5

Infinite Horizon Reinforcement Learning

<table>
<thead>
<tr>
<th>Contents</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1. Approximation in Value Space - Performance Bounds</td>
<td>p. 2</td>
</tr>
<tr>
<td>5.1.1. Limited Lookahead</td>
<td>p. 4</td>
</tr>
<tr>
<td>5.1.2. Rollout</td>
<td>p. 7</td>
</tr>
<tr>
<td>5.1.3. Approximate Policy Iteration</td>
<td>p. 11</td>
</tr>
<tr>
<td>5.2. Fitted Value Iteration</td>
<td>p. 14</td>
</tr>
<tr>
<td>5.3. Simulation-Based Policy Iteration with Parametric</td>
<td></td>
</tr>
<tr>
<td>5.3.1. Self-Learning and Actor-Critic Systems</td>
<td>p. 18</td>
</tr>
<tr>
<td>5.3.2. A Model-Based Variant</td>
<td>p. 20</td>
</tr>
<tr>
<td>5.3.3. A Model-Free Variant</td>
<td>p. 22</td>
</tr>
<tr>
<td>5.3.4. Implementation Issues of Parametric Policy Iteration</td>
<td>p. 24</td>
</tr>
<tr>
<td>5.3.5. Convergence Issues of Parametric Policy Iteration - Oscillations</td>
<td>p. 28</td>
</tr>
<tr>
<td>5.4. Q-Learning</td>
<td>p. 32</td>
</tr>
<tr>
<td>5.4.1. Optimistic Policy Iteration with Parametric Q-Factor</td>
<td></td>
</tr>
<tr>
<td>5.5. Additional Methods - Temporal Differences</td>
<td>p. 35</td>
</tr>
<tr>
<td>5.6. Exact and Approximate Linear Programming</td>
<td>p. 46</td>
</tr>
<tr>
<td>5.7. Approximation in Policy Space</td>
<td>p. 49</td>
</tr>
<tr>
<td>5.7.1. Training by Cost Optimization - Policy Gradient,</td>
<td></td>
</tr>
<tr>
<td>5.7.2. Expert Supervised Training</td>
<td>p. 60</td>
</tr>
<tr>
<td>5.7.3. Approximate Policy Iteration, Rollout, and Approximation in Policy Space</td>
<td>p. 62</td>
</tr>
<tr>
<td>5.8. Notes and Sources</td>
<td>p. 64</td>
</tr>
<tr>
<td>5.9. Appendix: Mathematical Analysis</td>
<td>p. 67</td>
</tr>
<tr>
<td>5.9.1. Performance Bounds for One-Step Lookahead</td>
<td>p. 68</td>
</tr>
<tr>
<td>5.9.2. Performance Bounds for Rollout</td>
<td>p. 70</td>
</tr>
<tr>
<td>5.9.3. Performance Bounds for Approximate Policy Iteration</td>
<td>p. 73</td>
</tr>
</tbody>
</table>
In this chapter, we consider the use of approximate DP/RL methods for suboptimal solution of the infinite horizon SSP and discounted problems of the preceding chapter. In particular, we will discuss approximate versions of the value iteration (VI) and policy iteration (PI) methods that we discussed in Sections 4.4 and 4.5. In the process, we will make frequent references to the DP operators $T$ and $T_{\mu}$ (or Bellman operators in the RL terminology), which map an $n$-dimensional vector $J$ into the $n$-dimensional vectors $TJ$ and $T_{\mu}J$, and provide shorthand notation for algorithms and analysis. For the purpose of easy reference, we state these operators here:

For SSP problems:

$$(TJ)(i) = \min_{u \in U(i)} \left[ p_{\mu}(u)g(i, u, t) + \sum_{j=1}^{n} p_{ij}(u) \left( g(i, u, j) + J(j) \right) \right], \quad (5.1)$$

for all $i = 1, \ldots, n$, and

$$(T_{\mu}J)(i) = p_{\mu}(i)g(i, \mu(i), t) + \sum_{j=1}^{n} p_{ij}(\mu(i)) \left( g(i, \mu(i), j) + J(j) \right), \quad (5.2)$$

for all policies $\mu$ and states $i = 1, \ldots, n$.

For discounted problems:

$$(TJ)(i) = \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u) \left( g(i, u, j) + \alpha J(j) \right), \quad i = 1, \ldots, n, \quad (5.3)$$

and

$$(T_{\mu}J)(i) = \sum_{j=1}^{n} p_{ij}(\mu(i)) \left( g(i, \mu(i), j) + \alpha J(j) \right), \quad i = 1, \ldots, n, \quad (5.4)$$

for all policies $\mu$ and states $i = 1, \ldots, n$.

### 5.1 APPROXIMATION IN VALUE SPACE - PERFORMANCE BOUNDS

In this section we will discuss the general framework for approximation in value space for infinite horizon DP, beginning with discounted problems. Consistently with the corresponding finite horizon schemes of Chapter 2, the general idea is to compute some approximation $\tilde{J}$ of the optimal cost function $J^*$, and then use one-step or multistep lookahead to implement a suboptimal policy $\tilde{\mu}$. Thus, a one-step lookahead policy applies at state $i$ the control $\tilde{\mu}(i)$ that attains the minimum in the expression

$$\min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u) \left( g(i, u, j) + \alpha \tilde{J}(j) \right), \quad (5.5)$$
Approximate minimization

\[
\min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u) \left( g(i, u, j) + \alpha \hat{J}(j) \right)
\]

First Step “Future”

Approximations:
- Replace \( E[\cdot] \) with nominal values (certainty equivalence)
- Adaptive simulation
- Monte Carlo tree search

Computation of \( \tilde{J} \):
- Problem approximation
- Rollout
- Approximate PI
- Parametric approximation
- Aggregation

\textbf{Figure 5.1.1} Schematic illustration of various options for approximation in value space with one-step lookahead in infinite horizon problems. The lookahead function values \( \tilde{J}(j) \) approximate the optimal cost-to-go values \( J^*(j) \), and can be computed by a variety of methods. There may be additional approximations in the minimization over \( u_k \) and the computation of the expected value.

see Fig. 5.1.1.

Similarly, at state \( i \), a two-step lookahead policy applies the control \( \tilde{\mu}(i) \) attaining the minimum in the preceding equation, where now \( \tilde{J} \) is obtained itself on the basis of a one-step lookahead approximation. In other words, for all states \( j \) that can be reached from \( i \), we have

\[
\tilde{J}(j) = \min_{u \in U(j)} \sum_{m=1}^{n} p_{jm}(u) \left( g(j, u, m) + \alpha \hat{J}(m) \right),
\]

where \( \tilde{J} \) is some approximation of \( J^* \). Thus \( \tilde{J} \) is the result of a single value iteration starting from \( \tilde{J} \). Policies with lookahead of more than two stages are similarly defined. In particular, the “effective one-step” cost approximation \( \tilde{J} \) in \( \ell \)-step lookahead is the result of \( \ell - 1 \) successive value iterations starting from some initial approximation \( \tilde{J} \). Otherwise expressed, \( \ell \)-step lookahead with \( \tilde{J} \) at the end is the same as one-step lookahead with \( T^{\ell-1} \tilde{J} \) at the end, where \( T \) is the DP operator.

In Chapter 2 we gave several types of limited lookahead schemes, where \( \tilde{J} \) is obtained in different ways, such as problem approximation, rollout, and others. Several of these schemes can be fruitfully adapted to infinite horizon problems; see Fig. 5.1.1. For example, the problem approximation approaches of Section 2.3 admit straightforward extensions to infinite horizon settings, whereby the function \( \tilde{J}(j) \) in Eq. (5.5) is obtained by solving exactly an infinite horizon (or even finite horizon) problem that is related to the original in some way, such as forms of certainty equivalence.
In this chapter, we will focus on approximate PI schemes, which operate as follows:

(a) Several policies $\mu^0, \mu^1, \ldots, \mu^m$ are generated, starting with an initial policy $\mu^0$.

(b) Each policy $\mu^k$ is evaluated approximately, with a cost function $\tilde{J}_{\mu^k}$, often with the use of a parametric approximation/neural network approach.

(c) The next policy $\mu^{k+1}$ is generated by one-step or multistep policy improvement based on $\tilde{J}_{\mu^k}$.

(d) The approximate evaluation $\tilde{J}_{\mu^m}$ of the last policy in the sequence is used as the lookahead approximation $\tilde{J}$ in the one-step lookahead minimization (5.5), or its multistep counterpart.

Performance bounds for this type of approximate PI scheme will be discussed in Section 5.1.3, following a discussion of general performance bounds and rollout in the next two subsections. Note that rollout can be viewed as the extreme special case of the preceding approximate PI procedure, where $m = 0$, and only the policy $\mu^0$ is evaluated and used for a single policy improvement. The aggregation approach will be discussed in Chapter 6.

5.1.1 Limited Lookahead

We will now consider performance bounds for $\ell$-step lookahead. In particular, if $\tilde{\mu}_0, \ldots, \tilde{\mu}_{\ell-1}$ attain the minimum in the $\ell$-step lookahead minimization below:

$$\min_{\mu_0, \ldots, \mu_{\ell-1}} E \left\{ \sum_{k=0}^{\ell-1} \alpha^k g(i_k, \mu_k(i_k), j_k) + \alpha^\ell \tilde{J}(i_\ell) \right\},$$

we focus on the suboptimal policy $\tilde{\mu} = \tilde{\mu}_0$. We will refer to $\tilde{\mu}$ as the $\ell$-step lookahead policy corresponding to $\tilde{J}$. Equivalently, in the shorthand notation of the DP operators $T$ and $T_{\tilde{\mu}}$ of Eqs. (5.3) and (5.4), the $\ell$-step lookahead policy $\tilde{\mu}$ is defined by

$$T_{\tilde{\mu}}(T^{\ell-1} \tilde{J}) = T^\ell \tilde{J}.$$  

In part (a) of the following proposition, we will derive a bound for the performance of $\tilde{\mu}$.

We will also derive a bound for the case of a useful generalized one-step lookahead scheme [part (b) of the following proposition]. This scheme aims to reduce the computation to obtain $\tilde{\mu}(i)$, by performing the lookahead minimization over a subset $\tilde{U}(i) \subset U(i)$. Thus, the control $\tilde{\mu}(i)$ used in this scheme is one that attains the minimum in the expression

$$\min_{u \in \tilde{U}(i)} \sum_{j=1}^{n} p_{ij}(u)(g(i, u, j) + \alpha \tilde{J}(j)).$$
This is attractive when by using some heuristic or approximate optimization, we can identify a subset \( U(i) \) of promising controls, and to save computation, we restrict attention to this subset in the one-step lookahead minimization.

**Proposition 5.1.1: (Limited Lookahead Performance Bounds)**

(a) Let \( \tilde{\mu} \) be the \( \ell \)-step lookahead policy corresponding to \( \tilde{J} \). Then

\[
\| J_{\tilde{\mu}} - J^* \| \leq \frac{2\alpha\ell}{1 - \alpha} \| \tilde{J} - J^* \|, \tag{5.6}
\]

where \( \| \cdot \| \) denotes the maximum norm \( \| J \| = \max_{i=1,...,n} |J(i)| \).

(b) Let \( \tilde{\mu} \) be the one-step lookahead policy obtained by minimization in the equation

\[
\hat{J}(i) = \min_{u \in \overline{U}(i)} \sum_{j=1}^{n} p_{ij}(u)(g(i, u, j) + \alpha \tilde{J}(j)), \quad i = 1, \ldots, n, \tag{5.7}
\]

where \( \overline{U}(i) \subset U(i) \) for all \( i = 1, \ldots, n \). Assume that for some constant \( c \), we have

\[
\hat{J}(i) \leq \tilde{J}(i) + c, \quad i = 1, \ldots, n. \tag{5.8}
\]

Then

\[
J_{\tilde{\mu}}(i) \leq \hat{J}(i) + \frac{c}{1 - \alpha}, \quad i = 1, \ldots, n. \tag{5.9}
\]

An important point regarding the bound (5.6) is that \( J_{\tilde{\mu}} \) is unaffected by a constant shift in \( \tilde{J} \) [an addition of a constant \( \beta \) to all values \( \tilde{J}(i) \)]. Thus \( \| J - J^* \| \) in Eq. (5.6) can be replaced by the potentially much smaller number

\[
\min_{\beta \in \mathbb{R}} \max_{i=1,...,n} |\tilde{J}(i) + \beta - J^*(i)|.
\]

We thus obtain the following performance bound,

\[
\| J_{\tilde{\mu}} - J^* \| \leq \frac{2\alpha\ell}{1 - \alpha} \min_{\beta \in \mathbb{R}} \max_{i=1,...,n} |\tilde{J}(i) + \beta - J^*(i)|,
\]

which is stronger than the one of Eq. (5.6) that corresponds to \( \beta = 0 \).

The preceding bound suggests that performance is improved when the length \( \ell \) of the lookahead is increased, and also when the lookahead cost approximation \( \tilde{J} \) is closer to the optimal cost \( J^* \) (when modified with an optimal constant shift \( \beta \)). Both of these conclusions are intuitive and also
consistent with practical experience. Note that we are not guaranteeing that multistep lookahead will lead to better performance than one-step lookahead; we know that this not is necessarily true (cf. Example 2.2.1). It is the performance bound that is improved when multistep lookahead is used.

Regarding the condition (5.8), we note that it guarantees that when \( c \leq 0 \), the cost \( J_{\tilde{\mu}} \) of the one-step lookahead policy is no larger than \( J \). When \( c = 0 \), this condition bears resemblance with the consistent improvement condition for deterministic rollout methods (cf. Section 2.4.1). If \( J = J_{\mu} \) for some policy \( \mu \) (as in the case of the pure form of rollout to be discussed in Section 5.1.2), then Eq. (5.8) holds as an equation with \( c = 0 \), and from Eq. (5.9), it follows that \( J_{\tilde{\mu}} \leq J_{\mu} \).

Unfortunately, the bound (5.6) is not very reassuring when \( \alpha \) is close to 1. Nonetheless, the following example shows that the bound is tight in very simple problems with just two states. What is happening here is that an \( O(\epsilon) \) difference in single stage cost between two controls can generate an \( O(\epsilon/(1-\alpha)) \) difference in policy costs, yet it can be “nullified” in Bellman’s equation by an \( O(\epsilon) \) difference between \( J^* \) and \( \tilde{J} \).

**Example 5.1.1**

Consider the two-state discounted problem shown in Fig. 5.1.2, where \( \epsilon \) is a positive scalar and \( \alpha \in [0,1) \) is the discount factor. The optimal policy \( \mu^* \) is to move from state 1 to state 2, and the optimal cost-to-go function is \( J^*(1) = J^*(2) = 0 \). Consider the cost function approximation \( \tilde{J} \) with

\[
\tilde{J}(1) = -\epsilon, \quad \tilde{J}(2) = \epsilon,
\]

so that

\[
\| \tilde{J} - J^* \| = \epsilon,
\]
as assumed in Eq. (5.6) [cf. Prop. 5.1.1(a)]. The policy \( \mu \) that decides to stay at state 1 is a one-step lookahead policy based on \( \tilde{J} \), because

\[
2\alpha \epsilon + \alpha \tilde{J}(1) = \alpha \epsilon = 0 + \alpha \tilde{J}(2).
\]

Moreover, we have

\[
J_\mu(1) = \frac{2\alpha \epsilon}{1 - \alpha} = \frac{2\alpha}{1 - \alpha} \| \tilde{J} - J^* \|
\]

so the bound of Eq. (5.6) holds with equality.

5.1.2 Rollout

Let us first consider rollout in its pure form, where \( \tilde{J} \) in Eq. (5.5) is the cost function of some stationary policy \( \mu \) (also called the base policy or base heuristic), i.e., \( \tilde{J} = J_\mu \). Thus, the rollout policy is the result of a single policy iteration starting from \( \mu \). The policy evaluation that yields the costs \( J_\mu(j) \) needed for policy improvement may be done in any suitable way. Monte-Carlo simulation (averaging the costs of many trajectories starting from \( j \)) is one major possibility. Of course if the problem is deterministic, a single simulation trajectory starting from \( j \) is sufficient, in which case the rollout policy is much less computationally demanding. Note also that in discounted problems the simulated trajectories can be truncated after a number of transitions, which is sufficiently large to make the cost of the remaining transitions insignificant in view of the discount factor.

An important fact is that in the pure form of rollout, the rollout policy improves over the base policy, as shown by the following proposition [a special case of Prop. 5.1.1(b) as noted earlier]. This is to be expected since rollout is one-step PI, so the general policy improvement property of PI applies.

**Proposition 5.1.2: (Cost Improvement by Rollout)** Let \( \tilde{\mu} \) be the rollout policy obtained by the one-step lookahead minimization

\[
\min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u) \left( g(i, u, j) + \alpha J_\mu(j) \right),
\]

where \( \mu \) is a base policy [cf. Eq. (5.7) with \( \tilde{J} = J_\mu \)] and we assume that \( \mu(i) \in U(i) \subset U(i) \) for all \( i = 1, \ldots, n \). Then \( J_\tilde{\mu} \leq J_\mu \).

Let us also mention the variation of rollout that uses multiple base heuristics, and simultaneously improves on all of them. This variant, also called parallel rollout because of its evident parallelization potential, extends its finite horizon counterpart; cf. Section 2.4.1.
Example 5.1.2 (Rollout with Multiple Heuristics)

Let $\mu_1, \ldots, \mu_M$ be stationary policies, let
\[
\tilde{J}(i) = \min \{ J_{\mu_1}(i), \ldots, J_{\mu_M}(i) \}, \quad i = 1, \ldots, n,
\]
let $U(i) \subset U(i)$, and assume that
\[
\mu_1(i), \ldots, \mu_M(i) \in U(i), \quad i = 1, \ldots, n.
\]
Then, for all $i$ and $m = 1, \ldots, M$, we have
\[
\hat{J}(i) = \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u) \left( g(i, u, j) + \alpha \tilde{J}(j) \right)
\]
\[
\leq \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u) \left( g(i, u, j) + \alpha J_{\mu_m}(j) \right)
\]
\[
\leq \sum_{j=1}^{n} p_{ij}(\mu_m(i)) \left( g(i, \mu_m(i), j) + \alpha J_{\mu_m}(j) \right)
\]
\[
= J_{\mu_m}(i),
\]
from which, by taking minimum of the right-hand side over $m$, it follows that
\[
\hat{J}(i) \leq \tilde{J}(i), \quad i = 1, \ldots, n.
\]
Using Prop. 5.1.1(b), we see that the rollout policy $\tilde{\mu}$, obtained by using $\tilde{J}$ as one-step lookahead approximation satisfies
\[
J_{\tilde{\mu}}(i) \leq \min \{ J_{\mu_1}(i), \ldots, J_{\mu_M}(i) \}, \quad i = 1, \ldots, n,
\]
i.e., it improves over each of the policies $\mu_1, \ldots, \mu_M$.

Truncated Rollout with Multistep Lookahead and Terminal Cost Function Approximation

Let us next discuss a truncated variant of the rollout approach, whereby we use $\ell$-step lookahead, we then apply rollout with policy $\mu$ for a limited number of steps, and finally we approximate the cost of the remaining steps using some terminal cost approximation $\tilde{J}$; see Fig. 5.1.3. We can view this form of rollout as a single optimistic policy iteration combined with multistep lookahead. This type of algorithm was used in Tesauro’s rollout-based backgammon player [TeG96] (it was also used in AlphaGo in a modified form, with Monte Carlo tree search in place of ordinary limited lookahead). We will give more details later.
The following proposition generalizes the performance bounds given for limited lookahead (cf. Props. 5.1.1). In particular, part (a) of the proposition follows by applying Prop. 5.1.1(a), since the truncated rollout scheme of this section can be viewed as $\ell$-step approximation in value space with terminal cost function $T^m_{\mu} \tilde{J}$ at the end of the lookahead, where $T_{\mu}$ is the DP operator corresponding to $\mu$.

**Proposition 5.1.3: (Performance Bound of Truncated Rollout with Terminal Cost Function Approximation)** Let $\ell$ and $m$ be positive integers, let $\mu$ be a policy, and let $\tilde{J}$ be a function of the state. Consider a truncated rollout scheme consisting of $\ell$-step lookahead, followed by rollout with a policy $\mu$ for $m$ steps, and a terminal cost function approximation $\tilde{J}$ at the end of the $m$ steps. Let $\tilde{\mu}$ be the policy generated by this scheme.
(a) We have
\[ \| J_{\tilde{\mu}} - J^* \| \leq \frac{2\alpha\ell}{1 - \alpha} \| T_{\mu}^m \tilde{J} - J^* \|, \]
where \( T_{\mu} \) is the DP operator, and \( \| \cdot \| \) denotes the maximum norm \( \| J \| = \max_{i=1,\ldots,n} |J(i)| \).

(b) Assume that for some constant \( c \), \( \tilde{J} \) and \( \mu \) satisfy the condition
\[ \hat{J}(i) = \sum_{j=1}^{n} p_{ij}(\mu(i)) \left( g(i,\mu(i),j) + \alpha \tilde{J}(j) \right) \leq \tilde{J}(i) + c, \quad (5.10) \]
for all \( i = 1, \ldots, n \). Then
\[ J_{\tilde{\mu}}(i) \leq \tilde{J}(i) + \frac{c}{1 - \alpha}, \quad i = 1, \ldots, n. \quad (5.11) \]

(c) Assume that for some constant \( c \), \( \tilde{J} \) and \( \mu \) satisfy
\[ \| \tilde{J} - J_{\mu} \| \leq \frac{c}{1 + \alpha}. \]
Then
\[ J_{\tilde{\mu}}(i) \leq J_{\mu}(i) + \frac{2c}{1 - \alpha^2}, \quad i = 1, \ldots, n. \]

Note the qualitative implication of part (c) of the preceding proposition: if \( \tilde{J} \) is nearly equal to \( J_{\mu} \), then \( \tilde{\mu} \) nearly improves over \( \mu \). Note also that part (b) is essentially Prop. 5.1.1(b) for the case where \( m = 0 \) (i.e., there is no rollout with a policy \( \mu \)).

Regarding the nature of the terminal cost approximation \( \tilde{J} \) in truncated rollout schemes, it may be heuristic, based on problem approximation, or based on a more systematic simulation methodology. For example, the values \( J_{\mu}(i) \) may be computed by simulation for all \( i \) in a subset of representative states, and \( \tilde{J} \) may be selected from a parametric class of vectors by a least squares regression of the computed values. This approximation may be performed off-line, outside the time-sensitive restrictions of a real-time implementation, and the result \( \tilde{J} \) may be used on-line in place of \( J_{\mu} \) as a terminal cost function approximation. Note that a good choice of terminal cost approximation is crucial in SSP problems when most or all of the cost is incurred upon reaching the termination state. Note also that once cost function approximation is introduced at the end of the rollout, the cost improvement property of the rollout policy over the base policy may be lost [cf. Prop. 5.1.3(c)].
The truncated rollout scheme of Fig. 5.1.3 has been adopted in the rollout backgammon algorithm of Tesauro and Galperin [TeG96]. The policy \( \mu \) and the terminal cost function approximation \( \hat{J} \) provided by the TD-Gammon algorithm of Tesauro [Tes94], which was based on a neural network, trained using a form of optimistic policy iteration and TD(\( \lambda \)). A similar type of algorithm was used in the AlphaGo program (Silver et al. [SHM16]), with the policy and the terminal cost function obtained with a deep neural network, trained using a form of approximate policy iteration. Also the multistep lookahead in the AlphaGo algorithm was implemented using Monte Carlo tree search (cf. Section 2.4.2).

5.1.3 Approximate Policy Iteration

When the number of states is very large, the policy evaluation step and/or the policy improvement step of the PI method may be implementable only through approximations. In an approximate PI scheme, each policy \( \mu^k \) is evaluated approximately, with a cost function \( \hat{J}_{\mu^k} \), often with the use of a feature-based architecture or a neural network, and the next policy \( \mu^{k+1} \) is generated by (perhaps approximate) policy improvement based on \( \hat{J}_{\mu^k} \).

To formalize this type of procedure, we assume an approximate policy evaluation error satisfying

\[
\max_{i=1,\ldots,n} |\hat{J}_{\mu^k}(i) - J_{\mu^k}(i)| \leq \delta,
\]

and an approximate policy improvement error satisfying

\[
\max_{i=1,\ldots,n} \left| \sum_{j=1}^{n} p_{ij}(\mu^{k+1}(i))(g(i, \mu^{k+1}(i), j) + \alpha \hat{J}_{\mu^k}(j)) - \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u)(g(i, u, j) + \alpha \hat{J}_{\mu^k}(j)) \right| \leq \epsilon,
\]

where \( \delta \) and \( \epsilon \) are some nonnegative scalars. The following proposition, proved in the appendix (and also in the original source [BeT96], Section 6.2.2), provides a performance bound for discounted problems (a similar result is available for SSP problems; see [BeT96], Section 6.2.2).

**Proposition 5.1.4:** (Performance Bound for Approximate PI)

Consider the discounted problem, and let \( \{\mu^k\} \) be the sequence generated by the approximate PI algorithm defined by the approximate policy evaluation (5.12) and the approximate policy improvement (5.13). Then the policy error
The preceding performance bound is not particularly useful in practical terms. Significantly, however, it is in qualitative agreement with the empirical behavior of approximate PI. In the beginning, the method tends to make rapid and fairly monotonic progress, but eventually it gets into an oscillatory pattern. This happens after $J_{\mu_k}^k$ gets within an error zone of size less than $(\epsilon + 2\alpha \delta)/(1 - \alpha)^2$. After that $J_{\mu_k}$ oscillates randomly within that zone; see Fig. 5.1.4. In practice, the error bound of Prop. 5.1.4 tends to be pessimistic, so the zone of oscillation is usually much narrower than what is suggested by the bound. However, the bound itself can be proved to be tight, in worst case. This is shown with an example in the book [BeT96], Section 6.2.3. Note also that the bound of Prop. 5.1.4 holds in the case of infinite state and control spaces discounted problems, when there are infinitely many policies (see [Ber18a], Prop. 2.4.3).

**Performance Bound for the Case Where Policies Converge**

Generally, the policy sequence $\{\mu^k\}$ generated by approximate PI may
Sec. 5.1 Approximation in Value Space - Performance Bounds

Figure 5.1.5 Illustration of typical behavior of approximate PI when policies converge. The method tends to make monotonic progress, and $J_{\mu^k}$ converges within an error zone of size less than

\[ \frac{\epsilon + 2\alpha\delta}{1 - \alpha}. \]

oscillate between several policies, as noted earlier. However, under some circumstances the sequence will converge to some policy $\hat{\mu}$, in the sense that

\[ \mu^{k+1} = \mu^k = \hat{\mu} \quad \text{for some } k. \quad (5.14) \]

An important case where this happens is aggregation methods, which will be discussed in Chapter 6. In this case the behavior of the method is more regular, and we can show a more favorable bound than the one of Prop. 5.1.4, by a factor

\[ \frac{1}{1 - \alpha}; \]

see Fig. 5.1.5.

**Proposition 5.1.5: (Performance Bound for Approximate PI when Policies Converge)** Let $\hat{\mu}$ be a policy generated by the approximate PI algorithm under conditions (5.12), (5.13), and (5.14). Then we have

\[ \max_{i=1, \ldots, n} |J_{\hat{\mu}}(i) - J^*(i)| \leq \frac{\epsilon + 2\alpha\delta}{1 - \alpha}. \]

We finally note that similar performance bounds can be obtained for optimistic PI methods, where the policy evaluation is performed with
Infinite Horizon Reinforcement Learning Chap. 5

just a few approximate value iterations, and policy improvement is approximate (cf. Section 4.5.2). These bounds are similar to the ones of the nonoptimistic PI case given in this section, but their derivation is quite complicated; see [Ber12], Chapter 2, or [Ber18a], Section 2.5.2, and the end-of-chapter references.

5.2 FITTED VALUE ITERATION

In Chapter 4 we discussed the VI algorithm for SSP problems,

\[
J_{k+1}(i) = \min_{u \in U(i)} \left[ p_{\mu}(u)g(i, u, t) + \sum_{j=1}^{n} p_{ij}(u) \left( g(i, u, j) + J_k(j) \right) \right], \quad (5.15)
\]

and its discounted version

\[
J_{k+1}(i) = \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u) \left( g(i, u, j) + \alpha J_k(j) \right). \quad (5.16)
\]

It is one of the principal methods for calculating the optimal cost function \( J^* \).

Unfortunately, when the number of states is large, the iterations (5.15) and (5.16) may be prohibitively time consuming. This motivates an approximate version of VI, which is patterned after the least squares regression/fitted VI scheme of Section 3.3 for finite horizon problems. We start with some initial approximation to \( J^* \), call it \( \tilde{J}_0 \). Then we generate a sequence \( \{\tilde{J}_k\} \) where \( \tilde{J}_{k+1} \) is equal to the exact value iterate \( T\tilde{J}_k \) plus some error [we are using here the shorthand notation for the DP operator \( T \) given in Eqs. (5.1) and (5.3). Assuming that values \( (T\tilde{J}_k)(i) \) may be generated for sample states \( i \), we may obtain \( \tilde{J}_{k+1} \) by some form of least squares regression. We will now discuss how the error \( (\tilde{J}_k - J^*) \) is affected by this type of approximation process.

Error Bounds and Pathologies of Fitted Value Iteration

We will focus on fitted VI for discounted problems. The analysis for SSP problems is qualitatively similar. We first consider estimates of the cost function error

\[
\max_{i=1,\ldots,n} \left| \tilde{J}_k(i) - J^*(i) \right|, \quad (5.17)
\]

and the policy error

\[
\max_{i=1,\ldots,n} \left| \tilde{\mu}_k(i) - J^*(i) \right|, \quad (5.18)
\]
where the policy \( \hat{\mu}_k \) is obtained from the minimization

\[
\hat{\mu}_k(i) \in \arg \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u) \left( g(i, u, j) + \alpha \tilde{J}_k(j) \right).
\]

It turns out that such estimates are possible, but under assumptions whose validity may be hard to guarantee. In particular, it is natural to assume that the error in generating the value iterates \((T \tilde{J}_k)(i)\) is within some \( \delta > 0 \) for every state \( i \) and iteration \( k \), i.e., that

\[
\max_{i=1,...,n} \left| \tilde{J}_{k+1}(i) - \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u) \left( g(i, u, j) + \alpha \tilde{J}_k(j) \right) \right| \leq \delta. \quad (5.19)
\]

It is then possible to show that asymptotically, as \( k \to \infty \), the cost error (5.17) becomes less or equal to \( \delta/(1 - \alpha) \), while the policy error (5.18) becomes less or equal to \( 2\delta/(1 - \alpha)^2 \).

Such error bounds are given in Section 6.5.3 of the book [BeT96] (see also Prop. 2.5.3 of [Ber12]), but it is important to note that the condition (5.19) may not be satisfied by the natural least squares regression/fitted VI scheme of Section 3.3. This is illustrated by the following simple example from [TsV96] (see also [BeT96], Section 6.5.3), which shows that the errors from successive approximate value iterations can accumulate to the point where the condition (5.19) cannot be maintained, and the approximate value iterates \( \tilde{J}_k \) can grow unbounded.

**Example 5.2.1 (Error Amplification in Approximate Value Iteration)**

Consider a two-state discounted problem with states 1 and 2, and a single policy. The transitions are deterministic: from state 1 to state 2, and from state 2 to state 2. The transitions are also cost-free; see Fig. 5.2.1. Thus the Bellman equation is

\[
J(1) = \alpha J(2), \quad J(2) = \alpha J(2),
\]

and its unique solution is \( J^*(1) = J^*(2) = 0 \). Moreover, exact VI has the form

\[
J_{k+1}(1) = \alpha J_k(2), \quad J_{k+1}(2) = \alpha J_k(2).
\]

We consider a VI approach that approximates cost functions within the one-dimensional subspace of linear functions \( S = \{(r, 2r) \mid r \in \mathbb{R}\} \); this is a favorable choice since the optimal cost function \( J^* = (0, 0) \) belongs to \( S \). We use a weighted least squares regression scheme. In particular, given \( \tilde{J}_k = (r_k, 2r_k) \), we find \( \tilde{J}_{k+1} = (r_{k+1}, 2r_{k+1}) \) as follows; see Fig. 5.2.2:

(a) We compute the exact VI iterate from \( \tilde{J}_k \):

\[
T \tilde{J}_k = (\alpha \tilde{J}_k(2), \alpha \tilde{J}_k(2)) = (2\alpha r_k, 2\alpha r_k).
\]
Figure 5.2.1 Illustration of the discounted problem of Example 5.2.1. There are two states, 1 and 2, and a single policy. The transitions are deterministic: from state 1 to state 2, and from state 2 to state 2. These transitions are also cost-free.

(b) For some weights \( \xi_1, \xi_2 > 0 \), we obtain the scalar \( r_{k+1} \) as

\[
r_{k+1} \in \arg \min_r \left[ \xi_1 (r - (T \tilde{J}_k)(1))^2 + \xi_2 (2r - (T \tilde{J}_k)(2))^2 \right],
\]

or

\[
r_{k+1} \in \arg \min_r \left[ \xi_1 (r - 2\alpha r_k)^2 + \xi_2 (2r - 2\alpha r_k)^2 \right].
\]

To perform the preceding minimization, we write the corresponding optimality condition (set to zero the derivative with respect to \( r \)), and obtain after some calculation

\[
r_{k+1} = \alpha \zeta r_k \quad \text{where} \quad \zeta = \frac{2(\xi_1 + 2\xi_2)}{\xi_1 + 4\xi_2} > 1. \quad (5.20)
\]

Thus if \( \xi_1 \) and \( \xi_2 \) are chosen so that \( \alpha > 1/\zeta \), the sequence \( \{r_k\} \) diverges and so does \( \{\tilde{J}_k\} \). In particular, for the natural choice \( \xi_1 = \xi_2 = 1 \), we have \( \zeta = 6/5 \), so the approximate VI scheme diverges for \( \alpha \) in the range \((5/6, 1)\); see Fig. 5.2.2.

The difficulty here is that the approximate VI mapping that generates \( \tilde{J}_{k+1} \) by a weighted least squares-based approximation of \( T \tilde{J}_k \) is not a contraction (even though \( T \) itself is a contraction). At the same time there is no \( \delta \) such that the condition (5.19) is satisfied for all \( k \), because of error amplification in each approximate VI.

The preceding example indicates that the choice of the least squares weights is important in determining the success of least squares-based approximate VI schemes. Generally, in regression-based parametric architecture training schemes of the type discussed in Section 3.1.2, the weights are related to the way samples are collected: the weight \( \xi_i \) for state \( i \) is the proportion of the number of samples in the least squares summation that correspond to state \( i \). Thus \( \xi_1 = \xi_2 = 1 \) in the preceding example means that we use an equal number of samples for each of the two states 1 and 2.

Now let us consider an approximation architecture \( \tilde{J}(i, \cdot) \) and a sampling process for approximating the value iterates. In particular, let

\[
\tilde{J}_k(i) = \tilde{J}(i, r_k), \quad i = 1, \ldots, n,
\]
where \( r_k \) is the parameter vector corresponding to iteration \( k \). Then the parameter \( r_{k+1} \) used to represent the next value iterate as

\[
\tilde{J}_{k+1}(i) = \tilde{J}(i, r_{k+1}), \quad i = 1, \ldots, n,
\]

is obtained by the minimization

\[
r_{k+1} \in \arg \min_r \sum_{s=1}^q (\tilde{J}(i^s, r) - \beta^s)^2,
\]

where \((i^s, \beta^s), s = 1, \ldots, q\), is a training set with each \( \beta^s \) being the value iterate at the state \( i^s \):

\[
\beta^s = (T\tilde{J}_k)(i^s).
\]

The critical question now is how to select the sample states \( i^s, s = 1, \ldots, q \), to guarantee that the iterates \( r_k \) remain bounded, so that a condition of the form (5.19) is satisfied and the instability illustrated with Example 5.2.1 is avoided. It turns out that there is no known general method
to guarantee this in infinite horizon problems. However, some practical methods have been developed. One such method is to weigh each state according to its “long-term importance,” i.e., proportionally to the number of its occurrences over a long trajectory under a “good” heuristic policy.† To implement this, we may run the system with the heuristic policy starting from a number of representative states, wait for some time for the system to approach steady-state, and record the generated states $i^s, s = 1, \ldots, q$, to be used in the regression scheme (5.21). There is no theoretical guarantee for the stability of this scheme in the absence of additional conditions: it has been used with success in several reported case studies, although its rationale has only a tenuous basis in analysis. For a discussion of this issue, we refer to [Ber12], Section 6.3, and other end-of-chapter references.

We finally note that in the absence of special modifications, optimistic PI with approximations is also subject to the error amplification phenomenon illustrated in Example 5.2.1. Indeed approximate VI, as described in Section 4.4, can be viewed as a special case of an optimistic PI method, where each policy evaluation is done with a single VI, and then approximated by least squares/regression.

5.3 SIMULATION-BASED POLICY ITERATION WITH PARAMETRIC APPROXIMATION

In this section we will discuss PI methods where the policy evaluation step is carried out with the use of a parametric approximation method and Monte-Carlo simulation. We will focus on the discounted problem, but similar methods can be used for the SSP problem.

5.3.1 Self-Learning and Actor-Critic Systems

The name “self-learning” in RL usually refers to some form of PI method that involves the use of simulation for approximate policy evaluation, and/or approximate Q-factor evaluation. A parametric architecture is used for this, and the algorithm that performs the policy evaluation is usually called a critic. If a neural network is used as the parametric architecture, it is called a critic network. The PI algorithm generates a sequence of stationary policies $\{\mu^k\}$ and a corresponding sequence of approximate cost function evaluations $\{\tilde{J}_k\}$ using a simulator of the system.

† In the preceding Example 5.2.1, weighing the two states according to their “long-term importance” would choose $\xi_2$ to be much larger than $\xi_1$, since state 2 is “much more important,” in the sense that it occurs almost exclusively in system trajectories. Indeed, from Eq. (5.20) it can be seen that when the ratio $\xi_1/\xi_2$ is close enough to 0, the scalar $\zeta$ is close enough to 1, making the scalar $\alpha\zeta$ strictly less than 1, and guaranteeing convergence of $\tilde{J}_k$ to $J^*$. 
As in all PI methods, the policy evaluation $\tilde{J}_{\mu_k}$ is used for policy improvement, to generate the next policy $\mu^{k+1}$. The algorithm that performs the policy improvement is usually called an *actor*, and if a neural network is involved, it is called an *actor network*.

The two operations needed at each policy iteration are as follows:

(a) *Evaluate the current policy $\mu^k$ (critic)*: Here algorithm, system, and simulator are merged in one, and the system “observes itself” by generating simulation cost samples under the policy $\mu^k$. It then combines these samples to “learn” a policy evaluation $\tilde{J}_{\mu_k}$. Usually this is done through some kind of incremental method that involves a least squares minimization/regression using cost samples, and either a linear architecture or a neural network.

(b) *Improve the current policy $\mu^k$ (actor)*: Given the approximate policy evaluation $\tilde{J}_{\mu_k}$, the system can generate or “learn” the new policy $\mu^{k+1}$ through the minimization

$$
\mu^{k+1}(i) \in \arg \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u) \left( g(i, u, j) + \alpha \tilde{J}_{\mu_k}(j) \right), \quad i = 1, \ldots, n.
$$

Alternatively the system can compute the minimizing control $u^*$ at a set of sample states $i^s$, $s = 1, \ldots, q$, through

$$
u^* \in \arg \min_{u \in U(i^s)} \sum_{j=1}^{n} p_{i^s j}(u) \left( g(i^s, u, j) + \alpha \tilde{J}_{\mu_k}(j) \right).
$$

These are the sample values of the improved policy $\mu^{k+1}$ at the sample states $i^s$. They are generalized to “learn” a complete policy $\mu^{k+1}$ by using some approximation in policy space scheme (cf. Section 2.1.3).

We can thus describe simulation-based PI as a process where the system learns better and better policies by observing its behavior. This is true up to the point where either policy oscillations occur (cf. Fig. 5.1.4) or the algorithm terminates (cf. Fig. 5.1.5), at which time learning essentially stops.

It is worth noting that the system learns by itself, but it does not learn itself, in the sense that it does not construct a mathematical model for itself. It only learns to behave better, i.e., construct improved policies, through experience gained by simulating state and control trajectories generated with these policies. We may adopt instead an alternative two-phase approach: first use system identification and simulation to construct a mathematical model of the system, and then use a model-based PI method. However, we will not discuss system identification and the construction of models in this book.
5.3.2 A Model-Based Variant

We will first provide an example of a model-based PI method that is conceptually simple, and then discuss its model-free version. In particular, we assume that the transition probabilities $p_{ij}(u)$ are available, and that the cost function $J_\mu$ of any given policy $\mu$ is approximated using a parametric architecture $\tilde{J}_\mu(i, r)$.

We recall that given any policy $\mu$, the exact PI algorithm for costs generates the new policy $\tilde{\mu}$ with a policy evaluation/policy improvement process. We approximate this process as follows; see Fig. 5.3.1.

(a) Approximate policy evaluation: To evaluate $\mu$, we determine the value of the parameter vector $r$ by generating a large number of training pairs $(i^s, \beta^s)$, $s = 1, \ldots, q$, and by using least squares training:

$$
\tau \in \arg \min_r \sum_{s=1}^{q} (\tilde{J}_\mu(i^s, r) - \beta^s)^2.
$$

(5.22)

For a given state $i^s$, the scalar $\beta^s$ is a sample cost corresponding to $i^s$ and $\mu$.

In particular $\beta^s$ is generated by starting at $i^s$, simulating a trajectory of states and controls using $\mu$ and the known transition probabilities for some number $N$ of stages, accumulating the corresponding discounted costs, and adding a terminal cost approximation

$$
\alpha^N \hat{J}(i_N),
$$

Figure 5.3.1 Block diagram of model-based approximate PI for cost functions.
where $i_N$ is the terminal state of the $N$-stage trajectory and $\hat{J}$ is some initial guess of $J_\mu$. The guess $\hat{J}$ may be obtained with additional training or some other means, such as using the result of the policy evaluation of the preceding policy $\mu^{k-1}$; this is similar to the cost function approximation implicitly used in optimistic policy iteration, cf. Section 4.5.2. It is also possible to simplify the method by using $\hat{J}(i_N) = 0$, or obtaining $\hat{J}$ via a problem approximation process.†

The approximate policy evaluation problem of Eq. (5.22) can be solved with the incremental methods discussed in Section 3.1.3. In particular the incremental gradient method is given by

$$r^{k+1} = r^k - \gamma^k \nabla \hat{J}(i^{*k}, r^k) \left( \hat{J}(i^{*k}, r^k) - \beta^{*k} \right),$$

where $(i^{*k}, \beta^{*k})$ is the state-cost sample pair that is used at the $k$th iteration, and $r^0$ is an initial parameter guess. Here the approximation architecture $\hat{J}(i, r)$ may be linear or may be nonlinear and differentiable. In the case of a linear architecture it is also possible to solve the problem (5.22) in closed form, i.e., by using an exact linear least squares formula.

(b) **Approximate policy improvement:** Having solved the approximate policy evaluation problem (5.22), the new “improved” policy $\hat{\mu}$ is obtained by the approximate policy improvement operation

$$\hat{\mu}(i) \in \arg \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u) \left( g(i, u, j) + \alpha \hat{J}(j, r) \right), \quad i = 1, \ldots, n,$$

(5.23)

where $r$ is the parameter vector obtained from the policy evaluation operation (5.22).

**Trajectory Reuse and Bias-Variance Tradeoff**

As we have noted, to each training pair $(i^s, \beta^s)$ there corresponds an $N$-stage trajectory over which the sample cost $\beta^s$ is accumulated, but the length of the trajectory may depend on $s$. This allows sampling effort economies based on trajectory reuse. In particular, suppose that starting at some state $i_0$ we generate a long trajectory $(i_0, i_1, \ldots, i_N)$ using the policy $\mu$. Then we can obtain the state-cost sample that corresponds to $i_0$, as discussed above, but we can also obtain additional cost samples for

† The choice of the terminal cost function approximation $J$ may pose difficulties in SSP problems, particularly when most of the cost is incurred upon termination. Dealing with this issue may be problem-dependent. It will not be discussed in this book.
the subsequent states $i_1, i_2, \ldots$, by using the tail portions of the trajectory $(i_0, i_1, \ldots, i_N)$ that start at these states.

Clearly, it is necessary to truncate the sample trajectories to some number of stages $N$, since we cannot simulate an infinite length trajectory in practice. If $N$ is large, then because of the discount factor, the error for neglecting the stage costs beyond stage $N$ will be small. However, there are other important concerns when choosing the trajectory lengths $N$.

In particular, a short length reduces the sampling effort, but is also a source of inaccuracy. The reason is that the cost of the tail portion of the trajectory (from stage $N$ to infinity) is approximated by $\alpha^N \hat{J}(i_N)$, where $i_N$ is the terminal state of the $N$-stage trajectory and $\hat{J}$ is the initial guess of $J_\mu$. This terminal cost compensates for the costs of the neglected stages in the spirit of optimistic PI, but adds an error to the cost samples $\beta^s$, which becomes larger as the trajectory length $N$ becomes smaller.

We note two additional benefits of using many training trajectories, each with a relatively short trajectory length:

1. The cost samples $\beta^s$ are less noisy, as they correspond to summation of fewer random stage costs. This leads to the so-called bias-variance tradeoff: short trajectories lead to larger bias but smaller variance of the cost samples.

2. With more starting states $i_0$, there is better opportunity for exploration of the state space. By this we mean adequate representation of all possible initial trajectory states in the sample set. This is a major issue in approximate PI, as we will discuss in Section 5.3.4.

Let us also note that the bias-variance tradeoff underlies the motivation for a number of alternative policy evaluation methods such as TD($\lambda$), LSTD($\lambda$), and LSPE($\lambda$), which we will summarize in Section 5.5; see Section 6.3 of the book [Ber12] and other approximate DP/RL books referenced earlier. The papers [Ber11b], [YuB12], and the book [Ber12], Section 6.4, discuss a broad range of short trajectory sampling methods.

### 5.3.3 A Model-Free Variant

We will now provide an example model-free PI method. Let us restate the PI method in terms of Q-factors, and in a form that involves approximations and simulation-based implementations. We recall that given any policy $\mu$, the exact PI algorithm for Q-factors (cf. Section 4.5.3) generates the new policy $\tilde{\mu}$ with a policy evaluation-policy improvement process.

To this end, we introduce a parametric architecture $\hat{Q}_\mu(i, u, r)$ for the Q-factors of $\mu$. This architecture may involve a neural network, or a feature-based linear architecture. The feature vector may depend on just the state, or on both the state and the control. In the former case, the architecture has the form

$$\hat{Q}_\mu(i, u, r) = r(u)' \phi(i), \quad (5.24)$$
where \( r(u) \) is a separate weight vector for each control \( u \). In the latter case, the architecture has the form
\[
\hat{Q}_\mu(i, u, r) = r^* \phi(i, u),
\]
where \( r \) is a weight vector that is independent of \( u \). The architecture (5.24) is suitable for problems with a relatively small number of control options at each stage.

We approximate this process as follows; see Fig. 5.3.2.

(a) Approximate policy evaluation: Here we determine the value of the parameter vector \( r \) by generating (using a simulator of the system) a large number of training triplets \((i^s, u^s, \beta^s)\), \( s = 1, \ldots, q \), and by using a least squares fit:
\[
\tau \in \arg\min_r \sum_{s=1}^{q} (\hat{Q}_\mu(i^s, u^s, r) - \beta^s)^2. \tag{5.26}
\]

In particular, for a given pair \((i^s, u^s)\), the scalar \( \beta^s \) is a sample Q-factor corresponding to \((i, u)\). It is generated by starting at \( i^s \), using \( u^s \) at the first stage, and simulating a trajectory of states and controls using \( \mu \) for a total of \( N \) stages, and accumulating the corresponding discounted costs. Thus, \( \beta^s \) is a sample of \( Q^N_\mu(i^s, u^s) \), the \( N \)-stage Q-factor of \( \mu \), given by
\[
Q^N_\mu(i, u) = \sum_{j=1}^{n} p_{ij}(u)(g(i, u, j) + \alpha J^{N-1}_\mu(j)),
\]
where $J^{N-1}_\mu(j)$ is the $(N - 1)$-stages cost of $\mu$ starting at $j$. The number of stages $N$ in the sample trajectories may be different for different samples, and can be either large, or fairly small. Moreover, a terminal cost approximation $\alpha N \hat{J}(i_N)$ may be added to $\beta^*$ as in the model-based case of Section 5.3.2. Again an incremental method may be used to solve the training problem (5.26).

(b) **Approximate policy improvement:** Here we compute the new policy $\tilde{\mu}$ according to

$$\tilde{\mu}(i) \in \arg \min_{u \in U(i)} \hat{Q}(i, u, \pi), \quad i = 1, \ldots, n, \quad (5.27)$$

where $\pi$ is the parameter vector obtained from the policy evaluation operation (5.26).

Unfortunately, trajectory reuse is more problematic in $Q$-factor evaluation than in cost evaluation, because each trajectory generates state-control pairs of the special form $(i, \mu(i))$ at every stage after the first, so pairs $(i, u)$ with $u \neq \mu(i)$ are not adequately explored; cf. the discussion in Section 5.3.2. For this reason, it is necessary to make an effort to include in the samples a rich enough set of trajectories that start at pairs $(i, u)$ with $u \neq \mu(i)$. We discuss this issue in Section 5.3.4.

An important alternative to the preceding procedure is a two-stage process for policy evaluation: first compute in model-free fashion a cost function approximation $\hat{J}(j, \pi)$, using the regression (5.22), and then use a second sampling process and regression to approximate further the (already approximate) $Q$-factor

$$\sum_{j=1}^{n} p_{ij}(u)(g(i, u, j) + \alpha \hat{J}(j, \pi)),$$

with some $\hat{Q}(i, u, \pi)$ possibly obtained with a policy approximation architecture (see the discussion of Section 2.1.3 on model-free approximation in policy space). Finally, once $\hat{Q}(i, u, \pi)$ is obtained with this approximation in policy space, the “improved” policy $\tilde{\mu}$ is obtained from the minimization (5.27). The overall scheme can be viewed as model-free approximate PI that is based on approximation in both value and policy space. In view of the two-fold approximation needed to obtain $\hat{Q}(i, u, \pi)$, this scheme is more complex, but allows trajectory reuse and thus deals better with the exploration issue.

### 5.3.4 Implementation Issues of Parametric Policy Iteration

Approximate PI in its various forms has been the subject of extensive research, both theoretical and applied. Let us provide a few comments, focusing on the preceding parametric PI schemes.
Architectural Issues and Cost Shaping

The choice of architectures for costs $\tilde{J}_\mu(i,r)$ and Q-factors $\tilde{Q}_\mu(i,u,r)$ is critical for the success of parametric approximation schemes. These architectures may involve the use of features, and they could be linear, or they could be nonlinear such as a neural network. A major advantage of a linear feature-based architecture is that the policy evaluations (5.22) and (5.26) involve linear least squares problems, which admit a closed-form solution. Moreover, when linear architectures are used, there is a broader variety of approximate policy evaluation methods with solid theoretical performance guarantees, such as TD($\lambda$), LSTD($\lambda$), and LSPE($\lambda$), which will be summarized in Section 5.5, and are described in detail in several textbook sources.

Another interesting possibility for architecture choice has to do with cost shaping, which we discussed in Section 4.2. This possibility involves a modified cost per stage $\hat{g}(i,u,j) = g(i,u,j) + V(j) - V(i)$, $i = 1, \ldots, n$, for SSP problems, where $V$ can be any approximation to $J^*$. The corresponding formula for discounted problems is $\hat{g}(i,u,j) = g(i,u,j) + \alpha V(j) - V(i)$, $i = 1, \ldots, n$.

As noted in Section 4.2, cost shaping may change significantly the suboptimal policies produced by approximate DP methods and approximate PI in particular. Generally, $V$ should be chosen close (at least in terms of “shape”) to $J^*$ or to the current policy cost function $J^{\mu,k}$, so that the difference $J^* - V$ or $J^{\mu,k} - V$, respectively, can be approximated by an architecture that matches well the characteristics of the problem. It is possible to approximate either $V$ or $J$ with a parametric architecture or with a different approximation method, depending on the problem at hand. Moreover, in the context of approximate PI, the choice of $V$ may change from one policy evaluation to the next.

The literature referenced at the end of the chapter provide some applications of cost shaping. An interesting possibility is to use complementary approximations for $V$ and for $J^*$ or $J^{\mu,k}$. For example $V$ may be approximated by a neural network-based approach that aims to discover the general form of $J^*$ or $J^{\mu,k}$, and then a different method may be applied to provide a local correction to $V$ in order to refine the approximation. In Chapter 6 will also illustrate this idea within the context of aggregation.

Exploration Issues

Generating an appropriate training set at the policy evaluation step of approximate PI poses considerable challenges, and the literature contains
several related proposals. Let us discuss the generic issue of *inadequate exploration*, which we noted in Section 5.3.2 in connection with model-based approximate PI.

We recall that in the PI variant of that section, to evaluate a policy \( \mu \), we determine the value of the parameter vector \( r \) by generating a large number of training pairs \((i^s, \beta^s), s = 1, \ldots, q\), where \( \beta^s \) is a sample cost corresponding to \( i^s \) and \( \mu \). We then use least squares training:

\[
\mathbf{r} \in \arg \min_r \sum_{s=1}^{q} (\hat{J}(i^s, r) - \beta^s)^2,
\]

[cf. Eq. (5.22)].

Each sample cost \( \beta^s \) is generated by starting at \( i^s \), simulating a trajectory of states and controls using \( \mu \) and the known transition probabilities for some number \( N_s \) of stages, accumulating the corresponding discounted costs, and adding a terminal cost approximation

\[
\alpha^{N_s} \hat{J}(i_{N_s}),
\]

where \( i_{N_s} \) is the terminal state of the \( N_s \)-stage trajectory and \( \hat{J} \) is some initial guess of \( J_\mu \). In this context, we also discussed trajectory reuse, which aims at reducing the sampling effort, by using the tail portions of any generated trajectory.

Thus with trajectory reuse, we will be generating many cost or Q-factor samples that start from states frequently visited by \( \mu \), and this may bias the simulation by underrepresenting states that are unlikely to occur under \( \mu \). As a result, the cost estimates of these underrepresented states may be highly inaccurate, causing potentially serious errors in the calculation of the improved policy \( \mathbf{r'} \) via the policy improvement operation.

One possibility to improve the exploration of the state space is to use a large number of initial states. It may then be necessary to use relatively short trajectories to keep the cost of the simulation low. To compensate for the short length of the trajectories it will then be important to introduce a terminal cost function approximation in the policy evaluation step in order to make the cost sample \( \beta^s \) more accurate, as noted earlier. Moreover, when selecting the initial states of these trajectories, we should make sure that they are a representative sample of the portion of the state space most visited not just under the current policy and also under other policies.

A simple plausible scheme when evaluating policy \( \mu^k \), is to use *randomization over a memory buffer*, which stores candidate initial states. In particular, we may use a set of initial states that consists of the union of multiple subsets \( I, I_0, \ldots, I_{k-1} \), where \( I \) is a starting set of initial states, and \( I_0, \ldots, I_{k-1} \) are generated while evaluating the previous policies \( \mu^0, \ldots, \mu^{k-1} \). Each subset \( I_m \) should consist of states generated while evaluating policy \( \mu^m \). We evaluate policy \( \mu^k \), by using short trajectories whose starting
states are chosen randomly from the subsets $I, I_0, \ldots, I_{k-1}$ with probabilities that reflect a bias toward more recent policies (i.e., the probability of selecting an initial state from set $I_m$ should increase with $m$). At the same time, while evaluating policy $\mu^k$, we should generate the subset of states $I_k$ to use when training the next policy $\mu^{k+1}$. The initial subset $I$ is special and should be carefully chosen so that it includes not only a subset of representative states, but also successor states under a variety of control choices.

Thus in the preceding scheme, all state transitions and associated transition costs used to generate the costs $\beta^s$ of the training set are generated by using the current policy $\mu^k$. However, the set of initial states of the trajectories that are used to produce the cost samples is quite diverse: it is obtained by randomization over the union of the starting set $I$ and the set $I_0 \cup \cdots \cup I_{k-1}$, which corresponds to states visited by all the preceding policies.

A potential weakness of the scheme just described is the need for a good terminal cost function approximation. Such a cost may not be easily available, particularly for SSP problems involving substantial “late costs;” for example when most of the cost is incurred upon reaching the destination. Such problems generally require “deep exploration,” i.e., long trajectories that reach or get close to the destination.

Exploration schemes like the one just described may also be used in the context of model-free variants of approximate PI for Q-factors; cf. Section 5.3.3. In fact, as we discussed in that section the need for exploration in the space of state-control pairs is more acute within that context. Again one may generate short trajectories that start with state-control pairs chosen randomly within a set of pairs that are representative of several policies to enhance exploration.

There have also been other related approaches to improve exploration, particularly in connection with the temporal difference methods to be discussed in Section 5.5. In some of these approaches, trajectories are generated through a mix of two policies: the policy being evaluated, sometimes called the target policy, to distinguish from the other policy, used with some probability at each stage, which is called behavior policy and is introduced to enhance exploration; see the end-of-chapter references. Also, methods that use a behavior policy are called off-policy methods, while methods that do not are called on-policy methods. A commonly suggested scheme is to use as off-policy one that is “$\epsilon$-greedy;” i.e., that attains the minimum in the policy improvement phase within some threshold $\epsilon > 0$, which is experimentally chosen. A variation of this scheme is to select off-policy controls by occasional randomization within the set of “$\epsilon$-greedy” controls.

It is important to note, however, that using a behavior policy biases the cost samples towards that policy. Special modifications are needed to eliminate this source of bias (see [Ber12], Section 6.4.2). More importantly, even with a behavior policy, it may still be difficult to ensure that the
mixed on-and-off policy will induce sufficient exploration.

Generally, the area of efficient sampling, and the attendant issue of balancing exploration and the choice of promising controls (the so-called exploration-exploitation tradeoff) is a subject continuing research. For some recent work, see the paper by Russo and Van Roy [RuV16], the monograph [RVK18]. The paper by Osband, VanRoy, Russo, and Wen [OVR19] develops special schemes to deal with issues of deep exploration.

5.3.5 Convergence Issues of Approximate Policy Iteration - Oscillations

We will now consider the sequence of policies generated by approximate PI. Contrary to exact PI, which is guaranteed to yield an optimal policy, in general approximate PI produces a sequence of policies, which are only guaranteed to lie asymptotically within a certain error bound from the optimal; cf. Prop. 5.1.4. Moreover, the generated policies may oscillate. By this we mean that after a few iterations, policies tend to repeat in cycles.

This oscillation phenomenon, first described by the author in a 1996 conference [Ber96], occurs systematically in the absence of special conditions, for both optimistic and nonoptimistic PI methods. It can be observed even in very simple examples, as we will demonstrate shortly.

To describe a generic mechanism that may cause policy oscillations in approximate PI, we focus on the discounted problem, and we introduce the so called greedy partition. For a given approximation architecture \( \tilde{J}(\cdot, r) \), this is a partition of the space \( \mathbb{R}^s \) of parameter vectors \( r \) into subsets \( R_\mu \), each subset corresponding to a stationary policy \( \mu \), and defined by \( \dagger \)

\[
R_\mu = \{ r \mid T_\mu(\tilde{J}(\cdot, r)) = T(\tilde{J}(\cdot, r)) \}
\]

or equivalently

\[
R_\mu = \left\{ r \mid \mu(i) = \arg \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u) \left( g(i, u, j) + \alpha \tilde{J}(j, r) \right), \ i = 1, \ldots, n \right\}
\]

Thus, \( R_\mu \) is the set of parameter vectors \( r \) for which \( \mu \) is “greedy” with respect to \( \tilde{J}(\cdot, r) \). Note that the greedy partition depends only on the approximation architecture \( \tilde{J}(\cdot, r) \) (which can be arbitrary, e.g., nonlinear), and does not depend on the method used for policy evaluation.

\( \dagger \) Since we are now dealing with multiple policies, our notation reflects the dependence of the stage costs and transition probabilities on the control. Also \( T_\mu \) is the Bellman equation mapping corresponding to a policy \( \mu \), while \( T \) is defined in terms of minimization of \( T_\mu \) over \( \mu \).
Sec. 5.3 Simulation-Based Policy Iteration

We will first consider the nonoptimistic version of approximate PI; a similar phenomenon occurs for optimistic PI, as we discuss later. For simplicity, let us assume that we use a policy evaluation method that for each given \( \mu \) produces a unique parameter vector denoted \( r_{\mu} \). The method starts with a policy \( \mu^0 \) and generates \( r_{\mu^0} \) by using the given policy evaluation method. It then finds a policy \( \mu^1 \) such that

\[
 r_{\mu^0} \in R_{\mu^1}.
\]

It then repeats the process with \( \mu^1 \) replacing \( \mu^0 \). If some policy \( \mu^k \) satisfying

\[
 r_{\mu^k} \in R_{\mu^k},
\]

is encountered, the method keeps generating that policy. This is the necessary and sufficient condition for policy convergence in the nonoptimistic PI method.

In the case of a lookup table representation where the parameter vector \( r_{\mu} \) is equal to the cost-to-go vector \( J_{\mu} \), the condition \( r_{\mu^k} \in R_{\mu^k} \) is equivalent to \( r_{\mu^k} = T r_{\mu^k} \), and is satisfied if and only if \( \mu^k \) is optimal. When there is cost function approximation, however, this condition need not be satisfied for any policy. Since there is a finite number of possible vectors \( r_{\mu} \), one generated from another in a deterministic way, the algorithm ends up repeating some cycle of policies \( \mu^k, \mu^{k+1}, \ldots, \mu^{k+m} \) with

\[
 r_{\mu^k} \in R_{\mu^{k+1}}, r_{\mu^{k+1}} \in R_{\mu^{k+2}}, \ldots, r_{\mu^{k+m-1}} \in R_{\mu^{k+m}}, r_{\mu^{k+m}} \in R_{\mu^k};
\]

(see Fig. 5.3.3). Furthermore, there may be several different cycles, and the method may end up converging to any one of them. The actual cycle obtained depends on the initial policy \( \mu^0 \). This is similar to gradient
Figure 5.3.4 Illustration of a trajectory of optimistic policy iteration with cost function approximation. The algorithm settles into an oscillation between policies $\mu_1, \mu_2, \mu_3$ with

$$r_{\mu_1} \in R_{\mu_2}, \quad r_{\mu_2} \in R_{\mu_3}, \quad r_{\mu_3} \in R_{\mu_1}.$$  

The parameter vectors converge to the common boundary of $R_{\mu_1}, R_{\mu_2}, R_{\mu_3}$.

Oscillations can in principle be particularly damaging, because there is no guarantee that the oscillating policies are “good” policies, and there is often no way to verify how well they perform relative to the optimal. Section 6.4.2 of the book [BeT96] provides an argument suggesting that oscillations may not degrade significantly the approximate PI performance for many types of problems. Moreover, we note that oscillations can be avoided and approximate PI can be shown to converge under special conditions, which arise in particular when an aggregation approach is used; see Chapter 6 and the approximate PI survey [Ber11a]. Also, when policies converge, there is a more favorable error bound, cf. Prop. 5.1.5.

Oscillations and Chattering in Optimistic PI Methods

We now consider policy oscillations when an optimistic variant of approximate policy evaluation methods is used. Then the trajectory of the method is less predictable and depends on the fine details of the iterative policy evaluation method, such as the frequency of the policy updates and the stepsize selection rule. Generally, given the current policy $\mu$, optimistic PI will move towards the corresponding “target” parameter $r_\mu$, for as long as $\mu$ continues to be greedy with respect to the current cost-to-go approximation $\tilde{J}(\cdot, r)$, that is, for as long as the current parameter vector $r$ belongs to the set $R_\mu$. Once, however, the parameter $r$ crosses into another set, say $R_\pi$, the policy $\pi$ becomes greedy, and $r$ changes course and starts moving.
towards the new “target” \( r^* \). Thus, the “targets” \( r_\mu \) of the method, and the corresponding policies \( \mu \) and sets \( R_\mu \) may keep changing, similar to nonoptimistic policy iteration. Simultaneously, the parameter vector \( r \) will move near the boundaries that separate the regions \( R_\mu \) that the method visits, following reduced versions of the cycles that nonoptimistic PI may follow (see Fig. 5.3.3). Furthermore, as Fig. 5.3.4 shows, if diminishing parameter changes are made between policy updates (such as for example when a diminishing stepsize is used by the policy evaluation method) and the method eventually cycles between several policies, the parameter vectors will tend to converge to the common boundary of the regions \( R_\mu \) corresponding to these policies. This is the so-called chattering phenomenon for optimistic PI, whereby there is simultaneously oscillation in policy space and convergence in parameter space.

An additional insight is that the choice of the approximate policy evaluation method and exploration scheme does not seem crucial for the quality of the final policy obtained. Using a different method changes the targets \( r_\mu \) somewhat, but leaves the greedy partition unchanged. As a result, different methods “fish in the same waters” and tend to yield similar ultimate cycles of policies.

When chattering occurs, the limit of optimistic PI tends to be on a common boundary of several subsets of the greedy partition and may not meaningfully represent a cost approximation for any of the corresponding policies, as illustrated by the preceding example. Thus, the limit to which the method converges cannot always be used to construct an approximation of the cost-to-go of any policy or the optimal cost-to-go. As a result, at the end of optimistic PI and in contrast with the nonoptimistic version, one must go back and perform a screening process; that is, evaluate by simulation the many policies generated by the method starting from the initial conditions of interest and select the most promising one. This is a disadvantage of optimistic PI that may nullify whatever practical rate of convergence advantages it may have over its nonoptimistic counterpart.

We note, however, that computational experience indicates that for many problems, the cost functions of the different policies involved in chattering may not be “too different.” Indeed, suppose that we have convergence to a parameter vector \( \mathbf{\tau} \) and that there is a steady-state policy oscillation involving a collection of policies \( \mathcal{M} \). Then, all the policies in \( \mathcal{M} \) are greedy with respect to \( \mathcal{J}(\cdot, \mathbf{\tau}) \), which implies that there is a subset of states \( i \) such that there are at least two different controls \( \mu_1(i) \) and \( \mu_2(i) \) satisfying

\[
\min_{u \in U(i)} \sum_j p_{ij}(u) \left( g(i, u, j) + \alpha \mathcal{J}(j, \mathbf{\tau}) \right) = \sum_j p_{ij}(\mu_1(i)) \left( g(i, \mu_1(i), j) + \alpha \mathcal{J}(j, \mathbf{\tau}) \right)
\]
Each equation of this type can be viewed as a constraining relation on the parameter vector $\mathbf{r}$. Thus, excluding singular situations, there will be at most $s$ relations of the preceding form holding, where $s$ is the dimension of $\mathbf{r}$. This implies that there will be at most $s$ “ambiguous” states where more than one control is greedy with respect to $\tilde{J}(\cdot, \mathbf{r})$.

Now assume that we have a problem where the total number of states is much larger than $s$, and in addition there are no “critical” states; that is, the cost consequences of changing a policy in just a small number of states (say, of the order of $s$) is relatively small. It then follows that all policies in the set $\mathcal{M}$ involved in chattering have roughly the same cost. Furthermore, for the methods of this section, one may argue that the cost approximation $\tilde{J}(\cdot, \mathbf{r})$ is close to the cost approximation $\tilde{J}(\cdot, \mu)$ that would be generated for any of the policies $\mu \in \mathcal{M}$. Note, however, that the assumption of “no critical states,” aside from the fact that it may not be easily quantifiable, it will not be true for many problems.

5.4 Q-LEARNING

In this section we will discuss various Q-learning algorithms for discounted problems, which can be implemented in model-free fashion. The original method of this type is related to the VI algorithm for Q-factors, described in Sections 4.2 and 4.3. Instead of approximating the cost functions of successive policies as in the PI method, it updates the Q-factors associated with an optimal policy, thereby avoiding the multiple policy evaluation steps of PI. We will consider Q-learning as well as a variety of related methods with the shared characteristic that they involve exact or approximate Q-factors.

We first discuss the original form of Q-learning for discounted problems; the books [BeT96] and [Ber12] contain discussions of Q-learning for SSP problems. Then we discuss PI algorithms for Q-factors, including optimistic asynchronous versions, which lead to algorithms with reduced overhead per iteration. Finally we focus on Q-learning algorithms with Q-factor approximation.

In the discounted problem, the optimal Q-factors are defined for all pairs $(i,u)$ with $u \in U(i)$, by

$$Q^*(i,u) = \sum_{j=1}^{n} p_{ij}(u) \left( g(i,u,j) + \alpha J^*(j) \right).$$

As discussed in Section 4.3, these Q-factors satisfy for all $(i,u)$,

$$Q^*(i,u) = \sum_{j=1}^{n} p_{ij}(u) \left( g(i,u,j) + \alpha \min_{v \in U(j)} Q^*(j,v) \right),$$
and are the unique solution of this set of equations. Moreover the optimal Q-factors can be obtained by the VI algorithm \( Q_{k+1} = FQ_k \), where \( F \) is the operator defined by

\[
(FQ)(i, u) = \sum_{j=1}^{n} p_{ij}(u) \left( g(i, u, j) + \alpha \min_{v \in \mathcal{U}(j)} Q(j, v) \right), \quad \text{for all } (i, u).
\]

(5.28)

It is straightforward to show that \( F \) is a contraction with modulus \( \alpha \), similar to the DP operator \( T \). Thus the algorithm \( Q_{k+1} = FQ_k \) converges to \( Q^* \) from every starting point \( Q_0 \).

The original and most widely known Q-learning algorithm ([Wat89]) is a stochastic version of VI, whereby the expected value in Eq. (5.28) is suitably approximated by sampling and simulation. In particular, an infinitely long sequence of state-control pairs \( \{(i_k, u_k)\} \) is generated according to some probabilistic mechanism. For each pair \( (i_k, u_k) \), a state \( j_k \) is generated according to the probabilities \( p_{ikj}(u_k) \). Then the Q-factor of \( (i_k, u_k) \) is updated using a stepsize \( \gamma^k \in (0,1] \) while all other Q-factors are left unchanged:

\[
Q_{k+1}(i, u) = (1 - \gamma^k)Q_k(i, u) + \gamma^k(F_kQ_k)(i, u), \quad \text{for all } (i, u), \quad (5.29)
\]

where

\[
(F_kQ_k)(i, u) = \begin{cases} 
 g(i_k, u_k, j_k) + \alpha \min_{v \in \mathcal{U}(j_k)} Q_k(j_k, v) & \text{if } (i, u) = (i_k, u_k), \\
 Q_k(i, u) & \text{if } (i, u) \neq (i_k, u_k). 
\end{cases}
\]

(5.30)

Note that \( (F_kQ_k)(i_k, u_k) \) is a single sample approximation of the expected value defining \( (FQ_k)(i_k, u_k) \) in Eq. (5.28).

To guarantee the convergence of the algorithm (5.29)-(5.30) to the optimal Q-factors, some conditions must be satisfied. Chief among these are that all state-control pairs \( (i, u) \) must be generated infinitely often within the infinitely long sequence \( \{(i_k, u_k)\} \), and that the successor states \( j \) must be independently sampled at each occurrence of a given state-control pair. Furthermore, the stepsize \( \gamma^k \) should satisfy the conditions

\[
\gamma^k > 0, \quad \text{for all } k, \quad \sum_{k=0}^{\infty} \gamma^k = \infty, \quad \sum_{k=0}^{\infty} (\gamma^k)^2 < \infty,
\]

which are typical of stochastic approximation methods (see e.g., the books [BeT96], [Ber12], Section 6.1.4), as for example when \( \gamma^k = c_1/(k + c_2) \), where \( c_1 \) and \( c_2 \) are some positive constants. In addition some other technical conditions should hold. A mathematically rigorous convergence proof was given in the paper [Tsi94], which embeds Q-learning within a broad class of asynchronous stochastic approximation algorithms. This proof
Infinite Horizon Reinforcement Learning Chap. 5

(also reproduced in [BeT96]) combines the theory of stochastic approximation algorithms with the convergence theory of asynchronous DP and asynchronous iterative methods; cf. the papers [Ber82], [Ber83], and the book [BeT89].

In practice, Q-learning has some drawbacks, the most important of which is that the number of Q-factors/state-control pairs \((i, u)\) may be excessive. To alleviate this difficulty, we may introduce a Q-factor approximation architecture. One of these possibilities will be discussed next.

### 5.4.1 Optimistic Policy Iteration with Parametric Q-Factor Approximation - SARSA and DQN

We have discussed so far Q-learning algorithms with an exact representation of Q-factors. We will now consider Q-learning with Q-factor approximation. As we noted earlier, we may view Q-factors as optimal costs of a certain discounted DP problem, whose states are the state-control pairs \((i, u)\) in addition to the original states. We may thus apply the approximate PI methods discussed earlier. For this, we need to introduce a parametric architecture \(\tilde{Q}(i, u, r)\). This architecture could be linear feature-based, or nonlinear such as one that uses a neural network.

We have already discussed in Section 4.7.3 a model-free approximate PI method that is based on Q-factors and least squares training/regression. There are also optimistic approximate PI methods, which use a policy for a limited number of stages with cost function approximation for the remaining states, and/or a few samples in between policy updates.

As an example, let us consider an extreme version that uses a single sample between policy updates. At the start of iteration \(k\), we have the current parameter vector \(r^k\), we are at some state \(i^k\), and we have chosen a control \(u^k\). Then:

1. We simulate the next transition \((i^k, i^{k+1})\) using the transition probabilities \(p_{i^k j}(u^k)\).
2. We generate the control \(u^{k+1}\) with the minimization
   \[
   u^{k+1} \in \arg \min_{u \in U(i^{k+1})} \tilde{Q}(i^{k+1}, u, r^k).
   \]
   [In some schemes, \(u^{k+1}\) is chosen with a small probability to be a random element of \(U(i^{k+1})\) or one that attains within some \(\epsilon\) the minimum above, in order to enhance exploration.]
3. We update the parameter vector via
   \[
   r^{k+1} = r^k - \gamma^k \nabla \tilde{Q}(i^k, u^k, r^k) q_k,
   \]
   where \(\gamma^k\) is a positive stepsize, \(\nabla(\cdot)\) denotes gradient with respect to \(r\) evaluated at the current parameter vector \(r^k\), and the scalar \(q_k\) is
given by
\[ q_k = \tilde{Q}(i^k, u^k, r^k) - \alpha \tilde{Q}(i^{k+1}, u^{k+1}, r^k) - g(i^k, u^k, i^{k+1}) . \]

The vector \( \nabla \tilde{Q}(i^k, u^k, r^k)q_k \) can be interpreted as an approximate gradient direction at \( r^k \) based on an underlying regression procedure. The scalar \( q_k \) is related to the notion of a temporal difference, which will be discussed in Section 5.5.

The process is now repeated with \( r^{k+1}, i^{k+1}, \) and \( u^{k+1} \) replacing \( r^k, i^k, \) and \( u^k, \) respectively. Note that the simulated state-control pairs \( (i_k, u_k) \) and associated next transition \( (i^k, i^{k+1}) \) using the transition probabilities \( p_{i^kj}(u^k) \), which are generated in Step (1) above, can be reused judiciously over time, in order to save in simulation overhead.

Extreme optimistic schemes of the type just described have been used in practice, and are often referred to as SARSA (State-Action-Reward-State-Action): see e.g., the books [BeT96], [BBD10], [SuB18]. When Q-factor approximation is used, their behavior is very complex, their theoretical convergence properties are unclear, and there are no associated performance bounds in the literature. The method is more commonly used in a less extreme/optimistic form, whereby several (perhaps many) state-control-transition cost-next state samples are collected and suitably averaged before updating the vector \( r^k \).

Other variants of the method attempt to save in sampling effort by storing the generated samples in a buffer and reusing them in some randomized fashion in subsequent iterations (cf. our discussion of exploration in Section 5.3.4). This is also called sometimes experience replay, an idea that has been been used since the early days of RL, both to save in sampling effort and to enhance exploration. The DQN (Deep Q Network) scheme, championed by DeepMind, is based on this idea (the term “Deep” is a reference to DeepMind’s affinity for deep neural networks, but experience replay does not depend on the use of a deep neural network architecture).

### 5.5 ADDITIONAL METHODS - TEMPORAL DIFFERENCES

In this section, we summarize a few additional methods for approximation in value space in infinite horizon problems. These include the simulation-based temporal difference methods for policy evaluation with a linear parametric architecture, whose primary aim is to address a bias-variance trade-off similar to the one discussed in Section 5.3.2. Our presentation is brief, somewhat abstract, and makes use of linear algebra mathematics. It may be skipped without loss of continuity. This is only a summary; it is meant to provide a connection to other material in this chapter, and orientation for further reading into both the optimization and artificial intelligence literature on the subject.
Our main concern in policy evaluation is to solve approximately the Bellman equation corresponding to a given policy \( \mu \). Thus, for discounted problems, we are interested in solving the linear system of equations

\[
J_\mu(i) = \sum_{i=1}^{n} p_{ij}(\mu(i)) \left( g(i, \mu(i), j) + \alpha J_\mu(j) \right), \quad i = 1, \ldots, n,
\]

or in shorthand,

\[
J_\mu = T_\mu J_\mu,
\]

where \( T_\mu \) is the DP operator for \( \mu \), given by

\[
(T_\mu J)(i) = \sum_{i=1}^{n} p_{ij}(\mu(i)) \left( g(i, \mu(i), j) + \alpha J(j) \right), \quad i = 1, \ldots, n.
\]

Let us consider the approximate solution of this equation by parametric approximation (cf. Section 4.7). This amounts to replacing \( J_\mu \) with some vector that lies within the manifold represented by the approximation architecture

\[
\mathcal{M} = \left\{ (\tilde{J}(1, r), \ldots, \tilde{J}(n, r)) \mid \text{all parameter vectors } r \right\}.
\]

The approximate solution of systems of equations within an approximation manifold of the form (5.33) has a long history in scientific computation, particularly when the manifold is linear. A central approach involves the use of projections with respect to a weighted quadratic norm

\[
\|J\|_2^2 = \sum_{i=1}^{n} \xi_i (J(i))^2,
\]

where \( J(i) \) are the components of the vector \( J \) and \( \xi_i \) are some positive weights. The projection of a vector \( J \) onto the manifold \( \mathcal{M} \) is denoted by \( \Pi(J) \). Thus

\[
\Pi(J) \in \arg \min_{V \in \mathcal{M}} \|J - V\|_2^2.
\]

Note that for a nonlinear parametric architecture, such as a neural network, the projection may not exist and may not be unique. However, in the case of a linear architecture, where the approximation manifold \( \mathcal{M} \) is a subspace, the projection does exist and is unique; this is a consequence of the fundamental orthogonal projection theorem of calculus and real analysis.

Let us consider three general approaches for approximation of \( J_\mu \).

(a) Project \( J_\mu \) onto \( \mathcal{M} \) to obtain \( \Pi(J_\mu) \), which is used as an approximation of \( J_\mu \).
(b) Start with some approximation $\hat{J}$ of $J_{\mu}$, perform $N$ value iterations to obtain $T_{\mu}^N \hat{J}$, and project onto $\mathcal{M}$ to obtain $\Pi(T_{\mu}^N \hat{J})$. We then use $\Pi(T_{\mu}^N \hat{J})$ as an approximation to $J_{\mu}$.

(c) Solve a projected version $J_{\mu} = \Pi(T_{\mu}J_{\mu})$ of the Bellman Eq. (5.31), and use the solution of this projected equation as an approximation to $J_{\mu}$. We will also discuss related projected versions that involve other operators in place of $T_{\mu}$.

The preceding three approaches cannot be implemented exactly; for example, (a) is impossible since we do not know the values of $J_{\mu}$. However, it turns out that it is possible to implement these approaches by using a Monte Carlo simulation methodology that is suitable for large problems.

To explain this methodology we first discuss the implementation of the projection operation through sampling for the case where the parametric architecture is linear and $\mathcal{M}$ is a subspace.

**Projection by Monte Carlo Simulation**

We will focus on the case where the manifold $\mathcal{M}$ is a subspace of the form

$$\mathcal{M} = \{ \Phi r \mid r \in \mathbb{R}^m \},$$

(5.36)

where $\mathbb{R}^m$ is the space of $m$-dimensional vectors, and $\Phi$ is an $n \times m$ matrix with rows denoted by $\phi(i)'$, $i = 1, \ldots, n$. Here we use the notational convention that all vectors are column vectors, and prime denotes transposition, so $\phi(i)'$ is an $m$-dimensional row vector, and the subspace $\mathcal{M}$ may be viewed as the space spanned by the $n$-dimensional columns of $\Phi$.

We consider projection with respect to the weighted Euclidean norm of Eq. (5.34), so $\Pi(J)$ is of the form $\Phi r^*$, where

$$r^* \in \arg \min_{r \in \mathbb{R}^m} \| \Phi r - J \|_2^2 = \arg \min_{r \in \mathbb{R}^m} \sum_{i=1}^n \xi_i (\phi(i)' r - J(i))^2. \quad (5.37)$$

By setting to 0 the gradient at $r^*$ of the minimized expression above,

$$2 \sum_{i=1}^n \xi_i \phi(i) (\phi(i)' r^* - J(i)) = 0,$$

we obtain the solution in closed form,

$$r^* = \left( \sum_{i=1}^n \xi_i \phi(i) \phi(i)' \right)^{-1} \sum_{i=1}^n \xi_i \phi(i) J(i), \quad (5.38)$$

assuming that the inverse above exists. The difficulty here is that when $n$ is very large, the matrix-vector calculations in this formula can be very time-consuming.
On the other hand, assuming (by normalizing $\xi$ if necessary) that $\xi = (\xi_1, \ldots, \xi_n)$ is a probability distribution, we may view the two terms in Eq. (5.38) as expected values with respect to $\xi$, and approximate them by Monte Carlo simulation. In particular, suppose that we generate a set of index samples $i^s$, $s = 1, \ldots, q$, according to the distribution $\xi$, and form the Monte Carlo estimates

$$
\frac{1}{q} \sum_{s=1}^{q} \phi(i^s) \phi(i^s) \approx \sum_{i=1}^{n} \xi_i \phi(i) \phi(i)',
\frac{1}{q} \sum_{s=1}^{q} \phi(i^s) \beta^s \approx \sum_{i=1}^{n} \xi_i \phi(i) J(i),
$$

where $\beta^s$ is a “noisy” sample of the exact value $J(i^s)$

$$
\beta^s = J(i^s) + n(i^s).
$$

For the Monte Carlo estimate in the right side of Eq. (5.39) to be asymptotically correct, we must have

$$
\frac{1}{q} \sum_{s=1}^{q} \phi(i^s)n(i^s) \approx 0,
$$

which is implied by a zero sample mean condition for the noise.†

Given the Monte Carlo approximation of the two terms in Eq. (5.38), we can estimate $r^*$ with

$$
\tau = \left( \sum_{s=1}^{q} \phi(i^s) \phi(i^s)' \right)^{-1} \sum_{s=1}^{q} \phi(i^s) \beta^s,
$$

† A suitable zero mean condition for the noise $n(i^s)$ has the form

$$
\lim_{q \to \infty} \frac{\sum_{s=1}^{q} \delta(i^s = i)n(i^s)}{\sum_{s=1}^{q} \delta(i^s = i)} = 0, \quad \text{for all } i = 1, \ldots, n,
$$

where $\delta(i^s = i) = 1$ if $i^s = i$ and $\delta(i^s = i) = 0$ if $i^s \neq i$. It states that the Monte Carlo averages of the noise terms corresponding to every state $i$ are zero. Then the expression in Eq. (5.40) has the form

$$
\frac{1}{q} \sum_{s=1}^{q} \phi(i^s)n(i^s) = \frac{1}{q} \sum_{s=1}^{q} \phi(i) \sum_{s=1}^{q} \delta(i^s = i)n(i^s)
\quad = \frac{1}{q} \sum_{s=1}^{q} \phi(i) \sum_{s'=1}^{q} \delta(i^s' = i) \delta(i^s = i)n(i^s')
\quad \sum_{s=1}^{q} \delta(i^s = i),
$$

and converges to 0 as $q \to \infty$, assuming that each index $i$ is sampled infinitely often so that Eq. (5.41) can be used.
(assuming sufficiently many samples are obtained to ensure the existence of the inverse above).† This is also equivalent to estimating \( r^* \) by approximating the least squares minimization (5.37) with the following least squares training problem

\[
\mathbf{r} \in \arg \min_{\mathbf{r} \in \mathbb{R}^m} \sum_{s=1}^{q} (\phi(i^s)'r - \beta^s)^2. 
\] (5.43)

Thus simulation-based projection can be implemented in two equivalent ways:

(a) Replacing expected values in the exact projection formula (5.38) by simulation-based estimates [cf. Eq. (5.42)].

(b) Replacing the exact least squares problem (5.37) with a simulation-based least squares approximation [cf. Eq. (5.43)].

These dual possibilities of implementing projection by simulation can be used interchangeably. In particular, the least squares training problems considered in this book may be viewed as simulation-based approximate projection calculations.

Generally, we wish that the estimate \( \mathbf{r} \) converges to \( r^* \) as the number of samples \( q \) increases. An important point is that it is not necessary that the simulation produces independent samples. Instead it is sufficient that the long term empirical frequencies by which the indices \( i \) appear in the simulation sequence are consistent with the probabilities \( \xi_i \) of the projection norm, i.e.,

\[
\xi_i = \lim_{k \to \infty} \frac{1}{q} \sum_{s=1}^{q} \delta(i^s = i), \quad i = 1, \ldots, n, 
\] (5.44)

where \( \delta(i^s = i) = 1 \) if \( i^s = i \) and \( \delta(i^s = i) = 0 \) if \( i^s \neq i \).

Another important point is that the probabilities \( \xi_i \) need not be predetermined. In fact, often the exact values of \( \xi_i \) do not matter much, and one may wish to first specify a reasonable and convenient sampling scheme, and let \( \xi_i \) be implicitly specified via Eq. (5.44).

**Projected Equation View of Approximate Policy Evaluation**

Let us now discuss the approximate policy evaluation method for costs of Section 5.3.2 [cf. Eq. (5.22)]. It can be interpreted in terms of a projected equation, written abstractly as

\[
\tilde{J}_\mu \approx \Pi(T^N_\mu \hat{J}),
\] (5.45)

† The preceding derivation and the formula (5.42) actually make sense even if \( \xi = (\xi_1, \ldots, \xi_n) \) has some zero components, as long as the inverses in Eqs. (5.38) and (5.42) exist. This is related to the concept of seminorm projection; see [YuB12] for an approximate DP-related discussion.
Infinite Horizon Reinforcement Learning Chap. 5

where:

(a) \( \hat{J} \) is some initial guess of \( J_\mu \) (the terminal cost function approximation discussed in Section 5.3.2), and \( \hat{J}_\mu \) is the vector

\[
\hat{J}_\mu = (\hat{J}(1, \tau), \ldots, \hat{J}(n, \tau)),
\]

which is the approximate policy evaluation of \( \mu \), used in the policy improvement operation (5.23). Here \( \tau \) is the solution of the training problem (5.22).

(b) \( T_\mu \) is the DP operator corresponding to \( \mu \), which maps a vector \( J = (J(1), \ldots, J(n)) \) into the vector \( T_\mu J \) of Eq. (5.32).

(c) \( T_\mu^N \) denotes the \( N \)-fold application of the operator \( T_\mu \), where \( N \) is the length of the sample trajectories used in the least squares regression problem (5.22). In particular, \( (T_\mu^N \hat{J})(i) \) is the cost associated with starting at \( i \), using \( \mu \) for \( N \) stages, and incurring a terminal cost specified by the terminal cost function \( \hat{J} \). The sample state-cost pairs \( (i^*, \beta^*) \) are obtained from trajectories corresponding to this \( N \)-stage problem.

(d) \( \Pi(T_\mu^N \hat{J}) \) denotes projection of the vector \( T_\mu^N \hat{J} \) on the manifold of possible approximating vectors \( \mathcal{M} \) with respect to a weighted norm, where each weight \( \xi_i \) represents the relative frequency of the state \( i \) as initial state of a training trajectory. This projection is approximated by the least squares regression (5.22). In particular, the cost samples \( \beta^* \) of the training set are noisy samples of the values \( (T_\mu^N \hat{J})(i^*) \), and the projection is approximated with a least squares minimization, to yield the function \( \hat{J}_\mu \) of Eq. (5.45).

Suppose now that \( T_\mu^N \hat{J} \) is close to \( J_\mu \) (which happens if either \( N \) is large or \( \hat{J} \) is close to \( J_\mu \), or both) and the number of samples \( q \) is large (so that the simulation-based regression approximates well the projection operation \( \Pi \)). Then from Eq. (5.45), the approximate evaluation \( \hat{J}_\mu \) of \( \mu \) approaches the projection of \( J_\mu \) on the approximation manifold (5.33), which can be viewed as the best possible approximation of \( J_\mu \) (at least relative to the distance metric defined by the weighted projection norm). This provides an abstract formal rationale for the parametric PI method of Section 5.3.2, which is based on Eq. (5.45).

**TD(\( \lambda \)), LSTD(\( \lambda \)), and LSPE(\( \lambda \))**

Projected equations also fundamentally underlie temporal difference methods (TD for short), a prominent class of simulation-based methods for ap-

† The equation (5.45) assumes that all trajectories have equal length \( N \), and thus does not allow trajectory reuse. If trajectories of different lengths are allowed, the term \( T_\mu^N \) in the equation should be replaced by a more complicated weighted sum of powers of \( T_\mu \); see the paper [YuB12] for related ideas.
proximate evaluation of a policy. Examples of such methods are TD(\(\lambda\)), LSTD(\(\lambda\)), and LSPE(\(\lambda\)), where \(\lambda\) is a scalar with \(0 \leq \lambda < 1\).†

These three methods require a linear parametric approximation architecture \(\tilde{J}_\mu = \Phi r\), and all aim at the same problem. This is the problem of solving a projected equation of the form

\[
\Phi r = \Pi \left( T_\mu^{(\lambda)} \Phi r \right), \tag{5.46}
\]

where \(T_\mu\) is the operator (5.32), \(T_\mu^{(\lambda)}\) \(J\) is defined by

\[
(T_\mu^{(\lambda)} J)(i) = (1 - \lambda) \sum_{\ell=0}^{\infty} \lambda^\ell (T_\mu^{\ell+1} J)(i), \quad i = 1, \ldots, n,
\]

and \(\Pi\) is projection on the approximation subspace

\[\mathcal{M} = \{ \Phi r \mid r \in \mathbb{R}^m \},\]

with respect to some weighted projection norm. One interpretation of the equation \(J = T_\mu^{(\lambda)} J\) is as a multistep version of Bellman’s equation. It has the same solution, \(J_\mu\), as the “one-step” Bellman equation \(J = T_\mu J\), which corresponds to \(\lambda = 0\).

Of course the projected equation (5.46) cannot be solved exactly when the number of states \(n\) is large, since the projection is a high dimensional operation that requires computations of order \(n\). Instead the key idea is to replace the projection by a simulation-based approximate projection, of the type discussed earlier. This yields the equation

\[
\Phi r = \tilde{\Pi} \left( T_\mu^{(\lambda)} \Phi r \right), \tag{5.47}
\]

where \(\tilde{\Pi}\) is the approximate projection obtained by sampling.

For a more concrete description, let the \(i\)th row of the matrix \(\Phi\) be the \(m\)-dimensional row vector \(\phi(i)'\), so that the cost \(J_\mu(i)\) is approximated as the inner product \(\phi(i)'r\):

\[J_\mu(i) \approx \phi(i)'r.\]

Suppose that we collect \(q\) samples of initial states \(i^s, s = 1, \ldots, q\), together with the corresponding transition costs \(g(i^s, i^{s+1}), s = 1, \ldots, q\). Then the parameter vector \(\tilde{r}\) that solves Eq. (5.47) satisfies

\[
\tilde{r} \in \arg \min_r \sum_{s=1}^{q} \left( \phi(i^s)'r - \text{sample of } (T_\mu^{(\lambda)} \Phi \tilde{r})(i^s) \right)^2, \tag{5.48}
\]

† TD stands for “temporal difference,” LSTD stands for “least squares temporal difference,” and LSPE stands for “least squares policy evaluation.”
Infinite Horizon Reinforcement Learning Chap. 5

[cf. Eq. (5.43)], and defines the approximate evaluation $\Phi^r$ of $J_\mu$. This relation can be expressed as a linear equation, which in principle can be solved in closed form [cf. Eq. (5.42)], and indeed LSTD($\lambda$) does exactly that. By contrast LSPE($\lambda$) and TD($\lambda$) solve this relation iteratively.

We will first give a high level description of the three methods, and then provide a more concrete description for the simpler case where $\lambda = 0$.

(a) The LSTD($\lambda$) method, after the $q$ samples have been collected, solves the relation (5.48) by matrix inversion, taking advantage of the fact that this relation can be written as a linear equation. In particular, it can be written as

$$ C^r = d, \quad \text{(5.49)} $$

where $C$ is some $m \times m$ square matrix, and $d$ is an $m$-dimensional vector. The components of $C$ and $d$ are explicitly computed, and LSTD($\lambda$) produces the approximate cost function $\tilde{J}_\mu(i) = \Phi^r$ where $\Phi^r = C^{-1}d$ is the solution of Eq. (5.49).

(b) The LSPE($\lambda$) method solves the projected equation (5.46) by using a simulation-based projected value iteration,

$$ J_{k+1} = \Pi(T^{(\lambda)} \mu J_k). \quad \text{(5.50)} $$

Here the projection is implemented iteratively, with sampling-based least squares regression, in a manner that resembles the incremental aggregated method of Section 3.1.3.

(c) The TD($\lambda$) method is a simpler iterative stochastic approximation method for solving the linear equation (5.49). It can also be viewed as a stochastic gradient method, or as a stochastic version of the proximal algorithm for solving this linear equation (the parameter $\lambda$ is related to the penalty parameter of the proximal algorithm; see the author’s papers [Ber16c] and [Ber18d]).

An interesting question is how to select $\lambda$ and what is its role. There is a bias-variance tradeoff here, similar to the one we discussed in Section 5.3.2. We will address this issue later in this section.

**TD(0), LSTD(0), and LSPE(0)**

Let us describe in more detail LSTD(0) for evaluation of a given policy $\mu$. We assume that the simulation generates a sample sequence of $q$ transitions using $\mu$:

$$(i^1, i^2), (i^2, i^3), \ldots, (i^q, i^{q+1}),$$

with corresponding transition costs

$$g(i^1, i^2), g(i^2, i^3), \ldots, g(i^q, i^{q+1}).$$
Here, to simplify notation, we do not show the dependence of the transition costs on the control applied by $\mu$. Let the $i$th row of the matrix $\Phi$ be the $m$-dimensional row vector $\phi(i)'$, so that the cost $J_\mu(i)$ is approximated as the inner product $\phi(i)'r$:

$$J_\mu(i) \approx \phi(i)'r.$$  

Since $\lambda = 0$, we have $T^{(\lambda)} = T$, the samples of $T_\mu\Phi r$ in Eq. (5.48) are

$$g(i^s, i^{s+1}) + \alpha \phi(i^{s+1})\tau,$$

and the least squares problem in Eq. (5.48) has the form

$$\min_r \sum_{s=1}^q (\phi(i^s)'r - g(i^s, i^{s+1}) - \alpha \phi(i^{s+1})\tau)^2. \tag{5.51}$$

By setting the gradient of the minimized expression to zero, we obtain the condition for $\tau$ to attain the above minimum:

$$\sum_{s=1}^q \phi(i^s)(\phi(i^s)\tau - g(i^s, i^{s+1}) - \alpha \phi(i^{s+1})\tau) = 0. \tag{5.52}$$

Solving this equation for $\tau$ yields the LSTD(0) solution:

$$\tau = \left(\sum_{s=1}^q \phi(i^s)(\phi(i^s) - \alpha \phi(i^{s+1})))\right)^{-1} \sum_{s=1}^q \phi(i^s)g(i^s, i^{s+1}). \tag{5.53}$$

Note that the inverse in the preceding equation must exist for the method to be well-defined; otherwise the iteration has to be modified. A modification may also be needed when the matrix inverted is nearly singular; in this case the simulation noise may introduce serious numerical problems. Various methods have been developed to deal with the near singularity issue; see Wang and Bertsekas [WaB13a], [WaB13b], and the DP textbook [Ber12], Section 7.3.

The expression

$$d^s(\tau) = \phi(i^s)\tau - g(i^s, i^{s+1}) - \alpha \phi(i^{s+1})\tau$$  

that appears in the least squares sum minimization (5.51) and Eq. (5.52) is referred to as the \textit{temporal difference associated with the} $s$th transition and parameter vector $\tau$. In the artificial intelligence literature, temporal differences are viewed as fundamental to learning and are accordingly interpreted, but we will not go further in this direction; see the RL textbooks that we have cited.

The LSPE(0) method is similarly derived. It consists of a simulation-based approximation of the projected value iteration method

$$J_{k+1} = \Pi(T_\mu J_k),$$
Infinite Horizon Reinforcement Learning Chap. 5

At the $k$th iteration, it uses only the samples $s = 1, \ldots, k$, and updates the parameter vector according to

$$r^{k+1} = r^k - \left( \sum_{s=1}^{k} \phi(i^s)\phi(i^s)' \right)^{-1} \sum_{s=1}^{k} \phi(i^s)d^s(r^k), \quad k = 1, 2, \ldots, (5.55)$$

where $d^s(r^k)$ is the temporal difference of Eq. (5.54), evaluated at the iterate $r^k$; the form of this iteration is derived similar to the case of LSTD(0). After $q$ iterations, when all the samples have been processed, the vector $r^q$ obtained is the one used for the approximate evaluation of $J_\mu$. Note that the inverse in Eq. (5.55) can be updated economically from one iteration to the next, using fast linear algebra operations (cf. the discussion of the incremental Newton method in Section 3.1.3).

Overall, it can be shown that LSTD(0) and LSPE(0) [with efficient matrix inversion in Eq. (5.55)] require essentially identical amount of work to process the $q$ samples associated with the current policy $\mu$ [this is also true for the LSTD(\lambda) and LSPE(\lambda) methods; see [Ber12], Section 6.3]. An advantage offered by LSPE(0) is that because it is iterative, it allows carrying over the final parameter vector $r^q$, as a “hot start” when passing from one policy evaluation to the next, in the context of an approximate PI scheme.

The TD(0) method has the form

$$r^{k+1} = r^k - \gamma^k \phi(i^k)d^k(r^k), \quad k = 1, 2, \ldots, (5.56)$$

where $\gamma^k$ is a diminishing stepsize parameter. It can be seen that TD(0) resembles an incremental gradient iteration for solving the least squares training problem (5.51), but with $\Phi^*$ replaced by the current iterate $r^k$. The reason is that the gradient of the typical $k$th term in the least squares sum of Eq. (5.51) is the vector $\phi(i^k)d^k(r^k)$ that appears in the TD(0) iteration (5.56) (cf. Section 3.1.3). Thus at each iteration, TD(0) uses only one sample, and changes $r^k$ in the opposite direction to the corresponding incremental gradient using a stepsize $\gamma^k$ that must be carefully controlled.

By contrast the LSPE(0) iteration (5.55) uses the full sum

$$\sum_{s=1}^{k} \phi(i^s)d^s(r^k),$$

which may be viewed as an aggregated incremental method, with scaling provided by the matrix $\left( \sum_{s=1}^{k} \phi(i^s)\phi(i^s)' \right)^{-1}$. This explains why TD(0) is generally much slower and more fragile than LSPE(0). On the other hand TD(0) is simpler than both LSTD(0) and LSPE(0), and does not require a matrix inversion, which may be inconvenient when the column dimension $m$ of $\Phi$ is large.
The properties, the analysis, and the implementation of TD methods in the context of approximate PI are quite complicated. In particular, the issue of exploration is important and must be addressed. Moreover, there are convergence, oscillation, and reliability issues to contend with. LSTD(\(\lambda\)) relies on matrix inversion and not on iteration, so it does not have a serious convergence issue, but the system (5.49) may be singular or near singular, in which case very accurate simulation is needed to approximate \(C\) well enough for its inversion to be reliable; remedies for the case of a singular or near singular system are discussed in the papers [WaB13a], [WaB13b] (see also [Ber12], Section 7.3). LSPE(\(\lambda\)) has a convergence issue because the mapping \(\Pi T_\mu(\lambda)\) may not be a contraction mapping (even though \(T_\mu\) is) and the projected value iteration (5.50) may not be convergent. It turns out that the mapping \(\Pi T_\mu(\lambda)\) is guaranteed to be a contraction for \(\lambda\) sufficiently close to 1, so the convergence difficulty may be circumvented by suitably increasing \(\lambda\).

**Direct and Indirect Policy Evaluation Methods**

In trying to compare the approximate policy evaluation methods discussed in this section, we may draw a distinction between direct methods, which aim to compute approximately the projection \(\Pi(J_\mu)\), and indirect methods, which try to solve the projected equation (5.46).

The method of Section 4.7.2 is direct and is based on Eq. (5.45). In particular, as \(N \to \infty\) and \(q \to \infty\), it yields the approximate evaluation \(\Pi(J_\mu)\). The TD methods are indirect, and aim at computing the solution of the projected equation (5.46). The solution of this equation is of the form \(\Phi r_\lambda^*\), where the parameter vector \(r_\lambda^*\) depends on \(\lambda\). In particular the projected equation solution \(\Phi r_\lambda^*\) is different from \(\Pi(J_\mu)\). It can be shown that it satisfies the error bound

\[
\|J_\mu - \Phi r_\lambda^*\|_\xi \leq \frac{1}{\sqrt{1 - \alpha_\lambda}} \|J_\mu - \Pi J_\mu\|_\xi,
\]

(5.57)

where

\[
\alpha_\lambda = \frac{\alpha(1 - \lambda)}{1 - \alpha \lambda}
\]

and \(\| \cdot \|_\xi\) is a special projection norm of the form (5.34), where \(\xi\) is the steady-state probability distribution of the controlled system Markov chain under policy \(\mu\). Moreover as \(\lambda \to 1\) the projected equation solution \(\Phi r_\lambda^*\) approaches \(\Pi(J_\mu)\). Based on this fact, methods which aim to compute \(\Pi(J_\mu)\), such as the direct method of Section 5.3.2 are sometimes called TD(1). We refer to [Ber12], Section 6.3, for an account of this analysis, which is beyond the scope of this book.

The difference \(\Phi r_\lambda^* - \Pi(J_\mu)\) is commonly referred to as the bias and is illustrated in Figure 5.5.1. As indicated in this figure and as the estimate
Figure 5.5.1 Illustration of the bias-variance tradeoff in estimating the solution of the projected equation for different values of $\lambda$. As $\lambda$ increases from $\lambda = 0$ towards $\lambda = 1$, the solution $\Phi^{\lambda}_r$ of the projected equation $\Phi_r = \Pi J^T(\lambda)(\Phi_r)$ approaches the projection $\Pi J_\mu$. The difference $\Phi^{\lambda}_r - \Pi J_\mu$ is the bias, and it decreases to 0 as $\lambda$ approaches 1, while the simulation error variance increases.

(5.57) suggests, there is a bias-variance tradeoff. As $\lambda$ is decreased, the solution of the projected equation (5.46) changes and more bias is introduced relative to the “ideal” approximation $\Pi J_\mu$ (this bias can be embarrassingly large as shown by examples in the paper [Ber95]). At the same time, however, the simulation samples of $T^{(\lambda)}_\mu J$ contain less noise as $\lambda$ is decreased. This provides another view of the bias-variance tradeoff, which we discussed in Section 5.3.2 in connection with the use of short trajectories.

5.6 EXACT AND APPROXIMATE LINEAR PROGRAMMING

Another method for exact solution of infinite horizon DP problems is based on linear programming ideas. In particular, $J^*$ can be shown to be the unique optimal solution of a certain linear program. Focusing on $\alpha$-discounted problems, the key idea is that $J^*$ is the “largest” (on a component-by-component basis) vector $J$ that satisfies the constraint

$$J(i) \leq \sum_{j=1}^{n} p_{ij}(u)\left(g(i, u, j) + \alpha J(j)\right), \quad \text{for all } i = 1, \ldots, n \text{ and } u \in U(i),$$

(5.58)
so that $J^*(1), \ldots, J^*(n)$ solve the linear program
\[
\text{maximize } \sum_{i=1}^{n} J(i) \tag{5.59}
\]
subject to the constraint (5.58),
(see Fig. 5.6.1).

To verify this, let us use the VI algorithm to generate a sequence of vectors $J_k = (J_k(1), \ldots, J_k(n))$ starting with an initial condition vector $J_0 = (J_0(1), \ldots, J_0(n))$ such that
\[
J_0(i) \leq \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u) \left( g(i, u, j) + \alpha J_0(j) \right) = J_1(i), \quad \text{for all } i.
\]
This inequality can be used to show that
\[
J_0(i) \leq J_1(i) \leq \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u) \left( g(i, u, j) + \alpha J_1(j) \right) = J_2(i), \quad \text{for all } i,
\]
and similarly
\[
J(i) = J_0(i) \leq J_k(i) \leq J_{k+1}(i), \quad \text{for all } i.
\]
Since \( J_k(i) \) converges to \( J^*(i) \) as \( k \to \infty \), it follows that we will also have
\[
J(i) = J_0(i) \leq J^*(i), \quad \text{for all } i.
\]
Thus out of all \( J \) satisfying the constraint (5.58), \( J^* \) is the largest on a component-by-component basis.

Unfortunately, for large \( n \) the dimension of the linear program (5.59) can be very large and its solution can be impractical, particularly in the absence of special structure. In this case, we may consider finding an approximation to \( J^* \), which can be used in turn to obtain a (suboptimal) policy through approximation in value space.

One possibility is to approximate \( J^*(i) \) with a linear feature-based architecture
\[
\tilde{J}(i, r) = \sum_{\ell=1}^{m} r_\ell \phi_\ell(i),
\]
where \( r = (r_1, \ldots, r_m) \) is a vector of parameters, and for each state \( i \), \( \phi_\ell(i) \) are some features. It is then possible to determine \( r \) by using \( \tilde{J}(i, r) \) in place of \( J^* \) in the preceding linear programming approach. In particular, we compute \( r \) as the solution of the program
\[
\begin{align*}
\text{maximize} & \quad \sum_{i \in \tilde{I}} \tilde{J}(i, r) \\
\text{subject to} & \quad \tilde{J}(i, r) \leq \sum_{i=1}^{n} p_{ij}(u)(g(i, u, j) + \alpha \tilde{J}(j, r)), \quad i \in \tilde{I}, \ u \in \tilde{U}(i),
\end{align*}
\]
where \( \tilde{I} \) is either the state space \( I = \{1, \ldots, n\} \) or a suitably chosen subset of \( I \), and \( \tilde{U}(i) \) is either \( U(i) \) or a suitably chosen subset of \( U(i) \). This is a linear program because \( \tilde{J}(i, r) \) is assumed linear in the parameter vector \( r \).

The major difficulty with this approximation approach is that while the dimension of \( r \) may be moderate, the number of constraints can be extremely large. It can be as large as \( nm \), where \( n \) is the number of states and \( m \) is the maximum number of elements of the control constraint sets \( U(i) \). Thus for a large problem it is essential to reduce drastically the number of constraints. Random sampling methods may be used to select a suitable subset of the constraints to enforce (perhaps using some known suboptimal policies), and progressively enrich the subset as necessary. With such constraint sampling schemes, the linear programming approach may be practical even for problems with a very large number of states. Its application, however, may require considerable sophistication, and a substantial amount of computation (see de Farias and Van Roy [DFV03], [DFV04], [DeF04]).

We finally mention the possibility of using linear programming to evaluate approximately the cost function \( J_\mu \) of a stationary policy \( \mu \) in the context of approximate PI. The motivation for this is that the linear program to evaluate a given policy involves fewer constraints.
5.7 APPROXIMATION IN POLICY SPACE

We will now consider briefly an alternative to approximation in value space: approximation within the space of policies, focusing on \(\alpha\)-discounted problems. In particular, we parametrize stationary policies with a parameter vector \(r\) and denote them by \(\tilde{\mu}(r)\), with components

\[
\tilde{\mu}(i, r), \quad i = 1, \ldots, n.
\]

The parametrization may be feature-based and/or may involve a neural network. The idea is then to optimize some measure of performance with respect to the parameter \(r\).

Note that it is possible for a suboptimal control scheme to employ both types of approximation: in policy space and in value space, with a distinct architecture for each case (examples of such schemes have been discussed briefly in Sections 2.1.5 and 5.3.3). When neural networks are used, this is known as the simultaneous use of a “policy network” (or “actor network”) and a “value network” (or “critic network”), each with its own set of parameters (see the following discussion on expert training).

Let us provide some examples where policy parametrizations are natural and/or have been successful in practice.

Example 5.7.1: (Supply chain parametrization)

There are many problems where the general structure of an optimal or near-optimal policy is known through analysis or insight into the problem’s structure. An important case are supply chain systems involving production, inventory, and retail centers that are connected with transportation links. A simple example is illustrated in Fig. 5.7.1. Here a retail center places orders to the production center, depending on current stock. There may be orders in transit, and demand and delays can be stochastic. Such a problem can be formulated by DP but can be very difficult to solve exactly. However, intuitively, a near-optimal policy has a simple form: when the retail inventory goes below some critical level \(r_1\), order an amount to bring the inventory to a target level \(r_2\). Here a policy is specified by the parameter vector \(r = (r_1, r_2)\), and can be trained by one of the methods of this section. This type of approach readily extends to the case of a complex network of production/retail centers, multiple products, etc.
Example 5.7.2: (PID control)

A popular and time-honored scheme for control system design is the PID (Proportional-Integral-Derivative) controller. It is widely used (among others) for maintaining the output of a single-input single-output dynamical system around a set point. The internal description of the system is not assumed known, but the error $e_k$ between the output and the set point at time $k$ can be measured.

The input/control $u_k$ applied at time $k$ is then the sum of three terms that depend on the observed errors $e_0, \ldots, e_k$ up to time $k$. The first term, called proportional, is $r_p e_k$ where $r_p$ is some constant. The second term, called integral, is $r_i \sum_{m=0}^{k} e_m$ (i.e., it is proportional to a running sum of the errors), where $r_i$ is another constant. The third term, called derivative, is $r_d d_k$, where $r_d$ is a third constant, and $d_k$ is the most recent error, $d_k = e_k - e_{k-1}$, or a damped version thereof generated by the recursion

$$d_k = (1 - \beta)d_{k-1} + \beta(e_k - e_{k-1}),$$

where $\beta$ is damping factor with $\beta < 1$ and $\beta \approx 1$. The three constants $(r_p, r_i, r_d)$ can be viewed as a parametrization of the controller, and they can be tuned to achieve good performance (typically a satisfactory measure of stability and transient behavior quality). Note that this is a model-free scheme: a model of the system is not assumed to be known.

There are many variations and extensions of the PID scheme, and many practical methods for tuning the parameters have been developed over the years, some of them manual. Our purpose here is to point out that the PID scheme can also be viewed within the context of approximation in policy space, which brings to bear the methodology of this section. The literature contains several discussions along this line.

Example 5.7.3: (Policy parametrization through cost parametrization)

In an important approach for parametrization of policies we start with a parametric cost function approximation $\tilde{J}(j, r)$. We then define a policy parametrization through the one-step lookahead minimization

$$\tilde{\mu}(i, r) \in \arg \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u) \left( g(i, u, j) + \alpha \tilde{J}(j, r) \right),$$

where $\tilde{J}$ is a function of a given form that depends on $r$. For example, $\tilde{J}$ may be a linear feature-based architecture, with features possibly obtained through a separately trained neural network. The policies $\tilde{\mu}(r)$ thus defined form a class of one-step lookahead policies parametrized by $r$. We may then determine $r$ through some form of policy training method if this is convenient.
Example 5.7.4: (Policy parametrization and multiagent problems)

An important insight to keep in mind is that approximation in policy space is a more broadly applicable methodology than approximation in value space. In particular, policy space approximation is not tied to the cost-to-go formalism of DP, which is the essential framework for the development of the approximate VI and PI methodologies. As a result, the idea of policy parametrization applies to problems that share some structure with the finite and infinite horizon DP problems we have discussed so far, but do not admit a formal treatment by DP.

An example is multiagent problems, where there is a dynamic system whose state evolves in time, and there are multiple decision makers that do not share the same information about the system state. Each agent has access to local observations, and may receive some of the other agents’ observations (or a summary thereof) with delay: the agent’s decision at each time is based on just the information available to him/her at that time. This type of information pattern is unconventional and is not allowed in the DP context. Consequently there is no legitimate DP framework, and attendant Bellman’s equation, and VI and PI methods for this type of problem. Still, however, while approximation in value space does not apply, it is possible to parametrize the policies of the agents and set up a framework for determining these parameters through some form of optimization of the type to be discussed.

In what follows we will discuss two training approaches for approximation in policy space. The first approach (Section 5.7.1) is based on optimization over the parameter \( r \) of some measure of cost derived from the given DP problem. The second approach (Section 5.7.2) is less ambitious and is reminiscent of supervised learning. We collect state-control data produced by a human or software “expert,” and we obtain the parameter \( r \) by matching approximately the decision making of the expert through some least squares error minimization. In Section 5.7.3, we use approximation in policy space for policy improvement within a PI framework. Here we apply the expert training approach of Section 5.7.2, with rollout being used as a software expert.

5.7.1 Training by Cost Optimization - Policy Gradient, Cross-Entropy, and Random Search Methods

According to the first approach, we parametrize the policies by the parameter vector \( r \), and we optimize the corresponding expected cost over \( r \). In particular, we determine \( r \) through the minimization

\[
\min_r E \{ J_{\tilde{\mu}(r)}(i_0) \}, \tag{5.60}
\]

where \( J_{\tilde{\mu}(r)}(i_0) \) is the cost of the policy \( \tilde{\mu}(r) \) starting from the initial state \( i_0 \), and the expected value above is taken with respect to a suitable probability
Here policies are parametrized with a parameter vector \( r \) and denoted by \( \tilde{\mu}(r) \), with components \( \tilde{\mu}(i, r), i = 1, \ldots, n \). Each parameter value \( r \) determines a policy \( \mu_r \), and a cost \( J_{\tilde{\mu}(r)}(i_0) \) for each initial state \( i_0 \), as indicated in the figure. The optimization approach determines \( r \) through the minimization

\[
\min_r E\{J_{\tilde{\mu}(r)}(i_0)\},
\]

where the expected value above is taken with respect to a suitable probability distribution of \( i_0 \).}

**Gradient Methods for Cost Optimization**

Let us first consider methods that perform the minimization (5.60) by using a gradient method, and for simplicity let us assume that the initial condition \( i_0 \) is known. Thus the aim is to minimize \( J_{\tilde{\mu}(r)}(i_0) \) over \( r \) by using the gradient method

\[
r^{k+1} = r^k - \gamma^k \nabla J_{\tilde{\mu}(r)}(i_0), \quad k = 0, 1, \ldots,
\]

assuming that \( J_{\tilde{\mu}(r)}(i_0) \) is differentiable with respect to \( r \). Here \( \gamma^k \) is a positive steps size parameter, and \( \nabla(\cdot) \) denotes gradient with respect to \( r \) evaluated at the current iterate \( r^k \).

The difficulty with this method is that the gradient \( \nabla J_{\tilde{\mu}(r)}(i_0) \) may not be explicitly available. In this case, the gradient must be approximated by finite differences of cost function values \( J_{\tilde{\mu}(r)}(i_0) \). Unfortunately, in the case of a stochastic problem, the cost function values may be computable only through Monte Carlo simulation. This may introduce a large amount of noise, so it is likely that many samples will need to be averaged in
order to obtain sufficiently accurate gradients, thereby making the method inefficient. On the other hand, when the problem is deterministic, this difficulty does not appear, and the use of the gradient method (5.61) or other methods that do not rely on the use of gradients (such as coordinate descent) is facilitated.

There is extensive literature on alternative and more efficient policy gradient methods for stochastic problems, which are based on gradient approximations through sampling. A popular type of method is based on the use of randomized policies [i.e., policies that map a state $i$ to a probability distribution over the set of controls $U(i)$, rather than mapping onto a single control].† The method also uses a convenient gradient formula that involves the natural logarithm of the sampling distribution, and is known as the log-likelihood trick. We will next provide an outline of the ideas underlying this method.

### Policy Gradient Methods for Randomized Policies

The detailed description and analysis of randomized policies and the associated policy gradient methods are beyond our scope. To get a sense of the general principle underlying this gradient-based approach, let us digress from the DP context of this chapter, and consider the generic optimization problem

$$\min_{z \in Z} F(z),$$

where $Z$ is a subset of the $m$-dimensional space $\mathbb{R}^m$, and $F$ is some real-valued function over $\mathbb{R}^m$.

We will take the unusual step of converting this problem to the stochastic optimization problem

$$\min_{p \in \mathcal{P}_Z} E_p \{ F(z) \},$$

where $z$ is viewed as a random variable, $\mathcal{P}_Z$ is the set of probability distributions over $Z$, $p$ denotes the generic distribution in $\mathcal{P}_Z$, and $E_p \{ \cdot \}$ denotes expected value with respect to $p$. Of course this enlarges the search space from $Z$ to $\mathcal{P}_Z$, but it allows the use of randomization schemes and simulation-based methods, even if the original problem is deterministic.

† The AlphaGo and AlphaZero programs (Silver et al. [SHM16], [SHS17]) also use randomized policies, and a policy adjustment scheme that involves incremental changes along “directions of improvement.” However, these changes are implemented through the MCTS algorithm used by these programs, without the explicit use of a gradient (see the discussion in Section 2.4.2). Thus it may be said that the AlphaGo and AlphaZero programs involve a form of approximation in policy space (as well as approximation in value space), which bears resemblance but cannot be classified as a policy gradient method.
At this point it is not clear how the stochastic optimization problem (5.62) relates to our stochastic DP context of this chapter. We will return to this question later, but for the purpose of orientation, we note that to obtain a problem of the form (5.62), we must enlarge the set of policies to include randomized policies, mapping a state $i$ into a probability distribution over the set of controls $U(i)$.

Suppose now that we restrict attention to a subset $\hat{\mathcal{P}}_Z \subset \mathcal{P}_Z$ of probability distributions $p(z;r)$ that are parametrized by some continuous parameter $r$, e.g., a vector in some $m$-dimensional space. In other words, we approximate the stochastic optimization problem (5.62) with the restricted problem

$$\min_r E_{p(z;r)} \{ F(z) \}.$$  

Then we may use a gradient method for solving this problem, such as

$$r^{k+1} = r^k - \gamma^k \nabla \left( E_{p(z;r^k)} \{ F(z) \} \right), \quad k = 0, 1, \ldots, \quad (5.63)$$

where $\nabla (\cdot)$ denotes gradient with respect to $r$ of the function in parentheses, evaluated at the current iterate $r^k$. Here is an example of an incremental variant of this method, which connects with a type of derivative-free methods that use only two sample function values per iteration. These are generally faster that methods that use a finite difference approximation of the entire cost function gradient, and require $2n$ function values per iteration; see the book by Spall [Spa03] for a detailed discussion.

**Example 5.7.5 (An Incremental Method Based on Two Function Values per Iteration)**

For simplicity we first assume that $z$ and $r$ are scalars, and later note the multidimensional case where $z$ and $r$ belong to $\mathbb{R}^n$. Assume that $p(z;r)$ is a discrete distribution that is symmetric around $r$ and is concentrated with probabilities $p_i$ at the points $r + \epsilon_i$ and $r - \epsilon_i$, where $\epsilon_1, \ldots, \epsilon_m$ are some small positive scalars. Thus we have

$$E_{p(z;r)} \{ F(z) \} = \sum_{i=1}^m p_i \left( F(r + \epsilon_i) + F(r - \epsilon_i) \right),$$

and

$$\nabla \left( E_{p(z;r)} \{ F(z) \} \right) = \sum_{i=1}^m p_i \left( \nabla F(r + \epsilon_i) + \nabla F(r - \epsilon_i) \right).$$

By approximating the gradients by finite differences,

$$\nabla F(r + \epsilon_i) \approx \frac{F(r + \epsilon_i) - F(r)}{\epsilon_i}, \quad \nabla F(r - \epsilon_i) \approx \frac{F(r) - F(r - \epsilon_i)}{\epsilon_i},$$
we have the gradient approximation

$$\nabla E_{p(z,r)} \{ F(z) \} \approx \sum_{i=1}^{m} \frac{p_i}{\epsilon_i} (F(r+\epsilon_i) - F(r-\epsilon_i)).$$

Thus, we can write the gradient iteration (5.63) as

$$r^{k+1} = r^k - \gamma^k \sum_{i=1}^{m} \frac{p_i}{\epsilon_i} (F(r^k+\epsilon_i) - F(r^k-\epsilon_i)), \quad k = 0, 1, \ldots, (5.64)$$

with the approximation error due to finite differencing subsumed in the step-size $\gamma^k$. One possible sample-based/incremental version of this iteration is

$$r^{k+1} = r^k - \gamma^k (F(r^k + \epsilon_{i,k}) - F(r^k - \epsilon_{i,k})), \quad (5.65)$$

where $i^k$ is an index generated with probabilities that are proportional to $p_{i,k}/\epsilon_{i,k}$. This algorithm uses one out of the $m$ terms of the gradient in Eq. (5.64).

The extension to the multidimensional case, where $z$ and $r$ belong to $\mathbb{R}^n$, is straightforward: at $r^k$ we first choose randomly a direction $d^k$ on the surface of the unit sphere of $\mathbb{R}^n$, and then sample $z^k$ along the line $\{r^k + \beta d^k \mid \beta \in \mathbb{R}\}$ similar to the iteration (5.65):

$$r^{k+1} = r^k - \gamma^k (F(r^k + \epsilon_{i,k} d^k) - F(r^k - \epsilon_{i,k} d^k)). \quad (5.66)$$

Algorithms of this type are known as random direction search methods. They use only two cost function values per iteration, and have been the subject of considerable research; see the end-of-chapter references.

We now return to the gradient method (5.63) and develop a sample-based variant that is better suited for our infinite horizon DP context.
Assuming for notational convenience that \( p(z; r) \) is a discrete distribution, and interchanging gradient and expected value, we have

\[
\nabla \left( E_{p(z; r)} \{ F(z) \} \right) = \sum_{z \in Z} \nabla p(z; r) F(z)
\]

\[
= \sum_{z \in Z} p(z; r) \frac{\nabla p(z; r)}{p(z; r)} F(z)
\]

\[
= \sum_{z \in Z} p(z; r) \nabla \left( \log p(z; r) \right) F(z),
\]

and finally

\[
\nabla \left( E_{p(z; r)} \{ F(z) \} \right) = E_{p(z; r)} \left\{ \nabla \left( \log p(z; r) \right) F(z) \right\},
\]

where for any given \( z \), \( \nabla \left( \log p(z; r) \right) \) is the gradient with respect to \( r \) of the function \( \log p(z; r) \), evaluated at \( r \) (assumed to exist).

The preceding formula suggests an incremental implementation of the gradient iteration (5.63) that approximates the expected value in the right side above with a single sample. They are conceptually similar to the incremental method of Example 5.7.5. The typical iteration of this method is as follows.

### Sample-Based Gradient Method for Parametric Approximation of \( \min_{z \in Z} F(z) \)

Let \( r^k \) be the current parameter vector.

(a) Obtain a sample \( z^k \) according to the distribution \( p(z; r^k) \).

(b) Compute the gradient \( \nabla \left( \log p(z^k; r^k) \right) \).

(c) Iterate according to

\[
r^{k+1} = r^k - \gamma^k \nabla \left( \log p(z^k; r^k) \right) F(z^k). \tag{5.67}
\]

The advantage of the preceding sample-based method is its simplicity and generality. It allows the use of parametric approximation for any minimization problem, as long as the logarithm of the sampling distribution \( p(z; r) \) can be differentiated with respect to \( r \), and samples of \( z \) can be obtained using the distribution \( p(z; r) \).

A striking observation (and also a potential cause for concern) is that in iteration (5.67), \( r \) is adjusted along a direction that does not involve at
all the gradient of $F$, only the gradient of the logarithm of the sampling distribution! Still, however, in specific problems the resemblance of iteration (5.67) with gradient descent-type methods that use random search directions can be made explicit, as has been illustrated by Example 5.7.5.

Let us emphasize that the iteration (5.67) requires the sample cost values $F(z^k)$ but not the gradient of $F$. As a result the iteration has a model-free character: we don’t need to know the form of the function $F$ as long as we have a simulator that produces the cost function value $F(z)$ for any given $z$. This is also a major advantage offered by many random search methods. There are, however, some challenging issues to consider.

The first of these is that the problem solved is a randomized version of the original. If the gradient iteration (5.67) produces a parameter $\tau$ in the limit and the distribution $p(z; \tau)$ is not atomic (i.e., it is not concentrated at a single point), then a solution $z \in Z$ must be extracted from $p(z; \tau)$. In the SSP and discounted problems of this chapter, the subset $\tilde{P}_Z$ of parametric distributions typically contains the atomic distributions, while it can be shown that minimization over the set of all distributions $P_Z$ produces the same optimal value as minimization over $Z$ (the use of randomized policies does not improve the optimal cost of the problem), so this difficulty does not arise.

Another issue is how to design the approximation architecture and how to collect the samples $z^k$. Different methods must strike a balance of convenient implementation, and a reasonable guarantee that the search space $Z$ is sufficiently well explored.

Finally, we must deal with the issue of efficient computation of the sampled gradient

$$\nabla \left( \log \left( p(z^k; r^k) \right) \right).$$

In the context of DP, including the SSP and discounted problems that we have been dealing with, there are some specialized procedures and corresponding parametrizations to approximate this gradient conveniently. The following is an example, where we assume that a model for the transition probabilities is available.

**Example 5.7.6 (Policy Gradient Method for Discounted DP)**

Consider the $\alpha$-discounted problem and denote by $z$ the infinite horizon state-control trajectory:

$$z = \{i_0, u_0, i_1, u_1, \ldots\}.$$ 

We consider a parametrization of randomized policies with parameter $r$, so the control at state $i$ is generated according to a distribution $p(u \mid i; r)$ over $U(i)$. Then for a given $r$, the state-control trajectory $z$ is a random vector with probability distribution denoted $p(z; r)$. The cost corresponding to the trajectory $z$ is

$$F(z) = \sum_{m=0}^{\infty} \alpha^m g(i_m, u_m, i_{m+1}),$$
and the problem is to minimize
\[ E_{p(z;r)} \{ F(z) \}, \]
over \( r \).

To apply the sample-based gradient method (5.67), given the current iterate \( r^k \), we must generate the sample state-control trajectory \( z^k \), according to the distribution \( p(z; r^k) \), compute the corresponding cost \( F(z^k) \), and also calculate the gradient
\[ \nabla \left( \log \left( p(z^k; r^k) \right) \right). \quad (5.68) \]

Let us assume a model-based context where the transition probabilities \( p_{ij}(u) \) are known, and let us also assume that the logarithm of the randomized policy distribution \( p(u | i; r) \) is differentiable with respect to \( r \). Then the logarithm that is differentiated in Eq. (5.68) can be written as
\[
\log \left( p(z^k; r^k) \right) = \log \prod_{m=0}^{\infty} p_{i_m i_{m+1}}(u_m)p(u_m | i_m; r^k) \\
= \sum_{m=0}^{\infty} \log \left( p_{i_m i_{m+1}}(u_m) \right) + \sum_{m=0}^{\infty} \log \left( p(u_m | i_m; r^k) \right),
\]
and its gradient (5.68), which is needed in the iteration (5.67), is given by
\[ \nabla \left( \log \left( p(z^k; r^k) \right) \right) = \sum_{m=0}^{\infty} \nabla \left( \log \left( p(u_m | i_m; r^k) \right) \right). \quad (5.69) \]

This gradient involves the transition probabilities and the current randomized policy, but does not involve at all the costs per stage.

Thus the policy gradient method (5.67) is very simple to implement: for the given parameter vector \( r^k \), we generate a sample trajectory \( z^k \) using the corresponding randomized policy \( p(u | i; r^k) \), we calculate the corresponding sample cost \( F(z^k) \), and the gradient (5.68) using the expression (5.69), and we update \( r^k \) using Eq. (5.67).

Policy gradient methods for other types of DP problems can be similarly developed, including for model-free contexts. A further discussion is beyond our scope, and we refer to the end-of-chapter literature for a variety of specific methods.

The main drawback of policy gradient methods is potential unreliability due to the stochastic uncertainty corrupting the calculation of the gradients, the slow convergence that is typical of gradient methods in many settings, and the presence of local minima. For this reason, methods based on random search have been considered as potentially more reliable alternatives. Viewed from a high level, random search methods are similar to policy gradient methods in that they aim at iterative cost improvement through sampling. However, they need not involve randomized policies, they are not subject to cost differentiability restrictions, and they offer some global convergence guarantees, so in principle they are not affected much by local minima.
Sec. 5.7  Approximation in Policy Space

Figure 5.7.4 Schematic illustration of the cross-entropy method. At the current iterate $r^k$, we construct an ellipsoid $E_k$ centered at $r^k$. We generate a number of random samples within $E_k$, and we “accept” a subset of the samples that have “low” cost. We then choose $r^{k+1}$ to be the sample mean of the accepted samples, and construct a sample “covariance” matrix of the accepted samples. We then form the new ellipsoid $E_{k+1}$ using this matrix and a suitably enlarged radius, and continue. Notice the resemblance with a policy gradient method: we move from $r^k$ to $r^{k+1}$ in a direction of cost improvement.

Random Search and Cross-Entropy Methods

Let us consider a random search approach for solving the problem

$$\min_r E\{J_{\tilde{\mu}}(\tilde{i})\},$$

cf. Eq. (5.60). Random search methods explore the space of the parameter vector $r$ in some randomized but intelligent fashion. There are several types of such methods for general optimization, and some of them have been suggested for approximate DP. We will briefly describe the cross-entropy method, which has gained considerable attention.

The method bears resemblance to policy gradient methods, in that it generates a parameter sequence $\{r^k\}$ by changing $r^k$ to $r^{k+1}$ along a direction of “improvement.” This direction is obtained by using the policy $\tilde{\mu}(r^k)$ to generate randomly cost samples corresponding to a set of sample parameter values that are concentrated around $r^k$. The current set of sample parameters are then screened: some are accepted and the rest are rejected, based on a cost improvement criterion. Then $r^{k+1}$ is determined as a “central point” or as the “sample mean” in the set of accepted sample parameters, some more samples are generated randomly around $r^{k+1}$, and the process is repeated; see Fig. 5.7.4. Thus successive iterates $r^k$ are “central points” of successively better groups of samples, so in some broad
sense, the random sample generation process is guided by cost improvement. This idea is shared with evolutionary programming; see e.g., the books [Bac96], [DeJ06].

The cross-entropy method is very simple to implement, does not suffer from the fragility of gradient-based optimization, does not involve randomized policies, and relies on some supportive theory. In fact the method does not require the calculation of gradients, and in fact it does not require differentiability of the cost function. Moreover, it does not need a model to compute the required costs of different policies; a simulator is sufficient. Like all random search methods, its convergence rate guarantees are limited, and its success depends on domain-specific insight and the skilled use of heuristics. However, the method has gained a favorable reputation through some impressive successes. In particular, it was used for learning a high-scoring strategy in the game of tetris; see Szita and Lorinz [SzL06], and Thiery and Scherrer [ThS09]. The parametrization in policy space in these papers has been derived through a feature-based parametrization in value space; cf. Example 5.7.3. There have also been reports of domain-specific successes with related random search methods; see Salimans et al. [SHC17]. We refer to the end-of-chapter literature for details and examples of implementation.

5.7.2 Expert Supervised Training

According to the second approach for approximation in policy space, we choose the parameter $r$ by “training” on a large number of sample state-control pairs $(i^s, u^s)$, $s = 1, \ldots, q$, such that for each $s$, $u^s$ is a “good” control at state $i^s$. This can be done for example by solving the least squares problem\footnote{It is implicitly assumed here (and in similar situations later) that the controls are members of a Euclidean space so that the distance between two controls can be measured by their normed difference.}

$$\min_r \sum_{s=1}^{q} \|u^s - \hat{\mu}(i^s, r)\|^2$$

(possibly with added regularization). In particular, we may determine $u^s$ by a human or a software “expert” that can choose “near-optimal” controls at given states, so $\hat{\mu}$ is trained to match the behavior of the expert. We have also discussed this approach in Section 2.1.3, in the context of finite horizon problems. In the context of artificial intelligence, it comes within the framework of supervised learning methods.\footnote{Tesauro [Tes89a], [Tes89b] constructed a backgammon player, trained by a neural network and a supervised learning approach (called “comparison learning”), which used examples from human expert play (he was the expert who provided the training samples). However, his subsequent TD-based algorithm [Tes92], [Tes94], [Tes95], performed substantially better, and his rollout-based...}
Another possibility is to suitably select a large number of sample states $i^s$, $s = 1, \ldots, q$, and generate the controls $u^s$, $s = 1, \ldots, q$, through a one-step lookahead minimization of the form

$$u^s = \arg \min_{u \in U(i^s)} \sum_{j=1}^{n} p_{ij}(u)(g(i^s, u, j) + \alpha \tilde{J}(j)), \quad (5.71)$$

where $\tilde{J}$ is a suitable one-step lookahead function (multistep lookahead can also be used). Similarly, once a parametric $Q$-factor approximation architecture $\tilde{Q}(i, u, r)$ is chosen, we can select a large number of sample states $i^s$, $s = 1, \ldots, q$, and then compute the controls $u^s$, $s = 1, \ldots, q$, through the one-step lookahead minimization

$$u^s = \arg \min_{u \in U(i^s)} \tilde{Q}(i^s, u, r). \quad (5.72)$$

In this case, we will be collecting sample state-control pairs $(i^s, u^s)$, $s = 1, \ldots, q$, using approximation in value space through Eq. (5.71) or Eq. (5.72), and then applying approximation in policy space through Eq. (5.70).

Note that once the sample state-control pairs $(i^s, u^s)$, $s = 1, \ldots, q$, have been collected, an alternative to solving the least squares problem (5.70) is to use interpolation (rather than parametric approximation). By this we mean to specify for each $i / \in \{i_1, \ldots, i_s\}$ a probability distribution $\{\phi_{i1}, \ldots, \phi_{is}\}$, and to use the policy $\tilde{\mu}$ defined by

$$\tilde{\mu}(i) = \sum_{s=1}^{q} \phi_{is} u^s, \quad i = 1, \ldots, n. \quad (5.73)$$

In general, this requires that the control constraint set is a convex subset of a Euclidean space so that the interpolated controls (5.73) are feasible. This is not necessary if all the interpolation probabilities $\phi_{is}$ are either 0 or 1. Interpolation approaches are central to the aggregation methodology of Chapter 6, and will be discussed in greater detail there.

Of course in the expert training approach we cannot expect to obtain a policy that performs better than the expert with which it is trained, in the case of Eq. (5.70), or the one-step lookahead policy that is based on the approximation $\tilde{J}$ or $\tilde{Q}$, in the case of Eq. (5.71) or Eq. (5.72), respectively. However, a major advantage is that once the parametrized policy is obtained, the on-line implementation of the policy is fast and does not involve extensive calculations such as minimizations of the form (5.71). This advantage is generally shared by schemes that are based on approximation in policy space.

algorithm [TeG96] performed even better. The Deepchess program by David, Netanyahu, and Wolf [DNW16] provides another example of an expert-based supervised training approach.
5.7.3 Approximate Policy Iteration, Rollout, and Approximation in Policy Space

In this section we revisit approximate PI, but with a view towards combining it with rollout and approximation in policy space. In particular, we describe how approximation in policy space offers an alternative PI implementation, namely *approximate the generated policies directly instead of approximating their cost functions or Q-factors.*

Looking back to Sections 5.3.2 and 5.3.3, we see that the approximate PI methods given there use approximate policy evaluation (approximation in value space to represent the cost function or Q-factors of the current policy) followed by fairly exact policy improvement through one-step or multistep lookahead. By contrast, the methods described in this section use fairly exact policy evaluation through the use of rollout for a sample set of states, followed by approximate policy improvement [representation of the improved (or rollout) policy using a policy architecture]. The idea is to view the PI algorithm as a *perpetual rollout process,* which uses one out of a parametrized collection of base policies, and occasionally “improves” the base policy using the rollout results and approximation in policy space.

As an example, let us consider a PI algorithm where at the typical iteration we have a policy \( \mu \), which we use as the base policy for generating by rollout many state-control sample pairs \((i^*, u^*), s = 1, \ldots, q\) (cf. the rollout algorithm of Section 5.1.2). We then obtain an “improved” policy \( \tilde{\mu}(i, \tau) \), using an approximation architecture, where the parameter \( \tau \) is
obtained from the least squares/regression minimization

$$\tau \in \arg \min_r \sum_{s=1}^{q} ||u^s - \tilde{\mu}(i^s,r)||^2$$ (5.74)

(possibly with added regularization); see Fig. 5.7.5. The “improved” policy $\tilde{\mu}(i,\tau)$ is then used as a base policy to generate samples of the corresponding rollout policy, which is then approximated in policy space, etc. This is similar to the expert training approach of Section 5.7.2 [cf. Eq. (5.70)]; we just use the rollout policy as the “expert” and emulate its decisions using sampling and supervised learning.†

Among notable characteristics of the scheme just described, we mention the substantial computation required to generate the rollout policy sample pairs $(i^s, u^s), s = 1, \ldots, q$, particularly for a stochastic problem (this computation can of course take advantage of parallelization). Thus the PI process must be performed off-line. However, once the final policy is obtained, it can be used on-line as a base policy for generation of rollout controls, thus allowing on-line replanning. Moreover, the scheme shares the common advantage of policy space approximation: it yields in the end a policy that can be easily implemented on-line, without the need for one-step or multistep lookahead minimization.

There are many variations of the approximate PI scheme just described. Basically, all the variants of rollout (multistep lookahead, rollout truncation, and terminal cost function approximation) are applicable. Moreover, optimistic variants of PI can be used, whereby only a small number of samples $q$ are generated between changes in the parameter vector and the attendant change of base policy. To implement such an optimistic variant, one may use an incremental gradient or Newton method (cf. Section 3.1.3) to solve the regression minimization (5.74) while adding new terms to the least squares sum as new samples $(i^s, u^s)$ with $u^s$ being the control obtained with the current policy (the one that corresponds to the current value of $r$). Naturally, as in all approximate PI methods, exploration is an important issue to address with a judicious choice of the sampled states $i^s$.

Here is an example of an extreme case of optimistic incremental gradient-type algorithm, where the parameter $r$ is updated after each sample of state control pair $(i^s, u^s)$ is obtained. The algorithm parallels the SARSA Q-learning algorithm given in Section 5.4.1.

At the start of iteration $k$, we have the current parameter vector $r^k$, and the corresponding policy $\mu^k = \tilde{\mu}(\cdot, r^k)$. Then:

1. We select a state $i^k$ (with due regard to exploration).

† A simpler alternative to parametric approximation and the least squares minimization (5.74) is the interpolation scheme (5.73).
(2) We compute the rollout control $u^k$ at $i^k$, using $\mu^k = \tilde{\mu}(\cdot, r^k)$ as base policy, i.e.,

$$u^k \in \arg\min_{u \in U(i^k)} \sum_{j=1}^{n} p_{i^k j}(u)(g(i^k, u, j) + \alpha J_{\mu^k}(j)).$$

(3) We update the parameter vector via

$$r^{k+1} = r^k - \gamma^k \nabla \tilde{\mu}(i^k, r^k)(\tilde{\mu}(i^k, r^k) - u^k),$$

where $\gamma^k$ is a positive stepsize, and $\nabla \tilde{\mu}(i^k, r^k)$ denotes the gradient matrix of $\tilde{\mu}(i^k, \cdot)$ evaluated at the current parameter vector $r^k$.

As in the case of SARSA, there are also less optimistic variants of the preceding algorithm, whereby several states and their rollout controls are computed before updating the parameter vector $r$. Overall the schemes of this section appear to be promising for problems that admit a natural policy parametrization (cf. Example 5.7.1). However, they have not been tested on a challenging problem so far.

5.8 NOTES AND SOURCES

Section 5.1: The performance bound of Props. 5.1.1 and 5.1.3 for multistep lookahead, rollout, and terminal cost function approximation are sharper versions of earlier results for one step lookahead, terminal cost function approximation, but no rollout; see Prop. 6.1.1 in the author’s DP textbook [Ber17] (and earlier editions), as well as [Ber18a], Section 2.2.

Section 5.2: Fitted VI algorithms have been used for finite horizon problems since the early days of DP. They are conceptually simple and easily implementable, and they are used widely for approximation of either optimal costs or Q-factors (see e.g., the papers by Gordon [Gor95], Longstaff and Schwartz [LoS01], Ormoneit and Sen [OrS02], Ernst, Geurts, and Wenkel [EGW06], Antos, Munos, and Szepesvari [AMS07], and Munos and Szepesvari [MuS08]).

Section 5.3: The approximate PI method of Section 5.3.3 has been proposed by Fern, Yoon, and Givan [FYG06], and variants have also been discussed and analyzed by several other authors. The method (with some variations) has been used to train a tetris playing computer program that performs impressively better than programs that are based on other variants of approximate policy iteration, and various other methods; see Scherrer [Sch13], Scherrer et al. [SGG15], and Gabillon, Ghavamzadeh, and Scherrer [GGS13], who also provide an analysis of the method.

Section 5.4: Q-learning was first proposed by Watkins [Wat89], and had a major impact in the development of the field. A rigorous convergence proof...
of Q-learning was given by Tsitsiklis [Tsi94], in a more general framework that combined several ideas from stochastic approximation theory and the theory of distributed asynchronous computation. This proof covered discounted problems, and SSP problems where all policies are proper. It also covered SSP problems with improper policies, assuming that the Q-learning iterates are either nonnegative or bounded. Convergence for SSP problems without the nonnegativity or the boundedness assumption was shown by Yu and Bertsekas [YuB13b]. Optimistic asynchronous versions of PI based on Q-learning, which have solid convergence properties, are given by Bertsekas and Yu [BeY10], [BeY12], [YuB13a]. The distinctive feature of these methods is that the policy evaluation process aims towards the solution of an optimal stopping problem rather than towards solution of the linear system of Bellman equations associated with the policy; this is needed for the convergence proof, to avoid the pathological behavior first identified by Williams and Baird [WiB93], and noted earlier.

The advantage updating idea, which was noted in the context of finite horizon problems in Section 3.3, can be readily extended to infinite horizon problems. In this context, it was proposed by Baird [Bai93], [Bai94]; see [BeT96], Section 6.6. A related variant of approximate policy iteration and Q-learning, called differential training, has been proposed by the author in [Ber97b] (see also Weaver and Baxter [WeB99]).

Section 5.5: Projected equations underlie Galerkin methods, which have a long history in scientific computation. They are widely used for many types of problems, including the approximate solution of large linear systems arising from discretization of partial differential and integral equations. The connection of approximate policy evaluation based on projected equations with Galerkin methods was first discussed by Yu and Bertsekas [YuB10], and Bertsekas [Ber11c], and is potentially important as it may lead to cross-fertilization of ideas. However, the Monte Carlo simulation ideas that are central in approximate DP differentiate the projected equation methods of the present chapter from the Galerkin methodology. On the other hand, Galerkin methods apply to a wide range of problems, far beyond DP, and the simulation-based ideas of approximate DP can consequently be extended to apply more broadly (see [Ber12], Section 7.3).

Temporal difference methods originated in RL, where they are viewed as a means to encode the error in predicting future costs of a given policy, which is associated and an approximation architecture. They were introduced in the works of Samuel [Sam59], [Sam67] on a checkers-playing program. The work by Sutton [Sut88], following earlier work by Barto, Sutton, and Anderson [BSA83], formalized temporal differences and proposed the TD(\(\lambda\)) method. This was a major development and motivated a lot of research in simulation-based DP, particularly following an impressive early success with the backgammon playing program of Tesauro [Tes92], [Tre94].
The three methods TD(\(\lambda\)), LSTD(\(\lambda\)), and LSPE(\(\lambda\)) are discussed in detail in the journal and textbook RL literature. For a discussion that extends our presentation of Section 4.9, see Chapters 6 and 7 of the book [Ber12].

The convergence of TD(\(\lambda\)) was proved by Tsitsiklis and Van Roy [TsV97], with extensions in [TsV99a] and [TsV99b]. The author’s papers [Ber16b], [Ber18d] describe the connection of TD and proximal methods, a central methodology in convex optimization. In particular, TD(\(\lambda\)) is shown to be a stochastic version of the proximal algorithm for solving linear systems of equations, and extensions of TD(\(\lambda\)) for solving nonlinear systems of equations are described.

The LSTD(\(\lambda\)) algorithm was first proposed by Bradtke and Barto [BrB96] for \(\lambda = 0\), and was extended for \(\lambda > 0\) later by Boyan [Boy02]. Convergence analyses of LSTD(\(\lambda\)) under assumptions of increasing generality were given by Nedić and Bertsekas [NeB03], Bertsekas and Yu [BeY09], and Yu [Yu12].

The LSPE(\(\lambda\)) algorithm was first proposed by Bertsekas and Ioffe [BeI96] under the name \(\lambda\)-policy iteration, and it was used to train a tetris playing program using the feature-based linear architecture described in Example 3.1.3. The motivation for \(\lambda\)-policy iteration was to provide a better alternative to TD(\(\lambda\))-based policy iteration, which failed within the tetris context. LSPE(\(\lambda\)) was also given in the book [BeT96], Section 2.3.1, with subsequent contributions by Nedic, Borkar, Yu, Scherrer, and the author [NeB03], [BBN04], [YuB07], [BeY09], [YuB09], [Ber11b], [Yu12], [Sch13], [Ber18a].

In our discussion here, we did not go much into the implementation details of TD(\(\lambda\)), LSTD(\(\lambda\)), and LSPE(\(\lambda\)); see the approximate DP/RL textbooks cited earlier, and the paper by Bertsekas and Yu [BeY09], which adapts the TD methodology to the solution of large systems of linear equations.

Section 5.6: The linear programming approach for exact infinite horizon DP was proposed by D’Epenoux [D’Ep60]. Approximation methods using basis functions and linear programming were suggested without analysis by Schweitzer and Seidman [ScS85], and have been further developed by de Farias and Van Roy [DFV03], [DFV04], [DeF04]. For more recent work, see Desai, Farias, and Moallemi [DFM12]. For a challenging application to pricing of network services, see Paschalidis and Tsitsiklis [PaT00]. Approximate linear programming, while not discussed at length in this book, is a promising approach that deserves further attention.

Section 5.7: Policy gradient methods have a long history. For a detailed discussion and references we refer to the book by Sutton and Barto [SuB18], the monograph by Deisenroth, Neumann, and Peters [DNP11], and the surveys by Peters and Schaal [PeS08], and Grondman et. al. [GBL12]. The use of the log-likelihood trick in the context of simulation-based DP is
generally attributed to Williams [Wil92].

There are several related early works on search along randomly chosen directions (Rastrigin [Ras63], Matyas [Mat65], Aleksandrov, Sysoyev, and Shemeneva [ASS68], Rubinstein [Rub69]); see also Spall [Spa92], [Spa03], Duchi, Jordan, Wainwright, and Wibisono [DJW12], [DJW15], and Nesterov and Spokoiny [NeS17], for more modern related works. For early works on simulation-based policy gradient schemes for various DP problems, see Glynn [Gly87], [Gly90], L’Ecuyer [L’Ec91], Fu and Hu [FuH94], Jaakkola, Singh, and Jordan [JSJ95], Cao and Chen [CaC97], Cao and Wan [CaW98]. The more recent works of Baxter and Bartlett [BaB01], Konda and Tsitsiklis [KoT99], [KoT03], Marbach and Tsitsiklis [MaT01], [MaT03], and Sutton et. al. [SMS99] have been influential.

The challenge in the successful implementation of policy gradient methods is twofold: the difficulties with slow convergence and local minima that are inherent in gradient optimization, and the detrimental effects of simulation noise. Much work has been directed towards variations that address these difficulties, including the use of scaling based on the so-called natural gradient, suggested in the RL context by Kakade [Kak02]. For an interesting discussion of the connections between PID control, model-free approximation in policy space, and policy gradient methods, we refer to the paper and blog of B. Recht [Ber18a], [Rec18b].

For textbook discussions of the cross-entropy method, see Rubinstein and Kroese [RuK04], [RuK17], and Busoniu et. al. [BBD10], and for surveys see de Boer et. al. [BKM05], and Kroese et. al. [KRC13]. The method was proposed for policy search by Mannor, Rubinstein, and Gat [MRG03]. It was applied with impressive success in the game of tetris by Szita and Lorinz [SzL06], and Thiery and Scherrer [ThS09].

The expert training methods of Section 5.7.2 are similar to the comparison training method discussed in Section 2.4.3, which was proposed by Tesauro [Tes89a], [Tes89b], [Tes01]. Methods that use learning from data generated by an expert are often referred in the literature by the names imitation learning and apprenticeship learning.

The PI/rollout methods of Section 5.7.3 are new to the author’s knowledge. Their main strength is that they rely on the reliability of the rollout method, which has been demonstrated by many practical studies. However, approximation in policy space of the rollout policies is subject to the same difficult exploration issues that we have discussed in the context of other approximate PI methods.

5.9 APPENDIX: MATHEMATICAL ANALYSIS

In this appendix we provide proofs of the mathematical results stated in this chapter. We also prove some supplementary results that are described in the chapter without formal statement.
We will make heavy use of the DP operators $T$ and $T_\mu$, particularly for the discounted problem:

$$(T J)(i) = \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u) \left( g(i, u, j) + \alpha J(j) \right), \quad i = 1, \ldots, n,$$

$$(T_\mu J)(i) = \sum_{i=1}^{n} p_{ij}(\mu(i)) \left( g(i, \mu(i), j) + \alpha J(j) \right), \quad i = 1, \ldots, n.$$

A key property is the monotonicity of these operators, i.e.,

$$T J \geq T J', \quad T_\mu J \geq T_\mu J', \quad \text{for all } J \text{ and } J' \text{ with } J \geq J'.$$

Also for the discounted problem, we have the “constant shift” property, which states that if the functions $J$ is increased uniformly by a constant $c$, then the functions $T J$ and $T_\mu J$ are also increased uniformly by the constant $\alpha c$.

### 5.9.1 Performance Bounds for One-Step Lookahead

We first prove the basic performance bounds for $\ell$-step lookahead schemes and discounted problems.

**Proposition 5.1.1: (Limited Lookahead Performance Bounds)**

(a) Let $\bar{\mu}$ be the $\ell$-step lookahead policy corresponding to $\bar{J}$. Then

$$\|J_{\bar{\mu}} - J^*\| \leq \frac{2\alpha \ell}{1 - \alpha} \|\bar{J} - J^*\|, \quad (5.75)$$

where $\| \cdot \|$ denotes the maximum norm $\|J\| = \max_{i=1, \ldots, n} |J(i)|$.

(b) Let $\bar{\mu}$ be the one-step lookahead policy obtained by minimization in the equation

$$\bar{J}(i) = \min_{u \in \overline{U}(i)} \sum_{j=1}^{n} p_{ij}(u) \left( g(i, u, j) + \alpha \bar{J}(j) \right), \quad i = 1, \ldots, n,$$

where $\overline{U}(i) \subset U(i)$ for all $i = 1, \ldots, n$. Assume that for some constant $c$, we have

$$\bar{J}(i) \leq \tilde{J}(i) + c, \quad i = 1, \ldots, n. \quad (5.77)$$

Then

$$J_{\bar{\mu}}(i) \leq \tilde{J}(i) + \frac{c}{1 - \alpha}, \quad i = 1, \ldots, n. \quad (5.78)$$
Proof: (a) The line of proof of this part is essentially to show Eq. (5.75) for the case of one-step lookahead ($\ell = 1$), and then generalize the argument to the case $\ell > 1$ by replacing $\tilde{J}$ with $T^{\ell-1}\tilde{J}$. In the course of the proof, we will use the contraction property of $T$ and $T_\mu$ (cf. Prop. 4.3.5).

We first prove a preliminary inequality. Using the triangle inequality, we write for every $k$,

$$
\|T^k_\mu J^* - J^*\| \leq \sum_{m=1}^k \|T^m_\mu J^* - T^{m-1}_\mu J^*\| \leq \sum_{m=1}^k \alpha^{m-1}\|T^m_\mu J^* - J^*\|.
$$

By taking the limit as $k \to \infty$ and using the fact $T^k_\mu \tilde{J} \to \tilde{J}$, we obtain

$$
\|J_\mu - J^*\| \leq \frac{1}{1-\alpha}\|T^k_\mu J^* - J^*\|. \tag{5.79}
$$

Denote now $\hat{J} = T^{\ell-1}\tilde{J}$. By using the triangle inequality and the fact $T^\ell_\mu J = T\hat{J}$ (which follows from the definition of $\hat{\mu}$), the rightmost expression of Eq. (5.79) is estimated as follows:

$$
\|T^\ell_\mu J^* - J^*\| \leq \|T^\ell_\mu J^* - T^\ell_\mu \hat{J}\| + \|T^\ell_\mu \hat{J} - T\hat{J}\| + \|T\hat{J} - J^*\|
= \|T^\ell_\mu J^* - T^\ell_\mu \hat{J}\| + \|T\hat{J} - T^\ell J^*\|
\leq 2\alpha\|\hat{J} - J^*\|
= 2\alpha\|T^{\ell-1}\tilde{J} - T^{\ell-1}J^*\|
\leq 2\alpha\ell\|\tilde{J} - J^*\|.
$$

By combining the preceding two relations, we obtain Eq. (5.75).

(b) Let us denote by $e$ the unit vector whose components are all equal to 1. Then by the assumption (5.77), we have

$$
T^\ell_\mu \hat{J} \leq \hat{J} + ce.
$$

Applying $T^\ell_\mu$ to both sides of this relation, and using the monotonicity and constant shift property of $T^\ell_\mu$, we obtain

$$
T^{\ell+1}_\mu \hat{J} \leq T^\ell_\mu \hat{J} + ace.
$$

Continuing similarly, we have

$$
T^{k+1}_\mu \hat{J} \leq T^k_\mu \hat{J} + \alpha^k ce, \quad k = 0, 1, \ldots.
$$

Adding these relations, we obtain

$$
T^{k+1}_\mu \hat{J} \leq \hat{J} + (1 + \alpha + \cdots + \alpha^k)ce, \quad k = 0, 1, \ldots.
$$

Taking the limit as $k \to \infty$, and using the fact $T^{k+1}_\mu \hat{J} \to J_\mu$, we obtain the desired inequality (5.78). \textbf{Q.E.D.}
5.9.2 Performance Bounds for Rollout

We next show the basic cost improvement property of rollout.

**Proposition 5.1.2: (Cost Improvement by Rollout)** Let $\tilde{\mu}$ be the rollout policy obtained by the one-step lookahead minimization

$$\min_{u \in \mathcal{U}(i)} \sum_{j=1}^{n} p_{ij}(u) \left( g(i, u, j) + \alpha J_\mu(j) \right),$$

where $\mu$ is a base policy [cf. Eq. (5.76) with $\bar{J} = J_\mu$] and we assume that $\mu(i) \in \overline{U}(i) \subset U(i)$ for all $i = 1, \ldots, n$. Then $J_{\tilde{\mu}} \leq J_\mu$.

**Proof:** Let us denote

$$\hat{J}(i) = \min_{u \in \mathcal{U}(i)} \sum_{j=1}^{n} p_{ij}(u) \left( g(i, u, j) + \alpha J_\mu(j) \right).$$

We have for all $i = 1, \ldots, n$,

$$\hat{J}(i) \leq \sum_{j=1}^{n} p_{ij}(u) \left( g(i, \mu(i), j) + \alpha J_\mu(j) \right) = J_\mu(i),$$

where the equality on the right holds by Bellman’s equation. Hence the hypothesis of Prop. 5.1.1(b) holds with $c = 0$, and the result follows from the bound (5.78). Q.E.D.

We finally show the following performance bound for the truncated rollout algorithm with cost function approximation.

**Proposition 5.1.3: (Performance Bound of Truncated Rollout with Terminal Cost Function Approximation)** Let $\ell$ and $m$ be positive integers, let $\mu$ be a policy, and let $\tilde{J}$ be a function of the state. Consider a truncated rollout scheme consisting of $\ell$-step lookahead, followed by rollout with a policy $\mu$ for $m$ steps, and a terminal cost function approximation $\tilde{J}$ at the end of the $m$ steps. Let $\tilde{\mu}$ be the policy generated by this scheme.

(a) We have

$$\| J_{\tilde{\mu}} - J^* \| \leq \frac{2\alpha}{1 - \alpha} \| T_\mu^m \hat{J} - J^* \|,$$

where $T_\mu$ is the DP operator of Eq. (5.4), and $\| \cdot \|$ denotes the maximum norm $\| J \| = \max_{i=1, \ldots, n} |J(i)|$. 


(b) Assume that for some constant \( c \), \( \tilde{J} \) and \( \mu \) satisfy the condition
\[
\hat{J}(i) = \sum_{j=1}^{n} p_{ij}(\mu(i)) \left( g(i, \mu(i), j) + \alpha \tilde{J}(j) \right) \leq \tilde{J}(i) + c, \tag{5.80}
\]
for all \( i = 1, \ldots, n \). Then
\[
J_{\mu}(i) \leq \tilde{J}(i) + \frac{c}{1 - \alpha}, \quad i = 1, \ldots, n. \tag{5.81}
\]

(c) Assume that for some constant \( c \), \( \tilde{J} \) and \( \mu \) satisfy
\[
\| \tilde{J} - J_{\mu} \| \leq \frac{c}{1 + \alpha}. \tag{5.82}
\]
Then
\[
J_{\mu}(i) \leq J_{\mu}(i) + \frac{2c}{1 - \alpha^2}.
\]

**Proof:** (a) This is simply Prop. 5.1.1(a) adapted to the truncated rollout scheme \([\tilde{J} \text{ is replaced in Eq. (5.75) by } T_{\mu}^m \tilde{J}]\).

(b) We first prove the result for the case where \( c = 0 \). Then the condition (5.80) can be written as \( \tilde{J} \geq T_{\mu} \tilde{J} \), from which by using the monotonicity of \( T \) and \( T_{\mu} \), we have
\[
\tilde{J} \geq T_{\mu}^m \tilde{J} \geq TT_{\mu}^m \tilde{J} \geq T^{\ell - 1} T_{\mu}^m \tilde{J} \geq T_{\mu}^m T^{\ell - 1} T_{\mu}^m \tilde{J}, \tag{5.83}
\]
so that
\[
\tilde{J} \geq T^{\ell - 1} T_{\mu}^m \tilde{J} \geq T_{\mu}^m T^{\ell - 1} T_{\mu}^m \tilde{J}.
\]
This relation and the monotonicity of \( T_{\mu} \), imply that \( \{T_{\mu}^k T^{\ell - 1} T_{\mu}^m \tilde{J}\} \) is monotonically nonincreasing as \( k \) increases, and is bounded above by \( \tilde{J} \). Since by the convergence property of VI, the sequence converges to \( J_{\mu} \) as \( k \to \infty \), the result follows.

To prove the result for general \( c \), we introduce the function \( J' \) given by
\[
J'(i) = \tilde{J}(i) + \frac{c}{1 - \alpha}, \quad i = 1, \ldots, n.
\]
Then the condition (5.80) can be written in terms of \( J' \) as
\[
\tilde{J}(i) = \sum_{j=1}^{n} p_{ij}(\mu(i)) \left( g(i, \mu(i), j) + \alpha J'(j) - \frac{\alpha c}{1 - \alpha} \right) \leq J'(i) - \frac{c}{1 - \alpha} + c,
\]
or equivalently as
\[ \sum_{j=1}^{n} p_{ij} (\mu(i)) \left( g(i, \mu(i), j) + \alpha J'(j) \right) \leq J'(i). \]

Since adding a constant to the components of \( \tilde{J} \) does not change \( \tilde{\mu} \), we can replace \( \tilde{J} \) with \( J' \), without changing \( \tilde{\mu} \). Then by using the version of the result already proved, we have \( J_{\tilde{\mu}} \leq J' \), which is equivalent to the desired relation (5.81).

(c) We prove this part as a special case of part (b). The hypothesis (5.82) implies that Eq. (5.80) is satisfied since we have
\[
\tilde{J}(i) = \sum_{j=1}^{n} p_{ij} (\mu(i)) \left( g(i, \mu(i), j) + \alpha \tilde{J}(j) \right) \\
\leq \sum_{j=1}^{n} p_{ij} (\mu(i)) \left( g(i, \mu(i), j) + \alpha J_{\tilde{\mu}}(j) \right) + \frac{\alpha c}{1 + \alpha} \\
= J_{\mu}(i) + \frac{\alpha c}{1 + \alpha} \\
\leq \tilde{J}(i) + \frac{c}{1 + \alpha} + \frac{\alpha c}{1 + \alpha} \\
= \tilde{J}(i) + c.
\]

Part (b) then shows that
\[
J_{\tilde{\mu}}(i) \leq \tilde{J}(i) + \frac{c}{1 - \alpha} \leq J_{\mu}(i) + \frac{c}{1 + \alpha} + \frac{c}{1 - \alpha} = J_{\mu}(i) + \frac{2c}{1 - \alpha^2}
\]
for all \( i \). Q.E.D.

The preceding proof allows a relaxation of the condition (5.80). For the relation (5.83) to hold it is sufficient that \( \tilde{J} \) and \( \mu \) satisfy the condition
\[
\tilde{J} \geq T_{\mu}^{m} \tilde{J} \geq T_{\mu}^{m+1} \tilde{J},
\]
or the even weaker condition
\[
\tilde{J} \geq T_{\mu}^{m} \tilde{J} \geq TT_{\mu}^{m} \tilde{J}.
\]
There is also an extension of the preceding condition for the case where \( m = 0 \), i.e., there is no rollout. It takes the form
\[
\tilde{J} \geq T \tilde{J},
\]
and it implies the bound \( J_{\tilde{\mu}} \leq \tilde{J} \). The proof is based on Eq. (5.83) where \( m \) is taken to be zero. In domain-specific contexts, the preceding conditions may be translated into meaningful results.
5.9.3 Performance Bounds for Approximate Policy Iteration

To prove the performance bound of Prop. 5.1.4, we focus on the discounted problem, and we make use of the contraction property of $T_\mu$:

$$
\|T_\mu J - T_\mu J'\| \leq \alpha \|J - J'\|, \quad \|TJ - TJ'\| \leq \alpha \|J - J'\|
$$

for all $J, J'$, and $\mu$, where $\|J\|$ is the maximum norm $\|J\| = \max_{i=1,\ldots,n} \{|J(i)|\}$.

We want to prove the following performance bound.

**Proposition 5.1.4: (Performance Bound for Approximate PI)**

Consider the discounted problem, and let $\{\mu^k\}$ be the sequence generated by the approximate PI algorithm defined by the approximate policy evaluation (5.12) and the approximate policy improvement (5.13). Then we have

$$
\limsup_{k \to \infty} \|J_{\mu^k} - J^*\| \leq \frac{\epsilon + 2\alpha\delta}{(1 - \alpha)^2}.
$$

The essence of the proof is contained in the following lemma, which quantifies the amount of approximate policy improvement at each iteration.

**Lemma 5.9.1:** Consider the discounted problem, and let $J, \tilde{\mu}$, and $\mu$ satisfy

$$
\|J - J_{\tilde{\mu}}\| \leq \delta, \quad \|T_{\tilde{\mu}}J - TJ\| \leq \epsilon,
$$

where $\delta$ and $\epsilon$ are some scalars. Then

$$
\|J_{\tilde{\mu}} - J^*\| \leq \alpha \|J_{\mu} - J^*\| + \frac{\epsilon + 2\alpha\delta}{1 - \alpha}.
$$

**Proof:** The contraction property of $T$ and $T_{\tilde{\mu}}$ implies that

$$
\|T_{\tilde{\mu}}J_{\mu} - T_{\tilde{\mu}}J\| \leq \alpha \delta, \quad \|TJ - TJ_{\mu}\| \leq \alpha \delta,
$$

and hence

$$
T_{\tilde{\mu}}J_{\mu} \leq T_{\tilde{\mu}}J + \alpha \delta e, \quad TJ \leq TJ_{\mu} + \alpha \delta e,
$$

where $e$ is the unit vector, i.e., $e(i) = 1$ for all $i$. Using also Eq. (5.84), we have

$$
T_{\tilde{\mu}}J_{\mu} \leq T_{\tilde{\mu}}J + \alpha \delta e \leq TJ + (\epsilon + \alpha \delta)e \leq TJ_{\mu} + (\epsilon + 2\alpha)e.
$$

(5.86)
Combining this inequality with $T J_\mu \leq T_\mu J_\mu = J_\mu$, we obtain

$$T_\tilde{\mu} J_\mu \leq J_\mu + (\epsilon + 2\alpha\delta)e. \quad (5.87)$$

We will show that this relation implies that

$$J_\tilde{\mu} \leq J_\mu + \frac{\epsilon + 2\alpha\delta}{1 - \alpha}e. \quad (5.88)$$

Indeed, by applying $T_\tilde{\mu}$ to both sides of Eq. (5.87), we obtain

$$T_2^2 J_\mu \leq T_\tilde{\mu} J_\mu + \alpha(\epsilon + 2\alpha\delta)e \leq J_\mu + (1 + \alpha)(\epsilon + 2\alpha\delta)e.$$  

Applying $T_\tilde{\mu}$ again to both sides of this relation, and continuing similarly, we have for all $k$,

$$T_k^k J_\mu \leq J_\mu + (1 + \alpha + \cdots + \alpha^{k-1})(\epsilon + 2\alpha\delta)e.$$  

By taking the limit as $k \to \infty$, and by using the VI convergence property $T_\tilde{\mu} J_\mu \to J_\tilde{\mu}$, we obtain Eq. (5.88).

Using now the contraction property of $T_\tilde{\mu}$ and Eq. (5.88), we have

$$J_\tilde{\mu} = T_\tilde{\mu} J_\tilde{\mu} = T_\tilde{\mu} J_\mu + (T_\tilde{\mu} J_\tilde{\mu} - T_\tilde{\mu} J_\mu) \leq T_\tilde{\mu} J_\mu + \frac{\alpha(\epsilon + 2\alpha\delta)}{1 - \alpha}e.$$  

Subtracting $J^*$ from both sides, we obtain

$$J_\tilde{\mu} - J^* \leq T_\tilde{\mu} J_\mu - J^* + \frac{\alpha(\epsilon + 2\alpha\delta)}{1 - \alpha}e. \quad (5.89)$$

Also from the contraction property of $T$,

$$T J_\mu - J^* = T J_\mu - T J^* \leq \alpha\|J_\mu - J^*\|e$$

which, in conjunction with Eq. (5.86), yields

$$T_\tilde{\mu} J_\mu - J^* \leq T J_\mu - J^* + (\epsilon + 2\alpha\delta)e \leq \alpha\|J_\mu - J^*\|e + (\epsilon + 2\alpha\delta)e.$$  

Combining this relation with Eq. (5.89), we obtain

$$J_\tilde{\mu} - J^* \leq \alpha\|J_\mu - J^*\|e + \frac{\alpha(\epsilon + 2\alpha\delta)}{1 - \alpha}e + (\epsilon + 2\alpha\delta)e = \alpha\|J_\mu - J^*\|e + \frac{\epsilon + 2\alpha\delta}{1 - \alpha}e,$$

which is equivalent to the desired relation (5.85).  \textbf{Q.E.D.}

\textbf{Proof of Prop. 5.1.4:} Applying Lemma 5.9.1, we have

$$\|J_{\mu^{k+1}} - J^*\| \leq \alpha\|J_{\mu^k} - J^*\| + \frac{\epsilon + 2\alpha\delta}{1 - \alpha}.$$
which by taking the lim sup of both sides as $k \to \infty$ yields the desired result.

Q.E.D.

We next prove the performance bound for approximate PI, assuming that the generated policy sequence is convergent. For this proof we use the triangle inequality, which holds for any norm $\| \cdot \|$, 

$$\|J + J'\| \leq \|J\| + \|J'\|, \quad \text{for all } J, J'.$$

**Proposition 5.1.5: (Performance Bound for Approximate PI when Policies Converge)** Let $\tilde{\mu}$ be a policy generated by the approximate PI algorithm under conditions (5.12), (5.13), and (5.14). Then we have

$$\max_{i=1, \ldots, n} |J_{\tilde{\mu}}(i) - J^*(i)| \leq \frac{\epsilon + 2\alpha \delta}{1 - \alpha}.$$ 

**Proof:** Let $\tilde{J}$ be the cost vector obtained by approximate policy evaluation of $\tilde{\mu}$. Then in view of Eqs. (5.12), (5.13), we have

$$\|\tilde{J} - J_{\tilde{\mu}}\| \leq \delta, \quad \|T_{\tilde{\mu}} \tilde{J} - T \tilde{J}\| \leq \epsilon.$$ 

From this relation, the fact $J_{\tilde{\mu}} = T_{\tilde{\mu}} J_{\tilde{\mu}}$, and the triangle inequality, we have

$$\|T J_{\tilde{\mu}} - J_{\tilde{\mu}}\| \leq \|T J_{\tilde{\mu}} - T \tilde{J}\| + \|T \tilde{J} - T_{\tilde{\mu}} \tilde{J}\| + \|T_{\tilde{\mu}} \tilde{J} - J_{\tilde{\mu}}\|$$

$$= \|T J_{\tilde{\mu}} - T \tilde{J}\| + \|T \tilde{J} - T_{\tilde{\mu}} \tilde{J}\| + \|T_{\tilde{\mu}} \tilde{J} - T_{\tilde{\mu}} J_{\tilde{\mu}}\|$$

$$\leq \alpha \|J_{\tilde{\mu}} - \tilde{J}\| + \epsilon + \alpha \|\tilde{J} - J_{\tilde{\mu}}\|$$

$$\leq \epsilon + 2\alpha \delta. \quad (5.90)$$

For every $k$, by using repeatedly the triangle inequality and the contraction property of $T$, we have

$$\|T^k J_{\tilde{\mu}} - J_{\tilde{\mu}}\| \leq \sum_{\ell=1}^{k} \|T^\ell J_{\tilde{\mu}} - T^{\ell-1} J_{\tilde{\mu}}\| \leq \sum_{\ell=1}^{k} \alpha^{\ell-1} \|T J_{\tilde{\mu}} - J_{\tilde{\mu}}\|,$$

and by taking the limit as $k \to \infty$,

$$\|J^* - J_{\tilde{\mu}}\| \leq \frac{1}{1 - \alpha} \|T J_{\tilde{\mu}} - J_{\tilde{\mu}}\|.$$

Combining this relation with Eq. (5.90), we obtain the desired performance bound. Q.E.D.