importantly, at long-times, the trajectories stay on the attractor.

The advantage stabilized neural ODEs have over the standard neural ODE, for the KSE, is the prediction at long times. In Fig. 5.6 we show the true joint PDF of the first (u_x) and second, (u_{xx}) derivatives of the state plotted for every spatiotemporal point of a trajectory evolved forward for 10^4 time units. Figure 5.6b therein, which corresponds to the nonlinear ODE prediction, shows the probability of finding a point in the nonlinear OSE prediction with the correct first and second derivatives is nearly zero. Both stabilized neural ODEs, shown in Figs. 5.6c and 5.6d, accurately match the true joint PDF.

Similar to the VBE, the presence of a linear term makes the KSE models highly robust to perturbations off the attractor. The robustness to perturbations of states off the attractor is particularly important in this case because, as shown in Fig. 5.5, even without the addition of noise the strictly nonlinear model will push trajectories far from the attractor. Figure 5.7 shows the short-time evolution of predictions from the CNN and the standard neural ODE approach with varying levels of noise. Here the initial condition is perturbed with Gaussian noise of magnitude $\mathcal{N}(0, \epsilon^2)$ at each grid location. In Figs. 5.7a and 5.7c we set $\epsilon = 0.5$ and 10 for the CNN approach. Despite the magnitude of the CNN linear term being much smaller than the true linear term, the CNN still rapidly damps high-wavenumbers like the true system. Similarly, in Figs. 5.7b and 5.7d we set $\epsilon = 0.5$ and 10 for the standard neural ODE approach. Without a damping linear term the noise in the initial condition grows in time.

The previous results highlight the method successfully pushes perturbed trajectories back

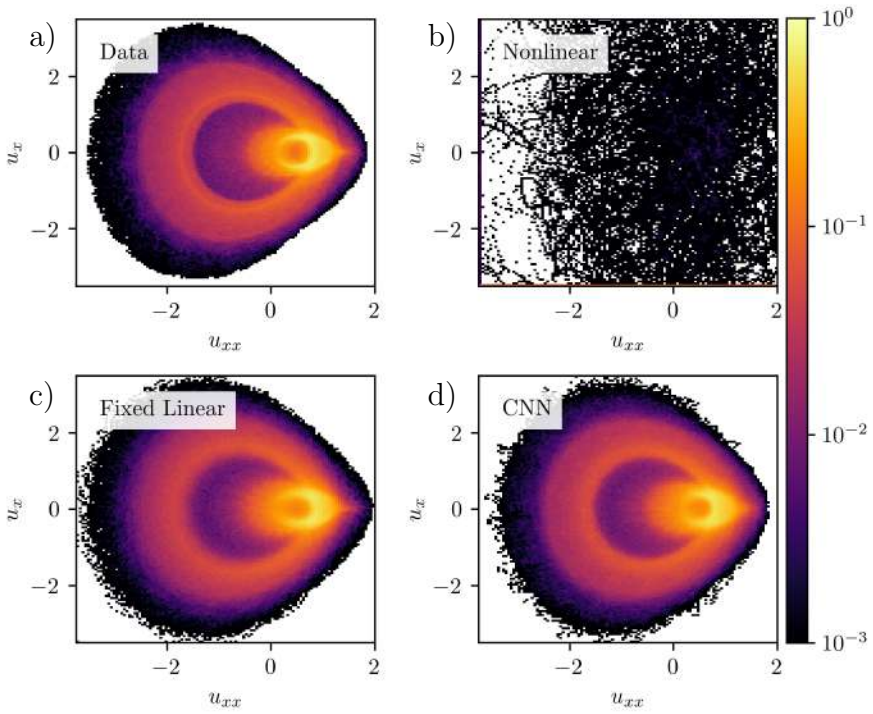


Figure 5.6: (a) is the true joint PDF at $L = 22$. (b)-(d) are the joint PDFs for the nonlinear, fixed linear, and CNN models, respectively.

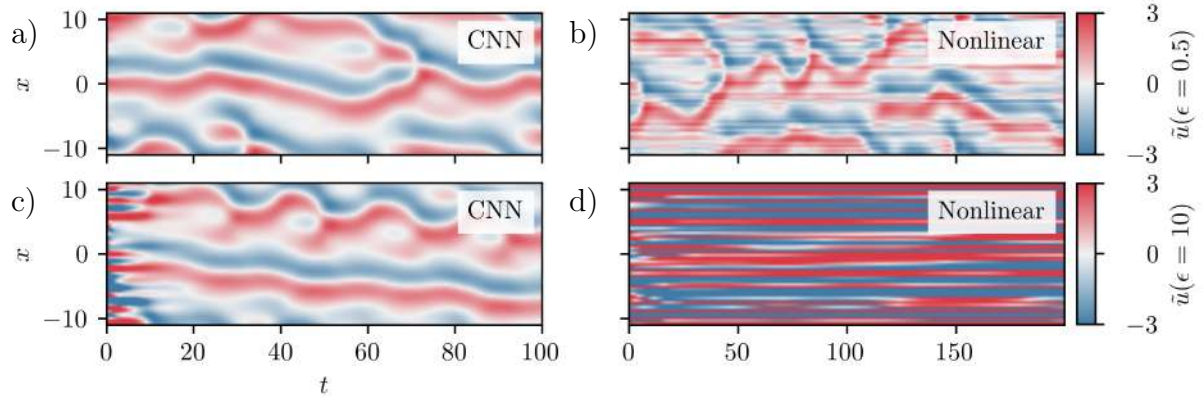


Figure 5.7: (a) and (c) are trajectories of the CNN model for initial conditions with $\epsilon = 0.5$ and 10. (b) and (d) are trajectories of the standard neural ODE model for initial conditions with $\epsilon = 0.5$ and 10.

to the manifold with clean training data. Next, we investigate whether the CNN-stabilized neural ODEs work with noisy training data. Figure 5.8 shows an example snapshot for three levels of noise applied, and the joint PDFs for running NNs trained with this noisy data. We apply noise to all snapshots of training data with noise at all grid locations in the same way described above. The initial condition used in generating the joint PDFs is free of noise so that the PDFs reflect how well the underlying dynamics were captured.

For training the NNs at the different levels of noise we used multiple snapshots, still separated every 0.25 time units, along a trajectory in the computation of the loss (Eq. 5.4) which we found to improve training in these cases. Specifically, we used $K = 1, 3$, and 5 for $\epsilon = 0.1, 0.3$, and 0.5. In these cases, we trained 10 CNN-stabilized neural ODEs and show the most accurate joint PDFs. For the smallest amount of noise, the joint PDF is very well captured, and as more noise is added the joint PDFs become more compact, with the $\epsilon = 0.5$ eventually landing on a fixed point. This shows that a stabilizing linear term can be discovered even from noisy training data.

Although the direct application of stabilized neural ODEs to training data can work for the KSE, we found in the case of the VBE equation that it only worked with very little noise. We believe this is due to noise being added at every gridpoint on a fine grid and due to the stiff nature of the VBE. In either case, the structure is not necessarily designed to deal with noisy training data. Instead, when dealing with noisy data a better approach may be to preprocess the training data – as is frequently done. For example, in the PDE-FIND method [182] described in the introduction a polynomial interpolation was used to preprocess the noisy data.

The other useful aspect of the stabilized neural ODE architecture is it lends itself naturally to reduced-order modeling (ROM). As mentioned in section 5.2, a standard approach to ROM involves projecting the true ODE (Eq. 5.2) onto a basis given by the leading d_p eigenvectors of A. This is the resolved dynamics in Eq. 5.6. Then, the unresolved dynamics, Eq. 5.7 can be approximated by setting $q = 0$, which is the standard Galerkin approach, or by using Eq.

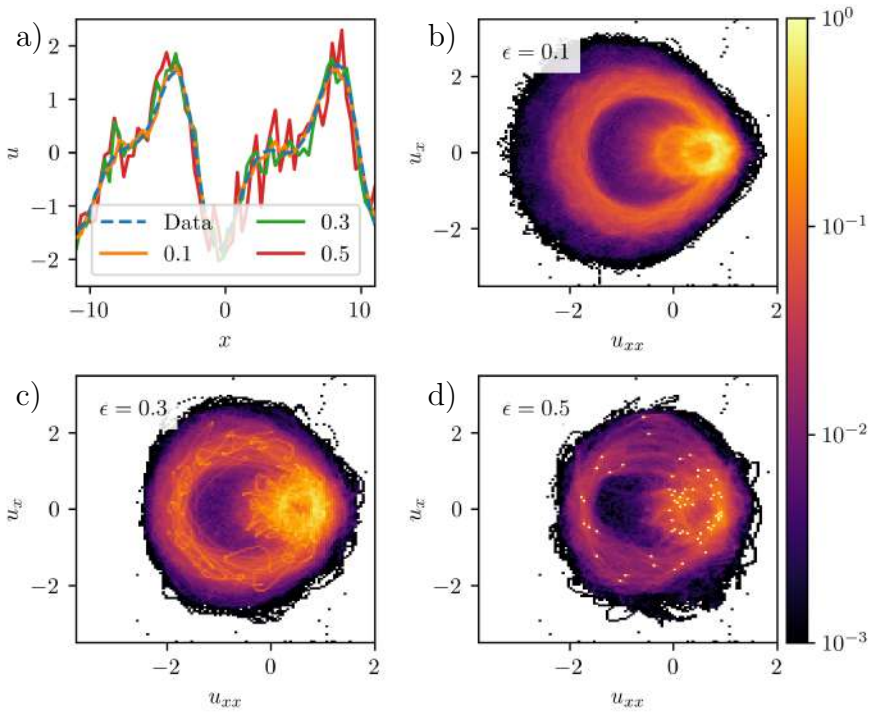


Figure 5.8: (a) shows three example snapshots of the training data with different levels of noise (ϵ). (b)-(d) are joint PDFs of the CNN model for $\epsilon = 0.1, 0.3$ and 0.5, respectively.

5.8 to solve for q, which is the nonlinear Galerkin approach. Here we show we can successfully apply both these ROM methods to our CNN neural ODE model that we find strictly from data, whereas typically Eq. 5.2 must be known. We first compare the projection operator of the CNN neural ODE to the true projection operator, and then we compare the long-time statistical performance of the ROM when varying dimension d_p.

For these ROM approaches the eigenvector basis is typically selected from largest to smallest eigenvalues. However, selecting the eigenvectors in this fashion does not necessarily result in the best agreement with the underlying assumption of this ROM – that $\mathrm{d}q/\mathrm{d}t = 0$. To investigate this assumption, we consider the dynamics from projecting the equation onto each eigenvector $d\mathrm{v}_i^T u/dt = \dot{\mathrm{v}}_i$. The ROM assumption is reasonable when, for some p, $\dot{\mathrm{v}}_i \approx 0$ for all $i = p+1, \ldots, d$.

In Fig. 5.9 we show the variance in $\dot{\mathrm{v}}_i$ sorted from largest to smallest eigenvalue λ_i. Each eigenvector can be considered separately because the linear term is symmetric in both the

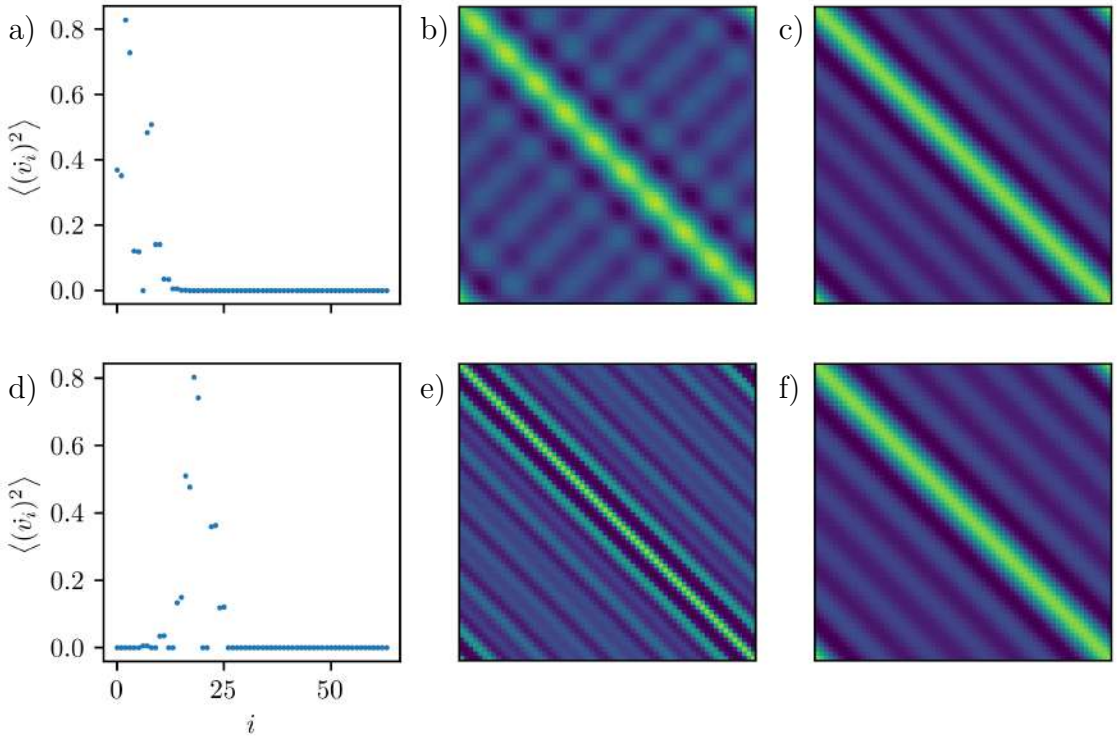


Figure 5.9: (a) is the variance of the true ODE projected onto each eigenvector of the true linear term sorted from largest to smallest eigenvalue. (b) and (c) are the projection operators from the eigenvectors of the true linear term sorted by the eigenvalues and the variance in (a), respectively. (d) is the variance of the CNN ODE projected onto each eigenvector of the CNN linear term sorted from largest to smallest eigenvalue. (e) and (f) are the projection operators from the eigenvectors of the CNN linear term sorted by the eigenvalues and the variance in (c), respectively. The projection operators are for 10 eigenvectors.

true system and from the CNN approach resulting in an orthogonal basis. Figure 5.9a shows the eigenvalues of the true linear term and Fig. 5.9d shows the eigenvalues of the CNN linear term. In both cases sorting by eigenvalue does not lead to a monotonic decrease in $\dot{\mathrm{v}}_i$. We show the projection matrices when sorting from largest to smallest eigenvalue for the true linear term in Fig. 5.9b, and for the CNN linear term in Fig. 5.9e. There is a substantial difference between the matrices when sorting the eigenvectors according to the eigenvalue, and the CNN projection matrix, in particular, is not useful for ROM.

In general, when only data is available there is no immediate way to compute $\dot{\mathrm{v}}_i$, however, because we learn the ODE directly from data computing $\dot{\mathrm{v}}_i$ after the fact is trivial. This motivates another reason why our stabilized neural ODE approach naturally lends itself to ROM. Instead of organizing according to eigenvalues, we can, instead, sort according to the variance of $\dot{\mathrm{v}}_i$, which more accurately reflects the assumption taken in nonlinear Galerkin. In Fig. 5.9c we show the projection matrix from the true linear term with this sorting and in Fig. 5.9f we show the projection matrix from the CNN linear term. The two projection matrices match nearly exactly.

With the projection selected according to this variance criterion, we show the standard Galerkin and nonlinear Galerkin approaches performance in Fig. 5.10. For this comparison we consider both the CNN neural ODE, sorted according to the variance criterion, and the true ODE, sorted by the standard eigenvalue criterion. We compare ROM performance with the Kullback-Leibler (KL) divergence of the u_x vs u_{xx} joint PDF

$$D_{KL}(\tilde{P}||P) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \tilde{P}(u_x, u_{xx}) \ln \frac{\tilde{P}(u_x, u_{xx})}{P(u_x, u_{xx})}\, du_x du_{xx}. \tag{5.13}$$

In this equation, P is the true joint PDF, and $\tilde{P}$ is the ROM joint PDF. When $P = 0$ or $\tilde{P} = 0$ we adopt the convention of setting the term in the integral to 0 as was done in [21].

In Fig. 5.10 we vary the dimension d_p (i.e. the number of modes on which we project) and compute the KL divergence between the true joint PDF and the ROM joint PDFs,

generated the same way as in Fig. 5.6. In the case of the Galerkin ROM, we see that the CNN approach does a better job than even the true Galerkin approach at reconstructing the joint PDF. In the case of nonlinear Galerkin, the CNN approach shows more error than the true nonlinear Galerkin approach but performs better than the standard Galerkin approaches. For all ROMs, we see excellent agreement around a dimension of $d_p \sim 18$, as can be seen in the inset, above which there is little change in the KL divergence and the ROM agree well with the full system. These results show that by using classical ROM approaches on our data-driven stabilized neural ODEs we can achieve excellent model performance at far fewer dimensions than the underlying system.

5.4 Summary

In this chapter, we showed that stabilizing neural ODEs with a linear and a nonlinear term is an effective way to improve the predictive capabilities of these models. By implementing this structure we find the models are more robust to noise, perform better in capturing statistics and provide a method for selecting a natural basis, different from conventional approaches, for reduced-order models. We test this setup first on the VBE, where the addition of the linear term results in a model that effectively captures the shock dynamics and the energy spectrum, along with generalizing to noisy initial conditions. We then test it on the KSE, where we find the linear term is necessary to keep trajectories on the attractor, and again the linear term provides excellent robustness to noise. With the KSE we further show our stabilized neural ODE method lends itself naturally to ROM, and that, with proper sorting of the eigenvectors of the linear term, we can accurately recreate the joint PDF of u_x and u_{xx} with far fewer dimensions than required for the full simulation.

The future directions of this research include applying the method to more complex problems, where the existence of an inertial manifold may be unknown, and investigating the importance of adding explicit constraints on the NNs for stability. In particular, we

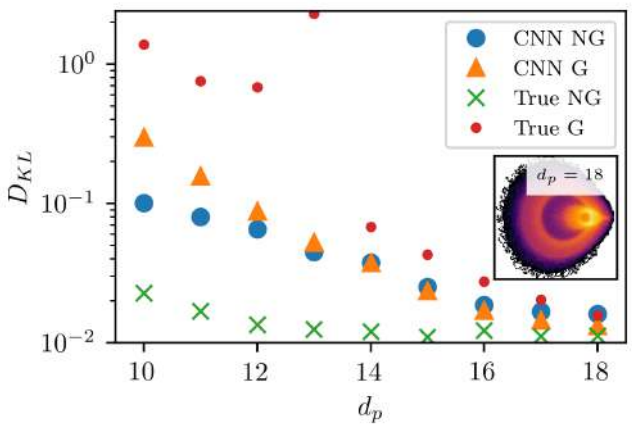


Figure 5.10: KL divergence of the joint PDFs generated with either standard Galerkin "G" or nonlinear Galerkin "NG" when using the CNN neural ODE "CNN" or the true equations "True". The inset is the CNN Galerkin ROM at a dimension $d_p = 18$.

are immediately interested in applying this method to canonical fluid flow problems (e.g. Couette flow). This modeling could either be directly data-driven, as in this paper, or for the closure of low-resolution physics-based models. Regardless, with problems of increasing complexity, it may require the use of adding additional constraints for stability. This can be done by enforcing that the linear term has a specific eigenvalue spectrum, or by tuning the Lipschitz constant of the nonlinearity. For example, we could enforce the spectral gap condition, which would guarantee the existence of an inertial manifold.

Chapter 6

Low-dimensional model of Couette flow [1]

We no explore applying the modeling approaches of the previous chapters to turbulent plane Couette flow. Because the Navier-Stokes equations are dissipative, the long-time dynamics of a flow in state space are expected to collapse onto a manifold whose dimension may be much lower than the dimension required for a resolved simulation. On this manifold, the state of the system can be exactly described in a coordinate system parameterizing the manifold. Describing the system in this low-dimensional coordinate system allows for much faster simulations and analysis. We show, for turbulent Couette flow, that this description of the dynamics is possible using a data-driven manifold dynamics modeling method. This approach consists of an autoencoder to find a low-dimensional manifold coordinate system and a set of ordinary differential equations defined by a neural network. Specifically, we apply this method to minimal flow unit turbulent plane Couette flow at $\mathrm{Re} = 400$, where a fully resolved simulation requires $\mathcal{O}(10^5)$ degrees of freedom. Using only data from this simulation we build models with fewer than 20 degrees of freedom that quantitatively capture key characteristics of the flow, including the streak breakdown and regeneration cycle.

[1] Parts of this chapter have been adapted from the manuscript titled "Dynamics of a data-driven low-dimensional model of turbulent minimal Couette flow" by A. J. Linot and M. D. Graham on arXiv and submitted for review to the *Journal of Fluid Dynamics*.

At short times, the models track the true trajectory for multiple Lyapunov times, and, at long times, the models capture the Reynolds stress and the energy balance. For comparison, we show that the models outperform POD-Galerkin models with ∼2000 degrees of freedom. Finally, we compute unstable periodic orbits from the models. Many of these closely resemble previously computed orbits for the full system; additionally, we find nine orbits that correspond to previously unknown solutions in the full system.

6.1 Introduction

A major challenge in dealing with chaotic fluid flows, whether it be performing experiments, running simulations, or interpreting the results, is the high-dimensional nature of the state. Even for simulations in the smallest domains that sustain turbulence (a minimal flow unit (MFU)), the state dimension may be $\mathcal{O}(10^5)$ [83, 105]. However, despite this nominal high-dimensionality, the dissipative nature of turbulent flows leads to the expectation that long-time dynamics collapse onto an invariant manifold of much lower dimension than the ambient dimension [94]. By modeling the dynamics in a manifold coordinate system, simulations could be performed with a drastically lower-dimensional state representation, significantly speeding up computations. Additionally, such a low-dimensional state representation is highly useful for downstream tasks like control or design. Finding a low-dimensional – or ideally a minimal dimensional – parameterization of the manifold and an evolution equation for this parameterization are both challenges. In this work, we aim to address these challenges with a data-driven model, specifically for the task of reconstructing turbulent plane Couette flow.

The classic way to perform dimension reduction from data is to use the proper orthogonal decomposition (POD), also known by Principal Component Analysis (PCA) or Karhunen-Loève decomposition [92]. This is a linear dimension reduction technique in which the state is projected onto the set of orthogonal modes that capture the maximum variance in the data.

The POD is widely used for flow phenomena, some examples of which include: turbulent channel flow [6, 146], flat-plate boundary layers [175], and free shear jet flows [3]. Smith et al. [193] showed how to incorporate system symmetries into the POD modes, the details of which we elaborate on in Sec. 6.3.

While the POD has seen wide use and is easy to interpret, more accurate reconstruction can be achieved with nonlinear methods – a result we highlight in Sec. 6.3. Some popular methods for nonlinear dimension reduction include kernel PCA [186], diffusion maps [35], local linear embedding (LLE) [180], isometric feature mapping (Isomap) [208], and t-distributed stochastic neighbor embedding (tSNE) [89]. These methods are described in more detail in Ch. 1, and an overview of other dimension reduction methods can be found in [212]. One drawback of all of these methods, however, is that they reduce the dimension, but do not immediately provide a means to move from a low-dimensional state back to the full state. A popular dimension reduction method without these complications is the undercomplete autoencoder [90], which uses a neural network (NN) to map the input data into a lower-dimensional "latent space" and another NN to map back to the original state space. We describe this structure in more detail in Sec. 6.2. Some examples where autoencoders have been used for flow systems include flow around a cylinder [147], flow around a flat plate [151], Kolmogorov flow [157, 163], and channel flow [143]. Although we will not pursue this approach in the present work, it may be advantageous for multiple reasons to parametrize the manifold with overlapping local representations, as done in [57].

After reducing the dimension, the time evolution for the dynamics can be approximated from the equations of motion or in a completely data-driven manner. The classical method is to perform a Galerkin projection wherein the equations of motion are projected onto a set of modes (e.g. POD modes) [92]. However, in this approach, all the higher POD modes are neglected. An extension of this idea, called nonlinear Galerkin, is to assume that the time derivative of the coefficients of all of the higher modes is zero, but not the coefficients themselves [58, 77, 209]; this is essentially a quasisteady state approximation for

the higher modes. This improves the accuracy, but comes at a higher computational cost than the Galerkin method, although this can be somewhat mitigated by using a postprocessing Galerkin approach [63]. Wan et al. [228] also showed a recurrent NN (RNN) – a NN that feeds into itself – can be used to improve the nonlinear Galerkin approximation. This RNN structure depends on a history of inputs, making it non-Markovian. In addition to these linear dimension reduction approaches, an autoencoder can be used with the equations of motion in the so-called manifold Galerkin approach, which Lee and Carlberg [121] developed and applied to the viscous Burgers equation.

When the equations of motion are assumed to be unknown, and only snapshots of data are available, a number of different machine learning techniques exist to approximate the dynamics. Two of the most popular techniques are RNNs and reservoir computers. Vlachas et al. [220] showed both these structures do an excellent job of capturing the chaotic dynamics of the Lorenz-96 equation and Kuramoto-Sivashinsky equation (KSE). For fluid flows, autoencoders and RNNs (specifically long-short term memory networks (LSTM)) have been used to model flow around a cylinders [51, 86], pitching airfoils [51], bluff bodies [85], and MFU plane Poiseuille flow (PPF) [152]. Although these methods often do an excellent job of predicting chaotic dynamics, the models are not Markovian, so the dimension of the system also includes some history, and these models perform discrete timesteps. These two properties are undesirable because the underlying dynamics are Markovian and continuous in time, and modeling them differently complicates applications and interpretations of the model. In particular, we want to use the model for state space analyses such as the determination of periodic orbits, where standard tools are available for ODEs that do not easily generalize to non-Markovian dynamic models.

Due to these issues, we use neural ordinary differential equations (ODE) [24]. In neural ODEs, the right-hand-side (RHS) of an ODE is represented as a NN that is trained to reconstruct the time evolution of the data from snapshots of training data. In Linot and Graham [129] it was shown that this is an effective method for modeling the chaotic dynamics

of the KSE. Rojas et al. [179] used neural ODEs to predict the periodic dynamics of flow around a cylinder, and Portwood et al. [167] used neural ODEs to predict the kinetic energy and dissipation of decaying turbulence.

In this work, we investigate the dynamics of MFU Couette flow. The idea behind the MFU, first introduced by Jiménez and Moin [105], is to reduce the simulation domain to the smallest size that sustains turbulence, thus isolating the key components of the turbulent nonlinear dynamics. Using an MFU for Couette flow at transitional Reynolds number, Hamilton et al. [83] outlined the regeneration cycle of wall-bounded turbulence called the "self-sustaining process" (SSP), which we describe in more detail in Sec. 6.3. This system was later analyzed with covariant Lyapunov analysis by Inubushi et al. [100], who found a Lyapunov time (the inverse of the leading Lyapunov exponent) of ~ 48 time units.

Many low-dimensional models have been developed to recreate the dynamics of the SSP. The first investigation of this topic was by Waleffe [226], who developed an 8-mode model for shear flow between free-slip walls generated by a spatially sinusoidal forcing. He selected the modes based on intuition from the SSP and performed a Galerkin projection onto these modes. Moehlis et al. [145] later added an additional mode to Waleffe's model which enables modification of the mean profile by the turbulence and made some modifications to the chosen modes. In this "MFE" model, Moehlis *et al.* found exact coherent states, which we discuss below, that did not exist in the 8-mode model. In addition, Moehlis et al. [144] also used the POD modes on a domain slightly larger than the MFU to generate POD-Galerkin models. These low-dimensional models have been used as a starting point for testing data-driven models. For example, both LSTMs [198] and a Koopman operator method with nonlinear forcing [52] have been used to attempt to reconstruct the MFE model dynamics. Borrelli et al. [12] then applied these methods to PPF.

Finally, we note that a key approach to understanding complex nonlinear dynamical phenomena, such as the SSP of near-wall turbulence, is through the study of the underlying state space structure of fixed points and periodic orbits. In the turbulence literature, these

are sometimes called "exact coherent states", or ECS [76, 110]. Turbulence organizes around ECS in the sense that trajectories chaotically move between different such states. The first ECS found were fixed point solutions in PCF [150]. Following this work, Waleffe [227] was able to connect ECS of PCF and PPF to the SSP. Later, more fixed point ECS were found in MFU PCF and visualized by Gibson et al. [68]. Unlike fixed points, which cannot capture dynamic phenomena at all, periodic orbits are able to represent key aspects of turbulent dynamics such as bursting behavior. Kawahara and Kida [109] found the first two periodic orbits (POs) for MFU PCF, one of which had statistics that agreed well with the SSP. Then, Viswanath [219] found another PO and 4 new relative POs (RPOs) in this domain, and Gibson made these solutions available in [69], along with a handful of others.

In the present work, we use autoencoders and neural ODEs , in a method we call "Data-driven Manifold Dynamics" (DManD) [132], to build a ROM for turbulent MFU PCF [83]. Section 6.2 outlines the details of the DManD framework. We then describe the details of the Couette flow in Sec. 6.3.1, the results of the dimension reduction in Sec. 6.3.2, and the DManD model's reconstruction of short- and long-time statistics in Sec. 6.3.3 and Sec. 6.3.4, respectively. After showing that the models accurately reproduce these statistics, we compute RPOs for the model in Sec. 6.3.5, finding several that are similar to previously known RPOs, as well as several that seem to be new. Finally, we summarize the results in Sec. 6.4.

6.2 Framework

Here we outline our method for an "exact" DManD modeling approach. In this sense "exact" means all of the functions described allow for perfect reconstruction, but error is introduced in approximating these functions due to insufficient data, error in learning the functions, or error in evolving them forward in time. This differs from coarse-grained ROMs, which approximate the physics to generate a closed set of equations. A key component allowing

DManD to be "exact" is that we only seek to discover the evolution of trajectories after they collapse onto an invariant manifold $\mathcal{M}$.

In general, the training data for the development of a DManD model comes in the form of snapshots $\{u_1, u_2, ..., u_M\}$, which are either the full state or measurements diffeomorphic to the full state (e.g. time delays [203, 239]). Here we consider full-state data u that lives in an ambient space $\mathbb{R}^d$. We generate a time series of data by evolving this state forward in time according to

$$\frac{du}{dt} = f(u). \tag{6.1}$$

(In the present context, this equation represents a fully-resolved direct numerical simulation (DNS).) With the full state, we can then define a mapping to a low-dimensional state representation

$$h = \chi(u), \tag{6.2}$$

with $h \in \mathbb{R}^{d_h}$ is a coordinate representation on the manifold. Finally, we define a mapping back to the full state

$$\tilde{u} = \check{\chi}(h). \tag{6.3}$$

For data that lies on a finite-dimensional invariant manifold these functions can exactly reconstruct the state (i.e. $\tilde{u} = u$). However, if the dimension d_h is too low, or there are errors in the approximation of these functions, then $\tilde{u}$ approximates the state. Then, with this low-dimensional state representation, we can define an evolution equation

$$\frac{dh}{dt} = g(h). \tag{6.4}$$

The DManD model consists of the three functions χ, $\check{\chi}$, and g. By approximating these functions, the evolution of trajectories on the manifold can be performed entirely in the manifold coordinates h, which requires far fewer operations than a full simulation, as $d_h \ll d$. We choose to approximate all of these functions using NNs, but other representations could

be used.

First, we train χ and $\check{\chi}$ using an undercomplete autoencoder. This is a NN structure consisting of an encoder, which reduces dimension (χ), and a decoder, which expands dimension ($\check{\chi}$). As mentioned in Sec. 6.1, a common approach to dimension reduction is to project onto a set of POD modes. POD gives the optimal linear projection in terms of reconstruction error, so we use this fact to train an encoder as the sum of POD and a correction in the form of an NN:

$$h = \chi(u;\theta_E) = U_{d_h}^T u + \mathcal{E}(U_r^T u;\theta_E). \tag{6.5}$$

In this equation, $U_k \in \mathbb{R}^{d\times k}$ is a matrix whose k columns are the first k POD modes as ordered by variance, and $\mathcal{E}$ is a NN. The first term ($U_{d_h}^T u$) is the projection onto the leading d_h POD modes, and the second term is the NN correction. The matrix U_r in this term may either be a full change of basis with no approximation ($r = d$), or involve some dimension reduction ($d > r > d_h$).

For mapping back to the full state (decoding), we again sum POD with a correction

$$\tilde{u} = \check{\chi}(h;\theta_E) = U_r([h,0]^T + \mathcal{D}(h;\theta_D)). \tag{6.6}$$

Here, $[h,0]^T$ is the h vector zero padded to the correct size, and $\mathcal{D}$ is a NN. The first term is the POD mapping back to the full space, and the second term is a NN correction. In Linot and Graham [128] we refer to this structure as a hybrid autoencoder. In Sec. 6.3.2 we contrast this to a "standard" autoencoder where $h = \mathcal{E}(U_r^T u;\theta_E)$ and $\tilde{u} = U_r\mathcal{D}(h;\theta_D)$. These hybrid autoencoder operations act as a shortcut connection on the optimal linear dimension reduction, which we [128] found useful for representing the data and achieving accurate reconstruction of u. Yu et al. [240] also took a similar approach with POD shortcut connections over each layer of the network.

We determine the NN parameters θ_E and θ_D by minimizing

$$L = \frac{1}{dK}\sum_{i=1}^{K} ||u(t_i) - \check{\chi}(\chi(u(t_i);\theta_E);\theta_D)||^2 + \frac{1}{d_h K}\sum_{i=1}^{K} \alpha||\mathcal{E}(U_r^T u(t_i);\theta_E) + \mathcal{D}_{d_h}(h(t_i);\theta_D)||^2. \tag{6.7}$$

The first term in this loss is the mean-squared error (MSE) of the reconstruction $\tilde{u}$, and the second term is a penalty that promotes accurate representation of the leading d_h POD coefficients. In this term, $\mathcal{D}_{d_h}$ denotes the leading d_h elements of the decoder output. For finite α, the autoencoder exactly matches the first d_h POD coefficients when this term vanishes. Details of the minimization procedure are discussed in Sec. 6.3.

Next, we approximate g using a neural ODE. A drawback of training a single dense NN for g is that the resulting dynamics may become weakly unstable, with linear growth at long times [129, 132]. To avoid this, we use a "stabilized" neural ODE approach by adding a linear damping term onto the output of the NN, giving

$$g(h(t_i);\theta_g) = g_{\mathrm{NN}}(h(t_i);\theta_g) + Ah. \tag{6.8}$$

Integrating Eq. 6.8 forward from time t_i to $t_i + \tau$ yields

$$\tilde{h}(t_i + \tau) = h(t_i) + \int_{t_i}^{t_i+\tau} g_{\mathrm{NN}}(h(t);\theta_g) + Ah(t)dt. \tag{6.9}$$

Depending on the situation, one may either learn A from data or fix it. Here we set it to the diagonal matrix

$$A_{ij} = -\beta\delta_{ij}\sigma_i(h) \tag{6.10}$$

where $\sigma_i(h)$ is the standard deviation of the ith component of h, β is a tunable parameter, and δ_{ij} is the Kronecker delta. This linear term attracts trajectories back to the origin, preventing them from moving far away from the training data. In Sec. 6.3.4 we show that this approach drastically improves the long-time performance of these models.

We then determine the parameters θ_g by minimizing the difference between the predicted state $\tilde{h}(t_i + \tau)$ and the true state $h(t_i + \tau)$, averaged over the data:

$$J = \frac{1}{d_h K} \sum_{i=1}^{K} \left(||h(t_i + \tau) - \tilde{h}(t_i + \tau)||_2^2 \right). \tag{6.11}$$

For clarity, we show the specific loss we use, which sums over only a single snapshot forward in time at a fixed τ. More generally, the loss can be formulated for arbitrary snapshot spacing and for multiple snapshots forward in time. To compute the gradient of J with respect to the neural network parameters θ_g, automatic differentiation can be used to backpropagate through the ODE solver that is used to compute the time integral in Eq. 6.9, or an adjoint problem can be solved backwards in time [24]. The adjoint method uses less memory than backpropagation, but h is low-dimensional and our prediction window for training is short, so we choose to backpropagate through the solver.

So far this approach to approximating χ, $\check{\chi}$, and g is general and does not directly account for the fact that the underlying equations are often invariant to certain symmetry operations. For example, one of the symmetries in PCF is a continuous translation symmetry in x and z (i.e. any solution shifted to another location in the domain gives another solution). This poses an issue for training, because in principle, the training data must include *all* these translations to accurately model the dynamics under any translation. We discuss these and other symmetries of PCF in Sec. 6.3.1.

In practice, accounting for continuous symmetries is most important along directions that sample different phases very slowly. For PCF, the mean flow is in the x direction, leading to good phase sampling along this direction. However, there is no mean flow in the z direction, so sampling all phases relies on the slow phase diffusion in that direction. Therefore, we will only explicitly account for the z-phase in Sec. 6.3, but in the current discussion, we present the general framework accounting for all continuous symmetries.

To address the issue of continuous translations, we add an additional preprocessing step

to the data, using the method of slices [18, 19] to split the state u into a pattern $u_p \in \mathbb{R}^d$ and a phase $\phi \in \mathbb{R}^c$. The number of continuous translation symmetries for which we explicitly account determines c. We discuss the details of computing the pattern and the phase in Sec. 6.3.1. Separating the pattern and phase is useful because the evolution of both the pattern and the phase only depend on the pattern. Thus, we simply replace u with u_p in all the above equations and then write one additional ODE for the phase

$$\frac{d\phi}{dt} = g_\phi(h; \theta_\phi). \tag{6.12}$$

We then fix the parameters of g to evolve h (from u_p) forward in time and use that to make a phase prediction

$$\tilde{\phi}(t_i + \tau) = \phi(t_i) + \int_{t_i}^{t_i+\tau} g_\phi(h(t_i); \theta_\phi) dt. \tag{6.13}$$

Finally, we determine the parameters θ_ϕ to minimize the difference between the predicted phase $\tilde{\phi}(t_i + \tau)$ and the true phase $\phi(t_i + \tau)$

$$J_\phi = \frac{1}{cK} \sum_{i=1}^{K} \left(||\phi(t_i + \tau) - \tilde{\phi}(t_i + \tau)||^2 \right), \tag{6.14}$$

using the method described above to compute the gradient of J_ϕ.

6.3 Results

6.3.1 Description of plane Couette flow data

In the following sections we apply DManD to DNS of turbulent PCF in a MFU domain. Specifically, we consider the well-studied Hamilton, Kim, and Waleffe (HKW) domain [83]. We made this selection to compare our DManD results to the analysis of the self-sustaining process in this domain, to compare our DManD results to other Galerkin-based ROMs, and to compare our DManD results to known unstable periodic solutions in this domain.

For PCF we solve the Navier-Stokes equations

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = -\nabla p + \mathrm{Re}^{-1} \nabla^2 \mathbf{u}, \quad \nabla \cdot \mathbf{u} = 0 \tag{6.15}$$

for a fluid confined between two plates moving in opposite directions with the same speed. Eq. 6.15 is the nondimensionalized form of the equations with velocities in the streamwise $x \in [0, L_x]$, wall-normal $y \in [-1, 1]$, and spanwise $z \in [0, L_z]$ directions defined as $\mathbf{u} = [u_x, u_y, u_z]$, and pressure p. We solve this equation for a domain with periodic boundary conditions in x and z ($\mathbf{u}(0, y, z) = \mathbf{u}(L_x, y, z), \mathbf{u}(x, y, 0) = \mathbf{u}(x, y, L_z)$) and no-slip, no-penetration boundary conditions in y ($u_x(x, \pm 1, z) = \pm 1, u_y(x, \pm 1, z) = u_z(x, \pm 1, z) = 0$). The complexity of the flow increases as the Reynolds number Re increases and the domain size L_x and L_z increase. Here we use the HKW cell, which is at $\mathrm{Re} = 400$ with a domain size $[L_x, L_y, L_z] = [1.75\pi, 2, 1.2\pi]$ [83]. The HKW cell is one of the simplest flows that sustains turbulence for extended periods of time before relaminarizing. We chose to use this flow because it is well studied (refer to Sec. 6.1), it isolates the SSP [83], and a library of ECS exists for this domain [69]. Here we are interested in modeling the turbulent dynamics, so we will remove data upon relaminarization as detailed below.

Eq. 6.15, under the boundary conditions described, is invariant (and its solutions equivariant) under the discrete symmetries of point reflections about $[x, y, z] = [0, 0, 0]$

$$\mathcal{P} \cdot [(u_x, u_y, u_z, p)(x, y, z, t)] = (-u_x, -u_y, -u_z, p)(-x, -y, -z, t) \tag{6.16}$$

reflection about the $z = 0$ plane

$$\mathcal{R} \cdot [(u_x, u_y, u_z, p)(x, y, z, t)] = (u_x, u_y, -u_z, p)(x, y, -z, t) \tag{6.17}$$

and rotation by π about the z-axis

$$\mathcal{RP} \cdot [(u_x, u_y, u_z, p)(x, y, z, t)] = (-u_x, -u_y, u_z, p)(-x, -y, z, t). \tag{6.18}$$

In addition to the discrete symmetries, there are also continuous translation symmetries in x and z

$$\mathcal{T}_{\sigma_x,\sigma_z} \cdot [(u_x, u_y, u_z, p)(x, y, z, t)] = (u_x, u_y, u_z, p)(x + \sigma_x, y, z + \sigma_z, t). \tag{6.19}$$

We incorporate all these symmetries in the POD representation [193], as we discuss further in Sec. 6.3.2. Then, we use the method of slices [18] to phase align in the z direction. By phase aligning in z we fix the location of the low-speed streak. Without the alignment in z, models performed poorly because the models needed to learn how to represent every spatial shift of every snapshot. In what follows, we only consider phase alignment in z, but we note that extending this work to phase alignment in x is straightforward. To phase-align the data, we use the first Fourier mode method-of-slices [18]. First, we compute a phase by taking the Fourier transform of the streamwise velocity in x and z ($\hat{u}_x(k_x, y, k_z) = \mathcal{F}_{x,z}(u_x)$) at a specific y location (y_1) to compute the phase

$$\phi = \text{atan2}(\text{imag}(\hat{u}_x(0, y_1, 1)), \text{real}(\hat{u}_x(0, y_1, 1))). \tag{6.20}$$

We select y_1 to be one grid point off the bottom wall. Then we compute the pattern dynamics by using the Fourier shift theorem to set the phase to 0 (i.e. move the low-speed streak to the center of the channel)

$$\mathbf{u}_p = \mathcal{F}^{-1}_{x,z}(\hat{\mathbf{u}} \exp(-ik_z\phi)). \tag{6.21}$$

We generate turbulent PCF trajectories using the pseudo-spectral *Channelflow* code developed by Gibson *et al.* (2012, 2021). In this code, the velocity and pressure fields are projected onto Fourier modes in x and z and Chebyshev polynomials of the first kind in y. These coefficients are evolved forward in time first using the multistage SMRK2 scheme

[197], then, after taking multiple timesteps, using the multistep Adams-Bashforth Backward-Differentiation 3 scheme [164]. At each timestep, a pressure boundary condition is found such that incompressibility is satisfied at the wall ($du_y/dy = 0$) using the influence matrix method and tau correction developed by Kleiser and Schumann [115].

Data was generated with $\Delta t = 0.0325$ on a grid of $[N_x, N_y, N_z] = [32, 35, 32]$ in x, y, and z for the HKW cell. Starting from random divergence-free initial conditions, we ran simulations forward for either $10,000$ time units or until relaminarization. Then we dropped the first $1,000$ time units as transient data and the last $1,000$ time units to avoid laminar data and repeated with a new initial condition until we had $91,562$ time units of data stored at intervals of one time unit. We split this data into 80% for training and 20% for testing. Finally, we preprocess the data by computing the mean $\langle \mathbf{u} \rangle (y)$ from the training data and subtracting it from all data $\mathbf{u}' = \mathbf{u} - \langle \mathbf{u} \rangle$, and then we compute the pattern $\mathbf{u}'_p$ and the phase ϕ as described above. The pattern u_p described in Sec. 6.2 is $\mathbf{u}'_p$ flattened into a vector (i.e. $d = 3N_xN_yN_z$). The POD and NN training use only the training data, and all comparisons use test data unless otherwise specified.

6.3.2 Dimension reduction and dynamic model construction

Linear dimension reduction with POD: From $\mathcal{O}(10^5)$ to $\mathcal{O}(10^3)$

The first task in DManD for this Couette flow data is finding a low-dimensional parameterization of the manifold on which the long-time dynamics lie. We parameterize this manifold in two steps. First, we reduce the dimension down from $\mathcal{O}(10^5)$ to 502 with the proper orthogonal decomposition (POD), and, second, we use an autoencoder to reduce the dimension down to d_h. The first step is simply a preprocessing step to reduce the size of the data, which reduces the number of parameters in the autoencoder. Due to Whitney's embedding theorem [232, 233], we know that as long as the manifold dimension is less than 251 ($d_{\mathcal{M}} < 251$) then this POD representation is diffeomorphic to the full state. As we show later, the manifold dimension appears to be far lower than $d_{\mathcal{M}} = 251$, so no information of the full state should

be lost with this first step.

Proper orthogonal decomposition (POD) originates with the question of what function $\mathbf{\Phi}$ maximizes

$$\frac{\left\langle \left| (\mathbf{u}', \mathbf{\Phi}) \right|^2 \right\rangle}{||\mathbf{\Phi}||^2}. \tag{6.22}$$

Solutions $\mathbf{\Phi}^{(n)}$ to this problem satisfy the eigenvalue problem

$$\sum_{j=1}^{3} \int_0^{L_x} \int_{-1}^{1} \int_0^{L_z} \left\langle u_i'(\mathbf{x}, t) u_j'^{*}(\mathbf{x}', t) \right\rangle \Phi_j^{(n)}(\mathbf{x}')\, d\mathbf{x}' = \lambda_i \Phi_i^{(n)}(\mathbf{x}) \tag{6.23}$$

[92, 193]. Unfortunately, upon approximating these integrals, with the trapezoidal rule for example, this becomes a $d \times d$ matrix, making computation intractable. Furthermore, computing the average in Eq. 6.23, without any modifications, results in POD modes that fail to preserve the underlying symmetries described above.

In order to make this problem computationally tractable, and preserve symmetries, we apply the POD method used in Smith et al. [193], with the slight difference that we first subtract off the mean of state before performing the analysis. The first step in this procedure is to treat the POD modes as Fourier modes in both the x and z directions. Holmes *et al.* show in [92] that for translation-invariant directions Fourier modes are the optimal POD modes. This step transforms the eigenvalue problem into

$$L_x L_z \sum_{j=1}^{3} \int_{-1}^{1} \left\langle \hat{u}_i'(k_x, y', k_z, t) \hat{u}_j'^{*}(k_x, y', k_z, t) \right\rangle \varphi_{jk_xk_z}^{(n)}(y')\, dy' = \lambda_{k_xk_z}^{(n)} \varphi_{ik_xk_z}^{(n)}(y), \tag{6.24}$$

which reduces the $d \times d$ eigenvalue problem down to a $3N_y \times 3N_y$ eigenvalue problem for every wavenumber pair (k_x, k_z) of Fourier coefficients. We used $5,000$ snapshots evenly sampled over the training data to compute the POD modes. Then, to account for the discrete symmetries, the data is augmented such that the mean in Eq. 6.24 is computed by adding all the discrete symmetries of each snapshot.

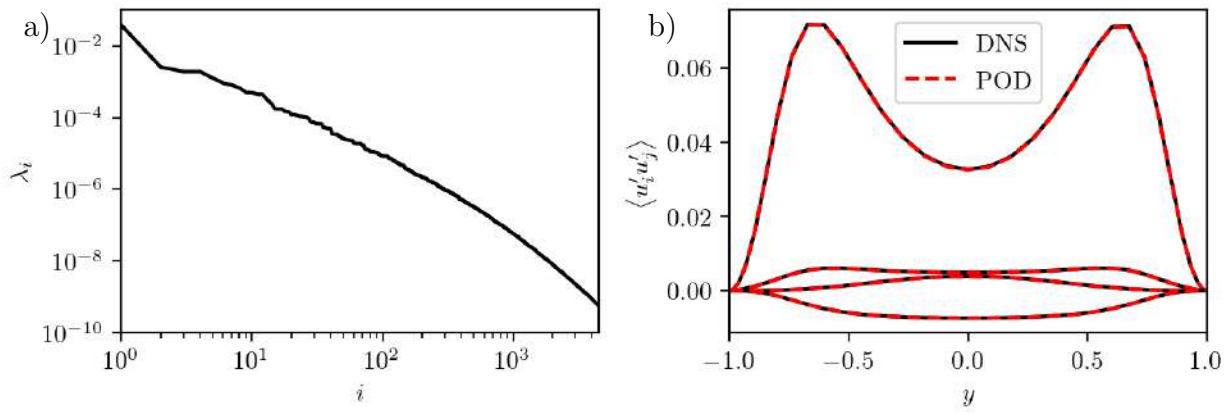


Figure 6.1: (a) Eigenvalues of POD modes sorted in descending order. (b) Components of the Reynolds stress for data generated by the DNS and this data projected onto 256 POD modes. In (a) the curves are, from top to bottom, $\langle u_x'^2 \rangle$, $\langle u_z'^2 \rangle$, $\langle u_y'^2 \rangle$, and $\langle u_x' u_y' \rangle$.

This analysis gives us POD modes

$$\mathbf{\Phi}^{(n)}_{k_x k_z}(\mathbf{x}) = \frac{1}{\sqrt{L_x L_z}} \exp\left(2\pi i \left(\frac{k_x x}{L_x} + \frac{k_z z}{L_z}\right)\right) \boldsymbol{\varphi}^{(n)}_{k_x k_z}(y), \tag{6.25}$$

and eigenvalues $\lambda^{(n)}_{k_x k_z}$. We sort the modes from largest eigenvalue to smallest eigenvalue (λ_i) and project onto the leading 256 modes, giving us a vector of POD coefficients $a(t)$. A majority of these modes are complex, so projecting onto these modes results in a 502-dimensional system. In Fig. 6.1a we plot the eigenvalues, which show a rapid drop and then a long tail that contributes little to the energy content. By dividing the eigenvalues of the leading 256 modes by the total, we find these modes contain 99.8% of the energy. To further illustrate that 256 modes is sufficient to represent the state in this case, we consider the reconstruction of statistics after projecting onto the POD modes. In Fig. 6.1b we show the reconstruction of four components of the Reynolds stress, $\langle u_x'^2 \rangle$, $\langle u_z'^2 \rangle$, $\langle u_y'^2 \rangle$, and $\langle u_x' u_y' \rangle$. The projection onto POD modes matches all of these quantities extremely well.

Now that we have converted the data to POD coefficients and filtered out the low energy modes, we next train an autoencoder to perform *nonlinear* dimension reduction. As mentioned in Sec. 6.2, we phase-align the data in the spanwise direction at this step using the

Table 6.1: Architectures of NNs. "Shape" indicates the dimension of each layer, "Activation" is the corresponding activation functions, and "sig" is the sigmoid activation."Learning Rate" gives the learning rate at multiple times during training. The learning rates was dropped at even intervals.

Function	Shape	Activation	Learning Rate
$\mathcal{E}$	$502/1000/d_h$	sig/linear	$[10^{-3}, 10^{-4}]$
$\mathcal{D}$	$d_h/1000/502$	sig/linear	$[10^{-3}, 10^{-4}]$
g_{NN}	$d_h/200/200/d_h$	sig/sig/linear	$[10^{-2}, 10^{-3}, 10^{-4}]$
g_ϕ	$d_h/200/200/1$	sig/sig/linear	$[10^{-2}, 10^{-3}, 10^{-4}]$

first Fourier mode method-of-slices. A common practice when training NNs is to normalize the data by subtracting the mean and dividing by the standard deviation of each component. We do not take this approach here because the standard deviation of the higher POD coefficients, which contribute less to the reconstruction, is much smaller than the lower POD coefficients. In order to retain the important information in the magnitudes, but put the values in a more amenable form for NN training, we instead normalize the POD coefficients by subtracting the mean and dividing by the maximum standard deviation. Then, we take this input and train autoencoders to minimize the loss in Eq. 6.7 using an Adam optimizer [113] in Keras [31]. We train for 500 epochs with a learning rate scheduler that drops the learning rate from 10^{-3} to 10^{-4} after 400 epochs. At this point, we see no improvement in the reconstruction error. For the hybrid autoencoder approach, we set $\alpha = 0.01$. Table 6.1 includes additional NN architecture details.

Nonlinear dimension reduction with autoencoders: From $\mathcal{O}(10^3)$ to $\mathcal{O}(10^1)$

With the above "preprocessing" step completed, we now turn to the reduction of dimension with the nonlinear approach enabled by the autoencoder structure. We consider three approaches to reducing the dimension of a: (1) Training a hybrid autoencoder, (2) Training a standard autoencoder, (3) linear projection onto a small set of POD modes. We describe the first two approaches in Sec. 6.2, noting that the POD projection onto 256 (complex) modes can be written as $a = U_r^T u$. The third approach just corresponds to setting $\mathcal{E}$ and $\mathcal{D}$ to

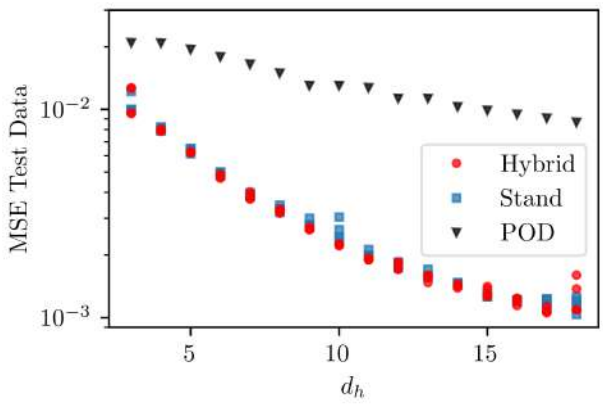


Figure 6.2: Mean squared error on test data for POD, standard autoencoders, and hybrid autoencoders at various dimensions d_h. At each dimension there are four standard and four hybrid autoencoders.

zero in Eqs. 6.5 and 6.6. In Fig. 6.2 we show the MSE of reconstructing a with these three approaches over a range of different dimensions d_h. We use NNs with the same architectures for both the standard and the hybrid autoencoder approaches. Due to the variability introduced into autoencoder training by randomly initialized weights and stochasticity in the optimization, we show the error for four separately trained autoencoders, at each d_h. We see that the autoencoders perform an order magnitude better than POD in the range of dimensions considered here. Both the standard and hybrid autoencoder approaches perform the same, so we select the hybrid approach because it has an advantage in interpretability – each component of h can be viewed as a POD mode amplitude with a nonlinear correction. Next, we use the low-dimensional representations from these autoencoders to train stabilized neural ODEs.

Neural ODE training

After training four autoencoders at each dimension d_h, we chose a set of damping parameters, β, and for each, then trained four stabilized neural ODEs for all four autoencoders at every dimension d_h. This results in 16 models at every d_h and β. The final β value of 0.1 was selected so that long-time trajectories neither blew up nor decayed too strongly. Before training the ODEs, we preprocess each autoencoder's latent space data set h by subtracting

the mean. It is important to center the data because the linear damping (Eq. 6.10) pushes trajectories towards the origin. We train the stabilized neural ODEs to predict the evolution of the centered data by using an Adam optimizer in Pytorch [24, 161] to minimize the loss in Eq. 6.11. We train using a learning rate scheduler that drops at three even intervals during training and we train until the learning curve stops improving. Table 6.1 shows the details of this NN. Unless otherwise stated, we show results for the one model out of those sixteen at each dimension with the lowest relative error averaged over all the statistics we consider.

6.3.3 Short-time tracking

In the following two sections, we evaluate the performance of the DManD models at reconstructing short-time trajectories and long-time statistics. Figure 6.3 shows snapshots of the streamwise velocity at the center plane of the channel, $y = 0$, for the DNS and DManD at $d_h = 18$. We choose to show results for $d_h = 18$ because the autoencoder error begins to level off around this dimension, and, as we will show, the error in statistics levels off before this dimension. The value $d_h = 18$ is not necessarily the minimal dimension required to model this system. In Fig. 6.3, both the DNS and the DmanD model show key characteristics of the SSP: (1) low-speed streaks become wavy, (2) the wavy low-speed streaks break down generating rolls, (3) the rolls lift fluid from the walls, regenerating streaks.

Not only does DManD capture the qualitative behavior of the SSP, but Fig. 6.3 also shows good quantitative agreement as well. To further illustrate this, in Fig. 6.4 we show the modal root-mean squared (RMS) velocity

$$M(k_x, k_z) = \left(\int_{-1}^{1} (\hat{u}_x^2(k_x, y, k_z) + \hat{u}_y^2(k_x, y, k_z) + \hat{u}_z^2(k_x, y, k_z)) dy \right)^{1/2}, \tag{6.26}$$

which Hamilton et al. [83] used to identify the different parts of the SSP. Specifically, we consider the $M(0, 1)$ mode, which corresponds to the low-speed streak, and the $M(1, 0)$ mode which corresponds to the x-dependence that appears when the streak becomes wavy

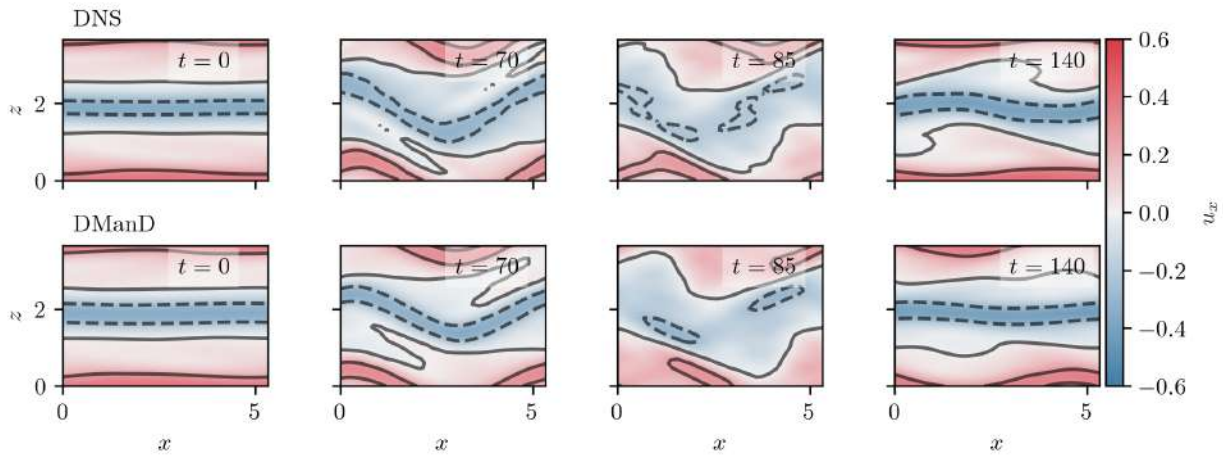


Figure 6.3: Snapshots of the streamwise velocity at $y = 0$ from the DNS and from the DManD model at $d_h = 18$.

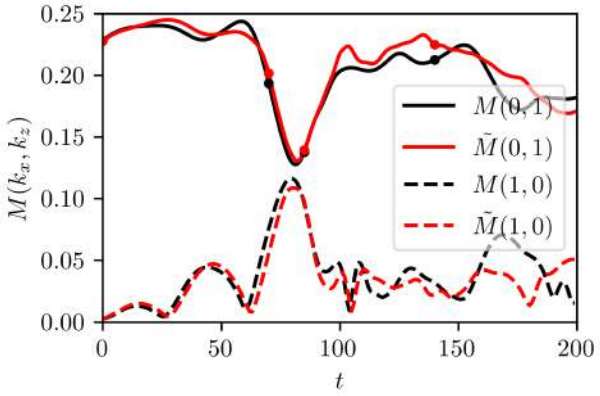


Figure 6.4: Modal RMS velocity from the DNS (M) and from the DManD model at $d_h = 18$ ($\tilde{M}$). The markers correspond to the times in Fig. 6.3.

and breaks up. In this example, the two curves match well over a cycle of the SSP and only start to move away after ~ 150 time units, which is about three Lyapunov times.

While the previous result shows a single example, we also consider ensembles of initial conditions. Figure 6.5 shows the tracking error $||a(t_i + t) - \tilde{a}(t_i + t)||$ of 10 trajectories, starting at t_i, for a model at $d_h = 18$. Here we normalize the tracking error by the error between solutions at random times t_i and t_j $D = \langle ||a(t_i) - a(t)|| \rangle$. In this case, the darkest line corresponds to the flow field in Figs. 6.3 and 6.4. When considering the other initial conditions in Fig. 6.5, there tends to be a relatively slow rise in the error over $\sim$50 time units and then a more rapid increase after this point. To better understand how this tracking

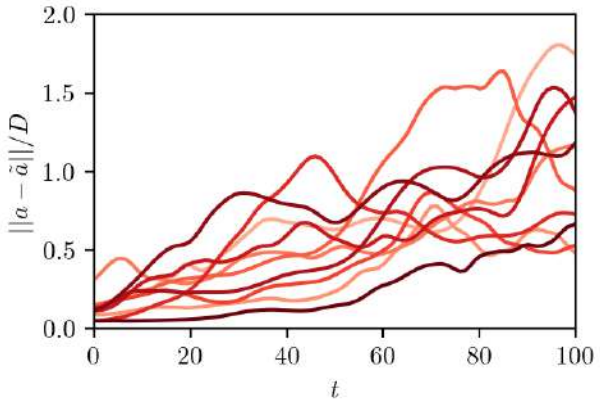


Figure 6.5: Normalized tracking error for 10 random initial conditions (different shades) using DManD with $d_h = 18$.

varies with the dimension of the model we next consider the ensemble-averaged tracking error.

In Fig. 6.6a we show the normalized ensemble-averaged tracking error for model dimensions between $d_h = 3$ and 18. For $d_h = 3 - 5$ there is a rapid rise in the error until ∼40 time units after which the error levels off. This behavior often happens due to trajectories quickly diverging and landing on stable fixed points or periodic orbits that do not exist in the true system. For $d_h = 6 - 10$ there is an intermediate behavior where lines diverge more quickly than the higher-dimensional models, but tend to approach the same tracking error at ∼100 time units. Then, for the remaining models $d_h = 11 - 18$, there is a smooth improvement in the tracking error over this time interval. As the dimension increases in this range the trends stay the same, but the error continues to decrease, which is partially due to improvement in the autoencoder performance.

The instantaneous kinetic energy of the flow is

$$E(t) = \frac{1}{2L_xL_z}\int_0^{L_z}\int_{-1}^{1}\int_0^{L_x}\frac{1}{2}\mathbf{u}\cdot\mathbf{u}dxdydz, \tag{6.27}$$

and we denote its fluctuating part as $k(t) = E(t) - \langle E\rangle$. In Fig. 6.6b we show the temporal autocorrelation of k. Again, for $d_h = 3 - 5$ we see clear disagreement between the true

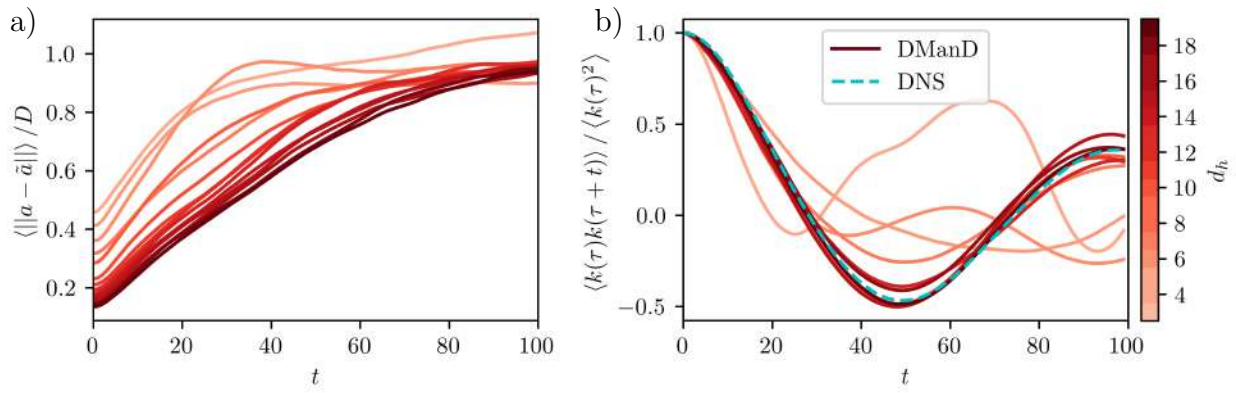


Figure 6.6: (a) ensemble averaged short-time tracking and (b) temporal autocorrelation of the kinetic energy for DmanD models of increasing dimension. In (b) odd numbers above $d_h = 5$ are omitted for clarity.

autocorrelation and the prediction. Above $d_h > 5$ all of the models match the temporal autocorrelation well, without a clear trend in the error as dimension changes. All these models match well for ~ 40 time units, with $d_h = 18$ (the darkest line) matching the data extremely well for two Lyapunov times.

Finally, before investigating the long-time predictive capabilities of the model, we show the tracking of phase dynamics for $d_h = 18$. As mentioned in Sec. 6.2, we decouple the phase and pattern dynamics such that the time evolution of the phase only depends upon the pattern dynamics. Here we take the $d_h = 18$ model and used it to train an ODE for the phase dynamics. For training, we repeat the process used for training g_{NN} to train g_ϕ with the loss in Eq. 6.14. Table 6.1 contains details on the NN architecture.

In Fig. 6.7a we show an example of the model phase evolution over 200 time units. In this example, the model follows the same downward drift in phase despite not matching exactly. Then, to show the statistical agreement between the DNS and the model, we show the mean squared phase displacement MSD $= \langle (\theta(t_i) - \theta(t_i + t))^2 \rangle$ for both the DNS and the model in Fig. 6.7b, as was done for Kolmogorov flow by [163]. The curves are in good agreement. All of the remaining long-time statistics we report are phase invariant, so the remaining results use only models for the pattern dynamics.

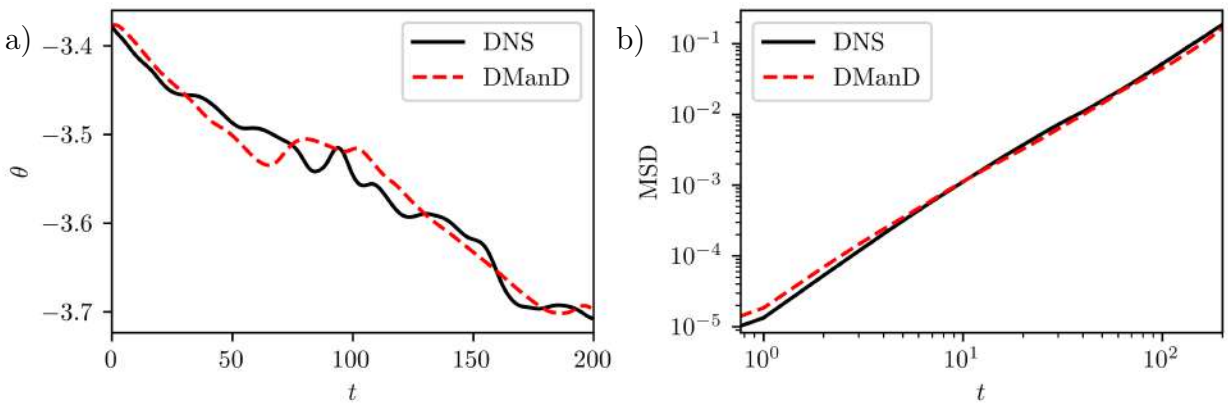


Figure 6.7: (a) an example of the phase evolution and (b) the MSD of the phase evolution for the DNS and the DManD model at $d_h = 18$.

6.3.4 Long-time statistics

Next, we investigate the ability of the DManD model to capture the long-time dynamics of PCF. An obvious prerequisite for models to capture long-time dynamics is the long-time stability of the models. As mentioned in Sec. 6.2, the long-time trajectories of standard neural ODEs often become unstable, which led us to use stabilized neural ODEs with an explicit damping term. We quantify this observation by counting, of the 16 models trained at each dimension d_h, how many become unstable with and without the presence of an explicit damping term. From our training data we know where h should lie, so if it falls far outside this range after some time we can classify the model as unstable. In particular, we classify models as unstable if the norm of the final state is two times that of the maximum in our data ($||\tilde{h}(T)|| > 2 \max_t ||h(t)||$), after $T = 10^4$ time units. In all of the unstable cases $||\tilde{h}(t)||$ follows the data over some short time range before eventually growing indefinitely.

In Fig. 6.8 we show the number of unstable models with and without damping. With damping, all of the models are stable, whereas without damping almost all models become unstable for $d_h = 5 - 16$, and around half become unstable in the other cases. Additionally, with longer runs or with different initial conditions, many of the models without damping labeled as stable here also eventually become unstable. This lack of stability happens

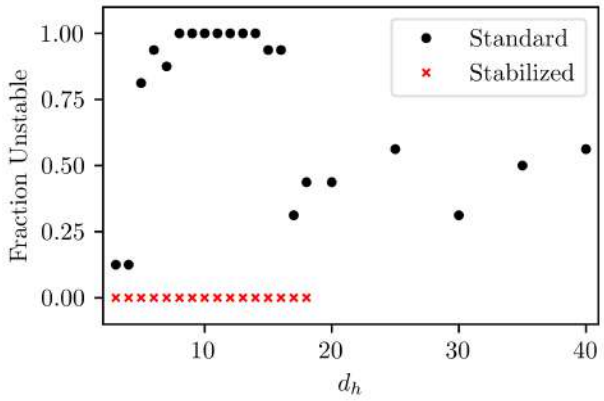


Figure 6.8: Fraction of unstable DManD models with standard neural ODEs and with stabilized neural ODEs at various dimensions.

when inaccuracies in the neural ODE model push trajectories off the attractor. Once off the attractor, the model is presented with states unlike the training data leading to further growth in this error. We show more results highlighting this behavior in Linot and Graham [129], Linot et al. [132]. So, although some standard neural ODE models do provide reasonable statistics, using these models presents challenges due to this lack of robustness. As such, all other results we show use stabilized neural ODEs.

While Fig. 6.8 indicates that stabilized neural ODEs predict $\tilde{h}$ in a similar range to that of the data, it does not quantify the accuracy of these predictions. In fact, with few dimensions many of these models do not remain chaotic, landing on fixed points or periodic orbits. The first metric we use to quantify the long-time performance of the DManD method is the mean-squared POD coefficient amplitudes ($\langle||a_n||^2\rangle$). We consider this quantity because Gibson reports it for POD-Galerkin in Gibson [66] at various levels of truncation. In Fig. 6.9 we show how well the DManD model, with $d_h = 18$, captures this quantity, in comparison to the POD-Galerkin model in Gibson [66]. The two data sets slightly differ because we subtract the mean before applying POD and Gibson did not. The DManD method, with only 18 degrees of freedom, matches the mean-squared amplitudes to high accuracy, far better than all of the POD-Galerkin models. It is not until POD-Galerkin keeps 1024 modes that the results become comparable, which corresponds to $\sim$ 2000 degrees of freedom because

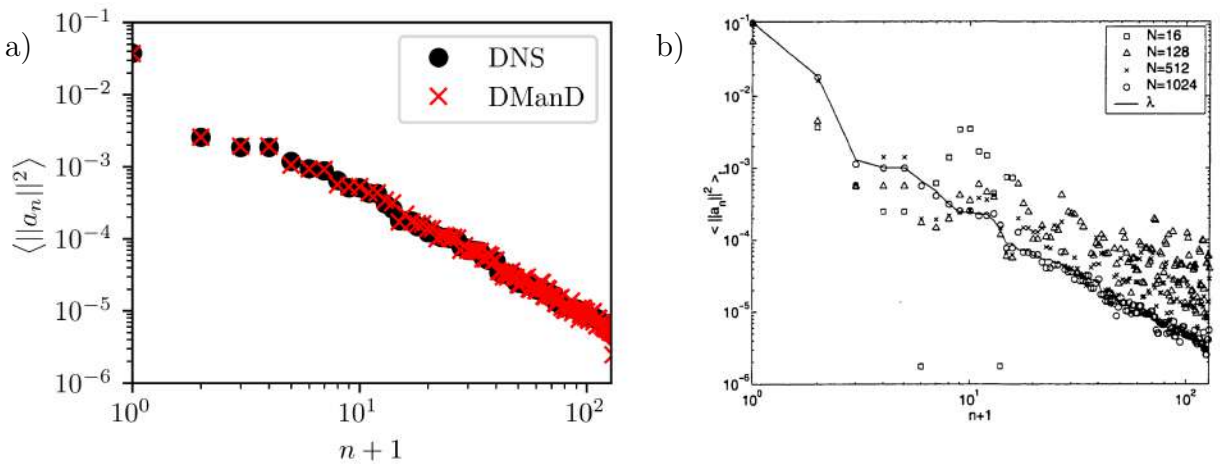


Figure 6.9: Comparison of $\langle ||a_n||^2 \rangle$ (mean-squared POD coefficient amplitudes) from the DNS to (a) $\langle ||a_n||^2 \rangle$ from the DManD model at $d_h = 18$ and (b) $\langle ||a_n||^2 \rangle$ from POD-Galerkin with N POD modes (reproduced with permission from Gibson [66]). In (b), the quantity λ is equivalent to $\langle ||a_n||^2 \rangle$ from the DNS.

most coefficients are complex. Additionally, our method requires only data, whereas the POD Galerkin approach requires both data for computing the POD and knowledge of the equations of motion for projecting the equations onto these modes.

We now investigate how the Reynolds stress and the power input vs. dissipation vary with dimension. Figure 6.10 shows four components of the Reynolds stress at various dimensions. For $\langle u_x'^2 \rangle$ and $\langle u_x' u_y' \rangle$, nearly all the models match the data, with relatively small deviations only appearing for $d_h \sim 3-6$. For $\langle u_y'^2 \rangle$ and $\langle u_z'^2 \rangle$, this deviation becomes more obvious, and the lines do not converge until around $d_h > 10$, with all models above this dimension exhibiting a minor overprediction in $\langle u_z'^2 \rangle$.

To evaluate how accurate the models are at reconstructing the energy balance, we look at joint PDFs of power input and dissipation. The power input is the amount of energy required to move the walls:

$$I = \frac{1}{2L_x L_z} \int_0^{L_x} \int_0^{L_z} \left. \frac{\partial u_x}{\partial y} \right|_{y=-1} + \left. \frac{\partial u_x}{\partial y} \right|_{y=1} dxdz, \tag{6.28}$$

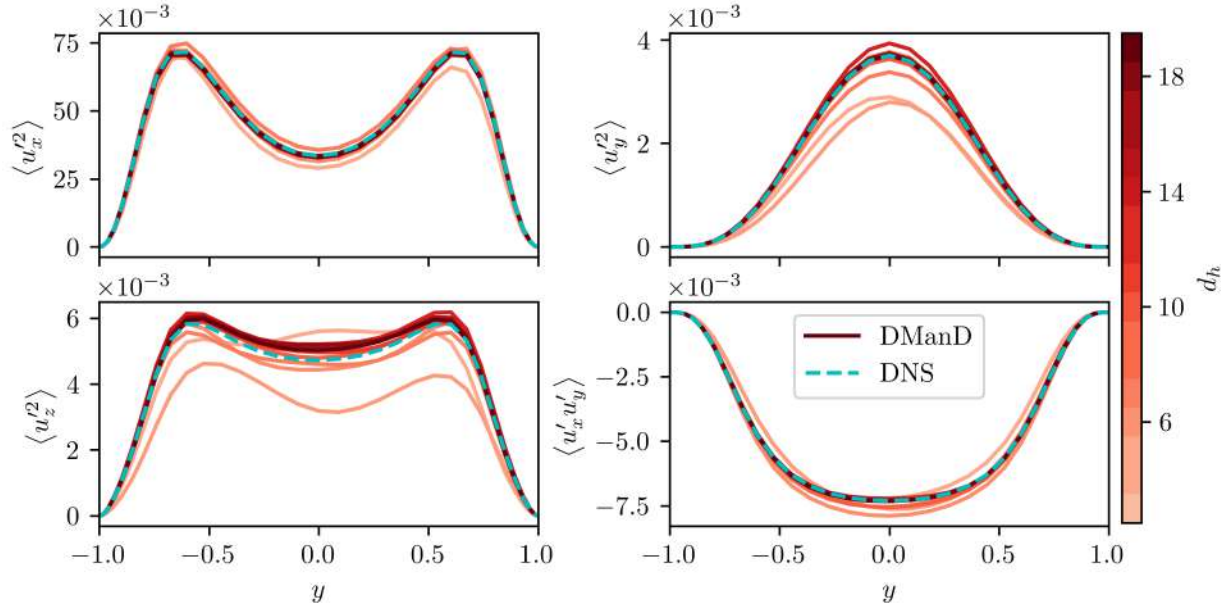


Figure 6.10: Components of the Reynolds stress with increasing dimension for DManD models at various dimensions. Odd numbers above $d_h = 5$ are omitted for clarity.

and the dissipation is the energy lost to heat due to viscosity:

$$D = \frac{1}{2L_x L_z} \int_0^{L_x} \int_{-1}^{1} \int_0^{L_z} |\nabla \times \mathbf{u}|^2 \, dxdydz. \tag{6.29}$$

These two terms define the rate of change of energy in the system $\dot{E} = I - D$, which must average to zero over long times. Checking this statistic is important to show the DManD models correctly balance the energy.

Figures 6.11a-6.11c show the PDF from the DNS, the PDF for $d_h = 6$ and the PDF for $d_h = 18$, generated from a single trajectory evolved for 5000 time units, and Figs. 6.11e and 6.11f show the the absolute difference between the true and model PDFs. With $d_h = 6$ the model overestimates the number of low dissipation states, while $d_h = 18$ matches the density well. In Fig. 6.11d we compare the joint PDFs at all dimensions with the true PDF using the earth movers distance (EMD) [181]. The EMD determines the distance between two PDFs as a solution to the *transportation problem* by treating the true PDF as "supplies" and the model PDF as "demands" and finding the flow which minimizes the work required

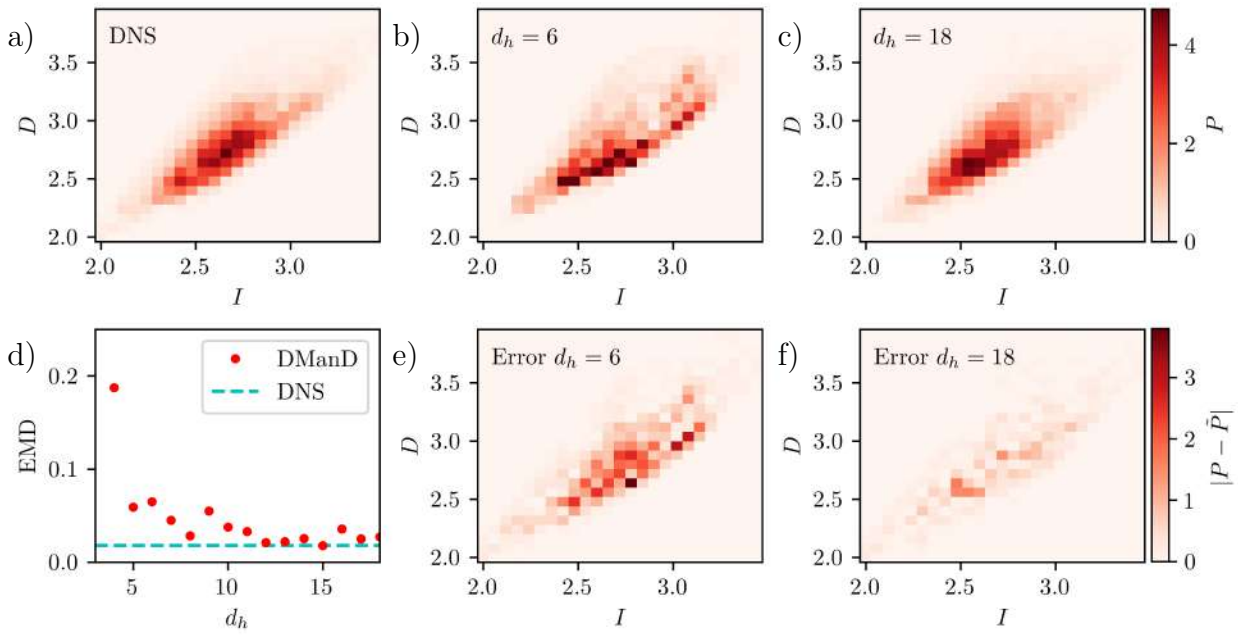


Figure 6.11: (a)-(c): examples of joint PDFs for the true system, the DManD model at $d_h = 6$, and the DManD model at $d_h = 18$. (d): earth movers distance between the PDF from the DNS and the PDFs predicted by the DManD model at various dimensions. "DNS" is the error between two PDFs generated from DNS trajectories of the same length with different initial conditions. (e) and (f): the error associated with the DManD model PDFs at $d_h = 6$ and $d_h = 18$.

to move one to the other. We compute the distance between PDFs using the EMD because it is a *cross-bin* distance, meaning the distance accounts for the density in neighboring bins. This is in contrast to *bin-to-bin* distances, like the KL divergence, which only uses the error at a given bin. Bin-to-bin distances can vary significantly with small shifts in one PDF (misalignment) and when changing the number of bins used to generate the PDF [127]. We choose the EMD because it does not suffer from these issues. In Fig. 6.11d we see a steep drop in the EMD at $d_h = 5$ and after $d_h > 10$ the joint PDFs are in excellent agreement with the DNS. The dashed line corresponds to the EMD between two different trajectories from the DNS.

6.3.5 Finding ECS in the model

Now that we know that the DManD model quantitatively captures many of the key characteristics of MFU PCF, we now want to explore using the model to discover ECS. In particular, we first investigate whether known periodic orbits of the DNS exist in the DManD model, and then we use the DManD model to search for new periodic orbits. Here we note that because our model predicts phase-aligned dynamics, the periodic orbits of the DManD model are either periodic or relative periodic orbits, depending on the phase evolution, which we have not tracked. In the following we omit all $\tilde{\cdot}$, so all functions should be assumed to come from a DManD model.

Here we outline the approach we take to find periodic orbits, which follows Cvitanović et al. [41]. When searching for periodic orbits we seek an initial condition to a trajectory that repeats after some time period. This is equivalent to finding the zeros of

$$H(h,T) = G_T(h) - h, \tag{6.30}$$

where $G_T(h)$ is the flow map forward T time units from h: i.e. $G_T(h(t)) = h(t+T)$. We compute $G_T(h)$ from Eq. 6.9. Finding zeros to Eq. 6.30 requires that we find both a point h^* on the periodic orbit and a period T^* such that $H(h^*, T^*) = 0$. One way to find h^* and T^* is by using the Newton-Raphson method.

By performing a Taylor series expansion of H we find near the fixed point h^*, T^* of H that

$$\begin{aligned} H(h^*,T^*) - H(h,T) &\approx D_h H(h,T)\Delta h + D_T H(h,T)\Delta T \\ -H(h,T) &\approx D_h H(h,T)\Delta h + g\left(G_T(h)\right)\Delta T, \end{aligned} \tag{6.31}$$

where D_h is the Jacobian of H with respect to h, D_T is the Jacobian of H with respect to the period T, $\Delta h = h^* - h$ and $\Delta T = T^* - T$. To have a complete set of equations for Δh and ΔT, we supplement Eq. 6.31 with the constraint that the updates Δh are orthogonal to

the vector field at h: i.e.,

$$g(h)^T \Delta h = 0. \tag{6.32}$$

With this constraint, at Newton step (i), the system of equations becomes

$$\begin{bmatrix} D_{h^{(i)}} H(h^{(i)}, T^{(i)}) & g(G_{T^{(i)}}(h^{(i)})) \\ g(h^{(i)})^T & 0 \end{bmatrix} \begin{bmatrix} \Delta h^{(i)} \\ \Delta T^{(i)} \end{bmatrix} = - \begin{bmatrix} H(h^{(i)}, T^{(i)}) \\ 0 \end{bmatrix}, \tag{6.33}$$

which, in the standard Newton-Raphson method, is used to update the guesses $h^{(i+1)} = h^{(i)} + \Delta h^{(i)}$ and $T^{(i+1)} = T^{(i)} + \Delta T^{(i)}$.

Typically, a Newton-Krylov method is used to avoid explicitly constructing the Jacobian [219]. However, with DManD, computing the Jacobian is simple, fast, and requires little memory because the state representation is dramatically smaller in the DManD model than in the DNS. We compute the Jacobian $D_h H(h, T)$ directly, with the same automatic differentiation tools used for training the neural ODE. Furthermore, if we had chosen to represent the dynamics in discrete, rather than continuous time, computation of general periodic orbits would not be possible, as the period T can take on arbitrary values and a discrete-time representation would limit T to multiples of the time step. When finding periodic orbits of the DManD model we used the Scipy "hybr" method, which uses a modification of the Powell hybrid method [218], and for finding periodic orbits of the DNS we used the Newton GMRES-Hookstep method built into Channelflow [70]. In the following trials we only consider DManD models with $d_h = 18$.

For the HKW cell there exists a library of POs made available by Gibson et al. [69]. To investigate if the DManD model finds POs similar to existing solutions, we took states from the known POs, encoded them, and used this as an initial condition in the DManD Newton solver to find POs in the model. In Fig. 6.12 we show projections of 12 known POs, which we identify by the period T, and compare them to projections of POs found using the DManD model. This makes up a majority of the POs made available by Gibson et al. [69]. Of the

Table 6.2: Details on the RPOs and POs found using initial conditions from the DManD model. The first 9 solutions are new and the last 3 had previously been found. "Label" indicates the label in Fig. 6.13b, σ_z corresponds to the phase-shift in z, T is the period of the orbit, and "Error" is ||shifted final state − initial state||/||initial state||, which is the same error as in [219].

Label	1	2	3	4	5	6	7	8	9	10	11	12
σ_z	1.91e-1	-9.66e-2	-1.77e-3	1.15e-1	-9.21e-3	-1.90e-1	-1.28e-2	-1.19e-1	-5.63e-5	4.64e-14	2.17e-14	2.73e-13
T	37.94	84.25	91.29	82.07	74.14	41.24	110.67	83.31	64.64	19.06	68.07	75.35
Error	2.23e-3	1.01e-3	3.92e-3	2.84e-3	1.87e-3	5.26e-4	1.25e-3	1.13e-3	2.25e-3	1.57e-4	2.55e-4	1.07e-4

other known solutions, three are RPOs with phase shifts in the streamwise direction that our model, with the current setup, can not capture. The other two have short periods of $T = 19.02$ and $T = 19.06$. Most of the POs found with DManD land on initial conditions near that of the DNS and follow similar trajectories to the DNS.

How close many of these trajectories are to the true PO is surprising and encouraging for many reasons. First, the data used for training the DManD model does not explicitly contain any POs. Second, this approach by no means guarantees convergence on a PO in the DManD model. Third, starting with an initial condition from a PO does not necessarily mean that the solution the Newton solver lands on will be the closest PO to that initial condition, so there may exist POs in the DManD model closer to the DNS solutions than what we present here.

Now that we know the DManD model can find POs similar to those known to exist for the DNS, we now use it to search for new POs. First, we searched for POs in three of the $d_h = 18$ models by randomly selecting 20 initial conditions and selecting 4 different periods $T = [20, 40, 60, 80]$. We then took the initial conditions and periods for converged periodic orbits and decoded and upsampled them onto a $48 \times 49 \times 48$ grid. We performed this upsampling because Viswanath [219] reported that solutions on the coarser grid can be computational artifacts. Finally, we put these new initial conditions into Channelflow and ran another Newton search for 100 iterations. This procedure resulted in us finding 9 new RPOs and 3 existing POs, the details of which we include in Table 6.2.

In Fig. 6.13a we show the new RPOs in the DManD model and in Fig. 6.13b we show what they converged to after putting them into the Channelflow Newton solver as initial

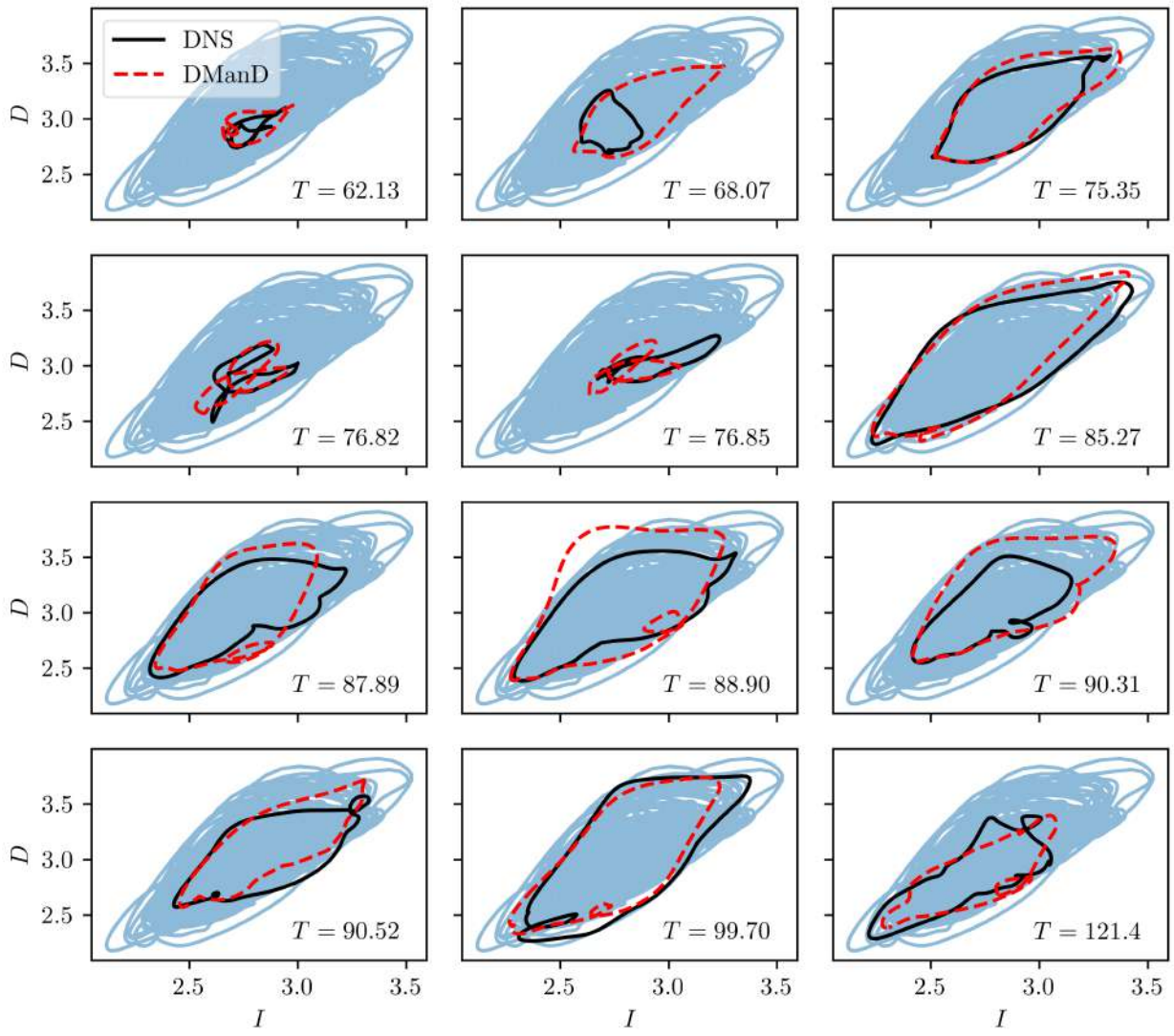


Figure 6.12: Power input vs. dissipation of known periodic orbits (period reported in the bottom right) from the DNS and periodic orbits found in the DManD model at $d_h = 18$. The blue line is a long trajectory of the DNS for comparison.

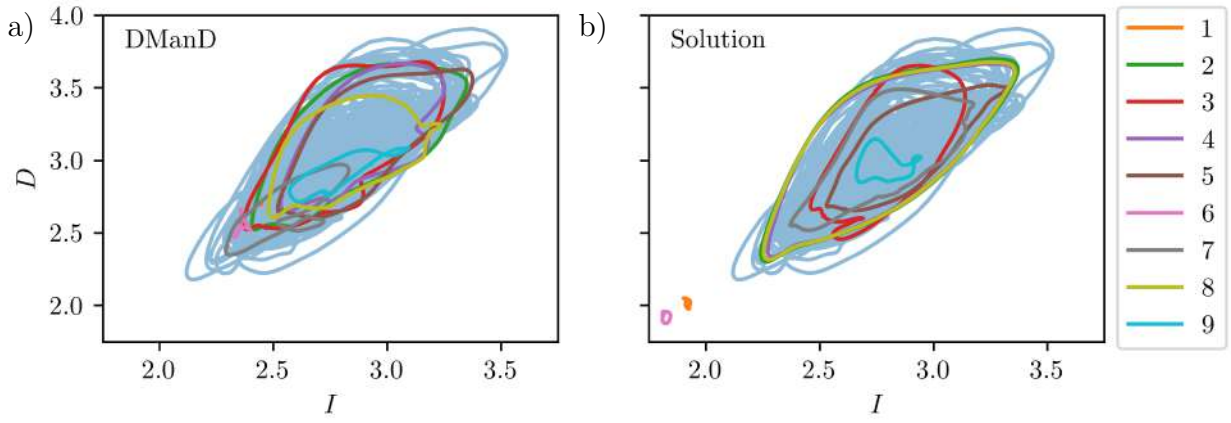


Figure 6.13: (a) Periodic orbits found in DManD models at $d_h = 18$ that converged to the (b) periodic orbits found in the DNS. Each of the colors corresponds to one of the new solutions in Table 6.2. The blue curve at the back is a long trajectory of the DNS for comparison.

guesses. Again, many of the RPOs end up following a similar path through this state space, with the biggest exceptions being RPO1 and RPO6, which converged to low-power input solutions. It is worth noting that this worked well, considering that the DManD initial conditions are POD coefficients from a model trained using data on a coarser grid than used to search for these solutions. We have described a new method to rapidly find new ECS, wherein an accurate low-dimensional model, like the DManD model presented here, is used to quickly perform a large number of ECS searches in the model, and then these solutions can be fine-tuned in the full simulation to land on new solutions.

6.4 Summary

In this chapter, we described a data-driven manifold dynamics method (DManD) and applied it for accurate modeling of MFU PCF with far fewer degrees of freedom ($\mathcal{O}(10)$) than required for the DNS ($\mathcal{O}(10^5)$). The DManD method consists of first finding a low-dimensional parameterization of the manifold on which data lies and then discovering an ODE to evolve this low-dimensional state representation forward in time. In both cases, we use NNs to approximate these functions from data. We find that an extremely low-dimensional param-

eterization of this manifold can be found using a hybrid autoencoder approach that corrects upon POD coefficients. Then, we use stabilized neural ODEs to accurately evolve the low-dimensional state forward in time.

The DManD model captures the self-sustaining process and accurately tracks trajectories and the temporal autocorrelation over short time horizons. For DManD models with $d_h > 10$ we found excellent agreement between the model and the DNS in computing the mean-squared POD coefficient amplitude, the Reynolds stress, and the joint PDF of power input vs. dissipation. For comparison, we showed that a POD-Galerkin model requires ~ 2000 degrees of freedom to get similar performance in matching the mean-squared POD coefficient amplitudes. Finally, we used the DManD model at $d_h = 18$ for PO searches. Using a set of existing POs, we successfully landed on nearby POs in the model. Finally, we found 9 previously undiscovered RPOs by first finding solutions in the DManD model and then using those as initial guesses to search in the full DNS.

The results reported here have both fundamental and technological importance. At the fundamental level, they indicate that the true dimension of the dynamics of a turbulent flow can be orders of magnitude smaller than the number of degrees of freedom required for a fully-resolved simulation. Technologically this point is important because it may enable, for example, highly sophisticated model-based nonlinear control algorithms to be used: Determining the control strategy from the low-dimensional DManD model rather than a full-scale DNS, and applying it to the full flow will speed up both learning and implementing a control policy [243, 244].

Chapter 7

Model-based control of Couette flow [1]

In this chapter, we apply the DManD-RL method to turbulent plane Couette flow. The high dimensionality and complex dynamics of turbulent flows remain an obstacle to the discovery and implementation of control strategies. Deep reinforcement learning (RL) is a promising avenue for overcoming these obstacles, but requires a training phase in which the RL agent iteratively interacts with the flow environment to learn a control policy, which can be prohibitively expensive when the environment involves slow experiments or large-scale simulations. We overcome this challenge using a framework we call "DManD-RL" (data-driven manifold dynamics-RL), which generates a data-driven low-dimensional model of our system that we use for RL training. With this approach, we seek to minimize drag in a direct numerical simulation (DNS) of a turbulent minimal flow unit of plane Couette flow at $\mathrm{Re} = 400$ using two slot jets on one wall. We obtain, from DNS data with $\mathcal{O}(10^5)$ degrees of freedom, a 25-dimensional DManD model of the dynamics by combining an autoencoder and neural ordinary differential equation. Using this model as the environment, we train an RL control agent, yielding a 440-fold speedup over training on the DNS, with equivalent control performance. The agent learns a policy that laminarizes 84% of unseen DNS test trajectories within 900 time units, significantly outperforming classical opposition control (58%), despite

[1] Parts of this chapter have been adapted from the manuscript titled "Turbulence control in plane Couette flow using low-dimensional neural ODE-based models and deep reinforcement learning" by A. J. Linot, K. Zeng, and M. D. Graham on arXiv and submitted for review to the *Int. Journal of Heat and Fluid Flow.*

the actuation authority being much more restricted. The agent often achieves laminarization through a counterintuitive strategy that drives the formation of two low-speed streaks, with a spanwise wavelength that is too small to be self-sustaining. The agent demonstrates the same performance when we limit observations to wall shear rate.

7.1 Introduction

Energy loss due to turbulent drag is ubiquitous in many industrial and commercial processes, ranging from air flowing over a plane wing, a ship in the ocean, or oil pumped through a pipe. In total, turbulent drag accounts for 25% of the energy used in industry and commerce, resulting in 5% of all man-made CO_2 emissions [104]. Even small reductions in this drag can yield massive savings in energy, which has long motivated the search for better flow control strategies.

Many control types for reducing turbulent drag exist, including, but not limited to, polymer/surfactant drag reduction [75, 217], riblets [194], wall oscillations [170], plasma actuators [30], and synthetic jets [71]. Due to the complexity of reducing drag, it has been most common to apply these control methods in an open-loop manner where the control policy at any given time is independent of the flow state [29]. However, the application of feedback control on a turbulent system could yield far better performance in controlling drag.

Unfortunately, the complexity of the problem has typically limited applications of feedback control to methods based on heuristics. A well-studied heuristic method is opposition control [28]. Here the wall-normal velocity at the wall is set to have the opposite sign as the wall-normal velocity at some detection plane in the channel, a straightforward actuation in simulations. This method has been applied in simulations [28, 34, 84, 99] and experiments (with some modifications) [25, 174], and extensions exist to use just wall observations [119, 160]. When heuristics are replaced with methods from optimal control theory, like

model predictive control (MPC), the drag reduction far outperforms opposition control [10] while using the same actuation scheme. However, the real-time implementation of MPC on DNS still remains infeasible because it involves solving the DNS forward over a time horizon (preferably a long one) and then solving an adjoint backwards in time for every actuation [10].

A potential approach to overcome the high computational cost of real-time optimization of a control strategy is deep reinforcement learning (RL) [202]. Deep RL gained significant traction when it was used to defeat the best professional players in GO [189], DOTA II [155], and Starcraft II [215], in addition to the best engines in GO, Chess, and Shogi [190]. In deep RL, a neural network (NN) control agent is trained through iterative interactions with the environment (i.e. the system to be controlled) to maximize a scalar total reward (i.e. control objective) that includes present as well as discounted future reward values. Once trained, the control agent can be deployed in real-time without the need for online optimization.

In recent years, RL has been applied in fluids simulations to reduce the drag experienced in flow around a cylinder [124, 171, 213], to optimize jets on an airfoil [229], and to find efficient swimming strategies [214]. RL has even been applied to experimental flow systems [54]. Recently multi-agent deep RL has been explored for the control of pressure-driven turbulent channel flow [79, 196] in a problem formulation similar to opposition control [28]. In these works, an RL policy is trained to map local detection plane observables to a wall-normal velocity response at the walls to reduce drag. Notably, the same RL policy is locally implemented at each wall grid point. We differentiate the control problem addressed from the previously mentioned works in that we limit the control authority to just two spatially localized jets on a single wall, with a zero-net flux constraint, as opposed to full spatial control of both walls. We feel that this is much closer to experimental realizability than an approach with control authority everywhere on the wall. A recent review of the application of deep RL applied to fluid mechanics problems is presented in Viquerat et al. [216].

In these active flow control problems, deep RL possesses the advantageous property of

being completely data-driven, allowing it to discover novel and nontrivial control strategies in complex systems from just data alone without the need to analytically derive or hard-code system properties into the method. However, the training portion of RL is a major bottleneck, requiring a tremendous number of interactions with the target environment to find an approximately optimal policy [47]. Practically speaking, this can correspond to running an enormous number of high-resolution simulations or flow experiments, both of which may be prohibitively expensive.

In the present work, we apply RL to control a minimal flow unit (MFU) (the smallest domain that sustains turbulence) [105] of plane Couette flow at $Re = 400$ using a pair of streamwise-aligned slot jets at one wall, with a no-net-flux constraint. Therefore there is only one degree of freedom for actuation. We select this system because the unactuated flow isolates the self-sustaining regeneration cycle of wall-bounded turbulence [83, 100]. This case is well-studied for tasks such as reduced-order modeling [66, 131, 145, 226], finding invariant solutions [68, 219], and applying opposition control [99]. We refer the reader to Ch. 6 for a more detailed review of this system.

In order to overcome the high computational cost of RL training in this environment, in this work we replace the high-resolution simulation with an accurate low-dimensional surrogate model, aiming to dramatically reduce the time required to train the control policy. We showed in [244] that this data-driven model-based RL approach, which we refer to as "Data-Driven Manifold Dynamics" RL (DManD-RL), works well for controlling spatiotemporal chaotic dynamics in the Kuramoto-Sivashinksy Equation. For further discussion on the various types of model-based RL, we refer the reader to Zeng et al. [244]. In Sec. 7.2 we introduce the control environment and the DManD-RL framework. Then, in Sec. 7.3 we describe the data used for training the DManD model, the performance of the model, and the results of applying RL to the DManD model and to the DNS environment. Finally, we conclude in Sec. 7.4 with a summary of the key results.

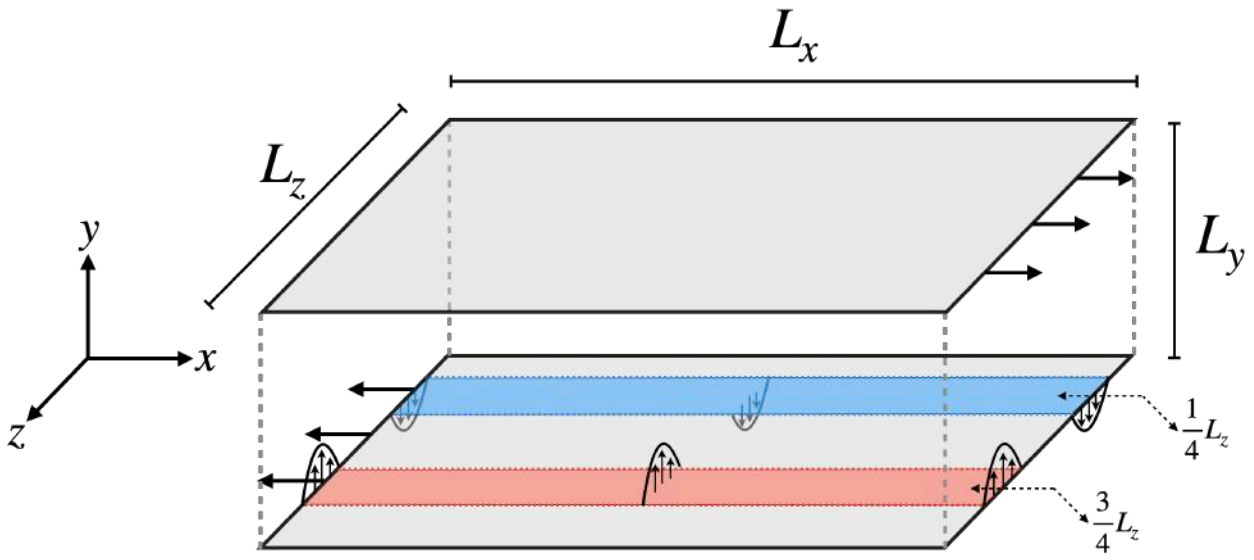


Figure 7.1: Schematic of the Couette flow domain with two slot jets on one wall.

7.2 Framework

7.2.1 Navier-Stokes equation with slot jets

The environment we consider is a direct numerical simulation (DNS) of the Navier-Stokes Equations (NSE)

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = -\nabla p + \mathrm{Re}^{-1} \nabla^2 \mathbf{u}, \quad \nabla \cdot \mathbf{u} = 0. \tag{7.1}$$

The velocities in the streamwise $x \in [0, L_x]$, wall-normal $y \in [-L_y/2, L_y/2]$, and spanwise $z \in [0, L_z]$ directions are defined as $\mathbf{u} = [u_x, u_y, u_z]$, and the pressure is p. Here we have nondimensionalized velocity by the speed U of the walls, length by the channel half-height ($h = L_y/2$), time with h/U and pressure with ρU^2, where ρ is the fluid density. The Reynolds number is $\mathrm{Re} = Uh/\nu$, where ν is the kinematic viscosity. The boundary conditions for this setup are periodic in x and z ($\mathbf{u}(0, y, z) = \mathbf{u}(L_x, y, z), \mathbf{u}(x, y, 0) = \mathbf{u}(x, y, L_z)$), no-slip boundary conditions at the walls ($u_x(x, \pm L_y/2, z) = \pm 1, u_z(x, \pm L_y/2, z) = 0$), no penetration at the top wall ($u_y(x, L_y/2, z) = 0$), and finally, the actuation on the bottom wall ($u_y(x, -L_y/2, z) = f_a(x, z)$), as we now describe.

The actuation on the bottom wall is in the form of two slot jets that are Gaussian in z

and travel the length of the channel:

$$f_a(x,z) = a(t)V_{\max}\left(\exp\left(-\frac{(z-L_z/4)^2}{2\sigma^2}\right) - \exp\left(-\frac{(z-3L_z/4)^2}{2\sigma^2}\right)\right). \tag{7.2}$$

We set $\sigma \approx 0.16$ so that the jets act "locally", and the velocity of the jet is dictated by $a(t)V_{\max}$, where $a(t) \in [-1,1]$ is the instantaneous actuation amplitude scaled by a maximum velocity $V_{\max} = 0.05$. For perspective, the root-mean-squared wall-normal velocity at the channel centerline for turbulent unactuated flow is ~ 0.063. We chose this small velocity to evaluate how the agent performs with limited control authority. In Fig. 7.1 we show a schematic illustrating this system.

The complexity of the flow increases as the Reynolds number and the domain size L_x and L_z increase. Here we chose the same setup as Hamilton et al. [83], $\mathrm{Re} = 400$ and $[L_x, L_y, L_z] = [1.75\pi, 2, 1.2\pi]$. These parameters isolate the "self-sustaining process" (SSP) that drives wall-bounded turbulence. In the SSP, low-speed streaks that have been lifted from the wall become wavy, this waviness leads to the breakdown of the streaks, generating streamwise rolls, and, finally, these rolls lift low-speed fluid off the wall to regenerate streaks, completing the cycle. By working in this well-studied domain that is dominated by the SSP, we can better identify the means by which a control strategy can disrupt or suppress this process.

In this work, the control strategy is to minimize the turbulent drag averaged between both walls

$$D = \frac{1}{2}\int_0^{L_x}\int_0^{L_z}\left(\left.\frac{\partial u_x}{\partial y}\right|_{y=1} - 1\right) + \left(\left.\frac{\partial u_x}{\partial y}\right|_{y=-1} - 1\right) dxdz, \tag{7.3}$$

subject to a quadratic penalty on actuation amplitude a. (If the relation between the pressure drop and actuation velocity for pumping fluid into/out of the domain is linear, then this penalty is proportional to the power consumption of the actuation.) Further details are described in Sec. 7.2.2. We report drag in this fashion because this quantity goes to 0 when the flow laminarizes.

We simulate the flow using a Fourier-Chebyshev pseudo-spectral code we implemented in Python [133], which is based on the *Channelflow* code developed by Gibson et al. ([67, 70]). Linear terms are treated implicitly and the nonlinear term explicitly. The specific time integration schemes we use are the multistage SMRK2 scheme [197] for the first two timesteps after every actuation, and the multistep Adams-Bashforth Backward-Differentiation 3 scheme [164] until the next actuation. The multistep scheme is more computationally efficient, but, because actuations change instantaneously, using previous steps with the incorrect boundary condition would lead to incorrect results. For all trials we evolve solutions forward using $\Delta t = 0.02$ on a grid of $[N_x, N_y, N_z] = [32, 35, 32]$ in x, y, and z from random divergence-free initial conditions that we evolve forward 100 time units so initial conditions are near the turbulent attractor.

While most of this approach is standard, here we include some details on the simulation procedure to highlight explicitly how we set the jet actuation boundary condition. At each time step, the approach involves solving the expression

$$\mathrm{Re}^{-1}\frac{d^2\hat{\mathbf{u}}^{i+1}_{k_x,k_z}}{dy^2} - \lambda\hat{\mathbf{u}}^{i+1}_{k_x,k_z} - \hat{\nabla}\hat{p}^{i+1}_{k_x,k_z} = -\hat{\mathbf{R}}^{i}_{k_x,k_z}, \tag{7.4}$$

where i is the timestep and $\hat{\cdot} = \mathcal{F}_{x,z}(\cdot)$ denotes the Fourier transform in x and z. The variable λ includes the timestep Δt and the x and z components of the diffusive term and $\mathbf{R}$ encompasses all the remaining explicit terms (for a multistep method this includes $\hat{\mathbf{u}}_{k_x,k_z}$ multiple steps back). We refer the reader to [67] for a more detailed discussion. Upon taking the divergence of Eq. 7.4, and accounting for incompressibility, we isolate the problem down to 4 sets of one-dimensional Helmholtz equations (for conciseness we suppress indices k_x, k_z, and i):

$$\mathrm{Re}^{-1}\frac{d^2\hat{u}_x}{dy^2} - \lambda\hat{u}_x - \frac{2\pi i k_x}{L_x}\hat{p} = -\hat{R}_x \qquad \hat{u}_x(\pm 1) = \pm\delta_{k_x,0}\delta_{k_z,0} \tag{7.5}$$

$$\mathrm{Re}^{-1}\frac{d^2\hat{u}_z}{dy^2} - \lambda\hat{u}_z - \frac{2\pi i k_z}{L_z}\hat{p} = -\hat{R}_z \qquad \hat{u}_z(\pm 1) = 0 \tag{7.6}$$

$$\mathrm{Re}^{-1}\frac{d^2\hat{u}_y}{dy^2} - \lambda\hat{u}_y - \frac{d\hat{p}}{dy} = -\hat{R}_y \qquad \hat{u}_y(-1) = \mathcal{F}_{x,z}(f_a), \quad \hat{u}_y(1) = 0 \tag{7.7}$$

$$\frac{d^2\hat{p}}{dy^2} - 4\pi^2\left(\frac{k_x^2}{L_x^2} + \frac{k_z^2}{L_z^2}\right)\hat{p} = \hat{\nabla}\cdot\hat{\mathbf{R}} \qquad \frac{d\hat{u}_y}{dy}(\pm 1) = 0. \tag{7.8}$$

These equations can be solved for every wavenumber pair k_x and k_z. The challenge in solving these equations is due to the coupling in Eq. 7.7 and Eq. 7.8. The pressure is coupled to the wall-normal velocity because an explicit boundary condition is unknown. Instead, from incompressibility, we know $d\hat{u}_y/dy(\pm 1) = 0$, which we substitute for the pressure boundary condition. To solve these coupled equations we use the influence matrix method and tau correction developed by Kleiser and Schumann [115]. Although we set the wall-normal boundary condition in Eq. 7.7 by the slot jets in Eq. 7.2, we note that it is simple to replace this boundary condition with any shape of actuation.

7.2.2 Data-driven framework

The objective in deep RL is to train an agent, commonly a neural network, to approximate the optimal control policy $a = \pi^*(s)$, which given a state observation s, outputs the optimal control action a. The optimal policy seeks to maximize the expected long time discounted cumulative reward,

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{l=0}^{\infty}\gamma^l(r_{t+l\tau})\right], \tag{7.9}$$

where $0 < \gamma < 1$ is the discount factor, τ is the time between control actions, and r_t is the reward, a scalar-valued control objective function decided by the user evaluated at time t. As our objective in this work is to minimize the drag of our turbulent Couette system while simultaneously avoiding the use of superfluous control actions, we define the reward function

as the following,

$$r_t = -\left\langle D(t) + c\|a(t)\|^2 \right\rangle_\tau , \tag{7.10}$$

where c is a scalar and $\langle\cdot\rangle_\tau$ is the average from t to $t+\tau$. We note here that our actuation penalty is proportional to the power required for actuation. In deep RL π^* is learned via repeated cyclic interactions between the agent and the environment i.e. the target system. A typical cycle consists of the following: given a state observation of the system at time t, s_t, the control agent outputs its estimated best control response a_t. This control action is then applied to the environment. The system is allowed to evolve for τ time units, and then the impact of the action is quantified by observing the resulting system state, $s_{t+\tau}$, as well as the reward signal, r_t. This iterate of data, [s_t,a_t,r_t,$s_{t+\tau}$], is then stored and used for updating the control agent for the next time interval.

7.2.3 DManD modeling framework

Applications of deep RL often require repeating this cycle $\mathcal{O}(10^{6+})$ times. Because deep RL conventionally requires an online realization of the target system during training, the practicality of training an RL agent for systems that are computationally or experimentally expensive to realize online, e.g. a DNS of turbulent channel flow, is especially bottlenecked by the expense of the environment itself [47].

To circumvent this bottleneck, we employ a method denoted "Data-driven Manifold Dynamics for RL" [244], or "DManD-RL" for short, with some modification. This framework consists of two main learning objectives, which can be broken down into five steps, illustrated in Fig. 7.2. The first objective is to obtain an efficient and accurate low-dimensional surrogate model of the underlying dynamics of the turbulent DNS, which we refer to as the DManD model. This objective is achieved via the first three steps outlined in Fig. 7.2: 1) collect data tuples of the target system experiencing random control actions, 2) obtain a low-dimensional representation of the environment's dynamics, 3) model the dynamics of

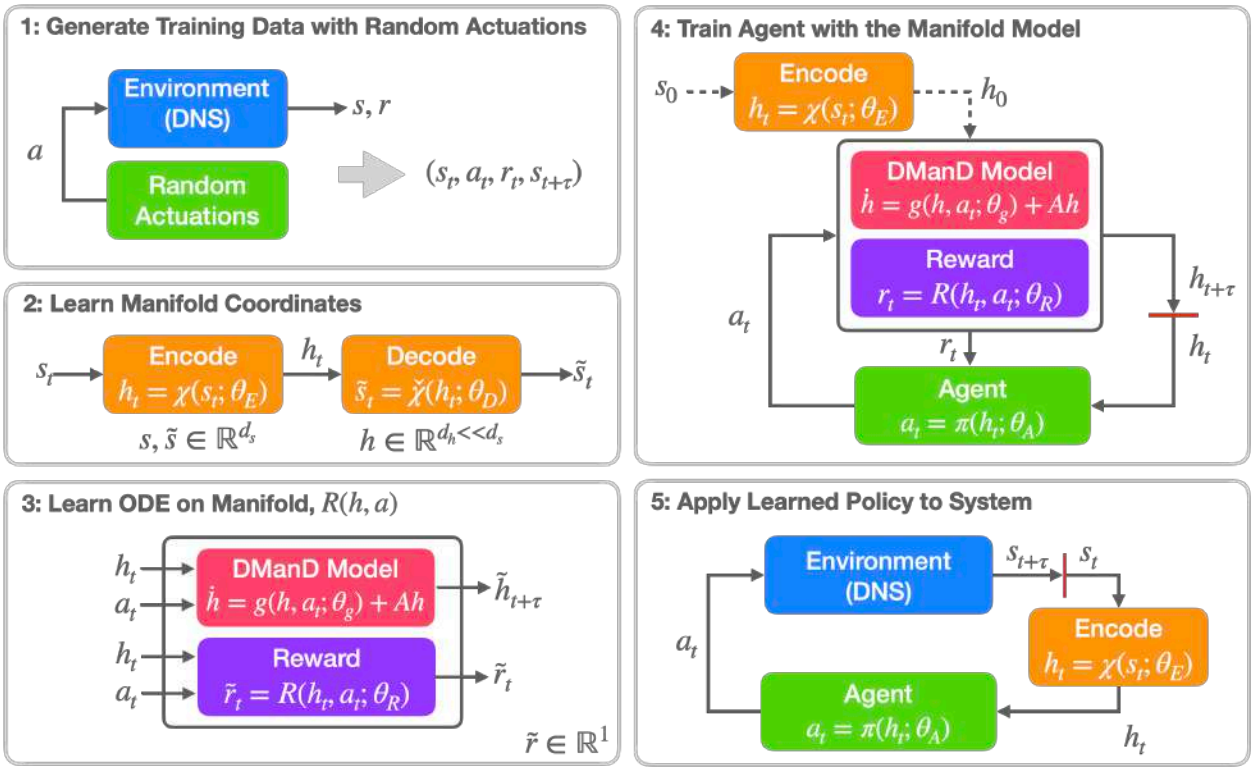


Figure 7.2: Schematic of the DManD-RL framework. After step 3, $\tilde{\cdot}$ is omitted for clarity.

the environment and its response to control inputs.

The second objective is to use this DManD model for RL training to quickly and efficiently obtain an effective control agent. This objective is achieved via the remaining two steps outlined in Fig. 7.2: 4) perform deep RL with the DManD model, and 5) deploy the control agent to the original environment. In the following sections we discuss the details for generating the DManD model in Sec. 7.2.3 and the method for training and deploying the DManD-RL agent in Sec. 7.2.4.

As this framework is completely data-driven, the first step involves collecting sufficient data to learn an accurate surrogate model. This model must capture the underlying flow system, the response of the dynamics to control inputs, and the impact the control inputs have on the objective. Generating this model requires that we have a large dataset that includes the cycle of data described above ($[s_t, a_t, r_t, s_{t+\tau}]$). In RL training actions are chosen by the policy, however, for training the model we do not necessarily have any policy to

generate this data. As such, we instead chose to randomly actuate the flow to generate the original dataset used in training the DManD model. Details on the specifics of the dataset are included in Sec. 7.3.1.

With this data, the second step of the DManD-RL framework involves finding a low-dimensional representation of the state. For many dissipative systems, there is either proof or evidence that the long-time dynamics collapse onto a finite-dimensional invariant manifold [44, 59, 60, 205, 241]. We can define a mapping to coordinates parameterizing this manifold

$$h_t = \chi(s_t), \tag{7.11}$$

where $h_t \in \mathbb{R}^{d_h}$ is the manifold coordinate system and an inverse mapping back to the state

$$s_t = \check{\chi}(h_t). \tag{7.12}$$

When the data lies on a finite-dimensional invariant manifold then the finite-dimensional manifold coordinate representation h_t contains the same information as the state s_t. Thus, if we know χ and $\check{\chi}$ we can simply use h_t in place of s_t for training the RL agent, which requires far fewer degrees of freedom. One subtlety that we gloss over here is that a $d_{\mathcal{M}}$-dimensional manifold may require a set of overlapping local representations called *charts* if one wants to represent the manifold with $d_{\mathcal{M}}$ parameters [57, 61, 120]. However, a manifold with a topological dimension $d_{\mathcal{M}}$ can be embedded in $\mathbb{R}^{2d_{\mathcal{M}}}$ [185, 233]. So, in the worst case, as long as $d_h \geq 2d_{\mathcal{M}}$, a single global coordinate representation can be used, as we do here.

In this work, we will approximate χ and $\check{\chi}$ using an undercomplete autoencoder. This consists of two NNs: an encoder (χ) that reduces the dimension and a decoder ($\check{\chi}$) that expands it. Here we train an autoencoder to find the correction from the linear map given by the proper orthogonal decomposition (POD) [128, 129]. As such we define the state observation s to come from projecting the flow field $\mathbf{u}$ onto a set of POD modes (i.e. s is the POD coefficients). Sec. 7.3.1 includes details on our POD implementation.

For the encoding, we sum the leading d_h POD coefficients with a correction from a NN that is a function of all the POD coefficients:

$$h_t = \chi(s_t; \theta_E) = s_{t,d_h} + \mathcal{E}(s_t; \theta_E), \tag{7.13}$$

where s_{t,d_h} is the first d_h components of s_t and $\mathcal{E}$ is a NN. For the decoding, we want to reconstruct all 500 POD coefficients from the d_h values we have from the encoding. These d_h values are approximately the leading POD coefficients so we can again just add these values to a NN that corrects the leading d_h POD coefficients and reconstructs the remaining POD coefficients:

$$\tilde{s}_t = \check{\chi}(h_t; \theta_D) = [h_t, 0]^T + \mathcal{D}(h_t; \theta_D). \tag{7.14}$$

Here, $[h_t, 0]^T$ represents h_t padded with zeros to the correct size, and $\mathcal{D}$ is a NN. The notation $\tilde{\cdot}$ indicates that this is an approximation of s_t. We refer to this autoencoder structure as a hybrid autoencoder, in contrast to the standard approach of simply treating $\chi(s_t; \theta_E)$ and $\check{\chi}(h_t; \theta_D)$ as NNs. We take this approach because it can achieve lower reconstruction errors [128] than the standard approach and the variables h, and is more interpretable, as a nonlinear correction to POD.

The NNs $\mathcal{E}$ and $\mathcal{D}$ are trained to minimize

$$L = \frac{1}{dK} \sum_{i=1}^{K} ||s_{t_i} - \check{\chi}(\chi(s_{t_i}; \theta_E); \theta_D)||_2^2 + \frac{1}{d_h K} \sum_{i=1}^{K} \xi ||\mathcal{E}(s_{t_i}; \theta_E) + \mathcal{D}_{d_h}(h_{t_i}; \theta_D)||_2^2, \tag{7.15}$$

where $\mathcal{D}_{d_h}$ is the first d_h components of the decoder, ξ is a scalar, and K is the batch size. In this loss, the first term is the reconstruction mean-squared error (MSE), and the second term promotes the accurate reconstruction of the leading d_h POD coefficients. We include this second term because it must go to zero if the reconstruction is perfect. This is because the modification that the NN makes in the encoder must be removed by the NN in the decoder. In Sec. 7.3.2 we provide details on autoencoder training.

Now that we have a low-dimensional representation of the state observation, in step three, we train a model to predict the evolution of h and the reward r_t. To predict the evolution of h (from h_t to $h_{t+\tau}$) we train a "stabilized" neural ordinary differential equation (ODE) [132]

$$\frac{dh}{dt} = g(h, a; \theta_g) + Ah, \tag{7.16}$$

which can be integrated forward in time to predict

$$\tilde{h}_{t+\tau} = h_t + \int_t^{t+\tau} g(h, a; \theta_g) + Ahdt, \tag{7.17}$$

where g is a NN and A is a matrix that can be learned from data or fixed. We chose to find an ODE instead of a discrete timestepping method because it allows us to evolve this equation to arbitrary times that may not align with the sampling rate of our data. This is a highly desirable trait as it means we can freely vary the time between actions after training the DManD model, which could not be done if we found a discrete time map from h_t to $h_{t+\tau}$.

The linear term in Eq. 7.16 is important for stability. Without this term, small errors in the dynamics can lead to linear growth at long times, which a linear damping term prevents [129, 131, 132]. In this work, we set the linear term

$$A_{ij} = -\beta \delta_{ij} \sigma_i(h), \tag{7.18}$$

where $\beta = 0.1$, δ_{ij} is the Kronecker delta, and $\sigma_i(h)$ is the standard deviation of the ith component of h. This term acts as a damping, preventing trajectories from moving far away from the training data. In [131] we show in Couette flow without actuation that this term prevents models from becoming unstable. We note that the addition of this damping term does not negatively impact the accuracy of the vector field in the region of state space where there is data, because the damping term is present when we train the NN. This allows the NN to compensate for the damping term in the region where the data lies. We train the NN

g to minimize

$$J = \frac{1}{d_h K} \sum_{i=1}^{K} ||h_{t_i+\tau} - \tilde{h}_{t_i+\tau}||_2^2. \tag{7.19}$$

We describe in more detail how the gradient of this loss is computed in [131].

The final piece of the algorithm is computing the reward. We could to this directly by mapping h back to $\mathbf{u}$ and computing the corresponding drag. However, this is undesirable because it is computationally expensive, and, in general, it may not be possible to directly compute the reward from the state. To overcome this difficulty, we use a NN to compute an estimate $\tilde{D}$ of the drag from h:

$$\tilde{D} = \mathcal{R}(h; \theta_R), \tag{7.20}$$

which we train to minimize $J_D = 1/K \sum_{i=1}^{K} ||D_{t_i} - \tilde{D}_{t_i}||_2^2$. Then, we compute the reward $\tilde{r}_t = R(h_t, a_t; \theta_R)$ by inserting $\tilde{D}$ into Eq. 7.10.

7.2.4 Reinforcement learning using DManD

With our DManD model of the underlying dynamics, we can now proceed to step four and efficiently obtain a control policy by training an RL agent to interact with the low-dimensional DManD model rather than the expensive DNS. Rather than learning $a_t = \pi(s_t; \theta_A)$, our goal shifts to learning

$$a_t = \pi(h_t; \theta_A), \tag{7.21}$$

where θ_A are the network parameters of the RL agent. In this work, we employ the Soft Actor-Critic (SAC) RL algorithm [82] but we emphasize that the DManD-RL framework works with any general RL algorithm. SAC was chosen in this application because it possesses several advantageous characteristics including the ability to output control signals from a continuous range, an off-policy formulation that allows the "reuse" of previously generated data, and twin critic networks to aid the brittleness commonly associated with many off-

policy deep RL algorithms. Distinctively, SAC has a stochastic actor with an additional entropy-maximizing formulation. This formulation modifies the typical RL objective of Eq. 7.9 to the following

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{l=0}^{\infty} \gamma^l \left(r_{t+l\tau} + \alpha\mathcal{H}(\pi(\cdot|s_t))\right)\right], \tag{7.22}$$

where the entropy of the policy, $\mathcal{H}$, is defined as

$$\mathcal{H}(\pi(\cdot|s_t)) = -\log(\pi(\cdot|s_t)). \tag{7.23}$$

Here α is the trade-off coefficient, which is set to 1.0. This entropy-regularized objective promotes wider state-action exploration and the ability to invest in multiple modes of near-optimal strategies in addition to maximizing the cumulative reward. In this work, the agent is trained stochastically but deployed deterministically during testing. Our implementation of SAC utilizes NNs to approximate the policy function (i.e. agent) π, the two critic functions, Q_1 and Q_2, and the value function V. For more details regarding the derivation and implementation of SAC, we refer the reader to Haarnoja et al. [82].

With a trained DManD-RL agent, we can proceed to the fifth and final step and deploy the control agent to the original turbulent channel DNS for application. As the RL agent learned its control strategy by observing the low dimensional manifold coordinate system, we must map the high-dimensional state observations made in the DNS to the proper input by using the previously obtained encoder function, χ,

$$a_t = \pi(\chi(s_t)). \tag{7.24}$$

The trained agent can now be deployed in a closed-loop control fashion.

7.3 Results

7.3.1 Description of data

The first step in DManD-RL is to generate a data set for training models. Our data set consists of 500 different initial conditions, which we evolved forward 300 time units with a random actuation chosen every 5 time units. We sample the random actuations uniformly between -1 and 1. Every 1 time unit we record the velocity field $\mathbf{u}$ and the action a, yielding $1.5 \cdot 10^5$ snapshots of data. We use 80% of this data to train the models, and the remaining 20% to test performance on the data never previously seen by the model.

Due to the high-dimensional nature of the data, learning a manifold coordinate system using the velocity field on the grid as the state is challenging. As such, we first preprocess the data using the proper orthogonal decomposition (POD) to reduce the dimension from $\mathcal{O}(10^5)$ to 500 and treat these 500 POD coefficients as the state observation s. In the POD, we find modes $\mathbf{\Phi}$, which we project onto to maximize

$$\frac{\left\langle |(\mathbf{u}', \mathbf{\Phi})|^2 \right\rangle}{||\mathbf{\Phi}||^2}, \tag{7.25}$$

which uses the fluctuating velocity $\mathbf{u}' = \mathbf{u} - \langle \mathbf{u} \rangle$ ($\langle \cdot \rangle$ denotes the average). As shown in [92, 193], these modes can be found by solving the following eigenvalue problem:

$$\sum_{j=1}^{3} \int_0^{L_x} \int_{-1}^{1} \int_0^{L_z} \langle u_i'(\mathbf{x}, t) \bar{u}_j(\mathbf{x}', t) \rangle \, \Phi_j^{(n)}(\mathbf{x}') \, d\mathbf{x}' = \lambda_i \Phi_i^{(n)}(\mathbf{x}), \tag{7.26}$$

where $\bar{\cdot}$ is the complex conjugate. Naïve implementation of POD requires solving a computationally expensive $d \times d$ eigenvalue problem after approximating these integrals. This formulation also fails to respect the streamwise translation invariance of our system.

We account for this translational invariance, and make the problem tractable, by exploiting the fact that in translation invariant directions POD eigenfunctions take the form of

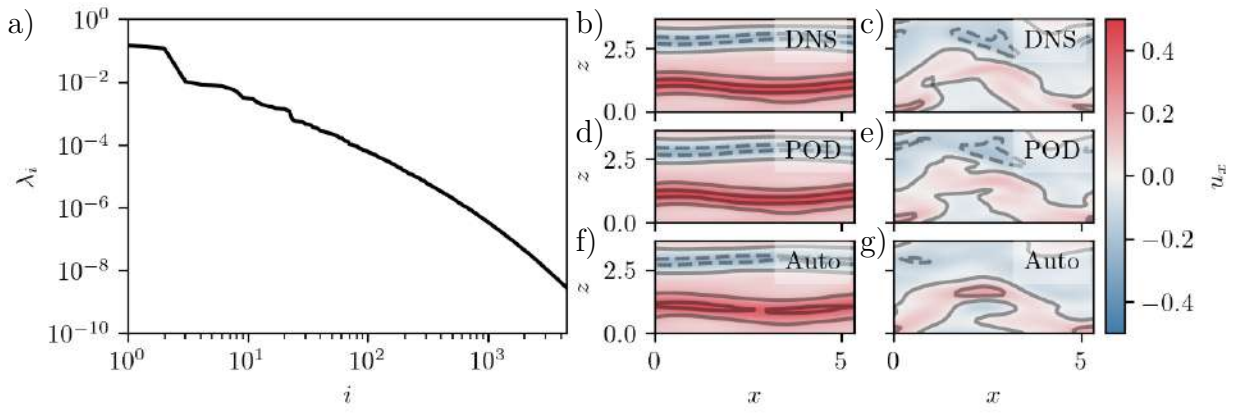


Figure 7.3: (a) Eigenvalues from the POD. (b) and (c) Snapshots of the centerline streamwise velocity ($u_x(y=0)$) from the DNS at representative low-drag (left) and high-drag (right) instants. (d) and (e) POD reconstruction with 500 modes of the DNS results in (b) and (c). (f) and (g) Autoencoder reconstruction of (d) and (e). Solid contour lines are positive and dotted contour lines are negative.

Fourier modes [92]. Note that the fixed position of the slot jets breaks translation invariance in z. This turns the eigenvalue problem in Eq. 7.26 into

$$L_x \sum_{j=1}^{3} \int_{-1}^{1} \int_{0}^{L_z} \left\langle \hat{u}_i'(k_x, y', z', t) \bar{\hat{u}}_j'(k_x, y', z', t) \right\rangle \varphi_{jk_x}^{(n)}(y', z')\, dy' dz' = \lambda_{k_x}^{(n)} \varphi_{ik_x}^{(n)}(y, z). \tag{7.27}$$

This is a $3N_yN_z \times 3N_yN_z$ eigenvalue problem for every wavenumber k_x. We speed up the computation of this eigenvalue problem by evenly sampling $5,000$ snapshots of training data.

Solving the eigenvalue problem in Eq. 7.27 results in eigenvectors

$$\mathbf{\Phi}_{k_x}^{(n)}(\mathbf{x}) = \frac{1}{\sqrt{L_x}} \exp\left(2\pi i \frac{k_x x}{L_x}\right) \boldsymbol{\varphi}_{k_x}^{(n)}(y, z), \tag{7.28}$$

and eigenvalues $\lambda_{k_x}^{(n)}$. We then project $\mathbf{u}$ onto the leading 305 modes, sorted according to the magnitude of λ, giving us the state observation s. This results in a 500-dimensional state because a majority of the modes are complex. Figure 7.3a shows the sorted eigenvalues. The eigenvalues drop off quickly resulting in the first 305 modes containing 99.8% of the energy.

7.3.2 Manifold coordinate system

Now that we reduced the dimension of the state observation s, we train an autoencoder as described in Sec. 7.2.3 to find h. In Linot and Graham [131] we varied d_h for the unactuated Couette flow system and found the DManD models to be highly accurate with fewer than 20 degrees of freedom. In the present system, additional degrees of freedom will be excited by the actuations, so we increased d_h slightly, to $d_h = 25$. For training, we first normalized s by subtracting the mean and dividing by the maximum standard deviation. We then trained four autoencoders until the training error stopped improving. We selected the autoencoder based on which yielded the best forecasting performance when coupled to a neural ODE described in the following section. The selected autoencoder had a test MSE of $1.45 \cdot 10^{-4}$. The architecture and parameters of the autoencoders are reported in Table 7.1.

In Figs. 7.3b-7.3g we compare the centerplane streamwise velocity of two flowfields – one in a bursting state (high-drag) and the other in a hibernating (low-drag) state – to their reconstruction with 305 POD modes and their reconstruction from the autoencoder with $d_h = 25$. When reconstructing the simpler hibernating state the reconstruction from POD and the autoencoder match almost exactly. In the case of the bursting snapshot, both the POD and the autoencoder still match the full high-dimensional state well, accurately capturing the magnitude and location of the streamwise velocity. In the case of POD, some error is introduced as it distorts the details in the DNS, and then the autoencoder further smooths some of these details. However, considering that we reduced the dimension of the problem from $\mathcal{O}(10^5)$ to 25, the reconstruction is excellent. Now that we can represent the state s in the manifold coordinate system as h, the next step is learning a time evolution model to control.

7.3.3 DManD performance

We now describe the neural ODE training and performance of the resulting model. Before training, we normalize h by subtracting the mean and dividing each component by its

Table 7.1: Architectures of NNs. "Shape" indicates the dimension of each layer, "Activation" the corresponding activation functions, and "sig" is the sigmoid activation."Learning Rate" gives the learning rate for the Adam optimizer [113]. When multiple learning rates are noted, the value was changed from one value to the next at even intervals during training.

Function	Shape	Activation	Learning Rate
$\mathcal{E}$	$500/1000/d_h$	sig/lin	$[10^{-3}, 10^{-4}]$
$\mathcal{D}$	$d_h/1000/500$	sig/lin	$[10^{-3}, 10^{-4}]$
g	$d_h/200/200/200/200/d_h$	sig/sig/sig/sig/lin	$[10^{-2}, 10^{-3}, 10^{-4}]$
$\mathcal{R}$	$d_h/100/100/1$	sig/sig/lin	$[10^{-3}]$
$\mathbb{O}$	$1024/100/100/d_h$	sig/sig/lin	$[10^{-3}]$
π	$d_h/256/128/128/2/1$	ReLU/ReLU/ReLU/lin/Tanh	$[3 \cdot 10^{-4}]$
Q_1	$d_h/256/128/128/1$	ReLU/ReLU/ReLU/lin	$[3 \cdot 10^{-3}]$
Q_2	$d_h/256/128/128/1$	ReLU/ReLU/ReLU/lin	$[3 \cdot 10^{-3}]$
V	$d_h/256/128/128/1$	ReLU/ReLU/ReLU/lin	$[3 \cdot 10^{-3}]$

standard deviation. In Table 7.1 we show the architecture for the neural ODEs used in this section. We trained four neural ODE models for each of the four autoencoders. Then we selected the autoencoder and neural ODE pair that best reconstructed the statistics we report below. Again, we trained the neural ODEs until we no longer saw an improvement in performance.

The first statistics we investigate validate the ability of the model to track the true dynamics over short times. In Fig. 7.4 we show an example of a randomly actuated trajectory from the DNS and reconstructed with the DManD model. In the first two snapshots, the trajectories are in quantitative agreement, after which the trajectories still appear qualitatively quite similar. Both the DNS and the DManD model exhibit the streak breakdown and regeneration cycle over this series of snapshots. At $t = 0$ rolls are forming, at $t = 35$ these rolls lift low-speed fluid off the wall forming streaks, at $t = 70$ the streaks become wavy, and, finally, at $t = 105$ the streaks have broken down.

Next, we consider model performance when averaged over many trajectories. In all of these cases we show both the error from time evolving with the DManD model, and the error incurred due to the autoencoder. We compute the error due to the autoencoder by inputting the true DNS solution through the autoencoder and computing the relevant statistic. Figure

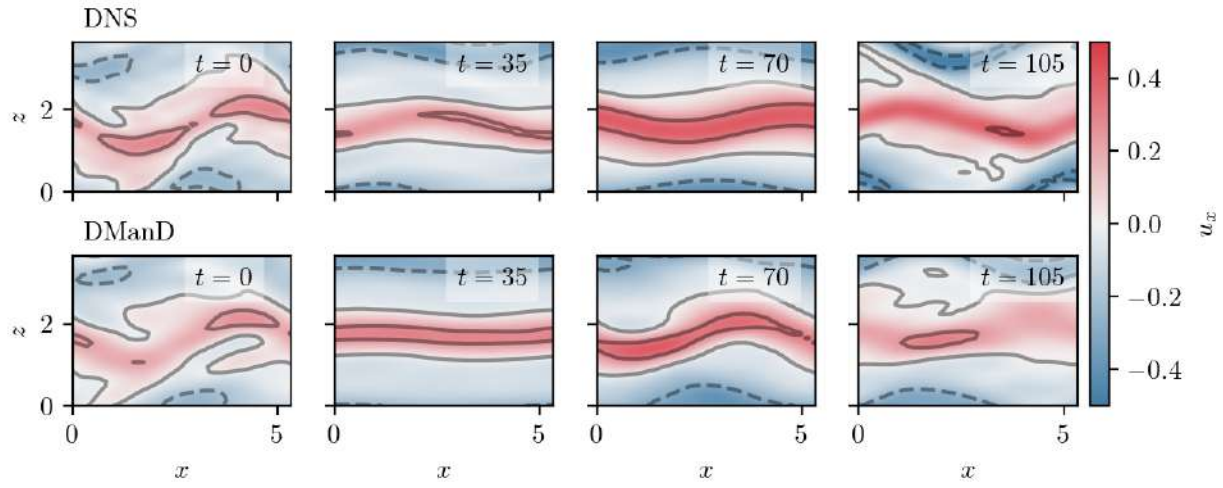


Figure 7.4: Comparison of snapshots from a randomly actuated trajectory from the DNS (top) with the DManD reconstruction (bottom). Times are given at the top right of each image.

7.5a is a plot of the normalized ensemble-averaged tracking error as a function of time. We compute this error by finding the difference between $s(t)$ and $\tilde{s}(t)$ from the DManD model for 100 initial conditions. Then, we normalize this error by computing the difference between two states on the attractor at random times t_i and t_j $N = \left\langle ||s_{t_i} - s_{t_j}|| \right\rangle$. With this normalization, the long-time dynamics of two slightly perturbed initial conditions with different random actuation sequences should approach unity. The error at $t = 0$ represents the discrepancy between the full state and its reconstruction when passed through the autoencoder. The ensembled-averaged tracking error rises steadily at one slope for the first 50 time units and then at a lower slope after that. Once the curve levels off, the true and model trajectories have become uncorrelated. Based on these results, the model tracks well for ~ 50 time units. For reference, the Lyapunov time (inverse of the Lyapunov exponent) for the unactuated system is $\tau_L = 48$ time units [100].

In addition to the tracking error, we also check the ability of the DManD model to capture the temporal autocorrelation of the kinetic energy. To compute this autocorrelation we take the instantaneous kinetic energy of the flow

$$E(t) = \frac{1}{2L_xL_z} \int_0^{L_z} \int_{-1}^{1} \int_0^{L_x} \frac{1}{2} \mathbf{u} \cdot \mathbf{u} \, d\boldsymbol{x}, \tag{7.29}$$

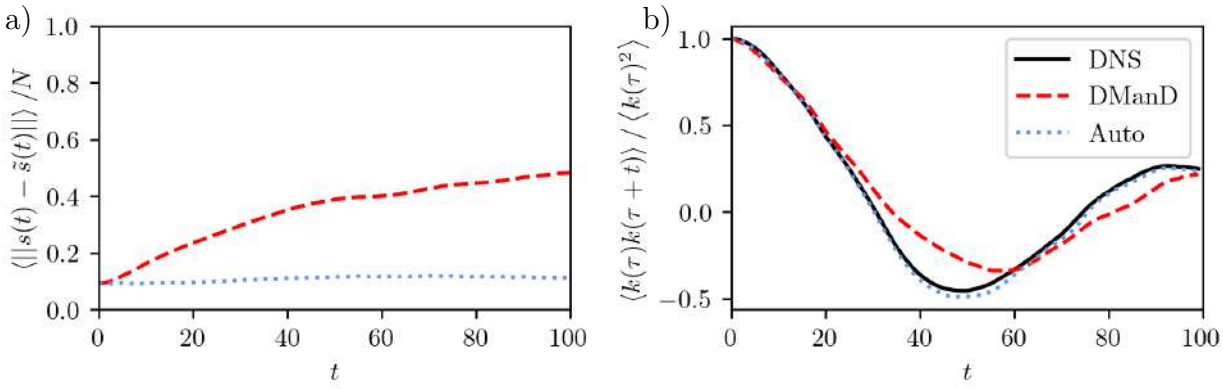


Figure 7.5: (a) Ensemble averaged tracking error and (b) temporal autocorrelation of kinetic energy for the DNS, the DManD model, and the autoencoder. "Auto" represents putting the DNS trajectories through the autoencoder without any time prediction: i.e. this represents the error of just reducing, then expanding the dimension.

and subtract the mean to yield $k(t) = E(t) - \langle E \rangle$. Figure 7.5b shows this temporal autocorrelation computed from the DNS, the DManD model, and the autoencoder. We see the temporal autocorrelation of the model matches the true temporal autocorrelation closely over the first ~ 30 time units.

Due to the chaotic nature of this system, short-time tracking is limited by the Lyapunov time. Nevertheless, a good model should still be able to capture the long-time statistics of the original system. In Fig. 7.6 we show the four components of the Reynolds stress for the DNS, the DManD model, and the autoencoder. We computed these statistics by averaging over the entire testing dataset (100 trajectories 300 time units in length). For all these quantities, the DManD model is in excellent agreement with the DNS. In the cases of $\langle u_y^2 \rangle$ and $\langle u_z^2 \rangle$, surprisingly, the autoencoder appears to perform worse than the DManD model at matching the DNS. With perfect prediction of the DManD model, it would exactly match the autoencoder. This slight disagreement indicates that some of the states in the manifold coordinates (h) are driven somewhat outside the expected range of values.

All of the statistics shown so far indicate that the DManD model accurately captures the dynamics of the randomly actuated DNS with only $d_h = 25$ degrees of freedom. The last step before using this model in the RL framework is training the reward network as

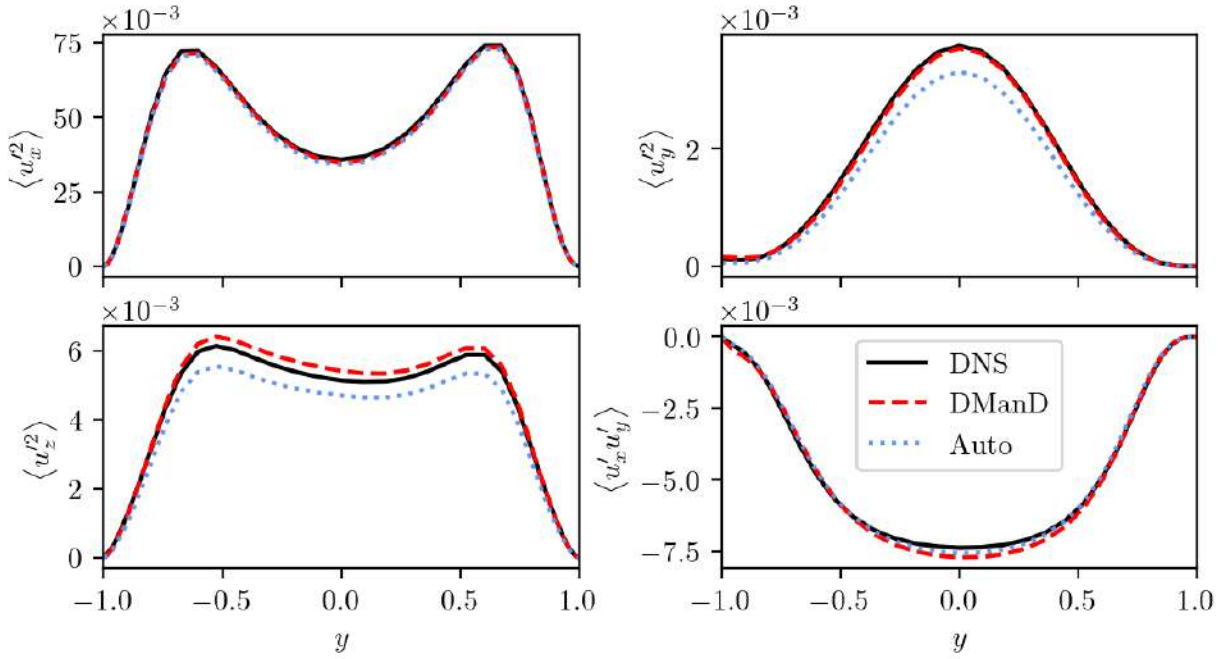


Figure 7.6: Four components of the Reynolds stress for the DNS, the DManD model, and the autoencoder.

described in Sec. 7.2.3. We trained a NN with the architecture in Table 7.1. In Fig. 7.7a we show a PDF of the parity plot between the true and predicted drag on test data, and report the MSE on the normalized data. The excellent agreement indicates we can compute accurate values of the reward directly from h.

Finally, although we computed the manifold coordinate system directly from the state, a natural extension of this work is to use a more limited set of observations. As a first step in this direction, we trained a NN $\mathbb{O}(\cdot)$ to map 32×32 wall shear rate observations at the bottom wall to the manifold coordinate system:

$$\tilde{h} = \mathbb{O}(\partial_y u_x|_{y=-1}; \theta_O). \tag{7.30}$$

The details of this NN are included in Table 7.1. This mapping allows us to test if it is possible to directly use wall observables that are experimentally realizable with our control policy. In Fig. 7.7b we show the parity plot of reconstructing h from wall observables and

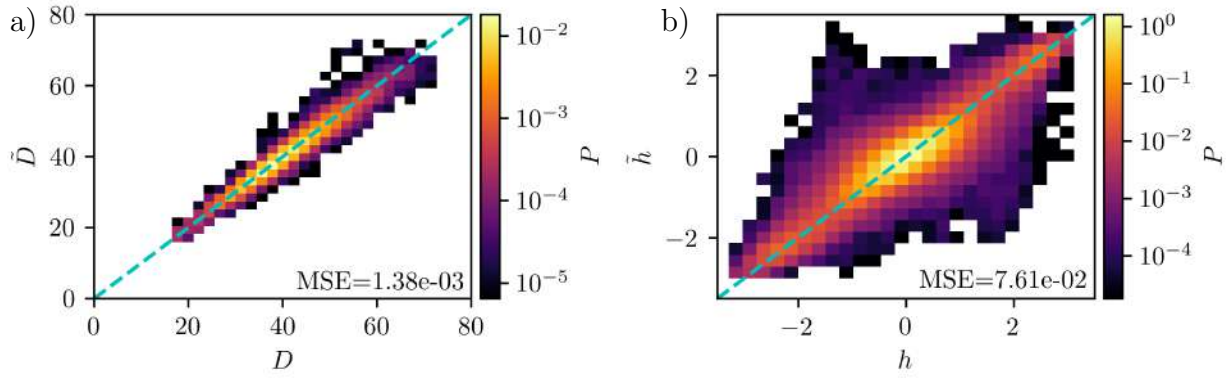


Figure 7.7: (a) Joint PDF of the true (D) and predicted drag ($\tilde{D}$) from the reward network, Eq. 7.20. (b) Joint PDF of the true (h) and predicted ($\tilde{h}$) low-dimensional state from the observation network, Eq. 7.30. Note the logarithmic scales. The cyan line indicates perfect reconstruction, and the MSE is for D and h with the mean subtracted and divided by the standard deviation.

report the MSE for normalized data. While the parity plot is not as sharply peaked as we might like, note that it is shown on a logarithmic scale.

7.3.4 DManD-RL performance

Now with an efficient and low-dimensional model of the underlying dynamics of the turbulent flow in hand, we can quickly obtain a control agent by performing deep RL on the DManD model rather than the original costly DNS. In this work, we employ the Soft Actor-Critic (SAC) RL algorithm [82], which requires training a policy function (i.e. agent) π, two critic functions, Q_1 and Q_2, and a value function V. The RL networks are trained for 10,000 episodes, with each episode consisting of DManD model trajectories of 300 time units. The initial condition for each episode is selected at random from on-attractor turbulent states. Here we choose the action time to be $\tau = 5.0$ and $\gamma = 0.99$. Accordingly, the discount factor over one Lyapunov time $\tau_L \approx 48$ of the unactuated system is $\gamma^{\tau_L/\tau} \approx 0.9$.

Once trained, we apply the DManD-RL agent to the original turbulent DNS, with which it has never directly seen or interacted. To deploy the agent for control, we insert the already-trained encoder, χ, between the agent and the environment to map state observations of the

turbulent DNS to the manifold representation, h, as this is the observation space where the DManD-RL agent was trained.

Shown in Fig. 7.8 are times series, of length 900, of the drag in various cases from DNS trials using 25 different unseen initial conditions. Here we define the % drag reduction,

$$DR = \frac{D_0 - D}{D_0} \cdot 100\%, \tag{7.31}$$

where D_0, and D is the time averaged drag experienced for the 25 test trajectories over 900 time units under no control and control, respectively. In Fig. 7.8a we plot the drag of the 25 test trajectories in the absence of control and highlight that the system predominately remains turbulent, with a few laminarizations. In Fig. 7.8b we show that utilizing a random jet actuation policy does not reduce drag and even results in increased drag $DR = -4\%$ across the 25 trajectories over the 900 time unit window. In Fig. 7.8c we show that our DManD-RL agent is able to significantly reduce the drag of the turbulent DNS and laminarize 21/25 turbulent initial conditions, yielding $DR = 48\%$. In comparison, we show in Fig. 7.8d that the conventionally trained RL agent, which was trained directly on the turbulent DNS, results laminarization of 23/25 test trajectories with $DR = 56\%$, similar to that of the DManD-RL method. We highlight here, however, that the DManD-RL control agent was obtained at a small fraction of the computational cost compared to its conventional counterpart. For reference, the DManD-RL agent required $\sim$ 3.7s per training episode, while a conventional application of deep RL required $\sim$ 1630s per episode, on a 2.40GHz Intel Xeon CPU E5-2640 v4. This corresponds to a 440 times speedup in training time.

Finally, we now limit the state observation of the DManD-RL agent to information observable at the wall i.e. wall shear rate, by pairing the DManD-RL agent trained for Fig. 7.8c with the observation network ($\mathbb{O}(\partial_y u_x|_{y=-1}; \theta_O)$). We demonstrate in Fig. 7.8e that the agent with only access to wall observations, $a_t = \pi(\mathbb{O}(\partial_y u_x|_{y=-1}; \theta_O); \theta_A)$, performs just as well as its counterparts, with 20/25 test trajectories laminarizing and $DR = 50\%$ over the

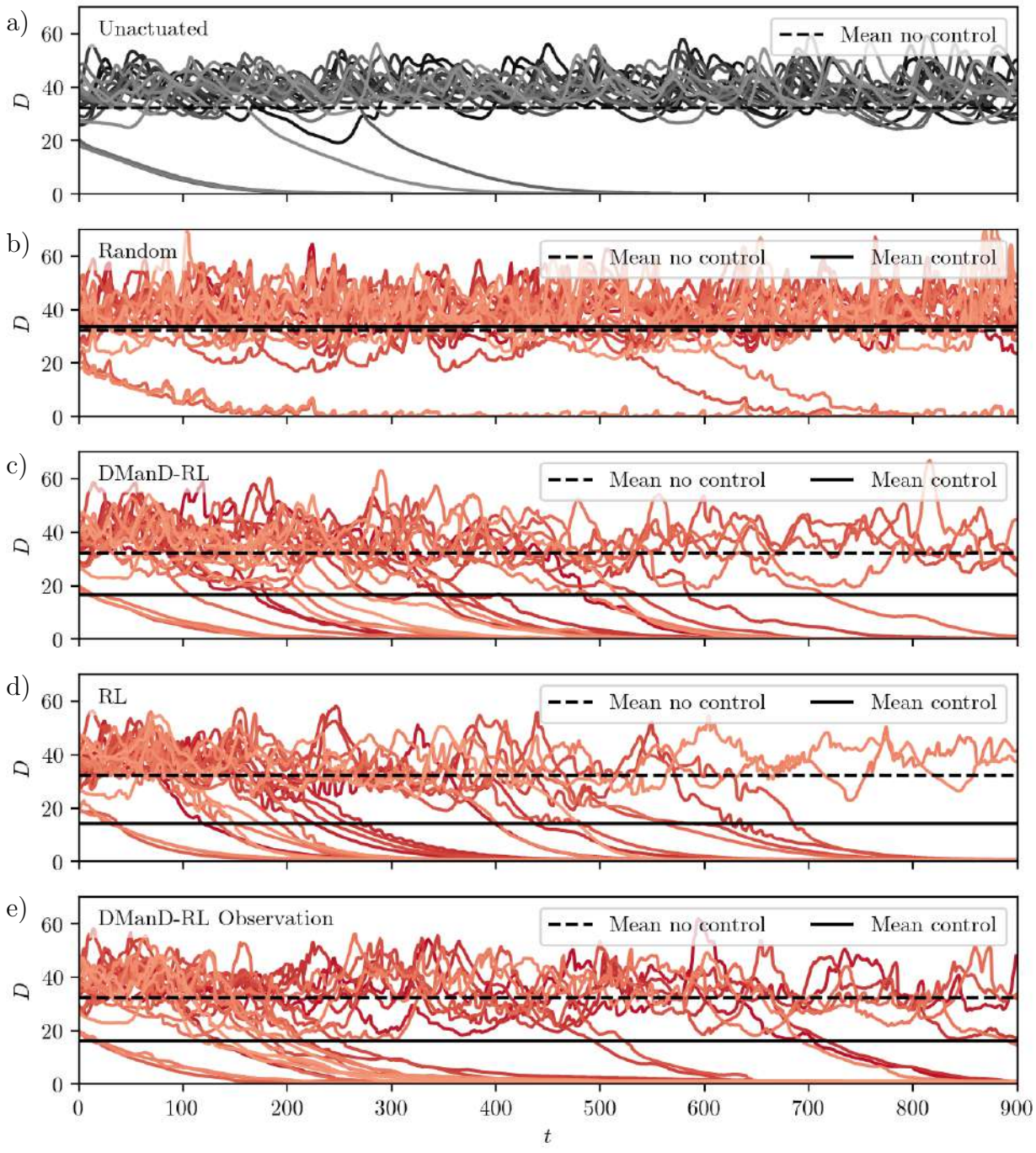


Figure 7.8: Trajectories beginning from test initial conditions with (a) no control, (b) random actuations, (c) DManD-RL control, (d) DNS-based RL control, and (e) DManD-RL control using wall observations. Each figure shows 25 test trajectories.

900 time unit window. Here we note that DR is influenced by how many and how quickly test trajectories laminarize, as well as how turbulent transient trajectories are attenuated by the various controllers and we emphasize that all of the RL controllers perform similarly.

Interpretation of the mechanism of drag reduction

Given the observed effectiveness of the control policy discovered by the RL algorithm, it is desirable to understand how the controller is modifying the flow. In this section, we describe two sets of observations that may shed some light on this issue. The first focuses on the control action in the time preceding a laminarization event. Shown in Fig. 7.9a is a time series depicting the drag and left actuator control signal of a DNS controlled by the DManD-RL agent. Fig. 7.9b shows the wall actuation and an isosurface of streamwise velocity illustrating the streamwise streak structure at $t = 149$, indicated by the first red dot on the drag time series in fig:Mechanism1-a. Here the left slot jet is sucking, and the right blowing, drawing the fluid (and low-speed streak indicated by the isosurface) to the left. At $t = 179$ (second red dot and Fig. 7.9c), the streak is now located over the left jet. Once the streak is above the left jet, the agent executes a series of actions that destabilize the low-speed streak, causing it to break down, shown in Fig. 7.9d. In the wake of the collapse, the agent initiates a strong actuation that leads to the formation of *two* low-speed streaks, shown in Fig. 7.9e. This double low-speed streak structure then proceeds to decay to the laminar state, with the agent applying weak, attenuating control actions to expedite the process.

A DNS initialized with this double-streak structure flow field in the absence of control results in the natural laminarization of the flow. In wall-bounded turbulence, streaks take on a characteristic spacing of 100 wall units [192], which is approximately the width of the MFU cell [83, 105]. The two-streak state has a spanwise length scale that is too small and thus too dissipative to self-sustain, leading to breakdown in the SSP and laminarization of the flow. In approximately half of the trials that laminarized, this double-streak structure

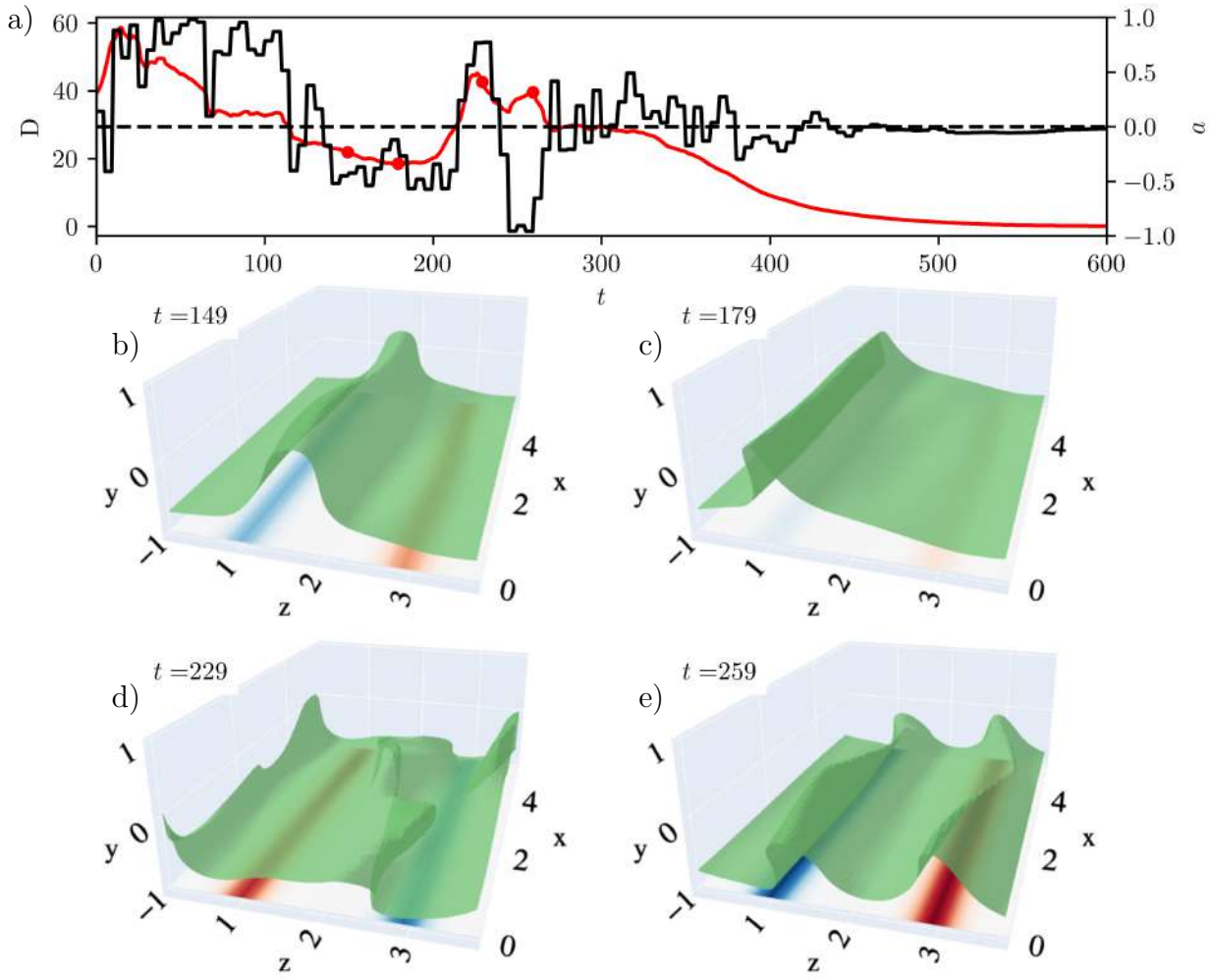


Figure 7.9: (a) Time series of drag (red) and actuation signal of the left jet (black) for an example DNS trajectory controlled by the DManD-RL agent. (b)-(e) snapshots of the trajectory at times marked in (a). These snapshots include an isosurface of $u_x = -0.35$ and the jet actutations (red is fluid injection, blue is fluid suction).

appeared before laminarization.

This is a very interesting and counterintuitive strategy, particularly because it is intrinsically *nonlinear*. At any given instant, the two-jet system can only drive a wall-normal flow with the same fundamental wavelength as the domain. But here the RL agent implements a time-dependent policy that ends up generating a flow structure with, roughly speaking, half the wavelength of the domain. I.e. it has figured out a way to drive structure to a smaller scale, where viscosity can take over and drive the flow to laminar.

The second set of observations we highlight addresses the relationship between the RL control policy and a flow control strategy, widely-studied with simulations, that we mentioned in Sec. 7.1 – opposition control. In opposition control, the entire wall-normal velocity field at both walls is set to have the opposite sign as the wall-normal velocity at an $x-z$ "detection plane" located at some y near the wall. We highlight that this method possesses much greater control authority than in the present work, as it has full spatial control at both walls whereas here we only actuate two spatially localized slot jets on a single wall. Furthermore, our controller holds a constant actuation for multiple time units, whereas opposition control updates on the time scale of a time step.

With these differences in mind, we investigate trajectories controlled by the DManD-RL policy in the context of potential similarities to opposition control. To make this comparison, we first must characterize the wall-normal velocity at the detection plane, but because the jets span the length of the channel this velocity varies. We take a characteristic detection-plane velocity to be the wall-normal velocity averaged over a jet weighted by the shape of the jet, denoted $\langle u_y \rangle_J$.

In Fig. 7.10 we plot a PDF of the maximum jet velocity v_J and the wall-normal velocity $\langle u_y \rangle_J$ (for both jets separately) at a detection plane of $y^+ \approx 10$ (Fig. 7.10a) and $y^+ = 33$ (Fig. 7.10b). We remove laminarization events from this dataset by omitting states with a drag less than 10 because we are interested in the control behavior while the flow is still turbulent. In these plots, we denote opposition control with a unit gain ($v_J = -\langle u_y \rangle_J$) with the cyan

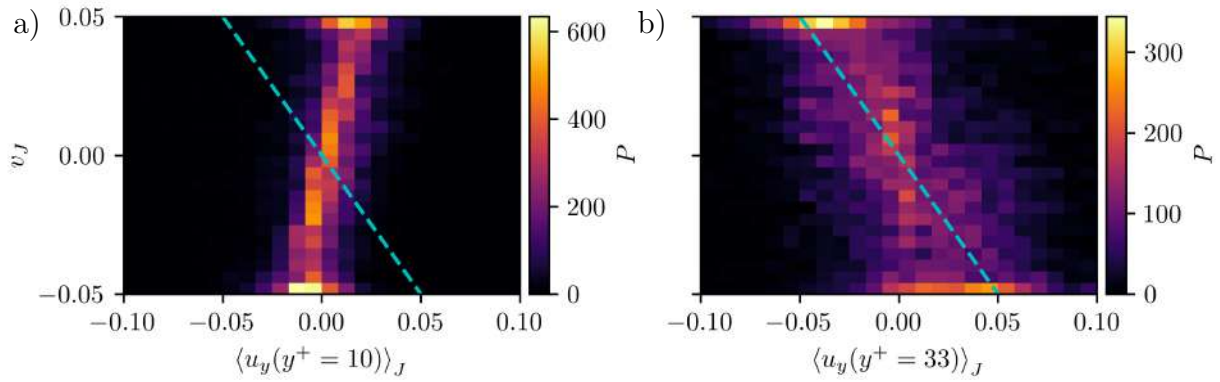


Figure 7.10: Joint PDF of max jet velocity and average wall-normal velocity at (a) $y^+ \approx 10$ and (b) $y^+ = 33$ (the cyan dotted line is $v_J = -\langle u_y \rangle_J$).

line. If the agent performs opposition control, the joint PDF should show a tight distribution with a negative slope. At the detection plane $y^+ \approx 10$ that has been reported to be optimal for opposition control, [28] the RL agent behaves like an anti-opposition controller because there is a positive slope in the PDF. However, at a higher detection plane of $y^+ = 33$, the control agent has a high probability of actuating with sign opposite to $\langle u_y \rangle_J$, as in opposition control. We highlight that the PDF is quite broad, which indicates although the control is opposition-like, the control response is much more complex and diverse than that of opposition control.

Furthermore, we can directly compare our results to Ibrahim et al. [99], who applied standard opposition control to this same Couette domain. In their work, the authors selected a detection plane of $y^+ = 10$ and varied the wall-normal velocity scale $\phi = [0.1, 0.2, 0.5, 1]$. For these parameters, the authors found that the probability of the flow remaining turbulent after 900 time units of control, the length of our test trajectories shown in Fig. 7.8, to be approximately $[0.77, 0.72, 0.69, 0.42]$. For comparison, with 21/25 trajectories laminarizing, our DManD-RL agent's probability of turbulence is 0.16. This improvement is extremely promising for the future use of RL in controlling turbulent flows, especially as the DManD-RL agent had major and realistic restrictions placed on its control authority when compared to opposition control. We also note that this improvement in control further supports the claim

that the agent learned a much more complex and diverse control strategy than opposition control.

7.4 Summary

In this chapter, we efficiently obtained a control strategy from a limited data set to reduce the drag in a turbulent Couette flow DNS via the control of two streamwise slot jets using the DManD-RL framework. Using a combination of POD and autoencoders, we extracted a low-dimensional manifold representation of the data, whose dynamics we modeled using a neural ODE. We show that our 25-dimensional DManD model qualitatively captures the turbulent self-sustaining process and has good short-time predictive capabilities, matching the kinetic energy temporal autocorrelation for 30 time units. Furthermore, our DManD model excellently captures long-time statistics such as Reynolds stress. In order for this DManD model to be viable for RL, we added an additional reward network to predict the system drag, D, given the low-dimensional manifold state, h.

We then obtained a control strategy from our DManD model using deep RL, which successfully transferred to and controlled the original DNS. The DManD-RL trained 440 times faster than a direct application of deep RL to the DNS. We additionally emphasize here that the data generation and DManD model training in this framework is a fixed one-time cost that is greatly exceeded by the cost of conventional DNS-based RL training. For reference, we found the DNS-based RL training required over a month of training to accomplish 1,000 training episodes, while all steps of the DManD-RL framework (data generation, dynamics model training, 10,000 episodes of RL training) were accomplished within two days.

We find that our DManD-RL agent can consistently drive unseen turbulent initial conditions in the original DNS to the laminar state, despite never having any direct observations or interactions with the DNS. We also demonstrate that there exists a mapping between wall observables and the manifold state, which allowed us to apply the DManD-RL agent with

equal effectiveness using only wall shear rate observations.

When investigating the mechanistic nature of the learned control strategy, we observed multiple control strategies executed by the DManD-RL agent. One novel strategy the agent appears to employ consists of manipulating the low-speed streak to a preferred location, causing the breakdown of the streak, and in the wake of the break-down forming two low-speed streaks in its place. These two low-speed streaks are unsustainable within the domain, breaking the SSP and resulting in laminarization.

When comparing the ensemble behavior of our control agent to opposition control, we find that the controller behaves anti-oppositionally at the commonly-used detection plane location of $y^+ = 10$. At a detection plane of $y^+ = 33$, we find that the ensemble behavior is opposition-like, however, the broadness of the control action distribution, as well as the observation of the two-streak structure, leads us to conclude that the learned controller behavior is much more complex and diverse than a simple opposition feedback rule. We also compare our DManD-RL agent's control performance to that of opposition control and we find that our control agent out-performs opposition control by a notable margin (16% vs. 42% probability of remaining turbulent after 900 time units of control) despite our control set-up and agent possessing much greater restrictions on it spatial and temporal control authority (fixed time intervals of control and localized jets on only the lower wall) compared to opposition control.

Chapter 8

Conclusions

8.1 General summary

In this thesis, we outlined an approach to parameterizing and evolving chaotic dynamics that lie on manifolds. We used this method for predicting and controlling the dynamics of the Kuramoto-Sivashinksy equation and turbulent plane Couette flow. The first half of this thesis involved developing this method and applying it to the KSE. In Ch. 2 we used a hybrid autoencoder that built upon PCA to approximate an 8-dimensional coordinate system for the chaotic dynamics of the $L = 22$ KSE. Then, we showed how to construct an ODE to evolve this system forward in time in Ch. 3. This approach used sparsely spaced training data and allowed us to evolve trajectories to arbitrary points in time. We modified this low-dimensional model to account for control actuation in Ch. 4. Using RL we found a control strategy for the model that, when deployed on the true system, pushes states to a low dissipation unstable fixed point.

The second half of the thesis involved extending these methods to the high-dimensional case of turbulent Couette flow. In Ch. 5, we discovered a method for making the neural ODEs used for time evolution stable – a challenge when evolving high-dimensional systems with neural ODEs. With this modeling improvement, we applied our method to turbulent Couettte flow in Ch. 6. With fewer than 20 degrees of freedom, we built accurate models

of Couette flow and used this to find unstable relative periodic orbits. Finally, we added control to this system via two slot jets applied to one wall in Ch. 7. We trained RL agents on a 25-dimensional model of this controlled Couette flow. When we implemented this control policy on the DNS the agent performed a complex control strategy that rapidly laminarized the flow at a higher rate than opposition control with far less control authority. The results presented in this thesis lay the groundwork for the application of extremely low-dimensional data-driven models for forecasting and control of chaotic dynamical systems.

In the following sections, we outline future directions for this research first in modeling and then in model-based control. The main challenges left to be addressed for modeling include: handling spatially extended systems, accounting for discrete symmetries, and incorporating the equations of motion. In regards to model-based control, some challenges we discuss include controlling higher Reynolds number turbulence, using different control strategies, and controlling with different types of actuation.

8.2 Future directions for modeling

8.2.1 Distributed models

Up to this point all of the DManD models we considered involved finding a low-dimensional representation of the full state u over the entire domain. In the case of the KSE, in Ch. 2 and Ch. 3, this resulted in a manifold parameterized by $d_{\mathcal{M}} = 8, 18, 28$ dimensions for domain sizes of $L = 22, 44, 66$. As the domain became larger the dimension estimate was more challenging and the model performance started decreasing. To overcome this issue for large domains perhaps we can consider "chunks" of the domain as separate dynamical systems that are equivalent and communicate with one another. This idea was motivated by related approaches that used reservoir computing methods locally [2, 162]. In Fig. 8.1 we show an example snapshot of the KSE for $L = 220$ with dotted lines indicating different chunks. We refer to this method as a distributed model because each chunk share information with

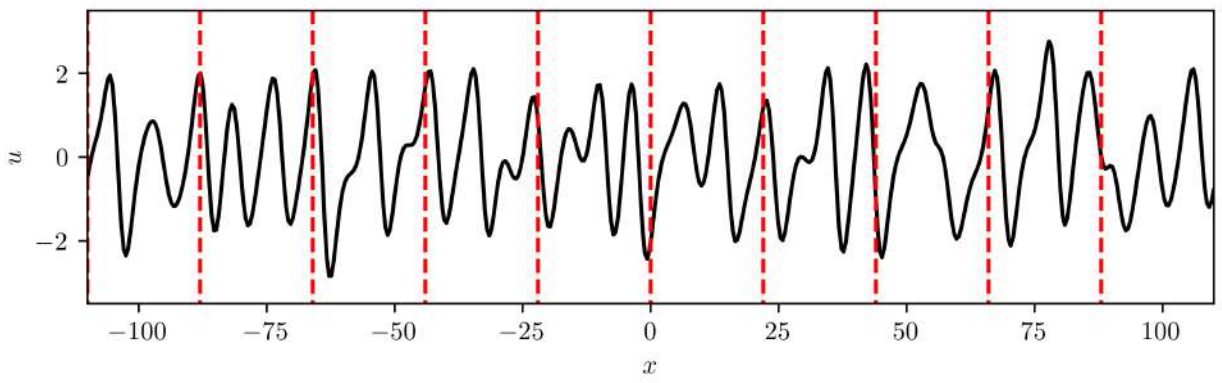


Figure 8.1: A snapshot of the KSE for domain size $L = 220$. The red dotted lines separate the domain for each of the 10 chunks.

neighboring chunks.

First, we define K "chunks" of data $u^{(k)} = u(x_k)$ for $k = 0, ..., K-1$ where $x_k \in [kL/K, (k+1)L/K]$. This means if $u \in \mathbb{R}^d$ then $u^{(k)} \in \mathbb{R}^{d/K}$. Now, instead of discovering a manifold coordinate representation for u we can find a low-dimensional representation for each chunk

$$h^{(k)} = \chi(u^{(k)}), \tag{8.1}$$

where $h^{(k)} \in \mathbb{R}^{d_h}$, and a mapping back to the chunk

$$u^{(k)} = \check{\chi}(h^{(k)}). \tag{8.2}$$

Then, the time evolution of each chunk depends on the other chunks according to

$$\frac{dh^{(k)}}{dt} = g(h^{(0)}, ..., h^{(K-1)}). \tag{8.3}$$

However, if we assume only neighboring chunks affect each other we can rewrite this as

$$\frac{dh^{(k)}}{dt} = g(h^{(k-1)}, h^{(k)}, h^{(k+1)}).\text{,} \tag{8.4}$$

where $h^{(-1)} = h^{(K)}$ and $h^{(K)} = h^{(0)}$ to account for periodic boundary conditions. Using these equations, we can simply approximate the functions χ, $\check{\chi}$, and g, and run the evolution of

each of the $h^{(k)}$ in parallel.

For a system that is spatially homogeneous, like the KSE, it seems reasonable for the chunks to share the same function, however, it is simple to extend this idea to different functions $\chi^{(k)}$, $\check{\chi}^{(k)}$, $g^{(k)}$ for each chunk. Also, by treating each chunk as the same function we effectively multiply the training data by K. Furthermore, to avoid sharp changes at the boundaries of the domains, we can overlap the chunks by some amount o by expanding the region of each chunk (i.e. $x_k \in [kL/K - o, (k+1)L/K + o]$). This simply increases the size of $u^{(k)}$ and requires that the overlapping areas are appropriately averaged to convert back to u.

To investigate the viability of this method, we test this approach on the KSE at $L = 220$ on a grid of dimension $d = 500$. For this system, we separate the data into $K = 10$ chunks with $3.2 \cdot 10^4$ snapshots of training data separated every 0.25 time units and used an overlap of 1 grid point. Then we train a single standard autoencoder shared among all of the chunks to reduce the dimension to $d_h = 12$. It is unclear how the boundary conditions and overlap affect the required number of dimensions, so we erred on the side of caution with this selection of dimension. This autoencoder outputs $\tilde{u}^{(k)} = \check{\chi}(\chi(u^{(k)}; \theta_E); \theta_D)$, the parameters of which we trained to minimize the loss

$$L = \frac{1}{dNK} \sum_{i=1}^{N} \sum_{k=1}^{K} ||u(t_i)^{(k)} - \tilde{u}(t_i)^{(k)}||_2^2. \tag{8.5}$$

The details of the autoencoder architecture are given in Table 8.1.

After training the autoencoder, we next train stabilized neural ODEs of the form

$$\frac{dh^{(k)}}{dt} = g_{\text{NN}}(h^{(k-1)}, h^{(k)}, h^{(k+1)}; \theta_g) - \alpha, \tag{8.6}$$

with $\alpha = 0.1$. We integrate this equation forward in time for all chunks at once resulting in

Table 8.1: Architectures of NNs used in the distributed DManD model. "Shape" indicates the dimension of each layer, "Activation" the corresponding activation functions, and "sig" is the sigmoid activation."Learning Rate" gives the learning rate for the Adam optimizer [113]. When multiple learning rates are noted, the value was changed from one value to the next at even intervals during training.

Function	Shape	Activation	Learning Rate
χ	$52/500/d_h$	sig/lin	$[10^{-3}, 10^{-4}]$
$\check{\chi}$	$d_h/500/52$	sig/lin	$[10^{-3}, 10^{-4}]$
g_{NN}	$3d_h/200/200/200/d_h$	sig/sig/sig/lin	$[10^{-2}, 10^{-3}, 10^{-4}]$

the prediction

$$\tilde{h}^{(k)}(t_i + \tau) = h^{(k)}(t_i) + \int_{t_i}^{t_i+\tau} g_{\mathrm{NN}}(h^{(k-1)}, h^{(k)}, h^{(k+1)}; \theta_g) - \alpha dt, \tag{8.7}$$

for all 10 chunks. Then we train the stabilized neural ODE to minimize

$$J = \frac{1}{dNK} \sum_{i=1}^{N} \sum_{k=1}^{K} ||h(t_i + \tau)^{(k)} - \tilde{h}(t_i + \tau)^{(k)}||_2^2. \tag{8.8}$$

We present the architecture of g_{NN} in Table 8.1.

In Fig. 8.2 we show the predictive capabilities of this distributed DManD model. The trajectories match for $\sim$ 30 time units before diverging. After diverging, the qualitative dynamics of the model continue to agree with the behavior of the underlying KSE. Figure 8.3 shows the joint PDF of u_x vs u_{xx} for the true system and the distributed DManD model. Here we see that the distributed model does an excellent job of capturing the long-time dynamics.

These results are extremely promising for the extension of DManD models to much larger systems. By modeling chunks of a large, distributed system, instead of the full state, we greatly ease training by learning a parallel set of low-dimensional models and by effectively increasing the dataset size by the number of chunks. We are currently extending this work to Kolmogorov flow systems. Then, we intend to model the Couette flow simulations with a larger domain size in the x and z directions. Turbulent flows are known to be multiscaled,

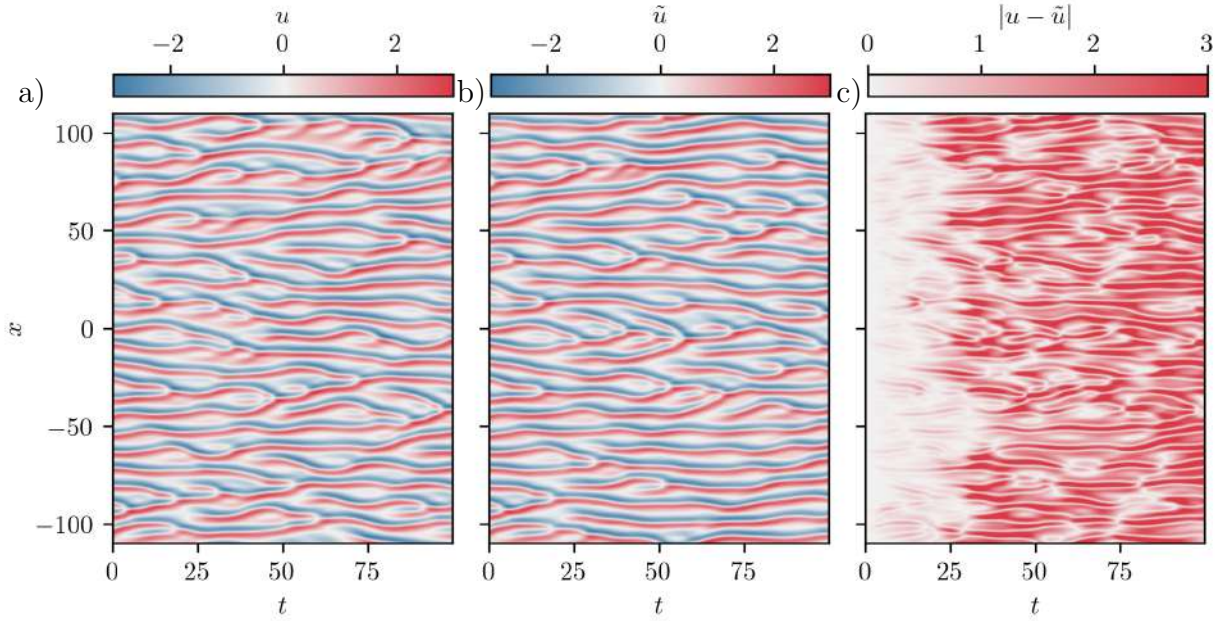


Figure 8.2: (a) and (b) are trajectories of the KSE at $L = 220$ for the true data and the distributed DManD model. (c) is the error between these trajectories.

so we may need to add a hierarchical structure to this distributed DManD approach. This could be implemented by using overlapping chunks of multiple lengthscales and reworking the way these chunks communicate through the ODE.

8.2.2 Symmetry charting[1]

Throughout many of the chapters, we accounted for the continuous symmetries of the underlying dynamical systems by decoupling the pattern and phase dynamics using the first Fourier mode method-of-slices [19]. As long as the magnitude of the first Fourier mode never went to 0 this method results in a smooth evolution of both the pattern and phase, with the phase only depending upon the pattern. Unfortunately, accounting for discrete symmetries is much harder because the indicator of a discrete symmetry evolves discontinuously through time.

To illustrate this point, and investigate how to account for these discrete symmetries, we

[1]The work in this section was performed in collaboration with Carlos E. Pérez De Jesús.

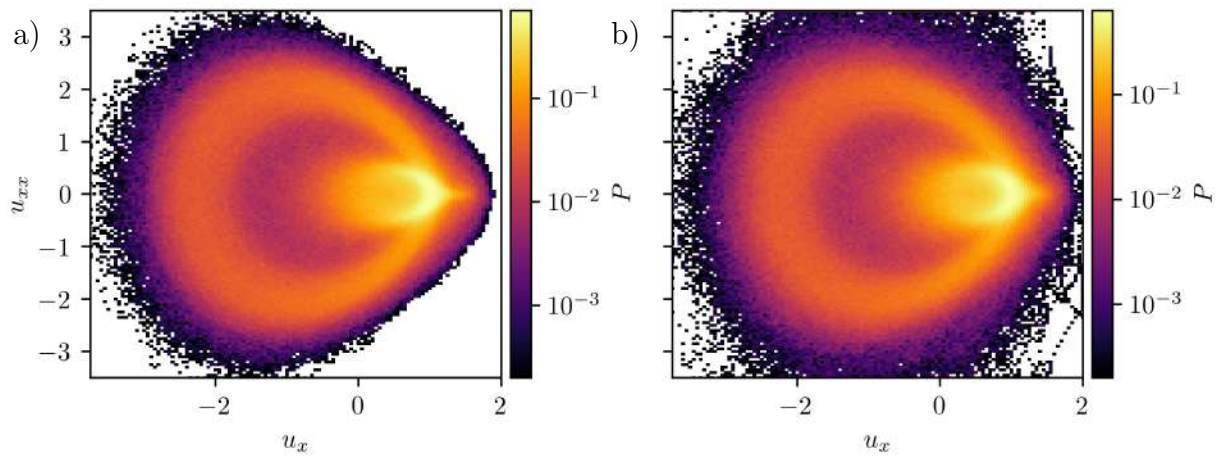


Figure 8.3: Joint PDFs of u_x vs u_{xx} for (a) the true data and (b) the distributed DManD model.

consider trajectories of the Kolmogorov flow system. We chose to use this system because it is more complicated and has more discrete symmetries than the KSE. Kolmogorov flow is a 2D flow with a sinusoidal forcing. This forcing is input into the nondimensional Navier-Stokes equation resulting in

$$\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} = -\nabla p + \mathrm{Re}^{-1} \nabla^2 \boldsymbol{u} + \sin(ny)\hat{\boldsymbol{x}}, \quad \nabla \cdot \boldsymbol{u} = 0. \tag{8.9}$$

For Kolmogorov flow there are two velocity components $\mathbf{u} = [u, v]$, on a periodic domain in $x \in [0, 2\pi/\alpha]$ and $y \in [0, 2\pi]$, with integer wavenumber forcing n, and $\hat{\mathbf{x}}$ is the unit vector in the x direction. In this case, the Reynolds number is $\mathrm{Re} = \sqrt{X} L_y^{3/2}/(\nu(2\pi)^{3/2})$, where X is the dimensional forcing amplitude, L_y is the domain size, and ν is the kinematic viscosity. In Pérez De Jesús and Graham [163], low-dimensional models, like the ones studied in this thesis, were generated for Kolmogorov flow with an aspect ratio $\alpha = 1$ and with a forcing $n = 2$ at $\mathrm{Re} = 14.4$, which is the smallest Reynolds number that exhibits chaotic dynamics. In this section, we run simulations in this same parameter regime as Pérez De Jesús and Graham [163] using the method described in Chandler and Kerswell [23].

Discrete and continuous symmetries appear when the equations of motion are invariant to certain operations. The equations and boundary conditions describing Kolmogorov flow

are invariant to three symmetry operations – a shift(in y)-reflect(in x), a rotation through π, and a continuous translation (in x) [23]:

$$\mathscr{S} : [u, v, \omega](x, y) \to [-u, v, -\omega] \left(-x, y + \frac{\pi}{n}\right) \tag{8.10}$$

$$\mathscr{R} : [u, v, \omega](x, y) \to [-u, -v, \omega](-x, -y) \tag{8.11}$$

$$\mathscr{T}_l : [u, v, \omega](x, y) \to [u, v, \omega](x + l, y) \quad \text{for } 0 \leqslant l < \frac{2\pi}{\alpha}. \tag{8.12}$$

Based on these equations, we can identify indicator functions for the discrete symmetries using the vorticity $\omega = \nabla \times \mathbf{u}$, and we can define a phase for the continuous symmetry using the method-of slices. In order to find these indicators first we consider the effect of these symmetry operations in Fourier space. After Fourier transforming the above equations and simplifying these are the actions of the symmetry operations in Fourier space:

$$\mathscr{S} : \hat{\omega}(k_x, k_y) \to -\hat{\omega}(-k_x, k_y) e^{ik_y \pi/n} \tag{8.13}$$

$$\mathscr{R} : \hat{\omega}(k_x, k_y) \to \bar{\hat{\omega}}(k_x, k_y) \tag{8.14}$$

$$\mathscr{T}_l : \hat{\omega}(k_x, k_y) \to \hat{\omega}(k_x, k_y) e^{-ik_x l}. \tag{8.15}$$

As a reminder $\bar{\cdot}$ denotes the complex conjugate.

Using these operations we classify a state as existing in a given symmetry (i.e. select an indicator) based on the sign of different Fourier coefficients. We still use the first-Fourier mode method of slices for removing the continuous symmetry. This means every snapshot is translated such that the imaginary part of $\hat{\omega}_I(1, 0) = 0$. From Eq. 8.15 we see that shift operations modify all Fourier coefficients with $k_x \neq 0$. Thus, as long as we use $k_x = 0$ modes to identify our discrete symmetries, the discrete indicator is independent of the phase.

First, for the shift-reflect operation, we consider the $(k_x = 0, k_y = 1)$ mode. In Eq. 8.13 the shift-reflect operation modifies this Fourier coefficient in this manner: $\hat{\omega}_R(0, 1) +$

$i\hat{\omega}_I(0,1) \rightarrow \hat{\omega}_I(0,1) - i\hat{\omega}_R(0,1)$. This shows that the shift-reflect operation maps between different quadrants of the $\hat{\omega}_R(0,1)$ and $\hat{\omega}_I(0,1)$ plane. Next, we consider the rotation operation on the $(k_x = 0, k_y = 2)$ mode. In Eq. 8.14 we see this operation simply flips the sign of the imaginary component $\hat{\omega}_I(0,2) \rightarrow -\hat{\omega}_I(0,2)$. The combination of these two operations can map data from any octet in the $\hat{\omega}_R(0,1)$, $\hat{\omega}_I(0,1)$, $\hat{\omega}_I(0,2)$ space to any other octets. As such, we use the sign of these three values to determine an indicator I between $1-8$. Then, we apply the appropriate discrete operations to map the state to the positive octet $(\hat{\omega}_R(0,1) > 0, \hat{\omega}_I(0,1) > 0, \hat{\omega}_I(0,2) > 0)$ reducing discrete symmetries. Following the reduction of discrete symmetries we then phase-align the data, mapping it to the "fundamental" space $(\tilde{w})$ [17].

In Fig. 8.4a we show a projection of the Kolmogorov data onto $\hat{\omega}_R(0,1)$, $\hat{\omega}_I(0,1)$, $\hat{\omega}_I(0,2)$ in the full space and in Fig. 8.4b we show this projection for the fundamental space. The fundamental space is much denser than the full space, which helps improve the modeling performance with fewer data points. Now that we have our data in the fundamental space we can train a single DManD model in the fundamental space and then track the discrete indicator and continuous phase through time. The challenge with this approach, as mentioned, is that the indicator is discontinuous in time because leaving the fundamental domain on one side requires entering from the other. Fortunately, Floryan and Graham [57] recently developed a method that solved this problem of tracking a discontinuous indicator through time. In that work, the indicator tracked "charts" on a manifold. Due to this result, we refer to the process in this section as symmetry charting.

Although we do not discuss the finer details of the process in Floryan and Graham [57], it consists of defining points interior and exterior to the charts (in our case the fundamental domain), where interior points are in the chart and exterior points are the data expanded slightly beyond the boundary of the chart. We train the DManD model on this expanded data set. Then during testing, while evolving the manifold coordinates h, we check if the point is in the interior or the exterior. If the point lands in the exterior, we decode the

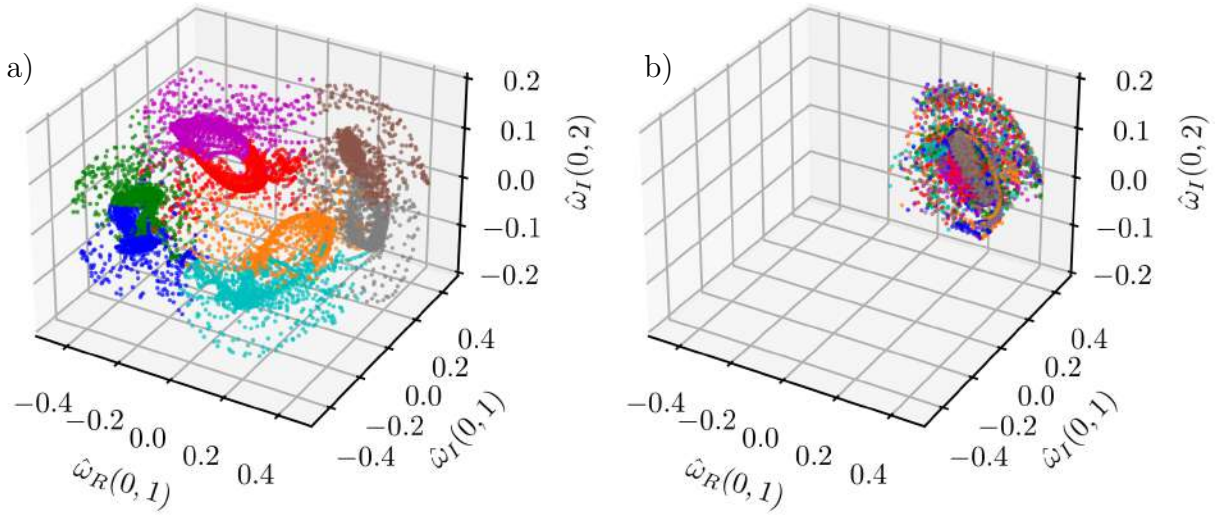


Figure 8.4: (a) shows a trajectory of Kolmogorov flow for $5,000$ time units projected onto $\hat{\omega}_R(0,1)$, $\hat{\omega}_I(0,1)$, $\hat{\omega}_I(0,2)$. The color corresponds to the indicator that identifies the symmetry group. (b) shows this data mapped to the fundamental space.

manifold coordinates h to get the vorticity on the exterior of the fundamental w_E, then we apply the appropriate operations to map it back to the fundamental space $\tilde{w}$, encode to get h, and update the indicator I. We emphasize that we train the autoencoder and the time evolution method with points just outside the fundamental space so we can accurately model one step into the exterior, which we immediately map back into the fundamental. We refer the reader to Floryan and Graham [57] for a discussion on how to identify when a trajectory enters the exterior.

By performing the evolution in this way we guarantee equivariance of trajectories under symmetry operations and improve the training efficiency of the DManD algorithm. To investigate the effectiveness of this method, we gathered $80,000$ snapshots of Kolmogorov flow data separated every 0.5 time units and used this to train hybrid autoencoders with $d_h = 15$ on 300 PCA coefficients. The architecture of this network is in Table 8.2.

In Fig. 8.5 we show the autoencoder performance as a function of varying the amount of data evenly sampled from the $80,000$ snapshots. The model that accounts for all of the symmetries achieves an MSE multiple orders of magnitude better than without symmetries

Table 8.2: Architectures of NNs used in the symmetry charting DManD model. Labels are the same as Table 8.1.

Function	Shape	Activation	Learning Rate
χ	$300/1000/500/d_h$	sig/sig/sig	$[10^{-4}, 10^{-5}]$
$\check{\chi}$	$d_h/500/1000/300$	sig/sig/lin	$[10^{-4}, 10^{-5}]$
G	$d_h/500/500/500/d_h$	sig/sig/sig/lin	$[10^{-4}, 10^{-5}]$

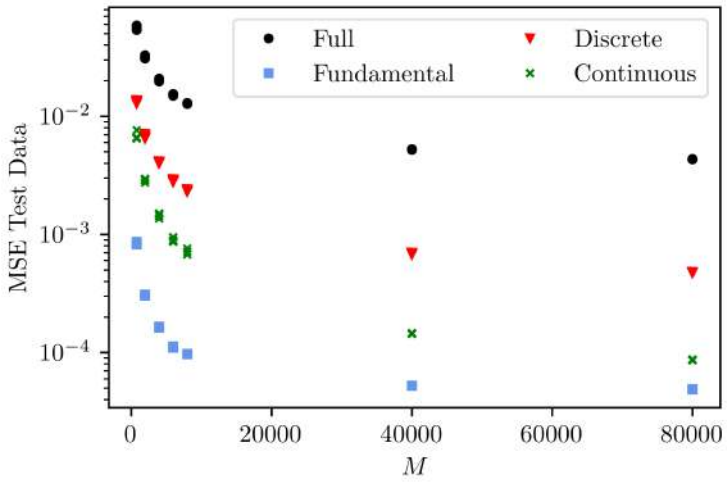


Figure 8.5: Autoencoder MSE vs training data size without accounting for symmetry (Full), account for discrete symmetry (Discrete), accounting for continuous (Continuous), and accounting for all symmetries (Fundamental).

with a fraction of the data. Notably, the autoencoders trained in the fundamental space using 800 snapshots of data outperform those trained in the full space using $80,000$ snapshots of data.

After training the autoencoder, we selected the one using $40,000$ snapshots of data and we trained a discrete-time map NN $h(t+\tau) = G(h(t); \theta_G)$, with the architecture in table 8.2. In Fig. 8.6 we show two trajectories of the enstrophy (integral of the vorticity squared) evolved from an initial condition that has been mapped to two starting discrete symmetry indicators. The two trajectories from the symmetry charting DManD model match exactly confirming we have enforced equivariance through this structure. Furthermore, we see good short-time tracking and the model exhibits similar bursting dynamics to the true Kolmogorov flow. We are currently working on quantifying the short- and long-time tracking for these

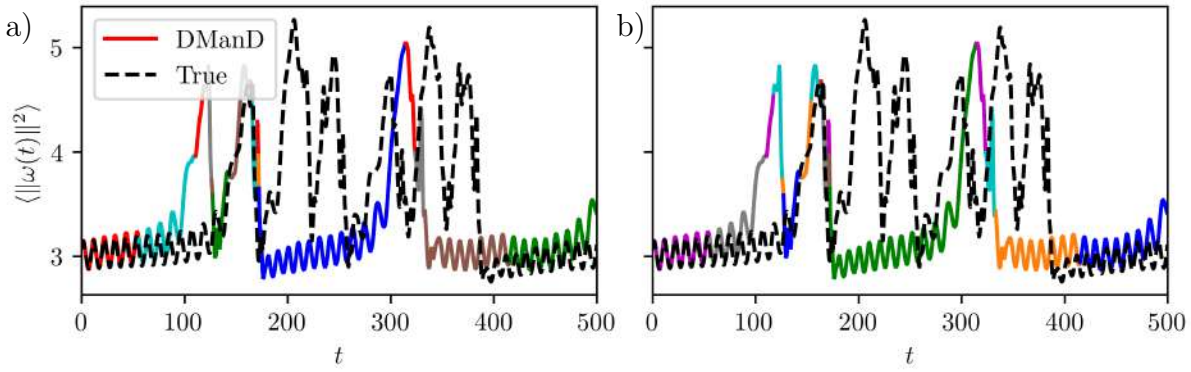


Figure 8.6: Trajectories of enstrophy for the true system and the symmetry aware DManD model starting with different discrete symmetries in (a) and (b). The color indicates the current discrete symmetry indicator.

models and we are varying the dimension d_h.

One additional subtlety this process introduces is that it modifies the phase evolution. Now the phase evolution is a function of both the manifold coordinate representation of the state h and the indicator I. For Kolmogorov flow, the resulting phase equation (in continuous time) is

$$\frac{d\phi}{dt} = g_I(I) g_\phi(h; \theta_\phi). \tag{8.16}$$

The function $g_I(I)$ equals 1 if an even number of discrete operations mapped the state back to the fundamental, or -1 if an odd number of discrete operations mapped the state back to the fundamental. This will be an important consideration when we incorporate the phase back into the model predictions.

8.2.3 Incorporating the underlying equations

All of the methods discussed so far have been strictly data-driven, however in many situations, we know the equations of motion. In fact, as shown in Eq. 1.7, if we know the method by which we are encoding then we also know the equations of motion for h. In Eq. 1.7 we

wrote the equation in terms of h, but we can also write it as

$$\frac{dh}{dt} = \frac{\partial \chi(u)}{\partial u}\bigg|_u f(u). \tag{8.17}$$

Furthermore, if we break down f into a linear and a nonlinear term then

$$\frac{dh}{dt} = c\frac{\partial \chi(u)}{\partial u}\bigg|_u Au + \frac{\partial \chi(u)}{\partial u}\bigg|_u \mathcal{N}(u), \tag{8.18}$$

where c is a scalar parameter (e.g. Reynolds number).

We could use the equations of motion directly to compute dh/dt or the two terms in Eq. 8.18 from u and then train NNs to approximate these functions. This approach is powerful because it applies to any encoding and provides a method of incorporating parameter dependency. We only need a set of states u, which could be sampled from a wider area of state space than can be achieved from only sampling states on the manifold. Furthermore, when the encoding comes from a NN the computation of $\partial \chi(u)/\partial u$ is easy and fast.

8.3 Future directions for control

8.3.1 Model predictive control

In this section, we discuss some additional results and future directions for the control problem presented in Ch. 7. Specifically, we discuss using model predictive control (MPC), extending the results to higher Reynolds numbers, and additional modifications to the flow setup worth exploring.

Although we chose to use RL in Ch. 7, any model-based control strategy will benefit from a faster model of the system. To investigate this we applied finite-time horizon nonlinear MPC in the fashion of Bewley et al. [10] to the Couette control problem at Re = 400. In our implementation of MPC, the objective was to find the set of actions a_t that minimize

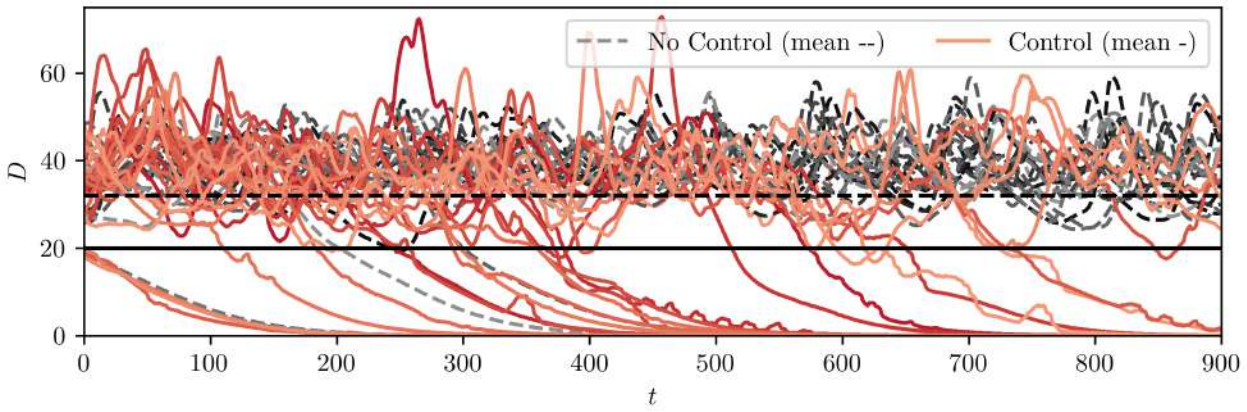


Figure 8.7: Trajectories of drag for 25 initial conditions without control and with control applied to a DNS at Re = 400. The control policy comes from applying MPC to the DManD model.

the sum of the negative reward $r_t = R(h_t, a_t; \theta_R)$

$$\begin{aligned} \min_{[a_t,\ldots,a_{t+T}]} \quad & J = \sum_{\tau=t}^{t+T} -R(h_\tau, a_\tau; \theta_R) \\ \text{s.t.} \quad & \frac{dh}{dt} = g(h, a; \theta_g) + Ah \end{aligned} \tag{8.19}$$

over some horizon of T actions. In this minimization problem, the evolution of h is dictated by the DManD model in Ch. 7. At each t, we perform this optimization over the T horizon, then we apply the first action a_t and discard the rest. After the system has evolved forward 5 time units (our actuation frequency from Ch. 7), we then repeat the optimization process to find a new action.

To perform this optimization procedure, at every step we used the automatic differentiation tools we used for training the neural ODE to compute the gradient $dJ/d[a_t, ..., a_{t+T}]$. We emphasize that this gradient is computed strictly from the $d_h = 25$ dimensional DManD model. Then we used the Adam optimizer to perform 100 gradient descent steps, with a learning rate that dropped from 10^{-2} to 10^{-3} midway through optimizing. In Fig. 8.7 we show trajectories of the DNS controlled by actuations selected from applying MPC to the DManD model. We chose an optimization horizon of 100 time units, which corresponds to $T = 20$ based on our actuation frequency. The MPC procedure successfully laminarizes a

majority of the trajectories and achieves similar drag reduction to the DManD-RL method presented in Ch. 7.

These MPC results can likely be improved by selecting longer time horizons and by exploring different optimization methods at each step. The MPC trials took ~ 6 times as long as the RL testing but there is no upfront training cost for MPC like there is for RL. The real-time implementation of MPC could also be sped up by first performing MPC to identify state-action pairs and then using a NN to map between the two. In the future, it would be interesting to rigorously investigate the application of different control schemes, like MPC, using the DManD models.

8.3.2 DManD-RL Re $= 500$

Next, we investigate the application of the DManD-RL control to Couette flow at a higher Reynolds number of Re $= 500$. To perform these trials we generated randomly actuated data as we did in Ch. 7 with two key differences: Re $= 500$ and $V_{\max} = 0.1$. With this dataset, we used the same POD method and autoencoder structure described in Ch. 7 to reduce the size of this dataset down to $d_h = 30$. In Fig. 8.8 we show the Reynolds stress terms for the DNS, the DManD model, and the autoencoder (as described in Ch. 7). The DManD model matches all of the autoencoder curves indicating that the dynamics are accurately captured. However, the autoencoder does a worse job at Re $= 500$ than at Re $= 400$ of reconstructing the Reynolds stress terms, particularly the wall-normal and spanwise terms. A higher dimension than $d_h = 30$ may be required to better capture these terms.

After training the DManD model, we again used it to train an RL agent. In Fig. 8.9 we show the implementation of this control policy in the DNS. The agent successfully laminarizes two of the trajectories over this time window, whereas no trajectories of the DNS laminarize. Overall, the RL agent achieves a drag reduction of $DR = 12\%$ over this time horizon. We note that during the DManD training, in this case, we iterated on the model twice to improve the results. This entailed training the model on the randomly actuated data, training an

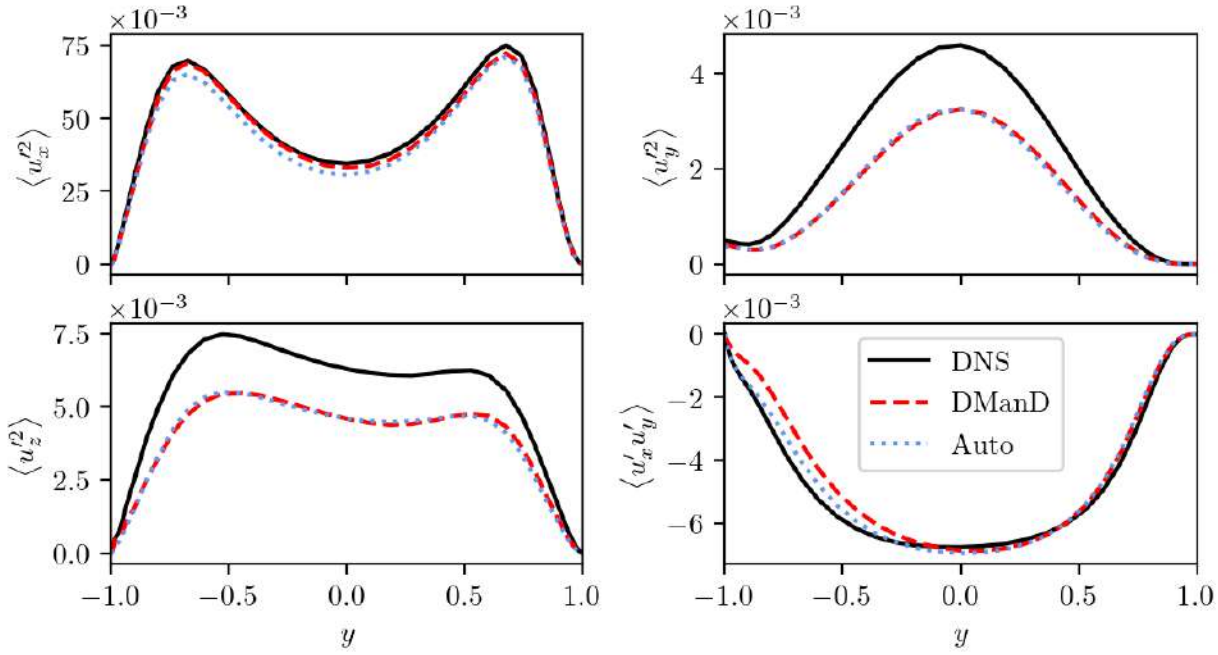


Figure 8.8: Four components of the Reynolds stress for the DNS, the DManD model, and the autoencoder at Re = 500.

RL agent on that model, deploying the model to generate new data, and augmenting the old dataset with this new data. With this new data, we repeated the training procedure to update the model. The RL agent in Fig. 8.9 comes from the third iteration of training (i.e. we augmented the training data with 2 DManD-RL agents).

An important challenge that was introduced by increasing the Reynolds number is the issue of controllability. Increasing the Reynolds number without changing the domain size results in a system that no longer isolates the SSP. At this Reynolds number and domain size, there are no longer just two sets of quasi-streamwise vortices lifting low-speed fluid off the wall, so actuating with two slot jets is a significant restriction on the control authority of this setup. That is why it is promising that despite these restrictions the agent still learns a policy that significantly reduces drag and can laminarize the flow.

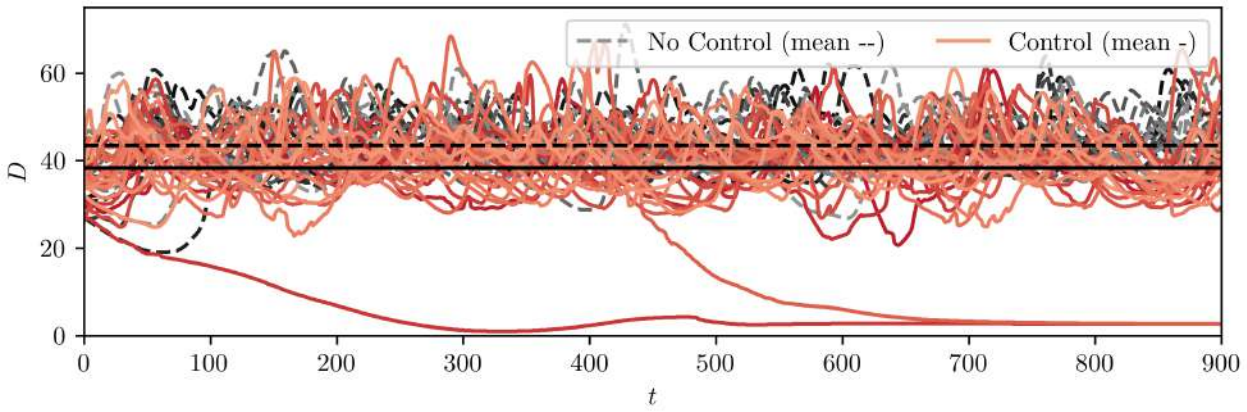


Figure 8.9: Trajectories of drag for 25 initial conditions without control and with control from the DManD-RL agent on a DNS at $\mathrm{Re} = 500$.

8.3.3 Application to different flow systems

For these higher Reynolds numbers, an important future direction will be considering different forms of actuation. In Linot et al. [133] we made our DNS publicly available with the ability to input any form of periodic wall-normal actuation. With small modifications, this could be extended for arbitrary streamwise and spanwise velocity boundary conditions and could be applied to pressure-driven systems. A particularly interesting case could be the application of a grid of circular jets applied to pressure-driven plane Poiseuille flow. This type of setup would be more representative of most wall-bounded flows, and an array of jets could provide the system with enough control authority to react to variable Reynolds numbers.

In this final chapter, we have outlined a few of the exciting directions for this work in terms of modeling and control. For improving the modeling, we presented preliminary results on how to apply this method to spatially extended systems, how to account for the underlying symmetries, and how to incorporate the equations of motion. Then, for control, we showed that the DManD models can be effectively used with MPC as the controller, we showed that the DManD-RL models can control higher Reynolds number cases, and we discussed the next flow problem we envision for the application of DManD-RL. We hope that this work has proven that we can both model and control high-dimensional chaotic

systems with low-dimensional models of these systems. And, in doing so, has provided a useful framework for viewing chaotic dynamical systems; one in which we think of describing these systems in terms of the manifold on which they lie instead of in terms of their ostensible high dimensionality.

References

[1] R. Agarwal, D. Schuurmans, and M. Norouzi. An optimistic perspective on offline reinforcement learning. In H. D. III and A. Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 104–114. PMLR, 13–18 Jul 2020.

[2] T. Arcomano, I. Szunyogh, A. Wikner, J. Pathak, B. R. Hunt, and E. Ott. A hybrid approach to atmospheric modeling that combines machine learning with a physics-based numerical model. *Journal of Advances in Modeling Earth Systems*, 14(3), 2022.

[3] R. E. A. Arndt, D. F. Long, and M. N. Glauser. The proper orthogonal decomposition of pressure fluctuations surrounding a turbulent jet. *Journal of Fluid Mechanics*, 340: 1–33, 1997.

[4] M. Asch, M. Bocquet, and M. Nodet. *Data Assimilation: Methods, Algorithms, and Applications*. SIAM, 2016.

[5] P. J. Baddoo, B. Herrmann, B. J. McKeon, and S. L. Brunton. Kernel learning for robust dynamic mode decomposition: Linear and nonlinear disambiguation optimization (LANDO). *arXiv preprint arXiv:2106.01510*, 2021.

[6] K. S. Ball, L. Sirovich, and L. R. Keefe. Dynamical eigenfunction decomposition of turbulent channel flow. *International Journal for Numerical Methods in Fluids*, 12(6): 585–604, 1991.

[7] E. Becht, L. McInnes, J. Healy, C.-A. Dutertre, I. W. H. Kwok, L. G. Ng, F. Ginhoux, and E. W. Newell. Dimensionality reduction for visualizing single-cell data using umap. *Nature Biotechnology*, 37(1):38–44, 2019.

[8] G. Beintema, A. Corbetta, L. Biferale, and F. Toschi. Controlling Rayleigh–Bénard convection via reinforcement learning. *Journal of Turbulence*, 21(9-10):585–605, 2020.

[9] Y. Bengio, J. F. Paiement, P. Vincent, O. Delalleau, N. Le Roux, and M. Ouimet. Out-of-sample extensions for LLE, Isomap, MDS, Eigenmaps, and spectral clustering. *Advances in Neural Information Processing Systems*, 2004.

[10] T. R. Bewley, P. Moin, and R. Temam. DNS-based predictive control of turbulence: an optimal benchmark for feedback algorithms. *Journal of Fluid Mechanics*, 447:179–225, 2001.

[11] C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Inc., USA, 1995.

[12] G. Borrelli, L. Guastoni, H. Eivazi, P. Schlatter, and R. Vinuesa. Predicting the temporal dynamics of turbulent channels through deep learning. *International Journal of Heat and Fluid Flow*, 96:109010, 2022.

[13] J. D. Boskovic, L. Chen, and R. K. Mehra. Adaptive Control Design for Nonaffine Models Arising in Flight Control. *Journal of Guidance, Control, and Dynamics*, 27 (2):209–217, 2004.

[14] C. D. Brummitt and J. C. Sprott. A search for the simplest chaotic partial differential equation. *Physics Letters, Section A: General, Atomic and Solid State Physics*, 373 (31):2717–2721, 2009.

[15] S. L. Brunton, J. L. Proctor, and J. N. Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 113(15):3932–3937, 2016.

[16] M. A. Bucci, O. Semeraro, A. Allauzen, G. Wisniewski, L. Cordier, and L. Mathelin. Control of chaotic systems by deep reinforcement learning. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 475(2231):20190351, 2019.

[17] N. B. Budanur and P. Cvitanović. Unstable Manifolds of Relative Periodic Orbits in the Symmetry-Reduced State Space of the Kuramoto-Sivashinsky System. *Journal of Statistical Physics*, 167(3-4):636–655, 2017.

[18] N. B. Budanur, D. Borrero-Echeverry, and P. Cvitanović. Periodic orbit analysis of a system with continuous symmetry-A tutorial. *Chaos*, 25(7), 2015.

[19] N. B. Budanur, P. Cvitanović, R. L. Davidchack, and E. Siminos. Reduction of SO(2) symmetry for spatially extended dynamical systems. *Physical Review Letters*, 114(8): 1–5, 2015.

[20] M. Budišić, R. Mohr, and I. Mezić. Applied koopmanism. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 22(4):047510, 2012.

[21] F. Cazáis and A. Lhéritier. Beyond two-sample-tests: Localizing data discrepancies in high-dimensional spaces. In *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 1–10, 2015.

[22] K. Champion, B. Lusch, J. N. Kutz, and S. L. Brunton. Data-driven discovery of coordinates and governing equations. *Proceedings of the National Academy of Sciences*, 116(45):22445–22451, 2019.

[23] G. J. Chandler and R. R. Kerswell. Invariant recurrent solutions embedded in a turbulent two-dimensional Kolmogorov flow. *Journal of Fluid Mechanics*, 722:554–595, 2013.

[24] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud. Neural ordinary differential equations. *arXiv preprint arXiv:1806.07366*, 2019.

[25] X. Cheng, Z. Qiao, X. Zhang, M. Quadrio, and Y. Zhou. Skin-friction reduction using periodic blowing through streamwise slits. *Journal of Fluid Mechanics*, 920:A50, 2021.

[26] K. Cho, B. van Merriënboer, D. Bahdanau, and Y. Bengio. On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111, Doha, Qatar, Oct. 2014. Association for Computational Linguistics.

[27] H. Choi and S. Choi. Robust kernel isomap. *Pattern Recognition*, 40(3):853–862, 2007.

[28] H. Choi, P. Moin, and J. Kim. Active turbulence control for drag reduction in wall-bounded flows. *Journal of Fluid Mechanics*, 262:75–110, 1994.

[29] H. Choi, W.-P. Jeon, and J. Kim. Control of flow over a bluff body. *Annual Review of Fluid Mechanics*, 40(1):113–139, 2008.

[30] K.-S. Choi, T. Jukes, and R. Whalley. Turbulent boundary-layer control with plasma actuators. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 369(1940):1443–1458, 2011.

[31] F. Chollet et al. Keras. `https://keras.io`, 2015.

[32] K. Chua, R. Calandra, R. McAllister, and S. Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[33] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NIPS 2014 Workshop on Deep Learning, December 2014*, 2014.

[34] Y. M. Chung and T. Talha. Effectiveness of active flow control for turbulent skin friction drag reduction. *Physics of Fluids*, 23(2):025102, 2011.

[35] R. R. Coifman, S. Lafon, A. B. Lee, M. Maggioni, B. Nadler, F. Warner, and S. W. Zucker. Geometric diffusions as a tool for harmonic analysis and structure definition of data: diffusion maps. *Proceedings of the National Academy of Sciences of the United States of America*, 102 21:7426–31, 2005.

[36] J. D. Cole. On a quasi-linear parabolic equation occurring in aerodynamics. *Quarterly of Applied Mathematics*, 9(3):225–236, 1951.

[37] P. Constantin, C. Foias, B. Nicolaenko, and R. Témam. Spectral barriers and inertial manifolds for dissipative partial differential equations. *Journal of Dynamics and Differential Equations*, 1(1):45–73, 1989.

[38] J. P. Cunningham and Z. Ghahramani. Linear dimensionality reduction: Survey, insights, and generalizations. *Journal of Machine Learning Research*, 16:2859–2900, 2015.

[39] P. Cvitanović, R. L. Davidchack, and E. Siminos. On the State Space Geometry of the Kuramoto-Sivashinsky Flow in a Periodic Domain. *SIAM Journal on Applied Dynamical Systems*, 9(1):1–33, 2010.

[40] P. Cvitanović, D. Borrero-Echeverry, K. M. Carroll, B. Robbins, and E. Siminos. Cartography of high-dimensional flows: A visual guide to sections and slices. *Chaos*, 22 (4), 2012.

[41] P. Cvitanović, R. Artuso, R. Mainieri, G. Tanner, and G. Vattay. *Chaos: Classical and Quantum*. Niels Bohr Inst., Copenhagen, 2016.

[42] D. de Ridder and R. P. Duin. Sammon's mapping using neural networks: A comparison. *Pattern Recognition Letters*, 18(11):1307–1316, 1997.

[43] X. Ding, H. Chaté, P. Cvitanović, E. Siminos, and K. A. Takeuchi. Estimating the Dimension of an Inertial Manifold from Unstable Periodic Orbits. *Physical Review Letters*, 117(2):1–5, 2016.

[44] C. R. Doering, J. D. Gibbon, D. D. Holm, and B. Nicolaenko. Low-dimensional behaviour in the complex Ginzburg-Landau equation. *Nonlinearity*, 1(2):279–309, 1988.

[45] J. Dormand and P. Prince. A family of embedded Runge-Kutta formulae. *Journal of Computational and Applied Mathematics*, 6(1):19–26, 1980.

[46] S. S. Du, S. M. Kakade, R. Wang, and L. F. Yang. Is a Good Representation Sufficient for Sample Efficient Reinforcement Learning? In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[47] G. Dulac-Arnold, N. Levine, D. J. Mankowitz, J. Li, C. Paduraru, S. Gowal, and T. Hester. Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. *Machine Learning*, 110(9):2419–2468, 2021.

[48] E. Dupont, A. Doucet, and Y. W. Teh. Augmented neural odes. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[49] J. P. Eckmann and D. Ruelle. Ergodic theory of chaos and strange attractors. *Reviews of Modern Physics*, 57(3):617–656, 1985.

[50] R. A. Edson, J. E. Bunder, T. W. Mattner, and A. J. Roberts. Lyapunov exponents of the Kuramoto-Sivashinsky PDE. *The ANZIAM Journal*, 61(3):270–285, 2019.

[51] H. Eivazi, H. Veisi, M. H. Naderi, and V. Esfahanian. Deep neural networks for nonlinear model order reduction of unsteady flows. *Physics of Fluids*, 32(10):105104, 2020.

[52] H. Eivazi, L. Guastoni, P. Schlatter, H. Azizpour, and R. Vinuesa. Recurrent neural networks and Koopman-based frameworks for temporal predictions in a low-order model of turbulence. *International Journal of Heat and Fluid Flow*, 90:108816, 2021.

[53] H. Eivazi, S. Le Clainche, S. Hoyas, and R. Vinuesa. Towards extraction of orthogonal and parsimonious non-linear modes from turbulent flows. *Expert Systems with Applications*, 202:117038, 2022.

[54] D. Fan, L. Yang, Z. Wang, M. S. Triantafyllou, and G. E. Karniadakis. Reinforcement learning for bluff body active flow control in experiments and simulations. *Proceedings of the National Academy of Sciences*, 117(42):26091–26098, 2020.

[55] V. Feinberg, A. Wan, I. Stoica, M. I. Jordan, J. E. Gonzalez, and S. Levine. Model-Based Value Estimation for Efficient Model-Free Reinforcement Learning. *arXiv*, abs/1803.00101, 2018.

[56] A. L. Ferguson, A. Z. Panagiotopoulos, I. G. Kevrekidis, and P. G. Debenedetti. Nonlinear dimensionality reduction in molecular simulation: The diffusion map approach. *Chemical Physics Letters*, 509(1-3):1–11, 2011.

[57] D. Floryan and M. D. Graham. Data-driven discovery of intrinsic dynamics. *Nature Machine Intelligence*, 4(12):1113–1120, 2022.

[58] C. Foias, M. S. Jolly, I. G. Kevrekidis, G. R. Sell, and E. S. Titi. On the computation of inertial manifolds. *Physics Letters A*, 131(7-8):433–436, 1988.

[59] C. Foias, O. Manley, and R. Temam. Modelling of the interaction of small and large eddies in two dimensional turbulent flows. *ESAIM: Mathematical Modelling and Numerical Analysis - Modélisation Mathématique et Analyse Numérique*, 22(1):93–118, 1988.

[60] C. Foias, B. Nicolaenko, G. R. Sell, and R. Temam. Inertial manifold for the Kuramoto-Sivashinsky equation and an estimate of their lowest dimension. *J. Math. Pure Appl.*, 67:197–226, 1988.

[61] A. J. Fox and M. D. Graham. Predicting extreme events in a data-driven model of turbulent shear flow using an atlas of charts. *arXiv preprint arXiv:2301.11818*, 2023.

[62] K. Fukami, T. Nakamura, and K. Fukagata. Convolutional neural network based hierarchical autoencoder for nonlinear mode decomposition of fluid field data. *Physics of Fluids*, 32(9):095110, 2020.

[63] B. García-Archilla, J. Novo, and E. S. Titi. Postprocessing the galerkin method: a novel approach to approximate inertial manifolds. *SIAM Journal on Numerical Analysis*, 35 (3):941–972, 1998.

[64] B. Geranmehr and S. R. Nekoo. Nonlinear suboptimal control of fully coupled non-affine six-DOF autonomous underwater vehicle using the state-dependent Riccati equation. *Ocean Engineering*, 96:248–257, 2015.

[65] B. Ghojogh, A. Ghodsi, F. Karray, and M. Crowley. Multidimensional scaling, sammon mapping, and isomap: Tutorial and survey. *arXiv preprint arXiv:2009.08136*.

[66] J. F. Gibson. *Dynamical systems models of wall-bounded, shear-flow turbulence*. PhD thesis, Cornell University, New York, Jan. 2002.

[67] J. F. Gibson. Channelflow: a spectral Navier-Stokes simulator in C++. *University of New Hampshire*, (July):1–41, 2012.

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

[69] J. F. Gibson, J. Halcrow, P. Cvitanović, and D. Viswanath. Unstable periodic orbits and heteroclinic connections in plane couette flow. *arXiv preprint*, 2008.

[70] J. F. Gibson, S. A. F. Reetz, A. Ferraro, T. Kreilos, H. Schrobsdorff, M. Farano, A. F. Yesil, S. S. Schütz, M. Culpo, and T. M. Schneider. Channelflow 2.0. 2021.

[71] A. Glezer and M. Amitay. Synthetic jets. *Annual Review of Fluid Mechanics*, 34(1): 503–529, 2002.

[72] F. J. Gonzalez and M. Balajewicz. Deep convolutional recurrent autoencoders for learning low-dimensional feature dynamics of fluid systems. *arXiv preprint arXiv:1808.01346*, 2018.

[73] R. Gonzalez-Garcia, R. Rico-Martinez, and I. G. Kevrekidis. Identification of distributed parameter systems: A neural net based approach. *Computers & Chemical EngineeringComputers & Chemical Engineering*, 22:S965–S968, 1998.

[74] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[75] M. D. Graham. Drag reduction and the dynamics of turbulence in simple and complex fluids. *Physics of Fluids*, 26(10):101301, 2014.

[76] M. D. Graham and D. Floryan. Exact Coherent States and the Nonlinear Dynamics of Wall-Bounded Turbulent Flows. *Annual Review of Fluid Mechanics*, 53(1):227–253, 2021.

[77] M. D. Graham, P. H. Steen, and E. S. Titi. Computational efficiency and approximate inertial manifolds for a Bénard convection system. *Journal of Nonlinear Science*, 3(1): 153–167, 1993.

[78] M. D. Graham, P. H. Steen, and E. S. Titi. Computational Efficiency and Approximate Inertial Manifolds for a Benard Convection System. *J Nonlinear Sci*, 3(2):153–167, 1993.

[79] L. Guastoni, J. Rabault, P. Schlatter, H. Azizpour, and R. Vinuesa. Deep reinforcement learning for turbulent drag reduction in channel flows. *arXiv preprint arXiv:2301.09889*, 2023.

[80] V. Guillemin and A. Pollack. *Differential Topology*. AMS Chelsea Publishing, Providence, Rhode Island, 2000.

[81] D. Ha and J. Schmidhuber. Recurrent World Models Facilitate Policy Evolution. In S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 2455–2467, 2018.

[82] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*.

[83] J. Hamilton, J. Kim, and F. Waleffe. Regeneration mechanisms of near-wall turbulence structures. *Journal of Fluid Mechanics*, 287:317–348, 1995.

[84] E. P. Hammond, T. R. Bewley, and P. Moin. Observed mechanisms for turbulence attenuation and enhancement in opposition-controlled wall-bounded flows. *Physics of Fluids*, 10(9):2421–2423, 1998.

[85] K. Hasegawa, K. Fukami, T. Murata, and K. Fukagata. Machine-learning-based reduced-order modeling for unsteady flows around bluff bodies of various shapes. *Theoretical and Computational Fluid Dynamics*, 34(4):367–383, 2020.

[86] K. Hasegawa, K. Fukami, T. Murata, and K. Fukagata. CNN-LSTM based reduced order modeling of two-dimensional unsteady flows around a circular cylinder at different reynolds numbers. *Fluid Dynamics Research*, 52(6):065501, Nov 2020.

[87] T. Heath. *A History of Greek Mathematics*. Number v. 2 in A History of Greek Mathematics. Clarendon Press, 1921.

[88] H. Hentschel and I. Procaccia. The infinite number of generalized dimensions of fractals and strange attractors. *Physica D: Nonlinear Phenomena*, 8(3):435–444, 1983.

[89] G. Hinton and S. Roweis. Stochastic neighbor embedding. *Advances in neural information processing systems*, 15:833–840, 2003.

[90] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[91] S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 11 1997.

[92] P. Holmes, J. L. Lumley, G. Berkooz, and C. W. Rowley. *Turbulence, Coherent Structures, Dynamical Systems and Symmetry*, volume 36. 1998.

[93] P. Holmes, J. L. Lumley, G. Berkooz, and C. W. Rowley. *Galerkin projection*, pages 106–129. Cambridge Monographs on Mechanics. Cambridge University Press, 2 edition, 2012.

[94] E. Hopf. A mathematical example displaying features of turbulence. *Communications on Pure and Applied Mathematics*, 1(4):303–322, 1948.

[95] E. Hopf. The partial differential equation $u_t + uu_x = \mu_{xx}$. *Communications on Pure and Applied Mathematics*, 3(3):201–230, 1950.

[96] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558, 1982.

[97] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991.

[98] A. C. Ian Goodfellow, Yoshua Bengio. *The Deep Learning Book*, volume 521. 2017.

[99] J. I. Ibrahim, Q. Yang, P. Doohan, and Y. Hwang. Phase-space dynamics of opposition control in wall-bounded turbulent flows. *Journal of Fluid Mechanics*, 861:29óÀẼ54, 2019.

[100] M. Inubushi, S.-i. Takehiro, and M. Yamada. Regeneration cycle and the covariant lyapunov vectors in a minimal wall turbulence. *Phys. Rev. E*, 92:023022, Aug 2015.

[101] R. Iten, T. Metger, H. Wilming, L. del Rio, and R. Renner. Discovering Physical Concepts with Neural Networks. *Physical Review Letters*, 124(1):10508, 2020.

[102] H. Jaeger. The "echo state" approach to analysing and training recurrent neural networks. GMD Report 148, GMD - German National Research Institute for Computer Science, 2001.

[103] F. Jauberteau, C. Rosier, and R. Temam. The nonlinear Galerkin method in computational fluid dynamics. *Applied Numerical Mathematics*, 6(5):361–370, 1990.

[104] J. Jiménez and A. Lozano-Durán. Coherent structures in wall-bounded turbulence. *Journal of Fluid Mechanics*, 842, 2018.

[105] J. Jiménez and P. Moin. The minimal flow unit in near-wall turbulence. *Journal of Fluid Mechanics*, 225:213–240, 1991.

[106] M. S. Jolly, I. G. Kevrekidis, and E. S. Titi. Approximate inertial manifolds for the Kuramoto-Sivashinsky equation: Analysis and computations. *Physica D: Nonlinear Phenomena*, 44, 1990.

[107] M. S. Jolly, R. Rosa, and R. Temam. Evaluating the dimension of an inertial manifold for the Kuramoto-Sivashinsky equation. *Advances in Differential Equations*, 5(1-3): 31–66, 2000.

[108] A.-K. Kassam and L. N. Trefethen. Fourth-order time-stepping for stiff PDEs. *SIAM Journal on Scientific Computing*, 26(4):1214–1233, 2005.

[109] G. Kawahara and S. Kida. Periodic motion embedded in plane couette turbulence: regeneration cycle and burst. *Journal of Fluid Mechanics*, 449:291–300, 2001.

[110] G. Kawahara, M. Uhlmann, and L. van Veen. The significance of simple invariant solutions in turbulent flows. *Annual Review of Fluid Mechanics*, 44(1):203–225, 2012.

[111] P. G. Kevrekidis, J. Cuevas-Maraver, Y. Drossinos, Z. Rapti, and G. A. Kevrekidis. Reaction-diffusion spatial modeling of covid-19: Greece and andalusia as case examples. *Phys. Rev. E*, 104:024412, Aug 2021.

[112] H. Khalil. *Nonlinear Systems*. Pearson Education. Prentice Hall, 2002.

[113] D. P. Kingma and J. L. Ba. Adam: A method for stochastic optimization. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pages 1–15, 2015.

[114] D. P. Kingma and M. Welling. Auto-Encoding Variational Bayes. *arXiv preprint arXiv:1312.6114*, 2013.

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

[116] P. V. Kuptsov and A. V. Kuptsova. Estimating of the inertial manifold dimension for a chaotic attractor of complex Ginzburg-Landau equation using a neural network . In D. E. Postnov, editor, *Saratov Fall Meeting 2018: Computations and Data Analysis: from Nanoscale Tools to Brain Functions*, volume 11067, page 110670N. International Society for Optics and Photonics, SPIE, 2019.

[117] J. N. Kutz, S. L. Brunton, B. W. Brunton, and J. L. Proctor. *Dynamic Mode Decomposition*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2016.

[118] A. Lasota and M. C. Mackey. *Chaos, Fractals and Noise: stochastic aspects of dynamics*. Springer. Springer, New York, second edition, 1994.

[119] C. Lee, J. Kim, D. Babcock, and R. Goodman. Application of neural networks to turbulence control for drag reduction. *Physics of Fluids*, 9(6):1740–1747, 1997.

[120] J. Lee. *Introduction to Smooth Manifolds*. Graduate Texts in Mathematics. Springer, 2003.

[121] K. Lee and K. T. Carlberg. Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders. *Journal of Computational Physics*, 404: 108973, 2020.

[122] R. Legenstein and W. Maass. Edge of chaos and prediction of computational performance for neural circuit models. *Neural Networks*, 20(3):323–334, 2007. Echo State Networks and Liquid State Machines.

[123] S. Levine, A. Kumar, G. Tucker, and J. Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *CoRR*, abs/2005.01643, 2020.

[124] J. Li and M. Zhang. Reinforcement-learning-based control of confined cylinder wakes with stability analyses. *Journal of Fluid Mechanics*, 932:A44, 2022.

[125] Q. Li, F. Dietrich, E. M. Bollt, and I. G. Kevrekidis. Extended dynamic mode decomposition with dictionary learning: A data-driven adaptive spectral decomposition of the koopman operator. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 27 (10):103111, 2017.

[126] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, 2016.

[127] H. Ling and K. Okada. An efficient earth mover's distance algorithm for robust histogram comparison. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(5):840–853, 2007.

[128] A. J. Linot and M. D. Graham. Deep learning to discover and predict dynamics on an inertial manifold. *Phys. Rev. E*, 101:062209, 2020.

[129] A. J. Linot and M. D. Graham. Data-driven reduced-order modeling of spatiotemporal chaos with neural ordinary differential equations. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 32(7):073110, 2022.

[130] A. J. Linot and M. D. Graham. KSE_ROM_NODE. `https://github.com/alinot5/KSE_ROM_NODE`, 2022.

[131] A. J. Linot and M. D. Graham. Dynamics of a data-driven low-dimensional model of turbulent minimal Couette flow. *arXiv preprint arXiv:2301.04638*, 2023.

[132] A. J. Linot, J. W. Burby, Q. Tang, P. Balaprakash, M. D. Graham, and R. Maulik. Stabilized neural ordinary differential equations for long-time forecasting of dynamical systems. *Journal of Computational Physics*, 474:111838, 2023.

[133] A. J. Linot, K. Zeng, and M. D. Graham. PyChannel. `https://github.com/alinot5/PyChannel`, 2023.

[134] X.-Y. Liu and J.-X. Wang. Physics-informed Dyna-style model-based deep reinforcement learning for dynamic control. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 477(2255):20210618, 2021.

[135] E. N. Lorenz. Deterministic Nonperiodic Flow. *Journal of the Atmospheric Sciences*, 20(2):130–141, 1963.

[136] H. F. Lui and W. R. Wolf. Construction of reduced-order models for fluid flows using deep feedforward neural networks. *Journal of Fluid Mechanics*, 872:963–994, 2019.

[137] B. Lusch, J. N. Kutz, and S. L. Brunton. Deep learning for universal linear embeddings of nonlinear dynamics. *Nat Comms*, 9(1):851–10, Nov. 2018.

[138] M. Marion and R. Temam. Nonlinear Galerkin methods: The finite elements case. *Numerische Mathematik*, 57(1):205–226, 1990.

[139] H. G. Matthies and M. Meyer. Nonlinear Galerkin methods for the model reduction of nonlinear dynamical systems. *Computers and Structures*, 81(12):1277–1286, 2003.

[140] R. Maulik, A. Mohan, B. Lusch, S. Madireddy, P. Balaprakash, and D. Livescu. Time-series learning of latent-space dynamics for reduced-order model closure. *Physica D: Nonlinear Phenomena*, 405:132368, 2020.

[141] L. McInnes, J. Healy, N. Saul, and L. Großberger. Umap: Uniform manifold approximation and projection. *Journal of Open Source Software*, 3(29):861, 2018.

[142] S. Meyn. *Control Systems and Reinforcement Learning*. Cambridge University Press, 2022.

[143] M. Milano and P. Koumoutsakos. Neural network modeling for near wall turbulent flow. *Journal of Computational Physics*, 182(1):1–26, 2002.

[144] J. Moehlis, T. R. Smith, P. Holmes, and H. Faisst. Models for turbulent plane couette flow using the proper orthogonal decomposition. *Physics of Fluids*, 14(7):2493–2507, 2002.

[145] J. Moehlis, H. Faisst, and B. Eckhardt. A low-dimensional model for turbulent shear flows. *New Journal of Physics*, 6(1):56, may 2004.

[146] P. Moin and R. D. Moser. Characteristic-eddy decomposition of turbulence in a channel. *Journal of Fluid Mechanics*, 200:471–509, 1989.

[147] T. Murata, K. Fukami, and K. Fukagata. Nonlinear mode decomposition with convolutional neural networks for fluid dynamics. *Journal of Fluid Mechanics*, 882:A13, 2020.

[148] O. Nachum, S. S. Gu, H. Lee, and S. Levine. Data-efficient hierarchical reinforcement learning. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[149] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine. Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning. In *2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018*, pages 7559–7566. IEEE, 2018.

[150] M. Nagata. Three-dimensional finite-amplitude solutions in plane couette flow: Bifurcation from infinity. *Journal of Fluid Mechanics*, 217:519–527, 1990.

[151] N. J. Nair and A. Goza. Leveraging reduced-order models for state estimation using deep learning. *Journal of Fluid Mechanics*, 897:1–13, 2020.

[152] T. Nakamura, K. Fukami, K. Hasegawa, Y. Nabae, and K. Fukagata. Convolutional neural network and long short-term memory based reduced order surrogate for minimal turbulent channel flow. *Physics of Fluids*, 33(2):025116, 2021.

[153] A. Y. Okulov. Structured light entities, chaos and nonlocal maps. *Chaos, Solitons and Fractals*, 133:109638, 2020.

[154] N. Omata and S. Shirayama. A novel method of low-dimensional representation for temporal behavior of flow fields using deep autoencoder. *AIP Advances*, 9(1):015006, Jan. 2019.

[155] OpenAI, C. Berner, G. Brockman, B. Chan, V. Cheung, P. P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. Pondé, D. O. Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski, and S. Zhang. Dota 2 with Large Scale Deep Reinforcement Learning. *ArXiv Preprint*, 2019.

[156] S. E. Otto and C. W. Rowley. Linearly Recurrent Autoencoder Networks for Learning Dynamics. *SIAM Journal on Applied Dynamical Systems*, 18(1):558–593, 2019.

[157] J. Page, M. P. Brenner, and R. R. Kerswell. Revealing the state space of turbulence using machine learning. *Phys. Rev. Fluids*, 6:034402, 2021.

[158] V. Y. Pan and Z. Q. Chen. The complexity of the matrix eigenproblem. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, STOC '99, pages 507–516, New York, NY, USA, 1999. Association for Computing Machinery.

[159] R. Paris, S. Beneddine, and J. Dandois. Robust flow control and optimal sensor placement using deep reinforcement learning. *Journal of Fluid Mechanics*, 913:A25, 2021.

[160] J. Park and H. Choi. Machine-learning-based feedback control for drag reduction in a turbulent channel flow. *Journal of Fluid Mechanics*, 904:A24, 2020.

[161] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[162] J. Pathak, B. Hunt, M. Girvan, Z. Lu, and E. Ott. Model-Free Prediction of Large Spatiotemporally Chaotic Systems from Data: A Reservoir Computing Approach. *Phys. Rev. Lett.*, 120(2):024102, Jan. 2018.

[163] C. E. Pérez De Jesús and M. D. Graham. Data-driven low-dimensional dynamic model of Kolmogorov flow. *arXiv preprint*, 2022.

[164] R. Peyret. *Spectral Methods for Incompressible Viscous Flow*. Applied Mathematical Sciences. Springer New York, 2002.

[165] S. Pirozzoli, J. Romero, M. Fatica, R. Verzicco, and P. Orlandi. One-point statistics for turbulent pipe flow up to $Re_\tau \approx 6000$. *Journal of Fluid Mechanics*, 926:A28, 2021.

[166] S. B. Pope. *Turbulent Flows*. Cambridge University Press, 2000.

[167] G. D. Portwood, P. P. Mitra, M. D. Ribeiro, T. M. Nguyen, B. T. Nadiga, J. A. Saenz, M. Chertkov, A. Garg, A. Anandkumar, A. Dengel, R. Baraniuk, and D. P. Schmidt. Turbulence forecasting via neural ODE. *arXiv preprint*, 2019.

[168] J. L. Proctor, S. L. Brunton, and J. N. Kutz. Dynamic Mode Decomposition with Control. *SIAM Journal on Applied Dynamical Systems*, 15(1):142–161, 2016.

[169] S. Qin, S. Wang, J. Rabault, and G. Sun. An application of data driven reward of deep reinforcement learning by dynamic mode decomposition in active flow control. *ArXiv Preprint*, 2021.

[170] M. Quadrio and P. Ricco. Critical assessment of turbulent drag reduction through spanwise wall oscillations. *Journal of Fluid Mechanics*, 521:251–271, 2004.

[171] J. Rabault, M. Kuchta, A. Jensen, U. Réglade, and N. Cerardi. Artificial neural networks trained through deep reinforcement learning discover control strategies for active flow control. *Journal of Fluid Mechanics*, 865:281–302, 2019.

[172] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Multistep Neural Networks for Data-driven Discovery of Nonlinear Dynamical Systems. *arXiv preprint arXiv:1801.01236*, pages 1–19, 2018.

[173] M. Raissi, P. Perdikaris, and G. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

[174] H. Rebbeck and K.-S. Choi. A wind-tunnel experiment on real-time opposition control of turbulence. *Physics of Fluids*, 18(3):035103, 2006.

[175] D. Rempfer and H. Fasel. Evolution of three-dimensional coherent structures in a flat-plate boundary layer. *Journal of Fluid Mechanics*, 260:351–375, Feb. 1994.

[176] F. Ren, J. Rabault, and H. Tang. Applying deep reinforcement learning to active flow control in weakly turbulent conditions. *Physics of Fluids*, 33(3):037121, 2021.

[177] M. Revay, R. Wang, and I. R. Manchester. Recurrent equilibrium networks: Unconstrained learning of stable and robust dynamical models. In *2021 60th IEEE Conference on Decision and Control (CDC)*, pages 2282–2287, 2021.

[178] J. Robinson. *Infinite-Dimensional Dynamical Systems: An Introduction to Dissipative Parabolic PDEs and the Theory of Global Attractors*. Cambridge Texts in Applied Mathematics. Cambridge University Press, 2001.

[179] C. J. G. Rojas, A. Dengel, and M. D. Ribeiro. Reduced-order model for fluid flows via neural ordinary differential equations. *arXiv preprint arXiv:2102.02248*, 2021.

[180] S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.

[181] Y. Rubner, C. Tomasi, and L. Guibas. A metric for distributions with applications to image databases. In *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*, pages 59–66, 1998.

[182] S. H. Rudy, S. L. Brunton, J. L. Proctor, and J. N. Kutz. Data-driven discovery of partial differential equations. *Science Advances*, 3(4):e1602614, 2017.

[183] J. W. Sammon. A nonlinear mapping for data structure analysis. *IEEE Transactions on computers*, 18(5):401–409, 1969.

[184] O. San and A. E. Staples. Stationary two-dimensional turbulence statistics using a markovian forcing scheme. *Computers & Fluids*, 71:1–18, 2013.

[185] T. Sauer, J. A. Yorke, and M. Casdagli. Embedology. *Journal of Statistical Physics*, 65(3):579–616, 1991.

[186] B. Schölkopf, A. Smola, and K.-R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10(5):1299–1319, 07 1998.

[187] J. Shalf. The future of computing beyond moore's law. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 378(2166): 20190061, 2020.

[188] J. Shawe-Taylor and N. Cristianini. *Basic kernels and kernel types*, page 291óÀẼ326. Cambridge University Press, 2004.

[189] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[190] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419):1140–1144, 2018.

[191] G. I. Sivashinsky. On Flame Propagation Under Conditions of Stoichiometry. *SIAM Journal on Applied Mathematics*, 39(1):67–82, 1980.

[192] C. R. Smith and S. P. Metzler. The characteristics of low-speed streaks in the near-wall region of a turbulent boundary layer. *Journal of Fluid Mechanics*, 129:27–54, 1983.

[193] T. R. Smith, J. Moehlis, and P. Holmes. Low-dimensional modelling of turbulence using the proper orthogonal decomposition: A tutorial. *Nonlinear Dynamics*, 41(1): 275–307, 2005.

[194] K. so Choi. Near-wall structure of a turbulent boundary layer with riblets. *Journal of Fluid Mechanics*, 208(April 2006):417–458, 1989.

[195] D. Sondak and P. Protopapas. Learning a reduced basis of dynamical systems using an autoencoder. *Phys. Rev. E*, 104:034202, Sep 2021.

[196] T. Sonoda, Z. Liu, T. Itoh, and Y. Hasegawa. Reinforcement learning of control strategies for reducing skin friction drag in a fully developed channel flow. *arXiv preprint arXiv:2206.15355*, 2022.

[197] P. R. Spalart, R. D. Moser, and M. M. Rogers. Spectral methods for the Navier-Stokes equations with one infinite and two periodic directions. *Journal of Computational Physics*, 96(2):297–324, 1991.

[198] P. A. Srinivasan, L. Guastoni, H. Azizpour, P. Schlatter, and R. Vinuesa. Predictions of turbulent shear flows using deep neural networks. *Physical Review Fluids*, 4(5):1–14, 2019.

[199] G. Strang. *Linear Algebra and Learning from Data*. Wellesley-Cambridge Press, 2019.

[200] B. Straughan. *The Energy Method, Stability, and Nonlinear Convection*. Applied mathematical sciences. Springer-Verlag, 1992.

[201] S. Strogatz. *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry and Engineering*. Studies in nonlinearity. Westview, 2000.

[202] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction, Second Edition*. MIT Press, 2018.

[203] F. Takens. Detecting strange attractors in turbulence. In D. Rand and L.-S. Young, editors, *Dynamical Systems and Turbulence, Warwick 1980*, pages 366–381, Berlin, Heidelberg, 1981. Springer Berlin Heidelberg.

[204] R. Temam. *Infinite-Dimensional Dynamical Systems in Mechanics and Physics*. Springer, New York, NY, 1977.

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

[206] R. Temam. Inertial manifolds. *The Mathematical Intelligencer*, 12(4):68–74, 1990.

[207] R. Temam and X. Wang. Estimates on the lowest dimension of inertial manifolds for the Kuramoto-Sivasbinsky equation in the general case. *Differential and Integral Equations*, 7(3-4):1095–1108, 1994.

[208] J. B. Tenenbaum, V. de Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

[209] E. S. Titi. On approximate Inertial Manifolds to the Navier-Stokes equations. *Journal of Mathematical Analysis and Applications*, 149(2):540–557, 1990.

[210] L. N. Trefethen and D. Bau. Numerical Linear Algebra. *Numerical Linear Algebra with Applications*, 12:361, 1997.

[211] L. van der Maaten and G. Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[212] L. J. P. Van Der Maaten, E. O. Postma, and H. J. Van Den Herik. Dimensionality Reduction: A Comparative Review. *Journal of Machine Learning Research*, 10:1–41, 2009.

[213] P. Varela, P. Suárez, F. Alcántara-Ávila, A. Miró, J. Rabault, B. Font, L. M. García-Cuevas, O. Lehmkuhl, and R. Vinuesa. Deep reinforcement learning for flow control exploits different physics for increasing reynolds number regimes. *Actuators*, 11(12), 2022.

[214] S. Verma, G. Novati, and P. Koumoutsakos. Efficient collective swimming by harnessing vortices through deep reinforcement learning. *Proceedings of the National Academy of Sciences*, 115(23):5849–5854, 2018.

[215] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. P. Agapiou, M. Jaderberg, A. S. Vezhnevets, R. Leblond, T. Pohlen, V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T. L. Paine, C. Gulcehre, Z. Wang, T. Pfaff, Y. Wu, R. Ring, D. Yogatama, D. Wünsch, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, K. Kavukcuoglu, D. Hassabis, C. Apps, and D. Silver. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575 (7782):350–354, 2019.

[216] J. Viquerat, P. Meliga, and E. Hachem. A review on deep reinforcement learning for fluid mechanics: an update. *arXiv preprint arXiv:2107.12206*, 2023.

[217] P. S. Virk. Drag reduction fundamentals. *AIChE Journal*, 21(4):625–656, 1975.

[218] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[219] D. Viswanath. Recurrent motions within plane couette turbulence. *Journal of Fluid Mechanics*, 580:339–358, 2007.

[220] P. Vlachas, J. Pathak, B. Hunt, T. Sapsis, M. Girvan, E. Ott, and P. Koumoutsakos. Backpropagation algorithms and reservoir computing in recurrent neural networks for the forecasting of complex spatiotemporal dynamics. *Neural Networks*, 126:191–217, 2020.

[221] P. Vlachas, G. Arampatzis, C. Uhler, and P. Koumoutsakos. Learning the effective dynamics of complex multiscale systems. *arXiv preprint arXiv:2006.13431*, 2021.

[222] P. R. Vlachas, W. Byeon, Z. Y. Wan, T. P. Sapsis, and P. Koumoutsakos. Data-driven forecasting of high-dimensional chaotic systems with long short-term memory networks. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 474(2213):20170844, 2018-05.

[223] P. R. Vlachas, G. Arampatzis, C. Uhler, and P. Koumoutsakos. Multiscale simulations of complex systems by learning their effective dynamics. *Nature Machine Intelligence*, 4(4):359–366, 2022.

[224] J. Vukadinovic. Global dissipativity and inertial manifolds for diffusive burgers equations with low-wavenumber instability. *Discrete & Continuous Dynamical Systems*, 29 (1):327–341, 2011.

[225] N. Wahlström, T. B. Schön, and M. P. Deisenroth. From Pixels to Torques: Policy Learning with Deep Dynamical Models. *arXiv*, abs/1502.02251, 2015.

[226] F. Waleffe. On a self-sustaining process in shear flows. *Physics of Fluids*, 883(1997), 1997.

[227] F. Waleffe. Three-dimensional coherent states in plane shear flows. *Physical Review Letters*, 81(19):4140–4143, 1998.

[228] Z. Y. Wan, P. Vlachas, P. Koumoutsakos, and T. Sapsis. Data-assisted reduced-order modeling of extreme events in complex dynamical systems. *PLoS ONE*, 13(5):1–22, 2018.

[229] Y.-Z. Wang, Y.-F. Mei, N. Aubry, Z. Chen, P. Wu, and W.-T. Wu. Deep reinforcement learning based synthetic jet control on disturbed flow over airfoil. *Physics of Fluids*, 34(3):033606, 2022.

[230] Z. Wang and C. Guet. Reconstructing a dynamical system and forecasting time series by self-consistent deep learning. *CoRR*, abs/2108.01862, 2021.

[231] M. Watter, J. T. Springenberg, J. Boedecker, and M. A. Riedmiller. Embed to Control: A Locally Linear Latent Dynamics Model for Control from Raw Images. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 2746–2754, 2015.

[232] H. Whitney. Differentiable Manifolds. *Annals of Mathematics*, 37(3):645–680, 1936.

[233] H. Whitney. The self-intersections of a smooth n-manifold in 2n-space. *Annals of Mathematics*, 45(2):220–246, 1944.

[234] M. O. Williams, I. G. Kevrekidis, and C. W. Rowley. A DataóÀÊDriven Approximation of the Koopman Operator: Extending Dynamic Mode Decomposition. *Journal of Nonlinear Science*, 25(6):1307–1346, 2015.

[235] A. P. Willis, P. Cvitanović, and M. Avila. Revealing the state space of turbulent pipe flow by symmetry reduction. *Journal of Fluid Mechanics*, 721:514–540, 2013.

[236] H.-l. Yang and G. Radons. Geometry of inertial manifolds probed via a lyapunov projection method. *Phys. Rev. Lett.*, 108:154101, Apr 2012.

[237] H. L. Yang, K. A. Takeuchi, F. Ginelli, H. Chaté, and G. Radons. Hyperbolicity and the effective dimension of spatially extended dissipative systems. *Physical Review Letters*, 102(7):1–4, 2009.

[238] E. Yeung, S. Kundu, and N. Hodas. Learning deep neural network representations for koopman operators of nonlinear dynamical systems. In *2019 American Control Conference (ACC)*, pages 4832–4839, 2019.

[239] C. D. Young and M. D. Graham. Deep learning delay coordinate dynamics for chaotic attractors from partial observable data. *arXiv preprint arXiv:2211.11061*, 2022.

[240] H. Yu, X. Tian, W. E, and Q. Li. Onsagernet: Learning stable and interpretable dynamics using a generalized onsager principle. *Phys. Rev. Fluids*, 6:114402, Nov 2021.

[241] S. Zelik. Inertial manifolds and finite-dimensional reduction for dissipative PDEs. *Proceedings of the Royal Society of Edinburgh Section A: Mathematics*, 144(6):1245–1327, 2014.

[242] K. Zeng and M. D. Graham. Symmetry reduction for deep reinforcement learning active control of chaotic spatiotemporal dynamics. *Phys. Rev. E*, 104:014210, Jul 2021.

[243] K. Zeng, A. Linot, and M. D. Graham. Learning turbulence control strategies with data-driven reduced-order models and deep reinforcement learning. In *12th International Symposium on Turbulence and Shear Flow Phenomena (TSFP12) Osaka, Japan (Online), July 19-22, 2022*, 2022.

[244] K. Zeng, A. J. Linot, and M. D. Graham. Data-driven control of spatiotemporal chaos with reduced-order neural ODE-based models and reinforcement learning. *Proceedings of the Royal Society A*, 478(2267):20220297, 2022.