Chapter 4

Infinite Horizon Reinforcement Learning

DRAFT

This is Chapter 4 of the draft textbook “Reinforcement Learning and Optimal Control.” The chapter represents “work in progress,” and it will be periodically updated. It more than likely contains errors (hopefully not serious ones). Furthermore, its references to the literature are incomplete. Your comments and suggestions to the author at dimitrib@mit.edu are welcome.

The date of last revision is given below. (A “revision” is any version of the chapter that involves the addition or the deletion of at least one paragraph or mathematically significant equation.)

April 29, 2019
4

Infinite Horizon Reinforcement Learning

Contents

4.1. An Overview of Infinite Horizon Problems . . . . . . . p. 3
4.2. Stochastic Shortest Path Problems . . . . . . . . . . . p. 6
4.3. Discounted Problems . . . . . . . . . . . . . . . . p. 16
4.4. Exact and Approximate Value Iteration . . . . . . . p. 21
4.5. Policy Iteration . . . . . . . . . . . . . . . . . . p. 25
  4.5.1. Exact Policy Iteration . . . . . . . . . . . . . p. 26
  4.5.2. Optimistic and Multistep Lookahead Policy Iteration p. 30
  4.5.3. Policy Iteration for Q-factors . . . . . . . . . . p. 32
4.6. Approximation in Value Space - Performance Bounds . . . p. 34
  4.6.1. Limited Lookahead Performance Bounds . . . . . p. 36
  4.6.2. Rollout . . . . . . . . . . . . . . . . . . p. 39
  4.6.3. Approximate Policy Iteration . . . . . . . . . . p. 43
4.7. Simulation-Based Policy Iteration with Parametric . . . . .
  Approximation . . . . . . . . . . . . . . . . . . p. 46
  4.7.1. Self-Learning and Actor-Critic Systems . . . . . p. 46
  4.7.2. A Model-Based Variant . . . . . . . . . . . . p. 47
  4.7.3. A Model-Free Variant . . . . . . . . . . . . p. 50
  4.7.4. Implementation Issues of Parametric Policy Iteration p. 52
4.8. Q-Learning . . . . . . . . . . . . . . . . . . . p. 54
          Approximation - SARSA and DQN . . . . . . . . p. 56
4.9. Additional Methods - Temporal Differences . . . . . . . p. 57
4.10. Exact and Approximate Linear Programming . . . . . . . p. 68
### Infinite Horizon Reinforcement Learning

**Chap. 4**

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.11. Approximation in Policy Space</td>
<td>71</td>
</tr>
<tr>
<td>4.11.1. Training by Cost Optimization - Policy Gradient, Cross Entropy, and Random Search Methods</td>
<td>75</td>
</tr>
<tr>
<td>4.11.2. Expert Supervised Training</td>
<td>85</td>
</tr>
<tr>
<td>4.11.3. Approximate Policy Iteration, Rollout, and Approximation in Policy Space</td>
<td>86</td>
</tr>
<tr>
<td>4.12. Notes and Sources</td>
<td>90</td>
</tr>
<tr>
<td>4.13. Appendix: Mathematical Analysis</td>
<td>95</td>
</tr>
<tr>
<td>4.13.2. Proofs for Discounted Problems</td>
<td>101</td>
</tr>
<tr>
<td>4.13.3. Convergence of Exact and Optimistic Policy Iteration</td>
<td>102</td>
</tr>
<tr>
<td>4.13.4. Performance Bounds for One-Step Lookahead</td>
<td>104</td>
</tr>
<tr>
<td>4.13.5. Performance Bounds for Rollout</td>
<td>106</td>
</tr>
<tr>
<td>4.13.6. Performance Bounds for Approximate Policy Iteration</td>
<td>109</td>
</tr>
</tbody>
</table>
In this chapter, we first provide an introduction to the theory of infinite horizon problems, and then consider the use of approximate DP/RL methods for suboptimal solution. Infinite horizon problems differ from their finite horizon counterparts in two main respects:

(a) The number of stages is infinite.

(b) The system is stationary, i.e., the system equation, the cost per stage, and the random disturbance statistics do not change from one stage to the next.

The assumption of an infinite number of stages is never satisfied in practice, but is a reasonable approximation for problems involving a finite but very large number of stages. The assumption of stationarity is often satisfied in practice, and in other cases it approximates well a situation where the system parameters vary relatively slowly with time.

Infinite horizon problems give rise to elegant and insightful analysis, and their optimal policies are often simpler than their finite horizon counterparts. For example, optimal policies are typically stationary, i.e., the optimal rule for choosing controls does not change from one stage to the next.

On the other hand, infinite horizon problems generally require a more sophisticated mathematical treatment. Our discussion will be limited to relatively simple finite-state problems (the theory of problems with an infinite state space is considerably more intricate; see the books [BeS78], [Ber12], [Ber18a]). Still some theoretical results will be needed for this chapter. They will be explained intuitively to the extent possible, and their mathematical proofs will be provided in the end-of-chapter appendix.

**4.1 AN OVERVIEW OF INFINITE HORIZON PROBLEMS**

We will focus on two types of infinite horizon problems, where we aim to minimize the total cost over an infinite number of stages, given by

\[
J_\pi(x_0) = \lim_{N \to \infty} \mathbb{E}_{\mu_k} \left\{ \sum_{k=0}^{N-1} \alpha^k g(x_k, \mu_k(x_k), w_k) \right\};
\]

see Fig. 4.1.1. Here, \( J_\pi(x_0) \) denotes the cost associated with an initial state \( x_0 \) and a policy \( \pi = \{\mu_0, \mu_1, \ldots\} \), and \( \alpha \) is a positive scalar. The meaning of \( \alpha < 1 \) is that future costs matter to us less than the same costs incurred at the present time.

Thus the infinite horizon costs of a policy is the limit of its finite horizon costs as the horizon tends to infinity. (We assume that the limit exists for the moment, and address the issue later.) The two types of problems, considered in Sections 4.2 and 4.3, respectively, are:
Infinite Horizon Reinforcement Learning  Chap. 4

Figure 4.1.1 Illustration of an infinite horizon problem. The system and cost per stage are stationary, except for the use of a discount factor $\alpha$. If $\alpha = 1$, there is typically a special cost-free termination state that we aim to reach.

(a) *Stochastic shortest path problems* (SSP for short). Here, $\alpha = 1$ but there is a special cost-free termination state; once the system reaches that state it remains there at no further cost. We will assume a problem structure such that termination is inevitable. Thus the horizon is in effect finite, but its length is random and may be affected by the policy being used.

(b) *Discounted problems*. Here, $\alpha < 1$ and there need not be a termination state. However, we will see that a discounted problem can be readily converted to an SSP problem. This can be done by introducing an artificial termination state to which the system moves with probability $1 - \alpha$ at every stage, thus making termination inevitable. As a result, our algorithms and analysis for SSP problems can be easily adapted to discounted problems.

**A Preview of Infinite Horizon Theory**

There are several analytical and computational issues regarding our infinite horizon problems. Many of them revolve around the relation between the optimal cost-to-go function $J^*$ of the infinite horizon problem and the optimal cost-to-go functions of the corresponding $N$-stage problems.

In particular, consider the SSP case and let $J_N(x)$ denote the optimal cost of the problem involving $N$ stages, initial state $x$, cost per stage $g(x, u, w)$, and zero terminal cost. This cost is generated after $N$ iterations of the DP algorithm

$$J_{k+1}(x) = \min_{u \in U(x)} E \left\{ g(x, u, w) + J_k(f(x, u, w)) \right\}, \quad k = 0, 1, \ldots, (4.1)$$

starting from the initial condition $J_0(x) = 0$ for all $x$.† The algorithm (4.1) is known as the value iteration algorithm (VI for short). Since the infinite horizon DP algorithm (4.1) is just the finite horizon DP algorithm of Chapter 1. However, we have reversed the time indexing to suit our purposes. Thus the index of the cost functions produced by the algorithm is incremented with each iteration, and not decremented as in the case of finite horizon.

† This is just the finite horizon DP algorithm of Chapter 1. However, we have reversed the time indexing to suit our purposes. Thus the index of the cost functions produced by the algorithm is incremented with each iteration, and not decremented as in the case of finite horizon.
An Overview of Infinite Horizon Problems

The horizon cost of a given policy is, by definition, the limit of the corresponding $N$-stage costs as $N \to \infty$, it is natural to speculate that:

1. The optimal infinite horizon cost is the limit of the corresponding $N$-stage optimal costs as $N \to \infty$; i.e.,

$$J^*(x) = \lim_{N \to \infty} J_N(x) \quad (4.2)$$

for all states $x$.

2. The following equation should hold for all states $x$,

$$J^*(x) = \min_{u \in U(x)} E \left\{ g(x, u, w) + J^*(f(x, u, w)) \right\}. \quad (4.3)$$

This is obtained by taking the limit as $N \to \infty$ in the VI algorithm (4.1) using Eq. (4.2). Equation (4.3) is really a system of equations (one equation per state $x$), which has as solution the costs-to-go of all the states. It can also be viewed as a functional equation for the optimal cost function $J^*$, and it is called Bellman’s equation.

3. If $\mu(x)$ attains the minimum in the right-hand side of the Bellman equation (4.3) for each $x$, then the policy $\{\mu, \mu, \ldots\}$ should be optimal. This type of policy is called stationary. Intuitively, optimal policies can be found within this class of policies, since the future optimization problem when starting at a given state looks the same regardless of the time when we start.

All three of the preceding results hold for SSP problems under our assumptions, as we will state later in Section 4.2 and prove in the appendix to this chapter. They also hold for discounted problems in suitably modified form that incorporates the discount factor. In fact the algorithms and analysis of this chapter are quite similar for SSP and discounted problems, to the point where we may discuss a particular method for one of the two problems with the understanding that its application to the other problem can be straightforwardly adapted.

Transition Probability Notation for Finite-State Infinite Horizon Problems

Throughout this chapter we assume a finite-state discrete-time dynamic system, and we will use a special transition probability notation that is suitable for such a system. We generally denote states by the symbol $i$ and successor states by the symbol $j$. We will assume that there are $n$ states (in addition to the termination state for SSP problems). These states are denoted $1, \ldots, n$, and the termination state is denoted $t$. The control $u$ is constrained to take values in a given finite constraint set $U(i)$, which may
depend on the current state \( i \). The use of a control \( u \) at state \( i \) specifies the transition probability \( p_{ij}(u) \) to the next state \( j \), at a cost \( g(i, u, j) \).\(^\dagger\)

Given an admissible policy \( \pi = \{\mu_0, \mu_1, \ldots\} \) (one with \( \mu_k(i) \in U(i) \) for all \( i \) and \( k \)) and an initial state \( i_0 \), the system becomes a Markov chain whose generated trajectory under \( \pi \), denoted \( \{i_0, i_1, \ldots\} \), has a well-defined probability distribution. The total expected cost associated with an initial state \( i \) is

\[
J_\pi(i) = \lim_{N \to \infty} E \left\{ \sum_{k=0}^{N-1} \alpha^k g(i_k, \mu_k(i_k), i_{k+1}) \middle| i_0 = i, \pi \right\},
\]

where \( \alpha \) is either 1 (for SSP problems) or less than 1 for discounted problems. The expected value is taken with respect to the joint distribution of the states \( i_1, i_2, \ldots \), conditioned on \( i_0 = i \) and the use of \( \pi \). The optimal cost from state \( i \), i.e., the minimum of \( J_\pi(i) \) over all policies \( \pi \), is denoted by \( J^*(i) \).

The cost function of a stationary policy \( \pi = \{\mu, \mu, \ldots\} \) is denoted by \( J_\mu(i) \). For brevity, we refer to \( \pi \) as the stationary policy \( \mu \). We say that \( \mu \) is optimal if

\[
J_\mu(i) = J^*(i) = \min_{\pi} J_\pi(i), \text{ for all states } i.
\]

As noted earlier, under our assumptions, we will show that there will always exist an optimal policy, which is stationary.

### 4.2 STOCHASTIC SHORTEST PATH PROBLEMS

In the SSP problem we assume that there is no discounting (\( \alpha = 1 \)), and that there is a special cost-free termination state \( t \). Once the system reaches that state, it remains there at no further cost, i.e.,

\[
p_{tt}(u) = 1, \quad g(t, u, t) = 0, \text{ for all } u \in U(t).
\]

We denote by \( 1, \ldots, n \) the states other than the termination state \( t \); see Fig. 4.2.1.

With this notation, the Bellman equation (4.3) and the VI algorithm (4.1) take the following form.

\(^\dagger\) To convert from the transition probability format to the system equation format used in the preceding chapters, we can simply use the system equation

\[
x_{k+1} = w_k,
\]

where \( w_k \) is the disturbance that takes values according to the transition probabilities \( p_{x_kw_k}(u_k) \).
Sec. 4.2 Stochastic Shortest Path Problems

Figure 4.2.1 The transition graph of an SSP problem. There are $n$ states, plus the termination state $t$, with transition probabilities $p_{ij}(u)$. The termination state is cost-free and absorbing.

Bellman Equation and Value Iteration for SSP Problems:

For all $i = 1, \ldots, n$, we have

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

(4.4)

For all $i = 1, \ldots, n$, and any initial conditions $J_0(1), \ldots, J_0(n)$, the VI algorithm generates the sequence $\{J_k\}$ according to

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

(4.5)

The right-hand bracketed expression in the Bellman equation (4.4) represents an expected value, which is similar to the expectation we have seen in earlier DP expressions. It is the sum of three terms:

(a) The contribution

$$p_{it}(u)g(i, u, t)$$

to the expected cost of the current stage of the terminating $i$-to-$t$ transition.

(b) The contribution

$$\sum_{j=1}^{n} p_{ij}(u)g(i, u, j)$$
to the expected cost of the current stage of the nonterminating \( i \)-to-\( j \) transitions.

(c) The optimal expected cost-to-go

\[
\sum_{j=1}^{n} p_{ij}(u) J^*(j)
\]

starting from the next state \( j \) [if the next state is \( t \), the corresponding optimal cost is \( J^*(t) \), which is zero, so it does not appear in the sum].

Note that the deterministic shortest path problem of Section 1.3.1 is obtained as the special case of the SSP problem where for each state-control pair \((i, u)\), the transition probability \( p_{ij}(u) \) is equal to 1 for a unique state \( j \) that depends on \((i, u)\). Moreover, any deterministic or stochastic finite-state, finite horizon problem with a termination state (cf. Section 1.3.3) can be converted to an SSP problem. In particular, the reader may verify that the finite-state \( N \)-step horizon problem of Chapter 1 can be obtained as a special case of an SSP problem by viewing as state the pair \((x_{N}, k)\) and lumping all pairs \((x_{N}, N)\) into a termination state \( t \).

We are interested in problems where reaching the termination state \( t \) is inevitable. Thus, the essence of the problem is to reach \( t \) with minimum expected cost. Throughout this chapter, when discussing SSP problems, we will make the following assumption, which will be shown to guarantee eventual termination under all policies.†

† The main analytical and algorithmic results for SSP problems are valid under more general conditions, which involve the notion of a proper policy (see the end-of-chapter references). In particular, a stationary policy is called proper if starting from every state, it is guaranteed to eventually reach the destination. The policy is called improper if it is not proper.

It can be shown that Assumption 4.2.1 is equivalent to the seemingly weaker assumption that all stationary policies are proper. However, the subsequent four propositions can also be shown under the genuinely weaker assumption that there exists at least one proper policy, and furthermore, every improper policy is “bad” in the sense that it results in infinite expected cost from at least one initial state (see [BeT89], [BeT91], or [Ber12], Chapter 3). These assumptions, when specialized to deterministic shortest path problems, are similar to the assumptions of Section 1.3.1. They imply that there is at least one path to the destination from every starting state and that all cycles have positive cost. In the absence of these assumptions, the Bellman equation may have no solution or an infinite number of solutions (see [Ber18a], Section 3.1.1 for discussion of a simple example, which in addition to \( t \), involves a single state 1 at which we can either stay at cost \( a \) or move to \( t \) at cost \( b \); anomalies occur when \( a = 0 \) and when \( a < 0 \)).
Assumption 4.2.1: (Termination is Inevitable Under All Policies) There exists an integer $m$ such that regardless of the policy used and the initial state, there is positive probability that the termination state will be reached after no more than $m$ stages; i.e., for all admissible policies $\pi$ we have

$$\rho_\pi = \max_{i=1, \ldots, n} P\{x_m \neq t \mid x_0 = i, \pi\} < 1.$$ 

Let $\rho$ be the maximum probability of not reaching $t$, over all starting states and policies:

$$\rho = \max_\pi \rho_\pi.$$ 

Note that $\rho_\pi$ depends only on the first $m$ components of the policy $\pi$. Furthermore, since the number of controls available at each state is finite, the number of distinct $m$-stage policies is also finite. It follows that there can be only a finite number of distinct values of $\rho_\pi$, so that

$$\rho < 1.$$ 

This implies that the probability of not reaching $t$ over a finite horizon diminishes to 0 as the horizon becomes longer, regardless of the starting state and policy used.

To see this, note that for any $\pi$ and any initial state $i$

$$P\{x_{2m} \neq t \mid x_0 = i, \pi\} = P\{x_{2m} \neq t \mid x_m \neq t, x_0 = i, \pi\}$$

$$\cdot P\{x_m \neq t \mid x_0 = i, \pi\} \leq \rho^2.$$ 

More generally, for each $\pi$, the probability of not reaching the termination state after $km$ stages diminishes like $\rho^k$ regardless of the initial state, i.e.,

$$P\{x_{km} \neq t \mid x_0 = i, \pi\} \leq \rho^k, \quad i = 1, \ldots, n. \quad (4.6)$$

This fact implies that the limit defining the associated total cost vector $J_\pi$ exists and is finite, and is central in the proof of the following results (given in the appendix to this chapter).

We now describe the main theoretical results for SSP problems; the proofs are given in the appendix to this chapter. Our first result is that the infinite horizon version of the DP algorithm, which is VI [cf. Eq. (4.1)], converges to the optimal cost function $J^*$. The optimal cost $J^*(t)$ starting from $t$ is of course 0, so it is just neglected where appropriate in the subsequent analysis. Generally, $J^*$ is obtained in the limit, after an infinite
number of iterations. However, there are important cases where convergence is obtained in finitely many iterations (see [Ber12], Chapter 3).

**Proposition 4.2.1: (Convergence of VI)** The optimal cost $J^*(i)$ is finite for each $i$. Moreover, given any initial conditions $J_0(1), \ldots, J_0(n)$, the sequence $\{J_k(i)\}$ generated by the VI algorithm

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

converges to $J^*(i)$ for each $i = 1, \ldots, n$.

Our next result is that the limiting form of the DP equation, Bellman’s equation, has $J^*$ as its unique solution.

**Proposition 4.2.2: (Bellman’s Equation)** The optimal cost function $J^* = (J^*(1), \ldots, J^*(n))$ satisfies for all $i = 1, \ldots, n$, the equation

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

and is the unique solution of this equation.

Our next result expresses that by restricting attention to a single policy $\mu$, we obtain a Bellman equation specific to $\mu$, which has $J_\mu$ as its unique solution.

**Proposition 4.2.3: (VI and Bellman’s Equation for Policies)** For any stationary policy $\mu$, the corresponding cost function $J_\mu = (J_\mu(1), \ldots, J_\mu(n))$ satisfies for all $i = 1, \ldots, n$ the equation

$$J_\mu(i) = p_{it}(\mu(i))g(i, \mu(i), t) + \sum_{j=1}^{n} p_{ij}(\mu(i))(g(i, \mu(i), j) + J_\mu(j)),$$
Sec. 4.2 Stochastic Shortest Path Problems

and is the unique solution of this equation. Furthermore, given any initial conditions \( J_0(1), \ldots, J_0(n) \), the sequence \( \{ J_k(i) \} \) generated by the VI algorithm that is specific to \( \mu \),

\[
J_{k+1}(i) = p_{id}(\mu(i))g(i, \mu(i), t) + \sum_{j=1}^{n} p_{ij}(\mu(i)) \left( g(i, \mu(i), j) + J_k(j) \right),
\]

converges to the cost \( J_\mu(i) \) for each \( i \).

Our final result provides a necessary and sufficient condition for optimality of a stationary policy.

**Proposition 4.2.4: (Optimality Condition)** A stationary policy \( \mu \) is optimal if and only if for every state \( i \), \( \mu(i) \) attains the minimum in the Bellman equation (4.7).

We provide an example illustrating Bellman’s equation.

**Example 4.2.1 (Maximum Expected Time to Termination)**

The case where

\[
g(i, u, j) = -1, \quad \text{for all } i, u \in U(i), \text{ and } j,
\]

corresponds to a problem where the objective is to terminate as late as possible on the average, while the opposite of the optimal cost, \( -J^*(i) \), is the maximum expected time to termination starting from state \( i \). Under our assumptions, the optimal costs \( J^*(i) \) uniquely solve Bellman’s equation, which has the form

\[
J^*(i) = \min_{u \in U(i)} \left[ -1 + \sum_{j=1}^{n} p_{ij}(u)J^*(j) \right], \quad i = 1, \ldots, n.
\]

In the special case of a single policy \( \mu \), where there is only one control at each state, \( -J_\mu(i) \) represents the expected time to reach \( t \) starting from \( i \). This is known as the mean first passage time from \( i \) to \( t \), and is given as the unique solution of the corresponding Bellman equation

\[
J_\mu(i) = -1 + \sum_{j=1}^{n} p_{ij}(\mu(i))J_\mu(j), \quad i = 1, \ldots, n.
\]

We will now provide an insightful mathematical result about SSP problems, which is proved in the appendix with the aid of the preceding
example. To this end let us introduce for any vector $J = (J(1), \ldots, J(n))$, the notation
\[
(TJ)(i) = \min_{u \in U(i)} \left[ p_{it}(u) g(i, u, t) + \sum_{j=1}^{n} p_{ij}(u) \left( g(i, u, j) + J(j) \right) \right],
\]
(4.8)
for all $i = 1, \ldots, n$, and
\[
(T\mu J)(i) = p_{it}(\mu(i)) g(i, \mu(i), t) + \sum_{j=1}^{n} p_{ij}(\mu(i)) \left( g(i, \mu(i), j) + J(j) \right),
\]
(4.9)
for all policies $\mu$ and states $i = 1, \ldots, n$. Here $T$ and $T\mu$ are the Bellman operators that map the vector $J$ into the vectors
\[
TJ = \left( (TJ)(1), \ldots, (TJ)(n) \right), \quad T\mu J = \left( (T\mu J)(1), \ldots, (T\mu J)(n) \right),
\]
respectively. Bellman’s equations can be written in terms of these operators as the fixed point equations $J^* = TJ^*$ and $J^*_\mu = T^*_\mu J^*_\mu$.

The next proposition states that $T$ and $T\mu$ are contraction mappings, so the unique fixed point property of this mapping follows from general mathematical results about contraction mappings (see e.g., [Ber12], [Ber18a]). Moreover the contraction property provides a convergence rate estimate for VI, and is the basis for further analysis of exact and approximate methods for SSP problems (see the author’s monograph [Ber18a] for a theoretical development of DP, which is based on fixed point theory and an abstract operator viewpoint).

**Proposition 4.2.5: (Contraction Property of the Bellman Operator)** The Bellman operator $T$ of Eqs. (4.8) is a contraction mapping with respect to some weighted norm
\[
\|J\| = \max_{i=1,\ldots,n} \frac{|J(i)|}{v(i)},
\]
defined by some vector $v = (v(1), \ldots, v(n))$ with positive components. In other words, there exists a positive scalar $\rho < 1$ such that for any two $n$-dimensional vectors $J$ and $J'$, we have
\[
\|TJ - TJ'\| \leq \rho \|J - J'\|.
\]
The same is true for the Bellman operator $T\mu$ (4.9) of any policy $\mu$.

The proof of the proposition, given in the appendix, shows that the weights $v(i)$ and the modulus of contraction $\rho$ are related to the maximum
expected number of steps $-m^*(i)$ to reach $t$ from $i$ (cf. Example 4.2.1). In particular, we have

$$v(i) = -m^*(i), \quad \rho = \max_{i=1,\ldots,n} \frac{v(i) - 1}{v(i)}.$$ 

Note that the weight vector $v$ and the corresponding weighted norm may be different for $T$ and for $T_\mu$, since the maximum expected number of steps to reach $t$ from $i$ may be different under $\mu$ and under an optimal policy.

Among others, the preceding contraction property provides a convergence rate estimate for VI, namely that the generated sequence $\{J_k\}$ satisfies

$$\|J_k - J^*\| \leq \rho^k \|J_0 - J^*\|.$$ 

This follows from the fact that $J_k$ and $J^*$ can be viewed as the results of the $k$-fold application of $T$ to the vectors $J_0$ and $J^*$, respectively.

**Bellman Equation and Value Iteration for Q-Factors**

The results just given have counterparts involving Q-factors. The optimal Q-factors are defined for all $i = 1, \ldots, n$, and $u \in U(i)$ by

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

As in the finite horizon case, $Q^*(i, u)$ can be interpreted as the cost of starting at $i$, using $u$ for the first stage, and using an optimal policy afterwards. Once $Q^*$ is computed by some method, an optimal policy $\mu^*$ can be obtained from the minimization

$$\mu^*(i) \in \arg\min_{u \in U(i)} Q^*(i, u), \quad i = 1, \ldots, n.$$ 

Similarly, if approximately optimal Q-factors $\tilde{Q}(i, u)$ are obtained by some method (model-based or model-free), a suboptimal policy $\tilde{\mu}$ can be obtained from the minimization

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

Our basic results relating Bellman’s equation and the VI algorithm are stated as follows.

**Bellman Equation and Value Iteration for Q-Factors and SSP Problems:**

For all $i = 1, \ldots, n$, and $u \in U(i)$ we have

$$Q^*(i, u) = p_{it}(u)g(i, u, t) + \sum_{j=1}^{n} p_{ij}(u) \left(g(i, u, j) + \min_{v \in U(j)} Q^*(j, v)\right).$$
State-Control Pairs

\[ (i, u) \quad g(i, u, j) \quad p_{ij}(u) \quad j \quad \text{States} \]

\[ (j, v) \quad \text{Control } v \quad \text{Cost } = 0 \]

\[ \text{States} \]

Figure 4.2.2 States, transition probabilities, and stage costs corresponding to a modified SSP problem, which yields the optimal Q-factors as well as the optimal costs. The states of this problem are the pairs \((i, u), u \in U(i)\), the original problem states \(i = 1, \ldots, n\), and the termination state \(t\). A control \(v \in U(j)\) is available only at the original system states \(j\), leading to the pair \((j, v)\) at cost 0. The transition from a pair \((i, u)\) leads to \(j\) with probability \(p_{ij}(u)\) and cost 0. The Bellman equation for this modified problem is

\[
Q^*(i, u) = p_{it}(u)g(i, u, t) + \sum_{j=1}^{n} p_{ij}(u) \left( g(i, u, j) + \min_{v \in U(j)} Q^*(j, v) \right),
\]

for the states \((i, u), u \in U(i)\), and

\[
J^*(j) = \min_{v \in U(j)} Q^*(j, v),
\]

for the states \(j = 1, \ldots, n\). Note that a policy \(\mu\) for this problem leads from a state \(j\) to the state \((j, \mu(j))\), so in any system trajectory, only pairs of the form \((j, \mu(j))\) are visited after the first transition.

For all \(i = 1, \ldots, n\), and \(u \in U(i)\), and any initial conditions \(Q_0(i, u)\), the VI algorithm generates the sequence \(\{Q_k\}\) according to

\[
Q_{k+1}(i, u) = p_{it}(u)g(i, u, t) + \sum_{j=1}^{n} p_{ij}(u) \left( g(i, u, j) + \min_{v \in U(j)} Q_k(j, v) \right).
\]

Actually, the optimal Q-factors \(Q^*(i, u)\) can be viewed as optimal state costs associated with a modified SSP problem, which involves a new state for each pair \((i, u)\) with transition probabilities \(p_{ij}(u)\) to the states \(j = 1, \ldots, n, t\); see Fig. 4.2.2. Then the preceding Bellman equation for the optimal Q-factors, together with the Bellman equation (4.7) for the optimal costs \(J^*(j)\), can be viewed as the Bellman equation for the modified SSP
Temporal Differences and Cost Shaping

Bellman’s equation can be written in an alternative form, which involves the differential
\[
\dot{J} = J^* - V,
\]
where \( V = (V(1), \ldots, V(n)) \) is any \( n \)-dimensional vector and \( V(t) = 0 \). In particular, by subtracting \( V(i) \) from both sides of the Bellman equation (4.7), and adding and subtracting \( V(j) \) within the right-hand side summation, we obtain

\[
\dot{J}(i) = p_{ti}(u)\dot{g}(i, u, t) + \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u)(\dot{g}(i, u, j) + \dot{J}(j)),
\]
for all \( i = 1, \ldots, n \), where

\[
\dot{g}(i, u, j) = \begin{cases} 
  g(i, u, j) + V(j) - V(i) & \text{if } i, j = 1, \ldots, n, \\
  g(i, u, t) - V(i) & \text{if } i = 1, \ldots, n, \ j = t.
\end{cases}
\]
We refer to Eq. (4.10) as the variational form of Bellman’s equation, and to the modified cost per stage \( \dot{g} \) as the temporal difference corresponding to \( V \). Temporal differences play a significant role in several algorithmic RL contexts; see Section 4.9, and the approximate DP/RL books referenced earlier.

Note that Eq. (4.10) is the Bellman equation for a cost-modified problem, where the cost per stage \( g \) has been replaced by the temporal difference \( \dot{g} \). Thus by applying Prop. 4.2.2 we have that \( \dot{J} = J^* - V \) is the unique solution of this equation, so that \( J^* \) can be obtained by solving either the original or the cost-modified version of the problem. Moreover, a policy \( \mu \), has cost function \( J_\mu = J_\mu - V \) in the cost-modified problem. It follows that the original and the cost-modified SSP problems are essentially equivalent, and the choice of \( V \) does not matter when exact DP methods are used to solve them. However, when approximate methods are used, different results may be obtained, which can be more favorable with an appropriate choice of \( V \).

In particular, we have the option to choose \( V \) and an approximation methodology that matches the differential \( \dot{J} = J^* - V \) better than it matches \( J^* \). For example, we may obtain \( V \) with some problem approximation scheme as a rough estimate of \( J^* \), and then use a different approximation in value space scheme, based on different principles, for the corresponding cost-modified problem. We refer to this as cost shaping (the name “reward shaping” is used in the RL literature, for problems involving reward maximization). While cost shaping does not change the optimal policies of the original DP problem, it may change significantly the suboptimal policies produced by approximate DP methods, such as the ones that we will discuss in this chapter and the next.
4.3 DISCOUNTED PROBLEMS

We now consider the discounted problem, where there is a discount factor \( \alpha < 1 \). Using our transition probability notation, the Bellman equation and the VI algorithm take the following form.

**Bellman Equation and Value Iteration for Discounted Problems:**

For all \( i = 1, \ldots, n \), we have

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

For all \( i = 1, \ldots, n \), and any initial conditions \( J_0(1), \ldots, J_0(n) \), the VI algorithm generates the sequence \( \{J_k\} \) according to

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

We will now show that the discounted problem can be converted to an SSP problem for which the analysis of the preceding section applies. To see this, let \( i = 1, \ldots, n \) be the states, and consider an associated SSP problem involving the states 1, \ldots, \( n \) plus an artificial termination state \( t \), with state transitions and costs obtained as follows: From a state \( i \neq t \), when control \( u \) is applied, the next state is \( j \) with probability \( \alpha p_{ij}(u) \) at a cost \( g(i, u, j) \), and \( t \) with probability \( 1 - \alpha \) at zero cost; see Fig. 4.3.1. Note that Assumption 4.2.1 of the preceding section is satisfied for this SSP problem, since \( t \) is reached with probability \( 1 - \alpha > 0 \) from every state in a single step.

Suppose now that we use the same policy in the discounted problem and in the associated SSP problem. Then, as long as termination has not occurred, the state evolution in the two problems is governed by the same transition probabilities. Furthermore, the expected cost of the \( k \)th stage of the associated shortest path problem is the expected value of \( g(i_k, \mu^k(i_k), i_{k+1}) \) multiplied by the probability that state \( t \) has not yet been reached, which is \( \alpha^k \). This is also the expected cost of the \( k \)th stage for the discounted problem. Thus the cost of any policy starting from a given state, is the same for the original discounted problem and for the associated SSP problem.

It follows that we can apply Props. 4.2.1-4.2.5 of the preceding section to the associated SSP problem and obtain corresponding results for
Sec. 4.3 Discounted Problems

Figure 4.3.1 Transition probabilities for an \(\alpha\)-discounted problem and its associated SSP problem. In the latter problem, the probability that the state is not \(t\) after \(k\) stages is \(\alpha^k\). The transition costs at the \(k\)th stage are \(g(i, u, j)\) for both problems, but they must be multiplied by \(\alpha^k\) because of discounting (in the discounted case) or because it is incurred with probability \(\alpha^k\) when termination has not yet been reached (in the SSP case).

the discounted problem, which properly incorporate the discount factor in accordance with the SSP-to-discounted equivalence just established.

**Proposition 4.3.1: (Convergence of VI)** The optimal cost \(J^*(i)\) is finite for each \(i\). Moreover, given any initial conditions \(J_0(1), \ldots, J_0(n)\), the sequence \(\{J_k(i)\}\) generated by the VI algorithm

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

converges to \(J^*(i)\) for each \(i\).

**Proposition 4.3.2: (Bellman’s Equation)** The optimal cost function

\[
J^* = (J^*(1), \ldots, J^*(n))
\]

satisfies for all \(i = 1, \ldots, n\), the equation

\[
J^*(i) = \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u)\left(g(i, u, j) + \alpha J^*(j)\right), \quad (4.12)
\]

and is the unique solution of this equation.
Proposition 4.3.3: (VI and Bellman’s Equation for Policies)
For any stationary policy \( \mu \), the corresponding cost function \( J_\mu = (J_\mu(1), \ldots, J_\mu(n)) \) is the unique solution of the equation
\[
J_\mu(i) = \sum_{j=1}^{n} p_{ij}(\mu(i)) \left( g(i, \mu(i), j) + \alpha J_\mu(j) \right), \quad i = 1, \ldots, n.
\]
Furthermore, given any initial conditions \( J_0(1), \ldots, J_0(n) \), the sequence \( \{J_k(i)\} \) generated by the VI algorithm
\[
J_{k+1}(i) = \sum_{j=1}^{n} p_{ij}(\mu(i)) \left( g(i, \mu(i), j) + \alpha J_k(j) \right), \quad i = 1, \ldots, n,
\]
converges to the cost \( J_\mu(i) \) for each \( i \).

Proposition 4.3.4: (Optimality Condition) A stationary policy \( \mu \) is optimal if and only if for every state \( i \), \( \mu(i) \) attains the minimum in the Bellman equation (4.12).

Bellman’s equation (4.12) has a familiar DP interpretation. At state \( i \), the optimal cost \( J^*(i) \) is the minimum over all controls of the sum of the expected current stage cost and the expected optimal cost of all future stages. The former cost is \( g(i, u, j) \). The latter cost is \( J^*(j) \), but since this cost starts accumulating after one stage, it is discounted by multiplication with \( \alpha \).

Similar to Prop. 4.2.5, there is a contraction mapping result and convergence rate estimate for value iteration. To this end we introduce the mappings
\[
(TJ)(i) = \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u)(g(i, u, j) + \alpha J(j)), \quad i = 1, \ldots, n, \quad (4.13)
\]
and
\[
(T_\mu J)(i) = \sum_{j=1}^{n} p_{ij}(\mu(i)) \left( g(i, \mu(i), j) + \alpha J(j) \right), \quad i = 1, \ldots, n, \quad (4.14)
\]
in analogy with their SSP counterparts of Eqs. (4.8) and (4.9). Similar to the SSP case, Bellman’s equations can be written in terms of these
operators as the fixed point equations $J^* = T J^*$, $J^*_\mu = T^*_\mu J^*$. The following contraction result is useful for the analysis of exact and approximate methods for discounted problems.

**Proposition 4.3.5: (Contraction Property of the Bellman Operator)** The Bellman operators $T$ and $T^*_\mu$ of Eqs. (4.13) and (4.14) are contraction mappings of modulus $\alpha$ with respect to the maximum norm

$$\| J \| = \max_{i=1,...,n} | J(i) |. \quad (4.15)$$

In particular, for any two $n$-dimensional vectors $J$ and $J'$, we have

$$\| T J - T J' \| \leq \alpha \| J - J' \|, \quad \| T^*_\mu J - T^*_\mu J' \| \leq \alpha \| J - J' \|.$$ 

Let us also mention that the cost shaping idea discussed for SSP problems, extends readily to discounted problems. In particular, the variational form of Bellman’s equation takes the form

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

for any given vector $V$, where

$$\hat{g}(i, u, j) = g(i, u, j) + \alpha V(j) - V(i), \quad i = 1, \ldots, n,$$

is the temporal difference corresponding to $V$; cf. Eqs. (4.10) and (4.11).

**Example 4.3.1 (Asset Selling)**

Consider of problem of selling an asset over an infinite number of periods. At each period an offer becomes available. We assume that offers at different periods are independent and that they can take $n$ values $v_1, \ldots, v_n$ with corresponding probabilities according to given probability $p(1), \ldots, p(n)$. Here, if accepted, the amount $i_k$ offered in period $k$, will be invested at a rate of interest $r$. By depreciating the sale amount to period 0 dollars, we view $(1+r)^{-k} i_k$ as the reward for selling the asset in period $k$ at a price $i_k$, where $r > 0$ is the rate of interest. Then we have a discounted reward problem with discount factor $\alpha = 1/(1+r)$. The analysis of the present section is applicable, and the optimal value function $J^*$ is the unique solution of Bellman’s equation

$$J^*(i) = \max \left[ v_i, \frac{1}{1+r} \sum_{j=1}^{n} p_{ij} J^*(j) \right].$$
Thus the optimal reward function is characterized by the critical number
\[ c = \frac{1}{1 + r} \sum_{j=1}^{n} p_j J^*(j). \]

An optimal policy is obtained by minimizing over the two controls. It is to sell if and only if the current offer \( i \) is greater than \( c \). The critical number \( c \) can be obtained by a simple form of VI (see [Ber17], Section 3.4).

A far more difficult version of the problem is one where the offers are correlated, so the offer at each stage may be viewed as an observation that provides information about future offers. A related difficult version of the problem is one where the probability distribution \( p = (p(1), \ldots, p(n)) \) of the offers is unknown, and is estimated as new offers are revealed. In both cases the problem can be formulated as a partial state information problem involving a belief state: the estimate of the distribution \( p \) given the past offers (suitable conditions are of course needed to ensure that the estimate of \( p \) can be in principle computed exactly or can be approximated as a practical matter). Some of the finite horizon approximation methods of Chapters 2 and 3 can be adapted to solve such a problem. However, an exact solution is practically impossible, since this would involve DP calculations over an infinite dimensional space of belief states.

**Bellman Equation and Value Iteration for Q-Factors**

As in the SSP case, the results just given have counterparts involving the optimal Q-factors, defined by
\[
Q^*(i, u) = \sum_{j=1}^{n} p_{ij}(u) \left( g(i, u, j) + \alpha J^*(j) \right), \quad i = 1, \ldots, n, \quad u \in U(i).
\]

They can be obtained from the corresponding SSP results, by viewing the discounted problem as a special case of the SSP problem. Once \( Q^* \) or an approximation \( \tilde{Q} \) is computed by some method (model-based or model-free), an optimal policy \( \mu^* \) or approximately optimal policy \( \tilde{\mu} \) can be obtained from the minimization
\[
\mu^*(i) \in \arg \min_{u \in U(i)} Q^*(i, u), \quad i = 1, \ldots, n,
\]
or the approximate version
\[
\tilde{\mu}(i) \in \arg \min_{u \in U(i)} \tilde{Q}(i, u), \quad i = 1, \ldots, n.
\]

Our basic results relating Bellman’s equation and the VI algorithm are stated as follows.
Bellman Equation and Value Iteration for Q-Factors and Discounted Problems:

For all $i = 1, \ldots, n$, and $u \in U(i)$ we have

$$Q^*(i, u) = \sum_{j=1}^{n} p_{ij}(u) \left( g(i, u, j) + \alpha \min_{v \in U(j)} Q^*(j, v) \right). \tag{4.16}$$

For all $i = 1, \ldots, n$, and $u \in U(i)$, and any initial conditions $Q_0(i, u)$, the VI algorithm generates the sequence $\{Q_k\}$ according to

$$Q_{k+1}(i, u) = \sum_{j=1}^{n} p_{ij}(u) \left( g(i, u, j) + \alpha \min_{v \in U(j)} Q_k(j, v) \right). \tag{4.17}$$

The VI algorithm (4.17) forms the basis for various $Q$-learning methods to be discussed later.

### 4.4 Exact and Approximate Value Iteration

We have already encountered the VI algorithm for SSP problems,

$$J_{k+1}(i) = \min_{u \in U(i)} \left[ p_t(u)g(i, u, t) + \sum_{j=1}^{n} p_{ij}(u)(g(i, u, j) + J_k(j)) \right], \tag{4.18}$$

and its discounted version

$$J_{k+1}(i) = \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u)(g(i, u, j) + \alpha J_k(j)). \tag{4.19}$$

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

Unfortunately, when the number of states is large, the iterations (4.18) and (4.19) 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. 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. (4.8) and (4.13)]. 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 Approximate Value Iteration

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

\[
\max_{i=1,...,n} |\tilde{J}_k(i) - J^*(i)|, \tag{4.20}
\]

and the policy error

\[
\max_{i=1,...,n} |\tilde{J}_{\tilde{\mu}_k}(i) - J^*(i)|, \tag{4.21}
\]

where the policy \(\tilde{\mu}_k\) is obtained from the minimization

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

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

\[
\max_{i=1,...,n} |\tilde{J}_{k+1}(i) - \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u)(g(i, u, j) + \alpha \tilde{J}_k(j))| \leq \delta. \tag{4.22}
\]

It is then possible to show that asymptotically, as \(k \to \infty\), the cost error (4.20) becomes less or equal to \(\delta/(1 - \alpha)\), while the policy error (4.21) 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 (4.22) 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 (4.22) cannot be maintained, and the approximate value iterates \(\tilde{J}_k\) can grow unbounded.

**Example 4.4.1 (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. 4.4.1. Thus the Bellman equation is

\[
J(1) = \alpha J(2), \quad J(2) = \alpha J(2),
\]
Sec. 4.4 Exact and Approximate Value Iteration

23

Figure 4.4.1 Illustration of the discounted problem of Example 4.4.1. 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.

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. 4.4.2:

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

\[T\tilde{J}_k = (\alpha \tilde{J}_k(2), \alpha \tilde{J}_k(2)) = (2\alpha r_k, 2\alpha r_k).\]

(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

\[r_{k+1} = \alpha \zeta r_k \quad \text{where} \quad \zeta = \frac{2(\xi_1 + 2\xi_2)}{\xi_1 + 4\xi_2} > 1.\]  \hspace{1cm} (4.23)

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. 4.4.2.

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
\[ \tilde{J}_k = (r_k, 2r_k) \]

**Figure 4.4.2** Illustration of Example 4.4.1. Iterates of approximate VI lie on the line \( \{(r, 2r) \mid r \in \mathbb{R}\} \). Given an iterate \( \tilde{J}_k = (r_k, 2r_k) \), the next exact VI iterate is

\[ (\alpha \tilde{J}_k(2), \alpha \tilde{J}_k(2)) = (2\alpha r_k, 2\alpha r_k). \]

The approximation of this iterate on the line \( \{(r, 2r) \mid r \in \mathbb{R}\} \) by least squares regression can be viewed as weighted projection onto the line, and depends on the weights \( (\xi_1, \xi_2) \). The range of weighted projections as the weights vary is shown in the figure. For the natural choice \( \xi_1 = \xi_2 = 1 \) and \( \alpha \) sufficiently close to 1, the new approximate VI iterate \( \tilde{J}_{k+1} \) is further away from \( J^* = (0, 0) \) than \( \tilde{J}_k \).

The difficulty here is that the mapping that consists of a VI followed by weighted projection onto the line \( \{(r, 2r) \mid r \in \mathbb{R}\} \) need not be a contraction.

No \( \delta \) such that the condition (4.22) 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 is 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

$$r_{k+1} \in \arg \min_r \sum_{s=1}^{q} (\tilde{J}(i^s, r) - \beta^s)^2,$$  \hspace{1em} (4.24)

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 (4.22) is satisfied and the instability illustrated with Example 4.4.1 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 (4.24). 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.

### 4.5 POLICY ITERATION

The major alternative to value iteration is policy iteration (PI for short). This algorithm starts with a stationary policy $\mu^0$, and generates iteratively a sequence of new policies $\mu^1, \mu^2, \ldots$. The algorithm has solid convergence guarantees when implemented in its exact form, as we will show shortly. When implemented in approximate form, as it is necessary when

† In the preceding Example 4.4.1, 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. (4.23) 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 $J^*$. 

the number of states is large, its performance guarantees are somewhat
more favorable than those of the approximate VI of the preceding section.

The closest analog of PI that we have encountered so far is the rollout
algorithm of Chapter 2. There we have started with some policy and pro-
duced an improved policy though a process of cost function evaluation and
one-step or multistep minimization. This idea is extended in the context
of PI, which consists of multiple successive policy evaluations and policy
improvements.

4.5.1 Exact Policy Iteration

Consider first the SSP problem. Here, each policy iteration consists of two
phases: policy evaluation and policy improvement; see Fig. 4.5.1.

**Exact Policy Iteration: SSP Problems**

Given the typical policy $\mu^k$:

**Policy evaluation** computes $J_{\mu^k}(i), i = 1, \ldots, n$, as the solution of the (linear) system of Bellman equations

$$J_{\mu^k}(i) = p_{it}(\mu^k(i))g(i, \mu^k(i), t) + \sum_{j=1}^{n} p_{ij}(\mu^k(i)) \left( g(i, \mu^k(i), j) + J_{\mu^k}(j) \right),$$

(cf. Prop. 4.2.3).


**Sec. 4.5 Policy Iteration**

Policy improvement then computes a new policy $\mu^{k+1}$ as

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

for $i = 1, \ldots, n$.

The process is repeated with $\mu^{k+1}$ used in place of $\mu^k$, unless we have

$$J_{\mu^{k+1}}(i) = J_{\mu^k}(i)
$$

for all $i$, in which case the algorithm terminates with the policy $\mu^k$.

The counterpart for discounted problems is as follows.

**Exact Policy Iteration: Discounted Problems**

Given the typical policy $\mu^k$:

**Policy evaluation** computes $J_{\mu^k}(i)$, $i = 1, \ldots, n$, as the solution of the (linear) system of Bellman equations

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

(cf. Prop. 4.2.3).

**Policy improvement** then computes a new policy $\mu^{k+1}$ as

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

The process is repeated with $\mu^{k+1}$ used in place of $\mu^k$, unless we have $J_{\mu^{k+1}}(i) = J_{\mu^k}(i)$ for all $i$, in which case the algorithm terminates with the policy $\mu^k$.

The following proposition, shown in the appendix, establishes the validity of PI, including finite termination with an optimal policy.
**Proposition 4.5.1: (Convergence of Exact PI)** For both the SSP and the discounted problems, the exact PI algorithm generates an improving sequence of policies, i.e.,

\[ J_{\mu^{k+1}}(i) \leq J_{\mu^k}(i), \quad \text{for all } i \text{ and } k, \quad (4.27) \]

and terminates with an optimal policy.

The proof of the policy improvement property (4.27) is quite intuitive and is worth summarizing for the discounted problem. Let \( \mu \) be a policy and \( \overline{\mu} \) be the policy obtained from \( \mu \) via a policy iteration. We want to show that \( J_{\overline{\mu}} \leq J_\mu \). To this end, let us denote by \( J_N \) the cost function of a policy that applies \( \overline{\mu} \) for the first \( N \) stages and applies \( \mu \) at every subsequent stage. We have the Bellman equation

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

which together with the policy improvement equation (4.26) imply that

\[ J_1(i) = \sum_{j=1}^{n} p_{ij}(\overline{\mu}(i)) \left( g(i, \overline{\mu}(i), j) + \alpha J_{\overline{\mu}}(j) \right) \leq J_{\mu}(i). \quad (4.28) \]

From the definition of \( J_2 \) and \( J_1 \) we have

\[ J_2(i) = \sum_{j=1}^{n} p_{ij}(\overline{\mu}(i)) \left( g(i, \overline{\mu}(i), j) + \alpha J_{\overline{\mu}}(j) \right), \quad (4.29) \]

so the preceding two relations imply that

\[ J_2(i) \leq J_1(i) \leq J_{\mu}(i), \quad \text{for all } i. \quad (4.30) \]

Continuing similarly, we obtain

\[ J_{N+1}(i) \leq J_N(i) \leq J_{\mu}(i), \quad \text{for all } i \text{ and } N. \quad (4.31) \]

Since \( J_N \rightarrow J_\overline{\mu} \) (cf. Prop. 4.3.2), it follows that \( J_\overline{\mu} \leq J_\mu \).

In practice, a lot of cost improvement is often obtained with the PI algorithm after the first few policies are generated. This may happen even if the number of iterations needed for termination is quite large. The following is an example where termination occurs after just two iterations.
Example 4.5.1 (Treasure Hunting)

A treasure hunter has obtained a lease to search a site that contains \( n \) treasures, and wants to find a searching policy that maximizes his expected gain over an infinite number of days. At each day, knowing the current number of treasures not yet found, he may decide to continue searching for more treasures at a cost \( c \) per day, or to permanently stop searching. If he searches on a day when there are \( i \) treasures on the site, he finds \( m \in [0, i] \) treasures with given probability \( p(m \mid i) \), where we assume that \( p(0 \mid i) < 1 \) for all \( i \geq 1 \), and that the expected number of treasures found,

\[
 r(i) = \sum_{m=0}^{i} m p(m \mid i),
\]

is monotonically increasing with \( i \). Each found treasure is worth 1 unit.

We formulate the problem as an SSP problem, with state equal to the number of treasures not yet found. The termination state is state 0, where the hunter stops searching. When the hunter decides to search at a state \( i \geq 1 \), the state moves to \( i - m \) with probability \( p(m \mid i) \). Here the inevitable termination Assumption 4.2.1 is satisfied, in view of our condition \( p(0 \mid i) < 1 \) for all \( i \). Bellman’s equation is

\[
 J^*(i) = \max \left\{ 0, r(i) - c + \sum_{m=0}^{i} p(m \mid i) J^*(i-m) \right\}, \quad i = 1, \ldots, n,
\]

with \( J^*(0) = 0 \).

Let us apply PI starting with the policy \( \mu^0 \) that never searches. This policy has value function

\[
 J_{\mu^0}(i) = 0, \quad \text{for all } i.
\]

The policy \( \mu^1 \) subsequently produced by PI is the one that searches at a state \( i \) if and only if \( r(i) > c \), and has value function satisfying the Bellman equation

\[
 J_{\mu^1}(i) = \begin{cases} 
 0 & \text{if } r(i) \leq c, \\
 r(i) - c + \sum_{m=0}^{i} p(m \mid i) J_{\mu^1}(i-m) & \text{if } r(i) > c.
\end{cases}
\] (4.32)

Note that the values \( J_{\mu^1}(i) \) are nonnegative for all \( i \), since by Prop. 4.5.1, we have

\[
 J_{\mu^1}(i) \geq J_{\mu^0}(i) = 0.
\]

The next policy generated by PI is obtained from the minimization

\[
 \mu^2(i) = \arg \max \left\{ 0, r(i) - c + \sum_{m=0}^{i} p(m \mid i) J_{\mu^1}(i-m) \right\}, \quad i = 1, \ldots, n.
\]
For $i$ such that $r(i) \leq c$, we have $r(j) \leq c$ for all $j < i$ because $r(i)$ is monotonically nondecreasing in $i$. Moreover, using Eq. (4.32), we have $J_{\mu_1}(i-m) = 0$ for all $m \geq 0$. It follows that for $i$ such that $r(i) \leq c$,

\[ 0 \geq r(i) - c + \sum_{m=0}^{i} p(m \mid i)J_{\mu_1}(i-m), \]

and $\mu^2(i) = \text{stop searching}.$

For $i$ such that $r(i) > c$, we have $\mu^2(i) = \text{search}$, since $J_{\mu_1}(i) \geq 0$ for all $i$, so that

\[ 0 < r(i) - c + \sum_{m=0}^{i} p(m \mid i)J_{\mu_1}(i-m). \]

Thus, $\mu^2$ is the same as $\mu^1$ and the PI algorithm terminates. By Prop. 4.5.1, it follows that $\mu^2$ is optimal.

### 4.5.2 Optimistic and Multistep Lookahead Policy Iteration

The PI algorithm that we have discussed so far uses exact policy evaluation of the current policy $\mu^k$ and one-step lookahead policy improvement, i.e., it computes exactly $J_{\mu_k}$, and it obtains the next policy $\mu^{k+1}$ by a one-step lookahead minimization using $J_{\mu_k}$ as an approximation to $J^*$. It is possible to use a more flexible algorithm whereby $J_{\mu_k}$ is approximated by any number of value iterations corresponding to $\mu_k$ (cf. Prop. 4.3.3) and the policy improvement is done using multistep lookahead.

A PI algorithm that uses a finite number $m_k$ of VI steps for policy evaluation of policy $\mu^k$ (in place of the infinite number required by exact PI) is referred to as optimistic. It can be viewed as a combination of VI and PI. The optimistic PI algorithm starts with a function $J_0$, an initial guess of $J^*$. It generates a sequence $\{J_k\}$ and an associated sequence of policies $\{\mu_k\}$, which asymptotically converge to $J^*$ and an optimal policy, respectively. The $k$th iteration starts with a function $J_k$, and first generates $\mu^k$. It then generates $J_{k+1}$ using $m_k$ iterations of the VI algorithm that corresponds to $\mu^k$, starting with $J_k$ as follows.

**Optimistic Policy Iteration: Discounted Problems**

Given the typical function $J_k$:

**Policy improvement** computes a policy $\mu^k$ such that

\[
\mu^k(i) \in \arg \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u) \left(g(i, u, j) + \alpha J_k(j)\right), \quad i = 1, \ldots, n. \quad (4.33)
\]
Optimistic policy evaluation starts with \( \hat{J}_{k,0} = J_k \), and uses \( m_k \) VI iterations for policy \( \mu^k \) to compute \( \hat{J}_{k,1}, \ldots, \hat{J}_{k,m_k} \) according to

\[
\hat{J}_{k,m+1}(i) = \sum_{j=1}^{n} p_{ij}(\mu^k(i)) \left( g(i, \mu^k(i), j) + \alpha \hat{J}_{k,m}(j) \right), \tag{4.34}
\]

for all \( i = 1, \ldots, n \), \( m = 0, \ldots, m_k - 1 \), and sets \( J_{k+1} = \hat{J}_{k,m_k} \).

From Eq. (4.34), it can be seen that one way to interpret optimistic PI is that we approximate \( J_{\mu_k} \) by using \( \mu^k \) for \( m_k \) stages, and adding a terminal cost function equal to the current cost estimate \( J_k \) instead of using \( \mu^k \) for an additional infinite number of stages. Accordingly, simulation-based approximations of optimistic PI, evaluate the cost function \( J_{\mu_k} \) by using \( m_k \)-stage trajectories, with the cost of future stages accounted for with some cost function approximation at the end of the \( m_k \) stages.

The convergence properties of optimistic PI are solid, although it may require an infinite number of iterations to converge to \( J^* \). To see why this is so, suppose that we evaluate each policy with a single VI. Then the method is essentially identical to the VI method, which requires an infinite number of iterations to converge. For the same reason, optimistic PI, when implemented with approximations similar to VI, as in Section 4.4, is subject to the instability phenomenon illustrated in Example 4.4.1. Generally, most practical approximate policy evaluation schemes are optimistic in nature.

The following proposition, shown in the appendix, establishes the validity of optimistic PI. There is a corresponding convergence property for SSP problems, but its currently available proof is fairly complicated. It is given in Section 3.5.1 of the book [Ber12]. Asynchronous versions of optimistic PI also involve theoretical convergence difficulties, which are discussed in Section 2.6.2 of [Ber12] and Section 2.6.3 of [Ber18a].

**Proposition 4.5.2: (Convergence of Optimistic PI)** For the discounted problem, the sequences \( \{J_k\} \) and \( \{\mu_k\} \) generated by the optimistic PI algorithm satisfy

\( J_k \to J^* \), \quad J_{\mu_k} \to J^* . \)
to $J_0$ without affecting the sequence $\{\mu^k\}$ generated by the algorithm. The proof of the appendix also shows that the generated policies $\mu^k$ are optimal after some $k$, but this fact cannot be exploited in practice because the verification that $\mu^k$ is optimal requires additional computations that essentially defeat the purpose of the method.

**Multistep Policy Improvement**

The motivation for multistep policy improvement is that it may yield a better policy $\mu^{k+1}$ than with one-step lookahead. In fact this makes even more sense when the evaluation of $\mu^k$ is approximate, since then the longer lookahead may compensate for errors in the policy evaluation. The method in its exact nonoptimistic form is given below (in a different version it may be combined with optimistic PI, i.e., with policy evaluation done using a finite number of VI iterations).

**Multistep Lookahead Exact Policy Iteration: Discounted Problems**

Given the typical policy $\mu^k$:

**Policy evaluation** computes $J_{\mu^k}(i), i = 1, \ldots, n$, as the solution of the (linear) system of Bellman equations

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

(cf. Prop. 4.2.3).

**Policy improvement with $\ell$-step lookahead** then solves the $\ell$-stage problem with terminal cost function $J_{\mu^k}$. If $\{\hat{\mu}_0, \ldots, \hat{\mu}_{\ell-1}\}$ is the optimal policy of this problem, then the new policy $\mu^{k+1}$ is $\hat{\mu}_0$.

The process is repeated with $\mu^{k+1}$ used in place of $\mu^k$, unless we have $J_{\mu^{k+1}}(i) = J_{\mu^k}(i)$ for all $i$, in which case the algorithm terminates with the policy $\mu^k$.

Exact PI with multistep lookahead has the same solid convergence properties as its one-step lookahead counterpart: it terminates with an optimal policy, and the generated sequence of policies is monotonically improving. The proof is based on a cost improvement property that will be shown as a special case of the subsequent Prop. 4.6.1.

**4.5.3 Policy Iteration for Q-factors**

Similar to VI, we may also equivalently implement PI through the use of
Sec. 4.5 Policy Iteration

Figure 4.5.2 Block diagram of exact PI for Q-factors. Each iteration consists of a policy evaluation using the current policy $\mu$, followed by generation of an improved policy $\bar{\mu}$.

Q-factors. To see this, first note that the policy improvement step may be implemented by minimizing over $u \in U(i)$ the expression

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

which we view as the Q-factor of the pair $(i, u)$ corresponding to $\mu$. Note that we have

$$J_{\mu}(j) = Q_{\mu}(j, \mu(j)).$$

(cf. Prop. 4.2.3).

The following algorithm is thus obtained; see Fig. 4.5.2.

**Exact Policy Iteration for Q-Factors: Discounted Problems**

Given the typical policy $\mu^k$:

**Policy evaluation** computes $Q_{\mu^k}(i, u)$, for all $i = 1, \ldots, n$, and $u \in U(i)$, as the solution of the (linear) system of equations

$$Q_{\mu^k}(i, u) = \sum_{j=1}^{n} p_{ij}(u) \left( g(i, u, j) + \alpha Q_{\mu^k}(j, \mu^k(j)) \right). \quad (4.35)$$
Policy improvement then computes a new policy $\mu^{k+1}$ as
\[
\mu^{k+1}(i) \in \arg \min_{u \in U(i)} Q_\mu^k(i, u), \quad i = 1, \ldots, n. \tag{4.36}
\]
The process is repeated with $\mu^{k+1}$ used in place of $\mu^k$, unless we have $J_{\mu^{k+1}}(i) = J_{\mu^k}(i)$ for all $i$, in which case the algorithm terminates with the policy $\mu^k$.

Note that the system (4.35) has a unique solution, since from the uniqueness of solution of Bellman’s equation, any solution must satisfy
\[
Q_{\mu^k}(j, \mu^k(j)) = J_{\mu^k}(j).
\]
Hence the Q-factors $Q_{\mu^k}(j, \mu^k(j))$ are uniquely determined, and then the remaining Q-factors $Q_{\mu^k}(i, u)$ are also uniquely determined from Eq. (4.35).

The PI algorithm for Q-factors is mathematically equivalent to PI for costs, as given in the preceding subsection. The only difference is that we calculate all the Q-factors $Q_{\mu^k}(i, u)$, rather than just the costs $J_{\mu^k}(j) = Q_{\mu^k}(j, \mu^k(j))$, i.e., just the Q-factors corresponding to the controls chosen by the current policy. However, the remaining Q-factors $Q_{\mu^k}(i, u)$ are needed for the policy improvement step (4.36), so no extra computation is required. It can be verified also that the PI algorithm (4.35)-(4.36) can be viewed as the PI algorithm for the discounted version of the modified problem of Fig. 4.2.2. Asynchronous and optimistic PI algorithms for Q-factors involve substantial theoretical convergence complications, as shown by Williams and Baird [WiB93], which have been resolved in papers by Bertsekas and Yu for discounted problems in [BeY12] and for SSP problems in [YuB13a].

4.6 APPROXIMATION IN VALUE SPACE - PERFORMANCE BOUNDS
We will focus on infinite horizon DP approximations, beginning with discounted problems. Consistently with the finite horizon approximation in value space 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
\[
\min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u)(g(i, u, j) + \alpha \tilde{J}(j)), \tag{4.37}
\]
Approximate minimization at state $i$

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

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

**Figure 4.6.1** 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. 4.6.1.

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 $T^{\ell-1}J$ at the end, where $T$ is the DP operator (4.13).

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. 4.6.1.

In this chapter, we will focus on rollout, and particularly 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 (4.37), or its multistep counterpart.

Performance bounds for this type of approximate PI scheme will be discussed in Section 4.6.3, 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.

### 4.6.1 Limited Lookahead Performance Bounds

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 consider 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. (4.13) and (4.14), the $\ell$-step lookahead policy $\tilde{\mu}$ is defined by

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

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 $\tilde{U}(i)$ of promising controls, and to save computation, we restrict attention to this subset in the one-step lookahead minimization.
Proposition 4.6.1: (Limited Lookahead Performance Bounds)

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

$$
\|J_{\hat{\mu}} - J^*\| \leq \frac{2\alpha^\ell}{1 - \alpha}\|\hat{J} - J^*\|, \quad (4.38)
$$

where $\| \cdot \|$ denotes the maximum norm (4.15).

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

in the equation

$$
\hat{J}(i) = \min_{u \in \mathcal{U}(i)} \sum_{j=1}^{n} p_{ij}(u)(g(i,u,j) + \alpha \hat{J}(j)), \quad i = 1, \ldots, n, \quad (4.39)
$$

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

$$
\hat{J}(i) \leq \hat{J}(i) + c, \quad i = 1, \ldots, n. \quad (4.40)
$$

Then

$$
J_{\hat{\mu}}(i) \leq \hat{J}(i) + \frac{c}{1 - \alpha}, \quad i = 1, \ldots, n. \quad (4.41)
$$

An important point regarding the bound (4.38) is that $J_{\hat{\mu}}$ is unaffected by a constant shift in $\hat{J}$ [an addition of a constant to all values $\hat{J}(i)$]. Thus $\|\hat{J} - J^*\|$ in Eq. (4.38) can be replaced by the potentially much smaller number

$$
\min_{\beta \in \mathbb{R}} \|\hat{J} + \beta c - J^*\| = \min_{\beta \in \mathbb{R}} \max_{i=1, \ldots, n} |\hat{J}(i) + \beta - J^*(i)|.
$$

We thus obtain the following performance bound,

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

which is stronger than the one of Eq. (4.38) that corresponds to $\beta = 0$.

The preceding bound shows that performance is improved when the length $\ell$ of the lookahead in increased, and also when the lookahead cost approximation $\hat{J}$ 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 asserting 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 (4.40), we note that it guarantees that when $c \leq 0$, the cost $J_{\mu}$ of the one-step lookahead policy is no larger than $\bar{J}$. When $c = 0$, this condition bears resemblance with the consistent improvement condition for deterministic rollout methods (cf. Section 2.4.1). If $\bar{J} = J_{\mu}$ for some policy $\mu$ (as in the case of the pure form of rollout to be discussed in Section 4.6.2), then Eq. (4.40) holds as an equation with $c = 0$, and from Eq. (4.41), it follows that $J_{\mu} \leq J_{\mu}$.

Unfortunately, the bound (4.38) 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 $\bar{J}$.

Example 4.6.1

Consider the two-state discounted problem shown in Fig. 4.6.2, 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. (4.38) (cf. Prop. 4.6.1(b)). 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. (4.38) holds with equality.

4.6.2 Rollout

Let us first consider rollout in its pure form, where \( \tilde{J} \) in Eq. (4.37) 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 must 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 the following proposition shows. This is to be expected since rollout is one-step PI, so Prop. 4.5.1 applies.

**Proposition 4.6.2: (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. (4.39) 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 4.6.2 (Rollout with Multiple Heuristics)**

Let \( \mu_1, \ldots, \mu_M \) be stationary policies, let
\[
\tilde{J}(i) = \min \left\{ J_{\mu_1}(i), \ldots, J_{\mu_M}(i) \right\}, \quad i = 1, \ldots, n,
\]
let \( T(i) \subset U(i) \), and assume that \( \mu_1(i), \ldots, \mu_M(i) \in T(i) \) for all \( i = 1, \ldots, n \). Then, for all \( i \) and \( m = 1, \ldots, M \), we have

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

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. 4.6.1(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 \( \mu_1, \ldots, \mu_M \).

**Combined Multistep Lookahead, Rollout, and Terminal Cost Approximation**

Let us next discuss a 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. 4.6.3. We can view this form of rollout as a single optimistic policy iteration combined with multistep lookahead; cf. Eqs. (4.33)-(4.34). 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 result generalizes the performance bounds given for limited lookahead and rollout of the preceding two subsections. In particular, part (a) of the proposition follows by applying Prop. 4.6.1(a), since the truncated rollout scheme of this section can be viewed as \( \ell \)-step approximation in value space with terminal cost function \( T_{\mu}^m \hat{J} \) at the end of the lookahead, where \( T_{\mu} \) is the DP operator of Eq. (4.14).
Sec. 4.6 Approximation in Value Space - Performance Bounds

Proposition 4.6.3: (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 $\|J\| = \max_{i=1,...,n} |J(i)|$. 

Figure 4.6.3 Illustration of two-step lookahead, rollout with a policy $\mu$ for a limited and state-dependent number of steps, and a terminal cost function approximation $\tilde{J}$. A Monte Carlo tree search scheme may also be used for multistep lookahead; cf. Section 2.4.2. Note that the three components of this scheme (multistep lookahead, rollout with $\mu$, and cost approximation $\tilde{J}$) can be designed independently of each other. Moreover, while the multistep lookahead is implemented online, $\mu$ and $\tilde{J}$ must be available from an earlier off-line computation.
(b) Assume that for some constant $c$, $\tilde{J}$ and $\mu$ satisfy the condition
\[
\sum_{j=1}^{n} p_{ij}(\mu(i)) \left( g(i, \mu(i), j) + \alpha \tilde{J}(j) \right) \leq \tilde{J}(i) + c, \quad i = 1, \ldots, n. \tag{4.42}
\]
Then
\[
J_\mu(i) \leq \tilde{J}(i) + \frac{c}{1-\alpha}, \quad i = 1, \ldots, n. \tag{4.43}
\]

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

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. 4.6.1(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, however, 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.

The truncated rollout scheme of Fig. 4.6.3 has been adopted in the rollout backgammon algorithm of Tesauro and Galperin [TeG96], with $\mu$ and the terminal cost function approximation $\tilde{J}$ 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).
4.6.3 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 $\tilde{J}_{\mu_k}$, 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 $\tilde{J}_{\mu_k}$.

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

$$\max_{i=1,\ldots,n} |\tilde{J}_{\mu_k}(i) - J_{\mu_k}(i)| \leq \delta,$$  \hspace{1cm} (4.44)

and an approximate policy improvement error satisfying

$$\max_{i=1,\ldots,n} \left| \sum_{j=1}^{n} p_{ij}(\mu_{k+1}(i)) \left( g(i, \mu_{k+1}(i), j) + \alpha \tilde{J}_{\mu_k}(j) \right) - \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u) \left( g(i, u, j) + \alpha \tilde{J}_{\mu_k}(j) \right) \right| \leq \epsilon,$$  \hspace{1cm} (4.45)

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 4.6.4: (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 (4.44) and the approximate policy improvement (4.45). Then the policy error

$$\max_{i=1,\ldots,n} |J_{\mu_k}(i) - J^*(i)|,$$

becomes less or equal to

$$\frac{\epsilon + 2\alpha \delta}{(1 - \alpha)^2},$$

asymptotically as $k \to \infty$.

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
Figure 4.6.4 Illustration of typical behavior of approximate PI. In the early iterations, the method tends to make rapid and fairly monotonic progress, until $J_{\mu k}$ 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.

Performance Bound for the Case Where Policies Converge

Generally, the policy sequence $\{\mu^k\}$ generated by approximate PI may oscillate between several policies, as noted earlier. However, under some circumstances the sequence will converge to some policy $\tilde{\mu}$, in the sense that

$$\mu^{k+1} = \mu^k = \tilde{\mu} \quad \text{for some} \ k.$$  \hspace{1cm} (4.46)

An important case where this happens is aggregation methods, which will be discussed in Chapter 5. In this case the behavior of the method is more regular, and we can show a more favorable bound than the one of Prop.
Proposition 4.6.5: (Performance Bound for Approximate PI when Policies Converge) Let $\tilde{\mu}$ be a policy generated by the approximate PI algorithm under conditions (4.44), (4.45), and (4.46). Then we have

$$\max_{i=1,\ldots,n} |J_{\tilde{\mu}}(i) - J^*(i)| \leq \frac{\epsilon + 2\alpha \delta}{1 - \alpha}.$$
the error amplification phenomenon illustrated in Example 4.4.1. 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.

4.7 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 SSP problems.

4.7.1 Self-Learning and Actor-Critic Methods

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 policy improvement. The algorithm that performs the policy evaluation is usually called a critic, and if a neural network is used as the parametric architecture, it is called a critic network. 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.

An optimistic PI-type method is often called an actor-critic method if it involves approximations in both its actor and its critic portions. When two separate neural networks are used as the parametric architecture in the actor and the critic, the method is said to involve an actor-critic 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. Note that the evaluation may be optimistic, in the sense that the number of samples used between policy updates may be limited, even very small.

(b) Improve the current policy \( \mu_k \) (actor): Given the approximate policy evaluation \( \tilde{J}_{\mu_k} \), we generate or “learn” the new policy \( \mu_k^{+1} \) through the lookahead minimization

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

(4.47)
Alternatively we compute the minimizing control $u^s$ at a set of sample states $i^s$, $s = 1, \ldots, q$, through

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

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 2.1.5). This scheme may also be “incremental” in the sense that the improved policy may be updated using very few sample pairs $(i^s, u^s)$.

These two operations are sequentially performed up to the point where some form of “convergence” occurs. To summarize, actor-critic systems can be viewed as a repeated application of a two-step process:

(a) **Critic step**: A form of least squares regression based on (many or few) cost samples, aimed to evaluate better the current policy (while keeping that policy unchanged).

(b) **Actor step**: The improved policy is computed at states of interest via the lookahead minimization (4.47). Alternatively, a form of least squares regression is performed based on (many or few) state-control sample pairs, aiming to improve the current policy (while keeping the policy evaluation unchanged).

Note that while approximate PI schemes of the type described above have been widely reported and advocated, they offer very little in the way of performance guarantees (at least when either costs or policies, or both, are represented with approximation architectures). Moreover, while in these schemes the system learns by itself, it does not learn itself, in the sense that it does not construct a mathematical model for itself. An alternative is to adopt instead a 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 (cf. Section 1.3.8). However, we will not discuss system identification and the construction of models in this book.

In what follows in this section, and Sections 4.8 and 4.9, we will focus on critic-only approximate PI methods, where the policy improvement steps are “exact” in the sense that they are performed with Eq. (4.47). Actor-only and actor-critic methods will be discussed further in Section 5.7.

### 4.7.2 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 [cf. Eqs. (4.25)-(4.26)] generates the new policy $\tilde{\mu}$ with a policy evaluation/policy improvement process. We approximate this process as follows; see Fig. 4.7.1.

(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:

$$r \in \arg \min_r \sum_{s=1}^{q} (\tilde{J}_\mu(i^s, r) - \beta^s)^2.$$  \hspace{1cm} (4.48)

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

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

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 $\mu^{b-1}$; 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. (4.48) can be solved with the incremental methods discussed in Section 3.1.3. In particular the incremental gradient method is given by

\[
\begin{align*}
    r^{k+1} &= r^k - \gamma^k \nabla \hat{J}(i^k, r^k) \left( \hat{J}(i^k, r^k) - \beta^k \right),
\end{align*}
\]

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 (4.48) using the exact linear least squares formula.

(b) Approximate policy improvement: Having solved the approximate policy evaluation problem (4.48), the new “improved” policy \( \hat{\mu} \) is obtained by the approximate policy improvement operation

\[
\begin{align*}
    \hat{\mu}(i) \in \arg \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u) \left( g(i, u, j) + \alpha \hat{J}(j, \tau) \right), \quad i = 1, \ldots, n,
\end{align*}
\]

(4.49)

where \( \tau \) is the parameter vector obtained from the policy evaluation operation (4.48).

Trajectory Reuse and Bias-Variance Tradeoff

As we have noted, to each training pair \((i^*, \beta^*)\) there corresponds an \( N \)-stage trajectory over which the sample cost \( \beta^* \) 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 subsequent states \( i_1, i_2, \) etc, 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 4.7.4.

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 4.9; 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.

### 4.7.3 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. Eqs. (4.35)-(4.36)] generates the new policy $\tilde{\mu}$ with a policy evaluation-policy improvement process. We approximate this process as follows; see Fig. 4.7.2.

(a) **Approximate policy evaluation**: Here we introduce a parametric architecture $\tilde{Q}_\mu(i, u, r)$ for the Q-factors of $\mu$. 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:

$$\mathbf{r} \in \arg \min \sum_{s=1}^{q} \left( \tilde{Q}_\mu(i^s, u^s, r) - \beta^s \right)^2. \quad (4.50)$$

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, and a terminal cost \( \alpha J_N(i) \) may be added as in the model-based case of Section 4.7.2. Again an incremental method may be used to solve the training problem (4.50).

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

\[
\tilde{\mu}(i) \in \arg \min_{u \in U(i)} \hat{Q}_\mu(i, u, \overline{\theta}), \quad i = 1, \ldots, n,
\]

where \( \overline{\theta} \) is the parameter vector obtained from the policy evaluation operation (4.50).

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 4.7.2. 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 \(u \neq \mu(i)\).

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 \(\tilde{J}_\mu(j, \tau)\), using the regression (4.48), and then use a second sampling process and regression to approximate further the (already approximate) Q-factor

\[
\sum_{j=1}^{n} p_{ij}(u)(g(i, u, j) + \alpha \tilde{J}_\mu(j, \tau)),
\]

with some \(\tilde{Q}_\mu(i, u, \tau)\) possibly obtained with a policy approximation architecture (see the discussion of Section 2.1.3 on model-free approximation in policy space). Finally, once \(\tilde{Q}_\mu(i, u, \tau)\) is obtained with this approximation in policy space, the “improved” policy \(\tilde{\mu}\) is obtained from the minimization (4.51). 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 \(\tilde{Q}_\mu(i, u, \tau)\), this scheme is more complex, but allows trajectory reuse and thus deals better with the exploration issue.

### 4.7.4 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 (4.48) and (4.50) 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 4.9, 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

\[
\hat{g}(i, u, j) = g(i, u, j) + V(j) - V(i), \quad i = 1, \ldots, n,
\]
Sec. 4.7  Simulation-Based Policy Iteration

[cf. Eq. (4.11)] for SSP problems, where $V$ can be any approximation to $J^*$. The corresponding formula for discounted problems is

$$\hat{g}(i, u, j) = g(i, u, j) + \alpha V(j) - V(i), \quad i = 1, \ldots, n.$$ 

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. The next chapter 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. We will discuss the generic issue of inadequate exploration, which we noted in earlier in connection with model-based approximate PI. This is indeed the Achilles’ heel of approximate simulation-based PI schemes, and is often not addressed seriously enough in many practical implementations.

In particular, when evaluating a policy $\mu$ with trajectory reuse, we will be generating many cost or Q-factor samples that start from states frequently visited by $\mu$, but this may bias the simulation by underrepresenting states that are unlikely to occur under $\mu$. As a result, the cost or Q-factor estimates of these underrepresented states may be highly inaccurate, causing potentially serious errors in the calculation of the improved policy $\pi$ via the policy improvement operation.

One possibility to improve the exploration of the state space is to use a large number of initial states to form a rich and representative subset, thereby limiting trajectory reuse. It may then be necessary to use relatively short trajectories to keep the cost of the simulation low. However, when using short trajectories it will 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.
There have been other related approaches to improve exploration, particularly in connection with the temporal difference methods to be discussed in Section 4.9. 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. Note, however, that it may still be difficult to ensure that the mixed on-and-off policy will induce sufficient exploration. 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], and the monograph [RVK18].

**Oscillation Issues**

Contrary to exact PI, which is guaranteed to yield an optimal policy, approximate PI produces a sequence of policies, which are only guaranteed to lie asymptotically within a certain error bound from the optimal; cf. Prop. 4.6.4. 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, and it is geometrically explained in the books [BeT96] (Section 6.4.2) and [Ber12] (Section 6.4.3).

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 5 and the approximate policy iteration survey [Ber11a]. Also, when policies converge, there is a more favorable error bound, cf. Prop. 4.6.5.

### 4.8 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

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

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

\[
(FQ)(i, u) = \sum_{j=1}^{n} p_{ij}(u) \left( g(i, u, j) + \alpha \min_{v \in U(j)} Q(j, v) \right) , \text{ for all } (i, u) . \tag{4.52}
\]

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. (4.52) 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:

\[
Q_{k+1}(i, u) = (1 - \gamma^k)Q_k(i, u) + \gamma^k(F_kQ_k)(i, u) , \text{ for all } (i, u) . \tag{4.53}
\]
Infinite Horizon Reinforcement Learning Chap. 4

where

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

(4.54)

Note that \((F_k Q_k)(i_k, u_k)\) is a single sample approximation of the expected value defining \((F Q_k)(i_k, u_k)\) in Eq. (4.52).

To guarantee the convergence of the algorithm (4.53)-(4.54) 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 (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.

### 4.8.1 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
   \[ u^{k+1} \in \arg \min_{u \in U(i^{k+1})} Q(i^{k+1}, u, r^k) \]
   [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
   \[ r^{k+1} = r^k - \gamma^k \nabla Q(i^k, u^k, r^k)q_k, \]
   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
   \[ q_k = \hat{Q}(i^k, u^k, r^k) - \alpha \hat{Q}(i^{k+1}, u^{k+1}, r^k) - g(i^k, u^k, i^{k+1}). \]

The vector $\nabla \hat{Q}(i^k, u^k, r^k)q_k$ 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 $p_{i^k j}(u^k)$, 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).

4.9 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 4.7.2. 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.

Approximate Policy Evaluation Using Projections

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 (4.57) 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

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

(4.58)

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

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

(4.59)

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. (4.55), 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

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

(4.60)

where \(\mathbb{R}^m\) denote 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. (4.58), so \( \Pi(J) \) is of the form \( \Phi r^* \), where

\[
\begin{align*}
  r^* & \in \arg \min_{r \in \mathbb{R}^m} \| \Phi r - J \|_2^2 = \arg \min_{r \in \mathbb{R}^m} \sum_{i=1}^n \xi_i (\phi(i)' r - J(i))^2. \\
  \text{(4.61)}
\end{align*}
\]

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,

\[
r^* = \left( \sum_{i=1}^n \xi_i \phi(i) \phi(i)' \right)^{-1} \sum_{i=1}^n \xi_i \phi(i) J(i), \quad \text{(4.62)}
\]

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. (4.62) 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

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

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 estimates (4.63) 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.†

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

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

\[
\text{(4.65)}
\]
Given the Monte Carlo approximation of the two terms in Eq. (4.62), we can estimate \( r^* \) with

\[
\mathbf{r} = \left( \sum_{s=1}^{q} \phi(i^s)\phi(i^s)' \right)^{-1} \sum_{s=1}^{q} \phi(i^s)\beta^s, \tag{4.66}
\]

(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 (4.61) with the following least squares training problem

\[
\mathbf{r} \in \arg\min_{\mathbf{r} \in \mathbb{R}^m} \sum_{s=1}^{q} (\phi(i^s)'\mathbf{r} - \beta^s)^2. \tag{4.67}
\]

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

(a) Replacing expected values in the exact projection formula (4.62) by simulation-based estimates [cf. Eq. (4.66)].

(b) Replacing the exact least squares problem (4.61) with a simulation-based least squares approximation [cf. Eq. (4.67)].

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

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. (4.64) has the form

\[
\frac{1}{q} \sum_{s=1}^{q} \phi(i^s)n(i^s) = \frac{1}{q} \sum_{i=1}^{n} \phi(i) \sum_{s=1}^{q} \delta(i^s = i)n(i^s) = \frac{1}{q} \sum_{i=1}^{n} \phi(i) \sum_{s'=1}^{q} \delta(i^s' = i) \sum_{s=1}^{q} \delta(i^s = i)n(i^s)
\]

and converges to 0 as \( q \to \infty \), assuming that each index \( i \) is sampled infinitely often so that Eq. (4.65) can be used.

‡ The preceding derivation and the formula (4.66) actually make sense even if \( \xi = (\xi_1, \ldots, \xi_n) \) has some zero components, as long as the inverses in Eqs. (4.62) and (4.66) exist. This is related to the concept of seminorm projection; see [YuB12] for an approximate DP-related discussion.
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.,

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

(4.68)

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. (4.68).

**Projected Equation View of Approximate Policy Evaluation**

Let us now discuss the approximate policy evaluation method for costs of Section 4.7.2 [cf. Eq. (4.48)]. It can be interpreted in terms of a projected equation, written abstractly as

$$\tilde{J}_\mu \approx \Pi(T^N_\mu \tilde{J}),$$

(4.69)

where:

1. $\tilde{J}$ is some initial guess of $J_\mu$ (the terminal cost function approximation discussed in Section 4.7.2), and $\tilde{J}_\mu$ is the vector

$$\tilde{J}_\mu = (\tilde{J}(1, \tau), \ldots, \tilde{J}(n, \tau)),$$

which is the approximate policy evaluation of $\mu$, used in the policy improvement operation (4.49). Here $\tau$ is the solution of the training problem (4.48).

2. $T_\mu$ 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. (4.56).

3. $T^N_\mu$ 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 (4.48). In particular, $(T^N_\mu \tilde{J})(i)$ is the cost associated with starting at $i$, using $\mu$ for $N$ stages, and incurring a terminal cost specified by the terminal cost function $\tilde{J}$. The sample state-cost pairs $(i^s, \beta^s)$ are obtained from trajectories corresponding to this $N$-stage problem.

† The equation (4.69) assumes that all trajectories have equal length $N$, and thus does not allow trajectory reuse. If trajectories of different lengths are allowed, the term $T^N_\mu$ in the equation should be replaced by a more complicated weighted sum of powers of $T_\mu$; see the paper [YuB12] for related ideas.
(d) $\Pi(T_N^N \hat{J})$ denotes projection of the vector $T_N^N \hat{J}$ 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 (4.48). In particular, the cost samples $\beta_s$ of the training set are noisy samples of the values $(T_N^N \hat{J})(i^s)$, and the projection is approximated with a least squares minimization, to yield the function $\hat{J}_\mu$ of Eq. (4.69).

Suppose now that $T_N^N \hat{J}$ 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. (4.69), the approximate evaluation $\hat{J}_\mu$ of $\mu$ approaches the projection of $J_\mu$ on the approximation manifold (4.57), 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 4.7.2, which is based on Eq. (4.69).

**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 approximate 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 $\hat{J}_\mu = \Phi r$, and all aim at the same problem. This is the problem of solving a projected equation of the form

$$\Phi r = \Pi(T^{(\lambda)}_\mu \Phi r),$$

where $T_\mu$ is the operator (4.56), $T^{(\lambda)}_\mu J$ is defined by

$$(T^{(\lambda)}_\mu J)(i) = (1 - \lambda) \sum_{\ell=0}^{\infty} \lambda^\ell (T^{\ell+1}_\mu J)(i), \quad i = 1, \ldots, n,$$

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 $J = T^{(\lambda)}_\mu J$ is as a multistep version of Bellman’s equation. It has

† TD stands for “temporal difference,” LSTD stands for “least squares temporal difference,” and LSPE stands for “least squares policy evaluation.”
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 (4.70) 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,

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

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 \( r \) that solves Eq. (4.71) satisfies

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

[cf. Eq. (4.67)], and defines the approximate evaluation \( \Phi \tau \) of \( J_\mu \). This relation can be expressed as a linear equation, which in principle can be solved in closed form [cf. Eq. (4.66)], 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 (4.72) 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

\[
C \tau = d,
\]

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 \( \tilde{J}_\mu(i) = \Phi \tau \) where \( \tau = C^{-1}d \) is the solution of Eq. (4.73).

(b) The LSPE(\( \lambda \)) method solves the projected equation (4.70) by using a simulation-based projected value iteration,

\[
J_{k+1} = \tilde{\Pi} (T_\mu^{(\lambda)} J_k).
\]
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 (4.73). It can also be viewed as a stochastic gradient method, or as a stochastic version of the proximal algorithm for solving this linear equation (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 4.7.2. 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 $T_\mu(\lambda) = T_\mu$, the samples of $T_\mu \Phi r$ in Eq. (4.72) are

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

and the least squares problem in Eq. (4.72) has the form

$$\min_r \sum_{s=1}^{q} \left( \phi(i^s)'r - g(i^s, i^{s+1}) - \alpha \phi(i^{s+1})'r \right)^2.$$  \hfill (4.75)

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

$$\sum_{s=1}^{q} \phi(i^s)(\phi(i^s)'r - g(i^s, i^{s+1}) - \alpha \phi(i^{s+1})'r) = 0.$$  \hfill (4.76)
Solving this equation for $\tau$ yields the LSTD(0) solution:

$$
\tau = \left( \sum_{s=1}^{q} \phi(i^s) \left( \phi(i^s) - \alpha \phi(i^{s+1}) \right)' \right)^{-1} \sum_{s=1}^{q} \phi(i^s) g(i^s, i^{s+1}). \tag{4.77}
$$

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

$$
d^s(\tau) = \phi(i^s)\tau - g(i^s, i^{s+1}) - \alpha \phi(i^{s+1})\tau \tag{4.78}
$$

that appears in the least squares sum minimization (4.75) and Eq. (4.76) is referred to as the 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

$$
J_{k+1} = \bar{\Pi}(T_{\mu}J_k),
$$

[cf. Eq. (4.74)]. At the $k$th iteration, it uses only the samples $s = 1, \ldots, k$, and updates the parameter vector according to

$$
r^{k+1} = r^k - \left( \sum_{s=1}^{k} \phi(i^s) \phi(i^s)' \right)^{-1} \sum_{s=1}^{k} \phi(i^s) d^s(r^k), \quad k = 1, 2, \ldots, \tag{4.79}
$$

where $d^s(r^k)$ is the temporal difference of Eq. (4.78), 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. (4.79) 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. (4.79)] 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

\[
    r^{k+1} = r^k - \gamma^k \phi(i^k)d_k(r^k), \quad k = 1, 2, \ldots, \tag{4.80}
\]

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 (4.75), but with \( \mathbf{r} \) 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. (4.75) is the vector \( \phi(i^k)d_k(r^k) \) that appears in the TD(0) iteration (4.80) (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 (4.79) 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 \( \left( \sum_{s=1}^{k} \phi(i^s)\phi(i^s)^\prime \right)^{-1} \). 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 (4.73) 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 \( \Pi_{T_{\mu}^\lambda}(\mathbf{r}) \) may not be a contraction mapping (even though \( T_{\mu} \) is) and the projected value iteration (4.74) may not be convergent (it turns out that the mapping \( \Pi_{T_{\mu}^\lambda}(\mathbf{r}) \) is guaranteed to be a contraction for \( \lambda \) sufficiently close to 1).

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
Infinite Horizon Reinforcement Learning Chap. 4

The difference $\Phi r^*_\lambda - \Pi(J_\mu)$ is commonly referred to as the bias and is illustrated in Figure 4.9.1. As indicated in this figure and as the estimate

$$\| J_\mu - \Phi r^*_\lambda \|_\xi \leq \frac{1}{\sqrt{1 - \alpha^2}} \| J_\mu - \Pi J_\mu \|_\xi,$$  \hspace{1cm} (4.81)
Sec. 4.10 Exact and Approximate Linear Programming

(4.81) suggests, there is a bias-variance tradeoff. As \( \lambda \) is decreased, the solution of the projected equation (4.70) 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 4.7.2 in connection with the use of short trajectories.

4.10 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

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

so that \( J^*(1), \ldots, J^*(n) \) solve the linear program

\[
\begin{align*}
\text{maximize} & \quad \sum_{i=1}^{n} J(i) \\
\text{subject to} & \quad \text{constraint (4.82)},
\end{align*}
\]

(see Fig. 4.10.1).

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

\[
J_0(i) \leq \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u)(g(i, u, j) + \alpha J_0(j)) = J_1(i), \quad \text{for all } i.
\]

This inequality can be used to show that

\[
J_0(i) \leq J_1(i) \leq \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u)(g(i, u, j) + \alpha J_1(j)) = J_2(i), \quad \text{for all } i,
\]

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 (4.82), $J^*$ is the largest on a component-by-component basis.

Unfortunately, for large $n$ the dimension of the linear program (4.83) 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

$$\text{maximize } \sum_{i \in \tilde{I}} \tilde{J}(i, r)$$

subject to

$$\tilde{J}(i, r) \leq \sum_{i=1}^{n} p_{ij}(u)(g(i, u, j) + \alpha \tilde{J}(j, r)), \quad i \in \tilde{I}, \ u \in \tilde{U}(i),$$

subject to

$$\tilde{J}(i, r) \leq \sum_{i=1}^{n} p_{ij}(u)(g(i, u, j) + \alpha \tilde{J}(j, r)), \quad i \in \tilde{I}, \ u \in \tilde{U}(i),$$

subject to

$$\tilde{J}(i, r) \leq \sum_{i=1}^{n} p_{ij}(u)(g(i, u, j) + \alpha \tilde{J}(j, r)), \quad i \in \tilde{I}, \ u \in \tilde{U}(i),$$

subject to

$$\tilde{J}(i, r) \leq \sum_{i=1}^{n} p_{ij}(u)(g(i, u, j) + \alpha \tilde{J}(j, r)), \quad i \in \tilde{I}, \ u \in \tilde{U}(i),$$

subject to

$$\tilde{J}(i, r) \leq \sum_{i=1}^{n} p_{ij}(u)(g(i, u, j) + \alpha \tilde{J}(j, r)), \quad i \in \tilde{I}, \ u \in \tilde{U}(i),$$

subject to

$$\tilde{J}(i, r) \leq \sum_{i=1}^{n} p_{ij}(u)(g(i, u, j) + \alpha \tilde{J}(j, r)), \quad i \in \tilde{I}, \ u \in \tilde{U}(i),$$

subject to

$$\tilde{J}(i, r) \leq \sum_{i=1}^{n} p_{ij}(u)(g(i, u, j) + \alpha \tilde{J}(j, r)), \quad i \in \tilde{I}, \ u \in \tilde{U}(i),$$

subject to

$$\tilde{J}(i, r) \leq \sum_{i=1}^{n} p_{ij}(u)(g(i, u, j) + \alpha \tilde{J}(j, r)), \quad i \in \tilde{I}, \ u \in \tilde{U}(i),$$

subject to

$$\tilde{J}(i, r) \leq \sum_{i=1}^{n} p_{ij}(u)(g(i, u, j) + \alpha \tilde{J}(j, r)), \quad i \in \tilde{I}, \ u \in \tilde{U}(i),$$
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 $J(i, r)$ 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.

### 4.11 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 Section 2.1.5). 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 4.11.1: (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. 4.11.1. 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 4.11.2: (PID control)

A popular and time-honored scheme for control system design is the PID (Proportional-Integral-Derivative) controller. It is widely used to maintain the output of a single-input single-output dynamic system around a set point (or to follow a sequence of set points). 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; see Fig. 4.11.2.

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 a recursion such as

$$ d_k = (1 - \beta)d_{k-1} + \beta(e_k - e_{k-1}), $$
Figure 4.11.2. Illustration of PID control for a system with a single control input \( u_k \) and a single output \( y_k \). The objective is to keep the output near a “set point” \( \bar{y} \). The controller observes the error \( e_k = y_k - \bar{y} \) and applies control

\[
u_k = r_p e_k + r_i z_k + r_d d_k,
\]

where \( r_p, r_i, \) and \( r_d \) are scalar parameters to be determined, \( z_k \) is the sum of all errors up up time \( k \), generated by

\[
z_k = z_{k-1} + e_k,
\]

and \( d_k \) is a damped version of the error difference \( e_k - e_{k-1} \), generated by

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

where \( \beta \) is damping factor with \( 0 < \beta < 1 \). A mathematical model of the system need not be known. The three terms comprising the controller, \( r_p e_k \), \( r_i z_k \), \( r_d d_k \) are the proportional, integral, and derivative terms, respectively.

where \( \beta \) is damping factor with \( 0 < \beta < 1 \) (this is to mitigate the effects of noise in the error \( e_k \)). 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, contrary to the MPC schemes discussed in Section 2.5.1, PID control is a model-free scheme: a mathematical model of the system need not be known. Moreover a single set of \((r_p, r_i, r_d)\) values may be sufficient to provide good performance over a wide range of operating conditions for the system.

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/heuristic; see e.g., the books by Astrom and Hagglund [AsH95], [AsH06]. The approach of optimizing the parameters of a controller by using a cost function that encodes the steady-state and transient system performance is known as extremum seeking control; see e.g., the books by Ariyur and Krstic [ArK03], and by Zhang and Ordonez [ZhO11]. Its application to the PID case is described in several sources; see e.g., the paper by Killingsworth and Krstic [KiK06]. For recent related work, see
Fridhauf, Krstic, and Basar [FKB13], Radenkovic and Krstic [RaK18], and the references quoted there. Another related optimization-based method that has been applied to PID control is iterative feedback tuning; see Lequin et al. [LGM03], and the references quoted there. Collectively these methods address reasonably well the tuning of PID-type schemes as applied to problems where the model parameters change slowly.

Our main purpose is this example is to point out that PID controller design can also be viewed within the context of approximation in policy space. This brings to bear the methodology of this section for the purpose of determining the controller parameters. We note, however, that PID control and other adaptive schemes are often intended for use in real-time control of systems with frequently and unpredictably changing parameters. This is a very challenging problem that RL methods are not able to address with the current state of the art.

Example 4.11.3: (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

$\dagger$ The policy parametrization (4.84) may not differentiable with respect to $r$, and this may make it unsuitable for use in some methods that rely on gradients. In such cases one may approximate the policy $\tilde{\mu}(i, r)$ of Eq. (4.84) with a randomized policy that applies at state $i$ a control $u \in U(i)$ with probability

$$
p(u \mid i, r) = \frac{e^{-\beta \hat{Q}(i, u, r)}}{\sum_{v \in U(i)} e^{-\beta \hat{Q}(i, v, r)}},
$$

where $\hat{Q}(i, u, r)$ is the approximate Q-factor given by

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

and $\beta$ is a positive scalar that controls the accuracy of the approximation. This is called the soft-min approximation in the literature (or soft-max approximation in the case of maximization of reward).
Sec. 4.11  Approximation in Policy Space

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.

As an illustration, this type of scheme has been 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.

**Example 4.11.4: (Policy parametrization and multiagent
problems)**

An important point 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 approxi-
mate VI and PI methodologies. As a result, the idea of policy parametriza-
tion 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 (collective) 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
associated Bellman’s equation for this type of problem. In particular, the VI
and PI methods are not valid anymore. 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 described.

In what follows we will discuss two training approaches for approx-
imation in policy space. The first approach (Section 4.11.1) is based on
determining the parameter \( r \) by optimization over some measure of cost
derived from the given DP problem. The second approach (Section 4.11.2)
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 4.11.3, we use
approximation in policy space for policy improvement within a PI frame-
work. Here we apply the expert training approach of Section 4.11.2, with
rollout being used as a software expert.
Figure 4.11.3 The optimization framework for approximation in policy space. 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 \( i_0 \).

4.11.1 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

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

(4.85)

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 distribution of the initial state \( i_0 \) (cf. Fig. 4.11.3). In the case where the initial state \( i_0 \) is known and fixed, the method involves just minimization of \( J_{\tilde{\mu}(r)}(i_0) \) over \( r \). This simplifies a great deal the minimization, particularly when the problem is deterministic.

Gradient Methods for Cost Optimization

Let us first consider methods that perform the minimization (4.85) 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

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

(4.86)
assuming that $J_{\tilde{\mu}(\cdot)}(i_0)$ is differentiable with respect to $r$. Here $\gamma^k$ is a positive stepsize 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 $\nabla J_{\tilde{\mu}(\cdot)}(r^k)(i_0)$ may not be explicitly available. In this case, the gradient must be approximated by finite differences of cost function values $J_{\tilde{\mu}(\cdot)}(i_0)$. 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 (4.86) 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. Popular methods of this type are based on incremental gradient ideas (cf. Section 3.1.3) and 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].† Some of these methods, called likelihood-ratio policy gradient methods, also use 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 relevant ideas.

### Policy Gradient Methods for Randomized Policies

To get a sense of the general principle underlying the gradient-based approach that uses sampling, let us digress from the DP context of this chapter, and consider the generic optimization problem

$$\min_{z \in Z} F(z), \quad (4.87)$$

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

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

$$\min_{p \in \mathcal{P}_Z} E_p \{ F(z) \}, \quad (4.88)$$

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 enhances the use of randomization schemes and simulation-based methods, even if the original problem is deterministic. Moreover, the stochastic optimization problem (4.88) may have some nice differentiability properties that are lacking in the original deterministic version (4.87); see the paper [Ber73] for an analysis of this issue under convexity assumptions on $F$.

At this point it is not clear how the stochastic optimization problem (4.88) 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 (4.88), 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 $\mathcal{P}_Z \subset \mathcal{P}_Z$ 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 (4.88) 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

$$r^{k+1} = r^k - \gamma^k \nabla \left( E_{p(z;r^k)} \{ F(z) \} \right), \quad k = 0, 1, \ldots, \quad (4.89)$$

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 a random search direction and only two sample function values per iteration. These are generally faster than 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, and the paper by Nesterov and Spokoiny [NeS17] for a more theoretical view.

**Example 4.11.5 (An Incremental Method Based on Two Function Values per Iteration)**

In this example, we assume that $p(z;r)$ is a discrete distribution. For simplicity we first consider the case where $z$ and $r$ are scalars, and later discuss
Sec. 4.11 Approximation in Policy Space

The discrete distribution \( p(z; r) \) used in Example 4.11.5.

Figure 4.11.4 The discrete distribution \( p(z; r) \) used in Example 4.11.5.

the multidimensional case where \( z \) and \( r \) belong to \( \mathbb{R}^n \). In particular, we assume that \( p(z; r) \) is symmetric 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

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

and

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

The gradient iteration (4.89) becomes

\[
r^{k+1} = r^k - \gamma^k \sum_{i=1}^{m} p_i (\nabla F(r + \epsilon_i) + \nabla F(r - \epsilon_i)), \quad k = 0, 1, \ldots, \quad (4.90)
\]

Let us now consider approximation of the gradient 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 approximate the gradient iteration (4.90) by

\[
r^{k+1} \approx r^k - \gamma^k \sum_{i=1}^{m} \frac{p_i}{\epsilon_i} \left( F(r^k + \epsilon_i) - F(r^k - \epsilon_i) \right), \quad k = 0, 1, \ldots, \quad (4.91)
\]

One possible sample-based/incremental version of this iteration is

\[
r^{k+1} \approx r^k - \gamma^k \left( F(r^k + \epsilon_i) - F(r^k - \epsilon_i) \right), \quad (4.92)
\]

where \( \epsilon_i \) 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. (4.91).
The extension to the multidimensional case, where $z$ and $r$ belong to $\mathbb{R}^n$, is straightforward. Here $p(z; r)$ is a probability distribution, whereby $z$ takes values of the form $r + \epsilon d$, where $d$ is a random vector that lies on the surface of the unit sphere, and $\epsilon$ is a scalar that takes values $\pm \epsilon_i$, $i = 1, \ldots, m$, with probabilities indicated in Fig. 4.11.4. Thus, at $r^k$ we first choose randomly a direction $d^k$ on the surface of the unit sphere of $\mathbb{R}^n$ according to the given distribution, and then change $r^k$ along the direction $d^k$ if the directional derivative $\nabla F(r^k)'d^k$ is negative, or along $-d^k$ if the directional derivative is positive. For a finite difference approximation of this iteration, we sample $z^k$ along the line \{ $r^k + \beta d^k \mid \beta \in \mathbb{R}$ \}, and similar to the iteration (4.92), we set

$$r^{k+1} = r^k - \gamma^k \left( F(r^k + \epsilon_{i,k} d^k) - F(r^k - \epsilon_{i,k} d^k) \right) d^k,$$

where $i^k$ is an index generated with probabilities that are proportional to $p_{i,k}/\epsilon_{i,k}$.

Let us also discuss the case where $p(z; r)$ is a discrete but nonsymmetric distribution, i.e., $z$ takes values of the form $r + \epsilon d$, where $d$ is a random vector that lies on the surface of the unit sphere, and $\epsilon$ is a zero mean scalar that takes values $\epsilon_i$, $i = 1, \ldots, m$. Then the analog of iteration (4.93) is

$$r^{k+1} = r^k - \gamma^k \left( F(r^k + \epsilon_i d^k) - F(r^k) \right) d^k,$$

where $i^k$ is an index generated with probabilities that are proportional to $p_{i,k}/\epsilon_{i,k}$. Thus in this case, we still require two function values per iteration. Generally, for a symmetric distribution $p(z; r)$, iteration (4.93) tends to be more accurate than iteration (4.94), and is preferred.

Algorithms of the form (4.93) and (4.94) are known as random direction search methods. They use only two cost function values per iteration, and a direction $d^k$ that need not be related to the gradient of $F$ in any way. By comparing the two function values, they determine (approximately) the sign of the directional derivative along $d^k$, and accordingly move along $d^k$ or its opposite $-d^k$.

**Likelihood-Ratio Policy Gradient Methods**

The preceding example describes one possible way to use random directions as a way of approximating the gradient method (4.89). We will now develop another sample-based random direction method, which is often better suited for our infinite horizon DP context. We assume that $p(z; r)$ is a discrete (not necessarily symmetric) distribution, and we also assume that $p(z; r)$ is differentiable with respect to $r$ for each of the values of $z$.

Then by interchanging gradient and expected value, we have

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

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

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

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

The preceding formula suggests an incremental implementation of the gradient iteration (4.89) that approximates the expected value on the right side above with a single sample. This is conceptually similar to the incremental methods of Example 4.11.5, but makes explicit use of the gradient of the sampling distribution. 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 \( \nabla \left( \log \left( p(z^k;r^k) \right) \right) \).

(c) Iterate according to
\[ r^{k+1} = r^k - \gamma^{k} \nabla \left( \log \left( p(z^k;r^k) \right) \right) F(z^k). \tag{4.95} \]

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 conveniently differentiated with respect to \( r \), and samples of \( z \) can be obtained using the distribution \( p(z;r) \).

Note that in iteration (4.95), \( r \) is adjusted along a random direction that does not involve at all the gradient of \( F \), only the gradient of the logarithm of the sampling distribution! This is similar to the methods described in Example 4.11.5.

Let us emphasize that the iteration (4.95) 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 (4.95) produces a parameter \( \pi \) in the
limit and the distribution \( p(z; \pi) \) is not atomic (i.e., it is not concentrated a single point), then a solution \( \pi \in Z \) must be extracted from \( p(z; \pi) \). In the SSP and discounted problems of this chapter, the subset \( \mathcal{P}_Z \) of parametric distributions typically contains the atomic distributions, while it can be shown that minimization over the set of all distributions \( \mathcal{P}_Z \) 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 4.11.6 (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 \( p(u | i; r) \) 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 (4.95), 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
\[
\nabla \left( \log \left( p(z^k; r^k) \right) \right).
\]
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 $p(u \mid i; r)$ is differentiable with respect to $r$. Then the logarithm that is differentiated in Eq. (4.96) can be written as

$$
\log \left( p(z^k; r^k) \right) = \log \prod_{m=0}^{\infty} p_{im_{m+1}}(u_m)p(u_m \mid i_m; r^k)
$$

$$
= \sum_{m=0}^{\infty} \log \left( p_{im_{m+1}}(u_m) \right) + \sum_{m=0}^{\infty} \log \left( p(u_m \mid i_m; r^k) \right),
$$

and its gradient (4.96), which is needed in the iteration (4.95), is given by

$$
\nabla \left( \log \left( p(z^k; r^k) \right) \right) = \sum_{m=0}^{\infty} \nabla \left( \log \left( p(u_m \mid i_m; r^k) \right) \right). \tag{4.97}
$$

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 (4.95) is very simple to implement: for the given parameter vector $r^k$, we generate a sample trajectory $z^k$ using the corresponding randomized policy $p(u \mid i; r^k)$, we calculate the corresponding sample cost $F(z^k)$, and the gradient (4.96) using the expression (4.97), and we update $r^k$ using Eq. (4.95).

Policy gradient methods for other types of DP problems can be similarly developed, including for model-free contexts where state transitions can only be generated through a simulator. 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.

**Random Search and Cross-Entropy Methods**

Let us consider a random search approach for solving the problem

$$
\min_r E\left\{ J_{\hat{\mu}(r)}(i_0) \right\},
$$
The cross-entropy method is very simple to implement, does not suffer from the fragility of gradient-based optimization, does not involve random-
ized policies, and relies on some supportive theory. In fact the method does not require the calculation of gradients, and 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 4.11.3. 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.

4.11.2 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\[†\]

\[
\min_r \sum_{s=1}^{q} \| u^s - \tilde{\mu}(i^s, r) \|^2 \tag{4.98}
\]

(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 \( \tilde{\mu} \) 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, and in Section 3.5 in the context of classification methods for approximation in policy space. In the context of artificial intelligence, it comes within the framework of supervised learning methods.\[‡\]

† 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.

‡ 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 algorithm [TeG96] performed even better. The DeepChess program by David, Netanyahu, and Wolf [DNW16] provides another example of an expert-based supervised training approach.
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

$$u^s = \arg \min_{u \in U(i^s)} \sum_{j=1}^{n} p_{ij}(u) (g(i^s, u, j) + \alpha \tilde{J}(j)), \quad \text{(4.99)}$$

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

$$u^s = \arg \min_{u \in U(i^s)} \tilde{Q}(i^s, u, r). \quad \text{(4.100)}$$

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. (4.99) or Eq. (4.100), and then applying approximation in policy space through Eq. (4.98).

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 (4.98) is to use interpolation (rather than parametric approximation). By this we mean to specify for each $i \notin \{i^1, \ldots, i^s\}$ a probability distribution $\{\phi_{i1}, \ldots, \phi_{is}\}$, and to use the policy $\tilde{\mu}$ defined by

$$\tilde{\mu}(i) = \sum_{s=1}^{q} \phi_{is} u^s, \quad i = 1, \ldots, n. \quad \text{(4.101)}$$

In general, this requires that the control constraint set is a convex subset of a Euclidean space so that the interpolated controls (4.101) 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. (4.98), or the one-step lookahead policy that is based on the approximation $\tilde{J}$ or $\tilde{Q}$, in the case of Eq. (4.99) or Eq. (4.100), 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 (4.99). This advantage is generally shared by schemes that are based on approximation in policy space.
4.11.3 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 \textit{approximate the generated policies directly instead of approximating their cost functions or Q-factors}. Looking back to Sections 4.7.2 and 4.7.3, 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 \textit{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 \((i^s, u^s), s = 1, \ldots, q\) (cf. the rollout algorithm of Section 4.6.2). We then obtain an “improved” policy \( \tilde{\mu}(i, \tau) \), using an approximation architecture (for example a neural network), where
the parameter $\varpi$ is obtained from the least squares/regression minimization

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

(possibly with added regularization); see Fig. 4.11.6. The “improved” policy $\hat{\mu}(i, \varpi)$ 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 4.11.2 [cf. Eq. (4.98)]; we just use the rollout policy as the “expert” and emulate its decisions using sampling and supervised learning.† The training problem (4.102) is similar to the ones typically solved in classification problems (often with the use of neural networks), and can be addressed with similar algorithms: the parameter $\varpi$ defines a classifier, which given a state $i$, classifies $i$ as requiring control $\hat{\mu}(i, \varpi)$; cf. Section 3.5.

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.

**Favorable Special Cases - Linear Quadratic Control Problems**

The approximate PI scheme of this section can actually produce an optimal policy in some interesting special cases. In particular, let $\mathcal{M}$ be the set of policies $\mu$ that can be represented by the approximation architecture, i.e., have the form $\mu = \hat{\mu}(r)$ for some parameter vector $r$. Suppose that $\mathcal{M}$ has the property that if a policy that belongs to $\mathcal{M}$ is used as a base policy, then the corresponding rollout policy also belongs to $\mathcal{M}$. Then it can be seen that the algorithm of this section, starting with a policy within $\mathcal{M}$ and using a large number of samples $q$ for the least squares minimization (4.102), produces a sequence of policies in $\mathcal{M}$, which is essentially the same sequence as the one produced by the exact PI algorithm.

It follows that in this favorable case, the algorithm of this section inherits the convergence properties of exact PI, without requiring a mathematical model; a simulator of the system is sufficient. This suggests that that if the set $\mathcal{M}$ “nearly contains” the policies produced by exact PI, then

† A simpler alternative to parametric approximation and the least squares minimization (4.102) is the interpolation scheme (4.101).
Sec. 4.11 Approximation in Policy Space

the policies generated by the approximate PI scheme of this section generates a sequence of policies whose performance oscillates at near optimal levels.

An important special case is when the system is linear, the cost function is infinite horizon-discounted quadratic, and $\mathcal{M}$ is the class of control laws that consist of a linear function of the state. It can then be shown (under mild assumptions, cf. [Ber12], Section 4.2) that the optimal policy belongs to $\mathcal{M}$. Moreover, exact PI, starting from a policy in $\mathcal{M}$, generates policies within $\mathcal{M}$ and converges to an optimal policy. Thus the scheme of this section has the same properties for this choice of $\mathcal{M}$, and works without requiring a model of the system.

Variations

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 (4.102) while adding new terms to the least squares sum as new samples $(i^*, u^*)$ with $u^*$ 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 $i^*$.

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 $(i^*, u^*)$ is obtained. The algorithm parallels the SARSA Q-learning algorithm given in Section 4.8.

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).
2. We compute the rollout control $u^k$ at $i^k$, using $\mu^k = \tilde{\mu}(\cdot, r^k)$ as base policy, i.e.,

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

† We have not discussed in this book infinite horizon problems with continuous state and control spaces. While such problems can be challenging in general and can exhibit counterintuitive behavior, the linear quadratic case is well behaved, and is supported by a solid analytical and computational methodology; see e.g., [Ber12]. For the purpose of this example, we simply quote results from this methodology without a proof elaboration.
(3) We update the parameter vector via

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

where $\gamma^k$ is a positive stepsize, and $\nabla \hat{\mu}(i^k, r^k)$ denotes the gradient matrix of $\hat{\mu}(i^k, \cdot)$ 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 $r$.

### 4.12 NOTES AND SOURCES

In this chapter we have provided an introduction to infinite horizon DP with a view towards approximate solution methods that are suitable for large-scale problems. We have restricted ourselves to finite-state problems with perfect state information. Infinite-state problems exhibit more complex behaviors and present many challenges for approximate DP methods. The theory of these problems is developed in several books, including the author’s [Ber12] and [Ber18a]. The latter book contains much recent advanced research on infinite-state deterministic and stochastic shortest path problems. Finite-state average cost problems are also more challenging than discounted problems. The book by Puterman [Put94] and the author’s [Ber12] contain a detailed account of average cost finite-state Markovian decision problems. Partial state information are considerably more challenging that their perfect state information counterparts, and not covered at any depth in this book. Semi-Markov problems are discussed in several textbooks; see e.g., [Ros70], [Ber12], [Ber17], [Put94].

The methods of VI and PI, and their optimistic variants are the cornerstones of infinite horizon computational DP, and they serve as the principal points of departure for approximations. Both of these methods were introduced by Bellman in his 1957 book [Bel57], and in earlier reports on which the book was based.†

Asynchronous distributed VI methods were introduced by the author [Ber82], and textbook descriptions were given in [BeT89], [Ber12], [Ber18a]. In addition to the computational topics covered in this chapter, we should mention that in addition to VI, optimistic PI can also be implemented

---

† Bellman used the term “successive approximation” for VI, and “approximation in policy space” for (nonoptimistic) PI in his book (in earlier works he also used the term “approximation in strategy space”). Of course Bellman’s analysis just introduced the basic methods, and proved their properties and their convergence in a limited number of special contexts. The methods and their variations have been discussed and analyzed extensively in subsequent works. The names “value iteration” and “policy iteration” are universally used at present.
via distributed asynchronous computation; see the author’s papers on DP and fixed point computations [Ber82], [Ber83], the paper by Williams and Baird on asynchronous PI [WiB93], and the series of papers by Bertsekas and Yu [BeY10], [BeY12], [YuB13a] on asynchronous optimistic PI and Q-learning. Generally, asynchronous and distributed algorithms are natural in computational contexts involving simulation, which by its nature is well suited to both multiprocessing and asynchronous implementation.

The variational form of Bellman’s equation (Section 4.2) has been used in various contexts, involving error bounds for value iteration, since the early days of DP theory, see e.g., [Ber12], Section 2.1.1. The variational form of Bellman’s equation is also implicit in the adaptive aggregation framework of Bertsekas and Castanon [BeC89]. In the context of RL, the variational equation has been used in various algorithmic contexts under the name *reward shaping* or *potential-based shaping* (we have used the term “cost shaping” here as we are focusing on cost minimization); see e.g., the papers by Ng, Harada, and Russell [NHR99], Wiewiora [Wie03], Asmuth, Littman, and Zinkev [ALZ08], Devlin and Kudenko [DeK11], Grzes [Grz17] for some representative works. While reward shaping does not change the optimal policies of the original DP problem, it may change significantly the suboptimal policies produced by approximate DP methods that use linear feature-based approximation. Basically, with reward shaping and a linear approximation architecture, $V$ is used as an extra feature. This is closely related with the idea of using approximate cost functions of policies as basis functions in approximation architectures; see the discussion in the neuro-dynamic programming book [BeT96], Section 3.1.4.

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 in wide use for approximation of either optimal costs or Q-factors (see e.g., Gordon [Gor95], Longstaff and Schwartz [LoS01], Ormonet and Sen [OrS02], Ernst, Geurts, and Wehenkel [EGW06], Antos, Munos, and Szepesvari [AMS07], and Munos and Szepesvari [MuS08]).

The performance bound of Props. 4.6.1 and 4.6.3 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. The approximate PI method of Section 4.7.3 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.

Q-learning (Section 4.8) 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 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 \textit{differential training}, has been proposed by the author in [Ber97b] (see also Weaver and Baxter [WeB99]).

Projected equations (Section 4.9) 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], [Tes94].
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 [BeY09b], 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], [BeY09b], [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 [BeY09b], which adapts the TD methodology to the solution of large systems of linear equations.

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 with little analysis by Schweitzer and Seidman [ScS85], and have been further developed by de Farias and Van Roy [DFV03], [DFV04], [DeF04]; see also Desai, Farias, and Moallemi [DFM12]. For a challenging application to pricing of network services, see Paschalidis and Tsitsiklis [PaT00]. For more recent work based on variants of the linear programming formulation, see Cogill et al. [CRV06], Wang, O’Donoghue, and Boyd [WOB15], and Beuchat, Warrington, and Lygeros [BWL19], and the references quoted there. Approximation based on linear programming, while not discussed at length in this book, is a promising approach that deserves further attention.

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]. An important paper in this context is the one by Williams [Wil92], who proposed among others the likelihood-ratio policy gradient method given here. The methods of [Wil92] are commonly referred to as REINFORCE in the literature (see e.g., [SuB18], Ch. 13).

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. Actor-critic algorithms that are suitable for POMDP and involve gradient estimation have been given by Yu [Yu05], and Estanjini, Li, and Paschalidis [ELP12].

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 variance reduction methods (Greensmith, Bartlett, and Baxter [GBB04], Greensmith [Gre05]), and scaling based on the so-called natural gradient (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 [Rec18a], [Rec18b].

The cross-entropy method was initially developed in the context of rare event simulation and was later adapted for use in optimization. For textbook accounts, see Rubinstein and Kroese [RuK04], [RuK13], [RuK16], 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 in an approximate DP context by Mannor, Rubinstein, and Gat [MRG03]. It was applied with success to the game of tetris by Szita and Lorinz [SzL06], and Thiery and Scherrer [ThS09]. For recent analysis, see Joseph and Bhatnagar [JoB16], [JoB18].

The expert training methods of Section 4.11.2 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; see e.g., Abbeel and Ng [AbN04], Neu and Szepesvari [NeS12], and Schaal [Sch99]. For a survey of expert training methods in the context of robotics, see Argall et al. [ACV09], and for some recent work, see Ben Amor et al. [BVE13] and Lee
Sec. 4.13 Appendix: Mathematical Analysis

For a recent analysis of related possibilities, see Hanawal et al. [HLZ18].

The PI/rollout methods of Section 4.11.3, depending on their implementation, are similar to several methods proposed in the literature, first by Lagoudakis and Parr [LaP03], and then by several other authors (see e.g., Dimitrakakis and Lagoudakis [DiL08], Lazaric, Ghavamzadeh, and Munos [LGM10], Gabillon et al. [GLG11], Liu and Wei [LiW14], Farahmand et al. [FPB15], and the references quoted there). The paper by Yan, Diaconis, Rusmevichientong, and Van Roy [YDR04] developed a different type of PI/rollout method based on recursive application of rollout policies for a version of solitaire. The main strength of the PI/rollout methods is that they rely on the reliability and robustness of the rollout approach, which has been demonstrated by many practical studies, combined with the use of well-developed classification algorithms. Note also that the generation of samples of the rollout policies, while computationally intensive, can benefit greatly from the availability of parallel computation. Still, the approximation in policy space of the rollout policies is subject to exploration issues, which, while different than those discussed in the context of other approximate PI methods, are quite substantial.

We have noted in Section 4.11.3 the application of PI with approximation in policy space for adaptive control in problems with a linear-quadratic structure. An alternative simulation-based PI method based on approximation in value space has been proposed by Bradtke, Ydstie, and Barto [BYB94]. In its idealized form (i.e., with an infinite number of simulation samples, and no simulation errors), it obtains the optimal policy without knowledge of the linear system equation parameters. Related methods for adaptive control of discrete and continuous-time systems have been discussed in the books by Vrabie, Vamvoudakis, and Lewis [VVL13], Jiang and Jiang [JiJ17], and Liu et al. [LWW17].

4.13 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.,

\[ TJ \geq T J', \quad T \mu J \geq T \mu J', \quad \text{for all } J \text{ and } J' \text{ with } J \geq J'. \]

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 \( TJ \) and \( T \mu J \) are also increased uniformly by the constant \( ac \).

### 4.13.1 Proofs for Stochastic Shortest Path Problems

We provide the proofs of Props. 4.2.1-4.2.5 from Section 4.2. A key insight for the analysis is that the expected cost incurred within an \( m \)-stage block vanishes exponentially as the start of the block moves forward (here \( m \) is the integer specified by Assumption 4.2.1, i.e., the termination state can be reached within \( m \) steps with positive probability from every starting state). In particular, the cost in the \( m \) stages between \( km \) and \( (k+1)m - 1 \) is bounded in absolute value by \( \rho^k C \), where

\[
C = m \max_{k=1,\ldots,n} \max_{j=1,\ldots,n} \max_{u \in U(i)} |g(i, u, j)|.
\]

Thus, we have

\[
|J_{\pi}(i)| \leq \sum_{k=0}^{\infty} \rho^k C = \frac{1}{1-\rho} C.
\]

This shows that the “tail” of the cost series,

\[
\sum_{k=mK}^{\infty} E\{g(x_k, \mu_k, w_k)\},
\]

vanishes as \( K \) increases to \( \infty \), since the probability that \( x_{mK} \neq t \) decreases like \( \rho^K \) [cf. Eq. (4.6)]. Intuitively, since the “tail” of the cost series can be neglected as \( K \to \infty \), it is valid to take the limit in the finite horizon DP algorithm, and obtain the infinite horizon Bellman equation and VI convergence. Mathematically, this is the essence of the following proofs.

**Proposition 4.2.1:** (Convergence of VI) The optimal cost \( J^*(i) \) is finite for each \( i \). Moreover, given any initial conditions \( J_0(1), \ldots, J_0(n) \), the sequence \( \{J_k(i)\} \) generated by the VI algorithm

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

converges to \( J^*(i) \) for each \( i = 1, \ldots, n \).
Sec. 4.13 Appendix: Mathematical Analysis

Proof: From Eq. (4.104), we see that \( J^*(i) \) is finite. For every \( K > 0 \), \( x_0 \), and \( \pi = \{ \mu_0, \mu_1, \ldots \} \), we break down the cost \( J_\pi(x_0) \) into the portions incurred over the first \( mK \) stages and over the remaining stages:

\[
J_\pi(x_0) = \lim_{N \to \infty} E \left\{ \sum_{k=0}^{N-1} g(x_k, \mu_k(x_k), w_k) \right\}
= E \left\{ \sum_{k=0}^{mK-1} g(x_k, \mu_k(x_k), w_k) \right\} + \lim_{N \to \infty} E \left\{ \sum_{k=mK}^{N-1} g(x_k, \mu_k(x_k), w_k) \right\}.
\]

The expected cost during the \( K \)th \( m \)-stage cycle [stages \( Km \) to \( (K+1)m-1 \)] is upper bounded by \( C \rho^K \) [cf. Eqs. (4.6) and (4.104)], so that

\[
\left| \lim_{N \to \infty} E \left\{ \sum_{k=mK}^{N-1} g(x_k, \mu_k(x_k), w_k) \right\} \right| \leq C \sum_{k=K}^{\infty} \rho^k = \frac{\rho^K C}{1 - \rho}.
\]

Also, denoting \( J_0(t) = 0 \), let us view \( J_0 \) as a terminal cost function and bound its expected value under \( \pi \) after \( mK \) stages. We have

\[
\left| E\{J_0(x_{mK})\} \right| = \sum_{i=1}^{n} P(x_{mK} = i \mid x_0, \pi)J_0(i)
\leq \left( \sum_{i=1}^{n} P(x_{mK} = i \mid x_0, \pi) \right) \max_{i=1, \ldots, n} |J_0(i)|
\leq \rho^K \max_{i=1, \ldots, n} |J_0(i)|,
\]

since the probability that \( x_{mK} \neq t \) is less or equal to \( \rho^K \) for any policy. Combining the preceding relations, we obtain

\[
-\rho^K \max_{i=1, \ldots, n} |J_0(i)| + J_\pi(x_0) - \frac{\rho^K C}{1 - \rho}
\leq E \left\{ J_0(x_{mK}) + \sum_{k=0}^{mK-1} g(x_k, \mu_k(x_k), w_k) \right\}
\leq \rho^K \max_{i=1, \ldots, n} |J_0(i)| + J_\pi(x_0) + \frac{\rho^K C}{1 - \rho}.
\]

Note that the expected value in the middle term of the above inequalities is the \( mK \)-stage cost of policy \( \pi \) starting from state \( x_0 \), with a terminal cost \( J_0(x_{mK}) \); the minimum of this cost over all \( \pi \) is equal to the value \( J_{mK}(x_0) \), which is generated by the DP recursion (4.105) after \( mK \) iterations. Thus, by taking the minimum over \( \pi \) in Eq. (4.106), we obtain for all \( x_0 \) and \( K \),

\[
-\rho^K \max_{i=1, \ldots, n} |J_0(i)| + J^*(x_0) - \frac{\rho^K C}{1 - \rho}
\leq J_{mK}(x_0)
\leq \rho^K \max_{i=1, \ldots, n} |J_0(i)| + J^*(x_0) + \frac{\rho^K C}{1 - \rho}.
\]
and by taking the limit as $K \to \infty$, we obtain
\[
\lim_{K \to \infty} J_{mK}(x_0) = J^*(x_0)
\]
for all $x_0$. It can be seen that for all $\ell = 1, \ldots, m$, we have
\[
|J_{mK+\ell}(x_0) - J_{mK}(x_0)| \leq \rho^K \left( C + \max_{i=1,\ldots,n} J_0(i) - \min_{i=1,\ldots,n} J_0(i) \right),
\]
implying that $\lim_{K \to \infty} J_{mK+\ell}(x_0)$ is the same for all $\ell = 1, \ldots, m$, so that
\[
\lim_{k \to \infty} J_k(x_0) = J^*(x_0).
\]
Q.E.D.

**Proposition 4.2.2: (Bellman’s Equation)** The optimal cost function $J^* = (J^*(1), \ldots, J^*(n))$ satisfies for all $i = 1, \ldots, n$, the equation
\[
J^*(i) = \min_{u \in U(i)} \left[ p_{it}(u) g(i, u, t) + \sum_{j=1}^{n} p_{ij}(u) (g(i, u, j) + J^*(j)) \right],
\]
and in fact it is the unique solution of this equation.

**Proof:** By taking the limit as $k \to \infty$ in the DP iteration (4.105) and using the result of Prop. 4.2.1, we see that $J^*(1), \ldots, J^*(n)$ satisfy Bellman’s equation (we are using here the fact that the limit and minimization operations commute when the minimization is over a finite number of alternatives). To show uniqueness, observe that if $J^*(1), \ldots, J^*(n)$ satisfy Bellman’s equation, then the DP iteration (4.105) starting from $J^*(1), \ldots, J^*(n)$ just replicates $J^*(1), \ldots, J^*(n)$. It follows from the convergence result of Prop. 4.2.1 that $J^*(i) = J^*(i)$ for all $i$. Q.E.D.

**Proposition 4.2.3: (VI and Bellman’s Equation for Policies)** For any stationary policy $\mu$, the corresponding cost function $J_\mu = (J_\mu(1), \ldots, J_\mu(n))$ satisfies for all $i = 1, \ldots, n$ the equation
\[
J_\mu(i) = p_{it}(\mu(i)) g(i, \mu(i), t) + \sum_{j=1}^{n} p_{ij}(\mu(i)) \left( g(i, \mu(i), j) + J_\mu(j) \right).
\]
and is in fact the unique solution of this equation. Furthermore, given any initial conditions $J_0(1), \ldots, J_0(n)$, the sequence $\{J_k(i)\}$ generated by the VI algorithm that is specific to $\mu$,

$J_{k+1}(i) = p_{it}(\mu(i))g(i, \mu(i), t) + \sum_{j=1}^{n} p_{ij}(\mu(i))\left(g(i, \mu(i), j) + J_k(j)\right)$,

converges to the cost $J_\mu(i)$ for each $i$.

**Proof:** Given $\mu$, consider a modified stochastic shortest path problem, which is the same as the original except that the control constraint set contains only one element for each state $i$, the control $\mu(i)$; i.e., the control constraint set is $\tilde{U}(i) = \{\mu(i)\}$ instead of $U(i)$. From Prop. 4.2.2, $J_\mu$ solves uniquely Bellman’s equation for this modified problem, i.e., for all $i$,

$J_\mu(i) = p_{it}(\mu(i))g(i, \mu(i), t) + \sum_{j=1}^{n} p_{ij}(\mu(i))\left(g(i, \mu(i), j) + J_\mu(j)\right)$.

From Prop. 4.2.1, the VI algorithm converges to $J_\mu(i)$. **Q.E.D.**

**Proposition 4.2.3 and this equation imply that $J_\mu(i) = J^*(i)$ for all $i$. Conversely, if $J_\mu(i) = J^*(i)$ for all $i$, Props. 4.2.2 and 4.2.3 imply this equation. **Q.E.D.**
Proposition 4.2.5: (Contraction Property of the Bellman Operator) The Bellman operator $T$ of Eqs. (4.8) is a contraction mapping with respect to some weighted norm

$$\|J\| = \max_{i=1,\ldots,n} \frac{|J(i)|}{v(i)},$$

defined by some vector $v = (v(1), \ldots, v(n))$ with positive components. In other words, there exists a positive scalar $\rho < 1$ such that for any two $n$-dimensional vectors $J$ and $J'$, we have

$$\|TJ - TJ'\| \leq \rho \|J - J'\|.$$

The same is true for the Bellman operator $T_\mu$ (4.9) of any policy $\mu$.

Proof: We first define the vector $v$ using the problem of Example 4.2.1. In particular, we let $v(i)$ be the maximal expected number of steps to termination starting from state $i$. From Bellman’s equation in Example 4.2.1, we have for all $i = 1, \ldots, n$, and stationary policies $\mu$,

$$v(i) = 1 + \max_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u)v(j) \geq 1 + \sum_{j=1}^{n} p_{ij}(\mu(i))v(j), \quad i = 1, \ldots, n.$$

Thus we obtain for all $\mu$,

$$\sum_{j=1}^{n} p_{ij}(\mu(i))v(j) \leq v(i) - 1 \leq \rho v(i), \quad i = 1, \ldots, n, \quad (4.107)$$

where $\rho$ is defined by

$$\rho = \max_{i=1,\ldots,n} \frac{v(i) - 1}{v(i)}.$$ 

Since $v(i) \geq 1$ for all $i$, we have $\rho < 1$.

We will now show that Eq. (4.107) implies the desired contraction property. Indeed, consider the operator $T_\mu$, which when applied to a vector $J = (J(1), \ldots, J(n))$ produces the vector $T_\mu J = ((T_\mu J)(1), \ldots, (T_\mu J)(n))$ defined by

$$(T_\mu J)(i) = p_{i\mu}(\mu(i))g(i, \mu(i), t) + \sum_{j=1}^{n} p_{ij}(\mu(i)) \left( g(i, \mu(i), j) + J(j) \right),$$
for all \( i = 1, \ldots, n \). We have for all \( J, J' \), and \( i \)
\[
(T_\mu J)(i) = (T_\mu J')(i) + \sum_{j=1}^{n} p_{ij}(\mu(i))(J(j) - J'(j))
\]
\[
= (T_\mu J')(i) + \sum_{j=1}^{n} p_{ij}(\mu(i))v(j)\frac{(J(j) - J'(j))}{v(j)}
\]
\[
\leq (T_\mu J')(i) + \sum_{j=1}^{n} p_{ij}(\mu(i))v(j)\|J - J'\|
\]
\[
\leq (T_\mu J')(i) + \rho v(i)\|J - J'\|
\]
where the last inequality follows from Eq. (4.107). By minimizing both sides over all \( \mu(i) \in U(i) \), we obtain
\[
(T J)(i) \leq (T J')(i) + \rho v(i)\|J - J'\|,
\]
Thus we have
\[
\frac{(T J)(i) - (T J')(i)}{v(i)} \leq \rho \|J - J'\|,
\]
Similarly, by reversing the roles of \( J \) and \( J' \), we obtain
\[
\frac{(T J')(i) - (T J)(i)}{v(i)} \leq \rho \|J - J'\|,
\]
By combining the preceding two inequalities, we have
\[
\frac{|(T J)(i) - (T J')(i)|}{v(i)} \leq \rho \|J - J'\|,
\]
and by maximizing the left-hand side over \( i \), the contraction property \( \|T J - T J'\| \leq \rho \|J - J'\| \) follows. **Q.E.D.**

### 4.13.2 Proofs for Discounted Problems

Since we have shown that the discounted problem can be converted to the equivalent SSP problem of Fig. 4.3.1, we can apply Props. 4.2.1-4.2.4. Then Props. 4.3.1-4.3.4 are obtained from the construction of Fig. 4.3.1. The contraction property of Prop. 4.3.5 can also be proved in the same way, since in the SSP problem of Fig. 4.3.1, the expected number of steps to terminate starting from a state \( i \neq t \) can be obtained as the mean of a geometrically distributed random variable with parameter \( 1 - \alpha \):
\[
v(i) = 1 \cdot (1 - \alpha) + 2 \cdot \alpha(1 - \alpha) + 3 \cdot \alpha^2(1 - \alpha) + \cdots = \frac{1}{1 - \alpha}, \quad i = 1, \ldots, n.
\]
so that the modulus of contraction is
\[
\rho = \frac{v(i) - 1}{v(i)} = \alpha.
\]
Thus by applying Prop. 4.2.5, we obtain Prop. 4.3.5. Note that there is similar contraction property for \( T_\mu \).
4.13.3 Convergence of Exact and Optimistic Policy Iteration

We provide a proof of the convergence of exact PI for the case of a discounted problem. The proof for the SSP problem is similar.

**Proposition 4.5.1: (Convergence of Exact PI)** For both the SSP and the discounted problems, the exact PI algorithm generates an improving sequence of policies [i.e., \( J_{\mu_{k+1}}(i) \leq J_{\mu_k}(i) \) for all \( i \) and \( k \)] and terminates with an optimal policy.

**Proof:** For any \( k \), consider the sequence generated by the VI algorithm for policy \( \mu^{k+1} \):

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

where \( N = 0, 1, \ldots, \) and

\[
J_0(i) = J_{\mu_k}(i), \quad i = 1, \ldots, n.
\]

From Eqs. (4.25) and (4.26), we have

\[
J_0(i) = \sum_{j=1}^{n} p_{ij} \left( \mu^k(i) \right) \left( g(i, \mu^k(i), j) + \alpha J_0(j) \right)
\geq \sum_{j=1}^{n} p_{ij} \left( \mu^{k+1}(i) \right) \left( g(i, \mu^{k+1}(i), j) + \alpha J_0(j) \right)
= J_1(i),
\]

for all \( i \). By using the above inequality we obtain

\[
J_1(i) = \sum_{j=1}^{n} p_{ij} \left( \mu^{k+1}(i) \right) \left( g(i, \mu^{k+1}(i), j) + \alpha J_0(j) \right)
\geq \sum_{j=1}^{n} p_{ij} \left( \mu^{k+1}(i) \right) \left( g(i, \mu^{k+1}(i), j) + \alpha J_1(j) \right)
= J_2(i),
\]

for all \( i \), and by continuing similarly we have

\[
J_0(i) \geq J_1(i) \geq \cdots \geq J_N(i) \geq J_{N+1}(i) \geq \cdots, \quad i = 1, \ldots, n. \quad (4.108)
\]
Since by Prop. 4.3.3, \( J_N(i) \to J_{\mu^{k+1}}(i) \), we obtain \( J_0(i) \geq J_{\mu^{k+1}}(i) \) or
\[
J_{\mu^k}(i) \geq J_{\mu^{k+1}}(i), \quad i = 1, \ldots, n, \quad k = 0, 1, \ldots.
\]
Thus the sequence of generated policies is improving, and since the number of stationary policies is finite, we must after a finite number of iterations, say \( k + 1 \), obtain \( J_{\mu^k}(i) = J_{\mu^{k+1}}(i) \) for all \( i \). Then we will have equality throughout in Eq. (4.108), which means that
\[
J_{\mu^k}(i) = \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u) \left( g(i, u, j) + \alpha J_{\mu^k}(j) \right), \quad i = 1, \ldots, n.
\]
Thus the costs \( J_{\mu^k}(1), \ldots, J_{\mu^k}(n) \) solve Bellman’s equation, and by Prop. 4.3.2, it follows that \( J_{\mu^k}(i) = J^*(i) \) and that \( \mu^k \) is optimal. \textbf{Q.E.D.}

We provide a proof of convergence of optimistic PI for discounted problems.

\begin{center}
\textbf{Proposition 4.5.2: (Convergence of Optimistic PI)} For the discounted problem, the sequences \( \{J_k\} \) and \( \{\mu^k\} \) generated by the optimistic PI algorithm satisfy
\[
J_k \to J^*, \quad J_{\mu^k} \to J^*.
\]
\end{center}

\begin{proof}
First we choose a scalar \( r \) such that the vector \( \bar{J}_0 \) defined by \( \bar{J}_0 = J_0 + re \), satisfies \( TJ_0 \leq \bar{J}_0 \) [here and later, \( e \) is the unit vector, i.e., \( e(i) = 1 \) for all \( i \)]. This can be done since if \( r \) is such that \( TJ_0 - J_0 \leq (1 - \alpha)re \), we have
\[
T\bar{J}_0 = TJ_0 + \alpha re \leq J_0 + re = \bar{J}_0,
\]
where \( e = (1, 1, \ldots, 1)' \) is the unit vector.

With \( \bar{J}_0 \) so chosen, define for all \( k, \bar{J}_{k+1} = T_{\mu^k} \bar{J}_k \). Then since we have
\[
T(J + re) = TJ + \alpha re, \quad T_{\mu}(J + re) = T_{\mu}J + \alpha re
\]
for any \( J \) and \( \mu \), it can be seen by induction that for all \( k \) and \( m = 0, 1, \ldots, m_k \), the vectors \( J_{k+1} = T_{\mu^k} J_k \) and \( \bar{J}_{k+1} = T_{\mu^k} \bar{J}_k \) differ by a multiple of the unit vector, namely
\[
r \alpha^{m_0 + \cdots + m_{k-1} + m} e.
\]
It follows that if \( J_0 \) is replaced by \( \bar{J}_0 \) as the starting vector in the algorithm, the same sequence of policies \( \{\mu^k\} \) will be obtained; i.e., for all \( k \), we have \( T_{\mu^k} \bar{J}_k = T_{\mu^k} \bar{J}_k \). Moreover, we have \( \lim_{k \to \infty} (\bar{J}_k - J_k) = 0 \).
Next we will show that $J^* \leq \tilde{J}_k \leq T^k \tilde{J}_0$ for all $k$, from which convergence will follow. Indeed, we have $T_{\mu^0} \tilde{J}_0 = T \tilde{J}_0 \leq \tilde{J}_0$, from which we obtain

$$T_{\mu^0}^m \tilde{J}_0 \leq T_{\mu^0}^{m-1} \tilde{J}_0, \quad m = 1, 2, \ldots,$$

so that

$$T_{\mu^1} \tilde{J}_1 = T \tilde{J}_1 \leq T_{\mu^0} \tilde{J}_1 = T_{\mu^0}^{m_0+1} \tilde{J}_0 \leq T_{\mu^0} \tilde{J}_0 = \tilde{J}_1 \leq T_{\mu^0} \tilde{J}_0 = T \tilde{J}_0.$$

This argument can be continued to show that for all $k$, we have $\tilde{J}_k \leq T \tilde{J}_{k-1}$, so that

$$\tilde{J}_k \leq T^k \tilde{J}_0, \quad k = 0, 1, \ldots,$$

On the other hand, since $T \tilde{J}_0 \leq \tilde{J}_0$, we have $J^* \leq \tilde{J}_0$, and it follows that successive application of any number of operators of the form $T_{\mu}$ to $\tilde{J}_0$ produces functions that are bounded from below by $J^*$. Thus,

$$J^* \leq \tilde{J}_k \leq T^k \tilde{J}_0, \quad k = 0, 1, \ldots,$$

By taking the limit as $k \to \infty$, we obtain $\lim_{k \to \infty} \tilde{J}_k(i) = J^*(i)$ for all $i$, and since $\lim_{k \to \infty} (\tilde{J}_k - \tilde{J}_k) = 0$, we obtain

$$\lim_{k \to \infty} J_k(i) = J^*(i), \quad i = 1, \ldots, n.$$

Finally, from the finiteness of the state and control spaces, it follows that there exists $\epsilon > 0$ such that if $\max_i |J(i) - J^*(i)| \leq \epsilon$ and $T_{\mu} J = T J$, so that $\mu$ is optimal. Since $J_k \to J^*$, this shows that $\mu^k$ is optimal for all sufficiently large $k$. Q.E.D.

4.13.4 Performance Bounds for One-Step Lookahead

We first prove the basic performance bounds for $\ell$-step lookahead schemes and discounted problems.
Proposition 4.6.1: (Limited Lookahead Performance Bounds)

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

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

(4.109)

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

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

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

(4.110)

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

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

(4.111)

Then

$$J_{\tilde{\mu}}(i) \leq \tilde{J}(i) + \frac{c}{1 - \alpha}, \quad i = 1, \ldots, n.$$  

(4.112)

Proof: (a) The line of proof of this part is essentially to show Eq. (4.109) 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 $k$,

$$\|T_{\tilde{\mu}}^{k} J^* - J^*\| \leq \sum_{m=1}^{k} \|T_{\tilde{\mu}}^{m} J^* - T_{\tilde{\mu}}^{m-1} J^*\| \leq \sum_{m=1}^{k} \alpha^{m-1} \|T_{\tilde{\mu}} J^* - J^*\|.$$  

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

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

(4.113)

Denote now $\tilde{J} = T^{\ell-1}\tilde{J}$. By using the triangle inequality and the fact $T_{\tilde{\mu}} \tilde{J} = T \tilde{J}$ (which follows from the definition of $\tilde{\mu}$), the rightmost
expression of Eq. (4.113) is estimated as follows:
\[ \|T_{\hat{\mu}} J^* - J\| \leq \|T_{\hat{\mu}} J^* - T_{\hat{\mu}} \hat{J}\| + \|T_{\hat{\mu}} \hat{J} - T \hat{J}\| + \|T \hat{J} - J^*\| \]
\[ = 2\alpha \|\hat{J} - J^*\| \]
\[ = 2\alpha \|T^{(t-1)} \hat{J} - T^{(t-1)} J^*\| \]
\[ \leq 2\alpha \|\hat{J} - J^*\|. \]

By combining the preceding two relations, we obtain Eq. (4.109).

(b) Let us denote by $e$ the unit vector whose components are all equal to 1. Then by the assumption (4.111), we have
\[ T_{\hat{\mu}} \hat{J} = \hat{J} \leq \hat{J} + ce. \]

Applying $T_{\hat{\mu}}$ to both sides of this relation, and using the monotonicity and constant shift property of $T_{\hat{\mu}}$, we obtain
\[ T_{\hat{\mu}}^2 \hat{J} \leq T_{\hat{\mu}} \hat{J} + c\alpha e. \]

Continuing similarly, we have,
\[ T_{\hat{\mu}}^{k+1} \hat{J} \leq T_{\hat{\mu}}^k \hat{J} + k \alpha e, \quad k = 0, 1, \ldots. \]

Adding these relations, we obtain
\[ T_{\hat{\mu}}^{k+1} \hat{J} \leq \hat{J} + (1 + \alpha + \cdots + \alpha^k) c\alpha e, \quad k = 0, 1, \ldots. \]

Taking the limit as $k \to \infty$, and using the fact $T_{\hat{\mu}}^{k+1} \hat{J} \to J_{\hat{\mu}}$, we obtain the desired inequality (4.112). \textbf{Q.E.D.}

4.13.5 Performance Bounds for Rollout

We next show the basic cost improvement property of rollout.

\begin{proposition}
(Cost Improvement by Rollout) Let $\hat{\mu}$ be the rollout policy obtained by the one-step lookahead minimization
\[ \min_{u \in U(i)} \sum_{j=1}^n p_{ij}(u)(g(i, u, j) + \alpha J_{\mu(j)}), \]
where $\mu$ is a base policy [cf. Eq. (4.110) with $\hat{J} = J_{\mu}$] and we assume that $\mu(i) \in \overline{U}(i) \subset U(i)$ for all $i = 1, \ldots, n$. Then $J_{\hat{\mu}} \leq J_{\mu}$.
\end{proposition}
Proof: Let us denote

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

We have for all \( i = 1, \ldots, n \),

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

where the equality on the right holds by Bellman’s equation. Hence the hypothesis of Prop. 4.6.1(b) holds with \( c = 0 \), and the result follows from the bound (4.112). Q.E.D.

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

**Proposition 4.6.3: (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 of Eq. (4.14), 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

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

(4.114)

Then

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

(4.115)
(c) Assume that for some constant \( c \), \( \tilde{J} \) and \( \mu \) satisfy
\[
\| \tilde{J} - J_\mu \| \leq c. \tag{4.116}
\]
Then
\[
J_\hat{\mu}(i) \leq J_\mu(i) + \frac{2c}{1 - \alpha}, \quad i = 1, \ldots, n.
\]

**Proof:** (a) This is simply Prop. 4.6.1(a) adapted to the truncated rollout scheme [\( \tilde{J} \) is replaced in Eq. (4.109) by \( T_\mu^m \tilde{J} \)].

(b) We first prove the result for the case where \( c = 0 \). Then the condition (4.114) can be written as \( \tilde{J} \geq T_\mu \tilde{J} \), from which by using the monotonicity of \( T \) and \( T_\mu \), we have
\[
\tilde{J} \geq T_\mu^m \tilde{J} \geq T T_\mu^m \tilde{J} \geq T T T_\mu^m \tilde{J} = T_\mu T T T_\mu^m \tilde{J}, \tag{4.117}
\]
so that
\[
\tilde{J} \geq T T T T T T T_\mu^m \tilde{J}.
\]
This relation and the monotonicity of \( T_\mu \), imply that \( \{ T_\mu^k T T T_\mu^m \tilde{J} \} \) is monotonically nonincreasing as \( k \) increases, and is bounded above by \( \tilde{J} \).

Since by the convergence property of VI, the sequence converges to \( J_\hat{\mu} \) as \( k \to \infty \), the result follows (for the case \( c = 0 \)).

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 (4.114) can be written in terms of \( J' \) as
\[
\sum_{j=1}^{n} p_{ij} (\mu(i)) \left[ g(i, \mu(i), j) + \alpha J'(j) - \frac{\alpha c}{1 - \alpha} \right] \leq J'(i) - \frac{c}{1 - \alpha} + c,
\]
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 \( \hat{\mu} \), we can replace \( \tilde{J} \) with \( J' \), without changing \( \hat{\mu} \). Then by using the version of the result already proved for \( c = 0 \), we have \( J_\hat{\mu} \leq J' \), which is equivalent to the desired relation (4.115).
(c) We prove this part as a special case of part (b). The hypothesis (4.116) implies that for all $i$ we have

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

$$= J(\mu) + \alpha c$$

$$\leq \bar{J}(i) + c + \alpha c.$$

Part (b) then shows that

$$J(\mu) \leq \bar{J}(i) + \frac{c + \alpha c}{1 - \alpha} \leq J(\mu) + c + \frac{c + \alpha c}{1 - \alpha} = J(\mu) + \frac{2c}{1 - \alpha}$$

for all $i$. Q.E.D.

The preceding proof allows a relaxation of the condition (4.114). For the relation (4.117) to hold it is sufficient that $\bar{J}$ and $\mu$ satisfy the condition

$$\bar{J} \geq T^{m} \bar{J} \geq T^{m+1} \bar{J},$$

or the even weaker condition

$$\bar{J} \geq T^{m} \bar{J} \geq TT^{m} \bar{J}.$$

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

$$\bar{J} \geq T \bar{J},$$

and it implies the bound $J(\mu) \leq \bar{J}$. The proof is based on Eq. (4.117) where $m$ is taken to be zero. In domain-specific contexts, the preceding conditions may be translated into meaningful results.

### 4.13.6 Performance Bounds for Approximate Policy Iteration

To prove the performance bound of Prop. 4.6.4, we focus on the discounted problem, and we make use of the contraction property of $T_{\mu}$:

$$\|T_{\mu} J - T_{\mu} J'\| \leq \alpha \|J - J'\|, \quad \|T J - T J'\| \leq \alpha \|J - J'\|,$$

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

We want to prove the following performance bound.
Proposition 4.6.4: (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 (4.44) and the approximate policy improvement (4.45). 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 4.13.1: Consider the discounted problem, and let \( J, \tilde{\mu}, \) and \( \mu \) satisfy
\[
\| J - J_\mu \| \leq \delta, \quad \| T_{\tilde{\mu}} J - T J \| \leq \epsilon,
\]
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
\[
\| T_{\tilde{\mu}} J_\mu - T_{\tilde{\mu}} J \| \leq \alpha \delta, \quad \| T J - T J_\mu \| \leq \alpha \delta,
\]
and hence
\[
T_{\tilde{\mu}} J_\mu \leq T_{\tilde{\mu}} J + \alpha \delta e, \quad T J \leq T J_\mu + \alpha \delta e,
\]
where \( e \) is the unit vector, i.e., \( e(i) = 1 \) for all \( i \). Using also Eq. (4.118), we have
\[
T_{\tilde{\mu}} J_\mu \leq T_{\tilde{\mu}} J + \alpha \delta e \leq T J + (\epsilon + \alpha \delta) e \leq T J_\mu + (\epsilon + 2\alpha) e. \tag{4.120}
\]
Combining this inequality with \( T J_\mu \leq T_\mu J_\mu = J_\mu \), we obtain
\[
T_{\tilde{\mu}} J_\mu \leq J_\mu + (\epsilon + 2\alpha \delta) e. \tag{4.121}
\]
We will show that this relation implies that
\[
J_{\tilde{\mu}} \leq J_\mu + \frac{\epsilon + 2\alpha \delta}{1 - \alpha} e. \tag{4.122}
\]
Indeed, by applying \( T_{\tilde{\mu}} \) to both sides of Eq. (4.121), we obtain
\[
T_{\tilde{\mu}} J_{\tilde{\mu}} \leq T_{\tilde{\mu}} J_\mu + \alpha (\epsilon + 2\alpha \delta) e \leq J_\mu + (1 + \alpha)(\epsilon + 2\alpha \delta) e.
\]
Applying $T_{\hat{\mu}}$ again to both sides of this relation, and continuing similarly, we have for all $k$,  

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

By taking the limit as $k \to \infty$, and by using the VI convergence property $T_{\hat{\mu}}^k J_{\mu} \to J_{\hat{\mu}}$, we obtain Eq. (4.122).

Using now the contraction property of $T_{\hat{\mu}}$ and Eq. (4.122), we have  

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

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

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

(4.123)

Also from the contraction property of $T$,  

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

which, in conjunction with Eq. (4.120), yields  

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

Combining this relation with Eq. (4.123), we obtain  

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

which is equivalent to the desired relation (4.119). Q.E.D.

**Proof of Prop. 4.6.4:** Applying Lemma 4.13.1, 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 4.6.5: (Performance Bound for Approximate PI when Policies Converge) Let $\hat{\mu}$ be a policy generated by the approximate PI algorithm under conditions (4.44), (4.45), and (4.46). Then we have

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

Proof: Let $\bar{J}$ be the cost vector obtained by approximate policy evaluation of $\hat{\mu}$. Then in view of Eqs. (4.44), (4.45), we have

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

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

$$\|TJ_{\hat{\mu}} - J_{\hat{\mu}}\| \leq \|TJ_{\hat{\mu}} - T\bar{J}\| + \|T\bar{J} - T_{\hat{\mu}}\bar{J}\| + \|T_{\hat{\mu}}\bar{J} - T_{\hat{\mu}}J_{\hat{\mu}}\|$$

$$= \|TJ_{\hat{\mu}} - T\bar{J}\| + \|T\bar{J} - T_{\hat{\mu}}\bar{J}\| + \|T_{\hat{\mu}}\bar{J} - T_{\hat{\mu}}J_{\hat{\mu}}\|$$

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

$$\leq \epsilon + 2\alpha \delta.$$ 

(4.124)

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

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

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

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

Combining this relation with Eq. (4.124), we obtain the desired performance bound. Q.E.D.