Introduction to Markov Decision Processes

Martin L. Puterman and Timothy C. Y. Chan

December 29, 2023

Chapter 1

Simulation-Based Methods for Tabular Models

During the 1950s, I decided, as did many others, that many practical problems were beyond analytic solution and that simulation techniques were required. At RAND, I participated in the building of large logistics simulation models; at General Electric, I helped build models of manufacturing plants.

Harry Markowitz - American economist and Nobel Prize recipient - (1927-2023)

This chapter develops simulation-based methods to evaluate value functions, state-action value functions and find "good" policies in models in which the set of states and set of actions are sufficiently small so that the value functions, state-action value functions and (deterministic) policies can be represented in tabular form. Chapter **??** generalizes these methods to allow value function approximations.

We use the expression *methods* instead of algorithms because the approaches proposed do not have precise stopping criteria. Moreover, we use the expression *good* rather than optimal because unlike the algorithms in Chapters **??**, **??** and **??**, the methods here are not guaranteed to find optimal policies. We use the expression *simulation* to refer to any method to generate subsequent states and rewards from current states and actions as in Figure 1.1. It can be based either on Markov decision process transition probabilities and rewards, a system simulator or experimentation in a real-world process.

The key concept in this chapter is that:

> Simulation replaces computation of expectations in Bellman equations.

The methods herein apply to two environments which are frequently referred to as *model-based* and *model-free*.

Model-based: The analyst has developed an Markov decision process model of the decision process as in the previous chapters of this book. Therefore S, $\{A_s :$

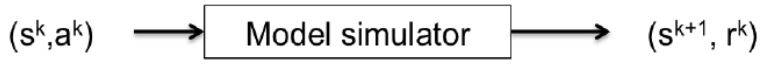


Figure 1.1: Symbolic representation of environmental model using a simulator or real-world process. It's purpose is to transform state-action pair (s^k, a^k) to state-reward pair (s^{k+1}, r^k) **(redraw with (s,a) transformed to (r,s') and call it a data generator.)**

$s \in S\}$,, $r(s, a, j)$ and $p(j|s, a)$ are known and can be used to generate data. Examples include inventory models, queuing control and revenue management.

Model-free: At a minimum the analyst knows the actions available at each decision epoch. The set of states may or not be known. State transitions and rewards are generated by a simulator or a real-world process. Examples include gaming, autonomous vehicle control and medical treatment. In episodic models there might not be an explicit reward structure so that the analyst must design one so that objectives are achieved as efficiently as possible.

Note that is some applications such as advanced appointment scheduling where an Markov decision process model can be derived (Section **??**), it might be easier to adopt a model-free approach even though a model is available[1]

When analyzing such methods, we distinguish two data generation[2] environments:

Simulation in which states may be generated from *either* a pre-specified distribution or the underlying transition structure.

Process-based in which states can only be generated by the possibly unknown underlying transition structure.

The reason for this distinction is that in the simulation environment, the analyst may restart the system at any state while in the process environment, states follow system dynamics. To make this precise, consider a robot control problem. When simulating it, the robot can restart anywhere at any time while in the process-based environment the robot can only transition to adjacent locations.

This chapter describes and illustrates policy evaluation and optimization methods for episodic and infinite horizon discounted models and also provide a brief discussion of average reward models. Its intent is not to be comprehensive, but to provide you with a high-level framework for delving into the voluminous and rapidly changing reinforcement learning literature.

[1]The idea here is that instead of deriving an analytical model for the MDP, it might be easier to simulate demand and implement a policy.

[2]The data we will use for analysis consists of a sequence of states, actions and rewards.

1.1 Preliminaries

Before describing computational methods, we discuss data generation, underlying trade-offs when choosing a method and provide some guidelines on how to choose method parameters.

1.1.1 Where does the data come from?

We begin by describing several possible ways that data may be generated. Starting with some $s^1 \in S$, the methods described below generate a sequence $(s^1, a^1, r^1, s^2, a^2, r^2, \ldots)$ to estimate value functions or state-action value functions and find good policies. In the model-based environment $r^n = r(s^n, a^n, s^{n+1})$ so it can be computed from sequence of state-action pairs while in the the model-free environment it must be observed directly.

We distinguish four data generating mechanisms:

Long sequence based: Choose $s^1 \in S$, policy π and generate $(s^1, a^1, r^1, s^2, a^2, r^2, \ldots)$.

State-based: At each iteration, generate $s \in S$, choose action $d(s)$[3], and generate r and s'.

Action-based: At each iteration "observe" $s \in S$, choose or generate $a \in A_s$, then generate r and s'.

State-action pair based: At each iteration, generate $s \in S$ and $a \in A_s$, and then generate r and s'.

We comment on the use, benefits and limitations of these data generation approaches.

1. Data may be generated either *sequentially* or *restarted after each iteration.* Sequential data may be realized in real-time as the output of a real or simulated process. Simulation must be used to generate random restart data because in a real-world system, states evolve according to a possibly unknown physical model so that random restarts would not be able to be physically realized.

2. Long sequence and state-based approaches can be used for policy evaluation.

3. Action-based methods are particularly useful in optimization because they allow exploration in the action space.

4. State based methods, in which states are generated either randomly or deterministically, enable generation of rewards in states that are infrequently (or not) visited under some policies.

[3]When d is randomized, this action can be generated from $w_d(a|s)$.

5. Either *online* or *offline* methods can be used to estimate value function or state-action value functions from generated data. Online methods update value function or state-action value functions in real-time as each (s, a, r) observation becomes available. Offline (*batch*) methods use a complete realization of the data for computation. In practice one may combine these two approaches using an off-line implementation early to obtain good starting values for subsequently applying an on-line method.

1.1.2 Trade-offs

We now describe some terminology and trade-offs that you will encounter when using the methods described in subsequent sections. We expand on these issues throughout the chapter.

1. **Synchronous vs asynchronous:** A *synchronous* method simultaneously updates value functions in **all** states or state-action value functions in **all** state-action pairs in a similar way to value and policy iteration. An *asynchronous* method updates values or q-functions one state or state-action pair at a time in a similar way to Gauss-Seidel iteration. Most of the iterative methods in this chapter are asynchronous.

2. **Exploration vs exploitation:** In light of Einstein's statement:

   "Insanity is doing the same thing over and over again and expecting different results"

   it is advantageous to try new things occasionally. Thus in order to find good policies, a method needs to deviate from greedy action selection in order to investigate the impact of other actions on system performance. We refer to greedy action selection as *exploitation* and on-greedy action selection as *exploration*. Exploration methods include ϵ-greedy and softmax sampling which are describe below. Exploitation may be attractive when the decision maker wishes to accrue reward during the learning phase.

3. **On-policy vs off-policy:** A method is said to be *on-policy* if the quantity used to update the state-action value function coincides with the action taken. A method is *off-policy* if they differ[4]. This issue is most relevant when the reward received during the learning phase of an implementation is of concern to the investigator.

4. **Variance vs bias:** In statistical estimation one often seeks a minimum-variance unbiased estimator. When evaluating a method, the analyst may be willing to accept some bias in exchange for reduced variance. The *root mean squared*

[4]This distinction will be most significant in Q-learning.

error for distinguished states or averaged over states captures this trade-off. (See Appendix A for a detailed discussion of this issue.)

1.2 Evaluating the value of a policy in an episodic model

This section provides the building blocks to investigate different simulation or data driven approaches for finding an optimal policy.

For transparency we start with episodic models. Recall that in such models, the system accrues rewards until it reaches a reward-free absorbing state at some, possibly random, time. This time may be determined either by the evolution of the system or externally. For example in a robot guidance model, an episode may terminate when either the robot achieves its goal[5]. As noted in Section **??**, a discounted model may be viewed as an episodic model in which the episode terminates at a geometric stopping time independent of the evolution of the system. Note also that finite horizon models may be viewed as episodic models with a fixed stopping time.

In episodic models, we seek to estimate the expected total reward of stationary policy $\pi = d^\infty$ given by

$$v^\pi(s) = E^{\pi,T}\left[\sum_{j=1}^{T-1} r(X_t, d(X_t), X_{t-1}) + g(X_T)\middle| X_1 = s\right] \tag{1.1}$$

where the random variable T represents the random time at which the process terminates. Since it was shown in Chapter **??** that deterministic stationary policies are optimal for such models, we restrict our attention to such policies in this section.

1.2.1 Monte Carlo algorithm

The most obvious approach for estimating $v^\pi(s^1)$ is to sum up the rewards for each episode and average over episodes. This leads to the following two *Monte Carlo* algorithms. In each, we assume that an episode terminates when the system reaches a state in the set S_Δ.

Starting state Monte Carlo

We formally state this simple version of a Monte Carlo algorithm for illustrative purposes. Its first-visit and every-visit variants are preferable.

[5]We could stop an episode when the robot fails, but our computations in various applications suggest it is more expedient to terminate the episode only when the goal is achieved

Algorithm 1.1. Model-based Monte Carlo policy evaluation in an episodic model; starting state update only

1. **Initialize:**
   (a) Choose $d \in D^{MD}$ and the number of replicates K.
   (b) For each $s \in S - S_\Delta$ create an empty list $Values(s)$.
   (c) $n \leftarrow 1$.
2. **Replicate episodes:** While $n < K$;
   (a) Generate $s^0 \in S - S_\Delta$, $s \leftarrow s^0$ and $v(s^0) \leftarrow 0$.
   (b) **Generate episode:** While $s \notin S_\Delta$:
      i. Generate $s' \in S$ from $p(\cdot|s, d(s))$.
      ii. Update value:

$$v(s^0) \leftarrow r(s, d(s), s') + v(s^0) \tag{1.2}$$

      iii. $s \leftarrow s'$.
   (c) **Terminate episode:**
      i. Append $v(s^0) + g(s)$ to $Values(s^0)$
      ii. $n \leftarrow n + 1$
3. **Compute estimates:** For each $s \in S - S_\Delta$, set $\hat{v}^{d^\infty}(s) = mean(Values(s))$.

Some comments about the algorithm follow:

1. The above algorithm estimates $v^{d^\infty}(s)$ by setting it equal to the average of observed total rewards for all episodes that start in state s.
2. Since the total rewards from each episode are independent and identically distributed, $\hat{v}^{d^\infty}(s)$ converges almost surely to it mean, $v^{d^\infty}(s)$, by the *strong law of large numbers* and is asymptotically normal by the *central limit theorem*.
3. By storing $v(s)$ in the list $Values(s)$, we can also investigate distributional properties of the estimates including the standard deviation and percentiles. Moreover we can can compute standard errors (standard deviation/square root number of obs) and confidence intervals for the estimates and use the width to guide choice of K, the number of episodes to evaluate.
4. Choice of s^0 in 2(a) should be such that we obtain values for all starting states; most important states can be chosen more often so that the value estimates are more accurate there.

5. Note that 2(c) is only executed when $s \in S^{\Delta}$.

6. The algorithm as stated above *updates values only for the starting state* s^0 of each episode. Obviously data would be used more efficiently if it also estimated $v(s)$ for *all* states visited during an episode. If a state repeats, the estimate for that state may be based on the total return accumulated from the first visit to that state (or from every visit to that state). Thus at each pass through 2(b), $v(s)$ should be updated for *all* states previously visited. We provide the first-visit variant of the algorithm below and compare the efficiency of the two algorithms in Example 1.1.

7. Algorithm 1.1 is an off-line algorithm since it requires all data to estimate $v^{d^\infty}(s)$ in step 3.

First-visit Monte Carlo

The following algorithm uses data more efficiently.

Algorithm 1.2. Monte Carlo policy evaluation in an episodic model; first visit updates

1. **Initialize:**
   (a) Choose $d \in D^{MD}$ and number of episodes K
   (b) Create empty list $Values(s)$ for all $s \in S - S_\Delta$
   (c) $n \leftarrow 1$
2. **Replicate episodes:** While $n < K$,
   (a) Create empty list $Visited$.
   (b) $v(s) \leftarrow 0$ for all $s \in S - S_\Delta$.
   (c) Select $s \in S$.
   (d) **Generate episode:** While $s \notin S_\Delta$:
      i. Append s to $Visited$.
      ii. Generate $s' \in S$ from $p(\cdot|s, d(s))$.
      iii. For all $s'' \in Visited$

$$v(s'') \leftarrow r(s, d(s), s') + v(s''). \tag{1.3}$$

      iv. $s \leftarrow s'$.
   (e) **Terminate episode:**
      i. For all $s \in Visited$ append $v(s) + g(s)$ to $Values(s)$.
      ii. $n \leftarrow n + 1$.
3. **Compute estimates:** For each $s \in S - S_\Delta$, set $\hat{v}^{d^\infty}(s) = mean(Values(s))$.

Some comments about this algorithm follow:

1. The list $Visited$ keeps track of the states visited during the episode and updates the value in each.
2. An alternative to this algorithm would be the every-visit variant which in each replicate, starts a new total every time a state is visited. That is, if the sequence of states was $(s_1, s_2, s_5, s_2, \ldots)$ such an algorithm would generate two estimates for s_2. We leave it to the reader to formally state and apply such an algorithm. **(Add exercise)**

An Example

We now illustrate these two algorithms by evaluating a policy in the grid-world model introduced in Section **??**. For simplicity, we assume an episode starts after the robot has filled the coffee cup in cell 13 and ends when either it reaches the office (cell 1), or falls down the stairs (cell 7). In this example, $S = \{1, \ldots, 15\}$ and $S_\Delta = \{1\}$[6]. We repeat the cell configuration diagram in Figure 1.2 below.

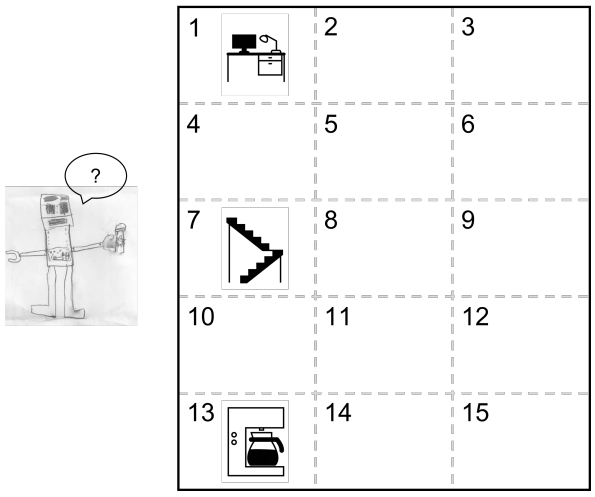


Figure 1.2: Schematic layout for grid-world navigation example

Example 1.1. Monte Carlo estimation in an episodic model:

We consider the grid-word model represented in Figure 1.2 with $p = 0.8$, a penalty of -200 for falling down the stairs, a reward of 50 for successfully delivering the coffee and a per unit move cost -1. This example analyzes the stationary policy d^∞ which aims to move to the lowest numbered neighboring cell except when in state 10 when the robot tries to move toward cell 11. It moves to the intended cell with probability p and ends up in each other adjacent cell with equal probability[a]. For example in cell 14, the robot will end up in cell 11 with probability p and cells 13 and 15 with probability $(1-p)/2$. We assume a cost of 1 for each move, a reward of 50 for completing the task and a penalty of 200 if the robot falls down the stairs.

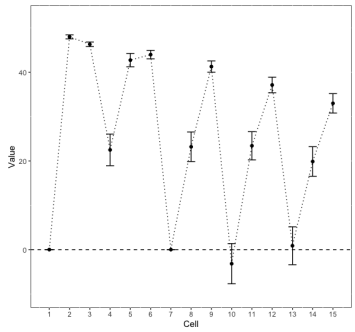
First we compare the two algorithms for $p = .8$. In Algorithm 1.1 we choose the initial state randomly from all non-terminal states and set the number of replicates equal to $50,000$ to obtain standard errors approximately equal to 1. We also ran the first-visit variant of the algorithm (Algorithm 1.2) but randomized the starting state to be either cell 13, 14 or 15.

[6] As noted above, we can alternatively set $S_\Delta = \{1, 7\}$.

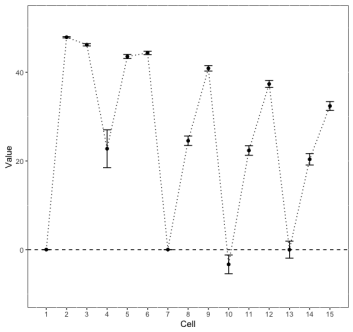
Figure 1.3c show the estimates of the values starting in each state using the two Monte Carlo Algorithms. Observe that estimates based on first visit updates had narrower simultaneous 95% confidence intervals than those based on the starting value only variant. For example the standard errors when starting in cell 13 were 1.13 for starting state only MC and 0.45 for first visit MC. When starting in cell 2, the respective standard errors were 0.38 and 0.13 respectively. Thus as expected, the first-visit Monte Carlo algorithm provides more accurate estimates with the same number of replications.

Figure 1.3c provides a histogram of values in cell 13 when using the first visit Monte-Carlo algorithm. Observe that the distribution is extremely bi-modal corresponding to the possibilities of falling down the stairs or alternatively successfully delivering the coffee. Consequently, the mean, which is approximately zero, doesn't well represent this distribution and is perhaps a poor choice of a value of a policy. Note that the 5th percentile is -204, the 25th percentile is 38, the median is 42 and the 75th and 95th percentiles equal 44. Thus one might wish to consider the use of different summaries of the distribution (such as the median or 75th percentile) when analyzing such models.

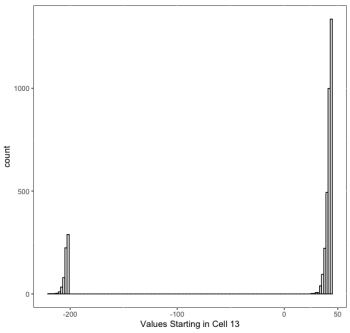
[a] We assume that that the robot cannot move diagonally.



(a) Value function estimates using Algorithm 1.1



(b) Value function estimates using Algorithm 1.2.



(c) Histogram of value estimates for cell 13 using Algorithm 1.2. Note that the mean of the estimates is 0.81 and the true value for this quantity is 0.68.

Figure 1.3: Graphical results for Example 1.1 based on 50,000 replicates and $p = 0.8$. Error bars in Figures 1.3a and 1.3b are based on simultaneous (Bonferroni) 95% confidence intervals.

1.2.2 Temporal differencing in an episodic model

Temporal differencing represents one of the main innovations in reinforcement learning. Here we describe it for models in tabular form. We generalize it to models which use function approximation in Chapter **??**.

We derive the temporal difference recursion by applying the Robbins-Monro recursion (B.1) from Appendix B on stochastic approximation. This approach is often referred to as *temporal-differencing* or *TD(0)* because it updates a value by a proportion of the difference between a value function and an estimate referred to as a *temporal difference*[7].

Temporal differencing is based on the following recursive form for $v^{d^\infty}(s)$, for $d \in D^{MR}$ which previously appeared as (**??**),

$$v(s) = E^d\left[r(X_1, Y_1, X_2) + v(X_2)\middle| X_1 = s\right] \tag{1.4}$$

and the following formula which appears in Appendix B to this chapter as (B.4)

$$x^{n+1} = x^n + \tau_n(z^n - x^n)$$

for estimating the mean of a random variable Z based on a sequence of observations $\{z^n; n = 1, 2, \ldots\}$ from its distribution. In this expression, the sequence of parameters $\{\tau_n : n = 1, 2, \ldots\}$ is referred to as the *learning rate*.

Taking $Z = r(X_1, Y_1, X_2) + v(X_2)$ and sampling from the distribution of (Y_1, X_2) conditional on $X_1 = s$, (B.4) becomes

$$v_{n+1}(s) = v_n(s) + \tau_n[r(s, a, s') + v_n(s') - v_n(s)]. \tag{1.5}$$

Note that if d was deterministic, $Y_1 = d(X_1)$ so that $a = d(s)$ in the above expression. Note that some authors refer to the quantity

$$r(s, a, s') + v_n(s') - v_n(s)$$

as a *Bellman error*. We will see in Chapter **??** that temporal differencing can be viewed as an application of gradient descent to a model in tabular form.

Online TD(0)

Recall that S_Δ denotes the set of reward free absorbing states.

[7]Such a scheme is referred to as *bootstrapping* because it uses current value-function estimates to update the future value-function estimates.

Algorithm 1.3. Temporal differencing for policy evaluation in an episodic model

1. **Initialization**
   (a) Specify a decision rule $d \in D^{MD}$.
   (b) Specify the number of episodes K.
   (c) Specify the learning rate sequence $\{\tau_n : n = 1, 2, \dots\}$.
   (d) $v(s) \leftarrow 0$ for all $s \in S$.
   (e) $count(s) \leftarrow 0$ for all $s \in S$.
   (f) $n \leftarrow 1$.
2. **Updating** While $n < K$.
3. Generate initial state $s \in S$.
4. While $s \notin S_\Delta$:
   (a) $count(s) \leftarrow count(s) + 1$
   (b) Generate s' from $p(\cdot|s, d(s))$.
   (c) Set
   $$v(s) \leftarrow v(s) + \tau_{count(s)}[\, r(s, d(s), s') + v(s') - v(s)\,] \qquad (1.6)$$
   (d) $s \leftarrow s'$.
5. $n \leftarrow n + 1$.

Some comments about Algorithm 1.3 follow:

1. Temporal differencing is an online algorithm with asynchronous updates.
2. In contrast to Monte Carlo estimation in which value function updates occur at the end of an episode, this algorithm updates the value function estimate within each episode after each transition. At the end of an episode, a new initial state is generated in step 3.
3. The initial state may be generated by any, possibly random, mechanism that is appropriate for a problem. For example, it is often advantageous to start far from absorbing states so that many states may be visited (and the values corresponding to those states updated) during each episode. However, a random start over all states may provide accurate estimates for states visited infrequently under decision rule d. This will be particularly relevant when seeking an optimal policy.

4. Monte Carlo methods might be used to get good starting values for $v(s)$ before applying temporal differencing.

5. The algorithm, uses the number of visits to a state as the index for the learning rate parameter τ_n.

6. Choosing τ_n is somewhat of an art. We recommend a systematic approach combining RMSE with graphical summaries for guidance as described in the chapter appendix A

7. After applying (1.6) in state s, it is possible to update values in states previously visited during an episode. This is the idea underlying TD(γ) algorithms which we describe below.

8. Since temporal differencing is an application of the Robbins-Munro recursion, when conditions (B.3) hold in state s, the temporal differences $v_n(s)$ converge to $v^{d^\infty}(s)$ in probability. We can achieve this by using $count(s)$ as the index of the learning rate and assuring that $count(s) \to \infty$ for each $s \in S$ which requires that state s be visited infinitely often.

Example 1.2. Application of temporal differencing in the gridworld example
We apply Algorithm 1.3 to the gridworld model previously analyzed using both Monte Carlo algorithms in Example 1.1. We consider the same policy as specified in that example and choose the starting state for each episode from cells $4, 13, 14$ and 15 with equal probability. Note we add cell 4 to the above list because it is visited infrequently under the specified policy. We set the number of replicates to 20000 (roughly 5000 for each starting state).
Figure 1.4 displays estimates of $v(13)$ for three choices of $\tau_n = an^{-b}$ that were shown to give good fits under the RMSE criterion over a range of random number seeds and parameter choices. Observe that the estimate with $b = 0.5$ has high variance, that with $b = 0.95$ had low variance but was biased and that with $b = 0.75$ had small bias and variance low variance. Note also that robot successfully delivered the coffee in 90% of the episodes and fell down the stairs in 10% of the episodes.

Hybrid TD-Monte Carlo

Some authors propose a hybrid form of Monte Carlo[8] that performs temporal differencing updates only *after* the completion of an episode. This is equivalent to applying

[8]Sutton and Barto [2018] refer to this as MC(α) or α-Monte Carlo.

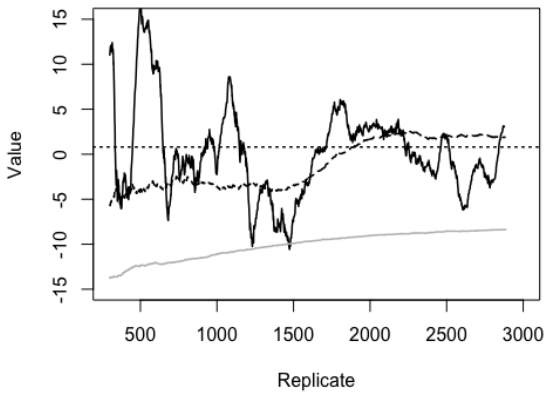


Figure 1.4: Estimates of $v^{d^\infty}(13)$ for Example 1.2 using Algorithm 1.3 with $\tau_n = .8n^{-0.5}$ (solid black), $\tau_n = .8n^{-0.75}$ (dashed black) and $\tau_n = .8n^{-0.95}$ (grey). The horizontal dotted line represents the true value of this quantity (0.638).

the Robbins-Munro recursion for estimating a mean to

$$v^\pi(s) = E^{\pi,T}\left[\sum_{j=1}^{T-1} r(X_t, d(X_t), X_{t-1}) + g(X_T)\middle| X_1 = s\right]$$

instead of to the Bellman recursion (1.4).

Hybrid TD-Monte Carlo may be viewed as a generalization of the online recursion for estimating a mean (B.5)

$$x^{n+1} = x^n + \frac{1}{n}(z^n - x^n).$$

This method replaces the learning rate of $\frac{1}{n}$ by the more general sequences $\{\tau_n; n = 1, 2, \ldots\}$. We leave it as an exercise to write out and apply the this algorithm.

1.2.3 TD(γ)

The temporal difference method described above, TD(0), updates values one state at a time, however ignores the effect of the current temporal difference on previously visited state values. TD(γ) update previous values in a structured way. Note the literature refers to these as TD(λ) methods but we have used λ to denote the discount factor so we will use γ to denote the algorithmic parameter.

We show how to use it for policy evaluation in episodic models. To provide some preliminary insight, suppose we start the robot in cell 4 in the gridworld example

(Figure 1.2) and it visits cells 5, 6, 3, and 2 before terminating in cell 1. Then after the transition from cell 6 to cell 3, the TD(0) update would be

$$v(6) = v(6) + \tau[-1 + v(3) - v(6)].$$

But we could just as well used the two-step update

$$v(6) = v(6) + \tau[-2 + v(2) - v(6)]$$

or even the three-step update

$$v(6) = v(6) + \tau[-3 + 50 + v(1) - v(6)].$$

As a compromise we could pool these in some way, and that is what TD(γ) does.

Motivation

We restate the argument in the previous paragraph in generality. The temporal difference algorithm to evaluate a deterministic stationary policy d^∞ performs an update in the current state s according to

$$v(s) \leftarrow v(s) + \tau\left[r(s, d(s), s') + v(s') - v(s)\right]$$

where s' is the subsequent state sampled from $p(\cdot|s, d(s))$. As noted above, this update is the result of applying stochastic approximation to estimate

$$v(s) = E\left[r(X_n, d(X_n), X_{n+1}) + v(X_{n+1})|X_n = s\right] \tag{1.7}$$

Upon applying a similar recursion for $v(X_{n+1})$ and substituting it back into (1.7) we obtain

$$v(s) = E\left[r(X_n, d(X_n), X_{n+1}) + r(X_{n+1}, d(X_{n+1}), X_{n+2}) + v(X_{n+2})|X_n = s\right]. \tag{1.8}$$

This suggest an alternative two-step update for $v(s)$ given by

$$v(s) \leftarrow v(s) + \tau\left[r(s, d(s), s') + r(s', d(s'), s'') + v(s'') - v(s)\right]$$

where s'' is a realization of the state two-decision epochs after being in state s. Clearly this can be generalized to updates of any number of steps into the future. In particular the m-step update can be based on the recursion

$$v(s) = E\left[\sum_{k=0}^{m-1} r(X_{n+k}, d(X_{n+k}), X_{n+k+1}) + v(X_{n+m})|X_n = s\right]. \tag{1.9}$$

Letting

$$R_n^m := \sum_{k=0}^{m-1}\left[r(X_{n+k}, d(X_{n+k}), X_{n+k+1}) + v(X_{n+m})\right]$$

for $m \geq 1$, (1.9) can be rewritten as $v(s) = E\left[R_n^m | X_n = s\right]$.

TD(γ) builds on this logic in two ways. It combines updates of every length using a weighting factor of $(1-\gamma)\gamma^n$, $0 \leq \gamma \leq 1$ for the $(n+1)$-th recursion, that is, it expresses $v(s)$ as

$$v(s) = E\left[\sum_{m=0}^{\infty}(1-\gamma)\gamma^{m-1}R_n^m \,\middle|\, X_n = s\right] \tag{1.10}$$

and estimates the quantity in brackets in (1.10) using an on-line procedure that updates estimates of $v(s)$ in previously visited of states based on an *eligibility trace* which keeps track of the appropriate weightings. We leave it to the reader to consult the literature to establish the validity of this online approach.

Algorithm and examples

We provide, discuss and apply an algorithm that implements TD(γ).

Algorithm 1.4. TD(γ) algorithm for evaluating a fixed deterministic stationary policy d^∞ in an episodic model

1. **Initialize variables and parameters:**
   (a) Set $v(s) \leftarrow 0$, $e(s) \leftarrow 0$ and $count(s) \leftarrow 0$ for all $s \in S$.
   (b) Choose $0 \leq \gamma \leq 1$ and a sequence $\{\tau_n\, n = 1, 2, \ldots\}$
   (c) Specify number of episodes K, $n \leftarrow 0$ and choose an $s \in S$.
2. **Iterate over replicates:** While $n < K$:
3. Select initial state $s \in S$.
4. **Within episode update:** While $s \notin S_\Delta$
   (a) $e(s) \leftarrow 1 + e(s)$.
   (b) Generate $s' \in S$ by sampling from $p(\cdot|s, d(s))$.
   (c) Evaluate the temporal difference

$$\delta \leftarrow r(s, d(s), s') + v(s') - v(s). \tag{1.11}$$

   (d) For all $s \in S$,

$$v(s) \leftarrow v(s) + \tau_{count(s)}\delta e(s) \tag{1.12}$$

   and

$$e(s) \leftarrow \gamma e(s).$$

   (e) $s \leftarrow s'$ and $n \leftarrow n + 1$.

We now comment on the algorithm:

1. As stated, this is an online algorithm.

2. This algorithm introduces an additional tuning parameter γ corresponding to the weighting of representations of $v(s)$ in (1.10). Small values of γ result in updating values for only a few previously visited states while values near 1 update this value for more iterates.

3. Some authors refer to the quantity $e(s)$ as an *eligibility trace*. It is a clever way to accomplish two things:

   (a) It keeps track of previously visited states so that their values can be updated at all subsequent iterations and

   (b) it exponentially damps the effect of the current temporal difference δ on the value in each state based on when it was last visited. The parameter γ is called the *trace decay* parameter.

4. *When using the update for $e(s)$ in 4(a), the resulting $e(s)$ is referred to as an *accumulating trace*. Note that it can take values greater than 1 if a state is visited frequently, especially when γ is close to 1. An alternative updating rule is to replace 4(a) by $e(s) = 1$ which is referred to as a *replacement trace*. This results in updating a value in a state only since its last visit.

5. The above algorithm has two special cases, TD(0) which is equivalent to the (one-step) temporal difference algorithm introduced earlier and TD(1) which is equivalent to a version of Monte Carlo updating.

We now illustrate the application of this algorithm to policy evaluation in the grid-world model.

Example 1.3. Policy evaluation using TD(γ) in an episodic model

We now apply TD(γ) to the grid-world model using the previous specified parameter values. We explore the impact of γ on estimates by choosing $\gamma \in \{0, 0.2, 0.4, 0.6, 0.8, 0.95\}$. We set $\tau_n = 0.8n^{-0.75}$ and apply Algorithm 1.4 using both the accumulating eligibility trace $e(s) \leftarrow e(s)+1$ and the replacement trace $e(s) \leftarrow 1$ in step 4(a). We set $K = 10,000$, used a common random number seed for all estimates, and compare choice of γ and trace type based on the RMSE of estimates of v^{d^∞} after deleting the first $1,000$ values.

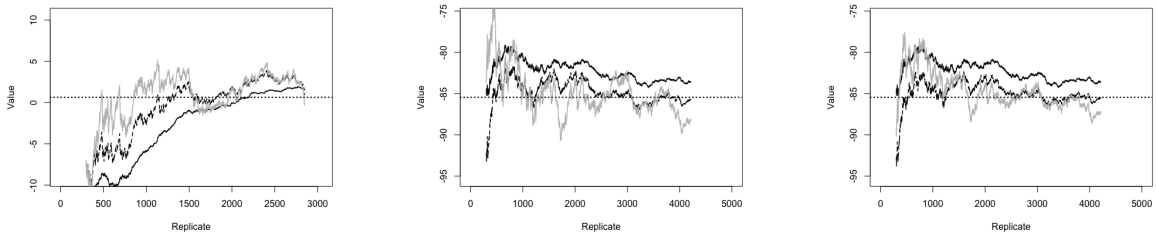
We found that when $p = 0.8$, $\gamma \in [0, 0.2]$ produced estimates with the smallest RMSE and the type of trace used didn't matter when γ was small. Figure 1.5a shows that in this case TD(0) produced the best estimates.

Outcomes were more varied when $p = 0.5$ and the best estimate was usually around $\gamma = 0.5$. The replacement trace estimates, shown in Figure 1.5c, had smaller RMSE (and were less variable) than the accumulating trace estimates in

Figure 1.5b. Moreover the TD(0) estimates converged quickly but to the wrong value. Note that when γ was close to 1, the accumulating trace values $e(s)$ became quite large. Note that when $p = 0.5$, there were more replicates that produced estimates in cell 13 because robot trajectories were varied than when $p = 0.8$.

As an aside, note that when p=0.5, the probability of successfully delivering the coffee is 0.61, that is is under this policy approximately 29% of the time, the robot fell down the stairs.

We conclude from this example that, TD(γ) provides an attractive alternative to TD(0) for policy evaluation. However care must go into choice of tuning parameters and the number of replicates. In a problem below, we ask you to conduct an in depth study of tuning parameter choice. We return to this issue in further examples in this chapter.



(a) Estimates when $p = 0.8$ using replacement trace.

(b) Estimates when $p = 0.5$ using accumulating trace.

(c) Estimates when $p = 0.5$ using replacement trace.

Figure 1.5: Estimates of $v^{d^\infty}(13)$ for $p = 0.5$ and $p = 0.8$ based on a single online implementation of Algorithm 1.4. The solid black line corresponds to $\gamma = 0$, the dashed line to $\gamma = 0.5$ and the grey line to $\gamma = 0.8$. The dotted line gives the true value of $v^{d^\infty}(13)$ for the specified value of p.

1.3 Estimating the value of a policy in an infinite horizon discounted model

In an infinite horizon discounted model, the approach used depends on how we represent the value function. TD(γ) is based on applying stochastic approximation for estimating a mean given by (B.4) to the representation for $v_\lambda^{d^\infty}(s)$ in the Bellman recursion

$$v_\lambda^{d^\infty}(s) = E^d\left[r(X_1, Y_1, X_2) + \lambda v_\lambda^{d^\infty}(X_2)\,\middle|\, X_1 = s\right]. \tag{1.13}$$

On the other hand, Monte Carlo or hybrid TD-Monte Carlo apply to approximating $v_\lambda^{d^\infty}(s)$ by a *truncated discounted sum*

$$v_\lambda^{d^\infty}(s) \approx E^{d^\infty}\left[\sum_{n=1}^{M} \lambda^{n-1} r(X_n, Y_n, X_{n+1}) \,\middle|\, X_1 = s\right] \tag{1.14}$$

for M finite and sufficiently large or by representing it as *undiscounted random sum*

$$v_\lambda^{d^\infty}(s) = E^{d^\infty, T}\left[\sum_{n=1}^{T} r(X_n, Y_n, X_{n+1}) \,\middle|\, X_1 = s\right]. \tag{1.15}$$

We emphasize that equations (1.13) and 1.15 are exact while 1.14 is an approximation[9].

In the random sum case, T is an independent geometric stopping time with parameter $1-\lambda$ and the expectations are with respect to the probability distributions of T and $(X_1, Y_1, X_2, \dots)$ generated by the specified policy. In (1.15), the discount factor appears implicitly in the expectation of T with respect to a geometric distribution with parameter $1-\lambda$.

Table 1.1 summarizes which algorithms are apply in each case. Note that Monte Carlo methods cannot be used directly in discounted models because the infinite sum

$$E^{d^\infty}\left[\sum_{n=1}^{\infty} \lambda^{n-1} r(X_n, Y_n, X_{n+1}) \,\middle|\, X_1 = s\right]$$

cannot be evaluated using simulation. When truncating rewards, all methods apply because the discount factor ensures convergence. However when using geometric stopping times, TD(γ) may diverge because of the absence of a discount factor. We ask you to explore these differences further in an exercise.

Case/ Method	Total Discounted reward	Truncated reward	Geometric stopping times
Monte Carlo	No	Yes	Yes
Hybrid TD-Monte Carlo	No	Yes	Yes
TD(γ)	Yes	Yes	No

Table 1.1: Summary of appropriate algorithms for policy evaluation in discounted models

1.3.1 TD(γ) in a discounted model

For ease of reference we restate the TD(γ) algorithm in a form suitable for use when data is generated by the long-sequence approach. A modification for a state-based approach is discussed below.

[9]Some authors refer to the quantities inside the expectations in (1.13) -(1.15) at *targets* for temporal difference updates.

Algorithm 1.5. TD(γ) algorithm for evaluating a fixed deterministic stationary policy d^∞ in a discounted model

1. **Initialize variables and parameters:**
   (a) Set $v(s) \leftarrow 0$, $e(s) \leftarrow 0$ and $count(s) \leftarrow 0$ for all $s \in S$.
   (b) Choose $0 \leq \gamma \leq 1$ and a sequence $\{\tau_n \, n = 1, 2, \ldots\}$
   (c) Specify number of iterates N, $n \leftarrow 0$ and choose an $s \in S$.
2. **Update values:** While $n < N$.
   (a) $e(s) \leftarrow 1 + e(s)$.
   (b) Generate $s' \in S$ by sampling from $p(\cdot|s, d(s))$.
   (c) Evaluate temporal difference
   $$\delta \leftarrow r(s, d(s), s') + \lambda v(s') - v(s). \tag{1.16}$$
   (d) For all $s \in S$,
   $$v(s) \leftarrow v(s) + \tau_{count(s)} \delta e(s) \tag{1.17}$$
   and
   $$e(s) \leftarrow \lambda\gamma e(s). \tag{1.18}$$
   (e) $s \leftarrow s'$ and $n \leftarrow n + 1$.

We now comment on some features of the algorithm:

1. In contrast to the undiscounted episodic model, the discount factor λ appears explicitly in (1.16) and (1.18). Note that in the latter equation the eligibility trace is damped by the factor $\lambda\gamma$ so that the impact on previous terms is less than in the episodic model. Thus distinguishing between the accumulating trace and the replacement trace should have less impact.

2. The Markov chain generated in 2(b) may not visit some states very often so that as a result of using a tabular representation, value function estimates for those states may be highly variable. When simulation is being used to generate values, one can instead generate new values of s from any distribution on S. In this case, the first part of step 2(e) could be replaced by "Choose $s \in S$ from a uniform distribution on S."

   We sometimes refer to this approach as *random restart*. Of course, if data is generated by a process rather than by a simulation, such exploration variant may not be possible.

3. As in the case of episodic models, step 2(a) generates an accumulating trace.

Alternatively, one can replace this step by $e(s) \leftarrow 1$ which corresponds to using a replacement trace. We investigate these two options in the example below.

4. The algorithm may be sensitive to initial values. Choosing a good initial value such as

$$v_0(s) = (1-\lambda)^{-1} E\{r(X_0, d(X_0), X_1) | X_0 = s\} = (1-\lambda)^{-1} r(s) \tag{1.19}$$

for each $s \in S$ may enhance convergence. This value of $v_0(s)$ corresponds to starting the system in state s and receiving an identical reward of $r(s)$ at each decision epoch over the infinite horizon.

5. The algorithm will converge if all states are visited infinitely often and the usual conditions on $\{\tau_n\}$ apply.

6. Again TD(0) is a special case when $\gamma = 0$. In this case, the discount rate enters only through temporal difference (1.16) since (1.18) sets $e(s) = 0$.

An example

We illustrate this algorithm in the context of the two-state model in Section **??**. Of course it would not be prudent to use simulation in such a simple model but we do so to illustrate and comment further on the estimation of value functions in discounted models with simulation.

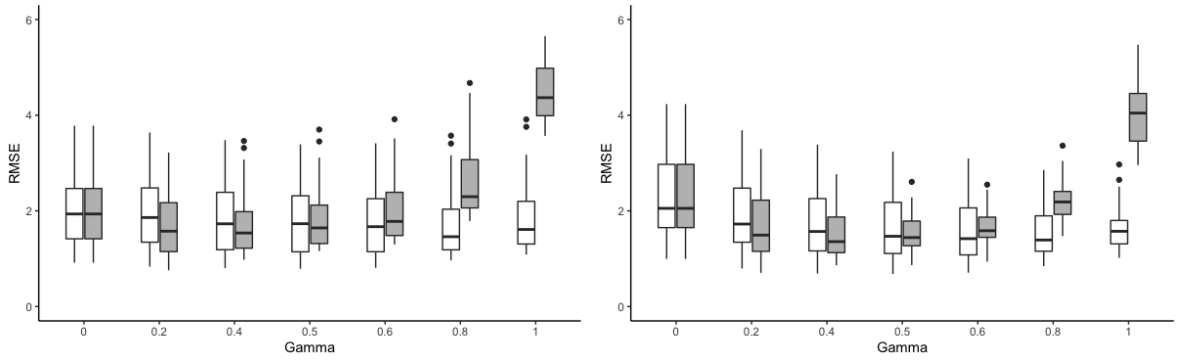
Example 1.4. TD(γ) in a discounted model.

We illustrate this algorithm in the context of the two-state model from Section **??**. We estimate the expected infinite horizon discounted reward, $v_\lambda^{d^\infty}(s)$, for the stationary deterministic policy d^∞ where $d(s_1) = a_{1,2}$ and $d(s_2) = a_{2,2}$. Setting $\lambda = 0.9$, solving $(I - \lambda P_d)v = r_d$ establishes that $v_\lambda^\pi(s_1) = 30.147$ and $v_\lambda^\pi(s_2) = 27.941$. These values will be used to enable comparison of estimates on the basis of RMSE.

We compare the long sequence (online) approach to the state-based random restart approach using TD(γ) as in Algorithm 1.5. After some preliminary experimentation We choose $\tau_n = 0.5n^{-0.6}$ and sequence length of $N = 10000$. We vary γ and the trace type and replicate the algorithm 100 times using common random seeds for all instances in each replicate. We assess the quality of estimates on the basis of the RMSE averaged over states and replicates. We delete the the first 1000 values when computing the RMSE for each replicate.

Figures 1.6a and 1.6b show box-plots of the output by γ and trace type. The most relevant findings are that:

1. The best estimates occurred using the accumulating trace and $\gamma = 0.4$ with the average RMSE equal 1.51 in the "random restart" case and 1.73 in the "long trajectory" case.



(a) Long trajectories (b) Random restart

Figure 1.6: Example 1.4 outputshowing RMSE for online TD(γ) by γ and trace type (accumulating - grey; replacement - white) based on 100 replicates of TD(γ) using Algorithm 1.5.

2. In both case, the accumulating trace estimates were less variable than the replacement trace estimates for $0.4 \leq \gamma \leq 0.8$.

3. TD(0) estimates were both less variable and had lower average RMSE when using long trajectories as opposed to state- based random restart.

4. Formal analysis using ANOVA indicated highly significant γ, trace and interaction effects in both cases.

1.3.2 Truncated reward sequences

Table 1.1 indicates that we can use Monte Carlo, Hybrid TD-Monte Carlo and TD(γ) together with truncation to estimate values in discounted models. We found that TD(γ) gave almost identical results to long-sequence approach in Algorithm 1.5 because except for restarts after reaching the truncation limit M, the two approaches are equivalent. We now describe a hybrid TD-Monte Carlo algorithm for truncated sequences.

Given a realization of states and actions, $(s^1, a^1, s^2, \dots)$, at issue is when to truncate $\sum_{n=1}^{\infty} \lambda^{n-1} r(s^n, a^n, s^{n+1})$. This can be done *a priori* by choosing N so that

$$\lambda^{n-1} \max_{s \in S, a \in A_s, s' \in S} |r(s, a, s')| \leq \epsilon \tag{1.20}$$

for some pre-specified tolerance ϵ. Alternatively one can choose M adaptively by stopping when the changes in values of successive estimates is small.

Algorithm 1.6. Estimating the value of a deterministic stationary policy using truncated discounted rewards.

1. **Initialize:**
   (a) Specify a decision rule $d \in D^{MD}$.
   (b) Specify $\{\tau_n : n = 1, 2, \ldots\}$ and the number of replications K.
   (c) Choose the truncation limit M and set $k = 1$.
2. **Replicate:** While $k < K$:
   (a) $u(s) \leftarrow 0$ for all $s \in S$ and $m \leftarrow 1$.
   (b) Randomly choose $\bar{s} \in S$ and set $s \leftarrow \bar{s}$.
   (c) **Generate an episode:** While $m < M$:
      i. Obtain s' by sampling from $p(\cdot|s, d(s))$
      ii. $u(s) \leftarrow u(s) + \lambda^{m-1} r(s, d(s), s')$
      iii. $s \leftarrow s'$
      iv. $m \leftarrow m + 1$.
   (d) **Update:**
   $$v(\bar{s}) \leftarrow v(\bar{s}) + \tau_k[u(\bar{s}) - v(\bar{s})] \tag{1.21}$$
   (e) $k \leftarrow k + 1$.

Some comments about the algorithm follow:

1. The above algorithm chooses a state $\bar{s}$ and then accumulates the rewards from realizations of a Markov chain starting in that state. It then updates the estimate of $v_\lambda^{d^\infty}(s)$ for that state using the TD(0) recursion (1.21). It is not clear whether a $TD(\gamma)$ variant would be beneficial.

2. In contrast to TD(γ), the index k of τ_k refers to the number of previous starts from state $\bar{s}$, not the number of visits to that state. Choosing $\tau_k = \frac{1}{k}$ provides Monte Carlo estimates however other choices are possible.

3. The above algorithm is based on starting-state Monte Carlo so (1.21) only updates the value function estimate for state $\bar{s}$. A more efficient approach would be to use either first-visit or every visit Monte Carlo. These modifications require adjusting the index of τ to correspond to the number of times a state has been evaluated.

4. In step 2(b) states can be chosen deterministically so as to repeat each state a sufficient number of times so that $v(s)$ accurately estimates $v^{d^\infty}(s)$.

1.3.3 Geometric stopping times

Using representation (1.14) for $v_\lambda^{d^\infty}(s)$ is equivalent to terminating each episode at the first "success" in a sequence of Bernoulli trials with success probability $1-\lambda$. Algorithmically, this corresponds to evaluating an *undiscounted* sum of rewards by either:

1. An *episodic model* in which the episode ends after the first success in series of Bernoulli trials with success probability $(1-\lambda)$, or

2. An *randomly truncated model* in which the time of truncation for each replicate is sampled from a geometric distribution with parameter $(1-\lambda)$ and we accumulate the non-discounted sum of rewards[10]

We leave it to the reader to formally state hybrid TD-Monte Carlo algorithms that use these two approaches. We use random truncation in the example below.

1.3.4 An example

Example 1.5. Comparison of truncation and random geometric stopping time estimates.

We analyze the two-state model of Section **??** using hybrid TD-Monte Carlo algorithms based on the representations (1.14) and (1.15). We set $\lambda = 0.9$ and evaluate the deterministic stationary policy d^∞ where as before $d(s_1) = a_{1,1}$ and $d(s_2) = a_{2,2}$. Preliminary calculations suggest that $\tau_n = 0.75n^{-1}$ and $\tau_n = n^{-1}$ provide good estimates. We compare fixed truncation at $M = 30, 50, 70$ and random geometric truncation. We generate 100 instances in which each instance consists of 5000 replicates starting in each state. We evaluate the quality of the estimates based on the average RMSE over the two states and 100 instances in which the first 1000 replicates have been deleted.

Table 1.2 provides estimates of the average RMSE and its standard deviation and shows that in each case, $\tau_n = n^{-1}$ yields a smaller and less variable average RMSE than that for $\tau_n = 0.75n^{-1}$. This is to be expected since the average over instances, which is equivalent to using $\tau_n = n^{-1}$, is the minimum variance unbiased estimator of the mean. Moreover, the table shows that:

1. The quality of truncation estimates improves with increasing M with those for $M = 50$ and $M = 70$ quite similar.

2. The geometric stopping time estimate has smaller mean but is more varaible) than $M = 30$ estimate. This is surprising since the the mean

[10]Note that some languages (such as R) represent the geometric distribution as the number of failures up to the first success. If this is the case, it is necessary to add 1 to the. simulated value to agree with our definition of a geometric distribution.

number of episodes evaluated by the geometric stopping time approach is only$(1-\lambda)^{-1} = 10$.

Figure 1.7 provides box plots of the average RMSE for the four methods. Such box plots allow us to comment on the variability of the estimates. Careful inspection reveals that

1. With the exception of the $M = 30$ case, the estimates are skewed to the right.
2. For $M = 50$ and $M = 70$, the estimates base on $\tau_n = n^{-1}$ are less variable than those for $\tau_n = 0.75n^{-1}$.
3. For all truncation estimates, the *median* of the average RMSE is smaller for $\tau_n = n^{-1}$.

Based on the above discussion of results we conclude that using the Monte Carlo estimate with $50 \leq M \leq 70$ gave the most precise estimates. Nothing was to be gained by using a hybrid TD-Monte Carlo approach.

Further, referring to Example 1.4 we see that any of the Monte Carlo estimates produced smaller average RMSE than the TD(γ) estimates.

Method	$\tau_n = 0.75n^{-1}$	$\tau_n = n^{-1}$
Truncation: M=30	1.330 (0.360)	1.310 (0.328)
Truncation: M=50	0.626 (0.288)	0.591 (0.239)
Truncation: M=70	0.579 (0.265)	0.562 (0.226)
Geometric Stopping Times	1.009 (0.440)	0.956 (0.336)

Table 1.2: Average RMSE over 100 instances for four estimators in the two state model. Standard deviations of average RMSE over instances appear in parentheses.

1.3.5 *Estimation in average reward models

Most research in simulation-based dynamic programming has focused on infinite horizon episodic and discounted models. Here we take a brief digression to discuss the use simulation in infinite horizon average reward models. In this section, we discuss using TD(0) to compute the average reward or *gain*, g of the stationary policy d^∞.

If our only goal was to find g through simulation, we could either:

1. Estimate the stationary distribution $\pi(s)$ by setting $\hat{\pi}(s)$ equal to the proportion of visits to each state under d^∞ and setting

$$g = \sum_{j \in S} \hat{\pi}(s)\tilde{r}(j) \tag{1.22}$$

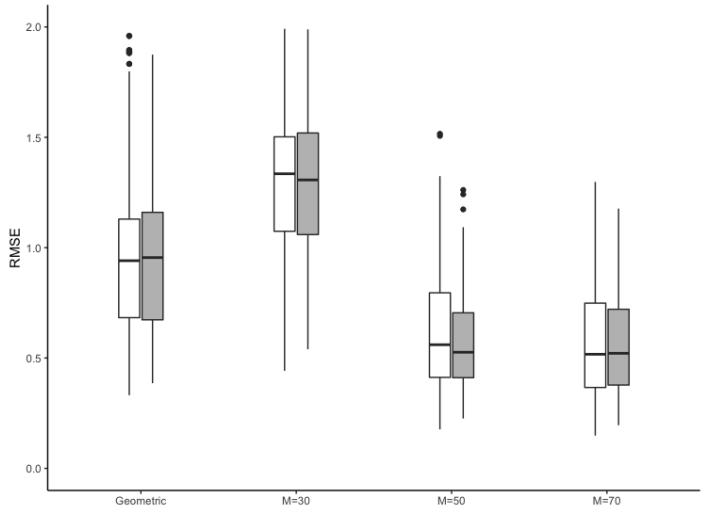


Figure 1.7: Box plots of geometric stopping time and truncation estimates of RMSE in two state model with $\tau_n = .75n^{-1}$(white) and $\tau_n = n^{-1}$(grey).

where $\tilde{r}(s)$ denote the expected one-period reward starting in state s and following deterministic decision rule $d(s)$, that is $\tilde{r}(s) = \sum_{j \in S} p(j|s, d(s))r(s, d(s), j)$.

2. Update g recursively by setting $g^0 = 0$ and iterating

$$g^{n+1} = g^n + \frac{1}{n}(r^n - g^n) \tag{1.23}$$

where r^n is the observed reward at iterate n. In this case $g^{n+1} = \frac{1}{n}\sum_{k=1}^{n} r^k$.

It is important to stress here that when we turn to finding an optimal policy, knowing g is not sufficient. We also require the bias $h(s)$. Chapter **??** shows that the gain and bias satisfy the system of equations

$$h(s) = \sum_{j \in S} p(j|s, d(s)) \left[r(s, d(s), j) + h(j)\right] - g. \tag{1.24}$$

Since there are $|S|$ equations in $|S| + 1$ unknowns, we need an extra condition to guarantee a unique solution. From the prospective of optimization we require only the relative values of the bias terms so we can set $h(s_0) = 0$ for a designated $s_0 \in S$. In that case, after rearranging terms, the s_0th equation becomes

$$g = \sum_{j \in S} p(j|s_0, d(s_0)) \left[r(s_0, d(s_0), j) + h(j)\right] \tag{1.25}$$

Writing (1.24) in expectation form as

$$h(s) = E^{d^\infty}\left[r(X_1, Y_1, X_2) + h(X_2) - g \,\middle|\, X_1 = s\right] \tag{1.26}$$

and (1.25) as

$$g = E^{d^\infty}\left[r(X_1, Y_1, X_2) + h(X_2) \,\middle|\, X_1 = s_0\right] \tag{1.27}$$

motivates the following TD(0) algorithm for infinite horizon models with average reward criterion. This online approach assumes data is generated in long trajectory form.

Algorithm 1.7. Estimating the average reward and relative values of a deterministic stationary policy using TD(0).

1. **Initialize**
   (a) Choose a decision rule $d \in D^{MD}$ and a distinguished state s_0.
   (b) Specify sequences $\{\tau_n\}$ and $\{\beta_n\}$ $n = 1, 2, \ldots$
   (c) Initialize $g = 0$ and $h(s) = 0$ and for all $s \in S$.
   (d) Initialize $count(s) = 0$ for all $s \in S$.
   (e) Choose $s \in S$.
   (f) Specify the number of replicates K.
   (g) $n \leftarrow 1$
2. **Repeat** While $n < K$:
   (a) Generate $s' \in S$.
   (b) $count(s) \leftarrow count(s) + 1$
   (c) If $s \neq s_0$,

$$h(s) \leftarrow h(s) + \tau_{count(s)}\left[r(s, d(s), s') + h(s') - h(s)\right] \tag{1.28}$$

   (d)

$$g' \leftarrow g + \beta_n(r(s') - g) \tag{1.29}$$

   (e) $n \leftarrow n + 1$
   (f) $s \leftarrow s'$

Some comments follow:

1. In this algorithm, recursion (1.28) is based on applying stochastic approximation to the expectation in (1.26).
2. It retains $h(s_0) = 0$ so the resulting estimates of h are relative values.

3. The algorithm requires specifying smoothing constants $\{\tau_n\}$ and $\{\beta_n\}$ which may differ. Note that $\beta_n = n^{-1}$ is equivalent to recursive estimate of the mean of g based on the observed rewards as in (1.23).

4. Note that the updates are asynchronous; g is updated at every iteration and $h(s)$ is only updated when visiting state $s = s_0$

5. A variant of the algorithm, based on using the representation for g in (1.27), specifies a state s_0, imposes the condition $h(s_0) = 0$ and when $s = s_0$, updates g at each iteration using the recursion

$$g' \leftarrow g + \tau_{count(s_0)} \left[(r(s_0, d(s_0), s') + h(s') \right] \tag{1.30}$$

   instead of with (1.29) and updates $g \leftarrow g'$ prior to the subsequent pass through the algorithm.

6. If $d \in D^{MR}$, in each state s, the action would need to be sampled from $w_d(s, a)$ prior to (1.28).

7. Tsitsiklis and Roy [1999] show that an TD(γ) variant of this algorithm converges (in probability) under the assumption that the Markov chain corresponding to d^∞ is irreducible and aperiodic and the learning rates are chosen appropriately. Note their result also applies to models with function approximation such as in Chapter **??**.

We compare this algorithm and its variant based on (1.30) in the context of the two-state model of Section **??**.

Example 1.6. TD(0) in an average reward model.

We again analyze the policy d^∞ where $d(s_1) = a_{1,1}$ and $d(s_2) = a_{2,2}$. Under this policy the Markov chain is aperiodic and irreducible so that we find the gain and relative value by solving equation (**??**) in Chapter **??** under the condition that $h(s_1) = 0$. Doing so shows that $g = 2.667$ and $h(s_2) = -1.667$.

We compare results using Algorithm 1.7 to that which updates g using (1.30). We set $\tau_n = 0.5n^{-0.6}$, $\beta_n = 0.7n^{-0.9}$ and $K = 15000$. These values were found through considerable experimentation as many learning rates resulted in highly inaccurate estimates of $h(s_2)$. Table 1.3 compares the two estimation approaches over 50 instances by computing the mean and standard deviations of RMSE's of the estimates of g and $h(s_2)$.

This table shows that Algorithm 1.7 provides more accurate estimates of g and less accurate estimates of $h(s_2)$ than the variant. Since the estimate of g has no direct impact on policy choice when using the Bellman equation to find an optimal policy, the variant which determines $h(s_2)$ more accurately may be preferable when using simulation to find optimal policies.

Figure 1.8 (based on a single instance) shows that estimates from the algo-

rithm and its variant are highly variable and not reliable. Note also that the estimate of the gain using the variant was positively biased in all replicates. We conclude that **estimation in the average reward setting is challenging because of the absence of a discount factor to damp out variability.** We ask the reader to consider a TD(γ) variant which might provide better estimates.

Method	RMSE g	RMSE $h(s_2)$
Algorithm as stated using (1.29)	0.142 (0.044)	1.214 (0.271)
Algorithm variant using (1.30)	0.271 (0.050)	1.159 (0.214)

Table 1.3: Mean (standard deviation) over 50 replicates of the RMSE of g and $h(s_2)$ in Example 1.6.

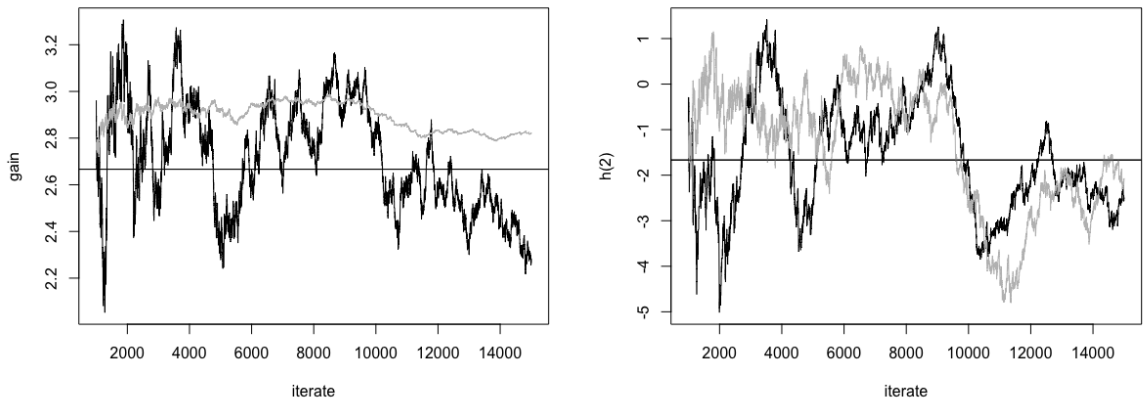


Figure 1.8: Estimates, **from a single instance**, of g (left) and $h(s_2)$ (right) obtained using Algorithm 1.7 and its variant based on (1.30). The horizontal line denotes the true value of the estimated quantities. The grey line corresponds to using the algorithm directly and the black line to its variant .

1.4 Finding optimal policies

We now describe some simulation-based methods for finding good policies and illustrate them in the episodic case with the grid-world model (Example 1.1) and in the discounted case with (Examples **??** and **??**). The methods either optimize step-by-step in a similar fashion to value iteration or combine one of the evaluation approaches above with some form of policy improvement step analogous to policy iteration. We

emphasize that the methods in this chapter assume a tabular table representation for the model.

Our approach generalizes the evaluation methods of the previous sections by focusing on the state-action value function $q(s, a)$. This necessitates estimating $\sum_{s \in S} |A_s|$ quantities.

One can view simulation-based optimization from two perspectives:

1. That of an agent interacting with its environment and learning a good policy by choosing a series of actions which result in rewards when desirable outcomes have occurred and penalties when failing.
2. That of an external controller (or analyst) using simulation as a tool to learn good policies to be used subsequently.

Of course, these distinctions are not mutually exclusive, but at a high level, they distinguish the computer science approach and the operations research approach.

1.4.1 Exploration vs Exploitation

The methods in this section require exploration in the action space to find good policies. Without such exploration, the methods may not produce reliable and accurate estimates of $q(s, a)$ for some state-action pairs. Consequently policies based on decision rules selected **greedily**, that is choosing $d(s) \in \arg\max_{a' \in A_s} q(s, a')$, may not be good. We describe two approaches to exploration; ϵ-greedy action selection and softmax sampling.

ϵ-greedy action sampling

An **ϵ-greedy** action selection approach combines greedy and **random** action selection. Under random action selection in state s, each action in A_s is chosen with probability $1/|A_s|$.

Given ϵ satisfying $0 < \epsilon < 1$, the ϵ-greedy approach chooses the greedy action with probability $(1 - \epsilon)$ and a random action with probability $\epsilon/|A_s|$. This corresponds to implementing a randomized decision rule d with

$$w_d(s, a) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|A - s|} & a = \arg\max_{a'} Q(s, a') \\ \frac{\epsilon}{|A - s|} & a \neq \arg\max_{a'} Q(s, a'). \end{cases} \tag{1.31}$$

When using this approach ϵ may be a decreasing function of n so that initially the algorithm evaluates many actions while later on, it mostly chooses the presumed best action. As result, estimates of values corresponding to optimal policies are quite accurate.

Softmax sampling

Another approach for exploration is to choose action a in state s with probability specified by a *softmax* distribution of the form

$$P(Y_n = a|X_n = s) = \frac{e^{\beta q(s,a)}}{\sum_{a' \in A_s} e^{\beta q(s,a')}} \tag{1.32}$$

where β is a constant to be selected. Sometimes the reciprocal of β is referred to as the *cooling rate*. When little information is available, the softmax picks all actions with nearly equal probability while as the $q(s,a)$'s become more distinct, the softmax is more likely to pick actions with lareger $q(s,a)$ values.

1.5 Q-learning and variants

The algorithms in this section generalize TD(0) and TD(γ) by focusing on state-action value functions. We distinguish between two algorithms; Q-learning, which uses the maximum of the state-action value function in the attained state and "Sarsa", which uses the attained state-action value function in updates.

Given an estimate of the value function v most optimization algorithms in Chapters 5-7 are based on evaluating expressions of the form

$$\max_{a \in A_s} \left\{ \sum_{j \in S} p(j|s,a) \left[r(s,a,j) + \lambda v(j) \right] \right\}$$

where $0 < \lambda \leq 1$.

The algorithms in this section focus on iteratively updating the expression inside the {}'s based on a sequence of states, actions and rewards obtained from a simulation or the execution of a learning activity. Instead of basing the updates on v, the state-action value function $q(s,a)$ becomes the quantity of interest and it is updated according to

$$q(s,a) = q(s,a) + \tau_n \left[r(s,a,s') + \lambda \max_{a' \in A_s} q(s',a') - q(s,a) \right]. \tag{1.33}$$

This update corresponds to applying stochastic approximation to the equation

$$q(s,a) = E\left[r(X_1, Y_1, X_2) + \lambda \max_{a' \in A_{X_2}} q(X_2, a') \middle| X_1 = s, Y_1 = a \right]. \tag{1.34}$$

Note that in episodic and average reward models, $\lambda = 1$. In the later case (1.33) also contains a $-g$ term to correspond to the appropriate recursion for the gain and bias.

The q-function we seek to estimate in episodic and discounted models is given by

$$q^*(s,a) = \sum_{j \in S} p(j|s,a) \left[r(s,a,j) + \lambda v^*(j) \right]. \tag{1.35}$$

where $v^*(s)$ is the corresponding value function. In average reward models the q-function becomes

$$q^*(s,a) = \sum_{j \in S} p(j|s,a) \left[r(s,a,j) - g^* + h^*(j) \right]. \tag{1.36}$$

where g^* denotes the gain of an optimal policy and h^* the bias normalized in some way. Thus, in experiments when we know v^* (or g^* and h^*), we can use the known value of $q^*(s,a)$ to assess the algorithmic performance.

Below we provide a separate yet parallel discussion of Q-learning algorithms for episodic, discounted and average reward models that distinguish subtle differences between them.

1.5.1 Episodic models

We now provide a Q-learning algorithm for an episodic model. Recall that in such a model, an episode terminates when reaching a state in S_Δ.

Algorithm 1.8. Q-Learning in an episodic model

1. **Initialize:**
   (a) $q(s,a) \leftarrow 0$ for each $a \in A_s$ and all $s \in S$.
   (b) $counts(s,a) \leftarrow 0$ for each $a \in A_s$ and all $s \in S$.
   (c) Specify the learning rate $\{\tau_n \; n = 1, 2, \ldots\}$.
   (d) Select an $s \in S$ and $a \in A_s$.
   (e) Specify the number of episodes K.
   (f) Specify a sequence $\{\epsilon_n \; n = 1, 2, \ldots\}$ for ϵ-greedy action selection.
   (g) $n \leftarrow 1$
2. While $n < K$
   (a) **Evaluate an episode:** Do until $s \in S_\Delta$.
      i. $counts(s,a) \leftarrow counts(s,a) + 1$.
      ii. Sample $s' \in S$ from $p(\cdot|s,a)$.
      iii. Choose $a \in A_{s'}$ ϵ_n-greedily.
      iv. **Update** $q(s,a)$**:**

$$q(s,a) \leftarrow q(s,a) + \tau_{counts(s,a)} \left[r(s,a,s') + \max_{a' \in A_{s'}} q(s',a') - q(s,a) \right] \tag{1.37}$$

      v. $s \leftarrow s'$
   (b) $n \leftarrow n + 1$

Some comments follow:

1. Under the assumption that each state-action pair is visited infinitely often and the learning rate satisfies the stochastic approximation requirements, Q-learning convergence in probability to the optimal state-action value function and identifies an optimal policy. **(I think this is independent of ϵ_n)**.
2. In step 2(a)iii., softmax action choice can replace ϵ_n-greedy action choice..
3. If the algorithm is being implemented through simulation, steps 2(a)ii -iii can be replaced by "Sample (s,a) randomly. It may converge faster if only step 2(a)ii is replaced by "Sample s randomly".
4. Note that in the update (1.37) $r(s,a,s')$ depends only on the *current* action choice while $\max_{a' \in A_{s'}} q(s',a')$ accounts for *future* action choice.

5. The above algorithm generalizes TD(0). We leave it to the reader to explore generalizations of TD(γ) .

6. Some authors classify Q-learning as an *off-policy* algorithm, because in the presence of exploration, the value of $q(s', a')$ in the update of $q(s, a)$ need not correspond to the policy the algorithm is following. An *on-policy* variant, referred to as *Sarsa*[11], follows.

Algorithm 1.9. Sarsa in an episodic model.

1. Initialize as in Q-learning algorithm above.
2. While $n < K$:
   (a) **Evaluate an episode:** Do until $s \in S_\Delta$.
      i. $counts(s, a) \leftarrow counts(s, a) + 1$.
      ii. Sample $s' \in S$ from $p(\cdot|s, a)$.
      iii. Choose $a \in A_{s'}$ ϵ_n-greedily.
      iv. Update $q(s, a)$ by
$$q(s, a) \leftarrow q(s, a) + \tau_{counts(s,a)} [r(s, a, s') + q(s', a') - q(s, a)] \tag{1.38}$$
      v. $s \leftarrow s'$, $a \leftarrow a'$.
   (b) $n \leftarrow n + 1$

(Make precise Sarsa may converge to sub-optimal policy)

We illustrate the difference between these two algorithms with the following simple deterministic example.

Example 1.7. Distinguishing Q-learning and Sarsa.

Consider the snippet of a deterministic Markov decision process depicted in Figure 1.9. In state s there are two actions, a and b and in state u there are two actions $\bar{a}$ and $\bar{b}$. Choosing action a in state s yields a reward of 1 and a transition to state u. All other rewards are immaterial for our discussion.

Suppose that $q(s, a) = 0$, $q(u, \bar{a}) = 5$, $q(u, \bar{b}) = 3$ and exploration resulted in choosing action $\bar{b}$ in state u. Then Q-learning would update $q(s, a)$ according to

$$q(s, a) = q(s, a) + \tau[r(s, a, u) + \max_{a'=\bar{a},\bar{b}} q(u, a') - q(s, a)] = 0 + \tau[1 + 5 - 0] = 6\tau$$

[11]The acronym *Sarsa* represents "state-action-reward-state-action" corresponding to the sequence of objects $(s, a, r(s, a, s'), s', a')$ involved in the update in (1.38) below. Note that when the reward depends on the subsequent state, as in the above sequence, the acronym *Sasra* would be more appropriate.

while Sarsa would update $q(s, a)$ according to

$$q(s, a) = q(s, a) + \tau[r(s, a, u) + q(u, \bar{b}) - q(s, a)] = 0 + \tau[1 + 3 - 0] = 3\tau.$$

Thus while both approaches use the same action in state u, the update of $q(s, a)$ using Q-learning differs from that of Sarsa. Moreover the process would follow the action that Sarsa used in its update while the q-learning update would be based on a different action. Of course if exploration chose action $\bar{a}$ the update in the two algorithms would agree.

Hence after repeating this process several times, the q values generated by these two methods would differ and so might action choice.

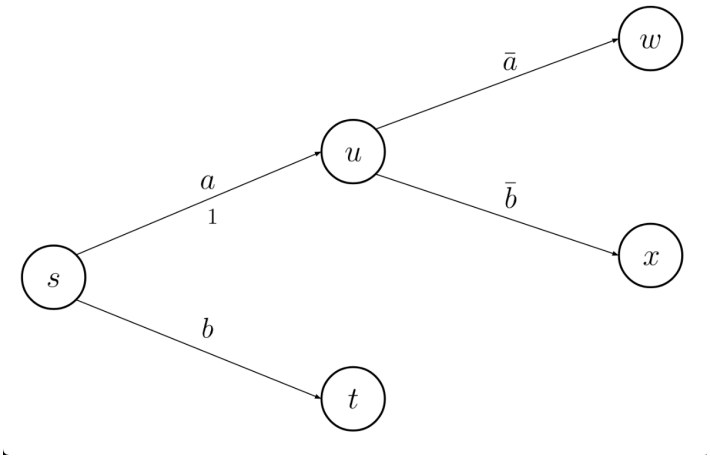


Figure 1.9: Graphical representation of model in Example 1.7.

The on-policy vs off-policy dilemma

Why would one care about this distinction? Analysis of a deterministic version of the grid-world example in Figure 1.2 provides further insight.

Example 1.8. Distinguishing on-policy and off-policy updates

Suppose the coffee-delivering robot occupies cell 11, plans to move upward (action 1) to cell 8 and the ϵ-greedy action choice for cell 8 results in a move to the left (a=2) so that the robot falls down the stairs. If this happens, the Sarsa and Q-learning updates differ.

The Sarsa update is qiven by

$$q(11, 1) \leftarrow q(11, 1) + \tau[-1 + q(8, 2) - q(11, 1)]$$

while the Q-learning update is

$$q(11,1) \leftarrow q(11,1) + \tau[-1 + \max_{a' \in A_8} q(8,a') - q(11,1)]$$

Suppose we initialize $q(s,a)$ at its optimal value so that $q(11,1) = 46$, $q(8,2) = -201$ and $q(8,1) = 47$[a]. Then the Sarsa update would yield

$$q(11,1) \leftarrow 46 + \tau[-1 - 201 - 46] = 46 - 248\tau$$

while the Q-learning update would be

$$q(11,1) \leftarrow 46 + \tau[-1 + 47 - 46] = 46.$$

Thus in future replicates, under Sarsa, the robot would not choose the optimal action (a=1) in cell 11 and attempt to move right while under Q-learning the robot would move to cell 8 risking the likelihood of falling down the stairs.

This example shows that as a consequence of ϵ-greedy action choice, the Sarsa-following robot will take more conservative actions.

[a]The robot receives a reward of 50 when delivering the coffee and incurs a cost of -1 for each transition.

What this means practically is that when using exploration in the action space with either ϵ-greedy or softmax sampling, *use Q-learning if your objective is to find an optimal policy for future use, while Sarsa might be preferable when outcomes during the learning phase are of concern.* In Example 1.8 when training the robot, frequently crashing it down the stairs may prove costly so that Sarsa might be preferable. Alternatively if we put a barrier across the top of the stairs and still incurred the penalty of -200, we would prefer Q-learning because it will find an optimal policy for future use.

Computational examples

We compare the performance of these two variants in the context of two versions of the grid-world example.

Example 1.9. Model-free grid-world In this version of the grid-world model, the robot can select any of the actions "up", "right", and "left" in any cell. Action choice produces a deterministic transition in the intended direction when possible. Otherwise the robot remains in the current cell. Each time the robot chooses an action in a state, the robot incurs a cost of -1, even if it cannot move.

We apply Q-learning with $q(s,a)$ initialized at 0 and initial state set at cell 13. An episode ends when the robot reaches cell 1. Randomness results from softmax

action choice (with $\beta = 1/25$). Thus an episode may consist of many instances when the robot falls down the stairs. In such instances (or upon successfully reaching the office), the process restarts in cell 13.

We found that with $\tau_n = .8n^{-.75}$ that Q-learning converged in 500 episodes to

$$q(s,a) = \begin{bmatrix} 0.00 & 0.00 & 0.00 \\ 48.00 & 47.00 & 49.00 \\ 47.00 & 47.00 & 48.00 \\ 49.00 & 46.98 & 47.99 \\ 48.000 & 46.000 & 49.00 \\ 47.00 & 46.00 & 46.00 \\ 3.50 & 3.97 & 7.59 \\ 47.00 & 45.00 & -112.77 \\ 46.00 & 45.00 & 46.00 \\ -114.59 & 45.00 & 44.00 \\ 46.00 & 44.00 & 44.00 \\ 45.00 & 44.00 & 45.00 \\ \mathbf{44.00b} & \mathbf{44.00} & \mathbf{43.00b} \\ 45.00b & 35.97 & 43.00 \\ 41.02 & 26.27 & 31.21 \end{bmatrix}$$

where the rows correspond to the cell number and columns to actions "up","right" and "left" respectively. The entries in bold correspond to starting in cell 13 and choosing the designated action. Choosing action "up" (or "right") generates a value of 44 which equals the total reward starting in 13 and following the shortest path.

Note that in several rows, two actions yielded the same value implying that there are several optimal policies. One such policy is $13 \to 10 \to 11 \to 8 \to 5 \to 2 \to 1$. More over knowledge of $q(s,a)$ allows determination of the shortest path to cell 1 from each cell. We leave it to the reader to identify other optimal policies and justify the values in row 7 among others.

We also compared the results to those obtained using Sarsa with softmax sampling. We found that with K=500, it identified the same optimal policy but the $q(s,a)$ values were considerably smaller than those obtained using Q-learning. Table 1.4 shows the average total reward per episode and the standard deviation of total reward per epsiode for Q-learning and Sarsa. Observe that as discussed above, the accumulated reward under Sarsa was greater than that obtained by Q-learning.

With this knowledge, the agent can abandon softmax action selection, encode the optimal policy and obtain the maximum reward.

We find the results in Example 1.9 quite remarkable. **With no intervention, and no knowledge of the world, the agent identified the optimal policy and**

method	Mean reward	Standard deviation
Q-Learning	31.36	11.10
Sarsa	35.22	13.92

Table 1.4: Mean and standard of reward per episode average over 500 replicates in Example 1.9.

accurately estimated the optimal state-action value function. comment also about fact transition behavior wasn't modelled!

The following example combines random movement with exploration in the model based variant.

Example 1.10. Comparison of Q-learning and Sarsa in grid-world
Guided by our analysis in Example 1.2 we choose $\tau_n = 0.8n^{-.0.75}$ and evaluated 15000 episodes randomly starting in cells 4,13,14 and 15. We used the same model parameters as in Example 1.1.

We found that results, which we assessed on the basis of the estimated q-function and how close the resulting policy was to optimum over 40 replicates, highly sensitive to choice of the sequence $\{\epsilon_n; n = 1, 2, \dots\}$. Ultimately we tried several options including:

$$\epsilon_n = 0.2 + 0.15I_{[0,5000]}(n) + 0.1I_{[0,1000]}(n) \tag{1.39}$$
$$\epsilon'_n = 0.05 + 0.1I_{[0,10000]}(n) + 0.1I_{[0,5000]}(n) + 0.1I_{[0,10000](n)} \tag{1.40}$$

We found representing ϵ_n by a step function in n gave better results than using a continuous function of n.

Table 1.5 displays some results. It shows that:

1. For both greedy parameter choices, estimates of $q(s, a)$ were considerably more accurate using Q-learning than Sarsa.

2. The greedy policy based on the estimated $q(s, a)$ was non-optimal more frequently under Q-learning than Sarsa.

3. The estimates of $q(s, a)$ were more accurate using the sequence ϵ'_n than using ϵ_n.

4. When greedy policies were non-optimal they differed from the optimal policy in only one state (either "cell 4" or "cell 8"). These states were visited infrequently under the optimal policy which steered the robot towards the right wall before reaching the stairs.

Additional experiments using softmax action choice and varied starting states produced similar results. Moreover, the performance of Q-learning improved as p was increased to 1.

Method	Greedy parameter	RMSE	Policy not optimal
Q-Learning	(1.39)	2.362 (0.571)	14
Q-Learning	(1.40)	2.752 (0.909)	9
Sarsa	(1.39)	12.991 (0.728)	1
Sarsa	(1.40)	9.541 (0.838)	4

Table 1.5: Comparison of Q-learning and Sarsa on the basis of RMSE of $q(s,a)$ estimate and policy choice. Note each row corresponds to 40 replicates with common seeds for all four cases.

1.5.2 Infinite horizon discounted models

We describe and illustrate Q-learning and Sarsa for infinite horizon discounted models.

Algorithm 1.10. Q-Learning in a discounted model

1. **Initialize**
   (a) $q(s,a) \leftarrow 0$ for each $a \in A_s$ and all $s \in S$.
   (b) $counts(s,a) \leftarrow 0$ for each $a \in A_s$ and all $s \in S$
   (c) Specify the learning rate $\{\tau_n : n = 1, 2, \ldots\}$.
   (d) Select an $s \in S$ and $a \in A_s$.
   (e) Specify the number of episodes K.
   (f) Specify a sequence $\{\epsilon_n \, n = 1, 2, \ldots\}$.
   (g) $n \leftarrow 1$
2. **Re-evaluate** While $n < K$
   (a) $counts(s,a) \leftarrow counts(s,a) + 1$
   (b) Sample s' from $(p(\cdot|s,a))$.
   (c) Choose $a \in A_{s'}$ ϵ_n-greedily.
   (d) **Update** $q(s,a)$**:**

$$q(s,a) \leftarrow q(s,a) + \tau_{counts(s,a)} \left[r(s,a,s') + \lambda \max_{a' \in A_s} q(s',a') - q(s,a) \right] \tag{1.41}$$

   (e) $s \leftarrow s'$.
   (f) $n \leftarrow n + 1$

As noted in the episodic case, an on-policy variant (Sarsa) replaces step 2(d) by

1. Update $q(s,a)$ by

$$q(s,a) \leftarrow q(s,a) + \tau_{counts(s,a)} \left[r(s,a,s') + \lambda q(s',a') - q(s,a) \right]. \tag{1.42}$$

2. $a \leftarrow a'$

At termination the presumed optimal policy, d^∞ can be obtained by choosing any $d(s) \in \arg\max_{a \in A_s} q(s,a)$. Note that it's value can be approximated by $q(s,d(s))$ or obtained using any evaluation method.

Example 1.11. Q-learning and Sarsa in a discounted model. We used Algorithm 1.10 to find an optimal policy for the discounted ($\lambda = 0.9$) version of the two-state model in Section **??**. Our implementation of the algorithm set $\tau_n = 0.5n^{-0.6}$, $K = 30000$ and used ϵ_n-greedy action selection in step 2a with

$$\epsilon_n = 0.01 + 0.1 I_{[0,10000]}(n) + 0.25 I_{[1,5000]}(n)$$

Estimates of $q(s,a)$ using Q-learning (left) and Sarsa (center) and the true value of q obtained from (1.35) (right) follow.

$$q_{QL} = \begin{bmatrix} 27.75 & 29.69 \\ 17.37 & 27.43 \end{bmatrix}, \; q_{Sarsa} = \begin{bmatrix} 11.89 & 8.99 \\ 4.40 & 10.20 \end{bmatrix} \text{ and } q^* = \begin{bmatrix} 29.73 & 30.15 \\ 20.14 & 27.94 \end{bmatrix}.$$

Observe that q_{QL} estimates better approximate q^* and identifies the optimal stationary policy $d(s_1) = a_{1,2}$ and $d(s_2) = a_{2,2}$ in agreement with the calculation in Chapter **??** which showed that d^∞ is optimal and that $v_\lambda^{d^\infty} = (30.15, 27.94)'$. Moreover $q_{QL}(s,d(s))$ provides a good approximation to $v_\lambda^{d^\infty}$.

We were surprised that the Sarsa estimates were so poor in this and many other experiments. In experiments in which Sarsa produced "in the ballpark" estimates, those of Q-learning were even better. In all cases the RMSE of Q-learning was significantly less than that of Sarsa.

We observed also that as λ approached 1 the quality of the approximations deteriorated. In such cases, the convergence could be enhanced by choosing better starting values such as $r/(1-\lambda)$ and using τ_n sequences that converged to zero more slowly than that above. With such starting values, fewer iterations are needed.

Note that Algorithm 1.10 converges almost surely **(check)** when the model is represented by a look-up table and every state-action pair is visited infinitely often.

1.5.3 Infinite horizon average reward models

We describe and illustrate an iterative approach to finding good policies in infinite horizon models with the expected average reward criteria. We focus on generalizations

of Q-learning and Sarsa. The following algorithm provides the Q-learning approach.

Algorithm 1.11. Q-Learning in an average reward model

1. **Initialize**
   (a) $q(s,a) \leftarrow 0$ for each $a \in A_s$ and all $s \in S$.
   (b) $counts(s,a) \leftarrow 0$ for each $a \in A_s$ and all $s \in S$
   (c) Specify number of iterations K.
   (d) Specify learning rates $\{\tau_n : n = 1, 2, \ldots\}$ and $\{\beta_n : n = 1, 2, \ldots\}$.
   (e) Specify ϵ-greedy parameters $\{\epsilon_n : n = 1, 2, \ldots\}$
   (f) Select an $s \in S$ and $a \in A_s$.
   (g) $n \leftarrow 1$
2. **Update q:** While $n < K$:
   (a) $counts(s,a) \leftarrow counts(s,a) + 1$
   (b) Sample $s' \in S$ from $p(\cdot|s,a)$.
   (c) Choose $a' \in A_{s'}$ ϵ_n-greedily.
   (d)
   $$q(s,a) \leftarrow q(s,a) + \tau_{counts(s,a)} \left[r(s,a,s') - g + \max_{a'' \in A_{s'}} q(s',a'') - q(s,a) \right] \tag{1.43}$$
   (e)
   $$g' \leftarrow g + \beta_n (r(s,a,s') - g) \tag{1.44}$$
   (f) $s \leftarrow s'$ and $a \leftarrow a'$.
   (g) $n \leftarrow n + 1$
3. For $s \in S$, choose $d(s) \in \arg\max_{a \in A_s} q(s,a)$.

Some comments follow:

1. Observe that there are 4 model specifications for tuning; the smoothing constants sequences $\{\tau_n\}$ and $\{\beta_n\}$, the rate at which random actions are sampled $\{\epsilon_n\}$ and the number of iterations K. Thus we recommend that you obtain good values for these parameters in the context of policy evaluation through temporal differencing (Algorithm 1.7).
2. As noted earlier in this chapter, the above implementation is off-policy, that is the action chosen for execution at the next iteration need not correspond to the

value $\max_{a'' \in A_s} q(s', a'')$ used in the update (1.43). A Sarsa (on-policy update) would replace step 2d by

$$q(s,a) \leftarrow q(s,a) + \tau_{counts(s,a)} \left[r(s,a,s') - g + q(s',a') - q(s,a) \right].$$

3. Softmax sampling can replace ϵ-greedy sampling in step 2(c).

4. **Relative Value Iteration** In some instances, a generalization of relative value iteration from Chapter **??** may be provide a stable alternative to the above algorithm. In it a state-action pair (s^*, a^*) is distinguished and $q(s^*, a^*)$ is subtracted at each pass through the algorithm. To be precise, steps 2(c) and 2(d) above are replaced by

$$q(s,a) \leftarrow q(s,a) + \tau_{counts(s,a)} \left[r(s,a,s') + \max_{a' \in A_{s'}} q(s,a) - q(s^*,a^*) - q(s,a) \right] \tag{1.45}$$

Note that the quantity $q(s^*, a^*)$ in (1.45) approximates the optimal gain in the limit.

5. If the algorithm is to be implemented off-line, sampling S at the start of step 2 might enhance convergence through more accurate estimate of $q(s,a)$ in some states.

6. We note that the condition $h(s_0) = 0$ does not generalize easily to Q-learning so we propose updating g through (2(a)iii) instead of with (1.30).

Example 1.12. Finding optimal policies in an average reward model.
We used Algorithm 1.11 and its variant in item above above to find an optimal policy for the average reward version of the two-state model in Section **??**. Our calculations in Chapter **??** show that the stationary policy that uses $d(s_1) = a_{1,2}$ and $d(s_2) = a_{2,2}$ is optimal, that $g* = 2.86$ and that $h^*(s_2) = -2.14$ subject to $h^*(s_1) = 0$.

Our implementation of the algorithm sets $\tau_n = 0.6n^{-0.5}$, $\beta_n = 0.8n^{-0.8}$, $K = 30000$ and used an ϵ_n-greedy approach in step 2(e) with

$$\epsilon_n = 0.01 + 0.09 I_{(5000,10000]}(n) + 0.25 I_{[0,5000]}(n).$$

Estimates of $q(s,a)$ using Q-learning (left) and Sarsa (center) and the true value of q obtained from (1.36) with $h^*(s_1) = 0$ (right) follow:

$$q_{QL} = \begin{bmatrix} 34.49 & 37.30 \\ 24.31 & 35.10 \end{bmatrix}, \; q_{Sarsa} = \begin{bmatrix} 5.60 & 0.42 \\ -3.76 & 1.44 \end{bmatrix} \text{ and } q^* = \begin{bmatrix} -0.29 & 0 \\ -10.00 & -2.14 \end{bmatrix}.$$

Observe that the estimates differ greatly from the true value of q^*. This is because neither account for the normalization $h^*(s_1) = 0$ used to estimate q^*.

Re-normalizing these q estimates by subtracting $q(s_1, a_{1,2})$ from each entry and denoting then by $\tilde{q}$ yields respectively

$$\tilde{q}_{QL} = \begin{bmatrix} -2.81 & 0 \\ -12.99 & -2.21 \end{bmatrix} \text{ and } \tilde{q}_{Sarsa} = \begin{bmatrix} 5.18 & 0 \\ -4.18 & 1.44 \end{bmatrix}.$$

We see that the Q-Learning estimate provides a better approximate to q^* then the Sarsa estimate and also identifies the optimal policy. The Sarsa variant provides an inaccurate approximation to q^* and incorrectly designates $d'(s_1) = a_{1,1}$ and $d'(s_2) = a_{2,2}$ as optimal. Moreover $\tilde{q}_{QL}(s, d(s))$ closely approximates $h^*(s)$ since it effectively evaluates the optimal policy after $n = 10000$. Figure 1.10 also shows that Q-learning well estimates g^* while Sarsa does not.

Note that it is not necessary to normalize q to $\tilde{q}$ if one only seeks an optimal policy and not an estimate of h^*. Clearly Q-learning is preferable to Sarsa.

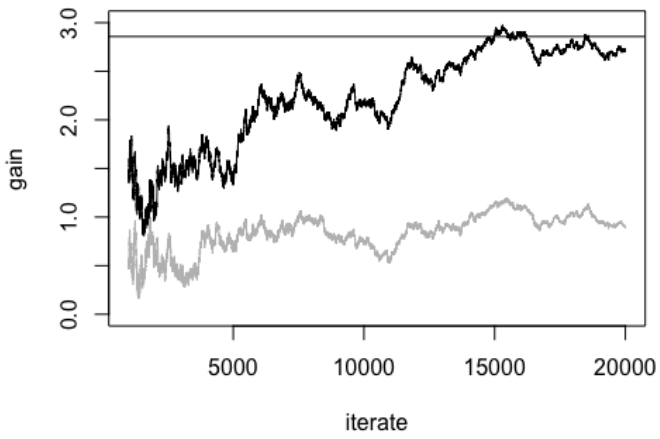


Figure 1.10: Typical Q-Learning (black) and Sarsa (grey) estimates of the average reward obtained from Algorithm 1.11 in Example 1.12. The horizontal line denotes the true value of the gain.

1.6 Policy iteration type algorithms

In this section, we focus on discounted models and leave episodic and average reward models to the reader. The Q-learning and Sarsa algorithms in the previous section may be viewed as adaptations of value iteration algorithms in (Section **??**) to a simulation

environment. Each iterate involved some form of maximization with Q-learning using the "max" to update $q(s, a)$ and Sarsa using a ϵ-greedy or softmax action selection in the update.

As was seen in Chapter **??** policy iteration and modified policy iteration updates resulted in faster convergence so we might expect simulation based versions of policy iteration (Section **??**) and modified policy iteration (Section **??**) may be more efficient. In general, the structure of such algorithms can be represented as in Figure 1.11 represents the flow in a policy iteration type algorithm. In earlier chapters, such an algorithm was implemented exactly using the Markov decision process model. Herein, we consider simulation-based alternatives in which the Evaluation step uses TD(0), TD(γ) or Monte Carlo and the Improvement step uses greedy policy choice. Note we do not consider Sarsa because it performed poorly in infinite horizon models. As a consequence of variability introduced through simulation, the stopping rule "$d' = d$?" requires modification.

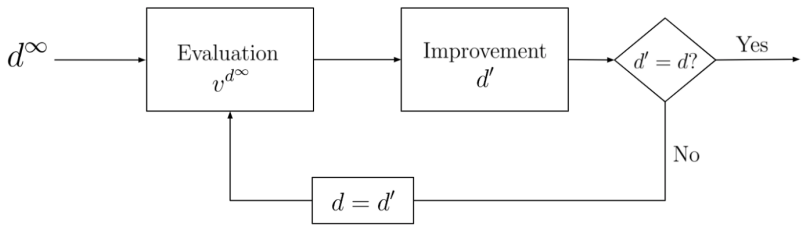


Figure 1.11: Schematic representation of a policy iteration type algorithm

1.6.1 Hybrid policy improvement

First we consider a value-function based algorithm in which the evaluation step uses simulation to estimate the value of a policy and then does "exact" improvement in the same way as deterministic policy iteration in Chapter **??**. While this may be impractical to implement when S and\or A_s are large, it will provide a useful baseline for algorithms in which improvement also involves simulation.

Algorithm 1.12. Hybrid Policy Improvement

1. **Initialization:** Select $d \in D^{MD}$ and set $\Delta = S$.
2. While Δ is non-empty:
   (a) **Evaluation:** Evaluate d using TD(0), TD(γ) or Monte Carlo to obtain $\hat{v}(s)$ for $s \in S$.
   (b) **Exact Improvement:** For all $s \in S$ choose
   $$d'(s) \in \arg\max_{a \in A_s} \left\{ \sum_{j \in S} p(j|s,a)[r(s,a,j) + \lambda \hat{v}(j)] \right\} \tag{1.46}$$
   setting $d'(s) = d(s)$ if possible.
   (c) $\Delta = \{s \in S \,|\, d'(s) \neq d(s)\}$
   (d) $d \leftarrow d'$
3. Return d.

We emphasize that the calculation in (1.46) is equivalent to setting

$$d'(s) = \arg\max_{a \in A_s} q(s,a)$$

where for all $a \in A_s$ and $s \in S$,

$$q(s,a) := p(j|s,a)[r(s,a,j) + \lambda \hat{v}(j)]. \tag{1.47}$$

This algorithm does not apply to model-free settings in which $p(j|s,a)$ and $r(s,a,j)$ are not known. However we can adopt a two-stage approach in which the "model" entities are **estimated** in the first stage to obtain $\hat{p}(j|s,a)$ and $\hat{r}(s,a,j)$ and in the second stage apply Algorithm 1.12 with

$$\hat{q}(s,a) := \hat{p}(j|s,a)[\hat{r}(s,a,j) + \lambda \hat{v}(j)]. \tag{1.48}$$

Note that these estimates can be enhanced while evaluating decision rules.

We explore direct application of Algorithm 1.12 in the following simple example.

Example 1.13. Hybrid policy improvement in the two-state model.
We use Algorithm 1.12 in the two-state state model from Section **??**. We set N=10000 and choose $\tau_n = 0.5n^{-0.6}$ as our learning rate and begin with the sub-optimal policy $d = (a_{1,1}, a_{2,1})$. We use TD(0) to estimate $v_\lambda^{d^\infty}(s)$ for $s = s_1, s_2$.

The corresponding q-functions are

$$\begin{aligned} q(s_1, a_{1,1}) &= 3 + 0.9(0.8\hat{v}(s_1) + 0.2\hat{v}(s_2)) \\ q(s_1, a_{1,2}) &= 5 + 0.9\hat{v}(s_2) \\ q(s_2, a_{2,1}) &= -5 + 0.9\hat{v}(s_2) \\ q(s_1, a_{2,2}) &= 2 + 0.9(0.4\hat{v}(s_1) + 0.6\hat{v}(s_2)) \end{aligned}$$

where $\hat{v}(s)$ is an estimate of $v_\lambda^{d^\infty}(s)$ for the policy being evaluated.

For a wide choice of random number seeds and using both random restart and long-trajectory (state,action)-pair generation, hybrid policy iteration identified the optimal policy $(d^*)^\infty$ where $d = (a_{1,2}, a_{2,2})$ in 3 iterations. Note that for the true value of the optimal policy, the difference between $q(s_1, a_{1,1})$ and $q(s_1, a_{1,2})$ is 0.237 which is 0.7% of the average value of these quantities so that accurate estimates of the value of this policy are required are required to avoid cycling.

1.6.2 Simulated policy improvement

We propose the following alternative implementations of policy iteration that are suitable for model-free applications. Since model-free implementations require q-functions, direct generalization of the above algorithm is problematic since it only "evaluates"

$$q(s, d(s)) = v_\lambda^{d^\infty}(s)$$

and does not provide estimates of $q(s, a)$ for other choices of $a \in A_s$. We provide two ways to get around this. The first is quite straightforward. After evaluating a policy d^∞, it uses its estimated value to simulate the q-function for other state action pairs.

A straightforward approach

We provide an algorithm that first estimates the value of a policy and then uses it to estimate the q-functions associated with that policy.

Algorithm 1.13. Two-step q-function estimation

1. **Initialization:** Select $d \in D^{MD}$ and set $\Delta = S$.
2. While Δ is non-empty:
   (a) **Evaluation:** Evaluate d using TD(0), TD(γ) or Monte Carlo to obtain $\hat{v}(s)$ for $s \in S$..
   (b) **Simulation-based q-function estimation:**
      i. $q(s,a) \leftarrow 0$ and $count(s,a) \leftarrow 0$ for $s \in S$ and $a \in A_s$.
      ii. Specify number of iterates N and a learning rate sequence $\{\tau_n : n = 1, 2, \ldots\}$
      iii. $n \leftarrow 1$.
      iv. For all $s \in S$ and $a \in A_s$.
      v. While $n < N$:
         A. $count(s,a) \leftarrow count(s,a) + 1$.
         B. Sample $s' \in S$ from $p(\cdot|s,a)$.
         C.
$$q(s,a) \leftarrow q(s,a) + \tau_{count(s,a)} \left[r(s,a,s') + \lambda \hat{v}(s') - q(s,a) \right]. \tag{1.49}$$
         D. $n \leftarrow n + 1$.
   (c) **Policy improvement:** Choose
$$d'(s) \in \arg\max_{a \in A_s} q(s,a) \tag{1.50}$$
   setting $d'(s) = d(s)$ if possible.
   (d) $\Delta = \{s \in S | d'(s) \neq d(s)\}$
   (e) $d \leftarrow d'$
3. Return d.

We comment briefly on the above algorithm.

1. Step 2(b) use TD(0) to estimate $q(s,a)$ for all state-action pairs with $\hat{v}(s)$ obtained i the evalaution step substituted in (1.49).
2. The algorithm resamples s' at each iterate, for the specified (s,a)-pair. Alternatively, one can choose an action in $A_{s'}$ and iterate to generate a long sequence.
3. The condition "setting $d'(s) = d(s)$ if possible" is included to avoided cycling when there are multiple optima in some states.

4. The stopping criterion "While Δ is non-empty" may be too stringent because of variability in the q-function estimates. An alternative stopping criterion to consider is

   "Stop when $|\Delta| \leq C$"

   where $|B|$ denotes the number of elements in the set B and C is a pre-specified constant.

5. The choice of N and $\{\tau_n : n = 1, 2, \ldots\}$ should be guided by experimentation when evaluating a fixed policy using Algorithm 1.3.

6. When the state space is ordered (such as in a queuing or inventory model) or partially ordered (such as in a multi-product inventory model or multi-class queuing model), smoothing or approximating $q(s, a)$ prior to selecting a decision rule as in Chapter **??** may enhance convergence.

Example 1.14. We apply Algorithm 1.13 to Example **??**. We consider long trajectory and random restart implementations of TD(0) (Algorithm 1.5) in the evaluation step. We initialize the algorithm with decision rule $d = (a_{1,1}, a_{2,1})$, use $\tau_k = 0.5k^{-.6}$ throughout, set $K = 25000$ for evaluation and $K = 5000$ when estimating $q(s, a)$.

Over a wide range of random number sequences, both implementations terminate with the optimal decision rule $d^* = (a_{1,2}, a_{2,2})$ after 3 iterations.

Online policy iteration

To overcome the absence of estimates of $q(s, a)$ for $a \neq d(s)$ when using Algorithms 1.12 or 1.13 the proposed algorithm (which has both long trajectory (online) and random restart variants) estimates $q(s, a)$ for all states and actions by replacing a deterministic policy d^∞ by a randomized policy that is "close" to d^∞. To do this, the algorithm evaluates an ϵ-modified[12] policy $\tilde{d}$ that chooses actions in state s according to

$$w_{\tilde{d}}(a|s) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|S|} & s = d(s) \\ \frac{\epsilon}{|S|-1} & s \neq d(s). \end{cases} \tag{1.51}$$

where $w_{\tilde{d}}(a|s)$ was defined in Chapter **??** as the probability that decision rule $\tilde{d}$ chooses action a in state s. This can be implemented by sampling action $d(s)$ with probability $1 - \epsilon$ and choosing a random action with probability ϵ. Of course, ϵ should decrease over the course of the algorithm.

[12] This is not an ϵ-greedy policy because the policy it is based on need not be optimal.

Algorithm 1.14. Online Policy Improvement

1. **Initialization:**
   (a) Select $d \in D^{MD}$ and the number of loops K.
   (b) Specify $\{\epsilon_n : n = 1, 2, \ldots\}$.
   (c) $n \leftarrow 1$.
2. While $n < K$:
   (a) **Evaluate ϵ-modified policy**
      i. Specify the ϵ_n-modified policy $\tilde{d}$.
      ii. Evaluate $\tilde{d}$ using TD(0) or TD(γ) to obtain $\hat{q}(s, a)$ for $a \in A_s$ and $s \in S$.
   (b) **Greedy Improvement:** For all $s \in S$ choose

$$d'(s) \in \arg\max_{a \in A_s} q(s, a) \tag{1.52}$$

   setting $d'(s) = d(s)$ if possible.
   (c) $d \leftarrow d'$ and $n \leftarrow n + 1$.
3. Return d.

Some comments about applying this algorithm follow:

1. Choosing $\epsilon_n = 1$ initiates the algorithm with a decision rule which randomizes over all actions. In general ϵ_n should decrease with n.
2. The stopping rule, "stop when a decision rule repeats", was too stringent when q-functions were estimated directly. Instead we recommend than one specify the number of times to execute Step 2 or instead stop when d' and d differ in a small number of states.
3. One can implement TD(0) or TD(γ) using either long trajectories or random restart at each iteration. The latter would be not be suitable for learning in real processes unless single actions can be easily repeated.
4. There are two options for the update in the evaluation step. An on-policy variant (analogous to Sarsa) which uses the *randomized* decision rule, $\tilde{d}$ in the target state and an off-policy variant (analogous to Q-learning) that uses the *deterministic* decision rule d in the target state[13]

We apply this algorithm to a the two-state example.

[13]This decision rule can be either specified in initialization or greedily in step 2(b).

Example 1.15. We apply Algorithm 1.14 to the two-state example using TD(0) to estimate $q(s,a)$ with the same choices for τ_n and K as in the previous example. We choose $\epsilon_n = n^{-.85}$ for $n = 1, 2, \ldots$ and terminate the algorithm after 50 improvement steps.

In most instances the algorithm either found the optimal policy $(d^*)^\infty$ where $d^* = (a_{1,2}, a_{2,2})$ or oscillated between the optimal policy and that based on decision rule $d = (a_{1,1}, a_{2,2})$. Relative to the other stationary policies, these two are closest in value. Such cycling, has been noted and explained in in the literature.

The random restart version terminated with the optimal policy more frequently than the long trajectory approach. The numerical experiments showed that the stopping rule "stop when a policy repeats" terminated early with sub-optimal policies and is not recommended.

Concluding remarks regarding policy iteration

Based on our numerical experiments, it appears that algorithms which estimate the value-function and then use it to estimate the q-function were convergent while the online implementation of policy iteration did not consistently identify the optimal policy and sometimes cycled. We revisit this issue in our in depth analysis of the queueing service rate control model.

1.7 Queuing service rate control revisited

The infinite horizon version of the queuing service rate control model introduced in Section **??** provides a good platform to investigate challenges that arise when implementing the above algorithms in a simple but more realistic model. Particular features that make this problem attractive for analysis include

1. the ordered and interpretable state space,
2. the known structure of value functions, and
3. the monotonicity of the optimal policy.

We focus on the discounted version but also discuss the average reward version. Our objective is to find an optimal policy for future use so it's not necessary to use an online implementation.

Even though our implementation involves only 51 states with 3 actions in each, the problem is sufficiently complex to raise a range of implementation issues; issues that must be considered when optimizing modern industrial strength applications. We experiment with a wide range of options and hopefully provide a framework that may be applied broadly.

We truncate the state space at 50, set the arrival rate $q = 0.2$, service rates $a_k \in \{0.2, 0.4, 0.6\}$, the holding cost, $f(s) = s^2$, the service rate cost $m(a) = 5a^3$ and the discount rate $\lambda = 0.9$. Because the objective is to minimize the expected discounted cost, we solve this as a **minimization** problem with positive costs and consequently replace "arg max" by "arg min" in algorithms.

Using value iteration and policy iteration (Chapter **??**) we found that when $\lambda = 0.9$ the optimal policy $(d^*)^\infty$ had the form

$$d^*(s) = \begin{cases} a_1 = 0.2 & s \in \{0, \ldots, 10\} \\ a_2 = 0.4 & s \in \{11, \ldots, 29\} \\ a_3 = 0.6 & s \in \{30, \ldots, 50\} \end{cases} \tag{1.53}$$

and the value function was monotone increasing and had monotone increasing differences (was convex) on $\{0, 1, \ldots, 48\}$[14]. Under the average reward criterion, the optimal policy uses actions $a_1 = 0.2$ in states $\{0, 1, 2\}$, $a_2 = 0.4$ in states $\{3, \ldots, 8\}$ and $a_3 = 0.6$ in states $\{9, \ldots, 50\}$.

Policy Evaluation

Evaluation in this model presents some challenges. Under the policy that uses action a_1 in every state, the system is equivalent to a symmetric reflecting random walk on $\{0, \ldots, 50\}$ so that all states are visited with equal probability in steady state. Therefore evaluation of such a policy would be straightforward since all states will be visited approximately the same number of times in a simulation. However evaluating other policies becomes problematic because the system primarily visits the low occupancy states[15] so that estimates of value functions or state-action value functions using long trajectories will be inaccurate in high occupancy states. Hence we would expect that random sampling states (or state-action pairs) after each transition would improve the accuracy of Monte Carlo and TD(γ) methods. Hence we considered this option only below.

We estimated the discounted value (for lambda=0.9) of the stationary policy that used action a_2 in every state using 10000 replicates of TD(γ) for each seed, learning rate constant, γ and trace type. Preliminary estimates were used to select learning rates for subsequent analysis. Comparisons were based on the RMSE over the states. We found (see Figure 1.12 that

1. for high values of γ the replacement trace had smaller RMSE than the accumulating trace. This is probably a result of using state resampling at each iteration.

2. $\tau_n = 0.65n^{-0.5}$ gave the lowest and least variable RMSE, and

[14] As a result of truncation the values for $s \in \{49, 50\}$ deviated from this pattern.

[15] We leave it as an exercise to compute the stationary distribution under policies that use action a_2 or a_3 in all states.

3. for this choice of τ_n, TD(0) provided the best value function fit.

Figure 1.13b shows that the TD(0) value function estimates deviates from the true value function by most 100. On a relative basis this error is at most 3%. Based on this analysis we will use TD(0) in subsequent analyses.

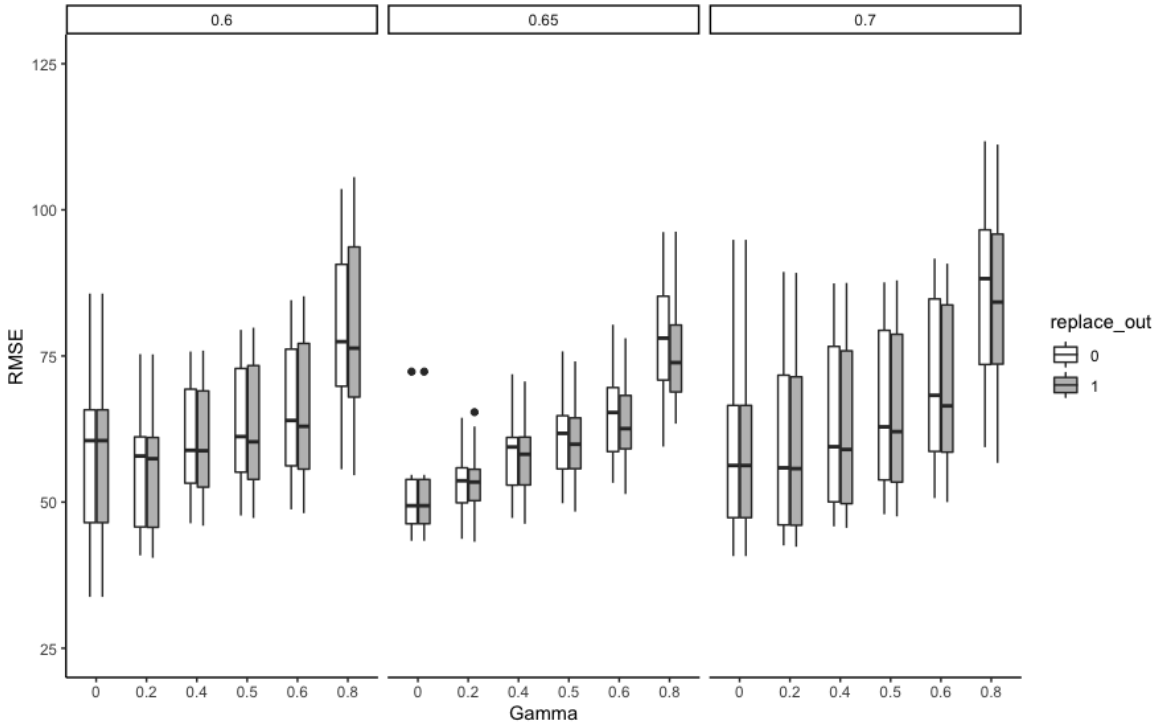
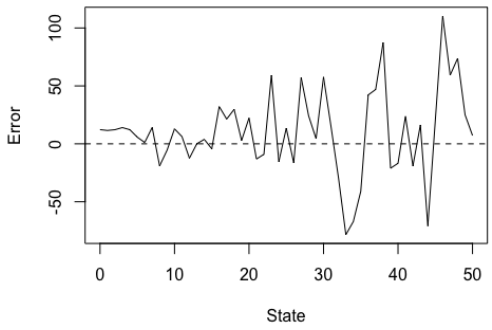


Figure 1.12: Graphical summary of dependence of RMSE on parameter choice using TD(γ) for value function approximation using the policy d^∞ with $d(s) = a_2$ for all $s \in S$. Each boxplot represents a summary over 10 replicates. The white boxes correspond to accumulating trace and grey boxes to replacement trace, the choice of γ appears on the horizontal axis and the top scale corresponds to the choice of constant in τ_n.

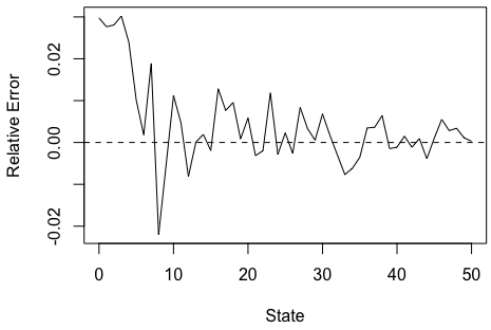
Some comments regarding Monte Carlo estimates based on truncation and geometric stopping times follow:

1. *Only starting state value function estimators are valid.* This is because the truncation horizon or geometric stopping time are defined in terms of the length of the horizon starting from the first transition. Hence first visit estimates, which would start later, would correspond to shorter horizons.

2. In this model, the estimated standard deviation $\hat{\sigma}(s)$ of $\hat{v}(s)$ (Figure 1.14a) varied over the states. Note that the "artificial" boundary at $s = 50$ resulted in the non-monotone behavior near the upper boundary. Thus sample sizes (number of



(a) Difference between estimated and true value function.



(b) Relative difference between estimated and true value function.

Figure 1.13: Error and relative error using TD(0) with learning rate $\tau_n = 0.65n^{-0.5}$ for estimating value of d^∞ where $d(s) = a_2$ for all $s \in S$.

Monte Carlo replicates) needed to obtain the same precision for all states must vary with state.

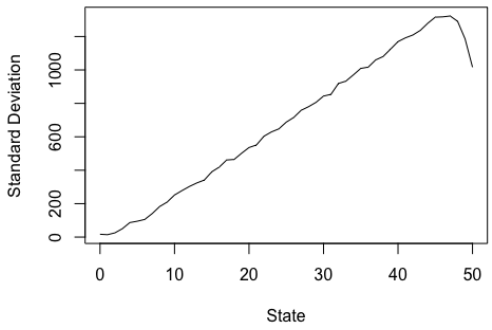
3. By the central limit theorem, the Monte Carlo estimate in state s is asymptotically normal with standard deviation approximately equal to $\hat{\sigma}(s)/n^{-0.5}$. Hence this relationship can be used to choose the sample size to obtain a pre-specified level of confidence.

4. As a result of previous comment, we replicated the Monte Carlo truncation estimator $\max\{100, 4s^2\}$ times in state s. Differences between the estimated and true value function are shown in Figure 1.14b. Note the RMSE of these estimates (over states) was 12.29. These estimates were more precise than the TD(0) estimates (Figure 1.13b however the total number of replicates was considerably larger.

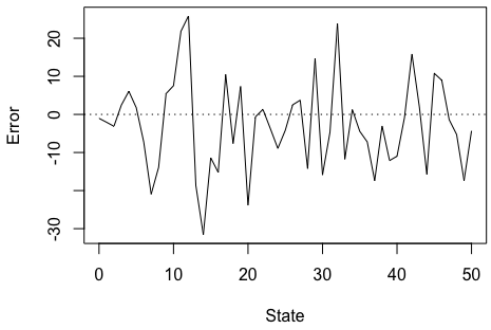
5. Geometric stopping time estimates were considerably more variable than the truncation estimates (standard deviation in state 50 was 200 times greater than that of the truncation estimator) and much less accurate overall (RMSE =103.7). Note that the expected number of terms in the geometric sums was 11 as opposed to 125 used by truncation.

1.7.1 Q-learning

We solve the discounted version of the queuing control problem using Algorithm 1.10. Based on results above, we set $\tau_n = 0.6n^{-0.5}$ and use random state regeneration at each iteration. We use common seeds for each choice of $K = 150000, 300000, 450000, 600000$



(a) Estimated standard deviation of Monte Carlo value function estimates.



(b) Difference (in one experiment) between estimated and true value function. The number of replicates in state s was set at $\max\{100, 4s^2\}$.

Figure 1.14: Some results for Monte Carlo estimates based on truncation at $N = 125$.

and ϵ-greedy action selection with $\epsilon = 0.1$. Since the optimal value function and policy was previously computed using policy iteration (Chapter **??**), the quality of these estimates can be assessed on the basis of RMSE over states. To clarify, the RMSE compares the value function[16] of the greedy policy identified by Q-learning and the optimal value function.

RMSEs from this analysis appear in Table 1.6. The table shows that the value of the policy identified through greedy action selection using the estimated state-action value function was close to the optimal value function. It suggests that choosing $K = 450000$, which is roughly 3000 times the number of state action pairs, produced good estimates.

Iterations	RMSE - Mean	RMSE - Standard Deviation
150000	114.52	59.23
300000	59.79	37.52
450000	40.71	23.31
600000	39.74	26.45

Table 1.6: Accuracy of Q-learning estimates as a function of the number of iterations averaged over 20 random number seeds.

Figure 1.15 shows the decision rule corresponding to the greedy policy (in circles) and the optimal policy (solid line) for a typical replicate. Observe, that the policies are quite similar but differ in several states. Moreover the the greedy decision rule is not monotone. Smoothing the state-action value function or using the function approximation methods in the next section should produce monotone policies.

[16]This is found by solving $(\mathbf{I} - \lambda \mathbf{P}_d)\mathbf{v} = \mathbf{r}_d$. It need not equal $\max_{a \in A_s} q(s, a)$.

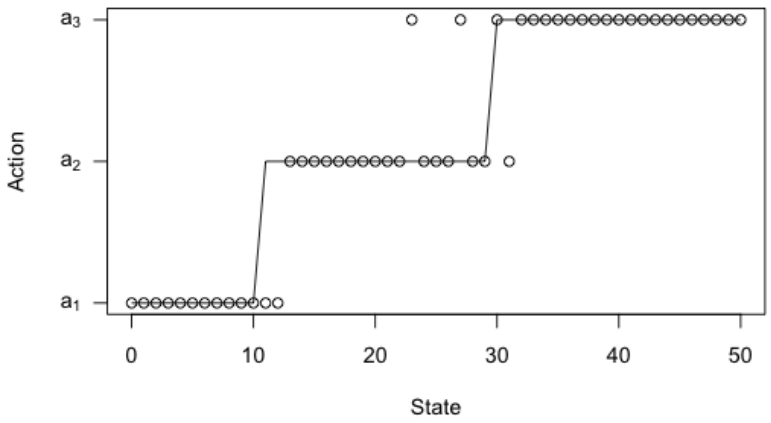


Figure 1.15: Comparison of action selected by a greedy policy (circles) and optimal policy (line) as a function of the state based on a typical replicate using Q-learning with $K = 450000$.

1.7.2 Approximate policy iteration

Approximate policy iteration offers a wide range of options for choosing and implementing an algorithm. We proposed three approximate policy iteration algorithms (Algorithms 1.12, 1.13 and 1.14). The first two evaluate a policy and then compute the q-function in the improvement step, while the third algorithm uses q-functions directly. Issues to address in implementing them[17]:

1. policy evaluation method: TD(0), TD(γ), or Monte Carlo with truncated discounted rewards or geometric stopping,
2. algorithmic parameter choice: $\{\tau_n\}$, $\{\epsilon_n\}$ when using online policy iteration and the number of replicates in evaluation and improvement,
3. next state generation: random restart or trajectory based
4. exact or estimated q-functions in the improvement step,
5. smoothed or unsmoothed value function estimates,
6. smoothed or unsmoothed q-function estimates,
7. the stopping criterion
8. the smoothing method

[17]Not all issues apply to each method.

We compared results on the basis of whether the algorithms terminated or cycled, how similar the identified policy was to the optimal policy and how well the value function approximated the optimal value function.

Smoothing

To ensure convergence in this example, it was often necessary to smooth $q(s, a)$ as a function of s prior to greedy improvement in step 2(b). We used least squares to fit a *quadratic B-spline*[18] Chapter **??** provides a systematic analysis of combined simulation and function approximation.

Hybrid policy iteration: Implementation

Algorithm 1.12 combines value function estimation to determine $\hat{v}(s)$ with greedy improvement. It requires an Markov decision process model in order to evaluate (1.46) which becomes

$$q(s,a) = \sum_{j \in S} p(j|s,a)[r(s,a,j) + \hat{v}(j)] = \tag{1.54}$$
$$\begin{cases} 5a^3 + (1-q)\hat{v}(0) + q\hat{v}(1) & s = 0 \\ s^2 + 5a^3 + a\hat{v}(s-1) + (1-a-q)\hat{v}(s) + q\hat{v}(s+1) & s \in \{1, 2, \ldots, 49\} \\ 50^2 + 5a^3 + a\hat{v}(49) + (1-a)\hat{v}(50) & s = 50 \end{cases}.$$

Note that $r(s, a, j)$ is independent of j.

Evaluation used TD(0) and starting-state Monte Carlo with truncation and geometric stopping. Improvement used either unsmoothed or smoothed q-functions. For each of these six combinations, we ran 30 replicates with common across each configuration.

Details of the implementation follows:

1. TD(0) parameters were specified as $\tau_n = 0.65n^{-0.5}$ and 500000[19] replicates. States were re-sampled after each transition.

2. Starting-state Monte Carlo was truncated after 125 iterations. Both truncated and geometric stopping versions used 10000 replicates with starting states randomly sampled and estimates from a replicate based on long trajectory transitions. To avoid storing past values, value functions were updated after completing a replicate using the recursive formula for a mean

$$v^{n+1}(s) = v^n(s) + \frac{1}{n}[\tilde{v}(s) - v^n(s)]$$

[18] B-splines are linear combinations of polynomials that provide flexible fits of non-linear functions. In R [2021], they are specified with the function *bs()* and fit with function *lm()*. To avoid pre-specifying the location of *knots* our implementation used the specification degree=2 and df=2.

[19] The algorithm did not reliable converge when the number of replicates was set to 100000.

where $\tilde{v}(s)$ is the estimate of the value function in state s at the nth replicate starting in state s and $v^n(s)$ is the recursive estimate of the mean after $n-1$ replicates.

The stopping criterion was "stop when $|\Delta| \leq 2$" where Δ denotes[20] the set of states on which the policy at two successive iterates differ. Note that more restrictive conditions lead to cycling.

Hybrid Policy Iteration: Results

Table 1.7 shows the results of the calculations. Without smoothing, only the estimate using TD(0) achieved the stopping criteria. In this case it terminated on average after 4.57 iterations and the value function of the identified policy was close to optimal. Combined TD(0) with smoothing resulted in 29 out of 30 instances terminating with the optimal policy.

The Monte Carlo methods required smoothing to converge. The hybrid policy iteration algorithm identified the optimal policy in two iterations for every instance while that based on geometric stopping converged quickly and generated policies with either optimal or close to optimal values. Note that on average geometric stopping truncated the sum of rewards after 10 iterations while the more computationally intensive truncated discounted reward method truncated the discounted sum of rewards after 125 iterations.

Evaluation Method	Smoothing q-function	Average Iterations	Range Iterations	Average RMSE	Range RMSE
TD(0)	No	4.57	(2, 14)	13.20	(0, 49.10)
	Yes	2	2	0.51	(0, 1.53)
Truncation	No	Did not converge	-	-	-
	Yes	2	2	0	0
Geometric Stopping	No	Did not converge	-	-	-
	Yes	3.2	(2, 7)	7.07	(0, 25.43)

Table 1.7: Comparison of evaluation methods for Algorithm 1.12. Summaries are over 30 random seeds.

Two step q-function estimation.

Next we use Algorithm 1.13. To determine how an unsmoothed algorithm would behave under ideal conditions, we set $\hat{v}(s)$ equal to the **optimal** value function (obtained exactly using policy iteration on the known model) and apply step 2(b) with $N = 5000$ and $\tau_n = 0.65n^{-0.5}$ as above.

[20] See Step 2(c) in the algorithm.

We observe that greedy policy differs considerably from the optimal policy (it chose non-optimal actions in 18 states) and its value exceeds that of the optimal policy by at most 256. On the other hand after smoothing the estimated q-function, the greedy policy is monotone, differs from the optimum in 2 states and the value function of this policy differed from the value of the optimal policy by at most 3.3. This analysis suggests that when using estimated value functions, q-function smoothing is required.

We then ran Algorithm 1.13 using TD(0) for evaluation (with specifications identical to the previous section), q-function estimation with $N = 2000$ and smoothing. Results are summarized in Table 1.8.

Evaluation Method	Smoothing q-function	Average Iterations	Range Iterations	Average RMSE	Range RMSE
TD(0)	Yes	4.47	(2,17)	25.6	(0,97.4)

Table 1.8: Summary of results of applying Algorithm 1.13 over 30 random seeds.

Online policy improvement: implementation

We implemented online policy improvement (Algorithm 1.14) using 500000 iterates of TD(0) for state-action value evaluation and set other parameters values to $K = 25$, $\tau_n = 0.65n^{-0.5}$ and $\epsilon_n = 0.5n^{-0.1}$. Subsequent states were randomly generated and actions were sampled using an ϵ-altered decision rule $\tilde{d}$. We compared using an on-policy version that updates the state-action value function according to

$$q(s,a) = q(s,a) + \tau_n[r(s,a,\tilde{d}(s)) + \lambda q(s,\tilde{d}(s)) - q(s,a)] \tag{1.55}$$

with an off-policy version which replaces the ϵ_n-altered decision rule $\tilde{d}$ in the above expression by the greedy decision rule d. Algorithm 1.14 as stated, does not specify a stopping rule. We again use "stop when the number of states in which $d'(s)$ differs from $d(s)$ is at most 2" as a stopping criterion.

Online policy iteration: results

We found over a range of seeds that decision rules generated using the on-policy and off-policy variants of (1.55) did not satisfy the proposed stopping criterion even after running 25 iterations of the algorithm[21]. In fact between 20 and 30 actions changed at each replicate.

Therefore we smoothed q-functions prior to selecting a greedy update. We compared the on-policy and off-policy variants of (1.55) using 30 common random seeds. In each, we randomly sampled the state after each transition.

Table 1.9 provides a summary of results. It shows that the off-policy algorithm converge in fewer iterations on average. Pairwise comparisons between instances showed

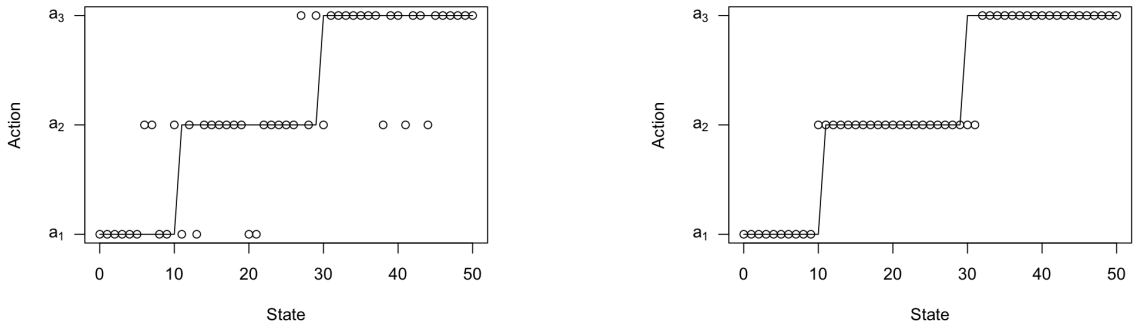
[21] This correspond to 25×500000 TD(0) steps

showed that the off-policy method required fewer iterations than the on-policy method in 6 out 0f 30 instances.

Method	Smoothing q-function	Average Iterations	Range Iterations	Average RMSE	Range RMSE
Off-policy	Yes	9.57	(2,45)	30.6	(0,159.4)
On-policy	Yes	6.90	(2,19)	35.6	(0,159.4)

Table 1.9: Summary of results of applying Algorithm 1.14 over 30 random seeds. Note that the off-policy variant required 26 and 46 iterations to converge in two instances.

Figure 1.16 shows decision rules determined by the smoothed and unsmoothed variants of on line policy iteration using a common seed. Clearly, the decision rule determined by the smoothed variant is superior to that determined by the unsmoothed variant. Moreover, the RMSE of the difference of the exact value function of the unsmoothed policy and that of the optimal policy was 501.18 and that of the policy based on smoothing $q(s, a)$ prior to greedy action selection was 36.45.



(a) Unsmoothed q-function.

(b) Smoothed q-function.

Figure 1.16: Decision rule (black circles) found using online policy iteration and that corresponding to the optimal policy (black lines) using the off-policy variant with a common seed.

We conclude that when using on line policy iteration, smoothing is required to obtain convergence and as well, identify a policy with the same structure as the optimal policy. This suggests that function approximation can be beneficial in the context of simulation. We pursue a more systematic approach to this issue in the next chapter.

Computational Summary

In our analysis of the discounted queuing control model we observed that:

1. TD(0) provided reliable estimates of a value function. TD(γ) estimates were less accurate.

2. Starting-state Monte Carlo provided accurate estimates but required more replicates than TD(0) to obtain the same degree of precision. Those using truncated discounted reward were more accurate than those using geometric sampling.

3. Q-learning estimates produced policies that were close to optimal. The policies differed from the optimal policies in a few states. We suspect that smoothing the q-functions prior to policy choice would result in monotone policies.

4. Policy iteration algorithms, behaved best with a stopping criterion of the form "Stop when the decision rule at successive iterates changes in at most two states." A more stringent stopping rule that requ ired agreement in fewer than two states often lead to cycling.

5. Only hybrid policy algorithm with TD(0) value function estimates terminated without q-function smoothing. However its behavior was greatly enhanced by smoothing. We did not investigate value-function smoothing.

6. Two-step q-function estimation required smoothing for convergence.

7. Online policy iteration required q-function smoothing for convergence. The on-policy variant convergence more reliably.

The above observations suggest that when a model is available, hybrid policy iteration with Monte Carlo value function estimation using truncation was most reliable and precise. In the absence of a model, variants of Q-learning, two-step q-function estimation and online policy iteration provided good alternatives to consider. Methods were most reliable when states were sampled at each transition.

Smoothing the q-function enhanced convergence and produced monotone policies. Our approach to smoothing involved estimating the q-function for all state action pairs and then using least squares to smooth the q-function. This was possible in a small model, the next chapter investigates incorporating function estimation directly in the algorithm and generalizes to models with large state spaces.

The methods in this chapter provide the analyst with a "toolbox" of potential algorithms to consider. However each requires considerable tuning to work well. Moreover its not obvious *a priori* which will work best in a given application.

1.7.3 Average reward queuing control

We applied Algorithm 1.11 to an average reward version of the queuing service rate control model with 51 state and 15 states Even in sample sizes on the order of 10^6, we

observed that an on-line (Markov chain updating) version of the algorithm identified highly non-optimal, unstructured policies and provided inaccurate estimates of g under a wide range of parameter choices and random sequences. The reason for this is under the average reward criterion, good policies control the system so that it remains in low-occupancy states. Hence a large number of the state-action pairs (corresponding to higher occupancy states) were infrequently visited and evaluated so that $q(s, a)$ estimates were inaccurate. Several other versions of the algorithm in the literature also produced poor results.

Since, under the average reward criterion, transient behavior does not effect the gain, on-line implementations seem inappropriate. We found that resampling states at each iteration combined with a relative value approach based on recursion (1.45) with $q(0, a_1)$ distinguished as (s^*, a^*), provided good estimates of g and after smoothing, policies that were close to optimal.

Figure 1.17 shows the policies identified when the procedure was run for $150,000$ and $500,000$ iterates. Observe that in the figure on the left, the policy without smoothing (red circles) was unstructured and very different than the optimal stationary policy (blue); however after smoothing the estimates of $q(s, a)$ with a quadratic spline, the policy (black line) was almost identical to the optimal policy (blue line). With the increased sample size the unsmoothed estimates of $q(s, a)$ was very close to optimal with only 46 and 48 deviating from optimum.

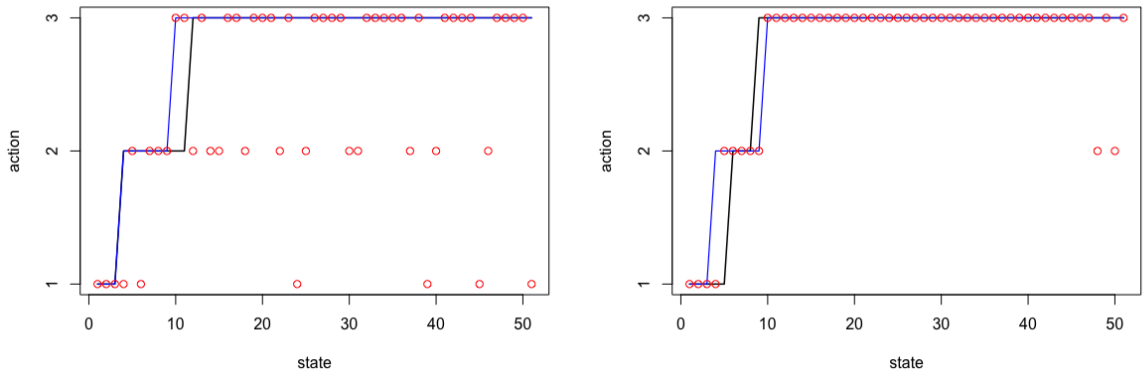


Figure 1.17: Stationary policy found using smoothed (solid black line) and unsmoothed (open red circles) q-functions when analyzing the queuing model by Q-learning. The blue line gives the average optimal stationary policy. In both cases, the learning rate was $.6counts(s, a)^{.5}$ and updates were based on (1.45). The figure on the left corresponds to 150000 iterations and and the figure on the right corresponds to 500000 iterations. Results are based on a single simulation run with identical seed and the same number of replicates.

The estimates of g were 21.06 when sample size was $150,000$ and 20.56 when sample

size was 500,000. The optimal g was 19.43. We explain this inaccuracy by noting that the identified policy was not optimal.

Based on this analysis of average reward models, we recommend using relative Q-learning with state sampling, q-function smoothing and a sufficiently large sample size.

1.8 Enhanced Stopping Rules

Chapter **??** contains bounds and stopping rules which turn iterative procedures into algorithms. In this section we discuss how they might be incorporated into the simulation based procedures above. To the best of our knowledge, they have not been used in this context.

Theorem **??** shows that for any function $v(s)$ on S,

$$\begin{aligned} v(s) + (1-\lambda)^{-1} \min_{s' \in S} (Lv(s') - v(s')) \leq v_\lambda^{d^\infty}(s) \\ \leq v_\lambda^*(s) \leq v(s) + (1-\lambda)^{-1} \max_{s' \in S} (Lv(s') - v(s')) \end{aligned} \tag{1.56}$$

where

$$Lv(s) = \max_{a \in A_s} \{ \sum_{j \in S} p(j|s,a) \left[r(s,a,j) + \lambda v(j) \right] \} = \max_{a \in A_s} q(s,a)$$

and

$$d(s) \in \arg\max_{a \in A_s} q(s,a).$$

Suppose now that $\hat{v}$ was determined in the evaluation step of hybrid policy iteration (Algorithm 1.12 . Then direct substitution shows that (1.56) can be written as

$$\begin{aligned} \hat{v}(s) + (1-\lambda)^{-1} \min_{s' \in S} \left[\max_{a' \in A_{s'}} q(s',a') - \hat{v}(s') \right] \leq v_\lambda^{d^\infty}(s) \\ \leq v_\lambda^*(s) \leq \hat{v}(s) + (1-\lambda)^{-1} \max_{s' \in S} \left[\max_{a' \in A_{s'}} q(s',a') - \hat{v}(s') \right] \end{aligned} \tag{1.57}$$

Since all of the quantities in (1.57) are available at each iteration of hybrid policy iteration, the above bounds can provide the basis for the stopping criterion based on

$$\max_{s' \in S} \left[\max_{a' \in A_{s'}} q(s',a') - \hat{v}(s') \right] - \min_{s' \in S} \left[\max_{a' \in A_{s'}} q(s',a') - \hat{v}(s') \right]$$

This analysis which was based on hybrid policy iteration generalizes (without formal justification) to Algorithm 1.13 if we replace the exact $q(s,a)$ by its estimate and to Algorithm **??** if ..

(To be continued)

Appendix A

Choosing good estimators

Methods in this chapter are concerned with estimates of an unknown quantity C based on a sequence of observations $X = (x^1, x^2, \ldots, x^N)$. The estimate may use the whole data set, such as a sample mean, or be obtained on-line and updated sequentially such as by the stochastic approximation algorithms described in Appendix B. Basic concepts from statistical estimation, such as bias, absolute bias, variance and mean squared error enable us to assess the quality of an estimator.

In most of the examples in this chapter, we will need to choose among estimates such as those represented by the different colored lines in Figure A.1. The estimates may be generated by different methods or from a single method using different parameter choices. Observe that the light grey estimate is the least variable but does not converge to the true value given by the horizontal black line, in other words, it is biased. The dark gray and black estimates both oscillate around the true value but the black is less variable. Consequently (as we're sure you'd agree) we would would prefer the estimator corresponding to the black line. Thus when choosing an estimator we need to trade-off bias and its variability. We quantify this trade-off below.

But some caveats are in order. The estimates in Figure A.1 are based on a single common sequence (using the same random number seed). Results may vary with different sequences. Also in practice we will choose a single estimation method and a pre-specified number of iterates. Clearly if we stopped sampling after approximately 4200 iterates, we would have found that the estimate based on the grey estimator gave the best approximation of the unknown quantity. By looking at plots such as these we are able to choose estimators that produce good estimates over a range of sample sizes.

RMSE for point estimation

Let $Y = (y^1, y^2, \ldots, y^N)$ denote a sequence of estimates of a single quantity C where y^k is based on the first k observations from a simulation experiment. Letting $\bar{Y}$ denote the mean of the estimates, define the *bias* by

$$bias(Y) := |\bar{Y} - C|.$$

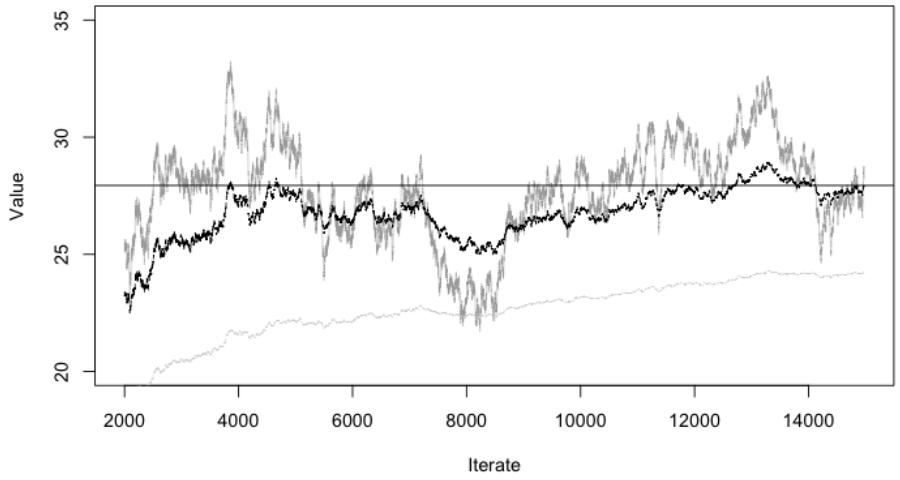


Figure A.1: Three sequences of estimates of a value from a recursive algorithm based on runs of length 15000. Each line corresponds to a different estimator. The horizontal line represents the true value (which is unknown in practice but known in experiments such as those throughout this book).

Other summary measures include the (population) *variance*, given by

$$var(Y) := \frac{1}{N}\sum_{k=1}^{N}(y^k - \bar{Y})^2$$

where $\bar{Y}$ denotes the mean of the $\{y^k; k = 1, \ldots, N\}$ and the *mean squared error (MSE)*given by

$$MSE(Y) := \frac{1}{N}\sum_{k=1}^{N}(y^k - C)^2.$$

We prefer using the *root mean squared error* (RMSE) given by

$$RMSE(Y) = \sqrt{MSE(Y)}$$

because its values are on the same scale as the original data. Note that the variance and MSE measure deviations from different quantities, the sample mean and true parameter respectively.

These quantities are related through the expression:

$$MSE(Y) = \frac{1}{N}\sum_{k=1}^{N}(y^k - \bar{Y})^2 + (\bar{Y} - C)^2 = var(Y) + bias(Y)^2 \qquad \text{(A.1)}$$

Hence the mean squared error contains both a variance and a bias component. To see this, note that

$$\begin{aligned}\sum_{k=1}^{N}(y^k - C)^2 &= \sum_{k=1}^{N}(y^k - \bar{Y} + \bar{Y} - C)^2 \\ &= \sum_{k=1}^{N}(y^k - \bar{Y})^2 + \sum_{k=1}^{N} 2(y^k - \bar{Y})(\bar{Y} - C) + \sum_{k=1}^{N}(\bar{Y} - C)^2 \\ &= \sum_{k=1}^{N}(y^k - \bar{Y})^2 + N(\bar{Y} - C)^2\end{aligned}$$

where the last equality follows by observing that the cross product term sums to zero.

Table A.1 provides various statistics for the estimators represented in Figure A.1. Observe that the estimator represented in light gray had small variance and high bias, the estimator represented in dark gray had high variance and low bias and that in black had both small variance and small bias. Moreover the RMSE of the estimator in black was the smallest. On this basis we would choose the estimator corresponding to the black line, consistent with the conclusions based on inspecting the Figure directly.

Estimator	RMSE	MSE	Variance	Bias^2
Black	1.59	2.54	1.17	1.37
Dark Gray	2.09	4.37	4.33	0.04
Light Gray	5.47	29.93	1.58	28.35

Table A.1: Characteristics of estimators in Figure A.1

RMSE for function estimation

The RMSE can also be used in another way. Suppose $\hat{v}(s)$ is an estimator of a function $v(s)$ defined over a finite state space S. Then we define $RMSE(\hat{\mathbf{v}})$ by

$$RMSE(\hat{\mathbf{v}}) := \sqrt{\left(\frac{1}{|S|}\sum_{s\in S}(\hat{v}(s) - v(s))^2\right)} \tag{A.2}$$

In studies of simulation based algorithms, such as in Section 1.7, we will obtain M replicates $\hat{v}^1(s), \ldots, \hat{v}^M(s)$ by varying the random seed of an algorithm. We can then summarize the performance of this algorithm by computing summary statistics such as means, standard deviations and ranges of $RMSE(\hat{\mathbf{v}}^1), \ldots, RMSE(\hat{\mathbf{v}}^M)$.

From a practical perspective, we recommend choosing estimators on the basis of graphical summaries and RMSE. Of course when our objective is to find good policies, precise estimation of values may be of secondary importance.

Appendix B

Stochastic Approximation

We provide a brief discussion of stochastic approximation, an approach that underlies temporal differencing and Q-learning methods described in this chapter and fundamental to reinforcement learning. Stochastic approximation originates with the seminal Robbins and Monro [1951] paper which developed an on-line algorithm for finding the root of a function based on noisy observations of the function value at selected points. It has become especially important for analyzing simulation data because it avoids storing large amounts of data.

Suppose we wish to find a value x^* for which $f(x^*) = 0$ based on a sequence of samples $(y^n : n = 1, 2, \ldots)$ from a distribution with conditional mean $f(x) = E[Y|X = x]$. For example, the sequence y^n may be generated by $y^n = f(x^n) + \varepsilon_n$ where ε_n denotes a random error term with mean 0 and the sequence $(x^n : n = 1, 2, \ldots)$ is chosen in some way.

The Robbins-Monro procedure generates a sequence $(x^n : n = 1, 2, \ldots)$ that represents both estimates of x^* and sample points using a recursion of the form

$$x^{n+1} = x^n - \tau_n y^n \tag{B.1}$$

where y^n denotes an observation from a distribution with conditional mean $E[Y|X = x^n]$ and the *step-size* parameter τ_n, represents a sequence of non-negative constants converging to 0.

The intuition underlying (B.1) (see Figure B.1) is that when y^n is greater than 0, x^{n+1} will be *corrected* downward from x^n and conversely when y^n is less than 0, x^{n+1} will be corrected upward from x^n. Moreover, when y^n is close to zero the correction will be smaller as will be the case for large n.

This recursion may be viewed as either an application of stochastic gradient descent (Appendix **??** or a stochastic generalization of Newton's method. When $f(x^n)$ is observable (and differentiable), Newton's method finds its zero by iterating

$$x^{n+1} = x^n - (f'(x^n))^{-1} f(x^n) \tag{B.2}$$

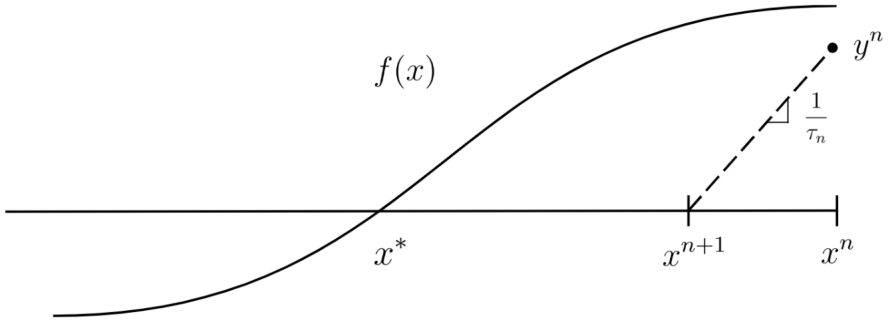


Figure B.1: Geometric representation of Robbins-Monro recursion for solving $f(x) = 0$. Since $y^n/(x^n - x^{n+1}) = 1/\tau^n$, $x^{n+1} = x^n - \tau_n y^n$.

Equation (B.1) replaces the reciprocal of the derivative by the constant τ_n and $f(x^n)$ by an observation sampled from a distribution with mean $f(x^n)$.

The following theorem (stated without proof) describes the limiting behavior of the sequence of iterates generated by (B.1).

Theorem B.1. Suppose $f(x) = E[Y|X = x]$,

1. $f(x)$ is monotone,
2. $E[(Y - f(X))^2|X = x] < M < \infty$ for some constant M, and
3. $|f(x)| \leq d|x| + c$ for $0 \leq c < \infty$ and $0 \leq c \leq \infty$, and
4. the sequence $\{\tau_n : n = 1, 2, \ldots\}$ satisfies

$$\tau_n \geq 0, \quad \sum_{n=1}^{\infty} \tau_n = \infty \quad \text{and} \quad \sum_{n=1}^{\infty} \tau_n^2 < \infty. \tag{B.3}$$

Then for any x^1, the sequence

$$x^{n+1} = x^n - \tau_n y^n$$

converges almost surely to x^* where $f(x^*) = 0$.

Some comments about this theorem follow:

1. The theorem as stated provides a stronger version of the result in Robbins and

Monro [1951] under slightly different conditions on $f(x)$. That is, it establishes almost sure convergence as opposed to L^2 convergence. The key conditions on the step-size in (B.3) remain the same.

2. Proofs of this theorem and its variants are beyond the scope of this book. They use subtle arguments and advanced probabilistic concepts mathematics. See the Section B for references.

3. Much of the literature refers to (B.3) as "the usual conditions on τ_n".

4. The conditions on τ_n in Theorem B.1 are satisfied when $\tau_n = k/n^b$ for $b \in (.5, 1]$ or $\tau_n = \frac{c}{d+n}$ where c and d are positive constants. Note the choice $\tau_n = 1/n$ satisfies these conditions.

5. Note that in addition to specifying the correction term to the current estimate of the solution of $f(x) = 0$, the Robbins-Monro recursion specifies at which value of x^n to generate a new observation.

6. This theorem focuses on a univariate method; a vector version also holds.

7. Example **??** below shows that in agreement with theory, when $\sum_{n=1}^{\infty} \tau_n < \infty$, the step-sizes converge to 0 too quickly leading to convergence to an incorrect value. Moreover, when $\sum_{n=1}^{\infty} \tau_n^2 = \infty$, variability is not damped out so that iterates oscillate rapidly. Finally when the conditions on the sums of τ_n and τ_n^2 are satisfied, the iterates converge.

8. Equation (B.1) can be rewritten as

$$x^{n+1} = (1 - \tau_n)x^n + \tau_n(x^n - y^n)$$

This representation writes x^{n+1} as a weighted average of the previous estimate x^n and the new observation y^n and is referred to as *exponential smoothing* in the forecasting literature.

Example B.1. Impact of τ_n on stochastic approximation estimates. This example shows how the conditions in (B.3) impact convergence of the stochastic approximation iterates. We consider the problem of finding the fifth root of 2. To put it in the stochastic approximation framework, this is equivalent to solving $f(x) = x^{1/5} - 2 = 0$ based on realizations of $f(x) + \varepsilon$ where ε is normally distributed with mean 0 and standard deviation 1. We consider three different choices for τ_n that illustrate cases when (B.3) hold and are violated in two different ways. We use $50,000$ iterates, common random number sequences and choose $x_0 = 20$ for each case. Of course $f(32) = 0$ so that $x^* = 32$.
Clearly conditions 1 and 3 of Theorem B.1 hold and since $var(\varepsilon) = 1$, (2) holds. Our choice of $\tau_n = .9n^{-.5}$ corresponds to the case in which both summation condi-

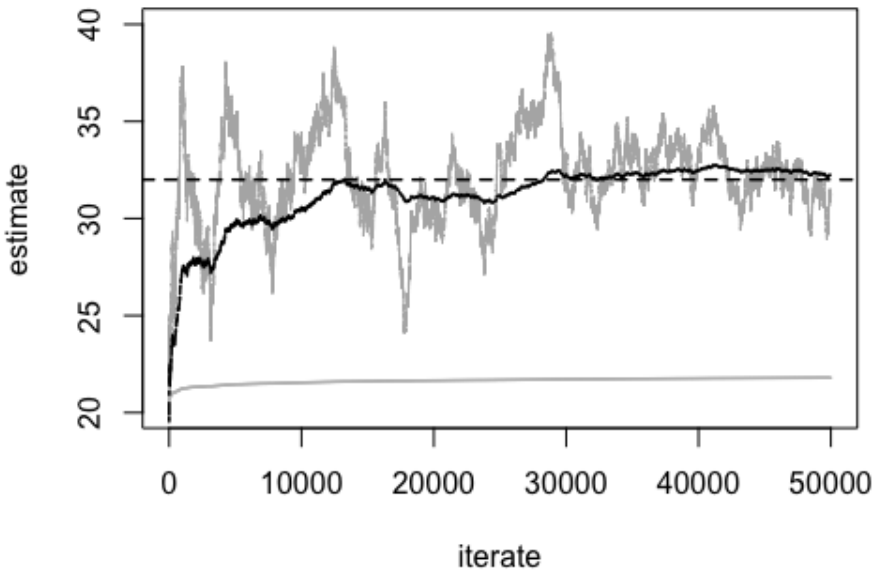


Figure B.2: Graphical results for Example B.1 showing three different approximations for finding the zero of $f(x) = x^{1/5} - 2$ based on realizations of $f(x) + \varepsilon$ where ε is normally distributed with mean 0 and standard deviation 1. The dashed line gives the solution $x^* = 32$, the dark grey line the iterates when $\tau_n = n^{-0.25}$, the light grey near the bottom of the figure corresponds to $\tau_n = n^{-1.0000001}$ and the black line to $\tau_n = .9n^{-0.5}$.

tions in (B.3) are satisfied, choosing $\tau_n = n^{-1.0000001}$ corresponds to $\sum_{n=1}^{\infty} \tau_n < \infty$ and $\sum_{n=1}^{\infty} \tau_n^2 < \infty$ and choosing $\tau_n = n^{-0.25}$ corresponds to $\sum_{n=1}^{\infty} \tau_n = \infty$ and $\sum_{n=1}^{\infty} \tau_n^2 = \infty$. Note that $\sum_{n=1}^{\infty} \tau_n = \infty$ and $\sum_{n=1}^{\infty} \tau_n^2 < \infty$ is impossible. (As an exercise, you might verify that these sequences satisfy the indicated conditions.) Figure B.2 shows the result of using these three step-size sequences. The dashed line corresponds to the solution $x^* = 32$. The black line, which corresponds to $\tau_n = .9n^{-.5}$ shows convergence of the sequence of iterates. The dark gray line, corresponding to $\tau_n = n^{-1.0000001}$, shows that the iterates converge, but to the wrong value. Finally the dark grey line which corresponds to $\tau_n = n^{-0.25}$ shows that iterates continue to oscillate around the target value $x^* = 32$.
From this analysis we conclude that different violations of the assumptions on τ_n have different impact on the sequence of iterates x^n.

We now illustrate the use of (B.1) with three further examples.

Example B.2. Using stochastic approximation to find the zero of a regression equation.
Suppose we seek to solve $\beta_0 + \beta_1 x = 0$, when β_0 and β_1 is unknown, based on sequentially observing data generated from a distribution with

$$E[Y|X = x^n] = \beta_0 + \beta_1 x^n.$$

When errors are additive, we can represent the generated sequence of observations by

$$y^n = \beta_0 + \beta_1 x^n + \varepsilon_n$$

where the random disturbance term ε_n has mean zero. However the more general problem definition based on conditional expectations allows us to solve more general this problem with general distributions such as arise in Poisson or logistic regression models.
Thus we seek an x^* such that $E[Y|X = x^*] = \beta_0 + \beta_1 x^* = 0$. In a regression setting, we would be concerned with finding estimates of β_0 and β_1. Here we instead are seeking a root of an unobservable linear function and do not care about the parameter estimates. Figure **??** shows iterates of two estimates of the zero of $y = -4 + 8x$, based on observations of $y^n = -4 + 8x^n + \varepsilon_n$ where ε_n is generated from a uniform distribution on $[-3, 3]$. Both estimates start at $x_1 = 1$, use $\tau_n = n^{-5}$ and $\tau_n = n^{-1}$. Figure B.3 shows that the estimates stabilize quickly and oscillate around the root $x^* = 0.5$. Moreover, that obtained using $\tau_n = 1/n$, is most stable and has smallest variance and MSE.

Example B.3. Using stochastic approximation to find a mean
We now consider the problem of estimating the expected value μ of a random variable Z. This can be regarding as finding the root of the equation $E[Z-x] = 0$. In the case of a normal distribution, this reduces to the model in Example B.2 with $\beta_0 = -\mu$ and $\beta_1 = 1$. Unlike in the previous example, β_1 is known. In other words, we observe

$$y^n = \mu - x^n + \varepsilon_n$$

and use (B.1) to obtain estimates of the mean.
Suppose instead of observing y^n as above, we observe a sequence of observations z^n from a distribution with mean $E[Z]$. This is equivalent to finding an x for which solving $x - E[Z] = 0$. In this case, the Robbins-Monro recursion (B.1) becomes

$$x^{n+1} = x^n - \tau_n(x^n - z^n) = x^n + \tau_n(z^n - x^n). \tag{B.4}$$

As an aside, we relate this to a well-known online formula for estimating the sample mean. Suppose we observe $\{z^1, z^2, \ldots, z^n\}$ and estimate $E[Z]$ by the sample mean which we denote by $x^{n+1} = \frac{1}{n}\sum_{i=1}^{n} z^i$. Then by some simple

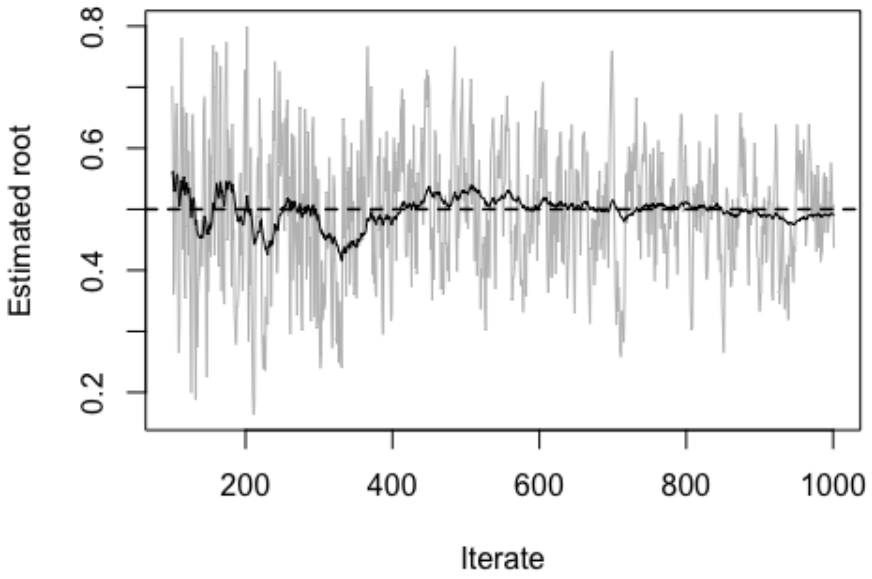


Figure B.3: Graphical results from Example B.2 showing estimates of the root of $E[Y|X = x] = -4 + 8x$, based on observations generated by $y^n = -4 + 8x^n + \varepsilon_n$ where ε_n is simulated from a uniform distribution on $[-3, 3]$. The grey line corresponds to estimates based on $\tau_n = n^{-.5}$ and the black line to $\tau_n = n^{-1}$. The dashed line represents the solution of $-4 + 8x = 0$ which equals 0.5.

algebra we obtain

$$\begin{aligned} x^{n+1} &= \frac{1}{n}[(n-1)x^n + z^n] \\ &= \frac{(n-1)x^n}{n} + \frac{z^n}{n} + \frac{x^n}{n} - \frac{x^{n-1}}{n} \\ &= x^n + \frac{1}{n}(z^n - x^n) \end{aligned} \tag{B.5}$$

Thus we see that the recursion (B.5) is a special case of the more general (B.4) with $x_1 = 0$ and $\tau_n = \frac{1}{n}$.

We now apply (B.4) for finding the mean of an exponential distribution with parameter $p = .1$. We again set $\tau_n = n^{-b}$ for $b = 0.5, 0.75, 1.0$, use a common seed and generate 5000 iterates starting with $x^1 = 1$. Figure (B.4) shows that the estimate using $\tau_n = n^{-1}$ is least variable and best estimates the true mean.

Moreover this conclusion appears to be valid even when observations are auto-correlated.

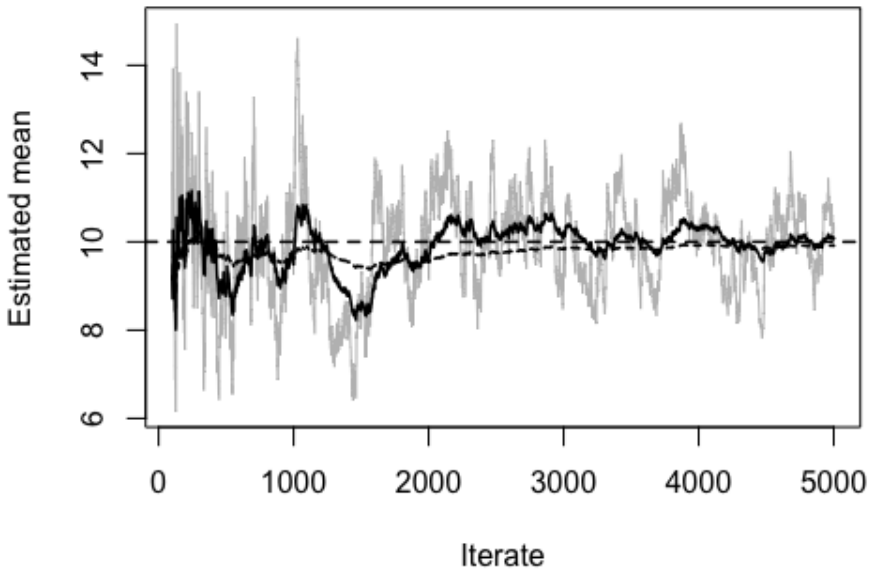


Figure B.4: Graphical results from Example B.3 showing stochastic approximation estimates using (B.4) based on observations generated by an exponential distribution with mean 10. The grey line corresponds to $\tau_n = n^{-0.5}$, the solid black line to $\tau_n = n^{-0.75}$ and the dashed black line to $\tau_n = n^{-1}$.

The following novel example shows that other choices for τ_n may produce better estimates.

Example B.4. Using stochastic approximation to find a percentile Since the pth, $0 \leq p \leq 100$ percentile of the distribution of a random variable Y can be represented as the solution x^* of $E[1_{\{Y \leq x\}}] = p/100$ we can use (B.1) to find this percentile. As an example, suppose we seek the 75th percentile of a Cauchy distribution[a]. In this case, (B.1) becomes

$$x^{n+1} = x^n - \tau_n(I_{\{y^n \leq x^n\}} - 0.75).$$

Note that the quantity $(I_{\{y^n \leq x^n\}} - 0.75)$ only takes on the values 0.25 and -0.75. We generate three sets of estimates of this percentile using $\tau_n = n^{-b}$ for $b = 0.5, 0.75, 1$. We use a common seed for the random number generator, start at

$x^1 = 2$ and generate $10,000$ iterates. Results appear in Figure B.5.The grey line corresponds to $\tau_n = n^{-.5}$, the black line to $\tau_n = n^{-0.75}$ and the dotted line to $\tau_n = n^{-1}$. The estimates when $\tau_n = n^{-.5}$ are more variable than those of $\tau_n = n^{-0.75}$ although each varies around the 75th percentile which equals 1. Observe that using $\tau_n = n^{-1}$ leads to quick convergence but to the wrong value. This is because it damps the values too quickly so the estimate is highly influenced by initial realizations.

[a]Recall that the Cauchy distribution has infinite variance.

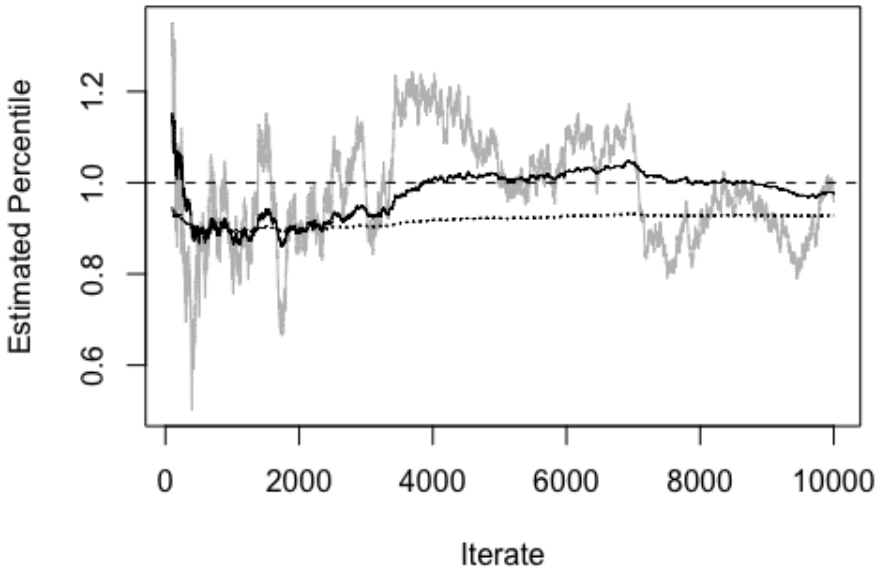


Figure B.5: Graphical results from Example B.4 showing three stochastic approximation estimates of 75th percentile of a Cauchy distribution. The grey line corresponds to $\tau_n = n^{-.5}$, the black line to $\tau_n = n^{-0.75}$ and the dotted line to $\tau_n = n^{-1}$. The horizontal line represents the true value of the 75th percentile.

We conclude from these examples that stochastic approximation provides a highly flexible approach for solving a range of interesting problems. However care must be taken when choosing step-size sequences.

Bibliographic Remarks

This chapter has attempted to provide a user-friendly introduction to simulation based methods for Markov decision processes. It is not intended to be comprehensive or even up to date. Gosavi [2015] and Szepesvari [2010] provide nice introductions and overviews of simulation methods for Markov decision processes at a level that should be accessible to all readers of this book.

Xiao et al. address data generation issues from the perspective of complexity theory and motivate our discussion in section 1.1.1.

We hope that after reading this chapter you can jump into some of the more comprehensive work especially Sutton and Barto [2018] which takes a computer science perspective on this material. Bertseksas [2012] and Meyn [2022] provide a deeper and more rigorous exposition from a control theory perspective. Moreover the reader can find numerous sets of teaching notes and lectures online.

Temporal difference learning, which presumably originates with early work of Sutton (see Sutton and Barto [2018]), is based on applying stochastic approximation to policy evaluation. They refer to this as a *prediction* problem.

Solving Markov decision processes in a model-free online environment originates with Watkins PhD thesis and the seminal paper Watkins and Dayan [1992] on Q-Learning. This paper provides a proof of that Q-learning converges with probability one in a discounted model when results are represented by a look-up table and all state-action pairs are sampled infinitely often.

The material on average reward temporal difference and Q-learning follows Tsitsiklis and Roy [1999], Mahadevan [1996] and p.548-551 in Bertseksas [2012]. TD(γ) is discussed in Sutton and Barto [2018] and many other places and originates in some highly original papers of Sutton referred to therein.

Our discussion of online policy iteration is motivated by Buşoniu et al. [2010] who describe a more general approach that includes function approximation.

Our discussion of stochastic approximation in the appendix dates back to the original papers of Robbins and Monro [1951] and Blum [1954]. More modern approaches to this work appear in Bertseksas [2012], Powell [2007] and Meyn [2022]. Our example using stochastic approximation to estimate percentiles is motivated by Ribes et al. [2019].

Exercises

1. For the grid-world model in Example 1.2 conduct a study of determining parameters in TD(γ) by varying:

   (a) the number of iterates,

   (b) the model parameters γ and τ_n,

   (c) the decision rule analyzed, and

   (d) the simulation seed.

   Use RMSE as the basis for comparison. Summarize your findings verbally and graphically.

2. Consider the deterministic version of grid-world as in Example 1.9 except assume that the robot can move "up","down", "right" and "left". Apply q-learning and Sarsa to find good policies using softmax and ϵ-greedy exploration. Compare these methods on the basis of the mean and standard deviation of the total reward per replicate.

3. State Q-learning and simulation based policy iteration algorithms for an episodic model.

   (a) Apply them to the grid-world example.

   (b) Summarize your conclusions verbally and graphically.

4. **Cliffwalking**[1] In this problem on a 4×12 grid, a robot must traverse the grid by moving from the "origin" in cell (1,1) in the lower left hand corner of the grid to the "destination" in cell (1,12) in the lower right hand corner of the grid without falling off the cliff between the origin and destination in cells (1,2) to (1,11). The robot incurs a cost of 1 unit for each step it takes and a penalty of 100 if it falls off the cliff. If it falls off the cliff it returns to the origin and starts again.

   The robot moves deterministically and can choose to move "up", "down", "left" or "right" in every cell. If it tries to move in a direction which is not possible, it incurs a cost of one unit and remains in that cell.

[1] This colorful example was proposed by Sutton and Barto [2018]. It has a similar structure to our grid-world example of a coffee delivering robot.

The goal is to learn an "optimal path" through this grid. Assume the robot does not "know" the layout and must learn it by ϵ-greedy exploration.

(a) Formulate this problem as an episodic model in which an episode starts with the robot at the origin and concludes when it reaches its destination.

(b) Find (and state) an optimal policy using Q-learning and Sarsa with a learning rate $\tau = 0.1$, $\epsilon = 0.1$ and a discount rate of 1.

(c) Accumulate the total reward per episode for Q-learning and Sarsa for the first 500 episodes of each. Compare them graphically.

(d) Investigate the impact of the τ and ϵ on algorithmic performance.

5. Show that using TD(γ) for policy evaluation is equivalent to temporal differencing when $\gamma = 0$ and Monte Carlo estimation when $\gamma = 1$. Note in the latter case, some care has to go into choosing the trace, specifying τ_n and distinguishing the various forms of Monte Carlo estimation.

6. In the context of the queuing control model in Section 1.7, investigate the convergence of variants of Algorithms 1.12 and 1.13 that smooth the **value** function prior to exact derivation using (1.47) or estimating the q-functionas in step 2(b) of Algorithm 1.13. Compare the effects of smoothing both the value function and the q-function to smoothing only the value function.

7. **A large queuing service rate control model** Use Q-learning and the policy iteration algorithms of Section 1.6 to analyze the larger version of the queuing service rate control model model previously solved with (deterministic) policy iteration in Section **??**. In it $N = 5000$, $A_s = \{0.2, 0.3, \ldots, 0.7\}$, $b = 0.2$, $m(a_k) = 2k^3$, $f(s) = s^2$ and the discount rate equals 0.4. We chose such a low discount rate to obtain an interesting optimal policy.

8. **Inventory control with hidden Markov demand** Consider the following inventory model. At the end of each day, the inventory manager can order j units from a supplier that arrives the next morning at a cost of c per item plus a fixed ordering cost of k per order. When the demand d on any day exceeds number of units on hand, the excess demand is unfilled and lost. If demand is less than or equal the number of units received, the system receives a revenue r for each unit sold. Unsold inventory is carried over to the next day at a cost of h per unit.

   Suppose that the demand follows a binomial distribution with parameters D and p and moreover p takes on the values p_L and p_H according to a Markov chain with transition probability matrix

$$P = \begin{bmatrix} 0.2 & 0.8 \\ 0.6 & 0.4 \end{bmatrix}$$

where row 1 corresponds to p_L. Suppose that $c = 10, K = 5, r = 20, h = 3, p_H = .5, p_L = 0.2$ and $D = 20$.

(a) Develop a simulation model of this system and use it to evaluate the policy that orders 5 units every day.

(b) Find a good policy for the infinite horizon discounted model with $\lambda = 0.95$.

9. Repeat the analysis in Example 1.6 using the exact value of g as computed in in the example. Does the estimate of $h(s_2)$ improve?

10. Use temporal differencing, truncated discounted rewards and geometric sampling to estimate the infinite horizon discounted reward for the following policies for the model in Exercise 1 of Chapter 2.

(a) The deterministic stationary policy that uses $d(s_i) = a_{i,1}$ in state s_i for $i = 1, 2, 3$.

(b) The randomized stationary policy that uses action $a_{i,1}$ with probability $e^{-0.5i}$ and action $a_{i,2}$ with probability $1 - e^{-0.5i}$ in state $s_i$for $i = 1, 2, 3$.

11. Develop a TD(γ) algorithm for an average reward model. Apply it to evaluate the gain and bias of both policies in Exercise 10.

12. Use the recursion (B.1) to find a root of the function $f(x) = \frac{1}{1+e^{-0.5x}} - .5$ based on observing $f(x) + \varepsilon$ where ε is normally distributed with mean 0 and standard deviation 0.1. Investigate the variability and accuracy of estimates to the choice of a and b when $\tau_n = an^{-b}$ and the number of observations.

13. Use the recursion in the (B.4) to estimate a mean when the data is generated from a first order auto-correlated process with mean 2. That is the data is generated according to $y^n = 2+u_n$ where $u_n = .9u_n-1+\varepsilon_n$ when $u_1 = 0$ and ε_n is generated from a normal distribution with mean 0 and standard deviation 1. Investigate the variability and accuracy of estimates to the choice of a and b when $\tau_n = an^{-b}$ and the total number of observations.

14. Consider the model in Exercise 1 of Chapter 2.

(a) Use Q-learning and approximate policy iteration to find an optimal policy for a discounted version of the model. Choose $\lambda = 0.9$ and $\lambda = 0.999$. Comment on the effect of λ on the execution of the algorithm.

(b) Use an appropriate version of Q-learning to find an optimal policy for the average reward version of this model.

15. Consider the admission control queuing model model of Section **??** with $R = 20, h(j) = j, w = 0.3$ and $b = 0.2$. Truncate the state space at $M = 50$.

(a) Find the optimal policy using policy iteration for a discounted version of this model with $\lambda = 0.95$.

(b) Use Q-learning and an approximate policy iteration to find a good policy for this model.

(c) Find the optimal policy using policy iteration for an average reward version of this model with $\lambda = 0.95$.

(d) Investigate the impact of truncation on the optimal policy and it's gain by varying M.

(e) Use appropriate forms of Q-learning and approximate policy iteration to find good policies for this model.

(Check if these parameters produce good results)

16. Develop a simulation based policy iteration algorithm for an average reward model and use it to solve the queuing service rate control model.

17. Carry out a detailed study of step size choice (τ_n) and sample size on a discounted version of Exercise 1 from Chapter 2. Compare results on the basis of MSE and visual inspection of the value functions.

18. Develop a Q-learning algorithm for a finite horizon model and apply it to solve the revenue management problem in Section **??**.

19. Apply TD(γ) to the estimate the infinite horizon expected discounted reward for both policies in the model in Exercise 1. Use MSE and visual inspection to select appropriate choices for model parameters.

Bibliography

D.P. Bertseksas. *Dynamic Programming and Optimal Control, Volume 2, 4th Edition*. Athena Scientific, 2012.

J.E. Blum. Approximation methods which converge with probability one. *Ann. Math. Stat.*, 25:382–386, 1954.

L. Buşoniu, D.Ernst, B. DeSchutter, and R. Babuska. Online lest-squares policy iteration for reinforcement learning control. *Proc. 2010 Amer. Cont. Conf*, pages 486–491, 2010.

A. Gosavi. *Simulation-Based Optimization - Parametric Optimization Techniques and Reinforcement Learning* . Springer, 2015.

S. Mahadevan. Average reward reinforcement learning:foundations, algorithms and empirical results. *Machine Learning*, 22:159–195, 1996.

S. Meyn. *Control Systems and Reinforcement Learning* . Cambridge University Press, 2022.

W. Powell. *Approximate Dynamic Programming*. J.Wiley and Sons, 2007.

R. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2021. URL `https://www.R-project.org/`.

A. Ribes, T. Terraz, B. Ioos, Y. Fournier, and B. Raffin. Large scale in transit computation of quantiles for ensemble runs. *hal-02016828v1*, 2019.

H. Robbins and S. Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22:400–407, 1951.

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An introduction, second edition*. MIT Press, Cambridge, MA, 2018.

C. Szepesvari. *Algorithms for Reinforcement Learning* . Morgan and Claypool, 2010.

J. Tsitsiklis and B. Van Roy. Average cost temporal-difference learning. *Automatica*, 35:1799–1808, 1999.

C. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.