A Tutorial for Reinforcement Learning

Abhijit Gosavi, PhD
Professor
Department of Engineering Management and Systems Engineering
Missouri University of Science and Technology
210 Engineering Management, Rolla, MO 65409
Email: gosavia@mst.edu

April 12, 2025

If you find this tutorial or the codes below useful, please do cite it!

I have also written a book on this topic. This tutorial cannot be used as a substitute for any book. The book is entitled: *Simulation-Based Optimization: Parametric Optimization Techniques and Reinforcement Learning*.

`https://simoptim.com/`

Contents

Chapter 1

Basics

1.1 Introduction

This tutorial is designed for readers seeking an introduction to reinforcement learning (RL), which is primarily used to solve Markov decision processes (MDPs) and their variants. In essence, problems suitable for RL should conform to the structure of an MDP or a related framework.

The primary objective is to present RL concepts intuitively. We begin by exploring policies and Markov chains, examining the limiting behavior of systems to build a solid foundation in MDPs through exhaustive enumeration. Next, we introduce dynamic programming algorithms for solving MDPs and their common variant, semi-Markov decision processes (SMDPs), before gradually transitioning to reinforcement learning, where transition probabilities are no longer required.

Throughout the tutorial, we provide numerical examples to illustrate various algorithms in practice. The tutorial concludes with the author's final reflections on the topic.

Limitations of this tutorial: This tutorial is not aimed at providing a comprehensive coverage of Markov decision processes or reinforcement learning. In particular, the TD($\overline{\lambda}$) framework of Sutton is not discussed nor is there any discussion on function approximation. These topics have been covered in textbooks named in the bibliography.

1.2 MDPs

The framework of the MDP has the following elements: (1) state of the system, (2) actions, (3) transition probabilities, (4) transition rewards, (5) a policy, and (6) a performance metric. We assume that the system is modeled by a so-called abstract stochastic process called the *Markov chain*. The basics of the Markov chain model and the decision-making problem that accompanies it are discussed above.

State: The *state* of a system is a parameter or a set of parameters that can be used to describe a system. For example, the geographical coordinates of a robot can be used to describe its state. A system whose state changes with time is called a *dynamic* system. Then, it is not hard to see why a moving robot produces a dynamic system.

Another example of a dynamic system is the queue that forms at the supermarket checkout counter. Imagine that the state of the queuing system is defined by the number of people in the queue. Then, it becomes evident that the system's state fluctuates with time and makes the queue a simple example of a dynamic system.

It is important to understand that the transition (i.e., moving) from one state to another in an MDP is governed by randomness. For example, consider a queue in which

there is one server with a single waiting line. Here, the system's state x, defined by the number of people in the queue, transitions to $x+1$ with some probability when a new customer arrives, and to $x-1$ with the remaining probability when a customer completes service and departs from the system.

In many real-world systems, numerous aspects of the system can be controlled via deliberately made decisions. These decisions will be referred to as *actions*.

Actions: In the case of the robot, its movement can be controlled in discrete steps; we would be interested in controlling the robot's motion in an *optimal* manner. For the robot, the actions will be: go North, go South, go East, or go West. These four options are also *controls* in control theory.

For the queuing system discussed above, an action would involve managing the congestion in the system: For instance, when the number of customers waiting in the line exceeds a prefixed number, also called threshold, (say 10), a new counter is opened, and the remaining customers are diverted to it. Hence, the two available actions for this system are: (1) Open a new counter (2) Do not open a new counter.

Transition Probability: Assume that action a is selected in state i. Let the next state be j. Let $p(i,a,j)$ denote the probability of going from state i to state j under the influence of action a in one step. This probability is called the *transition probability*. Then, if an MDP has 3 states and 2 actions, there will be 9 transition probabilities per action.

Immediate Rewards: Typically, in the model under consideration here, the system receives an immediate reward, which could be positive or negative, when it transitions from one state to another. This reward is denoted here by $r(i,a,j)$.

Policy: The so-called policy defines the action to be chosen in every state visited by the system. It is important to note that not all states need a decision. Such states, where no decisions have to be made, are excluded when setting up the transition probability, $p(i,a,j)$, and its associated transition reward, $r(i,a,j)$ in the model. States in which decisions are to be made, i.e., actions are to be chosen, are called *decision-making* states. Only decision-making states are included in the model, although in many models. In this tutorial, by states, we will only mean decision-making states. A policy will typically be denoted by $\hat{\mu}$ or $\hat{d}$. The $\hat{}$ above the symbol is meant to indicate that the policy is a structured collection of numbers, although it is not a vector. To be more specific, it is an *ordered* list of numbers in which there is one number for each state. The first entry is for the first state, the second for the second state, and so on. The entry for each state specifies the action to be chosen in that state. For example, consider a system with 3 states, each allowing 2 possible action: A policy $\hat{\mu} = (2,1,2)$ for this system indicates that action 2 is chosen in the first state, action 1 in the second state, and action 2 in the third state.

Performance Metric: Each policy is associated with a so-called performance metric that quantifies its effectiveness. The overall goal in MDP analysis is to identify the policy with the best performance, i.e., the one that optimizes the chosen performance metric.

We first consider a popular metric called the average reward. Thereafter, we discuss another widely used metric called discounted reward. In both cases, we assume that the system is operated over an infinite time horizon, meaning it is run for a very long time.

Time of transition: We will assume for the standard MDP that the time of transition is constant and without loss of generality it is set to 1. For the semi-MDP (SMDP), which is covered later in this chapter, this quantity is not a constant but rather a variable, as discussed there.

Average Reward: If a policy named $\hat{\mu}$ is to be followed, then $\mu(i)$ denotes the action selected by this policy for state i. Every time, the system transitions from one state to another under the influence of an action selected by the policy, the transition is technically associated to the Markov chain of the associated policy. It is important to note that as a transition may occur from a state to itself.

Define x_s as the state occupied by the system before the sth transition with x_1 being the initial states. Then, the average reward of the policy $\hat{\mu}$ starting at state i is defined as:

$$\rho_{\hat{\mu}}(i) = \lim_{k\to\infty} \frac{\mathsf{E}\left[\sum_{s=1}^{k} r(x_s, \mu(x_s), x_{s+1}) | x_1 = i\right]}{k}. \tag{1.1}$$

This average reward thus essentially denotes the sum of the total immediate rewards earned divided by the number of transitions, calculated over a very long time horizon (i.e., when k assumes a large value). In the above, the starting state is i and $\mu(x_s)$ denotes the action selected in state x_s. Also note that $\mathsf{E}[.]$ denotes the average value of the quantity inside the square brackets: [.].

It is not hard to show that when the underlying Markov chains in the system for the policy $\hat{\mu}$ satisfies a certain condition related to the so-called *regularity* of the Markov chain for that policy, the limit in Eqn. (1.1) is such that its value is the *same* for any value of x_1. In many real-world problems, this condition often satisfied. Then,

$$\rho_{\hat{\mu}}(i) = \rho_{\hat{\mu}} \text{ for any value of } i.$$

Under this condition, we can hence replace $\rho_{\hat{\mu}}(i)$ by $\rho_{\hat{\mu}}$, making this analysis easier.

The objective of the average-reward MDP under the regularity condition is to find the policy $\hat{\mu}^*$ that *maximizes* the **average reward**, i.e., the performance metric, of the policy.

Discounted Reward: Another popular performance metric, commonly used in the literature, is discounted reward. The following quantity is mathematically defined as the **discounted reward** of a policy $\hat{\mu}$ in which x_s denotes the state of the system before the sth transition of the Markov chain of the policy $\hat{\mu}$:

$$J_{\hat{\mu}}(i) = \lim_{k\to\infty} \mathsf{E}\left[\sum_{s=1}^{k} \lambda^{s-1} r(x_s, \mu(x_s), x_{s+1}) \middle| x_1 = i\right]. \tag{1.2}$$

In the above, λ denotes the so-called *discount factor*; λ, is less than 1 but greater than 0. In this setting, λ is essentially discounts money's value in the future, which is a commonly used concept in engineering economics. The following is a more intuitive interpretation of Eqn. (1.2):

$$J_{\hat{\mu}}(i) = \mathsf{E}[r(x_1, \mu(x_1), x_2) + \lambda r(x_2, \mu(x_2), x_3) + \lambda^2 r(x_3, \mu(x_3), x_4) + \cdots] \tag{1.3}$$

The discounted reward, thus, essentially measures the **present value** of the sum of the rewards earned in the future over an infinite time horizon in which future rewards are progressively diminished.

The discount factor is measured as follows from market conditions:

$$\lambda = \left(\frac{1}{1+\gamma}\right)^1, \tag{1.4}$$

where γ equals the rate of interest in the market. Here, the rate of interest is expressed as a fraction between 0 and 1 and not in percentage terms, commonly used in engineering economics literature. When $\gamma > 0$, Eqn. (1.4) implies that $0 < \lambda < 1$. It is also worth pointing out that in any MDP, $1/(1+\gamma)$ raised to the power 1 equals the discount factor, because in an MDP, the time duration of each transition is a constant equalling 1 and this time duration is also the power in the term in the right-hand side of Eqn. (1.4). Thus, this discounting mechanism effectively captures an important notion of the *time value of money* from economics within the MDP framework.

The objective of the discounted-reward MDP is to determine the policy that maximizes the performance metric (discounted reward) of the policy *starting from every state.*

Note that for average reward problems, the immediate reward in our algorithms can be earned during the entire duration of the transition. However, for discounted reward problems, we will assume in this tutorial that the immediate reward is earned immediately after the transition starts.

The MDP can be solved using the classical method of dynamic programming (DP). However, DP needs all the transition probabilities (i.e., the $p(i, a, j)$ terms) and the transition rewards (i.e., the $r(i, a, j)$ terms) of the MDP.

Two Curses of Dynamic Programming: Dynamic programming suffers from two major challenges:

- For systems with a large number of input random variables, it is often difficult to compute the exact *values* of the transition probabilities. This is referred to as the *curse of modeling.*
- For large-scale systems with millions of states, it is impossible to *store* the transition probabilities. This is referred to as the *curse of dimensionality.*

DP breaks down on problems that suffer from any one of these curses because it needs the values of all the transition probabilities.

Why Reinforcement Learning?

Reinforcement Learning (RL) offers a poweful alternative to dynamic programming. It can deliver *near-optimal* solutions to large and complex MDPs without the knowledge of the transition probabilities. In short, RL can make inroads into problems where dynamic programming fails because of one or more of the two above-named curses.

1.3 Toy Example

We discuss a simple toy example from [11]. Before considering this toy example, it is necessary to first understand the concept of regular Markov chains, steady-state probabilities of Markov chains, and the value function of a policy.

1.3.1 Regular Markov Chains

A **regular** Markov chain is defined as one whose TPM satisfies the following property: *There exists a finite positive value for n, call it n_*, such that for all $n \geq n_*$ and all i and j*:

$$P^{n_*}(i, j) > 0.$$

This implies that raising the TPM of a regular Markov chain to a positive finite power, one creates a matrix of which each element is strictly greater than zero. An example of such a Markov chain is:

$$\mathbf{P} = \begin{bmatrix} 0.7 & 0.0 & 0.3 \\ 0.4 & 0.6 & 0.0 \\ 0.2 & 0.7 & 0.1 \end{bmatrix}.$$

It is not difficult to verify that P can be raised to a suitable positive finite power to obtain a matrix in which each element is strictly greater than zero. An example of a Markov chain that is *not* regular is:

$$\mathbf{P} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}.$$

This is not regular because:

$$\mathbf{P}^n = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \text{ for odd } n \text{ and } \mathbf{P}^n = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \text{ for even } n.$$

1.3.2 Computing Steady-State Probabilities of a Policy

If one raises the TPM of a *regular* Markov chain to higher powers, the elements in any given column start *converging* to (that is, approaching) the same number. For example, consider the matrix $\mathbf{P}$.

$$\mathbf{P} = \begin{bmatrix} 0.7 & 0.3 \\ 0.4 & 0.6 \end{bmatrix}.$$

When raised to the second power, we obtain the following matrix:

$$\mathbf{P}^2 = \begin{bmatrix} 0.61 & 0.39 \\ 0.52 & 0.48 \end{bmatrix}.$$

When raised to the 3rd power, we obtain:

$$\mathbf{P}^3 = \begin{bmatrix} 0.5830 & 0.4170 \\ 0.5560 & 0.4440 \end{bmatrix}.$$

Note that the elements of any given *column* in $\mathbf{P}^n$ start approaching each other, as one increases the power n from 2 to 3 and so on. When $n = 7$, the matrix becomes:

$$\mathbf{P}^7 = \begin{bmatrix} 0.5715 & 0.4285 \\ 0.5713 & 0.4287 \end{bmatrix}.$$

In the matrix above, the elements of any column are very close to each other. In general, it can be proved that for a regular Markov chain, as the power n tends to infinity, for every $j, P^n(i,j)$ starts converging to a unique finite number. This implies that for every value of j:

$$\lim_{n\to\infty} P^n(i,j) \text{ exists and is unique.}$$

This unique probability is denoted by $\Pi(j)$ and is referred to as the **steady-state** or **limiting** probability of state j. In mathematical notation:

$$\Pi(j) \equiv \lim_{n\to\infty} P^n(i,j). \tag{1.5}$$

For regular Markov chains, the steady-state probability of each state can be determined from the transition probability matrix by solving the following set of linear

equations:

$$\begin{aligned} \sum_{i=1}^{|\mathcal{S}|} \Pi(i)P(i,j) &= \Pi(j) \text{ for every } j \in \mathcal{S} \\ \sum_{j=1}^{|\mathcal{S}|} \Pi(j) &= 1. \end{aligned} \tag{1.6}$$

Here, $\mathcal{S}$ denotes the set of states and $|\mathcal{S}|$ denotes the number of states. For instance, in a system with 4 states, $\mathcal{S} = \{1, 2, 3, 4\}$ and $|\mathcal{S}| = 4$.

1.3.3 Computing the Discounted Value Function of a Policy

For the discounted reward problem, the so-called value function of a policy, $\hat{\mu}$, is denoted by $J_{\hat{\mu}}$. It is a vector with one value for each state. The value function essentially represents the total discounted reward starting from a state and pursuing the policy, i.e., it represents the effectiveness of the policy when starting from a given state. The value function can be computed via solving the so-called Bellman equation *for a given policy*. This equation is defined as follows:

$$J_{\hat{\mu}}(i) = \bar{r}(i, \mu(i)) + \lambda \sum_{j=1}^{|\mathcal{S}|} p(i, \mu(i), j) J_{\hat{\mu}}(j). \tag{1.7}$$

We will shortly see that this equation is not difficult to solve for any given policy, and the solutions to all possible policies provided a simple way to determine the best policy (solution).

1.3.4 Details of Toy Example

Consider Fig. 1.1 that shows a simple MDP with two states and two actions allowed in each state. We will call this a toy example MDP, and it will be used to illustrate the main ideas in this tutorial. The caption below the figure explains the transition rewards and probabilities.

There are two states numbered 1 and 2 in an MDP, and two actions, which are also numbered 1 and 2, are allowed in each state. The TPM of action a is denoted by $\mathbf{P}_a$, while the TRM (Transition Reward Matrix) of action a is denoted by $\mathbf{R}_a$. The values of the transition probability matrices (TPM) associated with actions 1 and 2 are assumed in this toy example to be:

$$\mathbf{P}_1 = \begin{bmatrix} 0.7 & 0.3 \\ 0.4 & 0.6 \end{bmatrix} \text{ and } \mathbf{P}_2 = \begin{bmatrix} 0.9 & 0.1 \\ 0.2 & 0.8 \end{bmatrix}.$$

The TRM for actions 1 and 2 in the toy example are assumed to be:

$$\mathbf{R}_1 = \begin{bmatrix} 6 & -5 \\ 7 & 12 \end{bmatrix} \text{ and } \mathbf{R}_2 = \begin{bmatrix} 10 & 17 \\ -14 & 13 \end{bmatrix}.$$

As stated above, pictorially, the MDP is represented in Fig. 1.1, and it is instructive to understand the data via this figure.

In this MDP, there there are four possible policies. They are:

$$\hat{\mu}_1 = (1, 1), \hat{\mu}_2 = (1, 2), \hat{\mu}_3 = (2, 1), \text{ and } \hat{\mu}_4 = (2, 2).$$

The TPMs and TRMs of each of these policies can be constructed from the individual TPMs and TRMs of each action. We will now denote the TPM of policy $\hat{\mu}$ by $\mathbf{P}_\mu$ and the TRM of the policy μ by $\mathbf{R}_\mu$. The TPMs will be:

$$\begin{aligned}\mathbf{P}_{\hat{\mu}_1} &= \begin{bmatrix} 0.7 & 0.3 \\ 0.4 & 0.6 \end{bmatrix}; \mathbf{P}_{\hat{\mu}_2} = \begin{bmatrix} 0.7 & 0.3 \\ 0.2 & 0.8 \end{bmatrix}; \\ \mathbf{P}_{\hat{\mu}_3} &= \begin{bmatrix} 0.9 & 0.1 \\ 0.4 & 0.6 \end{bmatrix}; \mathbf{P}_{\hat{\mu}_4} = \begin{bmatrix} 0.9 & 0.1 \\ 0.2 & 0.8 \end{bmatrix}.\end{aligned}$$

The TRMs of each of the four policies are:

$$\begin{aligned}\mathbf{R}_{\hat{\mu}_1} &= \begin{bmatrix} 6 & -5 \\ 7 & 12 \end{bmatrix}; \mathbf{R}_{\hat{\mu}_2} = \begin{bmatrix} 6 & -5 \\ -14 & 13 \end{bmatrix}, \\ \mathbf{R}_{\hat{\mu}_3} &= \begin{bmatrix} 10 & 17 \\ 7 & 12 \end{bmatrix}; \mathbf{R}_{\hat{\mu}_4} = \begin{bmatrix} 10 & 17 \\ -14 & 13 \end{bmatrix}.\end{aligned}$$

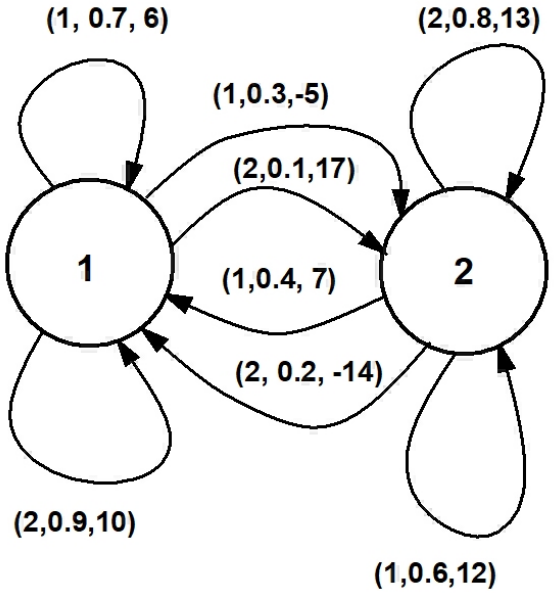

Figure 1.1: A two state MDP: Values on each arrow are defined as follows: (action, transition probability, transition reward)

From the TPMs, via Eqn. (1.6), one determines the steady-state probabilities of each state for a given policy. The equation solving process is demonstrated for one policy as an example. For policy $\hat{\mu}_1$, Eqn. (1.6) can be written as:

$$\begin{aligned}\Pi(1)0.7 + \Pi(2)0.4 &= \Pi(1) \\ \Pi(1)0.3 + \Pi(2)0.6 &= \Pi(2) \\ \Pi(1) + \Pi(2) &= 1\end{aligned}$$

There are three equations in the above but there are two unknowns, $\Pi(1)$ and $\Pi(2)$. Any two of the above three equations are used for solution, as the remaining third then becomes superfluous. The last equation, also called the *normalizing equation*, **must** be used as the third equation to obtain a useful solution. Choosing any one of the first two equations as well as the normalizing equation, you can obtain the following solution easily: $\Pi_{\hat{\mu}_1}(1) = 0.5714$ and $\Pi_{\hat{\mu}_1}(2) = 0.4286$. Using the same process for all the other

three policies, one obtains:

$$\begin{aligned}\Pi_{\hat{\mu}_2}(1) &= 0.4000 \text{ and } \Pi_{\hat{\mu}_2}(2) = 0.6000;\\ \Pi_{\hat{\mu}_3}(1) &= 0.8000 \text{ and } \Pi_{\hat{\mu}_3}(2) = 0.2000;\\ \Pi_{\hat{\mu}_4}(1) &= 0.6667 \text{ and } \Pi_{\hat{\mu}_4}(2) = 0.3333.\end{aligned}$$

Next, it is necessary to defined the expected immediate reward for choosing an action in a state. This, for any state-action pair, (i, a), is defined as:

$$\bar{r}(i,a) = \sum_{j=1}^{|\mathcal{S}|} p(i,a,j)r(i,a,j). \tag{1.8}$$

It should be noted that when the policy is known, then for a policy $\hat{\mu}$, the action selected in state i is $\mu(i)$. Then, the above becomes:

$$\bar{r}(i,\mu(i)) = \sum_{j=1}^{|\mathcal{S}|} p(i,\mu(i),j)r(i,\mu(i),j). \tag{1.9}$$

Average Reward Analysis

Next the average reward of each of the four policies is computed via an exhaustive evaluation of each policy. First evaluate the average *immediate* reward in each possible transition in the MDP, using Eq. (1.9). For this, the TPMs and the TRMs of every policy, which were provided above, are needed. Then, one has that:

$$\begin{aligned}\overline{r}(1,\mu_1(1)) &= p(1,\mu_1(1),1)r(1,\mu_1(1),1) + p(1,\mu_1(1),2)r(1,\mu_1(1),2)\\ &= 0.7(6) + 0.3(-5) = 2.7.\\ \overline{r}(2,\mu_1(2)) &= p(2,\mu_1(2),1)r(2,\mu_1(2),1) + p(2,\mu_1(2),2)r(2,\mu_1(2),2)\\ &= 0.4(7) + 0.6(12) = 10.\\ \overline{r}(1,\mu_2(1)) &= p(1,\mu_2(1),1)r(1,\mu_2(1),1) + p(1,\mu_2(1),2)r(1,\mu_2(1),2)\\ &= 0.7(6) + 0.3(-5) = 2.7.\\ \overline{r}(2,\mu_2(2)) &= p(2,\mu_2(2),1)r(2,\mu_2(2),1) + p(2,\mu_2(2),2)r(2,\mu_2(2),2)\\ &= 0.2(-14) + 0.8(13) = 7.6.\\ \overline{r}(1,\mu_3(1)) &= p(1,\mu_3(1),1)r(1,\mu_3(1),1) + p(1,\mu_3(1),2)r(1,\mu_3(1),2)\\ &= 0.9(10) + 0.1(17) = 10.7.\\ \overline{r}(2,\mu_3(2)) &= p(2,\mu_3(2),1)r(2,\mu_3(2),1) + p(2,\mu_3(2),2)r(2,\mu_3(2),2)\\ &= 0.4(7) + 0.6(12) = 10.\\ \overline{r}(1,\mu_4(1)) &= p(1,\mu_4(1),1)r(1,\mu_4(1),1) + p(1,\mu_4(1),2)r(1,\mu_4(1),2)\\ &= 0.9(10) + 0.1(17) = 10.7.\\ \overline{r}(2,\mu_4(2)) &= p(2,\mu_4(2),1)r(2,\mu_4(2),1) + p(2,\mu_4(2),2)r(2,\mu_4(2),2)\\ &= 0.2(-14) + 0.8(13) = 7.6.\end{aligned}$$

The *average reward* of a policy was previoulsy defined via Eqn. (1.1) as a limit. Nut it can also be expressed in terms of the steady-state (or limiting) probabilities and immediate rewards for each state. Hence, it is now defined as the steady-state probability of being in a state *times* the immediate reward earned in that state— a

quantity summed over all states. Thus, if one considers a state i, the average reward for a policy $\hat{\mu}$, denoted by $\rho_{\hat{\mu}}$, will be $\Pi(i)$ times $\bar{r}(i, \mu(i))$, summed over all states i. Mathematically:

$$\rho_{\hat{\mu}} = \sum_{i=1}^{|\mathcal{S}|} \Pi(i)\bar{r}(i, \mu(i)). \tag{1.10}$$

In this way, one can now compute the average reward of each policy. Via, Eq. (1.10):

$$\rho_{\hat{\mu}_1} = \Pi_{\hat{\mu}_1}(1)\overline{r}(1, \mu_1(1)) + \Pi_{\hat{\mu}_1}(2)\overline{r}(2, \mu_1(2)) = 0.5741(2.7) + 0.4286(10) = 5.83;$$

$$\rho_{\hat{\mu}_2} = \Pi_{\hat{\mu}_2}(1)\overline{r}(1, \mu_2(1)) + \Pi_{\hat{\mu}_2}(2)\overline{r}(2, \mu_2(2)) = 0.4(2.7) + 0.6(7.6) = 5.64;$$

$$\rho_{\hat{\mu}_3} = \Pi_{\hat{\mu}_3}(1)\overline{r}(1, \mu_3(1)) + \Pi_{\hat{\mu}_3}(2)\overline{r}(2, \mu_3(2)) = 0.8(10.7) + 0.2(10) = \mathbf{10.56};$$

$$\rho_{\hat{\mu}_4} = \Pi_{\hat{\mu}_4}(1)\overline{r}(1, \mu_4(1)) + \Pi_{\hat{\mu}_4}(2)\overline{r}(2, \mu_4(2)) = 0.6667(10.7) + 0.3333(7.6) = 9.6667.$$

It should be clear from the above calculations that policy $\hat{\mu}_3 = (2, 1)$ is the optimal policy, as it delivers the *maximum* value for average reward, i.e., 10.56. The process used above to solve the problem is called exhaustive enumeration.

Drawbacks of exhaustive enumeration: Exhaustive enumeration is feasible only on small problems and becomes computationally burdensome for larger problems. Also, it is not really an optimization procedure. For instance, on a problem with 10 states and 2 actions allowed for each state, using exhaustive enumeration requires evaluation of $2^{10} = 1024$ different policies. As such, this approach is ruled out for larger problems,

Next, we show an approach for exhaustive enumeration for discounted reward.

Discounted Reward Analysis

Unlike the average reward case, discounted reward requires the solution of the Bellman equation for a policy defined in Eqn. (1.7) defined above. This equation needs to be solved for every policy for the exhaustive evaluation. The same toy example is now worked out with $\lambda = 0.8$.

Begin with policy $\hat{\mu}_2 = (1, 2)$, whose TPM and TRM are given as:

$$\mathbf{P}_{\hat{\mu}_2} = \begin{bmatrix} 0.7 & 0.3 \\ 0.2 & 0.8 \end{bmatrix}; \mathbf{R}_{\hat{\mu}_2} = \begin{bmatrix} 6 & -5 \\ -14 & 13 \end{bmatrix}.$$

Then, via Eqn. (1.7) for state 1, one has that:

$$J_{\hat{\mu}_2}(1) = \bar{r}(1, \mu_2(1)) + 0.8\left[0.7J_{\hat{\mu}_2}(1) + 0.3J_{\hat{\mu}_2}(2)\right].$$

For state 2, the same equation leads to:

$$J_{\hat{\mu}_2}(2) = \bar{r}(2, \mu_2(2)) + 0.8\left[0.2J_{\hat{\mu}_2}(1) + 0.8J_{\hat{\mu}_2}(2)\right].$$

Remember that $\bar{r}(1, \mu_2(1)) = 2.7$ and $\bar{r}(2, \mu_2(2)) = 7.6$; these values were already obtained above for average reward. Plugging in these values into the two linear equations above and then solving the two equations for the two unknowns, one obtains: $J_{\hat{\mu}_2}(1)$ and $J_{\hat{\mu}_2}(2)$. This leads to:

$$J_{\hat{\mu}_2}(1) = 23.3 \text{ and } J_{\hat{\mu}_2}(2) = 31.47.$$

In a similar way, solving for the other polices leads to:

$$J_{\hat{\mu}_1}(1) = 25.03 \text{ and } J_{\hat{\mu}_1}(2) = 34.63;$$

$$J_{\hat{\mu}_3}(1) = \mathbf{53.03} \text{ and } J_{\hat{\mu}_3}(2) = \mathbf{51.87};$$

$$J_{\hat{\mu}_4}(1) = 50.68 \text{ and } J_{\hat{\mu}_4}(2) = 43.64.$$

From these values $J(.)$, for all states, it is clear that the policy $\hat{\mu}_3 = (2, 1)$ delivers the maximum value of 53.03 for state 1 and 51.87 for state 2, i.e., for both states, implying this *is* the optimal policy.

Eqn. (1.7) can also be used to derive the *policy iteration* algorithm [18] of dynamic programming (DP). There exists another well-known DP algorithm called *value iteration*. Both of these algorithms can be developed for discounted and average reward. These algorithms are significantly less computationally intensive than exhaustive evaluation and provide an elegant way to solve the MDP when the transition probabilities are available.

1.4 SMDPs

In this section, we briefly discuss semi-MDPs (SMDPs) in which the time of transition is not the same in every transition. Let $t(i, a, j)$ denote the transition time from state i to state j when action a is chosen in state i. This time can be a random variable or it can be a deterministic constant. Here, we will limit our analysis to the case where it is a deterministic constant; see [22, 11] for a more detailed discussion on the random case, as well as on this topic. In queuing control as well as scheduling preventive maintenance problems, the SMDP is often a more useful model than the MDP.

1.4.1 Average Reward

Note that Eqn. (1.1) was used to evaluate the average reward of a policy for the MDP. We now extend that concept to the SMDP. Let $\bar{t}(i, a, j)$ denote the expected transition time from state i to state j when action a is chosen in state i. For the average reward performance metric, the average reward of the policy $\hat{\mu}$ starting at state i for the SMDP can be defined as:

$$\rho_{\hat{\mu}}(i) = \lim_{k\to\infty} \frac{\mathsf{E}\left[\sum_{s=1}^{k} r(x_s, \mu(x_s), x_{s+1})|x_1 = i\right]}{\mathsf{E}\left[\sum_{s=1}^{k} \bar{t}(x_s, \mu(x_s), x_{s+1})|x_1 = i\right]}. \tag{1.11}$$

Thus, the average reward here essentially denotes the sum of the total immediate rewards earned divided by the sum of the total expected transition times, calculated over a very long time horizon (i.e., when k assumes a large value.) Intuitively, the average reward here is the expected reward earned per unit time, but the duration of the total time in teh denominator is not to be confused with the number of transitions over the time horizone. Just as in the MDP, when the Markov chain of every policy is regular, it can be shown for the SMDP that the average reward is independent of the starting state, i.e., we can replace $\rho_{\hat{\mu}}(i)$ by $\rho_{\hat{\mu}}$.

The SMDP's average reward can also be expressed in terms of the steady-state probabilities in a manner similar to that used for the MDP. Some additional notation is needed for that: Let $\tau(i, a)$ denote the average transition time when action a is selected in state i. It is defined as:

$$\tau(i, a) = \sum_{i=1}^{|\mathcal{S}|} p(i, a, j)\bar{t}(i, a, j).$$

Then, $\tau(i, \mu(i))$ denotes the average time spent in a transition from state i when policy $\hat{\mu}$ is selected in all states. Now, using the renewal reward theorem, it is easy to show

that the average reward of a policy for the SMDP can be expressed in terms of the steady-state probabilities of the policy's Markov chain as follows:

$$\rho_{\hat{\mu}} = \frac{\sum_{i=1}^{|\mathcal{S}|} \Pi(i)\bar{r}(i,\mu(i))}{\sum_{i=1}^{|\mathcal{S}|} \Pi(i)\tau(i,\mu(i))}. \tag{1.12}$$

1.4.2 Discounted Reward

The notion of the value function of the MDP can be extended to the SMDP as follows. Note that here we will use the rate of interest γ in our formulation. Since the discounted reward measures the **present value** of the sum of the rewards earned in the future over an infinite time horizon, Eqn. (1.4) of the MDP translates into:

$$\lambda^{\tau} = \left(\frac{1}{1+\gamma}\right)^{\tau} \approx \left(e^{-\gamma}\right)^{\tau} = e^{-\gamma\ \tau}, \tag{1.13}$$

where τ denotes the time period over which the discounting is considered. From the above, $e^{-\gamma\ \tau}$ can be viewed as the discount factor where τ denotes the time of transition. Then, under the assumption that all the reward in a transition is earned as a lump sum at the start of the transition, Eqn. (1.3) will have the following interpretation:

$$\begin{aligned} J_{\hat{\mu}}(i) = \mathsf{E}[r(x_1,\mu(x_1),x_2) + e^{-\gamma\ t(x_1,\mu(x_1),x_2)} r(x_2,\mu(x_2),x_3) \\ + e^{-\gamma\ (t(x_1,\mu(x_1),x_2)+t(x_2,\mu(x_2),x_3))} r(x_3,\mu(x_3),x_4) + \cdots] \end{aligned} \tag{1.14}$$

In the above, $e^{-\gamma t(i,a,j)}$ serves as the *discount factor* for one transition from i to j under the influence of action a. Note that although Eqn. (1.14) looks formidable, it can be handled elegantly in DP and RL, as there we will consider discounting over *only one transition at a time,* and the discount factor for one transition is conveniently given as $e^{-\gamma t(i,a,j)}$. This will become clearer in the discussion on reinforcement learning algorithms.

Chapter 2

Dynamic Programming

In this chapter, we will limit our discussion to a technique within dynamic programming (DP), commonly referred to as *value iteration*. DP can be used only when the values of the TPM's elements for each action are available. Of course, the TRM of each action is also needed for DP, but note that the TRM is required even for RL. However, in RL, the TPMs are not needed; instead one uses a simulator of the system to perform experiments, or alternatively, the experiments are performed in the real world.

It is first necessary to introduce elementary vector notation and other related concepts. Here, $\vec{x}$ will denote a vector whose ith element, or component, is denoted by $x(i)$. Thus, for instance, a vector with two elements, 5 and 6, will be denoted as $\vec{x} = (5, 6)$, where $x(1) = 5$ and $x(2) = 6$. Also, if $\mathcal{S}$ denotes a set, then $|\mathcal{S}|$ denotes the number of elements in the set. Finally, the max norm of a vector, denoted by $||x||$, is defined as:

$$||\vec{x}|| = \max_i |x(i)|.$$

Thus, for instance, if $\vec{x} = (-20, 1, 5)$, then: $||\vec{x}|| = \max(|-20|, |1|, |5|) = 20$.

2.1 Discounted Reward MDPs

In this section, we discuss the value iteration algorithm based on the Bellman equation for discounted reward MDPs. The steps in this algorithm are presented via Algorithm 1.

Algorithm 1 Value Iteration for Discounted Reward MDPs

Step 0 (Inputs): Initialize $k = 1$. Select arbitrary values for the elements of the vector, $\vec{J}^k$, of size $|\mathcal{S}|$. Specify a positive but small value for ϵ. Set $done = 0$.
while $done == 0$ **do**
Step 1: For each $i \in \mathcal{S}$, compute:

$$J^{k+1}(i) \leftarrow \max_{a \in \mathcal{A}(i)} \left[\overline{r}(i,a) + \lambda \sum_{j=1}^{|\mathcal{S}|} p(i,a,j) J^k(j) \right].$$

Step 2: If

$$||(\vec{J}^{k+1} - \vec{J}^k)|| < \epsilon(1-\lambda)/(2\lambda),$$

set $done$ to 1. Otherwise, increment k by 1.
end while
Step 3 (Output): For each $i \in \mathcal{S}$ choose

$$d(i) \in \arg\max_{a \in \mathcal{A}(i)} \left[\overline{r}(i,a) + \lambda \sum_{j=1}^{|\mathcal{S}|} p(i,a,j) J^k(j) \right] \text{ and stop.}$$

Numerical Example: The toy example used in the previous chapter is now employed to demonstrate how value iteration can be performed. Here ϵ is set at 0.001. The norm in the algorithm is checked against $0.5\epsilon(1-\lambda)/\lambda = 0.000125$. At $k = 53$, the ϵ-optimal policy, $(2, 1)$, is found. See Table 2.1 for step-by-step details of using the algorithm.

Table 2.1: Value iteration for discounted reward MDPs

k	$J^k(1)$	$J^k(2)$	Norm
1	0	0	-
2	10.700	10.000	10.7
3	19.204	18.224	8.504
4	25.984	24.892	6.781
.	...	...	...
.	...	...	...
53	53.032	51.866	0.000121

2.2 Discounted Reward SMDPs

Under the condition of using lump sum rewards earned at the end of the trajectory, the value iteration algorithm for the discounted reward SMDP can be easily derived from its counterpart for the MDP discussed above, under the assumption that the reward is earned as a lump sum at the start of the transition. The only changes in the algorithm are in replacing the discount factor appropriately. The detailed steps are provided below via Algorithm 2.

Algorithm 2 Value Iteration for Discounted Reward SMDPs

Step 0 (Inputs): Initialize $k = 1$. Select arbitrary values for the elements of the vector, $\vec{J}^k$, of size $|\mathcal{S}|$. Specify a positive but small value for ϵ. Set *done* $= 0$.
while *done* $== 0$ **do**
Step 1: For each $i \in \mathcal{S}$, compute:

$$J^{k+1}(i) \leftarrow \max_{a \in \mathcal{A}(i)} \left[\overline{r}(i,a) + \sum_{j=1}^{|\mathcal{S}|} p(i,a,j) e^{-\gamma \bar{t}(i,a,j)} J^k(j) \right].$$

Step 2: If

$$||(\vec{J}^{k+1} - \vec{J}^k)|| < \epsilon,$$

set *done* to 1. Otherwise, increment k by 1.
end while
Step 3 (Output): For each $i \in \mathcal{S}$ choose

$$d(i) \in \arg\max_{a \in \mathcal{A}(i)} \left[\overline{r}(i,a) + \sum_{j=1}^{|\mathcal{S}|} p(i,a,j) e^{-\gamma \bar{t}(i,a,j)} J^k(j) \right] \text{ and stop.}$$

2.3 Average Reward MDPs

This algorithm has the flavor of the value iteration algorithm presented above, except that a state is chosen at the start and its value is subtracted in every iteration. This state can be any state in the system and is referred to as the distinguished state. This algorithm requires the computation of the span semi-norm, rather than the max norm. The span semi-norm, often called the span, of a vector, $\vec{x}$, is defined as follows:

$$sp(\vec{x}) = \max_i x(i) - \min_i x(i).$$

Thus, if $\vec{x} = (-20, 1, 5)$, then: $sp(\vec{x}) = 5 - (-20) = 25$.

Algorithm 3 Steps in Relative Value Iteration for Average Reward MDPs

Step 0 (Inputs): Select any arbitrary state from $\mathcal{S}$ to be the distinguished state i^*. Set $k = 1$, and select arbitrary values for $J^1(i)$ for each state i. Specify the termination value $\epsilon > 0$. Set $done = 0$.

while $done == 0$ **do**

Step 1: Compute for each $i \in \mathcal{S}$ and a in $\mathcal{A}(i)$:

$$J^{k+1}(i) = \left[\overline{r}(i,a) + \sum_{j=1}^{|\mathcal{S}|} p(i,a,j) J^k(j) \right].$$

After calculations in this step for all states are complete, set $\rho^k = J^{k+1}(i^*)$.

Step 2: For each $i \in \mathcal{S}$, compute:

$$J^{k+1}(i) \leftarrow J^{k+1}(i) - \rho^k.$$

Step 3: If

$$sp(\vec{J}^{k+1} - \vec{J}^k) < \epsilon,$$

set $done = 1$. Otherwise increase k by 1.

end while

Step 4 (Outputs): For each $i \in \mathcal{S}$, choose

$$d(i) \in \arg\max_{a \in \mathcal{A}(i)} \left[\overline{r}(i,a) + \sum_{j=1}^{|\mathcal{S}|} p(i,a,j) J^k(j) \right],$$

and stop. The ϵ-optimal policy is $\hat{d}$, while ρ^k provides the optimal reward's final estimate.

Numerical Example: We use the toy example of the previous chapter for average reward. We use $\epsilon = 0.001$. Results with using relative value iteration are shown via Table 2.2: the ϵ-optimal policy is found at $k = 12$.

Table 2.2: Calculations in *Relative* value iteration for average reward MDPs

Iteration (k)	$J^k(1)$	$J^k(2)$	Span	ρ^{k-1}
1	0	0	-	-
2	0	-0.7	0.7	10.760
3	0	-1.05	0.35	10.630
4	0	-1.225	0.175	10.595
5	0	-1.3125	0.0875	10.578
6	0	-1.35625	0.04375	10.568
7	0	-1.378125	0.021875	10.564
8	0	-1.389063	0.010938	10.562
9	0	-1.394531	0.005469	10.561
10	0	-1.397266	0.002734	10.561
11	0	-1.398633	0.001367	10.560
12	0	-1.399316	0.000684	10.560

2.4 Average Reward SMDPs

We now present an algorithm based on value iteration that uses the so-called Q-values. These Q-values are different than the values of the value function discussed thus far and are used widely in reinforcement learning. A so-called Q-value is associated to each state-action pair, (i, a). The underlying idea here is that when one is maximizing average reward, the optimal action in each state is the one associated to the highest Q-value for that state.

There are other algorithms such as policy iteration and linear programming that can be used to solve the average reward SMDP [11, 22], but here we present a Q-value based algorithm from [17], as it is easier to understand and as it also leads us naturally to reinforcement learning, the next topic to be covered.

Algorithm 4 Steps in Relative Q-value Iteration for Average Reward SMDPs

Step 0. (Initialization): Set the number of iterations, k, to 0. Set all Q-values to 0, i.e., for all $i \in \mathcal{S}$ and all $a \in \mathcal{A}(i)$, set $Q^k(i,a) = 0$. Set ϵ to a small positive value, e.g., 0.001. Set $\alpha^k = \mathsf{A}/(\mathsf{B}+k)$ where A and B are pre-fixed constants that have to be determined empirically. Select any state-action pair arbitrarily as the distinguished state-action pair, (i^*, a^*).

Step 1: Update $Q^k(i,a)$ for each $i \in \mathcal{S}$ and each $a \in \mathcal{A}(i)$ as follows:

$$Q^{k+1}(i,a) = (1-\alpha^k)Q^k(i,a) + \alpha^k \sum_{j \in \mathcal{S}} p(i,a,j) \left[r(i,a,j) - Q^k(i^*,a^*)\bar{\mathsf{t}}(i,a,j) + \max_{b \in \mathcal{A}(j)} Q^k(j,b) \right]. \quad (2.1)$$

Step 2: Compute the value function in the kth and $(k+1)$th iteration, i.e., for each $i \in \mathcal{S}$, calculate: $J^k(i) = \max_{a \in \mathcal{A}(i)} Q^k(i,a)$ and $J^{k+1}(i) = \max_{a \in \mathcal{A}(i)} Q^{k+1}(i,a)$.

Step 3: If the infinity norm of the difference between the vectors, $\vec{J}^{k+1}$ and $\vec{J}^k$, is less than ϵ, i.e., $|| \vec{J}^{k+1} - \vec{J}^k ||_\infty < \epsilon$, go to Step 4; otherwise increment k by 1 and return to Step 1.

Step 4. (Outputs): Compute the ϵ-optimal policy, μ_ϵ, as follows: $\mu_\epsilon(i) \in \arg\max_{a \in \mathcal{A}(i)} Q^{k+1}(i,a)$ for every $i \in \mathcal{S}$ and stop.

Numerical Example: Again, we use a toy example for average reward SMDPs. There are two states numbered 1 and 2, and two actions, also numbered 1 and 2, are allowed in each state. The TPM of action a will be denoted by $\mathbf{P}_a$, while the TRM of action a will be denoted by $\mathbf{R}_a$. The TPMs associated with actions 1 and 2 will be assumed to be:

$$\mathbf{P}_1 = \begin{bmatrix} 0.7 & 0.3 \\ 0.4 & 0.6 \end{bmatrix} \text{ and } \mathbf{P}_2 = \begin{bmatrix} 0.9 & 0.1 \\ 0.2 & 0.8 \end{bmatrix}.$$

The TRMs for actions 1 and 2 are provided below.

$$\mathbf{R}_1 = \begin{bmatrix} 6 & -5 \\ 7 & 12 \end{bmatrix} \text{ and } \mathbf{R}_2 = \begin{bmatrix} 10 & 17 \\ -14 & 13 \end{bmatrix}.$$

The transition times will be assumed to be deterministic. The TTM (Transition Time Matrix) of action a will be denoted by $\mathbf{T}_a$. The TTMS of actions 1 and 2 are provided below:

$$\mathbf{T}_1 = \begin{bmatrix} 1 & 5 \\ 120 & 60 \end{bmatrix} \text{ and } \mathbf{T}_2 = \begin{bmatrix} 50 & 75 \\ 7 & 2 \end{bmatrix}.$$

The value for each $\tau(i, \mu(i))$ term can be calculated as follows:

$$\begin{aligned}\tau(1, \mu_1(1)) &= p(1, \mu_1(1), 1)\tau t(1, \mu_1(1), 1) + p(1, \mu_1(1), 2)\tau t(1, \mu_1(1), 2) \\ &= 0.7(1) + 0.3(5) = 2.20;\end{aligned}$$

Similarly,

$$\tau(2, \mu_1(2)) = 84;\ \tau(1, \mu_2(1)) = 2.20;\ \tau(2, \mu_2(2)) = 3.00;\ \tau(1, \mu_3(1)) = 52.5;$$
$$\tau(2, \mu_3(2)) = 84.0;\ \tau(1, \mu_4(1)) = 52.5;\ \tau(2, \mu_4(2)) = 3.00.$$

The corresponding $\overline{r}(.,)$ terms, needed for computing ρ via Eqn. (1.12), were calculated for the MDP. Then, using Eqn. (1.12), the average reward for each of the four policies is:

$$\rho_{\hat{\mu}_1} = 0.1564, \rho_{\hat{\mu}_2} = \mathbf{2.1045}, \rho_{\hat{\mu}_3} = 0.1796, \text{ and } \rho_{\hat{\mu}_4} = 0.2685.$$

Clearly, policy $\hat{\mu}_2$ is the optimal policy here, which prescribes action 1 for state 1 and action 2 for state 2 with an optimal average reward, ρ^*, of 2.1045.

In the algorithm, α was set to a constant of 0.01, ϵ was set to 0.01, and (i^*, a^*) was set to $(1, 1)$. For problems with a larger number of state-action pairs, a decaying value is set to α (as discussed later in the next chapter).

The algorithm terminates with the following Q-values: $Q(1, 1) = 2.0981$, $Q(1, 2) = -96.0971$, $Q(2, 1) = -159.3346$, and $Q(2, 2) = 8.4978$. Since $Q(1, 1) > Q(1, 2)$, the action prescribed by the algorithm for state 1 is action 1; and since $Q(2, 2) > Q(2, 1)$, the action prescribed by the algorithm for state 2 is action 2. Note that $Q(1, 1) = 2.0981 \approx 2.1045 = \rho^*$.

Chapter 3

Reinforcement Learning

3.1 Reinforcement Learning Based on Q-Learning

RL is primarily used when the transition probabilities of the underlying Markov chains are not available. RL is typically used to conduct trials of actions chosen in each of the state and then observing the resulting reward or cost in one of the following environments:

1. the real-world system

2. a simulator of the real-world system

A simulator of the real-world system can usually be written on the basis of the knowledge of easily accessible input variables for the system's behavior. For example, a queue can be simulated with the knowledge of the distributions of the inter-arrival time and the service time and the number of servers employed. It is necessary to emphasize here that the transition probabilities of the system are typically **not** required for creating a simulator.

Also, it is important to know that the RL algorithms described below require the updating of certain quantities (called Q-factors or Q-values or other forms of state-action and state values) in the computer. Whether a simulator is written in C, MATLAB, R, Python, or in any special package such as ARENA, the program updates these Q-factors or the other quantities required by the algorithm, whenever a new state is visited in the simulator.

Usually, the updating needed has to be performed immediately after a new state is visited. In the simulator, or in real time, it is indeed possible to keep track of the system state continually, so when the system changes, an update is performed.

The key idea in RL is store a so-called Q-factor for each state-action pair in the system, where $Q(i, a)$ denotes the Q-factor for state i and action a. The values of these Q-factors are initialized to arbitrary numbers at the beginning (e.g., zero or any small number). Then the system is simulated or trials are conducted in the real-world system using the algorithm. In each state visited, an action is selected and the system is allowed to transition to the next state. The immediate reward (and the transition time in case of the SMDP) generated in the transition is recorded as the feedback. The feedback is then used to update the Q-factor for the action selected in the previous state. Roughly speaking if the feedback is good, the Q-factor of that particular action and the state in which the action was selected is rewarded (increased when maximizing rewards or decreased when minimizing costs) using the Relaxed-SMART algorithm. If the feedback is poor, the Q-factor is punished by reducing its value (or increasing the value if costs are being minimized).

Then the same reward-punishment policy is carried out in the next state. This is done for a large number of transitions. At the end of this phase, also called the learning phase, the action whose Q-factor has the best (highest when maximizing and lowest when minimizing) value is declared to be the optimal action for that state.

3.1.1 Discounted Reward: Q-Learning

The steps described below are from the famous Q-Learning algorithm of Watkins [25] for an MDP. An SMDP extension is discussed below immediately after these steps. The algorithm will use the notion of step sizes, also called learning rates. The Q-Learning algorithm needs only one step size, the value of which in the kth iteration of the algorithm will be denoted by α^k.

The step size should be a positive value typically less than 1. Some additional conditions have to be satisfied by the step size that are described in detail in [4]. A commonly used example of step-sizes is:

$$\alpha^k = A/(B+k) \tag{3.1}$$

where for instance $A = 1500$ and $B = 3000$. Another example is the log rule: $log(k+1)/(k+1)$ (with k starting at 1) [14]. The rule $1/k$, where $A = 1$ and $B = 0$ in Eqn.(3.1), may not lead to good behavior [14]. Other rules include: $A/(Bk+k^2)$ and the search-then-converge rule [8].

Algorithm 5 Q-Learning

- **Inputs:** Set the Q-factors to arbitrary values, e.g., 0, i.e., set

$$Q^1(i,a) \leftarrow 0 \text{ for all } i \text{ and } a.$$

Let $\mathcal{A}(i)$ denote the set of actions allowed in state i. Let α^k denote the main learning rate in the kth iteration. Set k, the number of iterations in the algorithm, to 1. Set $k_{\max}$, the number of iterations for which the algorithm is run, to a large number.

while $k \leq k_{\max}$ **do**

- **Action Selection and Simulation:** Let the current state be i. Select an action a in state i with probability $1/|\mathcal{A}(i)|$, where $|\mathcal{A}(i)|$ denotes the number of elements in the set $\mathcal{A}(i)$. Alternatively, use an ϵ-greedy mechanism to select action a (this is discussed in the context of the next algorithm). Let the next state be denoted by j. Let $r(i,a,j)$ denote the transition reward under the influence of action a.
- **Updating Q-Factors:** Update $Q^k(i,a)$ as follows:

$$Q^{k+1}(i,a) \leftarrow (1-\alpha^k)Q^k(i,a) + \alpha^k \left[r(i,a,j) + \lambda \max_{b \in \mathcal{A}(j)} Q^k(j,b) \right],$$

where you should compute α^k using one of the rules discussed above.

- Set $k \leftarrow k+1$ and set $i \leftarrow j$.

end while

- **Output:** For each i, compute $d(i) \in \arg\max_{b \in \mathcal{A}(i)} Q^k(i,b)$ and declare $\hat{d}$ as the policy delivered by the algorithm.

Note that in the algorithm above, there is no decay of exploration. This is because in Q-Learning, the exploration need not decay. However, in practice, to get the algorithm to converge faster, people do employ decay of exploration. It is only recommended in online applications and not in simulators.

Table 3.1: Results of Q-Learning for Example A with a number of different step-size rules.

Step Size	$Q(1,1)$	$Q(1,2)$	$Q(2,1)$	$Q(2,2)$
$\alpha^k = 150/(300+k)$	44.40	52.97	51.84	46.63
$\alpha^k = 1/k$	11.46	18.74	19.62	16.52
$\alpha^k = log(k+1)/(k+1)$	39.24	47.79	45.26	42.24

For the SMDP extension of Q-Learning, you can use the following update in Step 3 of the above algorithm (see Bradtke and Duff [7] or [11] for more details):

$$Q^{k+1}(i,a) \leftarrow (1-\alpha^k)Q^k(i,a) + \alpha^k \left[r(i,a,j) + e^{-\gamma t(i,a,j)} \max_{b \in \mathcal{A}(j)} Q^k(j,b) \right],$$

where the exponential term arises as follows:

$$\lambda^\tau = \left(\frac{1}{1+\gamma} \right)^\tau \approx e^{-\gamma\tau}.$$

in which τ denotes the transition time; the discounting mechanism was explained above in Eqn.(1.4). Note that to obtain the exponential term in the above approximation, we use the fact that γ is a small positive number less than 1. Further note that the SMDP extension presented above is under the assumption that the reward is earned as a lump sum at the end of the transition, rather than continuously during the transition. Without this assumption, the problem gets somewhat more complicated and has been is covered in the context of DP in Puterman [22].

Numerical Results with Toy Example We illustrate here the use of Q-learning in the simulator of the Markov chains of the toy example discussed above. We choose $\lambda = 0.8$. Using different step sizes, the results obtained [11] are shown in Table 3.1. As is clear, since $Q(1,2) > Q(1,1)$, the action recommended in state 1 is 2, and since $Q(2,1) > Q(2,2)$, the action recommended in state 2 is 1. Thus, the algorithm returns the optimal policy.

3.1.2 Average reward: R-SMART

The algorithm that we will describe is called R-SMART (short for Relaxed Semi-Markov Average Reward Technique) [13, 12].

• **Learning Rates:** This algorithm will require two different learning rates. Let α^k denote the faster learning rate (i.e., step size) in the kth iteration for the Q-values and let β^k denote the slower learning rate in the kth iteration for the average reward's estimate, ρ^k. An example that can be used is: $\beta^k = \log(k+1)/(k+1)$ and $\alpha^k = A/(B+k)$ [20]. Another example is $\beta^k = A/(B+Ck)$ (with C as a small integer such as 3 or 4) and $\alpha^k = A/(B+k)$ [9].

• **ϵ-Greedy Action Selection:** Determine the action associated to the Q-factor that has the highest value in state i. For example, if there are two actions in a state i and their values are $Q(i,a) = 19$ and $Q(i,2) = 45$, then clearly action 2 has the greatest Q-factor. The action associated to the highest Q-factor is called the **greedy** action. Select the greedy action with probability $(1-\epsilon(k))$. There are multiple ways to define $\epsilon(k)$. E.g., $\epsilon(k) = \frac{1}{|\mathcal{A}(i)|}\frac{A}{B+k}$ where $A > B$ and A and B are large positive constants, e.g., 1000 and 2000 respectively. With a probability of $\frac{\epsilon(k)}{|\mathcal{A}(i)|-1}$, choose any one of the other actions. Another approach to define $\epsilon(k)$ is $\epsilon(k) = \frac{1}{|\mathcal{A}(i)|}\frac{\log(k+1)}{k+1}$. Note that as k

tends to infinity, $\epsilon(k)$ should tend to zero, implying that in the limit, one chooses only the greedy action.

We will describe a basic average reward RL algorithm next that can be used for average reward SMDPs. Note that by setting $t(i,a,j) = 1$ for all values of i, j, and a, one can obtain the steps for an MDP. Hence although our presentation will be for an SMDP, it can easily be translated into that of an MDP by just setting $t(i,a,j) = 1$ in the steps.

Algorithm 6 R-SMART: Learning Phase

- **Inputs:** Initialize all the Q-factors to arbitrary values, e.g., 0. Set the iteration count, k, to 1. Set ρ^1 to an arbitrary value, e.g., 0. Also set the total reward earned in the algorithm, TR^1, as well as the total time spent in the algorithm, TT^1, to zero. Set $k_{\max}$, the maximum number of iterations for which the algorithm is run, to a large number. Set η, a scaling constant, to a small positive value close to but strictly less than 1, e.g., 0.99.

while $k \leq k_{\max}$ **do**

- **Action Selection and Simulation:** Let the current state be i. Use an ϵ-greedy mechanism to select action a. Let the next state be denoted by j. Let $r(i,a,j)$ denote the transition reward and $t(i,a,j)$ denote the transition time associated to the transition from i to j under the influence of action a.
- **Updating Q-Factors:** Update $Q^k(i,a)$ as follows:

$$Q^{k+1}(i,a) \leftarrow (1-\alpha^k)Q^k(i,a)+\alpha^k \left[r(i,a,j) - \rho^k t(i,a,j) + \eta \max_{b \in \mathcal{A}(j)} Q^k(j,b) \right]. \quad (3.2)$$

- **Average Reward Update:** If the action a selected above was non-greedy, skip this step. Otherwise, if the action selected was greedy, set $TR^{k+1} \leftarrow TR^k + r(i,a,j)$ and $TT^{k+1} \leftarrow TT^k + t(i,a,j)$. Then, update average reward using:

$$\rho^{k+1} \leftarrow (1-\beta^k)\rho^k + \beta^k \frac{TR^{k+1}}{TT^{k+1}}. \quad (3.3)$$

- Set $k \leftarrow k+1$ and set $i \leftarrow j$.

end while

- **Output:** For each i, compute $d(i) \in \arg\max_{b \in \mathcal{A}(i)} Q^k(i,b)$ and declare $\hat{d}$ as the policy delivered by the algorithm.

If one is interested in estimating a value of ρ more accurate than the one delivered by the algorithm's policy, one can use the following frozen phase in which the system is re-simulated with the policy delivered by the algorithm. The next phase is called the frozen phase because the Q-factors are not changed here but only used to determine the fixed policy contained in the Q-factors.

- **Initialize** Set iteration count k to 0. Also set TR and TT to 0. Further, set $k_{\max}$ to a large number.

- DO WHILE $k \leq k_{\max}$:

- **Simulate Action:** For state i, select an action $u = \arg\max_{b \in \mathcal{A}(i)} Q(i,b)$. Simulate action u. Let the next state be denoted by j. Let $r(i,u,j)$ denote the transition reward and $t(i,u,j)$ denote the transition time.

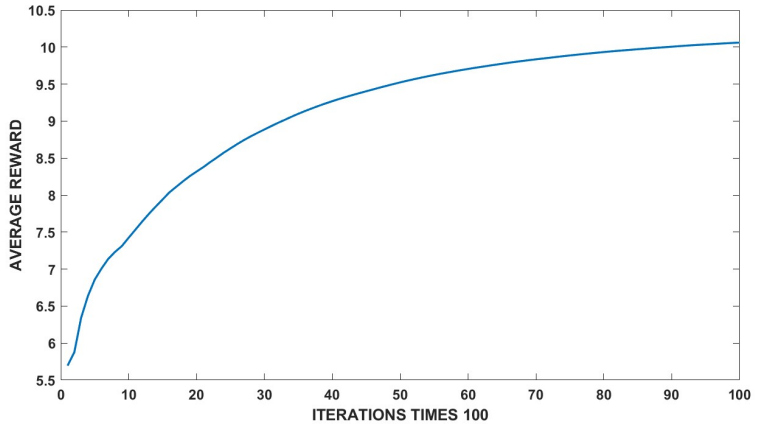

Figure 3.1: Progress of R-SMART on Example B during the algorithm's runtime

- **Update Average Reward Parameters:**

$$TR \leftarrow TR + r(i,u,j);\ TT \leftarrow TT + t(i,u,j);\ \rho = \frac{TR}{TT}.$$

- Increment k by 1 and set $i \leftarrow j$.
- END WHILE
- **Output:** The policy's average reward: ρ.

Numerical Results on Toy Example with R-SMART: This is the toy example adapted to the average reward MDP.

Exploration was decayed as follows in the R-SMART algorithm:

$$\epsilon(k) = 0.5(0.999)^{k-1}.$$

Using $\eta = 0.99$, the log-rule for β^k, and $\alpha^k = 9/(100+k)$, the Q-factors obtained are:

$$Q(1,1) = 28.7750, Q(1,2) = 48.3927, Q(2,1) = 45.0252, \text{ and } Q(2,2) = 20.1286.$$

The policy returned by R-SMART is $(2,1)$ (since action for state 1 is 2 because $Q(1,2) > Q(1,1)$ and action for state 2 is because $Q(2,1) > Q(2,2)$), which is the optimal policy. The optimal average reward here is 10.56. The algorithm converges to $\rho = 10.06$. Note that $10.06 \simeq 10.56$. Generating the optimal policy is key here, as a more accurate value for ρ can be re-estimated via the frozen phase. Fig. 3.1 shows a plot of the average reward, ρ^k versus k as the algorithm makes progress during its runtime.

3.1.3 Codes

MATLAB and C codes for these algorithms have been placed on the following website:

`https://simoptim.com/bookcodes.html`

3.2 Reinforcement Learning Based on Actor-Critics

The Actor-Critic Algorithm is based on the Bellman equation for a given policy, and it uses the value function instead of the Q-function, in addition to so-called actor-values

(one for each state-action pair) that help guide the algorithm to the optimal or near-optimal solution. The actor-critic has an interesting history, as it predates Q-Learning [2]. It was shown in [19] that the values of the actor can get unbounded, as the update in the algorithm is not a convex combination, leading to computational issues. Hence, in [19] a projection of the actor's value was suggested to circumvent the issue. Here, a different algorithm from Lawhead and Gosavi [20] in which the actor's values are *naturally bounded* and do *not* need a projection is presented. We first present the algorithm for the discounted reward MDP.

In the actor critic, one often uses the so-called Gibbs softmax approach to action selection in which the probability of selecting an action a in state i is given by:

$$\pi(i,a) = \frac{\exp(H(i,a))}{\sum_b \exp(H(i,b))}, \tag{3.4}$$

where $H(i,a)$ denotes the so-called actor's value for the state-action pair (i,a).

3.2.1 Discounted Reward

Algorithm 7 Bounded Actor Critic for Discounted Reward MDPs

Step 0. (Inputs) Initialize all J-values and H-values to 0, i.e., for all l, where $l \in \mathcal{S}$ and $u \in \mathcal{A}(l)$, set $J(l) \leftarrow 0$ and $H(l,u) \leftarrow 0$. Set k, the number of iterations, to 0. The algorithm will be run for $k_{\max}$ iterations, where $k_{\max}$ is chosen to be sufficiently large.

while $k \leq k_{\max}$ **do**

Step 1. Let the current state be i. Select action a with a probability of $\pi(i,a)$ defined in Eq.(3.4) or using the ϵ-greedy approach discussed above. Simulate action a. Let the next state be j. Let $r(i,a,j)$ be the immediate reward earned in going to j from i under a.

Step 2 (Actor's Update). Update $H(i,a)$ using a step size α as follows:

$$H(i,a) \leftarrow (1-\alpha)H(i,a) + \alpha[r(i,a,j) - J(i) + \lambda J(j)]. \tag{3.5}$$

Step 3 (Critic's Update). Update $J(i)$, using step size β as follows:

$$J(i) \leftarrow (1-\beta)J(i) + \beta\left[r(i,a,j) + \lambda J(j)\right]. \tag{3.6}$$

Step 4. Increment k by 1 and set $i \leftarrow j$.

end while

Step 5. (Output) For each $l \in S$, select $d(l) \in \arg\max_{b \in \mathcal{A}(l)} H(l,b)$. The policy (solution) generated by the algorithm is $\hat{d}$. Stop.

Another version of this algorithm from [16] uses the following update for the actor:

$$H'(i,a) \leftarrow (1-\alpha)H'(i,a) + \alpha[r(i,a,j) + \lambda J(j)].$$

We present numerical results from a two-state MDP discussed above (toy example). An ϵ-greedy action-selection strategy was used with the greedy action selected with a probability of $0.5(0.999)^{k-1}$. The algorithm was run for 10,000 iterations with $\alpha^k = 150/(300+k)$ and $\beta^k = log(k+1)/(k+1)$. The values returned for the critic are: $J(1) = 51.4708$ and $J(2) = 48.3144$. The values returned for the actor are $H(1,1) = -7.8285$, $H(1,2) = 0.1846$, $H(2,1) = 1.3059$, and $H(2,2) = 1.0329$. Since, $H(1,2) > H(1,1)$ and $H(2,1) > H(2,2)$, the policy delivered by the algorithm is $(2,1)$, which is the optimal policy. The values of the critic as generated during the algorithm's runtime are shown in Fig. 3.2.

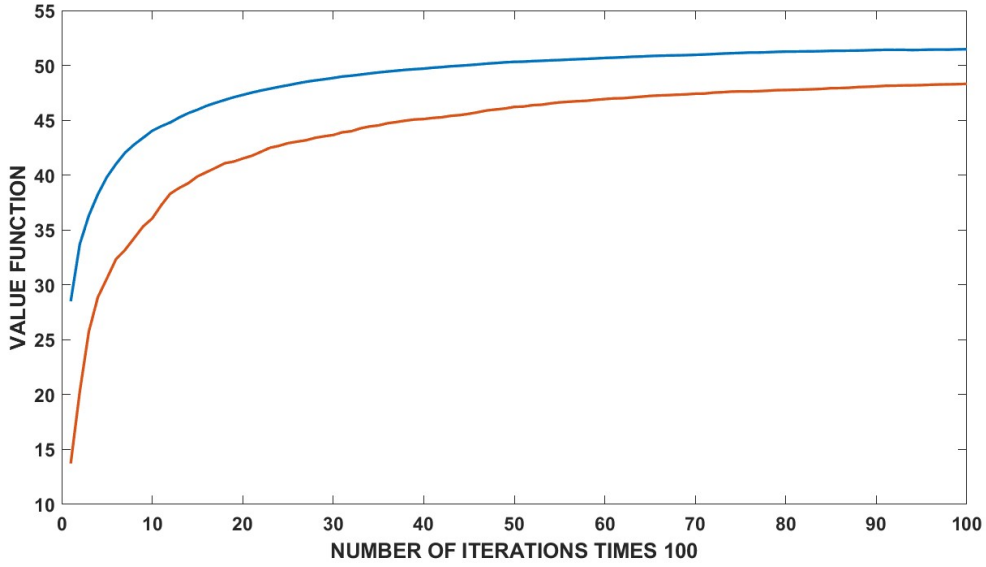

Figure 3.2: The values of the critic from the bounded actor critic on the two-state discounted reward MDP during the algorithm's runtime

3.2.2 Average Reward

The following version of the algorithm from [20] is for the average reward SMDP. The MDP version can be obtained by setting $t(.,.,.) = 1$ everywhere in the steps.

Algorithm 8 Bounded Actor Critic for Average Reward SMDPs

Step 0. (Inputs) Initialize all J-values and H'-values to 0, i.e., for all l, where $l \in \mathcal{S}$ and $u \in \mathcal{A}(l)$, set $J(l) \leftarrow 0$ and $H(l,u) \leftarrow 0$. Set k, the number of iterations, to 0. Set $\rho = 0$. Also, set both TR and TT, the total reward and the total time spent in the simulator, respectively, to 0. Select a positive value for η that is large enough but less than 1. The algorithm will be run for $k_{\max}$ iterations, where $k_{\max}$ is chosen to be sufficiently large.

while $k \leq k_{\max}$ **do**

Step 1. Let the current state be i. Select action a with a probability of $\pi(i,a)$ defined using the Gibbs softmax method or using an ϵ-greedy strategy. Simulate action a. Let the next state be j. Let $r(i,a,j)$ and $t(i,a,j)$ denote the immediate reward earned and the transition time spent, respectively, in going to j from i under the influence of a.

Step 2 (Actor's Update). Update $H(i,a)$ using a step size, α, as follows:

$$H(i,a) \leftarrow (1-\alpha)H(i,a) + \alpha[r(i,a,j) - \rho t(i,a,j) + \eta J(j) - J(i)].$$

Step 3 (Critic's Update). Update $J(i)$ via the following equation using the step-size, β:

$$J(i) \leftarrow (1-\beta)J(i) + \beta[r(i,a,j) - \rho t(i,a,j) + \eta J(j)].$$

Step 4 (Update of Average Reward). Update TR, TT, and ρ via the following equations:

$$TR \leftarrow TR + r(i,a,j);\ TT \leftarrow TT + t(i,a,j);\ \rho \leftarrow (1-\overline{\gamma})\rho + \overline{\gamma}TR/TT.$$

Step 5. Increment k by 1 and set $i \leftarrow j$.

end while

Step 6. (Output) For each $l \in S$, select $d(l) \in \arg\max_{b \in \mathcal{A}(l)} H(l,b)$. The policy (solution) generated by the algorithm is $\hat{d}$. Stop.

The average reward problem with two states and two actions was used for experiments with the actor-critic algorithm of this subsection. The same step sizes were used as in the previous subsection; the step size for the critic was also used for the average reward. The following values resulted: $J(1) = 29.8024$, $J(2) = 26.0745$, $H(1,1) = -7.8089$, $H(1,2) = 0.1326$, $H(2,1) = 2.3610$, and $H(2,2) = -1.6103$. As discussed in the context of the discounted reward in the previous subsection, from inspecting the actor values, the policy delivered can be shown to $(2,1)$, which is the optimal policy. The final value of ρ was 0.23. Fig. 3.3 shows the plot of the average reward against the number of iterations during the run time of the algorithm.

3.2.3 Codes

MATLAB codes for these algorithms can be found at the following website: go to paper number 34.

`https://simoptim.com/main/papers.html`

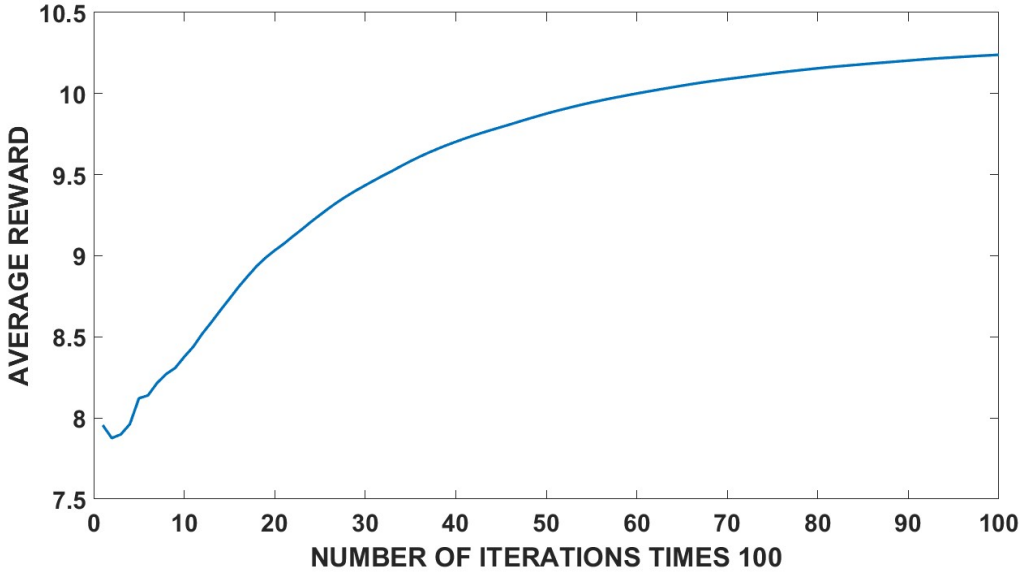

Figure 3.3: The value of average reward of the bounded actor critic on the two-state average reward MDP during the algorithm's runtime

Chapter 4

Convergence Properties

In this section, a brief overview of mathematical convergence properties of two main algorithms, Q-Learning for discounted reward MDPs and R-SMART for average reward SMDPs, is presented. To this end, the ordinary differential equation (ODE) approach of Borkar [6] is employed.

It is helpful to first ensure the reader understands what is meant by a transformation (also called mapping). A transformation is essentially a function that takes a vector as an input and creates (or delivers) another vector as its output.

4.1 Single Timescale Asynchronous Stochastic Approximation

The single timescale algorithm in which only one step size (e.g., α) is used. An example of this is Q-Learning.

4.1.1 Main Result

Consider the asynchronous stochastic approximation update in a single timescale algorithm defined as follows: For $l = 1, 2 \ldots, N$,

$$X^{k+1}(l) = X^k(l) + \alpha^k(l) \left[F(\vec{X}^k)(l) + w^k(l) \right] I(l = \Theta^k), \tag{4.1}$$

where

- $X^k(l)$ denotes the lth iterate in the kth iteration of the algorithm, which is essentially the Q-factor in the kth iteration, l is an index for the iterate for $l = 1, 2, \ldots, N$, and usually a unique state-action pair is associated to each iterate, implying there are N state-action pairs in the problem
- $\alpha^k(l)$ denotes the step size or learning rate for the lth iterate in the kth iteration
- $F(.)$ denotes the transformation or mapping used on the vector of iterates, $\vec{X}^k$; this is usually the transformation used in dynamic programming within the original Bellman equation and contains the transition probabilities
- $w^k(l)$ denotes the noise in the kth iteration of the algorithm associated to the lth iterate, which arises from the simulator or the real-world system in which the algorithm operates; note that added with $F(.)$, it ensures that the actual transformation used in the RL algorithm is captured accurately here for our analysis via

$$\left[F(\vec{X}^k)(l) + w^k(l) \right],$$

rather than the transformation $F(.)$ used in dynamic programming, which contains transition probabilities

- $I(l = \Theta^k)$ denotes the indicator function, which ensures that, in the kth iteration, only one Q-factor gets updated, and that the updated Q-factor is the one associated to the state-action pair visited in that iteration; the indicator function returns a zero for a Q-factor *not* visited in the kth iteration, indicating that in under asynchronous conditions, the Q-factors *not* visited in that iteration are *not* updated, and the trajectory of state-action pairs visited is denoted by:

$$\{\Theta^1, \Theta^2, \ldots\}.$$

Mathematically speaking: $l \equiv (i, a)$ and hence $X^k(l) \equiv Q^k(i, a)$, i.e., the Q-factor.

Note that $F(.)$ *is* the transformation used in the Bellman equation, and hence the above implies that the algorithm converges to the Bellman optimality solution. Intuitively, it is not difficult to see from Eq. (4.1) that if convergence of the sequence of iterates occurs, then, beyond a finite of value of k: For $l = 1, 2 \ldots, N$,

$$X^{k+1}(l) = X^k(l).$$

Then, since the noise is a term with zero mean and since it disappears in the limit, from Eq. (4.1), one has that: For $l = 1, 2 \ldots, N$,

$$0 = \alpha^k(l) \left[F(\vec{X}^k)(l) + w^k(l) \right] I(l = \Theta^k),$$

which implies that beyond a finite of value of k: For $l = 1, 2 \ldots, N$,

$$F(\vec{X}^k)(l) = 0.$$

The above says that the algorithm converges to the solution of the Bellman optimality equation.

The following is the main result that will be used to mathematically prove convergence of the iterates $X^k(.)$ in the algorithm. It essentially says that when the algorithm satisfies a number of conditions, the algorithm converges to the following limit:

$$\lim_{x \to k} F(\vec{X}^k) = \vec{0}.$$

Theorem 1. *Consider the update defined in Eq. (4.1). Assume that the following conditions hold:*

Condition 1. *Lipschitz continuity of $F(.)$: The transformation $F(.)$ is Lipschitz continuous.*

Condition 2. *Conditions on noise: For $l = 1, 2, \ldots, N$ and for every k, the following holds for the noise:*

$$\mathtt{E}\left[w^k(l) | \mathcal{F}^k \right] = 0; \ \mathtt{E}\left[\left(w^k(l) \right)^2 \middle| \mathcal{F}^k \right] \leq z_1 + z_2 ||\vec{X}^k||^2;$$

where z_1 and z_2 are scalar constants, $||.||$ could be any norm, and $\mathcal{F}^k$ is a set that contains the history of the algorithm until and including k.

Condition 3. *Standard step-size conditions of stochastic approximation: The step size $\alpha^k(l)$ satisfies the following conditions for every $l = 1, 2, \ldots, N$:*

$$\sum_{k=1}^{\infty} \alpha^k(l) = \infty; \sum_{k=1}^{\infty} (\alpha^k(l))^2 < \infty.$$

Condition 4. *Asymptotically stable critical point for ODE: Let* $\vec{x}$ *denote the continuous-valued variable underlying the iterate* $\vec{X}^k$*. Then, the ODE*

$$\frac{d\vec{x}}{dt} = F(\vec{x})$$

has a unique globally asymptotically stable equilibrium point, which is denoted by $\vec{x}_*$*.*

Condition 5. *Boundedness of iterates: The iterate* $\vec{X}^k$ *remains bounded with probability 1.*

Condition 6. *ITS (ideal tapering sizes) conditions on step sizes: For every* $l = 1, 2, \ldots, N$*,*

(i) $\alpha^{k+1}(l) \leq \alpha^k(l)$ *for some* k *onward*

(ii) For any $z \in (0,1), \sup_k \alpha^{[zk]^{int}}(l)/\alpha^k(l) < \infty$ *where* $[.]^{int}$ *denotes the integer part of what is inside the square brackets.*

(iii) For any $z \in (0,1)$*,*

$$\lim_{k\to\infty} \frac{\sum_{m=1}^{[zk]+1} \alpha^m(l)}{\sum_{m=1}^{k} \alpha^m(l)} = 1.$$

Condition 7. *EDU (evenly distributed updating) conditions on updating: Let* $V^k(l) = \sum_{m=1}^{k} I(l = \Theta^k)$ *for* $l = 1, 2, \ldots, N$*. Then, with probability 1, there exists a deterministic scalar* $\mathsf{A} > 0$ *such that for all* l*:*

$$\liminf_{k\to\infty} \frac{V^k(l)}{k} \geq \mathsf{A}.$$

Further, for any scalar $z > 0$*, define for every* l*,*

$$K^k(z,l) \equiv \min\left\{ m > k : \sum_{s=k+1}^{m} \alpha^s(l) > z \right\}.$$

Then, with probability 1, the following limit must exist for all (l, l') *pairs, where each of* l *and* l' *assumes values from* $\{1, 2, \ldots, N\}$*, and any* $z > 0$*:*

$$\lim_{k\to\infty} \frac{\sum_{m=V^k(1)}^{V^{K^k(z,l)}(l)} \alpha^k(l)}{\sum_{m=V^k(l)}^{V^{K^k(z,l')}(l)} \alpha^k(l)}.$$

Then, the sequence $\{\vec{X}^k\}_{k=1}^{\infty}$ *converges to* $\vec{x}_*$ *with probability 1. In other words, the sequence of iterates generated in the algorithm converges to*

$$F(.) = 0,$$

which is the Bellman optimality solution.

See Borkar [5] for the proof of the above. Condition 3 is standard in stochastic approximation, satisfied by a step size such as $A/(B+k)$. Conditions 6 and 7 are also satisfied by standard step sizes such as $A/(B+k)$, provided sufficient exploration is used (see also Rokhlin [23], who investigates related issues), but not by the polynomial step sizes or hasty (speedy) learning (which typically violates the EDU condition).

4.1.2 Q-Learning

This is a single timescale algorithm, i.e., only one step size, α, is used. The goal is to show that the Q-factor tracks an imaginary ordinary differential equation (ODE). If the ODE has a unique globally *asymptotically stable equilibrium*, then the Q-factors will also converge to it.

Proposition 1. *When the step sizes and action selection used in the algorithm satisfy Conditions 3, 6, and 7 of Theorem 1, with probability 1, the sequence of policies generated by the Q-Learning algorithm converges to an optimal policy.*

Proof. To invoke Theorem 1, it is first necessary to show that this algorithm is of the form defined in that result. To this end, transformations, $F(.)$, $F'(.)$ and $f'(.)$, on the vector $\vec{Q}^k$ need to be defined:

$$F(\vec{Q}^k)(i,a) = \sum_{j=1}^{|\mathcal{S}|} p(i,a,j)\left[r(i,a,j) + \lambda \max_{b\in\mathcal{A}(j)} Q^k(j,b)\right] - Q^k(i,a);$$

$$F'(\vec{Q}^k)(i,a) = \sum_{j=1}^{|\mathcal{S}|} p(i,a,j)\left[r(i,a,j) + \lambda \max_{b\in\mathcal{A}(j)} Q^k(j,b)\right]; \tag{4.2}$$

which implies that for all (i,a) pairs,

$$F(\vec{Q}^k)(i,a) = F'(\vec{Q}^k)(i,a) - Q^k(i,a). \tag{4.3}$$

Further define a transformation $f'(.)$ as follows, which contains the sample of the transformation in $F'(.)$:

$$f'(\vec{Q}^k)(i,a) = \left[r(i,a,\xi^k) + \lambda \max_{b\in\mathcal{A}(\xi^k)} Q^k(\xi^k,b)\right].$$

Now, if the noise term is defined as:

$$w^k(i,a) = f'(\vec{Q}^k)(i,a) - F'(\vec{Q}^k)(i,a), \tag{4.4}$$

then, the updating transformation in Q-Learning can be written as:

$$Q^{k+1}(i,a) = Q^k(i,a) + \alpha^k\left[f'(\vec{Q}^k)(i,a) - Q^k(i,a)\right].$$

Now, using Eq. (4.3), one can re-write the above:

$$Q^{k+1}(i,a) = Q^k(i,a) + \alpha^k\left[F(\vec{Q}^k)(i,a) + w^k(i,a)\right],$$

which is of the same form in Theorem 1 (replace X^k by Q^k and l by (i,a)). Then, one can invoke the following ODE as in Condition 4 of Theorem 1:

$$\frac{d\vec{q}}{dt} = F(\vec{q}), \tag{4.5}$$

where $\vec{q}$ represents the continuous-valued variable underlying the iterate $\vec{Q}$.

All the conditions of Theorem 1 will now be inspected.

Condition 1. Lipschitz continuity of $F(.)$ can be shown from the fact that the partial derivatives $F(\vec{Q}^k)$ with $Q^k(i,a)$ are bounded.

Condition 2. From its definition it is intuitively clear that the noise term $w^k(.,.)$ is the difference between a random value and a conditional mean of the value. Using rigorous mathematical arguments that are beyond our scope here the noise term can be shown to be a martingale, whose conditional mean is zero. Its (conditional) second moment can be bounded by a function of the square of the iterate.

Condition 3. These conditions are standard in stochastic approximation and can be proved for a step size such as $A/(B+k)$.

Condition 4. To show this condition, one needs to show that $F'(.)$ is a contraction mapping. This is shown via Lemma 3 in the Appendix.

Then, from Theorem 3 (see Appendix), the ODE in Eq. (4.5) must have a unique globally asymptotically stable equilibrium.

Condition 5. This is the boundedness condition, proved below in Lemma 1.

Conditions 6 and 7. These conditions are ensured with appropriate step sizes and by appropriate exploration, as discussed above.

Then, by Theorem 1, one has convergence with probability 1 to the unique globally asymptotically stable equilibrium of the ODE in Eq. (4.5); denote the equilibrium by $\vec{Q}^\infty$. In other words, for all (i,a) pairs:

$$F\left(\vec{Q}^\infty\right)(i,a) = 0, \text{i.e., } F'\left(\vec{Q}^\infty\right)(i,a) = Q^\infty(i,a).$$

□

One of the key conditions in these analyses is to show that iterates, i.e., Q-factors, themselves remain bounded. Hence, we first show that the Q-factors remain bounded in Q-Learning using a result from Gosavi [10].

Lemma 1. *In Q-Learning, under asynchronous updating, the value of $Q^k(i,a)$ for any state-action pair (i,a) in its kth update remains bounded, as long as the starting values for the Q-factors are finite and $\lambda \in [0,1)$.*

Proof. It is first claimed that for every state-action pair (i,a):

$$|Q^k(i,a)| \leq M(1+\lambda+\lambda^2+\cdots+\lambda^k), \tag{4.6}$$

where M is a positive finite number defined as:

$$M = \max\left\{r_{\max}, \max_{i\in\mathcal{S}, a\in\mathcal{A}(i)} Q^1(i,a)\right\}, \tag{4.7}$$

$$\text{in which } r_{\max} = \max_{i,j\in\mathcal{S}, a\in\mathcal{A}(i)} |r(i,a,j)|. \tag{4.8}$$

Since $r(.,.,.)$ is finite by definition, $r_{\max}$ is also finite. And since the algorithm is initiated with finite values for the Q-factors, M must be finite as well. Then, from the claim in (4.6), boundedness of the Q-factors must follow as $k \to \infty$,

$$\limsup_{k\to\infty} |Q^k(i,a)| \leq M\frac{1}{1-\lambda}$$

for all $i \in \mathcal{S}$ and $a \in \mathcal{A}(i)$, because $0 \leq \lambda < 1$ (using the geometric series). A simple induction argument is needed for the proof.

Because in asynchronous updating, the Q-factor for *only one* state-action pair is updated at one time, while the other Q-factors remain unchanged, for the kth iteration of the asynchronous algorithm, the update for $Q^k(i,a)$ functions either according to Case 1 or Case 2 below.

Case 1: The state-action pair is updated in the kth iteration as follows:

$$Q^{k+1}(i,\ a) = (1-\alpha)Q^k(i,a) + \alpha\left[r(i,a,j) + \lambda \max_{b\in\mathcal{A}(j)} Q^k(j,b)\right].$$

Case 2: The state-action pair is not updated in the kth iteration as follows:

$$Q^{k+1}(i,a) = Q^k(i,a).$$

For Case 1:

$$\begin{aligned}
|Q^2(i,a)| &\leq (1-\alpha)|Q^1(i,a)| + \alpha|r(i,a,j) + \lambda \max_{b\in\mathcal{A}(j)} Q^1(j,b)| \\
&\leq (1-\alpha)M + \alpha M + \alpha\lambda M \text{ (from (4.8) and (4.7))} \\
&\leq (1-\alpha)M + \alpha M + \lambda M \text{ (from the fact that } \alpha \leq 1) \\
&= M(1+\lambda)
\end{aligned}$$

For Case 2:

$$\begin{aligned}
|Q^2(i,a)| &= |Q^1(i,a)| \\
&\leq M \leq M(1+\lambda).
\end{aligned}$$

From the above, the claim in (4.6) is true for $k = 1$. Assuming the claim is true when $k = m$, one has that for all $(i,a) \in (\mathcal{S}\times\mathcal{A}(i))$.

$$|Q^m(i,a)| \leq M(1+\lambda+\lambda^2+\cdots+\lambda^m). \tag{4.9}$$

Now, if the update is carried out as in Case 1:

$$\begin{aligned}
|Q^{m+1}(i,a)| &\leq (1-\alpha)|Q^m(i,a)| + \alpha|r(i,a,j) + \lambda \max_{j\in\mathcal{A}(j)} Q^m(j,b)| \\
&\leq (1-\alpha)M(1+\lambda+\lambda^2+\cdots+\lambda^m) \\
&\quad +\alpha M + \alpha\lambda M(1+\lambda+\lambda^2+\cdots+\lambda^m) \text{ (from (4.9))} \\
&= M(1+\lambda+\lambda^2+\cdots+\lambda^m) \\
&\quad -\alpha M(1+\lambda+\lambda^2+\cdots+\lambda^m) \\
&\quad +\alpha M + \alpha\lambda M(1+\lambda+\lambda^2+\cdots+\lambda^m) \\
&= M(1+\lambda+\lambda^2+\cdots+\lambda^m) \\
&\quad -\alpha M(1+\lambda+\lambda^2+\cdots+\lambda^m) \\
&\quad +\alpha M + \alpha M(\lambda+\lambda^2+\cdots+\lambda^{m+1}) \\
&= M(1+\lambda+\lambda^2+\cdots+\lambda^m) \\
&\quad -\alpha M(1+\lambda+\lambda^2+\cdots+\lambda^m) + \alpha M \\
&\quad +\alpha M(\lambda+\lambda^2+\cdots+\lambda^m) + \alpha M\lambda^{m+1} \\
&= M(1+\lambda+\lambda^2+\cdots+\lambda^m) - \alpha M(1+\lambda+\lambda^2+\cdots+\lambda^m) \\
&\quad +\alpha M(1+\lambda+\lambda^2+\cdots+\lambda^m) + \alpha M\lambda^{m+1} \\
&= M(1+\lambda+\lambda^2+\cdots+\lambda^m) + \alpha M\lambda^{m+1} \\
&\leq M(1+\lambda+\lambda^2+\cdots+\lambda^m) + M\lambda^{m+1} \text{ (since } 0 < \alpha \leq 1) \\
&= M(1+\lambda+\lambda^2+\cdots+\lambda^m+\lambda^{m+1})
\end{aligned}$$

Now, if the update is carried out as in Case 2:

$$\begin{aligned}
|Q^{m+1}(i,a)| &= |Q^m(i,a)| \\
&\leq M(1+\lambda+\lambda^2+\cdots+\lambda^m) \\
&\leq M(1+\lambda+\lambda^2+\cdots+\lambda^m+\lambda^{m+1})
\end{aligned}$$

From the above, the claim in (4.6) is proved for $k = m+1$. □

4.2 Two Timescale Asynchronous Stochastic Approximation

A number of RL algorithms such as R-SMART and actor critics work on two timescales, i.e., there are two different classes of iterates being updated and they use different step sizes, usually, one decaying to zero faster than the other. Thus, there is a faster timescale and a slower timescale.

4.2.1 Main Result

Let $\vec{X}^k$ denote the vector of iterates on the *faster* time scale and $\vec{Y}^k$ that on the *slower* time scale, assuming there are N_1 iterates on the faster time scale and N_2 iterates on the slower time scale. Further, the simulator generates two trajectories:

$$\{\Theta_1^1, \Theta_1^2, \ldots\} \text{ and } \{\Theta_2^1, \Theta_2^2, \ldots\},$$

where Θ_1^k and Θ_2^k denote the state-action pair or state from the faster and slower time scale, respectively, whose value is updated in the kth iteration. Thus, for $k = 1, 2, \ldots$:

$$\Theta_1^k \in \{1, 2, \ldots, N_1\} \text{ and } \Theta_2^k \in \{1, 2, \ldots, N_2\}.$$

A two-timescale algorithm under asynchronous updating can be formally presented as: For $l = 1, 2 \ldots, N_1$ and $l_2 = 1, 2, \ldots, N_2$:

$$\begin{aligned} X^{k+1}(l_1) &= X^k(l_1) + \alpha^k(l_1)[F(\vec{X}^k, \vec{Y}^k)(l_1) + w_1^k(l_1)]I(l_1 = \Theta_1^k); \\ Y^{k+1}(l_2) &= Y^k(l_2) + \beta^k(l_2)[G(\vec{X}^k, \vec{Y}^k)(l_2) + w_2^k(l_2)]I(l_2 = \Theta_2^k); \end{aligned} \tag{4.10}$$

where

- $\alpha^k(.)$ and $\beta^k(.)$ are the step sizes for the faster and slower time-scale iterates respectively
- $F(.,.)$ and $G(.,.)$ denote the transformations driving the faster and slower time-scale updates respectively
- $\vec{w}_1^k$ and $\vec{w}_2^k$ denote the noise vectors in the kth iteration on the faster and slower time scales respectively

Note that $F(.,.)$ is a function of X and Y, and the same is true of $G(.,.)$. Clearly, hence, the fates of the iterates are intertwined (or coupled) because their values are inter-dependent.

The conditions use the format of Theorem 1.

Theorem 2. *Consider the two-time-scale asynchronous algorithm defined in Eq. (4.10). Assume that the following conditions hold:*

Condition 1. *Lipschitz continuity of the underlying transformations: The functions $F(.,.)$ and $G(.,.)$ are Lipschitz continuous.*

Condition 2. *Conditions on noise: For $l_1 = 1, 2, \ldots, N_1, l_2 = 1, 2, \ldots, N_2$, and for every k, the following should be true about the noise terms:*

$$\begin{aligned} \mathtt{E}\left[w_1^k(l_1)|\mathcal{F}^k\right] &= 0; \mathtt{E}\left[\left(w_1^k(l_1)\right)^2|\mathcal{F}^k\right] \leq z_1 + z_2||\vec{X}^k||^2 + z_3||\vec{Y}^k||^2; \\ \mathtt{E}\left[w_2^k(l_2)|\mathcal{F}^k\right] &= 0; \mathtt{E}\left[\left(w_2^k(l_2)\right)^2|\mathcal{F}^k\right] \leq z_1' + z_2'||\vec{X}^k||^2 + z_3'||\vec{Y}^k||^2; \end{aligned}$$

where $z_1, z_2, z_3, z_1', z_2'$, and z_3' are scalar constants and $||.||$ could be any norm.

Condition 3. *Step-size conditions of stochastic approximation: The step sizes satisfy the usual tapering size conditions for every $l_1 = 1, 2, \ldots, N_1$ and $l_2 = 1, 2, \ldots, N_2$:*

$$\sum_{k=1}^{\infty}(\alpha^k(l_1))^2 < \infty; \sum_{k=1}^{\infty}\alpha^k(l_1) = \infty;$$
$$\sum_{k=1}^{\infty}\beta^k(l_2) = \infty; \sum_{k=1}^{\infty}(\beta^k(l_2))^2 < \infty;$$

In addition, the step sizes must satisfy the following condition for every (l_1, l_2) pair:

$$\limsup_{k\to\infty}\frac{\beta^k(l_2)}{\alpha^k(l_1)} = 0;$$

Condition 4a. *ODE condition: For any fixed value of $\vec{y} \in \Re^{N_2}$, the ODE*

$$\frac{d\vec{x}}{dt} = F(\vec{x}, \vec{y}) \tag{4.11}$$

has a globally asymptotically stable equilibrium point which is a function of $\vec{y}$ and will be denoted by $\Omega(\vec{y})$, where $\Omega : \Re^{N_2} \to \Re^{N_1}$. Further, the function $\Omega(\vec{y})$ has to be Lipschitz continuous in $\vec{y}$.

Condition 4b. *ODE condition: The following ODE*

$$\frac{d\vec{y}}{dt} = G(\Omega(\vec{y}), \vec{y}) \tag{4.12}$$

has a globally asymptotically stable equilibrium $\vec{y}_*$.

Condition 5. *Boundedness of iterates: The iterates $\vec{X}^k$ and $\vec{Y}^k$ remain bounded with probability 1.*

Condition 6. *ITS Condition: The step sizes satisfy Condition 6 (ITS) of Theorem 1.*

Condition 7. *EDU Condition: The updating of all components of the X- and the Y-iterates is evenly distributed as discussed in Condition 7 (EDU) of Theorem 1.*

Then, with probability 1, the sequence of iterates $\{\vec{X}^k, \vec{Y}^k\}_{k=1}^{\infty}$ converges to $(\Omega(\vec{y}_*), \vec{y}_*)$.

4.2.2 Convergence Properties of R-SMART

R-SMART employs the following Q-factor version of the Bellman optimality equation for average reward SMDPs:

$$Q(i,a) = \sum_{j=1}^{|\mathcal{S}|} p(i,a,j)\left[r(i,a,j) - \rho\bar{t}(i,a,j) + \eta\max_{b\in\mathcal{A}(j)} Q(j,b)\right], \tag{4.13}$$

where η is a scalar that satisfies $0 < \eta < 1$. For any given value of ρ, the equation above can be shown to have a unique solution. This is because it behaves like the Bellman equation for the discounted reward MDP with its immediate reward altered to $r(i,a,j) - \rho\bar{t}(i,a,j)$. The parameter η will be referred to as the contraction factor (CF) here, since when ρ is set to a fixed value, the transformation defined in the equation above can be shown to be *contractive*.

Note that the Bellman optimality equation for average reward SMDPs, however, carries $\eta = 1$ and $\rho = \rho^*$. Hence, the algorithm requires the following condition for delivering the optimal solution:

Assumption 1. *There exists a value $\overline{\eta}$ in the open interval $(0,1)$ such that for all $\eta \in (\overline{\eta}, 1)$, the unique solution of Eq. (4.13), obtained by setting $\rho = \rho^*$, delivers a policy $\hat{d}$ whose average reward equals ρ^*.*

For all (i,a) pairs,

$$F'(\vec{Q}^k, \rho^k)(i,a) = \sum_{j=1}^{|\mathcal{S}|} p(i,a,j) \left[r(i,a,j) - \rho^k \overline{t}(i,a,j) + \eta \max_{b \in A(j)} Q^k(j,b) \right].$$

Then, for all (i,a) pairs, $F(\vec{Q}^k, \rho^k)(i,a) = F'(\vec{Q}^k, \rho^k)(i,a) - Q^k(i,a)$. We further define the noise term as: For all (i,a) pairs:

$$w_1^k(i,a) = \left[r(i,a,\xi^k) - \rho^k t(i,a,\xi^k) + \eta \max_{b \in \mathcal{A}(\xi^k)} Q^k(\xi^k, b) \right] - F'(\vec{Q}^k, \rho^k)(i,a).$$

Then, like in the case of Q-Learning, we can write the updating transformation on the faster time scale in our algorithm, (3.2), as:

$$Q^{k+1}(i,a) = Q^k(i,a) + \alpha^k \left[F(\vec{Q}^k, \rho^k)(i,a) + w_1^k(i,a) \right],$$

which is of the same form as the updating scheme for the faster time scale defined for Theorem 2 (replace X^k by Q^k and l by (i,a)). Then, if we fix the value of ρ^k to some constant, $\breve{\rho}$, we can invoke the following ODE as in Theorem 2:

$$\frac{d\vec{q}}{dt} = F(\vec{q}, \breve{\rho}), \tag{4.14}$$

where $\vec{q}$ denotes the continuous-valued variable underlying the iterate Q.

We now define the functions underlying the iterate on the slower time scale. For all (i,a) pairs,

$$G\ (\vec{Q}^k, \rho^k)(i,a) = \sum_{j=1}^{|\mathcal{S}|} p(i,a,j)[TR^k/TT^k] - \rho^k;$$
$$G'\ (\vec{Q}^k, \rho^k)(i,a) = G(\vec{Q}^k, \rho^k)(i,a) + \rho^k;$$
$$w_2^k(i,a) = [TR^k/TT^k] - G'(\vec{Q}^k, \rho^k)(i,a);$$

the above allows us to express the update on the slower time scale in the algorithm, Eq. (3.3), as:

$$\rho^{k+1} = \rho^k + \beta^k I \left(a \in \arg\max_{u \in \mathcal{A}(i)} Q^k(i,u) \right) \left[G(\vec{Q}^k, \rho^k)(i,a) + w_2^k(i,a) \right].$$

Now consider Condition 4b' defined in the Appendix. The following additional assumption is needed for showing the convergence result below:

Assumption 2. *The derivative condition in Assumption 4b' of the Appendix holds for $G(.,.)$ defined above.*

We now present our convergence result.

Proposition 2. *Assume that the step sizes used in the algorithm satisfy Conditions 3 and 6 of Theorem 2 and that exploratory policies are used in the learning. Further, assume that Assumptions 1 and 2 hold. Then, with probability 1, the sequence of policies generated by the R-SMART algorithm converges to an optimal policy.*

Proof. We now need to evaluate the conditions of Theorem 2. Condition 1 results from the fact that (i) the partial derivative of $F(.,.)$ with Q^k is bounded and (ii) $G(.,.)$ is a linear function of ρ^k Conditions 3 and 6 are satisfied by appropriate step-size selection. The PE policies ensure that Conditions 2 and 7 are met. As usual, Conditions 4 and 5 need additional work.

Condition 5: We show Condition 5 via the following result.

Lemma 2. *The sequence* $\{\vec{Q}^k, \rho^k\}_{k=1}^{\infty}$ *remains bounded with probability 1.*

Proof. The boundedness of the slower timescale iterate is first shown by claiming:

$$|\rho^k| \leq \overline{\rho} \; for \; all \; k,$$

where $\overline{\rho}$, a positive finite scalar, is defined as:

$$\overline{\rho} = \max \left\{ \frac{\max_{i,j \in \mathcal{S}, a \in \mathcal{A}(i)} |r(i,a,j)|}{\min_{i,j \in \mathcal{S}, a \in \mathcal{A}(i)} t(i,a,j)}, \rho^1 \right\}.$$

We prove the claim for $k = 1$ as follows:

$$|\rho^2| \leq (1-\beta^1)|\rho^1| + \beta^1 |r(i,a,j)/t(i,a,j)| \leq (1-\beta^1)\overline{\rho} + \beta^1 \overline{\rho} = \overline{\rho}$$

Now assuming the claim when $k = K$, we have that $|\rho^K| \leq \overline{\rho}$. Then,

$$\begin{aligned}
|\rho^{K+1}| &\leq (1-\beta^K)|\rho^K| + \beta^K \left| \frac{TR^K}{TT^K} \right| \\
&\leq (1-\beta^K)|\rho^K| + \beta^K \frac{K \max_{i,j \in \mathcal{S}, a \in \mathcal{A}(i)} |r(i,a,j)|}{K \min_{i,j \in \mathcal{S}, a \in \mathcal{A}(i)} t(i,a,j)} \\
&= (1-\beta^K)|\rho^K| + \beta^K \frac{\max_{i,j \in \mathcal{S}, a \in \mathcal{A}(i)} |r(i,a,j)|}{\min_{i,j \in \mathcal{S}, a \in \mathcal{A}(i)} t(i,a,j)} \\
&\leq (1-\beta^K)\overline{\rho} + \beta^K \overline{\rho} = \overline{\rho}.
\end{aligned}$$

In the above, we have used:

$$|TR^K| \leq K \max_{i,j \in \mathcal{S}, a \in \mathcal{A}(i)} |r(i,a,j)|$$

and

$$1/TT^K \leq 1/(K \min_{i,j \in \mathcal{S}, a \in \mathcal{A}(i)} t(i,a,j)).$$

Using Lemma 1, one can show that the Q-factors in R-SMART also remain bounded. For this, in Lemma 1, one needs to redefine $r_{\max}$ as follows:

$$r_{\max} \equiv \max_{i,j \in \mathcal{S}, a \in \mathcal{A}(i), k} |r(i,a,j) - \rho^k t(i,a,j)|.$$

Since ρ^k has already been established to be finite above and $t(i,a,j)$ is also finite, $r_{\max}$ must be finite. □

Condition 4: When ρ^k is fixed to any scalar $\breve{\rho}$, $F'(.,\breve{\rho})$ can be shown to be contractive in a manner similar to that for Q-Learning (via Lemma 3 in the Appendix), and hence the ODE in Eq. (4.14) must have a globally asymptotically stable equilibrium. Further, the equilibrium solution is:

$$\sum_{j=1}^{|\mathcal{S}|} p(i,a,j) \left[r(i,a,j) - \breve{\rho} \overline{t}(i,a,j) + \eta \max_{b \in \mathcal{A}(j)} Q^k(j,b) \right] \forall (i,a),$$

which is clearly Lipschitz in $Q^k(.,.)$. We have thus shown that Condition 4a holds.

To show Condition 4b, consider Condition $4b'$ defined in the Appendix, by setting $y_* = \rho^*$, where ρ^* is the optimal average reward of the SMDP. Note that $N_2 = 1$ for this algorithm (part (i) of Condition $4b'$). Now, when the value of ρ^k in the faster iterate is fixed to ρ^*, the faster iterates will converge since the transformation $F'(., \rho^*)$ is contractive (as argued above in Condition 4a), and then under Assumption 1, the faster iterates will converge to a solution of the Bellman optimality equation denoted by $\vec{Q}^*$. Since the slower iterate updates only when a greedy policy is chosen, in the limit, the slower iterate must converge to the average reward of the policy contained in $\vec{Q}^*$, which must be optimal. Thus, the lockstep condition (part (ii) in Condition $4b'$) holds for $y_* = \rho^*$. The derivative condition (part (iii) of Condition $4b'$) is true by assumption (Assumption 2). Then, Condition $4b'$ holds. Then, Theorem 4 in the Appendix implies that Condition 4b in Theorem 2 must hold. Then, one has convergence of the algorithm. But since ρ converges to ρ^*, under Assumption 1, the Q-factors must converge to the optimal solution as well. □

Chapter 5

Conclusions

The tutorial presented above discussed how to solve an MDP or an SMDP via DP when the underlying transition probabilities are available and via RL when they are not, through trials either in a simulator of the real-world system or in the real-world system itself. It needs to be emphasized that bypassing the transition probabilities of state transitions is the most attractive feature of RL. RL is currently of great interest in the subject of artificial intelligence and is being actively researched in academia and industry alike.

The approach discussed above is called the *lookup-table* or *tabular* approach in which the values for *all* the state-action pairs are stored explicitly (separately). It works for medium-sized problems. We did *not* discuss the solution techniques for large-scale problems with RL, as that requires what is referred to as *function approximation*, which was beyond the scope of this tutorial.

For large-scale problems, clearly, it is not possible to store all the values explicitly, because of the large number of such values. Instead, one must use approximate these values via functions such that only a few scalars need be stored; on demand, these scalars can generate the Q-factor for any state-action pair. This is essentially approach what is called *function approximation*, which often requires the backpropagation neural network algorithm [26].

I have worked in the area of RL for more than two decades now. I have also taught a graduate-level course on MDPs for several years. At conferences that I attend, I often meet researchers, including graduate students, dabbling with RL, who ask me questions about RL. Based on my experiences, I have the following friendly advice:

1. **Q-Learning and Related Algorithms**:
    - Q-Learning often fails when the step sizes used are *not* appropriate. Other tuning parameters of importance in RL include the exploration rates and the number of iterations for which the algorithm is run. If a neural network is used, it adds yet another layer of tuning parameters. A wide range of step sizes and tuning parameters should be experimented with before concluding that the approach is ineffective.
    - When using the average reward, one can compare the average reward generated by the policy delivered by the algorithm to that of competing algorithms. e.g., problem-specific heuristics. This allows one to tune the step sizes to improve the RL algorithm's behavior. In case of discounted reward, this author has seen the average reward being used for comparison, but technically this is not an appropriate and makes sense only heuristically, unless the discount factor is close to 1.
    - If one sets the discount factor, λ, very close to one in a discounted reward MDP (or the discount rate γ near zero in a discounted reward SMDP), the

algorithm generally slows down significantly. However, these cases really represent **average-reward** problems and are best solved via R-SMART or other average-reward algorithms. Also, it is critical to recognize that a discount factor in the Bellman equation really converts an infinite stage problem into one in which only a finite number of stages become relevant. Therefore, an average reward metric, where all stages are considered, may often be more appropriate for such problems. To gain such foundational insights on the Bellman equation, you can read Chap. 6 on dynamic programming of Gosavi [11], or consult the more advanced treatment in the two classics on dynamic programming: Puterman [22] and vol II of Bertsekas [3].

2. **The Linear Programming Viewpoint:** While it is certainly possible to formulate an average or discounted reward MDP (and SMDP) as a linear program (LP), much of reinforcement learning does *not* originate from LP methods. Instead, RL is rooted in dynamic programming. It is advisable to avoid initially viewing MDPs through the lens of LPs, as this approach provides limited structural insights into most RL algorithms and can be misleading. Moreover, certain formulations, such as a **variance-penalized MDP**, cannot be represented as an LP; it must instead be expressed as a **quadratic program** (QP). To gain a deeper understanding of the foundational principles of RL, it is more advantageous to focus on DP-based frameworks by consulting the references mentioned earlier.

3. **Analogies from Physics**: While analogies from physics are often helpful in explaining basic ideas, their use in RL should be ideally avoided by the beginner, as they provide little intuition or mathematical value. In fact, using such analogies can be often misleading. Cost is often referred to as energy in some literature, but note that in operations research, cost is the negative of revenues. Hence, maximization of the revenues (also called rewards) is equivalent to minimization of costs, and therefore minimization has no specific appeal unlike in physics where one often seeks to minimize energy. Another example is referring to the Bellman equation for a given policy (i.e., Eqn. (1.7)) as the *Poisson* equation. This is not even mathematically accurate, as the former is a simple linear equation while the latter involves a Laplacian! Similarly, some literature refers to the steady-state probabilities as *invariant* probabilities, borrowing terminology from the invariance equation of physics. However, the steady-state quantities have a well-understood meaning in operations research, **a subject unrelated to physics**, and they do *not* correspond to the invariant quantities of physics.

My understanding of this topic has been shaped by numerous books and hands-on experimentation with RL. The first book I studied in depth as a graduate student was Puterman [22], which has an in-depth discussion on MDPs and SMDPs, including a variety of objective functions. I strongly recommend this book to you. Subsequently, I explored other foundational texts, including Bertsekas and Tsitsiklis [4], Sutton and Barto [24], and Bertsekas [3], all of which I also recommend. Ultimately, reinforcement learning is a computational science, and the best way to learn it is through practical implementation, i.e., writing computer programs in your preferred language to solve real-world problems.

Bibliography

[1] J. Abounadi, D.P. Bertsekas, and V.S. Borkar. Learning algorithms for Markov decision processes with average cost. *SIAM Journal on Control and Optimization*, 40 (3): 681–698, 2001.

[2] A.G. Barto, R.S. Sutton, C.W. Anderson, Neuronlike elements that can solve difficult learning control problems. *IEEE Trans. Syst. Man Cybern.* 13, 835–846, 1983

[3] D. P. Bertsekas. *Dynamic Programming and Optimal Control.* Second edition. Athena, 2000.

[4] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming.* Athena, 1996.

[5] V.S. Borkar. Asynchronous stochastic approximations. *SIAM Journal on Control and Optimization*, 36(3), 840–851, 1998.

[6] V. S. Borkar. *Stochastic Approximation: A Dynamical Systems Viewpoint.* Hindusthan Book Agency, 2008.

[7] S. J. Bradtke and M. Duff. Reinforcement Learning Methods for Continuous-Time Markov Decision Problems. In *Advances in Neural Information Processing Systems* 7. MIT Press, 1995.

[8] C. Darken, J. Chang, and J. Moody, Learning rate schedules for faster stochastic gradient search, in *Neural Networks for Signal Processing 2 – Proceedings of the 1992 IEEE Workshop*, ed. by D.A. White, D.A. Sofge, IEEE, Piscataway, 1992.

[9] A. Encapera, A. Gosavi, S.L. Murray, Total productive maintenance of make-to-stock production-inventory systems via artificial-intelligence-based iSMART, *International Journal of Systems Science: Operations and Logistics*, 8(2), 154–166, 2021.

[10] A. Gosavi. Boundedness of iterates in Q-learning. *Systems & Control Letters*, 55(4), 347-349, 2006.

[11] A. Gosavi. *Simulation-Based Optimization: Parametric Optimization Techniques and Reinforcement Learning*, Second Edition, Springer, 2015.

[12] A. Gosavi. Target-Sensitive Control of Markov and Semi-Markov Processes. *International Journal of Control, Automation, and Systems*, 9(5):1-11, 2011.

[13] A. Gosavi. Reinforcement Learning for Long-Run Average Cost. *European Journal of Operational Research.* Vol 155, pp. 654-674, 2004.

[14] A. Gosavi. On Step Sizes, Stochastic Shortest Paths, and Survival Probabilities in Reinforcement Learning, *Conference Proceedings of the Winter Simulation Conference*, 2009.

[15] A. Gosavi. Reinforcement Learning: A Tutorial Survey and Recent Advances. *INFORMS Journal on Computing.* Vol 21(2), pp. 178–192, 2009. Available at:

[16] A. Gosavi, How to rein in the volatile actor: A new bounded perspective, *Complex Adaptive Systems Conference Proceedings*, 36, 500–507, 2014

[17] A. Gosavi and V.K. Le. Maintenance optimization in a digital twin for Industry 4.0, *Annals of Operations Research*, 340 (1), 245–269, 2024.

[18] R. Howard, *Dynamic Programming and Markov Processes* MIT, Cambridge, MA, 1960.

[19] V.R. Konda, V.S. Borkar, Actor-critic type learning algorithms for Markov decision processes. *SIAM J. Control Optim*, 38(1), 94–123, 1999.

[20] R.J. Lawhead and A. Gosavi. A bounded actor-critic reinforcement learning algorithm applied to airline revenue management, *Engineering Applications of Artificial Intelligence*, 82, 252–262, 2019.

[21] B.T. Polyak, *Introduction to Optimization* (Optimization Software, New York, 1987.

[22] M. L. Puterman. *Markov Decision Processes.* Wiley, 1994.

[23] D. B. Rokhlin, Robbins–Monro conditions for persistent exploration learning strategies. In *Modern Methods in Operator Theory and Harmonic Analysis: OTHA 2018, Rostov-on-Don, Russia, April 22-27, Selected, Revised and Extended Contributions* 8, pp. 237-247. Springer International Publishing, 2019.

[24] R. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction.* Second Edition, MIT Press, 2018.

[25] C. J. Watkins. *Learning from Delayed Rewards.* Ph.D. thesis, Kings College, Cambridge, UK, May 1989.

[26] P. J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis of Behavioral Sciences.*. Ph.D. thesis, Harvard University, 1974.

Chapter 6

Appendix

6.1 Contraction Mapping and Asymptotically Stable Equilibrium

Theorem 3. *Consider the transformation $F(.)$ defined for the stochastic approximation scheme. Define $F'(.)$, a continuous function from $\Re^N$ to $\Re^N$, such that for all integer values of k,*

$$F(\vec{X}^k) = F'(\vec{X}^k) - \vec{X}^k \text{ where } \vec{X}^k \in \Re^N.$$

If $F'(.)$ is a contractive transformation with respect to a weighted max *norm, the ODE $\frac{d\vec{x}}{dt} = F(\vec{x})$ has a unique globally asymptotically stable equilibrium point.*

The proof can be found on pp 127 of [6].

6.2 Contraction of Q-Learning Mapping

Lemma 3. *The mapping defined via Eqn. (4.2) is a contraction mapping.*

Proof. Consider two vectors $\vec{Q}_1^k$ and $\vec{Q}_2^k$ in $\Re^N$ From the definition of $F'(.)$ above, it follows that:

$$F'(\vec{Q}_1^k)(i,a) - F'(\vec{Q}_2^k)(i,a) = \lambda \sum_{j=1}^{|\mathcal{S}|} p(i,a,j) \left[\max_{b \in \mathcal{A}(j)} Q_1^k(j,b) - \max_{b \in \mathcal{A}(j)} Q_2^k(j,b) \right].$$

From this, we can write that for any (i,a) pair:

$$\begin{aligned}
|F'(\vec{Q}_1^k)(i,a) - F'(\vec{Q}_2^k)(i,a)| &\leq \lambda \sum_{j=1}^{|\mathcal{S}|} p(i,a,j) \left| \max_{b \in \mathcal{A}(j)} Q_1^k(j,b) - \max_{b \in \mathcal{A}(j)} Q_2^k(j,b) \right| \\
&\quad \text{(from triangle inequality);} \\
&\leq \lambda \sum_{j=1}^{|\mathcal{S}|} p(i,a,j) \max_{b \in \mathcal{A}(j)} |Q_1^k(j,b) - Q_2^k(j,b)| \\
&\leq \lambda \sum_{j=1}^{|\mathcal{S}|} p(i,a,j) \max_{j \in \mathcal{S}, b \in \mathcal{A}(j)} |Q_1^k(j,b) - Q_2^k(j,b)| \\
&= \lambda \sum_{j=1}^{|\mathcal{S}|} p(i,a,j) ||\vec{Q}_1^k - \vec{Q}_2^k||_\infty \\
&= \lambda ||\vec{Q}_1^k - \vec{Q}_2^k||_\infty \sum_{j=1}^{|\mathcal{S}|} p(i,a,j) = \lambda ||\vec{Q}_1^k - \vec{Q}_2^k||_\infty \cdot 1.
\end{aligned}$$

Thus, for any (i,a) : $|F'(\vec{Q}_1^k)(i,a) - F'(\vec{Q}_2^k)(i,a)| \leq \lambda ||\vec{Q}_1^k - \vec{Q}_2^k||_\infty$. Since the above holds for all values of (i,a), it also holds for the values that maximize the left hand side of the above. Therefore

$$||F'\vec{Q}_1^k - F'\vec{Q}_2^k||_\infty \leq \lambda ||\vec{Q}_1^k - \vec{Q}_2^k||_\infty.$$

Since $\lambda < 1, F'(.)$ is contractive with respect to the max norm. □

6.3 Lockstep and Derivative Conditions

Now consider the following special conditions collectively referred to as $4b'$:

6.3.1 Condition $4b'$

(i) $N_2 = 1$, i.e., there is a single iterate, $Y^k \in \Re$, on the slower time scale.

(ii) **(Lockstep Condition)** There exists a unique value $y_* \in \Re$ such that if one sets $Y^k = y_*$ in the update on the faster timescale for every k, i.e., the update of the iterate $\vec{X}^k$, then, with probability 1,

$$\lim_{k\to\infty} Y^k = y_*.$$

(iii) **(Derivative Condition)** In every iteration, the partial derivative of $G(.,.)$ with respect to the slower iterate's value is bounded and is strictly negative, i.e., for all k,

$$\frac{\partial G(\vec{X}^k, Y^k)}{\partial Y^k} < 0 \text{ and } \left|\frac{\partial G(\vec{X}^k, Y^k)}{\partial Y^k}\right| < \infty, \text{ where } Y^k \in \Re.$$

Note that the boundedness of the derivative is already ensured by the Lipschitz continuity of the function $G(.,.)$; however, this is restated to clarify that the derivative cannot equal negative infinity.

6.3.2 Condition $4b'$ implies Condition $4b$

In the following result, Condition $4b'$ is used to deliver Condition $4b$:

Theorem 4. *Consider the two-time-scale asynchronous algorithm defined above with a single iterate on the slower timescale. Replace Condition 4b in Theorem 2 by Condition $4b'$. Then y_* is the globally asymptotically stable equilibrium for the slower ODE in Eq. (4.12).*

The proof is from [12] and is based on ideas from a result in [1].

Proof. The iterate index l_2 will be dropped from the notation of the step size for the unique slower iterate. For the proof, the following limits, showed in Polyak [21], are needed.

For any non-negative sequence $\{\beta^n\}_{n=1}^\infty$ satisfying Condition 3 of Theorem 2, i.e., $\sum_{n=1}^\infty \beta^n = \infty$, and for any finite integer K,

$$\lim_{k\to\infty} \prod_{n=K}^{k+1} (1-\beta^n) = 0 \text{ for all } k > K; \tag{6.1}$$

further, for any sequence $\{\delta^k\}_{k=1}^{\infty}$ satisfying $\lim_{k\to\infty}\delta^k = 0$, one has that

$$\lim_{k\to\infty}\sum_{n=K}^{k+1}\left(\prod_{m=n+1}^{k}(1-\beta^m)\right)\beta^n\delta^n = 0 \text{ for all } k > K. \tag{6.2}$$

Set up a sequence $\{\Delta^k\}_{k=1}^{\infty}$ where $\Delta^k = Y^k - y_*$, so the goal becomes to show that $\Delta^k \to 0$. The road-map for the proof is as follows. First another sequence will be defined $\{\delta^k\}_{k=1}^{\infty}$ and Δ^k will be expressed in terms of δ^k. Then, upper and lower bounds on the partial derivative of $G(\vec{X}^k, Y^k)$ with respect to Y^k will be developed. These bounds, the sequence $\{\delta^k\}_{k=1}^{\infty}$, and the limits shown in (6.1)–(6.2) will together be leveraged to show that $\Delta^k \to 0$.

For a given value Y^k of the slower iterate, Condition 4a ensures that with probability 1, the sequence $\{\vec{X}^k\}_{k=1}^{\infty}$ will converge to a globally asymptotically stable critical point of the ODE in (4.11). This critical point will be denoted by $x_*(Y^k)$, where this point is a function of Y^k.

$$\text{Let } \delta^k = G\ (\vec{X}^k, Y^k) - G(x_*(Y^k), Y^k). \tag{6.3}$$

Condition 4a ensures that $\delta^k \to 0$, since $\vec{X}^k \to x_*(Y^k)$.

We now express Δ^k in terms of $G(.,.)$ and δ^k. From the definition of δ^k above in Eq. (6.3) and the fact that the update of Y^k can be expressed as follows:

$$Y^{k+1} = Y^k + \beta^k\ (G\ (\vec{X}^k, Y^k)), \tag{6.4}$$

$$\text{one has that } \Delta^{k+1} = \Delta^k + \beta^k G(x_*(Y^k), Y^k) + \beta^k\delta^k. \tag{6.5}$$

Now, the derivative condition, i.e., Condition 4b'(iii), implies that there exist negative, upper and lower bounds on the derivative, i.e., there exist $C_1, C_2 \in \Re$ where $0 < C_1 \leq C_2$ such that:

$$-C_2(Y_1 - Y_2) \leq G(x_*(Y_1), Y_1) - G(x_*(Y_2), Y_2) \leq -C_1(Y_1 - Y_2) \tag{6.6}$$

for any $Y_1, Y_2 \in \Re$ if $Y_1 > Y_2$. If $Y_2 > Y_1$, we will have the following inequality:

$$-C_2(Y_1 - Y_2) \geq G(x_*(Y_1), Y_1) - G(x_*(Y_2), Y_2) \geq -C_1(Y_1 - Y_2). \tag{6.7}$$

We first consider the case $Y_1 > Y_2$. Now, the lockstep condition (Condition 4b'(iii)) implies that if the faster iterates in the algorithm are updated using $Y^k \equiv y_*$ in $F(\vec{X}^k, Y^k)$, then $Y^k \to y_*$. This implies from (6.4) that $G(x_*(y_*), y_*) = 0$. Thus, if $Y_2 = y_*$ and $Y_1 = Y^k$, inequality (6.6) will lead to:

$$-C_2\Delta^k \leq G\ (x_*(Y^k), \quad Y^k) \leq -C_1\Delta^k.$$

Because $\beta^k > 0$, the above leads to:

$$-C_2\Delta^k\beta^k \leq G\ (x_*(Y^k), Y^k)\beta^k \leq -C_1\Delta^k\beta^k.$$

The above combined with (6.5) leads to:

$$(1 - C_2\beta^k)\Delta^k + \beta^k\delta^k \leq \Delta^{k+1} \leq (1 - C_1\beta^k)\Delta^k + \beta^k\delta^k. \tag{6.8}$$

Since, the above is true for any finite integral value of k, we have that for $k = K$ and $k = K + 1$,

$$(1 - C_2\beta^K)\Delta^K + \beta^K\delta^K \leq \Delta^{K+1}; \tag{6.9}$$

$$(1 - C_2\beta^{K+1})\Delta^{K+1} + \beta^{K+1}\delta^{K+1} \leq \Delta^{K+2}. \tag{6.10}$$

Multiplying both sides of (6.9) by $(1 - C_2\beta^{K+1})$ and adding $\beta^{K+1}\delta^{K+1}$ to both sides, we have:

$$(1 - C_2\beta^K)(1 - C_2\beta^{K+1})\Delta^K + (1 - C_2\beta^{K+1})\beta^K\delta^K + \beta^{K+1}\delta^{K+1} \leq$$
$$(1 - C_2\beta^{K+1})\Delta^{K+1} + \beta^{K+1}\delta^{K+1}$$

The above in combination with (6.10) leads to:

$$(1 - C_2\beta^K)(1 - C_2\beta^{K+1})\Delta^K + (1 - C_2\beta^{K+1})\beta^K\delta^K + \beta^{K+1}\delta^{K+1} \leq \Delta^{K+2}.$$

Using similar arguments on the inequality on the other side of Δ^{k+1} of (6.8), we have that

$$(1 - C_2\beta^K)(1 - C_2\beta^{K+1})\Delta^K + (1 - C_2\beta^{K+1})\beta^K\delta^K + \beta^{K+1}\delta^{K+1}$$
$$\leq \Delta^{K+2} \leq (1 - C_2\beta^K)(1 - C_2\beta^{K+1})\Delta^K + (1 - C_2\beta^{K+1})\beta^K\delta^K + \beta^{K+1}\delta^{K+1}$$

In this style, we can also show the above result when the sandwiched term is $\Delta^{K+3}, \Delta^{K+4} \ldots$. In general, then, for any $\overline{K} > K$, we obtain:

$$\prod_{n=K}^{\overline{K}+1}(1 - C_2\beta^n)\Delta^K + \sum_{n=K}^{\overline{K}}\left(\prod_{m=n+1}^{\overline{K}}(1 - C_2\beta^m)\right)\beta^n\delta^n \leq \Delta^{\overline{K}+1}$$
$$\leq \prod_{n=K}^{\overline{K}+1}(1 - C_1\beta^n)\Delta^K + \sum_{n=K}^{\overline{K}}\left(\prod_{m=n+1}^{\overline{K}}(1 - C_1\beta^m)\right)\beta^n\delta^n.$$

We now take the limits as $\overline{K} \to \infty$ on the above. Then, using (6.1) and (6.2), $0 \leq \lim_{\overline{K}\to\infty}\Delta^{\overline{K}+1} \leq 0$. Then, the sandwich theorem implies that: $\Delta^{\overline{K}+1} \to 0$. Identical arguments can now be repeated for the case $Y_2 > Y_1$, i.e., for (6.7), to obtain the same conclusion. □