5

Infinite Horizon Reinforcement Learning

Contents

In this chapter, we consider the use of approximate DP/RL methods for suboptimal solution of the infinite horizon SSP and discounted problems of the preceding chapter. In particular, we will consider approximate versions of the fundamental value iteration (VI) and policy iteration (PI) algorithms. In the process, we will make frequent references to the DP operators T and T_μ (or Bellman operators), which map an n-dimensional vector J into the n-dimensional vectors TJ and $T_\mu J$, and provide shorthand notation for algorithms and analysis. For the purpose of easy reference, we state these operators here:

For SSP problems:

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

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

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

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

For discounted problems:

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

and

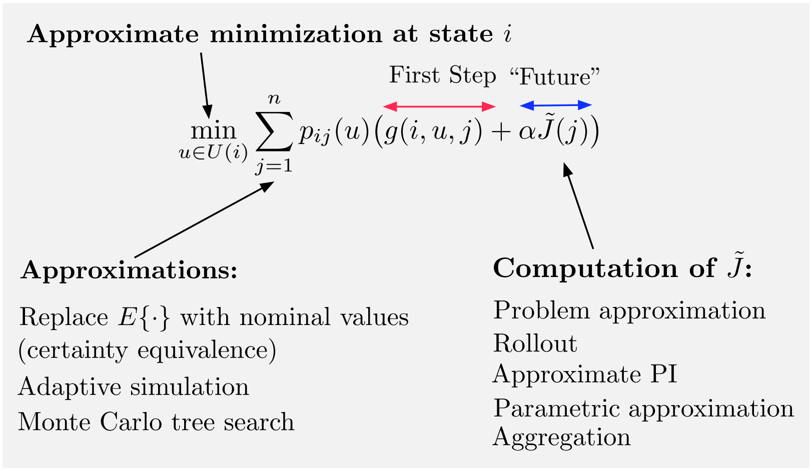
$$(T_\mu J)(i) = \sum_{j=1}^{n} p_{ij}\big(\mu(i)\big)\Big(g\big(i,\mu(i),j\big) + \alpha J(j)\Big), \quad (5.4)$$

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

5.1 APPROXIMATION IN VALUE SPACE - PERFORMANCE BOUNDS

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

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



Approximate minimization at state i
First Step
"Future"
$\min_{u\in U(i)} \sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j)+\alpha\tilde J(j)\big)$
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 5.1.1 Schematic illustration of various options for approximation in value space with one-step lookahead in infinite horizon problems. The lookahead function values $\tilde J(j)$ approximate the optimal cost-to-go values $J^*(j)$, and can be computed by a variety of methods. There may be additional approximations in the minimization over u_k and the computation of the expected value.

see Fig. 5.1.1.

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

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

where $\hat J$ is some approximation of J^*. Thus $\tilde J$ is the result of a single value iteration starting from $\hat J$. Policies with lookahead of more than two stages are similarly defined. *The "effective one-step" cost approximation $\tilde J$ in ℓ-step lookahead is the result of $\ell-1$ successive value iterations starting from some initial approximation $\hat J$.* Otherwise expressed, *ℓ-step lookahead with $\hat J$ at the end is the same as one-step lookahead with $T^{\ell-1}\hat J$ at the end*, where T is the Bellman operator [cf. Eq. (5.3)].

Types of Approximation in Value Space Schemes

In Chapter 2 we described several types of limited lookahead schemes, where $\tilde J$ is obtained in different ways, such as problem approximation, rollout, and others. Some of these schemes can be fruitfully adapted to infinite horizon problems; see Fig. 5.1.1. For example, the problem approximation approaches of Section 2.3 admit straightforward extensions to infinite horizon settings, whereby the function $\tilde J(j)$ in Eq. (5.5) is obtained by solving exactly an infinite horizon (or even finite horizon) problem that is related to the original in some way. Aggregation is another possible approximation approach, which will be discussed in Chapter 6.

Imperfect state observation problems pose special challenges in the case of an infinite horizon. Such problems can be reformulated to ones involving perfect observation of the belief state, which, however, is infinite dimensional (cf. Section 1.3.6).† Problem approximation based on forms of certainty equivalence is particularly interesting within this context, because a natural approximating problem is often evident. For example, the lookahead function $\tilde{J}$ may be derived by solving a perfect state information variant of the original problem, where an estimate of the system state is used as if it were exact. The perfect state information variant may be tractable, because it may be deterministic or it may involve a modest number of states.

In this chapter, the approximation in value space schemes that we will focus on in Sections 5.1-5.5 are primarily based on approximate (optimistic) PI and operate as follows, starting with an initial policy μ^0:

(a) We generate (usually off-line) several policies $\mu^0, \mu^1, \ldots, \mu^m$.

(b) We evaluate each policy μ^k approximately, with a cost function $\tilde{J}_{\mu^k}$, often by using a parametric approximation/neural network approach, and possibly including the use of truncated rollout.

(c) We generate the next policy μ^{k+1} using one-step or multistep policy improvement based on $\tilde{J}_{\mu^k}$.

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

We will view rollout as a simple form of approximate PI, which involves a single policy iteration, executed with the aid of simulation. The rollout may be truncated and supplemented with a (potentially sophisticated) terminal cost function approximation (cf. Section 2.4). We will give some performance bounds for limited lookahead schemes and for rollout in Sections 5.1.1 and 5.1.2, respectively. We will discuss performance bounds for the general approximate PI scheme in Section 5.1.3.

5.1.1 Limited Lookahead

We will now consider performance bounds for ℓ-step lookahead. In particular, for a given state i_0, let $\hat{\mu}_0, \ldots, \hat{\mu}_{\ell-1}$ attain the minimum in the ℓ-step

† Because of the infinite dimensionality of the belief space, the theory of Chapter 4 has to be extended before it can be applied, since it was developed for finite state space. Usually this is fairly straightforward for discounted problems, but less so for SSP problems. We will not provide further discussion in this book.

lookahead minimization

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

We focus on the suboptimal policy that applies control $\tilde{\mu}(i_0) = \hat{\mu}_0(i_0)$, and we refer to $\tilde{\mu}$ as the *ℓ-step lookahead policy corresponding to $\tilde{J}$*. Equivalently, in the shorthand notation of the Bellman operators T and $T_{\tilde{\mu}}$ of Eqs. (5.3) and (5.4), the ℓ-step lookahead policy $\tilde{\mu}$ is defined by

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

We will derive a bound for the performance of $\tilde{\mu}$ in part (a) of the following proposition, proved in the appendix.

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 $\overline{U}(i) \subset U(i)$. Thus, the control $\tilde{\mu}(i)$ used in this scheme is one that attains the minimum in the expression

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

This is attractive when by using some heuristic, we can identify a subset $\overline{U}(i)$ of promising controls, and to save computation, we restrict attention to this subset in the one-step lookahead minimization.

Proposition 5.1.1: (Limited Lookahead Performance Bounds)

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

$$\|J_{\tilde{\mu}} - J^*\| \le \frac{2\alpha^\ell}{1-\alpha}\|\tilde{J} - J^*\|, \tag{5.6}$$

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

(b) Define

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

where $\overline{U}(i) \subset U(i)$ for all $i = 1,\ldots,n$, and let $\tilde{\mu}$ be the one-step lookahead policy obtained by minimization in the right side of this equation. Then we have

$$J_{\tilde{\mu}}(i) \le \tilde{J}(i) + \frac{c}{1-\alpha}, \qquad i = 1, \ldots, n, \tag{5.8}$$

where

$$c = \max_{i=1,\ldots,n} \big(\hat{J}(i) - \tilde{J}(i)\big).$$

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

$$\min_{\beta \in \Re} \max_{i=1,\ldots,n} \big|\tilde{J}(i) + \beta - J^*(i)\big|. \tag{5.9}$$

Another interesting point is that it is sufficient to take maximum in the preceding expression over just the states i that can occur following the ℓ steps of lookahead, thus potentially improving the bound of Eq. (5.6) even further. This can be used to obtain sharper versions of Prop. 5.1.1(a), as well as the related subsequent Prop. 5.1.3(a), although we will not pursue this analysis further.

The bound (5.6) suggests that performance is improved when the length ℓ of the lookahead is increased, and also when the lookahead cost approximation $\tilde{J}$ is closer to the optimal cost J^* (when modified with an optimal constant shift β). Both of these conclusions are intuitive and also consistent with practical experience. Note that we are not guaranteeing that multistep lookahead will lead to better performance than one-step lookahead; we know that this is not necessarily true (cf. Example 2.2.1). It is the performance *bound* that is improved with multistep lookahead.

Regarding the bound (5.8), we note that it guarantees that when $c \le 0$, the cost $J_{\tilde{\mu}}$ of the one-step lookahead policy is no larger than $\tilde{J}$. The case $c = 0$ is equivalent to $\hat{J} \le \tilde{J}$, which resembles the consistent improvement condition for deterministic rollout methods (cf. Section 2.4.1). If $\tilde{J} = J_\mu$ for some policy μ with $\mu(i) \in \overline{U}(i)$ for all i (as in the case of the pure form of rollout to be discussed in Section 5.1.2), then $c = 0$, and from Eq. (5.8) it follows that we have cost improvement, i.e., $J_{\tilde{\mu}} \le J_\mu$.

Unfortunately, the bound (5.6) is not very reassuring when α is close to 1. Nonetheless, the following example shows that the bound can be tight even 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 $O\big(\epsilon/(1-\alpha)\big)$ difference in policy costs (through discounted accumulation over an infinite number of steps), yet it can be "nullified" in Bellman's equation by an $O(\epsilon)$ difference between J^* and $\tilde{J}$.

Example 5.1.1

Consider the two-state discounted problem shown in Fig. 5.1.2, where ϵ is a positive scalar and $\alpha \in [0, 1)$ is the discount factor. Here there are two control

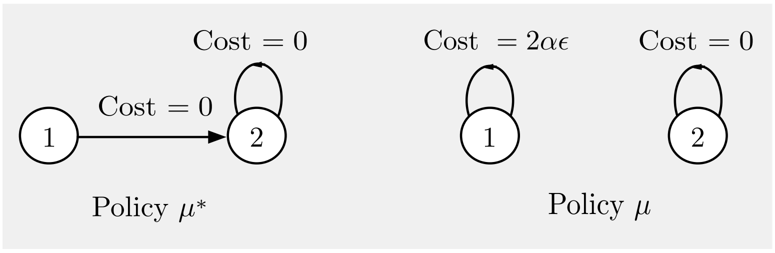


Figure 5.1.2 A two-state problem for proving the tightness of the performance bound of Prop. 5.1.1(b) (cf. Example 5.1.1). All transitions are deterministic as shown. At state 1 there are two possible decisions: move to state 2 at cost 0 (policy μ^*) or stay at state 1 at cost $2\alpha\epsilon$ (policy μ).

choices at state 1: move to state 2 at cost 0 (policy μ^*) or stay at state 1 at cost $2\alpha\epsilon$ (policy μ). The optimal policy is μ^*, and the optimal cost-to-go function is $J^*(1) = J^*(2) = 0$. Consider the cost function approximation $\tilde J$

$$\tilde J(1) = -\epsilon, \qquad \tilde J(2) = \epsilon,$$

so that

$$\|\tilde J - J^*\| = \epsilon.$$

The policy μ that decides to stay at state 1 is a one-step lookahead policy based on $\tilde J$, because

$$2\alpha\epsilon + \alpha\tilde J(1) = \alpha\epsilon = 0 + \alpha\tilde J(2).$$

Moreover, we have

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

so the bound of Eq. (5.6) holds with equality when $\ell = 1$.

5.1.2 Rollout and Approximate Policy Improvement

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

An important fact is that in the pure form of rollout, *the rollout policy improves over the base policy*, consistent with the finite horizon case; cf. Section 2.4. This is shown by the following proposition [a special case of Prop. 5.1.1(b) as noted earlier], and it is to be expected since rollout is one-step PI, so the general policy improvement property of PI applies. A related result is given as Lemma 5.9.1 in the appendix (Section 5.9.3).

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

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

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

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, is similar to its finite horizon counterpart; cf. Section 2.4.1.

Example 5.1.2 (Rollout with Multiple Heuristics)

Let $\mu_1,\ldots,\mu_M$ be stationary policies, let

$$\tilde{J}(i)=\min\big\{J_{\mu_1}(i),\ldots,J_{\mu_M}(i)\big\},\qquad i=1,\ldots,n,$$

let $\overline{U}(i)\subset U(i)$, and assume that

$$\mu_1(i),\ldots,\mu_M(i)\in \overline{U}(i),\qquad i=1,\ldots,n.$$

Then, for all i and $m=1,\ldots,M$, we have

$$\begin{aligned}\hat{J}(i)&=\min_{u\in \overline{U}(i)}\sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j)+\alpha\tilde{J}(j)\big)\\&\le \min_{u\in \overline{U}(i)}\sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j)+\alpha J_{\mu_m}(j)\big)\\&\le \sum_{j=1}^{n} p_{ij}\big(\mu_m(i)\big)\Big(g\big(i,\mu_m(i),j\big)+\alpha J_{\mu_m}(j)\Big)\\&=J_{\mu_m}(i),\end{aligned}$$

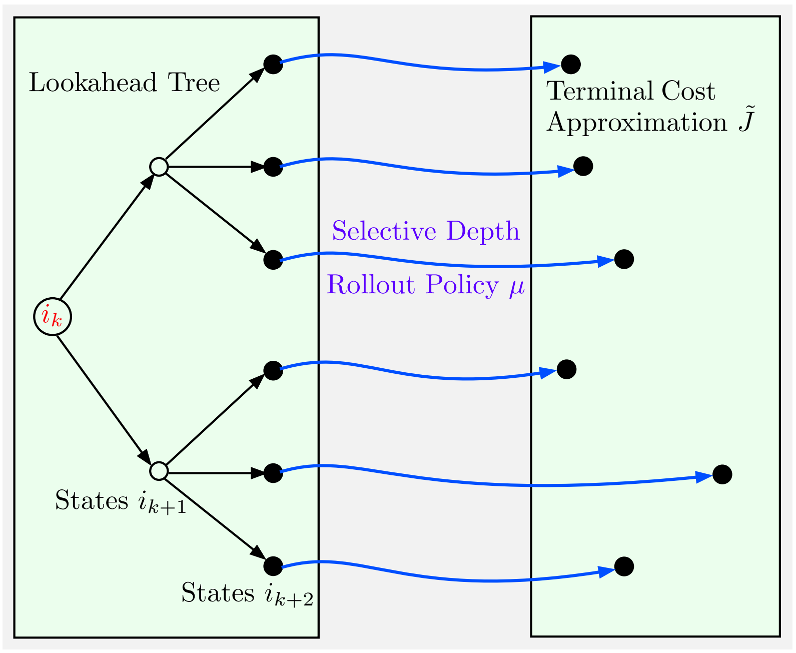


Figure 5.1.3 Illustration of two-step lookahead, rollout with a policy μ for a limited and state-dependent number of steps, and a terminal cost function approximation $\tilde{J}$. A Monte Carlo tree search scheme may also be used for multistep lookahead; cf. Section 2.4.2.

which, by taking minimum of the right-hand side over m, yields

$$\hat{J}(i) \leq \tilde{J}(i), \qquad i = 1, \ldots, n.$$

Using Prop. 5.1.1(b), we see that the rollout policy $\tilde{\mu}$, obtained by using $\tilde{J}$ as one-step lookahead approximation satisfies

$$J_{\tilde{\mu}}(i) \leq \tilde{J}(i) = \min\big\{J_{\mu_1}(i), \ldots, J_{\mu_M}(i)\big\}, \qquad i = 1, \ldots, n,$$

i.e., it improves over each of the policies $\mu_1, \ldots, \mu_M$.

Truncated Rollout with Multistep Lookahead and Terminal Cost Function Approximation

Let us next discuss a truncated variant of the rollout approach, whereby we use ℓ-step lookahead, we then apply rollout with policy μ for a limited number of steps, and finally we approximate the cost of the remaining steps using some terminal cost approximation $\tilde{J}$; see Fig. 5.1.3 for a case where $\ell = 2$. We can view this form of rollout as *a single optimistic policy iteration* *combined with multistep lookahead* (the iteration is optimistic because it evaluates μ with m value iterations starting from $\tilde{J}$). 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.

Note that the three components of this rollout scheme (multistep lookahead, rollout with μ, and cost approximation $\tilde{J}$) can be designed independently of each other. Moreover, while the multistep lookahead is implemented on-line, μ and $\tilde{J}$ should ordinarily be available from an earlier off-line computation.

The following proposition generalizes the performance bounds given for limited lookahead (cf. Props. 5.1.1). In particular, part (a) of the proposition follows by applying Prop. 5.1.1(a), since the truncated rollout scheme of this section can be viewed as ℓ-step approximation in value space with terminal cost function $T_\mu^m \tilde{J}$ at the end of the lookahead, where T_μ is the Bellman operator corresponding to μ.

Proposition 5.1.3: (Performance Bounds for Truncated Rollout with Terminal Cost Function Approximation) Let ℓ and m be positive integers, let μ be a policy, and let $\tilde{J}$ be a function of the state. Consider a truncated rollout scheme consisting of ℓ-step lookahead, followed by rollout with a policy μ 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

$$\|J_{\tilde{\mu}} - J^*\| \leq \frac{2\alpha^\ell}{1-\alpha}\|T_\mu^m \tilde{J} - J^*\|, \tag{5.10}$$

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

(b) We have

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

where

$$c = \max_{i=1,\ldots,n} \bigl((T_\mu \tilde{J})(i) - \tilde{J}(i)\bigr).$$

(c) We have

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

Some of the insights provided by the preceding proposition are:

(a) Part (a) of the proposition implies that as the size of lookahead ℓ increases, the bound on the performance of the rollout policy also improves. Moreover, assuming that μ is close to optimal (so that $T_\mu^m \tilde{J}$ is close to J^* as $m \to \infty$), the bound on the performance of the rollout policy $\tilde{\mu}$ improves as m increases (the improvement is enhanced if $\tilde{J}$ is also close to J_μ).†

(b) Part (c) suggests that if $\tilde{J}$ is close to J_μ, the performance of the rollout policy $\tilde{\mu}$ is nearly improved relative to the performance of the base policy μ. This is consistent with the cost improvement property of rollout, cf. Prop. 5.1.2. Part (b) admits a similar interpretation.

In summary, the guidelines for truncated rollout that these results suggest are to *choose as large lookahead as practical, and choose $\tilde{J}$ as close to J_μ or J^* as possible.* It is not clear to what extent increasing m, the length of rollout of μ, improves performance, and some examples suggest that m should not be chosen to be very large. A small value of m is also beneficial in another way: it limits the amount of computation needed for rollout, and reduces the variance of the cost estimates. In practice, for an infinite horizon problem, m is ordinarily chosen in some empirical fashion.

Regarding 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_μ as a terminal cost function approximation. Note that a good choice of terminal cost approximation is crucial for some types of SSP problems where most or all of the cost is incurred upon reaching the termination state (for example winning or losing a game). Note also that once cost function approximation is introduced at the end of the rollout, the cost improvement property of the rollout policy over the base policy may be lost [cf. Prop. 5.1.3(c)].

The truncated rollout scheme of Fig. 5.1.3 has been adopted in the rollout backgammon algorithm of Tesauro and Galperin [TeG96]. The policy μ and the terminal cost function approximation $\tilde{J}$ were 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(λ). A similar algorithm (with Monte Carlo tree search instead of ℓ-step lookahead) was used in the AlphaGo program (Silver et al. [SHM16]), with μ and $\tilde{J}$ obtained using an approximate PI scheme and a deep neural network.

† Note that $\tilde{\mu}$ is unaffected by a constant shift in $\tilde{J}$, so that an optimized constant β may be added to $\tilde{J}$ in the bound (5.10) [cf. Eq. (5.9)].

5.1.3 Approximate Policy Iteration

When the number of states is very large, the policy evaluation step and/or the policy improvement step of the PI method may be implementable only through approximations. In an approximate PI scheme, each policy μ^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 μ^{k+1} is generated by (perhaps approximate) policy improvement based on $\tilde{J}_{\mu^k}$.

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

$$\max_{i=1,\ldots,n} \big|\tilde{J}_{\mu^k}(i) - J_{\mu^k}(i)\big| \leq \delta, \tag{5.11}$$

and a policy improvement error satisfying

$$\max_{i=1,\ldots,n} \Bigg| \sum_{j=1}^{n} p_{ij}\big(\mu^{k+1}(i)\big)\big(g(i,\mu^{k+1}(i),j) + \alpha\tilde{J}_{\mu^k}(j)\big) - \min_{u\in U(i)} \sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j) + \alpha\tilde{J}_{\mu^k}(j)\big)\Bigg| \leq \epsilon, \tag{5.12}$$

where δ and ϵ are some nonnegative scalars. Here ϵ includes the simulation errors, plus any additional errors due to function approximation. Also δ is a measure of the accuracy of the lookahead minimization in the policy improvement operation (in many cases $\delta = 0$).

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

Proposition 5.1.4: (Performance Bound for Approximate PI) Consider the discounted problem, and let $\{\mu^k\}$ be the sequence generated by the approximate PI algorithm defined by the approximate policy evaluation (5.11) and the approximate policy improvement (5.12). Then the policy error

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

becomes less or equal to

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

asymptotically as $k \to \infty$.

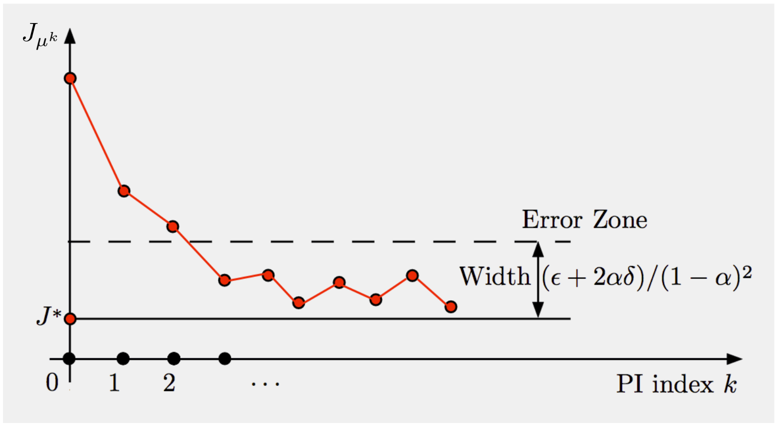


Figure 5.1.4 Illustration of typical behavior of approximate PI. In the early iterations, the method tends to make rapid and fairly monotonic progress, until J_{μ^k} gets within an error zone of size less than

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

After that J_{μ^k} oscillates randomly within that zone. The figure is oversimplified since it shows the difference $J_{\mu^k} - J^*$ at a single state. For different states the nature of the error oscillation may be different.

The preceding performance bound is important, because it is in qualitative agreement with the empirical behavior of approximate PI. Typically, in the beginning, the method tends to make rapid and fairly monotonic progress, but eventually it gets into an oscillatory pattern. This happens after J_{μ^k} gets within an error zone of size

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

or smaller, and then J_{μ^k} oscillates fairly randomly within that zone; see Fig. 5.1.4. In practice, the error bound of Prop. 5.1.4 tends to be pessimistic, so the zone of oscillation is usually much narrower than what is suggested by the bound. However, the bound itself can be proved to be tight, in worst case. This is shown with an example in the book [BeT96], Section 6.2.3. Note that the bound of Prop. 5.1.4 also holds in the case of infinite state and control spaces discounted problems, when there are infinitely many policies (see [Ber18a], Prop. 2.4.3).

Performance Bound for the Case Where Policies Converge

Generally, the policy sequence $\{\mu^k\}$ generated by approximate PI may oscillate between several policies, as noted earlier. However, under some

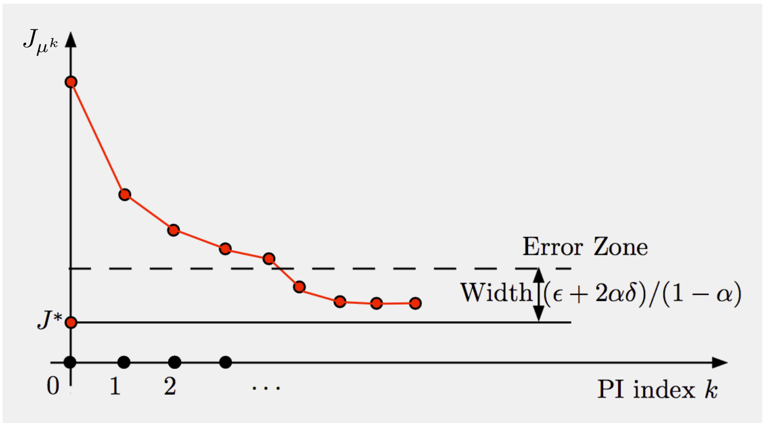


Figure 5.1.5 Illustration of typical behavior of approximate PI when policies converge. The method tends to make monotonic progress, and J_{μ^k} converges within an error zone of size less than $(\epsilon + 2\alpha\delta)/(1-\alpha)$.

circumstances the sequence will converge to some policy $\tilde{\mu}$, i.e.,

$$\mu^{\overline{k}+1} = \mu^{\overline{k}} = \tilde{\mu} \qquad \text{for some } \overline{k}. \tag{5.13}$$

An important case where this happens is aggregation methods, which will be discussed in Chapter 6. In this case the behavior of the method is more regular, and we can show the following bound, which is more favorable than the one of Prop. 5.1.4 by a factor $1/(1-\alpha)$, as illustrated in Fig. 5.1.5. For the proof, see the appendix (or the original source [BeT96], Section 6.2.2).

Proposition 5.1.5: (Performance Bound for Approximate PI when Policies Converge) Let $\tilde{\mu}$ be a policy generated by the approximate PI algorithm under conditions (5.11), (5.12), and (5.13). Then we have

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

We finally note that related performance bounds hold for optimistic PI methods, where the policy evaluation is performed with just a few approximate value iterations (cf. Section 4.6.2). These bounds are similar to the ones of the nonoptimistic method, and do not suggest superiority of one type of PI method over the other. Their derivation is quite complicated; see [Ber12], Chapter 2, or [Ber18a], Section 2.5.2, and the end-of-chapter references. Moreover, there are versions of the preceding bounds, which apply to more general abstract DP problems, such as the semi-Markov problems of Section 4.4 and others; see [Ber18a], Section 2.4.1.

5.2 FITTED VALUE ITERATION

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

$$J_{k+1}(i) = \min_{u\in U(i)} \left[p_{it}(u)g(i,u,t) + \sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j) + J_k(j)\big) \right], \quad (5.14)$$

and its discounted version

$$J_{k+1}(i) = \min_{u\in U(i)} \sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j) + \alpha J_k(j)\big). \quad (5.15)$$

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

Unfortunately, when the number of states is large, the iterations (5.14) and (5.15) may be prohibitively time consuming. This motivates an approximate version of VI, which is patterned after the least squares regression/fitted VI scheme of Section 3.3 for finite horizon problems.

We start with some initial approximation to J^*, call it $\tilde J_0$. Then we generate a sequence $\{\tilde J_k\}$, where $\tilde J_{k+1}$ is equal to the exact value iterate $T\tilde J_k$ plus some error [we are using here the shorthand notation for the Bellman operator T given in Eqs. (5.1) and (5.3)]. Assuming that values $(T\tilde J_k)(i)$ may be generated for sample states i, we may obtain $\tilde J_{k+1}$ by some form of least squares regression. We will now discuss how the error $(\tilde J_k - J^*)$ is affected by this type of approximation process.

Error Bounds and Pathologies of Fitted Value Iteration

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

$$\max_{i=1,\ldots,n} \big|\tilde J_k(i) - J^*(i)\big|, \quad (5.16)$$

and the *policy error*

$$\max_{i=1,\ldots,n} \big|J_{\tilde\mu^k}(i) - J^*(i)\big|, \quad (5.17)$$

where the policy $\tilde\mu^k$ is obtained from the one-step lookahead minimization

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

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

$$\max_{i=1,\ldots,n} \left| \tilde{J}_{k+1}(i) - \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j) + \alpha \tilde{J}_k(j)\big) \right| \leq \delta. \tag{5.18}$$

It is then possible to show that asymptotically, as $k \to \infty$, the cost error (5.16) becomes less or equal to $\delta/(1-\alpha)$, while the policy error (5.17) becomes less or equal to $2\delta/(1-\alpha)^2$.

Such error bounds are given in Section 6.5.3 of the book [BeT96] (see also Prop. 2.5.3 of [Ber12]), but it is important to note that the condition (5.18) may not be satisfied by the natural least squares regression/fitted VI scheme of Section 3.3. This is illustrated by the following example due to [TsV96] (see also [BeT96], Section 6.5.3), which shows that the errors from successive approximate value iterations can accumulate to the point where the condition (5.18) cannot be maintained (with a scalar δ that is independent of k), and the approximate value iterates $\tilde{J}_k$ can grow unbounded.

Example 5.2.1 (Error Amplification in Approximate Value Iteration)

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

$$J(1) = \alpha J(2), \qquad J(2) = \alpha J(2),$$

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

$$J_{k+1}(1) = \alpha J_k(2), \qquad J_{k+1}(2) = \alpha J_k(2).$$

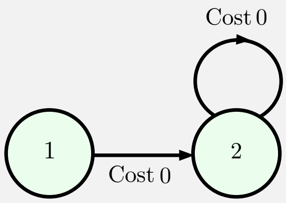
We consider a VI approach that approximates cost functions within the one-dimensional subspace of linear functions $S = \big\{(r, 2r) \mid r : \text{scalar}\big\}$; this is a favorable choice since the optimal cost function $J^* = (0,0)$ belongs to S. We use a weighted least squares regression scheme.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

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

$$T\tilde{J}_k = \big(\alpha \tilde{J}_k(2), \alpha \tilde{J}_k(2)\big) = (2\alpha r_k, 2\alpha r_k).$$

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

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



Bellman Eq: $J(1) = \alpha J(2),\ J(2) = \alpha J(2)$

$J^*(1) = J^*(2) = 0$

Exact VI: $J_{k+1}(1) = \alpha J_k(2),\ J_{k+1}(2) = \alpha J_k(2)$

Figure 5.2.1 Illustration of the discounted problem of Example 5.2.1. There are two states, 1 and 2, and a single policy. The transitions are deterministic: from state 1 to state 2, and from state 2 to state 2. These transitions are also cost-free.

or equivalently

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

To perform the preceding minimization, we set to zero the derivative with respect to r of the quadratic cost function, and obtain after some calculation

$$r_{k+1} = \alpha\zeta r_k \quad \text{where} \quad \zeta = \frac{2(\xi_1 + 2\xi_2)}{\xi_1 + 4\xi_2} > 1. \tag{5.19}$$

Thus if ξ_1 and ξ_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 α in the range $(5/6, 1)$; see Fig. 5.2.2.

The difficulty here is that the approximate VI operator that generates $\tilde J_{k+1}$ by a weighted least squares-based approximation of $T\tilde J_k$ is not a contraction (even though T itself is a contraction). At the same time there is no δ such that the condition (5.18) is satisfied for all k, because of error amplification in each approximate VI.

The preceding example indicates that the choice of the least squares weights is important in determining the success of least squares-based approximate VI schemes. Generally, in regression-based parametric architecture training schemes of the type discussed in Section 3.1.2, the weights are related to the way samples are collected: the weight ξ_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

$$\tilde J_k(i) = \tilde J(i, r_k), \qquad i = 1, \ldots, n,$$

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

$$\tilde J_{k+1}(i) = \tilde J(i, r_{k+1}), \qquad i = 1, \ldots, n,$$

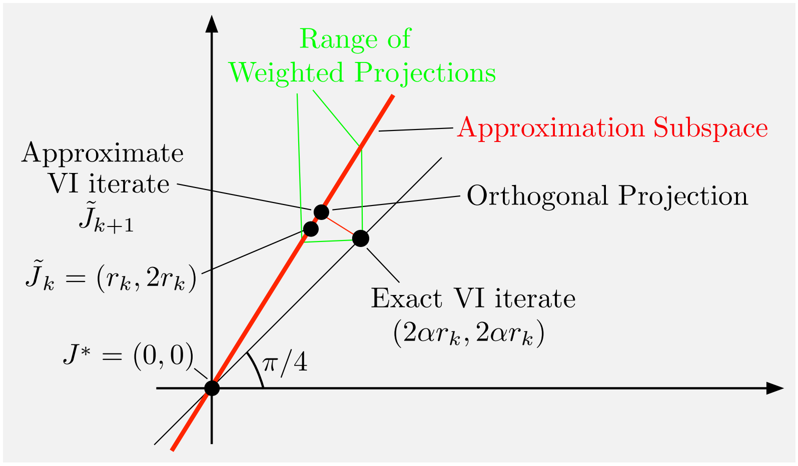

Range of Weighted Projections
Approximation Subspace
Approximate VI iterate $\tilde{J}_{k+1}$
Orthogonal Projection
$\tilde{J}_k = (r_k, 2r_k)$
Exact VI iterate $(2\alpha r_k, 2\alpha r_k)$
$J^* = (0,0)$
$\pi/4$

Figure 5.2.2 Illustration of Example 5.2.1. The iterates of approximate VI lie on the approximation subspace, which is the line $\big\{(r, 2r) \mid r \in \Re\big\}$. Given an iterate $\tilde{J}_k = (r_k, 2r_k)$, the next exact VI iterate is

$$\big(\alpha \tilde{J}_k(2), \alpha \tilde{J}_k(2)\big) = (2\alpha r_k, 2\alpha r_k).$$

This exact VI iterate is approximated by $\tilde{J}_{k+1}$ obtained by least squares regression, which can be viewed as weighted projection onto the line $\{(r, 2r) \mid r \in \Re\}$, and depends on the weights (ξ_1, ξ_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 α sufficiently close to 1, $\tilde{J}_{k+1}$ is further away from $J^* = (0,0)$ than $\tilde{J}_k$. The difficulty here is that the mapping that consists of an exact VI followed by weighted projection onto the line $\{(r, 2r) \mid r \in \Re\}$ need not be a contraction.

is obtained by the minimization

$$r_{k+1} \in \arg\min_r \sum_{s=1}^{q} \big(\tilde{J}(i^s, r) - \beta^s\big)^2, \tag{5.20}$$

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

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

The critical question now is how to select the sample states i^s, $s = 1, \ldots, q$, to guarantee that the iterates r_k remain bounded, so that a condition of the form (5.18) is satisfied and the instability illustrated with Example 5.2.1 is avoided. It turns out that there is no known general method to guarantee this in infinite horizon problems. However, some practical methods have been developed. One such method is to weigh each state according to its "long-term importance," i.e., proportionally to the number of its occurrences over a long trajectory under a "good" heuristic policy.† To implement this, we may run the system with the heuristic policy starting from a number of representative states, wait for some time for the system to approach steady-state, and record the generated states i^s, $s = 1, \ldots, q$, to be used in the regression scheme (5.20). 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; see the discussion of a related algorithm, called *projected value iteration*, in the book [Ber12], Section 6.3, and the end-of-chapter references.

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

5.3 SIMULATION-BASED POLICY ITERATION WITH PARAMETRIC APPROXIMATION

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

5.3.1 Self-Learning and Actor-Critic 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*.

A PI-type method is often called an *actor-critic method* if it involves approximations in both its actor and its critic portions. The term "actor-critic" actually applies more generally, to methods beyond PI-type, such as for example policy gradient methods to be discussed later.

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

† In the preceding Example 5.2.1, weighing the two states according to their "long-term importance" would choose ξ_2 to be much larger than ξ_1, since state 2 is "much more important," in the sense that it occurs almost exclusively in system trajectories. Indeed, from Eq. (5.19) it can be seen that when the ratio ξ_1/ξ_2 is close enough to 0, the scalar ζ is close enough to 1, making the scalar $\alpha\zeta$ strictly less than 1, and guaranteeing convergence of $\tilde{J}_k$ to J^*.

(a) *Evaluate the current policy μ^k (critic)*: Here algorithm, system, and simulator are merged in one, and the system "observes itself" by generating simulation cost samples under the policy μ^k. It then combines these samples to "learn" an approximate 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 feature-based 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, with cost function approximation used to correct for the small number of samples. In effect, the policy evaluation of the preceding policy is used as the starting iterate for an optimistic evaluation of the current policy (cf. Section 4.6.2).

(b) *Improve the current policy μ^k (actor)*: Given the approximate policy evaluation $\tilde{J}_{\mu^k}$, we generate or "learn" the new policy μ^{k+1} through the lookahead minimization

$$\mu^{k+1}(i) \in \arg\min_{u\in U(i)} \sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j)+\alpha\tilde{J}_{\mu^k}(j)\big), \qquad i=1,\ldots,n, \tag{5.21}$$

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

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

These are the sample values of the improved policy μ^{k+1} at the sample states i^s. They are generalized to "learn" a complete policy μ^{k+1} by using some parametric approximation in policy space scheme (cf. Section 2.1.5). This scheme *may be incremental* in the sense that the improved policy may be updated using very few sample pairs (i^s,u^s).

The preceding two operations are sequentially performed up to the point where some form of "convergence" occurs. To summarize, actor-critic PI methods 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, and aiming to evaluate better the cost function of the current policy (while keeping that policy unchanged).

(b) *Actor step*: The improved policy is computed when needed at states of interest via the lookahead minimization (5.21) (or its multistep version). Alternatively, a form of incremental least squares regression is

† Actor-critic methods also arise in other, non-PI contexts, which combine parametric approximation in both policy and value space, and involve gradient descent-type of parameter adjustments. We will briefly discuss such methods in Section 5.7.

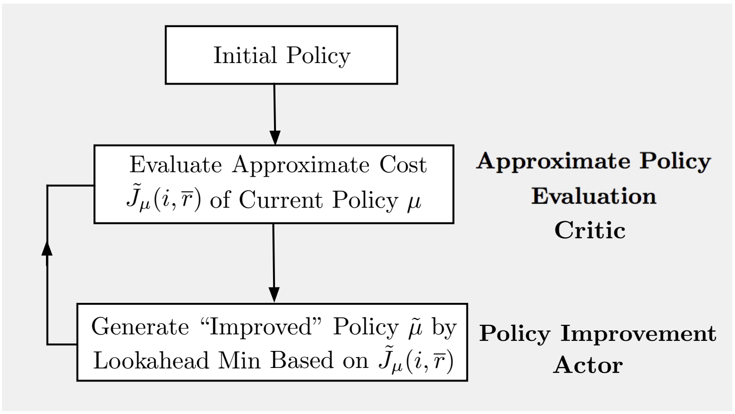

Initial Policy
Evaluate Approximate Cost $\tilde J_\mu(i,\bar r)$ of Current Policy μ
Generate "Improved" Policy $\tilde\mu$ by Lookahead Min Based on $\tilde J_\mu(i,\bar r)$
Approximate Policy Evaluation Critic
Policy Improvement Actor

Figure 5.3.1 Block diagram of model-based approximate PI for cost functions.

performed based on (many or few) state-control sample pairs, aiming to improve a parametric form of 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, in Section 5.4 (Q-learning, SARSA, and DQN), and in Section 5.5 (temporal differences), we will focus on *critic-only* PI methods, where the policy improvement steps are "exact" in the sense that they are performed with Eq. (5.21). Actor-critic methods, as well as *actor-only* methods that involve approximations only in the actor step, will be discussed further in Section 5.7.

5.3.2 Model-Based Variant of a Critic-Only Method

We will first provide an example of a model-based approximate PI implementation and then discuss its model-free version in Section 5.3.3. In particular, we assume that the cost per stage $g(i,u,j)$ and transition probabilities $p_{ij}(u)$ are available, and that the cost function J_μ of any given policy μ is approximated using a parametric architecture $\tilde J_\mu(i,r)$.

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

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

$$\overline{r} \in \arg\min_{r} \sum_{s=1}^{q} \big(\tilde{J}_\mu(i^s, r) - \beta^s\big)^2. \tag{5.22}$$

The scalar β^s is a sample cost corresponding to i^s and μ.

In particular, β^s is generated by starting at i^s, simulating a state-control trajectory using μ 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_μ. 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 policy preceding μ. Note that *this is similar to the cost function approximation implicitly used in optimistic policy iteration*; cf. Section 4.6.2. It is also possible to simplify the method by using $\hat{J}(i_N) = 0$, or obtaining $\hat{J}$ via a problem approximation process.†

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

$$r^{k+1} = r^k - \gamma^k \nabla \tilde{J}(i^{s_k}, r^k)\big(\tilde{J}(i^{s_k}, r^k) - \beta^{s_k}\big),$$

where γ^k is a stepsize, (i^{s_k}, β^{s_k}) is the state-cost sample pair that is used at the kth iteration, and r^0 is an initial parameter guess. Here the architecture $\tilde{J}(i, r)$ may be linear or may be nonlinear and differentiable. In the case of a linear architecture, problem (5.22) may also be solved in closed form, i.e., by using exact least squares.

(b) *Policy improvement*: Having solved the approximate policy evaluation problem (5.22), the new "improved" policy $\tilde{\mu}$ is obtained (at the states needed, when needed) by the policy improvement operation

$$\tilde{\mu}(i) \in \arg\min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u)\big(g(i, u, j) + \alpha \tilde{J}(j, \overline{r})\big), \tag{5.23}$$

where $\overline{r}$ is the parameter vector obtained from the policy evaluation (5.22). [The transition probabilities $p_{ij}(u)$ are required for this.]

† The choice of the terminal cost function approximation $\hat{J}$ may pose difficulties in SSP problems, particularly when most of the cost is incurred upon termination. Dealing with this issue may be problem-dependent. We will not discuss it further in this book.

Trajectory Reuse and Bias-Variance Tradeoff

As we have noted, to each training pair (i^s, β^s) there corresponds an N-stage trajectory over which the sample cost β^s is accumulated, but the length of the trajectory may depend on s. This allows sampling effort economies based on *trajectory reuse*. In particular, suppose that starting at some state i_0 we generate a long trajectory $(i_0, i_1, \ldots, i_N)$ using the policy μ. 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 $i_1, i_2, \ldots$, by using the tail portions of the trajectory $(i_0, i_1, \ldots, i_N)$ that start at these states.

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

In particular, a short length reduces the sampling effort, but is also a source of inaccuracy. The reason is that the cost of the tail portion of the trajectory (from stage N to infinity) is approximated by $\alpha^N \hat{J}(i_N)$, where i_N is the terminal state of the N-stage trajectory and $\hat{J}$ is the initial guess of J_μ. 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 β^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 β^s are less noisy, as they correspond to summation of fewer random stage costs. This leads to the so-called *bias-variance tradeoff*: short trajectories lead to larger bias but smaller variance of the cost samples.

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

Let us also note that the bias-variance tradeoff underlies the motivation for a number of alternative policy evaluation methods such as TD(λ), LSTD(λ), and LSPE(λ), which we will summarize in Section 5.5; see Section 6.3 of the book [Ber12] and other approximate DP/RL books referenced earlier. The papers [Ber13b], [YuB12], and the book [Ber12], Section 6.4, discuss a broad range of short trajectory sampling methods.

5.3.3 Model-Free Variant of a Critic-Only Method

We recall the exact PI algorithm for Q-factors (cf. Section 4.6.3), which given any policy μ, generates the new policy $\tilde{\mu}$ with a policy evalua-

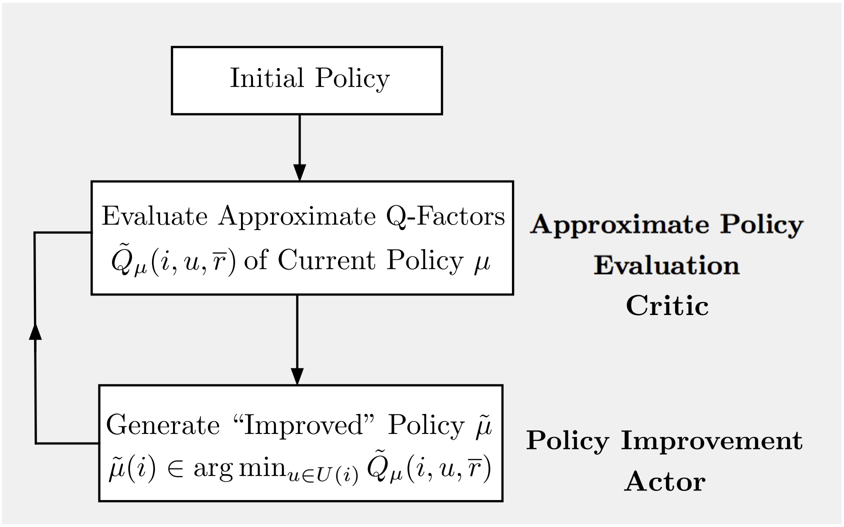

Initial Policy
Evaluate Approximate Q-Factors $\tilde{Q}_\mu(i,u,\overline{r})$ of Current Policy μ
Approximate Policy Evaluation Critic
Generate "Improved" Policy $\tilde{\mu}$
$\tilde{\mu}(i) \in \arg\min_{u\in U(i)} \tilde{Q}_\mu(i,u,\overline{r})$
Policy Improvement Actor

Figure 5.3.2 Block diagram of model-free approximate PI for Q-factors.

tion/policy improvement process. We will provide an approximate version of this algorithm, which involves a model-free simulation-based implementation. To this end, we will use Q-factors to bypass the need for transition probabilities $p_{ij}(u)$ in Eq. (5.23).

We introduce a parametric architecture $\tilde{Q}_\mu(i,u,r)$ for the Q-factors of μ. This architecture may be nonlinear, involving for example a neural network. It may also be a feature-based linear architecture, with a feature vector that depends either on both the state and the control or on just the state. In the former case, the architecture has the form

$$\tilde{Q}_\mu(i,u,r) = r'\phi(i,u),$$

where r is a weight vector that is independent of u. In the latter case, the architecture has the form

$$\tilde{Q}_\mu(i,u,r) = r(u)'\phi(i),$$

where $r(u)$ is a separate weight vector for each control u. This architecture is suitable for problems with a relatively small number of control options at each stage.

We approximate the PI process as follows; see Fig. 5.3.2.

(a) *Approximate policy evaluation*: Here, given a policy μ, we determine the value of the parameter vector r by generating (using a simulator of the system) a number of training triplets (i^s, u^s, β^s), $s = 1,\ldots,q$, and by using a least squares fit:

$$\overline{r} \in \arg\min_r \sum_{s=1}^{q} \big(\tilde{Q}_\mu(i^s,u^s,r) - \beta^s\big)^2. \tag{5.24}$$

In particular, for a given pair (i^s, u^s), the scalar β^s is a sample Q-factor corresponding to (i^s, u^s). It is generated by starting at i^s, using u^s at the first stage, simulating a trajectory of states and controls using μ for a total of N stages, accumulating the corresponding discounted costs, and adding a terminal cost approximation, similar to Section 5.3.2. Thus, β^s is a sample of

$$\sum_{j=1}^{n} p_{i^s j}(u^s)\big(g(i^s, u^s, j) + \alpha J_\mu^{N-1}(j) + \alpha^N \hat{J}(i_N)\big),$$

where $J_\mu^{N-1}(j)$ is the $(N-1)$-stages cost of μ starting at j, and $\alpha^N \hat{J}(i_N)$ is the terminal cost approximation. Note that β^s is computed similar to the scalar β^s of Eq. (5.22), except that *the first stage control is u^s rather than $\mu(i^s)$*. Similar to Section 5.3.2, the number of stages N in the sample trajectories may be different for different samples, and can be either large, or fairly small. Again an incremental method may be used to solve the training problem (5.24).

(b) *Policy improvement*: Here we compute the new policy $\tilde{\mu}$ (at the states needed, when needed) according to

$$\tilde{\mu}(i) \in \arg\min_{u \in U(i)} \tilde{Q}_\mu(i, u, \overline{r}), \qquad i = 1, \ldots, n, \tag{5.25}$$

where $\overline{r}$ is the parameter vector obtained from the policy evaluation operation (5.24). [Contrary to Eq. (5.23), this does not require the use of the transition probabilities $p_{ij}(u)$.]

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 $\big(i, \mu(i)\big)$ at every stage after the first, so *pairs (i, u) with $u \neq \mu(i)$ are not adequately explored*; cf. the discussion in Section 5.3.2. For this reason, it is necessary to make an effort to include in the samples a rich enough set of trajectories that start at pairs (i, u) with $u \neq \mu(i)$. We discuss this issue in Section 5.3.4.

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

$$\sum_{j=1}^{n} p_{ij}(u)\big(g(i, u, j) + \alpha \tilde{J}_\mu(j, \overline{r})\big),$$

with some $\tilde{Q}_\mu(i, u, \overline{r})$, possibly obtained with a model-free Q-factor approximation process such as the one of Section 2.1.4. In view of the two-fold approximation needed to obtain $\tilde{Q}_\mu(i, u, \overline{r})$, this scheme is more complex, but allows trajectory reuse and thus deals better with the exploration issue.

5.3.4 Implementation Issues of Parametric Policy Iteration

Approximate PI in its various forms has been the subject of extensive research, both theoretical and applied. Let us provide a few comments, focusing on the preceding critic-only 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 for PI. These architectures may involve the use of features, and they can be linear, or they can be nonlinear such as a neural network. A major advantage of a linear feature-based architecture is that the corresponding policy evaluations (5.22) and (5.24) 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(λ), LSTD(λ), and LSPE(λ), which will be summarized in Section 5.5, and are described in detail in several textbook sources.

Another interesting possibility for architecture choice has to do with *cost shaping*, which we discussed in Section 4.2. This possibility involves a modified cost per stage

$$\hat g(i,u,j) = g(i,u,j) + V(j) - V(i), \qquad i = 1,\dots,n,$$

for SSP problems, where V can be any approximation to J^* with $V(t) = 0$. The corresponding formula for discounted problems is

$$\hat g(i,u,j) = g(i,u,j) + \alpha V(j) - V(i), \qquad i = 1,\dots,n.$$

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_{μ^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 V with a parametric architecture or some other 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 papers referenced for Section 4.2 provide some applications of cost shaping (or reward shaping as it is sometimes called in RL contexts). An interesting possibility is to use complementary approximations for V and for J^* or J_{μ^k}. For example V may be approximated by a neural network-based approach that aims to discover the general form of J^* or J_{μ^k}, and then a different method may be applied to provide a local correction to V in order to refine the approximation. We will also illustrate this idea within the context of biased aggregation in Section 6.5.

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 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 practice.

We recall that in the PI variant of Section 5.3.2, in order to evaluate a policy μ, we determine the value of the parameter vector r by generating a large number of training pairs (i^s, β^s), $s = 1, \ldots, q$, where β^s is a sample cost corresponding to i^s and μ. We then use least squares training:

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

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

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

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

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

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

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

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

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

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

Let us also note the possibility to partition the state space, and select a rich enough representative set of initial states for each set of the partition. This allows a better control of exploration, and also ushers the possibility of using a distributed policy evaluation algorithm for each set of the partition; see our earlier discussion and the papers [BeY10], [BeY12], [YuB13a].

There have also been other related approaches to improve exploration, particularly in connection with the temporal difference methods to be discussed in Section 5.5. In some of these approaches, trajectories are generated through a mix of two policies: the policy being evaluated, sometimes called the *target policy*, to distinguish from the other policy, a probabilistically used policy, called the *behavior policy* that introduces enhanced exploration; see the end-of-chapter references. In the RL literature (see e.g., the book [SuB18]), methods that use a behavior policy are called *off-policy* methods, while methods that do not are called *on-policy* methods. A commonly suggested scheme is to use as an off-policy one that is "ϵ-greedy," i.e., that attains the minimum in the policy improvement phase within some threshold $\epsilon > 0$, which is experimentally chosen.

It is important to note, however, that using a behavior policy biases the cost samples towards that policy. Special modifications are needed, which can eliminate this source of bias and indeed can work exclusively with a behavior policy. Such modifications were introduced by Bertsekas and Yu [BeY09] (Section 3) to deal with the solution of general linear systems of equations that do not involve transition probabilities; see also the book [Ber12], Sections 6.4.2 and 7.3.1.

Generally, efficient sampling, and the issue of balancing exploration and the choice of promising controls (the exploration-exploitation tradeoff) is a subject of continuing research. Many authors have suggested practical schemes that may work in given contexts, including on-line situations. For some recent work, see the paper by Russo and Van Roy [RuV16], and the monograph [RVK18]. The paper by Osband, Van Roy, Russo, and Wen [OVR19] develops special schemes to deal with issues of deep exploration.

5.3.5 Convergence Issues of Approximate Policy Iteration-Oscillations

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

This oscillation phenomenon, first described by the author in a 1996 conference [Ber96b], occurs systematically in the absence of special conditions, for both optimistic and nonoptimistic PI methods. It can be observed even in simple examples (see [BeT96], Section 6.4.2, [Ber12], Section 6.4.3).

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

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

or equivalently

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

† Since we are now dealing with multiple policies, our notation reflects the dependence of the stage costs and transition probabilities on the control. Also T_μ is the Bellman operator corresponding to a policy μ, while T is defined in terms of minimization of T_μ over μ.

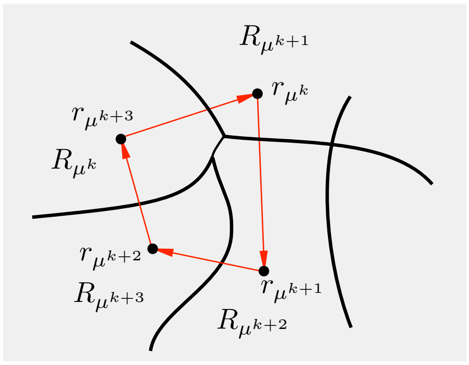


Figure 5.3.3 Greedy partition and cycle of policies generated by nonoptimistic PI with cost function approximation. In particular, μ yields $\overline{\mu}$ by policy improvement if and only if $r_\mu \in R_{\overline{\mu}}$. In this figure, the method cycles between four policies and the corresponding parameters r_{μ^k}, $r_{\mu^{k+1}}$, $r_{\mu^{k+2}}$, $r_{\mu^{k+3}}$.

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

We will first consider the nonoptimistic version of approximate PI; a similar phenomenon occurs for optimistic PI, as we discuss later. For simplicity, let us assume that we use a policy evaluation method that for each given μ produces a unique parameter vector denoted r_μ. The method starts with a policy μ^0 and generates r_{μ^0} by using the given policy evaluation method. It then finds a policy μ^1 such that $r_{\mu^0} \in R_{\mu^1}$. It then repeats the process with μ^1 replacing μ^0. If some policy μ^k satisfying $r_{\mu^k} \in R_{\mu^k}$ is encountered, the method converges to μ^k (it keeps generating μ^k). This is the necessary and sufficient condition for policy convergence in the nonoptimistic PI method.

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

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

(see Fig. 5.3.3). Furthermore, there may be many different cycles of this type, and the method may end up converging to any one of them. The actual cycle obtained depends on the initial policy μ^0. This is similar to gradient methods applied to minimization of functions with multiple local minima, where the limit of convergence depends on the starting point.

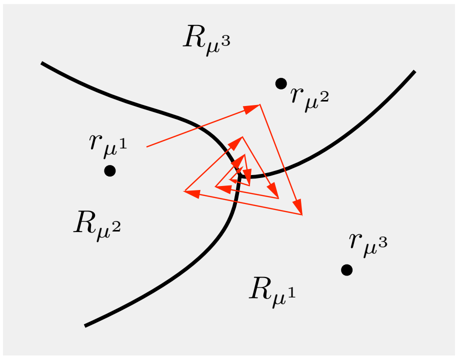


Figure 5.3.4 Illustration of a trajectory of optimistic policy iteration with cost function approximation. The algorithm settles into an oscillation between policies μ^1, μ^2, μ^3 with

$$r_{\mu^1} \in R_{\mu^2}, \qquad r_{\mu^2} \in R_{\mu^3}, \qquad r_{\mu^3} \in R_{\mu^1}.$$

The parameter vectors converge to the common boundary of R_{μ^1}, R_{μ^2}, R_{μ^3}.

Oscillations can in principle be quite 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. However, we will later give an argument (from Section 6.4.2 of the book [BeT96]), which suggests that oscillations may not degrade significantly the approximate PI performance for many types of problems. Moreover, we note that oscillations can be avoided and convergence can be shown under special conditions, which arise in particular when an aggregation approach is used; see Chapter 6 and the approximate PI survey [Ber11a]. Also, when policies converge, there is a more favorable error bound, cf. Prop. 5.1.5.

Oscillations and Chattering in Optimistic PI Methods

We now consider policy oscillations in the context of optimistic approximate PI. Then the trajectory of the method is less predictable and depends on the fine details of the policy evaluation, such as the frequency of the policy updates.

Generally, given the current policy μ, optimistic PI will move towards the corresponding "target" parameter r_μ, for as long as μ continues to be greedy with respect to the current cost-to-go approximation $\tilde{J}(\cdot, r)$, that is, for as long as the current parameter vector r belongs to the set R_μ. Once, however, the parameter r crosses into another set, say $R_{\overline{\mu}}$, the policy $\overline{\mu}$ becomes greedy, and r changes course and starts moving towards the new "target" $r_{\overline{\mu}}$. Thus, the "targets" r_μ of the method, and the corresponding policies μ and sets R_μ may keep changing, similar to nonoptimistic policy iteration. Simultaneously, the parameter vector r will move near the boundaries that separate the regions R_μ that the method visits, thus following reduced versions of the cycles that nonoptimistic PI may follow (see Fig. 5.3.3). Furthermore, as Fig. 5.3.4 indicates, if diminishing parameter changes are made between policy updates (such as for example when a diminishing stepsize is used by the policy evaluation method) and the method eventually cycles between several policies, the parameter vectors will tend to converge to the common boundary of the regions R_μ corresponding to these policies.

The simultaneous oscillation in policy space and convergence in parameter space in optimistic PI is called *chattering*. The literature reflects some confusion regarding the nature of this phenomenon, and in fact it is often erroneously assumed that if the sequence of generated parameters converges, the same is true for the sequence of generated policies.

An interesting observation is that the choice of the approximate policy evaluation method and exploration scheme may not be crucial for the quality of the final policy obtained. Using a different method changes the targets r_μ somewhat, but leaves the greedy partition unchanged. As a result, different methods "fish in the same waters" and tend to yield similar ultimate cycles of policies.

Note that when chattering occurs, the limit of optimistic PI tends to be on a common boundary of several subsets of the greedy partition and *may not meaningfully represent a cost approximation for any of the policies involved in the oscillation.* Thus, the parameter limit of the method cannot always be used to construct an approximation of the cost-to-go of any policy or the optimal cost-to-go. As a result, at the end of optimistic PI one may need to go back and perform a screening process; that is, evaluate the many policies generated by the method and select the most promising.

Still even with chattering, it is possible that the cost functions of the different policies involved in oscillation may not be "too different," so the cost functions of the generated policies may appear to be "nearly converging" (which is not the same as saying that they are all "good" policies). For an explanation, suppose that we have convergence to a parameter vector $\overline{r}$ and that there is a steady-state policy oscillation involving a collection of policies $\mathcal{M}$. Then, all the policies in $\mathcal{M}$ are greedy with respect to $\tilde{J}(\cdot,\overline{r})$, which implies that there is a subset of states i such that there are at least two different controls $\mu_1(i)$ and $\mu_2(i)$ satisfying

$$\begin{aligned}\min_{u\in U(i)} \sum_j p_{ij}(u)\big(g(i,u,j)+\alpha\tilde{J}(j,\overline{r})\big)\\ &= \sum_j p_{ij}\big(\mu_1(i)\big)\Big(g\big(i,\mu_1(i),j\big)+\alpha\tilde{J}(j,\overline{r})\Big)\\ &= \sum_j p_{ij}\big(\mu_2(i)\big)\Big(g\big(i,\mu_2(i),j\big)+\alpha\tilde{J}(j,\overline{r})\Big).\end{aligned}$$

Each equation of this type can be viewed as a constraining relation on the parameter vector $\overline{r}$. Thus, excluding singular situations, there will be at most s relations of the preceding form holding, where s is the dimension of $\overline{r}$. This implies that there will be at most s "ambiguous" states where more than one control is greedy with respect to $\tilde{J}(\cdot, \overline{r})$.

Now assume that we have a problem where the total number of states is much larger than s, and in addition there are no "critical" states; that is, the cost consequences of changing a policy in just a small number of states (say, of the order of s) is relatively small. It then follows that all policies in the set $\mathcal{M}$ involved in chattering have roughly the same cost function (although these policies are not necessarily "good" policies). Furthermore, one may argue that the cost approximation $\tilde{J}(\cdot, \overline{r})$ is close to the cost approximation $\tilde{J}(\cdot, r_\mu)$ that would be generated for any of the policies $\mu \in \mathcal{M}$. Note, however, that the assumption of "no critical states" may be hard to quantify and check, and may also not be true.

5.4 Q-LEARNING

In this section we will focus on discounted problems and discuss various Q-learning algorithms, which can be implemented in model-free fashion (if this is desirable). 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 aims to obtain the optimal Q-factors, 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. Some of these methods are also related to (highly) optimistic PI methods that are based on the use of Q-factors (cf. the methods of Section 5.3.3).

We first discuss the original form of Q-learning for discounted problems; the books [BeT96] and [Ber12], and the papers [Tsi94] and [YuB13b] contain discussions of Q-learning for SSP problems. Then we consider PI algorithms for Q-factors, including optimistic asynchronous versions, 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

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

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 Bellman operator for Q-factors defined by

$$(FQ)(i,u) = \sum_{j=1}^{n} p_{ij}(u) \left(g(i,u,j) + \alpha \min_{v \in U(j)} Q(j,v) \right), \quad \text{for all } (i,u). \tag{5.26}$$

It is straightforward to show that F is a contraction with modulus α, similar to the Bellman operator for costs [cf. Eq. (5.3)]. Thus the algorithm $Q_{k+1} = FQ_k$ converges to Q^* from every starting point Q_0.

The original and most widely known Q-learning algorithm ([Wat89]) is a stochastic version of VI, whereby the expected value in Eq. (5.26) 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 $p_{i_k j}(u_k)$. 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:

$$Q_{k+1}(i,u) = (1-\gamma^k) Q_k(i,u) + \gamma^k (F_k Q_k)(i,u), \qquad \text{for all } (i,u), \tag{5.27}$$

where

$$(F_k Q_k)(i,u) = \begin{cases} g(i_k,u_k,j_k) + \alpha \min_{v \in U(j_k)} Q_k(j_k,v) & \text{if } (i,u) = (i_k,u_k), \\ Q_k(i,u) & \text{if } (i,u) \neq (i_k,u_k). \end{cases} \tag{5.28}$$

Note that $(F_k Q_k)(i_k,u_k)$ is a single sample approximation of the expected value defining $(FQ_k)(i_k,u_k)$ in Eq. (5.26).

To guarantee the convergence of the algorithm (5.27)-(5.28) 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 γ^k should satisfy the conditions

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

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 considered next.

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

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

We have already discussed in Section 5.3.3 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 $p_{i^k j}(u^k)$.

(2) We generate the control u^{k+1} with the minimization

$$u^{k+1} \in \arg \min_{u \in U(i^{k+1})} \tilde{Q}(i^{k+1}, u, r^k).$$

[In some schemes, to enhance exploration, u^{k+1} is chosen with a small probability to be a random element of $U(i^{k+1})$ or one that is "ϵ-greedy," i.e., attains within some ϵ the minimum above.]

(3) We update the parameter vector via

$$\begin{aligned} r^{k+1} = r^k - \gamma^k \, \nabla \tilde{Q}(i^k, u^k, r^k) \\ \cdot \big(\tilde{Q}(i^k, u^k, r^k) - g(i^k, u^k, i^{k+1}) - \alpha \tilde{Q}(i^{k+1}, u^{k+1}, r^k)\big), \end{aligned} \tag{5.29}$$

where γ^k is a positive stepsize, and $\nabla(\cdot)$ denotes gradient with respect to r evaluated at the current parameter vector r^k. To get a sense for the rationale of this iteration, note that if $\tilde{Q}$ is a linear feature-based architecture, $\tilde{Q}(i,u,r) = \phi(i,u)'r$, then $\nabla\tilde{Q}(i^k,u^k,r^k)$ is just the feature vector $\phi(i^k,u^k)$, and iteration (5.29) becomes

$$r^{k+1} = r^k - \gamma^k \phi(i^k,u^k)\big(\phi(i^k,u^k)'r^k - g(i^k,u^k,i^{k+1}) - \alpha\phi(i^{k+1},u^{k+1})'r^k\big).$$

Thus r^k *is changed in an incremental gradient direction*: the one opposite to the gradient (with respect to r) of the incremental error

$$\big(\phi(i^k,u^k)'r - g(i^k,u^k,i^{k+1}) - \alpha\phi(i^{k+1},u^{k+1})'r^k\big)^2,$$

evaluated at the current iterate r^k.

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.

Extreme optimistic PI 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 just described is more commonly used in a less extreme/optimistic form, whereby several (perhaps many) state-control-transition cost-next state samples are batched together 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 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 (see Mnih et al. [MKS15]), is based on this idea (the term "Deep" is a reference to DeepMind's affinity for deep neural networks, but experience replay does not depend on the use of a deep neural network architecture).

5.5 ADDITIONAL METHODS - TEMPORAL DIFFERENCES

In this section, we summarize a few additional methods for approximation in value space in infinite horizon problems. We focus on *policy evaluation with a linear parametric architecture*, and on the simulation-based temporal difference methods. One of the aims of these methods is to address a bias-variance tradeoff that is similar in nature to the one discussed in Section 5.3.2. Our presentation is brief, somewhat abstract, and makes use of linear algebra mathematics. It may be skipped without loss of continuity. This is only a summary; it is meant to provide a connection to other material in this chapter, and orientation for further reading into both the optimization and artificial intelligence literature on the subject.

Approximate Policy Evaluation Using Projections

Our main concern in policy evaluation is to solve approximately the Bellman equation corresponding to a given policy μ. Thus, for discounted problems, we are interested in solving the linear system of equations

$$J_\mu(i) = \sum_{i=1}^{n} p_{ij}\big(\mu(i)\big)\Big(g\big(i,\mu(i),j\big) + \alpha J_\mu(j)\Big), \qquad i = 1,\ldots,n,$$

or in shorthand,

$$J_\mu = T_\mu J_\mu, \tag{5.30}$$

where T_μ is the Bellman operator for μ, given by

$$(T_\mu J)(i) = \sum_{i=1}^{n} p_{ij}\big(\mu(i)\big)\Big(g\big(i,\mu(i),j\big) + \alpha J(j)\Big), \qquad i = 1,\ldots,n. \tag{5.31}$$

Let us consider the approximate solution of this equation by parametric approximation (cf. Section 5.3). This amounts to replacing J_μ with some vector that lies within the manifold represented by the approximation architecture

$$\mathcal{M} = \Big\{\big(\tilde J(1,r),\ldots,\tilde J(n,r)\big) \mid \text{all parameter vectors } r\Big\}. \tag{5.32}$$

The approximate solution of systems of equations within an approximation manifold of the form (5.32) 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

$$\|J\|^2 = \sum_{i=1}^{n} \xi_i\big(J(i)\big)^2, \tag{5.33}$$

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

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

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.

We may consider three general approaches for approximation of J_μ.

(a) Project J_μ onto $\mathcal{M}$ to obtain $\Pi(J_\mu)$, which we use as an approximation of J_μ.

(b) Start with some approximation $\hat{J}$ of J_μ, 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_μ.

(c) Solve a projected version $J_\mu = \Pi(T_\mu J_\mu)$ of the Bellman Eq. (5.30), and use the solution of this projected equation as an approximation to J_μ. We will also discuss related projected versions that involve other operators in place of T_μ.

The preceding three approaches are hard to implement exactly; for example, (a) is impossible since we do not know the values of J_μ. However, it turns out that it is possible to implement approximately 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

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

$$\mathcal{M} = \{\Phi r \mid r \in \Re^m\}, \tag{5.34}$$

where $\Re^m$ is the space of m-dimensional vectors, and Φ 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 Φ.

We consider projection with respect to the weighted Euclidean norm of Eq. (5.33), so $\Pi(J)$ is of the form Φr^*, where

$$r^* \in \arg\min_{r \in \Re^m} \|\Phi r - J\|_\xi^2 = \arg\min_{r \in \Re^m} \sum_{i=1}^n \xi_i \big(\phi(i)' r - J(i)\big)^2. \tag{5.35}$$

We can perform the above minimization by setting to 0 the gradient of the cost function,

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

We then obtain the solution in closed form,

$$r^* = \left(\sum_{i=1}^n \xi_i \phi(i) \phi(i)' \right)^{-1} \sum_{i=1}^n \xi_i \phi(i) J(i), \tag{5.36}$$

assuming that the inverse above exists. One of the difficulties 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 ξ if necessary) that $\xi = (\xi_1, \dots, \xi_n)$ is a probability distribution, we may view the two terms in the right side of Eq. (5.36) as expected values with respect to ξ, and approximate them by Monte Carlo simulation. In particular, suppose that we generate a set of state index samples i^s, $s = 1, \dots, q$, according to the distribution ξ, and form the Monte Carlo estimates

$$\frac{1}{q}\sum_{s=1}^{q} \phi(i^s)\phi(i^s)' \approx \sum_{i=1}^{n} \xi_i \phi(i)\phi(i)', \qquad \frac{1}{q}\sum_{s=1}^{q} \phi(i^s)\beta^s \approx \sum_{i=1}^{n} \xi_i \phi(i)J(i), \tag{5.37}$$

where β^s is a simulation-based approximation to the exact value $J(i^s)$. It may be modeled as

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

where $n(i^s)$ is a random variable ("noise") resulting from the randomness of the simulation. For the Monte Carlo estimate in the right side of Eq. (5.37) to be asymptotically correct [in the sense that it converges to $\sum_{i=1}^n \xi_i \phi(i)\phi(i)'$ as $q \to \infty$], we must have

$$\frac{1}{q}\sum_{s=1}^{q} \phi(i^s)n(i^s) \approx 0, \tag{5.38}$$

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

Given the Monte Carlo approximations (5.37) of the two terms in Eq. (5.36), we can estimate r^* with

$$\overline{r} = \left(\sum_{s=1}^{q} \phi(i^s)\phi(i^s)'\right)^{-1} \sum_{s=1}^{q} \phi(i^s)\beta^s, \tag{5.40}$$

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

$$\lim_{q\to\infty} \frac{\sum_{s=1}^q \delta(i^s = i)n(i^s)}{\sum_{s=1}^q \delta(i^s = i)} = 0, \qquad \text{for all } i = 1, \dots, n, \tag{5.39}$$

where $\delta(i^s = i) = 1$ if $i^s = i$ and $\delta(i^s = i) = 0$ if $i^s \neq i$. It states that the Monte Carlo averages of the noise terms corresponding to every state i are zero. Then the expression in Eq. (5.38) has the form

$$\begin{aligned}\frac{1}{q}\sum_{s=1}^{q} \phi(i^s)n(i^s) &= \frac{1}{q}\sum_{i=1}^{n} \phi(i)\sum_{s=1}^{q} \delta(i^s = i)n(i^s)\\ &= \frac{1}{q}\sum_{i=1}^{n} \phi(i)\left(\sum_{s'=1}^{q} \delta(i^{s'} = i)\right)\frac{\sum_{s=1}^q \delta(i^s = i)n(i^s)}{\sum_{s=1}^q \delta(i^s = i)},\end{aligned}$$

and converges to 0 as $q \to \infty$, assuming Eq. (5.39) [and also that each index i is sampled infinitely often so that Eq. (5.39) makes sense].

(assuming sufficiently many samples are obtained to ensure the existence of the inverse above).† This is also equivalent to estimating r^* by approximating the least squares minimization (5.35) with the following least squares training problem

$$\overline{r} \in \arg\min_{r\in\Re^m} \sum_{s=1}^{q} \big(\phi(i^s)'r - \beta^s\big)^2. \tag{5.41}$$

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

(a) Replacing the expected values in the exact projection formula (5.36) with the simulation-based estimates (5.37) [cf. Eq. (5.40)].

(b) Replacing the exact least squares problem (5.35) with the simulation-based least squares approximation (5.41).

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

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

$$\xi_i = \lim_{k\to\infty} \frac{1}{q} \sum_{s=1}^{q} \delta(i^s = i), \qquad i = 1,\ldots,n, \tag{5.42}$$

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 ξ_i need not be predetermined. In fact, often the exact values of ξ_i do not matter much, and *one may wish to first specify a reasonable and convenient sampling scheme, and let ξ_i be implicitly specified via Eq. (5.42).*

Projected Equation View of Approximate Policy Evaluation

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

$$\tilde{J}_\mu \approx \Pi(T_\mu^N \hat{J}), \tag{5.43}$$

† The preceding derivation and the formula (5.40) actually make sense even if $\xi = (\xi_1,\ldots,\xi_n)$ has some zero components, as long as the inverses in Eqs. (5.36) and (5.40) exist. This is related to the concept of *seminorm projection*; see the paper by Yu and Bertsekas [YuB12] for a discussion related to approximate DP.

where:†

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

$$\tilde{J}_\mu = \big(\tilde{J}(1,\overline{r}),\ldots,\tilde{J}(n,\overline{r})\big),$$

which is the approximate policy evaluation of μ, used in the policy improvement operation (5.23). Here $\overline{r}$ is the solution of the training problem (5.22).

(b) T_μ is the Bellman operator corresponding to μ, which maps a vector $J = \big(J(1),\ldots,J(n)\big)$ into the vector $T_\mu J$ of Eq. (5.31).

(c) T_μ^N denotes the N-fold application of the operator T_μ, where N is the length of the sample trajectories used in the least squares regression problem (5.22). In particular, $(T_\mu^N \hat{J})(i)$ is the cost of starting at i, using μ for N stages, with terminal cost function $\hat{J}$ (or equivalently, the result of an optimistic policy evaluation using N value iterations starting from $\hat{J}$). The sample state-cost pairs (i^s,β^s) are obtained from trajectories corresponding to this N-stage problem.

(d) $\Pi(T_\mu^N \hat{J})$ denotes projection of the vector $T_\mu^N \hat{J}$ on the manifold of possible approximating vectors $\mathcal{M}$ with respect to a weighted norm, where each weight ξ_i represents the relative frequency of the state i as initial state of a training trajectory. This projection is approximated by the least squares regression (5.22). In particular, the cost samples β^s of the training set are noisy samples of the values $(T_\mu^N \hat{J})(i^s)$, and the projection is approximated with a least squares minimization, to yield the function $\tilde{J}_\mu$ of Eq. (5.43).

Suppose now that $T_\mu^N \hat{J}$ is close to J_μ (which happens if either N is large or $\hat{J}$ is close to J_μ, or both) and the number of samples q is large (so that the simulation-based regression approximates well the projection operation Π). Then from Eq. (5.43), the approximate evaluation $\tilde{J}_\mu$ of μ approaches the projection of J_μ on the approximation manifold (5.32), which can be viewed as the best possible approximation of J_μ (at least relative to the distance metric defined by the weighted projection norm). This provides an abstract formal rationale for the parametric PI method of Section 5.3.2, which is based on Eq. (5.43).

TD(λ), LSTD(λ), and LSPE(λ)

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

† The equation (5.43) 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_μ^N in the equation should be replaced by a more complicated weighted sum of powers of T_μ; see the paper [YuB12] for related ideas.

proximate evaluation of a policy. Examples of such methods are TD(λ), LSTD(λ), and LSPE(λ), where λ is a scalar with $0 \leq \lambda < 1$.[†]

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

$$\Phi r = \Pi\big(T_\mu^{(\lambda)} \Phi r\big), \tag{5.44}$$

where T_μ is the operator (5.31), $T_\mu^{(\lambda)} J$ is defined by

$$(T_\mu^{(\lambda)} J)(i) = (1-\lambda) \sum_{\ell=0}^{\infty} \lambda^\ell \, (T_\mu^{\ell+1} J)(i), \qquad i = 1, \ldots, n, \tag{5.45}$$

and Π is projection on the approximation subspace

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

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

Of course the projected equation (5.44) 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 leads to the problem of finding a vector $\overline{r}$ such that

$$\Phi \overline{r} = \tilde{\Pi}\big(T_\mu^{(\lambda)} \Phi \overline{r}\big), \tag{5.46}$$

where $\tilde{\Pi}$ is an approximate projection implemented by sampling.

The meaning of the above equation is to *find a vector $\Phi \overline{r} \in \mathcal{M}$, which when transformed with $T_\mu^{(\lambda)}$ and then projected approximately (using $\tilde{\Pi}$) back onto $\mathcal{M}$, yields itself*. To implement the approximate projection, suppose that for q samples of initial states i^s, $s = 1, \ldots, q$, we were able compute the corresponding samples of $(T_\mu^{(\lambda)} \Phi \overline{r})(i^s)$ (this is hypothetical since $\overline{r}$ is unavailable). Then the parameter vector $\overline{r}$ would be obtained as

$$\overline{r} \in \arg\min_r \sum_{s=1}^{q} \big(\phi(i^s)' r - \text{sample of } (T_\mu^{(\lambda)} \Phi \overline{r})(i^s)\big)^2, \tag{5.47}$$

where $\phi(i)'$ is the ith row of Φ, so that the inner product $\phi(i)'r$ denotes the ith component of the vector Φr. This least squares problem implements the

[†] TD stands for "temporal difference," LSTD stands for "least squares temporal difference," and LSPE stands for "least squares policy evaluation."

approximate projection $\tilde{\Pi}$ of Eq. (5.46) [i.e., $\overline{r}$, the solution of Eq. (5.46) minimizes over r the quadratic expression in Eq. (5.47) (which also involves $\overline{r}$!)].

A key fact now is that the optimality condition for the problem in Eq. (5.47) can be written as a closed form equation that can be solved for $\overline{r}$, and indeed LSTD(λ) does exactly that. By contrast LSPE(λ) and TD(λ) solve this optimality condition iteratively and incrementally, in the spirit of the methods of Section 3.1.3. We will first give a high level description of the three methods, and then focus on the simpler case where $\lambda = 0$.

(a) The LSTD(λ) method, after the q samples have been collected, writes the optimality condition for Eq. (5.47) as a linear equation of the form

$$C_\lambda \overline{r} = d_\lambda, \tag{5.48}$$

where C_λ is some $m \times m$ square matrix, and d_λ is an m-dimensional vector (which depend on λ). The components of C_λ and d_λ are explicitly computed, and LSTD(λ) produces the approximate cost function $\tilde{J}_\mu(i) = \Phi\overline{r}$, where $\overline{r} = C_\lambda^{-1} d_\lambda$ is the solution of Eq. (5.48).

(b) The LSPE(λ) method solves the projected equation (5.44) by using a simulation-based *projected value iteration*,

$$J_{k+1} = \tilde{\Pi}\big(T_\mu^{(\lambda)} J_k\big). \tag{5.49}$$

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

(c) The TD(λ) method is a simpler iterative stochastic approximation method for solving the linear equation (5.48), which resembles the incremental gradient method of Section 3.1.3. It can also be viewed as a stochastic version of the proximal algorithm for solving this linear equation (the parameter λ is related to the penalty parameter of the proximal algorithm; see the author's papers [Ber16c] and [Ber18d]).

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

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

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

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

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 μ. As earlier, let the ith row of the matrix Φ 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 $T_\mu \Phi \overline{r}$ in Eq. (5.47) are

$$g(i^s, i^{s+1}) + \alpha\phi(i^{s+1})'\overline{r}, \qquad s = 1, \ldots, q,$$

and the least squares problem in Eq. (5.47) has the form

$$\overline{r} \in \arg\min_r \sum_{s=1}^{q} \big(\phi(i^s)'r - g(i^s, i^{s+1}) - \alpha\phi(i^{s+1})'\overline{r}\big)^2. \tag{5.50}$$

(Note that this is an unusual optimization problem because its solution $\overline{r}$ appears in its quadratic cost function.) By setting the gradient of the minimized expression above to zero, we obtain the condition for $\overline{r}$ to attain the above minimum:

$$\sum_{s=1}^{q} \phi(i^s)\big(\phi(i^s)'\overline{r} - g(i^s, i^{s+1}) - \alpha\phi(i^{s+1})'\overline{r}\big) = 0. \tag{5.51}$$

Solving this equation for $\overline{r}$ yields the LSTD(0) solution:

$$\overline{r} = \left(\sum_{s=1}^{q} \phi(i^s)\big(\phi(i^s) - \alpha\phi(i^{s+1})\big)'\right)^{-1} \sum_{s=1}^{q} \phi(i^s) g(i^s, i^{s+1}). \tag{5.52}$$

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 is also 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 using regularization/stabilization ideas; see Wang and Bertsekas [WaB13a], [WaB13b], and the DP textbook [Ber12], Section 7.3.

The expression

$$d^s(\overline{r}) = \phi(i^s)'\overline{r} - g(i^s, i^{s+1}) - \alpha\phi(i^{s+1})'\overline{r} \tag{5.53}$$

that appears in Eq. (5.51) is referred to as the *temporal difference associated with the sth transition and parameter vector $\overline{r}$.* 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

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

[cf. Eq. (5.49)]. At the kth iteration, it uses only the samples $s = 1, \ldots, k$, and updates the parameter vector according to

$$r^{k+1} = r^k - \left(\sum_{s=1}^{k} \phi(i^s)\phi(i^s)'\right)^{-1} \sum_{s=1}^{k} \phi(i^s) d^s(r^k), \qquad k = 1, 2, \ldots, \tag{5.54}$$

where $d^s(r^k)$ is the temporal difference of Eq. (5.53), evaluated at the iterate r^k; the form of this iteration is derived similar to the case of LSTD(0). The vector r^q obtained after q iterations, when all the samples have been processed, is the one used for the approximate evaluation of J_μ. Note that the inverse in Eq. (5.54) can be updated economically from one iteration to the next, using fast linear algebra operations (cf. the use of the Sherman Morrison formula in the incremental Newton method in Section 3.1.3).

Overall, it can be shown that LSTD(0) and LSPE(0) [with efficient matrix inversion in Eq. (5.54)] require essentially identical amount of work to process the q samples associated with the current policy μ [this is also true for the LSTD(λ) and LSPE(λ) 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

$$r^{k+1} = r^k - \gamma^k \phi(i^k) d^k(r^k), \qquad k = 1, 2, \ldots, \tag{5.55}$$

where γ^k is a diminishing stepsize parameter and $d^k(r^k)$ is the temporal difference, cf. Eq. (5.53). It can be seen that TD(0) resembles an *incremental gradient* iteration for solving the least squares training problem (5.50), but with $\bar{r}$ replaced by the current iterate r^k. The reason is that the gradient of the typical kth term in the least squares sum of Eq. (5.50) is the vector $\phi(i^k)d^k(r^k)$ that appears in the TD(0) iteration (5.55) (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 γ^k that must be carefully controlled.

By contrast the LSPE(0) iteration (5.54) uses the full sum

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

which may be viewed as an *aggregated incremental* method, with scaling provided by the matrix $\left(\sum_{s=1}^{k} \phi(i^s)\phi(i^s)'\right)^{-1}$. This explains why TD(0) is generally much slower and more fragile than LSPE(0). While TD(0) does not require a matrix inversion, the same is true for LSPE(0), once modified to replace the scaling matrix with its diagonal approximation, similar to the incremental Newton method (cf. Example 3.1.9). On the other hand there is no way to avoid matrix inversion in the implementation of LSTD(0).

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(λ) relies on matrix inversion and not on iteration, so it does not have a serious convergence issue, but the system (5.48) may be singular or nearly singular, in which case very accurate simulation is needed to approximate the matrix C well enough for its inversion to be reliable, as we have noted earlier (see [WaB13a], [WaB13b], and [Ber12], Section 7.3). LSPE(λ) has a convergence issue because the mapping $\Pi T_\mu^{(\lambda)}$ may not be a contraction mapping (even though T_μ is) and the projected value iteration (5.49) may not be convergent. It turns out that the mapping $\Pi T_\mu^{(\lambda)}$ is guaranteed to be a contraction for λ sufficiently close to 1, so the convergence difficulty may be circumvented by suitably increasing λ (see [Ber12], Section 6.3.6).

Direct and Indirect Policy Evaluation Methods

In trying to compare the approximate policy evaluation methods discussed in this section, we may draw a distinction between *direct methods*, which aim to compute approximately the projection $\Pi(J_\mu)$, and *indirect methods*, which try to solve the projected equation (5.44).

The method of Section 5.3.2 is direct and is based on Eq. (5.43). In particular, as $N \to \infty$ and $q \to \infty$, it yields the approximate evaluation $\Pi(J_\mu)$. The TD methods are indirect, and aim at computing the solution of the projected equation (5.44). The solution of this equation is of the form Φr_λ^*, where the parameter vector r_λ^* depends on λ. In particular the projected equation solution Φr_λ^* is different from $\Pi(J_\mu)$. It can be shown that it satisfies the error bound

$$\|J_\mu - \Phi r_\lambda^*\|_\xi \leq \frac{1}{\sqrt{1-\alpha_\lambda^2}} \|J_\mu - \Pi(J_\mu)\|_\xi, \tag{5.56}$$

where

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

and $\|\cdot\|_\xi$ is a special projection norm of the form (5.33), where ξ is the steady-state probability distribution of the controlled system Markov chain under policy μ. Moreover as $\lambda \to 1$ the projected equation solution Φr_λ^* approaches $\Pi(J_\mu)$. Based on this fact, methods which aim to compute $\Pi(J_\mu)$, such as the direct method of Section 5.3.2 are sometimes called

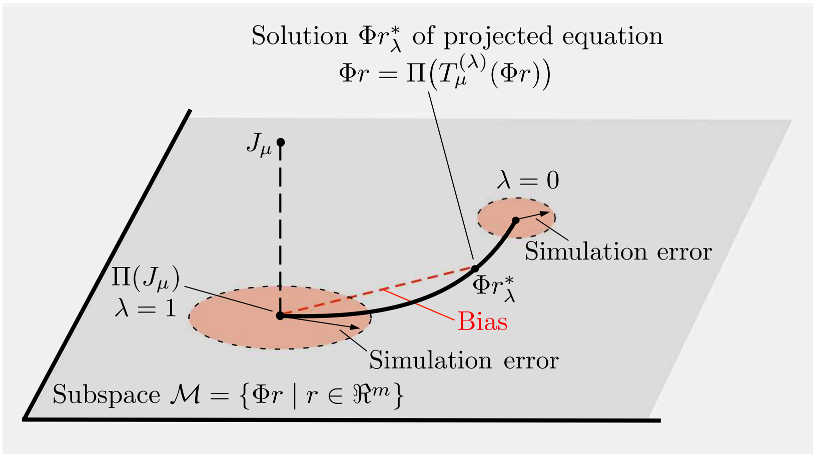

Solution Φr_λ^* of projected equation
$\Phi r = \Pi\big(T_\mu^{(\lambda)}(\Phi r)\big)$
J_μ
$\lambda = 0$
Simulation error
$\Pi(J_\mu)$
$\lambda = 1$
Φr_λ^*
Bias
Simulation error
Subspace $\mathcal{M} = \{\Phi r \mid r \in \Re^m\}$

Figure 5.5.1 Illustration of the bias-variance tradeoff in estimating the solution of the projected equation for different values of λ. As λ increases from $\lambda = 0$ towards $\lambda = 1$, the solution Φr_λ^* of the projected equation

$$\Phi r = \Pi\big(T^{(\lambda)}(\Phi r)\big)$$

approaches the projection $\Pi(J_\mu)$. The difference

$$\Phi r_\lambda^* - \Pi(J_\mu)$$

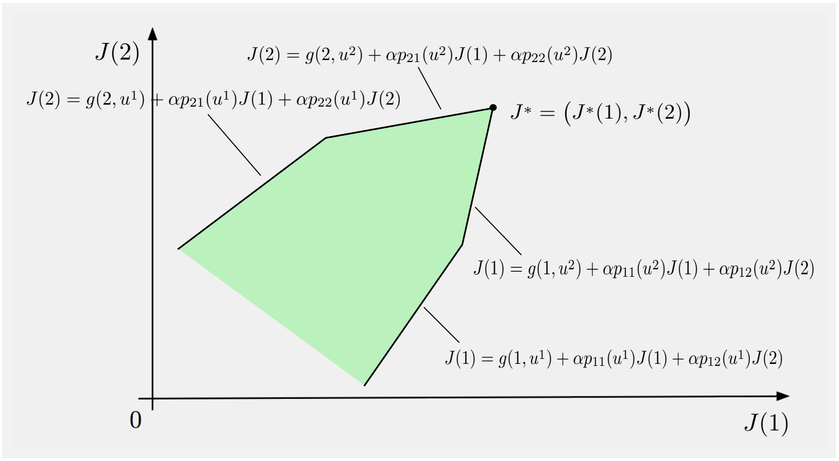
is the bias, and it decreases to 0 as λ approaches 1, while the simulation error variance increases.

TD(1). We refer to [Ber12], Section 6.3.6, for an account of this analysis, which is beyond the scope of this book.

The difference $\Phi r_\lambda^* - \Pi(J_\mu)$ is commonly referred to as the *bias* and is illustrated in Figure 5.5.1. As indicated in this figure and as the estimate (5.56) suggests, there is a *bias-variance tradeoff*. As λ is decreased, the solution of the projected equation (5.44) changes and more bias is introduced relative to the "ideal" approximation ΠJ_μ (this bias can be embarrassingly large as shown by examples in the paper [Ber95]). At the same time, however, the simulation samples of $T_\mu^{(\lambda)} J$ contain less noise as λ is decreased [cf. Eq. (5.45)]. This provides another view of the bias-variance tradeoff, which we discussed in Section 5.3.2 in connection with the use of short trajectories.

5.6 EXACT AND APPROXIMATE LINEAR PROGRAMMING

Another method for exact solution of infinite horizon DP problems is based on the use of linear programming. In particular, J^* can be shown to be the unique optimal solution of a certain linear program. Focusing



$J(2)$
$J(2) = g(2,u^2) + \alpha p_{21}(u^2)J(1) + \alpha p_{22}(u^2)J(2)$
$J(2) = g(2,u^1) + \alpha p_{21}(u^1)J(1) + \alpha p_{22}(u^1)J(2)$
$J^* = \big(J^*(1), J^*(2)\big)$
$J(1) = g(1,u^2) + \alpha p_{11}(u^2)J(1) + \alpha p_{12}(u^2)J(2)$
$J(1) = g(1,u^1) + \alpha p_{11}(u^1)J(1) + \alpha p_{12}(u^1)J(2)$
0
$J(1)$

Figure 5.6.1 A linear program associated with a two-state discounted problem. The constraint set is shaded, and the objective to maximize is $J(1) + J(2)$. Note that because we have $J(i) \leq J^*(i)$ for all i and vectors J in the constraint set, the vector J^* maximizes any linear cost function of the form $\sum_{i=1}^n \beta_i J(i)$, where $\beta_i \geq 0$ for all i. If $\beta_i > 0$ for all i, then J^* is the unique optimal solution of the corresponding linear program.

on α-discounted problems, the key idea is that J^* is the "largest" (on a component-by-component basis) vector J that satisfies the constraint

$$J(i) \leq \sum_{j=1}^n p_{ij}(u)\big(g(i,u,j) + \alpha J(j)\big), \qquad \text{for all } i = 1,\ldots,n \text{ and } u \in U(i), \tag{5.57}$$

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

$$\begin{aligned} &\text{maximize } \sum_{i=1}^n J(i) \\ &\text{subject to the constraint (5.57),} \end{aligned} \tag{5.58}$$

(see Fig. 5.6.1). The constraint (5.57) is sometimes called the *Bellman inequality*, and is used in a variety of DP contexts besides the framework of this section.

To verify this, let us use the VI algorithm to generate a sequence of vectors $J_k = \big(J_k(1),\ldots,J_k(n)\big)$ starting with an initial condition vector $J_0 = \big(J_0(1),\ldots,J_0(n)\big)$ such that

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

This inequality and the monotonicity property of the Bellman operator can be used to show that

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

and similarly

$$J(i) = J_0(i) \leq J_k(i) \leq J_{k+1}(i), \qquad \text{for all } i \text{ and } k.$$

Since $J_k(i)$ converges to $J^*(i)$ as $k \to \infty$, it follows that we will also have

$$J(i) = J_0(i) \leq J^*(i), \qquad \text{for all } i.$$

Thus out of all J satisfying the constraint (5.57), J^* is the largest on a component-by-component basis.

Unfortunately, for large n the dimension of the linear program (5.58) can be very large and its solution can be impractical, particularly in the absence of special structure. In this case, we may consider finding an approximation to J^*, which can be used in turn to obtain a (suboptimal) policy through approximation in value space.

One possibility is to approximate $J^*(i)$ with a linear feature-based architecture

$$\tilde J(i,r) = \sum_{\ell=1}^{m} r_\ell \phi_\ell(i),$$

where $r = (r_1,\ldots,r_m)$ is a vector of parameters, and for each state i, $\phi_\ell(i)$ are some features. It is then possible to determine r by using $\tilde J(i,r)$ in place of J^* in the preceding linear programming approach. In particular, we compute r as the solution of the program

$$\begin{aligned} &\text{maximize} \quad \sum_{i\in \tilde I} \tilde J(i,r) \\ &\text{subject to} \quad \tilde J(i,r) \leq \sum_{i=1}^{n} p_{ij}(u)\big(g(i,u,j) + \alpha \tilde J(j,r)\big), \quad i \in \tilde I,\ u \in \tilde U(i), \end{aligned}$$

where $\tilde I$ is either the state space $I = \{1,\ldots,n\}$ or a suitably chosen subset of I, and $\tilde U(i)$ is either $U(i)$ or a suitably chosen subset of $U(i)$. This is a linear program because $\tilde J(i,r)$ is assumed linear in the parameter vector r.

The major difficulty with this approximation approach is that while the dimension of r may be moderate, the number of constraints can be extremely large. It can be as large as nm, where n is the number of states and m is the maximum number of elements of the control constraint sets $U(i)$. Thus for a large problem it is essential to reduce drastically the number of constraints. Random sampling methods may be used to select a suitable subset of the constraints to enforce (perhaps using some known suboptimal policies), and progressively enrich the subset as necessary. With such constraint sampling schemes, the linear programming approach may be practical even for problems with a very large number of states. Its application, however, may require considerable sophistication and 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_μ of a stationary policy μ in the context of approximate PI. The motivation for this is that the linear program to evaluate μ involves fewer constraints (a single constraint for each state).

5.7 APPROXIMATION IN POLICY SPACE

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

$$\tilde{\mu}(i, r), \qquad i = 1, \dots, n.$$

The parametrization may involve features and/or a neural network. The idea is then to optimize some measure of performance with respect to r. Thus in marked difference to approximation in value space, the point of departure for approximation in policy space is not the DP framework, and the algorithmic ideas of VI and PI. Instead it is general purpose optimization methodologies, such as gradient descent and random search.

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

Let us provide some examples where policy parametrization is natural and/or has been successful in practice.

Example 5.7.1: (Supply Chain Parametrization)

There are many problems where the general structure of an optimal or near-optimal policy is known through analysis or insight into the problem's struc-

† The SSP case is somewhat more complicated because of the need to focus on proper policies, i.e., policies that are guaranteed to terminate starting from all initial states. However, the techniques of Sections 5.7.1 and 5.7.2 apply, with modest modifications, to finite horizon problems.

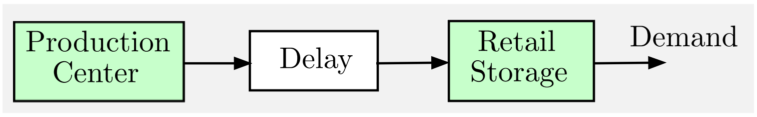


Figure 5.7.1. Illustration of a simple supply chain system.

ture. An important case are supply chain systems involving production, inventory, and retail centers that are connected with transportation links. A simple example is illustrated in Fig. 5.7.1.

Here a retail center places orders to the production center, depending on current stock. There may be orders in transit, and stochastic demand and delays. Such a problem can be formulated by DP but can be very difficult to solve exactly. However, intuitively, there is a near-optimal policy that 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 the policy is specified by the parameter vector $r = (r_1, r_2)$, and can be trained by one of the methods of this section. This type of approach readily extends to the case of a complex network of production/retail centers, multiple products, etc.

Example 5.7.2: (PID Control)

A time-honored and very popular 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. 5.7.2.†

The input/control u_k applied at time k is 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 difference,

$$d_k = e_k - e_{k-1},$$

† Note that PID control by its nature applies to problems with continuous state and control spaces, and it may not be readily applicable to the discounted and SSP problems of the present chapter. We discuss it here because it has a strong connection to the approximation in policy space material of this section, so it may become the starting point for extensions of the methodology given here.

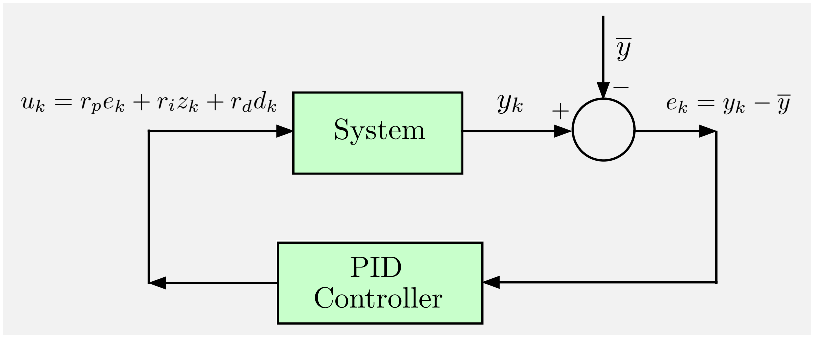

$u_k = r_p e_k + r_i z_k + r_d d_k$
System
y_k
$\overline{y}$
$+$
$-$
$e_k = y_k - \overline{y}$
PID Controller

Figure 5.7.2. 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" $\overline{y}$. The controller observes the error $e_k = y_k - \overline{y}$ 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 to time k,

$$z_k = \sum_{m=0}^{k} e_m,$$

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 β is a 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.

or a damped version thereof generated by a recursion such as

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

where β 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 stable closed-loop system, zero error in steady-state, and satisfactory transient behavior). The proportional term serves to drive the error towards 0. The integral term guarantees that for a stable closed-loop system, the error vanishes asymptotically, and among others, is very important to neutralize the effects of nonzero mean disturbances in the system's dynamics. The derivative term is often not needed, and when it is, its purpose is mainly to improve the transient behavior of the closed-loop system by "anticipating" changes in the error e_k.

Note that, *contrary to the MPC controllers 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 Frihauf, 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 in 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 related schemes are often intended for use in real-time adaptive control of systems with frequently and unpredictably changing parameters (cf. Section 1.3.8). This is a challenging application context that RL methods are not able to fully address with the current state of the art.

Example 5.7.3: (Policy Parametrization Through Cost Parametrization)

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

$$\tilde{\mu}(i, r) \in \arg\min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u)\big(g(i, u, j) + \alpha \tilde{J}(j, r)\big), \tag{5.59}$$

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 $\tilde{\mu}(r)$ of Eq. (5.59) 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.

† The policy parametrization (5.59) may not be 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. (5.59) with

As an illustration, we note that 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 was derived through a feature-based parametrization in value space (cf. Example 3.1.3). The algorithm used for training was the cross-entropy method, a random search algorithm to be discussed in Section 5.7.1.

Example 5.7.4: (Policy Parametrization Through Feature Parametrization)

An interesting special case of policy parametrization is based on state features, so that $\tilde{\mu}$ depends on the state i through some feature vector $\phi(i)$, i.e.,

$$\tilde{\mu}(i, r) = \hat{\mu}\big(\phi(i), r\big),$$

for some function $\hat{\mu}$. Then to implement the policy, we only need to know $\phi(i)$ rather than i. Moreover, to train the policy with most of the simulation-based methods of this section, we only need a cost simulator [to generate samples of $J_{\tilde{\mu}}(i)$], and a feature simulator [to generate feature values $\phi(i)$]. In particular, we do not need the simulated values of i. An example of feature-based policy parametrization is provided by the cost parametrization approach of the preceding example, in the case where the cost function approximation $\tilde{J}$ is based on features.

Another context where feature parametrization may be useful is in partial state information problems (POMDP), where we use as state at a given time k the entire observation and control history I_k up to k (the information record considered in Example 3.1.6). When the horizon is infinite, the size of I_k increases without bound, thus complicating the implementation of methods based on approximation in policy space. On the other hand, convenient features for policy parametrization may be obtained from sufficient statistics of the information record I_k, such as those discussed in Example 3.1.6. Some possibilities include a partial control-observation history (a subset of I_k), or some estimate of the state or other "good" features of I_k.

---

a *randomized* policy that applies at state i a control $u \in U(i)$ with probability

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

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

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

and β is a positive scalar, which 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).

The feature-based policy parametrization approach may also be convenient for rollout algorithms with a base policy that is feature-parametrized (e.g., a finite history controller for POMDP). It may also be useful in the context of the rollout-based approximate PI methods to be given later in this chapter (cf. Section 5.7.3).

Example 5.7.5: (Policy Parametrization, Unconventional Problem Structures, 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 approximate VI and PI methodologies. As a result, the idea of policy parametrization applies to problems that share some structure with the finite and infinite horizon DP problems we have discussed so far, but do not admit a formal treatment by DP.

Situations of this type are common in practice. For example, the transition probabilities between states may depend not just on the control, but also on preceding transitions. This dependence may be hard to understand or model mathematically, but may be captured in a simulator that can be used to implement approximation in policy space.

Another important 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 (except through the use of approximations; see, e.g., [BBD08], [Ber19c], [KMP13], [QuL19], [ZYL18], for related work and references). Consequently there is no legitimate DP framework and associated Bellman's equation for this type of problem, and 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 parameter optimization of the type to be described.

In what follows we will discuss three training approaches for approximation in policy space. The first approach (Section 5.7.1) is based on determining the parameter r by optimization of some measure of cost derived from the given DP problem. The second approach (Section 5.7.2) is less ambitious and is reminiscent of supervised learning. Here 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. The third approach (Section 5.7.3) uses approximation in policy space for policy improvement within an actor-only PI framework. Here we apply the expert training approach of Section 5.7.2, with rollout being used as a software expert.

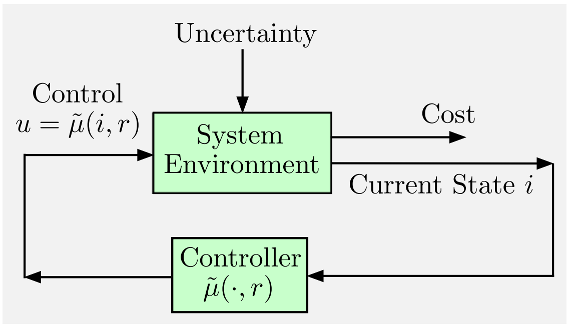


Figure 5.7.3 The optimization framework for approximation in policy space. Here policies are parametrized with a parameter vector r and denoted by $\tilde\mu(r)$, with components $\tilde\mu(i, r)$, $i = 1, \ldots, n$. Each parameter value r determines a policy $\mu(r)$, and a cost $J_{\tilde\mu(r)}(i_0)$ for each initial state i_0, as indicated in the figure. The optimization approach determines r through the minimization

$$\min_r E\big\{J_{\tilde\mu(r)}(i_0)\big\},$$

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

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

In this section we discuss a major training approach for approximation in policy space. It is based on controller parameter optimization: we parametrize the policies by a vector r, and we optimize the corresponding expected cost over r. In particular, we determine r through the minimization

$$\min_r E\big\{J_{\tilde\mu(r)}(i_0)\big\}, \tag{5.60}$$

where $J_{\tilde\mu(r)}(i_0)$ is the cost of the policy $\tilde\mu(r)$ starting from the initial state i_0, and the expected value above is taken with respect to a suitable probability distribution of the initial state i_0 (cf. Fig. 5.7.3).

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

$$r^{k+1} = r^k - \gamma^k \nabla J_{\tilde\mu(r^k)}(i_0), \qquad k = 0, 1, \ldots, \tag{5.61}$$

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

In this section we will focus on alternative and typically more efficient gradient-like methods for stochastic problems, which are based on sampling. Some 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].† We discuss these gradient-like methods next.

Incremental Gradient Methods Based on Randomization

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

$$\min_{z\in Z} F(z), \tag{5.62}$$

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

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

$$\min_{p\in\mathcal{P}_Z} E_p\big\{F(z)\big\}, \tag{5.63}$$

† 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.

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 (5.63) may have some nice differentiability properties that are lacking in the original deterministic version (5.62); see the paper [Ber73] for an analysis of this differentiability issue under convexity assumptions on F.

At this point it is not clear how the stochastic optimization problem (5.63) relates to our stochastic DP context of this chapter. We will return to this question later, but for the purpose of orientation, we note that to obtain a problem of the form (5.63), we must take another unusual step: 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 $\tilde{\mathcal{P}}_Z \subset \mathcal{P}_Z$ of probability distributions $p(z;r)$ that are parametrized by some continuous parameter r, e.g., a finite-dimensional vector in some Euclidean space.† In other words, we approximate the stochastic optimization problem (5.63) with the restricted problem

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

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

$$r^{k+1} = r^k - \gamma^k \nabla\Big(E_{p(z;r^k)}\big\{F(z)\big\}\Big), \qquad k = 0, 1, \ldots, \tag{5.64}$$

where $\nabla(\cdot)$ denotes gradient with respect to r of the function in parentheses, evaluated at the current iterate r^k.

Likelihood-Ratio Policy Gradient Methods

We will first consider an incremental version of the gradient method (5.64). This method requires that $p(z;r)$ is differentiable with respect to r. It relies on a convenient gradient formula, sometimes referred to as the *log-likelihood trick*, which involves the natural logarithm of the sampling distribution.

This formula is obtained by the following calculation, which is based on interchanging gradient and expected value, and using the gradient formula $\nabla(\log p) = \nabla p/p$. We have

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

† To be on safe mathematical ground, we assume that $p(z;r)$ is a discrete distribution in what follows in this section.

$$\begin{aligned} &= \sum_{z\in Z} \nabla p(z;r)F(z) \\ &= \sum_{z\in Z} p(z;r)\frac{\nabla p(z;r)}{p(z;r)}F(z) \\ &= \sum_{z\in Z} p(z;r)\nabla\Big(\log\big(p(z;r)\big)\Big)F(z), \end{aligned}$$

and finally

$$\nabla\Big(E_{p(z;r)}\big\{F(z)\big\}\Big) = E_{p(z;r)}\Big\{\nabla\Big(\log\big(p(z;r)\big)\Big)F(z)\Big\}, \tag{5.65}$$

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

The preceding formula suggests an incremental implementation of the gradient iteration (5.64) that approximates the expected value in the right side in Eq. (5.65) with a single sample (cf. Section 1.3). 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 $\nabla\Big(\log\big(p(z^k;r^k)\big)\Big)$.

(c) Iterate according to

$$r^{k+1} = r^k - \gamma^k\nabla\Big(\log\big(p(z^k;r^k)\big)\Big)F(z^k). \tag{5.66}$$

The advantage of the preceding sample-based method is its simplicity and generality. It allows the use of parametric approximation for *any* minimization problem (well beyond DP), 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 (5.66) r is adjusted along a random direction. This direction does not involve at all the gradient of F, only the gradient of the logarithm of the sampling distribution! 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.

An important issue is the efficient computation of the sampled gradient $\nabla\big(\log\big(p(z^k;r^k)\big)\big)$. 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.

Example 5.7.6 (Policy Gradient Method for Discounted DP)

Consider the α-discounted problem and denote by z the infinite horizon state-control trajectory:

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

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

$$F(z) = \sum_{m=0}^{\infty} \alpha^m g(i_m, u_m, i_{m+1}), \tag{5.67}$$

and the problem is to minimize over r

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

To apply the sample-based gradient method (5.66), 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

$$\nabla\Big(\log\big(p(z^k;r^k)\big)\Big). \tag{5.68}$$

Let us assume that the logarithm of the randomized policy distribution $p(u \mid i;r)$ is differentiable with respect to r (a soft-min policy parametrization is often recommended for this purpose). Then the logarithm that is differentiated in Eq. (5.68) can be written as

$$\begin{aligned}\log\big(p(z^k;r^k)\big) &= \log \prod_{m=0}^{\infty} p_{i_m i_{m+1}}(u_m)\,p(u_m \mid i_m; r^k)\\ &= \sum_{m=0}^{\infty} \log\big(p_{i_m i_{m+1}}(u_m)\big) + \sum_{m=0}^{\infty} \log\big(p(u_m \mid i_m; r^k)\big),\end{aligned}$$

and its gradient (5.68), which is needed in the iteration (5.66), is given by

$$\nabla\Big(\log\big(p(z^k;r^k)\big)\Big) = \sum_{m=0}^{\infty} \nabla\Big(\log\big(p(u_m \mid i_m; r^k)\big)\Big). \tag{5.69}$$

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

The policy gradient method (5.66) can now be implemented with a finite horizon approximation whereby r^k is changed after a finite number N of time steps [so the infinite cost and gradient sums (5.67) and (5.69) are replaced by finite sums]. The method takes the form

$$r^{k+1} = r^k - \gamma^k \sum_{m=0}^{N-1} \nabla \log \big(p(u_m \mid i_m; r^k)\big) F_N(z_N^k),$$

where $z_N^k = (i_0, u_0, \ldots, i_{N-1}, u_{N-1})$ is the generated N-step trajectory, and $F_N(z_N^k)$ is the corresponding cost. The initial state i_0 of the trajectory is chosen randomly, with due regard to exploration issues.

Policy gradient methods for other types of DP problems can be similarly developed, as well as variations involving a combination of policy and cost function approximations [e.g., replacing $F(z)$ of Eq. (5.67) by a parametrized estimate that has smaller variance]. Cost shaping is also useful in this connection (cf. Section 5.3.4). This leads to a class of actor-critic methods that differ from the PI-type methods that we discussed in Section 5.3.1. A further discussion is beyond our scope, and we refer to the end-of-chapter literature for a variety of specific schemes.

Implementation Issues

There are several issues to consider in the implementation of the sample-based gradient method (5.66). The first of these is that the problem solved is a randomized version of the original. If the method produces a parameter $\overline{r}$ in the limit and the distribution $p(z; \overline{r})$ is not atomic (i.e., it is not concentrated at a single point), then a solution $\overline{z} \in Z$ must be extracted from $p(z; \overline{r})$. In the SSP and discounted problems of this chapter, the subset $\tilde{\mathcal{P}}_Z$ 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 collect the samples z^k. Different methods must strike a balance between convenient implementation and a reasonable guarantee that the search space Z is sufficiently well explored.

Finally, there is the issue of improving sampling efficiency. To this end, let us note a simple generalization of the gradient method (5.66), which can often improve its performance. It is based on the gradient formula

$$\nabla\Big(E_{p(z;r)}\big\{F(z)\big\}\Big) = E_{p(z;r)}\Big\{\nabla\Big(\log\big(p(z;r)\big)\Big)\big(F(z) - b\big)\Big\}, \tag{5.70}$$

where b is any scalar. This formula generalizes Eq. (5.65), where $b = 0$, and holds in view of the following calculation, which shows that the term multiplying b in Eq. (5.70) is equal to 0:

$$\begin{aligned} E_{p(z;r)}\Big\{\nabla\Big(\log\big(p(z;r)\big)\Big)\Big\} &= E_{p(z;r)}\left\{\frac{\nabla p(z;r)}{p(z;r)}\right\} \\ &= \sum_{z\in Z} p(z;r)\frac{\nabla p(z;r)}{p(z;r)} \\ &= \sum_{z\in Z} \nabla p(z;r) = \nabla\left(\sum_{z\in Z} p(z;r)\right) = 0, \end{aligned}$$

where the last equality holds because $\sum_{z\in Z} p(z;r)$ is identically equal to 1 and hence does not depend on r.

Based on the gradient formula (5.70), we can modify the iteration (5.66) to read as follows:

$$r^{k+1} = r^k - \gamma^k \nabla\Big(\log\big(p(z^k;r^k)\big)\Big)\big(F(z^k) - b\big), \tag{5.71}$$

where b is some fixed scalar, called the *baseline*. Whereas the choice of b does not affect the gradient $\nabla\Big(E_{p(z;r)}\big\{F(z)\big\}\Big)$ [cf. Eq. (5.70)], it affects the incremental gradient

$$\nabla\Big(\log\big(p(z^k;r^k)\big)\Big)\big(F(z^k) - b\big),$$

which is used in Eq. (5.71). Thus, by optimizing the baseline b, empirically or through a calculation (see e.g., [DNP11]), we can improve the performance of the algorithm. Moreover, for discounted and SSP problems, state-dependent baseline functions may be used, whereby $F(z)$ is replaced in Eq. (5.67) by $F(z) + V(i_0)$, where $V(i_0)$ is a suitably obtained estimate of $F(z)$ or $J^*(i_0)$ (cf. the idea of cost shaping of Sections 4.2 and 5.3.4).

Random Direction Methods

We will now consider an alternative class of incremental versions of the policy gradient method (5.64), repeated here for convenience:

$$r^{k+1} = r^k - \gamma^k \nabla\Big(E_{p(z;r^k)}\big\{F(z)\big\}\Big), \qquad k = 0, 1, \ldots. \tag{5.72}$$

These methods are based on the use of a random search direction and only *two sample function values per iteration*. They are generally faster than methods that use a finite difference approximation of the entire cost function gradient; see the book by Spall [Spa03] for a detailed discussion, and the paper by Nesterov and Spokoiny [NeS17] for a more theoretical view. Moreover they do not require the derivative of the sampling distribution or its logarithm.

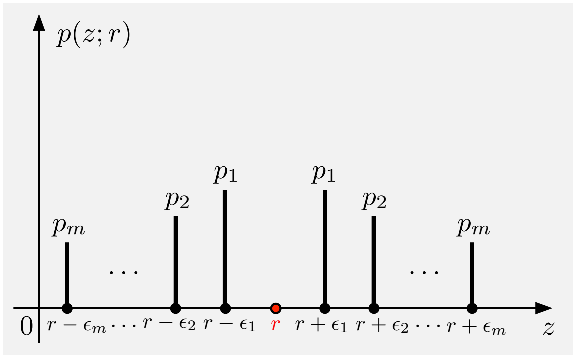


Figure 5.7.4 The distribution $p(z;r)$ used in the gradient iteration (5.73).

We first consider the case where z and r are scalars, and later discuss the multidimensional case. As earlier, to avoid mathematical complications, we assume that $p(z;r)$ is a discrete distribution; the development is similar for more general distributions. 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

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

and

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

The gradient iteration (5.72) becomes

$$r^{k+1} = r^k - \gamma^k\sum_{i=1}^m p_i\big(\nabla F(r+\epsilon_i)+\nabla F(r-\epsilon_i)\big), \qquad k=0,1,\ldots. \tag{5.73}$$

Let us now consider approximation of the gradient by finite differences:

$$\nabla F(r+\epsilon_i) \approx \frac{F(r+\epsilon_i)-F(r)}{\epsilon_i}, \qquad \nabla F(r-\epsilon_i) \approx \frac{F(r)-F(r-\epsilon_i)}{\epsilon_i}.$$

We approximate the gradient iteration (5.73) by

$$r^{k+1} = r^k - \gamma^k\sum_{i=1}^m p_i\frac{F(r^k+\epsilon_i)-F(r^k-\epsilon_i)}{\epsilon_i}, \qquad k=0,1,\ldots. \tag{5.74}$$

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

$$r^{k+1} = r^k - \gamma^k\frac{F(r^k+\epsilon_{i^k})-F(r^k-\epsilon_{i^k})}{\epsilon_{i^k}}, \tag{5.75}$$

where i^k is an index generated with probabilities that are proportional to p_{i^k}. This algorithm uses one out of the m terms of the gradient in Eq. (5.74).

The extension to the case, where z and r are multidimensional, 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 ϵ (independently of d) takes scalar values according to a distribution that is symmetric around 0. The idea is that at r^k, we first choose randomly a direction d^k on the surface of the unit sphere, and then change r^k along d^k or along $-d^k$, depending on the sign of the corresponding directional derivative. For a finite difference approximation of this iteration, we sample z^k along the line $\{r^k + \epsilon d^k \mid \epsilon \in \Re\}$, and similar to the iteration (5.75), we set

$$r^{k+1} = r^k - \gamma^k \frac{F(r^k + \epsilon^k d^k) - F(r^k - \epsilon^k d^k)}{\epsilon^k} d^k, \tag{5.76}$$

where ϵ^k is the sampled value of ϵ.

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 ϵ is a zero mean scalar. Then the analog of iteration (5.76) is

$$r^{k+1} = r^k - \gamma^k \frac{F(r^k + \epsilon^k d^k) - F(r^k)}{\epsilon^k} d^k, \tag{5.77}$$

where ϵ^k is the sampled value of ϵ. Thus in this case, we still require two function values per iteration. Generally, for a symmetric sampling distribution, iteration (5.76) tends to be more accurate than iteration (5.77), and is often preferred.

Algorithms of the form (5.76) and (5.77) are known as *random direction 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. There is some freedom in selecting d^k, which could potentially be exploited in specific schemes. However, selecting the stepsize γ^k and the sampling distribution for ϵ can be tricky, particularly when the values of F are noisy.

Random Search and Cross-Entropy Methods

The main drawback of the policy gradient methods that we have considered in this section 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

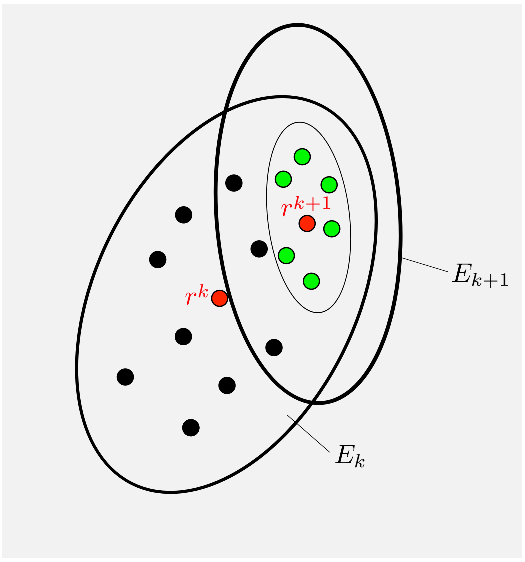


Figure 5.7.5 Schematic illustration of the cross-entropy method. At the current iterate r^k, we construct an ellipsoid E_k centered at r^k. We generate a number of random samples within E_k, and we "accept" a subset of the samples that have "low" cost. We then choose r^{k+1} to be the sample mean of the accepted samples, and construct a sample "covariance" matrix of the accepted samples. We then form the new ellipsoid E_{k+1} using this matrix and a suitably enlarged radius, and continue. Notice the resemblance with a policy gradient method: we move from r^k to r^{k+1} in a direction of cost improvement.

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.

Let us consider a parametric policy optimization approach based on solving the problem

$$\min_r E\big\{J_{\tilde{\mu}(r)}(i_0)\big\},$$

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

The method, when adapted to the approximate DP context, bears resemblance to policy gradient methods, in that it generates a parameter sequence $\{r^k\}$ by changing r^k to r^{k+1} along a direction of "improvement." This direction is obtained by using the policy $\tilde{\mu}(r^k)$ to generate randomly cost samples corresponding to a set of sample parameter values that are concentrated around r^k. The current set of sample parameters are then screened: some are accepted and the rest are rejected, based on a cost improvement criterion. Then r^{k+1} is determined as a "central point" or as the "sample mean" in the set of accepted sample parameters, some more samples are generated randomly around r^{k+1}, and the process is repeated; see Fig. 5.7.5. Thus successive iterates r^k are "central points" of successively better groups of samples, so in some broad sense, the random sample generation process is guided by cost improvement. This idea is shared with evolutionary programming; see e.g., the books [Bac96], [DeJ06].

The cross-entropy method is very simple to implement, does not suffer from the fragility of gradient-based optimization, does not involve randomized policies, and relies on some supportive theory. Importantly, 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, the convergence rate guarantees of the cross-entropy method are limited, and its success depends on domain-specific insights and the skilled use of heuristics. However, the method relies on solid ideas and has gained a favorable reputation. In particular, it was used with impressive success in the context of the game of tetris; see Szita and Lorinz [SzL06], and Thiery and Scherrer [ThS09]. There have also been reports of domain-specific successes with alternative but related random search methods; see Salimans et al. [SHC17]. We refer to the end-of-chapter literature for details and examples of implementation.

5.7.2 Expert-Based Supervised Learning

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†

$$\min_r \sum_{s=1}^{q} \big\| u^s - \tilde{\mu}(i^s, r) \big\|^2 \tag{5.78}$$

(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 2.4.3, in the context of finite horizon problems, and in Section 3.5 in the context of classification 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.

for approximation in policy space. In the context of artificial intelligence, it comes within the framework of *supervised learning* methods.†

Another possibility, which we have already discussed in various forms, is to select a large number of sample states i^s, $s = 1, \ldots, q$, and generate the controls u^s, $s = 1, \ldots, q$, through the one-step lookahead minimization

$$u^s = \arg \min_{u \in U(i^s)} \sum_{j=1}^{n} p_{ij}(u)\big(g(i^s, u, j) + \alpha \tilde{J}(j)\big), \tag{5.79}$$

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

$$u^s = \arg \min_{u \in U(i^s)} \tilde{Q}(i^s, u, r). \tag{5.80}$$

In this case, we will be collecting sample state-control pairs (i^s, u^s), $s = 1, \ldots, q$, using approximation in value space through Eq. (5.79) or Eq. (5.80), and then applying approximation in policy space through Eq. (5.78).

Note that once the sample state-control pairs (i^s, u^s), $s = 1, \ldots, q$, have been collected, an alternative to solving the least squares problem (5.78) 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

$$\tilde{\mu}(i) = \sum_{s=1}^{q} \phi_{is} u^s, \qquad i = 1, \ldots, n. \tag{5.81}$$

In general, this requires that the control constraint set is a convex subset of a Euclidean space so that the interpolated controls (5.81) are feasible. This is not necessary if all the interpolation probabilities ϕ_{is} are either 0 or 1 (an example in the *nearest neighbor approach*, where for a given state i, we set $\phi_{i\bar{s}} = 1$ for the state $i^{\bar{s}}$ that is "closest" to i in some sense). Interpolation approaches are central to the aggregation methodology of Chapter 6, and will be discussed in greater detail there.

† Tesauro [Tes89a], [Tes89b] constructed a backgammon player, trained by a neural network and a supervised learning approach (called "comparison training"), 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.

Of course in the expert training approach we cannot expect to obtain a policy that performs better than the expert with which it is trained, in the case of Eq. (5.78), or the one-step lookahead policy that is based on the approximation $\tilde{J}$ or $\tilde{Q}$, in the case of Eq. (5.79) or Eq. (5.80), respectively. However, a major advantage is that once the parametrized policy is obtained, its on-line implementation is fast and does not involve extensive calculations such as minimizations of the form (5.79). This advantage is generally shared by schemes that are based on approximation in policy space. Moreover the policy obtained by emulating the expert can be improved through rollout.

5.7.3 Approximate Policy Iteration, Rollout, and Approximation in Policy Space

In this section we revisit approximate PI, but with a view towards combining it with rollout and approximation in policy space. We describe how approximation in policy space can provide the basis for an alternative PI implementation, namely *approximate the generated policies directly instead of approximating their cost functions or Q-factors*. This is an *actor-only* method, as opposed to the approximate PI methods of Sections 5.3.2 and 5.3.3, which involve a critic component.

In particular, the approximate PI methods given in Sections 5.3.2 and 5.3.3 use approximate policy evaluation (approximation in value space to represent the cost function or Q-factors of the current policy) followed by fairly exact policy improvement through one-step or multistep lookahead. By contrast, the methods described in this section use fairly exact policy evaluation through the use of rollout for a sample set of states, followed by approximate policy improvement [representation of the improved (or rollout) policy using a policy architecture]. The idea is to view the PI algorithm as a *perpetual rollout process*, which uses at any one time 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 μ, which we use as the base policy for generating by (possibly truncated) rollout many state-control sample pairs (i^s, u^s), $s = 1, \ldots, q$ (cf. the rollout algorithm of Section 5.1.2).† We then obtain an "improved" policy $\tilde{\mu}(i, \overline{r})$, using an approximation architecture (such

† The approximation of the rollout policy using state control samples is subject to exploration issues, i.e., the choice of states starting from which we generate rollout trajectories with a given base policy. In particular, it is important to include in the sample set of initial states i^s, $s = 1, \ldots, q$, a subset of states that are "favored" by the rollout trajectories; e.g., start from some initial subset of states i^s and selectively add to this subset states that are encountered along the rollout trajectories.

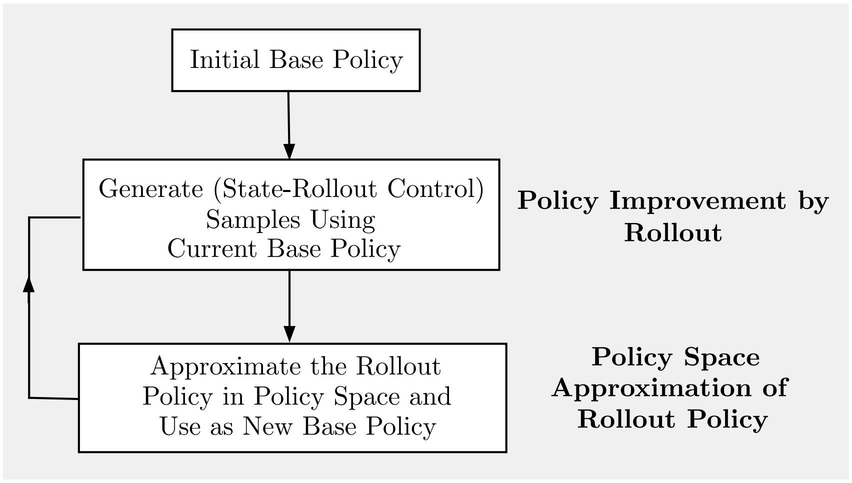

Initial Base Policy
Generate (State-Rollout Control) Samples Using Current Base Policy
Policy Improvement by Rollout
Approximate the Rollout Policy in Policy Space and Use as New Base Policy
Policy Space Approximation of Rollout Policy

Figure 5.7.6 Block diagram of approximate PI based on policy improvement by rollout and policy approximation in policy space. Here the policy evaluation and policy improvement processes may be exact, but the implementation of the "improved" policy is approximate, using approximation in policy space and regression.

as a neural network), where the parameter $\overline{r}$ is obtained from the least squares/regression minimization

$$\overline{r} \in \arg\min_{r} \sum_{s=1}^{q} \big\| u^s - \tilde{\mu}(i^s, r) \big\|^2 \tag{5.82}$$

(possibly with added regularization); see Fig. 5.7.6. The "improved" policy $\tilde{\mu}(i, \overline{r})$ is then used as a base policy to generate samples of the corresponding rollout policy, which is then approximated in policy space, etc. This is similar to the expert training approach of Section 5.7.2 [cf. Eq. (5.78)]; we just use the rollout policy as the "expert" and emulate its decisions using sampling and supervised learning. The training problem (5.82) relates to the ones solved in classification problems (often with the use of neural networks or other architectures), and can be addressed with similar algorithms: the parameter $\overline{r}$ defines a classifier, which given a state i, classifies i as requiring control $\tilde{\mu}(i, \overline{r})$; 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 PI process must be performed off-line. However, it can take advantage of parallelization and a partitioned architecture, whereby the state

† A simpler alternative to parametric approximation and the least squares minimization (5.82) is the interpolation scheme (5.81).

space is appropriately divided into subsets, and the rollout policy is approximated separately within each subset. 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. Alternatively, once the final policy has been obtained off-line, it can be used on-line as a base policy for generation of rollout controls, thus allowing on-line replanning.

Favorable Special Cases - Linear Quadratic Optimal Control Without a Mathematical Model

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 μ that can be represented by the approximation architecture, i.e., have the form $\mu = \tilde{\mu}(r)$ 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 $\mathcal{M}$ (i.e., exact policy improvement produces a policy in $\mathcal{M}$). 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 (5.82) (so that the rollout policy is the same as the one produced by exact policy improvement) 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 supports the conjecture that if the set $\mathcal{M}$ "nearly contains" the policies produced by exact PI, then 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 (so perfect state information is assumed). 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 also converges to an optimal policy for this choice of $\mathcal{M}$, and works without requiring a model of the system.

† We have not discussed in this book infinite horizon problems with continuous state and control spaces. While such problems can be challenging in general, the linear quadratic case is well behaved, and is supported by a solid VI and PI methodology; see e.g., [Ber12], [Ber17]. In our discussion here, we simply quote results from this methodology without a proof elaboration.

Rollout and Approximate PI for POMDP

We will now discuss the application of the rollout-based approximate PI algorithm to the partially observable version of the α-discounted problem. Here at each stage, instead of observing the current state, we obtain an observation z with probabilities $p(z \mid j, u)$, where j is the current state, and u is the preceding control. We assume that at each stage, the belief state $b = \big(b(1), \ldots, b(n)\big)$ is available, where $b(i)$ is the conditional probability that the state is i, given the past history, and that we can also compute $B(b, u, z)$, the next belief state, given that the current belief state is b, the control applied at b is u, and that the next measurement is z. We can view $B(b, u, z)$ as a *belief state generator*. Finally, we assume that the initial base policy and all the base policies generated by the policy space scheme (5.82) are functions of b.

Let us consider the computation of the rollout control $\tilde\mu(b)$ at some fixed b, using the truncated rollout algorithm, with μ being the current base policy (some function of the belief state), and $\tilde J$ being some cost function approximation. This rollout control is given by

$$\tilde\mu(b) \in \arg\min_{u \in U} \sum_{i=1}^{n} b(i) \tilde F_\mu(i, b, u), \tag{5.83}$$

where $\tilde F_\mu(i, b, u)$ can be viewed as an approximate Q-factor, obtained by starting from state i, applying control u, then applying the policy μ for some number m of stages, and finally approximating the cost of the remaining stages using the function $\tilde J$. [A multistep lookahead version of Eq. (5.83) is also possible.] For given values of b and u, the expression minimized in Eq. (5.83) can be approximated by simulation. Here is a possibility, which involves *simultaneous simulation of the states of the original system, as well as the belief states obtained by the belief state generator*; the corresponding simulator is illustrated in Fig. 5.7.7.

The state space is sampled according to the distribution b to obtain a dataset of states $\mathcal{I} = \{i^m \mid m = 1, \ldots, M\}$ (if n is relatively small, we can take $\mathcal{I} = \{1, \ldots, n\}$). Then for each state in the dataset, say i^m, and each control $u \in U$, we do the following: We obtain the next state j_1 according to the transition probabilities $p_{i^m j}(u)$, and we record the cost $g(i^m, u, j_1)$. Then we obtain a random observation sample z_1 according to $p(z_1 \mid j_1, u)$, and compute the next belief state $b_1 = B(b, u, z_1)$ [which allows us to compute the next control $\mu(b_1)$ for the next simulation step]. In a similar manner, we run a simulation trajectory starting from j_1 for m stages, using policy μ, while obtaining a sequence of m successive control-next state-observation-belief state quadruplets

$$\big(\mu(b_1), j_2, z_2, b_2\big), \ldots, \big(\mu(b_m), j_{m+1}, z_{m+1}, b_{m+1}\big);$$

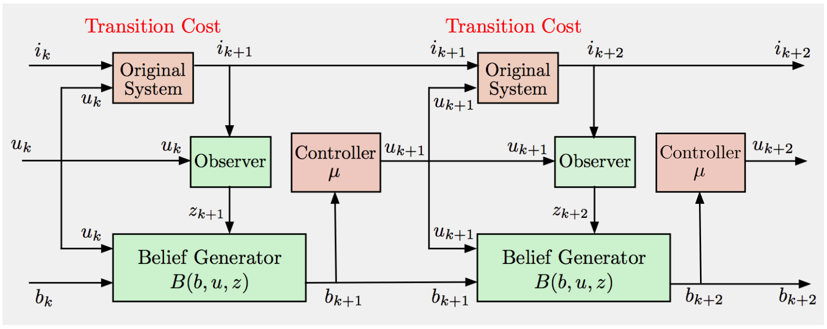


Figure 5.7.7 Block diagram of simultaneous simulation of the states of the original system, and the belief states obtained by the belief state generator, under a base policy $\mu(b)$.

cf. Fig. 5.7.7. Simultaneously, we obtain the corresponding sum of stage costs plus the terminal (approximate) cost

$$g(i^m,u,j_1)+\alpha g\big(j_1,\mu(b_1),j_2\big)+\cdots+\alpha^m g\big(j_m,\mu(b_m),j_{m+1}\big)+\alpha^{m+1}\tilde J(b_{m+1}).$$

In this way we obtain a single sample of $\tilde F_\mu(i^m,b,u)$ that corresponds to the state-control pair (i^m,u).

By averaging many such random samples for every state $i^m\in\mathcal{I}$, we compute an estimate of $\sum_{i=1}^n b(i)\tilde F_\mu(i,b,u)$ for each $u\in U$. By minimizing this estimate over u, we obtain the rollout control $\tilde\mu(b)$ at b; cf. Eq. (5.83). After collecting q such sample belief-rollout control pairs $\big(b^s,\tilde\mu(b^s)\big)$, $s=1,\ldots,q$, we can generate a new base policy by using approximation in policy space [cf. Eq. (5.82)]. The computations are similar to the perfect state information case, except that obtaining each belief-rollout control pair through the minimization of Eq. (5.83) requires one additional operation: sampling the state space using the current belief state b and carrying out the computations for each pair (i^m,u) with $i^m\in\mathcal{I}$ and $u\in U$, rather than a single state-control pair. Still, a lot more computation may be needed relative to the perfect state information case, depending on the number of states i such that $b(i)>0$. This argues for small lengths of lookahead minimization and rollout, and greater emphasis on the terminal cost function approximation $\tilde J$.

Variations

There are many variations of the actor-only approximate PI scheme of this section. Basically, all the variants of rollout (multistep lookahead, rollout truncation, and terminal cost function approximation) are applicable. Moreover, optimistic variants of PI can be used, whereby only a small number of samples q are generated between changes in the parameter vector and the attendant change of base policy. To implement such an optimistic variant, one may use an incremental gradient or Newton method (cf. Section 3.1.3) to solve the regression minimization (5.82) while adding new terms to the least squares sum as new samples (i^s, u^s) become available, with u^s being the control obtained with the current policy (the one that corresponds to the current value of r). Naturally, as in all approximate PI methods, exploration is an important issue to address, using a judicious choice of the sampled states i^s.

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

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

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

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

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

(3) We update the parameter vector via

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

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

As in the case of SARSA, there are also less optimistic variants of the preceding algorithm, whereby several states and their rollout controls are computed before updating the parameter vector r.

5.8 NOTES AND SOURCES

Section 5.1: The performance bound of Props. 5.1.1 for multistep lookahead with a terminal cost function approximation is well known; see e.g., Prop. 6.1.1 in the author's DP textbook [Ber17] (and earlier editions), as well as [Ber18a], Section 2.2. The bound of Prop. 5.1.3 for multistep lookahead, truncated rollout, and terminal cost function approximation is new.

The bounds of Section 5.1.3 for approximate PI were first given in the neuro-dynamic book by Bertsekas and Tsitsiklis [BeT96], Sections 6.2.2 and 6.2.3, for both discounted and SSP problems. These bounds, together with related bounds for approximate optimistic PI, due to Thiery and Scherrer [ThS10], are quite fundamental. While they are often conservative, they correctly describe the qualitative behavior of approximate PI and its variations, and they differentiate the behavior of these methods from their VI counterparts, which are generally unstable in the absence of a suitable sampling policy (cf. Example 5.2.1). Extensions of these bounds to approximate PI for general spaces abstract DP problems under contraction and monotonicity assumptions can be found in the author's abstract DP book [Ber18a], Sections 2.4.1 and 2.5.2.

Section 5.2: Fitted VI algorithms have been used for finite horizon problems since the early days of DP. They are conceptually simple and easily implementable, and they are used widely for approximation of either optimal costs or Q-factors (see e.g., the papers by Gordon [Gor95], Longstaff and Schwartz [LoS01], Ormoneit and Sen [OrS02], Ernst, Geurts, and Wehenkel [EGW06], Antos, Munos, and Szepesvari [AMS07], and Munos and Szepesvari [MuS08]). Approximations to VI may also be implemented by relaxing the constraints of the iteration and/or introducing bounding operations that simplify the calculations; see e.g., Lincoln and Rantzer [LiR06], and Spaan and Vlassis [SpV05] in the context of POMDP.

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

Section 5.4: Q-learning was proposed by Watkins [Wat89], and had a major impact in the development of the RL field. A rigorous convergence proof of Q-learning was given by Tsitsiklis [Tsi94], in a more general framework that combined several ideas from stochastic approximation theory and the theory of distributed asynchronous computation. This proof covered discounted problems, and SSP problems where all policies are proper. It also covered SSP problems with improper policies, assuming that the Q-learning iterates are either nonnegative or bounded. Convergence for SSP problems without the nonnegativity or the boundedness assumption was shown by Yu and Bertsekas [YuB13b]. Q-learning for optimal stopping problems was analyzed by Tsitsiklis and Van Roy [TsV99b], with followup work by Yu and Bertsekas [YuB07].

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 the Q-learning methods of [BeY12], [YuB13a] is that the policy evaluation process aims towards the solution of an optimal stopping problem rather than towards the solution of the Bellman equation associated with the policy; this is needed to avoid the pathological behavior identified by Williams and Baird [WiB93], and noted earlier in Section 4.6.3.

The original proposal of SARSA is attributed to Rummery and Niranjan [RuN94], with related work reported in the papers by Peng and Williams [PeW96], and Wiering and Schmidhuber [WiS98]. The ideas of the DQN algorithm attracted much attention following the paper by Mnih et al. [MKS15], which reported impressive test results on a suite of 49 classic Atari 2600 games.

The advantage updating idea, which was noted in the context of finite horizon problems in Section 3.4, 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 PI and Q-learning, called *differential training* and aiming to approximate cost-to-go differences rather than cost-to-go values, has been proposed by the author in [Ber97b]; see also Weaver and Baxter [WeB99].

Section 5.5: Projected equations underlie Galerkin methods, which have a long history in scientific computation. These methods are used widely for approximate solution of many types of large-scale problems, including linear systems arising from discretization of partial differential and integral equations. The connection of the approximate DP context of policy evaluation based on projected equations with Galerkin methods was first discussed by Yu and Bertsekas [YuB10], and Bertsekas [Ber11b]. 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 the papers by Bertsekas and Yu [BeY09], Wang and Bertsekas [WaB13a], [WaB13b], and the author's textbook [Ber12], Section 7.3, which discuss the application of TD methods to the solution of general linear systems of equations).

Temporal difference ideas 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 the use of temporal differences and proposed the TD(λ) method. This was a major development, and motivated a lot of research in RL and simulation-based DP, particularly following an impressive early success with the backgammon playing program of Tesauro [Tes92], [Tes94]. The relation of TD(λ) and the solution of projected equations was clarified in the works of Tsitsiklis and Van Roy [TsV97], [TsV99b], and was also described in the neuro-dynamic programming book [BeT96].

The three methods TD(λ), LSTD(λ), and LSPE(λ) are discussed in detail in the journal and textbook RL literature. For a discussion that extends our presentation of Section 5.5, see Chapters 6 and 7 of the book

[Ber12].

The convergence of TD(λ) was proved by Tsitsiklis and Van Roy [TsV97], with extensions in [TsV99a] and [TsV99b]. The author's papers [Ber16c], [Ber18d] describe the connection of TD and proximal algorithms, a central methodology in convex optimization. In particular, in these works, TD(λ) is shown to be a stochastic version of the proximal algorithm for solving linear systems of equations, and extensions of TD(λ) for solving nonlinear systems of equations are described (the parameter λ is related to the penalty parameter of the proximal algorithm).

The LSTD(λ) 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(λ) under assumptions of increasing generality were given by Nedić, Bertsekas, and Yu [NeB03], [BeY09], [Yu12].

The LSPE(λ) algorithm was first proposed by Bertsekas and Ioffe [BeI96] under the name *λ-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 was to provide a better alternative to TD(λ)-based PI, which failed within the tetris context. LSPE(λ) was also given in the book [BeT96], Section 2.3.1, with subsequent contributions, including versions that use diagonal scaling to avoid matrix inversion, by Nedić, Borkar, Yu, Scherrer, and the author [NeB03], [BBN04], [YuB07], [BeY09], [YuB09a], [Ber11a], [Ber11b], [Yu12], [Sch13], [Ber18a].

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

Section 5.6: The linear programming approach for exact infinite horizon DP was proposed by D'Epenoux [D'Ep60]. Approximation methods using basis functions and linear programming were suggested with little analysis by Schweitzer and Seidman [ScS85], and have been further developed by de Farias and Van Roy [DFV03], [DFV04], [DeF04]. 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], Desai, Farias, and Moallemi [DFM12], [DFM13], 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.

Section 5.7: Our coverage of approximation in policy space, policy gradient, and random search methods has been limited, and aimed to provide an entry point into the field. For a detailed discussion and references on policy gradient methods, we refer to the book by Sutton and Barto [SuB18], the monographs by Deisenroth, Neumann, and Peters [DNP11], and the surveys by Peters and Schaal [PeS08], and Grondman et al. [GBL12]. An influential 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 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 a baseline and variance reduction methods (Greensmith, Bartlett, and Baxter [GBB04], Greensmith [Gre05]), and scaling based on the so-called *natural gradient* (Kakade [Kak02]) or second order information (see Wang and Paschalidis [WaP17], and the references quoted there). 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].

We have not covered actor-critic methods within the policy gradient context. Such methods were introduced in the paper by Barto, Sutton, and Anderson [BSA83]. 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 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 5.7.2 are similar to the comparison training method discussed in Section 2.4.3, which was proposed by Tesauro [Tes89a], [Tes89b], [Tes01]. Methods that use learning from data generated by an expert are often referred in the literature by the names *imitation learning* and *apprenticeship learning*; see e.g., Abbeel and Ng [AbN04], Neu and Szepesvari [NeS12], and Schaal [Sch99].

Training using a human expert has generated considerable interest in robotics (where it is often referred to as *learning from demonstration*); see Argall et al. [ACV09], and for some recent work, see Ben Amor et al. [BVE13] and Lee [Lee17]. For a recent analysis of related possibilities, see Hanawal et al. [HLZ18].

The actor-only PI/rollout methods of Section 5.7.3, depending on their implementation, are similar to several methods proposed in the literature, first by Lagoudakis and Parr [LaP03], and then by 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.

Our discussion of the application to POMDP is new in the form given here. Generally POMDP are quite challenging for the current RL methodology, both theoretically and practically, because of the continuous nature of the belief space as well as the excessive accumulation of potentially useful information over time.

The main strength of the actor-only 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 mature and 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.

We have noted in Section 5.7.3 the application of PI with approximation in policy space for adaptive control in problems with a linear-quadratic structure and perfect state information. 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 perfect state information, and an infinite number of simulation samples), 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].

5.9 APPENDIX: MATHEMATICAL ANALYSIS

In this appendix we provide proofs of the mathematical results stated in this chapter. We also prove some supplementary results that are described in the chapter without formal statement.

We will use extensively the triangle inequality $\|J+J'\| \leq \|J\| + \|J'\|$, which holds for every norm $\|\cdot\|$. We will also use the Bellman operators T and T_μ for the discounted problem:

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

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

A key property for our analysis is that these operators are contractions, i.e., for all J, J', and μ, we have

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

where $\|J\|$ is the maximum norm $\|J\| = \max_{i=1,\ldots,n} \big|J(i)\big|$, cf. Prop. 4.3.5. Another key property is the monotonicity of these operators, i.e.,

$$TJ \geq TJ', \quad T_\mu J \geq T_\mu J', \qquad \text{for all } J \text{ and } J' \text{ with } J \geq J'.$$

Moreover, 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 αc.

5.9.1 Performance Bounds for Multistep Lookahead

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

Proposition 5.1.1: (Limited Lookahead Performance Bounds)

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

$$\|J_{\tilde\mu} - J^*\| \leq \frac{2\alpha^\ell}{1-\alpha} \|\tilde J - J^*\|, \tag{5.84}$$

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

(b) Define

$$\hat J(i) = \min_{u \in \overline{U}(i)} \sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j) + \alpha \tilde J(j)\big), \qquad i = 1, \ldots, n, \tag{5.85}$$

where $\overline{U}(i) \subset U(i)$ for all $i = 1, \ldots, n$, and let $\tilde\mu$ be the one-step lookahead policy obtained by minimization in the right side of this equation. Then we have

$$J_{\tilde{\mu}}(i) \le \tilde{J}(i) + \frac{c}{1-\alpha}, \qquad i = 1, \ldots, n, \tag{5.86}$$

where

$$c = \max_{i=1,\ldots,n} \big(\hat{J}(i) - \tilde{J}(i)\big). \tag{5.87}$$

Proof: (a) The line of proof of this part is to show Eq. (5.84) 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_μ (cf. Prop. 4.3.5).

We first prove a preliminary relation. Using the fact

$$\|T_{\tilde{\mu}}^m J^* - T_{\tilde{\mu}}^{m-1} J^*\| = \|T_{\tilde{\mu}}^{m-1}(T_{\tilde{\mu}} J^*) - T_{\tilde{\mu}}^{m-1} J^*\| \le \alpha^{m-1} \|T_{\tilde{\mu}} J^* - J^*\|,$$

and the triangle inequality, we have for every k,

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

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

$$\|J_{\tilde{\mu}} - J^*\| \le \frac{1}{1-\alpha} \|T_{\tilde{\mu}} J^* - J^*\|. \tag{5.88}$$

Denote now $\bar{J} = T^{\ell-1}\tilde{J}$. By using the triangle inequality and the definition of $\tilde{\mu}$, which implies that $T_{\tilde{\mu}}\bar{J} = T\bar{J}$, the rightmost expression of Eq. (5.88) is estimated as follows:

$$\begin{aligned}
\|T_{\tilde{\mu}} J^* - J^*\| &\le \|T_{\tilde{\mu}} J^* - T_{\tilde{\mu}}\bar{J}\| + \|T_{\tilde{\mu}}\bar{J} - T\bar{J}\| + \|T\bar{J} - J^*\| \\
&= \|T_{\tilde{\mu}} J^* - T_{\tilde{\mu}}\bar{J}\| + \|T\bar{J} - TJ^*\| \\
&\le 2\alpha \|\bar{J} - J^*\| \\
&= 2\alpha \|T^{\ell-1}\tilde{J} - T^{\ell-1} J^*\| \\
&\le 2\alpha^\ell \|\tilde{J} - J^*\|.
\end{aligned}$$

By combining the preceding two relations, we obtain Eq. (5.84).

(b) Let us denote by e the unit vector whose components are all equal to 1. Then by the definition (5.87) of c, we have

$$T_{\tilde{\mu}}\tilde{J} = \hat{J} \le \tilde{J} + ce.$$

Applying $T_{\tilde{\mu}}$ to both sides of this relation, and using the monotonicity and constant shift property of $T_{\tilde{\mu}}$, we obtain

$$T_{\tilde{\mu}}^2 \tilde{J} \le T_{\tilde{\mu}}\tilde{J} + \alpha ce.$$

Continuing similarly, we have

$$T_{\tilde{\mu}}^{k+1}\tilde{J} \le T_{\tilde{\mu}}^{k}\tilde{J} + \alpha^k ce, \qquad k = 0, 1, \ldots.$$

Adding the first $k+1$ of these relations, we obtain

$$T_{\tilde{\mu}}^{k+1}\tilde{J} \le \tilde{J} + (1 + \alpha + \cdots + \alpha^k)ce, \qquad k = 0, 1, \ldots.$$

Taking the limit as $k \to \infty$, and using the fact $T_{\tilde{\mu}}^{k+1}\tilde{J} \to J_{\tilde{\mu}}$, we obtain the desired inequality (5.86). **Q.E.D.**

5.9.2 Performance Bounds for Rollout

We next show the basic cost improvement property of rollout.

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

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

where μ is a base policy [cf. Eq. (5.85) with $\tilde{J} = J_\mu$] and we assume that $\mu(i) \in \overline{U}(i) \subset U(i)$ for all $i = 1, \ldots, n$. Then $J_{\tilde{\mu}} \le J_\mu$.

Proof: Let us denote

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

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

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

where the equality on the right holds by Bellman's equation. Thus we have $c = \max_{i=1,\ldots,n}\big(\hat{J}(i) - J_\mu(i)\big) \le 0$, and the result follows from Prop. 5.1.1(b) with $\tilde{J} = J_\mu$ [cf. Eq. (5.86)]. **Q.E.D.**

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

Proposition 5.1.3: (Performance Bound of Truncated Rollout with Terminal Cost Function Approximation) Let ℓ and m be positive integers, let μ be a policy, and let $\tilde{J}$ be a function of the state. Consider a truncated rollout scheme consisting of ℓ-step lookahead, followed by rollout with a policy μ 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_μ is the Bellman operator of Eq. (5.4), and $\|\cdot\|$ denotes the maximum norm $\|J\| = \max_{i=1,\ldots,n} \big|J(i)\big|$.

(b) We have

$$J_{\tilde{\mu}}(i) \leq \tilde{J}(i) + \frac{c}{1-\alpha}, \qquad i = 1, \ldots, n, \tag{5.89}$$

where

$$c = \max_{i=1,\ldots,n} \big((T_\mu \tilde{J})(i) - \tilde{J}(i)\big).$$

(c) We have

$$J_{\tilde{\mu}}(i) \leq J_\mu(i) + \frac{2}{1-\alpha}\|\tilde{J} - J_\mu\|, \qquad i = 1, \ldots, n. \tag{5.90}$$

Proof: (a) This is simply Prop. 5.1.1(a) adapted to the truncated rollout scheme [$\tilde{J}$ is replaced in Eq. (5.84) by $T_\mu^m \tilde{J}$].

(b) We first assume that $c = 0$ and prove the result for this case. If $c = 0$, by the definition of c, we have $\tilde{J} \geq T_\mu \tilde{J}$, which together with the monotonicity of T and T_μ, implies that

$$\tilde{J} \geq T_\mu^m \tilde{J} \geq T_\mu^{m+1} \tilde{J} \geq T T_\mu^m \tilde{J} \geq T^{\ell-1} T_\mu^m \tilde{J} \geq T^\ell T_\mu^m \tilde{J} = T_{\tilde{\mu}} T^{\ell-1} T_\mu^m \tilde{J}. \tag{5.91}$$

By writing this inequality for just the first, fifth, and last expressions above, we have

$$\tilde{J} \geq T^{\ell-1} T_\mu^m \tilde{J} \geq T_{\tilde{\mu}} T^{\ell-1} T_\mu^m \tilde{J}.$$

The right side of the preceding relation and the monotonicity of $T_{\tilde{\mu}}$, imply that the sequence $\{T_{\tilde{\mu}}^k T^{\ell-1} T_\mu^m \tilde{J}\}$ is monotonically nonincreasing as k increases, while the left side shows that the sequence is bounded above by $\tilde{J}$.

Since by the convergence property of VI, this sequence converges to $J_{\tilde{\mu}}$ as $k \to \infty$, we obtain $\tilde{J} \geq J_{\tilde{\mu}}$. Thus the result follows for the case $c = 0$.

To prove the result for general c, we introduce the function J' given by

$$J' = \tilde{J} + \frac{c}{1-\alpha} e,$$

where e is the unit vector whose components are all equal to 1. Applying T_μ to this equation, we have

$$T_\mu J' = T_\mu \tilde{J} + \frac{\alpha c}{1-\alpha} e,$$

while from the definition of c, it follows that

$$T_\mu \tilde{J} \leq \tilde{J} + ce.$$

Combining the preceding two relations, we obtain

$$T_\mu J' \leq \tilde{J} + ce + \frac{\alpha c}{1-\alpha} e = \tilde{J} + \frac{c}{1-\alpha} e = J',$$

or $T_\mu J' \leq J'$. The rollout policy $\tilde{\mu}$ does not change if we replace $\tilde{J}$ with J', since these two functions differ by a constant. Thus by using the version of the result already proved for $c = 0$ and the fact $T_\mu J' \leq J'$, we have $J_{\tilde{\mu}} \leq J'$, which is equivalent to the desired Eq. (5.89).

(c) Let $c = \|\tilde{J} - J_\mu\|$, so that $J_\mu \leq \tilde{J} + ce$ and $T_\mu \tilde{J} \leq T_\mu J_\mu + \alpha ce = J_\mu + \alpha ce$. Using these relations, it follows that

$$T_\mu \tilde{J} \leq J_\mu + \alpha ce \leq \tilde{J} + ce + \alpha ce.$$

By using part (b) and the fact $\tilde{J} \leq J_\mu + ce$ (which follows from the definition of c), we obtain

$$J_{\tilde{\mu}} \leq \tilde{J} + \frac{c + \alpha c}{1-\alpha} e \leq J_\mu + ce + \frac{c+\alpha c}{1-\alpha} e = J_\mu + \frac{2c}{1-\alpha} e.$$

Q.E.D.

A version of part (b) for the case where $m = 0$ is that if $T\tilde{J} = T_{\tilde{\mu}} \tilde{J} \leq \tilde{J} + ce$, then $J_{\tilde{\mu}} \leq \tilde{J} + \frac{c}{1-\alpha} e$. This follows by using induction to show that for all k we have

$$T_{\tilde{\mu}}^k \tilde{J} \leq \tilde{J} + (1 + \alpha + \cdots + \alpha^{k-1}) ce,$$

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

5.9.3 Performance Bounds for Approximate Policy Iteration

To prove the performance bound of Prop. 5.1.4, we focus on the discounted problem, and we make use of the contraction property of T_μ. We want to prove the following performance bound.

Proposition 5.1.4: (Performance Bound for Approximate PI) Consider the discounted problem, and let $\{\mu^k\}$ be the sequence generated by the approximate PI algorithm defined by the approximate policy evaluation (5.11) and the approximate policy improvement (5.12). 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. Note that this lemma relates to rollout, i.e., a single policy iteration that produces a rollout policy $\tilde{\mu}$ starting from a base policy μ.

Lemma 5.9.1: (Error Bound for Rollout with Approximations) Consider the discounted problem, and let J, $\tilde{\mu}$, and μ satisfy

$$\|J - J_\mu\| \leq \delta, \qquad \|T_{\tilde{\mu}} J - TJ\| \leq \epsilon, \tag{5.92}$$

where δ and ϵ are some scalars. Then we have

$$\|J_{\tilde{\mu}} - J^*\| \leq \alpha\|J_\mu - J^*\| + \frac{\epsilon + 2\alpha\delta}{1-\alpha}. \tag{5.93}$$

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

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

and hence

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

where e is the unit vector whose components are all equal to 1. Using also Eq. (5.92), we have

$$T_{\tilde{\mu}} J_\mu \leq T_{\tilde{\mu}} J + \alpha\delta\, e \leq TJ + (\epsilon + \alpha\delta)e \leq TJ_\mu + (\epsilon + 2\alpha\delta)e. \tag{5.94}$$

Combining this inequality with $TJ_\mu \leq T_\mu J_\mu = J_\mu$, we obtain

$$T_{\tilde{\mu}} J_\mu \leq J_\mu + (\epsilon + 2\alpha\delta)e. \tag{5.95}$$

We will show that this relation implies that

$$J_{\tilde{\mu}} \le J_\mu + \frac{\epsilon + 2\alpha\delta}{1-\alpha} e. \tag{5.96}$$

Indeed, by applying $T_{\tilde{\mu}}$ to both sides of Eq. (5.95), it follows that

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

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

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

By taking the limit as $k \to \infty$, and by using the VI convergence property $T_{\tilde{\mu}}^k J_\mu \to J_{\tilde{\mu}}$, we obtain Eq. (5.96).

By applying $T_{\tilde{\mu}}$ to Eq. (5.96), it follows that

$$T_{\tilde{\mu}} J_{\tilde{\mu}} \le T_{\tilde{\mu}} J_\mu + \frac{\alpha(\epsilon + 2\alpha\delta)}{1-\alpha} e,$$

so using the fact $J_{\tilde{\mu}} = T_{\tilde{\mu}} J_{\tilde{\mu}}$, we have

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

Subtracting J^* from both sides, we obtain

$$J_{\tilde{\mu}} - J^* \le T_{\tilde{\mu}} J_\mu - J^* + \frac{\alpha(\epsilon + 2\alpha\delta)}{1-\alpha} e. \tag{5.97}$$

Also from the contraction property of T,

$$TJ_\mu - J^* = TJ_\mu - TJ^* \le \alpha \| J_\mu - J^* \| e$$

which, in conjunction with Eq. (5.94), yields

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

Combining this relation with Eq. (5.97), we obtain

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

which is equivalent to the desired relation (5.93), since $J_{\tilde{\mu}} \ge J^*$.   **Q.E.D.**

Proof of Prop. 5.1.4: Applying Lemma 5.9.1, we have

$$\|J_{\mu^{k+1}} - J^*\| \le \alpha \|J_{\mu^k} - J^*\| + \frac{\epsilon + 2\alpha\delta}{1-\alpha},$$

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.

Proposition 5.1.5: (Performance Bound for Approximate PI when Policies Converge) Let $\tilde{\mu}$ be a policy generated by the approximate PI algorithm under conditions (5.11), (5.12), and (5.13). Then we have

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

Proof: Let $\bar{J}$ be the cost vector obtained by approximate policy evaluation of $\tilde{\mu}$. Then in view of Eqs. (5.11), (5.12), we have

$$\|\bar{J} - J_{\tilde{\mu}}\| \le \delta, \qquad \|T_{\tilde{\mu}}\bar{J} - T\bar{J}\| \le \epsilon.$$

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

$$\begin{aligned} \|TJ_{\tilde{\mu}} - J_{\tilde{\mu}}\| &\le \|TJ_{\tilde{\mu}} - T\bar{J}\| + \|T\bar{J} - T_{\tilde{\mu}}\bar{J}\| + \|T_{\tilde{\mu}}\bar{J} - J_{\tilde{\mu}}\| \\ &= \|TJ_{\tilde{\mu}} - T\bar{J}\| + \|T\bar{J} - T_{\tilde{\mu}}\bar{J}\| + \|T_{\tilde{\mu}}\bar{J} - T_{\tilde{\mu}}J_{\tilde{\mu}}\| \\ &\le \alpha\|J_{\tilde{\mu}} - \bar{J}\| + \epsilon + \alpha\|\bar{J} - J_{\tilde{\mu}}\| \\ &\le \epsilon + 2\alpha\delta. \end{aligned} \tag{5.98}$$

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

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

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

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

Combining this relation with Eq. (5.98), we obtain the desired performance bound. **Q.E.D.**