DEEP DATA-DRIVEN MODELING AND CONTROL OF HIGH-DIMENSIONAL NONLINEAR SYSTEMS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF AERONAUTICS AND ASTRONAUTICS
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Jeremy Morton
December 2019
I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Mykel Kochenderfer, Primary Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Zachary Manchester

Approved for the Stanford University Committee on Graduate Studies.

Patricia J. Gumport, Vice Provost for Graduate Education

This signature page was generated electronically upon submission of this dissertation in electronic format. An original signed hard copy of the signature page is on file in University Archives.
Abstract

The ability to derive models for dynamical systems is a central focus in many realms of science and engineering. However, for many systems of interest, the governing equations are either unknown or can only be evaluated to high accuracy at significant computational expense. Difficulties with modeling can be further exacerbated by additional complexities, such as high-dimensional states or nonlinearities in the dynamics. In turn, these challenges can hinder performance on important downstream tasks, such as prediction and control. This thesis presents techniques for learning dynamics models from data. By taking a data-driven approach, models can be derived even for systems with governing equations that are unknown or expensive to evaluate. Furthermore, training procedures can be tailored to provide learned models with desirable properties, such as low dimensionality (for efficient evaluation and storage) or linearity (for control).

The proposed techniques are primarily evaluated on their ability to learn from data generated by computational fluid dynamics (CFD) simulations. CFD data serves as an ideal test case for data-driven techniques because the simulated fluid flows are nonlinear and can exhibit a wide array of behaviors. Additionally, modeling and even storage of CFD data can prove challenging due to the large number of degrees of freedom in many simulations, which can cause time snapshots of the flow field to contain megabytes or even gigabytes of data.

First, this thesis proposes a multi-stage compression procedure to alleviate the storage overhead associated with running large-scale CFD simulations. Individual time snapshots are compressed through a combination of neural network autoencoders and principal component analysis. Subsequently, a dynamics model is learned that can
faithfully propagate the compressed representations in time. The proposed method is able to compress the stored data by a factor of over a million, while still allowing for accurate reconstruction of all flow solutions at all time instances.

The high computational cost of CFD simulations can make it impractical to run large numbers of simulations at diverse flow conditions. The second part of this thesis introduces a method for performing generative modeling, which allows for the efficient simulation of fluid flows at a wide range of flow conditions given data from only a subset of those conditions. The proposed method, which relies upon techniques from variational inference, is shown to generate accurate simulations at a range of conditions for both two- and three-dimensional fluid flow problems.

The equations that govern fluid flow are nonlinear, meaning that many control techniques, largely derived for linear systems, prove ineffective when applied to fluid flow control. This thesis proposes a method, grounded in Koopman theory, for discovering data-driven linear models that can approximate the forced dynamics of systems with nonlinear dynamics. The method is shown to produce stable dynamics models that can accurately predict the time evolution of airflow over a cylinder. Furthermore, by performing model predictive control with the learned models, a straightforward, interpretable control law is found that is capable of suppressing vortex shedding in the cylinder wake.

In the final part of this thesis, the Deep Variational Koopman (DVK) model is introduced, which is a method for inferring distributions over Koopman observations that can be propagated linearly in time. By sampling from the inferred distributions, an ensemble of dynamics models is obtained, which in turn provides a distribution over possible outcomes as a modeled system advances in time. Experiments show that the DVK model is capable of accurate, long-term prediction for a variety of dynamical systems. Furthermore, it is demonstrated that accounting for the uncertainty present in the distribution over dynamics models enables more effective control.
Acknowledgments

This work would not have been possible without the financial support and freedom provided by my funding sources. Thank you to Stanford University, the Stanford Strategic Energy Alliance, and the National Science Foundation Graduate Research Fellowship Program for funding various portions of my graduate studies.

Reflecting upon my experience at Stanford, the predominant sentiment I feel is a sense of luck to have worked with so many talented and competent people. Most of all, I feel fortunate to have worked with my advisor, Mykel Kochenderfer. Since I began working with Mykel, the number of students advised by him and enrolled in his courses has seemed to grow exponentially, yet he has somehow remained available, insightful, and enthusiastic throughout. His organization, attention to detail, positivity, and, most importantly, his kindness, have all helped to make a challenging process easier.

This would have been a very different thesis had I not begun working with Antony Jameson and Freddie Witherden a little over two years ago. Over that time, I’ve learned so much from both of them. Research is very rarely an individual effort, and my research certainly was not. I can’t even estimate how many questions the two of them have promptly and patiently answered in moments when I was feeling confused about certain topics or uncertain about what to work on next. I would additionally like to thank Freddie, as well as Zac Manchester, for serving on my reading committee, and Ken Hara and Ben Van Roy for serving on my defense committee.

Next I would like to thank my labmates and classmates in SISL and the Aero Astro department. In my coursework and research, I was lucky to be surrounded by friendly and collaborative people who always resisted the urge to make the environment overly serious or competitive. Because of the atmosphere they’ve created, I’ve made so many
great friends during my time at Stanford. Special thanks as well to all of my friends in Chicago for staying in touch and generously offering up their couch space whenever I’ve been in town.

Finally, I want to thank my family for creating such a supportive environment to grow up in (and return to over breaks), and for encouraging me to pursue a PhD. So much of what I know I’ve learned from you, and this thesis is built upon that foundation of knowledge.
# Generative Modeling of Dynamical Systems

3 Gen

3.1 Background ................................................. 37
3.2 Generative Modeling ................................. 39
  3.2.1 Variational Autoencoders .............................. 39
  3.2.2 Conditional Variational Autoencoders ................. 43
  3.2.3 Generative Modeling of Sequential Data ............... 44
  3.2.4 Parameter-Conditioned Generative Modeling .......... 48
3.3 Experiments .................................................. 51
  3.3.1 Counter-Rotating Cylinders ......................... 51
  3.3.2 Three-Dimensional Flow .............................. 66
3.4 Discussion .................................................. 71

# Dynamic Programming and LQR

4 Dyn

4.1 Dynamic Programming ................................. 73
4.2 The Linear Quadratic Regulator .......................... 76
  4.2.1 Alternative Cost Functions ......................... 78
4.3 The Iterative Linear Quadratic Regulator ............... 80
4.4 Differential Dynamic Programming ........................ 83
4.5 Discussion .................................................. 85

# Linear Dynamics and the Koopman Operator

5 Lin

5.1 Dynamic Mode Decomposition ......................... 87
5.2 Dynamic Mode Decomposition with Control ............... 90
  5.2.1 DMDc with known $B$-matrix ....................... 91
  5.2.2 DMDc with unknown $B$-matrix ..................... 91
5.3 The Koopman Operator ................................ 92
  5.3.1 Koopman Operator Example ....................... 95
5.4 The Koopman Operator and Control ..................... 97
  5.4.1 Closed-Loop Control Inputs ..................... 98
  5.4.2 Open-Loop Control Inputs ....................... 98
5.5 Discussion ................................................. 100

6 Koopman Invariant Subspaces for Control  ........................................... 101
  6.1 Background ............................................... 101
  6.2 Learning Koopman Invariant Subspaces  ........................................ 103
    6.2.1 Deep Koopman Model .................................. 104
  6.3 Test Case .................................................. 106
  6.4 Modeling Unforced Dynamics .............................................. 107
    6.4.1 Training Details ....................................... 107
    6.4.2 Baseline Models ........................................ 108
    6.4.3 Results .................................................. 109
  6.5 Modeling Forced Dynamics .............................................. 110
    6.5.1 Modified Test Case ..................................... 111
    6.5.2 Training Details ....................................... 112
  6.6 Model Predictive Control .......................................... 112
  6.7 Results ..................................................... 115
  6.8 Discussion .................................................. 118

7 Uncertainty-Aware Modeling and Control ............................................. 120
  7.1 Deep Variational Koopman Models ......................................... 121
    7.1.1 Inference Procedure .................................... 122
    7.1.2 Optimization Procedure .................................. 126
  7.2 Uncertainty-Aware Control .............................................. 129
    7.2.1 Approximate Quadratic Dynamics and Cost .................... 131
    7.2.2 Accounting for Uncertainty .................................. 132
  7.3 Experiments ................................................... 136
    7.3.1 Dynamics Modeling ....................................... 136
    7.3.2 Control .................................................. 142
  7.4 Discussion .................................................. 143

8 Conclusions ....................................................... 145
  8.1 Summary ..................................................... 145
8.2 Contributions ......................................................... 146
8.3 Future Work ......................................................... 148
   8.3.1 Incorporating Knowledge of Governing Equations ............ 148
   8.3.2 Accounting for Noise and Stochasticity ...................... 149
   8.3.3 Scaling to Larger or More Complex Problems ................. 149
   8.3.4 Emerging Machine Learning Techniques ..................... 149

A Generative Modeling Plots ........................................... 151

B Environment Governing Equations .................................. 155
List of Tables

1.1 Summary of Challenges ................................. 4

7.1 Neural Network Summary ............................... 127

7.2 Control Performance ................................. 143
List of Figures

1.1 Comparison of dynamics modeling approaches .................................. 5
2.1 Mesh visualization for the half-cylinder domain ................................. 14
2.2 Overview of the various compression stages ....................................... 16
2.3 Illustration of convolutional layer ..................................................... 19
2.4 Common activation functions ........................................................... 20
2.5 ResNet skip connections ................................................................. 21
2.6 Illustration of the network architecture for local compression ................. 23
2.7 Validation and test results ................................................................. 30
2.8 Relative error over time ................................................................. 31
2.9 Lift and drag coefficients on the surface of the cylinder ........................... 33
2.10 Iso-surfaces of Q-criterion .............................................................. 35
3.1 Illustration of VAE training procedure ............................................... 42
3.2 VAE samples from MNIST dataset ................................................... 43
3.3 Graphical models ........................................................................... 44
3.4 Graphical model for parameter-conditioned generative model ................. 48
3.5 Counter-rotating cylinder configuration ............................................ 52
3.6 Mesh used for simulation of counter-rotating cylinders ............................ 53
3.7 Format of inputs to the generative model .......................................... 53
3.8 Illustration of parameter space for counter-rotating cylinder simulations .... 54
3.9 Variation of lift with rotation speed .................................................. 55
3.10 Variation of $x$-velocity with rotation speed ...................................... 56
3.11 Variation of $x$-velocity with Reynolds number ................................... 56
3.12 Visual comparison of \( x \)-velocity over time at various flow conditions  
3.13 Mean lift coefficient as a function of rotation speed 
3.14 Variation of dominant frequency in lift signal 
3.15 Depiction of wake measurement location 
3.16 Pressure over time for CFD and generated solutions 
3.17 Variation of dominant frequency in pressure signal 
3.18 Identification of critical rotation speed 
3.19 Visualization of CFD-simulated and sampled flow 
3.20 Visual comparison of \( Q \)-values over time at various flow conditions 
3.21 Visual comparison of vorticity over time at various flow conditions 

4.1 Principle of optimality example 

6.1 Training algorithm for Deep Koopman models 
6.2 Mesh used for simulation of airflow over cylinder 
6.3 Format of inputs to Deep Koopman network 
6.4 Average prediction errors over time 
6.5 Illustration of actuation used to suppress vortex shedding 
6.6 Control inputs over time in training dataset 
6.7 MPC vortex suppression results 
6.8 Scaled residuals plotted alongside estimated cost 
6.9 MPC control inputs 
6.10 Measurement points for performing proportional control 
6.11 Calculated residuals over time for different measurement locations. 

7.1 Graphical model 
7.2 Illustration of distinction between DK and DVK models 
7.3 Approximate quadratic cost for DVK models 
7.4 Visualization of evaluation environments 
7.5 Predictions for the acrobot environment 
7.6 MSE and NLL for test data 

A.1 Density over time for CFD and generated solutions
A.2  $x$-velocity over time for CFD and generated solutions  . . . . . . . . 152
A.3  $y$-velocity over time for CFD and generated solutions  . . . . . . . . 152
A.4  Variation of dominant frequency in density signal  . . . . . . . . . . . . . 153
A.5  Variation of dominant frequency in $x$-velocity signal  . . . . . . . . . . 153
A.6  Variation of dominant frequency in $y$-velocity signal  . . . . . . . . . . 154
Chapter 1

Introduction

A dynamical system is a system that evolves with time. Examples of dynamical systems are varied, encompassing everything from airplanes to financial markets. The derivation of models that describe the evolution of dynamical systems serves as a central challenge in many domains of science and engineering, as the analysis, prediction, and control of dynamical systems all rely on the existence of such models. This thesis presents techniques for learning dynamics models from data. The following sections introduce key concepts relating to dynamics modeling and control, describe the associated challenges, and motivate the use of data-driven dynamics modeling. Subsequently, the field of computational fluid dynamics is introduced, which serves throughout this thesis as a testbed for many of the proposed techniques. The chapter concludes with a summary of contributions and an outline for the remainder of the thesis.

1.1 Modeling and Control of Dynamical Systems

The state of a dynamical system, denoted by \( \mathbf{x} \), represents a summary of the relevant information required to predict the system’s future behavior. In this thesis, it will be assumed that \( \mathbf{x} \in \mathbb{R}^n \), i.e. the state is an \( n \)-dimensional vector of real numbers. A dynamics model describes how the state changes with time. Dynamics models can be continuous-time or discrete-time. A continuous-time dynamics model provides a
mapping from the current state to the time derivative of the state:

\[
\frac{dx(\tau)}{d\tau} = \dot{x} = f(x),
\]

(1.1)

where \( \tau \in \mathbb{R}_{\geq 0} \). In contrast, a discrete-time dynamics model advances the state forward in time over a fixed time interval \( \Delta \tau \):

\[
x_{t+1} = F(x_t),
\]

(1.2)

where \( t \) is a time index and \( x_t = x((t-1) \cdot \Delta \tau) \). Because this thesis focuses on learning dynamics models from data, which is typically sampled over uniform intervals, an emphasis will be placed on discrete-time dynamics. The mapping from continuous- to discrete-time dynamics models can be obtained by integrating the continuous-time dynamics over the relevant time interval:

\[
x_{t+1} = F(x_t) = x_t + \int_{(t-1) \cdot \Delta \tau}^{t \cdot \Delta \tau} f(x(\tau))d\tau.
\]

(1.3)

Dynamical systems can be subject to control inputs that can alter their behavior. Examples of control inputs include acceleration/braking in a car or torques applied to the joints of a robot. The time evolution of the state in a system with control inputs is governed by a forced dynamics model:

\[
x_{t+1} = F(x_t, u_t),
\]

(1.4)

where \( u_t \in \mathbb{R}^p \) is a vector containing the control inputs.

Forced and unforced dynamics models are often constructed using knowledge about the physical laws that characterize a system’s behavior. For example, in classical mechanics, Newton’s second law or the Euler-Lagrange equation are commonly used to determine the equations of motion for a modeled system [1]. Once derived and calibrated, dynamics models can be used for many tasks; this thesis will focus primarily on prediction and control. Prediction entails forecasting how a system will behave in the future given knowledge of its current state and, perhaps, a set of control inputs.
that will be applied to the system. Control consists of selecting appropriate control inputs in order to make the system behave in a desired way. These two tasks are interrelated—being able to predict how a system will behave is a necessity in order to perform effective control.

*Linear dynamical systems* represent an important class of dynamical systems. A linear system is a system $g(\cdot)$ that obeys the homogeneity and additivity properties such that:

$$g(\alpha x + \beta y) = \alpha g(x) + \beta g(y),$$

where $\alpha$ and $\beta$ are scalars. In the absence of control inputs, discrete-time linear dynamical systems take the form:

$$x_{t+1} = Ax_t,$$

where $A \in \mathbb{R}^{n \times n}$. For systems with control inputs, linear dynamics models are given by:

$$x_{t+1} = Ax_t + Bu_t,$$

where $B \in \mathbb{R}^{n \times p}$. Linear dynamical systems have many desirable properties. Prediction can be performed efficiently and in closed-form for linear dynamical systems, which is not always true of nonlinear systems. Additionally, frequency domain techniques such as root locus and Bode plots [2], as well optimal control techniques such as the linear quadratic regulator (see Chapter 4), can be applied to controller design for broad classes of linear dynamical systems.

### 1.2 Challenges

There are several challenges that can arise when attempting to perform modeling and control on real-world dynamical systems. This thesis addresses a subset of these challenges, which are described within this section and summarized in Table 1.1.

One challenge is that most real-world dynamical systems are nonlinear. Nonlinear
TABLE 1.1: Summary of Challenges

<table>
<thead>
<tr>
<th>Challenge</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nonlinearity</td>
<td>Prediction and control more challenging than for linear systems</td>
</tr>
<tr>
<td>Modeling</td>
<td>Model may be unknown or too expensive to evaluate accurately</td>
</tr>
<tr>
<td>High-Dimensionality</td>
<td>Learning and storage are difficult for systems with many DOF</td>
</tr>
</tbody>
</table>

Systems are difficult to simulate because it is often hard to derive closed-form expressions for the state as a function of time from the governing equations. Furthermore, analysis and control of nonlinear systems can be challenging, as nonlinear systems often exhibit phenomena that are not observed in linear systems, such as limit cycles, chaos, and the presence of multiple fixed points [3]. Nonlinear control design techniques, including feedback linearization and sliding mode control [3], have been applied with success to many problems, but are generally not as broadly applicable as linear control techniques.

A second challenge is that modeling and control rely upon the ability to derive and apply a system’s governing equations. However, there exist many dynamical systems for which leveraging the dynamics in this manner is not yet possible or practical. For many complex systems of interest, such as biological processes or financial markets, the governing equations are not known. In other cases, the governing equations may be known, but cannot be solved analytically, requiring the use of computationally-expensive approximate solution techniques.

Despite the lack of (tractable) models that describe how dynamical systems will evolve over time, there is a growing ability to capture and store data describing how they have evolved over time. Technological advances such as the introduction of the internet and smartphones, as well improvements to modern computing hardware and software, have led to an explosion in the amount of data that is generated and stored. These vast quantities of data thus present an opportunity to learn dynamics models directly from data. The distinction between this data-driven modeling approach and the more standard physics-based modeling approach is illustrated in Fig. 1.1.

Often, data-driven modeling approaches must contend with additional challenges.
Figure 1.1: Comparison of physics-based dynamics modeling and data-driven dynamics modeling approaches for the pendulum system. Both approaches yield models that can be used for prediction and control.
One challenge is the *high dimensionality* of the data. As an example, datasets that are used to construct or calibrate climate models can include periodic temperature measurements from thousands of ground stations from around the globe [4]. Learning models to describe the time evolution of so many quantities can potentially present a much greater challenge than modeling systems with relatively few degrees of freedom (DOF). Beyond the challenge of modeling, however, there is also the challenge of *storing* high-dimensional data given the huge quantities of data being generated.

One domain where this is of legitimate concern is computational fluid dynamics (CFD), where fluid simulations can have millions or even billions of degrees of freedom, making it possible that simulation data can exceed the capacity of computer storage systems. A large portion of the experiments in this thesis are devoted to learning models for fluid flow from CFD data. As such, the following section contains an overview of computational fluid dynamics, describing the approaches for modeling fluid flow and motivating the use of CFD simulations as a testbed for the techniques presented in this thesis.

### 1.3 Computational Fluid Dynamics

A *fluid* is a substance that deforms when subjected to external stresses. *Fluid dynamics* is the study of fluid motion, and can be used to describe a wide range of phenomena, including the flow of air over wings, the flow of water around boats, and the flow of blood through veins. The equations that describe fluid flow can be derived from a set of conservation laws: the conservation of mass, momentum, and energy. These equations can be expressed as:

$$\frac{\partial \mathbf{q}}{\partial \tau} + \nabla \cdot f(\mathbf{q}, \nabla \mathbf{q}) = 0,$$

where $f(\cdot)$ is a flux function and $\mathbf{q} = [\rho, \rho v_x, \rho v_y, \rho v_z, \epsilon]^\top$ is a vector of fluid properties, wherein $\rho$ is the mass density of the fluid, $\mathbf{v} = [v_x, v_y, v_z]^\top$ is the fluid velocity vector, and $\epsilon$ is the total energy per unit volume.

Constitutive equations can be used to relate the density, momentum, and energy
quantities governed by Eq. (1.8) to other flow quantities of interest. The ideal gas law is an important constitutive relation for many gases, including air at standard conditions, for which intermolecular forces are negligible. The ideal gas law states that pressure $p$ can be determined through:

$$p = (\gamma - 1)e - \frac{1}{2}\rho\|v\|_2^2,$$

where $\gamma = c_p/c_v$ is the ratio of specific heats. Written in a different form, the ideal gas law can also relate pressure and density to the fluid temperature:

$$T = \frac{1}{c_v} \frac{1}{\gamma - 1} \frac{p}{\rho}.$$

The flux function in Eq. (1.8) can be written as $f = f^{(\text{inv})} - f^{(\text{vis})}$, the difference between inviscid and viscous components, where the inviscid terms are given by:

$$f^{(\text{inv})}(q, \nabla q) = \begin{bmatrix}
\rho v_x & \rho v_y & \rho v_z \\
\rho v_x^2 + p & \rho v_x v_y & \rho v_x v_z \\
\rho v_x v_y & \rho v_y^2 + p & \rho v_y v_z \\
\rho v_x v_z & \rho v_y v_z & \rho v_z^2 + p \\
v_x(e + p) & v_y(e + p) & v_z(e + p)
\end{bmatrix}
$$

and the viscous terms are described by:

$$f^{(\text{vis})}(q, \nabla q) = \begin{bmatrix}
0 & 0 & 0 \\
\tau_{xx} & \tau_{yx} & \tau_{zx} \\
\tau_{xy} & \tau_{yy} & \tau_{zy} \\
\tau_{xz} & \tau_{yz} & \tau_{zz} \\
v_i \tau_{ix} + \frac{\mu c_p}{Pr} \frac{\partial T}{\partial x} & v_i \tau_{iy} + \frac{\mu c_p}{Pr} \frac{\partial T}{\partial y} & v_i \tau_{iz} + \frac{\mu c_p}{Pr} \frac{\partial T}{\partial z}
\end{bmatrix}.$$

Here, $\tau$ is the stress-energy tensor, $\mu$ is the dynamic viscosity of a fluid, and $Pr$ is the Prandtl number, which defines the ratio of viscous dissipation to thermal conduction.

A Newtonian fluid is a fluid for which the viscous stresses are proportional to the fluid strain rate. Both air and water are Newtonian fluids. Under the assumption of
Newtonian fluids, the stress-energy tensor can be written in index notation as:

$$T_{ij} = \mu \left( \frac{\partial v_j}{\partial x_i} + \frac{\partial v_i}{\partial x_j} \right) - \frac{2}{3} \mu \delta_{ij} \nabla \cdot \mathbf{v},$$  \hspace{1cm} (1.13)

where $\delta_{ij}$ is the Kronecker delta and the partial derivatives represent the spatial derivatives of the fluid velocity. When the stress-energy tensor is written in this form, the governing equations become the Navier–Stokes equations [5]. The Navier–Stokes equations are a single set of expressions that can describe a vast array of behaviors, encompassing everything from steady laminar flow to chaotic turbulent flow. A key dimensionless quantity that governs these changes in behavior is the Reynolds number, which defines the ratio of inertial forces to viscous forces in a fluid flow, and is given by:

$$Re = \frac{\rho v L}{\mu},$$  \hspace{1cm} (1.14)

where $\rho$ is the velocity magnitude of an incident flow and $L$ is a characteristic length (such as the chord length of a wing).

In their basic form, the Navier–Stokes equations are nonlinear partial differential equations. However, the equations can be simplified according to assumptions made about the modeled flow. For example, flows that are both incompressible ($\rho$ is constant) and irrotational (vorticity is zero) can be shown to have a velocity potential function $\phi(\cdot)$ that satisfies Laplace’s equation [5]:

$$\nabla^2 \phi = 0,$$  \hspace{1cm} (1.15)

where the relationship between the flow velocity vector $\mathbf{v}$ and the velocity potential function is given by:

$$\mathbf{v} = \left[ \frac{\partial \phi}{\partial x}, \frac{\partial \phi}{\partial y}, \frac{\partial \phi}{\partial z} \right]^T.$$  \hspace{1cm} (1.16)

Laplace’s equation has been widely studied and has many known solutions. Thus, incompressible, irrotational flows are one class of flows for which closed-form expressions can be derived, but there exist few other flows of practical interest for which the analytical solutions are known.
Due to the difficulty of deriving analytical solutions to the Navier–Stokes equations, the field of computational fluid dynamics (CFD) aims to devise numerical techniques for fluid flow simulation. CFD relies upon discretizations in space and time to obtain approximate solutions to the Navier–Stokes equations. Several techniques have been proposed for performing the spatial discretization; three of the more popular approaches are finite difference [6], finite volume [7], and finite element [8] schemes.

Recent advances have enabled high-fidelity CFD simulation of industrially relevant flows for a variety of geometries and flow conditions [9]. However, these simulation capabilities come at a cost. The large number of degrees of freedom and the complexity inherent to fluid flows means that, even with high levels of parallelization and state-of-the-art hardware, it can take hours or even days to carry out simulations for many flows of interest. This high computational cost means that, for example, running many CFD simulations in the inner loop of a design procedure can become impractical. Furthermore, analysis and post-processing of these simulations potentially requires storing many solution snapshots for numerous flow conditions, which can comprise petabytes of simulation data.

In summary, fluid flows serve as compelling test cases for the techniques presented in this thesis because:

1. The governing equations are nonlinear, meaning that the design of fluid flow control schemes can be difficult.

2. Fluid simulations are often computationally expensive, motivating the learning of models that can be evaluated more efficiently.

3. CFD data can be extremely high-dimensional, with modeled systems potentially containing billions of degrees of freedom.

1.4 Overview and Summary of Contributions

This thesis introduces techniques for addressing the challenges listed in Table 1.1. A common thread is the use of neural networks [10], which are parameterized models that have the ability to learn complex nonlinear relationships from high-dimensional
data. In particular, neural networks are used to learn state mappings under which the simulated dynamics become lower-dimensional, more efficient to evaluate, and (in some cases) linear. Much of the work presented here builds upon prior publications [11–13]. While the proposed techniques are largely applied to computational fluid dynamics simulations, it is worth emphasizing their generality. All modeling is performed in an equation-free context, meaning that no knowledge of the governing equations is assumed. As such, the presented methods are broadly applicable to dynamical systems for which data is available, regardless of whether the data is obtained through simulation, experiments, or real-world measurements.

An outline of the thesis is given below. Chapters 4 and 5 provide a concise review of background material that is valuable for subsequent chapters, while Chapters 2, 3, 6, and 7 present the methods and results that together comprise the contributions of this thesis. These contributions are described at a high level below; they will be summarized in greater depth in Chapter 8.

Chapter 2 introduces a three-stage procedure for compressing data from high-fidelity computational fluid dynamics simulations with fine spatial discretizations. The first compression stage uses a neural network autoencoder to compress data contained within individual mesh elements. The second stage then uses piecewise principal component analysis to compress data across elements and obtain a lower-dimensional reduced state. The final compression stage learns a dynamics model to propagate the reduced states forward in time. The procedure is shown to compress the CFD data by a factor of over a million while still retaining key characteristics of the fluid flow.

Chapter 3 presents a method for performing generative modeling of physical systems. The proposed framework allows for efficient simulations with behavior that can vary according to prescribed physical parameters. Applied to a two-dimensional fluid flow problem, the approach is shown to be capable of performing simulations at a wide range of flow conditions with much greater efficiency than CFD simulations. Furthermore, it is explained how this enhanced efficiency can be used to guide decisions within a design procedure. The ability of the approach to scale to more challenging three-dimensional fluid flows is also demonstrated.

Chapter 4 reviews dynamic programming, with a specific emphasis on the linear
CHAPTER 1. INTRODUCTION

quadratic regulator (LQR). Two extensions to LQR, the iterative linear quadratic regulator and differential dynamic programming, are also described. These techniques provide the foundation for the control approaches used in subsequent chapters.

Chapter 5 discusses dynamic mode decomposition (DMD), a data-driven method that yields approximate linear dynamics models. Modifications to DMD that account for the effect of control inputs are described. Later, Koopman theory is reviewed, which is a method for mapping nonlinear dynamics to linear dynamics. Connections between dynamic mode decomposition and Koopman theory are discussed.

Chapter 6 connects the ideas from Chapters 4 and 5, providing a novel method for data-driven modeling that yields linear dynamics models for nonlinear dynamical systems, thereby making the learned models amenable to control. The method is applied to a canonical problem in fluid dynamics: the suppression of vortex shedding over a cylinder. The modeling and control procedure is shown to be successful at suppressing vortex shedding. Examining the selected control inputs shows that they are interpretable, allowing for the derivation of a simple control law that is both effective and easy to implement.

Chapter 7 extends the ideas from Chapter 6 to allow for uncertainty-aware dynamics modeling and control, introducing the Deep Variational Koopman (DVK) model, which allows for the sampling of an ensemble of linear models that, taken together, can encode uncertainty about how a modeled system will advance in time. In experiments, DVK models are effective at modeling a variety of dynamical systems while providing reasonable uncertainty estimates. Furthermore, a discussion is provided about how to incorporate the trained models into the control algorithms described in Chapter 4. Finally, it is shown on the inverted pendulum task that accounting for the uncertainty encoded by the ensemble of sampled models can allow for more effective control.

Chapter 8 provides a summary of this thesis along with ideas for further research.
Chapter 2

Multi-Stage Compression

This chapter introduces a multi-stage procedure for compressing data from computational fluid dynamics simulations. The compression procedure consists of first learning a mapping from the high-dimensional flow state to a reduced representation, then learning a dynamics model that can propagate the reduced representations forward in time. The chapter begins by providing motivation for this work and describing the test case, after which the compression procedure is introduced and evaluated.

2.1 Background and Motivation

Due to the large number of spatial degrees of freedom and small time steps required for many high-fidelity compressible flow simulations, such simulations have the potential to generate petabytes of solution data; this is well beyond what can be handled by current I/O and storage subsystems. Therefore, it is not practical to write out a sequence of finely spaced solution snapshots to disk for offline analysis and post-processing. Instead, the practitioner must instrument the simulation in advance. Such instrumentation typically includes the online accumulation of various volumetric expressions, integrating forces on boundaries, and the regular sampling of the flow field at specific points in the volume. However, doing this effectively requires a degree of a priori knowledge about the dynamics and evolution of the system, which negates many of the exploratory advantages inherent to simulation.
A common strategy for reducing the storage overhead associated with CFD simulations is to compress the data before writing to disk [14–16]. Although these approaches can substantially reduce file sizes, they do not necessarily reduce the overall number of time instances at which a file must be written. Additionally, several methods have been proposed to reconstruct missing simulation data, especially in the context of CFD. However, most of these methods either aim to reconstruct missing spatial data [17, 18] or are not amenable to scenarios where no data has been written at some time instances [19–21].

In contrast, this chapter considers the problem of accurately recovering full CFD solutions from compressed representations at any time instance, given that the representations have been written to disk at only a relatively small number of time instances. The test case under study is introduced in the next section, after which the remainder of the chapter is devoted to describing and evaluating the proposed compression procedure.

2.2 Test Case

The flow over non-streamlined, bluff bodies such as cylinders has been the subject of extensive experimental and numerical study. The scientific interest in such flows is motivated by a phenomenon known as vortex-induced vibration, where wake instabilities lead to unsteady vortex shedding over the surface of bluff bodies. Vortex-induced vibrations can arise in many engineered structures, such as buildings, bridges, and offshore piers. Vortex shedding gives rise to strong transverse forces, and is associated with higher drag and unsteady lift forces [22, 23]. The collapse of the Tacoma Narrows Bridge in 1940 serves as a particularly stark illustration of the destructive potential of the induced structural vibrations [24].

Due to the prevalence and severity of vortex-induced vibration, as well as the wide body of existing literature on flow over a cylinder, variants of the cylinder system will be considered for study throughout this thesis. The chosen test case for this chapter is the flow over an extruded half-cylinder at Reynolds number \( \text{Re} = 350 \) and an effectively incompressible Mach number of \( M = 0.2 \). This flow exhibits several
complex flow features, including separated shear layers, turbulent transition, and a fully turbulent wake. The dynamics of the half-cylinder are noteworthy relative to the circular cylinder because the flow separation point remains fixed at the boundary between the curved and flat surfaces. Flow over this configuration has been the subject of several numerical and experimental studies [25–27].

In the given test case, the cylinder is taken to have a diameter $d$ along its major axis. The domain is taken to be $[-9d, 25d]$, $[-9d, 9d]$, and $[0, \pi d]$ in the stream-, cross-, and span-wise directions, and the cylinder is positioned such that the back surface is centered at $(0, 0, 0)$. To nondimensionalize the system, the cylinder diameter is taken to be $d = 1$, the free-stream density is taken to be one, and the free-stream pressure is taken to be one. The free-stream velocity is thus $v_\infty = \sqrt{\gamma} M \approx 0.237$, where $\gamma = 1.4$ is the ratio of specific heats. The compressible Navier–Stokes equations are solved with viscosity fixed such that the Reynolds number based on the diameter of the cylinder is $Re = 350$.

A visualization of the mesh can be found in Fig. 2.1. The domain is divided into $N_{el} = 40584$ non-overlapping, conforming hexahedral elements. The simulation is
CHAPTER 2. MULTI-STAGE COMPRESSION

performed using the flux reconstruction (FR) approach [28], a high-order numerical method for unstructured grids. The solution inside each element is represented as a multivariate, discontinuous, third-order polynomial with Gauss-Legendre solution points. Due to the third-order solution polynomials, the local state within each element contains the density, \(x\)-momentum, \(y\)-momentum, \(z\)-momentum, and energy at 64 points, comprising a total of 320 values within each element. Thus, the dimensionality of the global state is \(\approx 13 \times 10^6\), which implies that a given solution consumes \(\sim 100\)MB of storage using double-precision arithmetic.

The system is simulated with the PyFR solver [29], starting at a constant initial condition corresponding to free-stream at time \(\tau = 0\) and terminating at time \(\tau = 600\). This corresponds to approximately 71 stream-wise passes over the half-cylinder. The initial phase of the simulation is discarded due to the cold start at free-stream conditions, leaving a time domain of interest comprised of \(\tau \in \{170.05 + 0.05t\}_{t=1}^{T}\) with \(T = 8600\) total time instances. Thus, in total the sequence of time snapshots consumes \(\sim 860\)GB of storage.

2.3 Compression Overview

To reduce the storage requirements associated with running simulations such as the half-cylinder test case described above, this chapter introduces a novel two-stage compression strategy consisting of: 1) dimensionality reduction and 2) dynamics learning. In dimensionality reduction, an (approximately) invertible mapping is learned between the global flow states \(\mathbf{x}_{1:T}\) and a (much) lower-dimensional set of reduced states \(\mathbf{z}_{1:T}\). Subsequently, the reduced states can be stored in place of global flow states, with approximate global flow states attainable by applying the learned inverse mapping. The storage requirements can then be further reduced by learning a dynamics model to describe the time evolution of the reduced states, thereby lessening the number of time instances at which the reduced states must be stored. An illustration of the compression procedure can be found in Fig. 2.2. The following subsections provide a high-level overview of the compression stages, with more detailed descriptions provided in Sections 2.4 to 2.6.
Figure 2.2: Overview of the various compression stages. First the individual time snapshots are compressed using a combination of local and global compression. Subsequently, temporal compression is performed by learning a dynamics model for the reduced states.
2.3.1 Local Compression

Let $r(\cdot)$ be the learned mapping between the global and reduced states and $p(\cdot)$ be the learned inverse mapping; dimensionality reduction will ideally satisfy $\tilde{x}_t \approx x_t$, $t = 1, \ldots, T$, where

$$
\tilde{x}_t = p(r(x_t)).
$$

This chapter introduces a novel hierarchical approach to dimensionality reduction that leverages the structure of high-order discretizations. It comprises local compression using autoencoders followed by global compression using principal component analysis (PCA).

This procedure computes an accurate, low-dimensional, nonlinear embedding of the original data while remaining computationally tractable for large-scale data. An autoencoder is a (typically deep) neural network that performs dimensionality reduction by learning a nonlinear mapping from a high-dimensional feature space to a lower-dimensional encoding [30]. Unfortunately, memory constraints prohibit training autoencoders on full solutions for many large-scale CFD simulations. Furthermore, due to the large number of parameters that must be optimized, neural networks typically require many training examples, yet full solutions may only be written to disk at a relatively small number of time instances. Thus, the local compression phase employs autoencoders to compress the degrees of freedom within individual mesh elements rather than full solutions. The network inputs are therefore of reasonable size (and pose no memory issue), and the number of training examples is the number of time instances at which the solution has been written to disk multiplied by the number of elements in the mesh (which may be very large, e.g. $\sim 10^6$).

Global compression is necessary because even if autoencoders significantly reduce the number of degrees of freedom in each element, the number of elements in the mesh may still be large. Principal component analysis is therefore applied (with some modifications to handle large-scale data) to reduce the dimensionality of the vector of local encodings across the entire spatial domain. Although PCA identifies a linear subspace, PCA in combination with autoencoders yields a nonlinear embedding of the global CFD solution.
CHAPTER 2. MULTI-STAGE COMPRESSION

2.3.2 Dynamics Learning

The second stage of the compression procedure is dynamics learning. The goal is to learn a discrete-time dynamics model that approximates the time evolution of the low-dimensional reduced states:

$$\tilde{z}_{t+1} = F(\tilde{z}_t), \ t = 1, \ldots, T - 1,$$

where $\tilde{z}_1 = z_1 = r(x_1)$. The function $F(\cdot)$ represents a (possibly nonlinear) mapping from the reduced state at one time instance to the reduced state at the next time instance. Given $F(\cdot)$, the entire sequence of approximate reduced states can be recovered by specifying only the initial state $z_1$.

2.4 Local Compression with Autoencoders

A standard autoencoder consists of two components: (1) an encoder network that performs the mapping between input data and the encodings and (2) a decoder network that attempts to reconstruct the original input from the encodings. The dimensionality of the encoding represents a trade-off, as lower-dimensional encodings offer higher levels of compression at the expense of lower reconstruction accuracy. The autoencoder and its variants have previously been used extensively for discovering low-dimensional representations of image data [31–33]. Image inputs pose a particular challenge for parameterized models such as autoencoders and image classifiers because their high dimensionality requires the model to have many parameters.

2.4.1 Convolutional Neural Networks

Convolutional neural networks [34] exploit the spatial correlations inherent to image data to achieve an overall reduction in the number of trainable parameters. The input into each layer of a convolutional neural network is transformed by a set of filters; each filter has an effective receptive field (e.g. $3 \times 3$) that is convolved with the entire input. The application of each filter to the entire input tends to result in a significant
parameter savings relative to fully-connected neural network architectures that learn distinct parameters for each element of the input.

A single convolutional layer, illustrated in Fig. 2.3, is characterized by four parameters:

- \( f \), the number of filters
- \( v \), the width of each filter
- \( s \), the stride
- \( p \), the amount of zero-padding

The filter width \( v \) defines each filter’s receptive field. The stride \( s \) defines the increment of movement as each filter is swept across the input; a stride of one centers the filter on each spatial location in the input, while a stride of two skips every other location. The zero padding \( p \) governs the number of rows/columns of zeros to be added to the boundary of the input, and can be altered to modify the size of the layer output. For an input of size \( w \times h \times d \), a layer output will have size \( \hat{w} \times \hat{h} \times \hat{d} \), where:

\[
\begin{align*}
\hat{w} &= (w - v + 2p)/s + 1, \\
\hat{h} &= (h - v + 2p)/s + 1, \\
\hat{d} &= f.
\end{align*}
\] (2.3)
A typical (non-grayscale) image will have \( d = 3 \) channels at each spatial location, where each channel corresponds to the red, blue, and green color intensity, respectively.

Each convolutional transformation is commonly followed by a nonlinearity known as an activation function. Common activation functions include sigmoid functions, hyperbolic tangents, and rectified linear units (ReLUs) [35]. See Fig. 2.4 for an illustration of the various activation functions. A deep convolutional neural network chains together many convolutional transformations and activation functions, which allows filters in later layers to have a larger effective receptive field on the network input. This enables deep convolutional networks to serve as hierarchical feature extractors, where early layers learn finer features of the input image, while later layers extract coarser features.

Much of the success of neural networks in learning to perform image-related tasks is attributed to this hierarchical feature extraction that occurs with deep architectures. For instance, in image classification, the classification accuracy of deep neural networks tends to be much higher than that of their shallow counterparts. However, this trend of improved performance with network depth only holds up to a point (\( \sim 50 \) layers), after which performance gains tend to slow significantly or even diminish entirely as more layers are added. ResNet architectures [36], illustrated in Fig. 2.5, introduce skip connections that can circumvent any layer that is not useful for the learning task, thereby allowing for the training of deeper architectures with corresponding performance improvements. ResNets achieve strong performance on
image classification tasks while demonstrating classification improvements even for very deep architectures ($\sim$ 150 layers).

The parameters of a neural network are typically learned through stochastic gradient descent by finding the gradient of the network parameters $\theta$ with respect to a scalar-valued loss function $\mathcal{L}$. For an autoencoder with an encoder represented by $\phi(\cdot)$ and decoder represented by $\psi(\cdot)$, the loss penalizes deviation between the network input $x$ and the reconstructed input $\tilde{x}$, obtained by running the encoding $\hat{x} = \phi(x)$ through the decoder network ($\tilde{x} = \psi(\hat{x})$). Hence, the loss for a deep autoencoder is often given by:

$$\mathcal{L} = \|x - \tilde{x}\|_2^2 + \lambda \Omega(\theta),$$ (2.4)

where $\Omega(\cdot)$ is a function that regularizes the network weights and $\lambda$ is a scalar parameter that trades off between the reconstruction and regularization objectives.

While convolutional networks are most often applied to learning problems involving image data, one-dimensional convolutional networks have also been used to extract temporal features from sequence data [37–39]. Furthermore, three-dimensional convolutional architectures have been shown to be capable of learning spatiotemporal features from video data, where the first two filter dimensions are used for extracting spatial features and the third dimension is used to extract temporal features across successive video frames [40–42].
2.4.2 Network Architecture

This chapter demonstrates that three-dimensional convolutional architectures can also be applied to CFD solutions, where the first three dimensions of the input correspond to spatial dimensions, and the final input dimension corresponds to different flow quantities at each point in space (rather than color intensities). For local compression, the 320 degrees of freedom per mesh element are reduced using autoencoders. This step is enabled by the observation that the number of available training examples is equal to the number of time snapshots multiplied by the total number of mesh elements. For fine spatial discretizations, the number of elements is large, which implies that there may exist a large amount of training data, even for a single simulation with $\sim 100$ time snapshots.

Because standard convolutional neural network architectures were devised for image data, they require the solution points to be equispaced within each element. However, the simulation uses nonuniform Gauss-Legendre points in order to minimize aliasing errors when projecting the nonlinear flux into the polynomial space of the solution. Thus, before applying the autoencoder, the solution is transformed by interpolating it from the Gauss-Legendre points to a $4 \times 4 \times 4$ grid of equispaced points. Since solution values at both sets of points define the same polynomial interpolant, the solution inside of each element is unchanged by this transformation. The size of the network inputs is therefore $4 \times 4 \times 4 \times 5$, where the first three dimensions correspond to spatial dimensions and the last dimension corresponds to the flow quantities. This approach does not account for the fact that the mesh elements can vary in size, and hence the spacing between points is not uniform across all training examples.

Each layer in the autoencoder consists of a convolutional transformation with ResNet skip connections. Between convolutional layers, batch normalization [43] and ReLU activations are applied. The encoder network consists of 5 layers, where each layer contains parameters associated with a set of three-dimensional convolutional filters $\theta_i \in \mathbb{R}^{(3 \times 3 \times 3 \times f_i)}$, and $f_i$ denotes the number of filters contained in layer $i$. There are 512, 128, 32, 16, and 8 filters in the five layers of the encoder. The dimensionality of the transformed input is initially large after the early layers to allow for adequate feature extraction, and is subsequently decreased gradually through the remaining
hidden layers of the encoder. In the first four layers of the encoder, the convolutions are performed with a stride of one, meaning that the first three dimensions of the layer output are equal in size to the first three dimensions of the layer input. In the final layer of the encoder, the convolutions are performed with a stride of two, meaning that the first three dimensions of the layer output are half the size of the first three dimensions of its input.

The output of the final convolutional layer is reshaped into a vector and subsequently mapped to encoding $\hat{x}_i^t$ with an affine transformation. The dimensionality of the local encoded state (i.e. the code) is set to be $c_{\text{dim}} = 24$. An affine transformation and reshaping operation are then used to create an input to the decoder that is the same size as the output of the encoder. The decoder network is constructed to invert all operations performed by the encoder in order to obtain reconstructed solutions. Figure 2.6 provides an illustration of the network architecture.

The training and validation sets for the autoencoder consist of local flow states contained within 100 time snapshots $x_t$ equispaced throughout the time domain. Stochastic gradient descent with the Adam optimizer [44] is used to solve the minimization problem described by Eq. (2.4), where $L_2$ regularization is applied to the network weights:

$$\Omega(\theta) = \|\theta\|^2_2.$$  \hspace{1cm} (2.5)

When an increase in validation loss is observed across training epochs, the learning rate is cut in half. Training is terminated once the loss on the validation set fails to decrease across three training epochs, which typically occurs once the learning rate has been decayed many times.
The dimensionality of the encoded states

$$\hat{x}_t = [x_t^1, \ldots, x_t^{N_{el}}]^\top$$

is $c_{\text{dim}} \cdot N_{el}$, where $N_{el}$ is the number of mesh elements. While this is significantly smaller than the dimensionality of the global states $x_t$, it will still be large if the number of elements $N_{el}$ is large, as is the case for fine spatial discretizations. To address this, the following section describes how principal component analysis (PCA) can be applied to reduce the dimensionality of the encoded states $\hat{x}_t$.

### 2.5 Global Compression with PCA

Dimensionality reduction is next applied to the encoded states $\hat{x}_t$. In this case, the number of available training examples is much smaller than in the case of local compression. This arises from the fact that local compression employs a training set with a size $\mathcal{O}(T \cdot N_{el})$. The size of the global compression training set is simply $\mathcal{O}(T)$. Thus, for global compression, principal component analysis is used, which is a dimensionality reduction method that does not rely upon access to a large amount of training data. Furthermore, a piecewise PCA approach is used due to the fact that it may be computationally costly to compute principal components of the encoded state vectors.

Define the number of time steps for PCA to be $T$ and let the associated encoded states be $\hat{x}_{1:T}$. Due to the prohibitive computational cost associated with performing PCA on the full encoded states, each $\hat{x}_t$ is decomposed into $n_x$ components $\hat{x}_{t,i}$, $i = 1, \ldots, n_x$ such that $\hat{x}_t = [\hat{x}_{t,1}^\top, \ldots, \hat{x}_{t,n_x}^\top]^\top$. Next, the singular value decomposition (SVD) is computed:

$$\begin{bmatrix}
(\hat{x}_{1,i} - x_i) & (\hat{x}_{2,i} - x_i) & \cdots & (\hat{x}_{T,i} - x_i)
\end{bmatrix} = U_i \Sigma_i V_i^\top, \quad i = 1, \ldots, n_x,$$  

(2.7)
where
\[ \bar{x}_i = \frac{1}{T} \sum_{t=1}^{T} \hat{x}_{t,i}, \quad i = 1, \ldots, n \]
(2.8)

denote the sample means and \( \Sigma_i = \text{diag}(\sigma_{t,i})_{t=1}^{T} \), with
\[ \sigma_{1,i} \geq \cdots \sigma_{T,i} \geq 0. \]
(2.9)

The columns of \( U_i \) can be viewed as a set of orthonormal basis vectors. Let \( p_u(x) \) represent the orthogonal projection of vector \( x \) onto vector \( u \); it can be shown that \( u_{1,i} \), the first column of \( U_i \), is the vector that satisfies
\[ u_{1,i} = \arg \min_{u: u^T u = 1} \sum_{t=1}^{T} \| (\hat{x}_{t,i} - \bar{x}_i) - p_u(\hat{x}_{t,i} - \bar{x}_i) \|^2. \]
(2.10)

Thus, \( u_{1,i} \) is the direction onto which the data can be projected with the lowest approximation error. An alternative interpretation is that \( u_{1,i} \) represents the direction that best captures the variance present within the training data. The vector \( u_{2,i} \) satisfies
\[ u_{2,i} = \arg \min_{u: u^T u = 1, u^T u_{1,i} = 0} \sum_{t=1}^{T} \| (\hat{x}_{t,i} - \bar{x}_i) - p_u(\hat{x}_{t,i} - \bar{x}_i) \|^2, \]
(2.11)
meaning that it does best job of capturing the variance in the training data among all unit vectors orthogonal to \( u_{1,i} \). In general, it can be shown that the first \( k \) vectors in \( U_i \) represent the optimal \( k \)-dimensional orthonormal basis onto which the data can be projected, as measured by the square error between the original and projected data.

The amount of variance captured by vector \( u_{t,i} \) is summarized by its associated singular value \( \sigma_{t,i} \). Hence, the quantity
\[ e_k = \frac{\sum_{t=1}^{k} \sigma_{t,i}}{\sum_{t=1}^{T} \sigma_{t,i}} \]
(2.12)
serves as a proxy for how much of the variance is captured by the first \( k \) basis vectors, and is often referred to as the energy contained within the subspace spanned by \( u_{1,i}, \ldots, u_{k,i} \). For many datasets, a relatively small number of basis vectors may
CHAPTER 2. MULTI-STAGE COMPRESSION

capture a large amount (e.g. 95% or 99%) of the variance in the training data. Under these circumstances it is possible to perform truncation, ignoring basis vectors $u_{k+1,i}, \ldots, u_{T,i}$, and thereby obtaining a low-dimensional basis into which the training data can be projected.

Principal component analysis truncates the vectors in $U_i$ to obtain the basis matrix

$$\tilde{U}_i = \begin{bmatrix} u_{1,i} & u_{2,i} & \cdots & u_{N_{\text{trunc},i},i} \end{bmatrix},$$

(2.13)

where $N_{\text{trunc},i}$ can, for example, be set according to a criterion based on the energy defined in Eq. (2.12). The piecewise PCA procedure then proceeds to calculate the singular value decomposition of the projection of the encoded states:

$$\begin{bmatrix} \text{diag} (\tilde{U}_i) \end{bmatrix}^\top \begin{bmatrix} (\hat{x}_1 - \bar{x}) & (\hat{x}_2 - \bar{x}) & \cdots & (\hat{x}_T - \bar{x}) \end{bmatrix} = U\Sigma V^\top,$$

(2.14)

with

$$\bar{x} = \frac{1}{T} \sum_{t=1}^{T} \hat{x}_t.$$  

(2.15)

Calculating the SVD of a matrix containing the encoded states $\hat{x}$ can be intractable due to the large number of mesh elements, but the SVD of a matrix containing the (lower-dimensional) projections of the encoded states can potentially be obtained at a much lower computational cost.

Next, the global basis matrix

$$\tilde{U} = \text{diag} (\tilde{U}_i)^\top \begin{bmatrix} u_1 & u_2 & \cdots & u_{N_{\text{trunc}}} \end{bmatrix},$$

(2.16)

can be found. If significant truncation is performed, then $N_{\text{trunc}}$ will be smaller than $\sum_i N_{\text{trunc},i}$, which will in turn be smaller than the dimensionality of the encoded states.
Finally, the reduced state can be defined:
\[ z_t = \hat{U}^t (\phi_{\text{global}}(x_t) - \bar{x}), \quad t = 1, \ldots, T, \] (2.17)
where
\[ \phi_{\text{global}}(x_t) = \left[ \phi(x_t^1)^T, \ldots, \phi(x_t^{N_{\text{el}}})^T \right]^T. \] (2.18)

The inverse mapping \( p(\cdot) \) is then given by:
\[ p(z_t) = \psi_{\text{global}} \left( \hat{U} z_t + \bar{x} \right), \quad t = 1, \ldots, T, \] (2.19)
where
\[ \psi_{\text{global}}(\hat{x}_t) = \left[ \psi(\hat{x}_t^1)^T, \ldots, \psi(\hat{x}_t^{N_{\text{el}}})^T \right]^T. \] (2.20)

To allow for tractable computation of the singular value decompositions, the encoded states \( \hat{x}_t \) associated with the half-cylinder data are divided into \( n_{\hat{x}} = 38 \) components. Each of the components contains data from 1068 cells, which, given the 24-dimensional encoding associated with each cell, yields vectors \( \hat{x}_{t,i} \) with 25,632 entries. The principal components are truncated such that \( N_{\text{trunc},i} = 500, \ i = 1, \ldots, 38 \). Furthermore, two values of \( N_{\text{trunc}}, N_{\text{trunc}} = 100 \) and \( N_{\text{trunc}} = 500 \), were tested. The level of truncation corresponds to the dimensionality of the reduced state \( z_t \in \mathbb{R}^{N_z} \), i.e. \( N_z = N_{\text{trunc}} \). A summary of the experiments in modeling the dynamics of these reduced states can be found in the following section.

### 2.6 Dynamics Learning

This section describes dynamics learning. In particular, it is assumed that the sequence of reduced states \( z_{1:T} \) can be recovered from the discrete-time dynamical system:
\[ z_{t+1} = F_{\text{true}}(z_t), \quad t = 1, \ldots, T - 1 \] (2.21)
for some unknown function \( F_{\text{true}}(\cdot) \). If such a function exists, then the entire sequence of reduced states can be recovered by specifying only the initial state \( z_1 \). Because the
true dynamics model is unknown, it must be approximated by a learned model \( F(\cdot) \). Once the model is learned, the reduced states can be approximated as \( \tilde{z}_{1:T} \), where \( \tilde{z}_1 =\tilde{z}_1 \) and \( \tilde{z}_{2:T} \) satisfy \( \tilde{z}_{t+1} = F(\tilde{z}_t) \). Subsequently, the sequence of global states \( x_{1:T} \) can be approximated as \( \tilde{x}_{1:T} \), where \( \tilde{x}_t = p(\tilde{z}_t) \).

In this regression problem, the goal is to learn a model \( F(\cdot) \) that can accurately propagate the reduced states forward in time. The models can be trained on a set of input/target pairs, where the inputs are reduced states at one time step and the target values are the desired model outputs (i.e. predictions for the reduced states at the next time step). The data can be divided into training and validation sets, where the training set is used to train the model, and the validation set evaluates the ability of the model to generalize to unseen data.

Because the total number of time instances, \( T \), is assumed to be small, the amount of available training data will be limited, and thus the focus here is restricted to regression models that tend to perform well in limited data settings. A wide range of regression techniques were investigated for this purpose, including support vector regression [45], random forests [46], boosted decision trees [47], \( k \)-nearest neighbors [48], the vectorial kernel orthogonal greedy algorithm [49], sparse identification of nonlinear dynamics [50], and dynamic mode decomposition [51]. An overview of these techniques can be found in an associated journal article by Carlberg et al. [13]. For conciseness, the focus in this thesis is restricted to the vectorial kernel orthogonal greedy algorithm, which attained the best performance of the studied methods. A description of this method can be found in the following subsection.

### 2.6.1 Vectorial Kernel Orthogonal Greedy Algorithm

The vectorial kernel orthogonal greedy algorithm (VKOGA) [49, 52] constructs a regression model of the form:

\[
F(z) = \sum_{i=1}^{n_{VKOGA}} \alpha_i K(z_i, z),
\]  

(2.22)
where \( K : \mathbb{R}^{N_x \times N_x} \rightarrow \mathbb{R} \) denotes a kernel function, \( z_i \in \mathbb{R}^{N_x}, \ i = 1, \ldots, n_{\text{VKOGA}} \) denote the kernel centers, and \( \alpha_i \in \mathbb{R}^{N_x}, \ i = 1, \ldots, n_{\text{VKOGA}} \) denote the vector-valued basis functions.

VKOGA first computes kernel functions \( K(z, \cdot) \) by a greedy algorithm. The greedy algorithm selects kernel centers from \( C = \{ z_1, \ldots, z_{n_{\text{train}}} \} \). Initially, let \( C_0 = \emptyset \). At stage \( k \), a kernel center \( z_k \) is chosen such that:

\[
    z_k = \arg \max_{z \in C \setminus C_{k-1}} \left| \langle F, \xi_{sz}^{k-1} \rangle \right|, \tag{2.23}
\]

where \( \langle \cdot, \cdot \rangle \) denotes the inner product and \( \xi_{sz}^{k-1} \) represents the orthogonal remainder of \( K(z, \cdot) \) with respect to the space spanned by functions \( K(z_1, \cdot), \ldots, K(z_{k-1}, \cdot) \). Then \( C_k = C_{k-1} \cup \{ z_k \} \). After the kernel centers have been computed, the basis functions \( \alpha_i, i = 1, \ldots, n_{\text{VKOGA}} \) are determined through a least-squares procedure.

### 2.6.2 Approximate Reduced-State Dynamics

In the numerical experiments, the VKOGA kernel functions were set to be Gaussian radial basis functions:

\[
    K(z_i, z_j) = \exp(-\gamma \| z_i - z_j \|_2^2), \tag{2.24}
\]

where \( \gamma \) is a scalar parameter. The number of kernel functions, \( n_{\text{VKOGA}} \), is a hyperparameter that needs to be specified, and can be selected by evaluating performance on a validation set. The left figure in Fig. 2.7 shows the relative mean-squared error on the training and validation sets plotted against the number of kernel functions.

First, note that the training and validation errors are extremely similar; this suggests that the training data is representative of the testing data, and that no overfitting has occurred. Furthermore, it is clear that prediction accuracy improves with the number of kernel functions. However, the cost of training and evaluating the dynamics model scales linearly with \( n_{\text{VKOGA}} \). Additionally, compression levels are reduced as more kernel centers are added, because the reduced states must be stored in order to define the corresponding kernel functions. Given these competing objectives of accuracy and efficiency, it was ultimately decided to set \( n_{\text{VKOGA}} = 200 \).
Once calibrated and trained, the learned dynamics model can be used to simulate the evolution of the reduced states across the entire time domain. The right figure in Fig. 2.7 shows the relative mean-squared error in predictions over time for two levels of truncation: $N_z = 100$ and $N_z = 500$. The learned model is shown to capture the reduced state dynamics well, with a relative error of less than 5% and 0.7% across all time steps for $N_z = 100$ and $N_z = 500$, respectively.

Given that increasing the dimensionality of the reduced state reduces the relative error by nearly an order of magnitude, the reduced state is set to be 500-dimensional in all subsequent experiments. Relative to the $\approx 13 \times 10^6$ degrees of freedom in the global states, this represents a compression ratio of 26000 : 1. An additional level of compression is obtained through the VKOGA dynamics model, which requires the storage of only 200 reduced states from throughout the time domain to serve as kernel centers. Combining the savings achieved through the spatial and temporal compression procedures, a compression level of $\approx 1.1 \times 10^6 : 1$ is obtained. The next section studies the accuracy of the approximate global states obtained from the compressed representations.
2.7 Compression Results

The results in Section 2.6.2 present the prediction error over time in the reduced state. Of greater interest is the prediction error relative to the global state $x_t$ once the reduced state values are passed through the mapping $p(\cdot)$ to obtain the approximate global state $\tilde{x}_t$. The prediction error across all time instances for the approximated solutions can be found in Fig. 2.8. To put these results in context, they are compared against the error obtained by passing each global state through the encoder and decoder without performing the PCA and dynamics learning procedures. It is apparent that very little error is introduced by compressing and reconstructing the global states, with a relative error of less than 0.13% across all time instances. More critically, the relative error for the approximated states is nearly identical to the error from the autoencoder reconstructions, meaning that very little error is introduced by the PCA and dynamics-learning steps.

Beyond solely considering the global error in reconstructions, it is also of interest whether local properties of the fluid flow are preserved during the various stages of dimensionality reduction, dynamics propagation, and reconstruction. One manner of determining whether flow properties are preserved is to consider the aggregate lift
and drag forces that act on the surface of the half-cylinder over time. By extracting
pressure values on the surface of the cylinder, the distribution of forces acting on the
cylinder in the downstream and cross-stream directions can be resolved.

Integrating over the surface of the half-cylinder, the lift and drag forces are obtained.
The lift coefficient is defined as:

\[ C_L = \frac{L}{\frac{1}{2}\rho_\infty v_\infty^2 S}, \]  

(2.25)

where \( L \) is the lift force, \( \rho_\infty \) is the free-stream density, \( v_\infty \) is the free-stream velocity, and \( S \) is the surface area of the half-cylinder. Likewise, the drag coefficient is defined as:

\[ C_D = \frac{D}{\frac{1}{2}\rho_\infty v_\infty^2 S}, \]  

(2.26)

where \( D \) is the drag force.

The left column of Fig. 2.9 reports these lift and drag coefficients plotted across all time instances for the CFD solutions \( \mathbf{x}_t \), autoencoder reconstructions \( \hat{\mathbf{x}}_t \), and approximated solutions \( \tilde{\mathbf{x}}_t \). The right column of Fig. 2.9 reports the error in lift and drag coefficients associated with the reconstructed solutions generated by the autoencoder and the approximated solutions. The absolute error is provided for lift values rather than the relative error due to some lift coefficient values being close to zero. From these results, it is clear that the error in lift and drag introduced by encoding and decoding solutions with the autoencoder is quite low: the absolute error is less than \( 1.2 \times 10^{-3} \) across all time steps for lift values and the relative error is less than 0.1% for drag values. Furthermore, the principal component analysis and dynamics-learning procedures introduce very little additional error in these quantities beyond what is introduced by the autoencoder.

Visualizations of the reconstructed solutions can also lend qualitative insight into the performance of the proposed method. One key question is whether the approximated solutions retain the important vortical structures that are present within the CFD solutions. This question can be answered by considering the velocity gradient tensor at each point in space. In index notation, the velocity gradient tensor
Figure 2.9: Lift and drag coefficients of the cylinder across all time instances. The left column shows lift and drag predictions, while the right column presents the error in lift and drag predictions. Absolute error is used rather than relative error to avoid numerical issues when lift values are close to zero.

is given by:

\[ D_{ij} = \frac{\partial v_i}{\partial x_j}, \]  

(2.27)

where \( v_i \) represents the local velocity along the spatial dimension \( x_i \). The velocity gradient tensor can be decomposed into a symmetric and skew-symmetric part \( D_{ij} = S_{ij} + \Omega_{ij} \) where \( S_{ij} \) is the rate-of-strain tensor given by

\[ S_{ij} = \frac{1}{2} \left( \frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i} \right) \]  

(2.28)
and $\Omega_{ij}$ is the vorticity tensor given by

$$
\Omega_{ij} = \frac{1}{2} \left( \frac{\partial v_i}{\partial x_j} - \frac{\partial v_j}{\partial x_i} \right).
$$

The characteristic equation for $D$ can be written as

$$
\lambda^3 + P\lambda^2 + Q\lambda + R = 0 \quad (2.30)
$$

where $P$, $Q$, and $R$ are the first, second, and third invariants of the velocity gradient tensor. In particular, the second invariant $Q$ can be determined through:

$$
Q = \frac{1}{2} \left( \text{tr}(D)^2 - \text{tr}(D^2) \right) = \frac{1}{2} \left( \|\Omega\|_F^2 - \|S\|_F^2 \right). \quad (2.31)
$$

A criterion known as the $Q$-criterion defines a vortex as a connected fluid region region where $Q$ is positive. By considering Eq. (2.31), it is clear that the $Q$-criterion defines vortices as regions where the vorticity magnitude exceeds the magnitude of rate-of-strain [53].

Figure 2.10 compares instantaneous iso-surfaces of $Q$ colored by velocity magnitude for CFD solutions, autoencoder reconstructions, and approximated solutions. Iso-surfaces are only generated for positive $Q$-values in order to identify the presence of vortices. The figure shows that for all considered cases, the autoencoder reconstruction retains all of the key vortical structures of the CFD solution, albeit with some noise in the iso-surfaces. There appears to be no distinguishable difference between the autoencoder reconstructions and the approximated solutions. This further supports the assertion that the principal component analysis and dynamics-learning procedures introduce very little additional error.

### 2.8 Discussion

This chapter introduced a methodology for compressing CFD data with fine discretizations. The first stage of the methodology, hierarchical dimensionality reduction, comprises a novel two-step technique tailored for high-order discretizations. In the
first step, autoencoders are applied for local compression; this step computes a 24-
dimensional nonlinear embedding of the 320 degrees of freedom within each element
of the mesh, without incurring significant errors. In the second dimensionality reduc-
tion step, principal component analysis was applied to compress the global vector of
encodings. Results showed that this second step was able to reduce the number of
global degrees of freedom from $\sim 10^6$ to only 500, constituting a compression ratio of
26000 : 1, while retaining very high levels of accuracy.

The second stage of the methodology, dynamics learning, applied methods from
machine learning to learn the discrete-time dynamics of the low-dimensional reduced
state. The vectorial kernel orthogonal greedy algorithm achieved the best performance
among all regression methods, and was shown to approximate the dynamics of the
reduced state with high accuracy. Ultimately, applying the proposed methodology with VKOGA and a 500-dimensional reduced state satisfied the objective of this work, as it enabled the original CFD data to be reconstructed with extremely high levels of accuracy, yielding low errors in the state, lift, and drag, as well as solution fields that match well visually (see Figs. 2.8 to 2.10).

In this chapter, the compression and dynamics-learning stages were performed disjointly, which was necessary due to the high-dimensionality of the global flow state. However, on smaller-scale problems these tasks can be performed simultaneously, allowing for the discovery of reduced representations for which the dynamics can be learned more easily. The next chapter introduces such a joint learning procedure, which is applied to the task of generative modeling.
Chapter 3

Generative Modeling of Dynamical Systems

The previous chapter focused on compressing data generated by computational fluid dynamics simulations. This chapter focuses on efficient simulation, presenting a method for learning models that can perform inexpensive parameterized simulations of dynamical systems. The chapter begins by motivating the use of data-driven methods for enabling efficient simulation. It then provides an overview of generative modeling, and introduces the proposed method, known as parameter-conditioned sequential generative modeling.

3.1 Background

CFD simulations (and other simulations that involve physics-based models) often require fine spatial and temporal discretizations to achieve high levels of fidelity. These fine discretizations can make the corresponding simulations expensive to run due to the large number of degrees of freedom that must be modeled. In applications such as design, which can require running many simulations across multiple design configurations, this high computational cost can be prohibitive. Instead of running expensive simulations across the entire design space, a technique known as surrogate modeling is often used [54–56], wherein models are trained to approximate the mapping
between design parameters and a scalar-valued objective that evaluates design quality. Surrogate models can be constructed from a limited amount of high-fidelity simulation data using techniques such as polynomial or Gaussian process regression, and allow for efficient evaluation of design objectives at new design points. However, while surrogate modeling approaches can enable faster analysis and design, they do not reduce the computational cost of running high-fidelity simulations.

In place of surrogate models, which only attempt to model the mapping from design parameters to high-level objectives, a great deal of recent attention has focused on using learning-based techniques to facilitate more efficient high-fidelity simulation. Similar to the compression approach outlined in Chapter 2, many proposed techniques seek a low-dimensional, reduced representation of the modeled system that can be simulated more efficiently than the full system dynamics. This generally requires first learning a mapping from the high-dimensional dynamical system states to reduced states, and subsequently devising a method for integrating the reduced states forward in time.

Projection-based approaches [57, 58] learn a mapping from high-dimensional states to a low-dimensional subspace, and then determine the reduced state dynamics through a projection onto known governing equations. The learned mappings in these approaches are generally limited to linear embeddings, although some recent work has explored the use of nonlinear embeddings [59, 60]. One drawback of projection-based approaches is that they must be integrated into existing simulation software, which can be challenging and time intensive [61].

An alternative approach for determining the reduced-state dynamics is to learn the dynamics directly from data, as outlined in Section 2.6. Such a data-driven approach requires no integration with existing simulation software and no knowledge of the system governing equations. Some recent works have applied two-stage (dimensionality reduction and dynamics learning) deep learning procedures to the task of modeling fluid flows [62, 63]. This chapter proposes a new method, grounded in variational inference, for learning-based modeling of dynamical systems, in which the state mapping and reduced state dynamics are learned jointly. In line with some recent work [61, 63], the derived learning procedure allows the trained model to perform parameterized
simulations, thus enabling the efficient simulation of modeled systems at a wide range of conditions.

To motivate the proposed method, the discussion now proceeds by reviewing relevant background material related to generative modeling.

### 3.2 Generative Modeling

Generative modeling refers to modeling the process by which data is generated in the real world [64]. As such, physical models like the Navier–Stokes equations can be viewed as a form of generative model. In the absence of known governing equations, learned generative models can attempt to model real-world systems by approximating the distribution over data, $p_{\text{data}}(x)$. Several powerful generative modeling paradigms have emerged in recent years, including variational autoencoders [33], generative adversarial networks [65], and flow-based models [66, 67]. The following section provides an introduction to variational autoencoders.

#### 3.2.1 Variational Autoencoders

The goal in generative modeling is to learn $p_{\theta}(x)$, a distribution over (possibly high-dimensional) data $x \in \mathcal{D}$, where $\theta$ represents the parameters that govern the learned distribution. An accurate learned distribution will be close to the true data distribution, $p_{\text{data}}(x)$, where the difference between the distributions can be measured through similarity measures such as the KL-divergence:

$$D_{\text{KL}}(p_{\text{data}}(x) \parallel p_{\theta}(x)) = \mathbb{E}_{x \sim p_{\text{data}}(x)} \left[ \log \frac{p_{\text{data}}(x)}{p_{\theta}(x)} \right]. \quad (3.1)$$

It can be shown that minimizing the KL-divergence between these distributions is equivalent to maximizing the log-likelihood the model assigns to the data [64]:

$$\log p_{\theta}(\mathcal{D}) = \sum_{x \in \mathcal{D}} \log p_{\theta}(x). \quad (3.2)$$

Success in generative modeling relies upon a model that (1) can be optimized
efficiently (i.e. the gradients $\nabla_{\theta} \log p_{\theta}(x)$ are known and easy to calculate) and (2) is sufficiently expressive in order to accurately approximate the true data distribution. For example, a joint Gaussian distribution over pixel color intensities could be used to represent a distribution over image data, but, if the true data distribution is non-Gaussian, then such a model would be incapable of representing $p_{\text{data}}(x)$ even under the optimal parameter settings.

The expressiveness of modeled distributions can be enhanced by introducing latent variables, denoted by $z$. Latent variables are unobserved, but are assumed to summarize high-level information about the data, and are often assumed to be (much) lower-dimensional than the data. As an example, in image data the latent variable $z$ may be taken to represent the class of an image (e.g. dog or human or car), and the data $x$ would then represent a particular instantiation of that class. Under the presence of latent variables, the modeled distribution becomes:

\[
p_{\theta}(x) = \int p_{\theta}(x, z) dz
\]

\[
= \int p_{\theta}(x \mid z) p_{\theta}(z) dz.
\]  

When the distribution is modeled in this manner then, even if $p_{\theta}(x \mid z)$ and $p_{\theta}(z)$ are Gaussian distributions, the resulting marginal distribution $p_{\theta}(x)$ can be quite complex.

Unfortunately, calculating (and in turn optimizing) the likelihood as defined in Eq. (3.3) is generally intractable due to the difficulty of evaluating the integral over $z$. Additionally, finding the posterior over $z$ given $x$, defined as:

\[
p_{\theta}(z \mid x) = \frac{p_{\theta}(x \mid z) p_{\theta}(z)}{p_{\theta}(x)}
\]  

is also intractable due to the presence of $p_{\theta}(x)$ in the denominator of the expression. In place of this intractable posterior distribution, variational inference [68] introduces an approximate posterior distribution $q_{\phi}(z \mid x)$, which is defined by a set of parameters $\phi$. Multiplying and dividing the likelihood expression by this approximate posterior
distribution yields:

$$\log p_\theta(x) = \log \int p_\theta(x \mid z) p_\theta(z) \frac{q_\phi(z \mid x)}{q_\phi(z \mid x)} \, dz$$

$$= \log \mathbb{E}_{z \sim q_\phi(x \mid z)} \left[ \frac{p_\theta(x \mid z) p_\theta(z)}{q_\phi(z \mid x)} \right].$$

(3.5)

Through Jensen’s inequality the following expression can be derived:

$$\log p_\theta(x) \geq \mathbb{E}_{z \sim q_\phi(x \mid z)} \left[ \log p_\theta(x \mid z) - \log \frac{q_\phi(z \mid x)}{p_\theta(z)} \right]$$

$$\geq \mathbb{E}_{z \sim q_\phi(x \mid z)} \log p_\theta(x \mid z) - D_{\text{KL}}(q_\phi(z \mid x) \mid \mid p_\theta(z)).$$

(3.6)

The right-hand side of the above expression is often referred to as the evidence lower bound (ELBO), and serves as a lower bound on the log-likelihood objective. It can be shown that the ELBO will be equal to \( \log p_\theta(x) \) when:

$$D_{\text{KL}}(q_\phi(z \mid x) \mid \mid p_\theta(z)) = 0,$$

(3.7)

i.e. the approximate posterior distribution matches the true posterior distribution over \( z \). Thus, optimizing the evidence lower bound with respect to parameters \( \theta \) and \( \phi \) serves the dual purpose of (1) approximately maximizing the true objective \( \log p_\theta(x) \) and (2) driving the approximate posterior closer to the true posterior distribution [64].

The variational autoencoder (VAE) [33] presents a method for performing variational inference with neural network models. Much like standard autoencoders, VAEs consist of encoder and decoder components. The encoder (or recognition model) is denoted by \( q_\phi(z \mid x) \), and maps an input \( x \) to a distribution over the latent variable \( z \). The decoder (or generative model) is denoted by \( p_\theta(x \mid z) \), and maps samples from \( q_\phi(z \mid x) \) to a distribution over \( x \). The outputs of the neural networks are the distribution parameters, which are often assumed to be Gaussian with diagonal or constant covariance matrices.

Training a variational autoencoder consists of optimizing the set of parameters \( \theta \)
and $\phi$ in order to maximize the evidence lower bound presented in Eq. (3.6). Figure 3.1 illustrates the VAE training procedure. Training inputs are fed into the encoder, which outputs parameters to a distribution over the latent variable $z$. Subsequently, samples from that distribution are mapped through the decoder to obtain a distribution over $x$. The optimization objective balances maximizing the likelihood assigned to $x$ with minimizing the KL-divergence between the approximate posterior and the prior distribution. For simplicity, the prior is often assumed to be a standard Gaussian distribution. Evaluating the gradients with respect to the encoder parameters requires the use of a technique known as the reparameterization trick [69], which treats the encoder as a deterministic mapping subject to exogenous noise, and allows for the propagation of gradients through the sampling operation.

A VAE that has been trained to maximize the evidence lower bound can subsequently be employed in a generative fashion, where samples are drawn directly from the prior $p_{\theta}(z)$ and passed through the decoder. This process can result in the generation of new data that appears as if it was drawn from $p_{\text{data}}(x)$. The following example demonstrates how VAEs can be used to construct generative models for the MNIST dataset.

**Example: MNIST**

The MNIST dataset is a database of handwritten digits, ranging from zero to nine, which is often used as a benchmark task for classification algorithms. Training a variational autoencoder on the MNIST data can yield a generative model that is capable of generating new handwritten digits that appear as if they are part of the
training data. Furthermore, an examination of the latent representation associated with different digits can provide insight into relationships and similarities between the different classes contained in the training data.

In this experiment, a VAE was trained on the MNIST data using a fully connected neural network with ReLU activations and a two-dimensional latent state. Figure 3.2 shows the result of generating samples by sweeping over the dimensions of the latent space. From this image, it is apparent that different areas of the latent space are allocated to different digits. Furthermore, it is noteworthy that interpolation within the latent space leads to a smoother variation between digit classes than what would be obtained by interpolation in the space of images. As an example, consider the smooth transformation between ones, sevens, and nines as $z_1$ is varied along the bottom rows of the image.

### 3.2.2 Conditional Variational Autoencoders

The conditional variational autoencoder (CVAE) [70] extends the VAE by attempting to learn $p_{\theta}(x \mid c)$, a *conditional* model for the data distribution. Rather than modeling
CHAPTER 3. GENERATIVE MODELING OF DYNAMICAL SYSTEMS

the distribution over the entire dataset, a CVAE models data distributions conditioned on context parameters $c$. For example, with the MNIST dataset, $c$ could correspond to a particular class (zero through nine), and the CVAE would in turn model the data distributions associated with each class. Alternatively, $c$ could represent a segment of a given image, and the CVAE would then model the distribution over the remainder of the image. The graphical model for the CVAE can be found in Fig. 3.3b.

As with the variational autoencoder, a lower bound to the log-likelihood objective can be derived for the CVAE:

$$
\log p_{\theta}(x \mid c) \geq \mathbb{E}_{z \sim q_{\phi}(z \mid x, c)} \log p_{\theta}(x \mid z, c) - D_{KL}(q_{\phi}(z \mid x, c) \parallel p_{\theta}(z \mid c)).
$$

(3.8)

Optimizing for this objective requires relatively few modifications to the VAE training procedure outlined in Section 3.2.1. Most notably, the encoder and decoder networks must be modified to accept the context parameters as inputs. Furthermore, the prior over $z$ must be modeled as a conditional prior distribution, which is typically accomplished by introducing an auxiliary neural network that is trained to take $c$ as an input and output the prior distribution parameters.

3.2.3 Generative Modeling of Sequential Data

Generative models for sequential data attempt to learn the joint distribution $p_{\theta}(x_{1:T})$, where $x_1, \ldots, x_T$ represent data with temporal relationships; in this thesis it is assumed that each $x_t$ represents the state of a dynamical system at a given time step. In constructing sequential generative models, it is common to assume the existence

Figure 3.3: Graphical models for VAE, CVAE, and sequential generative model.
of (lower-dimensional) latent variables $z_{1:T}$, also referred to as latent states, that govern the time evolution of the modeled system. Furthermore, the dynamics of the latent states are often assumed to be Markovian, meaning that a latent state $z_t$ is conditionally independent of $z_{1:t-2}$ given $z_{t-1}$ (i.e. $z_{t-1}$ summarizes all relevant past information). The graphical model that defines the problem structure can be found in Fig. 3.3c; a model of this form is commonly referred to as a state space model (SSM) [71].

Analogous to the material presented in Section 3.2.1, the likelihood expression can be written in a form that explicitly accounts for the presence of the latent variables:

$$
p_\theta(x_{1:T}) = \int p_\theta(x_{1:T}, z_{1:T}) dz_{1:T}
= \int p_\theta(x_{1:T} | z_{1:T}) p_\theta(z_{1:T}) dz_{1:T}. \tag{3.9}
$$

Evaluating this integral is generally intractable due to the need to marginalize over the latent states. Likewise, evaluating the posterior distribution $p_\theta(z_{1:T} | x_{1:T})$ is generally intractable\(^1\), once again motivating the introduction of an approximate posterior distribution $q_\phi(z_{1:T} | x_{1:T})$.

Given the conditional independence assumptions encoded by the graphical model, the conditional distribution $p_\theta(x_{1:T} | z_{1:T})$ can be rewritten as:

$$
p_\theta(x_{1:T} | z_{1:T}) = p_\theta(x_1 | z_{1:T}) \prod_{t=2}^{T} p_\theta(x_t | x_{1:t-1}, z_{1:T})
= \prod_{t=1}^{T} p_\theta(x_t | z_t). \tag{3.10}
$$

\(^1\)There do exist two classes of SSMs for which posterior inference is tractable: (1) linear Gaussian SSMs and (2) discrete-state hidden Markov models [72].
Likewise, the distribution \( p_\theta(z_{1:T}) \) can be simplified to:

\[
p_\theta(z_{1:T}) = p_\theta(z_1) \prod_{t=2}^{T} p_\theta(z_t \mid z_{1:t-1})
\]

\[
= p_\theta(z_1) \prod_{t=2}^{T} p_\theta(z_t \mid z_{t-1}).
\]

Finally, the approximate posterior distribution can be expressed as:

\[
q_\phi(z_{1:T} \mid x_{1:T}) = q_\phi(z_1 \mid x_{1:T}) \prod_{t=2}^{T} q_\phi(z_t \mid z_{1:t-1}, x_{1:T})
\]

\[
= q_\phi(z_1 \mid x_{1:T}) \prod_{t=2}^{T} q_\phi(z_t \mid z_{t-1}, x_{t:T}).
\]

Given these simplified expressions, a lower bound to the log-likelihood, \( \log p_\theta(x_{1:T}) \), can be derived by following the procedures outlined in Section 3.2.1. This lower bound is given by:

\[
\log p_\theta(x_{1:T}) \geq \sum_{t=1}^{T} \left[ \mathbb{E}_{z_t \sim q_\phi} \log p_\theta(x_t \mid z_t) \right] + D_{KL} (q_\phi(z_1 \mid x_{1:T}) \mid\mid p_\theta(z_1))
\]

\[
+ \sum_{t=2}^{T} D_{KL} (q_\phi(z_t \mid z_{t-1}, x_{t:T}) \mid\mid p_\theta(z_t \mid z_{t-1})).
\]

The ELBO thus consists of \( T \) likelihood components, representing the likelihood of states \( x_t \) given latent states \( z_t \), and \( T \) KL-divergence components, representing the difference between the approximate posterior and (conditional) prior distributions. If the latent states are assumed to follow non-Markovian dynamics, then the ELBO becomes:

\[
\log p_\theta(x_{1:T}) \geq \sum_{t=1}^{T} \left[ \mathbb{E}_{z_t \sim q_\phi} \log p_\theta(x_t \mid z_t) \right] + D_{KL} (q_\phi(z_1 \mid x_{1:T}) \mid\mid p_\theta(z_1))
\]

\[
+ \sum_{t=2}^{T} D_{KL} (q_\phi(z_t \mid z_{1:t-2}, x_{t-1:T}) \mid\mid p_\theta(z_t \mid z_{1:t-2})),
\]
where the elements that differentiate Eq. (3.14) from Eq. (3.13) are highlighted in blue.

As with the variational autoencoder, the prior distribution $p_\theta(z_1)$ can be defined to be a standard Gaussian. Additionally, the parameters of the conditional prior distributions can be output by auxiliary neural networks. In contrast with the ELBOs for the VAE and CVAE, the ELBOs derived in this section contain distributions that require conditioning on sequences of variables. These distributions can be represented by standard feedforward neural networks, but such architectures are generally incapable of accommodating exceedingly long or variable-length sequences as inputs. This consideration motivates the use of recurrent neural networks, which are discussed in the following subsection.

**Recurrent Neural Networks**

Recurrent neural networks (RNNs) are neural networks designed for learning problems that involve sequences of data $x_1, \ldots, x_T$. RNNs maintain an internal (or hidden) state, $h_t$, which is updated recursively, and thereby summarizes information about all previous inputs to the model. The model output, the form of which will vary according to the particular learning task, can then be determined as a function of this internal state.

The general form for the recurrence that defines the update to the RNN internal state is:

$$h_t = f_{\text{RNN}}(h_{t-1}, x_t).$$

A standard choice for the update equation is to pass an affine transformation of $h_{t-1}$ and $x_t$ through a hyperbolic tangent function [10]:

$$f_{\text{RNN}}(h_{t-1}, x_t) = \tanh(W_h h_{t-1} + W_x x_t + b),$$

where $W_h$ and $W_x$ are weight matrices and $b$ is a vector. This equation can be used to iteratively update the RNN internal state, thereby offering the possibility of learning from arbitrarily long input sequences.

In practice, however, the large number of operations that must be chained together
in order perform the internal state updates can cause the parameter gradients to either vanish or explode during training. These gradient instabilities can lead to difficulties in optimizing standard RNN architectures. There exist two popular RNN variants, long short term memory (LSTM) [73] and gated recurrent unit (GRU) [74] networks, which address the vanishing gradient problem by introducing gating mechanisms that make it easier to learn long-term dependencies from input sequences. Additionally, exploding gradients can be addressed by clipping any gradient values that exceed a predefined threshold.

Thus, LSTM or GRU models can be used in place of feedforward architectures to represent the distributions in Eqs. (3.13) and (3.14) that are conditioned on sequences of variables. Building on the material presented above, the following section introduces parameter-conditioned generative modeling of sequential data.

### 3.2.4 Parameter-Conditioned Generative Modeling

This thesis proposes an extension to generative models for sequential data that is akin to the CVAE, wherein the latent state dynamics are assumed to be governed by parameters $\mathbf{c}$. In the context of dynamical systems, $\mathbf{c}$ may represent physical parameters that can affect system behavior. Examples of such parameters include the mass and length of a pendulum, or the Reynolds number of a fluid flow. Thus, a parameter-conditioned generative model attempts to model the density $p_\theta(\mathbf{x}_{1:T} \mid \mathbf{c})$.

The graphical model for this problem can be found in Fig. 3.4. Following the same
procedure as in Section 3.2.3, the evidence lower bound for the parameter-conditioned sequential generative model can be derived. For the case of non-Markovian latent state dynamics, this lower bound is found to be:

$$
\log p(\mathbf{x}_{1:T} | \mathbf{c}) \geq \sum_{t=1}^{T} \left[ \mathbb{E}_{\mathbf{z}_t \sim q_{\phi}} \log p_{\theta}(\mathbf{x}_t | \mathbf{z}_t) \right] \\
+ D_{KL} \left( q_{\phi}(\mathbf{z}_1 | \mathbf{x}_{1:T}, \mathbf{c}) \| p_{\theta}(\mathbf{z}_1 | \mathbf{c}) \right) \\
+ \sum_{t=2}^{T} D_{KL} \left( q_{\phi}(\mathbf{z}_t | \mathbf{z}_{1:t-1}, \mathbf{x}_{t:T}, \mathbf{c}) \| p_{\theta}(\mathbf{z}_t | \mathbf{z}_{1:t-1}, \mathbf{c}) \right).
$$

(3.17)

Hence, all prior and approximate posterior distributions must now condition on the parameters $\mathbf{c}$.

A trained parameter-conditioned generative model provides the opportunity to generate sequential data that evolves according to the prescribed parameters $\mathbf{c}$. In the context of dynamical systems, this could enable the efficient study of how such parameters affect a system’s behavior. Given a parameter space of interest, training data can be collected by simulating (or experimentally evaluating) a system at a handful of parameter values. Subsequently, a generative model trained on the collected data could potentially provide insight into the nature of the studied system throughout the remainder of the parameter space.

A generative model’s ability to accurately simulate a system throughout the parameter space is partly dependent on how well it learns to account for the manner in which the parameters affect the system’s dynamics. A description of a new objective that encourages the model to learn these relationships can be found in the next subsection.

**Mutual Information Objective**

Entropy, $H(\mathbf{y})$, serves as a measure of uncertainty about a random variable $\mathbf{y}$, and is defined as:

$$
H(\mathbf{y}) = - \mathbb{E}_{\mathbf{y} \sim p(\mathbf{y})} \log p(\mathbf{y}).
$$

(3.18)
Intuitively, a random variable will have high entropy if its probability mass (or density) is spread evenly across its support. The mutual information $I(w, y)$ between random variables $w$ and $y$ is defined as:

$$I(w, y) = H(w) - H(w | y).$$  \(3.19\)

Thus, $I(w, y)$ represents the reduction in uncertainty about $w$ given knowledge of $y$; if $w$ and $y$ are independent, then $I(w, y) = 0$.

To ensure that parameter-conditioned generative models can be used to perform simulations throughout the parameter space, high mutual information is desired between the prescribed parameters $c$ and the latent states $z_1:T$. This can be explicitly enforced by training the generative model to maximize a mutual information objective, $\sum_{t=1}^{T} I(c, z_t)$, in addition to the evidence lower bound from Eq. (3.17). Unfortunately, maximizing the mutual information objective directly is challenging because it requires access to the generally intractable posterior distributions $p_\theta(c | z_t)$.

In a procedure similar to variational inference, variational mutual information maximization [75–77] introduces a lower bound to the mutual information objective that can be tractably optimized. Let $r_\psi(c | z_t)$ represent an approximate posterior distribution over $c$, which is defined by parameters $\psi$. Then the following lower bound to the mutual information objective can be derived:

$$I(c, z_t) \geq H(c) + \mathbb{E}_{c \sim p(c), z_t \sim p_\theta} \log r_\psi(c | z_t).$$  \(3.20\)

It is assumed that $p(c)$ represents the distribution over parameter values within the dataset, and thus $H(c)$ is treated as a constant. Hence, raising the lower bound on the mutual information objective requires maximizing the log-likelihood assigned to $c$ by the approximate posterior distribution. This can be accomplished in practice by defining a reconstruction model with parameters $\psi$, which takes in values of $z_t$ sampled from the prior distributions and outputs parameters to a distribution over $c$. All model parameters, $\theta$, $\phi$, and $\psi$, can then be jointly optimized to maximize the
objective:

\[ \mathcal{L} = \mathcal{L}_{\text{ELBO}} + \lambda \mathcal{L}_{\text{Info}}, \tag{3.21} \]

where \( \mathcal{L}_{\text{ELBO}} \) represents the lower bound to the likelihood objective, \( \mathcal{L}_{\text{Info}} \) represents the lower bound to the mutual information objective, and \( \lambda \) is a scalar value that trades off between the objectives.

The next section presents experimental results that evaluate the performance of parameter-conditioned generative models on simulating fluid flows.

### 3.3 Experiments

This section applies parameter-conditioned generative modeling to the task of simulating fluid flows. Evaluations are performed in two problem domains: (1) two-dimensional airflow over a pair of counter-rotating cylinders and (2) three-dimensional airflow over a half-cylinder as described in Section 2.2. The next section studies the ability of learned models to simulate two-dimensional airflow, evaluating the accuracy of generated solutions at a range of flow conditions. A discussion of generative modeling performance for three-dimensional flow follows thereafter.

#### 3.3.1 Counter-Rotating Cylinders

This section considers the test case of airflow over counter-rotating cylinders, as depicted in Fig. 3.5. The cylinder surfaces are assumed to be separated by a distance of \( 3d \), where \( d \) is the cylinder diameter, and rotate at a fixed non-dimensional rotation speed given by:

\[ \Omega = \frac{\omega d}{2v_\infty}, \tag{3.22} \]

where \( \omega \) is the angular velocity of the cylinder and \( v_\infty \) is the velocity of the incident flow.

As with flow over a single bluff body, flow over a pair of cylinders has been shown to exhibit unsteady vortex shedding [78–80]. Experimental and numerical results have demonstrated that there exists a critical rotation speed, \( \Omega_{\text{crit}} \), above which the wake
instabilities disappear and the flow becomes steady, leaving a constant transverse load acting upon the cylinders [81–83]. The thesis of Chan [24] performs a wide array of numerical experiments that characterize the behavior of the double cylinder system as the cylinder separation, Reynolds number, rotation speed, and rotation direction are varied. The experiments in this thesis only allow for variation in rotation speed and Reynolds number (i.e. $c = [\Omega, \text{Re}]^\top$); the cylinder separation and rotation direction are assumed to be fixed.

All simulations are performed using the PyFR solver [29]. The computational domain is defined as $[-30, 50]$ in the stream-wise direction and $[-30, 30]$ in the cross-wise direction. The diameter of each cylinder is set to a value of one; the upper cylinder is centered at $(0, 2)$, while the lower cylinder is centered at $(0, -2)$. A spatially varying velocity is applied to the surface of each cylinder to model cylinder rotation. The full mesh, depicted in the left figure of Fig. 3.6, consists of 3148 unstructured, quadratically curved quadrilateral elements. To align with previous work, the Mach number is set to $M = 0.1$, which implies a free-stream velocity of $v_\infty \simeq 0.118$.

Training a model to generate flow quantities across the entire region shown in the left figure of Fig. 3.6 would be challenging due to the limitations that memory constraints impose upon the size of neural network inputs. Furthermore, the non-uniform spacing between the solution points in the CFD simulations also presents a challenge, because many of the most powerful neural network architectures are
CHAPTER 3. GENERATIVE MODELING OF DYNAMICAL SYSTEMS

Figure 3.6: Left: View of mesh used for simulation of counter-rotating cylinders over entire computational domain. Right: Zoomed-in view of the mesh in the vicinity of the cylinders. Generative models are trained to only simulate flow in the region with fine mesh resolution.

Figure 3.7: Format of inputs to the generative model for \( \text{Re} = 150 \) and \( \Omega = 0 \). Different physical quantities are treated as different channels in the input.

designed to handle image data, and thus assume uniform spacing between points.

To account for these challenges, the generative models are only trained to simulate fluid flow on a grid of equispaced points in the vicinity of the cylinders. Formatting the data in this manner allows for training on a set of image-like inputs of dimension \( 175 \times 350 \times 4 \), where the first two input dimensions correspond to spatial dimensions and the final input dimension corresponds to different flow quantities. The modeled region spans from \(-2\) to \(14\) in the stream-wise direction and \(-4\) to \(4\) in the cross-wise direction, which is selected to coincide with the area of fine mesh resolution depicted in the right image of Fig. 3.6. Figure 3.7 shows an example training input at Reynolds number \( \text{Re} = 150 \) and rotation speed \( \Omega = 0 \); the modeled flow quantities are density, \( x \)-velocity, \( y \)-velocity, and pressure. In all neural network inputs, flow quantities inside the cylinders are set to be free-stream values, and the network predictions for those
Figure 3.8: Illustration of parameter space for counter-rotating cylinder simulations. The points at which CFD simulations are run are represented by the black dots.

points are overwritten with free-stream values during training and evaluation.

Data Generation

Training data is generated at a range of flow conditions, corresponding to a variety of Reynolds numbers and rotation speeds. The set of Reynolds numbers considered is \( \text{Re} \in \{75, 85, 100, 125, 150, 200\} \). This set is designed to span the range of Reynolds numbers over which vortex shedding occurs and the flow can still be considered two-dimensional. Because viscous forces are inversely proportional to Reynolds number for a fixed flow velocity, a change in Reynolds number has a more pronounced effect on flow behavior at \( \text{Re} = 75 \) than it does at e.g. \( \text{Re} = 150 \), and hence a finer spacing between simulation points is used at lower Reynolds numbers. At each Reynolds number, simulations are run at rotation speeds \( \Omega \in \{0, 0.25, 0.5, \ldots, 2.5\} \). This range of rotation speeds is selected such that the training data contains information about the system behavior with no cylinder rotation, as well as the behavior of the flow as \( \Omega \) is increased, including rotation speeds in excess of \( \Omega_{\text{crit}} \). Figure 3.8 shows the points within the parameter space at which simulations are performed. The hope is that a generative model could be trained on CFD data from each point in the plot, and subsequently be capable of simulating fluid flow anywhere in the parameter domain.
CHAPTER 3. GENERATIVE MODELING OF DYNAMICAL SYSTEMS

By examining the training data, the effects of varying the rotation speed and Reynolds number can be observed. Figure 3.9 shows the time evolution of the lift coefficients for the top and bottom cylinders as a function of rotation speed at Reynolds number Re = 150. The lift coefficient is defined as:

$$C_L = \frac{L}{\frac{1}{2} \rho_\infty v_\infty^2 d}$$

Figure 3.9: Lift coefficients on top and bottom cylinder over time as a function of rotation speed.

where $L$ is the lift force, $\rho_\infty$ is the free-stream density, $v_\infty$ is the free-stream velocity,
and $d$ is the cylinder diameter. From the individual plots, two trends can be observed. First, the lift curves separate as the rotation speed is increased, with the mean lift coefficient becoming more positive on the top cylinder and more negative on the bottom cylinder. Second, the amplitude of oscillation in the lift curves is damped out at higher rotation speeds, with a constant transverse force acting on the cylinders at a rotation speed of $\Omega = 2$. From examining these plots, it is apparent that the critical rotation speed, at which the flow transitions from unsteady to steady flow, is somewhere in the interval $\Omega_{\text{crit}} \in [1.75, 2]$.

Figures 3.10 and 3.11 show the change in behavior of the cylinder system as the rotation speed and Reynolds number are varied. Figure 3.10 visualizes the $x$-velocity component of the training inputs for four different rotation speeds at a Reynolds number of $Re = 150$. It is clear from these images that the curved wake regions of low $x$-velocity begin to flatten out and emanate from different locations on the cylinders as the rotation speed increases. Figure 3.11 likewise illustrates the variation in flow behavior for four Reynolds numbers with a fixed rotation speed of $\Omega = 0$. It can be seen that the extent of the region of low $x$-velocity decreases with increasing Reynolds number. Although it cannot be observed from these static images, an increase in Reynolds number also corresponds to a decreased period of vortex shedding.
Training Details

This section provides details about training parameter-conditioned generative models on data from the counter-rotating cylinder system. All components of the generative model are jointly defined and trained in Tensorflow [84]. The parameters to the prior and approximate posterior distributions over \( z_1 \), denoted by \( p_\theta(z_1 \mid c) \) and \( q_\phi(z_1 \mid x_{1:T}, c) \), are output by feedforward neural networks with a single hidden layer containing 128 units. The prior and approximate posterior distributions over all remaining latent states, denoted by \( p_\theta(z_t \mid z_{1:t-1}, c) \) and \( q_\phi(z_t \mid z_{1:t-1}, x_{t:T}, c) \), are represented by two-layer feedforward neural networks with 256 and 128 units in each hidden layer. Long short term memory recurrent neural networks with 256 hidden units are used to summarize information about the (latent) state sequences upon which these distributions are conditioned. As explained in Section 3.2.3, defining the distributions in this manner implicitly assumes that the dynamics of the latent states are non-Markovian; treating the latent states this way was empirically found to yield better performance than assuming Markovian dynamics in the latent space.

As defined above, the networks used to represent the approximate posterior distributions may have an insufficient number of parameters to learn effectively from the very high-dimensional flow states. Therefore, a feature extractor network is introduced, which is trained to provide a mapping from the network inputs, \( x_t \in \mathbb{R}^{(175 \times 350 \times 4)} \), to a low-dimensional set of features. The approximate posterior distributions over the latent states are then conditioned upon these lower-dimensional features rather than the full flow state.

The feature extractor is comprised of a series of two-dimensional convolutional layers with ResNet skip connections and 256, 128, 64, 32, 16, 8, and 4 filters in each layer. Convolutions are performed with a stride of two in the first five layers, and a striding of one is employed in the remaining layers. An affine transformation maps the output of the convolutional layers to a set of features with the same dimensionality as the latent states; based on performance in numerical experiments, the latent states are defined to be four-dimensional in all studies involving the counter-rotating cylinder system. The decoder network, denoted by \( p_\theta(x_t \mid z_t) \), is constructed to invert all operations performed by the feature extractor, which allows for the mapping of latent
state samples back to the state space. Finally, the reconstruction network introduced in Section 3.2.4, denoted by $r_{\psi}(c \mid z_t)$, is defined to have a single 128-unit hidden layer. Rectified linear units are used as the activation functions across all feedforward layers in the model.

During training, all network parameters are concurrently optimized to maximize the objective defined by Eq. (3.21). The likelihood components of the evidence lower bound are optimized by minimizing the $L_1$ error between the true flow states, $x_t$, and the output of the decoder network. This is equivalent to assuming that $p_\theta(x_t \mid z_t)$ is a Laplacian distribution, where the mean of the distribution is output by the decoder network and the variance is constant. $L_1$ error is used rather than $L_2$ error because the $L_1$ norm is more sensitive to small errors, which encourages more accurate simulation of the subtle temporal variations in the modeled flow quantities. To allow for the ELBO and mutual information objectives to be of the same scale, a weight of $\lambda = 10^4$ is applied to the mutual information objective.

The generative model is trained on 20 distinct 32-step sequences from each flow condition present within the training data. Training is initialized with a learning rate of $6 \times 10^{-4}$, which is decayed by a factor of 0.75 whenever the loss fails to decrease on a validation set. Training terminates once the learning rate falls below a value of $1 \times 10^{-5}$. The Adam optimizer [44] is used to perform updates to the model parameters. In total, training takes approximately 30 hours with the neural network parameters divided across two NVIDIA K40 GPUs. While this represents a significant upfront cost, flow simulations generated by the trained model are found to be approximately 120 times faster than the CFD simulations used to generate the training data. This speedup suggests that an immense amount of time could be saved by using generative models in place of CFD solvers, assuming that the generative models prove capable of performing accurate simulations at a wide range of flow conditions. The next section presents numerical experiments that evaluate the learned model’s ability to generate flow solutions for the counter-rotating cylinder system.
Figure 3.12: Visual comparison of $x$-velocity over time at four flow conditions. Solutions generated by the CFD solver are shown in the top row, while flows sampled from the generative model are shown in the bottom row.
Results

The quality of generated solutions can be evaluated by first qualitatively studying whether the trained model is capable of generating realistic solutions for multiple settings of the prescribed parameter \( c \). Figure 3.12 presents a visual comparison of \( x \)-velocity values over time at four distinct flow conditions. For each flow condition, the top row of images corresponds to the ground-truth CFD solutions, while the bottom row contains solutions that were sampled from the generative model. To facilitate easier comparison, care was taken to approximately align the CFD and generated solutions in time such that both simulations show the system at similar moments within the vortex-shedding cycle. However, it should be noted that the initial conditions for the respective solutions are not the same, so discrepancies that arise should not necessarily be attributed to modeling errors by the generative model.

The results in Fig. 3.12 provide confidence that the generative model effectively captures the changes in flow behavior that occur as the Reynolds number and rotation speed are varied. Additionally, the generative model is shown to not just generate individual time snapshots that are visually similar to the corresponding fluid flow, it also generates solutions that appear to evolve in the same manner as the fluid flow. This temporal consistency is critical if the generative model is to be used to simulate the system for long time periods.

The most noteworthy takeaway from Fig. 3.12 is the visual similarity between the CFD and generated solutions at Reynolds number \( Re = 175 \) and rotation speed \( \Omega = 1 \). These flow conditions are not contained within the training data, and yet the generative model is still able to generate solutions that are qualitatively similar to the ground-truth fluid flow. Thus, these results imply that it may be possible to use the trained generative model for simulation at many flow conditions, including flow conditions not contained in the training data.

A more quantitative manner of assessing the accuracy of the generated solutions is to look at how the lift forces acting upon the cylinder vary with prescribed Reynolds number and rotation speed. As noted previously in the discussion about Fig. 3.9, the lift curves for the top and bottom cylinders tend to separate as the rotation speed is increased, with the mean lift on the top cylinder becoming more positive.
and the mean lift on the bottom cylinder becoming more negative. To examine if the generated solutions are capable of recreating this trend, Fig. 3.13 presents the mean lift coefficient for the top and bottom cylinders as a function of rotation speed for flow at Reynolds number $Re = 150$. It is clear from these plots that the mean lift coefficients calculated from the generated solutions are nearly identical to those determined from the CFD solutions.

In addition to trends in the mean lift coefficient, it can also be studied whether the generative model is capable of capturing variations in the frequency content of the lift curves. The time variation of the lift coefficient values presented in Fig. 3.9 is roughly sinusoidal, meaning that the corresponding power spectral density (PSD) should show a single peak at a dominant frequency denoted by $f_{\text{max}}$. Figure 3.14 shows how the dominant frequency in the lift signal for the top cylinder varies with rotation speed and Reynolds number in both the CFD and generated solutions. Results are only shown up to a rotation speed of $\Omega = 1.5$ because $\Omega = 1.75$ is beyond the critical rotation speed for lower Reynolds numbers. The curves associated with CFD solutions show an increase in the dominant frequency with Reynolds number and a slight decrease with rotation speed. The curves associated with the generated solutions appear to capture these trends quite well, with only minor discrepancies visible between the dominant frequencies identified from the CFD and generated solutions.

Lift is determined by integrating pressure values over the surface of an object. As
such, lift values can be largely insensitive to local errors in the pressure distribution, as long as errors at one location on the object surface are balanced out by errors elsewhere. Therefore, the next set of numerical experiments evaluates how accurately the generative model is able to simulate the time evolution of flow quantities at a single point in space. The selected point, represented by the black dot in Fig. 3.15, is positioned directly in the wake of the top cylinder, located approximately two cylinder diameters downstream. The following discussion is restricted to the modeling of pressure values at this measurement location; similar results for density, $x$-velocity, and $y$-velocity can be found in Appendix A.

Figure 3.16 shows the time evolution of pressure values at the measurement location as a function of rotation speed for flow at Reynolds number $Re = 150$. The top row
shows the pressure signal obtained from the CFD simulations, while the bottom row shows the corresponding results for the generated solutions. These plots show that the nature of the pressure signal changes dramatically as the rotation speed increases, and the generative model is able to track these changes quite well. Furthermore, note that the generative model correctly predicts a steady solution at $\Omega = 2$.

Once again, the similarity between these signals can be evaluated quantitatively by considering the peaks in their power spectral density. In contrast with the PSD for the lift signals, the PSD for the pressure signals will generally have more than one peak. The comparison here thus considers only the frequency corresponding to the largest peak. Figure 3.17 shows the variation of $f_{\text{max}}$ with rotation speed and Reynolds number, and demonstrates strong agreement between the curves associated with the CFD and generated solutions. The largest relative error across all flow quantities, Reynolds numbers, and rotation speeds is 11%, suggesting that the generative model is quite effective at modeling the local properties of the flow. Furthermore, note that these curves include results from Reynolds number $Re = 175$, which is not present in the training data, implying that the model could be effective at modeling the flow properties even at unobserved flow conditions.
Having established confidence that the generative model can be used for accurate simulation at multiple flow conditions, the model is now used to carry out an experiment intended to mimic a design procedure. A design problem typically consists of a set of parameters and objectives, and the goal is to identify design parameter values that perform well according to the design objectives. Exhaustive searches of the design space can be impractical if evaluations of individual design parameters are expensive, as is the case with CFD simulations. However, given the relative low cost of sampling solutions from the generative model, it may be possible to rapidly simulate a given system at a variety of design parameters with the generative model in order to identify promising regions of the design space. In this experiment, the parameter-conditioned generative model is incorporated into a design-like procedure, where an exhaustive search across the parameter space is performed in order to estimate how the critical rotation speed, $\Omega_{\text{crit}}$, varies with Reynolds number.

The goal in this experiment is to identify, as a function of Reynolds number, a plausible range of rotation speeds within which the true critical rotation speed is likely to lie. This is accomplished by first using the generative model to generate solutions at a large number of Reynolds numbers and rotation speeds: $\text{Re} \in \{75, 76, \ldots, 200\}$ and $\Omega = \{0, 0.25, 0.5, 0.75\}$. The results are shown in Figure 3.17.

Figure 3.17: Variation of dominant frequency in pressure signal as a function of rotation speed and Reynolds number for CFD and generated solutions.
Figure 3.18: Left: Norm of time derivative in generated solutions as a function of $\Omega$ at $Re = 200$. Right: Plot of parameter space with points representing identified critical rotation speeds and the plausible region represented by the shaded area.

$\Omega \in \{1.10, 1.11, \ldots, 2.10\}$. This requires the generative model to simulate the cylinder system at nearly 13 000 different flow conditions. In total, running these simulations with the generative model takes a little under 10 hours; in contrast, running the same number of simulations with a CFD solver would take nearly two months.

The critical rotation speed corresponds to the transition from unsteady to steady flow. Thus, the critical rotation speed can be identified by considering the time variation between successive snapshots in generated solutions; this time variation should go to zero when steady flow is achieved. The blue line in the left plot of Fig. 3.18 shows the norm of the numerically estimated time derivative of generated solutions as a function of rotation speed for $Re = 200$. Note that the time derivatives do tend to decrease with rotation speed, with a sharp drop-off observed at $\Omega \approx 1.9$.

Given the smooth variation in the time derivatives, it is somewhat difficult to identify definitively which rotation speed should correspond to $\Omega_{\text{crit}}$. Instead, two time-derivative thresholds, represented by the black dotted lines in Fig. 3.18, are considered, which are assumed to yield a lower- and upper-bound on the true critical rotation speed. By considering these two thresholds, a plausible range for the critical rotation speed:

$$\Omega_{\text{low}} \leq \Omega_{\text{crit}} \leq \Omega_{\text{high}} \quad (3.24)$$
Furthermore, by identifying the plausible range at each Reynolds number, a plausible region of the parameter space can be determined, wherein all critical rotation speeds are assumed to lie. The shaded area in the right plot of Fig. 3.18 represents the plausible region identified from the simulations generated by the parameter-conditioned generative model. The size of this plausible region represents a significant reduction relative to the full parameter space considered in Fig. 3.8. To determine if this plausible region is reasonable, a bisection search was run at Reynolds numbers \(\text{Re} \in \{80, 100, 120, 140, 160, 180, 200\}\) to find the true critical rotation speeds according to the CFD solver. These identified values are represented by the plotted points in Fig. 3.18, and, importantly, are all found to lie within the plausible region.

This section demonstrated that parameter-conditioned generative models can be used to efficiently and accurately simulate two-dimensional fluid flows. The next section studies whether the same techniques can be applied to modeling three-dimensional flows.

### 3.3.2 Three-Dimensional Flow

The test case under consideration is three-dimensional flow over a half-cylinder, as described in Section 2.2. Because it is infeasible to train a model on data from the entire computational domain, the models are once again trained only on data sampled from equispaced points in the vicinity of the cylinder. The data at these points is obtained through multi-linear interpolation, where the ParaView software [85] is used to perform both the interpolation and flow visualizations. The sampling must be performed on a grid with sufficient fineness such that the vortical structures in the flow can be recovered. Thus, samples are drawn at a grid of points that is \(128 \times 64 \times 32\), spanning a domain of \([-1, 11], [-3.75, 3.75], [0, \pi]\) in the stream-, cross-, and span-wise directions, respectively. This sampling procedure yields training inputs that are \(128 \times 64 \times 32 \times 5\), where the first three input dimensions correspond to spatial dimensions and the final input dimension corresponds to the five modeled flow quantities: density, \(x\)-momentum, \(y\)-momentum, \(z\)-momentum, and pressure.
CHAPTER 3. GENERATIVE MODELING OF DYNAMICAL SYSTEMS

Figure 3.19: Visual comparison between flow generated by CFD simulation of entire computational domain and solution obtained by sampling flow quantities at a grid of equispaced points. The visualizations show iso-surfaces of vorticity magnitude, colored according to pressure.

Figure 3.19 shows a qualitative comparison of the flow obtained by running CFD simulations across the entire computational domain against the flow obtained by sampling the flow quantities on the specified grid of points. The visualizations show iso-surfaces of vorticity magnitude, colored according to pressure. It can be seen that the sampled solution retains many of the key features of the original flow, but does not contain any information about the fluid flow far downstream of the cylinder. Hence, the sampled solution represents an approximation to the flow field obtained through CFD simulations, but such an approximation is necessary for training to be tractable.

Training Details

In this experiment, only the Reynolds number is varied. Training data is generated by simulating the half-cylinder system at Reynolds numbers of $Re \in \{175, 200, 250, 300\}$. The flow at these Reynolds numbers exhibits a range of behaviors, with solutions at $Re = 175$ showing mostly laminar flow with some turbulence downstream, and solutions at $Re = 300$ exhibiting a significant amount of turbulence in the cylinder wake. The neural network architectures for the generative model are largely identical to those described for the two-dimensional flow in Section 3.3.1, with one notable exception: the two-dimensional convolutions in the feature extractor and decoder network are replaced with three-dimensional convolutions. Additionally, to account
for the added complexities inherent to modeling three-dimensional, turbulent flow, 32-dimensional latent states are used in place of the four-dimensional latent states employed previously.

Parameter-conditioned generative models are trained on 165 distinct sequences from each flow condition. Training a single model takes approximately three days with the model parameters divided across two NVIDIA K40 GPUs and one NVIDIA GeForce GTX 1070 GPU. The longer training time can be attributed to the larger size of the training inputs and the additional computational cost associated with performing three-dimensional convolutions. Flow simulations generated by the trained model are once again found to be approximately 120 times faster than the corresponding CFD simulations. The next section studies how well the generative models are able to model the fluid flow.

Results

The results presented in this thesis are solely meant to demonstrate that the generative modeling techniques can scale to larger problems; an in-depth quantitative evaluation of the generated results is reserved for future work. Therefore, only a qualitative evaluation of the generated solutions is presented in this section. Figures 3.20 and 3.21 present a visual comparison between the CFD and generated solutions at the four flow conditions contained within the training data. For each flow condition, the top row of images contains results from the CFD simulations, while the bottom row contains results sampled from the generative model. Figure 3.20 shows iso-surfaces of $Q$, the second invariant of the velocity gradient tensor (see the discussion in Section 2.7), colored according to vorticity in the downstream direction. Figure 3.21 shows iso-surfaces of vorticity magnitude, colored according to pressure.

A strong visual similarity can be observed between the CFD solutions and the solutions sampled from the generative model. In particular, the generative model seems effective at capturing the variation in the quantity and nature of vortical structures with Reynolds number. It is worth emphasizing that the iso-surfaces in all images are functions of spatial derivatives of the flow velocities. These spatial derivatives are estimated through finite difference, and thus the presence of the iso-surfaces is
Figure 3.20: Comparison of iso-surfaces of $Q$-values, colored according to downstream vorticity, over time at four flow conditions. CFD solutions are shown in the top row, while generated solutions are shown in the bottom row.
Figure 3.21: Comparison of iso-surfaces of vorticity magnitude, colored according to pressure, over time at four flow conditions. CFD solutions are shown in the top row, while generated solutions are shown in the bottom row.
strongly reliant upon accurately modeling the manner in which the velocity field varies in the cylinder wake. Hence, it is noteworthy that the trained model is capable of generating flow fields that not only contain these vortical structures, but also retain them across successive time steps while propagating them downstream.

3.4 Discussion

This chapter presented a method for learning parameter-conditioned sequential generative models. The central element of the proposed method is a variational inference procedure that enables the discovery of low-dimensional latent states capable of accurately capturing the dynamics of a modeled system. These inferred latent representations can be conditioned on parameters that are assumed to govern the modeled system’s behavior, thereby enabling the generative modeling of dynamical systems, where the modeled dynamics are a function of prescribed parameters.

The proposed method was first evaluated based on its ability to simulate two-dimensional airflow over a pair of counter-rotating cylinders. Extensive qualitative and quantitative experiments demonstrated that learned generative models were capable of effectively modeling both local and global properties of the flow field at a wide range of flow conditions. Furthermore, it was found that, relative to a CFD solver, a speedup of approximately 120× can be obtained by running simulations with a trained generative model. A final set of experiments with data from three-dimensional, turbulent flow simulations demonstrated that the generative modeling techniques can scale to complex and high-dimensional problems.

There are some limitations to the proposed approach that should be addressed in future work. First, in all experiments it was assumed that the flow geometries were fixed. However, many aerospace design problems, such as airfoil or wing design, focus predominantly on how changes in geometry affect the resulting flow. At the moment it is unclear how well generative models would perform if, for example, the separation between the cylinders were allowed to vary, thereby altering the location of the cylinders within the flow field. Hence, more work is needed to identify and address potential shortcomings in this regard. One further limitation is that the employed
neural network architectures require the flow quantities to be sampled at uniformly spaced points. Future work should be dedicated to overcoming this restriction, which will make the techniques better suited to modeling fluid flows from simulations with unstructured grids.
Chapter 4

Dynamic Programming and the Linear Quadratic Regulator

Thus far, this thesis has focused on dynamics modeling. This chapter shifts the focus to control, reviewing ideas from optimal control that will be relevant in later chapters. The discussion begins with an overview of dynamic programming. Subsequently, the linear quadratic regulator (LQR) algorithm is described. The remaining sections explain two extensions to LQR: the iterative linear quadratic regulator and differential dynamic programming.

4.1 Dynamic Programming

This chapter considers methods for optimal control in problems with discrete-time dynamics:

$$x_{t+1} = F(x_t, u_t) \quad (4.1)$$

and additive cost functions:

$$C(x_{1:H}, u_{1:H-1}) = c_H(x_H) + \sum_{t=1}^{H-1} c_t(x_t, u_t). \quad (4.2)$$
Alternative problem formulations can also account for stochasticity in the dynamics and optimize over expected cost. However, only deterministic dynamics are considered here.

A policy (or a control law) is a sequence of functions \( \pi = \{\pi_1, \pi_2, \ldots, \pi_{H-1}\} \) that map states to control inputs [86]:

\[
\mathbf{u}_t = \pi_t(\mathbf{x}_t). \tag{4.3}
\]

An optimal policy \( \pi^* \) is a policy that minimizes the total incurred cost:

\[
\pi^* = \arg\min_{\pi \in \Pi} \left( c_H(\mathbf{x}_H) + \sum_{t=1}^{H-1} c_t(\mathbf{x}_t, \pi_t(\mathbf{x}_t)) \right), \tag{4.4}
\]

where \( \Pi \) is the set of all possible policies. Furthermore, the optimal value function \( V_\tau(\mathbf{x}_\tau) \) represents the total cost incurred by starting in state \( \mathbf{x}_\tau \) at time \( t \) and following the optimal policy in subsequent time steps.

The dynamic programming algorithm provides a method to solve for optimal policies in control problems of this form. The core insight that underlies the dynamic programming algorithm is the principle of optimality, attributed to Richard E. Bellman [87]. The principle of optimality states that if \( \pi^* \) is an optimal policy for a control problem over time steps \( t = 1, \ldots, H \), then the truncated policy \( \pi_{\text{trunc}} = \{\pi_\tau, \pi_{\tau+1}, \ldots, \pi_{H-1}\} \) will be the optimal policy for the control subproblem that wishes to minimize the incurred cost over time steps \( t = \tau, \ldots, H \). The principle of optimality therefore suggests that an optimal policy can be constructed incrementally, where the optimal policy is first determined for the last time step, then the last two time steps, and so on until ultimately the optimal policy for the full control problem has been determined [86].

As an example, consider the optimization problem shown in Fig. 4.1, where the nodes represent various American cities and the edges represent the cost of traveling between cities. The goal is to find the minimum-cost path between San Francisco and New York City, with the optimal path highlighted in blue. Given that the optimal path passes through Chicago, the principle of optimality states that the sub-path between
Figure 4.1: Principle of optimality example. The goal is to find the minimum-cost path between San Francisco and New York City. The optimal path is highlighted in blue.

Chicago and New York City must be the minimum-cost path between Chicago and New York City. From examining Fig. 4.1, it is clear that this is true; there is no path between Chicago and New York City that incurs a lower cost than the highlighted path.

Dynamic programming uses the principle of optimality to derive a recursive relationship for the optimal value function:

$$V_t(x_t) = \min_{u_t} \left( c_t(x_t, u_t) + V_{t+1}(F(x_t, u_t)) \right). \quad (4.5)$$

Hence, the optimal value function can be defined as $V_H(x_H) = c_H(x_H)$ at the final time step, and a sweep backward in time with Eq. (4.5) can be performed to determine the value function at previous time steps. Thus, dynamic programming simplifies the optimal control problem by first solving shorter-horizon subproblems. While often effective, dynamic programming can become intractable when applied to large (or continuous) state spaces. However, the following section introduces an important class of problems for which dynamic programming can provide efficient solutions.
4.2 The Linear Quadratic Regulator

The linear quadratic regulator (LQR) provides an efficient algorithm for optimal control in domains with linear dynamics:

$$x_{t+1} = A_t x_t + B_t u_t$$ \hspace{1cm} (4.6)

and quadratic cost functions of the form:

$$c_t(x_t, u_t) = x_t^T Q_t x_t + u_t^T R_t u_t,$$ \hspace{1cm} (4.7)

where $Q_t$ is symmetric and positive semidefinite and $R_t$ is symmetric and positive definite. The goal in finite-horizon optimal control is to find a sequence of control inputs $u_1, \ldots, u_{H-1}$ that minimizes the total incurred cost over time horizon $H$:

$$C(x_{1:H}, u_{1:H-1}) = x_H^T Q_H x_H + \sum_{t=1}^{H-1} x_t^T Q_t x_t + u_t^T R_t u_t.$$ \hspace{1cm} (4.8)

For systems with linear dynamics and quadratic cost functions, this optimal control problem can be solved exactly through dynamic programming. In the subsequent analysis, it is assumed that the dynamics and cost matrices are time invariant, meaning that $A_t = A$, $B_t = B$, $Q_t = Q$, and $R_t = R$ for all time steps; this assumption will be relaxed later.

Let the optimal value function $V_t(x_t)$ be defined as in Section 4.1:

$$V_t(x_t) = \min_{u_t, \ldots, u_{H-1}} x_{t+1}^T Q_{t+1} x_{t+1} + u_{t+1}^T R_{t+1} u_{t+1}.$$ \hspace{1cm} (4.9)

Using the principle of optimality, $V_t(x_t)$ can be expressed in terms of $V_{t+1}(\cdot)$:

$$V_t(x_t) = \min_{u_t} (x_t^T Q x_t + u_t^T R u_t + V_{t+1}(A x_t + B u_t)).$$ \hspace{1cm} (4.10)
Thus, the optimal value function for the penultimate time step can be written as:

$$V_{H-1}(x_{H-1}) = \min_{u_{H-1}} (x_{H-1}^\top Q x_{H-1} + u_{H-1}^\top R u_{H-1} + V_H(A x_{H-1} + B u_{H-1}))$$ \hspace{2cm} (4.11)

where:

$$V_H(A x_{H-1} + B u_{H-1}) = (A x_{H-1} + B u_{H-1})^\top Q (A x_{H-1} + B u_{H-1})$$ \hspace{2cm} (4.12)

Taking the gradient of Eq. (4.11) with respect to $u_{H-1}$ and equating to zero, the optimal control input, $u_{H-1}^*$, can be found:

$$u_{H-1}^* = -(R + B^\top Q B)^{-1} B^\top Q A x_{H-1} = K_{H-1} x_{H-1}.$$ \hspace{2cm} (4.13)

Hence, the optimal policy $\pi_{H-1}^*(x_{H-1}) = u_{H-1}^*$ can be determined through feedback control on the state with gain $K_{H-1} = -(R + B^\top Q B)^{-1} B^\top Q A$. Substituting the expression for $u_{H-1}^*$ into the optimal value function $V_{H-1}(x_{H-1})$ and simplifying, the following result is obtained:

$$V_{H-1}(x_{H-1}) = x_{H-1}^\top (Q + A^\top Q A - A^\top Q B (R + B^\top Q B)^{-1} B^\top Q A) x_{H-1}$$

$$= x_{H-1}^\top P_{H-1} x_{H-1}.$$ \hspace{2cm} (4.14)

Thus, the optimal value function at the penultimate time step can be expressed as a quadratic function of the current state.

Again applying the dynamic programming principle, the value function $V_{H-2}(x_{H-2})$ can be expressed as:

$$V_{H-2}(x_{H-2}) = \min_{u_{H-2}} (x_{H-2}^\top Q x_{H-2} + u_{H-2}^\top R u_{H-2} + V_{H-1}(x_{H-1})).$$ \hspace{2cm} (4.15)

As with Eq. (4.11), writing the value function in this manner enables the derivation of a linear optimal control law and a quadratic optimal value function. In fact, the optimal value function at any time step $t = 1, \ldots, H - 1$ can be written in the form...
\[ V_t(x_t) = x_t P_t x_t, \text{ where } P_H = Q \text{ and} \]
\[ P_t = Q + A^T P_{t+1} A - A^T P_{t+1} B (R + B^T P_{t+1} B)^{-1} B^T P_{t+1} A. \quad (4.16) \]

This expression, which recursively describes the time evolution of matrix \( P_t \), represents a discrete-time algebraic Riccati equation [88]. The optimal control input at any time step \( t = 1, \ldots, H - 1 \) is then given by:
\[
\begin{align*}
\mathbf{u}_t^* &= - (R + B^T P_{t+1} B)^{-1} B^T P_{t+1} A \mathbf{x}_t \\
&= K_t \mathbf{x}_t.
\end{align*}
\quad (4.17)
\]

If the dynamics and cost matrices are time-varying, then the Riccati equation takes the form:
\[
\begin{align*}
P_t &= Q_t + A_t^T P_{t+1} A_t - A_t^T P_{t+1} B_t (R_t + B_t^T P_{t+1} B_t)^{-1} B_t^T P_{t+1} A_t \\
&= Q_t + A_t^T P_{t+1} A_t - A_t^T P_{t+1} B_t R_t^{-1} B_t^T P_{t+1} A_t. \quad (4.18)
\end{align*}
\]

and the optimal control input can be determined through:
\[
\begin{align*}
\mathbf{u}_t^* &= - (R_t + B_t^T P_{t+1} B_t)^{-1} B_t^T P_{t+1} A_t \mathbf{x}_t. \\
&= K_t \mathbf{x}_t.
\quad (4.19)
\end{align*}
\]

The LQR procedure is presented in Algorithm 1. The optimal control problem can be solved exactly by setting \( P_H = Q_H \), sweeping backward in time using Eq. (4.18) to determine \( P_{H-1}, \ldots, P_1 \), and finally determining the control inputs at all time steps from Eq. (4.19).

### 4.2.1 Alternative Cost Functions

The LQR techniques presented above can be applied to problems with cost functions that are more complex than the form given in Eq. (4.7). A common formulation for an optimal control problem involves evaluating state cost according to distance from
Algorithm 1 Linear Quadratic Regulator Algorithm

Input: $x_1, A_{1:H}, B_{1:H}, Q_{1:H}, R_{1:H-1}$

Set $P_H = Q_H$

for $t = H - 1, \ldots, 1$ do

$P_t = Q_t + A_t^\top P_{t+1} A_t + A_t^\top P_{t+1} B_t (R_t + B_t^\top P_{t+1} A_t)^{-1} B_t^\top P_{t+1} A_t$

end for

for $t = 1, \ldots, H - 1$ do

$K_t = -(R_t + B_t^\top P_{t+1} B_t)^{-1} B_t^\top P_{t+1} A_t$

$u_t^* = K_t x_t$

$x_{t+1} = A_t x_t + B_t u_t^*$

end for

a desired goal state $x_{\text{goal}}$:

$$c_t(x_t, u_t) = (x_t - x_{\text{goal}})^\top Q_t (x_t - x_{\text{goal}}) + u_t R_t u_t$$

$$= x_t^\top Q_t x_t + u_t R_t u_t - 2 x_{\text{goal}}^\top Q_t x_t + x_{\text{goal}}^\top Q_t x_{\text{goal}}$$

(4.20)

$$= x_t^\top Q_t x_t + u_t R_t u_t + q_t^\top x_t + q_t,$$

where $q_t = -2 Q_t x_{\text{goal}}$ and $q_t = x_{\text{goal}}^\top Q_t x_{\text{goal}}$. With a cost function of this form, it is still possible to find an exact solution to the optimal control problem. In fact, LQR techniques can be applied to any problem with a cost function that takes the general form:

$$c_t(x_t, u_t) = x_t^\top Q_t x_t + u_t R_t u_t + u_t^\top H_t x_t + q_t^\top x_t + r_t^\top u_t + q_t$$

(4.21)

Due to the presence of the affine and bilinear terms in the cost function, the optimal control inputs become affine functions of the current state and the optimal value function has the form:

$$V_t(x_t) = x_t^\top P_t x_t + p_t^\top x_t + p_t,$$

(4.22)

where $P_t$, $p_t$, and $p_t$ can all be determined through recursive updates analogous to Eq. (4.18).
4.3 The Iterative Linear Quadratic Regulator

The iterative linear quadratic regulator (iLQR) algorithm [89] extends LQR techniques to problems with nonlinear dynamics and non-quadratic cost functions. Beginning with a reference trajectory of states \( \hat{x}_{1:H} \) and control inputs \( \hat{u}_{1:H-1} \) that satisfy the system dynamics:

\[
\dot{x}_{t+1} = F(\hat{x}_t, \hat{u}_t),
\]

(4.23)

the iterative linear quadratic regulator aims to find updates to the sequence of control inputs \( \delta u_{1:H-1} \) in order to minimize the total incurred cost, where

\[
\delta u_t = u_t - \hat{u}_t
\]

(4.24)

and the incurred cost is given by Eq. (4.2). The change in control inputs in turn induce changes in the state trajectory \( \delta x_{1:H} \), where

\[
\delta x_t = x_t - \hat{x}_t.
\]

(4.25)

The algorithm begins by using Taylor expansions to find linear approximations to the system dynamics and quadratic approximations to the system cost. Defining the matrices:

\[
A_t = \frac{\partial F}{\partial x}(\hat{x}_t, \hat{u}_t) \quad \text{and} \quad B_t = \frac{\partial F}{\partial u}(\hat{x}_t, \hat{u}_t),
\]

(4.26)

the approximate system dynamics can be written as:

\[
x_{t+1} = F(x_t, u_t) \approx F(\hat{x}_t, \hat{u}_t) + A_t(x_t - \hat{x}_t) + B_t(u_t - \hat{u}_t).
\]

(4.27)

Hence, the dynamics of the state deviations can be approximated according to:

\[
\delta x_{t+1} = A_t \delta x_t + B_t \delta u_t.
\]

(4.28)
Furthermore, the instantaneous cost \( c_t(x_t, u_t) \) can be approximated as:

\[
c_t(x_t, u_t) \approx c_t(\hat{x}_t, \hat{u}_t) + \frac{1}{2} \delta x_t^T c_{xx,t} \delta x_t + \frac{1}{2} \delta u_t^T c_{uu,t} \delta u_t
\]

\[
+ \frac{1}{2} \delta x_t^T c_{ux,t} \delta u_t + \frac{1}{2} \delta u_t^T c_{ux,t} \delta x_t + c_{x,t}^T \delta x_t + c_{u,t}^T \delta u_t,
\]

where, for example:

\[
c_{ux,t} = \frac{\partial^2 c_t}{\partial u \partial x}(\hat{x}_t, \hat{u}_t).
\]

The LQR formulation requires that the value function \( V_t(x_t) \) can be expressed in quadratic form. At the final time step, the value function is simply the cost associated with the terminal state. Hence, \( V_H(\hat{x}_H) = c_H(\hat{x}_H) \) and, given the approximate local cost,

\[
V_H(x_H) = V_H(\hat{x}_H) + \frac{1}{2} \delta x_H^T P_H \delta x_H + p_H^T \delta x_H,
\]

where

\[
P_H = c_{xx,H}, \quad p_H = c_{x,H}.
\]

In general, the value function is defined by:

\[
V_t(x_t) = \min_{u_t} (c_t(x_t, u_t) + V_{t+1}(F(x_t, u_t)))
\]

\[
= \min_{u_t} Q_t(x_t, u_t),
\]

where \( Q(x_t, u_t) \) is the state-action value function, which describes the cost obtained by taking action \( u_t \) in state \( x_t \) and acting optimally in subsequent time steps. Thus, the optimal value of \( \delta u_t \) will be the value that minimizes \( Q(\hat{x}_t + \delta x_t, \hat{u}_t + \delta u_t) \). Using the derived approximations to the dynamics and cost function, this quantity can be written as:

\[
Q(\hat{x}_t + \delta x_t, \hat{u}_t + \delta u_t) = Q(\hat{x}_t, \hat{u}_t) + \frac{1}{2} \delta x_t^T Q_{xx,t} \delta x_t + \frac{1}{2} \delta u_t^T Q_{uu,t} \delta u_t
\]

\[
+ \frac{1}{2} \delta x_t^T Q_{xu,t} \delta u_t + \frac{1}{2} \delta u_t^T Q_{ux,t} \delta x_t + Q_{x,t}^T \delta x_t + Q_{u,t}^T \delta u_t,
\]

(4.34)
where:

\[ Q_{xx,t} = c_{xx,t} + A_t^T P_{t+1} A_t \]
\[ Q_{uu,t} = c_{uu,t} + B_t^T P_{t+1} B_t \]
\[ Q_{ux,t} = c_{ux,t} + B_t^T P_{t+1} A_t \]
\[ Q_{x,t} = c_{x,t} + p_{t+1}^T A_t \]
\[ Q_{u,t} = c_{u,t} + p_{t+1}^T B_t. \] (4.35)

and \( Q_{xu,t} = Q_{ux,t} \).

Given the approximations to the dynamics and cost function in the neighborhood of the reference trajectory, iLQR uses a procedure similar to that derived in Section 4.2 in order to find the optimal action perturbations. In order to do so, a formula for \( \delta u_t^* \) must be found, along with update equations for \( P_t \) and \( p_t \), the terms that define the value function. Taking the derivative of Eq. (4.34) with respect to \( \delta u_t \) and equating to zero, the optimal control perturbation is found to be:

\[ \delta u_t^* = K_t \delta x_t + k_t, \] (4.36)

where:

\[ K_t = -Q_{uu,t}^{-1} Q_{ux,t} \quad \text{and} \quad k_t = -Q_{uu,t}^{-1} Q_{u,t}. \] (4.37)

Thus, the new control input is given by:

\[ u_t = \hat{u}_t + \delta u_t^* \]
\[ = \hat{u}_t + K_t \delta x_t + k_t. \] (4.38)

Finally, substituting the derived value of \( \delta u_t^* \) into Eq. (4.33), the update equations for \( p_t \) and \( P_t \) can be derived:

\[ P_t = Q_{x,t} - K_t^T Q_{uu,t} K_t \]
\[ p_t = Q_{x,t} - K_t^T Q_{uu,t} k_t, \] (4.39)

which are functions of \( p_{t+1} \) and \( P_{t+1} \).
Algorithm 2 iLQR Forward Pass

**Input:** $\hat{x}_{1:H}, \hat{u}_{1:H-1}, K_{1:H-1}, k_{1:H-1}$  
Set $x_1 = \hat{x}_1$  
for $t = 1, \ldots, H$ do  
    $u_t = \hat{u}_t + K_t(x_t - \hat{x}_t) + k_t$  
    $x_{t+1} = F(x_t, u_t)$  
end for  
return $x_{1:H}, u_{1:H-1}$

Algorithm 3 iLQR Backward Pass

**Input:** $\dot{x}_{1:H}, \hat{u}_{1:H-1}$, approximate dynamics and cost function along trajectory  
Set $P_H = c_{xH}, P_H = c_{xH}$  
for $t = H-1, \ldots, 1$ do  
    Calculate $Q_{xx,t}, Q_{uu,t}, Q_{ux,t}, Q_{x,t}, Q_{u,t}$ through Eq. (4.41)  
    Calculate $K_t$ and $k_t$ through Eq. (4.37)  
    Calculate $P_t$ and $p_t$ through Eq. (4.39)  
end for  
return $K_{1:H-1}, k_{1:H-1}$

The iLQR algorithm iterates between performing backward and forward passes until convergence. Given an initial state and action trajectory, the backward pass consists of finding the control gains through Eq. (4.37) and components of the value function through Eq. (4.39). The forward pass simulates the system, using the derived control gains to generate a new nominal state and action trajectory. The forward pass and backward pass are summarized in Algorithm 2 and Algorithm 3, respectively. Iteration generally terminates once an update to the nominal trajectory yields a change in the incurred cost with a magnitude that falls below a predetermined threshold.

### 4.4 Differential Dynamic Programming

Differential Dynamic Programming (DDP) [90] is another approach for extending LQR techniques to problems with nonlinear dynamics and non-quadratic costs. DDP shares many similarities with iLQR, with one major difference. The iLQR algorithm first finds approximations to the dynamics and cost functions along a reference trajectory,
then derives the state-action value function $Q(x_t, u_t)$ assuming the approximations represent the true dynamics and cost. In contrast, the DDP algorithm forms an approximation to the state-action value function directly.

A second-order Taylor expansion of the $Q$-function can be written in exactly the same form as Eq. (4.34); however, the coefficients now represent derivatives of the $Q$-function. For example,

$$Q_{ux,t} = \frac{\partial^2 Q}{\partial u \partial x}(\hat{x}_t, \hat{u}_t).$$  \hspace{1cm} (4.40)

By evaluating each of these derivatives, it can be shown that [91]:

$$Q_{xx,t} = c_{xx,t} + A_t^\top P_{t+1} A_t + p_{t+1} \cdot F_{xx,t}$$
$$Q_{uu,t} = c_{uu,t} + B_t^\top P_{t+1} B_t + p_{t+1} \cdot F_{uu,t}$$
$$Q_{ux,t} = c_{ux,t} + B_t^\top P_{t+1} A_t + p_{t+1} \cdot F_{ux,t}$$
$$Q_{x,t} = c_{x,t} + p_{t+1}^\top A_t$$
$$Q_{u,t} = c_{u,t} + p_{t+1}^\top B_t,$$

where all quantities are as defined in Section 4.3, the dot represents tensor contraction, and the terms $F_{xx,t}$, $F_{uu,t}$, and $F_{ux,t}$ all represent second derivatives of the system dynamics. Thus, the iLQR and DDP algorithms are nearly identical, except DDP uses a second-order approximation to the system dynamics (note that for systems with linear dynamics, the two algorithms are identical).

As stated previously, both iLQR and DDP construct quadratic approximations to the state-action value function. Building such an approximation requires evaluating (or in the case of iLQR, approximating) the Hessian of the $Q$-function. In circumstances where the Hessian is not positive definite or the minimum is far from the nominal trajectory, unreliable or counterproductive updates may be performed. A common form of regularization to combat this problem is to add scalar values to the diagonal of the Hessian $Q_{uu,t}$ [92]:

$$\tilde{Q}_{uu,t} = Q_{uu,t} + \mu I,$$  \hspace{1cm} (4.42)

where $I$ is the identity matrix. This modification will more heavily penalize deviations from the nominal action sequence, and hence will make the optimal updates $\delta \hat{u}_t^\star$
more conservative. However, this regularization scheme can have non-uniform effects depending on the local sensitivity of the system dynamics to the control inputs. An alternative and often more effective form of regularization is to penalize deviations from the state trajectory rather than the controls, which requires the following modifications to the $Q$-function Hessians [91]:

\[
\begin{align*}
\dot{Q}_{uu,t} &= c_{uu,t} + B_t^\top (P_{t+1} + \mu I) B_t + p_{t+1} \cdot F_{uu,t} \\
\dot{Q}_{ux,t} &= c_{ux,t} + B_t^\top (P_{t+1} + \mu I) A_t + p_{t+1} \cdot F_{ux,t}.
\end{align*}
\] (4.43)

Another potential problem arises due to the fact that the approximate dynamics and cost functions are only valid in a region surrounding the nominal trajectory. When the system is simulated during the forward pass, it may stray into regions of the state space where the derived approximations are no longer valid. Under these circumstances, it may turn out that the incurred cost increases under the updated action sequence. For this reason, it is common to perform a backtracking line search during the forward pass in order to make sure the system remains in regions of the state space where the model is valid [91]. To perform the line search, a parameter $\alpha \in [0, 1]$ is introduced, and the applied actions during the forward pass are given by:

\[
u_t = \hat{u}_t + K_t (x_t - \dot{x}_t) + \alpha k_t
\] (4.44)

Hence, for $\alpha = 0$ there will be no change to the state and action trajectory and for $\alpha = 1$ the updated actions will be the same as those given by Eq. (4.38). The parameter $\alpha$ can be initialized to a value of one and decreased until the incurred cost during the forward pass improves relative to the previous nominal trajectory.

4.5 Discussion

This chapter introduced topics from optimal control that will be relevant to later chapters in this thesis. The chapter began by explaining the principle of optimality and its relation to the dynamic programming algorithm. Next, the linear quadratic regulator (LQR) was described, which relies upon dynamic programming to derive
optimal control policies for problems with linear dynamics and quadratic costs. Finally, the iLQR and differential dynamic programming algorithms were discussed, which are important extensions to LQR that allow for the application of LQR concepts to problems with nonlinear dynamics and non-quadratic cost functions.

An important takeaway from this chapter is that optimal control problems can be solved efficiently for systems with linear dynamics and quadratic costs. Thus, if data-driven dynamics models can be learned such that the derived models are linear, then this may enable more efficient and effective control. The next chapter introduces important concepts related to learning linear dynamics.
Chapter 5

Learning Linear Dynamics and the Koopman Operator

Chapter 4 established that linear models are desirable for control. This chapter explores methods for learning linear dynamics models from data. The primary focus of the ensuing discussion will be on Koopman theory, which describes how nonlinear dynamical systems can be mapped to linear systems. First, however, dynamic mode decomposition (DMD) is described, which is a data-driven method for deriving approximate linear dynamics models, and has important connections to Koopman theory.

5.1 Dynamic Mode Decomposition

Dynamic mode decomposition (DMD) [93, 94] is a data-driven technique for modeling dynamical systems that has proven effective for analysis, prediction, and control in a variety of problem domains. DMD can be applied to both continuous-time and discrete-time problems; the focus here will be restricted to discrete-time dynamics. Given data from a dynamical system that evolves according to:

$$x_{t+1} = F(x_t),$$

(5.1)
where the nature of $F(\cdot)$ is unknown, dynamic mode decomposition attempts to find the linear model

$$x_{t+1} = Ax_t$$  \hfill (5.2)

that best approximates the system dynamics in the least-squares sense, i.e. it minimizes the loss:

$$\mathcal{L} = \sum_{t=1}^{T-1} \| x_{t+1} - Ax_t \|_2^2.$$  \hfill (5.3)

Defining the matrices $X$ and $Y$ as

$$X = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots \\ x_1 & x_2 & \cdots & x_{T-1} \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix}, \quad Y = \begin{bmatrix} \vdots & \vdots & \vdots \\ x_2 & x_3 & \cdots & x_T \end{bmatrix},$$  \hfill (5.4)

the best-fit $A$-matrix will be given by:

$$A = YX^\dagger,$$  \hfill (5.5)

where $X^\dagger$ is the Moore-Penrose pseudoinverse of $X$, defined as:

$$X^\dagger = (X^\dagger X)^{-1}X^\dagger.$$  \hfill (5.6)

The derived $A$-matrix will be of size $n \times n$, where $n$ is the dimensionality of the state. If the state is very large, then it may be intractable to compute and store $A$ in the manner described by Eq. (5.5). For this reason, a more widely used version of the DMD algorithm instead attempts to find a low-rank approximation to the matrix $A$. The algorithm begins by performing the singular value decomposition of $X$:

$$X = U\Sigma V^\dagger.$$  \hfill (5.7)

Here, a low-rank truncation of the SVD is usually performed using, for example, the
energy criterion described in Section 2.5:

\[ X \approx \tilde{U} \tilde{\Sigma} \tilde{V}^\top, \]  

(5.8)

where \( \tilde{U} \in \mathbb{R}^{n \times r} \), \( \tilde{\Sigma} \in \mathbb{R}^{r \times r} \), \( \tilde{V} \in \mathbb{R}^{n \times r} \), and \( r \) represents the level of truncation performed (often \( r \ll n \)). The low-rank projection of \( A \) onto the SVD basis is then given by:

\[
\tilde{A} = \tilde{U}^\top A \tilde{U} \\
\triangleq \tilde{U}^\top \tilde{Y} \tilde{V} \tilde{\Sigma}^{-1},
\]  

(5.9)

which makes use of the fact that

\[ A = \tilde{Y} X^\dagger \approx \tilde{Y} \tilde{V} \tilde{\Sigma}^{-1} \tilde{U}. \]  

(5.10)

Note that, even if no truncation is performed, \( \tilde{A} \) will be of size \( T \times T \), and could have significantly fewer entries than \( A \) if \( T \ll n \).

The matrix \( \tilde{A} \) represents a linear model that can be used to describe the dynamics of reduced states \( \tilde{x}_t = \tilde{U}^\top x_t \), which represent the projection of the state vector onto the (truncated) SVD basis. The eigenvalue decomposition of \( \tilde{A} \) is given by:

\[ \tilde{A} W = W \Lambda, \]  

(5.11)

where the columns of \( W \) are the eigenvectors of \( \tilde{A} \) and the diagonal entries of \( \Lambda \) are the eigenvalues of \( \tilde{A} \). It can be shown that the eigenvalues of \( \tilde{A} \) are also the eigenvalues of \( A \); the eigenvectors of \( A \), denoted by \( \Phi \), can be found through \([93]\):

\[ \Phi = \tilde{Y} \tilde{V} \tilde{\Sigma}^{-1} W. \]  

(5.12)

The columns of \( \Phi \), denoted by \( \phi_i \), are known as DMD modes, which represent important spatial features of the modeled system. The corresponding eigenvalues, \( \lambda_i \), describe the temporal behavior of the modes, including the frequency of oscillation and rate of
growth or decay.

Given the DMD modes and eigenvectors, the state at a given time step can be expressed as:

\[ x_t = \sum_{i=1}^{r} \phi_i \lambda_i^{t-1} b_i \]
\[ = \Phi \Lambda^{t-1} b, \]

(5.13)

where \( b = \Phi^\dagger x_1 \). Hence, dynamic mode decomposition allows for very efficient simulation, as advancing the system forward in time only requires multiplying each mode by a constant value \( \lambda_i \). For this reason, DMD is often used for prediction tasks where, given a sequence of states \( x_{1:T} \), the goal is to predict the value of states \( x_{T+1:T+H} \) over some horizon \( H \). Note that DMD provides the best fit linear model for the observed state trajectory \( x_{1:T} \); there is no guarantee that the derived model will accurately predict the time evolution of the system in other regions of the state space. The next section shows how the DMD algorithm can be altered to model systems with control inputs.

5.2 Dynamic Mode Decomposition with Control

Dynamic mode decomposition with control (DMDc) [95] extends dynamic mode decomposition to systems with control inputs. Given data describing the time evolution of a system \( x_{1:T} \) subject to control inputs \( u_{1:T-1} \), DMDc finds the best fit linear model of the form:

\[ x_{t+1} = Ax_t + Bu_t, \]

(5.14)

where \( u \in \mathbb{R}^p \) and \( B \in \mathbb{R}^{n \times p} \). The DMDc algorithm relies upon many of the same ideas as dynamic mode decomposition, with the distinction that it must also account for the presence of the \( B \)-matrix that governs the effects of control inputs. There are two distinct approaches for performing DMDc depending on the amount of available information about the underlying system; one approach estimates the \( A \)-matrix given that \( B \) is known, while the other approach estimates \( A \) and \( B \) jointly. These approaches
are discussed in further detail in the following subsections.

### 5.2.1 DMDc with known $B$-matrix

The first approach attempts to estimate an $A$-matrix that can be used to simulate the time evolution of a modeled system for which $B$ is known. Define matrices $X$ and $Y$ as in Section 5.1. Additionally define the matrix $\Gamma$ as:

$$
\Gamma = \begin{bmatrix}
\mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_{T-1}
\end{bmatrix}.
$$

(5.15)

The goal is to find the linear model that best approximates the system dynamics such that

$$
Y \approx AX + B\Gamma.
$$

(5.16)

This can be framed as a least-squares problem with the associated solution

$$
A = (Y - B\Gamma)X^\dagger.
$$

(5.17)

Following the same procedure as in Section 5.1, a low-rank approximation of $A$ can be found through:

$$
\tilde{A} = \tilde{U}^\dagger(Y - B\Gamma)\tilde{V}\tilde{\Sigma}^{-1},
$$

(5.18)

and the dynamic modes of the system can be found through

$$
\Phi = (Y - B\Gamma)\tilde{V}\tilde{\Sigma}^{-1}W.
$$

(5.19)

### 5.2.2 DMDc with unknown $B$-matrix

A more realistic approach applies to scenarios where neither $A$ nor $B$ are known, and must be jointly estimated. Again, the task of finding a linear dynamics model that satisfies Eq. (5.16) can be framed as a least-squares problem, with the solution given
by:

\[ [A \ B] = YZ^\dagger, \]  \hspace{1cm} (5.20)

where \( Z \) is defined as:

\[ Z = \begin{bmatrix} X \\ \Gamma \end{bmatrix}. \]  \hspace{1cm} (5.21)

Let the low-rank truncation of the SVD of \( Z \) be given by \( Z \approx \tilde{U}\tilde{\Sigma}\tilde{V}^\dagger \) and let the truncation of the SVD of \( Y \) be given by \( Y \approx \hat{U}\hat{\Sigma}\hat{V}^\dagger \) (note that the level of truncation can be different for these two matrices). Also define the matrices \( \tilde{U}_x \in \mathbb{R}^{n \times r} \) and \( \tilde{U}_u \in \mathbb{R}^{p \times r} \) such that:

\[ \tilde{U} = \begin{bmatrix} \tilde{U}_x \\ \tilde{U}_u \end{bmatrix}. \]  \hspace{1cm} (5.22)

It can be shown that low-rank approximations to \( A \) and \( B \) can be found through [95]:

\[ \tilde{A} = \hat{U}^\dagger Y\tilde{V}\tilde{\Sigma}^{-1}\tilde{U}_x^\dagger \hat{U} \]  \hspace{1cm} (5.23)

\[ \tilde{B} = \hat{U}^\dagger Y\tilde{V}\tilde{\Sigma}^{-1}\tilde{U}_u^\dagger \]

and the dynamic modes can be found through

\[ \Phi = Y\tilde{V}\tilde{\Sigma}^{-1}\tilde{U}_x\hat{U}W. \]  \hspace{1cm} (5.24)

### 5.3 The Koopman Operator

Koopman theory [96], originally introduced by B.O. Koopman in 1931, provides a different lens through which general dynamical systems can be mapped to linear systems. Koopman theory posits the existence of a linear operator, known as the Koopman operator, that advances scalar-valued observation functions \( g(\cdot) \) forward in time:

\[ \mathcal{K}g(x_t) = g(F(x_t)) = g(x_{t+1}). \]  \hspace{1cm} (5.25)

Henceforth, the observation functions \( g(\cdot) \) will be referred to as observables and the output of the observables, \( g(x) \), will be referred to as observations. The Koopman
operator is infinite-dimensional because it operates on the space of observable functions. Thus, the Koopman operator describes the linear evolution of observables in an infinite-dimensional function space, rather than describing the (often nonlinear) evolution of state vectors in a finite-dimensional state space.

Given that $\mathcal{K}$ is a linear operator, it will have eigenvalue/eigenfunction pairs $\lambda_i \in \mathbb{C}$, $\varphi_i : \mathbb{R}^n \to \mathbb{C}$ that satisfy:

$$\mathcal{K}\varphi_i(x) = \lambda_i \varphi_i(x), \quad i \in \mathbb{N}. \quad (5.26)$$

Observables can be expressed as a linear combination of these eigenfunctions [94]:

$$g(x) = \sum_{i=1}^{\infty} c_i \varphi_i(x), \quad (5.27)$$

where the $c_i$ are scalar coefficients. Similar to dynamic modes, each term $c_i \varphi_i(x)$ is referred to as a Koopman mode [97], which can be advanced forward in time through multiplication by its corresponding eigenvalue:

$$g(x_t) = \sum_{i=1}^{\infty} \lambda_i^{t-1} \varphi_i(x_1)c_i. \quad (5.28)$$

The decay rate and frequency of each Koopman mode is governed by the amplitude and phase of its eigenvalue [98].

Analysis will now be restricted to a subset of these eigenfunctions $\{\varphi_1, \varphi_2, \ldots, \varphi_k\}$. While it will generally not be possible to capture the full dynamics of a system with a finite number of eigenfunctions, prudent selection of the eigenfunctions can allow for accurate approximations. Of particular interest are the eigenfunctions that describe the long timescale behavior of the Koopman operator, which are the eigenfunctions with eigenvalues near the unit circle in discrete time [97]. Define the vector-valued eigenfunction $\varphi$ as $\varphi = [\varphi_1, \varphi_2, \ldots, \varphi_k]^T$. It is clear that the time evolution of $\varphi$ is
governed by a finite-dimensional linear operator \( K \in \mathbb{R}^{k \times k} \), since:

\[
K \varphi(x_t) = \varphi(x_{t+1})
\]

\[
= \begin{bmatrix}
\lambda_1 & 0 & \cdots & 0 \\
0 & \lambda_2 & \cdots & 0 \\
0 & 0 & \ddots & 0 \\
0 & 0 & \cdots & \lambda_k
\end{bmatrix}
\begin{bmatrix}
\varphi_1(x_t) \\
\varphi_2(x_t) \\
\vdots \\
\varphi_k(x_t)
\end{bmatrix}
\]

\[
= K \varphi(x_t).
\]  

(5.29)

Thus, the evolution of the eigenfunctions can be described by a finite-dimensional linear model even if the underlying dynamics are nonlinear. Unfortunately, while some techniques have been proposed for discovering Koopman eigenfunctions [99, 100], their form is often unknown, particularly for high-dimensional systems and systems with unknown governing equations.

More generally, any set of observables that can be expressed as a linear combination of a finite set of eigenfunctions will also have finite-dimensional linear dynamics. Let \( g = [g_1, g_2, \ldots, g_m]^\top \) be a vector-valued observable, where each component \( g_i \) is in \( \text{span}\{\varphi_1, \varphi_2, \ldots, \varphi_k\} \). Then the time evolution of the vector-valued observable is described by:

\[
g(x_{t+1}) = \sum_{i=1}^{k} \lambda_i \varphi_i(x_t) c_i
\]

\[
= \hat{K} g(x_t),
\]  

(5.30)

where \( \hat{K} \in \mathbb{R}^{m \times m} \). The eigenvalues of \( \hat{K} \) will also be eigenvalues of the matrix \( K \) defined in Eq. (5.29), and Koopman eigenfunctions can be found through

\[
\varphi_i(x) = z_i^\top g(x),
\]  

(5.31)

where \( z_i \) is a left eigenvector of \( \hat{K} \) [98]. This implies that, if the observable \( g(x) = x \) can be expressed in terms of a finite number of Koopman eigenfunctions, then the time evolution of the state can be described by a finite-dimensional linear operator.
Furthermore, in these circumstances the dynamic modes recovered by DMD can correspond to Koopman modes under certain assumptions, providing theoretical justification for the procedure [93].

The assumption in DMD that the observable \( g(x) = x \) can be expressed in terms of a finite number of Koopman eigenfunctions also implies that Koopman eigenfunctions can be recovered through Eq. (5.31), where \( z_i \) are the left eigenvectors of the matrix \( A = YX^\dagger \). In other words, it assumes that Koopman eigenfunctions can be expressed as a linear combination of the individual components of the state vector. This assumption is generally unjustified, and highlights the importance of having a sufficiently rich set of observable functions contained in \( g(\cdot) \). A procedure known extended dynamic mode decomposition (eDMD) [97, 101] modifies DMD by replacing the matrices \( X \) and \( Y \) with matrices \( \tilde{X} \) and \( \tilde{Y} \) defined as:

\[
\tilde{X} = \begin{bmatrix}
g(x_1) & g(x_2) & \cdots & g(x_{T-1}) 
\end{bmatrix}, \quad \tilde{Y} = \begin{bmatrix}
g(x_2) & g(x_3) & \cdots & g(x_T)
\end{bmatrix},
\]

where each component of \( g(\cdot) \) is a candidate scalar-valued function of the state, which can be constructed to be more expressive than the \( g(x) = x \) observable used in DMD. Extended DMD typically relies upon hand-specifying a dictionary of candidate functions to include in \( g(x) \); such a process can be tedious and may require prior knowledge about a modeled system. This difficulty has motivated several recent approaches that attempt to learn appropriate observables from data [98, 102–104]. One such approach will be discussed in detail in Chapter 6.

5.3.1 Koopman Operator Example

The following examples, originally presented by Brunton et al. [105], illustrate how nonlinear dynamical systems can be mapped to linear dynamical systems. First, consider the following continuous time dynamical system that describes the evolution...
of state $\mathbf{x} = [x_1, x_2]^\intercal$:

$$
\begin{align*}
\dot{x}_1 &= \mu x_1 \\
\dot{x}_2 &= \lambda (x_2 - x_1^2).
\end{align*}
$$

(5.33)

This system is nonlinear due to the presence of the $x_1^2$ term in the time derivative of $x_2$. Now introduce the vector-valued observable function $\mathbf{g}(\mathbf{x}) = [g_1(\mathbf{x}), g_2(\mathbf{x}), g_3(\mathbf{x})]^\intercal$, where

$$
\mathbf{g}(\mathbf{x}) = \begin{bmatrix} x_1 \\
x_2 \\
x_1^2 \end{bmatrix}.
$$

(5.34)

The time derivatives of the components of $\mathbf{g}$ are given by:

$$
\begin{align*}
\dot{g}_1(\mathbf{x}) &= \dot{x}_1 = \mu x_1 = \mu g_1(\mathbf{x}) \\
\dot{g}_2(\mathbf{x}) &= \dot{x}_2 = \lambda (x_2 - x_1^2) = \lambda (g_2(\mathbf{x}) - g_3(\mathbf{x})) \\
\dot{g}_3(\mathbf{x}) &= 2x_1 \dot{x}_1 = 2\mu x_1^2 = 2\mu g_3(\mathbf{x}).
\end{align*}
$$

(5.35)

Thus, the time evolution of the observables can be described through the following linear dynamical system:

$$
\begin{aligned}
\frac{d}{d\tau} \mathbf{g}(\mathbf{x}) &= \begin{bmatrix} \mu & 0 & 0 \\
0 & \lambda & -\lambda \\
0 & 0 & 2\mu \end{bmatrix} \mathbf{g}(\mathbf{x}) \\
\implies \dot{\mathbf{g}}(\mathbf{x}) &= K \mathbf{g}(\mathbf{x}).
\end{aligned}
$$

(5.36)

An analogous example considers the discrete-time evolution of state $\mathbf{x} = [x_1, x_2]^\intercal$, where:

$$
F(\mathbf{x}) = \begin{bmatrix} \lambda x_1 \\
\mu x_2 + (\lambda^2 - \mu)x_1^2 \end{bmatrix}
$$

(5.37)

Again define the vector-valued observable $\mathbf{g}(\mathbf{x})$ as in Eq. (5.34). It is clear that the
update equation for $g(x)$ can be written as:

$$
\begin{bmatrix}
\lambda & 0 & 0 \\
0 & \mu & (\lambda^2 - \mu) \\
0 & 0 & \lambda^2
\end{bmatrix} g(x) = g(x_{t+1})
$$

(5.38)

$$
\implies g(x_{t+1}) = Kg(x),
$$

Hence, it is once again possible to map a nonlinear dynamical system to a finite-dimensional linear system. Here, the matrix $K$ is the finite-dimensional Koopman operator capable of advancing the observations $g(x)$ forward in time. Its associated eigenvalues are $\lambda_1 = \lambda$, $\lambda_2 = \mu$, $\lambda_3 = \lambda^2$. The left eigenvectors associated with these eigenvalues are:

$$
z_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad z_2 = \begin{bmatrix} 0 \\ 1 \\ -1 \end{bmatrix}, \quad \text{and} \quad z_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.
$$

(5.39)

The associated eigenfunctions are given by $\varphi_i(x) = z_i^T g(x)$. Thus,

$$
\varphi_1(x) = x_1, \quad \varphi_2(x) = x_2 - x_1^2, \quad \text{and} \quad \varphi_3(x) = x_1^2.
$$

(5.40)

Therefore, defining $\varphi(x) = [\varphi_1, \varphi_2, \varphi_3]^T$, the dynamics of the diagonalized system are described by:

$$
\varphi(x_{t+1}) = \begin{bmatrix}
\lambda & 0 & 0 \\
0 & \mu & 0 \\
0 & 0 & \lambda^2
\end{bmatrix} \varphi(x_t).
$$

(5.41)

5.4 The Koopman Operator and Control

In systems with control inputs, the Koopman operator acts on observables $h : \mathbb{R}^{n+p} \to \mathbb{R}$ that are functions of the state and control input:

$$
Kh(x_t, u_t) = h(F(x_t, u_t), u_{t+1}) = h(x_{t+1}, u_{t+1}).
$$

(5.42)
Proctor et al. [106] outlined how this definition can be modified depending on the form of control input. A summary of these possible definitions is provided in the following subsections.

### 5.4.1 Closed-Loop Control Inputs

In closed-loop control, a policy \( \pi : \mathbb{R}^n \rightarrow \mathbb{R}^p \) maps the current state to a control input. Under such circumstances, the system dynamics are described by:

\[
x_{t+1} = F(x_t, u_t) = F(x_t, \pi(x_t)),
\]

and are therefore only a function of the state. Similarly, the Koopman operator can be written in terms of only the state:

\[
K_h(x_t, \pi(x_t)) = h(x_{t+1}, \pi(x_{t+1})).
\]  

### 5.4.2 Open-Loop Control Inputs

If, rather than being a function of the state, the control input represents a constant forcing term \( \bar{u} \), then the Koopman operator is defined as:

\[
K_h(x_t, \bar{u}) = h(F(x_t, \bar{u}), \bar{u}) = h(x_{t+1}, \bar{u}).
\]
which uses observables of the current state and control input to only propagate forward observables that are functions of the state; no attempt is made to model the evolution of control inputs. This perspective makes it possible to draw a link between Koopman analysis and the DMDc procedure outlined in Section 5.2. Define the vector valued observable \( h(\cdot) \) such that:

\[
\begin{bmatrix}
g(x_t) \\
\ell(x_t, u_t) \\
m(u_t)
\end{bmatrix}
\]

where \( g(\cdot) \) is comprised of observables that only depend on the state, \( m(\cdot) \) contains observables that only depend on the action, and \( \ell(\cdot) \) contains observables that depend on both the state and action.

Given a state trajectory \( x_{1:T} \) and corresponding actions \( u_{1:T-1} \), the matrices \( \tilde{Y} \) and \( \tilde{Z} \) can be defined such that the columns of \( \tilde{Z} \) contain the observations \( h(x_t, u_t) \) at time steps \( t = 1, \ldots, T - 1 \) and the corresponding columns of \( \tilde{Y} \) contain the observations \( h(F(x_t, u_t), u_{t+1}) \). The form of the matrices can be written as:

\[
\tilde{Z} = \begin{bmatrix}
\tilde{Z}_x \\
\tilde{Z}_{xu} \\
\tilde{Z}_u
\end{bmatrix}, \quad
\tilde{Y} = \begin{bmatrix}
\tilde{Y}_x \\
\tilde{Y}_{xu} \\
\tilde{Y}_u
\end{bmatrix},
\]

where, for example, the columns of \( \tilde{Z}_x \) contain the observations \( g(x_1), \ldots, g(x_{T-1}) \). The matrices \( \tilde{Y} \) and \( \tilde{Z} \) will be the same as the matrices \( Y \) and \( Z \) from Section 5.2.1 under the condition that \( \tilde{Z}_x = X, \tilde{Z}_u = \Gamma, \tilde{Y}_x = Y, \) and \( \tilde{Z}_{xu} = \tilde{Y}_{xu} = \tilde{Y}_u = 0 \). In other words, DMDc assumes the existence of a finite-dimensional Koopman operator that can propagate linear observables of the state and control input, and has the form described by Eq. (5.46). As with extended dynamic mode decomposition, it can be preferable to consider more expressive observable functions, which is the focus of techniques presented in Chapters 6 and 7.
5.5 Discussion

This chapter reviewed methods for deriving linear dynamics models, even for nonlinear dynamical systems. Together with the control concepts from Chapter 4, the ideas from this chapter will serve as the foundation for the methods presented in the remainder of this thesis. The chapter began by presenting the DMD and DMDc algorithms, which are methods for obtaining linear dynamics models from data. DMD and DMDc perform modal decompositions of the dynamics matrices, allowing for efficient simulation and interpretable analysis of modeled systems.

Koopman theory was subsequently discussed, which posits the existence of an infinite-dimensional linear operator that advances observable functions forward in time. The conditions under which the Koopman operator becomes finite-dimensional were described, and parallels were drawn between DMD and Koopman theory. A key takeaway is that (extended) DMD will only yield accurate approximations to the Koopman operator if the selected observable functions are sufficiently expressive. The next chapter will outline a method for learning appropriate observables, and will demonstrate that this can enable the use of linear control techniques on systems with nonlinear dynamics.
Chapter 6

Learning Koopman Invariant Subspaces for Control

This chapter ties together ideas from Chapters 4 and 5, presenting a method for learning linear models that facilitate the application of linear control techniques to nonlinear dynamical systems. The following section provides relevant background material on learning-based control and fluid flow control. Subsequently, the Deep Koopman model is introduced, and its performance is evaluated on a benchmark flow control task.

6.1 Background

In recent years, the machine learning community has devoted a significant amount of attention to learning-based control of complex systems. In particular, the field of reinforcement learning (RL) has studied methods for automatically learning control policies that satisfy predefined objectives. Model-free RL approaches attempt to learn control policies without constructing explicit models for the environment dynamics, and have achieved impressive success in a variety of domains [107–109]. However, model-free methods require many interactions with an environment to learn effective policies, which may render them infeasible in safety-critical applications where environmental interaction is potentially dangerous, or in applications where data is difficult to obtain.
In contrast, model-based control approaches have the potential to learn effective controllers with far less data by first modeling the environment dynamics. Recent work has shown that learned dynamics model can enable more sample-efficient learning of effective controllers [110–115]. However, such success hinges on the need to construct accurate dynamics models.

The exact form of a dynamics model has strong implications for how easily the model can be incorporated into a control or planning framework. Neural networks are nonlinear functions, meaning neural dynamics models might not be well suited for many control methods designed for linear systems. For this reason, some recent approaches have sought to train neural networks to map states to a latent space where the dynamics can be evolved according to locally linear models that enable action selection through iLQR [116–118]. Koopman theory (see Chapter 5) offers an alternative viewpoint through which nonlinear dynamics can be mapped to linear dynamics. A number of recent works have sought to automatically learn Koopman observables using neural networks [98, 100, 103, 104, 119]. Furthermore, it has been shown that data-driven models that leverage Koopman theory can be used for control [99, 120, 121].

The methods presented in this chapter are ultimately applied to fluid flow control. The particular test case under study, the suppression of vortex shedding over a cylinder, has long served as a canonical test case for fluid flow control, and has been studied extensively experimentally and computationally, with the earliest experiments dating back to the 1960s [22, 23, 122–125]. These studies have shown that controller design can prove surprisingly difficult, as controller effectiveness is highly sensitive to flow conditions, measurement configuration, and feedback gains [123, 125]. In fact, the design of flow controllers in general represents a significant challenge due to the nonlinear nature of the Navier–Stokes equations. Nonetheless, at certain flow conditions it has been shown that vortex suppression can be achieved with a simple proportional control law based on a single sensor measurement in the cylinder wake. Thus, while the design of flow controllers may present considerable challenges, effective controllers may in fact prove relatively easy to implement.

The next section presents important insights relating to the discovery of Koopman
observable functions, ultimately leading to the introduction of the Deep Koopman model training procedure.

6.2 Learning Koopman Invariant Subspaces

As mentioned in Chapter 5, a procedure known as extended dynamic mode decomposition (eDMD) can be used to obtain a finite-dimensional approximation to the Koopman operator, which governs the time evolution of vector-valued observables \( g(x) \in \mathbb{R}^m \). The eDMD procedure constructs the matrices:

\[
\tilde{X} = \begin{bmatrix}
g(x_1) & g(x_2) & \cdots & g(x_{T-1}) \\
g(x_2) & g(x_3) & \cdots & g(x_T) \\
\vdots & \vdots & \ddots & \vdots \\
g(x_T) & g(x_1) & \cdots & g(x_{T-2})
\end{bmatrix}, \quad \tilde{Y} = \begin{bmatrix}
g(x_2) & g(x_3) & \cdots & g(x_T) \\
g(x_3) & g(x_4) & \cdots & g(x_1) \\
\vdots & \vdots & \ddots & \vdots \\
g(x_T) & g(x_1) & \cdots & g(x_{T-2})
\end{bmatrix},
\]

(6.1)

and performs linear least squares to obtain the matrix \( A = \tilde{Y} \tilde{X}^\dagger \), which serves as an approximation to the Koopman operator. The observable functions must be sufficiently expressive in order for this approximation to be accurate, and often must be hand-specified. Furthermore, for prediction and control it is desirable that the state is recoverable from the observations \( g(x) \), meaning that either the observable function is the identity mapping \( (g(x) = x) \), which may be insufficiently expressive, or the observable function is invertible. These aforementioned considerations motivate the use of a procedure that can automatically learn appropriate (and invertible) observable functions directly from data.

One property of Koopman eigenfunctions, and by extension observables composed from Koopman eigenfunctions, is that they span a Koopman invariant subspace. A Koopman invariant subspace is a subspace \( \mathcal{G} \) such that \( \mathcal{K} g \in \mathcal{G} \) for any \( g \in \mathcal{G} \) [98]. In other words, any function in \( \mathcal{G} \) remains in \( \mathcal{G} \) after being acted upon by the Koopman operator. The subspace span\{\( \varphi_1, \varphi_2, \ldots, \varphi_k \)\} is clearly a Koopman invariant subspace, and thus any subspace spanned by observables that are linear combinations of these eigenfunctions will also be a Koopman invariant subspace. Thus, the search for appropriate sets of observables \( \{g_1, g_2, \ldots, g_m\} \) can be reduced to a search for a set of
observables that span a Koopman invariant subspace.

Takeishi et al. [98] showed that the task of finding a set of observables that span a Koopman invariant subspace reduces to finding a state mapping \( g(\cdot) \) under which the eDMD linear least-squares procedure performs well, i.e. the loss:

\[
\mathcal{L}_{LS} = \left\| \tilde{Y} - \left( \tilde{Y} \tilde{X}^\dagger \right) \tilde{X} \right\|_F^2
\]  

(6.2)

is minimized. To practically implement this procedure, they proposed learning the state mapping with a neural network trained to minimize the loss:

\[
\mathcal{L} = \mathcal{L}_{LS} + \lambda \mathcal{L}_{inv},
\]

(6.3)

where \( \lambda \) is a scalar parameter and \( \mathcal{L}_{inv} \) is a loss term that enforces that the learned mapping is invertible. Further details about (a modified version of) this algorithm are provided in the following subsection.

### 6.2.1 Deep Koopman Model

The Deep Koopman model is now presented, which employs a modified form of the training algorithm proposed by Takeishi et al. to learn state mappings that approximately span a Koopman invariant subspace. The training algorithm is depicted in Fig. 6.1. First, a sequence of time snapshots \( x_{1:T} \) is used to construct the matrices \( X \) and \( Y \) defined in Eq. (5.4). These matrices are fed into an encoder neural network, which serves as the mapping \( g(x_t) \) and produces matrices \( \tilde{X} \) and \( \tilde{Y} \) defined in Eq. (6.1). Subsequently, a linear least-squares fit is performed to find an \( A \)-matrix that can propagate the state mappings forward in time. Finally, \( \tilde{X} \) and the propagated state mappings are fed into a decoder that functions as \( g^{-1}(\cdot) \) to yield the matrices \( \hat{X} \) and \( \hat{Y} \), approximations to \( X \) and \( Y \).

The Deep Koopman model is trained to minimize

\[
\mathcal{L} = \|X - \hat{X}\|_F^2 + \|Y - \hat{Y}\|_F^2,
\]

(6.4)
where $\hat{Y}$ is obtained by running $\tilde{Y}_{\text{pred}}$ through the decoder. Minimizing the error between $X$ and $\hat{X}$ enforces that the mapping $g(\cdot)$ is invertible and prevents the learning of trivial mappings such as $g(x) = 0$ for all $x$. Minimizing the error between $Y$ and $\hat{Y}$ enforces that the derived dynamics model can accurately simulate the time evolution of the system in the space of observations $g(x)$. One main difference between this algorithm and the one proposed by Takeishi et al. is that the model is forced to simulate the time evolution of the system during training in a manner that mirrors how the model will be deployed at test time. In particular, the derived $A$-matrix is applied recursively to the observation $g(x_1)$ to produce the matrix $\tilde{Y}_{\text{pred}}$ defined in Fig. 6.1, which is then mapped to $\hat{Y}$ through the decoder.

To better clarify another new feature of the proposed algorithm, it is worth drawing a distinction between reconstruction and prediction. If a dynamics model is constructed based on a sequence of states $x_{1:N}$, then simulations generated by the dynamics model would be reconstructing the already observed time evolution of the system for all time steps $t \leq N$ and predicting the time evolution of the system for all steps $t > N$. For control purposes, it is desirable to train a dynamics model that can ultimately be used to predict the future evolution of a given system. Thus, during training, the $A$-matrix is generated based on only the first $T/2$ entries of $X$ and $\hat{Y}$, thereby enforcing that the last $T/2$ entries of $\tilde{Y}_{\text{pred}}$ are purely predictions for how the system will evolve in time.
One of the advantages of this approach is its relative simplicity, as the neural network architecture is equivalent to that of a standard autoencoder. The dynamics matrix $A$ does not need to be modeled directly; rather, it is derived by performing least squares on the learned state mappings. The encoder and decoder neural networks can take the form of fully connected or convolutional neural networks depending on the form of input data. The entire model can be trained end-to-end to learn suitable state mappings $g(x_t)$.

The Deep Koopman model is now evaluated based on its ability to learn the dynamics of a system widely studied in the fluids literature: the two-dimensional flow of air over a cylinder. The following section provides more details about the particular test case.

### 6.3 Test Case

The system under consideration is a two-dimensional circular cylinder at Reynolds number 50 and an effectively incompressible Mach number of 0.2. This is a well studied test case [126, 127], which has been used for both validation purposes and as a legitimate research case in its own right. The chosen Reynolds number is just above the cut-off for laminar flow and thus results in the formation of a von Karman vortex street, where vortices are shed from the upper and lower surface of the cylinder in a periodic fashion.

The PyFR solver [29] is used to perform the fluid flow simulations. A computational domain of $[-30, 50]$ and $[-30, 30]$ is used in the stream- and cross-wise directions, respectively. The cylinder is centered at $(0, 0)$ with a diameter of one. The surface of the cylinder is modeled as a no-slip isothermal wall boundary condition, and Riemann invariant boundary conditions are applied at the far-field. The full mesh, depicted in the left figure of Fig. 6.2, consists of 5672 unstructured, quadratically curved quadrilateral elements. All simulations are run using quadratic solution polynomials and an explicit fourth-order Runge-Kutta time stepping scheme.
6.4 Modeling Unforced Dynamics

This section presents the results of modeling the unforced dynamics of the cylinder system, i.e. the dynamics in the absence of any control inputs. First, the training procedure for the Deep Koopman model is discussed, after which the baseline models are introduced. Finally, the results of numerical experiments are presented.

6.4.1 Training Details

A training set for learning the unforced dynamics of the cylinder system is constructed by saving time snapshots every 1500 solver steps. As in Section 3.3.1, the data is formatted into image-like inputs before storage. Solution quantities are sampled from a $128 \times 256$ grid of roughly equispaced points that span the region of fine mesh resolution in the right image of Fig. 6.2. Each grid point contains four channels corresponding to four physical quantities in the flow at that point: density, $x$-momentum, $y$-momentum,
and energy. An example snapshot can be found in Fig. 6.3, illustrating the qualitative differences between the four distinct input channels.

The Deep Koopman model is implemented and trained in Tensorflow. The encoder consists of five two-dimensional convolutional layers with ResNet skip connections. The layers of the encoder contain 256, 128, 64, 32, and 16 filters, respectively, and the observations $g(x_t)$ are set to be 32-dimensional. The decoder is constructed to invert all operations performed by the encoder. $L_2$ regularization is applied to all encoder and decoder weights. The Deep Koopman model is trained using the Adam optimizer with an initial learning rate of $7.5 \times 10^{-4}$, which is cut in half each time the loss fails to decrease on a validation set.

6.4.2 Baseline Models

The Deep Koopman model is compared against the Deep Variational Bayes Filter (DVBF) [117] to baseline performance. The Deep Variational Bayes Filter can be viewed as a form of state space model, which seeks to map high-dimensional inputs $x_t$ to lower-dimensional latent states $z_t$ that can be evolved forward in time. The DVBF is a recently proposed approach that improves upon previous state space models (e.g. Embed to Control [116]) and evolves latent states forward linearly in time, and thus serves as a suitable performance benchmark for the Deep Koopman model.

The same autoencoder architecture employed by the Deep Koopman model is used to perform the forward and inverse mappings between the inputs $x_t$ and the latent states $z_t$. As with the Deep Koopman model, the inputs are time snapshots from CFD simulations. The time evolution of the latent states is described by

$$z_{t+1} = A_t z_t + B_t u_t + C_t w_t,$$

(6.5)

where $u_t$ is the control input at time $t$ and $w_t$ represents the process noise. The matrices $A_t$, $B_t$, and $C_t$ are assumed to comprise a locally linear dynamics model, and are determined at each time step as a function of the current latent state and control input. The DVBF implementation used for these experiments ignores the effect of control inputs because it is only used to model the unforced dynamics of fluid flow.
In addition to the DVBF, the Deep Koopman model is also benchmarked against a model trained using the procedure proposed by Takeishi et al., which sets $\tilde{Y}_{\text{pred}} = A\tilde{X}$ rather than calculating $\tilde{Y}_{\text{pred}}$ by recursively applying $A$ to $g(x_1)$. Each model is trained on 32-step sequences of data extracted from two-dimensional cylinder simulations. The trained models are then used to recreate the time evolution of the system over 20 test sequences and the error is extracted over time. For a fair comparison, $g(x_t)$ and $z_t$ are defined to be 32-dimensional vectors. The Koopman method constructs its dynamics model based on state mappings from the first 16 time steps, then simulates the system for all time steps using the derived $A$-matrix. The Takeishi baseline derives its $A$-matrix based on state mappings from the first 32 time steps. The DVBF constructs a new locally linear dynamics model at each time step, but relies on information from the first 32 time steps to sample $w_0$, the initial value of the process noise.

### 6.4.3 Results

The results of these experiments can be found in Fig. 6.4a and Fig. 6.4b, where the error metric is the relative error, defined as the $L_1$-norm of the prediction error normalized by the $L_1$-norm of the ground-truth solution. Figure 6.4a shows that the error for the Takeishi baseline initially grows more rapidly than the error for the
other models. This illustrates the importance of training models to generate recursive predictions, since models trained to generate single-step predictions tend to generate poor multi-step predictions due to prediction errors compounding over time [128]. Over a 32-step horizon the Deep Koopman and DVBF models perform comparably, with the error for the Koopman model rising slightly at later time steps as it begins generating predictions for states that it did not have access to in constructing its dynamics model. A much starker contrast in model performance can be observed in Fig. 6.4b, where the Deep Variational Bayes Filter begins to rapidly accumulate error once it surpasses the 32-step horizon.

For the results shown in Fig. 6.4b, the Deep Variational Bayes Filter is performing reconstruction for 32 steps and prediction for 32 steps. Hence, it is clear that the DVBF is effective at reconstruction, but is unable to function stably in prediction on this task. In contrast, the Deep Koopman model, aided by its ability to construct state mappings that approximately span an invariant subspace, is able to generate stable predictions for much longer time horizons. In fact, while the prediction error of the Koopman model does grow with time, its mean prediction error remains less than 0.2% over a horizon of 128 time steps, corresponding to approximately eight periods of vortex shedding. Given the goal of obtaining a predictive dynamics model that can be incorporated into a control framework, these results provide confidence that the Deep Koopman model is well suited for the task.

6.5 Modeling Forced Dynamics

Section 6.4 explained how Deep Koopman models can learn state mappings that are suitable for modeling unforced dynamics. In accounting for control inputs, the goal is instead to construct a linear dynamics model of the form

$$g(x_{t+1}) = Ag(x_t) + Bu_t,$$  \hspace{1cm} (6.6)
where \( u_t \in \mathbb{R}^p \) and \( B \in \mathbb{R}^{m \times p} \). Hence, defining the matrix
\[
\Gamma = \begin{bmatrix}
\vdots & \vdots & \ddots & \vdots \\
\mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_{T-1} \\
\vdots & \vdots & \ddots & \vdots
\end{bmatrix},
\]
(6.7)

an accurate dynamics model would approximately satisfy
\[
\tilde{Y} = A\tilde{X} + B\Gamma.
\]
(6.8)

As discussed in Section 5.2, Proctor et al. presented several methods for estimating \( A \) and \( B \) given matrices \( \tilde{X} \) and \( \tilde{Y} \) [95]. In this chapter, \( B \) is treated as a known quantity, which means \( A \) can be estimated through a linear least-squares fit
\[
A = (\tilde{Y} - B\Gamma)\tilde{X}^\dagger.
\]
(6.9)

Chapter 7 will describe an alternative least-squares procedure for jointly estimating \( A \) and \( B \) in the event that \( B \) is treated as unknown.

Thus, the Deep Koopman training algorithm presented in Section 6.4 can be modified such that \( A \) is generated through Eq. (6.9). While \( B \) is treated as a known quantity in this formulation, in reality it is another parameter that must be estimated. This is accounted for by defining a global \( B \)-matrix that is treated as a neural network parameter, and is optimized through gradient descent concurrently with the training of the Deep Koopman model parameters.

### 6.5.1 Modified Test Case

With the ability to train Deep Koopman models that account for control inputs, a modified form of the test case presented in Section 6.3 is now considered that allows for a scalar control input to affect the fluid flow. In particular, the simulation is modified so that the cylinder can rotate with a prescribed angular velocity, as illustrated in Fig. 6.5. Cylinder rotation is modeled by applying a spatially varying velocity to the surface of the wall, thus enabling the grid to remain static. The angular velocity
Figure 6.5: Illustration of actuation used to suppress vortex shedding.

is allowed to vary every 1500 solver steps, with the value held constant during all intervening steps.

6.5.2 Training Details

Deep Koopman models are trained on data from the modified test case in order to learn the forced dynamics of the cylinder system. A training dataset is collected by simulating the system with time-varying angular velocity. Every 1500 solver steps, a time snapshot $x_t$ is stored and the control input $u_t$ is altered. In total, the training set contains 4238 snapshots of the system. These snapshots are then divided into 1600 staggered 32-step sequences for training the Deep Koopman model. As in the case of unforced dynamics, dynamics models are constructed based on information from the first 16 time steps, but the system is simulated for 32 time steps during training. Training a single model takes approximately 12 hours on a Titan X GPU.

The control inputs applied to the system while generating the training data can be found in Fig. 6.6. Analogous to frequency sweeps in system identification [129], the system is subjected to sinusoidal inputs with a frequency that varies linearly with time. These sinusoidal inputs are interspersed with periods with no control inputs to allow the model to learn the unforced system dynamics from different initial conditions.

6.6 Model Predictive Control

The quality of the learned Deep Koopman models is evaluated by studying their ability to enable effective control of the cylinder system. In particular, the learned dynamics models are incorporated into a control framework with the goal of suppressing vortex...
shedding. Vortex suppression can be framed as an optimal control problem, with the goal of selecting control inputs $u_{1:H-1}$ that minimize the cost function:

$$C(x_{1:H}, u_{1:H-1}) = c_H(x_H) + \sum_{t=1}^{H-1} c_t(x_t) + u_t^T R u_t$$

subject to the constraint that the states evolve according to the Navier–Stokes equations. The term $c_t(x_t)$ represents the instantaneous cost associated with a given flow field that measures the distance from the desired steady flow. Solving a general optimization problem of this form can be quite challenging, since the cost function and dynamics constraints are in general nonlinear and potentially nonconvex.

An alternative optimal control formulation takes the form:

$$C(x_{1:H}, u_{1:H-1}) = (g_H - g_{\text{goal}})^T Q (g_H - g_{\text{goal}})$$

$$+ \sum_{t=1}^{H-1} (g_t - g_{\text{goal}})^T Q (g_t - g_{\text{goal}}) + u_t^T R u_t$$

subject to $g_1 = g(x_1)$ and $g_{t+1} = Ag_t + Bu_t$. Here, $A$ and $B$ represent the matrices generated by the Deep Koopman model that describe the time evolution of observations $g(x_t)$, and $g_{\text{goal}}$ represents $g(x_{\text{goal}})$, the observation associated with the goal flow state corresponding to steady flow. As shown in Section 4.2, the linear quadratic regulator can efficiently solve a problem of this form in order to determine an optimal control law. If control saturation limits are imposed (e.g. $\|u_t\|_\infty \leq u_{\text{max}}$), the optimal control problem remains a quadratic program that can still be solved efficiently with convex
solvers such as CVXPY [130].

While appealing, there are potential drawbacks associated with formulating the control problem in this manner. First, this cost function requires knowledge of \( x_{\text{goal}} \), the desired flow state the controller is trying to reach, which may be known qualitatively (e.g. a flow with no vortex shedding) but not quantitatively (the exact value of the flow quantities at each point in space). Additionally, it assumes that the state cost \( c_t(x_t) \) can be well approximated by a quadratic cost of the form \((g_t - g_{\text{goal}})^T Q (g_t - g_{\text{goal}})\). Finally, it assumes the matrices \((A, B)\) exactly describe the system dynamics, which means that controller effectiveness may be sensitive to modeling inaccuracies.

In the subsequent experiments, \( x_{\text{goal}} \) is replaced by a snapshot of the steady flow observed when the cylinder system is simulated at a Reynolds number of 45, which is a sufficiently low Reynolds number that vortex shedding does not occur. While the flow at this lower Reynolds number, denoted by \( x_{45} \), will be different from the steady flow at a Reynolds number of 50, the assumption is that the difference between the corresponding observations, \( \|g(x_{45}) - g(x_{\text{goal}})\|_2 \), will be small. This assumption, along with the assumption that \( c_t(x_t) \) can be approximated by a quadratic function of the observations \( g(x_t) \), may not be justified for general problems, but the results presented in Section 6.7 will demonstrate that these approximations can indeed provide reasonable cost estimates for the vortex suppression task.

To combat problems that may arise due to modeling inaccuracies with the derived \( A \) and \( B \) matrices, the above optimization procedure is embedded within a model predictive control (MPC) loop. At each time step, the Deep Koopman model generates observations for the previous 16 time steps, and uses those observations in conjunction with the global \( B \)-matrix to find a suitable \( A \)-matrix for propagating the observations forward in time. The optimization problem described by Eq. (6.11) is then solved with horizon \( H = 16 \) to determine a sequence of optimal control inputs \( \mathbf{u}_{1:16}^* \). Rather than executing each of these actions in an open-loop fashion, only the first action \( \mathbf{u}_1^* \) is passed to the CFD solver, which advances the cylinder system forward in time. This MPC procedure allows for closed-loop control that is more robust to modeling errors.
Figure 6.7: Snapshots of $x$-momentum over time as the MPC algorithm attempts to suppress vortex shedding.

6.7 Results

This section presents the results of performing model predictive control on the vortex shedding problem. To measure the effectiveness of the derived controller, a measure of closeness to the desired outcome is needed; in this case, the desired outcome is to achieve a steady laminar flow devoid of vortex shedding. As discussed in Section 1.3, the Navier–Stokes equations are a conservation law, taking the form:

$$\frac{\partial \mathbf{q}}{\partial t} = -\nabla \cdot f(\mathbf{q}, \nabla \mathbf{q}).$$ \hspace{1cm} (6.12)

In the process of running CFD simulations, PyFR evaluates the right-hand side of this equation by calculating residuals. Note that if a steady flow is achieved, the time derivative will be zero and in turn the residuals will be zero. Thus, residual values can be used as a measure of closeness to the desired steady flow. In the subsequent results, the presented residual values represent the norm of the $x$- and $y$-momentum residuals over time.

In solving the optimization problem given in Eq. (6.11), the $Q$-matrix is set to be the identity matrix and $R$ is set to a value of $2 \times 10^5$, which accounts for the fact that $\|\mathbf{g}_t - \mathbf{g}_{\text{goal}}\|_2$ is typically orders of magnitude larger than $\|\mathbf{u}_t\|_2$, and discourages actions that are too extreme for the Deep Koopman model to handle accurately.
Figure 6.8: Scaled residuals plotted alongside estimated cost used for MPC action selection.

Figure 6.9: Control inputs selected by MPC plotted along with scaled y-velocity measurements.

Results from the model predictive control experiments can be found in Fig. 6.7 and Fig. 6.8. Figure 6.7 provides a qualitative picture of the effectiveness of the applied control, as the cylinder wake exhibits a curved region of low x-momentum characteristic of vortex shedding at early time steps, then flattens out to a profile more characteristic of steady flow over time. Figure 6.8 provides a quantitative picture of the controller performance, showing that the controller brings about a monotonic decrease in the x- and y-momentum residuals over time. Additionally, Fig. 6.8 plots a scaled version of $\|g(x_t) - g(x_{45})\|_2$. It is apparent that there is a strong correspondence between a decrease in the residuals and a decrease in the quadratic cost that model predictive control is attempting to minimize. This provides confidence that using this measure of cost in MPC is sensible for this problem.
Figure 6.10 shows the control inputs applied to the system over time. A dynamics model \((A, B)\) cannot be constructed until 16 states have been observed, so the control inputs are initially set to zero. Subsequently, the inputs appear to vary sinusoidally and decrease in amplitude over time. Remarkably, it is possible to find a location in the wake of the cylinder, at a point denoted by \(d^*\) in Fig. 6.10, where the variations in \(y\)-velocity are in phase with the selected control inputs. When scaled by a constant value of 0.4, a strong overlap can be observed between the control inputs and velocity values. Viewed in this light, the control inputs selected by model predictive control become quite interpretable, and are functionally similar to a proportional controller performing feedback control based on \(y\)-velocity measurements at \(d^*\) with a gain of 0.4.

Motivated by the insights gained from studying the MPC inputs, it is worth studying the effectiveness of a simple proportional control scheme in suppressing vortex shedding. Rather than selecting inputs through model predictive control, the angular velocity is set by applying a gain of 0.4 to measurements of the \(y\)-velocity at point \(d^*\) in the cylinder wake. While easy to implement, further experiments are performed to illustrate that such a control law is not easy to find. In these experiments, proportional control is attempted with the same gain based on measurements at two additional locations, \(\frac{1}{2}d^*\) and \(2d^*\), as illustrated in Fig. 6.10.

The results of these experiments, summarized in Fig. 6.11, demonstrate that proportional control based on measurements at \(d^*\) is indeed effective at suppressing vortex shedding. Meanwhile, proportional control laws based on measurements at the other locations are unable to drive the system closer to the desired steady flow. These results are in agreement with previous studies [123], which show that the effectiveness of proportional control is highly sensitive to measurement location.
6.8 Discussion

This chapter introduced a method for training Deep Koopman models, demonstrating that the learned models are capable of stably simulating airflow over a cylinder for significant prediction horizons. Furthermore, modifications to the training algorithm were described that allow the Koopman models to account for control inputs, and thereby be applied to control tasks. Learning sufficiently accurate dynamics models from approximately 4000 training examples, the method is very sample efficient, which is of high importance due to the large computational cost associated with CFD simulations. Most importantly, by incorporating the Deep Koopman models into a model predictive control framework for vortex suppression, it was demonstrated that the resulting control law is both interpretable and sensible, aligning with well studied flow control approaches from the literature.

This chapter has shown the success of this approach for one flow condition and one flow control problem; future studies should investigate the general applicability of this approach. Further, stronger forms of actuation that are likely to scale to higher Reynolds numbers should be considered for the vortex suppression task, as most previous studies have used blowing and suction on the cylinder surface as the control input rather than rotation [22, 23, 122–125]. To avoid the need to train different models at each flow condition, the Deep Koopman training procedure should be extended to allow for parameterization, similar to the models in Chapter 3.

One shortcoming of the Deep Koopman models is that they only provide point
estimates rather than distributions in their predictions. Thus, in scenarios where
dynamics modeling is challenging or data is limited, inaccuracies in the learned models
can lead to diminished control performance. For this reason, it can be advantageous
to learn models that are capable of representing a range of plausible outcomes for how
a modeled system may advance in time. The next chapter introduces an extension to
the Deep Koopman models that allows for uncertainty-aware dynamics modeling and
control.
Chapter 7

Uncertainty-Aware Dynamics Modeling and Control

Learned dynamics models are often imperfect, especially when data is limited. These imperfections can present a challenge for control; if a learned model is inaccurate, it may provide incorrect information about how to best reach a desired state. Thus, controller effectiveness can be enhanced if learned dynamics models also provide a notion of uncertainty in their predictions. In fact, learning-based control approaches that account for model uncertainty have been shown to attain strong performance on a variety of tasks while requiring less environmental interaction than state-of-the-art model-free approaches [110, 114].

This chapter introduces the Deep Variational Koopman (DVK) model, a method for inferring Koopman observations that can be propagated linearly in time. The method requires the training of a single neural network model, but enables the sampling of an ensemble of linear dynamics models in the space of observations. Taken together, this model ensemble effectively provides a distribution over the system dynamics. The chapter begins by deriving the DVK model objective function and presenting a tractable training algorithm for optimizing a lower bound on that objective. Subsequently, DVK models are evaluated on a series of benchmark tasks in dynamics modeling and control.
7.1 Deep Variational Koopman Models

This chapter considers Koopman observables of the form

\[ h(x_t, u_t) = g(x_t) + Lu_t, \]  

(7.1)

where \( g(x_t) \) represents an observation of the state \( x_t \) and \( L \in \mathbb{R}^{(1 \times p)} \) is a matrix. Under the assumption of a finite-dimensional Koopman operator defined by Eq. (5.46), and defining the vector-valued observables \( h = [h_1, \ldots, h_m]^T \) and \( g = [g_1, \ldots, g_m]^T \), the update equation:

\[ g(x_{t+1}) = Kh(x_t, u_t) = \begin{bmatrix} A & B \end{bmatrix} \begin{bmatrix} g(x_t) \\ u_t \end{bmatrix} \]

(7.2)

\[ = Ag(x_t) + Bu_t \]

can be derived. Equation (7.2) describes the forward-time evolution of the observations \( g(x_t) \); if \( A \) is invertible the reverse-time evolution can likewise be described as:

\[ g(x_t) = A^{-1}(g(x_{t+1}) - Bu_t). \]

(7.3)

Consider a sequence of control inputs \( u_{1:T-1} \) applied to a system with a finite-dimensional Koopman operator, resulting in a sequence of states \( x_{1:T} \). As in previous chapters, define the matrices \( \tilde{X} \in \mathbb{R}^{m \times (T-1)} \) and \( \tilde{Y} \in \mathbb{R}^{m \times (T-1)} \) as:

\[ \tilde{X} = \begin{bmatrix} g(x_1) & g(x_2) & \cdots & g(x_{T-1}) \end{bmatrix}, \quad \tilde{Y} = \begin{bmatrix} g(x_2) & g(x_3) & \cdots & g(x_T) \end{bmatrix}. \]  

(7.4)

Additionally, define the matrices \( \Gamma \in \mathbb{R}^{p \times (T-1)} \) and \( \tilde{Z} \in \mathbb{R}^{(m+p) \times (T-1)} \) as:

\[ \Gamma = \begin{bmatrix} u_1 & u_2 & \cdots & u_{T-1} \end{bmatrix}, \quad \tilde{Z} = \begin{bmatrix} \tilde{X} \\ \Gamma \end{bmatrix}. \]

(7.5)
Under the assumptions outlined above, $A$ and $B$ can be recovered through:

$$
[A \ B] = \bar{Y} \bar{Z}^\dagger,
$$

(7.6)

where $\bar{Z}^\dagger$ is the Moore-Penrose pseudoinverse of $\bar{Z}$. Unfortunately, as discussed in Chapter 5, the true form of the observables is generally unknown. The following section will describe a method for inferring a sequence of observations $g(x_1), \ldots, g(x_T)$ based on the time evolution of a dynamical system subject to a sequence of control inputs. Note that the value of the observations are inferred directly, and no attempt is made to model the observable functions $g(\cdot)$.

### 7.1.1 Inference Procedure

Consider a system subjected to a sequence of control inputs $u_{1:T-1}$, causing it to traverse a set of states $x_{1:T}$. It is assumed there exist observations of the states $g(x_t)$ such that the system can be simulated linearly in time as outlined in Eq. (7.2) and Eq. (7.3). Furthermore, it is assumed that the observed sequence is sufficiently long such that, when the matrices $\bar{Z}$ and $\bar{Y}$ are formed, the true $A$ and $B$ matrices can be found through Eq. (7.6) and $A^{-1}$ can be found through:

$$
A^{-1} = (\bar{Y} - B\Gamma)\bar{X}^\dagger.
$$

(7.7)

Let $g_t$ be a latent variable representing the observation $g(x_t)$. Additionally, let $z_{1:T}$ be a set of latent variables that enforce that multi-step predictions made with the derived dynamics model allow for accurate reconstructions of the states $x_{1:T}$. Because prediction error would be expected to grow with time, the $z_t$’s are simulated backward in time such that the lowest reconstruction error is generally obtained at time $T$, which in turn will allow for more accurate predictions for how a system will evolve in future, unobserved time steps. The values of $z_{1:T}$ can be determined through:

$$
z_T = g_T, \quad z_t = A^{-1}(z_{t+1} - Bu_t).
$$

(7.8)
Finally it is assumed that $x_t = g^{-1}(z_t)$, i.e. the states $x_t$ are generated by inverting the observable function $g(\cdot)$. Figure 7.1 shows the graphical model for this problem.

The goal is to train a model that maximizes the likelihood assigned to a sequence of observed states $x_{1:T}$ conditioned on actions $u_{1:T-1}$. This modeled density must account for the presence of latent variables $g_{1:T}$ and $z_{1:T}$. The expression for the likelihood can be written in terms of the observed and latent variables as:

$$p(x_{1:T} \mid u_{1:T-1}) = \int p(g_{1:T}, z_{1:T}, x_{1:T} \mid u_{1:T-1}) \, dg_{1:T} \, dz_{1:T}. \quad (7.9)$$

By the chain rule, the integrand can be factored into:

$$p(g_{1:T}, z_{1:T}, x_{1:T} \mid u_{1:T-1}) = p(g_{1:T} \mid u_{1:T-1}) \times p(z_{1:T} \mid g_{1:T}, u_{1:T-1}) \, p(x_{1:T} \mid g_{1:T}, z_{1:T}, u_{1:T-1}). \quad (7.10)$$

Each term can be simplified using the conditional independence assumptions encoded by the graphical model. The first term can be simplified to:

$$p(g_{1:T} \mid u_{1:T-1}) = p(g_1) \prod_{t=2}^{T} p(g_t \mid g_{t-1}, u_{t-1}). \quad (7.11)$$

Each element of the above expression can be thought of as a (conditional) prior over
an observation at time $t$.

The second term in the integrand describes the distribution over variables $z_{1:T}$, whose values can be determined exactly using Eq. (7.8) if $g_{1:T}$ and $u_{1:T-1}$ are known. Thus, it is found that:

$$p(z_{1:T} \mid g_{1:T}, u_{1:T-1}) = \delta(z_T \mid g_T) \prod_{t=1}^{T-1} \delta(z_t \mid z_{t+1}, g_{1:T}, u_{1:T-1}),$$  (7.12)

where $\delta(\cdot \mid \cdot)$ represents a deterministic relationship. From the structure of the graphical model, the last term becomes:

$$p(x_{1:T} \mid g_{1:T}, z_{1:T}, u_{1:T-1}) = \prod_{t=1}^{T} p(x_t \mid z_t).$$  (7.13)

Even with these simplifications, evaluating the likelihood expression is generally intractable because it requires marginalizing over the latent variables. Therefore, instead of optimizing this objective directly, variational inference is used to optimize a lower bound on the likelihood. As in Section 3.2, an approximation to the true posterior distribution is introduced, given by $q(g_{1:T}, z_{1:T} \mid x_{1:T}, u_{1:T-1})$. Multiplying and dividing the likelihood expression by this quantity, taking the logarithm of both sides, and invoking Jensen’s inequality, the following lower bound is found for the log-likelihood:

$$\log p(x_{1:T} \mid u_{1:T-1}) \geq \mathbb{E}_{z_{1:T} \sim q} \left[ \sum_{t=1}^{T} \log p(x_t \mid z_t) \right] + \mathbb{E}_{g_{1:T} \sim q} \left[ \log p(g_1) \right]$$

$$+ \mathbb{E}_{g_{1:T}, z_{1:T} \sim q} \left[ \sum_{t=2}^{T} \log p(g_t \mid g_{t-1}, u_{t-1}) - \log q(g_{1:T}, z_{1:T} \mid x_{1:T}, u_{1:T-1}) \right].$$  (7.14)

As with the integrand in Eq. (7.9), the expression for the approximate posterior distribution can be simplified using the chain rule and conditional independence relationships. First, the approximate posterior can be rewritten as:

$$q(g_{1:T}, z_{1:T} \mid x_{1:T}, u_{1:T-1}) = q(z_{1:T} \mid g_{1:T}, x_{1:T}, u_{1:T-1}) q(g_{1:T} \mid x_{1:T}, u_{1:T-1}).$$  (7.15)
Given knowledge of $g_{1:T}$ and $u_{1:T-1}$, $z_{1:T}$ can be inferred exactly. Thus, $q(z_{1:T} \mid g_{1:T}, x_{1:T}, u_{1:T-1})$ becomes:

$$q(z_{1:T} \mid g_{1:T}, x_{1:T}, u_{1:T-1}) = \delta(z_T \mid g_T) \prod_{t=1}^{T-1} \delta(z_t \mid z_{t+1}, g_{1:T}, u_{1:T-1}).$$ \hspace{1cm} (7.16)$$

Additionally, $q(g_{1:T} \mid x_{1:T}, u_{1:T-1})$ can be factorized as:

$$q(g_{1:T} \mid x_{1:T}, u_{1:T-1}) = q(g_1 \mid x_{1:T}, u_{1:T-1}) \prod_{t=2}^{T} q(g_t \mid g_{1:t-1}, x_{1:T}, u_{1:T-1}).$$ \hspace{1cm} (7.17)$$

Combining the above expressions and taking the logarithm of both sides yields:

$$\log q(g_{1:T}, z_{1:T} \mid x_{1:T}, u_{1:T-1}) = \log q(g_1 \mid x_{1:T}, u_{1:T-1})$$

$$+ \sum_{t=2}^{T} q(g_t \mid g_{1:t-1}, x_{1:T}, u_{1:T-1}).$$ \hspace{1cm} (7.18)$$

Incorporating this into Eq. (7.14), the following expression is found for the lower bound on the log-likelihood objective:

$$\log p(x_{1:T} \mid u_{1:T-1}) \geq \mathbb{E}_{z_{1:T} \sim q} \left[ \sum_{t=1}^{T} \log p(x_t \mid z_t) \right]$$

$$- \mathcal{D}_{KL} [q(g_1 \mid x_{1:T}, u_{1:T-1}) \mid\mid p(g_1)]$$

$$- \sum_{t=2}^{T} \mathcal{D}_{KL} [q(g_t \mid g_{1:t-1}, x_{1:T}, u_{1:T-1}) \mid\mid p(g_t \mid g_{t-1}, u_{t-1})].$$ \hspace{1cm} (7.19)$$

This lower bound is comprised of the likelihood of the observed states $x_{1:T}$ given $z_{1:T}$, as well as the KL-divergence between the approximate posterior and the (conditional) prior distributions over the observations $g_t$.

The maximization of the likelihood terms and the minimization of the KL-divergence terms can often be viewed as competing objectives. As in other approaches in variational inference such as the $\beta$-VAE [131], it is possible to introduce a scalar parameter that governs the tradeoff between these objectives. Assigning too low of
a weight to the KL-divergence objective can lead to small variance in the sampled models, which can negate many of the advantages of employing model ensembles. On the other hand, a heavy prioritization of the KL-divergence objective can inhibit control performance because the sampled models contain no useful signal about the dynamics of the modeled system. In the experiments detailed in Section 7.3, it was found that applying a weight of 0.1 to the KL-divergence components encoded a reasonable tradeoff between these objectives.

The following section provides a practical training procedure for maximizing the lower bound in Eq. (7.19).

7.1.2 Optimization Procedure

The expectation in the derived lower bound can be estimated through Monte Carlo sampling. To raise the lower bound, the parameters of six neural networks are simultaneously optimized, which together comprise the Deep Variational Koopman (DVK) model. As in Section 3.2, it is assumed that the prior and decoder networks are defined by parameters $\theta$ and the approximate posterior networks are defined by parameters $\phi$. The size and architecture of each neural network, which is held constant across all experiments, is listed in Table 7.1 (e.g. 64, 64 represents a two-layer neural network with 64 neurons in each layer).

1. The **Decoder Network** outputs $\mu_t$, the mean of a Gaussian distribution over state $x_t$ given $z_t$, represented by $p_\theta(x_t \mid z_t)$. The distribution over $x_t$ is assumed to have constant covariance. Hence, maximizing the log-likelihood is equivalent to minimizing the square error between $\mu_t$ and $x_t$.

2. The **Temporal Encoder Network** is an LSTM that maps a sequence of states $x_{1:T}$ and actions $u_{1:T-1}$ to a low-dimensional encoding that summarizes the system time evolution.

3. The **Initial Observation Inference Network** outputs the parameters to the approximate posterior distribution over observation $g_1$ given the output of the temporal encoder, represented by $q_\phi(g_1 \mid x_{1:T}, u_{1:T-1})$. 

Table 7.1: Neural Network Summary

<table>
<thead>
<tr>
<th>Network</th>
<th>Type</th>
<th>Number of Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decoder</td>
<td>Feedforward</td>
<td>64, 32</td>
</tr>
<tr>
<td>Temporal Encoder</td>
<td>LSTM</td>
<td>64</td>
</tr>
<tr>
<td>Initial Observation Inference</td>
<td>Feedforward</td>
<td>64</td>
</tr>
<tr>
<td>Observation Encoder</td>
<td>LSTM</td>
<td>64</td>
</tr>
<tr>
<td>Observation Inference</td>
<td>Feedforward</td>
<td>64, 64</td>
</tr>
<tr>
<td>Conditional Prior</td>
<td>Feedforward</td>
<td>64, 32</td>
</tr>
</tbody>
</table>

4. The Observation Encoder Network is a recurrent neural network that takes in observations $\mathbf{g}_{1:T-1}$ and outputs an encoding describing their time evolution. The encoding is updated iteratively as more observations are sampled.

5. The Observation Inference Network outputs the parameters to the approximate posterior distribution over observation $\mathbf{g}_t$ given the output of the Temporal and Observation Encoder Networks, represented by $q_\phi(\mathbf{g}_t | \mathbf{g}_{1:t-1}, \mathbf{x}_{1:T}, \mathbf{u}_{1:T-1})$.

6. The Conditional Prior Network outputs the parameters to a Gaussian conditional prior distribution over observation $\mathbf{g}_t$. The output distribution is conditioned on the previous observation and action, and is represented by $p_\theta(\mathbf{g}_t | \mathbf{g}_{t-1}, \mathbf{u}_{t-1})$.

As with the Deep Koopman models described in Chapter 6, Deep Variational Koopman models are trained to perform both reconstruction and prediction. The DVK model derives its linear dynamics matrices based on information about states $\mathbf{x}_{1:T}$ and actions $\mathbf{u}_{1:T-1}$, but is trained to minimize the error in modeling the states $\mathbf{x}_{1:T+H}$, where often the reconstruction and prediction horizons are equal ($T = H$). The procedure for performing a single parameter update for the DVK model is outlined in Algorithm 4; these updates can be performed iteratively until convergence using stochastic gradient descent.

Once trained, a Deep Variational Koopman model can be deployed for prediction and control on a modeled system. Each time a set of observations $\mathbf{g}_{1:T}$ is sampled from a trained DVK model, a new, globally linear dynamics model is obtained. By sampling many times, an ensemble of linear models can be constructed that can
Algorithm 4 Deep Variational Koopman model training algorithm

**Input:** States $x_{1:T+H}$ and actions $u_{1:T+H-1}$

1. Run states $x_{1:T}$ and actions $u_{1:T-1}$ through *Temporal Encoder Model* to get temporal encoding
2. Sample $g_1 \sim q_\phi(g_1 \mid x_{1:T}, u_{1:T-1})$
3. **for** $t = 2, \ldots, T$ **do**
   - Update encoding from *Observation Encoder Network*
   - Sample $g_t \sim q_\phi(g_t \mid g_{1:t-1}, x_{1:T}, u_{1:T-1})$
   - Find conditional prior distribution $p_\theta(g_t \mid g_{t-1}, u_t)$ from *Conditional Prior Model*
   **end for**
4. Solve for linear dynamics model from $[A \ B] = YZ^\dagger$ and $A^{-1} = (Y - BT)X^\dagger$
5. Set $z_T \leftarrow g_T$
6. **for** $t = T - 1, \ldots, 1$ **do**
   - $z_t = A^{-1}(z_{t+1} - Bu_t)$
   **end for**
7. **for** $t = T + 1, \ldots, T + H$ **do**
   - $z_t = Az_{t-1} + Bu_{t-1}$
   **end for**
8. **for** $t = 1, \ldots, T + H$ **do**
   - Find $\mu_t = p_\theta(x_t \mid z_t)$, the mean of distribution over $x_t$
   **end for**
9. Calculate lower bound to log-likelihood given by Eq. (7.19)
10. Find gradient with respect to model parameters and take a gradient step to optimize lower bound

Collectively provide a distribution over future outcomes for the modeled system. This is in contrast to the Deep Koopman model, which yields only a single linear dynamics model that encodes no notion of uncertainty in its predictions. A pictorial illustration of the difference between the Deep Koopman and Deep Variational Koopman models can be found in Fig. 7.2. Uncertainty-aware dynamics models can be appealing for a variety of tasks, including control in circumstances when data is limited. The next section details how DVK models can enable uncertainty-aware control.
Deep Koopman Model

Observed States and Actions $x_{1:T}$, $u_{1:T}$

Mapping $g(\cdot)$

Infer Distributions $g(x_{1:T})$...

Least Squares $A = (\bar{Y} - B\Gamma)\bar{X}^\dagger$

Deep Variational Koopman Model

Sample plus Least Squares $A_k, B_k$

Figure 7.2: Illustration of distinction between Deep Koopman and Deep Variational Koopman models.

7.2 Uncertainty-Aware Control

This section details many considerations relating to employing Deep Variational Koopman models for effective control. Deep Variational Koopman models provide linear dynamics, which makes them amenable for incorporation into many control frameworks. The goal is to select a sequence of control inputs $u_{T+1:T+H}$ that minimizes

$$C(z_{T+1:T+H}, u_{T+1:T+H}) = \sum_{t=T+1}^{T+H} c_t(z_t, u_t), \quad (7.20)$$

where $c_t(z_t, u_t)$ is the instantaneous cost, and is a function of the latent state and control input.

A straightforward application of the LQR techniques outlined in Section 4.2 to this problem is not possible because the linear quadratic regulator assumes the presence of a quadratic instantaneous cost function. However, cost functions are generally
defined in terms of the full state $x_t$ rather than the latent state $z_t$, meaning that the instantaneous cost is given by

$$c_t(z_t, u_t) = c_t(g^{-1}(z_t), u_t), \quad (7.21)$$

which is non-quadratic in the latent state. This form of the cost function therefore necessitates the use of the iterative linear quadratic regulator and differential dynamic programming techniques outlined in Section 4.3 and Section 4.4.

For general problems, iLQR techniques will be sufficient because linear models can be used to describe the time evolution of the latent states. However, some control problems may need to account for the presence of control input constraints, in which case the system dynamics can be modeled according to

$$z_{t+1} = Az_t + Bu_{\text{max}} \tanh(u_t), \quad (7.22)$$

where $u_{\text{max}}$ represents the control saturation limits, and the hyperbolic tangent function $\tanh(\cdot)$ maps the control inputs element-wise to values in the interval $[-1, 1]$. Hence, the latent state dynamics are nonlinear with respect to the selected control inputs $u_t$. Under these circumstances, differential dynamic programming can be used in place of iLQR. The first subsection below outlines how quadratic approximations to the system dynamics and cost function can be formed for incorporation into the DDP algorithm.

One additional consideration when using the techniques described in Chapter 4 to optimize action sequences in conjunction with Deep Variational Koopman models is that the LQR, iLQR, and DDP algorithms assume the presence of a single dynamics model $(A, B)$, while DVK models allow for the sampling of multiple sets of dynamics matrices $\{(A, B)_i\}_{i=1}^k$. The second subsection below details how the differential dynamic programming algorithm can be altered to account for the uncertainty encoded by the ensemble of dynamics models, and suggests two possible methods for performing uncertainty-aware control.
7.2.1 Approximate Quadratic Dynamics and Cost

As mentioned above, the latent state dynamics provided by the DVK models can
be nonlinear due to the presence of the hyperbolic tangent function used to enforce
control input constraints. To obtain approximately linear dynamics, a first-order
Taylor expansion can be performed about a reference control input $\hat{u}_t$:

$$z_{t+1} \approx Az_t + Bu_{max} (\tanh(\hat{u}_t) + \nabla u_t \tanh(\hat{u}_t)(u_t - \hat{u}_t))$$

$$= Az_t + Bu_{max} \left( \tanh(\hat{u}_t) + (1 - \tanh^2(\hat{u}_t)) (u_t - \hat{u}_t) \right),$$

where $1$ represents the all-ones vector. Approximate quadratic dynamics can likewise
be obtained by instead considering a second-order Taylor expansion about the reference
control input.

Additionally, the instantaneous cost function is typically a non-quadratic function
of the latent state $z_t$, because costs are easier to define as a function of the full state $x_t$.
One option for obtaining a quadratic cost in the latent space is to take the approach
outlined in Section 6.6 and define the cost to be the $L_2$-distance between the latent
representation of the current state and the representation for a goal state. Such a
choice can be justified when the states are represented as visual inputs [132], but is
generally undesirable, since two states with vastly different costs can potentially have
very similar latent space representations. In this work, an approximate quadratic
cost is constructed in the latent space by finding the gradient and Hessian of the
instantaneous cost, defined as a function of a given full state $\hat{x}_t$, with respect to the
associated latent state $\hat{z}_t$:

$$c_t(z_t, u_t) \approx \frac{1}{2} (z_t - \hat{z}_t)^t c_{zz,t} (z_t - \hat{z}_t) + c_{z,t}^t (z_t - \hat{z}_t) + c_t(\hat{z}_t, u_t),$$

where

$$c_{z,t} = \frac{\partial c_t}{\partial z_t}(\hat{z}_t, u_t) \quad \text{and} \quad c_{zz,t} = \frac{\partial^2 c_t}{\partial z^2_t}(\hat{z}_t, u_t).$$

The procedure of finding the gradient and Hessian of the cost with respect to
the latent state is illustrated in Fig. 7.3. In order for the Hessian to be nonzero, the
second derivatives in the Decoder Network must be nonzero, which precludes the use
of piecewise linear ReLU activations. In all experiments with the Deep Variational Koopman model, nonlinear extensions to ReLU activations, known as exponential linear units (ELUs) [133], were used in the Decoder Network.

### 7.2.2 Accounting for Uncertainty

The standard DDP algorithm assumes the existence of a single, (locally) linear or quadratic dynamics model. However, DVK models provide the ability to sample many possible dynamics models, which taken together can encode uncertainty about how a system will evolve in time. As demonstrated by robust control techniques such as $H_\infty$ [134], linear matrix inequalities (LMIs) [135], and minimax differential dynamic programming [136], accounting for model uncertainty can enable more effective control. The following subsections detail methods for performing uncertainty-aware control.
Optimize for Expected Cost

Given $k$ models with $\{(A, B)_i\}_{i=1}^k$ and $\{z_{T,i}\}_{i=1}^k$, construct a combined state $z_{t,\text{com}}$ that represents the concatenation of the $z_t$-values across all models:

$$z_{t,\text{com}} = [z_{t,1}^T, z_{t,2}^T, \ldots, z_{t,k}^T]^T.$$  \hfill (7.26)

The dynamics of the combined state will be described by creating a block-diagonal matrix out of the $A$-matrices and stacking the $B$-matrices into a single matrix:

$$z_{t+1,\text{com}} = \tilde{A}z_{t,\text{com}} + \tilde{B}u_{\text{max}}\tanh(u_t)$$  \hfill (7.27)

where

$$\tilde{A} = \begin{bmatrix} A_1 & 0 & \ldots & 0 \\ 0 & A_2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & A_k \end{bmatrix} \quad \text{and} \quad \tilde{B} = \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_k \end{bmatrix}.$$  \hfill (7.28)

A similar procedure can be performed with the cost gradients and Hessians to find quadratic approximations to the cost function for the combined state along the reference trajectory. The action sequence can be optimized according to the expected cost across all models.

Optimize for Worst-Case Cost

Given an initial action sequence and $k$ models, the worst-case optimization procedure identifies $(A_{\text{worst}}, B_{\text{worst}})$, the model predicting the largest cost given the initial action sequence and the corresponding initial latent state $z_{T,\text{worst}}$. The action sequence is then optimized based on the predictions from the worst-case dynamics model, and subsequently a new model is identified that predicts the largest cost under the optimized action sequence. This process is repeated until convergence.
Outlier Rejection

One practical problem that can arise while using Deep Variational Koopman models for control is that occasionally a model (or models) will be sampled that strongly disagree(s) with the remaining models in the ensemble. These models can be viewed as outliers, and in practice they can heavily skew the expected or worst-case cost estimates, leading to diminished control performance. To ignore the effect of these outliers, outlier detection techniques can be used.

One technique for identifying outliers is known as the $z$-score method. Given a dataset, the $z$-score method assumes that the data is normally distributed, and proceeds by calculating the empirical mean and standard deviation of the data. Let $C_1, \ldots, C_k$ be the predicted cost by $k$ sampled models. The empirical mean and standard deviation are given by:

$$
\hat{\mu} = \frac{1}{k} \sum_{i=1}^{k} C_i \quad \text{and} \quad \hat{\sigma} = \frac{1}{k} \left( C_i - \hat{\mu} \right)^2.
$$

(7.29)

The $z$-scores associated with the $k$ predicted costs are then given by [137]:

$$
z_i = \frac{C_i - \hat{\mu}}{\hat{\sigma}}, \quad i = 1, \ldots, k.
$$

(7.30)

If the costs are indeed normally distributed, then the $z$-scores should follow a standard normal distribution $z \sim \mathcal{N}(0,1)$, and thus any cost with e.g. $|z| \geq 3$ will be considered unlikely to have been drawn from the underlying data distribution, and will therefore be classified as an outlier.

One drawback to the $z$-score approach is that the empirical mean and standard deviation can be heavily influenced by the presence of outliers, especially for small datasets. This skewing of the empirical mean and standard deviation by the outliers in turn makes it less likely that the outliers are rejected. For this reason, it can be advantageous to perform outlier detection using measures that are robust to outliers,
such as the sample median. The modified z-score of costs $C_1,\ldots,C_k$ is defined as [137]:

$$m_i = \frac{0.6745(C_i - \bar{C})}{MAD}, \quad i = 1,\ldots,k,$$

(7.31)

where $\bar{C}$ is the median cost and $MAD$ is the median absolute deviation, given by:

$$MAD = \text{median}\{|C_1 - \bar{C}|,\ldots,|C_k - \bar{C}|\}.$$  

(7.32)

The constant in the numerator ensures that, for normally distributed data, the same acceptance/rejection criterion can be used for both the z-score and the modified z-score.

While the modified z-score is effective for outlier detection in many domains, another method proves to be more effective for outlier detection in conjunction with Deep Variational Koopman models. In experiments, it was found that more effective controller performance was obtained by rejecting any model with associated cost $C_i$ such that

$$\frac{C_i}{\bar{C}} \geq a$$

(7.33)

where $a$ is a constant that is typically set to $a = 2$. Given that costs are nonnegative, the interpretation of this threshold is that, as $\bar{C}$ decreases (and thus the system is closer to the goal state and more care is required in selecting appropriate actions), the outlier rejection criterion will increasingly become more aggressive. In contrast, the modified z-score will tend to be equally aggressive about outlier rejection regardless of the current state, which can make it hard to set a rejection criterion that performs well throughout the state space.

**Model Predictive Control**

In the presence of disturbances or model errors, executing an entire action sequence determined through DDP may be inadvisable. Because DVK dynamics models will not provide perfect predictions for the time evolution of a modeled system, model predictive control (MPC) is used for closed-loop trajectory planning. At each time step, the last $T$ observed states and actions are fed into the DVK model to find an
ensemble of dynamics models \{((A, B)_i)\}_{i=1}^k \text{ and initial states } \{z_{T,i}\}_{i=1}^k. \text{ Next, the DDP procedures outlined above are used to solve for an open-loop action sequence. The first action in the sequence is executed, and replanning occurs at the next time step.}

7.3 Experiments

This section evaluates the performance of the Deep Variational Koopman models on benchmarks problems for dynamics modeling and control. These particular experiments have been limited to low-dimensional problems because it is easier to visualize whether the models have provided reasonable uncertainty estimates. First, this section introduces the evaluation environments, after which the performance of the DVK models is presented and compared with baseline approaches.

7.3.1 Dynamics Modeling

Evaluations of the DVK models’ ability to learn dynamics were carried out on three benchmark problems: inverted pendulum, cartpole, and acrobot (or double pendulum). In the inverted pendulum environment, a torque is applied at the base of a pendulum, which can be used to swing up and balance the pendulum. In the cartpole environment, a cart with a pendulum affixed to it is positioned along a track that permits horizontal movement. Forces can be applied to the cart, which induce lateral movement, which in turn induces rotational motion in the pendulum. In the acrobot environment, two pendulums are chained together. A torque applied at the joint that connects the two pendulums can induce swinging motion in the double pendulum system. The OpenAI Gym [138] implementation of each environment is used, and the cartpole and acrobot environments were modified to give them continuous action spaces. An illustration of the three evaluation environments can be found in Fig. 7.4, and the governing equations for each system can be found in Appendix B.
CHAPTER 7. UNCERTAINTY-AWARE MODELING AND CONTROL

Baseline Models

The DVK models are benchmarked against three baseline models. For a fair comparison, whenever a baseline shares a component with the DVK model, such as the LSTM and decoder in the DVBF and LSTM models, the exact same hyperparameters are used across all models. In the experiments, all latent states were set to be four-dimensional.

The Deep Variational Bayes Filter (DVBF) model [117], originally introduced in Chapter 6, assumes the presence of latent states $z_t$ with locally linear dynamics

$$z_{t+1} = A_t z_t + B_t u_t + C_t w_t,$$

(7.34)

where $A_t$, $B_t$, and $C_t$ are functions of the current latent state and $w_t$ is a noise vector. The distribution over $w_1$ is output by an LSTM that encodes information about the sequence of states $x_{1:T}$, and $z_1$ is assumed to be a function of $w_1$.

The Long Short Term Memory (LSTM) model propagates a latent state $z_t$ with a recurrent neural network and uses a decoder network to map $z_t$ to $x_t$. The latent state at each time step is a function of the previous latent state, the previous control input, and the network hidden state. The initial latent state $z_1$ is drawn from a distribution output by an LSTM that encodes information about states $x_{1:T}$.

An ensemble of 10 feedforward neural networks, denoted by (FF-Ensemble), is also trained. Each model is a fully-connected neural network trained to map the current state and action to the next state. The range of predictions in the model ensemble can be used as a measure of prediction uncertainty. Recent work has shown that
ensembles of feedforward networks can serve as probabilistic dynamics models that enable effective control on a variety of tasks [114].

**Training Details**

All models were implemented in Tensorflow [84]. In each environment, 1000 trials were run for 256 time steps with random control inputs. Models were trained on subsequences of states and actions extracted from the trial data. Knowledge about a sequence of states $x_{1:T}$ is required to sample the initial latent state in the DVK, DVBF, and LSTM models (and the dynamics model for DVK). For this reason, a distinction is once again drawn between reconstruction, in which a model simulates the evolution of states $x_{1:T}$ about which it already has knowledge, and prediction, in which a model predicts the evolution of states $x_{T+1:T+H}$.

A dynamics model that achieves strong predictive performance is desirable for control applications. If a model is only trained to reconstruct states it has already observed, then there is a chance that it will not generalize well to the prediction task. Therefore, the DVK model is trained to minimize the reconstruction and prediction errors for states $x_{1:T+H}$, where it is often defined that $T = H$. Employing a similar training procedure was found to inhibit learning for the LSTM model, but did improve the predictive performance of the DVBF, and as such all DVBF results are from models trained in this manner.

**Results**

The trained models were evaluated on 5000 64-step test sequences from each environment. For each test sequence, 10 predictions were generated with each model. For each prediction, the DVBF and LSTM models sample different initial latent states $z_1$, and the DVK model samples different values of $z_T$ and dynamics matrices. Distinct predictions are obtained from the neural network ensemble by generating recursive predictions with each trained model. Figure 7.5 provides a qualitative picture of the DVK model’s ability to simulate the dynamics on one test sequence from the acrobot environment. A strong agreement can be observed between the model’s predictions
Figure 7.5: Predictions for the acrobot environment. The first 64 steps are reconstruction and the final 64 steps are prediction. Black lines indicate true state values, blue lines represent mean predictions, and shaded regions represent the range of predictions across all models.
and the true time evolution of the system, with higher uncertainty present near local minima and maxima.

The predictive performance of the models are quantified according to two metrics:

1. The mean squared error (MSE) as a function of prediction horizon.

2. The negative log-likelihood (NLL) of the test data as a function of prediction horizon, summed across trials.

The likelihood is calculated by fitting a Gaussian distribution to the 10 predictions generated by each model and determining the probability density the distribution assigns to the true state value. Figure 7.6 shows model performance according to these metrics on the three studied environments. The likelihood results for the LSTM model are omitted because its fitted distributions assigned zero likelihood to some of the test data, which corresponds to infinite negative log-likelihood.

The feedforward neural network ensemble performs quite well, achieving low prediction error and assigning high likelihood to the test data. However, the results for the pendulum problem, where the prediction error grows exponentially, illustrate one drawback of using models trained to make single-step predictions. The prediction errors for such models can grow exponentially when they are used to make multi-step predictions due to errors compounding over time [128]. In fact, over a horizon of 128 time steps in the pendulum environment, the mean-squared prediction error for the ensemble grows to values on the order of $10^6$. The DVK model achieves competitive performance with the neural network ensemble, while not suffering from the same instabilities and also providing linear dynamics models that can be used more easily for control.

The DVBF outperforms the LSTM baseline, and attains performance that is often close to that of the DVK model, but at a much higher computational cost. The DVK model computes a single dynamics model that it uses to propagate the latent state for all time steps, while the DVBF must compute a new dynamics model, which is a nonlinear function of the current latent state, at each time step. Therefore, the computational graph for the DVBF takes significantly longer to compile, and
Figure 7.6: MSE as a function of prediction horizon and NLL assigned to the test data (lower is better) in the three simulation environments.
furthermore the time required to perform a forward and backward pass during training was approximately an order of magnitude longer for the DVBF.

### 7.3.2 Control

The effectiveness of the control procedure detailed in Section 7.2 is evaluated on the inverted pendulum environment, in which the goal is to swing up and balance an underactuated pendulum. The cost function is given by:

\[
c(\theta, \dot{\theta}, \tau) = \theta^2 + 0.1\dot{\theta}^2 + 0.001u^2, \tag{7.35}
\]

where \(\theta\) is the angle of the pendulum with relative to vertical, \(\dot{\theta}\) is the angular velocity, and \(u \in [-2, 2]\) is the torque applied to the base of the pendulum. Hence, the cost function penalizes deviations of the pendulum from vertical, as well as nonzero angular velocities and control inputs. In each episode, the pendulum is initialized in a random state and the system is simulated for 256 time steps.

A total of 50 trials were run with random control inputs, and a Deep Variational Koopman model was trained on the trial data with a reconstruction and prediction horizon of \(T = H = 16\) and a KL-divergence weight of 0.1. Because the original trial data does not contain many instances where the pendulum is near the goal state, 20 additional trials were run where the actions were selected through model predictive control optimizing for expected cost with five sampled models. The DVK model was then finetuned on data from these trials before the final experiments were carried out.

The results for different ensemble sizes and optimization procedures, taken from 1000 seeded trials, can be found in Table 7.2. Performance was quantified according to three metrics:

1. The average cost incurred in each trial.
2. The fraction of the time the pendulum was vertical across all trials.
3. The average number of falls per trial.
Table 7.2: Control Performance

<table>
<thead>
<tr>
<th>Number of Models</th>
<th>Cost</th>
<th>Fraction of Time Vertical</th>
<th>Falls per Trial</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Worst Case</td>
<td>Expected</td>
<td>Worst Case</td>
</tr>
<tr>
<td>1</td>
<td>-</td>
<td>370.7</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>307.5</td>
<td>285.7</td>
<td>0.710</td>
</tr>
<tr>
<td>10</td>
<td>291.9</td>
<td>274.3</td>
<td>0.733</td>
</tr>
<tr>
<td>30</td>
<td>251.6</td>
<td>270.9</td>
<td>0.771</td>
</tr>
</tbody>
</table>

The pendulum is considered to be vertical when $\theta \in [-\pi/8, \pi/8]$ and a fall is classified as a scenario where $\theta \notin [-\pi/8, \pi/8]$ after being in the interval for more than 20 time steps. The best performance according to each metric is highlighted in bold.

The results show a clear benefit from sampling more models. The trend in performance improvement is more pronounced for the worst-case optimization scheme; when optimizing for expected cost there is an apparent benefit to sampling more models, but with diminishing returns. The best performance is obtained with 30 models, with the lowest average cost achieved through a worst-case optimization procedure. However, optimizing for expected cost leads to much better performance for smaller model ensembles, and thus could be a preferable approach if the goal is to obtain satisfactory performance while keeping the number of sampled models low.

### 7.4 Discussion

This chapter introduced the Deep Variational Koopman model, a method for inferring Koopman observations and sampling ensembles of linear dynamics models that can be used for prediction and control. The DVK models were demonstrated to perform accurate, long-term prediction on a series of benchmark tasks, while generating reasonable uncertainty estimates. Furthermore, the application of DVK models to control tasks was discussed, covering important considerations such as outlier rejection, control input constraints, and representing the cost function in the latent space. Finally, it was shown that accounting for the uncertainty encoded by multiple sampled models improved controller performance on the inverted pendulum task.

There are some shortcomings of the proposed approach that should be addressed.
In some cases, the matrix inversions that need to be performed in Eqs. (7.6) and (7.7) can be ill-conditioned, causing the operation to fail and training to terminate. This problem can usually be mitigated by adding small positive values to the diagonal entries of the inverted matrix. However, this form of regularization generally means that the derived \((A, A^{-1})\) matrices are not truly inverses of each other, which may inhibit learning by causing discrepancies when modeling the forward- and reverse-time dynamics. Thus, this problem should ideally be addressed in a more elegant way. Additionally, performance optimizations could potentially improve the efficiency of the control algorithm outlined in Section 7.2. Action selection with the control procedure currently takes approximately 0.5 seconds, which may be too slow for real-time control in some applications.

There are a few natural extensions to the methods and results presented in this chapter. First, the DVK model training algorithm could be extended to explicitly account for stochasticity in the dynamics and/or measurement noise. There are several related works that may provide insights into how to account for these phenomena [139, 140]. Furthermore, modifications should be explored that allow for the derived dynamics models to exhibit desirable properties, such as stability or controllability. Finally, while the experiments in this chapter were restricted to low-dimensional systems, future studies should test the applicability of the presented methods to modeling and control of higher-dimensional systems.
Chapter 8

Conclusions

For many real-world systems, dynamics models remain difficult to define. In recent years, the interest in learning dynamics models directly from data has intensified. The methods presented in this thesis represent new additions to this data-driven modeling paradigm. This chapter summarizes the content and contributions of this thesis, and concludes with an outline for future research directions.

8.1 Summary

Analysis, simulation, prediction, and control all rely on models that describe how dynamical systems change with time. Such models are often derived through an understanding of the physical laws that govern a system’s behavior. Unfortunately, for some dynamical systems, such as complex fluid flows, high-fidelity evaluation of the governing equations can be prohibitively expensive. For other systems, there are no underlying physics to exploit, making models difficult to define. In addition, other challenges can arise when constructing models for real-world systems; principal among these are nonlinearity and high-dimensionality. This section briefly summarizes how the data-driven methods introduced in this thesis have helped to address these challenges.

High-fidelity physics-based simulations such as CFD simulations can generate vast quantities of data and are often very computationally expensive due to the
large number of degrees of freedom that must be modeled. To help alleviate these costs, this thesis introduced new methods that enable (1) more efficient storage of CFD data and (2) more efficient parameterized simulations. The first method, a multi-stage compression procedure, was shown to significantly reduce the storage overhead associated with running CFD simulations, while still allowing for accurate reconstruction of flow solutions at all time instances. The second method, known as parameter-conditioned sequential generative modeling, was shown to generate flow solutions orders of magnitude faster than CFD solvers, and allowed for accurate simulation of both two- and three-dimensional fluid flows at a range of flow conditions.

Many real-world systems have nonlinear dynamics, making effective control of such systems a challenge. Inspired by Koopman theory, this thesis introduced data-driven modeling algorithms that map nonlinear dynamics to linear dynamics, thereby allowing the application of linear control techniques to nonlinear systems. First, a Koopman-based method was shown to accurately model the high-dimensional, nonlinear dynamics of vortex shedding over a cylinder. Incorporation of the learned model into a model predictive control procedure enabled effective suppression of the vortex shedding, while also providing insight into a simple control law that is equally effective at vortex suppression. Subsequently, the Deep Variational Koopman model was introduced, which is a Koopman-based method that facilitates uncertainty-aware modeling and control. Experiments demonstrated that the Deep Variational Koopman model is capable of accurately modeling benchmark dynamical systems while providing reasonable uncertainty estimates. Furthermore, it was shown that accounting for the uncertainty in the learned model’s predictions improved controller performance on the inverted pendulum task.

8.2 Contributions

Chapters 4 and 5 do not present any original techniques, but instead review existing ideas from optimal control and Koopman theory that are relevant to later chapters. The remainder of the thesis makes the following contributions:

- A novel multi-stage procedure for compressing high-dimensional CFD
data: Chapter 2 introduced a method for nonlinear dimensionality reduction of simulation data with high-order discretizations, consisting of (1) local compression with autoencoders (Section 2.4) and (2) global compression with principal component analysis (Section 2.5). This approach can be applied to high-dimensional data (e.g. with millions of degrees of freedom) even in circumstances where the number of training examples is relatively small. The use of three-dimensional convolutions to learn from three-dimensional fluid flow data is original. Furthermore, the use of autoencoders before performing principal component analysis yields a nonlinear embedding of the global flow states, which is generally not attainable with PCA. Finally, this work demonstrates how the dynamics of the compressed flow states can be learned (Section 2.6).

• An algorithm for parameter-conditioned sequential generative modeling of dynamical systems: Chapter 3 presents an extension to sequential generative modeling techniques that allows for conditioning generated solutions on physical parameters. A mutual information objective was introduced (Section 3.2.4), which encourages the learned models to account for the effect that prescribed parameters have on a modeled system’s dynamics. This work represents the first application of VAE-like architectures for modeling fluid flows. Furthermore, this work demonstrates how convolutional neural network architectures can be applied to modeling fluid flows that have been simulated on large, unstructured grids.

• A new method for applying Koopman theory to modeling and control of high-dimensional, nonlinear systems: Chapter 6 applies a modified form of a Koopman-based algorithm originally proposed by Takeishi et al. [98] to model the forced dynamics of high-dimensional fluid flows. Modifications to the algorithm were proposed that allowed the learned models to perform more accurate multi-step prediction (Section 6.2.1) and account for the presence of control inputs (Section 6.5). This work also represents that first application of the algorithm by Takeishi et al. to high-dimensional systems such as fluid flows. Finally, this work illustrates that data-driven methods can be used to discover
simple, effective control laws for nonlinear systems.

- An extension to Koopman-based methods that enables uncertainty-aware modeling and control: Chapter 7 introduced the idea that uncertainty can be incorporated into Koopman-based models by inferring distributions over the Koopman observations, and a tractable inference procedure was derived (Section 7.1.1) that yields the desired uncertainty-aware dynamics models. To make the learned models suitable for control, a method for obtaining approximate quadratic costs in the latent space was described (Section 7.2.1). Finally, modifications to the differential dynamic programming algorithm were detailed (Section 7.2.2) that enable the use of expected or worst-case optimization procedures.

## 8.3 Future Work

This thesis has drawn upon concepts from machine learning, (fluid) dynamics, and control. Each one of these fields contains a rich set of existing and emerging methods, of which only a small subset could be considered here. Many opportunities exist for extending the ideas presented in this work. Some areas of further study are highlighted below.

### 8.3.1 Incorporating Knowledge of Governing Equations

The dynamics modeling techniques presented in this thesis assume no knowledge about the governing equations of a modeled system. This is appealing because it allows the techniques to be broadly applicable to many classes of dynamical systems. However, equation-free modeling does not take advantage of the large amount of existing knowledge about many physical systems, which in some circumstances has been accrued over centuries of scientific study. Therefore, an opportunity exists for incorporating prior physical knowledge into learned dynamics models. Some interesting early work has already been done along these lines [141–144], but this remains an open field of research with the potential for high impact.
8.3.2 Accounting for Noise and Stochasticity

All methods introduced in this thesis assume noiseless observations and deterministic dynamics. However, to be applicable to many real-world systems, the methods must be extended to contend with noisy observations and stochasticity. Extensions would likely require explicitly accounting for these phenomena in the latent state dynamics. For example, filtering procedures could be introduced to account for measurement noise, and randomness could be introduced into the latent state transition model to account for stochasticity. Ideas for modifying the problem formulation can potentially draw on recent works related to learning latent state-space models [117, 139, 140].

8.3.3 Scaling to Larger or More Complex Problems

The impact that can be achieved with data-driven modeling techniques is partly a function of the number of problems to which the techniques can be applied. This thesis has demonstrated that data-driven techniques can be used to model the dynamics of a number of complex, high-dimensional systems. However, there still exist limitations relating to flexibility and scalability that should be addressed. In the context of modeling fluid flows, large-scale simulations, unstructured meshes, and variable flow geometries can all present problems for modern deep learning architectures. Hardware improvements may alleviate the memory constraints that currently prohibit learning from large-scale simulations, but algorithmic advances are likely required to derive the necessary neural network architectures and training techniques to learn more effectively from unstructured meshes and variable geometries.

8.3.4 Emerging Machine Learning Techniques

The methods presented in this thesis could be improved or extended by taking advantage of many powerful machine learning techniques that have emerged in recent years. One possibility is to extend the modeling techniques to dynamical systems with continuous-time dynamics. Neural ODEs [145] represent a new deep learning paradigm that, among other things, enables the formulation of continuous-time learning problems, and could thus be applied to this task. Additionally, there are a number of exciting
new generative modeling techniques [146, 147] that could be used to extend the work from Chapter 3.

Finally, this thesis has considered methods for learning-based *modeling* of dynamical systems, but has not studied any methods, such as (deep) reinforcement learning, for learning-based *control*. Given the difficulty of controller design for nonlinear dynamical systems, techniques that learn control policies directly through interaction with an environment could be appealing options for the discovery of effective control strategies. Some preliminary work has evaluated the effectiveness of deep reinforcement learning for fluid flow control [148], but there are many problems that have yet to be studied.
Appendix A

Generative Modeling Plots

Figure A.1: Comparison of density values over time for CFD (top) and generated (bottom) solutions as a function of rotation speed at Reynolds number $Re = 150$. 
Figure A.2: Comparison of $x$-velocity values over time for CFD (top) and generated (bottom) solutions as a function of rotation speed at Reynolds number $Re = 150$.

Figure A.3: Comparison of $y$-velocity values over time for CFD (top) and generated (bottom) solutions as a function of rotation speed at Reynolds number $Re = 150$. 

Figure A.4: Variation of dominant frequency in density signal as a function of rotation speed and Reynolds number for CFD and generated solutions.

Figure A.5: Variation of dominant frequency in $x$-velocity signal as a function of rotation speed and Reynolds number for CFD and generated solutions.
Figure A.6: Variation of dominant frequency in $y$-velocity signal as a function of rotation speed and Reynolds number for CFD and generated solutions.
Appendix B

Environment Governing Equations

Pendulum

The pendulum system has one degree of freedom – the angle of rotation $\theta$. The angular acceleration is given by:

$$\ddot{\theta} = -\frac{3g}{2\ell} \sin \theta + \frac{3}{m\ell^2} u,$$  \hspace{1cm} \text{(B.1)}

where $g$ is the acceleration due to gravity, $\ell$ is the length of the pendulum, $m$ is the mass of the pendulum, and $u$ is the control input.

Cartpole

The cartpole system has two degrees of freedom – the translation of the cart $x$ and the rotation of the pendulum $\theta$. The acceleration of the cart is given by:

$$\ddot{x} = \frac{1}{M + m} \left[ \frac{m\ell}{2} \left( \dot{\theta}^2 \sin \theta - \ddot{\theta} \cos \theta \right) + u \right],$$  \hspace{1cm} \text{(B.2)}

where $M$ is the mass of the cart and $m$ is the mass of the pendulum. The angular acceleration of the pendulum can be expressed as:

$$\ddot{\theta} = \frac{3}{2\ell} \left( g \sin \theta - \dot{x} \cos \theta \right).$$ \hspace{1cm} \text{(B.3)}
Acrobot

The acrobot system has two degrees of freedom, $\theta_1$ and $\theta_2$, corresponding to the rotation angle of each pendulum. The rotation angle of the second pendulum is defined relative to the first pendulum. It is assumed that the pendulums have the same mass $m$, length $\ell$ and moment of inertia $I$. The angular acceleration of the first pendulum can be found through [149]:

$$\ddot{\theta}_1 = -\frac{d_2 \dot{\theta}_2 - \phi_1}{d_1}$$  \hspace{1cm} (B.4)

and $\ddot{\theta}_2$ can be found through:

$$\ddot{\theta}_2 = \frac{u + \frac{d_2}{d_1} \phi_1 - \phi_2}{\frac{1}{4} m\ell^2 + I - \frac{d_2^2}{d_1^2}}$$  \hspace{1cm} (B.5)

where $d_1, d_2, \phi_1, \phi_2$ are defined as follows:

$$d_1 = \frac{3}{2} m\ell^2 + m\ell^2 \cos \theta_2 + 2I$$

$$d_2 = m \left( \frac{1}{4} \ell^2 + \frac{1}{2} \ell^2 \cos \theta_2 \right) + I$$

$$\phi_1 = -\frac{1}{2} m\ell^2 \dot{\theta}_2^2 \sin \theta_2 - m\ell^2 \dot{\theta}_1 \dot{\theta}_2 \sin \theta_2 + \frac{3}{2} m g \ell \sin \theta_1 + \phi_2$$

$$\phi_2 = \frac{1}{2} m g \ell \sin (\theta_1 + \theta_2).$$
Bibliography


