# CHAPTER 3

# INTRODUCTION TO MARKOV DECISION PROCESSES

There is a very elegant theory for solving stochastic, dynamic programs if we are willing to live within some fairly limiting assumptions. Assume that we have a discrete state space $\mathcal{S} = (1, 2, \ldots, |\mathcal{S}|)$, where $\mathcal{S}$ is small enough to enumerate. Next assume that there is a relatively small set of decisions or actions, which we denote by $a \in \mathcal{A}$, and that we can compute a cost (if minimizing) or contribution (if maximizing) given by $C(s, a)$. Finally, assume that we are given a transition matrix $p_t(S_{t+1}|S_t, a_t)$ which gives the probability that if we are in state $S_t$ (at time $t$) and take action $a_t$, then we will next be in state $S_{t+1}$.

From time to time, we are going to switch gears to consider problems where the decision is a vector. When this happens, we will use $x$ as our decision variable. However, there are many applications where the number of actions is discrete and small, and there are many algorithms that are specifically designed for small action spaces. In particular, the material in this chapter is designed for small action spaces, and as a result we use $a$ for action throughout.

There are many problems where states are continuous, or the state variable is a vector producing a state space that is far too large to enumerate. In addition, computing the one-step transition matrix $p_t(S_{t+1}|S_t, a_t)$ can also be difficult or impossible to compute. So why cover material that is widely acknowledged to work only on small or highly specialized problems? First, some problems have small state and action spaces and can be solved with these techniques. Second, the theory of Markov decision processes can be used to identify structural properties that can dramatically simplify computational algorithms. But far more importantly, this material provides the intellectual foundation for the types of algorithms that we present in later chapters. Using the framework in this chapter, we

can prove very powerful results that will provide a guiding hand as we step into richer and more complex problems in many real-world settings. Furthermore, the behavior of these algorithms provide important insights that guide the behavior of algorithms for more general problems.

There is a rich and elegant theory behind Markov decision processes. Even if the algorithms have limited application, the ideas behind these algorithms, which enjoy a rich history, represent the fundamental underpinnings of most of the algorithms in the remainder of this book. As with most of the chapters in the book, the body of this chapter focuses on the algorithms, and the convergence proofs have been deferred to the "Why does it work" section (section 3.10). The intent is to allow the presentation of results to flow more naturally, but serious students of dynamic programming are encouraged to delve into these proofs, which are quite elegant. This is partly to develop a deeper appreciation of the properties of the problem as well as to develop an understanding of the proof techniques that are used in this field.

## 3.1 THE OPTIMALITY EQUATIONS

In the last chapter, we illustrated a number of stochastic applications that involve solving the following objective function

$$\max_{\pi} \mathbb{E}^{\pi} \left\{ \sum_{t=0}^{T} \gamma^t C_t^{\pi}(S_t, A_t^{\pi}(S_t)) \right\}. \tag{3.1}$$

In chapter 2, we sometimes wrote our objective function using $\mathbb{E}\ldots$, and in other cases we wrote it as $\mathbb{E}^{\pi}\ldots$. We defer until chapter 5 a more complete discussion of this choice, but in a nutshell it has to do whether the decisions we make may influence the information we observe. If this is the case, we use $\mathbb{E}^{\pi}$, which is the more general form.

For most problems, solving equation (3.1) is computationally intractable, but it provides the basis for identifying the properties of optimal solutions and finding and comparing "good" solutions.

### 3.1.1 Bellman's equations

With a little thought, we realize that we do not have to solve this entire problem at once. Assume that we are solving a deterministic shortest path problem where $S_t$ is the index of the node in the network where we have to make a decision. If we are in state $S_t = i$ (that is, we are at node $i$ in our network) and take action $a_t = j$ (that is, we wish to traverse the link from $i$ to $j$), our transition function will tell us that we are going to land in some state $S_{t+1} = S^M(S_t, a_t)$ (in this case, node $j$). What if we had a function $V_{t+1}(S_{t+1})$ that told us the value of being in state $S_{t+1}$ (giving us the value of the path from node $j$ to the destination)? We could evaluate each possible action $a_t$ and simply choose the action $a_t$ that has the largest one-period contribution, $C_t(S_t, a_t)$, plus the value of landing in state $S_{t+1} = S^M(S_t, a_t)$ which we represent using $V_{t+1}(S_{t+1})$. Since this value represents the money we receive one time period in the future, we might discount this by a factor $\gamma$. In other words, we have to solve

$$a_t^*(S_t) = \arg\max_{a_t \in \mathcal{A}_t} \big( C_t(S_t, a_t) + \gamma V_{t+1}(S_{t+1}) \big),$$

where "$\arg\max$" means that we want to choose the action $a_t$ that maximizes the expression in parentheses. We also note that $S_{t+1}$ is a function of $S_t$ and $a_t$, meaning that we could write it as $S_{t+1}(S_t, a_t)$. Both forms are fine. It is common to write $S_{t+1}$ by itself, but the dependence on $S_t$ and $a_t$ needs to be understood.

The value of being in state $S_t$ is the value of using the optimal decision $a_t^*(S_t)$. That is

$$\begin{aligned} V_t(S_t) &= \max_{a_t \in \mathcal{A}_t} \big(C_t(S_t, a_t) + \gamma V_{t+1}(S_{t+1}(S_t, a_t))\big) \\ &= C_t(S_t, a_t^*(S_t)) + \gamma V_{t+1}(S_{t+1}(S_t, a_t^*(S_t))). \end{aligned} \tag{3.2}$$

Equation (3.2) is the optimality equation for deterministic problems.

When we are solving stochastic problems, we have to model the fact that new information becomes available after we make the decision $a_t$. The result can be uncertainty in both the contribution earned, and in the determination of the next state we visit, $S_{t+1}$. For example, consider the problem of managing oil inventories for a refinery. Let the state $S_t$ be the inventory in thousands of barrels of oil at time $t$ (we require $S_t$ to be integer). Let $a_t$ be the amount of oil ordered at time $t$ that will be available for use between $t$ and $t+1$, and let $\hat{D}_{t+1}$ be the demand for oil between $t$ and $t+1$. The state variable is governed by the simple inventory equation

$$S_{t+1}(S_t, a_t, \hat{D}_{t+1}) = \max\{0, S_t + a_t - \hat{D}_{t+1}\}.$$

We have written the state $S_{t+1}$ using $S_{t+1}(S_t, a_t)$ to express the dependence on $S_t$ and $a_t$, but it is common to simply write $S_{t+1}$ and let the dependence on $S_t$ and $a_t$ be implicit. Since $\hat{D}_{t+1}$ is random at time $t$ when we have to choose $a_t$, we do not know $S_{t+1}$. But if we know the probability distribution of the demand $\hat{D}$, we can work out the probability that $S_{t+1}$ will take on a particular value. If $\mathbb{P}^D(d) = \mathbb{P}[\hat{D} = d]$ is our probability distribution, then we can find the probability distribution for $S_{t+1}$ using

$$Prob(S_{t+1} = s') = \begin{cases} 0 & \text{if } s' > S_t + a_t, \\ \mathbb{P}^D(S_t + a_t - s') & \text{if } 0 < s' \leq S_t + a_t, \\ \sum_{d=S_t+a_t}^{\infty} \mathbb{P}^D(d) & \text{if } s' = 0. \end{cases}$$

These probabilities depend on $S_t$ and $a_t$, so we write the probability distribution as

$$\mathbb{P}(S_{t+1}|S_t, a_t) = \text{The probability of } S_{t+1} \text{ given } S_t \text{ and } a_t.$$

We can then modify the deterministic optimality equation in (3.2) by simply adding an expectation, giving us

$$V_t(S_t) = \max_{a_t \in \mathcal{A}_t} \big(C_t(S_t, a_t) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(S_{t+1} = s'|S_t, a_t) V_{t+1}(s')\big). \tag{3.3}$$

We refer to this as the *standard form* of Bellman's equations, since this is the version that is used by virtually every textbook on stochastic, dynamic programming. An equivalent form that is more natural for approximate dynamic programming is to write

$$V_t(S_t) = \max_{a_t \in \mathcal{A}_t} \big(C_t(S_t, a_t) + \gamma \mathbb{E}\{V_{t+1}(S_{t+1}(S_t, a_t, W_{t+1}))|S_t\}\big), \tag{3.4}$$

where we simply use an expectation instead of summing over probabilities. We refer to this equation as the *expectation form* of Bellman's equation. This version forms the basis for our algorithmic work in later chapters.

**Remark:** Equation (3.4) is often written in the slightly more compact form

$$V_t(S_t) = \max_{a_t \in \mathcal{A}_t} \big(C_t(S_t, a_t) + \gamma \mathbb{E}\{V_{t+1}(S_{t+1})|S_t\}\big), \tag{3.5}$$

where the functional relationship $S_{t+1} = S^M(S_t, a_t, W_{t+1})$ is implicit. At this point, however, we have to deal with some subtleties of mathematical notation. In equation (3.4) we have captured the *functional dependence* of $S_{t+1}$ on $S_t$ and $a_t$, while capturing the *conditional dependence* of $S_{t+1}$ (more specifically $W_{t+1}$) on the state $S_t$.

$$V_t(S_t) = \max_{a_t \in \mathcal{A}_t} \big(C_t(S_t, a_t) + \gamma \mathbb{E}\{V_{t+1}(S_{t+1})|S_t, a_t\}\big) \tag{3.6}$$

to capture the fact that $S_{t+1}$ may depend on $a_t$. However, it is important to understand whether $S_{t+1}$ is *functionally dependent on* $S_t$ and $a_t$, or if the distribution of $S_{t+1}$ is *probabilistically dependent on* $S_t$ and $a_t$. To see the difference, imagine that we have a problem where $W_{t+1}$ is the wind or some exogenous process whose outcomes are independent of $S_t$ or $a_t$. Then it is perfectly valid to write

$$V_t(S_t) = \max_{a_t \in \mathcal{A}_t} \big(C_t(S_t, a_t) + \gamma \mathbb{E} V_{t+1}(S_{t+1} = S^M(S_t, a_t, W_{t+1}))\big),$$

where we are explicitly capturing the functional dependence of $S_{t+1}$ on $S_t$ and $a_t$, but where the expectation is not conditioned on anything, because the distribution of $W_{t+1}$ does not depend on $S_t$ or $a_t$. However, there are problems where $W_{t+1}$ depends on the state, such as the random perturbations of a robot which depends on how close the robot is to a boundary. In this case, $S_{t+1}$ depends functionally on $S_t$, $a_t$ and $W_{t+1}$, but the *distribution* of $W_{t+1}$ also depends on $S_t$, in which case the expectation needs to be a conditional expectation. Then, there are problems where the distribution of $W_{t+1}$ depends on both $S_t$ and $a_t$, such as the random changes in a stock which might be depressed if a mutual fund is holding large quantities ($S_t$) and begins selling in large amounts ($a_t$). In this case, the proper interpretation of equation (3.6) is that we are computing the conditional expectation over $W_{t+1}$ which now depends on both the state and action.

The standard form of Bellman's equation (3.3) has been popular in the research community since it lends itself to elegant algebraic manipulation when we assume we know the transition matrix. It is common to write it in a more compact form. Recall that a policy $\pi$ is a rule that specifies the action $a_t$ given the state $S_t$. In this chapter, it is easiest if we always think of a policy in terms of a rule "when we are in state $s$ we take action $a$." This is a form of "lookup-table" representation of a policy that is very clumsy for most real problems, but it will serve our purposes here. The probability that we transition from state $S_t = s$ to $S_{t+1} = s'$ can be written as

$$p_{ss'}(a) = \mathbb{P}(S_{t+1} = s'|S_t = s, a_t = a).$$

We would say that "$p_{ss'}(a)$ is the probability that we end up in state $s'$ if we start in state $s$ at time $t$ when we are taking action $a$." Now assume that we have a function $A_t^\pi(s)$ that determines the action $a$ we should take when in state $s$. It is common to write the transition probability $p_{ss'}(a)$ in the form

$$p_{ss'}^\pi = \mathbb{P}(S_{t+1} = s'|S_t = s, A_t^\pi(s) = a).$$

We can now write this in matrix form

$$P_t^\pi = \text{The one-step transition matrix under policy } \pi,$$

where $p_{ss'}^\pi$ is the element in row $s$ and column $s'$. There is a different matrix $P^\pi$ for each policy (decision rule) $\pi$.

Now let $c_t^\pi$ be a column vector with element $c_t^\pi(s) = C_t(s, A_t^\pi(s))$, and let $v_{t+1}$ be a column vector with element $V_{t+1}(s)$. Then (3.3) is equivalent to

$$\begin{bmatrix} \vdots \\ v_t(s) \\ \vdots \end{bmatrix} = \max_\pi \left( \begin{bmatrix} \vdots \\ c_t^\pi(s) \\ \vdots \end{bmatrix} + \gamma \begin{bmatrix} \ddots & & \\ & p_{ss'}^\pi & \\ & & \ddots \end{bmatrix} \begin{bmatrix} \vdots \\ v_{t+1}(s') \\ \vdots \end{bmatrix} \right). \quad (3.7)$$

where the maximization is performed for each element (state) in the vector. In matrix/vector form, equation (3.7) can be written

$$v_t = \max_\pi \left( c_t^\pi + \gamma P_t^\pi v_{t+1} \right). \quad (3.8)$$

Here, we maximize over policies because we want to find the best action for *each* state. The vector $v_t$ is known widely as the *value function* (the value of being in each state). In control theory, it is known as the *cost-to-go function*, where it is typically denoted as $J$.

Equation (3.8) can be solved by finding $a_t$ for each state $s$. The result is a decision vector $a_t^* = (a_t^*(s))_{s \in \mathcal{S}}$, which is equivalent to determining the best policy. This is easiest to envision when $a_t$ is a scalar (how much to buy, whether to sell), but in many applications $a_t(s)$ is itself a vector. For example, assume our problem is to assign individual programmers to different programming tasks, where our state $S_t$ captures the availability of programmers and the different tasks that need to be completed. Of course, computing a vector $a_t$ for each state $S_t$ which is itself a vector is much easier to write than to implement.

It is very easy to lose sight of the relationship between Bellman's equation and the original objective function that we stated in equation (3.1). To bring this out, we begin by writing the expected profits using policy $\pi$ from time $t$ onward

$$F_t^\pi(S_t) = \mathbb{E} \left\{ \sum_{t'=t}^{T-1} C_{t'}(S_{t'}, A_{t'}^\pi(S_{t'})) + C_T(S_T) | S_t \right\}.$$

$F_t^\pi(S_t)$ is the expected total contribution if we are in state $S_t$ in time $t$, and follow policy $\pi$ from time $t$ onward. If $F_t^\pi(S_t)$ were easy to calculate, we would probably not need dynamic programming. Instead, it seems much more natural to calculate $V_t^\pi$ recursively using

$$V_t^\pi(S_t) = C_t(S_t, A_t^\pi(S_t)) + \mathbb{E} \left\{ V_{t+1}^\pi(S_{t+1}) | S_t \right\}.$$

It is not hard to show (by stepping backward in time) that

$$F_t^\pi(S_t) = V_t^\pi(S_t).$$

The proof, given in section 3.10.1, uses a proof by induction: assume it is true for $V_{t+1}^\pi$, and then show that it is true for $V_t^\pi$ (not surprisingly, inductive proofs are very popular in dynamic programming).

With this result in hand, we can then establish the following key result. Let $V_t(S_t)$ be a solution to equation (3.4) (or (3.3)). Then

$$\begin{aligned} F_t^* &= \max_{\pi \in \Pi} F_t^\pi(S_t) \\ &= V_t(S_t). \end{aligned} \quad (3.9)$$

Equation (3.9) establishes the equivalence between (a) the value of being in state $S_t$ and following the optimal policy and (b) the optimal value function at state $S_t$. While these are indeed equivalent, the equivalence is the result of a theorem (established in section 3.10.1). However, it is not unusual to find people who lose sight of the original objective function. Later, we have to solve these equations approximately, and we will need to use the original objective function to evaluate the quality of a solution.

### 3.1.2 Computing the transition matrix

It is very common in stochastic, dynamic programming (more precisely, Markov decision processes) to assume that the one-step transition matrix $P^\pi$ is given as data (remember that there is a different matrix for each policy $\pi$). In practice, we generally can assume we know the transition function $S^M(S_t, a_t, W_{t+1})$ from which we have to derive the one-step transition matrix.

Assume that the random information $W_{t+1}$ that arrives between $t$ and $t+1$ is independent of all prior information. Let $\Omega_{t+1}$ be the set of possible outcomes of $W_{t+1}$ (for simplicity, we assume that $\Omega_{t+1}$ is discrete), where $\mathbb{P}(W_{t+1} = \omega_{t+1})$ is the probability of outcome $\omega_{t+1} \in \Omega_{t+1}$. Also define the indicator function

$$1_{\{X\}} = \begin{cases} 1 & \text{if the statement "}X\text{" is true.} \\ 0 & \text{otherwise.} \end{cases}$$

Here, "$X$" represents a logical condition (such as, "is $S_t = 6$?"). We now observe that the one-step transition probability $\mathbb{P}_t(S_{t+1}|S_t, a_t)$ can be written

$$\begin{aligned} \mathbb{P}_t(S_{t+1}|S_t, a_t) &= \mathbb{E}1_{\{s'=S^M(S_t,a_t,W_{t+1})\}} \\ &= \sum_{\omega_{t+1}\in\Omega_{t+1}} \mathbb{P}(\omega_{t+1})1_{\{s'=S^M(S_t,a_t,\omega_{t+1})\}} \end{aligned}$$

So, finding the one-step transition matrix means that all we have to do is to sum over all possible outcomes of the information $W_{t+1}$ and add up the probabilities that take us from a particular state-action pair $(S_t, a_t)$ to a particular state $S_{t+1} = s'$. Sounds easy.

In some cases, this calculation is straightforward (consider our oil inventory example earlier in the section). But in other cases, this calculation is impossible. For example, $W_{t+1}$ might be a vector of prices or demands. In this case, the set of outcomes $\Omega_{t+1}$ can be much too large to enumerate. We can estimate the transition matrix statistically, but in later chapters (starting in chapter 4) we are going to avoid the need to compute the one-step transition matrix entirely. For the remainder of this chapter, we assume the one-step transition matrix is available.

### 3.1.3 Random contributions

In many applications, the one-period contribution function is a deterministic function of $S_t$ and $a_t$, and hence we routinely write the contribution as the deterministic function $C_t(S_t, a_t)$. However, this is not always the case. For example, a car traveling over a stochastic network may choose to traverse the link from node $i$ to node $j$, and only learn the cost of the movement after making the decision. For such cases, the contribution function

is random, and we might write it as

$\hat{C}_{t+1}(S_t, a_t, W_{t+1}) =$ The contribution received in period $t+1$ given the state $S_t$ and decision $a_t$, as well as the new information $W_{t+1}$ that arrives in period $t+1$.

In this case, we simply bring the expectation in front, giving us

$$V_t(S_t) = \max_{a_t} \mathbb{E}\left\{\hat{C}_{t+1}(S_t, a_t, W_{t+1}) + \gamma V_{t+1}(S_{t+1})|S_t\right\}. \tag{3.10}$$

Now let

$$C_t(S_t, a_t) = \mathbb{E}\{\hat{C}_{t+1}(S_t, a_t, W_{t+1})|S_t\}.$$

Thus, we may view $C_t(S_t, a_t)$ as the expected contribution given that we are in state $S_t$ and take action $a_t$.

### 3.1.4 Bellman's equation using operator notation*

The vector form of Bellman's equation in (3.8) can be written even more compactly using operator notation. Let $\mathcal{M}$ be the "max" (or "min") operator in (3.8) that can be viewed as acting on the vector $v_{t+1}$ to produce the vector $v_t$. If we have a given policy $\pi$, we can write

$$\mathcal{M}^\pi v(s) = C_t(s, A^\pi(s)) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}_t(s'|s, A^\pi(s)) v_{t+1}(s').$$

Alternatively, we can find the best action, which we represent using

$$\mathcal{M}v(s) = \max_a \big(C_t(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}_t(s'|s, a) v_{t+1}(s')\big).$$

Here, $\mathcal{M}v$ produces a vector, and $\mathcal{M}v(s)$ refers to element $s$ of this vector. In vector form, we would write

$$\mathcal{M}v = \max_\pi \big(c_t^\pi + \gamma P_t^\pi v_{t+1}\big).$$

Now let $\mathcal{V}$ be the space of value functions. Then, $\mathcal{M}$ is a mapping

$$\mathcal{M} : \mathcal{V} \to \mathcal{V}.$$

We may also define the operator $\mathcal{M}^\pi$ for a particular policy $\pi$ using

$$\mathcal{M}^\pi(v) = c_t^\pi + \gamma P^\pi v \tag{3.11}$$

for some vector $v \in \mathcal{V}$. $\mathcal{M}^\pi$ is known as a *linear operator* since the operations that it performs on $v$ are additive and multiplicative. In mathematics, the function $c_t^\pi + \gamma P^\pi v$ is known as an *affine function*. This notation is particularly useful in mathematical proofs (see in particular some of the proofs in section 3.10), but we will not use this notation when we describe models and algorithms.

We see later in the chapter that we can exploit the properties of this operator to derive some very elegant results for Markov decision processes. These proofs provide insights into the behavior of these systems, which can guide the design of algorithms. For this reason, it is relatively immaterial that the actual computation of these equations may be intractable for many problems; the insights still apply.

## 3.2 FINITE HORIZON PROBLEMS

Finite horizon problems tend to arise in two settings. First, some problems have a very specific horizon. For example, we might be interested in the value of an American option where we are allowed to sell an asset at any time $t \leq T$ where $T$ is the exercise date. Another problem is to determine how many seats to sell at different prices for a particular flight departing at some point in the future. In the same class are problems that require reaching some goal (but not at a particular point in time). Examples include driving to a destination, selling a house, or winning a game.

A second class of problems is actually infinite horizon, but where the goal is to determine what to do right now given a particular state of the system. For example, a transportation company might want to know what drivers should be assigned to a particular set of loads right now. Of course, these decisions need to consider the downstream impact, so models have to extend into the future. For this reason, we might model the problem over a horizon $T$ which, when solved, yields a decision of what to do right now.

When we encounter a finite horizon problem, we assume that we are given the function $V_T(S_T)$ as data. Often, we simply use $V_T(S_T) = 0$ because we are primarily interested in what to do now, given by $a_0$, or in projected activities over some horizon $t = 0, 1, \ldots, T^{ph}$, where $T^{ph}$ is the length of a planning horizon. If we set $T$ sufficiently larger than $T^{ph}$, then we may be able to assume that the decisions $a_0, a_1, \ldots, a_{T^{ph}}$ are of sufficiently high quality to be useful.

Solving a finite horizon problem, in principle, is straightforward. As outlined in figure 3.1, we simply have to start at the last time period, compute the value function for each possible state $s \in \mathcal{S}$, and then step back another time period. This way, at time period $t$ we have already computed $V_{t+1}(S)$. Not surprisingly, this method is often referred to as "backward dynamic programming." The critical element that attracts so much attention is the requirement that we compute the value function $V_t(S_t)$ for all states $S_t \in \mathcal{S}$.

---

**Step 0.** Initialization:

Initialize the terminal contribution $V_T(S_T)$.

Set $t = T - 1$.

**Step 1.** Calculate:

$$V_t(S_t) = \max_{a_t} \left\{ C_t(S_t, a_t) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|S_t, a_t) V_{t+1}(s') \right\}$$

for all $S_t \in \mathcal{S}$.

**Step 2.** If $t > 0$, decrement $t$ and return to step 1. Else, stop.

---

**Figure 3.1** A backward dynamic programming algorithm.

We first saw backward dynamic programming in section 2.2.1 when we described a simple decision tree problem. The only difference between the backward dynamic programming algorithm in figure 3.1 and our solution of the decision tree problem is primarily notational. Decision trees are visual and tend to be easier to understand, whereas in this section the methods are described using notation. However, decision tree problems tend to be always presented in the context of problems with relatively small numbers of

states and actions (What job should I take? Should the United States put a blockade around Cuba? Should the shuttle launch have been canceled due to cold weather?).

Another popular illustration of dynamic programming is the discrete asset acquisition problem. Assume that you order a quantity $a_t$ at each time period to be used in the next time period to satisfy a demand $\hat{D}_{t+1}$. Any unused product is held over to the following time period. For this, our state variable $S_t$ is the quantity of inventory left over at the end of the period after demands are satisfied. The transition equation is given by $S_{t+1} = [S_t + a_t - \hat{D}_{t+1}]^+$ where $[x]^+ = \max(x, 0)$. The cost function (which we seek to minimize) is given by $\hat{C}_{t+1}(S_t, a_t) = c^h S_t + c^o I_{\{a_t>0\}}$, where $I_{\{X\}} = 1$ if $X$ is true and 0 otherwise. Note that the cost function is nonconvex. This does not create problems if we solve our minimization problem by searching over different (discrete) values of $a_t$. Since all of our quantities are scalar, there is no difficulty finding $C_t(S_t, a_t)$.

To compute the one-step transition matrix, let $\Omega$ be the set of possible outcomes of $\hat{D}_t$, and let $\mathbb{P}(\hat{D}_t = \omega)$ be the probability that $\hat{D}_t = \omega$ (if this use of $\omega$ seems weird, get used to it - we are going to use it a lot).

The one-step transition matrix is computed using

$$\mathbb{P}(s'|s, a) = \sum_{\omega\in\Omega} \mathbb{P}(\hat{D}_{t+1} = \omega) 1_{\{s'=[s+a-\omega]^+\}}$$

where $\Omega$ is the set of (discrete) outcomes of the demand $\hat{D}_{t+1}$.

Another example is the shortest path problem with random arc costs. Assume that you are trying to get from origin node $q$ to destination node $r$ in the shortest time possible. As you reach each intermediate node $i$, you are able to observe the time required to traverse each arc out of node $i$. Let $V_j$ be the expected shortest path time from $j$ to the destination node $r$. At node $i$, you see the link time $\hat{\tau}_{ij}$ which represents a random observation of the travel time. Now we choose to traverse arc $i, j^*$ where $j^*$ solves $\min_j(\hat{\tau}_{ij} + V_j)$ ($j^*$ is random since the travel time is random). We would then compute the value of being at node $i$ using $V_i = \mathbb{E}\{\min_j(\hat{\tau}_{ij} + V_j)\}$.

## 3.3 INFINITE HORIZON PROBLEMS

We typically use infinite horizon formulations whenever we wish to study a problem where the parameters of the contribution function, transition function and the process governing the exogenous information process do not vary over time, although they may vary in cycles (for example, an infinite horizon model of energy storage from a solar panel may depend on time of day). Often, we wish to study such problems in steady state. More importantly, infinite horizon problems provide a number of insights into the properties of problems and algorithms, drawing off an elegant theory that has evolved around this problem class. Even students who wish to solve complex, nonstationary problems will benefit from an understanding of this problem class.

We begin with the optimality equations

$$V_t(S_t) = \max_{a_t\in\mathcal{A}} \mathbb{E}\left\{C_t(S_t, a_t) + \gamma V_{t+1}(S_{t+1})|S_t\right\}.$$

We can think of a steady-state problem as one without the time dimension. Letting $V(s) = \lim_{t\to\infty} V_t(S_t)$ (and assuming the limit exists), we obtain the steady-state opti-

mality equations

$$V(s) = \max_{a \in \mathcal{A}} \left\{ C(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s,a) V(s') \right\}. \tag{3.12}$$

The functions $V(s)$ can be shown (as we do later) to be equivalent to solving the infinite horizon problem

$$\max_{\pi \in \Pi} \mathbb{E} \left\{ \sum_{t=0}^{\infty} \gamma^t C_t(S_t, A_t^{\pi}(S_t)) \right\}. \tag{3.13}$$

Now define

$$\begin{aligned} P^{\pi,t} &= t\text{-step transition matrix, over periods } 0, 1, \ldots, t-1, \text{ given policy } \pi \\ &= \Pi_{t'=0}^{t-1} P_{t'}^{\pi}. \end{aligned} \tag{3.14}$$

We further define $P^{\pi,0}$ to be the identity matrix. As before, let $c_t^{\pi}$ be the column vector of the expected cost of being in each state given that we choose the action $a_t$ described by policy $\pi$, where the element for state $s$ is $c_t^{\pi}(s) = C_t(s, A^{\pi}(s))$. The infinite horizon, discounted value of a policy $\pi$ starting at time $t$ is given by

$$v_t^{\pi} = \sum_{t'=t}^{\infty} \gamma^{t'-t} P^{\pi,t'-t} c_{t'}^{\pi}. \tag{3.15}$$

Assume that after following policy $\pi_0$ we follow policy $\pi_1 = \pi_2 = \ldots = \pi$. In this case, equation (3.15) can now be written as (starting at $t = 0$)

$$v^{\pi_0} = c^{\pi_0} + \sum_{t'=1}^{\infty} \gamma^{t'} P^{\pi,t'} c_{t'}^{\pi} \tag{3.16}$$

$$= c^{\pi_0} + \sum_{t'=1}^{\infty} \gamma^{t'} \left( \Pi_{t''=0}^{t'-1} P_{t''}^{\pi} \right) c_{t'}^{\pi} \tag{3.17}$$

$$= c^{\pi_0} + \gamma P^{\pi_0} \sum_{t'=1}^{\infty} \gamma^{t'-1} \left( \Pi_{t''=1}^{t'-1} P_{t''}^{\pi} \right) c_{t'}^{\pi} \tag{3.18}$$

$$= c^{\pi_0} + \gamma P^{\pi_0} v^{\pi}. \tag{3.19}$$

Equation (3.19) shows us that the value of a policy is the single period reward plus a discounted terminal reward that is the same as the value of a policy starting at time 1. If our decision rule is stationary, then $\pi_0 = \pi_1 = \ldots = \pi_t = \pi$, which allows us to rewrite (3.19) as

$$v^{\pi} = c^{\pi} + \gamma P^{\pi} v^{\pi}. \tag{3.20}$$

This allows us to solve for the stationary reward explicitly (as long as $0 \leq \gamma < 1$), giving us

$$v^{\pi} = (I - \gamma P^{\pi})^{-1} c^{\pi}.$$

We can also write an infinite horizon version of the optimality equations using our operator notation. Letting $\mathcal{M}$ be the "max" (or "min") operator (also known as the Bellman operator), the infinite horizon version of equation (3.11) would be written

$$\mathcal{M}^{\pi}(v) = c^{\pi} + \gamma P^{\pi} v. \tag{3.21}$$

**Step 0.** Initialization:

Set $v^0(s) = 0 \ \forall s \in \mathcal{S}$.

Fix a tolerance parameter $\epsilon > 0$.

Set $n = 1$.

**Step 1.** For each $s \in \mathcal{S}$ compute:

$$v^n(s) = \max_{a \in \mathcal{A}} \big(C(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s,a) v^{n-1}(s')\big). \tag{3.22}$$

**Step 2.** If $\|v^n - v^{n-1}\| < \epsilon(1-\gamma)/2\gamma$, let $\pi^\epsilon$ be the resulting policy that solves (3.22), and let $v^\epsilon = v^n$ and stop; else set $n = n + 1$ and go to step 1.

**Figure 3.2** The value iteration algorithm for infinite horizon optimization

There are several algorithmic strategies for solving infinite horizon problems. The first, value iteration, is the most widely used method. It involves iteratively estimating the value function. At each iteration, the estimate of the value function determines which decisions we will make and as a result defines a policy. The second strategy is *policy iteration*. At every iteration, we define a policy (literally, the rule for determining decisions) and then determine the value function for that policy. Careful examination of value and policy iteration reveals that these are closely related strategies that can be viewed as special cases of a general strategy that uses value and policy iteration. Finally, the third major algorithmic strategy exploits the observation that the value function can be viewed as the solution to a specially structured linear programming problem.

## 3.4 VALUE ITERATION

Value iteration is perhaps the most widely used algorithm in dynamic programming because it is the simplest to implement and, as a result, often tends to be the most natural way of solving many problems. It is virtually identical to backward dynamic programming for finite horizon problems. In addition, most of our work in approximate dynamic programming is based on value iteration.

Value iteration comes in several flavors. The basic version of the value iteration algorithm is given in figure 3.2. The proof of convergence (see section 3.10.2) is quite elegant for students who enjoy mathematics. The algorithm also has several nice properties that we explore below.

It is easy to see that the value iteration algorithm is similar to the backward dynamic programming algorithm. Rather than using a subscript $t$, which we decrement from $T$ back to 0, we use an iteration counter $n$ that starts at 0 and increases until we satisfy a convergence criterion. Here, we stop the algorithm when

$$\|v^n - v^{n-1}\| < \epsilon(1-\gamma)/2\gamma,$$

where $\|v\|$ is the max-norm defined by

$$\|v\| = \max_s |v(s)|.$$

Thus, $\|v\|$ is the largest absolute value of a vector of elements. Thus, we stop if the largest change in the value of being in any state is less than $\epsilon(1-\gamma)/2\gamma$ where $\epsilon$ is a specified error tolerance.

Below, we describe a Gauss-Seidel variant which is a useful method for accelerating value iteration, and a version known as relative value iteration.

### 3.4.1 A Gauss-Seidel variation

A slight variant of the value iteration algorithm provides a faster rate of convergence. In this version (typically called the Gauss-Seidel variant), we take advantage of the fact that when we are computing the expectation of the value of the future, we have to loop over all the states $s'$ to compute $\sum_{s'} \mathbb{P}(s'|s,a)v^n(s')$. For a particular state $s$, we would have already computed $v^{n+1}(\hat{s})$ for $\hat{s} = 1, 2, \ldots, s-1$. By simply replacing $v^n(\hat{s})$ with $v^{n+1}(\hat{s})$ for the states we have already visited, we obtain an algorithm that typically exhibits a noticeably faster rate of convergence. The algorithm requires a change to step 1 of the value iteration, as shown in figure 3.3.

---

Replace Step 1 with

**Step 1'.** For each $s \in \mathcal{S}$ compute

$$v^n(s) = \max_{a \in \mathcal{A}} \left\{ C(s,a) + \gamma \left( \sum_{s'<s} \mathbb{P}(s'|s,a)v^n(s') + \sum_{s' \geq s} \mathbb{P}(s'|s,a)v^{n-1}(s') \right) \right\}$$

---

**Figure 3.3** The Gauss-Seidel variation of value iteration.

### 3.4.2 Relative value iteration

Another version of value iteration is called *relative value iteration*, which is useful in problems that do not have a discount factor or where the optimal policy converges much more quickly than the value function, which may grow steadily for many iterations. The relative value iteration algorithm is shown in 3.4.

In relative value iteration, we focus on the fact that we may be more interested in the convergence of the difference $|v(s) - v(s')|$ than we are in the values of $v(s)$ and $v(s')$. This would be the case if we are interested in the best policy rather than the value function itself (this is not always the case). What often happens is that, especially toward the limit, all the values $v(s)$ start increasing by the same rate. For this reason, we can pick any state (denoted $s^*$ in the algorithm) and subtract its value from all the other states.

To provide a bit of formalism for our algorithm, we define the *span* of a vector $v$ as follows:

$$sp(v) = \max_{s \in \mathcal{S}} v(s) - \min_{s \in \mathcal{S}} v(s).$$

Note that our use of "span" is different than the way it is normally used in linear algebra. Here and throughout this section, we define the norm of a vector as

$$\|v\| = \max_{s \in \mathcal{S}} v(s).$$

**Step 0.** Initialization:

- Choose some $v^0 \in \mathcal{V}$.
- Choose a base state $s^*$ and a tolerance $\epsilon$.
- Let $w^0 = v^0 - v^0(s^*)e$ where $e$ is a vector of ones.
- Set $n = 1$.

**Step 1.** Set

$$\begin{aligned} v^n &= \mathcal{M}w^{n-1}, \\ w^n &= v^n - v^n(s^*)e. \end{aligned}$$

**Step 2.** If $sp(v^n - v^{n-1}) < (1-\gamma)\epsilon/\gamma$, go to step 3; otherwise, go to step 1.

**Step 3.** Set $a^\epsilon = \arg\max_{a \in \mathcal{A}} \left(C(a) + \gamma P^\pi v^n\right)$.

**Figure 3.4** Relative value iteration.

Note that the span has the following six properties:

1) $sp(v) \geq 0$.
2) $sp(u+v) \leq sp(u) + sp(v)$.
3) $sp(kv) = |k| sp(v)$.
4) $sp(v + ke) = sp(v)$.
5) $sp(v) = sp(-v)$.
6) $sp(v) \leq 2\|v\|$.

Property $(4)$ implies that $sp(v) = 0$ does not mean that $v = 0$ and therefore it does not satisfy the properties of a norm. For this reason, it is called a *semi-norm*.

The relative value iteration algorithm is simply subtracting a constant from the value vector at each iteration. Obviously, this does not change the optimal decision, but it does change the value itself. If we are only interested in the optimal policy, relative value iteration often offers much faster convergence, but it may not yield accurate estimates of the value of being in each state.

### 3.4.3 Bounds and rates of convergence

One important property of value iteration algorithms is that if our initial estimate is too low, the algorithm will rise to the correct value from below. Similarly, if our initial estimate is too high, the algorithm will approach the correct value from above. This property is formalized in the following theorem:

**Theorem 3.4.1** *For a vector $v \in \mathcal{V}$:*

*(a) If $v$ satisfies $v \geq \mathcal{M}v$, then $v \geq v^*$.*

*(b) If $v$ satisfies $v \leq \mathcal{M}v$, then $v \leq v^*$.*

*(c) If $v$ satisfies $v = \mathcal{M}v$, then $v$ is the unique solution to this system of equations and $v = v^*$.*

The proof is given in section 3.10.3. It is a nice property because it provides some valuable information on the nature of the convergence path. In practice, we generally do not know the true value function, which makes it hard to know if we are starting from above or below (although some problems have natural bounds, such as nonnegativity).

The proof of the monotonicity property above also provides us with a nice corollary. If $V(s) = \mathcal{M}V(s)$ for all $s$, then $V(s)$ is the unique solution to this system of equations, which must also be the optimal solution.

This result raises the question: What if some of our estimates of the value of being in some states are too high, while others are too low? This means the values may cycle above and below the optimal solution, although at some point we may find that all the values have increased (decreased) from one iteration to the next. If this happens, then it means that the values are all equal to or below (above) the limiting value.

Value iteration also provides a nice bound on the quality of the solution. Recall that when we use the value iteration algorithm, we stop when

$$\|v^{n+1} - v^n\| < \epsilon(1-\gamma)/2\gamma \tag{3.23}$$

where $\gamma$ is our discount factor and $\epsilon$ is a specified error tolerance. It is possible that we have found the optimal policy when we stop, but it is very unlikely that we have found the optimal value functions. We can, however, provide a bound on the gap between the solution $v^n$ and the optimal values $v^*$ by using the following theorem:

**Theorem 3.4.2** *If we apply the value iteration algorithm with stopping parameter $\epsilon$ and the algorithm terminates at iteration $n$ with value function $v^{n+1}$, then*

$$\|v^{n+1} - v^*\| \leq \epsilon/2. \tag{3.24}$$

*Let $\pi^\epsilon$ be the policy that we terminate with, and let $v^{\pi^\epsilon}$ be the value of this policy. Then*

$$\|v^{\pi^\epsilon} - v^*\| \leq \epsilon.$$

The proof is given in section 3.10.4. While it is nice that we can bound the error, the bad news is that the bound can be quite poor. More important is what the bound teaches us about the role of the discount factor.

We can provide some additional insights into the bound, as well as the rate of convergence, by considering a trivial dynamic program. In this problem, we receive a constant reward $c$ at every iteration. There are no decisions, and there is no randomness. The value of this "game" is quickly seen to be

$$\begin{aligned} v^* &= \sum_{n=0}^{\infty} \gamma^n c \\ &= \frac{1}{1-\gamma} c. \end{aligned} \tag{3.25}$$

Consider what happens when we solve this problem using value iteration. Starting with $v^0 = 0$, we would use the iteration

$$v^n = c + \gamma v^{n-1}.$$

After we have repeated this $n$ times, we have

$$\begin{aligned} v^n &= \sum_{m=0}^{n-1} \gamma^n c \\ &= \frac{1-\gamma^n}{1-\gamma} c. \end{aligned} \tag{3.26}$$

Comparing equations (3.25) and (3.26), we see that

$$v^n - v^* = -\frac{\gamma^n}{1-\gamma}c. \tag{3.27}$$

Similarly, the change in the value from one iteration to the next is given by

$$\begin{aligned}
\|v^{n+1} - v^n\| &= \left|\frac{\gamma^{n+1}}{1-\gamma} - \frac{\gamma^n}{1-\gamma}\right| c \\
&= \gamma^n \left|\frac{\gamma}{1-\gamma} - \frac{1}{1-\gamma}\right| c \\
&= \gamma^n \left|\frac{\gamma-1}{1-\gamma}\right| c \\
&= \gamma^n c.
\end{aligned}$$

If we stop at iteration $n+1$, then it means that

$$\gamma^n c \leq \epsilon/2\left(\frac{1-\gamma}{\gamma}\right). \tag{3.28}$$

If we choose $\epsilon$ so that (3.28) holds with equality, then our error bound (from 3.24) is

$$\begin{aligned}
\|v^{n+1} - v^*\| &\leq \epsilon/2 \\
&= \frac{\gamma^{n+1}}{1-\gamma}c.
\end{aligned}$$

From (3.27), we know that the distance to the optimal solution is

$$|v^{n+1} - v^*| = \frac{\gamma^{n+1}}{1-\gamma}c,$$

which matches our bound.

This little exercise confirms that our bound on the error may be tight. It also shows that the error decreases geometrically at a rate determined by the discount factor. For this problem, the error arises because we are approximating an infinite sum with a finite one. For more realistic dynamic programs, we also have the effect of trying to find the optimal policy. When the values are close enough that we have, in fact, found the optimal policy, then we have only a Markov reward process (a Markov chain where we earn rewards for each transition). Once our Markov reward process has reached steady state, it will behave just like the simple problem we have just solved, where $c$ is the expected reward from each transition.

## 3.5 POLICY ITERATION

In policy iteration, we choose a policy and then find the infinite horizon, discounted value of the policy. This value is then used to choose a new policy. The general algorithm is described in figure 3.5. Policy iteration is popular for infinite horizon problems because of the ease with which we can find the value of a policy. As we showed in section 3.3, the value of following policy $\pi$ is given by

$$v^\pi = \left(I - \gamma P^\pi\right)^{-1} c^\pi. \tag{3.29}$$

**Step 0.** Initialization:

**Step 0a.** Select a policy $\pi^0$.

**Step 0b.** Set $n = 1$.

**Step 1.** Given a policy $\pi^{n-1}$:

**Step 1a.** Compute the one-step transition matrix $P^{\pi^{n-1}}$.

**Step 1b** Compute the contribution vector $c^{\pi^{n-1}}$ where the element for state $s$ is given by $c^{\pi^{n-1}}(s) = C(s, A^{\pi^{n-1}})$.

**Step 2.** Let $v^{\pi,n}$ be the solution to

$$(I - \gamma P^{\pi^{n-1}})v \quad = \quad c^{\pi^{n-1}}.$$

**Step 3.** Find a policy $\pi^n$ defined by

$$a^n(s) \quad = \quad \arg\max_{a \in \mathcal{A}} \big(C(a) + \gamma P^{\pi} v^n\big).$$

This requires that we compute an action for each state $s$.

**Step 4.** If $a^n(s) = a^{n-1}(s)$ for all states $s$, then set $a^* = a^n$; otherwise, set $n = n + 1$ and go to step 1.

**Figure 3.5** Policy iteration

While computing the inverse can be problematic as the state space grows, it is, at a minimum, a very convenient formula.

It is useful to illustrate the policy iteration algorithm in different settings. In the first, consider a batch replenishment problem where we have to replenish resources (raising capital, exploring for oil to expand known reserves, hiring people) where there are economies from ordering larger quantities. We might use a simple policy where if our level of resources $R_t < q$ for some lower limit $q$, we order a quantity $a_t = Q - R_t$. This policy is parameterized by $(q, Q)$ and is written

$$A^{\pi}(R_t) = \begin{cases} 0, & R_t \geq q, \\ Q - R_t, & R_t < q. \end{cases} \tag{3.30}$$

For a given set of parameters $\pi = (q, Q)$, we can compute a one-step transition matrix $P^{\pi}$ and a contribution vector $c^{\pi}$.

Policies come in many forms. For the moment, we simply view a policy as a rule that tells us what decision to make when we are in a particular state. In later chapters, we introduce policies in different forms since they create different challenges for finding the best policy.

Given a transition matrix $P^{\pi}$ and contribution vector $c^{\pi}$, we can use equation (3.29) to find $v^{\pi}$, where $v^{\pi}(s)$ is the discounted value of started in state $s$ and following policy $\pi$. From this vector, we can infer a new policy by solving

$$a^n(s) \quad = \quad \arg\max_{a \in \mathcal{A}} \big(C(a) + \gamma P^{\pi} v^n\big) \tag{3.31}$$

for each state $s$. For our batch replenishment example, it turns out that we can show that $a^n(s)$ will have the same structure as that shown in (3.30). So, we can either store $a^n(s)$ for each $s$, or simply determine the parameters $(q, Q)$ that correspond to the decisions produced by (3.31). The complete policy iteration algorithm is described in figure 3.5.

**Step 0.** Initialization:

- Set $n = 1$.
- Select a tolerance parameter $\epsilon$ and inner iteration limit $M$.
- Select some $v^0 \in \mathcal{V}$.

**Step 1.** Find a decision $a^n(s)$ for each $s$ that satisfies

$$a^n(s) \quad = \quad \arg\max_{a \in \mathcal{A}} \left\{ C(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s,a) v^{n-1}(s') \right\},$$

which we represent as policy $\pi^n$.

**Step 2.** Partial policy evaluation.

(a) Set $m = 0$ and let: $u^n(0) = c^\pi + \gamma P^{\pi^n} v^{n-1}$.

(b) If $\|u^n(0) - v^{n-1}\| < \epsilon(1-\gamma)/2\gamma$, go to step 3. Else:

(c) While $m < M$ do the following:

i) $u^n(m+1) = c^{\pi^n} + \gamma P^{\pi^n} u^n(m) = \mathcal{M}^\pi u^n(m)$.

ii) Set $m = m + 1$ and repeat $(i)$.

(d) Set $v^n = u^n(M), n = n + 1$ and return to step 1.

**Step 3.** Set $a^\epsilon = a^{n+1}$ and stop.

**Figure 3.6** Hybrid value/policy iteration

The policy iteration algorithm is simple to implement and has fast convergence when measured in terms of the number of iterations. However, solving equation (3.29) is quite hard if the number of states is large. If the state space is small, we can use $v^\pi = (I - \gamma P^\pi)^{-1} c^\pi$, but the matrix inversion can be computationally expensive. For this reason, we may use a hybrid algorithm that combines the features of policy iteration and value iteration.

## 3.6 HYBRID VALUE-POLICY ITERATION

Value iteration is basically an algorithm that updates the value at each iteration and then determines a new policy given the new estimate of the value function. At any iteration, the value function is not the true, steady-state value of the policy. By contrast, policy iteration picks a policy and then determines the true, steady-state value of being in each state given the policy. Given this value, a new policy is chosen.

It is perhaps not surprising that policy iteration converges faster in terms of the number of iterations because it is doing a lot more work in each iteration (determining the true, steady-state value of being in each state under a policy). Value iteration is much faster per iteration, but it is determining a policy given an approximation of a value function and then performing a very simple updating of the value function, which may be far from the true value function.

A hybrid strategy that combines features of both methods is to perform a somewhat more complete update of the value function before performing an update of the policy. Figure 3.6 outlines the procedure where the steady-state evaluation of the value function in equation (3.29) is replaced with a much easier iterative procedure (step 2 in figure 3.6).

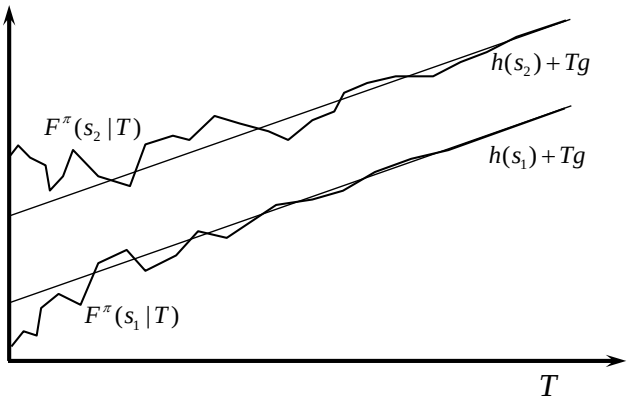



**Figure 3.7** Cumulative contribution over a horizon $T$ when starting in states $s_1$ and $s_2$, showing growth approaching a rate that is independent of the starting state.

This step is run for $M$ iterations, where $M$ is a user-controlled parameter that allows the exploration of the value of a better estimate of the value function. Not surprisingly, it will generally be the case that $M$ should decline with the number of iterations as the overall process converges.

## 3.7 AVERAGE REWARD DYNAMIC PROGRAMMING

There are settings where the natural objective function is to maximize the *average* contribution per unit time. Assume we start in state $s$. Then, the average reward from starting in state $s$ and following policy $\pi$ is given by

$$\max_{\pi} F^{\pi}(s) = \lim_{T\rightarrow\infty} \frac{1}{T}\mathbb{E}\sum_{t=0}^{T} C(S_t, A^{\pi}(S_t)). \tag{3.32}$$

Here, $F^{\pi}(s)$ is the expected reward *per time period*. In matrix form, the total value of following a policy $\pi$ over a horizon $T$ can be written as

$$V_T^{\pi} = \sum_{t=0}^{T} (P^{\pi})^t c^{\pi},$$

where $V_T^{\pi}$ is a column vector with element $V_T^{\pi}(s)$ giving the expected contribution over $T$ time periods when starting in state $s$. We can get a sense of how $V_T^{\pi}(s)$ behaves by watching what happens as $T$ becomes large. Assuming that our underlying Markov chain is ergodic (all the states communicate with each other with positive probability), we know that $(P^{\pi})^T \rightarrow P^*$ where the rows of $P^*$ are all the same.

Now define a column vector $g$ given by

$$g^{\pi} = P^* c^{\pi}.$$

Since the rows of $P^*$ are all the same, all the elements of $g^{\pi}$ are the same, and each element gives the average contribution per time period using the steady state probability of being in

each state. For finite $T$, each element of the column vector $V_T^\pi$ is not the same, since the contributions we earn in the first few time periods depends on our starting state. But it is not hard to see that as $T$ grows large, we can write

$$V_T^\pi \rightarrow h^\pi + Tg^\pi,$$

where $h^\pi$ captures the state-dependent differences in the total contribution, while $g^\pi$ is the state-independent average contribution in the limit. Figure 3.7 illustrates the growth in $V_T^\pi$ toward a linear function.

If we wish to find the policy that performs the best as $T \rightarrow \infty$, then clearly the contribution of $h^\pi$ vanishes, and we want to focus on maximizing $g^\pi$, which we can now treat as a scalar.

## 3.8 THE LINEAR PROGRAMMING METHOD FOR DYNAMIC PROGRAMS

Theorem 3.4.1 showed us that if

$$v \geq \max_a \big(C(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s,a)v(s')\big),$$

then $v$ is an upper bound (actually, a vector of upper bounds) on the value of being in each state. This means that the optimal solution, which satisfies $v^* = c + \gamma P v^*$, is the smallest value of $v$ that satisfies this inequality. We can use this insight to formulate the problem of finding the optimal values as a linear program. Let $\beta$ be a vector with elements $\beta_s > 0$, $\forall s \in \mathcal{S}$. The optimal value function can be found by solving the following linear program

$$\min_v \sum_{s \in \mathcal{S}} \beta_s v(s) \tag{3.33}$$

subject to

$$v(s) \quad \geq \quad C(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s,a)v(s') \quad \text{for all } s \text{ and } a, \tag{3.34}$$

The linear program has a $|\mathcal{S}|$-dimensional decision vector (the value of being in each state), with $|\mathcal{S}| \times |\mathcal{A}|$ inequality constraints (equation (3.34)).

This formulation was viewed as primarily a theoretical result for many years, since it requires formulating a linear program where the number of constraints is equal to the number of states and actions. While even today this limits the size of problems it can solve, modern linear programming solvers can handle problems with tens of thousands of constraints without difficulty. This size is greatly expanded with the use of specialized algorithmic strategies which are an active area of research as of this writing. The advantage of the LP method over value iteration is that it avoids the need for iterative learning with the geometric convergence exhibited by value iteration. Given the dramatic strides in the speed of linear programming solvers over the last decade, the relative performance of value iteration over the linear programming method is an unresolved question. However, this question only arises for problems with relatively small state and action spaces. While a linear program with 50,000 constraints is considered large, dynamic programs with 50,000 states and actions often arises with relatively small problems.

## 3.9 MONOTONE POLICIES*

One of the most dramatic success stories from the study of Markov decision processes has been the identification of the structure of optimal policies. A common example of structured policies is what are known as *monotone policies*. Simply stated, a monotone policy is one where the decision gets bigger as the state gets bigger, or the decision gets smaller as the state gets bigger (see examples).

---

**EXAMPLE 3.1**

A software company must decide when to ship the next release of its operating system. Let $S_t$ be the total investment in the current version of the software. Let $a_t = 1$ denote the decision to ship the release in time period $t$ while $a_t = 0$ means to keep investing in the system. The company adopts the rule that $a_t = 1$ if $S_t \geq \bar{S}$. Thus, as $S_t$ gets bigger, $a_t$ gets bigger (this is true even though $a_t$ is equal to zero or one).

**EXAMPLE 3.2**

An oil company maintains stocks of oil reserves to supply its refineries for making gasoline. A supertanker comes from the Middle East each month, and the company can purchase different quantities from this shipment. Let $R_t$ be the current inventory. The policy of the company is to order $a_t = Q - S_t$ if $S_t < R$. $R$ is the reorder point, and $Q$ is the "order up to" limit. The bigger $S_t$ is, the less the company orders.

**EXAMPLE 3.3**

A mutual fund has to decide when to sell its holding in a company. Its policy is to sell the stock when the price $\hat{p}_t$ is greater than a particular limit $\bar{p}$.

---

In each example, the decision of what to do in each state is replaced by a function that determines the decision (otherwise known as a policy). The function typically depends on the choice of a few parameters. So, instead of determining the right action for each possible state, we only have to determine the parameters that characterize the function. Interestingly, we do not need dynamic programming for this. Instead, we use dynamic programming to determine the structure of the optimal policy. This is a purely theoretical question, so the computational limitations of (discrete) dynamic programming are not relevant.

The study of monotone policies is included partly because it is an important part of the field of dynamic programming. It is also useful in the study of approximate dynamic programming because it yields properties of the value function. For example, in the process of showing that a policy is monotone, we also need to show that the value function itself is monotone (that is, it increases or decreases with the state variable). Such properties can be exploited in the estimation of a value function approximation.

To demonstrate the analysis of a monotone policy, we consider a classic batch replenishment policy that arises when there is a random accumulation that is then released in batches. Examples include dispatching elevators or trucks, moving oil inventories away from producing fields in tankers, and moving trainloads of grain from grain elevators.

### 3.9.1 The model

For our batch model, we assume resources accumulate and are then reduced using a batch process. For example, oil might accumulate in tanks before a tanker removes it. Money might accumulate in a cash account before it is swept into an investment.

Our model uses the following parameters:

$$\begin{aligned} c^r &= \text{The fixed cost incurred each time we dispatch a new batch.} \\ c^h &= \text{Penalty per time period for holding a unit of the resource.} \\ K &= \text{Maximum size of a batch.} \end{aligned}$$

Our exogenous information process consists of

$$\begin{aligned} Q_t &= \text{Quantity of new arrivals during time interval } t. \\ \mathbb{P}^Q(i) &= Prob(Q_t = i). \end{aligned}$$

Our state variable is

$$R_t = \text{Resources remaining at time } t \text{ before we have made a decision to send a batch.}$$

There are two decisions we have to make. The first is whether to dispatch a batch, and the second is how many resources to put in the batch. For this problem, once we make the decision to send a batch, we are going to make the batch as large as possible, so the "decision" of how large the batch should be seems unnecessary. It becomes more important when we later consider multiple resource types. For consistency with the more general problem with multiple resource types, we define

$$\begin{aligned} a_t &= \begin{cases} 1 & \text{if a batch is sent at time } t, \\ 0 & \text{otherwise,} \end{cases} \\ b_t &= \text{The number of resources to put in the batch.} \end{aligned}$$

In theory, we might be able to put a large number of resources in the batch, but we may face a nonlinear cost that makes this suboptimal. For the moment, we are going to assume that we always want to put as many as we can, so we set

$$\begin{aligned} b_t &= a_t \min\{K, R_t\}, \\ A^\pi(R_t) &= \text{The decision function that returns } a_t \text{ and } b_t \text{ given } R_t. \end{aligned}$$

The transition function is described using

$$R_{t+1} = R_t - b_t + Q_{t+1}. \tag{3.35}$$

The objective function is modeled using

$$\begin{aligned} C_t(R_t, a_t, b_t) &= \text{The cost incurred in period } t, \text{ given state } R_t \text{ and dispatch decision } a_t \\ &= c^r a_t + c^h(R_t - b_t). \end{aligned} \tag{3.36}$$

Our problem is to find the policy $A_t^\pi(R_t)$ that solves

$$\min_{\pi \in \Pi} \mathbb{E}\left\{ \sum_{t=0}^{T} C_t(R_t, A_t^\pi(R_t)) \right\}. \tag{3.37}$$

where $\Pi$ is the set of policies. If we are managing a single asset class, then $R_t$ and $b_t$ are scalars and the problem can be solved using standard backward dynamic programming techniques of the sort that were presented in chapter 3 (assuming that we have a probability model for the demand). In practice, many problems involve multiple asset classes, which makes standard techniques impractical. But we can use this simple problem to study the structure of the problem.

If $R_t$ is a scalar, and if we know the probability distribution for $Q_t$, then we can solve this using backward dynamic programming. Indeed, this is one of the classic dynamic programming problems in operations research. However, the solution to this problem seems obvious. We should dispatch a batch whenever the level of resources $R_t$ is greater than some number $\bar{r}_t$, which means we only have to find $\bar{r}_t$ (if we have a steady state, infinite horizon problem, then we would have to find a single parameter $\bar{r}$). The remainder of this section helps establish the theoretical foundation for making this argument. While not difficult, the mathematical level of this presentation is somewhat higher than our usual presentation.

### 3.9.2 Submodularity and other stories

In the realm of optimization problems over a continuous set, it is important to know a variety of properties about the objective function (such as convexity/concavity, continuity and boundedness). Similarly, discrete problems require an understanding of the nature of the functions we are maximizing, but there is a different set of conditions that we need to establish.

One of the most important properties that we will need is supermodularity (submodularity if we are minimizing). We assume we are studying a function $g(u), u \in \mathcal{U}$, where $\mathcal{U} \subseteq \Re^n$ is an $n$-dimensional space. Consider two vectors $u_1, u_2 \in \mathcal{U}$ where there is no particular relationship between $u_1$ and $u_2$. Now define

$$\begin{aligned} u_1 \wedge u_2 &= \min\{u_1, u_2\}, \\ u_1 \vee u_2 &= \max\{u_1, u_2\}, \end{aligned}$$

where the $\min$ and $\max$ are defined elementwise. Let $u^+ = u_1 \wedge u_2$ and $u^- = u_1 \vee u_2$. We first have to ask the question of whether $u^+, u^- \in \mathcal{U}$, since this is not guaranteed. For this purpose, we define the following:

**Definition 3.9.1** *The space $\mathcal{U}$ is a* **lattice** *if for each $u_1, u_2 \in \mathcal{U}$, then $u^+ = u_1 \wedge u_2 \in \mathcal{U}$ and $u^- = u_1 \vee u_2 \in \mathcal{U}$.*

The term "lattice" for these sets arises if we think of $u_1$ and $u_2$ as the northwest and southeast corners of a rectangle. In that case, these corners are $u^+$ and $u^-$. If all four corners fall in the set (for any pair $(u_1, u_2)$), then the set can be viewed as containing many "squares," similar to a lattice.

For our purposes, we assume that $\mathcal{U}$ is a lattice (if it is not, then we have to use a more general definition of the operators "$\vee$" and "$\wedge$"). If $\mathcal{U}$ is a lattice, then a general definition of supermodularity is given by the following:

**Definition 3.9.2** *A function $g(u), u \in \mathcal{U}$ is* **supermodular** *if it satisfies*

$$g(u_1 \wedge u_2) + g(u_1 \vee u_2) \geq g(u_1) + g(u_2) \tag{3.38}$$

Supermodularity is the discrete analog of a convex function. A function is **submodular** if the inequality in equation (3.38) is reversed. There is an alternative definition of supermodular when the function is defined on sets. Let $\mathcal{U}_1$ and $\mathcal{U}_2$ be two sets of elements, and let $g$ be a function defined on these sets. Then we have

**Definition 3.9.3** *A function $g : \mathcal{U} \mapsto \Re^1$ is* **supermodular** *if it satisfies*

$$g(\mathcal{U}_1 \cup \mathcal{U}_2) + g(\mathcal{U}_1 \cap \mathcal{U}_2) \geq g(\mathcal{U}_1) + g(\mathcal{U}_2) \tag{3.39}$$

We may refer to definition 3.9.2 as the vector definition of supermodularity, while definition 3.9.3 as the set definition. We give both definitions for completeness, but our work uses only the vector definition.

In dynamic programming, we are interested in functions of two variables, as in $f(s, a)$ where $s$ is a state variable and $a$ is a decision variable. We want to characterize the behavior of $f(s, a)$ as we change $s$ and $a$. If we let $u = (s, a)$, then we can put this in the context of our definition above. Assume we have two states $s^+ \geq s^-$ (again, the inequality is applied elementwise) and two decisions $a^+ \geq a^-$. Now, form two vectors $u_1 = (s^+, a^-)$ and $u_2 = (s^-, a^+)$. With this definition, we find that $u_1 \vee u_2 = (s^+, a^+)$ and $u_1 \wedge u_2 = (s^-, a^-)$. This gives us the following:

**Proposition 3.9.1** *A function $g(s, a)$ is supermodular if for $s^+ \geq s^-$ and $a^+ \geq a^-$, then*

$$g(s^+, a^+) + g(s^-, a^-) \geq g(s^+, a^-) + g(s^-, a^+). \tag{3.40}$$

For our purposes, equation (3.40) will be the version we will use.

A common variation on the statement of a supermodular function is the equivalent condition

$$g(s^+, a^+) - g(s^-, a^+) \geq g(s^+, a^-) - g(s^-, a^-) \tag{3.41}$$

In this expression, we are saying that the incremental change in $s$ for larger values of $a$ is greater than for smaller values of $a$. Similarly, we may write the condition as

$$g(s^+, a^+) - g(s^+, a^-) \geq g(s^-, a^+) - g(s^-, a^-) \tag{3.42}$$

which states that an incremental change in $a$ increases with $s$.

Some examples of supermodular functions include

(a) If $g(s, a) = g_1(s) + g_2(a)$, meaning that it is separable, then (3.40) holds with equality.

(b) $g(s, a) = h(s + a)$ where $h(\cdot)$ is convex and increasing.

(c) $g(s, a) = sa,\ s, a \in \Re^1$.

A concept that is related to supermodularity is *superadditivity*, defined by the following:

**Definition 3.9.4** *A superadditive function $f : \Re^n \to \Re^1$ satisfies*

$$f(x) + f(y) \quad \leq \quad f(x + y). \tag{3.43}$$

Some authors use superadditivity and supermodularity interchangeably, but the concepts are not really equivalent, and we need to use both of them.

### 3.9.3 From submodularity to monotonicity

It seems intuitively obvious that we should dispatch a batch if the state $R_t$ (the resources waiting to be served in a batch) is greater than some number (say, $\bar{r}_t$). The dispatch rule that says we should dispatch if $R_t \geq \bar{r}_t$ is known as a *control limit structure*. Similarly, we might be holding an asset and we feel that we should sell it if the price $p_t$ (which is the state of our asset) is over (or perhaps under) some number $\bar{p}_t$. A question arises: when is an optimal policy monotone? The following theorem establishes sufficient conditions for an optimal policy to be monotone.

**Theorem 3.9.1** *Assume that we are maximizing total discounted contribution and that*

*(a)* $C_t(R, a)$ *is supermodular on* $\mathcal{R} \times \mathcal{A}$.

*(b)* $\sum_{R' \in \mathcal{R}} \mathbb{P}(R'|R, a)v_{t+1}(R')$ *is supermodular on* $\mathcal{R} \times \mathcal{A}$.

*Then there exists a decision rule* $A^\pi(R)$ *that is nondecreasing on* $\mathcal{R}$.

The proof of this theorem is provided in section 3.10.6.

In the presentation that follows, we need to show submodularity (instead of supermodularity) because we are minimizing costs rather than maximizing rewards.

It is obvious that $C_t(R, a)$ is nondecreasing in $R$. So it remains to show that $C_t(R, a)$ satisfies

$$C_t(R^+, 1) - C_t(R^-, 1) \leq C_t(R^+, 0) - C_t(R^-, 0). \tag{3.44}$$

Substituting equation (3.36) into (3.44), we must show that

$$c^r + c^h(R^+ - K)^+ - c^r - c^h(R^- - K)^+ \leq c^h R^+ - c^h R^-.$$

This simplifies to

$$(R^+ - K)^+ - (R^- - K)^+ \leq R^+ - R^-. \tag{3.45}$$

Since $R^+ \geq R^-$, $(R^+ - K)^+ = 0 \Rightarrow (R^- - K)^+ = 0$. This implies there are three possible cases for equation (3.45):

**Case 1:** $(R^+ - K)^+ > 0$ and $(R^- - K)^+ > 0$. In this case, (3.45) reduces to $R^+ - R^- = R^+ - R^-$.

**Case 2:** $(R^+ - K)^+ > 0$ and $(R^- - K)^+ = 0$. Here, (3.45) reduces to $R^- \leq K$, which follows since $(R^- - K)^+ = 0$ implies that $R^- \leq K$.

**Case 3:** $(R^+ - K)^+ = 0$ and $(R^- - K)^+ = 0$. Now, (3.45) reduces to $R^- \leq R^+$, which is true by construction.

Now we have to show submodularity of $\sum_{R'=0}^{\infty} \mathbb{P}(R'|R, a)V(R')$. We will do this for the special case that the batch capacity is so large that we never exceed it. A proof is available for the finite capacity case, but it is much more difficult.

Submodularity requires that for $R^- \leq R^+$ we have

$$\sum_{R'=0}^{\infty} \mathbb{P}(R'|R^+, 1)V(R') - \sum_{R'=0}^{\infty} \mathbb{P}(R'|R^+, 0)V(R') \leq \sum_{R'=0}^{\infty} \mathbb{P}(R'|R^-, 1)V(R') - \sum_{R'=0}^{\infty} \mathbb{P}(R'|R^-, 0)V(R')$$

For the case that $R^-, R^+ \leq K$ we have

$$\begin{aligned}\sum_{R'=0}^{\infty} \mathbb{P}^A(R')V(R') - \sum_{R'=R^+}^{\infty} \mathbb{P}^A(R'-R^+)V(R') &\leq \sum_{R'=0}^{\infty} \mathbb{P}^A(R')V(R') \\ &\quad - \sum_{R'=R^-}^{\infty} \mathbb{P}^A(R'-R^-)V(R'),\end{aligned}$$

which simplifies to

$$\begin{aligned}\sum_{R'=0}^{\infty} \mathbb{P}^A(R')V(R') - \sum_{R'=0}^{\infty} \mathbb{P}^A(R')V(R'+R^+) &\leq \sum_{R'=0}^{\infty} \mathbb{P}^A(R')V(R') \\ &\quad - \sum_{R'=0}^{\infty} \mathbb{P}^A(R')V(R'+R^-).\end{aligned}$$

Since $V$ is nondecreasing we have $V(R'+R^+) \geq V(R'+R^-)$, proving the result.

## 3.10 WHY DOES IT WORK?**

The theory of Markov decision processes is especially elegant. While not needed for computational work, an understanding of why they work will provide a deeper appreciation of the properties of these problems.

Section 3.10.1 provides a proof that the optimal value function satisfies the optimality equations. Section 3.10.2 proves convergence of the value iteration algorithm. Section 3.10.3 then proves conditions under which value iteration increases or decreases monotonically to the optimal solution. Then, section 3.10.4 proves the bound on the error when value iteration satisfies the termination criterion given in section 3.4.3. Section 3.10.5 closes with a discussion of deterministic and randomized policies, along with a proof that deterministic policies are always at least as good as a randomized policy.

### 3.10.1 The optimality equations

Until now, we have been presenting the optimality equations as though they were a fundamental law of some sort. To be sure, they can easily look as though they were intuitively obvious, but it is still important to establish the relationship between the original optimization problem and the optimality equations. Since these equations are the foundation of dynamic programming, it seems beholden on us to work through the steps of proving that they are actually true.

We start by remembering the original optimization problem:

$$F_t^\pi(S_t) = \mathbb{E}\left\{\sum_{t'=t}^{T-1} C_{t'}(S_{t'}, A_{t'}^\pi(S_{t'})) + C_T(S_T)|S_t\right\}. \tag{3.46}$$

Since (3.46) is, in general, exceptionally difficult to solve, we resort to the optimality equations

$$V_t^\pi(S_t) = C_t(S_t, A_t^\pi(S_t)) + \mathbb{E}\left\{V_{t+1}^\pi(S_{t+1})|S_t\right\}. \tag{3.47}$$

Our challenge is to show that these are the same. In order to establish this result, it is going to help if we first prove the following:

**Lemma 3.10.1** *Let $S_t$ be a state variable that captures the relevant history up to time $t$, and let $F_{t'}(S_{t+1})$ be some function measured at time $t' \geq t+1$ conditioned on the random variable $S_{t+1}$. Then*

$$\mathbb{E}\left[\mathbb{E}\{F_{t'}|S_{t+1}\}|S_t\right] = \mathbb{E}\left[F_{t'}|S_t\right]. \tag{3.48}$$

**Proof:** This lemma is variously known as the law of iterated expectations or the tower property. Assume, for simplicity, that $F_{t'}$ is a discrete, finite random variable that takes outcomes in $\mathcal{F}$. We start by writing

$$\mathbb{E}\{F_{t'}|S_{t+1}\} = \sum_{f\in\mathcal{F}} f\mathbb{P}(F_{t'} = f|S_{t+1}). \tag{3.49}$$

Recognizing that $S_{t+1}$ is a random variable, we may take the expectation of both sides of (3.49), conditioned on $S_t$ as follows:

$$\mathbb{E}\left[\mathbb{E}\{F_{t'}|S_{t+1}\}|S_t\right] = \sum_{S_{t+1}\in\mathcal{S}}\sum_{f\in\mathcal{F}} f\mathbb{P}(F_{t'} = f|S_{t+1}, S_t)\mathbb{P}(S_{t+1} = S_{t+1}|S_t). \tag{3.50}$$

First, we observe that we may write $\mathbb{P}(F_{t'} = f|S_{t+1}, S_t) = \mathbb{P}(F_{t'} = f|S_{t+1})$, because conditioning on $S_{t+1}$ makes all prior history irrelevant. Next, we can reverse the summations on the right-hand side of (3.50) (some technical conditions have to be satisfied to do this, but these are satisfied if the random variables are discrete and finite). This means

$$\begin{aligned}
\mathbb{E}\left[\mathbb{E}\{F_{t'}|S_{t+1} = S_{t+1}\}|S_t\right] &= \sum_{f\in\mathcal{F}}\sum_{S_{t+1}\in\mathcal{S}} f\mathbb{P}(F_{t'} = f|S_{t+1}, S_t)\mathbb{P}(S_{t+1} = S_{t+1}|S_t) \\
&= \sum_{f\in\mathcal{F}} f \sum_{S_{t+1}\in\mathcal{S}} \mathbb{P}(F_{t'} = f, S_{t+1}|S_t) \\
&= \sum_{f\in\mathcal{F}} f\mathbb{P}(F_{t'} = f|S_t) \\
&= \mathbb{E}\left[F_{t'}|S_t\right],
\end{aligned}$$

which proves our result. Note that the essential step in the proof occurs in the first step when we add $S_t$ to the conditioning. □

We are now ready to show the following:

**Proposition 3.10.1** $F_t^\pi(S_t) = V_t^\pi(S_t)$.

**Proof:** To prove that (3.46) and (3.47) are equal, we use a standard trick in dynamic programming: proof by induction. Clearly, $F_T^\pi(S_T) = V_T^\pi(S_T) = C_T(S_T)$. Next, assume that it holds for $t+1, t+2, \ldots, T$. We want to show that it is true for $t$. This means that we can write

$$V_t^\pi(S_t) = C_t(S_t, A_t^\pi(S_t)) + \mathbb{E}\left[\underbrace{\mathbb{E}\left\{\sum_{t'=t+1}^{T-1} C_{t'}(S_{t'}, A_{t'}^\pi(S_{t'})) + C_t(S_T(\omega))\,\middle|\, S_{t+1}\right\}}_{F_{t+1}^\pi(S_{t+1})}\middle| S_t\right].$$

We then use lemma 3.10.1 to write $\mathbb{E}\left[\mathbb{E}\left\{\ldots|S_{t+1}\right\}|S_t\right] = \mathbb{E}\left[\ldots|S_t\right]$. Hence,

$$V_t^\pi(S_t) = C_t(S_t, A_t^\pi(S_t)) + \mathbb{E}\left[\sum_{t'=t+1}^{T-1} C_{t'}(S_{t'}, A_{t'}^\pi(S_{t'})) + C_t(S_T)|S_t\right].$$

When we condition on $S_t$, $A_t^\pi(S_t)$ (and therefore $C_t(S_t, A_t^\pi(S_t))$) is deterministic, so we can pull the expectation out to the front giving

$$\begin{aligned} V_t^\pi(S_t) &= \mathbb{E}\left[\sum_{t'=t}^{T-1} C_{t'}(S_{t'}, y_{t'}(S_{t'})) + C_t(S_T)|S_t\right] \\ &= F_t^\pi(S_t), \end{aligned}$$

which proves our result. □

Using equation (3.47), we have a backward recursion for calculating $V_t^\pi(S_t)$ for a given policy $\pi$. Now that we can find the expected reward for a given $\pi$, we would like to find the best $\pi$. That is, we want to find

$$F_t^*(S_t) = \max_{\pi\in\Pi} F_t^\pi(S_t).$$

If the set $\Pi$ is infinite, we replace the "$\max$" with "$\sup$". We solve this problem by solving the optimality equations. These are

$$V_t(S_t) = \max_{a\in\mathcal{A}}\left(C_t(S_t, a) + \sum_{s'\in\mathcal{S}} p_t(s'|S_t, a)V_{t+1}(s')\right). \tag{3.51}$$

We are claiming that if we find the set of $V's$ that solves (3.51), then we have found the policy that optimizes $F_t^\pi$. We state this claim formally as:

**Theorem 3.10.1** *Let $V_t(S_t)$ be a solution to equation (3.51). Then*

$$\begin{aligned} F_t^* &= V_t(S_t) \\ &= \max_{\pi\in\Pi} F_t^\pi(S_t). \end{aligned}$$

**Proof:** The proof is in two parts. First, we show by induction that $V_t(S_t) \geq F_t^*(S_t)$ for all $S_t \in \mathcal{S}$ and $t = 0, 1, \ldots, T-1$. Then, we show that the reverse inequality is true, which gives us the result.

Part 1:

We resort again to our proof by induction. Since $V_T(S_T) = C_t(S_T) = F_T^\pi(S_T)$ for all $S_T$ and all $\pi \in \Pi$, we get that $V_T(S_T) = F_T^*(S_T)$.

Assume that $V_{t'}(S_{t'}) \geq F_{t'}^*(S_{t'})$ for $t' = t+1, t+2, \ldots, T$, and let $\pi$ be an arbitrary policy. For $t' = t$, the optimality equation tells us

$$V_t(S_t) = \max_{a\in\mathcal{A}}\left(C_t(S_t, a) + \sum_{s'\in\mathcal{S}} p_t(s'|S_t, a)V_{t+1}(s')\right).$$

By the induction hypothesis, $F_{t+1}^*(s) \leq V_{t+1}(s)$, so we get

$$V_t(S_t) \geq \max_{a\in\mathcal{A}}\left(C_t(S_t, a) + \sum_{s'\in\mathcal{S}} p_t(s'|S_t, a)F_{t+1}^*(s')\right).$$

Of course, we have that $F^*_{t+1}(s) \geq F^\pi_{t+1}(s)$ for an arbitrary $\pi$. Also let $A^\pi(S_t)$ be the decision that would be chosen by policy $\pi$ when in state $S_t$. Then

$$\begin{aligned} V_t(S_t) &\geq \max_{a\in\mathcal{A}}\left(C_t(S_t,a) + \sum_{s'\in\mathcal{S}} p_t(s'|S_t,a)F^\pi_{t+1}(s')\right) \\ &\geq C_t(S_t,A^\pi(S_t)) + \sum_{s'\in\mathcal{S}} p_t(s'|S_t,A^\pi(S_t))F^\pi_{t+1}(s') \\ &= F^\pi_t(S_t). \end{aligned}$$

This means

$$V_t(S_t) \geq F^\pi_t(S_t) \quad \text{for all } \pi \in \Pi,$$

which proves part 1.

Part 2:

Now we are going to prove the inequality from the other side. Specifically, we want to show that for any $\epsilon > 0$ there exists a policy $\pi$ that satisfies

$$F^\pi_t(S_t) + (T-t)\epsilon \geq V_t(S_t). \tag{3.52}$$

To do this, we start with the definition

$$V_t(S_t) = \max_{a\in\mathcal{A}}\left(C_t(S_t,a) + \sum_{s'\in\mathcal{S}} p_t(s'|S_t,a)V_{t+1}(s')\right). \tag{3.53}$$

We may let $a_t(S_t)$ be the decision rule that solves (3.53). This rule corresponds to the policy $\pi$. In general, the set $\mathcal{A}$ may be infinite, whereupon we have to replace the "$\max$" with a "$\sup$" and handle the case where an optimal decision may not exist. For this case, we know that we can design a decision rule $a_t(S_t)$ that returns a decision $a$ that satisfies

$$V_t(S_t) \leq C_t(S_t,a) + \sum_{s'\in\mathcal{S}} p_t(s'|S_t,a)V_{t+1}(s') + \epsilon. \tag{3.54}$$

We can prove (3.52) by induction. We first note that (3.52) is true for $t = T$ since $F^\pi_T(S_t) = V_T(S_T)$. Now assume that it is true for $t' = t+1, t+2, \ldots, T$. We already know that

$$F^\pi_t(S_t) = C_t(S_t,A^\pi(S_t)) + \sum_{s'\in\mathcal{S}} p_t(s'|S_t,A^\pi(S_t))F^\pi_{t+1}(s').$$

We can use our induction hypothesis which says $F^\pi_{t+1}(s') \geq V_{t+1}(s') - (T-(t+1))\epsilon$ to get

$$\begin{aligned} F^\pi_t(S_t) &\geq C_t(S_t,A^\pi(S_t)) + \sum_{s'\in\mathcal{S}} p_t(s'|S_t,A^\pi(S_t))[V_{t+1}(s') - (T-(t+1))\epsilon] \\ &= C_t(S_t,A^\pi(S_t)) + \sum_{s'\in\mathcal{S}} p_t(s'|S_t,A^\pi(S_t))V_{t+1}(s') \\ &\quad - \sum_{s'\in\mathcal{S}} p_t(s'|S_t,A^\pi(S_t))\left[(T-t-1)\epsilon\right] \\ &= \left\{C_t(S_t,A^\pi(S_t)) + \sum_{s'\in\mathcal{S}} p_t(s'|S_t,A^\pi(S_t))V_{t+1}(s') + \epsilon\right\} - (T-t)\epsilon. \end{aligned}$$

Now, using equation (3.54), we replace the term in brackets with the smaller $V_t(S_t)$ (equation (3.54)):

$$F_t^\pi(S_t) \quad \geq \quad V_t(S_t) - (T-t)\epsilon,$$

which proves the induction hypothesis. We have shown that

$$F_t^*(S_t) + (T-t)\epsilon \geq F_t^\pi(S_t) + (T-t)\epsilon \geq V_t(S_t) \geq F_t^*(S_t).$$

This proves the result. □

Now we know that solving the optimality equations also gives us the optimal value function. This is our most powerful result because we can solve the optimality equations for many problems that cannot be solved any other way.

### 3.10.2 Convergence of value iteration

We now undertake the proof that the basic value function iteration converges to the optimal solution. This is not only an important result, it is also an elegant one that brings some powerful theorems into play. The proof is also quite short. However, we will need some mathematical preliminaries:

**Definition 3.10.1** *Let $\mathcal{V}$ be a set of (bounded, real-valued) functions and define the norm of $v$ by:*

$$\|v\| \quad = \quad \sup_{s \in \mathcal{S}} v(s)$$

*where we replace the "$sup$" with a "$max$" when the state space is finite. Since $\mathcal{V}$ is closed under addition and scalar multiplication and has a norm, it is a* **normed linear space**.

**Definition 3.10.2** *$T : \mathcal{V} \rightarrow \mathcal{V}$ is a* **contraction mapping** *if there exists a $\gamma$, $0 \leq \gamma < 1$ such that:*

$$\|Tv - Tu\| \quad \leq \quad \gamma\|v - u\|.$$

**Definition 3.10.3** *A sequence $v^n \in \mathcal{V},\ n = 1, 2, \ldots$ is said to be a* **Cauchy sequence** *if for all $\epsilon > 0$, there exists $N$ such that for all $n, m \geq N$ :*

$$\|v^n - v^m\| < \epsilon.$$

**Definition 3.10.4** *A normed linear space is* **complete** *if every Cauchy sequence contains a limit point in that space.*

**Definition 3.10.5** *A* **Banach space** *is a complete normed linear space.*

**Definition 3.10.6** *We define the norm of a matrix $Q$ as*

$$\|Q\| \quad = \quad \max_{s \in \mathcal{S}} \sum_{j \in \mathcal{S}} |q(j|s)|,$$

*that is, the largest row sum of the matrix. If $Q$ is a one-step transition matrix, then $\|Q\| = 1$.*

**Definition 3.10.7** *The* **triangle inequality** *means that given two vectors $a, b \in \Re^n$:*

$$\|a + b\| \quad \leq \quad \|a\| + \|b\|.$$

The triangle inequality is commonly used in proofs because it helps us establish bounds between two solutions (and in particular, between a solution and the optimum).

We now state and prove one of the famous theorems in applied mathematics and then use it immediately to prove convergence of the value iteration algorithm.

**Theorem 3.10.2** *(Banach Fixed-Point Theorem) Let $\mathcal{V}$ be a Banach space, and let $T : \mathcal{V} \rightarrow \mathcal{V}$ be a contraction mapping. Then:*

*(a) There exists a unique $v^* \in \mathcal{V}$ such that $Tv^* = v^*$.*

*(b) For an arbitrary $v^0 \in \mathcal{V}$, the sequence $v^n$ defined by: $v^{n+1} = Tv^n = T^{n+1}v^0$ converges to $v^*$.*

**Proof:** We start by showing that the distance between two vectors $v^n$ and $v^{n+m}$ goes to zero for sufficiently large $n$ and by writing the difference $v^{n+m} - v^n$ using

$$\begin{aligned} v^{n+m} - v^n &= v^{n+m} - v^{n+m-1} + v^{n+m-1} - \cdots - v^{n+1} + v^{n+1} - v^n \\ &= \sum_{k=0}^{m-1} (v^{n+k+1} - v^{n+k}). \end{aligned}$$

Taking norms of both sides and invoking the triangle inequality gives

$$\begin{aligned} \|v^{n+m} - v^n\| &= \|\sum_{k=0}^{m-1} (v^{n+k+1} - v^{n+k})\| \\ &\leq \sum_{k=0}^{m-1} \|(v^{n+k+1} - v^{n+k})\| \\ &= \sum_{k=0}^{m-1} \|(T^{n+k}v^1 - T^{n+k}v^0)\| \\ &\leq \sum_{k=0}^{m-1} \gamma^{n+k}\|v^1 - v^0\| \\ &= \frac{\gamma^n(1-\gamma^m)}{(1-\gamma)}\|v^1 - v^0\|. \end{aligned} \tag{3.55}$$

Since $\gamma < 1$, for sufficiently large $n$ the right-hand side of (3.55) can be made arbitrarily small, which means that $v^n$ is a Cauchy sequence. Since $\mathcal{V}$ is *complete*, it must be that $v^n$ has a limit point $v^*$. From this we conclude

$$\lim_{n\rightarrow\infty} v^n \rightarrow v^*. \tag{3.56}$$

We now want to show that $v^*$ is a fixed point of the mapping $T$. To show this, we observe

$$\begin{aligned} 0 &\leq \|Tv^* - v^*\| && (3.57)\\ &= \|Tv^* - v^n + v^n - v^*\| && (3.58)\\ &\leq \|Tv^* - v^n\| + \|v^n - v^*\| && (3.59)\\ &= \|Tv^* - Tv^{n-1}\| + \|v^n - v^*\| && (3.60)\\ &\leq \gamma\|v^* - v^{n-1}\| + \|v^n - v^*\|. && (3.61) \end{aligned}$$

Equation (3.57) comes from the properties of a norm. We play our standard trick in (3.58) of adding and subtracting a quantity (in this case, $v^n$), which sets up the triangle inequality in (3.59). Using $v^n = Tv^{n-1}$ gives us (3.60). The inequality in (3.61) is based on the assumption of the theorem that $T$ is a contraction mapping. From (3.56), we know that

$$\lim_{n\rightarrow\infty} \|v^* - v^{n-1}\| \quad = \quad \lim_{n\rightarrow\infty} \|v^n - v^*\| = 0. \tag{3.62}$$

Combining (3.57), (3.61), and (3.62) gives

$$0 \quad \leq \quad \|Tv^* - v^*\| \leq 0$$

from which we conclude

$$\|Tv^* - v^*\| \quad = \quad 0,$$

which means that $Tv^* = v^*$.

We can prove uniqueness by contradiction. Assume that there are two limit points that we represent as $v^*$ and $u^*$. The assumption that $T$ is a contraction mapping requires that

$$\|Tv^* - Tu^*\| \leq \gamma\|v^* - u^*\|.$$

But, if $v^*$ and $u^*$ are limit points, then $Tv^* = v^*$ and $Tu^* = u^*$, which means

$$\|v^* - u^*\| \leq \gamma\|v^* - u^*\|.$$

Since $\gamma < 1$, this is a contradiction, which means that it must be true that $v^* = u^*$. □

We can now show that the value iteration algorithm converges to the optimal solution if we can establish that $\mathcal{M}$ is a contraction mapping. So we need to show the following:

**Proposition 3.10.2** *If $0 \leq \gamma < 1$, then $\mathcal{M}$ is a contraction mapping on $\mathcal{V}$.*

**Proof:** Let $u, v \in \mathcal{V}$ and assume that $\mathcal{M}v \geq \mathcal{M}u$ where the inequality is applied element-wise. For a particular state $s$ let

$$a_s^*(v) \in \arg\max_{a\in\mathcal{A}} \left( C(s,a) + \gamma \sum_{s'\in\mathcal{S}} \mathbb{P}(s'|s,a)v(s') \right)$$

where we assume that a solution exists. Then

$$
\begin{aligned}
0 &\leq \mathcal{M}v(s) - \mathcal{M}u(s) && (3.63)\\
&= C(s, a_s^*(v)) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s, a_s^*(v))v(s') \\
&\quad - \left( C(s, a_s^*(u)) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s, a_s^*(u))u(s') \right) && (3.64)\\
&\leq C(s, a_s^*(v)) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s, a_s^*(v))v(s') \\
&\quad - \left( C(s, a_s^*(v)) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s, a_s^*(v))u(s') \right) && (3.65)\\
&= \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s, a_s^*(v))[v(s') - u(s')] && (3.66)\\
&\leq \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s, a_s^*(v))\|v - u\| && (3.67)\\
&= \gamma \|v - u\| \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s, a_s^*(v)) && (3.68)\\
&= \gamma \|v - u\|. && (3.69)
\end{aligned}
$$

Equation (3.63) is true by assumption, while (3.64) holds by definition. The inequality in (3.65) holds because $a_s^*(v)$ is not optimal when the value function is $u$, giving a reduced value in the second set of parentheses. Equation (3.66) is a simple reduction of (3.65). Equation (3.67) forms an upper bound because the definition of $\|v - u\|$ is to replace all the elements $[v(s) - u(s)]$ with the largest element of this vector. Since this is now a vector of constants, we can pull it outside of the summation, giving us (3.68), which then easily reduces to (3.69) because the probabilities add up to one.

This result states that if $\mathcal{M}v(s) \geq \mathcal{M}u(s)$, then $\mathcal{M}v(s) - \mathcal{M}u(s) \leq \gamma|v(s) - u(s)|$. If we start by assuming that $\mathcal{M}v(s) \leq \mathcal{M}u(s)$, then the same reasoning produces $\mathcal{M}v(s) - \mathcal{M}u(s) \geq -\gamma|v(s) - u(s)|$. This means that we have

$$
|\mathcal{M}v(s) - \mathcal{M}u(s)| \leq \gamma|v(s) - u(s)| \tag{3.70}
$$

for *all* states $s \in \mathcal{S}$. From the definition of our norm, we can write

$$
\begin{aligned}
\sup_{s \in \mathcal{S}} |\mathcal{M}v(s) - \mathcal{M}u(s)| &= \|\mathcal{M}v - \mathcal{M}u\| \\
&\leq \gamma \|v - u\|.
\end{aligned}
$$

This means that $\mathcal{M}$ is a contraction mapping, which means that the sequence $v^n$ generated by $v^{n+1} = \mathcal{M}v^n$ converges to a unique limit point $v^*$ that satisfies the optimality equations. □

### 3.10.3 Monotonicity of value iteration

Infinite horizon dynamic programming provides a compact way to study the theoretical properties of these algorithms. The insights gained here are applicable to problems even when we cannot apply this model, or these algorithms, directly.

We assume throughout our discussion of infinite horizon problems that the reward function is bounded over the domain of the state space. This assumption is virtually always satisfied in practice, but notable exceptions exist. For example, the assumption is violated if we are maximizing a utility function that depends on the log of the resources we have at hand (the resources may be bounded, but the function is unbounded if the resources are allowed to hit zero).

Our first result establishes a monotonicity property that can be exploited in the design of an algorithm.

**Theorem 3.10.3** *For a vector $v \in \mathcal{V}$:*

*(a) If $v$ satisfies $v \geq \mathcal{M}v$, then $v \geq v^*$.*

*(b) If $v$ satisfies $v \leq \mathcal{M}v$, then $v \leq v^*$.*

*(c) If $v$ satisfies $v = \mathcal{M}v$, then $v$ is the unique solution to this system of equations and $v = v^*$.*

**Proof:** Part $(a)$ requires that

$$\begin{aligned} v &\geq \max_{\pi \in \Pi} \{c^\pi + \gamma P^\pi v\} && (3.71)\\ &\geq c^{\pi_0} + \gamma P^{\pi_0} v && (3.72)\\ &\geq c^{\pi_0} + \gamma P^{\pi_0} \left(c^{\pi_1} + \gamma P^{\pi_1} v\right) && (3.73)\\ &= c^{\pi_0} + \gamma P^{\pi_0} c^{\pi_1} + \gamma^2 P^{\pi_0} P^{\pi_1} v. \end{aligned}$$

Equation (3.71) is true by assumption (part $(a)$ of the theorem) and equation (3.72) is true because $\pi_0$ is some policy that is not necessarily optimal for the vector $v$. Using similar reasoning, equation (3.73) is true because $\pi_1$ is another policy which, again, is not necessarily optimal. Using $P^{\pi,(t)} = P^{\pi_0} P^{\pi_1} \cdots P^{\pi_t}$, we obtain by induction

$$v \geq c^{\pi_0} + \gamma P^{\pi_0} c^{\pi_1} + \cdots + \gamma^{t-1} P^{\pi_0} P^{\pi_1} \cdots P^{\pi_{t-1}} c^{\pi_t} + \gamma^t P^{\pi,(t)} v. \quad (3.74)$$

Recall that

$$v^\pi = \sum_{t=0}^{\infty} \gamma^t P^{\pi,(t)} c^{\pi_t}. \quad (3.75)$$

Breaking the sum in (3.75) into two parts allows us to rewrite the expansion in (3.74) as

$$v \geq v^\pi - \sum_{t'=t+1}^{\infty} \gamma^{t'} P^{\pi,(t')} c^{\pi_{t'+1}} + \gamma^t P^{\pi,(t)} v. \quad (3.76)$$

Taking the limit of both sides of (3.76) as $t \to \infty$ gives us

$$\begin{aligned} v &\geq \lim_{t\to\infty} v^\pi - \sum_{t'=t+1}^{\infty} \gamma^{t'} P^{\pi,(t')} c^{\pi_{t'+1}} + \gamma^t P^{\pi,(t)} v && (3.77)\\ &\geq v^\pi \quad \forall \pi \in \Pi. && (3.78) \end{aligned}$$

The limit in (3.77) exists as long as the reward function $c^\pi$ is bounded and $\gamma < 1$. Because (3.78) is true for all $\pi \in \Pi$, it is also true for the optimal policy, which means that

$$\begin{aligned} v &\geq v^{\pi *}\\ &= v^*, \end{aligned}$$

which proves part (a) of the theorem. Part $(b)$ can be proved in an analogous way. Parts (a) and (b) mean that $v \geq v^*$ and $v \leq v^*$. If $v = \mathcal{M}v$, then we satisfy the preconditions of both parts (a) and (b), which means they are both true and therefore we must have $v = v^*$. □

This result means that if we start with a vector that is higher than the optimal vector, then we will decline monotonically to the optimal solution (almost – we have not quite proven that we actually get to the optimal). Alternatively, if we start below the optimal vector, we will rise to it. Note that it is not always easy to find a vector $v$ that satisfies either condition $(a)$ or $(b)$ of the theorem. In problems where the rewards can be positive and negative, this can be tricky.

### 3.10.4 Bounding the error from value iteration

We now wish to establish a bound on our error from value iteration, which will establish our stopping rule. We propose two bounds: one on the value function estimate that we terminate with and one for the long-run value of the decision rule that we terminate with. To define the latter, let $\pi^\epsilon$ be the policy that satisfies our stopping rule, and let $v^{\pi^\epsilon}$ be the infinite horizon value of following policy $\pi^\epsilon$.

**Theorem 3.10.4** *If we apply the value iteration algorithm with stopping parameter $\epsilon$ and the algorithm terminates at iteration $n$ with value function $v^{n+1}$, then*

$$\|v^{n+1} - v^*\| \leq \epsilon/2, \tag{3.79}$$

*and*

$$\|v^{\pi^\epsilon} - v^*\| \leq \epsilon. \tag{3.80}$$

**Proof:** We start by writing

$$\begin{aligned} \|v^{\pi^\epsilon} - v^*\| &= \|v^{\pi^\epsilon} - v^{n+1} + v^{n+1} - v^*\| \\ &\leq \|v^{\pi^\epsilon} - v^{n+1}\| + \|v^{n+1} - v^*\|. \end{aligned} \tag{3.81}$$

Recall that $\pi^\epsilon$ is the policy that solves $\mathcal{M}v^{n+1}$, which means that $\mathcal{M}^{\pi^\epsilon}v^{n+1} = \mathcal{M}v^{n+1}$. This allows us to rewrite the first term on the right-hand side of (3.81) as

$$\begin{aligned} \|v^{\pi^\epsilon} - v^{n+1}\| &= \|\mathcal{M}^{\pi^\epsilon}v^{\pi^\epsilon} - \mathcal{M}v^{n+1} + \mathcal{M}v^{n+1} - v^{n+1}\| \\ &\leq \|\mathcal{M}^{\pi^\epsilon}v^{\pi^\epsilon} - \mathcal{M}v^{n+1}\| + \|\mathcal{M}v^{n+1} - v^{n+1}\| \\ &= \|\mathcal{M}^{\pi^\epsilon}v^{\pi^\epsilon} - \mathcal{M}^{\pi^\epsilon}v^{n+1}\| + \|\mathcal{M}v^{n+1} - \mathcal{M}v^n\| \\ &\leq \gamma\|v^{\pi^\epsilon} - v^{n+1}\| + \gamma\|v^{n+1} - v^n\|. \end{aligned}$$

Solving for $\|v^{\pi^\epsilon} - v^{n+1}\|$ gives

$$\|v^{\pi^\epsilon} - v^{n+1}\| \leq \frac{\gamma}{1-\gamma}\|v^{n+1} - v^n\|.$$

We can use similar reasoning applied to the second term in equation (3.81) to show that

$$\|v^{n+1} - v^*\| \leq \frac{\gamma}{1-\gamma}\|v^{n+1} - v^n\|. \tag{3.82}$$

The value iteration algorithm stops when $\|v^{n+1} - v^n\| \leq \epsilon(1-\gamma)/2\gamma$. Substituting this in (3.82) gives

$$\|v^{n+1} - v^*\| \quad \leq \quad \frac{\epsilon}{2}. \tag{3.83}$$

Recognizing that the same bound applies to $\|v^{\pi^\epsilon} - v^{n+1}\|$ and combining these with (3.81) gives us

$$\|v^{\pi^\epsilon} - v^*\| \quad \leq \quad \epsilon,$$

which completes our proof. □

### 3.10.5 Randomized policies

We have implicitly assumed that for each state, we want a single action. An alternative would be to choose a policy probabilistically from a family of policies. If a state produces a single action, we say that we are using a *deterministic policy*. If we are randomly choosing an action from a set of actions probabilistically, we say we are using a *randomized policy*.

Randomized policies may arise because of the nature of the problem. For example, you wish to purchase something at an auction, but you are unable to attend yourself. You may have a simple rule ("purchase it as long as the price is under a specific amount") but you cannot assume that your representative will apply the same rule. You can choose a representative, and in doing so you are effectively choosing the probability distribution from which the action will be chosen.

Behaving randomly also plays a role in two-player games. If you make the same decision each time in a particular state, your opponent may be able to predict your behavior and gain an advantage. For example, as an institutional investor you may tell a bank that you not willing to pay any more than \$14 for a new offering of stock, while in fact you are willing to pay up to \$18. If you always bias your initial prices by \$4, the bank will be able to guess what you are willing to pay.

When we can only influence the likelihood of an action, then we have an instance of a randomized MDP. Let

$q_t^\pi(a|S_t) =$ The probability that decision $a$ will be taken at time $t$ given state $S_t$ and policy $\pi$ (more precisely, decision rule $A^\pi$).

In this case, our optimality equations look like

$$V_t^*(S_t) = \max_{\pi \in \Pi^{MR}} \sum_{a \in \mathcal{A}} \left[ q_t^\pi(a|S_t) \left( C_t(S_t, a) + \sum_{s' \in \mathcal{S}} p_t(s'|S_t, a) V_{t+1}^*(s') \right) \right]. \tag{3.84}$$

Now let us consider the single best action that we could take. Calling this $a^*$, we can find it using

$$a^* = \arg\max_{a \in \mathcal{A}} \left[ C_t(S_t, a) + \sum_{s' \in \mathcal{S}} p_t(s'|S_t, a) V_{t+1}^*(s') \right].$$

This means that

$$C_t(S_t, a^*) + \sum_{s' \in \mathcal{S}} p_t(s'|S_t, a^*) V_{t+1}^*(s') \quad \geq \quad C_t(S_t, a) + \sum_{s' \in \mathcal{S}} p_t(s'|S_t, a) V_{t+1}^*(s') \tag{3.85}$$

for all $a \in \mathcal{A}$. Substituting (3.85) back into (3.84) gives us

$$\begin{aligned} V_t^*(S_t) &= \max_{\pi \in \Pi^{MR}} \sum_{a \in \mathcal{A}} \left\{ q_t^\pi(a|S_t) \left( C_t(S_t, a) + \sum_{s' \in \mathcal{S}} p_t(s'|S_t, a) V_{t+1}^*(s') \right) \right\} \\ &\leq \max_{\pi \in \Pi^{MR}} \sum_{a \in \mathcal{A}} \left\{ q_t^\pi(a|S_t) \left( C_t(S_t, a^*) + \sum_{s' \in \mathcal{S}} p_t(s'|S_t, a^*) V_{t+1}^*(s') \right) \right\} \\ &= C_t(S_t, a^*) + \sum_{s' \in \mathcal{S}} p_t(s'|S_t, a^*) V_{t+1}^*(s'). \end{aligned}$$

What this means is that if you have a choice between picking exactly the action you want versus picking a probability distribution over potentially optimal and nonoptimal actions, you would always prefer to pick exactly the best action. Clearly, this is not a surprising result.

The value of randomized policies arise primarily in two-person games, where one player tries to anticipate the actions of the other player. In such situations, part of the state variable is the estimate of what the other play will do when the game is in a particular state. By randomizing his behavior, a player reduces the ability of the other player to anticipate his moves.

### 3.10.6 Optimality of monotone policies

The foundational result that we use is the following technical lemma:

**Lemma 3.10.2** *If a function $g(s, a)$ is supermodular, then*

$$a^*(s) = \max \left\{ a' \in \arg\max_a g(s, a) \right\} \tag{3.86}$$

*is monotone and nondecreasing in $s$.*

If the function $g(s, a)$ has a unique, optimal $a^*(s)$ for each value of $s$, then we can replace (3.86) with

$$a^*(s) = \max_a g(s, a). \tag{3.87}$$

**Discussion:** The lemma is saying that if $g(s, a)$ is supermodular, then as $s$ grows larger, the optimal value of $a$ given $s$ will grow larger. When we use the version of supermodularity given in equation (3.42), we see that the condition implies that as the state becomes larger, the value of increasing the decision also grows. As a result, it is not surprising that the condition produces a decision rule that is monotone in the state vector.

**Proof of the lemma:** Assume that $s^+ \geq s^-$, and choose $a \leq a^*(s^-)$. Since $a^*(s)$ is, by definition, the best value of $a$ given $s$, we have

$$g(s^-, a^*(s^-)) - g(s^-, a) \geq 0. \tag{3.88}$$

The inequality arises because $a^*(s^-)$ is the best value of $a$ given $s^-$. Supermodularity requires that

$$g(s^-, a) + g(s^+, a^*(s^-)) \geq g(s^-, a^*(s^-)) + g(s^+, a) \tag{3.89}$$

Rearranging (3.89) gives us

$$g(s^+, a^*(s^-)) \geq \underbrace{\left\{g(s^-, a^*(s^-)) - g(s^-, a)\right\}}_{\geq 0} + g(s^+, a) \quad \forall a \leq a^*(s^-) \tag{3.90}$$

$$\geq \quad g(s^+, a) \qquad \forall a \leq a^*(s^-) \tag{3.91}$$

We obtain equation (3.91) because the term in brackets in (3.90) is nonnegative (from (3.88)).

Clearly

$$g(s^+, a^*(s^+)) \quad \geq \quad g(s^+, a^*(s^-))$$

because $a^*(s^+)$ optimizes $g(s^+, a)$. This means that $a^*(s^+) \geq a^*(s^-)$ since otherwise, we would simply have chosen $a = a^*(s^-)$.

Just as the sum of concave functions is concave, we have the following:

**Proposition 3.10.3** *The sum of supermodular functions is supermodular.*

The proof follows immediately from the definition of supermodularity, so we leave it as one of those proverbial exercises for the reader.

The main theorem regarding monotonicity is relatively easy to state and prove, so we will do it right away. The conditions required are what make it a little more difficult.

**Theorem 3.10.5** *Assume that:*

*(a)* $C_t(s, a)$ *is supermodular on* $\mathcal{S} \times \mathcal{A}$.

*(b)* $\sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s, a) v_{t+1}(s')$ *is supermodular on* $\mathcal{S} \times \mathcal{A}$.

*Then there exists a decision rule* $a(s)$ *that is nondecreasing on* $\mathcal{S}$.

**Proof:** Let

$$w(s, a) \quad = \quad C_t(s, a) + \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s, a) v_{t+1}(s') \tag{3.92}$$

The two terms on the right-hand side of (3.92) are assumed to be supermodular, and we know that the sum of two supermodular functions is supermodular, which tells us that $w(s, a)$ is supermodular. Let

$$a^*(s) \quad = \quad \arg\max_{a \in \mathcal{A}} w(s, a)$$

From Lemma 3.10.2, we obtain the result that the decision $a^*(s)$ increases monotonically over $\mathcal{S}$, which proves our result.

The proof that the one-period reward function $C_t(s, a)$ is supermodular must be based on the properties of the function for a specific problem. Of greater concern is establishing the conditions required to prove condition $(b)$ of the theorem because it involves the property of the value function, which is not part of the basic data of the problem.

In practice, it is sometimes possible to establish condition $(b)$ directly based on the nature of the problem. These conditions usually require conditions on the monotonicity of the reward function (and hence the value function) along with properties of the one-step transition matrix. For this reason, we will start by showing that if the one-period reward

function is nondecreasing (or nonincreasing), then the value functions are nondecreasing (or nonincreasing). We will first need the following technical lemma:

**Lemma 3.10.3** *Let $p_j, p'_j, j \in \mathcal{J}$ be probability mass functions defined over $\mathcal{J}$ that satisfy*

$$\sum_{j=j'}^{\infty} p_j \quad \geq \quad \sum_{j=j'}^{\infty} p'_j \quad \forall j' \in \mathcal{J} \tag{3.93}$$

*and let $v_j, j \in \mathcal{J}$ be a nondecreasing sequence of numbers. Then*

$$\sum_{j=0}^{\infty} p_j v_j \quad \geq \quad \sum_{j=0}^{\infty} p'_j v_j \tag{3.94}$$

We would say that the distribution represented by $\{p_j\}_{j\in\mathcal{J}}$ *stochastically dominates* the distribution $\{p'_j\}_{j\in\mathcal{J}}$. If we think of $p_j$ as representing the probability a random variable $V = v_j$, then equation (3.94) is saying that $E^p V \geq E^{p'} V$. Although this is well known, a more algebraic proof is as follows:

**Proof:** Let $v_{-1} = 0$ and write

$$\begin{aligned}
\sum_{j=0}^{\infty} p_j v_j &= \sum_{j=0}^{\infty} p_j \sum_{i=0}^{j} (v_i - v_{i-1}) && (3.95)\\
&= \sum_{j=0}^{\infty} (v_j - v_{j-1}) \sum_{i=j}^{\infty} p_i && (3.96)\\
&= \sum_{j=1}^{\infty} (v_j - v_{j-1}) \sum_{j=i}^{\infty} p_i + v_0 \sum_{i=0}^{\infty} p_i && (3.97)\\
&\geq \sum_{j=1}^{\infty} (v_j - v_{j-1}) \sum_{i=j}^{\infty} p'_j + v_0 \sum_{i=0}^{\infty} p'_j && (3.98)\\
&= \sum_{j=0}^{\infty} p'_j v_j && (3.99)
\end{aligned}$$

In equation (3.95), we replace $v_j$ with an alternating sequence that sums to $v_j$. Equation (3.96) involves one of those painful change of variable tricks with summations. Equation (3.97) is simply getting rid of the term that involves $v_{-1}$. In equation (3.98), we replace the cumulative distributions for $p_j$ with the distributions for $p'_j$, which gives us the inequality. Finally, we simply reverse the logic to get back to the expectation in (3.99). □

We stated that lemma 3.10.3 is true when the sequences $\{p_j\}$ and $\{p'_j\}$ are probability mass functions because it provides an elegant interpretation as expectations. For example, we may use $v_j = j$, in which case equation (3.94) gives us the familiar result that when one probability distribution stochastically dominates another, it has a larger mean. If we use an increasing sequence $v_j$ instead of $j$, then this can be viewed as nothing more than the same result on a transformed axis.

In our presentation, however, we need a more general statement of the lemma, which follows:

**Lemma 3.10.4** *Lemma 3.10.3 holds for any real valued, nonnegative (bounded) sequences $\{p_j\}$ and $\{p'_j\}$.*

The proof involves little more than realizing that the proof of lemma 3.10.3 never required that the sequences $\{p_j\}$ and $\{p'_j\}$ be probability mass functions.

**Proposition 3.10.4** *Suppose that:*

*(a)* $C_t(s,a)$ *is nondecreasing (nonincreasing) in* $s$ *for all* $a \in \mathcal{A}$ *and* $t \in \mathcal{T}$.

*(b)* $C_T(s)$ *is nondecreasing (nonincreasing) in* $s$.

*(c)* $q_t(\bar{s}|s,a) = \sum_{s' \geq \bar{s}} \mathbb{P}(s'|s,a)$, *the reverse cumulative distribution function for the transition matrix, is nondecreasing in* $s$ *for all* $s \in \mathcal{S}, a \in \mathcal{A}$ *and* $t \in \mathcal{T}$.

*Then,* $v_t(s)$ *is nondecreasing (nonincreasing) in* $s$ *for* $t \in \mathcal{T}$.

**Proof:** As always, we use a proof by induction. We will prove the result for the nondecreasing case. Since $v_T(s) = C_t(s)$, we obtain the result by assumption for $t = T$. Now, assume the result is true for $v_{t'}(s)$ for $t' = t+1, t+2, \ldots, T$. Let $a^*_t(s)$ be the decision that solves:

$$\begin{aligned} v_t(s) &= \max_{a \in \mathcal{A}} C_t(s,a) + \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s,a) v_{t+1}(s') \\ &= C_t(s, a^*_t(s)) + \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s, a^*_t(s)) v_{t+1}(s') \end{aligned} \tag{3.100}$$

Let $\hat{s} \geq s$. Condition $(c)$ of the proposition implies that:

$$\sum_{s' \geq s} \mathbb{P}(s'|s,a) \leq \sum_{s' \geq s} \mathbb{P}(s'|\hat{s},a) \tag{3.101}$$

Lemma 3.10.4 tells us that when (3.101) holds, and if $v_{t+1}(s')$ is nondecreasing (the induction hypothesis), then:

$$\sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s,a) v_{t+1}(s') \leq \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|\hat{s},a) v_{t+1}(s') \tag{3.102}$$

Combining equation (3.102) with condition (a) of proposition 3.10.4 into equation (3.100) gives us

$$\begin{aligned} v_t(s) &\leq C_t(\hat{s}, a^*(s)) + \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|\hat{s}, a^*(s)) v_{t+1}(s') \\ &\leq \max_{a \in \mathcal{A}} C_t(\hat{s},a) + \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|\hat{s},a) v_{t+1}(s') \\ &= v_t(\hat{s}), \end{aligned}$$

which proves the proposition. □

With this result, we can establish condition (b) of theorem 3.10.5:

**Proposition 3.10.5** *If*

*(a)* $q_t(\bar{s}|s,a) = \sum_{s' \geq \bar{s}} \mathbb{P}(s'|s,a)$ *is supermodular on* $\mathcal{S} \times \mathcal{A}$ *and*

*(b)* $v(s)$ *is nondecreasing in* $s$,

*then* $\sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s,a)v(s')$ *is supermodular on* $\mathcal{S} \times \mathcal{A}$.

**Proof:** Supermodularity of the reverse cumulative distribution means:

$$\sum_{s' \geq \bar{s}} \mathbb{P}(s'|s^+,a^+) + \sum_{s' \geq \bar{s}} \mathbb{P}(s'|s^-,a^-) \quad \geq \quad \sum_{s' \geq \bar{s}} \mathbb{P}(s'|s^+,a^-) + \sum_{s' \geq \bar{s}} \mathbb{P}(s'|s^-,a^+)$$

We can apply Lemma 3.10.4 using $p_{\bar{s}} = \sum_{s' \geq \bar{s}} \mathbb{P}(s'|s^+,a^+) + \sum_{s' \geq \bar{s}} \mathbb{P}(s'|s^-,a^-)$ and $p'_{\bar{s}} = \sum_{s' \geq \bar{s}} \mathbb{P}(s'|s^+,a^-) + \sum_{s' \geq \bar{s}} \mathbb{P}(s'|s^-,a^+)$, which gives

$$\sum_{s' \in \mathcal{S}} \left(\mathbb{P}(s'|s^+,a^+) + \mathbb{P}(s'|s^-,a^-)\right) v(s') \quad \geq \quad \sum_{s' \in \mathcal{S}} \left(\mathbb{P}(s'|s^+,a^-) + \mathbb{P}(s'|s^-,a^+)\right) v(s')$$

which implies that $\sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s,a)v(s')$ is supermodular. □

**Remark:** Supermodularity of the reverse cumulative distribution $\sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s,a)$ may seem like a bizarre condition at first, but a little thought suggests that it is often satisfied in practice. As stated, the condition means that

$$\sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s^+,a^+) - \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s^+,a^-) \quad \geq \quad \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s^-,a^+) - \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s^-,a^-)$$

Assume that the state $s$ is the water level in a dam, and the decision $a$ controls the release of water from the dam. Because of random rainfalls, the amount of water behind the dam in the next time period, given by $s'$, is random. The reverse cumulative distribution gives us the probability that the amount of water is greater than $s^+$ (or $s^-$). Our supermodularity condition can now be stated as: "If the amount of water behind the dam is higher one month ($s^+$), then the effect of the decision of how much water to release ($a$) has a greater impact than when the amount of water is initially at a lower level ($s^-$)." This condition is often satisfied because a control frequently has more of an impact when a state is at a higher level than a lower level.

For another example of supermodularity of the reverse cumulative distribution, assume that the state represents a person's total wealth, and the control is the level of taxation. The effect of higher or lower taxes is going to have a bigger impact on wealthier people than on those who are not as fortunate (but not always: think about other forms of taxation that affect less affluent people more than the wealthy, and use this example to create an instance of a problem where a monotone policy may not apply).

We now have the result that if the reward function $C_t(s,a)$ is nondecreasing in $s$ for all $a \in \mathcal{A}$ and the reverse cumulative distribution $\sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s,a)$ is supermodular, then $\sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s,a)v(s')$ is supermodular on $\mathcal{S} \times \mathcal{A}$. Combine this with the supermodularity of the one-period reward function, and we obtain the optimality of a nondecreasing decision function.

## 3.11 BIBLIOGRAPHIC NOTES

This chapter presents the classic view of Markov decision processes, for which the literature is extensive. Beginning with the seminal text of Bellman (Bellman (1957)), there have been numerous, significant textbooks on the subject, including Howard (1960), Nemhauser (1966), White (1969), Derman (1970), Bellman (1971), Dreyfus & Law (1977), Dynkin & Yushkevich (1979), Denardo (1982), Ross (1983) and Heyman & Sobel (1984). As of this writing, the current high-water mark for textbooks in this area is the landmark volume

by Puterman (2005). Most of this chapter is based on Puterman (2005), modified to our notational style.

Section 3.8 - The linear programming method was first proposed in Manne (1960) (see subsequent discussions in Derman (1962) and Puterman (2005)). The so-called "linear programming method" was ignored for many years because of the large size of the linear programs that were produced, but the method has seen a resurgence of interest using approximation techniques. Recent research into algorithms for solving problems using this method are discussed in section 10.8.

Section 3.10.6 - In addition to Puterman (2005), see also Topkins (1978).

## PROBLEMS

**3.1** A classical inventory problem works as follows: Assume that our state variable $R_t$ is the amount of product on hand at the end of time period $t$ and that $D_t$ is a random variable giving the demand during time interval $(t-1, t)$ with distribution $p_d = \mathbb{P}(D_t = d)$. The demand in time interval $t$ must be satisfied with the product on hand at the beginning of the period. We can then order a quantity $a_t$ at the end of period $t$ that can be used to replenish the inventory in period $t+1$.

(a) Give the transition function that relates $R_{t+1}$ to $R_t$ if the order quantity is $a_t$ (where $a_t$ is fixed for all $R_t$).

(b) Give an algebraic version of the one-step transition matrix $P^\pi = \{p^\pi_{ij}\}$ where $p^\pi_{ij} = \mathbb{P}(R_{t+1} = j | R_t = i, A^\pi = a_t)$.

**3.2** Repeat the previous exercise, but now assume that we have adopted a policy $\pi$ that says we should order a quantity $a_t = 0$ if $R_t \geq s$ and $a_t = Q - R_t$ if $R_t < q$ (we assume that $R_t \leq Q$). Your expression for the transition matrix will now depend on our policy $\pi$ (which describes both the structure of the policy and the control parameter $s$).

**3.3** We are going to use a very simple Markov decision process to illustrate how the initial estimate of the value function can affect convergence behavior. In fact, we are going to use a Markov reward process to illustrate the behavior because our process does not have any decisions. Assume we have a two-stage Markov chain with one-step transition matrix

$$P = \begin{bmatrix} 0.7 & 0.3 \\ 0.05 & 0.95 \end{bmatrix}.$$

The contribution from each transition from state $i \in \{1, 2\}$ to state $j \in \{1, 2\}$ is given by the matrix

$$\begin{bmatrix} 10 & 30 \\ 30 & 5 \end{bmatrix}.$$

That is, a transition from state 1 to state 2 returns a contribution of 30. Apply the value iteration algorithm for an infinite horizon problem (note that you are not choosing a decision so there is no maximization step). The calculation of the value of being in each state will depend on your previous estimate of the value of being in each state. The calculations can be easily implemented in a spreadsheet. Assume that your discount factor is .8.

(a) Plot the value of being in state 1 as a function of the number of iterations if your initial estimate of the value of being in each state is 0. Show the graph for 50 iterations of the algorithm.

(b) Repeat this calculation using initial estimates of 100.

(c) Repeat the calculation using an initial estimate of the value of being in state 1 of 100, and use 0 for the value of being in state 2. Contrast the behavior with the first two starting points.

**3.4** Show that $\mathbb{P}(S_{t+\tau}|S_t)$, given that we are following a policy $\pi$ (for stationary problems), is given by (3.14). [Hint: first show it for $\tau = 1, 2$ and then use inductive reasoning to show that it is true for general $\tau$.]

**3.5** Apply policy iteration to the problem given in exercise 3.3. Plot the average value function (that is, average the value of being in each state) after each iteration alongside the average value function found using value iteration after each iteration (for value iteration, initialize the value function to zero). Compare the computation time for one iteration of value iteration and one iteration of policy iteration.

**3.6** Now apply the hybrid value-policy iteration algorithm to the problem given in exercise 3.3. Show the average value function after each major iteration (update of $n$) with $M = 1, 2, 3, 5, 10$. Compare the convergence rate to policy iteration and value iteration.

**3.7** An oil company will order tankers to fill a group of large storage tanks. One full tanker is required to fill an entire storage tank. Orders are placed at the beginning of each four week accounting period but do not arrive until the end of the accounting period. During this period, the company may be able to sell 0, 1 or 2 tanks of oil to one of the regional chemical companies (orders are conveniently made in units of storage tanks). The probability of a demand of 0, 1 or 2 is 0.40, 0.40 and 0.20, respectively.

A tank of oil costs \$1.6 million (M) to purchase and sells for \$2M. It costs \$0.020M to store a tank of oil during each period (oil ordered in period $t$, which cannot be sold until period $t + 1$, is not charged any holding cost in period $t$). Storage is only charged on oil that is in the tank at the beginning of the period and remains unsold during the period. It is possible to order more oil than can be stored. For example, the company may have two full storage tanks, order three more, and then only sell one. This means that at the end of the period, they will have four tanks of oil. Whenever they have more than two tanks of oil, the company must sell the oil directly from the ship for a price of \$0.70M. There is no penalty for unsatisfied demand.

An order placed in time period $t$ must be paid for in time period $t$ even though the order does not arrive until $t + 1$. The company uses an interest rate of 20 percent per accounting period (that is, a discount factor of 0.80).

(a) Give an expression for the one-period reward function $r(s, d)$ for being in state $s$ and making decision $d$. Compute the reward function for all possible states (0, 1, 2) and all possible decisions (0, 1, 2).

(b) Find the one-step probability transition matrix when your action is to order one or two tanks of oil. The transition matrix when you order zero is given by

| From-To | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 1 | 0 | 0 |
| 1 | 0.6 | 0.4 | 0 |
| 2 | 0.2 | 0.4 | 0.4 |

(c) Write out the general form of the optimality equations and solve this problem in steady state.

(d) Solve the optimality equations using the value iteration algorithm, starting with $V(s) = 0$ for $s = 0, 1$ and 2. You may use a programming environment, but the problem can be solved in a spreadsheet. Run the algorithm for 20 iterations. Plot $V^n(s)$ for $s = 0, 1, 2$, and give the optimal action for each state at each iteration.

(e) Give a bound on the value function after each iteration.

**3.8** Every day, a salesman visits $N$ customers in order to sell the $R$ identical items he has in his van. Each customer is visited exactly once and each customer buys zero or one item. Upon arrival at a customer location, the salesman quotes one of the prices $0 < p_1 \leq p_2 \leq \ldots \leq p_m$. Given that the quoted price is $p_i$, a customer buys an item with probability $r_i$. Naturally, $r_i$ is decreasing in $i$. The salesman is interested in maximizing the total expected revenue for the day. Show that if $r_i p_i$ is increasing in $i$, then it is always optimal to quote the highest price $p_m$.

**3.9** You need to decide when to replace your car. If you own a car of age $y$ years, then the cost of maintaining the car that year will be $c(y)$. Purchasing a new car (in constant dollars) costs $P$ dollars. If the car breaks down, which it will do with probability $b(y)$ (the breakdown probability), it will cost you an additional $K$ dollars to repair it, after which you immediately sell the car and purchase a new one. At the same time, you express your enjoyment with owning a new car as a negative cost $-r(y)$ where $r(y)$ is a declining function with age. At the beginning of each year, you may choose to purchase a new car ($z = 1$) or to hold onto your old one ($z = 0$). You anticipate that you will actively drive a car for another $T$ years.

(a) Identify all the elements of a Markov decision process for this problem.

(b) Write out the objective function which will allow you to find an optimal decision rule.

(c) Write out the one-step transition matrix.

(d) Write out the optimality equations that will allow you to solve the problem.

**3.10** You are trying to find the best parking space to use that minimizes the time needed to get to your restaurant. There are 50 parking spaces, and you see spaces $1, 2, \ldots, 50$ in order. As you approach each parking space, you see whether it is full or empty. We assume, somewhat heroically, that the probability that each space is occupied follows an independent Bernoulli process, which is to say that each space will be occupied with probability $p$, but will be free with probability $1 - p$, and that each outcome is independent of the other.

It takes 2 seconds to drive past each parking space and it takes 8 seconds to walk past. That is, if we park in space n, it will require $8(50 - n)$ seconds to walk to the restaurant. Furthermore, it would have taken you $2n$ seconds to get to this space. If you get to the last space without finding an opening, then you will have to drive into a special lot down the block, adding 30 seconds to your trip.

We want to find an optimal strategy for accepting or rejecting a parking space.

(a) Give the sets of state and action spaces and the set of decision epochs.

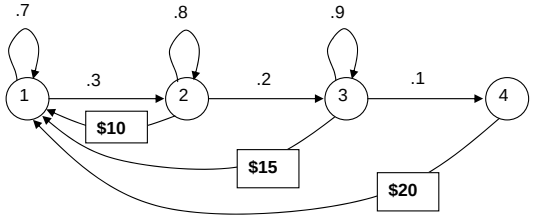
(b) Give the expected reward function for each time period and the expected terminal reward function.

(c) Give a formal statement of the objective function.

(d) Give the optimality equations for solving this problem.

(e) You have just looked at space 45, which was empty. There are five more spaces remaining (46 through 50). What should you do? Using $p = 0.6$, find the optimal policy by solving your optimality equations for parking spaces 46 through 50.

f) Give the optimal value of the objective function in part (e) corresponding to your optimal solution.




**3.11** We have a four-state process (shown in the figure). In state 1, we will remain in the state with probability 0.7 and will make a transition to state 2 with probability 0.3. In states 2 and 3, we may choose between two policies: Remain in the state waiting for an upward transition or make the decision to return to state 1 and receive the indicated reward. In state 4, we return to state 1 immediately and receive $20. We wish to find an optimal long run policy using a discount factor $\gamma = .8$. Set up and solve the optimality equations for this problem.

**3.12** Assume that you have been applying value iteration to a four-state Markov decision process, and that you have obtained the values over iterations 8 through 12 shown in the table below (assume a discount factor of 0.90). Assume you stop after iteration 12. Give the tightest possible (valid) bounds on the optimal value of being in each state.

**3.13** In the proof of theorem 3.10.3 we showed that if $v \geq \mathcal{M}v$, then $v \geq v^*$. Go through the steps of proving the converse, that if $v \leq \mathcal{M}v$, then $v \leq v^*$.

**3.14** Theorem 3.10.3 states that if $v \leq \mathcal{M}v$, then $v \leq v^*$. Show that if $v^n \leq v^{n+1} = \mathcal{M}v^n$, then $v^{m+1} \geq v^m$ for all $m \geq n$.

**3.15** Consider a finite-horizon MDP with the following properties:

| State | Iteration 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|
| 1 | 7.42 | 8.85 | 9.84 | 10.54 | 11.03 |
| 2 | 4.56 | 6.32 | 7.55 | 8.41 | 9.01 |
| 3 | 11.83 | 13.46 | 14.59 | 15.39 | 15.95 |
| 4 | 8.13 | 9.73 | 10.85 | 11.63 | 12.18 |

- $\mathcal{S} \in \Re^n$, the action space $\mathcal{A}$ is a compact subset of $\Re^n$, $\mathcal{A}(s) = \mathcal{A}$ for all $s \in \mathcal{S}$.
- $C_t(S_t, a_t) = c_t S_t + g_t(a_t)$, where $g_t(\cdot)$ is a known scalar function, and $C_T(S_T) = c_T S_T$.
- If action $a_t$ is chosen when the state is $S_t$ at time $t$, the next state is

$$S_{t+1} = A_t S_t + f_t(a_t) + \omega_{t+1},$$

where $f_t(\cdot)$ is scalar function, and $A_t$ and $\omega_t$ are respectively $n \times n$ and $n \times 1$-dimensional random variables whose distributions are independent of the history of the process prior to $t$.

(a) Show that the optimal value function is linear in the state variable.

(b) Show that there exists an optimal policy $\pi^* = (a_1^*, \ldots, a_{T-1}^*)$ composed of constant decision functions. That is, $A_t^{\pi^*}(s) = A_t^*$ for all $s \in \mathcal{S}$ for some constant $A_t^*$.

**3.16** Assume that you have invested $R_0$ dollars in the stock market which evolves according to the equation

$$R_t = \gamma R_{t-1} + \varepsilon_t$$

where $\varepsilon_t$ is a discrete, positive random variable that is independent and identically distributed and where $0 < \gamma < 1$. If you sell the stock at the end of period $t$, it will earn a riskless return $r$ until time $T$, which means it will evolve according to

$$R_t = (1 + r)R_{t-1}.$$

You have to sell the stock, all on the same day, some time before $T$.

(a) Write a dynamic programming recursion to solve the problem.

(b) Show that there exists a point in time $\tau$ such that it is optimal to sell for $t \geq \tau$, and optimal to hold for $t < \tau$.

(c) How does your answer to (b) change if you are allowed to sell only a portion of the assets in a given period? That is, if you have $R_t$ dollars in your account, you are allowed to sell $a_t \leq R_t$ at time $t$.