REINFORCEMENT LEARNING FOR THE OPTIMAL CONTROL OF DYNAMICAL SYSTEMS

THEÏLO TERRISSE

Abstract. The theory of optimal control is interested in controlling a dynamical system throughout a temporal evolution so as to optimize an objective quantified by a performance index. Many numerical methods are derived from this theory and find applications in many real-world problems. Yet, several reasons can bring these methods to fail, among which three limitations are identified: their attachment to an explicit mathematical model, their computational cost making them intractable in real-time, and their lack of robustness in complex non-linear cases. With this last limitation comes the fact that sufficient conditions are known for the optimality of a control in non-linear problems, but can hardly be exploited due to the so-called *curse of dimensionality*. This motivates the resort to dynamic programming and reinforcement learning, which we introduce in this memoir through a simple but foundational algorithm, Q-learning. Then, methods from reinforcement learning are implemented and we show how they adapt to time-continuous optimal control problems: the reinforcement learning agent interacts with the system (here implemented as a numerical simulator) in a time-discrete manner, receiving observations from the state and inputting actions at each time step. The cost function is discretized in time to build rewards consumed by the agent. This approach is compared with ones from optimal control to illustrate how it can serve as an alternative to address the aforementioned difficulties of optimal control: reinforcement learning can work *model-free* to avoid the resort to an explicit model; it alleviates the curse of dimensionality thanks to what is designated as the *exploration-exploitation dilemma*; and it offers cheap computations for real-time problems, at the expense of painstaking training procedures, in a *pay-offline-win-online* manner. Lastly, a first step into non-linear problems is made with the control of the Burgers' equations for fluid dynamics.

Acknowledgement

I want to address my sincere gratitude to the Chair for Dynamics, Control, Machine Learning and Numerics *in the* Department of Mathematics *at* FAU Erlangen-Nürnber *for offering this internship opportunity. This work was generously funded by the* Alexander von Humboldt *fellowship.*

I am also warmly thankful to my two supervisors Prof. Enrique Zuazua, head of the chair, and Nicola De Nitti. Their active supervision, the hindsight, the advice they shared with me, and the help they provided in the writing of this memoir have constituted precious support.

Lastly, I am grateful to the entire scientific team of the chair for their hospitality throughout the 6 months of the placement, and for the enriching discussions I have had the chance to share with them regarding their respective works, but also regarding the world of academic research in general.

Introduction

The work presented here is at the boundary between two sister mathematical fields: *optimal control* and *reinforcement learning*. Both disciplines address similar problems, namely those of controlling systems by acting upon them so as to achieve goals that can be quantified by some numerical performance index. Although the two fields have tightly intertwined histories, optimal control methods mostly inherit from well-established theoretical results in the lineage of calculus of variations and the theory of continuous time dynamical systems, that may be applied to a precise mathematical model of the system. Reinforcement learning, on the other hand, is relatively newer and is in the continuity of the machine learning research that drew inspiration from the study of biological intelligence in the 20^{th} century. It is heavily data-driven and offers a range of general algorithms that apply to many different contexts.

The objective of this memoir is mostly two-fold: to give an *introduction to reinforcement learning* (RL), its main concepts and its theoretical link to optimal control; and to explore through illustrative implementations how this relatively novel methodology can circumvent some *known limitations of optimal control*. We start by quickly recalling historical aspects of optimal control, numerical methods derived from it and their limitations. *Hamilton-Jacobi-Bellman* equations are emphasized as they make an entry-point to link the optimal control of continuous dynamical processes with the discrete processes of RL. In

Internship memoir written under the supervision of Prof. Dr. Enrique Zuazua and Nicola De Nitti at FAU Erlangen Nürnberg (Jan.–Jul., 2023).

a second section, the latter is presented and over-viewed, and time is spent on the historically important Q-learning algorithm to highlight some stark ideas of RL. The third section combines the two domains (optimal control and reinforcement learning) on two variants of a linear optimal problem, that of bringing a cart from a random initial position to a target position by pushing it along a 1D-axis. Using a numerical simulator for the system, the problem is solved using the adjoint method on the one hand, then using Q-learning and the proximal policy optimization algorithm on the other hand. We describe how the problem is moved into an RL framing through the discretization of the dynamics and cost function in time. This implementation fuels discussion on the advantages of each approach, as we illustrate that reinforcement learning can *work without an explicit mathematical model* of the system's dynamics, that it *circumvents the computational intractability* of dynamic programming and that it offers controls that are *cheap to evaluate* at the expense of *heavy training procedures*. Lastly, in Section 4 we start addressing harder, non-linear problems by applying reinforcement learning on a control problem from fluid dynamics, that of applying a forcing term on a 1-dimensional fluid ruled by the *Burgers* equation so as to bring a velocity field from an initial state to a target field.

Notation. *Throughout this document, unless stated otherwise:*

- *scalars and vectors are denoted in lowercase letters:* x, y;
- *scalar- and vector-valued functions of time representing a state or action trajectory are denoted in bold lowercase letters:* $\mathbf{x}$, $\mathbf{y}$;
- *time series (and vectors interpreted as such) are denoted in teletype font:* $\mathtt{x}, \mathtt{y}$;
- *matrices are denoted by upright capital letters:* A;
- *if* y *is a column vector,* y^{T} *is the corresponding row vector;*
- A^{T} *is the transpose of* A;
- $\dot{\mathbf{x}}$ *and* $\ddot{\mathbf{x}}$ *are the first and second (full) derivatives in time of* $\mathbf{x}$;
- ∂_s *denotes a partial derivative with respect to a variable* s;
- $\langle x, x' \rangle$ *is the inner product between* x *and* x'.

1. Control theory and its boundaries

1.1. **Definition and historical rudiments.** Control theory is interested in applying a control over a system ruled by given dynamics, typically so as to bring it from an initial state to a desired one. Let us recall how such a problem can be formulated mathematically, in the case where its dynamics are ruled by an ordinary differential equation (ODE).

Definition 1.1 (Control system)**.** *A dynamical system is described by its* state $x \in X$ *(where* X *is a state space that must be specified) which evolves over a time period* $[0, T]$ *(where* $T > 0$*), starting in* $x_0 \in X$ *and according to the dynamics*

$$\begin{cases} \dot{\mathbf{x}}(t) = f(t, \mathbf{x}(t), \mathbf{u}(t)), \quad \forall t \in [0, T], \\ \mathbf{x}(0) = x_0. \end{cases} \tag{1.1}$$

Here, $\mathbf{x} : [0, T] \to X$ *describes the* trajectory *of the system, that is its state at every point in time,* $f : [0, T] \times X \times U$ *describes the* dynamics *and* $\mathbf{u} : [0, T] \to U$ *(where* U *is an action space that must be specified) is the* control*, that is the action exerted over time. The space of trajectories* $\mathbf{x}$ *will be denoted* $\mathcal{X}$*, and the space of controls* $\mathbf{u}$ *will be denoted* $\mathcal{U}$.

Control theory tries to answer the question of *controllability*:

Definition 1.2. *System* (1.3) *is said to be* controllable *in time* T *from* x_0 *if and only if*

$$\forall x^* \in X, \ \exists \mathbf{u} \in \mathcal{U}, \ \mathbf{x}(T) = x^*.$$

Optimal control is then interested in choosing a control that minimizes some *cost functional*.

Definition 1.3 (Optimal control problem)**.** *In an optimal control problem, we want to*

$$\begin{aligned} \underset{\mathbf{u} \in \mathcal{U}}{\text{minimize}} \ J(\mathbf{x}, \mathbf{u}) &= C_f(\mathbf{x}(T)) + \int_0^T C(\mathbf{x}(t), \mathbf{u}(t)) \\ \text{s.t.} \ &\begin{cases} \dot{\mathbf{x}}(t) = f(t, \mathbf{x}(t), \mathbf{u}(t)), \quad \forall t \in [0, T], \\ \mathbf{x}(0) = x_0, \end{cases} \end{aligned} \tag{1.2}$$

where $C : X \times \mathcal{U} \to \mathbb{R}$ *is called the* running cost *and* $C_f : X \to \mathbb{R}$ *is called the* terminal cost*. A minimizer of* J *is called an* optimal control.

Optimal control lies in the lineage of calculus of variations, from which it notably inherited the *principle of maximum*, that can be used to derive conditions for the optimality of a control in (1.2). This principle was exploited to derive a necessary optimality condition by Pontryagin in 1962 [19], and a sufficient optimality condition by Bellman in 1957 [3]. The latter inscribed his research in the continuity of works from Hamilton and Jacobi in the 19$^{\text{th}}$ century to formulate the Hamilton-Jacobi-Bellman (HJB) equations.

As discussed further below, a majority of modern numerical approaches to optimal control build upon these foundational contributions, complemented with results from various other mathematical sub-fields. Among them, classical control theory offers engineers the tools to design closed-loop feedback control by tuning the parameters of the model and of the cost function; while (non-)linear programming can be leveraged after some clever discretization of the dynamics' equations.

More details on the history of optimal control can be found in the short but excellent 1996 overview by Arthur E. Bryson [6].

1.2. **Control theory: a sound framing for time-continuous linear control.** A well-studied case in control theory is that of linear dynamics.

Definition 1.4 (Linear control system)**.** *A control system is said to be* (autonomous) linear *if it is of the form*

$$\begin{cases} \dot{\mathbf{x}}(t) = \mathrm{A}\mathbf{x}(t) + \mathrm{B}\mathbf{u}(t), \quad \forall t \in [0,T], \\ \mathbf{x}(0) = x_0, \end{cases} \tag{1.3}$$

where $T > 0$, $\mathbf{x} : [0,T] \to X = \mathbb{R}^n$ *with* $n \in \mathbb{N}^*$, $\mathbf{u} : [0,T] \to \mathbb{R}^p$ *with* $p \in \mathbb{N}^*$, $\mathrm{A} \in \mathbb{R}^{n\times n}$ *and* $\mathrm{B} \in \mathbb{R}^{n\times p}$.

Of course, $\mathbf{u}$ should be searched in a space $\mathcal{U}$ such that for every $\mathbf{u} \in \mathcal{U}$ there exists a unique solution $\mathbf{x}$ to the system (1.3); for instance by taking $\mathcal{U} = L^1([0,T], \mathbb{R}^p)$. A foundational result in this framing is a necessary and sufficient condition for controllability given by Kálmán in 1960 [14].

Theorem 1.1 (Kálmán rank condition)**.** *System* (1.3) *is controllable in any time* T *and from any* x_0 *if and only if:*

$$\operatorname{rank}(\mathrm{B}, \mathrm{AB}, \dots, \mathrm{A}^{n-1}\mathrm{B}) = n. \tag{1.4}$$

This important results gives a lot of credit to the theory of control, because it supplies a simple condition to ensure that a system will be controllable, even in the case where $n >> p$.

However, the controllability of a system becomes much harder when the underlying dynamics are non-linear, which is the case for most real systems. In this case, a common approach is to locally linearize the system around trajectories of interest, for example in order to stabilize it around an equilibrium, but the resulting controls might not be robust, especially if the system is highly non-linear. Furthermore, Many non-linear ODEs have solutions for which no explicit analytical expression is known, hence the need for numerical methods.

1.3. **Optimal control in the non-linear case.** The past century has also brought some very general results for the non-linear case. From the theoretical perspective, among them, Pontryagin's maximum principle only gives necessary conditions for a control to be optimal for problem 1.3, and the Hamilton-Jacobi-Bellman equations give necessary and sufficient conditions.

The latter equation is part of a field known as *dynamic programming* and was introduced by R. Bellman in the 1950s. As this will serve as an entry point to reinforcement learning from control theory, we here recall this result in some details. Here, we take $X = \mathbb{R}^n$ ($n \in \mathbb{N}^*$) and U is a compact subset of $\mathbb{R}^p$, $p \in \mathbb{N}^*$.

H1. *In the control system 1.1, we assume that* f *is only dependent on time through* $\mathbf{x}$ *and* $\mathbf{u}$ *(*$f(t, \mathbf{x}(t), \mathbf{u}(t)) = f(\mathbf{x}(t), \mathbf{u}(t))$*) and that it is smooth enough to ensure the existence of an absolutely continuous solution* $\mathbf{x}$ *to the ODE.*

Definition 1.5 (Value function)**.** *The* value function *starting from position* $s \in X$ *at time* $t \in [0,T]$ *associated to* $\mathbf{u} \in \mathcal{U}$ *is defined by:*

$$\begin{aligned} V^{\mathbf{u}}(s,t) &= \int_t^T C(\mathbf{x}(\tau), \mathbf{u}(\tau))\,\mathrm{d}\tau + C_f(\mathbf{x}(T)) \\ &\text{s.t. } \begin{cases} \dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t)), \quad \forall t \in [0,T], \\ \mathbf{x}(0) = s \end{cases} \end{aligned} \tag{1.5}$$

This function represents accumulated cost over the trajectory starting from s *at time* t *up to time* T*, when applying the control* u*. We also define the* optimal value function *by*

$$V^*(s,t) = \inf_{\mathbf{u}\in\mathcal{U}} V^{\mathbf{u}}(s,t). \tag{1.6}$$

The main point of dynamic programming is that if the optimal value function is known, then (at least under smoothness conditions on V^*) an optimal control u^* can be derived easily. All the difficulty lies in finding the value function. Now, a first hint for finding this function is *Bellman's principle of optimality*: "An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision" [3, page 83]. In other words, the optimal value function can be deduced from a backward-time recursion:

Theorem 1.2 (Optimality conditions)**.**

$$\forall t < T,\ \forall s \in X,\ \forall h \in [0, T-t],\ V^*(s,t) = \inf_{\mathbf{u}\in\mathcal{U}} \left\{ \int_t^{t+h} C(\mathbf{x}(\tau), \mathbf{u}(\tau))\, \mathrm{d}\tau + V^*(\mathbf{x}(t+h), t+h) \right\} \quad \text{s.t. } \begin{cases} \dot{\mathbf{x}}(\tau) = f(\mathbf{x}(\tau), \mathbf{u}(\tau)), & \forall \tau \in [t, T], \\ \mathbf{x}(t) = s \end{cases} \tag{1.7}$$

In essence, we now wish to make h go to zero in this theorem, so as to express this condition as a partial differential equation (PDE) on V^*.

H2. *We assume that there exists $L > 0$ such that $\forall x, y \in X, u \in U$:*

$$|C(x,u)|, |C_f(x)| \leq L,$$
$$|C(x,u) - C(y,u)|, |C_f(x) - C_f(y)| \leq L|x-y|.$$

Theorem 1.3. *The value function V^* is the unique* viscosity solution *for the* terminal-*value problem for the HJB equation:*

$$\begin{cases} \partial_t V^* + \min_{u\in U} \{C(x,u) + \langle f(x,u), \partial_x V^* \rangle\} = 0 & \text{for } (x,t) \in X \times (0,T) \\ V^*(x,T) = C_f(x) & \text{for } x \in X, \end{cases} \tag{1.8}$$

where $\partial_x V^* = (\partial_{x_1} V^*, \partial_{x_2} V^*, \dots, \partial_{x_n} V^*)^{\mathrm{T}}$.

Note that V^* is a viscosity solution of the equation, which is a weak form of solution, but it can be interpreted in a strong sense if V^* is smooth enough. Further details on this can be found *e.g.* in [9].

One of the main drawbacks of dynamic programming is the *curse of dimensionality*: the PDE (1.8) is $(n+1)$-dimensional, with n the dimension of the state space, and this dimension can grow rapidly depending on the system being controlled (*e.g.* think of controlling the propagation of a disease in a population, where each inhabitant would be a state dimension). Now, for a state of dimension 3, such a PDE is already immensely computational to solve, making such a method mostly intractable.

This is why most modern numerical methods turn to more computationally-efficient approaches to tackle optimal control problems. These are generally split into two categories.

(1) Indirect methods iteratively solve forward ODEs on the state $\mathbf{x}$ and backward ODEs on the co-state $\mathbf{p}$ (which can be thought of as the Lagrangian multiplier associated to the problem 1.2), so as to exploit necessary conditions of optimality. A typical example is the adjoint method illustrated in Section 3.3.
(2) Direct methods, on the other hand, start by parameterizing the state and control functions (for example using Lagrange polynomials). This operation projects the functions on spaces of finite dimensions, which converts the optimal control problem 1.2 into a (non-)linear optimization problem that can be solved using one of the numerous optimization solvers that have been designed and polished over the last decades. These approaches are usually easier to implement, but can at most give a sub-optimal solution due to the parameterization that projects the state and control on a subset of the solution space.

These methods are generally more computationally efficient than dynamic programming, but they only exploit necessary conditions and may converge only towards a local minimum of the cost functional.

1.4. **Discussion on the limitations of optimal control.** Let us close this section by summing up some limitations of solving methods from optimal control:

(1) We have already mentioned that in the non-linear case, necessary and sufficient conditions are given by the Hamilton-Jacobi-Bellman equation, but the latter is usually intractable due to the curse of dimensionality.
(2) Most other numerical methods cannot guarantee to give the optimal control in the general, non-convex case.

(3) In many situations, one wants to get *closed-loop controls* (*i.e.* with feedback in the form of observations of the current state). Typically, when facing real-time problems, the initial and target states might not be known in advance, and the evolution of the system might involve a share of randomness. In these cases, decisions must be taken quickly, something the aforementioned methods can fail to provide due to their computational expense.
(4) Last but not least, all of these approaches require that a precise mathematical model of the system is given. Yet, most real systems are black boxes and their identification can be very tedious, require a lot of data and computation and sometimes never achieve the amount of precision required to get the desired outcomes.

With the relatively recent advent of machine learning techniques, much investigation has been carried in the field of data-driven approaches in an attempt to solve some of the difficulties listed here. Reinforcement learning is among these techniques. In this memoir, we ultimately hope to illustrate whether this approach can be used to find a compromise between obstacles 1 and 2 above by addressing increasingly non-linear problems and comparing reinforcement learning with the methods introduced so far (later referred to as "conventional methods"). However, difficulties 3 and 4 will also be discussed along the way.

2. Introduction to reinforcement Learning

2.1. **A first definition and intuition.** *Reinforcement learning* (RL) is the subfield of machine learning that studies how to learn from interaction with a dynamical system to enhance future manipulation of this system so as to maximize a reward over time.

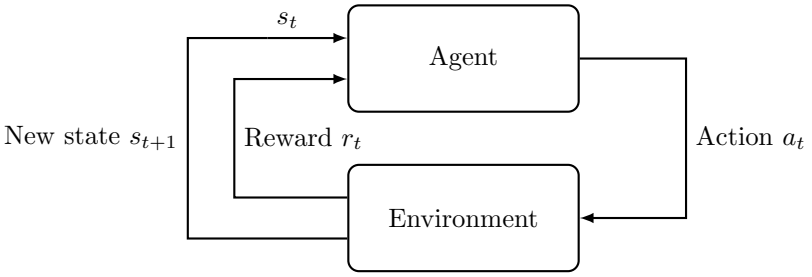


Figure 1. The agent–environment interaction in reinforcement learning. [23, Fig. 3.1].

More precisely, RL encompasses all problems (and methods to solve them) involving an *agent* (sometimes called a *controller*, or an *actor*) interacting with an *environment* over time as illustrated by Figure 1:

(1) at time t, the environment is in a state s_t;
(2) from a (potentially partial) observation of this state, the agent takes an action a_t over the environment;
(3) between times t and $t+1$, the environment sends a reward r_t to the agent and moves from state s_t to state s_{t+1} depending on the action a_t;
(4) this process is repeated from step (1), indefinitely or until a time-horizon T is reached.

Given this dynamical system, the aim of the problem is to define an agent so as to maximize the cumulative reward over time $\sum_{t=0}^{T-1} r_t + r_f$. This is why RL is often called a *semi-supervised* learning approach, in the sense that it does not train from labeled instances of a problem but the learning process is still guided by the reward.

Like many other machine learning domains, RL is strongly analogous to the way biological systems behave. For instance, the above process is similar to what a human would do when empirically learning to drive a car: a driver senses the position of the car and of surrounding obstacles, takes actions on the car through steering, gearing and pressing pedals and observes changes in the environment, getting positive feedback if the car evolved in the intended way. At first, the learner might take wrong actions resulting in a clumsy control of the car and to negative feedback. By trial-and-error and by observing the effect of each action on the environment, the driver eventually understands what action to take in what circumstances.

As well summarized in the introduction of [23, chapter I.3], an RL system is characterized by four prominent features: the *policy* (*i.e.* the mapping that matches a state of the system to an action), the *reward signal*, the *value function* and, optionally, the *model* of the environment. These elements are formulated more rigorously in Section 2.3.

2.2. **Historical development of RL.** [23, chapter 7] gives a historical overview of the development of RL, especially from a trial-and-error perspective. This review highlights an interesting aspect of reinforcement learning, namely that it inherits the work of two historically distinct branches: dynamic programming on the one hand, and psychology of animal learning on the other hand. The first gives sound and extensive mathematical foundations for exactly or approximately solving problems involving systems described as Markov decision processes (MDPs).

The second has been a source of inspiration and intuition for important threads of reinforcement learning theory. From dynamic programming, whose roots have been briefly discussed in Section 1, RL strongly benefited from research on approximate dynamic programming that has been carried out to fight the curse of dimensionality. The other branch, that of psychology and learning, bears the important notion of trial-and-error, and later that of temporal difference. Learning by trial-and-error is the idea that an intelligent system (such as an animal) can learn to solve a problem by attempting to solve it several times and memorizing what leads to positive outcomes and what does not. It can be dated back to Thorndike's "Law of effect" in 1911 [24], followed by Pavlov's notion of "reinforcement" in animals behaviour around 1927 [18]. Later, in his report from 1961, Minsky highlighted important questions at the heart of modern reinforcement learning [17], notably the "*basic credit-assignment problem for complex reinforcement learning systems*": How do you distribute credit for success among the many decisions that may have been involved in producing it? Another important work is that from Sutton and Barto on temporal difference [22]. Temporal difference is a technique for estimating the value function (the same as that defined in 1.5). It is based on the idea that when taking an action a from a state s, rewards received immediately after taking that action should have more influence on the estimation of $V(s,a)$ than rewards further in time. This is a fundamental idea in reinforcement learning.

The three threads (*dynamic programming*, *trial-and-error learning* and *temporal difference*) were united with Watkins' development of the *Q-learning* algorithm in 1989 [28] (detailed further below). This was followed by an extensive work on approximate dynamic programming and the integration of MDPs in reinforcement learning. See for example Bertsekas and Tsitsiklis' work on "neurodynamic programming" [5]. Since then, many algorithms have been proposed and recent work has been carried out to systematize their testing and comparison on classical benchmark problems.

The literature comprises many examples of problems that can be addressed by RL, namely, cooling data centers [15]; controlling and stabilizing a quadrotor [11]; portfolio management [13]; medical imaging [30, Section 4]; operation research [16]; controlling velocity fields in fluid dynamics [10].

2.3. **Formalism for reinforcement learning.** As announced, RL borrows much of its formalism from discrete dynamic programming. The latter is an algorithmic method that aims at minimizing a cost throughout the evolution of a dynamical system by breaking it down into simpler sub-problems in a recursive manner, similarly to what was presented in Section 1.3. We here introduce this formalism so as to introduce important notions and clarify the link between optimal control and RL. Note that the RL formalism accounts for the fact that the evolution may be stochastic, which makes a stark difference with HJB equations that cannot be easily adapted to stochastic problems.

Definition 2.1 (Stochastic dynamical system)**.** *The finite sequence $(s_t)_{t=0,\dots,T}$ is called a* stochastic dynamical system *if*

$$s_{t+1} = f_t(s_t, a_t, w_t), \quad \forall t \in \{0, \dots, T-1\}, \tag{2.1}$$

where T is the time-horizon *(which will here be considered $< +\infty$) and, for $t \in \{0, \dots, T-1\}$, we define:*

- *$s_t \in S_t$ $(\forall t \in \{0, \dots, T\})$, where S_t is the (finite or infinite) non-empty set of possible states at step t;*
- *$a_t \in \mathcal{A}_t(s_t)$ where $\mathcal{A}_t(s_t)$ is a subset of $\mathcal{A}_t$, the (finite or infinite) non-empty set of actions that can be taken at step t and which can depend on the current state;*
- *$w_t \in \mathcal{D}_t$ is a random variable, termed "random disturbance";*
- *$f_t : S_t \times \mathcal{A}_t \times \mathcal{D}_t \to S_{t+1}$ is the dynamics of the system.*

In almost every instance of an RL problem, the dynamical system is assumed to be an MDP, meaning that the evolution of the system from time t does not depend on its history before t.

Definition 2.2 (Markov Decision Process)**.** *The stochastic dynamical system of definition 2.1 is called a* Markov Decision Process *if,* $\forall t \in \{t = 0, \dots, T-1\}$, $\forall w \in \mathcal{D}_t$, $\forall (\tilde{s}_0, \dots, \tilde{s}_t) \in S_0 \times \dots \times S_t$, $\forall (\tilde{a}_0, \dots, \tilde{a}_t) \in \mathcal{A}_0 \times \dots \times \mathcal{A}_t$,

$$\mathbb{P}(w_t = w | s_0 = \tilde{s}_0, \dots, s_t = \tilde{s}_t; a_0 = \tilde{a}_0, \dots, a_t = \tilde{a}_t) = \mathbb{P}(w_t = w | s_t = \tilde{s}_t; a_t = \tilde{a}_t). \tag{2.2}$$

Definition 2.3 (Policy)**.** *To decide what actions to take as the system evolves, we give ourselves a policy, which is a series of functions* $\pi = (\pi_0, \pi_1, \dots, \pi_{T-1})$ *where* $\forall t \in \{0, \dots, T-1\}$, $\pi_t : S_t \to \mathcal{A}_t$. *In other words, at each time step* t, π_t *tells what action to take depending on the state* s_t.

Definition 2.4 (Value function)**.** *Let* $\forall t \in \{0, \dots, T-1\}$, $r_t : S_t \times \mathcal{A}_t \to \mathbb{R}$ *and let* $r_f : S_T \to \mathbb{R}$ *be the running and terminal rewards. For a given policy* π, *for all* $t \in \{0, \dots, T\}$ *and* $s \in S_t$, *we define the* value function *associated to* π *as follows:*

$$\begin{aligned} V^\pi(s,t) = \underset{w_t, w_{t+1}, \dots}{\mathbb{E}} & \left[\sum_{\tau=t}^{T-1} r_\tau(s_\tau, \pi_\tau(s_\tau)) + r_f(s_T) \right] \\ s.t. & \begin{cases} \forall t \le \tau < T, \ s_{\tau+1} = f_\tau(s_\tau, \pi(s_\tau), w_\tau), \\ s_t = s. \end{cases} \end{aligned} \tag{2.3}$$

Then the optimal value function is defined by

$$V^*(s,t) := \sup_\pi V^\pi(s,t). \tag{2.4}$$

The objective of RL is to find an optimal policy, that is a policy that maximizes $V^\pi(s,0)$ for all $s \in S_0$. Instead of a computationally unachievable exhaustive search, the spirit of the dynamic programming algorithm is to use the result of a "tail problem" (starting at time t) to resolve a larger problem (starting at time $t-1$), similarly to Bellman's principle of optimality stated in sub-Section 1.3. More precisely, the HJB equations become the following Bellman equation (see [4, chapter 2.2]).

Theorem 2.1 (Bellman equation)**.** *Assuming that the dynamical system is an MDP, the optimal value function* V^* *satisfies* $\forall s \in S_T$,

$$V^*(s,T) = r_f(s) \tag{2.5}$$

and $\forall t \le T-1, \ \forall s \in S_t$,

$$V^*(s,t) = \sup_{a \in \mathcal{A}_t(s_t)} \mathbb{E}_{w_t} \left[r_t(s_t, a_t) + V^*(f_t(s_t, a_t, w_t), t+1) | \ s_t = s, a_t = a \right]. \tag{2.6}$$

If the optimal value function is known, an optimal policy (if it exists) can be derived by taking the *greedy policy* associated to V^*.

Definition 2.5 (Greedy policy)**.** *The* greedy policy *associated to a value-function* V *is defined by* $\forall s \in S, \ \forall t \in \{0, ..., T-1\}$,

$$\pi_{\text{greedy}}(s,t) = \arg\max_a \mathbb{E}_{w_t} \left[r_t(s_t, a_t) + V(f_t(s_t, a_t, w_t), t+1) | \ s_t = s, a_t = a \right]. \tag{2.7}$$

Remark 2.1. *Here we introduced RL in an* episodic *framing, meaning that* T *is finite. An* episode *is one instance of the evolution of the process from time* 0 *to time* T. *Sometimes, the process never ends* ($T = +\infty$). *In this case, we assume that the evolution is* stationary *(meaning that the state space, action space, reward, policy and value functions don't depend on the time step but only on the state) and the definition 2.4 becomes:*

$$\begin{aligned} V^\pi(s) = \underset{w_0, w_1, \dots}{\mathbb{E}} & \left[\sum_{\tau=0}^{\infty} \gamma^\tau r_\tau(s_\tau, \pi(s_\tau)) \right] \\ s.t. & \begin{cases} \forall \tau \ge t, \ s_{\tau+1} = f_\tau(s_\tau, \pi(s_\tau), w_\tau), \\ s_0 = s. \end{cases} \end{aligned} \tag{2.8}$$

In this new definition, each reward is discounted exponentially in time according to a factor $\gamma \in (0,1)$ *to ensure the convergence of the sum and the existence and uniqueness of the value function. However, this factor can even be used in the case of a finite horizon with the following interpretation: for a biological system, an immediate reward tends be preferred over the same reward later in time. The Bellman equation now writes:*

$$V^*(s) = \sup_{a \in \mathcal{A}(s)} \mathbb{E}_w \left[r(s,a) + \gamma V^*(f(s,a,w)) \right], \quad \forall s \in S. \tag{2.9}$$

Remark 2.2 (Link between HJB and Bellman equations)**.** *Let us end this section by informally highlighting the link between the HJB equation* (1.8) *and the Bellman equation* (2.6)*, in the finite-horizon, deterministic case.*

Assume the dynamical system is of the form

$$s_{n+1} = s_n + h f(s_n, a_n). \tag{2.10}$$

Here, $h > 0$ denotes the length of the time step, meaning that one step for this discrete dynamical system corresponds to stepping from time t_n to time $t_{n+1} = t_n + h$. Also, to interpret V^* as continuous in time, we express the reward in proportion to the time step. Given $t \in [0, T]$ and $s \in S_t$, the Bellman equation is

$$V^*(s,t) = \sup_{a \in \mathcal{A}_t(s)} \{h r_t(s,a) + V^*(s + hf(s,a), t+h)\}.$$

This yields

$$\frac{V^*(s,t) - V^*(s,t+h)}{h} = \sup_{a \in \mathcal{A}_t(s)} \left\{ r_t(s,a) + \frac{V^*(s+hf(s,a),t+h) - V^*(s,t+h)}{h} \right\}.$$

Now, if V^* is smooth enough,

$$V^*(s+hf(s,a),t+h) = V^*(s,t) + h\langle f(s,a), \partial_s V^*(s,t)\rangle + h\partial_t V^*(s,t) + o(h)$$

and

$$V^*(s,t+h) = V^*(s,t) + h\partial_t V^*(s,t) + o(h);$$

therefore, letting $h \to 0$,

$$-\partial_t V^*(s,t) = \sup_{a \in \mathcal{A}_t(s)} \{r_t(s,a) + \langle f(s,a), \partial_s V^*(s,t)\rangle\},$$

which is equivalent to the HJB equation (1.8) *associated to the control system* (1.1) *(where we replace the* inf *by a* sup *because we are here trying to maximize a reward).*

2.4. **Overview of methods.** Reinforcement learning comes with a wide range of approaches, and trying to cover them all here would be of little interest. At least, we will introduce a common classification for RL methods and highlight some of there recurrent features.

We call *model-based* methods, those which rely on an explicit mathematical model for the evolution (*i.e.*, the f_t and the laws of the w_t are known, for all t). These methods generally aim at mitigating the curse of dimensionality by resorting to *value iteration* (or some variant of it) and *approximating* the model to reduce the amount and complexity of the computations.

Value iteration is the process of improving on an initial estimation V_0 of the optimal value function by iteratively applying the Bellman operator $\mathcal{T}$. The latter is obtained by replacing the equality by an affectation in Theorem 2.1:

$$\mathcal{T}[V](s) = \sup_{a \in \mathcal{A}(s)} \mathbb{E}_w \left[r(s,a) + \gamma V(f(s,a,w)) \right], \quad \forall s \in S. \tag{2.11}$$

This algorithm is detailed in Appendix A.1 for clarity. If the time horizon is finite, or if the discount factor γ is strictly smaller than 1, this algorithm can be proven to converge to the optimal value function V^*. The proof relies on Banach's fixed point theorem and on the fact that $\mathcal{T}$ is a contraction. Using value iteration is an alternative to the Bellman recursion and can be stopped after some arbitrary convergence criterion is met. Computations can also be further reduced by applying approximations of different kinds in the Bellman operator:

- approximation on the minimum (*e.g.* parameterization of the policy);
- approximation of the expected value (*e.g.* Monte-Carlo estimations);
- approximation of the model.

Model-free methods, on the other hand, don't explicitly rely on the model. [1] They can typically be divided into two categories:

- *value-based* methods try to estimate the value function, and derive the policy from it;
- *policy-based* methods typically parameterize the policy (for example using a deep neural network of weights θ) to directly express the accumulated reward as a function of the parameter θ. This function can be optimized with respect to θ, for instance using some gradient descent technique.

For further details, Recht gives a good review of reinforcement learning in analogy with optimal control in his 2019 article [20].

[1] Note, however, that the rewards are considered given in the problem and would typically be designed by an engineer. Reward design can be a difficult problem in itself and is not covered here.

2.5. **Q-learning.** To close this section on reinforcement learning, let us detail a simple but foundational example of model-free, value-based method, namely that of tabular Q-learning.

Here, for simplicity, we assume that the state space S and the action space $\mathcal{A}$ do not depend on time and are finite. Q-learning starts with the observation that the derivation of a greedy policy (*cf.* definition 2.5) requires knowing the model. To avoid this, we introduce a variant of the value-function called the *action-value* function.

Definition 2.6 (Action-value function (finite-horizon case))**.** *Recovering the notations of definition 2.4, the* action-value *function associated to a policy π is defined by;*

$$Q^\pi(s,a,t) = r_t(s,a) + \mathop{\mathbb{E}}_{w_t}\big[V^\pi(s_{t+1}, t+1)\,\big|s_t = s, a_t = a\big].$$

This function is the accumulated reward obtained when starting in s at time t, taking action a and following the policy π from then on. We also define the optimal action-value function as:

$$Q^*(s,a,t) = \max_\pi Q^\pi(s,a,t) \tag{2.12}$$

It may seem that introducing this function makes things more complicated because we added a dependency on the action a. However, deriving a greedy policy now becomes much easier: if Q^* is known, the associated optimal policy is given by

$$\forall t \in \{0, ..., T-1\},\ \forall s \in S,\ \pi_t^*(s) = \arg\max Q^*(s,a), \tag{2.13}$$

which does not involve f or the law of the w_t. Then, if we have access to the actual system (or a simulator for it), we can generate sample triplets $(s,a,r) \in S \times \mathcal{A} \times \mathbb{R}$ simply by interacting with the system and observing the resulting states and rewards. These can be used to perform algorithms similar to those from dynamic programming, such as value iteration, in order to estimate Q^*. To do that, note that the Bellman equation can be re-written for Q:

Theorem 2.2 (Bellman equation for Q)**.** *With the notations introduced for Definition 2.6 and assuming that the dynamical system is an MDP, in the finite horizon case, $\forall s \in S,\ \forall a \in \mathcal{A}$,*

$$Q^*(s,a,T-1) = r_{T-1}(s,a) + \mathbb{E}_{w_{T-1}}\left[r_f(s_T)\middle|\ s_{T-1} = s, a_{T-1} = a\right] \tag{2.14}$$

and $\forall t \leq T-1,\ \forall s \in S,\ \forall a \in \mathcal{A}$,

$$Q^*(s,a,t) = r_t(s,a) + \mathop{\mathbb{E}}_{w_t}\left[\max_{a' \in \mathcal{A}} Q^*(s_{t+1}, a', t)\ \middle|\ s_t = s, a_t = a\right]. \tag{2.15}$$

Q-learning is detailed in the algorithm of Appendix A.2. In this algorithm, we arbitrarily initialize an estimate Q for Q^*. Then, we apply a variant of value-iteration that involves two main changes. The first modification lies in the order with which the states are selected for the update, which is constrained by the order in which they are encountered during an episode. This depends on the initial state, on the randomness of the system (if applicable) and on the way the actions are chosen. One natural choice would be to always choose the best action according the current estimate Q (*i.e.* use the greedy policy); but ideally, we want to explore every state an infinite number of times to ensure convergence, while limiting variance to converge rapidly. One way to achieve this is to use a random policy to encourage exploration of ill-evaluated state-action pairs, for example using an ε-greedy policy ($\varepsilon > 0$):

$$\pi^\varepsilon(s,t) = \begin{cases} \arg\max_{a \in \mathcal{A}} Q(s,a,t) & \text{with probability } 1-\varepsilon \\ a \text{ chosen randomly in } \mathcal{A} & \text{with probability } \varepsilon. \end{cases} \tag{2.16}$$

Because the policy that is used to estimate Q^* is different from the greedy-policy associated to that estimate, we say that Q-learning is an *off-policy* method: it is not advised to use this algorithm if good performance should be achieved early on in the training of the RL agent!

A second difference with value iteration is the update rule, which involves a "learning rate" α. The latter evaluates the speed with which Q is updated towards the target $\mathcal{T}[Q]$ (with $\mathcal{T}$ the Bellman operator chosen as the right-hand side in (2.15)), allowing to apply the value iteration only partially. Indeed, if $\alpha = 1$, the update rule becomes equivalent to $Q \leftarrow \mathcal{T}[Q]$; but if $\alpha = 0$, Q is left unchanged. Now, this update rule can also be interpreted from the point of view of *temporal difference* when decomposed as follows:

$$Q(s,a,t) \leftarrow Q(s,a,t) + \alpha \underbrace{\left[\overbrace{r_t(s,a) + \max_{a' \in \mathcal{A}} Q(s,a',t+1)}^{\text{target}} - Q(s,a,t)\right]}_{\text{temporal difference}} \tag{2.17}$$

Here, the target is the term to which we want $Q(s,a,t)$ to get closer; Q is equal to the target in every point (s,a,t) if and only if it is equal to Q^*. The distance to the target is coined "temporal difference" because it is the difference between a term at time t and a term at time $t+1$. In fact, this difference could be extended over time, for example by storing states and actions taken at intermediate steps (s_t,a_t) and (s_{t+1},a_{t+1}) obtained using the ε-greedy policy and using a target:

$$r_t(s_t,a_t)+r_{t+1}(s_{t+1},a_{t+1})+\max_{a'\in\mathcal{A}}Q(s_{t+2},a',t+2). \tag{2.18}$$

Doing so would account for a more "far-sighted" agent, that is an agent capable of assimilating actions to consequences on longer terms. In practice, such a method is not well adapted to off-policy methods because one has to account for the fact that the policy used to learn Q is not the greedy one. More details can be found in the excellent introduction to reinforcement learning by Sutton and Barto [23, chapter 6].

Let us close this section by remarking that Q-learning can be proven to asymptotically converge towards Q^* with probability 1 under certain conditions on the learning rate and the boundedness of the reward. The proof for this convergence was established by Watkins in his PhD thesis in the late 1980s [29], and a more straightforward proof is given *e.g.* in [12].

3. Reinforcement learning for optimal control: a linear proof-of-concept

Now that both optimal control and RL have been introduced, we want to explore to what extent RL can compensate for the difficulties listed in 1.4. Therefore, we wish to compare these methods on control problems. As a proof-of-concept, this illustration is first carried-out on two variants of a simple linear-quadratic problem (*i.e.* linear dynamics and a quadratic cost). These are the occasion to implement the Q-learning algorithm presented in sub-Section 2.5, but also to sketch and pave the way for our comparative study of optimal control and RL on harder problems.

The two linear toy-problems that are considered involve a cart (represented as a mechanical point) that is translating along the real axis x. The cart starts in a random position in $[-1,0]$ and we wish to move it towards a target placed in position 0 over a certain time period by exerting a force on it, while minimizing the total exerted force. This is illustrated by Figure 2. The approach used is the following:

(1) An analytical solution is derived to get an optimal baseline.
(2) The dynamics are discretized in time to derive a numerical simulator, which is a function that takes a state and action as input and outputs the next state. Ideally, this simulator should be replaced by the actual system. The cost function is also decomposed into a sum over the discretized time steps to define rewards that will be consumed by the RL agent.
(3) A traditional method from optimal control, the adjoint method, is applied by identifying a mathematical model underlying the dynamics.
(4) Then the simulator and the decomposed cost are used to move the problem into an RL framing and solve it using an RL algorithm.
(5) Lastly, the two approaches are compared and discussed.

Note that this *discretize-then-optimize* approach places RL among the class of direct methods (as distinguished in sub-Section 1.3) that tackle the problem through successive solving of the forward state equations.

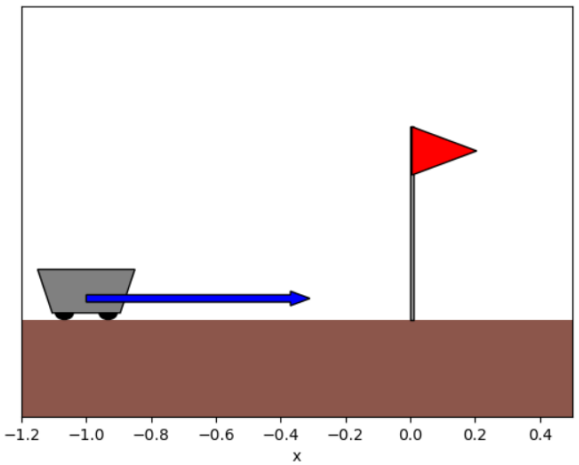


Figure 2. The cart problem, starting in position -1.

3.1. **A first variant solved with Q-learning.** In a first implementation, the dynamics are of the form $\dot{x} = u$, which makes the state space of dimension 1 and allows for a very direct control of the cart's position. As this problem is very similar to the second variant presented below and because we want to focus on the comparison with the optimal control approach, we will here only make a quick review of the results to avoid redundancy.

In particular, this implementation tests several choices of ε and α for Q-learning and visualizes the action-value function for each choice. The main conclusions are that greater ε fosters more exploration and thus better final policies, at the expense of a slower convergence; and that in the deterministic case it is generally advantageous to take a great learning rate (typically equal to 1, which amounts to applying an off-policy value iteration). This illustrates an important feature of RL, which is that it allows to find a compromise between *exploitation*, which is the fact of sticking to a policy that looks promising, and *exploration*, which is the fact of exploring other policies that may give better results. This is a common idea in optimization theory where we are trying to minimize a function by exploring several local minima and keeping the best one. Here, it is seen as a way to mitigate the curse of dimensionality, with the idea that a full-fledged Bellman recursion corresponds to a fully exploratory approach at the expense of intensive computations.

The code for this implementation was written in a commented notebook that is quite self-contained; it can be accessed and run on *Google Colab* through this *Github* link: `https://github.com/TheiloT/Q-learning-sped-up-cart/tree/main`.

3.2. **Introduction of the second variant: the cart-pushing problem.** For the second variant, we will use a pre-implemented RL algorithm and compare it with the adjoint method mentioned in sub-Section 1.3. Thereby, we hope to start illustrating the 4 challenges listed in sub-Section 1.4, but also flaws of RL such as its data-inefficacy and the difficulty of parameter-tuning. This section is also based on a notebook that can be found at `https://github.com/TheiloT/RL-accelerated-cart`

Problem 1. *The cart is assimilated to a mechanical point of mass $m > 0$ that moves along an axis. Its position is denoted by $x \in X = \mathbb{R}$. We want to bring it from position $x_0 \in X_0 = [-1, 0]$ to position 0 with a null velocity at time T, and we want to do this with minimal force. More precisely, we look for $\mathbf{u} \in \mathcal{U}$ where $\mathcal{U} = L^2([0, T], U)$ and we want to solve the optimal control problem (1.2) with*

$$J(\mathbf{x}, \dot{\mathbf{x}}, \mathbf{u}) = \lambda_P \mathbf{x}(T)^2 + \lambda_V \dot{\mathbf{x}}(T)^2 + \int_0^T \mathbf{u}(t)^2 \, \mathrm{d}t, \tag{3.1}$$

with λ_P, $\lambda_V > 0$ and the dynamics of the system are given by:

$$\begin{cases} m\ddot{\mathbf{x}}(t) = \mathbf{u}(t) \quad \forall t \in [0, T], \\ \mathbf{x}(0) = x_0, \\ \dot{\mathbf{x}}(0) = 0, \end{cases} \tag{3.2}$$

where x_0 is drawn randomly in X_0. Lastly, we assume that the mass of the cart $m > 0$ is unknown.

Remark 3.1. *Our choice of $\mathcal{U}$ ensures by the Cauchy-Lipschitz theorem that the Cauchy problem 3.2 does have a unique solution $\mathbf{x_u} \in AC([0, T], \mathbb{R})$ for every $\mathbf{u}$.*

Remark 3.2. *Several aspects make this problem interesting despite its simplicity. Firstly, assuming that m is unknown will invoke a form of model identification to apply numerical methods from optimal control. Secondly, the fact that the initial condition is not known in advance makes it a real-time problem, at least in a weak sense. A harder version of this problem would involve randomness throughout the evolution, but this case has not been covered. Lastly, a compromise must be found between short-term rewards, which is the minimization of the applied force, and a long-term reward, which is the final proximity to the target. Long-term problems are known to be a challenge in optimal control and RL makes no exception, although it comes with a certain toolbox for this case.*

3.2.1. *Analytical solution.* This problem is simple enough for us to derive an analytical solution that can be used for performance comparison. A short proof in Appendix B.1 gives the expression of the optimal $\mathbf{x}$, $\dot{\mathbf{x}}$ and $\mathbf{u}$.

To get familiar with the dependency of the solution to the parameters, Figure 3 shows the optimal trajectories (position and velocity), controls and associated costs for various values of λ_P, λ_V and m, for $T = 1$ and $x_0 = -1$. For our demonstrations, we will fix $\lambda_P = 100$, $\lambda_V = 50$ and $m = 1$ in what follows. We recall however that m will be considered unknown, contrary to λ_P and λ_V because they set the objective that we want to optimize.

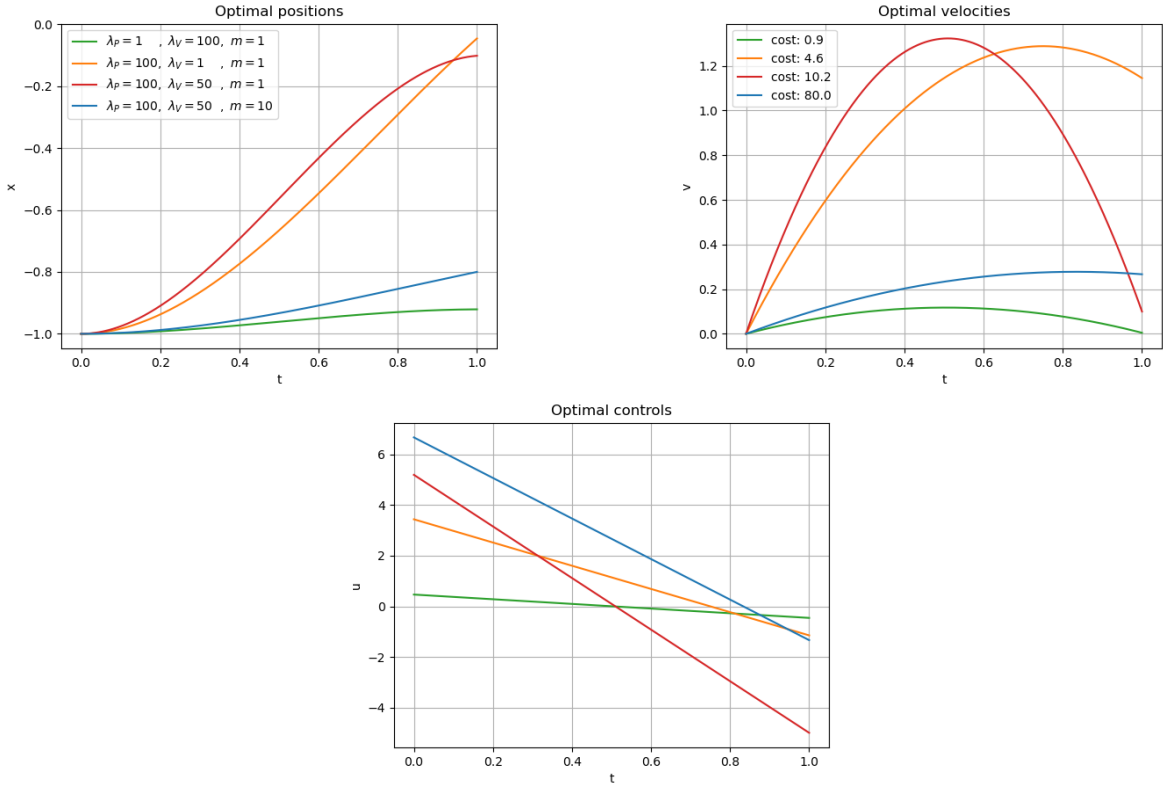


FIGURE 3. Optimal trajectories (position and velocity) and controls for various λ_P, λ_V and m, with associated costs

3.2.2. *Deriving a simulator.* The evolution is discretized in time, and this discretization will be used for reinforcement learning. Details on the discretization are in the notebook for this section, but we recall the notations here. We denote by $N \in \mathbb{N}^*$ the discretization resolution and $\Delta t = \frac{T}{N}$ the time step. Let $\forall n \in \{0, ..., N\},\ t_n = n\Delta t,\ u_n = \mathbf{u}(t_n)$ (arbitrarily taking $u_N = 0$), $x_n = \mathbf{x}(t_n)$ and $v_n = \dot{\mathbf{x}}(t_n)$. We also denote by $\mathtt{x}$, $\mathtt{v}$ and $\mathtt{u}$ the vectors $(x_0, ..., x_N)^{\mathrm{T}}$, $(v_0, ..., v_N)^{\mathrm{T}}$ and $(u_0, ..., u_{N-1})^{\mathrm{T}}$.
The control $\mathbf{u}$ is searched in

$$\mathcal{U}_{\Delta t} = \left\{ \sum_{n=0}^{N-1} u_n \mathbb{1}_{[t_n, t_{n+1})} \;\middle|\; \forall n \in \{0, ..., N-1\}, u_n \in U \right\}. \tag{3.3}$$

Then the solution $\mathtt{x}$ to the ODE verifies:

$$\begin{pmatrix} x_{n+1} \\ v_{n+1} \end{pmatrix} = \mathcal{P}(x_n, v_n, u_n) \tag{3.4}$$

where $\mathcal{P} : X \times V \times U \to X \times V$ is defined by

$$\mathcal{P}(x, v, u) = \begin{pmatrix} \mathcal{P}_x(x, v, u) \\ \mathcal{P}_v(v, u) \end{pmatrix} = \begin{pmatrix} x + \Delta t\, v + \frac{\Delta t^2}{2m} u \\ v + \frac{\Delta t}{m} u \end{pmatrix} \tag{3.5}$$

and will be called the "simulator". This simulator will represent the system under scrutiny: ideally, this would be the real physical system and we will use it to simulate the interactions that arise in the algorithms implemented.
The optimal control problem also rewrites:

$$\underset{\mathtt{u} \in \mathcal{U}_{\Delta t}}{\text{minimize}}\, J(\mathtt{x}, \mathtt{u}) \tag{3.6}$$

subject to the dynamics (3.5), where

$$J(\mathtt{x}, \mathtt{u}) = \lambda_P x_N^2 + \lambda_V v_N^2 + \Delta t \sum_{n=0}^{N-1} u_n^2. \tag{3.7}$$

In our case, we suppose we are able to compute each term of this cost exactly from the observation of $\mathtt{x}_n$, $\mathtt{v}_n$ and $\mathtt{u}_n$.

The parameters chosen are presented in Appendix B.3.

3.3. **Solving by adjoint method.** Let us now assume that we don't have an analytical solution. How can we solve this control problem? Strictly following optimal control theory, we would have to solve an inverse problem to fit a model and determine m thereby. From there, we can use one of the conventional methods from optimal control.

3.3.1. *Finding a model.* The model identification process here is quite simple if we assume that we know the laws of physics: the only unknown to be uncovered is now the mass m. Its derivation is pretty straightforward; its derivation is commented in the notebook of this section. In this case, the identification yields the exact value (up to numerical errors) in very little time and effort, because the model is simple, already well known and noise-less. For a more complex and noisy system, this task would be much more daunting, and maybe the hardest part of the whole solving process.

3.3.2. *Applying an optimal control method.* With this identified model in hand, optimal control offers a whole range of techniques to minimize the cost. Here assume that no analytical solution is known Here The idea is simply to apply a gradient descent to minimize the functional J. For simple problems like this one, we could directly work out an analytical expression of the gradient of J (here seen as a function of the time series $\mathbf{u}$, so as a function from U^N to $\mathbb{R}$), but for harder problems this would be a daunting task because of terms x_N and $\dot{x}_N$ which need integrating the ODE over the whole time period. We could approximate this gradient by finite difference, but then we would need to integrate the state equation in all directions of U^N. One way to ease this computation is to get an implicit expression through the computation of the adjoint state $\mathbf{p}$. Therefore, evaluating the gradient only requires integrating the state equation once (to get the final state on $\mathbf{p}$) and the adjoint equation once (to get the value of $\mathbf{p}$ over the full time period).

Appendix B.2 presents the computation of the gradient of J in this case and gives the following expression:

$$\nabla J(\mathbf{u}) = 2\mathbf{u} + 2\mathrm{B}^{\mathrm{T}}\mathbf{p}. \tag{3.8}$$

The state and adjoint equations can be integrated using a simple Euler integration scheme, which gives:

$$y_{n+1} \approx y_n + \Delta t\big(\mathrm{A}y_n + \mathrm{B}u_n\big), \tag{3.9}$$

where $\mathbf{y}(t) = \begin{pmatrix}\mathbf{x}(t)\\ \dot{\mathbf{x}}(t)\end{pmatrix}$, and

$$p_n \approx p_{n+1} + \Delta t \mathrm{A}^{\mathrm{T}} p_{n+1}. \tag{3.10}$$

Using `scipy`'s implementation of the BFGS algorithm, we get a final cost of 8.23, versus 8.24 for the analytical solution (the adjoint method giving a better result than the analytical solution might be due to a numerical or implementation error that hasn't been identified). Figure 4 shows the trajectory and control obtained with this method for $x_0 = -0.9$. The orange curve (ground truth) is obtained using the analytical solution. On the left, the trajectory obtained with the adjoint method is confounded with the ground truth; on the right, the control predicted by the adjoint method is represented piece-wise to emphasize the fact that the control is approximated by a piece-wise constant function.

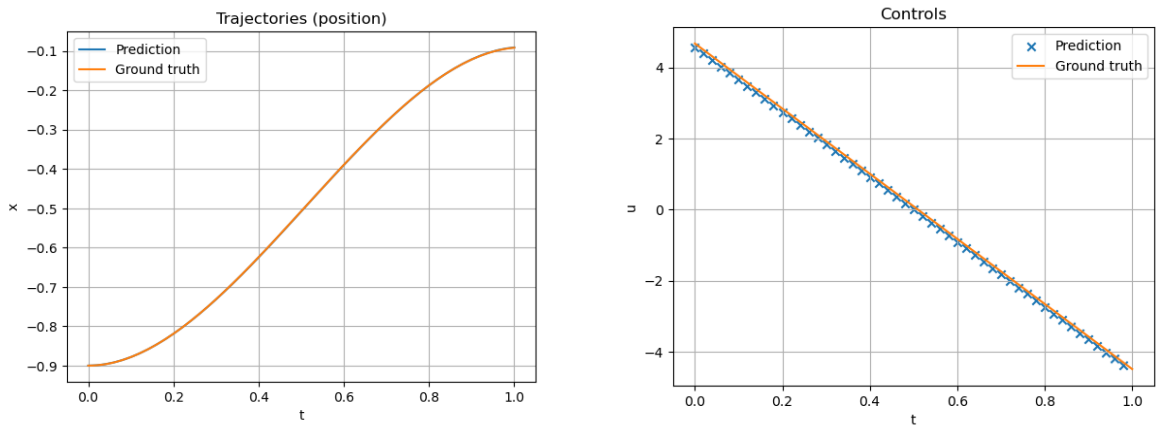


FIGURE 4. Trajectory and control given by the adjoint method for $x_0 = -0.9$

3.3.3. *Discussion on the adjoint method.* To conclude with this section, the adjoint method gives very good results, but it requires to identify a model, a process that can be painstaking for complex, random and/or noisy systems. It also requires analytically expressing the gradient of the cost in terms of the adjoint state; a difficulty that doesn't arise when using direct methods (such as using an NLP solver) and that also tends to be mitigated with the resort to automatic differentiation (see for instance Thuerey's work on differentiable physics [25]). Lastly, the gradient descent has to be run all over again for every new instance of the problem (here, for every new x_0).

3.4. **Solving by reinforcement learning.**

3.4.1. *Choice of an algorithm.* As announced in Section 2.4, there exists a wealth of reinforcement learning algorithms in the literature. In particular, tabular Q-learning could be used again through a proper discretization of U, but we will turn to a different kind of algorithm. The PPO (Proximal Policy Optimization) algorithm allows to work with continuous state and action spaces. Although little convergence results are known concerning PPO to this day, it empirically achieves fast convergence for a range of different problems. It also has the benefit of leveraging parallelization of computations on multiple computer threads during the training of the RL agent.

Note that PPO is quite different from Q-learning presented in Section 2.5: it is an actor-critic algorithm, meaning that it involves two components typically represented by deep neural networks (DNN):

- the actor (of parameters θ) is a function $\mu_\theta : S \to S$ that is used to build a probabilistic policy $\pi(\cdot;\theta)$(*i.e.* it maps each state to a probability distribution), and the probability of choosing action a when in state s is $\pi(a;s,\theta)$.
- the critic (of parameters ϕ) is an estimate $V(s;\phi)$ of the value function associated to $\pi(\cdot|\theta)$.

In a typical implementation of PPO, $\pi(\cdot;s,\theta)$ is a Gaussian distribution whose mean is $\mu_\theta(s)$. The variance of the distribution accounts for the amount of exploration in this algorithm, and this variance is progressively decreased to end up with a deterministic policy. With this policy in hand, a naive approach would consist in expressing the expected value of the cumulative reward $\mathbb{E}_\theta\left[\sum_{t=0}^{T-1} r_t(s,A_t)\right]$ as a function of θ and directly maximizing it. Instead, after each episode (or batch of episodes), the critic is used to build a surrogate objective that will be maximized. This surrogate objective measures how much a new value of θ improves on the current policy with regard to the accumulated rewards, while preventing steep changes in this policy that could harm the algorithm's robustness. More details can be found in the original paper [21].

3.4.2. *Formulation as an RL problem.* The problem needs to be formulated according to the formalism introduced in Section 2.3. For implementation reasons, U must be restrained to a bounded set, so we will look for u in $\tilde{U} = [U_L, U_R]$ (see Appendix B.3 for the choice of values). Therefore we give the following formulation:

- $S = \{0,...,T\} \times X \times V$. A state is $s = (t,x,v) \in S$ and represents the current time step and the position and velocity of the cart at this time step. Including time in the state is necessary as the optimal action may depend on the remaining time until the end of the evolution.
- $\mathcal{A} = \tilde{U}$. An action is $a \in \mathcal{A}$ and represents the control $\mathbf{u}$ at a given time step.
- The dynamics of the system is $f = \mathcal{P}$ (in other words, the system itself).

We also define the running rewards $\forall n \in \{0,...,N-1\},\ \forall s \in S,\ \forall a \in \mathcal{A}$,

$$r_n(s,a) := r(a) = -\Delta t\, a^2, \tag{3.11}$$

and the final reward $\forall s \in S, a \in \mathcal{A}$,

$$r_N(s,a) := r_f(x,v) = \lambda_P x^2 + \lambda_V v^2. \tag{3.12}$$

3.4.3. *Results and interpretation.* For numerical experiments we use the implementation of the PPO algorithm as coded in the `stable_baselines3` library of *Python*.

The training process follows the PPO algorithm presented in [21], choosing a random value for x_0 in $[-1,0]$ at the beginning of each new training episode. The chosen combination of parameters for the algorithm is presented in Appendix B.4. The history of the training process is visualized on a training curve constructed as follows: every 1000 time-steps, the training process is interrupted and the variance of the policy is momentarily set to 0. The agent is then evaluated on 50 episodes and the mean, standard-deviation, minimum value and maximum value of the cumulative rewards obtained over these 50 episodes are computed and make one point of the curve. The resulting curve is shown on Figure 5; the red line is a some arbitrary reference: it is the cumulative reward obtained with the analytical solution in the worst

case scenario ($x_0 = -1$). The policy that is retained to make the final agent is that which achieved the best mean performance at evaluation time.

Figures 6 and 7 show the trajectory and control predicted by the agent after training, for $x_0 = -0.9$ and $x_0 = -0.5$ respectively. For $x_0 = -0.9$, the associated cost is approximately 14.1 (versus 8.2 for the analytical solution) and for $x_0 = -0.5$, the associated cost is approximately 4.1 (versus 2.5 for the analytical solution).

The training curve can be decomposed into two parts: during the first 50 000 time-steps the agent gives bad control with high variance. Past this point, the evolution is quite bumpy but remains around a constant level, around a cumulative reward of -6. Note that most of the variance can be explained by the fact that x_0 is chosen randomly at the beginning of each time step. The algorithm globally achieved mitigated performance. One interesting observation is that the agent tends to accelerate too late in the early stages of an episode, causing it to end far from the objective, and it decelerates sharply in later stages. This is symptomatic of a short-sighted agent that hasn't explored early stages correctly; yet increasing the γ factor in the algorithm did not seem to improve the results. One obvious way to solve this issue would be to reduce the number of time-steps in an episode by increasing the length of a time-step; but that would result in a more coarse parameterization of the control and trajectory.

Let us finish with a comment on complexity. If we were to solve the problem by Bellman recursion, we would also have to discretize the position, velocity and action spaces. As detailed in the Google Colab notebook associated to this section, the complexity of such an approach is in $O(N^4 N_U^3)$, where N is the number of time steps, and N_U the number of possible values for the action. For the parameters specified in B.3 and using $N_U = 4$ and $N = 8$, the number of elementary operations is already of the order of 528×10^6 and is solved in about 8 minutes on a regular laptop. Computing the complexity for PPO would be trickier, but considering that the training is of the order of 10 000 time steps (as observed on Figure 5), this training procedure takes about 3 minutes on the same laptop using $N = 50$, and we did not have to discretize in position, velocity and action.

This gain in computational time at the expense of final performance again underlines the exploration-exploitation dilemma in RL and how it can be used to mitigate the curse of dimensionality.

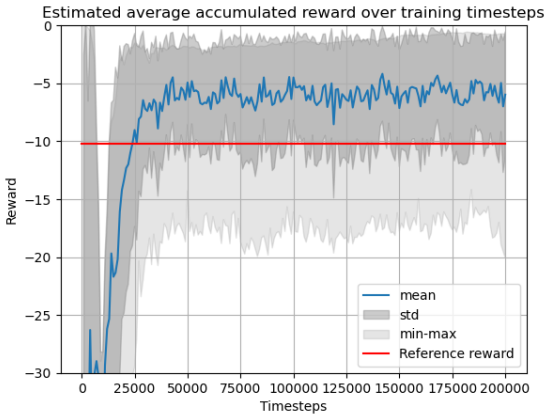


FIGURE 5. Training curve for the PPO agent.

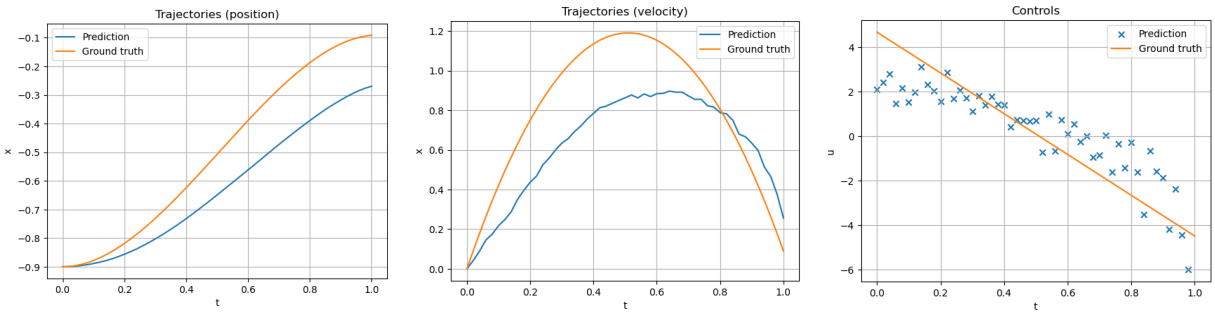


FIGURE 6. Trajectory and control given by the trained PPO agent for $x_0 = -0.9$.

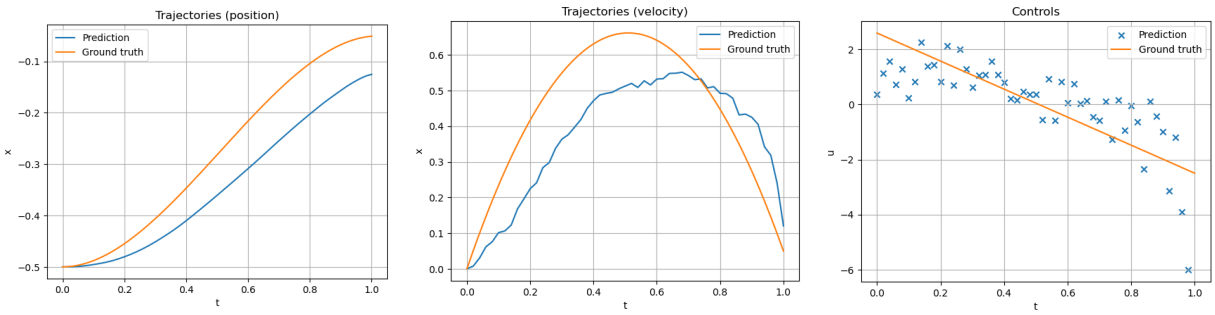


FIGURE 7. Trajectory and control given by the trained PPO agent for $x_0 = -0.5$.

3.5. **Discussion on the two methods.** To conclude this section, let us compare the two applied approaches. Let us first note that no model was given to the RL agent of sub-Section 3.4: the agent was not told the value of the mass m, nor did it know about the laws of physics. This avoids the whole process of model identification that is required by many optimal control methods like the adjoint method. Secondly, we illustrated the mitigation of the curse-of-dimensionality that can be done by solving the exploration-exploitation dilemma that characterizes RL. Lastly, let us note that to start tackling a new instance of the problem, the adjoint method has to solve the instance entirely. The RL agent, on the other hand, is about 10 times quicker to decide the first action for this simple problem, and this could make a huge difference for harder problems.

Note however, that this computational gain is at the expense of the training cost, both in terms of time and data samples. To insist on this last drawback, let us emphasize that as read on the training curve of Figure 5, the agent starts giving somewhat satisfactory results after about 50 000 samples, that is 90 seconds of training, versus about a tenth of a second and 100 samples for the adjoint method. This illustrates the idea in RL (and machine learning in general) of "paying offline to win online". One way to cope with this issue is to resort to *transfer learning*, which consists in having an agent pre-trained on a set of simple instances of a problem, and deriving new agents from it by training it further on specific instances. Doing this avoids using a lot of training computations each time a new agent has to be built. For a review on such ideas, see for instance the work of Zhu *et al.* on the subject [31]. Furthermore, the final performance of the RL agent is not as good as the near-optimal results of the adjoint method (up to time discretization); but this comparison should be carried out again on hard non-linear problems in which gradient descent methods may fall into local minima.

4. Controlling the Burgers' equations

Now, we want to move to harder problems that involve some form of non-linearity. Fluid dynamics offer an ideal playground for this, as they introduce a range of optimal control problems that involve equations of various non-linearity. In 2021, P. Garnier [10] reviewed several successful applications of RL to control problems for fluid dynamics. Here we will focus on the (viscous) Burgers' equation, first introduced by H. Bateman [2] and J. M. Burgers [8, 7] as an approximation for equations of fluid-flow.

In this section, we introduce the problem rigorously and tackle it with reinforcement learning.

Notation. *For this section, both scalar-valued functions and scalars will be denoted with lowercase letters: x, t, u. The disambiguation will be done by always writing the argument of a function when needed. Also, we will now denote by u the velocity field of the fluid and by a the control.*

4.1. **Mathematical formulation of the problem.** We address the one-dimensional Burgers' equation. More precisely, we aim to control the solution u to the following initial-value problem with zero boundary conditions (later called the *forward* problem as opposed to the control problem below):

$$\begin{cases} \partial_t u + \partial_x \left(\frac{u^2}{2}\right) - \nu \partial_{xx}^2 u = a(t,x), & t \in (0,T),\ x \in \Omega, \\ u(0,x) = u^0(x), & x \in \Omega, \\ u(t,x) = 0, & t \in (0,T),\ x \in \partial\Omega, \end{cases} \tag{4.1}$$

where $\nu > 0$ is the *viscosity parameter*, $\Omega = (0,1)$, T is the *time horizon*, $u^0 \in L^2(\Omega, \mathbb{R})$ is an initial value for the velocity field and the forcing term a_ν belongs to a suitable class of admissible controls.

Notation. *As in* [26]*, in this section we endow $L^2(\Omega)$ with its natural inner product and $H_0^1(\Omega)$ with the inner product*

$$(v,w)_{H_0^1(\Omega)} = (v', w')_{L^2(\Omega)}, \qquad \forall v, w \in H_0^1(\Omega).$$

We also denote by $L^2(0,T;H_0^1(\Omega))$ *the Bochner space of equivalence classes of measurable functions* v *which are square integrable:*

$$\int_0^T \|v(t)\|^2_{H_0^1(\Omega)}\,\mathrm{d}t < \infty.$$

We introduce

$$W(0,T) = \left\{v \in L^2(0,T;H_0^1(\Omega)) \,\middle|\, \partial_t v \in L^2(0,T;H_0^1(\Omega)')\right\},$$

which is a Hilbert space with the scalar product

$$(u,v)_{W(0,T)} = \int_0^T (u(t),v(t))_{H_0^1(\Omega}\,\mathrm{d}t + \int_0^T (u'(t),v'(t))_{H_0^1(\Omega)'}\,\mathrm{d}t.$$

Here, we have used the abbreviation $(F,G)_{H_0^1(\Omega)'} := (JF,JG)_{H_0^1(\Omega)}$*, where* $J : H_0^1(\Omega)' \to H_0^1(\Omega)$ *is the duality mapping from the* Riesz representation theorem *that assigns to each functional* $F \in H_0^1(\Omega)'$ *the corresponding* $f \in H_0^1(\Omega)$*. Likewise,*

$$\mathcal{W}(0,T) = \left\{v \in L^2(0,T;H^2(\Omega)\cap H_0^1(\Omega)) \,\middle|\, \partial_t v \in L^2((0,T)\times\partial\Omega)\right\}.$$

We refer to [26, Theorem 2.2] for the well-posedness of (4.1). More precisely, Theorem 4.1 below gives uniqueness and existence of the weak solution to the Burgers' equation, according to the following definition of a weak solution.

Definition 4.1 (Weak solution to the Burgers' equation)**.** *Let* $\nu > 0$*,* $u^0 \in L^2(\Omega,\mathbb{R})$*, and* $a : (0,T)\times\Omega \to \mathbb{R}$*. A function* $u : [0,T] \to$ *is called a* weak solution *to the Burgers' equation* (4.1) *if* $u \in W(0,T)$ *and*

$$(\partial_t u(t),v)_{H_0^1(\Omega)',H_0^1(\Omega)} + \nu\langle u(t),v\rangle_{H_0^1(\Omega)} + \langle u(t)\partial_x u(t),v\rangle_{L^2(\Omega)} = \langle a(t),v\rangle_{L^2(\Omega)}$$

for all $v \in H_0^1(\Omega)$ *and a.e.* $t \in [0,T]$*, and*

$$\left\langle u(0),\chi\right\rangle_{L^2(\Omega)} = \left\langle u^0,\chi\right\rangle_{L^2(\Omega)} \qquad \text{for all } \chi \in L^2(\Omega)$$

Theorem 4.1 (Well-posedness of the Burgers' equation)**.** *For all* $a \in L^2((0,T)\times\Omega)$ *and all* $u^0 \in L^2(\Omega)$ *there exists a unique weak solution* $u \in W(0,T)$ *to the Burgers' equation* (4.1) *satisfying*

$$\|u\|_{W(0,T)} \le C(1 + \|a\|^2_{L^2((0,T)\times\Omega)}) \tag{4.2}$$

where $C > 0$ *only depends on* u^0 *and* ν*. Moreover, if* $u^0 \in H_0^1(\Omega)$*, then* $u \in \mathcal{W}(0,T)$*.*

We are interested in studying the following optimal control problem.

Problem 2. *Given an initial velocity* $u^0 \in L^2(\Omega)$ *at time* $t = 0$ *and a target* $u^* \in L^2(\Omega)$*, exert a force* $a \in \mathcal{A}$ *to minimize the* L^2*-distance between* $u(T,\cdot)$ *and* u^* *with* "minimal effort"*:*

$$\min_{a\in\mathcal{A}} J_\nu(u,a), \tag{4.3}$$

where

$$J_\nu(u,a) := \beta\|u(T,\cdot) - u^*(\cdot)\|^2_{L^2(\Omega)} + \int_0^T \|a(t,\cdot)\|^2_{L^2(\Omega)}\,\mathrm{d}t \tag{4.4}$$

is the cost functional with $\beta \in \mathbb{R}$ *a* final reward *factor,* u *is the unique solution to the forward problem 4.1 and the forcing term* a *belongs to the class of admissible controls*

$$\mathcal{A} = \left\{ a \in L^2((0,T)\times\Omega) \,\middle|\, a_l \le a(t,x) \le a_u \text{ a.e. in } (0,T)\times\Omega \right\}, \tag{4.5}$$

where $0 < a_l < a_u$ *are two real numbers.*

The existence of an optimal pair satisfying the optimal control problem (4.3) is essentially contained in [26, Theorem 2.5]. It is recalled in the following theorem, whose proof is available in Appendix C.

Theorem 4.2 (Existence of an optimal control)**.** *Problem 2 has at least one globally optimal solution* $(u_{a_\nu^*}, a_\nu^*)$*.*

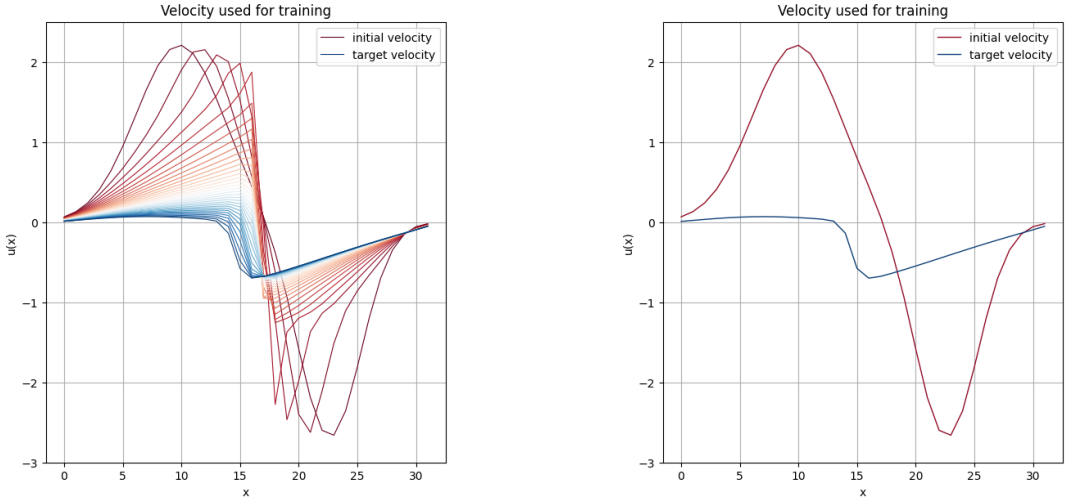
4.2. **Simulator and instance generation.** The implementation used here is based on a code from Thuerey's group at the Technical University of Munich [25, Chapter 16].

As for the linear problem, we have to build a simulator for the dynamics. We start by introducing a time discretization of step $\Delta t = \frac{1}{N}$ and associated points in time $t_n = n\Delta t$, as well as a space discretization of step $\Delta x = \frac{1}{N_X}$ and associated points in space $x_n = n\Delta x$. The velocity u and force a will be numerically evaluated on the discretized points and we denote $\forall n \in \{0, ..., N\}$, $u_n = u(t_n)$ and $a_n = a(t_n)$. The applied force will be considered piece-wise constant in time, *i.e.* $\forall t \in [0, T]$, $a(t) = a_{\lfloor \frac{t}{T} N \rfloor}$. To numerically solve forward problem (4.1), Thuerey *et al.* use their own framework for solving PDEs, called `phiflow`. We denote by $\mathcal{P} : (u, a) \in \mathbb{R}^{N_X+1} \times \mathbb{R}^{N_X+1} \mapsto u_{new} \in \mathbb{R}^{N_X+1}$ the simulator, implemented using `phiflow` as the succession of

(1) a numerical diffusion step applied on u using an explicit Euler scheme with viscosity ν and over time Δt;
(2) an advection step applied on u against itself using a semi-Lagrangian scheme over time Δt;
(3) the application of the force a in an additive manner, *i.e.* $u \leftarrow u + a$.

Iterating this simulator N times from an initial condition u^0 brings the velocity field to a final state u_f that we want close to the target u^*. Here, we choose $N_X = N = 32$ and the viscosity is chosen to be $\nu = 0.003$.

To get an initial and target velocities, we start by initializing a random velocity u_0 as the sum of two Gaussians of opposite directions, one on the left and one on the right, as seen on Figure 8b. Then, a random (but constant in time) force is applied throughout the evolution to give a target state u^*. Figure 8a shows the generation process with intermediate states over time. Note that the force used to generate the data is not necessarily the optimal control to go from u_0 to u^*.



(A) Generation of an initial and target velocities

(B) Initial and target velocities

FIGURE 8. Example of an instance of the problem.

4.3. **Solving by reinforcement learning.** The RL formulation of the problem is as follows:

- $S = \{0, ..., T\} \times \mathbb{R}^{N_X+1} \times \mathbb{R}^{N_X+1}$: a state is the data of the current time, the current velocity field, and the target field;
- $\mathcal{A} = \mathbb{R}^{N_X+1}$: the action gives the exerted force in every point x_k, $k \in \{0, ..., N_X\}$;
- the dynamics of the system is $\mathcal{P}$;
- the intermediate reward is $r_n(u, u^*, a) := r(a) = -\|a\|_{\ell^2}^2$;
- the final reward is $r_f(u, u^*) = -\beta\|u - u^*\|_{\ell^2}^2$.

The reason for including the final target in the state is that we want the RL agent to be able to solve several instances of the problem, *i.e.* for several values of u_0 and u^*, similarly to what was done in Section 3 with the varying initial condition x_0.

As in Section 3, the PPO algorithm is used as implemented in `stable_baselines3`. The training procedure is also similar, with instances of the problem generated on the go as in Figure 8. The evaluation procedure is also similar, except that evaluation is performed on a fixed validation set of 100 instances of the problem every 500 episodes. The resulting agent is tested and the result is presented in Figure 9: on

the left, there is the trajectory as controlled by the agent (with increasing time from red to blue); in the middle, the trajectory used to generate the instance; on the right, the trajectory for a null force applied. As an indication, for this instance, the cumulative reward is 1157 for the RL agent versus 1068 for the force used to generate the instance. RL performs 8% worse.

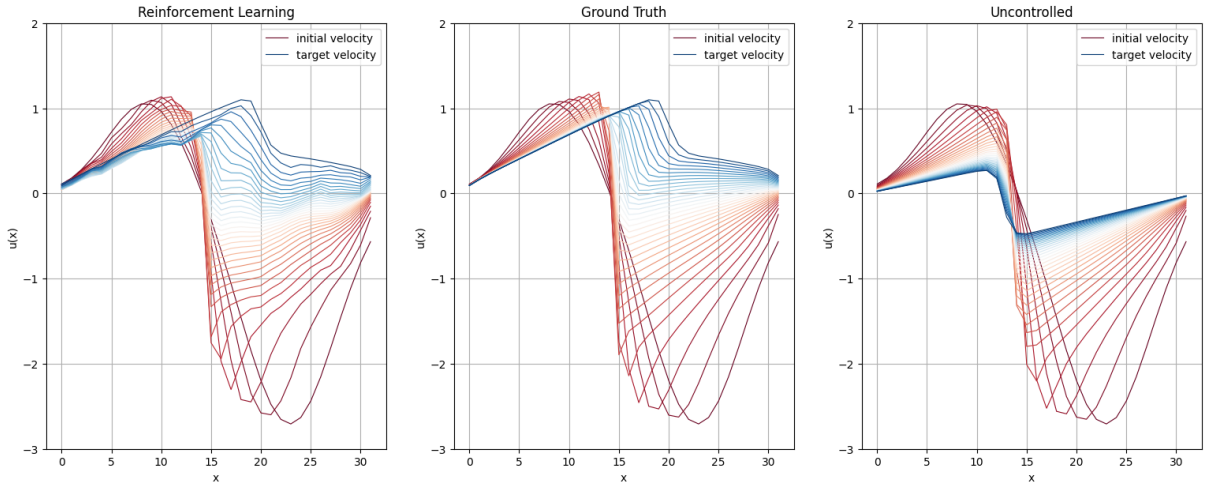


FIGURE 9. Performance of the trained agent.

A solver for this optimal control problem could be implemented for comparison. For instance, in [1], Allahverdi *et al.* resort to the adjoint method for an optimal design problem, where initial conditions are searched so as to minimize the distance to a target field. Another possibility is to leverage the automatic differentiation of the physical solver implemented in `phiflow`. In [25, Chapter 16], Thuerey *et al.* compare reinforcement learning with an adjoint-based method that resorts to such functionalities to get gradient information on the loss. Their results show better performance with their method for most instances of the problem, which depicts RL as hardly competitive with adjoint methods when a gradient is available.

5. CONCLUSIONS

In this memoir, we illustrated in which ways reinforcement learning can compensate for the limitations of some numerical methods from optimal control theory. Firstly, RL can work *model-free* by moving the problem of model identification to that of learning functions by sampling the dynamics of the controlled system through interactions. Secondly, in a *pay-offline-win-online* fashion, a reinforcement learning agent can take a lot of effort to train but can be evaluated very cheaply in the exploitation phase, which makes it suitable for real-time applications as long as the agent can be trained in advance. Thirdly, we show that reinforcement learning draws inspiration from Bellman's theory and tries to mitigate the curse of dimensionality, similarly to what is done in the field of dynamic programming (think about value-iteration); but its emphasis lies on the *exploration-exploitation* dilemma by tuning the variance of the path-exploration process. Lastly, reinforcement learning is able to tackle non-linear problems such as the Burgers' equation, but research led by Thuerey *et al.* in [25, Chapter 16] raises questions about whether reinforcement learning can compete with more common gradient-based approaches in terms of minimization of the cost functional. Further experiments should be carried out on this problem, as well as harder non-linear problems.

On the other hand, like many techniques in machine learning, reinforcement learning needs a lot of training samples to work. The amount of required data may hinder its usage for industrial applications. One way to work around this is to have an efficient simulator, but this somewhat diminishes the benefit of working model-free, and the transition to real instances of the problem is not guaranteed to be seamless. Furthermore, although this has not been clearly underlined in this work, some algorithms are highly sensitive to the choice of hyperparameters, which hinders the robustness and simplicity of the approach.

Further investigation on reinforcement learning and its relation to optimal control can be done in several directions. This starts with further experimenting with the Burgers' equations and, in particular, deriving a solver from the adjoint methodology for comparison with reinforcement learning in terms of minimization of the cost. Harder non-linear problems can later be addressed, for instance, involving the Navier-Stokes equations. Moreover, the usage of reinforcement learning for real-time problems can be investigated more deeply by making the presented dynamics noisy or stochastic. For instance, this can

be done by adding additive Gaussian noise in the right-hand side of (3.2). Lastly, work can be done to alleviate the weakness of these machine learning methods by analyzing how reinforcement learning behaves as regard *transfer learning*.

Appendix A. Dynamic programming and reinforcement learning algorithms

A.1. Value iteration.

Algorithm 1 Value-iteration (infinite horizon case)

Data: termination threshold η.
  $\forall s \in S$, initialize $V(s)$ arbitrarily.
  $\delta \leftarrow +\infty$
  while $\delta > \eta$ **do**
    $\delta \leftarrow 0$
    for $s \in S$ **do**
      $v \leftarrow V(s)$
      $V(s) \leftarrow \sup_{a \in \mathcal{A}(s)} \mathbb{E}_w \left[r(s,a) + \gamma V(f(s,a,w)) \right], \quad \forall s \in S$
      $\delta \leftarrow \max(\delta, |v - V(s)|)$
    end for
  end while

A.2. Q-learning.

Algorithm 2 Q-learning

Data: Exploration factor $\varepsilon > 0$, learning rate $\alpha > 0$, number of episodes N_{episodes}, time-horizon $T > 0$.
  $\forall t \in \{0, ..., T-1\}$, $\forall s \in S$, $\forall a \in \mathcal{A}$, initialize $Q(s,a,t)$ arbitrarily.
  for `episode` $= 1, 2, \ldots, N_{\text{episodes}}$ **do**
    Initialize s.
    for $t = 0, 1, \ldots, T-1$ **do**
      $a \leftarrow \pi^{\varepsilon}(s)$.
      Sample w.
      $s' \leftarrow f(s,a,w)$.
      if $t = T-1$ **then**
        $Q(s,a,T-1) \leftarrow Q(s,a,T-1) + \alpha \left[r_{T-1}(s,a) + r_f(s) - Q(s,a,T-1) \right]$.
      else
        $Q(s,a,t) \leftarrow Q(s,a,t) + \alpha \left[r_t(s,a) + \max_{a' \in \mathcal{A}} Q(s,a',t+1) - Q(s,a,t) \right]$.
      end if
      $s \leftarrow s'$.
    end for
  end for

Appendix B. Cart-pushing problem

B.1. **Proof for the analytical solution of the accelerated cart problem 1.** Problem 1 can be seen as a linear quadratic problem:

$$\begin{cases} \dot{\mathbf{y}}(t) = \mathrm{A}\mathbf{y}(t) + \mathrm{B}\mathbf{u}(t) & \forall t \in [0,T], \\ \mathbf{y}(0) = (0,0)^{\mathrm{T}}, \end{cases} \tag{B.1}$$

with

$$\forall t \in [0,T], \qquad \mathbf{y}(t) = \begin{pmatrix} \mathbf{x}(t) \\ \dot{\mathbf{x}}(t) \end{pmatrix}, \quad \mathrm{A} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}, \quad \text{and} \qquad \mathrm{B} = \frac{1}{m} \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \tag{B.2}$$

and

$$J(\mathbf{u}) = \int_0^T (\mathbf{u}(t)^{\mathrm{T}} \mathrm{R} \mathbf{u}(t))\, \mathrm{d}t + \mathbf{y}(T)^{\mathrm{T}} \mathrm{D} \mathbf{y}(T) \qquad \forall \mathbf{u} \in \mathcal{U} \tag{B.3}$$

with

$$\mathrm{R} = 1 \qquad \text{and} \qquad \mathrm{D} = \begin{pmatrix} \lambda_P & 0 \\ 0 & \lambda_V \end{pmatrix}. \tag{B.4}$$

Under this formulation, the problem can easily be solved using the Principle of Minimum of Pontryagin. Introducing the adjoint state of the system $\mathbf{p} \in AC([0,T], \mathbb{R}^2)$ with $\mathbf{p} = (\mathbf{p}_1 \ \mathbf{p}_2)^{\mathrm{T}}$, we define the Hamiltonian of the system by

$$H(y,p,u) = p^{\mathrm{T}}(\mathrm{A}y + \mathrm{B}u) + u^2, \qquad \forall y, p, u \in \mathbb{R}^2 \times \mathbb{R}^2 \times U. \tag{B.5}$$

With this definition in place, we know that the (it is unique for the LQR) optimal control $\mathbf{u}$ verifies

$$\mathbf{u}(t) = \arg\min_{u \in U} H(\mathbf{y}(t), \mathbf{p}(t), u), \qquad \forall t \in [0,T], \tag{B.6}$$

where

$$\begin{cases} \dot{\mathbf{y}}(t) = \mathrm{A}\mathbf{y}(t) + \mathrm{B}\mathbf{u}(t), \\ \mathbf{y}(0) = \begin{pmatrix} x_0 \\ 0 \end{pmatrix} \end{cases} \tag{B.7}$$

and

$$\begin{cases} \dot{\mathbf{p}}(t) = -\nabla_y H(\mathbf{y}(t), \mathbf{p}(t), \mathbf{u}(t)) = -\mathrm{A}^{\mathrm{T}}\mathbf{p}(t) \\ \mathbf{p}(T) = \mathrm{D}\mathbf{y}(T). \end{cases} \tag{B.8}$$

After solving by hand, we get that, for all $t \in [0,T]$,

$$\begin{aligned} \mathbf{p}(t) &= \begin{pmatrix} \mathbf{p}_1(T) \\ \mathbf{p}_1(T)(T-t) + \mathbf{p}_2(T) \end{pmatrix}, \\ \mathbf{u}(t) &= -\frac{\mathbf{p}_2(t)}{2m}, \\ \mathbf{x}(t) &= \frac{\mathbf{p}_1(T)}{12m^2}(T^3 - (T-t)^3) - \frac{\mathbf{p}_2(T)}{4m^2}t^2 - \frac{T^2\mathbf{p}_1(T)}{4m^2}t + x_0, \\ \dot{\mathbf{x}}(t) &= \frac{\mathbf{p}_1(T)}{4m^2}(T-t)^2 - \frac{\mathbf{p}_2(T)}{2m^2}t - \frac{T^2\mathbf{p}_1(T)}{4m^2}, \end{aligned}$$

where

$$\mathbf{p}_1(T) = \frac{2C_4}{C_1C_4 - C_2C_3}\lambda_P x_0 \qquad \text{and} \qquad \mathbf{p}_2(T) = \frac{-2C_3}{C_1C_4 - C_2C_3}\lambda_P x_0, \tag{B.9}$$

and

$$C_1 = (1 + \frac{\lambda_P T^3}{3m^2}), \qquad C_2 = \frac{\lambda_P T^2}{2m^2}, \qquad C_3 = \frac{\lambda_V T^2}{2m^2}, \qquad \text{and} \qquad C_4 = (1 + \frac{\lambda_V T}{m^2}). \tag{B.10}$$

This expression can then be implemented numerically.

B.2. **Computation of the gradient of the cost for the adjoint method used for problem 1.** The cost writes:

$$J(\mathbf{u}) = \int_0^T (\mathbf{u}(t)^{\mathrm{T}}\mathrm{R}\mathbf{u}(t))\,\mathrm{d}t + \mathbf{y}(T)^{\mathrm{T}}\mathrm{D}\mathbf{y}(T) \tag{B.11}$$

with

$$\mathrm{R} = 1 \qquad \text{and} \qquad \mathrm{D} = \begin{pmatrix} \lambda_P & 0 \\ 0 & \lambda_V \end{pmatrix} \tag{B.12}$$

subject to the differential constraint

$$\begin{cases} \dot{\mathbf{y}}(t) = \mathrm{A}\mathbf{y}(t) + \mathrm{B}\mathbf{u}(t) \quad \forall t \in [0,T], \\ \mathbf{y}(0) = (0,0)^{\mathrm{T}}, \end{cases} \tag{B.13}$$

with

$$\forall t \in [0,T], \qquad \mathbf{y}(t) = \begin{pmatrix} \mathbf{x}(t) \\ \dot{\mathbf{x}}(t) \end{pmatrix}, \qquad \mathrm{A} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}, \qquad \text{and} \qquad \mathrm{B} = \frac{1}{m}\begin{pmatrix} 0 \\ 1 \end{pmatrix}. \tag{B.14}$$

We see that J can be decomposed into $J = J_R + J_D$ as follows:

$$J_R(\mathbf{u}) = \|\mathbf{u}\|_{L^2} \quad \text{and} \quad J_D(\mathbf{u}) = \mathbf{y}(T)^{\mathrm{T}}\mathrm{D}\mathbf{y}(T). \tag{B.15}$$

It is easily shown that $\nabla_{\mathbf{u}} J_R = 2\mathbf{u}$.
To compute the gradient of J_D, this time we want to avoid having to compute $\nabla_{\mathbf{u}}\mathbf{y}(T)$. Let us consider a small perturbation of the control $\delta\mathbf{u} \in L^2([0,T],U)$ and let us introduce $\mathbf{y}_{\mathbf{u}+\delta\mathbf{u}}$ associated with the control $\mathbf{u} + \delta\mathbf{u}$. Then, by the linearity of the state equation,

$$\mathbf{y}_{\mathbf{u}+\delta\mathbf{u}} = \mathbf{y}_{\mathbf{u}} + \delta\mathbf{y} \quad \text{with} \quad \delta\dot{\mathbf{y}} = \mathrm{A}\delta\mathbf{y} + \mathrm{B}\delta\mathbf{u}. \tag{B.16}$$

Since D is symmetric,

$$\begin{aligned} J_D(\mathbf{u} + \delta\mathbf{u}) &= \mathbf{y}_{\mathbf{u}+\delta\mathbf{u}}(T)^{\mathrm{T}}\mathrm{D}\mathbf{y}_{\mathbf{u}+\delta\mathbf{u}}(T)^{\mathrm{T}} \\ &= \mathbf{y}_{\mathbf{u}}(T)^{\mathrm{T}}\mathrm{D}\mathbf{y}_{\mathbf{u}}(T) + 2\mathbf{y}_{\mathbf{u}}(T)^{\mathrm{T}}\mathrm{D}\,\delta\mathbf{y}(T) + \delta\mathbf{y}(T)^{\mathrm{T}}\mathrm{D}\,\delta\mathbf{y}(T). \end{aligned} \tag{B.17}$$

Duhamel's formula (with $\delta\mathbf{y}(0) = 0$) yields

$$\delta\mathbf{y}(t) = \int_0^t e^{(t-s)\mathrm{A}}\mathrm{B}\delta\mathbf{u}(s)\,\mathrm{d}s; \tag{B.18}$$

hence, $\|\delta\mathbf{y}\|_{C^0} \leq C\|\delta\mathbf{u}\|_{L^2}$ where C is a constant that does not depend on $\delta\mathbf{u}$.
Furthermore, since $D = \begin{pmatrix} \lambda_P & 0 \\ 0 & \lambda_V \end{pmatrix}$, then $|\delta\mathbf{y}(T)^{\mathrm{T}}\mathrm{D}\delta\mathbf{y}(T)|^2 \leq \max(\lambda_P, \lambda_V)|\delta\mathbf{y}(T)|^2 \leq C\max(\lambda_P, \lambda_V)$.
So, $\delta\mathbf{y}(T)^{\mathrm{T}}\mathrm{D}\,\delta\mathbf{y}(T) = o(\delta\mathbf{u})$.
Therefore,

$$J_D(\mathbf{u}+\delta\mathbf{u}) = J_D(\mathbf{u}) + 2\mathbf{y}_{\mathbf{u}}(T)^{\mathrm{T}}\mathrm{D}\,\delta\mathbf{y}(T) + o(\delta\mathbf{u}), \tag{B.19}$$

hence

$$\langle \nabla J_D(\mathbf{u}), \delta\mathbf{u}\rangle = 2\mathbf{y}_{\mathbf{u}}(T)^{\mathrm{T}}\mathrm{D}\,\delta\mathbf{y}(T), \qquad \forall \mathbf{u} \in L^2([0,T], U). \tag{B.20}$$

Remembering the definition of the adjoint system for the LQR,

$$\begin{cases} \dot{\mathbf{p}}(t) = -\mathrm{A}^{\mathrm{T}}\mathbf{p}(t), \\ \mathbf{p}(T) = \mathrm{D}\mathbf{y}_{\mathbf{u}}(T), \end{cases} \tag{B.21}$$

we have

$$\begin{aligned} \langle \nabla J_D(\mathbf{p}), \delta\mathbf{u}\rangle &= 2\delta\mathbf{y}(T)^{\mathrm{T}}\mathbf{p}(T) \\ &= 2[\delta\mathbf{y}(t)^{\mathrm{T}}\mathbf{p}(t)]_0^T \\ &= 2\int_0^T \frac{\mathrm{d}}{\mathrm{d}t}\big(\delta\mathbf{y}(t)^{\mathrm{T}}\mathbf{p}(t)\big)\,\mathrm{d}t \\ &= 2\int_0^T \big[\delta\dot{\mathbf{y}}(t)^{\mathrm{T}}\mathbf{p}(t) + \delta\mathbf{y}(t)^{\mathrm{T}}\dot{\mathbf{p}}(t)\big]\,\mathrm{d}t \\ &= 2\int_0^T \big[(\mathrm{A}\delta\mathbf{y}(t) + \mathrm{B}\delta\mathbf{u}(t))^{\mathrm{T}}\mathbf{p}(t) - \delta\mathbf{y}(t)^{\mathrm{T}}\mathrm{A}^{\mathrm{T}}\mathbf{p}(t)\big]\,\mathrm{d}t \\ &= 2\int_0^T \delta\mathbf{u}(t)^{\mathrm{T}}\mathrm{B}^{\mathrm{T}}\mathbf{p}(t)\;\mathrm{d}t \\ &= \langle 2\mathrm{B}^{\mathrm{T}}\mathbf{p}, \delta\mathbf{u}\rangle. \end{aligned} \tag{B.22}$$

Therefore, the gradient of J is

$$\nabla J(\mathbf{u}) = 2\mathbf{u} + 2\mathrm{B}^{\mathrm{T}}\mathbf{p}. \tag{B.23}$$

B.3. **Choice of parameters for the numerical simulations of Section 3.**

m	λ_P	λ_V	T	Δt	N	U_L	U_R
1	100	50	1	0.02	50	-6	6

TABLE 1. Numerical parameters for the accelerated cart problem

B.4. **Choice of hyper-parameters for the numerical simulations of Section 3.4.** Table 2 details the choice of hyper-parameters for the PPO algorithm, where the following notations are adopted (see [21] for a better understanding of each parameter):

- γ is the discount factor as defined in Remark 2.1;
- λ_{GAE} is the λ factor used in the generalized advantage estimation;
- $\varepsilon_{\mathrm{clip}}$ is the clipping threshold;
- c_1 is the value function factor in the surrogate objective, that sets the balance between optimizing the policy and the value function;
- n_{steps} is the number of steps run per parallel environment between two updates of the policy;
- α is the learning rate used in the Adam algorithm applied to maximize the surrogate objective;
- the batch size refers to the number of transition samples used in the Adam algorithm;
- the number of epochs refers to the number of epochs used in the Adam algorithm;
- n_{env} is the number of environments that are run in parallel during training and evaluation.

Note that no entropy was used for the training. Also, we use the default choice of neural network architecture for PPO: an MLP network consisting of two layers of 64 neurons and a Hyperbolic Tangent as the activation function. At the end of this network, a policy output layer and a value output layer are appended, that share the outputs of the MLP network.

γ	λ_{GAE}	$\varepsilon_{\mathrm{clip}}$	c_1	n_{steps}	α	batchsize	epochs	n_{env}
0.99	0.95	0.2	0.5	50	0.001	100	10	8

TABLE 2. Hyper-parameters for the PPO algorithm

Appendix C. Existence of a globally optimal solution for the problem of Burgers

Proof of Theorem 4.2. Let us first remark that $J(u,a)$ is bounded on the set

$$\{\,(u,a)\,|\,a\in\mathcal{A},\ u \text{ solution to the forward problem (4.1)}\}\,.$$

Indeed, letting $a\in\mathcal{A}$ and u be the associated solution to the Burgers' forward problem, we have

$$J_\nu(u,a)\leq \beta\|u(T,\cdot)\|^2_{L^2(\Omega)}+\beta\|u^*(\cdot)\|^2_{L^2(\Omega)}+T|\Omega|\max(a_l^2,a_u^2); \tag{C.1}$$

by Poincaré's inequality, there exists $C_\Omega>0$ independent of u and a such that

$$\begin{aligned}\|u(T,\cdot)\|_{L^2(\Omega)}&\leq C_\Omega\|u(T,\cdot)\|_{H^1_0(\Omega)}\\&\leq C_EC_\Omega\|u(T,\cdot)\|_{W(0,T)}\quad\text{using the embedding }\|u\|_{C([0,T];H^1_0(\Omega))}\leq C_E\|u\|_{W(0,T)}\\&\leq C_\Omega C(1+\|a\|^2_{L^2((0,T)\times\Omega)})\quad\text{using Theorem 4.1}\\&\leq C_\Omega C(1+T|\Omega|\max(a_l^2,a_u^2));\end{aligned} \tag{C.2}$$

hence the boundedness of J follows. So, there exists a $\zeta\geq 0$ with

$$\zeta=\inf\{J_\nu(u,a)\,|\,(u,a)\in W(0,T)\times\mathcal{A}\text{ solves (4.1)}\}. \tag{C.3}$$

Therefore, there exists a minimizing sequence $(u^n,a^n)_{n\in\mathbb{N}}\in(W(0,T)\times\mathcal{A})^{\mathbb{N}}$ such that $\zeta=\lim_{n\to\infty}J_\nu(u^n,a^n)$ and (u^n,a^n) solves (4.1) for all $n\in\mathbb{N}$. Since $\mathcal{A}$ is bounded in $L^2((0,T)\times\Omega)$ as seen above, there exists $a^*_\nu\in L^2((0,T)\times\Omega)$ and $(a^{n_k})_{k\in\mathbb{N}}\in L^2((0,T)\times\Omega)^{\mathbb{N}}$ such that $a^{n_k}\rightharpoonup_{k\to\infty}a_\nu$ (where $\rightharpoonup$ denotes weak convergence in $L^2((0,T)\times\Omega)$). Likewise, from Theorem 4.1, we get the boundedness of $(u^{n_k})_{k\in\mathbb{N}}$ in $W(0,T)$, so $u^{n_k}\rightharpoonup_{k\to\infty}u_{a^*_\nu}$ where $u_{a^*_\nu}\in W(0,T)$ (possibly selecting a new subsequence). It was proven in [27] that the pair (u,a_ν) solves (4.1). Moreover, since $\mathcal{A}$ is closed and convex in $L^2((0,T)\times\Omega)$, then $a^*_\nu\in\mathcal{A}$. Lastly, J is weakly lower semicontinuous w.r.t. (u,a). Therefore, $J_\nu(u_{a^*_\nu},a^*_\nu)=\zeta$. Hence, a^*_ν is optimal. □

References

[1] N. Allahverdi, A. Pozo, and E. Zuazua. Numerical aspects of large-time optimal control of Burgers equation. *ESAIM: Mathematical Modelling and Numerical Analysis*, 50(5):1371–1401, Sept. 2016.

[2] H. Bateman. Some recent researches on the motion of fluids. *Monthly Weather Review*, 43(4):163–170, 1915.

[3] R. E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, 2010.

[4] D. P. Bertsekas. *Dynamic programming and stochastic control*, volume 125 of *Mathematics in Science and Engineering*. Academic Press [Harcourt Brace Jovanovich, Publishers], New York-London, 1976.

[5] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-dynamic programming*. Belmont, MA: Athena Scientific, 1996.

[6] A. Bryson. Optimal control-1950 to 1985. *IEEE Control Systems Magazine*, 16(3):26–33, June 1996. Conference Name: IEEE Control Systems Magazine.

[7] J. M. Burgers. A mathematical model illustrating the theory of turbulence. In *Advances in applied mechanics*, volume 1, pages 171–199. Elsevier, 1948.

[8] J. M. Burgers. Hydrodynamics.—application of a model system to illustrate some points of the statistical theory of free turbulence. In *Selected Papers of JM Burgers*, pages 390–400. Springer, 1995.

[9] L. C. Evans. *Partial differential equations*, volume 19 of *Graduate Studies in Mathematics*. American Mathematical Society, Providence, RI, 2nd edition, 2010.

[10] P. Garnier, J. Viquerat, J. Rabault, A. Larcher, A. Kuhnle, and E. Hachem. A review on deep reinforcement learning for fluid mechanics. *Computers & Fluids*, 225:104973, July 2021.

[11] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter. Control of a quadrotor with reinforcement learning. *IEEE Robotics and Automation Letters*, 2(4):2096–2103, Oct. 2017.

[12] T. Jaakkola, M. I. Jordan, and S. P. Singh. On the Convergence of Stochastic Iterative Dynamic Programming Algorithms. *Neural Computation*, 6(6):1185–1201, Nov. 1994.

[13] Z. Jiang, D. Xu, and J. Liang. A deep reinforcement learning framework for the financial portfolio management problem. *ArXiv:1706.10059*, 2017.

[14] R. E. Kálmán. Contributions to the theory of optimal control. *Boletín de la Sociedad Matemática Mexicana*, page 102, 1960.

[15] Y. Li, Y. Wen, D. Tao, and K. Guan. Transforming cooling optimization for green data center via deep reinforcement learning. *IEEE Transactions on Cybernetics*, 50(5):2002–2013, 2020.

[16] C.-L. Liu, C.-C. Chang, and C.-J. Tseng. Actor-critic deep reinforcement learning for solving job shop scheduling problems. *IEEE Access*, 8:71752–71762, 2020.

[17] M. Minsky. Steps toward Artificial Intelligence. *Proceedings of the IRE*, 49(1):8–30, 1961.

[18] I. P. Pavlov. *Conditioned reflexes: an investigation of the physiological activity of the cerebral cortex*. Conditioned reflexes: an investigation of the physiological activity of the cerebral cortex. Oxford Univ. Press, Oxford, England, 1927. Pages: xv, 430.

[19] L. Pontryagin. *Mathematical Theory of Optimal Processes*. Routledge, 1st edition, 1987.

[20] B. Recht. A tour of Reinforcement Learning: The view from continuous control. *Annual Review of Control, Robotics, and Autonomous Systems*, 2(1):253–279, 2019.

[21] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal Policy Optimization algorithms. *ArXiv:1707.06347*, 2017.
[22] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, Aug. 1988.
[23] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction, 2nd ed.* The MIT Press, Cambridge, MA, US, 2018.
[24] E. L. Thorndike. *Animal intelligence: experimental studies.* The Macmillan company, New York, 1911. Open Library ID: OL7172495M.
[25] N. Thuerey, P. Holl, M. Mueller, P. Schnell, F. Trost, and K. Um. *Physics-based Deep Learning.* 2021. Book available at `https://physicsbaseddeeplearning.org`.
[26] F. Tröltzsch and S. Volkwein. The SQP method for control constrained optimal control of the Burgers equation. *ESAIM Control Optim. Calc. Var.*, 6:649–674, 2001.
[27] S. Volkwein. Distributed Control Problems for the Burgers Equation. *Computational Optimization and Applications*, 18(2):115–140, Feb. 2001.
[28] C. Watkins. Learning from delayed rewards. 1989.
[29] C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992.
[30] S. K. Zhou, H. N. Le, K. Luu, H. V. Nguyen, and N. Ayache. Deep reinforcement learning in medical imaging: A literature review. *ArXiv:2103.05115*, 2021.
[31] Z. Zhu, K. Lin, A. K. Jain, and J. Zhou. Transfer Learning in Deep Reinforcement Learning: A Survey. *ArXiv:2009.07888*, May 2022.

Email address: `theilo.terrisse@eleves.enpc.fr`