Contents

3

Dynamic programming and reinforcement learning in large and continuous spaces

This chapter describes dynamic programming and reinforcement learning for large and continuous-space problems. In such problems, exact solutions cannot be found in general, and approximation is necessary. The algorithms of the previous chapter can therefore no longer be applied in their original form. Instead, approximate versions of value iteration, policy iteration, and policy search are introduced. Theoretical guarantees are provided on the performance of the algorithms, and numerical examples are used to illustrate their behavior. Techniques to automatically find value function approximators are reviewed, and the three categories of algorithms are compared.

3.1 Introduction

The classical dynamic programming (DP) and reinforcement learning (RL) algorithms introduced in Chapter 2 require exact representations of the value functions and policies. In general, an exact value function representation can only be achieved by storing distinct estimates of the return for every state-action pair (when Q-functions are used) or for every state (in the case of V-functions). Similarly, to represent policies exactly, distinct actions have to be stored for every state. When some of the variables have a very large or infinite number of possible values (e.g., when they are continuous), such exact representations are no longer possible, and value functions and policies need to be represented approximately. Since most problems of practical interest have large or continuous state and action spaces, approximation is essential in DP and RL.

Approximators can be separated into two main types: parametric and nonparametric. *Parametric* approximators are mappings from a parameter space into the space of functions they aim to represent. The form of the mapping and the number of parameters are given *a priori*, while the parameters themselves are tuned using data about the target function. A representative example is a weighted linear combination of a fixed set of basis functions, in which the weights are the parameters. In contrast, the structure of a *nonparametric* approximator is derived from the data. Despite its name, a nonparametric approximator typically still has parameters, but unlike in the parametric case, the number of parameters (as well as their values) is determined from the data. For instance, kernel-based approximators considered in this book define one kernel per data point, and represent the target function as a weighted linear combination of these kernels, where again the weights are the parameters.

This chapter provides an extensive, in-depth review of approximate DP and RL in large and continuous-space problems. The three basic classes of DP and RL algorithms discussed in Chapter 2, namely value iteration, policy iteration, and policy search, are all extended to use approximation, resulting in *approximate value iteration*, *approximate policy iteration*, and *approximate policy search*. Algorithm derivations are complemented by theoretical guarantees on their performance, by numerical examples illustrating their behavior, and by comparisons of the different approaches. Several other important topics in value function and policy approximation are also treated. To help in navigating this large body of material, Figure 3.1 presents a road map of the chapter in graphical form, and the remainder of this section details this road map.

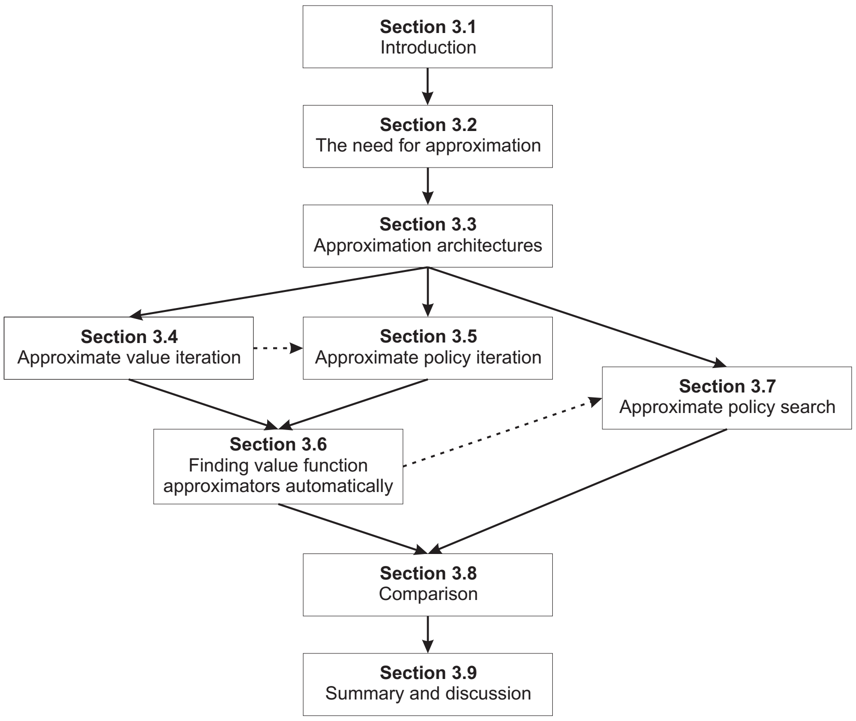

Section 3.1
Introduction
Section 3.2
The need for approximation
Section 3.3
Approximation architectures
Section 3.4
Approximate value iteration
Section 3.5
Approximate policy iteration
Section 3.7
Approximate policy search
Section 3.6
Finding value function approximators automatically
Section 3.8
Comparison
Section 3.9
Summary and discussion

FIGURE 3.1
A road map of this chapter. The arrows indicate the recommended sequence of reading. Dashed arrows indicate optional ordering.

In Section 3.2, the need for approximation in DP and RL for large and continuous spaces is explained. Approximation is not only a problem of compact representation, but also plays a role in several other parts of DP and RL algorithms. In Section 3.3, parametric and nonparametric approximation architectures are introduced and compared.

This introduction is followed by an in-depth discussion of approximate value iteration in Section 3.4, and of approximate policy iteration in Section 3.5. Techniques to automatically derive value function approximators, useful in approximate value iteration and policy iteration, are reviewed right after these two classes of algorithms, in Section 3.6. Approximate policy search is discussed in detail in Section 3.7. Representative algorithms from each of the three classes are applied to a numerical example involving the optimal control of a DC motor.

In closing the chapter, approximate value iteration, policy iteration, and policy search are compared in Section 3.8, while Section 3.9 provides a summary and discussion.

In order to reasonably restrict the scope of this chapter, several choices are made regarding the material that will be presented:

- In the context of value function approximation, we focus on Q-function approximation and Q-function based algorithms, because a significant portion of the remainder of this book concerns such algorithms. Nevertheless, a majority of the concepts and algorithms introduced extend in a straightforward manner to the case of V-function approximation.

- We mainly consider parametric approximation, because the remainder of the book relies on this type of approximation, but we also review nonparametric approaches to approximate value iteration and policy iteration.

- When discussing parametric approximation, whenever appropriate, we consider general (possibly nonlinear) parametrizations. Sometimes, however, we consider linear parametrizations in more detail, e.g., because they allow the derivation of better theoretical guarantees on the resulting approximate solutions.

Next, we give some additional details about the organization of the core material of this chapter, which consists of approximate value iteration (Section 3.4), approximate policy iteration (Section 3.5), and approximate policy search (Section 3.7). To this end, Figure 3.2 shows how the algorithms selected for presentation are organized, using a graphical tree format. This organization will be explained below. All the terminal (right-most) nodes in the trees correspond to subsections in Sections 3.4, 3.5, and 3.7. Note that Figure 3.2 does not contain an exhaustive taxonomy of all the approaches.

Within the context of approximate value iteration, algorithms employing parametric approximation are presented first, separating model-based from model-free approaches. Then, value iteration with nonparametric approximation is reviewed.

Approximate policy iteration consists of two distinct problems: approximate policy evaluation, i.e., finding an approximate value function for a given policy, and policy improvement. Out of these two problems, approximate policy evaluation poses more interesting theoretical questions, because, like approximate value iteration, it

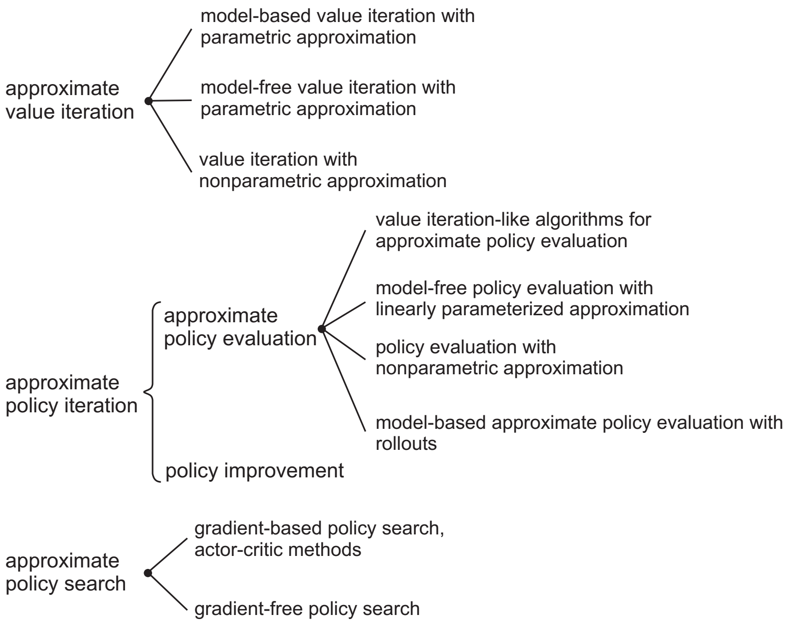

approximate value iteration
model-based value iteration with parametric approximation
model-free value iteration with parametric approximation
value iteration with nonparametric approximation
approximate policy iteration
approximate policy evaluation
value iteration-like algorithms for approximate policy evaluation
model-free policy evaluation with linearly parameterized approximation
policy evaluation with nonparametric approximation
model-based approximate policy evaluation with rollouts
policy improvement
approximate policy search
gradient-based policy search, actor-critic methods
gradient-free policy search

FIGURE 3.2
The organization of the algorithms for approximate value iteration, policy iteration, and policy search presented in this chapter.

involves finding an approximate solution to a Bellman equation. Special requirements have to be imposed to ensure that a meaningful approximate solution exists and can be found by appropriate algorithms. In contrast, policy improvement relies on solving maximization problems over the action variables, which involve fewer technical difficulties (although they may still be hard to solve when the action space is large). Therefore, we pay special attention to approximate policy evaluation in our presentation. We first describe a class of algorithms for policy evaluation that are derived along the same lines as approximate value iteration. Then, we introduce model-free policy evaluation with linearly parameterized approximation, and briefly review nonparametric approaches to approximate policy evaluation. Additionally, a model-based, direct simulation approach for policy evaluation is discussed that employs Monte Carlo estimates called “rollouts.”

From the class of approximate policy search methods (Section 3.7), gradient-based and gradient-free methods for policy optimization are discussed in turn. In the context of gradient-based methods, special attention is paid to the important category of actor-critic techniques.

3.2 The need for approximation in large and continuous spaces

The algorithms for exact value iteration (Section 2.3) require the storage of distinct return estimates for every state (if V-functions are used) or for every state-action pair (in the case of Q-functions). When some of the state variables have a very large or infinite number of possible values (e.g., when they are continuous), exact storage is no longer possible, and the value functions must be represented approximately. Large or continuous action spaces make the representation of Q-functions additionally challenging. In policy iteration (Section 2.4), value functions and sometimes policies also need to be represented approximately in general. Similarly, in policy search (Section 2.5), policies must be represented approximately when the state space is large or continuous.

Approximation in DP/RL is not only a problem of representation. Two additional types of approximation are needed. First, sample-based approximation is necessary in any DP/RL algorithm. Second, value iteration and policy iteration must repeatedly solve potentially difficult nonconcave maximization problems over the action variables, whereas policy search must find optimal policy parameters, which involves similar difficulties. In general, these optimization problems can only be solved approximately. These two types of approximation are detailed below.

Sample-based approximation is required for two distinct purposes in value function estimation. Consider first, as an example, the Q-iteration algorithm for deterministic problems, namely Algorithm 2.1. Every iteration of this algorithm would have to be implemented as follows:

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

When the state-action space contains an infinite number of elements, it is impossible to loop over all the state-action pairs in finite time. Instead, a sample-based, approximate update has to be used that only considers a finite number of state-action samples.

Such sample-based updates are also necessary in stochastic problems. Moreover, in the stochastic case, sample-based approximation is required for a second, distinct purpose. Consider, e.g., the Q-iteration algorithm for general stochastic problems, which for every state-action pair (x,u) considered would have to be implemented as follows:

$$Q_{\ell+1}(x,u) = \mathrm{E}_{x' \sim \tilde{f}(x,u,\cdot)} \left\{ \tilde{\rho}(x,u,x') + \gamma \max_{u'} Q_\ell(x',u') \right\} \tag{3.2}$$

Clearly, the expectation on the right-hand side of (3.2) cannot be computed exactly in general, and must be estimated from a finite number of samples, e.g., by using Monte Carlo methods. Note that, in many RL algorithms, the estimation of the expectation does not appear explicitly, but is performed implicitly while processing samples. For instance, Q-learning (Algorithm 2.3) is such an algorithm, in which stochastic approximation is employed to estimate the expectation.

The maximization over the action variable in (3.1) or (3.2) (as well as in other value iteration algorithms) has to be solved for every sample considered. In large or continuous action spaces, this maximization is a potentially difficult nonconcave optimization problem, which can only be solved approximately in general. To simplify this problem, many algorithms discretize the action space in a small number of values, compute the value function for all the discrete actions, and find the maximum among these values using enumeration.

In policy iteration, sample-based approximation is required at the policy evaluation step, for reasons similar to those explained above. The maximization issues affect the policy improvement step, which in the case of Q-functions computes a policy $h_{\ell+1}$ using (2.34), repeated here for easy reference:

$$h_{\ell+1}(x) \in \arg\max_{u} Q^{h_\ell}(x,u)$$

Note that these sampling and maximization issues also affect algorithms that employ V-functions.

In policy search, some methods (e.g., actor-critic algorithms) estimate value functions and are therefore affected by the sampling issues mentioned above. Even methods that do not employ value functions must estimate returns in order to evaluate the policies, and return estimation requires sample-based approximation, as described next. In principle, a policy that maximizes the return from every initial state should be found. However, the return can only be estimated for a finite subset of initial states (samples) from the possibly infinite state space. Additionally, in stochastic problems, for every initial state considered, the expected return (2.15) must be evaluated using a finite set of sampled trajectories, e.g., by using Monte Carlo methods.

Besides these sampling problems, policy search methods must of course find the best policy within the class of policies considered. This is a difficult optimization problem, which can only be solved approximately in general. However, it only needs to be solved once, unlike the maximization over actions in value iteration and policy iteration, which needs to be solved for every sample considered. In this sense, policy search methods are less affected from the maximization difficulties than value iteration or policy iteration.

A different view on the benefits of approximation can be taken in the model-free, RL setting. Consider a value iteration algorithm that estimates Q-functions, such as Q-learning (Algorithm 2.3). Without approximation, the Q-value of every state-action pair must be estimated separately (assuming it is possible to do so). If little or no data is available for some states, 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 other, usually nearby, states (this requires the assumption of a certain degree of smoothness for the Q-function). Then, if 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.

3.3 Approximation architectures

Two major classes of approximators can be identified, namely parametric and nonparametric approximators. We introduce parametric approximators in Section 3.3.1, nonparametric approximators in Section 3.3.2, and compare the two classes in Section 3.3.3. Section 3.3.4 contains some additional remarks.

3.3.1 Parametric approximation

Parametric approximators are mappings from a parameter space into the space of functions they aim to represent (in DP/RL, value functions or policies). The functional form of the mapping and the number of parameters are typically established in advance and do not depend on the data. The parameters of the approximator are tuned using data about the target function.

Consider a Q-function approximator parameterized by an n-dimensional vector[1] θ. The approximator is denoted by an *approximation mapping* $F : \mathbb{R}^n \to \mathscr{Q}$, where $\mathbb{R}^n$ is the parameter space and $\mathscr{Q}$ is the space of Q-functions. Every parameter vector θ provides a compact representation of a corresponding approximate Q-function:

$$\widehat{Q} = F(\theta)$$

or equivalently, element-wise:

$$\widehat{Q}(x,u) = [F(\theta)](x,u)$$

where $[F(\theta)](x,u)$ denotes the Q-function $F(\theta)$ evaluated at the state-action pair (x,u). So, instead of storing distinct Q-values for every pair (x,u), which would be impractical in many cases, it is only necessary to store n parameters. When the state-action space is discrete, n is usually much smaller than $|X| \cdot |U|$, thereby providing a compact representation (recall that, when applied to sets, the notation $|\cdot|$ stands for cardinality). However, since the set of Q-functions representable by F is only a subset of $\mathscr{Q}$, an arbitrary Q-function can generally only be represented up to a certain approximation error, which must be accounted for.

In general, the mapping F can be nonlinear in the parameters. A typical example of a nonlinearly parameterized approximator is a feed-forward neural network (Hassoun, 1995; Bertsekas and Tsitsiklis, 1996, Chapter 3). However, linearly parameterized approximators are often preferred in DP and RL, because they make it easier to analyze the theoretical properties of the resulting DP/RL algorithms. A linearly parameterized Q-function approximator employs n basis functions (BFs) $\phi_1, \dots, \phi_n : X \times U \to \mathbb{R}$ and an n-dimensional parameter vector θ. Approximate Q-values are computed with:

$$[F(\theta)](x,u) = \sum_{l=1}^{n} \phi_l(x,u)\theta_l = \phi^{\mathrm{T}}(x,u)\theta \tag{3.3}$$

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

where $\phi(x,u) = [\phi_1(x,u), \dots, \phi_n(x,u)]^{\mathrm{T}}$ is the vector of BFs. In the literature, the BFs are also called features (Bertsekas and Tsitsiklis, 1996).

Example 3.1 Approximating Q-functions with state-dependent BFs and discrete actions. As explained in Section 3.2, in order to simplify the maximization over actions, in many DP/RL algorithms the action space is discretized into a small number of values. In this example we consider such a discrete-action approximator, which additionally employs state-dependent BFs to approximate over the state space.

A discrete, finite set of actions $u_1, \dots, u_M$ is chosen from the original action space U. The resulting discretized action space is denoted by $U_{\mathrm{d}} = \{u_1, \dots, u_M\}$. A number of N state-dependent BFs $\bar{\phi}_1, \dots, \bar{\phi}_N : X \to \mathbb{R}$ are defined and replicated for each discrete action in U_{d}. Approximate Q-values can be computed for any state-discrete action pair with:

$$[F(\theta)](x,u_j) = \phi^{\mathrm{T}}(x,u_j)\,\theta, \tag{3.4}$$

where, in the state-action BF vector $\phi^{\mathrm{T}}(x,u_j)$, all the BFs that do not correspond to the current discrete action are taken to be equal to 0:

$$\phi(x,u_j) = [\underbrace{0,\dots,0}_{u_1},\dots,0,\underbrace{\bar{\phi}_1(x),\dots,\bar{\phi}_N(x)}_{u_j},0,\dots,\underbrace{0,\dots,0}_{u_M}]^{\mathrm{T}} \in \mathbb{R}^{NM} \tag{3.5}$$

The parameter vector θ therefore has NM elements. This type of approximator can be seen as representing M distinct state-dependent slices through the Q-function, one slice for each of the M discrete actions. Note that it is only meaningful to use such an approximator for the discrete actions in U_{d}; for any other actions, the approximator outputs 0. For this reason, only the discrete actions are considered in (3.4) and (3.5).

In this book, we will often use such discrete-action approximators. For instance, consider normalized (elliptical) Gaussian radial basis functions (RBFs). This type of RBF can be defined as follows:

$$\bar{\phi}_i(x) = \frac{\phi_i'(x)}{\sum_{i'=1}^{N} \phi_{i'}'(x)}, \quad \phi_i'(x) = \exp\left(-\frac{1}{2}[x-c_i]^{\mathrm{T}} B_i^{-1} [x-c_i]\right) \tag{3.6}$$

Here, ϕ_i' are the nonnormalized RBFs, the vector $c_i = [c_{i,1}, \dots, c_{i,D}]^{\mathrm{T}} \in \mathbb{R}^D$ is the center of the ith RBF, and the symmetric positive-definite matrix $B_i \in \mathbb{R}^{D \times D}$ is its width. Depending on the structure of the width matrix, RBFs of various shapes can be obtained. For a general width matrix, the RBFs are elliptical, while axis-aligned RBFs are obtained if the width matrix is diagonal, i.e., if $B_i = \mathrm{diag}(b_{i,1}, \dots, b_{i,D})$. In this case, the width of an RBF can also be expressed using a vector $b_i = [b_{i,1}, \dots, b_{i,D}]^{\mathrm{T}}$. Furthermore, spherical RBFs are obtained if, in addition, $b_{i,1} = \cdots = b_{i,D}$.

Another class of discrete-action approximators uses *state aggregation* (Bertsekas and Tsitsiklis, 1996, Section 6.7). For state aggregation, the state space is partitioned into N disjoint subsets. Let X_i be the ith subset in this partition, for $i = 1, \dots, N$. For a given action, the approximator assigns the same Q-values for all the states in X_i. This corresponds to a BF vector of the form (3.5), with binary-valued (0 or 1) state-dependent BFs:

$$\bar{\phi}_i(x) = \begin{cases} 1 & \text{if } x \in X_i \\ 0 & \text{otherwise} \end{cases} \tag{3.7}$$

Because the subsets X_i are disjoint, exactly one BF is active at any point in the state space. All the individual states belonging to X_i can thus be seen as a single, larger *aggregate* (or quantized) state; hence the name "state aggregation" (or state quantization). By additionally identifying each subset X_i with a prototype state $x_i \in X_i$, state aggregation can also be seen as *state discretization*, where the discretized state space is $X_\mathrm{d} = \{x_1, \dots, x_N\}$. The prototype state can be, e.g., the geometrical center of X_i (assuming this center belongs to X_i), or some other representative state.

Using the definition (3.7) of the state-dependent BFs and the expression (3.5) for the state-action BFs, the state-action BFs can be written compactly as follows:

$$\phi_{[i,j]}(x,u) = \begin{cases} 1 & \text{if } x \in X_i \text{ and } u = u_j \\ 0 & \text{otherwise} \end{cases} \tag{3.8}$$

The notation $[i, j]$ represents the scalar index corresponding to i and j, which can be computed as $[i, j] = i + (j-1)N$. If the n elements of the BF vector were arranged into an $N \times M$ matrix, by first filling in the first column with the first N elements, then the second column with the subsequent N elements, etc., then the element at index $[i, j]$ of the vector would be placed at row i and column j of the matrix. Note that exactly one state-action BF (3.8) is active at any point of $X \times U_\mathrm{d}$, and no BF is active if $u \notin U_\mathrm{d}$. □

Other types of linearly parameterized approximators used in the literature include tile coding (Watkins, 1989; Sherstov and Stone, 2005), multilinear interpolation (Davies, 1997), and Kuhn triangulation (Munos and Moore, 2002).

3.3.2 Nonparametric approximation

Nonparametric approximators, despite their name, still have parameters. However, unlike in the parametric case, the number of parameters, as well as the form of the nonparametric approximator, are derived from the available data.

Kernel-based approximators are typical representatives of the nonparametric class. Consider a kernel-based approximator of the Q-function. In this case, the *kernel function* is a function defined over two state-action pairs, $\kappa : X \times U \times X \times U \to \mathbb{R}$:

$$(x, u, x', u') \mapsto \kappa((x,u),(x',u')) \tag{3.9}$$

that must also satisfy certain additional conditions (see, e.g., Smola and Schölkopf, 2004). Under these conditions, the function κ can be interpreted as an inner product between feature vectors of its two arguments (the two state-action pairs) in a high-dimensional feature space. Using this property, a powerful approximator can be obtained by only computing the kernels, without ever working explicitly in the feature space. Note that in (3.9), as well as in the sequel, the state-action pairs are grouped together for clarity.

A widely used type of kernel is the Gaussian kernel, which for the problem of approximating the Q-function is given by:

$$\kappa((x,u),(x',u')) = \exp\left(-\frac{1}{2}\begin{bmatrix}x-x'\\u-u'\end{bmatrix}^{\mathrm{T}} B^{-1}\begin{bmatrix}x-x'\\u-u'\end{bmatrix}\right) \tag{3.10}$$

where the kernel width matrix $B \in \mathbb{R}^{(D+C)\times(D+C)}$ must be symmetric and positive definite. Here, D denotes the number of state variables and C denotes the number of action variables. For instance, a diagonal matrix $B = \operatorname{diag}(b_1,\dots,b_{D+C})$ can be used. Note that, when the pair (x',u') is fixed, the kernel (3.10) has the same shape as a Gaussian state-action RBF centered on (x',u').

Assume that a set of state-action samples is available: $\{(x_{l_\mathrm{s}},u_{l_\mathrm{s}}) \mid l_\mathrm{s} = 1,\dots,n_\mathrm{s}\}$. For this set of samples, the kernel-based approximator takes the form:

$$\widehat{Q}(x,u) = \sum_{l_\mathrm{s}=1}^{n_\mathrm{s}} \kappa((x,u),(x_{l_\mathrm{s}},u_{l_\mathrm{s}}))\theta_{l_\mathrm{s}} \tag{3.11}$$

where $\theta_1,\dots,\theta_{n_\mathrm{s}}$ are the parameters. This form is superficially similar to the linearly parameterized approximator (3.3). However, there is a crucial difference between these two approximators. In the parametric case, the number and form of the BFs were defined in advance, and therefore led to a fixed functional form F of the approximator. In contrast, in the nonparametric case, the number of kernels and their form, and thus also the number of parameters and the functional form of the approximator, are determined from the samples.

One situation in which the kernel-based approximator can be seen as a parametric approximator is when the set of samples is selected in advance. Then, the resulting kernels can be identified with predefined BFs:

$$\phi_{l_\mathrm{s}}(x,u) = \kappa((x,u),(x_{l_\mathrm{s}},u_{l_\mathrm{s}})), \quad l_\mathrm{s} = 1,\dots,n_\mathrm{s}$$

and the kernel-based approximator (3.11) is equivalent to a linearly parameterized approximator (3.3). However, in many cases, such as in online RL, the samples are not available in advance.

Important classes of nonparametric approximators that have been used in DP and RL include kernel-based methods (Shawe-Taylor and Cristianini, 2004), among which support vector machines are the most popular (Schölkopf et al., 1999; Cristianini and Shawe-Taylor, 2000; Smola and Schölkopf, 2004), Gaussian processes, which also employ kernels (Rasmussen and Williams, 2006), and regression trees (Breiman et al., 1984; Breiman, 2001). For instance, kernel-based and related approximators have been applied to value iteration (Ormoneit and Sen, 2002; Deisenroth et al., 2009; Farahmand et al., 2009a) and to policy evaluation and policy iteration (Lagoudakis and Parr, 2003b; Engel et al., 2003, 2005; Xu et al., 2007; Jung and Polani, 2007a; Bethke et al., 2008; Farahmand et al., 2009b). Ensembles of regression trees have been used with value iteration by Ernst et al. (2005, 2006a) and with policy iteration by Jodogne et al. (2006).

Note that nonparametric approximators are themselves driven by certain meta-parameters, such as the width B of the Gaussian kernel (3.10). These meta-parameters influence the accuracy of the approximator and may require tuning, which can be difficult to perform manually. However, there also exist methods for automating the tuning process (Deisenroth et al., 2009; Jung and Stone, 2009).

3.3.3 Comparison of parametric and nonparametric approximation

Because they are designed in advance, parametric approximators have to be flexible enough to accurately model the target functions solely by tuning the parameters. Highly flexible, nonlinearly parameterized approximators are available, such as neural networks. However, when used to approximate value functions in DP and RL, general nonlinear approximators make it difficult to guarantee the convergence of the resulting algorithms, and indeed can sometimes lead to divergence. Often, linearly parameterized approximators (3.3) must be used to guarantee convergence. Such approximators are specified by their BFs. When prior knowledge is not available to guide the selection of BFs (as is usually the case), a large number of BFs must be defined to evenly cover the state-action space. This is impractical in high-dimensional problems. To address this issue, methods have been proposed to automatically derive a small number of good BFs from data. We review these methods in Section 3.6. Because they derive BFs from data, such methods can be seen as residing between parametric and nonparametric approximation.

Nonparametric approximators are highly flexible. However, because their shape depends on the data, it may change while the DP/RL algorithm is running, which makes it difficult to provide convergence guarantees. A nonparametric approximator adapts its complexity to the amount of available data. This is beneficial in situations where data is costly or difficult to obtain. It can, however, become a disadvantage when a large amount of data is used, because the computational and memory demands of the approximator usually grow with the number of samples. For instance, the kernel-based approximator (3.11) has a number of parameters equal to the number of samples n_{s} used. This is especially problematic in online RL algorithms, which keep receiving new samples for their entire lifetime. There exist approaches to mitigate this problem. For instance, in kernel-based methods, the number of samples used to derive the approximator can be limited by only employing a subset of samples that contribute significantly to the accuracy of the approximation, and discarding the rest. Various measures can be used for the contribution of a given sample to the approximation accuracy. Such kernel sparsification methods were employed by Xu et al. (2007); Engel et al. (2003, 2005), and a related, so-called subset of regressors method was applied by Jung and Polani (2007a). Ernst (2005) proposed the selection of informative samples for an offline RL algorithm by iteratively choosing those samples for which the error in the Bellman equation is maximal under the current value function.

3.3.4 Remarks

The approximation architectures introduced above for Q-functions can be extended in a straightforward manner to V-function and policy approximation. For instance, a linearly parameterized policy approximator can be described as follows. A set of state-dependent BFs $\varphi_1, \dots, \varphi_{\mathcal{N}} : X \to \mathbb{R}$ are defined, and given a parameter vector $\vartheta \in \mathbb{R}^{\mathcal{N}}$, the approximate policy is:

$$\widehat{h}(x) = \sum_{i=1}^{\mathcal{N}} \varphi_i(x) \vartheta_i = \varphi^{\mathrm{T}}(x) \vartheta \tag{3.12}$$

where $\varphi(x) = [\varphi_1(x), \dots, \varphi_{\mathcal{N}}(x)]^{\mathrm{T}}$. For simplicity, the parametrization (3.12) is only given for scalar actions, but it can easily be extended to the case of multiple action variables. Note that we use calligraphic notation 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 denoted by φ, whereas the value function parameter is θ and the value function BFs are denoted by ϕ. Furthermore, the number of policy parameters and BFs is $\mathcal{N}$. When samples are used to approximate the policy, their number is denoted by $\mathcal{N}_s$.

In the parametric case, whenever we wish to explicitly highlight the dependence of an approximate policy $\widehat{h}$ on the parameter vector ϑ, we will use the notation $\widehat{h}(x; \vartheta)$. Similarly, when the dependence of a value function on the parameters needs to be made explicit without using the mapping F, we will use $\widehat{Q}(x, u; \theta)$ and $\widehat{V}(x; \theta)$ to denote Q-functions and V-functions, respectively.

Throughout the remainder of this chapter, we will mainly focus on DP and RL with parametric approximation, because the remainder of the book relies on this type of approximation, but we will also overview nonparametric approaches to value iteration and policy iteration.

3.4 Approximate value iteration

In order to apply value iteration to large or continuous-space problems, the value function must be approximated. Figure 3.3 (repeated from the relevant part of Figure 3.2) shows how our presentation of the algorithms for approximate value iteration is organized. First, we describe value iteration with parametric approximation in detail. Specifically, in Section 3.4.1 we present model-based algorithms from this class, and in Section 3.4.2 we describe offline and online model-free algorithms. Then, in Section 3.4.3, we briefly review value iteration with nonparametric approximation.

Having completed our review of the algorithms for approximate value iteration, we then provide convergence guarantees for these algorithms, in Section 3.4.4. Finally, in Section 3.4.5, we apply two representative algorithms for approximate value iteration to a DC motor control problem.

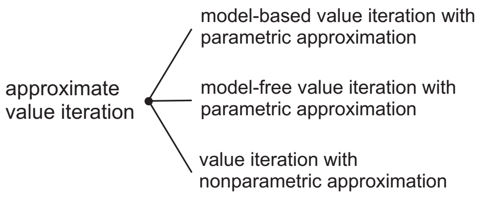

approximate value iteration
model-based value iteration with parametric approximation
model-free value iteration with parametric approximation
value iteration with nonparametric approximation

FIGURE 3.3
The organization of the algorithms for approximate value iteration presented in this section.

3.4.1 Model-based value iteration with parametric approximation

This section considers Q-iteration with a general parametric approximator, which is a representative model-based algorithm for approximate value iteration.

Approximate Q-iteration is an extension of the exact Q-iteration algorithm introduced in Section 2.3.1. Recall that exact Q-iteration starts from an arbitrary Q-function Q_0 and at each iteration ℓ updates the Q-function using the rule (2.25), repeated here for easy reference:

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

where T is the Q-iteration mapping (2.22) or (2.23). In approximate Q-iteration, the Q-function Q_ℓ cannot be represented exactly. Instead, an approximate version is compactly represented by a parameter vector $\theta_\ell \in \mathbb{R}^n$, using a suitable approximation mapping $F : \mathbb{R}^n \to \mathscr{Q}$ (see Section 3.3):

$$\widehat{Q}_\ell = F(\theta_\ell)$$

This approximate Q-function is provided, instead of Q_ℓ, as an input to the Q-iteration mapping T. So, the Q-iteration update would become:

$$Q^{\ddagger}_{\ell+1} = (T \circ F)(\theta_\ell) \tag{3.13}$$

However, in general, the newly found Q-function $Q^{\ddagger}_{\ell+1}$ cannot be explicitly stored, either. Instead, it must also be represented approximately, using a new parameter vector $\theta_{\ell+1}$. This parameter vector is obtained by a *projection mapping* $P : \mathscr{Q} \to \mathbb{R}^n$:

$$\theta_{\ell+1} = P(Q^{\ddagger}_{\ell+1})$$

which ensures that $\widehat{Q}_{\ell+1} = F(\theta_{\ell+1})$ is as close as possible to $Q^{\ddagger}_{\ell+1}$. A natural choice for P is least-squares regression, which, given a Q-function Q, produces:[2]

$$P(Q) = \theta^{\ddagger}, \text{ where } \theta^{\ddagger} \in \arg\min_{\theta} \sum_{l_s=1}^{n_s} \left(Q(x_{l_s}, u_{l_s}) - [F(\theta)](x_{l_s}, u_{l_s})\right)^2 \tag{3.14}$$

[2]In the absence of additional restrictions, the use of least-squares projections can cause convergence problems, as we will discuss in Section 3.4.4.

for some set of state-action samples $\{(x_{l_s}, u_{l_s}) \,|\, l_s = 1, \dots, n_s\}$. Some care is required to ensure that $\theta^\ddagger$ exists and that it is not too difficult to find. For instance, when the approximator F is linearly parameterized, (3.14) is a convex quadratic optimization problem.

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

$$\theta_{\ell+1} = (P \circ T \circ F)(\theta_\ell) \tag{3.15}$$

Of course, in practice, the intermediate results of F and T cannot be fully computed and stored. Instead, $P \circ T \circ F$ can be implemented as a single mapping, or T and F can be sampled at a finite number of points. The algorithm is stopped once a satisfactory parameter vector $\widehat{\theta}^*$ has been found (see below for examples of stopping criteria). Ideally, $\widehat{\theta}^*$ is near to a fixed point θ^* of $P \circ T \circ F$. In Section 3.4.4, we will give conditions under which a unique fixed point exists and is obtained asymptotically as $\ell \to \infty$.

Given $\widehat{\theta}^*$, a greedy policy in $F(\widehat{\theta}^*)$ can be found, i.e., a policy h that satisfies:

$$h(x) \in \arg\max_u [F(\widehat{\theta}^*)](x,u) \tag{3.16}$$

Here, as well as in the sequel, we assume that the Q-function approximator is structured in a way that guarantees the existence of at least one maximizing action for any state. Because the approximator is under the control of the designer, ensuring this property should not be too difficult.

Figure 3.4 illustrates approximate Q-iteration and the relations between the various mappings, parameter vectors, and Q-functions considered by the algorithm.

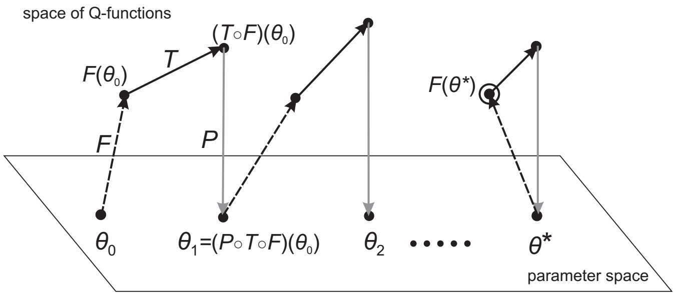


FIGURE 3.4
A conceptual illustration of approximate Q-iteration. At every iteration, the approximation mapping F is applied to the current parameter vector to obtain an approximate Q-function, which is then passed through the Q-iteration mapping T. The result of T is then projected back onto the parameter space with the projection mapping P. Ideally, the algorithm asymptotically converges to a fixed point θ^*, which leads back to itself when passed through $P \circ T \circ F$. The asymptotically obtained solution of approximate Q-iteration is then the Q-function $F(\theta^*)$.

Algorithm 3.1 presents an example of approximate Q-iteration for a deterministic Markov decision process (MDP), using the least-squares projection (3.14). At line 4 of this algorithm, $Q^{\ddagger}_{\ell+1}(x_{l_s}, u_{l_s})$ has been computed according to (3.13), in which the definition (2.22) of the Q-iteration mapping has been substituted.

ALGORITHM 3.1 Least-squares approximate Q-iteration for deterministic MDPs.

```
Input: dynamics $f$, reward function $\rho$, discount factor $\gamma$,
    approximation mapping $F$, samples $\{(x_{l_s}, u_{l_s}) \mid l_s = 1, \dots, n_s\}$
1: initialize parameter vector, e.g., $\theta_0 \leftarrow 0$
2: repeat at every iteration $\ell = 0, 1, 2, \dots$
3:     for $l_s = 1, \dots, n_s$ do
4:         $Q^{\ddagger}_{\ell+1}(x_{l_s}, u_{l_s}) \leftarrow \rho(x_{l_s}, u_{l_s}) + \gamma \max_{u'} [F(\theta_\ell)](f(x_{l_s}, u_{l_s}), u')$
5:     end for
6:     $\theta_{\ell+1} \leftarrow \theta^{\ddagger}$, where $\theta^{\ddagger} \in \arg\min_\theta \sum_{l_s=1}^{n_s} \left( Q^{\ddagger}_{\ell+1}(x_{l_s}, u_{l_s}) - [F(\theta)](x_{l_s}, u_{l_s}) \right)^2$
7: until $\theta_{\ell+1}$ is satisfactory
Output: $\widehat{\theta}^* = \theta_{\ell+1}$
```

There still remains the question of when to stop approximate Q-iteration, i.e., when to consider the parameter vector satisfactory. One possibility is to stop after a predetermined number of iterations L. Under the (reasonable) assumption that, at every iteration ℓ, the approximate Q-function $\widehat{Q}_\ell = F(\theta_\ell)$ is close to the Q-function Q_ℓ that would have been obtained by exact Q-iteration, the number L of iterations can be chosen with Equation (2.27) of Section 2.3.1, repeated here:

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

where $\varsigma_{\text{QI}} > 0$ is a desired bound on the suboptimality of a policy greedy in the Q-function obtained at iteration L. Of course, because $F(\theta_\ell)$ is not identical to Q_ℓ, it cannot be guaranteed that this bound is achieved. Nevertheless, L is still useful as an initial guess for the number of iterations needed to achieve a good performance.

Another possibility is to stop the algorithm when the distance between $\theta_{\ell+1}$ and θ_ℓ decreases below a certain threshold $\varepsilon_{\text{QI}} > 0$. This criterion is only useful if approximate Q-iteration is convergent to a fixed point (see Section 3.4.4 for convergence conditions). When convergence is not guaranteed, this criterion should be combined with a maximum number of iterations, to ensure that the algorithm stops in finite time.

Note that we have not explicitly considered the maximization issues or the estimation of expected values in the stochastic case. As explained in Section 3.2, one way to address the maximization difficulties is to discretize the action space. The expected values in the Q-iteration mapping for the stochastic case (2.23) need to be estimated from samples. For additional insight into this problem, see the fitted Q-iteration algorithm introduced in the next section.

A similar derivation 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 discretization procedures sometimes use interpolation, which leads to linearly parameterized approximators similar to (3.3).

3.4.2 Model-free value iteration with parametric approximation

From the class of model-free algorithms for approximate value iteration, we first discuss offline, batch algorithms, followed by online algorithms. Online algorithms, mainly approximate versions of Q-learning, have been studied since the beginning of the nineties (Lin, 1992; Singh et al., 1995; Horiuchi et al., 1996; Jouffe, 1998; Glorennec, 2000; Tuyls et al., 2002; Szepesvári and Smart, 2004; Murphy, 2005; Sherstov and Stone, 2005; Melo et al., 2008). A strong research thread in offline model-free value iteration emerged later (Ormoneit and Sen, 2002; Ernst et al., 2005; Riedmiller, 2005; Szepesvári and Munos, 2005; Ernst et al., 2006b; Antos et al., 2008a; Munos and Szepesvári, 2008; Farahmand et al., 2009a).

Offline model-free approximate value iteration

In the offline model-free case, the transition dynamics f and the reward function ρ are unknown.[3] Instead, only a batch of transition samples is available:

$$\{(x_{l_s}, u_{l_s}, x'_{l_s}, r_{l_s}) \,|\, l_s = 1, \ldots, n_s\}$$

where for every l_s, the next state x'_{l_s} and the reward r_{l_s} have been obtained as a result of taking action u_{l_s} in the state x_{l_s}. The transition samples may be independent, they may belong to a set of trajectories, or to a single trajectory. For instance, when the samples come from a single trajectory, they are typically ordered so that $x_{l_s+1} = x'_{l_s}$ for all $l_s < n_s$.

In this section, we present *fitted Q-iteration* (Ernst et al., 2005), a model-free version of approximate Q-iteration (3.15) that employs such a batch of samples. To obtain this version, two changes are made in the original, model-based algorithm. First, one has to use a sample-based projection mapping that considers only the samples (x_{l_s}, u_{l_s}), such as the least-squares regression (3.14). Second, because f and ρ are not available, the updated Q-function $Q^{\ddagger}_{\ell+1} = (T \circ F)(\theta_\ell)$ (3.13) at a given iteration ℓ cannot be computed directly. Instead, the Q-values $Q^{\ddagger}_{\ell+1}(x_{l_s}, u_{l_s})$ are replaced by quantities derived from the available data.

To understand how this is done, consider first the deterministic case. In this case,

[3]We take the point of view prevalent in the RL literature, which considers that the learning controller has no prior information about the problem to be solved. This means the reward function is unknown. In practice, of course, the reward function is almost always designed by the experimenter and is therefore known.

the updated Q-values are:

$$Q^{\ddagger}_{\ell+1}(x_{l_s}, u_{l_s}) = \rho(x_{l_s}, u_{l_s}) + \gamma \max_{u'} [F(\theta_\ell)](f(x_{l_s}, u_{l_s}), u') \tag{3.17}$$

where the Q-iteration mapping (2.22) has been used. Recall that $\rho(x_{l_s}, u_{l_s}) = r_{l_s}$ and that $f(x_{l_s}, u_{l_s}) = x'_{l_s}$. By performing these substitutions in (3.17), we get:

$$Q^{\ddagger}_{\ell+1}(x_{l_s}, u_{l_s}) = r_{l_s} + \gamma \max_{u'} [F(\theta_\ell)](x'_{l_s}, u') \tag{3.18}$$

and hence the updated Q-value can be computed exactly from the transition sample $(x_{l_s}, u_{l_s}, x'_{l_s}, r_{l_s})$, without using f or ρ.

Fitted Q-iteration works in deterministic and stochastic problems, and replaces each Q-value $Q^{\ddagger}_{\ell+1}(x_{l_s}, u_{l_s})$ by the quantity:

$$Q^{\ddagger}_{\ell+1, l_s} = r_{l_s} + \gamma \max_{u'} [F(\theta_\ell)](x'_{l_s}, u') \tag{3.19}$$

identical to the right-hand side of (3.18). As already discussed, in the deterministic case, this replacement is exact. In the stochastic case, the updated Q-value is the expectation of a random variable, of which $Q^{\ddagger}_{\ell+1, l_s}$ is only a *sample*. This updated Q-value is:

$$Q^{\ddagger}_{\ell+1}(x_{l_s}, u_{l_s}) = \mathrm{E}_{x' \sim \tilde{f}(x_{l_s}, u_{l_s}, \cdot)} \left\{ \tilde{\rho}(x_{l_s}, u_{l_s}, x') + \gamma \max_{u'} [F(\theta_\ell)](x', u') \right\}$$

where the Q-iteration mapping (2.23) has been used (note that $Q^{\ddagger}_{\ell+1}(x_{l_s}, u_{l_s})$ is the true Q-value and not a data point, so it is no longer subscripted by the sample index l_s). Nevertheless, most projection algorithms, including the least-squares regression (3.14), seek to approximate the expected value of their output variable conditioned by the input. In fitted Q-iteration, this means that the projection actually looks for θ_ℓ such that $F(\theta_\ell) \approx Q^{\ddagger}_{\ell+1}$, even though only samples of the form (3.19) are used. Therefore, the algorithm remains valid in the stochastic case.

Algorithm 3.2 presents fitted Q-iteration using least-squares projection (3.14). Note that, in the deterministic case, fitted Q-iteration is identical to model-based approximate Q-iteration (e.g., Algorithm 3.1), whenever both algorithms use the same approximator F, the same projection P, and the same state-action samples (x_{l_s}, u_{l_s}).

The considerations of Section 3.4.1 about the stopping criteria of approximate Q-iteration also apply to fitted Q-iteration, so they will not be repeated here. Moreover, once fitted Q-iteration has found a satisfactory parameter vector, a policy can be derived with (3.16).

We have introduced fitted Q-iteration in the parametric case, to clearly establish its link with model-based approximate Q-iteration. Neural networks are one class of parametric approximators that have been combined with fitted Q-iteration, leading to the so-called "neural fitted Q-iteration" (Riedmiller, 2005). However, fitted Q-iteration is more popular in combination with nonparametric approximators, so we will revisit it in the nonparametric context in Section 3.4.3.

ALGORITHM 3.2 Least-squares fitted Q-iteration with parametric approximation.

```
Input: discount factor $\gamma$,
    approximation mapping $F$, samples $\{(x_{l_s}, u_{l_s}, x'_{l_s}, r_{l_s}) \mid l_s = 1, \dots, n_s\}$
1: initialize parameter vector, e.g., $\theta_0 \leftarrow 0$
2: repeat at every iteration $\ell = 0, 1, 2, \dots$
3:     for $l_s = 1, \dots, n_s$ do
4:         $Q^{\ddagger}_{\ell+1,l_s} \leftarrow r_{l_s} + \gamma \max_{u'} [F(\theta_\ell)](x'_{l_s}, u')$
5:     end for
6:     $\theta_{\ell+1} \leftarrow \theta^{\ddagger}$, where $\theta^{\ddagger} \in \arg\min_\theta \sum_{l_s=1}^{n_s} \left( Q^{\ddagger}_{\ell+1,l_s} - [F(\theta)](x_{l_s}, u_{l_s}) \right)^2$
7: until $\theta_{\ell+1}$ is satisfactory
Output: $\widehat{\theta}^* = \theta_{\ell+1}$
```

Although we have assumed that the batch of samples is given in advance, fitted Q-iteration, together with other offline RL algorithms, can also be modified to use different batches of samples at different iterations. This property can be exploited, e.g., to add new, more informative samples in-between iterations. Ernst et al. (2006b) proposed a different, but related approach that integrates fitted Q-iteration into a larger iterative process. At every larger iteration, the entire fitted Q-iteration algorithm is run on the current batch of samples. Then, the solution obtained by fitted Q-iteration is used to generate new samples, e.g., by using an ε-greedy policy in the obtained Q-function. The entire cycle is then repeated.

Online model-free approximate value iteration

From the class of online algorithms for approximate value iteration, approximate versions of Q-learning are the most popular (Lin, 1992; Singh et al., 1995; Horiuchi et al., 1996; Jouffe, 1998; Glorennec, 2000; Tuyls et al., 2002; Szepesvári and Smart, 2004; Murphy, 2005; Sherstov and Stone, 2005; Melo et al., 2008). Recall from Section 2.3.2 that the original Q-learning updates the Q-function with (2.30):

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

after observing the next state x_{k+1} and reward r_{k+1}, as a result of taking action u_k in state x_k. A straightforward way to integrate approximation in Q-learning is by using gradient descent. We next explain how gradient-based Q-learning is obtained, following Sutton and Barto (1998, Chapter 8). We require that the approximation mapping F is differentiable in the parameters.

To simplify the formulas below, we denote the approximate Q-function at time k by $\widehat{Q}_k(x_k, u_k) = [F(\theta_k)](x_k, u_k)$, leaving the dependence on the parameter vector implicit. In order to derive gradient-based Q-learning, assume for now that after taking action u_k in state x_k, the algorithm is provided with the true optimal Q-value of the current state action pair, $Q^*(x_k, u_k)$, in addition to the next state x_{k+1} and reward r_{k+1}. Under these circumstances, the algorithm could aim to minimize the squared error between this optimal value and the current Q-value:

$$\begin{aligned}\theta_{k+1} &= \theta_k - \frac{1}{2}\alpha_k \frac{\partial}{\partial \theta_k} \left[Q^*(x_k,u_k) - \widehat{Q}_k(x_k,u_k) \right]^2 \\ &= \theta_k + \alpha_k \left[Q^*(x_k,u_k) - \widehat{Q}_k(x_k,u_k) \right] \frac{\partial}{\partial \theta_k} \widehat{Q}_k(x_k,u_k)\end{aligned}$$

Of course, $Q^*(x_k,u_k)$ is not available, but it can be replaced by an estimate derived from the Q-iteration mapping (2.22) or (2.23):

$$r_{k+1} + \gamma \max_{u'} \widehat{Q}_k(x_{k+1},u')$$

Note the similarity with the Q-function samples (3.19) used in fitted Q-iteration. The substitution leads to the approximate Q-learning update:

$$\theta_{k+1} = \theta_k + \alpha_k \left[r_{k+1} + \gamma \max_{u'} \widehat{Q}_k(x_{k+1},u') - \widehat{Q}_k(x_k,u_k) \right] \frac{\partial}{\partial \theta_k} \widehat{Q}_k(x_k,u_k) \tag{3.20}$$

We have actually obtained, in the square brackets, an approximation of the temporal difference. With a linearly parameterized approximator (3.3), the update (3.20) simplifies to:

$$\theta_{k+1} = \theta_k + \alpha_k \left[r_{k+1} + \gamma \max_{u'} \left(\phi^{\mathrm{T}}(x_{k+1},u')\theta_k \right) - \phi^{\mathrm{T}}(x_k,u_k)\theta_k \right] \phi(x_k,u_k) \tag{3.21}$$

Note that, like the original Q-learning algorithm of Section 2.3.2, approximate Q-learning requires exploration. As an example, Algorithm 3.3 presents gradient-based Q-learning with a linear parametrization and ε-greedy exploration. For an explanation and examples of the learning rate and exploration schedules used in this algorithm, see Section 2.3.2.

ALGORITHM 3.3 Q-learning with a linear parametrization and ε-greedy exploration.

```
Input: discount factor $\gamma$,
    BFs $\phi_1,\dots,\phi_n : X \times U \to \mathbb{R}$,
    exploration schedule $\{\varepsilon_k\}_{k=0}^{\infty}$, learning rate schedule $\{\alpha_k\}_{k=0}^{\infty}$
1: initialize parameter vector, e.g., $\theta_0 \leftarrow 0$
2: measure initial state $x_0$
3: for every time step $k = 0,1,2,\dots$ do
4:     $u_k \leftarrow \begin{cases} u \in \arg\max_{\bar{u}} \left( \phi^{\mathrm{T}}(x_k,\bar{u})\theta_k \right) & \text{with probability } 1-\varepsilon_k \text{ (exploit)} \\ \text{a uniform random action in } U & \text{with probability } \varepsilon_k \text{ (explore)} \end{cases}$
5:     apply $u_k$, measure next state $x_{k+1}$ and reward $r_{k+1}$
6:     $\theta_{k+1} \leftarrow \theta_k + \alpha_k \left[ r_{k+1} + \gamma \max_{u'} \left( \phi^{\mathrm{T}}(x_{k+1},u')\theta_k \right) - \phi^{\mathrm{T}}(x_k,u_k)\theta_k \right] \phi(x_k,u_k)$
7: end for
```

In the literature, Q-learning has been combined with a variety of approximators, for example:

- linearly parameterized approximators, including tile coding (Watkins, 1989; Sherstov and Stone, 2005), as well as so-called interpolative representations (Szepesvári and Smart, 2004) and "soft" state aggregation (Singh et al., 1995).
- fuzzy rule-bases (Horiuchi et al., 1996; Jouffe, 1998; Glorennec, 2000), which can also be linear in the parameters.
- neural networks (Lin, 1992; Touzet, 1997).

While approximate Q-learning is easy to use, it typically requires many transition samples (i.e., many steps, k) before it can obtain a good approximation of the optimal Q-function. One possible approach to alleviate this problem is to store transition samples in a database and reuse them multiple times, similarly to how the batch algorithms of the previous section work. This procedure is known as experience replay (Lin, 1992; Kalyanakrishnan and Stone, 2007). Another option is to employ so-called eligibility traces, which allow the parameter updates at the current step to also incorporate information about recently observed transitions (e.g., Singh and Sutton, 1996). This mechanism makes use of the fact that the latest transition is the causal result of an entire trajectory.

3.4.3 Value iteration with nonparametric approximation

In this section, we first describe fitted Q-iteration with nonparametric approximation. We then point out some other algorithms that combine value iteration with nonparametric approximators.

The fitted Q-iteration algorithm was introduced in a parametric context in Section 3.4.2, see Algorithm 3.2. In the nonparametric case, fitted Q-iteration can no longer be described using approximation and projection mappings that remain unchanged from one iteration to the next. Instead, fitted Q-iteration can be regarded as generating an entirely new, nonparametric approximator at every iteration. Algorithm 3.4 outlines a general template for fitted Q-iteration with nonparametric approximation. The nonparametric regression at line 6 is responsible for generating a new approximator $\widehat{Q}_{\ell+1}$ that accurately represents the updated Q-function $Q^{\ddagger}_{\ell+1}$, using the information provided by the available samples $Q^{\ddagger}_{\ell+1,l_s}$, $l_s = 1, \ldots n_s$.

Fitted Q-iteration has been combined with several types of nonparametric approximators, including kernel-based approximators (Farahmand et al., 2009a) and ensembles of regression trees (Ernst et al., 2005, 2006b); see Appendix **??** for a description of such an ensemble.

Of course, other DP and RL algorithms besides fitted Q-iteration can also be combined with nonparametric approximation. For instance, Deisenroth et al. (2009) employed Gaussian processes in approximate value iteration. They proposed two algorithms: one that assumes that a model of the (deterministic) dynamics is known, and another that estimates a Gaussian-process approximation of the dynamics from

ALGORITHM 3.4 Fitted Q-iteration with nonparametric approximation.

```
Input: discount factor $\gamma$,
    samples $\{(x_{l_s}, u_{l_s}, x'_{l_s}, r_{l_s}) \,|\, l_s = 1, \dots, n_s\}$
1: initialize Q-function approximator, e.g., $\widehat{Q}_0 \leftarrow 0$
2: repeat at every iteration $\ell = 0, 1, 2, \dots$
3:     for $l_s = 1, \dots, n_s$ do
4:         $Q^{\ddagger}_{\ell+1,l_s} \leftarrow r_{l_s} + \gamma \max_{u'} \widehat{Q}_\ell(x'_{l_s}, u')$
5:     end for
6:     find $\widehat{Q}_{\ell+1}$ using
               nonparametric regression on $\{((x_{l_s}, u_{l_s}), Q^{\ddagger}_{\ell+1,l_s}) \,|\, l_s = 1, \dots, n_s\}$
7: until $\widehat{Q}_{\ell+1}$ is satisfactory
Output: $\widehat{Q}^* = \widehat{Q}_{\ell+1}$
```

transition data. Ormoneit and Sen (2002) employed kernel-based approximation in model-free approximate value iteration for discrete-action problems.

3.4.4 Convergence and the role of nonexpansive approximation

An important question in approximate DP/RL is whether the approximate solution computed by the algorithm converges, and, if it does converge, 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.

Convergence of model-based approximate value iteration

The convergence proofs for approximate value iteration often rely on contraction mapping arguments. Consider for instance approximate Q-iteration (3.15). The Q-iteration mapping T is a contraction in the infinity norm with factor $\gamma < 1$, as already explained in Section 2.3.1. If the composite mapping $P \circ T \circ F$ of approximate Q-iteration is also a contraction, i.e., if for any pair of parameter vectors θ, θ' and for some $\gamma' < 1$:

$$\|(P \circ T \circ F)(\theta) - (P \circ T \circ F)(\theta')\|_\infty \leq \gamma' \|\theta - \theta'\|_\infty$$

then approximate Q-iteration asymptotically converges to a unique fixed point, which we denote by θ^*.

One way to ensure that $P \circ T \circ F$ is a contraction is to require F and P to be nonexpansions, i.e.:

$$\begin{aligned} \|F(\theta) - F(\theta')\|_\infty &\leq \|\theta - \theta'\|_\infty \quad \text{for all pairs } \theta, \theta' \\ \|P(Q) - P(Q')\|_\infty &\leq \|Q - Q'\|_\infty \quad \text{for all pairs } Q, Q' \end{aligned}$$

Note that in this case the contraction factor of $P \circ T \circ F$ is the same as that of T: $\gamma' = \gamma < 1$. Under these conditions, as we will describe next, suboptimality bounds can be derived on the approximate Q-function $F(\theta^*)$ and on any policy $\widehat{h}^*$ that is greedy in this Q-function, i.e., that satisfies:

$$\widehat{h}^*(x) \in \arg\max_u [F(\theta^*)](x,u) \tag{3.22}$$

Denote by $\mathscr{F}_{F\circ P} \subset \mathscr{Q}$ the set of fixed points of the composite mapping $F \circ P$, which is assumed nonempty. Define the minimum distance between Q^* and any fixed point of $F \circ P$:[4]

$$\varsigma^*_{\text{QI}} = \min_{Q' \in \mathscr{F}_{F\circ P}} \|Q^* - Q'\|_\infty$$

This distance characterizes the representation power of the approximator: the better the representation power, the closer the nearest fixed point of $F \circ P$ will be to Q^*, and the smaller ς^*_{QI} will be. Using this distance, the convergence point θ^* of approximate Q-iteration satisfies the following suboptimality bounds:

$$\|Q^* - F(\theta^*)\|_\infty \le \frac{2\varsigma^*_{\text{QI}}}{1-\gamma} \tag{3.23}$$

$$\|Q^* - Q^{\widehat{h}^*}\|_\infty \le \frac{4\gamma\varsigma^*_{\text{QI}}}{(1-\gamma)^2} \tag{3.24}$$

where $Q^{\widehat{h}^*}$ is the Q-function of the near-optimal policy $\widehat{h}^*$ (3.22). These bounds can be derived similarly to those for approximate V-iteration found by Gordon (1995); Tsitsiklis and Van Roy (1996). Equation (3.23) gives the suboptimality bound of the approximately optimal Q-function, whereas (3.24) gives the suboptimality bound of the resulting approximately optimal policy, and may be more relevant in practice. The following relationship between the policy suboptimality and the Q-function suboptimality was used to obtain (3.24), and is also valid in general:

$$\|Q^* - Q^h\|_\infty \le \frac{2\gamma}{(1-\gamma)}\|Q^* - Q\|_\infty \tag{3.25}$$

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

Ideally, the optimal Q-function Q^* is a fixed point of $F \circ P$, in which case $\varsigma^*_{\text{QI}} = 0$, and approximate Q-iteration asymptotically converges to Q^*. For instance, when Q^* happens to be exactly representable by F, a well-chosen tandem of approximation and projection mappings should ensure that Q^* is in fact a fixed point of $F \circ P$. In practice, of course, ς^*_{QI} will rarely be 0, and only near-optimal solutions can be obtained.

In order to take advantage of these theoretical guarantees, F and P should be nonexpansions. When F is linearly parameterized (3.3), it is fairly easy to ensure its nonexpansiveness by normalizing the BFs ϕ_l, so that for every x and u, we have:

$$\sum_{l=1}^{n} \phi_l(x,u) = 1$$

[4]For simplicity, we assume that the minimum in this equation exists. If the minimum does not exist, then ς^*_{QI} should be taken as small as possible so that there still exists a $Q' \in \mathscr{F}_{F\circ P}$ with $\|Q' - Q^*\|_\infty \le \varsigma^*_{\text{QI}}$.

Ensuring that P is nonexpansive is more difficult. For instance, the least-squares projection (3.14) can in general be an expansion, and examples of divergence when using it have been given (Tsitsiklis and Van Roy, 1996; Wiering, 2004). One way to make least-squares projection nonexpansive is to choose exactly $n_s = n$ state-action samples (x_l, u_l), $l = 1, \dots, n$, and require that:

$$\phi_l(x_l, u_l) = 1, \quad \phi_{l'}(x_l, u_l) = 0 \;\; \forall l' \neq l$$

These samples could be, e.g., the centers of the BFs. Then, the projection (3.14) simplifies to an assignment that associates each parameter with the Q-value of the corresponding sample:

$$[P(Q)]_l = Q(x_l, u_l) \tag{3.26}$$

where the notation $[P(Q)]_l$ refers to the lth component in the parameter vector $P(Q)$. This mapping is clearly nonexpansive. More general, but still restrictive conditions on the BFs under which convergence and near optimality are guaranteed are given in (Tsitsiklis and Van Roy, 1996).

Convergence of model-free approximate value iteration

Like in the model-based case, convergence guarantees for offline, batch model-free value iteration typically rely on nonexpansive approximation. In fitted Q-iteration with parametric approximation (Algorithm 3.2), care must be taken when selecting F and P, to prevent possible expansion and divergence. Similarly, in fitted Q-iteration with nonparametric approximation (Algorithm 3.4), the nonparametric regression algorithm should have nonexpansive properties. Certain types of kernel-based approximators satisfy this condition (Ernst et al., 2005). The convergence of the kernel-based V-iteration algorithm of Ormoneit and Sen (2002) is also guaranteed under nonexpansiveness assumptions.

More recently, a different class of theoretical results for batch value iteration have been developed, which do not rely on nonexpansion properties and do not concern the asymptotic case. Instead, these results provide probabilistic bounds on the suboptimality of the policy obtained by using a finite number of samples, after a finite number of iterations. Besides the number of samples and iterations, such *finite-sample* bounds typically depend on the representation power of the approximator and on certain properties of the MDP. For instance, Munos and Szepesvári (2008) provided finite-sample bounds for approximate V-iteration in discrete-action MDPs, while Farahmand et al. (2009a) focused on fitted Q-iteration in the same type of MDPs. Antos et al. (2008a) gave finite-sample bounds for fitted Q-iteration in the more difficult case of continuous-action MDPs.

In the area of online approximate value iteration, as already discussed in Section 3.4.2, the main representative is approximate Q-learning. Many variants of approximate Q-learning are heuristic and do not guarantee convergence (Horiuchi et al., 1996; Touzet, 1997; Jouffe, 1998; Glorennec, 2000; Millán et al., 2002). Convergence of approximate Q-learning has been proven for linearly parameterized approximators, under the requirement that the policy followed by Q-learning remains *unchanged* during the learning process (Singh et al., 1995; Szepesvári and Smart, 2004; Melo et al., 2008). This requirement is restrictive, because it does not allow the controller to improve its performance, even if it has gathered knowledge that would enable it to do so. Among these results, Singh et al. (1995) and Szepesvári and Smart (2004) proved the convergence of approximate Q-learning with nonexpansive, linearly parameterized approximation. Melo et al. (2008) showed that gradient-based Q-learning (3.21) converges without requiring nonexpansive approximation, but at the cost of other restrictive assumptions.

Consistency of approximate value iteration

Besides convergence, another important theoretical property of algorithms for approximate DP and RL is consistency. In model-based value iteration, and more generally in DP, an algorithm is said to be consistent if the approximate value function converges to the optimal one as the approximation accuracy increases (e.g., Gonzalez and Rofman, 1985; Chow and Tsitsiklis, 1991; Santos and Vigo-Aguiar, 1998). In model-free value iteration, and more generally in RL, consistency is sometimes understood as the convergence to a well-defined solution as the number of samples increases. The stronger result of convergence to an optimal solution as the approximation accuracy also increases was proven in (Ormoneit and Sen, 2002; Szepesvári and Smart, 2004).

3.4.5 Example: Approximate Q-iteration for a DC motor

In closing the discussion on approximate value iteration, we provide a numerical example involving a DC motor control problem. This example shows how approximate value iteration algorithms can be used in practice. The first part of the example concerns a basic version of approximate Q-iteration that relies on a gridding of the state space and on a discretization of the action space, while the second part employs the state-of-the-art, fitted Q-iteration algorithm with nonparametric approximation (Algorithm 3.4).

Consider a second-order discrete-time model of an electrical DC (direct current) motor:

$$\begin{aligned} x_{k+1} &= f(x_k, u_k) = Ax_k + Bu_k \\ A &= \begin{bmatrix} 1 & 0.0049 \\ 0 & 0.9540 \end{bmatrix}, \quad B = \begin{bmatrix} 0.0021 \\ 0.8505 \end{bmatrix} \end{aligned} \tag{3.27}$$

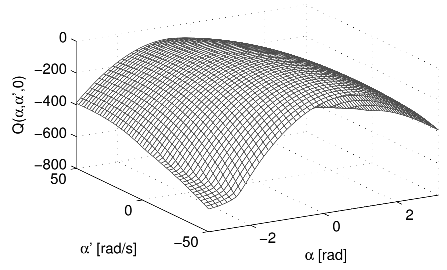
This model was obtained by discretizing a continuous-time model of the DC motor, which was developed by first-principles modeling (e.g., Khalil, 2002, Chapter 1) of a real DC motor. The discretization was performed with the zero-order-hold method (Franklin et al., 1998), using a sampling time of $T_s = 0.005$ s. Using saturation, the shaft angle $x_{1,k} = \alpha$ is bounded to $[-\pi, \pi]$ rad, the angular velocity $x_{2,k} = \dot{\alpha}$ to $[-16\pi, 16\pi]$ rad/s, and the control input u_k to $[-10, 10]$ V.

The control goal is to stabilize the DC motor in the zero equilibrium ($x = 0$). The following quadratic reward function is chosen to express this goal:

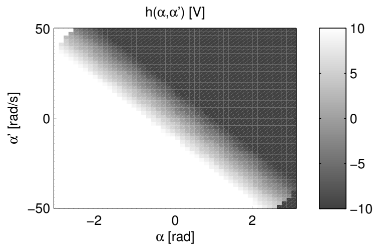
$$
\begin{aligned}
r_{k+1} &= \rho(x_k, u_k) = -x_k^{\mathrm{T}} Q_{\text{rew}} x_k - R_{\text{rew}} u_k^2 \\
Q_{\text{rew}} &= \begin{bmatrix} 5 & 0 \\ 0 & 0.01 \end{bmatrix}, \quad R_{\text{rew}} = 0.01
\end{aligned}
\tag{3.28}
$$

This reward function leads to a discounted quadratic regulation problem. A (near-)optimal policy will drive the state (close) to 0, while also minimizing the magnitude of the states along the trajectory and the control effort. The discount factor was chosen to be $\gamma = 0.95$, which is sufficiently large to lead to an optimal policy that produces a good stabilizing control behavior.[5]

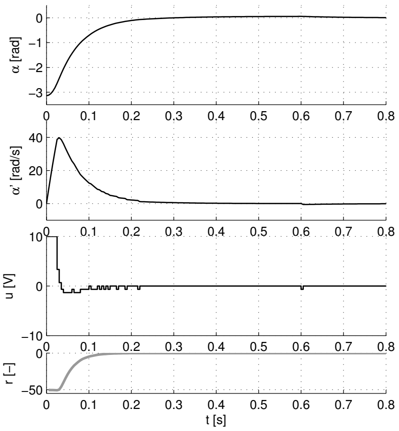
Figure 3.5 presents a near-optimal solution to this problem, including a representative state-dependent slice through the Q-function (obtained by setting the action



(a) Slice through a near-optimal Q-function, for $u = 0$.



(b) A near-optimal policy.



(c) Controlled trajectory from $x_0 = [-\pi, 0]^{\mathrm{T}}$.

FIGURE 3.5 A near-optimal solution for the DC motor.

[5]Note that a distinction is made between the optimality under the chosen reward function and discount factor, and the actual (albeit subjective) quality of the control behavior.

argument u to 0), a greedy policy in this Q-function, and a representative trajectory that is controlled by this policy. To find the near-optimal solution, the convergent and consistent fuzzy Q-iteration algorithm (which will be discussed in detail in Chapter 4) was applied. An accurate approximator over the state space was used, together with a fine discretization of the action space, which contains 31 equidistant actions.

Grid Q-iteration

As an example of approximate value iteration, we apply a Q-iteration algorithm that relies on state aggregation and action discretization, a type of approximator introduced in Example 3.1. The state space is partitioned into N disjoint rectangles. Denote by X_i the ith rectangle in the state space partition. For this problem, the following three discrete actions suffice to produce an acceptable stabilizing control behavior: $u_1 = -10$, $u_2 = 0$, $u_3 = 10$ (i.e., applying maximum torque in either direction, and no torque at all). So, the discrete action space is $U_\mathrm{d} = \{-10, 0, 10\}$. Recall from Example 3.1 that the state-action BFs are given by (3.8), repeated here for easy reference:

$$\phi_{[i,j]}(x,u) = \begin{cases} 1 & \text{if } x \in X_i \text{ and } u = u_j \\ 0 & \text{otherwise} \end{cases} \tag{3.29}$$

where $[i, j] = i + (j-1)N$. To derive the projection mapping P, the least-squares projection (3.14) is used, taking the cross-product of the sets $\{x_1, \dots, x_N\}$ and U_d as state-action samples, where x_i denotes the center of the ith rectangle X_i. These samples satisfy the conditions to simplify P to an assignment of the form (3.26), namely:

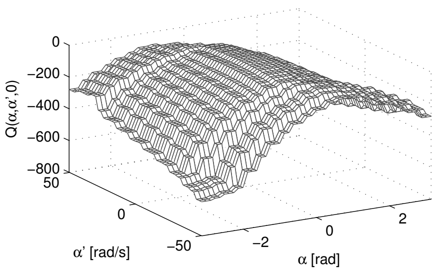
$$[P(Q)]_{[i,j]} = Q(x_i, u_j) \tag{3.30}$$

Using a linearly parameterized approximator with the BFs (3.29) and the projection (3.30) yields the grid Q-iteration algorithm. Because F and P are nonexpansions, the algorithm is convergent.

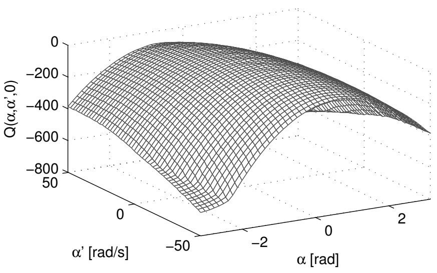
To apply grid Q-iteration to the DC motor problem, two different grids over the state space are used: a coarse grid, with 20 equidistant bins on each axis (leading to $20^2 = 400$ rectangles); and a fine grid, with 400 equidistant bins on each axis (leading to $400^2 = 160\,000$ rectangles). The algorithm is considered to have converged 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 occurred after 160 iterations, and for the fine grid, after 123. This shows that the number of iterations required for convergence does not necessarily increase with the number of parameters.

Figure 3.6 shows slices through the resulting Q-functions, together with corresponding policies and representative controlled trajectories. The accuracy in representing the Q-function and policy is better for the fine grid (Figures 3.6(b) and 3.6(d)) than for the coarse grid (Figures 3.6(a) and 3.6(c)). Axis-oriented policy artifacts appear for both grid sizes, due to the limitations of the chosen type of approximator. For instance, the piecewise-constant nature of the approximator is clearly visible in Figure 3.6(a). Compared to the near-optimal trajectory of Figure 3.5(c), the grid Q-iteration trajectories in Figures 3.6(e) and 3.6(f) do not reach the goal state $x = 0$ with the same accuracy. With the coarse-grid policy, there is a large steady-state error of the angle α, while the fine-grid policy leads to chattering of the control action.

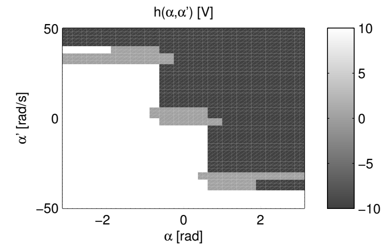
The execution time of grid Q-iteration was 0.06 s for the coarse grid, and 7.80 s



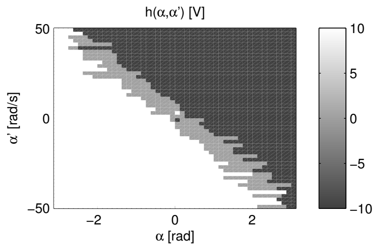
(a) Slice through coarse-grid Q-function, for $u = 0$.



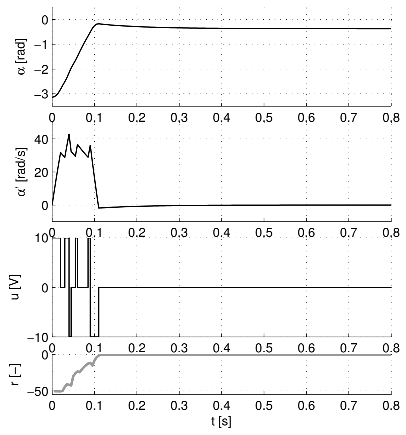
(b) Slice through fine-grid Q-function, for $u = 0$.



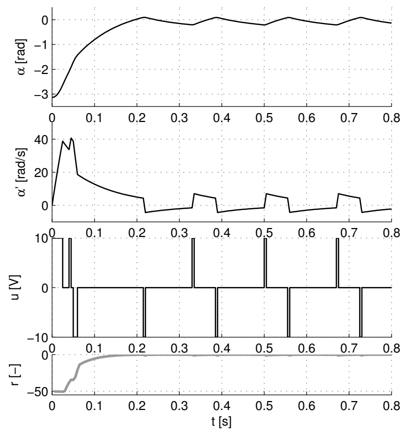
(c) Coarse-grid policy.



(d) Fine-grid policy.



(e) Trajectory from $x_0 = [-\pi, 0]^{\mathrm{T}}$, controlled by the coarse-grid policy.



(f) Trajectory from $x_0 = [-\pi, 0]^{\mathrm{T}}$, controlled by the fine-grid policy.

FIGURE 3.6
Grid Q-iteration solutions for the DC motor. The results obtained with the coarse grid are shown on the left-hand side of the figure, and those obtained with the fine grid on the right-hand side.

for the fine grid.[6] The fine grid is significantly more computationally expensive to use, because it has a much larger number of parameters to update (480 000, versus 1200 for the coarse grid).

Fitted Q-iteration

Next, we apply fitted Q-iteration (Algorithm 3.4) to the DC motor problem, using ensembles of extremely randomized trees (Geurts et al., 2006) to approximate the Q-function. For a description of this approximator, see Appendix **??**. The same discrete actions are employed as for grid Q-iteration: $U_\mathrm{d} = \{-10, 0, 10\}$. A distinct ensemble of regression trees is used to approximate the Q-function for each of these discrete actions – in analogy to the discrete-action grid approximator. The construction of the tree ensembles is driven by three meta-parameters:

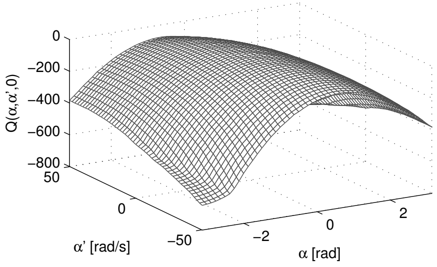
- Each ensemble contains N_tr trees. We set this parameter equal to 50.
- To split a node, K_tr randomly chosen cut directions are evaluated, and the one that maximizes a certain score is selected. We set K_tr equal to the dimensionality 2 of the input to the regression trees (the 2-dimensional state variable), which is its recommended default value (Geurts et al., 2006).
- A node is only split further when it is associated with at least $n_\mathrm{tr}^\mathrm{min}$ samples. Otherwise, it remains a leaf node. We set $n_\mathrm{tr}^\mathrm{min}$ to its default value of 2, which means that the trees are fully developed.

Fitted Q-iteration is supplied with a set of samples consisting of the cross-product between a regular grid of 100×100 points in the state space, and the 3 discrete actions. This ensures the meaningfulness of the comparison with grid Q-iteration, which employed similarly placed samples. Fitted Q-iteration is run for a predefined number of 100 iterations, and the Q-function found after the 100th iteration is considered satisfactory.

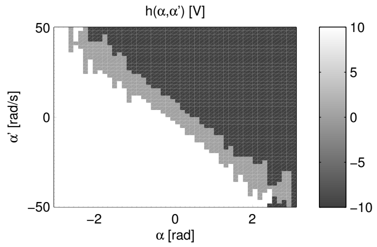
Figure 3.7 shows the solution obtained. This is similar in quality to the solution obtained by grid Q-iteration with the fine grid, and better than the solution obtained with the coarse grid (Figure 3.6).

The execution time of fitted Q-iteration was approximately 2151 s, several orders of magnitude larger than the execution time of grid Q-iteration (recall that the latter was 0.06 s for the coarse grid, and 7.80 s for the fine grid). Clearly, finding a more powerful nonparametric approximator is much more computationally intensive than updating the parameters of the simple, grid-based approximator.

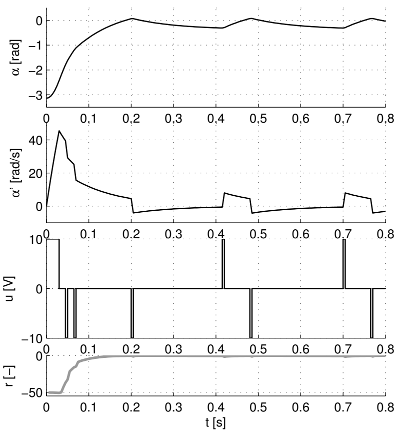
[6]All the execution times reported in this chapter were recorded while running the algorithms in MATLAB® 7 on a PC with an Intel Core 2 Duo T9550 2.66 GHz CPU and with 3 GB RAM. For value iteration and policy iteration, the reported execution times do not include the time required to simulate the system for every state-action sample in order to obtain the next state and reward.



(a) Slice through Q-function for $u = 0$.



(b) Policy.



(c) Controlled trajectory from $x_0 = [-\pi, 0]^T$.

FIGURE 3.7 Fitted Q-iteration solution for the DC motor.

3.5 Approximate policy iteration

Policy iteration algorithms evaluate policies by constructing their value functions, and use these value functions to find new, improved policies. They were introduced in Section 2.4. In large or continuous spaces, policy evaluation cannot be solved exactly, and the value function has to be approximated. *Approximate policy evaluation* is a difficult problem, because, like approximate value iteration, it involves finding an approximate solution to a Bellman equation. Special requirements must be imposed to ensure that a meaningful approximate solution exists and can be found by appropriate algorithms. *Policy improvement* relies on solving maximization problems over the action variables, which involve fewer technical difficulties (although they may still be hard to solve when the action space is large). Often, an explicit representation of the policy can be avoided, by computing improved actions on demand from the current value function. Alternatively, the policy can be represented explicitly, in which case policy approximation is generally required. In this case, solving a classical supervised learning problem is necessary to perform policy improvement.

Algorithm 3.5 outlines a general template for approximate policy iteration with Q-function policy evaluation. Note that at line 4, when there are multiple maximizing actions, the expression "$\approx \arg\max_u \ldots$" should be interpreted as "approximately equal to one of the maximizing actions."

ALGORITHM 3.5 Approximate policy iteration with Q-functions.

```
1: initialize policy $\widehat{h}_0$
2: repeat at every iteration $\ell = 0, 1, 2, \ldots$
3:     find $\widehat{Q}^{\widehat{h}_\ell}$, an approximate Q-function of $\widehat{h}_\ell$                    ▷ policy evaluation
4:     find $\widehat{h}_{\ell+1}$ so that $\widehat{h}_{\ell+1}(x) \approx \arg\max_u \widehat{Q}^{\widehat{h}_\ell}(x,u), \forall x \in X$ ▷ policy improvement
5: until $\widehat{h}_{\ell+1}$ is satisfactory
Output: $\widehat{h}^* = \widehat{h}_{\ell+1}$
```

Figure 3.8 (repeated from the relevant part of Figure 3.2) illustrates the structure of our upcoming presentation. We first discuss in detail the approximate policy evaluation component, starting in Section 3.5.1 with a class of algorithms that can be derived along the same lines as approximate value iteration. In Section 3.5.2, model-free policy evaluation algorithms with linearly parameterized approximation are introduced, which aim to solve a projected form of the Bellman equation. Section 3.5.3 briefly reviews policy evaluation with nonparametric approximation, and Section 3.5.4 outlines a model-based, direct simulation approach for policy evaluation. In Section 3.5.5, we move on to the policy improvement component and the resulting approximate policy iteration. Theoretical results about approximate policy iteration are reviewed in Section 3.5.6, and a numerical example is provided in Section 3.5.7 (the material of these last two sections is not represented in Figure 3.8).

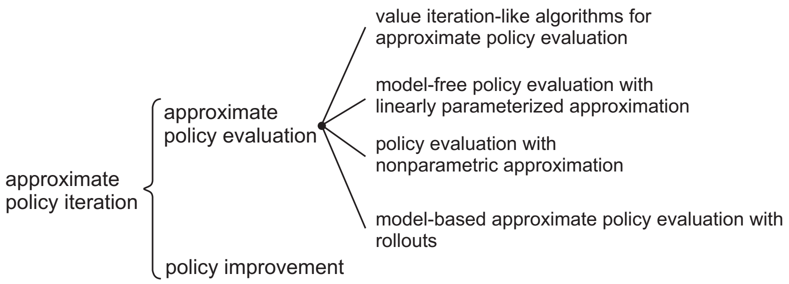

value iteration-like algorithms for approximate policy evaluation
model-free policy evaluation with linearly parameterized approximation
approximate policy evaluation
policy evaluation with nonparametric approximation
approximate policy iteration
model-based approximate policy evaluation with rollouts
policy improvement

FIGURE 3.8
The organization of the algorithms for approximate policy evaluation and policy improvement presented in the sequel.

3.5.1 Value iteration-like algorithms for approximate policy evaluation

We start our discussion of approximate policy evaluation with a class of algorithms that can be derived along entirely similar lines to approximate value iteration. These algorithms can be model-based or model-free, and can use parametric or nonparametric approximation. We focus here on the parametric case, and discuss two representative algorithms, one model-based and the other model-free. These two algorithms are similar to approximate Q-iteration (Section 3.4.1) and to fitted Q-iteration (Section 3.4.2), respectively. In order to streamline the presentation, we will often refer to these counterparts and to their derivation.

The first algorithm that we develop is based on the model-based, iterative policy evaluation for Q-functions (Section 2.3.1). Denote the policy to be evaluated by h. Recall that policy evaluation for Q-functions starts from an arbitrary Q-function Q_0^h, which is updated at each iteration τ using (2.38), repeated here for easy reference:

$$Q_{\tau+1}^h = T^h(Q_\tau^h)$$

where T^h is the policy evaluation mapping, given by (2.35) in the deterministic case and by (2.36) in the stochastic case. The algorithm asymptotically converges to the Q-function Q^h of the policy h, which is the solution of the Bellman equation (2.39), also repeated here:

$$Q^h = T^h(Q^h) \tag{3.31}$$

Policy evaluation for Q-functions can be extended to the approximate case in a similar way as approximate Q-iteration (see Section 3.4.1). As with approximate Q-iteration, an approximation mapping $F : \mathbb{R}^n \to \mathscr{Q}$ is used to compactly represent Q-functions using parameter vectors $\theta^h \in \mathbb{R}^n$, and a projection mapping $P : \mathscr{Q} \to \mathbb{R}^n$ is used to find parameter vectors that represent the updated Q-functions well.

The iterative, *approximate policy evaluation for Q-functions* starts with an arbitrary (e.g., identically 0) parameter vector θ_0^h, and updates this vector at every iteration τ using the composition of mappings P, T^h, and F:

$$\theta_{\tau+1}^h = (P \circ T^h \circ F)(\theta_\tau^h) \tag{3.32}$$

The algorithm is stopped once a satisfactory parameter vector $\widehat{\theta}^h$ has been found. Under conditions similar to those for value iteration (Section 3.4.4), the composite mapping $P \circ T^h \circ F$ is a contraction, and therefore has a fixed point θ^h to which the update (3.32) asymptotically converges. This is true, e.g., if both F and P are nonexpansions.

As an example, Algorithm 3.6 shows approximate policy evaluation for Q-functions in the case of deterministic MDPs, using the least-squares projection (3.14). In this algorithm, $Q_{\tau+1}^{h,\ddagger}$ denotes the intermediate, updated Q-function:

$$Q_{\tau+1}^{h,\ddagger} = (T^h \circ F)(\theta_\tau^h)$$

Because deterministic MDPs are considered, $Q_{\tau+1}^{h,\ddagger}(x_{l_s}, u_{l_s})$ is computed at line 4 using the policy evaluation mapping (2.35). Note the similarity of Algorithm 3.6 with the approximate *Q-iteration* for MDPs (Algorithm 3.1).

ALGORITHM 3.6 Approximate policy evaluation for Q-functions in deterministic MDPs.

```
Input: policy $h$ to be evaluated, dynamics $f$, reward function $\rho$, discount factor $\gamma$,
    approximation mapping $F$, samples $\{(x_{l_s}, u_{l_s}) \mid l_s = 1, \dots, n_s\}$
1: initialize parameter vector, e.g., $\theta_0^h \leftarrow 0$
2: repeat at every iteration $\tau = 0, 1, 2, \dots$
3:     for $l_s = 1, \dots, n_s$ do
4:         $Q_{\tau+1}^{h,\ddagger}(x_{l_s}, u_{l_s}) \leftarrow \rho(x_{l_s}, u_{l_s}) + \gamma [F(\theta_\tau^h)](f(x_{l_s}, u_{l_s}), h(f(x_{l_s}, u_{l_s})))$
5:     end for
6:     $\theta_{\tau+1}^h \leftarrow \theta^{h,\ddagger}$, where $\theta^{h,\ddagger} \in \arg\min_\theta \sum_{l_s=1}^{n_s} \left(Q_{\tau+1}^{h,\ddagger}(x_{l_s}, u_{l_s}) - [F(\theta)](x_{l_s}, u_{l_s})\right)^2$
7: until $\theta_{\tau+1}^h$ is satisfactory
Output: $\widehat{\theta}^h = \theta_{\tau+1}^h$
```

The second algorithm that we develop is an analogue of fitted Q-iteration, so it will be called *fitted policy evaluation for Q-functions*. It can also be seen as a model-free variant of the approximate policy evaluation for Q-functions developed above. In this variant, a batch of transition samples is assumed to be available:

$$\{(x_{l_s}, u_{l_s}, x'_{l_s}, r_{l_s}) \mid l_s = 1, \dots, n_s\}$$

where for every l_s, the next state x'_{l_s} and the reward r_{l_s} have been obtained after taking action u_{l_s} in the state x_{l_s}. At every iteration, samples of the updated Q-function $Q_{\tau+1}^{h,\ddagger}$ are computed with:

$$Q_{\tau+1,l_s}^{h,\ddagger} = r_{l_s} + \gamma [F(\theta_\tau)](x'_{l_s}, h(x'_{l_s}))$$

In the deterministic case, the quantity $Q_{\tau+1,l_s}^{h,\ddagger}$ is identical to the updated Q-value $Q_{\tau+1}^{h,\ddagger}(x_{l_s}, u_{l_s})$ (see, e.g., line 4 of Algorithm 3.6). In the stochastic case, $Q_{\tau+1,l_s}^{h,\ddagger}$ is a *sample* of the random variable that has the updated Q-value as its expectation. A complete iteration of the algorithm is obtained by computing an updated parameter vector with a projection mapping, using the samples $((x_{l_s}, u_{l_s}), Q_{\tau+1,l_s}^{h,\ddagger})$.

Algorithm 3.7 presents fitted policy evaluation for Q-functions, using the least-squares projection (3.14). Note that, in the deterministic case, fitted policy evaluation is identical to model-based, approximate policy evaluation (e.g., Algorithm 3.6), if both algorithms use the same approximation and projection mappings, together with the same state-action samples (x_{l_s}, u_{l_s}).

3.5.2 Model-free policy evaluation with linearly parameterized approximation

A different, dedicated framework for approximate policy evaluation can be developed when linearly parameterized approximators are employed. By exploiting the linearity of the approximator in combination with the linearity of the policy evaluation mapping (see Section 2.4.1), it is possible to derive a specific approximate form

ALGORITHM 3.7 Fitted policy evaluation for Q-functions.

```
Input: policy $h$ to be evaluated, discount factor $\gamma$,
    approximation mapping $F$, samples $\{(x_{l_s}, u_{l_s}, x'_{l_s}, r_{l_s}) \,|\, l_s = 1, \dots, n_s\}$
1: initialize parameter vector, e.g., $\theta_0^h \leftarrow 0$
2: repeat at every iteration $\tau = 0, 1, 2, \dots$
3:     for $l_s = 1, \dots, n_s$ do
4:         $Q_{\tau+1,l_s}^{h,\ddagger} \leftarrow r_{l_s} + \gamma [F(\theta_\tau^h)](x'_{l_s}, h(x'_{l_s}))$
5:     end for
6:     $\theta_{\tau+1}^h \leftarrow \theta^{h,\ddagger}$, where $\theta^{h,\ddagger} \in \arg\min_\theta \sum_{l_s=1}^{n_s} \left( Q_{\tau+1,l_s}^{h,\ddagger} - [F(\theta)](x_{l_s}, u_{l_s}) \right)^2$
7: until $\theta_{\tau+1}^h$ is satisfactory
Output: $\widehat{\theta}^h = \theta_{\tau+1}^h$
```

of the Bellman equation, called the "projected Bellman equation," which is linear in the parameter vector.[7] Efficient algorithms can be developed to solve this equation. In contrast, in approximate value iteration, the maximum operator leads to nonlinearity even when the approximator is linearly parameterized.

We next introduce the projected Bellman equation, along with several important model-free algorithms that can be used to solve it.

Projected Bellman equation

Assume for now that X and U have a finite number of elements, $X = \{x_1, \dots, x_{\bar{N}}\}$, $U = \{u_1, \dots, u_{\bar{M}}\}$. Because the state space is finite, a transition model of the form (2.14) is appropriate, and the policy evaluation mapping T^h can be written as a sum (2.37), repeated here for easy reference:

$$[T^h(Q)](x,u) = \sum_{x'} \bar{f}(x,u,x') \left[\tilde{\rho}(x,u,x') + \gamma Q(x', h(x')) \right] \tag{3.33}$$

In the linearly parameterized case, an approximate Q-function $\widehat{Q}^h$ that has the form (3.3) is sought:

$$\widehat{Q}^h(x,u) = \phi^{\mathrm{T}}(x,u)\theta^h$$

where $\phi(x,u) = [\phi_1(x,u), \dots, \phi_n(x,u)]^{\mathrm{T}}$ is the vector of BFs and θ^h is the parameter vector. This approximate Q-function satisfies the following approximate version of

[7]Another important class of policy evaluation approaches aims to minimize the *Bellman error* (residual), which is the difference between the two sides of the Bellman equation (Baird, 1995; Antos et al., 2008b; Farahmand et al., 2009b). For instance, in the case of the Bellman equation for Q^h (3.31), the (quadratic) Bellman error is $\int_{X \times U} (\widehat{Q}^h(x,u) - [T^h(\widehat{Q}^h)](x,u))^2 \mathrm{d}(x,u)$. We choose to focus on projected policy evaluation instead, as this class of methods will be required later in the book.

the Bellman equation for Q^h (3.31), called the *projected Bellman equation*:[8]

$$\widehat{Q}^h = (P^w \circ T^h)(\widehat{Q}^h) \tag{3.34}$$

where P^w performs a weighted least-squares projection onto the space of representable (approximate) Q-functions, i.e., the space spanned by the BFs:

$$\left\{\phi^{\mathrm{T}}(x,u)\theta \mid \theta \in \mathbb{R}^n\right\}$$

The projection P^w is defined by:

$$\begin{aligned}
&[P^w(Q)](x,u) = \phi^{\mathrm{T}}(x,u)\theta^{\ddagger}, \text{ where} \\
&\theta^{\ddagger} \in \arg\min_{\theta} \sum_{(x,u)\in X\times U} w(x,u)\left(\phi^{\mathrm{T}}(x,u)\theta - Q(x,u)\right)^2
\end{aligned} \tag{3.35}$$

in which the weight function $w : X \times U \to [0,1]$ controls the distribution of the approximation error. The weight function is always interpreted as a probability distribution over the state-action space, so it must satisfy $\sum_{x,u} w(x,u) = 1$. For instance, the distribution given by w will later be used to generate the samples used by some model-free policy evaluation algorithms. Under appropriate conditions, the projected Bellman mapping $P^w \circ T^h$ is a contraction, and so the solution (fixed point) $\widehat{Q}^h$ of the projected Bellman equation exists and is unique (see Bertsekas (2007, Section 6.3) for a discussion of the conditions in the context of V-function approximation).

Figure 3.9 illustrates the projected Bellman equation.

Matrix form of the projected Bellman equation

We will now derive a matrix form of the projected Bellman equation, which is given in terms of the parameter vector. This form will be useful in the sequel, when developing algorithms to solve the projected Bellman equation. To introduce the matrix form, it will be convenient to refer to the state and the actions using explicit indices, e.g., x_i, u_j (recall that the states and actions were temporarily assumed to be discrete).

As a first step, the policy evaluation mapping (3.33) is written in matrix form $\boldsymbol{T}^h : \mathbb{R}^{\bar{N}\bar{M}} \to \mathbb{R}^{\bar{N}\bar{M}}$, as:

$$\boldsymbol{T}^h(\boldsymbol{Q}) = \tilde{\boldsymbol{\rho}} + \gamma \bar{\boldsymbol{f}} \boldsymbol{h} \boldsymbol{Q} \tag{3.36}$$

Denote by $[i,j]$ the scalar index corresponding to i and j, computed with $[i,j] = i + (j-1)\bar{N}$. The vectors and matrices in (3.36) are then defined as follows:[9]

[8]A multistep version of this equation can also be given. Instead of the (single-step) policy evaluation mapping T^h, this version uses the following multistep mapping, parameterized by the scalar $\lambda \in [0,1)$:

$$T^h_\lambda(Q) = (1-\lambda)\sum_{k=0}^{\infty} \lambda^k (T^h)^{k+1}(Q)$$

where $(T^h)^k$ denotes the k-times composition of T^h with itself, i.e., $T^h \circ T^h \circ \cdots \circ T^h$. In this chapter, as well as in the remainder of the book, we only consider the single-step case, i.e., the case in which $\lambda = 0$.

[9]Note that boldface notation is used for vector or matrix representations of functions and mappings. Ordinary vectors and matrices are displayed in normal font.

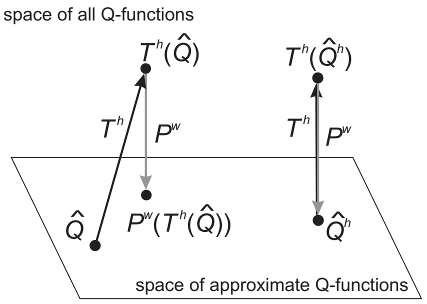


FIGURE 3.9
A conceptual illustration of the projected Bellman equation. Applying T^h and then P^w to an ordinary approximate Q-function $\widehat{Q}$ leads to a different point in the space of approximate Q-functions (left). In contrast, applying T^h and then P^w to the fixed point $\widehat{Q}^h$ of the projected Bellman equation leads back to the same point (right).

- $\boldsymbol{Q} \in \mathbb{R}^{\bar{N}\bar{M}}$ is a vector representation of Q, with $\boldsymbol{Q}_{[i,j]} = Q(x_i, u_j)$.
- $\tilde{\boldsymbol{\rho}} \in \mathbb{R}^{\bar{N}\bar{M}}$ is a vector representation of $\tilde{\rho}$, where the element $\tilde{\boldsymbol{\rho}}_{[i,j]}$ is the expected reward after taking action u_j in state x_i, i.e., $\tilde{\boldsymbol{\rho}}_{[i,j]} = \sum_{i'} \bar{f}(x_i, u_j, x_{i'}) \tilde{\rho}(x_i, u_j, x_{i'})$.
- $\bar{\boldsymbol{f}} \in \mathbb{R}^{\bar{N}\bar{M}\times\bar{N}}$ is a matrix representation of $\bar{f}$, with $\bar{\boldsymbol{f}}_{[i,j],i'} = \bar{f}(x_i, u_j, x_{i'})$. Here, $\bar{\boldsymbol{f}}_{[i,j],i'}$ denotes the element at row $[i, j]$ and column i' of matrix $\bar{\boldsymbol{f}}$.
- $\boldsymbol{h} \in \mathbb{R}^{\bar{N}\times\bar{N}\bar{M}}$ 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. Note that stochastic policies can easily be represented, by making $\boldsymbol{h}_{i,[i,j]}$ equal to the probability of taking u_j in x_i, and $\boldsymbol{h}_{i',[i,j]} = 0$ for all $i' \neq i$.

Consider now the setting of approximate policy evaluation. Define the BF matrix $\boldsymbol{\phi} \in \mathbb{R}^{\bar{N}\bar{M}\times n}$ and the diagonal weighting matrix $\boldsymbol{w} \in \mathbb{R}^{\bar{N}\bar{M}\times\bar{N}\bar{M}}$ by:

$$\boldsymbol{\phi}_{[i,j],l} = \phi_l(x_i, u_j)$$
$$\boldsymbol{w}_{[i,j],[i,j]} = w(x_i, u_j)$$

Using $\boldsymbol{\phi}$, the approximate Q-vector corresponding to a parameter θ is:

$$\widehat{\boldsymbol{Q}} = \boldsymbol{\phi}\theta$$

The projected Bellman equation (3.34) can now be written as follows:

$$\boldsymbol{P}^w \boldsymbol{T}^h(\widehat{\boldsymbol{Q}}^h) = \widehat{\boldsymbol{Q}}^h \tag{3.37}$$

where $\boldsymbol{P}^w$ is a matrix representation of the projection operator P^w, which can be written in a closed form (see, e.g., Lagoudakis and Parr, 2003a):

$$\boldsymbol{P}^w = \boldsymbol{\phi}(\boldsymbol{\phi}^{\mathrm{T}}\boldsymbol{w}\boldsymbol{\phi})^{-1}\boldsymbol{\phi}^{\mathrm{T}}\boldsymbol{w}$$

By substituting this closed-form expression for $\boldsymbol{P}^w$, the formula (3.36) for $\boldsymbol{T}^h$, and the expression $\widehat{\boldsymbol{Q}}^h = \boldsymbol{\phi}\, \theta^h$ for the approximate Q-vector into (3.37), we get:

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

Notice that this is a linear equation in the parameter vector θ^h. After a left-multiplication with $\boldsymbol{\phi}^{\mathrm{T}}\boldsymbol{w}$ and a rearrangement of the terms, we have:

$$\boldsymbol{\phi}^{\mathrm{T}}\boldsymbol{w}\boldsymbol{\phi}\,\theta^h = \gamma\boldsymbol{\phi}^{\mathrm{T}}\boldsymbol{w}\bar{\boldsymbol{f}}\boldsymbol{h}\boldsymbol{\phi}\,\theta^h + \boldsymbol{\phi}^{\mathrm{T}}\boldsymbol{w}\tilde{\boldsymbol{\rho}}$$

By introducing the matrices $\Gamma, \Lambda \in \mathbb{R}^{n\times n}$ and the vector $z \in \mathbb{R}^n$, given by:

$$\Gamma = \boldsymbol{\phi}^{\mathrm{T}}\boldsymbol{w}\boldsymbol{\phi}, \quad \Lambda = \boldsymbol{\phi}^{\mathrm{T}}\boldsymbol{w}\bar{\boldsymbol{f}}\boldsymbol{h}\boldsymbol{\phi}, \quad z = \boldsymbol{\phi}^{\mathrm{T}}\boldsymbol{w}\tilde{\boldsymbol{\rho}}$$

the projected Bellman equation can be written in the final, matrix form:

$$\Gamma\theta^h = \gamma\Lambda\theta^h + z \tag{3.38}$$

So, instead of the original, high-dimensional Bellman equation (3.31), approximate policy evaluation only needs to solve the low-dimensional system (3.38). A solution θ^h of this system can be employed to find an approximate Q-function using (3.3).

It can also be shown that matrices Γ, Λ and vector z can be written as sums of simpler matrices and vectors (e.g., Lagoudakis and Parr, 2003a):

$$\begin{aligned}
\Gamma &= \sum_{i=1}^{\bar{N}}\sum_{j=1}^{\bar{M}}\Big[\phi(x_i,u_j)w(x_i,u_j)\phi^{\mathrm{T}}(x_i,u_j)\Big]\\
\Lambda &= \sum_{i=1}^{\bar{N}}\sum_{j=1}^{\bar{M}}\left[\phi(x_i,u_j)w(x_i,u_j)\sum_{i'=1}^{\bar{N}}\Big(\bar{f}(x_i,u_j,x_{i'})\phi^{\mathrm{T}}(x_{i'},h(x_{i'}))\Big)\right]\\
z &= \sum_{i=1}^{\bar{N}}\sum_{j=1}^{\bar{M}}\left[\phi(x_i,u_j)w(x_i,u_j)\sum_{i'=1}^{\bar{N}}\Big(\bar{f}(x_i,u_j,x_{i'})\rho(x_i,u_j,x_{i'})\Big)\right]
\end{aligned} \tag{3.39}$$

To understand why the summation over i' enters the equation for z, recall that each element $\tilde{\boldsymbol{\rho}}_{[i,j]}$ of the vector $\tilde{\boldsymbol{\rho}}$ is the *expected* reward after taking action u_j in state x_i.

Model-free projected policy evaluation

Some of the most powerful algorithms for approximate policy evaluation solve the matrix form (3.38) of the projected Bellman equation in a model-free fashion, by estimating Γ, Λ, and z from transition samples. Because (3.38) is a linear system of equations, these algorithms are computationally efficient. They are also sample-efficient, i.e., they approach their solution quickly as the number of samples they consider increases, as shown in the context of V-function approximation by Konda (2002, Chapter 6) and by Yu and Bertsekas (2006, 2009).

Consider a set of transition samples:

$$\{(x_{l_s}, u_{l_s}, x'_{l_s} \sim \bar{f}(x_{l_s}, u_{l_s}, \cdot), r_{l_s} = \tilde{\rho}(x_{l_s}, u_{l_s}, x'_{l_s}))\,|\, l_s = 1, \ldots, n_s\}$$

This set is constructed by drawing state-action samples (x,u) from a distribution given by the weight function w: the probability of each pair (x,u) is equal to its weight $w(x,u)$. Using this set of samples, estimates of Γ, Λ, and z can be constructed as follows:

$$\begin{aligned}
&\Gamma_0 = 0, \quad \Lambda_0 = 0, \quad z_0 = 0 \\
&\Gamma_{l_s} = \Gamma_{l_s-1} + \phi(x_{l_s}, u_{l_s})\phi^{\mathrm{T}}(x_{l_s}, u_{l_s}) \\
&\Lambda_{l_s} = \Lambda_{l_s-1} + \phi(x_{l_s}, u_{l_s})\phi^{\mathrm{T}}(x'_{l_s}, h(x'_{l_s})) \\
&z_{l_s} = z_{l_s-1} + \phi(x_{l_s}, u_{l_s}) r_{l_s}
\end{aligned} \tag{3.40}$$

These updates can be derived from (3.39).

The *least-squares temporal difference for Q-functions (LSTD-Q)* (Lagoudakis et al., 2002; Lagoudakis and Parr, 2003a) is a policy evaluation algorithm that processes the samples using (3.40) and then solves the equation:

$$\frac{1}{n_s}\Gamma_{n_s}\widehat{\theta}^h = \gamma \frac{1}{n_s}\Lambda_{n_s}\widehat{\theta}^h + \frac{1}{n_s} z_{n_s} \tag{3.41}$$

to find an approximate parameter vector $\widehat{\theta}^h$. Notice that $\widehat{\theta}^h$ appears on both sides of (3.41), so this equation can be simplified to:

$$\frac{1}{n_s}(\Gamma_{n_s} - \gamma\Lambda_{n_s})\widehat{\theta}^h = \frac{1}{n_s} z_{n_s}$$

Although the division by n_s is not necessary from a formal point of view, it helps to increase the numerical stability of the algorithm (the elements in the Γ_{n_s}, Λ_{n_s}, z_{n_s} can be very large when n_s is large). LSTD-Q is an extension of an earlier, similar algorithm for V-functions, called least-squares temporal difference (Bradtke and Barto, 1996; Boyan, 2002).

Another method, the *least-squares policy evaluation for Q-functions (LSPE-Q)* (e.g., Jung and Polani, 2007a) starts with an arbitrary initial parameter vector θ_0 and updates it incrementally, with:

$$\begin{aligned}
&\theta_{l_s} = \theta_{l_s-1} + \alpha(\theta^{\ddagger}_{l_s} - \theta_{l_s-1}), \text{ where:} \\
&\frac{1}{l_s}\Gamma_{l_s}\theta^{\ddagger}_{l_s} = \gamma\frac{1}{l_s}\Lambda_{l_s}\theta_{l_s-1} + \frac{1}{l_s} z_{l_s}
\end{aligned} \tag{3.42}$$

in which α is a step size parameter. To ensure the invertibility of the matrix Γ at the start of the learning process, when only a few samples have been processed, it can be initialized to a small multiple of the identity matrix. The division by l_s increases the numerical stability of the updates. Like LSTD-Q, LSPE-Q is an extension of an earlier algorithm for V-functions, called least-squares policy evaluation (LSPE) (Bertsekas and Ioffe, 1996).

Algorithms 3.8 and 3.9 present LSTD-Q and LSPE-Q in a procedural form. LSTD-Q is a one-shot algorithm, and the parameter vector it computes does not depend on the order in which the samples are processed. On the other hand, LSPE-Q

ALGORITHM 3.8 Least-squares temporal difference for Q-functions.

```
Input: policy $h$ to be evaluated, discount factor $\gamma$,
    BFs $\phi_1,\dots,\phi_n : X \times U \to \mathbb{R}$, samples $\{(x_{l_s}, u_{l_s}, x'_{l_s}, r_{l_s}) \mid l_s = 1,\dots,n_s\}$
1: $\Gamma_0 \leftarrow 0$, $\Lambda_0 \leftarrow 0$, $z_0 \leftarrow 0$
2: for $l_s = 1,\dots,n_s$ do
3:     $\Gamma_{l_s} \leftarrow \Gamma_{l_s-1} + \phi(x_{l_s}, u_{l_s})\phi^T(x_{l_s}, u_{l_s})$
4:     $\Lambda_{l_s} \leftarrow \Lambda_{l_s-1} + \phi(x_{l_s}, u_{l_s})\phi^T(x'_{l_s}, h(x'_{l_s}))$
5:     $z_{l_s} \leftarrow z_{l_s-1} + \phi(x_{l_s}, u_{l_s}) r_{l_s}$
6: end for
7: solve $\frac{1}{n_s}\Gamma_{n_s}\widehat{\theta}^h = \gamma\frac{1}{n_s}\Lambda_{n_s}\widehat{\theta}^h + \frac{1}{n_s}z_{n_s}$ for $\widehat{\theta}^h$
Output: $\widehat{\theta}^h$
```

ALGORITHM 3.9 Least-squares policy evaluation for Q-functions.

```
Input: policy $h$ to be evaluated, discount factor $\gamma$,
    BFs $\phi_1,\dots,\phi_n : X \times U \to \mathbb{R}$, samples $\{(x_{l_s}, u_{l_s}, x'_{l_s}, r_{l_s}) \mid l_s = 1,\dots,n_s\}$,
    step size $\alpha$, a small constant $\beta_\Gamma > 0$
1: $\Gamma_0 \leftarrow \beta_\Gamma I$, $\Lambda_0 \leftarrow 0$, $z_0 \leftarrow 0$
2: for $l_s = 1,\dots,n_s$ do
3:     $\Gamma_{l_s} \leftarrow \Gamma_{l_s-1} + \phi(x_{l_s}, u_{l_s})\phi^T(x_{l_s}, u_{l_s})$
4:     $\Lambda_{l_s} \leftarrow \Lambda_{l_s-1} + \phi(x_{l_s}, u_{l_s})\phi^T(x'_{l_s}, h(x'_{l_s}))$
5:     $z_{l_s} \leftarrow z_{l_s-1} + \phi(x_{l_s}, u_{l_s}) r_{l_s}$
6:     $\theta_{l_s} \leftarrow \theta_{l_s-1} + \alpha(\theta^\ddagger_{l_s} - \theta_{l_s-1})$, where $\frac{1}{l_s}\Gamma_{l_s}\theta^\ddagger_{l_s} = \gamma\frac{1}{l_s}\Lambda_{l_s}\theta_{l_s-1} + \frac{1}{l_s}z_{l_s}$
7: end for
Output: $\widehat{\theta}^h = \theta_{n_s}$
```

is an incremental algorithm, so the current parameter vector θ_{l_s} depends on the previous values $\theta_0,\dots,\theta_{l_s-1}$, and therefore the order in which samples are processed is important.

In the context of V-function approximation, such least-squares algorithms have been shown to converge to the fixed point of the projected Bellman equation, namely by Nedić and Bertsekas (2003) for the V-function analogue of LSTD-Q, and by Nedić and Bertsekas (2003); Bertsekas et al. (2004) for the analogue of LSPE-Q. These results also extend to Q-function approximation. To ensure convergence, the weight (probability of being sampled) $w(x,u)$ of each state-action pair (x,u) should be equal to the steady-state probability of this pair along an infinitely-long trajectory generated with the policy h.[10]

Note that collecting samples by using only a *deterministic* policy h is insuffi-

[10]From a practical point of view, note that LSTD-Q is a one-shot algorithm and will produce a solution whenever Γ_{l_s} is invertible. This means the experimenter need not worry excessively about divergence *per se*. Rather, the theoretical results concern the uniqueness and meaning of the solution obtained. LSTD-Q can, in fact, produce meaningful results for many weight functions w, as we illustrate later in Section 3.5.7 and in Chapter 5.

cient for the following reason. If only state-action pairs of the form $(x, h(x))$ were collected, no information about pairs (x, u) with $u \neq h(x)$ would be available (equivalently, the corresponding weights $w(x, u)$ would all be zero). As a result, the approximate Q-values of such pairs would be poorly estimated and could not be relied upon for policy improvement. To alleviate this problem, *exploration* is necessary: sometimes, actions different from $h(x)$ have to be selected, e.g., in a random fashion. Given a stationary (time-invariant) exploration procedure, LSTD-Q and LSPE-Q are simply evaluating the new, exploratory policy, and so they remain convergent.

The following intuitive (albeit informal) line of reasoning is useful to understand the convergence of LSTD-Q and LSPE-Q. Asymptotically, as $n_\mathrm{s} \to \infty$, it is true that $\frac{1}{n_\mathrm{s}}\Gamma_{n_\mathrm{s}} \to \Gamma$, $\frac{1}{n_\mathrm{s}}\Lambda_{n_\mathrm{s}} \to \Lambda$, and $\frac{1}{n_\mathrm{s}}z_{n_\mathrm{s}} \to z$, for the following two reasons. First, as the number n_s of state-action samples generated grows, their empirical distribution converges to w. Second, as the number of transition samples involving a given state-action pair (x, u) grows, the empirical distribution of the next states x' converges to the distribution $\bar{f}(x, u, \cdot)$, and the empirical average of the rewards converges to its expected value, given x and u.

Since the estimates of Γ, Λ, and z asymptotically converge to their true values, the equation solved by LSTD-Q asymptotically converges to the projected Bellman equation (3.38). Under the assumptions for convergence, this equation has a unique solution θ^h, so the parameter vector of LSTD-Q asymptotically reaches this solution. For similar reasons, whenever it converges, LSPE-Q asymptotically becomes equivalent to LSTD-Q and the projected Bellman equation. Therefore, if LSPE-Q converges, it must in fact converge to θ^h. In fact, it can additionally be shown that, as n_s grows, the solutions of LSTD-Q and LSPE-Q converge to each other faster than they converge to their limit θ^h. This was proven in the context of V-function approximation by Yu and Bertsekas (2006, 2009).

One possible advantage of LSTD-Q over LSPE-Q may arise when their assumptions are violated, e.g., when the policy to be evaluated changes as samples are being collected. This situation can arise in the important context of optimistic policy iteration, which will be discussed in Section 3.5.5. Violating the assumptions may introduce instability and possibly divergence in the iterative LSPE-Q updates (3.42). In contrast, because it only computes one-shot solutions, LSTD-Q (3.41) may be more resilient to such instabilities. On the other hand, the incremental nature of LSPE-Q offers some advantages over LSTD-Q. For instance, LSPE-Q can benefit from a good initial value of the parameter vector. Additionally, by lowering the step size α, it may be possible to mitigate the destabilizing effects of violating the assumptions. Note that an incremental version of LSTD-Q can also be given, but the benefits of this version are unclear.

While for the derivation above it was assumed that X and U are finite, the updates (3.40), together with LSTD-Q and LSPE-Q, can also be applied without any change in infinite and uncountable (e.g., continuous) state-action spaces.

From a computational point of view, the linear systems in (3.41) and (3.42) can be solved in several ways, e.g., by matrix inversion, by Gaussian elimination, or by incrementally computing the inverse with the Sherman-Morrison formula. The computational cost is $\mathrm{O}(n^3)$ for "naive" matrix inversion. More efficient algorithms than matrix inversion can be obtained, e.g., by incrementally computing the inverse, but the cost of solving the linear system will still be larger than $O(n^2)$. In an effort to further reduce the computational costs, variants of the least-squares temporal difference have been proposed in which only a few of the parameters are updated at a given iteration (Geramifard et al., 2006, 2007). Note also that, when the BF vector $\phi(x,u)$ is sparse, the computational efficiency of the updates (3.40) can be improved by exploiting this sparsity.[11]

As already outlined, analogous least-squares algorithms can be given to compute approximate V-functions (Bertsekas and Ioffe, 1996; Bradtke and Barto, 1996; Boyan, 2002; Bertsekas, 2007, Chapter 6). However, as explained in Section 2.2, policy improvement is more difficult to perform using V-functions. Namely, a model of the MDP is required, and in the stochastic case, expectations over the transitions must be estimated.

Gradient-based policy evaluation

Gradient-based algorithms for policy evaluation historically precede the least-squares methods discussed above (Sutton, 1988). However, under appropriate conditions, they find, in fact, a solution of the projected Bellman equation (3.34). These algorithms are called temporal-difference learning in the literature, and are more popular in the context of V-function approximation (Sutton, 1988; Jaakkola et al., 1994; Tsitsiklis and Van Roy, 1997). Nevertheless, given the focus of this chapter, we will present gradient-based policy evaluation for the case of Q-function approximation.

We use SARSA as a starting point in developing such an algorithm. Recall that SARSA (Algorithm 2.7) uses tuples $(x_k, u_k, r_{k+1}, x_{k+1}, u_{k+1})$ to update a Q-function online (2.40):

$$Q_{k+1}(x_k,u_k) = Q_k(x_k,u_k) + \alpha_k[r_{k+1} + \gamma Q_k(x_{k+1},u_{k+1}) - Q_k(x_k,u_k)] \tag{3.43}$$

where α_k is the learning rate. When u_k is chosen according to a fixed policy h, SARSA actually performs policy evaluation (see also Section 2.4.2). We exploit this property and combine (3.43) with gradient-based updates to obtain the desired policy evaluation algorithm. As before, linearly parameterized approximation is considered. By a derivation similar to that given for gradient-based Q-learning in Section 3.4.2, the following update rule is obtained:

$$\theta_{k+1} = \theta_k + \alpha_k \left[r_{k+1} + \gamma \phi^{\mathrm{T}}(x_{k+1},u_{k+1})\theta_k - \phi^{\mathrm{T}}(x_k,u_k)\theta_k\right] \phi(x_k,u_k) \tag{3.44}$$

where the quantity in square brackets is an approximation of the temporal difference. The resulting algorithm for policy evaluation is called *temporal difference for Q-functions (TD-Q)* . Note that TD-Q can be seen as an extension of a corresponding algorithm for V-functions, which is called temporal difference (TD) (Sutton, 1988).

Like the least-squares algorithms presented earlier, TD-Q requires exploration to

[11]The BF vector is sparse, e.g., for the discrete-action approximator described in Example 3.1. This is because the BF vector contains zeros for all the discrete actions that are different from the current discrete action.

obtain samples (x,u) with $u \neq h(x)$. Algorithm 3.10 presents TD-Q with ε-greedy exploration. In this algorithm, because the update at step k involves the action u_{k+1} at the next step, this action is chosen prior to updating the parameter vector.

ALGORITHM 3.10 Temporal difference for Q-functions, with ε-greedy exploration.

```
Input: discount factor $\gamma$, policy $h$ to be evaluated,
    BFs $\phi_1, \dots, \phi_n : X \times U \to \mathbb{R}$,
    exploration schedule $\{\varepsilon_k\}_{k=0}^{\infty}$, learning rate schedule $\{\alpha_k\}_{k=0}^{\infty}$
1: initialize parameter vector, e.g., $\theta_0 \leftarrow 0$
2: measure initial state $x_0$
3: $u_0 \leftarrow \begin{cases} h(x_0) & \text{with probability } 1-\varepsilon_0 \\ \text{a uniform random action in } U & \text{with probability } \varepsilon_0 \text{ (explore)} \end{cases}$
4: for every time step $k = 0, 1, 2, \dots$ do
5:     apply $u_k$, measure next state $x_{k+1}$ and reward $r_{k+1}$
6:     $u_{k+1} \leftarrow \begin{cases} h(x_{k+1}) & \text{with probability } 1-\varepsilon_{k+1} \\ \text{a uniform random action in } U & \text{with probability } \varepsilon_{k+1} \end{cases}$
7:     $\theta_{k+1} \leftarrow \theta_k + \alpha_k \left[ r_{k+1} + \gamma \phi^{\mathrm{T}}(x_{k+1}, u_{k+1})\theta_k - \phi^{\mathrm{T}}(x_k, u_k)\theta_k \right] \phi(x_k, u_k)$
8: end for
```

A comprehensive convergence analysis of gradient-based policy evaluation was provided by Tsitsiklis and Van Roy (1997) in the context of V-function approximation. This analysis extends to Q-function approximation under appropriate conditions. An important condition is that the stochastic policy $\tilde{h}$ resulting from the combination of h with exploration should be time-invariant, which can be achieved by simply making the exploration time-invariant, e.g., in the case of ε-greedy exploration, by making ε_k the same for all steps k. The main result is that TD-Q asymptotically converges to the solution of the projected Bellman equation for the exploratory policy $\tilde{h}$, for a weight function given by the steady-state distribution of the state-action pairs under $\tilde{h}$.

Gradient-based algorithms such as TD-Q are less computationally demanding than least-squares algorithms such as LSTD-Q and LSPE-Q. The time and memory complexity of TD-Q are both $\mathrm{O}(n)$, since they store and update vectors of length n. The memory complexity of LSTD-Q and LSPE-Q is at least $\mathrm{O}(n^2)$ (since they store matrices of size n) and their time complexity is $\mathrm{O}(n^3)$ (when "naive" matrix inversion is used to solve the linear system). On the other hand, gradient-based algorithms typically require more samples than least-squares algorithms to achieve a similar accuracy (Konda, 2002; Yu and Bertsekas, 2006, 2009), and are more sensitive to the learning rate (step size) schedule. LSTD-Q has no step size at all, and LSPE-Q works for a wide range of constant step sizes, as shown in the context of V-functions by Bertsekas et al. (2004) (this range includes $\alpha = 1$, leading to a nonincremental variant of LSPE-Q).

Efforts have been made to extend gradient-based policy evaluation algorithms to off-policy learning, i.e., evaluating one policy while using another policy to generate the samples (Sutton et al., 2009b,a). These extensions perform gradient descent on error measures that are different from the measure used in the basic temporal-difference algorithms such as TD-Q (i.e., different from the squared value function error for the current sample).

3.5.3 Policy evaluation with nonparametric approximation

Nonparametric approximators have been combined with a number of algorithms for approximate policy evaluation. For instance, kernel-based approximators were combined with LSTD by Xu et al. (2005), with LSTD-Q by Xu et al. (2007); Jung and Polani (2007b); Farahmand et al. (2009b), and with LSPE-Q by Jung and Polani (2007a,b). Rasmussen and Kuss (2004) and Engel et al. (2003, 2005) used the related framework of Gaussian processes to approximate V-functions in policy evaluation. Taylor and Parr (2009) showed that, in fact, the algorithms in (Rasmussen and Kuss, 2004; Engel et al., 2005; Xu et al., 2005) produce the same solution when they use the same samples and the same kernel function. Fitted policy evaluation (Algorithm 3.7) can be extended to the nonparametric case along the same lines as fitted Q-iteration in Section 3.4.3. Such an algorithm was proposed by Jodogne et al. (2006), who employed ensembles of extremely randomized trees to approximate the Q-function.

As explained in Section 3.3.2, a kernel-based approximator can be seen as linearly parameterized if all the samples are known in advance. In certain cases, this property can be exploited to extend the theoretical guarantees about approximate policy evaluation from the parametric case to the nonparametric case (Xu et al., 2007). Farahmand et al. (2009b) provided performance guarantees for their kernel-based LSTD-Q variant for the case when only a finite number of samples is available.

An important concern in the nonparametric case is controlling the complexity of the approximator. Originally, the computational demands of many nonparametric approximators, including kernel-based methods and Gaussian processes, grow with the number of samples considered. Many of the approaches mentioned above employ kernel sparsification techniques to limit the number of samples that contribute to the solution (Xu et al., 2007; Engel et al., 2003, 2005; Jung and Polani, 2007a,b).

3.5.4 Model-based approximate policy evaluation with rollouts

All the policy evaluation algorithms discussed above obtain a value function by solving the Bellman equation (3.31) approximately. While this is a powerful approach, it also has its drawbacks. A core problem is that a good value function approximator is required, which is often difficult to find. Nonparametric approximation alleviates this problem to some extent. Another problem is that the convergence requirements of the algorithms, such as the linearity of the approximate Q-function in the parameters, can sometimes be too restrictive.

Another class of policy evaluation approaches sidesteps these difficulties by avoiding an explicit representation of the value function. Instead, the value function is evaluated on demand, by Monte Carlo simulations. A model is required to perform the simulations, so these approaches are model-based. For instance, to estimate the Q-value $\widehat{Q}^h(x,u)$ of a given state-action pair (x,u), a number N_{MC} of trajectories are simulated, where each trajectory is generated using the policy h, has length K, and starts from the pair (x,u). The estimated Q-value is then the average of the sample returns obtained along these trajectories:

$$\widehat{Q}^h(x,u) = \frac{1}{N_{\mathrm{MC}}} \sum_{i_0=1}^{N_{\mathrm{MC}}} \left[\tilde{\rho}(x,u,x_{i_0,1}) + \sum_{k=1}^{K} \gamma^k \tilde{\rho}(x_{i_0,k}, h(x_{i_0,k}), x_{i_0,k+1}) \right] \tag{3.45}$$

where N_{MC} is the number of trajectories to simulate. For each trajectory i_0, the first state-action pair is fixed to (x,u) and leads to a next state $x_{i_0,1} \sim \tilde{f}(x,u,\cdot)$. Thereafter, actions are chosen using the policy h, which means that for $k \geq 1$:

$$x_{i_0,k+1} \sim f(x_{i_0,k}, h(x_{i_0,k}), \cdot)$$

Such a simulation-based estimation procedure is called a *rollout* (Lagoudakis and Parr, 2003b; Bertsekas, 2005b; Dimitrakakis and Lagoudakis, 2008). The length K of the trajectories can be chosen using (2.41) to ensure $\varepsilon_{\mathrm{MC}}$-accurate returns, where $\varepsilon_{\mathrm{MC}} > 0$. Note that if the MDP is deterministic, a single trajectory suffices. In the stochastic case, an appropriate value for the number N_{MC} of trajectories will depend on the problem.

Rollouts can be computationally expensive, especially in the stochastic case. Their computational cost is proportional to the number of points at which the value function must be evaluated. Therefore, rollouts are most beneficial when this number is small. If the value function must be evaluated at many (or all) points of the state(-action) space, then methods that solve the Bellman equation approximately (Sections 3.5.1 – 3.5.3) may be computationally less costly than rollouts.

3.5.5 Policy improvement and approximate policy iteration

Up to this point, approximate policy evaluation has been considered. To obtain a complete algorithm for approximate policy iteration, a method to perform policy improvement is also required.

Exact and approximate policy improvement

Consider first policy improvement in the case where the policy is not represented explicitly. Instead, greedy actions are computed on demand from the value function, for every state where a control action is required. For instance, when Q-functions are employed, an improved action for the state x can be found with:

$$h_{\ell+1}(x) = u, \text{ where } u \in \arg\max_{\bar{u}} \widehat{Q}^{h_\ell}(x,\bar{u}) \tag{3.46}$$

The policy is thus implicitly defined by the value function. In (3.46), it was assumed that a greedy action can be computed exactly. This is true, e.g., when the action space only contains a small, discrete set of actions, and the maximization in the policy improvement step is solved by enumeration. In this situation, policy improvement is exact, but if greedy actions cannot be computed exactly, then the result of the maximization is approximate, and the (implicitly defined) policy thus becomes an approximation.

Alternatively, the policy can also be represented explicitly, in which case it generally must be approximated. The policy can be approximated, e.g., by a linear parametrization (3.12):

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

where $\varphi_i(x)$, $i = 1, \dots, \mathcal{N}$ are the state-dependent BFs and ϑ is the policy parameter vector (see Section 3.3.4 for a discussion of the notation used for policy approximation). A scalar action was assumed, but the parametrization can easily be extended to multiple action variables. For this parametrization, approximate policy improvement can be performed by solving the linear least-squares problem:

$$\vartheta_{\ell+1} = \vartheta^{\ddagger}, \text{ where } \vartheta^{\ddagger} \in \arg\min_{\vartheta} \sum_{i_{\mathrm{s}}=1}^{\mathcal{N}_{\mathrm{s}}} \left(\varphi^{\mathrm{T}}(x_{i_{\mathrm{s}}})\vartheta - u_{i_{\mathrm{s}}}\right)^2 \tag{3.47}$$

to find a parameter vector $\vartheta_{\ell+1}$, where $\{x_1, \dots, x_{\mathcal{N}_{\mathrm{s}}}\}$ is a set of state samples to be used for policy improvement, and $u_1, \dots, u_{\mathcal{N}_{\mathrm{s}}}$ are corresponding greedy actions:

$$u_{i_{\mathrm{s}}} \in \arg\max_{u} \widehat{Q}^{\widehat{h}_\ell}(x_{i_{\mathrm{s}}}, u) \tag{3.48}$$

Note that the previous policy $\widehat{h}_\ell$ is now also an approximation. In (3.48), it was implicitly assumed that greedy actions can be computed exactly; if this is not the case, then $u_{i_{\mathrm{s}}}$ will only be approximations of the true greedy actions.

Such a policy improvement is therefore a two-step procedure: first, greedy actions $u_{i_{\mathrm{s}}}$ are chosen using (3.48), and then these actions are used to solve the least-squares problem (3.47). The solution depends on the greedy actions chosen, but remains meaningful for any combination of choices, since for any such combination, it approximates one of the possible greedy policies in the Q-function.

Alternatively, policy improvement could be performed with:

$$\vartheta_{\ell+1} = \vartheta^{\ddagger}, \text{ where } \vartheta^{\ddagger} \in \arg\max_{\vartheta} \sum_{i_{\mathrm{s}}=1}^{\mathcal{N}_{\mathrm{s}}} \widehat{Q}^{\widehat{h}_\ell}(x_{i_{\mathrm{s}}}, \varphi^{\mathrm{T}}(x_{i_{\mathrm{s}}})\vartheta) \tag{3.49}$$

which maximizes the approximate Q-values of the actions chosen by the policy in the state samples. However, (3.49) is generally a difficult nonlinear optimization problem, whereas (3.47) is (once greedy actions have been chosen) a convex optimization problem, which is easier to solve.

More generally, for any policy representation (e.g., for a nonlinear parametrization), a regression problem generalizing either (3.47) or (3.49) must be solved to perform policy improvement.

Offline approximate policy iteration

Approximate policy iteration algorithms can be obtained by combining a policy evaluation algorithm (e.g., one of those described in Sections 3.5.1 – 3.5.3) with a policy improvement technique (e.g., one of those described above); see again Algorithm 3.5 for a generic template of approximate policy iteration. In the offline case, the approximate policy evaluation is run until (near) convergence, to ensure the accuracy of the value function and therefore an accurate policy improvement.

For example, the algorithm resulting from combining LSTD-Q (Algorithm 3.8) with exact policy improvement is called *least-squares policy iteration (LSPI)*. LSPI was proposed by Lagoudakis et al. (2002) and by Lagoudakis and Parr (2003a), and has been studied often since then (e.g., Mahadevan and Maggioni, 2007; Xu et al., 2007; Farahmand et al., 2009b). Algorithm 3.11 shows LSPI, in a simple variant that uses the same set of transition samples at every policy evaluation. In general, different sets of samples can be used at different iterations. The explicit policy improvement at line 4 is included for clarity. In practice, the policy $h_{\ell+1}$ does not have to be computed and stored for every state. Instead, it is computed on demand from the current Q-function, only for those states where an improved action is necessary. In particular, LSTD-Q only evaluates the policy at the state samples x'_{l_s}.

ALGORITHM 3.11 Least-squares policy iteration.

```
Input: discount factor $\gamma$,
    BFs $\phi_1, \dots, \phi_n : X \times U \to \mathbb{R}$, samples $\{(x_{l_s}, u_{l_s}, x'_{l_s}, r_{l_s}) \,|\, l_s = 1, \dots, n_s\}$
1: initialize policy $h_0$
2: repeat at every iteration $\ell = 0, 1, 2, \dots$
3:     evaluate $h_\ell$ using LSTD-Q (Algorithm 3.8), yielding $\theta_\ell$   ▷ policy evaluation
4:     $h_{\ell+1}(x) \leftarrow u$, $u \in \arg\max_{\bar{u}} \phi^{\mathrm{T}}(x, \bar{u})\theta_\ell$ for each $x \in X$   ▷ policy improvement
5: until $h_{\ell+1}$ is satisfactory
Output: $\widehat{h}^* = h_{\ell+1}$
```

Policy iteration with rollout policy evaluation (Section 3.5.4) was studied, e.g., by Lagoudakis and Parr (2003b) and by Dimitrakakis and Lagoudakis (2008), who employed nonparametric approximation to represent the policy. Note that rollout policy evaluation (which represents value functions implicitly) should not be combined with implicit policy improvement. Such an algorithm would be impractical, because neither the value function nor the policy would be represented explicitly.

Online, optimistic approximate policy iteration

In online learning, the performance should improve once every few transition samples. This is in contrast to the offline case, in which only the performance at the end of the learning process is important. One way in which policy iteration can take this requirement into account is by performing policy improvements once every few transition samples, before an accurate evaluation of the current policy can be completed. Such a variant is sometimes called *optimistic* policy iteration (Bertsekas and Tsitsiklis, 1996, Section 6.4; Sutton 1988; Tsitsiklis 2002). In the extreme case, the policy is improved after every transition, and then applied to obtain a new transition sample that is fed into the policy evaluation algorithm. Then, another policy improvement takes place, and the cycle repeats. This variant is called fully optimistic. In general, the policy is improved once every several (but not too many) transitions; this variant is partially optimistic. As in any online RL algorithm, exploration is also necessary in optimistic policy iteration.

Optimistic policy iteration was already outlined in Section 2.4.2, where it was also explained that SARSA (Algorithm 2.7) belongs to this class. So, an approximate version of SARSA will naturally be optimistic, as well. A gradient-based version of SARSA can be easily obtained from TD-Q (Algorithm 3.10), by choosing actions with a policy that is greedy in the current Q-function, instead of with a fixed policy as in TD-Q. Of course, exploration is required in addition to greedy action selection. Algorithm 3.12 presents approximate SARSA with an ε-greedy exploration procedure. Approximate SARSA has been studied, e.g., by Sutton (1996); Santamaria et al. (1998); Gordon (2001); Melo et al. (2008).

ALGORITHM 3.12 SARSA with a linear parametrization and ε-greedy exploration.

```
Input: discount factor $\gamma$,
    BFs $\phi_1, \dots, \phi_n : X \times U \to \mathbb{R}$,
    exploration schedule $\{\varepsilon_k\}_{k=0}^{\infty}$, learning rate schedule $\{\alpha_k\}_{k=0}^{\infty}$
1: initialize parameter vector, e.g., $\theta_0 \leftarrow 0$
2: measure initial state $x_0$
3: $u_0 \leftarrow \begin{cases} u \in \arg\max_{\bar{u}} \left(\phi^{\mathrm{T}}(x_0, \bar{u})\theta_0\right) & \text{with probability } 1-\varepsilon_0 \text{ (exploit)} \\ \text{a uniform random action in } U & \text{with probability } \varepsilon_0 \text{ (explore)} \end{cases}$
4: for every time step $k = 0, 1, 2, \dots$ do
5:     apply $u_k$, measure next state $x_{k+1}$ and reward $r_{k+1}$
6:     $u_{k+1} \leftarrow \begin{cases} u \in \arg\max_{\bar{u}} \left(\phi^{\mathrm{T}}(x_{k+1}, \bar{u})\theta_k\right) & \text{with probability } 1-\varepsilon_{k+1} \\ \text{a uniform random action in } U & \text{with probability } \varepsilon_{k+1} \end{cases}$
7:     $\theta_{k+1} \leftarrow \theta_k + \alpha_k \left[r_{k+1} + \gamma\phi^{\mathrm{T}}(x_{k+1}, u_{k+1})\theta_k - \phi^{\mathrm{T}}(x_k, u_k)\theta_k\right] \phi(x_k, u_k)$
8: end for
```

Other policy evaluation algorithms can also be used in optimistic policy iteration. For instance, optimistic policy iteration with LSPE-Q was applied by Jung and Polani (2007a,b), while a V-function based algorithm similar to approximate SARSA was proposed by Jung and Uthmann (2004). In Chapter 5 of this book, an online, optimistic variant of LSPI will be introduced in detail and evaluated experimentally.

3.5.6 Theoretical guarantees

Under appropriate assumptions, *offline* policy iteration eventually produces policies with a bounded suboptimality. However, in general it cannot be guaranteed to converge to a fixed policy. The theoretical understanding of *optimistic* policy iteration is currently limited, and guarantees can only be provided in some special cases. We first discuss the properties of policy iteration in the offline setting, and then continue to the online, optimistic setting.

Theoretical guarantees for offline approximate policy iteration

As long as the policy evaluation and improvement errors are bounded, offline approximate policy iteration eventually produces policies with a bounded suboptimality. This result applies to any type of value function or policy approximator, and can be formalized as follows.

Consider the general case where both the value functions and the policies are approximated. Consider also the case where Q-functions are used, and assume that the error at every policy evaluation step is bounded by ς_Q:

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

and that the error at every policy improvement step is bounded by ς_h, in the following sense:

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

where $T^{\widehat{h}_{\ell+1}}$ is the policy evaluation mapping for the improved (approximate) policy, and T is the Q-iteration mapping (2.22). Then, approximate policy iteration eventually produces policies with performances that lie within a bounded distance from the optimal performance (e.g., Lagoudakis and Parr, 2003a):

$$\limsup_{\ell\to\infty} \left\|\widehat{Q}^{\widehat{h}_\ell} - Q^*\right\|_\infty \leq \frac{\varsigma_h + 2\gamma\varsigma_Q}{(1-\gamma)^2} \tag{3.50}$$

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

$$\limsup_{\ell\to\infty} \|\widehat{Q}^{h_\ell} - Q^*\|_\infty \leq \frac{2\gamma\varsigma_Q}{(1-\gamma)^2} \tag{3.51}$$

where $\|\widehat{Q}^{h_\ell} - Q^{h_\ell}\|_\infty \leq \varsigma_Q$, for any $\ell \geq 0$. Note that finding ς_Q and (when approximate policies are used) ς_h may be difficult in practice, and the existence of these bounds may require additional assumptions.

These guarantees do not necessarily imply the convergence to a fixed policy. 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. Convergence to limit cycles can indeed happen, as will be seen in the upcoming example of Section 3.5.7. Similarly, when exact policy improvements are used, the value function parameter may oscillate, implicitly leading to an oscillating policy. This is a disadvantage with respect to offline approximate value iteration, which under appropriate assumptions converges monotonically to a unique fixed point (Section 3.4.4).

Similar results hold when V-functions are used instead of Q-functions (Bertsekas and Tsitsiklis, 1996, Section 6.2).

Theoretical guarantees for online, optimistic policy iteration

The performance guarantees given above for offline policy iteration rely on bounded policy evaluation errors. Because optimistic policy iteration improves the policy before an accurate 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.

The behavior of optimistic policy iteration has not been properly understood yet, and can be very complicated. Optimistic policy iteration 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, Section 6.4).

Theoretical guarantees can, however, be provided in certain special cases. Gordon (2001) showed that the parameter vector of approximate SARSA cannot diverge when the MDP has terminal states and the policy is only improved in-between trials (see Section 2.2.1 for the meaning of terminal states and trials). Melo et al. (2008) improved on this result, by showing that approximate SARSA converges with probability 1 to a fixed point, if the dependence of the policy on the parameter vector satisfies a certain Lipschitz continuity condition. This condition prohibits using fully greedy policies, because those generally depend on the parameters in a discontinuous fashion.

These theoretical results concern the gradient-based SARSA algorithm. However, in practice, least-squares algorithms may be preferable due to their improved sample efficiency. While no theoretical guarantees are available when using least-squares algorithms in the optimistic setting, some promising empirical results have been reported (Jung and Polani, 2007a,b); see also Chapter 5 for an empirical evaluation of optimistic LSPI.

3.5.7 Example: Least-squares policy iteration for a DC motor

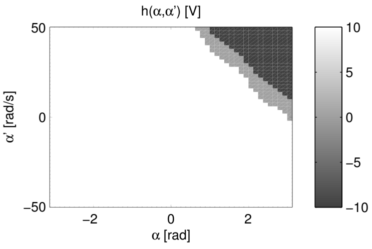
In this example, approximate policy iteration is applied to the DC motor problem introduced in Section 3.4.5. In a first experiment, the original LSPI (Algorithm 3.11) is applied. This algorithm represents policies implicitly and performs exact policy improvements. The results of this experiment are compared with the results of approximate Q-iteration from Section 3.4.5. In a second experiment, LSPI is modified to use approximate policies and sample-based, approximate policy improvements. The resulting solution is compared with the solution found with exact policy improvements.

In both experiments, the policies are evaluated using their Q-functions, which are approximated with a discrete-action parametrization of the type described in Example 3.1. Recall that such an approximator replicates state-dependent BFs for every discrete action, and in order to obtain the state-action BFs, it sets to 0 all the BFs that do not correspond to the current discrete action. Like in Section 3.4.5, the action space is discretized into the set $U_{\mathrm{d}} = \{-10, 0, 10\}$, so the number of discrete actions is $M = 3$. The state-dependent BFs are axis-aligned, normalized Gaussian RBFs (see Example 3.1). The centers of the RBFs are arranged on a 9×9 equidistant grid over the state space, so there are $N = 81$ RBFs in total. All the RBFs are identical in shape, and their width b_d along each dimension d is equal to ${b'_d}^2/2$, where b'_d is the distance between adjacent RBFs along that dimension (the grid step). These RBFs yield a smooth interpolation of the Q-function over the state space. Recalling that the domains of the state variables are $[-\pi, \pi]$ for the angle and $[-16\pi, 16\pi]$ for the angular velocity, we obtain $b'_1 = \frac{2\pi}{9-1} \approx 0.79$ and $b'_2 = \frac{32\pi}{9-1} \approx 12.57$, which lead to $b_1 \approx 0.31$ and $b_2 \approx 78.96$. The parameter vector θ contains $n = NM = 243$ parameters.

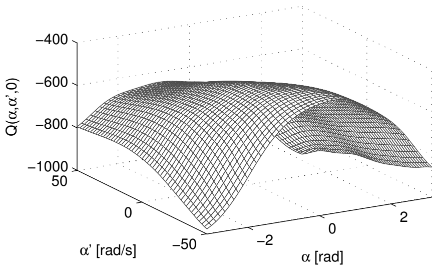
Least-squares policy iteration with exact policy improvement

In the first part of the example, the original LSPI algorithm is applied to the DC motor problem. Recall that LSPI combines LSTD-Q policy evaluation with exact policy improvement.

The same set of $n_s = 7500$ samples is used at every LSTD-Q policy evaluation. The samples are random, uniformly distributed over the state-discrete action space $X \times U_d$. The initial policy h_0 is identically equal to -10 throughout the state space. To illustrate the results of LSTD-Q, Figure 3.10 presents the first improved policy found by the algorithm, h_1, and its approximate Q-function, computed with LSTD-Q. Note that this Q-function is the *second* found by LSPI; the first Q-function evaluates the initial policy h_0.



(a) Policy h_1.

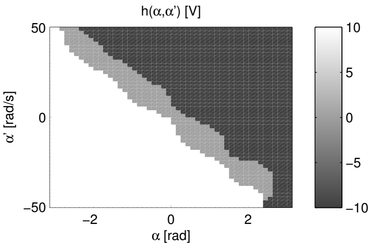


(b) Slice through the Q-function for $u = 0$. Note the difference in vertical scale from the other Q-functions shown in this chapter.

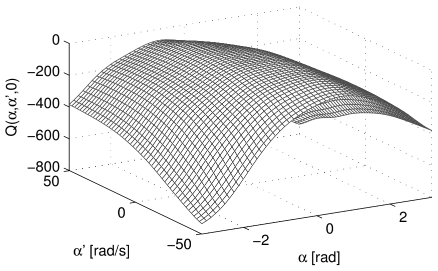
FIGURE 3.10
An early policy and its approximate Q-function, for LSPI with exact policy improvements.

In this problem, LSPI fully converged in 11 iterations. Figure 3.11 shows the resulting policy and Q-function, together with a representative controlled trajectory. The policy and the Q-function in Figure 3.11 are good approximations of the near-optimal solution in Figure 3.5.

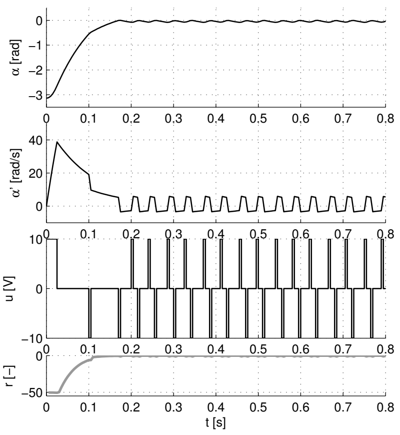
Compared to the results of grid Q-iteration in Figure 3.6, LSPI needs fewer BFs (81 rather than 400 or 160000) while still being able to find a similarly accurate approximation of the policy. This is mainly because the Q-function is largely smooth (see Figure 3.5(a)), and thus can be represented more easily by the wide RBFs of



(a) Policy $\widehat{h}^*$.



(b) Slice through the Q-function for $u = 0$.



(c) Controlled trajectory from $x_0 = [-\pi, 0]^{\mathrm{T}}$.

FIGURE 3.11 Results of LSPI with exact policy improvements for the DC motor.

the approximator employed in LSPI. In contrast, the grid BFs give a discontinuous approximate Q-function, which is less appropriate for this problem. Although certain types of continuous BFs can be used with Q-iteration, using wide RBFs such as these in combination with the least-squares projection (3.14) is unfortunately not possible, because they do not satisfy the assumptions for convergence, and indeed lead to divergence when they are too wide. The controlled trajectory in Figure 3.11(c) is comparable in quality with the trajectory controlled by the fine-grid policy, shown in Figure 3.6(f); however, it does produce more chattering.

Another observation is that LSPI converged in significantly fewer iterations than grid Q-iteration did in Section 3.4.5 (12 iterations for LSPI, instead of 160 for grid Q-iteration with the coarse grid, and 123 with the fine grid). Such a convergence rate advantage of policy iteration over value iteration is often observed in practice. However, while LSPI did converge faster, it was actually more computationally intensive than grid Q-iteration: it required approximately 23 s to run, whereas grid Q-iteration required only 0.06 s for the coarse grid and 7.80 s for the fine grid. Some insight into this difference can be obtained by examining the asymptotic complexity of the two algorithms. The complexity of policy evaluation with LSTD-Q is larger than $O(n^2)$ due to solving a linear system of size n. For grid Q-iteration, when binary search is used to locate the position of a state on the grid, the cost is $O(n\log(N))$, where $n = NM$, N is the number of elements on the grid, and M the number of discrete actions. On the other hand, while the convergence of grid Q-iteration to a fixed point was guaranteed by the theory, this is not the case for LSPI (although for this problem LSPI did, in fact, fully converge).

Compared to the results of fitted Q-iteration in Figure 3.7, the LSPI solution is of a similar quality. LSPI introduces some curved artifacts in the policy, due to the limitations of the wide RBFs employed. On the other hand, the execution time of 2151 s for fitted Q-iteration is much larger than the 23 s for LSPI.

Least-squares policy iteration with policy approximation

The aim of the second part of the example is to illustrate the effects of approximating policies. To this end, LSPI is modified to work with approximate policies and sample-based, approximate policy improvement.

The policy approximator is linearly parameterized (3.12) and uses the same RBFs as the Q-function approximator. Such an approximate policy produces *continuous* actions, which must be quantized (into discrete actions belonging to U_d) before performing policy evaluation, because the Q-function approximator only works for discrete actions. Policy improvement is performed with the linear least-squares procedure (3.47), using a number $\mathcal{N}_\mathrm{s} = 2500$ of random, uniformly distributed state samples. The same samples are used at every iteration. As before, policy evaluation employs $N_\mathrm{s} = 7500$ samples.

In this experiment, both the Q-functions and the policies *oscillate* in the steady state of the algorithm, with a period of 2 iterations. The execution time until the oscillation was detected was 58 s. 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 3.12 shows the evolution of the policy parameter that changes the most in steady state, for which the oscillation is clearly visible. The appearance of oscillations may be related to the fact that the weaker suboptimality bound (3.50) applies when approximate policies are used, rather than the stronger bound (3.51), which applies for exact policy improvements.

Figure 3.13 presents one of the two policies from the limit cycle, one of the Q-functions, and a representative controlled trajectory. The policy and Q-function have a similar accuracy to those computed with exact, discrete-action policy improvements. One advantage of the approximate policy is that it produces continuous actions. The beneficial effects of continuous actions on the control performance are apparent in the trajectory shown in Figure 3.13(c), which is very close to the near-optimal trajectory of Figure 3.5(c).

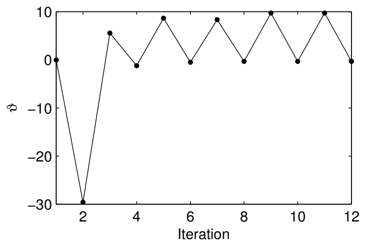
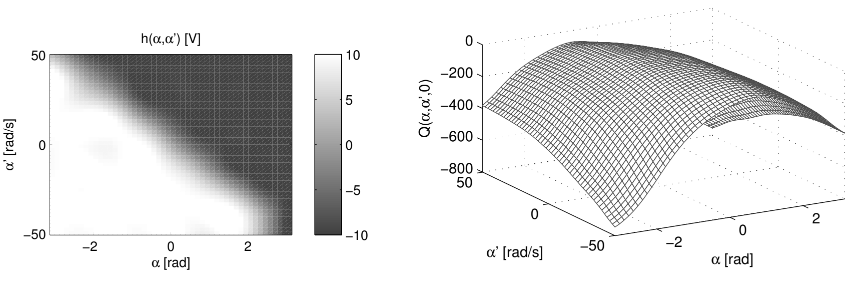
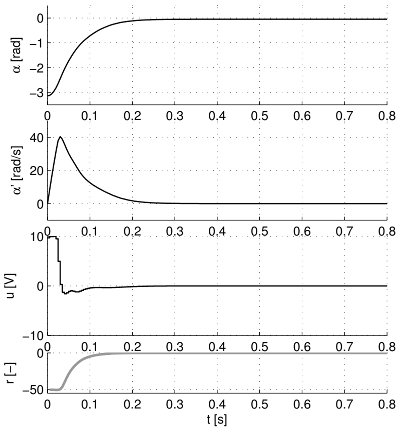


FIGURE 3.12
The variation of one of the policy parameters for LSPI with policy approximation on the DC motor.



(a) Policy.

(b) Slice through the Q-function for $u = 0$.



(c) Controlled trajectory from $x_0 = [-\pi, 0]^{\mathrm{T}}$.

FIGURE 3.13 Results of LSPI with policy approximation for the DC motor.

3.6 Finding value function approximators automatically

Parametric approximators of the value function play an important role in approximate value iteration and approximate policy iteration, as seen in Sections 3.4 and 3.5. Given the functional form of such an approximator, the DP/RL algorithm computes its parameters. However, there still remains the problem of finding a good functional form, well suited to the problem at hand. For concreteness, we will consider linearly parameterized approximators (3.3), in which case a good set of BFs has to be found. This focus is motivated by the fact that many methods to find good approximators work in such a linear setting.

The most straightforward solution is to design the BFs in advance, in which case two approaches are possible. The first is to design the BFs so that a uniform resolution is obtained over the entire state space (for V-functions) or over the entire state-action space (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 variables, and in the case of Q-functions, also with the number of action variables. The second approach is to focus the resolution on 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 regions of the state (or state-action) space is necessary in this case. Unfortunately, such prior knowledge is often nonintuitive and very difficult to obtain without actually computing the value function.

A more general alternative is to devise a method to automatically find BFs suited to the problem at hand, rather than designing them manually. Two major categories of methods to find BFs automatically are BF optimization and BF construction. *BF optimization* methods search for the best placement and shape of a (usually fixed) number of BFs. *BF construction* methods are not constrained by a fixed number of BFs, but add new or remove old BFs to improve the approximation accuracy. The newly added BFs may have different shapes, or they may all have the same shape. Several subcategories of BF construction can be distinguished, some of the most important of which are defined next.

- BF refinement methods work in a top-down fashion. They start with a few BFs (a coarse resolution) and refine them as needed.
- BF selection methods work oppositely, in a bottom-up fashion. Starting from a large number of BFs (a fine resolution), they select a small subset of BFs that still ensure a good accuracy.
- Bellman error methods for BF construction define new BFs using the Bellman error of the value function represented with the current BFs. The Bellman error (or Bellman residual) is the difference between the two sides of the Bellman equation, where the current value function has been filled in (see also the upcoming Section 3.6.1 and, e.g., (3.52)).

Figure 3.14 summarizes this taxonomy.

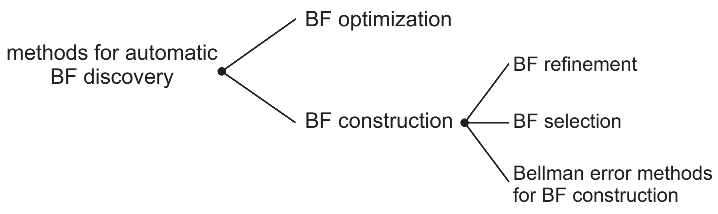

methods for automatic BF discovery
BF optimization
BF construction
BF refinement
BF selection
Bellman error methods for BF construction

FIGURE 3.14 A taxonomy of methods for the automatic discovery of BFs.

In the remainder of this section, we first describe BF optimization, in Section 3.6.1, followed by BF construction in Section 3.6.2, and by some additional remarks in Section 3.6.3.

3.6.1 Basis function optimization

BF optimization methods search for the best placement and shape of a (typically fixed) number of BFs. Consider, e.g., the linear parametrization (3.3) of the Q-function. To optimize the n BFs, they are parameterized by a vector of BF parameters ξ that encodes their locations and shapes. The approximate Q-function is:

$$\widehat{Q}(x,u) = \phi^{\mathrm{T}}(x,u;\xi)\theta$$

where the parameterized BFs have been denoted by:

$$\phi^{\mathrm{T}}(x,u;\xi) : X \times U \to \mathbb{R}, \quad l = 1,\dots,n$$

to highlight their dependence on ξ. For instance, an RBF is characterized by its center and width, so for an RBF approximator, the vector ξ contains the centers and widths of all the RBFs.

The BF optimization algorithm searches for an optimal parameter vector ξ^* that optimizes a criterion related to the accuracy of the value function approximator. Many optimization algorithms can be applied to this problem. For instance, gradient-based optimization has been used for policy evaluation with temporal difference (Singh et al., 1995), with LSTD (Menache et al., 2005; Bertsekas and Yu, 2009), and with LSPE (Bertsekas and Yu, 2009). Among these works, Bertsekas and Yu (2009) gave a general framework for gradient-based BF optimization in approximate policy evaluation, and provided an efficient recursive procedure to estimate the gradient. The cross-entropy method has been applied to LSTD (Menache et al., 2005). In Chapter 4 of this book, we will employ the cross-entropy method to optimize approximators for Q-iteration.

The most widely used optimization criterion (score function) is the *Bellman error*, also called Bellman residual (Singh et al., 1995; Menache et al., 2005; Bertsekas and Yu, 2009). This error measures how much the estimated value function violates the Bellman equation, which would be precisely satisfied by the exact value function. For instance, in the context of policy evaluation for a policy h, the Bellman error for an estimate $\widehat{Q}^h$ of the Q-function Q^h can be derived from the Bellman equation (3.31) as:

$$[T^h(\widehat{Q}^h)](x,u) - \widehat{Q}^h(x,u) \tag{3.52}$$

at the state-action pair (x,u), where T^h is the policy evaluation mapping. This error was derived from the Bellman equation (3.31). A quadratic Bellman error over the entire state-action space can therefore be defined as:

$$\int_{X\times U} \left([T^h(\widehat{Q}^h)](x,u) - \widehat{Q}^h(x,u)\right)^2 \mathrm{d}(x,u) \tag{3.53}$$

In the context of value iteration, the quadratic Bellman error for an estimate $\widehat{Q}$ of the optimal Q-function Q^* can be defined similarly:

$$\int_{X\times U} \left([T(\widehat{Q})](x,u) - \widehat{Q}(x,u)\right)^2 \mathrm{d}(x,u) \tag{3.54}$$

where T is the Q-iteration mapping. In practice, approximations of the Bellman errors are computed using a finite set of samples. A weight function can additionally be used to adjust the contribution of the errors according to the importance of each region of the state-action space.

In the context of policy evaluation, the distance between an approximate Q-function $\widehat{Q}^h$ and Q^h is related to the infinity norm of the Bellman error as follows (Williams and Baird, 1994):

$$\|\widehat{Q}^h - Q^h\|_\infty \le \frac{1}{1-\gamma}\|T^h(\widehat{Q}^h) - \widehat{Q}^h\|_\infty$$

A similar result holds in the context of value iteration, where the suboptimality of an approximate Q-function $\widehat{Q}$ satisfies (Williams and Baird, 1994; Bertsekas and Tsitsiklis, 1996, Section 6.10):

$$\|\widehat{Q} - Q^*\|_\infty \le \frac{1}{1-\gamma}\|T(\widehat{Q}) - \widehat{Q}\|_\infty$$

Furthermore, the suboptimality of $\widehat{Q}$ is related to the suboptimality of the resulting policy by (3.25), hence, in principle, minimizing the Bellman error is useful. However, in practice, *quadratic* Bellman errors (3.53), (3.54) are often employed. Because minimizing such quadratic errors may still lead to large *infinity-norm* Bellman errors, it is unfortunately unclear whether this procedure leads to accurate Q-functions.

Other optimization criteria can, of course, be used. For instance, in approximate value iteration, the return of the policy obtained by the DP/RL algorithm can be directly maximized:

$$\sum_{x_0\in X_0} w(x_0)R^h(x_0) \tag{3.55}$$

where h is obtained by running approximate value iteration to (near-)convergence using the current approximator, X_0 is a finite set of representative initial states, and $w : X_0 \to (0, \infty)$ is a weight function. The set X_0 and the weight function w determine the performance of the resulting policy, and an appropriate choice of X_0 and w depends on the problem at hand. The returns $R^h(x_0)$ can be estimated by simulation, as in approximate policy search, see Section 3.7.2.

In approximate policy evaluation, if accurate Q-values $Q^h(x_{l_s}, u_{l_s})$ can be obtained for a set of n_s samples (x_{l_s}, u_{l_s}), then the following error measure can be minimized instead of the Bellman error (Menache et al., 2005; Bertsekas and Yu, 2009):

$$\sum_{l_s=1}^{n_s} \left(Q^h(x_{l_s}, u_{l_s}) - \widehat{Q}^h(x_{l_s}, u_{l_s}) \right)^2$$

The Q-values $Q^h(x_{l_s}, u_{l_s})$ can be obtained by simulation, as explained in Section 3.5.4.

3.6.2 Basis function construction

From the class of BF construction methods, we discuss in turn BF refinement, BF selection, and Bellman error methods for BF construction (see again Figure 3.14). Additionally, we explain how some nonparametric approximators can be seen as techniques to construct BFs automatically.

Basis function refinement

BF refinement is a widely used subclass of BF construction methods. Refinement methods work in a top-down fashion, by starting with a few BFs (a coarse resolution) and refining them as needed. They can be further classified into two categories:

- Local refinement (splitting) methods evaluate whether the value function is represented with a sufficient accuracy in a particular region of the state space (corresponding to one or several neighboring BFs), and add new BFs when the accuracy is deemed insufficient. Such methods have been proposed, e.g., for Q-learning (Reynolds, 2000; Ratitch and Precup, 2004; Waldock and Carse, 2008), V-iteration (Munos and Moore, 2002), and Q-iteration (Munos, 1997; Uther and Veloso, 1998).

- Global refinement methods evaluate the global accuracy of the representation and, if the accuracy is deemed insufficient, they refine the BFs using various techniques. All the BFs may be refined uniformly (Chow and Tsitsiklis, 1991), or the algorithm may decide that certain regions of the state space require more resolution (Munos and Moore, 2002; Grüne, 2004). For instance, Chow and Tsitsiklis (1991); Munos and Moore (2002); and Grüne (2004) applied global refinement to V-iteration, while Szepesvári and Smart (2004) used 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, was given by Munos and Moore (2002). For instance, local refinement in a certain region can be performed:

- when the value function is not (approximately) constant in that region (Munos and Moore, 2002; Waldock and Carse, 2008);
- when the value function is not (approximately) linear in that region (Munos and Moore, 2002; Munos, 1997);
- when the Bellman error (see Section 3.6.1) is large in that region (Grüne, 2004);
- using various other heuristics (Uther and Veloso, 1998; Ratitch and Precup, 2004).

Global refinement can be performed, e.g., until a desired level of solution accuracy is met (Chow and Tsitsiklis, 1991). The approach of Munos and Moore (2002) works for discrete-action problems, and globally identifies the regions of the state space that must be more accurately approximated to find a better policy. To this end, it refines regions that satisfy two conditions: (i) the V-function is poorly approximated in these regions, and (ii) this poor approximation affects, in a certain sense, (other) regions where the actions that are dictated by the policy change.

BF refinement methods increase the memory and computational demands of the DP/RL algorithm when they increase the resolution. Thus, care must be taken to prevent the memory and computation costs from becoming prohibitive, especially in the online case. This is an important concern in both approximate DP and approximate RL. Equally important in approximate RL are the restrictions imposed on BF 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.

Basis function selection

BF selection methods work in a bottom-up fashion, by starting from a large number of BFs (a fine resolution), and then selecting a smaller subset of BFs that still provide a good accuracy. When using this type of methods, care should be taken to ensure that selecting the BFs and running the DP/RL algorithm with the selected BFs is less expensive than running the DP/RL algorithm with the original BFs. The cost may be expressed in terms of computational complexity or in terms of the number of samples required.

Kolter and Ng (2009) employed regularization to select BFs for policy evaluation with LSTD. Regularization is a technique that penalizes functional complexity in the approximate value function. In practice, the effect of regularization in the linear case is to drive some of the value function parameters (close) to 0, which means that the corresponding BFs can be ignored. By incrementally selecting the BFs, Kolter and Ng (2009) obtained a computational complexity that is linear in the total number of BFs, in contrast to the original complexity of LSTD which is at least quadratic (see Section 3.5.2).

Bellman error basis functions

Another class of BF construction approaches define new BFs by employing the Bellman error of the value function represented with the currently available BFs (3.53), (3.54). For instance, Bertsekas and Castañon (1989) proposed a method to interleave automatic state aggregation steps with iterations of a model-based policy evaluation algorithm. The aggregation steps group together states with similar Bellman errors. In this work, convergence speed was the main concern, rather than limited representation power, so the value function and the Bellman error function were assumed to be exactly representable.

More recently, Keller et al. (2006) proposed a method that follows similar lines, but that explicitly addresses the approximate case, by combining LSTD with Bellman-error based BF construction. At every BF construction step, this method computes a linear projection of the state space onto a space in which points with similar Bellman errors are close to each other. Several new BFs are defined in this projected space. Then, the augmented set of BFs is used to generate a new LSTD solution, and the cycle repeats. Parr et al. (2008) showed that in policy evaluation with linear parametrization, the Bellman error can be decomposed into two components: a transition error component and a reward error component, and proposed adding new BFs defined in terms of these error components.

Nonparametric approximators as methods for basis function construction

As previously explained in Section 3.3, some nonparametric approximators can be seen as methods to automatically generate BFs from the data. A typical example is kernel-based approximation, which, in its original form, generates a BF for every sample considered. An interesting effect of nonparametric approximators is that they adapt the complexity of the approximator to the amount of available data, which is beneficial in situations where obtaining data is costly.

When techniques to control the complexity of the nonparametric approximator are applied, they can sometimes be viewed as BF selection. For instance, regularization techniques were used in LSTD-Q by Farahmand et al. (2009a) and in fitted Q-iteration by Farahmand et al. (2009b). (In both of these cases, however, the advantage of regularization is a reduced functional complexity of the solution, while the computational complexity is not reduced.) Kernel sparsification techniques also fall in this category (Xu et al., 2007; Engel et al., 2003, 2005), as well as sample selection methods for regression tree approximators (Ernst, 2005).

3.6.3 Remarks

Some of the methods for automatic BF discovery work offline (e.g., Menache et al., 2005; Mahadevan and Maggioni, 2007), while others adapt the BFs while the DP/RL algorithm is running (e.g., Munos and Moore, 2002; Ratitch and Precup, 2004). Since convergence guarantees for approximate value iteration and approximate policy evaluation typically rely on a fixed set of BFs, adapting the BFs online invalidates 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 guarantee asymptotic convergence (Ernst et al., 2005).

The presentation above has not been exhaustive, and BFs can also be found using various other methods. For instance, in (Mahadevan, 2005; Mahadevan and Maggioni, 2007), a spectral analysis of the MDP transition dynamics is performed to find BFs for use with LSPI. Because the BFs represent the underlying topology of the state transitions, they provide a good accuracy in representing the value function. Moreover, while we have focused above on the popular approach of finding linearly parameterized approximators, nonlinearly parameterized approximators can also be found automatically. For example, Whiteson and Stone (2006) introduced an approach to optimize the parameters *and* the structure of neural network approximators for a tailored variant of Q-learning. This approach works in episodic tasks, and optimizes the total reward accumulated along episodes.

Finally, note that a fully worked-out example of finding an approximator automatically is beyond the scope of this chapter. Instead, we direct the interested reader to Section 4.4, where an approach to optimize the approximator for a value iteration algorithm is developed in detail, and to Section 4.5.4, where this approach is empirically evaluated.

3.7 Approximate policy search

Algorithms for approximate policy search represent the policy approximately, most often using a parametric approximator. An optimal parameter vector is then sought using optimization techniques. In some special cases, the policy parametrization may represent an optimal policy exactly. For instance, when the transition dynamics are linear in the state and action variables and the reward function is quadratic, the optimal policy is linear in the state variables. So, a linear parametrization in the state variables can exactly represent this optimal policy. However, in general, optimal policies can only be represented approximately.

Figure 3.15 (repeated from the relevant part of Figure 3.2) shows in a graphical form how our upcoming presentation of approximate policy search is organized. In Section 3.7.1, gradient-based methods for policy search are described, including the important category of actor-critic techniques. Then, in Section 3.7.2, gradient-free policy optimization methods are discussed.

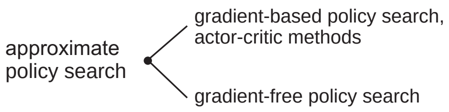


FIGURE 3.15
The organization of the algorithms for approximate policy search presented next.

Having completed our review, we then provide a numerical example involving policy search for a DC motor in Section 3.7.3.

3.7.1 Policy gradient and actor-critic algorithms

An important class of methods for approximate policy search relies on gradient-based optimization. In such *policy gradient* methods, the policy is represented using a differentiable parametrization, and gradient updates are performed to find parameters that lead to (locally) maximal returns. Some policy gradient methods estimate the gradient without using a value function (Marbach and Tsitsiklis, 2003; Munos, 2006; Riedmiller et al., 2007). Other methods compute an approximate value function of the current policy and use it to form the gradient estimate. These are called *actor-critic* methods, where the actor is the approximate policy and the critic is the approximate value function. By extension, policy gradient methods that do not use value functions are sometimes called *actor-only* methods (Bertsekas, 2007, Section 6.7).

Actor-critic algorithms were introduced by Barto et al. (1983) and have been investigated often since then (Berenji and Khedkar, 1992; Sutton et al., 2000; Konda and Tsitsiklis, 2003; Berenji and Vengerov, 2003; Borkar, 2005; Nakamura et al., 2007). Many actor-critic algorithms approximate the policy and the value function using neural networks (Prokhorov and Wunsch, 1997; Pérez-Uribe, 2001; Liu et al., 2008). Actor-critic methods are similar to policy iteration, which also improves the policy on the basis of its value function. The main difference is that in policy iteration, the improved policy is greedy in the value function, i.e., it *fully maximizes* this value function over the action variables (3.46). In contrast, actor-critic methods employ gradient rules to update the policy in a *direction* that increases the received returns. The gradient estimate is constructed using the value function.

Some important results for policy gradient methods have been developed under the expected average return criterion for optimality. We therefore discuss this setting first, in a temporary departure from the main focus of the book, which is the discounted return. We then return to the discounted setting, and present an online actor-critic algorithm for this setting.

Policy gradient and actor-critic methods for average returns

Policy gradient and actor-critic methods have often been given in the average return setting (see also Section 2.2.1). We therefore introduce these methods in the average-return case, mainly following the derivation of Bertsekas (2007, Section 6.7). We assume that the MDP has a finite state-action space, but under appropriate conditions these methods can also be extended to continuous state-action spaces (see, e.g., Konda and Tsitsiklis, 2003).

Consider a stochastic MDP with a finite state space $X = \{x_1, \dots, x_{\bar{N}}\}$, a finite action space $U = \{u_1, \dots, u_{\bar{M}}\}$, a transition function $\bar{f}$ of the form (2.14), and a reward function $\tilde{\rho}$. A stochastic policy of the form $\tilde{h} : X \times U \to [0,1]$ is employed, parameterized by the vector $\vartheta \in \mathbb{R}^{\mathcal{N}}$. This policy takes an action u in state x with the probability:

$$\mathrm{P}(u\,|\,x) = \tilde{h}(x,u;\vartheta)$$

The functional dependence of the policy on the parameter vector must be designed in advance, and must be differentiable.

The *expected average return* of state x_0 under the policy parameterized by ϑ is:

$$R^{\vartheta}(x_0) = \lim_{K\to\infty} \frac{1}{K} \mathrm{E}_{\substack{u_k \sim \tilde{h}(x_k,\cdot;\vartheta) \\ x_{k+1} \sim \bar{f}(x_k,u_k,\cdot)}} \left\{ \sum_{k=0}^{K} \tilde{\rho}(x_k,u_k,x_{k+1}) \right\}$$

Note that we have directly highlighted the dependence of the return on the parameter vector ϑ, rather than on the policy $\tilde{h}$. A similar notation will be used for other policy-dependent quantities in this section.

Under certain conditions (see, e.g., Bertsekas, 2007, Chapter 4), the average return is the same for every initial state, i.e., $R^{\vartheta}(x_0) = \mathscr{R}^{\vartheta}$ for all $x_0 \in X$, and together with the so-called *differential* V-function, $V^{\vartheta} : X \to \mathbb{R}$, satisfies the Bellman equation:

$$\mathscr{R}^{\vartheta} + V^{\vartheta}(x_i) = \tilde{\rho}^{\vartheta}(x_i) + \sum_{i'=1}^{\bar{N}} \bar{f}^{\vartheta}(x_i,x_{i'}) V^{\vartheta}(x_{i'}) \tag{3.56}$$

The differential value of a state x can be interpreted as the expected excess return, on top of the average return, obtained from x (Konda and Tsitsiklis, 2003). The other quantities appearing in (3.56) are defined as follows:

- $\bar{f}^{\vartheta} : X \times X \to [0,1]$ gives the state transition probabilities under the policy considered, from which the influence of the actions has been integrated out.[12] These probabilities can be computed with:

$$\bar{f}^{\vartheta}(x_i,x_{i'}) = \sum_{j=1}^{\bar{M}} \left[\tilde{h}(x_i,u_j;\vartheta) \bar{f}(x_i,u_j,x_{i'}) \right]$$

- $\tilde{\rho}^{\vartheta} : X \to \mathbb{R}$ gives the expected rewards obtained from every state by the policy considered, and can be computed with:

$$\tilde{\rho}^{\vartheta}(x_i) = \sum_{j=1}^{\bar{M}} \left[\tilde{h}(x_i,u_j;\vartheta) \sum_{i'=1}^{\bar{N}} \left(\bar{f}(x_i,u_j,x_{i'}) \tilde{\rho}(x_i,u_j,x_{i'}) \right) \right]$$

Policy gradient methods aim to find a (locally) optimal policy within the class of parameterized policies considered. An optimal policy maximizes the average return, which is the same for every initial state. So, a parameter vector that (locally)

[12]For simplicity, a slight abuse of notation is made by using $\bar{f}$ to denote both the original transition function and the transition probabilities from which the actions have been factored out. Similarly, the expected rewards are denoted by $\tilde{\rho}$, like the original reward function.

maximizes the average return must be found. To this end, policy gradient methods perform gradient ascent on the average return:

$$\vartheta \leftarrow \vartheta + \alpha \frac{\partial \mathscr{R}^{\vartheta}}{\partial \vartheta} \tag{3.57}$$

where α is the step size. When a local optimum has been reached, the gradient is zero, i.e., $\frac{\partial \mathscr{R}^{\vartheta}}{\partial \vartheta} = 0$.

The core problem is to estimate the gradient $\frac{\partial \mathscr{R}^{\vartheta}}{\partial \vartheta}$. By differentiating the Bellman equation (3.56) with respect to ϑ and after some calculations (see Bertsekas, 2007, Section 6.7), the following formula for the gradient is obtained:

$$\frac{\partial \mathscr{R}^{\vartheta}}{\partial \vartheta} = \sum_{i=1}^{\bar{N}} \zeta^{\vartheta}(x_i) \left[\frac{\partial \tilde{\rho}^{\vartheta}(x_i)}{\partial \vartheta} + \sum_{i'=1}^{\bar{N}} \left(\frac{\partial \bar{f}^{\vartheta}(x_i, x_{i'})}{\partial \vartheta} V^{\vartheta}(x_{i'}) \right) \right] \tag{3.58}$$

where $\zeta^{\vartheta}(x_i)$ is the steady-state probability of encountering the state x_i when using the policy given by ϑ. Note that all the gradients in (3.58) are $\mathscr{N}$-dimensional vectors.

The right-hand side of (3.58) can be estimated using simulation, as proposed, e.g., by Marbach and Tsitsiklis (2003), and the convergence of the resulting policy gradient algorithms to a locally optimal parameter vector can be ensured under mild conditions. An important concern is controlling the variance of the gradient estimate, and Marbach and Tsitsiklis (2003) focused on this problem. Munos (2006) considered policy gradient methods in the continuous-time setting. Because the usual methods to estimate the gradient lead to a variance that grows very large as the sampling time decreases, other methods are necessary to keep the variance small in the continuous-time case (Munos, 2006).

Actor-critic methods explicitly approximate the V-function in (3.58). This approximate V-function can be found, e.g., by using variants of the TD, LSTD, and LSPE techniques adapted to the average return setting (Bertsekas, 2007, Section 6.6).

The gradient can also be expressed in terms of a Q-function, which can be defined in the average return setting by using the differential V-function, as follows:

$$Q^{\vartheta}(x_i, u_j) = \sum_{i'=1}^{\bar{N}} \left[\bar{f}(x_i, u_j, x_{i'}) \left(\tilde{\rho}(x_i, u_j, x_{i'}) - \mathscr{R}^{\vartheta} + V^{\vartheta}(x_{i'}) \right) \right]$$

Using the Q-function, the gradient of the average return can be written as (Sutton et al., 2000; Konda and Tsitsiklis, 2000, 2003):

$$\frac{\partial \mathscr{R}^{\vartheta}}{\partial \vartheta} = \sum_{i=1}^{\bar{N}} \sum_{j=1}^{\bar{M}} \left[w^{\vartheta}(x_i, u_j) Q^{\vartheta}(x_i, u_j) \phi^{\vartheta}(x_i, u_j) \right] \tag{3.59}$$

where $w^{\vartheta}(x_i, u_j) = \zeta^{\vartheta}(x_i)\tilde{h}(x_i, u_j; \vartheta)$ is the steady-state probability of encountering the state-action pair (x_i, u_j) when using the policy considered, and:

$$\phi^{\vartheta} : X \times U \to \mathbb{R}^{\mathscr{N}}, \quad \phi^{\vartheta}(x_i, u_j) = \frac{1}{\tilde{h}(x_i, u_j; \vartheta)} \frac{\partial \tilde{h}(x_i, u_j; \vartheta)}{\partial \vartheta} \tag{3.60}$$

The function ϕ^{ϑ} is regarded as a *vector of state-action BFs*, for reasons that will become clear shortly. It can be shown that (3.59) is equal to (Sutton et al., 2000; Konda and Tsitsiklis, 2003):

$$\frac{\partial \mathscr{R}^{\vartheta}}{\partial \vartheta} = \sum_{i=1}^{\bar{N}} \sum_{j=1}^{\bar{M}} \left[w(x_i, u_j) [P^{w^{\vartheta}}(Q^{\vartheta})](x_i, u_j) \phi^{\vartheta}(x_i, u_j) \right]$$

where the exact Q-function has been substituted by its weighted least-squares projection (3.35) onto the space spanned by the BFs ϕ^{ϑ}. So, in order to find the *exact* gradient, it is sufficient to compute an *approximate* Q-function – provided that the BFs ϕ^{ϑ}, computed with (3.60) from the policy parametrization, are used. In the literature, such BFs are sometimes called “compatible” with the policy parametrization (Sutton et al., 2000) or “essential features” (Bertsekas, 2007, Section 6.7). Note that other BFs can be used in addition to these.

Using this property, actor-critic algorithms that linearly approximate the Q-function using the BFs (3.60) can be given. These algorithms converge to a locally optimal policy, as shown by Sutton et al. (2000); Konda and Tsitsiklis (2000, 2003). Konda and Tsitsiklis (2003) additionally extended their analysis to the case of continuous state-action spaces. This theoretical framework was used by Berenji and Vengerov (2003) to prove the convergence of an actor-critic algorithm relying on fuzzy approximation.

Kakade (2001) proposed an improvement to the gradient update formula (3.57), by scaling it with the inverse of the (expected) Fisher information matrix of the stochastic policy (Schervish, 1995, Section 2.3.1), and thereby obtaining the so-called natural policy gradient. Peters and Schaal (2008) and Bhatnagar et al. (2009) employed this idea to develop some natural actor-critic algorithms. Riedmiller et al. (2007) provided an experimental comparison of several policy gradient methods, including the natural policy gradient.

An online actor-critic algorithm for discounted returns

We now come back to the discounted return criterion for optimality, and describe an actor-critic algorithm for this discounted setting (rather than in the average-return setting, as above). This algorithm works online, in problems with continuous states and actions. Denote by $\widehat{h}(x;\vartheta)$ the (deterministic) approximate policy, parameterized by $\vartheta \in \mathbb{R}^{\mathscr{N}}$, and by $\widehat{V}(x;\theta)$ the approximate V-function, parameterized by $\theta \in \mathbb{R}^{N}$. The algorithm does not distinguish between the value functions of different policies, so the value function notation is not superscripted by the policy. Although a deterministic approximate policy is considered, a stochastic policy could also be used.

At each time step, an action u_k is chosen by adding a random, exploratory term to the action recommended by the policy $\widehat{h}(x;\vartheta)$. This term could be drawn, e.g., from a zero-mean Gaussian distribution. After the transition from x_k to x_{k+1}, an approximate temporal difference is computed with:

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

This temporal difference can be obtained from the Bellman equation for the policy

V-function (2.20). It is analogous to the temporal difference for Q-functions, used, e.g., in approximate SARSA (Algorithm 3.12). Once the temporal difference is computed, the policy and V-function parameters are updated with the following gradient formulas:

$$\vartheta_{k+1} = \vartheta_k + \alpha_{\mathrm{A},k} \frac{\partial \widehat{h}(x_k; \vartheta_k)}{\partial \vartheta} [u_k - \widehat{h}(x_k; \vartheta_k)]\, \delta_{\mathrm{TD},k} \tag{3.61}$$

$$\theta_{k+1} = \theta_k + \alpha_{\mathrm{C},k} \frac{\partial \widehat{V}(x_k; \theta_k)}{\partial \theta} \delta_{\mathrm{TD},k} \tag{3.62}$$

where $\alpha_{\mathrm{A},k}$ and $\alpha_{\mathrm{C},k}$ are the (possibly time-varying) step sizes for the actor and the critic, respectively. Note that the action signal is assumed to be scalar, but the method can be extended to multiple action variables.

In the actor update (3.61), due to exploration, the actual action u_k applied at step k can be different from the action recommended by the policy. 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 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 to be better than the predicted one. In the critic update (3.62), 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 $r_{k+1} + \gamma \widehat{V}(x_{k+1}; \theta_k)$ suggested by the Bellman equation (2.20), thus leading to the temporal difference.

This actor-critic method is summarized in Algorithm 3.13, which generates exploratory actions using a Gaussian density with a standard deviation that can vary over time.

ALGORITHM 3.13 Actor-critic with Gaussian exploration.

```
Input: discount factor $\gamma$,
    policy parametrization $\widehat{h}$, V-function parametrization $\widehat{V}$,
    exploration schedule $\{\sigma_k\}_{k=0}^{\infty}$, step size schedules $\{\alpha_{\mathrm{A},k}\}_{k=0}^{\infty}$, $\{\alpha_{\mathrm{C},k}\}_{k=0}^{\infty}$
1: initialize parameter vectors, e.g., $\vartheta_0 \leftarrow 0$, $\theta_0 \leftarrow 0$
2: measure initial state $x_0$
3: for every time step $k = 0, 1, 2, \ldots$ do
4:     $u_k \leftarrow \widehat{h}(x_k; \vartheta_k) + \bar{u}$, where $\bar{u} \sim \mathcal{N}(0, \sigma_k)$
5:     apply $u_k$, measure next state $x_{k+1}$ and reward $r_{k+1}$
6:     $\delta_{\mathrm{TD},k} = r_{k+1} + \gamma \widehat{V}(x_{k+1}; \theta_k) - \widehat{V}(x_k; \theta_k)$
7:     $\vartheta_{k+1} = \vartheta_k + \alpha_{\mathrm{A},k} \frac{\partial \widehat{h}(x_k; \vartheta_k)}{\partial \vartheta} [u_k - \widehat{h}(x_k; \vartheta_k)]\, \delta_{\mathrm{TD},k}$
8:     $\theta_{k+1} = \theta_k + \alpha_{\mathrm{C},k} \frac{\partial \widehat{V}(x_k; \theta_k)}{\partial \theta} \delta_{\mathrm{TD},k}$
9: end for
```

3.7.2 Gradient-free policy search

Gradient-based policy optimization is based on the assumption that the locally optimal parameters found by the gradient method are good enough. This may be true when the policy parametrization is simple and well suited to the problem at hand. However, in order to design such a parametrization, prior knowledge about a (near-)optimal policy is required.

When prior knowledge about the policy is not available, a richer policy parametrization must be used. In this case, the optimization criterion is likely to have many local optima, and may also be nondifferentiable. This means that gradient-based algorithms are unsuitable, and global, gradient-free optimization algorithms are required. Even when a simple policy parametrization can be designed, global optimization can help by avoiding local optima.

Consider the DP/RL problem under the expected discounted return criterion. Denote by $\widehat{h}(x;\vartheta)$ the approximate policy, parameterized by $\vartheta \in \mathbb{R}^{\mathcal{N}}$. Policy search algorithms look for an optimal parameter vector that maximizes the return $R^{\widehat{h}(\cdot;\vartheta)}(x)$ for all $x \in X$. When X is large or continuous, computing the return for every initial 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 the states in X_0, and the score function (optimization criterion) is the weighted average return over these states:

$$s(\vartheta) = \sum_{x_0 \in X_0} w(x_0) R^{\widehat{h}(\cdot;\vartheta)}(x_0) \tag{3.63}$$

where $w : X_0 \rightarrow (0,1]$ is the weight function.[13] The return from each representative state is estimated by simulation. A number of $N_{\text{MC}} \geq 1$ independent trajectories are simulated from every representative state, and an estimate of the expected return is obtained by averaging the returns obtained along these sample trajectories:

$$R^{\widehat{h}(\cdot;\vartheta)}(x_0) = \frac{1}{N_{\text{MC}}} \sum_{i_0=1}^{N_{\text{MC}}} \sum_{k=0}^{K} \gamma^k \tilde{\rho}(x_{i_0,k}, h(x_{i_0,k};\vartheta), x_{i_0,k+1}) \tag{3.64}$$

For each trajectory i_0, the initial state $x_{i_0,0}$ is equal to x_0, and actions are chosen with the policy h, which means that for $k \geq 0$:

$$x_{i_0,k+1} \sim f(x_{i_0,k}, h(x_{i_0,k};\vartheta), \cdot)$$

If the system is deterministic, a single trajectory suffices, i.e., $N_{\text{MC}} = 1$. In the stochastic case, a good value for N_{MC} will depend on the problem at hand. Note that this Monte Carlo estimation procedure is similar to a rollout (3.45).

The infinite-horizon return is approximated by truncating each simulated trajectory after K steps. A value of K that guarantees that this truncation introduces an

[13]More generally, a density $\tilde{w}$ over the initial states can be considered, and the score function is then $\mathrm{E}_{x_0 \sim \tilde{w}(\cdot)}\left\{R^{h(\cdot;\xi,\vartheta)}(x_0)\right\}$, i.e., the expected value of the return when $x_0 \sim \tilde{w}(\cdot)$.

error of at most $\varepsilon_{\mathrm{MC}} > 0$ can be chosen using (2.41), repeated here:

$$K = \left\lceil \log_\gamma \frac{\varepsilon_{\mathrm{MC}}(1-\gamma)}{\|\tilde{\rho}\|_\infty} \right\rceil \tag{3.65}$$

In the stochastic context, Ng and Jordan (2000) assumed the availability of a simulation model that offers access to the random variables driving the stochastic transitions. They proposed to pregenerate sequences of values for these random variables, and to use the same sequences when evaluating every policy. This leads to a deterministic optimization problem.

Representative set of initial states and weight function. The set X_0 of representative states, together with the weight function w, determines the performance of the resulting policy. Of course, this performance is in general only approximately optimal, since maximizing the returns from states in X_0 cannot guarantee that returns from other states in X are maximal. A good choice of X_0 and w will depend on the problem at hand. For instance, if the process only needs to be controlled starting from a known set X_{init} of initial states, then X_0 should be equal to X_{init}, or included in it when X_{init} is too large. 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 and identical weights equal to $\frac{1}{|X_0|}$ should be assigned to every element of X_0.

A wide range of gradient-free, global optimization techniques can be employed in policy search, including evolutionary optimization (e.g., genetic algorithms, see Goldberg, 1989), tabu search (Glover and Laguna, 1997), pattern search (Torczon, 1997; Lewis and Torczon, 2000), the cross-entropy method (Rubinstein and Kroese, 2004), etc. For instance, evolutionary computation was applied to policy search by Barash (1999); Chin and Jafari (1998); Gomez et al. (2006); Chang et al. (2007, Chapter 3), and cross-entropy optimization was applied by Mannor et al. (2003). Chang et al. (2007, Chapter 4) described an approach to find a policy by using the so-called "model-reference adaptive search," which is closely related to the cross-entropy method. In Chapter 6 of this book, we will employ the cross-entropy method to develop a policy search algorithm. A dedicated algorithm that optimizes the parameters and structure of neural network policy approximators was given by Whiteson and Stone (2006). General policy modification heuristics were proposed by Schmidhuber (2000).

In another class of model-based policy search approaches, near-optimal actions are sought online, by executing at every time step a search over open-loop sequences of actions (Hren and Munos, 2008). The controller selects a sequence leading to a maximal estimated return and applies the first action in this sequence. Then, the entire cycle repeats.[14] The total number of open-loop action sequences grows exponentially with the time horizon considered, but by limiting the search to promising sequences only, such an approach can avoid incurring excessive computational costs.

[14]This is very similar to how model-predictive control works (Maciejowski, 2002; Camacho and Bordons, 2004).

Hren and Munos (2008) studied this method of limiting the computational cost in a deterministic setting. In a stochastic setting, open-loop sequences are suboptimal. However, some approaches exist to extend this open-loop philosophy to the stochastic case. These approaches model the sequences of random transitions by scenario trees (Birge and Louveaux, 1997; Dupacová et al., 2000) and optimize the actions attached to the tree nodes (Defourny et al., 2008, 2009).

3.7.3 Example: Gradient-free policy search for a DC motor

In this example, approximate, gradient-free policy search is applied to the DC motor problem introduced in Section 3.4.5. In a first experiment, a general policy parametrization is used that does not rely on prior knowledge, whereas in a second experiment, a tailored policy parametrization is derived from prior knowledge. The results obtained with these two parametrizations are compared.

To compute the score function (3.63), a set X_0 of representative states and a weight function w have to be selected. We aim to obtain a uniform performance across the state space, so a regular grid of representative states is chosen:

$$X_0 = \{-\pi, -2\pi/3, -\pi/3, \dots, \pi\} \times \{-16\pi, -12\pi, -8\pi, \dots, 16\pi\}$$

and these initial states are weighted uniformly by $w(x_0) = \frac{1}{|X_0|}$, where the number of states is $|X_0| = 63$. A maximum error $\varepsilon_{\text{MC}} = 0.01$ is imposed in the estimation of the return. A bound on the reward function (3.28) for the DC motor problem can be computed with:

$$\begin{aligned}
\|\rho\|_\infty &= \sup_{x,u} \left| -x_k^{\text{T}} Q_{\text{rew}} x_k - R_{\text{rew}} u_k^2 \right| \\
&= \left| -[\pi \;\, 16\pi] \begin{bmatrix} 5 & 0 \\ 0 & 0.01 \end{bmatrix} \begin{bmatrix} \pi \\ 16\pi \end{bmatrix} - 0.01 \cdot 10^2 \right| \\
&\approx 75.61
\end{aligned}$$

To find the trajectory length K required to achieve the precision ε_{MC}, the values of ε_{MC}, $\|\rho\|_\infty$, and $\gamma = 0.95$ are substituted into (3.65); this yields $K = 233$. Because the problem is deterministic, simulating multiple trajectories from every initial state is not necessary; instead, a single trajectory from every initial state will suffice.

We use the global, gradient-free pattern search algorithm to optimize the policy (Torczon, 1997; Lewis and Torczon, 2000). The algorithm is considered convergent when the score variation decreases below the threshold $\varepsilon_{\text{PS}} = 0.01$ (equal to ε_{MC}).[15]

Policy search with a general parametrization

Consider first the case in which no prior knowledge about the optimal policy is available, which means that a general policy parametrization must be used. The linear

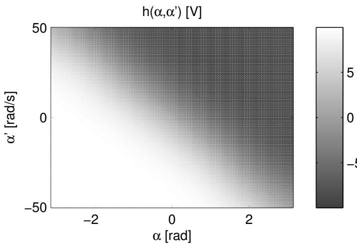
[15] We use the pattern search algorithm from the *Genetic Algorithm and Direct Search Toolbox* of MATLAB 7.4.0. The algorithm is configured to use the threshold ε_{PS} and to cache the score values for the parameter vectors it already evaluated, in order to avoid recomputing them. Besides these changes, the default settings of the algorithm are employed.

policy parametrization (3.12) is chosen:

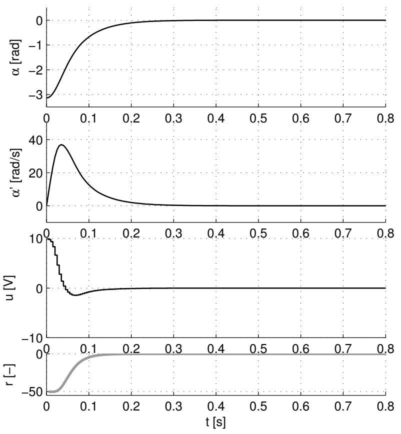
$$\widehat{h}(x) = \sum_{i=1}^{\mathscr{N}} \varphi_i(x)\vartheta_i = \varphi^{\mathrm{T}}(x)\vartheta$$

Axis-aligned, normalized RBFs (see Example 3.1) are defined, with their centers arranged on an equidistant 7×7 grid in the state space. All the RBFs are identical in shape, and their width b_d along each dimension d is equal to $b_d'^2/2$, where b_d' is the distance between adjacent RBFs along that dimension (the grid step). Namely, $b_1' = \frac{2\pi}{7-1} \approx 1.05$ and $b_2' = \frac{32\pi}{7-1} \approx 16.76$, which lead to $b_1 \approx 0.55$ and $b_2 \approx 140.37$. In total, 49 parameters (for 7×7 RBFs) must be optimized.

Pattern search optimization is applied to find an optimal parameter vector ϑ^*, starting from an identically zero parameter vector. Figure 3.16 shows the policy obtained and a representative trajectory that is controlled by this policy. The policy is largely linear in the state variables (within the saturation limits), and leads to a good convergence to the zero state.



(a) Policy.



(b) Controlled trajectory from $x_0 = [-\pi, 0]^{\mathrm{T}}$.

FIGURE 3.16
Results of policy search with the general policy parametrization for the DC motor.

In this experiment, the pattern search algorithm required 18173 s to converge. This execution time is larger than for all other algorithms applied earlier to the DC motor (grid Q-iteration and fitted Q-iteration in Section 3.4.5, and LSPI in Section 3.5.7), illustrating the large computational demands of policy search with general parametrizations.

Policy search spends the majority of its execution time estimating the score function (3.63), which is a computationally expensive operation. For this experiment, the score of 11440 different parameter vectors had to be computed until convergence. The computational cost of evaluating each parameter vector can be decreased by taking a smaller X_0 or larger $\varepsilon_{\mathrm{MC}}$ and $\varepsilon_{\mathrm{PS}}$, at the expense of a possible decrease in control performance.

Policy search with a tailored parametrization

In this second part of the example, we employ a simple policy parametrization that is well suited to the DC motor problem. This parametrization is derived by using prior knowledge. 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 (Bertsekas, 2007, Section 3.2).[16] Now taking into account the constraints on the action, we assume that a good approximation of an optimal policy is linear in the state variables, within the constraints on the action:

$$\widehat{h}(x;\vartheta) = \text{sat}\{\vartheta_1 x_1 + \vartheta_2 x_2, -10, 10\} \tag{3.66}$$

where "sat" denotes saturation. In fact, an examination of the near-optimal policy in Figure 3.5(b) on page 67 reveals that this assumption is largely correct: the only non-linearities appear in the top-left and bottom-right corners of the figure; they are probably due to the constraints on the state variables, which were not taken into account when deriving the parametrization (3.66). We employ this tailored parametrization to perform policy search. Note that only 2 parameters must be optimized, significantly fewer than the 49 parameters required by the general parametrization used earlier.

Figure 3.17 shows the policy obtained by pattern search optimization, together with a representative controlled trajectory. As expected, the policy closely resembles the near-optimal policy of Figure 3.5(b), with the exception of the nonlinearities in the corners of the state space. The trajectory obtained is also close to the near-optimal one in Figure 3.5(c). Compared to the general-parametrization solution of Figure 3.16, the policy varies more quickly in the linear portion, which results in a more aggressive control signal. This is because the tailored parametrization can lead to a large slope of the policy, whereas the wide RBFs used in Figure 3.16 lead to a smoother interpolation. The score obtained by the policy of Figure 3.17 is -229.25, slightly better than the score of -230.69 obtained by the RBF policy of Figure 3.16.

The execution time of pattern search with the tailored parametrization was approximately 75 s. As expected, the computational cost is much smaller than for the general parametrization, because only 2 parameters must be optimized, instead of 49. This illustrates the benefits of using a compact policy parametrization that is appropriate for the problem at hand. Unfortunately, deriving an appropriate parametrization requires prior knowledge, which is not always available. The execution time is larger than that of grid Q-iteration in Section 3.4.5, which was 7.80 s for the fine grid and 0.06 s for the coarse grid. It has the same order of magnitude as the execution time of LSPI in Section 3.5.7, which was 23 s when using exact policy improvements, and 58 s with approximate policy improvements; but it is smaller than the execution

[16] This optimal linear state feedback is given by:

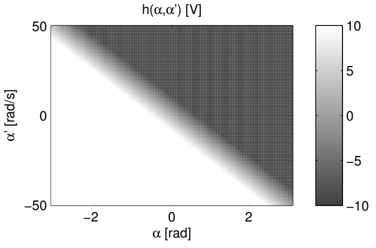
$$h(x) = Kx = -\gamma(\gamma B^{\mathrm{T}} Y B + R_{\text{rew}})^{-1} B^{\mathrm{T}} Y A x$$

where Y is the stabilizing solution of the Riccati equation:

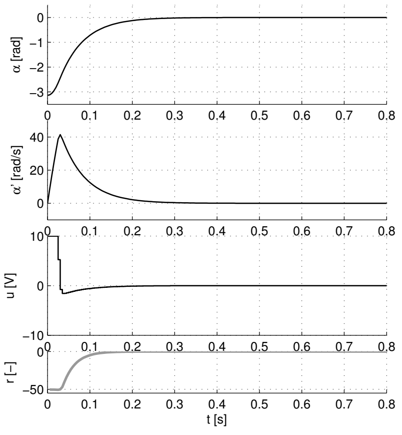
$$Y = A^{\mathrm{T}}[\gamma Y - \gamma^2 Y B(\gamma B^{\mathrm{T}} Y B + R_{\text{rew}})^{-1} B^{\mathrm{T}}]A + Q_{\text{rew}}$$

Substituting A, B, Q_{rew}, R_{rew}, and γ in these equations leads to a state feedback gain of $K \approx [-11.16, -0.67]^{\mathrm{T}}$ for the DC motor.

time 2151 s of fitted Q-iteration. To enable an easy comparison of all these execution times, they are collected in Table 3.1.[17]



(a) Policy.



(b) Controlled trajectory from $x_0 = [-\pi, 0]^{\mathrm{T}}$.

FIGURE 3.17
Results of policy search with the tailored policy parametrization (3.66) on the DC motor. The policy parameter is $\widehat{\vartheta}^* = [-16.69, -1]^{\mathrm{T}}$.

TABLE 3.1
Execution time of approximate DP and RL algorithms for the DC motor problem.

Algorithm	**Execution time** [s]
grid Q-iteration with a coarse grid	0.06
grid Q-iteration with a fine grid	7.80
fitted Q-iteration	2151
LSPI with exact policy improvement	23
LSPI with exact policy approximation	58
policy search with a general parametrization	18173
policy search with a tailored parametrization	75

[17] Recall that all these execution times were recorded on a PC with an Intel Core 2 Duo T9550 2.66 GHz CPU and with 3 GB RAM.

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

This section provides a general, qualitative comparison of approximate value iteration, approximate policy iteration, and approximate policy search. A more specific comparison would of course depend on the particular algorithms considered and on the problem at hand.

Approximate value iteration versus approximate policy iteration

Offline approximate policy iteration often converges in a small number of iterations, possibly smaller than the number of iterations taken by offline approximate value iteration. This was illustrated for the DC motor example, in which LSPI (Section 3.5.7) converged faster than grid Q-iteration (Section 3.4.5). However, this does not mean that approximate policy iteration is computationally less demanding than approximate value iteration, since approximate policy evaluation is a difficult problem by itself, which must be solved at every single policy iteration. One advantage of approximate value iteration is that it usually guarantees convergence to a unique solution, whereas approximate policy iteration is generally only guaranteed to converge to a sequence of policies that all provide a guaranteed level of performance. This was illustrated in Section 3.5.7, where LSPI with policy approximation converged to a limit cycle.

Consider now the approximate policy evaluation step of policy iteration, in comparison to approximate value iteration. Some approximate policy evaluation algorithms closely parallel approximate value iteration and converge under similar conditions (Section 3.5.1). However, approximate policy evaluation can additionally benefit from the linearity of the Bellman equation for a policy's value function, e.g., (2.7), whereas the Bellman optimality equation, which characterizes the optimal value function, e.g., (2.8), is highly nonlinear due to the maximization in the right-hand side. A class of algorithms for approximate policy evaluation exploit this linearity property by solving a projected form of the Bellman equation (Section 3.5.2). One advantage of such algorithms is that they only require the approximator to be linearly parameterized, whereas in approximate value iteration the approximator must lead to contracting updates (Section 3.4.4). Moreover, some of these algorithms, such as LSTD-Q and LSPE-Q, are highly sample-efficient. However, a disadvantage of these algorithms is that their convergence guarantees typically require a sample distribution identical with the steady-state distribution under the policy being evaluated.

Approximate policy search versus approximate value iteration and policy iteration

For some problems, deriving a good policy parametrization using prior knowledge may be easier and more natural than deriving a good value function parametrization. If a good policy parametrization is available and this parametrization is differentiable, policy gradient algorithms can be used (Section 3.7.1). Such algorithms are backed by useful convergence guarantees and have moderate computational demands. Policy gradient algorithms have the disadvantage that they can only find local optima in the class of parameterized policies considered, and may also suffer from slow convergence.

Note that the difficulty of designing a good value function parametrization can be alleviated either by automatically finding the parametric approximator (Section 3.6) or by using nonparametric approximators. Both of these options require less tuning than a predefined parametric approximator, but may increase the computational demands of the algorithm.

Even when prior knowledge is not available and a good policy parametrization cannot be obtained, approximate policy search can still be useful in its gradient-free forms, which do not employ value functions (Section 3.7.2). One situation in which value functions are undesirable is when value-function based algorithms fail to obtain a good solution, or require too restrictive assumptions. In such situations, a general policy parametrization can be defined, and a global, gradient-free optimization technique can be used to search for optimal parameters. These techniques are usually free from numerical problems – such as divergence to infinity – even when used with general nonlinear parametrizations, which is not the case for value and policy iteration. However, because of its generality, this approach typically incurs large computational costs.

3.9 Summary and discussion

In this chapter, we have introduced approximate dynamic programming (DP) and approximate reinforcement learning (RL) for large or continuous-space problems. After explaining the need for approximation in such problems, parametric and nonparametric approximation architectures have been presented. Then, approximate versions for the three main categories of algorithms have been described: value iteration, policy iteration, and policy search. Theoretical results have been provided and the behavior of representative algorithms has been illustrated using numerical examples. Additionally, techniques to automatically determine value function approximators have been reviewed, and the three categories of algorithms have been compared. Extensive accounts of approximate DP and RL, presented from different perspectives, can also be found in the books of Bertsekas and Tsitsiklis (1996); Powell (2007); Chang et al. (2007); Cao (2007).

Approximate DP/RL is a young, but active and rapidly expanding, field of research. Important challenges still remain to be overcome in this field, some of which are pointed out next.

When the problem considered is high-dimensional and prior knowledge is not available, it is very difficult to design a good parametrization that does not lead to excessive computational costs. An additional, related difficulty arises in the model-free (RL) setting, when only a limited amount of data is available. In this case, if the approximator is too complex, the data may be insufficient to compute its parameters. One alternative to designing the approximator in advance is to find a good parametrization automatically, while another option is to exploit the powerful framework of nonparametric approximators, which can also be viewed as deriving a parametrization from the data. Adaptive and nonparametric approximators are often studied in the context of value iteration and policy iteration (Sections 3.4.3, 3.5.3, and 3.6). In policy search, finding good approximators automatically is a comparatively underexplored but promising idea.

Actions that take continuous values are important in many problems of practical interest. For instance, in the context of automatic control, stabilizing a system around an unstable equilibrium requires continuous actions to avoid chattering, which would otherwise damage the system in the long run. However, in DP and RL, continuous-action problems are more rarely studied than discrete-action problems. A major difficulty of value iteration and policy iteration in the continuous-action case is that they rely on solving many potentially difficult, nonconcave maximization problems over the action variables (Section 3.2). Continuous actions are easier to handle in actor-critic and policy search algorithms, in the sense that explicit maximization over the action variables is not necessary.

Theoretical results about approximate value iteration traditionally rely on the requirement of nonexpansive approximation. To satisfy this requirement, the approximators are often confined to restricted subclasses of linear parameterizations. Analyzing approximate value iteration without assuming nonexpansiveness can be very beneficial, e.g., by allowing powerful nonlinearly parameterized approximators, which may alleviate the difficulties of designing a good parametrization in advance. The work on finite-sample performance guarantees, outlined in Section 3.4.4, provides encouraging results in this direction.

In the context of approximate policy iteration, least-squares techniques for policy evaluation are very promising, owing to their sample efficiency and ease of tuning. However, currently available performance guarantees for these algorithms require that they process relatively many samples generated using a fixed policy. From a learning perspective, it would be very useful to analyze how these techniques behave in online, optimistic policy iteration, in which the policy is not kept fixed for a long time, but is improved once every few samples. Promising empirical results have been reported using such algorithms, but their theoretical understanding is still limited (see Section 3.5.6).

The material in this chapter provides a broad understanding of approximate value iteration, policy iteration, and policy search. In order to deepen and strengthen this understanding, in each of the upcoming three chapters we treat in detail a particular algorithm from one of these three classes. Namely, in Chapter 4, a model-based value iteration algorithm with fuzzy approximation is introduced, theoretically analyzed, and experimentally evaluated. The theoretical analysis illustrates how convergence and consistency guarantees can be developed for approximate DP. In Chapter 5, least-squares policy iteration is revisited, and several extensions to this algorithm are introduced and empirically studied. In particular, an online variant is developed, and some important issues that appear in online RL are emphasized along the way. In Chapter 6, a policy search approach relying on the gradient-free cross-entropy method for optimization is described and experimentally evaluated. This approach highlights one possibility for developing techniques that scale better to high-dimensional state spaces, by focusing the computation only on important initial states.

Bibliography

Åström, K. J., Klein, R. E., and Lennartsson, A. (2005). Bicycle dynamics and control. *IEEE Control Systems Magazine*, 24(4):26–47.

Abonyi, J., Babuška, R., and Szeifert, F. (2001). Fuzzy modeling with multivariate membership functions: Gray-box identification and control design. *IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics*, 31(5):755–767.

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.

Antos, A., Munos, R., and Szepesvári, Cs. (2008a). Fitted Q-iteration in continuous action-space MDPs. In Platt, J. C., Koller, D., Singer, Y., and Roweis, S. T., editors, *Advances in Neural Information Processing Systems 20*, pages 9–16. MIT Press.

Antos, A., Szepesvári, Cs., and Munos, R. (2008b). Learning near-optimal policies with Bellman-residual minimization based fitted policy iteration and a single sample path. *Machine Learning*, 71(1):89–129.

Ascher, U. and Petzold, L. (1998). *Computer methods for ordinary differential equations and differential-algebraic equations*. Society for Industrial and Applied Mathematics (SIAM).

Audibert, J.-Y., Munos, R., and Szepesvári, Cs. (2007). Tuning bandit algorithms in stochastic environments. In *Proceedings 18th International Conference on Algorithmic Learning Theory (ALT-07)*, pages 150–165, Sendai, Japan.

Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite time analysis of multiarmed bandit problems. *Machine Learning*, 47(2–3):235–256.

Auer, P., Jaksch, T., and Ortner, R. (2009). Near-optimal regret bounds for reinforcement learning. In Koller, D., Schuurmans, D., Bengio, Y., and Bottou, L., editors, *Advances in Neural Information Processing Systems 21*, pages 89–96. MIT Press.

Baird, L. (1995). Residual algorithms: Reinforcement learning with function approximation. In *Proceedings 12th International Conference on Machine Learning (ICML-95)*, pages 30–37, Tahoe City, US.

Balakrishnan, S., Ding, J., and Lewis, F. (2008). Issues on stability of ADP feedback controllers for dynamical systems. *IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics*, 4(38):913–917.

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. and Mahadevan, S. (2003). Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems: Theory and Applications*, 13(4):341–379.

Barto, A. G., Sutton, R. S., and Anderson, C. W. (1983). Neuronlike adaptive elements that 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. (2005a). *Dynamic Programming and Optimal Control*, volume 1. Athena Scientific, 3rd edition.

Bertsekas, D. P. (2005b). Dynamic programming and suboptimal control: A survey from ADP to MPC. *European Journal of Control*, 11(4–5):310–334. Special issue for the CDC-ECC-05 in Seville, Spain.

Bertsekas, D. P. (2007). *Dynamic Programming and Optimal Control*, volume 2. Athena Scientific, 3rd edition.

Bertsekas, D. P., Borkar, V., and Nedić, A. (2004). Improved temporal difference methods with linear function approximation. In Si, J., Barto, A., and Powell, W., editors, *Learning and Approximate Dynamic Programming*. IEEE Press.

Bertsekas, D. P. and Castañon, D. A. (1989). Adaptive aggregation methods for infinite horizon dynamic programming. *IEEE Transactions on Automatic Control*, 34(6):589–598.

Bertsekas, D. P. and Ioffe, S. (1996). Temporal differences-based policy iteration and applications in neuro-dynamic programming. Technical Report LIDS-P-2349, Massachusetts Institute of Technology, Cambridge, US. Available at http://web.mit.edu/dimitrib/www/Tempdif.pdf.

Bertsekas, D. P. and Shreve, S. E. (1978). *Stochastic Optimal Control: The Discrete Time Case*. Academic Press.

Bertsekas, D. P. and Tsitsiklis, J. N. (1996). *Neuro-Dynamic Programming*. Athena Scientific.

Bertsekas, D. P. and Yu, H. (2009). Basis function adaptation methods for cost approximation in MDP. In *Proceedings 2009 IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL-09)*, pages 74–81, Nashville, US.

Bethke, B., How, J., and Ozdaglar, A. (2008). Approximate dynamic programming using support vector regression. In *Proceedings 47th IEEE Conference on Decision and Control (CDC-08)*, pages 3811–3816, Cancun, Mexico.

Bhatnagar, S., Sutton, R., Ghavamzadeh, M., and Lee, M. (2009). Natural actor-critic algorithms. *Automatica*, 45(11):2471–2482.

Birge, J. R. and Louveaux, F. (1997). *Introduction to Stochastic Programming.* Springer.

Borkar, V. (2005). An actor-critic algorithm for constrained Markov decision processes. *Systems & Control Letters*, 54(3):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.

Boyan, J. (2002). Technical update: Least-squares temporal difference learning. *Machine Learning*, 49:233–246.

Bradtke, S. J. and Barto, A. G. (1996). Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 22(1–3):33–57.

Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32.

Breiman, L., Friedman, J., Stone, C. J., and Olshen, R. (1984). *Classification and Regression Trees.* Wadsworth International.

Brown, M. and Harris, C. (1994). *Neurofuzzy Adaptive Modeling and Control.* Prentice Hall.

Bubeck, S., Munos, R., Stoltz, G., and Szepesvári, C. (2009). Online optimization in X-armed bandits. In Koller, D., Schuurmans, D., Bengio, Y., and Bottou, L., editors, *Advances in Neural Information Processing Systems 21*, pages 201–208. MIT Press.

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., Ernst, D., De Schutter, B., and Babuška, R. (2007). 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.

Buşoniu, L., Ernst, D., De Schutter, B., and Babuška, R. (2009). Policy search with cross-entropy optimization of basis functions. In *Proceedings 2009 IEEE International Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL-09)*, pages 153–160, Nashville, US.

Camacho, E. F. and Bordons, C. (2004). *Model Predictive Control.* Springer-Verlag.

Cao, X.-R. (2007). *Stochastic Learning and Optimization: A Sensitivity-Based Approach.* Springer.

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. N. (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(5):573–580.

Cristianini, N. and Shawe-Taylor, J. (2000). *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods.* Cambridge University Press.

Davies, S. (1997). Multidimensional triangulation and interpolation for reinforcement learning. In Mozer, M. C., Jordan, M. I., and Petsche, T., editors, *Advances in Neural Information Processing Systems 9*, pages 1005–1011. MIT Press.

Defourny, B., Ernst, D., and Wehenkel, L. (2008). Lazy planning under uncertainties by optimizing decisions on an ensemble of incomplete disturbance trees. In Girgin, S., Loth, M., Munos, R., Preux, P., and Ryabko, D., editors, *Recent Advances in Reinforcement Learning*, volume 5323 of *Lecture Notes in Computer Science*, pages 1–14. Springer.

Defourny, B., Ernst, D., and Wehenkel, L. (2009). Planning under uncertainty, ensembles of disturbance trees and kernelized discrete action spaces. In *Proceedings 2009 IEEE International Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL-09)*, pages 145–152, Nashville, US.

Deisenroth, M. P., Rasmussen, C. E., and Peters, J. (2009). Gaussian process dynamic programming. *Neurocomputing*, 72(7–9):1508–1524.

Dietterich, T. G. (2000). Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303.

Dimitrakakis, C. and Lagoudakis, M. (2008). Rollout sampling approximate policy iteration. *Machine Learning*, 72(3):157–171.

Dorigo, M. and Colombetti, M. (1994). Robot shaping: Developing autonomous agents through learning. *Artificial Intelligence*, 71(2):321–370.

Doya, K. (2000). Reinforcement learning in continuous time and space. *Neural Computation*, 12(1):219–245.

Dupacová, J., Consigli, G., and Wallace, S. W. (2000). Scenarios for multistage stochastic programs. *Annals of Operations Research*, 100(1–4):25–53.

Edelman, A. and Murakami, H. (1995). Polynomial roots from companion matrix eigenvalues. *Mathematics of Computation*, 64:763–776.

Engel, Y., Mannor, S., and Meir, R. (2003). Bayes meets Bellman: The Gaussian process approach to temporal difference learning. In *Proceedings 20th International Conference on Machine Learning (ICML-03)*, pages 154–161, Washington, US.

Engel, Y., Mannor, S., and Meir, R. (2005). Reinforcement learning with Gaussian processes. In *Proceedings 22nd International Conference on Machine Learning (ICML-05)*, pages 201–208, Bonn, Germany.

Ernst, D. (2005). Selecting concise sets of samples for a reinforcement learning agent. In *Proceedings 3rd International Conference on Computational Intelligence, Robotics and Autonomous Systems (CIRAS-05)*, Singapore.

Ernst, D., Geurts, P., and Wehenkel, L. (2005). Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6:503–556.

Ernst, D., Glavic, M., Capitanescu, F., and Wehenkel, L. (2009). Reinforcement learning versus model predictive control: A comparison on a power system problem. *IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics*, 39(2):517–529.

Ernst, D., Glavic, M., Geurts, P., and Wehenkel, L. (2006a). Approximate value iteration in the reinforcement learning context. Application to electrical power system control. *International Journal of Emerging Electric Power Systems*, 3(1). 37 pages.

Ernst, D., Glavic, M., Stan, G.-B., Mannor, S., and Wehenkel, L. (2007). The cross-entropy method for power system combinatorial optimization problems. In *Proceedings of Power Tech 2007*, pages 1290–1295, Lausanne, Switzerland.

Ernst, D., Stan, G.-B., Gonçalves, J., and Wehenkel, L. (2006b). 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.

Fantuzzi, C. and Rovatti, R. (1996). On the approximation capabilities of the homogeneous Takagi-Sugeno model. In *Proceedings 5th IEEE International Conference on Fuzzy Systems (FUZZ-IEEE'96)*, pages 1067–1072, New Orleans, US.

Farahmand, A. M., Ghavamzadeh, M., Szepesvári, Cs., and Mannor, S. (2009a). Regularized fitted Q-iteration for planning in continuous-space Markovian decision problems. In *Proceedings 2009 American Control Conference (ACC-09)*, pages 725–730, St. Louis, US.

Farahmand, A. M., Ghavamzadeh, M., Szepesvári, Cs., and Mannor, S. (2009b). Regularized policy iteration. In Koller, D., Schuurmans, D., Bengio, Y., and Bottou, L., editors, *Advances in Neural Information Processing Systems 21*, pages 441–448. MIT Press.

Feldbaum, A. (1961). Dual control theory, Parts I and II. *Automation and Remote Control*, 21(9):874–880.

Franklin, G. F., Powell, J. D., and Workman, M. L. (1998). *Digital Control of Dynamic Systems*. Prentice Hall, 3rd edition.

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.

Geurts, P., Ernst, D., and Wehenkel, L. (2006). Extremely randomized trees. *Machine Learning*, 36(1):3–42.

Ghavamzadeh, M. and Mahadevan, S. (2007). Hierarchical average reward reinforcement learning. *Journal of Machine Learning Research*, 8:2629–2669.

Glorennec, P. Y. (2000). Reinforcement learning: An overview. In *Proceedings European Symposium on Intelligent Techniques (ESIT-00)*, pages 17–35, Aachen, Germany.

Glover, F. and Laguna, M. (1997). *Tabu Search*. Kluwer.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley.

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. L. 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.

Gordon, G. J. (2001). Reinforcement learning with function approximation converges to a region. In Leen, T. K., Dietterich, T. G., and Tresp, V., editors, *Advances in Neural Information Processing Systems 13*, pages 1040–1046. MIT Press.

Grüne, L. (2004). Error estimation and adaptive discretization for the discrete stochastic Hamilton-Jacobi-Bellman equation. *Numerische Mathematik*, 99(1):85–112.

Hassoun, M. (1995). *Fundamentals of Artificial Neural Networks*. MIT Press.

Hengst, B. (2002). Discovering hierarchy in reinforcement learning with HEXQ. In *Proceedings 19th International Conference on Machine Learning (ICML-02)*, pages 243–250, Sydney, Australia.

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.

Hren, J.-F. and Munos, R. (2008). Optimistic planning of deterministic systems. In Girgin, S., Loth, M., Munos, R., Preux, P., and Ryabko, D., editors, *Recent Advances in Reinforcement Learning*, volume 5323 of *Lecture Notes in Computer Science*, pages 151–164. Springer.

Istratescu, V. I. (2002). *Fixed Point Theory: An Introduction*. Springer.

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.

Jodogne, S., Briquet, C., and Piater, J. H. (2006). Approximate policy iteration for closed-loop learning of visual tasks. In *Proceedings 17th European Conference on Machine Learning (ECML-06)*, volume 4212 of *Lecture Notes in Computer Science*, pages 210–221, Berlin, Germany.

Jones, D. R. (2009). DIRECT global optimization algorithm. In Floudas, C. A. and Pardalos, P. M., editors, *Encyclopedia of Optimization*, pages 725–735. Springer.

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. (2007a). Kernelizing LSPE(λ). In *Proceedings 2007 IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL-07)*, pages 338–345, Honolulu, US.

Jung, T. and Polani, D. (2007b). Learning robocup-keepaway with kernels. In *Gaussian Processes in Practice*, volume 1 of *JMLR Workshop and Conference Proceedings*, pages 33–57.

Jung, T. and Stone, P. (2009). Feature selection for value function approximation using Bayesian model selection. In *Machine Learning and Knowledge Discovery in Databases, European Conference (ECML-PKDD-09)*, volume 5781 of *Lecture Notes in Computer Science*, pages 660–675, Bled, Slovenia.

Jung, T. and Uthmann, T. (2004). Experiments in value function approximation with sparse support vector regression. In *Proceedings 15th European Conference on Machine Learning (ECML-04)*, volume 3201 of *Lecture Notes in Artificial Intelligence*, pages 180–191, Pisa, Italy.

Kaelbling, L. P. (1993). *Learning in Embedded Systems.* MIT Press.

Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1–2):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.

Kakade, S. (2001). A natural policy gradient. In Dietterich, T. G., Becker, S., and Ghahramani, Z., editors, *Advances in Neural Information Processing Systems 14*, pages 1531–1538. MIT Press.

Kalyanakrishnan, S. and Stone, P. (2007). Batch reinforcement learning in a complex domain. In *Proceedings 6th International Conference on Autonomous Agents and Multi-Agent Systems*, pages 650–657, Honolulu, US.

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.

Khalil, H. K. (2002). *Nonlinear Systems*. Prentice Hall, 3rd edition.

Kirk, D. E. (2004). *Optimal Control Theory: An Introduction*. Dover Publications.

Klir, G. J. and Yuan, B. (1995). *Fuzzy Sets and Fuzzy Logic: Theory and Applications*. Prentice Hall.

Knuth, D. E. (1976). Big Omicron and big Omega and big Theta. *SIGACT News*, 8(2):18–24.

Kolter, J. Z. and Ng, A. (2009). Regularization and feature selection in least-squares temporal difference learning. In *Proceedings 26th International Conference on Machine Learning (ICML-09)*, pages 521–528, Montreal, Canada.

Konda, V. (2002). *Actor-Critic Algorithms*. PhD thesis, Massachusetts 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.

Kruse, R., Gebhardt, J. E., and Klowon, F. (1994). *Foundations of Fuzzy Systems*. Wiley.

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.

Levine, W. S., editor (1996). *The Control Handbook*. CRC Press.

Lewis, R. M. and Torczon, V. (2000). Pattern search algorithms for linearly constrained minimization. *SIAM Journal on Optimization*, 10(3):917–941.

Li, L., Littman, M. L., and Mansley, C. R. (2009). Online exploration in least-squares policy iteration. In *Proceedings 8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-09)*, volume 2, pages 733–739, Budapest, Hungary.

Lin, C.-K. (2003). A reinforcement learning adaptive fuzzy controller for robots. *Fuzzy Sets and Systems*, 137(3):339–352.

Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3–4):293–321. Special issue on reinforcement learning.

Liu, D., Javaherian, H., Kovalenko, O., and Huang, T. (2008). Adaptive critic learning techniques for engine torque and air-fuel ratio control. *IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics*, 38(4):988–993.

Lovejoy, W. S. (1991). Computationally feasible bounds for partially observed Markov decision processes. *Operations Research*, 39(1):162–175.

Maciejowski, J. M. (2002). *Predictive Control with Constraints.* Prentice Hall.

Madani, O. (2002). On policy iteration as a Newton's method and polynomial policy iteration algorithms. In *Proceedings 18th National Conference on Artificial Intelligence and 14th Conference on Innovative Applications of Artificial Intelligence AAAI/IAAI-02*, pages 273–278, Edmonton, Canada.

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.

Mamdani, E. (1977). Application of fuzzy logic to approximate reasoning using linguistic systems. *IEEE Transactions on Computers*, 26:1182–1191.

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(1–2):111–148.

Matarić, M. J. (1997). Reinforcement learning in the multi-robot domain. *Autonomous Robots*, 4(1):73–83.

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.

Melo, F. S., Meyn, S. P., and Ribeiro, M. I. (2008). An analysis of reinforcement learning with function approximation. In *Proceedings 25th International Conference on Machine Learning (ICML-08)*, pages 664–671, Helsinki, Finland.

Menache, I., Mannor, S., and Shimkin, N. (2005). Basis function adaptation in temporal difference reinforcement learning. *Annals of Operations Research*, 134(1):215–238.

Millán, J. d. R., Posenato, D., and Dedieu, E. (2002). Continuous-action Q-learning. *Machine Learning*, 49(2–3):247–265.

Moore, A. W. and Atkeson, C. R. (1995). The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. *Machine Learning*, 21(3):199–233.

Morris, C. (1982). Natural exponential families with quadratic variance functions. *Annals of Statistics*, 10(1):65–80.

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)*, volume 1224 of *Lecture Notes in Artificial Intelligence*, 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, Cs. (2008). Finite time bounds for fitted value iteration. *Journal of Machine Learning Research*, 9:815–857.

Murphy, S. (2005). A generalization error for Q-learning. *Journal of Machine Learning Research*, 6:1073–1097.

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(6):723–735.

Nedić, A. and Bertsekas, D. P. (2003). Least-squares policy evaluation algorithms with linear function approximation. *Discrete Event Dynamic Systems: Theory and Applications*, 13(1–2):79–110.

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.

Panait, L. and Luke, S. (2005). Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11(3):387–434.

Parr, R., Li, L., Taylor, G., Painter-Wakefield, C., and Littman, M. (2008). An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning. In *Proceedings 25th Annual International Conference on Machine Learning (ICML-08)*, pages 752–759, Helsinki, Finland.

Pazis, J. and Lagoudakis, M. (2009). Binary action search for learning continuous-action control policies. In *Proceedings of the 26th Annual International Conference on Machine Learning (ICML-09)*, pages 793–800, Montreal, Canada.

Pérez-Uribe, A. (2001). Using a time-delay actor-critic neural architecture with dopamine-like reinforcement signal for learning in autonomous robots. In Wermter, S., Austin, J., and Willshaw, D. J., editors, *Emergent Neural Computational Architectures Based on Neuroscience*, volume 2036 of *Lecture Notes in Computer Science*, pages 522–533. Springer.

Perkins, T. and Barto, A. (2002). Lyapunov design for safe reinforcement learning. *Journal of Machine Learning Research*, 3:803–832.

Peters, J. and Schaal, S. (2008). Natural actor-critic. *Neurocomputing*, 71(7–9):1180–1190.

Pineau, J., Gordon, G. J., and Thrun, S. (2006). Anytime point-based approximations for large POMDPs. *Journal of Artificial Intelligence Research (JAIR)*, 27:335–380.

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.

Powell, W. B. (2007). *Approximate Dynamic Programming: Solving the Curses of Dimensionality*. Wiley.

Press, W. H., Flannery, B. P., Teukolsky, S. A., and Vetterling, W. T. (1986). *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press.

Prokhorov, D. and Wunsch, D.C., I. (1997). Adaptive critic designs. *IEEE Transactions on Neural Networks*, 8(5):997–1007.

Puterman, M. L. (1994). *Markov Decision Processes—Discrete Stochastic Dynamic Programming*. Wiley.

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.

Rasmussen, C. E. and Kuss, M. (2004). Gaussian processes in reinforcement learning. In Thrun, S., Saul, L. K., and Schölkopf, B., editors, *Advances in Neural Information Processing Systems 16*. MIT Press.

Rasmussen, C. E. and Williams, C. K. I. (2006). *Gaussian Processes for Machine Learning*. MIT Press.

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.

Reynolds, S. I. (2000). Adaptive resolution model-free reinforcement learning: Decision boundary partitioning. In *Proceedings Seventeenth International Conference on Machine Learning (ICML-00)*, pages 783–790, Stanford University, US.

Riedmiller, M. (2005). Neural fitted Q-iteration – first experiences with a data efficient neural reinforcement learning method. In *Proceedings 16th European Conference on Machine Learning (ECML-05)*, volume 3720 of *Lecture Notes in Computer Science*, pages 317–328, Porto, Portugal.

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, US.

Rubinstein, R. Y. and Kroese, D. P. (2004). *The Cross Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation, and Machine Learning*. Springer.

Rummery, G. A. and Niranjan, M. (1994). On-line Q-learning using connectionist systems. Technical Report CUED/F-INFENG/TR166, Engineering Department, Cambridge University, UK. Available at http://mi.eng.cam.ac.uk/reports/svr-ftp/rummery_tr166.ps.Z.

Russell, S. and Norvig, P. (2003). *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2nd edition.

Russell, S. J. and Zimdars, A. (2003). Q-decomposition for reinforcement learning agents. In *Proceedings 20th International Conference of Machine Learning (ICML-03)*, pages 656–663, Washington, US.

Santamaria, J. C., Sutton, R. S., and Ram, A. (1998). Experiments with reinforcement learning in problems with continuous state and action spaces. *Adaptive Behavior*, 6(2):163–218.

Santos, M. S. and Vigo-Aguiar, J. (1998). Analysis of a numerical dynamic programming algorithm applied to economic models. *Econometrica*, 66(2):409–426.

Schervish, M. J. (1995). *Theory of Statistics*. Springer.

Schmidhuber, J. (2000). Sequential decision making based on direct search. In Sun, R. and Giles, C. L., editors, *Sequence Learning*, volume 1828 of *Lecture Notes in Computer Science*, pages 213–240. Springer.

Schölkopf, B., Burges, C., and Smola, A. (1999). *Advances in Kernel Methods: Support Vector Learning*. MIT Press.

Shawe-Taylor, J. and Cristianini, N. (2004). *Kernel Methods for Pattern Analysis*. Cambridge University Press.

Sherstov, A. and Stone, P. (2005). Function approximation via tile coding: Automating parameter choice. In *Proceedings 6th International Symposium on Abstraction, Reformulation and Approximation (SARA-05)*, volume 3607 of *Lecture Notes in Computer Science*, pages 194–205, Airth Castle, UK.

Shoham, Y., Powers, R., and Grenager, T. (2007). If multi-agent learning is the answer, what is the question? *Artificial Intelligence*, 171(7):365–377.

Singh, S., Jaakkola, T., Littman, M. L., and Szepesvári, Cs. (2000). Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning*, 38(3):287–308.

Singh, S. and Sutton, R. (1996). Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22(1–3):123–158.

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. MIT Press.

Singh, S. P., James, M. R., and Rudary, M. R. (2004). Predictive state representations: A new theory for modeling dynamical systems. In *Proceedings 20th Conference in Uncertainty in Artificial Intelligence (UAI-04)*, pages 512–518, Banff, Canada.

Smola, A. J. and Schölkopf, B. (2004). A tutorial on support vector regression. *Statistics and Computing*, 14(3):199–222.

Sutton, R., Maei, H., Precup, D., Bhatnagar, S., Silver, D., Szepesvari, Cs., and Wiewiora, E. (2009a). Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *Proceedings 26th International Conference on Machine Learning (ICML-09)*, pages 993–1000, Montreal, Canada.

Sutton, R. S. (1988). Learning to predict by the method 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. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. In Touretzky, D. S., Mozer, M. C., and Hasselmo, M. E., editors, *Advances in Neural Information Processing Systems 8*, pages 1038–1044. MIT Press.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction.* MIT Press.

Sutton, R. S., Barto, A. G., and Williams, R. J. (1992). Reinforcement learning is adaptive optimal control. *IEEE Control Systems Magazine*, 12(2):19–22.

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.

Sutton, R. S., Szepesvári, Cs., and Maei, H. R. (2009b). A convergent $O(n)$ temporal-difference algorithm for off-policy learning with linear function approximation. In Koller, D., Schuurmans, D., Bengio, Y., and Bottou, L., editors, *Advances in Neural Information Processing Systems 21*, pages 1609–1616. MIT Press.

Szepesvári, Cs. and Munos, R. (2005). Finite time bounds for sampling based fitted value iteration. In *Proceedings 22nd International Conference on Machine Learning (ICML-05)*, pages 880–887, Bonn, Germany.

Szepesvári, Cs. and Smart, W. D. (2004). Interpolation-based Q-learning. In *Proceedings 21st International Conference on Machine Learning (ICML-04)*, pages 791–798, Banff, 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(1):116–132.

Taylor, G. and Parr, R. (2009). Kernelized value function approximation for reinforcement learning. In *Proceedings 26th International Conference on Machine Learning (ICML-09)*, pages 1017–1024, Montreal, Canada.

Thrun, S. (1992). The role of exploration in learning control. In White, D. and Sofge, D., editors, *Handbook for Intelligent Control: Neural, Fuzzy and Adaptive Approaches.* Van Nostrand Reinhold.

Torczon, V. (1997). On the convergence of pattern search algorithms. *SIAM Journal on Optimization*, 7(1):1–25.

Touzet, C. F. (1997). Neural reinforcement learning for behaviour synthesis. *Robotics and Autonomous Systems*, 22(3–4):251–281.

Tsitsiklis, J. N. (1994). Asynchronous stochastic approximation and Q-learning. *Machine Learning*, 16(1):185–202.

Tsitsiklis, J. N. (2002). On the convergence of optimistic policy iteration. *Journal of Machine Learning Research*, 3:59–72.

Tsitsiklis, J. N. and Van Roy, B. (1996). Feature-based methods for large scale dynamic programming. *Machine Learning*, 22(1–3):59–94.

Tsitsiklis, J. N. and Van Roy, B. (1997). An analysis of temporal difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42(5):674–690.

Tuyls, K., Maes, S., and Manderick, B. (2002). Q-learning in simulated robotic soccer – large state spaces and incomplete information. In *Proceedings 2002 International Conference on Machine Learning and Applications (ICMLA-02)*, pages 226–232, Las Vegas, US.

Uther, W. T. B. and Veloso, M. M. (1998). Tree based discretization for continuous state space reinforcement learning. In *Proceedings 15th National Conference on Artificial Intelligence and 10th Innovative Applications of Artificial Intelligence Conference (AAAI-98/IAAI-98)*, pages 769–774, Madison, US.

Vrabie, D., Pastravanu, O., Abu-Khalaf, M., and Lewis, F. (2009). Adaptive optimal control for continuous-time linear systems based on policy iteration. *Automatica*, 45(2):477–484.

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, UK.

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)*, volume 3201 of *Lecture Notes in Artificial Intelligence*, 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.

Xu, X., Xie, T., Hu, D., and Lu, X. (2005). Kernel least-squares temporal difference learning. *International Journal of Information Technology*, 11(9):54–63.

Yen, J. and Langari, R. (1999). *Fuzzy Logic: Intelligence, Control, and Information.* Prentice Hall.

Yu, H. and Bertsekas, D. P. (2006). Convergence results for some temporal difference methods based on least-squares. Technical Report LIDS 2697, Massachusetts Institute of Technology, Cambridge, US. Available at http://www.mit.edu/people/dimitrib/lspe_lids_final.pdf.

Yu, H. and Bertsekas, D. P. (2009). Convergence results for some temporal difference methods based on least squares. *IEEE Transactions on Automatic Control*, 54(7):1515–1531.