Reinforcement learning in continuous state and action spaces

Busoniu, IL

Publication date
2009

Document Version
Final published version

Citation (APA)
Busoniu, IL. (2009). *Reinforcement learning in continuous state and action spaces*. [Dissertation (TU Delft), Delft University of Technology].

Reinforcement Learning in Continuous State and Action Spaces

Lucian Buşoniu

Reinforcement Learning in Continuous State and Action Spaces

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof.dr.ir. J.T. Fokkema,
voorzitter van het College van Promoties,
in het openbaar te verdedigen op dinsdag 13 januari 2009 om 12:30 uur
door

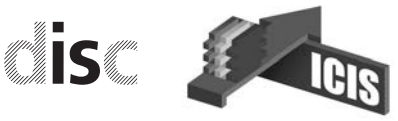
Ion Lucian BUŞONIU

Master of Science, Technische Universiteit van Cluj-Napoca, Roemenië,
geboren te Petroşani, Roemenië.

Dit proefschrift is goedgekeurd door de promotoren:
Prof.dr. R. Babuška, M.Sc.
Prof.dr.ir. B. De Schutter

Samenstelling promotiecommissie:

Rector Magnificus	voorzitter
Prof.dr. R. Babuška, M.Sc.	Technische Universiteit Delft, promotor
Prof.dr.ir. B. De Schutter	Technische Universiteit Delft, promotor
Prof.dr. R. Munos	INRIA Futurs Lille
Dr. D. Ernst	University of Liege
Prof.dr.ir. F.C.A. Groen	Universiteit van Amsterdam
Prof.drs.dr. L. Rothkrantz	Nederlandse Defensie Academie & Technische Universiteit Delft
Prof.dr. C. Witteveen	Technische Universiteit Delft
Prof.dr.ir. M. Verhaegen	Technische Universiteit Delft (reservelid)



This thesis has been completed in partial fulfillment of the requirements of the Dutch Institute for Systems and Control (DISC) for graduate studies. The research described in this thesis was financially supported by Senter, Ministry of Economic Affairs of the Netherlands within the BSIK-ICIS project “Interactive Collaborative Information Systems” (grant no. BSIK03024) and by the NWO Van Gogh grant VGP 79-99.

Published and distributed by: Lucian Buşoniu
E-mail: i.l.busoniu@tudelft.nl
Web: http://www.dcsc.tudelft.nl/∼lbusoniu

ISBN 978-90-9023754-1

Printed in the Netherlands

Acknowledgements

First and foremost, I thank my advisers, Professors Robert Babuška and Bart De Schutter, who complement each other so well. With Robert's endless enthusiasm, energy, and string of new ideas, it is impossible to get stuck with your research. With his steadfast support and encouragement, together with his attention to detail, Bart balances and completes the formula for perfect supervision. This does not mean that one lacked encouragement and support, or the other energy and new ideas — far from it! I thank them both for always taking time from their busy schedule (often even during evenings and weekends!) to read through and criticize my work, and to offer suggestions and ideas on how to improve it.

It has been great to work with Damien Ernst, owing to *his* own brand of endless enthusiasm, energy, and string of new ideas, together with his inspiring belief in our work. He also made possible my stay in France, which is a great country from so many points of view, starting with the restaurants and ending with the coastlines.

I would like to thank my former teacher and current friend, Paula, without whose support I would not even have started my PhD in Delft. And to my friend Zsófi, for her companionship and support. I am grateful to my friends back in Romania and elsewhere, Luminiţa, Laura, Sorin, Aurelian, Mugur, Radu — to name but a few — who made my holidays great and helped me regain energy for my research. My thanks also go to Professors Liviu Miclea and Dorin Isoc of the Technical University of Cluj-Napoca, who guided my first steps into the academia.

It has been a pleasure to work in the excellent environment of the Delft Center for Systems and Control, and this is thanks to all my colleagues, staff and students alike. I would like to mention especially Roland Tóth, Jelmer van Ast, Rudy Negenborn, Diederick Joosten, and (former colleague and current friend) Tudor Ionescu. My students Xu, Maarten, Sander, Sjoerd, and Thijs helped me with my work and allowed me to contribute in more ways than just doing research. I enjoyed collaborating with Erik Schuitema and his colleagues at BioMechanical Engineering within TUDelft.

I am grateful also to my colleagues in the Interactive Collaborative Information Systems project, especially to Martijn Neef for his help with administrative issues. I acknowledge the efforts of the members of my committee, who read through my thesis and provided helpful comments and suggestions: Professors Rémi Munos, Frans Groen, Léon Rothkrantz, Cees Witteveen, Michel Verhaegen, and of course, again, Damien, Robert, and Bart.

I also wish to thank my family for their support. Finally, since it is not possible to mention here each and every person who contributed to my work in one way or another, I apologize to these persons for not including them, and thank them all together.

Lucian Buşoniu
Delft, November 2008

Contents

Chapter 1

Introduction

This chapter introduces dynamic programming and reinforcement learning, and the main challenge in this field: the need for approximating the solution. This challenge motivates the research presented in this thesis. Then, the contributions of the thesis are described. The chapter closes with an outline of the thesis.

1.1 Dynamic programming and reinforcement learning

The framework of dynamic programming (DP) and reinforcement learning (RL) can be used to address important problems arising in a variety of fields, including e.g., automatic control, operations research, computer science, and economy. This thesis focuses on DP and RL for automatic control (Bertsekas, 2007). From the perspective of automatic control, the DP/RL framework comprises a general, nonlinear and stochastic optimal control problem. Algorithms that solve such a problem in general are extremely useful for optimal control. Moreover, RL algorithms solve the problem using only data, without requiring prior knowledge about the process.

The application of DP and RL to artificial intelligence is equally important, although it is not the focus of this thesis. From the perspective of artificial intelligence, RL promises a methodology to build an artificial agent that learns how to survive and optimize its behavior in an unknown environment, without requiring any prior knowledge (Sutton and Barto, 1998). Building such an agent is an important goal of artificial intelligence.

The DP/RL field combines research from optimal control theory and from artificial intelligence, and this mixed origin is reflected by the use of two sets of names for identical concepts in the field: e.g., 'controller' has the same meaning as 'agent', and 'process' has the same meaning as 'environment'. In this thesis, control-theoretical terminology will be used. At each discrete time step, the controller measures the state of the process and applies an action, according to a control policy. As a result of this action, the process transits into a new state, and a scalar reward signal is sent to the controller to indicate the quality of the transition. The controller measures the new state, and the whole cycle repeats. State transitions can in general be nonlinear and stochastic. This pattern of interaction is represented in Figure 1.1. The goal is to maximize the cumulative reward (the return) over the course of interaction (Bertsekas, 2007; Sutton and Barto, 1998; Bertsekas and Tsitsiklis, 1996).

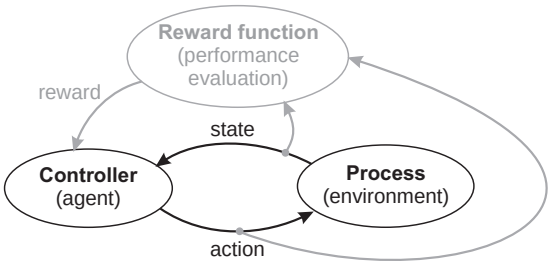


Figure 1.1: The basic elements of DP and RL, and their flow of interaction. The elements related to the reward are depicted in gray.

As a conceptual example, consider a garbage-collecting robot. This robot measures its own position, and the positions of the surrounding objects; these positions are the state variables. The software of the robot is the controller. It receives the position measurements, and sends commands to the wheel motors; these commands are the actions. The dynamics describe the rule according to which states (positions) change as a result of the actions. The reward signal could, e.g., be positive at every time step in which the robot picks up trash, and zero otherwise. Note that in DP and RL, the process (environment) also includes the physical body of the robot.

Solving the DP/RL problem amounts to finding an optimal policy, i.e., a policy that maximizes the return. To conveniently compute an optimal policy, many DP/RL algorithms use value functions, which give the returns from every state (we call this type of value function a V-function in this thesis) or from every state-action pair (Q-function). Three categories of algorithms can be identified by the path they take to search for an optimal policy. The first category includes *value iteration* algorithms, which search for the optimal value function and then use it to compute an optimal policy. The second category includes *policy iteration* algorithms, which iteratively improve policies. In each iteration, the value function of the current policy is found (instead of the optimal value function), and is then used to computed a new, improved policy. Policy iteration converges to an optimal policy, the value function of which is by definition optimal. The third category includes *direct policy search* algorithms, which search for an optimal policy with optimization techniques, without using a value function.

DP algorithms require a model of the process dynamics and the reward function. Note that the name 'DP' traditionally refers to algorithms that aim to compute the optimal value function, either explicitly (like value iteration) or implicitly (like policy iteration) (Bertsekas and Tsitsiklis, 1996). In this thesis, we use the name 'DP' to refer to the larger class of all model-based algorithms that solve the DP/RL problem, including direct policy search. Besides the three categories of algorithms described above, a fourth category of (model-based) algorithms is *model-predictive control* (Bertsekas, 2005). Model-predictive control works online, by computing at every time step an open-loop policy, and performing the action indicated by this policy in the current state. The overall resulting policy is time-varying. Because this thesis does not employ model-predictive control algorithms, this category of techniques will not be considered in the sequel.

Unlike DP techniques, RL algorithms do not require a model. Instead, they use data obtained from the process or from a simulation model of the process. This data can be a set

of samples, a set of process trajectories, or a single process trajectory. So, RL can be seen as model-free, sample-based or trajectory-based DP, and DP can be seen as model-based RL. Some RL algorithms work offline, using data collected in advance. However, most RL algorithms work online, i.e., they compute a solution while also controlling the process.

The main challenge in DP and RL stems from the fact that the classical algorithms for DP/RL can only be implemented when the state space consists of a finite (and not too large) number of elements. This is because these algorithms require to store and update exact representations of value functions and/or policies. Exactly representing state-action value functions additionally requires that the action space consists of a finite (and not too large) number of elements. Unfortunately, very few problems satisfy these requirements. For instance, the majority of the control problems have continuous state and action variables, which means that the state and action spaces have infinitely many elements. For such problems, versions of the classical algorithms that represent value functions and/or policies *approximately* have to be used (Bertsekas and Tsitsiklis, 1996). Even for finite state-action spaces, the cost of storing and computing value functions and policies grows exponentially with the number of state variables, and approximate representations are needed when this number is large. The field of *approximate DP and RL* comprises the development and study of algorithms that use approximate representations of the value functions and/or policies. Value iteration, policy iteration, and policy search can all be modified to use approximation.

The need for approximation is not the only challenge of DP/RL. For instance, it is currently still unclear how to represent certain important high-level control goals using the reward function, and also how to guarantee a predictable performance improvement during online learning. This thesis, however, will focus on approximate DP and RL, as detailed in the next section.

1.2 Focus and contributions of the thesis

This thesis aims to develop effective techniques for DP and RL in control problems. Since an overwhelming majority of the control problems have continuous state and action variables, approximate representations of the value functions and policies are required. The goal of the techniques developed is to find an (approximately) optimal, stationary policy, which (approximately) maximizes the infinite-horizon discounted sum of rewards. All the value function and policy representations employed in this thesis rely on basis functions (BFs) defined over the state space or over the state-action space.

A novel technique is developed and studied for each of the three categories of approximate DP/RL: approximate value iteration, policy iteration, and policy search. The development of each new technique is motivated by an important open issue in its category. This section outlines these three novel techniques. First, fuzzy Q-iteration, an algorithm for approximate value iteration, is described. Fuzzy Q-iteration is model-based and represents the Q-function using a linear combination of the BFs. Next, online least-squares, approximate policy iteration is presented. This algorithm is model-free and, like fuzzy Q-iteration, represents Q-functions using linear combinations of BFs. Finally, we describe cross-entropy policy search, our contribution to the field of approximate policy search. In cross-entropy policy search, the policy is represented by associating state-dependent BFs to discrete actions. Whereas fuzzy Q-iteration and online least-squares policy iteration employ fixed BFs, cross-entropy policy search optimizes the BFs.

Fuzzy Q-iteration

Fuzzy approximation is a promising way to represent the value function for *approximate value iteration.* It has often been used for approximate RL, but only in a heuristic fashion and without convergence guarantees. Therefore, as a first major contribution of this thesis, Chapter 4 develops a provably convergent DP algorithm with fuzzy approximation. This *fuzzy Q-iteration* algorithm represents state-action value functions (Q-functions) using a fuzzy partition of the state space and a discretization of the action space. Each fuzzy set in the partition is described by a membership function, which can be identified with a BF. The resulting Q-function representation is linear in the parameters, so that existing convergence guarantees for DP with linearly parameterized approximators can be used as a starting point. These guarantees are extended to account for the action discretization, and subsequently used to show that fuzzy Q-iteration asymptotically converges to a Q-function that lies within a bounded distance from the optimal Q-function. A version of fuzzy Q-iteration where parameters are updated in an asynchronous fashion converges at least as fast as the synchronous variant. A bound on the suboptimality of the solution obtained in a finite number of iterations is derived. Under continuity assumptions on the dynamics and on the reward function, we also prove that the algorithm is consistent, i.e., that the optimal Q-function is asymptotically obtained as the approximation accuracy increases. As an alternative to designing the membership functions in advance, we propose an algorithm to optimize them using the cross-entropy method. To evaluate each configuration of membership functions, a policy is computed with fuzzy Q-iteration using these membership functions, and the performance of this policy is evaluated using simulation.

An extensive numerical and experimental evaluation is conducted to assess the performance of fuzzy Q-iteration in practice. We study the influence of the number of fuzzy sets and of discrete actions on the performance, for a continuous as well as a discontinuous reward function. Such studies are not typically reported in the literature, where reward functions are usually discontinuous and the action discretization is fixed.

Online least-squares policy iteration

Least-squares methods for policy evaluation are sample-efficient and have relaxed convergence requirements. They are used to approximate the value functions of policies in certain *approximate policy iteration* algorithms. Among these algorithms, least-squares policy iteration (LSPI) is especially attractive because it relies on Q-functions, which makes policy improvements easier than when V-functions are used. Like in fuzzy Q-iteration, the Q-functions are represented using a linear combination of BFs. An important limitation of LSPI is that it only works offline: before every policy improvement, a batch of samples has to be processed to accurately approximate the value function of the current policy. As a second major contribution of this thesis, Chapter 5 extends LSPI to the more challenging setting of online learning: *online LSPI* is obtained. The main difference from the offline algorithm is that the performance of each intermediate policy is important, which means that policy improvements have to be performed at short intervals. In these intervals, only a small number of samples can be processed. Unlike offline LSPI, which can use samples drawn from an arbitrary distribution over the state-action space, the online algorithm only uses samples collected along trajectories of the process. This also implies that online LSPI has to actively explore in order to collect informative samples. A thorough numerical and experimental study is conducted

to assess the performance of online LSPI.

While RL (model-free) algorithms are typically designed to work in the absence of prior knowledge, a certain amount of prior knowledge is almost always available in practice. This prior knowledge can be used to speed up learning. We describe a method to use prior knowledge about the monotonicity of the policy in the state variables, in order to speed up online LSPI. Monotonous policies are suitable for controlling e.g., (nearly) linear systems, or systems that are (nearly) linear and have monotonous input nonlinearities, such as saturation or dead-zone nonlinearities. We also investigate polynomial, continuous-action approximation for LSPI. Continuous actions are useful in important classes of control problems. For instance, when a system has to be stabilized around an unstable equilibrium, any discrete-action policy leads to chattering of the control action and to undesirable limit cycles.

Cross-entropy policy search

In the literature on *approximate policy search*, typically ad-hoc policy parameterizations are designed for specific problems, using intuition and prior knowledge. For instance, if a process needs to be stabilized around an equilibrium, and is approximately linear in the operating range of the controller, then a policy parameterization that is linear in the state variables can be used. Unfortunately, such specific parameterizations cannot be used to solve general problems. As a third major contribution of this thesis, Chapter 6 develops and applies highly flexible policy approximators for direct policy search. These approximators can be used to solve a large class of problems. A discretized action space is used, and the policies are represented using state-dependent BF, where a discrete action is assigned to each BF. The type of BFs and their number are specified in advance and determine the approximation power of the policy representation. The locations and shapes of the BFs, together with the action assignments, are the parameters subject to optimization — unlike in fuzzy Q-iteration and LSPI, where the BFs are fixed. The optimization criterion is a weighted sum of the returns from a set of representative initial states, where each return is estimated with Monte Carlo simulations. The representative states together with the weight function can be used to focus the algorithm on important parts of the state space. This is in contrast to other policy search techniques, which attempt to optimize the return over the entire state space.

Because of the rich policy parameterization, the performance may be a complicated function of the parameter vector, e.g., non-convex, non-differentiable, with many local optima. This means that a global optimization technique has to be used. We select the cross-entropy method for optimization, and obtain *cross-entropy policy search*. Numerical evaluations of cross-entropy policy search are conducted on a double-integrator example with two state variables, as well as on two more involved problems: bicycle balancing (with four state variables) and the simulated treatment of infection with the human immunodeficiency virus (with six state variables).

1.3 Thesis outline

Figure 1.2 presents a graphical roadmap depicting the organization of this thesis. Chapters 2 and 3 introduce the background and notation necessary to understand the remainder of the thesis. In particular, Chapter 2 describes the DP and RL problem, presents some representative classical algorithms that solve this problem, and illustrates how these algorithms work

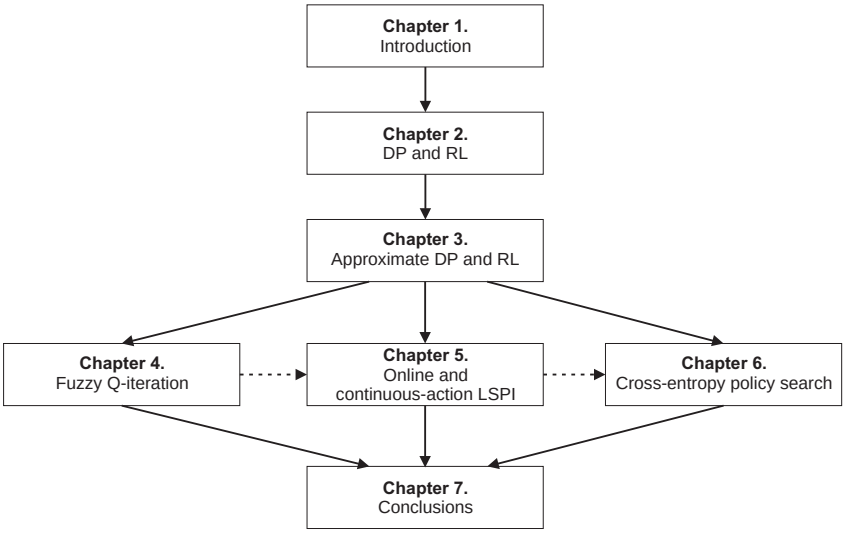


Figure 1.2: A roadmap of the thesis. The source chapter of an arrow should be read before the destination chapter. Dashed arrows indicate optional ordering.

using a simple, garbage-collecting robot example with discrete states and actions. Chapter 3 contributes a survey of approximate DP and RL. Algorithms for approximate value iteration, policy iteration, and policy search, are reviewed in turn. An example involving servo-system stabilization is used to illustrate the behavior of a representative algorithm from each category. Convergence results are given for approximate value iteration and approximate policy iteration, and techniques to automatically derive value function approximators are reviewed.

The next three chapters present in detail the contributions of this thesis. Chapter 4 introduces and analyzes the fuzzy Q-iteration algorithm. Chapter 5 presents online LSPI and LSPI with polynomial action approximation. Chapter 6 describes cross-entropy policy search. These three chapters evaluate the respective algorithms using extensive numerical experiments. Fuzzy Q-iteration and online LSPI are also applied to the real-time control of an inverted pendulum. Chapters 4, 5, and 6 can be read in any order, although, if possible, they should be read in sequence.

Chapter 7 closes the thesis with some concluding remarks, and identifies opportunities for future research.

Two appendices are included in this thesis. Appendix A introduces the cross-entropy method for rare-event simulation and optimization. It is useful (but not required) to read this appendix before Chapters 4 and 6, which use the cross-entropy method after describing it only briefly. Appendix B derives an algorithm for least-squares policy evaluation for Q-functions. Least-squares policy evaluation can be used to derive an online policy iteration algorithm similar to online LSPI.

Chapter 2

Dynamic programming and reinforcement learning

This chapter introduces Markov decision processes, and the two main approaches to solve them: dynamic programming and reinforcement learning. Both stochastic and deterministic Markov decision processes are discussed, and their optimal solution is characterized. Three categories of algorithms are described: value iteration, policy iteration, and direct search for control policies. In closing, the main open issues of dynamic programming and reinforcement learning are discussed.

2.1 Introduction

Markov decision processes (MDPs) can be used to formalize important problems arising in a variety of fields, among which automatic control, operations research, computer science, and economy. In an MDP, at each discrete time step, the controller (agent, decision maker) measures the state of the process (environment), and applies an action, according to a control policy. As a result of this action, the process transits into a new state, and a scalar reward signal is generated that indicates the quality of the transition. The controller measures the new state, and the whole cycle repeats (see also Figure 1.1). State transitions can in general be nonlinear and stochastic. The goal is to find an optimal policy, which maximizes the cumulative reward (the return) over the course of interaction. Therefore, solving an MDP amounts to finding an optimal policy.

In control theory, the terms ‘controller’ and ‘process’ are used, instead of ‘agent’ and ‘environment’; the latter are preferred in the field of artificial intelligence. In this thesis, the control-theoretical terminology and notation will be used. The two sets of names for identical concepts reflect the mixed origin of the field, which represents the convergence of research in optimal control theory (Bertsekas and Tsitsiklis, 1996; Bertsekas, 2007) and artificial intelligence (Sutton and Barto, 1998). The term ‘reinforcement’ originates in psychology.

The most important criterion to classify the techniques that solve MDPs is whether they require a model of the MDP or not. A model includes the transition dynamics and the reward function. In this thesis, we use the name ‘DP’ to refer to the class of all model-based algorithms that solve MDPs, including policy search. This class is larger than the category

of algorithms traditionally called 'dynamic programming (DP)'; the latter only includes algorithms that aim to compute the optimal value function (Bertsekas, 2007; Bertsekas and Tsitsiklis, 1996). DP algorithms typically work offline, producing a policy which is then used to control the process. Usually, analytical expressions for the dynamics and the reward function are not required. Instead, given a state and an action, the model is only required to generate (deterministically or stochastically) a next state and the corresponding reward. It is often easier to design an algorithm that generates state-reward pairs in this fashion, than to derive analytical expressions for the transition dynamics.

Model-free algorithms that solve MDPs are collectively known as reinforcement learning (RL) (Sutton and Barto, 1998). RL is useful when a mathematical model of the process is difficult or costly to derive. RL algorithms use data obtained from the process. This data can be a set of samples, a set of process trajectories, or a single process trajectory. RL algorithms either use no explicit model at all, or build a model from the data; we call this latter type of algorithms 'model-learning'. So, RL can be seen as model-free, sample-based or trajectory-based DP, and DP can be seen as model-based RL. RL algorithms have less control over the data they use than DP algorithms (DP has full control over the data, because it can use the model to sample the state-action space at arbitrary locations, an arbitrary number of times). This makes RL more challenging than DP.

Some RL algorithms work offline, using data collected in advance. However, most RL algorithms work online, i.e., they compute a solution simultaneously with controlling the process. Online learning is useful when it is difficult or costly to obtain data in advance. In the online case, RL algorithms must balance the need to collect informative data with the need to control the process well. This makes online RL more challenging than offline RL. Although online RL algorithms are theoretically sound only when the process does not change over time, in practice they may also be able to deal with processes that are slowly changing, by adapting the solution to take these changes into account.

Section 2.2 describes the MDP formalism and characterizes the optimal solution of an MDP, in the deterministic as well as the stochastic setting. DP and RL techniques can be classified further in three categories, according to the path they take to look for an optimal policy. The first category includes *value iteration* algorithms, which search for the optimal value function. The optimal value function consists of the maximum returns from every state, or from every state-action pair. After finding the optimal value function, value iteration algorithms use it to compute an optimal policy. Value iteration algorithms are discussed in Section 2.3. The second category includes *policy iteration* algorithms, which iteratively improve policies. In each iteration, the value function of the current policy is found (instead of the optimal value function), and is then used to computed a new, improved policy. Policy iteration algorithms are discussed in Section 2.4. The third category includes *direct policy search* algorithms, which search for an optimal policy with optimization techniques, without using a value function at all; they are discussed in Section 2.5. Section 2.6 concludes the chapter and discusses the most important open issues in DP/RL.

The theoretical background presented in this chapter is loosely based on the books of Bertsekas (2007) and Sutton and Barto (1998).

2.2 Markov decision processes

In this section, MDP are formally described and their optimal solution is characterized. First, deterministic MDPs are introduced, followed by their extension to the stochastic case.

2.2.1 Deterministic setting

A deterministic MDP consists of the following elements: a state space X, an action space U, a transition function $f : X \times U \to X$, and a reward function $\rho : X \times U \to \mathbb{R}$.[1] As a result of the control action u_k applied in the state x_k at the discrete time step k, the state changes to $x_{k+1} = f(x_k, u_k)$. At the same time, the controller receives the scalar reward signal $r_{k+1} = \rho(x_k, u_k)$, which evaluates the immediate effect of action u_k (the transition from x_k to x_{k+1}), but in general does not say anything about its long-term effects.

Given f and ρ, the current state x_k and action u_k are sufficient to precisely determine both the next state x_{k+1} and the reward r_{k+1}. This is the *Markov property*, which is essential in providing theoretical guarantees about DP/RL algorithms.

The controller chooses actions according to its policy $h : X \to U$, using $u_k = h(x_k)$. The goal is to find an optimal policy, i.e., a policy that maximizes from any initial state x_0, the infinite-horizon discounted return:

$$R^h(x_0) = \sum_{k=0}^{\infty} \gamma^k r_{k+1} = \sum_{k=0}^{\infty} \gamma^k \rho(x_k, h(x_k)) \tag{2.1}$$

where $\gamma \in [0, 1)$ is the discount factor and $x_{k+1} = f(x_k, h(x_k))$ for $k \geq 0$. The infinite-horizon discounted return compactly represents the reward accumulated by the controller in the long run. The task is therefore to maximize the long-term performance, while only using feedback about the immediate, one-step performance. Discounting ensures that, given bounded rewards, the returns will always be bounded. The discount factor can be interpreted intuitively as a measure of how 'far-sighted' the controller is in considering its rewards, or as a way to take into account an increasing uncertainty about rewards that will be received in the future.

Instead of discounting the rewards, they can also be averaged over time, or they can simply be added together without weighting (Kaelbling et al., 1996). Furthermore, it is also possible to use a finite-horizon discounted return:

$$\sum_{k=0}^{K} \gamma^k r_{k+1}$$

In the finite-horizon case, optimal policies and the optimal value function depend in general on the time step k. In contrast, in the infinite-horizon case, optimal policies and the optimal value function are stationary, i.e., they do not depend on time. Only *infinite-horizon discounted* returns, leading to stationary optimal policies and value functions, will be considered in this thesis.

[1] The standard control-theoretical notation is preferred in this thesis to the artificial intelligence notation typically used in reinforcement learning. For instance, the state is denoted by x, the state space by X, the control action by u, the action space by U, and the process dynamics by f. Furthermore, we denote reward functions by ρ, to distinguish them from the instantaneous rewards r and from the returns R. Control policies are denoted by h.

Example 2.1 The cleaning robot. Consider the deterministic, discrete problem depicted in Figure 2.1: a cleaning robot has to collect a used can and also has to recharge its batteries.

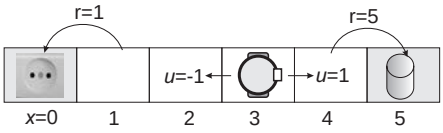


Figure 2.1: The cleaning robot problem.

The state space is discrete and contains six distinct states, denoted by the integer numbers 0 to 5: $X = \{0, 1, 2, 3, 4, 5\}$. The robot can move to the left ($u = -1$) or to the right ($u = 1$); the action space is therefore $U = \{-1, 1\}$. States 0 and 5 are terminal, which means that once the robot reaches either of them it can no longer leave that state, regardless of the action. The corresponding transition function is:

$$f(x,u) = \begin{cases} x+u & \text{if } \ 1 \le x \le 4 \\ x & \text{if } \ x = 0 \ \text{ or } \ x = 5 \ \ \text{(regardless of } u\text{)} \end{cases}$$

In state 5, the robot finds a can and the transition into this state is rewarded with 5. In state 0, the robot can recharge its batteries and the transition into this state is rewarded with 1. All other rewards are 0. In particular, taking any action while in a terminal state results in a reward of 0, which means that the robot will not accumulate (undeserved) rewards in the terminal states. The corresponding reward function is:

$$\rho(x,u) = \begin{cases} 5 & \text{if } x = 4 \ \text{ and } \ u = 1 \\ 1 & \text{if } x = 1 \ \text{ and } \ u = -1 \\ 0 & \text{otherwise} \end{cases}$$

□

While the discount factor γ can be theoretically regarded as a given part of the problem, in practice, a good value of γ has to be chosen. Typically, choosing γ involves a tradeoff between the quality of the solution and the convergence rate of the DP/RL algorithm. Algorithms tend to converge faster when γ is smaller, but if γ is too small, the solution may be unsatisfactory because it does not sufficiently take into account rewards that are obtained after a large number of steps. Unfortunately, there is no generally valid procedure for choosing γ. Consider however, as an example, a typical stabilization control problem, where from every initial state the process is expected to reach a steady state and remain there. In such a problem, γ should be chosen large enough so that the rewards received upon reaching the steady state have a measurable influence on the returns from every initial state. This means that if the maximum number of steps taken by a reasonably good policy to reach the steady-state from any initial state is K, then γ should be chosen so that γ^K is not too small. Finding K is a difficult problem in itself. It could be determined, e.g., by using a suboptimal policy obtained by other means.

A convenient way to characterize policies is using their value function. Two types of value functions exist: state-action value functions (for short, Q-functions) and state value functions (for short, V-functions). Note that usually, the name 'value function' is used for the

V-function in the literature; however, we will use the names 'Q-function' and 'V-function' to clearly differentiate between the two types of value functions. We use the name 'value function' to refer to Q-functions and V-functions collectively.

First, Q-functions are defined and characterized, followed by V-functions. The Q-function $Q^h : X \times U \to \mathbb{R}$ gives the return obtained when starting from a given state, applying a given action, and following the policy h thereafter:

$$Q^h(x, u) = \rho(x, u) + \gamma R^h(f(x, u)) \tag{2.2}$$

Here, $R^h(f(x, u))$ is the return from the next state $f(x, u)$, which can be computed with (2.1). The optimal Q-function is defined as $Q^*(x, u) = \max_h Q^h(x, u)$. Any policy that selects for every state an action with the highest optimal Q-value:

$$h^*(x) = \arg\max_u Q^*(x, u) \tag{2.3}$$

is optimal (it maximizes the return). A policy that maximizes a Q-function in this way is said to be *greedy* in that Q-function. So, finding an optimal policy can be done by first finding Q^*, and then computing the greedy policy in Q^* according to (2.3).

A central result in DP and RL is the *Bellman optimality equation*:

$$Q^*(x, u) = \rho(x, u) + \gamma \max_{u'} Q^*(f(x, u), u') \tag{2.4}$$

This equation states that the optimal value of action u taken in state x is the immediate reward plus the discounted optimal value that can be obtained from the next state. The Bellman optimality equation can be used to iteratively compute the optimal Q-function, as explained later in Section 2.3.

The V-function $V^h : X \to \mathbb{R}$ gives the return when starting from a particular state and following policy h. It can computed using the Q-function:

$$V^h(x) = Q^h(x, h(x)) \tag{2.5}$$

The optimal V-function is the V-function of any optimal policy, and can be defined as:

$$V^*(x) = \max_h V^h(x) = \max_u Q^*(x, u) \tag{2.6}$$

The optimal V-function satisfies the following variant of the Bellman optimality equation:

$$V^*(x) = \max_u [\rho(x, u) + \gamma V^*(f(x, u))] \tag{2.7}$$

An optimal policy can be computed from V^* using:

$$h^*(x) = \arg\max_u [\rho(x, u) + \gamma V^*(f(x, u))] \tag{2.8}$$

This formula is more complex than (2.3); in particular, a model of the MDP is required in the form of the reward function ρ and the dynamics f. For this reason, Q-functions will be preferred to V-functions throughout this thesis. The disadvantage of using Q-functions is that they are more costly to represent than V-functions, because in addition to x they also depend on u.

2.2.2 Stochastic setting

In a stochastic MDP, the deterministic transition function f is replaced by a transition probability function $\tilde{f} : X \times U \times X \to [0, \infty)$. The probability that the next state x_{k+1} belongs to a region $X_{k+1} \subset X$ of the state space after action u_k is taken in state x_k is $\int_{X_{k+1}} \tilde{f}(x_k, u_k, x')\mathrm{d}x'$.[2] For any x and u, $\tilde{f}(x, u, \cdot)$ is assumed to define a valid probability density function of the argument '$\cdot$', which stands for the random variable x_{k+1}. Because rewards are associated with transitions, and the transitions are no longer fully determined by the current state and action, the reward function also has to depend on the next state, $\tilde{\rho} : X \times U \times X \to \mathbb{R}$. After each transition to a state x_{k+1}, a reward $r_{k+1} = \tilde{\rho}(x_k, u_k, x_{k+1})$ is received. In the stochastic case, the Markov property requires that x_k and u_k precisely determine the probability density function (the statistics) of the next state, and thereby the statistics of the next received reward. As in the deterministic case, the Markov property is essential to providing theoretical guarantees about the DP/RL algorithms.

The *expected* infinite-horizon discounted return of an initial state x_0 under a (deterministic) policy h is:

$$\begin{aligned} R^h(x_0) &= \lim_{K\to\infty} \mathrm{E}_{x_{k+1}\sim\tilde{f}(x_k,h(x_k),\cdot)} \left\{ \sum_{k=0}^{K} \gamma^k r_{k+1} \right\} \\ &= \lim_{K\to\infty} \mathrm{E}_{x_{k+1}\sim\tilde{f}(x_k,h(x_k),\cdot)} \left\{ \sum_{k=0}^{K} \gamma^k \tilde{\rho}(x_k, h(x_k), x_{k+1}) \right\} \end{aligned} \tag{2.9}$$

where the notation $x_{k+1} \sim \tilde{f}(x_k, h(x_k), \cdot)$ means that the random variable x_{k+1} is drawn from the density $\tilde{f}(x_k, h(x_k), \cdot)$ at each step k. The discussion of Section 2.2.1 regarding the choice of γ also applies to the stochastic case. For any stochastic or deterministic MDP, when using the infinite-horizon discounted return, there exists at least one deterministic optimal policy (Bertsekas, 2007). Therefore, only *deterministic policies* will be considered in the sequel.

The Q-function of a policy h is the *expected* return under the stochastic transitions, when starting in a particular state, applying a particular action, and following the policy h thereafter:

$$Q^h(x, u) = \mathrm{E}_{x'\sim\tilde{f}(x,u,\cdot)} \left\{ \tilde{\rho}(x, u, x') + \gamma R^h(x') \right\} \tag{2.10}$$

where $x' \sim \tilde{f}(x, u, \cdot)$ means that x' is drawn from the density $\tilde{f}(x, u, \cdot)$. The definition for Q^* remains unchanged. The expectation enters the Bellman optimality equation for Q^* as follows:

$$Q^*(x, u) = \mathrm{E}_{x'\sim\tilde{f}(x,u,\cdot)} \left\{ \tilde{\rho}(x, u, x') + \gamma \max_{u'} Q^*(x', u') \right\} \tag{2.11}$$

An optimal policy can still be computed from Q^* as in the deterministic case, with (2.3). Value functions can still be computed from Q-functions using (2.5) and (2.6). However, the Bellman optimality equation for V^* changes to:

$$V^*(x) = \max_u \mathrm{E}_{x'\sim\tilde{f}(x,u,\cdot)} \left\{ \tilde{\rho}(x, u, x') + \gamma V^*(x') \right\} \tag{2.12}$$

[2] It is not useful to characterize the probability of ending up in a given state x_{k+1}, because in general this probability is 0.

and the computation of the optimal policy from V^* becomes even more complicated, involving an expectation that did not appear in the deterministic case:

$$h^*(x) = \arg\max_u \mathrm{E}_{x' \sim \tilde{f}(x,u,\cdot)} \{\tilde{\rho}(x,u,x') + \gamma V^*(x')\} \tag{2.13}$$

Since computing the optimal policy from Q^* is as simple as in the deterministic case, this is yet another reason for using Q-functions in practice.

Clearly, all the equations for deterministic MDPs are special cases of the equations for stochastic MDPs. The deterministic case is obtained by using a degenerate density $\tilde{f}(x,u,\cdot)$ that assigns all the probability mass to $f(x,u)$. The deterministic reward function is $\rho(x,u) = \tilde{\rho}(x,u,f(x,u))$.

2.3 Value iteration

Value iteration techniques use the Bellman optimality equation to iteratively compute an optimal value function, from which an optimal policy is derived. DP (model-based) algorithms like V-iteration (Bertsekas, 2007) solve the Bellman optimality equation by using knowledge of the transition and reward functions. RL (model-free) techniques like Q-learning (Watkins and Dayan, 1992), SARSA (Rummery and Niranjan, 1994), and Dyna (Sutton, 1990) either learn a model, or do not use an explicit model at all.

Next, the model-based Q-iteration algorithm is introduced. Let the set of all the Q-functions be denoted by $\mathcal{Q}$. Define the *Q-iteration mapping* $T : \mathcal{Q} \to \mathcal{Q}$, which computes the right-hand side of the Bellman optimality equation (2.4) for any Q-function:[3]

$$[T(Q)](x,u) = \rho(x,u) + \gamma \max_{u'} Q(f(x,u),u') \tag{2.14}$$

In the stochastic case, simply use the right-hand side of the stochastic Bellman optimality equation (2.11), instead of (2.4):

$$[T(Q)](x,u) = \mathrm{E}_{x' \sim \tilde{f}(x,u,\cdot)} \left\{ \tilde{\rho}(x,u,x') + \gamma \max_{u'} Q(x',u') \right\} \tag{2.15}$$

The same notation is used for the Q-iteration mapping in both the deterministic and the stochastic case, because all the statements below hold in both cases; additionally, the definition (2.14) of T is a special case of (2.15).

Using the Q-iteration mapping, the Bellman optimality equation states that Q^* is a fixed point of T, i.e., $Q^* = T(Q^*)$. It can be shown that T is a contraction with factor $\gamma < 1$ in the infinity norm, i.e., for any pair of functions Q and Q', it is true that $\|T(Q) - T(Q')\|_\infty \leq \gamma \|Q - Q'\|_\infty$. This can easily be shown by extending the proof that the mapping which computes V-functions for V-iteration (analogous to T), is a contraction (Bertsekas, 2007).

The Q-iteration algorithm starts from an arbitrary Q-function Q_0 and in each iteration ℓ updates the Q-function using:

$$Q_{\ell+1} = T(Q_\ell) \tag{2.16}$$

[3]The term 'mapping' is used to refer to functions that work with other functions as inputs and/or outputs; as well as to compositions of such functions. The term is used to differentiate mappings from ordinary functions, which only work with numerical scalars, vectors, or matrices.

Because T is a contraction, it has a unique fixed point. From (2.4), this point is Q^*, so Q-iteration asymptotically converges to Q^* as $\ell \to \infty$.

Next, the computational cost of Q-iteration is investigated for the case of a deterministic MDP with a finite number of states and actions. Denote by $|\cdot|$ the cardinality of the argument set '$\cdot$', so that $|X|$ denotes the finite number of states and $|U|$ denotes the finite number of actions. In every iteration, the Q-values of a number of $|X|\,|U|$ state-action pairs have to be updated. Assume that the maximization over the action space U in (2.14) is solved by enumeration over its $|U|$ elements. Assume also that $f(x,u)$ and $\rho(x,u)$ are evaluated once for every (x,u) pair and then cached and reused. Then, every iteration ℓ requires $|X|\,|U|\,(2+|U|)$ function evaluations, where the functions being evaluated are f, ρ, and the current Q-function Q_ℓ. Furthermore, a finite number L of iterations can be (conservatively) chosen so that the suboptimality of the greedy policy in Q_L is guaranteed to be at most $\varepsilon_{\text{QI}} > 0$, using:

$$L = \left\lceil \log_\gamma \frac{\varepsilon_{\text{QI}}(1-\gamma)^2}{2\|\rho\|_\infty} \right\rceil \tag{2.17}$$

Here, $\|\rho\|_\infty = \max_{x,u} |\rho(x,u)|$ is assumed to be finite and $\lceil\cdot\rceil$ gives the smallest integer larger than or equal to the argument (ceiling).[4] Therefore, the total cost of Q-iteration for a deterministic, finite MDP is:

$$L\,|X|\,|U|\,(2+|U|) \tag{2.18}$$

A similar, V-iteration algorithm can be given to compute the optimal V-function V^*, using the Bellman optimality equation (2.7), or (2.12) in the stochastic case. Note that usually, the name 'value iteration' is used for the V-iteration algorithm in the literature; however, we will use the names 'Q-iteration' and 'V-iteration' to clearly differentiate between the two algorithms. We use the term value iteration to refer collectively to algorithms that compute optimal value functions (model-free or model-based, offline or online).

From the class of (model-free) RL algorithms, Q-learning will be described next. Q-learning starts from an arbitrary initial Q-function Q_0 and updates it without requiring a model, using instead observed transitions $(x_k, u_k, x_{k+1}, r_{k+1})$ (Watkins, 1989; Watkins and Dayan, 1992). After each such transition, the Q-function is updated using:

$$Q_{k+1}(x_k,u_k) = Q_k(x_k,u_k) + \alpha_k[r_{k+1} + \gamma \max_{u'} Q_k(x_{k+1},u') - Q_k(x_k,u_k)] \tag{2.19}$$

where $\alpha_k \in (0,1]$ is the learning rate. The term between square brackets is the temporal difference, i.e., the difference between the current estimate $Q_k(x_k,u_k)$ of the optimal Q-value of (x_k,u_k) and the updated estimate $r_{k+1} + \gamma \max_{u'} Q_k(x_{k+1},u')$. This new estimate is actually the Q-iteration operator (2.14) applied to Q_k in the state-action pair (x_k,u_k), where $\rho(x_k,u_k)$ has been replaced by the observed reward r_{k+1}, and $f(x_k,u_k)$ by the observed next state x_{k+1}. In the deterministic case, these replacements provide an exact value of the Q-iteration mapping (2.14); in the stochastic case, they provide a single sample of the expectation in (2.15).

Q-learning asymptotically converges to Q^* as $k \to \infty$, under the following conditions (Watkins and Dayan, 1992; Jaakkola et al., 1994):

[4] Equation (2.17) follows from the bound $\|V^{h_L} - V^*\|_\infty \le 2\frac{\gamma^L\|\rho\|_\infty}{(1-\gamma)^2}$ on the suboptimality $\|V^{h_L} - V^*\|_\infty$ of the greedy policy $h_L(x) = \arg\max_u Q_L(x,u)$, by requiring that $2\frac{\gamma^L\|\rho\|_\infty}{(1-\gamma)^2} \le \varepsilon_{\text{QI}}$ (Ernst et al., 2005).

- The sum $\sum_{k=0}^{\infty} \alpha_k^2$ is finite, whereas the sum $\sum_{k=0}^{\infty} \alpha_k$ is infinite.

- All the state-action pairs are visited infinitely often as $k \to \infty$.

The second condition can be satisfied if, among other things, the controller has a non-zero probability of selecting any action in every encountered state; this is called exploration. Since the Q-learning algorithm does not specify how actions are chosen, the controller is free to explore. Exploration is usually achieved by selecting actions randomly, using a distribution that assigns a non-zero probability to each action at every time step. However, the controller cannot always choose fully random actions, because the control performance is also important in online learning. Sometimes greedy actions, which maximize the Q-values of the current state, have to be selected; this is called exploitation (recall that any optimal policy is greedy in the optimal Q-function). Usually, a diminishing exploration schedule is used, so that asymptotically, as $Q_k \to Q^*$, the policy used also converges to the (exploiting) greedy, and therefore optimal, policy.

Example 2.2 Q-iteration for the cleaning robot. Consider again the cleaning robot problem of Example 2.1. Take a discount factor $\gamma = 0.5$. Starting from an identically zero initial Q-function, $Q_0 = 0$, Q-iteration produces the sequence of Q-functions given in Table 2.1, where each cell shows the Q-values of the two actions in a particular state, separated by a semicolon. For instance:

$$Q_3(2,1) = \rho(2,1) + \gamma \max_u Q_2(f(2,1),u) = 0 + 0.5 \max_u Q_2(3,u) = 0 + 0.5 \cdot 2.5 = 1.25$$

Table 2.1: Q-iteration results for the cleaning robot

	$x=0$	$x=1$	$x=2$	$x=3$	$x=4$	$x=5$
Q_0	0 ; 0	0 ; 0	0 ; 0	0 ; 0	0 ; 0	0 ; 0
Q_1	0 ; 0	1 ; 0	0.5 ; 0	0.25 ; 0	0.125 ; 5	0 ; 0
Q_2	0 ; 0	1 ; 0.25	0.5 ; 0.125	0.25 ; 2.5	1.25 ; 5	0 ; 0
Q_3	0 ; 0	1 ; 0.25	0.5 ; 1.25	0.625 ; 2.5	1.25 ; 5	0 ; 0
Q_4	0 ; 0	1 ; 0.625	0.5 ; 1.25	0.625 ; 2.5	1.25 ; 5	0 ; 0
h^*	$*$	-1	1	1	1	$*$
$V^*(x)$	0	1	1.25	2.5	5	0

The algorithm converges after 4 iterations; $Q_4 = Q_3 = Q^*$. The last two rows of the table also give the optimal policy, computed from Q^* with (2.3), and the optimal V-function V^*, computed with (2.6). In the policy representation, the symbol $*$ means that any action can be taken in that state without changing the quality of the policy.

The total number of function evaluations required by the algorithm is $4\,|X|\,|U|\,(2 + |U|) = 4 \cdot 6 \cdot 2 \cdot 4 = 192$. Note that the number L of iterations given by (2.17) for a guaranteed suboptimality bound $\varepsilon_{\text{QI}} = 0.01$ is $L = 12$ (where $\|\rho\|_\infty = 5$ and $\gamma = 0.5$ have been used). So, in this case, the algorithm (fully) converges to its fixed point in far fewer iterations than the conservative number given by (2.17). □

2.4 Policy iteration

Policy iteration techniques iteratively evaluate and improve policies (Bertsekas and Tsitsiklis, 1996; Bertsekas, 2007). Consider a policy iteration algorithm that uses Q-functions. In every iteration ℓ, such an algorithm computes the Q-function Q^{h_ℓ} of the current policy h_ℓ; this step is called policy evaluation. When policy evaluation is complete, a new policy $h_{\ell+1}$ that is greedy in Q^h is computed; this step is called policy improvement. Policy iteration algorithms can be either model-based or model-free.

To implement policy evaluation, a Bellman equation different from (2.4) is used that characterizes the Q-function Q^h of a policy h. In the deterministic case, this equation is:

$$Q^h(x,u) = \rho(x,u) + \gamma Q^h(f(x,u), h(f(x,u))) \tag{2.20}$$

and in the stochastic case:

$$Q^h(x,u) = \mathrm{E}_{x' \sim \tilde{f}(x,u,\cdot)} \left\{ \tilde{\rho}(x,u,x') + \gamma Q^h(x', h(x')) \right\} \tag{2.21}$$

Analogously to (2.14), define the *policy evaluation mapping* $T^h : \mathcal{Q} \to \mathcal{Q}$, which computes the right-hand side of (2.20) (or of (2.21) in the stochastic case) for an arbitrary Q-function:

$$[T^h(Q)](x,u) = \rho(x,u) + \gamma Q(f(x,u), h(f(x,u))) \tag{2.22}$$

The Bellman equation for Q^h can be written concisely using the policy evaluation mapping as:

$$Q^h = T^h(Q^h) \tag{2.23}$$

Like the Q-iteration mapping T, T^h is a contraction with a factor $\gamma < 1$ in the infinity norm, i.e., for any pair of functions Q and Q', it is true that $\|T^h(Q) - T^h(Q')\|_\infty \leq \gamma \|Q - Q'\|_\infty$. However, unlike the Bellman optimality equation, which characterizes Q^*, equation (2.23) is linear in the Q-values. This is obvious in the deterministic case (2.20). When X and U have a finite cardinality, the stochastic equation (2.21) can be written in a linear form by writing the expectation as a sum.

Policy evaluation algorithms solve (2.23) to find Q^h. For instance, a model-based iterative policy evaluation algorithm can be given that works similarly to Q-iteration. This algorithm starts from an arbitrary Q-function Q^h_0 and in each iteration τ updates it using:[5]

$$Q^h_{\tau+1} = T^h(Q^h_\tau) \tag{2.24}$$

Because T^h is a contraction, this algorithm asymptotically converges to Q^h. Other ways to solve (2.23) include online, sample-based techniques similar to Q-learning, and directly solving the linear system of equations provided by (2.23), which is possible when the cardinality of $X \times U$ is not very large (Bertsekas, 2007).

The (offline) policy iteration starts with an arbitrary policy h_0. In each iteration ℓ, policy evaluation is performed to obtain the Q-function Q^{h_ℓ} of the current policy is h_ℓ. Then, an improved policy is computed which is greedy in Q^{h_ℓ}:

$$h_{\ell+1}(x) = \arg\max_u Q^{h_\ell}(x,u) \tag{2.25}$$

[5] A different iteration index τ is used for policy evaluation, because policy evaluation runs in the inner loop of every policy iteration ℓ.

The Q-functions computed by policy iteration asymptotically converge to Q^* as $\ell \to \infty$. Simultaneously, the policies converge to h^*.

The entire derivation can be repeated, and similar algorithms given, for V-functions V^h instead of Q-functions. For instance, the Bellman equation for V^h in the deterministic case is:

$$V^h(x) = \rho(x, h(x)) + V^h(f(x, h(x))) \tag{2.26}$$

Policy improvement is more problematic for V-functions, because a model is needed to compute a policy that is greedy in a V-function, see e.g., (2.8). Also, in the stochastic case expectations need to be evaluated using a formula similar to (2.13).

The main reason for which policy iteration algorithms are attractive is that the Bellman equation for Q^h (2.23) is linear in Q^h. This makes policy evaluation easier to solve than the Bellman optimality equation, which characterizes Q^* (2.4), and which is highly nonlinear due to the maximization in the right-hand side. Consider e.g., the iterative policy evaluation algorithm (2.24). The computational cost of every iteration of this algorithm, measured by the number of function evaluations, is $4\,|X|\,|U|$, where the functions being evaluated are ρ, f, h, and the current Q-function Q^{h_ℓ}. In contrast, a single Q-iteration (2.16) requires $|X|\,|U|\,(2 + |U|)$ function evaluations, which is larger whenever $|U| > 2$. Another advantage of policy evaluation is the following. When Q-functions need to be approximated, the class of approximators that guarantee convergence when combined with (2.20), is larger than the class of approximators that are provably convergent in combination with (2.4). This is discussed in more detail in Chapter 3.

Moreover, in practice, policy iteration algorithms usually converge in fewer iterations than value iteration algorithms (Bertsekas and Tsitsiklis, 1996; Bertsekas, 2007). However, this does not mean that policy iteration is less computationally costly than value iteration. For instance, even though policy evaluation using Q-functions is generally less costly than Q-iteration, *every* single policy iteration requires a complete policy evaluation. Additionally, it is unclear how the number of policy iterations can be usefully bounded. Therefore, it is an interesting open question how value iteration compares with policy iteration from the point of view of computational cost.

A similar comparison can be made between policy iteration and value iteration when V-functions are used instead of Q-functions.

Online policy iteration algorithms can also be given. In the online case, when iterative policy evaluation methods are used, policy improvements cannot wait until policy evaluation has converged (since that is only guaranteed to happen asymptotically). Instead, policy improvements have to be performed after a finite number of policy evaluation updates. A widely used type of online policy iteration consists of actor-critic algorithms, where the actor is the policy and the critic is the value function (Sutton et al., 2000; Konda and Tsitsiklis, 2003). In actor-critic methods, the policy is improved incrementally after every transition sample.

Example 2.3 Policy iteration for the cleaning robot. Consider again the cleaning robot problem of Example 2.1. Starting from a policy that always moves right ($h_0(x) = 1$ for all x), the policy iteration algorithm produces the sequence of Q-functions and policies given in Table 2.2 ($\gamma = 0.5$). Policy evaluation is implemented using (2.24), starting from identically zero Q-functions.

The algorithm converges after 2 (policy improvement) iterations; $h_2 = h_1 = h^*$. In fact, the policy is already optimal after the first policy improvement. Recall that the computational

Table 2.2: Policy iteration results for the cleaning robot

	$x=0$	$x=1$	$x=2$	$x=3$	$x=4$	$x=5$
h_0	$*$	1	1	1	1	$*$
$Q_0^{h_0}$	$0\,;\,0$	$0\,;\,0$	$0\,;\,0$	$0\,;\,0$	$0\,;\,0$	$0\,;\,0$
$Q_1^{h_0}$	$0\,;\,0$	$1\,;\,0$	$0\,;\,0$	$0\,;\,0$	$0\,;\,5$	$0\,;\,0$
$Q_2^{h_0}$	$0\,;\,0$	$1\,;\,0$	$0\,;\,0$	$0\,;\,2.5$	$1.25\,;\,5$	$0\,;\,0$
$Q_3^{h_0}$	$0\,;\,0$	$1\,;\,0$	$0\,;\,1.25$	$0.625\,;\,2.5$	$1.25\,;\,5$	$0\,;\,0$
$Q_4^{h_0}$	$0\,;\,0$	$1\,;\,0.625$	$0.3125\,;\,1.25$	$0.625\,;\,2.5$	$1.25\,;\,5$	$0\,;\,0$
$Q_5^{h_0}$	$0\,;\,0$	$1\,;\,0.625$	$0.3125\,;\,1.25$	$0.625\,;\,2.5$	$1.25\,;\,5$	$0\,;\,0$
h_1	$*$	-1	1	1	1	$*$
$Q_0^{h_1}$	$0\,;\,0$	$0\,;\,0$	$0\,;\,0$	$0\,;\,0$	$0\,;\,0$	$0\,;\,0$
$Q_1^{h_1}$	$0\,;\,0$	$1\,;\,0$	$0.5\,;\,0$	$0\,;\,0$	$0\,;\,5$	$0\,;\,0$
$Q_2^{h_1}$	$0\,;\,0$	$1\,;\,0$	$0.5\,;\,0$	$0\,;\,2.5$	$1.25\,;\,5$	$0\,;\,0$
$Q_3^{h_1}$	$0\,;\,0$	$1\,;\,0$	$0.5\,;\,1.25$	$0.625\,;\,2.5$	$1.25\,;\,5$	$0\,;\,0$
$Q_4^{h_1}$	$0\,;\,0$	$1\,;\,0.625$	$0.5\,;\,1.25$	$0.625\,;\,2.5$	$1.25\,;\,5$	$0\,;\,0$
h_2	$*$	-1	1	1	1	$*$

cost of every policy evaluation iteration (2.24) is $4\,|X|\,|U|$. Assuming that the maximization over U in (2.25) is solved by enumeration, the computational cost of every policy improvement is $|X|\,|U|$. Evaluating the policy takes 5 iterations for the first policy; and 4 for the second. So, the first policy iteration requires $5 \cdot 4 \cdot |X|\,|U| + |X|\,|U|$ function evaluations, and the second $4 \cdot 4 \cdot |X|\,|U| + |X|\,|U|$, for a total cost of $38\,|X|\,|U| = 38 \cdot 6 \cdot 2 = 456$. Compare this to the cost 192 of Q-iteration in Example 2.2; in this case, policy iteration is more computationally expensive. □

2.5 Direct policy search

Direct policy search techniques do not use value functions at all. Instead, they represent closed-loop policies, and search directly for an optimal policy using optimization techniques. An optimal policy h^* maximizes the return $R^{h^*}(x)$ for all $x \in X$ (note that $R^h(x) = V^h(x)$ by definition). The optimization criterion should therefore be a combination (e.g., weighted average) of the returns from every initial state.

In practice, the evaluation of the returns has to be performed in a finite time. To this end, the infinite sum in the return (2.1) (or (2.9) in the stochastic case) is approximated with a finite sum over the first K steps. To guarantee a maximum absolute error in estimating the returns $\varepsilon_{\text{MC}} > 0$, the number K can be chosen (e.g., Mannor et al., 2003):

$$K = \left\lceil \log_\gamma \frac{\varepsilon_{\text{MC}}(1-\gamma)}{\|\rho\|_\infty} \right\rceil \tag{2.27}$$

In the stochastic case, usually many sample trajectories need to be simulated to obtain an accurate estimate of the expected return.

In principle, any optimization technique can be used to search for the optimal policy. For a general MDP however, the optimization criterion may be a non-differentiable, non-convex

function with many local optima. This means that in general, global optimization techniques will obtain better policies than local, gradient-based methods. Particular examples include multi-start local optimization, genetic algorithms, tabu search, pattern search, and the cross-entropy method discussed in Chapter 6.

Evaluating the optimization criterion of direct policy search requires estimating returns from all the initial states. Estimating returns can be computationally expensive, especially in the stochastic case. Since optimization algorithms typically require many evaluations of the criterion, it follows that direct policy search algorithms are computationally expensive, usually more expensive than value iteration and policy iteration. If prior knowledge about the structure of the optimal policy is available (e.g., it is known that the optimal policy is linear in the state variables), the optimization problem can be simplified, and the computational efficiency of direct policy search improves. Direct policy search offers more flexibility in approximate DP/RL, where it allows general policy parameterizations without endangering convergence. In contrast, convergence proofs for approximate value iteration and policy iteration usually rely on restricted value function parameterizations (typically linear in the parameters). Algorithms for approximate DP/RL are discussed in detail in Chapter 3.

Next, the complexity of a policy search implementation for deterministic finite MDPs is investigated. Since the state and action spaces are finite (and therefore discrete), any combinatorial optimization technique could be used to look for the optimal policy. However, for simplicity, we consider an exhaustive search of the entire policy space. Because the number of possible policies is $|U|^{|X|}$, the return has to be evaluated for $|X|$ initial states, and the estimation of every return requires K simulation steps, the total number of simulation steps that have to be performed to find the optimal policy is at most $K\,|U|^{|X|}\,|X|$. Since f, ρ, and h are each evaluated once in every step, the computational cost, measured by the number of function evaluations, is at most $3K\,|U|^{|X|}\,|X|$. Compare this with the complexity $L\,|X|\,|U|\,(2+|U|)$ of Q-iteration (2.18). Clearly, this implementation of policy search is much more costly than Q-iteration for most values of $|X|$ and $|U|$.

Example 2.4 Direct policy search for the cleaning robot. Consider again the cleaning robot problem of Example 2.1, and assume the exhaustive policy search described above is applied. Take the approximation tolerance in the evaluation of the return $\varepsilon_{\mathrm{MC}} = 0.01$, which is equal to the Q-iteration threshold $\varepsilon_{\mathrm{QI}}$ of Example 2.2. Using $\|\rho\|_\infty = 5$, $\gamma = 0.5$, and applying (2.27), a time horizon of $K = 10$ steps is obtained. Therefore, the computational cost of the algorithm is bounded from above by $3K\,|U|^{|X|}\,|X| = 3 \cdot 2^6 \cdot 6 \cdot 10 = 11520$. By observing that it is unnecessary to look for optimal actions and to evaluate returns in the terminal states, this number can further be reduced to $3 \cdot 2^4 \cdot 4 \cdot 10 = 1920$. The actual number will be smaller since some trajectories will end up in the terminal states in less than 10 steps. Compare these numbers to the value 192 of the Q-iteration cost in Example 2.2, and to the value 456 of the policy iteration cost in Example 2.3. In this example, the exhaustive implementation of direct policy search is likely to be more expensive than both Q-iteration and policy iteration. □

2.6 Conclusions and open issues

In this chapter, the DP/RL problem has been introduced, together with the three main categories of algorithms to solve it: value iteration, policy iteration, and direct policy search.

Most of the representative algorithms selected for presentation have been chosen because they will be extended later on in the thesis. Many other algorithms exist; the interested reader is referred to the textbooks on DP and RL (Bertsekas, 2007; Sutton and Barto, 1998).

The most important open problem in DP/RL stems from the fact that the algorithms described above are not implementable for general MDPs. They can only be implemented when the state and action spaces consist of a finite number of elements. For state spaces with an infinite number of elements (e.g., continuous), the value function and the policy are infinite-dimensional objects that in general cannot be represented and computed exactly (in the case of Q-functions, an infinite number of actions also prohibits an exact representation). Even for finite MDPs, the cost of storing and computing value functions and policies grows exponentially with the number of state variables (and action variables, for Q-functions), so the algorithms are impractical for MDPs with many state and action variables. This latter problem is the curse of dimensionality. To cope with these problems, versions of the classical algorithms that represent value functions and/or policies approximately have to be used. This thesis focuses on approximate DP/RL for MDPs with continuous state and action variables.

Other major open issues include:

- Designing the reward function. Classical texts on RL have long recommended that the reward function should be kept as simple as possible; it should only reward the achievement of the final goal (Sutton and Barto, 1998). There are two problems with this approach. The first problem is that a simple reward function often makes online learning very slow. So, more information may need to be included in the reward function. The second problem is that often additional, higher-level requirements have to be considered in addition to the final goal. In stabilization problems from control theory, an example of such a requirement is a limit on the overshoot of the process's response, which is imposed in addition to the basic requirement of reaching the desired steady-state value. It is an open question how to translate such higher-level requirements into the 'language' of rewards.

- Providing guaranteed performance during learning, which is essential for applying RL in practice. Ideally, online RL algorithms should guarantee a monotonous increase in their expected performance. Unfortunately, this is most likely impossible because all the online RL algorithms need to explore, i.e., try out actions that may be suboptimal, in order to make sure their performance does not remain stuck in a local optimum. Therefore, weaker requirements could be used, where an overall trend of increased performance is guaranteed, while still allowing for bounded and temporary decreases in performance due to exploration. Most algorithms available today cannot guarantee such a steady performance improvement.

- Addressing problems where the state is not directly measurable, which are called partially observable in the DP/RL literature (Kaelbling et al., 1998; Porta et al., 2006). Usually, computationally intensive techniques that use belief states and general Bayesian inference are used to tackle these problems. However, unmeasured state variables can often be estimated from the trajectories of measured variables, using a model of the process dynamics. Control-theoretical analysis can determine whether such an estimation is possible (Isidori, 1995). If the estimation is possible, specialized, efficient state estimation techniques can be used to determine the values of the unmeasured state variables (Arulampalam et al., 2002). An important caveat is that, because state estimation

requires a model of the process dynamics, it is only applicable to DP. Nevertheless, even the opportunity of combining DP with state estimation is largely unexplored. It is also an important open question whether the approach can be extended to the model-free, RL case, and if so, in what way it could be extended.

- Using the prior knowledge about optimal or near-optimal policies, about the value function, or in the model-free (RL) case, about the process (recall that DP algorithms already assume a complete knowledge of the model). Prior knowledge about the value function could help, e.g., to initialize an algorithm with an informative initial value function, thereby decreasing the time it requires to converge. Prior knowledge about the policy can be used in policy search and policy iteration, to make the algorithms more efficient by restricting the class of policies they consider. In RL, prior knowledge about the process dynamics could, e.g., be used to form a partial model and pre-compute a (suboptimal) solution with DP. This solution could then be used to initialize the RL algorithm.

- Decentralized and distributed DP/RL. A number of new challenges arise when multiple controllers (agents) are involved. The curse of dimensionality is made worse by the fact that multiple agents are present in the system. The control actions of the agents have to be coordinated or else they could lead to unexpected and possibly severe consequences. Furthermore, if a given agent does not explicitly take the other agents into account, the Markov property is no longer valid: the evolution of the state depends on the (unknown) actions of the other agents. Due mostly to the severity of the curse of dimensionality in the multi-agent case, most of the work in multi-agent DP/RL deals with simple, often static (stateless) tasks (Buşoniu et al., 2008a, 2006a).[6] This setting is inappropriate for most distributed control problems. In this context, it can be very useful to exploit results in distributed and decentralized adaptive control, see e.g., (Jain and Khorrami, 1997; Jiang, 2000; Liu et al., 2007).

[6]Other work by the author of this thesis in the field of multi-agent DP/RL has been reported in (Buşoniu et al., 2005, 2006b).

Chapter 3

Approximate dynamic programming and reinforcement learning

This chapter reviews the literature on approximate dynamic programming and reinforcement learning. First, the need for approximation is explained. Then, approximate versions are given for the three categories of algorithms introduced in Chapter 2: value iteration, policy iteration, and direct policy search. The convergence properties of approximate value iteration and approximate policy iteration are discussed. Methods to automatically discover basis functions for linearly parameterized value function approximators are reviewed. The three classes of algorithms are compared. The chapter closes with a list of open issues in approximate dynamic programming and reinforcement learning.

3.1 Introduction

All the classical dynamic programming (DP) and reinforcement learning (RL) algorithms of Chapter 2 require the storage and updating of entire, exactly represented value functions (V-functions or Q-functions) and/or policies. Value iteration algorithms require exact representations of value functions. This means that distinct estimates of the return need to be stored and updated for every state (V-functions) or for every state-action pair (Q-functions). Algorithms for exact policy search require exact representations of policies, which means that distinct actions need to be stored and updated for every state. Policy iteration algorithms always require the storage of value functions. Certain policy iteration algorithms can avoid storing policies, as will be explained in Section 3.4; in general, however, the policies have to be stored.

When some of the state variables have a very large or infinite number of possible values (e.g., they are continuous), exact storage is no longer possible. Instead, value functions and (when explicitly stored) policies need to be approximated. Action variables with an infinite number of possible values make the representation of Q-functions additionally challenging. Since an overwhelming majority of the control problems have continuous state and action

variables, approximation is essential in DP and RL for control, and is required to solve all but the simplest problems. Even if the state and action variables only take a finite number of values, this number may still be very large, in which case approximation is also necessary.

An additional benefit of using approximation, rather than exact representations, is seen in online RL. Consider for instance the Q-learning algorithm of Section 2.3. Without approximation, the Q-value of every state-action pair has to be estimated separately (assuming it is possible to do so). When some of the states are visited only rarely, their Q-values are poorly estimated, and the algorithm makes poor control decisions in those states. However, when approximation is used, the approximator can be designed so that the Q-values of each state influence the Q-values of nearby states. This means that when good estimates of the Q-values of a certain state are available, the algorithm can also make reasonable control decisions in nearby states. This is called *generalization* in the RL literature, and can help algorithms work well despite using only a limited number of samples.

Two main types of approximators can be identified, namely parametric and nonparametric approximators. *Parametric* approximators are functions of a set of parameters. These parameters are tuned using data about the target value function or policy. However, typically the form of the approximator and its number of parameters do not depend on the data. A representative example is a linear combination of a fixed set of basis functions (BFs). In contrast, the form and number of parameters of a *nonparametric* approximator are derived from the available data. For instance, kernel-based approximators also represent the target function as a linear combination of BFs, but, unlike parametric approximation, they define one BF for each data point. In this chapter, the focus is placed on parametric approximation, because the contributions of the thesis (described in Chapters 4 – 6) rely on this type of approximation. Nevertheless, some of the approaches that will be reviewed in Section 3.5 change the form and number of parameters of a parametric approximator in order to best fit the data.

A significant research effort has been devoted to investigate theoretical guarantees and practical algorithms for approximate DP and RL. The theoretical properties of approximate value iteration and approximate policy evaluation (the latter used in approximate policy iteration) can be more easily analyzed when the value function is linearly parameterized. Because of these properties, linearly parameterized value function approximators are widely used in approximate DP and RL. Nevertheless, nonlinearly parameterized approximators such as neural networks are also popular. When combined with value iteration and policy iteration, they are more powerful than linearly parameterized approximators, but the resulting algorithms are more difficult to analyze. In the context of approximate policy search, there is a greater freedom in choosing the policy approximator, which can be nonlinearly parameterized without endangering convergence.

After presenting in more detail the need for approximation in DP/RL, in Section 3.2, the chapter continues with an overview of the literature on approximate DP/RL. Approximate value iteration is discussed first, in Section 3.3, followed by approximate policy iteration in Section 3.4. Section 3.5 reviews techniques to automatically derive value function approximators for value iteration and policy iteration. Approximate policy search is described in Section 3.6. Finally, Section 3.7 compares the three categories of algorithms, and Section 3.8 concludes the chapter.

3.2 The need for approximation in DP and RL

As explained in Section 3.1, the main reason for which approximation is required in DP and RL is that value functions and policies cannot be represented exactly when the state-action space contains an infinite number of elements. However, approximation in DP/RL is not only a problem of representation. Four other types of approximation are required to solve general DP/RL problems: sample-based updates, approximate maximization, sample-based estimation of expected values, and finite-time termination. This section explains where and why these approximations are necessary.

To understand why sampled updates are needed, consider e.g., the Q-iteration algorithm of Section 2.3. This algorithm iteratively applies the Q-iteration mapping: $Q_{\ell+1} = T(Q_\ell)$. The Q-iteration mapping would have to be implemented as follows:

$$\textbf{for every } (x,u): \quad Q_{\ell+1}(x,u) = \rho(x,u) + \gamma \max_{u' \in U} Q_\ell(f(x,u),u') \tag{3.1}$$

When the state-action space contains an infinite number of elements (e.g., when some of the variables are continuous), this implementation can no longer run in a finite time. Instead, an approximate version of the Q-function update has to be used, which only considers a finite number of state-action samples. Sampled updates are also necessary in approximate policy iteration. They are needed in the policy evaluation step, and also in the policy improvement step when the policies are explicitly stored. The following need for approximation arises in approximate policy search, and roughly corresponds to the need for sampled updates in value function techniques. The performance (return) cannot be optimized for every initial state; instead, a finite set of representative states has to be selected, see Section 3.6 for details.

In continuous or large action spaces, the maximization over the action variable in value iteration algorithms (e.g., in (3.1) for Q-iteration) is a nonlinear optimization problem that can only be solved approximately in general. Approximate maximization is also needed in the policy improvement step (2.25) of policy iteration, which would have to be implemented as:

$$\textbf{for every } x: \quad h_{\ell+1}(x) = \arg\max_u Q^{h_\ell}(x,u)$$

This potentially difficult maximization problem has to be solved for every sample x or (x,u). To simplify this maximization problem, most algorithms for approximate value iteration and policy iteration discretize the action space in a small number of values, and then solve the maximization problem by enumeration. Approximate maximization is not necessary in direct policy search. It is not necessary for the actor-critic subclass of policy iteration algorithms, either, because actor-critic algorithms use gradient updates of the policy instead of explicit policy improvements.

Next, the need for estimating expected values using samples is explained. When solving stochastic DP/RL problems, expectations over the stochastic transitions have to be computed. Consider the update of a single Q-value in the Q-iteration algorithm, which in the stochastic case is:

$$Q_{\ell+1}(x,u) = \mathrm{E}_{x' \sim \tilde{f}(x,u,\cdot)} \left\{ \tilde{\rho}(x,u,x') + \gamma \max_{u' \in U} Q_\ell(x',u') \right\}$$

Usually, the expectation cannot be computed exactly, but has to be evaluated approximately with Monte Carlo methods. Note that in many model-free algorithms, like Q-learning (2.19),

the Monte Carlo approximation does not appear explicitly in the update rules, but is performed implicitly while processing samples. In approximate policy search, the expected returns that enter the optimization criterion also have to be estimated with Monte Carlo methods.

Finally, the finite-time termination is explained. Iterative DP/RL algorithms have to be stopped in a finite time, although they usually only offer asymptotic convergence guarantees. This problem is typically handled by stopping the algorithms whenever the changes in the solution decrease below a certain threshold. The solution obtained by stopping the algorithm in this way is in general only an approximation of the solution that would be obtained asymptotically. In certain cases, bounds on the suboptimality of this approximate solution can be derived, see e.g., Section 2.3 for such a bound that applies to Q-iteration.

3.3 Approximate value iteration

Value iteration algorithms use the Bellman optimality equation to iteratively compute an optimal value function, from which an optimal policy is then derived (see Section 2.3). When the state-action space is large or continuous, the value function has to be approximated. The most widely used value function approximators are linear in the parameters. This is because the convergence properties of the resulting DP/RL algorithms can be analyzed more easily. Consider for instance a linearly parameterized approximator for the Q-function. Such an approximator has n BFs $\phi_1, \ldots, \phi_n : X \times U \to \mathbb{R}$, and is parameterized by a vector[1] of n parameters $\theta \in \mathbb{R}^n$. Given a parameter vector θ, approximate Q-values are computed with the formula:

$$\widehat{Q}(x,u) = \sum_{l=1}^{n} \phi_l(x,u)\theta_l \tag{3.2}$$

So, the parameter θ is a finite-dimensional representation of a Q-function. Note that, by tuning the number and form of the BFs, together with the parameters, an arbitrarily accurate representation of a given Q-function can be obtained. However, theoretical guarantees about approximate value iteration require that the BFs are fixed, which limits the representation power of (3.2). See however Section 3.5 for a discussion of methods to tune the BFs.

Nonlinearly parameterized approximators like neural networks are also popular. When combined with DP and RL, they are more powerful than linearly parameterized approximators, but the resulting algorithms are more difficult to analyze.

Algorithms for model-based (DP), approximate value iteration are presented in Section 3.3.1, followed by model-free (RL) algorithms in Section 3.3.2. We describe approximate DP in more detail, because our contribution to approximate value iteration is a model-based, DP algorithm (Chapter 4). Consequently, a more detailed background of approximate DP is required. Section 3.3.3 discusses the convergence properties of the algorithms for approximate value iteration, together with the important role of non-expansive approximators in guaranteeing convergence.

[1] All the vectors used in this thesis are column vectors.

3.3.1 Approximate model-based value iteration

This section details Q-iteration with a parametric approximator. This algorithm is an extension of the exact Q-iteration algorithm of Section 2.3. Recall that Q-iteration starts from an arbitrary Q-function Q_0 and in each iteration ℓ updates the Q-function using $Q_{\ell+1} = T(Q_\ell)$, where T is the Q-iteration mapping defined by (2.14), or (2.15) in the stochastic case. Approximate Q-iteration parameterizes the Q-function using a parameter vector $\theta \in \mathbb{R}^n$. In addition to T, two other mappings are needed to formalize approximate Q-iteration:

1. The *approximation* mapping $F : \mathbb{R}^n \to \mathcal{Q}$, which for a given value of the parameter vector θ produces an approximate Q-function $\widehat{Q} = F(\theta)$.

2. The *projection* mapping $P : \mathcal{Q} \to \mathbb{R}^n$, which computes a parameter vector θ such that $F(\theta)$ is as close as possible to the target Q-function Q (e.g., in a least-squares sense).

The notation $[F(\theta)](x, u)$ refers to the value of the Q-function $F(\theta)$ for the state-action pair (x, u). For instance, an approximator that is linear in the parameters would lead to $[F(\theta)](x, u) = \sum_{l=1}^{n} \phi_l(x, u)\theta_l$, according to (3.2). The notation $[P(Q)]_l$ refers to the lth component in the parameter vector $P(Q)$.

Approximate Q-iteration starts with an arbitrary (e.g., identically 0) parameter vector θ_0, and updates this vector in every iteration ℓ using the composition of the mappings P, T, and F:

$$\theta_{\ell+1} = P \circ T \circ F(\theta_\ell) \tag{3.3}$$

Of course, the results of F and T cannot be fully computed and stored. Instead, $P \circ T \circ F$ can be implemented as a single entity, or sampled versions of the full F and T mappings can be applied. Once a satisfactory parameter vector θ^* has been found, the following, approximately optimal policy can be used:

$$\widehat{h}^*(x) = \arg\max_u [F(\theta^*)](x, u) \tag{3.4}$$

A similar formalism can be given for approximate V-iteration, which is more popular in the literature (Gonzalez and Rofman, 1985; Chow and Tsitsiklis, 1991; Gordon, 1995; Tsitsiklis and Van Roy, 1996; Munos and Moore, 2002; Grüne, 2004). Many results from the literature deal with the discretization of continuous-variable problems (Gonzalez and Rofman, 1985; Chow and Tsitsiklis, 1991; Munos and Moore, 2002; Grüne, 2004). Such discretizations are not necessarily crisp, but can use interpolation schemes, which lead to linearly parameterized approximators of the form (3.2). The reader interested in a more detailed account of approximate DP is referred to (Bertsekas and Tsitsiklis, 1996; Rust, 1996; Bertsekas, 2007).

3.3.2 Approximate model-free value iteration

Approximate model-free value iteration has also been extensively studied (Singh et al., 1995; Moore and Atkeson, 1995; Horiuchi et al., 1996; Touzet, 1997; Jouffe, 1998; Glorennec, 2000; del R. Millán et al., 2002; Ormoneit and Sen, 2002; Szepesvári and Smart, 2004; Ernst et al., 2005). The Q-learning algorithm is the most popular, and has been combined with a variety of approximators, among which:

- Linearly parameterized approximators, used under several names such as interpolative representations (Szepesvári and Smart, 2004) and soft state aggregation (Singh et al., 1995).
- Neural networks and self-organizing maps (Touzet, 1997).
- Fuzzy rule-bases (Glorennec, 2000; Horiuchi et al., 1996; Jouffe, 1998).

Some algorithms for approximate model-free value iteration work offline and require a batch of samples collected in advance. A good example is the fitted Q-iteration algorithm of Ernst et al. (2005). This algorithm is similar to approximate Q-iteration (3.3), but the exact Q-iteration mapping T is replaced by an approximation derived from the available samples. Ensembles of regression trees are used to approximate the Q-function. The projection mapping P is replaced by a process that derives a new ensemble of regression trees in every iteration, in order to best approximate the current Q-function. Another batch algorithm is kernel-based RL, introduced by Ormoneit and Sen (2002). Both the ensembles of regression trees and the kernel-based representations are nonparametric.

3.3.3 Convergence and the role of non-expansive approximators

An important question in approximate DP/RL is whether the approximate solution computed by the algorithm converges. If it does converge, another important question is how far the convergence point is from the optimal solution. Convergence is important because a convergent algorithm is more amenable to analysis and meaningful performance guarantees. For the category of value iteration algorithms, the convergence proofs nearly always rely on contraction mapping arguments.

Consider for instance approximate Q-iteration, given by (3.3). The Q-iteration mapping T is a contraction in the infinity norm with factor $\gamma < 1$, as mentioned in Section 2.3. If the composite mapping $P \circ T \circ F$ of approximate Q-iteration is also a contraction, i.e., $\|P \circ T \circ F(\theta) - P \circ T \circ F(\theta')\|_\infty \leq \gamma' \|\theta - \theta'\|_\infty$ for all θ, θ', and for some $\gamma' < 1$, then approximate Q-iteration asymptotically converges to a unique fixed point.

Denote this fixed point by θ^*, and denote by $\mathcal{F}_{F \circ P} \subset \mathcal{Q}$ the set of fixed points of the composite mapping $F \circ P$ (this set is assumed non-empty). Define $\varepsilon'_Q = \min_{Q' \in \mathcal{F}_{F \circ P}} \|Q^* - Q'\|_\infty$, the minimum distance between Q^* and any fixed point of $F \circ P$. Then, the convergence point θ^* of approximate Q-iteration satisfies the following suboptimality bounds (Gordon, 1995; Tsitsiklis and Van Roy, 1996):

$$\|Q^* - F(\theta^*)\|_\infty \leq \frac{2\varepsilon'_Q}{1-\gamma} \tag{3.5}$$

$$\|Q^* - Q^{\widehat{h}^*}\|_\infty \leq \frac{4\gamma\varepsilon'_Q}{(1-\gamma)^2} \tag{3.6}$$

where $Q^{\widehat{h}^*}$ is the Q-function of the greedy policy $\widehat{h}^*$ in $F(\theta^*)$ (3.4). Equation (3.5) gives the suboptimality bound of the approximately optimal Q-function, whereas (3.6) gives the suboptimality bound of the resulting, approximately optimal policy. The latter may be more relevant in practice. The following relationship between the policy suboptimality and the Q-function suboptimality was used to obtain (3.6), and is also valid in general:

$$\|Q^* - Q^h\|_\infty \leq \frac{2\gamma}{(1-\gamma)} \|Q^* - Q\|_\infty \tag{3.7}$$

where the policy h is greedy in the (arbitrary) Q-function Q.

To be able to take advantage of these theoretical guarantees, it must be ensured that $P \circ T \circ F$ is a contraction. One way to do that is to ensure that F and P are non-expansions, i.e., that $\|F(\theta) - F(\theta')\|_\infty \leq \|\theta - \theta'\|_\infty$ for all θ, θ' and $\|P(Q) - P(Q')\|_\infty \leq \|Q - Q'\|_\infty$ for all Q, Q' (Gordon, 1995). In this case, $P \circ T \circ F$ is a contraction with factor $\gamma' = \gamma < 1$. When F is linearly parameterized (3.2), it is fairly easy to ensure its non-expansiveness by normalizing the BFs ϕ_l, so that for every x and u, we have $\sum_{l=1}^{n} \phi_l(x, u) = 1$.

Ensuring that P is non-expansive is more complicated. For instance, the most natural choice for P is a least-squares projection:

$$P(Q) = \arg\min_{\theta} \sum_{l_\mathrm{s}=1}^{n_\mathrm{s}} |Q(x_{l_\mathrm{s}}, u_{l_\mathrm{s}}) - [F(\theta)](x_{l_\mathrm{s}}, u_{l_\mathrm{s}})|^2 \tag{3.8}$$

for an arbitrary set of samples $\{(x_{l_\mathrm{s}}, u_{l_\mathrm{s}}) \mid l_\mathrm{s} = 1, \ldots, n_\mathrm{s}\}$. Unfortunately, such a projection can in general be an expansion, and examples of divergence when using this projection have been given (Tsitsiklis and Van Roy, 1996; Wiering, 2004). One way to make P non-expansive is to choose exactly $n_\mathrm{s} = n$ samples (for instance, the centers of the BFs), and require that $\phi_{l_\mathrm{s}}(x_{l_\mathrm{s}}, u_{l_\mathrm{s}}) = 1$ and $\phi_{l_\mathrm{s}'}(x_{l_\mathrm{s}}, u_{l_\mathrm{s}}) = 0$ for $l_\mathrm{s} \neq l_\mathrm{s}'$. Then, the projection mapping (3.8) reduces to an assignment that associated each parameter with the Q-value of the corresponding sample:

$$[P(Q)]_{l_\mathrm{s}} = Q(x_{l_\mathrm{s}}, u_{l_\mathrm{s}}) \tag{3.9}$$

and is clearly non-expansive. More general (but still restrictive) conditions under which $P \circ T \circ F$ is a contraction are given by Tsitsiklis and Van Roy (1996).

In the area of (model-free) approximate RL, many approaches are heuristic and do not provide any convergence guarantees (del R. Millán et al., 2002; Touzet, 1997; Horiuchi et al., 1996; Jouffe, 1998; Glorennec, 2000). Those that do guarantee convergence use linearly parameterized approximators (Singh et al., 1995; Ormoneit and Sen, 2002; Szepesvári and Smart, 2004; Ernst et al., 2005), and employ conditions related to the non-expansiveness properties above, e.g., for Q-learning (Singh et al., 1995; Szepesvári and Smart, 2004), or batch V-iteration (Ormoneit and Sen, 2002).

Another important theoretical property of algorithms for approximate DP and RL is consistency. In DP, an algorithm is consistent if the approximate value function converges to the optimal one as the approximation accuracy increases. Many consistency results for DP can be found for discretization-based approximators (Gonzalez and Rofman, 1985; Chow and Tsitsiklis, 1991; Santos and Vigo-Aguiar, 1998). In RL, consistency is usually understood as the convergence to a unique solution as the number of samples increases. The stronger result of convergence to an optimal solution as the approximation accuracy also increases is proven in (Ormoneit and Sen, 2002; Szepesvári and Smart, 2004).

Example 3.1 Grid Q-iteration for a servo-system. Consider a second-order discrete-time model of a servo-system:

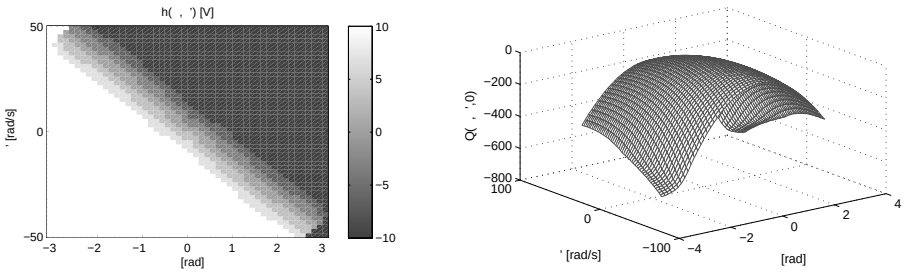
$$\begin{aligned} x_{k+1} &= f(x_k, u_k) = Ax_k + Bu_k \\ A &= \begin{bmatrix} 1 & 0.0049 \\ 0 & 0.9540 \end{bmatrix}, \quad B = \begin{bmatrix} 0.0021 \\ 0.8505 \end{bmatrix} \end{aligned} \tag{3.10}$$

This discrete-time model is obtained by discretizing a continuous-time model of the servo-system, which was determined by first-principles modeling of the real servo-system. The discretization is performed with the zero-order-hold method, using a sampling time of $T_{\mathrm{s}} = 0.005$ s. The position $x_{1,k} = \alpha$ is bounded to $[-\pi, \pi]$ rad, the velocity $x_{2,k} = \dot{\alpha}$ to $[-16\pi, 16\pi]$ rad/s, and the control input[2] u_k to $[-10, 10]$ V.

In our example, a discounted, linear quadratic regulation problem has to be solved. This problem is described by the following reward function:

$$\begin{aligned} r_{k+1} &= \rho(x_k, u_k) = -x_k^{\mathrm{T}} Q_{\mathrm{rew}} x_k - R_{\mathrm{rew}} u_k^2 \\ Q_{\mathrm{rew}} &= \begin{bmatrix} 5 & 0 \\ 0 & 0.01 \end{bmatrix}, \quad R_{\mathrm{rew}} = 0.01 \end{aligned} \tag{3.11}$$

The discount factor was chosen $\gamma = 0.95$ (this value is sufficient to produce a good control policy). A near-optimal solution to this problem is presented in Figure 3.1. This near-optimal solution was computed with the fuzzy Q-iteration algorithm of Chapter 4, using a very accurate approximator.



(a) A near-optimal policy.

(b) Projection of a near-optimal Q-function on X, for $u = 0$.

Figure 3.1: A near-optimal solution of the servo-system.

As an example of approximate value iteration, we develop a Q-iteration algorithm relying on a gridding of the state space, and on a discretization of the action space into a set of finitely many values, $U_{\mathrm{d}} = \{u_1, \ldots, u_M\}$. For this problem, three discrete actions are sufficient to find an acceptable stabilizing policy, $U_{\mathrm{d}} = \{-10, 0, 10\}$. The state space is gridded (partitioned) into a set of N disjoint rectangles. Let X_i be the surface of the ith rectangle in this partition, with $i = 1, \ldots, N$. The Q-function approximator assigns the same Q-values for all the states in X_i. This corresponds to a linearly parameterized approximator with binary-valued (0 or 1) BFs over the state-discrete action space $X \times U_{\mathrm{d}}$:

$$\phi_{[i,j]}(x, u) = \begin{cases} 1 & \text{if } x \in X_i \text{ and } u = u_j \\ 0 & \text{otherwise} \end{cases} \tag{3.12}$$

where $[i, j]$ denotes the single-dimensional index corresponding to i and j, which can be computed with $[i, j] = i + (j - 1)N$. Note that because the rectangles are disjoint, exactly one BF is active at any point of $X \times U_{\mathrm{d}}$.

[2]For all the examples in this thesis, the measurement units of variables are mentioned only once in the text, when the variables are introduced, after which they are omitted. However, for clarity, time measurements are always accompanied by units.

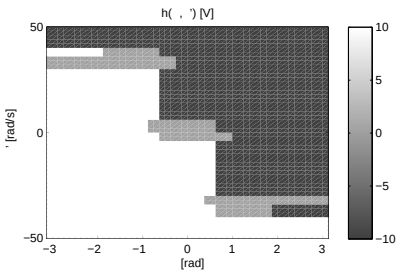
To derive the projection mapping P, the least-squares projection (3.8) is used, taking as samples the cross product between the set of centers c_i of all the rectangles, and U_d. These samples satisfy the conditions to simplify P to an assignment of the type (3.9):

$$[P(Q)]_{[i,j]} = Q(x_i, u_j) \tag{3.13}$$

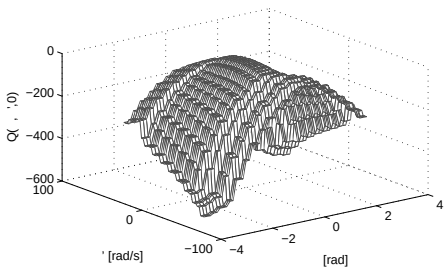
Using the linearly parameterized approximator (3.2) with the BFs (3.12) and the projection (3.13) yields the grid Q-iteration algorithm. Because F and P are non-expansions, the algorithm is convergent.

Next, we apply grid Q-iteration to the servo-system problem. Two different grids over the state space are used: a coarse grid, with 20 equidistant bins on each axis (leading to $20^2 = 400$ rectangles); and a fine grid, with 100 equidistant bins on each axis (leading to $100^2 = 10000$ rectangles). The algorithm is considered convergent when the maximum amount by which any parameter changes between two consecutive iterations does not exceed $\varepsilon_{\mathrm{QI}} = 0.001$. For the coarse grid, convergence occurs after 160 iterations, and for the fine grid, after 118. This shows that the number of iterations to convergence is not monotonously increasing in the number of parameters.

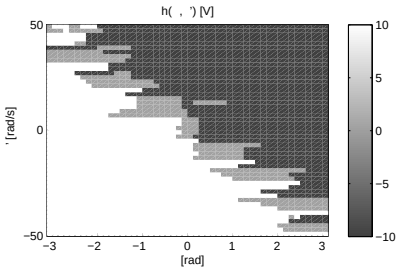
Projections of the resulting Q-functions on the state space, together with the corresponding policies computed with (3.4), are given in Figure 3.2. The accuracy in representing the Q-function is worse for the coarse grid, in Figure 3.2(b), than for the fine grid, in Figure 3.2(d). In Figure 3.2(b), the piecewise-constant nature of the approximator is clearly visible. The policy is not represented well in either case, although its structure is more clearly visible



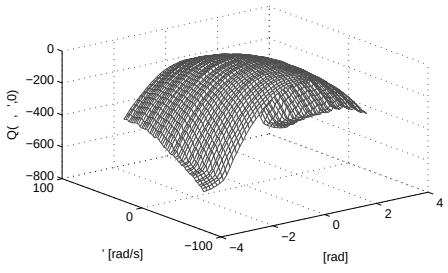
(a) Coarse-grid policy.



(b) Projection of coarse-grid Q-function on X, for $u = 0$.



(c) Fine-grid policy.



(d) Projection of fine-grid Q-function on X, for $u = 0$.

Figure 3.2: Grid Q-iteration solutions of the servo-system stabilization.

in Figure 3.2(c). Axis-oriented artifacts appear for both grid sizes. If these artifacts indeed decrease the returns obtained by the policy, they illustrate the fact that the bound on the suboptimality of the policy (3.6) is larger than the bound on the suboptimality of the value function (3.5). □

3.4 Approximate policy iteration

Policy iteration techniques compute in each iteration the value function of the current policy. Then, they compute a new, improved policy, which is greedy in the current value function, and repeat the cycle (see Section 2.4).

In general, both the value function and the policy are approximated. Approximating the value function is necessary, for the reasons described in Section 3.1. It is also a more difficult problem than approximating the policy. Approximating the policy can sometimes be avoided entirely, by computing actions on demand, on the basis of the current value function (Lagoudakis et al., 2002; Lagoudakis and Parr, 2003a). Alternatively, parametric approximators of the policy can be used. Even in that case, solving a static optimization problem is usually sufficient to determine the policy parameters.

Like in approximate value iteration, linearly parameterized approximators of the value function are widely used, because convergence can be guaranteed more easily with such approximators. Nonlinear approximators, especially neural networks, are also used often in the so-called actor-critic algorithms (Sutton et al., 2000; Konda and Tsitsiklis, 2003; Berenji and Vengerov, 2003).

This section focuses on the class of model-free, sample-based policy iteration algorithms, because our contribution to approximate policy iteration (Chapter 5) belongs to this class. So, a more detailed background of these methods is necessary. Approximate policy evaluation is discussed first in Section 3.4.1. Then, Section 3.4.2 describes exact and approximate policy improvement. Section 3.4.3 outlines some convergence results for approximate policy iteration. Finally, Section 3.4.4 presents an important subclass of approximate policy iteration, consisting of actor-critic techniques.

3.4.1 Approximate policy evaluation

Linearly parameterized approximators of the value function, combined with least-squares techniques to compute the parameters, offer the most powerful algorithms for approximate policy evaluation available to date (Bertsekas, 2007). These algorithms solve a linear system of equations to compute the value function parameters. They generally require fewer samples than their gradient-based counterparts.

Here, we focus on a particular algorithm called least-squares temporal difference for Q-functions (LSTD-Q) (Lagoudakis and Parr, 2003a). This algorithm is central to our approach of Chapter 5. For simplicity, we only consider deterministic Markov decision processes (MDPs), but the formalism can be easily extended to stochastic MDPs. LSTD-Q belongs to a class of policy evaluation algorithms that solve a projected form of the Bellman equation (Bertsekas, 2007). To introduce the projected Bellman equation, recall first the policy evaluation mapping (2.22):

$$[T^h(Q)](x,u) = \rho(x,u) + \gamma Q(f(x,u), h(f(x,u)))$$

and assume for now that X and U have a finite number of elements, $X = \{x_1, \ldots, x_{\bar{N}}\}$, $U = \{u_1, \ldots, u_{\bar{M}}\}$. Then, the policy evaluation mapping can be written in matrix form as:

$$\boldsymbol{T}^{\boldsymbol{h}}(\boldsymbol{Q}) = \boldsymbol{\rho} + \gamma \boldsymbol{f}\boldsymbol{h}\boldsymbol{Q} \tag{3.14}$$

where $\boldsymbol{T}^{\boldsymbol{h}} : \mathbb{R}^{\bar{N}\bar{M}} \to \mathbb{R}^{\bar{N}\bar{M}}$. Denote by $[i,j]$ the single-dimensional index corresponding to i and j, which can be computed with $[i,j] = i + (j-1)\bar{N}$. Then, the vectors and matrices in (3.14) can be defined as follows:[3]

- $\boldsymbol{Q} \in \mathbb{R}^{\bar{N}\bar{M}}$ is a vector representation of Q, with $\boldsymbol{Q}_{[i,j]} = Q(x_i, u_j)$.
- $\boldsymbol{\rho} \in \mathbb{R}^{\bar{N}\bar{M}}$ is a vector representation of ρ, with $\boldsymbol{\rho}_{[i,j]} = \rho(x_i, u_j)$.
- $\boldsymbol{f} \in \mathbb{R}^{\bar{N}\bar{M}\times\bar{N}}$ is a matrix representation of f, with $\boldsymbol{f}_{[i,j],i'} = 1$ if $f(x_i, u_j) = x_{i'}$, and 0 otherwise.
- $\boldsymbol{h} \in \mathbb{R}^{\bar{N}\times\bar{N}\bar{M}}$ is a matrix representation of h, with $\boldsymbol{h}_{i',[i,j]} = 1$ if $i' = i$ and $h(x_i) = u_j$, and 0 otherwise.

Consider a linearly parameterized approximator with n state-action BFs $\phi_1, \ldots, \phi_n : X \times U \to \mathbb{R}$. Define the matrix $\boldsymbol{\phi} \in \mathbb{R}^{\bar{N}\bar{M}\times n}$ with $\boldsymbol{\phi}_{[i,j],l} = \phi_l(x_i, u_j)$. The approximate Q-vector corresponding to a parameter θ is $\widehat{\boldsymbol{Q}} = \boldsymbol{\phi}\,\theta$. The set of exactly representable Q-vectors is the space spanned by the BFs, and can be written as $\widehat{\boldsymbol{\mathcal{Q}}} = \{\boldsymbol{\phi}\,\theta \mid \theta \in \mathbb{R}^n\}$, $\widehat{\boldsymbol{\mathcal{Q}}} \subset \mathbb{R}^{\bar{N}\bar{M}}$.

The *projected Bellman equation* corresponding to $\boldsymbol{\phi}$ is:

$$P^w \boldsymbol{T}^{\boldsymbol{h}}(\widehat{\boldsymbol{Q}}^h) = \widehat{\boldsymbol{Q}}^h \tag{3.15}$$

where P^w is a weighted Euclidean (least-squares) projection operator, which takes any Q-vector and projects it onto $\widehat{\boldsymbol{\mathcal{Q}}}$, according to:

$$\begin{aligned} P^w(\boldsymbol{Q}) &= \boldsymbol{\phi}\,\theta_{\boldsymbol{Q}} \\ \text{where} \quad \theta_{\boldsymbol{Q}} &= \arg\min_{\theta\in\mathbb{R}^n} \sum_{i=1}^{\bar{N}} \sum_{j=1}^{\bar{M}} w(x_i, u_j) \left| \phi^{\mathrm{T}}(x_i, u_j)\,\theta - \boldsymbol{Q}_{[i,j]} \right|^2 \end{aligned} \tag{3.16}$$

The weight function $w : X \times U \to [0,1]$ controls the distribution over the state-action space of the error (difference) between a Q-vector $\boldsymbol{Q}$ and its projection $P^w\boldsymbol{Q}$. It satisfies $\sum_{i,j} w(x_i, u_j) = 1$. The accuracy of the Q-function approximation is indirectly related to the weight function, via the projected Bellman equation (3.15). The projection matrix can be written in a closed form as $P^w = \boldsymbol{\phi}(\boldsymbol{\phi}^{\mathrm{T}}\boldsymbol{w}\boldsymbol{\phi})^{-1}\boldsymbol{\phi}^{\mathrm{T}}\boldsymbol{w}$, where $\boldsymbol{w} \in \mathbb{R}^{\bar{N}\bar{M}\times\bar{N}\bar{M}}$ is a diagonal matrix representation of w, with $\boldsymbol{w}_{[i,j],[i,j]} = w(x_i, u_j)$.

By substituting $\boldsymbol{T}^{\boldsymbol{h}}$ from (3.14), the closed-form expression for P^w, and the expression of the optimal approximate Q-vector $\widehat{\boldsymbol{Q}}^h = \boldsymbol{\phi}\theta^*$, the projected Bellman equation (3.15) becomes:

$$\boldsymbol{\phi}(\boldsymbol{\phi}^{\mathrm{T}}\boldsymbol{w}\boldsymbol{\phi})^{-1}\boldsymbol{\phi}^{\mathrm{T}}\boldsymbol{w}(\boldsymbol{\rho} + \gamma\boldsymbol{f}\boldsymbol{h}\boldsymbol{\phi}\theta^*) = \boldsymbol{\phi}\theta^*$$

[3] Throughout this thesis, boldface notation is used for vector or matrix representations of functions and mappings. Ordinary vectors and matrices are displayed in normal font.

This is an equation in θ^*, and can be transformed further into:

$$\boldsymbol{\phi}^{\mathrm{T}}\boldsymbol{w}(\boldsymbol{\phi} - \gamma \boldsymbol{f}\boldsymbol{h}\boldsymbol{\phi})\theta^* = \boldsymbol{\phi}^{\mathrm{T}}\boldsymbol{w}\boldsymbol{\rho} \tag{3.17}$$

Using the notations $\Gamma = \boldsymbol{\phi}^{\mathrm{T}}\boldsymbol{w}(\boldsymbol{\phi} - \gamma \boldsymbol{f}\boldsymbol{h}\boldsymbol{\phi})$ with $\Gamma \in \mathbb{R}^{n\times n}$ and $z = \boldsymbol{\phi}^{\mathrm{T}}\boldsymbol{w}\boldsymbol{\rho}$ with $z \in \mathbb{R}^n$, this equation can be written as:

$$\Gamma\theta^* = z \tag{3.18}$$

The matrix Γ and the vector z can both be written as sums of $\bar{N}\bar{M}$ terms, one for each state-action pair:

$$\begin{aligned}\Gamma &= \sum_{i=1}^{\bar{N}}\sum_{j=1}^{\bar{M}} \phi(x_i,u_j)w(x_i,u_j)[\phi(x_i,u_j) - \gamma\phi(f(x_i,u_j),h(f(x_i,u_j)))]^{\mathrm{T}} \\ z &= \sum_{i=1}^{\bar{N}}\sum_{j=1}^{\bar{M}} \phi(x_i,u_j)w(x_i,u_j)\rho(x_i,u_j)\end{aligned} \tag{3.19}$$

LSTD-Q uses a set of samples $\{(x_{l_\mathrm{s}}, u_{l_\mathrm{s}}, x'_{l_\mathrm{s}} = f(x_{l_\mathrm{s}}, u_{l_\mathrm{s}}), r_{l_\mathrm{s}} = \rho(x_{l_\mathrm{s}}, u_{l_\mathrm{s}})) \,|\, l_\mathrm{s} = 1,\ldots,n_\mathrm{s}\}$, with $(x_{l_\mathrm{s}}, u_{l_\mathrm{s}})$ drawn from a density defined by the weight function w: the probability of drawing the sample $(x_{l_\mathrm{s}}, u_{l_\mathrm{s}})$ is $w(x_{l_\mathrm{s}}, u_{l_\mathrm{s}})$. From these samples, Γ and z are estimated using the following update rules, which are derived from (3.19):

$$\begin{aligned}&\Gamma_0 = 0, \quad z_0 = 0 \\ &\Gamma_{l_\mathrm{s}} = \Gamma_{l_\mathrm{s}-1} + \phi(x_{l_\mathrm{s}}, u_{l_\mathrm{s}})\left[\phi(x_{l_\mathrm{s}}, u_{l_\mathrm{s}}) - \gamma\phi(x'_{l_\mathrm{s}}, h(x'_{l_\mathrm{s}}))\right]^{\mathrm{T}} \\ &z_{l_\mathrm{s}} = z_{l_\mathrm{s}-1} + \phi(x_{l_\mathrm{s}}, u_{l_\mathrm{s}})r_{l_\mathrm{s}}\end{aligned} \tag{3.20}$$

Note that the sample index is used to denote the update index, because Γ and z are updated after every sample. An approximate solution θ_{l_s} to (3.18) can be obtained after a number of l_s samples have been processed, by solving:

$$\frac{1}{l_\mathrm{s}}\Gamma_{l_\mathrm{s}}\theta_{l_\mathrm{s}} = \frac{1}{l_\mathrm{s}}z_{l_\mathrm{s}} \tag{3.21}$$

Because the empirical distribution of samples converges to w as $l_\mathrm{s} \to \infty$, and using (3.19), we have $\lim_{l_\mathrm{s}\to\infty} \frac{1}{l_\mathrm{s}}\Gamma_{l_\mathrm{s}} = \Gamma$ and $\lim_{l_\mathrm{s}\to\infty} \frac{1}{l_\mathrm{s}}z_{l_\mathrm{s}} = z$. Therefore, (3.21) asymptotically becomes equivalent to (3.18), and θ_{l_s} asymptotically converges to θ^*. For any finite l_s, the normalization by the number of samples is not necessary[4], and (3.21) is equivalent to:

$$\Gamma_{l_\mathrm{s}}\theta_{l_\mathrm{s}} = z_{l_\mathrm{s}} \tag{3.22}$$

This estimation of Γ and z from samples, followed by solving (3.22) to find θ_{l_s} (usually after the entire batch of samples has been processed, i.e., $l_\mathrm{s} = n_\mathrm{s}$), is the LSTD-Q algorithm. LSTD-Q has two important properties that make it efficient. Firstly, (3.22) is obviously linear, so it can be solved with an efficient procedure such as Gaussian elimination. Secondly, the matrices $\Gamma_{l_\mathrm{s}} \in \mathbb{R}^{n\times n}$ and the vectors $z_{l_\mathrm{s}} \in \mathbb{R}^n$ have manageable sizes. In the LSTD-Q algorithm, a vector or matrix having $\bar{N}\bar{M}$ elements or more never has to be stored or updated. Furthermore, the updates (3.20) can be applied without any change to state-action spaces

[4]Normalization may still be useful to avoid numerical errors.

with infinitely many elements (e.g., continuous). Note also that, when the BF vector $\phi(x,u)$ is sparse,[5] the computational efficiency of the updates (3.20) can be improved further by exploiting this sparsity.

An analogous least-squares algorithm can be given to compute linearly-parameterized, approximate V-functions (Bertsekas, 2007). However, for such an algorithm, samples cannot be collected arbitrarily; instead, they have to be collected from system trajectories controlled with the policy h that is being evaluated.

An algorithm similar to LSTD-Q, called least-squares policy evaluation, aims to compute the solution of the same projected Bellman equation (3.15), but in a different way. Least-squares policy evaluation was originally given for V-functions (Bertsekas and Ioffe, 1996; Bertsekas, 2007). We extend it to compute Q-functions in Appendix B. The main idea of the extended algorithm is to iteratively apply approximations of the projected Bellman mapping to update the parameter vector, rather than aiming directly for the optimal parameter vector, like LSTD-Q does. The (exact) projected Bellman mapping takes the parameter θ_{l_s} as input, and computes a new parameter $\theta_{l_\mathrm{s}+1}$ by solving:

$$\boldsymbol{\phi}\theta_{l_\mathrm{s}+1} = P^w \boldsymbol{T}^{\boldsymbol{h}}(\boldsymbol{\phi}\theta_{l_\mathrm{s}})$$

The approximations of this mapping become increasingly accurate as the number l_s of samples increases. Gradient-based versions of policy evaluation with linearly parameterized approximators can also be given (Sutton, 1988), but they are less sample-efficient than these least-squares techniques.

3.4.2 Policy improvement

Once the approximate value function of the policy h_ℓ in iteration ℓ is available, policy improvement has to be performed. When Q-functions are used, the policy can be improved with:

$$h_{\ell+1}(x) = \arg\max_u \widehat{Q}^{h_\ell}(x,u) \tag{3.23}$$

When V-functions are used, policy improvements are more complicated and require a model:

$$h_{\ell+1}(x) = \arg\max_u [\rho(x,u) + \gamma \widehat{V}^{h_\ell}(f(x,u))] \tag{3.24}$$

Note that in the stochastic case, (3.23) remains unchanged, but (3.24) includes an additional expectation inside the maximization criterion, like in (2.13). This makes policy improvements based on the V-function significantly more computationally expensive.

Consider first the case where policies are approximated and parameterized by a vector $\vartheta \in \mathbb{R}^{\mathcal{N}}$. For instance, a linearly parameterized policy approximator uses a set of state-dependent BFs $\varphi_1, \ldots, \varphi_{\mathcal{N}} : X \to \mathbb{R}$, and computes the approximate policy with:[6]

$$\widehat{h}(x) = \sum_{i=1}^{\mathcal{N}} \varphi_i(x)\vartheta_i = \varphi^{\mathrm{T}}(x)\vartheta \tag{3.25}$$

[5] The BF vector is sparse, e.g., when the discrete-action approximator of the upcoming Example 3.2 is used. This is because the BF vector contains zeros for all the discrete actions different from the current discrete action.

[6] Calligraphic notation is used to differentiate variables related to policy approximation, from variables related to value function approximation. So, the policy parameter is ϑ and the policy BFs are φ, whereas the value function parameter is θ and the value function BFs are ϕ. Furthermore, the number of policy parameters and BFs is $\mathcal{N}$, and the number of samples for policy approximation is $\mathcal{N}_\mathrm{s}$.

where $\varphi(x) = [\varphi_1(x), \ldots, \varphi_{\mathcal{N}}(x)]^{\mathrm{T}}$. For simplicity, the parameterization (3.25) is only given for scalar actions, but it is easy to extend it to the case of multiple action variables. Approximate policy improvement can be performed by solving the least-squares problem:

$$\vartheta_{\ell+1} = \arg\min_{\vartheta \in \mathbb{R}^{\mathcal{N}}} \sum_{i_{\mathrm{s}}=1}^{\mathcal{N}_{\mathrm{s}}} \left\| \varphi^{\mathrm{T}}(x_{i_{\mathrm{s}}})\vartheta - \arg\max_{u} \phi^{\mathrm{T}}(x_{i_{\mathrm{s}}}, u)\theta_\ell \right\|_2^2 \tag{3.26}$$

to find an improved policy parameter vector $\vartheta_{\ell+1}$, where $\{x_1, \ldots, x_{\mathcal{N}_{\mathrm{s}}}\}$ is a set of samples for policy improvement. In this formula, $\arg\max_u \phi^{\mathrm{T}}(x_{i_{\mathrm{s}}}, u)\theta_\ell = \arg\max_u \widehat{Q}^{\widehat{h}_\ell}(x_{i_{\mathrm{s}}}, u)$ is the greedy action for the sample $x_{i_{\mathrm{s}}}$; notice that the policy $\widehat{h}_\ell$ is now also an approximation. Alternatively, policy improvement could be performed using:

$$\vartheta_{\ell+1} = \arg\max_{\vartheta \in \mathbb{R}^{\mathcal{N}}} \sum_{i_{\mathrm{s}}=1}^{\mathcal{N}_{\mathrm{s}}} \phi^{\mathrm{T}}(x_{i_{\mathrm{s}}}, \varphi^{\mathrm{T}}(x_{i_{\mathrm{s}}})\vartheta)\theta_\ell = \arg\max_{\vartheta \in \mathbb{R}^{\mathcal{N}}} \sum_{i_{\mathrm{s}}=1}^{\mathcal{N}_{\mathrm{s}}} \widehat{Q}^{\widehat{h}_\ell}(x_{i_{\mathrm{s}}}, \varphi^{\mathrm{T}}(x_{i_{\mathrm{s}}})\vartheta) \tag{3.27}$$

which maximizes the approximate Q-values of the actions chosen by the policy in the state samples. However, (3.27) can generally be a difficult nonlinear optimization problem, whereas (3.26) is a convex optimization problem, which is easier to solve.

Some algorithms for approximate policy iteration avoid an explicit parameterization of the policy. Instead, they compute actions on demand for every state where a control action is required, using (3.23) or (3.24). For instance, if only a small, discrete set of actions is considered in the policy improvement step, the maximization can be solved by enumeration. In this case, policy improvement is exact.

The algorithm obtained by combining exact policy improvement with policy evaluation by LSTD-Q is least-squares policy iteration (LSPI) (Lagoudakis et al., 2002; Lagoudakis and Parr, 2003a). LSPI forms the basis of our approach of Chapter 5.

3.4.3 Convergence guarantees

Like for approximate value iteration, convergence is an important theoretical question also for approximate policy iteration. Consider the general case where both the value functions and the policies are approximated. Consider also the case where Q-functions are used. Assume that the error in every policy evaluation step is bounded by ε_Q:

$$\|\widehat{Q}^{\widehat{h}_\ell} - Q^{\widehat{h}_\ell}\|_\infty \leq \varepsilon_Q, \quad \text{for any } \ell \geq 0$$

and the error in every policy improvement step is bounded by ε_h, in the following sense:

$$\|T^{\widehat{h}_{\ell+1}}(\widehat{Q}^{\widehat{h}_\ell}) - T(\widehat{Q}^{\widehat{h}_\ell})\|_\infty \leq \varepsilon_h, \quad \text{for any } \ell \geq 0$$

where $T^{\widehat{h}_{\ell+1}}$ is the policy evaluation mapping for the improved (approximate) policy, and T is the Q-iteration mapping (2.14). Then, approximate policy iteration eventually produces policies whose performance is within a bounded distance from the optimal performance (Lagoudakis and Parr, 2003a):

$$\limsup_{\ell \to \infty} \left\| \widehat{Q}^{\widehat{h}_\ell} - Q^* \right\|_\infty \leq \frac{\varepsilon_h + 2\gamma\varepsilon_Q}{(1-\gamma)^2} \tag{3.28}$$

For an algorithm that performs exact policy improvements, such as LSPI, $\varepsilon_h = 0$ and the bound is strengthened to:

$$\limsup_{\ell \to \infty} \|\widehat{Q}^{h_\ell} - Q^*\|_\infty \leq \frac{2\gamma\varepsilon_Q}{(1-\gamma)^2} \tag{3.29}$$

where $\|\widehat{Q}^{h_\ell} - Q^{h_\ell}\|_\infty \leq \varepsilon_Q$, for any $\ell \geq 0$. Note that computing ε_Q (and, when approximate policies are used, computing ε_h) may be difficult in practice, and the existence of these error bounds may require additional assumptions on the MDP, such as the Lipschitz continuity of the dynamics and reward function.

These convergence guarantees do not imply that the policy will settle (i.e., converge to a stationary function). For instance, both the value function and policy parameters might converge to limit cycles, so that every point on the cycle yields a policy that satisfies the bound. Similarly, when exact policy improvements are used, the value function parameter may oscillate, implicitly leading to an oscillating policy.

Similar results hold when V-functions are used instead of Q-functions (Bertsekas and Tsitsiklis, 1996).

In sample-based policy iteration, instead of waiting with policy improvement until a large number of samples have been processed and an accurate approximation of Q-function for the current policy has been obtained, policy improvements can also be performed after a small number of samples. In the extreme case, a policy that is greedy in the current value function can be computed and applied at every step; then, the new sample is obtained and the value function parameter is updated as well. Such a variant is sometimes called *optimistic* policy iteration (Bertsekas and Tsitsiklis, 1996; Bertsekas, 2007). When parameter updates are performed after each sample, the method is called fully optimistic. Otherwise, it is partially optimistic. One particular instance where optimistic updates are necessary, is when approximate policy iteration is applied online, such as in our method of Chapter 5.

The convergence behavior of optimistic policy iteration is not fully understood. It can e.g., exhibit a phenomenon called chattering, whereby the value function converges to a stationary function, while the policy sequence oscillates, because the limit of the value function parameter corresponds to multiple policies (Bertsekas and Tsitsiklis, 1996; Bertsekas, 2007).

Example 3.2 LSPI for the servo-system. In this example, LSPI will be applied to the servo-system problem of Example 3.1. The action space is again discretized into the set $U_\mathrm{d} = \{-10, 0, 10\}$, which contains $M = 3$ actions. Only these discrete actions are allowed into the set of samples. A set of N normalized Gaussian radial basis functions (RBFs) $\bar{\phi}_i : X \to \mathbb{R}$, $i = 1, \ldots, N$, is used as an approximator over the state space.[7] The RBFs are defined as follows:

$$\bar{\phi}_i(x) = \frac{\phi'_i(x)}{\sum_{i'=1}^N \phi'_{i'}(x)}, \qquad \phi'_i(x) \quad = \exp\left[-\sum_{d=1}^2 \frac{(x_d - c_{i,d})^2}{b_{i,d}^2}\right] \tag{3.30}$$

where ϕ'_i are (non-normalized) Gaussian axis-parallel RBFs, $(c_{i,1}, c_{i,2})$ is the center of the ith RBF, and $(b_{i,1}, b_{i,2})$ is its radius. The centers of the RBFs are arranged on an equidistant 9×9

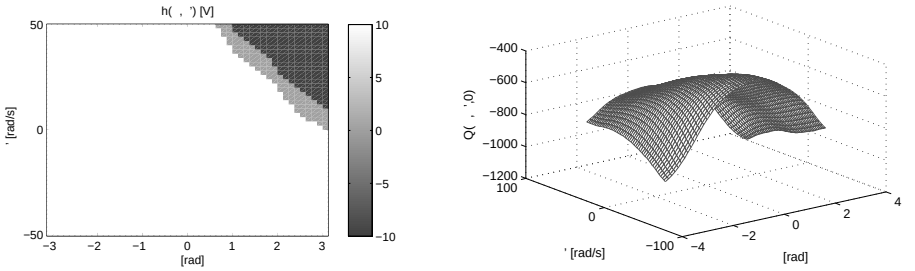
[7] In the context of value function approximation, we denote BFs that only depend on the state by $\bar{\phi}$, to differentiate them from the state-action BFs ϕ, whenever these two types of BFs appear together. When there is no possibility of confusion, the simpler notation ϕ is used for both types of BFs.

grid in the state space. The radii of the RBFs along each dimension are taken identical to the distance along that dimension between two adjacent RBFs; this yields a smooth interpolation of the Q-function in the state space. The RBFs are replicated for every discrete action. To compute the state-discrete action BFs, all RBFs that do not correspond to the current discrete action are taken equal to 0. Approximate Q-values can then be computed with $\widehat{Q}(x, u_j) = \phi^{\mathrm{T}}(x, u_j)\,\theta$, for the state-action BF vector:

$$\phi(x, u_j) = [\underbrace{0, \ldots, 0}_{u_1}, \ldots, 0, \underbrace{\bar{\phi}_1(x), \ldots, \bar{\phi}_N(x)}_{u_j}, 0, \ldots, \underbrace{0, \ldots, 0}_{u_M}]^{\mathrm{T}} \in \mathbb{R}^{NM}$$

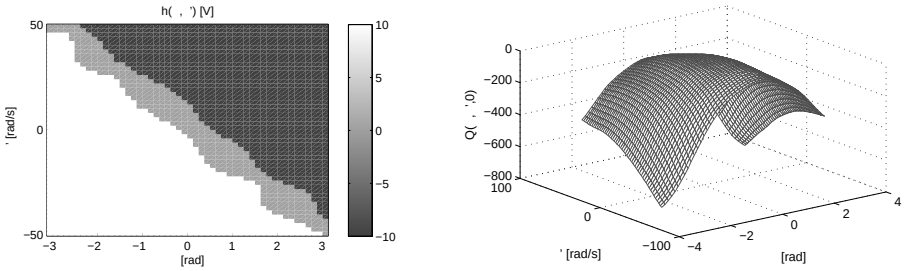
and a parameter vector $\theta \in \mathbb{R}^n$ with $n = NM = 3N$.

First, LSPI with exact policy improvements is applied, starting from an identically zero initial policy h_0. The same set of $n_{\mathrm{s}} = 7500$ samples is used in every LSTD-Q policy evaluation. The samples are random, uniformly distributed over the state-discrete action space $X \times U_{\mathrm{d}}$. To illustrate the results of LSTD-Q, Figure 3.3 presents the first improved policy computed by the algorithm, h_1, and its approximate Q-function, computed with LSTD-Q.



(a) Policy. (b) Projection of the Q-function on X, for $u = 0$.

Figure 3.3: An early policy and its approximate Q-function, for LSPI with exact policy improvements.



(a) Policy. (b) Projection of the Q-function on X, for $u = 0$.

Figure 3.4: Results of LSPI with exact policy improvements on the servo-system.

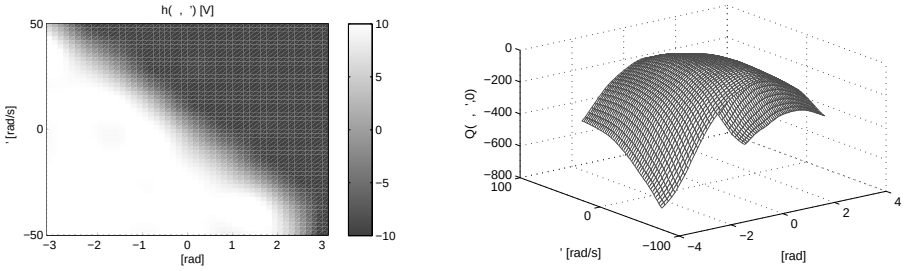
The complete LSPI algorithm converged in 11 iterations. Figure 3.4 shows the resulting policy and Q-function. Both the policy and the Q-function in Figure 3.4 are good approx-

imations of the near-optimal ones in Figure 3.1, even though only 9×9 BFs were used. Compared with the results of grid Q-iteration in Figure 3.2, LSPI needed fewer BFs and was able to find a much better approximation of the policy. This is mainly because the Q-function is largely smooth (see Figure 3.1(b)), which means it can be represented well using the wide RBFs considered. In contrast, the grid BFs give a discontinuous approximate Q-function, which is inappropriate for this problem. Although certain types of continuous BFs can be used with Q-iteration, using wide RBFs such as these is unfortunately not possible, because they do not satisfy the assumptions for convergence, and indeed lead to divergence when they are too wide.

A disadvantage of these wide RBFs is that they fail to identify the policy nonlinearities in the top-left and bottom-right corners in Figure 3.1(a), and the corresponding non-smooth changes in the Q-function. Another observation is that even though the number of parameters is not much smaller than for the coarse grid in Example 3.1 ($9 \cdot 9 \cdot 3 = 243$ for LSPI, and $20 \times 20 \times 3 = 1200$ for the coarse grid), the algorithm converges in a significantly fewer iterations: 11 instead of 160. This is an indication that the convergence rate of exact policy iteration over exact value iteration (seen in Example 2.3) is maintained also in the approximate case.

Next, LSPI with approximate policy improvements (and approximate policies) is applied. The policy approximator is (3.25) and uses the same RBFs as the Q-function approximator ($\varphi_i = \bar{\phi}_i$). The approximate policy produces *continuous* actions. These have to be quantized (into discrete actions elements of U_d) before performing policy evaluation, because the Q-function approximator only works for discrete actions. Policy improvement is performed with (3.26), using a set of $\mathcal{N}_\mathrm{s} = 2500$ random, uniformly distributed state samples. The same set is used in every iteration.

In this experiment, both the Q-functions and the policies *oscillate* in the steady state of the algorithm, with a period of 2 iterations. Figure 3.5 presents one of the two policies from the limit cycle, and one of the Q-functions. The differences between the two distinct policies and Q-functions on the limit cycle are too small to be noticed in a figure. Instead, Figure 3.6 shows the evolution of the policy parameter that changes the most in steady state. Its oscillation is clearly visible. The policy and Q-function have a similar accuracy to those computed with exact policy improvements. The approximate policy has the added advantage that it produces continuous actions.



(a) Policy.

(b) Projection of the Q-function on X, for $u = 0$.

Figure 3.5: Results of LSPI with approximate policy improvement on the servo-system.

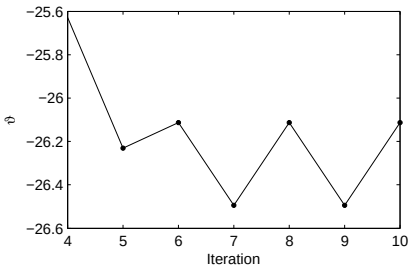


Figure 3.6: The variation of one of the policy parameters, starting with the 4th iteration and until the oscillation was detected.

The appearance of oscillations when parameterized policies is related to the following fact. The introduction of parameterized policies requires approximate policy improvements. Therefore, instead of the bound (3.29) for exact policy improvements, which applied to the discrete-action experiment in the first part of the example (Figures 3.3 and 3.4), the weaker bound (3.28) now applies. □

3.4.4 Actor-critic methods

Actor-critic techniques are a special case of online approximate policy iteration. They were introduced by Barto et al. (1983), and have been investigated often since then, e.g., (Berenji and Khedkar, 1992; Konda and Tsitsiklis, 2003; Berenji and Vengerov, 2003; Borkar, 2005; Nakamura et al., 2007). In actor-critic methods, both the policy and the value function are approximated using differentiable approximators (often neural networks), and updated using gradient rules. The 'critic' is the approximate value function, and the 'actor' is the approximate policy. The critic is typically a V-function, although using Q-functions is also possible. The critic updates aim to estimate the value function of the current policy. Because this policy keeps changing after every sample, the estimate will only be accurate if and when the algorithm has converged. Note that, because they use gradient updates, actor-critic algorithms can remain stuck in locally optimal solutions.

Next, a typical actor-critic algorithm is formalized. Denote by $\widehat{V}(x;\theta)$ the approximate V-function, parameterized by $\theta \in \mathbb{R}^N$, and by $\widehat{h}(x;\vartheta)$ the approximate policy, parameterized by $\vartheta \in \mathbb{R}^{\mathcal{N}}$. We use the notation $\widehat{V}(x;\theta)$ (and respectively, $\widehat{h}(x;\vartheta)$) to make explicit the dependence of the parameter vector θ (and respectively, ϑ). Because the algorithm does not distinguish between the value functions of the different policies, the value function notation is not superscripted by the policy. After each transition from x_k to x_{k+1}, as a result of action u_k, the so-called temporal difference is computed:

$$\delta_{\mathrm{TD},k} = r_{k+1} + \gamma \widehat{V}(x_{k+1};\theta_k) - \widehat{V}(x_k;\theta_k)$$

This is the difference between the right-hand and left-hand sides of the Bellman equation for the policy V-function (2.26), or a sample of the difference in the stochastic case. This temporal difference is analogous to the temporal difference for Q-functions, used e.g., in Q-learning (2.19). Of course, the exact values of the current state, $V(x_k)$, and of the next state, $V(x_{k+1})$ are not available, so they are replaced by their approximations.

Once the temporal difference $\delta_{\mathrm{TD},k}$ is available, the policy and V-function parameters are updated with:

$$\theta_{k+1} = \theta_k + \alpha_\mathrm{C} \left.\frac{\partial \widehat{V}(x;\theta)}{\partial \theta}\right|_{x_k,\theta_k} \delta_{\mathrm{TD},k} \tag{3.31}$$

$$\vartheta_{k+1} = \vartheta_k + \alpha_\mathrm{A} \left.\frac{\partial \widehat{h}(x;\vartheta)}{\partial \vartheta}\right|_{x_k,\vartheta_k} [u_k - \widehat{h}(x_k;\vartheta_k)]\, \delta_{\mathrm{TD},k} \tag{3.32}$$

where α_C and α_A are learning rates (step sizes) of the critic and the actor, respectively, and the notation $\left.\frac{\partial \widehat{V}(x;\theta)}{\partial \theta}\right|_{x_k,\theta_k}$ means that the derivative $\frac{\partial \widehat{V}(x;\theta)}{\partial \theta}$ is evaluated for the state x_k and the parameter θ_k (and analogously in (3.32)). In the critic update (3.31), the temporal difference takes the place of the prediction error $V(x_k) - \widehat{V}(x_k;\theta_k)$, where $V(x_k)$ is the exact value of x_k given the current policy. Since this exact value is not available, it is replaced by the estimate $r_{k+1} + \gamma \widehat{V}(x_{k+1};\theta_k)$ offered by the Bellman equation (2.26), thus leading to the temporal difference. In the actor update (3.32), the actual action u_k applied at step k can be different from the action $\widehat{h}(x_k;\vartheta_k)$ indicated by the policy. This change of the action indicated by the policy is the form taken by exploration in the actor-critic algorithm. When the exploratory action u_k leads to a positive temporal difference, the policy is adjusted towards this action. Conversely, when $\delta_{\mathrm{TD},k}$ is negative, the policy is adjusted away from u_k. This is because, like in the critic update, the temporal difference is interpreted as a correction of the predicted performance, so that e.g., if the temporal difference is positive, the obtained performance is considered better than the predicted one.

Note that because the exploration $[u_k - \widehat{h}(x_k;\vartheta_k)]$ is used to compute the error signal, without using greedy actions as targets, it is not necessary to solve a difficult optimization problem over the action variable to perform policy improvement. This means that continuous actions are easy to handle. In fact, actor-critic algorithms are specifically designed for continuous-action MDPs.

In general, the convergence of actor-critic methods cannot be guaranteed. However, for a particular version of the actor-critic algorithm, convergence guarantees are possible. Unlike all the algorithms presented up to this point, this convergent actor-critic algorithm does not aim to maximize the discounted return (2.1) (or (2.9) in the stochastic case). Instead, it tries to maximize, by optimizing the policy, the average reward:

$$\bar{\rho}^h = \int_X \int_U \rho(x,u) \mathrm{P}^h(x,u) \mathrm{d}u\, \mathrm{d}x$$

where $\mathrm{P}^h(x,u)$ is the steady-state probability of the state-action pair (x,u) under the policy h.

The actor is a *stochastic* policy $\tilde{h}(x,u;\vartheta)$ parameterized by $\vartheta \in \mathbb{R}^\mathcal{N}$, where $\tilde{h}(x,u;\vartheta)$ gives the probability of selecting u in x, given the parameter ϑ. The critic approximates the *advantage* Q-function $Q_\mathrm{adv} : X \times U \to \mathbb{R}$ of the current policy, which can be defined e.g., for deterministic MDPs by the equation:

$$Q_\mathrm{adv}(x,u) = \rho(x,u) - \bar{\rho}^h + \mathrm{E}_{u' \sim \tilde{h}(x,\cdot;\vartheta)} \{Q_\mathrm{adv}(f(x,u),u')\}$$

This Q-function needs to be approximated using a linear parameterization with a specific set

of $\mathcal{N}$ BFs, defined by:

$$[\phi_1(x,u),\ldots,\phi_{\mathcal{N}}(x,u)]^{\mathrm{T}} = \frac{1}{\tilde{h}(x,u;\vartheta)} \cdot \frac{\partial \tilde{h}(x,u;\vartheta)}{\partial \vartheta}$$

The derivative on the right-hand side produces $\mathcal{N}$ components, which are then used as the Q-function BFs. In fact, other BFs can be used in addition to these $\mathcal{N}$, without losing the convergence guarantees. The Q-function parameter can be obtained by solving a projected Bellman equation analogous to (3.15), with gradient or least-squares techniques.

Explaining in detail the convergence properties of this algorithm is beyond the scope of this chapter. In simple terms, when using a specific form of gradient update, the algorithm can be guaranteed to converge to a policy that yields a locally optimal average reward $\bar{\rho}^h$. This algorithm and its convergence proof were given independently by Konda and Tsitsiklis (2000, 2003) and Sutton et al. (2000). The convergence guarantee was used by Berenji and Vengerov (2003) to prove the convergence of an actor-critic algorithm with a fuzzy parameterization of the actor.

3.5 Finding value function approximators automatically

The majority of value function approximators used in the literature are parametric. Given an approximator, the DP/RL algorithm computes its parameters. There still remains the problem of finding a good approximator. Since most approaches consider linearly parameterized approximators, finding an approximator is in most cases equivalent to finding a set of basis functions (BFs).

The BFs can be designed in advance, in which case two approaches are possible. The first approach is to design the BFs so that a uniform resolution (approximation accuracy) is obtained over the entire state space (for V-functions) or over the entire state and action spaces (for Q-functions). Unfortunately, such an approach suffers from the curse of dimensionality: the complexity of a uniform approximator grows exponentially with the number of state (and possibly action) variables. The second approach is to focus the resolution in certain parts of the state (or state-action) space, where the value function has a more complex shape, or where it is more important to approximate it accurately. Prior knowledge about the shape of the value function or about the importance of certain areas of the state-action space is necessary in this case. Unfortunately, such knowledge is often non-intuitive and very difficult to obtain without actually computing the value function.

A more general alternative is to find BFs automatically, rather than designing them. Such an approach should provide BFs suited to each particular problem. BFs can be either constructed offline (e.g., Menache et al., 2005; Mahadevan and Maggioni, 2007), or adapted while the DP/RL algorithm is running (e.g., Munos and Moore, 2002; Ratitch and Precup, 2004). Since convergence guarantees typically rely on a fixed set of BFs, adapting the BFs while the DP/RL algorithm is running leads to a loss of these guarantees. Convergence guarantees can be recovered by ensuring that BF adaptation is stopped after a finite number of updates; fixed-BF proofs can then be applied to show asymptotic convergence (e.g., Ernst et al., 2005). Most of the approaches to the automatic discovery of BFs assume that the MDP has a discrete action space containing a not too large number of actions. The effort is then focused on finding good state-dependent BFs.

Using many uniformly-distributed, narrow BFs to ensure a desired resolution may be detrimental to online RL algorithms, in addition to being computationally expensive. This is because narrow BFs lead to a poor generalization of control decisions across neighboring states. To have good generalization, it is preferable to use fewer, wider BFs. This tradeoff between resolution and generalization is specific to approximate RL. Algorithms to find BFs automatically aim, either explicitly or implicitly, for a set of BFs that ensures a good balance between resolution and generalization.

A considerable amount of effort has been spent in this research area, and a large number of techniques is available to derive BFs automatically (Chow and Tsitsiklis, 1991; Munos and Moore, 2002; Ernst et al., 2005; Menache et al., 2005; Mahadevan and Maggioni, 2007). In the remainder of this section, we give a brief overview of these techniques. Section 3.5.1 presents resolution refinement techniques, Section 3.5.2 discusses BF optimization, and Section 3.5.3 presents some other techniques for constructing BFs.

3.5.1 Resolution refinement

Resolution refinement techniques start with a few BFs (a coarse resolution) and then refine the BFs as the need arises. They can be further classified in two categories:

- Local refinement (splitting) techniques evaluate whether a particular area of the state space (corresponding to one or several neighboring BFs) has a sufficient accuracy, and add new BFs when the accuracy is deemed insufficient. Such techniques have been proposed e.g., for V-iteration (Munos and Moore, 2002), Q-iteration (Munos, 1997), Q-learning (Ratitch and Precup, 2004; Waldock and Carse, 2008), and policy evaluation (Grüne, 2004).

- Global refinement techniques evaluate the global accuracy of the representation, and refine the BFs if the accuracy is deemed insufficient. All the BFs can be refined uniformly (Chow and Tsitsiklis, 1991), or the algorithm may decide which areas of the state space require more resolution (Munos and Moore, 2002). For instance, Chow and Tsitsiklis (1991); Munos and Moore (2002) apply global refinement to V-iteration, while Szepesvári and Smart (2004) use it for Q-learning.

A variety of criteria are used to decide when the BFs should be refined. An overview of typical criteria, and a comparison between them in the context of V-iteration, is given by Munos and Moore (2002). For instance, local refinement in a certain area can be performed:

- when the value function is not (approximately) constant in that area (Munos and Moore, 2002);

- when the value function is not (approximately) linear in that area (Munos and Moore, 2002; Munos, 1997);

- when the Bellman error (the error between the left-hand and right-hand sides of the Bellman equation, see (3.33)) is large in that area (Grüne, 2004);

- or using various other heuristics (Ratitch and Precup, 2004; Waldock and Carse, 2008).

Global refinement can be performed e.g., until a desired level of solution accuracy is met (Chow and Tsitsiklis, 1991). Munos and Moore (2002) refine the areas where the V-function is poorly approximated *and* that affect other areas where the actions dictated by the policy change. This approach globally identifies the areas of the state space that have to be approximated more accurately in order to find a good policy, but is limited to discrete-action MDPs.

Resolution refinement techniques increase the memory and computational expenses of the DP/RL algorithm whenever they increase the resolution. Care must be taken to prevent the memory and computation expenses from becoming prohibitive, especially in the online case. This is an important concern both in approximate DP and in approximate RL. Equally important in approximate RL are the restrictions imposed on resolution refinement by the limited amount of data available. Increasing the power of the approximator means that more data will be required to compute an accurate solution, so the resolution cannot be refined to arbitrary levels for a given amount of data.

3.5.2 Basis function optimization

Basis function optimization techniques search for the best placement and shape of a fixed number of BFs. Consider e.g., the linear parameterization (3.2) of the Q-function. To optimize the n BFs, they can be parameterized by a vector of BF parameters ξ that encodes their locations and shapes. For instance, a radial BF is characterized by its center and width. Denote the parameterized BFs by $\varphi_l(x;\xi) : X \times U \to \mathbb{R}$, $l = 1, \ldots, n$, to highlight their dependence on ξ. The BF optimization algorithm searches for an optimal parameter vector ξ, which optimizes a certain criterion related to the accuracy of the value function representation. Note that unlike resolution refinement techniques, BF optimization cannot provide arbitrary accuracies across the entire state space, because the number of BFs is limited.

Many optimization techniques can be applied to compute the BF parameters ξ. For instance, gradient-based optimization has been used for actor-critic (Berenji and Khedkar, 1992), temporal difference (Singh et al., 1995), and least-squares temporal difference algorithms (Menache et al., 2005). The cross-entropy method has been applied to least-squares temporal difference (Menache et al., 2005) and Q-iteration algorithms (Buşoniu et al., 2008d). Our approach from (Buşoniu et al., 2008d) will be detailed in Chapter 4.

An approximator that natively optimizes the BFs is the self-organizing map, used e.g., in combination with Q-learning by Touzet (1997).

The most widely used optimization criterion is the Bellman error, also called Bellman residual (Singh et al., 1995; Menache et al., 2005). This is a measure on how much the estimated value function violates the Bellman equation, which would be precisely satisfied by the exact value function. For instance, the Bellman error for an estimate $\widehat{Q}$ of the optimal Q-function Q^*, in a deterministic MDP, is derived from the Bellman optimality equation (2.4), and is given by:

$$\int_X \int_U \left| \widehat{Q}(x,u) - \rho(x,u) - \gamma \max_{u'} \widehat{Q}(f(x,u),u') \right|^2 \mathrm{d}u \, \mathrm{d}x \tag{3.33}$$

In practice, an approximation of the Bellman error is computed using a finite set of samples. The suboptimality $\|\widehat{Q} - Q^*\|_\infty$ of the resulting Q-function is bounded by a constant multiple of the infinity norm of the Bellman error $\|\widehat{Q} - T\widehat{Q}\|_\infty$ (Williams and Baird, 1994; Bertsekas

and Tsitsiklis, 1996). Furthermore, the suboptimality of the Q-function is related to the suboptimality of the resulting policy by (3.7), which means that, in principle, minimizing the Bellman error is useful. Unfortunately, minimizing the *quadratic* Bellman error (3.33) may lead to a large *infinity norm* of the Bellman error, so it is unclear whether minimizing (3.33) leads to a near-optimal approximate Q-function and policy.

Another possible criterion for optimizing the BFs is the performance of the (suboptimal) policy obtained by the DP/RL algorithm, as in our approach from (Buşoniu et al., 2008d).

3.5.3 Other methods for basis function construction

It is also possible to construct BFs using various specialized techniques that are different from resolution refinement and optimization. For instance, Ernst et al. (2005) use regression trees to represent the Q-function in every iteration of an algorithm for approximate Q-iteration. The method to build the regression trees implicitly determines a set of BFs that represent well the Q-function of the current iteration.

Mahadevan (2005) and Mahadevan and Maggioni (2007) perform a spectral analysis of the MDP transition dynamics to find BFs, which are then used in LSPI. Because the BFs represent the underlying topology of the state transitions, they provide a good accuracy in representing the value function.

Xu et al. (2007) use a kernel-based approximator for LSPI. Originally, kernel-based approximation uses a BF for each sample, but Xu et al. (2007) employ a kernel sparsification procedure that automatically determines a reduced number of BFs.

These three techniques are model-free.

Example 3.3 Finding RBFs for LSPI in the servo-system problem. Consider again the servo-system problem of Example 3.1, and its solution using LSPI described in Example 3.2. As already noted in Example 3.2, the LSPI solution of Figure 3.4(a) does not take into account the nonlinearities of the policy seen in the top-left and bottom-right corners of Figure 3.1(a). This is because the corresponding non-smooth variations in the Q-function, seen in Figure 3.1(b), are not represented in the approximate solutions. This is an example of the generalization-resolution tradeoff: the wide RBFs used offer good generalization, at the cost of poor resolution. To improve the resolution in the corners where the Q-function is not smooth, a resolution refinement technique could be applied.

An alternative is to parameterize the RBFs (3.30), and optimize their locations and shapes. In this case, the RBF parameter vector, denoted by ξ, contains the two-dimensional centers and radii of all the RBFs: $\xi = [c_{1,1}, c_{1,2}, b_{1,1}, b_{1,2}, \ldots, c_{N,1}, c_{N,2}, b_{N,1}, b_{N,2}]^{\mathrm{T}}$. Such an approach is proposed by Menache et al. (2005), who minimize the Bellman error using gradient descent and cross-entropy optimization. □

3.6 Approximate policy search

Algorithms for approximate policy search parameterize the policy directly, and use optimization techniques to search for an optimal parameter vector. When the state space is finite and not too large, the parameterization might exactly represent the optimal policy. This special case was discussed already in Section 2.5. However, in general, optimal policies can only be represented approximately.

Denote by $\widehat{h}(x;\vartheta)$ the approximate policy, parameterized by $\vartheta \in \mathbb{R}^{\mathcal{N}}$. Policy search algorithms search for an optimal ϑ^* that maximizes the return $R^{\widehat{h}(x;\vartheta)}$ for all $x \in X$. As already outlined in Section 3.2, three further types of approximation are necessary to implement a general policy search algorithm:

1. When X is large or continuous, computing the return for every state is not possible. A practical procedure to circumvent this difficulty requires choosing a finite set X_0 of representative initial states. Returns are estimated only for states in X_0, and the optimization criterion (score) is the weighted average return over X_0 (Marbach and Tsitsiklis, 2003; Munos, 2006):

$$s(\vartheta) = \sum_{x_0 \in X_0} w(x_0) R^{\widehat{h}(x;\vartheta)}(x_0) \tag{3.34}$$

   The representative states are weighted by $w : X_0 \to (0, 1]$. The set X_0, together with the weight function w, will determine the performance of the resulting policy. For instance, if the process only has to be controlled starting from a known set of initial states, then X_0 should be equal to this set, or included it when the set is too large. Also, initial states that are deemed more important can be assigned larger weights. When all initial states are equally important, the elements of X_0 should be uniformly spread over the state space, randomly or equidistantly, and identical weights equal to $\frac{1}{|X_0|}$ should be assigned to every element of X_0 ($|\cdot|$ denotes set cardinality). Maximizing only the returns from states in X_0 only results in an approximately optimal policy, because it cannot guarantee that returns from other states in X are maximal.

2. In the computation of the returns, the infinite sum in (2.1) has to be replaced by a finite sum over K steps. For discounted returns, it is easy to determine a value of K that guarantees a required estimation accuracy, using (2.27).

3. Finally, in stochastic MDPs, Monte Carlo simulations are required to estimate the expected returns. This procedure is consistent, i.e., as the number of simulations approaches infinity, the estimate converges to the correct expectation. Results from Monte Carlo simulation can be applied to bound the approximation error for a finite number of simulations.

In the literature, ad-hoc policy parameterizations are typically designed using intuition and prior knowledge about an optimal (or near-optimal) policy. For instance, parameterizations that are linear in the state variables can be used, if it is known that a (near-)optimal policy has the form of a linear state feedback. Ad-hoc parameterizations are typically combined with gradient-based optimization (Baird and Moore, 1999; Sutton et al., 2000; Marbach and Tsitsiklis, 2003; Konda and Tsitsiklis, 2003; Munos, 2006; Riedmiller et al., 2007). Theoretical results have been developed for this class of techniques, with recent work focusing on variance reduction for the gradient estimates (Marbach and Tsitsiklis, 2003; Munos, 2006).

When prior knowledge about the policy is not available, a richer policy parameterization has to be used. In this case, the optimization criterion is likely to have many local optima. Since gradient-based algorithms can only find locally optimal solutions, they will typically perform poorly for general policy parameterizations. Global optimization techniques are more suitable in this case. One example of global optimization technique which has been

applied to approximate policy search is the cross-entropy (CE) method. CE policy search was introduced by Mannor et al. (2003). Chang et al. (2007, Chapter 4) recently proposed finding a policy with the model-reference adaptive search, which is closely related to the CE method. Both works focus on solving finite, small MDPs, although they also propose solving large or continuous-variable MDPs with policy parameterizations based on prior knowledge. Our approach from Chapter 6 also uses the CE method with flexible policy parameterizations, in order to solve *continuous-state* MDPs. Evolutionary computation is another optimization technique that has been applied to policy search, both when a model is available (Barash, 1999; Chin and Jafari, 1998; Chang et al., 2007, Chapter 3) and when it is not (Gomez et al., 2006).

Example 3.4 Approximate policy search for the servo-system. Consider again the servo-system problem of Example 3.1. Because the system is linear and the reward function is quadratic, the optimal policy would be a linear state feedback if the constraints on the state and action variables were disregarded. Taking now into account the constraints on the control action, we assume that a good approximation of the optimal policy will be linear in the state variables, up to the constraints on the control action:

$$\widehat{h}(x) = \mathrm{sat}\left\{\vartheta_1 x_1 + \vartheta_2 x_2, -10, 10\right\} \tag{3.35}$$

where 'sat' denotes saturation. In fact, an examination of the near-optimal policy of Figure 3.1(a) reveals that this assumption is largely correct; the only nonlinearities appear in the top-left and bottom-right corners of the figure, and they are probably due to the constraints on the state variables. A policy search algorithm could use the parameterization (3.35) for the policy, and search for an optimal parameter vector $\vartheta^* = [\vartheta_1^*, \vartheta_2^*]^{\mathrm{T}}$. Many optimization algorithms could be used to achieve this. Because the gradients of the score function (3.34) are difficult to compute, a gradient-free method is recommended.

As explained above, further approximations are necessary to evaluate the score function:

- A set X_0 of representative states has to be selected. To make the policy perform equally well across the state space, the states in X_0 can be distributed on a regular grid, e.g., $X_0 = \{-16\pi, -15\pi, \ldots, 16\pi\} \times \{-\pi, -5\pi/6, ..., \ldots, \pi\}$, and uniformly weighted by $w(x_0) = \frac{1}{|X_0|}$.
- The infinite-horizon returns need to be estimated in a finite number of steps K. We impose a maximum error $\varepsilon_{\mathrm{MC}} = 0.01$ in the estimation of the return. A bound on the reward function (3.11) can be computed with:

$$\|\rho\|_\infty = [16\pi\ \pi] \begin{bmatrix} 5 & 0 \\ 0 & 0.01 \end{bmatrix} \begin{bmatrix} 16\pi \\ \pi \end{bmatrix} \approx 75.6142$$

  Substituting $\varepsilon_{\mathrm{MC}}$, $\|\rho\|_\infty$, and $\gamma = 0.95$ in (2.27) leads to $K = 233$ time samples.

Because the problem is deterministic, simulating multiple trajectories from every initial state is not necessary; instead, a single trajectory will suffice. Since the number of states in X_0 is $33 \cdot 13 = 429$, a total number of $429 \cdot 233 = 99957$ simulation steps have to be performed to evaluate each policy parameter. Because the optimization procedure is likely to evaluate many different parameter vectors, and samples cannot be reused to evaluate different policy

parameters, the resulting algorithm is computationally very expensive. The complexity can be decreased by taking a smaller X_0 or a larger ε_{MC}, at the expense of a possible decrease in the performance of the obtained policy.

By comparison, the grid Q-iteration with the fine grid of Example 3.1 required 30000 samples (10000 rectangle centers times 3 discrete actions). The algorithm reused these samples in each of the 118 iterations necessary for convergence. The LSPI algorithm with exact policy improvements, applied to the servo-system problem in Example 3.2, used even fewer transition samples, 7500. The algorithm required 11 iterations for convergence, so it reused the samples 11 times.

Consider now the case in which no prior knowledge about the optimal policy is available. In this case, a general policy parameterization has to be used, e.g., the linear policy parameterization (3.25) with the state-dependent RBFs (3.30). In contrast to Example 3.2, where the policy parameter vector ϑ has been optimized by computing approximate Q-functions and then improving ϑ on the basis of these Q-functions, here ϑ is optimized directly to maximize (3.34). The score of every policy parameter (optimization criterion) is computed in the same way as for the simpler parameterization derived from prior knowledge. However, because there is a significantly larger number of parameters to find than for simple parameterization (81 parameters for 9×9 RBFs, in contrast to 2 parameters for the simple parameterization), the optimization algorithm will probably require more score evaluations, and therefore more computations. □

3.7 Comparison of approximate value iteration, policy iteration, and policy search

Currently, an extensive theoretical or empirical comparison between the three categories of algorithms (approximate value iteration, policy iteration, and policy search) is missing from the literature. However, a few general remarks can be made.

As already discussed in Section 2.4, policy iteration algorithms typically converge in fewer iterations than value iteration algorithms. However, because the cost of an individual policy iteration may be greater than the cost of an individual value iteration, it is unclear how value iteration compares with policy iteration from the point of view of computational cost. In the approximate case, convergence guarantees for policy evaluation are less restrictive than value iteration guarantees, which is an advantage for approximate policy iteration. Namely, for policy evaluation it suffices that the BFs are linear in the parameters, as discussed in Section 3.4.3. For value iteration, additional properties are required to ensure that the approximate value iteration mapping is a contraction, as discussed in Section 3.3.3. Moreover, efficient least-squares algorithms such as LSTD-Q can be used to compute a 'one-shot' solution to the policy evaluation problem. Both these advantages stem from the *linearity* of the Bellman equation for the value function of a given policy, e.g., (2.20); whereas the Bellman optimality equation, which characterizes the optimal value function, e.g., (2.4), is *highly nonlinear* due to the maximization in the right-hand side. Note however that for value iteration, monotonous convergence to a unique solution is usually guaranteed, whereas policy iteration is generally only guaranteed to converge to a sequence of policies that all provide a guaranteed level of performance (see Section 3.4.3).

Approximate policy search is potentially very useful in two situations. The first situation

is when the form of the optimal policy (or a near-optimal policy) is known, and only the parameters need to be determined. In this case, a parametric approximator easily follows, and an optimization technique can be used to find an optimal parameter vector (locally optimal, in the case of local optimization techniques such as gradient-based optimization). The second situation is when a DP/RL problem has to be solved, but it is undesirable to compute value functions, e.g., because approximate value iteration and policy iteration fail to obtain a good solution, or because their assumptions are too restrictive. In such cases, one may use a general policy parameterization and a global optimization technique to search for the parameters. An assumption of value function techniques that may be too restrictive in certain problems is e.g., the linearity of the value function parameterization, which is typically required for convergence. Approximate policy search allows general policy parameterizations without endangering convergence. Note that, because of its generality, this approach incurs a high computational cost. Although it does not apply directly to the approximate case, the comparison of Section 2.5 between the computational costs of policy search and value iteration can offer a rough idea about how the two classes of algorithms compare when they use approximation.

3.8 Conclusions and open issues

In this chapter, the need for approximation in DP and RL has been explained, and approximate versions for the three main categories of DP/RL algorithms have been discussed: approximate value iteration, policy iteration, and policy search. Additionally, the three categories of approximate techniques have been compared, and techniques to automatically determine BFs for value function approximators have been reviewed. By necessity, we have only selected for presentation a few representative algorithms and central theoretical results; the breadth of the field prohibits an exhaustive discussion of all the available approaches and results. For the reader interested in a more extensive discussion, the textbooks of Bertsekas and Tsitsiklis (1996), and of Bertsekas (2007, Chapter 6) provide more detailed introductions to approximate DP and RL.

The following list describes some of the main open issues in approximate DP and RL:

- Owing to their sample efficiency and relaxed convergence guarantees, least-squares techniques may be the most promising approach for solving DP/RL problems. However, these techniques are usually given in an offline setting. From an RL perspective, it would be interesting to study least-squares techniques in the online case. While it is not difficult to derive online variants of the least-squares algorithms, these versions are typically not studied in the literature, and their behavior in practice is unknown. In Chapter 5, we extend the LSPI algorithm to the online case and empirically study the resulting algorithm.

- Continuous-action MDPs are not studied often in the literature. This is mainly because of the difficulties with continuous actions discussed in Section 3.1. However, for some problems continuous actions are important. For instance, stabilizing a system around an unstable equilibrium requires chattering of the control action when only discrete actions are available. In practice, chattering increases wear and may damage the system. In Chapter 4, we study the performance effect of the accuracy in discretizing actions,

for a model-based algorithm. In Chapter 5, we study LSPI with continuous-action parameterizations.

- Algorithms for approximate policy search typically require that the form of an optimal (or near-optimal) policy is known, and they optimize the parameters. Such an approach cannot be applied when prior knowledge about the form of the policy is not available. In that case, more general parameterizations have to be used. We propose such a general parameterization in Chapter 6, and we use cross-entropy (CE) optimization to search for the policy parameters.

- While adaptive approximators for value functions have been extensively studied (they were reviewed in Section 3.5), a comparative survey of the various methods is missing, so it is not clear which method is best for a given problem. Also, for techniques that adapt the basis functions while estimating the value function, no convergence proof is available. In Chapter 4, we give an algorithm to optimize BFs that evaluates each BF parameter by running the value function estimation to convergence. Thus, convergence problems are avoided, at the expense of more computations.

- It is unclear how more general types of prior knowledge than the form of the policy can be used in approximate DP/RL. Since prior knowledge about the value function is typically difficult to obtain, it is more interesting to investigate prior knowledge about the policy. For instance, the form (parameterization) of the policy may be unknown, but more general constraints may be available. It may be known, e.g., that the policy is monotonous in the state variables, or linear constraints on the state and action variables may be available. Policy iteration and policy search algorithms are most suited to incorporate policy constraints, because they can represent policies explicitly. In Chapter 5, we investigate the effect of incorporating monotonicity constraints on the policy in an online variant of the LSPI algorithm.

Some open questions from the field of exact DP/RL remain valid also in approximate DP/RL. Examples include defining the reward function, guaranteeing performance during learning, dealing with non-measurable state variables (partially observable MDPs), and decentralized (multi-agent) approximate DP/RL. Some of these questions gain additional importance when approximate solutions are considered. For instance, one may ask what properties the reward function should have in order to improve the convergence rate of a given algorithm for approximate DP, or the learning rate of a given algorithm for approximate RL. In Chapter 4, we show that in an example with continuous state and action variables, continuous reward functions are important for a predictable improvement in the performance of the approximate solution, as the approximation accuracy increases.

Chapter 4

Approximate value iteration with a fuzzy representation

In this chapter, we propose an algorithm for approximate dynamic programming that relies on a fuzzy partition of the state space, and on a discretization of the action space. We show that the algorithm converges to a solution that lies within a bound of the optimal solution. Under continuity assumptions on the dynamics and on the reward function, we also show that the algorithm is consistent, i.e., that the optimal solution is asymptotically obtained as the approximation accuracy increases. As an alternative to designing membership functions in advance, we propose a technique to optimize triangular membership functions using the cross-entropy method. An extensive numerical and experimental evaluation indicates that the algorithm has a good performance in practice, among others producing better policies than Q-iteration with a radial basis function approximator. The experiments also indicate that a continuous reward function is important for a predictable improvement of the performance as the approximation accuracy increases.

4.1 Introduction

Classical value iteration algorithms require an exact representation of the value function. When some of the state or action variables have a very large or infinite number of possible values (e.g., they are continuous), the value function can no longer be represented exactly, but has to be approximated. Fuzzy approximators are one of the possible representations of the value function. They have been applied mostly to model-free, reinforcement learning (RL) techniques such as Q-learning (Glorennec, 2000; Horiuchi et al., 1996; Jouffe, 1998) and actor-critic algorithms (Berenji and Khedkar, 1992; Berenji and Vengerov, 2003; Vengerov et al., 2005; Lin, 2003). Two desirable properties of the algorithms for approximate value iteration are the convergence to a near-optimal solution and consistency, which means asymptotical convergence to the optimal solution as the approximation accuracy increases. Unfortunately, most of the algorithms exploiting fuzzy representations are heuristic in nature, and their theoretical properties have not been investigated yet.

This chapter proposes a model-based, dynamic programming (DP) algorithm that represents Q-functions using a Takagi-Sugeno fuzzy rule base (Takagi and Sugeno, 1985). The rule

base receives the state as input, and produces the Q-values of the discrete actions as outputs. The discrete actions are selected beforehand from the (possibly continuous) original action space. The proposed algorithm is called *fuzzy Q-iteration*, because it combines the classical Q-iteration algorithm with a fuzzy approximator. The fuzzy sets in the rule base are described by membership functions (MFs), which can be identified with state-dependent basis functions (BFs). So, the fuzzy approximator belongs to the class of BF-dependent representations used throughout this thesis. We show that fuzzy Q-iteration asymptotically converges to an approximate Q-function that lies within a bounded distance from the optimal Q-function. A bound on the suboptimality of the solution obtained in a finite number of iterations is also derived. We show that fuzzy Q-iteration is consistent: under appropriate continuity assumptions on the process dynamics and on the reward function, the approximate Q-function converges to the optimal one as the approximation accuracy increases. These properties hold both when the parameters of the approximator are updated in a synchronous fashion, and when they are updated asynchronously. Additionally, we show that the asynchronous algorithm converges at least as fast as the synchronous one.

Fuzzy Q-iteration requires the MFs to be designed beforehand. Either prior knowledge about the optimal Q-function is required to design good MFs, or many MFs have to be defined to provide a good resolution over the entire state space. As an alternative, we propose to optimize the parameters of a constant number of MFs. This is useful when prior knowledge is not available and the number of MFs is limited. The proposed approach belongs to the class of approximator optimization techniques discussed in Section 3.5.2. To evaluate each set of MFs, a policy is computed with fuzzy Q-iteration using these MFs, and then the performance of this policy is estimated by simulation. Using the cross-entropy method for optimization, we design an algorithm to optimize the locations of triangular MFs.

We perform a thorough numerical and experimental evaluation of fuzzy Q-iteration. First, the convergence rates of synchronous and asynchronous fuzzy Q-iteration are compared for the optimal control of a (simulated) servo-system. Using the same system, we investigate the practical impact of reward function discontinuities on the consistency of the algorithm, and compare fuzzy Q-iteration to Q-iteration with radial basis function approximation (Tsitsiklis and Van Roy, 1996). Then, fuzzy Q-iteration is applied to stabilize a (simulated) two-link manipulator, and to swing up and stabilize a real-life inverted pendulum. For these three examples, the MFs are designed in advance. Finally, the performance of fuzzy Q-iteration with optimized MFs is evaluated on the classical car-on-the-hill DP/RL benchmark (Moore and Atkeson, 1995; Munos and Moore, 2002; Ernst et al., 2005).

The remainder of this chapter is organized as follows. Section 4.2 gives a brief overview of the literature related to our results. Section 4.3 describes fuzzy Q-iteration, and Section 4.4 analyzes its convergence and consistency. Section 4.5 describes our approach to optimize the MFs using the cross-entropy method. Section 4.6 gives an extensive numerical and experimental evaluation of the algorithm, and compares it to Q-iteration with radial basis function approximation. Finally, Section 4.7 concludes the chapter.

Parts of this chapter have been published in (Buşoniu et al., 2007b,a, 2008c,b,d).

4.2 Related work

This section gives a brief overview of the literature related to fuzzy Q-iteration and MF optimization. Fuzzy approximators have typically been used in model-free (RL) techniques such

as Q-learning (Glorennec, 2000; Horiuchi et al., 1996; Jouffe, 1998) and actor-critic algorithms (Berenji and Khedkar, 1992; Berenji and Vengerov, 2003; Vengerov et al., 2005; Lin, 2003). Most of these approaches are heuristic in nature, and their theoretical properties have not been investigated. Notable exceptions are the provably convergent actor-critic algorithms of Berenji and Vengerov (2003); Vengerov et al. (2005), which use the actor-critic convergence result reviewed in Section 3.4.4. In this chapter, we use fuzzy approximation with a *model-based* (DP) algorithm, and provide a detailed theoretical and empirical analysis of its convergence and consistency properties.

A rich body of literature concerns the theoretical analysis of approximate value iteration, both in the DP (model-based) setting (Chow and Tsitsiklis, 1991; Gordon, 1995; Tsitsiklis and Van Roy, 1996; Santos and Vigo-Aguiar, 1998; Munos and Szepesvári, 2008; Wiering, 2004) and in the RL (model-free) setting (Szepesvári and Smart, 2004; Singh et al., 1995; Ernst, 2003). In most cases, convergence is ensured by using approximators that are linear in the parameters (Gordon, 1995; Tsitsiklis and Van Roy, 1996; Szepesvári and Smart, 2004). The fuzzy approximator considered in this chapter is also linear in the parameters. Our convergence analysis for synchronous fuzzy Q-iteration is related to the convergence analysis of approximate V-iteration for discrete state-action spaces in (Gordon, 1995; Tsitsiklis and Van Roy, 1996). We additionally consider continuous state-action spaces, and introduce an explicit discretization procedure of the continuous action space. We also consider asynchronous fuzzy Q-iteration. While (exact) asynchronous value iteration is widely used (Bertsekas, 2007), asynchronous algorithms for approximate DP are not often studied. Many consistency results for model-based (DP) algorithms are found for discretization-based approximators (Gonzalez and Rofman, 1985; Chow and Tsitsiklis, 1991; Santos and Vigo-Aguiar, 1998). Such discretizations sometimes use interpolation schemes similar to those employed by fuzzy Q-iteration.

Techniques that change the number, position, and shape of the BFs have also been investigated (Munos and Moore, 2002; Ratitch and Precup, 2004; Menache et al., 2005; Ernst et al., 2005; Mahadevan, 2005; Keller et al., 2006). In some approaches, the BFs are adapted simultaneously with estimating a DP/RL solution, which leads to a loss of the convergence guarantees. In our approach, because each policy is computed by running fuzzy Q-iteration to convergence with fixed MF parameters, these convergence problems do not appear. The work that is most closely related to ours is that of Menache et al. (2005), who optimized the location and shape of a constant number of BFs for policy evaluation. The optimization criterion used by Menache et al. (2005) was the quadratic Bellman error of the value function. In principle, minimizing the Bellman error is useful, because a small infinity norm of the Bellman error leads to a good approximation of the Q-function (Williams and Baird, 1994). However, minimizing the *quadratic* Bellman error, as done by Menache et al. (2005), may lead to a large *infinity norm* of the Bellman error, and therefore does not necessarily lead to a good approximate Q-function. Our optimization approach does not suffer from such problems, because we maximize the empirical return of the computed policy, which directly expresses the control performance.

4.3 Fuzzy Q-iteration

In this section, the fuzzy Q-iteration algorithm is introduced. The state space X and the action space U of the problem may be either continuous or discrete, but they are assumed to

be subsets of Euclidean spaces, such that the 2-norm of the states and actions is well-defined. For the simplicity of notation, X and U are also assumed to be closed sets.

The proposed approximation scheme relies on a fuzzy partition of the state space, and on a discretization of the action space. The fuzzy partition contains N fuzzy sets χ_i, each described by a membership function (MF) $\mu_i : X \to [0, 1]$, $i = 1, \ldots, N$. A state x belongs to each set i with a degree of membership $\mu_i(x)$. The following requirement is imposed.

Requirement 4.1 *Each MF has its unique maximum in a single point, i.e., for every i there exists a unique x_i for which $\mu_i(x_i) > \mu_i(x)\, \forall x \neq x_i$. Additionally, $\mu_{i'}(x_i) = 0$ for all $i' \neq i$.*

This requirement is imposed here for brevity in the description and analysis of the algorithms; it can be relaxed using results of (Tsitsiklis and Van Roy, 1996). Because all the other MFs take zero values in x_i, it can be assumed without loss of generality that $\mu_i(x_i) = 1$. The state x_i is then called the core of the ith set.

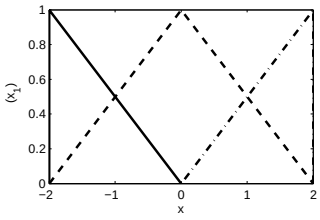
Example 4.1 *Triangular fuzzy partitions.* A simple type of fuzzy partition that satisfies Requirement 4.1 can be obtained as follows. For each state variable x_d with $d = 1, \ldots, D$, a number N_d of triangular MFs are defined as follows:

$$\phi_{d,1}(x_d) = \max\left(0, \frac{c_{d,2} - x_d}{c_{d,2} - c_{d,1}}\right)$$

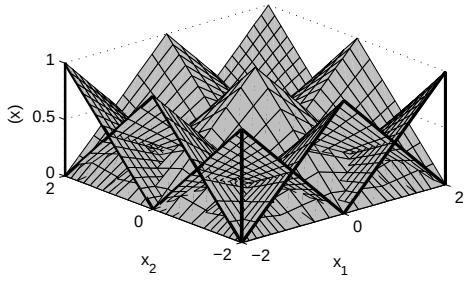
$$\phi_{d,i}(x_d) = \max\left[0, \min\left(\frac{x_d - c_{d,i-1}}{c_{d,i} - c_{d,i-1}}, \frac{c_{d,i+1} - x_d}{c_{d,i+1} - c_{d,i}}\right)\right] \qquad \text{for } i = 2, \ldots, N_d - 1$$

$$\phi_{d,N_d}(x_d) = \max\left(0, \frac{x_d - c_{d,N_d-1}}{c_{d,N_d} - c_{d,N_d-1}}\right)$$

where $c_{d,1} < \cdots < c_{d,N_d}$ is the array of cores along dimension d, which completely determines the shape of the MFs, and $x_d \in [c_{d,1}, c_{d,N_d}]$. Adjacent functions always intersect at a 0.5 level. Then, the product of each combination of (single-dimensional) MFs gives a pyramid-shaped D-dimensional MF in the fuzzy partition of X. Examples of one-dimensional and two-dimensional triangular partitions are given in Figure 4.1.



(a) A set of one-dimensional triangular MFs, each plotted in a different line style.



(b) Two-dimensional MFs, obtained by combining two sets of single-dimensional MFs, each identical to the set in Figure 4.1(a).

Figure 4.1: Examples of triangular MFs.

Note that beyond five or six dimensions, this type of product-space triangular MFs cannot be used effectively in fuzzy Q-iteration due to the curse of dimensionality, i.e., the exponential increase in memory and computation with the number of dimensions incurred by this type of MFs. □

A discrete set of actions U_d is chosen from the possibly larger (continuous or discrete) action space U:

$$U_\mathrm{d} = \{u_j | u_j \in U, j = 1, \ldots, M\} \tag{4.1}$$

The fuzzy approximator stores a parameter vector θ with $n = NM$ elements. Each pair (μ_i, u_j) of a MF and a discrete action is associated to one parameter $\theta_{[i,j]}$. The notation $[i, j]$ represents the single-dimensional index corresponding to i and j, which can be computed with $[i, j] = i + (j - 1)N$.

The Q-function is approximated using a multiple-input, multiple-output Takagi-Sugeno rule base with singleton consequents. The rule base takes the (D-dimensional) state x as input, and produces M outputs $q_1, \ldots, q_M$, which are the Q-values corresponding to each of the discrete actions $u_1, \ldots, u_M$. The ith rule has the form:

$$R_i : \textbf{ if } x \textbf{ is } \chi_i \textbf{ then } q_1 = \theta_{[i,1]}; q_2 = \theta_{[i,2]}; \ldots ; q_M = \theta_{[i,M]}$$

To compute the Q-value of the pair (x, u), the output q_j corresponding to the discrete action u_j that is closest to the value u is selected. The entire procedure can be written concisely as the following approximation mapping:

$$\widehat{Q}(x, u) = [F(\theta)](x, u) = \sum_{i=1}^{N} \phi_i(x)\theta_{[i,j]} \qquad \text{with } j = \arg\min_{j'} \|u - u_{j'}\|_2 \tag{4.2}$$

where $\| \cdot \|_2$, here as well as in the sequel, is the Euclidean norm of the argument, and ϕ_i are the normalized MFs (degrees of fulfillment):[1]

$$\phi_i(x) = \frac{\mu_i(x)}{\sum_{i'=1}^{N} \mu_{i'}(x)}$$

Equation (4.2) describes a linearly parameterized approximator, a special case of (3.2). To ensure that $F(\theta)$ is a well-defined function for any θ, the ties in the minimization in (4.2) have to be broken consistently (e.g., always in favor of the smallest index that satisfies the condition). For a fixed x, the approximator is constant over the sets of actions $U_j = \{u \in U \mid \|u - u_j\|_2 < \|u - u_{j'}\|_2 \, \forall j' \neq j \}$, for every j. The set U_j is the interior of the Voronoi cell for u_j (a convex polytope).

In fuzzy Q-iteration, the Takagi-Sugeno fuzzy rule base is used to approximate the Q-function, and the MFs do not necessarily have to be associated with meaningful linguistic labels. Nevertheless, if expert knowledge is available on the shape of the optimal Q-function, then MFs with meaningful linguistic labels can be defined by the expert. However, such knowledge is typically very difficult to obtain without actually computing the optimal Q-function.

[1] The MFs are already assumed normal, which means that each MF takes the value 1 for at least one point. However, they may not yet be *normalized*; the sum of normalized MFs is 1 for any value of x.

The projection mapping of fuzzy Q-iteration infers from a Q-function the values of the approximator parameters according to the relation:

$$\theta_{[i,j]} = [P(Q)]_{[i,j]} = Q(x_i, u_j) \tag{4.3}$$

This is the solution θ to the problem:

$$\sum_{i=1}^{N}\sum_{j=1}^{M}\big([F(\theta)](x_i,u_j) - Q(x_i,u_j)\big)^2 = 0$$

because $\phi_i(x_i) = 1$ and $\phi_{i'}(x_i) = 0$ for $i \neq i'$. The projection mapping (4.3) is a special case of the projection (3.9) introduced in Chapter 3.

The (synchronous) ***fuzzy Q-iteration*** is obtained by using the approximation mapping (4.2) and the projection mapping (4.3) in the approximate Q-iteration (3.3), repeated here for easy reference:

$$\theta_{\ell+1} = P \circ T \circ F(\theta_\ell) \tag{4.4}$$

The algorithms starts with an arbitrary $\theta_0 \in \mathbb{R}^n$ and stops when $\|\theta_{\ell+1} - \theta_\ell\|_\infty \leq \varepsilon_{\mathrm{QI}}$. Then, an approximately optimal parameter vector is $\widehat{\theta}^* = \theta_{\ell+1}$, and an approximately optimal policy can be computed with:

$$\widehat{\widehat{h}}^*(x) = u_{j^*}, \quad j^* = \arg\max_j [F(\widehat{\theta}^*)](x, u_j) \tag{4.5}$$

This policy is greedy in $F(\widehat{\theta}^*)$. Equation (4.5) is a special case of the approximately optimal policy given by (3.4). Of course, because the approximate Q-function is constant inside every Voronoi cell, any action in the j^*th cell could be used and the resulting policy would still satisfy (3.4). The notation $\widehat{\widehat{h}}^*$ is used to differentiate from the policy $\widehat{h}^*$ that is greedy in $F(\theta^*)$, where θ^* is the parameter vector obtained *asymptotically*, as $\ell \to \infty$ (see Section 4.4). The policy $\widehat{h}^*$ can be computed by substituting $\widehat{\theta}^*$ by θ^* and $\widehat{\widehat{h}}^*$ by $\widehat{h}^*$ in (4.5).

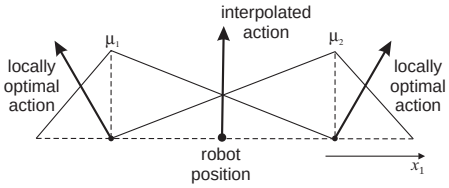
The policy (4.5) produces discrete actions. It is also possible to obtain a continuous-action policy using the following heuristic. For any state, an action is computed by interpolating between the best local actions for every MF core, using the MFs as weights:

$$h(x) = \sum_{i=1}^{N} \phi_i(x) u_{j_i^*}, \quad j_i^* = \arg\max_j [F(\widehat{\theta}^*)](x_i, u_j) = \arg\max_j \widehat{\theta}^*_{[i,j]} \tag{4.6}$$

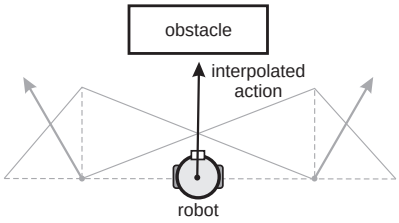
The index j_i^* corresponds to the best local action for the core x_i. For instance, when used with triangular MFs (Example 4.1), the interpolation procedure (4.6) is well suited to problems where (near-)optimal policies are affine or piecewise affine. Interpolated policies may, however, be a poor choice for other problems,[2] as illustrated in the next example.

Example 4.2 *Interpolated policies may perform poorly.* Consider the problem schematically represented in Figure 4.2. In this problem, a robot has to avoid an obstacle. The action of steering left is locally optimal for the MF core to the left of the robot's position. Similarly,

[2]Note that the approximation power of the fuzzy approximator is limited, because the number of MFs is fixed.



(a) MFs and actions.



(b) Robot and obstacle, with MFs and locally optimal actions repeated in gray color.

Figure 4.2: An obstacle avoidance problem, where interpolation between two locally optimal actions leads to incorrect behavior.

the action of steering right is locally optimal for the MF core to the right. These actions are locally optimal because they would take the robot around the obstacle, if the robot were located in the respective cores of the MFs. However, interpolating between these two actions at the robot's current position makes the robot (incorrectly) move forward and collide with the obstacle. In this case, rather than interpolating, a good policy would apply either of the two locally optimal actions, e.g., by randomly picking one of them. □

Note that our theoretical analysis of Section 4.4 only considers discrete-action policies of the form (4.5). Meaningful theoretical guarantees about policies of the form (4.6) are more difficult to provide.

Because all the approximate Q-functions considered by fuzzy Q-iteration are constant inside every Voronoi cell, it suffices to consider only the discrete actions when computing the maximal Q-values in the Q-iteration mapping. This discrete-action version of the Q-iteration mapping is defined as follows:

$$[T_{\mathrm{d}}(Q)](x,u) = \rho(x,u) + \gamma \max_j Q(f(x,u),u_j) \tag{4.7}$$

This property is very useful in the practical implementation of fuzzy Q-iteration. Namely, by using the fact that $T \circ F(\theta) = T_{\mathrm{d}} \circ F(\theta)$ for any θ, we get $P \circ T \circ F(\theta) = P \circ T_{\mathrm{d}} \circ F(\theta)$ for any θ, and each approximate Q-iteration (3.3) can be implemented as:

$$\theta_{\ell+1} = P \circ T_{\mathrm{d}} \circ F(\theta_\ell) \tag{4.8}$$

The maximization over U in the original T mapping can be replaced with a maximization over the discrete set U_{d} of (4.1), which can be solved using enumeration for moderate M. Furthermore, the norms in (4.2) do not need to be computed to implement (4.8).

The Q-iteration algorithm using (4.8) can be written as Algorithm 4.1. To establish the equivalence between Algorithm 4.1 and fuzzy Q-iteration in the form (4.8), observe that the right-hand side in line 4 of Algorithm 4.1 corresponds to $[T_{\mathrm{d}}(\widehat{Q}_\ell)](x_i,u_j)$, where $\widehat{Q}_\ell = F(\theta_\ell)$. Hence, line 4 can be written $\theta_{\ell+1,[i,j]} \leftarrow [P \circ T_{\mathrm{d}} \circ F(\theta_\ell)]_{[i,j]}$ and the entire **for** loop described by lines 3–5 is equivalent to (4.8).

Algorithm 4.2 is a different version of fuzzy Q-iteration that makes more efficient use of the parameter updates, by using the latest updated values of the parameters at each step

Algorithm 4.1 Synchronous fuzzy Q-iteration

Input:
dynamics f, reward function ρ, discount factor γ, threshold ε_{QI}
MFs ϕ_i, $i = 1, \ldots, N$; action discretization U_{d}

1: $\theta_0 \leftarrow 0_{NM}$
2: **repeat** in every iteration $\ell = 0, 1, 2, \ldots$
3: **for** $i = 1, \ldots, N, j = 1, \ldots, M$ **do**
4: $\theta_{\ell+1,[i,j]} \leftarrow \rho(x_i, u_j) + \gamma \max_{j'} \sum_{i'=1}^{N} \phi_{i'}(f(x_i, u_j))\theta_{\ell,[i',j']}$
5: **end for**
6: **until** $\|\theta_{\ell+1} - \theta_\ell\|_\infty \leq \varepsilon_{\text{QI}}$

Output: $\widehat{\theta}^* = \theta_{\ell+1}$

Algorithm 4.2 Asynchronous fuzzy Q-iteration

Input:
dynamics f, reward function ρ, discount factor γ, threshold ε_{QI}
MFs ϕ_i, $i = 1, \ldots, N$; action discretization U_{d}

1: $\theta_0 \leftarrow 0_{NM}$
2: **repeat** in every iteration $\ell = 0, 1, 2, \ldots$
3: $\theta \leftarrow \theta_\ell$
4: **for** $i = 1, \ldots, N, j = 1, \ldots, M$ **do**
5: $\theta_{[i,j]} \leftarrow \rho(x_i, u_j) + \gamma \max_{j'} \sum_{i'=1}^{N} \phi_{i'}(f(x_i, u_j))\theta_{[i',j']}$
6: **end for**
7: $\theta_{\ell+1} \leftarrow \theta$
8: **until** $\|\theta_{\ell+1} - \theta_\ell\|_\infty \leq \varepsilon_{\text{QI}}$

Output: $\widehat{\theta}^* = \theta_{\ell+1}$

of the computation. Since the parameters are updated in an asynchronous fashion, this version is called *asynchronous fuzzy Q-iteration*. To differentiate between the two versions, Algorithm 4.1 is hereafter called *synchronous fuzzy Q-iteration*. Although the exact version of asynchronous Q-iteration is widely used (Bertsekas, 2007; Sutton and Barto, 1998), the approximate variant has received little attention in the context of approximate DP/RL. In Algorithm 4.2 parameters are updated in sequence, but they can actually be updated in any order and our analysis still holds.

In this chapter, fuzzy Q-iteration is described and analyzed for deterministic problems only. Nevertheless, it can be extended to stochastic problems. For instance, the asynchronous update in line 5 of Algorithm 4.2 becomes in the stochastic case:

$$\theta_{[i,j]} \leftarrow \mathrm{E}_{x' \sim \tilde{f}(x_i, u_j, \cdot)} \left\{ \tilde{\rho}(x_i, u_j, x') + \gamma \max_{j'} \sum\nolimits_{i'=1}^{N} \phi_{i'}(x')\theta_{[i',j']} \right\}$$

where x' is sampled from the density function $\tilde{f}(x_i, u_j, \cdot)$ of the next state given x_i and u_j, and the dot stands for the random variable x'. If this expectation can be computed exactly, e.g., if there is a finite number of possible successor states, our results of Section 4.4 apply. In general, Monte Carlo estimation can be used to compute the expectation. Asymptotically, as the number of samples goes to infinity, the estimate converges to the true expectation and our

results can again be applied. When the number of samples is finite, Munos and Szepesvári (2008) provide error bounds for the related algorithm for approximate V-iteration. These bounds could be extended to approximate Q-iteration. Such bounds are important in practice, where the number of samples used is always finite.

4.4 Analysis of fuzzy Q-iteration

In Section 4.4.1, we show that synchronous and asynchronous fuzzy Q-iteration are convergent, and we characterize the suboptimality of their solution. In Section 4.4.2, fuzzy Q-iteration is shown to be consistent, which means that its solution asymptotically converges to Q^* as the approximation accuracy increases. These results show that fuzzy Q-iteration is a theoretically sound algorithm.

4.4.1 Convergence

First, it is shown that there exists a parameter vector θ^* such that for both synchronous and asynchronous fuzzy Q-iteration, $\theta_\ell \to \theta^*$ as $\ell \to \infty$. To prove convergence, we use the framework of non-expansive approximators described in Section 3.3.3. The parameter vector θ^* yields an approximate Q-function that is within a bound of the optimal Q-function, and a policy with a bounded suboptimality. Similar bounds are derived for the solution $\widehat{\theta}^*$ obtained in a finite number of iterations, by using a convergence threshold $\varepsilon_{\text{QI}} > 0$. Additionally, asynchronous fuzzy Q-iteration is shown to converge at least as fast as the synchronous version.

Theorem 4.2 (Convergence of synchronous fuzzy Q-iteration) *Synchronous fuzzy Q-iteration (Algorithm 4.1) converges.*

Proof: We show that $P \circ T \circ F$ is a contraction in the infinity norm with factor $\gamma < 1$, i.e., $\|P \circ T \circ F(\theta) - P \circ T \circ F(\theta')\|_\infty \leq \gamma \|\theta - \theta'\|_\infty$, for any θ, θ'. It is well-known that T is a contraction with factor $\gamma < 1$ (Bertsekas, 2007), and it is obvious from (4.3) that P is a non-expansion. Also, F is a non-expansion:

$$
\begin{aligned}
&|[F(\theta)](x,u) - [F(\theta')](x,u)| \\
&= \left| \sum_{i=1}^{N} \phi_i(x)\theta_{[i,j]} - \sum_{i=1}^{N} \phi_i(x)\theta'_{[i,j]} \right| \qquad \text{(where } j = \arg\min_{j'} \|u - u_{j'}\|_2\text{)} \\
&\leq \sum_{i=1}^{N} \phi_i(x) \left| \theta_{[i,j]} - \theta'_{[i,j]} \right| \\
&\leq \sum_{i=1}^{N} \phi_i(x) \|\theta - \theta'\|_\infty = \|\theta - \theta'\|_\infty
\end{aligned}
$$

Therefore, $P \circ T \circ F$ is a contraction with factor $\gamma < 1$. It follows that $P \circ T \circ F$ has a unique fixed point θ^*, and so synchronous fuzzy Q-iteration converges to θ^* as $\ell \to \infty$. □

This proof is similar to the proof of convergence for (synchronous) value iteration with averagers (Gordon, 1995), or with interpolative representations (Tsitsiklis and Van Roy, 1996),

where the approximation and projection mappings were also non-expansions. The fuzzy approximator can be seen as an extension of an averager (Gordon, 1995) or of an interpolative representation (Tsitsiklis and Van Roy, 1996) for approximating Q-value functions, additionally using discretization to deal with large or continuous action spaces.

In the sequel, a synthetic notation of asynchronous fuzzy Q-iteration is needed. To develop this notation, recall first that $n = NM$, and that $[i, j] = i + (j - 1)N$. Define for all $l = 0, \ldots, n$ recursively the mappings $S_l : \mathbb{R}^n \to \mathbb{R}^n$ as:

$$S_0(\theta) = \theta$$
$$[S_l(\theta)]_{l'} = \begin{cases} [P \circ T \circ F(S_{l-1}(\theta))]_{l'} & \text{if } l' = l \\ [S_{l-1}(\theta)]_{l'} & \text{if } l' \in \{1, \ldots, n\} \setminus \{l\} \end{cases}$$

So, S_l for $l > 0$ corresponds to updating the first l parameters using approximate asynchronous Q-iteration, and S_n is a complete iteration of the asynchronous algorithm.

Theorem 4.3 (Convergence of asynchronous fuzzy Q-iteration) *Asynchronous fuzzy Q-iteration (Algorithm 4.2) converges.*

Proof: We show that S_n is a contraction, i.e., $\|S_n(\theta) - S_n(\theta')\|_\infty \leq \gamma \|\theta - \theta'\|_\infty$, for any θ, θ'. This can be done element by element. By the definition of S_l, the first element is only updated by S_1:

$$\begin{aligned} |[S_n(\theta)]_1 - [S_n(\theta')]_1| &= |[S_1(\theta)]_1 - [S_1(\theta')]_1| \\ &= |[P \circ T \circ F(\theta)]_1 - [P \circ T \circ F(\theta')]_1| \\ &\leq \gamma \|\theta - \theta'\|_\infty \end{aligned}$$

The last step follows from the contraction mapping property of $P \circ T \circ F$. Similarly, the second element is only updated by S_2:

$$\begin{aligned} |[S_n(\theta)]_2 - [S_n(\theta')]_2| &= |[S_2(\theta)]_2 - [S_2(\theta')]_2| \\ &= |[P \circ T \circ F(S_1(\theta))]_2 - [P \circ T \circ F(S_1(\theta'))]_2| \\ &\leq \gamma \|S_1(\theta) - S_1(\theta')\|_\infty \\ &= \gamma \max\{|[P \circ T \circ F(\theta)]_1 - [P \circ T \circ F(\theta')]_1|, |\theta_2 - \theta_2'|, \ldots, |\theta_n - \theta_n'|\} \\ &\leq \gamma \|\theta - \theta'\|_\infty \end{aligned}$$

where the contraction mapping property of $P \circ T \circ F$ is used twice. Continuing in this fashion, we obtain $|[S_n(\theta)]_l - [S_n(\theta')]_l| \leq \gamma \|\theta - \theta'\|_\infty$ for all l, and thus S_n is a contraction. Therefore, asynchronous fuzzy Q-iteration converges. This convergence proof is similar to that for *exact* asynchronous V-iteration (Bertsekas, 2007). □

This proof is actually more general, showing that approximate asynchronous Q-iteration converges for any approximation mapping F and projection mapping P for which $P \circ T \circ F$ is a contraction. It can also be easily shown that synchronous and asynchronous fuzzy Q-iteration converge to the same parameter vector; indeed, the repeated application of any contraction mapping will converge to its unique fixed point regardless of whether it is applied in a synchronous or asynchronous (element-by-element) fashion.

We now show that asynchronous fuzzy Q-iteration converges at least as fast as the synchronous version. For that, we first need the following monotonicity lemma. In this lemma,

as well as in the sequel, vector and vector function inequalities are understood to be satisfied element-wise.

Lemma 4.4 (Monotonicity) *If $\theta \leq \theta'$, then $P \circ T \circ F(\theta) \leq P \circ T \circ F(\theta')$.*

Proof: It is well-known that T is monotonous, i.e., $T(Q) \leq T(Q')$ if $Q \leq Q'$ (Bertsekas, 2007). It will be shown that F and P are monotonous.

To show that F is monotonous we show that, given $\theta \leq \theta'$, it follows that for all x, u:

$$[F(\theta)](x,u) \leq [F(\theta')](x,u)$$

This inequality is true because it is equivalent to:

$$\sum_{i=1}^{N} \phi_i(x)\theta_{[i,j]} \leq \sum_{i=1}^{N} \phi_i(x)\theta'_{[i,j]}$$

where $j = \arg\min_{j'} \|u - u_{j'}\|_2$. The last inequality is true by the assumption $\theta \leq \theta'$.

To show that P is monotonous we show that, given $Q \leq Q'$, it follows that for all i, j:

$$[P(Q)]_{[i,j]} \leq [P(Q')]_{[i,j]}$$

This inequality is true because it is equivalent to $Q(x_i, u_j) \leq Q'(x_i, u_j)$, which is true by the assumption $Q \leq Q'$. Therefore, the composite mapping $P \circ T \circ F$ is monotonous. □

Asynchronous fuzzy Q-iteration converges at least as fast as the synchronous algorithm, in the sense that ℓ iterations of the asynchronous algorithm move the parameter vector at least as close to the convergence point as ℓ iterations of the synchronous algorithm. This is stated formally as follows.

Proposition 4.5 (Convergence rate) *If a parameter vector θ satisfies $\theta \leq P \circ T \circ F(\theta) \leq \theta^*$, then:*

$$(P \circ T \circ F)^{\ell}(\theta) \leq S_n^{\ell}(\theta) \leq \theta^* \quad \forall \ell \geq 1$$

where by $S_n^{\ell}(\theta)$ we denote the composition of $S_n(\theta)$ ℓ-times with itself, i.e., $S_n^{\ell}(\theta) = S_n \circ S_n \circ \cdots \circ S_n(\theta)$, and similarly for $(P \circ T \circ F)^{\ell}(\theta)$.

Proof: This follows from the monotonicity of $P \circ T \circ F$, and can be shown element-wise, in a similar fashion to the proof of Theorem 4.3. Note that this result is an extension of Bertsekas' result on exact V-iteration (Bertsekas, 2007). □

In the remainder of this section, in addition to examining the asymptotical properties of fuzzy Q-iteration, we also consider an implementation that stops when $\|\theta_{\ell+1} - \theta_{\ell}\|_{\infty} \leq \varepsilon_{\mathrm{QI}}$, with a convergence threshold $\varepsilon_{\mathrm{QI}} > 0$. This implementation returns the solution $\widehat{\theta}^* = \theta_{\ell+1}$. Such an implementation was given in Algorithm 4.1 for synchronous fuzzy Q-iteration, and in Algorithm 4.2 for asynchronous fuzzy Q-iteration.

Proposition 4.6 (Finite-time termination) *For any choice of the threshold $\varepsilon_{\mathrm{QI}} > 0$ and any initial parameter vector $\theta_0 \in \mathbb{R}^n$, synchronous and asynchronous fuzzy Q-iteration stop in a finite time.*

Proof: Consider synchronous fuzzy Q-iteration. Because the mapping $P \circ T \circ F$ is a contraction with factor $\gamma < 1$ and with the fixed point θ^*, we have:

$$\|\theta_{\ell+1} - \theta^*\|_\infty = \|P \circ T \circ F(\theta_\ell) - P \circ T \circ F(\theta^*)\|_\infty \leq \gamma \|\theta_\ell - \theta^*\|_\infty$$

By induction, $\|\theta_\ell - \theta^*\|_\infty \leq \gamma^\ell \|\theta_0 - \theta^*\|_\infty$ for any $\ell > 0$. By the Banach fixed point theorem (see e.g., Kirk and Khamsi, 2001), the optimal parameter vector θ^* is bounded. Because the initial parameter vector θ_0 is bounded, $\|\theta_0 - \theta^*\|_\infty$ is bounded. Using the notation $B_0 = \|\theta_0 - \theta^*\|_\infty$, we have that B_0 is bounded and that $\|\theta_\ell - \theta^*\|_\infty \leq \gamma^\ell B_0$ for any $\ell > 0$. Using this property, we have:

$$\begin{aligned}\|\theta_{\ell+1} - \theta_\ell\|_\infty &\leq \|\theta_{\ell+1} - \theta^*\|_\infty + \|\theta_\ell - \theta^*\|_\infty \\ &\leq \gamma^\ell(\gamma + 1)B_0\end{aligned}$$

Using this inequality, it is possible to choose for any $\varepsilon_{\text{QI}} > 0$ a number of iterations $L = \left\lceil \log_\gamma \frac{\varepsilon_{\text{QI}}}{(\gamma+1)B_0} \right\rceil$ that guarantees that $\|\theta_{L+1} - \theta_L\|_\infty \leq \varepsilon_{\text{QI}}$, and therefore that the algorithm stops in at most L iterations. Here, $\lceil \cdot \rceil$ gives the smallest integer larger than or equal to the argument (ceiling). Because B_0 is bounded, L is finite. Since the computational cost of every iteration is finite for finite N and M, synchronous fuzzy Q-iteration stops in a finite time.

The proof for asynchronous fuzzy Q-iteration proceeds in exactly the same way, because the asynchronous Q-iteration mapping S_n is also a contraction with factor $\gamma < 1$ and with the fixed point θ^*. □

The following bounds on the suboptimality of the resulting approximate Q-function and policy hold.

Theorem 4.7 (Near-optimality) *Denote by $\mathcal{F}_{F \circ P} \subset \mathcal{Q}$ the set of fixed points of the mapping $F \circ P$, and define $\varepsilon'_Q = \min_{Q' \in \mathcal{F}_{F \circ P}} \|Q^* - Q'\|_\infty$, the minimum distance between Q^* and any fixed point of $F \circ P$. The convergence point θ^* of asynchronous and synchronous fuzzy Q-iteration satisfies:*

$$\|Q^* - F(\theta^*)\|_\infty \leq \frac{2\varepsilon'_Q}{1 - \gamma} \tag{4.9}$$

Additionally, the parameter vector $\widehat{\theta}^*$ obtained by asynchronous or synchronous fuzzy Q-iteration using a threshold ε_{QI} satisfies:

$$\|Q^* - F(\widehat{\theta}^*)\|_\infty \leq \frac{2\varepsilon'_Q + \gamma\varepsilon_{\text{QI}}}{1 - \gamma} \tag{4.10}$$

Furthermore, if $U_{\text{d}} = U$ in (4.1)*, which means that the original action space is discrete and that all the discrete values are used in the fuzzy Q-iteration algorithm, then:*

$$\|Q^* - Q^{\widehat{h}^*}\|_\infty \leq \frac{4\gamma\varepsilon'_Q}{(1 - \gamma)^2} \tag{4.11}$$

$$\|Q^* - Q^{\widehat{\widehat{h}}^*}\|_\infty \leq \frac{2\gamma(2\varepsilon'_Q + \gamma\varepsilon_{\text{QI}})}{(1 - \gamma)^2} \tag{4.12}$$

where $Q^{\widehat{h}^*}$ is the Q-function of the policy $\widehat{h}^*$ that is greedy in $F(\theta^*)$, and $Q^{\widehat{\widehat{h}}^*}$ is the Q-function of the policy $\widehat{\widehat{h}}^*$ that is greedy in $F(\widehat{\theta}^*)$ (4.5)*.*

Proof: The proof proceeds in exactly the same way for synchronous and asynchronous fuzzy Q-iteration, because it only relies on the contracting nature of their updates.

The bound (4.9) can be proven in a similar way to the bound shown in (Gordon, 1995; Tsitsiklis and Van Roy, 1996) for V-iteration. It is an application of the result (3.5) of Section 3.3.

In order to obtain (4.10), a bound on $\|\widehat{\theta}^* - \theta^*\|_\infty$ is computed first. Let L be the number of iterations after which the algorithm stops, which is finite by Proposition 4.6. Therefore, $\widehat{\theta}^* = \theta_{L+1}$. We have:

$$\begin{aligned}\|\theta_L - \theta^*\|_\infty &\leq \|\theta_{L+1} - \theta_L\|_\infty + \|\theta_{L+1} - \theta^*\|_\infty \\ &\leq \varepsilon_{\text{QI}} + \gamma\|\theta_L - \theta^*\|_\infty\end{aligned}$$

where the last step follows from the convergence condition $\|\theta_{L+1} - \theta_L\|_\infty \leq \varepsilon_{\text{QI}}$ and from the contracting nature of the updates (see also the proof of Proposition 4.6). From the last inequality, it follows that $\|\theta_L - \theta^*\|_\infty \leq \frac{\varepsilon_{\text{QI}}}{1-\gamma}$ and therefore:

$$\|\theta_{L+1} - \theta^*\|_\infty \leq \gamma\|\theta_L - \theta^*\|_\infty \leq \frac{\gamma\varepsilon_{\text{QI}}}{1-\gamma}$$

which is equivalent to:

$$\|\widehat{\theta}^* - \theta^*\|_\infty \leq \frac{\gamma\varepsilon_{\text{QI}}}{1-\gamma} \tag{4.13}$$

Using this inequality, the suboptimality of the Q-function $F(\widehat{\theta}^*)$ can be bounded using:

$$\begin{aligned}\|Q^* - F(\widehat{\theta}^*)\|_\infty &\leq \|Q^* - F(\theta^*)\|_\infty + \|F(\theta^*) - F(\widehat{\theta}^*)\|_\infty \\ &\leq \|Q^* - F(\theta^*)\|_\infty + \|\widehat{\theta}^* - \theta^*\|_\infty \\ &\leq \frac{2\varepsilon'_Q}{1-\gamma} + \frac{\gamma\varepsilon_{\text{QI}}}{1-\gamma} = \frac{2\varepsilon'_Q + \gamma\varepsilon_{\text{QI}}}{1-\gamma}\end{aligned}$$

where the second step is true because F is a non-expansion (which was shown in the proof of Theorem 4.2), and the third step follows from (4.9) and (4.13). So, (4.10) has been obtained.

The bounds (4.11) and (4.12), which characterize the suboptimality of the policies resulting from θ^* and $\widehat{\theta}^*$, follow from (3.7). Recall that (3.7) relates the suboptimality of an arbitrary Q-function Q with the suboptimality of the policy h that is greedy in this Q-function:

$$\|Q^* - Q^h\|_\infty \leq \frac{2\gamma}{(1-\gamma)}\|Q^* - Q\|_\infty$$

To obtain (4.11) and (4.12), this inequality is applied to the Q-functions $F(\theta^*)$ and $F(\widehat{\theta}^*)$, using their suboptimality bounds (4.9) and (4.10). □

Examining (4.10) and (4.12), it can be seen that the suboptimality of the solution computed in finite time is given by a sum of two terms. The second term depends linearly on the precision ε_{QI} with which the solution is computed, and is easy to control by setting ε_{QI} as close to 0 as needed. The first term in the sum depends linearly on ε'_Q, which is in turn related to the accuracy of the fuzzy approximator. This term also contributes to the suboptimality of the asymptotic solutions, see (4.9) and (4.11). The distance ε'_Q is more difficult to control. An ideal situation is when the optimal Q-function is a fixed point of $F \circ P$, because in this case

$\varepsilon'_Q = 0$. For instance, any Q-function that satisfies $Q(x,u) = \sum_{i=1}^{N} \phi_i(x) Q(x_i, u_j)$ for all x, u, where $j = \arg\min_{j'} \|u - u_{j'}\|_2$, is a fixed point of $F \circ P$. If the optimal Q-function has this form, i.e., is exactly representable by the chosen fuzzy approximator, the algorithm will asymptotically converge to it, and (when $U = U_\mathrm{d}$) the corresponding policy will be optimal. Section 4.4.2 provides additional insight into the relationship between the suboptimality of the solution and the accuracy of the fuzzy approximator.

4.4.2 Consistency

Next, we analyze the consistency of synchronous and asynchronous fuzzy Q-iteration. It is shown that the approximate solution $F(\theta^*)$ asymptotically converges to the optimal Q-function Q^*, as the maximum distance between the cores of adjacent fuzzy sets and the maximum distance between adjacent discrete actions decrease to 0. An explicit relationship between the suboptimality of $F(\theta^*)$ and the accuracy of the approximator is also derived.

The state resolution step δ_x is defined as the largest distance between any point in the state space and the MF core that is closest to it. The action resolution step δ_u is defined similarly for the discrete actions. Formally:

$$\delta_x = \max_{x \in X} \min_{i=1,\ldots,N} \|x - x_i\|_2 \tag{4.14}$$

$$\delta_u = \max_{u \in U} \min_{j=1,\ldots,M} \|u - u_j\|_2 \tag{4.15}$$

where x_i is the core of the ith MF, and u_j is the jth discrete action. Smaller values of δ_x and δ_u indicate a higher resolution. The goal is to show that $\lim_{\delta_x \to 0,\ \delta_u \to 0} F(\theta^*) = Q^*$.

We assume that f and ρ are Lipschitz continuous.

Assumption 4.8 (Lipschitz continuity) *The dynamics f and the reward function ρ are Lipschitz continuous, i.e., there exist finite constants $L_f \geq 0$ and $L_\rho \geq 0$ such that:*

$$\begin{aligned} \|f(x,u) - f(\bar{x},\bar{u})\|_2 &\leq L_f(\|x - \bar{x}\|_2 + \|u - \bar{u}\|_2) \\ |\rho(x,u) - \rho(\bar{x},\bar{u})| &\leq L_\rho(\|x - \bar{x}\|_2 + \|u - \bar{u}\|_2) \\ &\forall x, \bar{x} \in X, u, \bar{u} \in U \end{aligned}$$

We also require that the MFs are Lipschitz continuous.

Requirement 4.9 *Every MF ϕ_i is Lipschitz continuous, i.e., for every i there exists a finite constant $L_{\phi_i} \geq 0$ such that:*

$$\|\phi_i(x) - \phi_i(\bar{x})\|_2 \leq L_{\phi_i} \|x - \bar{x}\|_2, \quad \forall x, \bar{x} \in X$$

Finally, the MFs should be local and evenly distributed, in the following sense.

Requirement 4.10 *Every MF ϕ_i has a bounded support, which is contained in a ball with a radius proportional to δ_x. Formally, there exists a finite $\nu > 0$ such that:*

$$\{x \mid \phi_i(x) > 0\} \subset \{x \mid \|x - x_i\|_2 \leq \nu \delta_x\}, \quad \forall i$$

Furthermore, for every x, only a finite number of MFs are non-zero. Formally, there exists a finite $\kappa > 0$ such that:

$$|\{i \mid \phi_i(x) > 0\}| \leq \kappa, \quad \forall x$$

where $|\cdot|$ denotes set cardinality.

Lipschitz continuity conditions such as those of Assumption 4.8 are typically needed to prove the consistency of algorithms for approximate DP. Requirement 4.9 is not restrictive. Requirement 4.10 is satisfied in many cases of interest. For instance, it is satisfied by convex fuzzy sets with their cores distributed on an (equidistant or irregular) rectangular grid in the state space, such as the triangular partitions of Example 4.1. In such cases, every point x falls inside a hyperbox defined by the two adjacent cores that are closest to x_d on each axis d. Some points will fall on the boundary of several hyperboxes, in which case we can just pick any of these hyperboxes. Given Requirement 4.1 and because the fuzzy sets are convex, only the MFs with the cores in the corners of the hyperbox can take non-zero values in the chosen point. The number of corners is 2^D where D is the dimension of X. Therefore, $|\{i \mid \phi_i(x) > 0\}| \leq 2^D$, and a choice $\kappa = 2^D$ satisfies the second part of Requirement 4.10. Furthermore, a MF will always extend at most two hyperboxes along each axis of the state space. By the definition of δ_x, the largest diagonal of any hyperbox is $2\delta_x$. Therefore, $\{x \mid \phi_i(x) > 0\} \subset \{x \mid \|x - x_i\|_2 \leq 4\delta_x\}$, and a choice $\nu = 4$ satisfies the first part of Requirement 4.10.

The next lemma bounds the approximation error introduced by every iteration of the synchronous fuzzy Q-iteration algorithm. Since we are ultimately interested in characterizing the convergence point θ^*, which is the same for both algorithms, the final consistency result (Theorem 4.12) applies to the asynchronous algorithm as well.

Lemma 4.11 (Bounded error) *There exists a finite $\varepsilon_\delta \geq 0$ such that any approximate Q-function $\widehat{Q}$ considered by the synchronous fuzzy Q-iteration algorithm satisfies:*

$$\|F \circ P \circ T(\widehat{Q}) - T(\widehat{Q})\|_\infty \leq \varepsilon_\delta$$

Furthermore, $\varepsilon_\delta = \mathrm{O}(\delta_x) + \mathrm{O}(\delta_u)$.

Proof: For any pair (x, u):

$$\begin{aligned}
&\left|[F \circ P \circ T(\widehat{Q})](x,u) - [T(\widehat{Q})](x,u)\right| \\
&= \left|\left(\sum_{i=1}^N \phi_i(x)[T(\widehat{Q})](x_i,u_j)\right) - [T(\widehat{Q})](x,u)\right| \qquad (\text{where } j = \arg\min_{j'} \|u - u_{j'}\|_2) \\
&= \Big|\left(\sum_{i=1}^N \phi_i(x)\left[\rho(x_i,u_j) + \gamma \max_{u'} \widehat{Q}(f(x_i,u_j),u')\right]\right) \\
&\qquad - \left[\rho(x,u) + \gamma \max_{u'} \widehat{Q}(f(x,u),u')\right]\Big| \\
&\leq \left|\left(\sum_{i=1}^N \phi_i(x)\rho(x_i,u_j)\right) - \rho(x,u)\right| \\
&\qquad + \gamma\left|\left(\sum_{i=1}^N \phi_i(x) \max_{u'} \widehat{Q}(f(x_i,u_j),u')\right) - \max_{u'} \widehat{Q}(f(x,u),u')\right| \qquad (4.16)
\end{aligned}$$

The first term on the right-hand side of (4.16) is equal to:

$$\begin{aligned}\left|\sum_{i=1}^{N}\phi_i(x)\left[\rho(x_i,u_j)-\rho(x,u)\right]\right| &\leq \sum_{i=1}^{N}\phi_i(x)L_\rho(\|x_i-x\|_2+\|u_j-u\|_2)\\ &= L_\rho\left[\|u_j-u\|_2+\sum_{i=1}^{N}\phi_i(x)\|x_i-x\|_2\right]\\ &\leq L_\rho(\delta_u+\kappa\nu\delta_x) \end{aligned} \tag{4.17}$$

where the Lipschitz continuity of ρ was used, and the last step follows from the definition of δ_u and Requirement 4.10.

The second term in the right-hand side of (4.16) is equal to:

$$\begin{aligned}&\gamma\left|\sum_{i=1}^{N}\phi_i(x)\left[\max_{u'}\widehat{Q}(f(x_i,u_j),u')-\max_{u'}\widehat{Q}(f(x,u),u')\right]\right|\\ &\leq\gamma\sum_{i=1}^{N}\phi_i(x)\left|\max_{j'}\widehat{Q}(f(x_i,u_j),u_{j'})-\max_{j'}\widehat{Q}(f(x,u),u_{j'})\right|\\ &\leq\gamma\sum_{i=1}^{N}\phi_i(x)\max_{j'}\left|\widehat{Q}(f(x_i,u_j),u_{j'})-\widehat{Q}(f(x,u),u_{j'})\right|\end{aligned} \tag{4.18}$$

The first step is true because $\widehat{Q}$ is of the form $F(\theta)$ given in (4.2) for some θ, and is therefore constant in the Voronoi cell of each discrete action. The second step is true because the difference between the maxima of two functions (of the same variable) is at most the maximum of the difference of the functions. Writing $\widehat{Q}$ explicitly as $F(\theta)$ in (4.2), we have:

$$\begin{aligned}&\left|\widehat{Q}(f(x_i,u_j),u_{j'})-\widehat{Q}(f(x,u),u_{j'})\right|\\ &=\left|\sum_{i'=1}^{N}\left[\phi_{i'}(f(x_i,u_j))\theta_{[i',j']}-\phi_{i'}(f(x,u))\theta_{[i',j']}\right]\right|\\ &\leq\sum_{i'=1}^{N}\left|\phi_{i'}(f(x_i,u_j))-\phi_{i'}(f(x,u))\right|\left|\theta_{[i',j']}\right|\end{aligned} \tag{4.19}$$

Define $I'=\{i'\mid\phi_{i'}(f(x_i,u_j))\neq 0 \text{ or } \phi_{i'}(f(x,u))\neq 0\}$. Using Requirement 4.10, $|I'|\leq 2\kappa$. Denote $L_\phi=\max_i L_{\phi_i}$. Then, the right-hand side of (4.19) is equal to:

$$\begin{aligned}&\sum_{i'\in I'}\left|\phi_{i'}(f(x_i,u_j))-\phi_{i'}(f(x,u))\right|\left|\theta_{[i',j']}\right|\\ &\leq\sum_{i'\in I'}L_\phi L_f(\|x_i-x\|_2+\|u_j-u\|_2)\|\theta\|_\infty\\ &\leq 2\kappa L_\phi L_f(\|x_i-x\|_2+\|u_j-u\|_2)\|\theta\|_\infty\end{aligned} \tag{4.20}$$

Using (4.19) and (4.20) into (4.18) yields:

$$\begin{aligned}
&\gamma \sum_{i=1}^{N} \phi_i(x) \max_{j'} \left| \widehat{Q}(f(x_i,u_j),u_{j'}) - \widehat{Q}(f(x,u),u_{j'}) \right| \\
&\leq \gamma \sum_{i=1}^{N} \phi_i(x) \max_{j'} 2\kappa L_\phi L_f (\|x_i - x\|_2 + \|u_j - u\|_2) \|\theta\|_\infty \\
&\leq 2\gamma\kappa L_\phi L_f \|\theta\|_\infty \left[\|u_j - u\|_2 + \sum_{i=1}^{N} \phi_i(x) \|x_i - x\|_2 \right] \\
&\leq 2\gamma\kappa L_\phi L_f \|\theta\|_\infty (\delta_u + \kappa\nu\delta_x) \qquad (4.21)
\end{aligned}$$

where the last step follows from the definition of δ_u and Requirement 4.10. Finally, substituting (4.17) and (4.21) into (4.16) yields:

$$\left| [F \circ P \circ T(\widehat{Q})](x,u) - [T(\widehat{Q})](x,u) \right| \leq (L_\rho + 2\gamma\kappa L_\phi L_f \|\theta\|_\infty)(\delta_u + \kappa\nu\delta_x) \qquad (4.22)$$

Given a bounded initial parameter vector θ_0, all the parameter vectors considered by the algorithm are bounded. This can be shown as follows. By the Banach fixed point theorem (see e.g., Kirk and Khamsi, 2001), the optimal parameter vector θ^* (the unique fixed point of $P \circ T \circ F$) is finite. Also, since $P \circ T \circ F$ is a contraction with factor $\gamma < 1$, we have $\|\theta_\ell - \theta^*\|_\infty \leq \gamma^\ell \|\theta_0 - \theta^*\|_\infty$ (see also the proof of Proposition 4.6). Since $\|\theta_0 - \theta^*\|_\infty$ is bounded, all the other distances are bounded, and all the parameter vectors θ_ℓ are bounded. Let $B_\theta = \max_{\ell \geq 0} \|\theta_\ell\|_\infty$, which is finite.

Therefore, $\|\theta\|_\infty \leq B_\theta$ in (4.22), and the proof is complete with $\varepsilon_\delta = (L_\rho + 2\gamma\kappa L_\phi L_f B_\theta)(\delta_u + \kappa\nu\delta_x) = \mathrm{O}(\delta_x) + \mathrm{O}(\delta_u)$. □

Theorem 4.12 (Consistency) *Under Assumption 4.8 and if Requirements 4.9 and 4.10 are satisfied, synchronous and asynchronous fuzzy Q-iteration are consistent, i.e.,* $\lim_{\delta_x \to 0, \delta_u \to 0} F(\theta^*) = Q^*$. *Furthermore, under the same conditions and for any resolution steps* δ_x *and* δ_u*, the suboptimality of the approximate Q-function satisfies* $\|F(\theta^*) - Q^*\|_\infty = \mathrm{O}(\delta_x) + \mathrm{O}(\delta_u)$.

Proof: It is easy to show by induction that $\|F(\theta^*) - Q^*\|_\infty \leq \varepsilon_\delta/(1-\gamma)$, see, e.g., (Bertsekas and Tsitsiklis, 1996). From Lemma 4.11, $\lim_{\delta_x \to 0, \delta_u \to 0} \varepsilon_\delta = \lim_{\delta_x \to 0, \delta_u \to 0} (L_\rho + 2\gamma\kappa L_\phi L_f B_\theta)(\delta_u + \kappa\nu\delta_x) = 0$, and the first part of the theorem is proven. Furthermore, using the same lemma, $\varepsilon_\delta = \mathrm{O}(\delta_x) + \mathrm{O}(\delta_u)$, which completes the proof. □

In addition to guaranteeing consistency, Theorem 4.12 also relates the suboptimality of the Q-function $F(\theta^*)$ with the accuracy of the fuzzy approximator. Furthermore, using Theorem 4.7, the accuracy can also be related with the suboptimality of the policy $\widehat{h}^*$ greedy in $F(\theta^*)$, and with the suboptimality of the solution obtained after a finite number of iterations (the Q-function $F(\widehat{\theta}^*)$ and the corresponding greedy policy $\widehat{\widehat{h}}^*$).

4.5 Optimizing the membership functions

Fuzzy Q-iteration requires the MFs to be designed beforehand. In general, prior knowledge about the shape of the optimal Q-function is needed in this design process. When prior

knowledge is not available, a large number of MFs have to be defined to provide a good coverage and resolution over the entire state space, even in areas that will eventually be irrelevant to the policy. In this section, we propose to optimize the shape of the MFs, while keeping their number constant. The proposed approach is useful when prior knowledge about the shape of the optimal Q-function is not available and the number of MFs is limited.

Let the MFs be parameterized by a vector $\xi \in \Xi$. Usually, ξ includes information about the location and shape of the MFs. Denote the MFs by $\phi_i(x;\xi) : X \to \mathbb{R}$, $i = 1, \ldots, N$, to highlight their dependence on ξ. The goal is to find a parameter vector ξ^* that maximizes the return from every initial state. Since generally it is not possible to compute returns from every initial state, a finite set X_0 of representative initial states has to be chosen, like in approximate policy search, see Section 3.6. The score function to maximize is:

$$s(\xi) = \sum_{x_0 \in X_0} w(x_0) R^h(x_0) \tag{4.23}$$

where the policy h is computed by running fuzzy Q-iteration to convergence with the MFs specified by the parameters ξ. The return from each initial state x_0 is approximated in a finite time, with a precision ε_{MC}, using (2.27). The set X_0 of representative states is weighted by $w : X_0 \to (0, 1]$. The set X_0, together with the weight function w, will determine the performance of the resulting policy. For instance, if the process only has to be controlled starting from a known set of initial states, then X_0 should be equal to this set, or included in it when the set is too large. Also, initial states that are deemed more important can be assigned larger weights. When all initial states are equally important, the elements of X_0 should be uniformly spread over the state space, randomly or equidistantly, and identical weights equal to $\frac{1}{|X_0|}$ should be assigned to every element of X_0 ($|\cdot|$ denotes set cardinality).

Because each policy is computed by running fuzzy Q-iteration to convergence with fixed MF parameters, this technique does not suffer from the convergence problems associated with altering the approximator simultaneously with estimating a DP/RL solution. Also note that the score function (4.23) is directly related to the performance of the computed policy in the task.

This technique is not restricted to a particular optimization algorithm to search for ξ^*. However, the score function (4.23) is a complicated function of ξ, and is in general likely to have many local optima. This means a global optimization technique is preferable. In Section 4.5.2, a complete algorithm is given to optimize the locations of triangular MFs for fuzzy Q-iteration, using the cross-entropy (CE) method for optimization. Triangular MFs are chosen because they are the simplest MFs that ensure the convergence of fuzzy Q-iteration. Many other optimization algorithms could be applied to optimize the MFs, e.g., genetic algorithms, tabu search, pattern search, etc. CE is chosen here as an illustrative example of a global optimization technique that can be used for this problem.

4.5.1 Cross-entropy optimization

This section provides a brief introduction to the CE method for optimization (Rubinstein and Kroese, 2004). For a full description, see Appendix A.[3] Consider the following optimization

[3]The CE notation used in this section is specialized to the MF optimization problem, by replacing the generic optimization variable a from Appendix A by the MF parameters ξ.

problem:

$$\max_{\xi \in \Xi} s(\xi) \tag{4.24}$$

where $s : \Xi \to \mathbb{R}$ is the score function to maximize, and the parameters ξ take values in the domain Ξ. Denote the maximum by s^*. The CE method for optimization maintains a probability density with support Ξ. In each iteration, a number of samples are drawn from this density and the score values for these samples are computed. A (smaller) number of samples that have the highest scores are kept, and the remaining samples are discarded. The probability density is then updated using the selected samples, such that at the next iteration the probability of drawing better samples is increased. The algorithm stops when the score of the worst selected sample no longer improves significantly.

Formally, a family of probability densities $\{p(\cdot; v)\}$ has to be chosen, where the dot stands for the random variable. This family has support Ξ and is parameterized by v. In each iteration τ of the CE algorithm, a number N_{CE} of samples is drawn from the density $p(\cdot; v_{\tau-1})$, their scores are computed, and the $(1 - \varrho_{\mathrm{CE}})$ quantile[4] λ_τ of the sample scores is determined, with $\varrho_{\mathrm{CE}} \in (0, 1)$. Then, a so-called associated stochastic problem is defined, which involves estimating the probability that the score of a sample drawn from $p(\cdot; v_{\tau-1})$ is at least λ_τ:

$$\mathrm{P}_{\xi \sim p(\cdot; v_{\tau-1})}(s(\xi) \geq \lambda_\tau) = \mathrm{E}_{\xi \sim p(\cdot; v_{\tau-1})} \{I(s(\xi) \geq \lambda_\tau)\} \tag{4.25}$$

where I is the indicator function, equal to 1 whenever its argument is true, and 0 otherwise.

The probability (4.25) can be estimated by importance sampling. For this problem, an importance sampling density is one that increases the probability of the interesting event $s(\xi) \geq \lambda_\tau$. The best importance sampling density in the family $\{p(\cdot; v)\}$, in the smallest cross-entropy sense, is given by the solution of:

$$\arg\max_{v} \mathrm{E}_{\xi \sim p(\cdot; v_{\tau-1})} \{I(s(\xi) \geq \lambda_\tau) \ln p(\xi; v)\} \tag{4.26}$$

An approximate solution of (4.26) is computed by:

$$v_\tau = \arg\max_{v} \frac{1}{N_{\mathrm{CE}}} \sum_{i_{\mathrm{s}}=1}^{N_{\mathrm{CE}}} I(s(\xi_{i_{\mathrm{s}}}) \geq \lambda_\tau) \ln p(\xi_{i_{\mathrm{s}}}; v) \tag{4.27}$$

CE optimization then proceeds with the next iteration using the new density parameter v_τ (the probability (4.25) is never actually computed). The updated density aims at generating good samples with higher probability, thus bringing $\lambda_{\tau+1}$ closer to the optimum s^*. The goal is to eventually converge to a density which generates samples close to optimal value(s) of ξ with very high probability. The highest score among the samples generated in all the iterations is taken as the approximate solution of the optimization problem, and the corresponding sample as an approximate location of the optimum. The algorithm can be stopped when the $(1 - \varrho_{\mathrm{CE}})$-quantile of the sample performance improves for d_{CE} consecutive iterations, and these improvements do not exceed $\varepsilon_{\mathrm{CE}}$; alternatively, the algorithm stops when a maximum number of iterations $\tau_{\max}$ is reached.

[4] If the score values of the samples are ordered increasingly and indexed such that $s_1 \leq \cdots \leq s_{N_{\mathrm{CE}}}$, then the $(1 - \varrho_{\mathrm{CE}})$ quantile is: $\lambda_\tau = s_{\lceil (1-\varrho_{\mathrm{CE}}) N_{\mathrm{CE}} \rceil}$ where $\lceil \cdot \rceil$ rounds the argument to the next greater or equal integer number (ceiling).

4.5.2 Fuzzy Q-iteration with CE optimization of the membership functions

In this section, a complete algorithm is given to optimize the MFs for fuzzy Q-iteration. Using the cross-entropy method, we optimize the locations of triangular MFs. Let the state space dimension be D. In the sequel, it is assumed that the state space is a hyperbox centered on the origin, i.e., $X = [-x_{\max,1}, x_{\max,1}] \times \cdots \times [-x_{\max,D}, x_{\max,D}]$ where $x_{\max,d} \in (0, \infty)$, $d = 1, \ldots, D$. This assumption can be relaxed and is only made here for simplicity.

Separately for each state variable x_d, a triangular fuzzy partition is defined with core values $c_{d,1} < \cdots < c_{d,N_d}$, which give N_d triangular MFs. The first and last core values are always equal to the limits of the domain: $c_{d,1} = -x_{\max,d}$ and $c_{d,N_d} = x_{\max,d}$. The product of each combination of (single-dimensional) MFs gives a pyramid-shaped D-dimensional MF in the fuzzy partition of X, like in Example 4.1. The parameters to optimize are the (scalar) free cores on all the axes; the first and last core values on each axis are not free. The number of free cores is therefore $N_\xi = \sum_{d=1}^{D}(N_d - 2)$. The parameter vector ξ can be obtained by concatenating the free cores, $\xi = [c_{1,2}, \ldots, c_{1,N_1-1}, \ldots\ldots, c_{D,2}, \ldots, c_{D,N_D-1}]^{\mathrm{T}}$. The domain of ξ is $\Xi = (-x_{\max,1}, x_{\max,1})^{N_1-2} \times \cdots \times (-x_{\max,D}, x_{\max,D})^{N_D-2}$.

The goal is to find a parameter vector ξ^* that maximizes the score function (4.23), using CE optimization. To this end, a family of densities has to be chosen. We choose a family with independent Gaussian components for each of the N_ξ parameters.[5] Gaussian densities are parameterized by their mean η and standard deviation σ. Using Gaussian densities has two advantages. The first advantage is that (4.27) has a closed-form solution (Rubinstein and Kroese, 2004). This solution is the mean and standard deviation of the best samples, and yields explicit update rules for the density parameters:

$$\eta_\tau = \frac{\sum_{i_\mathrm{s}=1}^{N_\mathrm{CE}} I(s(\xi_{i_\mathrm{s}}) \geq \lambda_\tau)\xi_{i_\mathrm{s}}}{\sum_{i_\mathrm{s}=1}^{N_\mathrm{CE}} I(s(\xi_{i_\mathrm{s}}) \geq \lambda_\tau)} \tag{4.28}$$

$$\sigma_\tau = \sqrt{\frac{\sum_{i_\mathrm{s}=1}^{N_\mathrm{CE}} I(s(\xi_{i_\mathrm{s}}) \geq \lambda_\tau)\left|\xi_{i_\mathrm{s}} - \eta_\tau\right|^2}{\sum_{i_\mathrm{s}=1}^{N_\mathrm{CE}} I(s(\xi_{i_\mathrm{s}}) \geq \lambda_\tau)}} \tag{4.29}$$

The second advantage is that with a Gaussian family the CE method can find a precise optimum location ξ^*. This is because the Gaussian density can converge, for $\sigma \to 0$ and $\eta = \xi^*$, to a degenerate (Dirac) distribution that assigns all the probability mass to the value ξ^*.

Because the support of the chosen density is $\mathbb{R}^{N_\xi}$, which is larger than Ξ, samples that do not belong to Ξ are rejected and generated again. The density parameter vector v consists of the mean η and standard deviation σ, each of them a vector with N_ξ elements. The parameters η and σ are initialized using:

$$\eta_0 = 0_{N_\xi}, \quad \sigma_0 = [x_{\max,1}, \ldots, x_{\max,1}, \ldots\ldots, x_{\max,D}, \ldots, x_{\max,D}]^{\mathrm{T}}$$

where each bound $x_{\max,d}$ is replicated $N_d - 2$ times, for $d = 1, \ldots, D$. These values ensure a good coverage of the state space with samples in the first iteration of the CE method.

[5] If for a sample of the CE algorithm two or more core values on a certain axis are identical, a single triangular MF with that core value is used in the fuzzy Q-iteration algorithm.

4.6 Experimental studies

In this section, the performance of fuzzy Q-iteration is studied experimentally. In Section 4.6.1, an application to a simulated servo-system is used to illustrate the theoretical convergence and consistency results about fuzzy Q-iteration. The same example is used to compare fuzzy Q-iteration with a Q-iteration algorithm that approximates the Q-function using radial basis functions (RBFs). Then, fuzzy Q-iteration is used to stabilize a (simulated) two-link manipulator in Section 4.6.2, and to swing-up and stabilize a real-life inverted pendulum in Section 4.6.3. For these three examples (Sections 4.6.1–4.6.3), the MFs are designed in advance. In Section 4.6.4, the CE optimization algorithm is applied to optimize the MFs for the classical car on the hill DP/RL benchmark.

4.6.1 Servo-system

This section illustrates the theoretical convergence and consistency results about fuzzy Q-iteration in an application to a (simulated) servo-system. This linear system (up to the saturation limits) was chosen because it allows for a good control policy to be computed analytically, and for extensive simulations to be performed with reasonable computational costs. First, the convergence rates of the synchronous and asynchronous fuzzy Q-iteration are compared. Then, the consistency of the algorithm is investigated, for continuous, as well as discontinuous reward functions. Additionally, fuzzy Q-iteration is compared to Q-iteration with RBF approximation.

Model and a near-optimal solution

Consider again the second-order discrete-time model of a servo-system, introduced in Example 3.1:

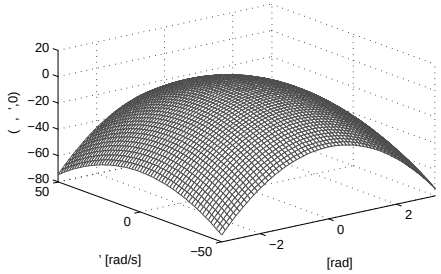
$$\begin{aligned} x_{k+1} &= f(x_k, u_k) = Ax_k + Bu_k \\ A &= \begin{bmatrix} 1 & 0.0049 \\ 0 & 0.9540 \end{bmatrix}, \quad B = \begin{bmatrix} 0.0021 \\ 0.8505 \end{bmatrix} \end{aligned} \tag{4.30}$$

This discrete-time model is obtained by discretizing a continuous-time model of the servo-system, which was determined by first-principles modeling of the real servo-system. The discretization is performed with the zero-order-hold method, using a sampling time of $T_\mathrm{s} = 0.005$ s. The position $x_{1,k} = \alpha$ is bounded to $[-\pi, \pi]$ rad, the velocity $x_{2,k} = \dot{\alpha}$ to $[-16\pi, 16\pi]$ rad/s, and the control input $u_k \in [-10, 10]$ V.

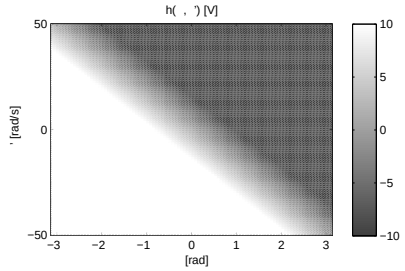
In our experiments, fuzzy Q-iteration was used to solve a discounted, quadratic regulation problem, described by the reward function:

$$\begin{aligned} r_{k+1} &= \rho(x_k, u_k) = -x_k^\mathrm{T} Q_\mathrm{rew} x_k - R_\mathrm{rew} u_k^2 \\ Q_\mathrm{rew} &= \begin{bmatrix} 5 & 0 \\ 0 & 0.01 \end{bmatrix}, \quad R_\mathrm{rew} = 0.01 \end{aligned} \tag{4.31}$$

This reward function is shown in Figure 4.3(a). The discount factor was chosen $\gamma = 0.95$ (this value is sufficient to produce a good control policy). This reward function is smooth and has bounded support; therefore, it is Lipschitz. The transition function is Lipschitz with the constant $L_f \leq \max\{\|A\|_2, \|B\|_2\}$ (this expression for L_f will hold for any system with linear dynamics). Therefore, the problem satisfies Assumption 4.8.



(a) A projection of the reward function (4.31) on the state space, for $u = 0$.



(b) A near-optimal policy obtained as the analytical solution of the linear quadratic regulation problem.

Figure 4.3: The reward function and a near-optimal policy for the servo-system.

Because the dynamics are linear and the reward function is quadratic, a solution (policy) for this problem can be computed analytically if the constraints on the states and actions are disregarded. This policy is a linear state feedback, and can be obtained by solving the Riccati equation (Bertsekas, 2007):

$$Y = A^{\mathrm{T}}[\gamma Y - \gamma^2 Y B(\gamma B^{\mathrm{T}} Y B + R_{\mathrm{rew}})^{-1} B^{\mathrm{T}}]A + Q_{\mathrm{rew}}$$

and substituting the solution Y in:

$$\widehat{h}^*(x) = -\gamma(\gamma B^{\mathrm{T}} Y B + R_{\mathrm{rew}})^{-1} B^{\mathrm{T}} Y A x \tag{4.32}$$

The policy $\widehat{h}^*$ is presented in Figure 4.3(b), where additionally the control input has been restricted to the admissible range $[-10, 10]$, using saturation. This policy is suboptimal, because the constraints were not taken into account when computing it. Because this policy is piecewise affine, the interpolation procedure (4.6) is likely to work well for computing a fuzzy Q-iteration policy. However, because the main goal of this example is to illustrate the theoretical guarantees about fuzzy Q-iteration, and the interpolated policy (4.6) would lead to a loss of these guarantees, the discrete-action policy (4.5) will be used instead.

Synchronous and asynchronous convergence

In order to study the convergence of fuzzy Q-iteration, a triangular fuzzy partition with $N' = 41$ equidistant cores for each state variable was defined, leading to $N = 41^2$ fuzzy sets in the two-dimensional partition of X. The action space was discretized into 15 equidistant values. First, (synchronous) fuzzy Q-iteration was run with a very small threshold $\varepsilon_{\mathrm{QI}} = 10^{-8}$ to obtain an accurate approximation $\widehat{\theta}^*$ of the optimal parameter vector. Then, synchronous and asynchronous fuzzy Q-iteration were run with a larger threshold $\varepsilon_{\mathrm{QI}} = 10^{-5}$, to examine how their parameter vectors approach the near-optimal $\widehat{\theta}^*$.

Figure 4.4 presents the evolution of the infinity-norm distance between the parameter vectors θ_ℓ and $\widehat{\theta}^*$ with the number of iterations ℓ, for both variants of the algorithm. The asynchronous algorithm approaches $\widehat{\theta}^*$ faster than the synchronous one, and terminates 20 iterations earlier (in 112 iterations, whereas the synchronous algorithm requires 132). Because the complexity of an iteration is nearly the same for the two algorithms, this translates directly into computational savings for the asynchronous version.

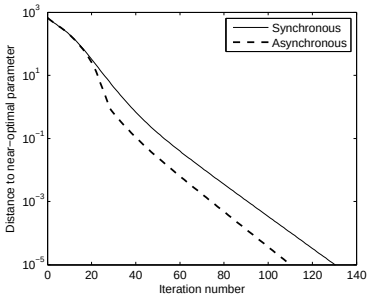


Figure 4.4: Convergence of synchronous and asynchronous fuzzy Q-iteration. The distance to the near-optimal parameter is computed as $\|\theta_\ell - \widehat{\theta}^*\|_\infty$.

Consistency and the effect of discontinuous rewards

In order to study the consistency of fuzzy Q-iteration, a triangular fuzzy partition with N' equidistant cores for each state variable was defined, leading to a total number of $N = N'^2$ fuzzy sets in the two-dimensional partition of X. The value of N' was gradually increased from 3 to 41. Similarly, the action was discretized into M equidistant values, with M ranging in $\{3, 5, \ldots, 15\}$ (only odd values were used because the 0 action is necessary for a good policy). The convergence threshold was set to $\varepsilon_{\mathrm{QI}} = 10^{-5}$ to ensure that the obtained parameter vector is close to θ^*. In a first set of experiments, fuzzy Q-iteration was run for each combination of N and M, and with the reward function (4.31). This reward function satisfies the theoretical conditions for the consistency of fuzzy Q-iteration, so it is expected that the solution improves as N and M increase.

The aim of a second set of experiments was to study the practical effect of violating the consistency conditions, by adding discontinuities to the reward function. Discontinuous rewards are common practice due to the origins of DP and RL in artificial intelligence, where discrete-valued tasks are often considered. In our experiment, the choice of the discontinuous reward function cannot be arbitrary. Instead, to ensure a meaningful comparison between the solutions obtained with the original reward function (4.31) and those obtained with the new reward function, the quality of the policies must be preserved. One way to preserve it is to add a term of the form $\gamma\varsigma(f(x_k, u_k)) - \varsigma(x_k)$ to each reward $\rho(x_k, u_k)$, where $\varsigma : X \to \mathbb{R}$ is an arbitrary bounded function (Ng et al., 1999). A discontinuous function ς is chosen:

$$\varsigma(x) = \begin{cases} 10 & \text{if } |x_1| \leq \pi/4 \text{ and } |x_2| \leq 4\pi \\ 0 & \text{otherwise} \end{cases} \tag{4.33}$$

$$\rho'(x_k, u_k) = \rho(x_k, u_k) + \gamma\varsigma(f(x_k, u_k)) - \varsigma(x_k)$$

The quality of the policies is preserved by this modification, in the sense that for any policy h, $Q^h_{\rho'} - Q^*_{\rho'} = Q^h_\rho - Q^*_\rho$, where Q_ρ is a Q-function under the reward ρ. Indeed, it is easy to show by replacing ρ' in the expression (2.2) for the Q-function, that for any policy h, including any optimal policy, $Q^h_{\rho'}(x, u) = Q^h_\rho(x, u) - \varsigma(x)\ \forall x, u$ (Ng et al., 1999). In particular, a policy is optimal for ρ' if and only if it is optimal for ρ.

The function ς is chosen positive in a rectangular region around the origin. Therefore, the newly added term rewards transitions that take the state inside this rectangular region, and

penalizes transitions that take it outside. A projection of ρ' on X, for $u = 0$, is presented in Figure 4.5. The additional positive rewards are visible as crests above the quadratic surface. The penalties are not visible in the figure, because their corresponding, downward-oriented crests are situated under the surface.

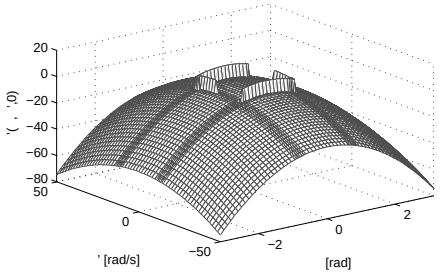


Figure 4.5: A projection of the modified reward function (4.33) on the state space, for $u = 0$.

The performance of the policies obtained with fuzzy Q-iteration is given in Figure 4.6. Each point in these graphs corresponds to the return of the policy, averaged over the grid of initial states:

$$X_0 = \{-\pi, -5\pi/6, -4\pi/6, \ldots, \pi\} \times \{-16\pi, -14\pi, \ldots, 16\pi\} \tag{4.34}$$

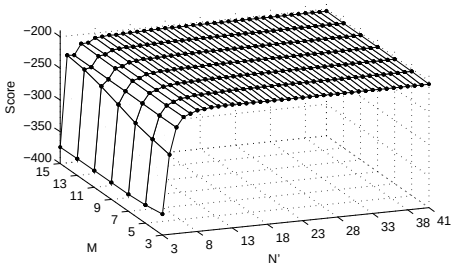
The returns are evaluated using simulation, with a precision of $\varepsilon_{\mathrm{MC}} = 0.1$. The infinite-horizon return is approximated in a finite time by simulating only the first K steps of the trajectory, with K given by (2.27).

While the reward functions used for Q-iteration are different, the performance evaluation is always done with the reward (4.31). As explained, the change in the reward function preserves the quality of the policies, so comparing policies in this way is meaningful. The qualitative evolution of the performance is similar when evaluated with (4.33).

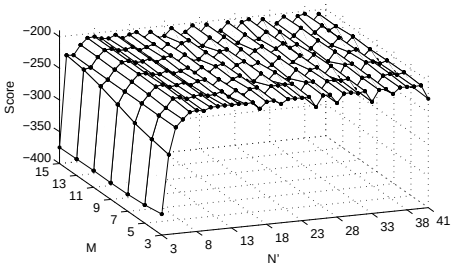
When the continuous reward is used, the performance of fuzzy Q-iteration is already close to the near-optimal value of -203.2 (computed with the policy (4.32)) for $N' = 20$ and is relatively smooth for $N' \geq 20$ – see Figures 4.6(a) and 4.6(c). The performance for $N' = 41$ and $M = 15$ is -203.3, very close to that of the policy (4.32). Also, the influence of the number of discrete actions is small for $N' \geq 4$. However, when the reward is changed to the discontinuous function (4.33), the performance varies significantly as N' increases – see Figure 4.6(b). For many values of N', the influence of M also becomes significant. Additionally, for many values of N' the performance is worse than with the continuous reward function – see Figure 4.6(d).

An interesting and somewhat counterintuitive fact is that the performance is not monotonous in N' and M. For a given value of N', the performance sometimes *decreases* as M increases. Similar situations occur as M is kept fixed and N' varies. This effect is present with both reward functions, but is much more pronounced in Figure 4.6(b) than in Figure 4.6(a) (see also Figure 4.6(c)). The magnitude of the changes decreases significantly as N' and M become large in Figures 4.6(a) and 4.6(c); this is not the case in Figure 4.6(b).

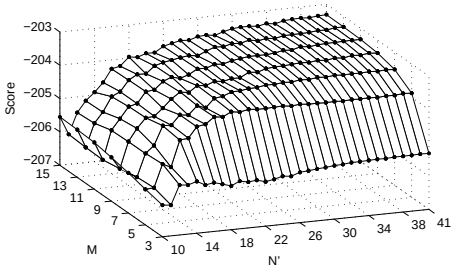
The negative effect of reward discontinuities on the consistency of the algorithm can be explained as follows. The discontinuous reward function (4.33) produces discontinuities in the optimal Q-function. As the placement of the MFs changes with increasing N', the



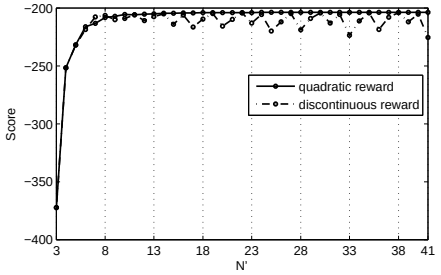
(a) Quadratic reward (4.31).



(b) Discontinuous reward (4.33); evaluation with quadratic reward.



(c) Quadratic reward, detail.



(d) Average performance over M, for varying N'.

Figure 4.6: The performance of fuzzy Q-iteration as a function of N and M, for quadratic and discontinuous reward.

accuracy with which the fuzzy representation captures these discontinuities changes as well. This accuracy depends less on the *number* of MFs, than on their *positions* (MFs should be ideally concentrated around the discontinuities). So, it may happen that for a certain, smaller value of N' the performance is better than for another, larger value. In contrast, the smoother optimal Q-function resulting from the continuous reward function (4.31) is easier to approximate using triangular MFs. Using discontinuous MFs instead (or nonlinear MFs with steeper slopes) is possible, but will probably not improve performance significantly in the absence of prior knowledge about the location of the discontinuities. Such prior knowledge is difficult to derive without knowing the optimal Q-function. Additionally, discontinuous MFs violate Requirement 4.9.

Remark: Similar behaviors were observed with different discontinuous reward functions (those experiments are not reported here). In particular, adding more discontinuities similar to those in (4.33) does not significantly influence the performance of Figure 4.6(b). Decreasing the magnitude of the discontinuities (e.g., replacing the value 10 by 1 in (4.33)) decreases the magnitude of the performance variations, but they are still present and they do not decrease as N and M increase. Note also that a consistency study for the inverted pendulum problem of Section 4.6.3 produces very similar results.

Comparison with RBF Q-iteration

There exist other approximators than fuzzy partitions that could be combined with Q-iteration to derive convergent algorithms. These approximators are usually linear in the parameters,

and use BFs that satisfy conditions related to (but different from) Requirement 4.1 of Section 4.3. This section compares fuzzy Q-iteration with one of these convergent algorithms, namely Q-iteration with normalized RBF approximation. RBFs are a common choice of BFs for approximate DP/RL (Tsitsiklis and Van Roy, 1996; Ormoneit and Sen, 2002).

Define a set of N normalized RBFs $\phi_i : X \to \mathbb{R}$, $i = 1, \ldots, N$, as follows:

$$\phi_i(x) = \frac{\phi'_i(x)}{\sum_{i'=1}^{N} \phi'_{i'}(x)}, \qquad \phi'_i(x) \quad = \exp\left[-\sum_{d=1}^{D} \frac{(x_d - c_{i,d})^2}{b_{i,d}^2}\right]$$

where ϕ'_i are (non-normalized) Gaussian axis-parallel RBFs, $c_i = [c_{i,1}, \ldots, c_{i,D}]^{\mathrm{T}}$ is the D-dimensional center of the ith RBF, and $b_i = [b_{i,1}, \ldots, b_{i,D}]^{\mathrm{T}}$ is its D-dimensional radius. Axis-parallel RBFs were selected for a fair comparison, because the triangular fuzzy partitions are also defined separately for each variable and then combined. The action space is discretized as for fuzzy Q-iteration (4.1).

Denote $\phi(x) = [\phi_1(x), \ldots, \phi_N(x)]^{\mathrm{T}}$. Assume that $c_1, \ldots, c_N$ are all distinct from each other. Form the matrix $\boldsymbol{\phi} = [\phi(c_1), \ldots, \phi(c_N)] \in \mathbb{R}^{N \times N}$, which is invertible by construction. Define a matrix $\boldsymbol{Q} \in \mathbb{R}^{N \times M}$ that collects the Q-values of the RBF centers-discrete action pairs: $\boldsymbol{Q}_{i,j} = Q(x_i, u_j)$. RBF Q-iteration uses the following approximation and projection mappings:

$$[F(\theta)](x, u_j) = \sum_{i=1}^{N} \phi_i(x)\theta_{[i,j]}, \; j = \arg\min_{j'} \|u - u_{j'}\|_2$$
$$[P(Q)]_{[i,j]} = [(\boldsymbol{\phi}^{-1})^{\mathrm{T}} \boldsymbol{Q}]_{i,j}$$

The convergence of RBF Q-iteration to a fixed point θ^* can be guaranteed if:

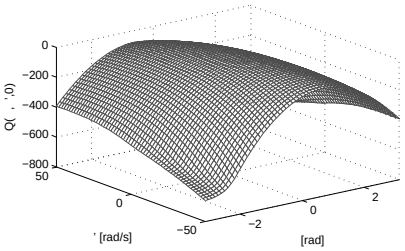
$$\sum_{i'=1, i' \neq i}^{N} \phi_{i'}(c_i) < \frac{1 - \gamma/\gamma'}{2} \tag{4.35}$$

for all i and some $\gamma' \in (\gamma, 1)$. Equation (4.35) restricts the sum of the values that the other RBFs take at the center of the ith RBF. This is an extension of the result in (Tsitsiklis and Van Roy, 1996) for normalized RBFs (the original result is given for non-normalized RBFs).

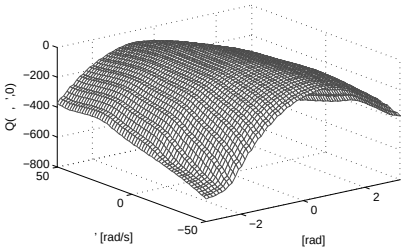
For a fair comparison, a number of MFs equal to the number of RBFs was used, and the cores (centers) of the MFs were chosen identical to the centers of the RBFs. A number of 41 equidistant cores were defined for each state variable, and an action discretization with 15 equidistant values was used for both algorithms. To ensure convergence, a set of radii that satisfies the inequalities (4.35) is required. Sufficient conditions to satisfy (4.35) can be given that are linear in the radii of the RBFs. Using these conditions, the radii are found by solving a problem involving linear constraints.

The approximate Q-functions and policies computed by the two algorithms, together with controlled trajectories from the initial state $x_0 = [-\pi, 0]^{\mathrm{T}}$, are presented in Figure 4.7. The top-left and bottom-right corner differences between the policies in Figures 4.7(c) and 4.7(d) and the policy in Figure 4.3(b) can be explained by observing that Q-iteration takes into account the constraints on the state and action variables, whereas the policy (4.32) does not.

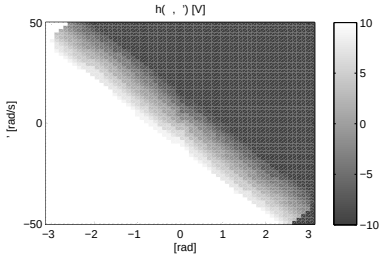
Comparing the fuzzy Q-iteration policy with the RBF Q-iteration policy, it appears that RBF Q-iteration introduces nonlinear artifacts in regions where the policy should actually be



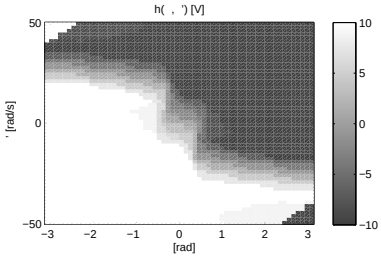
(a) A projection of the fuzzy approximate Q-function on the state space, for $u = 0$.



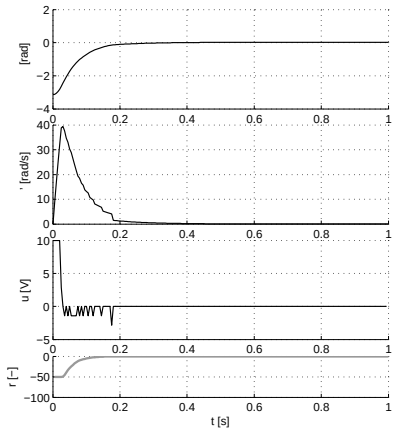
(b) A projection of the RBF approximate Q-function on the state space, for $u = 0$.



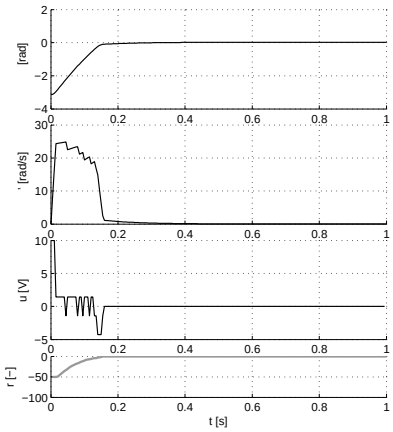
(c) Fuzzy Q-iteration policy.



(d) RBF Q-iteration policy.



(e) A trajectory controlled with the fuzzy Q-iteration policy, starting from $x_0 = [-\pi, 0]^{\mathrm{T}}$.



(f) A trajectory controlled with the RBF Q-iteration policy, starting from $x_0 = [-\pi, 0]^{\mathrm{T}}$.

Figure 4.7: Comparison between fuzzy Q-iteration and RBF Q-iteration.

linear. This has a negative effect on the control performance, as illustrated in Figure 4.7(f); compare with Figure 4.7(e). RBF Q-iteration obtains an average return of -210.7 for the grid X_0 defined by (4.34), whereas fuzzy Q-iteration obtains -203.3. The worse results of RBF approximation can be (informally) explained as follows. The optimal Q-function is largely smooth, because the transition and reward functions are largely smooth. However, the convergence constraints (4.35) impose restrictive limits on the RBF radii, which lead to narrow RBFs and therefore to a less smooth approximate Q-function. This Q-function poorly approximates the optimal Q-function. On the other hand, triangular MFs lead essentially to multi-linear interpolation, and therefore produce a more accurate approximation of the optimal Q-function. The variation introduced by the RBF approximator in the approximate Q-function can be seen upon a close examination of Figure 4.7(b); compare with Figure 4.7(a).

4.6.2 Two-link manipulator

In this section, fuzzy Q-iteration is used to stabilize a two-link manipulator, which is a more complex system than the servo-system of Section 4.6.1. This illustrates that fuzzy Q-iteration is able to control systems with more state and action variables than the servo-system, which had two state variables and a scalar control action.

The two-link manipulator, depicted in Figure 4.8, is described by the fourth-order nonlinear model:

$$M(\alpha)\ddot{\alpha} + C(\alpha, \dot{\alpha})\dot{\alpha} + G(\alpha) = \tau \tag{4.36}$$

where $\alpha = [\alpha_1, \alpha_2]^{\mathrm{T}}$, $\tau = [\tau_1, \tau_2]^{\mathrm{T}}$. The state signal contains the angles and angular velocities of the two links: $x = [\alpha_1, \dot{\alpha}_1, \alpha_2, \dot{\alpha}_2]^{\mathrm{T}}$, and the control signal is $u = \tau$. The angles wrap around in the interval $[-\pi, \pi]$ rad, and the velocities (measured in rad/s) and torques (measured in Nm) are bounded as in Table 4.1. The discrete time step is set to $T_{\mathrm{s}} = 0.05$ s, and the discrete-time dynamics f are obtained by numerical integration of (4.36) between the consecutive time steps.

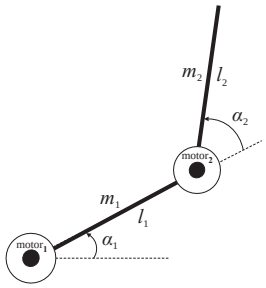


Figure 4.8: Schematic drawing of the two-link manipulator.

In the sequel, it is assumed that the manipulator operates in a horizontal plane, leading to $G(\alpha) = 0$. The mass matrix $M(\alpha)$ and the Coriolis and centrifugal forces matrix $C(\alpha, \dot{\alpha})$

Table 4.1: Physical variables of the manipulator

Symbol	Domain or value	Units	Meaning
α_1; α_2	$[-\pi, \pi]$; $[-\pi, \pi]$	rad	link angles
$\dot{\alpha}_1$; $\dot{\alpha}_2$	$[-2\pi, 2\pi]$; $[-2\pi, 2\pi]$	rad/s	link angular velocities
τ_1; τ_2	$[-1.5, 1.5]$; $[-1, 1]$	Nm	motor torques
l_1; l_2	0.4; 0.4	m	link lengths
m_1; m_2	1.25; 0.8	kg	link masses
I_1; I_2	0.066; 0.043	kg m^2	link inertias
c_1; c_2	0.2; 0.2	m	center of mass coordinates
b_1; b_2	0.08; 0.02	kg/s	dampings in the joints

have the following form:

$$M(\alpha) = \begin{bmatrix} P_1 + P_2 + 2P_3 \cos\alpha_2 & P_2 + P_3 \cos\alpha_2 \\ P_2 + P_3 \cos\alpha_2 & P_2 \end{bmatrix} \tag{4.37}$$

$$C(\alpha, \dot{\alpha}) = \begin{bmatrix} b_1 - P_3 \dot{\alpha}_2 \sin\alpha_2 & -P_3(\dot{\alpha}_1 + \dot{\alpha}_2)\sin\alpha_2 \\ P_3 \dot{\alpha}_1 \sin\alpha_2 & b_2 \end{bmatrix} \tag{4.38}$$

The meaning and values (or domains) of the physical variables in the system are given in Table 4.1. Using these, the rest of the parameters in (4.36) can be computed as follows: $P_1 = m_1 c_1^2 + m_2 l_1^2 + I_1$, $P_2 = m_2 c_2^2 + I_2$, and $P_3 = m_2 l_1 c_2$.

The control goal is the stabilization of the system around $\alpha = \dot{\alpha} = 0$, and is expressed by the following quadratic reward function:

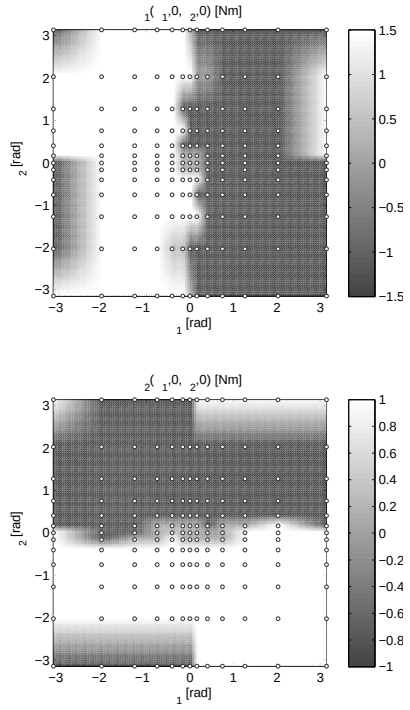
$$\rho(x, u) = -x^{\mathrm{T}} Q_{\mathrm{rew}} x, \quad \text{with } Q_{\mathrm{rew}} = \mathrm{diag}[1, 0.1, 1, 0.1] \tag{4.39}$$

The discount factor is set to $\gamma = 0.98$. To apply fuzzy Q-iteration, triangular fuzzy partitions were defined for every state variable and then combined as in Example 4.1. For the angles, a core was placed at the origin, and 6 logarithmically-spaced cores were placed on each side of the origin. For the velocities, a core was placed in the origin, and 3 logarithmically-spaced cores were used on each side of the origin. This leads to a total number of $(2 \cdot 6 + 1)^2 \cdot (2 \cdot 3 + 1)^2 = 8281$ MFs. The cores were spaced logarithmically to ensure a higher accuracy of the solution around the origin, while using only a limited number of MFs. This represents a mild form of prior knowledge about the importance of the region of the state space close to the origin. Each torque variable was discretized using 5 values: $\tau_1 \in \{-1.5, -0.36, 0, 0.36, 1.5\}$ and $\tau_2 \in \{-1, -0.24, 0, 0.24, 1\}$. These values are logarithmically spaced along the two axes of the action space. The convergence threshold was set to $\varepsilon_{\mathrm{QI}} = 10^{-5}$.

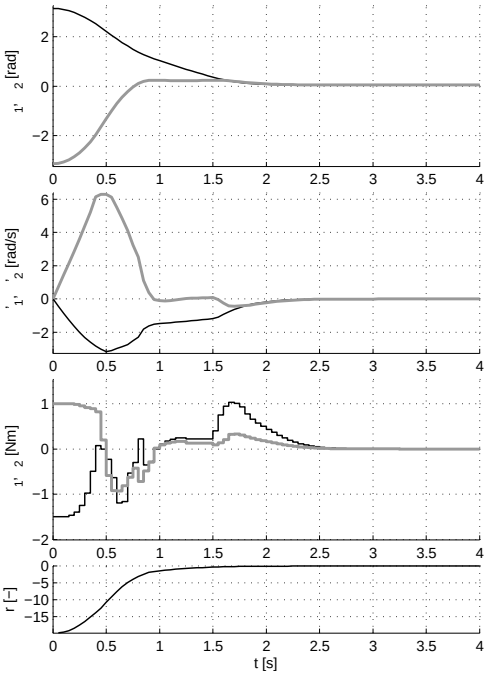
Figure 4.9(a) presents a slice through the policy computed with fuzzy Q-iteration, for zero angular velocities. This policy outputs continuous actions, and was computed with (4.6). Figure 4.9(b) presents a controlled trajectory starting from the initial state $x_0 = [\pi, 0, -\pi, 0]^{\mathrm{T}}$, together with the corresponding command and reward signals. The controller successfully stabilizes the system in about 2.5 s.

4.6.3 Inverted pendulum

Next, fuzzy Q-iteration is used to swing up and to stabilize a (real-life) under-actuated inverted pendulum. The inverted pendulum is obtained by placing an off-center weight on a



(a) A projection of the policy obtained with fuzzy Q-iteration on the plane (α_1, α_2), for $\dot{\alpha}_1 = \dot{\alpha}_2 = 0$. Darker colors correspond to negative actions, lighter colors to positive actions. The fuzzy cores for the angle variables are represented as small white disks with dark edges.



(b) A controlled trajectory of the two-link manipulator (thin black line – link 1, thick gray line – link 2). The initial state is $x_0 = [\pi, 0, -\pi, 0]^{\mathrm{T}}$.

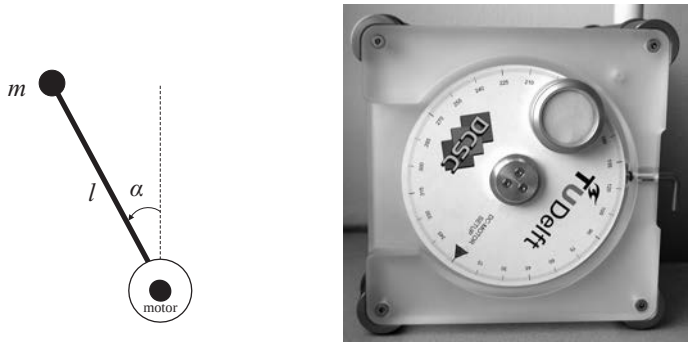
Figure 4.9: Fuzzy Q-iteration results for the two-link manipulator.

disk driven by a DC motor (Figure 4.10). The disk rotates in a vertical plane. This is different from the classical cart-pendulum problem, where the pendulum is attached to a cart and is indirectly actuated via the acceleration of the cart. Here, the pendulum is actuated directly, and the system only has two state variables. However, the control signal does not provide enough power to push the pendulum up in a single rotation. Instead, the pendulum needs to be swung back and forth (destabilized) to gather energy, prior to being pushed up and stabilized. This creates a difficult, highly nonlinear control problem.

The dynamics of the inverted pendulum are:

$$\ddot{\alpha} = \frac{1}{J}\left(mgl\sin(\alpha) - b\dot{\alpha} - \frac{K^2}{R}\dot{\alpha} + \frac{K}{R}u\right) \tag{4.40}$$

The parameters values are: $J = 1.91 \cdot 10^{-4}\,\mathrm{kgm}^2$, $m = 0.055\,\mathrm{kg}$, $g = 9.81\,\mathrm{m/s}^2$, $l = 0.042\,\mathrm{m}$, $b = 3 \cdot 10^{-6}\,\mathrm{kg/s}$, $K = 0.0536\,\mathrm{Nm/A}$, $R = 9.5\,\Omega$. Note some of these parameters (e.g., J, m, l) are rough estimates, and the real system exhibits unmodeled dynam-



(a) A schematic representation. (b) The real system.

Figure 4.10: The inverted pendulum.

ics, e.g., static friction. The sample time T_s is chosen equal to $0.005\,\mathrm{s}$. The state signal is $x = [\alpha, \dot{\alpha}]^\mathrm{T} = [x_1, x_2]^\mathrm{T}$. The angle α varies in the interval $[-\pi, \pi)$ rad, with $\alpha = 0$ pointing up, and 'rolls over' so that e.g., a rotation of $3\pi/2$ corresponds to $\alpha = -\pi/2$. The velocity $\dot{\alpha}$ is restricted to the interval $[-15\pi, 15\pi]$ rad/s, and the control action u_k is constrained to $[-3, 3]$ V. The goal is to stabilize the pendulum in the unstable equilibrium $x = 0$ (pointing up). The following reward function expresses this goal:

$$\begin{aligned} r_{k+1} &= \rho(x_k, u_k) = -x_k^\mathrm{T} Q_\mathrm{rew} x_k - R_\mathrm{rew} u_k^2 \\ Q_\mathrm{rew} &= \begin{bmatrix} 5 & 0 \\ 0 & 0.1 \end{bmatrix}, \quad R_\mathrm{rew} = 1 \end{aligned} \tag{4.41}$$

The discount factor is $\gamma = 0.98$. This discount factor is large so that rewards around the goal state (unstable equilibrium) influence the values of states early in the trajectories. This leads to an optimal policy that swings up the pendulum successfully.

Triangular fuzzy partitions with 19 equidistant cores were defined for both state variables, and then combined as in Example 4.1. This relatively large number of MFs was chosen to ensure a good accuracy of the solution. The control action was discretized using 5 equidistant values. A number of 3 values (maximum actuation in both directions, and zero actuation) would have sufficed to solve the problem. However, because the desired equilibrium is unstable, any discrete-action policy leads to chattering. To limit the effect of chattering, two more actions, smaller in magnitude, were added to the discretized set. The convergence threshold was $\varepsilon_\mathrm{QI} = 10^{-5}$.

Figure 4.11 presents the policy computed with fuzzy Q-iteration. Figure 4.12 presents swing-up trajectories starting from the stable equilibrium $x_0 = [-\pi, 0]^\mathrm{T}$ (pointing down), together with the corresponding command and reward signals. Figure 4.12(a) is a simulation result with the model (4.40), and Figure 4.12(b) is the result with the real system. For the real system, only the position is measured, and the angular velocity is estimated using a discrete difference, which results in a noisy signal. Although the model is simplified and does not include stochastic effects such as measurement noise, the policy resulting from fuzzy Q-iteration performs well: it stabilizes the real system in about $1.5\,\mathrm{s}$, around $0.25\,\mathrm{s}$ longer than in simulation. This discrepancy is due to the differences between the model and the real system. Note that the control action chatters, because only discrete actions are available.

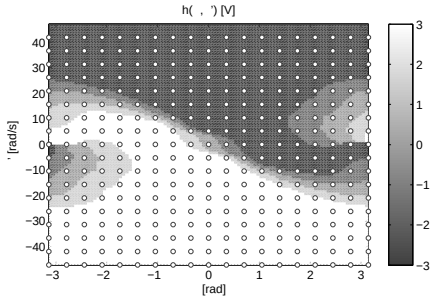
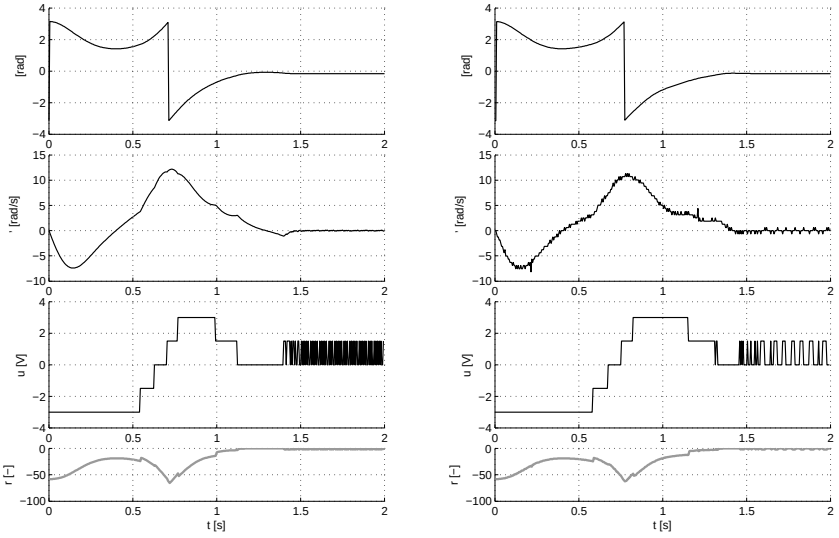


Figure 4.11: Fuzzy Q-iteration policy for the inverted pendulum. The cores are represented as small white disks with dark edges.



(a) In simulation. (b) The real inverted pendulum.

Figure 4.12: Swing-up trajectories of the inverted pendulum.

4.6.4 Optimizing membership functions for the car on the hill

In this section, we apply fuzzy Q-iteration with MF optimization to the car-on-the-hill problem. This is a classical benchmark for approximate DP/RL, first described by Moore and Atkeson (1995). For instance, Munos and Moore (2002) used this problem as a primary benchmark for their variable-resolution V-iteration technique. Ernst et al. (2005) used the car on the hill, among other examples, to validate an RL algorithm that approximates Q-functions with ensembles of regression trees.

In this problem, a point mass (the 'car') has to be driven past the top of a frictionless hill by applying a horizontal force. The problem is schematically represented in Figure 4.13. For some initial states, the maximum available force is not sufficient to drive the car straight up the hill. Instead, it has to be driven up the opposite slope (left) and gather momentum prior

to accelerating towards the goal (right). This problem is similar to the inverted pendulum of Section 4.6.3; there, the pendulum had to be swung back and forth to gather momentum, which corresponds to driving the car left and then right. The position and velocity of the car are bounded. Whenever these bounds are exceeded, the car reaches a terminal state from which it can no longer escape, and the task terminates.

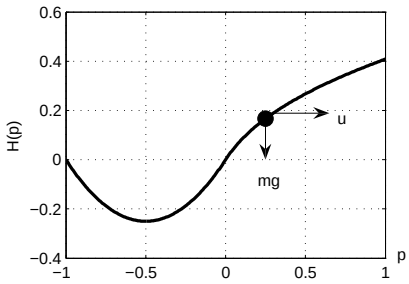


Figure 4.13: The car on the hill. The 'car' is represented as a black bullet, and its goal is to drive out of the figure to the right.

The horizontal position of the car (in meters) is denoted by p, and the shape of the hill is:

$$H(p) = \begin{cases} p^2 + p & \text{if } p < 0 \\ \frac{p}{\sqrt{1+5p^2}} & \text{if } p \geq 0 \end{cases}$$

The dynamics of this problem are (Ernst et al., 2005):

$$\ddot{p} = \frac{1}{1 + \left(\frac{\mathrm{d}H(p)}{\mathrm{d}p}\right)^2} \left(u - g \frac{\mathrm{d}H(p)}{\mathrm{d}p} - \dot{p}^2 \frac{\mathrm{d}H(p)}{\mathrm{d}p} \frac{\mathrm{d}^2 H(p)}{\mathrm{d}^2 p} \right) \tag{4.42}$$

where a unity mass was assumed and $g = 9.81$ is the gravitational acceleration. The notations $\dot{p}$ and $\ddot{p}$ indicate the first and second time derivatives of p. The discrete time step is set to $T_\mathrm{s} = 0.1$ s, and the discrete-time dynamics f are obtained by numerical integration of (4.42) between consecutive time steps.

The state signal consists of the horizontal position and speed of the car, $x = [p, \dot{p}]^\mathrm{T}$, and the control signal u is the horizontal force applied to the car. The state space is $X = [-1, 1] \times [-3, 3]$ plus a terminal state (see below), and the discrete action space is $U = \{-4, 4\}$. Whenever x_{k+1} is not in X (i.e., the position or speed exceed the bounds), a terminal state is reached. The goal is drive past the top of the hill to the right with a speed within the allowed limits. Reaching a terminal state in any other way is considered a failure. The reward function chosen to express this goal is:

$$\rho(x_k, u_k) = \begin{cases} -1 & \text{if } x_{1,k+1} < -1 \text{ or } |x_{2,k+1}| > 3 \\ 1 & \text{if } x_{1,k+1} > 1 \text{ and } |x_{2,k+1}| \leq 3 \\ 0 & \text{otherwise} \end{cases} \tag{4.43}$$

The discount factor is $\gamma = 0.95$. This discount factor is sufficiently large to obtain a good control policy. The reward function is chosen discontinuous for two reasons. The first reason

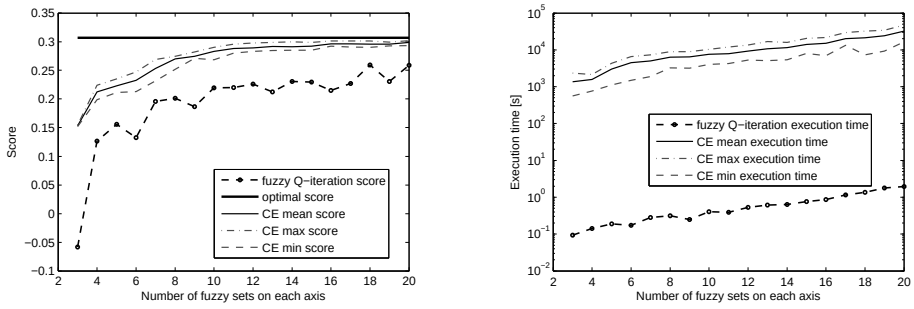
is that such discontinuous rewards are typically used with the car on the hill (Ernst et al., 2005). The second reason is to provide a challenging problem to the MF optimization algorithm. According to the results of the consistency study of Section 4.6.1, a discontinuous reward function such as (4.43) is likely to give an unpredictable performance when used with equidistant MFs. The MF optimization algorithm will have to place the limited number of MFs in a way that captures well the discontinuities of the Q-function. In this way, a more predictable performance should be recovered.

To compute the optimization criterion (4.23), the following grid of representative initial states is used:

$$X_0 = \{-1, -0.75, -0.5, \ldots, 1\} \times \{-3, -2, -1, \ldots, 3\}$$

where each point is weighted by $\frac{1}{|X_0|}$. The parameters of the algorithm are set as follows: $N_{\mathrm{CE}} = 5 \cdot 2 \cdot N_\xi$ (5 times the number of parameters needed to describe the probability density used in the CE optimization algorithm), $\varrho_{\mathrm{CE}} = 0.05$, $d_{\mathrm{CE}} = 5$. These are typical default values for CE optimization (Rubinstein and Kroese, 2004). The maximum number of CE iterations is $\tau_{\max} = 50$. The same value 10^{-3} is used as admissible error $\varepsilon_{\mathrm{MC}}$ in the return estimation (2.27), as the fuzzy Q-iteration convergence threshold $\varepsilon_{\mathrm{QI}}$, and as the CE convergence threshold $\varepsilon_{\mathrm{CE}}$.

The algorithm is run with the same number of MFs for both state variables, $N_1 = N_2$, and this number is gradually increased from 3 to 20.[6] Each such experiment is run 10 times with independent sets of samples, and the average score, together with the maximum and the minimum score in any run, are reported for every value of $N_1 = N_2$. Figure 4.14(a) compares the score obtained by fuzzy Q-iteration with optimized MFs, with the score obtained using the same number of equidistant MFs. The graph also shows the optimal score, computed by an exhaustive search for optimal open-loop control sequences. Fuzzy Q-iteration with optimized MFs consistently and reliably provides a better performance than with the same number of equidistant MFs. As expected, the discontinuous reward function gives unpredictable variations in the performance as the number of equidistant MFs is increased. Optimizing the MFs recovers a more predictable performance increase, probably because the MFs are adjusted to



(a) Performance. (b) Execution time.

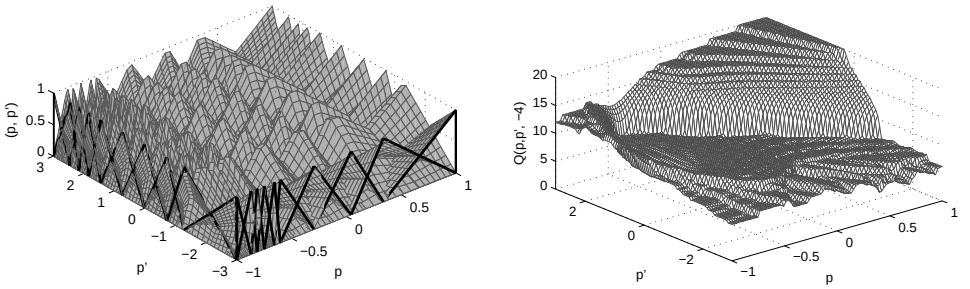
Figure 4.14: Comparison between fuzzy Q-iteration with optimized and equidistant MFs for the car on the hill.

[6]The experiments stop at 20 MFs to limit computation times per experiment in the order of hours on our 3 GHz Pentium IV machine, using MATLAB 7.1.

better represent the discontinuities of the Q-function. For $N_1 = N_2 \geq 12$, the performance of CE fuzzy Q-iteration is close to the optimal performance.

Figure 4.14(b) compares the computational cost of optimizing the MFs with the cost of using equidistant MFs. The performance increase obtained by optimizing the MFs comes at a large computational cost, several orders of magnitude higher than the cost incurred by fuzzy Q-iteration with equidistant MFs.

Figure 4.15(a) presents a representative set of MFs optimized using the CE method. The number of MFs on each axis is $N' = 10$. Figure 4.15(b) shows a projection of an approximately optimal Q-function on the state space, for $u = -4$ (the projection for $u = 4$ has a similar shape).[7] This Q-function has a large number of discontinuities. It is impossible to capture all these discontinuities with only 10 MFs on each axis. Instead, the MF optimization algorithm concentrates most of the MFs in the region of the state space where $p \approx -0.8$. In this region the car, having accumulated sufficient momentum, has to stop from moving left and accelerate toward the right; this is a critical control decision. Therefore, this placement of MFs illustrates that, when the number of MFs is insufficient to accurately represent Q-function over the entire state space, the optimization algorithm focuses the approximator on the regions that *matter the most* for the performance. The MFs on the velocity axis $\dot{p}$ are concentrated towards large values, possibly in order to represent more accurately the top-left region of the Q-function, which is the most irregular in the neighborhood of $p = -0.8$.



(a) Optimized MFs. (b) A projection of the Q-function on X, for $u = -4$.

Figure 4.15: A representative set of optimized MFs, together with an approximately optimal Q-function.

4.7 Conclusions and open issues

In this chapter, we have considered an algorithm for approximate value iteration that represents Q-functions using a fuzzy partition of the state space, and a discretization of the action space. The algorithm has been shown to be convergent to a near-optimal solution, and consistent under continuity assumptions on the dynamics and on the reward function. A version of the algorithm where parameters are updated in an asynchronous fashion has been shown to converge at least as fast as the synchronous variant, and has converged faster in an example.

[7] This Q-function was computed with fuzzy Q-iteration using a very fine fuzzy partition. Note that, because of the discontinuous nature of the reward function, the consistency of the algorithm is not guaranteed. Nevertheless, Figure 4.15(b) should provide at least a rough approximation of the optimal Q-function.

As an alternative to designing MFs in advance, we have proposed to optimize the parameters of a constant number of MFs. To evaluate each set of MFs, a policy is computed with fuzzy Q-iteration and its performance is computed by simulation. Using the cross-entropy method for optimization, we have designed an algorithm to optimize the locations of triangular MFs.

A thorough experimental evaluation of fuzzy Q-iteration has been carried out. The algorithm has shown a good performance in simulation and in practice. Fuzzy Q-iteration has worked better in an example than Q-iteration with RBF approximation. In the same example, increasing the approximation accuracy has resulted in a more predictable performance improvement when a continuous reward function was used, than with a discontinuous reward function. This indicates that discontinuous rewards can harm the performance in continuous-variable control tasks. Discontinuous rewards are common practice due to the origins of DP/RL in artificial intelligence, where discrete-valued tasks are often considered. For the car-on-the-hill benchmark, fuzzy Q-iteration has provided a better performance when using MFs that were optimized with the cross-entropy method, than when using equidistant MFs. Optimizing the MFs also recovered a predictable performance increase with an increasing number of MFs, despite the fact that the reward function was discontinuous. However, these performance improvements come at a large computational cost, several orders of magnitude higher than the cost incurred by fuzzy Q-iteration with equidistant MFs.

The triangular MFs used in this chapter make fuzzy Q-iteration exponentially complex in the number of state dimensions. This means that beyond five or six dimensions, triangular MFs cannot be used effectively. Other types of MFs could be used instead, to obtain a fuzzy partition with less than exponential complexity. Furthermore, action-space approximators more powerful than discretization could be studied, e.g., approximators based on fuzzy partitions of the action space.

It is important to analyze fuzzy Q-iteration for stochastic problems, when the expected values have to be *approximated* (since the analysis of Section 4.4 holds when the expected values can be evaluated exactly). It is also interesting to investigate the effects of using discontinuous rewards in the stochastic case, and to verify whether these effects are different from those observed in the deterministic case. Another research direction is developing model-free (RL) algorithms with fuzzy approximation that can guarantee convergence and consistency. Results from kernel-based or interpolation-based RL may be used in the theoretical study of such algorithms (Ormoneit and Sen, 2002; Szepesvári and Smart, 2004).

An extensive comparison of the performance (convergence, suboptimality, consistency) of the various types of linearly parameterized approximators that can be combined with the Q-iteration algorithm (e.g., radial basis functions, Kuhn triangulations, etc.) would be very useful. Such a comparison is currently missing from the literature.

While the CE method for optimization has been employed in this chapter to optimize the MFs, in principle there is no obstacle to applying any optimization technique to determine a good set of MFs. In particular, other meta-heuristic optimization techniques like genetic algorithms, tabu search, pattern search, etc., could be used. Moreover, instead of using optimization to find the MFs, resolution refinement techniques (Section 3.5.1) could be explored. Such techniques can help reduce the computational complexity of finding the MFs.

Chapter 5

Online and continuous-action least-squares policy iteration

This chapter introduces an extension of the least-squares policy iteration (LSPI) algorithm to online learning. In the online case, the performance of every intermediate policy is important, so policy improvements have to be performed often, at intervals during which only a few samples can be processed. This is in contrast to offline LSPI, which requires to process large batches of samples between consecutive policy improvements. In addition, an approach is introduced to use prior knowledge about the policy with online LSPI. A thorough numerical and experimental study is conducted to assess the performance of online LSPI, compare online LSPI with offline LSPI, and evaluate the effects of using prior knowledge in online LSPI. A continuous-action, polynomial Q-function approximator for LSPI is also investigated.

5.1 Introduction

The previous chapter has considered fuzzy Q-iteration, an approximate value iteration technique which iteratively computes an approximately optimal value function. The present chapter concerns approximate policy iteration, which computes in every iteration an approximate value function of the current policy, and then determines a new, improved policy which is greedy in this value function (see also Section 2.4). Least-squares policy iteration (LSPI) is a model-free algorithm for approximate policy iteration that uses a least-squares method to evaluate each policy (i.e., to compute its approximate value function). Least-squares techniques are among the most efficient methods for policy evaluation available to date (Konda, 2002; Bertsekas, 2007). Like fuzzy Q-iteration, LSPI approximates Q-functions with a linear combination of basis functions (BFs). Using Q-functions, rather than V-functions, simplifies the policy improvement step.

The original LSPI algorithm is offline (Lagoudakis et al., 2002; Lagoudakis and Parr, 2003a). It improves the policy only after a large batch of samples has been processed, and only the performance of the final policy is important. However, one of the main goals of reinforcement learning (RL) is to develop algorithms that learn online, by interacting with the controlled process. Therefore, this chapter extends LSPI to *online* learning. The main difference from the offline case is that the performance of every intermediate policy is impor-

tant, so the policy must be improved once every few transitions (sometimes, even after every transition). This means that only a small number of samples can be processed between consecutive policy improvements, and the algorithm cannot wait until every intermediate value function is accurately approximated. Unlike offline LSPI, which can use samples drawn from an arbitrary distribution over the state-action space, the online algorithm only has access to samples collected along trajectories of the process. This also implies that online LSPI has to actively explore in order to collect informative samples. Using an inverted pendulum swingup problem, we investigate the performance of online LSPI and the effects of varying its tuning parameters. The performance of online LSPI is compared to that of offline LSPI, using the same example. Online LSPI is also applied to stabilize a two-link manipulator with four state variables and two action variables.

To improve the learning rate of online LSPI, we propose to use prior knowledge about the policy. This is in contrast to most model-free RL algorithms, which typically work under the assumption that no prior knowledge is available. However, in practice, a certain amount of prior knowledge is almost always available. We develop a variant of online LSPI that uses knowledge about the monotonicity of the policy in the state variables. Monotonous policies are suitable for controlling e.g., (nearly) linear systems, or systems that are (nearly) linear and have monotonous input nonlinearities, such as saturation or dead-zone nonlinearities. In our approach, the policy is linearly parameterized using equidistant and identically shaped radial basis functions (RBFs). For such a parameterization, the monotonicity requirements can be expressed in terms of linear constraints on the policy parameters. These constraints are enforced in every policy improvement step. We investigate the effects of using prior knowledge in an example involving the control of a servo-system.

Additionally, we investigate a continuous-action Q-function approximator that uses orthogonal polynomial approximation in the action space. LSPI implementations from the literature use discrete actions (Lagoudakis et al., 2002; Lagoudakis and Parr, 2003a; Mahadevan and Maggioni, 2007). However, there exist important classes of control problems where continuous actions are important, and therefore where continuous-action Q-function approximation can help. For instance, when a system has to be stabilized around an unstable equilibrium, any discrete-action policy will lead to chattering of the control action and to limit cycles. We investigate the effects of using polynomial approximation in the inverted pendulum swingup problem.

The remainder of this chapter is organized as follows. Section 5.2 briefly describes the offline LSPI algorithm. Section 5.3 introduces the polynomial, continuous-action Q-function approximator. Section 5.4 describes online LSPI, and Section 5.5 presents our approach to using prior knowledge about the monotonicity of the policy. Section 5.6 gives a numerical evaluation of the proposed techniques. First, we evaluate offline LSPI with continuous-action Q-function approximation. Then, an extensive numerical and experimental evaluation of online LSPI is provided, and the effects of using prior knowledge in online LSPI are investigated. Section 5.7 concludes the chapter and outlines some open issues, together with promising approaches to address these issues.

5.2 Least-squares policy iteration

LSPI is an efficient algorithm for approximate policy iteration (Lagoudakis and Parr, 2003a). In each iteration ℓ of LSPI, the least-squares temporal difference for Q-functions (LSTD-Q) is

used to compute an approximate Q-function of the current policy h_ℓ. LSTD-Q was described in Section 3.4.1, and will be presented again here only briefly. It represents Q-functions using a linearly parameterized approximator with n BFs $\phi_l : X \times U \to \mathbb{R}$, $l = 1, \ldots, n$. To find the parameters θ of the approximator, LSTD-Q solves a projected Bellman equation of the form:

$$\Gamma\theta^* = z \tag{5.1}$$

where $\Gamma \in \mathbb{R}^{n \times n}$ and $z \in \mathbb{R}^n$. An approximate solution to this equation can be computed using state-action samples, together with the corresponding next states and rewards. These samples are collected in a set $\left\{(x_{l_\mathrm{s}}, u_{l_\mathrm{s}}, x'_{l_\mathrm{s}} = f(x_{l_\mathrm{s}}, u_{l_\mathrm{s}}), r_{l_\mathrm{s}} = \rho(x_{l_\mathrm{s}}, u_{l_\mathrm{s}})) \mid l_\mathrm{s} = 1, \ldots, n_\mathrm{s}\right\}$, where n_s is the number of samples. In the stochastic case, the next state samples are drawn from the transition probability function: $x'_{l_\mathrm{s}} \sim \tilde{f}(x_{l_\mathrm{s}}, u_{l_\mathrm{s}}, \cdot)$, and the rewards are computed accordingly: $r_{l_\mathrm{s}} = \tilde{\rho}(x_{l_\mathrm{s}}, u_{l_\mathrm{s}}, x'_{l_\mathrm{s}})$. The procedure described below can be applied both in the deterministic and in the stochastic case.

Using the n_s samples, a matrix Γ_{n_s} and a vector z_{n_s} are computed as follows:

$$\begin{aligned}
&\Gamma_0 = 0, \quad z_0 = 0 \\
&\Gamma_{l_\mathrm{s}} = \Gamma_{l_\mathrm{s}-1} + \phi(x_{l_\mathrm{s}}, u_{l_\mathrm{s}}) \left[\phi(x_{l_\mathrm{s}}, u_{l_\mathrm{s}}) - \gamma\phi(x'_{l_\mathrm{s}}, h_\ell(x'_{l_\mathrm{s}}))\right]^\mathrm{T} \\
&z_{l_\mathrm{s}} = z_{l_\mathrm{s}-1} + \phi(x_{l_\mathrm{s}}, u_{l_\mathrm{s}}) r_{l_\mathrm{s}}
\end{aligned} \tag{5.2}$$

where $\gamma \in [0, 1)$ is the discount factor. An approximately optimal parameter vector $\widehat{\theta}^*$ can be obtained by solving an approximate version of (5.1), obtained by replacing Γ and z with Γ_{n_s} and z_{n_s}:

$$\Gamma_{n_\mathrm{s}}\widehat{\theta}^* = z_{n_\mathrm{s}} \tag{5.3}$$

The approximate Q-function of h_ℓ is then $\widehat{Q}^{h_\ell}(x, u) = \phi^\mathrm{T}(x, u)\widehat{\theta}^*$. Using this Q-function, an improved policy is then computed with:

$$h_{\ell+1}(x) = \arg\max_u \widehat{Q}^{h_\ell}(x, u) \tag{5.4}$$

The accuracy of policy evaluation is indirectly influenced by the sampling distribution via the projected Bellman equation, as explained in Section 3.4.1. When some areas of the state-action space are insufficiently sampled, the resulting approximate Q-function may be inaccurate. In the regions of the state space where the Q-function is inaccurate, the improved policy will also be inaccurate.

Algorithm 5.1 summarizes LSPI. Note that, in practice, the policy $h_{\ell+1}$ is not explicitly stored. Instead, it is computed on demand for any given state x, using (5.4). This requires that the maximization (5.4) is solved exactly and efficiently for any x. Because (5.4) is solved exactly, LSPI performs *exact* policy improvements. Efficiency is important because the maximization has to be performed n_s times in every iteration of the LSPI algorithm, to evaluate in (5.2) the improved policy for each of the n_s samples. Solving (5.4) exactly and efficiently is possible when a suitable Q-function parameterization is chosen. For instance, when a discrete-action parameterization is used, (5.4) can be solved by enumeration over the set of discrete actions. This is the case for the parameterization described next, in Example 5.1.

Example 5.1 Approximation with Gaussian RBFs and discrete actions. Consider the following type of Q-function approximator, already introduced in Example 3.2 for the restricted

Algorithm 5.1 Least-squares policy iteration

Input:

BFs $\phi_l : X \times U \to \mathbb{R}$, $l = 1, \ldots, n$; samples $\left\{(x_{l_\mathrm{s}}, u_{l_\mathrm{s}}, x'_{l_\mathrm{s}}, r_{l_\mathrm{s}}) \mid l_\mathrm{s} = 1, \ldots, n_\mathrm{s}\right\}$
discount factor γ; convergence threshold $\varepsilon_{\mathrm{LSPI}}$

initialize policy h_0
repeat in every iteration $\ell = 0, 1, 2, \ldots$
  $\Gamma_0 \leftarrow 0_{n \times n}$, $z_0 \leftarrow 0_n$
  for $l_\mathrm{s} = 1, \ldots, n_\mathrm{s}$ **do**
    $\Gamma_{l_\mathrm{s}} \leftarrow \Gamma_{l_\mathrm{s}-1} + \phi(x_{l_\mathrm{s}}, u_{l_\mathrm{s}}) \left[\phi(x_{l_\mathrm{s}}, u_{l_\mathrm{s}}) - \gamma\phi(x'_{l_\mathrm{s}}, h_\ell(x'_{l_\mathrm{s}}))\right]^\mathrm{T}$
    $z_{l_\mathrm{s}} \leftarrow z_{l_\mathrm{s}-1} + \phi(x_{l_\mathrm{s}}, u_{l_\mathrm{s}})\, r_{l_\mathrm{s}}$
  end for
  compute θ_ℓ by solving $\Gamma_{n_\mathrm{s}}\theta_\ell = z_{n_\mathrm{s}}$
  $h_{\ell+1}(x) \leftarrow \arg\max_u \widehat{Q}^{h_\ell}(x, u) = \arg\max_u \phi(x, u)\theta_\ell$
until $\ell \geq 1$ **and** $\|\theta_\ell - \theta_{\ell-1}\| \leq \varepsilon_{\mathrm{LSPI}}$

Output: $h_{\ell+1}$

case of a two-dimensional state space. The action space is discretized into a small number M of discrete values $U_\mathrm{d} = \{u_1, \ldots, u_M\}$. Only these actions are allowed to appear in the set of samples. A number N of state-dependent, normalized Gaussian RBFs $\bar{\phi}_i : X \to \mathbb{R}$, $i = 1, \ldots, N$, are used to approximate the Q-function over the D-dimensional state space:

$$\bar{\phi}_i(x) = \frac{\phi'_i(x)}{\sum_{i'=1}^{N} \phi'_{i'}(x)}, \quad \phi'_i(x) = \exp\left[-\sum_{d=1}^{D} \frac{(x_d - c_{i,d})^2}{b_{i,d}^2}\right] \tag{5.5}$$

Here, ϕ'_i are the non-normalized RBFs, $c_i = [c_{i,1}, \ldots, c_{i,D}]^\mathrm{T}$ is the D-dimensional center of the ith RBF, and $b_i = [b_{i,1}, \ldots, b_{i,D}]^\mathrm{T}$ is its D-dimensional radius. A number $n = NM$ of state-discrete action BFs, $\phi_l : X \times U_\mathrm{d} \to \mathbb{R}$, $l = 1, \ldots, n$, are obtained in the following way. The RBFs are replicated for every discrete action, and all the RBFs that do not correspond to the current discrete action are taken equal to 0. Therefore, approximate Q-values can be computed with $\widehat{Q}(x, u_j) = \phi^\mathrm{T}(x, u_j)\,\theta$, for the state-action BF vector:

$$\phi(x, u_j) = [\underbrace{0, \ldots, 0}_{u_1}, \ldots, 0, \underbrace{\bar{\phi}_1(x), \ldots, \bar{\phi}_N(x)}_{u_j}, 0, \ldots, \underbrace{0, \ldots, 0}_{u_M}]^\mathrm{T} \in \mathbb{R}^{NM}$$

Because only a few discrete actions are considered for this approximator, the maximization in (5.4) can be solved by enumeration:

$$h_{\ell+1}(x) = u_{j^*}, \quad j^* = \arg\max_j \widehat{Q}^{h_\ell}(x, u_j)$$

This type of approximator will be used often in the sequel.

Note that a similar approximator (based on state-dependent RBFs and discrete actions) has been used with approximate Q-iteration in Section 4.6.1, where it has led to poor results. Those poor results have been caused by the severe restrictions placed on the radii of the RBFs by the convergence conditions for approximate Q-iteration. Because the Q-function was largely smooth, the narrow RBFs have led to a poor approximation of the optimal Q-function.

In contrast, the convergence of approximate policy evaluation requires no restrictions on the shape of the RBFs. This implies that the discrete-action RBF approximator works well for approximating (relatively) smooth Q-functions. □

5.3 LSPI with polynomial action approximation

Most implementations of LSPI from the literature use discrete actions (Lagoudakis et al., 2002; Lagoudakis and Parr, 2003a; Mahadevan and Maggioni, 2007). Usually, like in Example 5.1, a set of N BFs are defined over the state space only, and are replicated for each of the M discrete actions. However, there exist important classes of control problems where continuous actions are required. For instance, when a system has to be stabilized around an unstable equilibrium, any discrete-action policy will lead to undesirable chattering of the control action and to limit cycles. Therefore, this section proposes a continuous-action Q-function approximator for LSPI, which works for problems with scalar control actions. This approximator uses state-dependent BFs and orthogonal polynomials of the action variable, thus separating approximation over the state space from approximation over the action space. Orthogonal polynomials are preferred to plain polynomials in regression because they lead to numerically better conditioned problems. Because the action that maximizes the Q-function for a given state is not restricted to discrete values in (5.4) (as it was the case, e.g., in Example 5.1), this approximator produces continuous-action policies.

Like for the discrete-action approximator, a set of N state-dependent BFs is defined: $\bar{\phi}_i : X \to \mathbb{R}$, $i = 1, \ldots, N$. Only scalar control actions u are considered, bounded to an interval $U = [u_{\mathrm{L}}, u_{\mathrm{H}}]$. Requiring this type of actions is restrictive in general, but the example used in Section 5.6.1 to evaluate polynomial action approximation satisfies these conditions. To approximate over the action dimension of the state-action space, Chebyshev polynomials of the first kind are chosen as an illustrative example of orthogonal polynomials; many other types of orthogonal polynomials can be used. The Chebyshev polynomials of the first kind are defined by the recurrence relation:

$$\begin{aligned} \psi_0(\bar{u}) &= 1 \\ \psi_1(\bar{u}) &= \bar{u} \\ \psi_{j+1}(\bar{u}) &= 2\bar{u}\psi_j(\bar{u}) - \psi_{j-1}(\bar{u}) \end{aligned}$$

They are orthogonal to each other on the interval $[-1, 1]$ relative to the weight function $1/\sqrt{1-\bar{u}^2}$, which means that they satisfy:

$$\int_{-1}^{1} \frac{1}{\sqrt{1-\bar{u}^2}} \psi_j(\bar{u})\psi_{j'}(\bar{u}) d\bar{u} = 0, \quad \forall j \geq 0, j' \geq 0, j \neq j'$$

In order to take advantage of the orthogonality property, the action space U has to be scaled and translated into the interval $[-1, 1]$. This is easily accomplished using the affine transformation:

$$\bar{u} = -1 + 2\frac{u - u_{\mathrm{L}}}{u_{\mathrm{H}} - u_{\mathrm{L}}} \tag{5.6}$$

The approximate Q-values for an orthogonal polynomial approximator of degree M_{p} are

computed as follows:

$$\widehat{Q}(x,u) = \sum_{j=0}^{M_\mathrm{p}} \psi_j(\bar{u}) \sum_{i=1}^{N} \bar{\phi}_i(x)\theta_{[i,j+1]} \tag{5.7}$$

This can be written concisely as $\widehat{Q}(x,u) = \phi^\mathrm{T}(x,\bar{u})\theta$ for the state-action BF vector:

$$\begin{aligned}\phi(x,\bar{u}) = [&\bar{\phi}_1(x)\psi_0(\bar{u}), \ldots, \bar{\phi}_N(x)\psi_0(\bar{u}),\\ &\bar{\phi}_1(x)\psi_1(\bar{u}), \ldots, \bar{\phi}_N(x)\psi_1(\bar{u}),\\ &\ldots,\\ &\bar{\phi}_1(x)\psi_{M_\mathrm{p}}(\bar{u}), \ldots, \bar{\phi}_N(x)\psi_{M_\mathrm{p}}(\bar{u})]^\mathrm{T}\end{aligned} \tag{5.8}$$

The total number of state-action BFs (and therefore the total number of parameters) is $n = N(M_\mathrm{p}+1)$. Therefore, given the same number N of state BFs, a polynomial approximator of degree M_p has the same number of parameters as a discrete-action approximator with $M = M_\mathrm{p}+1$ discrete actions.

To find the greedy action (5.4) for a given state x, the approximate Q-function (5.7) for that value of x is first computed, which yields a polynomial in $\bar{u}$. Then, the roots of the derivative of this polynomial that lie in the interval $(-1,1)$ are found, the approximate Q-values are computed for each root and also for -1 and 1, and the action that corresponds to the largest Q-value is chosen. This action is then translated back into $U = [u_\mathrm{L}, u_\mathrm{H}]$:

$$\begin{aligned}h(x) &= u_\mathrm{L} + (u_\mathrm{H} - u_\mathrm{L})\frac{\arg\max_{\bar{u}\in\bar{U}(x)} \left[\phi^\mathrm{T}(x,\bar{u})\theta\right] + 1}{2}\\ \bar{U}(x) &= \{-1,1\} \cup \left\{ \bar{u}' \in (-1,1) \;\middle|\; \left.\frac{\mathrm{d}\psi(\bar{u})}{\mathrm{d}\bar{u}}\right|_{\bar{u}'} = 0, \text{where } \psi(\bar{u}) = \phi^\mathrm{T}(x,\bar{u})\theta \right\}\end{aligned} \tag{5.9}$$

In some cases, the polynomial will attain its maximum inside the interval $(-1,1)$. However, in other cases, it may not, so the boundaries $\{-1,1\}$ also have to be tested. Efficient algorithms can be used to compute the polynomial roots with high accuracy.[1] Therefore, the proposed parameterization allows the maximization problems in the policy improvement (5.4) to be solved efficiently and with high accuracy, and is well-suited for the use with LSPI.

In Section 5.6.2, we will investigate the effects of using continuous-action, polynomial Q-function approximation with offline LSPI, for the problem of swinging up an inverted pendulum. In this problem, polynomial Q-function approximation will not give a better performance than discrete actions. So, discrete-action approximation will be used when applying online LSPI to the inverted pendulum. The other problems to which online LSPI is applied (the servo-system of Section 5.6.4 and the robot arm of Section 5.6.3) involve stable systems, which can be satisfactorily stabilized with a discrete-action policy. Therefore, we will not consider polynomial approximation in the online context. Instead, only discrete-action Q-function approximators will be used for online LSPI.

5.4 Online LSPI

One of the main goals of RL is to develop algorithms that learn online, by interacting with the controlled process. In the online case, the performance is important not only at the end of the

[1] In our implementation, the roots are computed as the eigenvalues of the companion matrix of the polynomial $\frac{\mathrm{d}\psi(\bar{u})}{\mathrm{d}\bar{u}}$, using the `roots` function of MATLAB; see e.g., (Edelman and Murakami, 1995).

learning process, as is the case with offline algorithms, but also during learning. Ideally, the performance should improve after each observed state transition and corresponding reward. However, the original LSPI algorithm works offline (Lagoudakis et al., 2002; Lagoudakis and Parr, 2003a). It improves the policy only after a batch of samples is processed with LSTD-Q, and only the performance of the final policy is important.

This section extends the (offline) LSPI algorithm to online learning. Because the performance of each intermediate policy is important, policy improvements have to be performed at short intervals. In the extreme case, online LSPI improves the policy after every transition sample. This improved policy is applied to obtain a new transition sample. Next, another policy improvement takes place, and the cycle repeats. Such a variant is sometimes called *fully optimistic* LSPI. In general, policy improvements are performed once every several transitions. Such a method is partially optimistic. Even in partially optimistic variants, the number of transitions between consecutive policy improvements should not be very large. Optimistic policy iteration was proposed by Sutton (1988), and was also studied by Bertsekas and Tsitsiklis (1996) and Bertsekas (2007). Another important difference between online and offline LSPI is the following. While offline LSPI can use samples drawn from any distribution over the state-action space, online LSPI algorithm only has access to samples collected along trajectories of the process.

Online LSPI can store old samples and reuse them to update the matrix Γ and the vector z. However, its computational cost and memory requirements grow with the number of samples reused. In general, a growing computational cost makes it impractical to reuse many samples for online real-time learning. This is because the sampling time may be small, in which case Γ and z have to be updated fast. Instead of reusing all the samples, a limited window preserving only the latest samples can be kept, or a small subset containing only the most relevant samples can be selected. In the latter case, a well-defined measure of relevance is required. For simplicity, in the sequel we only consider algorithms that process each sample only once. For such algorithms, the computational cost incurred at every time step and the memory requirements are independent of the number of samples already observed.

Offline LSPI discards Γ and z after every policy improvement. In contrast, online LSPI processes only a few samples between consecutive policy improvements, and cannot afford to discard Γ and z. This is because the few samples that arrive before the next policy improvement are not sufficient to construct informative new values of Γ and z. To circumvent this problem, we never reset Γ and z, but keep updating them. The assumption underlying this heuristic is that the Q-functions of subsequent policies are similar, which means that the previous values of Γ and z are also representative for the improved policy.

A central concern with online LSPI is exploration: trying other actions than those given by the current policy. Without exploration, only the actions dictated by the current policy would be performed in every state, and samples of the other actions in that state would not be available. This would lead to a poor estimation of the Q-values of these other actions, and the resulting Q-function would not be reliable for policy improvement. Furthermore, exploration helps obtaining data from regions of the state space that would not be reached using only the greedy policy. Nevertheless, in addition to exploring, it is also important to exploit the current policy in order to control the process well. Exploitation becomes especially important if the current policy approaches the optimal one. This tradeoff between exploration and exploitation is typical for RL algorithms. In this chapter, ε-greedy exploration is used (see e.g., Sutton and Barto, 1998). This is a simple and classical way to solve the exploration

problem: at every step k, an exploratory action is applied with probability $\varepsilon_k \in [0, 1]$, and the greedy action with probability $(1 - \varepsilon_k)$. Exploratory actions are chosen randomly, from a uniform distribution over the action space. Additionally, ε_k typically decreases over time (as k increases), so that the algorithm increasingly exploits the current policy. Throughout this chapter, the exploration probability is initially set to a value ε_0, and decays exponentially once every second with a decay rate of $\varepsilon_\mathrm{d} \in (0, 1)$. This leads to the exploration schedule:

$$\varepsilon_k = \varepsilon_0 \, \varepsilon_\mathrm{d}^{\lfloor kT_\mathrm{s} \rfloor} \tag{5.10}$$

where T_s is the sampling time, and $\lfloor \cdot \rfloor$ denotes the largest integer smaller than or equal to the argument (floor). Many other exploration schedules are possible.

It is an open theoretical issue under what conditions the parameter vector of online LSPI converges, or indeed even if it converges at all. In fact, optimistic policy iteration is known to exhibit oscillations of the policy and other complex behaviors (Bertsekas and Tsitsiklis, 1996; Bertsekas, 2007). Note also that, in practice, online LSPI may be able to deal with processes that are slowly changing over time, by adapting the policy to take these changes into account. This is possible because the algorithm processes only new samples, which are representative for the current dynamics of the process.

Algorithm 5.2 presents online LSPI with ε-greedy exploration. Because only a few samples are processed before the first updates of θ, Γ will probably not be invertible for these first updates. This problem is solved here by initializing Γ to a small multiple $\beta_\Gamma > 0$ of the identity matrix, as suggested e.g., by Lagoudakis and Parr (2003a). Any small positive value will work for β_Γ. In this chapter, β_Γ is always 0.001.

Algorithm 5.2 Online LSPI with ε-greedy exploration

Input:

BFs $\phi_l : X \times U \to \mathbb{R}$, $l = 1, \ldots, n$
discount factor γ; policy improvement interval K_θ; exploration schedule ε_k, $k \geq 0$
a small constant $\beta_\Gamma > 0$

$\ell \leftarrow 0$; initialize policy h_0; $\Gamma_0 \leftarrow \beta_\Gamma \cdot I_{n \times n}$; $z_0 \leftarrow 0_n$
measure initial state x_0
for each time step $k \geq 0$ **do**
    $u_k \leftarrow \begin{cases} h_\ell(x_k) & \text{with probability } (1 - \varepsilon_k) \text{ (exploit)} \\ \text{a uniform random action in } U & \text{with probability } \varepsilon_k \text{ (explore)} \end{cases}$
    apply u_k, measure next state x_{k+1} and reward r_{k+1}
    $\Gamma_{k+1} \leftarrow \Gamma_k + \phi(x_k, u_k) \left[\phi(x_k, u_k) - \gamma \phi(x_{k+1}, h_\ell(x_{k+1}))\right]^\mathrm{T}$
    $z_{k+1} \leftarrow z_k + \phi(x_k, u_k) \, r_{k+1}$
    if $k = (\ell + 1)K_\theta$ **then** ▷ k is the next positive multiple of K_θ
        compute θ_ℓ by solving $\Gamma_{k+1}\theta_\ell = z_{k+1}$
        $h_{\ell+1}(x) \leftarrow \arg\max_u \phi(x, u)\theta_\ell$ ▷ policy improvement
        $\ell \leftarrow \ell + 1$
    end if
end for

Besides β_Γ, online LSPI uses two other parameters that were not present in offline LSPI: the number $K_\theta \in \mathbb{N}^*$ of samples (transitions) between consecutive policy improvements,

and the exploration schedule ε_k, $k \geq 0$. The number K_θ is the most important parameter of online LSPI. When $K_\theta = 1$, the policy is updated after every sample and online LSPI is fully optimistic. When $K_\theta > 1$, the algorithm is partially optimistic. The exploration schedule ε_k is also important for the performance of the algorithm. The number K_θ should not be chosen too large, and a significant amount of exploration is recommended, i.e., ε_k should not approach 0 too fast. The effects of K_θ and of the exploration schedule on the performance will be studied experimentally in Section 5.6.2.

Some Markov decision processes (MDPs) have terminal states. For instance, many robot manipulators have safeguards that stop the robot's motion if its pose gets outside the operating range, after which human intervention is required. In an MDP framework, that would correspond to the robot entering a terminal state. The process cannot leave a terminal state, so once such a state is reached, the learning trial is terminated as well. At the beginning of each new trial, the process has to be reset in some fashion to an initial state. The same state can be used for every trial, or the process can be reset to different states in different trials. If the process does not have terminal states, there is the possibility of learning from a single, infinitely-long trial. However, even if the process does not have terminal states, or does not enter them, it may still be beneficial for learning to terminate trials artificially. For instance, when learning a stabilizing control law, if the process has been successfully stabilized and the exploration is insufficient to drive the state away from the equilibrium, there is little more to be learned from that trial, and a new trial starting from a new state will be more useful. The effects of the trial length on the learning performance will be studied experimentally in Section 5.6.2.

5.5 Using prior knowledge in online LSPI

Typically, model-free RL algorithms work under the assumption that no prior knowledge is available about the process and about the optimal solution. However, in practice, a certain amount of prior knowledge is almost always available. Prior knowledge can refer to, e.g., the process dynamics, the optimal policy, or the optimal value function. In this chapter, we focus on using prior knowledge about the optimal policy, or more generally about good control policies that are not necessarily optimal. This focus is motivated by the fact that it is often easier to obtain knowledge about the policy than about the value function.

A general way of specifying knowledge about the policy is to use constraints. For instance, one might know, and therefore require, that the policy is (globally or piecewise) monotonous in the state variables. Or, inequality constraints might be available on the state and action variables. The possibility of constraining policies is to our knowledge unexplored in the field of approximate RL. The main benefit of constraining policies is an expected speedup of the learning process. A speedup is expected because the algorithm no longer invests valuable learning time in trying unsuitable policies that do not satisfy the constraints. This is especially relevant in online learning, although it may help reduce the computational load for offline learning as well. Another benefit is the guaranteed constraint satisfaction.

The original LSPI algorithm does not explicitly represent policies, but computes them on demand by using (5.4). Therefore, the policy is *implicitly* defined by the Q-function. In principle, it is possible to use the constraints on the policy in order to derive corresponding constraints on the Q-function. However, this derivation is very hard to perform in general, because of the complex relationship between a policy and its Q-function. A simpler solution is

to *explicitly* parameterize the policy, and to enforce the constraints in the policy improvement step. This is the solution adopted in the sequel.

In this section, we develop an online LSPI algorithm that uses explicitly parameterized policies that are globally monotonous in the state variables. This means the policies are monotonous with respect to any state variable, if the other state variables are held constant (see Section 5.5.2 for a formal definition). Such monotonous policies are suitable for controlling important classes of systems. For instance, they are suitable for controlling (nearly) linear systems, or a nonlinear system in a neighborhood of an equilibrium where the system is nearly linear. This is because linear policies, which work well for controlling linear systems, are monotonous. Monotonous policies also work well for linear systems with monotonous input nonlinearities (such as saturation or dead-zone nonlinearities). In such cases, the policy may be strongly nonlinear, but still monotonous. Of course, in general, the global monotonicity requirement is restrictive. It can be made more general e.g., by requiring that the policy is monotonous only over a restricted region of the state space, such as in a neighborhood of an equilibrium. Multiple monotonicity regions can also be considered. However, as a first step, in this chapter we only consider globally monotonous policies.

In our approach, the policy is parameterized using equidistant and identically shaped RBFs. For this choice of RBFs, the monotonicity requirements can be expressed in terms of linear inequality constraints on the policy parameters. These constraints are enforced in every policy improvement step.

5.5.1 Parameterized policies

As in Section 3.4.2, we consider a policy parameterization that is linear in the parameters $\vartheta \in \mathbb{R}^{\mathcal{N}}$ and uses a set of state-dependent BFs $\varphi_1, \ldots, \varphi_{\mathcal{N}} : X \to \mathbb{R}$. The approximate policy is:[2]

$$\widehat{h}(x) = \sum_{i=1}^{\mathcal{N}} \varphi_i(x)\vartheta_i = \varphi^{\mathrm{T}}(x)\vartheta \tag{5.11}$$

with $\varphi(x) = [\varphi_1(x), \ldots, \varphi_{\mathcal{N}}(x)]^{\mathrm{T}}$. In the sequel, Gaussian RBFs of the type introduced in Example 5.1 are used to parameterize the policy. When no prior knowledge about the policy is available, the policy improvement step can be performed by solving a least-squares problem to find an improved policy parameter vector:

$$\vartheta_{\ell+1} = \arg\min_{\vartheta \in \mathbb{R}^{\mathcal{N}}} \sum_{i_{\mathrm{s}}=1}^{\mathcal{N}_{\mathrm{s}}} \left| \varphi^{\mathrm{T}}(x_{i_{\mathrm{s}}})\vartheta - \arg\max_{u} \phi^{\mathrm{T}}(x_{i_{\mathrm{s}}}, u)\theta_\ell \right|^2 \tag{5.12}$$

where $\{x_1, \ldots, x_{\mathcal{N}_{\mathrm{s}}}\}$ is a set of samples for policy improvement. In this formula, the term:

$$\arg\max_{u} \phi^{\mathrm{T}}(x_{i_{\mathrm{s}}}, u)\theta_\ell = \arg\max_{u} \widehat{Q}^{\widehat{h}_\ell}(x_{i_{\mathrm{s}}}, u)$$

is the greedy action for the i_{s}th sample.[3] To obtain online LSPI with parameterized policies, the exact policy improvement in line 10 of Algorithm 5.2 is replaced by (5.12), and the parameterized policy (5.11) is used instead of exact, greedy actions.

[2] Recall that calligraphic notation is used to emphasize quantities related to policy approximation.

[3] Alternatively, policy improvement could be performed using (3.27) (Section 3.4.2), which maximizes the approximate Q-values of the actions chosen by the policy in the state samples. However, (3.27) can generally be a difficult nonlinear optimization problem, whereas (5.12) is a convex optimization problem, which is easier to solve.

Note that, for simplicity, only scalar control actions are considered in this derivation. However, it is easy to extend the policy parameterization, together with the monotonicity constraints that will be introduced in Section 5.5.2, to the case of multiple action variables.

The approximate policy (5.11) produces continuous actions. This policy approximator could be combined with a continuous-action Q-function approximator, such as the polynomial approximator introduced in Section 5.3. However, as already noted in Section 5.3, in this chapter we do not use continuous-action Q-function approximation for online LSPI.[4] Instead, the continuous-action policy approximator will be used in combination with a discrete-action Q-function approximator, such as that of Example 5.1. Two implications of this choice have to be considered. Firstly, the greedy action samples in (5.12) are always discrete actions. Secondly, during learning, the continuous actions given by the policy have to be quantized into actions belonging to the discrete set U_d. If the actions were not quantized, for any continuous action different from all the discrete actions, the Q-function BFs would all be zero, and the update of Γ in (5.2) would be meaningless. Using the quantization procedure implies that the policy evaluation step actually estimates the Q-function of a *quantized* version of the policy. Throughout this chapter, the following quantization function is used:

$$q_\mathrm{d} : U \to U_\mathrm{d}, \quad q_\mathrm{d}(u) = \underset{u_j \in U_\mathrm{d}}{\arg\min} |u - u_j| \tag{5.13}$$

5.5.2 Monotonous policies

A policy is monotonous in the state variables if and only if it satisfies the following conditions for every state space axis $d = 1, \ldots, D$, where D is the dimension of the state space:

$$\delta_{\mathrm{mon},d} h(x) \leq \delta_{\mathrm{mon},d} h(\bar{x}) \quad \text{for all } x, \bar{x} \text{ that satisfy } x_d \leq \bar{x}_d \text{ and } x_{d'} = \bar{x}_{d'}, \forall d' \neq d \tag{5.14}$$

where $\delta_\mathrm{mon} \in \{-1, 1\}^D$ specifies the monotonicity directions along each axis: if $\delta_{\mathrm{mon},d}$ is -1 then h is decreasing along the dth axis of the state space, and if it is 1 then h is increasing.

In this chapter, policies are approximated using normalized RBFs that are distributed on an equidistant grid and have identical radii. The centers c_i of such RBFs are of the form $[c_{1,i_1}, \ldots, c_{D,i_D}]^\mathrm{T}$ where $i_d = 1, \ldots, \mathcal{N}_d$. Without loss of generality, let the centers be ordered along each axis:

$$c_{d,1} < \cdots < c_{d,\mathcal{N}_d}, \quad d = 1, \ldots, D$$

When using these BFs, in order to satisfy (5.14) it is sufficient that the parameters corresponding to each sequence of RBFs along all the grid lines in every dimension of X, are ordered. An example of this ordering relationship, for a 3×3 grid of RBFs, is:

$$\begin{array}{ccccc} \vartheta_{[1,1]} & \leq & \vartheta_{[1,2]} & \leq & \vartheta_{[1,3]} \\ \wedge & & \wedge & & \wedge \\ \vartheta_{[2,1]} & \leq & \vartheta_{[2,2]} & \leq & \vartheta_{[2,3]} \\ \wedge & & \wedge & & \wedge \\ \vartheta_{[3,1]} & \leq & \vartheta_{[3,2]} & \leq & \vartheta_{[3,3]} \end{array} \tag{5.15}$$

[4] In fact, we have also studied experimentally the performance of online LSPI with polynomial Q-function approximation and parameterized, continuous-action policies. The results were unsatisfactory, and worse than those obtained using discrete-action Q-function approximation with parameterized policies. These results are not reported here.

where the second and fourth line express inequalities along the vertical axis. In this case, the policy is decreasing along the first axis of X (vertically in the equation), and increasing along the second axis (horizontally in the equation).

The monotonicity conditions on the parameters can be written in a matrix form as:

$$\Delta_{\mathrm{mon}}\vartheta \leq 0_{N_{\mathrm{mon}}} \tag{5.16}$$

where the number of constraints N_{mon} is:

$$N_{\mathrm{mon}} = \sum_{d=1}^{D}(\mathcal{N}_d - 1) \prod_{d'=1,\ d'\neq d}^{D} \mathcal{N}_{d'}$$

and the matrix $\Delta_{\mathrm{mon}} \in \mathbb{R}^{N_{\mathrm{mon}}\times\mathcal{N}}$ contains one row $\bar{i}$ for every combination of RBF indices $i = [i_1, \ldots, i_D]$ and $i' = [i'_1, \ldots, i'_D]$ such that $i'_d = i_d + 1$, $i'_{d'} = i_{d'}\ \forall d' \neq d$. This row is formed as follows:

$$\Delta_{\mathrm{mon},\bar{i},i} = \delta_{\mathrm{mon},d}, \quad \Delta_{\mathrm{mon},\bar{i},i'} = -\delta_{\mathrm{mon},d}, \quad \Delta_{\mathrm{mon},\bar{i},j} = 0 \text{ for } j \neq i \text{ and } j \neq i'$$

The notation $[i_1, \ldots, i_D]$ is used to map a D-dimensional index $(i_1, \ldots, i_D)$ into the corresponding single-dimensional index i. The RBF located at indices i_d along every axis d is multiplied by the policy parameter $\vartheta_{[i_1,\ldots,i_D]}$ when the approximate policy is computed. In two dimensions, $[i_1, i_2] = i_1 + (i_2 - 1)\mathcal{N}_1$. Generalizing to D dimensions, we have:

$$[i_1, \ldots, i_D] = i_1 + (i_2 - 1)\mathcal{N}_1 + (i_3 - 1)\mathcal{N}_1\mathcal{N}_2 + \cdots\cdots + (i_D - 1)\mathcal{N}_1\mathcal{N}_2\cdots\mathcal{N}_{D-1}$$

To clarify the notation, the following equation shows how the matrix Δ_{mon} is constructed for the constraints (5.15):

Δ_{mon}	$[1,1]$	$[2,1]$	$[3,1]$	$[1,2]$	$[2,2]$	$[3,2]$	$[1,3]$	$[2,3]$	$[3,3]$
$\vartheta_{[1,1]} \leq \vartheta_{[1,2]}$	1	0	0	−1	0	0	0	0	0
$\vartheta_{[1,2]} \leq \vartheta_{[1,3]}$	0	0	0	1	0	0	−1	0	0
$\vartheta_{[2,1]} \leq \vartheta_{[2,2]}$	0	1	0	0	−1	0	0	0	0
$\vartheta_{[2,2]} \leq \vartheta_{[2,3]}$	0	0	0	0	1	0	0	−1	0
$\vartheta_{[3,1]} \leq \vartheta_{[3,2]}$	0	0	1	0	0	−1	0	0	0
$\vartheta_{[3,2]} \leq \vartheta_{[3,3]}$	0	0	0	0	0	1	0	0	−1
$\vartheta_{[1,1]} \geq \vartheta_{[2,1]}$	−1	1	0	0	0	0	0	0	0
$\vartheta_{[2,1]} \geq \vartheta_{[3,1]}$	0	−1	1	0	0	0	0	0	0
$\vartheta_{[1,2]} \geq \vartheta_{[2,2]}$	0	0	0	−1	1	0	0	0	0
$\vartheta_{[2,2]} \geq \vartheta_{[3,2]}$	0	0	0	0	−1	1	0	0	0
$\vartheta_{[1,3]} \geq \vartheta_{[2,3]}$	0	0	0	0	0	0	−1	1	0
$\vartheta_{[2,3]} \geq \vartheta_{[3,3]}$	0	0	0	0	0	0	0	−1	1

The constraints along the horizontal axis of (5.15) are added to Δ_{mon} first, followed by the constraints along the vertical axis. To help with reading the matrix, the inequality constraints expressed by the matrix rows are shown to the left of the matrix, and the parameter indices corresponding to the matrix columns are shown along the top of the matrix.

The monotonicity condition (5.16) is enforced in Algorithm 5.2 (online LSPI) by replacing the unconstrained policy improvement problem in line 10 by the constrained least-squares

problem:

$$\vartheta_{\ell+1} = \underset{\vartheta \in \mathbb{R}^{\mathcal{N}},\ \Delta_{\text{mon}}\vartheta \leq 0_{N_{\text{mon}}}}{\arg\min} \sum_{i_{\text{s}}=1}^{\mathcal{N}_{\text{s}}} \left| \varphi^{\text{T}}(x_{i_{\text{s}}})\vartheta - \arg\max_{u} \phi^{\text{T}}(x_{i_{\text{s}}}, u)\theta_{\ell} \right|^2 \tag{5.17}$$

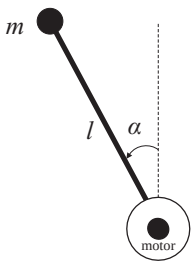
and by using the approximate, monotonous policy $\widehat{h}_{\ell}(x) = \varphi^{\text{T}}(x)\vartheta_{\ell}$ at every iteration $\ell > 0$. The problem (5.17) is solved using quadratic programming (see e.g., Nocedal and Wright, 2006).

5.6 Experimental studies

In this section, the performance of LSPI with orthogonal polynomial approximation is investigated, as well as the performance of online LSPI. In Section 5.6.1, the effects of using orthogonal polynomial approximation with LSPI are studied, for the inverted pendulum problem introduced in Section 4.6.3. In Section 5.6.2, the same problem is used to carry out an extensive experimental study of online LSPI. The effects of varying the tuning parameters of online LSPI are studied, and online LSPI is compared with offline LSPI. Then, online LSPI is used to learn how to control the real-life inverted pendulum. In Section 5.6.3, online LSPI is used to stabilize the two-link manipulator introduced in Section 4.6.2. Section 5.6.4 illustrates the effects of using prior knowledge about the monotonicity of the policy in online LSPI, for the servo-system problem introduced in Example 3.1.

5.6.1 LSPI with polynomial approximation for the inverted pendulum

In this section, the inverted pendulum problem introduced in Section 4.6.3 is used to study the performance of polynomial action approximation for offline LSPI. The inverted pendulum is obtained by placing an off-center weight on a disk driven by a DC motor, as shown in Figure 5.1. The disk rotates in a vertical plane. The control input is insufficient to push the pendulum up in a single rotation from every initial state. Instead, from certain states, the pendulum needs to be swung back and forth (destabilized) to gather energy, prior to being pushed up and stabilized.



(a) A schematic representation.

(b) The real system.

Figure 5.1: The inverted pendulum.

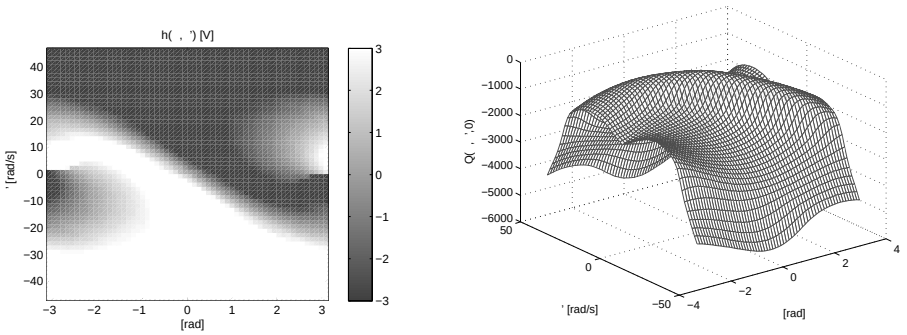
A model of the system dynamics is:

$$\ddot{\alpha} = \frac{1}{J}\left(mgl\sin(\alpha) - b\dot{\alpha} - \frac{K^2}{R}\dot{\alpha} + \frac{K}{R}u\right) \tag{5.18}$$

where $J = 1.91 \cdot 10^{-4}\,\text{kgm}^2$, $m = 0.055\,\text{kg}$, $g = 9.81\,\text{m/s}^2$, $l = 0.042\,\text{m}$, $b = 3 \cdot 10^{-6}\,\text{kg/s}$, $K = 0.0536\,\text{Nm/A}$, $R = 9.5\,\Omega$. The process is controlled using a sampling time $T_\text{s} = 0.005\,\text{s}$. The state is $x = [\alpha, \dot{\alpha}]^\text{T}$. The angle α varies in the interval $[-\pi, \pi)$ rad, with $\alpha = 0$ pointing up, and 'rolls over' so that e.g., a rotation of $3\pi/2$ corresponds to $\alpha = -\pi/2$. The velocity $\dot{\alpha}$ is restricted to the interval $[-15\pi, 15\pi]$ rad/s, and the control action u is constrained to $[-3, 3]$ V. The goal is to stabilize the pendulum in the unstable equilibrium $x = 0$ (pointing up), and is expressed by the reward function:

$$\begin{aligned} r_{k+1} &= \rho(x_k, u_k) = -x_k^\text{T} Q_\text{rew} x_k - R_\text{rew} u_k^2 \\ Q_\text{rew} &= \begin{bmatrix} 5 & 0 \\ 0 & 0.1 \end{bmatrix}, \quad R_\text{rew} = 1 \end{aligned} \tag{5.19}$$

The discount factor is $\gamma = 0.98$. This discount factor is sufficiently large to obtain a good control policy. A near-optimal solution (policy and Q-function) of this problem is given in Figure 5.2. This solution was computed with fuzzy Q-iteration with a fine grid of membership functions in the state space, and a fine discretization of the action space.



(a) A near-optimal policy.

(b) Projection of a near-optimal Q-function on X, for $u = 0$.

Figure 5.2: A near-optimal solution of the inverted pendulum swing-up problem.

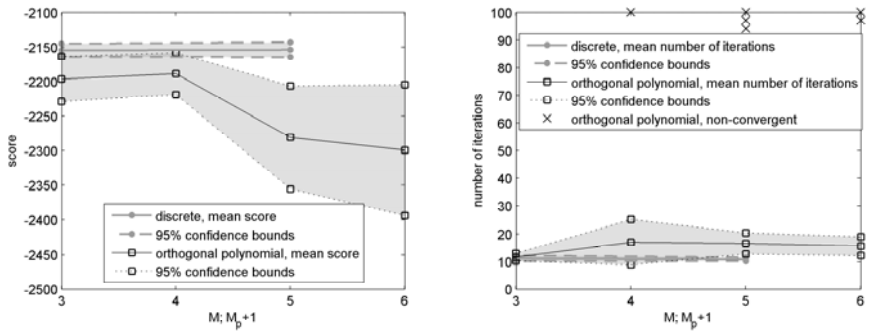
Identically shaped RBFs are used to approximate the Q-function over the state space. The RBFs are distributed on an equidistant 11×11 grid in the state space, and their radii along each dimension are identical to the distance between two adjacent RBFs along that dimension. This yields a smooth interpolation of the Q-function over the state space. Two sets of experiments are performed: one using discrete-action approximators of the type described in Example 5.1, and another using orthogonal polynomial approximation in the action space, as described in Section 5.3. For the discrete-action approximators, the number of discrete actions M takes two values: 3 and 5. The first set of actions is $\{-3, 0, 3\}$, and the second $\{-3, -0.7208, 0, 0.7208, 3\}$ (these five values are logarithmically spaced around the origin). The degree M_p of the polynomial approximators varies in $\{2, 3, 4, 5\}$.

The samples used for policy evaluation are random, uniformly distributed over the state-discrete action space $X \times U_\mathrm{d}$ for the discrete-action approximators, and over the state-continuous action space $X \times U$ for the polynomial approximators. A number $n_\mathrm{s} = 10000$ of samples is used for the approximators with $M = M_\mathrm{p} + 1 = 3$, and for the other experiments the number of samples is changed proportionally to the number n of parameters. Because N is constant, this means that n_s is proportional to M or respectively $M_\mathrm{p} + 1$. For instance, when $M_\mathrm{p} = 3$, $n_\mathrm{s} = 13334$ samples are used, and when $M = M_\mathrm{p} + 1 = 5$, $n_\mathrm{s} = 16667$. A number of 20 runs are performed for each experiment, with independent sets of samples. An experiment is considered convergent when the 2-norm of the difference between consecutive parameter vectors does not exceed $\varepsilon_\mathrm{LSPI} = 0.01$ (see Algorithm 5.2). The policy resulting from every convergent experiment are evaluated in simulation, by using it to control the system from every initial state on the grid:

$$X_0 = \{-\pi, -5\pi/6, -4\pi/6 \ldots, 0, \pi\} \times \{-10\pi, -9\pi, -8\pi, \ldots, 10\pi\} \tag{5.20}$$

The return obtained from each state on this grid is estimated with a precision of $\varepsilon_\mathrm{MC} = 0.1$, and then an average return over these states is computed. This average is the score (performance index) of the policy. The infinite-horizon return is estimated in a finite time by simulating only the first K steps of the trajectory, with K given by (2.27). Due to the form of the reward function, the score will always be negative; the performance is better when the absolute value of the score is smaller.

Figure 5.3(a) reports the mean performance across the 20 experiments, together with 95% confidence intervals for the mean performance. Figure 5.3(b) shows the number of iterations to convergence. The polynomial approximators share the same graphs with the discrete-action approximators, arranged so that the polynomial approximator of degree M_p has the same horizontal coordinate as the discrete-action approximator with $M = M_\mathrm{p} + 1$ discrete actions. This is done so that experiments sharing the same horizontal coordinate use the same number of parameters and state-action BFs.



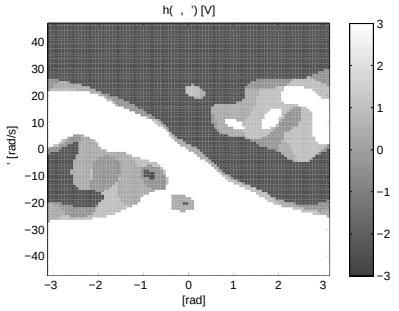
(a) Performance. (b) Convergence.

Figure 5.3: Comparison between discrete-action and orthogonal polynomial approximation. Non-convergent runs are excluded from the computation of the mean and of the confidence intervals.

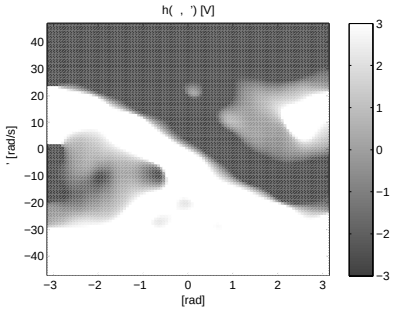
The differences between the performance of discrete-action approximation and the performance of polynomial approximation are inconclusive when the polynomial approximator

has a low degree. When the degree increases, the performance of the polynomial approximator becomes worse, and LSPI with polynomial approximation requires more iterations to converge than with discrete actions. The decrease in performance with the polynomial degree may be due to overfitting. Moreover, for some runs with polynomial approximation, LSPI does not converge to a fixed parameter vector. Each of these runs is marked along the top of Figure 5.3(b) with a symbol '$\times$', placed at the horizontal coordinate corresponding to the non-converging run.

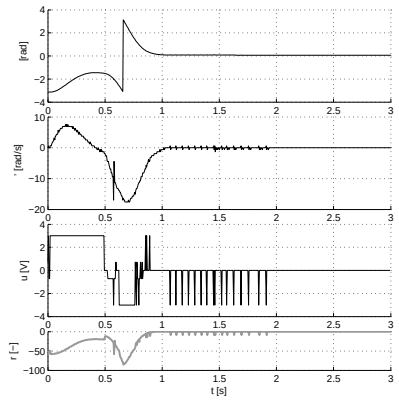
Figures 5.4(a) and 5.4(b) show representative policies computed with discrete and polynomial Q-function approximation, respectively. The discrete-action policy is computed using $M = 5$ discrete actions in the Q-function approximator, and the continuous-action policy is computed with a 2nd degree polynomial Q-function approximator. Figures 5.4(c) and 5.4(d) present real-time swingup trajectories of the inverted pendulum, controlled with the policies of Figures 5.4(a) and 5.4(b), and starting from the initial state $x_0 = [-\pi, 0]^{\mathrm{T}}$. These figures illustrate that continuous actions are useful to reduce chattering, even though Figure 5.3(a)



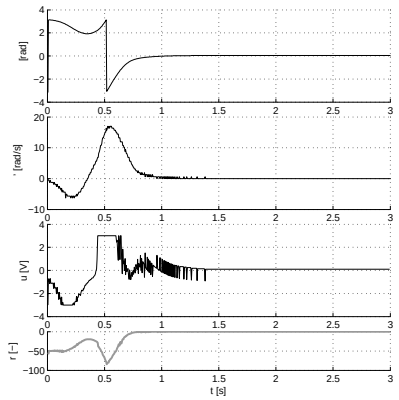
(a) Discrete-action policy, $M = 5$ actions.



(b) Continuous-action policy, $M_{\mathrm{p}} = 2$.



(c) Swingup of the real system using the discrete-action policy.



(d) Swingup of the real system using the continuous-action policy.

Figure 5.4: Representative policies and real-time swingup trajectories with discrete-action and polynomial approximation.

showed that the return obtained using continuous actions is not better than the return obtained with discrete actions. Because the real system has static friction, which is not modeled in (5.18), the equilibrium $x = [0, 0]^{\mathrm{T}}$ (pointing up) is stable in reality, and the discrete-action policy is also able to stabilize while applying zero action in steady state.

The worse results obtained by the polynomial approximator appear surprising at a first examination, because stabilizing the inverted pendulum around its unstable equilibrium leads to suboptimal chattering when continuous actions are not available. However, in this problem, the performance of discrete actions is difficult to improve upon. To understand why this is so, examine the initial part of the trajectories in Figures 5.4(c) and 5.4(d); this part can be closely approximated by a bang-bang control law, which can be realized using only three discrete actions: applying maximum voltages in either direction, or zero voltage. In the final part of the trajectory undesirable chattering occurs; however, due to the exponential discounting of the rewards, the negative rewards received due to the chattering have a small influence on the total return. Such a control sequence (which applies the action(s) with the maximum magnitude initially, followed by chattering) may be near-optimal from many initial states. The fact that a few discrete actions perform reasonably well can be seen in Figure 5.3(a), where increasing the number of discrete actions from 3 to 5 does not lead to a significant increase of the performance.

To better assess the potential of polynomial Q-function approximation, it should be evaluated also using other problems. For instance, when it is crucial to avoid chattering, a reward function that strongly penalizes chattering has to be designed. Furthermore, in problems with many action variables, independent discretization of the action variables may be unfeasible due to the exponential growth of the discrete action space with the number of action variables. In this case, it may be necessary to use a continuous-action approximator. Nevertheless, in the sequel, only discrete-action Q-function approximators will be used, for the reasons explained in Section 5.3.

5.6.2 Online LSPI for the inverted pendulum

In this section, we study experimentally the online LSPI algorithm, using the inverted pendulum problem of Section 5.6.1. First, extensive simulation experiments are performed to study the effects of the tuning parameters of online LSPI, and to compare online LSPI with the offline algorithm. Then, online LSPI is used in real time, to learn how to control the real inverted pendulum.

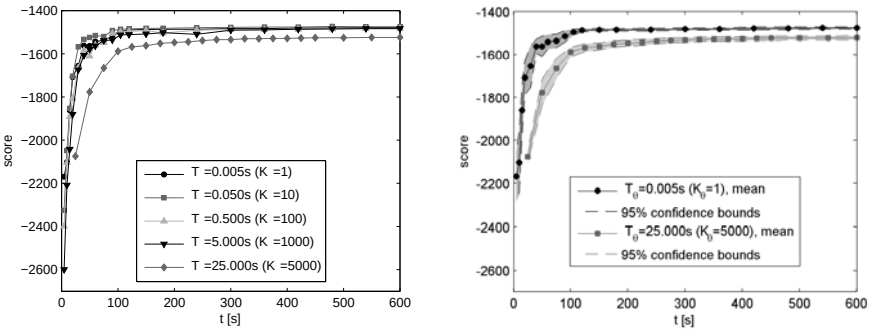
The RBF approximator with discrete actions introduced in Example 5.1 is used to approximate the Q-function. The 11×11 grid of RBFs described in Section 5.6.1 is employed. The discretized action space contains $M = 3$ actions: $U_{\mathrm{d}} = \{-3, 0, 3\}$. After each simulated online LSPI experiment is completed, snapshots of the current policy at increasing moments of time are evaluated. This produces a curve recording the performance of the policy as it evolves over time. During performance evaluation, exploration is turned off. When comparing online LSPI with offline LSPI, only the final policy in the experiment is evaluated. Policies are evaluated by estimating with a precision $\varepsilon_{\mathrm{MC}} = 0.1$ their average return over the grid[5] of initial states $X_0 = \{-\pi, -\pi/2, 0, \pi/2\} \times \{-10\pi, -3\pi, -\pi, 0, \pi, 3\pi, 10\pi\}$.

[5]To ensure that evaluating multiple policies for every run of online LSPI does not lead to excessive computational costs, this grid of initial states contains fewer elements than the grid (5.20), which was used to evaluate the final policies of offline LSPI.

Effects of the policy improvement interval

In this section, we study the effects of varying the number of transitions (samples) between consecutive policy improvements. This is a central parameter of the online LSPI algorithm. The following values will be used: $K_\theta \in \{1, 10, 100, 1000, 5000\}$, which given $T_\mathrm{s} = 0.005\,\mathrm{s}$ correspond to improving the policy once every 0.005, 0.05, 0.5, 5, and $25\,\mathrm{s}$, respectively. This time interval between two consecutive improvements is denoted by T_θ. The first experiment, for $K_\theta = 1$ ($T_\theta = T_\mathrm{s}$), corresponds to fully optimistic LSPI, where the policy is improved after every sample. Each experiment is run for $600\,\mathrm{s}$ of simulated time, and the trial length is set to $T_\mathrm{trial} = 1.5\,\mathrm{s}$. This trial length is sufficient for a good control policy to swing up and stabilize the inverted pendulum. In the beginning of each trial, the state is reset to uniform random values. The decaying exploration schedule (5.10) is used, with the initial exploration probability $\varepsilon_0 = 1$, chosen so that a fully random policy is used at first, and the decay rate $\varepsilon_\mathrm{d} = 0.9962$, chosen so that the exploration probability becomes 0.1 when $t = 600\,\mathrm{s}$. No experimental tuning was performed to choose these values.

A number of 20 independent runs are performed for each value of K_θ. Figure 5.5(a) shows how the performance of the policies learned by online LSPI evolves. Each curve represents the mean performance across the 20 runs. Figure 5.5(b) shows the mean performance with 95% confidence intervals, for the extreme values of K_θ, 1 and 5000.



(a) Mean performance for all the experiments.

(b) Mean performance with 95% confidence intervals, for the extreme values of K_θ.

Figure 5.5: The effects of the interval between policy improvements. The marker locations indicate the moments in time when the policies were evaluated.

The performance of the policies computed with online LSPI converges quickly, in roughly $120\,\mathrm{s}$. The algorithm is robust to changes in the K_θ parameter, with all the values resulting in a similar performance except when the policy is updated very rarely (once every 5000 samples). In this latter case, the performance is worse, and the difference from the performance obtained with small values of K_θ is statistically significant, as illustrated in Figure 5.5(b). This difference shows that policy improvements in online LSPI should not be performed too rarely.

Figure 5.6 shows the evolution of the difference between consecutive parameter vectors of online LSPI, for a few values of K_θ. The curves represent averages of the 20 independent runs. The parameter vector does not converge, even though the performance of the policy

does converge in Figure 5.5. The differences between parameters are smaller for smaller K_θ; this is probably because processing a smaller number of samples between updates adds less information to Γ and z, which means that subsequent solutions (parameter vectors) to $\Gamma\theta = z$ will be closer together.

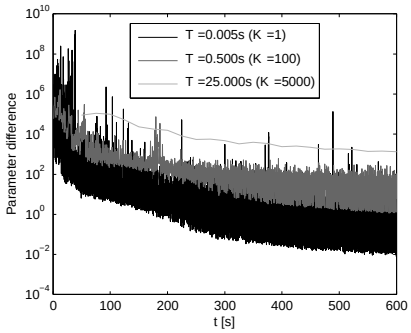


Figure 5.6: The evolution of parameter differences over time. Note that the parameter update frequencies are different between the three experiments. Parameter differences are computed with $\|\theta_\ell - \theta_{\ell-1}\|_2$.

Comparison of online LSPI and offline LSPI

In this section, the solutions obtained by online LSPI in the experiments of the previous section are compared with solutions obtained by offline LSPI. Offline LSPI employs the same approximator as online LSPI. A number of $n_\mathrm{s} = 20000$ random samples are used for offline LSPI, uniformly distributed throughout the state-discrete action space. Each experiment is run 20 times with independent sets of samples, and the policy computed in each of the runs is evaluated. Figure 5.7 compares the performance of the policies computed offline, with the performance of the policies at the end of each online experiment (at $t = 600\,\mathrm{s}$).

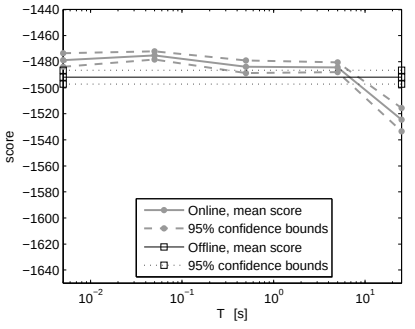
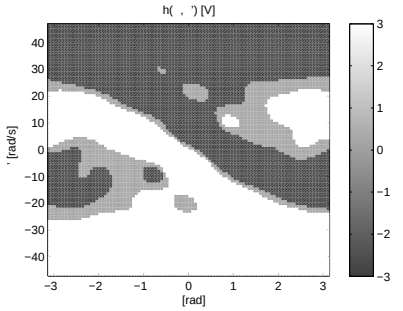
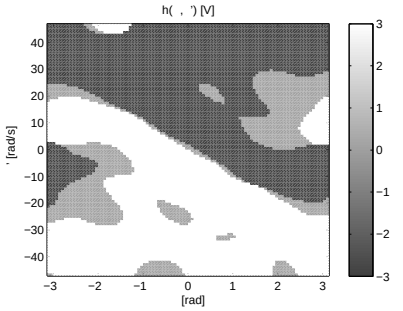


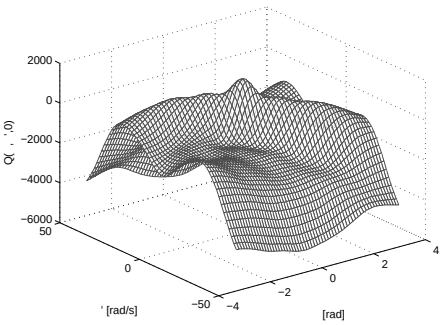
Figure 5.7: Comparison between the performance of offline and online LSPI. The horizontal lines for offline LSPI correspond to a single set of 20 experiments (offline LSPI does not use the parameter T_θ).



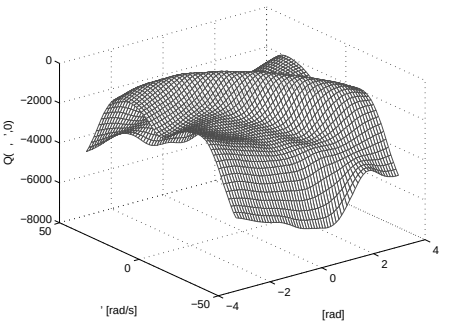
(a) Offline, policy.



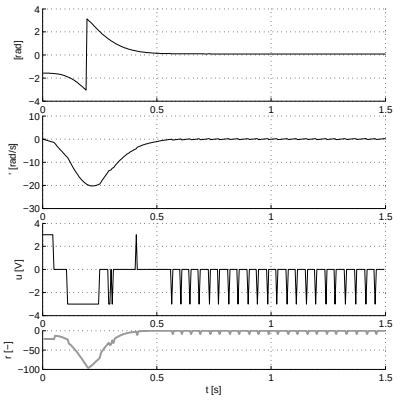
(b) Online, policy.



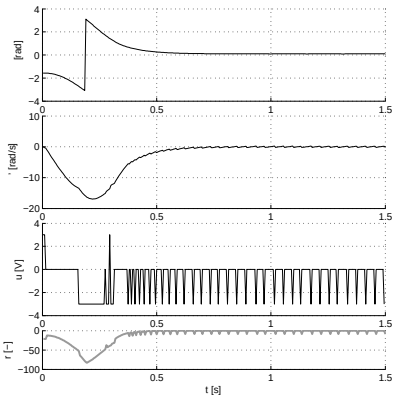
(c) Projection on X of the Q-function computed offline, for $u = 0$.



(d) Projection on X of the Q-function computed online, for $u = 0$.



(e) Offline, simulated trajectory from $x_0 = [-\frac{\pi}{2}, 0]^{\mathrm{T}}$.



(f) Online, simulated trajectory from $x_0 = [-\frac{\pi}{2}, 0]^{\mathrm{T}}$.

Figure 5.8: Representative solutions with offline and online LSPI.

The performance obtained by online LSPI is comparable with the performance of offline LSPI. For $K_\theta \in \{1, 10\}$ (corresponding to $T_\theta \in \{0.005, 0.05\}$ s), the online algorithm obtains better performance than the offline one. The only value of K_θ for which the performance is worse than in the offline case is $K_\theta = 5000$ ($T_\theta = 25$ s), which illustrates again that policy improvements should not be performed too rarely in online LSPI. Offline LSPI employs 20000 samples; the online algorithm processes the same number of samples in 120 s of simulated time, and 120000 samples during the entire learning process. Nevertheless, Figure 5.5 showed that the online performance is already good after 120 s, i.e., 20000 samples. Moreover, the offline algorithm loops through the samples once at every iteration, whereas the online algorithm processes samples only once. Therefore, the online algorithm compares favorably with the offline one, both in final performance and number of samples required to reach a good performance.

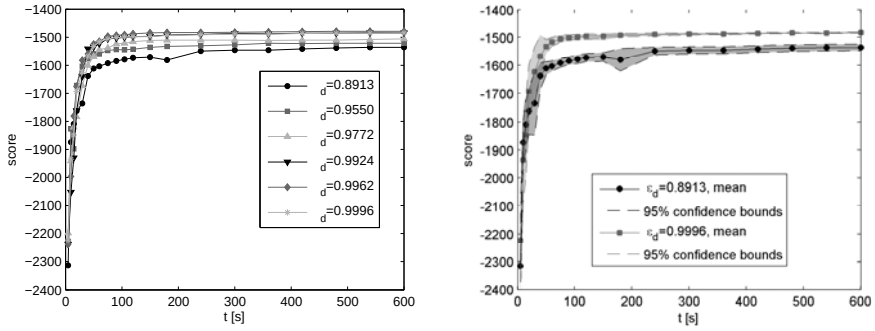
Figure 5.8 presents some representative solutions of offline and online LSPI. For online LSPI, a solution computed with $K_\theta = 10$ is selected. The online algorithm produces a smoother approximation of the Q-function (Figure 5.8(d)), which more closely resembles the near-optimal Q-function of Figure 5.2(b). The most important differences between the policies of Figures 5.8(a) and 5.8(b) occur in the regions with destabilizing actions. For instance, the offline policy indicates a zero action should be taken in the equilibrium $x = [-\pi, 0]^{\mathrm{T}}$. This is incorrect, because it means that the pendulum is kept immobile when $x = [-\pi, 0]^{\mathrm{T}}$. It also means that a controlled trajectory starting from $x_0 = [-\pi, 0]^{\mathrm{T}}$ would be uninformative; therefore, the initial state $x_0 = [-\frac{\pi}{2}, 0]^{\mathrm{T}}$ is selected to produce the (informative) trajectories of Figures 5.8(e) and 5.8(f). Note that this problem is not due to a limitation of the offline algorithm, but of the chosen approximator; sometimes, online LSPI also produces solutions with the same property.

Effects of the exploration schedule

In this section, we study the effects of the exploration schedule on the performance of online LSPI. Like in the earlier experiments, the decaying schedule (5.10) is used, with the initial exploration probability $\varepsilon_0 = 1$, so that a fully random policy is used at first. The following values of the decay rate ε_{d} are used: $\{0.8913, 0.9550, 0.9772, 0.9924, 0.9962, 0.9996\}$. They are chosen as follows. The first 5 values are chosen so that the exploration probability becomes 0.1 after, respectively: 20, 50, 100, 300, and 600 s. The last value leads to an exploration probability of 0.8 at $t = 600$ s, i.e., nearly all the actions taken during learning are exploratory. Larger values of ε_{d} correspond to more exploration. As for the study of the influence of K_θ, each experiment runs for 600 s of simulated time, the trial length is 1.5 s, and the state is reset to uniform random values in the beginning of each trial. For all the experiments, the policy is improved once every $K_\theta = 10$ samples. This value of K_θ gave a good performance in the earlier experiments.

A number of 20 independent runs are performed for each value of ε_{d}. Figure 5.9(a) presents the performance of the policies learned with online LSPI, averaged over the 20 runs. There is no discernible effect on the learning rate. However, when the exploration probability is too small (it decays to $\varepsilon = 0.1$ in 100 s or less), the final performance is worse. Given sufficient exploration, (i.e., when it decays to $\varepsilon = 0.1$ in 300 s or more), the final performance is good, and is robust to variations in the exploration decay rate. The difference between the performance of large and small exploration schedules is statistically significant, as illustrated in Figure 5.9(b). These results are expected, because the considerations in Section 5.4 already

indicated that a significant amount of exploration is essential for online LSPI. Note however that too much exploration will decrease the control performance obtained *during* learning; this effect is not visible in Figure 5.9, because exploration is turned off when the policies are evaluated.



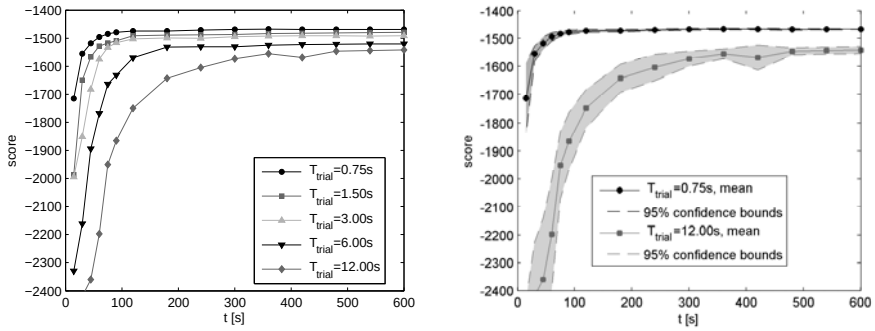
(a) Mean performance for all the experiments.

(b) Mean performance with 95% confidence intervals, for the extreme values of ε_{d}.

Figure 5.9: The effects of the exploration schedule.

Effects of the trial length

In this section, we study the effects of the trial length on the learning performance. The following values of the trial length will be used: $T_{\mathrm{trial}} \in \{0.75, 1.5, 3, 6, 12\}$ s. This corresponds to, respectively, 800, 400, 200, 100, and 50 learning trials in the 600 s of learning. The state is reset to uniform random values in the beginning of each trial. The policy is improved once every $K_\theta = 10$ samples. The initial exploration probability is $\varepsilon_0 = 1$, and the exploration decay rate is $\varepsilon_{\mathrm{d}} = 0.9962$, chosen so that the exploration probability decays to 0.1 in $t = 600$ s. These settings gave a good performance in our earlier experiments.



(a) Mean performance for all the experiments.

(b) Mean performance with 95% confidence intervals, for the extreme values of T_{trial}.

Figure 5.10: The effects of the trial length.

A number of 20 independent runs are performed for each value of T_{trial}. Figure 5.10(a) reports the performance of the policies learned by online LSPI, averaged over the 20 runs. Long trials (6 and 12 s) are detrimental to the learning rate, as well as to the final performance. Short trials do better because the more frequent resets to random states provide more information to the LSPI algorithm. This difference between the performance with short and long trials is statistically significant, as illustrated in Figure 5.10(b).

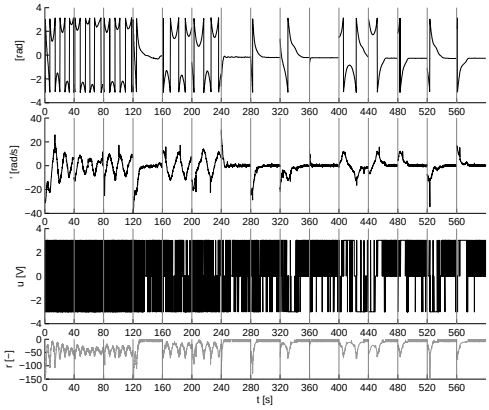
Online LSPI for the real inverted pendulum

In this section, the online LSPI algorithm is used to learn how to control the inverted pendulum system in real time, rather than in simulation as in the earlier sections. The same approximator is used as in the simulation experiments, and a single experiment is run for 600 s. The trials are 2 s long, the initial exploration probability is $\varepsilon_0 = 1$ and decays exponentially with $\varepsilon_{\text{d}} = 0.9924$, which leads to a final value of $\varepsilon = 0.01$ at $t = 600$ s. The exploration schedule was chosen smaller than in the simulation case, so that the real system is controlled better during learning.

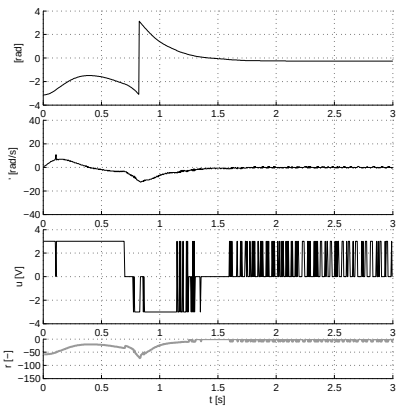
Some practical differences from the simulation experiments arise. Firstly, the state cannot be reset to arbitrary random values in the beginning of each trial, because a controller that is able to perform these resets is not available. Instead, a sequence of random actions is applied for 1 s in-between trials, to obtain a random initial state for the next trial. During these random sequences, no learning is performed, and the time required to apply them is not included in the learning time. In order to provide a more efficient exploration of the state space, the random actions are allowed to be more powerful, $u \in [-5, 5]$, than during learning, when they are limited to $[-3, 3]$. Secondly, policy improvements are performed only after the end of each trial, because solving the linear system in line 9 of Algorithm 5.2 at arbitrary discrete time steps may take longer than the (short) sampling time $T_{\text{s}} = 0.005$ s.

Figure 5.11(a) presents snapshots of the trajectories followed by the system during learning. These snapshots are taken once every 40 s, and include the effects of exploration. Figure 5.11(b) presents a swingup of the pendulum from the stable equilibrium (pointing down), with the final policy and without exploration. The controller learns how to swing up and stabilize the system, giving a good performance as early as 120 s into learning. This learning rate is similar to that observed in the simulation experiments. The final policy is able to swing the pendulum up effectively, although suboptimally.

Figure 5.12 examines the final policy obtained; compare it with the near-optimal policy of Figure 5.12(b) (repeated from Figure 5.2(a) for easy reference). For small velocities (around the zero coordinate on the vertical axis), the policy computed online roughly resembles the near-optimal policy, although its shape is different and suboptimal. The policy is not correct for large velocities. The reason for this becomes apparent in Figure 5.12(c), which also shows all the state samples collected during learning: with the procedure described above to reset the state in the beginning of each trial, the samples are concentrated in particular areas of the state space. The high-velocity regions are especially poorly represented in the set of samples. The performance would improve significantly if a (suboptimal) controller were available to reset the state to arbitrary locations. However, such a controller may be difficult to obtain in the model-free context of online LSPI.

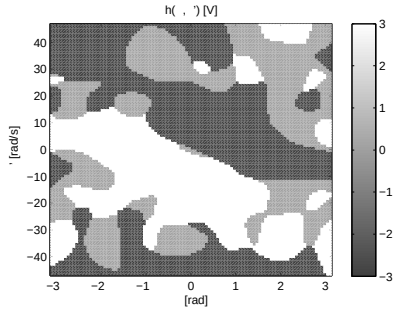


(a) During learning. Each trajectory is 2 s long. Only some of the trajectories are shown, separated by vertical lines. The starting time of each trajectory is given on the horizontal axis.

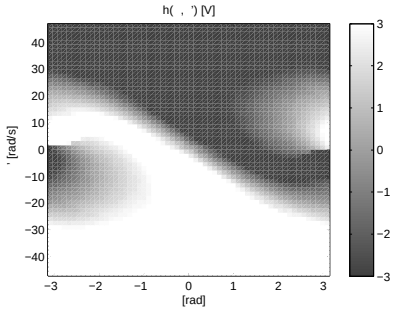


(b) A swingup with the final policy.

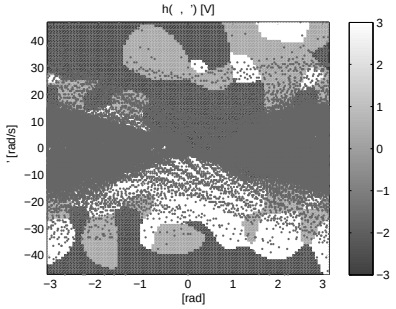
Figure 5.11: Trajectories of the real inverted pendulum.



(a) Online LSPI policy.



(b) Near-optimal policy, repeated from Figure 5.2(a).



(c) Policy with state samples added on top.

Figure 5.12: The policy learned by online LSPI on the real inverted pendulum, compared with a near-optimal policy.

5.6.3 Online LSPI for the two-link manipulator

In this section, online LSPI is applied to the stabilization of the two-link manipulator introduced in Section 4.6.2. This example illustrates that online LSPI is able to learn how to control systems with more state and action variables than those considered until this point, which only had two state variables and scalar actions.

The two-link manipulator, depicted in Figure 5.13, is described by the fourth-order nonlinear model:

$$M(\alpha)\ddot{\alpha} + C(\alpha, \dot{\alpha})\dot{\alpha} + G(\alpha) = \tau \tag{5.21}$$

where $\alpha = [\alpha_1, \alpha_2]^{\mathrm{T}}$, $\tau = [\tau_1, \tau_2]^{\mathrm{T}}$. The state signal contains the angles and angular velocities of the two links: $x = [\alpha_1, \dot{\alpha}_1, \alpha_2, \dot{\alpha}_2]^{\mathrm{T}}$, and the control signal is $u = \tau$. The angles wrap around in the interval $[-\pi, \pi]$ rad, so that e.g., a rotation of $3\pi/2$ corresponds to an angle of $-\pi/2$. The velocities are restricted to the interval $[-2\pi, 2\pi]$ rad/s, using saturation. The torques are limited so that $\tau_1 \in [-1.5, 1.5]$ Nm and $\tau_2 \in [-1, 1]$ Nm. In the sequel, the manipulator operates in a horizontal plane, leading to $G(\alpha) = 0$. For the values of the mass matrix $M(\alpha)$ and of the Coriolis and centrifugal forces matrix $C(\alpha, \dot{\alpha})$, see Section 4.6.2. The discrete time step is set to $T_{\mathrm{s}} = 0.05$ s, and the discrete-time dynamics f are obtained by numerical integration of (5.21) between the consecutive time steps.

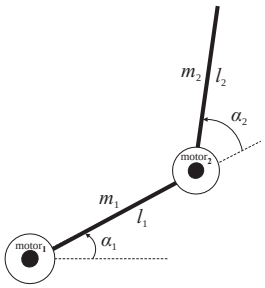


Figure 5.13: Schematic drawing of the two-link manipulator.

The control goal is the stabilization of the system around $x = 0$, and is expressed by the following quadratic reward function:

$$\rho(x, u) = -x^{\mathrm{T}} Q_{\mathrm{rew}} x, \quad \text{with } Q_{\mathrm{rew}} = \mathrm{diag}[1, 0.1, 1, 0.1] \tag{5.22}$$

The discount factor is set to $\gamma = 0.98$. This value is sufficiently large to obtain a good control policy.

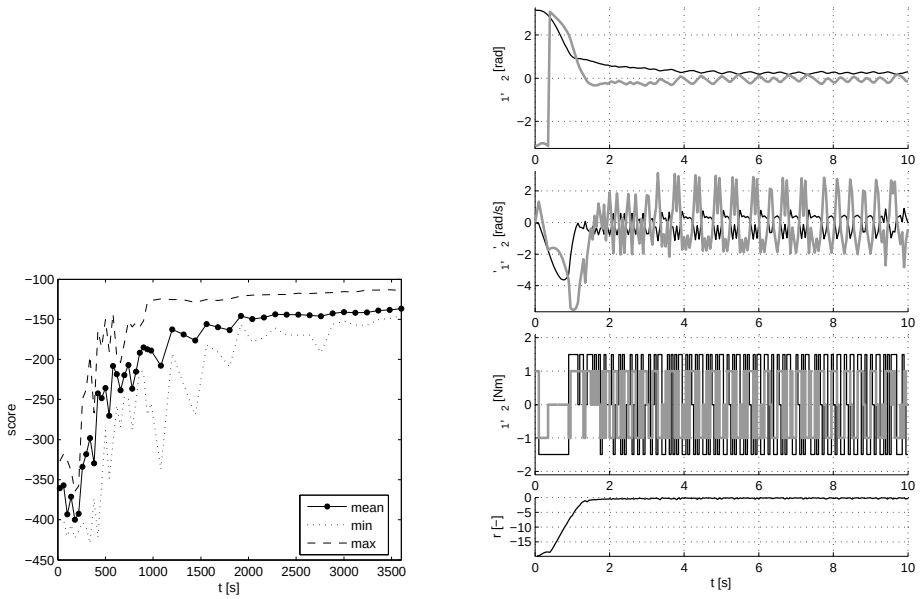
To apply online LSPI, the discrete-action RBF approximator of Example 5.1 is used, like for the inverted pendulum example. The centers of the RBFs are arranged on an equidistant grid with 5 RBFs on each axis of the state space; this leads to a total number of $N = 5^4 = 625$ RBFs. The radii of the RBFs along each dimension are taken identical to the distance along that dimension between adjacent RBFs. The discretized action space contains $M = 9$ actions, obtained by discretizing each torque variable using 3 values: $U_{\mathrm{d}} = \{-1.5, 0, 1.5\} \times \{-1, 0, 1\}$. Therefore, the total number of state-action BFs is $NM = 5625$ ($N = 625$ RBFs replicated $M = 9$ times, once for each discrete action). Online LSPI learns in simulation for 3600 s (one hour), and its parameters are set as follows: the policy is improved once

every $K_\theta = 50$ samples; the length of each trial is 10 s, which is sufficient for a good policy to stabilize the system; and the exploration probability decays from $\varepsilon_0 = 1$ with the decay factor $\varepsilon_\mathrm{d} = 0.9981$, which leads to $\varepsilon = 0.001$ at $t = 3600$ s. No experimental tuning was performed to choose these values. The performance of the policies computed online is evaluated using the average return over the set of initial states:

$$X_0 = \{-\pi, -2\pi/3, -\pi/3, ..., \pi\} \times \{0\} \times \{-\pi, -2\pi/3, -\pi/3, ..., \pi\} \times \{0\} \qquad (5.23)$$

This set contains a regular grid of initial angles. The initial angular velocities are always zero, to ensure that the set X_0 does not become too large and hence, does not lead to an excessive computational cost of evaluating the performance. The returns are estimated with a precision of $\varepsilon_\mathrm{MC} = 0.1$.

Figure 5.14(a) reports the mean, minimum, and maximum performance across five independent runs of the experiment. Fewer runs were performed than for the inverted pendulum example, to keep the computational cost of the experiments reasonable (because the number of BFs is very large, online LSPI is very computationally intensive). Figure 5.14(b) presents a trajectory controlled with the final policy obtained in one of the runs. The algorithm learns how to stabilize the system, but the control performance in Figure 5.14(b) is worse than the performance of fuzzy Q-iteration in Figure 4.9(b) of Section 4.6.2. Despite the worse performance from this particular initial state, the LSPI policy actually performs better on average, for the states in X_0, than the fuzzy Q-iteration policy of Section 4.6.2. While the performance of the final policies of online LSPI, averaged over the five experiments is -136.9, fuzzy Q-iteration has only obtained a performance of -217.2.



(a) Performance.

(b) Controlled trajectory from $x_0 = [\pi, 0, -\pi, 0]^\mathrm{T}$ (thin black line – link 1, thick gray line – link 2).

Figure 5.14: Results of online LSPI for the two-link manipulator.

The performance of online LSPI could probably be improved by increasing the number of equidistant RBFs and/or the action discretization. However, because of the very large number of BFs, the current approximator is already very complex. Although the simulated time is only one hour, an experiment takes more than 20 hours of CPU time to execute, which means that this approximator cannot be used in real time. Additionally, increasing the number of approximator parameters may require more samples, which increases the learning time. Therefore, making the approximator even more complex is not realistic. This difficulty shows that to apply online LSPI to complex systems with fast dynamics, it is essential to obtain a small number of BFs well suited to the problem at hand, rather than using a generic approximator with many equidistant BFs. Techniques to find good BFs automatically have been proposed for offline LSPI (Mahadevan and Maggioni, 2007; Xu et al., 2007). However, these techniques are geared towards the offline computation of BFs, and need to be modified to work in the online case. Developing new techniques that construct the BFs online is a promising alternative. One such technique has been proposed by Jung and Polani (2007) for online policy iteration with least-squares policy evaluation. This technique can easily be extended to online LSPI (which uses LSTD-Q instead of least-squares policy evaluation).

5.6.4 Online LSPI with prior knowledge for the servo-system

In this section, we investigate the effects of using prior knowledge about the monotonicity of the policy in online LSPI. The example used is the servo-system control task, introduced in Example 3.1 and used again in Section 4.6.1. A model of the servo-system is:

$$\begin{aligned} x_{k+1} &= f(x_k, u_k) = Ax_k + Bu_k \\ A &= \begin{bmatrix} 1 & 0.0049 \\ 0 & 0.9540 \end{bmatrix}, \quad B = \begin{bmatrix} 0.0021 \\ 0.8505 \end{bmatrix} \end{aligned} \tag{5.24}$$

This discrete-time model is obtained by discretizing a continuous-time model of the servo-system, which was determined by first-principles modeling of the real servo-system. The discretization is performed with the zero-order-hold method, using a sampling time of $T_\mathrm{s} = 0.005$ s. The position $x_{1,k} = \alpha$ is bounded to $[-\pi, \pi]$ rad, the velocity $x_{2,k} = \dot{\alpha}$ to $[-16\pi, 16\pi]$ rad/s, and the control input u_k to $[-10, 10]$ V. A discounted, quadratic regulation problem is considered, described by the reward function:

$$\begin{aligned} r_{k+1} &= \rho(x_k, u_k) = -x_k^\mathrm{T} Q_\mathrm{rew} x_k - R_\mathrm{rew} u_k^2 \\ Q_\mathrm{rew} &= \begin{bmatrix} 5 & 0 \\ 0 & 0.02 \end{bmatrix}, \quad R_\mathrm{rew} = 0.05 \end{aligned} \tag{5.25}$$

The discount factor is chosen $\gamma = 0.97$, which is sufficiently large to produce a good policy.[6]

Because the dynamics are linear (up to the saturation limits), and the reward function is quadratic, a linear state feedback policy for this problem can be computed using an extension of linear-quadratic regulation theory to the discounted case (Bertsekas, 2007), if the constraints on the states and actions are disregarded. A policy computed by applying this method to the servo-system problem is presented in Figure 5.15, where additionally the control input has been restricted to the admissible range $[-10, 10]$, using saturation. See Section 4.6.1

[6]Note that the weights Q_rew and R_rew, and the discount factor γ have different values than in the experiments of Section 4.6.1.

for details on the procedure to compute this policy. This policy is monotonously decreasing along both axes of the state space. This monotonicity property will be used in the sequel with online LSPI. Note that the actual values of the linear state feedback gains are not required to derive this prior knowledge. Instead, only the sign of these gains has to be known.

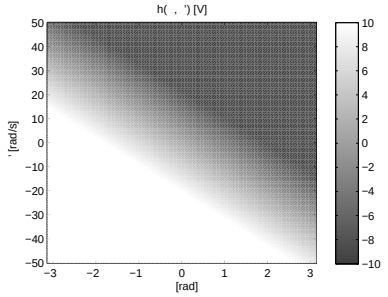


Figure 5.15: A near-optimal policy for the servo-system.

To assess the effects of using prior knowledge and parameterized policies, three experiments are performed. For these experiments, the Q-function is represented using the discrete-action RBF approximator introduced in Example 5.1.

1. In the first experiment, the original online LSPI algorithm is used, without prior knowledge. This algorithm improves the policy using (5.4), repeated here for easy reference:

$$h_{\ell+1}(x) = \arg\max_u \widehat{Q}^{h_\ell}(x,u) \tag{5.26}$$

   Such a policy is implicitly defined by the Q-function, and always produces discrete actions with the discrete-action Q-function approximator employed.

2. In the second experiment, an approximate policy that is linear in the parameters ϑ (5.11) is used. Policy improvements are constrained using the prior knowledge about the monotonicity of the policy. The improved policy parameter is therefore computed with (5.17), repeated here for easy reference:

$$\vartheta_{\ell+1} = \arg\min_{\vartheta\in\mathbb{R}^{\mathcal{N}},\ \Delta_{\mathrm{mon}}\vartheta\leq 0_{N_{\mathrm{mon}}}} \sum_{i_\mathrm{s}=1}^{\mathcal{N}_\mathrm{s}} \left| \varphi^\mathrm{T}(x_{i_\mathrm{s}})\vartheta - \arg\max_u \phi^\mathrm{T}(x_{i_\mathrm{s}},u)\theta_\ell \right|^2 \tag{5.27}$$

   where $\Delta_{\mathrm{mon}}\vartheta \leq 0_{N_{\mathrm{mon}}}$ are the monotonicity constraints. The resulting approximate policies produce continuous actions, but these actions have to be discretized during learning, for the reasons explained in Section 5.5.1. Comparing the results of this experiment with the results of Experiment 1 allows us to assess the effects of using prior knowledge in online LSPI.

3. In the third experiment, a linearly parameterized, approximate policy (5.11) is used, but this policy is improved without using prior knowledge. The improved policy parameter is therefore computed with (5.12), repeated here for easy reference:

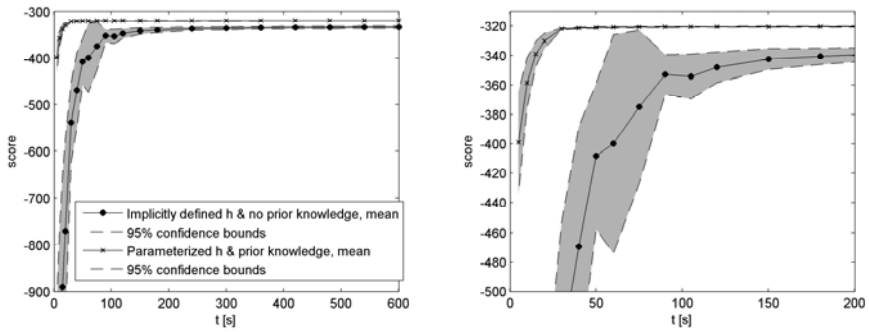
$$\vartheta_{\ell+1} = \arg\min_{\vartheta\in\mathbb{R}^{\mathcal{N}}} \sum_{i_\mathrm{s}=1}^{\mathcal{N}_\mathrm{s}} \left| \varphi^\mathrm{T}(x_{i_\mathrm{s}})\vartheta - \arg\max_u \phi^\mathrm{T}(x_{i_\mathrm{s}},u)\theta_\ell \right|^2 \tag{5.28}$$

As for Experiment 2 above, the resulting policies produce continuous actions which have to be discretized during learning. Comparing the results of this experiment with the results of Experiment 1 allows us to isolate the effects of using parameterized policies, in the absence of prior knowledge.

The centers of the RBFs are arranged on an equidistant 9×9 grid in the state space, and the radii of the RBFs along each dimension are taken identical to the distance along that dimension between two adjacent RBFs. The discretized action space contains $M = 3$ actions: $U_{\mathrm{d}} = \{-10, 0, 10\}$. When approximate policies are used, the policy improvements (5.27) and (5.28) use 1000 pre-generated, random and uniformly distributed state samples. These samples do not include information about transitions, so the algorithm can still be applied online and without requiring a model. The algorithm learns for $600\,\mathrm{s}$, and this interval is divided in trials having a length of $T_{\mathrm{trial}} = 1.5\,\mathrm{s}$. The initial states at the beginning of each trial are random and uniformly distributed. The following learning parameters are used: $K_\theta = 100$, $\varepsilon_0 = 1$, $\varepsilon_{\mathrm{d}} = 0.9962$ (leading to an exploration probability of 0.1 at $t = 600\,\mathrm{s}$). Policies are evaluated by estimating with a precision $\varepsilon_{\mathrm{MC}} = 0.1$ their average return over the grid of initial states:

$$X_0 = \{-\pi, -\pi/2, 0, \pi/2, \pi\} \times \{-10\pi, -5\pi, -2\pi, -\pi, 0, \pi, 2\pi, 5\pi, 10\pi\}$$

A number of 20 independent runs are performed for each of the three experiments. Figure 5.16 compares online LSPI with prior knowledge and parameterized policies (Experiment 2), with online LSPI without prior knowledge and policies implicitly derived from Q-functions (Experiment 1). The performance of parameterized policies is evaluated by using continuous-action control (5.11). Using prior knowledge leads to much faster learning, when compared to online LSPI without prior knowledge. With prior knowledge, the algorithm converges in under $40\,\mathrm{s}$, whereas without prior knowledge it requires more than $100\,\mathrm{s}$. The monotonous, parameterized policies also perform better than the discrete-action policies implicitly derived from the Q-functions.

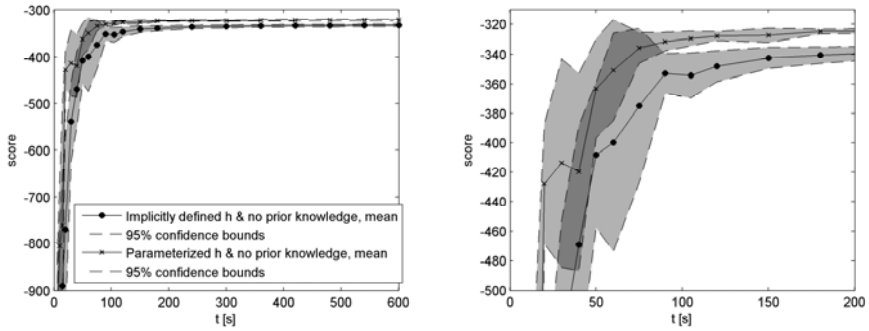


(a) Performance. (b) Magnification of (a).

Figure 5.16: Comparison between online LSPI with prior knowledge (Experiment 2) and the original online LSPI algorithm (Experiment 1).

Figure 5.17 compares online LSPI without prior knowledge but with parameterized policies (Experiment 3), with online LSPI without prior knowledge and policies implicitly de-

rived from Q-functions (Experiment 1). Parameterizing policies without using prior knowledge does not change the learning rate, but, like in Experiment 2, the parameterized policies perform better than the policies implicitly defined by the Q-function. This is because the parameterized policies produce continuous actions, whereas policies implicitly defined by the Q-function can only produce discrete actions.



(a) Performance.

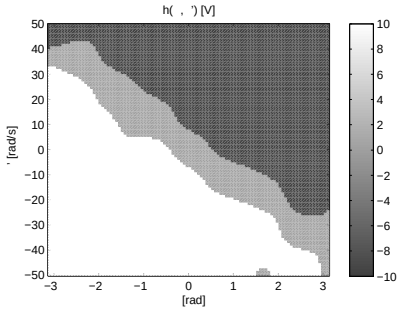
(b) Magnification of (a).

Figure 5.17: Comparison between parameterized policies computed without prior knowledge (Experiment 3) and policies computed with the original algorithm (Experiment 1).

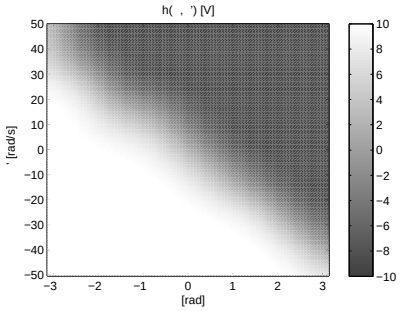
Figure 5.18 compares a representative solution obtained using prior knowledge and parameterized policies, with a representative solution obtained with implicit policies. The controlled trajectories are obtained in simulation. The policy obtained without using prior knowledge violates monotonicity in several areas. Because it can apply continuous actions, the control performance of the monotonous policy is better.

In Figures 5.16 and 5.17, the performance of the parameterized policies was evaluated using continuous-action control (5.11). However, the actions taken during learning have to be quantized with (5.13), for the reasons explained in Section 5.5.1. To offer a clearer image of the performance that can be achieved during learning, the performance of the parameterized policies is evaluated again, but this time the continuous actions produced by the policies are quantized into the set $U_{\mathrm{d}} = \{-10, 0, 10\}$. Recall that the policies computed with the original online LSPI algorithm (5.26) already produce discrete actions in this set, so their performance does not have to be reevaluated.

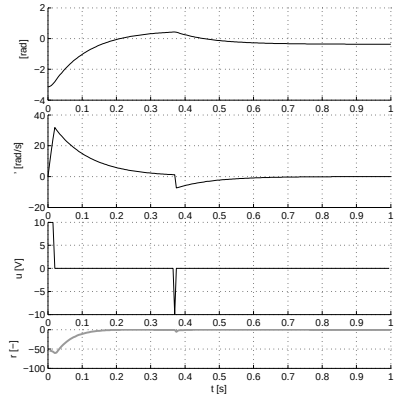
Figure 5.19 compares Experiment 2, which uses prior knowledge, with Experiment 1, for the case of quantized policies (compare with Figure 5.16). Figure 5.20 compares Experiment 3, which uses parameterized policies, but no prior knowledge, with Experiment 1, for the case of quantized policies (compare with Figure 5.17). The differences in the performance between the parameterized policies, and the policies implicitly defined by the Q-function, diminishes when the parameterized policies are quantized. However, the learning rate advantage of online LSPI with prior knowledge is still apparent.



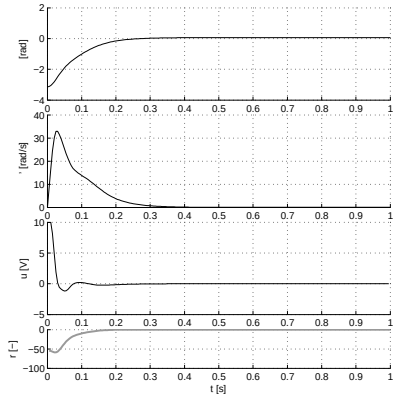
(a) Policy computed with the original online LSPI algorithm, without prior knowledge.



(b) Monotonous policy computed using prior knowledge.

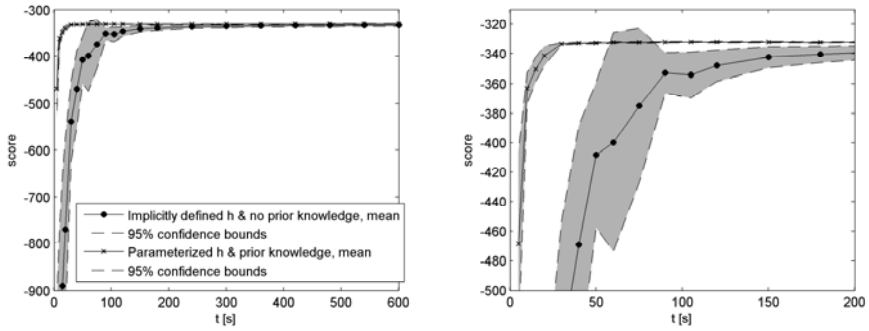


(c) Trajectory controlled with the policy of (a).



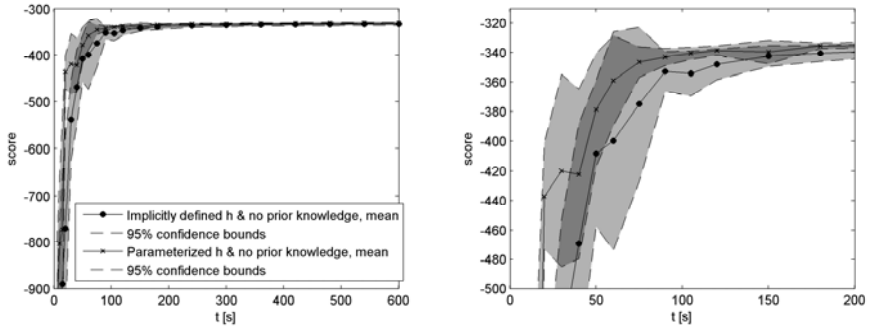
(d) Trajectory controlled with the policy of (b).

Figure 5.18: Representative solutions without prior knowledge (Experiment 1) and with prior knowledge (Experiment 2).



(a) Performance. (b) Magnification of (a).

Figure 5.19: Comparison between the *quantized*, parameterized policies computed using prior knowledge (Experiment 2) and the already discrete-action policies computed with the original algorithm (Experiment 1).



(a) Performance. (b) Magnification of (a).

Figure 5.20: Comparison between *quantized*, parameterized policies computed without using prior knowledge (Experiment 3) and the already discrete-action policies computed with the original algorithm (Experiment 1).

5.7 Conclusions and open issues

This chapter has extended least-squares policy iteration (LSPI) to online learning. In experiments with an inverted pendulum swingup problem, online LSPI has converged to a performance similar to that obtained by offline LSPI, without requiring more samples to reach this performance. Online LSPI has also exhibited a good robustness to variations in its tuning parameters, and has been able to learn in real time how to control the (real) inverted pendulum. We have also described a variant of online LSPI that uses prior knowledge about the monotonicity of the policy in the state variables. In an example involving the control of a servo-system, using this type of prior knowledge has sped up online LSPI by at least a factor of two. Overall, online LSPI is a promising algorithm for online RL.

We have also investigated a continuous-action, polynomial Q-function approximator for

offline LSPI. In the inverted pendulum problem, this approximator has reduced chattering, although it has not improved upon the performance (measured using the discounted return) obtained with discrete actions. To better assess the potential of polynomial Q-function approximation, it is important to also evaluate it in other problems.

The experiments in Section 5.6 have shown that the parameter vector of online LSPI does not converge in general. Moreover, the asymptotical performance guarantees of offline policy iteration rely on bounded errors in the policy evaluation and policy improvement steps. Because online LSPI improves the policy before an accurate estimation of the value function is available, the policy evaluation error can be very large, and the performance guarantees for offline policy iteration are not useful in the online case. Analyzing whether the asymptotical performance of online LSPI can be guaranteed is an important research topic.

In problems with a large number of state and action variables, the use of generic, equidistant BFs leads to excessive computational costs. This has already been apparent for the two-link manipulator, which has four state variables and two action variables. To reduce the computational cost, the Sherman-Morrison formula (Sherman and Morrison, 1950) can be used to incrementally compute the inverse of the matrix Γ in the linear system (5.3), thus eliminating the need to solve the linear system explicitly during policy improvement. Another possibility is to use gradient updates of the Q-function parameters, instead of solving a linear system of equations. Such an incremental least-squares temporal difference algorithm was proposed by Geramifard et al. (2006, 2007). While these solutions may significantly speed up the computations, they do not address the core issue: the exponential growth of the computational complexity with the size of the problem. It may be possible to limit this growth by automatically discovering a small number of good BFs. Methods to do this have been developed for offline LSPI (Mahadevan and Maggioni, 2007; Xu et al., 2007), but they are geared towards the offline construction of BFs, and would need to be modified to work in the online case. Techniques that construct the BFs online, such as the growing kernel approach of Jung and Polani (2007), are a promising alternative.

Online LSPI uses the least-squares temporal difference algorithm for policy evaluation. Other policy evaluation techniques can be used for online learning, as well. One alternative is the least-squares policy evaluation algorithm. It was originally developed for V-functions (Bertsekas and Ioffe, 1996; Bertsekas, 2007), but we extend it to Q-functions in Appendix B. Least-squares temporal difference and least-squares policy evaluation can behave quite differently when they compute Q-functions online, just like when they compute V-functions online (Bertsekas, 2007). Therefore, it is important to determine which of these two algorithms performs better in online learning.

Although in this chapter online LSPI has only been applied to time-invariant processes, it may also be able to deal with processes that are slowly changing over time, by adapting the policy to take these changes into account.

Chapter 6

Cross-entropy policy search

This chapter introduces an approximate policy search algorithm that works for continuous states and discrete actions. We use a highly flexible policy parameterization able to solve general problems, unlike previous approaches from the literature, which typically use ad-hoc parameterizations developed for specific problems. The algorithm looks for the best closed-loop policy that can be represented using a set of basis functions, where a discrete action is assigned to each basis function. The type and number of basis functions are specified in advance. The locations and shapes of the basis functions are optimized, together with the action assignments. The optimization is carried out with the cross-entropy method and evaluates the policies by their empirical return from a representative set of initial states. Simulation experiments are carried out on problems involving two to six state variables. In these experiments, the algorithm reliably obtains good policies, requiring only a small number of basis functions.

6.1 Introduction

The previous two chapters have considered techniques that compute value functions and use these value functions to find policies. The present chapter concerns direct policy search techniques, which aim to compute a policy directly, without using value functions. The main motivation for direct policy search is that techniques that rely on value functions do not scale well to high-dimensional problems. Although a general statement cannot be made about this limitation of value function techniques, most state of the art algorithms relying on value functions have been applied to problems with up to six continuous state variables (Munos and Moore, 2002; Lagoudakis and Parr, 2003a; Ernst et al., 2005; Mahadevan and Maggioni, 2007). Moreover, approximating value functions is often a difficult, counterintuitive task. For instance, it might be necessary that the value function is represented accurately in some regions of the state space, to have a good accuracy of the policy in another region (Munos and Moore, 2002).

In policy search, a class of parameterized policies is chosen, and an optimal parameter vector is sought that maximizes the performance of the policy. In the literature, typically ad-hoc policy parameterizations with a few parameters are designed for specific problems, using intuition and prior knowledge about the optimal policy (Marbach and Tsitsiklis, 2003;

Munos, 2006; Riedmiller et al., 2007). On the other hand, in the area of value function approximation, significant efforts have been invested to develop techniques that automatically find good approximators, without relying on prior knowledge (Munos and Moore, 2002; Mahadevan, 2005; Keller et al., 2006; Ernst et al., 2005; Mahadevan and Maggioni, 2007). These techniques have been reviewed in Section 3.5. A majority of these techniques represent value functions using a linear combination of basis functions (BFs), and automatically find both the BFs and the linear weights.

The aim of this chapter is to develop and evaluate highly flexible policy approximators for direct policy search. These approximators can be used to solve a large class of Markov decision processes (MDPs), unlike previous approaches from the literature, which typically use ad-hoc parameterizations developed for specific problems. The action space of the MDP is assumed discrete (or discretized). Inspired by the work on the optimization of BFs for value function approximation, we propose to represent policies using $\mathcal{N}$ state-dependent BFs, where the BFs are associated to discrete actions by a many-to-one mapping. The type of BFs and their number $\mathcal{N}$ are specified in advance and determine the approximation power of the representation. The locations and shapes of the BFs, together with the action assignments, are the parameters subject to optimization. The optimization criterion is a weighted sum of the returns from a set of representative initial states, where each return is computed with Monte Carlo simulations (3.34). The representative states can be uniformly distributed and equally weighted. Alternatively, the representative states together with the weight function can be used to focus the algorithm on important parts of the state space. This is in contrast to other policy search techniques, which optimize the average reward over the entire state space (Marbach and Tsitsiklis, 2003; Konda and Tsitsiklis, 2003), or the returns from every point in the discrete state space (Mannor et al., 2003). In general, it is impossible to take into account the return from every state, because the state space is continuous.

It is important to note that because of the rich policy parameterization, the performance may be a complicated function of the parameter vector, e.g., non-differentiable, non-convex, with many local optima. One way to address this problem is to rely on global optimization techniques. Here, the cross-entropy (CE) method for optimization (Rubinstein and Kroese, 2004) is selected as an illustrative example from the wide range of techniques that could be applied to this problem. Other options include e.g., genetic algorithms, tabu search, pattern search, etc. Although our approach can be easily extended to hybrid state spaces, where some variables are continuous and others discrete, for simplicity we only consider continuous state spaces. Extensive numerical evaluations of the proposed technique are conducted on a simple example with linear dynamics, as well as on two nonlinear control problems with four and six state variables, respectively.

Like CE policy search, fuzzy Q-iteration (Chapter 4) and online least-squares policy iteration (Chapter 5) have used BFs to construct their approximators. However, fuzzy Q-iteration and online least-squares policy iteration have employed *fixed* BFs, whereas CE policy search *optimizes* the BFs. Moreover, because the BFs themselves are parameterized, and also due to the discrete nature of the BF-action mapping, the policy approximator is highly nonlinear, while the approximators of Chapters 4 and 5 are linearly parameterized. The flexibility of CE policy search lies in this nonlinear parameterization, which can be used to represent a large class of policies.

CE policy search is a model-based, dynamic programming (DP) approach. However, it is less restrictive than typical DP techniques, in the sense that it only requires trajectories to be

generated from every representative state, whereas typical DP techniques require samples to be generated from arbitrary states. CE policy search is nevertheless more restrictive than typical reinforcement learning (RL) techniques, which can work with a single trajectory without requiring any resets of the state.

The remainder of this chapter is structured as follows. Section 6.2 gives a brief overview of the literature related to our approach. Section 6.3 describes the CE method for rare-event simulation and optimization. In Section 6.4, the CE algorithm for policy search is introduced and instantiated for a policy representation using radial basis functions (RBFs). Section 6.5 reports the results of extensive numerical experiments, in which CE policy search is used to control a double integrator, to balance a bicycle riding at constant speed, and to control the treatment of infection with the human immunodefficiency virus (HIV). Using the double integrator problem, CE policy search is compared to the fuzzy Q-iteration algorithm of Chapter 4. Section 6.6 summarizes the results of the numerical experiments and concludes the chapter.

6.2 Related work

Gradient-based optimization has been the focus of much theoretical and empirical work in policy search (Baird and Moore, 1999; Sutton et al., 2000; Marbach and Tsitsiklis, 2003; Konda and Tsitsiklis, 2003; Munos, 2006; Riedmiller et al., 2007). Such work is based on the assumption that the locally optimal solution found by the gradient method is good enough, which may be true when the policy parameterization is simple and well suited for the particular problem. Because our policy parameterization is very rich, the optimization criterion is likely to have many local optima and may also be non-differentiable. Gradient techniques are therefore unsuitable in our case. We select the (global) CE method for optimization from the plethora of available global, gradient-free optimization techniques. Another global optimization method that has been applied to policy search is evolutionary computation, both when a model is available (Barash, 1999; Chin and Jafari, 1998; Chang et al., 2007, Chapter 3) and when it is not (Gomez et al., 2006).

The use of the CE method for policy optimization was introduced by Mannor et al. (2003). Chang et al. (2007, Chapter 4) recently proposed finding a policy with the model-reference adaptive search, which is closely related to the CE method. Both works focus on solving finite, small MDPs, although they also propose solving large MDPs with parameterized policies. In contrast, we focus on highly flexible policy parameterizations to solve *continuous-state* MDPs. Additionally, we consider representative states associated with weights as a tool to focus the optimization on important initial states, and as a way to circumvent the need to estimate returns for every value of the state, which is impossible when the states are continuous. Chang et al. (2007, Chapter 4) only optimized the return starting from one initial state, whereas Mannor et al. (2003) optimize the returns from every (discrete) initial state.

Our policy parameterization is inspired by the techniques to automatically find BFs for value function approximation. These BF construction techniques were reviewed in Section 3.5. They include among others local and global refinement (Munos and Moore, 2002; Ratitch and Precup, 2004), optimization (Menache et al., 2005; Whiteson and Stone, 2006), regression techniques (Ernst et al., 2005), and spectral analysis of the transition function (Mahadevan, 2005; Mahadevan and Maggioni, 2007). Out of these options, we choose optimization, but we search for a policy approximator, rather than a value function approximator.

Instead of minimizing the Bellman residual (3.33), as is typically done to find value function BFs (Menache et al., 2005), we maximize the empirical return from the representative states. In value function approximation, changing the parameters of the BFs while running the DP/RL algorithm can lead to a loss of convergence. In our policy search approach, the BF parameters can be optimized without causing convergence problems.

Although most of the proposed approaches to find value function BFs cannot be used to find policy BFs, one possible exception is the spectral analysis technique of Mahadevan (2005); Mahadevan and Maggioni (2007). This technique constructs global BFs that obey the topology of the MDP state transitions, and uses these BFs to estimate approximate value functions. Such BFs could also be used to parameterize the policy. This possibility has not been explored yet.

Note that the concept of representative states used in this chapter is different from the one used in approximate value iteration with representative states (Bertsekas and Tsitsiklis, 1996, Chapter 6). There, an auxiliary MDP with the state space consisting of the representative states is solved. Then, the solution is extended to the original MDP. In this chapter, the return from the representative states is optimized, but the policies considered are defined over the entire state space.

Our method uses Monte Carlo simulations to estimate the performance of policies, and in this sense it is related to rollout policy iteration (Lagoudakis and Parr, 2003b; Bertsekas, 2005). A rollout is the computation of a Monte Carlo estimate of the V-value of a state, or of the Q-value of a state-action pair. Performing rollouts for many states or state-action pairs, and using the resulting value function estimate to perform policy improvement, leads to rollout policy iteration. Rollouts are a way to find value functions without solving the Bellman equation; in our approach, we do not use value functions at all.

6.3 Cross-entropy optimization

This section provides a brief introduction to the CE method for optimization (Rubinstein and Kroese, 2004). For a full description, see Appendix A. This introduction is similar to that given in Section 4.5.1. Consider the following optimization problem:

$$\max_{a \in \mathcal{A}} s(a) \tag{6.1}$$

where $s : \mathcal{A} \to \mathbb{R}$ is the score function to maximize, and the variable a takes values in the domain $\mathcal{A}$. Denote the maximum by s^*. The CE method for optimization maintains a probability density with support $\mathcal{A}$. In each iteration, a number of samples are drawn from this density and the score values for these samples are computed. A (smaller) number of samples that have the highest scores are kept, and the remaining samples are discarded. The probability density is then updated using the selected samples, such that at the next iteration the probability of drawing better samples is increased. The algorithm stops when the score of the worst selected sample no longer improves significantly.

Formally, a family of probability densities[1] $\{p(\cdot; v)\}$ has to be chosen, where the dot stands for the random variable. This family has support $\mathcal{A}$ and is parameterized by v. In each

[1] For simplicity, we will abuse the terminology by using the term 'density' to refer to probability density functions (which describe probabilities of continuous random variables), as well as to probability mass functions (which describe probabilities of discrete random variables).

iteration $\tau \geq 1$ of the CE algorithm, a number N_{CE} of samples is drawn from the density $p(\cdot; v_{\tau-1})$, their scores are computed, and the $(1 - \varrho_{\text{CE}})$ quantile[2] λ_τ of the sample scores is determined, with $\varrho_{\text{CE}} \in (0, 1)$. Then, a so-called associated stochastic problem is defined, which involves estimating the probability that the score of a sample drawn from $p(\cdot; v_{\tau-1})$ is at least λ_τ:

$$\mathrm{P}_{a\sim p(\cdot;v_{\tau-1})}(s(a) \geq \lambda_\tau) = \mathrm{E}_{a\sim p(\cdot;v_{\tau-1})}\left\{I(s(a) \geq \lambda_\tau)\right\} \tag{6.2}$$

where I is the indicator function, equal to 1 whenever its argument is true, and 0 otherwise.

The probability (6.2) can be estimated by importance sampling. For this problem, an importance sampling density is one that increases the probability of the interesting event $s(a) \geq \lambda_\tau$. The best importance sampling density in the family $\{p(\cdot; v)\}$, in the smallest cross-entropy sense, is given by the solution of:

$$\arg\max_v \mathrm{E}_{a\sim p(\cdot;v_{\tau-1})}\left\{I(s(a) \geq \lambda_\tau) \ln p(a; v)\right\} \tag{6.3}$$

An approximate solution of (6.3) is computed with the so-called stochastic counterpart:

$$v_\tau = \arg\max_v \frac{1}{N_{\text{CE}}} \sum_{i_{\text{s}}=1}^{N_{\text{CE}}} I(s(a_{i_{\text{s}}}) \geq \lambda_\tau) \ln p(a_{i_{\text{s}}}; v) \tag{6.4}$$

CE optimization then proceeds with the next iteration using the new density parameter v_τ (the probability (6.2) is never actually computed). The updated density aims at generating good samples with higher probability, thus bringing $\lambda_{\tau+1}$ closer to the optimum s^*. The goal is to eventually converge to a density which generates samples close to optimal value(s) of a with very high probability. The highest score among the samples generated in all the iterations is taken as the approximate solution of the optimization problem, and the corresponding sample as an approximate location of the optimum. The algorithm is stopped when the improvement in the $(1 - \varrho_{\text{CE}})$-quantile does not exceed ε_{CE} for d_{CE} successive iterations, or when a maximum number of iterations τ_{max} is reached.

Under certain assumptions on $\mathcal{A}$ and $p(\cdot; v)$, the stochastic counterpart (6.4) can be solved analytically. One particularly important case when this happens is when $p(\cdot; v)$ belongs to the natural exponential family. For instance, when $\{p(\cdot; v)\}$ is the family of Gaussians parameterized by the mean η and the standard deviation σ (so, $v = [\eta, \sigma]^{\mathrm{T}}$), the solution of the stochastic counterpart is the mean and the standard deviation of the best samples:

$$\eta_\tau = \frac{\sum_{i_{\text{s}}=1}^{N_{\text{CE}}} I(s(a_{i_{\text{s}}}) \geq \lambda_\tau) a_{i_{\text{s}}}}{\sum_{i_{\text{s}}=1}^{N_{\text{CE}}} I(s(a_{i_{\text{s}}}) \geq \lambda_\tau)} \tag{6.5}$$

$$\sigma_\tau = \sqrt{\frac{\sum_{i_{\text{s}}=1}^{N_{\text{CE}}} I(s(a_{i_{\text{s}}}) \geq \lambda_\tau)(a_{i_{\text{s}}} - \eta_\tau)^2}{\sum_{i_{\text{s}}=1}^{N_{\text{CE}}} I(s(a_{i_{\text{s}}}) \geq \lambda_\tau)}} \tag{6.6}$$

The most important parameters in the CE method for optimization are the number of samples N_{CE} and the quantile of best samples used to update the density, ϱ_{CE}. The number of samples should be at least a multiple of the number of parameters N_v, so $N_{\text{CE}} = c_{\text{CE}} N_v$ with

[2] If the score values of the samples are ordered increasingly and indexed such that $s_1 \leq \cdots \leq s_{N_{\text{CE}}}$, then the $(1 - \varrho_{\text{CE}})$ quantile is: $\lambda_\tau = s_{\lceil (1-\varrho_{\text{CE}}) N_{\text{CE}} \rceil}$ where $\lceil \cdot \rceil$ rounds the argument to the next greater or equal integer number (ceiling).

$c_{\mathrm{CE}} \in \mathbb{N}$, $c_{\mathrm{CE}} \geq 2$. The parameter ϱ_{CE} can be taken around 0.01 when many samples are used, or larger, around $\ln(N_{\mathrm{CE}})/N_{\mathrm{CE}}$, if there are only a few samples ($N_{\mathrm{CE}} < 100$) (Rubinstein and Kroese, 2004).

CE optimization has found many applications in recent years, e.g., in biomedicine (Mathenya et al., 2007), vector quantization (Boubezoul et al., 2008), vehicle routing (Chepuri and de Mello, 2005), and clustering (Rubinstein and Kroese, 2004). While its convergence is not guaranteed in general, and the theoretical understanding of the CE method is currently limited, the algorithm is usually convergent in practice (Rubinstein and Kroese, 2004). Convergence proofs are available only for combinatorial optimization (Rubinstein and Kroese, 2004; Margolin, 2005; Costa et al., 2007). The most general proof available to date shows that the CE method for combinatorial optimization asymptotically converges, with probability 1, to a unit mass density, i.e., a density that always generates samples equal to a single point. Furthermore, this convergence point can be made equal to the optimal solution by using an adaptive smoothing technique (Costa et al., 2007).

6.4 CE policy search

In this section, the proposed algorithm for policy optimization using the cross-entropy method is described. The algorithm works for discrete-action, stochastic or deterministic MDPs. First, the general parameterization of the policy is given and the CE score function is defined. Then, an instantiation of the general algorithm is given that uses RBFs to parameterize policies.

6.4.1 Policy parameterization and CE score function

Consider a stochastic MDP with the state space X, the action space U_{d}, the transition probability function $\tilde{f} : X \times U_{\mathrm{d}} \times X \to [0, \infty)$, and the reward function $\tilde{\rho} : X \times U_{\mathrm{d}} \times X \to \mathbb{R}$. Denote by D the number of state variables (the dimension of X). In the sequel, it is assumed that the action space is discrete and contains M distinct actions, $U_{\mathrm{d}} = \{u_1, \ldots, u_M\}$. The discrete set U_{d} can result from the discretization of an originally continuous action space.

The policy parameterization is defined in the following way. A set of $\mathcal{N}$ BFs over the state space is defined. The BFs are parameterized by a vector $\xi \in \Xi$ that typically gives their locations and shapes. Denote these BFs by $\varphi_i(x; \xi) : X \to \mathbb{R}$, $i = 1, \ldots, \mathcal{N}$, to highlight their dependence on ξ. The BFs are associated to the discrete actions by a many-to-one mapping. This mapping can be represented as a vector $\vartheta \in \{1, \ldots, M\}^{\mathcal{N}}$ that associates each BF φ_i to a discrete action index $\vartheta(i)$, or equivalently to a discrete action $u_{\vartheta(i)}$. The expression $\vartheta(i)$ is used to denote the i-th element of ϑ. A schematic representation of this parameterization is given in Figure 6.1.[3]

Once ξ and ϑ are known, there are still several ways in which they can be used to form the policy. Two different policy parameterizations are considered in the sequel, which will be called the nearest-BF and the sum-of-BFs policies, respectively. The first type of policy chooses for any point x the action associated to the BF that takes the largest value at x:

$$h(x) = u_{\vartheta(i^*)}, \quad i^* = \arg\max_i \varphi_i(x; \xi) \tag{6.7}$$

[3]Recall that calligraphic symbols are used to denote quantities related to policy approximation.

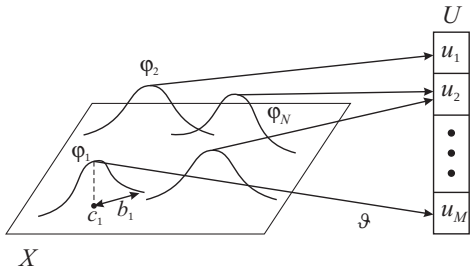


Figure 6.1: A schematic representation of the policy parameterization. The vector ϑ associates the BFs to the discrete actions. In this example, the BFs are parameterized by their centers c_i and widths b_i, so that $\xi = [c_1^{\mathrm{T}}, b_1^{\mathrm{T}}, \ldots, c_{\mathcal{N}}^{\mathrm{T}}, b_{\mathcal{N}}^{\mathrm{T}}]^{\mathrm{T}}$.

Because many types of BFs used in practice take their maximum value at a center point and decrease with the distance from this point, the policy (6.7) is called in the sequel a *nearest-BF* policy. For the second type of policy, the values of the BFs associated to each discrete action are summed up, and the action with the largest sum is chosen:

$$h(x) = u_{j^*}, \quad j^* = \arg\max_j \sum_{i \in \{ i' \mid 1 \leq i' \leq \mathcal{N},\ \vartheta(i') = j \}} \varphi_i(x; \xi) \tag{6.8}$$

This is called a *sum-of-BFs* policy in the sequel. The performance of these two parameterizations is compared in Section 6.5.1.

Any of these two policies is uniquely determined by the value of the parameter vector $a = [\xi^{\mathrm{T}}, \vartheta^{\mathrm{T}}]^{\mathrm{T}}$, ranging in the set $\mathcal{A} = \Xi \times \{1, \ldots, M\}^{\mathcal{N}}$. CE optimization will be used to search for an optimal parameter vector $a^* = [\xi^{*\mathrm{T}}, \vartheta^{*\mathrm{T}}]^{\mathrm{T}}$ that maximizes the following score function:

$$s(\xi, \vartheta) = \sum_{x_0 \in X_0} w(x_0) \widehat{R}^h(x_0) \tag{6.9}$$

where $\widehat{R}^h$ is the estimated return of the policy h determined by ξ and ϑ, and X_0 is a given finite set of representative states, weighted by $w : X_0 \to (0, 1]$.[4] The set X_0, together with the weight function w, will determine the performance of the resulting policy. Some problems only require to optimally control the system from a restricted set of initial states; X_0 should then be equal to this set, or included in it when the set is too large. Also, initial states that are deemed more important can be assigned larger weights. When all initial states are equally important, the elements of X_0 should be uniformly spread over the state space, randomly or equidistantly, and identical weights equal to $\frac{1}{|X_0|}$ should be assigned to every element of X_0 ($|\cdot|$ denotes set cardinality).

The return for each state in X_0 is estimated by Monte Carlo simulations:

$$\widehat{R}^h(x_0) = \frac{1}{N_{\mathrm{MC}}} \sum_{i_0=1}^{N_{\mathrm{MC}}} \sum_{k=0}^{K} \gamma^k \tilde{\rho}(x_{i_0,k}, h(x_{i_0,k}), x_{i_0,k+1}) \tag{6.10}$$

[4] More generally, a density w over the initial states can be considered, and the score function is then $\mathrm{E}_{x_0 \sim w(\cdot)} \{ R^h(x_0) \}$, i.e., the expected return when $x_0 \sim w(\cdot)$. Such a score function can be evaluated by Monte Carlo methods. In this chapter, we only use finite sets X_0 associated with weighting functions w as in (6.9).

where $x_{i_0,0} = x_0$, $x_{i_0,k+1} \sim \tilde{f}(x_{i_0,k}, h(x_{i_0,k}), \cdot)$, and N_{MC} is the number of Monte Carlo simulations to carry out. So, each Monte Carlo simulation i_0 makes use of a system trajectory that is K steps long and generated using the policy h. The system trajectories are generated independently, so the score computation is unbiased. To guarantee that an error of at most $\varepsilon_{\mathrm{MC}}$ is introduced by truncating the discounted sum of rewards after K steps, the value of K can be computed with Equation (2.27), copied here for easy reference:

$$K = \left\lceil \log_\gamma \varepsilon_{\mathrm{MC}}(1-\gamma) / \left\|\tilde{\rho}\right\|_\infty \right\rceil \tag{6.11}$$

where $\left\|\tilde{\rho}\right\|_\infty = \max_{x,u,x'} |\tilde{\rho}(x,u,x')|$ is assumed bounded (see e.g., Mannor et al., 2003).

In order to define the associated stochastic problem (6.2) for the optimization problem of maximizing (6.9), it is necessary to choose a family of densities with support $\Xi \times \{1,\ldots,M\}^{\mathcal{N}}$. In general, Ξ may not be discrete set, so it is convenient to use separate densities for the two parts ξ and ϑ of the parameter vector. Denote the density for ξ by $p_\xi(\cdot; v_\xi)$, parameterized by v_ξ and with support Ξ, and the density for ϑ by $p_\vartheta(\cdot; v_\vartheta)$, parameterized by v_ϑ and with support $\{1,\ldots,M\}^{\mathcal{N}}$. Let N_{v_ξ} be the number of elements in the vector v_ξ, and N_{v_ϑ} the number of elements in v_ϑ.

The CE algorithm for direct policy search is given as Algorithm 6.1. For easy reference, Table 6.1 collects the meaning of the parameters and variables of CE policy search. The stochastic counterparts in lines 8 and 9 of Algorithm 6.1 have been simplified, using the fact that the samples are already sorted in the ascending order of their scores. When $\varepsilon_{\mathrm{CE}} = 0$, the algorithm terminates when λ remains constant for d_{CE} consecutive iterations. When $\varepsilon_{\mathrm{CE}} > 0$, the algorithm terminates when λ improves for d_{CE} consecutive iterations, and these improvements do not exceed $\varepsilon_{\mathrm{CE}}$. The integer $d_{\mathrm{CE}} > 1$ accounts for the random nature of the algorithm, by ensuring that the latest performance improvements did not decrease below $\varepsilon_{\mathrm{CE}}$ accidentally, but the decrease remains steady for d_{CE} iterations. Because the algorithm is not guaranteed to converge in general, a maximum number of iterations $\tau_{\max}$ has to be chosen to ensure the algorithm terminates in a finite time.

Algorithm 6.1 Cross-entropy policy search

Input:

dynamics $\tilde{f}$, reward function $\tilde{\rho}$, discount factor γ, representative states X_0, weights w
density families $\{p_\xi(\cdot; v_\xi)\}, \{p_\vartheta(\cdot; v_\vartheta)\}$, initial density parameters $v_{\xi,0}$, $v_{\vartheta,0}$
parameters $\mathcal{N}$, ϱ_{CE}, c_{CE}, d_{CE}, $\varepsilon_{\mathrm{CE}}$, $\varepsilon_{\mathrm{MC}}$, N_{MC}, $\tau_{\max}$

1: set number of samples $N_{\mathrm{CE}} \leftarrow c_{\mathrm{CE}}(N_{v_\xi} + N_{v_\vartheta})$
2: $\tau \leftarrow 1$
3: **repeat**
4: generate samples $\xi_1,\ldots,\xi_{N_{\mathrm{CE}}}$ from $p_\xi(\cdot; v_{\xi,\tau-1})$; $\vartheta_1,\ldots,\vartheta_{N_{\mathrm{CE}}}$ from $p_\vartheta(\cdot; v_{\vartheta,\tau-1})$
5: compute $s(\xi_{i_{\mathrm{s}}}, \vartheta_{i_{\mathrm{s}}})$ by (6.9), $i_{\mathrm{s}} = 1,\ldots,N_{\mathrm{CE}}$
6: reorder and reindex s.t. $s_1 \le \cdots \le s_{N_{\mathrm{CE}}}$
7: $\lambda_\tau \leftarrow s_{\lceil (1-\varrho_{\mathrm{CE}})N_{\mathrm{CE}} \rceil}$
8: $v_{\xi,\tau} \leftarrow \arg\max_{v_\xi} \sum_{i_{\mathrm{s}}=\lceil (1-\varrho_{\mathrm{CE}})N_{\mathrm{CE}} \rceil}^{N_{\mathrm{CE}}} \ln p_\xi(\xi_{i_{\mathrm{s}}}; v_\xi)$
9: $v_{\vartheta,\tau} \leftarrow \arg\max_{v_\vartheta} \sum_{i_{\mathrm{s}}=\lceil (1-\varrho_{\mathrm{CE}})N_{\mathrm{CE}} \rceil}^{N_{\mathrm{CE}}} \ln p_\vartheta(\vartheta_{i_{\mathrm{s}}}; v_\vartheta)$
10: $\tau \leftarrow \tau + 1$
11: **until** ($\tau > d_{\mathrm{CE}}$ **and** $0 \le \lambda_{\tau-\tau'} - \lambda_{\tau-\tau'-1} \le \varepsilon_{\mathrm{CE}}$, for $\tau' = 1,\ldots,d_{\mathrm{CE}}-1$) **or** $\tau > \tau_{\max}$

Output: $\widehat{\xi}^*, \widehat{\vartheta}^*$, the best sample; and $\widehat{s}^* = s(\widehat{\xi}^*, \widehat{\vartheta}^*)$

Table 6.1: Parameters and variables for CE policy search.

Symbol	Meaning
$\mathcal{N}$; M	number of BFs; number of discrete actions
ξ; ϑ	BF parameters; assignment of discrete actions to BFs
v_ξ; v_ϑ	parameters of the probability density for ξ; and for ϑ
N_{CE}	number of samples used at every CE iteration
ϱ_{CE}	proportion of samples used in the CE updates
λ	$(1-\varrho_{\mathrm{CE}})$ quantile of the sample performance
c_{CE}	how many times more samples to use than the number of density parameters
d_{CE}	number of iterations for which λ should stay roughly constant
$\varepsilon_{\mathrm{CE}}$; $\varepsilon_{\mathrm{MC}}$	convergence threshold; admissible error in computing returns
N_{MC}	number of Monte Carlo simulations for each representative state
τ; τ_{max}	iteration index; maximum number of iterations

Many times it is convenient to use distributions with unbounded support (e.g., Gaussians) when the BF parameters are continuous. However, the set Ξ must typically be bounded, e.g., when ξ contains centers of radial BFs, which must remain inside a bounded state space. Whenever this situation arises, samples can be generated from the density with larger support, and those samples that do not belong to Ξ can be rejected. The procedure continues until N_{CE} valid samples are generated, and the rest of the algorithm remains unchanged. The situation is entirely similar for the discrete action assignments ϑ, when it is convenient to use a family of densities $p_\vartheta(\cdot; v_\vartheta)$ with a support larger than $\{1,\ldots,M\}^{\mathcal{N}}$. An equivalent algorithm that uses all the samples can always be given by extending the score function to give samples falling outside the domain very large negative scores (larger in magnitude than for any valid sample). Additionally, for this equivalent algorithm, N_{CE} and ϱ_{CE} have to be adapted at each iteration such that a constant number of valid samples is generated, and that a constant number of best samples is used for the parameter updates. The theoretical basis of the CE optimization procedure is therefore not affected by sample rejection.

Section 6.3 has already discussed typical settings for the parameters c_{CE} (yielding N_{CE}) and ϱ_{CE}. Here, other parameters specific to policy search in MDPs are discussed. The parameter $\varepsilon_{\mathrm{MC}} > 0$ can be chosen a few orders of magnitude smaller than the typical return obtained by a policy from the states in X_0. Since it does not make sense to impose a convergence threshold smaller than the precision of the score function, $\varepsilon_{\mathrm{CE}}$ should be chosen larger than or equal to $\varepsilon_{\mathrm{MC}}$. A good value is $\varepsilon_{\mathrm{CE}} = \varepsilon_{\mathrm{MC}}$. The number $\mathcal{N}$ of BFs determines the accuracy of the policy approximator, and a good value for $\mathcal{N}$ will depend on the problem. In Sections 6.5.1 and 6.5.2, we study the effect of varying $\mathcal{N}$ in two example problems. For deterministic MDPs, a single trajectory is simulated for every initial state in X_0, so $N_{\mathrm{MC}} = 1$. For stochastic MDPs, several trajectories should be simulated, $N_{\mathrm{MC}} > 1$, with a good value of N_{MC} depending on the MDP considered.

6.4.2 CE policy search with radial basis functions

In this section, Markov decision processes with bounded, Euclidean state spaces are considered. Furthermore, it is assumed that the state space is a hyperbox centered in the origin, such that $|x| \leq x_{\max}\ \forall x \in X$ where $x_{\max} \in (0,\infty)^D$, and where the absolute value and relational

operators are applied element-wise. These assumption can easily be relaxed, but are made here for simplicity of the exposition.[5] Throughout the remainder of this chapter, Gaussian RBF are used to represent the policy:

$$\varphi_i(x;\xi) = \exp\left[-\sum_{d=1}^{D}\frac{(x_d - c_{i,d})^2}{b_{i,d}^2}\right] \tag{6.12}$$

where D is the number of state variables, $c_i = [c_{i,1}, \ldots, c_{i,D}]^{\mathrm{T}}$ is the D-dimensional center, and $b_i = [b_{i,1}, \ldots, b_{i,D}]^{\mathrm{T}}$ the D-dimensional radius of the i-th RBF. Many other types of BFs could be used instead, including e.g., splines and polynomials. However, because Gaussian RBFs are a common choice for approximate DP/RL (see e.g., Tsitsiklis and Van Roy, 1996; Ormoneit and Sen, 2002), we choose to focus on RBFs for this study.

Denote the vector of centers by $c = [c_1^{\mathrm{T}}, \ldots, c_{\mathcal{N}}^{\mathrm{T}}]^{\mathrm{T}}$ and the vector of radii by $b = [b_1^{\mathrm{T}}, \ldots, b_{\mathcal{N}}^{\mathrm{T}}]^{\mathrm{T}}$. So, $c_{i,d}$ and $b_{i,d}$ are scalars, c_i and b_i are D-dimensional vectors that collect the scalars for all D dimensions, and c and b are $D\mathcal{N}$-dimensional vectors that collect the D-dimensional vectors for all $\mathcal{N}$ RBFs. The centers of the RBFs have to lie within the bounded state space X, and the radii have to be strictly positive, i.e., $c \in X^{\mathcal{N}}$ and $b \in (0,\infty)^{D\mathcal{N}}$. The BFs parameter vector is therefore $\xi = [c^{\mathrm{T}}, b^{\mathrm{T}}]^{\mathrm{T}}$ and takes values in $\Xi = X^{\mathcal{N}} \times (0,\infty)^{D\mathcal{N}}$.

To define the associate stochastic problem for CE optimization, Gaussian densities are selected for the parameter vector ξ. The density for each center $c_{i,d}$ is parameterized by the mean $\eta_{i,d}^c$ and standard deviation $\sigma_{i,d}^c$; and the density for each radius $b_{i,d}$ is parameterized by the mean $\eta_{i,d}^b$ and the standard deviation $\sigma_{i,d}^b$. Similarly to the centers and radii themselves, denote the $D\mathcal{N}$-dimensional vectors of means and standard deviations as follows:

$$\begin{aligned}
\eta^c &= [\eta_{1,1}^c, \ldots, \eta_{1,D}^c, \ldots\ldots, \eta_{\mathcal{N},1}^c, \ldots, \eta_{\mathcal{N},D}^c]^{\mathrm{T}} \\
\sigma^c &= [\sigma_{1,1}^c, \ldots, \sigma_{1,D}^c, \ldots\ldots, \sigma_{\mathcal{N},1}^c, \ldots, \sigma_{\mathcal{N},D}^c]^{\mathrm{T}} \\
\eta^b &= [\eta_{1,1}^b, \ldots, \eta_{1,D}^b, \ldots\ldots, \eta_{\mathcal{N},1}^b, \ldots, \eta_{\mathcal{N},D}^b]^{\mathrm{T}} \\
\sigma^b &= [\sigma_{1,1}^b, \ldots, \sigma_{1,D}^b, \ldots\ldots, \sigma_{\mathcal{N},1}^b, \ldots, \sigma_{\mathcal{N},D}^b]^{\mathrm{T}}
\end{aligned}$$

The parameter for the density $p_\xi(\cdot; v_\xi)$ is then $v_\xi = [(\eta^c)^{\mathrm{T}}, (\sigma^c)^{\mathrm{T}}, (\eta^b)^{\mathrm{T}}, (\sigma^b)^{\mathrm{T}}]^{\mathrm{T}} \in \mathbb{R}^{4D\mathcal{N}}$. Since the support of this density is $\mathbb{R}^{2D\mathcal{N}}$ instead of the required $\Xi = X^{\mathcal{N}} \times (0,\infty)^{D\mathcal{N}}$, samples that do not belong to Ξ are rejected.

The mean and standard deviation for the RBF centers and radii are initialized for all i as follows:

$$\begin{aligned}
\eta_i^c &= 0_D & \sigma_i^c &= x_{\max} \\
\eta_i^b &= \frac{x_{\max}}{2(\sqrt[D]{\mathcal{N}} - 1)} & \sigma_i^b &= \frac{x_{\max}}{4(\sqrt[D]{\mathcal{N}} - 1)}
\end{aligned}$$

where 0_D is a vector of D zeros. The initial means and standard deviations for the RBF centers are chosen to ensure a good coverage of the state space. The initial means and standard deviations for the RBF radii are initialized heuristically to have a similar amount of overlap between RBFs as $\mathcal{N}$ and D vary.[6] The standard deviations also ensure that most of the radii

[5] They will be relaxed e.g., when CE policy search is applied to the HIV infection control problem of Section 6.5.3.

[6] If $\mathcal{N}$ were a power of D, and the RBF centers were distributed on a grid with $\sqrt[D]{\mathcal{N}}$ equidistant points on each axis, then along each axis the expected radii η_i^b would be a quarter of the distance between neighboring RBFs.

samples drawn at the first iteration are positive and therefore do not have to be rejected. At the end of each iteration τ of Algorithm 6.1, the means and standard deviations are updated element-wise according to (6.5) and (6.6).

For the assignment ϑ of discrete actions to BFs, the Bernoulli distribution is chosen. This distribution has a discrete binary support, $\{0, 1\}$, and a single parameter $\eta^{\text{bin}} \in [0, 1]$. It produces the value 1 with probability η^{bin} and the value 0 with probability $(1 - \eta^{\text{bin}})$. To use the Bernoulli distribution, each element in the indices vector ϑ is represented in binary code, using $\mathcal{N}^{\text{bin}} = \lceil \log_2 M \rceil$ bits. When sampling, the bits are drawn independently. Whenever M is not a power of 2, bit combinations corresponding to invalid indices are rejected while drawing samples. For instance, if $M = 3$, $\mathcal{N}^{\text{bin}} = 2$, the binary value 00 points to the first discrete action u_1 (since the binary representation is zero-based), 01 points to u_2, 10 to u_3, and 11 is invalid and will be rejected. Such a concatenation of Bernoulli distributions can converge to any binary value (i.e., to a degenerate distribution that associates all the probability mass with that binary value), which means it can find a precise optimum location. Other distributions with discrete support that have this property typically require more than $\lceil \log_2 M \rceil$ parameters.

Denote by ϑ^{bin} the binary representation of the action assignments ϑ. This representation has $\mathcal{N}^{\text{bin}}$ bits for each individual action assignment $\vartheta(i)$. Therefore, the entire binary representation ϑ^{bin} has $\mathcal{N}\mathcal{N}^{\text{bin}}$ bits. Because each bit is drawn from its own Bernoulli distribution, the density parameter vector v_ϑ for the action assignments also has $\mathcal{N}\mathcal{N}^{\text{bin}}$ elements: $v_\vartheta \in [0, 1]^{\mathcal{N}\mathcal{N}^{\text{bin}}}$. The initial value for all the parameters is 0.5, which means that the bits 0 and 1 are initially equiprobable for every position of ϑ^{bin}. Because the Bernoulli distribution belongs to the natural exponential family, the stochastic counterpart in line 9 of Algorithm 6.1 can be solved analytically:

$$v_{\vartheta,\tau} = \frac{\sum_{i_s=1}^{N_{\text{CE}}} I(s(\xi_{i_s}, \vartheta_{i_s}) \geq \lambda_\tau) \vartheta_{i_s}^{\text{bin}}}{\sum_{i_s=1}^{N_{\text{CE}}} I(s(\xi_{i_s}, \vartheta_{i_s}) \geq \lambda_\tau)}$$

where ϑ_{i_s} is the integer value corresponding to the binary sample $\vartheta_{i_s}^{\text{bin}}$.

Because the form of the BFs, the densities for the BF parameters and action assignments, and the initial density parameters in Algorithm 6.1 have all been fixed, the parameters left to be chosen are $\mathcal{N}$, ϱ_{CE}, c_{CE}, d_{CE}, ε_{CE}, ε_{MC}, N_{MC}, and τ_{max} (see Table 6.1). The number of density parameters is $N_{v_\xi} = 4D\mathcal{N}$ (mean and variance for each individual center coordinate and radius value) and $N_{v_\vartheta} = \mathcal{N}\mathcal{N}^{\text{bin}}$. Therefore, $N_{\text{CE}} = c_{\text{CE}}\mathcal{N}(4D + \mathcal{N}^{\text{bin}})$.

Next, the complexity of this instantiation of CE policy search is briefly investigated. The largest amount of computation is spent by the algorithm in the Monte Carlo simulations required to estimate the score. Neglecting therefore the other computations, the complexity of one iteration of the algorithm is at most:

$$t_{\text{step}}[c_{\text{CE}}\mathcal{N}(4D + \mathcal{N}^{\text{bin}}) \, |X_0| \, N_{\text{MC}} K] \tag{6.13}$$

where $N_{\text{CE}} = c_{\text{CE}}\mathcal{N}(4D + \mathcal{N}^{\text{bin}})$ is the number of samples evaluated at each iteration, K is the maximum length (6.11) of each trajectory, and t_{step} is the time needed to compute $h(x)$ for a fixed x and to simulate the controlled system for one time step.

6.5 Experimental studies

In this section, extensive numerical experiments are carried out to assess the performance of CE policy search. CE policy search is used to control a double integrator in Section 6.5.1, to balance a bicycle riding at constant speed in Section 6.5.2, and to control the treatment of infection with the HIV in Section 6.5.3.

6.5.1 Discrete-time double integrator

In this section, a simple control problem is used to evaluate the proposed CE policy search algorithm, and to compare it with the fuzzy Q-iteration algorithm of Chapter 4. The example involves the optimal control of a double integrator, and is chosen so that optimal and near-optimal trajectories from any initial state terminate in a small number of steps. This property allows for extensive simulation experiments to be run and for the direct computation of an optimal solution, without excessive computational costs.

Model and an optimal solution

The optimal control of a discrete-time, double integrator is considered. This problem is deterministic with a two-dimensional continuous state space $X = [-1, 1] \times [-0.5, 0.5]$, a binary action space $U_\mathrm{d} = \{-0.1, 0.1\}$, and the following dynamics:

$$x_{k+1} = f(x_k, u_k) = \mathrm{sat}\left\{[x_{1,k} + x_{2,k}, x_{2,k} + u_k]^\mathrm{T}, -x_\mathrm{max}, x_\mathrm{max}\right\} \tag{6.14}$$

where $x_\mathrm{max} = [1, 0.5]^\mathrm{T}$ and 'sat' denotes saturation, which is used to restrict the evolution of the state to X. The states for which $|x_1| = 1$ are terminal. Applying any action in a terminal state brings the process back to the same state, with a zero reward. The goal is to drive the position x_1 to either boundary of the interval $[-1, 1]$ (i.e., to a terminal state), in such a way that at the moment when x_1 reaches the boundary, the speed x_2 as small as possible in magnitude. This goal is expressed by the reward function:

$$r_{k+1} = \rho(x_k, u_k) = -(1 - |x_{1,k+1}|)^2 - x_{2,k+1}^2 x_{1,k+1}^2 \tag{6.15}$$

This reward function is depicted in Figure 6.2. The discount factor γ is set to 0.95.

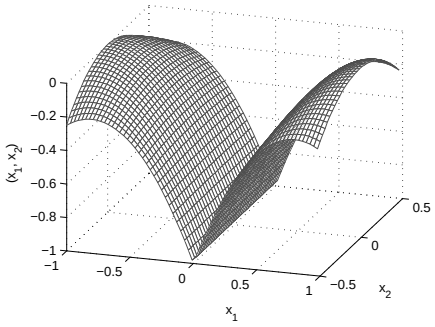
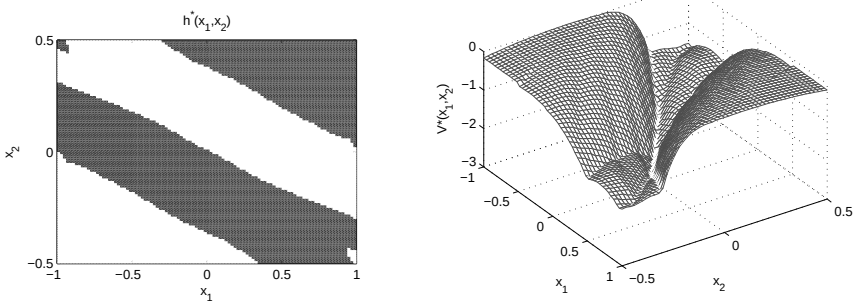


Figure 6.2: The reward function for the double-integrator example.

Figure 6.3(a) presents the optimal actions for a regular grid of 101×101 points covering the state space. This figure gives an accurate representation of the optimal policy. The optimal actions are obtained using the following brute-force procedure. All the possible sequences of actions of a sufficient length are generated, and the system is controlled with all these sequences starting from every state on the grid. For every state, the sequence that produces the best discounted return is the optimal sequence; the first action in this sequence is the optimal action; and the return accumulated by this sequence is the optimal value for that state. This procedure required 12841 s to run.[7] Figure 6.3(b) presents the optimal values for the states on the grid. Note that this brute-force procedure can only be applied because the considered MDP has terminal states, and all the optimal trajectories reach a terminal state after a relatively small number of steps. The procedure cannot be applied for tasks without terminal states, and quickly becomes impractical as the possible length of optimal trajectories increases.



(a) An optimal policy. The black color corresponds to the action -0.1, and white to $+0.1$.

(b) The optimal returns (optimal value function).

Figure 6.3: An optimal solution for the double-integrator example.

Results of CE policy search

To apply CE policy search, the following set of representative initial states was used to compute the score (6.9):

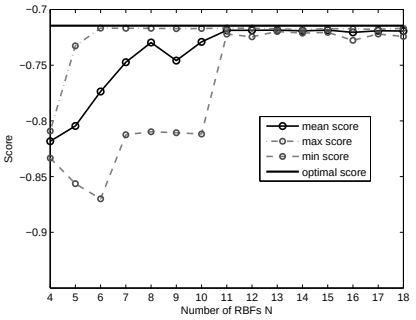
$$X_0 = \{-1, -0.9, \ldots, 1\} \times \{-0.5, -0.3, -0.1, 0, 0.1, 0.3, 0.5\}$$

The states were equally weighted using $w(x_0) = 1/|X_0|$ for any x_0. The parameter settings for the algorithm were $c_{\mathrm{CE}} = 5$, $\varrho_{\mathrm{CE}} = 0.05$, $\varepsilon_{\mathrm{CE}} = \varepsilon_{\mathrm{MC}} = 0.005$, $d_{\mathrm{CE}} = 5$, $\tau_{\max} = 50$. Little or no tuning was necessary to choose these values. Because the system is deterministic, $N_{\mathrm{MC}} = 1$. For all the experiments presented below, the maximum number of iterations $\tau_{\max}$ was never reached before convergence.

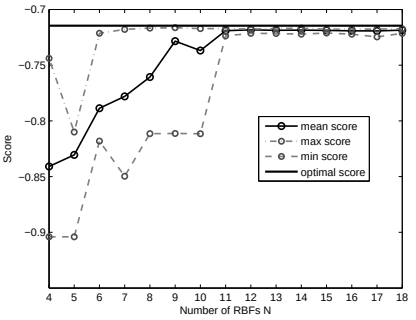
With these parameter settings, CE policy search was run while gradually increasing the number $\mathcal{N}$ of BFs from 4 to 18. Separate experiments were run for the nearest-RBF and the

[7] All the computation times reported in this chapter were recorded while running the algorithms in MATLAB 7.1 on a PC with an Intel Pentium IV 3 GHz CPU, 512 MiB RAM, and using Windows XP.

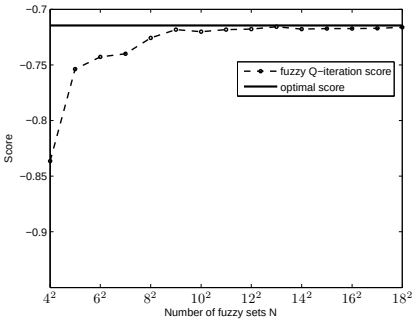
sum-of-RBFs policies. Ten experiments were run for every combination of $\mathcal{N}$ and type of policy. A synthesis of the obtained performance is given in Figures 6.4(a) and 6.4(b). For comparison, the exact optimal score for X_0 was computed using a brute-force search for optimal open-loop action sequences, as explained in Section 6.5.1.



(a) Nearest-RBF policy.



(b) Sum-of-RBFs policy.



(c) Fuzzy Q-iteration.

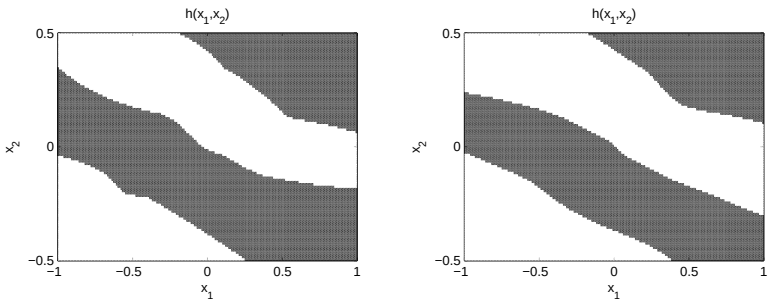
Figure 6.4: Performance of CE policy search and fuzzy Q-iteration on the double-integrator example. For CE policy search, the average performance is shown as a function of $\mathcal{N}$, together with the maximum and minimum performance achieved in any of the 10 experiments. Note the difference in the scale of the horizontal axes between (a)–(b) (linear), and (c) (quadratic).

CE policy search is compared with a representative technique relying on value functions, namely the fuzzy Q-iteration algorithm of Chapter 4. To apply fuzzy Q-iteration to the double integrator problem, equidistant triangular membership functions (MFs) were defined for both state variables, and the two-dimensional MFs were computed as the products of each combination of one-dimensional MFs, as in Example 4.1. The evolution of the fuzzy Q-iteration performance with the number of MFs is shown in Figure 6.4(c). The number of MFs *for each state variable* was gradually increased from 4 to 18. The total number of pyramidal, two-dimensional MFs is the *square* of this number. The convergence threshold for fuzzy Q-iteration was chosen equal to 10^{-7}, to ensure that the parameter vector θ converges to a near-optimal value. Note that for this example, fuzzy Q-iteration is not guaranteed to be consistent, i.e., to converge to the optimal Q-function as the approximation accuracy increases.

This is because the reward function (6.15) is discontinuous at the transitions into the terminal states. Recall that a terminal state is entered whenever the position x_1 reaches a boundary of the interval $[-1, 1]$. To see that the reward function is discontinuous, it is sufficient to note that all rewards obtained in the terminal state are 0 (by definition), but rewards obtained for states x_1 close to -1 or 1 are strictly negative when x_2 is non-zero.

For both the nearest-RBF and the sum-of-RBFs policies, CE policy search gives a large variance in the performance when $\mathcal{N} = 4, \ldots, 10$, although the average performance is increasing and a near-optimal performance is reached in some experiments starting from $\mathcal{N} = 7$. Starting from $\mathcal{N} = 10$, the algorithm consistently and reliably obtains a near-optimal performance. In contrast, fuzzy Q-iteration obtains a near-optimal performance starting from 9 MFs *on each axis*, which corresponds to a (much larger) number of 81 MFs. Overall, to obtain a similar performance to CE policy search with $\mathcal{N}$ BFs, fuzzy Q-iteration requires a number of MFs roughly equal to the square of this number. This difference is mainly due to the fact that the MFs used by fuzzy Q-iteration are equidistant and identically shaped, whereas the CE algorithm optimizes the shapes and locations of the BFs. There are no major differences between the performance of nearest-RBF and sum-of-RBFs policies.

Figure 6.5(a) presents an example of a nearest-RBF policy and Figure 6.5(b) of a sum-of-RBFs policy, both computed by CE policy search. Both policies resemble only roughly the optimal policy in the left part of Figure 6.3. Due to the RBF shapes, the edges where the actions change are more curved than in the optimal policy. Figures 6.6(a) and 6.6(b) show the convergence of the runs that produced the policies in Figures 6.5(a) and 6.5(b). The convergence is not monotonous; this is because the algorithm uses random samples in every iteration. Note that the variation of the $(1 - \varrho_{\mathrm{CE}})$ quantile of the sample performance is significant, although it appears to be small at the scale of the figure. As seen in Figure 6.4, differences in the performance as small as 0.1 are significant.

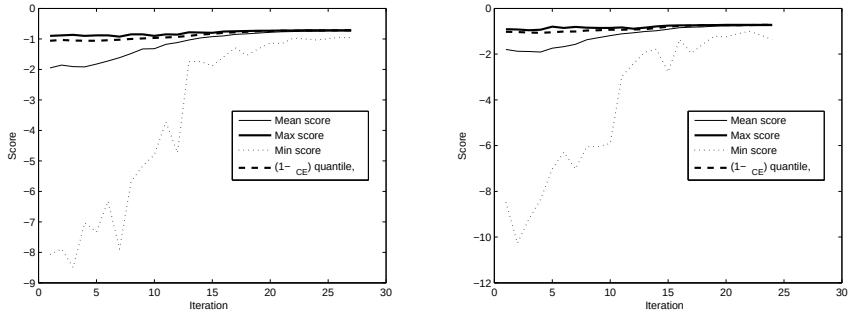


(a) A nearest-RBF policy, $\mathcal{N} = 15$. (b) A sum-of-RBFs policy, $\mathcal{N} = 15$.

Figure 6.5: Some representative policies computed with CE policy search.

We have also evaluated the usefulness for our algorithm of *smooth* updates of the Bernoulli density parameters v_ϑ. With smoothing, the new density parameter $v_{\vartheta,\tau}$ is computed as a convex combination between the solution of the stochastic counterpart in line 9 of Algorithm 6.1, and the old parameter $v_{\vartheta,\tau-1}$. The convex update is parameterized by the smoothing parameter $\alpha_{\mathrm{CE}} \in (0, 1]$:

$$v_{\vartheta,\tau} = (1 - \alpha_{\mathrm{CE}}) v_{\vartheta,\tau-1} + \alpha_{\mathrm{CE}} \arg\max_{v_\vartheta} \sum\nolimits_{i_\mathrm{s}=\lceil (1-\varrho_{\mathrm{CE}}) N_{\mathrm{CE}} \rceil}^{N_{\mathrm{CE}}} \ln p_\vartheta(\vartheta_{i_\mathrm{s}}; v_\vartheta)$$

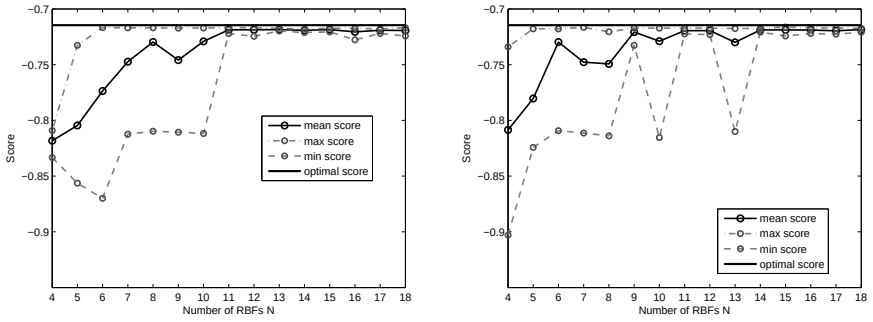


(a) Convergence for a nearest-RBF policy, $\mathcal{N} = 15$. (b) Convergence for a sum-of-RBFs policy, $\mathcal{N} = 15$.

Figure 6.6: Convergence of some representative runs of CE policy search.

The motivation for smoothed updates is that they prevent the problem described next. Without smoothing (i.e., $\alpha_{\mathrm{CE}} = 1$), the update might fix some parameters of the Bernoulli densities to the values 0 or 1. The components that have reached these values will remain constant in subsequent iterations, which may cause the algorithm to remain stuck in local optima. With smoothing, the parameters can converge to 0 or 1 only asymptotically (Rubinstein and Kroese, 2004).

We use smoothed updates with $\alpha_{\mathrm{CE}} = 0.7$ (a typical value) for the Bernoulli density parameter v_ϑ and run another series of experiment for the nearest-RBF policy, for the same values of $\mathcal{N} = 3, \ldots, 18$. Figure 6.7 compares the performance of the resulting policies with those computed without smoothing. It appears that smoothing does not provide any significant benefit for CE policy search, at least not in this example.



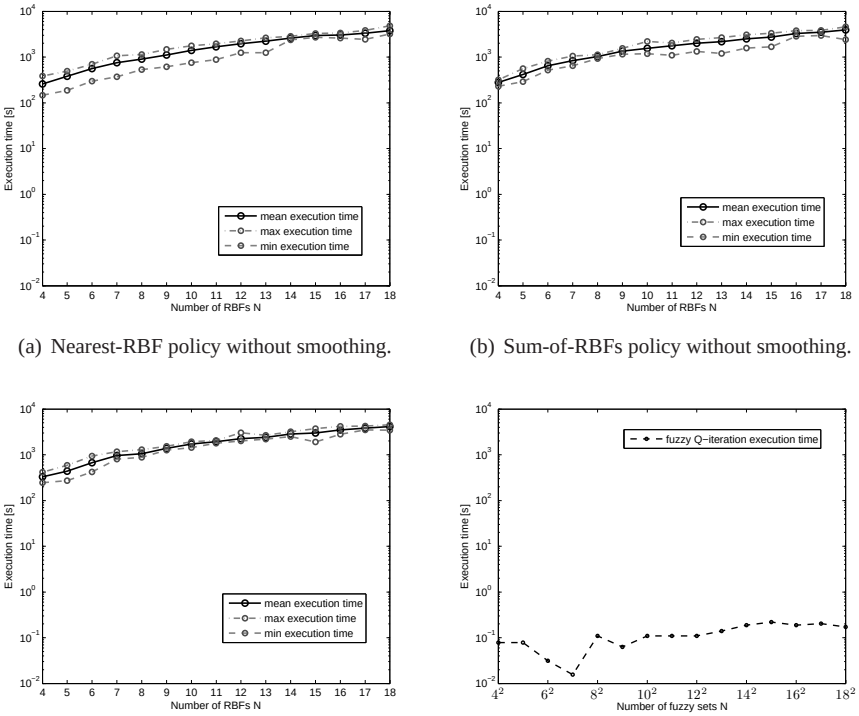
(a) Performance without smoothing.

(b) Performance with the smoothing parameter $\alpha_{\mathrm{CE}} = 0.7$ for the Bernoulli densities.

Figure 6.7: CE policy search with nearest-RBF policies: performance with and without smoothing.

Until now, the *performance* of CE policy search was studied and compared with the performance of fuzzy Q-iteration. Next, we investigate the *execution time* required by CE policy search to solve the double-integrator problem, and compare it with the execution time

of fuzzy Q-iteration. Figure 6.8 presents the execution time of CE policy search and fuzzy Q-iteration, as a function of $\mathcal{N}$, and Figure 6.9 presents the number of iterations required for convergence by the two algorithms. The number of CE iterations until convergence increases roughly linearly with $\mathcal{N}$, which combined with (6.13) means that the total execution time of the algorithm increases roughly quadratically with $\mathcal{N}$. There is no significant difference between the complexity with the nearest-RBF and sum-of-RBFs policies. Also, there is no significant difference between the complexity with and without smoothing. Although the number of fuzzy Q-iterations required for convergence is one order of magnitude larger than the number of CE iterations, the execution time of fuzzy Q-iteration is comparatively very small.[8] This is because an iteration of CE policy search is much more computationally expensive than a fuzzy Q-iteration. So, optimizing the parameters of the BFs leads to a better performance than using equidistant MFs, but at significantly higher computational costs.



(a) Nearest-RBF policy without smoothing.

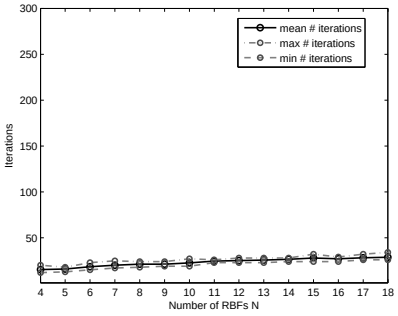
(b) Sum-of-RBFs policy without smoothing.

(c) Nearest-RBF policy with $\alpha_{\mathrm{CE}} = 0.7$.

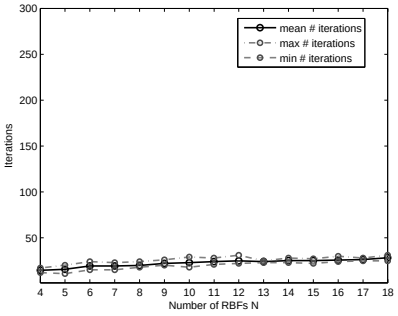
(d) Fuzzy Q-iteration.

Figure 6.8: Execution time of CE policy search and fuzzy Q-iteration. Note the difference in the scale of the horizontal axes between (a)–(c) (linear) and (d) (quadratic).

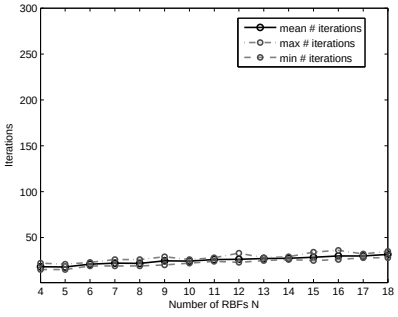
[8] Also note that the number of fuzzy Q-iterations to convergence often decreases as the number of MFs increases. This is probably because the better approximation accuracy provided by a larger number of MFs makes convergence easier.



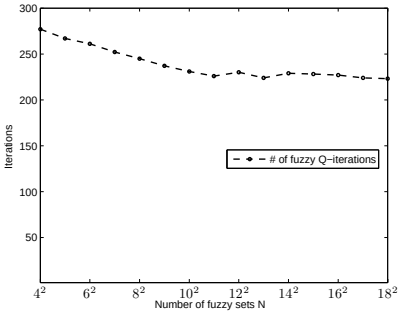
(a) Nearest-RBF policy without smoothing.



(b) Sum-of-RBFs policy without smoothing.



(c) Nearest-RBF policy with $\alpha_{\mathrm{CE}} = 0.7$.



(d) Fuzzy Q-iteration.

Figure 6.9: Number of iterations required for convergence of the CE policy search and fuzzy Q-iteration. Note the difference in the scale of the horizontal axes between (a)–(c) (linear) and (d) (quadratic).

6.5.2 Bicycle balancing

In this section, the CE policy search algorithm is applied to a more involved problem, namely the bicycle balancing problem (Randløv and Alstrøm, 1998; Lagoudakis and Parr, 2003a; Ernst et al., 2005). The task is to prevent from falling a bicycle that rides at constant speed on a horizontal surface. The steering column of the bicycle is vertical. Because of this, the bicycle is not self-stabilizing and has to be actively stabilized to prevent it from falling.

Bicycle model and reward function

A schematic representation of the bicycle, showing the state and control variables, is given in Figure 6.10. The state variables are the roll angle of the bicycle measured from the vertical axis, ω [rad], the roll rate $\dot{\omega}$ [rad/s], the angle of the handlebar α [rad], taken equal to 0 when the handlebar is in its neutral position, and the angular velocity $\dot{\alpha}$ [rad/s]. The control variables are the displacement δ [m] of the bicycle-rider common center of mass perpendicular to the plane of the bicycle, and the torque τ [Nm] applied to the handlebar. Two sets of experiments are considered. In the first set of experiments, the system is deterministic, while in the second set of experiments δ is affected by additive noise, denoted by w [m]. The noise enters

the model in (6.16), via the term β_k, and is drawn according to a uniform distribution in the interval $[-0.02, 0.02]$. The state vector is $x = [\omega, \dot{\omega}, \alpha, \dot{\alpha}]^{\mathrm{T}}$, and the control action vector is $u = [\delta, \tau]^{\mathrm{T}}$.

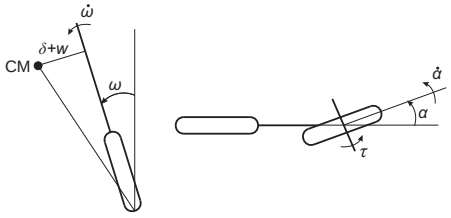


Figure 6.10: A schematic representation of the bicycle seen from behind (left) and from the top (right).

The dynamic model of the bicycle is (Ernst et al., 2005):[9]

$$\dot{\omega}_{k+1} = \dot{\omega}_k + T_{\mathrm{s}} \frac{1}{I_{\mathrm{bc}}} \Big[\sin(\beta_k)(M_{\mathrm{c}} + M_{\mathrm{r}})gh - \cos(\beta_k)\Big[\frac{I_{\mathrm{dc}} v}{r}\dot{\alpha}_k + \mathrm{sign}(\alpha_k) v^2 \Big(\frac{M_{\mathrm{d}} r}{l}(|\sin(\alpha_k)| + |\tan(\alpha_k)|) + \frac{(M_{\mathrm{c}} + M_{\mathrm{r}})h}{r_{\mathrm{CM}k}}\Big)\Big]\Big] \tag{6.16}$$

$$\omega_{k+1} = \omega_k + T_{\mathrm{s}}\dot{\omega}_k \tag{6.17}$$

$$\dot{\alpha}_{k+1} = \begin{cases} \dot{\alpha}_k + T_{\mathrm{s}} \frac{1}{I_{\mathrm{dl}}} \left(\tau - \frac{I_{\mathrm{dv}} v}{r}\dot{\omega}_k\right) & \text{if } |\alpha_k + T_{\mathrm{s}}\dot{\alpha}_k| > \frac{80\pi}{180} \\ 0 & \text{otherwise} \end{cases} \tag{6.18}$$

$$\alpha_{k+1} = \mathrm{sat}\left\{\alpha_k + T_{\mathrm{s}}\dot{\alpha}_k, \frac{-80\pi}{180}, \frac{80\pi}{180}\right\} \tag{6.19}$$

with

$$\beta_k = \omega_k + \arctan\frac{\delta_k + w_k}{h}, \qquad \frac{1}{r_{\mathrm{CM}k}} = \begin{cases} \frac{1}{\sqrt{(l-c)^2 + \frac{l^2}{\sin^2(\alpha_k)}}} & \text{if } \alpha_k \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

In (6.19), the steering angle α is restricted to the interval $[\frac{-80\pi}{180}, \frac{80\pi}{180}]$ due to physical constraints, and whenever this bound is reached the angular velocity $\dot{\alpha}$ is set to 0 by (6.18). The meaning and values of the physical parameters in the bicycle model are given in Table 6.2. The moments of inertia in the model can be computed as:

$$I_{\mathrm{bc}} = \frac{13}{3}M_{\mathrm{c}}h^2 + M_{\mathrm{r}}(h + d_{\mathrm{CM}})^2 \qquad I_{\mathrm{dc}} = M_{\mathrm{d}}r^2$$
$$I_{\mathrm{dv}} = \frac{3}{2}M_{\mathrm{d}}r^2 \qquad I_{\mathrm{dl}} = \frac{1}{2}M_{\mathrm{d}}r^2$$

Equations (6.16)–(6.19) are a discrete-time representation of the continuous-time bicycle dynamics, using Euler integration with a time step of T_{s}. The time step T_{s} is taken equal to 0.01 s in the sequel.

[9] In these equations, $\dot{\omega}_{k+1}$ is used to denote the value of the continuous-time state variable $\dot{\omega}$ at the sampling instant $k+1$, i.e., $\dot{\omega}((k+1)T_{\mathrm{s}})$. A similar notation is used for the other state variables.

Table 6.2: List of parameters in the bicycle model and their values.

Symbol	Value	Units	Meaning
g	9.81	$\mathrm{m/s^2}$	gravitational acceleration
v	$10/3.6$	$\mathrm{m/s}$	speed of the bicycle
h	0.94	m	height from the ground of the common center of mass (i.e., the center of mass of the bicycle and rider as a total)
l	1.11	m	distance between the front tire and the back tire at the point where they touch the ground
r	0.34	m	wheel radius
d_{CM}	0.3	m	vertical distance between the bicycle and rider centers of mass
c	0.66	m	horizontal distance between the point where the front wheel touches the ground and the common center of mass
M_{c}	15	kg	mass of the bicycle
M_{d}	1.7	kg	mass of a tire
M_{r}	60	kg	mass of the rider

When the roll angle is larger than $\frac{12\pi}{180}$ in either direction, the bicycle is supposed to have fallen, and a failure state is reached. This state is terminal and transitions into it are penalized with negative rewards. Therefore, ω is bounded to $[\frac{-12\pi}{180}, \frac{12\pi}{180}]$. Additionally, the roll rate and the steering angular velocity are restricted to the interval $[-2\pi, 2\pi]$, using saturation. The balancing task requires that the bicycle be prevented from falling, i.e., that the roll angle is kept within the allowed interval $[\frac{-12\pi}{180}, \frac{12\pi}{180}]$. For this task, it is not necessary to model the motion of the bicycle on the horizontal surface. The following reward function is chosen to express the balancing goal:

$$r_{k+1} = \begin{cases} 0 & \text{if } \omega_{k+1} \in [\frac{-12\pi}{180}, \frac{12\pi}{180}] \\ -1 & \text{otherwise} \end{cases} \tag{6.20}$$

The discount factor is $\gamma = 0.98$. To apply CE policy search, the rider displacement is discretized into three levels $\{-0.02, 0, 0.02\}$, and the torque on the handlebar into $\{-2, 0, 2\}$. This gives the discrete action space $U_{\mathrm{d}} = \{-0.02, 0, 0.02\} \times \{-2, 0, 2\}$, which is sufficient to balance the bicycle.

The model, the values of the parameters, the action discretization, and the reward function are all identical to those used by Ernst et al. (2005).

Balancing a deterministic bicycle

For the first set of experiments with the bicycle, the noise is eliminated from the model, so $w_k = 0$ for all k. The parameter settings for the CE algorithm are the same as for the double-integrator example of Section 6.5.1, i.e., $c_{\mathrm{CE}} = 5$, $\varrho_{\mathrm{CE}} = 0.05$, $\varepsilon_{\mathrm{CE}} = \varepsilon_{\mathrm{MC}} = 0.005$, $d_{\mathrm{CE}} = 5$, $\tau_{\text{max}} = 50$. Because the system is deterministic, a single trajectory needs to be simulated from every state in X_0, i.e., $N_{\mathrm{MC}} = 1$. The maximum number of 50 iterations was never reached before convergence. The sum-of-RBFs policy was used for all the bicycle experiments. This choice between the sum-of-RBFs and nearest-RBF policies was arbitrary, because both types of policy performed equally well in Section 6.5.1.

In order to study the influence of the representative set of states X_0 on the performance of the resulting policies, experiments were run for two different sets of representative states. In both cases, the initial states were uniformly weighted (i.e., $w(x_0) = 1/|X_0|$ for any x_0). Because we are mainly interested in the behavior of the bicycle starting from different initial rolls and roll velocities, the initial steering angle α and velocity $\dot{\alpha}$ are always taken equal to zero; considering also many values of α and $\dot{\alpha}$ would lead to excessive computational costs for the CE algorithm.

The first set of representative states contains a few evenly-spaced values for the initial roll of the bicycle, and the rest of the state variables are initially zero:

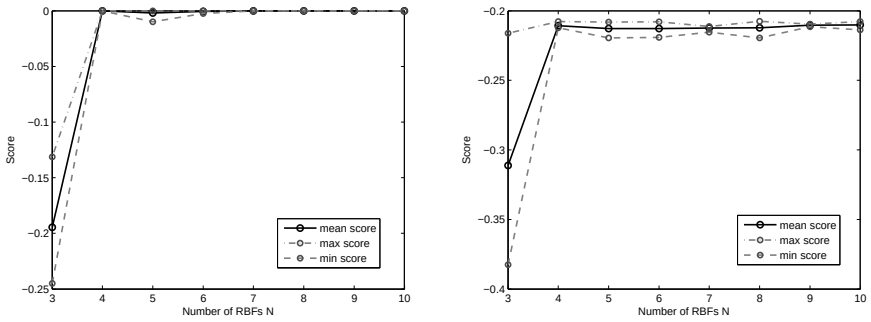
$$X_{0,1} = \left\{ [\omega, 0, 0, 0]^{\mathrm{T}} \;\middle|\; \omega = \frac{-10\pi}{180}, \frac{-5\pi}{180}, \ldots, \frac{10\pi}{180} \right\} \tag{6.21}$$

The second set is the cross-product of a finer roll grid and a few values of the roll velocity:

$$X_{0,2} = \left\{ [\omega, \dot{\omega}, 0, 0]^{\mathrm{T}} \;\middle|\; \omega = \frac{-10\pi}{180}, \frac{-8\pi}{180}, \ldots, \frac{10\pi}{180},\ \dot{\omega} = \frac{-30\pi}{180}, \frac{-15\pi}{180}, \ldots, \frac{30\pi}{180} \right\} \tag{6.22}$$

This set allows to study the effect of considering non-zero roll velocities in the representative states. The initial roll velocities are not taken large in magnitude to prevent including in $X_{0,2}$ too many states from which falling is unavoidable.

CE policy search was run for $\mathcal{N}$ ranging between 3 and 10. Ten independent experiments were run for each value of $\mathcal{N}$. The resulting performance is shown in Figure 6.11. For $X_{0,1}$, the optimal score is 0, because a good policy can always prevent the bicycle from falling for any initial roll value in $X_{0,1}$. This is no longer true for $X_{0,2}$: when ω and $\dot{\omega}$ are large in magnitude and have the same sign the bicycle cannot be prevented from falling by any control policy. For $X_{0,1}$ and $\mathcal{N} \geq 7$, all the runs reached precisely the optimal score of 0. For $X_{0,2}$, the optimal score is unknown. Nevertheless, for $\mathcal{N} \geq 4$ the algorithm obtains maximum scores around -0.21, without significant improvements as $\mathcal{N}$ continues to grow. This suggests the optimal score cannot be much higher than this value. Also for $\mathcal{N} \geq 4$, CE policy search reliably provides policies with scores that are close to this value.

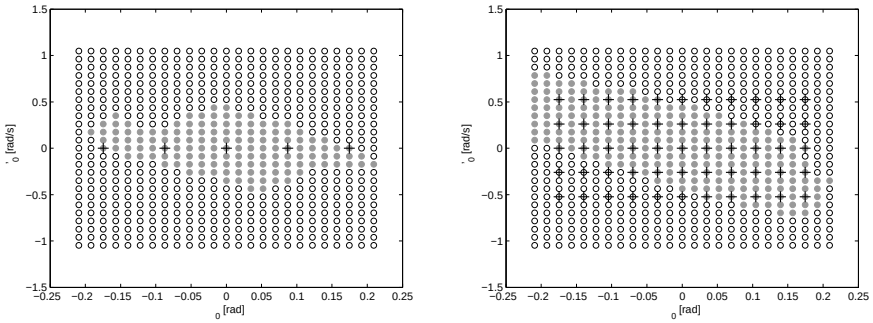


(a) Performance for $X_{0,1}$.

(b) Performance for $X_{0,2}$.

Figure 6.11: Performance of CE policy search for deterministic bicycle balancing. Note the difference in the scale of the vertical axes between (a) and (b).

Figure 6.12 illustrates the typical quality of the policies computed by the CE algorithm, by verifying how they perform when applied from initial states that do not belong to X_0 (i.e., how they generalize). For both policies, the number of RBFs is $\mathcal{N} = 8$. The initial steering angle and velocity are always 0. The figures contain binary markers that indicate whether the bicycle has been balanced successfully for 50 s (the horizon resulting from $\varepsilon_{\mathrm{MC}} = 0.005$ is $K = 456$ steps, corresponding to 4.56 s or roughly 10 times smaller). The benefit of the larger set $X_{0,2}$ of initial states is obvious, as the bicycle is balanced for a much larger portion of the $(\omega, \dot{\omega})$ plane. It also appears that some states in $X_{0,2}$ are failure states, from where the bicycle cannot be balanced. The failure states are characterized by values of ω and $\dot{\omega}$ that are large in magnitude and have the same sign. This explains the fact that all the score values obtained for $X_{0,2}$ are negative in Figure 6.11(b).

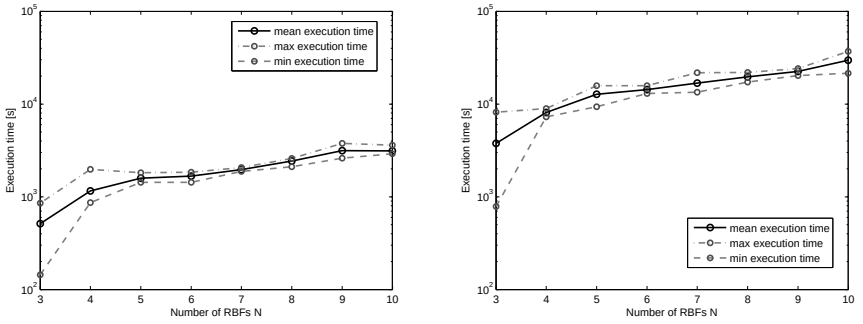


(a) Generalization for $X_{0,1}$.

(b) Generalization for $X_{0,2}$.

Figure 6.12: Generalization of two typical policies over initial states not in X_0, for the deterministic bicycle and $\mathcal{N} = 8$. White markers mean the bicycle was not balanced starting from that initial state; gray markers mean the bicycle was properly balanced. Black crosses mark the initial states in X_0.

Figure 6.13 shows the execution time of CE policy search for the deterministic bicycle. The execution times are comparable with those for the double integrator, seen in Figure 6.8,



(a) Execution time for $X_{0,1}$.

(b) Execution time for $X_{0,2}$.

Figure 6.13: Execution time of CE policy search for deterministic bicycle balancing.

even though the number of representative states is smaller for the bicycle. This is because simulating transitions for the bicycle requires the numerical integration of the nonlinear dynamics (6.16) – (6.19), which much is more costly than computing the linear transitions of the double integrator. Figure 6.14 illustrates the convergence of a typical run of CE policy search. The number of iterations necessary for convergence is similar to the double-integrator example (see Figure 6.6). Although the mean performance does not reach a near-optimal value in this number of iterations, the maximum performance and $(1 - \varrho_{\mathrm{CE}})$ quantile of the sample performance do converge to near-optimal values.

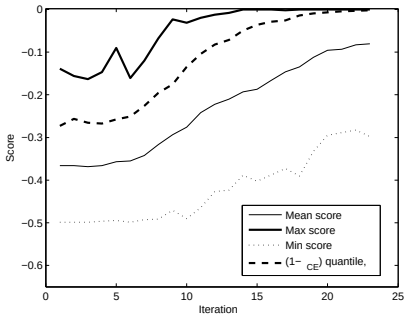


Figure 6.14: Typical convergence of CE policy search for deterministic bicycle balancing, for $X_{0,1}$ and $\mathcal{N} = 8$.

Balancing a stochastic bicycle

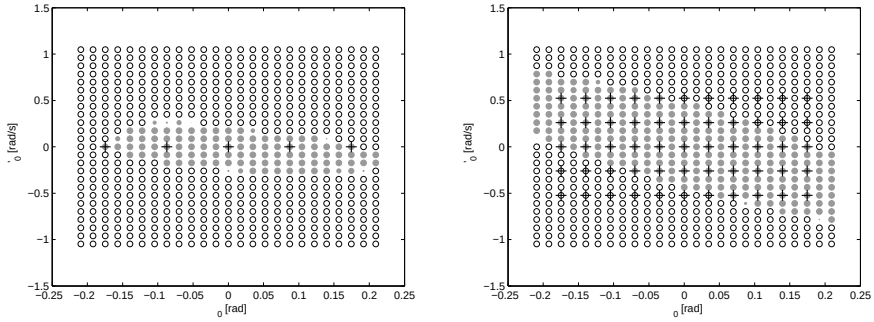
For the second set of experiments with the bicycle, the noise samples w_k were drawn independently according to a uniform distribution over the interval $[-0.02, 0.02]$, like in (Ernst et al., 2005). A number of $N_{\mathrm{MC}} = 10$ trajectories were simulated from every initial state to compute the Monte Carlo score (this number was not selected too large to keep the computational requirements of the algorithm manageable). The rest of the parameters were not changed, i.e., $c_{\mathrm{CE}} = 5$, $\varrho_{\mathrm{CE}} = 0.05$, $\varepsilon_{\mathrm{CE}} = \varepsilon_{\mathrm{MC}} = 0.005$, $d_{\mathrm{CE}} = 5$, $\tau_{\max} = 50$. Like for the deterministic bicycle experiments, the sum-of-RBFs policy was used, and the maximum number of 50 iterations was never reached before convergence.

Assuming that an optimal policy for the stochastic case is not significantly more complex than for the deterministic case, a number of $\mathcal{N} = 8$ RBFs was selected (the policies with $\mathcal{N} = 8$ gave reliable performance for the deterministic bicycle). A number of 10 independent experiments were run for each of the two sets of representative states $X_{0,1}$ and $X_{0,2}$ defined in (6.21) and (6.22). The results are reported in Table 6.3. It can be seen that all the scores for $X_{0,1}$ are optimal, and that the scores for $X_{0,2}$ are very close to the best scores obtained in the deterministic case. This shows that the policies computed have a good quality.

Figure 6.15 illustrates the typical quality of the computed policies, by verifying how they generalize to initial states that do not belong to X_0. The initial steering angle and velocity are always 0. A number of 10 controlled trajectories are simulated for every initial state. As before, the length of each trajectory is 50 s. Gray markers indicate that the bicycle has been balanced successfully for at least one of these trajectories, and the size of the markers is proportional to the number of times it was balanced successfully. White markers indicate the

Table 6.3: Performance of CE policy search for the stochastic bicycle, compared with the performance for the deterministic bicycle ($\mathcal{N} = 8$).

Score	Stochastic		Deterministic	
	$X_{0,1}$	$X_{0,2}$	$X_{0,1}$	$X_{0,2}$
maximum	0	-0.2096	0	-0.2075
mean	0	-0.2108	0	-0.2122
minimum	0	-0.2123	0	-0.2195



(a) Generalization for $X_{0,1}$.

(b) Generalization for $X_{0,2}$.

Figure 6.15: Generalization of two typical policies over initial states not in X_0, for the stochastic bicycle and $\mathcal{N} = 8$. White markers mean the bicycle was not balanced starting from that initial state; gray markers are proportional in size with the number of times the bicycle was properly balanced out of 10 experiments. Black crosses mark the initial states in X_0.

bicycle always fell when started from that initial state. The generalization is worse than in the deterministic case of Figure 6.12 for $X_{0,1}$, and better than in the deterministic case for $X_{0,2}$. It appears that noise can actually benefit the performance when a richer set of initial states is used. This may happen because the noise is driving the system over a larger area of the state space, and the CE algorithm is offered the opportunity to train the policy over this larger area. This phenomenon can be regarded as 'inherent exploration' caused by the stochastic nature of the system.

Table 6.4 presents the execution times of CE policy search for the stochastic bicycle. They are one order of magnitude larger than for the deterministic bicycle, which is expected because $N_{\mathrm{MC}} = 10$, rather than 1 as in the deterministic case.

Table 6.4: Execution time of CE policy search for the stochastic bicycle, compared with the execution time for the deterministic bicycle ($\mathcal{N} = 8$).

Execution time [s]	Stochastic		Deterministic	
	$X_{0,1}$	$X_{0,2}$	$X_{0,1}$	$X_{0,2}$
maximum	27519	218589	2597	21912
mean	22697	180856	2422	19625
minimum	16063	153201	2110	17255

Figure 6.16 shows a trajectory of the bicycle controlled with a policy resulting from CE policy search. The bicycle is successfully kept from falling, but is not stabilized to the vertical position ($\omega = 0$). This is because the reward function makes no difference between zero and non-zero values of the roll. The control actions chatter, which is necessary because only discrete actions are available to stabilize the unstable bicycle.

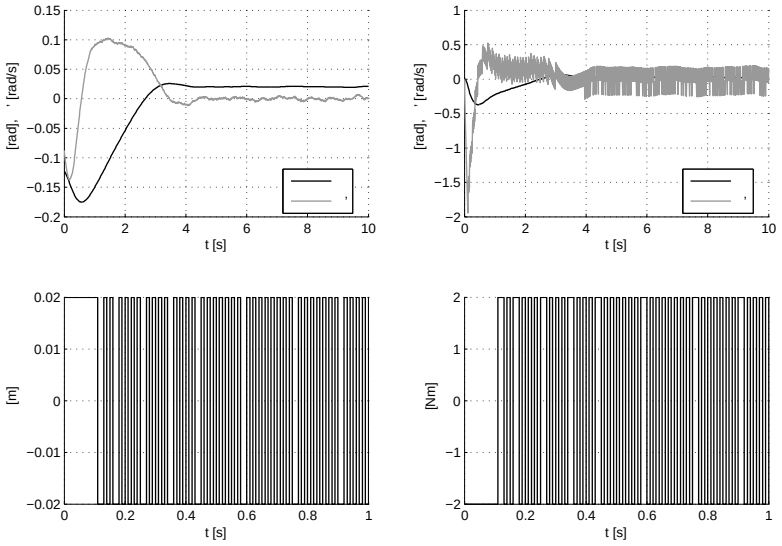


Figure 6.16: A controlled trajectory of the bicycle, starting from $\omega_0 = \frac{-7\pi}{180}$, $\dot{\omega}_0 = \frac{-5\pi}{180}$, $\alpha_0 = \dot{\alpha}_0 = 0$. The time axis of the controls is truncated at 1 s to maintain readability.

Other approaches to control the bicycle found in the literature use a more involved version of the problem, where the bicycle also has to ride to a target position in addition to being kept upright (Lagoudakis and Parr, 2003a; Ng and Jordan, 2000; Ernst et al., 2005). Therefore, our results cannot be directly compared with the results of those approaches. Nevertheless, Lagoudakis and Parr (2003a) use e.g., a total number of 100 BFs to compute Q-functions in the context of policy iteration, whereas in our experiments as few as 8 *automatically optimized* BFs suffice to obtain a reliable performance.[10] In the approach of Ernst et al. (2005), an ensemble of decision trees is automatically constructed to approximate the Q-function in the context of value iteration. The derivation automatically produces a large number of BFs, in the order of tens of thousands or more.

6.5.3 Structured treatment interruptions for HIV infection control

In this section, CE policy search is used to find a control law for the simulated treatment of infection with HIV. The prevalent HIV treatment strategy involves two types of drugs, called reverse transcriptase inhibitors (RTI) and protease inhibitors (PI). The negative side

[10] We also applied the algorithm of Lagoudakis and Parr (2003a) to bicycle balancing using equidistant BFs, and failed to obtain good results with much larger numbers of BFs (more than 5000). These results are not reported here.

effects of these drugs in the long term motivate the investigation of optimal strategies for their use. Such treatment strategies might also boost adaptive cellular immune response, which leads to better immune control of the disease (Wodarz and Nowak, 1999). One such strategy involves structured treatment interruptions (STI), where the patient is cycled on and off RTI and PI therapy (e.g., Adams et al., 2004). In some remarkable cases, STI strategies eventually allowed the patients to control the infection in the absence of treatment (Lisziewicz et al., 1999).

HIV infection dynamics and the STI problem

The following model of the HIV infection dynamics will be used (Adams et al., 2004):

$$
\begin{aligned}
\dot{T}_1 &= \lambda_1 - d_1 T_1 - (1-\epsilon_1) k_1 V T_1 \\
\dot{T}_2 &= \lambda_2 - d_2 T_2 - (1-f\epsilon_1) k_2 V T_2 \\
\dot{T}_1^{\mathrm{t}} &= (1-\epsilon_1) k_1 V T_1 - \delta T_1^{\mathrm{t}} - m_1 E T_1^{\mathrm{t}} \\
\dot{T}_2^{\mathrm{t}} &= (1-f\epsilon_1) k_2 V T_2 - \delta T_2^{\mathrm{t}} - m_2 E T_2^{\mathrm{t}} \\
\dot{V} &= (1-\epsilon_2) N_T \delta (T_1^{\mathrm{t}} + T_2^{\mathrm{t}}) - cV - [(1-\epsilon_1)\rho_1 k_1 T_1 + (1-f\epsilon_1)\rho_2 k_2 T_2] V \\
\dot{E} &= \lambda_E + \frac{b_E (T_1^{\mathrm{t}} + T_2^{\mathrm{t}})}{(T_1^{\mathrm{t}} + T_2^{\mathrm{t}}) + K_{\mathrm{b}}} E + \frac{d_E (T_1^{\mathrm{t}} + T_2^{\mathrm{t}})}{(T_1^{\mathrm{t}} + T_2^{\mathrm{t}}) + K_{\mathrm{d}}} E - \delta_E E
\end{aligned}
$$

The model describes two co-circulating populations of target cells (called type 1 and type 2 in the sequel). The values and meaning of the model parameters are given in Table 6.5. For a description of the model and the rationale behind the parameter values, we refer the reader to (Adams et al., 2004). The six-dimensional state vector is $x = [T_1, T_2, T_1^{\mathrm{t}}, T_2^{\mathrm{t}}, V, E]^{\mathrm{T}}$, and the state variables are:

- $T_1 \geq 0$ and $T_2 \geq 0$ — counts of healthy type 1 and type 2 target cells $\left[\frac{\text{cells}}{\text{ml}}\right]$,
- $T_1^{\mathrm{t}} \geq 0$ and $T_2^{\mathrm{t}} \geq 0$ — counts of infected type 1 and type 2 target cells $\left[\frac{\text{cells}}{\text{ml}}\right]$,
- $V \geq 0$ — number of free virus copies $\left[\frac{\text{copies}}{\text{ml}}\right]$,
- $E \geq 0$ — number of immune response cells $\left[\frac{\text{cells}}{\text{ml}}\right]$.

The positivity of the state variables is ensured during simulations by using saturation. The variable $\epsilon_1 \in [0, 0.7]$ is the effectiveness of the RTI drug, and $\epsilon_2 \in [0, 0.3]$ is the effectiveness of the PI drug. Like Adams et al. (2004), we do not model the relationship between the quantity of administered drugs and their effectiveness. We assume instead that ϵ_1 and ϵ_2 can be directly controlled, which leads to the two-dimensional control vector $u = [\epsilon_1, \epsilon_2]^{\mathrm{T}}$.

The system has three uncontrolled equilibria:

$$
\begin{aligned}
x_{\mathrm{n}} &= [1000000, 3198, 0, 0, 0, 10]^{\mathrm{T}} \\
x_{\mathrm{u}} &= [163573, 5, 11945, 46, 63919, 24]^{\mathrm{T}} \\
x_{\mathrm{h}} &= [967839, 621, 76, 6, 415, 353108]^{\mathrm{T}}
\end{aligned}
\tag{6.23}
$$

The x_{n} equilibrium is unstable and represents an uninfected patient (as soon as V becomes nonzero due to the introduction of virus copies, the patient becomes infected and the state

Table 6.5: List of parameters in the HIV model and their values.

Symbol	Value	Units	Meaning
$\lambda_1; \lambda_2$	$10,000; 31.98$	$\frac{\text{cells}}{\text{ml}\cdot\text{day}}$	production rates of target cell types 1 and 2
$d_1; d_2$	$0.01; 0.01$	$\frac{1}{\text{day}}$	death rates of target cell types 1 and 2
$k_1; k_2$	$8 \cdot 10^{-7}; 10^{-4}$	$\frac{\text{ml}}{\text{copies}\cdot\text{day}}$	infection rates of population 1 and 2
δ	0.7	$\frac{1}{\text{day}}$	infected cell death date
f	0.34	—	treatment effectiveness reduction in population 2
m_1, m_2	$10^{-5}; 10^{-5}$	$\frac{\text{ml}}{\text{cells}\cdot\text{day}}$	immune-induced clearance rates of populations 1 and 2
N_T	100	$\frac{\text{virions}}{\text{cell}}$	virions produced per infected cell
c	13	$\frac{1}{\text{day}}$	virus natural death rate
$\rho_1; \rho_2$	$1; 1$	$\frac{\text{virions}}{\text{cell}}$	mean number of virions infecting a type 1 (type 2) cell
λ_E	1	$\frac{\text{cells}}{\text{ml}\cdot\text{day}}$	immune effector production rate
b_E	0.3	$\frac{1}{\text{day}}$	maximum birth rate for immune effectors
K_b	100	$\frac{\text{cells}}{\text{ml}}$	saturation constant for immune effector birth
d_E	0.25	$\frac{1}{\text{day}}$	maximum death rate for immune effectors
K_d	500	$\frac{\text{cells}}{\text{ml}}$	saturation constant for immune effector death
δ_E	0.1	$\frac{1}{\text{day}}$	natural death rate for immune effectors

drifts away from this equilibrium). The unhealthy equilibrium x_u is stable and represents a patient where the infection has reached dangerous levels, and there is a very low immune response. The healthy equilibrium x_h is stable and represents a patient whose immune system controls the infection without the need of drugs.

In STI, drugs are either fully administered (they are 'on'), or not at all (they are 'off'). As explained before, the relationship between drug quantity and drug effectiveness is not modeled, so that the drugs have either full or 0 effectiveness. A fully effective RTI drug corresponds to $\epsilon_1 = 0.7$, while a fully effective PI drug corresponds to $\epsilon_2 = 0.3$. This leads to the discrete action space $U_\text{d} = \{0, 0.7\} \times \{0, 0.3\}$. It is not clinically feasible to allow changing the drug strategy with a daily frequency. It is therefore assumed that the state is measured and the drugs are switched on or off once every 5 days (like in Adams et al., 2004). This means the system can be controlled in discrete time with a sample time of 5 days. The dynamics are numerically integrated between the consecutive samples.

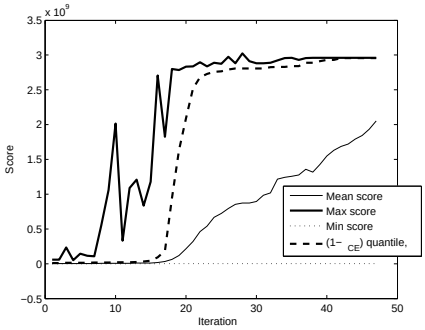
The control goal that we consider is to use STI from the initial state x_u such that the immune response of the patient is maximized, while also taking into account the count of virus copies, and the control effort. The reward function is (as in Adams et al., 2004):

$$\rho(x, u) = -QV - R_1\epsilon_1^2 - R_2\epsilon_2^2 + SE \tag{6.24}$$

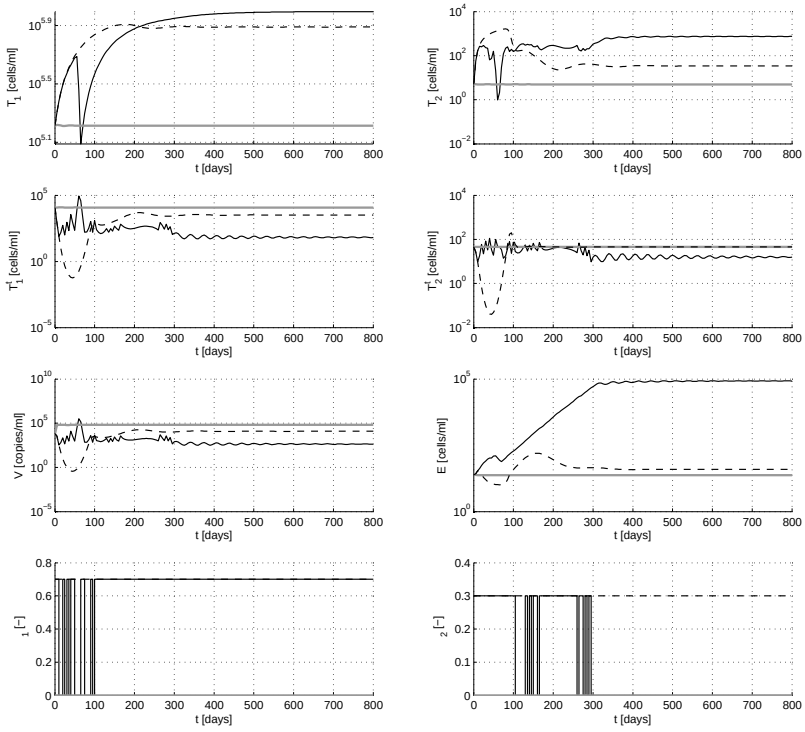
where $Q = 0.1, R_1 = R_2 = 20000, S = 1000$. The term $-QV$ penalizes the amount of virus copies, while the terms $-R_1\epsilon_1^2$ and $-R_2\epsilon_2^2$ penalize drug use. The term SE rewards the amount of immune response.

Results of CE policy search

In order to apply CE policy search, a discount factor of $\gamma = 0.99$ is used. To compute the Monte Carlo return, the number of simulation steps is set to $K = T_\text{f}/T_\text{s}$ where $T_\text{f} = 800$



(a) Convergence.



(b) System trajectories from $x_0 = x_{\mathrm{u}}$. Black, continuous: policy computed with CE policy search. Gray: no treatment. Black, dashed: fully effective treatment.

Figure 6.17: Results of CE policy search for the HIV problem.

days is a sufficiently long time horizon for a good policy to control the infection (Adams et al., 2004; Ernst et al., 2006). This leads to $K = 160$. The state variables vary in a domain spanning several orders of magnitude; to limit the effects of this large variation, a transformed state vector is used. This transformed vector is computed as the base 10 logarithm of the original state vector. A nearest-RBF policy with $\mathcal{N} = 8$ RBFs is used, and only the unhealthy initial state is used to compute the score, $X_0 = \{x_{\mathrm{u}}\}$. Two bits ($\mathcal{N}^{\mathrm{bin}} = 2$) are sufficient to index the 4 elements of the discrete action space. The parameters of the algorithm are set as follows: $c_{\mathrm{CE}} = 10$, $\varrho_{\mathrm{CE}} = 0.02$, $\varepsilon_{\mathrm{CE}} = 0$, $d_{\mathrm{CE}} = 5$, $\tau_{\mathrm{max}} = 50$. A larger number of samples than for the examples in Sections 6.5.1 and 6.5.2 is necessary for reliable convergence ($N_{\mathrm{CE}} = 2080$, corresponding to $c_{\mathrm{CE}} = 10$). This large number of samples also allows the value of ϱ_{CE} to be smaller.

Figure 6.17(a) shows how the algorithm converges. The execution time was approximately $404200\,\mathrm{s}$.[11] The trajectory of the controlled system when started from x_{u} is given in Figure 6.17(b), and compared with the evolution with no treatment, and with fully effective treatment. The computed solution switches the PI drug off after approximately 300 days, but the RTI drug is left on in steady state, which means that the healthy equilibrium x_{h} is not reached. This solution is worse than the best solutions available in the literature, which do reach the healthy equilibrium. Nevertheless, the immune response in steady state is very strong ($E \approx 10^5$), and the infection is handled much better than without STI. In fact, the steady-state value of x for this controlled trajectory, $x_{\mathrm{f}} \approx [989730, 744.8, 66.3, 15.8, 432.1, 84955]^{\mathrm{T}}$, is close to the healthy equilibrium x_{h} in (6.23).

In the literature, the healthy equilibrium is reached by driving the state into the basin of attraction of this equilibrium using STI, and then switching off the RTI and PI drugs. Adams et al. (2004) computed such a solution iteratively for 30-day subintervals of a 900-day time horizon. The sampling time had the same value of 5 days that was used in our experiment, so that the sequence of controls to optimize had 6 elements for each subinterval. Each optimal sequence was found using an exhaustive search of all the possible sequences. Ernst et al. (2006) computed an infinite-horizon solution with fitted Q-iteration, which also controlled the system to the healthy equilibrium. Our solution does not reach the healthy equilibrium, but still performs remarkably well given that the policy is parameterized using only $\mathcal{N} = 8$ BFs.

6.6 Conclusions and open issues

In this chapter, we have introduced and evaluated a policy search technique that can be applied to a large class of continuous-state, discrete-action MDPs. To this end, a highly flexible policy parameterization has been proposed, inspired by the work on the optimization of basis functions (BFs) for value function approximation. This parameterization represents policies using $\mathcal{N}$ state-dependent BFs, where a discrete action is assigned to each BF. The type of BFs and their number $\mathcal{N}$ are specified in advance and determine the approximation power of the representation. The locations and shapes of the BFs, together with the action assignments, are optimized using the cross-entropy (CE) method. The optimization criterion is a weighted sum of the returns from a set of representative initial states, where each return is computed with Monte Carlo simulations. The representative states and the weight function can be used

[11] This is the most computationally expensive simulation experiment reported in this thesis.

to focus the algorithm on important parts of the state space.

CE policy search has performed well both in a two-dimensional example with linear dynamics and in two more difficult, nonlinear control problems: bicycle balancing (with four state variables) and structured treatment interruptions for HIV infection control (with six state variables). The algorithm has reliably offered a good performance, using only a small number of BFs to represent the policy. For the two-dimensional example, the algorithm has required much fewer BFs than fuzzy Q-iteration to deliver the same performance. However, this performance has come at a large computational cost, several orders of magnitude higher than fuzzy Q-iteration. Solving the bicycle balancing and HIV infection control problems has also required a large amount of computations.

The theoretical study of CE policy search is a first opportunity for further research. Convergence results for the CE method are only available for combinatorial optimization (Rubinstein and Kroese, 2004; Margolin, 2005; Costa et al., 2007). The convergence results for the related model-reference adaptive search algorithm (Chang et al., 2007, Chapter 4) could be extended. These results apply to more general cases, including stochastic score functions evaluated by Monte Carlo integration (as is the case for CE policy search), but they unfortunately require the restrictive assumption that the optimal policy parameter is unique.

While in this chapter the CE method for optimization has been employed, there is in principle no obstacle to applying any optimization technique to determine a good policy parameter vector. In particular, other meta-heuristic optimization techniques like genetic algorithms, tabu search, pattern search, etc., could be used. Another interesting direction is extending CE policy search to work for continuous-action policy parameterizations. A natural extension of the current policy parameterization can be developed to handle continuous actions. This extension computes the policy by interpolating the actions assigned to the BFs, using the BF values as weights (3.25). CE policy search can also be extended to problems having hybrid state spaces and/or hybrid action spaces, where some of the variables are continuous and some are discrete.

Chapter 7

Conclusions and outlook

This thesis has proposed and studied techniques for dynamic programming (DP) and reinforcement learning (RL) in problems with continuous variables. This chapter summarizes the findings of our studies and some important open issues regarding the proposed techniques. The chapter closes with a discussion on more fundamental open issues in the field of DP and RL for control.

7.1 Summary and conclusions

This section summarizes our findings about the novel techniques for approximate DP/RL described in Chapters 4, 5, and 6.

Chapter 4 has introduced **fuzzy Q-iteration**, a model-based algorithm for approximate value iteration that represents Q-functions using a fuzzy partition of the state space and a discretization of the action space. The algorithm is theoretically sound: it asymptotically converges to a near-optimal solution, and it is consistent under continuity assumptions on the dynamics and on the reward function. A bound on the suboptimality of the solution obtained in a finite number of iterations is also derived. A version of the algorithm where parameters are updated in an asynchronous fashion converges at least as fast as the synchronous variant. As an alternative to designing in advance the membership functions (MFs) for the fuzzy sets in the partition, we have proposed to optimize the parameters of a constant number of MFs. To evaluate each configuration of MFs, a policy is computed with fuzzy Q-iteration and its performance is evaluated in simulation. Using the cross-entropy (CE) method for optimization, we have designed an algorithm to optimize the locations of triangular MFs.

An extensive numerical and experimental evaluation has indicated that fuzzy Q-iteration has a good performance in practice, among others producing better policies than Q-iteration with radial basis function approximation, and successfully swinging up and stabilizing a real-life inverted pendulum. In a simulation example involving servo-system stabilization, we have studied the influence of the number of fuzzy sets and of discrete actions on the performance. Increasing the number of fuzzy sets and discrete actions has resulted in a more predictable performance improvement with a continuous reward function, than with a discontinuous reward function. This indicates that discontinuous reward functions, although commonly used in RL, can harm the performance in continuous-variable control tasks.

For the car-on-the-hill problem, fuzzy Q-iteration has provided a better performance when using MFs that were optimized with the CE method, than when using equidistant MFs. However, this performance increase comes at a large computational cost, several orders of magnitude higher than the cost incurred by fuzzy Q-iteration with equidistant MFs.

Chapter 5 has introduced **online least-squares policy iteration**, an extension of the least-squares policy iteration (LSPI) algorithm to online learning. In the online case, the performance of every policy is important, so policy improvements have to be performed at short intervals. This means that only a small number of samples can be processed between consecutive policy improvements, and the algorithm cannot wait until every value function is accurately approximated. A variant of online LSPI has been described that uses prior knowledge about the monotonicity of the policy in the state variables.

Extensive numerical experiments have indicated that online LSPI is a promising algorithm for online RL. In these experiments, online LSPI has converged to a performance similar to that obtained by offline LSPI. Online LSPI has also exhibited a good robustness to variations in its tuning parameters, and has been able to learn in real time how to swing up and stabilize an inverted pendulum. In an example involving servo-system stabilization, for which there exist monotonous near-optimal policies, using prior knowledge about this monotonicity property has sped up online LSPI by a factor of two.

Additionally, a continuous-action, polynomial Q-function approximator for offline LSPI has been investigated. In the inverted pendulum problem, polynomial Q-function approximation has reduced chattering, even though it has not given a better performance (discounted returns) than discrete actions.

Chapter 6 has introduced **CE policy search**, a direct policy search technique that can be applied to a large class of problems. CE policy search employs a flexible policy parameterization, inspired by the work on the optimization of basis functions (BFs) for value function approximation. Policies are represented using state-dependent BFs and discretized actions, where a discrete action is assigned to each BF. The type of BFs and their number are specified in advance and determine the approximation power of the representation. The locations and shapes of the BFs, together with the action assignments, are optimized using the CE method. The optimization criterion is a weighted sum of the returns from a set of finitely many representative initial states, where each return is computed with Monte Carlo simulations. The representative states and the weight function can be used to focus the algorithm on important parts of the state space.

Numerical evaluations of cross-entropy policy search have been conducted on a double-integrator example with two state variables, as well as on two more involved problems: bicycle balancing (with four state variables) and the simulated treatment of infection with the human immunodeficiency virus (with six state variables). In these experiments, the algorithm has reliably obtained good policies, requiring only a small number of BFs. In the double-integrator example, the algorithm has required fewer BFs than fuzzy Q-iteration to deliver the same performance. However, this performance has incurred a large computational cost, e.g., several orders of magnitude higher than the cost of fuzzy Q-iteration.

7.2 Open issues and outlook

This section presents some important open issues regarding the techniques proposed in this thesis, together with more fundamental open issues in the field of DP and RL for control.

7.2.1 Open issues and future research

A number of open issues regarding the approaches proposed in this thesis are given below, together with some research directions that could be taken to address these issues.

Fuzzy Q-iteration has used triangular MFs in all the examples of Chapter 4. This type of MFs leads to a computational complexity that grows exponentially with the number of state variables. Beyond five or six variables, triangular MFs cannot be used effectively, and other types of MFs should be used to obtain a fuzzy partition with less than exponential complexity.

It is important to analyze fuzzy Q-iteration for stochastic problems, when the expected values have to be *approximated* (since the analysis presented in Chapter 4 holds when the expected values can be evaluated exactly). It is also interesting to investigate the effects of using discontinuous rewards in the stochastic case, and to verify whether these effects are different from those observed in the deterministic case. Another important step is developing model-free (RL) algorithms with fuzzy approximation that can guarantee convergence and consistency. Results from kernel-based or interpolation-based RL may be used in the theoretical study of such algorithms (Ormoneit and Sen, 2002; Szepesvári and Smart, 2004).

An extensive comparison of the performance (convergence, suboptimality, consistency) of the various types of linearly parameterized approximators that can be combined with Q-iteration (e.g., radial basis functions, Kuhn triangulations, etc.) would be very useful. Such a comparison is currently missing from the literature.

Instead of using optimization to find the MFs, resolution refinement techniques could be explored Section 3.5.1. Such techniques can help reduce the computational complexity of finding the MFs.

Online LSPI has used identically shaped, equidistant BFs (Chapter 5). Unfortunately, using such BFs in problems with a large number of state and action variables leads to an excessive computational cost. One possibility to address this problem is to simplify the computations required by the algorithm, e.g., by using gradient updates of the Q-function parameters instead of solving a linear system of equations (Geramifard et al., 2006, 2007). Another approach to reduce the computational cost is to automatically discover a small number of good BFs. Methods to do this have been developed for offline LSPI (Mahadevan and Maggioni, 2007; Xu et al., 2007), but they are geared towards the offline construction of BFs, and would need to be modified to work in the online case. Techniques that construct the BFs online, such as the non-parametric growing kernel approach of Jung and Polani (2007), are a promising alternative.

Because online LSPI only processes a few samples between consecutive policy improvements, the policy evaluation error can be very large. Since the asymptotic performance bound of offline, approximate policy iteration is affine in the policy evaluation error, this bound is not useful in the online case. Analyzing whether the asymptotic performance of online LSPI can be guaranteed is an important research topic.

Online LSPI uses the least-squares temporal difference algorithm for policy evaluation. Other policy evaluation techniques can be used for online learning, as well. One alternative is the least-squares policy evaluation algorithm. It was originally developed for V-functions (Bertsekas and Ioffe, 1996; Bertsekas, 2007), but we extend it to Q-functions in Appendix B. Least-squares temporal difference and least-squares policy evaluation algorithms can behave quite differently when they compute Q-functions online, just like when they compute V-functions online (Bertsekas, 2007). Therefore, it is important to determine which of these two algorithms performs better in online learning.

In Chapter 5, LSPI with polynomial Q-function approximation has been applied to the inverted pendulum problem. To better assess the potential of polynomial Q-function approximation, it is important to also evaluate it in other problems. Although in our experiments online LSPI has only been applied to time-invariant processes, it may also be able to deal with processes that are slowly changing over time, by adapting the policy to take these changes into account.

A first research opportunity for **CE policy search** (Chapter 6) is to study its convergence properties. Convergence results for the CE method are only available for combinatorial optimization (Rubinstein and Kroese, 2004; Margolin, 2005; Costa et al., 2007). The convergence results for the related model-reference adaptive search (Chang et al., 2007, Chapter 4) could be extended to CE policy search. These results apply to more general cases, including stochastic score functions evaluated by Monte Carlo integration (as is the case for CE policy search), but they unfortunately require the restrictive assumption that the optimal policy parameter is unique.

Another interesting direction is extending CE policy search to work for continuous-action policy parameterizations. A natural parameterization can be developed to handle continuous actions, which represents the policy using a linear combination of the BFs.

The CE method has been used to optimize the policy parameters for policy search in Chapter 6, and to optimize the MFs for fuzzy Q-iteration in Chapter 4. There is in principle no obstacle to applying any optimization technique to solve these two problems. However, in both cases, the optimization criterion may be a complicated function of the parameter vector, e.g., non-differentiable, non-convex, with many local optima. Therefore, global, gradient-free optimization techniques should be used, such as genetic algorithms, tabu search, pattern search, simulated annealing, etc. It is an interesting question how these optimization techniques compare with each other when used to optimize policies or MFs.

This thesis has focused on RL and DP for control, and the algorithms developed have been applied only to control problems (e.g., the inverted pendulum, the robotic manipulator, bicycle balancing). These problems have continuous state variables, and most often also continuous action variables. However, as mentioned in Chapter 1, DP and RL can also be applied to problems in other fields, including, e.g., artificial intelligence, operations research, and economics. All our results and algorithms can be directly applied to problems from these fields when the state variables are continuous and, in the case of fuzzy Q-iteration and continuous-action LSPI, also when the action variables are continuous. However, in certain problems some or all of the state and action variables are discrete. This is the case, e.g., in many problems from artificial intelligence. Some care is required to apply our algorithms and results to the discrete or hybrid[1] setting. For fuzzy Q-iteration, the state-action space can be discrete, but has to be a subset of a Euclidean space. This assumption can be eliminated, but eliminating it requires revising the consistency results of Section 4.4.2. For online LSPI, discrete or hybrid state-action spaces are not problematic. However, LSPI with polynomial action approximation (Section 5.3) naturally makes sense only for continuous action spaces; and the way in which the monotonicity requirements are enforced in Section 5.5 requires a continuous state-action space. In the form given in Chapter 6, CE policy search works for continuous states and discrete actions. It can, however, easily be extended to problems having discrete or hybrid state spaces, by using appropriately defined BFs and probability densities.

[1] A hybrid state-action space is defined by a mixture of continuous and discrete variables.

7.2.2 Outlook of DP and RL for control

Section 7.2.1 has outlined a number of open issues regarding the approaches presented in this thesis, together with some promising directions to address these issues. This section discusses a number of more fundamental open issues in the field of DP and RL for control.

- The most important open issue is the scalability of the algorithms to a large number of state and action variables. Most state of the art algorithms relying on value functions have been applied to problems with up to six continuous state variables (Lagoudakis and Parr, 2003a; Ernst et al., 2005; Mahadevan and Maggioni, 2007). Policy search algorithms scale up better when the policy parameterization is well suited to the problem at hand. However, such policy parameterizations cannot be used to solve general problems. The most promising approach to scalable DP and RL is the efficient, automatic discovery of a good approximator with low complexity. For value function approximation, linearly parameterized approximators are typically used. Discovering such an approximator boils down to discovering a good set of BFs. One challenge of automatic BF discovery is that, when the approximator is changed while estimating the value function, the convergence results for linear value function approximation no longer hold. Another important challenge, which affects both value function and policy approximation, is that finding an approximator is a difficult task which can incur a large computational cost (as seen e.g., in Section 4.6.4). It is crucial that the computational cost of the approximator discovery technique does not nullify the computational advantage of using the lower complexity approximator found.

- A major open issue in online RL is providing guaranteed performance during learning. This is essential for applying RL in practice. Ideally, online RL algorithms should guarantee a monotonous increase in their expected performance. Unfortunately, this is most likely impossible because all the online RL algorithms need to explore, i.e., try out actions that may be suboptimal, in order to make sure their performance does not remain stuck in a local optimum. Therefore, weaker requirements could be used, where an overall trend of increased performance is guaranteed, while still allowing for bounded and temporary decreases in performance due to exploration. From a control theoretical perspective, it is also very important that the DP/RL policy ensures the stability of the control loop. This is true for the current policy of online algorithms, as well as for the final policy resulting from offline algorithms. Most algorithms available today can guarantee neither a steady performance improvement nor stability.

- Classical texts on RL have long recommended that the reward function should be kept as simple as possible; it should only reward the achievement of the final goal (Sutton and Barto, 1998). There are several problems with this approach. The first problem is that a simple reward function often makes online learning very slow, and may lead to a large number of samples being required for offline learning. So, more information may need to be included in the reward function. The second problem is that often additional, higher-level requirements have to be considered in addition to the final goal. In the stabilization problem of control theory, an example of such a requirement is a limit on the overshoot of the process's response, which is imposed in addition to the basic requirement of reaching the desired steady-state value. It is an open question how to translate such higher-level requirements into the 'language' of rewards. To do this, it may be

necessary to augment the state signal, because requirements such as the overshoot are dynamical characteristics of the controlled trajectory, while the reward function only evaluates one-step transitions. Another problem is that the discontinuous reward functions typically used in RL may harm the performance in continuous-variable control tasks, as illustrated in Chapter 4.

- Solving DP/RL problems where the state is not directly measurable is an open, active field of research. Such problems are usually called partially observable in the DP/RL literature (Kaelbling et al., 1998; Porta et al., 2006), although from a control theoretical perspective this term is incorrect.[2] An important opportunity for DP/RL in partially measurable problems is to use control theoretical insights. For instance, observability theory clearly identifies which unmeasured state variables are observable from the measured system outputs (Isidori, 1995). If a state variable is necessary for control (e.g., if state feedback control is used, such as in DP/RL), but that state variable is not observable in the control-theoretical sense, then the problem cannot be solved. In such a case, more measurement variables are necessary, which means more sensors need to be installed on the real system. Moreover, there exist efficient, specialized Bayesian techniques to infer the values of unmeasured observable states, e.g., particle filtering (Arulampalam et al., 2002). These estimates could be used in the DP algorithm, instead of relying on techniques that use general Bayesian inference. Such techniques are typically computationally expensive, which means they can only be applied to simple problems. An important caveat is that state estimation, as well as observability analysis, require a model of the system, so they are only applicable to DP. It is an important open question whether they can be extended to the model-free, RL case, and if so, in what way they could be extended.

- Model-based, DP algorithms could be compared analytically and experimentally with other techniques for nonlinear optimal control. These comparisons may lead to useful insights about the strengths and weaknesses of DP and RL. One such optimal control technique is (nonlinear) model-predictive control, which works by computing at every time step an open-loop policy, and performing the action indicated by this policy in the current state (Bertsekas, 2005). This leads to a time-varying policy, whereas in this thesis we have focused on algorithms that aim for an optimal stationary policy.

- It is important to identify a set of suitable benchmark problems for DP/RL for control. Some problems commonly used to benchmark RL algorithms can be solved using classical control design techniques. Two examples are cart-inverted pendulum stabilization (Lagoudakis and Parr, 2003a) and bicycle balancing (Randløv and Alstrøm, 1998; Ernst et al., 2005). RL techniques do have the added advantage that they learn a solution without a model, whereas classical control design requires a model. However, it is still important to find motivating practical problems where DP/RL algorithms work well, and that are very difficult or impossible to solve using other techniques. These

[2] Informally, in the control theoretical context, a state variable is observable if its trajectory can be determined from the trajectories of the actions and of the measurable system outputs. System outputs can include states and/or quantities computed algebraically from the state variables and the actions. A system is observable if all its state variables are observable, and is partially observable if there exist unobservable state variables. So, a partially measurable system (which in DP/RL terminology is called 'partially observable') can actually be fully observable in the control theoretical sense, in which case its state variables can be estimated.

problems should have highly nonlinear, possibly stochastic dynamics around the operating points of interest. For instance, while the bicycle has nonlinear dynamics in general, the dynamics are approximately linear in the restricted state domain for which the problem is typically used (a few degrees of roll angle to either side) (Randløv and Alstrøm, 1998; Ernst et al., 2005), which means that the problem can be solved with linear control design.

- In practice, it is also important to take advantage of any prior knowledge available about the system or the solution. Prior knowledge about the value function could help e.g., to initialize an algorithm that finds BFs automatically, or perhaps to avoid using such an algorithm altogether and to design the BFs directly. Prior knowledge about the policy can be used in policy search algorithms, to find a suitable, simple policy parameterization and therefore to improve the computational efficiency of the optimization algorithm. Policy knowledge can also be used in policy iteration algorithms, as exemplified in Chapter 5. Knowledge about the system dynamics could, e.g., be used to form a partial model and pre-compute an approximate solution with DP. This solution can then be used to initialize a model-free, RL algorithm.

- The application of DP/RL to decentralized and distributed control is an important research direction. Significant research efforts have been made in the related field of multi-agent DP/RL. However, most approaches to multi-agent DP/RL deal with simple, often static (stateless), tasks (Buşoniu et al., 2008a). This is because of the severity of the curse of dimensionality in the multi-agent case, where the size of the joint state-action space grows exponentially with the number of agents. Unfortunately, the static setting is inappropriate for most distributed control problems. Therefore, it is important to also address dynamic problems in multi-agent DP/RL. In this context, it can be very useful to exploit results in distributed and decentralized adaptive control, see e.g., (Jain and Khorrami, 1997; Jiang, 2000; Liu et al., 2007).

Appendix A

The cross-entropy method

This appendix provides an introduction to the cross-entropy (CE) method. First, the CE algorithm for rare-event simulation is given, followed by the CE algorithm for optimization. The presentation is based on Sections 2.3, 2.4, and 4.2 of (Rubinstein and Kroese, 2004).

A.1 Rare-event simulation using the CE method

We consider the problem of estimating the probability of a rare event using sampling. Because the event is rare, its probability is small, and straightforward Monte Carlo sampling is impractical because it would require too many samples. Instead, an importance sampling density[1] has to be chosen that increases the probability of the interesting event. The CE method for rare-event simulation looks for the best importance sampling density from a given, parameterized class of densities, using an iterative approach. At the first iteration, the algorithm draws a set of samples from an initial density. Using these samples, an easier problem than the original one is defined, in which the probability of the rare event is artificially increased so that a good importance sampling density is easier to find. This density is then used to obtain better samples in the next iteration, which allow to define a more difficult problem, and therefore give a sampling density closer to the optimal one, and so on. When the problems considered in every iteration become at least as difficult as the original problem, the current density can be used for importance sampling in the original problem.

Next, the CE method for rare-event simulation is formally described. Let a be a random vector taking values in the space $\mathcal{A}$. Let $\{p(\cdot; v)\}$ be a family of probability densities on $\mathcal{A}$, parameterized by the vector $v \in \mathbb{R}^{N_v}$. Let the nominal parameter $\bar{v} \in \mathbb{R}^{N_v}$ be given. Let $s : \mathcal{A} \to \mathbb{R}$ be a score function. The goal is to estimate the probability that $s(a) \geq \lambda$ where the level $\lambda \in \mathbb{R}$ is given, and a is drawn from the density $p(\cdot; \bar{v})$ with the nominal parameter $\bar{v}$. This probability can be written as:

$$\nu = \mathrm{P}_{a \sim p(\cdot;\bar{v})}(s(a) \geq \lambda) = \mathrm{E}_{a \sim p(\cdot;\bar{v})} \{I(s(a) \geq \lambda)\} \tag{A.1}$$

[1] For simplicity, we will abuse the terminology by using the term 'density' to refer to probability density functions (which describe probabilities of continuous random variables), as well as to probability mass functions (which describe probabilities of discrete random variables).

where $I(s(a) \geq \lambda)$ is the indicator function, equal to 1 whenever $s(a) \geq \lambda$ and 0 otherwise. When this probability is very small (10^{-6} or less), the event $\{s(a) \geq \lambda\}$ is called a rare event.

A straightforward way to estimate ν is to use Monte Carlo simulation. A random sample $a_1, \ldots, a_{N_{\mathrm{CE}}}$ is drawn from $p(\cdot; \bar{v})$, and the estimated value of ν is computed as:

$$\widehat{\nu} = \frac{1}{N_{\mathrm{CE}}} \sum_{i_{\mathrm{s}}=1}^{N_{\mathrm{CE}}} I(s(a_{i_{\mathrm{s}}}) \geq \lambda) \tag{A.2}$$

However, this procedure is computationally inefficient when $\{s(a) \geq \lambda\}$ is a rare event, since a very large number of samples N_{CE} must be used for an accurate estimation of ν. A better way to estimate ν is to draw the samples from an importance sampling density $q(\cdot)$ on $\mathcal{A}$, instead of $p(\cdot; \bar{v})$. The density $q(\cdot)$ is chosen to increase the probability of the interesting event $\{s(a) \geq \lambda\}$, therefore requiring fewer samples for an accurate estimation of ν. Then, ν can be estimated with the importance sampling estimator:

$$\widehat{\nu} = \frac{1}{N_{\mathrm{CE}}} \sum_{i_{\mathrm{s}}=1}^{N_{\mathrm{CE}}} I(s(a_{i_{\mathrm{s}}}) \geq \lambda) \frac{p(a_{i_{\mathrm{s}}}; \bar{v})}{q(a_{i_{\mathrm{s}}})} \tag{A.3}$$

From (A.3) it follows that the importance sampling density:

$$q^*(a) = \frac{I(s(a) \geq \lambda) p(a; \bar{v})}{\nu} \tag{A.4}$$

makes the argument of the summation equal to ν, when substituted in (A.3). Therefore, a single sample a for which $I(s(a) \geq \lambda)$ is nonzero suffices to find ν, and the density $q^*(\cdot)$ is optimal in this sense. It is important to note that the entire procedure is driven by the value of the nominal parameter $\bar{v}$. Therefore, among others, q, q^*, and ν all depend on $\bar{v}$.

The obvious difficulty is that ν is unknown. Moreover, q^* can in general have a complicated shape, which makes it difficult to find. It is often more convenient to choose an importance sampling density from the family of densities $\{p(\cdot; v)\}$. The best importance sampling density in this family can be found by minimizing over the parameter v a measure of the distance between $p(\cdot; v)$ and $q^*(\cdot)$. In the CE method, this measure is the cross-entropy, also known as Kullback-Leibler divergence, and defined as follows:

$$\begin{aligned} \mathcal{D}(q^*(\cdot), p(\cdot; v)) &= \mathrm{E}_{a \sim q^*(\cdot)} \left\{ \ln \frac{q(a)}{p(a; v)} \right\} \\ &= \int q^*(a) \ln q^*(a) \mathrm{d}a - \int q^*(a) \ln p(a; v) \mathrm{d}a \end{aligned} \tag{A.5}$$

Noticing that the first term in this distance does not depend on v and using (A.4) in the second term, the parameter that minimizes the cross-entropy is:

$$\begin{aligned} v^* &= \arg\max_v \int \frac{I(s(a) \geq \lambda) p(a; \bar{v})}{\nu} \ln p(a; v) \mathrm{d}a \\ &= \arg\max_v \mathrm{E}_{a \sim p(\cdot; \bar{v})} \left\{ I(s(a) \geq \lambda) \ln p(a; v) \right\} \end{aligned} \tag{A.6}$$

Unfortunately, the expectation $\mathrm{E}_{a \sim p(\cdot; \bar{v})} \left\{ I(s(a) \geq \lambda) \ln p(a; v) \right\}$ is difficult to compute by direct Monte Carlo sampling, because the indicators $I(s(a) \geq \lambda)$ will still be 0 for a

most of the samples. Therefore, this expectation also has to be computed with importance sampling. Let the importance sampling density be given by the parameter z. Then, the maximization problem (A.6) is rewritten as:

$$v^* = \arg\max_v \mathrm{E}_{a\sim p(\cdot;z)} \{I(s(a) \geq \lambda) W(a;\bar{v},z) \ln p(a;v)\} \tag{A.7}$$

where $W(a;\bar{v},z) = p(a;\bar{v})/p(a;z)$. An approximate solution $\widehat{v}^*$ is computed by drawing a random sample $a_1,\ldots,a_{N_{\mathrm{CE}}}$ from the importance density $p(\cdot;z)$ and solving:

$$\widehat{v}^* = \arg\max_v \frac{1}{N_{\mathrm{CE}}} \sum_{i_\mathrm{s}=1}^{N_{\mathrm{CE}}} I(s(a_{i_\mathrm{s}}) \geq \lambda) W(a;\bar{v},z) \ln p(a_{i_\mathrm{s}};v) \tag{A.8}$$

The problem (A.8) is called the *stochastic counterpart* of (A.7).

Under certain assumptions on $\mathcal{A}$ and $p(\cdot;v)$, the stochastic counterpart can be solved analytically. One particularly important case when this happens is when $p(\cdot;v)$ belongs to the natural exponential family. For instance, when $\{p(\cdot;v)\}$ is the family of Gaussians parameterized by the mean η and the standard deviation σ (so, $v = [\eta,\sigma]^{\mathrm{T}}$), the solution of the stochastic counterpart is the mean and the standard deviation of the best samples:

$$\widehat{\eta} = \frac{\sum_{i_\mathrm{s}=1}^{N_{\mathrm{CE}}} I(s(a_{i_\mathrm{s}}) \geq \lambda) a_{i_\mathrm{s}}}{\sum_{i_\mathrm{s}=1}^{N_{\mathrm{CE}}} I(s(a_{i_\mathrm{s}}) \geq \lambda)} \tag{A.9}$$

$$\widehat{\sigma} = \sqrt{\frac{\sum_{i_\mathrm{s}=1}^{N_{\mathrm{CE}}} I(s(a_{i_\mathrm{s}}) \geq \lambda)(a_{i_\mathrm{s}} - \widehat{\eta})^2}{\sum_{i_\mathrm{s}=1}^{N_{\mathrm{CE}}} I(s(a_{i_\mathrm{s}}) \geq \lambda)}} \tag{A.10}$$

Choosing directly a good importance sampling parameter z is difficult. If z is poorly chosen, most of the indicators $I(s(a_{i_\mathrm{s}}) \geq \lambda)$ in (A.8) will be 0, and the estimated parameter $\widehat{v}^*$ will be a poor approximation of the optimal parameter v^*. To alleviate this difficulty, the CE algorithm uses an iterative approach. Each iteration τ can be viewed as the application of the above methodology with a modified level λ_τ and using the importance density parameter $z = v_{\tau-1}$, where:

- The level λ_τ is chosen at each iteration such that the probability of the event $\{s(a) \geq \lambda_\tau\}$ under the density $p(\cdot;v_{\tau-1})$ is approximately $\varrho_{\mathrm{CE}} \in (0,1)$, with ϱ_{CE} chosen not too small (e.g., $\varrho_{\mathrm{CE}} = 0.05$).
- The parameter $v_{\tau-1}$, $\tau \geq 2$, is the solution of the stochastic counterpart at the previous iteration; v_0 is initialized at $\bar{v}$.

The value λ_τ is computed as the $(1-\varrho_{\mathrm{CE}})$ quantile of the score values of the random sample $a_1,\ldots,a_{N_{\mathrm{CE}}}$ drawn from $p(\cdot;v_{\tau-1})$. If these score values are ordered increasingly and indexed such that $s_1 \leq \cdots \leq s_{N_{\mathrm{CE}}}$, then the $(1-\varrho_{\mathrm{CE}})$ quantile is:

$$\lambda_\tau = s_{\lceil (1-\varrho_{\mathrm{CE}}) N_{\mathrm{CE}} \rceil} \tag{A.11}$$

where $\lceil\cdot\rceil$ rounds the argument to the next greater or equal integer number (ceiling).

When the inequality $\lambda_{\tau^*} \geq \lambda$ is satisfied for some $\tau^* \geq 1$, the rare-event probability is estimated using the density $p(\cdot;v_{\tau^*})$ for importance sampling ($N_1 \in \mathbb{N}^*$):

$$\widehat{\nu} = \frac{1}{N_1} \sum_{i_\mathrm{s}=1}^{N_1} I(s(a_{i_\mathrm{s}}) \geq \lambda) W(a_{i_\mathrm{s}};\bar{v},v_{\tau^*}) \tag{A.12}$$

A.2 CE optimization

Consider the following optimization problem:

$$\max_{a \in \mathcal{A}} s(a) \tag{A.13}$$

where $s : \mathcal{A} \to \mathbb{R}$ is the score function to maximize, and the variable a takes values in the domain $\mathcal{A}$. Denote the maximum by s^*. The CE method for optimization maintains a density with support $\mathcal{A}$. In each iteration, a number of samples are drawn from this density and the score values for these samples are computed. A (smaller) number of samples that have the highest scores are kept, and the remaining samples are discarded. The density is then updated using the selected samples, such that at the next iteration the probability of drawing better samples is increased. The algorithm stops when the score of the worst selected sample no longer improves.

Formally, a family of densities $\{p(\cdot; v)\}$ has to be chosen. This family has support $\mathcal{A}$ and is parameterized by v. An *associated stochastic problem* to (A.13) is the problem of finding the probability:

$$\nu(\lambda) = \mathrm{P}_{a \sim p(\cdot; v')}(s(a) \geq \lambda) = \mathrm{E}_{a \sim p(\cdot; v')} \{I(s(a) \geq \lambda)\} \tag{A.14}$$

where the random vector a has the density $p(\cdot; v')$ for some parameter vector v'. Consider now the problem of estimating $\nu(\lambda)$ for a λ that is close to s^*. Typically, $\{s(a) \geq \lambda\}$ is a rare event. The CE procedure can therefore be used to solve (A.14).

Contrary to the CE method for rare-event simulation, in optimization there is no known nominal λ; its place is taken by s^*, which is unknown. The CE method for optimization circumvents this difficulty by redefining the associated stochastic problem at every iteration τ, using the density with parameter $v_{\tau-1}$, so that λ_τ is likely to converge to s^* as τ increases. Consequently, the stochastic counterpart at iteration τ of CE optimization:

$$v_\tau = \arg\max_v \frac{1}{N_{\mathrm{CE}}} \sum_{i_\mathrm{s}=1}^{N_{\mathrm{CE}}} I(s(a_{i_\mathrm{s}}) \geq \lambda_\tau) \ln p(a_{i_\mathrm{s}}; v) \tag{A.15}$$

is different from the one used in rare-event simulation (A.8), and corresponds to maximizing over v the expectation $\mathrm{E}_{a \sim p(\cdot; v_{\tau-1})} \{I(s(a_{i_\mathrm{s}}) \geq \lambda_\tau) \ln p(a_{i_\mathrm{s}}; v)\}$. This determines the optimal parameter associated with $\mathrm{P}_{a \sim p(\cdot; v_{\tau-1})}(s(a) \geq \lambda_\tau)$, rather than with $\mathrm{P}_{a \sim p(\cdot; \bar{v})}(s(a) \geq \lambda_\tau)$ as in rare-event simulation. So, the term W from (A.7) and (A.8) does not play a role here. The parameter $\bar{v}$, which in rare-event simulation was the unique nominal parameter under which the rare-event probability has to be estimated, no longer plays a role either. Instead, in CE optimization an initial value v_0 of the density parameter is required, which only serves to define the associated stochastic problem at the first iteration, and which can be chosen in a fairly arbitrary way.

The most important parameters in the CE method for optimization are the number of samples N_{CE} and the quantile of best samples used to update the density, ϱ_{CE}. The number of samples should be at least a multiple of the number of parameters N_v, so $N_{\mathrm{CE}} = c_{\mathrm{CE}} N_v$ with $c_{\mathrm{CE}} \in \mathbb{N}$, $c_{\mathrm{CE}} \geq 2$. The parameter ϱ_{CE} can be taken around 0.01 for large numbers of samples, or larger, around $\ln(N_{\mathrm{CE}})/N_{\mathrm{CE}}$, if there are only a few samples ($N_{\mathrm{CE}} < 100$) (Rubinstein and Kroese, 2004).

The CE method for optimization is summarized in Algorithm A.1. Note that in line 7, the stochastic counterpart (A.15) was simplified by using the fact that the samples are already sorted in the ascending order of their scores. When $\varepsilon_{\mathrm{CE}} = 0$, the algorithm terminates when λ remains constant for d_{CE} consecutive iterations. When $\varepsilon_{\mathrm{CE}} > 0$, the algorithm terminates when λ improves for d_{CE} consecutive iterations, and these improvements do not exceed $\varepsilon_{\mathrm{CE}}$. The integer $d_{\mathrm{CE}} > 1$ accounts for the random nature of the algorithm, by ensuring that the latest performance improvements did not decrease below $\varepsilon_{\mathrm{CE}}$ accidentally, but that instead the decrease remains steady for d_{CE} iterations. Because the algorithm is not guaranteed to converge in general, a maximum number of iterations $\tau_{\max}$ has to be chosen to ensure the algorithm terminates in a finite time.

Algorithm A.1 The cross-entropy method for optimization

Input:

family $\{p(\cdot; v)\}$, score function s, initial density parameter v_0

parameters ϱ_{CE}, N_{CE}, d_{CE}, $\varepsilon_{\mathrm{CE}}$, maximum number of iterations $\tau_{\max}$

1: $\tau \leftarrow 1$
2: **repeat**
3: generate sample $a_1, \ldots, a_{N_{\mathrm{CE}}}$ from $p(\cdot; v_{\tau-1})$
4: compute scores $s(a_{i_{\mathrm{s}}})$, $i_{\mathrm{s}} = 1, \ldots, N_{\mathrm{CE}}$
5: reorder and reindex s.t. $s_1 \leq \cdots \leq s_{N_{\mathrm{CE}}}$
6: $\lambda_\tau \leftarrow s_{\lceil (1-\varrho_{\mathrm{CE}}) N_{\mathrm{CE}} \rceil}$
7: $v_\tau \leftarrow \arg\max_v \sum_{i_{\mathrm{s}}=\lceil (1-\varrho_{\mathrm{CE}}) N_{\mathrm{CE}} \rceil}^{N_{\mathrm{CE}}} \ln p(a_{i_{\mathrm{s}}}; v)$
8: $\tau \leftarrow \tau + 1$
9: **until** ($\tau > d_{\mathrm{CE}}$ **and** $0 \leq \lambda_{\tau-\tau'} - \lambda_{\tau-\tau'-1} \leq \varepsilon_{\mathrm{CE}}$, for $\tau' = 1, \ldots, d_{\mathrm{CE}} - 1$) **or** $\tau > \tau_{\max}$

Output: $\widehat{a}^*$, the best sample encountered at any iteration τ; and $\widehat{s}^* = s(\widehat{a}^*)$

CE optimization has found many applications in recent years, e.g., in biomedicine (Mathenya et al., 2007), vector quantization (Boubezoul et al., 2008), vehicle routing (Chepuri and de Mello, 2005), and clustering (Rubinstein and Kroese, 2004). While its convergence is not guaranteed in general, and the theoretical understanding of the CE method is currently limited, the algorithm is usually convergent in practice (Rubinstein and Kroese, 2004). Convergence proofs are available only for combinatorial optimization (Rubinstein and Kroese, 2004; Margolin, 2005; Costa et al., 2007). The most general proof available to date shows that the CE method for combinatorial optimization asymptotically converges, with probability, to a unit mass density, i.e., a density that always generates samples equal to a single point. Furthermore, this convergence point can be made equal to the optimal solution by using an adaptive smoothing technique (Costa et al., 2007). With smoothing, the new parameter v_τ is computed as a convex combination of the solution of (A.15) and the old value $v_{\tau-1}$. The convex update is parameterized by the smoothing parameter $\alpha_{\mathrm{CE}} \in (0, 1]$:

$$v_\tau = (1 - \alpha_{\mathrm{CE}}) v_{\tau-1} + \alpha_{\mathrm{CE}} \arg\max_v \sum_{i_{\mathrm{s}}=\lceil (1-\varrho_{\mathrm{CE}}) N_{\mathrm{CE}} \rceil}^{N_{\mathrm{CE}}} \ln p(a_{i_{\mathrm{s}}}; v) \tag{A.16}$$

Appendix B

Least-squares policy evaluation for Q-functions

The online least-squares policy iteration algorithm introduced in Chapter 5 uses least-squares temporal difference to evaluate policies. Other policy evaluation algorithms can be used instead. One such algorithm is least-squares policy evaluation, which was originally developed for V-functions (Bertsekas and Ioffe, 1996; Bertsekas, 2007). In this appendix, we extend it to compute Q-functions instead of V-functions. Section B.1 presents this new algorithm, called least-squares policy evaluation for Q-functions. Section B.2 shows how this algorithm can be used in online policy iteration.

B.1 Least-squares policy evaluation for Q-functions

The derivation of least-squares policy evaluation for Q-functions (LSPE-Q) is similar to the derivation of least-squares temporal difference for Q-functions (LSTD-Q), which was presented in Section 3.4.1. Note that, for simplicity, we use a deterministic formalism to introduce LSPE-Q. Nevertheless, the final algorithm can be applied directly to the stochastic case, as well.

Assume, like in Section 3.4.1, that the state space X and the action space U have a finite number of elements, $X = \{x_1, \ldots, x_{\bar{N}}\}$, $U = \{u_1, \ldots, u_{\bar{M}}\}$. Consider a linearly parameterized approximator with n state-action basis functions (BFs) $\phi_1, \ldots, \phi_n : X \times U \to \mathbb{R}$. We will use a matrix representation of the BFs and a vector representation of the Q-function. The matrix representation of the BFs, $\boldsymbol{\phi} \in \mathbb{R}^{\bar{N}\bar{M} \times n}$, is defined by $\boldsymbol{\phi}_{[i,j],l} = \phi_l(x_i, u_j)$. Here, $[i,j]$ is used to denote the single-dimensional index corresponding to i and j, which can be computed with $[i,j] = i + (j-1)\bar{N}$. The vector representation of a Q-function Q is $\boldsymbol{Q} \in \mathbb{R}^{\bar{N}\bar{M}}$ with $\boldsymbol{Q}_{[i,j]} = Q(x_i, u_j)$. The Q-vector corresponding to a parameter θ is $\widehat{\boldsymbol{Q}} = \boldsymbol{\phi}\,\theta$. The set of exactly representable Q-vectors is the space spanned by the BFs, and can be written as $\widehat{\boldsymbol{\mathcal{Q}}} = \{\boldsymbol{\phi}\,\theta \mid \theta \in \mathbb{R}^n\}$, $\widehat{\boldsymbol{\mathcal{Q}}} \subset \mathbb{R}^{\bar{N}\bar{M}}$.

The *projected Bellman equation* corresponding to the BFs $\boldsymbol{\phi}$ is:

$$\boldsymbol{\phi}\theta^* = P^w \boldsymbol{T}^{\boldsymbol{h}} \boldsymbol{\phi}\theta^* \tag{B.1}$$

where $\boldsymbol{T^h}$ is the matrix form (3.14) of the policy evaluation mapping T^h (2.22), and P^w is a weighted Euclidean (least-squares) projection operator, which takes any Q-vector and projects it onto $\widehat{\boldsymbol{\mathcal{Q}}}$. The weight function $w : X \times U \to [0, 1]$ controls the distribution over the state-action space of the error (difference) between a Q-vector $\boldsymbol{Q}$ and its projection $P^w\boldsymbol{Q}$. It satisfies $\sum_{x,u} w(x, u) = 1$. The projection matrix can be written in a closed form as $P^w = \boldsymbol{\phi}(\boldsymbol{\phi}^{\mathrm{T}}\boldsymbol{w}\boldsymbol{\phi})^{-1}\boldsymbol{\phi}^{\mathrm{T}}\boldsymbol{w}$, where $\boldsymbol{w} \in \mathbb{R}^{\bar{N}\bar{M}\times\bar{N}\bar{M}}$ is a diagonal matrix representation of w, with $\boldsymbol{w}_{[i,j],[i,j]} = w(x_i, u_j)$.

Instead of aiming directly at a solution θ^* of (B.1), like LSTD-Q does, the (idealized) LSPE-Q algorithm applies (B.1) as an iterative update:

$$\boldsymbol{\phi}\theta_{l_\mathrm{s}} = P^w\boldsymbol{T^h}\boldsymbol{\phi}\theta_{l_\mathrm{s}-1}, \quad l_\mathrm{s} \geq 1 \tag{B.2}$$

starting from an arbitrary parameter vector $\theta_0 \in \mathbb{R}^n$. The iterative update (B.2) converges with probability 1 to θ^*, under conditions similar to those required for the convergence of least-squares policy evaluation for V-functions (Bertsekas, 2007). Because θ_{l_s} asymptotically converges to θ^* as $l_\mathrm{s} \to \infty$, the update (B.2) becomes asymptotically equivalent to (B.1).

By substituting $\boldsymbol{T^h}$ from (3.14) and the closed-form expression for P^w, the update (B.2) becomes:

$$\boldsymbol{\phi}\theta_{l_\mathrm{s}} = \boldsymbol{\phi}(\boldsymbol{\phi}^{\mathrm{T}}\boldsymbol{w}\boldsymbol{\phi})^{-1}\boldsymbol{\phi}^{\mathrm{T}}\boldsymbol{w}(\boldsymbol{\rho} + \gamma\boldsymbol{f}\boldsymbol{h}\boldsymbol{\phi}\theta_{l_\mathrm{s}-1})$$

and further, by left-multiplication with $\boldsymbol{\phi}^{\mathrm{T}}\boldsymbol{w}$:

$$\boldsymbol{\phi}^{\mathrm{T}}\boldsymbol{w}\boldsymbol{\phi}\theta_{l_\mathrm{s}} = \boldsymbol{\phi}^{\mathrm{T}}\boldsymbol{w}\boldsymbol{\rho} + \gamma\boldsymbol{\phi}^{\mathrm{T}}\boldsymbol{w}\boldsymbol{f}\boldsymbol{h}\boldsymbol{\phi}\theta_{l_\mathrm{s}-1}$$

where $\boldsymbol{f}$ is a matrix representation of the dynamics f, $\boldsymbol{h}$ a matrix representation of the policy h, and $\boldsymbol{\rho}$ a vector representation of the reward function ρ (for the definitions of $\boldsymbol{f}$, $\boldsymbol{h}$, and $\boldsymbol{\rho}$, see Section 3.4.1). Denote $\bar{\Gamma} = \boldsymbol{\phi}^{\mathrm{T}}\boldsymbol{w}\boldsymbol{\phi}$, $z = \boldsymbol{\phi}^{\mathrm{T}}\boldsymbol{w}\boldsymbol{\rho}$, and $\Lambda = \gamma\boldsymbol{\phi}^{\mathrm{T}}\boldsymbol{w}\boldsymbol{f}\boldsymbol{h}\boldsymbol{\phi}$. Then, the update equation becomes:

$$\bar{\Gamma}\theta_{l_\mathrm{s}} = z + \Lambda\theta_{l_\mathrm{s}-1} \tag{B.3}$$

The matrices $\bar{\Gamma}$ and Λ, and the vector z, can be written as sums of $\bar{N}\bar{M}$ terms, one for each state-action pair:

$$\begin{aligned}
\bar{\Gamma} &= \sum_{i=1}^{\bar{N}}\sum_{j=1}^{\bar{M}} \phi(x_i, u_j)w(x_i, u_j)\phi^{\mathrm{T}}(x_i, u_j) \\
z &= \sum_{i=1}^{\bar{N}}\sum_{j=1}^{\bar{M}} \phi(x_i, u_j)w(x_i, u_j)\rho(x_i, u_j) \\
\Lambda &= \gamma\sum_{i=1}^{\bar{N}}\sum_{j=1}^{\bar{M}} \phi(x_i, u_j)w(x_i, u_j)\phi^{\mathrm{T}}(f(x_i, u_j), h(f(x_i, u_j)))
\end{aligned} \tag{B.4}$$

A practical implementation of LSPE-Q uses a set of samples $\{(x_{l_\mathrm{s}}, u_{l_\mathrm{s}}, x'_{l_\mathrm{s}} = f(x_{l_\mathrm{s}}, u_{l_\mathrm{s}}), r_{l_\mathrm{s}} = \rho(x_{l_\mathrm{s}}, u_{l_\mathrm{s}})) \,|\, l_\mathrm{s} = 1, \ldots, n_\mathrm{s}\}$ with $(x_{l_\mathrm{s}}, u_{l_\mathrm{s}})$ drawn from a density defined by the weight function w: the probability of drawing the sample $(x_{l_\mathrm{s}}, u_{l_\mathrm{s}})$ is $w(x_{l_\mathrm{s}}, u_{l_\mathrm{s}})$. From these samples, $\bar{\Gamma}$, z, and Λ are incrementally approximated using the following update rules, derived

from (B.4):

$$\begin{aligned}
\bar{\Gamma}_0 &= 0, \quad z_0 = 0, \quad \Lambda_0 = 0 \\
\bar{\Gamma}_{l_\mathrm{s}} &= \bar{\Gamma}_{l_\mathrm{s}-1} + \phi(x_{l_\mathrm{s}}, u_{l_\mathrm{s}})\phi^\mathrm{T}(x_{l_\mathrm{s}}, u_{l_\mathrm{s}}) \\
z_{l_\mathrm{s}} &= z_{l_\mathrm{s}-1} + \phi(x_{l_\mathrm{s}}, u_{l_\mathrm{s}})r_{l_\mathrm{s}} \\
\Lambda_{l_\mathrm{s}} &= \Lambda_{l_\mathrm{s}-1} + \gamma\phi(x_{l_\mathrm{s}}, u_{l_\mathrm{s}})\phi^\mathrm{T}(x'_{l_\mathrm{s}}, h(x'_{l_\mathrm{s}}))
\end{aligned} \tag{B.5}$$

After each sample has been processed, LSPE-Q updates the parameter vector using:

$$\frac{1}{l_\mathrm{s}}\bar{\Gamma}_{l_\mathrm{s}}\theta_{l_\mathrm{s}} = \frac{1}{l_\mathrm{s}}z_{l_\mathrm{s}} + \frac{1}{l_\mathrm{s}}\Lambda_{l_\mathrm{s}}\theta_{l_\mathrm{s}-1} \tag{B.6}$$

It is true that $\lim_{l_\mathrm{s}\to\infty}\frac{1}{l_\mathrm{s}}\bar{\Gamma}_{l_\mathrm{s}} = \bar{\Gamma}$, $\lim_{l_\mathrm{s}\to\infty}\frac{1}{l_\mathrm{s}}z_{l_\mathrm{s}} = z$, and $\lim_{l_\mathrm{s}\to\infty}\frac{1}{l_\mathrm{s}}\Lambda_{l_\mathrm{s}} = \Lambda$. Therefore, (B.6) asymptotically becomes equivalent to (B.3), as $l_\mathrm{s} \to \infty$. Since (B.3) is in turn asymptotically equivalent to (B.1), LSPE-Q and LSTD-Q are asymptotically equivalent. Note that the parameter vector can be updated after an arbitrary number of samples has been processed (instead of after every sample) and the two algorithms are still asymptotically equivalent.

In practice, the number of samples is always finite, so the normalization by the number of samples is not necessary[1], and (B.6) is equivalent to:

$$\bar{\Gamma}_{l_\mathrm{s}}\theta_{l_\mathrm{s}} = z_{l_\mathrm{s}} + \Lambda_{l_\mathrm{s}}\theta_{l_\mathrm{s}-1} \tag{B.7}$$

Note that the matrices $\bar{\Gamma}$ and Λ, as well as the vector z, have manageable sizes which only depend on the number of BFs n, and not on the number of states $\bar{N}$ or actions $\bar{M}$. Also, the updates (B.5) and (B.7) can be applied without any change to state-action spaces with infinitely many elements (e.g., continuous).

B.2 Online policy iteration with LSPE-Q

Because LSTD-Q and LSPE-Q become equivalent as the number of samples increases, there is little reason to prefer one over the other for offline algorithms that use large batches of samples, such as least-squares policy iteration (LSPI). However, like their V-function counterparts (Bertsekas, 2007), LSTD-Q and LSPE-Q can produce quite different parameters when only a few samples are available for the evaluation of every policy. This means that their behavior can be very different when they are used in online (optimistic) policy iteration. Recall that in online policy iteration, policy improvements are performed at short intervals, during which only a small number of samples can be processed. An analytical or experimental comparison between LSTD-Q and LSPE-Q in the context of online policy iteration is an important research direction.

As an illustrative example, Algorithm B.1 outlines online (optimistic) policy iteration with LSPE-Q policy evaluation and ε-greedy exploration. This algorithm is similar to online LSPI (Algorithm 5.2), which was studied in Chapter 5. The algorithm has similar parameters to online LSPI: the interval between policy improvements K_θ, the exploration schedule ε_k, and the constant $\beta_{\bar{\Gamma}}$, used to ensure that $\bar{\Gamma}$ is not singular for the first parameter updates.

Note that $\bar{\Gamma}$, z, and Λ are never reset after the policy is improved. The algorithm could reset them and reuse old samples to compute new $\bar{\Gamma}$, z, and Λ for each policy.[2] However,

[1] Normalization may still be useful to avoid numerical errors.

[2] In fact, only Λ depends on the policy and has to be recomputed. However, if different sets of samples were reused for every policy, then $\bar{\Gamma}$ and z would need to be recomputed as well.

Algorithm B.1 Online policy iteration with LSPE-Q and ε-greedy exploration

Input:

BFs $\phi_l : X \times U \to \mathbb{R}$, $l = 1, \ldots, n$
discount factor γ; policy improvement interval K_θ; exploration schedule ε_k, $k \geq 0$
a small constant $\beta_{\bar{\Gamma}} > 0$

$\ell \leftarrow 0$; initialize policy h_0; initialize parameters θ_0
$\bar{\Gamma}_0 \leftarrow \beta_{\bar{\Gamma}} \cdot I_{n \times n}$; $z_0 \leftarrow 0_n$; $\Lambda_0 \leftarrow 0_{n \times n}$
measure initial state x_0
for each time step $k \geq 0$ **do**
    $u_k \leftarrow \begin{cases} h_\ell(x_k) & \text{with probability } (1 - \varepsilon_k) \text{ (exploit)} \\ \text{a uniform random action in } U & \text{with probability } \varepsilon_k \text{ (explore)} \end{cases}$
    apply u_k, measure next state x_{k+1} and reward r_{k+1}
    $\bar{\Gamma}_{k+1} \leftarrow \bar{\Gamma}_k + \phi(x_k, u_k)\phi^{\mathrm{T}}(x_k, u_k)$
    $z_{k+1} \leftarrow z_k + \phi(x_k, u_k) r_{k+1}$
    $\Lambda_{k+1} \leftarrow \Lambda_k + \gamma\phi(x_k, u_k)\phi^{\mathrm{T}}(x_{k+1}, h_\ell(x_{k+1}))$
    compute θ_{k+1} by solving $\bar{\Gamma}_{k+1}\theta_{k+1} = z_{k+1} + \Lambda_{k+1}\theta_k$
    if $k = (\ell + 1)K_\theta$ **then** ▷ k is the next positive multiple of K_θ
        $h_{\ell+1}(x) \leftarrow \arg\max_u \phi(x, u)\theta_{k+1}$ ▷ policy improvement
        $\ell \leftarrow \ell + 1$
    end if
end for

due to computational cost considerations, reusing samples is not always practical when the algorithm has to be run online. Therefore, the algorithm cannot afford to discard the values of $\bar{\Gamma}$, z, and Λ, because the few samples that arrive before the next policy improvement are not sufficient to construct informative new estimates. Instead, the values of $\bar{\Gamma}$, z, and Λ are preserved when the policy is improved. The assumption underlying this heuristic is that the Q-functions of subsequent policies are similar, which means that the previous values of $\bar{\Gamma}$, z, and Λ are representative also for the new policy.

Bibliography

Adams, B., Banks, H., Kwon, H.-D., and Tran, H. (2004). Dynamic multidrug therapies for HIV: Optimal and STI control approaches. *Mathematical Biosciences and Engineering*, 1(2):223–241.

Arulampalam, S., Maskell, S., Gordon, N. J., and Clapp, T. (2002). A tutorial on particle filters for on-line nonlinear / non-Gaussian Bayesian tracking. *IEEE Transactions on Signal Processing*, 50:174–188.

Baird, L. and Moore, A. (1999). Gradient descent for general reinforcement learning. In Kearns, M. S., Solla, S. A., and Cohn, D. A., editors, *Advances in Neural Information Processing Systems 11*, pages 968–974. MIT Press.

Barash, D. (1999). A genetic search in policy space for solving Markov decision processes. In *AAAI Spring Symposium on Search Techniques for Problem Solving under Uncertainty and Incomplete Information*, Palo Alto, US.

Barto, A. G., Sutton, R. S., and Anderson, C. W. (1983). Neuronlike adaptive elements than can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 13(5):833–846.

Berenji, H. R. and Khedkar, P. (1992). Learning and tuning fuzzy logic controllers through reinforcements. *IEEE Transactions on Neural Networks*, 3(5):724–740.

Berenji, H. R. and Vengerov, D. (2003). A convergent actor-critic-based FRL algorithm with application to power management of wireless transmitters. *IEEE Transactions on Fuzzy Systems*, 11(4):478–485.

Bertsekas, D. P. (2005). Dynamic programming and suboptimal control: A survey from ADP to MPC. *European Journal of Control*, 11(4–5). Special issue for the CDC-ECC-05 in Seville, Spain.

Bertsekas, D. P. (2007). *Dynamic Programming and Optimal Control*, volume 2. Athena Scientific, 3nd edition.

Bertsekas, D. P. and Ioffe, S. (1996). Temporal differences-based policy iteration and applications in neuro-dynamic programming. Technical Report LIDS-P-2349, Massachusets Institute of Technology.

Bertsekas, D. P. and Tsitsiklis, J. N. (1996). *Neuro-Dynamic Programming*. Athena Scientific.

Borkar, V. (2005). An actor-critic algorithm for constrained Markov decision processes. *Systems & Control Letters*, 54:207–213.

Boubezoul, A., Paris, S., and Ouladsine, M. (2008). Application of the cross entropy method to the GLVQ algorithm. *Pattern Recognition*, 41(10):3173–3178.

Buşoniu, L., Babuška, R., and De Schutter, B. (2006a). Multi-agent reinforcement learning: A survey. In *Proceedings 9th International Conference of Control, Automation, Robotics, and Vision (ICARCV-06)*, pages 527–532, Singapore.

Buşoniu, L., Babuška, R., and De Schutter, B. (2008a). A comprehensive survey of multi-agent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics. Part C: Applications and Reviews*, 38(2):156–172.

Buşoniu, L., De Schutter, B., and Babuška, R. (2005). Multiagent reinforcement learning with adaptive state focus. In *Proceedings 17th Belgian-Dutch Conference on Artificial Intelligence (BNAIC-05)*, pages 35–42, Brussels, Belgium.

Buşoniu, L., De Schutter, B., and Babuška, R. (2006b). Decentralized reinforcement learning control of a robotic manipulator. In *Proceedings 9th International Conference of Control, Automation, Robotics, and Vision (ICARCV-06)*, pages 1347–1352, Singapore.

Buşoniu, L., Ernst, D., De Schutter, B., and Babuška, R. (2007a). Continuous-state reinforcement learning with fuzzy approximation. In *Adaptive Learning Agents and Multi-Agent Systems (ALAMAS-07) Symposium*, pages 21–35, Maastricht, The Netherlands.

Buşoniu, L., Ernst, D., De Schutter, B., and Babuška, R. (2007b). Fuzzy approximation for convergent model-based reinforcement learning. In *Proceedings 2007 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE-07)*, pages 968–973, London, UK.

Buşoniu, L., Ernst, D., De Schutter, B., and Babuška, R. (2008b). Consistency of fuzzy model-based reinforcement learning. In *Proceedings 2008 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE-08)*, pages 518–524, Hong Kong.

Buşoniu, L., Ernst, D., De Schutter, B., and Babuška, R. (2008c). Continuous-state reinforcement learning with fuzzy approximation. In Tuyls, K., Nowé, A., Guessoum, Z., and Kudenko, D., editors, *Adaptive Agents and Multi-Agent Systems III*, volume 4865 of *Lecture Notes in Computer Science*, pages 27–43. Springer.

Buşoniu, L., Ernst, D., De Schutter, B., and Babuška, R. (2008d). Fuzzy partition optimization for approximate fuzzy Q-iteration. In *Proceedings 17th IFAC World Congress (IFAC-08)*, pages 5629–5634, Seoul, Korea.

Chang, H. S., Fu, M. C., Hu, J., and Marcus, S. I. (2007). *Simulation-Based Algorithms for Markov Decision Processes.* Springer.

Chepuri, K. and de Mello, T. H. (2005). Solving the vehicle routing problem with stochastic demands using the cross-entropy method. *Annals of Operations Research*, 134(1):153–181.

Chin, H. H. and Jafari, A. A. (1998). Genetic algorithm methods for solving the best stationary policy of finite Markov decision processes. In *Proceedings 30th Southeastern Symposium on System Theory*, pages 538–543, Morgantown, US.

Chow, C.-S. and Tsitsiklis, J. (1991). An optimal one-way multigrid algorithm for discrete-time stochastic control. *IEEE Transactions on Automatic Control*, 36(8):898–914.

Costa, A., Jones, O. D., and Kroese, D. (2007). Convergence properties of the cross-entropy method for discrete optimization. *Operations Research Letters*, 35:573–580.

del R. Millán, J., Posenato, D., and Dedieu, E. (2002). Continuous-action Q-learning. *Machine Learning*, 49(2-3):247–265.

Edelman, A. and Murakami, H. (1995). Polynomial roots from companion matrix eigenvalues. *Mathematics of Computation*, 64:763–776.

Ernst, D. (2003). *Near Optimal Closed-loop Control. Application to Electric Power Systems.* PhD thesis, University of Liège, Belgium.

Ernst, D., Geurts, P., and Wehenkel, L. (2005). Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6:503–556.

Ernst, D., Stan, G.-B., Gonçalves, J., and Wehenkel, L. (2006). Clinical data based optimal STI strategies for HIV: a reinforcement learning approach. In *Proceedings 45th IEEE Conference on Decision & Control*, pages 667–672, San Diego, US.

Geramifard, A., Bowling, M., Zinkevich, M., and Sutton, R. S. (2007). iLSTD: Eligibility traces & convergence analysis. In Schölkopf, B., Platt, J., and Hofmann, T., editors, *Advances in Neural Information Processing Systems 19*, pages 440–448. MIT Press.

Geramifard, A., Bowling, M. H., and Sutton, R. S. (2006). Incremental least-squares temporal difference learning. In *Proceedings 21st National Conference on Artificial Intelligence and 18th Innovative Applications of Artificial Intelligence Conference (AAAI-06)*, pages 356–361, Boston, US.

Glorennec, P. Y. (2000). Reinforcement learning: An overview. In *Proceedings European Symposium on Intelligent Techniques (ESIT-00)*, pages 17–35, Aachen, Germany.

Gomez, F. J., Schmidhuber, J., and Miikkulainen, R. (2006). Efficient non-linear control through neuroevolution. In *Proceedings 17th European Conference on Machine Learning (ECML-06)*, volume 4212 of *Lecture Notes in Computer Science*, pages 654–662, Berlin, Germany.

Gonzalez, R. and Rofman, E. (1985). On deterministic control problems: An approximation procedure for the optimal cost I. The stationary problem. *SIAM Journal on Control and Optimization*, 23(2):242–266.

Gordon, G. (1995). Stable function approximation in dynamic programming. In *Proceedings 12th International Conference on Machine Learning (ICML-95)*, pages 261–268, Tahoe City, US.

Grüne, L. (2004). Error estimation and adaptive discretization for the discrete stochastic Hamilton-Jacobi-Bellman equation. *Numerical Mathematics*, 99:85–112.

Horiuchi, T., Fujino, A., Katai, O., and Sawaragi, T. (1996). Fuzzy interpolation-based Q-learning with continuous states and actions. In *Proceedings 5th IEEE International Conference on Fuzzy Systems (FUZZ-IEEE-96)*, pages 594–600, New Orleans, US.

Isidori, A. (1995). *Nonlinear Control Systems*. Springer Verlag, 3rd edition.

Jaakkola, T., Jordan, M. I., and Singh, S. P. (1994). On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, 6(6):1185–1201.

Jain, S. and Khorrami, F. (1997). Decentralized adaptive control of a class of large-scale interconnected nonlinear systems. *IEEE Transactions on Automatic Control*, 42(2):136–154.

Jiang, Z.-P. (2000). Decentralized and adaptive nonlinear tracking of large-scale systems via output feedback. *IEEE Transactions on Automatic Control*, 45(11):2122–2128.

Jouffe, L. (1998). Fuzzy inference system learning by reinforcement methods. *IEEE Transactions on Systems, Man, and Cybernetics—Part C: Applications and Reviews*, 28(3):338–355.

Jung, T. and Polani, D. (2007). Kernelizing LSPE(λ). In *Proceedings 2007 IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pages 338–345, Honolulu, Hawaii, US.

Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134.

Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285.

Keller, P. W., Mannor, S., and Precup, D. (2006). Automatic basis function construction for approximate dynamic programming and reinforcement learning. In *Proceedings 23rd International Conference on Machine Learning (ICML-06)*, pages 449–456, Pittsburgh, US.

Kirk, W. A. and Khamsi, M. A. (2001). *An Introduction to Metric Spaces and Fixed Point Theory*. John Wiley.

Konda, V. (2002). *Actor-Critic Algorithms*. PhD thesis, Massachusets Institute of Technology, Cambridge, US.

Konda, V. R. and Tsitsiklis, J. N. (2000). Actor-critic algorithms. In Solla, S. A., Leen, T. K., and Müller, K.-R., editors, *Advances in Neural Information Processing Systems 12*, pages 1008–1014. MIT Press.

Konda, V. R. and Tsitsiklis, J. N. (2003). On actor-critic algorithms. *SIAM Journal on Control and Optimization*, 42(4):1143–1166.

Lagoudakis, M., Parr, R., and Littman, M. (2002). Least-squares methods in reinforcement learning for control. In *Methods and Applications of Artificial Intelligence*, volume 2308 of *Lecture Notes in Artificial Intelligence*, pages 249–260. Springer.

Lagoudakis, M. G. and Parr, R. (2003a). Least-squares policy iteration. *Journal of Machine Learning Research*, 4:1107–1149.

Lagoudakis, M. G. and Parr, R. (2003b). Reinforcement learning as classification: Leveraging modern classifiers. In *Proceedings 20th International Conference on Machine Learning (ICML-03)*, pages 424–431. Washington, US.

Lin, C.-K. (2003). A reinforcement learning adaptive fuzzy controller for robots. *Fuzzy Sets and Systems*, 137(3):339–352.

Lisziewicz, J., Rosenberg, E., and Liebermann, J. (1999). Control of HIV despite the discontinuation of antiretroviral therapy. *New England Journal of Medicine*, 340:1683–1684.

Liu, S.-J., Zhang, J.-F., and Jiang, Z.-P. (2007). Decentralized adaptive output-feedback stabilization for large-scale stochastic nonlinear systems. *Automatica*, 43(2):238–251.

Mahadevan, S. (2005). Samuel meets Amarel: Automating value function approximation using global state space analysis. In *Proceedings 20th National Conference on Artificial Intelligence and the 17th Innovative Applications of Artificial Intelligence Conference (AAAI-05)*, pages 1000–1005, Pittsburgh, US.

Mahadevan, S. and Maggioni, M. (2007). Proto-value functions: A Laplacian framework for learning representation and control in Markov decision processes. *Journal of Machine Learning Research*, 8:2169–2231.

Mannor, S., Rubinstein, R. Y., and Gat, Y. (2003). The cross-entropy method for fast policy search. In *Proceedings 20th International Conference on Machine Learning (ICML-03)*, pages 512–519, Washington, US.

Marbach, P. and Tsitsiklis, J. N. (2003). Approximate gradient methods in policy-space optimization of Markov reward processes. *Discrete Event Dynamic Systems: Theory and Applications*, 13:111–148.

Margolin, L. (2005). On the convergence of the cross-entropy method. *Annals of Operations Research*, 134(1):201–214.

Mathenya, M. E., Resnic, F. S., Arora, N., and Ohno-Machado, L. (2007). Effects of SVM parameter optimization on discrimination and calibration for post-procedural PCI mortality. *Journal of Biomedical Informatics*, 40(6):688–697.

Menache, I., Mannor, S., and Shimkin, N. (2005). Basis function adaptation in temporal difference reinforcement learning. *Annals of Operations Research*, 134:215–238.

Moore, A. W. and Atkeson, C. R. (1995). The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. *Machine Learning*, 21:199–233.

Munos, R. (1997). Finite-element methods with local triangulation refinement for continuous reinforcement learning problems. In *Proceedings 9th European Conference on Machine Learning (ECML-97)*, pages 170–182, Prague, Czech Republic.

Munos, R. (2006). Policy gradient in continuous time. *Journal of Machine Learning Research*, 7:771–791.

Munos, R. and Moore, A. (2002). Variable-resolution discretization in optimal control. *Machine Learning*, 49(2-3):291–323.

Munos, R. and Szepesvári, C. (2008). Finite time bounds for fitted value iteration. *Journal of Machine Learning Research*, 9:815–857.

Nakamura, Y., Moria, T., Satoc, M., and Ishiia, S. (2007). Reinforcement learning for a biped robot based on a CPG-actor-critic method. *Neural Networks*, 20:723–735.

Ng, A. Y., Harada, D., and Russell, S. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings 16th International Conference on Machine Learning (ICML-99)*, pages 278–287, Bled, Slovenia.

Ng, A. Y. and Jordan, M. I. (2000). PEGASUS: A policy search method for large MDPs and POMDPs. In *Proceedings 16th Conference in Uncertainty in Artificial Intelligence (UAI-00)*, pages 406–415, Palo Alto, US.

Nocedal, J. and Wright, S. J. (2006). *Numerical Optimization*. Springer-Verlag, 2nd edition.

Ormoneit, D. and Sen, S. (2002). Kernel-based reinforcement learning. *Machine Learning*, 49(2–3):161–178.

Porta, J. M., Vlassis, N., Spaan, M. T., and Poupart, P. (2006). Point-based value iteration for continuous POMDPs. *Journal of Machine Learning Research*, 7:2329–2367.

Randløv, J. and Alstrøm, P. (1998). Learning to drive a bicycle using reinforcement learning and shaping. In *Proceedings 15th International Conference on Machine Learning (ICML-98)*, pages 463–471, Madison, US.

Ratitch, B. and Precup, D. (2004). Sparse distributed memories for on-line value-based reinforcement learning. In *Proceedings 15th European Conference on Machine Learning (ECML-04)*, volume 3201 of *Lecture Notes in Computer Science*, pages 347–358, Pisa, Italy.

Riedmiller, M., Peters, J., and Schaal, S. (2007). Evaluation of policy gradient methods and variants on the cart-pole benchmark. In *Proceedings 2007 IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL-07)*, pages 254–261, Honolulu, Hawaii.

Rubinstein, R. Y. and Kroese, D. P. (2004). *The Cross Entropy Method. A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation, and Machine Learning*. Information Science and Statistics. Springer.

Rummery, G. A. and Niranjan, M. (1994). On-line Q-learning using connectionist systems. Technical Report 166, Engineering Department, Cambridge University, Cambridge, UK.

Rust, J. (1996). Numerical dynamic programming in economics. In Amman, H. M., Kendrick, D. A., and Rust, J., editors, *Handbook of Computational Economics*, volume 1, chapter 14, pages 619–729. Elsevier.

Santos, M. S. and Vigo-Aguiar, J. (1998). Analysis of a numerical dynamic programming algorithm applied to economic models. *Econometrica*, 66(2):409–426.

Sherman, J. and Morrison, W. J. (1950). Adjustment of an inverse matrix corresponding to a change in one element of a given matrix. *Annals of Mathematical Statistics*, 21(1):124–127.

Singh, S. P., Jaakkola, T., and Jordan, M. I. (1995). Reinforcement learning with soft state aggregation. In Tesauro, G., Touretzky, D. S., and Leen, T. K., editors, *Advances in Neural Information Processing Systems 7*, pages 361–368.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44.

Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings 7th International Conference on Machine Learning (ICML-90)*, pages 216–224, Austin, US.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press.

Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In Solla, S. A., Leen, T. K., and Müller, K.-R., editors, *Advances in Neural Information Processing Systems 12*, pages 1057–1063. MIT Press.

Szepesvári, C. and Smart, W. D. (2004). Interpolation-based Q-learning. In *Proceedings 21st International Conference on Machine Learning (ICML-04)*, pages 791–798, Bannf, Canada.

Takagi, T. and Sugeno, M. (1985). Fuzzy identification of systems and its applications to modeling and control. *IEEE Transactions on Systems, Man, and Cybernetics*, 15:116–132.

Touzet, C. F. (1997). Neural reinforcement learning for behaviour synthesis. *Robotics and Autonomous Systems*, 22(3–4):251–81.

Tsitsiklis, J. N. and Van Roy, B. (1996). Feature-based methods for large scale dynamic programming. *Machine Learning*, 22(1–3):59–94.

Vengerov, D., Bambos, N., and Berenji, H. R. (2005). A fuzzy reinforcement learning approach to power control in wireless transmitters. *IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics*, 35(4):768–778.

Waldock, A. and Carse, B. (2008). Fuzzy Q-learning with an adaptive representation. In *Proceedings 2008 IEEE World Congress on Computational Intelligence (WCCI-08)*, pages 720–725, Hong Kong.

Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards*. PhD thesis, King's College, Oxford.

Watkins, C. J. C. H. and Dayan, P. (1992). Q-learning. *Machine Learning*, 8:279–292.

Whiteson, S. and Stone, P. (2006). Evolutionary function approximation for reinforcement learning. *Journal of Machine Learning Research*, 7:877–917.

Wiering, M. (2004). Convergence and divergence in standard and averaging reinforcement learning. In *Proceedings 15th European Conference on Machine Learning (ECML'04)*, pages 477–488, Pisa, Italy.

Williams, R. J. and Baird, L. C. (1994). Tight performance bounds on greedy policies based on imperfect value functions. In *Proceedings 8th Yale Workshop on Adaptive and Learning Systems*, pages 108–113, New Haven, US.

Wodarz, D. and Nowak, M. A. (1999). Specific therapy regimes could lead to long-term immunological control of HIV. *Proceedings of the National Academy of Sciences of the United States of America*, 96(25):14464–14469.

Xu, X., Hu, D., and Lu, X. (2007). Kernel-based least-squares policy iteration for reinforcement learning. *IEEE Transactions on Neural Networks*, 18(4):973–992.

Glossary

Conventions

The following **terminology** conventions are used in this thesis:

- The term 'mapping' is used to refer to functions that work with other functions as inputs and/or outputs; as well as to compositions of such functions. The term is used to differentiate mappings from ordinary functions, which only work with numerical scalars, vectors, or matrices.

- We refer to state value functions as 'V-functions', and to state-action value functions as 'Q-functions', to clearly differentiate between the two types of value functions. We use the name 'value function' to refer to Q-functions and V-functions collectively. Similarly, 'V-iteration' and 'Q-iteration' are used to refer to value iteration algorithms that work with V-functions and Q-functions, respectively.

The following conventions are used for **symbols and notations**:

- The standard control-theoretical notation is preferred to the artificial intelligence notation typically used in reinforcement learning. For instance, the state is denoted by x, the state space by X, the control action by u, the action space by U, and the process dynamics by f. Furthermore, we denote reward functions by ρ, to distinguish them from the instantaneous rewards r and from the returns R. Control policies are denoted by h.

- All the vectors used in this thesis are column vectors. The transpose of a vector is denoted by the superscript T. For instance, the transpose of θ is θ^{T}.

- Boldface notation is used for vector or matrix representations of functions and mappings, e.g., $\boldsymbol{f}$ is a matrix representation of the dynamics f. However, ordinary vectors and matrices are displayed in a normal font, e.g., θ, Γ.

- Calligraphic notation is used to differentiate variables related to policy approximation, from variables related to value function approximation. So, the policy parameter is ϑ and the policy basis functions (BFs) are φ, whereas the value function parameter is θ and the value function BFs are ϕ. Furthermore, the number of policy BFs is $\mathcal{N}$, and the number of samples for policy approximation is $\mathcal{N}_{\mathrm{s}}$.

- In the context of value function approximation, we denote BFs that only depend on the state by $\bar{\phi}$, to differentiate them from the state-action BFs ϕ, whenever these two types of BFs appear together. When there is no possibility of confusion, the simpler notation ϕ is used for both types of BFs.
- For all the examples in this thesis, the measurement units of variables are mentioned only once in the text, when the variables are introduced, after which they are omitted. Time measurements are always accompanied by units, however.

List of symbols and notations

Below, a list is given containing the mathematical symbols and notations that are used most frequently in this thesis.

General notations

$\lvert\cdot\rvert$	absolute value (for numeric arguments); cardinality (for arguments that are sets)
$\lVert\cdot\rVert_p$	p-norm of the argument
$\lfloor\cdot\rfloor$	the largest integer smaller than or equal to the argument (floor)
$\lceil\cdot\rceil$	the smallest integer larger than or equal to the argument (ceiling)
$\mathcal{F}$	set of fixed points of a mapping

Probability theory

a; $\mathcal{A}$	generic random variable; domain of generic random variable
p, q	probability densities
$a \sim p(\cdot)$	random sample a is drawn from the density p
$\mathrm{P}(\cdot)$	probability of the random variable given as argument
$\mathrm{E}\{\cdot\}$	expectation of the random variable given as argument
η; σ	mean of a Gaussian density; standard deviation of a Gaussian density
η^{bin}	parameter (success probability) of a Bernoulli density

Markov decision processes. Exact dynamic programming and reinforcement learning

x; X	state; state space
u; U	control action; action space
f; $\tilde{f}$	deterministic transition function; stochastic transition function
ρ; $\tilde{\rho}$	deterministic reward function; stochastic reward function
h; $\tilde{h}$	deterministic control policy; stochastic control policy
r; R	reward; return
γ	discount factor
k	discrete time index
T_{s}	sampling time
Q; V	Q-function (state-action value function); V-function (state value function)
Q^*; V^*	optimal Q-function; optimal V-function
Q^h; V^h	Q-function of policy h; V-function of policy h
$\mathcal{Q}$	set of all Q-functions

T	Q-iteration mapping
T^h	policy evaluation mapping
ℓ, τ	iteration indices
X_0	representative set of initial states
ε_{MC}	admissible error in the estimation of the return
K	discrete time horizon for estimating the return

Approximate dynamic programming and reinforcement learning

$\widehat{Q}$; $\widehat{V}$	approximate Q-function; approximate V-function
$\widehat{\mathcal{Q}}$	set of all Q-functions that can be represented by the considered approximator
$\widehat{h}$	approximate policy
D; d	number of dimensions of the state space; dimension index
F; P	approximation mapping; projection mapping
U_{d}	set of discrete actions
θ; ϕ	value function parameter vector; basis functions for value function approximation
ϑ; φ	policy parameter vector; basis functions for policy approximation
$\bar{\phi}$	state-dependent basis functions for Q-function approximation
n	number of: Q-function parameters, state-action basis functions
N	number of state-dependent basis functions for value function approximation
M	number of discrete actions
$\mathcal{N}$	number of: policy parameters, state-dependent policy basis functions
l	index of: Q-function parameter, state-action basis function
i	index of: policy parameter, state-dependent basis function, discrete state
j	index of discrete action
$[i,j]$	state-action index corresponding to i and j, usually $[i,j] = i + (j-1)N$
n_{s}	number of state-action samples for Q-function approximation
$\mathcal{N}_{\text{s}}$	number of state samples for policy approximation
l_{s}	index of state-action sample
i_{s}	index of state sample, as well as generic sample index
ξ; Ξ	basis functions parameter vector; set of basis function parameters
c; b	center of basis function; width of basis function
$\bar{N}$; $\bar{M}$	number of states; and number of actions in a finite Markov decision process
Γ	matrix on the left-hand side of the projected Bellman equation
z	vector on the right-hand side of the projected Bellman equation
w	weight function of the approximation errors, or of the representative states X_0
P^w	weighted Euclidean projection matrix, for the error weight function w
ε_Q	Q-function approximation error
ε_h	policy approximation error
ε_{QI}	convergence threshold of Q-iteration
$\varepsilon_{\text{LSPI}}$	convergence threshold of least-squares policy iteration

Fuzzy Q-iteration

χ; μ	fuzzy set; membership function
x_i	core of the ith fuzzy set
T_{d}	discrete-action Q-iteration mapping

S	serial fuzzy Q-iteration mapping
δ_x; δ_u	state resolution step; action resolution step
L	Lipschitz constant of a function
B	upper bound

Online and continuous-action least-squares policy iteration

K_θ	number of samples between two consecutive policy improvements
T_θ	time interval between two consecutive policy improvements
ε; ε_d	exploration probability; decay rate of exploration probability
T_trial	trial length
δ_mon	monotonicity direction (increasing or decreasing)
Δ_mon	matrix expressing the monotonicity constraints
N_mon	number of monotonicity constraints
q_d	action quantization function
ψ; M_p	polynomial; degree of polynomial approximator

The cross-entropy method. Cross-entropy policy search

s	score function
v, z	parameter vectors of a parameterized probability density
$p(\cdot; v)$	parameterized density with parameter v
$I\{\cdot\}$	indicator function, equal to 1 when the argument is true, and 0 otherwise
$\mathcal{D}$	cross-entropy, also known as Kullback-Leibler divergence
N_CE	number of samples used at every iteration
ϱ_CE	proportion of samples used in the cross-entropy updates
λ	probability level or $(1-\varrho_\mathrm{CE})$ quantile of the sample performance
c_CE	how many times more samples to use than the number of density parameters
ε_CE	convergence threshold
d_CE	number of iterations for which λ should stay roughly constant
α_CE	smoothing parameter
ϑ	in this context, assignment of discrete actions to basis functions
N_MC	number of Monte Carlo simulations for each representative state

List of abbreviations

This list below collects the abbreviations used most frequently in this thesis.

MDP	Markov decision process
DP	dynamic programming
RL	reinforcement learning
BF	basis function
RBF	radial basis function
MF	membership function
CE	cross-entropy
LSPI	least-squares policy iteration
LSTD-Q	least-squares temporal difference for Q-functions
LSPE-Q	least-squares policy evaluation for Q-functions

Summary

The framework of dynamic programming (DP) and reinforcement learning (RL) can be used to express important problems arising in a variety of fields, including e.g., automatic control, operations research, economy, and computer science. From the perspective of automatic control, DP/RL expresses an optimal control problem for general, nonlinear, and stochastic systems. Moreover, RL algorithms solve the problem without requiring prior knowledge about the process, and online RL algorithms do not even require data in advance; they start learning a solution to the problem at the moment when they are placed in the control loop.

In the DP/RL framework, a controller measures at each discrete time step the state of a process, and applies an action according to a control policy. As a result of this action, the process transits into a new state, and a scalar reward signal is sent to the controller to indicate the quality of this transition. The controller measures the new state, and the whole cycle repeats. The goal is to find an optimal policy, i.e., a policy that maximizes the cumulative reward over the course of interaction (the return). DP algorithms search for an optimal policy using a model of the process dynamics and the reward function. RL algorithms do not require a model, but use data obtained from the process. Many DP and RL algorithms use value functions, which give the returns from every state (V-functions) or from every state-action pair (Q-functions).

This thesis develops effective DP and RL techniques for control. Classical DP/RL algorithms represent value functions and policies exactly. However, the majority of the control problems have continuous state and action variables, in which case value functions and policies cannot be represented exactly, but have to be *approximated*. Three categories of algorithms for approximate DP/RL can be identified, according to the path they take to search for an (approximately) optimal policy: approximate value iteration, approximate policy iteration, and approximate policy search. Algorithms for approximate value iteration search for an approximation of the optimal value function, and use it to compute an approximately optimal policy. Algorithms for approximate policy iteration iteratively improve policies. In each iteration, an approximate value function of the current policy is found, which is then used to computed a new, improved policy. Algorithms for approximate policy search parameterize the policy and optimize its parameters directly, without using a value function.

The main contribution of this thesis is the development and study of algorithms for each of the three categories of approximate DP/RL described above. The development of each algorithm is motivated by an important open issue in its category. The goal of the algorithms developed is to find an (approximately) optimal, stationary policy that (approximately) maximizes the infinite-horizon discounted sum of rewards. All the value function and policy representations employed in this thesis rely on basis functions defined over the state space or over the state-action space. Below, the proposed algorithms are described in more detail.

Fuzzy Q-iteration

Fuzzy approximation is a promising way to represent the value function for *approximate value iteration*. It has been used often for approximate RL, but only in a heuristic fashion and without convergence guarantees. Therefore, this thesis develops *fuzzy Q-iteration*, a theoretically sound DP algorithm that represents Q-functions using a fuzzy partition of the state space, and a discretization of the action space. Each fuzzy set in the partition is described by a membership function, which can be identified with a basis function. The resulting Q-function representation is linear in the parameters, so that existing convergence guarantees for DP with linearly parameterized approximators can be used as a starting point. These results are extended to account for the action discretization, and subsequently used to show that fuzzy Q-iteration converges to a Q-function that lies within a bounded distance from the optimal Q-function. Under continuity assumptions on the dynamics and on the reward function, fuzzy Q-iteration is also consistent, i.e., it asymptotically obtains the optimal Q-function as the approximation accuracy increases. As an alternative to designing membership functions in advance, we propose a technique to optimize the membership functions using cross-entropy optimization.

An extensive numerical and experimental evaluation indicates that fuzzy Q-iteration has a good performance in practice, among others producing better policies than Q-iteration with radial basis function approximation. In a servo-system control example, increasing the number of fuzzy sets and discrete actions results in a more predictable performance improvement when a continuous reward function is used, than with a discontinuous reward function. This indicates that discontinuous reward functions, although commonly used in RL, can harm the performance in continuous-variable control tasks. For the car-on-the-hill problem, fuzzy Q-iteration gives a better performance with optimized membership functions than with equidistant membership functions.

Online and continuous-action least-squares policy iteration

Least-squares methods for policy evaluation are sample-efficient and have relaxed convergence requirements. They are used to evaluate policies in certain *approximate policy iteration* algorithms. Among these algorithms, least-squares policy iteration (LSPI) is especially attractive because it relies on Q-functions, which makes policy improvements easier than when V-functions are used. Like in fuzzy Q-iteration, the Q-functions are represented using a linear combination of basis functions. An important limitation of LSPI is that it only works offline: before every policy improvement, a batch of samples has to be processed to accurately approximate the value function of the current policy. Therefore, this thesis extends LSPI to the more challenging setting of online learning: *online LSPI* is obtained. The main difference from the offline algorithm is that the performance of each intermediate policy is important, which means that policy improvements have to be performed at short intervals. In these intervals, only a small number of samples can be processed. Unlike offline LSPI, the online algorithm has to actively explore in order to collect informative samples.

Extensive numerical experiments indicate that online LSPI is an effective algorithm for online RL. In these experiments, the algorithm converges to a performance similar to that obtained by offline LSPI. Online LSPI also exhibits a good robustness to variations in its tuning parameters. In an example involving servo-system control, for which the optimal policy is monotonous, using prior knowledge about this monotonicity property speeds up online LSPI

by a factor of two. Online LSPI is also used to learn in real time how to swing up and stabilize an inverted pendulum. Additionally, a continuous-action Q-function approximator for offline LSPI is investigated. In the inverted pendulum problem, continuous-action approximation reduces chattering, although it does not give better returns than using discrete actions.

Cross-entropy policy search

In the literature on *approximate policy search*, typically ad-hoc policy parameterizations are designed for specific problems, using intuition and prior knowledge. For instance, if a process needs to be stabilized around an equilibrium, and if the process is approximately linear in the operating range of the controller, then a policy parameterization that is linear in the state variables can be used. Unfortunately, such specific parameterizations cannot be used to solve general problems. Therefore, this thesis develops and applies highly flexible policy approximators for direct policy search, which can be used to solve a large class of problems. A discretized action space is used. The policies are represented using state-dependent basis function, where a discrete action is assigned to each basis function. The type of basis functions and their number are specified in advance, and these choices determine the approximation power of the policy representation. The locations and shapes of the basis functions, together with the action assignments, are the parameters subject to optimization — unlike in fuzzy Q-iteration and LSPI, where the basis functions are fixed. The optimization criterion is a weighted sum of the returns from a set of representative initial states, where each return is estimated with Monte Carlo simulations. The representative states together with the weight function can be used to focus the algorithm on important parts of the state space. The cross-entropy method is selected to optimize the policy parameters, leading to a *cross-entropy policy search* algorithm.

Numerical evaluations of cross-entropy policy search are conducted on a two-dimensional double integrator example, as well as on two more involved problems: bicycle balancing (with four state variables) and the simulated treatment of infection with the human immunodeficiency virus (with six state variables). Cross-entropy policy search reliably obtains good policies for every problem, requiring only a small number of basis functions. However, this performance comes at a large computational cost, e.g., several orders of magnitude higher than the cost of fuzzy Q-iteration.

The thesis closes with some concluding remarks and a discussion of important open issues regarding the three proposed approaches. Additionally, some more fundamental unsolved issues in the field of DP and RL for control are discussed, and promising research directions to address these issues are suggested.

Samenvatting

Dynamisch programmeren (DP) en *reinforcement learning* (RL) kunnen gebruikt worden om belangrijke problemen te beschrijven die in tal van gebieden voorkomen waaronder b.v. automatische regeling, operationeel onderzoek, economie en informatica. Vanuit het oogpunt van automatische regeling drukken DP en RL een optimale-regelingsprobleem uit voor algemene, niet-lineaire en stochastische systemen. Algoritmen voor het oplossen van dergelijke problemen bieden zeer beloftevolle vooruitzichten voor optimale regeling. Daarenboven lossen RL-algoritmen het probleem op zonder voorafgaande kennis over het proces te vereisen. Online RL-algoritmen hebben zelfs op voorhand geen data nodig: ze starten met het leren van een oplossing van zodra ze in de regellus geplaatst worden.

In het DP/RL-kader meet een regelaar op discrete tijdstippen de toestand van een proces en voert hij een actie uit overeenkomstig de regelstrategie. Ten gevolge van deze actie komt het proces in een nieuwe toestand terecht en wordt er een scalair beloningssignaal naar de regelaar gestuurd dat de kwaliteit van de overgang aangeeft. Vervolgens meet de regelaar de nieuwe toestand en wordt de hele cyclus herhaald. Het doel is om een optimale strategie te vinden, d.w.z. een strategie die de cumulatieve beloning (de opbrengst) over de ganse looptijd van de interactie maximaliseert. DP-algoritmen maken bij het zoeken naar een optimale strategie gebruik van een model van het proces en de beloningsfunctie. RL-algoritmen hebben geen model nodig, maar gebruiken data verkregen uit het proces. Vele DP- en RL-algoritmen werken met waardefuncties, die de opbrengst aangeven vanuit elke toestand (V-functies) of vanuit elk toestands-actiepaar (Q-functies).

In dit proefschrift worden doeltreffende DP- en RL-methoden voor regeling ontwikkeld. Klassieke DP- en RL-algoritmen werken met een exacte representatie van de waardefuncties en van de strategieën. Het grootste deel van de regelproblemen wordt echter gekenmerkt door continue toestands- en actievariabelen. In dat geval kunnen waardefuncties en strategieën niet meer exact voorgesteld worden, maar moeten ze benaderd worden. Er kunnen drie categorieën van algoritmen voor benaderend DP/RL onderscheiden worden, al naar gelang het pad dat ze nemen om een (bij benadering) optimale strategie te zoeken: benaderende waarde-iteratie, benaderende strategie-iteratie en benaderende strategie-optimalisatie. Algoritmen voor benaderende waarde-iteratie zoeken een benadering van de optimale waardefunctie en gebruiken deze dan om een bij benadering optimale strategie te berekenen. Algoritmen voor benaderende strategie-iteratie verbeteren de strategieën iteratief: in elke iteratie wordt een benaderende waardefunctie van de huidige strategie bepaald, die vervolgens gebruikt wordt om een nieuwe, verbeterde strategie te berekenen. Algoritmen voor benaderende strategie-optimalisatie parametriseren de strategie en optimaliseren de parameters van de strategie direct, zonder gebruik te maken van de waardefunctie.

De belangrijkste bijdrage van dit proefschrift is de ontwikkeling en de studie van baanbrekende algoritmen voor elk van de drie hogervermelde categorieën van benaderend DP/ RL. De ontwikkeling van elk algoritme is gedreven door een belangrijk open probleem in de desbetreffende categorie. Het doel van de ontwikkelde algoritmen is om een (bij benadering) optimale stationaire strategie te bepalen die (bij benadering) de oneindige-horizon verdisconteerde som van beloningen maximaliseert. Alle representaties van waardefuncties en strategieën die in dit proefschrift gebruikt worden, zijn gebaseerd op basisfuncties die over de toestandsruimte of over de toestands-actieruimte gedefinieerd zijn. De voorgestelde algoritmen worden nu in meer detail beschreven.

Fuzzy Q-iteratie

Fuzzy benadering is een beloftevolle manier om de waardefunctie voor benaderende waardeiteratie te representeren. Deze techniek is al vaak gebruikt voor benaderend RL, maar dan enkel op een heuristische manier en zonder convergentie-garantie. Daarom ontwikkelen wij in dit proefschrift fuzzy Q-iteratie, een theoretisch goed onderbouwd DP-algoritme dat de Q-functies representeert met behulp van een fuzzy partitie van de toestandsruimte en een discretisatie van de actieruimte. Elke fuzzy verzameling in partitie wordt beschreven door een lidmaatschapsfunctie, die kan geïdentificeerd worden met een basisfunctie. De resulterende Q-functie-representatie is lineair in de parameters zodat de bestaande convergentie-garanties voor DP met lineair geparametriseerde benaderingen als vertrekpunt gebruikt kunnen worden. Deze resultaten worden uitgebreid om de discretisatie van de acties in rekening te brengen en om vervolgens aan te tonen dat fuzzy Q-iteratie naar een Q-functie convergeert die binnen een begrensde afstand ligt van de optimale Q-functie. Onder continuïteitsveronderstellingen over de dynamica en de beloningsfunctie is fuzzy Q-iteratie ook consistent, d.w.z., naarmate de benaderingsnauwkeurigheid verhoogt, wordt asymptotisch de optimale Q-functie verkregen. Als een alternatief voor het op voorhand bepalen van de lidmaatschapsfuncties stellen wij ook een techniek voor om de lidmaatschapsfuncties te optimaliseren via *cross-entropy* optimalisatie.

Een uitgebreide numerieke en experimentele evaluatie geeft aan dat fuzzy Q-iteratie in de praktijk goed presteert, waarbij het onder andere betere strategieën oplevert dan Q-iteratie met een benadering gebaseerd op radiale basisfuncties. Voor een voorbeeld met een servosysteemregeling resulteert een groter aantal fuzzy verzamelingen en discrete acties in een meer voorspelbare verbetering van de prestatie wanneer een continue beloningsfunctie gebruikt wordt dan in het geval van een discontinue beloningsfunctie. Dit duidt erop dat hoewel discontinue beloningsfuncties vaak gebruikt worden in RL, ze een negatieve impact kunnen hebben voor regelproblemen met continue variabelen. Voor het probleem met het wagentje op de heuvel geeft fuzzy Q-iteratie een betere prestatie met geoptimaliseerde lidmaatschapsfuncties dan met equidistante lidmaatschapsfuncties.

Online en continue-actie kleinste-kwadraten strategie-iteratie

Kleinste-kwadratenmethoden voor strategie-evaluatie zijn efficiënt voor wat het aantal *samples* betreft en hebben minder strenge convergentievereisten. Ze worden gebruikt om strategieën te evalueren in bepaalde benaderende strategie-iteratie-algoritmen. Binnen de klasse van dergelijke algoritmen is kleinste-kwadraten strategie-iteratie (LSPI – *least-squares policy iteration*) bijzonder aantrekkelijk omdat het gebruik maakt van Q-functies. Dit maakt het

verbeteren van de strategie gemakkelijker dan wanneer V-functies gebruikt worden. Net zoals in fuzzy Q-iteratie worden Q-functies bij LSPI gerepresenteerd met behulp van een lineaire combinatie van basisfuncties. Een belangrijke beperking van LSPI is dat het enkel offline kan gebruikt worden: voor elke strategieverbetering moet een reeks *samples* verwerkt worden om de waardefunctie van de huidige strategie nauwkeurig te benaderen. Daarom wordt LSPI in dit proefschrift uitgebreid naar het meer uitdagende kader van online leren en op deze manier wordt online LSPI ontwikkeld. Het belangrijkste verschil met het offline algoritme is dat de prestatie van elke tussentijdse strategie belangrijk is, wat inhoudt dat de strategieverbeteringen met korte tussenpozen moeten uitgevoerd worden. In die tussenpozen kan slechts een klein aantal *samples* verwerkt worden. In tegenstelling tot offline LSPI moet het online algoritme actief exploreren om *samples* te verzamelen die voldoende informatie bevatten.

Uitvoerige numerieke experimenten geven aan dat online LSPI een effectief algoritme is voor online RL. In deze experimenten convergeert het algoritme naar een prestatie die vergelijkbaar is met die van offline LSPI. Online LSPI is ook erg robuust tegenover variaties in de instelwaarden van het algoritme. Voor een voorbeeld met een servo-systeemregeling waarvoor de optimale strategie monotoon is, leidt het gebruik van a priori kennis over de monotoniciteitseigenschap tot een versnelling van online LSPI met een factor twee. Online LSPI is ook gebruikt om in reële tijd te leren hoe een omgekeerde slinger kan opgeslingerd en gestabiliseerd worden. Daarenboven is een continue-actie Q-functie benadering voor offline LSPI onderzocht. Voor de omgekeerde slinger leidt een continue-actie-benadering tot minder *chattering*, maar levert ze geen betere opbrengst (cumulatieve beloning) op dan het gebruik van discrete acties.

Cross-entropy strategie-optimalisatie

In de literatuur over benaderende strategie-optimalisatie worden bijna altijd ad-hoc strategieparametrisaties ontworpen voor specifieke problemen op basis van intuïtie en voorkennis. Als bijvoorbeeld een proces moet gestabiliseerd worden rond een evenwichtspunt en als het proces zich min of meer lineair gedraagt in het werkingsgebied van de regelaar, dan kan een parametrisatie gebruikt worden die lineair is in de toestandvariabelen. Dergelijke specifieke parametrisaties kunnen echter niet gebruikt worden om meer algemene problemen op te lossen. Daarom worden in dit proefschrift zeer flexibele stategiebenaderingsmethoden ontwikkeld en toegepast voor directe strategie-optimalisatie, waarmee een grote klasse van problemen kan opgelost worden. Bij deze methode wordt een gediscretiseerde actieruimte gebruikt en worden de strategieën gerepresenteerd met behulp van toestandsafhankelijke basisfuncties, waarbij een discrete actie toegekend wordt aan elke basisfunctie. Het type basisfunctie en het aantal basisfuncties worden op voorhand vastgelegd. Deze keuzes bepalen de benaderingskracht van de strategierepresentatie. In tegenstelling tot fuzzy Q-iteratie en LSPI, waar de basisfuncties volledig vastgelegd worden en blijven, zijn de optimalisatievariabelen nu de posities en de vorm van de basisfuncties, alsmede de actietoekenningen. Het optimalisatiecriterium is een gewogen som van de opbrengsten voor een verzameling representatieve begintoestanden, waarbij de opbrengsten geschat worden met behulp van Monte-Carlo-simulaties. De representatieve begintoestanden en de gewichtsfunctie kunnen gebruikt worden om het algoritme te laten focussen op belangrijke gebieden in de toestandsruimte. De *cross-entropy* methode wordt gekozen om de strategieparameters te optimaliseren, wat resulteert in een *cross-entropy* strategie-optimalisatie-algoritme.

Numerieke evaluaties van de *cross-entropy* strategie-optimalisatie worden uitgevoerd op

een voorbeeld met een 2-dimensionele dubbele integrator en op twee meer complexe problemen: het balanceren van een fiets (4 toestandsvariabelen) en de gesimuleerde behandeling van een infectie met het HIV-virus (6 toestandsvariabelen). *Cross-entropy* strategie-optimalisatie geeft op een betrouwbare wijze goede resultaten voor elk van deze problemen en dit gebruik makend van een beperkt aantal basisfuncties. Deze prestatie is echter verbonden met een zware rekenkost die b.v. enkele ordes van grootte hoger is dan de rekenkost van fuzzy Q-iteratie.

Het proefschrift eindigt met een aantal afsluitende opmerkingen en een discussie van belangrijke open problemen in verband met de drie voorgestelde methoden. Daarnaast worden ook een aantal meer fundamentele onopgeloste problemen op het gebied van DP en RL bediscussieerd en worden een aantal veelbelovende richtingen besproken om deze problemen aan te pakken.

Curriculum vitae

Lucian Buşoniu was born in 1979 in Petroşani, Romania. In 1998, he graduated from the Computer Science High School of the same city. He then studied Control Engineering at the Technical University of Cluj-Napoca, Romania, where he obtained his Master of Science (in its equivalent form of a 5-year Engineer's Diploma) in 2003, with the best grades of his year. His thesis was entitled "Multi-Agent Systems in Distributed Built-In Self-Testing and Distributed Testing". He then enrolled in a postgraduate program on Automatic Control at the same University, from which he graduated in 2004. During his MSc and postgraduate research work, he was advised by Prof.dr. Liviu Miclea.

Since 2005, Lucian Buşoniu has been working on his PhD project at the Center for Systems and Control of the Delft University of Technology, the Netherlands. His PhD research has dealt mainly with approximate reinforcement learning and dynamic programming for continuous-variable problems, and has been performed under the supervision of Prof.dr. Robert Babuška, M.Sc. and Prof.dr.ir. Bart De Schutter. He collaborated in his work with dr. Damien Ernst, whom he visited in 2007 at Supélec in Rennes, France. During his PhD project, Lucian Buşoniu obtained the DISC certificate for fulfilling the course program requirements of the Dutch Institute for Systems and Control, and advised a number of MSc and BSc students.

Lucian Buşoniu's research interests include learning techniques for control problems, with a focus on reinforcement learning and its model-based counterpart, dynamic programming; and multi-agent and distributed learning.