Imperial College London

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

Optimal Control of High-Dimensional Dynamical Systems

Author:
Bowen Fan

Supervisors:
Dr Antonio Del Rio Chanona
Dr Panagiotis Petsagkourakis

Submitted in partial fulfillment of the requirements for the MSc degree in Computing of Imperial College London

September 2021

Abstract

Dynamical systems are applicable to almost every scientific field of study: from engineering systems to climatology and from quantum physics to epidemiology. They are often described by complex system models that can be high-dimensional and nonlinear. Where a dynamical system can be controlled through external inputs, it is useful to know the best control policy that can be applied to drive the system towards a desired state. This is known as optimal control. Deriving optimal control solutions for high-dimensional and nonlinear dynamical systems is a challenging problem that is actively researched.

This thesis makes 2 contributions in developing optimal control methods for such systems. The first method uses an optimal value approximation neural network that borrows concepts from Q-learning. With this approach, a system can be optimally controlled by applying actions that are predicted to maximise its optimal value function. The neural network learns to approximate this value function through an experience replay training process. The second method combines an autoencoder neural network for model reduction with SINDYc, an algorithm to identify system models from simulation or experimental data. By identifying the dynamics between control inputs and the state variables reduced by the autoencoder, a simplified system model can be obtained, thereby greatly reducing the computational cost of solving for optimal controls.

The first method was found to work well. It demonstrates to be capable of finding optimal control actions for a nonlinear case study system with 20 dimensions that is subject to state constraints. This approach offer notable improvements over more commonly used methods for model reduction. The second method was found to work moderately well for unconstrained systems, but less so for those subject to state constraints. Nevertheless, it is a versatile method that is likely extensible to constrained systems with further work.

Acknowledgements

I am sincerely grateful to my supervisors, Dr Antonio Del Rio Chanona and Dr Panagiotis Petsagkourakis, for the opportunity to do this thesis project. Through this project, I have learnt much about dynamical systems, model reduction, control, neural networks, scientific computing and computational optimisation. This is certainly the biggest and most fulfilling project I have worked on during my studies.

I wish to further thank my supervisors for their patient teachings and advice, their insightful feedback in our discussions, and for the academic resources that they pointed me to when I have gaps in my understanding, which were often precisely what I needed. All in all, I am very happy to have chosen this challenging project.

Contents

Chapter 1

Introduction

1.1 Introduction to Dynamical Systems and Control

A dynamical system is a system whose state evolves over time according to a set of evolution rules. The study of dynamical systems is wide ranging and concerns almost every field in science, from engineering systems such as aircraft engines, to weather systems such as the movements of hurricanes. The system may accept external inputs which partly govern its evolution, where it is referred to as a forced system. For example, a pendulum rod may have a motor attached at its pivot point, which can exert a torque on the rod to swing it upwards and keep it at the upright position. Hence, by means of applying input(s), it is possible to drive the system towards a desired state: this is known as control.

1.1.1 Optimal Control

The optimal control of a dynamical system pertains to finding an optimal control policy that would most efficiently drive the system towards a desired state. More precisely, it involves deriving a control policy that would maximise an objective function, or minimise a cost function. Such a cost function is usually formulated as a weighted combination of 2 terms: a setpoint cost which is a function of how far the system is from its desired state, and a cost of applying control inputs. They will be referred to in this thesis as the setpoint cost and the controller cost respectively. The cost terms are often quadratic, as they represent penalties for deviating from an ideal system or controller state, where the deviation is undesirable in either direction.

The scope of this thesis is limited to discrete-time control policies, where a control policy is expressed as a sequence of control actions (Section 1.1.3.2). The control of a system may be subject to constraints on both the controller and system states. To illustrate, there are often upper limits on control actuators, for example in a car where there is a limit to the engine torque, thereby also limiting its acceleration. There are also

often state constraints. For example, the car's speed must not exceed the legal speed limit. As such, the optimal control problem (OCP) for the car may be to minimise the distance deviation from its destination while economising on fuel, while being subject to acceleration and speed constraints. Equation 1.4a shows an example of the OCP formulation for a linear system whose cost function is quadratic.

1.1.2 High-Dimensional Dynamical Systems

Dynamical systems, particularly those whose evolution is governed by partial differential equations (PDEs), can be complex to model. In many cases, the solution of a PDE involves converting it into a high-dimensional set of ordinary differential equations (ODEs), for example via spatial discretisation. A finer discretisation scheme may help to obtain a more accurate model, but this is at the expense of the complexity of the resultant system, in terms of its state space and the number of ODEs. For example, finite element models of turbulent fluid flow systems may involve millions to billions of state variables [1]. Indeed, much of the earlier motivation in reducing complex models to more tractable lower-dimensional approximations had arisen in the field of fluid mechanics [2–5].

To illustrate, a dynamical system that will be used as a case study in this thesis is the distribution of heat in a one-dimensional thin rod. This system evolves according to the intuitive rule that heat flows down a temperature gradient over time. More formally, this system is governed by the PDE:

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}, \tag{1.1}$$

where $u(x,t)$ is the temperature at length x of the rod at time t, and α is the thermal diffusivity constant, characterising the ease of heat transfer in the material. This relatively simple PDE can be spatially discretised into an arbitrarily large number of rod segments, known as finite elements, such that the temperature evolution of each finite element would be governed only by an ODE. An illustration of a 20-finite element discretisation is given in Section 3.1.1.

1.1.3 Types of Dynamical System Models

The scope of this thesis is limited to time-invariant dynamical systems whose system model do not change with time. For example, a linear time-invariant system can be expressed in its state-space representation $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t)$, where $\mathbf{x}$ represents the internal states of the system and $\mathbf{u}$ represents inputs applied to it. In this thesis, system inputs refer exclusively to control inputs, where $\mathbf{u}$ is externally manipulated to drive a system to a desired state. $\mathbf{A}$ and $\mathbf{B}$ are constant matrices that provide the coefficients to $\mathbf{x}$ and $\mathbf{u}$. In contrast, a time-variant system may be represented in the form $\dot{\mathbf{x}} = \mathbf{A}(t)\mathbf{x}(t) + \mathbf{B}(t)\mathbf{u}(t)$, where the coefficients from $\mathbf{A}$ and $\mathbf{B}$ are themselves a

function of time. In this thesis, both linear and nonlinear time-invariant systems will be studied.

1.1.3.1 Linear and nonlinear systems

Dynamical systems can be *linear* (1.2a), *control-affine nonlinear* (1.2b), or *general nonlinear* (1.2c).

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t) \tag{1.2a}$$
$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) + \mathbf{g}(\mathbf{x})\mathbf{u} \tag{1.2b}$$
$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}) \tag{1.2c}$$

where $\mathbf{x} \in \mathbb{R}^m$, $\mathbf{A} \in \mathbb{R}^{m \times m}$, $\mathbf{u} \in \mathbb{R}^n$, $\mathbf{B} \in \mathbb{R}^{m \times n}$ for the linear system. $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t)$ is usually simply written as $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$. In general, the study of linear systems is much more developed than that of nonlinear systems. This includes techniques for computing the optimal control policy and for reducing the dimensionality of the system states, where many methods developed for linear systems cannot be applied effectively to nonlinear ones.

A common method to make nonlinear systems more tractable is to approximate the nonlinear terms with linear functions around the system's operating point(s) [6]. A key drawback is that these linearised models are only accurate around the linearisation point(s), thus multiple linearised systems may be needed to describe a nonlinear system [7]. The methods developed in this thesis will be applicable to both linear and nonlinear systems.

1.1.3.2 Continuous and discrete-time systems

Equations 1.2a to 1.2c are defined as differential equations in continuous time. Where it is of interest to study, simulate, or control a system in discrete time steps, the system model can be instead defined in discrete-time, usually as a set of difference equations (1.3a to 1.3c). A system model in continuous time can be discretised via finite element methods such as zero-order hold; conversely, methods such as interpolation can be used to approximate a continuous-time model from a discrete-time model [8]. In discrete-time models, the difference equations act a map from one set of system states and control actions to the next set of system states:

$$\mathbf{x}_{t+1} = \mathbf{A}\mathbf{x}_t + \mathbf{B}\mathbf{u}_t \tag{1.3a}$$
$$\mathbf{x}_{t+1} = \mathbf{f}(\mathbf{x}_t) + \mathbf{g}(\mathbf{x}_t)\mathbf{u}_t \tag{1.3b}$$
$$\mathbf{x}_{t+1} = \mathbf{f}(\mathbf{x}_t, \mathbf{u}_t), \tag{1.3c}$$

where the matrices and functions are converted to their discrete-time form.

1.1.3.3 System Inputs and Outputs

A dynamical system may have one or more inputs, and one or more outputs. Correspondingly, they are referred to as single-input single-output (SISO), or multiple-input multiple-output (MIMO), or permutations thereof (SIMO, MISO). The case study system for this thesis (Section 3.1.1) is a MIMO system with 2 inputs and 2 outputs.

System output(s) $\mathbf{y}$ are measurements of the internal system state(s), such that $\mathbf{y} = \mathbf{C}\mathbf{x}$. For example, a thermometer may be fixed to a position on a metal rod, measuring the temperature of that position in particular. The temperature of the other internal states of the rod thus cannot be observed directly. For this thesis, when the case study system described in Section 3.1.1 is simulated and its internal states recorded, it is implicitly assumed that these internal states can all be measured. More precisely, the simulation measurements in this thesis assume that $\mathbf{C}$ is the identity matrix, such that the true values of $\mathbf{x}$ are directly obtained. Hence, in the context of simulation, 20 outputs are assumed. This approach is typical for computer-simulated dynamical systems, but is less applicable to real-world experiments [9].

1.2 Model Predictive Control

Model Predictive Control (MPC) is a versatile optimal control strategy that involves iteratively solving an optimal control problem for each time step as the system under control moves forward in time. For example, on a linear system with a quadratic cost function, the OCP is a quadratic program in the general form:

$$\begin{aligned}
\underset{\mathbf{u}}{\text{minimise}} \quad & J(\mathbf{x}, \mathbf{u}) = \int_0^T \mathbf{x}^\mathbf{T}\mathbf{Q}\mathbf{x} + \mathbf{u}^\mathbf{T}\mathbf{R}\mathbf{u}\, dt && \text{(1.4a)} \\
\text{subject to} \quad & \dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} && \text{(1.4b)} \\
& \mathbf{x} \in \mathbb{R}^{m_x},\ \mathbf{x}(0) = \mathbf{x}_0 && \text{(1.4c)} \\
& \mathbf{u} \in \mathbb{R}^{n_u} && \text{(1.4d)}
\end{aligned}$$

$J(\mathbf{x}, \mathbf{u})$ is the cost function to be minimised over a control time horizon from 0 to T, $\mathbf{x}$ is the vector of the system state and $\mathbf{u}$ is the vector of the control inputs. $\mathbf{Q}$ and $\mathbf{R}$ are matrices of the relative weights for the setpoint costs and the controller costs respectively. The control inputs are typically bounded to a set of admissible control actions $\mathbf{u} \in \mathbb{R}^{n_u}$, and the actions must leave the system in an admissible state $\mathbf{x} \in \mathbb{R}^{m_x}$.

The MPC controller solves the discrete form of this optimal control problem repeatedly, while the system advances in discrete time steps:

1. The sampling time starts at $t = 0$ and the system state $\mathbf{x}(0)$ is measured. The optimal control problem is solved, with $\mathbf{x}(0)$ as the initial state, from 0 to the length of the control horizon T, thus obtaining:

(a) the optimal control policy from 0 to $T-1$, $\{\mathbf{u}^*(0), \mathbf{u}^*(1), \ldots, \mathbf{u}^*(T-1)\}$

(b) the predicted state trajectory $\{\hat{\mathbf{x}}(1), \hat{\mathbf{x}}(2), \ldots, \hat{\mathbf{x}}(T)\}$ as a result of the control sequence, which together is expected to minimise $J(\mathbf{x}, \mathbf{u})$.

The prediction horizon may be longer than the control horizon, such that the MPC controller makes a forecast of the state trajectory beyond the last computed control action up to this prediction horizon. The controls after the control horizon are set according to some terminal control law (often kept constant).

2. Only the first set of control actions in the sequence, $\mathbf{u}(0)$, is applied.

3. The system moves 1 step forward in time to $t = 1$ and $\mathbf{x}(1)$ is measured. It is expected that the previously predicted $\hat{\mathbf{x}}(1)$ and $\mathbf{x}(1)$ are not exactly equal. The optimal control problem is now solved from 1 to $T+1$, and again only the first set of control actions $\mathbf{u}(1)$ in the new control sequence is applied. This process is repeated for as long as the system is running under control.

Figure 1.1 illustrates the MPC sequence described. Several characteristics of MPC control are apparent:

- MPC requires the knowledge of the system model because it needs to make forecasts on the state trajectory. This may be derived from first principles or approximated from operation data using model identification techniques.

- If this system model used by the MPC controller were perfect, and if there were no disturbances and measurement errors, then the predicted trajectory from $\hat{\mathbf{x}}$(t) to $\hat{\mathbf{x}}$(t+T) would be identical to the actual measured trajectory. In this case it would be unnecessary to repeatedly solve for the sequence of optimal control actions and apply only the immediate actions: the optimal sequence will remain constant in such an ideal case.

- Naturally, for the operation of real-world systems, these ideal conditions do not hold: it is expected that there will be some degree of plant–model mismatch, disturbances, and measurement errors. Thus, it is necessary to re-solve the optimal control problem such that the control actions remain optimal when the actual system state deviates from the predicted state.

- Arising from this, the computational cost of using MPC is evident. The controls are calculated in real-time, implying that computation time must not exceed the sampling time step. Thus, MPC has historically been more applicable to systems that have slower dynamics for which larger sampling time steps are suitable, e.g. HVAC control [10] and industrial process control.

- To ensure that controls can be computed in real-time for complex systems with fast dynamics, there have been much research interest in simplifying these com-

plex systems so that they can be more efficiently controlled. Other methods for optimal control such as reinforcement learning may not require a system model. A literature review of these methods is given in Chapter 2.

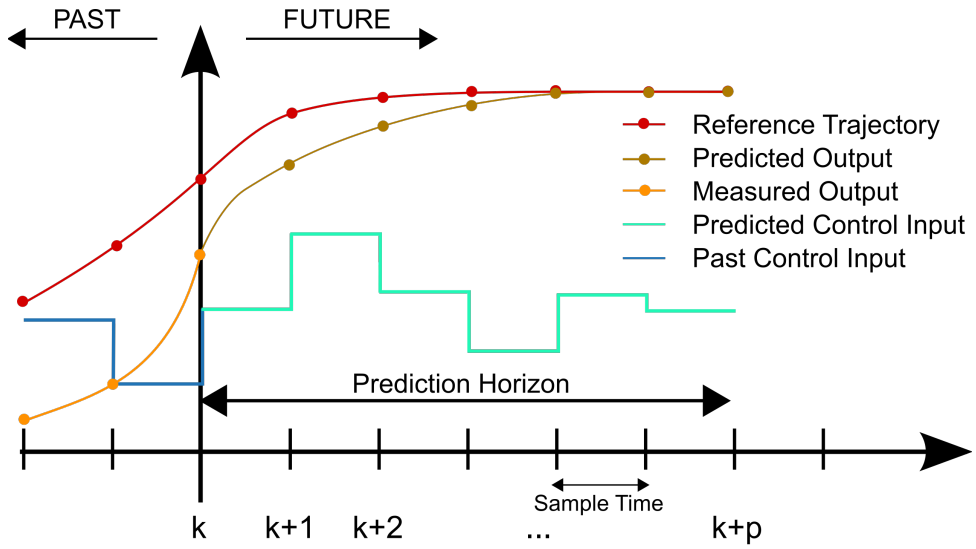


Figure 1.1: Model Predictive Control sequence, image credit: [11]

In this thesis, references are often made to MPC-controlled systems that serve as benchmarks for the control methods developed. This is meant to denote that the system's optimal control actions are computed by directly solving the optimal control problem, as opposed to the value approximation approach used in Chapter 3. The receding horizon characteristic is useful in real-world applications for correcting setpoint errors due to plant-model mismatch and disturbances, but is not an area of focus in this thesis. The case study system used is modelled without introducing plant-model mismatch or disturbances, thus the predicted trajectory will be identical to the actual trajectory. This thesis is focused on comparing the results of the proposed methods against the best attainable controls found by solving the OCP.

1.3 Main Software Libraries Used

1.3.1 Pyomo

Pyomo [12, 13] is a set of Python libraries for modelling and optimisation, developed by a collaboration of authors at the Sandia National Laboratories and at several academic institutions. Pyomo, or more precisely the optimal control solutions computed by Pyomo, is used as the MPC-controller in this thesis (again the focus is on the OCP being solved directly, not on the receding horizon methodology used in MPC).

The system model for a dynamical system is formulated in Pyomo as a set of variables, along with a set of corresponding differential equations. The control objective is itself formulated as a differential equation due to a technical limitation in modelling integrals

in PYOMO. For example, the generic optimisation objective in Equation 1.4a would be formulated as $\dot{J} = \mathbf{x^T Q x} + \mathbf{u^T R u}$. PYOMO is then instructed to solve the optimal control problem by finding the sequence of control actions that would minimise the difference between the final and initial values of J, i.e. $\min (J_T - J_0)$, thereby implicitly performing a numerical integration on J.

1.3.2 IPOPT

PYOMO solves the optimal control problem by making subroutine calls to IPOPT ("Interior Point Optimiser") [14]. IPOPT can solve constrained nonlinear optimisation problems, making it precisely suitable for the optimal control problem formulations in this thesis. The "Primal-Dual Interior Point Filter Line Search Algorithm" employed by IPOPT would be best explained by its authors, Wächter et al., in [14].

1.3.3 CASADI

CASADI [15] is used in conjunction with PYOMO to simulate a system with a fixed set of control actions. The simulation is perform via numerical integration, and is used for testing the performance of the neural network controller developed in this thesis. PYOMO makes internal calls to CASADI when a simulation is needed but not the solutions to the optimal control problem. Conveniently, PYOMO translates the model formulated in itself to a CASADI model when a system simulation is run; only simple adaptations are needed in PYOMO to call the CASADI API.

1.3.4 PYTORCH

PYTORCH [16] is used for implementing the feedforward neural networks (NNs) in this thesis project.

1.4 Project Aim and Objectives

The aim of the project is to devise and implement methods to efficiently control high-dimensional and possibly nonlinear dynamical systems.

To this aim, the following objectives are identified:

1. Perform a literature review of existing model reduction methods and other methods for the control of high-dimensional systems. Classify these existing methods, analyse their respective advantages and disadvantages, and identify learning points. Implement an existing method that is commonly used; this will be a benchmark for the methods developed in Objectives 4 and 5.

2. Implement an optimal control problem solver in PYOMO. This will function as the

MPC controller which is expected to produce the best possible solution in terms of satisfying system objectives and constraints, but not necessarily in terms of computation time. This will be used as the best attainable performance benchmark for the control methods devised in Objectives 4 and 5.

3. Implement a system simulator in PYOMO/CASADI to generate simulated experimental data sets of system trajectories. The simulated data and the system simulator itself will be used in the implementations of the methods developed in Objectives 4 and 5.

4. Develop a neural network that can control a high-dimensional system without having to explicitly solve its optimal control problem. This implementation will borrow concepts from Q-learning, identifying optimal control actions by means of approximating an optimal value function. Evaluate and discuss the performance of this method against the benchmarks described in Objectives 1 and 2.

5. Develop an autoencoder neural network for the reduction of system state variables. Employ a system identification method to discover the dynamics of the reduced system model, thereby formulating a reduced optimal control problem whose explicit solution is expected to be more efficient to compute. Evaluate and discuss the performance of this method against the benchmarks described in Objectives 1 and 2.

6. Evaluate the methods developed in 4 and 5 on their strengths and limitations, and propose potential improvements for these limitations.

1.5 Thesis Organisation

This thesis consists of 5 main chapters, including the current Introduction chapter. The contents of each subsequent chapter are briefly described in this section.

Chapter 2 presents a literature review of the more commonly used methods for simplifying high-dimensional systems so that they can be more efficiently controlled. Next, a discussion of ideas in reinforcement learning that may be applicable to the challenges of high-dimensional control will be given. Finally, the learning points and limitations of these existing methods will be summarised and related to the potential contributions of this thesis.

Chapter 3 presents a method for the optimal control of complex systems using concepts borrowed from Q-learning. Namely, a neural network will be trained as an optimal value function approximator, such that the system may be optimally controlled by seeking the control actions that maximises the value function / minimise the cost function. It will be trained using experience replay, again an idea borrowed from Q-learning. The results obtained with this method's implementation will be presented and compared to

that from an existing model reduction method.

Chapter 4 presents another method where an autoencoder neural network is first used to simplify the state space of a complex system, then a model discovery algorithm is used to fit a system model onto the reduced states and the original control variables. Thus, it is expected that the control actions can be more efficiently computed in the reduced space. The results obtained will be presented and compared to those found in Chapter 3.

Chapter 5 is the concluding chapter. It includes a review on the limitations of the methods implemented in Chapter 3 and Chapter 4, and a discussion on further work that may overcome their limitations or improve their performance. Finally, a review of ethical, legal, and professional considerations in this project will be given. The review will be made with reference to the ethics checklist provided by Imperial College, which is included in Appendix A.

The source code for the implementations in this thesis is hosted on a GitHub repository, accessible via the repository URL at `https://github.com/bowenfan96/mpc-rom`.

Chapter 2

Background and Literature Review

2.1 Model Order Reduction

As described in Section 1.2, the computation time for solving the optimal control problem of a high-dimensional dynamical system may be prohibitively high for model predictive control to be used. This is one of the many factors that contribute to the significant research interest in reduced-order modelling (ROM), where the aim is to simplify high-dimensional systems while preserving their dynamics as much as possible. Many techniques in ROM were first developed for tackling problems in fluid mechanics, particularly those involving turbulent flow. An overview will be provided in the following sections on the more commonly applied methods in MOR, including the Proper Orthogonal Decomposition (POD), Balanced Truncation (BT), and the Balanced Proper Orthogonal Decomposition (BPOD).

More recently, there has been research into artificial neural network (NN) approximations for model reduction, including the use of encoder-decoder networks. In the context of control, the need for model reduction arises out of the computational limitations of the controller, especially where the controlled system has fast dynamics. Thus, some researchers have trained NNs to learn the dynamics of the full-order model in more expensive offline computations, but once trained these NNs are capable of mapping the current system state to the optimal control action that could be applied at a greatly reduced computational cost.

2.1.1 Proper Orthogonal Decomposition

The Proper Orthogonal Decomposition (POD) [3, 4, 17–19] is a widely used model reduction method for dynamical systems which is based on the Singular Vector Decomposition (SVD). Its relative in statistics is well known as Principal Component Analysis, while similar SVD-based transformations have various names in the context of different domains (e.g. Karhunen–Loève expansion for stochastic processes). POD *decomposes* a

system model into more fundamental components in a way that is reminiscent of the separation of variables method for linear PDEs, where it is assumed that a function can be decomposed into independent spatial and temporal components, thereby revealing the building blocks of its dynamics. Following a similar concept, POD starts with the assumption that the solution $u(x,t)$ can be approximated by a finite sum of space-time separable basis functions:

$$u(x,t) \approx \sum_{i=1}^{M} \theta_{\mathbf{i}}(x)\ \mathbf{a_i}(t), \tag{2.1}$$

where $\theta_{\mathbf{i}}(x)$ are basis modes with only spatial dependence, while $\mathbf{a_i}(t)$ are coefficients corresponding to these modes, dependent only on time. The objective of POD is to find the most efficient basis functions, i.e. those that best approximate u for a given number of summation terms M. POD is data-driven, and the data set usually consists of values of u observed at various points of space and time, x and t. This process of taking "snapshots" of the system state was first formalised by Sirovich [2] for the study of turbulent flow. In applying POD to a system, the matrix $\mathbf{S}$ is usually first composed via such a series of snapshots of the system state. $\mathbf{S}$ is as tall as the number of state variables n and as wide as the number of snapshots taken m. These snapshots can be taken by experimental data, or by simulations of the full-order model [2, 9, 18–20].

$$\mathbf{S} = \begin{bmatrix} u(x_1,t_1) & u(x_1,t_2) & u(x_1,t_3) & \dots & u(x_1,t_m) \\ u(x_2,t_1) & u(x_2,t_2) & u(x_2,t_3) & \dots & u(x_2,t_m) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ u(x_n,t_1) & u(x_n,t_2) & u(x_n,t_3) & \dots & u(x_n,t_m) \end{bmatrix} \tag{2.2}$$

The SVD theorem states that, given any matrix (in this case only a real matrix is considered) $\mathbf{X} \in \mathbb{R}^{n\times m}$, it is possible to factorise $\mathbf{X}$ into 3 component matrices (named $\mathbf{U}$, $\mathbf{\Sigma}$, and $\mathbf{V}$), such that:

$$\mathbf{X}_{n\times m} = \mathbf{U}_{n\times n}\mathbf{\Sigma}_{n\times m}\mathbf{V}^T_{m\times m} \tag{2.3}$$

For example,

$$\begin{bmatrix} S_{11} & \dots \\ \dots & \dots \\ \dots & \dots \\ \dots & \dots \\ \dots & S_{nm} \end{bmatrix} = \begin{bmatrix} U_{11} & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & U_{nn} \end{bmatrix} \begin{bmatrix} \sigma_{11} & 0 \\ 0 & \ddots \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} V^T_{11} & \dots \\ \dots & V^T_{mm} \end{bmatrix} \tag{2.4}$$

In this case, $\mathbf{X}$ is the matrix of snapshots $\mathbf{S}$. $\mathbf{U}$ and $\mathbf{V}$ are orthogonal matrices, i.e. $\mathbf{U}^T\mathbf{U} = \mathbf{I}$ and $\mathbf{U}^{-1} = \mathbf{U}^T$ (likewise for $\mathbf{V}$), while $\mathbf{\Sigma}$ is a diagonal matrix of non-negative values, called the singular value matrix. When SVD is applied for POD, $\mathbf{\Sigma}$ is sorted in descending order, where the largest singular value inhabit σ_{11}, the next largest $\sigma_{\mathbf{22}}$ and so forth. $\mathbf{U}$ represents the spatial embedding of $\mathbf{S}$ onto a set of orthonormal bases

vectors, known as the POD modes. $\mathbf{V}$ represents the evolution of each modal element in time, and it is a set a orthonormal temporal coefficient vectors.

Each singular value in $\mathbf{\Sigma}$ represents the importance of its corresponding basis vector in $\mathbf{U}$, in terms of the amount of energy captured by each POD mode. This was used by Sirovich to identify spatial modes in which the highest fluid kinetic energies are contained [2]. The term energy may not refer to the system's actual physical energy; it can perhaps be understood also as the fraction of the data variance that can be accounted for by each basis vector. It is then possible to obtain an approximation of $\mathbf{S}$ by truncating the lower diagonal terms of $\mathbf{\Sigma}$, such that:

$$\mathbf{S}_{n\times m} \approx \tilde{\mathbf{U}}_{n\times r}\tilde{\mathbf{\Sigma}}_{r\times r}\tilde{\mathbf{V}}^T_{r\times m} = \sum_{i=1}^{r} \sigma_\mathbf{i}\mathbf{u_i}\mathbf{v_i^T} \tag{2.5}$$

where the right $n-r$ columns are truncated from $\mathbf{U}$, the right $m-r$ columns and bottom $n-r$ rows are truncated from $\mathbf{\Sigma}$, and the bottom $m-r$ rows are truncated from V^T. This gives the *rank-r reduced* approximation matrices $\tilde{\mathbf{U}}_{n\times r}$, $\tilde{\mathbf{\Sigma}}_{r\times r}$ and $\tilde{\mathbf{V}}^T_{r\times m}$. As $\mathbf{\Sigma}$ is diagonal, $\mathbf{S}$ can also be expressed as a summation of the singular values with their corresponding column vectors $\mathbf{u_i}$ and $\mathbf{v_i^T}$. The summation in Equation 2.5 thus fulfils the initial separation assumption in Equation 2.1, representing $u(x,t)$ as a summation of spatially and temporally separated basis vectors. It has been well-established [9, 19] that such a truncation always gives the best *rank-r* approximation to $\mathbf{X}$ in terms of the Euclidean norm between the original and reconstructed matrices: $\|\mathbf{X} - \tilde{\mathbf{X}}_r\|$.

2.1.1.1 Galerkin Projection

Consider a spatially discretised general PDE, where the system it governs has been reduced by POD using the method of snapshots. Having obtained from POD the series of most energetic to least energetic spatial modes, the full system dynamics now need to be projected onto these modes to give a reduced system. The *rank-r* SVD decomposition in Equation 2.5 can be substituted into the general form of the PDE, where L is a linear coefficient and $N(\cdot)$ is a nonlinear function [5, 19, 21, 22]:

$$\frac{d\mathbf{u}(t)}{dt} = L\mathbf{u}(t) + N(\mathbf{u}(t)), \qquad \text{where } \mathbf{u} \text{ was reduced as:} \tag{2.6a}$$

$$\mathbf{u}(x,t) \approx \tilde{\mathbf{U}}_{n\times r}\tilde{\mathbf{\Sigma}}_{r\times r}\tilde{\mathbf{V}}^T_{r\times m} = \tilde{\mathbf{U}}\mathbf{a}(t). \tag{2.6b}$$

$$\frac{d}{dt}\left(\tilde{\mathbf{U}}\mathbf{a}(t)\right) = L\tilde{\mathbf{U}}\mathbf{a}(t) + N\left(\tilde{\mathbf{U}}\mathbf{a}(t)\right) \qquad \text{As } \tilde{\mathbf{U}} \text{ is time-independent,} \tag{2.6c}$$

$$\tilde{\mathbf{U}}\frac{d\mathbf{a}(t)}{dt} = L\tilde{\mathbf{U}}\mathbf{a}(t) + N\left(\tilde{\mathbf{U}}\mathbf{a}(t)\right) \qquad \text{Since } \tilde{\mathbf{U}}^{-1} = \tilde{\mathbf{U}}^T, \tag{2.6d}$$

$$\frac{d\mathbf{a}(t)}{dt} = \tilde{\mathbf{U}}^T\left[L\tilde{\mathbf{U}}\mathbf{a}(t) + N\left(\tilde{\mathbf{U}}\mathbf{a}(t)\right)\right] \tag{2.6e}$$

Hence the dynamics of the reduced model can be obtained as:

$$\frac{d\mathbf{a}(t)}{dt} = \tilde{\mathbf{U}}^T L \tilde{\mathbf{U}} \mathbf{a}(t) + \tilde{\mathbf{U}}^T N\left(\tilde{\mathbf{U}}\mathbf{a}(t)\right) \qquad \text{for a nonlinear system, or} \qquad (2.6f)$$

$$\frac{d\mathbf{a}(t)}{dt} = \tilde{\mathbf{U}}^T L \tilde{\mathbf{U}} \mathbf{a}(t) \qquad \text{for a linear system.} \qquad (2.6g)$$

Thus, the system model is reduced to $\mathbf{u}(x,t) \approx \tilde{\mathbf{U}}\mathbf{a}(t)$, and the time-variance is entirely represented by Equation 2.6f. The transformation from Equation 2.6d to 2.6e takes advantage of the fact that SVD results in orthonormal POD modes. If the system is linear, then the time-independent linear coefficients in Equation 2.6g need only be computed offline once to obtain the reduced model.

2.1.1.2 Limitations of POD

On the other hand, it is apparent from Equation 2.6f that there is no way to compute a fixed value for the nonlinear term, as $N(\cdot)$ is time-dependent in $\mathbf{a}(t)$. This means that $N(\cdot)$ has to be updated as the system evolves, thus requiring online computations. These computations will outweigh the efficiency gained by model reduction as $N(\cdot)$ increases in complexity, reflecting a key limitation of applying POD onto nonlinear systems. To address this limitation, methods such as POD with the "Discrete Empirical Interpolation Method" (POD-DEIM) have been developed by various authors [23–25]. In POD-DEIM, the nonlinear function is itself approximated as a summation of linearly separable basis vectors and coefficients. Research on nonlinear POD is relatively lacking. Indeed, the POD-DEIM technique is comparatively recent, having been developed by Chaturantabut [24] in 2011.

Furthermore, as noted by Rowley [5, 21, 22], while POD can give the best *rank-r* approximation of a data set, the POD modes may not best describe the dynamics of the underlying system whose snapshots created that data set. This is because POD favours high energy modes, thus important dynamical features that are lower in energy may be discarded. Rowley cites the example of a cavity oscillation fluids system reduced using POD, where the POD modes directly relate to the amount of kinetic energy they carry. Thus, in the reduced model, the lower energy acoustic dynamics will be discarded in favour of the much more energetic hydrodynamic pressure changes. However, the lower energy acoustic waves are actually the interesting dynamical phenomena in the fluids system, reflecting that the modes that are higher in energy may not be more important to the system dynamics. Indeed, additional POD modes may actually decrease a reduced model's fidelity in preserving the original dynamics [5, 26].

2.1.2 Balanced Truncation

Moore formulated the Balanced Truncation (BT) model reduction method in 1981, for reducing linear time-invariant (LTI) systems while preserving as much of the input-output dynamics as possible for a *rank-r* truncation [27]. As introduced in Section

1.1.3.1, LTI systems have the state space representation:

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \tag{2.7a}$$

$$\mathbf{y} = \mathbf{C}\mathbf{x}, \tag{2.7b}$$

where $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{A} \in \mathbb{R}^{n\times n}$, $\mathbf{u} \in \mathbb{R}^m$, $\mathbf{B} \in \mathbb{R}^{n\times m}$, $\mathbf{y} \in \mathbb{R}^k$, and $\mathbf{C} \in \mathbb{R}^{k\times n}$. BT aims to find a coordinate transformation for the state vector $\mathbf{x}$ in the form $\mathbf{x} = \mathbf{T}\mathbf{z}$, where $\mathbf{T} \in \mathbb{R}^{n\times n}$ is an invertible matrix and $\mathbf{z} \in \mathbb{R}^n$ are the new coordinates, such that the system is balanced in its inputs and outputs. $\mathbf{T}$ and $\mathbf{z}$ would be sorted by how much each transformed state contributes to the system's input-output dynamics, thus allowing one to truncate the less important transformed states, thereby obtaining a reduced model [5, 9, 25]. Substituting the transformation into the state space model, and using the same notations as in [9],

$$\mathbf{x} = \mathbf{T}\mathbf{z} \tag{2.8a}$$

$$\dot{\mathbf{x}} = \mathbf{T}\dot{\mathbf{z}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}, \qquad \mathbf{y} = \mathbf{C}\mathbf{x} \tag{2.8b}$$

$$\mathbf{T}\dot{\mathbf{z}} = \mathbf{A}\mathbf{T}\mathbf{z} + \mathbf{B}\mathbf{u} \tag{2.8c}$$

$$\dot{\mathbf{z}} = \mathbf{T}^{-1}\mathbf{A}\mathbf{T}\mathbf{z} + \mathbf{T}^{-1}\mathbf{B}\mathbf{u}, \qquad \mathbf{y} = \mathbf{C}\mathbf{T}\mathbf{z} \tag{2.8d}$$

If a reduced system of *r* states is required, then $\mathbf{T}$ and $\mathbf{z}$ can be truncated:

$$\dot{\mathbf{z}}_\mathbf{r} = {\mathbf{T}_\mathbf{r}}^{-1}\mathbf{A}\mathbf{T}_\mathbf{r}\mathbf{z}_\mathbf{r} + {\mathbf{T}_\mathbf{r}}^{-1}\mathbf{B}\mathbf{u}, \qquad \mathbf{y} = \mathbf{C}\mathbf{T}_\mathbf{r}\mathbf{z}_\mathbf{r}, \tag{2.8e}$$

where $\mathbf{T}_\mathbf{r} \in \mathbb{R}^{n\times r}$ and $\mathbf{z}_\mathbf{r} \in \mathbb{R}^r$ keeps only the first *r* columns of $\mathbf{T}$ and first *r* rows of $\mathbf{z}$. Simplifying by substituting $\tilde{\mathbf{x}} = \mathbf{z}_\mathbf{r}$, $\tilde{\mathbf{A}} = {\mathbf{T}_\mathbf{r}}^{-1}\mathbf{A}\mathbf{T}_\mathbf{r}$, $\tilde{\mathbf{B}} = {\mathbf{T}_\mathbf{r}}^{-1}\mathbf{B}$, and $\tilde{\mathbf{C}} = \mathbf{C}\mathbf{T}_\mathbf{r}$,

$$\dot{\tilde{\mathbf{x}}} = \tilde{\mathbf{A}}\tilde{\mathbf{x}} + \tilde{\mathbf{B}}\mathbf{u}, \qquad \mathbf{y} = \tilde{\mathbf{C}}\tilde{\mathbf{x}}, \tag{2.8f}$$

hence obtaining the reduced model. The input-output dynamics of a LTI system can be characterised by its controllability and observability Gramians, $\mathbf{W_c}$ and $\mathbf{W_o}$ respectively. $\mathbf{W_c}$ represents how much each system state responds to an input, while $\mathbf{W_o}$ represents how much each system output responds to a system state. Thus, after the balancing transformation, the least controllable state in $\mathbf{z}$ should also be the least observable, and vice versa; this implies that $\mathbf{W_c}$ and $\mathbf{W_o}$ need to be equal: $\mathbf{W_c} = \mathbf{W_o} = \mathbf{\Sigma}$ [5, 9, 25]. The Gramians for a continuous-time LTI system are given as [5, 27]:

$$\mathbf{W_c} = \int_0^\infty \exp(\mathbf{A}t)\mathbf{B}\mathbf{B}^\mathbf{T}\exp(\mathbf{A}^\mathbf{T}t)\,dt \tag{2.9a}$$

$$\mathbf{W_o} = \int_0^\infty \exp(\mathbf{A}^\mathbf{T}t)\mathbf{C}^\mathbf{T}\mathbf{C}\exp(\mathbf{A}t)\,dt, \tag{2.9b}$$

whose solutions can be obtained through solving these Lyapunov equations [25]:

$$\mathbf{A}\mathbf{W_c} + \mathbf{W_c}\mathbf{A}^\mathbf{T} + \mathbf{B}\mathbf{B}^\mathbf{T} = 0 \tag{2.9c}$$

$$\mathbf{A^T W_o} + \mathbf{W_o A} + \mathbf{C^T C} = 0 \tag{2.9d}$$

The Gramians can then be used to solve for $\mathbf{T}$, using the eigendecomposition [5, 9, 25]:

$$\mathbf{W_c} = \mathbf{W_o} = \mathbf{\Sigma} \tag{2.9e}$$

$$\mathbf{W_c W_o T} = \mathbf{T \Sigma}^2 \tag{2.9f}$$

It is worth noting that solving Lyapunov equations for high-dimensional systems is computationally challenging: the complexity scales at $O(n^3)$ [27, 28], where n is the number of system states and $n \times n$ is the dimension of both $\mathbf{W_c}$ and $\mathbf{W_o}$. Moore thus devised to empirically approximate the Gramians via a snapshot method similar to that in POD. This is done via impulse response measurements, where a unit impulse is applied to a system whose initial states are at 0. Illustrating the effect of an unit impulse on a discrete-time system $\mathbf{x}_{t+1} = \mathbf{A x}_t + \mathbf{B u}_t$, where $\mathbf{x}_0 = 0$ and $\mathbf{u}_0 = \mathbf{I}$:

$$\mathbf{x}_1 = \mathbf{B} \tag{2.10a}$$

$$\mathbf{x}_2 = \mathbf{A x}_1 = \mathbf{AB} \tag{2.10b}$$

$$\mathbf{x}_3 = \mathbf{A x}_2 = \mathbf{A^2 B}, \quad \text{and so on.} \tag{2.10c}$$

Thus obtaining the snapshots describing the effect of a *control* impulse on the system, forming an empirical controllability matrix: $\mathcal{C} = [\mathbf{B\ AB\ A^2 B\ \dots A^{m-1} B}]$, where m is the number of snapshots [9, 27]. Additionally, Moore introduced the mathematical construct of an *adjoint* system, through which one can identify the system states that have the most significant effect on the output $\mathbf{y}$. The adjoint system model is given as $\mathbf{x}_{t+1} = \mathbf{A^T x}_t + \mathbf{C^T y}_t$ [9, 27]. Using this artificial construct, it is possible to impulse the original system output and observe its effect on the system states, deriving an empirical observability matrix: $\mathcal{O} = [\mathbf{C\ CA\ CA^2\ \dots CA^{m-1}}]^{\mathbf{T}}$, which is also derived via the snapshot method illustrated in Equations 2.10a to 2.10c.

Moore then proved that the controllability and observability Gramians can be approximated using the empirical $\mathcal{C}$ and $\mathcal{O}$ matrices [5, 9, 27], such that:

$$\mathbf{W_c} \approx \mathcal{C}\mathcal{C}^T \tag{2.11a}$$

$$\mathbf{W_o} \approx \mathcal{O}^T\mathcal{O} \tag{2.11b}$$

The approximations become more accurate as the intervals between snapshots decrease and as the duration over which snapshots are taken increase. Thus, this method of empirical Gramians circumvents the need to compute solutions to the Lyapunov equations. Taking these snapshots are still computationally expensive for large systems, albeit less so than solving the Lyapunov equations. Developed by later authors and built on Moore's work, the Balanced Proper Orthogonal Decomposition (BPOD) was formulated, in part motivated by the time-consuming procedure of deriving the empirical Gramians.

2.1.3 Balanced Proper Orthogonal Decomposition

Willcox et al. [25] and Rowley [5, 26] extended Moore's work, each deriving more efficient adaptions for BT. In particular, Rowley proved that a balanced transformation can be made without even computing empirical Gramians. In [5], Rowley devised a transformation method by applying the SVD to the Hankel matrix $\mathbf{H} = \mathcal{O}^{\mathbf{T}}\mathcal{C}$, such that:

$$\mathbf{H} = \mathcal{O}^{\mathbf{T}}\mathcal{C} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^{\mathbf{T}} \tag{2.12a}$$

$$\approx \tilde{\mathbf{U}}\tilde{\boldsymbol{\Sigma}}\tilde{\mathbf{V}}^{\mathbf{T}} \tag{2.12b}$$

Rowley then defines 2 matrices that will be the projection bases for the full system model $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$: $\mathbf{T}$ and $\mathbf{S}$. The reduced bases $\tilde{\mathbf{T}}$ and $\tilde{\mathbf{S}}$ can be obtained from the truncated SVD in Equation 2.12b:

$$\mathbf{T} = \mathcal{C}\mathbf{V}\boldsymbol{\Sigma}^{-\frac{1}{2}}, \qquad \tilde{\mathbf{T}} = \mathcal{C}\tilde{\mathbf{V}}\tilde{\boldsymbol{\Sigma}}^{-\frac{1}{2}} \tag{2.12c}$$

$$\mathbf{S} = \mathbf{T}^{-1} = \boldsymbol{\Sigma}^{-\frac{1}{2}}\mathbf{U}\mathcal{O}, \qquad \tilde{\mathbf{S}} = \tilde{\boldsymbol{\Sigma}}^{-\frac{1}{2}}\tilde{\mathbf{U}}\mathcal{O} \tag{2.12d}$$

The reduced model is obtained by projecting the full model onto the reduced bases [5]:

$$\tilde{\mathbf{A}} = \tilde{\mathbf{S}}^{\mathbf{T}}\mathbf{A}\tilde{\mathbf{T}} \tag{2.12e}$$

$$\tilde{\mathbf{B}} = \tilde{\mathbf{S}}^{\mathbf{T}}\mathbf{B} \tag{2.12f}$$

$$\tilde{\mathbf{C}} = \mathbf{C}\tilde{\mathbf{T}} \tag{2.12g}$$

In Section 3.3, BPOD is implemented to obtain a ROM that will be used as a benchmark. The BPOD MATLAB tutorial in [9] was used as reference for this implementation.

2.1.4 Encoder-Decoder Neural Networks (Autoencoders)

Encoder-decoder Neural Networks, also called autoencoders (Figure 2.1), are a class of self-supervised neural networks trained to learn an identity function for its inputs, i.e. to predict outputs that are as close to its inputs as possible. By constricting the hidden layers, the network is encouraged to learn a fidelity-preserving low-dimensional transformation of the original state variables. Its loss function is the error between its full state space input and its output, projected back from the low-dimensional latent variables. Hence, autoencoders can function as a method for model order reduction. In fact, it has been established by several authors [29–31] that a linearly-activated autoencoder with one hidden layer can extract the same basis modes as POD.

Autoencoders have been widely applied to reduce the dimensionality of image-based data, for example on the MNIST digits data set and the Olivetti face data set by various authors [32–35], while their application to high-dimensional dynamical systems for model reduction is more recent. Agostini [36] implemented an autoencoder to study 2D unsteady fluid flow around a cylinder, where the full-order system model has

spatial dimensions of 256×88 arising from the mesh discretisation applied, and a temporal dimension of 12 from the number of snapshots taken. This was constricted to the bottleneck layer of the autoencoder comprising only 3 nodes. Despite this aggressive compression, the full system could be reconstructed with minimal error, far outperforming the *rank-3* POD reduction used as the benchmark. Other authors [37–40] applied similar methods to both linear and nonlinear dynamical systems, generally reaching the same conclusion that autoencoders outperforms a POD reduction of the same rank.

2.1.4.1 Possible improvements to existing methods, pertaining to Objective 5.

Although autoencoders can project data onto reduced dimensions, it is not within their scope to identify the dynamics between the reduced variables, nor between the reduced variables and the control inputs. Thus, for autoencoders to be useful in the study of dynamical systems, especially for control, they must be used in conjunction with a method to discover the dynamical model of the reduced system. Then, an optimal control problem can be formulated in terms of the reduced system, such that it can be solved more efficiently for the real-time MPC control of a high-dimensional system.

On this note, Agostini [36] used a spectral clustering algorithm to discover the dynamics of the autoencoder-reduced fluids system, in the form of a stochastic Markov chain model. Champion et al. [41] combined an autoencoder with SINDy [42] (Section 2.1.5), a tool for data-driven model discovery, to derive the dynamical relations between the autoencoder-reduced variables. They obtained good approximates of the true models of three systems: the Lorenz system, a reaction-diffusion system, and a nonlinear pendulum. However, these were not controlled systems. Hence, while Champion et al.'s work demonstrates the feasibility of using autoencoder with SINDy to model reduced dynamical systems, there is an additional challenge in using their approach for control: to use the reduced model for control, it is necessary to map also the control objective and possible constraint(s) to the reduced space.

For example, for the optimal control of the case study system (Section 3.1.1), it is necessary to translate the setpoint and controller objectives, and the state constraints, to the reduced space. This is not straightforward. To illustrate, if a system's full state variables, $x_1, x_2, \ldots, x_{10}$, is reduced to 2 variables, z_1, z_2, by an autoencoder, then both reduced variables are a function of possibly all 10 variables in the full space: $z_1 = f(x_1, x_2, \ldots, x_{10})$ and $z_2 = g(x_1, x_2, \ldots, x_{10})$. Hence, in the presence of state constraint(s) on the system (e.g. $x_1 \leq 300$), it is a challenge to determine the permissible values for z_1 and z_2 that would realise the constraint(s) in the full space without placing incorrect constraints on the free variables. Research is lacking in using autoencoders with model discovery for control, and even more so in the context of constrained systems. This will be tackled as part of Objective 5 in this thesis.

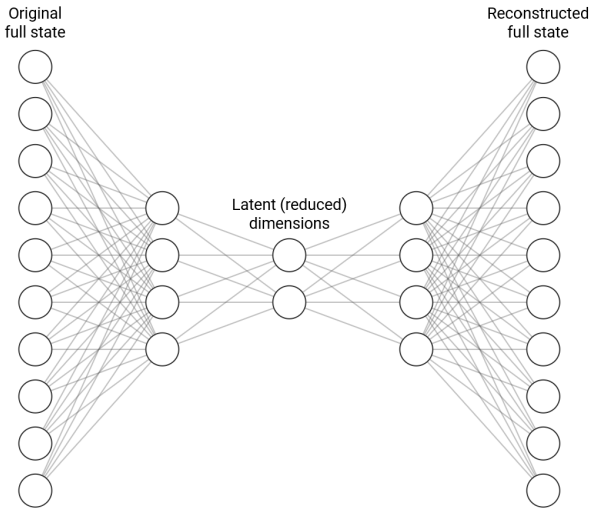


Figure 2.1: An example of an encoder-decoder neural network structure

2.1.5 Systems Identification with SINDy/SINDyc

SINDy, "Sparse Identification of Nonlinear Dynamical Systems", is a data-driven system identification tool developed by Brunton et al. [42–44]. The function of SINDy is to discover a best-fit function governing the evolution of the system state, $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}(t))$, given the state trajectory data of the system. Borrowing the subsequent notations from Brunton et al., SINDy works by finding the best candidate vector of coefficients $\boldsymbol{\Xi}$ for the equation $\dot{\mathbf{X}} = \boldsymbol{\Theta}(\mathbf{X})\boldsymbol{\Xi}$. It uses LASSO ("least absolute shrinkage and selection operator") regularisation to encourage sparsity in the fitted coefficients $\boldsymbol{\Xi}$ such that simpler dynamics are preferred. $\mathbf{X}$ and $\dot{\mathbf{X}}$ are known matrices of the snapshots of the system state and state derivatives respectively; they are of the same format as the snapshot matrix $\mathbf{S}$ described in the POD method of snapshots (Equation 2.2). $\boldsymbol{\Theta}$, the matrix of library functions, is arguably the core of SINDy. It is a user-customisable library of various functions of $\mathbf{X}$, expressed by Brunton et al. [42, 43] in the following form:

$$\boldsymbol{\Theta}(\mathbf{X}) = \begin{bmatrix} | & | & | & | & & | & | & \\ 1 & \mathbf{X} & \mathbf{X^{P_2}} & \mathbf{X^{P_3}} & \dots & \sin(\mathbf{X}) & \cos(\mathbf{X}) & \dots \\ | & | & | & | & & | & | & \end{bmatrix}, \tag{2.13}$$

where $\mathbf{X^{P_i}}$ are themselves matrices of polynomial terms of degree i, possibly including interaction terms, like x_1x_2 in $\mathbf{X^{P_2}}$:

$$\mathbf{X}^{\mathbf{P_2}} = \begin{bmatrix} x_1^2(t_1) & x_1(t_1)x_2(t_1) & x_2^2(t_1) & \dots & x_n^2(t_1) \\ x_1^2(t_2) & x_1(t_2)x_2(t_2) & x_2^2(t_2) & \dots & x_n^2(t_2) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_1^2(t_m) & x_1(t_m)x_2(t_m) & x_2^2(t_m) & \dots & x_n^2(t_m) \end{bmatrix} \tag{2.14}$$

It is apparent that $\boldsymbol{\Theta}$ scales factorially [42] with the dimensionality of the system n, therefore the number of possible permutations in the coefficient vector $\boldsymbol{\Xi}$ for a large system may be intractably high. Indeed, as noted by its authors, SINDy is unlikely to perform well in discovering accurate dynamics for high-dimensional systems, and they recommend the use of dimensionality reduction such that SINDy is given the reduced system to fit [42]. This is an important caveat as it creates a trade-off between the fidelity of the reduced system and the accuracy of its fitted dynamics: a model that is aggressively reduced is ideal for SINDy but may lose accuracy in its approximation of the full system.

SINDy was later extended by Brunton et al. to be able to fit the function describing $\dot{\mathbf{x}}$ for systems with control, $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t))$. This version of SINDy is called SINDYc [43]. Its methodology is identical to SINDy, with the state matrix $\mathbf{X}$ effectively extended to include control variables. As described in Section 2.1.4, Champion et al. used SINDy, not SINDYc, to discover the governing equations for the reduced state variables for *uncontrolled* high-dimensional systems. The challenges in adapting this approach for optimal control are discussed in Section 2.1.4 and will be addressed as part of Objective 5 of this thesis.

2.2 Neural Networks for Control

Cybenko [45], Hornik et al. [46, 47], and other authors [48–50] have proved the capability of multi-layered neural networks as universal approximators. Thus, various authors have found neural networks attractive in using them to replace functions that are expensive to compute. In the context of dynamical modelling and control, neural networks have been used to approximate the state evolution function and to directly approximate the solution of the optimal control problem. This section provides a literature review of using neural networks for control in the context of high-dimensional systems. The applications of autoencoder neural networks for model reduction without the aspect of control has been discussed in Section 2.1.4.

Draeger et al. [51] taught a neural network to predict the pH value in a neutralisation reactor at sampling time step $t + 1$, using the pH values and sodium hydroxide flow rate over the last 5 time steps as inputs. As such, the network learns the dynamics, i.e. evolution functions of the reactor system. Draeger et al. then used this network as a black-box system model in a model predictive controller. Hence, while the model predictive controller can query the neural network for the next system state given a

control action, the optimal control problem itself is not solved: the controller still have to compute the optimal control policy given the system state. As their work aimed to replace the plant's existing PI-controller by using the model predictive controller on the neural net system model, addressing this challenge was not within their scope.

Long short-term memory (LSTM) networks have also been used for the similar purpose of learning the system dynamics: Jeon et al. [52] used a LSTM network as the building energy model and Particle Swarm Optimisation (PSO) for discovering optimal control actions for the control of building temperatures. Their motivation for this was also for the efficient control of a complex model: the LSTM network and PSO together replace the original MPC controller, which required a long time to compute the optimal control trajectory due to the complexity of the original building energy model. For a similar problem of HVAC control, Zou et al. [53] trained the controller using Reinforcement Learning (RL) instead of using PSO to search for optimal controls, but likewise used a LSTM network for system approximation. Various other authors [54–57] took similar approaches, wherein the neural network is trained to serve as a black-box system model for a MPC, particularly when the original system dynamics are nonlinear.

Another approach that developed more recently supplants the need for a controller to explicitly compute the optimal control policy, by using a neural network to directly predict the optimal action to take given the system state. Thus, the trained neural network directly acts as the controller. This approach borrows the idea from explicit MPC, wherein the optimal control actions for all system states in the expected operation range are pre-computed offline and stored for reference, thereby replacing the optimal control problem with much simplified function evaluation. For example, Maddalena et al. [58] trained a neural network for the voltage-current control of a multicell DC-DC converter, reporting over 80% savings in the time required for computing the control action online, at minimal expense to the steady-state error performance. Various authors [59–61] have applied this concept to other systems with success. However, the neural network still needs to be taught by the original model predictive controller. Furthermore, due to the relatively poor extrapolation performance of neural networks beyond their data support regime [62–64], it needs to be taught across all the possible system states where control is needed.

It is pertinent to note that the authors who applied neural networks to control were generally motivated by the need to efficiently control a complex system. As this thesis project shares the same aim, learning points can be drawn from the advantages and disadvantages of their methods. The caveat of using a neural network to approximate the system model is that the control problem itself is not solved: the controller still need to call simulations of the system via the neural network to compute the optimal control trajectory, though now these simulations can be performed faster. On the other hand, replacing the controller with a neural network trained on the known optimal control(s) for each system state requires an MPC controller to first act as a teacher. Hence, a solution that can act as an optimal controller without having to learn from the MPC

reference trajectory is desirable. On this note, concepts from Reinforcement Learning (RL) may be applicable:

2.2.1 Applicable Concepts from Reinforcement Learning

Various authors, including Lin et al., Görges, Ernst et al., and Sutton et al. [65–67], have observed parallels between Model Predictive Control and Reinforcement Learning: both MPC and RL aim to maximise a value (or minimise a cost) over a duration, usually advancing in discrete time steps. MPC solves this optimisation problem based on the closed-form expression of the cost function and the model of the system dynamics. Model-based RL do not require explicit foreknowledge of the system dynamics, learning the transition and reward functions of its Markov Decision Process—analogous to the MPC's system model and cost function respectively—via interaction experience with the system [67]. In contrast, model-free RL algorithms such as Q-learning (QL) [68, 69] generally seek to directly maximise the cumulative reward of each action without learning the system model.

In Q-learning, a Q-function $Q(x, a)$ that gives the Q-value of an action in an environment state is built as the agent interacts with the environment and learns. The Q-value represents the expected cumulative reward from making an action a while in state x and making actions in subsequent steps that result in the maximum Q-value in each of those subsequent steps: $Q(x_t, a) = r(x_t, a) + \gamma \max Q(x_{t+1}, a') \,|\, x_t$, where r is the reward from this time step and γ is a discount factor balancing the relative importance of immediate and future rewards. This is a recursively defined Bellman function: the Q-values can be updated by value iteration while storing them in a lookup table; for higher-dimensionality or continuous state space problems, the Q-function can be approximated using a neural network, which is trained as the agent gathers more experience.

A RL problem can be solved by value iteration or policy iteration. In the context of Q-learning, the goal of value iteration is to obtain an accurate optimal Q-function, such that an action that maximises this Q-function given the state can be determined to be optimal. This is done starting with a random Q-function, which is improved iteratively to converge towards predicting the true Q-value of a state-action pair. In policy iteration, the RL agent begins with a random policy, then evaluates and implicitly updates the Q-function corresponding to this policy. It then tries a new policy that maximises the Q-function and does so until no further improvement can be found.

Bradtke et al. [70] proved mathematically that a policy iteration approach in Q-learning can discover control policies that would converge to the explicitly solved optimal controls of a linear–quadratic regulator (LQR). Building on this theoretical work, Kiumarsi et al. [71] implemented a QL-based controller for an LQR system with 2 states and 1 control input, while Lee et al. [72] did likewise for a system with 4 states and 1 control input. Luo et al. [73] extended the work of previous authors on linear discrete-

time systems to general nonlinear continuous-time systems. They further proved that a value iteration approach will converge to the optimal Q-function and is also feasible for discovering optimal controls. They tested both their policy and value iteration implementations on 2 low-dimensional systems—a linear system with 3 states and a nonlinear system with 2 states, both with only 1 controller—where they were found to be effective.

2.2.1.1 Possible improvements to existing methods, pertaining to Objective 4.

However, research is lacking in the implementation of a QL-based controller for a high-dimensional nonlinear system. Furthermore, and importantly, the implementations described are not formulated for *constrained* systems. For the control of systems that must stay within state constraints, further work is needed to synthesise the concept of Q-values with penalties for control actions that may lead to a violation of said constraints. In addressing both of these shortcomings in current literature, i.e. high-dimensional and constrained systems, there is a novel possibility of using an encoder network (Section 2.1.4) with a control network that takes into account system constraints. The encoder network would reduce the state space of a high-dimensional nonlinear system, then the augmented control network would predict both the value of a state-action pair and whether this state-action pair may lead to a subsequent constraint violation. Thus, the optimal control action(s) for a state can be identified by searching the control network for the actions that are expected to maximise value, given the state.

Experience Replay in Q-learning A further augmentation to the proposed implementation involves using *experience replay* in the training process. Experience replay is a concept first applied in Q-learning [74] where the RL agent remembers its experience at each time step, comprising its current state, action performed, reward attained and the next state: $e_t = (x_t, a_t, r_t, x_{t+1})$. With experience replay, the Q network, i.e. the Q-function approximator, is trained on samples from its memory of both immediate and past experiences. In QL, experience replay helps to break correlations between sequential experiences that may be similar, which are detrimental to the approximation accuracy of the Q network. It is implemented in this thesis for training the value approximation neural network in Chapter 3 so that the quality of the training data can be improved after each training round. With this approach, it is expected that the value approximation neural network can learn to generate system trajectories of increasing quality from initially random simulation data. Details on the experience replay training process will be given in Chapter 3.

Chapter 3

Value Approximation Neural Network for Optimal Control

3.1 Case Study System

3.1.1 Thermal Control of a 1D Thin Rod (Heat Equation)

The dynamical system used as the main case study in this thesis is the heat flow and control within a 1D thin rod. This system is modelled by the familiar heat equation:

$$\frac{\partial T}{\partial t} = \alpha \frac{\partial^2 T}{\partial x^2}, \tag{3.1}$$

where $T(x,t)$ is the temperature at length x of the rod at time t; T used instead of the conventional u to avoid confusion with the control inputs $\mathbf{u}$. α is the thermal diffusivity constant, characterising the ease of heat transfer in the material. This system, while simple to understand, can be spatially discretised to arbitrarily many dimensions, thus simulating a complex model on which model reduction methods can be applied. Furthermore, it can be modelled to take Dirichlet boundary controls at each end, where 2 heater/cooler controllers, $\mathbf{u}_1$ and $\mathbf{u}_2$ can input heat into/draw heat away from its ends:

$$\begin{aligned} T(0,t) &= \mathbf{u}_1(t) \\ T(1,t) &= \mathbf{u}_2(t) \end{aligned} \tag{3.2}$$

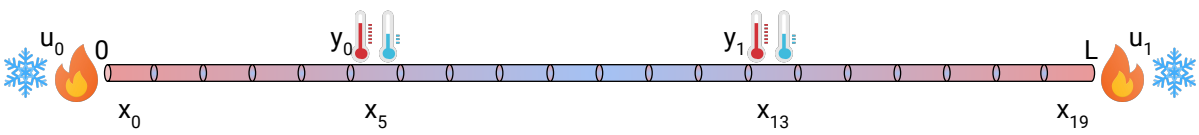


Figure 3.1: Illustration of the spatial-discretised 1D thin rod heat control system

For the case study system, the temperature sensors i.e. system outputs $\mathbf{y}$ are located approximately at $\frac{L}{3}$ and $\frac{2L}{3}$ along the rod. The length of the rod L was set as 1 m. After spatial discretisation into 20 segments [75], these outputs are located at $\mathbf{x}_5$ and $\mathbf{x}_{13}$ (Figure 3.1). Therefore, this is a 2-input 2-output linear system with 20 dimensions. Initially, the rod is modelled to be at a uniform temperature of 273 K. The system can be given in its state-space representation:

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t) \qquad & \mathbf{x}(0) &= 273 \\ \mathbf{y} &= \mathbf{C}\mathbf{x}(t), & \text{where:} \end{aligned} \tag{3.3}$$

$$\mathbf{A} = \frac{\alpha}{l^2}\begin{bmatrix} -2 & 1 & 0 & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & 0 & 1 & -2 \end{bmatrix} \in \mathbb{R}^{20\times 20} \qquad \mathbf{B} = \frac{\alpha}{l^2}\begin{bmatrix} 1 & 0 \\ 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{20\times 2}$$

$$\mathbf{C} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & \dots & & & & & & & \\ & & & & & & \dots & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \in \mathbb{R}^{2\times 20} \tag{3.4}$$

$l = \frac{L}{20}$ is the length of each rod segment while the thermal diffusivity α is set as 0.3. Figure 3.1 illustrates the system model. Subsequently, for testing the performance of the implementations derived in this thesis on a nonlinear system, a radiative loss term was added to Equation 3.1:

$$\frac{\partial T}{\partial t} = \alpha\frac{\partial^2 T}{\partial x^2} - k(T^4 - v^4), \tag{3.5}$$

such that each discrete element now loses heat at a rate of $k(T^4 - v^4)$, where $v = 273\ K$ is the temperature of the environment and k is a constant depending on the emissivity and specific heat capacity of the rod segments; it was set as 2.8×10^{-8}, which was found to cause enough heat loss to challenge the controllers.

The system is controlled over a control horizon $T = 1$ s, with the objective primarily to keep the temperature at the $\frac{1}{3}$ position of the rod near $\mathbf{x}_5^* = 303$ K and the temperature at the $\frac{2}{3}$ position near $\mathbf{x}_{13}^* = 333$ K, and secondarily to save on heating/cooling costs while reaching the temperature setpoints:

$$\underset{\mathbf{u}}{\text{minimise}} \quad J(\mathbf{x},\mathbf{u}) = \int_0^T w_x \left[(\mathbf{x}_5(t)-\mathbf{x}_5^*)^2 + (\mathbf{x}_{13}(t)-\mathbf{x}_{13}^*)^2\right] + w_u \left[(\mathbf{u}_0(t)-v)^2 + (\mathbf{u}_1(t)-v)^2\right]\ dt \tag{3.6a}$$

$$\begin{aligned}
\text{subject to} \quad & \dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}, && \text{or subsequently with radiative loss,} \\
& \dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} + \mathbf{r}(\mathbf{x}), && \text{where } \mathbf{r}_i(\mathbf{x}_i) = -kc(\mathbf{x}_i^4 - v^4),\ v = 273. \\
& \mathbf{x}_5(t) \leq 313, && \text{or subsequently,} \\
& \mathbf{x}_5(t) \leq 5t^2 + 10t + 293, && \text{to test the handling of path constraints.} \\
& 173 \leq \mathbf{u}(t) \leq 373, && \text{or subsequently with radiative loss,} \\
& 173 \leq \mathbf{u}(t) \leq 473, && \text{to compensate for the radiative loss.}
\end{aligned} \tag{3.6b}$$

$w_x = 0.995$ is the weighing factor for the state setpoints and $w_c = 0.005$ is the weighing factor for energy costs of the controllers, in terms of the thermal gradient relative to the environmental temperature $v = 273$ K that either controller has to create.

3.1.2 System Simulation to Generate Training Data

Trajectory data for the system is needed to train the neural network controller (NN controller) described in Section 3.2. The data is obtained by sending random control input signals u_0 and u_1 to the system as it is being simulated in CASADI. The initial training data set consists of 240 trajectories and 10 snapshots of the system state at 0.1 s time steps for each trajectory. These trajectories are stored in the sequence of 3 trajectories that satisfy the system state constraint, then 1 trajectory in which the state constraint is violated (hyperparameter 2 in Table 3.1). Thus, there are 2400 rows of training data in total, with 30 rows of 3 passed trajectories, then 10 rows of 1 failed trajectory, and so on. The data generation procedure is illustrated in Algorithm 1.

Algorithm 1: Generate N random trajectories, of which $\frac{N}{4}$ violate constraints

```
repeat
    repeat
        Generate trajectory with random controls
    until 3 trajectories that satisfies constraints at all time steps are obtained
    repeat
        Generate trajectory with random controls
    until 1 trajectory that do not satisfy constraints at any time step is obtained
    Append 4 trajectories to dataframe
until N trajectories are obtained
```

3.2 Neural Network Design and Implementation

The design of a value approximation neural network to act as an optimal controller is detailed in this section. The neural network itself predicts the optimal value function, i.e. the "cost-to-go", and the expected constraint deviation when given the state of the system $\mathbf{x}$ and control actions $\mathbf{u}$. By using a basin-hopping minimisation search (Section 3.2.3.1), the optimal control actions $\mathbf{u}^*$ can be found for a given system state $\mathbf{x}_t$. After $\mathbf{u}^*$ is applied, the system advances to its next state $\mathbf{x}_{t+1}$, where the search for optimal controls is repeated.

The combination of the basin-hopping search with the value approximation neural network for control is subsequently referred to as the *NN controller*. In discussions that relate to the neural network per se, for example in its hyperparameter configuration, it is referred to as the *value NN* or the *cost-to-go NN*.

Figure 3.2 illustrates the structure of the cost-to-go NN: it consists of a model reducer (encoder neural network) and the optimal value and constraint approximator. The control inputs are decoupled from the model reducer. Thus, the optimal control actions are actually searched with respect to the reduced state variables, $\tilde{\mathbf{x}}$. The primary purpose of adopting this structure is to demonstrate that an encoder can reduce the state variables of a high-dimensional system while preserving sufficient accuracy for $\tilde{\mathbf{x}}$ to be used for optimal control. The training process for the NN controller, which involves *experience replay* (Section 2.2.1.1), is summarised in Algorithm 2. The rest of this section explains the procedures in Algorithm 2 in detail.

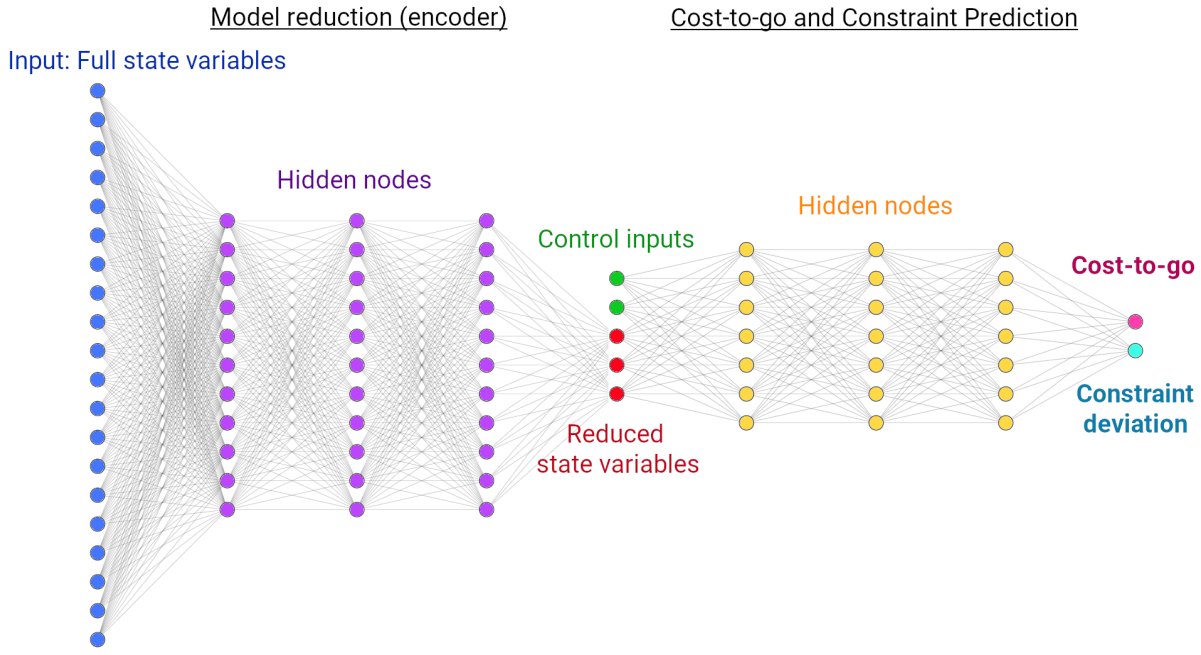


Figure 3.2: Structure of the value function and constraint approximation neural network

Algorithm 2: Train the cost-to-go NN with experience replay

Component functions:
Feasible *(system state* $\mathbf{x}$*, controller actions* $\mathbf{u}$*)*:
    if *neural network predicts constraints to be satisfied* **then**
        return True
    else
        return False

Cost-to-go V *(system state* $\mathbf{x}$*, controller action* $\mathbf{u}$*)*:
    $V \leftarrow$ neural network prediction for cost-to-go
    if `Feasible(`$\mathbf{x}, \mathbf{u}$`)` **then**
        return V
    else
        return $V\times$ Penalty Constant

Basinhopper *(system state* $\mathbf{x}$*)*:
    $\mathbf{u}^* \leftarrow \underset{\mathbf{u}}{\operatorname{argmin}}\ (V(\mathbf{x}, \mathbf{u} \mid \mathbf{x}))$ via Powell's Method (local minima)
    for $i \leftarrow 0$ **to** *Max Iterations* **do**
        $\mathbf{u} \leftarrow$ `Perturb(`$\mathbf{u}^*$`)` by "hopping" to new coordinates
        $\mathbf{u_{hop}} \leftarrow \underset{\mathbf{u}}{\operatorname{argmin}}\ (V(\mathbf{x}, \mathbf{u} \mid \mathbf{x}))$ via Powell's Method (local minima)
        if $V(\mathbf{x}, \mathbf{u_{hop}}) < V(\mathbf{x}, \mathbf{u}^*)$ **then**
            $\mathbf{u}^* \leftarrow \mathbf{u_{hop}}$
    return $\mathbf{u}^*$

Run experience replay for R *rounds, each containing* r *runs:*
for $round \leftarrow 0$ **to** R **do**
    for $run \leftarrow 0$ **to** r **do**
        $\mathbf{x} \leftarrow$ initial system state
        for $t \leftarrow 0$ **to** T **do**
            $\mathbf{u}^* \leftarrow$ `Basinhopper(`$\mathbf{x}$`)`
            $\mathbf{x} \leftarrow$ `Simulate(`$\mathbf{x}$, $\mathbf{u}^*$`)`
            if *size of replay buffer exceeds N* **then**
                Overwrite earliest entry with $\mathbf{x}$ and $\mathbf{u}^*$
            else
                Append $\mathbf{x}$ and $\mathbf{u}^*$ to replay buffer
        Record simulation results and the objective function score of this run
    Retrain neural network on refreshed replay buffer

3.2.1 Implementation of the Cost-to-go Function Node

As seen in Figure 3.2, the neural network is tasked to approximate the optimal cost-to-go / optimal value function $V^*(\mathbf{x}, \mathbf{u})$, such that the optimal control policy $\mathbf{u}^*$ is defined as:

$$\mathbf{u}^* = \underset{\mathbf{u}}{\operatorname{argmin}}\ V^*(\mathbf{x}, \mathbf{u}), \quad \text{where} \tag{3.7a}$$

$$V^*(\mathbf{x}, \mathbf{u}) = \min_{\mathbf{u}} \int_t^T w_x \left[(\mathbf{x}_5(t) - \mathbf{x}_5^*)^2 + (\mathbf{x}_{13}(t) - \mathbf{x}_{13}^*)^2\right] + w_u \left[(\mathbf{u}_1(t) - v)^2 + (\mathbf{u}_2(t) - v)^2\right]\ dt, \tag{3.7b}$$

$$V^*(\mathbf{x}, \mathbf{u}) = \min_{\mathbf{u}} \sum_{t \in \{0, 0.1, \ldots 1\}}^{T} w_x \left[(\mathbf{x}_5(t) - \mathbf{x}_5^*)^2 + (\mathbf{x}_{13}(t) - \mathbf{x}_{13}^*)^2\right] + w_u \left[(\mathbf{u}_1(t) - v)^2 + (\mathbf{u}_2(t) - v)^2\right] \tag{3.7c}$$

Equation 3.7b is the continuous-time formulation of V^* while Equation 3.7c is its approximation using a summation of instantaneous costs over discrete time steps. As the control system is simulated and controlled in discrete time steps of 0.1 s, Equation 3.7c is the relevant formulation implemented in the value NN. The corresponding symbols for Equations 3.7b and 3.7c are given with Equation 3.6a. Thus, V^* represents the minimum attainable cost in controlling the system when given a system state $\mathbf{x}$ at time $t < T$, with the controllers performing optimally for all $t < T$. The value/cost-to-go function used in this neural network share similarities with the Q-function described in Section 2.2.1, particularly in that they both characterise the "quality" of being in a given system state. It is apparent that $V^*(\mathbf{x}, \mathbf{u})$ satisfies Bellman's principle of optimality [76, 77]:

$$\begin{aligned} V^*(\mathbf{x}, \mathbf{u}) &= \min_{\mathbf{u}}\ \left[C(\mathbf{x}, t) + V^*(\mathbf{x}, (t + t_s))\right], && \text{for } t < T \\ &= C(\mathbf{x}, T), && \text{for } t = T, \end{aligned} \tag{3.8}$$

where $C(\mathbf{x}, t)$ is the cost incurred in the current time step and $t_s = 0.1$ is the step size. Thus, assuming that an accurate function for V^* can be approximated, the optimal control actions can be found by querying this function as in Equation 3.7a. This will yield the best possible system state in the next time step, where the query is repeated. Initially, the neural network is trained on trajectories arising from random control actions, such that the cost-to-go values V are not expected to be accurate: V is not minimised w.r.t. $\mathbf{u}$ as $\mathbf{u}$ is entirely random. V is calculated for the training data by performing a recursive summation of the instantaneous cost at each time step, i.e. similar to Equation 3.8 but with no attempt to minimise V w.r.t. $\mathbf{u}$. This is improved by performing the following series of training actions, such that the prediction from the value NN, $\hat{V}$, approaches V^*.

1. Train the neural network to predict $\hat{V}$ using the training data, which initially comprises only random trajectories.

2. Starting from $t = 0$, derive the optimal actions $\hat{\mathbf{u}^*}$ given the system state, by using basin-hopping (Section 3.2.3.1) to search the cost-to-go output node for

the values of $\mathbf{u}$ that minimises the predicted $\hat{V}$. Repeat until the control horizon is complete, i.e. $t = T$, thus obtaining a new trajectory.

3. $\{V_{new}\}$, the set of cost-to-go values for each time step, is calculated for the new trajectory using the same recursive summation as before. However, the difference is that $\{V_{new}\}$ is the result of the $\mathbf{u}^*$ values found that minimised $\hat{V}$ at each time step. This is the best control strategy learned from the training data. In this way, the $\min_{\mathbf{u}}$ component of Equation 3.8 is implicitly learned by the neural network. Thus, it is expected that values of $\{V_{new}\}$ will be more accurate approximations for V^*.

4. Append the state trajectory obtained in 3. to the training data set, pushing out the first trajectory in the training data set if the *experience replay memory buffer* is full.

5. Retrain the neural network on this updated training data set, thus performing *experience replay*. As the training data now includes $\{V_{new}\}$, which are expected to be better approximations for the optimal value function, it is expected also that neural network will be more accurate in predicting $\hat{V}$. Repeat training step 2.

6. The basin-hopping minimisation can now work with more accurate values of $\hat{V}$, and thus is expected to find control actions $\hat{\mathbf{u}}^*$ that increasingly approximate the optimal actions $\mathbf{u}^*$, thereby functioning as a optimal controller for system.

The experience replay memory buffer is implemented as a circular queue, where newer trajectories with more accurate cost-to-go values will push out older data from the start of the queue. Thus, the network is trained on increasingly better data that should allow it to approximate the true optimal value function.

3.2.2 Implementation of the Constraint Deviation Node

In order to satisfy state constraints while controlling the system, the NN controller should also predict if a control action will cause the system to violate its constraints at any point in the control horizon. The challenge thus is to establish the *causality* between current control actions $\mathbf{u}_t$ and a constraint violation in the future, e.g. 3 time steps later at $t + 0.3s$, such that control actions that are expected to cause a constraint violation will be avoided.

It can be deduced that, if the system is already at a "point of no return"—where constraints are currently satisfied but there is no admissible control action $\mathbf{u}_t$ that can be applied now such that constraints will still be satisfied in the next time step—then $\mathbf{u}_t$ is *not* the cause of the constraint violation. For example, it may be the case that $\mathbf{x}_5$ is just slightly below its constraint path at this time step, but the rod's thermal inertia is such that the controllers cannot cool $\mathbf{x}_5$ down to within its constraint in the next time step, even if they were pushed to their lower limits.

Thus, the first set of control actions that caused the system to reach the "point of no return" can be established to have caused future constraint violations. However, without backtracking and without explicitly solving the system using the MPC controller, it is impossible for the NN controller to identify an infeasible control action in advance. Naturally, it is not possible for the NN controller to use a backtracking algorithm to return to the last feasible point while controlling the system. As a result, the NN controller can only be certain that a set of control actions and its associated state trajectory is feasible if the entire trajectory stays within constraints after the actions are applied.

Thus, for the control of the case study system, if $\mathbf{x}_5$'s trajectory fails to stay within its constraint, then the constraint deviation value for the entire trajectory is the maximum deviation value in it. This implies that earlier control actions will be penalised if later actions cause a constraint violation (but this is necessary precisely because the causality cannot be established), which may result in the NN controller being too conservative in avoiding constraints. An evaluation of the experimental results found regarding this is given in Section 3.4.3.

3.2.3 Design Considerations

3.2.3.1 The search for globally optimal control actions

Several search algorithms were explored for the discovery of the control actions $\mathbf{u}$ that would minimise the predicted cost-to-go $\hat{V}(\mathbf{x}, \mathbf{u})$:

- local minimisation using the *L-BFGS-B* algorithm
- gradientless local minimisation using the *Nelder-Mead Simplex* method
- gradientless local minimisation using *Powell's* method
- global minimisation using basin-hopping with a local minimisation method
- global minimisation using evolutionary algorithms from the *DEAP* library
- brute-force grid search

The *L-BFGS-B* algorithm [78, 79], *Nelder-Mead Simplex* method [80], *Powell's* method [81], and basin-hopping [82, 83] were implemented using the the `minimize` library from SciPy. Design considerations include:

- The search space should be constrained as there are bounds on the control inputs
- Whether $\hat{V}(\mathbf{x}, \mathbf{u})$ is always differentiable
- Whether the search space contain local minima giving significantly different V

As the need for minimisation with input constraints was immediately apparent, the search for optimal controls was first attempted using the *L-BFGS-B* algorithm, which require the first and second derivatives to be numerically approximated. When tested, the search frequently returned control actions that were far from optimal when compared to the control choices of the MPC controller. A study into the cause of this revealed that the $\hat{V}(\mathbf{x}, \mathbf{u})$ function often exhibits non-differentiable behaviour—a simplified ex-

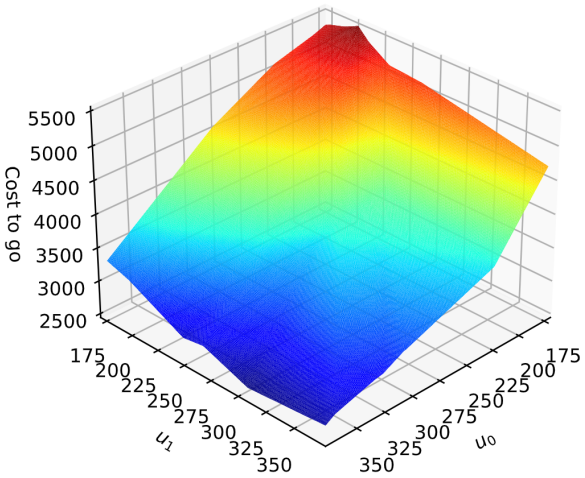
ample can be seen in Figure 3.3b. This may be a result of the *Leaky ReLU* activation functions used in the neural net, which is itself non-differentiable at the origin. It was also discovered that $\hat{V}(\mathbf{x}, \mathbf{u})$ can also be too flat for the forward-difference Jacobian approximation used in *L-BFGS-B* to work within its tolerances: a very small forward difference step size can be set to overcome this, but this was found to be too costly to the convergence speed. As such, it was deduced that a minimisation algorithm that do not require computing derivatives would be more suitable, thus restricting the choice to between the *Nelder-Mead Simplex* algorithm and *Powell's* method.

It was discovered at the same time that $\hat{V}(\mathbf{x}, \mathbf{u})$ contains many local minima—Figure 3.3c illustrates a simplified case. Local minimisation may often be sufficient if the local minima do not differ too much in the function value, but this was not the case here: being stuck in the wrong local minima will result in choosing control actions that stray far from the optimal. Hence, a global optimisation method may be required. As evolutionary algorithms such as Differential Evolution (DE) [84] are capable of global optimisation without calculating derivatives, it was conceived that they would be a suitable tool for this problem. Thus, DE was implemented using templates from the DEAP library [85]. It was tested and found to be able to discover near-optimal control actions, but was too slow for the purpose of control.

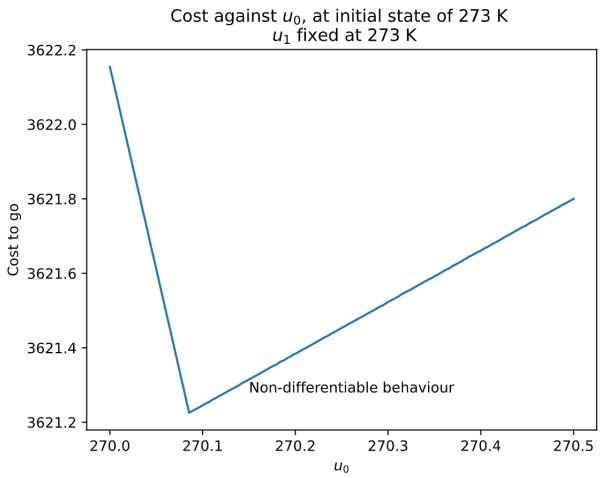
Finally, a basin-hopping search was implemented. Basin-hopping [82, 83] is a two-part method that perform several local minimisation searches from different coordinates. At each iteration, it calls a local minimisation algorithm, then take a random step in the search space such that the local minimisation starts from a new set of coordinates. *Powell's* method was found to be more efficient than *Nelder-Mead* for local minimisation, so it was combined with the basin-hopper. This was found to perform efficiently even with limited iterations in both basin-hopping and in *Powell's* method: limiting the number of hops to 5 and the number of *Powell* iterations to 10 (Table 3.1) was found to be able to discover the optimal control actions quickly.

3.2.3.2 Hyperparameters

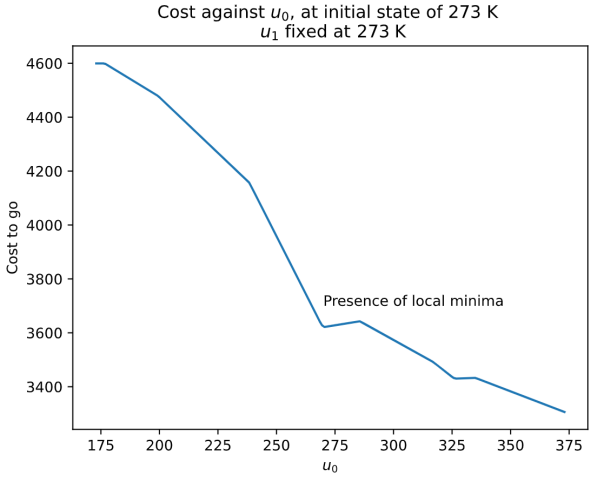
The hyperparameter configurations for the experience replay training process are given in Table 3.1. The configurations were chosen mostly via experimentation. The hyperparameters pertaining to the experience replay training process (hyperparameters 1 to 9 in Table 3.1) were chosen based on the control performance of the NN controller, i.e. the objective value it achieved in controlling the system. For the configuration of the value NN itself (hyperparameters 10 to 19 in Table 3.1), hyperparameters were chosen based on its performance on a held-out validation data set. Namely, the performance is characterised by the mean squared errors obtained from the predicted cost-to-go and constraint deviation values against validation data entries. The validation data set is a separately generated set of random trajectories, in the same format as the training data set (methodology discussed in Section 3.1.2).



(a) Control action search space at uniform initial state of 273 K



(b) Non-differentiable behaviour of NN output



(c) Presence of local minima requires global search

Figure 3.3: Considerations of the NN output properties in the search for optimal actions

Hidden layers and activation functions It was found that combining Leaky ReLU activation functions with 3 hidden layers on each of the sub-networks (purple and yellow layers in Figure 3.2) gave the best performance on the validation data set. Initially, `tanh` was tested with 1 hidden layer for each of the sub-networks, then more `tanh` hidden layers were added. This yielded no improvement over using just 1 hidden layer, which is partly expected: it is well-documented that backpropagation in deeper networks are more prone to vanishing or exploding gradient problems when paired with a saturated activation function [86]. Thus, -eLU based activation functions were explored and Leaky ReLU was found to work slightly better than eLU, both of which worked better than ReLU, perhaps as both improves over ReLU in addressing the dead neurons problem [87].

	Hyperparameter	Value	Range/values explored
1	# of trajectories used to train neural net	240	60, 120, 240, 360
2	# of constraint pass : fail in initial trajectories	3 : 1	(3 : 1), (2 : 1), keep only passed
3	# of trajectories kept in experience replay buffer	360	90, 240, 360, 480 - dependent on (1)
4	# of rounds/generations of experience replay	90	60, 90, 150
5	# of runs in each round	6	2, 6
6	"eagerness to explore"	[-2, 2]	[0, 0], [-2, 2], [-5, 5]
7	# of max basin-hopping iterations	5	[2, 10]
8	# of max Powell iterations	10	[2, stop after tolerance threshold]
9	Penalty constant applied if constraints fail	10x	2x, 5x, 10x, 100x, 1000x
10	# of hidden layers in model reduction net	3	[1, 4]
11	# of nodes in each hidden layer of (9)	(11, 11, 11)	(5, 5, 5), (11, 11, 11), (12, 12, 12)
12	Activation functions in hidden layers of (9)	Leaky Relu	Tanh, Relu, Elu, Leaky Relu
13	# of hidden layers in cost-to-go net	3	[1, 4]
14	# of nodes in each hidden layer of (12)	(7, 7, 7)	(5, 5, 5) to (12, 12, 12)
15	Activation functions in hidden layers of (12)	Leaky Relu	Tanh, Relu, Elu, Leaky Relu
16	# of training epochs	500	300, 500, 1000
17	Batch size in minibatch gradient descent	120	60, 120, 240 - Dependent on (2)
18	Learning rate for the cost-to-go node	0.05	0.01, 0.05, 0.1
19	Learning rate for the constraint deviation node	0.05	0.01, 0.05, 0.1

Table 3.1: Hyperparameters in the experience replay training process

Mini-batch gradient descent As discussed in Section 3.1.2, there is a both a temporal and a logical structure to the training data, where each trajectory comprises 10 data entries and where every 4 trajectories is a representative mini-batch. Given this structure, using mini-batch gradient descent for model weight updates was assessed to be the most suitable. Using stochastic gradient descent would break the time series of the data and was expected to result in noisy updates. To preserve the time series and pass:fail structure of the data, it was deduced that the batch size using in the mini-batch gradient descent should be a multiple of 40, so that the ratio of 3:1 pass:fail can be maintained. A batch size was 120 was found to result in the best performance on the validation data set.

Number of experience replay rounds In running the experience replay training as described in Section 3.2, it was found that control performance generally stagnated after 60 rounds. Each round contains 6 runs to allow the NN controller to explore, thereby generating 6 trajectories. Given the initial data set size of 240 trajectories and a replay buffer capacity of 360 trajectories, 60 rounds was also the point where all the initial trajectories were pushed out by new experiences, which perhaps were becoming repetitive after this point. Nevertheless the training was kept at 90 rounds as infrequent improvements were sometimes made between rounds 60 to 90. Increasing the number of rounds from 90 to 150 yielded no benefit to the resultant NN controller at all.

Powell and Basin-hopping Iterations As discussed in Section 3.2.3.1, Powell's method is used with Basin-hopping to find the optimal controls given the system state. More iterations of each would increase the confidence of the found results being close to the true global optimal, but at the expense of search time. It was found that increasing Powell iterations to beyond 10 yield only minute differences in the results found, for example u_1^* may be found to be 300 K after 10 iterations and 301.5 K after 20. For basin-hopping, it appeared that the cost-to-go function do not contain too many local minima, such that 5 basin-hops was the point beyond which no improvement in control performance was observed.

"Eagerness to explore" This is the amount of noise added to the optimal controls found by the basin-hopper to encourage it to discover new control policies. This eagerness, like in RL, should be a balance between exploiting the found optimal action and exploring possibly better control paths. It was found that adding randomly generated noise within the bounds of $[-2, 2]$ to each controller was sufficient to create observably different trajectories in each run (of which there are 6 in a round). Increasing the noise to $[-5, 5]$ was found to be detrimental to convergence.

3.3 BPOD as a Benchmark

The Balanced Proper Orthogonal Decomposition, as described in Section 2.1.3, was applied to the linear full-order model (FOM) of 20 dimensions, i.e. the system model without radiative loss, to obtain a reduced-order model (ROM). This is such that the BPOD ROM may serve as a comparison to the performance of the neural network implementation. MATLAB was used for the BPOD reduction, with the BPOD tutorial in [9] used as reference for the code implementation. A plot of the Hankel singular values (HSVs) of each BPOD mode is shown in 3.4. Theoretically, using the first 5 BPOD modes should result in a virtually perfect approximation of the system dynamics. However, it is pertinent to note that these HSVs are computed directly from the matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$, while the actual BPOD ROM is empirically constructed using impulse response measurements (Section 2.1.3).

3.3.1 BPOD Performance

The performance of the MPC controller on the BPOD system reduced to 5 states can be seen in Figures 3.5 and 3.6. Subfigure 3.5b shows the best attainable performance in controlling the system, with the optimal controls solved for using the full-order system model. The identical Subfigures 3.5a and 3.6a show the actual performance of the BPOD solved controls when applied to the full-order model. This is the result when the control actions calculated using the BPOD MPC are extracted and applied to a simulated FOM system at the corresponding time steps. Thus, it reflects how the system actually responds to the BPOD controls.

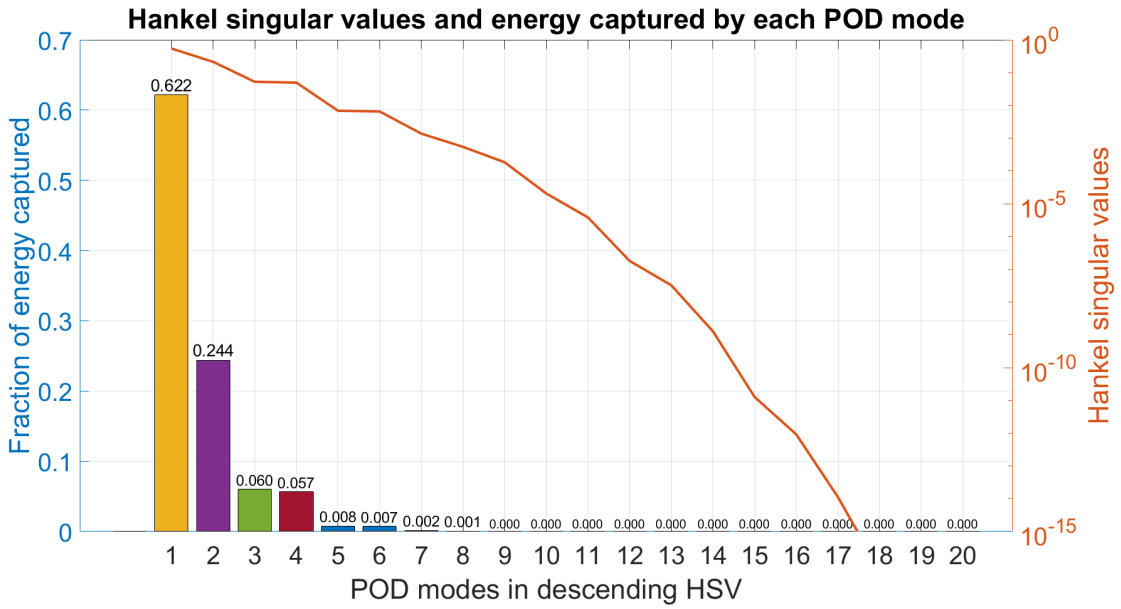


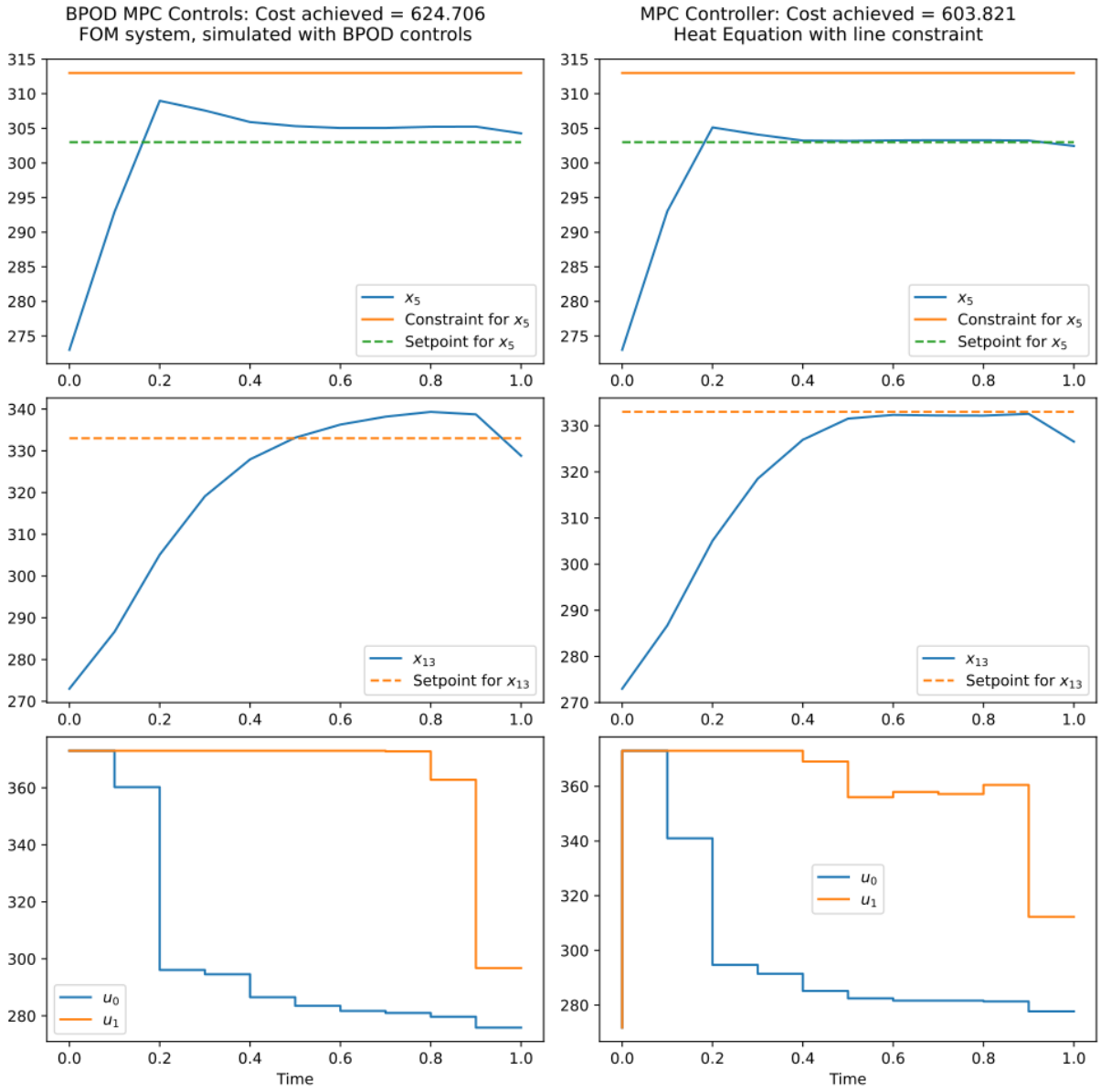
Figure 3.4: Hankel Singular Values

On the other hand, Subfigure 3.6b shows what the BPOD controller sees as it is controlling the system in its reduced state space. From the comparison in Figure 3.6, it is evident that some input and/or output dynamics are lost in the BPOD model reduction: using the reduced model, the BPOD controller computed control inputs for $\mathbf{u}_0$ and $\mathbf{u}_1$ that were excessive after 0.5 s, thereby overshooting the setpoints for both $\mathbf{x}_5$ and $\mathbf{x}_{13}$ when these controls are applied on the real system.

It appears that the fidelity of the reduced model is not preserved as perfectly as the system's theoretical Hankel Singular Values (Figure 3.4) would suggest: 5 POD modes should be enough to preserve virtually all the input/output dynamics of the system. This is attributed to the fact that the BPOD system model was built from empirical impulse response measurements (discussed in Section 2.1.3). In using these empirical snapshots for BPOD, the performance of the reduced system is expected to approach that predicted by the theoretical HSVs only if the impulse response sampling rate is high enough and the duration across which snapshots are taken is long enough. On MATLAB, the sampling rate was set at once every 0.01 s (100 Hz) and the sampling duration was 26 s.

Time Performance The time taken to solve the optimal control problem for the 5-state BPOD ROM and the 20-state FOM are recorded from IPOPT. The average of 3 runs were taken as the solve time did not vary greatly between each run. The BPOD ROM was solved in 0.140 s, while the FOM was solved in 0.211 s. The 36% decrease in OCP solve time reflects the contribution of a reduced model to control efficiency.

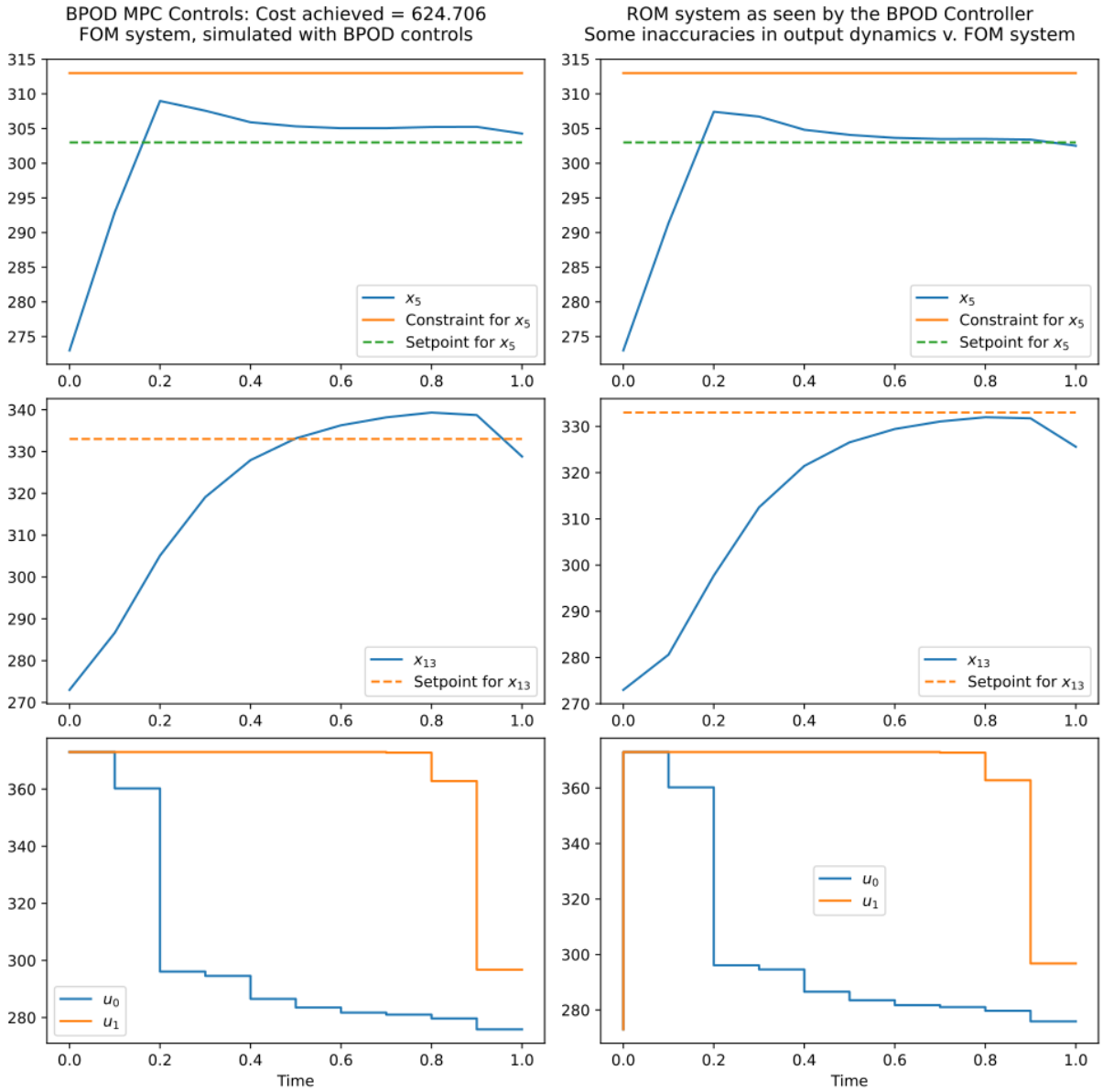
BPOD MPC performance (left) as compared to the FOM MPC (right): Cost of 625 v. 604



(a) MPC performance on BPOD ROM ($Cost = 625$) **(b)** MPC performance on FOM ($Cost = 604$)

Figure 3.5: Heat control, model reduction from 20 to 5 states using BPOD

BPOD MPC performance in the ROM system (right) as compared to the actual effect of its control actions on the FOM system (left)



(a) Actual FOM system trajectory resulting from the control inputs calculated by the BPOD MPC

(b) Trajectory of the ROM system that the BPOD MPC thinks the FOM system is following

Figure 3.6: Some input/output dynamics are lost in the ROM, causing some tracking error

3.4 Results and Evaluation

The value approximation neural network, whose implementation is described in in Section 3.2, was tested for the heat control of the 1D thin rod heat equation system. The tests were performed on variations of the system which should pose increasing degrees of challenge to the NN controller; these scenarios were briefly described in Equation 3.6b. The control objective, as detailed in Equation 3.6a, is consistent across all scenarios. The test scenarios are:

Scenario 1 Heat equation with only conduction, $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$, and a time-invariant state constraint, $\mathbf{x}_6(t) \leq 313$. The controller bounds are $[173, 373]$. The value NN (Figure 3.2) is constricted to 5 nodes for the reduced state variables (red nodes), thereby forcing it to discover an accurate projection from 20 state variables in the FOM to 5 reduced states that will be used to predict the value function and the constraint satisfaction.

It can be seen from Figure 3.4 that, theoretically for the conduction-only linear system, 5 reduced states should be sufficient to capture virtually all of the system dynamics. Here, the MPC control of the BPOD ROM (Figure 3.5a) will be compared to the performance of the NN controller.

Scenario 2 Conduction with radiative loss, $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} + \mathbf{r}(\mathbf{x})$, where $\mathbf{r}_i(\mathbf{x}_i) = -kc(\mathbf{x}_i^4 - 273^4)$. The radiative loss term causes the system to be control-affine nonlinear, and the intent is to test if the NN controller can handle the nonlinearity. The $\mathbf{x}_6(t) \leq 313$ state constraint remains but the original controller upper limit was found to be insufficient to compensate for the radiative loss, and it was raised such that the controller bounds are now $[173, 473]$. The value NN is constricted to work with 5 reduced states, as it was in Scenario 1.

Scenario 3 Conduction with radiative loss, as described in Scenario 2. However, the time-invariant constraint in Scenario 1 and Scenario 2 was replaced with a time-variant path constraint, $\mathbf{x}_6(t) \leq 5t^2 + 10t + 293$, which was expected to pose considerable challenge to the NN controller. Moreover, the reduced state space was further constricted to 3 nodes, thus Figure 3.2 represents the network structure of the value NN used in Scenario 3.

3.4.1 Test Scenario 1

The result obtained in Scenario 1 can be seen in Figure 3.7. The NN controller was able to learn appropriate controls relatively quickly, and after 9 rounds of experience replay training achieved an overall cost of 627, against the cost of 604 of the optimal control path. The evolution of its performance can be seen through Subfigures 3.8a, 3.8b, and 3.7a, showing the control profiles and state trajectories initially, after 6 rounds, and after 9 rounds of experience replay training respectively. Further training of up to 90 rounds of experience replay did not create a better controller.

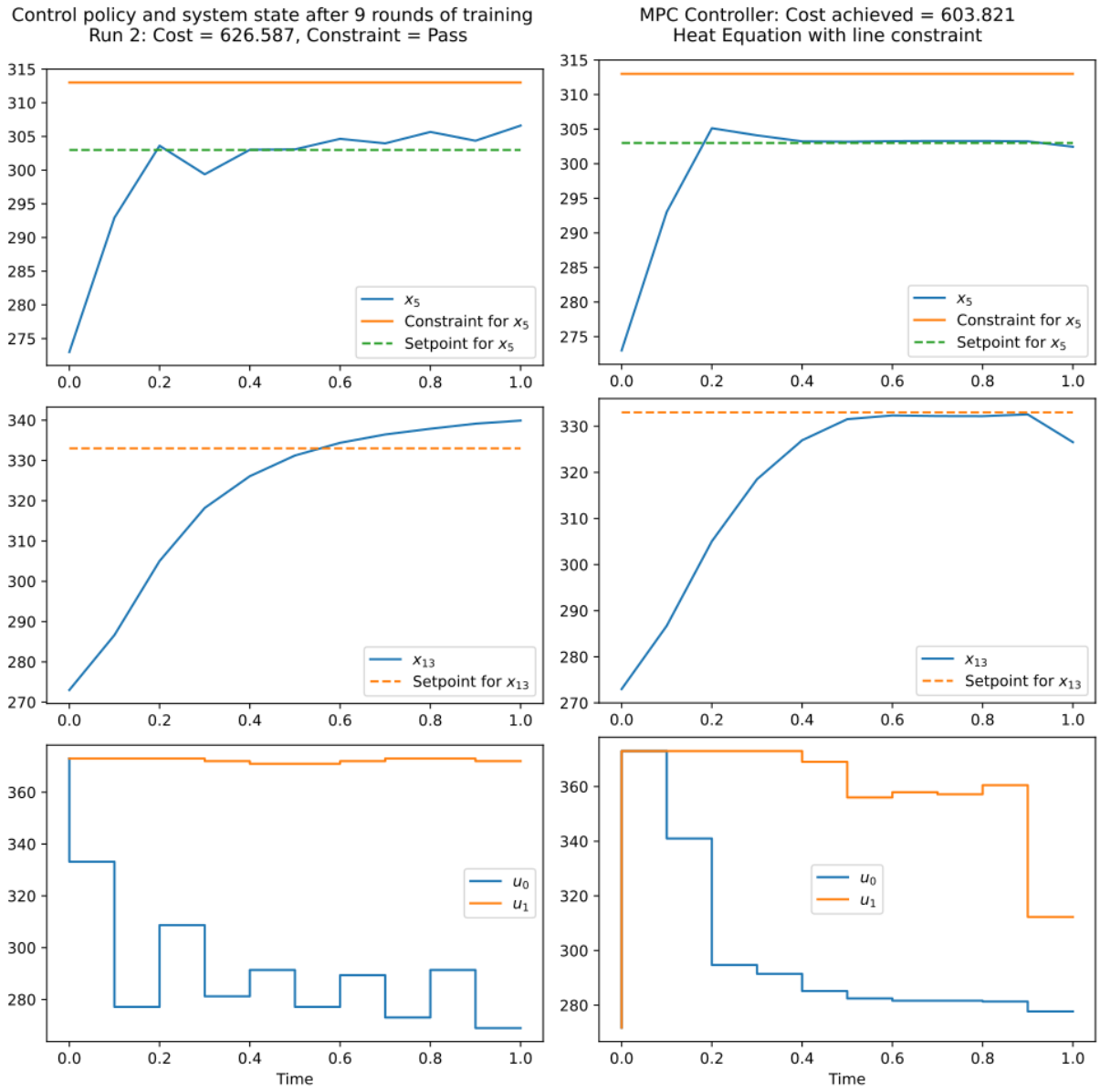
It can be observed from Subfigure 3.8a that the NN controller has already learnt a rough idea of the state setpoints and the control policy to get there without any experience replay. It is pertinent to note that this has been learnt from *a set of 240 randomly generated trajectories*: it was never shown what an optimal control profile looked like. This demonstrates the feasibility of the value approximation approach in replacing the explicit solution of the optimal control problem.

As visible from subfigures 3.8a, 3.8b, and 3.7a, the process of learning the setpoint for $\mathbf{x}_5$ involved a "pulse-width modulation" (PWM) style of regulating the heat input from the left controller, $\mathbf{u}_0$. The alternation between the significantly different high and low values of $\mathbf{u}_0$ initially caused distinctive peaks and valleys in the temperature profile of $\mathbf{x}_5$, roughly centred about its setpoint of 303 K. The controller improved at tracking this setpoint as it learnt the appropriate levels for the heating pulses, but there remains some "ringing" in the series of peaks and valleys of $\mathbf{u}_0$ that remains (Subfigure 3.7a), albeit being far less pronounced than in the earlier training rounds. For tracking the setpoint of $\mathbf{x}_{13}$, $\mathbf{u}_1$ was first under-applied such that $\mathbf{x}_{13}$ falls short of its setpoint of 333 K (Subfigure 3.8a). The NN controller learnt to compensate for this by raising $\mathbf{u}_1$, though perhaps by too much, as it is evident that $\mathbf{x}_{13}$ starts to get heated beyond 333 K after 0.5 s.

3.4.1.1 Comparison to BPOD

A comparison between the NN controller's performance against the benchmark BPOD MPC from Section 3.3 is shown in Figure 3.9. In both cases the controllers acted on a reduced system with 5 states. They achieved similar objective values: the NN controller incurred a cost of 627 while the BPOD MPC incurred a cost of 625. From Figure 3.9, it can be observed that $\mathbf{u}_1$'s control trajectory is more consistent in the BPOD MPC: the BPOD $\mathbf{u}_1$ values are generally the average of 2 consecutive pulses in the NN controller, effectively smoothing out the PWM-like behaviour. The comparable performance between the BPOD MPC and the NN controller demonstrates that the value approximation approach is feasible in replacing a BPOD ROM where efficient control of a high-dimensional system is required.

Scenario 1: Control performance of the 5-state NN controller (left) as compared to the FOM MPC (right): Cost of 627 v. 604

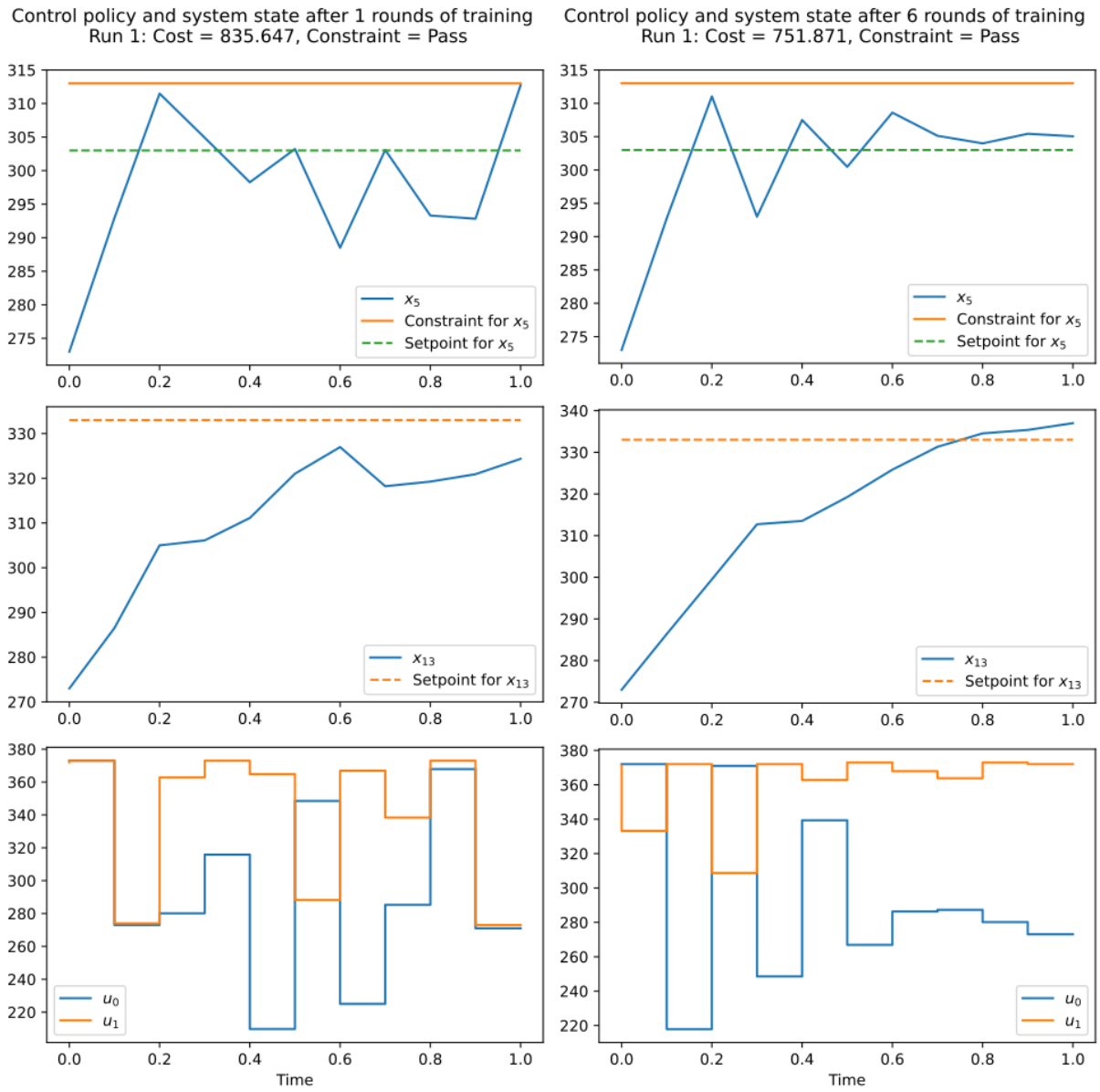


(a) Best performance of NN ($Cost = 627$)

(b) Performance of MPC ($Cost = 604$)

Figure 3.7: Scenario 1: Thermal control, reduction to 5 states in value NN

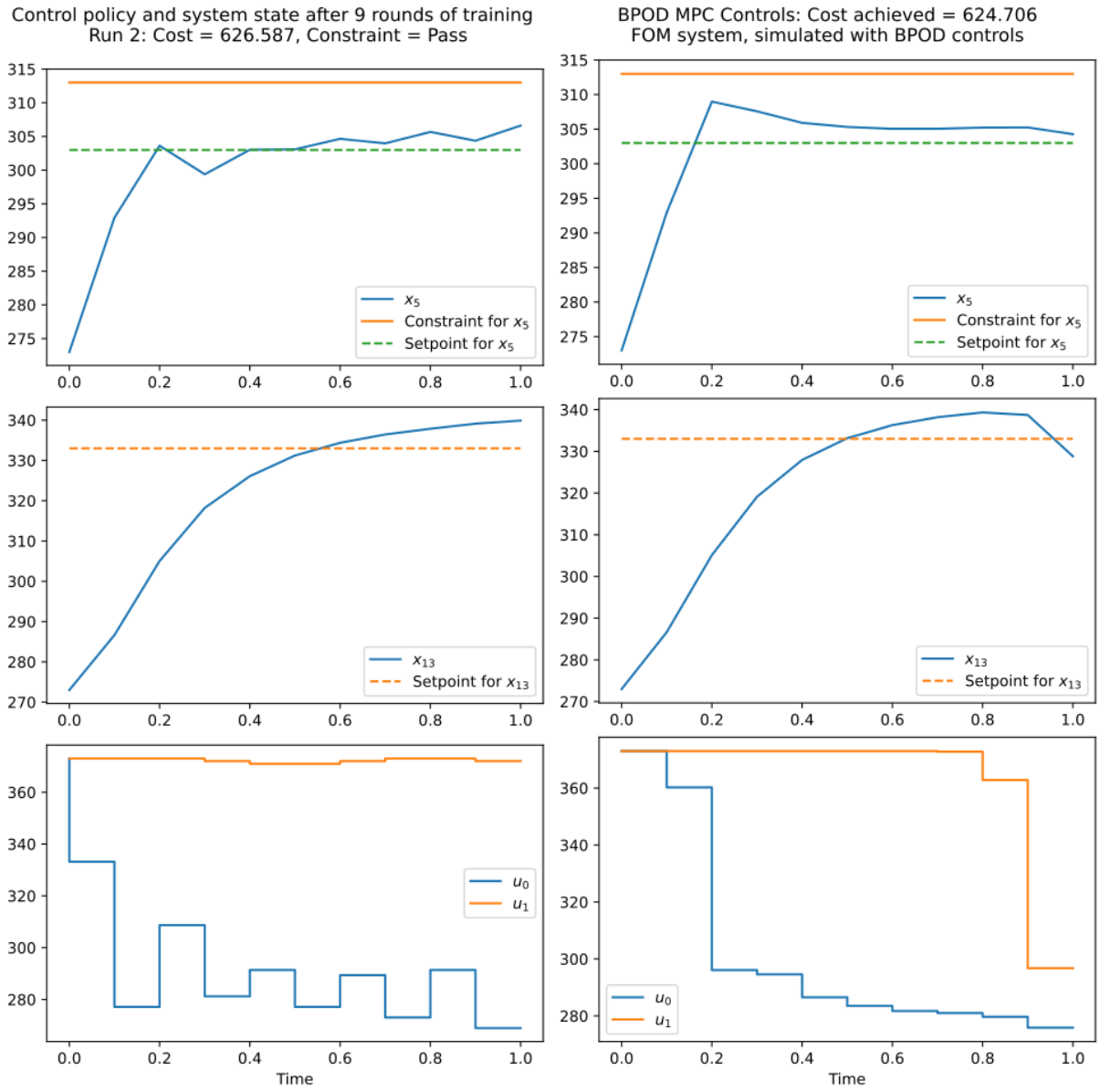
Scenario 1: Training improvements in NN controller, showing its initial performance (left), performance after 6 rounds of experience replay (right), and best performance obtained after 9 rounds (Subfigure 3.7a)



(a) Initial performance of NN ($Cost = 836$) **(b)** Performance after 6 rounds ($Cost = 752$)

Figure 3.8: Scenario 1: Training improvements in NN controller performance

Scenario 1: Control performance of the 5-state NN controller (left) as compared to the 5-state BPOD ROM MPC from Section 3.3 (right): Cost of 627 v. 625



(a) Best performance of NN ($Cost = 627$) **(b)** MPC performance on BPOD ROM ($Cost = 625$)

Figure 3.9: Scenario 1: Similar performance between NN controller and BPOD MPC

3.4.2 Test Scenario 2

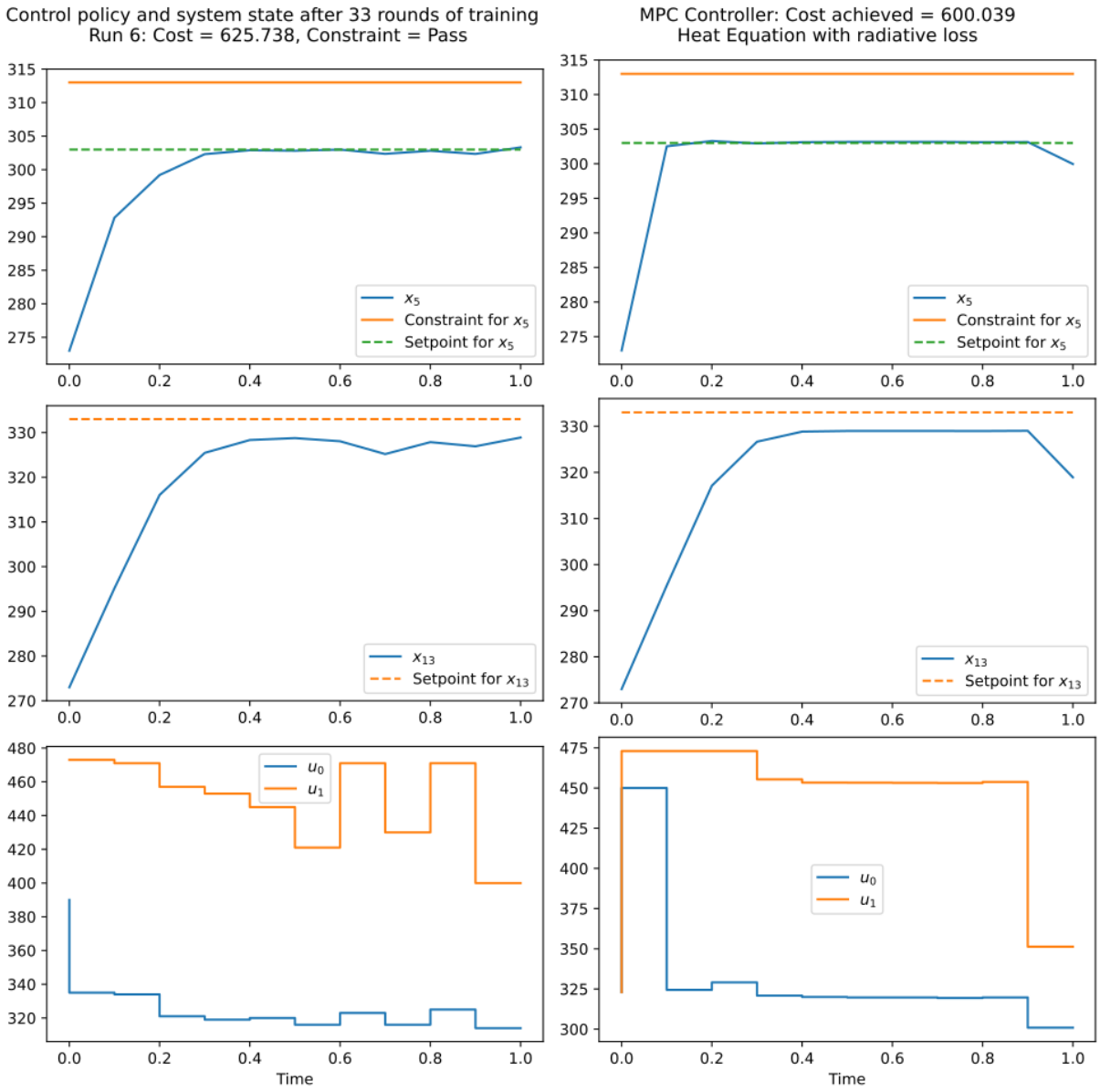
The radiative loss term added to make the system control-affine nonlinear did not result in a decrease of control cost-performance of the NN controller. However, the number of experience replay training rounds required to obtain the best performance increased from 9 rounds in Scenario 1 to 33 rounds. The performance of the resultant NN controller is shown in Subfigure 3.10a.

As seen in Figure 3.10, the state trajectory from the NN controller closely approximates that of the MPC-solved system. The NN controller achieved a cost of 626 against the cost of 600 of the optimal control path. From Subfigure 3.11a, it can be seen that the initial NN controller far exceeded the state constraint for $\mathbf{x_5}$. This demonstrates that the constraint deviation node in the value NN was initially inaccurate. However, after 10 rounds of experience replay training (Subfigure 3.11b, the NN controller learnt to avoid overshooting the setpoint for $\mathbf{x_5}$ by sharply turning down $\mathbf{u_0}$ each time $\mathbf{x_5}$ crosses its setpoint, again in a PWM-like manner. Subsequently, the value function predictions were refined, such that the actual setpoint for $\mathbf{x_5}$ was learnt and accurately tracked (Subfigure 3.10a)

In Figure 3.10b, there is an apparent steady-state tracking error even in the MPC-solved system. A closer examination reveals that this is the point at which the effect of the $w_u = 0.005$ controller weight in objective function sets in (Equation 3.6a). The MPC controller deliberately drops u_1 from 473 K to approximately 453 K after 0.2 s, while a steady-state error of approximately 5 K is constant in tracking the setpoint of x_{13}. With other factors held constant, a 5 K setpoint error contributes a cost of $w_x \times 5^2 \approx 25\ s^{-1}$ while ramping up the controller from 453 K to 473 K contributes a cost of $w_u \times [(473-273)^2 - (453-273)^2] = 38\ s^{-1}$: it has become cheaper to accept the setpoint error at 5 K than to maintain heat from $\mathbf{u}_1$.

Thus, it is pertinent to note that the NN controller has managed to learn this behaviour as well, but without being given the cost function (Figure 3.10a). This suggests that the value NN has learnt an accurate approximation for the optimal value function, and demonstrates that the value approximation approach can take into account objective functions with both state setpoint and controller terms. Nevertheless, some noise in the cost function approximation can be observed, again via the PWN-like behaviour of $\mathbf{u}_1$ from 0.6 s to 0.8 s.

Scenario 2: Control performance of the 5-state NN controller (left) as compared to the FOM MPC (right): Cost of 626 v. 600

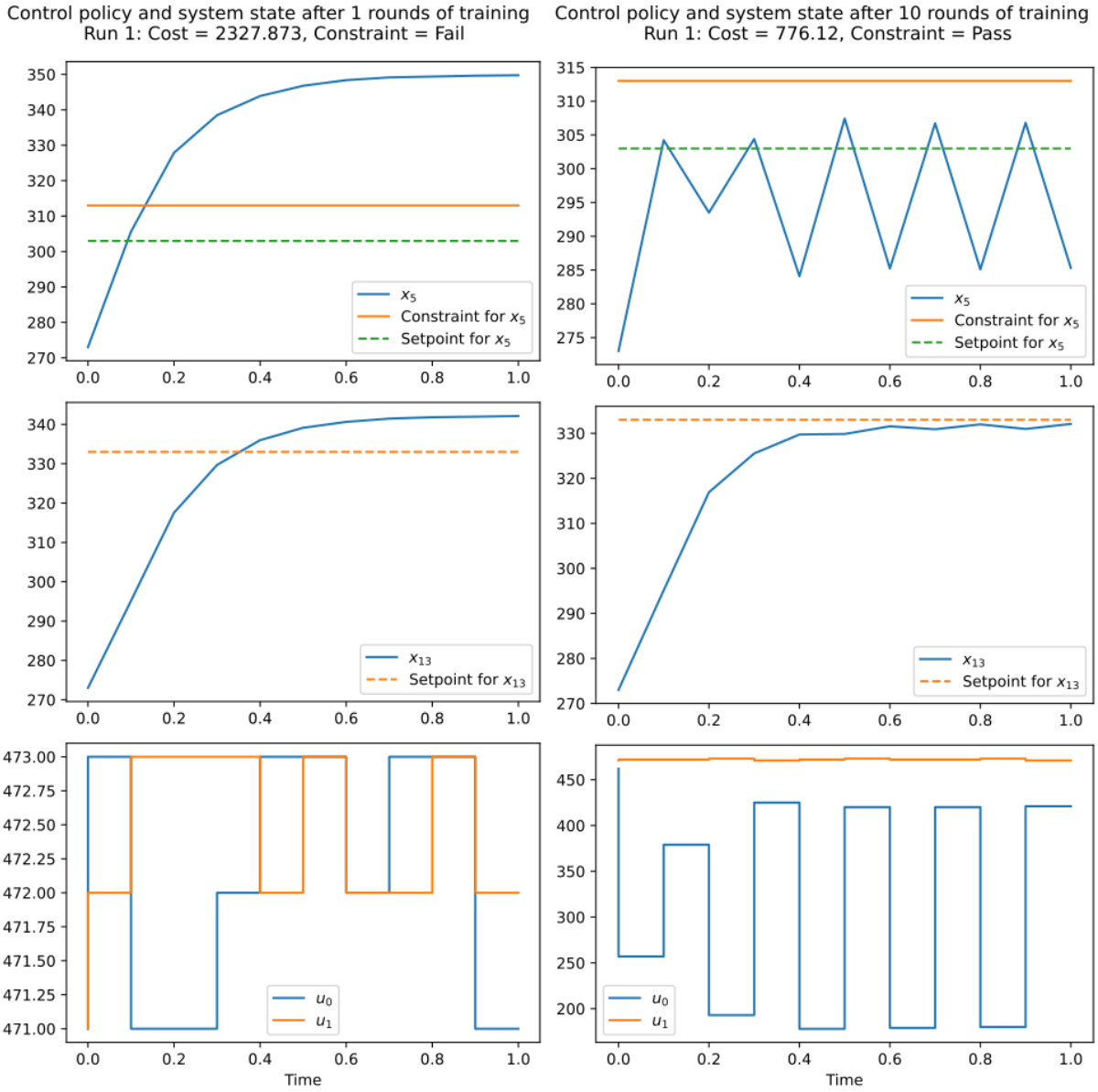


(a) Best performance of NN ($Cost = 626$)

(b) Performance of MPC ($Cost = 600$)

Figure 3.10: Scenario 2: With radiative loss, reduction to 5 states in value NN

Scenario 2: Training improvements in NN controller, showing its initial performance (left), performance after 10 rounds of experience replay (right), and best performance obtained after 33 rounds (Subfigure 3.10a)



(a) Initial performance of NN ($Cost = 2328$)

(b) Performance after 10 rounds ($Cost = 776$)

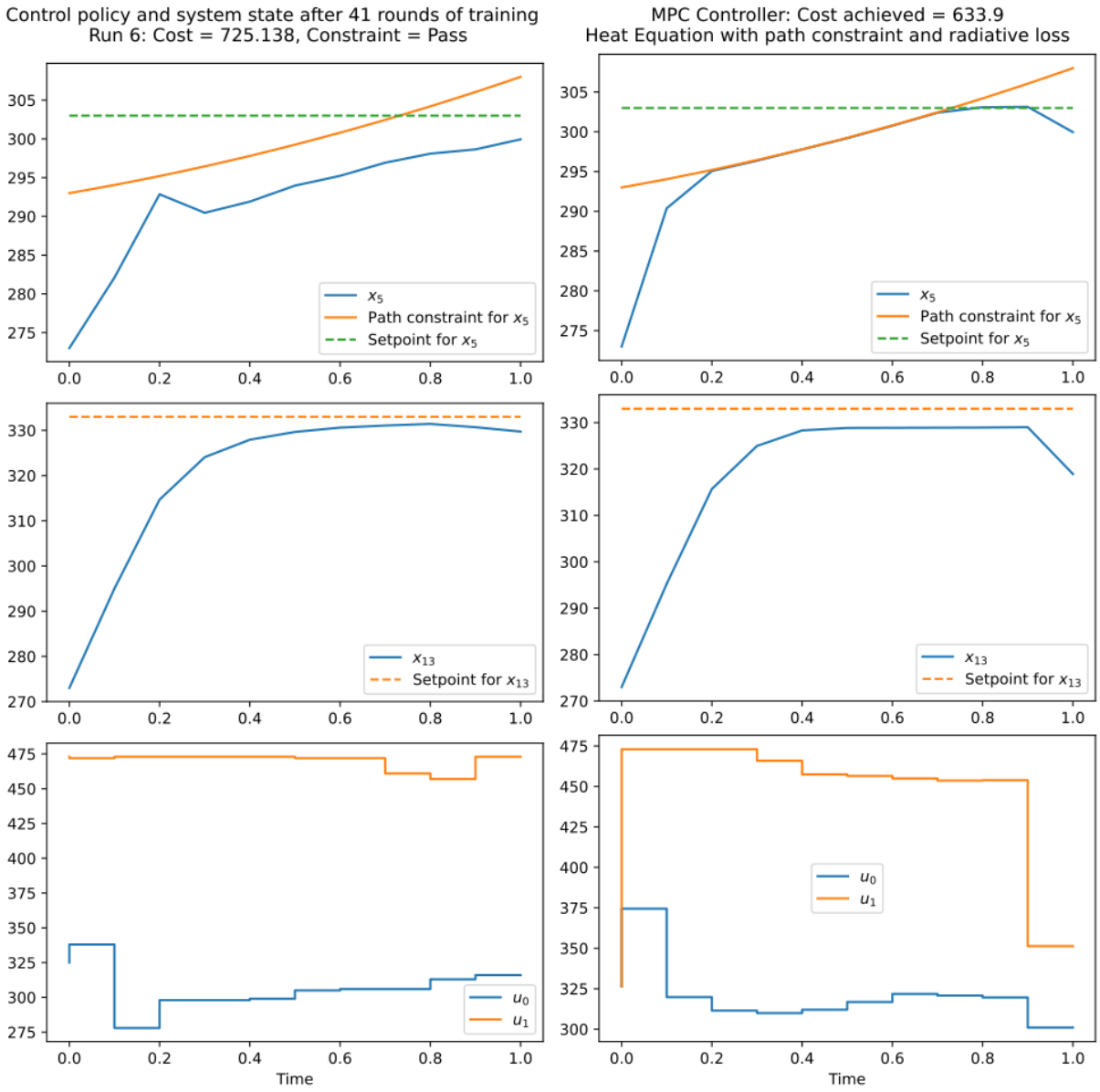
Figure 3.11: Scenario 2: Training improvements in NN controller performance

3.4.3 Test Scenario 3

The result obtained in Scenario 3 can be seen in Figure 3.12. As opposed to the time-invariant constraint for $\mathbf{x}_{13}$ in Scenario 1 and Scenario 2, Scenario 3 contained a time-variant path constraint that proved to be challenging to the NN controller. In Figure 3.12a, it can be seen that the NN controller is too averse to approaching the path constraint, abruptly reducing the heat supplied from $\mathbf{u}_0$ at 0.2 s. After 0.3 s, $\mathbf{x}_5$ was controlled such that it trended towards the setpoint in parallel to the constraint path. This demonstrates that the constraint deviation node is able to track the time-variance in the constraint, albeit not very accurately, as the gap between the paths of $\mathbf{x}_5$ and its constraint was consistently about 5 K.

A discussion on how the "maximum deviation" implementation of the constraint node may lead to over-correction in constraint avoidance was given in Section 3.2.2. An additional factor was identified that is specific to time-variant constraints: it is harder for the value NN to learn the path of the time-variant constraint, because 2 identical values of $\mathbf{x}_5$ may yield different constraint satisfaction results. Thus, the constraint prediction cannot be made using only the value of $\mathbf{x}_5$; the value NN needs to use the values of the other state variables as a proxy to infer the trajectory progress. It is therefore expected that more training rounds would be needed to accurately learn time-variant constraints. Subfigures 3.13a and 3.13b demonstrate the gradual improvement of the NN controller in constraint handling over 25 training rounds, where the extent of $\mathbf{x}_5$'s constraint deviation decreases as the constraint node improves its prediction accuracy.

Scenario 3: Control performance of the 3-state NN controller (left) as compared to the FOM MPC (right): Cost of 725 v. 634

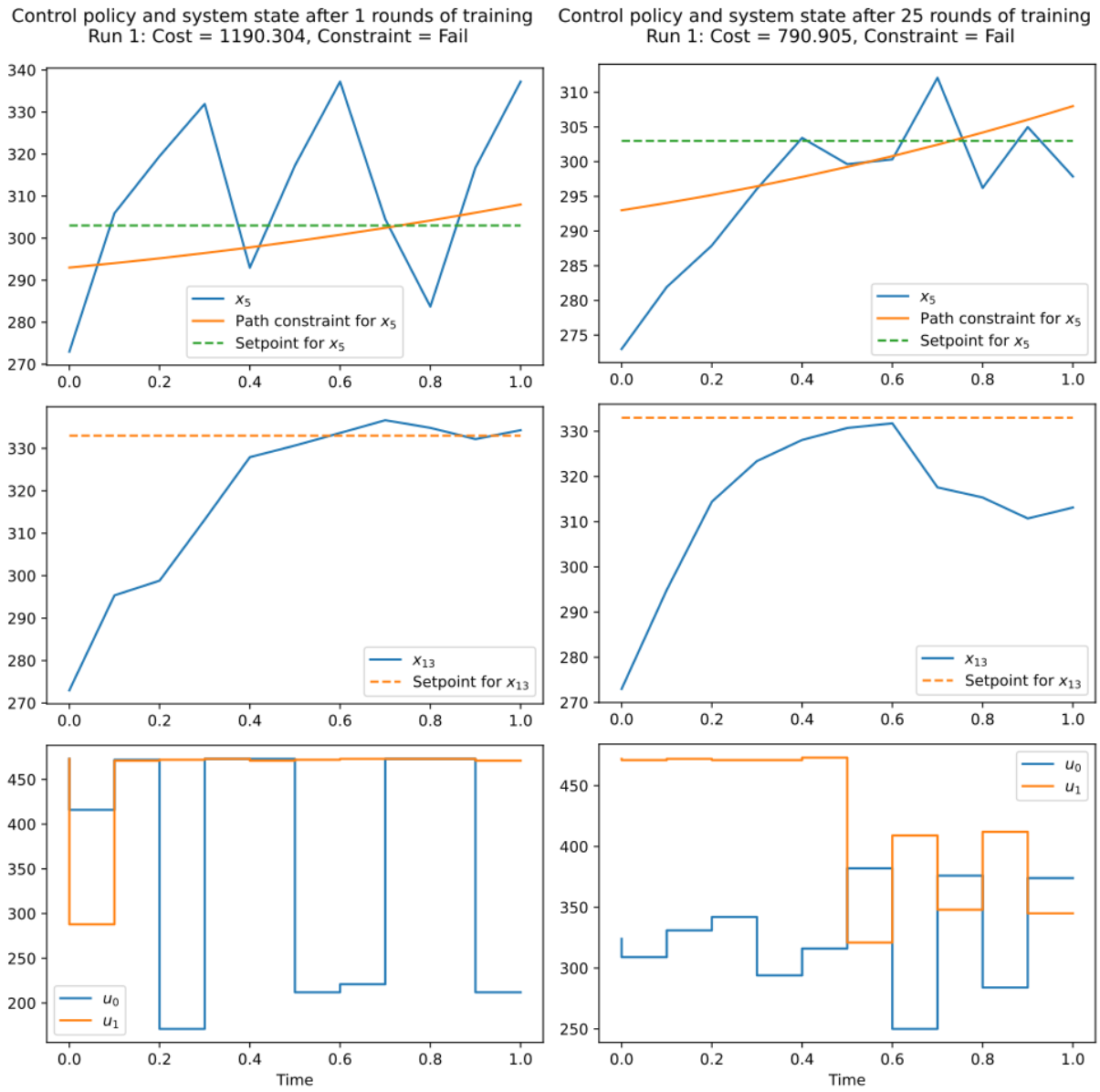


(a) Best performance of NN ($Cost = 725$)

(b) Performance of MPC ($Cost = 634$)

Figure 3.12: Scenario 3: With radiative loss and path constraint, reduction to 3 states

Scenario 3: Training improvements in NN controller, showing its initial performance (left), performance after 25 rounds of experience replay (right), and best performance obtained after 41 rounds (Subfigure 3.12a)



(a) Initial performance of NN ($Cost = 1190$)

(b) Performance after 25 rounds ($Cost = 791$)

Figure 3.13: Scenario 3: Training improvements in NN controller performance

3.4.4 Time Performance as Compared to MPC

For this case study system, the NN controller mostly did not reduce the time required to derive optimal control actions: each basin-hopping search, with parameters configured to those in Rows 7 and 8 of Table 3.1, took between approximately 0.6 to 1.5 s in all 3 scenarios. Thus, the total search time across 10 time steps was approximately between 6 to 15 s.

In comparison, IPOPT solved the OCP of Scenario 1 in 0.211 s, that of Scenario 2 in 2.096 s, and that of Scenario 3 in 4.037 s. The results for Scenarios 2 and 3 were found to be more varied, and thus the average of 5 runs were taken. The tenfold increase in solve time from Scenario 1 to 2 is attributed to the nonlinear radiative loss term added to the model, while all other factors stayed constant. The twofold increase from Scenario 2 to 3 is attributed to the complexity introduced by the time-variant path constraint, while all other factors stayed constant.

However, the NN controller *removed the impact of nonlinearities, and state and constraint complexity* on the *online* solve time. It is only dependent on the complexity of the *control search space*, namely the number of control variables and the range of admissible control input values. Thus, the value approximation approach has transferred the computational burden from online during real-time control to offline, where the experience replay training process only needs to be performed once to produce a NN controller that can be used for online optimal control. It has demonstrated to be *effective for nonlinear systems*, which is a key advantage over conventional (non-DEIM) BPOD model reduction (Section 2.1.3). Typically, there are far fewer control inputs than state variables in a high-dimensional system; the value approximation approach is thus especially suited to such cases.

Additionally, it is worth nothing that the OCP is not solved repetitively in each time step, as it would be if the receding horizon nature of MPC were implemented. This would have increased the total OCP solve time across 10 time steps by a factor a 10, which would result in a favourable comparison for the NN controller's time performance. For the reasons discussed in Section 1.2, namely the absence of disturbances and plant-model mismatch, this was not necessary. It is further worth nothing that the NN controller was trained solely with experimental data and does not assume any knowledge of a system model, thereby being inherently less susceptible to plant-model mismatches. Further reduction in the online search time for the NN controller may be possible by replacing the basin-hopping search with another neural network, which will be directly responsible for control. As such, the new control network would be trained to learn the $\min_{\mathbf{u}} V(\mathbf{x}, \mathbf{u})$ function. Hence, 2 areas for further work are identified:

1. Improving constraint prediction performance for time-variant path constraints.
2. Further reduction of the online search time, possibly by replacing the basin-hopper with a neural network.

Chapter 4

Autoencoder with System Identification

This chapter details the design, implementation, and performance of using an autoencoder neural network (as introduced in Section 2.1.4) in conjunction with SINDYc (Section 2.1.5) for the purpose of model reduction with control. The case study system for this implementation is the nonlinear heat equation with radiative loss, used in Scenario 2 and Scenario 3 in Chapter 3.

4.1 Design and Implementation

4.1.1 Identification of a Reduced-Order Empirical Model

An autoencoder is first trained using the trajectory data from the MPC-controlled full-order model. 500 trajectories were generated with the FOM state variables $\mathbf{x}$ initialised to random values between $[243, 273]$, with 400 allocated to the train data set and 100 to the test data set. This procedure of obtaining training data is in contrast to the method used in Chapter 3 (Section 3.1.2), where the initial states of $\mathbf{x}$ were constant at 273 K while random control inputs were applied. Trajectories generated using random controls were found to be too noisy for SINDYc to compute a good fit; ROM system models fitted by SINDYc using them were inaccurate and unusable for control.

Next, the training data set is encoded using the autoencoder to obtain the ROM training data set for SINDYc. This ROM data set contains the state of the reduced variables $\tilde{\mathbf{x}}$ and original control inputs $\mathbf{u}$ for each time step in the 400 trajectories. SINDYc is then used to fit a system model between the reduced $\tilde{\mathbf{x}}$ and original controls $\mathbf{u}$. Finally, the empirical ROM found by SINDYc is formulated in Pyomo, where its performance is tested.

4.1.2 Autoencoder Neural Network Structure

The autoencoder is implemented as 2 components: an encoder network which accepts the full state variables $\mathbf{x}$ as inputs and outputs the reduced variables $\tilde{\mathbf{x}}$, and a decoder network that accept $\tilde{\mathbf{x}}$ and outputs the reconstructed full state variables $\hat{\mathbf{x}}$. Thus, it is possible to call the encoder and decoder individually to obtain the desired transformation. Figure 4.1 illustrates the neural network structure of an autoencoder constricted to 3 ROM states. The FOM has 20 states (Section 3.1.1), thus the autoencoder has 20 nodes on both its input and output layers.

The level of reduction can be adjusted by changing the number of nodes in the reduced layer; the autoencoder was able to fit a good function for projecting 20 FOM states to *only 1 ROM state*. On a held-out test data set generated separately from the training trajectories, the autoencoder—constricted to 1 ROM state—achieved a mean squared error of 0.00212 and a mean absolute error of 0.0262 on the normalised state variables. Relaxing the constriction to 3 ROM states, it achieved a mean squared error of 0.00131 and a mean absolute error of 0.0188 (Table 4.2). Examining the network predictions from the 3-state autoencoder, they were found to be accurate to within approximately ± 6 K across the operating range of the system. These findings demonstrate the feasibility of using autoencoders as tools for accurate model reduction.

The hyperparameters for the autoencoder were tuned by experimentation. The best configuration found and the values explored are shown in Table 4.1.

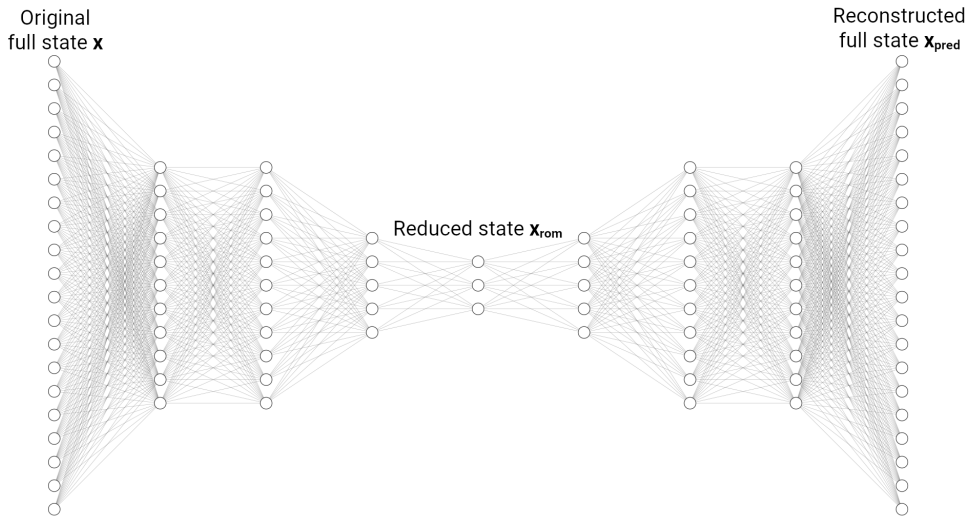


Figure 4.1: Network structure of the autoencoder used to reduce the FOM to 3 states

4.1.3 SINDYc Configuration

As described in Section 2.1.5, the library functions used to derive the best-fit empirical model can include polynomial and Fourier terms. Generally, using more terms from the library improves SINDYc's fit accuracy, though this at the expense of the complexity

	Hyperparameter	Value	Range/values explored
1	# of trajectories used to train autoencoder	400	240, 400, 600
2	# of hidden layers in encoder net	3	[1, 4]
3	# of nodes in each hidden layer of (2)	(11, 11, 5)	(5, 5, 5), (11, 11, 5), (11, 11, 11)
4	Activation functions in hidden layers of (2)	Elu	Sigmoid, Relu, Elu, Leaky Relu
5	# of hidden layers in decoder net	3	[1, 4]
6	# of nodes in each hidden layer of (5)	(5, 11, 11)	(5, 5, 5), (5, 11, 11), (11, 11, 11)
7	Activation functions in hidden layers of (5)	Elu	Sigmoid, Relu, Elu, Leaky Relu
8	# of training epochs	2000	500, 1000, 2000
9	Batch size in minibatch gradient descent	100	30, 60, 100, 120, 200
10	Learning rate	0.01	0.005, 0.01, 0.05, 0.1

Table 4.1: Hyperparameters for training the autoencoder neural network

of the ROM. An accurate but complex ROM is usually highly nonlinear, thus possibly becoming harder to control than the original FOM, thereby defeating the purpose of model reduction. The practical limit for the case study system—beyond which the ROM MPC takes longer to compute in PYOMO than the FOM—was found to be a 3rd-order polynomial fit without interaction (e.g. $\tilde{\mathbf{x}}_0^2\tilde{\mathbf{x}}_1^3$) between the terms.

Various fit configurations in SINDYC were tested for ROMs with 1, 2, and 3 states. The results are listed in Table 4.2. An illustration of an accurate but highly complex fit, obtained using the configuration in Row 4, is given in Appendix B. The R2, mean squared error, and mean absolute error are computed on $\tilde{x}$ normalised to within $[0, 1]$. They represent the error between the actual finite difference approximated time derivative of $\tilde{x}$, i.e. $\dot{\tilde{x}}$, and that predicted by SINDYC's empirical model, $\hat{\dot{\tilde{x}}}$. These error metrics were calculated on a held-out test data set of 100 trajectories. It is observed that linear fits are able to capture most of the dynamics in the ROM, while complex fits such as that in Row 4 of Table 4.2 could produce a nearly perfect empirical model. The configuration in Row 9 is tested in Section 4.2.1, and the configurations in Rows 1 and 2 are tested in Section 4.2.2.

4.1.4 Mapping FOM Control Objective and State Constraints to ROM

To control the ROM system, it is necessary to map the FOM system control objective and state constraints in Equation 3.6a to the reduced state space. As discussed in Section 2.1.4, this is challenging because each ROM state variable is a function of possibly all the FOM variables after passing through the encoder. It was thus devised that the ROM MPC should track setpoints that correspond to the steady-state of the FOM MPC system. The controller cost term is removed on the FOM to avoid the steady-state error discussed in 3.4.2. This steady-state is a snapshot of the FOM system when $\mathbf{x}_5$ and $\mathbf{x}_{13}$ are approximately at their setpoints of 303 K and 333 K respectively. Thus, steady-state values of the 20 FOM state variables ($\mathbf{x}_\mathbf{0}^*$ to $\mathbf{x}_\mathbf{19}^*$ are passed through the encoder network,

	ROM Order	ROM MSE	ROM MAE	ROM MAPE	Polynomial Degree	Fourier Degree	Interaction Terms	SINDYc R2	SINDYc MSE	SINDYc MAE
1					1	NA	No	0.981	0.024	0.129
2	1	0.00212	0.0262	0.2570	2	NA	No	0.996	0.004	0.044
3					5	NA	No	0.999	0.000	0.012
4					5	3	Yes	0.999	0.000	0.007
5					1	NA	No	0.991	0.009	0.047
6	2	0.00129	0.0193	0.0649	2	NA	No	0.995	0.004	0.035
7					5	NA	No	0.996	0.003	0.022
8					5	3	Yes	0.996	0.004	0.020
9					1	NA	No	0.961	0.046	0.131
10					1	1	Yes	0.992	0.008	0.040
11					2	2	No	0.993	0.007	0.029
12	3	0.00131	0.0188	0.0792	3	NA	Yes	0.993	0.007	0.028
13					5	NA	Yes	0.992	0.009	0.031
14					5	NA	No	0.993	0.007	0.031
15					5	3	Yes	0.993	0.007	0.026

Table 4.2: Accuracy of the SINDYc-fitted ROM under various configurations
MSE: Mean squared error, MAE: Mean absolute error,
MAPE: Mean absolute percentage error, R2: Coefficient of determination
All error metrics are obtained on variables normalised to $[0, 1]$

obtaining their corresponding reduced states, e.g. $\tilde{\mathbf{x}}_0^*$, $\tilde{\mathbf{x}}_1^*$ and $\tilde{\mathbf{x}}_2^*$ for a 3rd-order ROM; these are the state setpoints for the ROM. Hence, the control objective for the ROM is obtained as:

$$\begin{aligned} &\underset{\mathbf{u}}{\text{minimise}} \\ &J(\tilde{\mathbf{x}}, \mathbf{u}) = \int_0^T w_x \left[(\tilde{\mathbf{x}}_0(t) - \tilde{\mathbf{x}}_0^*)^2 + (\tilde{\mathbf{x}}_1(t) - \tilde{\mathbf{x}}_1^*)^2 + (\tilde{\mathbf{x}}_2(t) - \tilde{\mathbf{x}}_2^*)^2\right] + \\ &\qquad\qquad w_u \left[(\mathbf{u}_0(t) - v)^2 + (\mathbf{u}_1(t) - v)^2\right] \, dt \end{aligned} \tag{4.1a}$$

$$\begin{aligned} &\text{subject to} \quad \dot{\tilde{\mathbf{x}}} = \mathbf{f}(\tilde{\mathbf{x}}, \mathbf{u}), \quad \text{where } \mathbf{f}(\tilde{\mathbf{x}}, \mathbf{u}) \text{ is identified by SINDYc} \\ &\qquad\qquad 173 \leq \mathbf{u} \leq 473, \end{aligned} \tag{4.1b}$$

where $w_x = 0.995$ is the setpoint weight and $w_c = 0.005$ is the weight for the controller costs. $v = 273$ K is the environmental temperature. Equation 4.1a is applicable to a ROM with 3 states. For a ROM with 1 state (used in Section 4.2.2), the setpoint term will be simply $w_x \left[(\tilde{\mathbf{x}}_0(t) - \tilde{\mathbf{x}}_0^*)^2\right]$.

While this approach of using an "ideal snapshot" is feasible for discovering the corresponding objectives in the ROM, state constraints cannot be mapped this way: it is not possible to decouple constrained state variables from unconstrained ones when they are projected onto the reduced space. To address this, 2 methods were devised and they are discussed in the following sections.

4.1.4.1 Constraint/Objective Mapping by Decoupling the Autoencoder

To decouple state variables that are part of the control objective or are constrained, a ROM in the following form is desired:

$$\dot{\tilde{\mathbf{x}}} = \mathbf{f}(\tilde{\mathbf{x}}, \mathbf{x}_5, \mathbf{x}_{13}, \mathbf{u}) \tag{4.2a}$$

$$\dot{\mathbf{x}_5} = g(\tilde{\mathbf{x}}, \mathbf{x}_5, \mathbf{x}_{13}, \mathbf{u}) \tag{4.2b}$$

$$\dot{\mathbf{x}_{13}} = h(\tilde{\mathbf{x}}, \mathbf{x}_5, \mathbf{x}_{13}, \mathbf{u}), \tag{4.2c}$$

where $\tilde{\mathbf{x}}$ is the vector of reduced variables obtained by encoding the FOM variables while *excluding* $\mathbf{x}_5$ and $\mathbf{x}_{13}$. $\mathbf{f}$, g and h are differential equations that govern the empirical model, as fitted by SINDYc. The other symbols are consistent with their previous definitions in Equations 3.6a and 4.1a. Thus, the optimal control problem can be formulated independently from the reduced variables, preserving the original control objective and state constraints:

$$\begin{aligned} &\underset{\mathbf{u}}{\text{minimise}} \\ &J(\mathbf{x}_5, \mathbf{x}_{13}, \mathbf{u}) = \int_0^T w_x \left[(\mathbf{x}_5(t) - \mathbf{x}_5^*)^2 + (\mathbf{x}_{13}(t) - \mathbf{x}_{13}^*)^2\right] + \\ &\qquad\qquad w_u \left[(\mathbf{u}_0(t) - v)^2 + (\mathbf{u}_1(t) - v)^2\right] \, dt \end{aligned} \tag{4.3a}$$

$$\begin{aligned} \text{subject to} \quad \dot{\tilde{\mathbf{x}}} &= \mathbf{f}(\tilde{\mathbf{x}}, \mathbf{x}_5, \mathbf{x}_{13}, \mathbf{u}) \\ \dot{\mathbf{x}_5} &= g(\tilde{\mathbf{x}}, \mathbf{x}_5, \mathbf{x}_{13}, \mathbf{u}) \\ \dot{\mathbf{x}_{13}} &= h(\tilde{\mathbf{x}}, \mathbf{x}_5, \mathbf{x}_{13}, \mathbf{u}) \\ \mathbf{x}_5 &\leq 313, \quad \text{or} \quad \mathbf{x}_5 \leq 5t^2 + 10t + 293 \\ 173 &\leq \mathbf{u} \leq 473, \end{aligned} \tag{4.3b}$$

Infeasible Models This approach was implemented and tested. SINDYc was able to fit both linear and quadratic empirical models with accuracy comparable to those in Table 4.2. The accuracies are $MSE = 0.0282$, $MAE = 0.0724$, and $MSE = 0.0136$, $MAE = 0.0548$ for the linear and quadratic models respectively. However, the OCP of these models proved to be infeasible.

To investigate the cause of the infeasibility, various other configurations up to a cubic polynomial fit with interaction terms were tested. All configurations were found to be infeasible even with the state constraints removed. This suggests that the decoupling method may result in empirical system models that exhibit aphysical behaviour. It was suspected that the resultant systems may be uncontrollable, thus the controllability function `ctrb` in MATLAB was used to calculate the controllability of the linear system. However, based on `ctrb`'s Gramian calculations, there is no uncontrollable state. Further investigation revealed that CasADi is unable to simulate any of the empirical models with the control inputs set at their default values of 273 K. Namely, the under-

lying Newton solver in CASADI cannot converge. This suggests that it was not the OCP itself that was infeasible; rather, the empirical models may have unbounded trajectories and/or other intractable properties. Further work is required to understand the cause of the empirical model's aphysical behaviour; this is discussed in Section 5.2.4.

4.1.4.2 Constraint/Objective Mapping by Integrating Decoder into PYOMO

As the autoencoder-decoupling method resulted in intractable empirical models, an alternate approach was devised in which PYOMO will decode $\tilde{\mathbf{x}}$ to obtain $\hat{\mathbf{x}_5}$, thus using $\hat{\mathbf{x}_5}$ to verify constraint satisfaction, e.g. $\hat{\mathbf{x}_5} \leq 313$. Ideally, PYOMO would make calls to the decoder network, but it was found that PYOMO do not accept external function evaluations in the formulation of the model objective and constraints.

Implementing the neural network forward pass within Pyomo A workaround to PYOMO being unable to call the external decoder was to directly compute the decoder forward pass in PYOMO. To derive the analytical expressions for the decoder component functions, the weights and biases from each of the 4 decoder layers were extracted in PYTORCH. Technical implementation details are given in Appendix D.

This workaround is compatible only activation functions that are not conditional: this precludes the use of all -eLU activation functions in the decoder network, which are conditionally defined. This limitation likely arises out of the underlying IPOPT solver. As a gradient-based solver, it is not designed to work with non-differentiable functions. Thus, the decoder network used with this method was adapted to use `tanh` activation functions. It is recognised that such a workaround is rather inelegant: it adds significant overhead to the OCP, due to the matrix multiplications needed to decode $\tilde{\mathbf{x}}$. This was found to negate the time-savings gained by model reduction significantly. More importantly, the model accuracy obtained using this workaround is too low to be useful for control. The results obtained via this workaround are shown and further discussed in Section 4.2.2.3.

4.2 Results and Evaluation

The results obtained using the autoencoder-SINDYC methodology are given in this section. The configurations that were tested are:

1. Linear empirical model fitted on 3 $\tilde{\mathbf{x}}$ states in the ROM, corresponding to Row 9 in Table 4.2. Its results are given in 4.2.1.1.

2. Linear empirical model fitted on 1 $\tilde{\mathbf{x}}$ state in the ROM, corresponding to Row 1 in Table 4.2. Its results are given in 4.2.2.1.

3. Quadratic empirical model fitted on 1 $\tilde{\mathbf{x}}$ state in the ROM, corresponding to Row 2 in Table 4.2. Its results are given in 4.2.2.2.

4. Constrained quadratic empirical model fitted on 1 $\tilde{\mathbf{x}}$ state in the ROM, whose implementation is described in 4.1.4.2. Its results are given in 4.2.2.3.

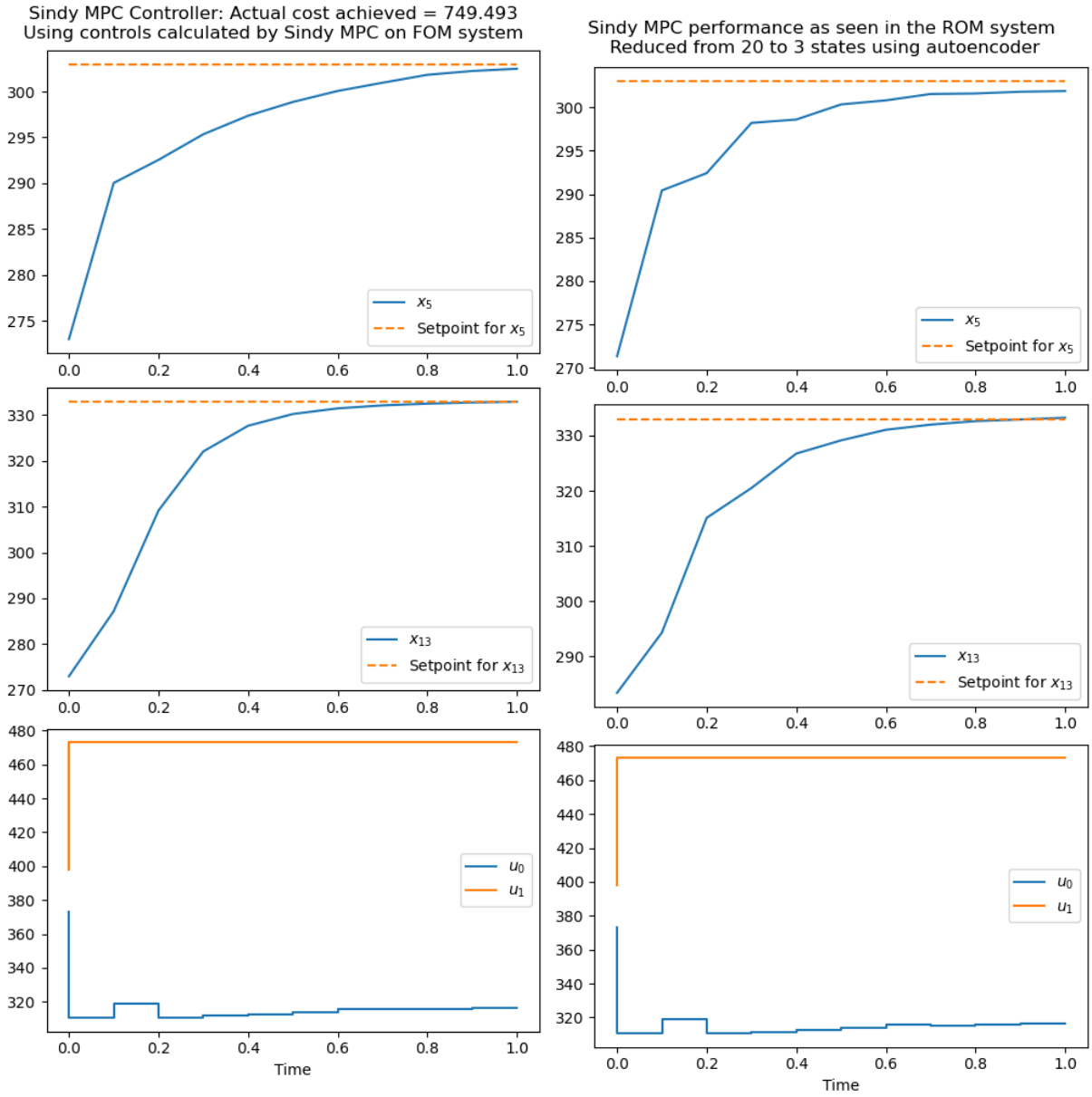
For the unconstrained reduced models, the computation time in PYOMO were greatly reduced, but with significant impact on performance in minimising the cost function. Generally, the results suggest that the reduced empirical models may be too simplistic to capture the dynamics of the FOM sufficiently for accurate OCP solutions. This can be observed from the simplistic control policies calculated: in all test cases, $\mathbf{u}_0$ was kept constantly low while $\mathbf{u}_1$ was pushed to its maximum of 473 K for the entire tracjectory. Although such a control policy is functional in driving the system towards the setpoints, it is clearly suboptimal when compared to the MPC performance on the original FOM. The optimal control trajectory corresponding to the FOM MPC is shown in Figure 3.10b from Test Scenario 2 in Chapter 3; it achieved a cost of 600, which is 19% less than the best performance attained from the ROM MPC at 741.

In the result figures 4.3, 4.4, 4.2 and 4.5, subfigures (b) show the ROM system trajectory seen by the ROM MPC, while subfigures (a) show the actual effect of the ROM MPC control policy on the FOM system. By comparing subfigures (a) and (b), the mismatch between the ROM and FOM dynamics can be observed. It can be seen that the empirical model fitted to 3 reduced states is considerably more accurate than that fitted to only 1 reduced state. Hence, the slightly better cost performance of the linear 1-state ROM over the 3-state ROM is likely only coincidental.

The time performance reported in the results are an average across 3 runs. This is the time taken by IPOPT to solve for the optimal control problem. The most extreme model reduction in Configuration 2 resulted in a ROM that could be solved in 0.103 s, reducing the solve time required by 93% as compared to the FOM. Configuration 1 resulted in a ROM that most balances the trade-off between accuracy and OCP solve time: it reduced the solve time by 90% while most preserving the system dynamics, as visible in Subfigures 4.2a and 4.2b.

4.2.1 Autoencoder-ROM with 3 Reduced States

4.2.1.1 Linear fit



(a) FOM system trajectory resulting from controls computed in ROM MPC ($Cost = 749$)

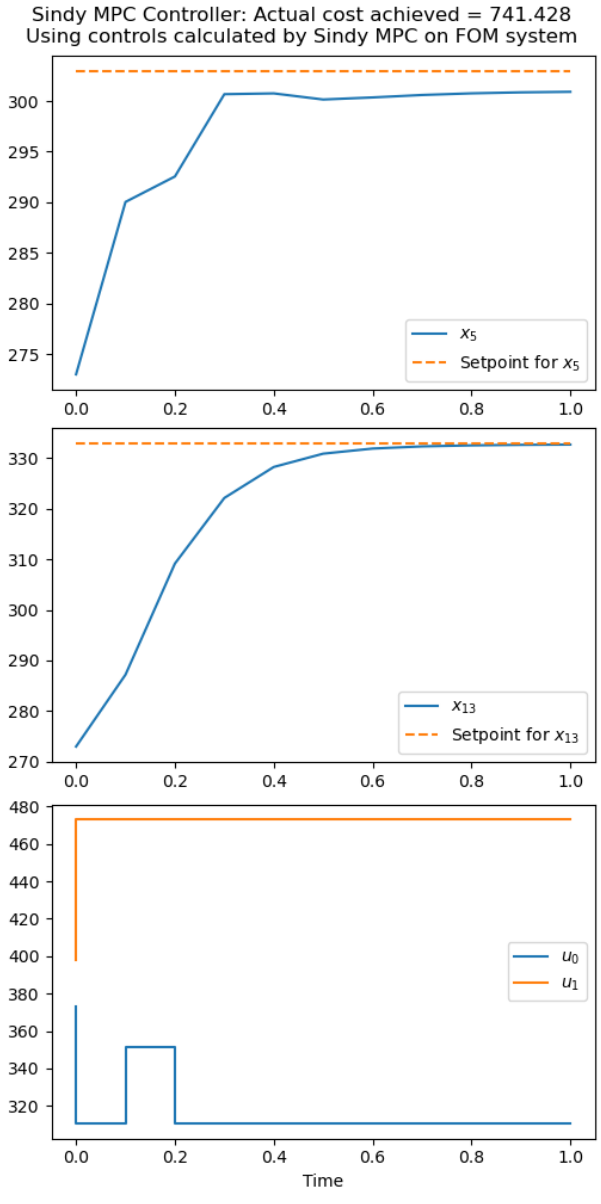
(b) ROM MPC system trajectory

Figure 4.2: MPC of ROM system with 3 reduced states; Linear ROM fitted by SINDYc

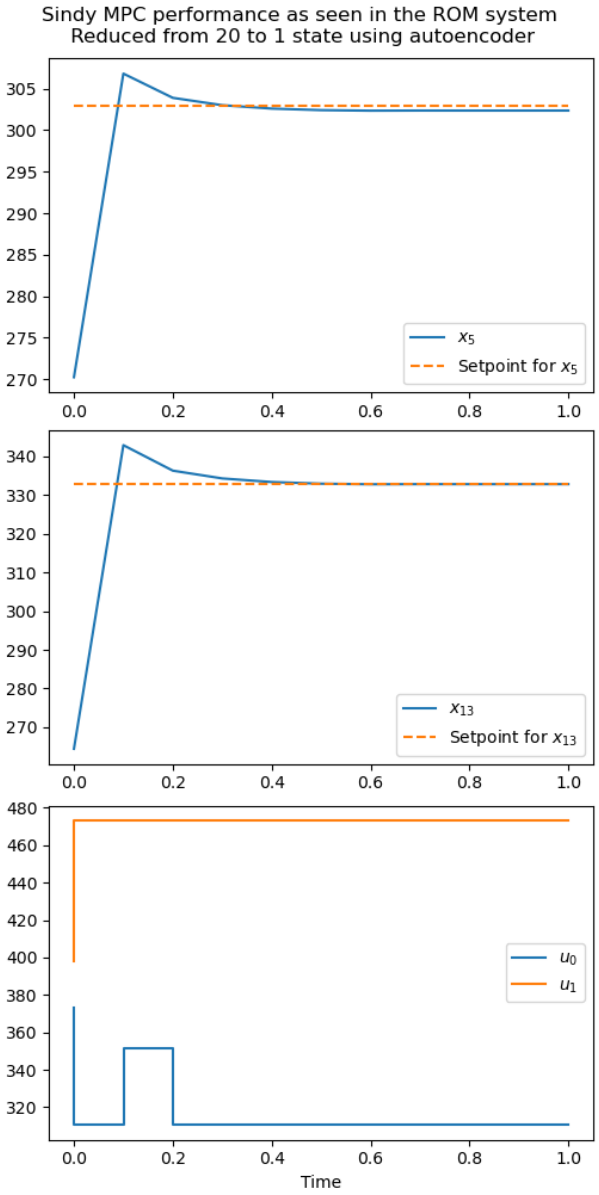
Time Performance The average solve time for the ROM OCP was 0.147 s, while it was 1.568 s for the corresponding FOM OCP from Scenario 2 of Chapter 3, but with the constraint $\mathbf{x}_5 \leq 313$ removed for a fair comparison. The FOM MPC achieved a cost of 600, which is 20% less than the cost of 749 achieved by the ROM MPC. However, the ROM solve time is sped up tenfold. Given the fast solve time, it may be possible to compensate for the ROM's inaccuracy by using smaller MPC sampling time steps.

4.2.2 Autoencoder-ROM with 1 Reduced State

4.2.2.1 Linear fit



(a) FOM system trajectory resulting from controls computed in ROM MPC ($Cost = 741$)

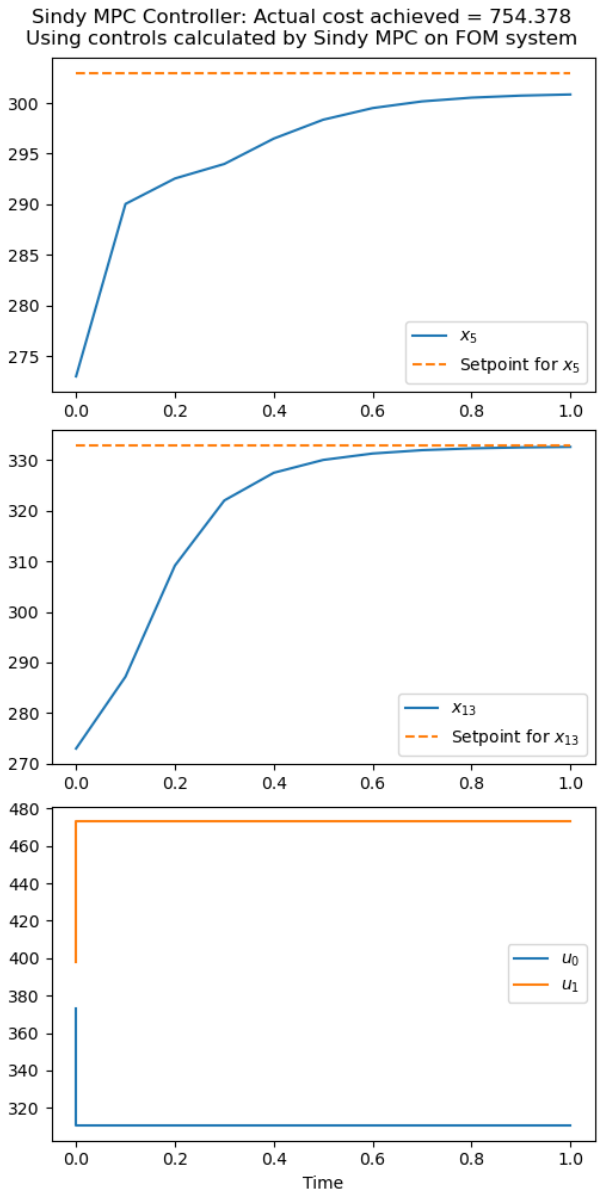


(b) ROM MPC system trajectory

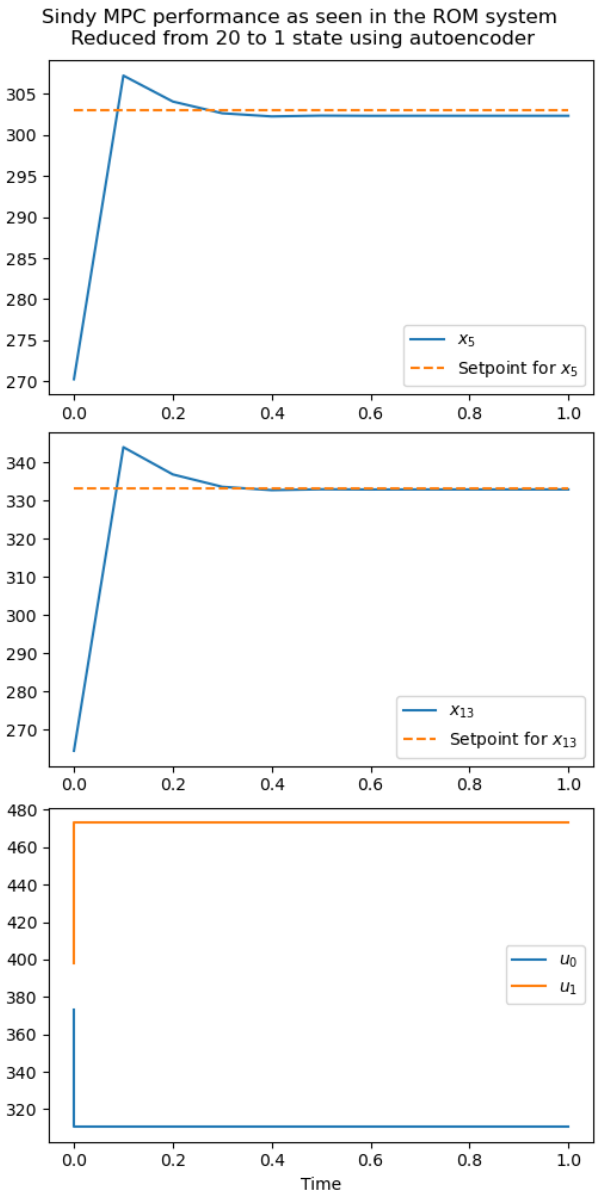
Figure 4.3: MPC of ROM system with 1 reduced state; Linear ROM fitted by SINDYc

Time Performance The average OCP solve time for this extremely reduced and linearly-fitted system was 0.103 s, as compared to 1.568 s for the corresponding FOM OCP. The 20-1 state reduction resulted in a significant loss of the original dynamics in the ROM, visible from the different trajectory profiles in the FOM and ROM when the same control actions are applied.

4.2.2.2 Quadratic fit



(a) FOM system trajectory resulting from controls computed in ROM MPC ($Cost = 754$)

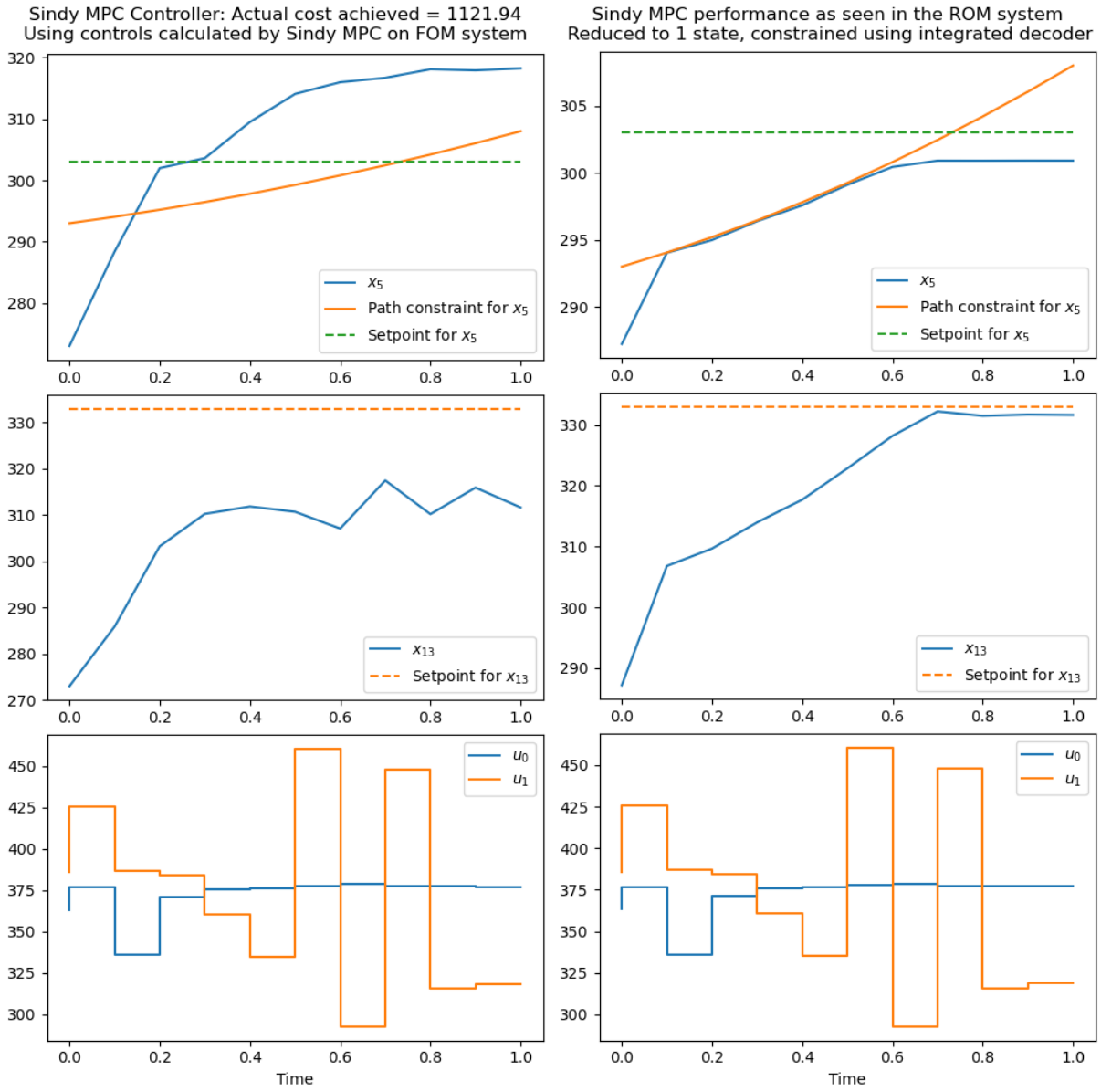


(b) ROM MPC system trajectory

Figure 4.4: MPC of ROM system with 1 reduced state; Quadratic ROM fitted by SINDYc

Time Performance As compared to the linear ROM's solve time of 0.103 s (Section 4.2.2.1), the quadratic ROM required an average time of 0.205 s to solve. This can be attributed to the increased OCP complexity caused by the nonlinear terms. The quadratic 1-state ROM was not more accurate than the linear 1-state ROM.

4.2.2.3 Quadratic Fit, Subject to State Constraints



(a) FOM system trajectory resulting from controls computed in ROM MPC

(b) ROM MPC system trajectory, constrained using workaround in Section 4.1.4.2

Figure 4.5: MPC of ROM system with 1 reduced state; Linear ROM fitted by SINDYc Constrained by integrating the decoder forward pass within PYOMO (Section 4.1.4.2)

Time Performance The average solve time for the ROM OCP was 1.543 s, while it was 4.037 s for the corresponding constrained FOM OCP from Scenario 3 of Chapter 3. Although the ROM was faster to solve, the ROM dynamics were too inaccurate for it to be usable for control. A minor part of this inaccuracy can be attributed to the replacing of the `elu` activation functions in the decoder to `tanh`, as described in Section 4.1.4.2: using `tanh` slightly reduced the prediction accuracies of the autoencoder. A larger part of this inaccuracy can perhaps be attributed to the poor extrapolation performance of

the empirical model, which will be discussed in Section 4.2.3. Further work is required to make the autoencoder-SINDYc approach feasible for the reduction of constrained control systems.

4.2.3 Discerning Between the Effects of State and Control Variables

While investigating the poor performance of the reduced empirical models for constrained systems, it was observed that SINDYc may be unable to distinguish between a change in $\mathbf{x}$ due to the system's internal dynamics and due to external controls. This has implication on the extrapolation performance of the empirical model. The difference between the actual and empirical models can be more clearly illustrated through a simple system: using a example from [88], a simple system is simulated to obtain trajectory data, which is fitted onto an empirical model by SINDYc with virtually perfect accuracy against a test data set. The system has only 2 state variables and 1 control input:

$$\dot{x_0}(t) = x_1(t) \tag{4.4a}$$
$$\dot{x_1}(t) = -x_1(t) + u(t) \tag{4.4b}$$
$$x_0(t=0) = 0 \tag{4.4c}$$
$$x_2(t=0) = -1 \tag{4.4d}$$

The empirical model fitted by SINDYc is:

$$\dot{x_0}(t) = 1.004 x_1(t) \tag{4.5a}$$
$$\dot{x_1}(t) = -0.3721 - 0.164\ x_0(t) - 2.252\ x_1(t) + 0.588\ u(t) \tag{4.5b}$$

While the empirical model can almost perfectly approximate the actual model within its data support regime, it is characterised by fundamentally different dynamics (Equation 4.5b). Thus, the empirical model is unlikely to perform well beyond the operating range for which it was trained. This includes the range of both state and controller variables, thus providing a possible explanation to the poor performance for constrained cases: for the same system model, the MPC control profiles are different in constrained and unconstrained cases, while the empirical models were fitted using only data from the unconstrained system. Further work is necessary to determine if this is indeed the cause of the inaccuracies.

Chapter 5

Conclusions

5.1 Thesis Contributions

This thesis makes 2 primary contributions in developing 2 practical methodologies for the optimal control of high-dimensional dynamical systems:

1. In Chapter 3, it was demonstrated that the optimal control of both linear and nonlinear systems can be implemented via learning the optimal value function of the control problem. This approach improves the time complexity of solving an optimal control problem by eliminating the effect of nonlinear terms and of the state space on computation time: the search time for optimal controls was previously dependent on the complexity of the governing equations, and on the number of state and control variables. In contrast, it is now only dependent on the size of the control action search space.

   It was further proved that the value function approximation remains accurate if a high-dimensional system is projected onto a reduced-state space, such that the value approximation neural network trains only on data from the reduced state variables. Finally, by extending the network to predict constraint deviation values, it was demonstrated that state constraints can be satisfied while controlling the system.

2. In Chapter 4, it was demonstrated that the combination of using an autoencoder for model reduction and SINDYc for system identification is effective for the optimal control of a high-dimensional nonlinear system without state constraints. In particular, it was demonstrated that an autoencoder can reduce a full-order model of 20 states to a reduced-order model of only 1 state, while still preserving sufficiently fidelity for control.

   Minor contributions are made in devising methodologies to map the control ob-

jective and state constraint(s) of the full-order model to the reduced space. It was demonstrated that control objectives can be mapped by projecting the ideal steady-state of the full system to the reduced space. However, the "decoupling" methodology, where states that are subject to constraints are not compressed through the encoder, was not found to be feasible. This is a significant drawback that requires further work (Section 5.2.4).

5.2 Further Work

A summary of the further work that could improve the implementations in this thesis is given in this section.

5.2.1 Systematic Hyperparameter Tuning using RAYTUNE

The hyperparameters for both the value approximation and the autoencoder neural networks (Chapter 3, Table 3.1, and Chapter 4, Table 4.1 respectively) were tuned by trial and error. These configurations were found to provide satisfactory performance for optimal control. Where there are performance deficiencies in control, it was observed that the bottlenecks were not in the network prediction (in)accuracies. Thus, hand-tuned configurations were used without a more exhaustive search of the hyperparameter configuration space. However, it is likely that the performance of both neural networks can be further improved with a more thorough hyperparameter search.

RAYTUNE [89], a hyperparameter tuning library developed to work with PYTORCH neural networks, was identified as an improvement to this project. It offers an extensive selection of optimisation algorithms, including genetic algorithms ("Population Based Training"), Bayesian optimisation, and brute-force grid search. Thus, further work in implementing RAYTUNE is expected to improve both optimal control methods in this thesis.

5.2.2 Replacing Basin-Hopping Search with a Neural Network

As discussed in Section 3.4.4, it may be possible to train a new control network to learn the $\min_{\mathbf{u}} V(\mathbf{x}, \mathbf{u})$ function, which would supplant the need for an iterative basin-hopper/Powell search. The data required to train this new network can be generated by performing an extensive search on the entire control action and state variable search space while recording $V(\mathbf{x}, \mathbf{u})$, $\mathbf{x}$, and $\mathbf{u}$. It may also take advantage of the model reduction within the value NN by searching in the reduced state search space, thus recording $V(\tilde{\mathbf{x}}, \mathbf{u})$, $\tilde{\mathbf{x}}$, and $\mathbf{u}$. In the latter case, the model reducer in the value NN would first be called to obtain $\tilde{\mathbf{x}}$, which would then be passed to the new control network to obtain $\mathbf{u}^*$.

5.2.3 Improving Constraint Handling in the NN Controller

As seen in Scenario 3 in Chapter 3, the constraint prediction accuracy of the value NN can be improved. This could possibly be accomplished by adding a time node as an input to the value NN, thus allowing the value NN to learn the relation between constraint satisfaction and the current trajectory progress for systems with time-variant path constraints. Constraint handling may also improve with hyperparameter tuning, as the penalty applied to the cost-to-go is a training hyperparameter (Row 9, Table 3.1) that determines the "eagerness" of the NN controller to attempt cheaper control paths that risk constraint violation.

5.2.4 Implementing Effective Constraint Handling in SINDYc-ROM Systems

Further work is necessary in discovering why the autoencoder decoupling method resulted in infeasible empirical models, as discussed in Section 4.1.4.1. This is the more preferable method for constraint handling in the empirical ROM, as the decoder forward pass workaround demonstrated to be inaccurate and inelegant in that it negates much of the computational efficiency gained from model reduction. Alternatively, further work is necessary in devising other methods to map state constraints from a full system model to an empirical ROM obtained from SINDYc.

5.3 Legal, Social, Ethical, and Professional Evaluation

The discussion in this section is made with reference to the Imperial College research ethics checklist, given in Appendix A. There are no implications identified in terms of privacy, environmental protection, animal rights, social equality, and legality. This thesis project do not involve the collection of personal data nor biological matter such as human cells. The contributions of this thesis to efficient optimal control is not expected to cause any harm to the environment, and has minute possibility of being abused for malevolent purposes relating to Section 9 of the checklist.

Relating to Section 8, optimal control methods for dynamical systems are applicable to military purposes. Indeed, 2 authors whose work was frequently cited in this thesis and who made significant contributions in this field, Brunton and Kutz [9, 42, 43], are partly funded by branches of the United States military [42, 43]. As the study of dynamical systems ranges widely, military applications are expected to be only a small subset of the possible applications. Finally, relating to Section 10, the software libraries used in this thesis are mostly open source projects. These libraries were used in compliance to their respective open source licenses. The only proprietary software used was MATLAB, which was used in accordance to the academic software license obtained via Imperial College, i.e. only for academic purposes.

References

[1] P. Benner, V. Mehrmann, and D. C. Sorensen, *Dimension reduction of large-scale systems*. Springer, 2005, vol. 45.

[2] L. Sirovich, "Turbulence and the dynamics of coherent structures. ii. symmetries and transformations", *Quarterly of Applied Mathematics*, vol. 45, no. 3, pp. 573–582, 1987, ISSN: 0033-569X. DOI: 10.1090/qam/910463.

[3] J. Lumley, "Coherent structures in turbulence", in *Transition and Turbulence*, R. E. MEYER, Ed., Academic Press, 1981, pp. 215–242, ISBN: 978-0-12-493240-1. DOI: https://doi.org/10.1016/B978-0-12-493240-1.50017-X. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B978012493240150017X.

[4] G. Berkooz, P. Holmes, and J. L. Lumley, "The proper orthogonal decomposition in the analysis of turbulent flows", *Annual Review of Fluid Mechanics*, vol. 25, no. 1, pp. 539–575, 1993, ISSN: 0066-4189. DOI: 10.1146/annurev.fl.25.010193.002543.

[5] C. W. ROWLEY, "Model reduction for fluids, using balanced proper orthogonal decomposition", *International Journal of Bifurcation and Chaos*, vol. 15, no. 03, pp. 997–1013, 2005. DOI: 10.1142/S0218127405012429. eprint: https://doi.org/10.1142/S0218127405012429. [Online]. Available: https://doi.org/10.1142/S0218127405012429.

[6] M. Ababneh, M. Salah, and K. Alwidyan, "Linearization of nonlinear dynamical systems: A comparative study", *Jordan Journal of Mechanical and Industrial Engineering*, vol. 5, no. 6, pp. 567–571, 2011.

[7] J.-J. E. Slotine, W. Li, *et al.*, *Applied nonlinear control*, 1. Prentice hall Englewood Cliffs, NJ, 1991, vol. 199.

[8] L. Shieh, H. Wang, and R. Yates, "Discrete-continuous model conversion", *Applied Mathematical Modelling*, vol. 4, no. 6, pp. 449–455, 1980.

[9] S. L. Brunton and J. N. Kutz, *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press, 2019. DOI: 10.1017/9781108380690.

[10] A. Afram and F. Janabi-Sharifi, "Theory and applications of hvac control systems–a review of model predictive control (mpc)", *Building and Environment*, vol. 72, pp. 343–355, 2014.

[11] M. Behrendt, *A basic working principle of Model Predictive Control*, Oct. 2009. [Online]. Available: https://en.wikipedia.org/wiki/File:MPC_scheme_basic.svg.

[12] M. L. Bynum, G. A. Hackebeil, W. E. Hart, C. D. Laird, B. L. Nicholson, J. D. Siirola, J.-P. Watson, and D. L. Woodruff, *Pyomo–optimization modeling in python*, Third. Springer Science & Business Media, 2021, vol. 67.

[13] W. E. Hart, J.-P. Watson, and D. L. Woodruff, "Pyomo: Modeling and solving mathematical programs in python", *Mathematical Programming Computation*, vol. 3, no. 3, pp. 219–260, 2011.

[14] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming", *Mathematical Programming*, vol. 106, no. 1, pp. 25–57, 2006, ISSN: 0025-5610. DOI: 10.1007/s10107-004-0559-y. [Online]. Available: https://dx.doi.org/10.1007/s10107-004-0559-y.

[15] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi – A software framework for nonlinear optimization and optimal control", *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019. DOI: 10.1007/s12532-018-0139-4.

[16] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library", in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[17] C. Wu, Y. Liang, W. Lin, H. Lee, and S. Lim, "A note on equivalence of proper orthogonal decomposition methods", *Journal of Sound and Vibration*, vol. 265, no. 5, pp. 1103–1110, 2003, ISSN: 0022-460X. DOI: 10.1016/s0022-460x(03)00032-4.

[18] Y. Liang, H. Lee, S. Lim, W. Lin, K. Lee, and C. Wu, "Proper orthogonal decomposition and its applications—part i: Theory", *Journal of Sound and Vibration*, vol. 252, no. 3, pp. 527–544, 2002, ISSN: 0022-460X. DOI: 10.1006/jsvi.2001.4041.

[19] T. R. Smith, J. Moehlis, and P. Holmes, "Low-dimensional modelling of turbulence using the proper orthogonal decomposition: A tutorial", *Nonlinear Dynamics*, vol. 41, no. 1-3, pp. 275–307, 2005, ISSN: 0924-090X. DOI: 10.1007/s11071-005-2823-y.

[20] M. Khalil, S. Adhikari, and A. Sarkar, "Linear system identification using proper orthogonal decomposition", *Mechanical Systems and Signal Processing*, vol. 21, no. 8, pp. 3123–3145, 2007, ISSN: 0888-3270. DOI: 10.1016/j.ymssp.2007.03.007.

[21] C. W. Rowley and S. T. Dawson, "Model reduction for flow analysis and control", *Annual Review of Fluid Mechanics*, vol. 49, no. 1, pp. 387–417, 2017, ISSN: 0066-4189. DOI: 10.1146/annurev-fluid-010816-060042.

[22] M. Ilak and C. W. Rowley, "Modeling of transitional channel flow using balanced proper orthogonal decomposition", *Physics of Fluids*, vol. 20, no. 3, p. 034 103, 2008, ISSN: 1070-6631. DOI: 10.1063/1.2840197. [Online]. Available: http://arxiv.org/pdf/0707.4112.

[23] V. B. Nguyen, S. B. Q. Tran, S. A. Khan, J. Rong, and J. Lou, "Pod-deim model order reduction technique for model predictive control in continuous chemical processing", *Computers & Chemical Engineering*, vol. 133, p. 106 638, 2020, ISSN: 0098-1354. DOI: 10.1016/j.compchemeng.2019.106638.

[24] S. Chaturantabut and D. C. Sorensen, "Nonlinear model reduction via discrete empirical interpolation", *SIAM Journal on Scientific Computing*, vol. 32, no. 5, pp. 2737–2764, 2010, ISSN: 1064-8275. DOI: 10.1137/090766498. [Online]. Available: https://scholarship.rice.edu/bitstream/1911/70218/1/ChaturantabutS.pdf.

[25] K. Willcox and J. Peraire, "Balanced model reduction via the proper orthogonal decomposition", *AIAA Journal*, vol. 40, no. 11, pp. 2323–2330, 2002, ISSN: 0001-1452 1533-385X. DOI: 10.2514/2.1570.

[26] C. W. Rowley, T. Colonius, and R. M. Murray, "Model reduction for compressible flows using pod and galerkin projection", *Physica D: Nonlinear Phenomena*, vol. 189, no. 1-2, pp. 115–129, 2004, ISSN: 0167-2789. DOI: 10.1016/j.physd.2003.03.001.

[27] B. Moore, "Principal component analysis in linear systems: Controllability, observability, and model reduction", *IEEE Transactions on Automatic Control*, vol. 26, no. 1, pp. 17–32, 1981, ISSN: 0018-9286. DOI: 10.1109/tac.1981.1102568.

[28] T. Penzl, *Advances in Computational Mathematics*, vol. 8, no. 1/2, pp. 33–48, 1998, ISSN: 1019-7168. DOI: 10.1023/a:1018979826766.

[29] T. R. F. Phillips, C. E. Heaney, P. N. Smith, and C. C. Pain, "An autoencoder-based reduced-order model for eigenvalue problems with application to neutron diffusion", *International Journal for Numerical Methods in Engineering*, vol. 122, no. 15, pp. 3780–3811, 2021, ISSN: 0029-5981. DOI: 10.1002/nme.6681.

[30] E. Oja, "Simplified neuron model as a principal component analyzer", *Journal of Mathematical Biology*, vol. 15, no. 3, pp. 267–273, 1982, ISSN: 0303-6812. DOI: 10.1007/bf00275687.

[31] H. Bourlard and Y. Kamp, "Auto-association by multilayer perceptrons and singular value decomposition", *Biological Cybernetics*, vol. 59, no. 4-5, pp. 291–294, 1988, ISSN: 0340-1200. DOI: 10.1007/bf00332918.

[32] Y. Wang, H. Yao, and S. Zhao, "Auto-encoder based dimensionality reduction", *Neurocomputing*, vol. 184, pp. 232–242, 2016, ISSN: 0925-2312. DOI: 10.1016/j.neucom.2015.08.104.

[33] *Dimensionality reduction strategy based on auto-encoder*, 2015. DOI: 10.1145/2808492.2808555.

[34] J. Wang, H. He, and D. V. Prokhorov, "A folded neural network autoencoder for dimensionality reduction", *Procedia Computer Science*, vol. 13, pp. 120–127, 2012, ISSN: 1877-0509. DOI: 10.1016/j.procs.2012.09.120. [Online]. Available: https://doi.org/10.1016/j.procs.2012.09.120.

[35] M. Ramamurthy, Y. H. Robinson, S. Vimal, and A. Suresh, "Auto encoder based dimensionality reduction and classification using convolutional neural networks for hyperspectral images", *Microprocessors and Microsystems*, vol. 79, p. 103 280, 2020, ISSN: 0141-9331. DOI: 10.1016/j.micpro.2020.103280.

[36] L. Agostini, "Exploration and prediction of fluid dynamical systems using auto-encoder technology", *Physics of Fluids*, vol. 32, no. 6, p. 067 103, 2020, ISSN: 1070-6631. DOI: 10.1063/5.0012906.

[37] S. Fresca, L. Dede', and A. Manzoni, "A comprehensive deep learning-based approach to reduced order modeling of nonlinear time-dependent parametrized pdes", *Journal of Scientific Computing*, vol. 87, no. 2, 2021, ISSN: 0885-7474. DOI: 10.1007/s10915-021-01462-7.

[38] P. Wu, S. Gong, K. Pan, F. Qiu, W. Feng, and C. Pain, "Reduced order model using convolutional auto-encoder with self-attention", *Physics of Fluids*, vol. 33, no. 7, p. 077 107, 2021, ISSN: 1070-6631. DOI: 10.1063/5.0051155.

[39] K. Lee and K. T. Carlberg, "Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders", *Journal of Computational Physics*, vol. 404, p. 108 973, 2020, ISSN: 0021-9991. DOI: 10.1016/j.jcp.2019.108973.

[40] P. Rajendra and V. Brahmajirao, "Modeling of dynamical systems through deep learning", *Biophysical Reviews*, vol. 12, no. 6, pp. 1311–1320, 2020, ISSN: 1867-2450. DOI: 10.1007/s12551-020-00776-4.

[41] K. Champion, B. Lusch, J. N. Kutz, and S. L. Brunton, "Data-driven discovery of coordinates and governing equations", *Proceedings of the National Academy of Sciences*, vol. 116, no. 45, pp. 22 445–22 451, 2019, ISSN: 0027-8424. DOI: 10.1073/pnas.1906995116.

[42] S. L. Brunton, J. L. Proctor, and J. N. Kutz, "Discovering governing equations from data by sparse identification of nonlinear dynamical systems", *Proceedings of the National Academy of Sciences*, vol. 113, no. 15, pp. 3932–3937, 2016, ISSN: 0027-8424. DOI: 10.1073/pnas.1517384113. [Online]. Available: https://www.pnas.org/content/pnas/113/15/3932.full.pdf.

[43] ——, "Sparse identification of nonlinear dynamics with control (sindyc)**slb acknowledges support from the u.s. air force center of excellence on nature inspired flight technologies and ideas (fa9550-14-1-0398). jlp thanks bill and melinda gates for their activ", *IFAC-PapersOnLine*, vol. 49, no. 18, pp. 710–715, 2016, ISSN: 2405-8963. DOI: 10.1016/j.ifacol.2016.10.249.

[44] E. Kaiser, J. N. Kutz, and S. L. Brunton, "Sparse identification of nonlinear dynamics for model predictive control in the low-data limit", *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 474, no. 2219, p. 20 180 335, 2018, ISSN: 1364-5021. DOI: 10.1098/rspa.2018.0335. [Online]. Available: http://arxiv.org/pdf/1711.05501.

[45] G. Cybenko, "Approximation by superpositions of a sigmoidal function", *Mathematics of Control, Signals, and Systems*, vol. 2, no. 4, pp. 303–314, 1989, ISSN: 0932-4194. DOI: 10.1007/bf02551274.

[46] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators", *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989, ISSN: 0893-6080. DOI: 10.1016/0893-6080(89)90020-8. [Online]. Available: http://linkinghub.elsevier.com/retrieve/articleSelectSinglePerm.

[47] K. Hornik, "Approximation capabilities of multilayer feedforward networks", *Neural Networks*, vol. 4, no. 2, pp. 251–257, 1991, ISSN: 0893-6080. DOI: 10.1016/0893-6080(91)90009-t.

[48] J. Castro, C. Mantas, and B. J.M., "Neural networks with a continuous squashing function in the output are universal approximators", *Neural Networks*, vol. 13, no. 6, pp. 561–563, 2000, ISSN: 0893-6080. DOI: 10.1016/s0893-6080(00)00031-9.

[49] I. Ciuca and J. A. Ware, "Layered neural networks as universal approximators", in *Lecture Notes in Computer Science*. Lecture Notes in Computer Science, 1997, pp. 411–415. DOI: 10.1007/3-540-62868-1_133.

[50] B. Lang, "Monotonic multi-layer perceptron networks as universal approximators", in *Lecture Notes in Computer Science*. Lecture Notes in Computer Science, 2005, pp. 31–37. DOI: 10.1007/11550907_6.

[51] A. Draeger, S. Engell, and H. Ranke, "Model predictive control using neural networks", *IEEE Control Systems*, vol. 15, no. 5, pp. 61–66, 1995, ISSN: 1066-033X. DOI: 10.1109/37.466261.

[52] B.-K. Jeon and E.-J. Kim, "Lstm-based model predictive control for optimal temperature set-point planning", *Sustainability*, vol. 13, no. 2, p. 894, 2021, ISSN: 2071-1050. DOI: 10.3390/su13020894.

[53] Z. Zou, X. Yu, and S. Ergan, "Towards optimal control of air handling units using deep reinforcement learning and recurrent neural network", *Building and Environment*, vol. 168, p. 106 535, 2020, ISSN: 0360-1323. DOI: 10.1016/j.buildenv.2019.106535.

[54] J. Saint-Donat, N. Bhat, and T. J. Mcavoy, "Neural net based model predictive control", *International Journal of Control*, vol. 54, no. 6, pp. 1453–1468, 1991, ISSN: 0020-7179. DOI: 10.1080/00207179108934221.

[55] K. Hunt, D. Sbarbaro, R. Żbikowski, and P. Gawthrop, "Neural networks for control systems—a survey", *Automatica*, vol. 28, no. 6, pp. 1083–1112, 1992, ISSN: 0005-1098. DOI: 10.1016/0005-1098(92)90053-i.

[56] S. Piche, B. Sayyar-Rodsari, D. Johnson, and M. Gerules, "Nonlinear model predictive control using neural networks", *IEEE Control Systems*, vol. 20, no. 3, pp. 53–62, 2000, ISSN: 1066-033X. DOI: 10.1109/37.845038.

[57] Y. Pan and J. Wang, "Model predictive control of unknown nonlinear dynamical systems based on recurrent neural networks", *IEEE Transactions on Industrial Electronics*, vol. 59, no. 8, pp. 3089–3101, 2012, ISSN: 0278-0046. DOI: 10.1109/tie.2011.2169636.

[58] E. Maddalena, C. D. S. Moraes, G. Waltrich, and C. Jones, "A neural network architecture to learn explicit mpc controllers from data", *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 11 362–11 367, 2020, ISSN: 2405-8963. DOI: 10.1016/j.ifacol.2020.12.546.

[59] *Approximating Explicit Model Predictive Control Using Constrained Neural Networks*, 2018. DOI: 10.23919/acc.2018.8431275.

[60] K. Kiš and M. Klaučo, *Neural network based explicit mpc for chemical reactor control*, 2019. arXiv: 1912.04684 [cs.LG].

[61] B. M. Åkesson and H. T. Toivonen, "A neural network model predictive controller", *Journal of Process Control*, vol. 16, no. 9, pp. 937–946, 2006, ISSN: 0959-1524. DOI: 10.1016/j.jprocont.2006.06.001.

[62] K. Kosanovich, A. Gurumoorthy, E. Sinzinger, and M. Piovoso, "Improving the extrapolation capability of neural networks", in. DOI: 10.1109/isic.1996.556233.

[63] E. Barnard and L. Wessels, "Extrapolation and interpolation in neural network classifiers", *IEEE Control Systems*, vol. 12, no. 5, pp. 50–53, 1992, ISSN: 1066-033X. DOI: 10.1109/37.158898.

[64] P. Haley and D. Soloway, "Extrapolation limitations of multilayer feedforward neural networks", in. DOI: 10.1109/ijcnn.1992.227294.

[65] D. Ernst, M. Glavic, F. Capitanescu, and L. Wehenkel, "Reinforcement learning versus model predictive control: A comparison on a power system problem", *IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics)*, vol. 39, no. 2, pp. 517–529, 2009, ISSN: 1083-4419. DOI: 10.1109/tsmcb.2008.2007630.

[66] Y. Lin, J. Mcphee, and N. L. Azad, "Comparison of deep reinforcement learning and model predictive control for adaptive cruise control", *IEEE Transactions on Intelligent Vehicles*, vol. 6, no. 2, pp. 221–231, 2021, ISSN: 2379-8904. DOI: 10.1109/tiv.2020.3012947.

[67] R. Sutton, A. Barto, and R. Williams, "Reinforcement learning is direct adaptive optimal control", *IEEE Control Systems*, vol. 12, no. 2, pp. 19–22, 1992, ISSN: 1066-033X. DOI: 10.1109/37.126844.

[68] C. J. C. H. Watkins and P. Dayan, "Q-learning", *Machine Learning*, vol. 8, no. 3-4, pp. 279–292, 1992, ISSN: 0885-6125. DOI: 10.1007/bf00992698.

[69] C. J. C. H. Watkins, "Learning from delayed rewards", Ph.D. dissertation, King's College, Cambridge, UK, May 1989. [Online]. Available: http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf.

[70] *Adaptive linear quadratic control using policy iteration*. DOI: 10.1109/acc.1994.735224.

[71] B. Kiumarsi, F. L. Lewis, H. Modares, A. Karimpour, and M.-B. Naghibi-Sistani, "Reinforcement -learning for optimal tracking control of linear discrete-time systems with unknown dynamics", *Automatica*, vol. 50, no. 4, pp. 1167–1175, 2014, ISSN: 0005-1098. DOI: 10.1016/j.automatica.2014.02.015.

[72] J. Y. Lee, J. B. Park, and Y. H. Choi, "Integral q-learning and explorized policy iteration for adaptive optimal control of continuous-time linear systems", *Automatica*, vol. 48, no. 11, pp. 2850–2859, 2012, ISSN: 0005-1098. DOI: 10.1016/j.automatica.2012.06.008.

[73] B. Luo, D. Liu, and T. Huang, *Q-learning for optimal control of continuous-time systems*, 2014. arXiv: 1410.2954 [cs.SY].

[74] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching", *Machine Learning*, vol. 8, no. 3-4, pp. 293–321, 1992, ISSN: 0885-6125. DOI: 10.1007/bf00992699.

[75] A. S. A. Al-Jilawi, "Solving heat equation of higher-dimensional of partial differential equations", *Journal of University of Babylon*, vol. 21, no. 4, 2013.

[76] R. Bellman, "The theory of dynamic programming", *Bulletin of the American Mathematical Society*, vol. 60, no. 6, pp. 503–515, 1954.

[77] ——, "On the theory of dynamic programming", *Proceedings of the National Academy of Sciences of the United States of America*, vol. 38, no. 8, p. 716, 1952.

[78] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu, "A limited memory algorithm for bound constrained optimization", *SIAM Journal on scientific computing*, vol. 16, no. 5, pp. 1190–1208, 1995.

[79] C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal, "Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization", *ACM Trans. Math. Softw.*, vol. 23, no. 4, pp. 550–560, Dec. 1997, ISSN: 0098-3500. DOI: 10.1145/279232.279236. [Online]. Available: https://doi.org/10.1145/279232.279236.

[80] J. A. Nelder and R. Mead, "A simplex method for function minimization", *The computer journal*, vol. 7, no. 4, pp. 308–313, 1965.

[81] M. J. D. Powell, "An efficient method for finding the minimum of a function of several variables without calculating derivatives", *The Computer Journal*, vol. 7, no. 2, pp. 155–162, Jan. 1964, ISSN: 0010-4620. DOI: 10.1093/comjnl/7.2.155. eprint: https://academic.oup.com/comjnl/article-pdf/7/2/155/959784/070155.pdf. [Online]. Available: https://doi.org/10.1093/comjnl/7.2.155.

[82] D. J. Wales and J. P. K. Doye, "Global optimization by basin-hopping and the lowest energy structures of lennard-jones clusters containing up to 110 atoms", *The Journal of Physical Chemistry A*, vol. 101, no. 28, pp. 5111–5116, 1997, ISSN: 1089-5639. DOI: 10.1021/jp970984n.

[83] B. Olson, I. Hashmi, K. Molloy, and A. Shehu, "Basin hopping as a general and versatile optimization framework for the characterization of biological macromolecules", *Advances in Artificial Intelligence*, vol. 2012, pp. 1–19, 2012, ISSN: 1687-7470. DOI: 10.1155/2012/674832.

[84] K. V. Price, "Differential evolution", in *Handbook of optimization*, Springer, 2013, pp. 187–214.

[85] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné, "DEAP: Evolutionary algorithms made easy", *Journal of Machine Learning Research*, vol. 13, pp. 2171–2175, Jul. 2012.

[86] *Vanishing Gradient Mitigation with Deep Learning Neural Network Optimization*, 2019. DOI: 10.1109/icscc.2019.8843652.

[87] B. Xu, N. Wang, T. Chen, and M. Li, *Empirical evaluation of rectified activations in convolutional network*, 2015. arXiv: 1505.00853 [cs.LG].

[88] B. Chachuat, "Nonlinear and dynamic optimization: From theory to practice", 2007.

[89] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica, "Tune: A research platform for distributed model selection and training", *arXiv preprint arXiv:1807.05118*, 2018.

Bibliography

[1] B. M. Åkesson and H. T. Toivonen, "A neural network model predictive controller", *Journal of Process Control*, vol. 16, no. 9, pp. 937–946, 2006, ISSN: 0959-1524. DOI: 10.1016/j.jprocont.2006.06.001.

[2] S. Sabzevari, R. Heydari, M. Mohiti, M. Savaghebi, and J. Rodriguez, "Model-free neural network-based predictive control for robust operation of power converters", *Energies*, vol. 14, no. 8, p. 2325, 2021, ISSN: 1996-1073. DOI: 10.3390/en14082325.

[3] P. Benner, S. Gugercin, and K. Willcox, "A survey of projection-based model reduction methods for parametric dynamical systems", *SIAM Review*, vol. 57, no. 4, pp. 483–531, 2015, ISSN: 0036-1445. DOI: 10.1137/130932715. [Online]. Available: https://dspace.mit.edu/bitstream/1721.1/100939/1/Benner-2015-Survey%20of%20projection-based.pdf.

[4] *Continuous Travel Time Prediction for Transit Signal Priority Based on a Deep Network*, 2015. DOI: 10.1109/itsc.2015.92.

[5] P. Baldi and K. Hornik, "Neural networks and principal component analysis: Learning from examples without local minima", *Neural Networks*, vol. 2, no. 1, pp. 53–58, 1989, ISSN: 0893-6080. DOI: 10.1016/0893-6080(89)90014-2.

[6] *Variational Autoencoder for End-to-End Control of Autonomous Driving with Novelty Detection and Training De-biasing*, 2018. DOI: 10.1109/iros.2018.8594386.

[7] *Neural networks for control*, 1999. DOI: 10.1109/acc.1999.786109.

[8] *Implicit vs explicit MPC — Similarities, differences, and a path owards a unified method*, 2016. DOI: 10.1109/ecc.2016.7810353.

[9] D. Görges, "Relations between model predictive control and reinforcement learning", *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 4920–4928, 2017, ISSN: 2405-8963. DOI: 10.1016/j.ifacol.2017.08.747.

[10] C. Greatwood and A. G. Richards, "Reinforcement learning and model predictive control for robust embedded quadrotor guidance and control", *Autonomous Robots*, vol. 43, no. 7, pp. 1681–1693, 2019, ISSN: 0929-5593. DOI: 10.1007/s10514-019-09829-4. [Online]. Available: https://link.springer.com/content/pdf/10.1007%2Fs10514-019-09829-4.pdf.

[11] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey", *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996, ISSN: 1076-9757. DOI: 10.1613/jair.301.

[12] *A comparison of direct and model-based reinforcement learning*. DOI: 10.1109/robot.1997.606886.

[13] K. G. Vamvoudakis, "Q-learning for continuous-time linear systems: A model-free infinite horizon optimal control approach", *Systems & Control Letters*, vol. 100, pp. 14–20, 2017, ISSN: 0167-6911. DOI: 10.1016/j.sysconle.2016.12.003.

[14] J. Li, T. Chai, F. L. Lewis, Z. Ding, and Y. Jiang, "Off-policy interleaved q -learning: Optimal control for affine nonlinear discrete-time systems", *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 5, pp. 1308–1320, 2019, ISSN: 2162-237X. DOI: 10.1109/tnnls.2018.2861945.

[15] B. Luo, Y. Yang, and D. Liu, "Adaptive q -learning for data-based optimal output regulation with experience replay", *IEEE Transactions on Cybernetics*, vol. 48, no. 12, pp. 3337–3348, 2018, ISSN: 2168-2267. DOI: 10.1109/tcyb.2018.2821369.

[16] *Q-learning with experience replay in a dynamic environment*, 2016. DOI: 10.1109/ssci.2016.7849368.

[17] S. Adam, L. Busoniu, and R. Babuska, "Experience replay for real-time reinforcement learning control", *IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 2, pp. 201–212, 2012, ISSN: 1094-6977. DOI: 10.1109/tsmcc.2011.2106494.

[18] *On-policy Q-learning for adaptive optimal control*, 2014. DOI: 10.1109/adprl.2014.7010649.

[19] S. J. Wright, "Interior point methods for optimal control of discrete time systems", *Journal of Optimization Theory and Applications*, vol. 77, no. 1, pp. 161–187, 1993, ISSN: 0022-3239. DOI: 10.1007/bf00940784.

[20] Q. Wang, J. S. Hesthaven, and D. Ray, "Non-intrusive reduced order modeling of unsteady flows using artificial neural networks with application to a combustion problem", *Journal of Computational Physics*, vol. 384, pp. 289–307, 2019, ISSN: 0021-9991. DOI: 10.1016/j.jcp.2019.01.031.

[21] N. Smaoui, "Linear versus nonlinear dimensionality reduction of high-dimensional dynamical systems", *SIAM Journal on Scientific Computing*, vol. 25, no. 6, pp. 2107–2125, 2004, ISSN: 1064-8275. DOI: 10.1137/s1064827502412723.

[22] M. Azaïez, L. Lestandi, and T. C. Rebollo, *Low Rank Approximation of Multidimensional Data*. 2019, pp. 187–250. DOI: 10.1007/978-3-030-17012-7_5.

[23] M. Diehl, H. J. Ferreau, and N. Haverbeke, *Efficient Numerical Methods for Nonlinear MPC and Moving Horizon Estimation*. 2009, pp. 391–417. DOI: 10.1007/978-3-642-01094-1_32.

[24] A. Marquez, J. J. E. Oviedo, and D. Odloak, "Model reduction using proper orthogonal decomposition and predictive control of distributed reactor system", *Journal of Control Science and Engineering*, vol. 2013, pp. 1–19, 2013, ISSN: 1687-5249. DOI: 10.1155/2013/763165. [Online]. Available: https://doi.org/10.1155/2013/763165.

[25] *Autoencoder-based Dimensionality Reduction for QSAR Modeling*, 2020. DOI: 10.1109/iccais48893.2020.9096747.

[26] *Fast robust model predictive control of high-dimensional systems*, 2015. DOI: 10.1109/ecc.2015.7330834.

[27] S. Wang, S. Khatir, and M. A. Wahab, "Proper orthogonal decomposition for the prediction of fretting wear characteristics", *Tribology International*, vol. 152, p. 106 545, 2020, ISSN: 0301-679X. DOI: 10.1016/j.triboint.2020.106545.

[28] G. Tadmor, "Observers and feedback control for a rotating vortex pair", *IEEE Transactions on Control Systems Technology*, vol. 12, no. 1, pp. 36–51, 2004, ISSN: 1063-6536. DOI: 10.1109/tcst.2003.821950.

[29] H. V. Ly and H. T. Tran, "Modeling and control of physical processes using proper orthogonal decomposition", *Mathematical and Computer Modelling*, vol. 33, no. 1-3, pp. 223–236, 2001, ISSN: 0895-7177. DOI: 10.1016/s0895-7177(00)00240-5.

[30] K. Kunisch and S. Volkwein, "Control of the burgers equation by a reduced-order approach using proper orthogonal decomposition", *Journal of Optimization Theory and Applications*, vol. 102, no. 2, pp. 345–371, 1999, ISSN: 0022-3239. DOI: 10.1023/a:1021732508059.

[31] G. Stabile, S. Hijazi, A. Mola, S. Lorenzi, and G. Rozza, "Pod-galerkin reduced order methods for cfd using finite volume discretisation: Vortex shedding around a circular cylinder", *Communications in Applied and Industrial Mathematics*, vol. 8, no. 1, pp. 210–236, 2017, ISSN: 2038-0909. DOI: 10.1515/caim-2017-0011. [Online]. Available: https://re.public.polimi.it/bitstream/11311/1048462/1.

[32] M. Isoz, "Pod-deim based model order reduction for speed-up of flow parametric studies", *Ocean Engineering*, vol. 186, p. 106 083, 2019, ISSN: 0029-8018. DOI: 10.1016/j.oceaneng.2019.05.065.

[33] S. Lall, J. E. Marsden, and S. Glavaški, "Empirical model reduction of controlled nonlinear systems", *IFAC Proceedings Volumes*, vol. 32, no. 2, pp. 2598–2603, 1999, ISSN: 1474-6670. DOI: 10.1016/s1474-6670(17)56442-3.

[34] M. Frasca, A. Rizzo, L. Gallo, L. Fortuna, and M. Porfiri, "Dimensionality reduction in epidemic spreading models", *EPL (Europhysics Letters)*, vol. 111, no. 6, p. 68 006, 2015.

Appendix A

Ethics Checklist

The ethics checklist used as reference for evaluating this thesis project's ethical, social, legal, and professional implications is given in the following two pages. The evaluation itself is given in Section 5.3 in the Conclusions chapter.

	Yes	No
Section 1: HUMAN EMBRYOS/FOETUSES		
Does your project involve Human Embryonic Stem Cells?		✓
Does your project involve the use of human embryos?		✓
Does your project involve the use of human foetal tissues / cells?		✓
Section 2: HUMANS		
Does your project involve human participants?		✓
Section 3: HUMAN CELLS / TISSUES		
Does your project involve human cells or tissues? (Other than from "Human Embryos/Foetuses" i.e. Section 1)?		✓
Section 4: PROTECTION OF PERSONAL DATA		
Does your project involve personal data collection and/or processing?		✓
Does it involve the collection and/or processing of sensitive personal data (e.g. health, sexual lifestyle, ethnicity, political opinion, religious or philosophical conviction)?		✓
Does it involve processing of genetic information?		✓
Does it involve tracking or observation of participants? It should be noted that this issue is not limited to surveillance or localization data. It also applies to Wan data such as IP address, MACs, cookies etc.		✓
Does your project involve further processing of previously collected personal data (secondary use)? For example Does your project involve merging existing data sets?		✓
Section 5: ANIMALS		
Does your project involve animals?		✓
Section 6: DEVELOPING COUNTRIES		
Does your project involve developing countries?		✓
If your project involves low and/or lower-middle income countries, are any benefit-sharing actions planned?		✓
Could the situation in the country put the individuals taking part in the project at risk?		✓
Section 7: ENVIRONMENTAL PROTECTION AND SAFETY		
Does your project involve the use of elements that may cause harm to the environment, animals or plants?		✓
Does your project deal with endangered fauna and/or flora /protected areas?		✓
Does your project involve the use of elements that may cause harm to humans, including project staff?		✓
Does your project involve other harmful materials or equipment, e.g. high-powered laser systems?		✓
Section 8: DUAL USE		
Does your project have the potential for military applications?	✓	
Does your project have an exclusive civilian application focus?		✓
Will your project use or produce goods or information that will require export licenses in accordance with legislation on dual use items?		✓

Does your project affect current standards in military ethics – e.g., global ban on weapons of mass destruction, issues of proportionality, discrimination of combatants and accountability in drone and autonomous robotics developments, incendiary or laser weapons?		✓
Section 9: MISUSE		
Does your project have the potential for malevolent/criminal/terrorist abuse?		✓
Does your project involve information on/or the use of biological-, chemical-, nuclear/radiological-security sensitive materials and explosives, and means of their delivery?		✓
Does your project involve the development of technologies or the creation of information that could have severe negative impacts on human rights standards (e.g. privacy, stigmatization, discrimination), if misapplied?		✓
Does your project have the potential for terrorist or criminal abuse e.g. infrastructural vulnerability studies, cybersecurity related project?		✓
SECTION 10: LEGAL ISSUES		
Will your project use or produce software for which there are copyright licensing implications?		✓
Will your project use or produce goods or information for which there are data protection, or other legal implications?		✓
SECTION 11: OTHER ETHICS ISSUES		
Are there any other ethics issues that should be taken into consideration?		✓

Appendix B

SINDYc-fitted ROM illustrative equation

The most accurate empirical system model fitted by SINDYc is shown below to illustrate the trade-off between fit accuracy and model complexity. This is the configuration in Row 4 of Table 4.2. The complexity demonstrates that, while the accuracy of the empirical model can be improved by adding more polynomial and/or Fourier terms, the resultant improved-accuracy model is practically unusable for optimal control.

$$
\begin{aligned}
\dot{\tilde{x}} = {} & 53512926584.7451 + 11537995584.112\ \tilde{x} + 7416074679.569\ u_0 + 26307817851.747\ u_1 + \\
& 5603801577.803\ \tilde{x}^2 + 5423604969.242\ \tilde{x}\ u_0 + 5769526621.472\ \tilde{x}\ u_1 + \\
& 480098697.633\ u_0^2 + 3708067723.704\ u_0\ u_1 + 18505591358.739\ u_1^2 + \\
& 1702944736.367\ \tilde{x}^3 + 3523503069.905\ \tilde{x}^2\ u_0 + 1868204338.595\ \tilde{x}^2\ u_1 + \\
& 315898269.390\ \tilde{x}\ u_0^2 + 1807847537.667\ \tilde{x}\ u_0\ u_1 + 423083.887\ \tilde{x}\ u_1^2 + 21407432.168\ u_0^3 + \\
& 160009972.763\ u_0^2\ u_1 + 20732.296\ u_0\ u_1^2 + 10703040211.818\ u_1^3 + 164930731.609\ \tilde{x}^4 + \\
& 1715545529.863\ \tilde{x}^3\ u_0 + 163224.742\ \tilde{x}^3\ u_1 + 155807186.939\ \tilde{x}^2\ u_0^2 + \\
& 51276.189\ \tilde{x}^2\ u_0\ u_1 + 1867816778.394\ \tilde{x}^2\ u_1^2 + 10580837.735\ \tilde{x}\ u_0^3 + 39088.345\ \tilde{x}\ u_0^2\ u_1 + \\
& 1807895178.799\ \tilde{x}\ u_0\ u_1^2 + 5769003448.854\ \tilde{x}\ u_1^3 + 661620.235\ u_0^4 + 1453.049\ u_0^3\ u_1 + \\
& 160036722.961\ u_0^2\ u_1^2 + 3708030757.347\ u_0\ u_1^3 + 2899975267.089\ u_1^4 + 1058.594\ \tilde{x}^5 + \\
& 16202.847\ \tilde{x}^4\ u_0 + 164915001.387\ \tilde{x}^4\ u_1 + 23412.586\ \tilde{x}^3\ u_0^2 + 1715624495.691\ \tilde{x}^3\ u_0\ u_1 + \\
& 1703058160.335\ \tilde{x}^3\ u_1^2 + 14172.128\ \tilde{x}^2\ u_0^3 + 155888106.846\ \tilde{x}^2\ u_0^2\ u_1 + \\
& 3523416109.768\ \tilde{x}^2\ u_0\ u_1^2 + 5604254940.234\ \tilde{x}^2\ u_1^3 + 4424.373\ \tilde{x}\ u_0^4 + \\
& 10614215.151\ \tilde{x}\ u_0^3\ u_1 + 315845895.081\ \tilde{x}\ u_0^2\ u_1^2 + 5423661714.890\ \tilde{x}\ u_0\ u_1^3 + \\
& 11538902008.682\ \tilde{x}\ u_1^4 + 647.514\ u_0^5 + 667095.786\ u_0^4\ u_1 + 21389607.775\ u_0^3\ u_1^2 + \\
& 480105897.998\ u_0^2\ u_1^3 + 7416103468.369\ u_0\ u_1^4 + 4903507263.361\ u_1^5
\end{aligned}
$$

Appendix C

Example of an infeasible model obtained using the decoupling method

This section pertains to the autoencoder-decoupling method discussed in Section 4.1.4.1. The following example shows the infeasible linear empirical model; quadratic and cubic models were also found to be infeasible in various configurations. As discussed, the system trajectory appears to be unbounded. Further work is required to examine why this decoupling method lead to such aphysical models.

$$\dot{\tilde{\mathbf{x}}} = -6.815\tilde{\mathbf{x}} - 3.198\mathbf{x}_5 + 3.796\mathbf{x}_{13} + 0.691\mathbf{u}_0 - 0.654\mathbf{u}_1 \tag{C.1a}$$

$$\dot{\mathbf{x}_5} = 46.374\tilde{\mathbf{x}} + 7.060\mathbf{x}_5 + 28.011\mathbf{x}_{13} - 0.761\mathbf{u}_0 - 34.936\mathbf{u}_1 \tag{C.1b}$$

$$\dot{\mathbf{x}_{13}} = 12.406\tilde{\mathbf{x}} + 5.897\mathbf{x}_5 - 3.365\mathbf{x}_{13} - 3.547\mathbf{u}_0 - 2.327\mathbf{u}_1, \tag{C.1c}$$

where $\tilde{\mathbf{x}}$ contains the reduced FOM variables $\mathbf{x}_0 \dots \mathbf{x}_4, \mathbf{x}_6 \dots \mathbf{x}_{12}, \mathbf{x}_{14} \dots \mathbf{x}_{19}$. $\mathbf{x}_5$ and $\mathbf{x}_{13}$ are unreduced. All variables were normalised before being passed to SINDYc. The error metrics for this model are: $R2 = 0.969$, $MSE = 0.0282$, $MAE = 0.0724$. These are comparable to the unconstrained linear model tested in Section 4.2.2, which was found to work.

Appendix D

Workaround to compute decoder forward pass within PYOMO

This section documents the workaround to implement the decoder forward pass within PYOMO to make constraint satisfaction possible in the SINDYC-ROM. This was discussed in Section 4.1.4.2.

Here, the weights and biases of each of the 4 decoder layers were extracted in PYTORCH, then loaded into the PYOMO ROM. These are the `W` and `B` variables in Figure D.1. The forward pass is then computed by successive matrix multiplications, along with the activation function transformations to elements in the matrices. The matrix multiplications and `tanh` functions are not evaluations but rather symbolic PYOMO expressions. Likewise, the final result obtained, `x_hat[5]`, is not a numerical variable but a complex symbolic PYOMO expression object, containing symbolic elements from all the prior functions; its values will only be numerically populated when the model is solved by PYOMO. This adds significant complexity to the ROM, which partly negates the computational efficiency gained by model reduction.

```python
# Constraint for x5 <= 313 K
def _constraint_x5(m, _t):
    # Array of Pyomo model variables
    x_hat = np.array([m.x0[_t], m.x1[_t], m.x2[_t]]).reshape(1, 3)

    # Forward pass
    x_hat = x_hat @ W[0] + B[0]
    for row in range(x_hat.shape[0]):
        for col in range(x_hat.shape[1]):
            x_hat[row, col] = tanh(x_hat[row, col])

    x_hat = x_hat @ W[1] + B[1]
    for row in range(x_hat.shape[0]):
        for col in range(x_hat.shape[1]):
            x_hat[row, col] = tanh(x_hat[row, col])

    x_hat = x_hat @ W[2] + B[2]
    for row in range(x_hat.shape[0]):
        for col in range(x_hat.shape[1]):
            x_hat[row, col] = tanh(x_hat[row, col])

    x_hat = x_hat @ W[3] + B[3]

    x_hat = np.array(x_hat).flatten().reshape(1, 20)
    x_hat = self.autoencoder.scaler_x.inverse_transform(x_hat)
    x_hat = x_hat.flatten()
    return x_hat[5] <= 313
```

Figure D.1: Workaround to compute decoder forward pass within PYOMO