UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Faculty for Computer Science, Electrical Engineering and Mathematics
Department of Computer Science
Research Group Data Science for Engineering

Master's Thesis

Submitted to the Data Science for Engineering Research Group
in Partial Fulfilment of the Requirements for the Degree of

Master of Science

Development of a Reinforcement Learning Interface for Partial Differential Equation Control

by
AAKASH KHATRI

Thesis Supervisor:
Jun.-Prof. Dr. Sebastian Peitz

Paderborn, November 13, 2022

Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen worden ist. Alle Ausführungen, die wörtlich oder sinngemäß übernommen worden sind, sind als solche gekennzeichnet.

Paderborn, 13th Nov, 22.

Ort, Datum

Unterschrift

Abstract. Partial differential equations are essential to many physics engineering problems, and the systems governed by such equations are usually complex and dynamic. Controlling such systems using conventional control methods is difficult because of the partial availability of a system model and the high dimensionality involved both in the state and control spaces. Machine learning techniques such as reinforcement learning, which work well with high-dimensional control problems and can also be data-driven, have been extensively studied to control such complex dynamic systems. For further development of such techniques, an interface is needed, which makes it easy to develop and test different control concepts and compare them with state of art. Therefore PhysicsGym, a reinforcement learning interface for control optimization of systems governed by partial differential equations, is proposed. It can use reinforcement learning methodology and help discover novel control strategies for different control problems. PhysicsGym is a python-based interface that integrates Phiflow's partial differential solver interface and the standardized, easy-to-use interface of reinforcement learning by OpenAI Gym. This interface provides easy controller integration and testing. We demonstrate the effectiveness of our interface by two simple control problems with high dimensional state space and control space—one involving a system governed by the Burgers equation and the other with the Heat equation.

Contents

1 Introduction

Complex systems consist of many parts that interact with each other. Examples include neurons in the human brain, computers communicating on the internet, the earth's global climate, transportation or communication systems, and the entire universe. Control is essential in complex technical systems, e.g., autonomously ensuring a drone's stability in heavy wind conditions while scanning a windmill for maintenance. Thus, optimally controlling these complex systems will benefit many industries worldwide.

Partial differential equation (PDE) control is a field of control theory that focuses on controlling dynamic systems governed by a set of partial differential equations. Generally, in a linear or non-linear control problem, the system known as the plant is described by an ordinary differential equation whose state depends on a single independent variable. On the other hand, in a PDE control problem, the plant is a partial differential equation where the state is a combination of more than one independent variable. PDE control holds applications in various engineering disciplines, including space engineering, traffic flow control, control of temperature, fluid mechanics, vibration control and many more. In space engineering, one application of PDE control is to design robots with flexible components modelled by beam equations given by Euler-Bernoulli beam theory Bauchau and Craig [2009]. In the field of traffic flow control, the dynamics of multiple vehicles on the road can be accurately modelled by using partial differential equations, assisting in reducing traffic jams Liard et al. [2020] e.g. using speed limit as control actuators. PDE control also has applications in controlling the room's temperature with the dynamic system described by the Heat equation Widder [1976], and the control actuator could be an HVAC unit Pan et al. [2018a]. In fluid mechanics, PDE control has many applications for active flow control, which requires the actuators to interact with the flow resulting in the applications like reducing the drag, the lift generated in a turbulent flow Rabault et al. [2019] and shape optimisation Mohammadi and Pironneau [2004].

There are different types of controllers for controlling a system in general. The most common type is on-off controllers, aka. Two-position controllers because the controller's output varies only between two values (e.g. maximum and minimum). Another simple controller is Proportional-Integral-Derivative (PID) controller, which gives continuous output values unlike the previous one. Quoting the authors in the book, "Reinforcement learning and Optimal control" [Bertsekas, 2019], "PID control aims to maintain the output of a single-input single-output dynamic system around a set point or to follow a given trajectory, as the system parameters change within a relatively broad range". Considering the high number of parameters in a complex system, these controllers are too simple to control these complex systems optimally. The methodologies of feedback control can control the dynamic systems governed by partial differ-

ential equations. Feedback control, in general, is a strategy which involves controlling a system by analysing its response to the control input. The system response, termed the feedback, is a difference between the current state of the system and the reference state we are trying to reach. There are many methods of feedback control which are discussed later in the chapter 4, but the two main classes of methods that we will focus on in this thesis are Model predictive control (MPC) and reinforcement learning (RL).

MPC is an optimisation strategy representing state-of-the-art for real-time optimal control. MPC requires constructing a model of the problem at hand and iteratively refining it until a sufficiently accurate model is achieved. It uses a system model to predict the behaviour of the system by solving a constrained optimisation problem in an online manner Schwenzer et al. [2021]. It solves this optimisation problem for a fixed number of steps, known as the prediction horizon. MPC has been widely popular in the control industry for decades. The paper by Qin and Badgwell [2003] provides a survey of the state-of-the-art in industrial MPC. There are also applications of MPC in PDE control. The thesis by Pirkelmann [2020] discusses the application of non-linear MPC to control a system governed by the Heat equation Widder [1976]. Another paper Dietze and Grepl [2021] discusses control of parabolic partial differential equations along with the reduced basis method Ohlberger and Rave [2015] for the model order reduction.

Although MPC is state of the art, it does solve the optimisation problem at each time step which becomes computationally expensive and also, there is no learning involved in MPC. On the other hand, offline-trained learning-based methods require comparatively less computation time during runtime. Reinforcement learning (RL) is a learning-based method for optimal decision-making and control Bertsekas [2019]. It is another feedback control technique that uses rewards from the system to learn the quality of actions taken to influence the environment. In reinforcement learning, an agent takes action based on observations of the environment's state and receives a reward. Reinforcement learning maximises cumulative discounted reward by learning an optimal state-action mapping policy through trial and error [Bertsekas, 2019]. Recently it has gained widespread popularity in controlling Atari games Mnih et al. [2013], the game of Go Silver et al. [2016], and autonomous control of systems Kiumarsi et al. [2018]. There is also some work in partial differential control using reinforcement learning. The paper Pan et al. [2018a] showcases the control of a dynamic system to a heat ventilation and control unit problem using DDPG, an actor-critic RL algorithm. Another paper, Rabault et al. [2019] shows the application of RL to find optimal control strategies for active flow control.

This thesis aims to bridge the gap between complex physics simulations and various black-box control schemes using an interface. We have developed a reinforcement learning interface named PhysicsGym for controlling partial differential equations for stabilisation and trajectory matching problems. This interface will allow researchers to develop and test various partial differential equation control problems effortlessly using reinforcement learning. We have used Phiflow Holl et al. [2020] as the partial differential equation solver, and OpenAI Gym Brockman et al. [2016b] as the reinforcement learning framework. To show how the interface works and how to use it, we have developed two experiments involving the control of the Heat equation Widder [1976] and the Burgers equation Lewis and Nualart [2017] in chapter 8. Both these experiments solve a control problem of reaching a reference state and we compare the reinforcement learning control strategies with that of a baseline agent and a model predictive control controller. The inspiration for the base design of our interface comes from the control problem of *Controlling Burgers equation* in Thuerey et al. [2021], which uses reinforcement learning to solve an inverse problem using the Burgers equation Fatimah et al. [2015].

In Chapter 2 we will discuss related work in the field of RL and MPC for PDE control and the inspirations for our interface design. Chapters 3 to 6 serve as the foundational chapters covering necessary explanations for understanding this thesis's relevant topics. In Chapter 3 we

will discuss the basics of partial differential equations with examples, including the concepts of boundary conditions and how to solve a PDE. We will briefly discuss two well-known methods of solving PDEs, namely the finite difference method and finite element method, followed by a visualisation of the behaviour of the Heat equation and the Burgers equation. In chapter 4 we will discuss different feedback control strategies, starting from simple controllers like PID controllers to adaptive strategies like MPC and finally, the learning-based method of reinforcement learning. This chapter will discuss the basic structure of these methodologies and how they relate to feedback control. We will later discuss RL and MPC in more detail in the following chapters. In chapter 5 we will explain the working of MPC along with its main components in detail. We will also discuss how to solve an optimisation problem using MPC, followed by framing a diffusion equation and the Heat equation as an MPC control problem. In chapter 6 we will do precisely as in the previous chapter but for reinforcement learning. We also briefly compare RL and MPC methodologies in terms of approach, terminology, model usage and computational efforts.

In chapter 7, we will introduce our PhysicsGym interface, discussing the design and how to use the interface in detail. In chapter 8, we will use our interface and define two control problems, one with the Heat equation and the other with the Burgers equation. In both of these experiments, we will define the problem and then showcase the behaviour of PDE with an uncontrolled simulation. Then we will use three different agents, namely, the baseline agent, the MPC agent and the RL agent, to solve the control problem and later evaluate the results by comparing them.

2

Related work

In this section, we will compare different research works in solving common physical control problems using Reinforcement learning (RL) and Model predictive control (MPC) and motivate the need for an RL interface as a benchmark. In the first four paragraphs, we will discuss papers related to RL and MPC, along with their advantages and limitations. Then we discuss studies comparing these two methods for two scenarios: adaptive cruise control and controlling the diffusion of heat. The main reason behind this comparison is that both RL and MPC methods have advantages and disadvantages based on the problem. However, there does not exist any interface that allows for easy implementation and comparison of these methods for partial differential equation problems. This is where our thesis comes in. Later on, we also talk about a similar toolbox named "OpenModelica Mocrogrid Gym" [Heid et al., 2020] that does something similar to ours but for microgrid problems.

A 2019 paper [Rabault and Kuhnle, 2019], by authors Rabault, Kuchta, Jensen, Reglade and Cerardi, showcases the first application of Artificial Neural Networks trained through deep Reinforcement learning for drag and lift reduction contributing to the discovery of optimal control strategies in active flow control. In this paper, the agent learns an active control strategy from experimenting with the mass flow rates of two jets on the sides of a cylinder resulting in the stabilised vortex and drag reduction by about 8%. They use FEniCS Logg et al. [2011] for simulations and Tensorforce [Kuhnle et al., 2017] for defining and executing the agent. The environment is first initialised and then executed, where the agent evolves for the given number of iterations by interacting with the environment and obtains a reward, e.g. in the form of an average drag value for that evolution. They use a class of RL algorithms called "Proximal Policy Optimization" (PPO). The PPO agent further learns to maximise this drag reduction reward over time, which happens internally in Tensorforce. Another paper by them, [Tang et al., 2020] uses RL to discover active flow control strategies using a range of Reynolds numbers (the Reynolds number is the ratio of fluid momentum force to viscous shear force [Rehm et al., 2008]). As a result, the controller achieves a drag reduction of up to 38% for a Reynolds number of 400. It also uses the PPO algorithm with a new smoothing interpolation function that effectively suppresses the lift's problematic jumps. Here we witness the advantage of the RL interface in Tensorforce, resulting in the minimalistic implementation of the RL framework, but their implementation is specific to FEniCS. There is no easy way to incorporate other control problems within this implementation.

Another paper [Wei et al., 2019] about solving nonlinear differential equations introduces a rule-based self-learning approach using Reinforcement learning to find general solutions for nonlinear differential equations like Schrödinger's equation [Chaichenets et al., 2018], Burgers

equation [Lewis and Nualart, 2017], Navier Stokes equation [Gallavotti, 2019] and few more. The RL solutions here are efficient and provide consistent results compared to discrete physics solutions, thereby proving the efficiency of RL in controlling systems governed by certain nonlinear differential equations. Although the work by these authors provides a solid groundwork for RL in flow control, they do not provide portability to other problems, e.g. controlling heat diffusion in a complex system which requires further abstraction in both RL and FEniCS implementation to bridge this gap.

The paper Farahmand et al. [2016] shows the application of Reinforcement learning for controlling the room temperature in the presence of an external source. The problem discussed here is named "The Heat Invader problem", where the Heat equation describes the dynamic system and the control actuator is boundary control and fans in the room. They show how to form a continuous PDE control problem with high dimensional state spaces as an RL problem by expressing the components of the PDE control problem as a Markov Decision Process. However, this work is limited to finite-dimensional action spaces. Another paper Pan et al. [2018b] solves this problem but for high dimensional action spaces by using the concept of "action descriptors", which exploit the spatial regularities among the action dimensions to control high dimensional PDEs. Both these papers used the DDPG algorithm, and the latter shows successful results and comparisons for action spaces of different sizes. Although their work provides successful results in applying RL to the problem, their code was private during the time of writing this thesis, so it was not possible to review and extend this directly in the implementation of this thesis. However, the theoretical concepts described in these papers served as the foundation for expressing the PDE control problem as an RL problem.

The 1993 PhD study [Gullapalli, 1992] motivates using RL for control problems. Training controller methods fall under two classes, direct and indirect methods. This study shows that using model predictive methods, the complexity of model construction can often exceed that of solving the original control problem using direct reinforcement learning methods rendering such indirect methods relatively inefficient. This further motivates the use of direct reinforcement learning methods for learning to solve control problems. They also present techniques for augmenting the power of reinforcement learning methods, including using local models, implementing a procedure from experimental psychology called "shaping" to improve learning efficiency and implementing a multi-level architecture for knowledge extraction from previous iterations.

The thesis by Simon Pirkelmann [Rawlings et al., 2012] shows the usage of Model predictive control for time-varying systems in an energy-efficient way. They do so by solving the convection-diffusion equation [Wikipedia contributors, 2022] within a time-variant system by formulating the problem as a constraint optimisation problem. According to the author, learning a closed-loop model appears to be a good choice because the problem is challenging to solve numerically for large or infinite time horizons and with less availability of weather forecast data. They conclude that their results do not prove the convergence of state and control trajectories. Furthermore, they also use FEniCS for solving partial differential equations.

Another study [Lin et al., 2021] compares RL and MPC methods for adaptive cruise control design in car-following scenarios. It uses "Deep Deterministic Policy Gradient" (DDPG) [Silver et al., 2014] for RL and "Interior Point Optimization" [Nemirovski and Todd, 2008] for MPC problem. The results showcase that when there are modelling errors due to control delays and disturbances, the RL-trained policy performs better with significant modelling errors while having similar performance as MPC when the modelling errors are minor. These studies showcase an excellent comparison of RL and MPC methods and even show RL outperforming MPC under certain constraints. An abstract interface is required to incorporate new RL algorithms and compare them with MPC methods. This interface should allow defining different types of

partial differential equation problems in an easy-to-use PDE solver library, Phiflow, and then solving them using any black box methods like RL or MPC. Next, we will look OpenModelica Microgrid Gym [Heid et al., 2020], a framework that also helped develop a good understanding of how to develop an RL interface for a control problem.

OpenModelica Microgrid Gym (OMG) is an open-source toolbox that provides a simulation framework for local energy grids based on power electronic converters [Heid et al., 2020]. It provides a graphical user interface from OpenModelica [Fritzson, 2015] that allows setting up a problem in the form of arbitrary electric grid designs and then performing simulations on closed control loop problems. It also provides a standardised OpenAI Gym interface to experiment with data-driven reinforcement learning algorithms [Brockman et al., 2016a]. OMG is python-based and uses Stable Baselines3 [Raffin et al., 2021], TF-Agents [Abadi et al., 2015] and Keras-RL [Chollet et al., 2015] for implementing the interface.

The main inspiration for our interface comes from a simple PDE control problem of solving an inverse problem using the Burgers equation in the book "Physics-based deep learning" Thuerey et al. [2021]. The control problem discussed here served as the foundation for our PhysicsGym interface. They have used Phiflow Holl et al. [2020] as the PDE solver and Stable-baselines3 for defining the RL environment; however, their version only works with an older version of Phiflow (1.5). Some significant architectural changes in the later versions of Phiflow (>2.0) are incompatible with their work. Our interface works with the newer version of Phiflow mentioned here. We designed our interface by taking different PDE solvers into account such that the extension to another PDE solver library is possible. We chose OpenAI Gym's environment interface, which is state of the art for defining RL environments and used agents defined in Stable baselines3 for training and execution. Also, the stable baselines library provides direct support to OpenAI Gym environments making it simple to test with different algorithms. We have added additional interface methods to make control inputs more realistic by applying the control only at a particular part of the domain and more flexible visualisations with debugging.

3

Partial Differential Equations

3.1 Introduction

Differential equations are mathematical tools that help formulate the laws of physics governing simple and complex dynamical systems. They are used to model various physical phenomena like fluid flow, propagation of sound waves, dissipation of heat and many more. For example, the flow of water in a pipe Pandit et al. [2009] or the airflow around the wings of an aeroplane can be described by the Navier-Stokes equation Doering and Gibbon [1995] and the transfer of heat between any two objects at different temperatures can be modelled by the Heat equation Widder [1976]. Generally speaking, these equations combine one or more functions and their derivatives. The functions generally represent a physical quantity like the velocity of a particle or temperature, and the function's derivative represents its rate of change. Consider a function z made up of a single variable x, $z = f(x)$, where x is also called the independent variable Gould et al. [1988] and z the dependent variable as its value depends on x. The derivative of this function represents the rate of change of z w.r.t x and is represented by the Leibniz notationStewart [2012] dz/dx. When the dependent variable is dependent on more than one variable, e.g. x and y, its partial derivative w.r.t. x is represented by notation $\partial z/\partial x$.

One of the simplest forms of differential equations is ordinary differential equations, ODE in short. ODEs consist of 1) a single independent variable and 2) an unknown function along with its 3) derivatives. They are used to model physical phenomena like the movement of electricity, the to-and-fro motion of a pendulum, and various concepts related to the exchange of energy in a simple system. The equation z = f(x) is an ordinary differential equation as it involves a single independent variable and, if it is possible to calculate the derivatives of its function. Another simple ODE is the wave propagation equation, given by $d^2y/dt^2 + \omega^2 y = 0$; here, the constant ω represents the frequency of oscillation of wave particles. A solution to this ODE can be given by $y(t) = Asin(wt + \phi)$ where A is the amplitude of the wave Chapra and Canale [2005], and ϕ is the phase. This ODE represents a simple harmonic motion of a particle exhibiting periodic behaviour in time Messaoudi and Talahmeh [2017].

Although ODEs are useful for modelling many simple physical phenomena, more is needed to represent a physical system with only a single variable regarding complex systems. Partial differential equations work better for this purpose. A partial differential equation (PDE) consists of functions dependent on one or more independent variables. The order of a partial differential equation represents the highest derivative of the unknown function. Consider a simple linear

parabolic partial differential equation known as the Heat equation,

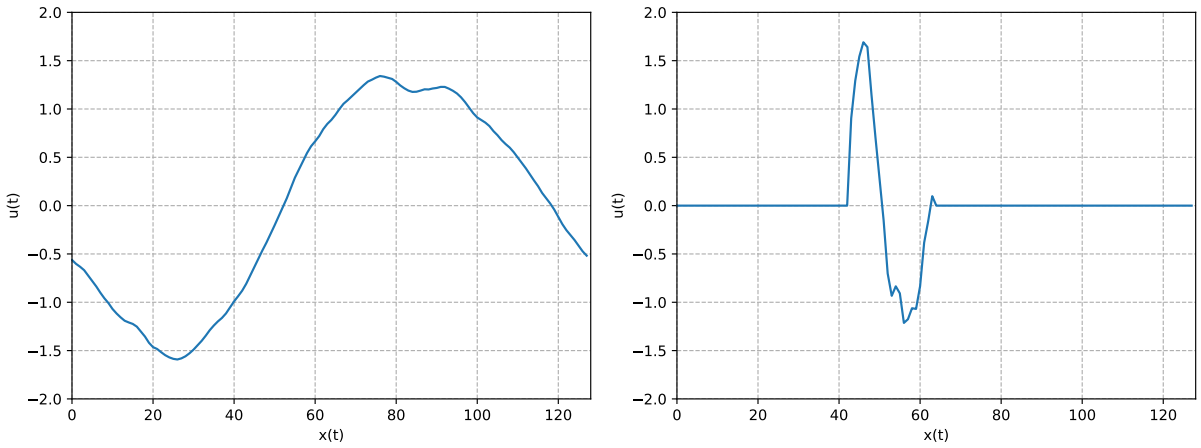
$$\frac{\partial T(x,y)}{\partial t} = \alpha \cdot \frac{\partial^2 T(x,t)}{\partial x^2} \tag{3.1}$$

here α is the diffusivity constant, $\frac{\partial T}{\partial t}$ represents the rate of change of temperature(T) w.r.t. time and $\frac{\partial^2 T}{\partial x^2}$ represents the second order derivative. So, this equation calculates the temperature change rate at each point in space at that particular time. Another common PDE is a convection-diffusion equation,

$$\frac{\partial u(x,t)}{\partial t} = D \cdot \frac{\partial^2 u(x,t)}{\partial x^2} - u(x,t) \cdot \frac{\partial u(x,t)}{\partial x}$$

Here, u is the dependent variable representing the velocity of fluid or temperature in a heat transfer system, and D is the diffusivity constant, e.g. diffusivity of a particle in motion, thermal diffusivity or viscosity in terms of fluid flow.

3.2 Boundary conditions



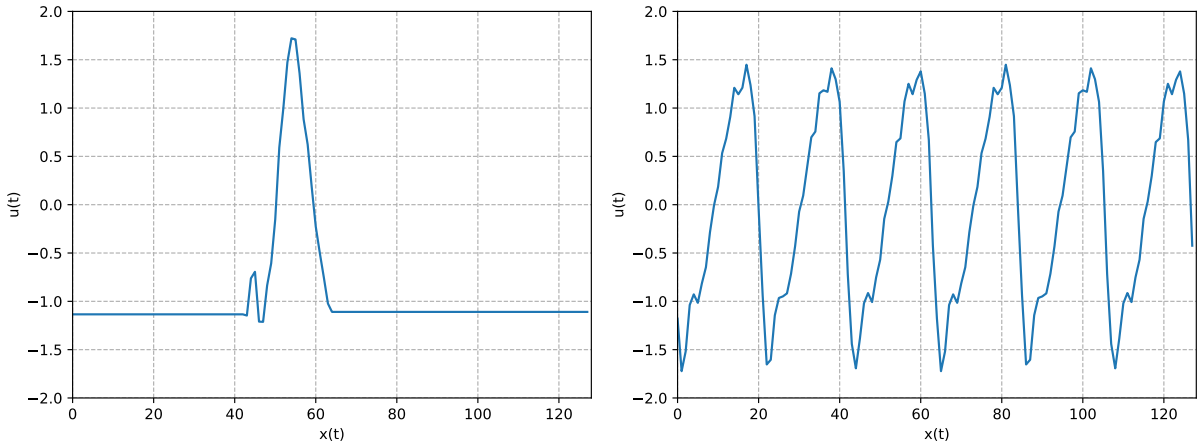
(a) initial state

(b) Zero extrapolation (Dirichlet)

Figure 3.1: Dirichlet boundary conditions.

Differential equations can have infinitely many solutions. For example, consider the equation of curve $dy/dx = x^2$, with a general solution of $y = x^3/3 + c$, where c is some arbitrary constant. Here, infinite solutions exist because c can be any integer. To get a unique solution, external constraints must be imposed, which involves specifying bounds to the solutions domain. For the above example, let us say we only consider the curve that passes through the point $(x, y) = (0, 1)$, then the only solution that satisfies this condition is with $c = 1$, $y = \frac{x^3}{3} + 1$. There are many different types of problems based on such differential equations, viz. initial value problems Brenan et al. [1995], boundary value problems and more. Initial value problems require specifying the initial value of the independent variable, and then the value of this variable is traced forward in time based on the equation.

On the other hand, boundary value problems also require specifications for the values at the domain's boundaries. In this thesis, we will focus on boundary value problems. The different types of boundary conditions typically encountered in the solution of partial differential equations are 1) Dirichlet boundary conditions Bazilevs and Hughes [2007], 2) Neumann boundary



(a) Boundary extrapolation (Neumann)

(b) Periodic boundary condition

Figure 3.2: Neumann and Cyclic boundary conditions

conditions Ng et al. [1999], 3) Robin boundary conditions Arendt and Warma [2003], 4) periodic boundary conditions and many more. Let us see visualise these boundary conditions.

Figure 3.1 (a) shows the initial state of a simple dynamic system. Starting with the Dirichlet boundary conditions, a.k.a. fixed boundary conditions, they extend the domain's boundary with constant values. Figure 3.1 (b) shows the Dirichlet boundary conditions where we sample the state from the initial state between domain values $x = 40$ to $x = 60$. For the rest of the values, we apply a constant extrapolation of 0. The following boundary condition is the Neumann boundary condition which extends the domain with the values at the boundary. Figure 3.2 (a) shows this scenario. Here we sample the values between $x = 40$ and $x = 62$ from the initial state and then extrapolate the rest of the values from the boundary values, slightly below $u = -1.0$. The final constraint is the periodic or cyclic boundary condition which copies all the values in the domain forward in a repeating manner, as shown in figure 3.2 (b). This condition requires the connection between any two sides of the domain for forward propagation.

3.3 Solution methods

The solution methods for partial differential equations generally fall under two main categories: analytical and numerical. Analytical methods involve calculating the exact solution given some initial and boundary conditions, while numerical methods calculate an average of finite approximations. The most common analytical method is *Separation of variables* that involves representing a partial differential equation as a product of multiple ordinary differential equations. It involves rearranging independent variables such that each type of variable is on one side of the equation. For example, consider the equation

$$\frac{\mathrm{d}y}{\mathrm{d}x} = \frac{x^2}{y^2} \rightarrow y^2 \cdot dy = x^2 \cdot dx$$

Integrating this equation gives the solution $y^3/3 = x^3/3 + c$. Let's say the initial condition is $(x, y) = (0, 1)$, substituting this we get $c = 1$, so the solution for initial condition $(x, y) = (0, 1)$ is $y^3 = x^3 + 1$.

Although analytical methods provide the best solution to the problem, it is only sometimes possible to find an analytical solution that satisfies all the constraints. The reason is that an

analytical solution requires complete knowledge of how the problem works, which is impossible for many complex problems. For complex problems, numerical methods are more feasible. Roughly speaking, numerical methods involve framing a problem using the trial and error approach across a set of candidate solutions. It achieves satisfactory results, mainly because we are often concerned with an approximately good solution rather than the exact one. A few well-known numerical methods for solving a PDE are the finite element method, finite difference method, finite volume method and more. All these methods use discretisation to approximate the solution to PDE, which involves evolving a discrete set of values and taking small steps in time to approximate the continuous function of space and time Chapra and Canale [2005]. In this thesis, we have used Phiflow library Holl et al. [2020] as the partial differential equation solver, which uses the finite difference method. We will discuss the finite difference method next, followed by the finite element method.

3.3.1 Finite Difference Method

One way to solve a continuous PDE is using the Finite difference method, dividing the domain into smaller equal parts and then calculating the value at each part using the central difference formula Kong et al. [2021]. For example, consider an ordinary differential equation of the form:

$$t'(x) = a \cdot t(x) + b \tag{3.2}$$

The Euler's method Biswas et al. [2013] for solving this equation is given by

$$\frac{t(x+h) - t(x)}{h} \approx t'(x)$$

where $t(x+h)$ is the finite difference term. Substituting the value of $t'(x)$ from 3.2, we get,

$$\begin{aligned} \frac{t(x+h) - t(x)}{h} &= a \cdot t(x) + b \\ \therefore t(x+h) - t(x) &= (a \cdot t(x) + b) \cdot h \\ &= ah \cdot t(x) + bh \\ \therefore t(x+h) &= t(x) + (a \cdot t(x) + b) \cdot h \end{aligned}$$

The above equation constitutes the finite difference equation for this ODE, and solving this equation, gives an approximate solution to this equation. Chapter 20 of the book "Numerical methods for Engineers" Chapra and Canale [2005] explains the finite difference method in detail. Here we will briefly explain this method using an example from the book for the Laplace equation. For more equations like other elliptic or parabolic equations, please refer to chapters 29 and 30 of the book.

To understand how the finite difference method works in more detail, we consider the solution of a simple Laplace equation that models various problems involving the potential of an unknown variable as described in Chapra and Canale [2005]. The second-order linear Laplace equation is given by,

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0 \tag{3.3}$$

Consider a square grid as the domain, Ω_{ij} where i,j represents points in the domain's x and y direction, respectively. The central differences are given by

$$\frac{\partial^2 T}{\partial x^2} = \frac{T_{i+1,j} - 2T_{ij} + T_{i-1,j}}{\Delta \cdot x^2}$$
$$\frac{\partial^2 T}{\partial y^2} = \frac{T_{i,j+1} - 2T_{ij} + T_{i,j-1}}{\Delta \cdot y^2}$$

Here $T_{i+1,j}$ and $T_{i-1,j}$ are the neighbours of $T_{i,j}$ on the right and left side, respectively. Similarly, $T_{i,j+1}$ and $T_{i,j-1}$ are the neighbours of $T_{i,j}$ on the top and bottom side respectively. Since the grid is square, $\Delta x = \Delta y$, the Laplace difference equation at 3.3 becomes,

$$T_{i+1} + T_{i-1,j} + T_{i,j+1} + T_{i,j-1} - 4T_{i,j} = 0$$

The boundary conditions need to be specified to obtain the exact solution of any PDE. Consider the Dirichlet boundary conditions, which means setting each boundary of the grid to some constant value, say $t_{0,1} = 75$, $t_{1,0} = 0$. Now solving for $i = 1, j = 1$, we get,

$$T_{2,1} + T_{0,1} + T_{1,2} + T_{1,0} - 4T_{1,1} = 0$$

Substituting the boundary conditions we get,

$$-4T_{1,1} + T_{1,2} + T_{2,1} = -75$$

Similarly, we can calculate the rest of the equations for other values of i and j in the grid. Solving these equations will give the approximate solution using the finite difference method. Also, since the grid is quite small, it is possible to calculate the equations by hand, but in practice, "The Liebmann method" Frankel [1950] is used for the calculation of complex grids. (refer to chapter 29 of book Chapra and Canale [2005] for more detail).

3.3.2 Finite Element Method

The finite difference method introduced before does not work well for irregular geometry, complex boundary conditions or heterogeneous compositions. For this, the alternative is the finite element method which begins by dividing the solution domain into elements which are simpler in shape and then finding an approximate solution for each element function and thereby assembling them into a unified solution Chapra and Canale [2005]. As in the previous method, we provide a brief description of the steps involved in solving a PDE with the finite element method, as discussed in the book. For a more detailed explanation, refer to chapter 31 of the book "Numerical methods for engineers" Chapra and Canale [2005].

The general implementation of the finite element method consists of 5 main steps: i) discretisation of the domain, ii) constructing element equations, iii) assembling them, iv) applying boundary conditions, and v) final solution. The discretisation step involves dividing the domain into a finite number of elements Chapra and Canale [2005] e.g. For a one-dimensional domain, a line, the division can be done by dividing the line using a finite number of points on the line. For a two-dimensional domain, one can use triangular or quadrilateral elements and a plane of a hexahedron shape for a three-dimensional domain. The second step is to develop differential equations for each discretised element. It consists of two steps. The first step involves choosing approximate functions, a.k.a. basis functions, which approximate the individual elements. For example, consider the following equation of line Chapra and Canale [2005],

$$u(x) = a_0 + a_1 \cdot x$$

where $u(x)$ is the dependent variable, a_0 and a_1 are the constants and x is the independent variable. Here, the dependent variable can be written as a linear combination of approximate functions as follows,

$$u = N_1 u_1 + N_2 u_2$$

where u_1 and u_2 are the approximate functions with, $u_1 = a_0 + a_1 x_1$, $u_2 = a_0 + a_1 x_2$ and $N_1 = \frac{x_2 - x}{x_2 - x_1}$, $N_2 = \frac{x - x_1}{x_2 - x_1}$ The next step is to find the correct value for coefficients that can help solve the problem optimally, which involves methods like the variational approach Beltzer [1990] and the method of weighing residuals Fletcher [1988]. Following the development of individual element functions, the next step is assembling them, combining all the individual element equations into one unified equation representing the entire system. Then, the unified system is constrained with the boundary conditions before finally solving the system using methods like LU decomposition Padua [2011], Chapra and Canale [2005].

3.4 PDE Examples

This section will discuss some well-known partial differential equations, explain their properties, and explain some relevant control problems.

3.4.1 The Burgers equation

Burgers equation is one of the simplest PDEs describing the velocity of a fluid with respect to time and is given by,

$$\frac{\mathrm{d}u(x,t)}{\mathrm{d}t} = -u(x,t) \cdot \frac{\mathrm{d}u(x,t)}{\mathrm{d}t} + v \cdot \frac{\mathrm{d}^2 u(x,t)}{\mathrm{d}t^2} \tag{3.4}$$

where u is the velocity field, and v is the viscosity of the fluid. It is also called a convection-diffusion equation because the quantities $u(x,t) \cdot \frac{\mathrm{d}u(x,t)}{\mathrm{d}t}$ and $\frac{\mathrm{d}^2 u(x,t)}{\mathrm{d}t^2}$ represent the convection term and diffusion term respectively. When the viscosity is 0, this equation simplifies to the inviscous Burgers equation, which is a prototype for equations that develop shock waves Fatimah et al. [2015] given by,

$$\frac{\mathrm{d}u(x,t)}{\mathrm{d}t} + u(x,t) \cdot \frac{\mathrm{d}u(x,t)}{\mathrm{d}t} = 0$$

Next, we visualise the time evolution of the Burgers equation. For this, we consider a 1-dimensional domain $\Omega \in \mathcal{R}^d$ where $d = 128$, the viscosity of 0.29 and the initial condition given by $u(x,0) = f(x)$ where f is a Gaussian function. We will use the Dirichlet condition for boundary conditions, which sets the velocity at boundaries to 0.

Figure 3.3 shows the time evolution of initial velocity for a time period of $t = 0, 1/3, 2/3, 1$. At each time step, all the positive values tend to move to the right and negative towards the left of the grid. The reason is that the function differential is proportional to u in 3.4. These two trends moving inwards from opposite direction creates a shockwave approximately in the centre of the grid. There are some control applications involving the application of an external force to this system governed by the Burgers equation to dampen shockwaves(Thuerey et al. [2021])

3.4.2 The Heat equation

The Heat equation is a simple diffusion equation that calculates the change in the temperature at each point in space for that particular time and is given by,

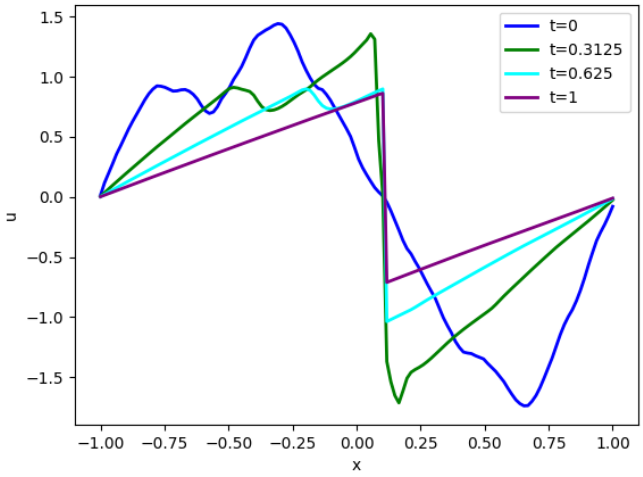


Figure 3.3: Simulation of 1-d Burgers equation

$$\frac{\partial u(x,t)}{\partial t} = D \cdot \frac{\partial^2 u(x,t)}{\partial t^2}$$

Here, u is the temperature at each point x in space at time t, D is the diffusivity constant that defines the rate at which the exchange of heat occurs in the field, and the final second-order differential is the diffusion term.

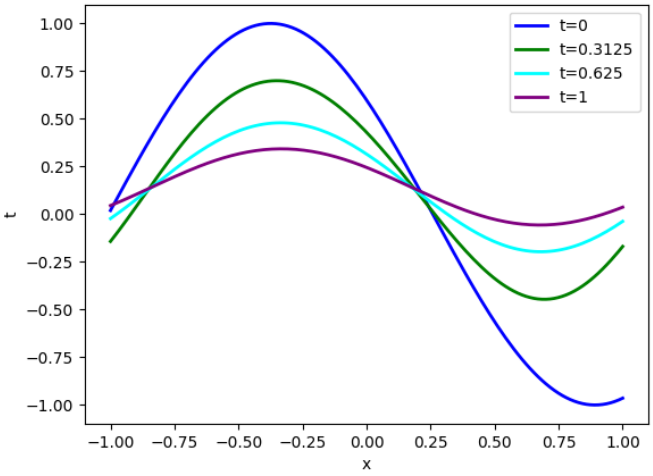


Figure 3.4: Simulation of 1-d Heat equation

Consider a simulation of heat transfer in a pre-heated rod. We assume that the rod is insulated, i.e. there is no exchange of heat between the rod and the environment from outside of the rod, and the two ends of the rod are at a constant temperature, say 0. Now we know from the law of conservation of energy that the exchange of heat occurs from hotter areas to colder areas until the temperature achieves an equilibrium state everywhere in the domain. Figure 3.4 shows this phenomenon by simulating the heat transfer inside the rod for a finite number of steps. It is evident from the line plot that the local maxima move downward towards 0 as they

are at a higher temperature than their neighbours. In contrast, the local minima move upward towards 0, stabilising near 0.

In the control problems involving the Heat equation, this equation is sometimes used in conjunction with other equations in an application. For example, suppose we are talking about controlling the temperature of a room with a fan. In that case, the airflow from a fan can be calculated by the Navier Stokes equation or the Burgers equation, and the temperature change with the Heat equation, e.g. in Pan et al. [2018a].

4

Feedback Control

In this chapter, we will introduce the basic concepts of a feedback control system compared to open-loop systems. We will then compare controllers that follow the principles of feedback control systems: a proportional integral and derivative (PID) controller based on Model predictive control and reinforcement learning.

4.1 Introduction

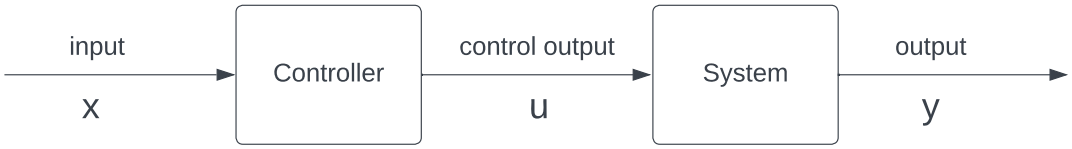


Figure 4.1: Block diagram of an Open-loop control system.

In control theory, two main control systems exist open-loop and closed-loop. An open-loop system involves only a one-way flow of control, which means that the control input is pre-decided based on the prior knowledge of the system. Then the controller is not affected based on how the system responds. Consider a simple example of a washing machine as a dynamic system. This system takes as input the information about the type of wash cycle and its time duration and comes to a halt when it reaches the end of the cycle. The machine will stop even if the clothes are not washed successfully. Figure 4.1 shows the basic block diagram of an open-loop control system. The controller takes as input some information about the system that helps decide the controller output, which is then fed to the system, and the system produces an output. Open-loop control can control a system accurately only if the entire model specification of the system is known beforehand, which is only sometimes the case for complex dynamic systems. For such a purpose, closed-loop systems are more feasible. A closed-loop system consists of sensors that can measure the system response and use it as feedback to adapt to the system's behaviour. Such systems allow the controller to modify the system to reach a goal adaptively, so they are the main go-to strategies for complex dynamic systems.

A feedback control system works on the closed-loop control principle that involves influencing the future state behaviour of a system using its output in the form of feedback. A feedback

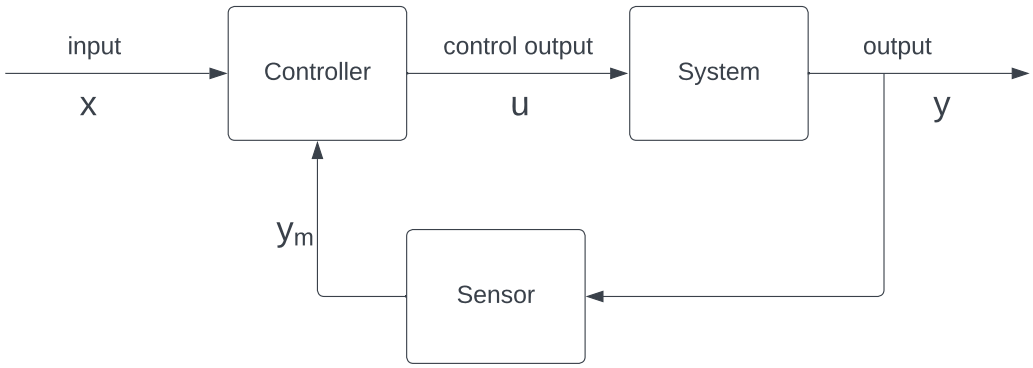


Figure 4.2: Block diagram of a Closed-loop control system.

control system comprises a controller, a plant (a system) and a sensor. Figure 4.2 shows the basic block diagram of a feedback control system. Here, the controller takes the system's current state x as input and calculates the control input u. On application of the control input, the system responds by transitioning into a new state y. The sensor measures this output, and the measurement y_m is taken into account by the controller, which then uses this information to decide better control inputs in future. For example, consider a control system that adjusts the temperature of a room based on some comfort level using a fan as a controller. Such a system will take, as input, the desired temperature value and the room's current temperature. The controller will then predict actions that minimise the difference between the two inputs and update the actions based on the feedback from the system.

In the rest of the chapter, we will discuss three different types of controllers following the principles of the feedback control system, namely, PID controllers and controllers based on MPC and RL.

4.2 PID controller

A PID controller is ubiquitous in the control industry due to its simplicity and robust performance in various operating conditions. In this type of control, the controller continuously calculates an error value $e(t)$ as the difference between the reference state $r(t)$ and a measured system output $y_m(t)$ and applies a correction based on proportional, integral, and derivative terms Paz [2001].

Fig 4.3 shows the basic block diagram of this controller. The P in PID controller stands for proportional error, $e(t) = K_p \cdot (r(t) - y_m(t))$, where K_p is a constant. Here, the more the deviation from the desired state, the more the error, hence the name. If the change is big, the error will be higher because of the constant K_p. The second term, I, denotes the integral and considers past error measurements by integrating them over time. For example, suppose there is an error after implementing proportional control. In that case, the integral term diminishes or eliminates this error based on the cumulative values of past error measurements Li [2010]. The final term, D, derivative, involves learning the future trend of error values based on how rapidly the error values change over a fixed period. Finally, the output of the PID controller is the sum of all three terms combined Li [2010],

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)\, d\tau + K_d \frac{e(t)}{dt}$$

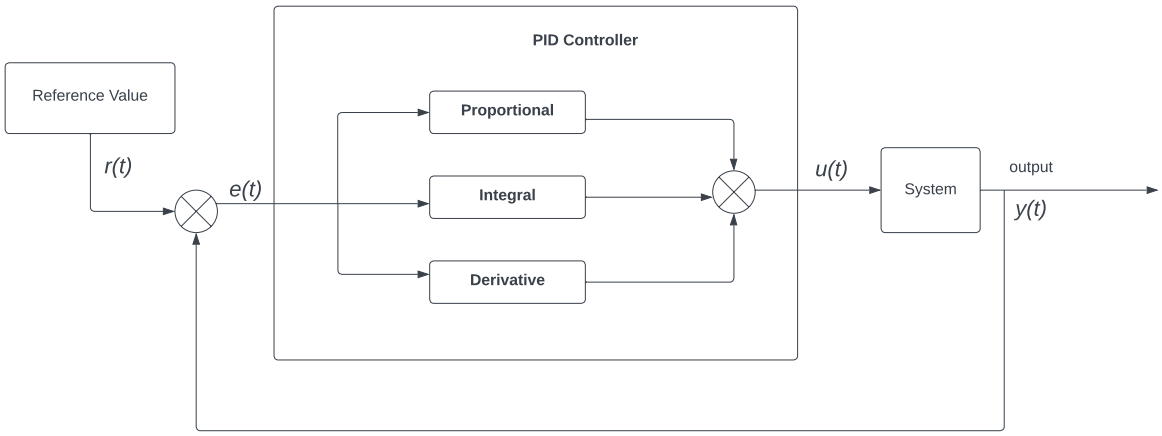


Figure 4.3: PID feedback control Li [2010]

where the non-negative, K_p,K_i, and K_d denote the coefficients for the proportional, integral, and derivative terms, respectively.

The main advantage of a PID controller is that it is simple since it only depends on the system measurements and does not require any prior knowledge of the system. Nevertheless, it is unsuitable for controlling complex systems that depend on multiple input variables. Apart from this, the PID controller cannot deal with constraints Balaji and Rajaji [2013]. The following control strategy we will discuss is the MPC controller, which overcomes these limitations of the PID controller.

4.3 Model predictive control controller

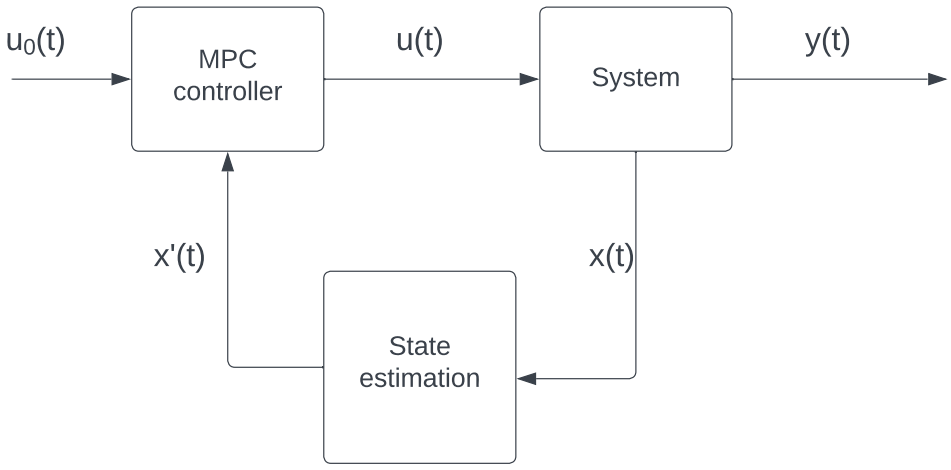


Figure 4.4: Basic block diagram of an MPC control system.

Model predictive control (MPC) is a class of algorithms that aims at learning the model of a system and using it to predict the new state of the system while iteratively improving the model based on the feedback from the system. Figure 4.4 shows the basic block diagram

of the MPC controller. PID controllers are single-input and single-output (SISO) controllers, while controllers in MPC are multiple inputs and multiple-output (MIMO) controllers. MPC is considered an iterative finite-horizon optimisation approach. At each timestep t, the MPC controller uses a sampled state to execute a control strategy for a finite time horizon $(t + \Delta t)$. At every successive time step, the controller samples the system's state again and repeats the calculations, which results in a new state and control parameters. The controller does this iteratively until optimal controller performance is achieved based on the defined objective.

The primary advantage of MPC over PID is its ability to handle constraints and MIMO systems. However, it also requires the model of a system which is only sometimes possible for many complex dynamic systems. To overcome this, we will discuss a machine learning technique, reinforcement learning, that shows promising results even when a complete model of the system is unavailable.

4.4 Reinforcement learning controller

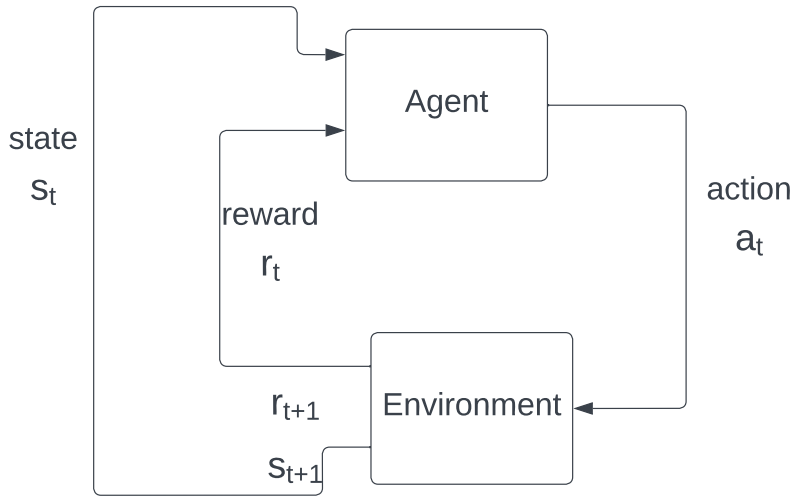


Figure 4.5: Block diagram of a Reinforcement learning control system De Castro et al. [2020]

Reinforcement learning is a class of machine learning techniques involving learning from interaction with an environment. It has gained widespread popularity because of its promising results in the field of control. The environment in reinforcement learning could be any physical description of the real world, e.g. road for self-driving cars, room for a cleaning robot, or an airfield for a self-driving drone. The learner or agent in RL interacts with the environment by performing actions chosen using a policy. In return, the environment responds with some feedback about the quality of actions in the form of a reward. The main goal of the RL agent is to maximise this reward. Figure 4.5 shows the basic block diagram of a reinforcement learning control system. Here, the agent learns a policy by interacting with the environment for several time steps. The agent then predicts the optimal action a_t to be applied to the environment to which the environment responds with a reward r_{t+1} and new updated state s_{t+1}. RL also follows the principle of feedback control, as the reward from the environment acts as a feedback to the controller (agent), which affects future predictions.

In the following three chapters, we will discuss control strategies using MPC and RL in more

detail, especially concerning the control of a PDE system, followed by a short comparison.

5

Model Predictive Control

This chapter will discuss the Model predictive control (MPC) framework as an optimal control strategy for controlling partial differential equation problems. We will begin with an explanation of the main components of MPC, followed by framing the partial differential equation control problem as an optimization problem. There onward, we will conclude with an example problem about controlling the heat equation using MPC defining how the discussed components of MPC are applied in a problem setting.

5.1 Introduction

Model predictive control is a powerful optimization strategy for feedback control that has been around since the 1980s. It involves using a system model to predict the control variables of a dynamic system by minimizing a specific cost function. One factor for its widespread popularity is its ability to handle multiple-input multiple-output systems Karamanakos et al. [2021], which the controllers like PID fail to control optimally. Another factor is its ability to incorporate constraints which involves limiting the domain of control parameters to resemble real-world behaviour Pirkelmann [2020]. For example, consider an optimal control problem of keeping an autonomous car in the desired lane. The system/plant is the environment which simulates a car, e.g. a road with cars in multiple lanes, with the control parameter being the car's speed. So, one possibility of imposing a constraint is to limit the car's speed within the road's speed limits. MPC incorporates such a constraint by binding the controller's domain with the speed limit value. Despite these benefits, MPC requires a powerful and fast processor since it solves the optimization problem online at each timestep.

Figure 5.1 shows the basic block diagram of a system using MPC as the controller. There are four main components in MPC, a system model, a cost function, the constraints and a function optimizer. A system model can be constructed by following the methodologies of system identification Ljung [1998], and the accuracy of the control strategy depends on a sound system model. The cost function is specific to the problem at hand and involves penalizing the controller for taking specific actions using the system's state and other variables as measurements. The constraints are also problem specific and are used to limit the system's behaviour within the domain to resemble real-world use cases. Finally, optimizers, in general, are used to maximize or minimize an objective function. The MPC controller solves the optimization problem of minimizing the cost function for several steps forward in time. The number of time steps that MPC uses to evaluate the cost function is given by the prediction horizon, which in other words,

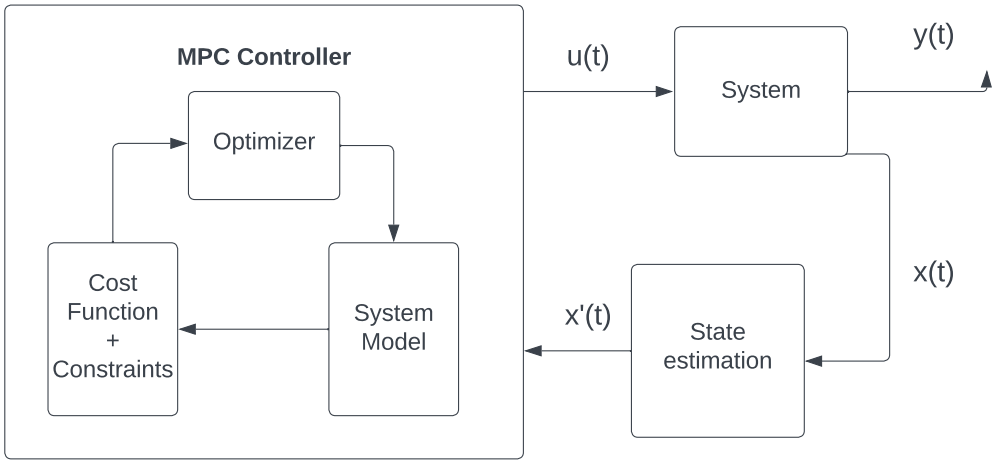


Figure 5.1: Extended block diagram of an MPC control system.

is also known as a look-ahead because it allows the controller to evaluate the predicted action at each time step by looking ahead in the future. The optimizer minimizes the cost function, which means iteratively finding the best control value for which the function has the lowest value, e.g. using methods like gradient descent. Once the control input predicted by the MPC controller is applied to the system, the plant then moves forward in time resulting in a new state. The controller then takes the new forecasted system state and repeats the entire process until the system reaches the desired behaviour.

5.2 Components of MPC

In this section, we will discuss the main components of MPC previously discussed in more detail and define them mathematically. We have derived the general idea behind this section, and the examples from the PhD thesis *Economic Model Predictive Control and Time-Varying Systems* Pirkelmann [2020]. For interested readers, we revert to Chapter 2 *Fundamentals of Model predictive control* in Pirkelmann [2020] and the book Rakovi and Levine [2018].

$$x(t+1) = f(x, u, t) \text{ for } t \in \mathbb{N} \tag{5.1}$$

Consider a dynamic system given by the function f, which is the system model in the above equation 5.1. Here $x \in \mathcal{X}$ is an independent variable representing some physical quantity, $t = 1, 2, ...$ represents the time steps and $u \in \mathbf{U}$ represents the control input at each time step. Also, $\mathcal{X} \subset \mathbb{R}^d$ is the state space, $\mathbf{U} \subset \mathbb{R}^d$ is the control space and $d = 1, 2, ...$ which can be different for state space and control space. Like any control problem, let us say the initial state is represented as $x(t = 0) = x_0$, and the controller predicts the control input u, which is applied to the system to change its state. Iteratively applying u leads to several different states known as the state trajectories defining the system model as $f : \mathcal{X} \times \mathbf{U} \to \mathcal{X}$.

A control problem for such a system can be defined as, given some initial state of the system, finding the optimal sequence of control inputs that terminates the system in a given reference state. The choice of reference state depends on the problem specification. For example, for stabilization problems, the reference state is given by the state when the system is in equilibrium, and for trajectory matching problems, the reference state is given explicitly. This problem of reaching a desired final state can be modelled as an optimal control problem which comes

from the optimal control theory where the goal is to find the control parameters such that an objective function is optimized. The objective function is also known as the cost function previously described, and it is a mapping from the state space ($\mathcal{X}$) and control space ($\mathbf{U}$) to a scalar value ($\mathbb{R}$). The cost function for this continuous time system is given as $J : \mathcal{X} \times \mathbf{U} \to \mathbb{R}$, Pirkelmann [2020]

$$J(x, u) := \sum_{t=0}^{\infty} l(x_u{}', x)$$

Here $x_u{}'$ is the updated state of the system as a result of applying the control input u. The function l is also known as the loss function whose choice differs based on the problem and should be chosen in such a way that it penalizes the controller when the controller chooses control inputs that take the system further away from the reference state. Let us take a temperature control problem as an example to understand what different choices of cost functions are possible. Consider a simple control problem where the controller's task is to control the temperature in a room by using a heating, ventilation and air conditioning unit (HVAC). Here, one possibility for defining the control space is the control parameters on the HVAC unit, which could be the cooling fan's speed or the temperature setting. The reference state can be defined based on some pre-defined value, i.e. the room temperature. For this scenario, the optimal choice for the cost function would be to calculate the average squared difference between the reference state and the system's current state. Such a cost function should successfully capture the deviation from the reference state and penalize the actions that take the system very far away from the reference state because of the squared term in the difference. However, if we also want to consider the energy costs, then we would not want our controller to fluctuate between the higher speed and lower speed too much as that would increase the electricity consumption in the unit, thereby increasing the operating costs. Such a scenario can be handled by including the magnitude of control inputs in the cost function, e.g., using the square of the control inputs applied.

The next step to make this control problem resemble real-world behaviour is to apply constraints on the type of values for states and actions. One simple constraint for this control problem could be to limit the range of actions between certain values, e.g. $u \in [-1.0, 1.0]$. Such a constraint resembles the limited control input for the speed of an air conditioning fan in the HVAC unit, as the fan's speed cannot be increased above a certain value because of physical limitations. Another constraint could be on the walls of the room, which means the walls of the room are insulated such that the temperature outside the room does not affect the temperature inside the room. To keep things simple, we will consider the constraints described in the PhD thesis Pirkelmann [2020], which are the permitted states and actions that belong to $\mathcal{X}' \subset \mathbb{R}^d$ and $\mathbf{U}' \subset \mathbb{R}^d$ respectively, resulting in the control space defined by,

$$\mathbf{U}^{\infty}(x) := \{u \in \mathbf{U}' | x_u{}'(t; x) \in \mathcal{X}' \text{ for all } t \in \mathbb{N}\}$$

Finally, we can model this continuous-time system with constraints and cost function as the following optimization problem,

$$\begin{aligned} &\arg\min_{u \in \mathbf{U}^{\infty}} J(x, u) \\ &\quad \text{s.t. } x(t+1) = f(x, u, t) \\ &\qquad\quad x(0) = x_0 \text{ for all } x \in \mathcal{X}', u \in \mathbf{U}' \text{ and } t \in \mathbb{N}. \end{aligned} \tag{5.2}$$

The control problem discussed here is a continuous time problem that the MPC controller solves by iterating over a finite number of steps given by the prediction horizon $n \leq \mathbb{N}$. It

then subsequently updates the control input by selecting only the first control input from the resulting control sequences and then repeating this for an infinite time. This approach converts the equation of the continuous time problem at 5.2 to the following 5.3 as the final optimization problem to be solved by the MPC controller at each time step. The equation 5.4 represents the finite time cost function, and the equation 5.5 represents the finite time control space as described in Pirkelmann [2020].

$$\begin{aligned} &\arg\min_{u \in \mathbf{U}^N} J'(x,u) \\ &\quad \text{s.t. } x(t+1) = f(x,u,t) \\ &\qquad\quad x(0) = x_0 \text{ for all } x \in \mathcal{X}', u \in \mathbf{U}' \text{ and } t \in \mathbb{N}. \end{aligned} \tag{5.3}$$

$$J'(x,u) := \sum_{t=0}^{N-1} l({x_u}', x) \tag{5.4}$$

$$\mathbf{U^N}(x) := \{u \in \mathbf{U^N} | {x_u}'(t;x) \in \mathcal{X}' \text{ for all } t \in \mathbb{N}\} \tag{5.5}$$

5.3 Using MPC for partial differential equation control

This section will discuss one example problem in the Model predictive control setting. Chapter 8 discusses this problem and the results in more detail. Here we will define how we apply the actions in the MPC setting, the respective cost functions and the final optimization problem as an example of the previous section.

The control problem is about controlling a dynamic system governed by the heat equation where the domain of the system is given by $\Omega \in [0, L]$ where L is the total length of the one-dimensional domain. We use the following heat equation with control input,

$$\frac{\partial u(x,t)}{\partial t} = v \cdot \frac{\partial^2 u(x,t)}{\partial t^2} + a(x,t)$$

Here, u is the independent variable representing the temperature distribution in the domain, v is the diffusivity of the medium, $\frac{\partial^2 u(x,t)}{\partial t^2}$ is the diffusion term, and a represents the control input as predicted by the MPC controller. Let us say the total number of time steps is given by $T > 0$. The control space is given by $a \in \mathcal{A} \subset \mathbb{R}$, and the state space is $u \in \mathcal{X} \subset \Omega$. We define the control problem as finding the optimal control sequence a^* such that the system terminates in a reference state $u' \in \mathcal{X}$.

The initial state $u_0 = u(t = 0)$ is given by a random Gaussian function. The constraints are applied on the boundary of the domain, namely Dirichlet boundary conditions, such that the temperature at the boundary of the domain is insulated by the outside temperature. Such boundary conditions are given by,

$$u(x = 0, t) = u(x = L, t) = 0$$

The control input is given by,

$$A^L(u) := \{a \in A^L | {u'}_a(t;u) \in \mathcal{X} \text{ for all } t \in \mathbb{N}\}$$

One example of calculating cost function is using the mean least squares method Gauss and Davis [1964] i.e. the squared difference between the current state x and the reference state x' and is given as,

$$J(u,a) := \frac{1}{L}\sum_{t=0}^{T}(u_a(t) - u'(t))^2$$

Here, $u_a(t)$ is the system state resulting from applying control input a at time t. This cost function penalizes the deviation from the reference state, but it can be modified to also include the cost of control inputs by adding the square of control inputs in the cost function. The new cost function then becomes,

$$\begin{aligned} J(u,a) &= \frac{1}{L*T}\sum_{t=0}^{T}(u_a(t) - u'(t))^2 + \frac{1}{T}\sum_{t=0}^{T} a(t)^2 \\ &= \frac{1}{T}\sum_{t=0}^{T}\left[\frac{1}{L}(u_a(t) - u'(t))^2 + a(t)^2\right] \end{aligned} \tag{5.6}$$

Using the above cost function 5.6, the initial and boundary conditions 5.3, the minimization problem can be defined as,

$$\begin{aligned} &\arg\min_{a\in\mathcal{A}^L} J(u,a) \\ &\text{s.t. } \frac{\partial u(x,t)}{\partial t} = v\cdot\frac{\partial^2 u(x,t)}{\partial t^2} + a(x,t) \\ &\qquad u(0) = u_0 \text{ for all } u\in\mathcal{X}, a\in\mathcal{A} \text{ and } t\in\mathbb{N} \\ &\qquad u(0,t) = u(L,t) = 0. \end{aligned} \tag{5.7}$$

The MPC algorithm will solve this optimization problem at each time step and select the control input that gives the least value of this cost function.

6

Reinforcement Learning

Reinforcement learning is a framework of optimal control and machine learning that aims at solving "goal-oriented" problems by interacting with a complex dynamic system. It comprises an environment and an agent. The environment is a physical representation of a system that can be updated. For example, in fluid mechanics control problems, the environment is governed by partial differential equations. The agent is a controller that can interact with the environment to control it, keeping an objective in mind and following the principles of feedback control—the interaction results in an update of the environment's state. The environment then responds in the form of a reward as feedback for the chosen actions. Learning to control such a system is done by maximising this reward by considering the system's future state. The goal of an agent, then, is to choose actions in the direction of maximising this cumulative sum of rewards.

This form of learning is different from classical machine learning methods, like supervised and unsupervised learning. Supervised learning is about inferring a model using a set of labelled training datasets explicitly prepared for a problem. For better accuracy, the training data must be wholesome, i.e. comprising enough examples for each type of objective. For many problems, obtaining such training data is only sometimes possible. On the other hand, reinforcement learning can use partial observations of the environment and does not require training data in general. Reinforcement learning also differs from unsupervised learning, where the main focus is finding hidden structures in unlabelled data. Even though the information received from the environment is unlabelled, it is always accompanied by some feedback in the form of a reward, which makes the reinforcement learning method different from unsupervised learning. The goal of RL is to maximise a reward function. However, the technique of finding hidden structures in data can also enhance the capabilities of RL, e.g. Feng et al. [2020].

In recent years, reinforcement learning has gained widespread popularity with its ability to learn a task with ease, especially in the field of robotics Khan et al. [2020], revenue management Bondoux et al. [2020], and autonomous control of systems Kiumarsi et al. [2017]. The main focus of this thesis is the application of reinforcement learning for a set of physics problems, including but not limited to optimal flow control. The environment is derived using a set of physics laws expressed in partial differential equations for such physics problems. One of the challenges in RL is the trade-off between exploration and exploitation Sutton and Barto [2018]. In order to maximise the reward, the agent has to consider actions that proved effective in the past and finding such actions requires exploration of the action space. The idea is to balance exploration and exploitation so the RL agent can learn from past experiences while also exploring new actions that yield better rewards. In the following sections, we will discuss the main components of RL and how to formulate PDE problems as a Markov Decision Process (MDP) using an example of

the Burgers equation control problem.

6.1 Components of RL

RL has six main components as described in the book Sutton and Barto [2018]: an environment, an agent, a policy, a reward signal, a value function and optionally, a model. For the physics problems in focus here, an environment can be considered as the description of the physical world expressed by a single or a combination of partial differential equations and is represented by its state at a particular time. It also holds the properties of observability, i.e. the agent is allowed to measure the state of the environment partially, and its interactive ability where the agent is authorised to influence the state of the environment with its actions Sutton and Barto [2018].

The agent is the learner and decision-maker. At each timestep, the agent interacts with the environment by performing actions which update the state of the environment. In return, the environment responds with feedback about the quality of actions in the form of a scalar reward. The goal of the agent is to maximise this reward. Following the technical specification as in the book Sutton and Barto [2018], consider the agent interacts with the environment for a sequence of time steps $t = 1, 2, 3, ..., N$. At each timestep t, the agent receives a partial observation of the environment state given by $s_t \in \mathcal{S}$, where $\mathcal{S}$ is a set of all possible states. Using s_t, the agent selects an action $a_t \in \mathcal{A}$ where $\mathcal{A}$ is the set of all allowed actions and then applies it to its environment. This application of action results in a new state $s_{t+1} \in \mathcal{S}$, and in return, the agent receives a scalar reward from the environment of the form $r_t \in \mathbb{R}$.

During the training phase, the agent learns the environment's behaviour by learning a policy based on the defined problem. A policy, in general, is a mapping from states to the action given by $\pi := \mathcal{S} \rightarrow \mathcal{A}$. A policy determines the probability of each action for a given state from the action space and selects the actions with the highest probability to maximise the cumulative reward. The primary learning task for the agent is how to update the policy.

A reward formalises the goal of a reinforcement learning agent. The reward is calculated in the environment itself, and the agent has no access to how it is calculated. As pointed out in the book, "Reward is a way to communicate to the agent what needs to be achieved rather than how it should be achieved". For example, in a game of chess, a learning agent receives a reward only if it wins, loses or draws and does not receive rewards based on how many pieces it won. The reason is that there are strategies in the game which result in a win even after losing more pieces.

As mentioned, learning happens by maximising a cumulative sum of rewards and not just individual rewards at a timestep. This generally refers to maximising an expected return G_t. For example, for a discrete problem with a finite timestep of $T \in \mathbb{N}$, the expected return is given by $G_t = r_t + r_{t+1} + r_{t+2}, ... + r_T$. Where r_t is the reward at timestep t, this makes sense for discrete tasks, but things get more complex for continuous problems. A terminal state is unknown for continuous problems, so equally weighing all the rewards could result in an infinite reward cycle that prevents the agent from exploring new actions. This can be solved by weighing each action differently based on when the rewards are received using a discounting factor called the discount rate Sutton and Barto [2018]. The expected return for continuous tasks can be formulated as $G_t = r_{t+1} + \gamma \cdot r_{t+2} + \gamma^2 r_{t+3} + ...$ where γ is the discount rate and $0 \leq \gamma \leq 1$. The discount factor is a measure of how important future rewards are. If $\gamma < 1$, the bounded reward sequence r_t results in a finite expected return. For $\gamma = 0$, the agent only considers the immediate reward, and as γ approaches 1, the agent weighs the future rewards more Sutton and Barto [2018].

Just as rewards describe the quality of action for the current timestep, a value function represents the quality of actions for all the time steps. There cannot be a value function without a

reward. The environment gives a reward, while the value function must be estimated separately. It represents the expectation of total reward given a state and is given by,

$$V^{\pi}(s) := \mathbb{E}\left[\sum_{i=1}^{T} \gamma^{i-1} r_i\right]$$

Here, $V^{\pi}(s)$ is the expectation given a policy π. The final component is a model that is the learned representation of the environment and predicts how the environment will behave for specific actions. The model helps evaluate the chosen actions before the environment executes them. The RL methods using a model are termed model-based methods, while the others are model-free. In this thesis, we focus solely on model-free methods when it comes to RL.

6.2 PDE as MDP

In this section, we will explain how to frame a PDE control problem as a Markov decision process (MDP) which is the mathematical framework for solving a problem as a reinforcement learning problem. The examples in this section are originally the work of authors in the paper Pan et al. [2018a], and we have included them because it serves as an excellent example for representing continuous state and action spaces in PDE as MDP problems.

In the book introduction to reinforcement learning Sutton and Barto [2018], MDP is defined as a process that comprises states that follow the Markov property, which represents those states that retain all relevant information throughout the environment's execution. We will refer to the formal definition of MDP as provided in Pan et al. [2018a]: A finite-action discounted MDP is given as $M = (\mathcal{X}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$; where $\mathcal{X}$ and $\mathcal{A}$ are state and action spaces, $\mathcal{P}$ is a mapping from $\mathcal{S} \times \mathcal{A} \to \mathcal{R}$ called the transition probability, $\mathcal{R}$ is the reward and γ is the discount factor.

Next, we explain how to formulate a continuous PDE control problem with the Burgers equation as an RL problem. The control problem discussed here is about controlling the velocity of a dynamic system governed by the burgers equation for T number of time steps and is given by,

$$\frac{\partial u(x,t)}{\partial t} = v \cdot \frac{\partial^2 u(x,t)}{\partial t^2} - u(x,t) \cdot \frac{\partial u(x,t)}{\partial x} + a(x,t) \tag{6.1}$$

Here, $u(x,t) \in \Omega$ is the independent variable describing the velocity of a field in a domain given by $\Omega \in \mathbb{R}^d$, $d = 1, 2, ...\mathbb{N}$ and L as the total length of the domain. The terms $v \cdot \frac{\partial^2 u(x,t)}{\partial t^2}$ and $u(x,t) \cdot \frac{\partial u(x,t)}{\partial x}$ represent the diffusion and advection processes respectively. The diffusion rate in the domain is given by the viscosity constant v, and $a(x,t)$ represents the actions applied to the system.

Consider a circular domain such that we can apply the periodic boundary conditions 3.2. This means that the velocity values in the domain are repeated outside the domain. These boundary conditions can be represented as,

$$u(0,t) = u(L,t) \tag{6.2}$$

Note that using the Dirichlet boundary conditions here will imply a loss of energy at the domain's boundaries. So, we have chosen periodic boundary conditions. The next step is to define the state/observation and action spaces. We will consider the velocity field along with the time information for observation space. Let us say the observation space is given by $\mathcal{X}$,

$$\mathcal{X} = \{x = (u,t) : u \in \Omega, t \in T\}$$

We can also incorporate more information in the observations provided to the agent based on the problem specification. For example, we can include reference state information for the trajectory matching problem. The next step is to define the action space. We use two jets as the control actuators, and the speed of the jets is the control input. Our agent will predict scalar actions in the range $\mathcal{A} \in [-1, 1]$, which the speed of the jets can describe. The control outputs are a_1 and a_2 for jets 1 and 2, respectively. Before applying these actions, we will transform these actions such that they are compatible with the velocity field described here. This transformation function can be given by $f^* : \mathbb{R} \to \mathbb{R}^d$. The action space is defined as,

$$A = \{(a_1, a_2, t) : a_1 \in \mathbb{R}, a_2 \in \mathbb{R}, t \in T\}$$

The next step is to define the transition probability kernel. Let X be a random variable representing all states' distribution. Then the value of state at each time step is given by the probability of a state given the previous state and the action at that time, represented as Pan et al. [2018a],

$$X_{t+1} \sim \mathcal{P}(\cdot|X(t), a(t))$$

Until now, we have defined the observation space, action space and the transition probability kernel. The next step is to define our reward and the respective value function. The reward function here takes the states and actions into account and is given by $r : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$. For each control problem, the reward changes based on the problem objective. In the current problem, the goal is to reach a reference state. So, we can use Mean Least Squares method to calculate the deviation from the reference state. If we want to consider the cost of actions, we can also include that in the reward. This usually is helpful when we want to restrict the fluctuations in the agent's action choices. The final reward is represented by

$$R_t(x, a) = -\sum_{t=0}^{L} |u(x, t) - u^*(x, t)|^2$$

Here, u^* represents the reference state at time t. The policy that learns the mapping from state to actions is given by $\pi : X \to A$. For each timestep, the actions are chosen using the policy, i.e. $A_t = \pi(x_t)$. Based on policy the action-value function can be defined $Q^\pi : X \times A \to \mathbb{R}$ Pan et al. [2018a],

$$Q^\pi(x, a) := \mathbb{E}\left[\sum_{t=1}^{L} \gamma^{t-1} R_t | X = x, A = a\right]$$

and the optimal value function is given by Pan et al. [2018a],

$$Q^*(x, a) = \sup_\pi Q^\pi(x, a)$$

The RL algorithm uses this optimal value function to choose the best policy that maximises reward and also minimises cost.

6.3 Comparison of RL and MPC

In this section, we will discuss some differences in MPC and RL frameworks in terms of the terminology, the approach, the usage of model and the computational efforts. There are a few similarities on a high level between each component e.g. plant and environment, cost function and reward, control input u_k and action a_k. However, there are also some minor differences

among them. For example, The state in MPC is related only to the internal properties of the system as defined by the model but in RL this state can also represent both, the internal and external properties of the system Arroyo et al. [2022]. The plant in MPC represents a system process and only gives out the system's state information while the environment in MDP also gives out information about rewards along with the state. MPC control focuses on minimizing an objective function while MDP is about maximizing the cumulative reward using a value function. In the deterministic setting the immediate reward in MDP, r_{t+1} can be related to the objective function as follows Arroyo et al. [2022]:

$$r_{t+1} = -(J_{t+1} - J_t)$$

Technically speaking, the MPC approach is different from RL in terms of the feedback received and how the objective function is used. In MPC, the controller solves the optimization problem to choose the best set of actions for a finite amount of time and predicts the future set of actions for a finite amount of time in future using only the state of the system as feedback from the plant. At each timestep t, the observer receives a measurement y_t from the plant which is converted to the state of the system x_t that characterizes the controller model at the current time. Using this information, the controller solves a minimization problem for a finite number of time steps dt. This optimization is done for the objective function J using the controller model F in the presence of constraints for the PDE. Solving this optimization problem results in mapping the state vector to a sequence of actions for a finite horizon out of which only the first action is chosen as the action being applied followed by the system moving forward in time. There onward, the entire process is repeated. The objective function used in MPC is the integral of a function l of the model variables along the prediction horizon Arroyo et al. [2022].

$$J = \int_{t=0}^{t+\Delta t} l(x(t), u(t), z(t), d(t))dt$$

In terms of the application of MPC to PDE control, the state vector x typically represents physical quantities like the entire temperature of the room or the total velocity of the fluid in a domain. u represents the control actions that could be external forces applied to the system e.g. airflow induced by turning a fan ON/OFF. z represents the dependent variables like the individual temperature values or velocity at a certain point in space at a particular time and d represents the set of uncontrollable variables e.g. heat source or solar irradiation.

On the other hand, MDP in RL consists of a state space X, action space A, transition probability P, reward distribution R and discount factor γ. The transition probability function is different from the model F in MPC as it represents the ground truth information rather than being a simplified representation of the system Arroyo et al. [2022]. At each timestep, the agent receives a partial observation of the environment in the form of a state vector similar to MPC but here the agent maps the state information to actions using a policy function that internally uses the transition probability function. The output of the policy function is an action that is applied to the environment to proceed to the next step and in return, the environment responds with a scalar reward R_k describing how good the action was. The main idea behind the policy is to maximize a discounted cumulative sum of rewards given by G_t Arroyo et al. [2022],

$$G_t = r_{t+1} + \gamma r_{k+2} + \gamma^2 r_{k+3} + ... = \sum_{i=0}^{\infty} \gamma^i r_{t+i+1}$$

In terms of how the models and function approximations are used, a controller model in MPC is obtained through techniques like system identification and is a representation of system dynamics. There is also a further requirement of simplifying the controller model to guarantee

convergence but this results in performance loss Arroyo et al. [2022]. While in RL, models approximate the policy or value function which does not directly represent system dynamics.

7

PhysicsGym Interface

In this chapter, we will discuss the main components of our PhysicsGym interface, its design decisions, and how to configure and extend the interface. Physics-gym interface establishes a connection between the RL framework Openai-gym Brockman et al. [2016b] and the partial differential equation solver library Phiflow Holl et al. [2020]. It is designed to use reinforcement learning to find optimal control strategies for different physics engineering problems, consisting of a dynamic system governed by the rules of partial differential equations.

7.1 Interface design

(a) OpenAi Gym

(b) Phiflow

Figure 7.1: Main methods in OpenAI gym and Phiflow's PDE solver interface.

We begin by explaining the main methods in Phiflow's PDE solver interface and OpenAI gym interface and then explain how the PhysicsGym interface will help connect the two. Figure 7.1 (a) shows the main methods in OpenAI gym's environment interface. It starts with initialising the environment, which defines the observation space, a.k.a. state space and the action space

of the problem. The reset method resets all the variables of the environment, representing the system's current and cumulative state. Lastly, the step method is the primary method that takes action as input and applies it to the environment updating its state. This results in the new state of the system and respective rewards, along with some information about execution. Other methods in the OpenAI Gym interface also exist, but they are left out of this discussion because they do not explicitly affect our interface.

Figure 7.1 (b) shows the main methods in Phiflow's PDE solver. Phiflow provides a *Physics* interface with two main methods, initialisation and update. The initialisation method takes as input all the variables necessary for executing the simulation, e.g. viscosity for the burgers equation, diffusivity for the heat equation and more. The update method calls routines for performing diffusion and advection, which use the finite difference method to find the solution to a PDE.

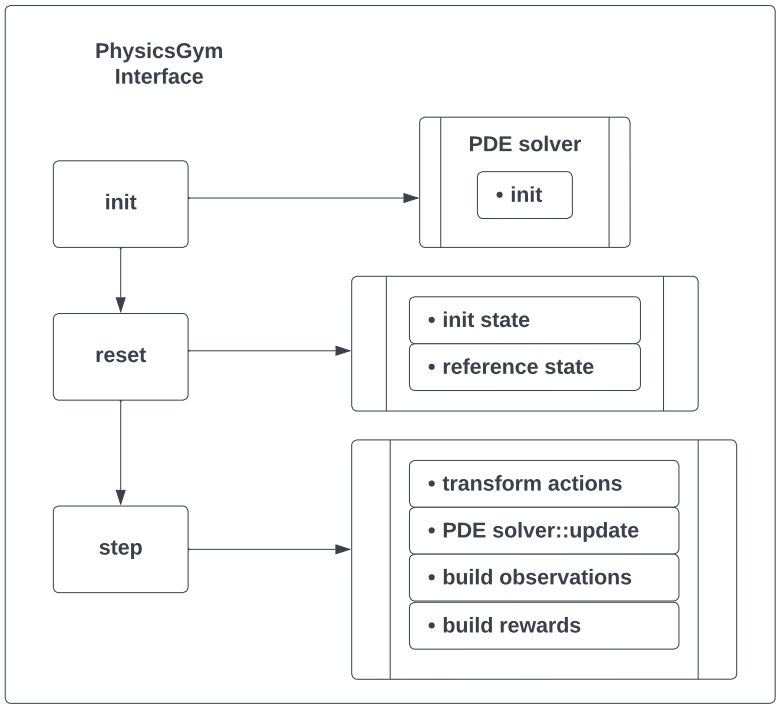


Figure 7.2: PhysicsGym interface representation.

Figure 7.2 shows the basic block diagram of our PhysicsGym interface. The structure resembles that of OpenAI gym's environment interface, but internally we provide some conversion methods that act as a middle-ware between the PDE solver and any reinforcement learning agent.

7.1.1 The constructor

PhysicsGym starts with initialising the environment in *init* method, which involves instantiating three main variables, observation space, action space, the physics object from Phiflow, and other problem-specific variables like domain, resolution and time-step information. Both observation and action spaces are defined using the Box object from Gym, which involves setting the low value and the high value of each space along with the shape of the space, which is specific to the problem at hand. For example, if we want our control agent to predict a scalar action in

the range [-1,1], then low=-1, high=1, shape=(1,). For the problems targeted in this thesis, the observation space is continuous in the range [-inf, inf].

7.1.2 The *reset* method

The *reset* method re-initialises the essential variables which define the environment's state and the problem when it is called in the first step. This method explicitly sets the initial and reference states and the counter variables that record the time state of the environment. The type[1] of state can be anything compatible with the PDE solver, but the observation returned by this method should always be an array of Numpy objects to be compatible with reinforcement learning framework: OpenAI Gym. The step method in OpenAI gym takes as input the actions predicted by the agent, updates the environment and gives feedback on the actions. Our interface enhances this method by incorporating additional processing steps that make it possible to use any PDE solver.

7.1.3 The *step* method

Our *step* method comprises three steps:

1. Pre-process actions
2. Apply actions
3. Post-process results

To understand each step, let us consider the simple problem of controlling the temperature in a three-dimensional room where a control action is performed by adjusting the speed of a fan inside the room. Such a control input results in action being scalar and continuous. Applying this control action on the environment requires simulation of airflow from the fan scaled by the control action, e.g. speed of the fan. We handle this in the pre-processing step, which takes a scalar action and transforms it into a field that can be applied to the environment and influence it. The transformation also works if the predicted actions are a high dimensional vector instead of scalar, the only requirement is that the shape of actions should be compatible with the environment so it can be applied to the environment. For the purpose of transformation of actions, we provide a helper function named, *action_transform* which is invoked by our *step* method on each subsequent calls.

The second step is about updating the environment by applying the prepared actions and involves calling Phiflow's step/update method from the Physics interface. The step method in the physics interface involves calling routines like diffusion, advection and more. And then mathematically updating the resulting state by adding or subtracting the actions based on the type of problem. The result of this step is an updated state that is still type-compatible with the PDE solver. In the case of Phiflow, this object is of type 'field' that cannot be used directly with the RL agent. So, the final step does some necessary post-processing to make things compatible with the RL framework. It does so by first converting the updated state to a Numpy object, second calculating the reward and normalising it and third by preparing some information about the current state of execution of the experiment.

The output of the entire step method is a four tuple consisting of (observation, reward, done, and info). Here *observation* is the updated state of the environment. We provide the helper method *build_observations* which is invoked after the physics::step method and it is used to convert the observation vector based on the defined problem. *Reward* is the normalised

[1]Here by type, we are referring to the object type in python language setting.

reward for the current action and one should define the reward calculations in the helper method *build_rewards* which is invoked after the *build_observations* method. Finally *done* represents if the environment has reached the final stage of execution (to be defined during initialisation) and *info* is a way to capture some intermediate variables for the evaluation of the learner. *Info* can involve individual variables that define the reward.

7.2 How to use

Now we will explain how to use the PhysicsGym interface to define stabilisation or trajectory-matching problems. We will use the Phiflow library as the PDE solver, but one can use any other library and appropriate conversion functions. Phiflow's physics interface consists of predefined routines that can perform diffusion or advection on the given state (field), which we use in this thesis. However, one can also define a custom method using Laplacian operators defined in Phiflow. The physics interface comes with a step function that uses these routines and applies the actions to update the state of the environment. Consider the problem discussed previously for controlling the heat equation. Here the domain is a 3d room, and the control action is the fan's speed. In the constructor of PhysicsGym implementation, we first define the domain(=3) and resolution(=0.25), which gives the total number of points (N) in the field to be (domain/resolution=) 12. We then define the observation space to be of shape equal to N, and since the action space is scalar, the shape is a tuple (1,). The final thing needed for initialisation is the physics object, and for this scenario, we will use the Heat equation solver defined in Phiflow. Next, in the reset method, we will define the initial state to be sampled from a uniform distribution of observation space and the reference state to be 0 for stabilisation problems. For trajectory-matching problems, the reference state will be the target vector.

Following this, we will define the components of the step method, where the first step is to define the action transformation function. In this scenario, we use uniform distribution with the mean at the centre of the domain. This is equivalent to the fan being on the bottom centre of the room with airflow directed towards the top. In a real-world scenario, the airflow generated by the fan can be calculated more precisely using the Navier-Stokes equation Doering and Gibbon [1995]. However, to save computational time, we have used a uniform distribution to resemble airflow in the centre of the room. On each update step, the agent receives a reward, defined as the weighted sum of the squared difference between the current state and the reference state. The step method also normalises, and one can use routines like Rootmeansquare from stable_baseleines3 Raffin et al. [2021]. One important thing to note is that the agent performs the action and then lets the environment simulate a certain number of steps with the same action before changing it; this is done to make it more realistic. The number of steps involved in this simulation can be controlled separately while defining the environment.

In the next section, we will discuss some detailed experiments using PhysicsGym to control environments governed by the heat equation and the burgers equation. We will also evaluate the results for a baseline, RL, and MPC agent.

8 Experiments

In this chapter, we will conduct two experiments using the heat equation and the Burgers equation. For both experiments, we will first show the uncontrolled simulation, which will give a general idea about how the equations behave in the domain given an initial state and what actions we perform to resemble real-world scenarios. Following this, we will use three different agents to interact with an environment developed using our Physics-Gym interface. Then we will analyse their performance on the respective control problems.

The controllers used in these experiments can only influence the environment by applying scalar actions $a \in \mathbb{R}$. We will transform these scalar actions into a vector of the size of the domain such that the actions are effective only on the left half of the domain. This way, the actions only influence a part of the domain but are still propagated to other points in time because of advection and diffusion mechanism. This way of applying actions resembles a real-world scenario where one cannot influence each quantity in the system. This transformation function is given by: $\mathbb{R} \to \mathbb{R}^N$. The controller's task then is to predict the correct magnitude and directions of the actions for each time step, bringing the system's state closer to the reference state. This type of setup is specific to the PDE control problem discussed here. Changing this transformation function can influence how the actions affect the environment.

We will use the first agent based on the MPC algorithm but with a prediction horizon of 1. We will use this as a baseline to evaluate the performance of other agents. At each step, the baseline agent will solve an optimisation problem of finding the minimum of a cost function. The cost function is defined using the Least mean squares error function that highlights how far the system's state is from the reference state due to the chosen actions.

The second agent is the RL agent, which uses an actor-critic algorithm that works with continuous state and action spaces. It aims to learn a policy $\pi : \mathcal{X} \times \mathcal{A} \to \mathcal{A}$ by evaluating randomly chosen actions using the reward received from the environment. The reward is calculated similarly to the cost function in our baseline agent, i.e. based on how far the current state is from the reference state. We will train the RL agent for fewer time steps than during the test to get a good idea if the agent learnt the system behaviour and generalised properly. For this, we train our agent on randomised state values. Then finally, we will use the MPC agent, which will predict optimal actions similarly by minimising the same cost function as the other two agents. However, it will be able to simulate and check the quality of its actions for a prediction horizon of 5/10 steps.

8.1 Heat equation control experiment

In this experiment, we will solve a simple control problem using a dynamic system governed by the Heat equation given by,

$$\frac{\partial u(x,t)}{\partial t} = -v \cdot \frac{\partial^2 u(x,t)}{\partial t^2} \tag{8.1}$$

The problem statement goes like this, consider a dynamic system $y(t) = f(x, a, t)$ with a domain of size N, $x \in \mathcal{X}$ being the temperature of each particle in the domain and $a \in \mathcal{A}$ being the control input and v is the diffusivity of the domain. Find the optimal control sequence $a^* \in \mathcal{A}$ such that the optimal control results in the system terminating in state 0. Here 0 state means that the temperature of all particles is 0. Introducing the control term in 8.1, the equation becomes,

$$\frac{\partial u(x,t)}{\partial t} = v \cdot \frac{\partial^2 u(x,t)}{\partial t^2} + a(x,t) \tag{8.2}$$

This dynamic system governed by new heat equation 8.2 calculates the temperature of each particle in the domain by diffusion process given by $v \cdot \frac{\partial^2 u(x,t)}{\partial t^2}$. We have used Phiflow Holl et al. [2020] as the PDE solver where the solver is given by $f' : \mathcal{X} \times \mathcal{A} \to \mathcal{X}$, here $\mathcal{X}$ represents the state space and $\mathcal{A}$, the action space. The initial state $x(t = 0) = x_0 \in \mathcal{X}$ is chosen uniformly at random for each agent being evaluated. The boundary constraints are set to Dirichlet boundary conditions, meaning that the domain's boundaries are insulated, and the temperature at the boundaries is 0.

In the rest of the chapter, we will first show how the uncontrolled simulation of our problem statement looks like followed by the performance of our baseline agent, RL agent and MPC agent and finally, concluding this chapter with a short comparison between the three regarding generated state trajectories and the received rewards.

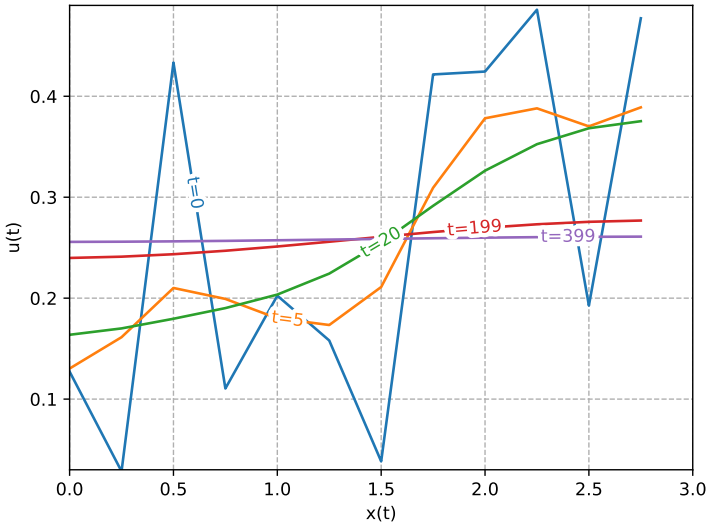


Figure 8.1: Simulation of Heat equation 8.1 with no control.

8.1.1 Uncontrolled simulation

Next, we will show how our system with the heat equation at 8.1 behaves without any control input. Roughly speaking, the equation propagates the particles with higher temperatures

downwards and those with lower temperatures upwards in the domain. Figure 8.1 shows the time evolution of the temperature field for different time steps, demonstrating this behaviour. At $t = 0$, the state is initialised randomly from a domain of size $N = 12$. This initial state has three local maxima ($x = 0.5, 2.0, 2.75$), a global maximum ($x = 2.25$), three local minima ($x = 0.75, 1.5, 2.5$) and a global minimum. $t = 5$ shows the reduction in temperature at all four maxima and increment at the minima, which continues until the temperature is the same at each point in the domain, in this case at $u = 0.25$. Observing the evolving states at each time step, it is clear that all the higher temperature values tend to move downwards and the lower temperature upwards in the domain. This displacement occurs at a rate proportional to the diffusivity of the medium. For this experiment, we have chosen the domain of size 12 with a diffusivity of 2.0 and updated the environment with a step size of 0.01 for 400-time steps. Based on the environment's behaviour, it is evident that the optimal control sequence would be to choose an action in the direction opposite to that of the state but only until a certain point. After that, the equation should balance itself, which in our case, means bringing the temperature of all particles to an equilibrium.

Now we will analyse how different types of agents will find the solution to this problem and then compare them.

8.1.2 Baseline agent

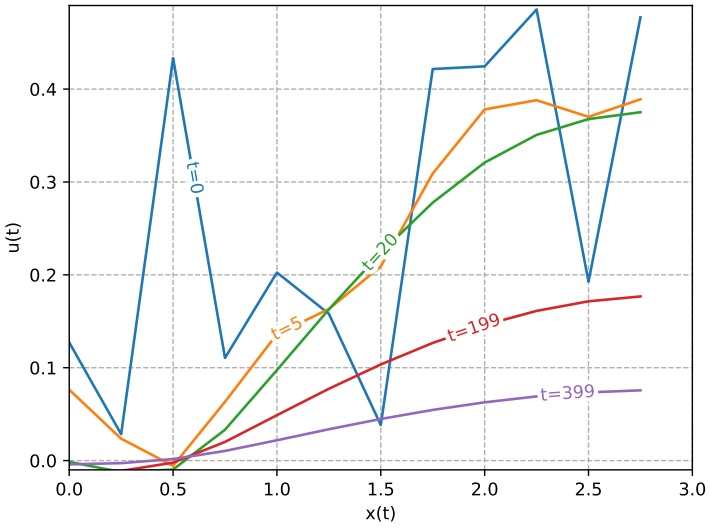
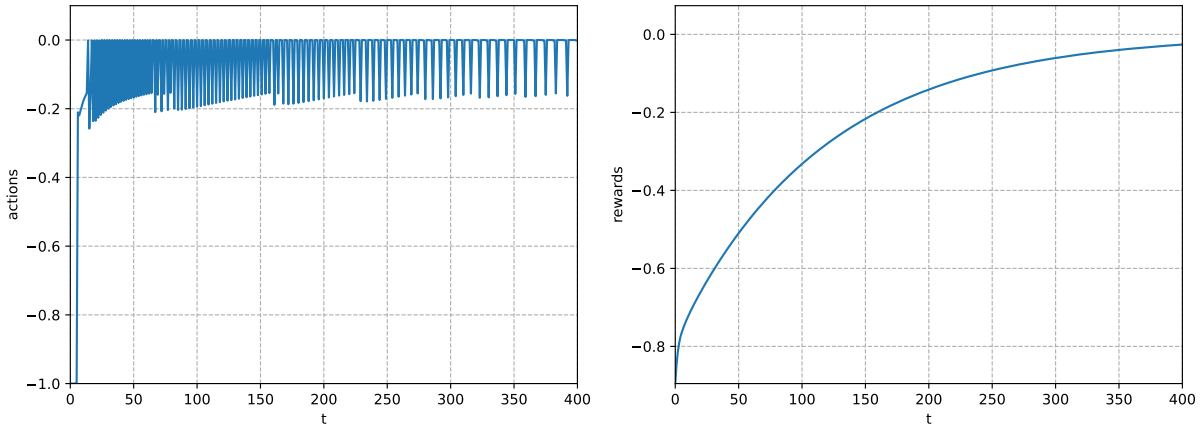


Figure 8.2: Simulation of Heat equation as controlled by the baseline agent.

Starting with the baseline agent, figure 8.2 shows the state trajectory due to the actions predicted by our baseline agent. It is clear from the figure that the agent can predict correct actions similar to the optimal actions discussed before. The reason is that it minimises the cost function $J : \mathcal{X} \to \mathcal{A}$, which in our case is given by mean least squares function Gauss and Davis [1964]. So, any action that takes the state further away from the reference state is penalised. The actions predicted by our baseline agent are shown in figure 8.3. In the beginning, the agent predicts the action with the highest magnitude -1.0 in the direction opposite to the state value at the point of application ($x = 0.5$) and then reduces the magnitude of actions gradually in future, oscillating between 0 and -0.2. The final result is a state close to the 0 states ($x = 0.0$ to 0.5) while the rest of the values terminate between ($u = 0.0$ to 0.1). Now we will see how the reinforcement learning agent performs in this scenario.



(a) actions (b) rewards

Figure 8.3: Actions predicted by baseline agent and respective rewards.

8.1.3 RL agent

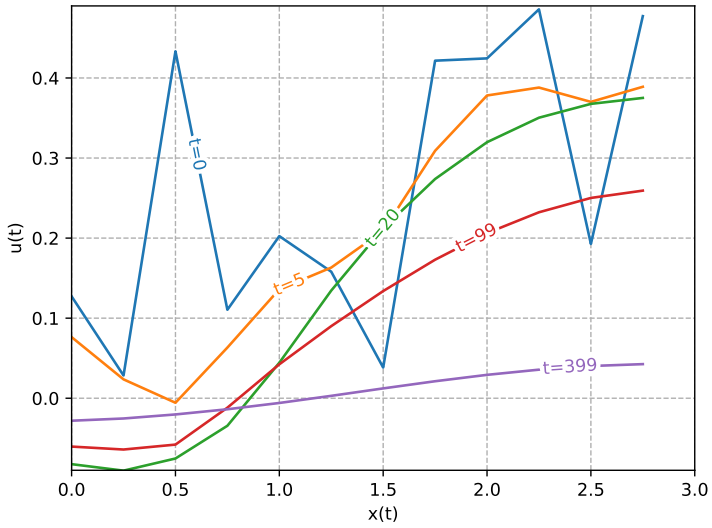
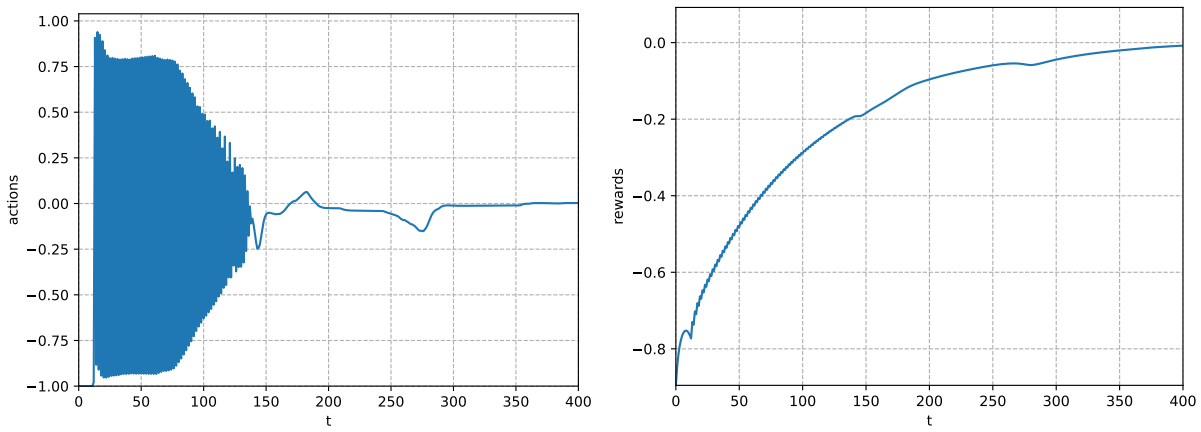


Figure 8.4: Simulation of Heat equation under actions predicted by the RL agent.

For learning the environment's behaviour, we used the DDPG agent, a baseline for controlling systems that involve continuous state and action spaces. We used the default implementation of stable-baselines3 Raffin et al. [2021] DDPG agent, which we trained for 1000 epochs with a learning rate of 10^{-4}. Figure 8.4 shows the state trajectories resulting from the actions predicted by the RL agent. Here we have used random initial states, so the agent gets access to diverse state information during training and kept the reward function similar to that of the baseline agent. It is clear from the figure that the RL agent can predict optimal actions and can bring the temperature of each particle closer to 0. It does so by choosing the action of the highest magnitude 1.0 in the direction opposite to the temperature value at the beginning $t = 0$ and then reducing the magnitude in further steps until $t = 150$ approximately in figure 8.5 (a). From $t = 150$ onward, the agent reduces the magnitude of actions for a short time interval to bring the state below 0 value and then the actions are closer to 0 because the equation will balance

itself near the 0 states by itself in the remaining time steps. The smooth reward curve in figure 8.5 (b) shows the agent can successfully maximise the rewards from -0.8 to 0.0.



(a) actions (b) rewards

Figure 8.5: Actions and rewards analysis, predicted by the RL agent.

8.1.4 MPC agent

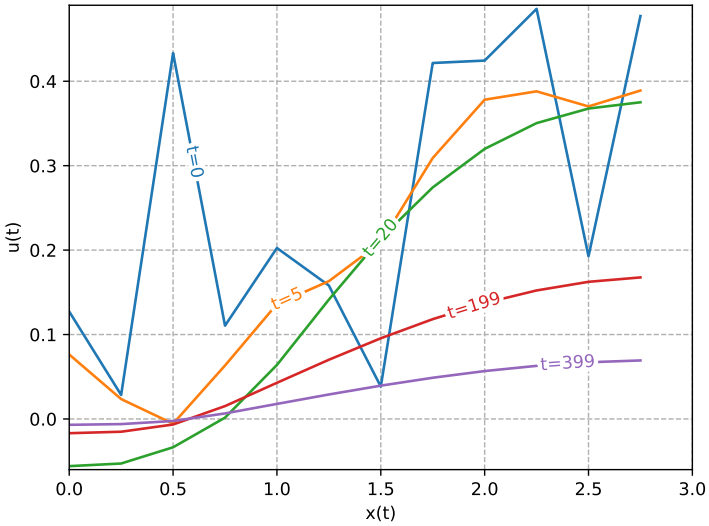
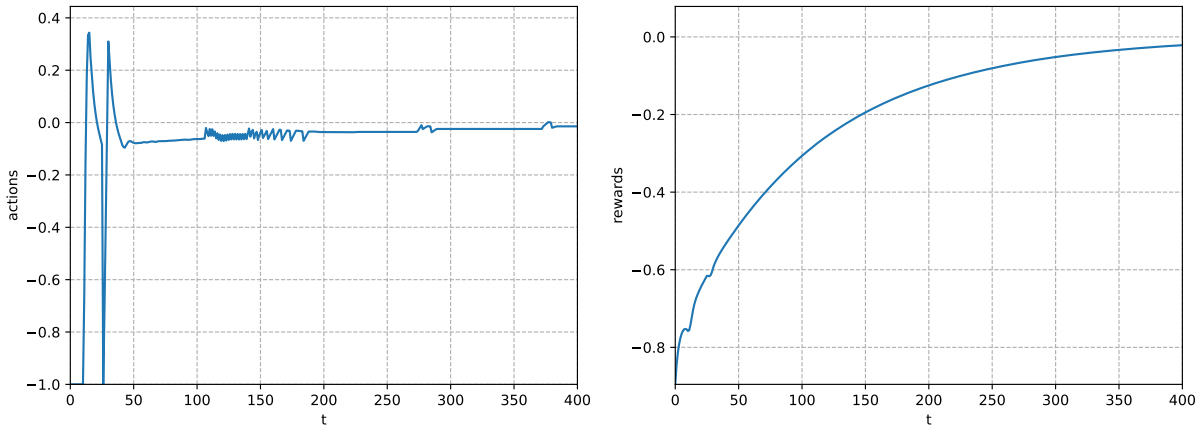


Figure 8.6: Simulation of heat equation using MPC agent.

Figure 8.6 shows the intermediate states resulting from actions predicted by the MPC agent. We used the prediction horizon of 10 in this scenario. It is evident from the figure that the MPC agent can stabilise the equation close to 0 but not at 0. This is because the state value between $x = 0.0$ to 0.5 is slightly below 0. On further time steps, the equation will balance itself slightly above 0, unlike the RL agent's output. The actions predicted by the agent are shown in figure 8.7 (a), which are close to the trend we predicted for optimal actions. It starts with actions of the highest magnitude in the direction opposite to the temperature values and oscillates till $t = 50$ between -1.0 to 0.4, thereby gradually stabilising to 0. The agent here took the path of reducing the temperature value to the lowest value possible with the actions and then letting

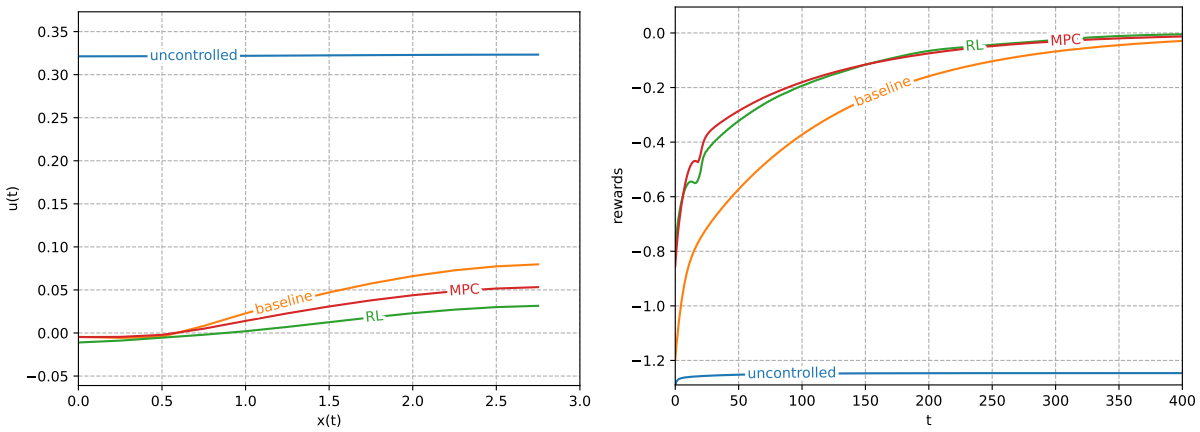
the equation balance itself closer to 0. In an HVAC unit, this could resemble turning the air conditioning on at the highest cold setting for a while and then turning it off. If the room is adequately insulated, then the temperature will reduce in the whole room at a rate proportional to the diffusivity of the domain. The rewards generated by the MPC agent are shown in figure 8.7 (b), which shows a smooth curve similar to the baseline and RL agent. One thing to note about MPC performance in this scenario is that the results can be better if a prediction horizon of higher magnitude was used because the MPC agent will have a higher look ahead to predict actions which bring it closer to the goal.



(a) actions (b) rewards

Figure 8.7: Actions predicted by the MPC agent and respective rewards.

8.1.5 Evaluation



(a) final states (b) rewards

Figure 8.8: Analysis of states and rewards, for baseline, RL and MPC agents.

Finally, comparing all three agents for ten random initial states, the resulting final states are shown in figure 8.8 (a). Here, RL and MPC agents are better than the baseline in reducing the temperature value closer to 0, i.e. between $x = 0.0 - 0.05$, with RL agents performing slightly better than MPC with the current setup. Note that the performance of MPC can be improved

with an increase in prediction horizon. The respective rewards are shown in figure 8.8 (b), which shows RL and MPC agents with similar rewards from $t = 100$ onward. The RL agent shows slightly lower rewards than MPC before $t = 100$. This is because the RL agent chose actions that brought the state below 0 during that time step, reducing the rewards. Nevertheless, this control strategy, in turn, helped the RL agent to bring the final state closer to 0. Hence the performance is slightly better than the MPC agent in this scenario.

8.2 Burgers equation control experiment

In this experiment, we will solve a simple control problem using a dynamic system governed by the viscous burgers equation given by,

$$\frac{\partial u(x,t)}{\partial t} = v \cdot \frac{\partial^2 u(x,t)}{\partial t^2} - u(x,t) \cdot \frac{\partial u(x,t)}{\partial x} \tag{8.3}$$

The problem statement goes like this, consider a dynamic system $y(t) = f(x, a, t)$ with the domain of size N, $x \in \mathcal{X}$ being the velocity of each particle in the system and $a \in \mathcal{A}$ being the control input. Find the optimal control sequence $a^* \in \mathcal{A}$ such that the optimal control results in the system terminating in state 0, which means that the velocity of all particles is 0. Introducing the control term in 8.3, the equation becomes,

$$\frac{\partial u(x,t)}{\partial t} = v \cdot \frac{\partial^2 u(x,t)}{\partial t^2} - u(x,t) \cdot \frac{\partial u(x,t)}{\partial x} + a(x,t) \tag{8.4}$$

This dynamic system governed by the new burgers equation calculates the velocity of each particle in the domain by advective and diffusive processes given by $u(x,t) \cdot \frac{\partial u(x,t)}{\partial x}$ and $v \cdot \frac{\partial^2 u(x,t)}{\partial t^2}$ respectively. We have used Phiflow Holl et al. [2020] as the PDE solver where the solver is given by $f' : \mathcal{X} \times \mathcal{A} \to \mathcal{X}$, here $\mathcal{X}$ represents the state space and $\mathcal{A}$, the action space similar to the setup in the previous experiment. The initial state $x(t = 0) = x_0 \in \mathcal{X}$ is derived from the normal distribution, and the boundary conditions are set to periodic. This means that the values propagating outside the domain are repeated with the values from inside the domain. This differs from the previous experiment, where Dirichlet boundary conditions would imply a loss of energy at the boundaries.

We will use a similar action space as the previous experiment, and the actions will only influence the velocities between $x = 0.0$ to 0.5 in the domain. There-onward, this effect should be propagated forward because of the advection-diffusion mechanism.

In the rest of the chapter, we will first show what the uncontrolled simulation of our problem statement looks like, followed by the performance of our baseline agent, RL agent and MPC agent and finally, concluding this chapter with a small comparison between the three.

8.2.1 Uncontrolled simulation

Next, we will show how the burgers equation at 8.4 behaves in the absence of any control input. Roughly speaking, the equation propagates the particles with positive velocity forward and those with negative velocity backwards in the domain. Figure 8.9 shows the time evolution of the velocity field for multiple time steps, demonstrating this behaviour. Observing the evolving states at each time step, it is clear that all the positive values tend to move towards the right and the negative values towards the left of the domain. This displacement occurs at a rate inversely proportional to the viscosity of the medium. This experiment is performed on the domain of size 4 with a viscosity of 3×10^{-3} for a total time step of 400, where each step size is 0.01. Based on the equation's behaviour, it is evident that the optimal control sequence would be to choose

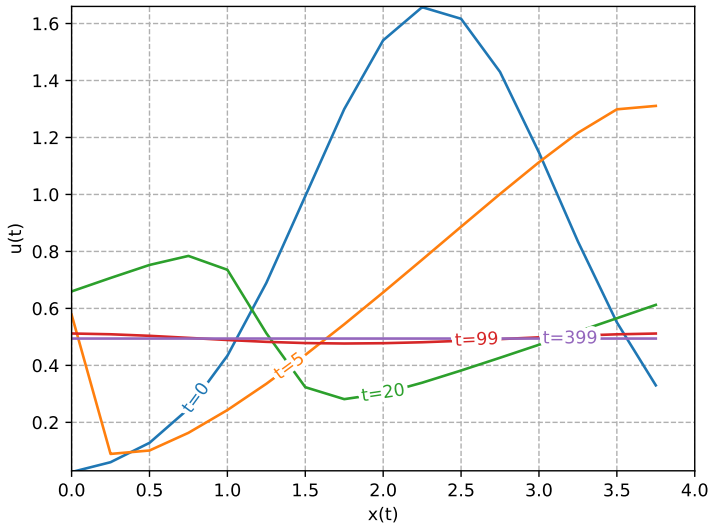


Figure 8.9: Simulation of Burgers equation 8.4 with no control.

an action in the direction opposite to that of the state but only till a certain point. After that, the equation should balance itself, which in our case, means bringing the velocity of all particles to 0.

Now we will analyse how different types of agents will find the solution to this problem and then compare them.

8.2.2 Baseline agent

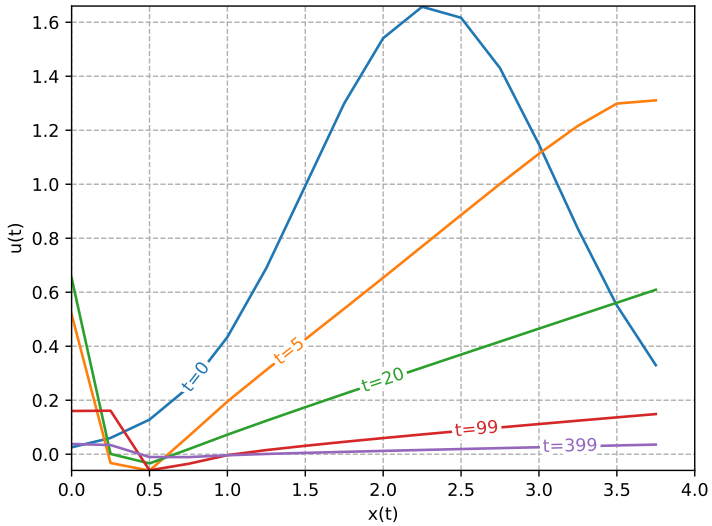
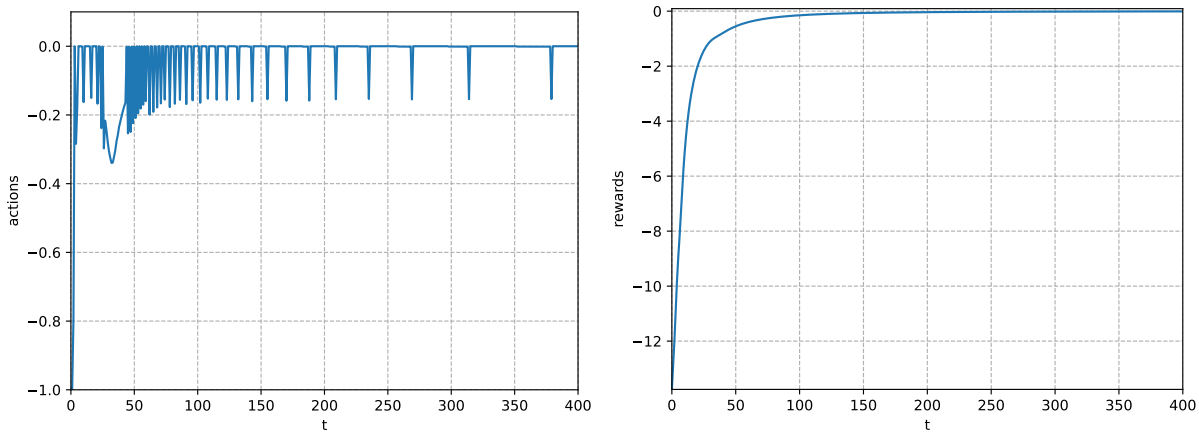


Figure 8.10: Simulation of Burgers equation using the baseline agent.

Starting with the baseline agent, figure 8.10 shows the state trajectory due to the actions predicted by our baseline agent. Here the initial state is defined by a uniformly random normal distribution to keep the diversity of states intact. It is clear from the figure that the agent can predict good actions similar to the optimal actions discussed before. This is because it minimises the cost function $J : \mathcal{X} \to \mathcal{A}$ which in this case is also given by mean least squares function



(a) actions (b) rewards

Figure 8.11: Simulation of Burgers equation and the actions predicted by the baseline agent.

Gauss and Davis [1964].

The actions predicted by our baseline agent are shown in figure 8.11 (a). In the beginning, the agent predicts the action with the highest magnitude -1.0 in the direction opposite to the state value at the point of application ($x = 0.5$) and then reduces this magnitude in future, oscillating between 0 and -0.2 approximately. The final result is a state partially closer to the 0 states ($x = 0.5$ to 1.5) while the rest of the values terminate between ($u = 0.0$ to 0.05). Now we will see how the reinforcement learning agent performs in this scenario.

8.2.3 RL agent

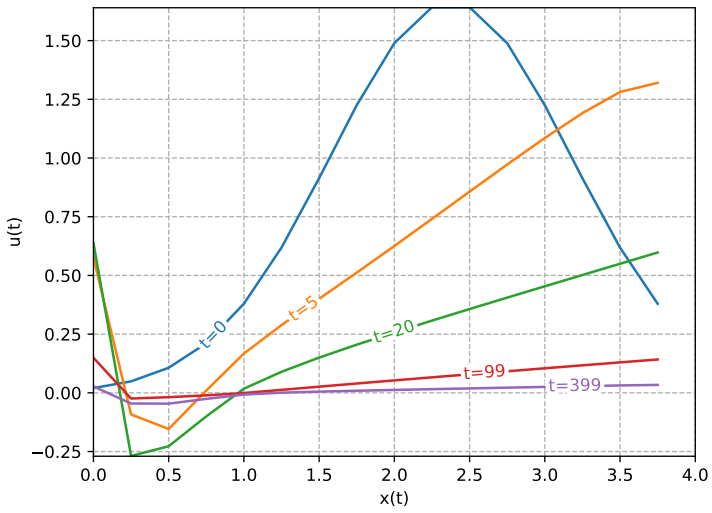
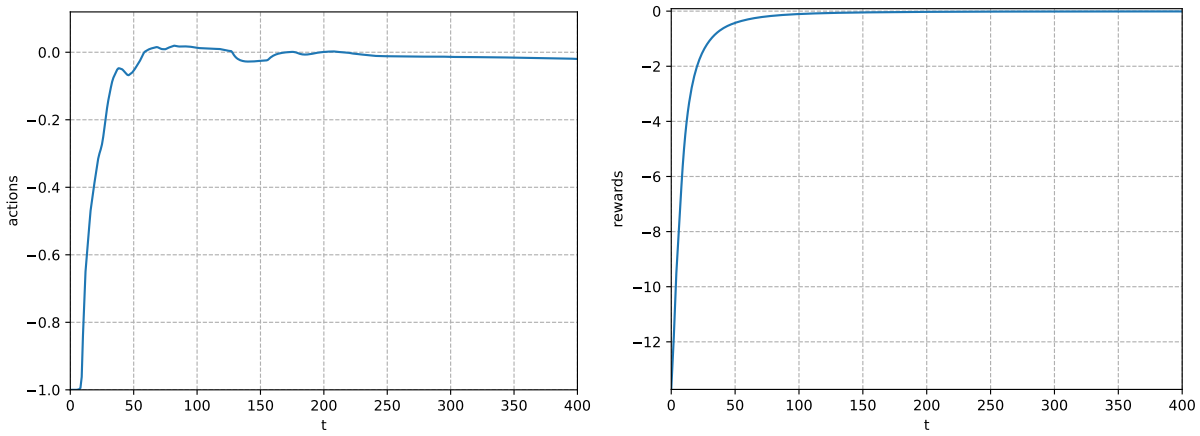


Figure 8.12: Simulation of burgers equation under actions predicted by the RL agent.

Here also, we have used the DDPG agent as in the previous experiment and trained it for 100 epochs with a learning rate of 10^{-4} for 200-time steps. Figure 8.12 shows the state trajectories resulting from the actions predicted by the RL agent. Here we have used random initial states, so the agent gets access to diverse state information during training and kept the reward function

similar to that of the baseline agent. It is clear from the figure that the RL agent can predict optimal actions and can bring the velocities of each particle closer to 0. It does so by choosing the action of maximum magnitude 1.0 in the direction opposite to the velocity value at the beginning $t = 0$ and then reducing the magnitude till $t = 50$ approximately. There-onward the agent applies a very low control value, nearly 0, and lets the system stabilise itself. In the case of a state represented by negative velocity values, our RL agent predicts positive actions and shows similar behaviour.



(a) actions (b) rewards

Figure 8.13: Actions and rewards analysis, predicted by the RL agent.

Figure 8.13 shows the rewards and the predicted actions at each time step. Here, the agent can successfully maximise the rewards from -12.0 to 0.0 in a stable manner after $t = 100$ time steps.

8.2.4 MPC agent

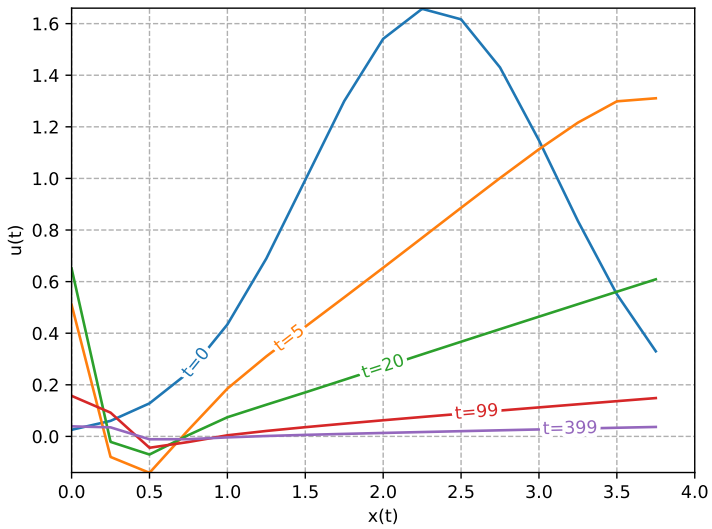
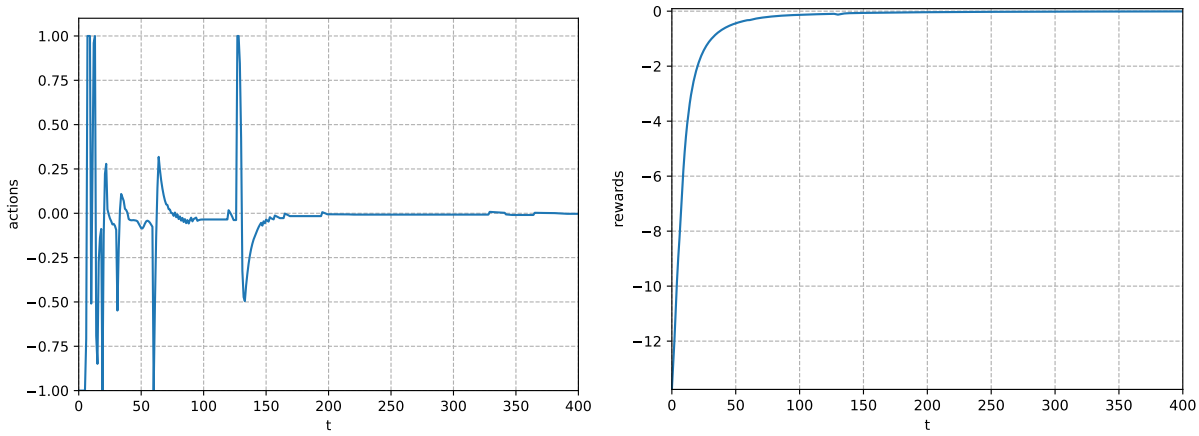


Figure 8.14: Simulation of Burgers equation using the MPC agent.

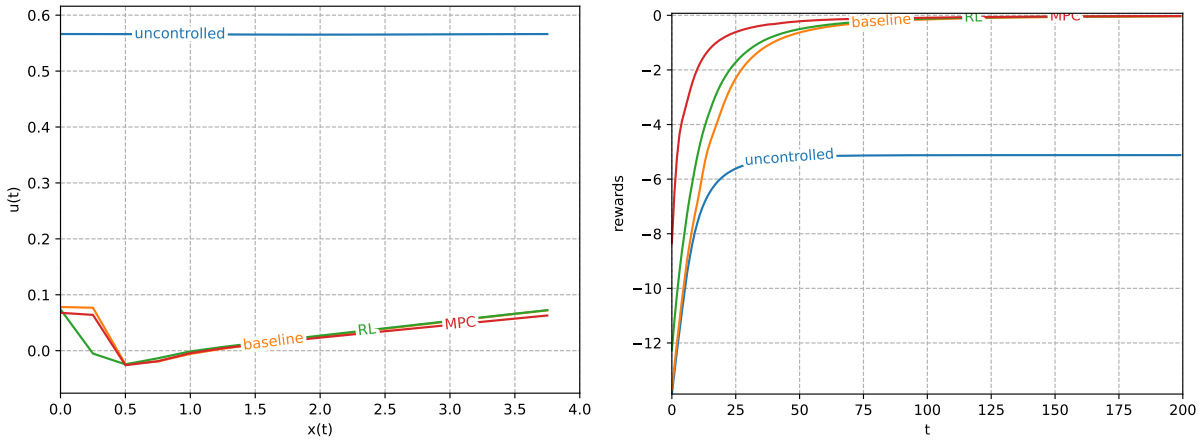


(a) actions (b) rewards

Figure 8.15: Actions as predicted by the MPC agent and respective rewards.

Figure 8.14 shows the intermediate states resulting from actions predicted by the MPC agent. The prediction horizon of 5 was used in this scenario. It is evident from the figure that the MPC agent can stabilise the equation close to 0 at $t = 400$. The actions predicted by the agent are shown in figure 8.15. The actions oscillate between positive and negative values till $t = 150$, and then the equation stabilises itself. RL agent was able to optimise this state by approximately 50 time steps 8.15. The agent here took the path of reducing the velocity with the highest magnitude and then increasing it with a similar magnitude, thereby gradually reaching a point of equilibrium.

8.2.5 Evaluation



(a) final states (b) rewards

Figure 8.16: Analysis of states and rewards, for baseline, RL and MPC agents.

Finally, comparing all three agents for ten random initial states, the resulting final states are shown in figure 8.16 (a). Like in the previous experiment, the performance of RL and MPC agents is better than the baseline, and each agent can successfully predict actions that bring the velocity of the final state closer to 0. Between $x = 0.0$ to 0.5, we can see the RL state is slightly

better than the MPC agent in choosing actions that take the state below 0, so the equation can balance itself closer to 0 in future. Figure 8.16 (b) shows the rewards predicted by each agent and captures this behaviour. Each agent can maximise the rewards to 0, but the MPC agent does this the fastest. However, this can also be interpreted as the RL agent choosing actions that took the state below 0, so the equation can be balanced in future time steps. In this scenario, the RL agent predicted that the final state was better than MPC. Here, we can improve the performance of MPC further by increasing the prediction horizon.

9 Conclusion

In this thesis, we developed a Reinforcement learning (RL) interface named PhysicsGym that helps with easy setup for conducting control experiments with Partial differential equations solver Phiflow. For this reason, we investigated different implementations of PDE control problems in the RL framework among PDE solver libraries like FEniCS, Phiflow and more. We developed the interface keeping scalability to problems with high dimensional states and action spaces in mind. To test out the performance of our interface, we implemented two experiments with control problems involving the Burgers and the Heat equation. We compared the results of the RL approach with that of a baseline agent and the Model predictive control (MPC) approach.

Our interface makes it possible to perform experiments to compare different methodologies like Model predictive control and many other techniques with that of RL with ease. The experiments showcase the successful performance of RL as compared to our baseline agent and MPC, with just a few training epochs for the control problems discussed here. However, the performance of state of the art in control, MPC, can be improved by increasing the prediction horizon such that the controller has a good idea about the quality of actions over a more extended period.

Overall our interface showed the successful application of RL in controlling very high dimensional control problems for the experiments conducted in this thesis. In future work, we propose integrating Phiflow's parallelisation capabilities into our interface with GPU support for Phiflow and training the Neural Network. This parallelisation involves making the calculations specific to Phiflow objects compatible with those that support Tensorflow. Another extension would be implementing variations of control transformation functions that support different PDE control problems.

Bibliography

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

Wolfgang Arendt and Mahamadi Warma. The laplacian with robin boundary conditions on arbitrary domains. *Potential Analysis*, 19(4):341–363, 2003.

Javier Arroyo, Carlo Manna, Fred Spiessens, and Lieve Helsen. Reinforced model predictive control (rl-mpc) for building energy management. *Applied Energy*, 309:118346, 2022. ISSN 0306-2619. doi: https://doi.org/10.1016/j.apenergy.2021.118346. URL `https://www.sciencedirect.com/science/article/pii/S0306261921015932`.

Vishak Balaji and Dr. L. Rajaji. Comparative study of pid and mpc controller using lab view. In *COMPARATIVE STUDY OF PID AND MPC CONTROLLER USING LAB VIEW*, 2013.

O. A. Bauchau and J. I. Craig. *Euler-Bernoulli beam theory*, pages 173–221. Springer Netherlands, Dordrecht, 2009. ISBN 978-90-481-2516-6. doi: 10.1007/978-90-481-2516-6_5. URL `https://doi.org/10.1007/978-90-481-2516-6_5`.

Yuri Bazilevs and Thomas JR Hughes. Weak imposition of dirichlet boundary conditions in fluid mechanics. *Computers & fluids*, 36(1):12–26, 2007.

Abraham I. Beltzer. *Variational Approach and Equations of Motion*, pages 44–88. Springer Berlin Heidelberg, Berlin, Heidelberg, 1990. ISBN 978-3-642-83914-6. doi: 10.1007/978-3-642-83914-6_2. URL `https://doi.org/10.1007/978-3-642-83914-6_2`.

Dimitri P Bertsekas. *Reinforcement learning and optimal control / by Dimitri P. Bertsekas.* Athena Scientific optimization and computation series. Athena Scientific, Belmont, Massachusetts, 2019. ISBN 9781886529397.

BN Biswas, Somnath Chatterjee, SP Mukherjee, and Subhradeep Pal. A discussion on euler method: A review. *Electronic Journal of Mathematical Analysis and Applications*, 1(2):2090–2792, 2013.

Nicolas Bondoux, Anh Quan Nguyen, Thomas Fiig, and Rodrigo Acuna-Agost. Reinforcement learning applied to airline revenue management. *Journal of Revenue and Pricing Management*, 19(5):332–348, 2020.

Kathryn Eleda Brenan, Stephen L Campbell, and Linda Ruth Petzold. *Numerical solution of initial-value problems in differential-algebraic equations*. SIAM, 1995.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016a.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016b.

Leonid Chaichenets, Dirk Hundertmark, Peer Kunstmann, and Nikolaos Pattakos. Nonlinear Schrödinger equation, differentiation by parts and modulation spaces. CRC 1173 Preprint 2018/1, Karlsruhe Institute of Technology, feb 2018. URL `https://www.waves.kit.edu/downloads/CRC1173_Preprint_2018-1.pdf`. Revised version from April 2018.

Steven C. Chapra and Raymond Canale. *Numerical Methods for Engineers*. McGraw-Hill, Inc., USA, 5 edition, 2005. ISBN 0073101567.

Francois Chollet et al. Keras, 2015. URL `https://github.com/fchollet/keras`.

Martina De Castro, Umberto Zona, and Fabio Bocci. *"L'apprendimento macchinico tra Skinner box e Deep Reinforcement Learning. Rischi e opportunità. Machine Learning between Skinner box and Deep Reinforcement Learning. Risks and opportunities", in "Dalle Teaching Machines al Machine Learning" a cura di Graziano Cecchinato, Valentina Grion - PREPRINT*, pages 27–33. 07 2020. ISBN 978-88-6938-199-7.

Saskia Dietze and Martin A. Grepl. Reduced order model predictive control for parametrized parabolic partial differential equations, 2021. URL `https://arxiv.org/abs/2111.00597`.

Charles R Doering and John D Gibbon. *Applied analysis of the Navier-Stokes equations*. Cambridge university press, 1995.

Amir-massoud Farahmand, Saleh Nabi, Piyush Grover, and Daniel N. Nikovski. Learning to control partial differential equations: Regularized fitted q-iteration approach. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, pages 4578–4585, 2016. doi: 10.1109/CDC.2016.7798966.

Bakre Fatimah, Ashiribo Wusu, and Moses Akanbi. Solving ordinary differential equations with evolutionary algorithms. *Open Journal of Optimization*, 04:69–73, 01 2015. doi: 10.4236/ojop.2015.43009.

Fei Feng, Ruosong Wang, Wotao Yin, Simon S Du, and Lin F Yang. Provably efficient exploration for rl with unsupervised learning. *arXiv preprint arXiv:2003.06898*, 2020.

Clive A. J. Fletcher. *Weighted Residual Methods*, pages 98–162. Springer Berlin Heidelberg, Berlin, Heidelberg, 1988. ISBN 978-3-642-97035-1. doi: 10.1007/978-3-642-97035-1_5. URL `https://doi.org/10.1007/978-3-642-97035-1_5`.

Stanley P Frankel. Convergence rates of iterative treatments of partial differential equations. *Mathematics of Computation*, 4(30):65–75, 1950.

Peter Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical Approach*. Wiley-IEEE Press, 2 edition, April 2015. ISBN 978-1-118-85912-4.

Giovanni Gallavotti. Navier-stokes equation: irreversibility turbulence and ensembles equivalence, 2019. URL https://arxiv.org/abs/1902.09610.

Carl Friedrich Gauss and Charles Henry Davis. Theory of the motion of the heavenly bodies moving about the sun in conic sections : a translation of theoria motus. In *Theory of the motion of the heavenly bodies moving about the sun in conic sections : a translation of Theoria motus.*, 1964.

Harvey Gould, Jan Tobochnik, and Don E. Harrison. An introduction to computer simulation methods: Applications to physical systems, part 1 and part 2. *Computers in Physics*, 2(1):90–91, 1988. doi: 10.1063/1.4822668. URL https://aip.scitation.org/doi/abs/10.1063/1.4822668.

Vijaykumar Gullapalli. Reinforcement learning and its application to control, 1992. UMI Order No. GAX92-19438.

Stefan Heid, Daniel Weber, Henrik Bode, Eyke Hüllermeier, and Oliver Wallscheid. Omg: A scalable and flexible simulation and testing environment toolbox for intelligent microgrid control. *Journal of Open Source Software*, 5(54):2435, 2020. doi: 10.21105/joss.02435. URL https://doi.org/10.21105/joss.02435.

Philipp Holl, Vladlen Koltun, Kiwon Um, and Nils Thuerey. phiflow: A differentiable pde solving framework for deep learning via physical simulations. In *NeurIPS Workshop*, volume 2, 2020.

Petros Karamanakos, Mehrdad Nahalparvari, and Tobias Geyer. Fixed switching frequency direct model predictive control with continuous and discontinuous modulation for grid-tied converters with lcl filters. *IEEE Transactions on Control Systems Technology*, 29(4):1503–1518, 2021. doi: 10.1109/TCST.2020.3008030.

Md Al-Masrur Khan, Md Rashed Jaowad Khan, Abul Tooshil, Niloy Sikder, MA Parvez Mahmud, Abbas Z Kouzani, and Abdullah-Al Nahid. A systematic review on reinforcement learning-based robotics within the last decade. *IEEE Access*, 8:176598–176623, 2020.

Bahare Kiumarsi, Kyriakos G Vamvoudakis, Hamidreza Modares, and Frank L Lewis. Optimal and autonomous control using reinforcement learning: A survey. *IEEE transactions on neural networks and learning systems*, 29(6):2042–2062, 2017.

Bahare Kiumarsi, Kyriakos G. Vamvoudakis, Hamidreza Modares, and Frank L. Lewis. Optimal and autonomous control using reinforcement learning: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, 29(6):2042–2062, 2018. doi: 10.1109/TNNLS.2017.2773458.

Qingkai Kong, Timmy Siauw, and Alexandre M. Bayen. Chapter 23 - boundary-value problems for ordinary differential equations (odes). In Qingkai Kong, Timmy Siauw, and Alexandre M. Bayen, editors, *Python Programming and Numerical Methods*, pages 399–414. Academic Press, 2021. ISBN 978-0-12-819549-9. doi: https://doi.org/10.1016/B978-0-12-819549-9.00033-6. URL https://www.sciencedirect.com/science/article/pii/B9780128195499000336.

Alexander Kuhnle, Michael Schaarschmidt, and Kai Fricke. Tensorforce: a tensorflow library for applied reinforcement learning. Web page, 2017. URL https://github.com/tensorforce/tensorforce.

Peter Lewis and David Nualart. Stochastic burgers' equation on the real line: Regularity and moment estimates, 2017. URL https://arxiv.org/abs/1709.06966.

Ang Li. Comparison between model predictive control and pid control for water-level maintenance in a two-tank system by. In *COMPARISON BETWEEN MODEL PREDICTIVE CONTROL AND PID CONTROL FOR WATER-LEVEL MAINTENANCE IN A TWO-TANK SYSTEM by*, 2010.

Thibault Liard, Raphael Stern, and Maria Laura Delle Monache. Optimal driving strategies for traffic control with autonomous vehicles. *IFAC-PapersOnLine*, 53(2):5322–5329, 2020. ISSN 2405-8963. doi: https://doi.org/10.1016/j.ifacol.2020.12.1219. URL https://www.sciencedirect.com/science/article/pii/S2405896320316177. 21st IFAC World Congress.

Yuan Lin, John McPhee, and Nasser L. Azad. Comparison of deep reinforcement learning and model predictive control for adaptive cruise control. *IEEE Transactions on Intelligent Vehicles*, 6(2):221–231, 2021. doi: 10.1109/TIV.2020.3012947.

Lennart Ljung. *System Identification*, pages 163–173. Birkhäuser Boston, Boston, MA, 1998. ISBN 978-1-4612-1768-8. doi: 10.1007/978-1-4612-1768-8_11. URL https://doi.org/10.1007/978-1-4612-1768-8_11.

Anders Logg, Garth Wells, and Kent-Andre Mardal. *Automated solution of differential equations by the finite element method. The FEniCS book*, volume 84. Springer Berlin, Heidelberg, 04 2011. ISBN 978-3-642-23098-1. doi: 10.1007/978-3-642-23099-8.

Salim A. Messaoudi and Ala A. Talahmeh. On wave equation: review and recent results. *Arabian Journal of Mathematics*, 7(2):113–145, nov 2017. doi: 10.1007/s40065-017-0190-4. URL https://doi.org/10.1007%2Fs40065-017-0190-4.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013. URL http://arxiv.org/abs/1312.5602.

Bijan Mohammadi and Olivier Pironneau. Shape optimization in fluid mechanics. *Annual review of fluid mechanics*, 36(1):255–279, 2004.

Arkadi S. Nemirovski and Michael J. Todd. Interior-point methods for optimization. *Acta Numerica*, 17:191–234, 2008. doi: 10.1017/S0962492906370018.

Michael K Ng, Raymond H Chan, and Wun-Cheung Tang. A fast algorithm for deblurring models with neumann boundary conditions. *SIAM Journal on Scientific Computing*, 21(3): 851–866, 1999.

Mario Ohlberger and Stephan Rave. Reduced basis methods: Success, limitations and future challenges. *arXiv: Numerical Analysis*, 2015.

David Padua, editor. *LU Factorization*, pages 1087–1087. Springer US, Boston, MA, 2011. ISBN 978-0-387-09766-4. doi: 10.1007/978-0-387-09766-4_2029. URL https://doi.org/10.1007/978-0-387-09766-4_2029.

Yangchen Pan, Amir-massoud Farahmand, Martha White, Saleh Nabi, Piyush Grover, and Daniel Nikovski. Reinforcement learning with function-valued action spaces for partial differential equation control. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the*

35th International Conference on Machine Learning, volume 80 of *Proceedings of Machine Learning Research*, pages 3986–3995. PMLR, 10–15 Jul 2018a. URL https://proceedings.mlr.press/v80/pan18a.html.

Yangchen Pan, Amir-massoud Farahmand, Martha White, Saleh Nabi, Piyush Grover, and Daniel Nikovski. Reinforcement learning with function-valued action spaces for partial differential equation control. *CoRR*, abs/1806.06931, 2018b. URL http://arxiv.org/abs/1806.06931.

Rahul Pandit, Prasad Perlekar, and Samriddhi Sankar Ray. Statistical properties of turbulence: An overview. *Pramana*, 73(1):157–191, Jul 2009. ISSN 0973-7111. doi: 10.1007/s12043-009-0096-6. URL https://doi.org/10.1007/s12043-009-0096-6.

Robert A Paz. The design of the pid controller. *Klipsch school of Electrical and Computer engineering*, 8:1–23, 2001.

Simon Pirkelmann. *Economic Model Predictive Control and Time-Varying Systems*. PhD thesis, Uni Bayreuth, Bayreuth, June 2020. URL https://epub.uni-bayreuth.de/4905/.

S.Joe Qin and Thomas A. Badgwell. A survey of industrial model predictive control technology. *Control Engineering Practice*, 11(7):733–764, 2003. ISSN 0967-0661. doi: https://doi.org/10.1016/S0967-0661(02)00186-7. URL https://www.sciencedirect.com/science/article/pii/S0967066102001867.

Jean Rabault and Alexander Kuhnle. Accelerating deep reinforcement learning strategies of flow control through a multi-environment approach. *Physics of Fluids*, 31(9):094105, Sep 2019. ISSN 1089-7666. doi: 10.1063/1.5116415. URL http://dx.doi.org/10.1063/1.5116415.

Jean Rabault, Miroslav Kuchta, Atle Jensen, Ulysse Ré glade, and Nicolas Cerardi. Artificial neural networks trained through deep reinforcement learning discover control strategies for active flow control. *Journal of Fluid Mechanics*, 865:281–302, feb 2019. doi: 10.1017/jfm.2019.62. URL https://doi.org/10.1017%2Fjfm.2019.62.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.

Saa V. Rakovi and William S. Levine. *Handbook of Model Predictive Control*. Birkhäuser Basel, 1st edition, 2018. ISBN 3319774883.

James B. Rawlings, David Angeli, and Cuyler N. Bates. Fundamentals of economic model predictive control. In *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, pages 3851–3861, 2012. doi: 10.1109/CDC.2012.6425822.

Bill Rehm, Drilling Consultant, Arash Haghshenas, Amir Saman Paknejad, and Jerome Schubert. Chapter two - situational problems in mpd. In Bill Rehm, Jerome Schubert, Arash Haghshenas, Amir Saman Paknejad, and Jim Hughes, editors, *Managed Pressure Drilling*, pages 39–80. Gulf Publishing Company, 2008. ISBN 978-1-933762-24-1. doi: https://doi.org/10.1016/B978-1-933762-24-1.50008-5. URL https://www.sciencedirect.com/science/article/pii/B9781933762241500085.

Max Schwenzer, Muzaffer Ay, Thomas Bergs, and Dirk Abel. Review on model predictive control: an engineering perspective. *The International Journal of Advanced Manufacturing Technology*, 117(5):1327–1349, Nov 2021. ISSN 1433-3015. doi: 10.1007/s00170-021-07682-3. URL https://doi.org/10.1007/s00170-021-07682-3.

David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 387–395, Bejing, China, 22–24 Jun 2014. PMLR. URL https://proceedings.mlr.press/v32/silver14.html.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, Jan 2016. ISSN 1476-4687. doi: 10.1038/nature16961. URL https://doi.org/10.1038/nature16961.

James Stewart. *Calculus : early transcendentals*. Brooks/Cole, Cengage Learning, Belmont, Cal., 2012. ISBN 0538497904 9780538497909 0840058853 9780840058850 0538498714 9780538498715 0538498870 9780538498876 0840048254 9780840048257.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL http://incompleteideas.net/book/the-book-2nd.html.

Hongwei Tang, Jean Rabault, Alexander Kuhnle, Yan Wang, and Tongguang Wang. Robust active flow control over a range of reynolds numbers using an artificial neural network trained through deep reinforcement learning. *Physics of Fluids*, 32(5):053605, May 2020. ISSN 1089-7666. doi: 10.1063/5.0006492. URL http://dx.doi.org/10.1063/5.0006492.

Nils Thuerey, Philipp Holl, Maximilian Mueller, Patrick Schnell, Felix Trost, and Kiwon Um. Physics-based deep learning. *arXiv preprint arXiv:2109.05237*, 2021.

Shiyin Wei, Xiaowei Jin, and Hui Li. General solutions for nonlinear differential equations: a rule-based self-learning approach using deep reinforcement learning. *Computational Mechanics*, 64 (5):1361–1374, May 2019. ISSN 1432-0924. doi: 10.1007/s00466-019-01715-1. URL http://dx.doi.org/10.1007/s00466-019-01715-1.

David Vernon Widder. *The heat equation*, volume 67. Academic Press, 1976.

Wikipedia contributors. Convection–diffusion equation — Wikipedia, the free encyclopedia, 2022. URL https://en.wikipedia.org/w/index.php?title=Convection%E2%80%93diffusion_equation&oldid=1068635760. [Online; accessed 24-April-2022].

List of Figures