Università di Genova
DIBRIS DIPARTIMENTO
DI INFORMATICA, BIOINGEGNERIA,
ROBOTICA E INGEGNERIA DEI SISTEMI

Approximate Solution Methods in Reinforcement Learning

Riccardo Cicala

Master's Thesis

Università di Genova, DIBRIS
Via Dodecaneso, 35 16146 Genova, Italy
https://www.dibris.unige.it/

Università
di Genova

Computer Science MSc
Data Science and Engineering Curriculum

Approximate Solution Methods in Reinforcement Learning

Riccardo Cicala

Advisor: Alessandro Verri

Examiner: Lorenzo Rosasco

March, 2026

Abstract

Reinforcement Learning is a powerful learning paradigm inspired by the way humans learn during childhood, through trial and error and exploration.

This framework has been extensively studied, and many approaches have been proposed to address reinforcement learning problems. With the advent of deep learning methods, the role of supervised techniques and function approximation has become particularly relevant and widely investigated.

This thesis investigates reinforcement learning with function approximation, focusing on the guarantees associated with approximate solutions and the factors that influence their quality. Particular attention is devoted to sampling strategies, as reinforcement learning inherently violates the independent and identically distributed (i.i.d.) assumption that underlies essentially all standard supervised learning theory.

Different sampling techniques are analyzed and combined with various machine learning approaches for value function approximation. Their performance is evaluated within a Linear Quadratic Regulator environment, allowing for a controlled and theoretically grounded comparison. The empirical results are critically analyzed to assess stability, approximation accuracy, and convergence behavior.

Table of Contents

Chapter 1

Introduction

When considering how complex behaviors are acquired by humans, a natural example is how a child learns to walk. This process does not rely on explicit instructions or a predefined model of movement. Instead, it emerges from continuous interaction with the surrounding environment. A child attempts to take a step, observes the outcome of that attempt, and adjusts subsequent actions accordingly. Through repeated testing of different movements across varying conditions and surfaces, the child progressively refines his behavior. What initially appears as a difficult and unstable task gradually becomes smooth and natural through this process of exploration and adaptation.

This type of problem can be addressed using a learning paradigm called Reinforcement Learning. The idea is to maximize an unknown reward function that comes from the environment. This reward function is usually impossible to calculate exactly and therefore impossible to maximize perfectly. For this reason, we typically refer to approximate solution methods in Reinforcement Learning that allow us to maximize this type of function approximately.

In this thesis, we will focus primarily on approximation methods that involve function approximation and solve the problem using supervised learning techniques. Even though these methods have already been studied and defined, there is still no satisfactory answer to some questions.

The principal problem is that, unlike standard supervised learning, the samples collected in reinforcement learning are not independent and identically distributed (i.i.d.), since each state and action in a trajectory depends on the previous ones through the environment dynamics and the policy. Furthermore, the distribution of states and actions can vary at each time step, meaning that the data collected in the early stages of an episode may follow a very different distribution from the data collected in later stages. These properties make the design and analysis of sampling methods critical, as poor sampling can lead to biased estimates or unstable learning.

The current literature is more focused on explaining methods and algorithms that solve the reinforcement learning problem using approximation techniques, or on identifying the algorithms that achieve the best performance with the smallest errors. Some works focus on theoretical guarantees of the algorithms and on the quality of the solution; however, there is relatively little focus on what the algorithms are actually learning and in which way.

The idea of this thesis is therefore to analyze two algorithms, Fitted Value Iteration (FVI) and Fitted Q-Iteration (FQI), using machine learning methods and different sampling techniques in a fully known environment such as a Linear Quadratic Regulator. In this way, we can compare our solutions with the true one and better understand what is happening behind the scenes, aiming to identify the guarantees required to learn a good solution from an experimental and empirical perspective.

Following this idea, in Chapter 2, we formalize the Reinforcement Learning paradigm and present the theoretical foundations underlying it. This allows us to clarify why approximate methods are necessary to address this class of problems.

In Chapter 3, we describe the methods employed in our experimental analysis. We begin with the algorithms Fitted Value Iteration (FVI) and Fitted Q-Iteration (FQI), then discuss the sampling techniques adopted, and finally present the machine learning models used for function approximation.

Moving forward, in Chapter 4, we introduce the Linear Quadratic Regulator environment and formally define the problem under consideration.

Finally, in Chapters 5 and 6, we analyze and discuss the experimental results, with the aim of understanding what the models are effectively learning and why. We conclude by summarizing the main findings and presenting the key takeaways of this work.

Chapter 2

Background

Reinforcement learning [Ber19] is a mathematical approach to sequential decision-making where an agent interacts with a dynamic system in order to achieve a goal. At each time step, the agent observes the current state of the system and chooses a control action. The system, usually called environment, responds by evolving to a new state and returning a measure of performance, called a reward. Over time, the agent learns to select actions that maximize its cumulative performance.

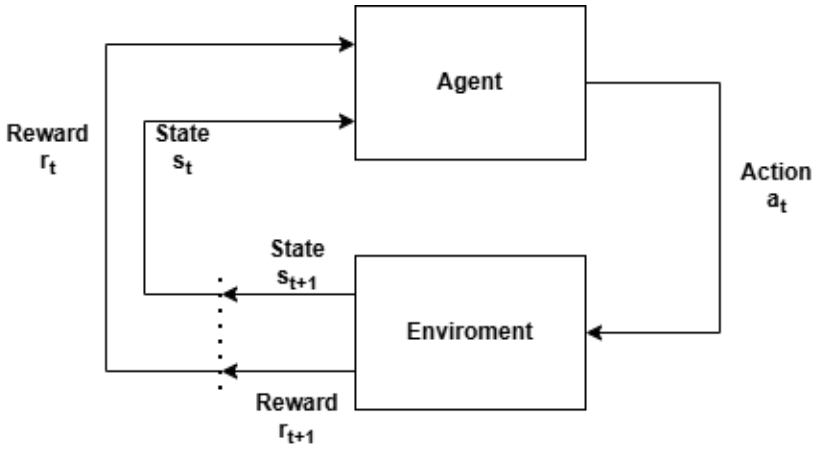


Figure 2.1: Dynamic system of reinforcement learning

Generally, states, rewards and actions are random variables, so they are sampled from a probability distribution.

The aim of reinforcement learning in such systems is not only to determine the immediate best action, but to design a policy that performs optimally over all the steps, taking into account system dynamics, constraints, and future consequences of current actions. If the probability distributions are knwon the problem can be solved directly by dynamic programming [Ber76], otherwise reinforcement learning provides a more complex paradigm in order to solve this problem. The key idea is that the agent learns from the environment, interacting with it and understanding when it makes mistakes through the reward mechanism. The agent, like a human being throughout its life, will try and explore various dynamics and choices, learning along the way and iteratively improving its knowledge. This perspective is formalized using the mathematical notion of Markov Decision Process

(MDP).

2.1 Markov Decision Process

First of all we need to define some notation:

- T horizon length (number of time steps)
- t discrete time index, $t = 0, 1, \ldots, T$
- n dimension of the state
- $\mathcal{S} \subseteq \mathbb{R}^n$ set of all the possible states
- $s_t \in \mathcal{S}$ state at time t
- m dimension of the action
- $\mathcal{A} \subseteq \mathbb{R}^m$ set of all the possible actions
- $a_t \in \mathcal{A}$ action at time t

We can now define our reward distribution as:

$$R_t(\cdot \mid s_t, a_t) \in \Delta(\mathbb{R}), \qquad r_{t+1} \sim R_t(\cdot \mid s_t, a_t). \tag{2.1}$$

where $\Delta(\cdot)$ represents all the possible distributions over a set.
This distribution specifies the immediate reward received at time t when taking action a_t in state s_t.
In our setting, both the actions and the states are continuous, and we consider only finite-horizon problems, that is, with $T < \infty$. In the finite-horizon case, the reward distribution may depend explicitly on time t.
At the terminal time T, there is no action to be taken, and the terminal reward depends only on the terminal state. We define:

$$R_T(\cdot \mid s_T) \in \Delta(\mathbb{R}), \qquad r_{T+1} \sim R_T(\cdot \mid s_T). \tag{2.2}$$

To simplify the notation and to make the terminal reward more similar to the other rewards, we also define a terminal reward that explicitly depends on the action, even though it is artificial and, in reality, the agent does not take any action.

$$R_T(\cdot \mid s_T, a_T) = R_T(\cdot \mid s_T) \quad \forall a_T \in \mathcal{A} \tag{2.3}$$

The key property is that the system evolves without memory (Markov property), so the transition distribution is defined as:

$$\mathcal{P}(\cdot \mid s_t, a_t) \in \Delta(\mathcal{S}), \qquad s_{t+1} \sim \mathcal{P}(\cdot \mid s_t, a_t). \tag{2.4}$$

In this formulation, we omit the joint distribution between the reward and the state transition, assuming that the reward and transition dynamics are statistically independent. This simplification is common among Markov Decision Process models and allows us to treat the reward function and transition model independently during the definition of our problem.

Since the reward function is time-dependent, it should always formally be denoted by R_t. However, to simplify the notation, throughout this thesis R should be understood as denoting the collection of reward distributions $\{R_t\}_{t=0}^{T}$, with the appropriate R_t implicitly selected at each time step t.

Our Markov decision process is defined as the 4-tuple $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, R, \mathcal{P}\}$.

Thanks to this formalization, we can introduce the building blocks necessary to understand the reinforcement learning paradigm. The central idea is to define a policy that specifies how the agent should behave based on the situation it finds itself in. We define a stochastic, time-dependent, policy as a probability distribution over actions conditioned on the current state:

$$\pi_t(\cdot \mid s_t) \in \Delta(\mathcal{A}), \qquad a_t \sim \pi_t(\cdot \mid s_t). \tag{2.5}$$

As with the reward function, throughout this thesis the notation π denotes the collection of time-indexed policies $\{\pi_t\}_t$, with the appropriate π_t implicitly selected at each time step t.

Using this concept of policy, we can define a metric to evaluate the quality of a policy and formalize the decision-making objective in reinforcement learning. We define the state-value function as the expected cumulative reward starting from a state s at time t, with future rewards discounted by a factor $\gamma \in [0, 1]$:

$$V_t^\pi(s) = \mathbb{E}_{\pi,R,\mathcal{P}}\Big[\sum_{k=t}^{T} \gamma^{k-t} r_{k+1} \mid s_t = s\Big] \tag{2.6}$$

Here, the expectation $\mathbb{E}_{\pi,R,\mathcal{P}}[\cdot]$ is taken over all sources of randomness: the stochastic policy $a_k \sim \pi_k(\cdot \mid s_k)$, the stochastic rewards $r_{k+1} \sim R_k(\cdot \mid s_k, a_k)$, and the stochastic state transitions $s_{k+1} \sim \mathcal{P}(\cdot \mid s_k, a_k)$.
Usually in finite-horizon problems the discount factor is not included, setting $\gamma = 1$ recovers the standard finite-horizon formulation without discounting future rewards.

Another closely related metric is the action-value function, which represents the expected cumulative reward starting from state s at time t, after taking the initial action a, while all subsequent actions from time $t + 1$ onward are determined by the stochastic policy.

$$Q_t^\pi(s, a) = \mathbb{E}_{\pi,R,\mathcal{P}}\Big[\sum_{k=t}^{T} \gamma^{k-t} r_{k+1} \mid s_t = s, a_t = a\Big] \tag{2.7}$$

Having introduced the state-value function $V_t^\pi(s)$, describing the expected cumulative reward obtained by following a policy π, we now focus on the concept of optimality. The optimal value function is defined as the supremum of the value functions over all admissible policies. Formally:

$$V_t^*(s) = \sup_\pi V_t^\pi(s) \quad \forall s \in \mathcal{S} \tag{2.8}$$

Intuitively, $V_t^*(s)$ represents the maximum expected return achievable from state s at time t, when acting optimally over the remaining horizon.

In many cases of interest, the supremum is attained and may therefore be written as a maximum, from now on we will assume an optimal policy exists and that is deterministic.

This deterministic assumption is without loss of generality: at each time step whenever the maximum over all actions is attained, we can form a deterministic optimal policy by assigning probability 1 to any action that achieves the maximal expected value (computed with respect to the stochasticity of the transitions and rewards), and probability 0 to all other actions. Hence, randomization is unnecessary for attaining the optimal value.

In general, the supremum may fail to be attained when the structure of the action set introduces restrictions, for example, when the set is not closed or when certain actions cannot be taken in a given state due to the problem's construction (e.g., state-dependent constraints that remove some actions). In such cases, no action actually achieves the supremum, and an optimal policy may not exist. Throughout this thesis, we will not consider these pathological situations and will assume that the maximum is always attained.

We can define the same optimality concept also for the action-value function, as:

$$Q_t^*(s,a) = \sup_\pi Q_t^\pi(s,a) \quad \forall s \in \mathcal{S}, \forall a \in \mathcal{A} \tag{2.9}$$

2.2 Bellman Equation

A remarkable property of any state-value function is that it satisfies the following recursive relation, known as the Bellman equation:

$$V_t^\pi(s) = \mathbb{E}_{\pi,R,\mathcal{P}}\Big[r_{t+1} + \gamma V_{t+1}^\pi(s_{t+1}) \mid s_t = s\Big] \tag{2.10}$$

This recursion is a direct consequence of the Markov property, which ensures that both the immediate reward and the next state s_{t+1} depend only on the current state-action pair (s_t, a_t), and not on the full history.

To see why it holds, we recall 2.6 and separate the immediate reward from the rest of the future return:

$$V_t^\pi(s) = \mathbb{E}_{\pi,R,\mathcal{P}}\Big[r_{t+1} + \sum_{k=t+1}^{T} \gamma^{k-t} r_{k+1} \mid s_t = s\Big] \tag{2.11}$$

Because of the Markov property, the distribution of future rewards (from time $t+1$ onward) depends only on the next state s_{t+1}. Therefore using the law of total expectation:

$$V_t^\pi(s) = \mathbb{E}_{\pi,R,\mathcal{P}}\Big[r_{t+1} + \gamma \mathbb{E}_{\pi,R,\mathcal{P}}\Big[\sum_{k=t+1}^{T} \gamma^{k-(t+1)} r_{k+1} \mid s_{t+1}\Big] \mid s_t = s\Big] \tag{2.12}$$

But the inner expectation is precisely the state-value function at the next time step, as defined earlier:

$$\mathbb{E}_{\pi,R,\mathcal{P}}\Big[\sum_{k=t+1}^{T} \gamma^{k-(t+1)} r_{k+1} \mid s_{t+1}\Big] = V_{t+1}^\pi(s_{t+1}) \tag{2.13}$$

Substituting this expression yields the Bellman equation 2.10. Thus the Bellman equation compactly expresses the state-value function at time t as the expected immediate reward plus the discounted state-value function at time $t+1$.

Under the assumption that an optimal deterministic policy exists, the optimal state-value function satisfies the Bellman optimality equation:

$$V_t^*(s) = \max_{a \in \mathcal{A}} \left\{ \bar{r}_t(s,a) + \gamma\, \mathbb{E}_{s_{t+1} \sim \mathcal{P}(\cdot|s,a)} \left[V_{t+1}^*(s_{t+1}) \right] \right\} \tag{2.14}$$

Where $\bar{r}_t(s,a)$ is defined as the expected value of the reward, formally:

$$\bar{r}_t(s,a) = \mathbb{E}_{r_t \sim R(\cdot|s,a)}[r_t] \tag{2.15}$$

Crucially, the Bellman equation characterizes the optimal state-value function without any explicit reference to policies; once $V_t^*(s)$ is known, an optimal policy $\pi_t^*(s)$ can be recovered by selecting actions that maximize the optimal Bellman equation (Equation (2.14)), this is called Bellman optimality condition:

$$\pi_t^*(s) \in \underset{a \in \mathcal{A}}{\operatorname{argmax}} \left\{ \bar{r}_t(s,a) + \gamma \, \mathbb{E}_{s_{t+1} \sim \mathcal{P}(\cdot|s,a)} \left[V_{t+1}^*(s_{t+1}) \right] \right\} \tag{2.16}$$

Therefore, even in cases where the policy class is complex or the rewards and the state transitions distributions are unknown, the Bellman equation provides a conceptual and computational framework for optimal decision-making.

The same concept can be applied to the action-value function, so we can write it recursively as:

$$Q_t^\pi(s,a) = \mathbb{E}_{\pi,R,\mathcal{P}} \Big[r_{t+1} + \gamma Q_{t+1}^\pi(s_{t+1}, a_{t+1}) \mid s_t = s, a_t = a \Big] \tag{2.17}$$

We can demonstrate that this formulation is correct similarly to before, so we have to recall 2.7 and split the first reward:

$$Q_t^\pi(s,a) = \mathbb{E}_{\pi,R,\mathcal{P}} \Big[r_{t+1} + \sum_{k=t+1}^{T} \gamma^{k-t} r_{k+1} \mid s_t = s, a_t = a \Big] \tag{2.18}$$

Then we know that future rewards depend only on the s_{t+1} state, and thanks to the law of total expectation we can also add the action a_{t+1} as given.

$$Q_t^\pi(s,a) = \mathbb{E}_{\pi,R,\mathcal{P}} \Big[r_{t+1} + \gamma \mathbb{E}_{\pi,R,\mathcal{P}} \Big[\sum_{k=t+1}^{T} \gamma^{k-(t+1)} r_{k+1} \mid s_{t+1}, a_{t+1} \Big] \mid s_t = s, a_t = a \Big] \tag{2.19}$$

Finally we can define the inner expectation as the next action-value function:

$$\mathbb{E}_{\pi,R,\mathcal{P}} \Big[\sum_{k=t+1}^{T} \gamma^{k-(t+1)} r_{k+1} \mid s_{t+1}, a_{t+1} \Big] = Q_{t+1}^\pi(s_{t+1}, a_{t+1}) \tag{2.20}$$

We can also define the optimal action-value function as:

$$Q_t^*(s,a) = \bar{r}_t(s,a) + \gamma \mathbb{E}_{s_{t+1} \sim \mathcal{P}(\cdot|s,a)} \Big[\max_{a^* \in \mathcal{A}} \left\{ Q_{t+1}^*(s_{t+1}, a^*) \right\} \Big] \tag{2.21}$$

Here, the maximum is taken inside the expectation because we do not want to select a single action that is optimal for all possible next states s_{t+1}. Instead, we choose the optimal action for each specific state s_{t+1}, after it has been realized. Therefore the optimal action-value function represents the maximum expected return achievable starting from state s after taking action a, assuming that all subsequent actions are chosen optimally.

The relation between the optimal state-value function and the action-value function is simply:

$$V_t^*(s) = \max_{a \in \mathcal{A}} Q_t^*(s,a) \tag{2.22}$$

which reflects the fact that the first action on the action-value function is, also, chosen optimally, after which all subsequent actions also follow the optimal policy.

Given the optimal action-value function, the optimal policy can be recovered as:

$$\pi_t^*(s) \in \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, Q_t^*(s, a) \tag{2.23}$$

We can clearly see that, if our goal is to retrieve the optimal policy, it is often preferable to rely on the action-value function. This is because it quantifies the expected return associated with each possible control action in a given state, providing a direct way to select the optimal control input. Moreover, this formulation avoids the need to compute additional expected-value terms over actions, which are required when working with the state-value function. This is particularly advantageous in settings where the reward distribution and the transition dynamics are unknown and observable only thanks to the environment, as it reduces the propagation of estimation errors and leads to more stable policy extraction.

2.3 Dynamic programming

Thanks to the Bellman equation, we can characterize the optimal policy, which not only selects actions that maximize the immediate reward but also accounts for all future consequences induced by the current decision.

In the absence of stochasticity, the Bellman equation allows the problem to be reformulated as a standard dynamic programming problem. The main idea in this setting is to adopt a bottom-up approach: first solving the optimization problem at the final time step and then recursively working backward through the preceding steps.

This perspective enables a complex global maximization problem to be decomposed into a sequence of simpler optimization problems, a property that directly follows from the recursive structure of the Bellman equation.

To illustrate this idea, we consider a simple deterministic example in which both the reward function and the system dynamics are known.

2.3.1 Battery management problem

This example illustrates a simple battery management problem. The system admits three possible battery states: high, medium, and low. The agent can choose among three actions: charging the battery, using the device intensively, or using it lightly. We assume a finite horizon of length two.

The horizon can be interpreted as the number of hours during which the agent uses the device. The time index t denotes the hour at which the agent begins using the device, while the actions represent the manner in which the device is used over a single hour.

In this example we assume that the reward function is time-invariant, except for the terminal reward.

To summarize:

- $T = 2$
- $t \in \{0, 1, 2\}$
- $\mathcal{S}$ ={high, medium, low}
- $\mathcal{A}$ ={intense use, light use, charge}

The reward and the evolution functions are defined as follows:

s	a	$R(s,a)$	$\mathcal{P}(s,a)$
high	intense use	0	medium
high	light use	0	high
high	charge	-1	high
medium	intense use	-1	low
medium	light use	-1	medium
medium	charge	-2	high
low	intense use	-2	low
low	light use	-2	low
low	charge	-3	medium

Table 2.1: The reward and the evolution functions of the battery management problem

The terminal reward is defined as follows:

s	$R_T(s)$
high	-0
medium	-3
low	-5

Table 2.2: The terminal reward function of the battery management problem

The Bellman equation, together with the bottom-up approach, provides a framework for designing a policy that specifies, for each time step t and each state s, the optimal action for the agent.

At time $t = T = 2$, no action is taken. Consequently, the state-value function and the action-value function coincide and are equal to the terminal reward associated with the final state. In this case, it is not meaningful to define an optimal policy.

At time $t = T - 1 = 1$, the action-value function is computed as :

$$Q_1(s,a) = R(s,a) + R_T(\mathcal{P}(s,a)) \tag{2.24}$$

s	a	$Q_1(s,a)$
high	intense use	-3
high	light use	0
high	charge	-1
medium	intense use	-5
medium	light use	-3
medium	charge	-1
low	intense use	-5
low	light use	-5
low	charge	-4

Table 2.3: Values of the $Q_1(s,a)$

The optimal policy at time $t = 1$ can be obtained by applying Equation (2.23):

s	$\pi_1^*(s)$
high	light use
medium	charge
low	charge

Table 2.4: Values of the $\pi_1^*(s)$

By combining Equations 2.21 and 2.22, we can express the optimal action-value function at time $t = 0$ as the sum of two terms, defined as follows:

$$Q_0^*(s,a) = R(s,a) + V_1^*(\mathcal{P}(s,a)) \tag{2.25}$$

We can calculate the optimal state-value function at time $t = 1$ using the Equation (2.22):

s	$V_1^*(s)$
high	0
medium	-1
low	-4

Table 2.5: Values of the $V_1^*(s)$

Thus, the optimal action-value function at time $t = 0$ is equal to:

s	a	$Q_0^*(s,a)$
high	intense use	-1
high	light use	0
high	charge	-1
medium	intense use	-4
medium	light use	-1
medium	charge	-1
low	intense use	-4
low	light use	-4
low	charge	-2

Table 2.6: Values of the $Q_0^*(s)$

As before, we can now compute the optimal policy at time $t = 0$:

s	$\pi_0^*(s)$
high	light use
medium	light use/charge
low	charge

Table 2.7: Values of the $V_1^*(s)$

In this way, we have reconstructed the optimal policy for every time step t. If the horizon length is extended, the same algorithm can be applied to compute the optimal policy at each time step using the Bellman optimality equation.

The main advantage of this approach is that it avoids enumerating all possible trajectories starting from a given time step t and state s in order to determine the optimal action. Since the number of such trajectories grows exponentially with the number of time steps, actions, and states, this exhaustive approach quickly becomes intractable.

Instead, the problem is decomposed into a sequence of simpler computations, each involving a sum of two terms, by recursively evaluating the state-value functions backward in time.

To illustrate this process more concretely, we show all possible trajectories starting from $t = 0$ and state $s =$low, together with their cumulative rewards, in Figure 2.2

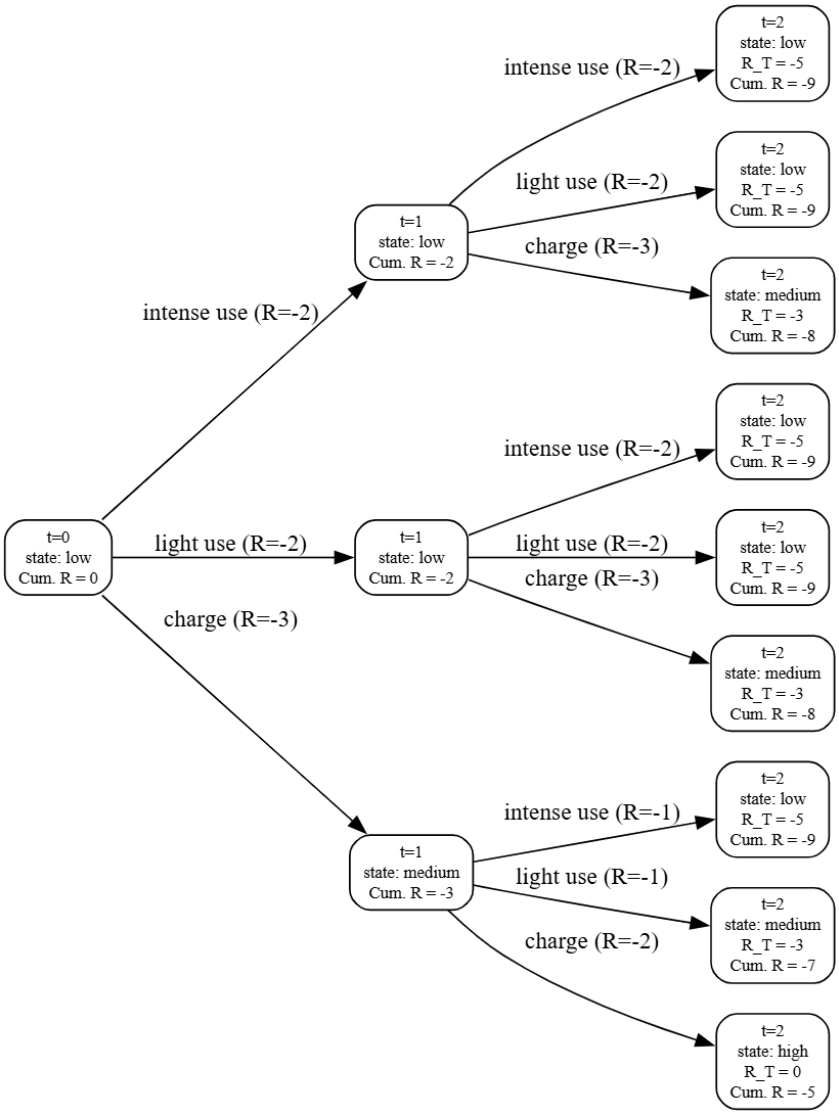


Figure 2.2: Trajectories battery management problem

We can clearly see that the optimal trajectory, which maximizes the cumulative reward, is the one in which the device is charged at both time steps, in agreement with the optimal policy calculated previously.

2.4 Approximate solution

While the Bellman optimality equation and the dynamic programming paradigm provide a complete characterization of the optimal action-value or state-value function and the corresponding policy, in most practical settings they cannot be solved analytically due to several intertwined challenges.

In the previous section, we presented an example of dynamic programming in a deterministic setting, where no stochasticity is present. When randomness is introduced into the problem, the same framework can still be applied in principle, provided that the underlying probability distributions are known. In such cases, the expectations appearing

in the Bellman equations can be computed exactly.

Even when these distributions are unknown, the problem remains theoretically tractable, as the required expectations can be approximated using Monte Carlo simulation. Therefore, neither stochasticity nor unknown model dynamics alone constitute the main source of intractability. The primary difficulty arises when the size of the state and action spaces becomes large. In this regime, computing Monte Carlo estimates of the expected returns for all state-action pairs becomes computationally infeasible.

Another major challenge arises, even when the model is known and Monte Carlo simulation is therefore unnecessary, when the state or action spaces become continuous or excessively large, making a full enumeration of the state-value or action-value functions infeasible. It is important to note that there are notable exceptions, which will be discussed in a later chapter, where under specific structural assumptions even in continuous state and action spaces it is possible to compute the Bellman optimality equation exactly.

These limitations motivate the use of approximation methods, which aim to estimate the state-value function, action-value function, or the policy without computing the Bellman equation exactly. Such approaches simplify the problem either by reducing the effective dimensionality of the state or action spaces, or by estimating the relevant functions directly from data.

The key idea is to replace certain components of the Bellman equation with simpler approximate counterparts in order to make the computation tractable. For instance, heuristic approximations can be employed, such as manually designed rules that replace the optimal state-value function V^*_{t+1} in Equation (2.16) with an approximate version $\tilde{V}^*_{t+1}$.

Another class of methods relies on state or action aggregation techniques, such as clustering, which group similar states or actions together to reduce the number of distinct elements considered. This results in replacing the original state and action spaces $\mathcal{S}$ and $\mathcal{A}$ with reduced spaces $\tilde{\mathcal{S}}$ and $\tilde{\mathcal{A}}$.

Finally, function approximation methods, which will be the focus of this thesis, can be used to estimate the state-value function, the action-value function, or the policy directly from sampled trajectories. In this setting, the optimal functions are replaced by a data-driven estimates. For example in Equation (2.23), the optimal action-value function Q^*_t is replaced by its estimate $\hat{Q}^*_t$.

These strategies, representing only a subset of the many possible approaches and often applied in more complex forms than illustrated here, enable reinforcement learning algorithms to operate effectively in all settings where exact computation of the Bellman equation is infeasible.

Chapter 3

Methodology

In this chapter, we describe the approximate methods considered in this work, together with their implementation details, all of them are applied to a finite-horizon settings with continuous states and actions spaces. All the algorithms presented are well-established in the literature and do not constitute original contributions. The primary objective of this thesis is instead to combine and apply these methodologies to a representative reinforcement learning problem, in order to gain a deeper understanding of their behavior and underlying mechanisms, rather than solely focusing on identifying the most efficient or best-performing approach.

We begin by introducing the Fitted Value Iteration algorithm (Section 3.1) and the Fitted Q-Iteration algorithm (Section 3.2), which address the problem of approximating state-value and action-value functions from sampled data. We then present several sampling techniques (Section 3.3) that support these algorithms by improving generalization, since in reinforcement learning settings the independent and identically distributed (i.i.d.) assumption is violated. Finally, we describe the machine learning methods employed to approximate the value functions in practice, including the function approximators and the training pipeline (Sections 3.4 and 3.5).

3.1 Fitted Value Iteration

The central idea behind Fitted Value Iteration is to approximate the optimal state-value function by evaluating the optimal Bellman equation for state values (Equation (2.14)) on a finite set of representative states. In FVI, the exact optimal state-value function V_t^* is replaced by a learned approximation $\hat{V}_t^*$, formally we can define an approximate version $\tilde{V}_t^*$ of the true state-value function as:

$$\tilde{V}_t^*(s) = \max_{a \in \mathcal{A}} \left\{ \bar{r}_t(s,a) + \gamma \, \mathbb{E}_{s_{t+1} \sim \mathcal{P}(\cdot|s,a)} \left[\hat{V}_{t+1}^*(s_{t+1}) \right] \right\} \tag{3.1}$$

From this point onward, we adopt a mild abuse of notation. When writing Monte Carlo simulations of expectations, we use the same symbol to denote both a probability distribution and the corresponding sampling operation, treating it as a function. This reflects the fact that the environment is accessed as a black box that generates samples according

to the underlying distributions. In particular:

$$R_t(s,a) = r_{t+1} \sim R_t(\cdot \mid s,a), \quad \mathcal{P}(s,a) = s_{t+1} \sim \mathcal{P}(\cdot \mid s,a) \tag{3.2}$$

To learn $\hat{V}_t^*$, FVI relies on a collection of sampled states $\{s^{(i,t)}\}_{i=1}^N$ drawn from a chosen distribution μ_t over the state space. Assuming that the true distributions of rewards and state transitions are unknown, the Bellman targets for each sampled state are estimated via Monte Carlo sampling:

$$\tilde{V}_t^*(s^{(i,t)}) = y^{(i,t)} = \max_{a\in\mathcal{A}} \Big\{ \big(\frac{1}{K}\sum_{k=1}^{K} R_t^{(k)}(s^{(i,t)},a)\big) + \gamma \hat{V}_{t+1}^*\big(\frac{1}{K}\sum_{k=1}^{K} \mathcal{P}^{(k)}(s^{(i,t)},a)\big)\Big\} \tag{3.3}$$

where K is the number of Monte Carlo simulations.

Here, the maximum cannot be computed exactly, since the true distributions of rewards and state transitions are unknown. Therefore, we use a nonlinear optimization method to maximize the function, specifically the Powell algorithm [Pow64].

These target values are then fitted with a function approximator, enabling the construction of an approximate state-value function that generalizes beyond the sampled points. In this thesis, we consistently use a least-squares approach to learn the function approximation, so $\hat{V}_t^*$ is obtained as:

$$\hat{V}_t^* \leftarrow \operatorname*{argmin}_{f\in\mathcal{F}_\mathcal{V}} \frac{1}{N}\sum_{i=1}^{N} \big(f(s^{(i,t)}) - y^{(i,t)}\big)^2 \tag{3.4}$$

where $\mathcal{F}_\mathcal{V}$ represents the function space from which the state-value function approximation $\hat{V}_t^*$ is chosen.

Following the dynamic programming paradigm 2.3, it is possible to construct an algorithm that, using a bottom-up approach, reconstructs the state-value function for each time step t; this procedure corresponds to the Fitted Value Iteration algorithm 1.

Once the approximate state-value function $\hat{V}_t^*$ has been learned, an approximate optimal policy can be obtained from the Bellman optimality condition (Equation (2.16)) by replacing the exact expectations with Monte Carlo estimates and the true state-value function V_t^* with its approximation. Formally, the resulting policy is defined as:

$$\tilde{\pi}_t^*(s) = \operatorname*{argmax}_{a\in\mathcal{A}} \Big[\big(\frac{1}{K}\sum_{k=1}^{K} R_t^{(k)}(s,a)\big) + \gamma \hat{V}_{t+1}^*\big(\frac{1}{K}\sum_{k=1}^{K} \mathcal{P}^{(k)}(s,a)\big)\Big] \tag{3.5}$$

As in the previous case, the maximization is performed using the Powell optimization algorithm and Monte Carlo sampling.

In this particular implementation, we additionally introduce a function approximator $\hat{\pi}_t^*$ to represent the approximated policy $\tilde{\pi}_t^*$. This approximation is obtained by solving:

$$\hat{\pi}_t^* \leftarrow \operatorname*{argmin}_{f\in\mathcal{F}_\pi} \frac{1}{N}\sum_{i=1}^{N} \big(f(s^{(i,t)}) - \tilde{\pi}_t^*(s^{(i,t)})\big)^2 \tag{3.6}$$

where $\mathcal{F}_\pi$ represents the function space from which the policy approximation $\hat{\pi}_t^*$ is chosen. This concept is not new in reinforcement learning and is known as the actor-critic method

[Sut+99], since it combines an actor (the learned policy $\hat{\pi}_t^*$) with a critic (the learned state-value function $\hat{V}_t^*$).

It is important to distinguish between two terminologies in reinforcement learning. An approach that uses $\tilde{\pi}_t^*$ is considered an online method, since the agent must interact with the environment to select actions, as the reward distributions and state-transition dynamics are unknown. In contrast, using $\hat{\pi}_t^*$ corresponds to an offline method, because the agent can determine the action for any given state without interacting with the environment, as the learned policy depends only on the current state.

Another important distinction in reinforcement learning is between model-free and model-based algorithms. Since the FVI algorithm does not attempt to replicate the reward or state-transition distributions, it is considered a model-free method. In contrast, if an algorithm explicitly constructs or uses a model of the environment's dynamics to find an optimal policy, it is referred to as model-based.

Algorithm 1 Finite-Horizon FVI

Input: Horizon T, Discount factor γ, Environment rewards $\{R_t\}_{t=0}^{T}$ (unknown, accessed via interaction), Environment transition dynamics $\mathcal{P}$ (unknown, accessed via interaction), State samples $\{\{s^{(i,t)}\}_{i=1}^{N}\}_{t=0}^{T}$, Function space for the state-value function $\mathcal{F}_{\mathcal{V}}$, Function space for the policy $\mathcal{F}_{\pi}$, Number of Monte Carlo simulations K

for each sampled state $s^{(i,T)}$ **do**

Compute the target value

$$\tilde{V}_t^*(s^{(i,T)}) = y^{(i,T)} = \frac{1}{K}\sum_{k=1}^{K} R_T^{(k)}(s^{(i,T)})$$

end for

Fit the value function:

$$\hat{V}_T^* \leftarrow \underset{f\in\mathcal{F}_{\mathcal{V}}}{\operatorname{argmin}} \frac{1}{N}\sum_{i=1}^{N}\big(f(s^{(i,T)}) - \tilde{V}_t^*(s^{(i,T)})\big)^2$$

for $t = T-1, T-2, \ldots, 0$ **do**

for each sampled state $s^{(i,t)}$ **do**

Compute the optimal action

$$a^{*(i,t)} = \underset{a\in\mathcal{A}}{\operatorname{argmax}} \Big[\big(\frac{1}{K}\sum_{k=1}^{K} R_t^{(k)}(s^{(i,t)}, a)\big) + \gamma \hat{V}_{t+1}^*\big(\frac{1}{K}\sum_{k=1}^{K}\mathcal{P}^{(k)}(s^{(i,t)}, a)\big)\Big]$$

Compute the target value

$$\tilde{V}_t^*(s^{(i,t)}) = y^{(i,t)} = \big(\frac{1}{K}\sum_{k=1}^{K} R_t^{(k)}(s^{(i,t)}, a^{*(i,t)})\big) + \gamma \hat{V}_{t+1}^*\big(\frac{1}{K}\sum_{k=1}^{K}\mathcal{P}^{(k)}(s^{(i,t)}, a^{*(i,t)})\big)$$

end for

Fit the value function:

$$\hat{V}_t^* \leftarrow \underset{f\in\mathcal{F}_{\mathcal{V}}}{\operatorname{argmin}} \frac{1}{N}\sum_{i=1}^{N}\big(f(s^{(i,t)}) - \tilde{V}_t^*(s^{(i,t)})\big)^2$$

Fit the policy function:

$$\hat{\pi}_t^* \leftarrow \underset{f\in\mathcal{F}_{\pi}}{\operatorname{argmin}} \frac{1}{N}\sum_{i=1}^{N}\big(f(s^{(i,t)}) - a^{*(i,t)}\big)^2$$

end for

Output: Sequence of fitted state-value functions $\{\hat{V}_t^*\}_{t=0}^{T}$, Sequence of fitted policy $\{\hat{\pi}_t^*\}_{t=0}^{T-1}$,

The optimal approximated policy at stage t:

$$\tilde{\pi}_t^*(s) = \underset{a\in\mathcal{A}}{\operatorname{argmax}} \Big[\big(\frac{1}{K}\sum_{k=1}^{K} R_t^{(k)}(s, a)\big) + \gamma \hat{V}_{t+1}^*\big(\frac{1}{K}\sum_{k=1}^{K}\mathcal{P}^{(k)}(s, a)\big)\Big]$$

3.2 Fitted Q-Iteration

Similar to the FVI alghorithm (Section 3.1), the main idea of the Fitted Q-Iteration (FQI) is to approximate the optimal action-value function by evaluating the optimal Bellman equation for action values (Equation (2.21)) on a finite set of representative state-action pairs. In FQI, the exact optimal action-value function Q_t^* is replaced by a learned approximation $\hat{Q}_t^*$, formally we can define an approximate version $\tilde{Q}_t^*$ of the true action-value function as:

$$\tilde{Q}_t^*(s,a) = \bar{r}_t(s,a) + \gamma\, \mathbb{E}_{s_{t+1}\sim\mathcal{P}(\cdot|s,a)} \left[\max_{a\in\mathcal{A}} \left\{ \hat{Q}_{t+1}^*(s_{t+1},a) \right\} \right] \tag{3.7}$$

To learn $\hat{Q}_t^*$, FQI relies on a collection of state-action sample pairs $\{(s^{(i,t)}, a^{(i,t)})\}_{i=1}^N$ drawn from a chosen distribution μ_t over the state-action space. Assuming that the true distributions of rewards and state transitions are unknown, the Bellman targets for each sampled are estimated via Monte Carlo sampling:

$$\tilde{Q}_t^*(s^{(i,t)}, a^{(i,t)}) = y^{(i,t)} = \frac{1}{K}\sum_{k=1}^{K} R_t^{(k)}(s^{(i,t)}, a^{(i,t)}) + \gamma \max_{a\in\mathcal{A}} \left\{ \hat{Q}_{t+1}^*\Big(\frac{1}{K}\sum_{k=1}^{K} \mathcal{P}^{(k)}(s^{(i,t)}, a^{(i,t)}), a\Big) \right\} \tag{3.8}$$

where K is the number of Monte Carlo simulations.

Here, in contrast to Fitted Value Iteration (FVI), the maximization can be computed exactly, since the only function appearing inside the maximization is the approximated action-value function. Depending on the function approximator used, the maximum can be computed explicitly. It is important to underline that this is possible only because the next state s_{t+1} can be generated via Monte Carlo simulation, and the maximization is performed only after the next states have been sampled, as indicated by Equation (3.7).

These target values are then fitted using a least-squares approach, so $\hat{Q}_t^*$ is obtained as:

$$\hat{Q}_t^* \leftarrow \operatorname*{argmin}_{f\in\mathcal{F}_\mathcal{Q}} \frac{1}{N}\sum_{i=1}^{N} \big(f(s^{(i,t)}, a^{(i,t)}) - y^{(i,t)}\big)^2 \tag{3.9}$$

where $\mathcal{F}_\mathcal{Q}$ represents the function space from which the action-value function approximation $\hat{Q}_t^*$ is chosen. It is important to emphasize that, at the terminal step, the approximated action-value function $\hat{Q}_T^*$ must belong to a specific subspace of $\mathcal{F}_Q$ in which the action variable is not taken into account. Although our notation allows the terminal reward to be expressed as depending on the final action, the terminal Bellman target is independent of the action, since no further actions can be taken, as indicated by Equation (2.3).

We denote this terminal subspace by $\mathcal{F}_{Q_T} \subset \mathcal{F}_Q$.

Following the dynamic programming paradigm 2.3, it is possible to construct an algorithm that, using a bottom-up approach, reconstructs the action-value function for each time step t; this procedure corresponds to the Fitted-Q Iteration algorithm 2.

Once the approximate action-value function $\hat{Q}_t^*$ has been learned, an approximate optimal policy can be obtained from the Bellman optimality condition (Equation (2.23))

by replacing the true action-value function Q_t^* with its approximation. Formally, the resulting policy is defined as:

$$\tilde{\pi}_t^*(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, \hat{Q}_t^*(s, a) \tag{3.10}$$

In contrast to Fitted Value Iteration (FVI), the argmax in this expression can be computed exactly, as it depends solely on the chosen function approximator for the action-value function.

Similar to FVI, we further introduce a function approximator $\hat{\pi}_t^*$ to represent the approximated policy $\tilde{\pi}_t^*$. This approximation is obtained by solving:

$$\hat{\pi}_t^* \leftarrow \underset{f \in \mathcal{F}_\pi}{\operatorname{argmin}} \frac{1}{N} \sum_{i=1}^{N} \left(f(s^{(i,t)}) - \tilde{\pi}_t^*(s^{(i,t)}) \right)^2 \tag{3.11}$$

where $\mathcal{F}_\pi$ denotes the function space from which the policy approximation $\hat{\pi}_t^*$ is selected.

As in FVI, Fitted Q-Iteration (FQI) is a model-free method.

This formulation highlights why, in model-free methods, it is often preferable to work with the action-value function. The policy in Equation (3.10), which is based on the action-value function, can be computed directly. In contrast, Equation (3.5), which relies on the state-value function, requires solving a nonlinear optimization problem via Monte Carlo simulation when the reward and transition distributions are unknown.

Algorithm 2 Finite-Horizon FQI

Input: Horizon T, Discount factor γ, Environment rewards $\{R_t\}_{t=0}^{T}$ (unknown, accessed via interaction), Environment transition dynamics $\mathcal{P}$ (unknown, accessed via interaction), Pairs state-action samples $\{\{(s^{(i,t)}, a^{(i,t)})\}_{i=1}^{N}\}_{t=0}^{T}$, Function space for the action-value function $\mathcal{F}_{\mathcal{Q}}$, Function space for the last state-value function $\mathcal{F}_{\mathcal{Q}_T}$, Function space for the policy function $\mathcal{F}_{\pi}$, Number of Monte Carlo simulations K

for each sampled state-action $(s^{(i,T)}, a^{(i,T)})$ **do**

Compute the target value

$$\tilde{Q}_t^*(s^{(i,T)}, a^{(i,T)}) = y^{(i,T)} = \frac{1}{K}\sum_{k=1}^{K} R_T^{(k)}(s^{(i,T)}, a^{(i,T)})$$

end for

Fit the value function:

$$\hat{Q}_T^* \leftarrow \underset{f \in \mathcal{F}_{\mathcal{Q}_T}}{\operatorname{argmin}} \frac{1}{N}\sum_{i=1}^{N} \big(f(s^{(i,T)}, a^{(i,T)}) - \tilde{Q}_t^*(s^{(i,T)}, a^{(i,T)})\big)^2$$

for $t = T-1, T-2, \ldots, 0$ **do**

for each sampled state-action $(s^{(i,t)}, a^{(i,t)})$ **do**

Compute the next state

$$s_{t+1}^{(i,t)} = \frac{1}{K}\sum_{k=1}^{K} \mathcal{P}^{(k)}(s^{(i,t)}, a^{(i,t)})$$

if $t = T-1$ **then**

Compute the target value

$$\tilde{Q}_t^*(s^{(i,t)}, a^{(i,t)}) = y^{(i,t)} = \big(\frac{1}{K}\sum_{k=1}^{K} R_t^{(k)}(s^{(i,t)}, a^{(i,t)})\big) + \gamma \hat{Q}_T^*(s_{t+1}^{(i,t)}, a^{(i,t)})$$

else

Compute the optimal action

$$a_{t+1}^{*(i,t)} = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, \hat{Q}_{t+1}^*(s_{t+1}^{(i,t)}, a)$$

Compute the target value

$$\tilde{Q}_t^*(s^{(i,t)}, a^{(i,t)}) = y^{(i,t)} = \big(\frac{1}{K}\sum_{k=1}^{K} R_t^{(k)}(s^{(i,t)}, a^{(i,t)})\big) + \gamma \hat{Q}_T^*(s_{t+1}^{(i,t)}, a_{t+1}^{*(i,t)})$$

end if

end for

Fit the value function:

$$\hat{Q}_t^* \leftarrow \underset{f \in \mathcal{F}_{\mathcal{Q}}}{\operatorname{argmin}} \frac{1}{N}\sum_{i=1}^{N} \big(f(s^{(i,t)}, a^{(i,t)}) - \tilde{Q}_t^*(s^{(i,t)}, a^{(i,t)})\big)^2$$

Fit the policy:

$$\hat{\pi}_t^* \leftarrow \underset{f \in \mathcal{F}_{\pi}}{\operatorname{argmin}} \frac{1}{N}\sum_{i=1}^{N} \big(f(s^{(i,t)}) - \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, \hat{Q}_t^*(s^{(i,t)}, a)\big)^2$$

end for

Output: Sequence of fitted action-value functions $\{\hat{Q}_t^*\}_{t=0}^{T}$, Sequence of fitted policy $\{\hat{\pi}_t^*\}_{t=0}^{T-1}$,

The optimal approximate policy at stage t:

$$\tilde{\pi}_t^*(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, \hat{Q}_t^*(s, a)$$

3.3 Sampling tecniques

In reinforcement learning settings, the independent and identically distributed (i.i.d.) assumption is generally violated due to the sequential nature of data collection. As a consequence, the choice of a sampling strategy becomes crucial when applying algorithms such as Fitted Value Iteration (FVI, Section 3.1) and Fitted Q-Iteration (FQI, Section 3.2), which rely on function approximation techniques commonly based on machine learning or deep learning methods.

Before discussing sampling techniques, it is important to define what we mean by a trajectory. A trajectory is the sequence of states visited and actions taken in a single rollout of the reinforcement learning problem, formally expressed as

$$(s_0, a_0, s_1, a_1, \ldots, s_T). \tag{3.12}$$

Although the Bellman equation and algorithms such as Fitted Value Iteration (FVI) and Fitted Q-Iteration (FQI) typically focus on a single step of this trajectory, considerations of sampling require attention to the trajectory as a whole. This is because, when constructing a training dataset, one must respect the nature of the system dynamics. For instance, even if the initial state is chosen randomly, it is not guaranteed that all subsequent states are reachable or that all actions can be applied at the next step. To properly reflect the underlying transition probabilities, it is often preferable to generate trajectories by simulating the environment, rather than selecting states and actions independently at random at each step.

Furthermore, when these trajectories are used to learn the optimal policy or the optimal state-value function, it is desirable that they follow near-optimal paths, as such samples provide the most informative data for the function approximator and better reflect the true distribution of states and actions under optimal behavior. At the same time, it is important to include trajectories that deviate from optimal behavior to avoid overfitting and to maintain a degree of exploration in less frequently visited regions of the state-action space.

These issues are well documented in the reinforcement learning literature [SB18; MS08; Mun05; Yan+20; San+24].

For these reasons, the following subsection discusses several sampling techniques that we implement and empirically evaluate.

3.3.1 Random Sampling

Random sampling is the simplest sampling technique, and its core idea is to draw samples randomly from a given distribution until the required number of samples is obtained. This technique is typically employed when we have almost complete knowledge of the environment, allowing us to reconstruct a distribution that closely approximates the desired one.

Random sampling is also used in scenarios where little or no prior knowledge of the environment is available and interaction with it is very limited or impossible. In such

cases, it is not feasible to infer a better sampling strategy, and one must rely on purely random trajectories, or even uniformly sample states and actions at each time step t.

Algorithm 3 Random Sampling

1: **Input:** Number of samples N, Sampling distribution μ(e.g., uniform)
2: **for** $i = 1, 2, \ldots, N$ **do**
3: Sample

$$x^{(i)} \sim \mu$$

4: **end for**
5: **Output:** Samples $\{x^{(i)}\}_{i=1}^{N}$

3.3.2 Heuristic Action Sampling

Heuristic action sampling is often preferable, especially when some knowledge of the system dynamics is available or when a human expert with domain expertise can be consulted. In such cases, it is possible to construct a heuristic function that, given a state, returns an action close to a desired trajectory. In the case of optimal policy problem, a heuristic can be designed to approximate the function that maps each state to a sub-optimal action.

Algorithm 4 Heuristic Action Sampling

1: **Input:** Current state $s \in \mathcal{S}$, Heuristic function H
2: Choose the action

$$a = H(s)$$

3: **Output:** action $a \in \mathcal{A}$

Given this algorithm and an initial state s, typically chosen at random, a rollout trajectory can be generated by iteratively applying the heuristic policy. This procedure can be repeated to obtain the desired number of samples.

To introduce stochasticity or to enable exploration of trajectories that deviate from the heuristic ones, two alternative strategies can be employed. The first consists in adding noise, commonly Gaussian noise, to the actions returned by the heuristic or to the states observed during interaction with the environment. The second strategy is to select a random action instead of the heuristic one, with a fixed probability; for example, with probability 0.2, a random action is chosen instead of the heuristic action. These mechanisms help prevent overfitting to heuristic-generated samples, which may be suboptimal and could otherwise bias the learned function approximators.

3.3.3 Greedy Multistep Lookahead Sampling

A more sophisticated and powerful method for optimal-trajectory sampling is greedy multistep lookahead sampling.

This technique consists of selecting a sequence of actions that maximizes the cumulative reward over a finite number of steps L, augmented with a terminal approximated state-value function $\tilde{V}$ that estimates the optimal rewards of all future steps beyond the lookahead horizon.

Given this sequence, we can extract the first optimal action and use it to rollout a sub-optimal trajectory.

The approximated state-value function $\tilde{V}$ can be a heuristic, an approximation learned from an algorithm such as Fitted Value Iteration (FVI) 3.1, or even a simple constant map that returns 0 for every state s.

Algorithm 5 Greedy Multistep Lookahead Sampling

1: **Input:** Current time step t_c, Current state $s_{t_c} \in \mathcal{S}$, Environment rewards $\{R_t\}_{t=0}^{T}$ (unknown, accessed via interaction), Environment transition dynamics $\mathcal{P}$ (unknown, accessed via interaction), Terminal value-state function $\tilde{V}$, Number of steps L, Horizon T, Discount factor γ,

2: Compute the maximum number of steps

$$L_{max} = min\{L, T - t_c + 1\}$$

3: Compute the optimal actions

$$(a_{t_c}, \dots, a_{t_c+L_{\max}-1}) = \underset{(a_{t_c},\dots,a_{t_c+L_{\max}-1})\in\mathcal{A}^{L_{\max}}}{\operatorname{argmax}} \left(\gamma^{L_{\max}} \tilde{V}(s_{t_c+L_{\max}}) + \sum_{t=t_c}^{t_c+L_{\max}-1} \gamma^{t-t_c} \frac{1}{K} \sum_{k=1}^{K} R_t^{(k)}(s_t, a_t^{(j)}) \right)$$

where $s_{t+1} = \frac{1}{K} \sum_{k=1}^{K} \mathcal{P}_t^{(k)}(s_t, a_t^{(j)})$

4: Take the first action

$$a^\star = a_{t_c}$$

5: **Output:** sub-optimal action $a^* \in \mathcal{A}$

When the number of steps L is sufficiently large, or the terminal state-value function $\tilde{V}$ is sufficiently close to the optimal one, this algorithm produces trajectories that are very close to the optimal trajectory.

To encourage exploration and avoid overfitting, as in heuristic action sampling 3.3.2, noise can be added to the chosen actions or to the states. Alternatively, a random action can be selected with a fixed probability, rather than always using the action recommended by the algorithm.

The main difficulty of this algorithm lies in the maximization over the action space, since the reward distribution and the transition dynamics are typically unknown. As a consequence, Monte Carlo simulations are required to approximate the expected return. Moreover, the maximization problem cannot usually be solved in closed form and therefore requires the use of a nonlinear optimization algorithm, such as Powell's method [Pow64].

Methods based on direct nonlinear optimization tend to become computationally expen-

sive as the action space grows. For this reason, we consider an alternative approach that is widely used in reinforcement learning for computing approximate optimal policies or sub-optimal sequences of actions, namely the Cross-Entropy Method (CEM) [Boe+05].

The idea behind the Cross-Entropy Method (CEM) is to iteratively adapt a probability distribution in order to sample actions that are increasingly close to the optimal ones. To illustrate this approach, we present a CEM algorithm based on a mixture of Gaussians (MoG), which iteratively updates the parameters of the distribution, namely, the mean and the variance, so as to concentrate probability mass around near-optimal actions.

In this version of the algorithm, we assume that the action space $\mathcal{A}$ is $\mathbb{R}^m$, where m is the dimension of the action.

The core idea of the algorithm is to start from initial means and variances for the mixture of Gaussians (MoG), sample N action sequences, and select only a fraction of elite samples based on their values of the target function. Given this elite set, a clustering method, such as k-means [Mac67], is applied to partition the samples into M clusters, corresponding to the number of Gaussian components. The means and variances of each Gaussian are then updated using the empirical mean and variance of the samples in the corresponding cluster. After several iterations, the algorithm returns the action sequence from the last iteration that maximizes the objective function.

Algorithm 6 Cross-Entropy Method with Mixture of Gaussians (MoG)

Input: Current time step t_c, Current state $s_{t_c} \in \mathcal{S}$, Environment rewards $\{R_t\}_{t=0}^{T}$ (unknown, accessed via interaction), Environment transition dynamics $\mathcal{P}$ (unknown, accessed via interaction), Number of Monte Carlo simulations K, Terminal value-state function $\tilde{V}$, Number of steps L, Horizon T, Discount factor γ, Action dimension m, Number of mixture components M, Initial means $\{\mu_k\}_{k=1}^{M}$, $\mu_k \in \mathbb{R}^{L_{max} \times m}$, Initial variances $\{\sigma_k^2\}_{k=1}^{M}$, $\sigma_k^2 \in \mathbb{R}^{L_{max} \times m}$, Mixture weights $\{\pi_k^0\}_{k=1}^{M}$, Number of samples N, Elite fraction $\rho \in (0, 1)$, Number of iterations I

Compute the maximum number of steps

$$L_{max} = min\{L, T - t_c + 1\}$$

for $i = 1$ to I **do**

Sample N action sequences from the mixture distribution

$$a^{(j)} = (a_{t_c}^{(j)}, \ldots, a_{t_c+L_{\max}-1}^{(j)}) \sim \sum_{k=1}^{M} \pi_k \mathcal{N}(\mu_k, \mathrm{diag}(\sigma_k^2)), \quad j = 1, \ldots, N$$

Evaluate each sequence

$$J^{(j)} = \gamma^{L_{\max}} \tilde{V}(s_{t_c+L_{\max}}) + \sum_{t=t_c}^{t_c+L_{\max}-1} \gamma^{\,t-t_c} \frac{1}{K} \sum_{k=1}^{K} R_t^{(k)}(s_t, a_t^{(j)})$$

where $s_{t+1} = \frac{1}{K} \sum_{k=1}^{K} \mathcal{P}_t^{(k)}(s_t, a_t^{(j)})$

Select the elite set of action sequences

$$\mathcal{E} = \left\{ a^{(j)} : j \in \underset{\substack{\mathcal{S} \subset \{1,\ldots,N\} \\ |\mathcal{S}| = \lfloor \rho N \rfloor}}{\mathrm{argmax}} \sum_{k \in \mathcal{S}} J^{(k)} \right\}$$

Cluster elite sequences into M clusters using a distance-based clustering method (e.g., k-means)

for $k = 1$ to M **do**

Let $\mathcal{E}_k$ be the set of elite samples assigned to cluster k

for $\ell = 0$ to $L_{max} - 1$ **do**

Update mean

$$\mu_{k,\ell} \leftarrow \frac{1}{|\mathcal{E}_k|} \sum_{a^{(j)} \in \mathcal{E}_k} a_{t_c+\ell}^{(j)}$$

Update variance

$$\sigma_{k,\ell}^2 \leftarrow \frac{1}{|\mathcal{E}_k|} \sum_{a^{(j)} \in \mathcal{E}_k} \left(a_{t_c+\ell}^{(j)} - \mu_{k,\ell}\right)^2$$

end for

Update mixture weight

$$\pi_k \leftarrow \frac{|\mathcal{E}_k|}{|\mathcal{E}|}$$

end for

end for

Output: best action sequence at last iteration

$$(a_{t_c}, \ldots, a_{t_c+L_{max}-1})^* = \underset{a^{(j)} \in \{a^{(1)}, \ldots, a^{(N)}\}}{\mathrm{argmax}} J^{(j)}$$

As usual, the expectations are replaced with Monte Carlo simulations when the underlying distributions are unknown.

3.4 Function approximators

In this section, we present the models used to empirically evaluate the function approximations in the Fitted Value Iteration (FVI) and Fitted Q-Iteration (FQI) algorithms (Sections 1 and 2). All models are based on standard machine learning approaches and rely on least-squares optimization with regularization.

3.4.1 Linear model

The first model we present is the simplest one and is referred to as regularized linear regression. The key idea is to approximate the target function using a linear function.

The coefficients of the regression are estimated by fitting the linear function to a finite set of input-output samples, by minimizing the squared difference between the predicted values and the observed targets. This criterion, commonly referred to as the least-squares approach, encourages the model to produce predictions that are close to the available data.

To avoid excessively large coefficients values and improve the generalization properties of the model, a regularization term equal to the squared norm of the coefficients vector is added to the objective function and weighted by a parameter $\lambda > 0$. This form of regularized least-squares regression is commonly known as ridge regression.

Combining these elements, the optimization problem can be written as:

$$\min_{w \in \mathbb{R}^n} \frac{1}{N} \left\| Xw - Y \right\|^2 + \lambda \left\| w \right\|^2 \tag{3.13}$$

where $X \in \mathbb{R}^{N \times n}$ denotes the matrix of input samples and $Y \in \mathbb{R}^N$ denotes the vector of target values. This optimization problem is referred to as empirical risk minimization (ERM).

Once the optimal parameter vector w has been learned, the resulting function approximator is defined as:

$$f_w(x) = x^\top w \tag{3.14}$$

where $x \in \mathbb{R}^n$ denotes a single sample.

3.4.2 Quadratic model

A more complex extension of the linear model is the quadratic model. This model is closely related to the linear model described in Section 3.4.1. However, instead of learning a vector of linear coefficients $w \in \mathbb{R}^n$, we infer a matrix $W \in \mathbb{R}^{n \times n}$, which defines a quadratic function approximator of the form:

$$f_W(x) = x^\top W x \tag{3.15}$$

The matrix W is learned by minimizing the same empirical risk used for the linear model, as defined in Equation (3.13).

To make this formulation explicit, the quadratic function can be rewritten as:

$$x^\top W x = \sum_{i,j} W_{ij} x_i x_j,$$
$$\phi : \mathbb{R}^n \to \mathbb{R}^{n^2}$$
$$\phi(x) = \mathrm{vec}(xx^\top)$$
$$f_w(x) = \phi(x)^\top w$$

where $vec(\cdot)$ denotes the vectorization operator and $w = vec(W) \in \mathbb{R}^{n^2}$.

The mapping $\phi(\cdot)$ is referred to as the feature map, and its codomain $\mathbb{R}^{n^2}$ is called the feature space. By applying the feature map, the original input $x \in \mathbb{R}^n$ is represented in a higher-dimensional space where quadratic interactions between input components are explicitly encoded.

With this representation, the quadratic model is equivalent to a linear model in a transformed feature space. Consequently, the empirical risk minimization problem can be written as

$$\min_{w \in \mathbb{R}^{n^2}} \frac{1}{N} \left\| \Phi w - Y \right\|^2 + \lambda \left\| w \right\|^2 \tag{3.16}$$

where $\Phi \in \mathbb{R}^{N \times n^2}$ is the design matrix whose rows are given by $\phi(x^{(i)})^\top$ for each training sample $x^{(i)}$.

Moreover, the model depends only on the symmetric part of the matrix W.

Indeed, any matrix $W \in \mathbb{R}^{n \times n}$ can be decomposed as:

$$W = \frac{1}{2}(W + W^\top) + \frac{1}{2}(W - W^\top) \tag{3.17}$$

where the first term is symmetric and the second term is skew-symmetric.

Substituting this decomposition into the quadratic form yields:

$$x^\top W x = x^\top \left(\frac{1}{2}(W + W^\top) \right) x + x^\top \left(\frac{1}{2}(W - W^\top) \right) x \tag{3.18}$$

Since $W - W^\top$ is skew-symmetric, it follows that:

$$x^\top (W - W^\top) x = 0 \qquad \forall x \in \mathbb{R}^n \tag{3.19}$$

and therefore

$$x^\top W x = x^\top \left(\frac{1}{2}(W + W^\top) \right) x \tag{3.20}$$

Consequently, the quadratic model is completely determined by the symmetric part of W. Without loss of generality, we may therefore restrict W to be symmetric, which reduces the number of independent parameters from n^2 to $n(n+1)/2$, corresponding to the entries of the upper triangular part of the matrix.

3.4.3 Polynomial kernel model

A more expressive extension of the linear and quadratic models (Sections 3.4.1 and 3.4.2) is the polynomial kernel model.

This model builds upon the feature space formulation introduced in the quadratic case, but avoids the explicit construction of feature vectors $\phi(x^{(i)})$. Instead, it employs kernel methods, which allow the learning algorithm to operate implicitly in a nonlinear feature space by relying solely on inner products between feature-mapped samples [SSM08].

The key idea of the kernel approach is that, for a suitable choice of feature map, the empirical risk minimization problem can be reformulated such that it depends only on inner products between feature-mapped samples:

$$\langle \phi(x^{(i)}), \phi(x^{(j)}) \rangle \tag{3.21}$$

which can be computed directly through a kernel function:

$$k(x^{(i)}, x^{(j)}) = \langle \phi(x^{(i)}), \phi(x^{(j)}) \rangle \tag{3.22}$$

This technique, commonly referred to as the kernel trick.

To understand why this formulation holds, we first consider the empirical risk minimization problem in Equation 3.16. It can be shown that its solution admits two equivalent closed-form expressions:

$$\begin{aligned} w &= (\Phi^\top \Phi + \lambda N I)^{-1} \Phi^\top Y \\ w &= \Phi^\top (\Phi \Phi^\top + \lambda N I)^{-1} Y \end{aligned} \tag{3.23}$$

where I denotes the identity matrix of appropriate dimension.

We introduce a new parameter vector c such that

$$\begin{aligned} w &= \Phi^\top c \\ c &= (k(X, X) + \lambda N I)^{-1} Y \end{aligned} \tag{3.24}$$

where $k(X, X)$ denotes the matrix whose (i, j)-th entry is given by the kernel evaluation $k(x^{(i)}, x^{(j)})$ for all pairs of input samples.

Using this dual representation, the function approximator can be expressed as:

$$f_c(x) = k(x, X)c \tag{3.25}$$

where X denotes the matrix whose rows correspond to the training samples used to estimate c, and x the new input sample for which the approximated value is computed.

In the case of the polynomial kernel, the kernel function is given by:

$$k(x^{(i)}, x^{(j)}) = \left(x^{(i)\top} x^{(j)} + 1 \right)^d \tag{3.26}$$

where $d \in \mathbb{N}$ denotes the degree of the polynomial.

This kernel implicitly corresponds to a feature space that contains all polynomial interactions terms between input variables up to degree d, thereby generalizing quadratic and, more generally, polynomial regression models of degree d.

3.4.4 Gaussian kernel model

The last and most complex model considered is the Gaussian kernel model.

This model follows the same approach as the polynomial kernel model (Section 3.4.3) and differs only in the choice of the kernel function, which is defined as:

$$k(x^{(i)}, x^{(j)}) = e^{-\frac{\left\|x^{(i)} - x^{(j)}\right\|^2}{2\sigma^2}} \tag{3.27}$$

where $\sigma > 0$ is a kernel parameter that controls the width of the Gaussian function.

This kernel model implicitly corresponds to an infinite-dimensional feature space in which similarity between samples is measured through a Gaussian function with variance σ^2.

As before, the function approximator can be expressed as:

$$f_c(x) = k(x, X)c \tag{3.28}$$

where c is calculated as:

$$c = (k(X, X) + \lambda N I)^{-1} Y \tag{3.29}$$

3.5 Training pipeline

To conclude the methodology section and to enable a clear understanding of the experiments presented in the results chapter (Chapter 5), we describe the procedure used to train our function approximators (Section 3.4).

Our function approximators are machine learning models characterized by two distinct types of parameters The first type consists of the model parameters, denoted by w or c, which are real-valued parameters learned by solving a risk minimization problem (Equation (3.16)). The second type comprises the hyperparameters, which must be specified before the optimization process. These hyperparameters determine the structure and behavior of the model, for example the parameter σ in the Gaussian kernel model (Section 3.4.4) or the parameter λ, which represents the regularization parameter.

An important distinction that must be introduced concerns the samples used to train the models. We first introduce the sampling techniques (Section 3.3), which we employ to generate a set of N samples for each time step t. For each sample set, we uniformly split the data into a training set (50%) and a validation set (50%). We assume that each sample set provides adequate coverage of the underlying space; therefore, a uniform split does not introduce bias into the training process. The validation set is used to determine the optimal hyperparameters of the model through a greedy search procedure. In this greedy search, given a predefined list of hyperparameters, we select the configuration that yields the lowest error on the validation set.

Below we present the pseudocode of the procedure used in the training pipeline.

Algorithm 7 Training pipeline

Input: Model M, Input samples X, Output samples Y, Number of samples N, hyperparameter set $\mathcal{H}$
Construct the dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$
Uniformly split $\mathcal{D}$ into training set $\mathcal{D}_{\text{train}}$ (70%) and validation set $\mathcal{D}_{\text{val}}$ (30%)
Initialize minimum validation error $E_{\min} \leftarrow +\infty$
Initialize optimal hyperparameters $h^* \leftarrow \emptyset$
for each hyperparameter configuration $h \in \mathcal{H}$ **do**
    Train the model M by solving the ERM problem on $\mathcal{D}_{\text{train}}$ using hyperparameters h
    Compute the Mean Absolute Error on the validation set:

$$E_h = \frac{1}{|\mathcal{D}_{\text{val}}|} \sum_{(x_i, y_i) \in \mathcal{D}_{\text{val}}} |y_i - M(x_i)|$$

    if $E_h < E_{\min}$ **then**
        $E_{\min} \leftarrow E_h$
        $h^* \leftarrow h$
    end if
end for
Retrain the model M on $\mathcal{D}_{\text{train}}$ using optimal hyperparameters h^*
Output: Retrained model M, optimal hyperparameters h^*

Once the optimal hyperparameters have been identified through grid search, the model is retrained on the combined training and validation datasets. Finally, its performance is evaluated on a separate test set. This test set is kept completely independent from both training and model selection and is used only once, in order to obtain an unbiased assessment of the model's ability to generalize to unseen data.

Chapter 4

Problem setup

To assess the performance of our approximate methods, we consider the Linear-Quadratic Regulator (LQR) problem [AM89], a classical benchmark in optimal control and reinforcement learning.

4.1 Linear Quadratic Regulator

The LQR setting provides a controlled dynamical system with linear state evolution and quadratic reward, for which the optimal policy and value functions are known in closed form. This makes it an ideal framework for evaluating approximate reinforcement learning methods, as it allows us to compare the results produced by our algorithm with the exact analytical solution. In this section both the reward and the evolution are deterministic to simplifies the equations, but in reality both can be affected by noise.

First, we define the set of all the possible states $\mathcal{S}$ and actions $\mathcal{A}$ as:

$$\mathcal{S} = \mathbb{R}^n$$
$$\mathcal{A} = \mathbb{R}^m$$

The transition dynamics is definited as:

$$s_{t+1} = \mathcal{P}(s_t, a_t) = As_t + Ba_t \tag{4.1}$$

where A and B define the linear state-update dynamics of the system.

The rewards are definited by the formula:

$$R_t(s, a) = -(s^\top S_t s + a^\top C_t a) \tag{4.2}$$

and the terminal reward depends only on the state s:

$$R_T(s) = -(s^\top S_T s) = R_T(s, a) \tag{4.3}$$

Here $\{S_t\}_{t=0}^{T}$ and $\{C_t\}_{t=0}^{T-1}$ define the quadratic rewards of the system.

The optimal state-value function can be written as:

$$V_t^*(s) = \max_{a \in \mathbb{R}^m} \{R_t(s,a) + V_{t+1}^*(As + Ba)\} \tag{4.4}$$

and the optimal action-value function as:

$$Q_t^*(s,a) = R_t(s,a) + \max_{a^* \in \mathbb{R}^m} \{Q_{t+1}^*(As + Ba, a^*)\} \tag{4.5}$$

We make the following assumptions:

- For all t, $S_t \in \mathbb{R}^{n \times n}$ is symmetric and positive semidefinite ($S_t \succeq 0$).
- For all t, $C_t \in \mathbb{R}^{m \times m}$ is symmetric and positive definite ($C_t \succ 0$).
- Matrices A, B are given.
- The sets of matrices $\{S_t\}_{t=0}^{T}$ and $\{C_t\}_{t=0}^{T-1}$ are given.

To obtain a closed-form solution, we consider quadratic state-value functions of the form:

$$V_t^*(s) = -s^\top P_t s, \qquad \text{s.t. } P_t = P_t^\top, P_t \succeq 0 \tag{4.6}$$

4.1.1 Riccati recursion

Based on the quadratic form of the state-value function, the optimal solution can be computed recursively using the Riccati recursion to find all the P_t.

Substituting the quadratic form (4.6) into the optimal state-value function (4.4), we obtain:

$$V_t^*(s) = \max_{a \in \mathbb{R}^m} \Big[-(s^\top S_t s + a^\top C_t a) - (As + Ba)^\top P_{t+1}(As + Ba) \Big] \tag{4.7}$$

Expanding the quadratic term yields:

$$(As + Ba)^\top P_{t+1}(As + Ba) = s^\top A^\top P_{t+1} A s + 2 s^\top A^\top P_{t+1} B a + a^\top B^\top P_{t+1} B a \tag{4.8}$$

Thus, the maximization reduces to a quadratic function of the action:

$$\Phi(a) = -s^\top (S_t + A^\top P_{t+1} A) s - 2 s^\top A^\top P_{t+1} B a - a^\top (C_t + B^\top P_{t+1} B) a \tag{4.9}$$

The gradient of $\Phi(a)$ with respect to a is given by:

$$\nabla_a \Phi(a) = -2 s^\top A^\top P_{t+1} B - 2 a^\top (C_t + B^\top P_{t+1} B) \tag{4.10}$$

Setting the gradient equal to zero yields:

$$s^\top A^\top P_{t+1} B + a^\top (C_t + B^\top P_{t+1} B) = 0$$

Solving for the optimal action, we obtain:

$$a^* = -(C_t + B^\top P_{t+1} B)^{-1} B^\top P_{t+1} A s \tag{4.11}$$

Hence, the optimal feedback control law, which maps the current state to the optimal control action, is linear in the state and is given by:

$$K_t = -(C_t + B^\top P_{t+1} B)^{-1} B^\top P_{t+1} A \tag{4.12}$$

And the optimal policy is defined as:

$$\pi_t^*(s) = K_t s \tag{4.13}$$

Substituting the optimal action (Equation (4.11)) into the optimal-state value function (Equation (4.7)) and using expanded quadratic form (Equation (4.9)), we obtain:

$$\begin{aligned}
V_t^*(s) &= -s^\top (S_t + A^\top P_{t+1} A)s - 2s^\top A^\top P_{t+1} B K_t s - s^\top K_t^\top (C_t + B^\top P_{t+1} B) K_t s \\
V_t^*(s) &= -s^\top \Big[(S_t + A^\top P_{t+1} A) - 2A^\top P_{t+1} B (C_t + B^\top P_{t+1} B)^{-1} B^\top P_{t+1} A \\
&\qquad + A^\top P_{t+1} B (C_t + B^\top P_{t+1} B)^{-1} B^\top P_{t+1} A \Big] s \\
V_t^*(s) &= -s^\top \Big[(S_t + A^\top P_{t+1} A) - A^\top P_{t+1} B (C_t + B^\top P_{t+1} B)^{-1} B^\top P_{t+1} A \Big] s
\end{aligned}$$

Comparing with the quadratic form (Equation (4.6)), we obtain the Riccati recursion:

$$P_t = (S_t + A^\top P_{t+1} A) - A^\top P_{t+1} B (C_t + B^\top P_{t+1} B)^{-1} B^\top P_{t+1} A, \quad \text{s.t. } P_T = S_T \tag{4.14}$$

Starting from the terminal condition at time T, the matrices P_t and K_t can be computed recursively backward in time. This allows the reconstruction of the optimal state-value function and the optimal policy at each time step t for any given state s. Clearly, K_T and π_T^* are not defined, and there is no need to compute them, since no action is taken at the terminal time step T.

Using this notation, the optimal action-value function can be expressed as:

$$Q_t^*(s, a) = -(s^\top S_t s + a^\top C_t a) - (As + Ba)^\top P_{t+1} (As + Ba) \tag{4.15}$$

Chapter 5

Results

In this chapter, we describe all the experiments and results related to the methodology presented in Chapter 3, in order to gain a better understanding of the advantages and limitations of the proposed approximate solutions.

5.1 Environment Description

We evaluate our algorithms in an LQR environment (Chapter 4). The environment consists of a two-dimensional car that must reach the target position $(0, 0)$. The system dynamics are governed by basic kinematic equations, and the reward function is defined as the negative of a cost expressed in quadratic form. This cost accounts for the vehicle's velocity, its distance from $(0, 0)$, and its acceleration. The objective is to reach $(0, 0)$ while minimizing travel time and control effort, and ensuring that the vehicle is stationary upon arrival.

The details of the environment are as follows:

- **State vector $s \in \mathbb{R}^4$:**
$$s = \begin{bmatrix} p_x \\ p_y \\ v_x \\ v_y \end{bmatrix}$$
where (p_x, p_y) are the car's position coordinates and (v_x, v_y) are its velocity components.
- **Action vector $a \in \mathbb{R}^2$:**
$$a = \begin{bmatrix} a_x \\ a_y \end{bmatrix}$$
representing the accelerations applied in the x and y directions.
- **Dynamics matrices:**
$$A = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

matrix A encodes position and velocity update, while B maps control accelerations into the state update.
With these matricies we have that each components evolve with this formulas:

$$\begin{aligned} p_x(t+1) &= p_x(t) + v_x(t) + 0.5a_x(t) \\ p_y(t+1) &= p_y(t) + v_y(t) + 0.5a_y(k) \\ v_x(t+1) &= v_x(t) + a_x(t) \\ v_y(t+1) &= v_y(t) + a_y(t) \end{aligned}$$

- **State penalty matrix $S_t \in \mathbb{R}^{4\times4}$:**

$$S_t = \text{diag}(4, 4, 4, 4)$$

the values are chosen according to Bryson's rule [BH75], which sets each diagonal entry as the inverse square of the maximum acceptable deviation from 0:

$$[S_t]_{ii} = \frac{1}{(s_i^{\max})^2} \tag{5.1}$$

where $s_i^{\max}$ is taken to be 0.5 for all the state dimensions.

- **Action penalty matrix $C_t \in \mathbb{R}^{2\times2}$:**

$$C_t = \text{diag}(0.25, 0.25)$$

similarly, the control weights are chosen following Bryson's rule, which corresponds to keeping each action input within ±2 units.

With this decision we can see that our reward is calculated as:

$$R_t(s, a) = -(4p_x^2 + 4p_y^2 + 4v_x^2 + 4v_y^2 + 0.25a_x^2 + 0.25a_y^2) \tag{5.2}$$

- **Dimensions:**

$$n = 4 \quad \text{(state dimension)}, \quad m = 2 \quad \text{(action dimension)}.$$

- **Horizon length:**

$$T = 5$$

- **Discount factor:**

$$\gamma = 1$$

- **Number of Monte Carlo simulations:**

$$K = 100$$

To simulate uncertainty in the car parameters and potential noise in the reward returned by the environment, we perturb both the reward and the system dynamics with Gaussian noise:

$$r_{t+1} = R_t(s_t, a_t) + \eta, \quad \eta \sim \mathcal{N}(0, \sigma^2) \tag{5.3}$$

$$s_{t+1} = \mathcal{P}(s_t, a_t) + \eta, \quad \eta \sim \mathcal{N}(0, \sigma^2) \tag{5.4}$$

5.2 Datasets

To evaluate our function approximators and algorithms, we first construct three different datasets, each corresponding to a distinct sampling technique (Section 3.3). In this section, we focus exclusively on state-action sampling, which defines the inputs to our models, since the target values are computed according to the FVI and FQI algorithms (Algorithms 1 and 2). We, then, empirically investigate whether sampling from a distribution that follows the optimal policy provides any advantage compared to alternative sampling strategies.

Each dataset contains $N = 1000$ samples. To better understand their composition, we visualize the sampled positions (p_x, p_y), using different colors to represent different time steps t. Analogous plots are provided for the velocity components (v_x, v_y).

The dataset varies slightly when using the FQI algorithm (Section 3.2), since in this case each sample must include an action as input. However, for each sampling technique, actions can be obtained in a straightforward manner. In random sampling, actions can be sampled uniformly. For both the Greedy Multistep Lookahead and Heuristic sampling methods, states are determined along trajectories, so we can select the action corresponding to the same time step t within a trajectory; with the exception of the last step T, where no action is taken so and the action is set to $(0, 0)$. Following this approach, we can also plot the action components (a_x, a_y) of the input samples, which are used exclusively by the FQI algorithm.

For sampling methods that aim to approximate the optimal state distribution, we expect the samples to increasingly concentrate around $(0, 0)$ as t increases, since the agent's objective is to terminate in the state

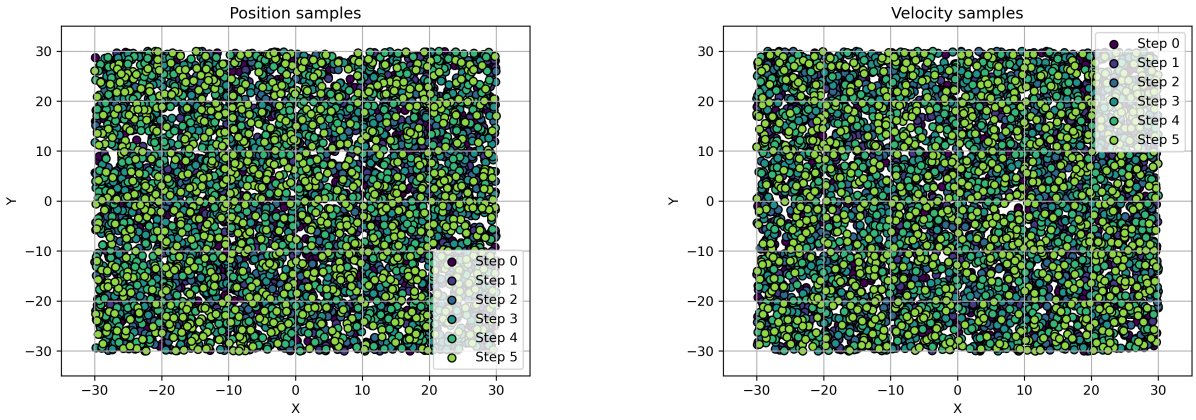
$$s_T^* = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

For the final evaluation, the test set consists of one hundred uniformly sampled random samples.

5.2.1 Random Sampling

The random uniform sampling method is restricted to a subset of the real space, where each component no longer belongs to $\mathbb{R}$ but instead lies in the interval $[-30, 30]$, for computational reason.

The plots of the position and velocity components obtained using this sampling technique (Section 3.3.1) are shown in Figure 5.1, while the plot of the acceleration components is shown in Figure 5.2.



(a) Position components

(b) Velocity components

Figure 5.1: Position and velocity components obtained using random sampling

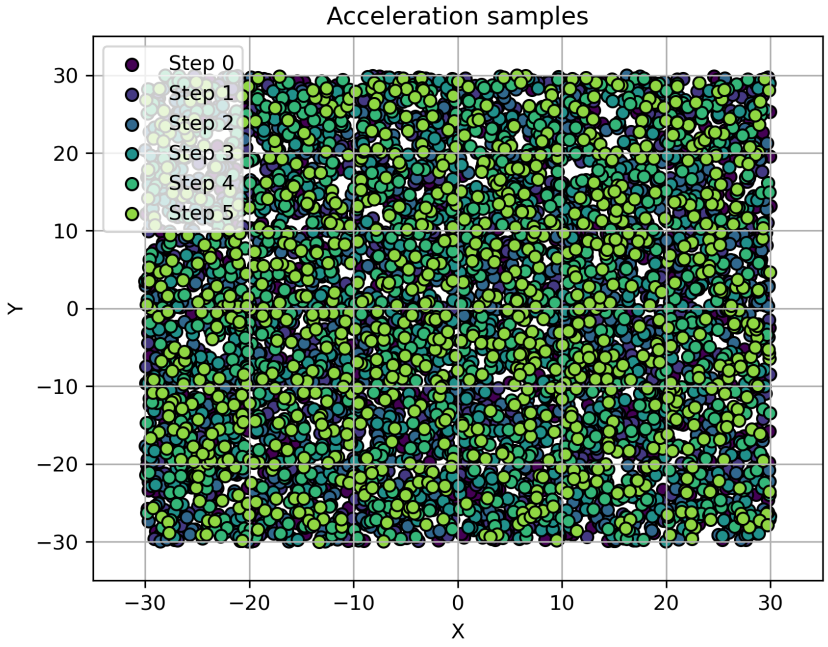


Figure 5.2: Acceleration components obtained using random sampling

5.2.2 Heuristic Action Sampling

Following the alghorithm described in Section 3.3.2, we employ a simple heuristic policy, defined as

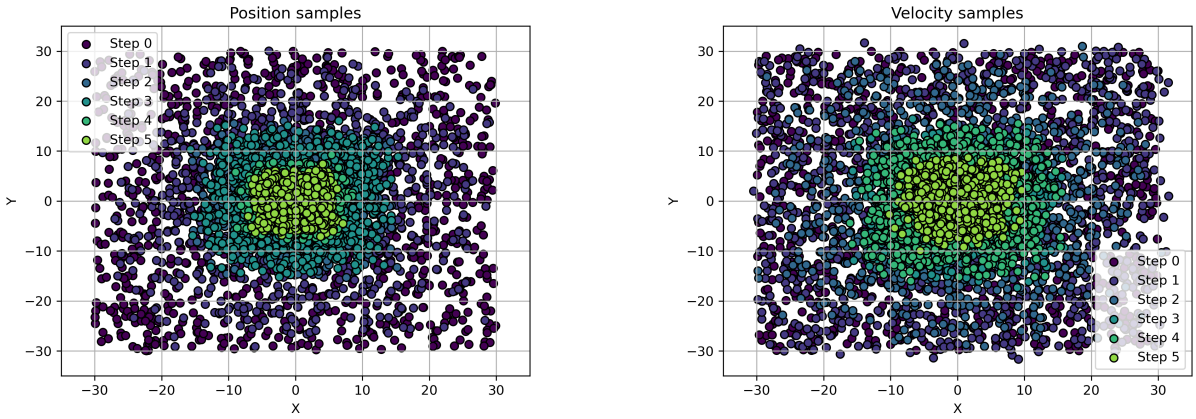
$$H(s) = \begin{bmatrix} -p_x - v_x \\ -p_y - v_y \end{bmatrix},$$

which applies an action in the direction opposite to the sum of position and velocity components, encouraging convergence toward the origin in both dimensions.

To preserve the dynamics of the trajectories while enabling exploration and reducing overfitting, we add Gaussian noise with mean $\mu = 0$ and variance $\sigma^2 = 1$ to each com-

ponent of the sampled actions. The rollout trajectories are initialized from a random uniform state sample.

The plots of the components at each timestemp t are shown in Figures 5.3 and 5.4.



(a) Position components

(b) Velocity components

Figure 5.3: Position and velocity components obtained using Heuristic sampling

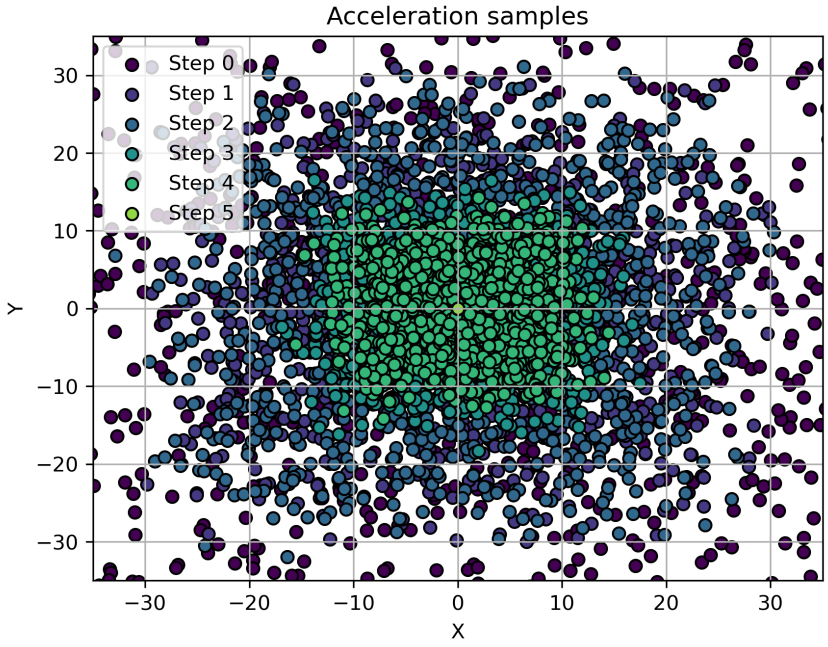


Figure 5.4: Acceleration components obtained using Heuristic sampling

We can already observe that, with a simple heuristic function, the samples converge to the origin.

5.2.3 Greedy Multistep Lookahead Sampling

Here, we follow the algorithm described in Section 3.3.3, using the Cross-Entropy Method (Algorithm 6) to solve the resulting maximization problem.

Specifically, we set the lookahead horizon to $L = 3$, the number of mixture components to $M = 3$, and the number of CEM iterations to $I = 20$. At each iteration, $N = 200$ samples are drawn, with an elite fraction of $\rho = 0.5$.

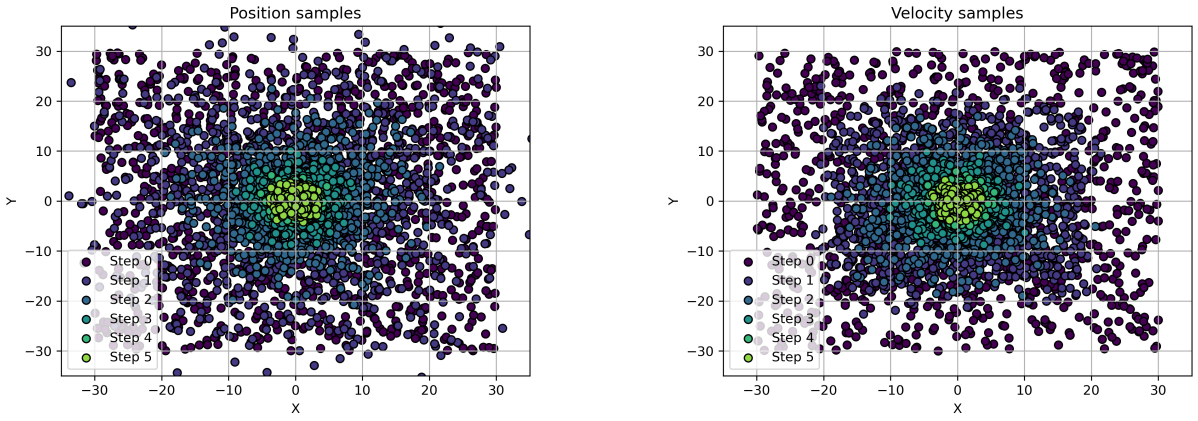
The initial means of all mixture components are set to $\mu = 0$, the variance is set to $\sigma^2 = 1$, and the mixture weights are uniformly initialized to $\frac{1}{3}$.

The terminal value function is fixed as $\tilde{V}(s) = 0, \quad \forall s \in \mathcal{S}$. In this way, the algorithm simply tries to minimize the L step, without taking into account the dynamics of the problem beyond that.

To preserve the dynamics of the trajectories while enabling exploration and reducing overfitting, we add Gaussian noise with mean $\mu = 0$ and variance $\sigma^2 = 1$ to each component of the sampled actions. The rollout trajectories are initialized from a random uniform state sample.

The Gaussian noise in the environment reward function 5.3 is defined with variance $\sigma^2 = 1$, while the noise in the environment evolution dynamics 5.4 has variance $\sigma^2 = 0.1$.

The plots of the components at each timestemp t are shown in Figure 5.5 and 5.6.



(a) Position components

(b) Velocity components

Figure 5.5: Position and velocity components obtained using Greedy Multistep Lookahead sampling

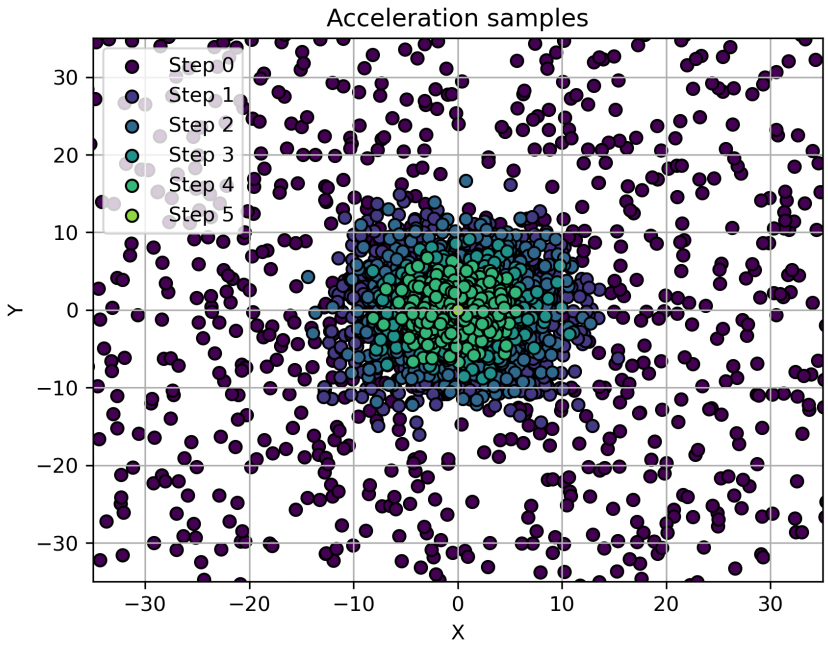


Figure 5.6: Acceleration components obtained using Greedy Multistep Lookahead sampling

Here, we see more clearly that, with greedy minimization, the samples converge faster to the origin compared to the heuristic function.

5.3 Experimental Results

In this section, we present the empirical results obtained by applying the proposed approximate techniques to the considered problem.

First, we recall the definitions introduced in the Background and Methodology chapters (Chapters 2 and 3), which are necessary to properly interpret the metrics presented in this section. In particular, we recall the definition of the state-value function V; analogous definitions apply to the action-value function Q.

We denote by V_t^* the true optimal state-value function, computed using the Riccati equation (Equation 4.6). Similarly, π_t^* denotes the optimal policy obtained from the Riccati equation (Equation 4.13).

The notation $\tilde{V}_t^*$ refers to the approximate optimal state-value function used as the target in the function approximation step, as defined in Equation 3.3. This corresponds to the target variable y in the associated machine learning problem. In contrast, $\hat{V}_t^*$ denotes the learned approximation of the optimal state-value function (Equation 3.4).

Similarly, $\tilde{\pi}_t^*$ denotes the approximate optimal policy obtained by combining the Bellman equation with the learned state-value function (Equation 3.5), whereas $\hat{\pi}_t^*$ represents the policy directly learned using a machine learning method (Equation 3.6).

We recall that the true optimal functions V_t^* and π_t^* are generally unknown and can be computed in this work only because the experiments are conducted in an LQR setting.

The quantities $\tilde{V}_t^*$ and $\tilde{\pi}_t^*$ are obtained through the Fitted Value Iteration (FVI) algorithm (Section 3.1), while $\hat{V}_t^*$ and $\hat{\pi}_t^*$ correspond to their approximations learned via machine learning techniques.

Finally, we define the rollout state-value function $V_t^{\tilde{\pi}^*}$, which represents the evaluation of the policy $\tilde{\pi}^*$ starting from time t. It corresponds to the expected cumulative discounted reward obtained by following the policy $\tilde{\pi}^*$ from time t and initial state s. Formally:

$$V_t^{\tilde{\pi}^*}(s) = \mathbb{E}_{\tilde{\pi}^*,R,\mathcal{P}}\left[\sum_{k=t}^{T} \gamma^{k-t} r_{k+1} \,\middle|\, s_t = s\right]. \tag{5.5}$$

In practice, this expectation is approximated using Monte Carlo simulations.

The rollout state-value function is a fundamental quantity for the evaluation of approximate solutions. In general settings, where the true optimal state-value function is unknown, it is not possible to directly assess how far a solution is from the optimal one, nor to quantify the bias of the corresponding policy.

What can be computed instead is the approximation error between the target state-value function $\tilde{V}_t^*$ and the learned state-value function $\hat{V}_t^*$ at each time step t. However, this error alone may be misleading. Due to error propagation, it is possible to observe a small discrepancy between $\tilde{V}_t^*$ and $\hat{V}_t^*$ even when the actions selected by the resulting policy are far from optimal.

The rollout state-value function provides a more informative evaluation metric. It captures the expected return obtained by starting from a state s and following the policy $\tilde{\pi}^*$. Therefore, if the rollout value significantly differs from the value predicted by the learned function $\hat{V}_t^*$, it indicates that the function approximation does not generalize well to the underlying reinforcement learning problem, even when the training or the test error of the machine learning model is low.

All the errors between the considered metrics are computed using the Mean Absolute Percentage Error (MAPE). This choice is motivated by the fact that we are not interested in analyzing the absolute magnitude of the error, but rather the relative distance between the approximated quantities and the corresponding optimal ones.

The MAPE is defined as

$$\text{MAPE}(\tilde{f}, f) = \frac{1}{N}\sum_{i=1}^{N}\left|\frac{\tilde{f}_i - f_i}{f_i}\right| \times 100 \tag{5.6}$$

where $\tilde{f}_i$ and f_i correspond to the i-th evaluation of the approximated and true function, respectively, over a set of N samples.

In the next sections, we present the results separately for each model used to approximate the state-value function $\hat{V}_t^*$.

For the policy model $\hat{\pi}_t^*$, we always employ a linear model with a regularization parameter set to $\lambda = 10^{-2}$, since the true optimal policy is known to be linear. This choice is motivated by the fact that the primary objective of this work is the analysis of the approximate algorithms FVI and FQI (Algorithms 1 and 2). The actor-critic setting, in which the policy is also learned via machine learning techniques, is therefore of secondary interest.

By fixing a linear structure for the policy model, we are able to keep the focus on the quality of the state-value and action-value function approximations. Moreover, since the optimal policy is linear, learning this function does not pose significant generalization challenges. In this context, the key aspect is obtaining an accurate approximate optimal policy $\tilde{\pi}_t^*$; if this is achieved, the corresponding learned policy $\hat{\pi}_t^*$ is expected to perform well.

In all the experiments the Gaussian noise in the environment reward function 5.3 is defined with variance $\sigma^2 = 1$, while the noise in the environment evolution dynamics 5.4 has variance $\sigma^2 = 0.1$.

All the results reported in the tables are rounded to two decimal places.

5.3.1 Linear model

The choice of using a linear model (Section 3.4.1) is motivated by the fact that, although the problem is known to be quadratic, we intentionally underfit the state-value function in order to analyze the performance degradation in the underlying reinforcement learning problem.

Since this modeling choice is intentionally far from the optimal one, we present only the results obtained using the uniform dataset (Section 5.2.1) and by applying the Fitted Value Iteration (FVI) algorithm (Algorithm 1).

The hyperparameter λ is tuned by evaluating 50 logarithmically spaced values in the range $\lambda \in [10^{-10}, 10^{1}]$ on the validation set.

In Table 5.1 is reported the training error of the model with the best hyperparameter.

t	$(\hat{V}_t^*,\ \tilde{V}_t^*)$	$(\hat{V}_t^*,\ V_t^*)$	$(\hat{V}_t^*,\ V_t^{\tilde{\pi}^*})$	$(\hat{\pi}_t^*,\ \pi_t^*)$	$(\tilde{\pi}_t^*,\ \pi_t^*)$
0	178.86	103.04	100	248.1	3608.75
1	271.83	99.98	99.99	177.32	2355.38
2	181.97	99.67	99.92	129.06	1263.3
3	112.57	99.6	99.42	108.04	465.61
4	100.91	99.5	99.43	102.23	235.07
5	99.79	99.79	99.79		

Table 5.1: Training MAPE Linear model and random sampling FVI

And the test error is reported in Table 5.2.

t	$(\hat{V}_t^*, \tilde{V}_t^*)$	$(\hat{V}_t^*, V_t^*)$	$(\hat{V}_t^*, V_t^{\tilde{\pi}^*})$	$(\hat{\pi}_t^*, \pi_t^*)$	$(\tilde{\pi}_t^*, \pi_t^*)$
0	169.31	99.88	100	182.87	1629.56
1	140.15	99.35	99.99	134.24	1193.21
2	168.33	99.13	99.92	114.89	836.44
3	100.31	98.47	99.44	108.94	418.43
4	98.18	98.34	98.73	98.7	331.98
5	99.03	99.03	99.03		

Table 5.2: Test MAPE Linear model and random sampling FVI

We observe that the training and test errors are very similar and both relatively high. This indicates that the model is underfitting the approximation of the optimal value function.

To better understand this behavior, we plot the optimal state-value function V_t^* and its learned approximation $\hat{V}_t^*$ at time $t = 0$, fixing the velocities to $v_x = 0$ and $v_y = 0$ and varying the position over the range $[-30, 30]$, as shown in Figure 5.7.

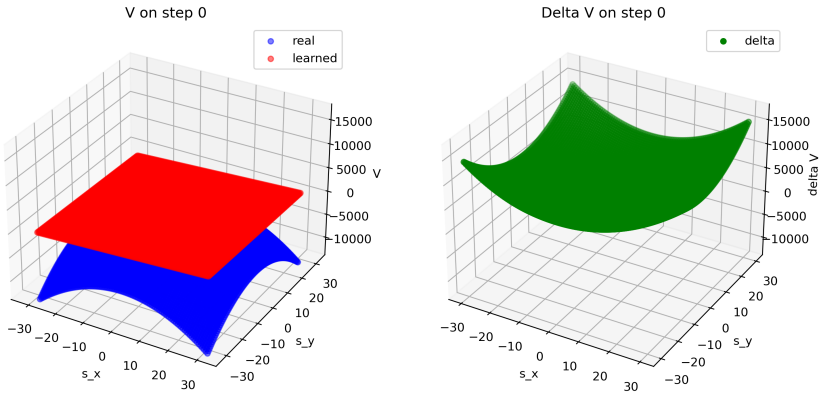


Figure 5.7: Comparison between the optimal and the approximated state-value functions at time $t = 0$.

It is evident that a linear function is unable to accurately approximate a quadratic value function, which explains the observed underfitting.

Another interesting observation that can be drawn from the error values is that the learned policy $\hat{\pi}_t^*$ exhibits a significantly smaller error than the approximated policy $\tilde{\pi}_t^*$, with both being compared to the optimal policy. Although the errors remain relatively large for both policies, it is noteworthy that the learned policy achieves better performance despite being trained using the approximated policy. This improvement can be attributed to the presence of the regularization parameter.

Indeed, when selecting the optimal action by computing the max operator within a linear model, the resulting action values tend to diverge (Equation 3.5). The only factor limiting this divergence is the immediate reward. As a consequence, the actions selected by the approximated policy are more aggressive, whereas the learned policy directly penalizes large actions through the regularization parameter λ, leading to more stable behavior.

To illustrate this effect in practice, Figure 5.8 shows trajectories generated by the optimal policy, the approximated policy, and the learned policy, all starting from the same randomly selected initial state from the test set. In the figure, the origin is marked with a red cross, and the car evolves from the same initial condition by following the action-induced transitions at each time step. To create a clearer and more informative plot, the axes have been limited to the range $[-35, 35]$, even though the cars may leave this range.

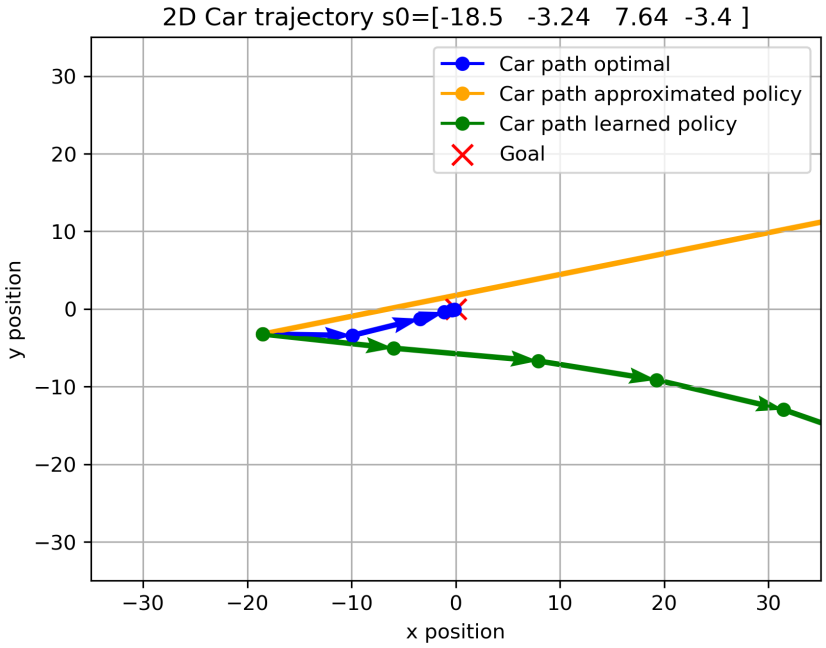


Figure 5.8: Car trajectories linear model and random sampling FVI

5.3.2 Quadratic model

The choice of the quadratic model (Section 3.4.2) is motivated by the analytical form of the optimal state-value and action-value functions (Equations 4.6 and 4.15), which are both quadratic. In this section, we focus primarily on different sampling techniques in order to investigate whether, given a perfect model, different datasets lead to different approximation errors.

The hyperparameter λ is consistently tuned by evaluating 50 logarithmically spaced values in the range $\lambda \in [10^{-10}, 10^{1}]$ on the validation set.

Starting from the Fitted Value Iteration (FVI) algorithm (Algorithm 1) and the uniform dataset (Section 5.2.1), we report the training error in Table 5.3.

t	$(\hat{V}_t^*, \tilde{V}_t^*)$	$(\hat{V}_t^*, V_t^*)$	$(\hat{V}_t^*, V_t^{\tilde{\pi}^*})$	$(\hat{\pi}_t^*, \pi_t^*)$	$(\tilde{\pi}_t^*, \pi_t^*)$
0	0.03	0.02	0.03	0.48	7.41
1	0.03	0.03	0.03	0.57	7.25
2	0.03	0.03	0.03	0.45	6.04
3	0.03	0.02	0.03	0.32	5.04
4	0.03	0.02	0.03	0.36	7.78
5	0	0	0		

Table 5.3: Training MAPE Quadratic model and random sampling FVI

The corresponding test error is reported in Table 5.4.

t	$(\hat{V}_t^*, \tilde{V}_t^*)$	$(\hat{V}_t^*, V_t^*)$	$(\hat{V}_t^*, V_t^{\tilde{\pi}^*})$	$(\hat{\pi}_t^*, \pi_t^*)$	$(\tilde{\pi}_t^*, \pi_t^*)$
0	0.03	0.02	0.04	0.21	2.56
1	0.03	0.02	0.03	0.22	2.76
2	0.03	0.03	0.04	0.33	3.58
3	0.03	0.02	0.03	0.15	3.7
4	0.03	0.02	0.03	0.54	5.07
5	0	0	0		

Table 5.4: Test MAPE Quadratic model and random sampling FVI

We can clearly observe that all the MAPE values associated with the state-value functions are close to zero. This indicates that, aside from a small amount of noise, the quadratic model is able to generalize the optimal state-value function almost perfectly.

Moreover, the learned policy $\hat{\pi}_t^*$ consistently exhibits a smaller error compared to the approximated policy $\tilde{\pi}_t^*$. This occurs because, even when the state-value function is approximated with high accuracy, the approximated policy still depends on the reward function and the transition dynamics, which contain noise, to compute the optimal action.

By contrast, an actor-critic method that is able to generalize the optimal policy directly avoids the propagation of noise after the learning phase. Therefore, using the learned policy $\hat{\pi}_t^*$ is preferable to relying on the approximated optimal policy $\tilde{\pi}_t^*$.

We further analyze the policies by visualizing the resulting trajectories. Figure 5.9 shows the trajectories of the cars starting from a randomly selected initial condition from the test set.

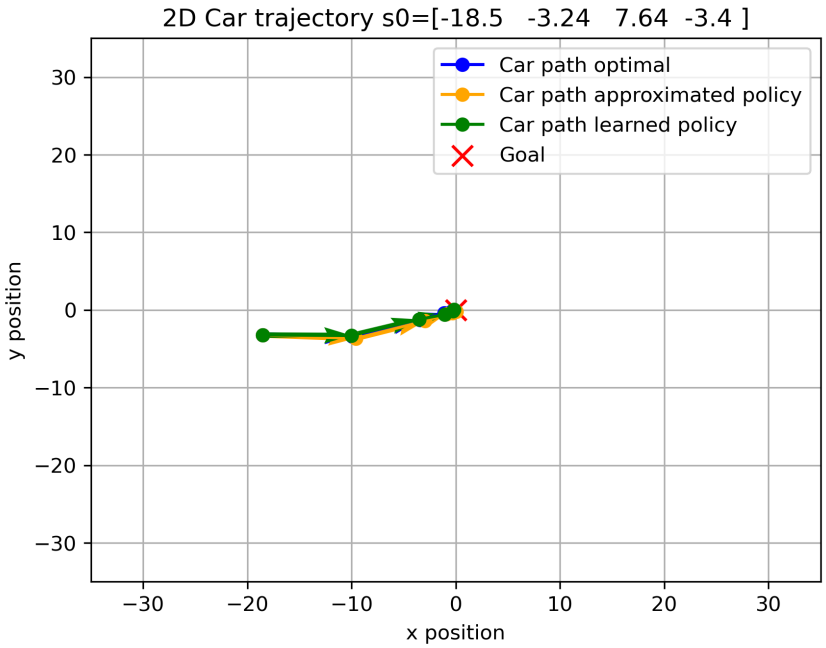
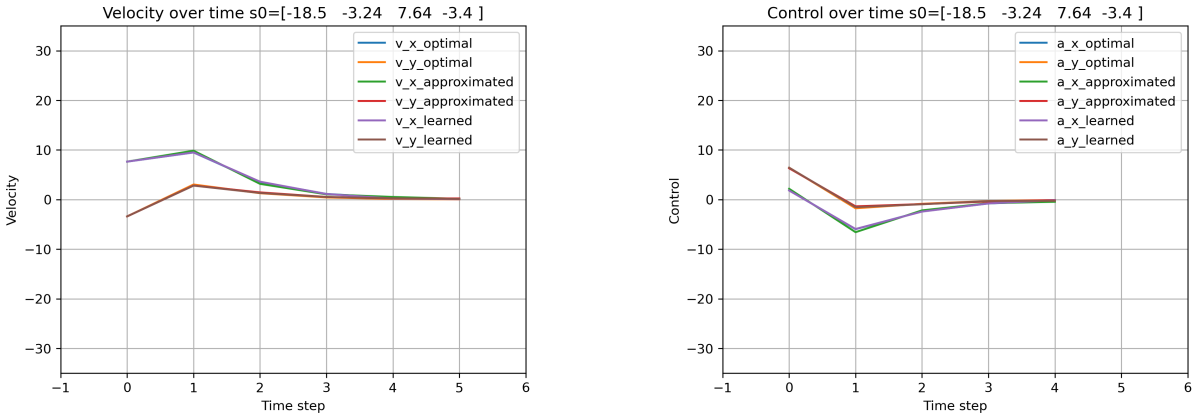


Figure 5.9: Car trajectories quadratic model and random sampling FVI

Figure 5.10 illustrates the velocity and acceleration profiles of the same trajectories over time t.



(a) Velocity

(b) Acceleration

Figure 5.10: Velocity and acceleration over time t of different policy

From these results, we observe that the trajectories generated by the learned policies are almost indistinguishable from the optimal ones.

It is important to emphasize that all state components position, velocity, and acceleration converge rapidly to zero when following the optimal policy. This behavior should be taken into account when interpreting MAPE values for variables of this type. As shown in Table 5.4, the approximated policy $\tilde{\pi}_t^*$ exhibits larger error compared to the learned policy $\hat{\pi}_t^*$. However, when the policy is rolled out, the resulting trajectories are almost equivalent. This is due to the presence of many small control actions in the dataset, for which even a unit error can correspond to a 100% relative error.

We now analyze the results obtained using the heuristic dataset (Section 5.2.1). The training errors are reported in Table 5.5.

t	$(\hat{V}_t^*, \tilde{V}_t^*)$	$(\hat{V}_t^*, V_t^*)$	$(\hat{V}_t^*, V_t^{\tilde{\pi}^*})$	$(\hat{\pi}_t^*, \pi_t^*)$	$(\tilde{\pi}_t^*, \pi_t^*)$
0	0.03	0.03	0.03	0.61	5.59
1	0.03	0.02	0.03	0.32	4.29
2	0.06	0.04	0.06	0.31	5.39
3	0.08	0.05	0.09	0.47	15.79
4	0.08	0.04	0.08	0.35	4.41
5	0.08	0.01	0.08		

Table 5.5: Training MAPE Quadratic model and heuristic sampling FVI

The corresponding test errors are reported in Table 5.6.

t	$(\hat{V}_t^*, \tilde{V}_t^*)$	$(\hat{V}_t^*, V_t^*)$	$(\hat{V}_t^*, V_t^{\tilde{\pi}^*})$	$(\hat{\pi}_t^*, \pi_t^*)$	$(\tilde{\pi}_t^*, \pi_t^*)$
0	0.03	0.03	0.04	0.24	3.45
1	0.03	0.04	0.03	0.16	3.62
2	0.04	0.05	0.04	0.21	2.88
3	0.04	0.06	0.04	0.16	3.57
4	0.04	0.06	0.05	0.93	5.41
5	0.01	0.01	0.01		

Table 5.6: Test MAPE Quadratic model and heuristic sampling FVI

We also analyze the results obtained using the Greedy Multistep Lookahead dataset (Section 5.2.3), which is considered jointly with the heuristic dataset due to their very similar results. The training errors for this dataset are reported in Table 5.7.

t	$(\hat{V}_t^*, \tilde{V}_t^*)$	$(\hat{V}_t^*, V_t^*)$	$(\hat{V}_t^*, V_t^{\tilde{\pi}^*})$	$(\hat{\pi}_t^*, \pi_t^*)$	$(\tilde{\pi}_t^*, \pi_t^*)$
0	0.03	0.03	0.04	0.32	4.26
1	0.05	0.02	0.06	0.85	13.95
2	0.1	0.05	0.12	0.93	13.61
3	0.26	0.1	0.31	2.51	16.99
4	0.45	0.14	0.59	3.04	36.1
5	0.66	0.02	0.6		

Table 5.7: Training MAPE Quadratic model and greedy sampling FVI

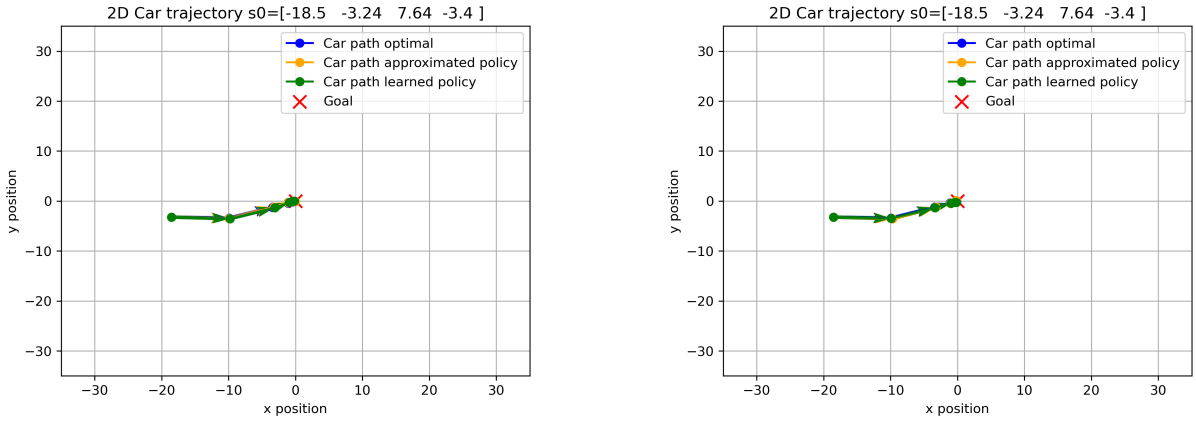
The corresponding test errors are reported in Table 5.8.

t	$(\hat{V}_t^*,\ \tilde{V}_t^*)$	$(\hat{V}_t^*,\ V_t^*)$	$(\hat{V}_t^*,\ V_t^{\tilde{\pi}^*})$	$(\hat{\pi}_t^*,\ \pi_t^*)$	$(\tilde{\pi}_t^*,\ \pi_t^*)$
0	0.03	0.03	0.03	0.25	2.8
1	0.05	0.03	0.05	0.8	2.72
2	0.13	0.16	0.13	0.55	2.83
3	0.09	0.15	0.13	0.89	3.7
4	0.13	0.15	0.12	1.63	9.18
5	0.02	0.02	0.02		

Table 5.8: Test MAPE Quadratic model and greedy sampling FVI

We observe that the test errors obtained with both the heuristic and greedy datasets are very similar to those obtained using the random dataset. This indicates that the model is able to learn the optimal state-value function effectively, regardless of the sampling strategy adopted.

Figure 5.11 shows the rollout trajectories starting from the same randomly chosen initial condition, for the two datasets used during training. In both cases, the resulting trajectories are almost indistinguishable from the optimal one.



(a) Heuristic dataset

(b) Greedy Multistep Lookahead dataset

Figure 5.11: Car trajectories quadratic model and heuristic and greedy sampling FVI

The only noteworthy aspect is that, for larger values of t, the training errors are higher than the corresponding test errors. This behavior arises because the adopted sampling strategies aim to mimic the optimal state distribution. As a consequence, for time steps close to the end of the horizon T, both the optimal state-value function and the optimal policy take values close to zero in the training dataset. As previously discussed, this leads to larger MAPE values even when the absolute error remains very small.

Finally, we analyze the Fitted Q-Iteration (FQI) algorithm (Algorithm 2).

The choice of the function approximator for this algorithm must be made carefully, since the approximated policy is obtained by maximizing the learned action-value function (Equation 3.10). Therefore, the model must guarantee the existence of a well-defined maximum for every input state.

The quadratic model does not provide such a guarantee in general, as it depends on the

concavity of the learned quadratic function. However, since the true optimal action-value function is quadratic and admits a unique maximizer, we nevertheless conducted the experiments under the assumption that the learned functions would preserve the correct concavity.

Here, we focus on the results obtained using random sampling (Section 3.3.1). The results corresponding to the other sampling techniques are very similar to those obtained with the uniform dataset and therefore are not discussed further.

The training errors for this dataset are reported in Table 5.9.

t	$(\hat{Q}_t^*, \tilde{Q}_t^*)$	$(\hat{Q}_t^*, Q_t^*)$	$(\hat{Q}_t^*, Q_t^{\tilde{\pi}^*})$	$(\hat{\pi}_t^*, \pi_t^*)$	$(\tilde{\pi}_t^*, \pi_t^*)$
0	0.03	0.02	0.04	0.11	0.11
1	0.03	0.01	0.04	0.05	0.05
2	0.03	0.01	0.03	0.06	0.06
3	0.03	0.01	0.03	0.05	0.05
4	0.02	0	0.02	0.02	0.03
5	0	0	0		

Table 5.9: Training MAPE Quadratic model and random sampling FQI

The corresponding test errors are reported in Table 5.10.

t	$(\hat{Q}_t^*, \tilde{Q}_t^*)$	$(\hat{Q}_t^*, Q_t^*)$	$(\hat{Q}_t^*, Q_t^{\tilde{\pi}^*})$	$(\hat{\pi}_t^*, \pi_t^*)$	$(\tilde{\pi}_t^*, \pi_t^*)$
0	0.03	0.02	0.04	0.05	0.05
1	0.03	0.01	0.03	0.03	0.03
2	0.03	0.01	0.04	0.04	0.04
3	0.03	0.01	0.04	0.02	0.02
4	0.02	0	0.02	0.03	0.03
5	0	0	0		

Table 5.10: Test MAPE Quadratic model and random sampling FQI

We observe that, also in this case, the quadratic model is able to learn the optimal action-value function and the corresponding policy with high accuracy, apart from a small amount of noise. This confirms that, when the model is correctly specified, no significant difficulties arise in approximating the optimal solution.

An important aspect to highlight is that the approximated policy $\tilde{\pi}_t^*$ exhibits a significantly smaller error compared to the one obtained using the FVI algorithm, and is much closer to the learned policy $\hat{\pi}_t^*$. This empirical result supports the hypothesis formulated in the Background chapter (Chapter 2), namely that when the primary objective is to learn an optimal policy, working directly with the action-value function is often preferable, as it provides a more direct way to compute the policy.

5.3.3 Polynomial kernel model

The choice of the polynomial kernel model is motivated by the aim of assessing whether a more expressive model, which implicitly contains a quadratic function that corresponds to the true solution, can generalize the problem without overfitting.

The hyperparameters are fixed as follows: the polynomial degree is set to $d = 3$, allowing for polynomials up to third order, while the regularization parameter λ is selected by evaluating 50 logarithmically spaced values in the range $\lambda \in [10^{-10}, 10^{1}]$ on the validation set.

We analyze only the FVI algorithm (Algorithm 1), since as we metion in the Quadratic model result (Section 5.3.2), the FQI algorithm need the guarantee to have a max to any input, and in this case the Polynomial kernel model has no one guarantee.

Starting from the random sampling dataset (Section 3.3.1), the Table 5.11 report the training error.

t	$(\hat{V}_t^*,\ \tilde{V}_t^*)$	$(\hat{V}_t^*,\ V_t^*)$	$(\hat{V}_t^*,\ V_t^{\tilde{\pi}^*})$	$(\hat{\pi}_t^*,\ \pi_t^*)$	$(\tilde{\pi}_t^*,\ \pi_t^*)$
0	0.03	0.06	0.04	0.59	5.79
1	0.03	0.06	0.04	0.5	6.65
2	0.03	0.04	0.04	0.37	4.34
3	0.03	0.03	0.03	0.17	4.95
4	0.03	0.02	0.03	0.91	7.51
5	0	0	0		

Table 5.11: Training MAPE Polynomial kernel model and random sampling FVI

The corresponding test errors are reported in Table 5.12.

t	$(\hat{V}_t^*,\ \tilde{V}_t^*)$	$(\hat{V}_t^*,\ V_t^*)$	$(\hat{V}_t^*,\ V_t^{\tilde{\pi}^*})$	$(\hat{\pi}_t^*,\ \pi_t^*)$	$(\tilde{\pi}_t^*,\ \pi_t^*)$
0	0.03	0.07	0.05	0.29	2.84
1	0.04	0.06	0.05	0.22	2.9
2	0.04	0.05	0.04	0.23	3.25
3	0.03	0.04	0.03	0.15	3.68
4	0.03	0.02	0.03	0.9	5.5
5	0	0	0		

Table 5.12: Test MAPE Polynomial kernel model and random sampling FVI

We observe that the results are very similar to those obtained with the quadratic model (Section 5.3.2). This suggests that, even when using a more expressive model, no overfitting issues arise as long as the noise level remains moderate and the samples adequately cover the state space.

Similar results are also obtained with the heuristic dataset (Section 5.2.2), leading to the same conclusions as in the quadratic model case. For this reason, these results are not discussed further.

More interesting insights are obtained from the Greedy Multistep Lookahead dataset (Section 5.2.3). The Table 5.13 report the training error.

t	$(\hat{V}_t^*, \tilde{V}_t^*)$	$(\hat{V}_t^*, V_t^*)$	$(\hat{V}_t^*, V_t^{\tilde{\pi}^*})$	$(\hat{\pi}_t^*, \pi_t^*)$	$(\tilde{\pi}_t^*, \pi_t^*)$
0	0.03	0.04	0.03	0.45	3.94
1	0.05	0.05	0.06	1.46	36.66
2	0.1	0.1	0.11	1.48	11.68
3	0.23	0.22	0.31	3.02	17.71
4	0.43	0.22	0.53	2.76	41.06
5	0.62	0.09	0.62		

Table 5.13: Training MAPE Polynomial kernel model and greedy sampling FVI

The corresponding test errors are reported in Table 5.14.

t	$(\hat{V}_t^*, \tilde{V}_t^*)$	$(\hat{V}_t^*, V_t^*)$	$(\hat{V}_t^*, V_t^{\tilde{\pi}^*})$	$(\hat{\pi}_t^*, \pi_t^*)$	$(\tilde{\pi}_t^*, \pi_t^*)$
0	0.03	0.04	0.03	0.26	3.08
1	0.18	0.18	0.18	0.74	2.7
2	0.59	0.59	0.59	0.87	3.17
3	0.19	0.2	0.19	0.82	3.41
4	0.27	0.25	0.25	1.44	6.38
5	0.59	0.59	0.59		

Table 5.14: Test MAPE Polynomial kernel model and greedy sampling FVI

Here, differently from the quadratic model, both the training and test errors tend to decrease for lower values of t. In the quadratic case, this behavior is observed only for the training error, which is due to the composition of the training dataset. This indicates that the polynomial kernel model exhibits slight overfitting when using the greedy dataset at larger time step t. The reason is that the greedy dataset emphasizes trajectories that, for any initial state s_0, terminate very close to the origin. As a result, even when some noise is added to the input sample, the dataset at the last time steps T does not sufficiently cover the state space. This concept becomes even clearer in the results obtained with the Gaussian model (Section 5.3.4).

The most informative error to examine is $(\hat{V}_t^*, V_t^{\tilde{\pi}^*})$, which measures the difference between the state-value function of the learned model and that of the rollout approximated policy. From this, we can see that, when starting from $t = 0$, the generalization of the state-value function is good, even though slight overfitting occurs at later time steps. This is because overfitting mainly affects the model's ability to generalize to states far from the origin. However, when starting from a well-represented initial state and rolling out the trajectories, the states remain close to the optimal ones at each time step t, which aligns with the coverage provided by the greedy dataset. This is illustrated more clearly in Figure 5.12, where the trajectories are nearly identical.

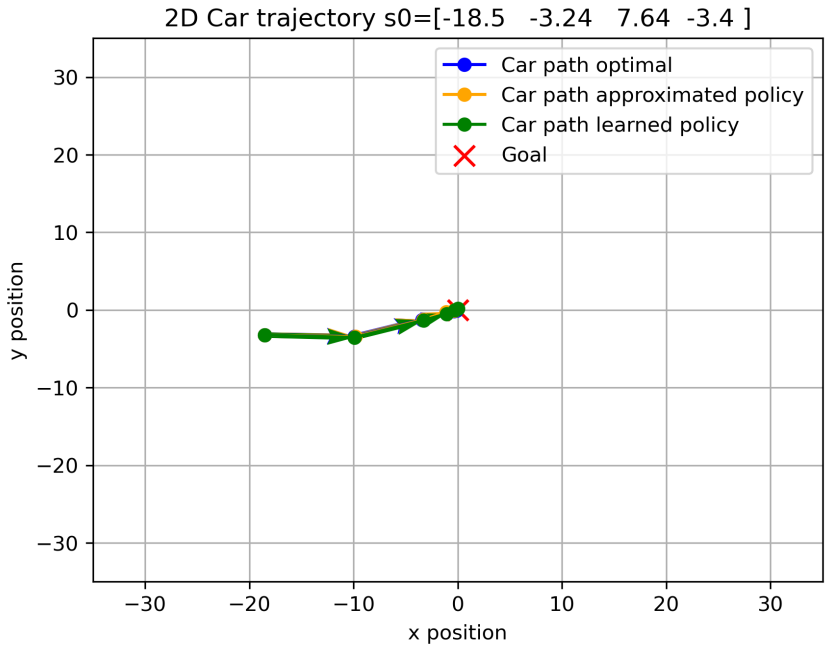


Figure 5.12: Car trajectories Polynomial kernel model and greedy sampling FVI

5.3.4 Gaussian kernel model

The choice of the Gaussian kernel is motivated by the need to assess the performance of a highly flexible model capable of capturing complex patterns, even when applied to this relatively simple quadratic problem. The focus is on identifying in which datasets and under which conditions we risk overfitting and encountering the curse of dimensionality.

The hyperparameters are selected through grid search. In particular, the regularization parameter λ is chosen by evaluating 50 logarithmically spaced values in the range $\lambda \in [10^{-10}, 10^{1}]$, while the kernel bandwidth parameter σ is selected by evaluating 50 logarithmically spaced values in the range $\sigma \in [10^{-2}, 10^{4}]$. Both parameters are selected based on performance on the validation set.

We begin by analyzing the results of the FVI algorithm (Algorithm 1) with random sampling (Section 3.3.1). Table 5.15 reports the training error.

t	$(\hat{V}_t^*, \tilde{V}_t^*)$	$(\hat{V}_t^*, V_t^*)$	$(\hat{V}_t^*, V_t^{\tilde{\pi}^*})$	$(\hat{\pi}_t^*, \pi_t^*)$	$(\tilde{\pi}_t^*, \pi_t^*)$
0	0.1	0.28	0.29	1.27	5.26
1	0.11	0.28	0.29	1.11	10.08
2	0.1	0.25	0.27	1.1	4.8
3	0.09	0.24	0.26	1.32	5.17
4	0.1	0.28	0.29	1.19	6.5
5	0.08	0.08	0.08		

Table 5.15: Training MAPE Gaussian kernel model and random sampling FVI

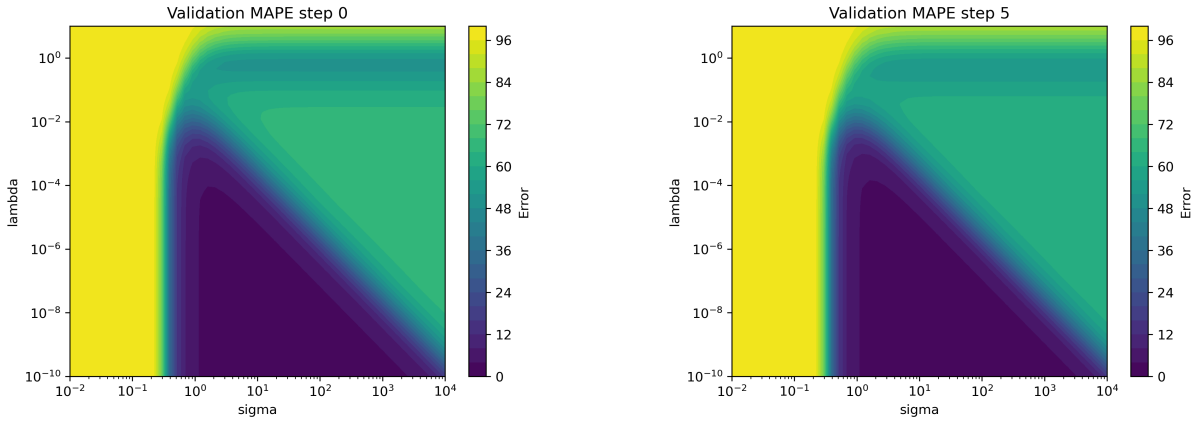
The corresponding test errors are reported in Table 5.16.

t	$(\hat{V}_t^*, \tilde{V}_t^*)$	$(\hat{V}_t^*, V_t^*)$	$(\hat{V}_t^*, V_t^{\tilde{\pi}^*})$	$(\hat{\pi}_t^*, \pi_t^*)$	$(\tilde{\pi}_t^*, \pi_t^*)$
0	1.1	1.14	1.15	0.96	2.92
1	1.17	1.14	1.16	0.68	3.52
2	1.08	1.08	1.09	0.9	3.49
3	0.92	0.97	0.98	1	3.87
4	0.99	1	1.01	1.12	4.63
5	0.77	0.77	0.77		

Table 5.16: Test MAPE Gaussian kernel model and random sampling FVI

We observe that the test error is slightly higher than the training error for all time steps t, but the difference is limited and does not raise significant concerns. All errors are comparable to those obtained with the quadratic model (Section 5.3.2), indicating that the Gaussian kernel model is able to learn the true optimal function effectively when trained on the random, but enoughbig, dataset.

When using a more expressive model, greater care must be taken in selecting the hyperparameters. Poor choices can quickly introduce bias, and due to error propagation across iterations of the algorithm, this may lead to highly suboptimal policies and large errors, as observed in the linear model (Section 3.4.1). To better understand this aspect, we report the validation MAPE for the hyperparameter grid at time steps $t = 5$ and $t = 0$. In Figure 5.13, the x-axis represents σ and the y-axis represents λ.



(a) $t = 5$ (b) $t = 5$

Figure 5.13: validation MAPE Gaussian kernel model and random sampling FVI

The results are very similar for the two time steps. In both cases, there exists a combination of hyperparameters that allows the model to learn the problem accurately. Conversely, inappropriate hyperparameter choices lead to a sharp increase in validation error, as the model either fails to generalize and fails to learn the underlying structure of the problem.

We now move to the analysis of the heuristic dataset (Section 5.2.2). Table 5.17 reports the training error.

t	$(\hat{V}_t^*, \tilde{V}_t^*)$	$(\hat{V}_t^*, V_t^*)$	$(\hat{V}_t^*, V_t^{\tilde{\pi}^*})$	$(\hat{\pi}_t^*, \pi_t^*)$	$(\tilde{\pi}_t^*, \pi_t^*)$
0	0.12	0.4	0.48	1.99	6.43
1	0.11	0.4	0.43	1.9	6.54
2	0.3	0.46	0.5	0.67	5.11
3	0.18	1.17	1.05	25.81	15.55
4	0.22	0.29	0.32	0.77	6.8
5	0.2	0.2	0.21		

Table 5.17: Training MAPE Gaussian kernel model and heuristic sampling FVI

The corresponding test errors are reported in Table 5.18.

t	$(\hat{V}_t^*, \tilde{V}_t^*)$	$(\hat{V}_t^*, V_t^*)$	$(\hat{V}_t^*, V_t^{\tilde{\pi}^*})$	$(\hat{\pi}_t^*, \pi_t^*)$	$(\tilde{\pi}_t^*, \pi_t^*)$
0	1.14	1.29	1.3	0.95	3.35
1	11.9	11.61	11.71	2.08	5.73
2	19.41	23.84	24.42	0.45	7.48
3	42.23	47.11	50.95	7.14	27.15
4	52.8	60.18	62.74	1.83	39.68
5	76.29	76.29	76.29		

Table 5.18: Test MAPE Gaussian kernel model and heuristic sampling FVI

In this case, the model clearly tends to overfit the training set. Moreover, for larger time steps t, the training data provide limited coverage of the state space, resulting in poor generalization performance. This behavior becomes particularly evident when comparing training and test errors at higher values of t. This is due to the fact that, for larger values of t, the sub-optimal samples tend to concentrate around the origin and therefore fail to provide sufficient coverage of the remaining regions of the state space.

To further investigate this phenomenon, Figure 5.14 shows the validation error surface at time $t = 5$. The validation errors are comparable to the training errors, suggesting that the main issue is insufficient state-space coverage during training rather than hyperparameter selection.

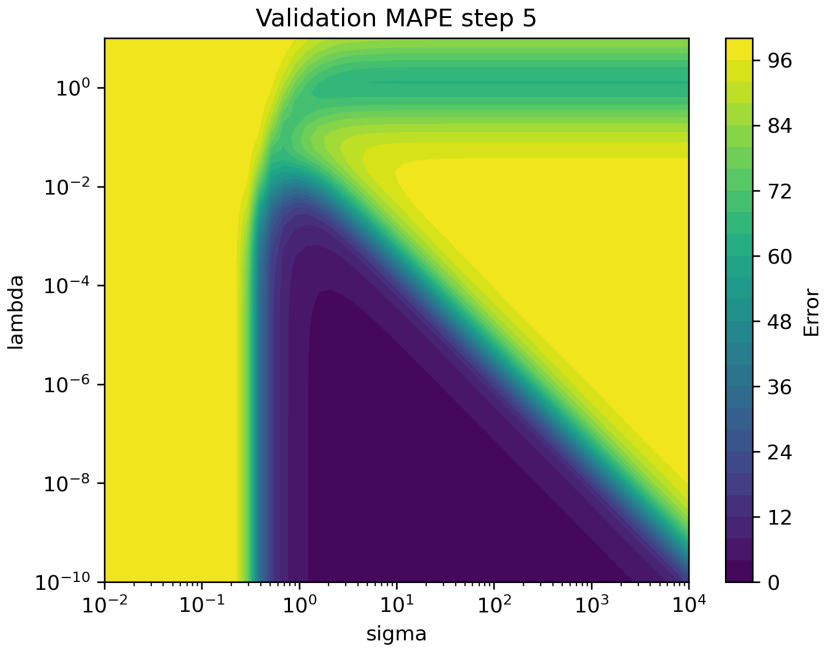


Figure 5.14: validation MAPE Gaussian kernel model and Heuristic sampling FVI $t = 5$

An interesting aspect of this sampling method, which attempts to mimic the optimal distribution, is that the three test MAPE errors decrease as t decreases. This indicates that, although the model may not generalize well over the entire state space, it is capable of generating trajectories that are very close to the optimal ones.

Indeed, at time $t = 0$, even though overfitting is evident at other time steps, the training and test errors are very similar, including those relative to the true optimal state-value function and policy. This observation is further supported by the rollout trajectories shown in Figure 5.15, where the three trajectories are nearly identical.

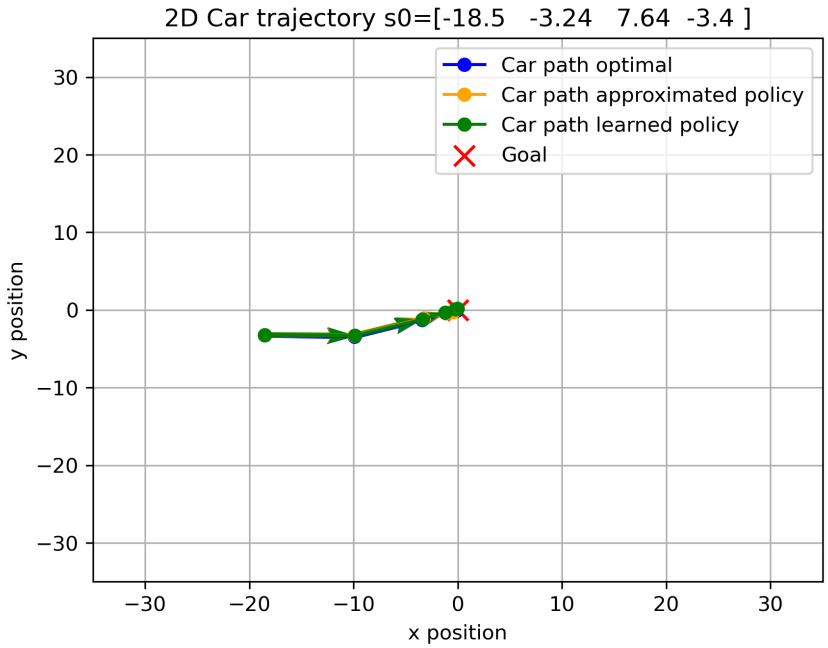


Figure 5.15: Car trajectories Gaussian kernel model and heuristic sampling FVI

However, caution is required when using this type of sampling technique. In this simple problem, the sampling strategy successfully identifies trajectories close to optimal. In more complex problems, indeed, combining a highly expressive model with a sampling strategy that concentrates around near-optimal distributions, without adequately covering the full state space, may result in a model that learns near-optimal trajectories very accurately but fails to generalize and identify the truly optimal actions.

Another important observation is that the learned policy $\hat{\pi}_*$ exhibits lower error than the approximated policy, even on the test set. This suggests that the actor-critic approach improves generalization.

It is important to remember that the learned policy relies on a model that closely approximates the true optimal policy.

Finally, we conclude the analysis of the FVI algorithm by considering the Greedy Multi-step Lookahead sampling dataset (Section 5.2.3). Table 5.19 reports the training error.

t	$(\hat{V}_t^*, \tilde{V}_t^*)$	$(\hat{V}_t^*, V_t^*)$	$(\hat{V}_t^*, V_t^{\tilde{\pi}^*})$	$(\hat{\pi}_t^*, \pi_t^*)$	$(\tilde{\pi}_t^*, \pi_t^*)$
0	0.11	0.34	1.18	8.48	5.84
1	0.35	0.56	0.97	89.51	55.82
2	0.47	0.6	0.68	5.08	13.69
3	0.33	0.54	0.69	3.06	20.19
4	0.24	0.62	0.79	3.8	30.6
5	0.21	0.69	0.92		

Table 5.19: Training MAPE Gaussian kernel model and greedy sampling FVI

The corresponding test errors are reported in Table 5.18.

t	$(\hat{V}_t^*, \tilde{V}_t^*)$	$(\hat{V}_t^*, V_t^*)$	$(\hat{V}_t^*, V_t^{\tilde{\pi}^*})$	$(\hat{\pi}_t^*, \pi_t^*)$	$(\tilde{\pi}_t^*, \pi_t^*)$
0	0.99	1.02	1	7.73	4.67
1	15.57	18.7	31.18	23.18	32.47
2	36.07	47.21	63.46	2.93	66.03
3	72.94	79.79	87.86	1.14	79.46
4	89.01	91.85	94.19	1.8	101.23
5	94.81	94.81	94.81		

Table 5.20: Test MAPE Gaussian kernel model and greedy sampling FVI

The results are similar to those obtained with heuristic sampling, showing the same general trend but with larger test errors. This behavior can be explained by the same underlying issue: both sampling methods attempt to mimic the optimal distribution. However, the greedy approach concentrates even more heavily on near-optimal regions and explores the non-optimal areas of the state space less thoroughly, which leads to increased overfitting and poorer generalization performance.

Even in this case, although overfitting is clearly present, when we generate rollout trajectories starting from time step $t = 0$, as done during dataset construction, the resulting trajectories remain very close to the optimal ones, even when evaluated on the test set. In other words, despite the limited global generalization performance, the policy behaves well along the regions of the state space that are actually visited during execution.

Figure 5.16 shows the trajectories obtained with the model trained on the greedy dataset.

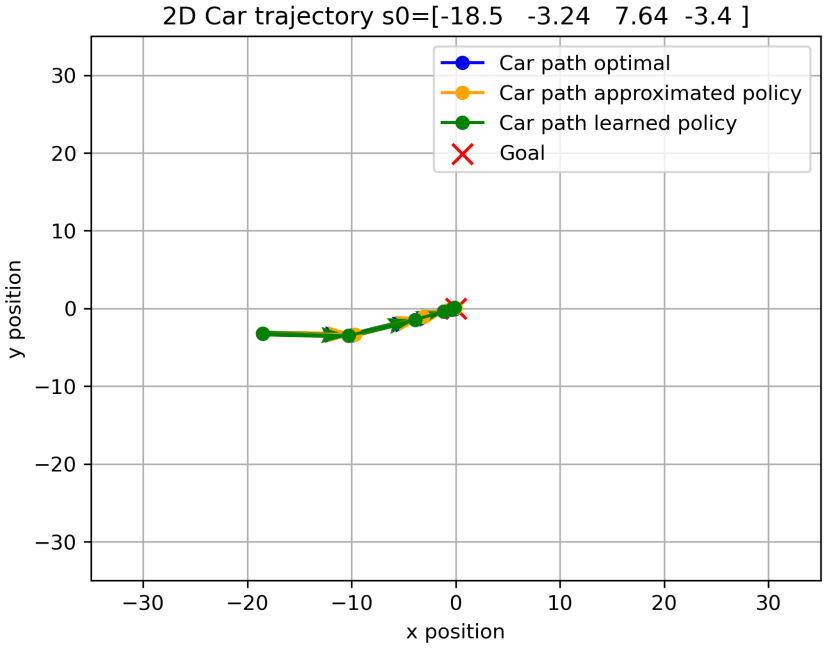


Figure 5.16: Car trajectories Gaussian kernel model and greedy sampling FVI

This behavior can be explained by the nature of the greedy sampling strategy, which concentrates most of the training data around near-optimal trajectories. As a consequence,

the model becomes highly accurate in those specific regions, while its performance deteriorates in less explored areas of the state space. However, since the rollout starts from $t = 0$ and follows a near-optimal path, the system remains within the well-covered region. Therefore, the lack of global generalization does not significantly affect the quality of the generated trajectories.

To mitigate the overfitting issue, several strategies can be considered.

One approach, already mentioned in Sections 3.3.2 and 3.3.3, consists of introducing additional exploration during data collection. Instead of merely adding noise to the selected action, a completely random action can be taken with some probability p. This increases exploration of the state space and improves coverage. However, this strategy may generate trajectories that deviate substantially from realistic or near-optimal system behavior.

Another possible solution is to construct a joint dataset that combines uniform and greedy sampling. In this way, the dataset would contain both information from near-optimal trajectories (which improves performance along relevant paths) and samples from the broader state space (which enhances generalization). Such a hybrid strategy could provide a better balance between exploitation and exploration.

If the objective is only to learn an optimal policy starting from $t = 0$ in a relatively simple problem, relying exclusively on the greedy dataset may still be acceptable. As observed, even in the presence of overfitting, the resulting rollout trajectories remain very close to the optimal ones. In this case, greedy sampling may also reduce computational cost and improve training efficiency, since learning is concentrated on the most relevant regions of the state space.

This phenomenon is an instance of the distribution shift problem: the model performs well under the state distribution induced by the greedy policy but generalizes poorly to regions of the state space not represented in that distribution, like the random dataset used for testing [Lev+20].

We now focus on the results obtained with the FQI algorithm (Algorithm 2). The Gaussian kernel model is the only function approximator presented that, by construction, admits a well-defined maximum with respect to the action variable. This structural property is necessary for the FQI algorithm in order to solve Equation 3.10, which requires maximizing the action-value function over the action space.

Starting from the Random dataset (Section 5.2.1), Table 5.21 reports the training error.

t	$(\hat{Q}_t^*, \tilde{Q}_t^*)$	$(\hat{Q}_t^*, Q_t^*)$	$(\hat{Q}_t^*, Q_t^{\tilde{\pi}^*})$	$(\hat{\pi}_t^*, \pi_t^*)$	$(\tilde{\pi}_t^*, \pi_t^*)$
0	0.19	33.05	18.9	52.85	54.21
1	0.13	22.83	15.04	58.06	58.52
2	0.14	16.49	13.27	39.16	44.13
3	0.1	12.24	13.25	35.87	42.52
4	0.11	8.62	8.62	26.19	43.49
5	0.08	0.08	0.08		

Table 5.21: Training MAPE Gaussian kernel model and random sampling FQI

The corresponding test errors are reported in Table 5.22.

t	$(\hat{Q}_t^*, \tilde{Q}_t^*)$	$(\hat{Q}_t^*, Q_t^*)$	$(\hat{Q}_t^*, Q_t^{\tilde{\pi}^*})$	$(\hat{\pi}_t^*, \pi_t^*)$	$(\tilde{\pi}_t^*, \pi_t^*)$
0	26.52	39.19	25.52	34.05	36.91
1	15.71	28.93	21.7	40.74	46.78
2	8.96	19.9	17.66	32.66	34.85
3	6.16	14.89	15.98	32.58	36.2
4	10.83	10.83	10.83	28.95	27.53
5	0.77	0.77	0.77		

Table 5.22: Test MAPE Gaussian kernel model and random sampling FQI

Here we can observe several interesting results.

Firstly, we can see that the performance is much worse compared to the FVI algorithm and to the FQI algorithm with the quadratic model. This is because we are using the same dataset for a more complex problem.

In FVI we aim to learn the value of being in a given state s at time t. Instead, in FQI we also need to generalize the consequence of taking an action a given a state s at time t. This adds complexity and, more importantly, increases the dimensionality of the problem.

As we can see from the results, with only 1000 samples the model is not able to generalize over the action space and tends to overfit the training data, since the test errors are significantly larger than the training errors.

Another important aspect to highlight is the role of random sampling in the action space. If the dataset does not contain enough actions close to the optimal one for each state, the model may struggle to infer the correct maximization structure. A non-parametric model such as the Gaussian kernel cannot reliably extrapolate the optimal action if the training data do not include sufficiently informative near-optimal samples. This limitation contributes to overfitting and poor generalization.

This behavior contrasts with that of the quadratic model. Since the quadratic model perfectly matches the true functional form of the optimal action-value function, it does not require explicit samples of the optimal action in the training set. Instead, it only needs sufficient information about the local curvature (i.e., the slope and second-order structure) to estimate the coefficients correctly. Therefore, it can generalize effectively even with limited action coverage.

The errors increase as t decreases, which can be attributed to bias propagation. Errors made at later time steps are propagated backward through the Bellman updates, amplifying their impact at earlier time steps.

Finally, the training MAPE computed with respect to the rollout action-value function $Q_t^{\tilde{\pi}^*}$ already indicates the presence of overfitting. Its trend closely mirrors that of the MAPE computed with respect to the true optimal action-value function Q_t^*. This observation suggests that monitoring the rollout-based error can serve as a useful proxy for detecting overfitting when a separate test set is not available. In scenarios with limited data, where all samples must be used for training and validation, this diagnostic can provide valuable insight into the generalization behavior of the model.

In Figure 5.17 we show the rollout trajectories of the model starting from a random point of the test set.

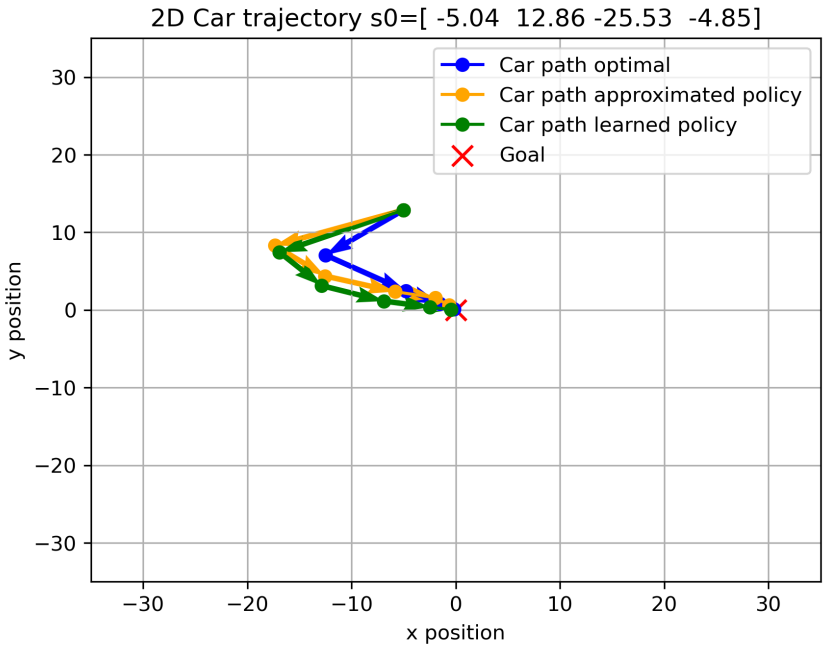


Figure 5.17: Car trajectories Gaussian kernel model and random sampling FQI

We can see that even though the model overfits, the rollout trajectories are still good.

To conclude the analysis, we present the results obtained using the Greedy Multistep Lookahead dataset (Section 5.2.3). The Heuristic dataset (Section 5.2.2) is omitted, as it yields very similar results and leads to the same conclusions.

Table 5.23 reports the training errors.

t	$(\hat{Q}_t^*, \tilde{Q}_t^*)$	$(\hat{Q}_t^*, Q_t^*)$	$(\hat{Q}_t^*, Q_t^{\tilde{\pi}^*})$	$(\hat{\pi}_t^*, \pi_t^*)$	$(\tilde{\pi}_t^*, \pi_t^*)$
0	0.12	1.57	6.57	191.94	170.91
1	0.42	1.38	5.12	744.25	461.65
2	0.44	2.13	2.41	71.43	172.14
3	0.28	1.5	1.66	72.5	110.56
4	0.16	0.47	0.61	49.06	65.94
5	0.21	0.69	0.9		

Table 5.23: Training MAPE Gaussian kernel model and greedy sampling FQI

The corresponding test errors are reported in Table 5.24.

t	$(\hat{Q}_t^*, \tilde{Q}_t^*)$	$(\hat{Q}_t^*, Q_t^*)$	$(\hat{Q}_t^*, Q_t^{\tilde{\pi}^*})$	$(\hat{\pi}_t^*, \pi_t^*)$	$(\tilde{\pi}_t^*, \pi_t^*)$
0	30.18	57.63	83.74	161.37	156.6
1	23.22	58.49	82.51	121.66	118.77
2	38.15	77.63	86.87	41.44	121.7
3	73.4	91.25	93.23	35.44	174.17
4	95.39	95.39	95.39	32.78	336
5	94.81	94.81	94.81		

Table 5.24: Test MAPE Gaussian kernel model and greedy sampling FQI

We observe once again that the model overfits, although in this case the phenomenon is more pronounced.

On the training set, the MAPE with respect to the true optimal action-value function Q_t^* is very small, indicating an almost perfect fit even on the real optimal values. However, the actions selected by the approximated policy $\tilde{\pi}_t^*$ exhibit very large MAPEs. This behavior can be explained by the fact that the model accurately fits the near-optimal trajectories contained in the dataset but fails to generalize over the rest of the action space. Indeed, apart from a small amount of injected noise, the dataset consists almost exclusively of near-optimal trajectories.

Since the model fails to generalize beyond the near-optimal trajectories contained in the dataset, the test errors become significantly larger, consistently with the distribution shift phenomenon previously discussed for the FVI case.

The resulting trajectories, initialized from random states in the test set, deviate substantially from the true optimal trajectories, as illustrated in Figure 5.18.

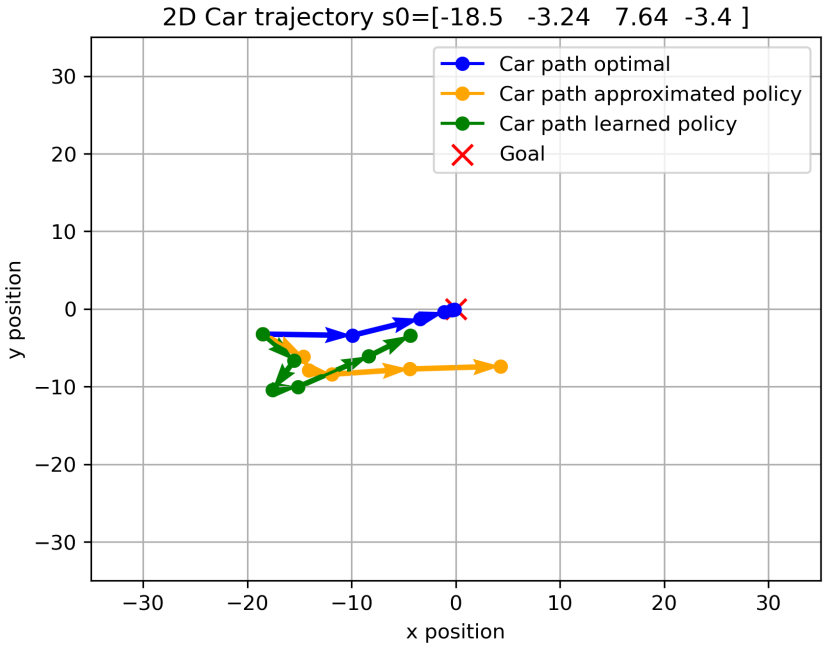


Figure 5.18: Car trajectories Gaussian kernel model and greedy sampling FQI

To mitigate the overfitting phenomenon observed across all three datasets, several workarounds

can be considered.

First, increasing the size of the dataset and combining random and greedy sampling strategies could potentially alleviate the issues arising from the intrinsic complexity of the problem. Such a dataset would contain both near-optimal and exploratory actions, thereby improving the model's ability to generalize. Moreover, a larger number of samples may help counterbalance the increase in dimensionality and model complexity.

A second workaround, specific to the Gaussian kernel model, consists in introducing different bandwidth parameters instead of using a single common value of σ. In the standard formulation, a unique σ imposes the same level of smoothness across all state and action components. However, in reinforcement learning problems, different state variables and actions typically influence the system dynamics and the return with different sensitivities. Some components may have a strong impact on the value function, while others play a more marginal role. Using dimension-dependent bandwidths allows the model to reflect these differences, adapting its smoothness to the relative importance of each component and potentially improving generalization.

As mentioned in the Sampling Techniques section (Section 3.3), these issues have been extensively studied in the reinforcement learning literature, and several approaches have been proposed to mitigate them; however, challenges related to generalization, overfitting, and distributional coverage are still actively studied in reinforcement learning [SB18; MS08; Mun05; Yan+20; San+24].

Chapter 6

Conclusion

The primary objective of this thesis was to gain a deeper understanding of what is effectively learned when approximate methods are employed in Reinforcement Learning.

We began by introducing the necessary notation and theoretical framework in order to clarify why approximation methods are required and how they are formally defined. We then compared the performance of the two algorithms under study, Fitted Value Iteration and Fitted Q-Iteration, under different sampling strategies within a Linear Quadratic Regulator environment, whose optimal solution is known to be quadratic for both objective functions.

When adopting a linear model for function approximation, the algorithm consistently underfit the underlying problem across all sampling techniques, resulting in unstable policies. This behavior highlights the inadequacy of a misspecified function class when the true solution lies outside the representational capacity of the model.

In contrast, quadratic models and polynomial kernel models successfully recover the underlying solution across all considered sampling strategies and algorithms. This outcome is consistent with the fact that these models belong to the same functional class as the true solution of the Linear Quadratic Regulator, thereby eliminating approximation bias.

The Gaussian kernel model yields the most informative results, as it reveals both the strengths and limitations of the adopted methods. In particular, Fitted Value Iteration appears to approximate a structurally simpler function compared to Fitted Q-Iteration. As a consequence, it requires fewer samples and achieves stable performance across all the different sampling strategies. Fitted Q-Iteration, on the other hand, faces greater difficulty when trained on the same datasets, due to the higher complexity of the function it seeks to approximate. However, this additional complexity result in a more direct derivation of the approximated policy in the FQI algorithm.

The Gaussian model also highlights the critical role of sampling. Purely random sampling fails to exploit the structure induced by the optimal policy distribution, whereas sampling from a suboptimal distribution may lead to generalization issues and convergence toward suboptimal solutions. More robust performance can be obtained when combining datasets from different distributions or using more complex sampling techniques. These findings emphasize the importance of accounting for distribution shift when interpreting the results, as neglecting this aspect may lead to misleading conclusions.

Overall, the choice of sampling strategy strongly depends on the characteristics of the underlying problem and algorithm, and also on the specific objectives of the analysis.

The actor-critic approach, which jointly learns a policy and value functions through function approximation, also demonstrates strong empirical performance in this setting. However, it is important to note that the underlying problem is a relatively simple linear-quadratic system, which facilitates accurate approximation when the data are sufficiently informative.

A notable limitation of these approximate methods is their computational cost, as they aim to approximate multiple components of the Reinforcement Learning problem rather than directly optimizing a policy. Nevertheless, their main advantage lies precisely in this comprehensive perspective: instead of learning only an approximate optimal policy, they provide an approximation of the entire set of functions characterizing the Reinforcement Learning problem. This broader perspective allows us to inspect and analyze the learned value functions and action-value functions directly, thereby enabling a deeper understanding of what is effectively being learned by the model.

Future research directions include a more rigorous theoretical analysis of the guarantees associated with these approximation schemes, as well as a deeper investigation of the convergence properties of actor-critic methods. Such analysis could provide further insight into more complex settings in which policy and value functions are learned jointly in high-dimensional or nonlinear environments.

Bibliography

[AM89] Brian D. O. Anderson and John B. Moore. *Optimal Control: Linear Quadratic Methods*. Englewood Cliffs, NJ: Prentice Hall, 1989. ISBN: 978-0136386516.

[Ber76] Dimitri P. Bertsekas. *Dynamic Programming and Stochastic Control*. Academic Press, 1976.

[Ber19] Dimitri P. Bertsekas. *Reinforcement Learning and Optimal Control*. Athena Scientific, 2019.

[Boe+05] Pieter-Tjerk de Boer, Dirk P. Kroese, Shie Mannor, and Reuven Y. Rubinstein. "A Tutorial on the Cross-Entropy Method". In: *Annals of Operations Research* 134.1 (2005), pp. 19–67. DOI: 10.1007/s10479-005-5724-z.

[BH75] A. E. Bryson and Y.-C. Ho. *Applied Optimal Control: Optimization, Estimation, and Control*. Taylor & Francis, 1975.

[Lev+20] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. "Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems". In: *arXiv preprint arXiv:2005.01643* (2020). URL: https://arxiv.org/abs/2005.01643.

[Mac67] James B. MacQueen. "Some Methods for Classification and Analysis of Multivariate Observations". In: *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*. 1967, pp. 281–297.

[Mun05] Rémi Munos. "Error Bounds for Approximate Value Iteration". In: *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI)*. AAAI Press / The MIT Press, 2005, pp. 1006–1011. URL: https://aaai.org/Papers/AAAI/2005/AAAI05-159.pdf.

[MS08] Rémi Munos and Csaba Szepesvári. "Finite-Time Bounds for Fitted Value Iteration". In: *Journal of Machine Learning Research* 9 (2008), pp. 815–857. URL: https://www.jmlr.org/papers/volume9/munos08a/munos08a.pdf.

[Pow64] M. J. D. Powell. "An Efficient Method for Finding the Minimum of a Function of Several Variables Without Calculating Derivatives". In: *The Computer Journal* 7.2 (1964), pp. 155–162. DOI: 10.1093/comjnl/7.2.155.

[San+24] Pedro P. Santos, Diogo S. Carvalho, Alberto Sardinha, and Francisco S. Melo. "The Impact of Data Distribution on Q-learning with Function Approximation". In: *Machine Learning* 113 (2024), pp. 6141–6163. DOI: 10.1007/s10994-024-06564-5.

[SSM08] Bernhard Schölkopf, Alexander J. Smola, and Klaus-Robert Müller. "Kernel methods in machine learning". In: *The Annals of Statistics* 36.3 (2008), pp. 1171–1220.

[SB18] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. 2nd. Cambridge, MA, USA: MIT Press, 2018.

[Sut+99] Richard S. Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. "Policy Gradient Methods for Reinforcement Learning with Function Approximation". In: *Advances in Neural Information Processing Systems 12*. The MIT Press, 1999, pp. 1057–1063.

[Yan+20] Zhuoran Yang, Chi Jin, Zhaoran Wang, Mengdi Wang, and Michael I. Jordan. "On Function Approximation in Reinforcement Learning: Optimism in the Face of Large State Spaces". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2020.